VHDL implementacija FFTa za obradu audio signala

Mautner, Marijan

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

Sveučilišni studij

VHDL IMPLEMENTACIJA FFT-A ZA OBRADU AUDIO SIGNALA

Diplomski rad

Marijan Mautner

Osijek, 2018.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 13.05.2018.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Marijan Mautner
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 832 R, 23.09.2017.
OIB studenta:	26882945416
Mentor:	Doc.dr.sc. Tomislav Matić
Sumentor:	Leon Šneler
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Ivan Aleksi
Član Povjerenstva:	Dr.sc. Ivan Vidović
Naslov diplomskog rada:	VHDL implementacija FFTa za obradu audio signala
Znanstvena grana rada:	**Arhitektura računalnih sustava (zn. polje računarstvo)**
Zadatak diplomskog rada:	U diplomskom radu potrebno je implementirati VHDL modul za FFT obradu audio signala. Ispitati ograničenja implementacije na AXI sabirnici. Sumentor: Leon Šneler
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene mentora:	13.05.2018.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 01.06.2018.

Ime i prezime studenta:	Marijan Mautner
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 832 R, 23.09.2017.
Ephorus podudaranje [%]:	4%

Ovom izjavom izjavljujem da je rad pod nazivom**: VHDL implementacija FFTa za obradu audio signala**

izrađen pod vodstvom mentora Doc.dr.sc. Tomislav Matić

i sumentora Leon Šneler

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD

Fourierova transformacija često se koristi u analizi signala. Zbog velikih računalnih zahtjeva transformaciju je teško izvoditi u realnom vremenu. U radu je demonstrirana implementacija brze Fourierove transformacije u FPGA tehnologiji kako bi se algoritam sklopovski ubrzao. Objašnjene su osnovne korištene metode iz područja digitalne obrade signala. Prikazane su prilagodbe algoritma kako bi se implementirao u FPGA tehnologiji i objašnjene sve kreirane komponente i izvršene operacije. Predstavljene su mane i prednosti implementiranog dizajna. Objašnjeno je korišteno sklopovlje. Izvedena su mjerenja točnosti i brzine rada implementiranog sustava te su uspoređena s točnošću i brzinom obrade osobnog računala koristeći MATLAB programski paket.

U drugom poglavlju dano je teoretsko objašnjenje Fourierove transformacije i napravljen je izvod brze Fourierove transformacije. U trećem poglavlju objašnjene su korištene tehnologije i alati. Implementacija dizajna objašnjena je u četvrtom poglavlju. Testiranje i rezultati implementiranog sustava obrađuju se u petom poglavlju.

1.1. Zadatak diplomskog rada

U diplomskom radu potrebno je implementirati VHDL modul za FFT obradu audio signala. Ispitati ograničenja implementacije na AXI sabirnici.

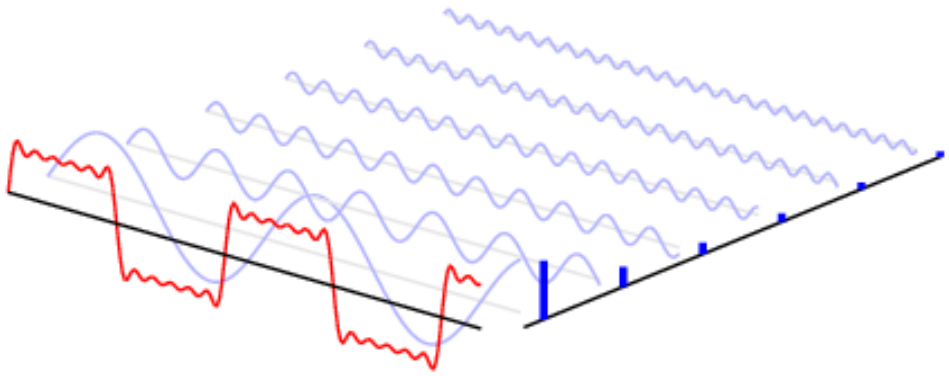
2. FOURIEROVA TRANSFORMACIJA

2.1. Definicija Fourierove transformacije

Transformacija nosi ime po francuskom matematičaru Jeanu Baptisteu Josephu Fourieru kojemu pripadaju najveće zasluge u ovoj disciplini. Potpuno definirana verzija transformacije nastaje početkom 19. stoljeća iako je prihvaćena tek nekoliko desetljeća kasnije. Temeljna ideja transformacije je reprezentacija bilo kojeg signala pomoću zbroja sinusnih i kosinusnih signala. Transformaciju nije moguće točno napraviti u nederiviabilnim točkama kontinuiranih signala, kao što su uglovi trokutastog ili kvadratnog signala. U praktičnom se smislu ove netočnosti zanemaruju jer je točnost dovoljna za sve praktične upotrebe [1].

Ovisno o vrsti signala nad kojim se vrši transformacija postoji više izvedenica, odnosno posebnih oblika Fourierove transformacije. Osnovni oblici su:

- **Fourierova transformacija** – za aperiodične kontinuirane signale
- **Fourierova serija** – za periodične kontinuirane signale
- **diskretna vremenska Fourierova transformacija (DTFT)** – za aperiodične diskretne signale
- **diskretna Fourierova transformacija (DFT)** –za periodične diskretne signale.



Sl. 2.1: Rastav signala na više sinusnih i kosinusnih signala.

Na slici 2.1 prikazano je rastavljanje signala na više sinusnih signala. Rastavljanjem signala određuju se amplitude pojedinih sinusnih signala. Amplitudne vrijednosti prikazane su na slici 2.1 kao stupci s desne strane. Na ovaj način izvedena je frekvencijska analiza signala. Fourierova transformacija često se primjenjuje u tehničkim znanostima pri rješavanju nekih vrsta parcijalnih

diferencijalnih jednadžbi, u teoriji signala (radiofrekvencijska spektralna analiza), obradi slike itd. [1].

Fourierova transformacija temelji se na Fourierovom integralu:

$$f(x)=\frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ixz}dz \int_{-\infty}^{\infty} f(u)e^{-izu}du \tag{2-1}$$

Za postojanje integrala (2-1) dovoljna su dva uvjeta:

1. funkcija f je apsolutno integrabilna na „R“
2. funkcija f je apsolutno integrabilna u svakom konačnom intervalu.

Integral pod (2-1) može se rastaviti na dva dijela [1]:

$$F(z)=\int_{-\infty}^{\infty} f(u)e^{-izu}du \tag{2-2}$$

$$f(x)=\frac{1}{2\pi}\int_{-\infty}^{\infty} F(z)e^{izu}dz \tag{2-3}$$

Funkcija F u (2-2) predstavlja Fourierovu transformaciju funkcije f, a integral pod (2-3) predstavlja inverznu Fourierovu transformaciju. U tehničkim znanostima integrali (2-2) i (2-3) označavaju se na sljedeći način [1]:

$$F(\omega)=\int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt \tag{2-4}$$

$$f(t)=\frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{j\omega t}d\omega \tag{2-5}$$

2.2. Diskretna Fourierova transformacija (DFT)

Diskretna Fourierova transformacija izvedenica je Fourierove transformacije za diskretne, konačne signale.

Fourierovu transformaciju, koristeći izraz (2-2), za neke funkcije nije moguće odrediti, također neke su funkcije određene diskretnim vrijednostima, npr. mjerenje određene fizikalne veličine u pravilnim vremenskim intervalima. Za ove je funkcije Fourierova transformacija određena samo u danim diskretnim intervalima i predstavlja aproksimaciju izraza (2-2) [2].

DFT predstavlja aproksimaciju funkcije $\mathcal{F}[f]$ i najčešće se primjenjuje kada je funkciju nemoguće riješiti koristeći izraz (2-2). Funkcija se aproksimira nad intervalom *a, b* tako da se uzorkuje konačni broj puta (*N*) u jednakim razmacima (ako je nezavisna varijabla vrijeme *t*, funkcija se

uzorkuje konačni broj puta u jednakim vremenskim razmacima Δt) da bi se dobio niz vrijednosti [2].

$$t_0 = a < t_1 < \cdots < t_{N-1} = b \qquad t_k = a + k\Delta t, \Delta t = \frac{b-a}{N-1}, k = 0,1,2, \dots, N-1 \tag{2-6}$$

Aproksimacija funkcije $\mathcal{F}[f(t)]$ dana je izrazom (2-7), označava se $\mathcal{F}_D[f(t_k)] = \mathcal{F}_D(n)$ i predstavlja diskretnu Fourierovu transformaciju.

$$\mathcal{F}_D[f(n)] = \sum_{k=0}^{N-1} f(t_k) e^{-\frac{i2\pi nk}{N}}, n \in \mathbb{Z} \tag{2-7}$$

Ako pretpostavimo da je f definiran nad konačnim skupom $f=\{f(0), f(1),\ldots,f(N-1)\}$ izraz (2-7) može se pisati:

$$\mathcal{F}_D[f(n)] = \sum_{k=0}^{N-1} f(k) e^{-\frac{i2\pi nk}{N}}, n \in \mathbb{Z}_N \tag{2-8}$$

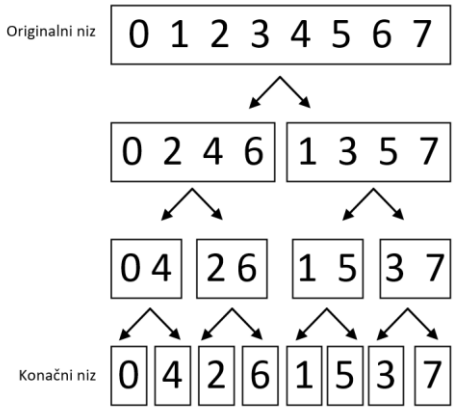
2.3. Brza Fourierova transformacija (FFT)

Brza Fourierova transformacija (engl. *Fast Fourier transform*) jedan je od algoritama za izračunavanje diskretne Fourierove transformacije. Ovaj algoritam jedan je od najbržih poznatih algoritama za izračunavanje DFT-a koristeći računalo i upravo je to razlog njegova korištenja u ovom radu. Primjenom transformacije nad signalom otkrivaju se harmonici od kojih se signal sastoji [1]. Efikasno računanje DFT-a postiže se dekompozicijom računanja u manje DFT izračune. Algoritam u kojemu se dekompozicija temelji na dekompoziciji niza u niz manjih nizova naziva se decimacija u vremenu (engl. *Decimation-in-time algorithms-DIT*) [1], [2].

Primjenom FFT-a na niz od *n* brojeva smanjuje se kompleksnost DFT izračuna. Broj množenja (ako je broj elemenata $N=2m$) smanjuje se sa N^2 na $Nm/2$, a broj zbrajanja sa $N(N-1)$ na $N*m$ zbrajanja [2].

2.3.1 Decimacija u vremenu (DIT)

Signal u vremenskoj domeni koji sadrži *N* uzoraka rastavlja se na *N* signala koji sadrže jedan uzorak [1]. Postupak se temelji na iskorištavanju simetričnosti i periodičnosti kompleksnog eksponenta faktora $W_N^{kn} = e^{-j\left(\frac{2\pi}{N}\right)kn}$. Niz se rastavlja u više koraka tako što se grupiraju parni i neparni uzorci. Veličina nizova korištenih u algoritmu potencije su broja 2. Postupak rastavljanja signala od 8 uzoraka prikazan je na slici 2.2.



Sl. 2.2: Rastavljanje ulaznog signala u DIT postupku.

Zapisom indeksa originalnog niza i konačnog niza u binarnom sustavu vidljivo je kako su indeksi konačnog niza rezultat čitanja indeksa originalnog niza s lijeva na desno. Ovaj postupak naziva se preokretanje bitova (engl. *Bit reversal*). Preokretanje bitova koristi se kao metoda reorganiziranja ulaznog niza prije početka izračuna. Primjer za četiri indeksa prikazan je u tablici 2.1 [1].

Tablica 2.1: Primjer preokretanja bitova.

Indeks originalnog niza baza 10	Indeks originalnog niza baza 2		Indeks konačnog niza baza 2	Indeks konačnog niza baza 10
1	001	→	100	4
2	010	→	010	2
3	011	→	110	6
4	100	→	001	1

Pojednostavljenjem izraza (2-4) dobiva se DFT izraz prikazan jednadžbom (2-9).

$$S(k) = \sum_{n=0}^{N-1} s(n) W_N^{kn} \cdot, k = 0, \dots, N-1 \tag{2-9}$$

Ulazni niz dijeli se na dva niza jednake duljine, parne ($s_0(m)$) i neparne uzorke ($s_1(m)$). Izrazi za navedene nizove dani su jednadžbom (2-10) [2].

$$m = 0 \dots \frac{N}{2} - 1 \;\; \therefore \; S_0(m) = s(2m)\,, S_1(m) = s(2m+1) \tag{2-10}$$

Uvrštavanjem izraza (2-10) u DFT izraz prikazan jednadžbom (2-9), dobiva se izraz za izračun DFT-a podijeljen na sumu parnih i neparnih članova:

$$S(k) = \sum_{m=0}^{\frac{N}{2}-1} s(2m) W_N^{2mk} + W_N^k \sum_{m=0}^{\frac{N}{2}-1} s(2m+1) W_N^{2mk} \quad k = 0 \dots N-1 \tag{2-11}$$

Izvedbom izraza (2-11) za indekse $k = 0, \ldots, N/2 - 1$ te prilagodbom W faktora ($W_N^k = W_{\frac{N}{2}}^{mk}$) dobiva se izraz za prvu polovica S(k) niza [2]:

$$S(k) = \sum_{m=0}^{\frac{N}{2}-1} s(2m) W_{\frac{N}{2}}^{mk} + W_N^k \sum_{m=0}^{\frac{N}{2}-1} s(2m+1) W_{\frac{N}{2}}^{mk} \tag{2-12}$$
$$= S_0(K) + W_N^k S_1(k), k = 0 \ldots \frac{N}{2} - 1$$

Izvedbom izraza (2-11) za indekse $S\left(k + \frac{N}{2}\right)$ te prilagodbom W faktora (prikazano izrazom (2-13)) dobiva se izraz za drugu polovicu S(k) niza [2].

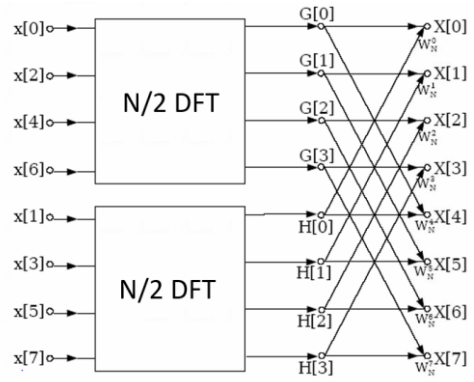
$$W_N^{2m\left(k+\frac{N}{2}\right)} = W_{\frac{N}{2}}^{mK} \quad , \quad W_N^{(2m+1)\left(k+\frac{N}{2}\right)} = -W_N^k W_{\frac{N}{2}}^{mK} \tag{2-13}$$

$$S\left(k + \frac{N}{2}\right) = \sum_{m=0}^{\frac{N}{2}-1} s(2m) W_{\frac{N}{2}}^{mk} - W_N^k \sum_{m=0}^{\frac{N}{2}-1} s(2m+1) W_{\frac{N}{2}}^{mk} \tag{2-14}$$
$$= S_0(K) - W_N^k S_1(k), k = 0 \ldots \frac{N}{2} - 1$$

Izračun vrijednosti W faktora definiranih izrazom $W_N^{kn} = e^{-j\left(\frac{2\pi}{N}\right)kn}$ izvršava se trigonometrijskim prikazom izraza. Na ovaj način dobivaju se realne i imaginarne vrijednosti W faktora. Način izračuna prikazan je jednadžbom (2-15).

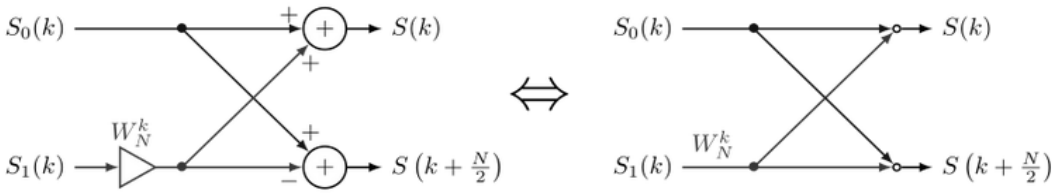
$$w_{N\,real}^k = \cos\left(\frac{2\pi k}{N}\right), \qquad w_{N\,img}^k = \sin\left(\frac{2\pi k}{N}\right) \tag{2-15}$$

Jednadžbama (2-12) i (2-14) izračun se svodi na dva DFT-a veličine *N*/2. Izračun se naziva RADIX-2. Nazvan po podjeli skupa na dva dijela, parne i neparne uzorke [3], [2]. Grafički prikaz izračuna s dva *N*/2 DFT-a za 8 uzoraka prikazan je slikom 2.3.



Sl. 2.3: Prikaz izračuna DFT-a podjelom na parne i neparne uzorke.

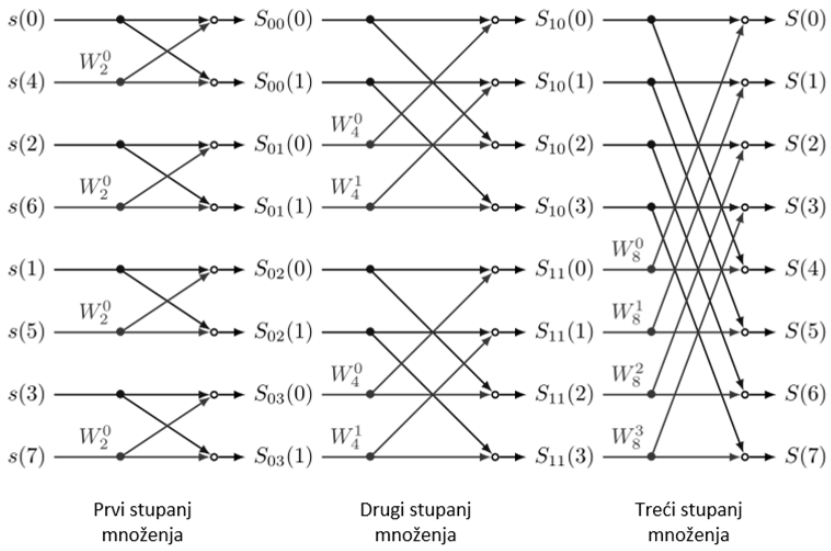
Daljnjim rastavljanjem ulaznog niza metodom RADIX-2 dobivaju se DFT izračuni za dva uzorka. Jedan DFT izračun za dva uzorka prikazan je slikom 2.4. Zbog oblika grafičkog prikaza naziva se leptir (engl. *Butterfly*) [3].



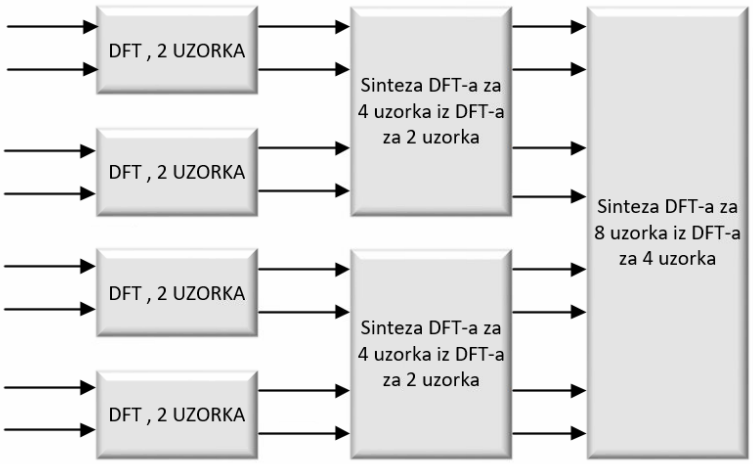
Sl. 2.4: Grafički prikaz izračuna DFT-a za dva uzorka (leptir).

Kako bi se svi izračuni mogli izvesti DFT-om za dva uzorka potrebno je više stupnjeva obrade.

Promatramo li slučaj za osam uzoraka u prvom stupnju obrade računaju se DFT-ovi za dva uzorka. U drugom stupnju obrade računaju se DFT-ovi za četiri uzorka kombinacijom DFT-ova za dva uzorka i iz dobivenih prijašnjih rezultata. U trećem stupanju množenja kombiniranjem DFT-ova za dva uzorka s prijašnjim rezultatima izračunava se DFT za osam uzoraka. Broj stupnjeva množenja ovisi o veličini ulaznog niza. Ako je veličina ulaznog niza 2^N, izračun ima *N* stupnjeva množenja, svaki stupanj sadrži *N/2* leptir-izračuna [3]. Grafički prikaz izračuna pomoću leptir-dijagrama za 8 uzoraka prikazan je na slici 2.5. Vidljiva su 3 stupnja množenja i različite kombinacije leptir-dijagrama u ovisnosti o stupnju množenja. Funkcionalni blok prikaz izračuna za 8 uzoraka prikazan je slikom 2.6.



Sl. 2.5: Grafički prikaz izračuna FFT-a za 8 uzoraka.



Sl. 2.6: Blok prikaz DIT izračuna za 8 uzoraka.

3. PRIMIJENJENE TEHNOLOGIJE

3.1. FPGA

FFT algoritam implementiran je u FPGA tehnologiji (engl. *field-programmable gate array*). FPGA spada u skupinu programabilnih logičkih sklopova koja se zasniva na međusobno povezanim integriranim krugovima (engl. *Integrated circuit − IC*) koji sadrže programabilne blokove. Programiranjem programabilnog dijela IC-ova, postiže se njihovo međusobno spajanje na fizičkoj razini u ovisnosti o željenoj funkciji. Programabilni logični sklop − CLB (engl. *configurable logic block*) osnovni je element FPGA tehnologije [4], [5].

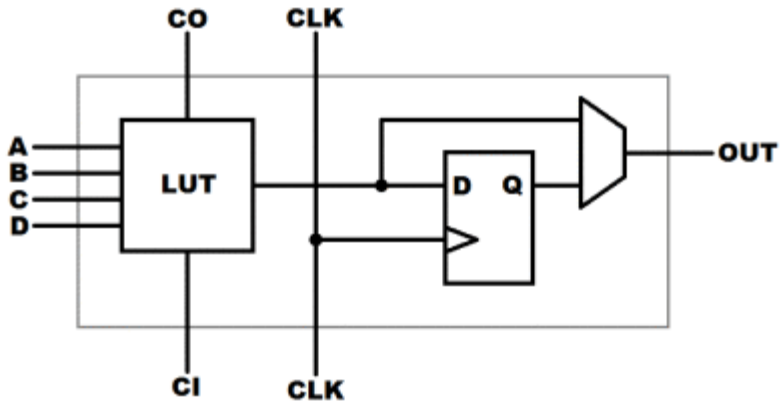
Tehnologije sa sličnim funkcionalnostima uključuju programabilne logičke uređaje −PLD (engl. *programmable logic device*), kao CPLD tehnologija.

Kompleksni programabilni logički uređaji CPLD (engl. *Complex programmable logic device*) tehnologija je programabilnih logičkih uređaja. Unutrašnja arhitektura određena je proizvođačem te je moguće ograničeno funkcionalno konfiguriranje. Ne sadrži mnogo specijaliziranih IC-ova i memorije. Upotreba je ograničena manjom fleksibilnošću i nemogućnošću implementiranja kompleksnih funkcija [5].

ASIC tehnologija sadržava mnogo logičnih vrata te je moguće implementirati funkcije visoke kompleksnosti. Svaki ASIC IC proizveden je za specifičnu funkciju i nije ga moguće reprogramirati. Implementacija kompleksnih funkcija u ovoj tehnologiji nudi najbolje performanse, ali je njihova upotreba dodatno ograničena visokim troškovima dizajna i proizvodnje [5].

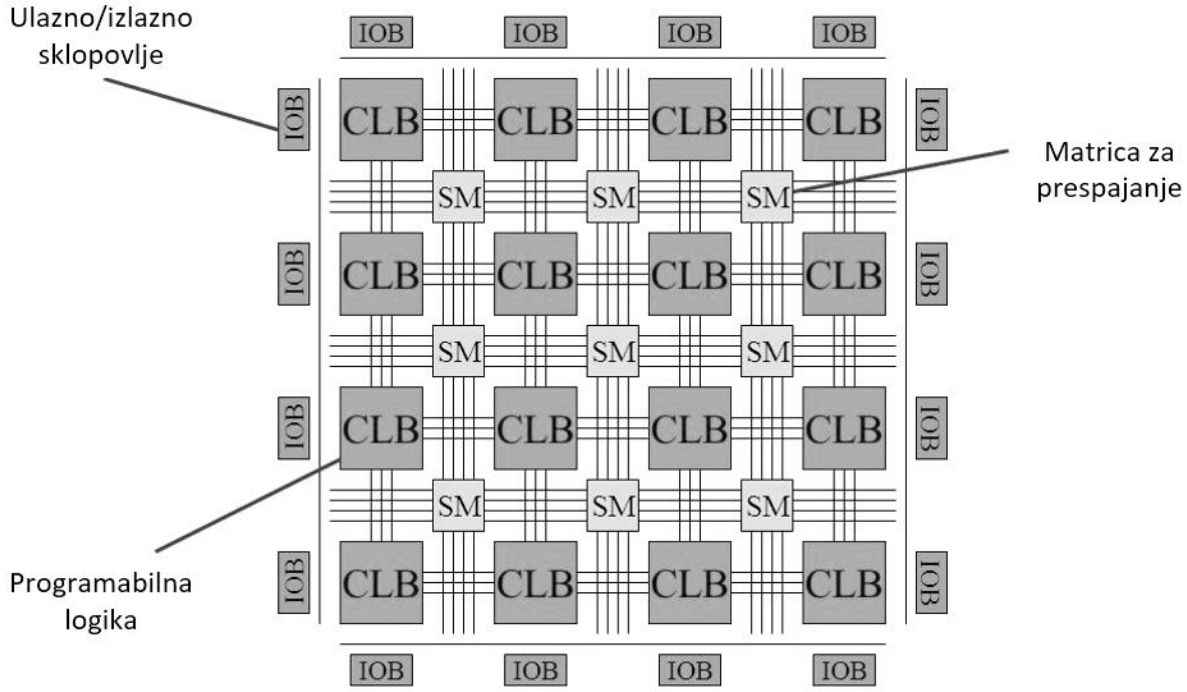
FPGA tehnologija nudi programabilnost, poput PLD-ova, te također mogućnost implementacije funkcija visoke kompleksnosti zbog velike količine različitih integriranih krugova poput ASIC-a.

CLB ćelija kao osnovni element FPGA-a uobičajeno se sastoji od ulazne četiribitne LUT jedinice −pregledne tablice (engl. *Lookup table*), D bistabila i multipleksora na izlazu. Blokovski prikaz pojednostavljene CLB ćelije prikazan je na slici 3.1 [5].



Sl. 3.1: Pojednostavljeni blokovski prikaz CLB ćelije.

Osnovna arhitektura FPGA sklopovlja prikazana je slikom 3.2. Prikazane su CLB ćelije povezane programabilnim međuvezama (matrice za prespajanja) i ulazno-izlazno sklopovlje. Sklop također sadržava dodatne sabirnice koje ne prolaze serijski kroz CLB-ove, nego direktno povezuju različite elemente sustava kako bi se postiglo što manje kašnjenje signala [6].



Sl. 3.2: Pojednostavljeni blokovski prikaz FPGA arhitekture.

FPGA tehnologija osamdesetih se godina prošloga stoljeća, kad se pojavljuje, koristila vrlo ograničeno. Najčešće se koristi samo kao međuveza između različitih sklopovlja. Povećanjem kompleksnosti i pristupačnosti FPGA IC-ova, širi se i njihovo područje uporabe [6].

Današnja uporaba FPGA tehnologije moguća je u različitim područjima. Koristi se u izradi prototipova za implementaciju u ASIC sustavima. Povećanjem kompleksnosti FPGA-ova cjelokupni razvoj ASIC sustava moguće je obaviti na FPGA sustavima [5], [6].

FPGA tehnologija sve više se upotrebljava u digitalnoj obradi signala. Današnji FPGA-ovi sadrže ugrađena množila, posvećeno aritmetičko sklopovlje i sabirnice te veće količine ugrađenog RAM-a od prijašnjih. Zbog velike mogućnosti paralelizacije u brojnim aplikacijama postižu se veće brzine obrade nego na tradicionalnim DSP-ovima (engl. *Digital signal processor*).

Starije generacije FPGA-a uređaja postaju povoljnije i imaju mogućnost implementacije mekih procesora (engl. *Soft processor*). Meki procesori su HDL opisani procesori ili mikroupravljači implementirani u FPGA. Meki procesora imaju dovoljno računalne snage za sve više aplikacija i mogućnost rekonfiguriranja te se koriste umjesto tradicionalnih ugrađenih mikroupravljača [5].

Novije generacije FPGA uređaja imaju ugrađene mikroprocesore (engl. *hard core*) ili mikroupravljače kao odvojeno sklopovlje. nude više računalne snage i ne koriste FPGA resusre [5].

Korištenje FPGA-a za sklopovsko ubrzavanje programskih algoritama sve više se koristi u aplikacijama gdje je bitna brza obrada podataka, odnosno u području rekonfigurabilnog računarstva [5].

3.2. VHDL

VHDL jezik služi za opisivanje digitalnog sklopovlja *(*engl. *Hardware Description Language*). Opisuje ponašanje digitalnih elektroničkih krugova i sustava. Opis elektroničkog sustava moguće je implementirati u sklopovlju. Nastaje kao dio VHSIC (engl. *Very High Speed Integrated Circuit*) projekta vojske Sjedinjenih Američkih Država, osamdesetih godina 20. stoljeća. Glavna područja upotrebe VHDL-a su u FPGA-ovima, CPLD-ovima (engl. *Complex programmable logic device*) i projektiranju ASIC sklopovlja, [5].

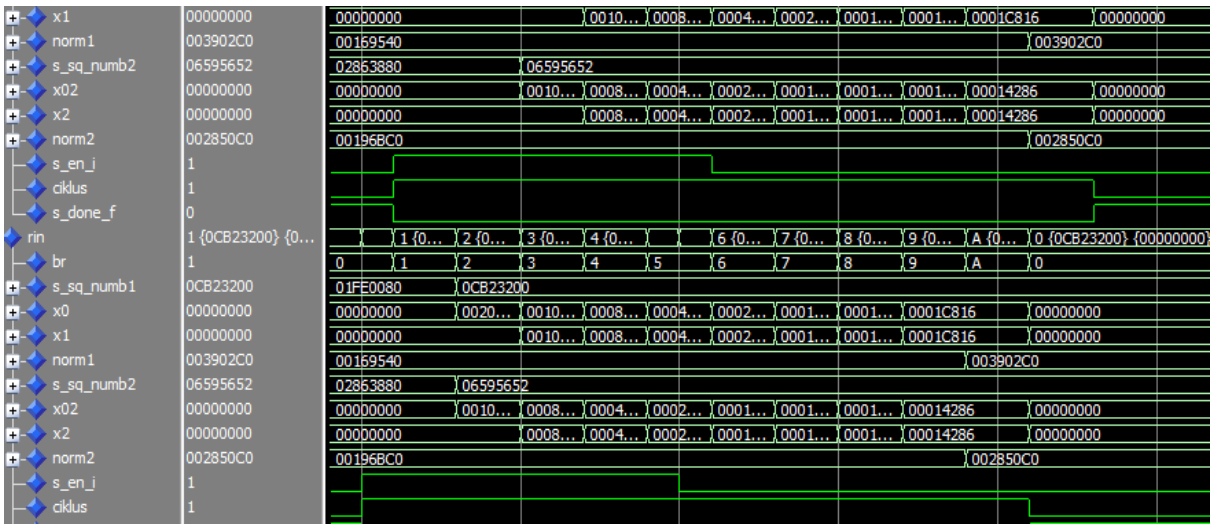
Kao koncept nastaje iz potrebe za dokumentacijom ASIC dizajna. Razvijeni su simulatori koji bi simulirali rad ASIC dizajna. Kasnije se razvijaju sintetizatori VHDL koda koji omogućavaju implementaciju VHDL dizajna u sklopovlje. Većina sintakse za jezik posuđena je od programskog jezika Ada. VHDL standard je neovisan o proizvođačima FPGA sklopovlja te mu upotreba nije ovisna o željenoj implementacijskoj platformi, stoga je prenosiv i ponovno upotrebljiv. Za razliku od programskih jezika, VHDL se izvodi u potpunosti paralelno, jedino gdje je moguće slijedno izvođenje jest u procesima, procedurama i funkcijama [4], [5], [6].

Izvršavanjem sinteze sustava prvo se vrši sintetza VHDL koda. Sintetiziranje je proces u kojemu se zapis u VHDL jeziku konvertira u digitalnu shemu na sklopovskoj razini. Drugi je korak optimizacija dizajna. Zadnji je korak stvaranje stvarne implementacije iz sheme za specifično korišteno sklopovlje [4].

3.3. ModelSim simulacijsko okruženje

ModelSim simulacijsko je okruženje za HDL jezike poput VHDL-a i Verilog. Služi za verifikaciju dizajna kroz izvođenje simulacija. Smatra se programom niže kompleksnosti i nije namijenjen za profesionalni rad. Ima ugrađen C debugger [4].

Simulacija se izvodi grafički, putem korisničkog sučelja ili preko automatiziranih skripti. Simulacijom je moguće promatrati stanja svih signala unutar HDL dizajna, što nije moguće ako je sustav implementiran u sklopovlje. Simulacija je najvažniji alat u verifikaciji dizajna, a u postupku je cilj pobuditi sva moguća stanja sustava. Kako bi se sustav što kvalitetnije testirao, testovi moraju biti prilagođeni dizajnu. Testovi su često pisani u jezicima koji imaju fleksibilniju sintaksu, kako bi pisanje kompleksnih testova bilo što jednostavnije, kao System Verilog. Za jednostavna testiranja koristit se i VHDL. Na slici 3.3 prikazan je izgled sučelja simulacije [4], [5].



Sl. 3.3: Prikaz simulacije u ModelSim-u.

3.4. Vivado Design Suite

Vivado Design Suite programski je paket razvijen od tvrtke Xilinx. Namijenjen je za dizajniranje i sintezu HDL dizajna. U radu je korišten nakon što su svi dijelovi HDL dizajna opširno testirani u ModelSim okruženju [7].

Vivado okruženje ima mogućnost grafičkog prikaza dizajna. Grafički prikaz dizajna temelji se na međusobnom spajanju IP-ova (engl. *intellectual property*). IP-ovi su zapakirane cjeline HDL dizajna. Mogu biti kreirane od strane korisnika ili od strane proizvođača. IP-ovi kreirani od strane proizvođača najčešće ne nude uvid u sam HDL dizajn u njima, a način korištenja opisan je dokumentacijom. Često nude iscrpne dokumentacije o korištenju, mogućnosti konfiguriranja i upotrebi sklopovskih resursa. U radu se koriste IP-ovi kreirani od proizvođača (Xilinx) kao i korisnički definirani IP-ovi. Vivado je novije razvojno okruženje tvrtke Xilinx, razvijeno kao zamjena za Xilinx ISE. Ima mogućnost rada sa Xilinx FPGA-ovima 7. generacije [7].

Mogućnosti koje nudi su:

- simulacija dizajna
- sinteza dizajna
- proračun vremena
- pregled RTL dizajna
- sinteza visoke razine.

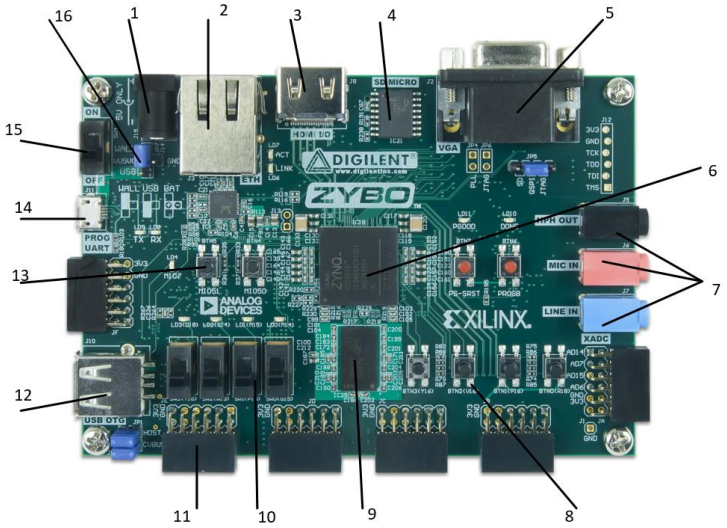
3.5. Zybo razvojni sustav

U radu se koristi razvojna platforma Zybo ZYNQ 7000, razvijena od strane firme Digilent. Razvojna platforma sagrađena je oko ZYNQ SoC-a [8].

Razvojna platforma sadrži:

- ZYNQ XC7Z010-1CLG400C
- 512MB x32 DDR3
- HDMI port
- VGA port
- MicroSD port
- OTG USB 2.0
- audio ulaz/izlaz
- JTAG programiranje
- pretvornik UART na USB
- GPIO: 6 tipkala, 4 prekidača, 5 LE dioda
- šest Pmod portova.

Zybo razvojna platforma programira se preko USB 2.0 sučelja, preko kojega može biti i napajan. U potpunosti je kompatibilna s Vivado razvojnim okruženjem. Razvojni sustav prikazan je slikom 3.4, pojedini dijelovi razvojnog sustava objašnjeni su u tablici 3.1.



Sl. 3.4: Zybo razvojna platforma.

Tab 3.1: Opis pojedinih komponenti Zybo razvojne platforme.

Broj	Opis komponente	Broj	Opis komponente
1	Izvor napajanja	9	DDR 3 memorija
2	Ethernet RJ45 priključak	10	Prekidači
3	HDMI	11	Pmod priključci
4	Čitač SD kartice	12	OTG USB 2.0
5	VGA	13	MIO tipkala
6	ZYNQ 7000 SoC	14	UART/JTAG USB priključak
7	Zvučni priključci	15	Prekidač napajanja
8	Tipkala	16	Prespojnik za odabir napajanja

Razvojno okruženje ima mogućnost rada s operacijskim sustavima. Operacijski sustav učitava se sa SD kartice. Postoji više Linux distribucija koje je moguće koristiti [8].

Razvojno okruženje sadrži USB/UART pretvornik. Dostupni su upravljački programi za PC. Sklopovlje i programska podrška omogućuju serijsku komunikaciju s računalom. Komunikacija se izvodi na brzini od 115200 baud-a, s jednim stop bitom, bez pariteta, duljina riječi je 8 bita [8].

Serijska komunikacija s ZINQ SoC-om izvodi se putem dvožičnog serijskog Tx/Rx porta. Razvojno okruženje sadrži dvije statusne LE diode koje signaliziraju protok podataka, jedna za Tx i jedna za Rx sučelje [8].

4. IMPLEMENTACIJA

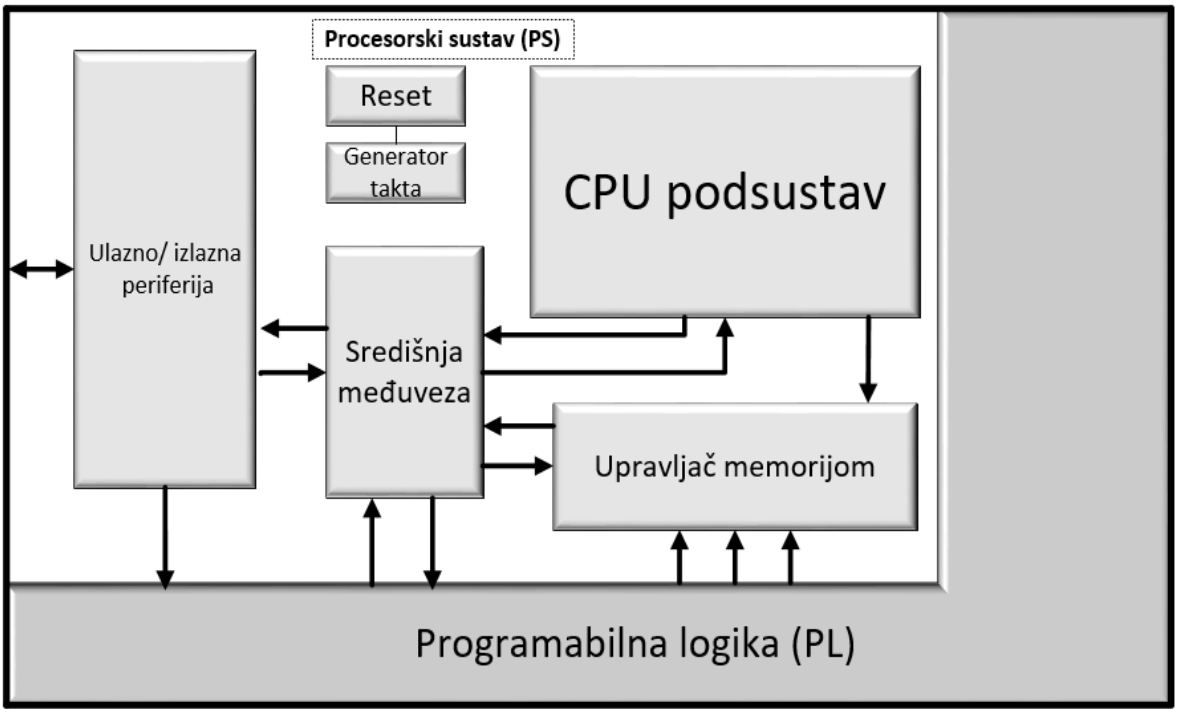
4.1. ZYNQ-7000

ZYNQ-7000 je SoC (*engl. system on chip*), integrirani sustav, razvijen od tvrtke Xilinx. Glavne dijelove čine procesorski sustav (PS) i programabilna logika (PL).

ZYNQ-7000 sastoji se od integriranog CPU podsustava (*hard-core CPU*) koji se također naziva jedinica za obradu aplikacija −APU (engl. *Application Processing Unit*), upravljača memorije, ulazno-izlazne periferije te centralnog sklopa za povezivanje.

Ugradnjom procesora u SoC s FPGA-om postiže se fleksibilnost sustava koja nije bila moguća s prije upotrebljavanom obitelji procesora s mekom jezgrom (engl. *soft-core processor*). Procesori s mekom jezgrom su u potpunosti implementirani u FPGA, odnosno dio su HDL dizajna. Ugrađeni *hard-core* procesori nude bolje performanse i ne koriste FPGA resurse [7].

Pojednostavljeni blokovski prikaz arhitekture ZYNQ SoC-a prikazan je na slici 4.1.



Sl. 4.1: Pojednostavljeni blok-prikaz ZYNQ-7000 SoC-a.

4.2. Procesorski sustav (PS)

Procesorski sustav može sadržavati dvojezgreni ili jednojezgreni procesor baziran na ARM Cortex-A9 procesorskom sustavu.

Osnovne karakteristike ugrađenog procesora:

- 2.5 DMIPS/MHz
- tri watchdoga i jedan globalni tajmer
- *floating point* izračun jednostruke i dvostruke preciznosti
- 32 KB (L1) + 512 KB(L2) Cache
- integrirani boot ROM
- DDR3 i DDR 2 podrška
- ECC podrška
- Podrška za IEEE Std 802.3 i IEEE Std 1588 rev. 2.0
- DMA
- USB 2.0 OTG, CAN 2.0B Bus, SPI, UART, I2C
- 32x4 GPIO portova.

Procesorski sustav programira se pomoću Xilinx SDK okruženja, koristi se programski jezik C.

Adresiranjem registara AXI protokola ostvarena je komunikacija s programabilnom logikom. Procesor se koristi za učitavanje podataka za obradu, čitanje rezultata i mjerenje vremena potrebnog za obradu.

4.2.1 AMBA AXI 4

ZYNQ SoC kao sabirnicu za komunikaciju unutar integriranog sklopa koristi AMBA AXI 4 sabirnicu (engl. *Advanced Microcontroller Bus Architecture (AMBA*), *Advanced Extensible Interface (AXI)*). Sabirnica je osmišljena 1996. godine kao rješenje za povezivanje funkcionalnih dijelova SoC sklopovlja koje sadrži ARM procesore. Godine 2013. izlazi peta iteracija protokola AMBA 5 CHI. Danas su protokoli temeljeni na različitim AMBA iteracijama standard u industriji ugrađenih računalnih sustava [7].

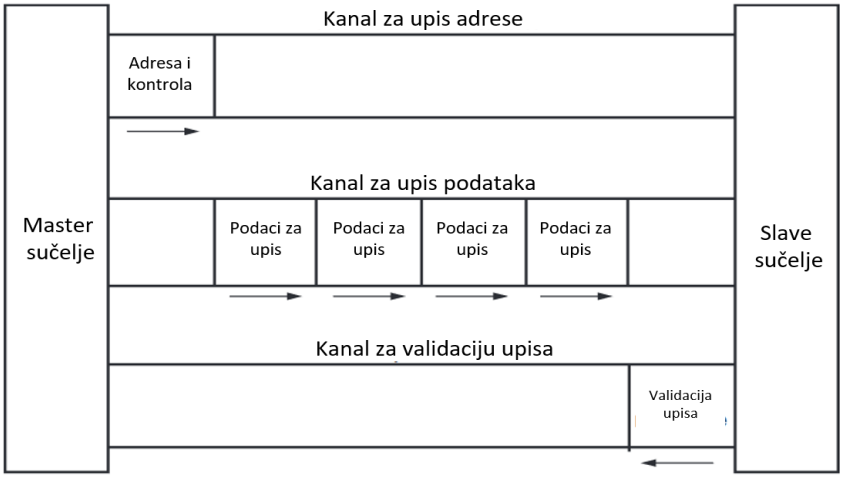
Vrste AMBA AXI 4 protokola:

- AXI4 – koristi se za povezivanje memorijskih mapa, pruža najbolje performanse od svih izvedenica AXI protokola. Podržava prijenos podataka u valovima. Maksimalnog kapaciteta do 256 podatkovnih riječi
- AXI4 lite – jednostavnija je verzija AXI4 protokola, prenosi jedan po jedan podatak, svaki podatak je zasebno adresiran
- AXI-Stream – koristi se za tok podataka, ne zahtijeva adresiranje svakog podatka nego samo početnu adresu i količinu podataka za prijenos.

AXI4 lite sadži 5 komunikacijskih kanala:

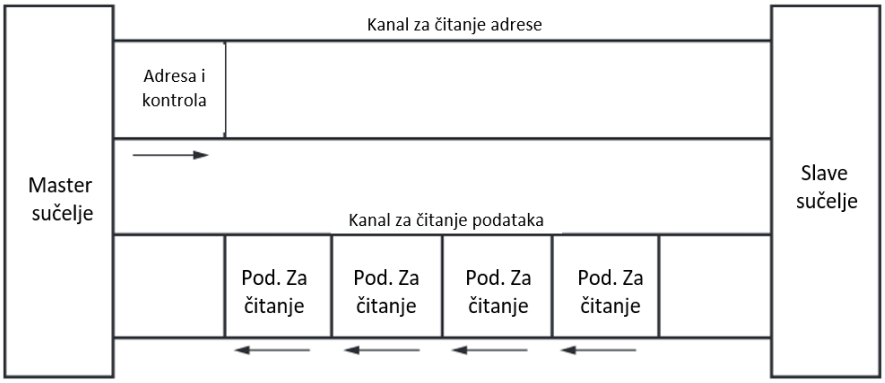
- kanal za čitanje adrese
- kanal za pisanje adrese
- kanal za čitanje podataka
- kanal za upis podataka
- kanal za validaciju upisa.

AXI4 lite protokol u radu se koristi za povezivanje procesorskog dijela s programabilnom logikom.



Sl. 4.2 AXI lite proces upisa podataka.

Na slici 4.3 prikazan je ciklus čitanja podatka. AXI *master* šalje adresu AXI *slave-u* nakon čega mu *slave* šalje podatke na traženoj adresi. Na slici 4.2 prikazan je proces upisa podataka od strane *mastera* [7].



Sl. 4.3: AXI lite proces čitanje podataka.

4.3. Programabilna logika (PL)

Programabilna logika (PL) je FPGA dio ZYNQ SoC-a. Spajanje s procesorskim sustavom izvršava se preko AXI sabirnice. Za razvoj sustava implementiranog u PL-u korišteno je Vivado razvojno okruženje. U radu se koriste IP-ovi kreirani od proizvođača (Xilinx) kao i korisnički definirani IP-ovi.

4.3.1 Brojevni sustav

Postupak izračuna DFT-a korištenim algoritmom uključuje višestruka množenja i zbrajanja. U drugom poglavlju rada objašnjen je DIT algoritam u kojemu se koriste W faktori. Jednadžba (2-15) prikazuje način izračuna W koeficijenata. Vrijednosti W faktora racionalni su brojevi unutar intervala 0, 1. Zbog toga svojstva W faktora u izračunima se ne mogu koristiti samo cjelobrojne vrijednosti. FPGA ima mogućnost rada s brojevima s pomičnim zarezom (engl. *floating point numbers*), ali je upotreba ograničena zbog potrebe za specijaliziranim sklopovljem. U radu se koristi notacija racionalnih brojeva s nepomičnim zarezom (engl. *fixed point*) [9].

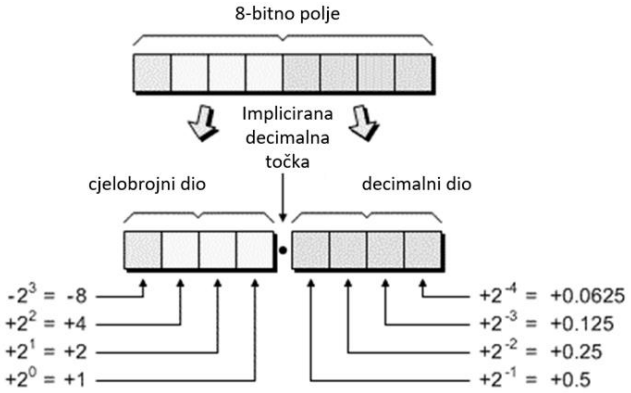
Brojevi s nepomičnim zarezom često nude manju preciznost i fleksibilnost od brojeva s pomičnom zarezom. Dizajner mora voditi računa o potrebnoj preciznosti i maksimalnoj vrijednosti rezultata. Izračuni koji uključuju brojeve s nepomičnim zarezom ne zahtijevaju posebnu vrstu sklopovlja kao s pomičnim zarezom [6]. U sklopovlje se implementiraju kao dva binarna broja spojena posebnom vezom te je njihovo izvođenje brže nego broja s pomičnim zarezom. Na slici 4.4 prikazana je reprezentacija racionalnog broja fiksnim zarezom.

U radu se koriste ulazni pozitivni podaci veličine 8 bita. Podaci se spremaju u vektor veličine 22 bita. Korištena veličina cjelobrojnog dijela vektora je 14 bita, veličina koja je potrebna kako bi se mogla spremiti najveća moguća vrijednost nakon obrade FFT algoritmom. Brojevima koji se

koriste u obradi dodjeljuje se 8 bita decimalne preciznosti. Veličina decimalnog dijela određena je eksperimentalno kako bi prve dvije znamenke decimalnog dijela konačnog rezultata bile točne (u dekadskom sustavu). W faktori imaju drugačiji format od korištenih podataka za obradu. Cjelobrojni dio W faktora veličine je 6 bita dok je decimalni dio 16 bita. Do ovih veličina također se došlo eksperimentalnim putem kako bi točnost konačnog rezultata bila zadovoljavajuća.

Za prikaz svih podataka u sustavu pomoću brojeva s nepomičnim zarezom koristi se biblioteka *fixed_pkg*.

Biblioteka *fixed_pkg* sadrži specijalizirane funkcije za smanjivanje veličine binarnog zapisa. Ova ugrađena funkcionalnost koristi se u radu kako binarni zapis podataka ne bi postajao nepotrebno veći sa svakim množenjem i zbrajanjem. Biblioteka ima ugrađeno zaokruživanje brojeva.



Sl. 4.4: Prikaz racionalnog broja zapisanog fiksnim zarezom.

4.3.2 Princip rada implementiranog sustava

Implementacija FFT algoritma i izračuna apsolutne vrijednosti u radu je u potpunosti izvedena unutar FPGA-a dijela ZYNQ SoC-a. Procesorski sustav koristi se za predavanje podataka PL dijelu i čitanje iz PL dijela. Procesorski sustav slijedno šalje po 64 uzorka signala PL dijelu na obradu, čita rezultate obrade i sprema rezultate. Na ovaj način postignuto je sklopovsko ubrzanje algoritma brze Fourierove transformacije.

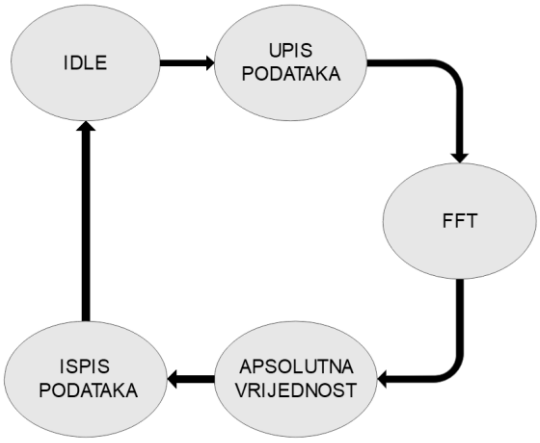
Upis podataka izvršava se preko AXI4 lite sučelja. Za potrebe upisa izrađen je poseban IP koji je implementiran u PL dijelu sustava. Za prijenos se koriste AXI4 lite *slave* registri. IP vrši prilagodbu podataka primljenih preko AXI protokola i upis podataka u BRAM.

FFT algoritam izvršava se nakon upisa podataka. Za potrebe obrade izrađen je poseban IP koji se spaja na BRAM-ove. Algoritam izračunava dvije po dvije vrijednosti te sve međurezultate i konačan rezultat upisuje u iste BRAM-ove.

Apsolutna vrijednost i normiranje uključuje se nakon izračuna FFT-a**.** Rezultati FFT algoritma kompleksni su brojevi te je za prikaz rezultata potrebno izračunati apsolutne vrijednosti kompleksnih rezultata te ih normirati. Za ovu svrhu izrađen je poseban IP koji izračunava jednu po jednu potrebnu vrijednost. IP je spojen na BRAM memoriju gdje i sprema sve rezultate.

Ispis rezultata izvršava se prijenosom rezultata iz BRAM memorije u procesorski sustav. Ispis se obavlja putem AXI4 lite sučelja. Za potrebe ispisa izrađen je poseban IP koji od procesorskog sustava prima adresu podatka te nakon toga šalje podatak procesorskom sustavu.

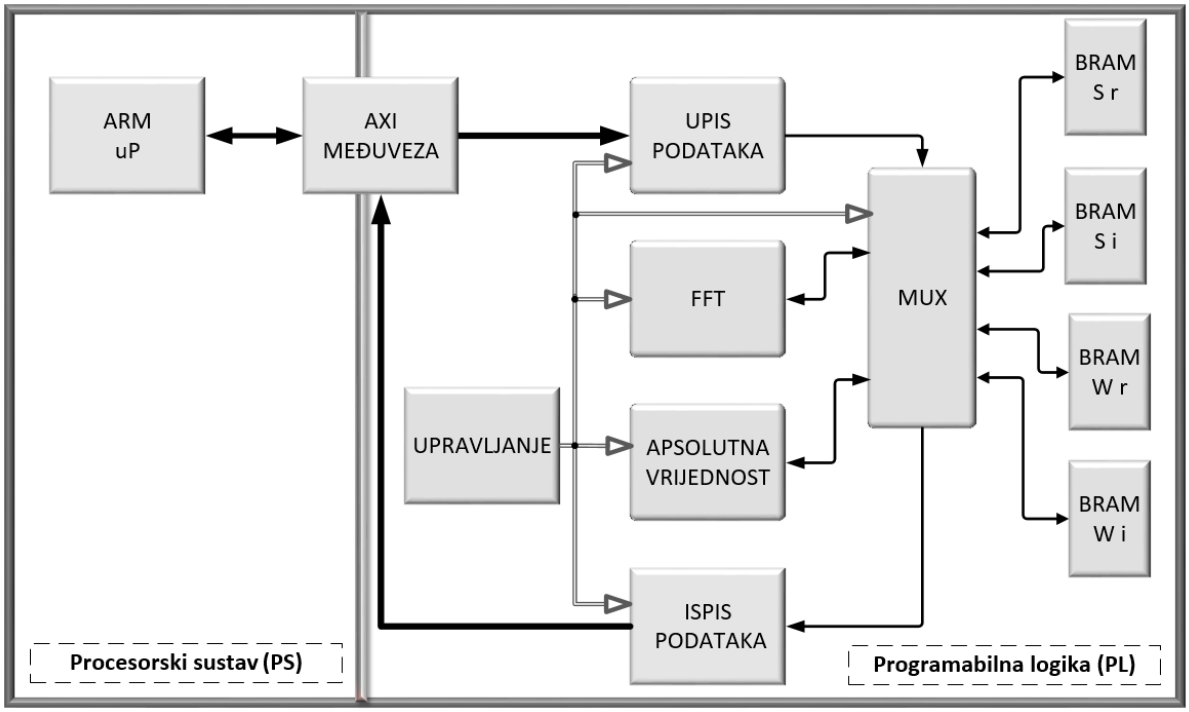
Osnovna stanja u kojima se sustav nalazi tijekom obrade prikazana su na slici 4.5.



Sl. 4.5: Osnovna stanja upravljačkog IP-a.

Stanja prikazana slikom 4.5 stanja su izrađenog upravljačkog IP-a. Svi IP-ovi u sustavu spojeni su na upravljački IP koji upravljačkom sabirnicom uključuje pojedine IP-e, ovisno o stanju u kojemu se nalazi. Po završetku svojih zadataka svi IP-ovi šalju kontrolni signal upravljačkom IP-u.

Pojednostavljena blokovska shema sustava prikazana je slikom 4.6. Svaki blok u PL dijelu slike predstavlja implementirani IP u sustavu.



Sl. 4.6: Pojednostavljeni blok-prikaz implementiranog sustava.

4.3.3 Blok RAM

Osnovne karakteristike blok RAM-a:

- dvokanalni način rada sa širinom porta do 76 bita
- programabilna FIFO logika
- ugrađeno sklopovlje za mogućnost provjere pogreške.

Svaki uređaj iz ZYNQ-7000 obitelji ima 755 dvokanalnih blok RAM-ova, svaki kapaciteta 36 Kb. Operacije čitanja i pisanja obavljaju se sinkrono, sve vrijednosti unesenih adresa kao i svih ostalih portova čuvaju se do iduće promjene. Ovakav pristup osigurava sigurniji prijenos podataka iako operacija čitanja zahtijeva više signala takta kako bi se ispravno izvršila [7].

U radu se koristi četiri Xilinx IP-a koji direktno koriste BRAM resurse sklopovlja.

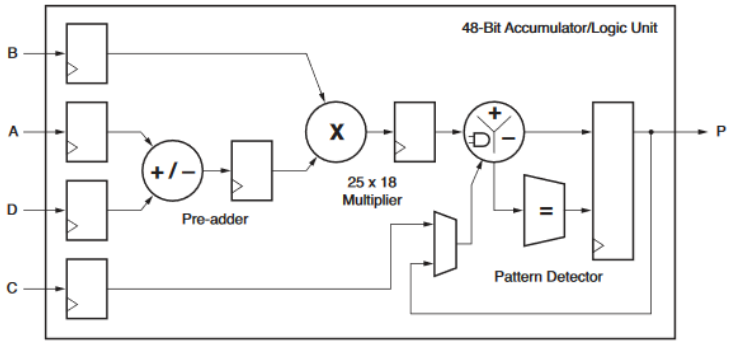
4.3.4 DSP ćelije

Osnovne karakteristike:

- 25 množila dvojnih komplemenata od 18 bita
- 25 x 48 bita akumulatori

- predzbrajala
- aritmetičko logična jedinica.

DSP-ovi se mogu koristiti za zbrajanja i množenja. Ćelije DSP-ova koje su implementirane u sedmu generaciju Xilinx uređaja omogućuju visoku paralelizaciju obrade kao i visok signal takta sklopa. U radu se koriste za mnogobrojne operacije množenja kod FFT algoritma. Pojednostavljen blok-prikaz DSP48E1 ćelije prikazan je na slici 4.7.



Sl. 4.7: Pojednostavljeni blok-prikaz DSP48E1 ćelije.

4.3.5 CORDIC

CORDIC (engl. *COordinate Rotation DIgital Computer*) je algoritam primarno korišten za izračunavanje trigonometrijskih funkcija. Algoritam izvodi rotaciju vektora (X, Y) za određeni kut te stvaranje novog vektora (X', Y'). Izraz za izračun novog vektora rotiranog za kut Θ dan je izrazom [10]:

$$\begin{aligned} X' &= Z_i(\cos(\Theta)X - \sin(\Theta)Y) \\ Y' &= Z_i(\cos(\Theta)Y + \sin(\Theta)X) \\ Z' &= \frac{1}{\prod_{i=1}^{n} \operatorname{acos}(\operatorname{atan}(2^{-i}))} \end{aligned} \tag{4-1}$$

Rotacija vektora izvodi se slijednim rotacijama za sve manji kut rotacije. Jednadžbom (4-2) dan je izraz za i-tu iteraciju gdje je i indeks iteracije koji ide od 0 do n. Smjer rotacije označen je s α_i, te može imati vrijednost 1 ili -1 [10].

$$\begin{aligned} x_{i+1} &= x_i - \alpha_i\, y_i 2^{-i} \\ y_{i+1} &= y_i + \alpha_i\, x_i 2^{-i} \\ \Theta_{i+1} &= \Theta_i + \alpha_i \operatorname{atan}\left(2^{-i}\right) \end{aligned} \tag{4-2}$$

Rotacija vektora izvodi se tako da se α_i vrijednost postavi tako da kut Θ teži ka nuli. Rezultat je operacije skalirani vektor $Z_i * (X', Y')$ [10].

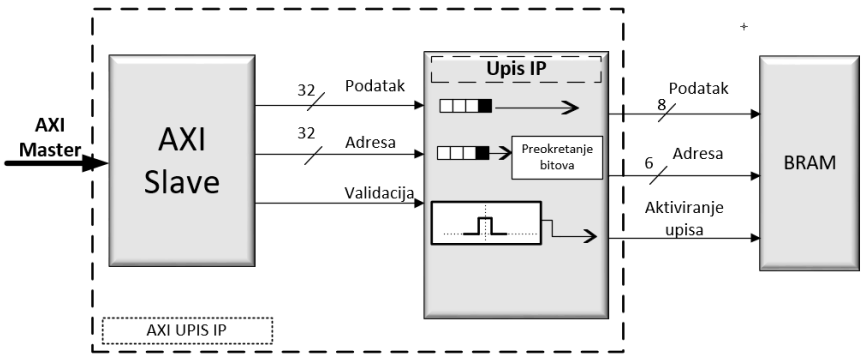
Iz jednadžbe (4-2) vidljivo je kako se za izračun *x* i *y* koordinata koristi jedino operacija zbrajanja i logičkog posmaka. Vrijednost koeficijenta α_i uvijek je pozitivna (+1) jer operacija određivanja korijena broja kao ulaz ima samo pozitivne brojeve. Operacije vezane za broj 2 obavljaju se logičkim posmakom. Vrijednosti *Z* koeficijenata unaprijed su izračunate za različiti broj iteracija. Nakon *n* iteracija dobivaju se vrijednosti x_n parametra koji se množi s odgovarajućim *Z* koeficijentom kako bi se dobila vrijednost korijena broja [10].

4.3.6 Upis podataka

Za potrebe upisa kreiran je poseban IP s AXI komunikacijskim mogućnostima. Podaci preneseni preko AXI sučelja veličine su 32 bita. Koriste se odvojeni *slave* registri za slanje adrese i za slanje podatka. Primljeni podaci obrađuju se kako bi se iz 32 bita dobilo 8 bita koji se koriste za brojčanu vrijednost i 6 bita za adrese. Implementiran je algoritam preokretanja bitova kako bi podaci spremljeni u BRAM bili pravilno raspoređeni za FFT obradu. Upis podataka izvršava se u BRAM-ove namijenjene za spremanje realnih vrijednosti.

Točno primanje podataka osigurano je dodatnim validacijskim signalom. Validacijskom signalu zadaje se logičko stanje nula pri početku upisa procesorskog dijela u AXI registre te se po završetku upisa stavlja u logično stanje jedan. Prijelaz validacijskog signala iz logičnog stanja nula u logično stanje jedan aktivira IP za upis podataka koji tada zaprima podatke, obrađuje ih i sprema u BRAM. Validacijski signal potreban je jer PL dio ZYNQ-a u manje ciklusa takta izvrši upis u BRAM-ove, nego što procesorski sustav može upisivati podatke u AXI registre.

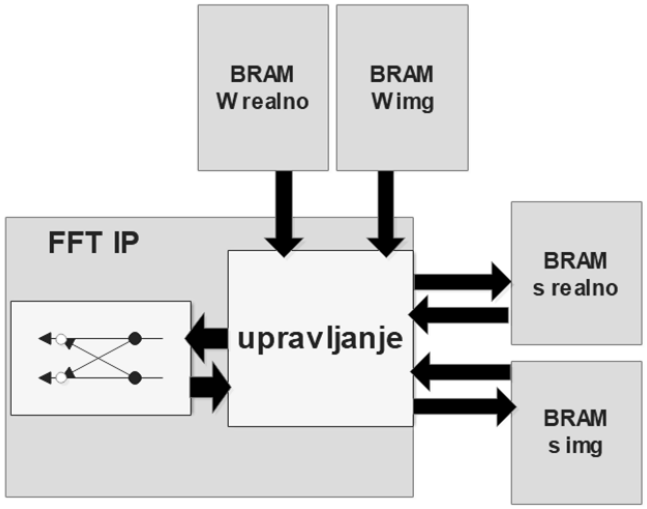
Kako bi IP za upis podataka ostvario AXI komunikaciju zapakiran je u AXI IP. Pojednostavljeni blokovski prikaz upisa podataka putem AXI sučelja prikazana je na slici 4.8.



Sl. 4.8: Pojednostavljeni blokovski prikaz IP-a za upis podataka.

4.3.7 Implementacija algoritma decimacije u vremenu (DIT)

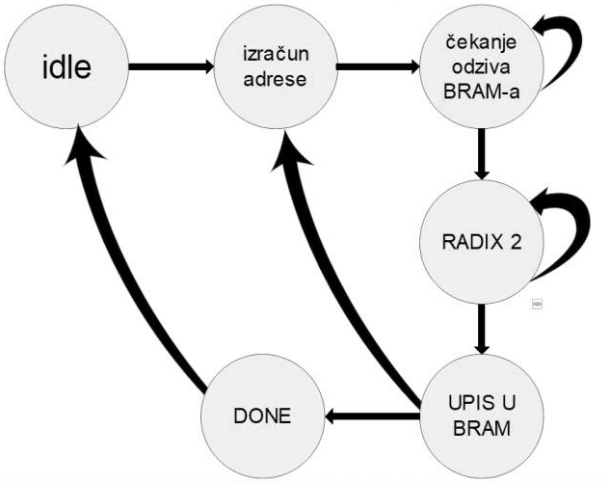
Nakon upisa podataka u BRAM-ove pokreće se DIT algoritam. FFT IP koristi 4 odvojena BRAM IP-a za pohranu vrijednosti. Kao što je prikazano na slici 4.9 korištena su dva BRAM-a za pohranjivanje vrijednosti izračuna (s realno, s img) i dva BRAM-a za spremanje vrijednosti W faktora. Vrijednosti W faktora unaprijed su izračunani koristeći izraz (2-15) i predefinirani u BRAM blokove. BRAM blokovi namijenjeni za rezultate koriste se za upis i ispis podataka, kao što je to naznačeno strelicama na slici 4.9 . Na slici nije prikazan multiplekser koji je sastavni dio sustava.



Sl. 4.9 Pojednostavljeni blok-prikaz povezivanja FFT IP-a s BRAM-ovima.

Kreirani FFT IP sastoji se od upravljačkog dijela i dijela za izračun rezultata. Upravljački dio adresira podatke iz BRAM-ova, prosljeđuje zaprimljene podatke na obradu i sprema rezultate u BRAM-ove. Osnovna stanja automata upravljačkog dijela IP-a prikazana su slikom 4.10.

Automat se nalazi u „idle“ stanju dok se upisuju podaci u PL. Nakon što IP zaprimi aktivacijski signal automat prelazi u stanje „izračun adrese“. U stanju „izračun adrese“ adresiraju se dva podatka iz BRAM-a s realnim podacima i dva podatka iz BRAM-a s imaginarnim podacima. Također se adresiraju odgovarajući realni i imaginarni W faktori. Izračun se izvršava slijedno, računajući jedan rezultat leptir-dijagrama prikazanog slikom 2.4. Nakon izračuna automat sprema rezultate u BRAM-ove i vraća se u stanje „izračun adrese“ te se cijeli ciklus ponavlja. Nakon svih obavljenih izračuna automat odlazi u stanje „done“ te zatim u stanje „idle“ u kojemu ostaje do unosa novih podataka za obradu.

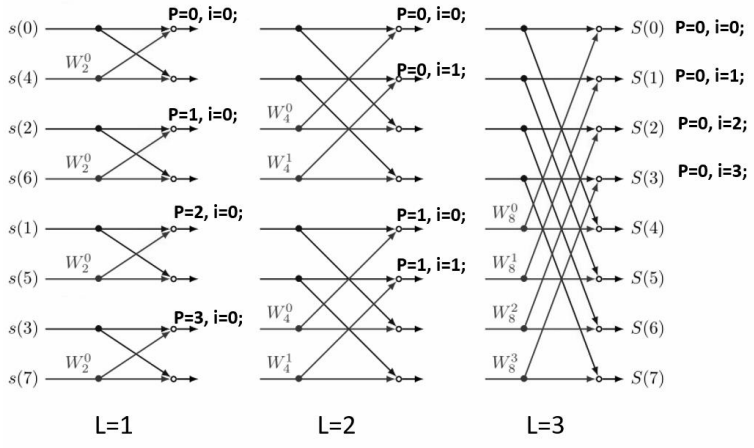


Sl. 4.10: Prikaz stanja upravljačkog automata FFT IP-a.

Izračun adresa radi se u ovisnosti o stupnju množenja i broju uzoraka. Označimo li stupanj množenja s L definirani se koeficijenti:

$$\begin{aligned} \mathrm{P} &= 0, \dots, \frac{\mathrm{N}}{2^{\mathrm{L}}} - 1 \\ \mathrm{i} &= 0, \dots, 2^{\mathrm{L}-1} - 1 \end{aligned} \tag{4-3}$$

Koeficijenti *P* i *i* su brojači. Kao što je objašnjeno u potpoglavlju 2.3.1 za izračun DFT-ova koristi se DFT za dva uzorka, leptir-dijagram. *P* brojač broji DFT-ove u pojedinom stupnju množenja. U ovisnosti o stupnju množenja DFT-ovi izvedeni pomoću leptir-dijagrama izvode DFT izračun za 2^L uzoraka [3]. Brojač *i* broji leptir-dijagrame u određenom DFT-u za 2^L uzoraka. Stanja brojača u ovisnosti o stupnju množenja za ulazni signal od 8 uzoraka prikazana su slikom 4.11.



Sl. 4.11: Prikaz stanja P i i brojača u ovisnosti o stupnjevima množenja L.

Koristeći izraz (4-3) kreirani su izrazi za adresiranje parnih ($adresa_1$) i neparnih ($adresa_2$) elemenata niza, kao i W faktora. Izraz za izračunavanje adresa u ovisnosni o stupnju množenja dan je izrazom (4-4).

$$\begin{aligned} adresa_1 &= i + (P * 2^L) \\ adresa_2 &= i + 2^{L-1} + (P * 2^L) \\ adresa_W &= i\left(\frac{N}{2^L}\right) \end{aligned} \tag{4-4}$$

Dio IP-a za izvršavanje izračuna kreiran je iz jednadžbi (2-12) i (2-14) te predstavlja jedan leptir-dijagram, prikazan slikom 2.4. Kako bi se povećala efikasnost izračuna, način kodiranja prilagođen je korištenju DSP ćelija. Nakon primanja podataka od upravljačkog dijela svi podaci se spremaju lokalno. Kao što je vidljivo iz jednadžbi (2-12) i (2-14) u izračunu se koriste isti podaci, drugačije raspoređeni, za dobivanje različitih rezultata. Kako bi se podaci pripremili za izračun u DSP ćelijama, svi faktori u izračunu moraju biti odvojeno spremljeni. Rezultate izračuna također je potrebno lokalno spremiti te ih se ne smije direktno prosljeđivati upravljačkom dijelu. DSP ćelije moraju imati točno definirane, odvojene lokacije spremanja rezultata [6].

Rezultati izračuna kompleksni su brojevi te se izračun realnog i imaginarnog dijela radi odvojeno. Implementirani oblik jednadžbi (2-12) i (2-14) uz korištenje izraza (2-15) dan je izrazom (4-5).

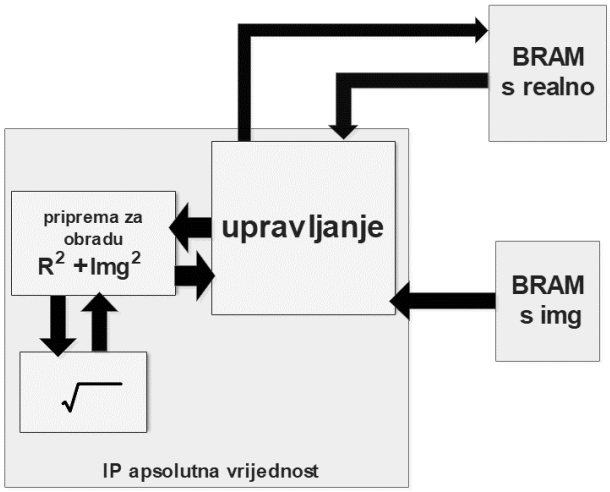
$$\begin{aligned} R1_{real} &= s1_{real} + (s2_{real} * w_{real} - s2_{img} * w_{img}) \\ R1_{img} &= s1_{img} + (s2_{real} * w_{img} + s2_{img} * w_{real}) \\ R2_{real} &= s1_{real} - (s2_{real} * w_{real} - s2_{img} * w_{img}) \\ R2_{img} &= s1_{img} - (s2_{real} * w_{img} + s2_{img} * w_{real}) \end{aligned} \tag{4-5}$$

Jednadžba (2-12) u kojoj je dan izraz za izračunavanje parnih elemenata niza rastavljena je na dva dijela. Izračun realnog dijela označen je kao $R1_{real}$, a imaginarni $R1_{img}$. Jednadžba (2-11) u kojoj je dan izraz za neparne elemente niza rastavljena je na isti način i prikazana oznakama $R2_{real}$ i $R2_{img}$.

4.3.8 Izračun apsolutne vrijednosti

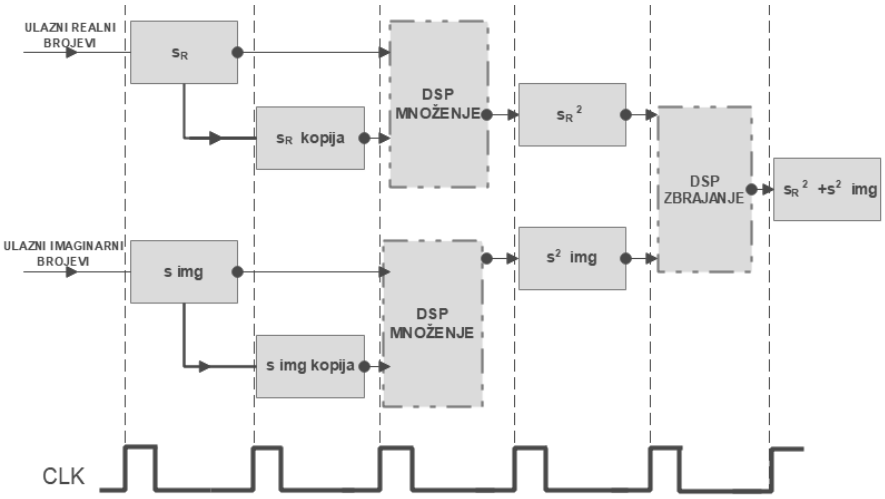
Rezultati DIT algoritma kompleksni su brojevi te je za prikaz rezultata potrebno izračunati njihove apsolutne vrijednosti. Kao što je vidljivo na slici 4.6 izrađen je IP u kojemu je implementirano adresiranje podataka, potreban izračun te spremanje podataka u BRAM. Osnovni funkcionalni dijelovi implementiranog IP-a su upravljanje, priprema podataka za izračun i izračun apsolutne vrijednosti kompleksnog broja.

Upravljački dio IP-a izvršava adresiranje i primanje podataka iz BRAMA-a, predavanje podataka na obradu i upis rezultata u BRAM. Realni i imaginarni rezultati DIT algoritma moraju se kvadrirati i zbrojiti kako bi bili predani dijelu IP-a za računanje korijena. Nakon izračunavanja korijena rezultati se predaju nazad upravljačkom dijelu koji ih sprema u BRAM. Pojednostavljena blokovska struktura s istaknutim smjerovima toka podataka prikazana je na slici 4.12.



Sl. 4.12: Pojednostavljeni blok-prikaz IP-a za izračun apsolutne vrijednosti.

Operacije zbrajanja i kvadriranja koje se izvršavaju za pripremu korjenovanja prilagođene su upotrebi DSP ćelija. Zbog veličine podataka od 22 bita operacije bi zauzimale previše resursa te bi stvarale veliku latenciju u sustavu bez upotrebe DSP ćelija. Kako bi se operacija kvadriranja izvršila upotrebom DSP ćelija, ulazne podatke potrebno je duplicirati i posebno spremiti prije predaje na obradu DSP-u. Blokovski prikaz operacija množenja i zbrajanja prikazan je slikom 4.13 [1].

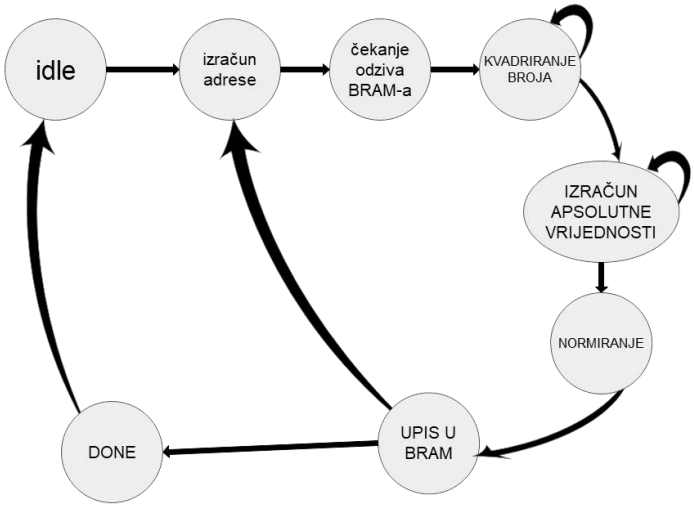


Sl. 4.13: Blokovski prikaz operacija množenja i zbrajanja uz upotrebu DSP ćelija.

Za izračun vrijednosti korijena koristi se Xilinx CORDIC v6.0 IP podešen za računanje cjelobrojnih vrijednosti. Rezultati dobiveni kvadriranjem i zbrajanjem rezultata DIT-a zaokružuju se te se IP-u prosljeđuju cjelobrojne vrijednosti. IP ima mogućnost podešavanja veličine ulaznog vektora i broja ciklusa takta potrebnih za izračun. IP ima mogućnost povezivanja AXI-Stream protokolom, ali ta mogućnost nije korištena u radu.

Nakon izračuna vrijednosti korijena rezultate je potrebno normirati, odnosno podijeliti s brojem uzoraka, u ovom slučaju 64. Dijeljenje je implementirano logičkim posmakom.

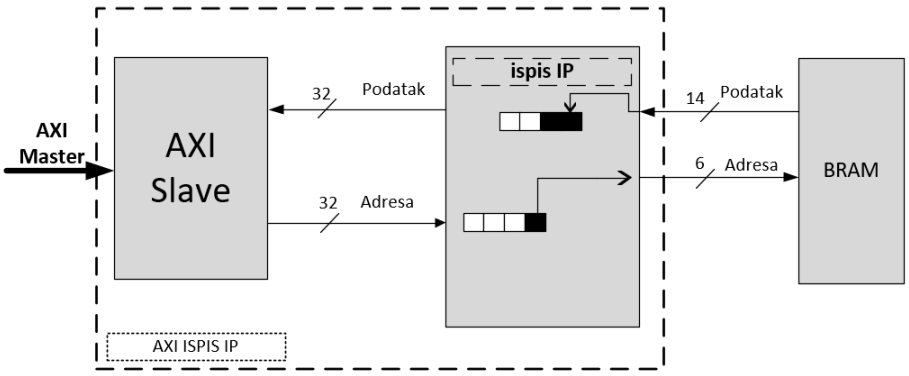
Slijed operacija implementiranog IP-a prikazan je slikom 4.14.



Sl. 4.14: Osnovna stanja implementiranog IP-a.

4.3.9 Ispis podataka

Ispis podataka odvija se putem kreiranog IP-a. Kreirani IP zapakiran je u AXI IP kako bi se ostvarila komunikacija s procesorskim sustavom. Ispis podataka izvodi se nakon što se završe sve potrebne obrade podataka. Ispis se inicijalizira od strane procesorskog sustava slanjem adrese podatka. IP dohvaća adresu te adresira podatak spremljen u BRAM-u. Nakon što je podatak očitan iz BRAM-a IP ga upisuje u AXI slave registar iz kojeg ga iščitava procesorski sustav. Ne koriste se nikakvi dodatni validacijski ili aktivacijski signali. Rutina unutar IP-a aktivira se promjenom stanja AXI registra adrese. Ispis podataka na računalo izvodi se UART/USB pretvorbom koja je moguća zbog ugrađenog UART/USB pretvornika na ZYBO razvojnoj platformi [8].



Sl. 4.15: Pojednostavljeni blokovski prikaz rada IP-a za ispis podataka.

5. TESTIRANJE I REZULTATI

5.1. Generirani testovi

Generirani testovi kombinacija su zbrajanja sinusnih signala. Signali su generirani u programskom paketu MATLAB.

Slika 5.1 prikazuje kod korišten za definiranje vremenskih i frekvencijskih vektora koji će biti korišteni za prikazivanje rezultata kao i za zadavanje broja uzoraka i izračun perioda. Najveća je frekvencija prikaza pola frekvencije uzorkovanja te je zbog toga frekvencijski vektor pola veličine vremenskog vektora [2].

Linija	Kod	
1	`N=64;`	`%Broj uzoraka`
2	`T=1/N;`	`%Period`
3	`fs=64;`	
4	`t=(0:1:N-1);`	`%Vrijeme`
5	`t=t*T;`	`%vremenski vektor`
6	`f = fs*(0:(N/2))/N; %`	`%frekvencijski vektor`

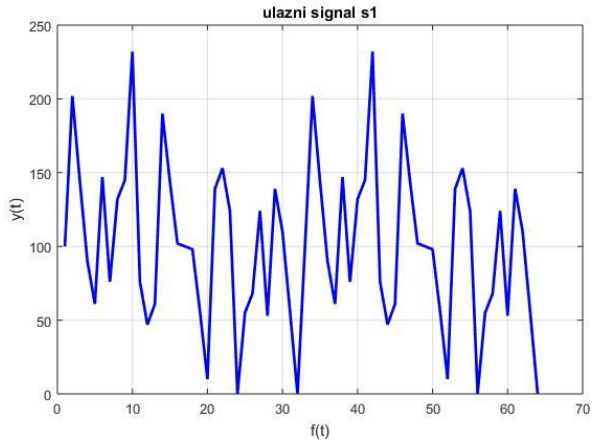
Sl. 5.1: Definiranje početnih vrijednosti.

Korišteni kod za generiranje testnih signala prikazan je slikom 5.2. Signali su definirani u 8 bitnom pozitivnom formatu. Kako bi sinusni signali bili pozitivni, a zadržali svoj oblik dodana im je istosmjerna komponenta. Dodavanje istosmjerne komponente potrebno je jer je sustav predviđen za ulazne pozitivne cjelobrojne podatke.

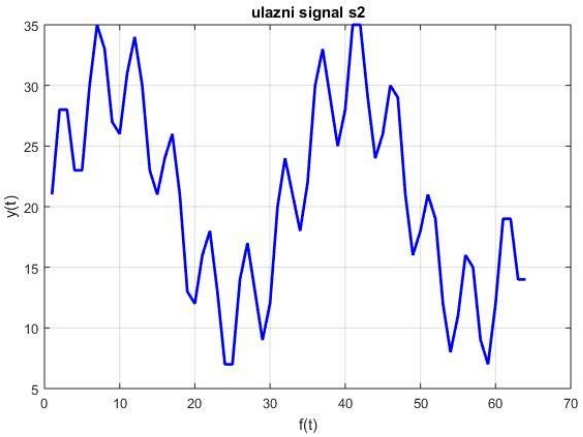
Oblik generiranih sinusnih signala prikazani su grafovima na slikama 5.3, 5.4 i 5.5.

Linija	Kod
1	`s1=uint8(100+25*sin(2*pi*2*t)+50*sin(2*pi*10*t)` `+50*sin(2*pi*16*t) + 30*sin(2*pi*30*t));`
2	`S2 =uint8(21 + 10*sin(2*pi*2*t)+5*sin(2*pi*13*t));`
3	`S3=uint8(120+25*sin(2*pi*2*t)+38*sin(2*pi*11*t)+` `40*sin(2*pi*16*t) +34*sin(2*pi*30*t));`

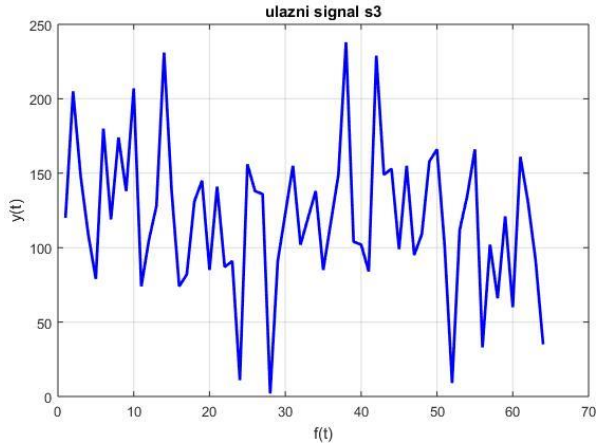
Sl. 5.2: Testni signali.



Sl. 5.3: Ulazni signal s1.



Sl. 5.4: Ulazni signal s2.



Sl. 5.5: Ulazni signal s3.

5.2. Matlab rezultati

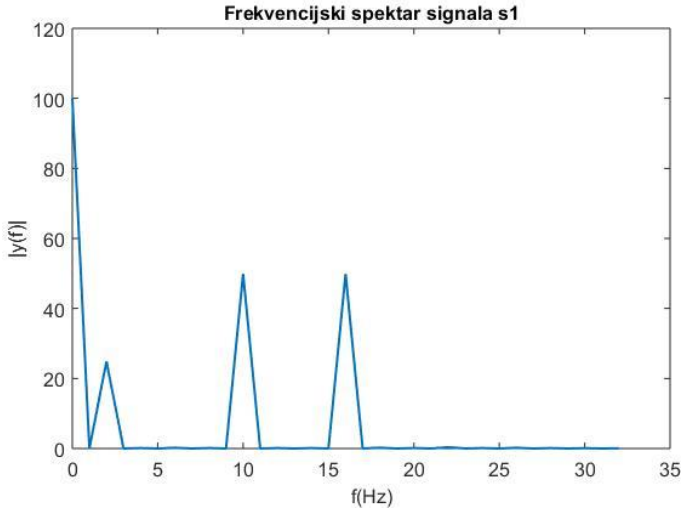
Za potrebe testiranja razvijena je MATLAB skripta koja obavlja sve izračune implementirane u sustavu. Izračuni se vrše nad definiranim testnim signalima s1, s2 i s3.

Svi testni signali grupiraju se u jednu varijablu zbog lakšeg pisanja koda. Nakon što se izračuna FFT i apsolutna vrijednost podaci se normiraju i prilagođavaju za prikaz. Korišteni MATLAB kod prikazan je slikom 5.6.

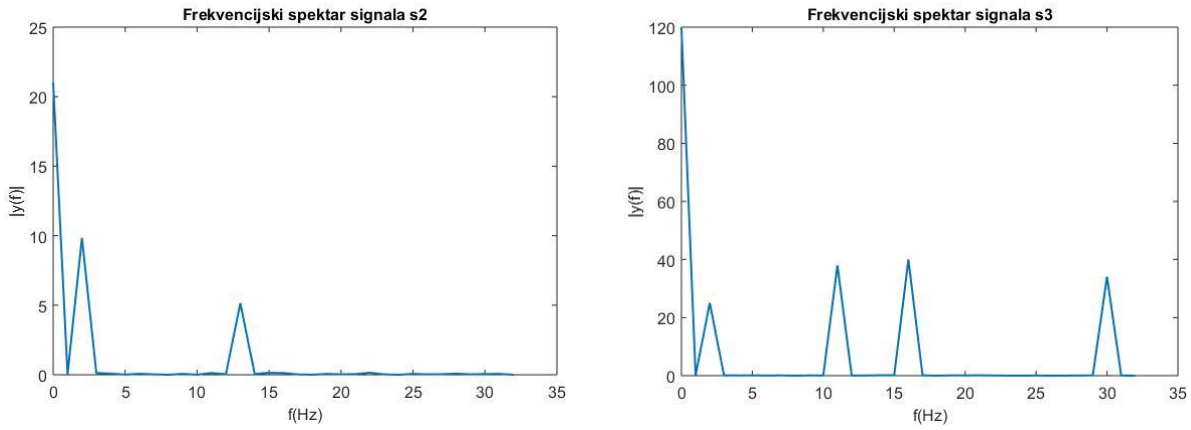
Linija	Kod	komentari
1	`s=[s1;s2;s3];`	`%`
2	`n=3;`	`%broj signala`
3	`for i = 1:n`	
4	`  y(i,:)=fft(s(i,:));`	`%FFT izracun`
5	`  kor(i,:)=abs(y(i,:)/N);`	`% apsolutna vr. i normiranje`
6	`  nor(i,:)=(kor(i,1:N/2 + 1));`	
7	`  nor(i,2:end-1) = 2*nor(i,2:end-1);`	`%prosirenje za jedan uzorak`
8	`end`	`%`

Sl. 5.6: MATLAB kod za izračunavanje rezultata.

Rezultati obrade signala prikazani su na slikama 5.7 i 5.8. Na frekvencijskim karakteristikama svih signala jasno su vidljive pojedine frekvencijske komponente koje su zadane u signalima. Najveću amplitudu ima istosmjerna komponenta koja je dodana signalima kako bi bili pozitivnih iznosa.



Sl. 5.7: Prikaz frekvencijskog spektra testnog signala s1.



a) *b)*

Sl. 5.8: Prikaz frekvencijskog spektra testnih signala s1 i s2.

5.3. Rezultati sustava

5.3.1 Točnost obrada podataka

Nakon što su izgenerirani signali u MATLAB-u unose se u Xilinx SDK pomoću kojeg se programira procesorski sustav ZYNQ SoC-a.

U svrhu analize rezultata razvijena je MATLAB skripta. Analizirana je apsolutna i relativna pogreška implementiranog sustava s obzirom na rezultate u MATLAB-u. Korišteni MATLAB rezultati u računanju apsolutne pogreške nisu zaokruženi na cjelobrojne vrijednosti kako bi se odredilo odstupanje rezultata sustava od stvarnih vrijednosti. Kod računanja relativne pogreške MATLAB rezultati zaokruženi su na cjelobrojne vrijednosti. Za vrijednosti relativne pogreške implementirano je ispravljanje nedefiniranih rezultata (NaN).

Izračun postotne i apsolutne pogreške dan je izrazom (5-1).

$$E_{REL\,\%} = \left|\frac{X_{FPGA} - X_{MATLAB}}{X_{MATLAB}}\right| * 100$$
$$E_{Aps} = |X_{FPGA} - X_{MATLAB}| \quad (5\text{-}1)$$

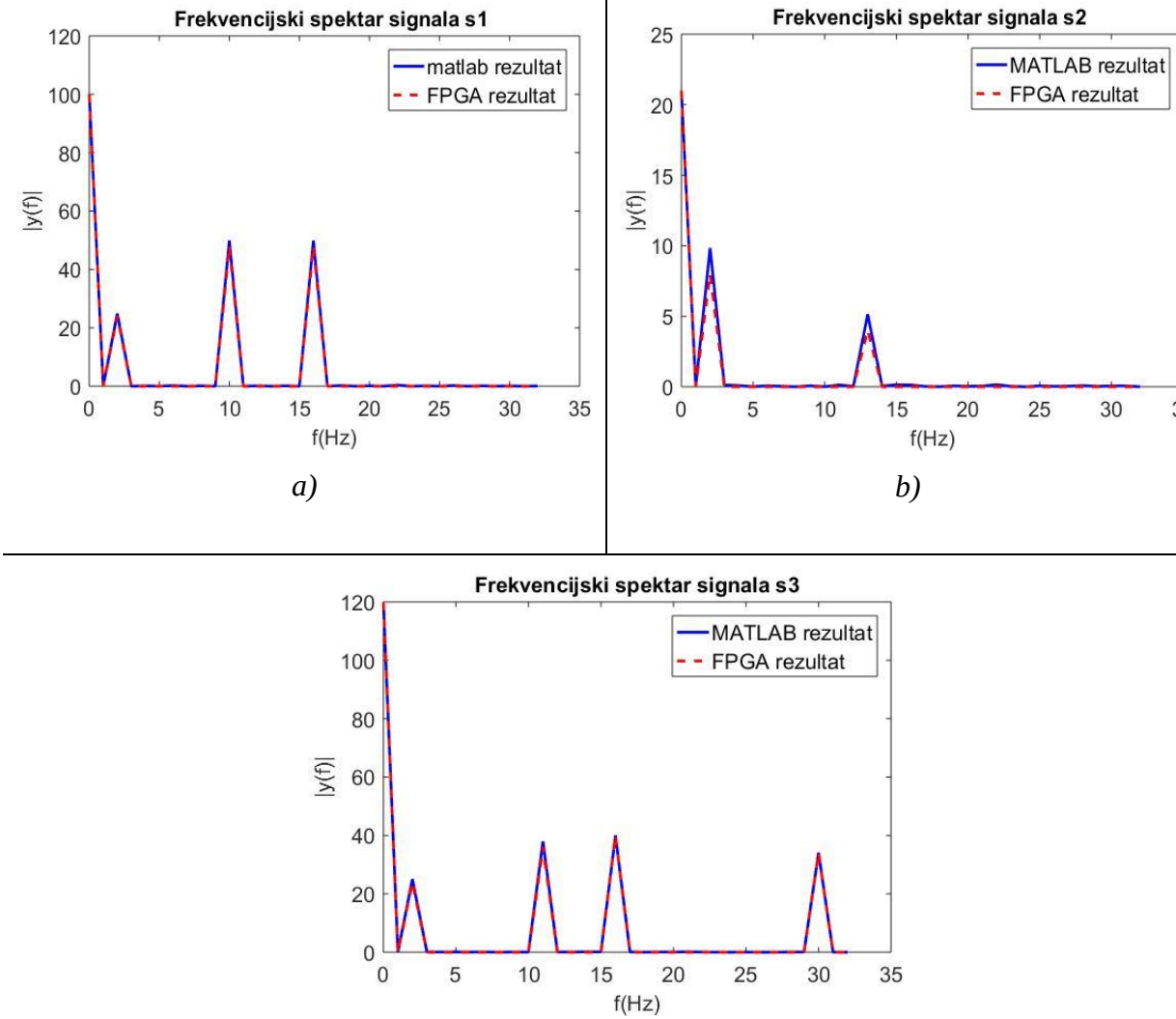
Slika 5.9 prikazuje kod za izračun apsolutne i postotne greške.

Linija Kod

```
1       error_aps(i,:)=abs((nor_matlab(i,:)) - nor_fpga(i,:));
3       error_pos(i,:)=(round(error_aps(i,:))./nor_matlab(i,:))*100;
```

Sl. 5.9: Kod korišten za izračun apsolutne i postotne greške.

Na slici 5.10 prikazani su rezultati MATLAB izračuna i FPGA izračuna. Vidljivo je veće odstupanje za signal s2. Generirani signal s2 manje je amplitude od signala s1 i s3 te je to razlog većeg odstupanja. Apsolutna vrijednost kompleksnih rezultata FFT-a računa se iz zaokružene vrijednosti zbroja kvadrata realnog i imaginarnog dijela broja. Postupak unosi veće pogreške za manje vrijednosti amplitude početnog signala.

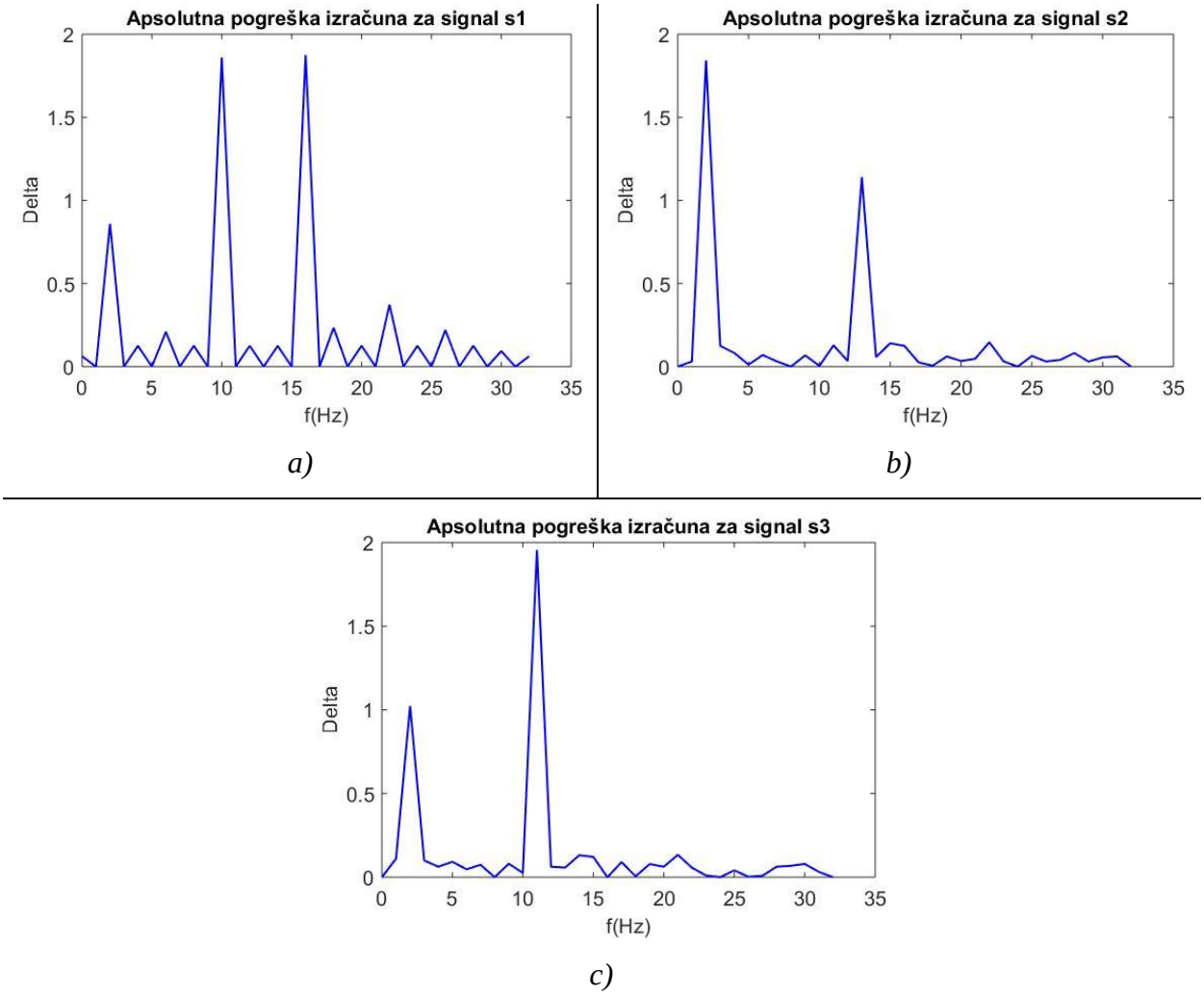


a) *b)*

c)

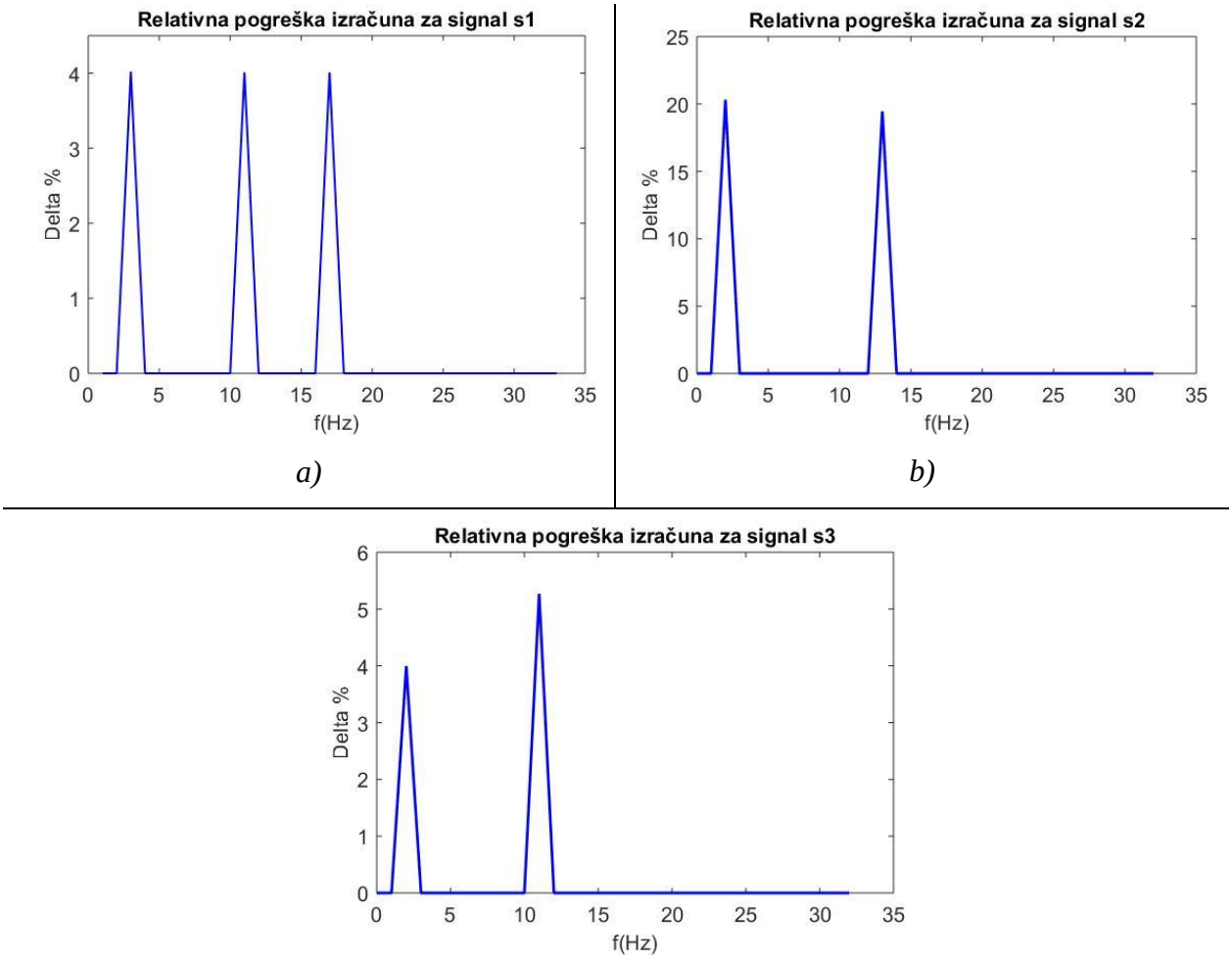
Sl. 5.10: Usporedba rezultata, MATLAB i FPGA.

Na slici 5.11 prikazana je apsolutna pogreška implementiranog sustava s obzirom na MATLAB izračun. Izračun je implementacija izraza danog u (5-1). Na grafovima su vidljive manje pogreške koje nastaju kao rezultat korištenja ne cjelobrojnih rezultata MATLAB izračuna i cjelobrojnih vrijednosti FPGA izračuna.



Sl. 5.11: Apsolutne vrijednosti grešaka implementiranog sustava.

Izračun relativne pogreške sustava izveden je po izrazu danom u (5-1). MATLAB rezultati zaokruženi su na cjelobrojne vrijednosti. Grafovi pogrešaka za pojedine signale prikazani su slikom 5.12. Relativne pogreške rezultata implementiranog sustava manje su za signale s1 i s3 nego za signal s2. Iz slike 5.4 vidljivo je da signal s2 ima manje vrijednosti amplitude od signala s1 i s3 što nepovoljno utječe na točnost izračuna. Greške su izražene postotno.



Sl. 5.12: Relativna pogreška implementiranog sustava.

5.3.2 Brzina obrade podataka

Mjerenje brzine obrade implementirano je brojanjem signala takta i mjerenjem vremena. Mjerenje vremena implementirano je u procesorskom sustavu koristeći gotove ugrađene funkcije. Mjeri se vrijeme koje je potrebno procesoru za izvršavanje upisa i ispisa podataka. Upis i ispis izvedeni su pomoću *for* petlji, u petljama se nalazi samo nužan kod za upis i ispis te se ne izvode ispisi u konzolu kako ne bi utjecali na rezultate mjerenja.

Brojanje signala takta implementirano je u PL dijelu sustavu. Brojači su implementirani u upravljački IP te broje koliko je taktova upravljački IP u pojedinom stanju automata. Veza s procesorskim sustavom ostvaruje se pomoću AXI GPIO IP-a. Mjerenje je izvedeno za jedan signal od 64 uzorka.

Mjerenja izvedena na implementiranom sustavu uspoređena su s vremenom koje je potrebno MATLAB programskom paketu za izvršavanje istih operacija. Mjerenje je izvedeno za jedan signal od 64 uzorka te je izvršeno u 100 000 iteracija kako bi se dobio statistički značajan podatak. Signal je definiran kao uint8 podatak kako bi ulazni podaci bili isti. Mjerenja u MATLAB-u izvedena su tic-toc metodom. Korištena MATLAB skripta prikazana je slikom 5.13. Dodatna mjerenja izvedena su samo za FFT izračun kao i za apsolutnu vrijednost i normiranje. Dodatna mjerenja napravljena su prepravljanjem postojeće skripte, odnosno ne računanjem jednog od izračuna kako bi se posebno izmjerilo vrijeme drugog.

Specifikacije računala korištenog za izvedbu MATLAB skripte dane su tablicom 5.1. Korištena je MATLAB 16.0a verzija.

Tab. 5.1: Specifikacije računala korištenoga za izvođenje MATLAB izračuna

Procesor	Intel core i5 6402p, 2.8-3.4 GHz
Radna memorija	8 GB DDR4, 2144 MHz
Matična ploča	AsRock B150 HDI
Vanjska memorija	SSD AMD 120 GB
Operacijski sustav	Windows 10, 10.0.16299 education, x64

Linija **Kod**

```
1       N = 100000;
2       data = uint8(rand(1,64)*255);
3       tic                                     %pocni mjerenje
4       for i=0:N
5         Fdata = fft(data);                    %FFT izracun
6         spectar = abs(Fdata/64);              %ap. vr. i normiranje
7       end
8       time = toc                              %zavrsi s mjerenjem
9       (time/N)*1000000
```

Sl. 5.13: Mjerenje vremena MATLAB obrade.

Nabrojani broj taktova u PL mjerenju potrebno je preračunati u vrijeme. Frekvencija takta implementiranog sustava je 45.45 MHz. Izraz za računanje vremena u ovisnosti o frekvenciji i stanju brojača dan je jednadžbom (5-2).

$$t = \frac{1}{f} n_{brojač} \tag{5-2}$$

Izračunana vremena obrade dana su tablicom 5.2. Implementirani sustav sporiji je od MATLAB obrade na računalu. Tablicom su dana i vremena upisa i ispisa podataka iz PL-a iako to ne utječe na vrijeme obrade te se u daljnjoj analizi zanemaruje.

Tab. 5.2: Usporedba vremena obrade za FPGA i MATLAB.

Operacija	**Broj taktova**	**FPGA (µs)**	**MATLAB (µs)**	$\frac{\text{FPGA (µs)}}{\text{MATLAB (µs)}}$
FFT	1542	33,92	4,45	7,6
Ap. vrijednost	2054	45,18	3,01	14,9
FFT+Ap. vrijednost	**3596**	**79,1**	**7,47**	**10,5**
Upis	–	110		
Ispis	–	50		
Ukupno	–	239		

Iz podataka za vrijeme obrade u tablici 5.2 izračunano je koliko je implementirani sustav sporiji od MATLAB-a.

Temeljna frekvencija rada procesora korištenog u računalu je 2,8 GHz. Kada bi implementirani sustav radio na frekvenciji od 2,8 GHz, obrada bi podataka bez upisa i ispisa trajala 1,2843 µs. Na istoj frekvenciji rada implementirani sustav bio bi brži 5,8 puta od računalne MATLAB obrade. Detaljni pregled usporedbe rada na istoj teorijskoj frekvenciji dan je tablicom 5.3.

Tab.5.3: Teorijska usporedba brzine rada FPGA sustava i MATLAB-a na istoj radnoj frekvenciji.

Operacija	**FPGA (µs)**		**MATLAB (µs)**	$\frac{\text{MATLAB (µs)}}{\text{FPGA (µs)}}$
	45.45 MHz	**2.8 GHz**		**2.8 GHz**
FFT	33.92	0,55	4,45	8,1
Ap. vrijednost	45.18	0,73	3,01	4,12
Ukupno	**79.1**	**1,2843**	**7,47**	**5,8**

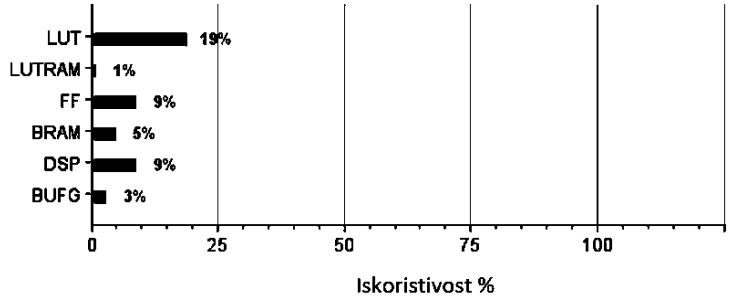
Operacije upisa i ispisa podataka zanemaruju se u promatranju brzine rada implementiranog sustava jer nisu dio obrade podataka. Resursi AXI sabirnice nisu u potpunosti iskorišteni jer se podaci šalju jedan po jedan i iskorištava se samo 8 od 32 bita širine AXI registara. Unos podataka dodatno je usporen validacijskim operacijama nakon svakog prijenosa. Validacijski signal nije potreban ako se iskoriste AXI signali za validaciju prijenosa. Upis podataka dvostruko je sporiji od ispisa zbog upotrebe validacijskog signala.

5.4. Zauzeće resursa

Zauzeće resursa bio je najbitniji faktor u razvoju sustava. Iskorištenost resursa prikazana je tablicom 5.4 i grafički slikom 5.14. Raspored zauzetosti resursa PL-a prikazan je slikom 5.15. Potrošnja energije pojedinih IC-ova, procesorskog sustava i korištenih signala prikazana je tablicom 5.5.

Tab. 5.4: Iskorištenost resursa ZYNQ SoC-a.

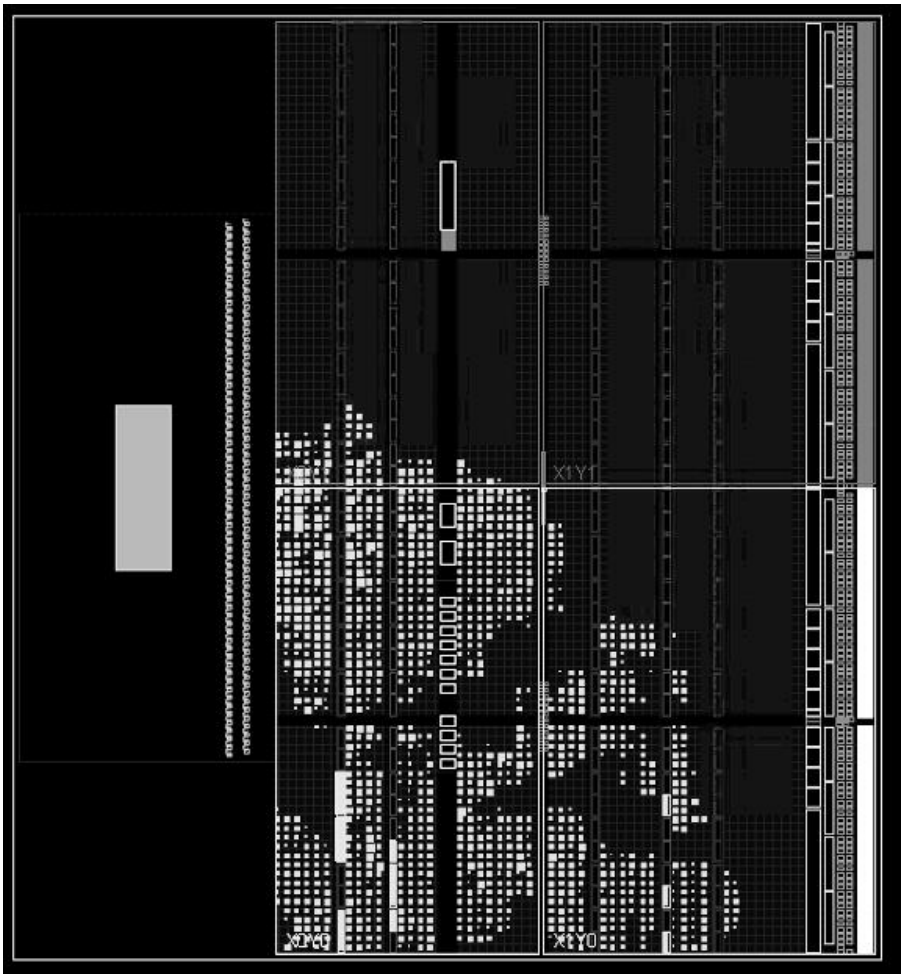
Resurs	Iskorišteno	Dostupno	Iskorišteno %
LUT	3354	17600	19,06
LUTRAM	87	6000	1,45
BISTABILI (FF)	3278	35200	9,31
BRAM	3	60	5,0
DSP	7	80	8,75
BUFG	1	32	3,13



Sl. 5.14: Grafički prikaz iskorištenosti resursa sustava.

Tab. 5.5: Potrošnja energije u sustavu.

	Potrošnja energije (W)
Procesorski sustav	1,555
Signali takta	0,006
Signali	0,004
Programabilna logika	0,003
BRAM	<0,0001
DSP	<0,0001
Statična potrošnja	0,127
Ukupno	1,695



Sl. 5.15: Raspored zauzetosti programabilne logike.

6. ZAKLJUČAK

Radom je demonstrirana implementacija FFT algoritma u FPGA. Algoritam zahtijeva brojne kalkulacije pa se obrada morala raditi serijski kako bi implementirani sustav imao dovoljno resursa za rad. Pokušaji dodatne paralelizacije dizajna nisu uspjeli zbog prevelikih resursnih zahtjeva.

Operacija izračuna kompleksnih koeficijenata brža je od operacije izračuna korijena jer je više vremena utrošeno na optimizaciju tog dijela dizajna.

U radu se korištenje DSP ćelija pokazalo kao ključni faktor pri ubrzavanju rada sustava. Implementirani sustav izvršava svoju zadaću, ali su moguća brojna poboljšanja. Da bi se dizajn mogao koristiti za detekciju većih frekvencije potrebna je dodatna optimizacija samog izračuna, kako bi se mogao proširiti za upotrebu s više uzoraka. Najveća poboljšanja moguća su pri upisu i ispisu podataka koji zajedno traju više nego cijela obrada podataka. Operacije upisa i ispisa podataka izvršavaju se putem AXI4 sabirnice koju je potrebno bolje iskoristiti.

Točnost i brzina rada implementiranog sustava uspoređeni su s rezultatima programskog paketa MATLAB. Točnost je sustava zadovoljavajuća i greške ne prelaze vrijednost od 5% za ulazne vrijednosti signala s istosmjernom komponentom većom od 100. Za ulazne signale s manjom vrijednošću amplituda pojavljuju se značajnije greške u izračunu amplitude pojedinih frekvencija. Implementiranom sustavu potrebno je više vremena za izračun nego MATLAB programskom paketu. Ako bi oba sustava radila na istoj frekvenciji implementirani sustav bio bi 5.8 puta brži od MATLAB programskog paketa.

LITERATURA

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

[2] A. V. Oppenheim, R. W. Schafer, J. R. Buck, *Discrete-Time Signal Processing*, 2nd ed. , Prentice Hall, New Jersey, 1998.

[3] D. L. Jones, *Decimation-in-time (DIT) Radix-2 FFT*, OpenStax-CNX, 2016.

[4] V. A. Pedroni, *Circuit Design and Simulation with VHDL*, second edition, The MIT Press, Cambridge, Massachusetts, 2010.

[5] C. Maxfield, *The Design Warrior's Guideto FPGA's*, Newnes, 2004.

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

[7] Xilinx Inc. , *Zynq-7000 All Programmable SoC, Technical Reference Manual*, Xilinx Inc. , 2016.

[8] Digilent INC. , *ZYBO board refrence manual*, 2017.

[9] D. Bishop, *Fixed point package user's guide*, IEEE P1076 Working Group, 2015.

[10] Xilinx Inc. , *CORDIC v6.0 LogiCORE IP Product Guide*, December 20, 2017.

SAŽETAK

Naslov: VHDL implementacija FFT-a za obradu audio signala.

U radu je opisana implementacija FFT algoritma u FPGA tehnologiji. Opisana je primijenjena tehnologija i svi korišteni alati u razvoju. Demonstrirano je na koji se način izvršava FFT algoritam, matematička podloga rješenja i optimizacija algoritma za rad na sklopovlju. Opisane su sve korištene komponente i protokoli. Detaljno je opisan način rada svake konstruirane komponente. Dizajn sustava izveden je u Vivado razvojnom okruženju, a implementiran u ZYNQ 7000 SoC-u. Brzina i točnost sustava uspoređeni su s brzinom rada i točnošću MATLAB programskog paketa.

Ključne riječi: Fourier, FFT, Radix-2, FPGA, VHDL, Vivado, Zybo, ZYNQ.

ABSTRACT

Title: VHDL implementation of FFT algorithm for audio signal processing.

This Paper describes the implementation of FFT algorithm in FPGA technology. Paper gives a description of all technology and development tools used for development and implementation of design. Implementation and running of FFT algorithm are demonstrated as well as mathematical description of algorithm. All of created components are described in detail. Design of the implemented system has been made in VIVADO design suite and is implemented in ZYNQ 7000 SoC. Performance of the system is compared to that of a MATLAB.

Keywords: Fourier, FFT, Radix-2, FPGA, VHDL, Vivado, Zybo, ZYNQ.

ŽIVOTOPIS

Marijan Mautner rođen je 31. kolovoza 1991. godine u Požegi od majke Ljiljane i oca Jakoba. Živi s obitelji u Novim Bankovcima, a osnovnu školu „Ivan Goran Kovačić“ pohađa u Velikoj. Po završetku osnovne škole upisuje srednju Tehničku školu u Požegi, smjer tehničar za računalstvo. U srpnju 2010. g. upisuje preddiplomski smjer Elektrotehnike na Elektrotehničkom fakultetu u Osijeku. Diplomski studij računarstva upisuje u rujnu 2015. godine, smjer procesno računarstvo. Suradnju s tvrtkom „Institut RT-RK Osijek“ započinje u travnju 2017. g. gdje se usavršava u području FPGA tehnologije.

Potpis

_____

PRILOG

Na CD-u priloženi su:

1. cjelokupni projekt izrađen u Vivado Design Suit-u
2. kreirani IP-evi
3. kreirane VHDL datoteke
4. C programski kod
5. korištene MATLAB datoteke
6. korištene Python skripte
7. sadržaj priloga.