



UNIVERZITET U NOVOM SADU  
**FAKULTET TEHNIČKIH NAUKA**  
KATEDRA ZA ELEKTRONIKU

---



# **Hardversko-softverska realizacija akceleratora za konvoluciju matrica**

Diplomski rad

Osnovne akademske studije


kandidat

Vladimir Vincan, EE5-2015

mentor

dr Vuk Vranković, docent

Novi Sad, Oktobar 2019

	UNIVERZITET U NOVOM SADU • FAKULTET TEHNIČKIH NAUKA 21000 NOVI SAD , Trg Dositeja Obradovića 6	Broj:
	<b>ZADATAK ZA ZAVRŠNI (BACHELOR) RAD</b>	Datum:

(Podatke unosi predmetni nastavnik - mentor)

Vrsta studija:	Osnovne akademske studije
Studijski program:	Energetika, elektronika i telekomunikacije
Rukovodilac studijskog programa:	Dr Milan Sečujski, vanredni profesor

Student:	Vladimir Vincan	Broj indeksa:	EE5/2015
Oblast:	Projektovanje složenih digitalnih sistema		
Mentor:	dr Vuk Vranković, docent		

NA OSNOVU PODNETE PRIJAVE, PRILOŽENE DOKUMENTACIJE I ODREDBI STATUTA FAKULTETA IZDAJE SE ZADATAK ZA ZAVRŠNI (Bachelor) RAD, SA SLEDEĆIM ELEMENTIMA:

- problem – tema rada;
- način rešavanja problema i način praktične provere rezultata rada, ako je takva provera neophodna;
- literatura

### NASLOV ZAVRŠNOG (BACHELOR) RADA:

Hardversko-sofverska implementacija akceleratora za konvoluciju matrica

### TEKST ZADATKA:

1. Izvršiti projektovanje i analizu na sistemskom nivou hardversko-sofverskog rešenja konvolucije matrica zasnovane na brzim Furijeovim transformacijama.
2. Projektovati digitalni sistem za brzu Furijeovu transformaciju matrica.
3. Verifikovati projektovani digitalni sistem koristeći UVM metodologiju.
4. Razvoj Linuks drajvera za komunikaciju sa projektovanim sistemom i pisanje korisničke aplikacije.

Rukovodilac studijskog programa:	Mentor rada:
dr Milan Sečujski	dr Vuk Vranjković

Primerak za: O- Studenta; O- Studentsku službu fakulteta



UNIVERZITET U NOVOM SADU • FAKULTET TEHNIČKIH NAUKA  
21000 Novi Sad, Trg Dositeja Obradovića 6

## KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj, <b>RBR:</b>		
Identifikacioni broj, <b>IBR:</b>		
Tip dokumentacije, <b>TD:</b>		Monografska dokumentacija
Tip zapisa, <b>TZ:</b>		Tekstualni štampani materijal
Vrsta rada, <b>VR:</b>		Diplomski rad
Autor, <b>AU:</b>		Vladimir Vincan
Mentor, <b>MN:</b>		dr Vuk Vranković, docent
Naslov rada, <b>NR:</b>		Hardversko-softverska realizacija akceleratora za konvoluciju matrica
Jezik publikacije, <b>JP:</b>		Srpski
Jezik izvoda, <b>Ji:</b>		Srpski
Zemlja publikovanja, <b>ZP:</b>		Srbija
Uže geografsko područje, <b>UGP:</b>		Vojvodina
Godina, <b>GO:</b>		2019
Izdavač, <b>IZ:</b>		Autorski reprint
Mesto i adresa, <b>MA:</b>		21000 Novi Sad, Trg Dositeja Obradovića 6
Fizički opis rada, <b>FO:</b> (poglavlja/strana/citata/tabela/slika/grafika/priloga)		(10/71/23/7/39/0/0)
Naučna oblast, <b>NO:</b>		Elektronika
Naučna disciplina, <b>ND:</b>		Embeded Sistemi
Predmetna odrednica/Ključne reči, <b>PO:</b>		FPGA, Hardverski akcelerator, Furijeova transformacija, Konvolucija
<b>UDK</b>		
Čuva se, <b>ČU:</b>		Biblioteka Fakulteta Tehničkih Nauka 21000 Novi Sad, Trg Dositeja Obradovića 6
Važna napomena, <b>VN:</b>		Nema
Izvod, <b>IZ:</b>		U ovom diplomskom radu izvršeno je projektovanje hardversko-softverskog rešenja akceleratora za konvoluciju matrica korišćenjem dvodimenzionalnih brzih Furijeovih transformacija. Akcelerator je implementiran na FPGA čipu.
Datum prihvatanja teme, <b>DP:</b>		01.09.2019.
Datum odbrane, <b>DO:</b>		21.10.2019.
Članovi komisije, <b>KO:</b>	Predsednik:	dr. Rastislav Struharik, vanredni profesor
	Član:	dr. Predrag Teodorović, docent
	Član, mentor	dr. Vuk Vranković, docent
		<b>Potpis mentora</b>



UNIVERSITY OF NOVI SAD • FACULTY OF TECHNICAL SCIENCES  
21000 Novi Sad, Trg Dositeja Obradovića 6

## KEY WORDS DOCUMENTATION

Accession number, <b>ANO:</b>		
Identification number, <b>INO:</b>		
Document type, <b>DT:</b>	Monographic publication	
Type of record, <b>T3:</b>	Textual material, printed	
Contents code, <b>CC:</b>	Graduation thesis	
Author, <b>AU:</b>	Vladimir Vincan	
Mentor, <b>MN:</b>	Vuk Vranković, PhD	
Title, <b>TI:</b>	Hardware-software realization of a matrix convolution accelerator	
Language of text, <b>LT:</b>	Serbian	
Language of abstract, <b>LA:</b>	Serbian	
Country of publication, <b>CP:</b>	Serbia	
Locality of publication, <b>LP:</b>	Vojvodina	
Publication year, <b>PY:</b>	2019	
Publisher, <b>PB:</b>	Author's reprint	
Publication place, <b>PP:</b>	21000 Novi Sad, Trg Dositeja Obradovića 6	
Physical description, <b>PD:</b> (chapters/ pages/ ref. / tables/ pictures/ graphs/ appendixes)	(10/71/23/7/39/0/0)	
Scientific field, <b>SF:</b>	Electrical engineering	
Scientific discipline, <b>SD:</b>	Embedded Systems	
Subject/ Key words, <b>S/KW:</b>	FPGA, Hardware accelerator, Fourier transform, Convolution	
<b>UC</b>		
Holding data, <b>HD:</b>	Library of Faculty of Technical Sciences 21000 Novi Sad, Trg Dositeja Obradovića 6	
Note, <b>N:</b>	None	
Abstract, <b>AB:</b>	This graduation thesis describes the hardware-software codesign solution for a matrix convolution accelerator. The convolution algorithm is realized with two-dimensional fast Fourier transformations. The accelerator is implemented on a FPGA chip.	
Accepted by the Scientific Board on, <b>ASB:</b>	01.09. 2019.	
Defended on, <b>DE:</b>	25.10.2019.	
Defended board, <b>DB:</b>	President:	Rastislav Struharik, PhD, full professor
	Member:	Predrag Teodorović, PhD, assistant professor
	Member, Mentor	Vuk Vranković, PhD, assistant professor
		<b>Mentor's signature</b>

# Sadržaj

Glava 1 Uvod .....	1
Glava 2 Konvolucija matrica .....	2
2.1    Konvolucija u jednoj dimenziji.....	2
2.2    Konvolucija u dve dimenzije .....	3
2.3    Diskretna Furijeova transformacija u jednoj dimenziji.....	4
2.4    Diskretna Furijeova transformacija u dve dimenzije .....	5
2.5    Brza Furijeova transformacija.....	6
2.6    Odnos konvolucije i Furijeove transformacije.....	8
Glava 3 Projektovanje aplikacije na sistemskom nivou.....	10
3.1    ESL Metodologija.....	10
3.2    Specifikacija.....	13
3.3    Analiza pre particionisanja.....	13
3.4    Particionisanje.....	2
3.5    Implementacija SystemC modela posle particionisanja.....	2
3.6    Implementacija Furijeove transformacije u dve dimenzije u SystemC-u ..	<b>Error!</b>
<b>Bookmark not defined.</b>	
Glava 4 Projektovanje hardverskog IP jezgra.....	3
4.1    Uvod.....	3
4.1.1    RT metodologija .....	3
4.1.2    IP jezgro .....	5
4.1.3    Četvorofazni hendšejk protokol .....	6
4.2    Realizovana implementacija i diskusija mogućih implementacija .....	6
4.3    Implementacija leptir modula (BUTTERFLY).....	9
4.3.1    Definisanje interfejsa .....	9
4.3.2    Projektovanje upravljačkog modula.....	10
4.3.3    Projektovanje modula za obradu podataka .....	11
4.4    Implementacija bloka za brzu Furijeovu transformaciju u jednoj dimenziji.....	12
4.4.1    Interfejs .....	12
4.4.2    Projektovanje upravljačkog modula.....	13
4.4.3    Projektovanje modula za obradu podataka .....	5
4.5    Implementacija bloka za brzu Furijeovu transformaciju u dve dimenzije .....	2
4.5.1    Definisanje interfejsa .....	2
4.5.2    Projektovanje upravljačkog modula.....	3
4.5.3    Projektovanje modula za obradu podataka .....	31

4.6	Integrisanje u sistem i merenje performansi .....	32
Glava 5 Funkcionalna verifikacija projektovanog IP bloka.....		33
5.1	Uvod.....	33
5.2	Projektovanje verifikacionog okruženja za dizajnirani IP blok .....	33
5.2.1	Projektovanje sekvenci i sekvencera.....	33
5.2.2	Projektovanje drajvera .....	33
5.2.3	Projektovanje monitora .....	33
5.2.4	Projektovanje agenta .....	33
5.2.5	Projektovanje skorborda .....	33
5.2.6	Projektovanje modula za skupljanje pokrivenosti.....	33
5.2.7	Projektovanje okruženja.....	33
5.2.8	Projektovanje top modula i povezivanje sa IP modulom .....	33
5.3	Testovi i skupljanje pokrivenosti .....	33
Glava 6 Linuks drajver.....		34
Glava 7 Zaključak .....		35
Dodatak A Kodovi .....		36
Literatura.....		37

## Slike

Slika 1 Izgled nizova $f$ i $g$ za $P=N=5$ i $M=3$ .....	3
Slika 2 Rekurzivno izvršavanje brze Furijeove transformacije.....	6
Slika 3 Leptir operacija .....	8
Slika 4 Primer brze Furijeove transformacije za signal sa osam elemenata .....	8
Slika 5 Šematski prikaz podsistema za obradu podataka i upravljačkog podsistema .....	4
Slika 6 Šematski prikaz protokola četvorofaznog hendšejk protokola .....	6
Slika 7 Šematski prikaz implementiranog IP jezgra za algoritam $fft2$ .....	7
Slika 8 ASM dijagram leptir bloka.....	11
Slika 9 Modul za obradu podataka unutar leptir bloka .....	12
Slika 10 ASM dijagram $fft$ bloka .....	4
Slika 11 Izgled modula za obradu podataka $fft$ bloka .....	5
Slika 12 ASM dijagram $fft2$ modula .....	30
Slika 13 Modul za obradu podataka $fft2$ bloka .....	31
Slika 14 Blok dijagram celokupnog sistema .....	32

## Listing koda

Listing 1 Algoritam za obrtanje redosleda bita .....	2
Listing 2 Iterativni algoritam za rekurzivnu podelu niza i pozivanje leptir operacije .....	3



# 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 projektovani 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:

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

Rezultujući signal  $h_{\infty}$  može imati maksimalno  $N+M+1$  nenultih vrednosti. U zavisnosti od toga da li konvolucionni proizvod uključuje sve nenulte članove dobijenog niza  $h_{\infty}$ , razlikujemo tri tipa konvolucionog množenja:

- **Potpuno množenje** – rezultujući niz  $h$  će uključivati sve elemente niza  $h_{\infty}$ , koji mogu imati nenultu vrednost. Niz  $h$  će posedovati  $N+M+1$  elemenata.
- **Množenje iste veličine** – rezultujući niz  $h$  će uključivati samo članove niza  $h_{\infty}$ , 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  $h_{\infty}$ , kod kojih su svi neprošireni 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:

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

- **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  $f(k) = f(k \bmod P)$  i  $g(k) = g(k \bmod P)$ . Na slici 1 se nalazi primer za  $P=N=5$  i  $M=3$ .

$f_2$	$f_3$	$f_4$	$f_0$	$f_1$	$f_2$	$f_3$	$f_4$	$f_0$	$f_1$	$f_2$
0	$g_2$	$g_1$	$g_0$	0	0	$g_2$	$g_1$	$g_0$	0	0

**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:

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

Pri čemu važi:

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

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

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:

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

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:

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

Pri čemu važi:

$$\begin{aligned} f_{\infty}[i, j] &= \begin{cases} f[i, j], & 0 \leq i \leq H_1, 0 \leq j \leq W_1, \\ 0, & \text{inače} \end{cases} \\ g_{\infty}[i, j] &= \begin{cases} g[i, j], & 0 \leq i \leq H_2, 0 \leq j \leq W_2, \\ 0, & \text{inače} \end{cases} \end{aligned}$$

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:

$$\begin{aligned} h_p[k_1, k_2] &\triangleq (f *_{p_1, p_2} g)[k_1, k_2] = \\ &= \sum_{i=0}^{P_1-1} \sum_{j=0}^{P_2-1} f[i, j] \cdot g[k_1 - i, k_2 - j] = (g *_{p_1, p_2} f)[k_1, k_2], \\ &\quad \forall k_1, k_2 \in \mathbb{Z} \end{aligned}$$

Pri čemu važi:

$$\begin{aligned} f_{p_H, p_W}[i, j] &= \sum_{p_1=-\infty}^{\infty} \sum_{p_2=-\infty}^{\infty} f_{\infty}[i + p_1 \cdot P_H, j + p_2 \cdot P_W] \\ &= f_{\infty}[i \bmod P_H, j \bmod P_W], \quad \forall i, j \in \mathbb{Z} \\ g_{p_H, p_W}[i, j] &= \sum_{p_1=-\infty}^{\infty} \sum_{p_2=-\infty}^{\infty} g_{\infty}[i + p_1 \cdot P_H, j + p_2 \cdot P_W] \\ &= g_{\infty}[i \bmod P_H, j \bmod P_W], \quad \forall i, j \in \mathbb{Z} \end{aligned}$$

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

### 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

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

- 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:

$$\begin{aligned} F[k, l] &= \mathcal{F}\{f\}[k, l] \triangleq \\ &\triangleq \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[n, m] e^{-\frac{2i\pi lm}{M}} e^{-\frac{2i\pi kn}{N}} \\ &= \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[n, m] W_M^{lm} W_N^{kn} \end{aligned}$$

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

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.

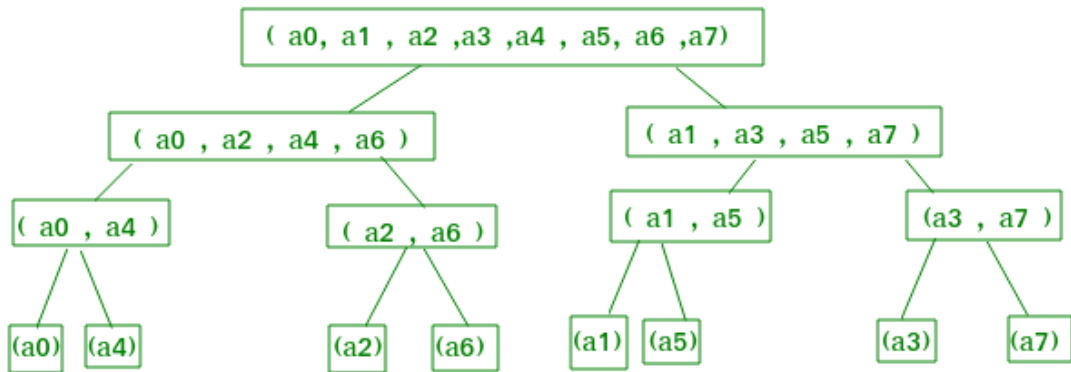
Inverzna Furijeova u dve dimenzije se može realizovati izvršavanjem dvodimenzione Furijeove transformacije nad konjugovanom matricom:

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

## 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, rekursivnim 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** Rekursivno izvršavanje brze Furijeove transformacije

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

$$\begin{aligned}
 F[k] &= \mathcal{F}\{f\}[k] = \sum_{n=0}^{N-1} f[n]W_N^{nk} = \\
 &= \sum_{n=0}^{\frac{N}{2}-1} f[2n]W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} f[2n+1]W_N^{(2n+1)k} = \\
 &= \sum_{n=0}^{\frac{N}{2}-1} f[2n]W_N^{2nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} f[2n+1]W_N^{2nk} \\
 & \quad k \in [0, \dots, N-1]
 \end{aligned}$$

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

$$\begin{aligned}
 F[k] &= \sum_{n=0}^{\frac{N}{2}-1} f[2n]W_{N/2}^{nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} f[2n+1]W_{N/2}^{nk} = F_p[k] + W_N^k F_n[k] \\
 & \quad k \in [0, \dots, \frac{N}{2}-1]
 \end{aligned}$$

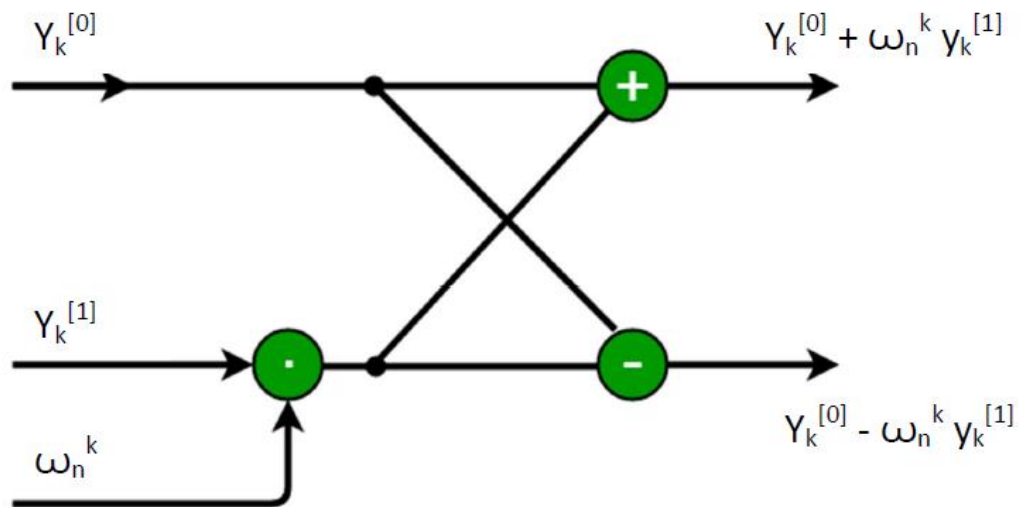
Na osnovu osobine kompleksno konjugovane simetričnosti, redundantnosti i periodičnosti, može se jednostavno, na osnovu već izračunatih vrednosti, dobiti vrednost člana  $F[k+N/2]$ :

$$\begin{aligned}
 F\left[k + \frac{N}{2}\right] &= F_p\left[k + \frac{N}{2}\right] + W_N^{k+\frac{N}{2}} F_n\left[k + \frac{N}{2}\right] = F_p[k] - W_N^k F_n[k] \\
 & \quad k \in [0, \dots, \frac{N}{2}-1]
 \end{aligned}$$

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

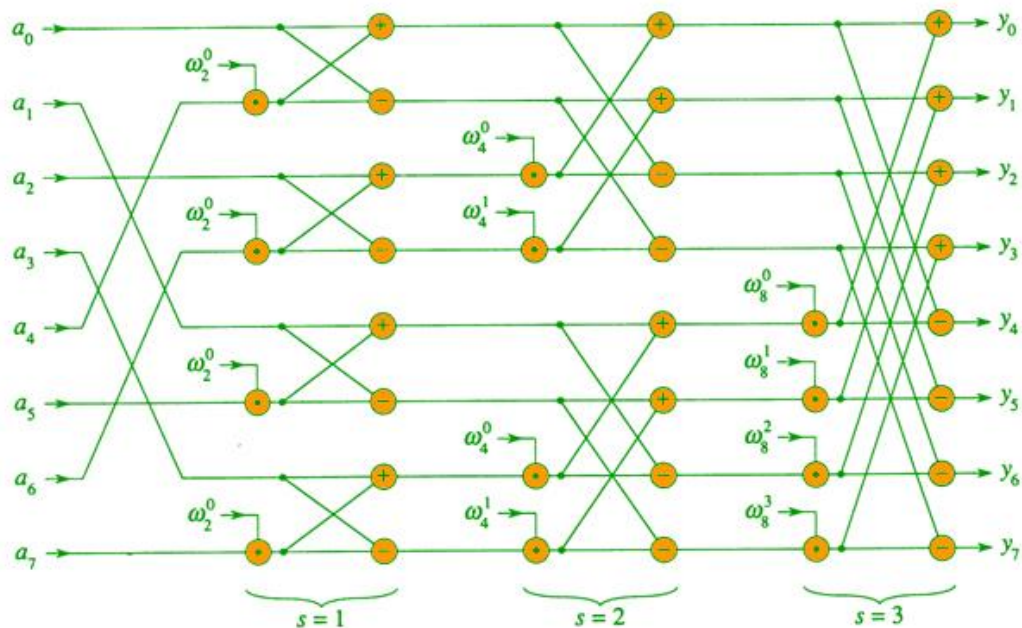
$$\begin{aligned}
 F[k] &= F_p[k] + W_N^k F_n[k] \\
 F[k + N/2] &= F_p[k] - W_N^k F_n[k]
 \end{aligned}$$

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  $g$ .

$$\begin{aligned} \mathcal{F}\{(f *_P g)\}[k] &= \sum_{i=0}^{P-1} \left( (f *_P g)[i] W_P^{ki} \right) = \sum_{i=0}^{P-1} \sum_{j=0}^{P-1} f_P[j] \cdot g_P[i-j] W_P^{ki} \\ &= \sum_{j=0}^{P-1} f_P[j] W_P^{kj} \sum_{i=0}^{P-1} g_P[i-j] W_P^{k(i-j)} \\ &= \sum_{j=0}^{P-1} f_P[j] W_P^{kj} \sum_{i=0}^{P-1} g_P[i] W_P^{k(i)}, \\ &\quad \text{(zbog periodičnosti } g_P) \\ &= \mathcal{F}\{f_P\}[k] \cdot \mathcal{F}\{g_P\}[k] \end{aligned}$$

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

$$\begin{aligned} h[k] &= \sum_{i=\max(0, k-M+1)}^{\min(N-1, k)} f[i] \cdot g[k-i] = (f *_P g)[k] = \mathcal{F}^{-1}\{\mathcal{F}\{f_P\} \mathcal{F}\{g_P\}\}[k], \\ &\quad \forall k \in [0, N+M-2] \end{aligned}$$

## Glava 3

### Projektovanje aplikacije na sistemskom nivou

#### 3.1 ESL Metodologija

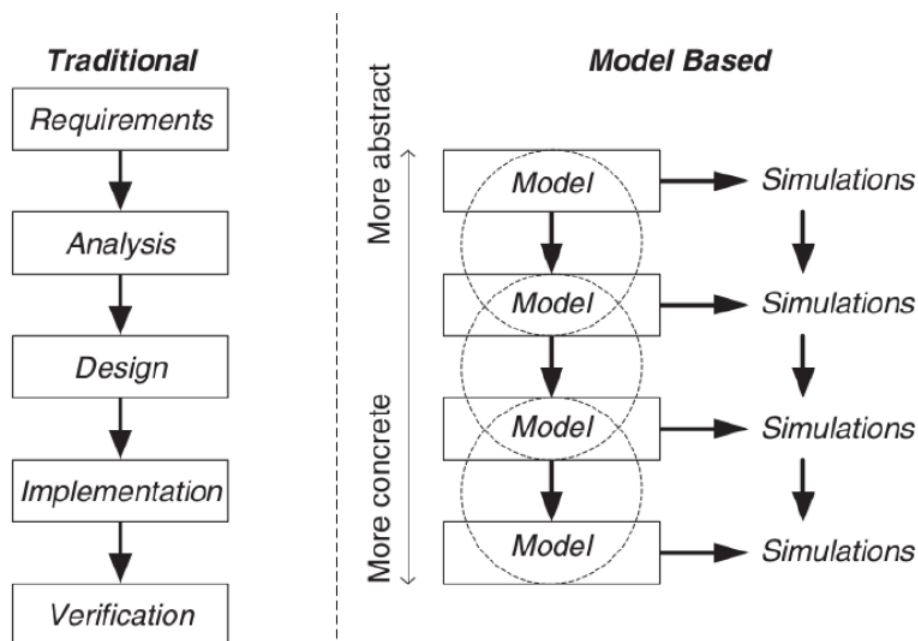
ESL (eng. *Electronic System Level*) metodologija predstavlja dizajn metodologiju za razvoj elektronskih uređaja. Zasnovana je na projektovanju modela uređaja na visokom nivou apstrakcije (odnosno na zahtevima poslodavca ili tržišta) i njegovim postepenim profinjavanjem do prostornog rasporeda svih hardverskih komponenti u uređaju i signala koji ih povezuju.

Korišćenje ESL metodologije je motivisano činjenicom da tradicionalne metodologije za razvoj elektronskih uređaja ne uspevaju da ispune zadate zahteve prilikom realizacije složenih elektronskih uređaja:

- Preko 70% dizajnova ima barem 30% niže performanse od zadatih.
- Preko 30% dizajnova nema korektno realizovano barem 50% zadate funkcionalnosti.
- Oko 54% dizajnova zakasni sa rokom realizacije, pri čemu je prosek kašnjenja četiri meseca.
- Oko 13% dizajnova se prekine pre završetka projekta.

Među glavnim razlozima za neuspeh realizacije dizajnova pomoću tradicionalnih metodologija su ograničena mogućnost preglednosti dizajna, i kontrole njegovog izvršavanja. Da bi se postiglo povećanje kontrolabilnosti i preglednosti, moraju se meriti karakteristike individualnih signala i logičkih kapija, sa preciznošću reda veličine jedne nanosekunde. Datim nivoima apstrakcije se dobijaju ogromne količine malo korisnih informacija.

ESL metodologija omogućava razvoj elektronski uređaja na sistemskom nivou, odnosno celokupno izvršavanje dizajna se apstrahuje u formi procesorskih instrukcija, poziva funkcija, pristupa memoriji i slanja paketa podataka. Na datom nivou se najjednostavnije i najbrže particioniše dizajn na hardverske i softverske komponente, i biraju IP blokovi. Prednosti ESL metodologije su povećani kvalitet i pouzdanost, realizacija optimalnih dizajnova, smanjenje vremena do plasiranja rešenja na tržište (eng. *time to market*), i mogućnost konkurentnog razvoja softvera i hardvera.



**Slika 5** Odnos između tradicionalnih i ESL metodologije razvoja elektronskih uređaja

ESL metodologija se sastoji iz sedam koraka:

1. **Pisanje specifikacije i modelovanje** – predstavlja razvoj dokumenta koji opisuje sistem ili funkcionalnost koju uređaj treba da ispuni, zajedno sa njegovim ograničenjima. Specifikacija na najvišem nivou je često napisana prirodnim jezikom, čime se ostavlja dovoljno prostora da model na sledećem, nižem nivou apstrakcije zadovolji zahtevane uslove bez da bude ograničen na specifičnu implementaciju. Za razliku od specifikacije napisane prirodnim jezikom, specifikacije koje se izvršavaju iskazuju željeno ponašanje na nedvosmislen način iz perspektive crne kutije. S druge strane, implementacije koje se izvršavaju reflektuju miktoarhitekturne odluke. Postoji veliki broj programskih jezika koji se koriste za ESL specifikaciju, u koje se ubrajaju MATLAB M-Code, SystemC i drugi.
2. **Analiza pre partitionisanja** – predstavlja proces upoređivanja karakteristika različitih implementacija. U ključne karakteristike se ubrajaju:
  - Vreme izvršavanja – algoritam može biti serijski ili konkurentno realizovan
  - Memorijski zahtevi
  - Potrošnja - najviše utiču spekulativno izvršavanje, keširanje podataka i izbor algoritma
  - Vreme potrebno da bi se plasirao proizvod na tržište – kašnjenje može da uzrokuje veoma velike finansijske gubitke

- Složenosti – pravljenje kompromisa između performansi i mogućnošću verifikacije rada

Da bi se mogle uporediti karakteristike mogućih implementacija, mogu se vršiti različite analize, kao što su statička, dinamička, algoritamska, i druge. U ovom radu će se vršiti profajliranje koda, na osnovu kojeg će se izvršiti particionisanje hardvera i softvera.

- 3. Particionisanje** – predstavlja proces raspodele algoritama (ili njihovih delova), definisanih u specifikaciji, na one koji će biti realizovani u softverskim (procesori), odnosno hardverskim komponentama. Pravi se arhitekturni opis jedinica za procesiranje, memoriju i komunikaciju. Hardverska particija donosi izbore vezane za broj korišćenih jezgara (distribuirani sistem) i da li će jezgra biti upravljana aperiodičnim događajima ili periodičnim signalima (klokom). Softversko particionisanje se odnosi na izbor operativnog sistema, biblioteka, tipa procesora (simetrični ili heterogeni), i na nižem procesorskom nivou na niti i procese. Konačno, u slučaju distribuiranog sistema, mora biti izabran algoritam za planiranje (eng. *scheduler*).
- 4. Analiza i debugovanje nakon particionisanja** – predstavlja proces ponovne analize karakteristika i upoređivanja karakteristikama dobijenim u analizi pre particionisanja. Algoritmi koji će biti izvršavani na procesoru su napisani u nekom programskom jeziku (na primer C, ili C++), dok su algoritmi koji će biti izvršavani u hardveru realizovani u nekom HDL (eng. *hardware description language*) jeziku. U ovom koraku često može doći do ponovnih iteracija između particionisanja i analize nakon particionisanja, sa ciljem optimizacije performansi.
- 5. Verifikacija nakon particionisanja** – predstavlja proces ispitivanja rada dizajna, sa ciljem da se utvrdi da li je željena funkcionalnost odgovarajućih komponenti pre particionisanja sačuvana nakon particionisanja dizajna. Tipičan tok verifikacije predstavlja planiranje verifikacije, implementacija verifikacionog okruženja i na kraju analiza verifikacionih rezultata. Uz pomoć formalne analize i simulacije, istražuje se prostor stanja u kojem se dizajn može naći. Verifikaciono okruženje se implementira pomoću HLVL (eng. *High-Level Verification Languages*) jezika. Ovo poglavlje će detaljnije biti razjašnjeno u glavi 5.
- 6. Hardversko-softverska implementacija** – Hardverska implementacija predstavlja proces kreiranja modela koji se mogu sintetizovati do nivoa kapija, i obično se pišu na RT nivou. Postoji pet mogućih hardverskih implementacija, a to su ekstenzibilni procesori (eng. *extensible processors*), DSP (eng. *digital signal processing*) koprocesori, VLIW (eng. *very long instruction word*) koprocesori, namenski (eng. *application-specific*) koprocesori, i ASIC ili FPGA. Softverska implementacija se može klasifikovati prema vidljivosti sistemskih komponenti (klok, uređaj za enkripciju i slično), vremenskim

intervalima između susednih događaja, platforme na kojoj se izvršava i vremenskih performansi.

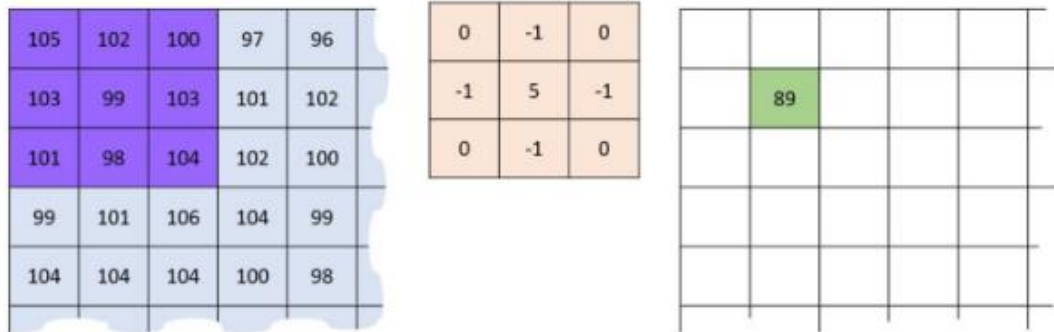
7. **Verifikacija implementacije** – predstavlja verifikaciju ispravnog rada celokupnog sistema. Može se postići simulacijom mešovitenih modela, hardverskom emulacijom, ili realizacijom prototipa pomoću FPGA. Radi lakšeg debugovanja hardvera, često se ugrađuju komponente kao što su (eng. *Joint Test Action Group*), ILA (eng. *integrated logic analyzer*) komponente.

### 3.2 Specifikacija i modelovanje

Konvolucija je operacija dodavanja svakog elementa svojim susedima, pomnoženih sa odgovarajućim koeficijentima. Ako imamo dve 3x3 matrice istih dimenzija, onda konvolucija se definiše na sledeći način:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} * \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} [2,2] = (i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9).$$

Rezultat konvolucije se smešta u novu matricu, na istu poziciju na kojoj se nalazi i centralno polje parčeta originalne slike (pogledati sliku ispod, polje [2,2]). Slično se definiše i za veće dimenzije matrica. Formalna matematička definicija konvolucije je data u glavi 2, gde je takođe pokazano kako je Furijeova transformacija povezana sa konvolucijom.



**Slika 6** Konvolucija matrica

### 3.3 Analiza pre particionisanja

U ovom koraku je izvršeno upoređivanje performansi različitih algoritama i delova algoritma. Implementacijom konvolucije po definiciji se postiže vremenska složenost od  $\mathcal{O}(A_W \cdot A_H \cdot B_W \cdot B_H)$ , gde su A i B matrice, dok bi se korišćenjem brzih Furijeovih transformacija na osnovu „Radix 2“ algoritma ta vremenska složenost mogla smanjiti na  $\mathcal{O}\left((A_W + B_W - 1) \cdot (A_H + B_H - 1) \cdot \left(\log_2(A_H + B_H - 1) + \log_2(A_W + B_W - 1)\right)\right)$ . S druge strane, korišćenje brzih Furijeovih transformacija zahteva preko duplo više memorijskih lokacija, koje su zadužene za

skladištenje vrednosti imaginarnih članova matrica i rotacionih faktora. Zbog značajne uštede vremena izvršavanja, izabrana je implementacija zasnovana na brznoj Furijeovoj transformaciji.

Profajliranjem izvornog koda napisanog u C++ jeziku dobijeni su sledeći rezultati odnosa vremena izvršavanja konvolucije matrica i brze Furijeove transformacije:

Veličina matrica	Relativno vreme izvršavanja konvolucije [%]	Relativno vreme izvršavanja brze Furijeove transformacije [%]
8x8	34.02	65.52
128x128	6.24	93.24

**Tabela 1** Rezultati profajliranja

Za još veće matrice, odnos vremena izvršavanja brze Furijeove transformacije u dve dimenzije u odnosu na konvoluciju će biti još veći, pošto je vreme izvršavanja konvolucije linearno sa brojem elemenata matrice,  $\mathcal{O}(n)$ , za razliku od dvodimenzionalne Furijeove transformacije koja je oblika  $\mathcal{O}(n \log n)$ , kao što je malo pre prikazano.

### 3.4 Particionisanje

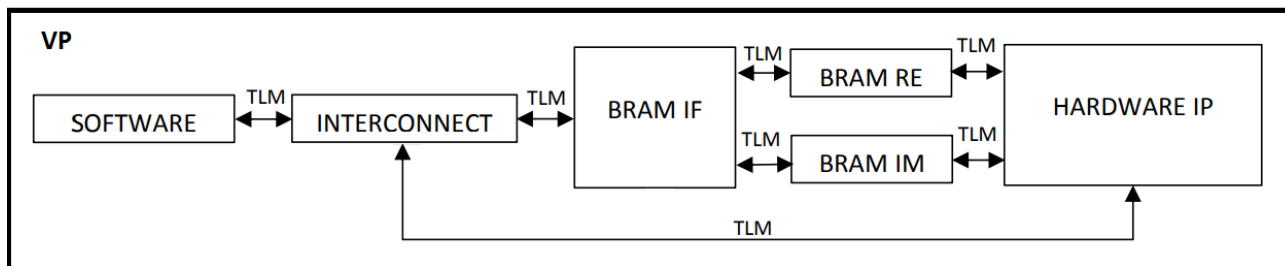
Dvodimenzionalna Furijeova transformacija je vremenski značajno složeniji algoritam od konvolucije. Zbog toga je odlučeno da se Furijeova transformacija izvršava na hardveru, dok će se konvolucija izvršavati u softveru. Realni brojevi u hardveru će biti predstavljeni sa fiksnom tačkom.

Ciljana platforma u ovom sistemu će biti Zybo ploča iz familije Zynq-7010. Zybo predstavlja sistem na čipu koji se sastoji iz programabilne logike (FPGA) i procesirajućeg dvojezrog sistema ARM Cortex A9. Pošto nije zamišljeno da se ubrzava konvolucija sa paralelizacijom softverske particije, koristiće se samo jedno jezgro procesora. Operativni sistem na ploči će biti *Xilinx Linux Kernel 4.10*.

Matrice koje se obrađuju će biti skladištene u BRAM-u. Direktna komunikacija između softverskog i hardverskog dela će se obavljati preko AXI Lite magistrale. Particije će razmenjivati informacije o veličini matrica preko date magistrale, kao i signale za početak i završetak obrade matrica. Softverska particija komunicira sa BRAM-om putem AXI-BRAM kontrolera, dok je hardverska particija direktno povezana putem BRAM interfejsa sa memorijom.

### 3.5 Implementacija i analiza modela posle particionisanja

Model je implementiran u programskom jeziku *SystemC*. Komponente su instancirane u vidu klasa, i mogu se podeliti na komponente zadužene za skladištenje memorije (BRAM RE i BRAM IM), softversku i hardversku particiju (Hard i Soft) i komponente koje povezuju date module (Interconnect i BRAM Controller).



**Slika 7** Šematski prikaz virtuelne platforme

Sve komponente komuniciraju na nivou transakcija putem TLM (*eng.* Transaction Level Modelling) mehanizma. Komunikacija na nivou transakcija omogućava razdvajanje implementacije protokola od funkcionalnosti koju treba da obavljaju komunikacioni kanali. Na osnovu toga se jednostavno mogu menjati protokoli u skladu sa zahtevima dizajna. Transakcija koja se prenosi putem TLM mehanizma sadrži nekoliko parametara, od kojih su najznačajniji dužina niza koji se šalje (svi podaci su osmobitni), pokazivač na niz i komanda koja je oblika „čitaj“ ili „upiši“.

Komponenta *Soft* vrši implementaciju konvolucije, odnosno šalje zahteve hardverskom modulu da izvrši dvodimenzionalne Furijeove transformacije nad matricama, obavlja zamenu redosleda matrica u memoriji, kao i jednostavne operacije nad elementima matrica. Pseudo kod je napisan u listingu ispod.

1. Slanje zahteva hardverskoj particiji za izvršavanje dvodimenzionalne Furijeove transformacije nad ulaznom realnom matricom A
2. Zamena redosleda obrađene matrice A i matrice B
3. Slanje zahteva hardverskoj particiji za izvršavanje dvodimenzionalne Furijeove transformacije nad ulaznom realnom matricom B
4. Množenje elemenata novih matrica A i B i skladištenje na lokaciji matrice B (zove se matrica C) i invertovanje vrednosti imaginarnog dela matrice C
5. Slanje zahteva hardverskoj particiji za izvršavanje dvodimenzionalne Furijeove transformacije nad ulaznom realnom matricom C
6. Deljenje svakog elementa sa proizvodom visine i širine matrice C

Listing 1 Pseudo-kod algoritma koji izvršava *Soft* komponenta

Hard komponenta izvršava dvodimenzionalnu Furijeovu transformaciju, koja je opisana u poglavljima 2.4 i 2.5.

*Interconnect* komponenta služi za rutiranje podataka koji šalje komponenta *Soft*, i na osnovu prvih osam bita tridesetdvobitne adrese utvrđuje da li da transakciju prosledi memoriji ili direktno u hardverski blok. BRAM kontroler prima transakciju iz *Interconnect* komponente. Transakcija koja stiže u BRAM kontroler sadrži niz dvostruke dužine, pri čemu prva polovina niza predstavlja lokaciju za podatke koji treba da se učitaju ili upišu u memoriju sa realnim podacima, dok druga polovina niza je rezervisana za memoriju sa imaginarnim podacima. Stoga, može

se reći da BRAM kontroler jednu transakciju deli na dva dela, i ponovo sastavlja transakcije nakon uspešnog pisanja ili čitanja.

Memorijske komponente BRAM RE i BRAM IM skladište vrednosti dve kompleksne matrice. Prva polovina memorijskih lokacija je rezervisana za jednu matricu (u početku se to odnosi na matricu A, do zamene redosleda), dok je druga polovina rezervisana za drugu matricu. Da bi se omogućilo istovremeno upisivanje realnog i imaginarnog dela (i time uštedelo na vremenu), realni i imaginarni delovi matrica su razdeljeni u dva BRAM-a.

Ceo sistem se zove virtuelna platforma. Klasa VP (eng. *Virtual Platform*) instancira sve komponente, i spaja odgovarajuće interfejsse jedne sa drugima.

### 3.6 Analiza modela posle particionisanja

Realne vrednosti matrica su reprezentovane sa fiksnom tačkom. Da bi se utvrdila optimalna širina celog dela, i dela iza fiksne tačke, neophodno je izvršiti analizu širine. Napisana su dva koda sa identičnim algoritmom, u C++ i SystemC programskim jezicima, pri čemu se u prvom slučaju koristi reprezentacija realnih brojeva sa pokretnom tačkom, dok u drugom slučaju se koristi reprezentacija sa fiksnom tačkom. Vršena je analiza nad realnim matricama veličine 128x128 proizvoljnih vrednosti, i mereno je odstupanje između odgovarajućih elemenata u rezultujućim matricama.

Širina <ceolobrojni deo, realni deo>	Odstupanje
<32,12>	12288
<32,13>	8192.02
<32,14>	16384
<32,15>	0.0273132
<32,16>	0.02348330
<32,17>	0.0534661
<32,18>	0.118255
<48,18>	0.0233966
<48,19>	0.023397
<48,20>	0.023398
<48,21>	0.0234

**Tabela 2** Analiza širine

Po datim rezultatima, može se zaključiti da širina od 32 bita, sa 16 bita koji se nalaze iza fiksne tačke, predstavlja minimalnu širinu realnih brojeva za matrice veličine 128x128. Za veće matrice bi širina realnih brojeva morala biti još veća.



## Glava 4

### Projektovanje hardverskog IP jezgra

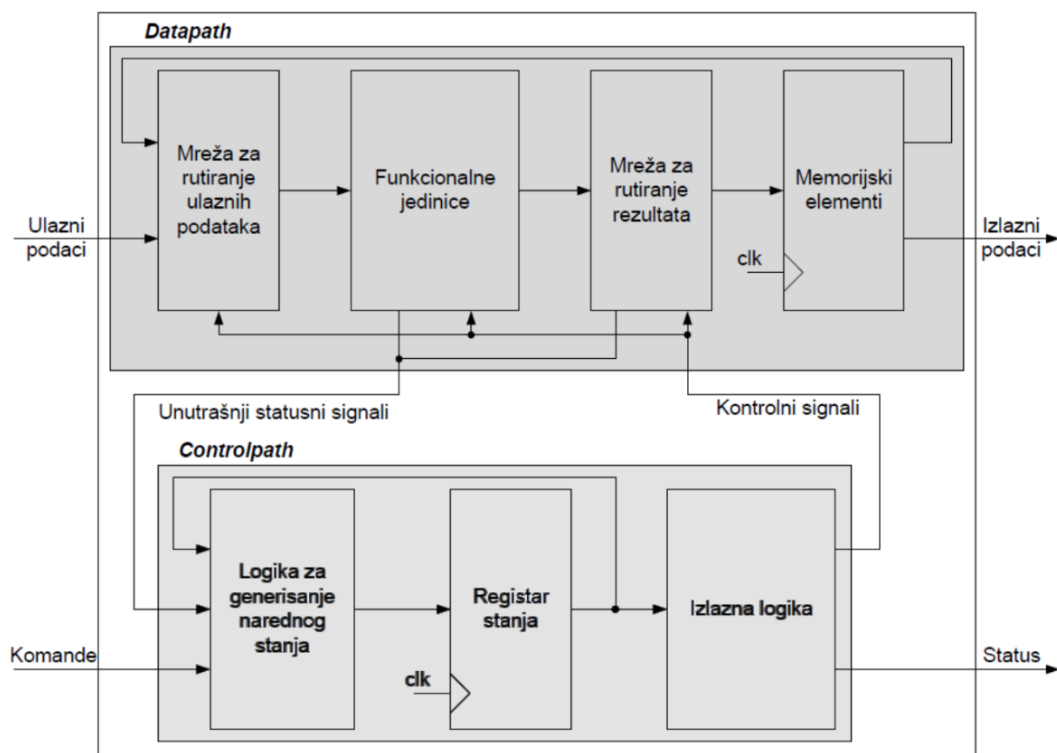
#### 4.1 Uvod

##### 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 u registre, i sekvenci koraka nad varijablama u 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 – sastoje 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 8** 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 multipleksterska 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 blok 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:

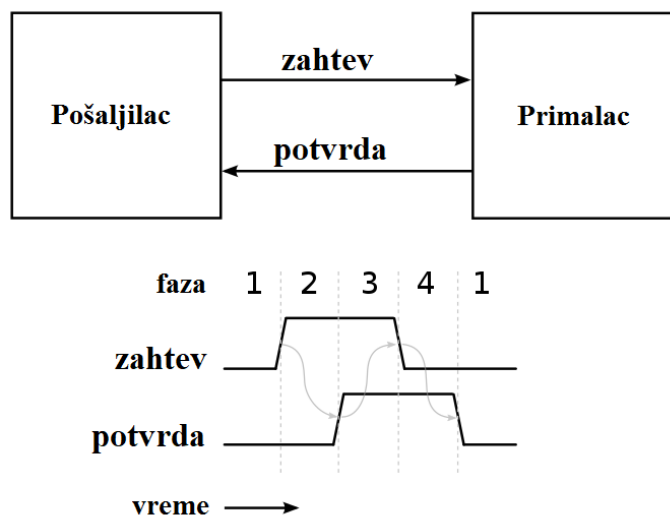
- Module za skladištenje podataka – u ovu grupu spadaju IP jezgra koja implementiraju mehanizme za skladištenje podataka, kao što su SRAM, DRAM, ROM, Flash, FIFO, i drugi.
- Module za prenos podataka - u ovu grupu spadaju IP jezgra koja implementiraju i međusobno prevode različite komunikacione protokole, kao što su AXI, PLB, PCIe, Ethernet, I<sup>2</sup>C, 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 pomoću njih, pa se stoga uglavnom posmatraju odvojeno. Osnovni 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 pošaljioa 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, pošaljalac 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 pošaljalac 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.

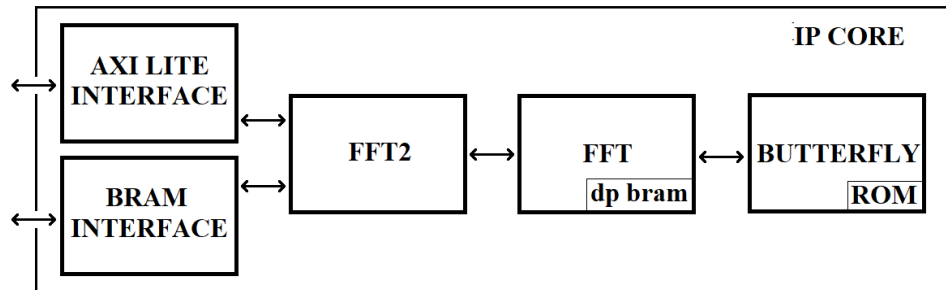


Slika 9 Šematski prikaz protokola četvorofaznog hendšejk protokola

## 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 unutrašnji dvoprístupni 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 izvršava leptir operaciju nad dva kompleksna broja. Unutar BUTTERFLY modula se nalazi ROM (eng. *Read Only Memory*) iz kojih se čitaju vrednosti rotacionih faktora.



**Slika 10** 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 (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 korišćenja većeg broja istih modula (na primer više *fft* ili *butterfly* modula), može se jednostavnije realizovati modul za arbitraciju i raspoređivanje (eng. *scheduling*) 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 izvršavanja 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<sub>i</sub> i topIM<sub>i</sub>).

#### 4.3.1 Definisane interfejsa

- Ulazni interfejs

topRE<sub>i</sub> – 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<sub>i</sub> – 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<sub>i</sub> – 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<sub>i</sub> – tipa STD\_LOGIC\_VECTOR (WIDTH-1 downto 0) – predstavlja donji imaginarni ulaz unutar leptir bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED\_POINT\_WIDTH.

$k$  – tipa STD\_LOGIC\_VECTOR ( $\log_2(\text{FFT\_SIZE}/2)-1$  downto 0) – označava trenutnu vrednost rotacionog faktora  $W_{2^{\text{size}}}^k$ .

$\text{size}$  – tipa STD\_LOGIC\_VECTOR ( $\log_2(\log_2(\text{FFT\_SIZE}/2)-1$  downto 0)) – predstavlja logaritam veličine niza za koji se trenutno računa Furijeova transformacija. Zajedno sa veličinom  $k$  u potpunosti opisuje trenutnu vrednost rotacionog faktora  $W_{2^{\text{size}}}^k$ .

- Izlazni interfejs

$\text{topRE\_o}$  – tipa STD\_LOGIC\_VECTOR ( $\text{WIDTH}-1$  downto 0) – predstavlja gornji realni izlaz unutar leptir bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED\_POINT\_WIDTH.

$\text{topIM\_o}$  – tipa STD\_LOGIC\_VECTOR ( $\text{WIDTH}-1$  downto 0) – predstavlja gornji imaginarni izlaz unutar leptir bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED\_POINT\_WIDTH.

$\text{botRE\_o}$  – tipa STD\_LOGIC\_VECTOR ( $\text{WIDTH}-1$  downto 0) – predstavlja donji realni izlaz unutar leptir bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED\_POINT\_WIDTH.

$\text{botIM\_o}$  – tipa STD\_LOGIC\_VECTOR ( $\text{WIDTH}-1$  downto 0) – predstavlja donji imaginarni izlaz unutar leptir bloka. Vrednost je predstavljena sa fiksnom tačkom širine FIXED\_POINT\_WIDTH.

- Komandni interfejs

$\text{start}$  – tipa STD\_LOGIC – kontroliše početak rada leptir bloka

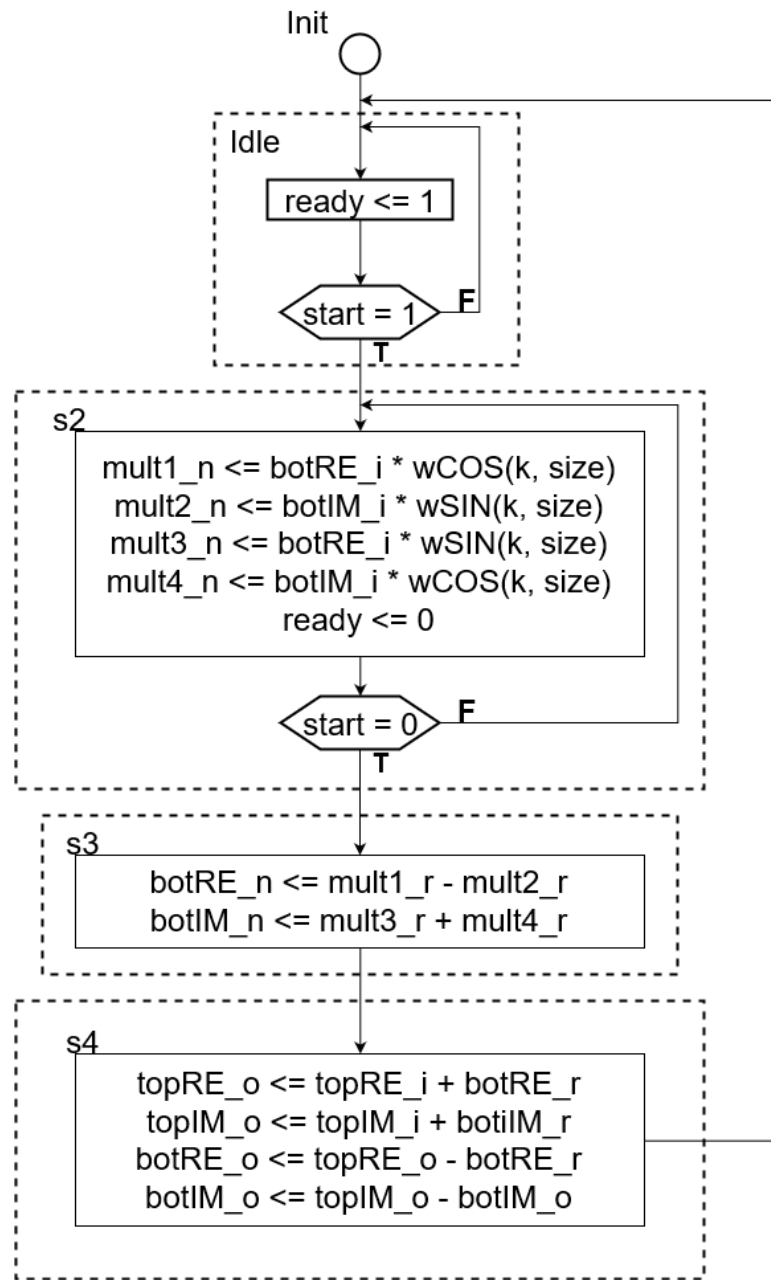
- Statusni interfejs

$\text{ready}$  – 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.

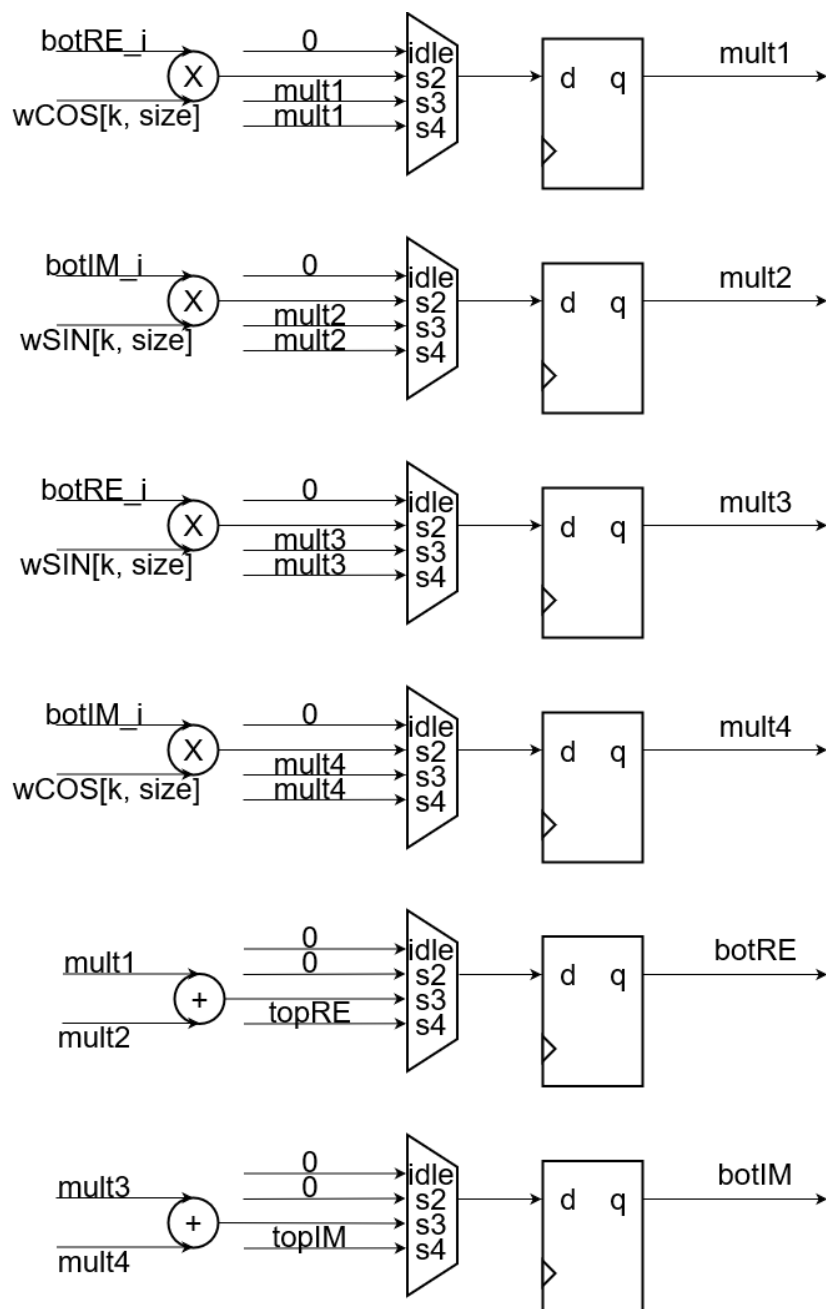




Slika 11 ASM dijagram leptir bloka

### 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 12** 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 **obrtnanje 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: 0001 1100 1100 → 0000 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 2** Algoritam za obrtnanje redosleda bita

Upravljački modul algoritma za obrtnanje 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 *kk* (linije 2, 3 i 4 listinga), u stanju **L2** se ažuriraju vrednosti promenljivih *reversed* i *temp* (linije 5, 6 i 7 listinga), 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 potvrđan odgovor da su stigli podaci na ulazne portove. U stanju **RD** se upisuju vrednosti sa ulaza u unutrašnju memoriju (linije 9 i 10 listinga), 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.

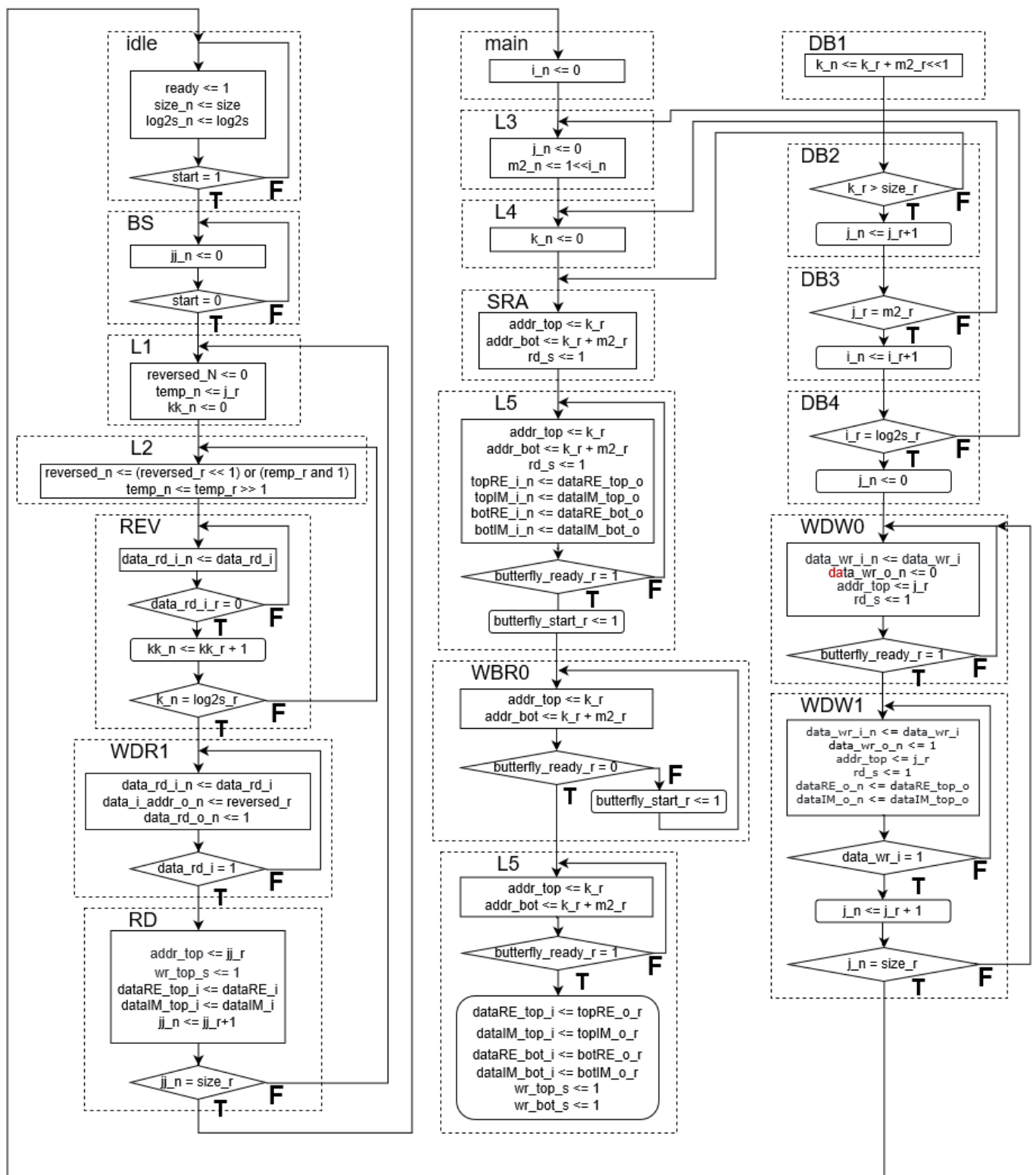
```

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 3** Iterativni algoritam za rekurzivnu podelu niza i pozivanje leptir operacije

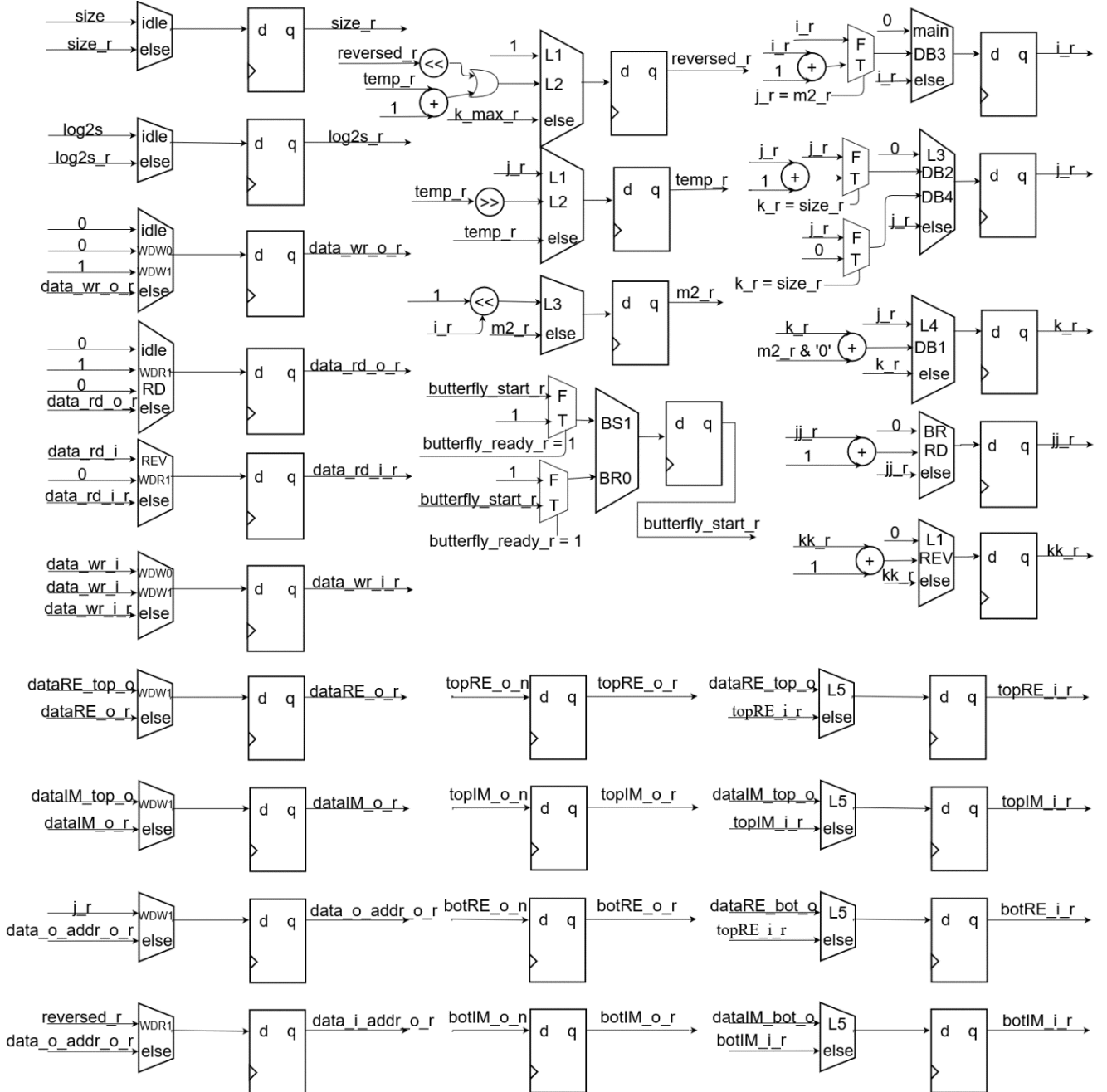
Upravljački modul algoritma za rekurzivnu podelu niza se sastoji iz stanja **MAIN**, **L3**, **L4**, **SRA** (*set read address*), **L5**, **WBR0** (*wait butterfly ready 0*), **WBR1** (*wait butterfly ready 1*), **DB1**, **DB2**, **DB3**, **DB4** (*do butterfly*), **WDW0** (*wait data write 0*) i **WDW1** (*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 13 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 direktno 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 14 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

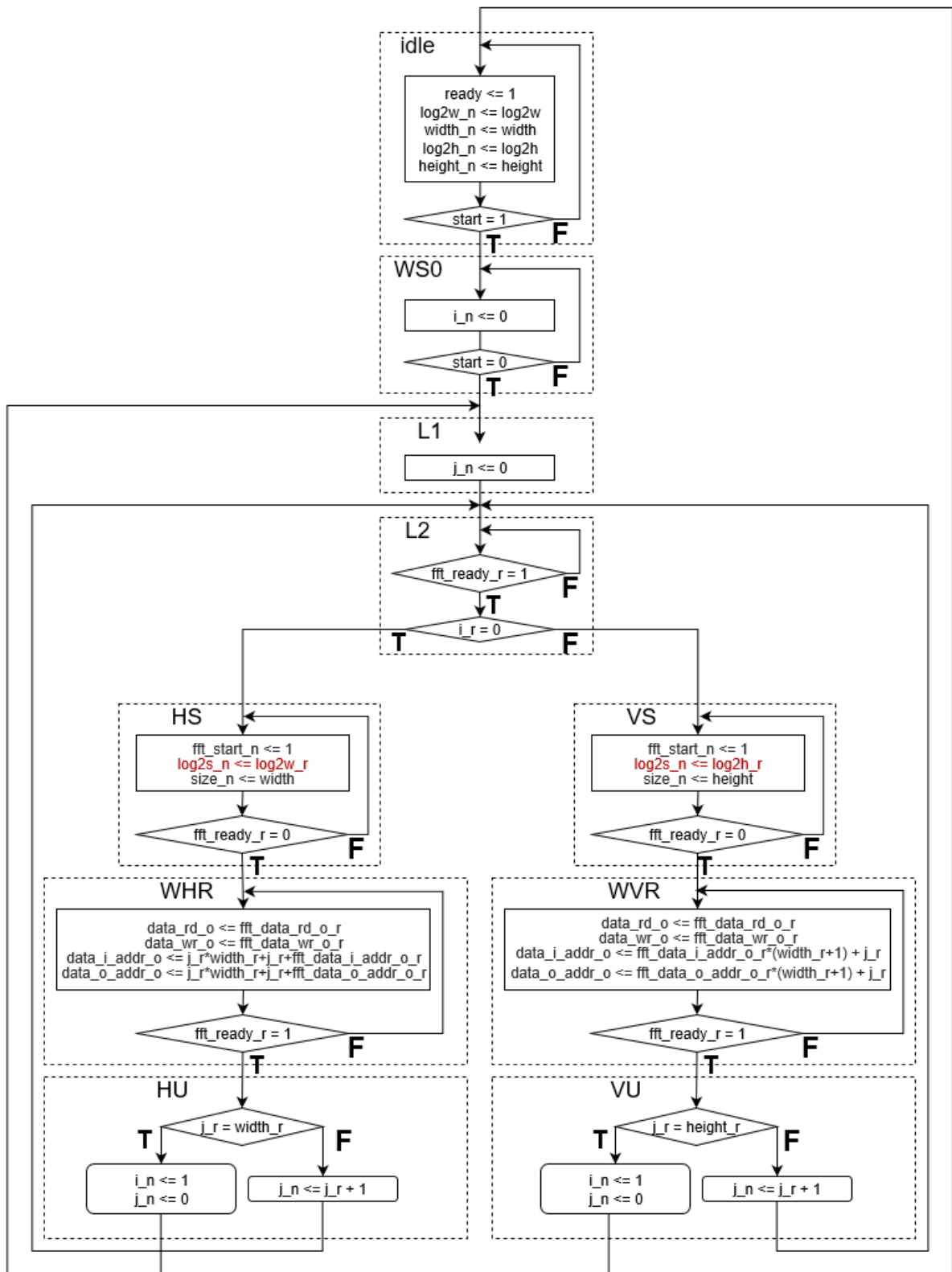
#### 4.5.2 Projektovanje upravljačkog modula

Blok za dvodimenzionu Furijeovu transformaciju ima ulogu da bude most između memorija u kojima su uskladištene 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 omogućiti 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, i njihovih logaritama, dok god je *start* signal na niskom logičkom 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 (*i\_r* ukazuje da li se kolone ili redovi obrađuju).

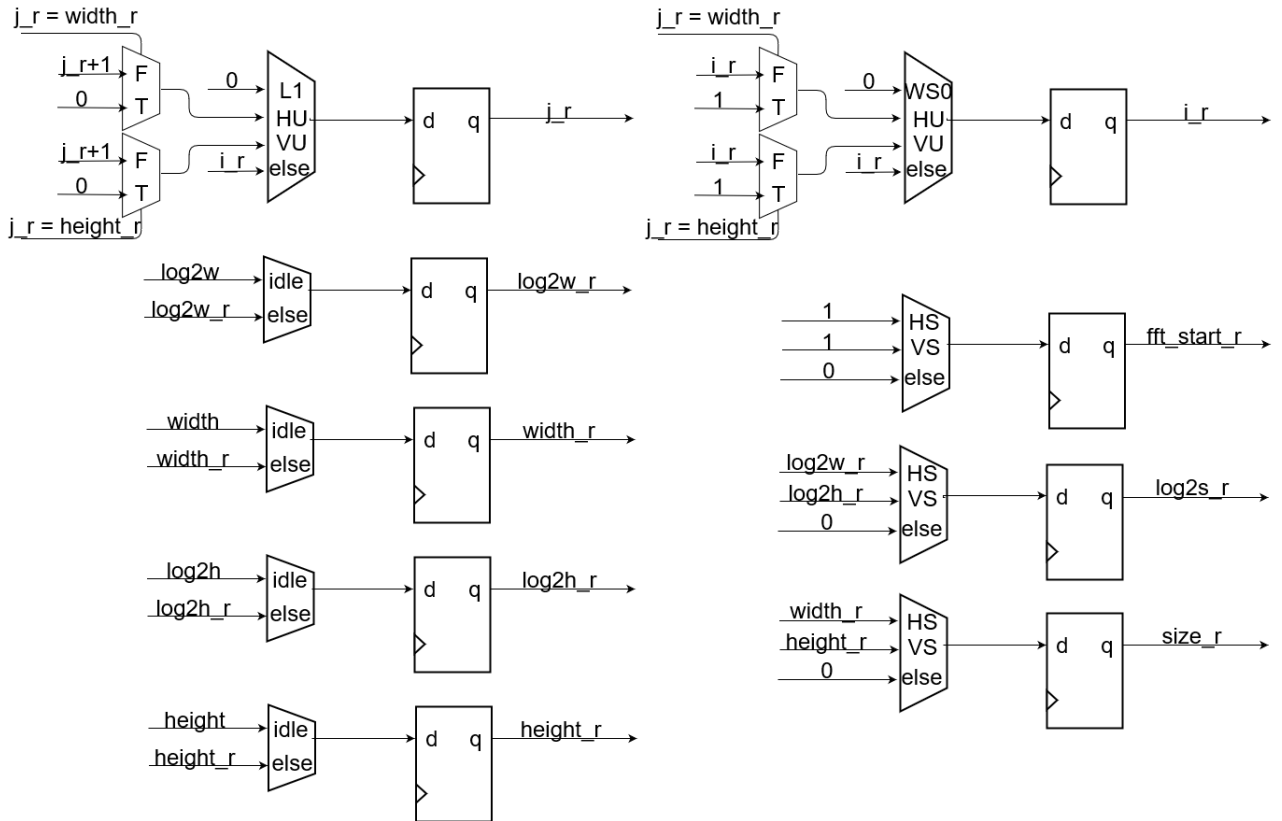
U stanju **L1** se inicijalizuje vrednost *j* registra, koji označava broj reda ili kolone koji se trenutno obrađuje u *fft* bloku. U stanju **L2** se proverava da li je *fft* blok spreman da obrađuje sledeći niz. Ukoliko jeste, podaci o veličini i *start* signal se šalju u **HS (VS)** stanju. U stanju **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 vertikalni) 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 15 ASM dijagram *fft2* modula

### 4.5.3 Projektovanje modula za obradu podataka

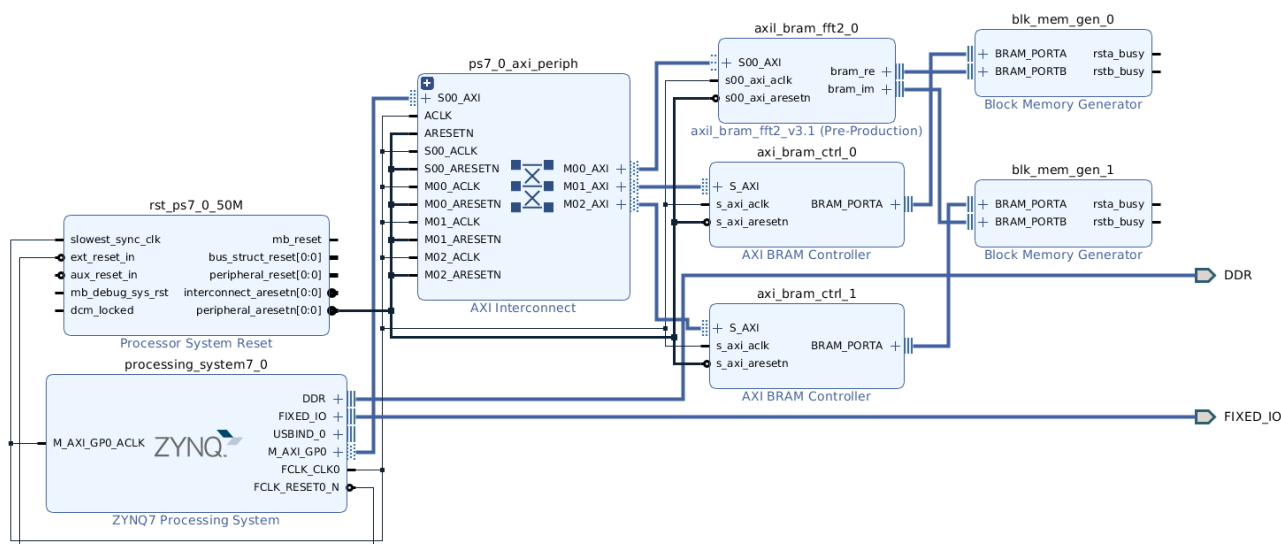
Modul za obradu podataka *fft2* bloka se sastoji iz devet registara. Četiri registra skladište ulazne podatke, odnosno širinu i visinu matrica, kao i njihove logaritme. Veličine tih registara su  $\text{ld}(\text{FFT\_SIZE} \times \text{FFT\_SIZE})$  i  $\text{ld}(\text{ld}(\text{FFT\_SIZE} \times \text{FFT\_SIZE}))$ . Dva registra služe da skladište veličinu niza koji će biti obrađivan u bloku *fft*, kao i njegov logaritam, i veličina tih registara je  $\text{ld}(\text{FFT\_SIZE})$  i  $\text{ld}(\text{ld}(\text{FFT\_SIZE}))$ . Indikator  $i\_r$  je jednobitni registar, dok je iterator  $j\_r$  veličine  $\text{ld}(\text{FFT\_SIZE})$ .



Slika 16 Modul za obradu podataka *fft2* bloka

## 4.6 Integrisanje u sistem i merenje performansi

Nakon završetka projektovanja IP jezgra, mogu se povezati sve komponente u jedinstven sistem, prikazano na slici ispod, i pristupiti generisanju binarnog koda na osnovu kojeg će se programirati FPGA. Sistem se sastoji iz Zynq7 procesora, dva *true dual-port* BRAM-a povezana preko dvoprístupnog BRAM interkonekta sa procesorom, i IP jezgra za dvodimenzionu Furijeovu transformaciju. IP jezgro je direktno povezano sa memorijom i Zynq7 procesorom pomoću BRAM i AXI Lite interfejsa redom, pri čemu IP jezgro koristi port B od memorija, dok BRAM interkonekt koristi port A. BRAM interkonekt je takođe povezan sa procesorom preko AXI Lite interfejsa.



Slika 17 Blok dijagram celokupnog sistema

Nakon obavljanja vremenske (eng. *timing*) analize, dobijamo sledeće rezultate:

Setup time (Worst Slack):	6.143ns
Hold time (Worst Slack):	0.026ns
Pulse Width (Worst Slack):	9.020ns

Drugim rečima, kad se saberu prethodne vrednosti vremena, minimalno trajanje jedne periode klock signala iznosi 15.189ns, odnosno maksimalna frekvencija rada ploče iznosi 65.837MHz.

## 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  
<https://www.elektronika.ftn.uns.ac.rs/projektovanje-slozenih-digitalnih-sistema/specifikacija/specifikacija-predmeta/>
- [3] Rastislav Struharik, vežbe i predavanja za predmeta Funkcionalna verifikacija  
<https://www.elektronika.ftn.uns.ac.rs/funkcionalna-verifikacija/specifikacija/specifikacija-predmeta/>
- [4] Vuk Vranjković, vežbe i predavanja za predmeta Projektovanje elektronskih uređaja na sistemskom nivou  
<https://www.elektronika.ftn.uns.ac.rs/projektovanje-elektronskih-uredjaja-na-sistemskom-nivou/specifikacija/specifikacija-predmeta>
- [5] <https://www.geeksforgeeks.org/convert-a-real-number-between-0-and-1-to-binary-string/>, pogledano dana 13.9.2019.
- [6] <https://www.geeksforgeeks.org/iterative-fast-fourier-transformation-polynomial-multiplication>, pogledano dana 19.9.2019.
- [7] [http://alwayslearn.com/DFT%20and%20FFT%20Tutorial/DFTandFFT\\_FFT\\_TwiddleFactor.html](http://alwayslearn.com/DFT%20and%20FFT%20Tutorial/DFTandFFT_FFT_TwiddleFactor.html), pogledano dana 19.9.2019.