UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
KATEDRA ZA MIKRORAČUNARSKU ELEKTRONIKU

JEDNO HARDVERSKO-SOFTVERSKO KODIZAJN REŠENJE AKCELERATORA ZA KONVOLUCIJU MATRICA KORIŠĆENJEM BRZIH FURIJEOVIH TRANSFORMACIJA

diplomska teza (bečelor)

kandidat
Vladimir Vincan, EE5-2015

mentor
dr Vuk Vranković, vanredni profesor

Jul 2014

Sadržaj

Slike

Listing koda

Glava 1

Uvod

Operacija konvolucije predstavlja bazičan algoritam u velikom broju aplikacija, među kojima se u značajnije predstavnike ubrajaju digitalna obrada slike i konvolucione neuronske mreže. Akceleracija date operacije bi značajno ubrzala algoritme zasnovane na konvoluciji, zato što vreme izvršavanja datih algoritama dominantno zavisi od konvolucije.

Naivna implementacija konvolucije nad diskretnim dvodimenzionim signalom sa $N \cdot M$ elemenata ima složenost $\mathcal{O}(N^2 \cdot M^2)$. Korišćenjem brze Furijeove transformacije i osobina koje povezuju konvoluciju sa brzom Furijeovom transformacijom, vremenska složenost algoritma se može svesti na $\mathcal{O}(N \cdot M \cdot (\log N + \log M))$. Dati projekat realizuje jedno softversko – hardversko kodizajn rešenje implementacije konvolucije matrica, zasnovano na ESL metodologiji.

Rad se sastoji iz sledećih celina:

1. Prvog, uvodnog poglavlja.
2. U drugom poglavlju su objašnjene operacije konvolucije matrica i brze Furijeove transformacije, njihova veza, kao i osobine koje smanjuju složenost implementiranog algoritma.
3. U trećem poglavlju je opisan tok projektovanja dizajna na sistemskom nivou. To podrazumeva vremensku i prostornu analizu performansi sistema, kao i particionisanje na hardverski i softverski deo.
4. U četvrtom poglavlju su diskutovane moguće implementacije, i opisan je projketovani hardverski (IP) blok.
5. U petom poglavlju je opisan postupak funkcionalne verifikacije projektovanog hardverskog (IP) bloka, zasnovanog na UVM metodologiji.
6. U šestom poglavlju je opisan implementirani Linuks drajver projektovanog hardverskog (IP) bloka.

Glava 2

Konvolucija matrica

2.1 Konvolucija u jednoj dimenziji

Posmatrajmo dva niza, *f[n]* i *g[m]*, sa brojem elemenata N i M ($N \geq M$), redom. Pomoću njih će biti objašnjena dva osnovna tipa diskretne konvolucije – linearna i kružna.

- **Linearna konvolucija** se zasniva na proširivanju signala f i g sa nulama i sa leve i sa desne strane, do beskonačnosti, i potom izvršavanje sledeće operacije:

$$h_\infty[k] = (f * g)[k] \triangleq \sum_{i=-\infty}^{\infty} f_\infty[i] \cdot g_\infty[k-i] = (g * f)[k]$$

  Pri čemu važi:

$$f_\infty[n] = \begin{cases} f[n], 0 \leq n \leq N \\ 0, inače \end{cases}$$

$$g_\infty[m] = \begin{cases} g[n], 0 \leq m \leq M \\ 0, inače \end{cases}$$

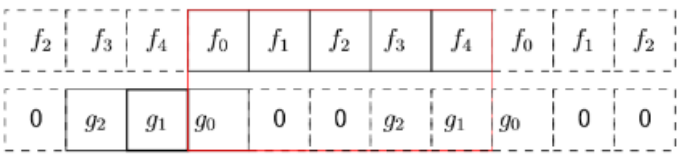
  Rezultujući signal h_∞ može imati maksimalno *N+M+1* nenultih vrednosti. U zavisnosti od toga da li konvolucioni proizvod uključuje sve nenulte članove dobijenog niza h_∞, razlikujemo tri tipa konvolucionog množenja:

  - **Potpuno množenje** – rezultujući niz h će uključivati sve elemente niza h_∞, koji mogu imati nenultu vrednost. Niz h će posedovati *N+M+1* elemenata.
  - **Množenje iste veličine** – rezultujući niz h će uključivati samo članove niza h_∞, kod kojih je centralni element niza g pomnožen sa sa nekim članom neproširenog niza f. Niz h će posedovati N elemenata.
  - **Validno množenje** – rezultujući niz h će uključivati samo članove niza h_∞, kod kojih su svi neproširenи elementi niza g pomnoženi sa sa neproširenim članovima niza f. Niz h će posedovati *N-M+1* elemenata.

  Dalje u radu će se pod linearnim konvolucionim množenjem podrazumevati potpuno množenje, osim ukoliko ne bude drugačije naznačeno, pošto je algoritam za konvoluciju implementiran pomoću potpunog množenja. Matematički, formula za potpuno konvoluciono množenje se definiše na sledeći način:

$$h[k] \triangleq \sum_{i=\max(0,k-M+1)}^{\min(N-1,k)} f[i] \cdot g[k-i], \qquad \forall k \in [0, N+M-2]$$

- **Kružna konvolucija** se zasniva na „obmotavanju“ nizova f i g sa sopstvenim članovima, umesto što se proširuju sa nulom do beskonačnosti. Time se postiže da novodobijeni nizovi f_P i g_P budu periodični sa istim periodom P. Konačni nizovi f i g su prošireni sa nulama tek toliko, da bi se zadovoljio uslov da nizovi budu iste periodičnosti *($P = N_P = M_P \geq N \geq M$)*. Drugim rečima, za bilo koji ceo broj k, važi *f(k) = f(k mod P)* i *g(k) = g(k mod P)*. Na slici 1 se nalazi primer za *P=N=5* i *M=3*.



Slika 1 Izgled nizova *f* i *g* za P=N=5 i M=3

Matematički, formula za potpuno konvoluciono množenje sa modulom P se definiše na sledeći način:

$$h_P[k] = (f *_P g)[k] \triangleq \sum_{i=0}^{P-1} f_P[i] \cdot g_P[k-i] = (g *_P f)[k], \qquad \forall k \in \mathbb{Z}$$

Pri čemu važi:

$$f_P[i] = \sum_{p=-\infty}^{\infty} f_\infty[i + p \cdot P] = f_\infty[i \bmod P], \ \forall i \in \mathbb{Z}$$

$$g_P[i] = \sum_{p=-\infty}^{\infty} g_\infty[i + p \cdot P] = g_\infty[i \bmod P], \ \forall i \in \mathbb{Z}$$

U slučaju da je moduo kružne konvolucije P isti kao veličina niza *f (P = N)*, jednostavno se može pokazati da linearna i kružna konvolucija predstavljaju identičnu operaciju:

$$(f * g)[k] = (f *_P g)[k], \quad ako\ P = N$$

Uloga i značaj pojma kružne konvolucije je u tome što ona predstavlja spregu između diskretne Furijeove transformacije i linearne konvolucije signala.

2.2 Konvolucija u dve dimenzije

Celokupna teorija diskutovana za konvoluciju u jednoj dimenziji može biti primenjena i na konvoluciju u više dimenzija. Ovaj rad će biti ograničen na dve

dimenzije, odnosno ograničen na konvolucije nad matricama. Linearna konvolucija nad matricama f i g se definiše na sledeći način:

$$h_\infty[k_1, k_2] \triangleq (f * g)[k_1, k_2]$$
$$= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f_\infty[i,j] \cdot g_\infty[k_1 - i, k_2 - j] = (g * f)[k_1, k_2]$$

Pri čemu važi:

$$f_\infty[i,j] = \begin{cases} f[i,j], 0 \le i \le H_1, 0 \le j \le W_1, \\ 0, inače \end{cases}$$

$$g_\infty[i,j] = \begin{cases} g[i,j], 0 \le i \le H_2, 0 \le j \le W_2, \\ 0, inače \end{cases}$$

Potpuna linearna konvolucija će biti matrica veličine *(H_1+H_2-1, W_1+W_2-1)*. Kružno konvoluciono množenje nad matricama f i g se definiše na sledeći način:

$$h_P[k_1, k_2] \triangleq \left(f *_{P_1,P_2} g\right)[k_1, k_2] =$$

$$= \sum_{i=0}^{P_1-1} \sum_{j=0}^{P_2-1} f[i,j] \cdot g[k_1 - i, k_2 - j] = \left(g *_{P_1,P_2} f\right)[k_1, k_2],$$

$$\forall k_1, k_2 \in \mathbb{Z}$$

Pri čemu važi:

$$f_{P_H,P_W}[i,j] = \sum_{p_1=-\infty}^{\infty} \sum_{p_2=-\infty}^{\infty} f_\infty[i + p_1 \cdot P_H, j + p_2 \cdot P_W]$$
$$= f_\infty[i \bmod P_H, j \bmod P_W], \quad \forall i,j \in \mathbb{Z}$$

$$g_{P_H,P_W}[i,j] = \sum_{p_1=-\infty}^{\infty} \sum_{p_2=-\infty}^{\infty} g_\infty[i + p_1 \cdot P_H, j + p_2 \cdot P_W]$$
$$= g_\infty[i \bmod P_H, j \bmod P_W], \ \forall i,j \in \mathbb{Z}$$

I u dvodimenzionalnom slučaju se može pokazati da kružna i linearna konvolucija predstavljaju identičnu operaciju ukoliko važi da je $P_H = H_1 \ge H_2$ i $P_W = W_1 \ge W_2$.

2.3 Diskretna Furijeova transformacija u jednoj dimenziji

Diskretna Furijeova transformacija nad diskretnim signalom *f[n]* dužine *N* se definiše na sledeći način:

$$F[k] = \mathcal{F}\{f\}[k] \triangleq \sum_{n=0}^{N-1} f[n] e^{-\frac{2i\pi kn}{N}} = \sum_{n=0}^{N-1} f[n] W_N^{kn}$$

Vrednosti kompleksne eksponencijalne funkcije sa kojima se množe odbirci diskretnog signala *f[n]* nazivaju se rotacioni „twiddle“ faktori. Rotacioni faktori imaju sledeće osobine, koje će biti značajni za izvođenje brze Furijeove transformacije:

- Kompleksno konjugovana simetričnost

$$W_N^{k(N-n)} = W_N^{-kn} = \left(W_N^{kn}\right)^C$$

- Periodičnost

$$W_N^{kn} = W_N^{k(n+N)} = W_N^{(k+N)n}$$

- Redundantnost

$$W_N^{2kn} = W_{N/2}^{kn}$$

Inverzna diskretna Furijeova transformacija nad istim signalom se definiše na sledeći način:

$$f[n,m] = \mathcal{F}^{-1}\{F\}[n] \triangleq \frac{1}{N}\sum_{k=0}^{N-1} F[k]e^{\frac{2i\pi kn}{N}} = \frac{1}{N}\left(\sum_{k=0}^{N-1} F[k]^C e^{-\frac{2i\pi kn}{N}}\right)^C$$

Iz formule za inverznu Furijeovu transformaciju se može zaključiti da Furijeova transformacija nad konjugovanim signalom u frekvencijskom domenu predstavlja inverznu Furijeovu transformaciju. Konjugacija nad rezultujućim signalom nije neophodna, pošto za dati algoritam su značajne samo realne vrednosti inverzne Furijeove transformacije. Korišćenjem date osobine se povećava konciznost koda, i omogućava se korišćenje već realizovane funkcije za direktnu Furijeovu transformaciju, sa neznatnim usporavanjem vremena izvršavanja. U embeded aplikacijama, sa memorijskim ograničenjima, ova osobina se može pokazati veoma korisnom.

2.4 Diskretna Furijeova transformacija u dve dimenzije

Diskretna Furijeova transformacija nad diskretnim višedimenzionim signalom *f[n]* se definiše analogno kao u jednodimenzionom slučaju. Formula za Furijeovu transformaciju u dve dimenzije nad matricom *f* dimenzija *(N, M)* glasi:

$$F[k,l] = \mathcal{F}\{f\}[k,l] \triangleq$$

$$\triangleq \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} f[n,m]e^{-\frac{2i\pi lm}{M}}e^{-\frac{2i\pi kn}{N}}$$

$$= \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} f[n,m]W_M^{lm}W_N^{kn}$$

$$= \sum_{n=0}^{N-1} \left(\sum_{m=0}^{M-1} f[n,m] W_M^{lm} \right) W_N^{kn}$$

Data jednačina pokazuje kako se pomoću jednodimenzione diskretne Furijeove transformacije može dobiti dvodimenziona Furijeova transformacija. Prvo se izvrši Furijeova transformacija nad svim kolonama, a potom po svim vrstama, ili obratno, da bi se dobila željena dvodimenziona transformacija. Ova osobina je značajna, jer pojednostavljuje implementaciju algoritma, i može se primeniti nad beskonačno dimenzija.

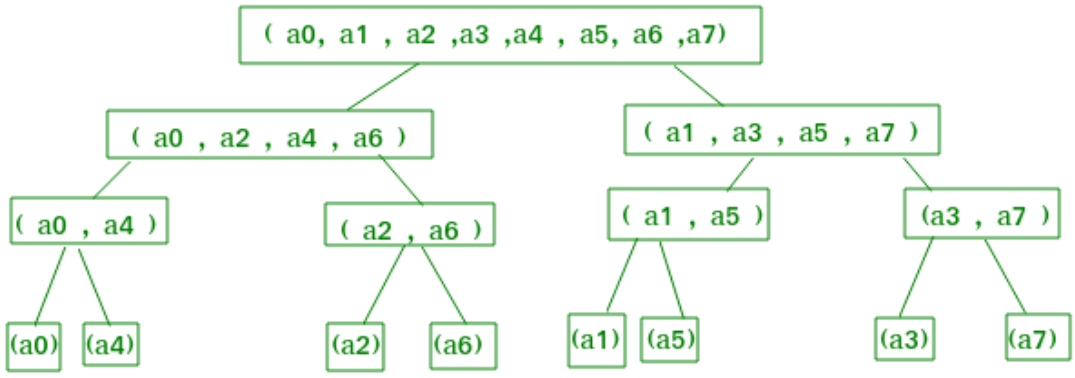
Inverzna Furijeova u dve dimenzije se može realizovati izvšavanjem dvodimenzione Furijeove transformacije nad konjugovanom matricom:

$$f[n,m] = \mathcal{F}^{-1}\{F\}[n,m] = (\mathcal{F}\{F^C\}[n,m])^C$$

2.5 Brza Furijeova transformacija

Postoji veliki broj algoritama koji smanjuju vreme izvršavanja jednodimenzione Furijeove transformacije sa $\mathcal{O}(N^2)$ na $\mathcal{O}(N \log N)$, kao što su Kuli-Tukijev algoritam („Radix 2“), Prajm faktor algoritam, Raderov algoritam, Vinogradov algoritam, i drugi. U ovom radu će biti implementiran „Radix 2“ algoritam u vremenskom domenu, pošto predstavlja najstariji i najpoznatiji algoritam za brzu Furijeovu transformaciju.

„Radix 2“ postiže ubrzanje u odnosu na diskretnu Furijeovu transformaciju izbegavanjem ponovnog računanja određenih izraza. Pripada klasi „zavadi pa vladaj“ algoritama, i kao preduslov neophodno je da broj članova niza bude stepen dvojke. Tokom preprocesiranja, izmenimo redosled članova niza pomoću algoritma „bit reversal“. Potom, rekurzivnim ponavljanjem, podelimo članove na parne i neparne, izračunamo brzu Furijeovu transformaciju za oba novodobijena niza i spojimo rezultate, koristeći osobine korena jedinice (rotacionih faktora) u polju kompleksnih brojeva (slika 2).



Slika 2 Rekurzivno izvršavanje brze Furijeove transformacije

U narednih nekoliko koraka će biti izvedene formule za „Radix 2“ algoritam brze Furijeove transformacije:

$$F[k] = \mathcal{F}\{f\}[k] = \sum_{n=0}^{N-1} f[n] W_N^{nk} =$$

$$\sum_{n=0}^{\frac{N}{2}-1} f[2n] W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} f[2n+1] W_N^{(2n+1)k} =$$

$$\sum_{n=0}^{\frac{N}{2}-1} f[2n] W_N^{2nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} f[2n+1] W_N^{2nk}$$

$$k \in [0, \dots, N-1]$$

Na osnovu osobine redundantnosti, a potom i periodičnosti rotacionih faktora, formula za Furijeovu transformaciju postaje:

$$F[k] = \sum_{n=0}^{\frac{N}{2}-1} f[2n] W_{N/2}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} f[2n+1] W_{N/2}^{nk} = F_p[k] + W_N^k F_n[k]$$

$$k \in [0, \dots, \frac{N}{2} - 1]$$

Na osnovu osobine kompleksno konjugovane simetričnosti, redundantnosti i periodičnosti, može se jednostavno, na osnovu već izračunatih vrednost, dobiti vrednost člana F[k+N/2]:

$$F\left[k + \frac{N}{2}\right] = F_p\left[k + \frac{N}{2}\right] + W_N^{k+\frac{N}{2}} F_n\left[k + \frac{N}{2}\right] = F_p[k] - W_N^k F_n[k]$$

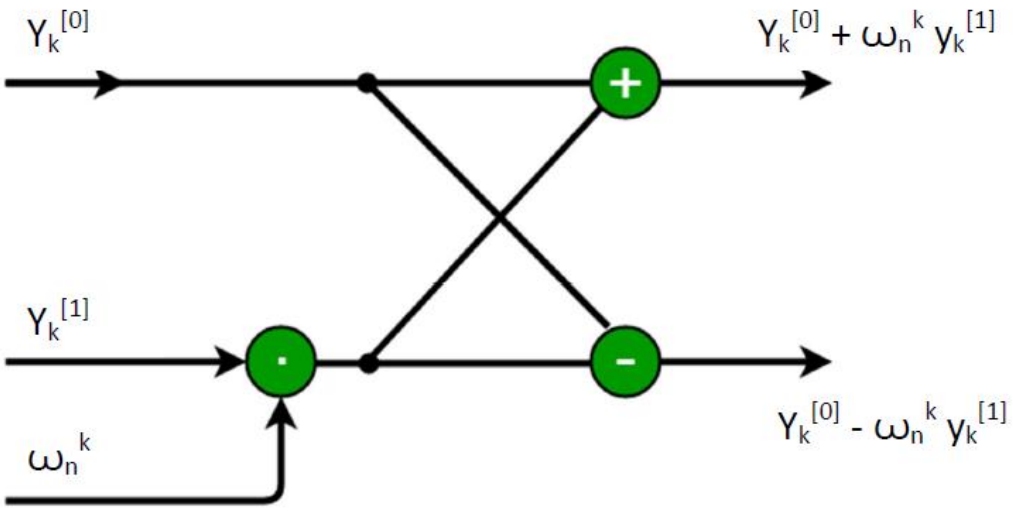
$$k \in [0, \dots, \frac{N}{2} - 1]$$

Iz poslednjih jednačina možemo primetiti rekurzivnu prirodu Furijeove transformacije i konačan izgled „Radix 2“ algoritma:

$$F[k] = F_p[k] + W_N^k F_n[k]$$

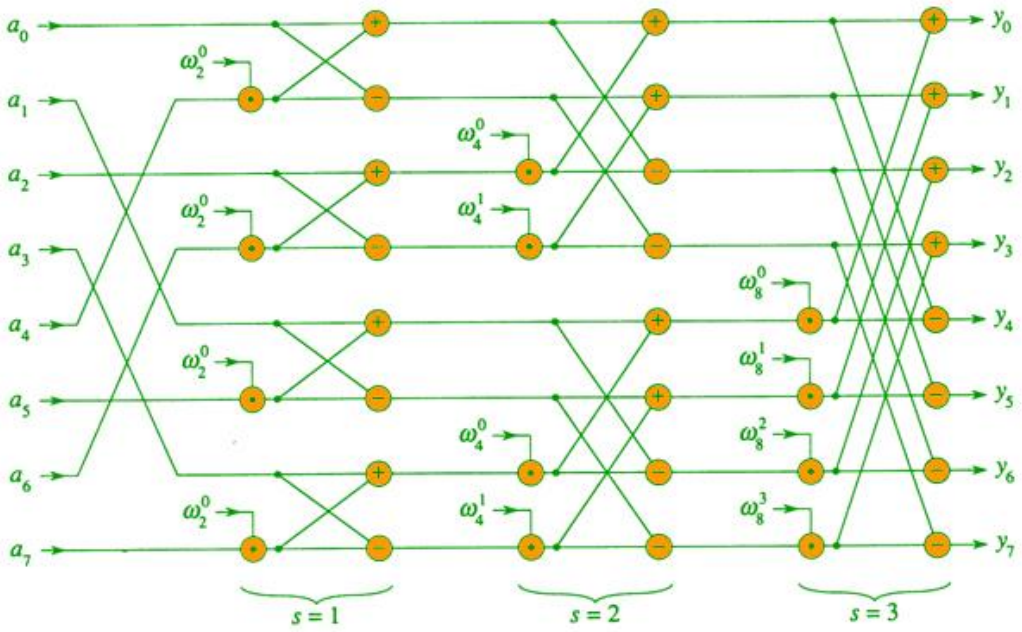
$$F[k + N/2] = F_p[k] - W_N^k F_n[k]$$

One nam omogućavaju da značajno uštedimo broj upotrebljenih operacija. Operacije računanja vrednosti F[k] i F[k+N/2], odnosno množenje neparnog člana sa rotacionim faktorom, sabiranje i oduzimanje sa parnim članom se naziva leptir (eng. „butterfly“) operacija, slika 3.



Slika 3 Leptir operacija

Na slici 4 je prikazan postupak dobijanja Furijeove transformacije niza sa osam članova pomoću „Radix 2“ algoritma.



Slika 4 Primer brze Furijeove transformacije za signal sa osam elemenata

2.6 Odnos konvolucije i Furijeove transformacije

Neka su *f* i *g* dva niza veličine *N* i *M*, redom, i neka je $P \geq N \geq M$ moduo kružne konvolucije. Tada formula koja spaja kružnu konvoluciju sa diskretnom Furijeovom transformacijom glasi:

$$(f *_P g)[k] = \mathcal{F}^{-1}\{\mathcal{F}\{f_P\} \cdot \mathcal{F}\{g_P\}\}[k]$$

Formula se može dokazati računanjem Furijeove transformacije kružne konvolucije signala f i g.

$$\mathcal{F}\{(f *_P g)\}[k] = \sum_{i=0}^{P-1} \left((f *_P g)[i] W_P^{ki}\right) = \sum_{i=0}^{P-1} \sum_{j=0}^{P-1} f_P[j] \cdot g_P[i-j] W_P^{ki}$$

$$= \sum_{j=0}^{P-1} f_P[j] W_P^{kj} \sum_{i=0}^{P-1} g_P[i-j] W_P^{k(i-j)}$$

$$= \sum_{j=0}^{P-1} f_P[j] W_P^{kj} \sum_{i=0}^{P-1} g_P[i] W_P^{k(i)},$$

$$(zbog\ periodičnosti\ g_P)$$

$$= \mathcal{F}\{f_P\}[k] \cdot \mathcal{F}\{g_P\}[k]$$

Povezivanjem date formule sa odnosom između potpune linearne i kružne konvolucije, dobijemo formulu za računanje potpune linearne konvolucije korišćenjem Furijeovih transformacija:

$$h[k] = \sum_{i=\max(0,k-M+1)}^{\min(N-1,k)} f[i] \cdot g[k-i] = (f *_P g)[k] = \mathcal{F}^{-1}\{\mathcal{F}\{f_P\}\mathcal{F}\{g_P\}\}[k],$$

$$\forall k \in [0, N+M-2]$$

Glava 3

Projektovanje aplikacije na sistemskom nivou

3.1 ESL Metodologija

3.2 Specifikacija i izvorni kod šahovskog programa

3.3 Analiza pre particionisanja

3.4 Particionisanje

3.5 Implementacija SystemC modela posle particionisanja

3.6 Implementacija Furijeove transformacije u dve dimenzije u SystemC-u

Glava 4

Projektovanje hardverskog IP bloka

4.1 RT metodologija

4.2 Diskusija mogućih implementacija

4.3 Implementacija leptir (butterfly) bloka

4.3.1 Definisanje interfejsa

- Ulazni interfejs

  topRE_i – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja gornji realni ulaz unutar leptir bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

  topIM_i – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja gornji imaginarni ulaz unutar leptir bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

  botRE_i – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja donji realni ulaz unutar leptir bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

  botIM_i – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja gornji realni ulaz unutar leptir bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

  k – tipa STD_LOGIC_VECTOR (log2c(FFT_SIZE/2)-1 downto 0) – označava trenutnu vrednost rotacionog faktora W_{size}^{k}.

  size – tipa STD_LOGIC_VECTOR (log2c(log2c(FFT_SIZE/2)-1 downto 0)) – predstavlja veličinu niza za koji se trenutno računa Furijeova transformacija. Zajedno sa veličinom k opisuje trenutnu vrednost rotacionog faktora W_{size}^{k}.

- Izlazni interfejs

  topRE_o – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja gornji realni izlaz unutar leptir bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

  topIM_o – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja gornji imaginarni izlaz unutar leptir bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

botRE_o – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja donji realni izlaz unutar leptir bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

botIM_o – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja gornji realni izlaz unutar leptir bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

- Komandni interfejs

  start – tipa STD_LOGIC – kontroliše početak rada leptir bloka

- Statusni interfejs

  ready – tipa STD_LOGIC - ukazuje da li je leptir blok trenutno aktivan

4.3.2 Projektovanje *Controlpath* modula

4.3.3 Projektovanje *Datapath* modula

4.4 Implementacija bloka za jednodimenzionu Furijeovu transformaciju

4.4.1 Interfejs

- Ulazni interfejs

  data_i_addr_o – tipa STD_LOGIC_VECTOR (ld(FFT_SIZE)-1 downto 0) - predstavlja adresu člana niza dužine *size* koja se trenutno učitava.

  dataRE_i – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja realni ulaz unutar *fft* bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

  dataIM_i – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja imaginarni ulaz unutar *fft* bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

  data_rd_o – tipa STD_LOGIC – označava da li *fft* blok zahteva učitavanje narednog člana niza.

  data_rd_i – tipa STD_LOGIC – označava da li su vrednosti dataRE_i i dataIM_i stabilne i spremne da budu učitane u unutrašnju memoriju.

  log2s – tipa STD_LOGIC_VECTOR (ld(ld(FFT_SIZE))-1 downto 0), predstavlja celobrojnu vrednost logaritma veličine niza koji se obrađuje zaokruženu na gore.

  size – tipa STD_LOGIC VECTOR (ld(FFT_SIZE)-1 downto 0), predstavlja veličinu niza koji se obrađuje.

- Izlazni interfejs

data_o_addr_o – tipa STD_LOGIC_VECTOR (ld (FFT_SIZE)-1 downto 0) - predstavlja adresu člana niza dužine *size* koja se trenutno učitava.

dataRE_o – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja realni izlaz iz *fft* bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

dataIM_o – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja imaginarni izlaz iz *fft* bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

data_rd_o – tipa STD_LOGIC – označava da li *fft* blok zahteva upis člana niza u spoljašnju memoriju i da li su vrednosti dataRE_o i dataIM_o stabilne na izlazu.

data_rd_i – tipa STD_LOGIC – označava da li su vrednosti dataRE_i i dataIM_i učitane u spoljašnju matričnu memoriju.

- Komandni interfejs

  start – tipa STD_LOGIC – kontroliše početak rada *fft* bloka

- Statusni interfejs

  ready – tipa STD_LOGIC - ukazuje da li je *fft* blok trenutno aktivan

4.4.2 Projektovanje *Controlpath* modula

4.4.3 Projektovanje *Datapath* modula

4.5 Implementacija bloka za Furijeovu transformaciju u dve dimenzije

4.5.1 Definisanje interfejsa

- Ulazni interfejs

  data_i_addr_o – tipa STD_LOGIC_VECTOR (ld(FFT_SIZE)-1 downto 0) - predstavlja adresu člana niza dužine *size* koja se trenutno učitava.

  dataRE_i – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja realni ulaz unutar *fft2* bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

  dataIM_i – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja imaginarni ulaz unutar *fft2* bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

  data_rd_o – tipa STD_LOGIC – označava da li *fft2* blok zahteva učitavanje narednog člana niza.

data_rd_i – tipa STD_LOGIC – označava da li su vrednosti dataRE_i i dataIM_i stabilne i spremne da budu učitane u spoljašnju memoriju matrice.

log2w – tipa STD_LOGIC_VECTOR (ld(ld(FFT_SIZE))-1 downto 0), predstavlja celobrojnu vrednost logaritma širine matrice koja se obrađuje zaokruženu na gore.

log2h – tipa STD_LOGIC_VECTOR (ld(ld(FFT_SIZE))-1 downto 0), predstavlja celobrojnu vrednost logaritma visine matrice koja se obrađuje zaokruženu na gore.

width – tipa STD_LOGIC VECTOR (ld(FFT_SIZE)-1 downto 0), predstavlja širinu matrice koji se obrađuje.

height – tipa STD_LOGIC VECTOR (ld(FFT_SIZE)-1 downto 0), predstavlja visinu matrice koji se obrađuje.

- Izlazni interfejs

  data_o_addr_o – tipa STD_LOGIC_VECTOR (ld(FFT_SIZE)-1 downto 0) - predstavlja adresu člana niza dužine *size* koja se trenutno učitava.

  dataRE_o – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja realni izlaz iz *fft2* bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

  dataIM_o – tipa STD_LOGIC_VECTOR (WIDTH-1 downto 0) – predstavlja imaginarni izlaz iz *fft2* bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED_POINT_WIDTH.

  data_rd_o – tipa STD_LOGIC – označava da li *fft2* blok zahteva upis člana niza u spoljašnju memoriju i da li su vrednosti dataRE_o i dataIM_o stabilne na izlazu.

  data_rd_i – tipa STD_LOGIC – označava da li su vrednosti dataRE_i i dataIM_i učitane u spoljašnju memoriju matrice.

- Komandni interfejs

  start – tipa STD_LOGIC – kontroliše početak rada *fft2* bloka

- Statusni interfejs

  ready – tipa STD_LOGIC - ukazuje da li je *fft2* blok trenutno aktivan

4.5.2 Projektovanje *Controlpath* modula

4.5.3 Projektovanje *Datapath* modula

4.6 Integrisanje u sistem i merenje performansi

Glava 5
Funkcionalna Verifikacija projektovanog IP bloka

5.1 UVM metodologija i SystemVerilog

5.2 Projektovanje verifikacionog okruženja za dizajnirani IP blok

5.2.1 Projektovanje sekvenci i sekvencera

5.2.2 Projektovanje drajvera

5.2.3 Projektovanje monitora

5.2.4 Projektovanje agenta

5.2.5 Projektovanje skorborda

5.2.6 Projektovanje modula za skupljanje pokrivenosti

5.2.7 Projektovanje okruženja

5.2.8 Projektovanje top modula i povezivanje sa IP modulom

5.3 Testovi i skupljanje pokrivenosti

Glava 6

Linuks drajver

Glava 7
Zaključak

Dodatak A

Kodovi

[1] Github repozitorijum za SystemC

[2] Github repozitorijum za VHDL

[3] Github repozitorijum za SystemVerilog

[4] Github repozitorijum za Linuks drajver

Dodatak B

Listing koda

Literatura

[1] Efficient convolution using the Fast Fourier Transform,
Jeremy Fix, 2011

[2] Rastislav Struharik, “Vežbe i Predavanja za predmet Projektovanje Složenih Digitalnih Sistema“,
http://www.elektronika.ftn.uns.ac.rs/index.php?option=com_content&task=category&sectionid=4&id=140&Itemid=139

[3] https://www.geeksforgeeks.org/converting-a-real-number-between-0-and-1-to-binary-string/, pogledano dana 13.9.2019.

[4] https://www.geeksforgeeks.org/iterative-fast-fourier-transformation-polynomial-multiplication, pogledano dana 19.9.2019.

[5] http://alwayslearn.com/DFT%20and%20FFT%20Tutorial/DFTandFFT_FFT_TwiddleFactor.html, pogledano dana 19.9.2019.