UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
KATEDRA ZA MIKRORAČUNARSKU ELEKTRONIKU

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

Diplomski rad

Osnovne akademske studije

kandidat
Vladimir Vincan, EE5-2015

mentor
dr Vuk Vranković, docent

Oktobar 2019

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:

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

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

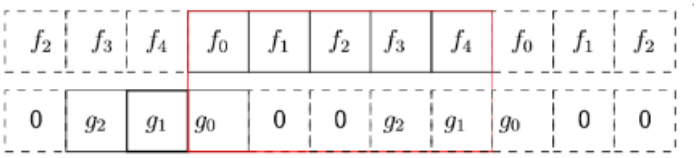
  Rezultujući signal h_∞ može imati maksimalno *N+M+1* nenultih vrednosti. U zavisnosti od toga da li konvolucioni proizvod uključuje sve nenulte članove dobijenog niza h_∞, razlikujemo tri tipa konvolucionog množenja:

  - **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:

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

$$= \sum_{i=0}^{\mathrm{P}_1-1} \sum_{j=0}^{P_2-1} f[i,j] \cdot g[k_1 - i, k_2 - j] = \left(g *_{P_1,P_2} f\right)[k_1, k_2],$$

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

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.

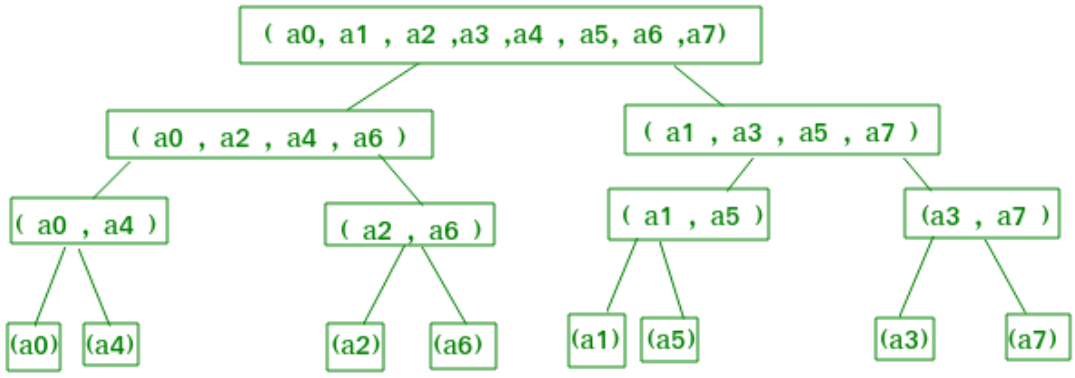
Inverzna Furijeova u dve dimenzije se može realizovati izvšavanjem dvodimenzione Furijeove transformacije nad konjugovanom matricom:

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

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]:

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

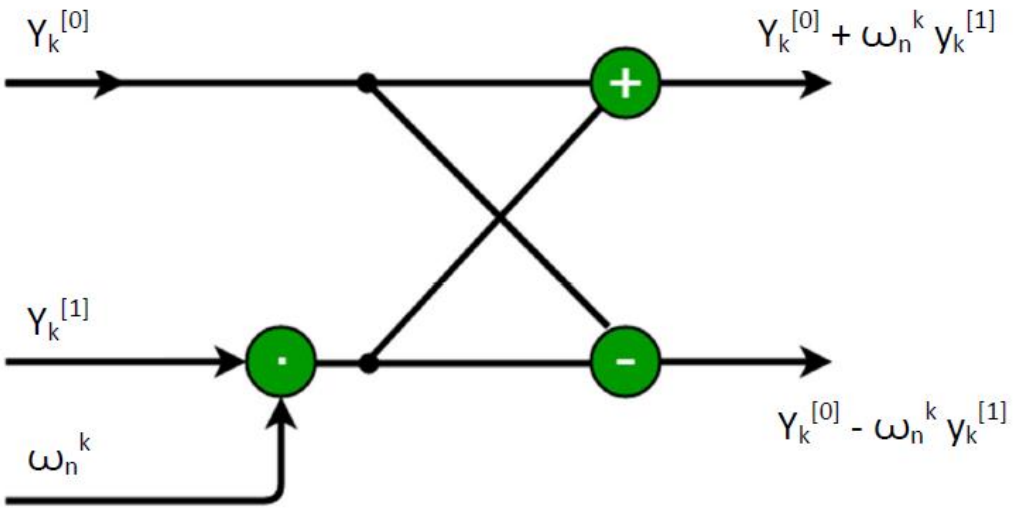
$$k \in [0, \dots, \frac{N}{2} - 1]$$

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

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

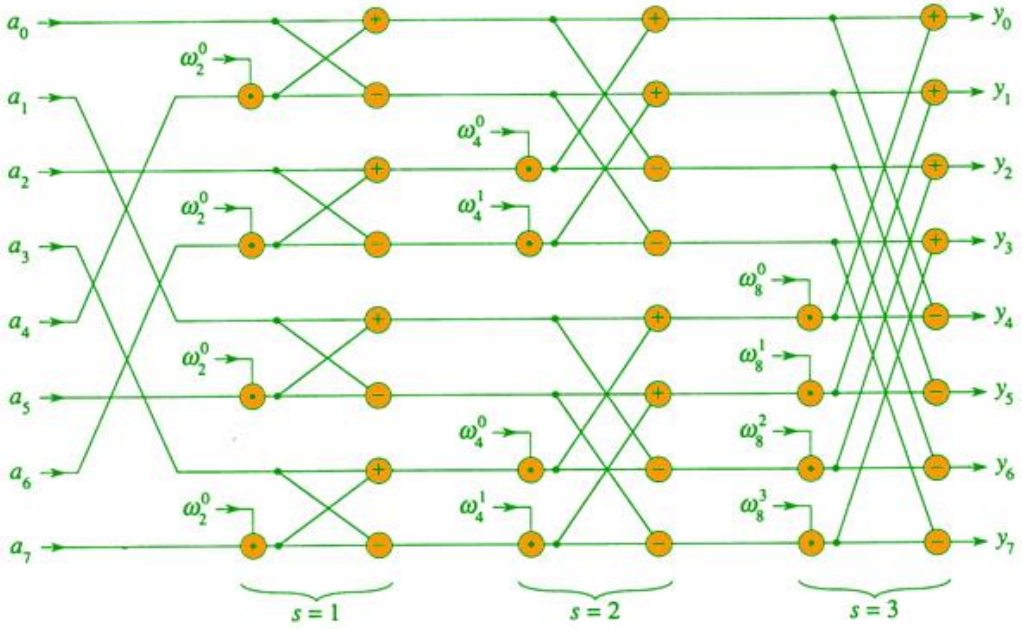
$$F[k + N/2] = F_p[k] - W_N^k F_n[k]$$

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

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 jezgra

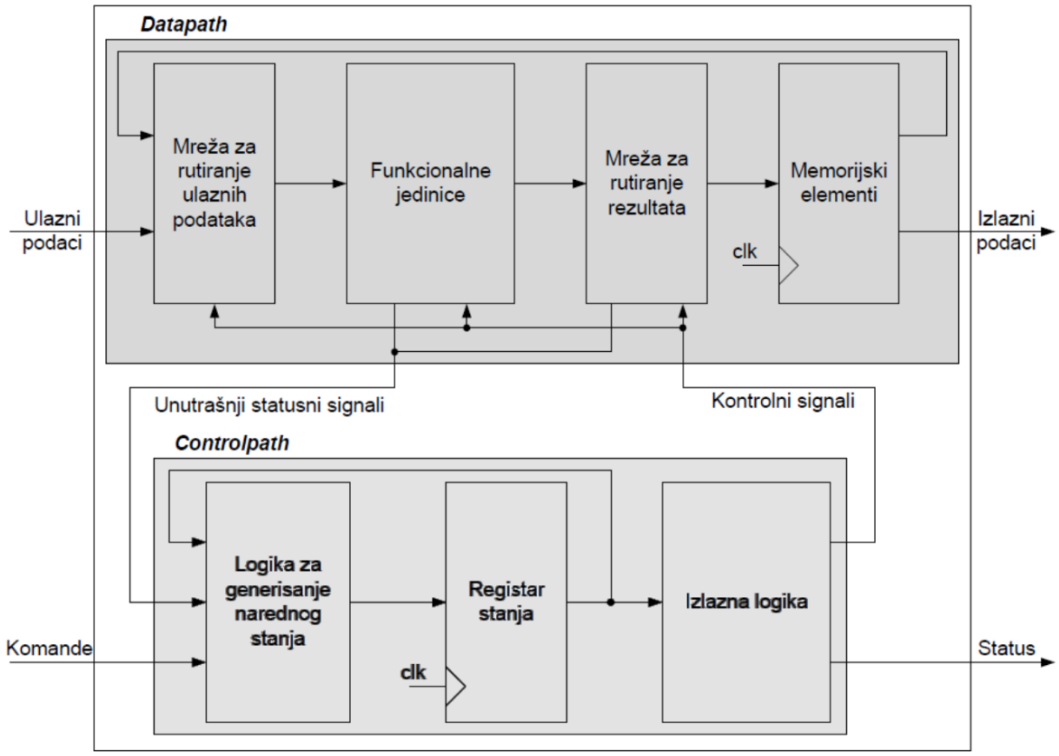
4.1 Uvod

4.1.1 RT metodologija

Prevođenje algoritma, odnosno detaljne sekvence akcija ili koraka neophodnih da se reši problem, u digitalno elektronsko kolo, se vrši pomoću RT metodologije. **RT** (eng. *Register Transfer*) **metodologija** predstavlja formalizovani postupak pomoću kojega se vrši mapiranje varijabli i sekvenci koraka nad varijablama u registre i manipulacije nad registrima i signalima (RT operacije). RT metodologija podrazumeva projektovanje dva podsistema:

- **Podsistem za obradu podataka** (eng. *Data path*) je modul koji sadrži sve hardverske resurse neophodne da bi se izvršili svi koraci definisani algoritmom i koji će biti izvršeni u datom IP jezgru, svih memorijskih resursa neophodnih da bi se skladištile varijable algoritma, kao i mreže za rutiranje koja na željeni način povezuje date hardverske komponente. Podsistem za obradu podataka se sastoji iz:
    - Mreže za rutiranje ulaznih podataka – dato kombinaciono digitalno elektronsko kolo rutira ulazne signale i vrednosti memorijskih elemenata sa odgovarajućim funkcionalnim jedinicama. Uglavnom je realizovano pomoću multipleksera.
    - Mreže za rutiranje rezultata – dato kombinaciono digitalno elektronsko kolo rutira izlaze funkcionalnih jedinica sa odgovarajućim memorijskim elementima. Uglavnom se realizuje pomoću multipleksera.
    - Funkcionalnih jedinica – sastoji se iz svih aritmetičkih, logičkih i relacionih operatora implementiranih u obliku kombinacionih mreža.
    - Memorijskih elemenata koji skladište sve podatke koji se koriste u toku izvršavanja algoritma. Najčešće su implementirani pomoću registara, mada mogu se koristiti registarske banke, jednopristupne i višepristupne memorije, FIFO (eng. *first in first out*) baferi i drugi elementi.
- **Upravljački podsistem** (eng. *Control path*) je modul čija je uloga da upravlja redosledom operacija koje se izvršavaju unutar podsistema za obradu podataka i signalima koji se u datom trenutku skladište u memoriji, u skladu sa koracima definisanim algoritmom. Upravljački podsistem se stoga najčešće realizuje kao konačni automat stanja. **Konačni automat stanja** (eng. *Finite State Machine, FSM*), predstavlja digitalno elektronsko kolo sa unutrašnjim stanjima. Tokom vremena

FSM menja stanje u kom se nalazi, u zavisnosti od trenutnog stanja i ulaznih signala. Sastoji se iz logike za generisanje narednog stanja, registra stanja i izlazne logike. Grafička reprezentacija konačnog automata stanja se može vršiti pomoću ASM (eng *Algorithmic State Machine*) dijagrama.



Slika 5 Šematski prikaz podsistema za obradu podataka i upravljačkog podsistema

Projektovanje digitalnog sistema koji treba da izvršava željeni algoritam, odnosno podsistema za obradu podataka i upravljačkog podsistema, pomoću RT metodologije, se sastoji iz pet koraka:

1. Eliminacija naredbi ponavljanja – sve naredbe ponavljanja unutar algoritma se moraju zameniti sa odgovarajućim *if-goto* naredbama.
2. Definisanje interfejsa digitalnog sistema koji se projektuje – analizom algoritma neophodno je utvrditi ulazne i izlazne promenljive (signale), kao i širine datih signala.
3. Projektovanje upravljačkog podsistema – na osnovu algoritma je neophodno kreirati odgovarajući ASM dijagram, kao i razmotriti potencijalne optimizacije.
4. Projektovanje podsistema za obradu podataka – sastoji se iz četiri koraka:
    a. Identifikacija svih RT operacija koje postoje unutar ASM dijagrama, i dimenzionisanje registara koji će biti pridruženi datim operacijama.
    b. Grupisanje RT operacija prema odredišnim registrima.

c. Za svaku grupu operacija, izvršavaju se tri koraka:
    i. Formira se odredišni registar.
    ii. Formiraju se kombinacione mreže koje implementiraju sve funkcionalne transformacije signala u datoj grupi.
    iii. Dodaju se multiplekserska i rutirajuća kola ispred odredišnog registra ukoliko je on asociran sa većim brojem RT operacija.
d. Dodaju se kombinacione mreže koje formiraju izlazne statusne signale.

5. Pisanje HDL modela – na osnovu ASM dijagrama dobijenog u trećem koraku može se napisati odgovarajući HDL model upravljačkog podsistema, a na osnovu blog dijagrama formiranog u četvrtom koraku može se napisati HDL model podsistema za obradu podataka. Konačno, objedinjavanjem datih modela, dobija se HDL model kompletnog digitalnog elektronskog kola koje implementira željeni algoritam.

4.1.2 IP jezgro

Hardverska IP (eng. *Intellectual Property*) **jezgra** su digitalni sistemi projektovani korišćenjem RT metodologije, koji enkapsuliraju neku jasno definisanu funkcionalnost. Uglavnom predstavljaju gradivne blokove većih i složenijih sistema na čipu. Da bi se olakšala ponovna upotrebljivost jezgara, za razmenu informacija se koriste standardni interfejsi kao što je AXI (eng. *Advanced eXtensible Interface*), PLB, OPB, LocalLink, i drugi. Na osnovu funkcionalnosti koju IP moduli obavljaju, dele se na:

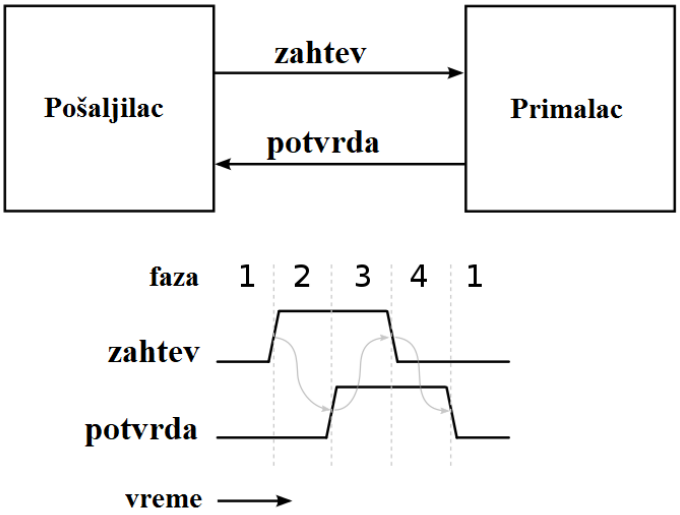
- Module za skladištenje podataka – u ovu grupu spadaju IP jezgra koja implementiraju mehanizme za skladištenje podataka, kao što su SRAM, DRAM, ROM, Flash, FIFO, i drugi.
- Module za prenos podataka - u ovu grupu spadaju IP jezgra koja implementiraju i međusobno prevode različite komunikacione protokole, kao što su AXI, PLB, PCIe, Ethernet, I^2C, SPI, i drugi.
- Module za obradu podataka - u ovu grupu spadaju IP jezgra koja vrše obradu podataka, odnosno algoritme kao što su filtriranje (FIR i IIR filtri), unitarne transformacije (FFT, DCT, DWT...), kompresiju, enkripciju i zaštitno kodovanje podataka, i druge često korišćene specifične algoritme.
- Procesorske module - u ovu grupu spadaju IP jezgra koja takođe vrše obradu podataka, i zapravo pripadaju prethodnoj grupi, ali su složeniji i moguće je implementirati proizvoljni algoritam na njima, pa se stoga uglavnom posmatraju odvojeno. Tipovi procesorskih modula su skalarni, superskalarni, VLIW, vektorski, višejezgarni i konfigurabilni procesori.

4.1.3 Četvorofazni hendšejk protokol

Četvorofazni hendšejk protokol (eng. *Four-Phase Handshake Protocol*) se koristi za sinhronu i asinhronu razmenu informacija između različitih modula. Protokol se sastoji iz pošaljioca i primaoca (eng. *talker* i *listener*) koji komuniciraju putem signala za zahtev i potvrdu (eng. *request* i *acknowledge*).

Redosled operacija se može podeliti na četiri koraka (otuda naziv četvorofazni):

1. Ukoliko su signali za zahtev i potvrdu na niskom logičkom nivou, pošaljilac može da uspostavi komunikaciju, odnosno aktivira signal za zahtev.
2. Kada primalac primeti da je signal za zahtev na visokom logičkom nivou, aktivira signal za potvrdu.
3. Kada pošaljilac primeti da je signal za potvrdu na visokom logičkom nivou, deaktivira signal za zahtev.
4. Kada primalac primeti da je signal za zahtev na niskom logičkom nivou, deaktivira signal za potvrdu.

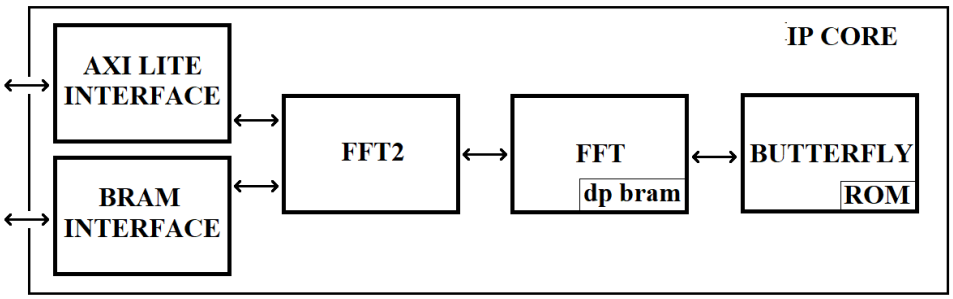


Slika 6 Šematski prikaz protokola četvorofaznog hendšejk protokola

4.2 Realizovana implementacija i diskusija mogućih implementacija

Algoritam za dvodimenzionu Furijeovu transformaciju matrica je ekvivalentan izvršavanju algoritma za jednodionu Furijeovu transformaciju prvo nad svim redovima, a potom nad svim kolonama matrice. IP jezgro komunicira sa dva spoljašnja BRAM-a (eng. *Block Random Access Memory*) u koje su smešteni realni i imaginarni deo matrice, kao i sa procesorom pomoću AXI Lite interfejsa, koji mu šalje veličinu matrice i zahtev za početak rada. Unutar FFT modula se nalazi unutrašnja dvopristupni BRAM (jedan pristup za čitanje i jedan pristup za pisanje) u koji se skladišti trenutni red ili kolona, nad kojim se vrši jednodimenziona Furijeova transformacija. Moduli FFT2, FFT i BUTTERFLY komuniciraju preko četvorofaznog hendšejk protokola. U ovakvoj konfiguraciji, modul FFT2 ima

jedinu ulogu kao konektor između BRAM-a i FFT modula, pri čemu mora da vodi računa o trenutnom redu ili koloni nad kojim se vrši obrada. FFT modul u toku izvršavanja više puta poziva BUTTERFLY modul, koji izvšava leptir operaciju nad dva kompleksna broja. Unutar BUTTERFLY modula se nalazi ROM (eng. *Read Only Memory*) iz kojih se čitaju vrednosti rotacionih faktora.



Slika 7 Šematski prikaz implementiranog IP jezgra za algoritam *fft2*

Postoji nekoliko mogućih razlika u implementaciji koji su razmatrani:

- Korišćenje AXI Full interfejsa umesto **BRAM interfejsa** za transport članova realnog i imaginarnog dela kompleksne matrice – izabran je BRAM isključivo zbog jednostavnosti implementacije. AXI Full interfejs bi mogao da obavlja daleko kompleksniju komunikaciju između komponenti koja za potrebe ovog projekta nisu potrebne.
- Korišćenje **unutrašnje memorije** za skladištenje međurezultata – umesto čuvanja samo jedne vrste ili kolone matrice, bilo je moguće uopšte ne implementirati unutrašnju memoriju ili skladištiti i realni i imaginarni deo matrice unutar IP jezgra (efektivno dve matrice iste veličine). U prvom slučaju, obavljanje komunikacije sa spoljašnjim BRAM-om bi značajno usporilo vreme izvršavanja. U drugom slučaju, zauzimanje dodatne memorije bi veoma uticalo na ukupnu veličinu matrice koja se može obrađivati, jer su resursi na ploči ograničeni. Izabrana je realizacija sistema u kojem se čuva vrednost samo jedne vrste odnosno kolone matrice, kao optimalan odnos između vremena izvršavanja i korišćenja hardverskih resursa.
- Korišćenje dvofaznog umesto **četvorofaznog hendšejk protokola** za *start* i *ready* signale između različitih modula i za pristup memoriji – iako je dvofazni hendšejk protokol jednostavniji za realizaciju i neznatno brži, izabran je četvorofazni protokol zbog jasnijeg praćenja rada algoritma i eventualnog otklanjanja grešaka u implementaciji celog algoritma. Dodatno, korišćenje protokola za pristup memoriji povećava sigurnost rada, jer dobijamo povratnu informaciju da se vrednost u memoriju zaista i upisala. U slučaju paralelizacije i korišćenje većeg broja istih modula (na primer više *fft* ili *butterfly* modula), može se

jednostavnije realizovati modul za arbitraciju i skedžulovanje zahteva za upis vrednosti u memoriju.

- Računanje **rotacionih faktora** umesto čitanja iz ROM-a – računanje rotacionih faktora podrazumeva implementaciju modula za računanje sinusa i kosinusa, ili skladištenje vrednosti sinusa i kosinusa za različite vrednosti ulaznog signala. U oba slučaja bi se vreme izvšavanja celog algoritma značajno povećalo (u prvom pogotovo), dok bi se u drugom slučaju svakako morale skladištiti vrednosti u memoriju, čime nikakva dobit ne bi bila postignuta.
- **Pajplajnovanje leptir operacije** – pajplajnovanjem izvršavanja date operacije povećavamo broj stanja kroz koja algoritam mora proći (samim tim i ukupan broj taktova), ali smanjujemo trajanje jednog takta, čime se efektivno postiže ubrzanje izvršavanja algoritma. U ovom radu su međusobno zavisne operacije maksimalno pajplajnovane, dok se nezavisne operacije izvršavaju u istom taktu.
- **Računanje logaritma veličine niza (matrice)** – u najjednostavnijoj hardverskoj realizaciji bi se zahtevalo korišćenje memorije velike veličine zarad jednog registra koji se neće menjati tokom izvršavanja algoritma. Izabrano je da taj logaritam računa u softveru i da se šalje u IP jezgro putem AXI Lite interfejsa.

4.3 Implementacija leptir modula (BUTTERFLY)

Kao što je već opisano, uloga leptir modula je da izvrši leptir operaciju:

$$F[k] = F_p[k] + W_{size}^{k} F_n[k]$$

Prethodna jednačina je predstavljena u kompleksnom obliku. Raščlanjivanjem na realni i imaginarni deo dobijemo sledeće formule:

$$topRE_o = topRE_i + botRE_i * wCOS[k] - botIM_i * wSIN[k]$$
$$topIM_o = topIM_i + botRE_i * wSIN[k] + botIM_i * wCOS[k]$$
$$botRE_o = topRE_i - botRE_i * wCOS[k] + botIM_i * wSIN[k]$$
$$botIM_o = topRE_i - botRE_i * wSIN[k] - botIM_i * wCOS[k]$$

Pri čemu važi:

$$wCOS[k] = cos(-2\pi \frac{k}{size})$$
$$wSIN[k] = sin(-2\pi \frac{k}{size})$$

Rotacioni faktori wCOS i wSIN su realizovani kao ROM memorija unutar leptir bloka, i potrebno ih je ukupno N/2, pri čemu N predstavlja broj elemenata niza, zbog osobine simetričnosti.

Radi povećavanja maksimalne frekvencije rada algoritma na FPGA ploči, pajplajnovano je izvršavanje leptir operacije, pa se vrednosti sabiranja i množenja

izvršavaju u različitim taktovima (prvo množenja, pa suma proizvoda, i potom dodavanje vrednosti $topRE_i$ i $topIM_i$).

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 $W_{2^{size}}^{k}$.

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

- 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

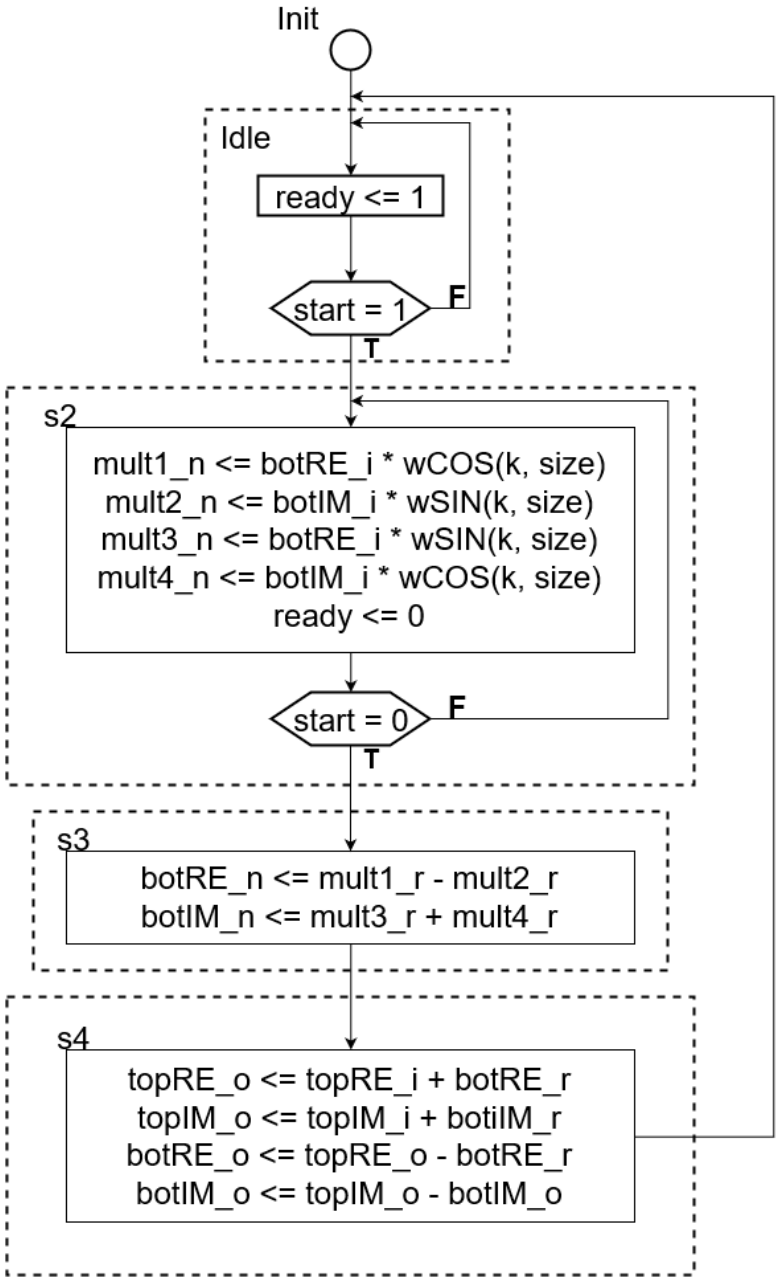
  start – tipa STD_LOGIC – kontroliše početak rada leptir bloka

- Statusni interfejs

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

4.3.2 Projektovanje upravljačkog modula

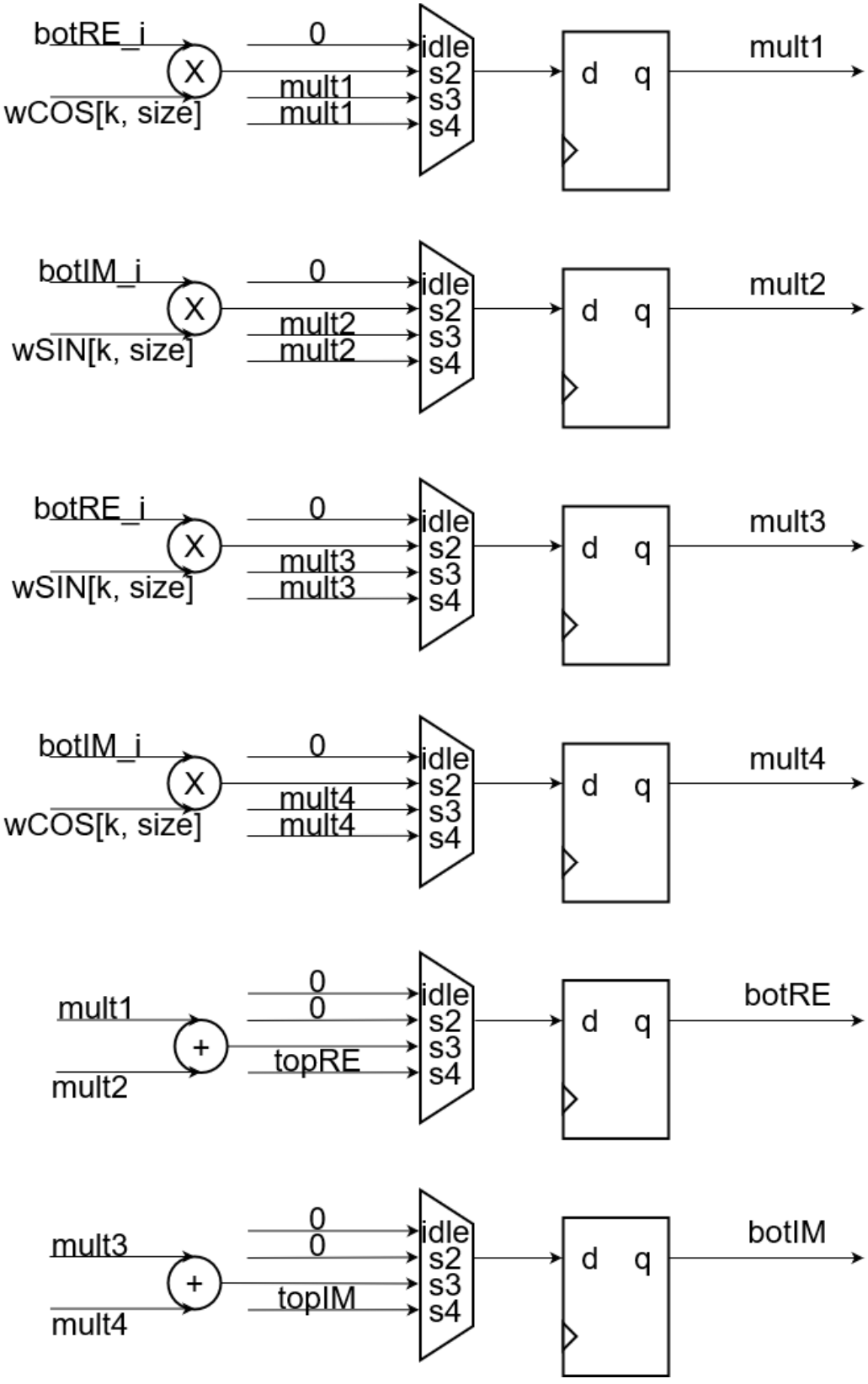
ASM dijagram ima četiri stanja. U prvom (*idle*) stanju se čeka na instrukciju za početak obrade ulaznih signala (*start*), signal *ready* je na visokom logičkom nivou. U drugom (*s2*) stanju se vrši množenje rotacionih faktora sa donjim ulaznim signalima leptir bloka, i spušta se *ready* signal na nisku logičku vrednost. U trećem (*s3*) se stanju vrši sabiranje pomnoženih signala. U četvrtom (*s4*) stanju se sabiraju vredosti gornjih ulaznih signala sa rezultujućim donjim signalima. Konačno, ponovo se ulazi u početno (*idle*) stanje i podiže se *ready* signal na visok logički nivo.



Slika 8 ASM dijagram leptir bloka

4.3.3 Projektovanje modula za obradu podataka

U datom sistemu postoji šest registara – četiri što skladište vrednosti množenja, i dva što skladište vrednosti zbira proizvoda. Množači se aktiviraju u drugom stanju, nakon što se detektuje da je aktivan start signal. Nakon što se izvrši množenje, odnosno u trećem stanju, vrši se sabiranje proizvoda i njegovo memorisanje u registre botRE i botIM. U petom stanju se vrši sabiranje vrednosti ulaznih signala topRE_i i topIM_i sa registrima botRE i botIM i prosleđuje na izlaz. Dakle, ovaj modul je realizovan kao Milijev automat i zahteva se konstantna vrednost ulaznih signala tokom trajanja izvršavanja datog modula.



Slika 9 Modul za obradu podataka unutar leptir bloka

4.4 Implementacija bloka za brzu Furijeovu transformaciju u jednoj dimenziji

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_wr_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_wr_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 upravljačkog modula

Blok za jednodimenzionu Furijeovu transformaciju se sastoji iz dvadeset stanja, koji se mogu podeliti na dve celine: deo za permutovanje članova ulaznog niza korišćenjem algoritma za obrtanje redosleda bita (eng. *bit reversal*) i učitavanje niza – prvih šest stanja, kao i deo za rekurzivnu podelu niza na parne i neparne članove i izvršavanje leptir operacija nad članovima – preostalih trinaest stanja (*idle* je dvadeseto stanje).

U **neaktivnom (*idle*) stanju** se kontinualno u registre skladište ulazne vrednosti dužine i logaritma dužine niza, dok god je *start* signal na niskom nivou. Kad *start* signal postane aktivan, učitane vrednosti dužine niza ostaju nepromenjene do kraja izvršavanja Furijeove transformacije.

Algoritam za **obrtanje redosleda bita** funkcioniše na sledeći način. Pretpostavimo da je širina adresnog porta 12 bita, a da za trenutni niz koji se obrađuje se koristi 9 bita. U tom slučaju će se poslednjih sedam bita obrnuti, dok će prva tri ostati nepromenjena, odnosno biće na niskoj logičnoj vrednosti (na primer: *000*1 1100 1100 →*000*0 0110 0111). Podaci sa obrnute adrese se učitavaju u memoriju.

```
for (int jj = 0; jj < size; ++jj){
  int reversed = 0;
  int temp = jj;
  for (int kk = 0; kk < log2s-1; ++kk){
    reversed = reversed << 1;
    reversed = reversed | (temp&1);
    temp = temp >> 1;
  }
  dataRE[jj] = dataRE_i[reversed];
  dataIM[jj] = dataIM_i[reversed];
}
```

Listing 1 Algoritam za obrtanje redosleda bita

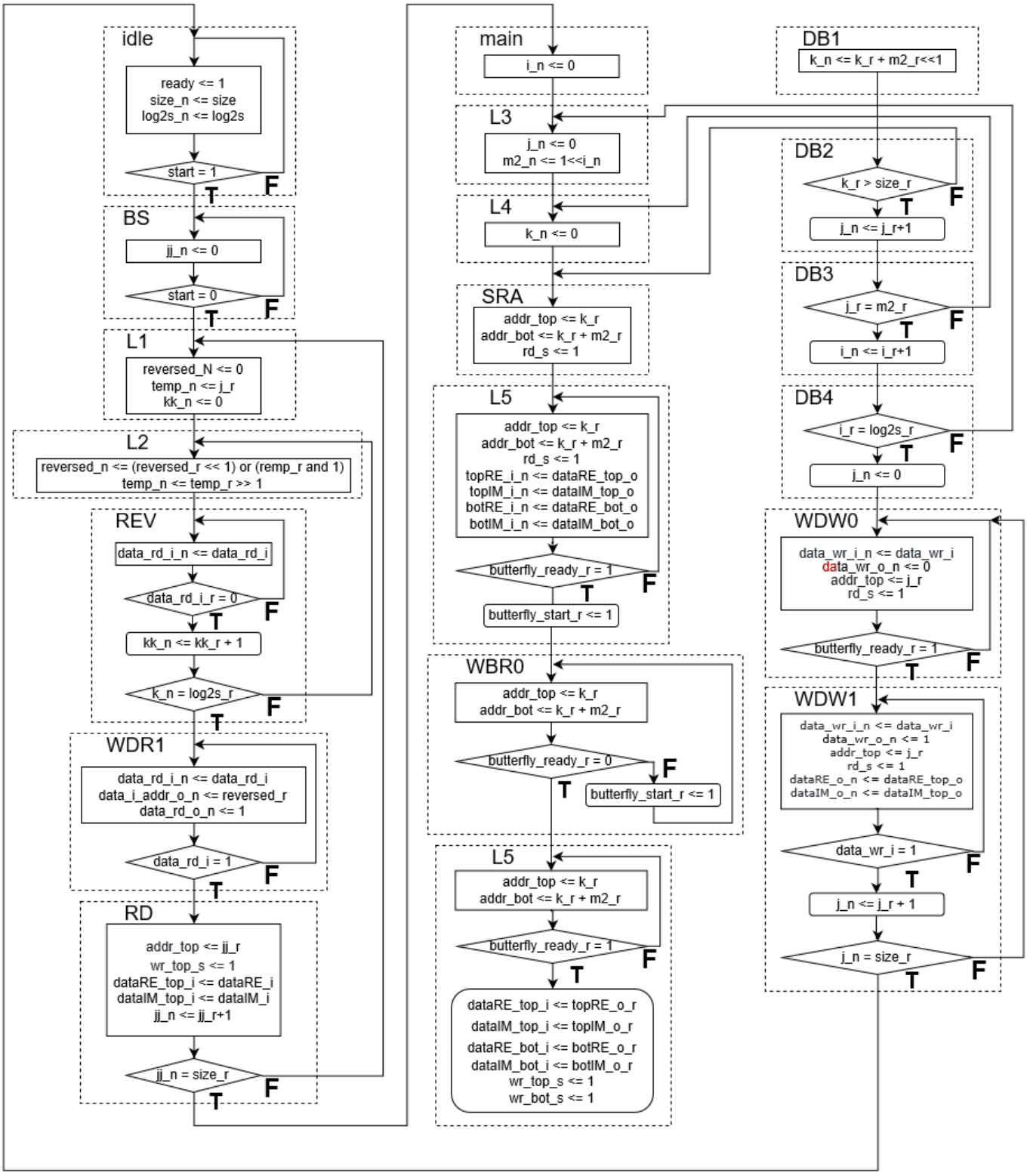
Upravljački modul algoritma za obrtanje redosleda bita se sastoji iz stanja BS (*bit reversal*), L1, L2, REV (*reverse*), WDR1 (*wait data read 1*) i RD (*read*). U stanju **BS** se inicijalizuje promenljiva *jj* na nulu, i čeka da se spusti *start* signal na nisku logičku vrednost. U stanju **L1** se inicijalizuju promenljive *reversed*, *temp* i k*k* (linije 2, 3 i 4 listinga), u stanju **L2** se ažuriraju vrednosti promenljivih *reversed* i *temp*, a u stanju **REV** se utvrđuje da li je cela adresa, odnosno promenljiva obrnula redosled bita. Ukoliko jeste, nastavlja se sa učitavanjem realnih i imaginarnih članova niza. U stanju **WDR1** se šalje zahtev za čitanje iz memorije sve dok se ne dobije potvrdan odgovor da su stigli podaci na ulazne portove. U stanju **RD** se upisuju vrednosti sa ulaza u unutrašnju memoriju, i ukoliko su svi podaci upisani nastavlja se sa glavnim delom Furijeove transformacije, inače se vraća na početak (stanje BS) i inkrementuje vrednost registra *jj*.

Iterativni algoritam za **rekurzivnu podelu niza** na parne i neparne članove i **pozivanje leptir operacije** nad tekućim elementima je prikazan u listingu 2 i izvršava operacije kao na slici 4. Napisan je u iterativnom obliku (dokazano je da svaki rekurzivni algoritam može biti napisan u iterativnom obliku, Čerč-Tjuringova teza) zato što RT metodologija ne omogućava mapiranje rekurzivnih funkcija na hardver.

```
for (int i=0; i < log2size-1; ++i){
  int m2 = 1 << i;
  for (int j=0; j < m2; ++j){
    for (int k=j; k < size; k+=m2<<1){
      double topRE = dataRE[k];
      double topIM = dataIM[k];
      double bottomRE = dataRE[k+m2];
      double bottomIM = dataIM[k+m2];
      butterfly(
        topRE,
        topIM,
        bottomRE,
        bottomIM,
        dataRE+k,
        dataIM+k,
        dataRE+k+m2,
        dataIM+k+m2,
        j << (log2size-i-2)
      );
    }
  }
}
```

Listing 2 Iterativni algoritam za rekurzivnu podelu niza i pozivanje leptir operacije

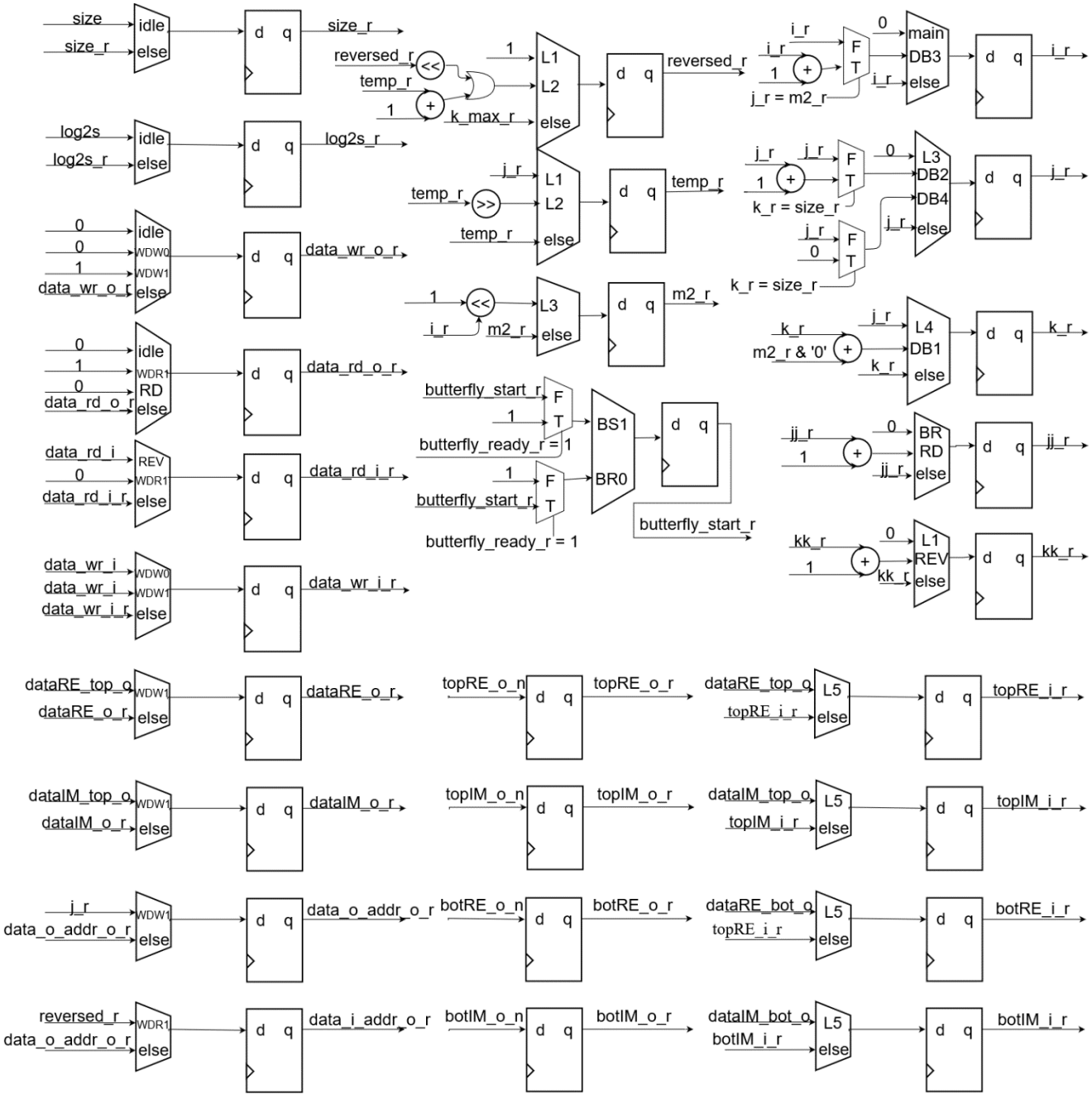
Upravljački modul algoritma za rekurzivnu podelu niza se sastoji iz stanja MAIN, L3, L4, SRA (*set read address*), L5, WBR0 (*wait butterfly ready 0*), WBR1 (*wait butterfly ready 1*), DB1, DB2, DB3, DB4 (*do butterfly*), WDW0 (*wait data write 0*) i WDW1 (*wait data write 1*). U stanju **MAIN** se inicijalizuje vrednost *i* registra. U stanju **L3** se inicijalizuju vrednosti *j* i *m2* registra. U stanju **L4** se inicijalizuje vrednost *k* registra. U stanju **SRA** se postavljaju adrese za učitavanje vrednosti iz memorije koji treba da se obrađuju u leptir operaciji, a potom se u stanju **L5** učitane vrednosti prosleđuju portovima leptir modula i drži se *start* signal na visokom logičkom nivou, sve dok leptir modul ne potvrdi da je primio naredbu. U stanju **WBR0** se čeka potvrda od leptir bloka da je primio *start* signal, a u **WBR1** se čeka potvrda da je leptir blok završio obradu podataka koji se potom upisuju u unutrašnju memoriju. U stanjima **DB1**, **DB2**, **DB3** i **DB4** se utvrđuje da li su registri *i*, *j* ili *k* došli do maksimalnih dozvoljenih vrednosti unutar petlji. Ukoliko je registar *i* došao do maksimalne vrednosti, podaci se upisuju u matricu pomoću stanja **WDW0** i **WDW1**.



Slika 10 ASM dijagram *fft* bloka

4.4.3 Projektovanje modula za obradu podataka

U *fft* bloku postoji dvadeset tri registra, ne računajući unutrašnju memoriju u kojoj se skladište sve vrednosti imaginarnog i realnog niza koji se obrađuje. Signali *dataRE_o_r*, *dataIM_o_r*, *data_o_addr_r*, *data_i_addr_r*, *data_wr_i* i *data_rd_o* su diretkno povezani sa odgovarajućim izlaznim portovima. Veličine registara direktno zavise od širine ulaznih signala u registre, i veličine mogu biti DATA_WIDTH, ld(FFT_SIZE), ili ld(ld(FFT_SIZE)).



Slika 11 Izgled modula za obradu podataka *fft* bloka

4.5 Implementacija bloka za brzu 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_wr_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_wr_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

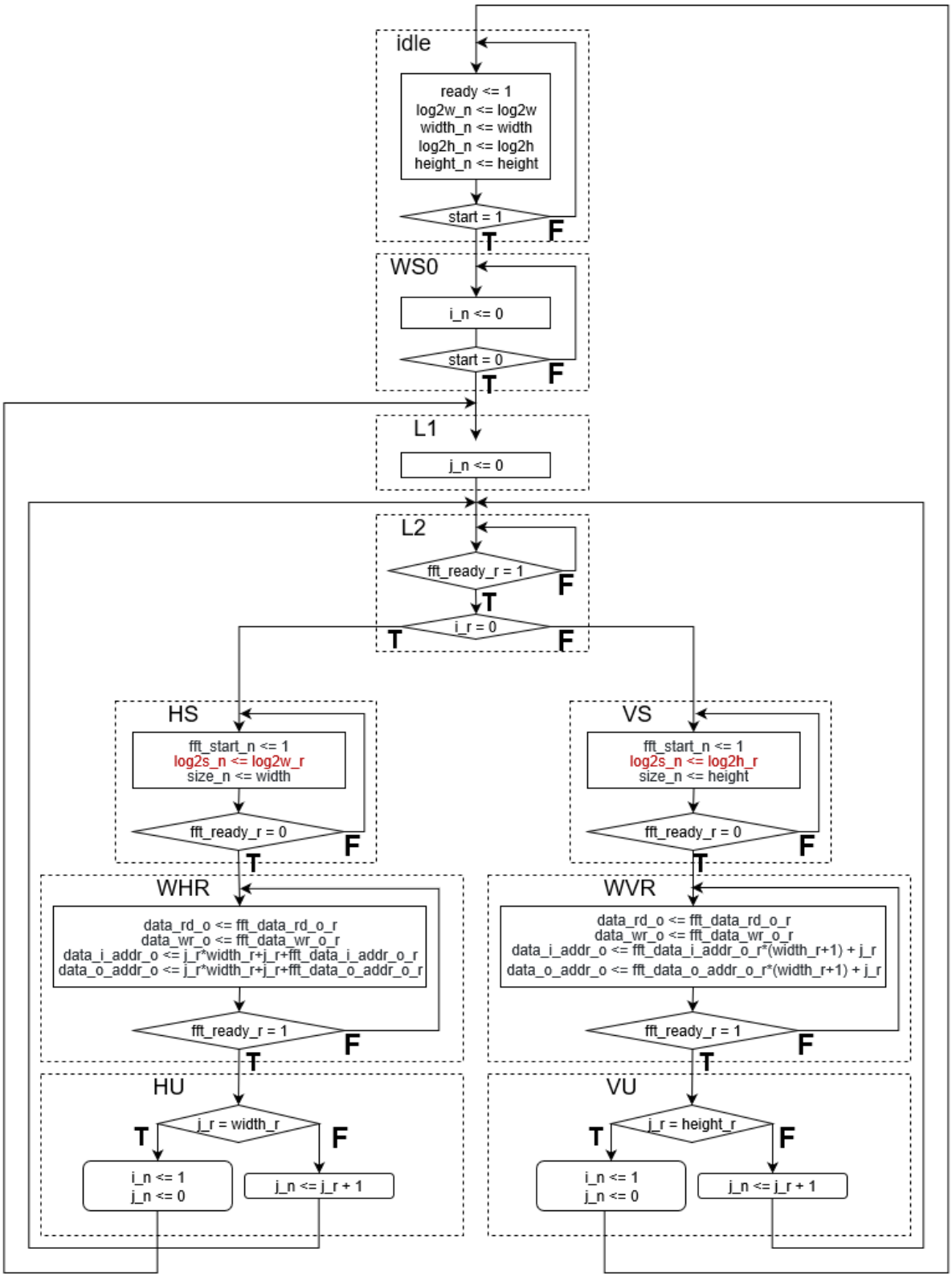
4.5.2 Projektovanje upravljačkog modula

Bloka za dvodimenzionu Furijeovu transformaciju ima ulogu da bude most između memorija u kojima su skladištene imaginarna i realna matrica, i bloka za jednodimenzionu Furijeovu transformaciju. Drugim rečima, *fft2* blok mora da pamti i vodi računa koji se red odnosno kolona u matricama trenutno obrađuje, kao i da omogući komunikaciju između memorija i *fft* bloka.

Upravljački modul se sastoji iz deset stanja – IDLE, WSO (*wait start 0*), L1, L2, HS (*horizontal start*), WHR (*wait horizontal ready*), HU (*horizontal update*), VS (*vertical start*), WVR (*wait vertical ready*), VU (*vertical ready*).

U **neaktivnom (*idle*) stanju**, slično kao i u slučaju sa jednodimenzionom Furijeovom transformacijom, kontinualno se u registre skladište ulazne vrednosti visine i širine matrica, kao i njihovih logaritama, dok god je *start* signal na niskom nivou. Kad *start* signal postane aktivan, učitane vrednosti dužine niza ostaju nepromenjene do kraja izvršavanja dvodimenzione Furijeove transformacije. U stanju **WS0** se čeka na spuštanje *start* signala, da bi se započelo obrađivanje podataka (u skladu sa četvorofaznim hendšejk protokolom), i podešava se signal registar *i_r* na nisku logičku vrednost, što označava da se trenutno obrađuju redovi matrica.

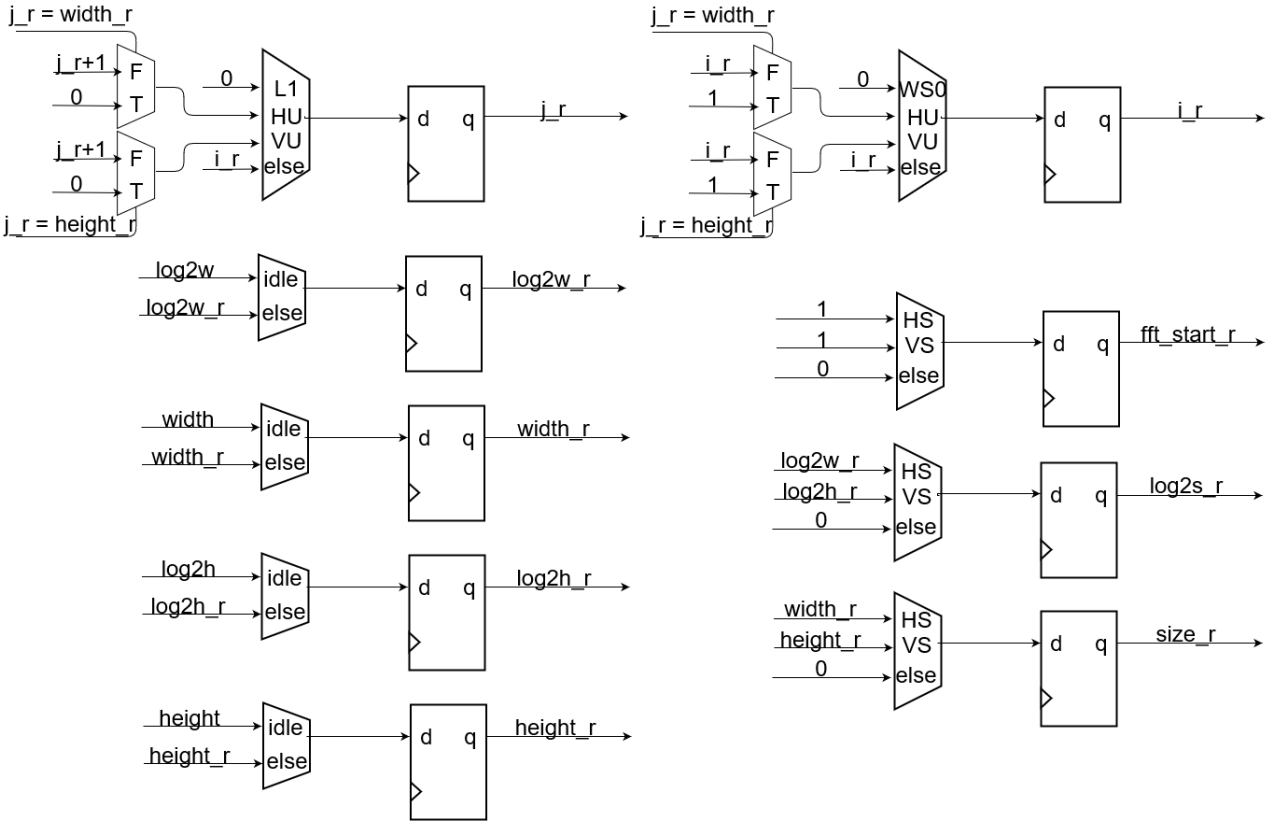
U stanju **L1** se inicijalizuje vrednost *j* registra, koji označava koji se trenutno red ili kolona obrađuje. U stanju **L2** se proverava da li je *fft* blok spreman da obrađuje sledeći niz. Ukoliko jeste, podaci o veličini i start signal se šalju u **HS** (**VS**) stanju. U stanju **WHR** (**WVR**) se vrši transformacija adrese sa jednodimenzionog niza na matricu, kao i obostrano prosleđivanje podataka. U stanja **HU** i **VU** se ulazi tek nakon što je obrađen čitav niz, i tu se proverava da li su svi nizovi (horizontalni ili veritikalni) obrađeni. Ukoliko jesu, nakon HU stanja se podiže signal *i_r* na logički visok nivo, kao indikator da treba da se krene sa obradom vertikalnih nizova. Nakon VU stanja se završava za obradom matrice i vraća u *idle* stanje.



Slika 12 ASM dijagram *fft2* modula

4.5.3 Projektovanje modula za obradu podataka

Modul za obradu podataka *fft2* bloka se sastoji iz devet registara. Četiri registra skladište ulazne podatke, odnosno širinu i visinu matrica, kao i njihove logartime. Veličine tih registara su ld(FFT_SIZE *FFT_SIZE) i ld(ld(FFT_SIZE *FFT_SIZE)). Dva registra služe da skladište veličinu niza koji će biti obrađivan, kao i njegov logaritam, i veličina tih registara je ld(FFT_SIZE) i ld(ld(FFT_SIZE)). Indikator *i_r* je jednobitni registar, dok je iterator *j_r* veličine ld(FFT_SIZE).



Slika 13 Modul za obradu podataka *fft2* bloka

4.6 Integrisanje u sistem i merenje performansi

Glava 5
Funkcionalna verifikacija projektovanog IP bloka

5.1 Uvod

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

Literatura

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

[2] Rastislav Struharik, vežbe i predavanja za predmeta 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.