



UNIVERZITET U NOVOM SADU  
FAKULTET TEHNIČKIH NAUKA  
KATEDRA ZA ELEKTRONIKU



# **Hardverska implementacija akceleratora za ispravljanje sadržaja slika na Zybo razvojnoj ploči**

diplomska teza (bečelor)

Kandidat  
Srđan Babić EE114/2014

Mentor  
prof. dr Vuk Vranjković

Novi Sad, oktobar 2022.



## 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>	Srđan Babić		
Mentor, <b>MN:</b>	Prof. dr Vuk Vranjković		
Naslov rada, <b>NR:</b>	Hardverska implementacija akceleratora za ispravljanje sadržaja slika na Zybo razvojnoj ploči		
Jezik publikacije, <b>JP:</b>	Srpski		
Jezik izvoda, <b>Jl:</b>	Srpski		
Zemlja publikovanja, <b>ZP:</b>	Srbija		
Uže geografsko područje, <b>UGP:</b>	Vojvodina		
Godina, <b>GO:</b>	2022.		
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)	4/39/6/1/5/37/0/0		
Naučna oblast, <b>NO:</b>	Elektrotehnika i računarstvo		
Naučna disciplina, <b>ND:</b>	Embedded sistemi i algoritmi		
Predmetna odrednica/Ključne reči, <b>PO:</b>	Obrada slika, hardverska implementacija, razvojna ploča, IP jezgro		
<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 radu je projektovan sistem za ispravljanje slika belih cifara na crnoj pozadini. Iskorišćen je algoritam koji se zasniva na afinoj transformaciji slike. Određen je vremenski najzahtevniji deo algoritma, i primenom RTL metodologije je implementirano IP jezgro koje ubrzava izvršavanje kritičnog dela algoritma. Opisane su ciljana i postignuta frekvenciju rada, kritične putanje u jezgri, kao i zauzeće hardverskih resursa nakon implementacije na FPGA Zybo razvojnoj ploči.		
Datum prihvatanja teme, <b>DP:</b>	14.10.2022.		
Datum odbrane, <b>DO:</b>	.		
Članovi komisije, <b>KO:</b>	Predsednik:		
	Član:		Potpis mentora
	Član, Mentor		



# 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> :		Graduate thesis	
Author, <b>AU</b> :		Srđan Babić	
Mentor, <b>MN</b> :		Ph.D Vuk Vranjković	
Title, <b>TI</b> :		Hardware implementation of Image content deskewing accelerator using Zybo development board	
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> :		2022	
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)		4/39/6/1/5/37/0/0	
Scientific field, <b>SF</b> :		Electrical engineering	
Scientific discipline, <b>SD</b> :		Embedded systems and algorithms	
Subject/ Key words, <b>S/KW</b> :		Image preprocessing, hardware implementation, development board, IP core	
<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> :		A system for deskewing images of white digits and numbers on black background is implemented and described in this paper. The used algorithm is based on affine transform. The most time consuming part of the algorithm is selected for hardware acceleration. RTL methodology is used to develop an IP core for algorithm acceleration. Desired and achieved operating frequency are described, as well as critical paths and hardware resource utilisation after the system is placed on the FPGA Zybo development board.	
Accepted by the Scientific Board on, <b>ASB</b> :		14.10.2022.	
Defended on, <b>DE</b> :			
Defended board, <b>DB</b> :	President:		
	Member:		Mentor's signature
	Member, Mentor		

## Izjava o akademskoj čestitosti

Student/kinja: \_\_\_\_\_

Broj indeksa: \_\_\_\_\_

Student/kinja: osnovnih ili master akademskih studija \_\_\_\_\_

Autor/ka rada pod nazivom:

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_


Potpisivanjem izjavljujem:

- da je rad isključivo rezultat mog sopstvenog istraživačkog rada;
- da sam rad i mišljenja drugih autora koje sam koristio/la u ovom radu naznačio/la ili citirao/la i navedeni u spisku literature/referenci koji su sastavni deo ovog rada;
- da sam dobio/la sve dozvole za korišćenje autorskog dela koji se u potpunosti/celosti unose u predati rad i da sam to jasno naveo/la;
- da sam svestan/na da je plagijat korišćenje tuđih radova u bilo kom obliku (kao citata, parafraza, slika, tabela, dijagrama, dizajna, planova, fotografija, filma, muzike, formula, veb sajtova, kompjuterskih programa i sl.) bez navođenja autora ili predstavljanje tuđih autorskih dela kao mojih, kažnjivo po zakonu (Zakon o autorskom i srodnim pravima, Službeni glasnik Republike Srbije, br. 104/2009, 99/2011, 119/2012), kao i drugih zakona i odgovarajućih akata Univerziteta u Novom Sadu;
- da sam svestan/na da plagijat uključuje i predstavljanje, upotrebu i distribuiranje rada predavača ili drugih studenata kao sopstvenih;
- da sam svestan/na posledica koje kod dokazanog plagijata mogu prouzrokovati na predati rad i moj status;
- da je elektronska verzija rada identična štampanom primerku i pristajem na njegovo objavljivanje pod uslovima propisanim aktima Univerziteta.

Novi Sad, \_\_\_\_\_

Potpis studenta/kinje

\_\_\_\_\_

	UNIVERZITET U NOVOM SADU & FAKULTET TEHNIČKIH NAUKA 21000 NOVI SAD, Trg Dositeja Obradovića 6	Datum:
	<b>ZADATAK ZA IZRADU DIPLOMSKOG (BACHELOR) RADA</b>	List/Listova
		1/1

(Podatke unosi predmeti nastavnik - mentor)

Vrsta studija	<input type="checkbox"/> Osnovne akademske studije		
Studijski program:	Energetika, elektronika i telekomunikacije		
Rukovodilac studijskog programa	Dr Milan Sečujski		
Student:	Srđan Babić	Broj Indeksa	EE 53/2014
Oblast	Projektovanje složenih digitalnih sistema		
Mentor	Prof. dr Vuk Vranjković		
NA OSNOVU PODNETE PRIJAVA, PRILOŽENE DOKUMENTACIJE I ODREDBI STATUTA FAKULTETA IZDAJE SE ZADATAK ZA DIPLOMSKI (Bachelor) RAD, SA SLEDEĆIM ELEMENTIMA: <ul style="list-style-type: none"> <li>- Problem-tema rada</li> <li>- Način rešavanja problema i način praktične provere rezultata rada, ako je takva provera neophodna;</li> <li>- Literatura</li> </ul>			

### NASLOV DIPLOMSKOG (BACHELOR) RADA

Hardverska implementacija akceleratora za ispravljanje sadržaja slika na Zybo razvojnoj ploči

### TEKST ZADATKA:

Odrediti vremenski najzahtevniji deo algoritma u sistemu koji ispravlja sadržaj slika. Primenom RTL metodologije projektovati i implementirati IP jezgro koje će ubrzati izvršavanje kritičnog dela algoritma. Odrediti frekvenciju rada, kritičnu putanju, kao i zauzeće hardverskih resursa na FPGA Zybo razvojnoj ploči.

Rukovodilac studijskog programa:	Mentor rada:
	Vuk Vranjković
Primerak za: <input type="checkbox"/> Studenta; <input type="checkbox"/> Mentora;	

## Slike

Slika 1: Primer affine transformacije (preuzeto sa [2]).....	3
Slika 2: Grafički prikaz bilinearne interpolacije .....	6
Slika 3: Struktura ZYNQ 7010 SoC-a, preuzeto sa [9] .....	7
Slika 4: Struktura virtuelne platforme.....	8
Slika 5: Particionisanje IP jezgra na Controlpath i Datapath[7] .....	10
Slika 6: Simulaciono okruženje za testiranje RTL modela.....	11
Slika 7: Ulazna i izlazna slika za simulaciju RTL modela .....	12
Slika 8: Interfejsi IP-ja .....	13
Slika 9: Podela kontrolne jedinice i njeni interfejsi .....	14
Slika 10: AXI LITE transakcija upisa.....	15
Slika 11: AXI LITE transakcija čitanja .....	15
Slika 12: Šematski prikaz AXI LITE kontrolera .....	16
Slika 13: Mašina stanja kanala za upis AXI kontrolera .....	17
Slika 14: Mašina stanja kanala za čitanje AXI kontrolera .....	17
Slika 15: Mreža za upis i čuvanje vrednosti konfiguracionih registara .....	18
Slika 16: Logika za postavljanje i čišćenje statusnih signala .....	19
Slika 17: Logika za čitanje registara .....	20
Slika 18: Interfejs kontrolera prekida .....	20
Slika 19: Konfiguracioni FSM.....	21
Slika 20: Struktura i interfejs Datapath modula .....	23
Slika 21: Struktura i interfejs memorijskog kontrolera.....	24
Slika 22: Blok šema FIFO bafera.....	24
Slika 23: Adresni generator .....	25
Slika 24: Struktura adresnog akumulatora .....	26
Slika 25: Akumulator za računanje prostornog momenta nultog reda .....	26
Slika 26: Kolo za računanje prostornih momenata .....	27
Slika 27: Kolo za izračunavanje težišta slike.....	27
Slika 28: Interfejs delitelja za računanje koordinata težišta slike .....	28
Slika 29: Kolo za računanje centralnih momenata slike.....	28
Slika 30: Kolo za računanje zakrivljenosti i elementa transformacione matrice.....	29
Slika 31: Kolo za računanje elementa M02 .....	30
Slika 32: Množac iz IP kataloga korišćen za računanje M02 .....	30
Slika 33: Kolo za računanje adresa piksela iz zakrivljene slike .....	31
Slika 34: Kolo za računanje nove vrednosti piksela .....	32

Slika 35: Blok šema sistema za ispravljanje slike .....	33
Slika 36: Kritična putanja u projektovanom IP jezgru.....	35
Slika 37: Ulazna i izlazna slika dimenzija 256 x 256 piksela.....	37

## **Jednačine**

Jednačina 1: Uopštena reprezentacija affine transformacije .....	2
Jednačina 2: Transformaciona matrica za zakrivljenje slike u opštem obliku.....	3
Jednačina 3: Koordinata X težišta slike .....	4
Jednačina 4: Koordinata Y težišta slike .....	4
Jednačina 5: Računanje mere zakrivljenosti slike .....	5

## **Formule**

Formula 1: Prostorni moment slike (i+j)-tog reda .....	4
Formula 2: Izračunavanje prostornog momenta nultog reda .....	4
Formula 3: Izračunavanje prostornog momenta prvog reda po x koordinati.....	4
Formula 4: Izračunavanje prostornog momenta prvog reda po y koordinati.....	4
Formula 5: Računanje centralnog momenta slike (p+q)-tog reda .....	5
Formula 6: Računanje intenziteta piksela pomoću bilinearne interpolacije .....	5
Formula 7: Računanje elementa R1 interpolacije.....	5
Formula 8: Računanje elementa R2 interpolacije.....	5
Formula 9: Skraćeni oblik interpolacije za računanje nove vrednosti piksela.....	6

## **Tabele**

Tabela 1: Registarska mapa IP-ja .....	18
Tabela 2: Iskorišćeni resursi za implementaciju sistema .....	34
Tabela 3: Iskorišćenost DSP-jeva .....	34
Tabela 4: Iskorišćenost BRAM blokova.....	34
Tabela 5: Kritične putanje u projektovanom jezgru .....	34

## **Listinzi**

Listing 1: Računanje prostornih momenata slike u C++ jeziku.....	25
Listing 2: Deklaracija MAC modula za sintezu koristeći DSP.....	26
Listing 3: Računanje centralnih momenata drugog reda .....	28
Listing 4: Deklaracija MAC modula za rad sa označenim brojevima .....	29
Listing 5: Zakrivljenost slike i element transformacione matrice .....	29
Listing 6: Računanje nove vrednosti piksela .....	30
Listing 7: C++ aplikacija za kontrolu procesa ispravljanja slika.....	36

# Sadržaj

1	Uvod .....	1
2	Opis algoritma za ispravljanje slike.....	2
2.1	Potrebne transformacije.....	2
2.2	Transformaciona matrica, zakrivljenost i momenti slike .....	3
2.2.1	Prostorni momenti i težište slike.....	4
2.2.2	Centralni momenti i zakrivljenost slike .....	5
2.2.3	Ispravljanje slike i interpolacija .....	5
3	Metodologije korišćene tokom projektovanja .....	7
3.1	ESL metodologija.....	8
3.1.1	Virtuelna platforma.....	8
3.2	RTL Metodologija.....	9
3.2.1	Particionisanje IP jezgra .....	10
3.3	Testiranje RTL-a koristeći mešovito UVM okruženje.....	11
4	Projektovanje hardvera .....	13
4.1	Interfejsi .....	13
4.2	Kontrolna jedinica .....	13
4.2.1	AXI LITE BRIDGE.....	14
4.2.2	Registarski blok .....	18
4.2.3	DSQW_CTRL.....	20
4.3	Podsistem za obradu podataka - Datapath.....	23
4.3.1	Memorijski kontroler i baferi.....	24
4.3.2	Računanje prostornih momenata slike .....	25
4.3.3	Računanje centralnih momenata slike .....	27
4.3.4	Računanje zakrivljenosti slike i elementa transformacione matrice.....	29
4.3.5	Računanje nove vrednosti piksela i interpolacija.....	30
5	Integracija i analiza iskorišćenih resursa .....	33
5.1	Rezultati vremenske analize.....	34
5.2	Programiranje ploče .....	36
6	Zaključak .....	38
7	Literatura .....	40



# 1 Uvod

Digitalna obrada slike je oblast računarstva koja sve češće pronalazi praktičnu primenu u svakodnevnicima. Sa sve bržim razvojem komercijalnih procesora, mogućnosti za razvoj aplikacija za manipulaciju slikom su sve veće, i sve češće iskorišćene. U kombinaciji sa algoritmima koji pripadaju grupi algoritama za mašinsko učenje, digitalna obrada slike pronalazi primenu aplikacijama poput sigurnosnih sistema u automobilske industriji, optičko prepoznavanje tekstualnih karaktera, građenje 3D modela pomoću skupa fotografija objekta i mnogih drugih.

Tema ovog rada je hardverska implementacija sistema za digitalnu obradu slike, i to specifično jezgra koje priprema sliku cifre ili broja za rad klasifikatora istih, na taj način što otklanja postojeću zakošenost karaktera sa slike.

Primena sistema za klasifikaciju cifara je najčešća u slučajevima kada je potrebno pretvoriti sadržaj slike u tekstualnu formu (npr automatsko popunjavanje uplatinca na osnovu skeniranih podataka, prepoznavanje registarskih tablica automobila, itd.). U slučaju prepoznavanja rukopisa, nije retkost da karakteri budu zakošeni ili zakrivljeni na neku stranu. Korišćenje ovakvih sistema pokazalo je da uspešnost klasifikatora u mnogome može zavisiti od orijentacije i zakošenosti samih karaktera, te se najčešće pre same klasifikacije vrši ispravljanje ovih nesavršenosti, ukoliko postoje.

Jedan ovakav sistem prolazi kroz nekoliko faza razvoja. U te faze spadaju analiza i particionisanje algoritma na hardverski i softverski deo, sistemsko projektovanje i određivanje potrebnih resursa za reprezentaciju podataka, testiranje i verifikacija kroz simulaciju, modelovanje na RTL nivou, integracija i implementacija sistema na razvojnoj ploči, kao i pisanje odgovarajućeg softvera. O svakom od ovih koraka biće reči u narednim poglavljima.

Centralni deo ovog rada obrađuje korišćenje RTL metodologije za hardversku implementaciju akceleratora za ispravljanje sadržaja slike.

U drugom poglavlju će detaljno biti objašnjena transformacija koja je korišćena za implementaciju ovog algoritma kao i sam algoritam. Treće poglavlje sadrži pregled tehnologija, metodologija i koraka koji su korišćeni i urađeni u procesu projektovanja sistema. Četvrto poglavlje obrađuje hardversku implementaciju jezgra za ispravljanje slike, što je i glavna tema ovog rada. Peto poglavlje analizira projektovani sistem – opisuje iskorišćene hardverske resurse, strukturu hardverskog dela sistema kao i odgovarajući softver. Šesto poglavlje je zaključak, gde je dat pregled svih koraka, diskutovana uspešnost projektovanja sistema, analizirane su greške koje su pronađene i predloženi koraci za unapređenje sistema.

## 2 Opis algoritma za ispravljanje slike

Motivacija za razvoj algoritma za ispravljanje slika crifara dolazi iz potreba sistema za klasifikaciju istih. U slučaju klasifikacije cifara sa slike rukopisa, vrlo je verovatno da će tekst biti zakošen. Empirijiski je utvrđeno da postojeći algoritmi za klasifikaciju imaju veći procenat uspešnosti za slike na kojima cifre nisu zakošene. Zato se kao pretkorak klasifikaciji najčešće vrši ispravljanje zakošene slike.

Verzija algoritma čija je hardverska implementacija opisana ovim dokumentom uvodi neka ograničenja:

- Širina i visina slike moraju biti identične (izražene u broju piksela)
- Osvetljenje jednog piksela predstavljeno je celim nenegativnim brojem i to tako da:
  - Bela boja (maksimalno osvetljenje) je predstavljeno brojem 255
  - Crna boja (nema osvetljenja) je predstavljena brojem 0
- Slika treba da predstavlja cifru (ili broj) bele boje na crnoj pozadini

### 2.1 Potrebne transformacije

Algoritam se zasniva na afinnoj transformaciji slike. Kao što je navedeno u [1], pod afinom transformacijom slike se podrazumeva metoda mapiranja izvorne slike na novu, pri čemu se sadržaj slike (odnosno sve tačke) zadržavaju, sve linije koje su paralelne na originalnoj slici ostaju paralelne i na novonastaloj slici. Najbitnija osobina ove transformacije jeste da sadržaj dobijene slike ostaje u istoj ravni kao i sadržaj izvorne slike. U klasu ovih transformacija spadaju neke geometrijske transformacije poput translacije, skaliranja, smicanja i rotacije. Kada bi bilo potrebno zakriviti originalnu sliku, to bi se moglo uraditi primenom transformacije za smicanje slike.

U stvarnosti, sadržaj slike nije uvek centriran, već je izmešten. Da bi sadržaj slike bio uspešno zakrivljen, osim primene afine transformacije potrebno je uključiti i izmeštenost sadržaja od centra slike u algoritam. U ovom slučaju se podrazumeva da su koordinate izražene celim brojevima. Afina transformacija svakom od piksela originalne slike dodeljuje nove koordinate u novoj slici, i to se može predstaviti na sledeći način:

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} M_{00} & M_{01} & M_{02} \\ M_{10} & M_{11} & M_{12} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

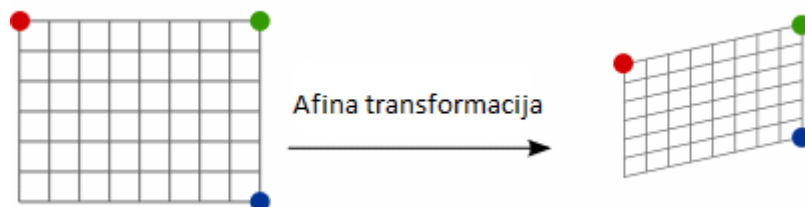
*Jednačina 1: Uopštena reprezentacija afine transformacije*

U navedenoj jednačini,  $\begin{bmatrix} x_p \\ y_p \end{bmatrix}$  predstavlja nove koordinate piksela sa originalnih koordinata  $\begin{bmatrix} x \\ y \end{bmatrix}$ . U procesu ispravljanja slike, sadržaj piksela sa novih koordinata je potrebno dodeliti pikselu sa originalnih koordinata.

Izmeštenost sadržaja slike od centra će uticati na to da elementi transformacione matrice (matrica iz *Jednačine 1* čiji su elementi označeni velikim slovom **M**) ne budu celi brojevi. Samim tim, ni izračunate nove koordinate neće biti celi brojevi. Budući da na slici ne postoji piksel sa takvim koordinatama, u algoritam je potrebno uvesti proces interpolacije, kojim će biti izračunata nova vrednost osvetljenosti piksela na osnovu osvetljenosti piksela iz okoline onog koji je predstavljen datim koordinatama.

## 2.2 Transformaciona matrica, zakrivljenost i momenti slike

Da bi slika bila zakrivljena u jednoj ravni, potrebno je izabrati tri tačke sa originalne slike, a zatim zadati željenu poziciju tih tačaka na zakrivljenoj slici, i izračunati transformacionu matricu koja će koristeći *Jednačinu 1* dati željene vrednosti koordinata za izabrane tačke.



Slika 1: Primer afine transformacije (preuzeto sa [2])

Za potrebne ovde opisanog algoritma, transformaciona matrica je unapred određena u opštem obliku, kako je navedeno u [3]:

$$M = \begin{bmatrix} 1 & skew & -0.5 * skew * img\_dim \\ 0 & 1 & 0 \end{bmatrix}$$

Jednačina 2: Transformaciona matrica za zakrivljenje slike u opštem obliku

U navedenoj jednačini, moguće je primetiti da su uvedene dve promenljive: *skew* i *img\_dim*. Prva predstavlja zakrivljenost slike, o čijem će izračunavanju biti reči kasnije u tekstu. Druga promenljiva predstavlja jednu od dimenzija slike (širinu ili visinu, tj. obe, uzevši u obzir da moraju biti identične) izraženu u broju piksela.

Zakrivljenost slike je predstavljena realnim brojem. Za njeno izračunavanje potrebno uvesti pojmove momenata slike. Ove veličine služe za izračunavanje težišta slike, odnosno mesta na slici oko koga je ravnomerno raspoređen sadržaj, a u odnosu na koje se računa zakrivljenost.

Dva najbitnija tipa momenata za implementaciju algoritma su prostorni i centralni momenti. Zakrivljenost se može izraziti kao odnos dva centralna momenta slike drugog reda. Svaki od ovih momenata je funkcija težišta slike, tačnije njegovih koordinata, a težište slike je funkcija prostornih momenata nultog i prvog reda. U daljem tekstu će biti objašnjeni redom prostorni momenti, težište slike, centralni momenti i sama zakrivljenost.

### 2.2.1 Prostorni momenti i težište slike

Uopšteno govoreći, moment slike se može definisati kao težinska suma intenziteta svih piksela slike, ili kao funkcija drugih momenata. Prostorni momenti  $(i+j)$ -tog reda se opisuje sledećom formulom:

$$m_{ij} = \sum_x \sum_y x^i y^j I(x, y)$$

*Formula 1: Prostorni moment slike  $(i+j)$ -tog reda*

Prostorni momenti koji su neophodni za pronalaženje težišta slike su:

- Moment nultog reda – označen sa  $m_{00}$
- Moment prvog reda po x koordinati – označen sa  $m_{10}$
- Moment prvog reda po y koordinati – označen sa  $m_{01}$

$$m_{00} = \sum_x \sum_y I(x, y)$$

*Formula 2: Izračunavanje prostornog momenta nultog reda*

$$m_{10} = \sum_x \sum_y x I(x, y)$$

*Formula 3: Izračunavanje prostornog momenta prvog reda po x koordinati*

$$m_{01} = \sum_x \sum_y y I(x, y)$$

*Formula 4: Izračunavanje prostornog momenta prvog reda po y koordinati*

Moment nultog reda predstavlja intenzitet slike – sumu intenziteta svih piksela. U slučaju binarne slike (gde je svaki piksel ili 1 ili 0), ovaj moment predstavlja površinu figure ili sadržaja slike. Ova tri momenta zajedno se mogu iskoristiti za pronalaženje težišta slike. Specifično, koordinate težišta slike  $(\bar{x}, \bar{y})$  je moguće izračunati na sledeći način:

$$\bar{x} = \frac{m_{10}}{m_{00}}$$

*Jednačina 3: Koordinata X težišta slike*

$$\bar{y} = \frac{m_{01}}{m_{00}}$$

*Jednačina 4: Koordinata Y težišta slike*

### 2.2.2 Centralni momenti i zakrivljenost slike

Za razliku od prostornih momenata, koji zavise od pozicije sadržaja na slici, centralni momenti zavise isključivo od ukupnog intenziteta slike. Ovo znači da ukoliko postoji više slika istih dimenzija i orijentacije jedne te iste cifre, identičnog oblika, samo drugačije pozicionirane na slici, centralni momenti tih slika uvek treba da budu isti. Ovakvi momenti su pogodni za računanje osobine zakrivljenosti sadržaja slike jer isključuju okolinu sadržaja.

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q I(x, y)$$

Formula 5: Računanje centralnog momenta slike (p+q)-tog reda

Može se zaključiti da su centralni momenti sada funkcija prostornih momenata, jer zavise od težišta slike koje je izračunato pomoću prostornih momenata. Dva prostorna momenta koji su pogodni za računanje zakrivljenosti slike jesu momenti drugog reda, i to momenti označeni sa  $\mu_{11}$  i  $\mu_{02}$ . Mera zakrivljenosti slike se definiše na sledeći način:

$$skew = \frac{\mu_{11}}{\mu_{02}}$$

Jednačina 5: Računanje mere zakrivljenosti slike

### 2.2.3 Ispravljanje slike i interpolacija

Kada je izračunata mera zakrivljenosti slike, potrebno je izračunati elemente transformacione matrice i primeniti *Jedančinu 1* nad svim pikselima slike kako bi slika bila ispravljena. Kako je ranije navedeno, koordinate izračunate na ovaj način neće uvek biti celi brojevi, i zato je potrebno uvesti metodu interpolacije kako bi intenzitet piksela bio preciznije izračunat. Ovde se koristi bilinearna interpolacija koja se definiše na sledeći način:

$$P = R2 + \left( \frac{y_p - y_1}{y_2 - y_1} \right) * (R1 - R2)$$

Formula 6: Računanje intenziteta piksela pomoću bilinearne interpolacije

$$R1 = I(x_1, y_1) + \left( \frac{x_p - x_1}{x_2 - x_1} \right) * (I(x_2, y_1) - I(x_1, y_1))$$

Formula 7: Računanje elementa R1 interpolacije

$$R2 = I(x_1, y_2) + \left( \frac{x_p - x_1}{x_2 - x_1} \right) * (I(x_2, y_2) - I(x_1, y_2))$$

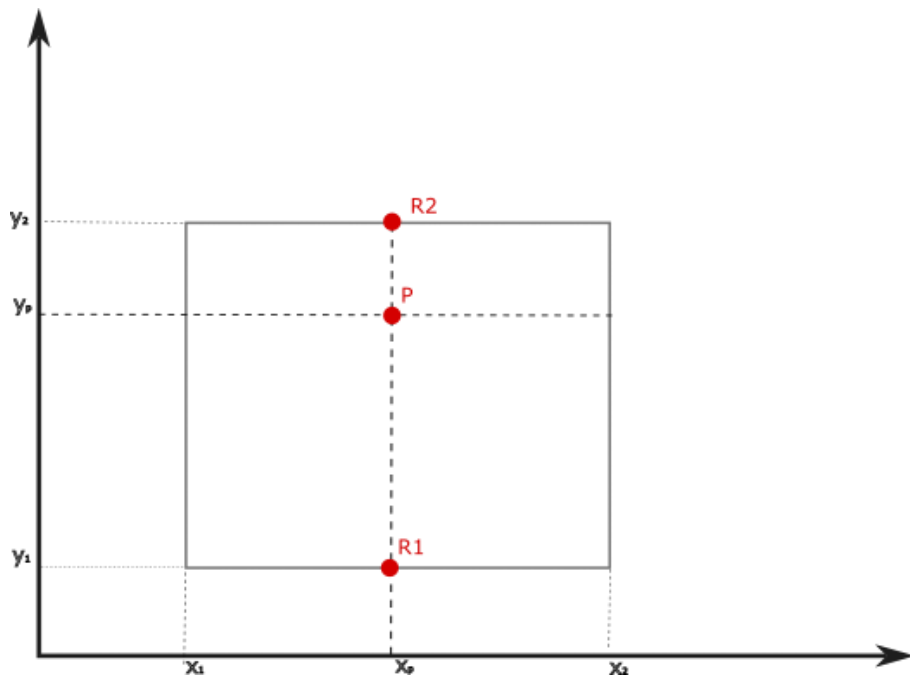
Formula 8: Računanje elementa R2 interpolacije

U prethodnim formulama promenljive  $x_p$  i  $y_p$  su elementi resultantnog vektora iz *Jednačine 1*. Promenljive  $x_1$  i  $x_2$  su okolina pozicije  $x_p$ , i to tako da je  $x_1$  celobrojni deo od  $x_p$ , a  $x_2$  za 1 veći od  $x_1$ . Isto pravilo važi i za promenljive  $y_p$ ,  $y_1$  i  $y_2$ .

Kada se u *Formulu 3* uvrste spomenute relacije i vrednosti promenljivih  $x_p$  i  $y_p$  zamene rezultatom *Jednačine 1* koja za transformacionu matricu koristi onu iz *Jednačine 2*, može se uočiti da je drugi sabirak iz *Formule 3* uvek nula, a  $y_p$  je isto što i samo  $y$ . Sada se nova vrednost piksela računa na sledeći način:

$$P = I(x_1, y_2) + (x_p - x_1) * (I(x_2, y_2) - I(x_1, y_2))$$

Formula 9: Skraćeni oblik interpolacije za računanje nove vrednosti piksela



Slika 2: Grafički prikaz bilinearne interpolacije

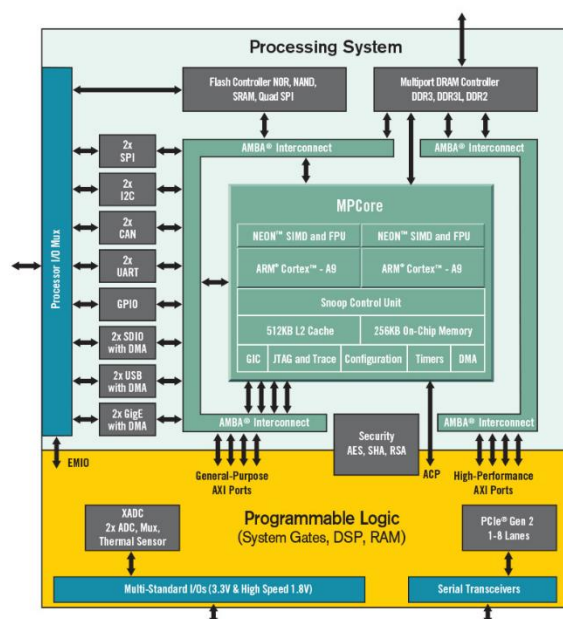
### 3 Metodologije korišćene tokom projektovanja

Proces projektovanja jednog složenog digitalnog sistema sastoji se iz nekoliko faza. Broj i vrsta faza variraju u odnosu na to koji je krajnji cilj ovog procesa. Ono što je zajedničko za sve sisteme jeste početni korak, a to je definisanje funkcionalnosti (ili algoritma) koju projektovani hardver treba da sadrži, kao i definisanje tehnologije u kojoj će krajnji sistem biti implementiran. Sistem koji je opisan u ovom radu je predodređen za implementaciju koristeći FPGA platformu.

FPGA (eng. *Field Programmable Gate Array*) čipovi spadaju u grupu integrisanih kola koja imaju mogućnost rekonfiguracije nakon fabrikacije. Gradivne jedinice ovakvih integrisanih kola su nizovi standardnih ćelija (kako je navedeno u [4]) kao i nizovi prekidačkih kola. Konfigurisanjem prekidačkih kola gde svaki od prekidačkih elemenata radi ili u režimu otvorenog ili u režimu kratkog spoja, moguće je na različite načine povezati ostale funkcionalne jedinice i na taj način dobiti specifičnu funkcionalnost.

Platforma izabrana za implementaciju sistema u ovom slučaju je razvojna ploča ZYBO Z7 koja na sebi ima ZYNQ 7010 SoC (eng. *System on a Chip*). Ovo je sistem koji se sastoji FPGA komponente i procesorske jedinice (detaljnije u [5]). Za komunikaciju ove dve komponente koristi se standardni AXI (eng. *Advanced eXtensible Interface*) protokol.

Kada je određena tehnologija i željena platforma za implementaciju celog sistema, moguće je nabrojati faze u postupku projektovanja sistema. Prva faza je sistemsko projektovanje koristeći ESL (eng. *Electronic System Level*) metodologiju i SystemC jezik. Drugi korak je projektovanje IP (eng. *Intellectual Property*) jezgra koje implementira algoritam u hardveru. Treći korak je testiranje ovog jezgra kroz simulacije. Četvrti korak je integracija jezgra u veći sistem, što podrazumeva konfigurisanje FPGA komponente tako da odgovara funkcionalnosti projektovanog jezgra, kao i pisanje odgovarajućeg softvera. Kao dodatni korak prirodno sledi testiranje celog sistema.



Slika 3: Struktura ZYNQ 7010 SoC-a, preuzeto sa [9]

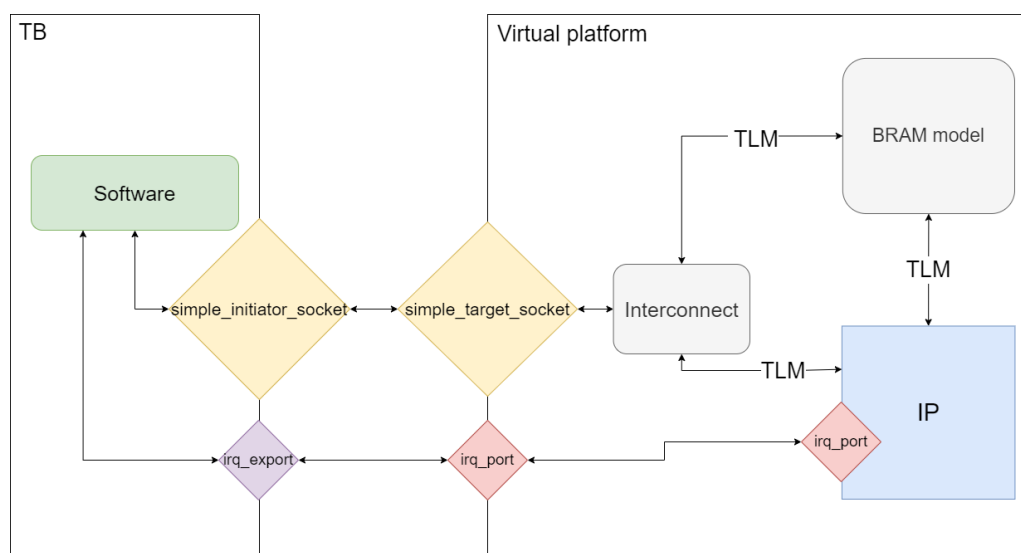
### 3.1 ESL metodologija

ESL metodologija podrazumeva postupak projektovanja nekog složenog digitalnog sistema, u kome je prvenstveni fokus na projektovanje na višem nivou apstrakcije [6]. Koristeći ovu metodologiju, može se lakše upravljati profinjnjem modela sistema koji se projektuje i sistematično se može pristupiti izradi modela na nižim nivoima apstrakcije, sve do samog RTL (eng. *Register Transfer Level*) modela i integracije softvera i hardvera.

Postoji nekoliko standardnih koraka koje je potrebno ispratiti kako bi sistem bio projektovan u skladu s metodologijom. Prvi korak je pisanje specifikacije kako prirodnim jezikom tako i modelovanjem u nekom programskom jeziku, čime se dobija tzv. izvršna specifikacija. Najčešće korišćeni jezici su *SystemC*, *SystemVerilog*, *MATLAB M-Code* i drugi. Zatim slede koraci pomoću kojih je moguće particionisati sistem na softver i hardver. Pre particionisanja se vrši analiza kako bi se imao uvid u to koji deo sistema je pogodan za implementaciju u hardveru, a koji deo u softveru. Analiza se takođe vrši i nakon particionisanja, kako bi se utvrdilo da nije došlo do prekoračenja nekih od početnih ograničenja. Na kraju se nalazi skup koraka koji podrazumeva implementaciju hardvera i softvera, kao i njihovu integraciju i testiranje.

#### 3.1.1 Virtuelna platforma

Prvi korak u projektovanju ovde opisanog sistema je projektovanje virtuelne platforme koristeći *SystemC* jezik. Ovakav softverski model sistema je korišćen za precizno utvrđivanje potrebnih hardverskih resursa za predstavljanje rezultata aritmetičkih operacija algoritma. Svaka od komponenti sistema je modelovana kao zasebna klasa, dok je za komunikaciju ovih komponenti korišćen TLM (eng. *Transaction Level Modeling*) standard. Komunikacija između procesorske jedinice i FPGA komponente na izabranoj platformi se vrši uglavnom koristeći memorijsko mapiranje, te je za projektovanje virtuelne platforme izabran TLM kao najpogodniji mehanizam za modelovanje ovog vida komunikacije.



Slika 4: Struktura virtuelne platforme

Slika 3 predstavlja strukturu virtuelne platforme projektovanje za reprezentaciju sistema za ispravljanje slika. Plavom bojom je označen IP koji je implementiran u hardveru. Sivom bojom označeni su interkonekt i BRAM gotovi hardverski IP-jevi koji



će biti korišćeni, a koji su ovde modelovani u SystemC jeziku. Softver i procesorska jedinica su ovde predstavljeni u TB (eng. *TestBench*) komponenti, i komuniciraju sa ostatkom sistema koristeći standardni interfejs (TLM) i liniju prekida. U hardveru, ova komunikacija je implementirana koristeći industrijski standard, AXI-Lite protokol.

### 3.2 RTL Metodologija

Prateći ESL metodologiju, nakon particionisanja sistema, potrebno je implementirati softver i hardver. Projektovanju hardvera je moguće pristupiti na više različitih načina. U ovom slučaju je za dizajn hardverskog dela izabrana RTL metodologija.

Ova metodologija definiše nivo apstrakcije na kome je sistem opisan koristeći neki od jezika za opis hardvera, tzv. HDL (eng. *Hardware Description Language*). Projektovanju složenih sistema se pristupa na taj način što se sistem rekurzivno deli na manje celine kojima se može lakše upravljati. Ovakav pristup se naziva *hijerarhijski dizajn*. Prednosti ovakvog pristupa projektovanju su olakšano upravljanje složenošću sistema i ponovno korišćenje nekih od podsistema [7]. Jednom kada je određena podela dizajna, svakom od podsistema se može pristupiti ponaosob, i to u paraleli, na taj način što će na svakom podsystemu raditi po jedan tim inženjera. Osim toga, ukoliko je podela dobro odrađena, moguće je ubrzati proces projektovanja celog sistema na više načina:

- Ukoliko postoje delovi sistema koji su isti ili slični po funkcionalnosti, moguće je projektovati jedinstveni podsystem koji će više puta biti iskorišćen u većem sistemu
- Neke gotove standardne komponente je moguće unapred uvesti u sistem, i time uštedeti vreme za razvoj i verifikaciju tih podsistema

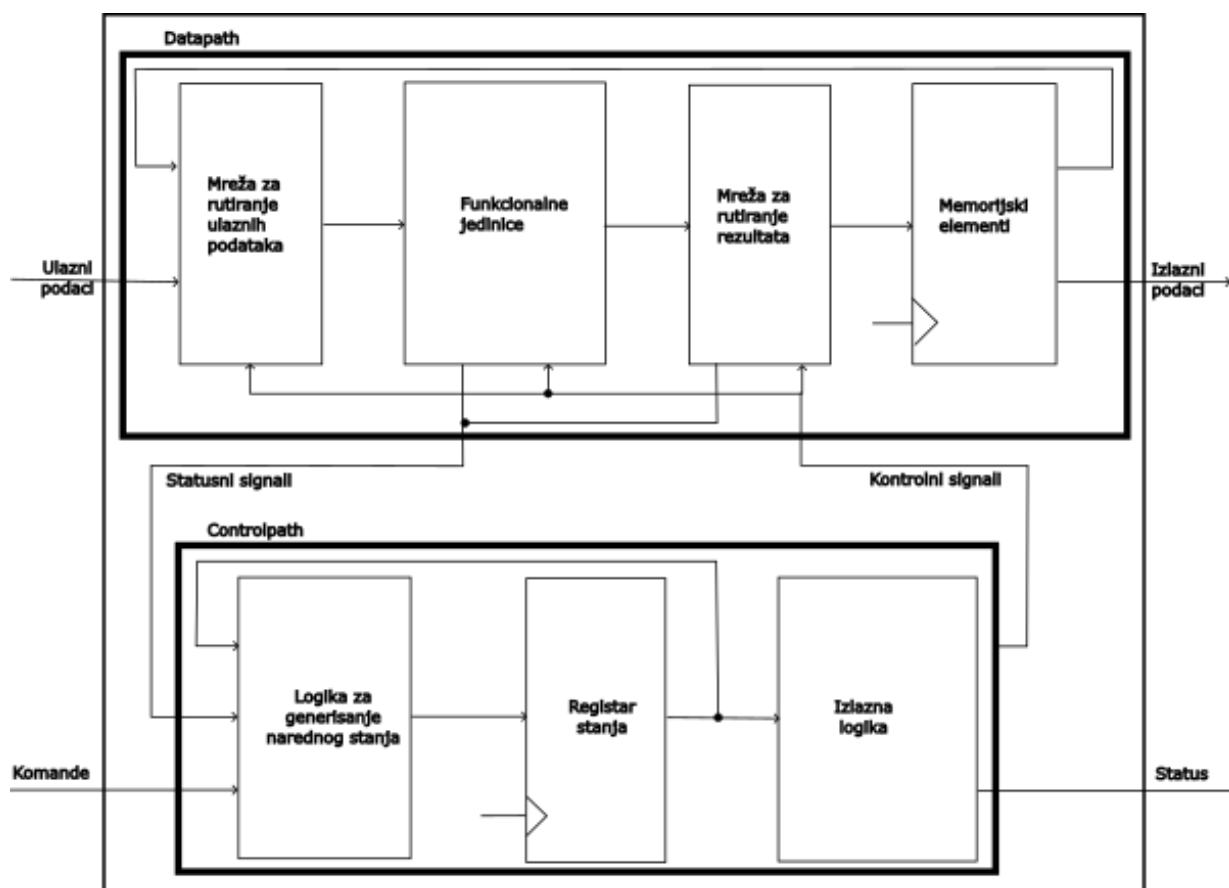
Pomenuti nivo apstrakcije koji RTL metodologija definiše podrazumeva da su osnovni gradivni blokovi jedinice sačinjene od jednostavnih logičkih kapija i registara [8]. U klasu ovih komponenti spadaju funkcionalne jedinice poput sabirača, množača, komparatora, sami registri kao i neke komponente za rutiranje podataka poput multipleksa. Takođe, tu se mogu uvrstiti i memorijski elementi poput FIFO (eng. *First In First Out*) bafera.

Kako se navodi u [8], *Register-Transfer* je originalno termin koji označava metodologiju u kojoj je sistem opisan po tome na koji način su podaci obrađeni i preneseni između registara. Ovde su osnovni gradivni blokovi jedinice srednje složenosti, stoga se koristi termin *Register Transfer nivo apstrakcije* (eng. *Register Transfer Level*).

Podela dizajna na manje jedinice se još naziva i *particionisanje dizajna*. Sa stanovišta sinteze projektovanog sistema, mogu se razlikovati *fizičko particionisanje* i *logičko particionisanje*. Prvi tip jasno određuje na koji način je sistem podeljen za proces sinteze, dok logičko particionisanje određuju inženjeri, i kao takvo uglavnom raščlanjuje sistem na veći broj podjedinica nego fizičko particionisanje. Logičko particionisanje je moguće primeniti na različitim nivoima hijerarhije sistema. U ovom radu je fokus na particionisanje jednog IP jezgra[7].

### 3.2.1 Particionisanje IP jezgra

Moguće je izdvojiti dve najveće celine u svakom IP jezgru. Prva je podsistem za obradu podataka (eng. *Datapath*), a druga je kontrolna jedinica (eng. *Controlpath*). *Datapath* se sastoji od elemenata za čuvanje podataka poput registara i bafera, zatim funkcionalnih jedinica za izvršavanje aritmetičko-logičkih operacija poput sabirača, množača, pomeračkih registara, i mreže za rutiranje ulaznih i izlaznih podataka, kao i međurezultata, koja se najčešće sastoji od multipleksera. Komunicira sa kontrolnom jedinicom preko statusnih signala koje generiše i kontrolnih signala koje dobija od kontrolne jedinice. *Controlpath* je podsistem koji treba da osigura tačan redosled osnovnih operacija nad podacima, kao i da upravlja statusnim signalima pomoću kojih komunicira sa ostatkom sistema u kome se samo jezgro nalazi. Najčešće se sastoji od jedne ili više mašina stanja, tj. FSM-ova (eng. *Finite State Machine*). Elementi od kojih se sastoji su registar stanja, logika za izračunavanje sledećeg stanja i logika za generisanje izlaznih signala.



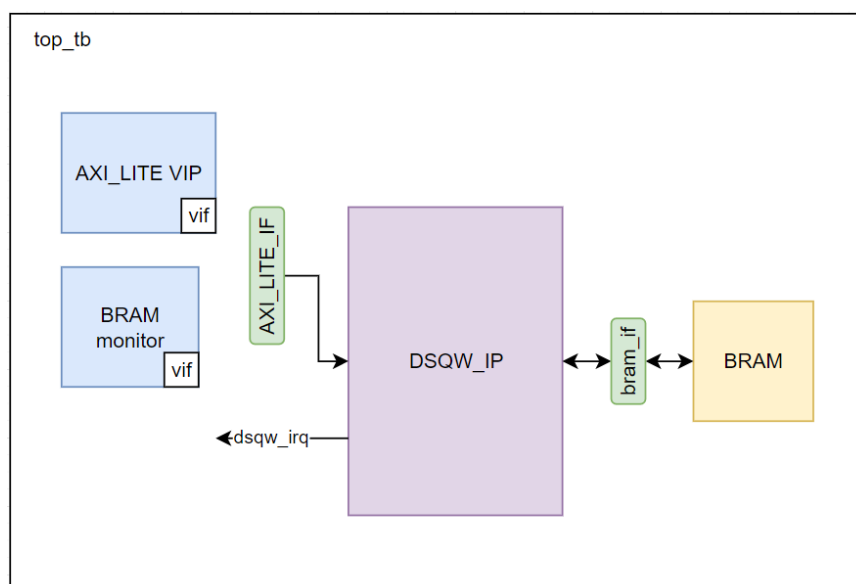
Slika 5: Particionisanje IP jezgra na Controlpath i Datapath[7]

### 3.3 Testiranje RTL-a koristeći mešovito UVM okruženje

Prilikom razvoja RTL modela sistema, potrebno je utvrditi ispravnost projektovane funkcionalnosti. Ovaj model je pogodan za testiranje kroz simulaciju. Kako je za ovaj sistem najbitnije da u dovoljnoj meri ispravi sliku, provera rada sistema je urađena u dva nivoa:

- Razvijeno je test okruženje koje koristi principe UVM (eng. *Universal Verification Methodology*) metodologije
- Rezultantna slika je iscrtana i vizuelno proverena (korisnik upoređuje originalnu i obrađenu sliku)

Za pokretanje bihevioralne simulacije korišćen je alat *Vivado*, kompanije *Xilinx*. Osnovu simulacije čine testbenč (eng. *Testbench*) i UVM test koji diktira scenario. Testbenč komponenta u sebi instancira projektovani IP, zatim BRAM komponentu koja je iskorišćeni gotovi IP, kao i dve verifikacione komponente napisane u *SystemVerilog* jeziku koristeći UVM biblioteku.



Slika 6: Simulaciono okruženje za testiranje RTL modela

Osim spomenutih komponenti, u testbenču se nalaze i dva interfejsa (eng. *Interface*), takođe napisana u *SystemVerilog* jeziku. Kako je ranije spomenuto, ovde nije ispraćen tok verifikacije koji nalaže UVM metodologija, već su samo neki koncepti koje ona nudi iskorišćeni za projektovanje sledećih komponenti:

- **AXI\_LITE\_VIP**
  - Ova komponenta služi za stvaranje stimulusa na AXI interfejsu IP-ja, kako bi se pristupilo registrima. Sastoji se od drajvera i sekvencera i sadrži pokazivač na AXI interfejs. Transakcije koje se izvršavaju na ovom interfejsu su kontrolisane iz testa.
- **BRAM monitor**
  - Pasivna komponenta koja nadgleda interfejs između IP-ja i BRAM bloka. Njen zadatak je da prepozna svaki upis u memoriju i skladišti te podatke lokalno. Na kraju simulacije ispisuje sadržaj memorije u izlazni fajl, na osnovu koga može biti iscrtana slika.

Za svaku simulaciju, ulazna slika je unapred učitana u BRAM memoriju koristeći *koeficijent fajl*, koji opisuje sadržaj memorije. Struktura testa je sledeća:

- Konfigurisati IP kroz upise u registre koristeći AXI\_LITE VIP komponentu
- Sačekati na signal prekida iz IP-ja
- Iščitati statusne registre i proveriti njihovu vrednost
- Očistiti prekide
- Ispisati sadržaj BRAM memorije u izlazni fajl

Po završetku simulacije, dostupan je fajl koji sadrži vrednosti osvetljenja svih piksela izlazne slike. Zatim je taj fajl prosleđen *python* skripti koja iscrtava izlaznu sliku.



*Slika 7: Ulazna i izlazna slika za simulaciju RTL modela*

## 4 Projektovanje hardvera

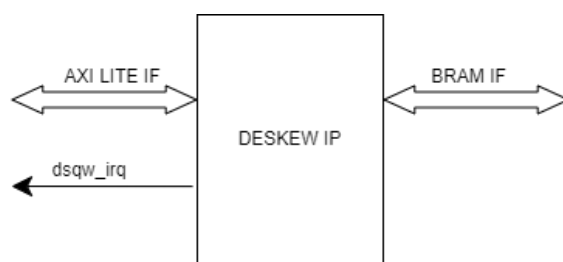
Za projektovanje jezgra za otkrivanje slika korišćena je prethodno pomenuta RTL metodologija. Sistem je podeljen na manje celine, od kojih su najveće dve *Controlpath* i *Datapath*. Jezgro treba da komunicira sa procesorskom jedinicom, sa jedne strane, koja ima pristup kontrolnim i statusnim registrima IP-ja, i sa memorijom sa druge strane. U narednim poglavljima biće opisani interfejsi jezgra, registarska mapa, kontrolna jedinica kao i svaka od funkcionalnih podjedinica zasebno.

### 4.1 Interfejsi

IP sa procesorskom jedinicom komunicira putem AXI-Lite interfejsa i linije prekida. Ovaj interfejs jezgra sadrži samo podskup neophodnih signala za transakcije za pristup registrima. Registri su širine 32 bita i ima ih ukupno 5, svaki poravnat na adresu od 4 bajta, pa je zato širina adrese registara 8 bita.

Linija prekida je signal širine jednog bita i povezana je direktno na kontroler prekida procesorske jedinice.

Interfejs ka memoriji je projektovan tako da odgovara BRAM IP-ju koji je dostupan u alatu *Vivado*. Podaci su širine 8 bita, a adresna magistrala širine 17 bita. Za svaki od pomenutih interfejsa projektovan je zaseban kontroler, i svi će biti detaljnije opisani u narednim poglavljima.



Slika 8: Interfejsi IP-ja

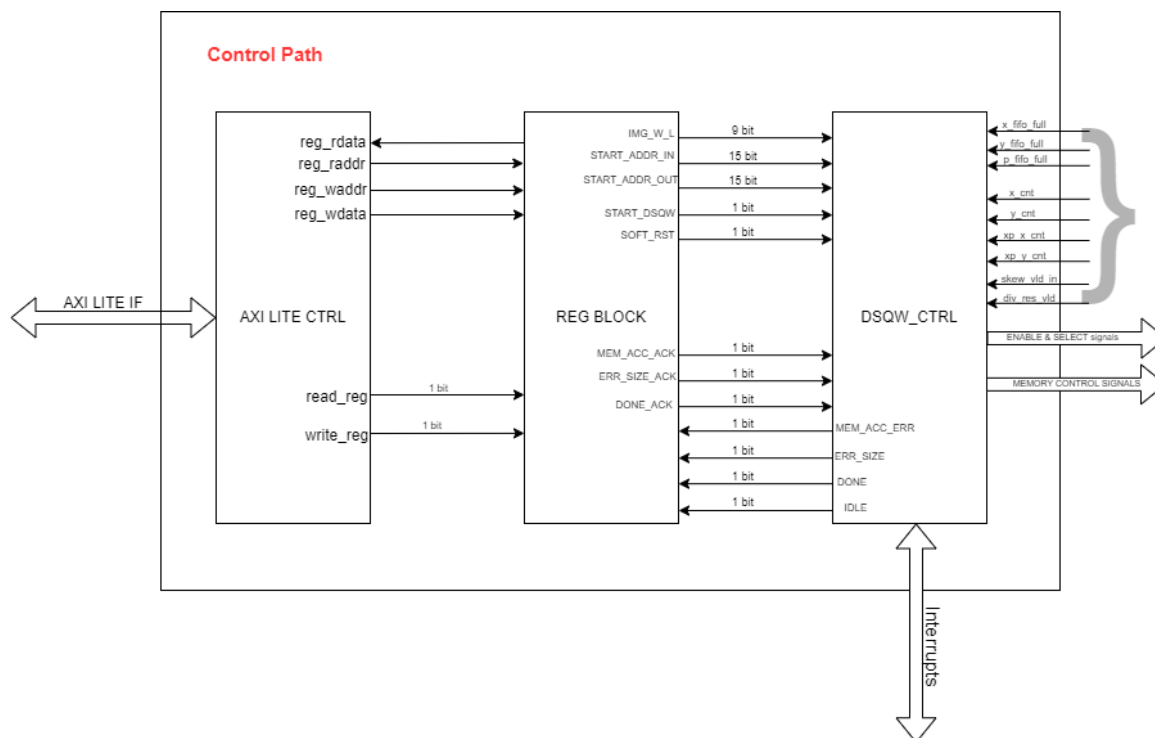
### 4.2 Kontrolna jedinica

Kako je ranije objašnjeno, kontrolna jedinica ima nekoliko zadataka:

- upravlja operacijama koje se dešavaju u podsistemu za obradu podataka
- generiše statusne signale
- prihvata komande (od procesorske jedinice)

Ovde je *Controlpath* podeljen na tri dela:

- AXI LITE BRIDGE
  - Komponenta koja implementira slejv (eng. *slave*) stranu protokola i pretvara AXI transakcije u operacije nad registrima. S obzirom na to da komande pristižu kao upisi u registar, ova komponenta je zadužena za prihvatanje komandi.
- Registarski blok
  - Sadrži logiku za upis u registre i čitanje iz njih. Zajedno sa AXI interfejs kontrolerom prihvata komande od procesorske jedinice
- *DSQW\_CTRL*
  - Deo kontrolne jedinice zadužen za proveru konfiguracije IP-ja, kontrolu linije prekida i generisanje kontrolnih signala za *Datapath*



Slika 9: Podela kontrolne jedinice i njeni interfejsi

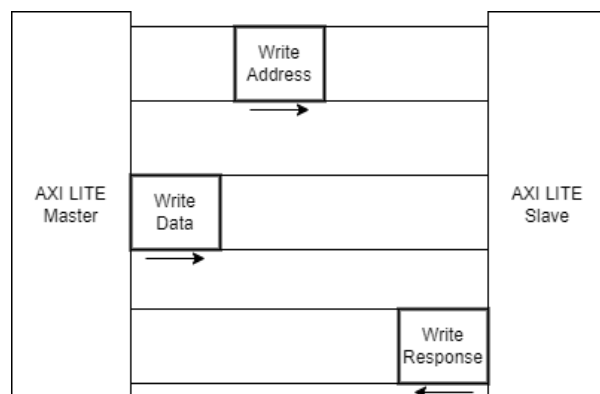
#### 4.2.1 AXI LITE BRIDGE

AXI Lite protokol pripada grupi **AMBA** (eng. *Advanced Microcontroller Bus Architecture*) protokola. Protokol funkcioniše po principu *master-slave*, i topologija u kojoj se može naći podrazumeva da postoje jedan ili više master uređaja, i jedan ili više slejv uređaja. Podaci mogu da teku u oba smera istovremeno, i komunikacija se odvija u nekoliko faza.

Interfejs je podeljen na pet kanala, i to tri za upis podataka i dva za čitanje podataka. Ti kanali su:

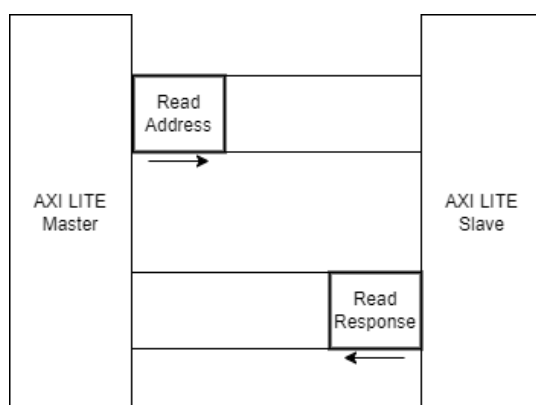
- Adresni kanal za upis podataka (eng. *Write Address Channel*)
- Kanal podataka za upis (eng. *Write Data Channel*)
- Kanal za odgovor pri upisu (eng. *Write Response Channel*)
- Adresni kanal za čitanje podataka (eng. *Read Address Channel*)
- Kanal podataka za čitanje (eng. *Read Data Channel*)

**Operacija upisa** se odvija u 3 faze. Master šalje zahtev za upis tako što postavlja adresu na adresni kanal, podatak na odgovarajući kanal, u bilo kom redosledu, i čeka na odgovor slejva da je prihvatio ove zahteve. Kada se upis završi, slejv inicira komunikaciju na kanalu za odgovor. Transakcija upisa se završava kada master prihvati odgovor.



Slika 10: AXI LITE transakcija upisa

**Operacija čitanja** se odvija u dve faze. Master započinje postavljanjem adrese na adresnom kanalu i čeka da slejv prihvati zahtev za čitanje. Nakon toga slejv postavlja pročitani podatak na kanal za čitanje podataka i signalizira masteru da je podatak spreman. Pored samog podatka, slejv šalje i odgovor koji označava status greške pri čitanju. Transakcija se završava kada master prihvati podatak i status od slejva.



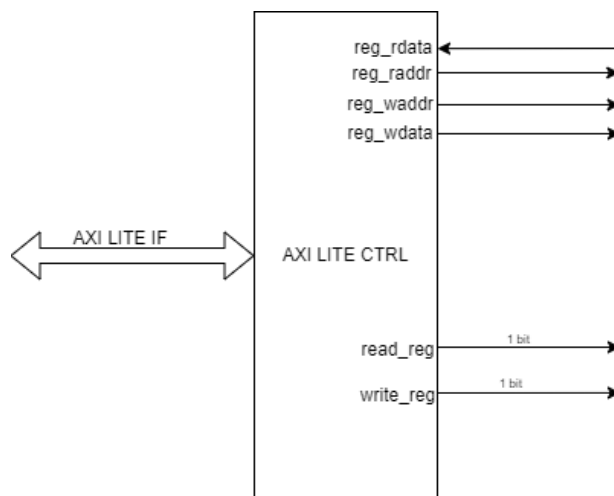
Slika 11: AXI LITE transakcija čitanja

Signali IP-ja koji čine ovaj interfejs su:

- **Globalni signali:**
  - ACLK – izvor takt signala;
  - ARESETN – globalni reset signal, aktivan na niskom logičkom nivou.
- **Adresni kanal operacije čitanja:**
  - ARADDR – adresa čitanja;
  - ARVALID – zahtev za čitanje postoji;
  - ARREADY – IP je spreman da primi zahtev.
- **Kanal čitanja podataka:**
  - RDATA – pročitani podatak (32-bitna vrednost);
  - RRESP – statusni signal koji predstavlja uspešnost operacije;
  - RVALID – IP je postavio podatak na magistralu;
  - RREADY – master je spreman da primi pročitani podatak.
- **Adresni kanal operacije upisa:**
  - AWADDR – adresa za upis podatka. Poravnata na 32-bita;
  - AWVALID – zahtev za upis je važeći;
  - AWREADY – IP je spreman da prihvati zahtev za upis.

- **Kanal upisa podataka:**
  - WDATA – podatak za upis (32-bitna vrednost);
  - WVALID – podatak je spreman za upis;
  - WREADY – IP je spreman da primi podatak.
- **Kanal za odgovor na upis podatka:**
  - BRESP – odgovor na upis. Ukazuje na prisutnost greške;
  - BVALID – IP je postavio odgovor na magistralu;
  - BREADY – inicijator je spreman da primi odgovor.

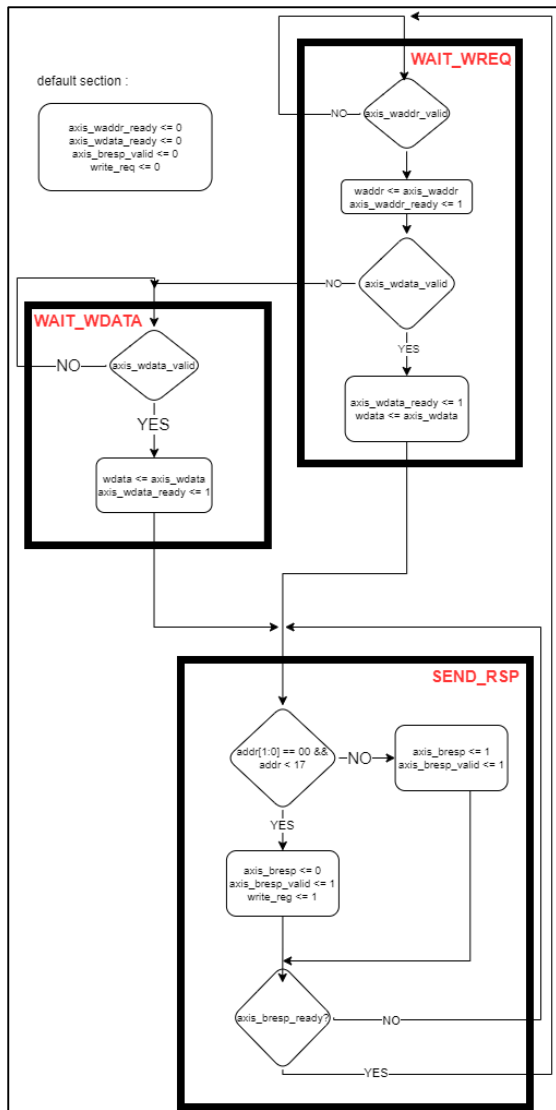
Kontroler je implementiran koristeći dve mašine stanja: jednu za operacije upisa, a drugu za operacije čitanja. Ukoliko master proba da pristupi adresi koja nije poravnata na 4 bajta ili adresi koja je van adresnog opsega, slejv u odgovoru postavlja statusni signal na vrednost koja predstavlja grešku. Tada je pročitani podatak nevalidan i ne odgovara nijednom od registara, a upisani podatak nije prosleđen u registre.



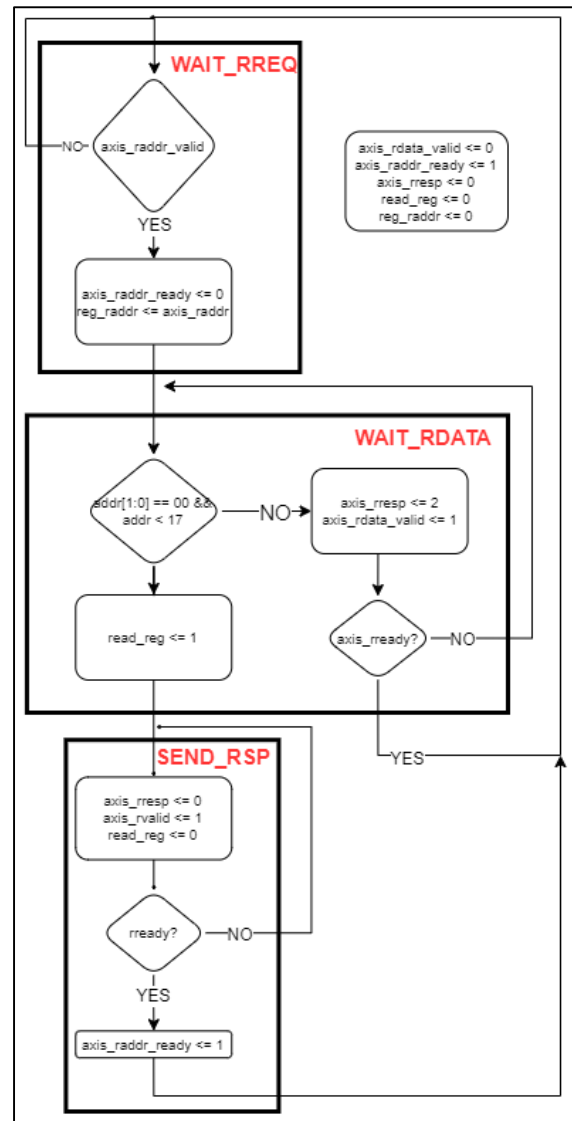
Slika 12: Šematski prikaz AXI LITE kontrolera

Osim signala koji pripadaju AXI LITE interfejsu, kontroler ima signale za komunikaciju sa registarskim blokom. Signali za upis i čitanje su odvojeni, i operacija upisa se odvija u jednom ciklusu dok operacija čitanja traje dva ciklusa.





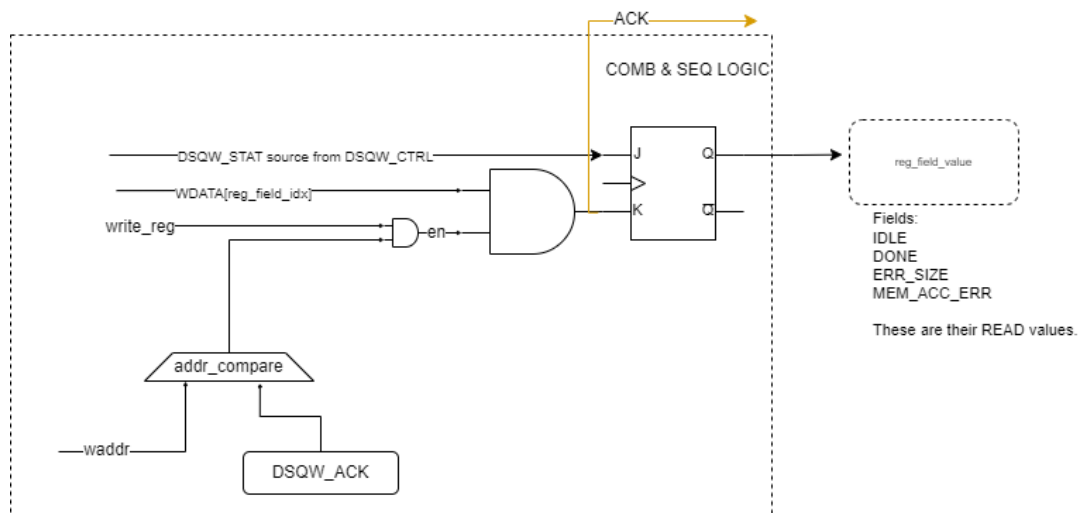
Slika 13: Mašina stanja kanala za upis AXI kontrolera



Slika 14: Mašina stanja kanala za čitanje AXI kontrolera



Registri **DSQW\_STAT** i **DSQW\_ACK** pripadaju grupi statusnih registara. Prvi sadrži četiri polja koja opisuju trenutno stanje sistema. Postavljanje visoke vrednosti bilo kojeg od tih polja će izazvati postavljanje visoke vrednosti na liniju prekida. Drugi registar je logička celina koju nije moguće pročitati, već služi za čišćenje polja **DSQW\_STAT** registra. Svako polje upisano u **ACK** registar izaziva čišćenje respektivnog polja u **STAT** registru. Na sledećoj slici predstavljena je logika za postavljanje i čišćenje svakog od polja statusa.



Slika 16: Logika za postavljanje i čišćenje statusnih signala

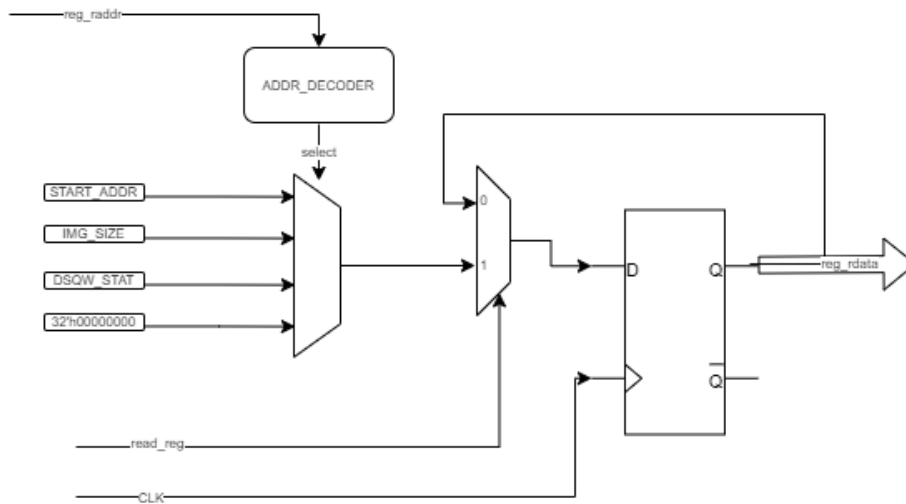
Slika 15 obeležava memorijski element kao JK-flipflop. Uzevši u obzir da je jezgro projektovano za FPGA platformu, u implementaciji je iskorišćena konstrukcija iste funkcionalnosti koja koristi D-flipflop. Takođe, treba naglasiti da polje **IDLE** ne može biti očišćeno niti utiče na signal prekida.

Poslednji registar je jedini kontrolni registar. **DSQW\_CFG** sadrži dva polja:

- **START\_DSQW**
  - Nakon čišćenja postojećih prekida i konfigurisanja IP-ja, korisnik upisuje '1' u ovo polje kako bi otpočeo proces ispravljanja slike
- **SOFT\_RST**
  - Ukoliko dođe do prekida i signal greške je postavljen, korisnik treba da upiše '1' u ovo polje kako bi izveo sistem iz stanja greške

Ova polja nije moguće pročitati (pročitana vrednost će uvek biti 0). Pri upisu u ova polja, umesto čuvanja vrednosti, generišu se pulsevi koji služe kao kontrolni signali za mašinu stanja koja se nalazi u **DSQW\_CTRL** modulu.

Registri čiju je vrednost moguće pročitati su **IMG\_DIM**, **START\_ADDR** i **DSQW\_STAT**. Čitanje druga dva registra uvek postavlja pročitani podatak na vrednost 0.



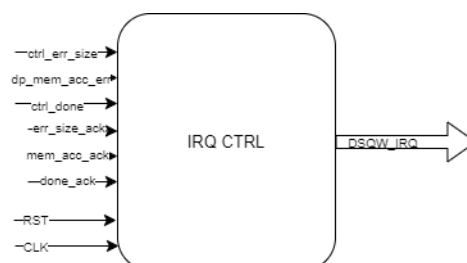
Slika 17: Logika za čitanje registara

### 4.2.3 DSQW\_CTRL

Ova komponenta predstavlja srž kontrolne jedinice. Sastoji se iz tri manje kopponente:

- **Kontroler prekida**
  - Prihvata statusne signale od ostalih modula i postavlja i čisti liniju prekida
- **Konfiguracioni FSM**
  - Pokreće ga operacija upisa u registar **DSQW\_CTRL**
  - Proverava konfiguraciju IP-ja i ukoliko je nespravna, zaustavlja rad IP-ja i čeka na *soft reset*. U suportnom signalizira *Kontrolnom FSM-u* da pokrene proces ispravljanja slike
- **Kontrolni FSM**
  - Kada dobije signal za početak od *Konfiguracionog FSM-a*, otpočinje ispravljanje slike tako što generiše kontrolne signale ka *Datapath-u*

*Kontroler prekida* dobija informaciju od *Konfiguracionog FSM-a* da li je došlo do greške u konfiguraciji i da li je obrada slike završena. Od *Datapath-a* dobija informaciju da li je došlo do prekoračenja opsega memorijskih bafera. Iz registarskog bloka dobija informacije o tome da li je potrebno očistiti liniju prekida.

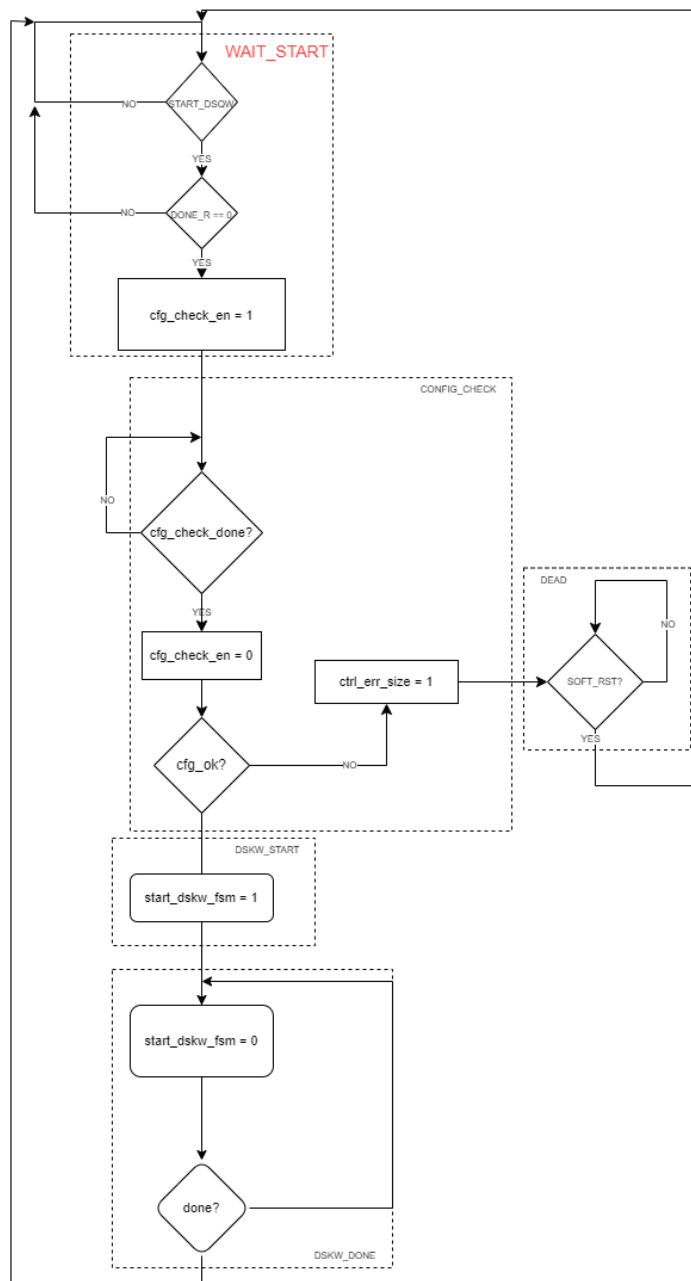


Slika 18: Interfejs kontrolera prekida

*Konfiguracioni FSM* dobija signal starta od registrarskog bloka, pokreće logiku za proveru ispravnosti konfiguracije. Provera konfiguracije podrazumeva:

- Da li su početne adrese ulaznog i izlaznog bafera identične
- Da li je dimenzija slike prevelika
- Da li je došlo do preklapanja bafera
- Da li bilo koji od bafera izlazi iz opsega raspoložive memorije

Ukoliko greška ne postoji, pokreće *Kontrolni FSM* i čeka na njegov signal da je slika obrađena. Tada signalizira *Kontroleru prekida* kraj operacije. Ukoliko greška postoji, ona je takođe signalizirana. Tada se očekuje od korisnika da izvrši *soft reset* upisom u DSQW\_CTRL registar.



Slika 19: Konfiguracioni FSM

*Kontrolni FSM* je konačni automat koji ima 59 stanja i 44 kontrolna signala koji su ulazi u *Datapath*. Pored toga ima 9 ulaznih statusnih signala koji dolaze takođe iz *Datapath-a*. Ovaj FSM prati tok obrade podataka i generiše kontrolne signale za funkcionalne jedinice. Zbog obima ovog modula, detalji implementacije neće biti izloženi u ovom dokumentu, već će biti dostupni u prilogu.

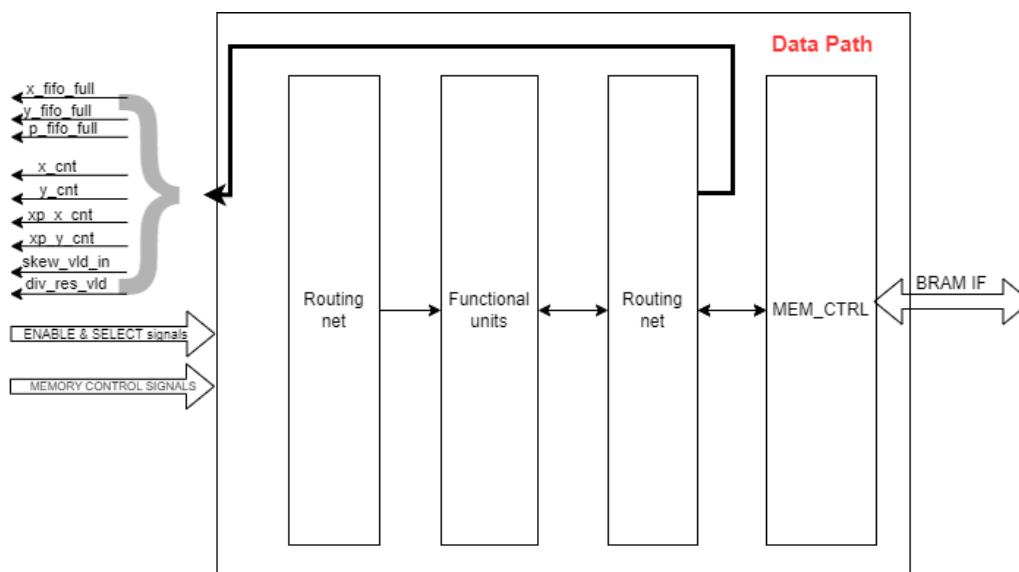
Algoritam se može podeliti na 3 veće celine, a samim tim se mogu te celine identifikovati u pomenutom FSM-u. Prva celina kontroliše funkcionalne jedinice za računanje prostornih momenata nultog i prvog reda. Sastoji se od 18 stanja.

Druga celina kontroliše jedinice za računanje centralnih momenata drugog reda, koordinata težišta slike, zakrivljenosti slike i potrebnog elementa transformacione matrice. Sastoji se od 26 stanja.

Treća celina je zadužena za izračunavanje nove vrednosti svakog od piksela slike. Samim tim kontroliše čitanje iz memorije i upis u nju, kao i računanje neophodnih koordinata piksela potrebnih za operaciju interpolacije. Takođe kontroliše sve ALU jedinice potrebne za izvršavanje procesa interpolacije. Sastoji se od 15 stanja.

### 4.3 Podsistem za obradu podataka - Datapath

Ovaj podsistem sadrži sve funkcionalne jedinice neophodne za izvršenje operacija predviđenih algoritmom. Prateći RTL metodologiju može se podeliti na skup funkcionalnih jedinica, mrežu za rutiranje podatka i mrežu za računanje izlaza. U ovom slučaju ulazni podaci su podaci pročitani iz memorije, a izlazni podaci su podaci koji su upisani u memoriju. Kontrolni signali dolaze iz *Controlpath* modula, a statusni signali su vrednosti unutrašnjih brojača, statusi bafera i indikatori završetka nekih aritmetičkih operacija. Uzevši to u obzir, struktura ovog podsistema je predstavljena na slici ispod.



Slika 20: Struktura i interfejs Datapath modula

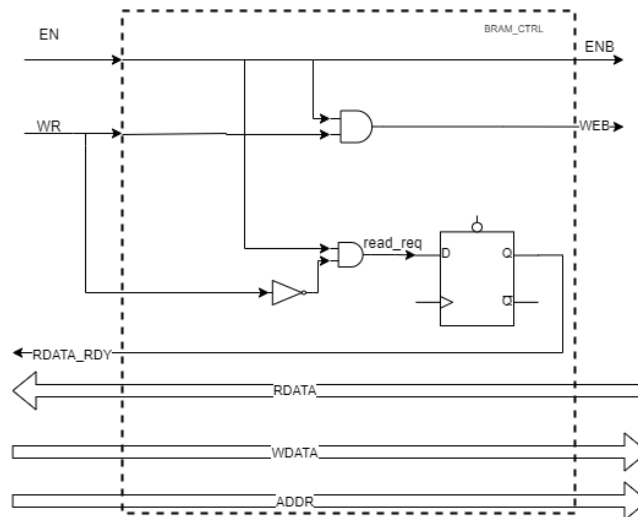
Algoritam je struktuiran tako da se može podeliti na nekoliko celina. Osim elemenata za pristup memoriji i čuvanje podataka iz memorije (memorijski kontroler i FIFO baferi), za svaku od celina algoritma predodređeni su zasebni hardverski resursi. Faze algoritma su:

- Računanje prostornih momenata
- Računanje centralnih momenata
  - Podrazumeva i računanje koordinata težišta slike
- Računanje zakrivljenosti slike
- Računanje potrebnog elementa transformacione matrice
- Računanje nove vrednosti svakog od piksela metodom interpolacije

Elementi korišćeni u svakoj od ovih faza biće zasebno objašnjeni u daljim poglavljima. Za implementaciju funkcionalnih jedinica korišćena je kombinacija gotovih IP-jeva koji su dostupni u alatu *Vivado*, zatim modula koji su napisani u Verilog HDL-u, sa instrukcijama kompajleru kako bi njihova funkcionalnost bila imlementirana koristeći postojeće DSP-jeve (eng. *Digital Signal Processor*), kao i moduli posebne namene opisani takođe u Verilog jeziku.

### 4.3.1 Memorijski kontroler i baferi

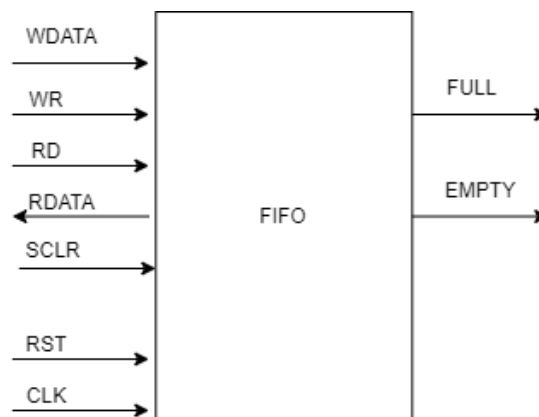
Komponenta koja se koristi u većini faza izvršavanja algoritma je memorijski kontroler. Služi za pristup BRAM memoriji kako bi se iz nje učitali pikseli ulazne slike, kako i u nju upisali pikseli izlazne slike. Dobija komande od *Controlpath* jedinice za operacije čitanja i upisa, pretvara ih u operacije na BRAM interfejsu, i za operacije čitanja označava kada je pročitani podatak dostupan ostatku sistema. Struktura i interfejs ovog modula predstavljeni su na slici ispod.



Slika 21: Struktura i interfejs memorijskog kontrolera

Gledano sa strane *Datapath* modula, operacija upisa se izvršava u jednom takt ciklusu, dok se operacija čitanja izvršava u dva ciklusa taktnog signala. Bitno je napomenuti da ovakav kontroler očekuje da je podatak pročitao iz memorije dostupan u ciklusu koji sledi nakon zahteva za čitanje. U suprotnom, kontroler neće ispravno čitati podatke, jer signal koji označava validnost pročitanoog podatka neće biti sinhronizovan sa dostupnim podatkom.

Za računanje nekih varijabli je neophodno znati vrednost piksela kao i koordinate na kojima se taj piksel nalazi u ulaznoj slici. Koordinate su poznate nekoliko ciklusa pre nego što je podatak sa tih koordinata pročitao, te su za sinhronizaciju korišćene tri instance FIFO bafera. Po jedna za čuvanje koordinata, i jedna za čuvanje pročitanih piksela. Sve tri veličine su izražene neoznačenim brojevima širine 8 bita.



Slika 22: Blok šema FIFO bafera



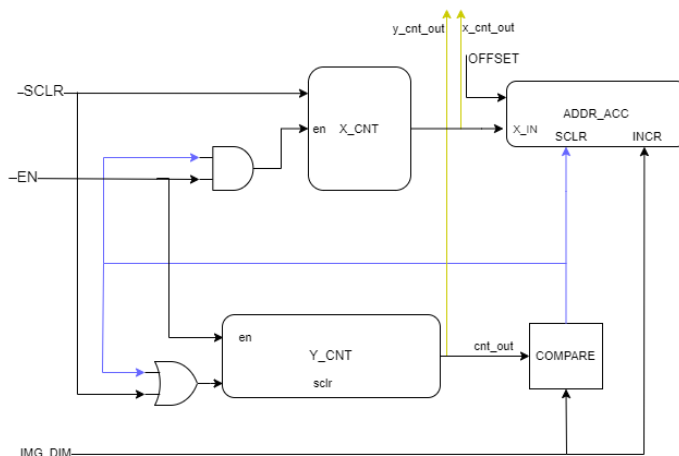
### 4.3.2 Računanje prostornih momenata slike

Prvi korak u algoritmu za ispravljanje slike jeste računanje prostornih momenata nultog i prvog reda. Kako je objašnjeno u drugom poglavlju ovog dokumenta, potrebno je iterirati kroz sve koordinate slike i izračunati težinsku sumu svih koordinata. Softverska implementacija ovakvog algoritma je ostvarena koristeći dve ugnježdene petlje, i to na sledeći način:

```
for ( x=0; x<a_img_dimension; x++)
  for( y = 0; y<a_img_dimension; y++)
  {
    m00 += a_image_vector[x+y*a_img_dimension];
    m10 += a_image_vector[x+y*a_img_dimension]*x;
    m01 += a_image_vector[x+y*a_img_dimension]*y;
  }
```

*Listing 1: Računanje prostornih momenata slike u C++ jeziku*

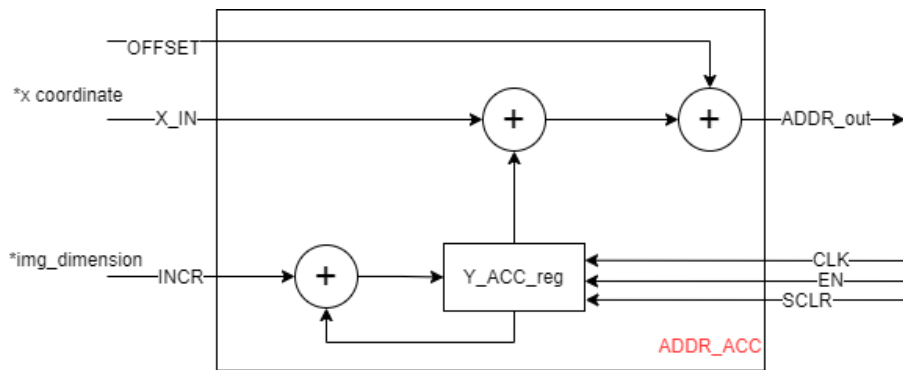
Iz Listinga 1 je moguće videti da se pikseli čitaju iz memorije sa adresnim skokovima koji su jednaki dimenziji slike u broju piksela. Ovde treba uzeti u obzir da se ulazna slika može nalaziti na bilo kojoj adresi u memoriji. Za svrhe izračunavanje nove adrese u svakom ciklusu projektovana je komponenta zvana *Adresni generator* koja koristi akumulator umesto množača kako bi postigla isti efekat.



*Slika 23: Adresni generator*

Adresni generator se sastoji iz dva brojača, koji oponašaju istoimene promenljive iz Listinga 1, komparatora koji označava kraj iteracije petlje koja iterira po  $y$  koordinati i komponente *Adresni akumulator*. Izlaz komparatora služi kao signal za čišćenje brojača  $y$  koordinate kao i signal dozvole brojanja brojača  $x$  koordinate. Izlaz komparatora je visok samo u trenutku kada  $y$  dostigne vrednost dimenzije slike.

**Adresni akumulator** izračunava novu adresu u svakom ciklusu u skokovima od po **IMG\_DIM** memorijskih lokacija. Ovaj podatak dolazi direktno iz registarskog bloka. Početna adresa za svaku iteraciju spoljne petlje je zbir početne adrese ulazne slike i trenutne vrednosti koordinate **x**. Izlaz iz komparatora čisti akumulacioni registar.



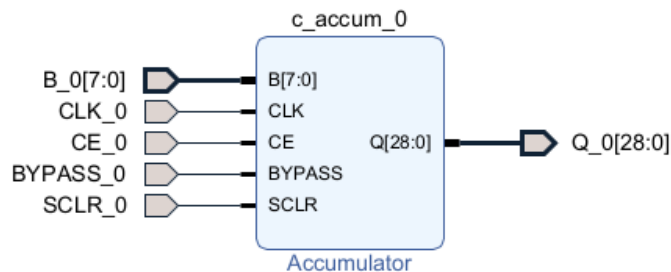
Slika 24: Struktura adresnog akumulatora

Treba primetiti da je ulaz pod nazivom **OFFSET** zapravo početna adresa ulazne slike.

Za računanje prostornih momenata potrebno je u jednom trenutku znati vrednosti koordinata na kojima se piksel nalazi kao i intenzitet tog piksela. Kao što je napomenuto u prethodnim poglavljima, operacija čitanja traje dva ciklusa, pa je zato potrebno iskoristiti FIFO bafere opisane u prethodnom poglavlju.

Kada su svi potrebni podaci dostupni, potrebno je akumulirati sve vrednosti kako je opisano u Listingu 1. Korišćene su dve vrste akumulatora. Jedna u sebi ima samo sabirač i registar, a druga vrsta u sebi ima i množač. Ovaj drugi se još naziva MAC (eng. *Multiply And Accumulate*).

Moment nultog reda je akumuliran koristeći gotov IP iz kataloga IP-jeva koje nudi alat *Vivado*. U pitanju je jednostavni akumulator koji radi sa neoznačenim brojevima. Ulaz je širine 8 bita, a izlazni podatak je širine 29 bita.

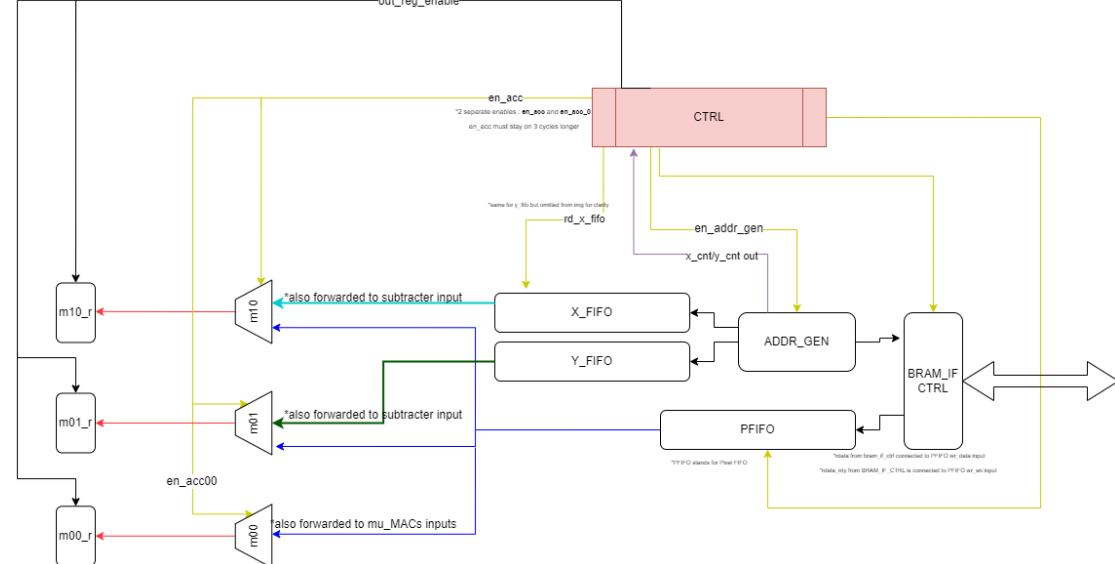


Slika 25: Akumulator za računanje prostornog momenta nultog reda

Momenti prvog reda računati su koristeći MAC module koji implementirani u Verilog-u. Sadrže direktivu za sintezu u kojoj se zahteva da budu implementirani koristeći dostupne DSP-jeve.

```
(* use_dsp = "yes" *) module mult_accum
#(parameter A_IN_WIDTH = 8,
parameter B_IN_WIDTH = 8,
parameter ACC_OUT_WIDTH = 29)
(
input [A_IN_WIDTH-1 : 0] dataa,
input [B_IN_WIDTH-1 : 0] datab,
input clk, aclr, clken, sload,
output reg [ACC_OUT_WIDTH-1 : 0] mac_out
);
```

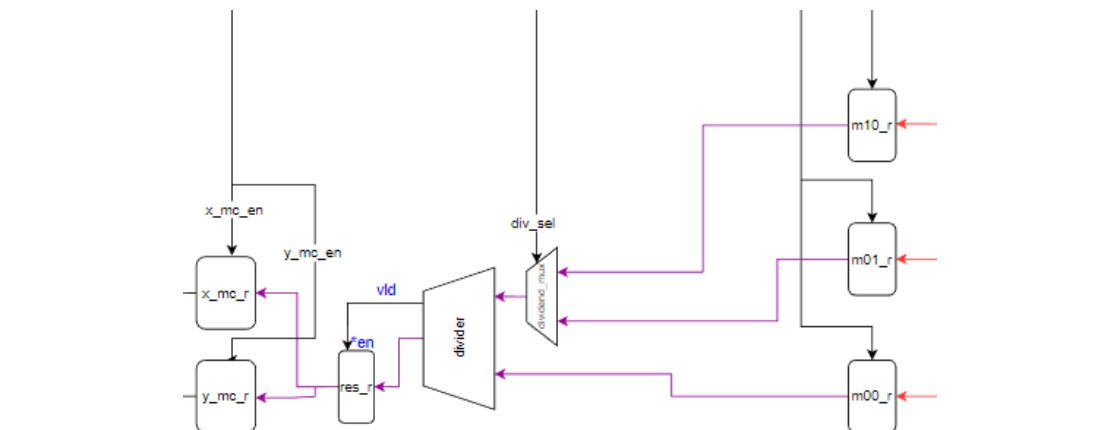
Listing 2: Deklaracija MAC modula za sintezu koristeći DSP



Slika 26: Kolo za računanje prostornih momenata

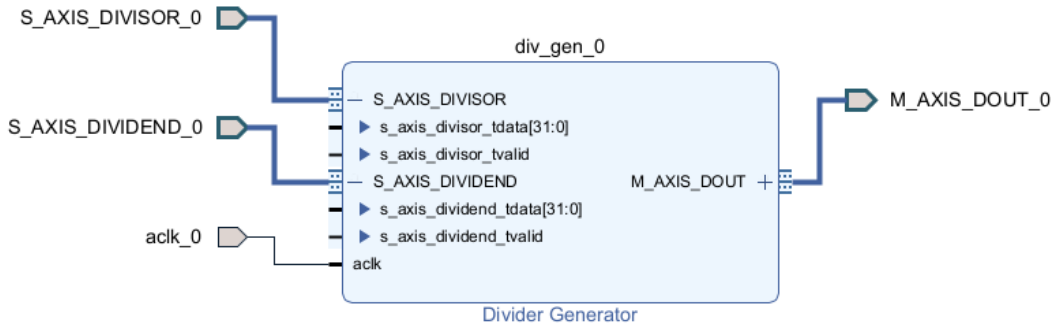
### 4.3.3 Računanje centralnih momenata slike

Prateći *Formulu 5*, neophodno je izračunati centralne momente. Pre toga je potrebno izračunati koordinate težišta slike, a što je moguće uraditi koristeći *Jednačine 3 i 4*. Za reprezentaciju ovih vrednosti izabrani su brojevi sa fiksnom tačkom u formatu 9 bita za celobrojni deo i 3 bita za razlomljeni deo, ukupno 12 bita.



Slika 27: Kolo za izračunavanje težišta slike

U kolu sa Slike 27 je predstavljen delitelj koji je već postojeći IP iz kataloga IP-jeva alata *Vivado*. Delitelj podržava protočnu obradu i za računanje jedne operacije potrebno je 36 ciklusa takta. Za računanje koordinata započete su dve operacije deljenja u dva uzastopna ciklusa. Ukupno trajanje računanja obe koordinate traje 37 ciklusa takt signala.



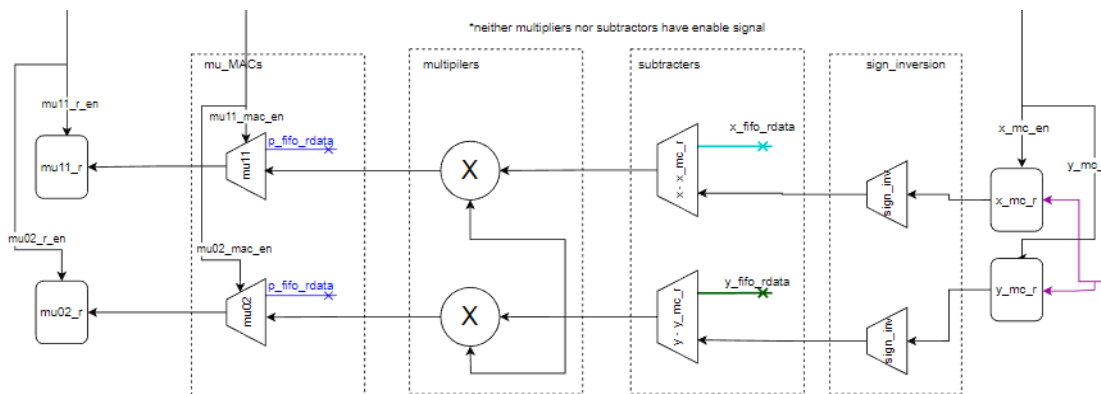
Slika 28: Interfejs delitelja za računanje koordinata težišta slike

Kada je težište izračunato, vrednosti koordinata se mogu uvrstiti u već pomenutu *Formulu 5*. Za potrebe algoritma se računaju dva centralna momenta drugog reda. U sledećem listingu je predstavljena implementacija ovog dela algoritma u C++ jeziku.

```
for (int x=0; x<a_img_dimension; x++)
  for(int y = 0; y<a_img_dimension; y++)
  {
    mu02 += (a_image_vector[x+y*a_img_dimension]*pow((y-y_mc),2));
    mu11 += (a_image_vector[x+y*a_img_dimension]*(x-x_mc)*(y-y_mc));
  }
```

Listing 3: Računanje centralnih momenata drugog reda

Ugnježđene petlje iz Listinga 3 odgovaraju onima iz Listinga 1, zato je i za ovu namenu korišćena ista struktura sačinjena od *Adresnog generatora* i tri FIFO bafera. Pročitani izlazi iz FIFO bafera se dovode na ulaze oduzimača, a njihovi rezultati na kaskadnu vezu množača i MAC modula kako bi se ostvarila računanja opisana u telu petlje iz Listinga 3.

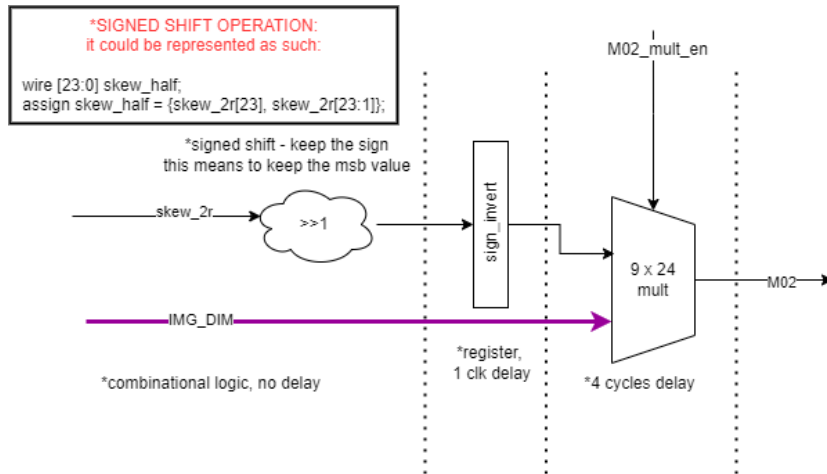


Slika 29: Kolo za računanje centralnih momenata slike

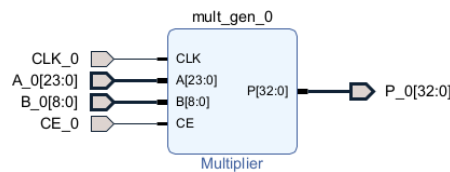
Na slici se vidi da je operacija oduzimanja izvršena kao računanje komplementa dvojke umanjioaca, a zatim se vrši sabiranje označenih brojeva. Množači takođe rade sa označenim brojevima, i u pitanju su dve instance gotovih IP-jeva, takođe iz kataloga alata *Vivado*. MAC moduli su, kao i u slučaju računanja prostornih momenata, implementirani u *Verilogu*, sa instrukcijama za korišćenje DSP-jeva prilikom sinteze.

*Listing 4: Deklaracija MAC modula za rad sa označenim brojevima*

Zakrivljenost slike je propagirana kroz dva registra kako bi se u pravom trenutku pojavila na ulazu kola za računanje elementa transformacione matrice, zbog kašnjenja kontrolnog FSM-a čiji su izlazi signali dozvole i registrovani su, a zavise od statusnog signala delitelja. Kolo sa Slike 30 koje je označeno kao *calc\_transform\_M02* izvršava operaciju množenja sa konstantom  $-0,5$ , na način objašnjen u prethodnom pasusu. Množenje je realizovano koristeći još jedan IP iz već spomenutog kataloga. Unosi kašnjenje od četiri ciklusa takta.



Slika 31: Kolo za računanje elementa M02



Slika 32: Množač iz IP kataloga korišćen za računanje M02

#### 4.3.5 Računanje nove vrednosti piksela i interpolacija

U drugom poglavlju ovog rada je opisana afina transformacija i na koji način je pomoću nje moguće zakriviti ili otkriviti sliku. Neophodno je iterirati kroz sve tačke na izlaznoj slici, zatim za svaku od tih tačaka izračunati koja od tačaka (piksela) na ulaznoj slici treba da joj se dodeli. Pre same dodele vrši se računanje koordinata tačaka okoline te tačke i pomoću njih se računa nova vrednost koristeći metodu interpolacije.

```

for (int x=0; x<a_img_dimension; x++)
for (int y = 0; y<a_img_dimension; y++)
{
    xp = (x + skew*y + transform_M02);
    xl = (int)xp;
    if(xp<(a_img_dimension-1) && y<(a_img_dimension-1) && xp>= 0 && y >=0)
    {
        x2=(xl+1);
        y2=(y+1);
        P = a_image_vector[xl+y2*a_img_dimension] + (xp-xl)*(a_image_vector[x2+y2*a_img_dimension]-a_image_vector[xl+y2*a_img_dimension]);
        new_image[x+y*a_img_dimension]= P;
    }
    else
    {
        new_image[x+y*a_img_dimension]= 0;
    }
}

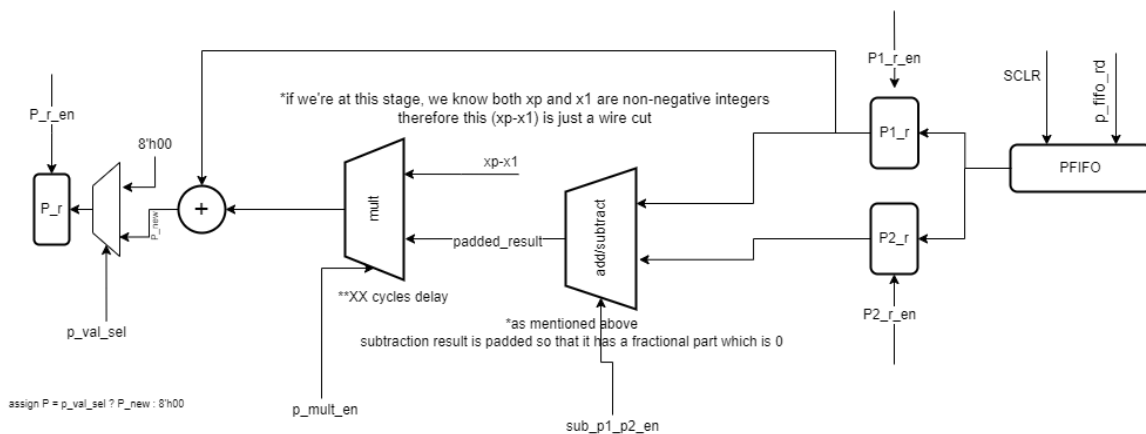
```

Listing 6: Računanje nove vrednosti piksela

The diagram illustrates the PRD\_ADDR generation logic. It starts with input signals  $xp\_gen\_en$  and  $xp\_gen\_sclr$  entering the  $xp\_gen$  block. The output of  $xp\_gen$  is  $xp$ , which is then split into  $xp\_r$  and  $xp[16:9]$ .  $xp\_r$  is a 16-bit register.  $xp[16:9]$  is a 7-bit signal.  $xp\_r$  is also connected to  $Py\_r$ , which is a 16-bit register. The output of  $Py\_r$  is  $y\_cnt$ , which is then incremented by 1 to produce  $y2$ .  $y2$  is then connected to  $calc\_P1\_addr$ , which is a 16-bit register.  $calc\_P1\_addr$  is also connected to  $RD\_MUX$ , which is a 16-bit register.  $RD\_MUX$  is then connected to  $addr\_mux$ , which is a 16-bit register. The output of  $addr\_mux$  is  $addr$ . The diagram also shows a feedback loop from  $addr$  back to  $addr\_mux$  via  $bram\_addr\_sel$ . The logic involves calculating the PRD\_ADDR based on the current address ( $p1\_addr\_out\_r$ ) and the current address ( $p2\_addr\_out\_r$ ). The final PRD\_ADDR is then used to select the address from the BRAM ( $addr\_mux$ ).

Na Slici 33 je predstavljena komponenta označena sa **xp\_gen**, koja računa koordinatu  $x_p$  po prethodno objašnjenom pravilu. Adresa piksela izlazne slike je predstavljena signalom *addr\_gen.addr\_out*, i predstavlja izlaz iz adresnog generatora koji se ne nalazi na slici, a već je prethodno objašnjen. U ovom slučaju, na njegov ulaz OFFSET je dovedena početna adresa izlazne slike (pogledati Sliku 24).

31



Slika 34: Kolo za računanje nove vrednosti piksela

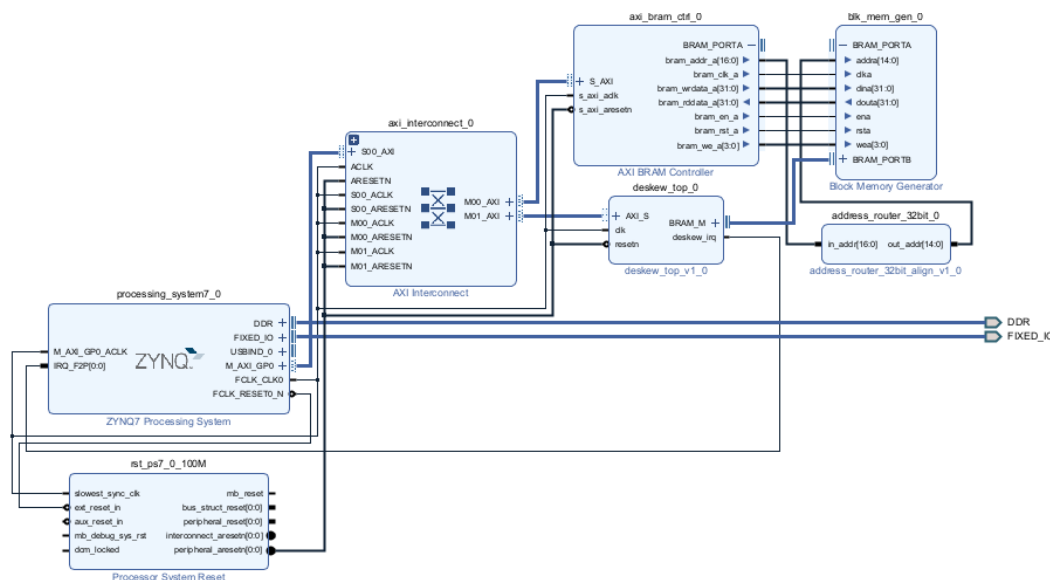
Na Slici 34 se može videti da ispred registra za čuvanje nove vrednosti piksela postoji multiplexer kojim se bira da li će nova vrednost biti ona izračunata interpolacijom ili nula. Treba napomenuti da kada se izračunaju koordinate tačke oko koje se vrši interpolacija, treba proveriti da li se ta tačka nalazi u granicama slike, i ako ne, onda piksel dobija vrednost nula, i ne troši se vreme na interpolaciju.

Ovime je završen proces računanja nove vrednosti piksela, i potrebno je to uraditi za svaki piksel posebno, kako je naznačeno u Listingu 6.



## 5 Integracija i analiza iskorišćenih resursa

Kada je jezgro projektovano i njegova funkcionalnost proverena kroz simulaciju, izvršena je sinteza jezgra, prilagođena za ciljanu platformu, opisanu u drugom poglavlju ovog rada. Jezgro je prvo zapakovano kao dostupni IP, i dodato u biblioteku IP Integratora alata Vivado. Kao takvo je iskorišćeno za kreiranje sistema koji će biti implementiran na ciljanoj platformi. Korišćena verzija alata je *Vivado v2022.1*.



Slika 35: Blok šema sistema za ispravljanje slike

Prvi deo sistema se sastoji iz procesorske jedinice i podsistema za upravljanje reset signalom, zatim BRAM bloka u kome se nalaze ulazna i izlazna slika. Kako je predviđeno, memorija i projektovani IP su procesorskoj jedinici dostupni pomoću memorijskog mapiranja, koristeći AXI port. Za ove svrhe je iskorišćena komponenta *AXI Interconnect*, koja rutira zahteve procesora na memoriju ili **DESKEW\_IP**. Za pristup BRAM memoriji od strane procesora, neophodno je da postoji kontroler koji prevodi AXI transakciju na skup akcija na BRAM interfejsu. Poslednji deo je projektovani IP.

BRAM blok koji je dostupan u IP integratoru može da se konfiguriše na različite načine. Kada je u pitanju konfiguracija pristupa memoriji, dostupne su dve opcije : *AXI BRAM Controller*, i *Standalone*. S obzirom na to da je memorija konfigurisana kao *True Dual Port RAM*, ima dva pristupa preko kojih je moguće vršiti i upis i čitanje, ali tako da su širine podataka različite. Za pristup procesora je određena širina podatka od 32 bita, dok projektovani IP treba pristupa podacima širine 8 bita. Ovo dalje rezultuje time da su širine adresa različite, i to za pristup procesora širina adrese je 15 bita dok je adresa za pristup IP-ja 17 bita. Zbog ovoga je bilo neophodno uvesti jednostavnu komponentu koja će odseći donja dva bita adrese koju postavlja *AXI BRAM Controller* komponenta, i vezati samo gornjih 15 bita na adresnu magistralu BRAM memorije.

Nakon sinteze i implementacije ovog sistema, dobijen je uvid u iskorišćenje resursa i vremensku analizu sistema. Sva ograničenja su zadovoljena, i sistem kao takav je moguće implementirati na željenoj platformi.

Resource	Utilization	Available	Utilization %
LUT	3766	17600	21.40
LUTRAM	177	6000	2.95
FF	6032	35200	17.14
BRAM	32.50	60	54.17
DSP	23	80	28.75
BUFG	1	32	3.13

Tabela 2: Iskorišćeni resursi za implementaciju sistema

4. DSP						
Site Type	Used	Fixed	Prohibited	Available	Util%	
DSPs	23	0	0	80	28.75	
DSP48E1 only	23					

Tabela 3: Iskorišćenost DSP-jeva

3. Memory						
Site Type	Used	Fixed	Prohibited	Available	Util%	
Block RAM Tile	32.5	0	0	60	54.17	
RAMB36/FIFO*	32	0	0	60	53.33	
RAMB36E1 only	32					
RAMB18	1	0	0	120	0.83	
RAMB18E1 only	1					

Tabela 4: Iskorišćenost BRAM blokova

## 5.1 Rezultati vremenske analize

Jezgro je projektovano sa ograničenjem da treba da podrži frekvenciju rada od 100MHz. U ranijim fazama projektovanja, procenjena frekvencija rada je bila približno 94 MHz, što je čak i sporije od postignute brzine rada. Ovde su od interesa kritične putanje koje su projektovane u jezgru, kao i diskusija koja rešenja su moguća kako bi bi se ove kritične putanje rasteretile.

Name	Slack	Levels	Routes	High Fanout	From	To	Total ...	Logic Delay	Net Delay
Path 4	1.357	11	7	38	deskew_system_iides...top/py_r_reg[0]C	deskew_system_iideskew_top...p1_addr_out_r_reg[16]D	8.607	3.804	4.803
Path 1	0.601	0	1	32	deskew_system_iides...th_top/p_r_reg[5]C	deskew_system_iiblk_mem_g...MPLE_PRIM36.ram/DIBD[5]	8.513	0.456	8.057
Path 5	1.482	11	7	38	deskew_system_iides...top/py_r_reg[0]C	deskew_system_iideskew_top...p1_addr_out_r_reg[13]D	8.482	3.810	4.672
Path 6	1.503	11	7	38	deskew_system_iides...top/py_r_reg[0]C	deskew_system_iideskew_top...p1_addr_out_r_reg[15]D	8.461	3.789	4.672
Path 7	1.577	11	7	38	deskew_system_iides...top/py_r_reg[0]C	deskew_system_iideskew_top...p1_addr_out_r_reg[14]D	8.387	3.715	4.672
Path 8	1.593	11	7	38	deskew_system_iides...top/py_r_reg[0]C	deskew_system_iideskew_top...p1_addr_out_r_reg[12]D	8.371	3.699	4.672
Path 2	1.084	0	1	32	deskew_system_iides...th_top/p_r_reg[5]C	deskew_system_iiblk_mem_g...MPLE_PRIM36.ram/DIBD[5]	8.022	0.456	7.566
Path 3	1.229	0	1	32	deskew_system_iides...th_top/p_r_reg[5]C	deskew_system_iiblk_mem_g...MPLE_PRIM36.ram/DIBD[5]	7.873	0.456	7.417
Path 10	1.820	3	3	1	deskew_system_i...36.ram/CLKBWRCLK	deskew_system_iideskew_t...em_reg_0_7_0_5/RAMC_D1I	7.793	3.114	4.679
Path 9	1.714	0	1	32	deskew_system_iides...th_top/p_r_reg[5]C	deskew_system_iiblk_mem...MPLE_PRIM36.ram/DIBD[5]	7.394	0.456	6.938

Tabela 5: Kritične putanje u projektovanom jezgru

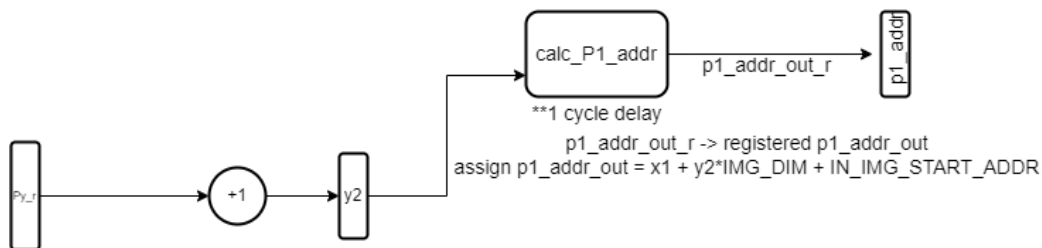
U Tabeli 5 su izlistane putanje jednobitnih signala koje imaju najduže kašnjenje. Može se uočiti da su u pitanju signali koji odgovaraju izlazima dva specifična registra. Putanje označene brojevima 1, 2, 3 i 9 odgovaraju izlazu registra koji sadrži novu izračunatu vrednost piksela. Krajnje odredište ovih signala je ulaz za podatak u BRAM blok. Ova putanja ne prolazi kroz bilo kakvu kombinacionu mrežu, već je direktna linija između registra i memorije:

- Registar početka putanje:  
*deskew\_top.deskew\_datapath.p\_r\_reg*
  - Predstavljen na Slici 34, označen sa **P\_r**
- Registar kraja putanje:  
*deskew\_system\_i.blk\_mem\_g....SIMPLE\_PRIM36.ram.DIBDI*
  - Ovaj registar je interni element BRAM generator IP-ja, i krajnji korisnik nema direktni pristup
- Putanja prolazi kroz memorijski kontroler bez dodatne logike u njemu. Ovo je prikazano na Slici 21

Ova kašnjenja bi mogla biti smanjena ukoliko bi se u memorijskom kontroleru, koji je opisan u poglavlju 4.3.1 ovog dokumenta, dodao jedan bafer za signale upisa u memoriju. To bi dalje prouzrokovalo potencijalni problem sa operacijama čitanja, pa bi i one morale da dobiju identičan nivo baferovanja, tako da bi se pristup memoriji pomerio za jedan takt ciklus. U ovom slučaju su vremenska ograničenja zadovoljena, pa nije bilo potrebe za ovim akcijama.

Putanje označene brojevima 4, 5, 6, 7 i 8 su izlazi iz registra označenog sa **Py\_r** na Slici 33. Ovaj signal pripada kolu za računanje adrese piksela koji su ulazni podaci za operaciju interpolacije. Prolazi kroz kaskadnu vezu dva sabirača i jednog množača. U ovom slučaju nije korišćen posebno projekovani modul koji bi ubrzao rad sistema, niti je primenjena protočna obrada podataka. Napisan je Verilog skup naredbi kontinualne dodele vrednosti signalu, što je rezultiralo u korišćenju neoptimizovane kombinacione mreže.

- Registar početka ove putanje je **Py\_r**, na putanji  
*deskew\_top.deskew\_datapath.py\_r\_reg*
- Registar na završetku putanje je **p1\_addr\_out\_r**, na putanji :  
*deskew\_top.deskew\_datapath.p1\_addr\_out\_r\_reg*



Slika 36: Kritična putanja u projektovanom IP jezgru

Inicijalna ideja je bila da nakon prvog sabirača unese registar, koji bi na slici stajao na mestu elementa označenog sa y2. Krajnja implementacija je takva da je umesto registra postavljen samo signal sa istom oznakom. Takođe, na Slici 36 je ispod elementa sa oznakom *calc\_P1\_addr* napisan Verilog kod kojim je to kolo implementirano, gde je i bilo očekivano da se u sintezi ovde pojavi kombinaciona mreža. Ova putanja je mogla biti skraćena korišćenjem dodatnih sabirača i množača koji podržavaju protočnu obradu podataka, ili čak samo dodavanjem registra na mesto već pomenute y2 komponente. Ovo bi dalje rezultiralo u prilagođavanju kontrolne mašine stanja u *Controlpath*-u. Odlučeno je da se ne primeni nijedno od navedenih rešenja, jer nije prekršeno nijedno ograničenje za implementaciju sistema.

## 5.2 Programiranje ploče

Poslednji korak u projektovanju sistema jeste pisanje odgovarajuće softverske aplikacije, kao i pokretanje na ploči i otklanjanje grešaka na ovom nivou. Kako je opisano u poglavlju 3.1.1, softver je napisan po modelu koji je koršćen u *Testbench*-u razvijene virtuelne platforme. Skup koraka je isti, i svaki od koraka je u implementaciji prilagođen za hardver na kome se izvršava.

Ulazna slika je učitana u fajlu koje je C++ zaglavlje, a prethodno je koristeći *python* skriptu prilagođen njen format (iz *.bmp* formata je pretvorena u niz neoznačenih brojeva). Napisan je glavni program kao i servisna rutina prekida, koja dozvoljava glavnom programu da nastavi sa izvršavanjem nakon što pokrene proces ispravljanja slike. Koraci iz kojih se sastoji aplikacija su:

- Inicijalizacija hardverske platforme i osposobljavanje kontrolera prekida
  - Registrovanje željenog prekida i željene servisne rutine prekida
- Učitavanje ulazne slike u memoriju
- Konfigurisanje IP-ja i slanje signala za početak operacije
- Čekanje na prekid
- Čitanje izlazne slike iz memorije

```
int main()
{
    int Status; //IRQ Init fucntion status
    u32 img_array_length;
    u32 in_img_ptr;
    u32 out_img_ptr;
    u32 img_dim;

    DSQW_INTERRUPTS_CLEARED = 0;

    init_platform();

    /* INITIALIZE SYSTEM INTERRUPTS*/
    Status = InitDeskewIrq(XPAR_PS7_SCUGIC_0_DEVICE_ID);
    if (Status != XST_SUCCESS)
    {
        xil_printf("IRQ INIT FAILED. Status : %0d\n", Status);
        return Status;
    }

    /*CONFIGURE IMAGE PARAMETERS*/
    img_array_length = 65536;
    img_dim = 256;
    in_img_ptr = 0x00000000;
    out_img_ptr = 0x00010000;

    /*LOAD IMAGE TO MEMORY*/
    load_input_image(in_img_ptr, img_array_length);

    /*Kick-off the Deskewer*/
    ProgramDeskewIp(in_img_ptr, out_img_ptr, img_dim);

    xil_printf("Deskew Programmed!\n");

    /*Wait for the interrupt*/
    while(DSQW_INTERRUPTS_CLEARED != 1)
    {
        sleep(10);
    }

    DSQW_INTERRUPTS_CLEARED = 0;

    xil_printf("Interrupt acknowledged!\n");

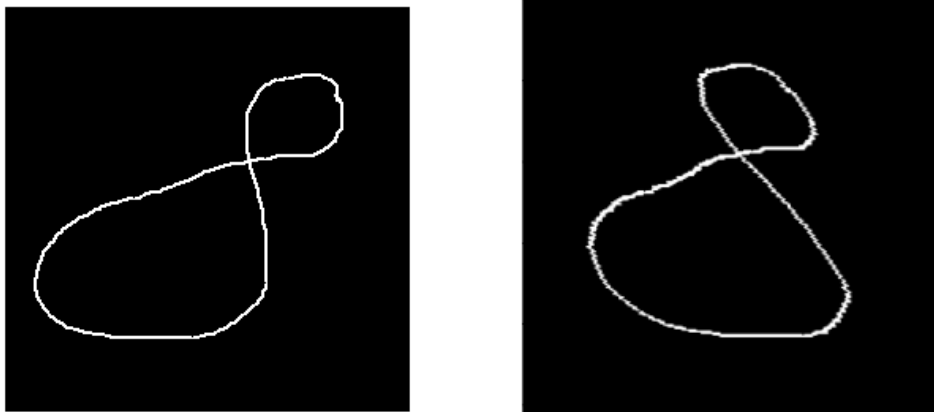
    /*READ OUTPUT IMAGE FROM MEMORY*/
    read_output_image(out_img_ptr, img_array_length);

    /*Format the rest of the print*/
    xil_printf("\n\n");

    /*LOAD IMAGE TO MEMORY*/
    print("Deskew done\n\n");
    cleanup_platform();
    return 0;
}
```

Listing 7: C++ aplikacija za kontrolu procesa ispravljanja slika

Ploča na kojoj je implementiran ceo sistem je povezana sa računarom sa kojim komunicira preko serijskog porta, kako bi krajnji korisnik mogao da pročitati informacije o trenutnom stanju sistema. Izlazna slika je takođe prosleđena preko serijskog porta kako bi bila iscrtana koristeći *python* skriptu.



*Slika 37: Ulazna i izlazna slika dimenzija 256 x 256 piksela*

## 6 Zaključak

Kranji sistem je uspešno implementiran i testiran na željenoj platformi. Prateći ESL metodologiju, razvoj sistema je prošao kroz nekoliko ključnih koraka. Prvi korak je bio odabir dostupne platforme i razumevanje ograničenja koje ta platforma unosi. Ovde je uvedeno prvo ograničenje za maksimalnu dimenziju slike 256 x 256 piksela. Odrednica je takođe bila da maksimalna dimenzija slike bude izražena u broju koji je stepen broja dva. Ukupna iskorišćenost memorije je preko 50%, a za sliku dimenzije 512 x 512 piksela bi bilo potrebno osam puta više memorije, iz čega zaključujemo da je ovo ograničenje bilo neophodno.

Sledeći korak je bio pisanje izvršne specifikacije u C++ jeziku, za kojom je sledilo pisanje virtuelne platforme koristeći SystemC jezik. U ovom koraku je određena komunikacija između modula, kao i particionisanje na hardverski i softverski deo. Kako je tu bilo pretpostavljeno, softver je bio implementiran u testbenču virtuelne platforme, i ista struktura je zadržana pri implementiranju softvera za krajnji sistem.

Podela hardvera nije promenjena, uvedeni su gotovi IP-jevi koji su mogli biti izostavljeni u virtuelnoj platformi, i zatim je projektovano jezgro koje implementira algoritam za ispravljanje slike. Virtuelna platforma je iskorišćena kako bi se unapred odredili neophodni hardverski resursi za reprezentaciju rezultata.

Procena frekvencije rada je nadmašena, u granicama normale. Procenjena frekvencija rada jezgra je bila 94 MHz, a postignuta frekvencija je 100MHz. U poglavlju 5.1 su razmatrana rešenja za skraćivanje kritičnih putanja, što bi potencijalno ostavilo prostor za još brži rad sistema, odnosno veću frekvenciju takt signala.

Jezgro je testirano kroz simulacije koristeći SystemVerilog testbenč, zajedno sa komponentama razvijenim koristeći UVM biblioteku. Nakon ovoga je jezgro zapakovano kao dostupni IP. Integrisano je u krajnji sistem, kako je ranije opisano.

U fazi razvoja softvera otkrivene su greške poput pogrešnog konfigurisanja korišćenih IP-jeva, neporavnanja u adresnoj mapi, neodgovarajuće širine adresa između AXI BRAM kontrolera i same memorije. Za detekciju ovih grešaka, bilo je potrebno koristiti tzv. *Logic analyzer-e*, kojima je bilo moguće pristupiti signalima sistema dok je sistem bio aktivan.

Osim pomenutih grešaka, u projektovanom jezgru je otkriven problem sa AXI kontrolerom koji nije mogao da procesuirati transakcije u kojima se prvo pojavljivao podatak, a zatim adresa na kanalu za upis podataka.

Druga greška u projektovanom jezgru koja je pronađena na ovom nivou je obrada slike dimenzije 256 x 256 piksela. Utvrđeno je da širine brojača u adresnom generatoru nisu bile ogovarajuće. Samo jezgro je prošlo kroz nekoliko revizija u procesu otklanjanja grešaka.

Trenutna verzija sistema ne koristi pun potencijal dostupnih sposobnosti izabrane platforme. Ispravljanje slike se izvršava dovoljno brzo, i nije razmatrano nijedno poboljšanje osim povećanja frekvencije rada. Najsporiji deo sistema je deo za učitavanje slike u BRAM memoriju, i čitanje izlazne slike, koja je zatim poslata ka korisniku koristeći serijski port. Ovaj deo se izvršava uzastopnim AXI transakcijama, od kojih svaka prenosi po 32 bita podataka. Kada bi se u sistem dodao DMA modul, koji bi podržavao tzv. *AXI Burst-ove*, gde bi količina podataka koja se prenosi u jednom transferu bila znatno veća, ovaj nedostatak bi se znatno zanemario.

Kao krajnja verzija sistema, mogla bi biti razvijena aplikacija koja preko serijskog porta dobija ulaznu sliku i komande za konfigurisanje IP-ja. Komunicirala bi sa aplikacijom na drugoj mašini, odakle bi korisnik mogao da šalje slike koje želi da obradi. Takođe, tada bi i procesi parsiranja ulazne slike i iscrtavanja izlazne slike mogli da budu automatizovani, sistem bi bio znatno bolje iskorišćen.

## 7 Literatura

- [1] - <https://www.mathworks.com/discovery/affine-transformation.html#:~:text=Affine%20transformation%20is%20a%20linear,with%20non%20ideal%20camera%20angles>. (15.10.2022.)
- [2] - <https://www.graphicsmill.com/docs/gm/affine-and-projective-transformations.htm> (19.10.2022.)
- [3] - <https://learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial> , (19.10.2022.)
- [4] - Rastislav Struharik, "Projektovanje složenih digitalnih sistema" - Sinteza kombinacionih i sekvencijalnih mreža
- [5] - Vuk Vranjković, "Projektovanje elektronskih uređaja na sistemskom nivou" - Zynq FPGA, Fakultet tehničkih nauka
- [6] - Vuk Vranjković, "Projektovanje elektronskih uređaja na sistemskom nivou" - ESL tok projektovanja, Fakultet tehničkih nauka
- [7] - Rastislav Struharik, "Projektovanje složenih digitalnih sistema" - Hijerarhijski i parametrizovani dizajn
- [8] - Pong P. Chu, „RTL Hardware Design using VHDL”, Wiley-Interscience, 2006.
- [9] - <https://www.xilinx.com/content/dam/xilinx/imgs/block-diagrams/zynq-mp-core-dual.png> (20.10.2022.)