Filtriranje slike pomoću filtra za smanjenje šuma

Bešlić, Tomislav

Master's thesis / Diplomski rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:993419

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

FILTRIRANJE SLIKE POMOĆU FILTRA ZA SMANJENJE ŠUMA

Diplomski rad

Tomislav Bešlić

Osijek, 2017.

SADRŽAJ

1. UVOD 1
1.1 ZADATAK DIPLOMSKOG RADA 1
2. PRIMIJENJENE TEHNOLOGIJE 2
2.1 FPGA TEHNOLOGIJA 2
2.2 MODELSIM 3
2.3 FILTRIRANJE SLIKE 5
2.4 POSTUPAK PROJEKTIRANJA SUSTAVA 6
3. FILTRIRANJE SLIKE POMOĆU FPGA TEHNOLOGIJE 12
3.1 SKLOPOVSKI DIO MAKETE 12
3.1.1 Procesorski sustav Zynq uređaja (PS) 12
3.1.2 Jedinica za obradu aplikacija (APU) 13
3.1.3 Programibilna logika (PL) 15
3.2 PROGRAMSKO SUČELJE MIKROUPRAVLJAČA 17
3.2.1 Obrada slike iz BRAM-a 17
3.2.2 Obrada slike iz RAM-a 19
3.2.3 Upravljač memorije i obrade slike 20
3.2.4 Kvantizacija jezgre 22
3.3 PROGRAMSKO SUČELJE STOLNOG RAČUNALA 24
3.3.1 Programsko sučelje izvedeno s BRAM-om 24
3.3.2 Programsko sučelje izvedeno s RAM-om 25
3.3.3 AXI standard 27
3.3.4 Blok RAM 30
3.3.5 DSP48E1 31
3.3.6 DMA 32
4. EKSPERIMENTALNI REZULTATI 34
5. ZAKLJUČAK 38

1. UVOD

Cilj ovog diplomskog rada je filtriranje ulazne slike pomoću FPGA tehnologije koristeći Gauss 2D filtar u svrhu smanjenja šuma. Filtriranje slike je potrebno obavljati u realnom vremenu, ali implementacija Gauss 2D filtra zahtjeva pozamašne računalne resurse za obradu. Iz tog razloga se primjenjuje FPGA tehnologija koja omogućava efikasniju i bržu aritmetiku za obradu. U ovom radu će se testirati brzine obrade slike različitih veličina na dva različita tipa memorije i dva reda jezgre filtra, te će se izmjerene brzine obrade usporediti.

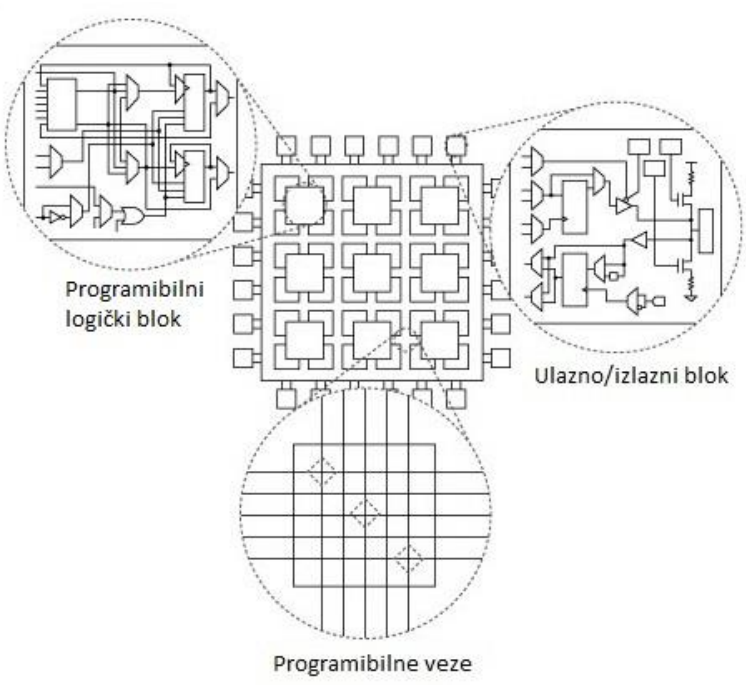
1.1 Zadatak diplomskog rada

U ovom diplomskom radu potrebno je implementirati filtriranje 2D slike pomoću Gauss 2D filtra na FPGA tehnologiji.

2. PRIMIJENJENE TEHNOLOGIJE

2.1 FPGA tehnologija

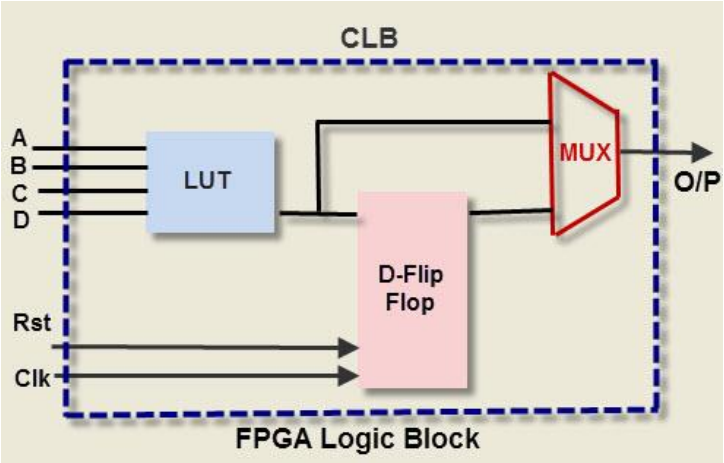
FPGA *(engl. field - programmable gate array)* – Programibilni logički sklopovi su uređaji uvedeni u upotrebu od strane tvrtke Xilinx sredinom 1980-ih. Razlikuju se od CPLD *(engl. complex programmable logic device)* u arhitekturi, tehnologiji pohrane, broju ugrađenih značajki i cijeni. Osnovna ideja FPGA sklopova je da se proizvedeni čip može prilagoditi, odnosno programirati, od strane korisnika nakon proizvodnje. Namijenjeni su za implementaciju čipova visokih performansi.



Slika 2.1. Arhitektura FPGA čipa

Osnovna arhitektura FPGA je prikazana iznad (Slika 2.1). Sastoji se od matrice programibilnih logičkih blokova – CLB, koji su međusobno povezani programibilnim vezama. Na svakom izlazu FPGA čipa se nalaze ulazno/izlazni blokovi.

CLB je osnovni element FPGA tehnologije. Sastoje se od nekoliko logičkih ćelija. Tipična ćelija se sastoji od četveroulazne LUT *(engl. Lookup table)*, D bistabila te multipleksera. Izlaz s multipleksera može biti sinkron ili asinkron, ovisno o programiranju multipleksera. Blokovski prikaz arhitekture CLB prikazan je na slici 2.2.

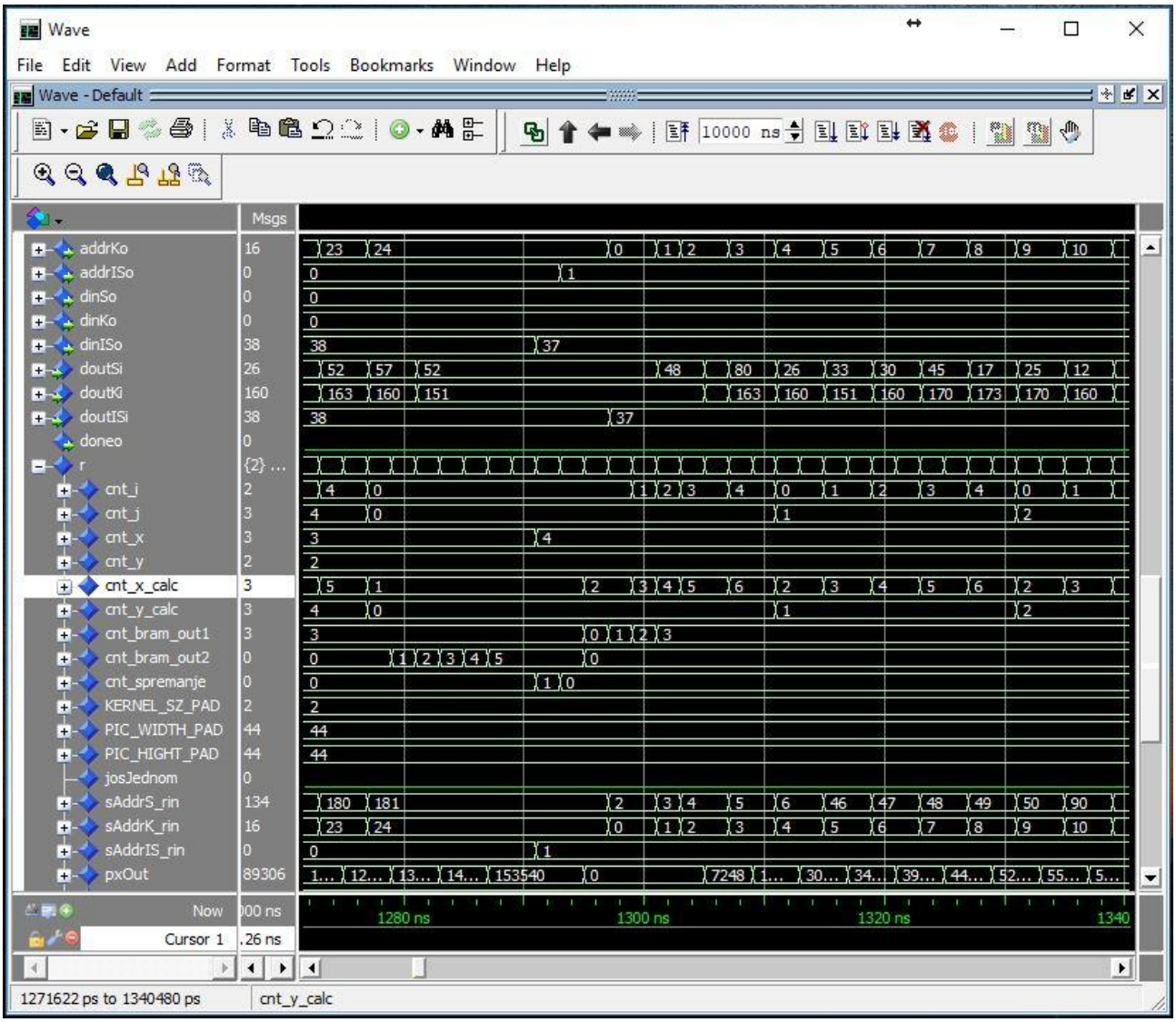


Slika 2.2. Arhitektura CLB-a

2.2 ModelSim

ModelSim je simulacijska okolina za više različitih HDL jezika koju je razvila tvrtka Mentor Graphics u svrhu simulacije opisnih jezika hardvera poput VHDL-a, Verilog-a ili SystemC-a te uključuje već ugrađeni C debugger. Simulacija se izvršava preko grafičkog korisničkog sučelja ili automatski koristeći skripte. FPGA uređaji su prošli radikalne promjene s vremenom te su postali kompleksni. Kao rezultat zahtijevaju napredniju verifikacijsku tehnologiju za poboljšanje dizajniranog FPGA sustava. Ovakav grafički način omogućava jednostavnije otklanjanje pogrešaka u opisu hardvera. Na slici 2.3 prikazan je prozor ModelSim programskog sučelja u kojemu se promatraju simulirani signali u sustavu koji se simulira.

Simulacija je najčešće korištena metoda verifikacije, prilikom koje se izrađuje funkcionalni model dizajniranog sustava. Nakon toga se model pobuđuje u simulatoru te se analizira odziv. Za potrebe izrade modela potreban je opis sustava u nekom od jezika za opis hardvera. Prilikom simulacije potrebno je opisati ulaznu pobudu za dizajnirani model, što se nakon toga učitava u simulator i izvršava. Kao rezultat, simulator vraća grafički prikaz odziva signala koje se želi pratiti. Simulacija omogućava ispitivanje rada sustava te uočavanje grešaka prije realizacije na fizičkom sustavu. Ispravan rad modela u simulatoru ne mora sa sigurnošću značiti i ispravan rad na fizičkom sustavu. Model može generirati ispravan odziv na sve pobude, ali postoji mogućnost da se prilikom verifikacije



Slika 2.3. Izgled ModelSim programskog sučelja u kojem se promatraju signali u simuliranom sustavu

nije pobudio svaki dio sustava, te takvom greškom doveo do propusta verifikacije cjelokupnog sustava, koji možda sadrži grešku u nekoj komponenti koja nije zahvaćena pobudom. S druge strane, postoji mogućnost da je sustav toliko kompleksan da je broj stimulansa, odnosno test vektora, prevelik za simulaciju, tako da bi izvođenje simulacije trajalo iznimno dugo.

2.3 Filtriranje slike

Filtri za obradu slike se koriste kako bi smanjili visoke ili niske frekvencije u slici. U slučaju prostorne domene, operacija filtriranja se izvodi direktno na slici bez transformacije. U frekvencijskoj domeni slika se prvo transformira u frekvencijsku domenu određenim oblikom FFT *(engl. Fast Fourier Transform)* algoritma, te se nakon toga primjenjuje željeni filtar. Filtri prostorne domene koriste operacije poput konvolucije, množenja piksel-po-piksel i sumiranja, dok filtri frekvencijske domene zahtijevaju maskiranje, eliminaciju i pred-filtarsko izjednačavanje.

Filtri prostorne domene uključuju konvoluciju slike *f(m, n)* s funkcijom filtra *h(m, n)*. Izraz ima slijedeći oblik [1]:

$$g(m,n) = f(m,n) \otimes h(m,n) \tag{2-1}$$

Ova matematička operacija je identična množenju u frekvencijskoj domeni, ali rezultati implementacija variraju jer se funkcija filtra mora odrediti pomoću diskretne i konačne jezgre *(engl. kernel)*.

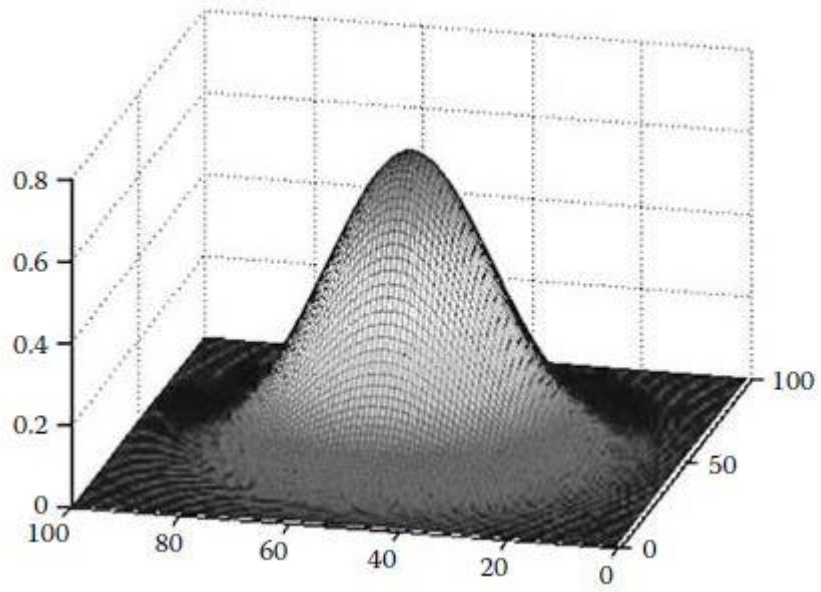
Gauss operator izglađivanja slike je 2D konvolucijski operator koji se koristi da se uklone detalji i šum, odnosno da se slika „zamuti“. Gauss filtar koristi drugačiju jezgru koja predstavlja oblik Gauss krivulje zvonastog oblika. Gauss distribucija u 2D prostoru je sljedećeg oblika [1]:

$$h(m,n) = \frac{1}{2\pi\sigma^2} e^{-\frac{(m-m_0)^2+(n-n_0)^2}{2\sigma^2}} \tag{2-2}$$

Gauss distribucija je prikazana na Slici 2.4 Gauss izglađivanje koristi 2D distribuciju kao funkciju širenja točke što se postiže konvolucijom jezgre sa slikom koja se obrađuje. Općenito govoreći, slika se sprema kao skupina diskretnih piksela, te se stoga mora odrediti diskretna aproksimacija Gauss funkcije prije nego se može obaviti konvolucija. Teoretski, Gauss funkcija je različita od nule u

svakom dijelu funkcije, što bi značilo da bi jezgra trebala biti beskonačno velika. U praksi se ipak uzima da je funkcija jednaka nuli nakon 3 standardne devijacije od srednje vrijednosti te se stoga jezgra može skratiti na realnu veličinu. Nakon određivanja željene jezgre, može se obaviti postupak konvolucije.

Efekt Gauss izglađivanja je zamućivanje. Jakost zamućivanja ovisi o standardnoj devijaciji.



Slika 2.4. Gauss distribucija sa srednjom vrijednošću 50% i s = 1[1]

2.4 Postupak projektiranja sustava

Kako bi se projektirao nekakav digitalni sustav potrebno je napraviti mnogo koraka prije njegove realizacije. Za projektiranje digitalnog sustava projektantu je na raspolaganju više različitih tehnologija i metodologija, kao i za provjere ispravnosti sustava koji se izrađuje. Također, na raspolaganju su i alati za transformaciju u oblik koji je pogodan za fizičku realizaciju. Prilikom razrade takvog sustava potrebno je više koraka, a nakon svakog koraka potrebna je validacija rješenja prethodnog koraka kako bi se pronašle i ispravile eventualne pogreške nastale prilikom projektiranja. Spomenuta validacija osigurava ispravnost završnog proizvoda.

Kako je navedeno u tekstu iznad, potrebno je izvršiti više koraka prilikom projektiranja digitalnih sustava. Koraci su slijedeći:

- Sinteza,

- Fizičko projektiranje,
- Verifikacija,
- Testiranje.

Sinteza je proces razrade čiji je cilj da se uz pomoć komponenti raspoloživih na nižem nivou apstrakcije realizira sustav opisan na višem nivou apstrakcije. [3] Opisivanje dizajna sustava se može raditi kao strukturni i kao funkcionalni tip, ali konačan rezultat će uvijek biti strukturni koji se nalazi na nižem apstraktnom nivou. Napredovanjem sinteze, opis sustava postaje sve detaljniji, a kao konačan rezultat daje strukturnu reprezentaciju na razini logičkih krugova koja se sastoji od osnovnih logičkih blokova ciljane tehnologije. Sam proces sinteze se dijeli na više manjih postupaka gdje svaki korak obavlja neku specifičnu vrstu transformacije. Koraci su:

- Sinteza visoke razine – transformira algoritam u funkcionalan RTL *(engl. Register Transfer Level)* opis. Algoritam namijenjen za sintezu može biti napisan u nekim od viših programskih jezika poput C-a, što je nemoguće izravno sintetizirati. Stoga se vrši sinteza visoke razine kako bi se takav algoritam mogao prevesti u neki od jezika za opis sklopovlja poput VHDL-a ili Verilog-a.
- RTL sinteza – provodi transformaciju funkcionalnog RTL opisa u strukturni opis korištenjem komponenti s RTL razine. Prilikom ove sinteze izvršavaju se razne optimizacije kako bi se smanjio broj korištenih komponenti na sklopovlju.
- Logička sinteza – provodi transformaciju strukturnog RTL opisa u mrežu logičkih sklopova koja može biti kombinacijska i sekvencijalna. Prvi koraci logičke sinteze provode optimizaciju mreže u svrhu smanjenja količine potrebnih logičkih sklopova i/ili u svrhu smanjenja propagacijskog kašnjenja. Prilikom ove sinteze koriste se standardne logičke komponente te se vrši optimizacija standardnim metodama poput Karnoovih mapa i sl. Ova sinteza je neovisna od ciljane tehnologije.
- Tehnološko mapiranje – posljednji korak u sintezi koji uzima podatke dobivene nakon logičke sinteze, te ih primjenjuje na ciljanu tehnologiju. Ciljana tehnologija može biti FPGA tehnologija, standardne stanice, NI i NILI sklopovi i sl.

Fizičko projektiranje vrši transformaciju dobivenih podataka od strane sinteze u fizičku reprezentaciju na samom sklopovlju, te vrši analizu i optimizaciju sustava koji se kreirao.

Fizičko projektiranje ima nekoliko faza, a to su: prostorno planiranje, raspoređivanje i povezivanje. Prva faza je prostorno planiranje prilikom kojega se funkcionalne cjeline grupiraju u smislene cjeline te se raspoređuju na određene lokacije na čipu. Dijele se na način da se primjerice procesor smješta u jedan kut čipa, memorija u drugi, a preostale komponente se smještaju na preostali dio. Druga faza je raspoređivanje gdje se određuju točne fizičke lokacije kreiranih komponenti, nakon čega dolazi faza povezivanja. Prilikom povezivanja se kreiraju sve potrebne veze pri čemu se obraća pozornost na optimalnu, odnosno najkraću putanju. Nakon spomenutih faza kreirana je sklopovska reprezentacija sustava koja se može implementirati na sklopovlje.

Nakon obavljenih postupaka raspoređivanja i povezivanja, vrši se izračun popratnih parazitnih kapacitivnosti i otpornosti. Proces se naziva ekstrakcija parametara. Kako su poznate električne karakteristike upotrijebljenih komponenti može se moguće je izračunati propagacijska kašnjenja te emisija energije.

Verifikacija se vrši kako bi se provjerio ostvareni dizajn u smislu zadovoljavanja postavljenih zahtjeva i performansi. Vrši se neposredno nakon svakog koraka sinteze i fizičkog projektiranja. Prilikom verifikacije provjeravaju se dvije stavke: funkcionalnost i performanse. Funkcionalna verifikacija vrši provjeru generiranja očekivanog odziva na zadanu pobudu. Performanse se zadaju vremenskim ograničenjima u vidu maksimalnog kašnjenja, potrošnje, zauzeća FPGA resursa i sl. Verifikacijom performansi provjerava se je li sustav generirao odziv unutar postavljenih ograničenja.

Testiranje i verifikacija u jezičnom smislu imaju približno isto značenje, no kada je riječ o razvoju digitalnih sustava to su dva različita pojma. Kako je navedeno u tekstu iznad, verifikacija provjerava zadovoljavanje funkcionalnosti i performansi sustava, dok se testiranjem otkrivaju fizičke pogreške na gotovom proizvodu (pogreške koje mogu nastati tokom proizvodnje). Testiranje se izvodi na gotovom proizvodu kako bi se izdvojile ispravne jedinice od neispravnih.

Testiranje upotrebljava testne slučajeve u kojima testira odziv jedinice na svaku ulaznu pobudu, nakon čega se prati odziv na izlazu. U praksi je često neizvedivo testirati sve slučajeve zbog ogromnog broja ulaznih kombinacija, stoga se postupak testiranja na fizičkoj jedinici izvodi uz pomoć prethodno generiranih testnih slučajeva kako bi sklop bio što potpunije testiran. Generiranje testnih slučajeva se vrši pomoću posebnih algoritama.

CAD alati *(engl. Computer-Aided Design)* se koriste za automatizaciju pojedinih zadataka u procesu projektiranja sustava. To je računalni softver koji pomaže u razvoju većih digitalnih sustava, a koristi složene postupke projektiranja i manipulira velikom količinom informacija. CAD alati su nezamjenjiva podrška projektiranju te se koriste u svim fazama projektiranja. Takvi alati imaju različite uloge prilikom projektiranja sustava, te se prema tome mogu podijeliti u pet kategorija:

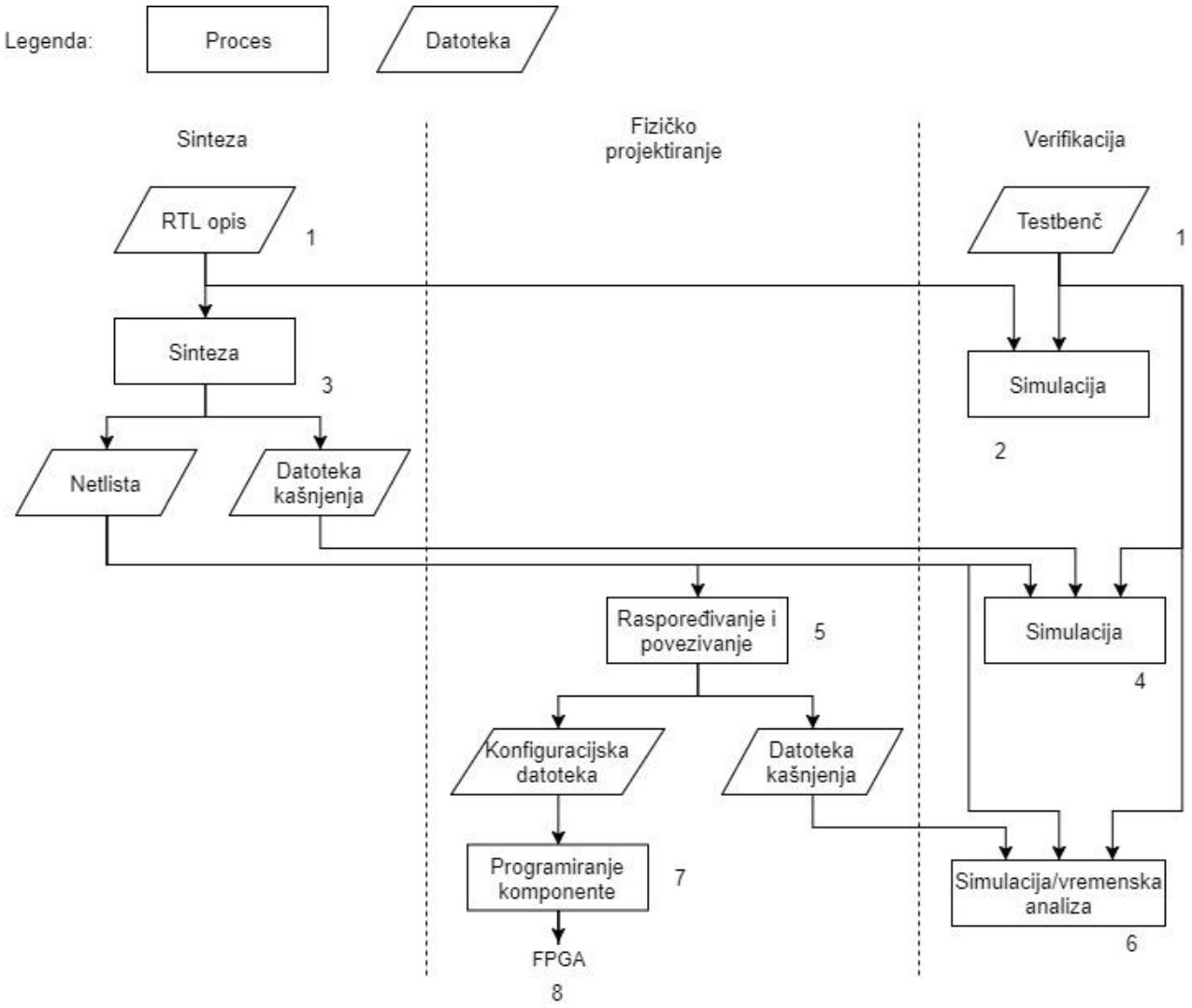
- Alati za opis i modeliranje,
- Alati za sintezu,
- Alati za verifikaciju i simulaciju,
- Alati za raspoređivanje i povezivanje,
- Alati za generiranje testnih nizova.

Iako su CAD alati jako moćan alat za projektiranje sustava, nisu svemogući. Alat za sintezu ne može promijeniti lošu ideju dizajna u dobru, niti promijeniti primijenjenu arhitekturu sustava. Glavna odgovornost za dizajniranje još uvijek leži na inženjeru sustava.

Tok projektiranja, kako je već prethodno spomenuto, je iterativni proces razrade i verifikacije digitalnog sustava prilikom čega se razrađuje visoki nivo apstrakcije u detaljan strukturni opis niskog nivoa. Tok projektiranja je različit za različite ciljane tehnologije, a ovisi i o veličini sustava.

Povećanjem veličine digitalnog sustava rastu hardverski zahtjevi za projektiranje samog sustava, odnosno za potrebe sinteze rastu procesorski i memorijski zahtjevi. Današnji CAD alati su dostatni za sustave srednje razine (od 2000 do 50000 logičkih sklopova). U slučajevima većih sustava potrebno je podijeliti sustav na blokove te ih nezavisno procesuirati.

Projektiranje započinje kreiranjem dizajn datoteke u jeziku za opis hardvera, koja sadrži funkcionalni opis RTL razine. U sklopu sustava izrađuje se i takozvana datoteka ispitnog okruženja u svrhu opisa virtualnog eksperimentalnog okruženja za simulaciju i verifikaciju dizajna sustava. Opis eksperimenata objedinjuje opis koji generira simulaciju, opis modela sustava koji se verificira te opis koji nadgleda i analizira odzive generirane tijekom simulacije.



Slika 2.5 Tok projektiranja digitalnog sustava srednje veličine za FPGA tehnologiju[3]

Tok projektiranja podrazumijeva slijedeće korake (Slika 2.5):

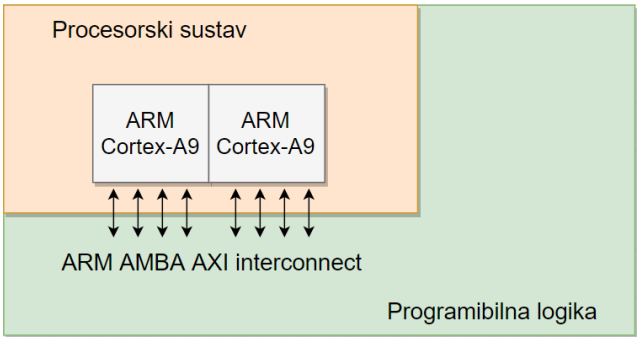
1. Razvoj dizajn-datoteke i datoteke ispitnog okruženja,
2. Funkcionalna simulacija, s ciljem verifikacije RTL opisa,
3. Sinteza,
4. Datoteka koja sadrži netlistu korištenu za simulaciju i vremensku analizu, u svrhu verifikacije funkcionalne ispravnosti sintetizirane logičke mreže te preliminarne provjere vremenskih ograničenja,
5. Raspoređivanje i povezivanje. Logička vrata i veze iz netliste se preslikavaju na fizičke elemente FPGA čipa,

6. Izvedba preciznih podataka o vremenskim ograničenjima. Netlista, proširena navedenim podacima koristi se za simulaciju i vremensku analizu u cilju verifikacije raspoređivanja i povezivanja,
7. Generiranje konfiguracijske datoteke i programiranje FPGA čipa,
8. Verifikacija rada fizičkog sustava.

3. FILTRIRANJE SLIKE POMOĆU FPGA TEHNOLOGIJE

3.1 Sklopovski dio makete

Testiranje diplomskog rada se izvodi na Zybo razvojnoj pločici na kojoj se nalazi Xilinx Zynq-7000 SoC *(engl. System on a Chip)* s 240 KB Blok RAM memorije. Navedeni čip je takozvani sustav na čipu, što znači da se unutar istog čipa nalazi dvojezgreni ARM Cortex-A9 procesor te FPGA (Slika 3.1). Takav sustav objedinjuje jednostavnost programiranja ugrađenog procesora sa sklopovskom fleksibilnošću FPGA.



Slika 3.1. Arhitektura Zynq – 7000 SoC

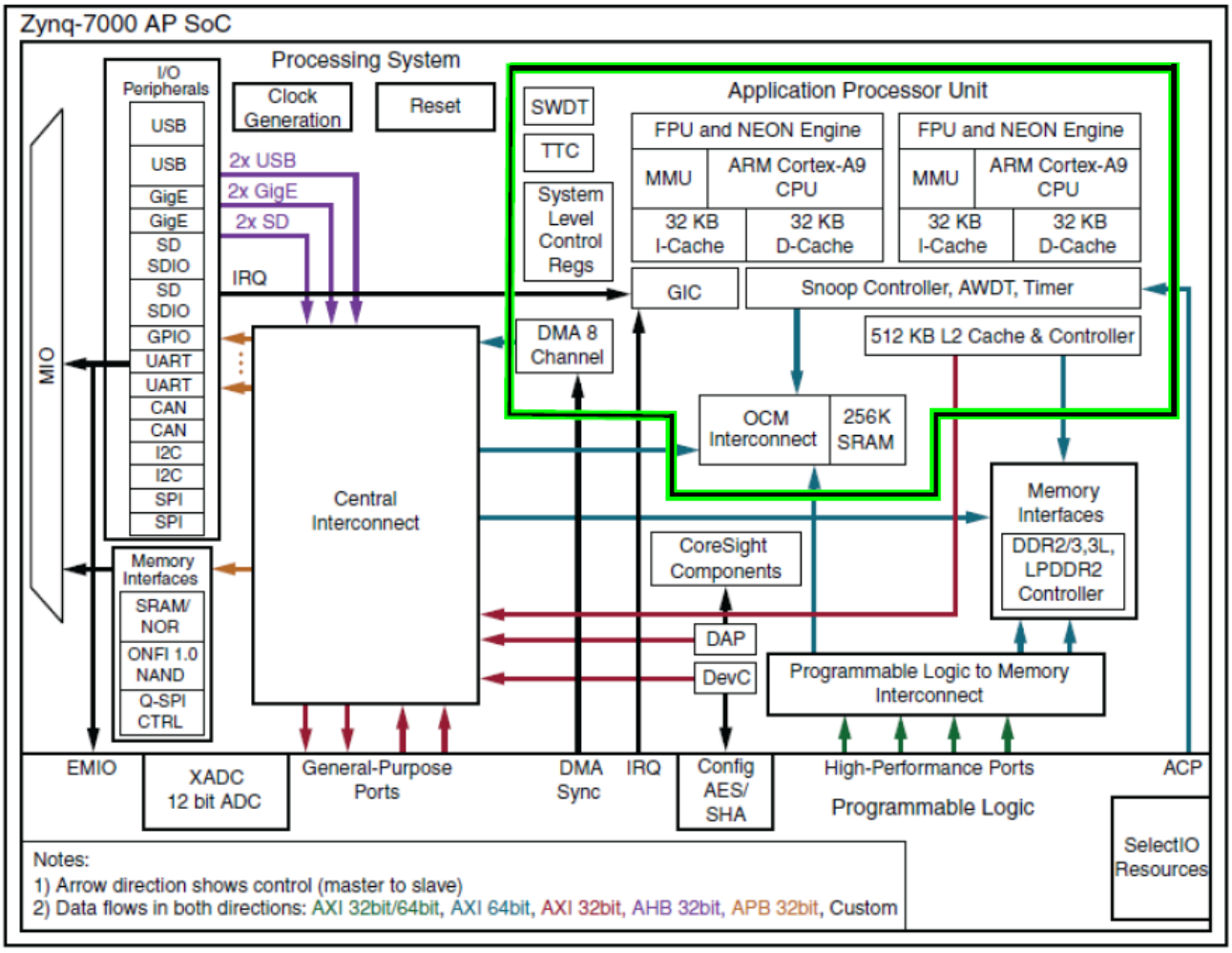
3.1.1 Procesorski sustav Zynq uređaja (PS)

Svi Zynq uređaji imaju istu bazičnu arhitekturu, te svaki u sebi sadržava dvojezgreni ARM Cortex-A9 procesor. Taj procesor je ugrađen sklop unutar Zynq uređaja, te postoji kao dedicirana i optimizirana komponenta.

Usporedbe radi, alternativa ugrađenom procesoru je programirani procesor, poput Xilinx Microblaze-a, koji je kreiran kombinacijom logičkih sklopova dijela programibilne logike. Stoga je implementacija procesora u smislu programiranja ekvivalentan izradi IP-a na programibilnom dijelu uređaja. Prednost implementacije procesora na programibilni dio je fleksibilnost u broju primjenjivih procesora. S druge strane, prethodno ugrađeni procesori mogu postići veću razinu efikasnosti.

Jedna od mogućnosti je i korištenje implementiranog procesora u suradnji s ugrađenim procesorom. Microblaze procesori mogu obavljati neke jednostavnije zadatke kao pomoć glavnom procesoru u svrhu raspodijele posla i rasterećenja glavnog procesora.

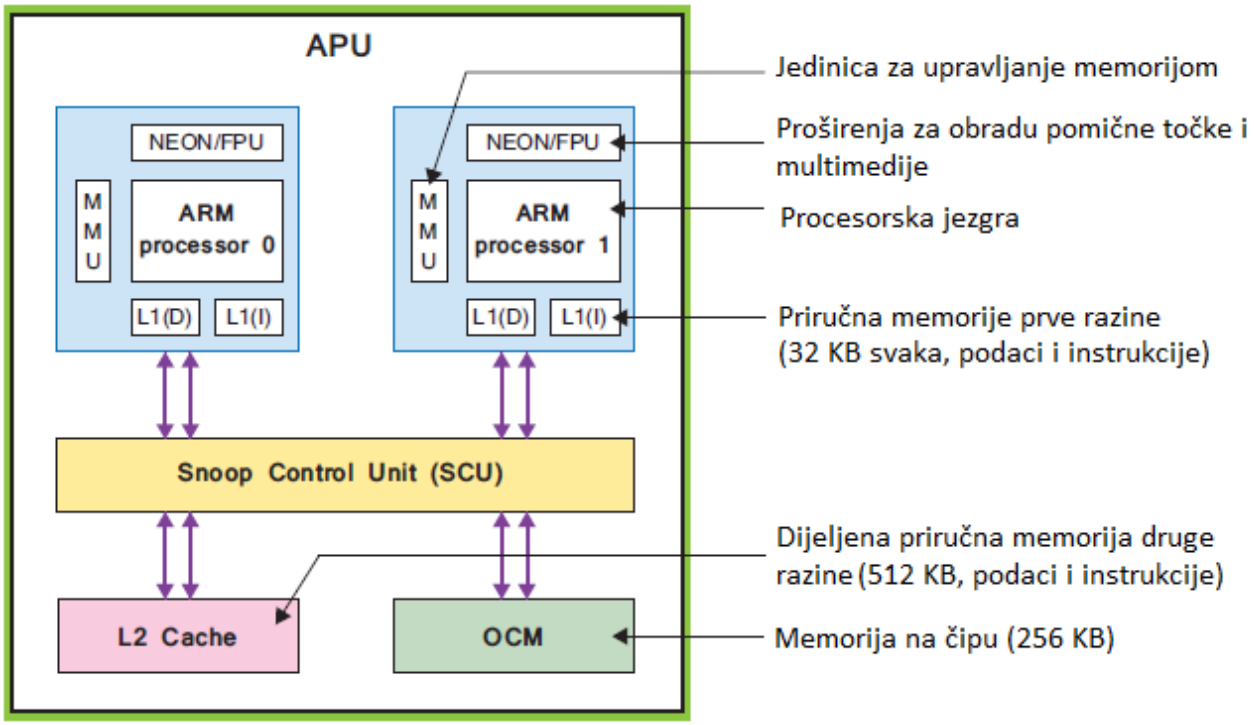
Zynq procesorski sustav nije samo ARM procesor, već set popratnih resursa koji skupa čine jedinicu za obradu aplikacija - APU *(engl. Application Procesing Unit)*, te ostale popratne komponente poput priručne memorije, memorijskih sučelja, poveznica, i generatora takta. Na Slici 3.2 prikazana je arhitektura procesorskog sustava, gdje je označen APU.



Slika 3.2 Zynq Arhitektura[2]

3.1.2 Jedinica za obradu aplikacija (APU)

Pojednostavljeni blok dijagram je prikazan na Slici 3.3. APU se sastoji od dvije ARM procesorske jezgre, od kojih svaka ima svoje dedicirane jedinice: NEON MPE, FPU, MMU te priručnu memoriju prve razine. APU također posjeduje i priručnu memoriju druge razine te udaljenu memoriju na čipu. SCU je poveznica između procesorskih jezgri i priručne memorije druge razine i udaljene memorije na čipu. ARM Cortex-A9 procesor može raditi na taktu do 1 GHz, ovisno o vrsti Zynq uređaja.

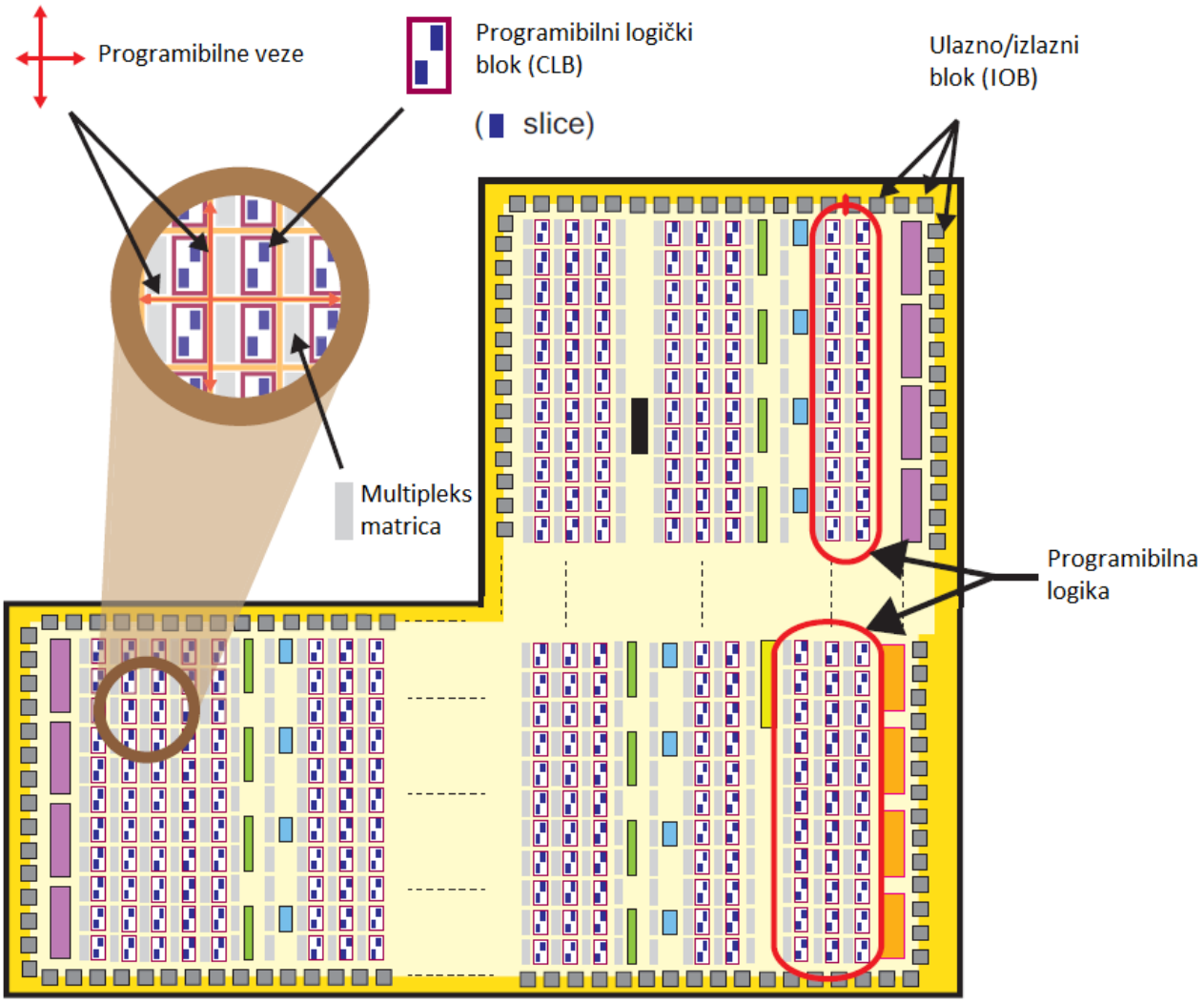


Slika 3.3 Blok dijagram jedinice za obradu aplikacija (APU)[2]

Iz perspektive programiranja procesora, podrška za ARM instrukcije je pružena u Xilinx *Software development kit* (SDK) programu, koji podržava sve potrebne komponente za razvoj softvera na spomenutoj arhitekturi. Kompajler podržava ARM *Thumb* instrukcijski set (16-bitni ili 32-bitni) uz 8-bitne Java byte-kodove.

3.1.3 Programibilna logika (PL)

Drugi dio koji se nalazi unutar Zynq arhitekture je programibilna logika bazirana na Artix-7 i Kintex-7 FPGA arhitekturi. Dio programibilne logike je dominantno popunjen s generičkim FPGA logičkim sklopovima, koji se sastoje od programibilnih logičkih blokova i isječaka (engl. slice), te ulazno/izlaznih blokova (IOB) koji služe svrhu sučelja.



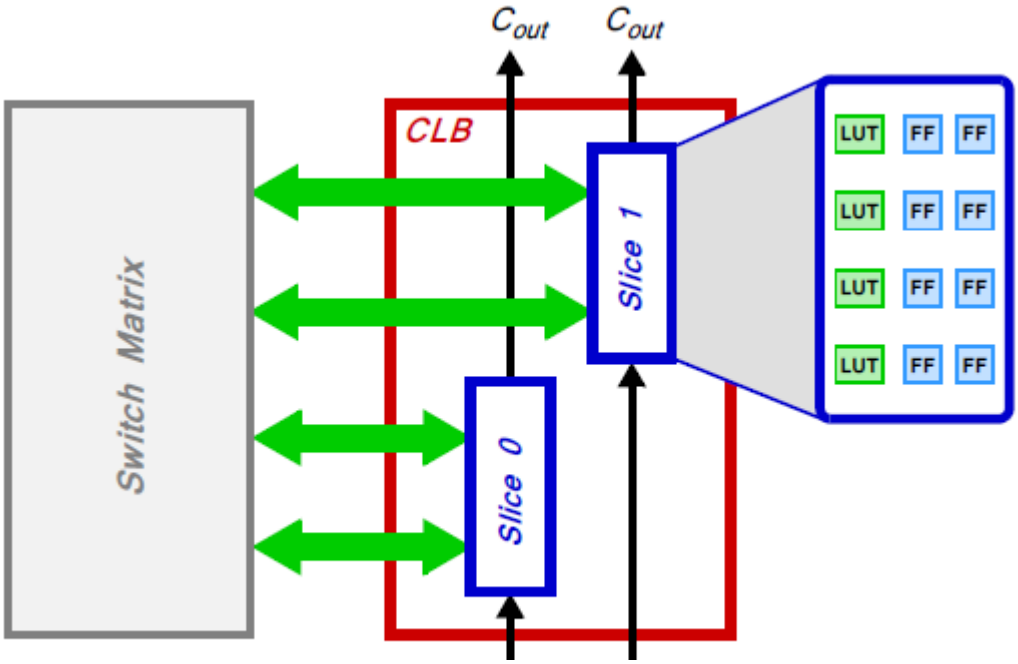
Slika 3.4 Programibilna logika i popratni elementi[2]

Na Slici 3.4 prikazane su slijedeće komponente:

- **Programibilni logički blokovi (CLB)** – male grupe logičkih sklopova koje su rasprostranjene u dvodimenzionalnom polju na programibilnom dijelu, te su spojeni na slične resurse pomoću

programibilnih veza. Svaki CLB se nalazi pored matrice za multipleksiranje i sadržava dva logička dijela

- **Isječak** – manja jedinica unutar CLB-a, koja sadrži resurse za implementaciju kombinacijske i sekvencijalne logike. Kao što je prikazano na Slici 3.5, Zynq isječak je sastavljen od 4 LUT-a, 8 bistabila i ostale logike
- **Pregledna tablica** *(engl. lookup table)* – fleksibilan resurs koji ima sposobnost implementacije logičkih funkcija do 6 ulaza, može služiti kao ROM, kao RAM ili pomični registar. Mogu biti kombinirane tako da tvore veće elemente navedenih funkcija
- **Bistabil** (FF) – sekvencijalni sklop koji implementira 1-bitni registar s reset funkcijom
- **Multipleks matrica** – nalazi se pored CLB-a, te pruža fleksibilno ostvarivanje veza između elemenata CLB-a, ali i ostvarivanje veza između CLB-a i ostalih resursa s programibilne logike
- **Prijenosna logika** – aritmetički sklopovi zahtijevaju prijenosne signale kako bi bili propagirani između susjednih isječaka. Sastoji se od lanca ruta i multipleksera za povezivanje isječaka po stupcima
- **Ulazno/izlazni blokovi (IOB)** – resurs koji pruža poveznicu između resursa programibilne logike i fizičkih izvoda korištenih za spajanje na eksterne komponente. Svaki IOB podržava 1-bitni ulazni ili izlazni signal. Uobičajeno su smješteni blizu kraja uređaja



Slika 3.5 Programibilni logički blok

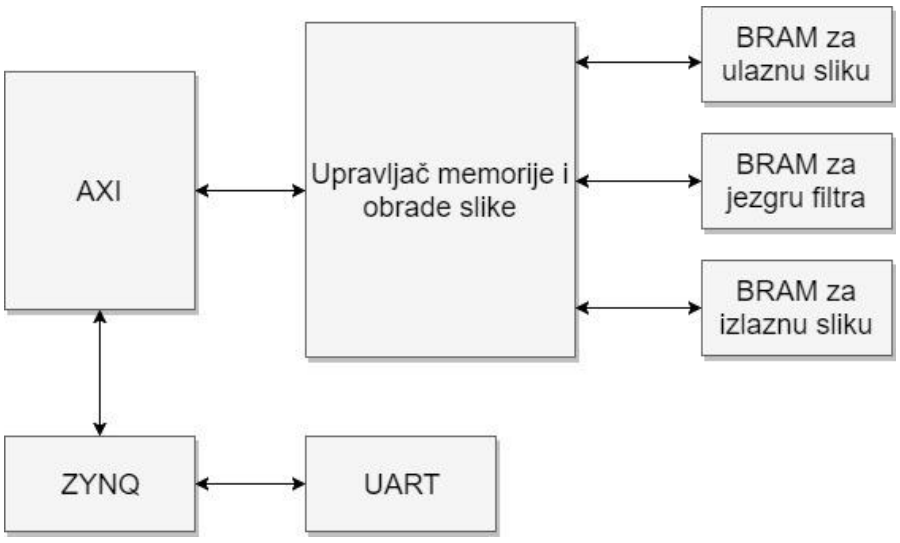
3.2 Programsko sučelje mikroupravljača

Kako bi se dizajnirao sustav za FPGA tehnologiju potrebno je koristiti Vivado programsko sučelje od tvrtke Xilinx. Za izvedbu sustava potrebno je koristiti nekoliko IP-a koje sadrži Vivado program, a koje je moguće vidjeti na slici 3.6. Potrebni IP su:

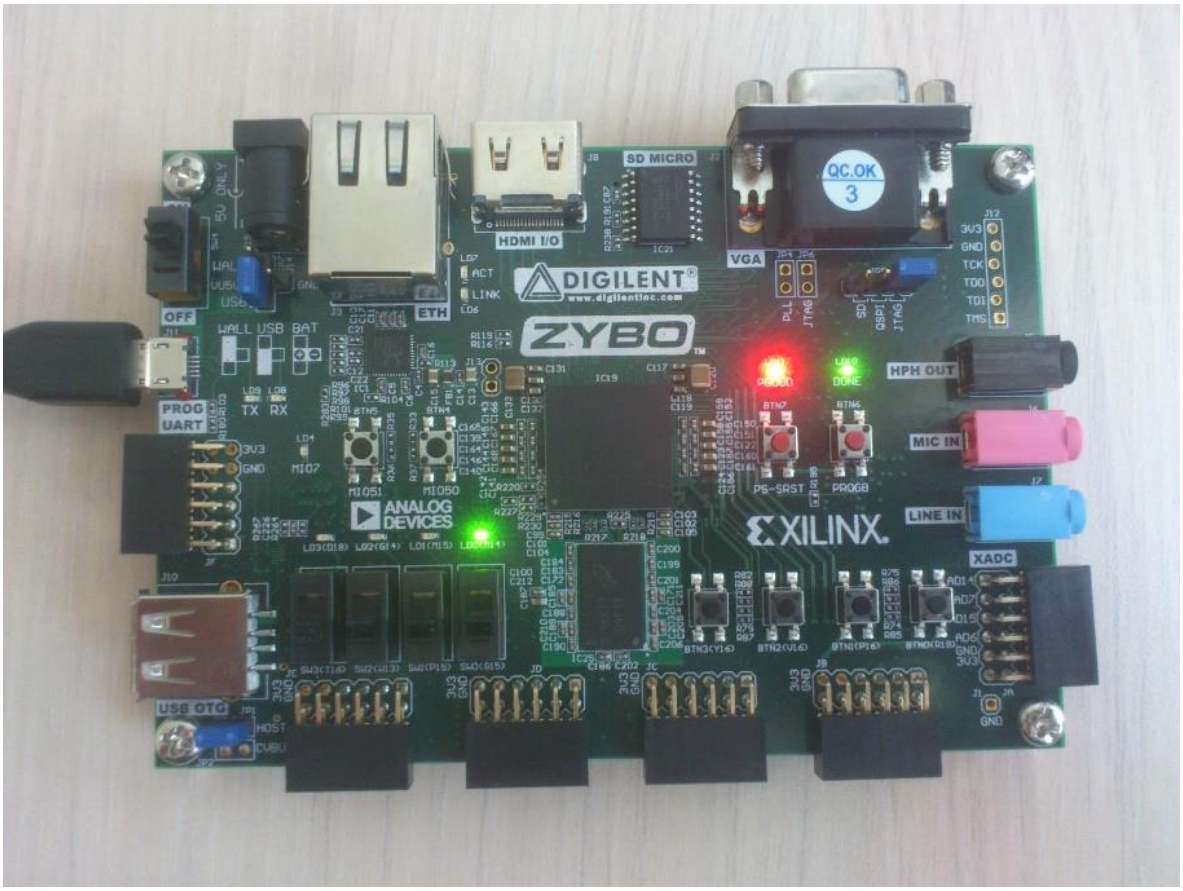
- Zynq procesorski sustav,
- AXI sabirnica,
- BRAM,
- Upravljač memorije i obrade slike
- DMA,
- DMA na BRAM IP.

3.2.1 Obrada slike iz BRAM-a

Sustav s BRAM-om radi na način da se slika namijenjena za obradu sprema u BRAM za ulaznu sliku, a jezgra Gauss filtra se sprema u BRAM za jezgru. Filtrirana slika se sprema u BRAM za izlaznu sliku. Upravljač memorije i obrade slike pristupa BRAM-u koji sadrži sliku za obradu te BRAM-u koji sadrži jezgru filtra. Zatim se obrađuju ulazni podaci nakon čega se obrađeni podaci spremaju u BRAM namijenjen za izlaznu filtriranu sliku. Detaljan opis rada upravljača memorije i obrade slike objašnjen je u daljnjem tekstu.



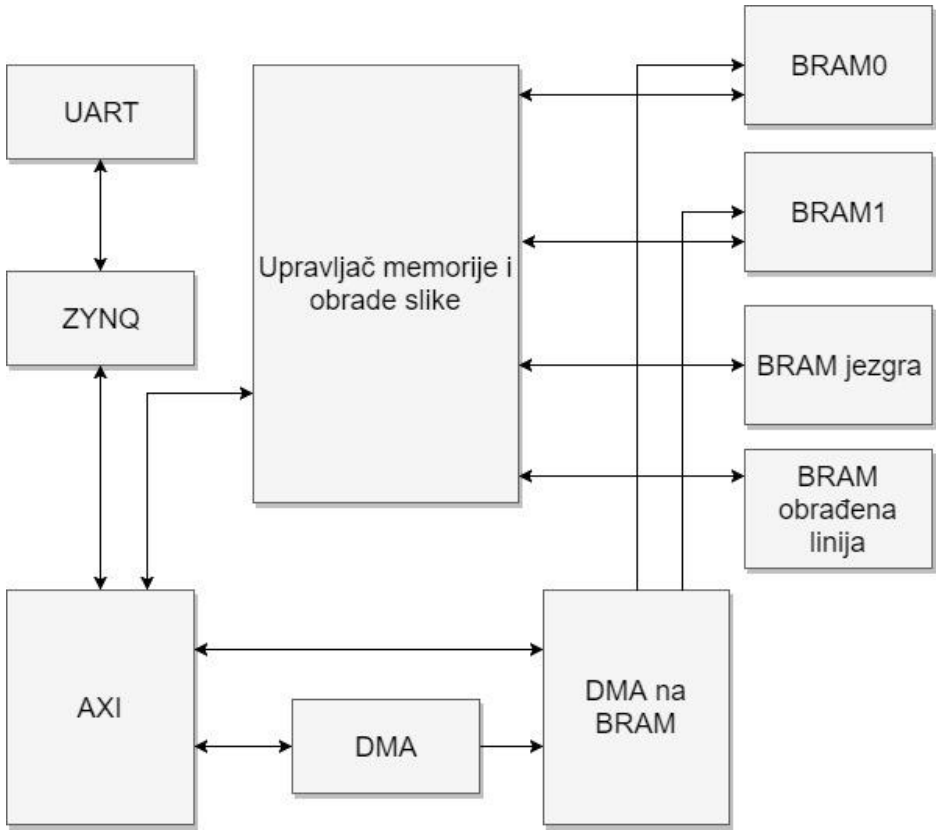
Slika 3.6. Blok shema sustava izvedenog s BRAM memorijom



Slika 3.7. Zybo razvojna pločica na kojoj je sustav implementiran i testiran

Na blokovskoj shemi sustava (Slika 3.6) moguće je vidjeti korištene IP *(engl. Intellectual Property)* jezgre Vivado programskog sučelja, te izlaze iz sustava. Na slici 3.7 je prikazana Zybo razvojna pločica na koju je implementiran sustav. Iz slike 3.3 se može vidjeti kako svijetli zelena LED (dioda LD1, nalazi se iznad prvog desnog prekidača), što znači da je sustav u stanju čekanja na obradu slike.

3.2.2 Obrada slike iz RAM-a



Slika 3.8. Blok shema sustava izvedenog s RAM memorijom

Na blokovskoj shemi sustava (Slika 3.8) obrade slike izvedenog s RAM-om mogu se vidjeti sve IP jezgre sustava osim RAM-a.

Sustav radi na način da se slika namijenjena za obradu sprema u RAM, te se potom preko DMA *(engl. Direct Memory Access)* upravljača šalje prvih 5 redova slike na BRAM 0, te nakon završetka prijenosa podataka upravljač memorije i obrade slike započinje obradu prvog reda slike. Upravljač memorije i obrade slike je isti upravljač koji se koristi za sustav izveden s BRAM-om ali je modificiran na način da obrađuje red po red slike. Za obradu jednog reda slike upravljaču je potrebno onoliko redova slike koliki je red jezgre filtra. U ovom slučaju red filtra je 5, te se u BRAM šalju podaci od 5 redova slike.

Nakon završetka prijenosa podataka i početka obrade slike započinje prijenos podataka slike u BRAM 1 paralelno s obradom slike. Podaci koji se šalju u BRAM 1 su također 5 redova slike, ali pomaknuti za jedan red slike. Kada se završi prijenos podataka u BRAM, programski kod čeka zastavicu koja

označava da je završila obrada prethodnog reda slike. Po završetku obrade jednog reda slike započinje se obrada slijedećeg reda slike iz BRAM-a 1, te se započinje prijenos podataka u BRAM 0. Prijenos podataka i obrada slike se ciklički izvršavaju sve dok se ne obradi cijela slika.

3.2.3 Upravljač memorije i obrade slike

Od navedenih IP-a potrebno je izraditi upravljačku jedinicu koja se povezuje na memorijske module, a komunicira s vanjskim svijetom preko AXI sabirnice povezane na mikroupravljač. Upravljačka jedinica dobavlja adrese pojedinih piksela i koeficijenata jezgre koji su u tom trenutku potrebni za obradu, preuzima podatke s memorijskih lokacija te ih obrađuje po primitku.

Adrese se prosljeđuju BRAM modulima koji zatim dobavljaju vrijednosti s traženih memorijskih lokacija. Ako se slika reprezentira u Kartezijevom koordinatnom sustavu tako da prvi piksel slike u gornjem lijevom kutu ima vrijednost (0,0), tada se lako mogu izračunati memorijske adrese potrebne za obradu. Obrada pojedinog piksela zahtjeva praćenje tri odvojene vrijednosti apscisa i ordinata koordinatnog sustava. Vrijednosti koordinata koje se prate su sljedeće:

- položaj piksela koji se obrađuje za izlaznu sliku,
- položaj piksela ulazne slike koji se množi s pripadajućim koeficijentom iz jezgre,
- položaj koeficijenta u jezgri koja se množi s pripadajućim pikselom.

Za opći primjer dobavljanja memorijskih adresa koristi se poopćena formula za pronalaženje adrese trenutnog koeficijenta jezgre. Formula je sljedećeg oblika:

$$adresa = x + (y \cdot red_jezgre) \tag{3-1}$$

gdje je 'red_jezgre' red matrice jezgre, 'y' je trenutni 'j' redak u matrici na kojem se nalazi koeficijent za obradu, a 'x' je trenutni 'i' stupac u matrici. Izračunom se dobije adresa, odnosno memorijska lokacija koeficijenta koji se trenutno koristi u obradi. Za dobivanje adrese piksela slike potrebno je umjesto konstante 'red_jezgre' koristiti konstantu 'visina_slike'. Ovakav način pronalaženja adresa je potreban jer se slika u memoriji nalazi kao jednodimenzionalni niz.

Nakon pronalaska potrebnih adresa moguće je preuzeti vrijednosti iz memorije s tih memorijskih adresa te ih poslati na obradu. Za jezgru reda 3x3 množi se 9 vrijednosti matrice jezgre s pripadajućim pikselima sa slike koja se obrađuje.

Obrada izlaznog piksela se obavlja unutar kontrolnog modula pomoću sljedeće formule:

$$izlazni_piksel = izlazni_piksel + (piksel_za_obradu \cdot koeficijent_jezgre) \quad (3\text{-}2)$$

gdje je izlazni_piksel vrijednost trenutnog piksela koji se obrađuje, piksel_za_obradu trenutno korištena vrijednost piksela ulazne slike koja se obrađuje, a koeficijent_jezgre je trenutno korišteni koeficijent iz jezgre.

Kako bi se upravljač mogao koristiti na Zybo razvojnoj pločici potrebno je isti prilagoditi za AXI protokol u svrhu komunikacije s procesorskim dijelom SoC-a. Upravljač je prilagođen tako da se preko procesorskog dijela može upravljati s određenim načinima na koje upravljač radi. Postoje tri načina rada upravljača, a to su:

- Upisivanje slike u BRAM,
- Ispisivanje slike iz BRAM,
- Obrada slike.

Na slici 3.9 je prikazana hijerarhijska struktura IP jezgre upravljača memorije i obrade slike. Iz hijerarhijske strukture vidljivo je da je potrebno izrađeni upravljač spojiti na automatski generirane AXI instance tako da se upravljač instancira unutar S00 AXI instance te se signali proslijede



Slika 3.9. Hijerarhijska struktura IP jezgre upravljačke jedinice memorije i obrade slike

određenim registrima, odnosno portovima. Također je potrebno korisnički kreirane portove proslijediti vršnom elementu strukture da bi se prikazali na kreiranom IP.

Kako bi procesor mogao slati naredbe potrebno je pristupati nultom registru AXI sabirnice. Upravljač je podešen na takav način da se čitaju vrijednosti tog registra, te se te vrijednosti prosljeđuju

odgovarajućem portu na upravljač. Vrijednosti tog registra ne mogu biti bilo koje, već se postavljaju tako da se postavljaju željeni bitovi, ovisno o željenom radu upravljača. Na primjer, za pokretanje obrade slike potrebno je postaviti šesti bit u registru na logičku jedinicu, što je u heksadecimalnom zapisu jednako 0x00000040. Potrebno je napomenuti da procesor mora nakon slanja spomenute naredbe za pokretanje obrade postaviti vrijednost upravljačkog registra na nulu kako se obrada slike ne bi ciklički izvodila. Ostale vrijednosti koje je moguće izravno kontrolirati su direktno vezane na određene BRAM. Značenja bitova u nultom registru za kontrolu upravljača su (počevši od nultog bita u registru):

0. Način ispisa slike,
1. Način upisa slike,
2. Omogući BRAM koji sadrži sliku za obradu,
3. Omogući upis u BRAM koji sadrži sliku,
4. Omogući BRAM koji sadrži obrađenu sliku,
5. Omogući upis u BRAM koji sadrži obrađenu sliku,
6. Pokreni obradu slike.

Kontrolne bitove je potrebno grupirati u grupe korištenja jer se za ispis slike koristi samo BRAM u kojemu se nalazi filtrirana slika, pa se stoga moraju zajedno koristiti nulti, četvrti i peti bit. Također, moraju se zajedno koristiti prvi, drugi i treći bit za ispis obrađene slike. Grupiranje bitova je potrebno jer je upravljač konstruiran na takav način da ako se označi bit za upisivanje slike, tada se signali prosljeđuju samo BRAM-u koji će sadržavati sliku za obradu. Isto vrijedi i za ispis slike, odnosno ako se označi bit za ispisivanje slike, omogućeno je samo pristupanje BRAM-u koji sadrži obrađenu sliku.

3.2.4 Kvantizacija jezgre

Upravljač je konstruiran na način da prima cjelobrojne koeficijente jezgre. Koeficijenti jezgre Gauss filtra množe se skalarom kako bi se izbjeglo množenje brojeva s fiksnom decimalnom točkom koji zahtijevaju veće računalne resurse.

Neophodne operacije prilikom filtriranja podrazumijevaju operacije zbrajanja i oduzimanja između podataka slike i podataka jezgre. Jezgra filtra je normalizirana matrica koja prvenstveno sadrži brojeve između 0 i 1. Koeficijenti matrice se dobivaju uz pomoć funkcije MATLAB programskog paketa naziva *fspecial*. Funkcijom se odabire izračun željenog 2D filtra, prilikom čega se odabiru parametri reda matrice filtra, kao i željena sigma vrijednost. Korištene vrijednosti jezgre filtra u ovom diplomskom radu prikazane su na Slici 3.10.

```
>> G = fspecial('gaussian', [5 5], 5)

G =

    0.0369    0.0392    0.0400    0.0392    0.0369
    0.0392    0.0416    0.0424    0.0416    0.0392
    0.0400    0.0424    0.0433    0.0424    0.0400
    0.0392    0.0416    0.0424    0.0416    0.0392
    0.0369    0.0392    0.0400    0.0392    0.0369
```

Slika 3.10 Ispis funkcije fspecial MATLAB programskog paketa

Kako bi se vrijednosti koeficijenata jezgre pripremili za obradu, potrebno je koristiti ugrađenu funkciju iz MATLAB programskog paketa naziva *quantizer*. Funkcija služi za namještanje parametara koji zaokružuju koeficijente dobivene izračunom 2D Gaussian funkcijom na željeni način. U ovom slučaju potrebni koeficijenti se odsijecaju prema gore, te se koristi mogućnost postavljanja svakog koeficijenta na najbliži djeljivi broj s nekim od eksponenata broja 2 (binarnog sustava). Što je veći eksponent, koeficijent ima bližu vrijednost nakon dijeljenja. Za potrebe ovog diplomskog rada, vrijednost eksponenta iznosi 12.

Nakon postavljanja kvatizatora, potrebno je pokrenuti funkciju *quantize*. Navedena funkcija uzima vrijednosti koje su podešene uz pomoć funkcije *quantizer* te ih upotrebljava za kvantizaciju željene

matrice. U ovom slučaju kvantizira se matrica jezgre Gauss filtra. Ispis funkcije quantize nalazi se na Slici 3.11.

```
>> quant = quantizer('ufixed', 'floor', 'saturate', [12 11])
>> Gq = quantize(quant, G)

Gq =

    0.0366    0.0391    0.0396    0.0391    0.0366
    0.0391    0.0415    0.0420    0.0415    0.0391
    0.0396    0.0420    0.0430    0.0420    0.0396
    0.0391    0.0415    0.0420    0.0415    0.0391
    0.0366    0.0391    0.0396    0.0391    0.0366
```

Slika 3.11 Ispis funkcije *quantize*

Kako je vidljivo na Slici 3.11, koeficijenti matrice su u decimalnoj reprezentaciji. Vidljive koeficijente je naposljetku potrebno pomnožiti sa skalarom 2^{12} kako bi se dobile vrijednosti koeficijenata koje je moguće koristiti za filtriranje. Koeficijenti jezgre prikazani su na Slici 3.12.

```
>> Gq_jezgra = Gq * 2^12

Gq_jezgra =

   150   160   162   160   150
   160   170   172   170   160
   162   172   176   172   162
   160   170   172   170   160
   150   160   162   160   150
```

Slika 3.12 Koeficijenti matrice jezgre filtra

3.3 Programsko sučelje stolnog računala

Za razvoj programskog sučelja stolnog računala se koristi Xilinx SDK razvojni program. U navedenom razvojnom programu se piše C programski kod koji upravlja procesorskim dijelom, te omogućava ispis rezultata u vanjski svijet, ali i kontrolu sustava iz vanjskog svijeta.

3.3.1 Programsko sučelje izvedeno s BRAM-om

Programsko rješenje napisano za sustav izveden s BRAM-om upravlja prihvatom slike, početkom obrade, izvedbom mjerenja te ispisom obrađene slike.

Slika koja se obrađuje šalje se preko UART protokola na Zybo razvojnu pločicu. Na početku C programa šalje se informacija o veličini slike na upravljač memorije, nakon čega se šalje informacija o početaku prihvata slike za obradu. Slika za obradu se šalje u obliku pojedinačnih vrijednosti piksela koji su odvojeni znakom za novi red. Broj 256 označava završetak prijenosa slike. Prilikom prihvata vrijednosti piksela, program prepoznaje pojedinačne piksele koje potom šalje preko AXI protokola na upravljač, koji te informacije zatim prosljeđuje direktno na BRAM u koji se sprema slika za obradu. Nakon završetka primanja i spremanja slike, programski kod šalje informaciju upravljaču za početak obrade. Odmah nakon početka obrade počinje mjerenje vremena obrade. Upravljač započinje obradu slike, te po završetku obrade šalje informaciju preko AXI protokola procesoru da je obrada završila. Upravljač istovremeno pali i LED na razvojnoj pločici za dodatnu informaciju o završetku. Procesor prepoznaje završetak obrade te zaustavlja mjerenje vremena, izračunava i ispisuje izmjereno vrijeme u konzolnom prozoru. Nakon obavljenih operacija programski kod šalje naredbu za ispisivanje podataka obrađene slike.

Na slici 3.13 prikazan je ispis konzolnog prozora. Zeleni brojevi u konzolnom prozoru su poslane vrijednosti piksela slike koja se šalje na obradu. Kao što je gore navedeno, broj 256 označava završetak prijenosa slike.

```
219
255
255
255
255
256
256
pocinjem upis u BRAM...
Pocinjem obradu i mjerenje...
Gotova obrada.
Obrada slike je trajala 121.50 ms.
Unesite sliku:
```

Slika 3.13. Ispis konzolnog prozora nakon unosa slike te pokretanja obrade

3.3.2 Programsko sučelje izvedeno s RAM-om

Programsko rješenje napisano za sustav izveden s RAM-om upravlja prihvatom slike, prijenosom podataka, određivanjem BRAM memorija za korištenje, početkom obrade, kontrolom obrade, izvedbom mjerenja te ispisom obrađene slike.

Na početku programskog rješenja potrebno je definirati veličinu slike koja se predaje na obradu. Zatim je potrebno unijeti sliku, nakon čega se započinje prijenos podataka preko UART sučelja. Isto kao u prethodnom programskom rješenju, potrebno je na kraju svih podataka označiti kraj podataka slike brojem 256. Primljeni podaci se spremaju u RAM, te ondje čekaju daljnju obradu. Nakon računanja količine podataka za slanje i pomaka adrese, prelazi se na obradu slike. Programsko rješenje je dizajnirano na način da se podaci iz RAM-a šalju u paketima od 128 bajta preko DMA na određeni BRAM kako bi tamo čekali obradu. Nakon što se prvi podaci prenesu u namijenjeni BRAM, procesorski sustav pokreće obradu. Početkom obrade, mijenjaju se pokazivači za upis/ispis iz BRAM-a, pomiče se pokazivač trenutnog reda koji se obrađuje, te se započinje prijenos podataka u drugi BRAM koji se trenutno ne koristi u obradi podataka. Programski kod čeka s nastavljanjem sve dok upravljač memorije i obrade slike ne postavi zastavicu da je završila obrada jednog reda slike, nakon čega se podaci obrađenog reda zapisuju u RAM. Završetkom upisa reda u RAM nastavlja se s obradom slike, ali se obrada podataka vrši iz drugog BRAM-a u kojemu se nalaze podaci od slijedećem redu slike koji je namijenjen za obradu. Ovaj dio programskog koda je smješten u dvije ugniježđene petlje. Unutarnja petlja služi za praćenje slanja svih paketa pri prijenosu podataka s DMA na BRAM, dok vanjska petlja prati trenutni red slike koji se obrađuje te se kreće u rasponu od nultog reda slike, pa sve do visine slike.

U tekstu iznad spominje se prijenos podataka s DMA na BRAM, što je potrebno pobliže prikazati kako se odvija. Naime, prijenos podataka zahtijeva pozivanje funkcije za prijenos podataka iz biblioteke xaxidma.h. Funkcija prima četiri parametra koji su potrebni za početak prijenosa podataka. Parametri koje prima funkcija su:

- Instanca DMA resursa,
- Pokazivač na spremnik u kojemu se nalaze podaci za slanje,
- Količina podataka za slanje,
- Smjer prijenosa podataka.

Instanca DMA resursa je potrebna kako bi funkcija mogla prepoznati koji DMA upravljač koristiti. Pokazivač na spremnik *(engl. buffer)* je pokazivač koji pokazuje na početnu adresu u memoriji na kojoj se nalaze podaci. Podaci se spremaju u spremnik direktnim kopiranjem željenih podataka iz RAM-a, a količina podataka za slanje postavlja ograničenje količine podataka za prijenos. Varijabla

'smjer prijenosa podataka' označava smjer u kojemu će se slati, odnosno primati podaci, a može imati samo dva smjera. Smjerovi prijenosa podataka su s DMA na BRAM i s BRAM na DMA. U programskom rješenju ovog diplomskog rada koristi se isključivo jedan smjer prijenosa podataka, a to je prijenos podataka s DMA u BRAM.

Ispis linije slike je također funkcija napravljena u sklopu programskog rješenja koja se koristi prilikom ispisivanja jednog obrađenog reda slike iz BRAM-a namijenjenog za obrađeni red. Funkcija prima samo jedan parametar koji sadrži vrijednost pomaka adrese. Pomak adrese je potreban kako bi se znalo na koje mjesto u RAM je potrebno spremiti podatke, odnosno ako se radi o prvom redu slike koju je potrebno spremiti u memoriju, tada je potrebno napraviti pomak adrese za jednu širinu slike. Pomak adrese se može dobiti iz jednostavne formule koja množi podatak o trenutnom redu slike sa širinom slike kako bi se dobio pomak za bilo koji red koji je potrebno spremiti.

$$pomakAdrese = trenutnaLinijaObrade \cdot (sirinaSlike + 4) \quad (3\text{-}3)$$

U formuli navedenoj iznad može se vidjeti da je širina slike uvećana za 4 iz razloga što je jezgra filtra reda 5. Kako je sliku potrebno obraditi i do rubova, slika se mora proširiti sa svake strane za red jezgre umanjen za jedan, odnosno

3.3.3 AXI standard

Ključ upotrebe programibilne logike u suradnji s procesorskim sustavom u svrhu tvorenja cjelovitih ugrađenih sustava je brza komunikacija između navedenih sustava. Za te potrebe je osmišljen standard koji se naziva AXI poveznica *(engl. AXI interconnect)* i sučelje koji stvara poveznicu između ta dva dijela SoC-a.

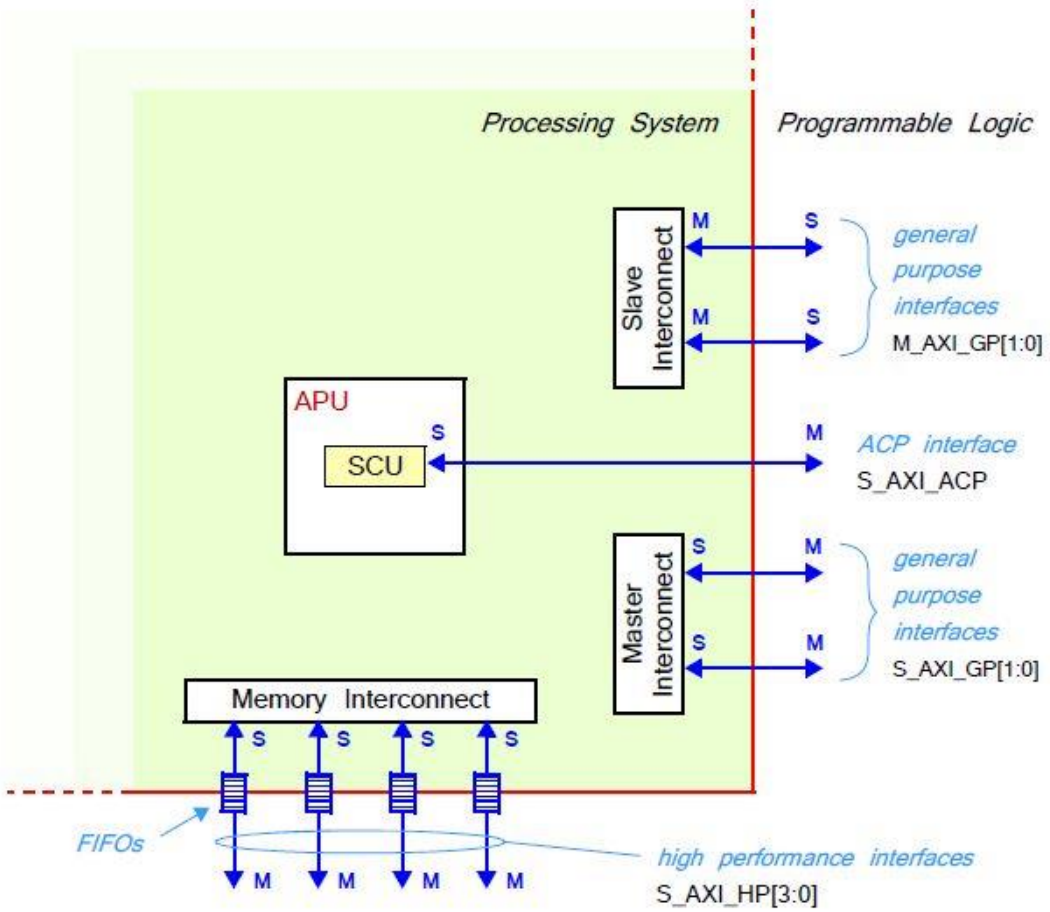
AXI je kratica za naziv napredno proširivo sučelje *(engl. Advanced eXtensible Interface)*. Trenutna najnovija verzija tog standarda naziva se AXI4 verzija, a dio je ARM AMBA 3.0 standarda. Mnogi uređaji i IP blokovi izrađeni od treće strane koriste ovaj standard.

AMBA standard je razvijen od strane ARM-a 1996. godine u svrhu upotrebe u mikroupravljačkim sustavima. Danas je standard unaprijeđen do razine da ga ARM naziva „nativnim standardom za komunikaciju na čipu“. Trenutno se radi fokus na SoC tehnologiju koja se temelji na FPGA, gdje tvrtka Xilinx pridonosi definiranjem AXI4 kao optimalne tehnologije povezivanja u FPGA arhitekturama.

Najčešća uporaba AXI sabirnice je prilikom povezivanja IP blokova s procesorskim dijelom sustava unutar ugradbenog sustava. Za AXI4 standard postoje tri vrste protokola koje se koriste unutar ugrađenog sustava. Ovisno o zahtjevima za konekciju koristi se određeni protokol. AXI 4 protokoli su:

- AXI4 – koristi se za povezivanje memorijskih mapa, te pruža najbolje performanse. Prosljeđuje se adresa nakon koje slijedi tok podataka *(engl. data burst)* maksimalnog kapaciteta do 256 podatkovnih riječi bez dodatnog slanja adrese
- AXI4-Lite – pojednostavljeni AXI4 link koji podržava samo jedan prijenos podataka po konekciji (bez toka podataka). Ovaj protokol isto koristi memorijske mape, ali u ovom slučaju se odvija prijenos adrese te jednog podatka.
- AXI4-Stream – koristi se u svrhu toka podataka *(engl. data stream)* velike brzine, a podržava tokove neograničene brzine. U ovom protokolu se ne koriste adresni mehanizmi već se predaje početna adresa prijenosa te količina slijednih podataka nakon te adrese namijenjenih za prijenos.

U navodima iznad se koristi pojam 'memorijska mapa', što je potrebno pobliže opisati. Ako protokol koristi memorijske mape, tada nadređeni *(engl. master)* mora slati adresu koja se nalazi unutar memorijskog adresnog prostora. U slučaju AXI4-Lite protokola koji podržava prijenos jednog podatka po konekciji, podaci se pišu u, odnosno čitaju s tražene adrese. U slučaju AXI4 toka podataka, za prijenos podataka potrebna je početna adresa prvog podatka koji se šalje, te količina podataka koji



se šalju, nakon čega podređeni *(engl. slave)* mora računati adresu svakog slijedećeg dolaznog podatka. Na Slici 3.14 mogu se vidjeti dva ključna pojma, a to su poveznica i sučelje:

Slika 3.14 Struktura AXI poveznica i sučelja[2]

- Poveznica – preklopnik koji se brine o usmjeravanju podataka između povezanih AXI sučelja. Unutar procesorskog sustava postoji nekoliko poveznica, od kojih su neke povezane direktno na dio koji sadržava programibilnu logiku, dok se ostali koriste samo unutar procesorskog sustava.

- Sučelje – direktna veza od točke do točke *(engl. point-to-point)* za prosljeđivanje podataka, adresa, i razne pomoćne signale za komunikaciju između nadređenog i podređenog unutar sustava.

3.3.4 Blok RAM

Karakteristike blok RAM-a u Zynq-7000 arhitekturi su:

- 36 KB blok RAM s dvostrukim portom te širinom porta do 72
- Programabilna FIFO logika
- Ugrađen opcionalan sklop za ispravke pogrešaka

Svaki SoC gore navedene arhitekture sadrži između 60 i 465 blok RAM-a s dvostrukim portom od kojih svaki može spremiti do 36 KB podataka. Svaki blok RAM podržava dva odvojena porta koji dijele samo podatke, s time da obije strane mogu imati promjenjive i/ili različite širine porta. Na SoC-u korištenom u diplomskom radu se nalazi 240 KB blok RAM-a.

Takva memorija radi u sinkronom načinu rada na način da je svaki pristup memoriji kontroliran taktom. Adresa koja se predaje memoriji je uvijek sinkronizirana, a podaci s te adrese se zadržavaju sve do slijedeće operacije. Postoji i mogućnost povećanja frekvencije takta, ali to dolazi po cijeni latencije na izlaznim podacima.

Svaki blok RAM može generirati osam dodatnih bitova Hammingovog koda i obaviti jednobitnu ili dvobitnu ispravku pogreške nad izlaznim podacima.

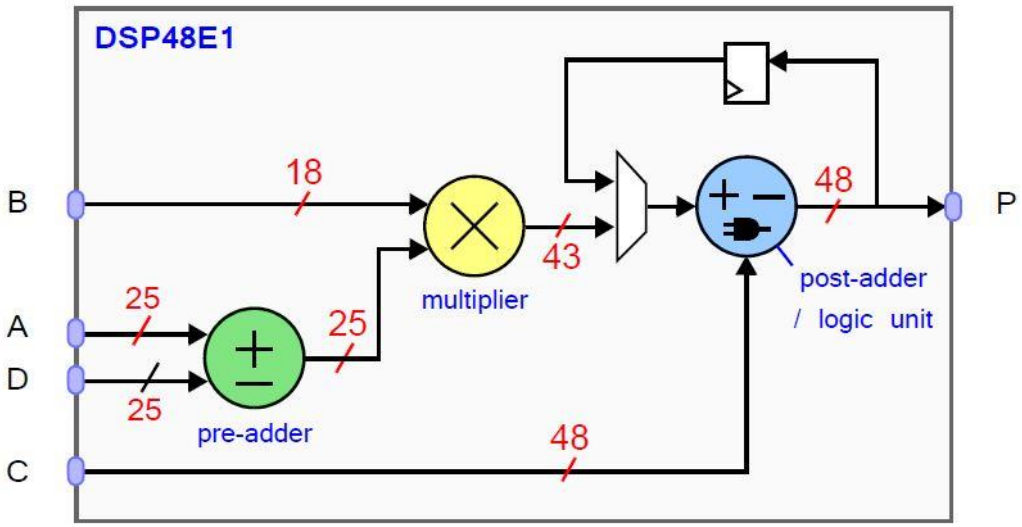
Svaki port memorije može biti konfiguriran tako da radi u načinu 'prvo čitaj' *(engl. read first)* ili 'prvo piši' *(engl. write first)*. Kada je port konfiguriran kao 'prvo čitaj', ulazni podaci su istovremeno zapisani u memoriju te proslijeđeni na izlaz. Kada je port konfiguriran kao 'prvo piši', ako je zastavica koja omogućuje upis podataka omogućena, prvo pročita stare podatke s memorijske adrese te ih ispiše kao izlazni podatak, a zatim spremi nove podatke na odabranu memorijsku adresu (pročitaj prije zapisivanja).

Blok RAM može biti konfiguriran kao FIFO memorija. U navedenoj arhitekturi postoji dodatna logika koja omogućava dva porta za FIFO memoriju koji mogu biti sinkroni (na istom taktu), te asinkroni (na dva odvojena takta). Ako se blok RAM konfigurira na takav način, tada je port A memorije

korišten kao port za čitanje, a port B kao port za pisanje. Također, u takvom načinu rada nije moguće postaviti različite širine portova A i B.

3.3.5 DSP48E1

To je komponenta posebne namijene koja se koristi u svrhu vrlo brze aritmetike. Takva komponenta se sastoji od predzbrajala/oduzimala, množila te naknadnog zbrajala/oduzimala, što je prikazano na Slici 3.15.



Slika 3.15 Struktura DSP48E1 komponente[2]

Na Slici 3.7 su također prikazane širine riječi koje se mogu obraditi. Takve širine su primjenjive u većini aplikacija, no u slučaju potrebe za većom širinom riječi moguće je povezati više istih komponenti kako bi se povećala širina. Kompleksnija aritmetika je također izvediva povezivanjem više takvih komponenti, a širine riječi omogućavaju i implementaciju aritmetike pomične točke. Takvi sklopovi imaju prednost te su primamljivi za primjenu u aplikacijama koje imaju zahtjevne aritmetičke operacije zbog male iskoristivosti energije te visokog radnog takta. Također, koriste se kod izrade filtara i u digitalnoj obradi signala zbog svojstava kompleksnih aritmetičkih zahtjeva, a najčešće je korištena kod implementacije filtra beskonačnog impulsnog odziva. Optimalna uporaba navedene komponente je kada se koristi u sprezi s blok RAM-om.

U ovom diplomskom radu se koriste 3 DSP48E1 komponente kod oba dizajna i to kod izračunavanja trenutno potrebnih memorijskih adresa, ali i kod izračunavanja vrijednosti trenutnog piksela. Korištenje navedene komponente omogućuje ubrzanje aritmetike dizajniranog sustava.

3.3.6 DMA

Jedan način rasterećenja procesora je korištenje DMA za prijenose koji uključuju memoriju. Koristeći ovakav pristup, procesor izdaje naredbu DMA upravljaču (DMAC) za prijenos iz memorije, koji potom izvodi prijenos. Ovakav način rada omogućuje procesoru da može izvoditi druge stvari dok se u pozadini odvija prijenos podataka. U tom slučaju DMAC služi funkciju i nadređenog i podređenog na sabirnici. Kao nadređeni, DMAC komunicira s upravljačem memorije dok ujedno i arbitrira za sabirnicu. Kao podređeni, DMAC postavlja prijenose u memoriji odgovaranjem na zahtjeve izdane od strane nadređenog za sabirnicu (u najčešćem slučaju to je procesor).

DMAC koristi 64-bitni AXI nadređeno sučelje, te radi na duplo većoj frekvenciji od procesora kako bi izveo DMA prijenose iz/u memoriju iz PL dijela čipa. Svi prijenosi se kontroliraju preko DMA pokretača izvođenja instrukcija *(engl. instruction execution engine).* DMA pokretač radi na uskom instrukcijskom setu naredbi koji omogućavaju jednostavne metode specificiranja DMA prijenosa podataka. Ova metoda pruža veliku fleksibilnost u usporedbi s mogućnostima metoda DMA upravljača.

Dio programskog koda za DMA pokretač zapisan je softverski u područje memorije kojemu pristupa upravljač koristeći AXI nadređeno sučelje. DMA pokretač izvođenja instrukcija uključuje instrukcije za DMA prijenose podataka i upravljanje instrukcijama za kontroliranje sustava. Upravljač može biti podešen tako da ima do najviše 8 DMA kanala. Svaki kanal je jedna nit koja se izvršava na procesorskoj jedinici DMA pokretača. Kada DMA nit izvrši 'učitaj' ili 'spremi' naredbu, DMA pokretač prosljeđuje memorijski zahtjev u odgovarajući red čekanja (red čekanja za pisanje, odnosno čitanje). DMAC koristi te redove čekanja kako bi se podaci smjestili u međuspremnik za potrebe prijenosa čitanja/pisanja preko AXI sučelja. Upravljač sadrži i više-kanalnu FIFO memoriju kako bi se spremili podaci tijekom DMA prijenosa podataka.

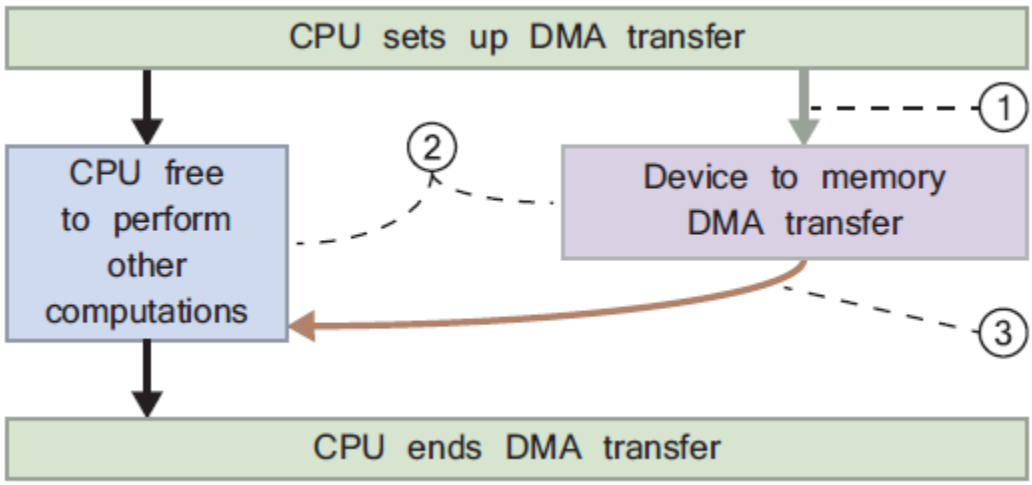
DMAC ima sposobnost prijenosa velike količine podataka bez intervencije procesora. Početna i odredišna točka u memoriji mogu biti bilo gdje na sustavu, bilo procesorskom dijelu, bilo programibilnom dijelu. Memorijska mapa za navedeni upravljač uključuje DDR, OCM, linearno adresiranu QSPI memoriju za čitanje, SMC memoriju te razne module s PL dijela čipa koji su spojeni na M_GP_AXI sučelje.

Upravljač ima dva seta kontrolnih i statusnih registara. Jedan set je dostupan u sigurnom načinu rada, a drugi u običnom načinu. Upravljač ne može raditi na dva načina rada istovremeno.

Metoda kontrole toka za prijenose koji uključuju memorije procesorskog sustava koriste AXI poveznicu. Pristup komponentama na programibilnoj logici može koristiti AXI kontrolu toka ili DMAC sučelje za zahtjeve prema periferiji.

Kako bi se pokrenuo prijenos, DMAC-u se moraju predati slijedeće informacije:

- Izvorišna adresa,
- Odredišna adresa,
- Duljina podataka koji se šalju.



Slika 3.16 Izvođenje DMA prijenosa podataka[2]

Na Slici 3.16 prikazan je proces DMA prijenosa podataka i memorije. Proces je slijedeći:

1. Procesor postavlja uređaj koji želi koristiti DMA prijenos podataka u memoriju izdajući DMA naredbu te onemogućavanjem svih DMA prekida,
2. DMA upravljač prenosi podatke s perifernog uređaja u memoriju ostavljajući procesor slobodnim za druge poslove,
3. Po završetku prijenosa podataka, šalje se prekid procesoru kako bi ga se informiralo o završetku prijenosa.

4. EKSPERIMENTALNI REZULTATI

Na razvijenom sustavu su provedena mjerenja za različite veličine slika, te su uspoređene brzine obrade korištenjem ulazne slike iz BRAM-a, iz RAM-a. Kao referenca provedenim mjerenjima, navedeni su rezultati iz MATLAB programskog paketa u kojemu se filtrirala slika istih dimenzija te s istim parametrima jezgre filtra.

Specifikacije sustava na kojemu se izvodila MATLAB skripta su slijedeće:

- Procesor: Intel Core i5 6600 (3.9GHz)
- Matična ploča: Asus H110M-K
- RAM: 2x4GB DDR4 2133MHz
- SSD: PNY CS1311 240GB
- Softver: Windows 10; MATLAB R2017a

		Trajanje obrade (ms):		
Veličina slike	Broj piksela	BRAM	RAM	MATLAB
300 x 300	90.000	121,5	51,47	0,45
1280 x 853	1.091.840	x	771,38	2,10
1663 x 935	1.554.905	x	1095,90	2,70
1920 x 1080	2.073.600	x	1182,52	3,40

Tablica 1. Trajanje obrade slike za jezgru filtra veličine 5x5

		Trajanje obrade (ms):		
Veličina slike	Broj piksela	BRAM	RAM	MATLAB
300 x 300	90.000	95,44	46,86	0,39
1280 x 853	1.091.840	x	558,70	1,80
1663 x 935	1.554.905	x	794,52	2,20
1920 x 1080	2.073.600	x	1059.11	3,00

Tablica 2. Trajanje obrade slike za jezgru filtra veličine 3x3

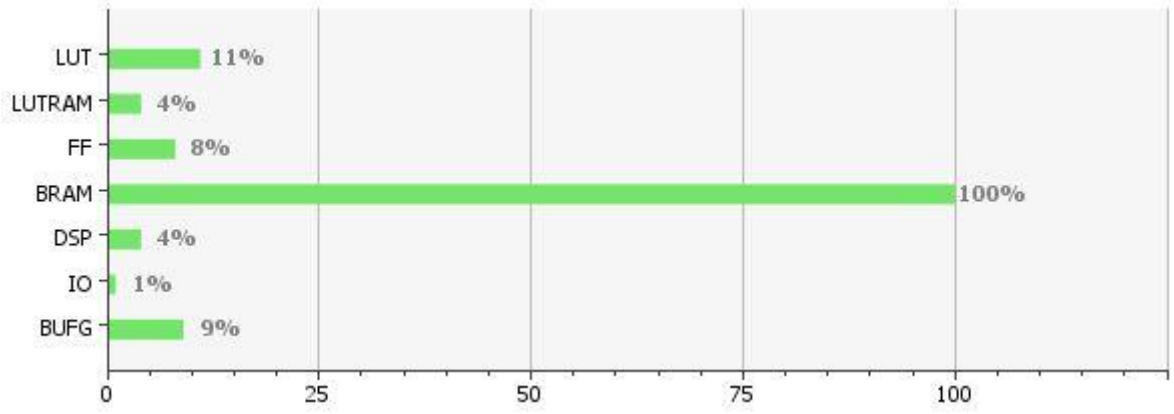
U tablicama je vidljivo trajanje obrade slike na pojedinom sustavu. Može se uvidjeti da se smanjenjem reda jezgre filtra smanjuje i trajanje obrade na svakom testiranom sustavu. To je i očekivano jer se smanjenjem reda jezgre smanjuje i broj potrebnih množenja i zbrajanja, s 25 množenja za jezgru reda 5, na 9 množenja za jezgru reda 3.

Na sustavu izvedenom isključivo s BRAM-om nije moguće filtrirati sliku veću od 340x340 jer Zynq uređaj na kojem je sustav testiran ima samo 240 KB BRAM-a. Teoretski je moguće filtrirati sliku do maksimalno 480x480, ali je to nepraktično, jer se u tom slučaju ne može spremati filtrirana slika, već bi se takva slika morala konstantno tijekom filtriranja slati preko AXI sučelja na procesor, što bi drastično povećalo vrijeme izvođenja. Stoga je filtrirana slika navedene veličine, te je vrijeme filtriranja uspoređeno s ostalim sustavima.

U Tablici 3 i Tablici 4 prikazane su iskoristivosti resursa FPGA čipa za obije izvedbe, a na Slici 4.2 i Slici 4.2 podaci su prikazani grafički. Može se uvidjeti kako je iskorištenost BRAM resursa na 100 % kod sustava izvedenog isključivo s BRAM-om, dok je kod sustava iskoristivost istog resursa na 14 %. Zbog dodatnog sklopa za prijenos podataka s DMA na BRAM može se vidjeti da druga izvedba sustava sa RAM-om zahtjeva približno 9 % više LUT-ova od prve izvedbe, te približno 4 % više bistabila. Iskorištenost DSP resursa je ista kod oba sustava jer se koristi ista komponenta za upravljanje memorijom i obradom slike. Navedena komponenta se kod oba sustava koristi pri ubrzanju izračunavanja vrijednosti izlaznog piksela i adresa.

Resurs	Iskorištenost	Raspoloživo	Iskorištenost %
LUT	1939	17600	11,01
LUTRAM	247	6000	4,11
FF	2722	35200	7,73
BRAM	60	60	100
DSP	3	80	3,75
BUFG	3	32	9,37

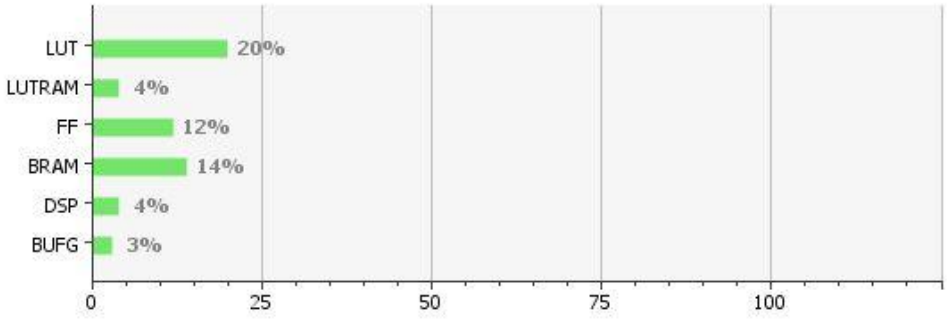
Tablica 3. Iskorištenost resursa sustava izvedenog s BRAM-om



Slika 4.1 Grafički prikaz iskorištenosti resursa sustava izvedenog s BRAM-om

Resurs	Iskorištenost	Raspoloživo	Iskorištenost %
LUT	3510	17600	19,94
LUTRAM	241	6000	4,01
FF	4262	35200	12,10
BRAM	8,5	60	14,16
DSP	3	80	3,75
BUFG	1	32	3,12

Tablica 4. Iskorištenost resursa za sustav izveden s RAM-om



Slika 4.2 Grafički prikaz iskorištenosti resursa sustava izvedenog s RAM-om

U Tablici 5 je vidljivo kako je približno jednaka potrošnja oba sustava, ali ipak na nekim komponentama dolazi do odstupanja. Do povećanja potrošnje dolazi kod AXI protokola u sustavu izvedenom sa RAM-om jer se na AXI sabirnicu spajaju još dvije dodatne IP komponente. Također dolazi do povećanja potrošnje i kod signala, ali kod sustava izvedenog s BRAM-om. Uspoređujući oba dizajna koji se nalaze u prilogu P.4.1 i P.4.2 vidljivo je da su komponente raširenije po čipu na sustavu izvedenom sa BRAM-om, gdje su potrebni dulji signali za povezivanje komponenti i BRAM-ova, te zbog toga dolazi do veće potrošnje.

BRAM sustav		RAM sustav	
	potrošnja energije (W)		potrošnja energije (W)
Procesor	0,27	Procesor	0,27
Memorijsko sučelje	0,884	Memorijsko sučelje	0,884
U/I sučelje	0,001	U/I sučelje	0,001
PLL	0,352	PLL	0,352
AXI	0,002	AXI	0,006
DSP	0,002	DSP	0,002
BRAM	0,007	BRAM	0,007
Programibilna logika	0,005	Programibilna logika	0,004
Signali	0,008	Signali	0,005
Takt	0,012	Takt	0,013
Statična potrošnja	0,136	Statična potrošnja	0,134
∑	**1,679**	∑	**1,678**

Tablica 5. Rezultati analize potrošnje energije za oba sustava

5. ZAKLJUČAK

Ovim radom prikazano je dva načina pristupa rješavanju problema filtriranja slike pomoću Gauss filtra na FPGA tehnologiji. Pristup filtriranju koristeći isključivo BRAM nije dao obećavajuće rezultate zbog ograničenosti BRAM-a na samo 240 KB, što je dalo ograničenje veličine slike na nešto više od 300 x 300 piksela. Drugi pristup je bolji, jer se ne mora brinuti o memorijskim zahtjevima. Slika se nalazi na RAM-u koji ima dovoljno memorijskih lokacija za spremanje više slika HD *(engl. High Definition)* kvalitete. Drugi pristup također koristi BRAM tehnologiju u obradi i to u svrhu dvostrukog međuspremnika za ulaznu sliku, što se pokazalo kao dobro rješenje jer se izbjegava čekanje na obradu za vrijeme prijenosa slike, već se obrada reda slike i prijenos podataka slijedećeg reda odvija paralelno. Iako se očekivalo da pristup obradi slike dvostrukim međuspremnikom zahtjeva isto ili duže vrijeme filtriranja slike istih dimenzija, no to nije slučaj, jer se slika filtrira u kraćem vremenu.

Diplomski rad je iskoristiv kao takav, no moguća su poboljšanja u blokovskoj izvedbi sustava. Jedno od mogućih poboljšanja je primjena istog principa za ispis slike kao što se koristi za sliku namijenjenu za obradu, odnosno dvostruki izlazni međuspremnik. U ovom rješenju se na spremanje obrađene slike potroši približno 45% vremena, što je moguće izbjeći navedenom idejom rješenja. Također, moguće je koristiti više upravljača memorije i obrade slike koji rade paralelno. Tako se postiže brža obrada slike, no tu se dolazi do fizičkih ograničenja primijenjene arhitekture.

LITERATURA

[1] C.H., Chen; U., Qidwai, *Digital Image Processing: An Algorithmic Approach with MATLAB*, Taylor & Francis Group, 2009.

[2] L.H., Crocket; R.A., Elliot; M.A., Enderwitz; R.W., Stewart, *The Zynq Book*, Strathclyde Academic Media, 2014.

[3] G.Lj., Đorđević, *Arhitekture mikrosistema*, 2009.

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

[5] F. Cabello; J. Leon; Y. Iano and R. Arthur, *„Implementation of a Fixed-Point 2D Gaussian Filter for Image Processing based on FPGA“*, Signal processing: Algorithms, Architectures, Arrangements, and Applications (SPA), pp, 28 – 33, Poznan, 2015.

[6] Slika korištena u obradi, [11.12.2017.] http://www.worldglitz.com/wonderful-zen-image

SAŽETAK

U ovom radu opisana su dva rješenja za filtriranje slike pomoću Gauss 2D filtra. Prvi pristup rješenju temelji se isključivo na BRAM-u. Drugi pristup je unaprijeđeno prvo rješenje, a izvedeno je preko RAM-a. U radu su opisane sve korištene komponente i protokoli, te je detaljno opisano na koji način radi pojedina konstruirana komponenta.

Ključne riječi: Filtriranje slike, Gauss filtar, Zynq, BRAM, RAM, AXI.

ABSTRACT

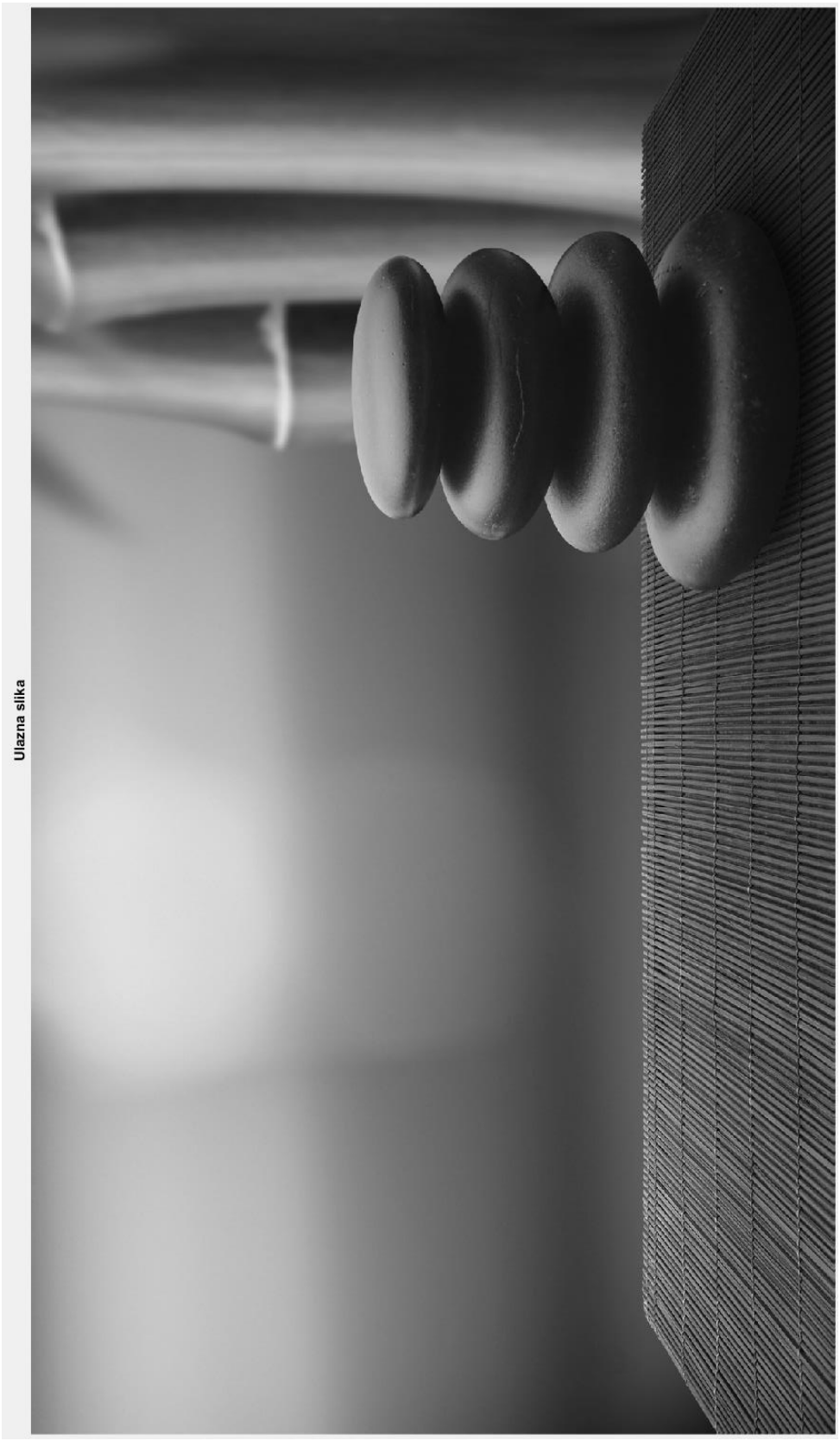
This paper describes two aproaches for image filtering using Gaussian 2D filter. First aproach is based exclusively on BRAM. Second aproach upgrades first by adding RAM. Here are described all used components and protocols, and it is explained in detail how each component that is constructed works.

Keywords: Image filtering, Gaussian blur, Zynq, BRAM, RAM, AXI.

ŽIVOTOPIS

Rođen 21. siječnja 1994. g. u Novoj Gradiški od oca Željka i majke Branke. Živi sa svojom obitelji u Novoj Gradiški, gdje pohađa osnovnu školu „Mato Lovrak“. Po završetku osnovne škole upisuje srednju Elektrotehničku školu u Novoj Gradiški, smjer tehničar za računalstvo. U srpnju 2012. g. upisuje se na Elektrotehnički fakultet u Osijeku, preddiplomski sveučilišni studij računarstva. U rujnu 2015. g. upisuje diplomski studij računarstva - smjer računalno inženjerstvo. Dobitnik je dva Pro-Student priznanja za dvije izrađene makete. Prvo Pro-Student priznanje dobio je 2015. g. za maketu s nazivom "Računalna igra pong na 16 matričnih pokaznika", a drugo priznanje 2016. g. za maketu s nazivom „Mikroupravljački sustav za prepoznavanje oblika“. U travnju 2017. g. započinje suradnju s tvrtkom „Institut RT-RK Osijek“, gdje se usavršava u području FPGA tehnologije.

PRILOG



Prilog 1. Ulazna slika[6]

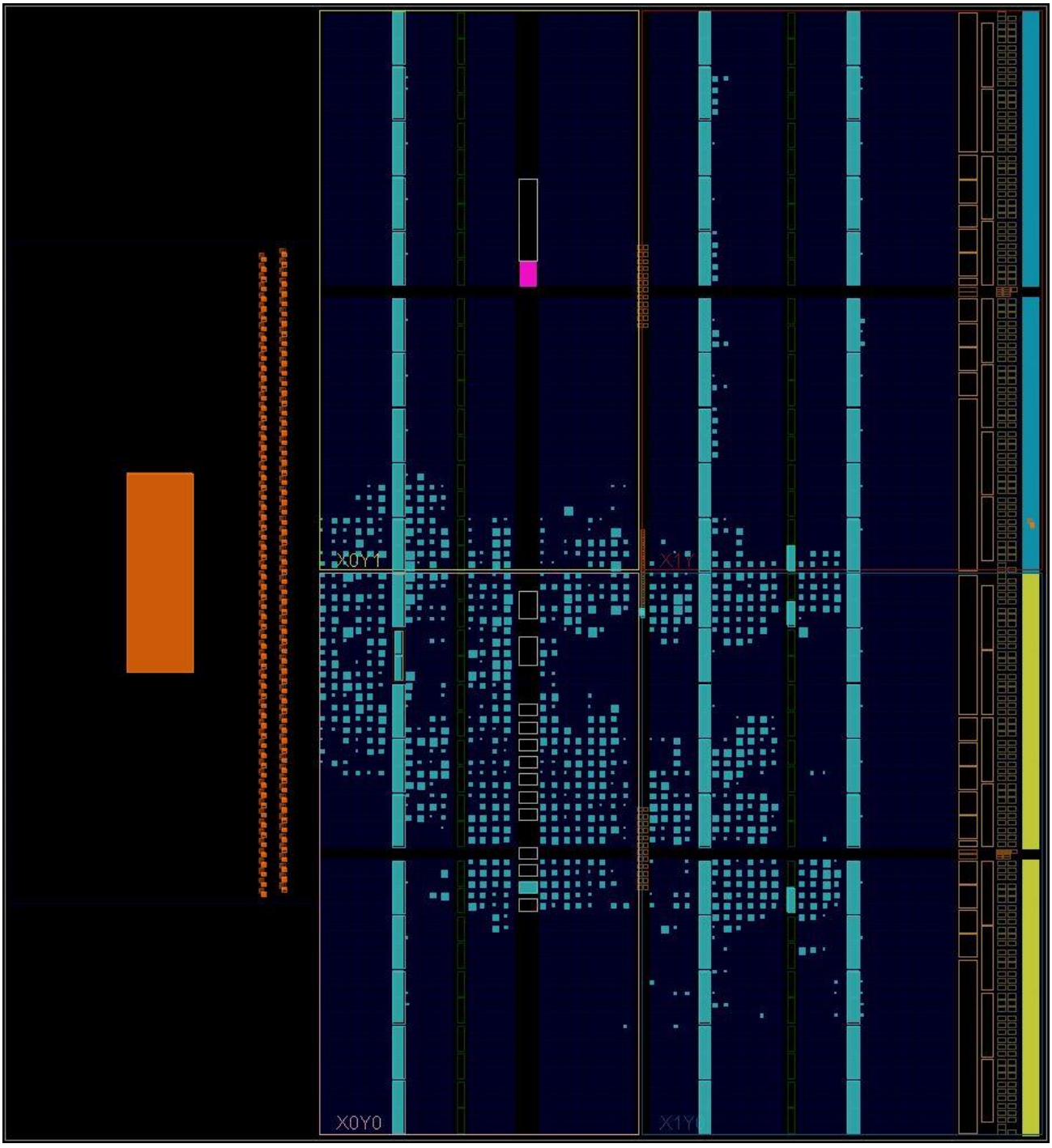


Prilog 2. Izlazna slika iz MATLAB programskog paketa, σ = 5



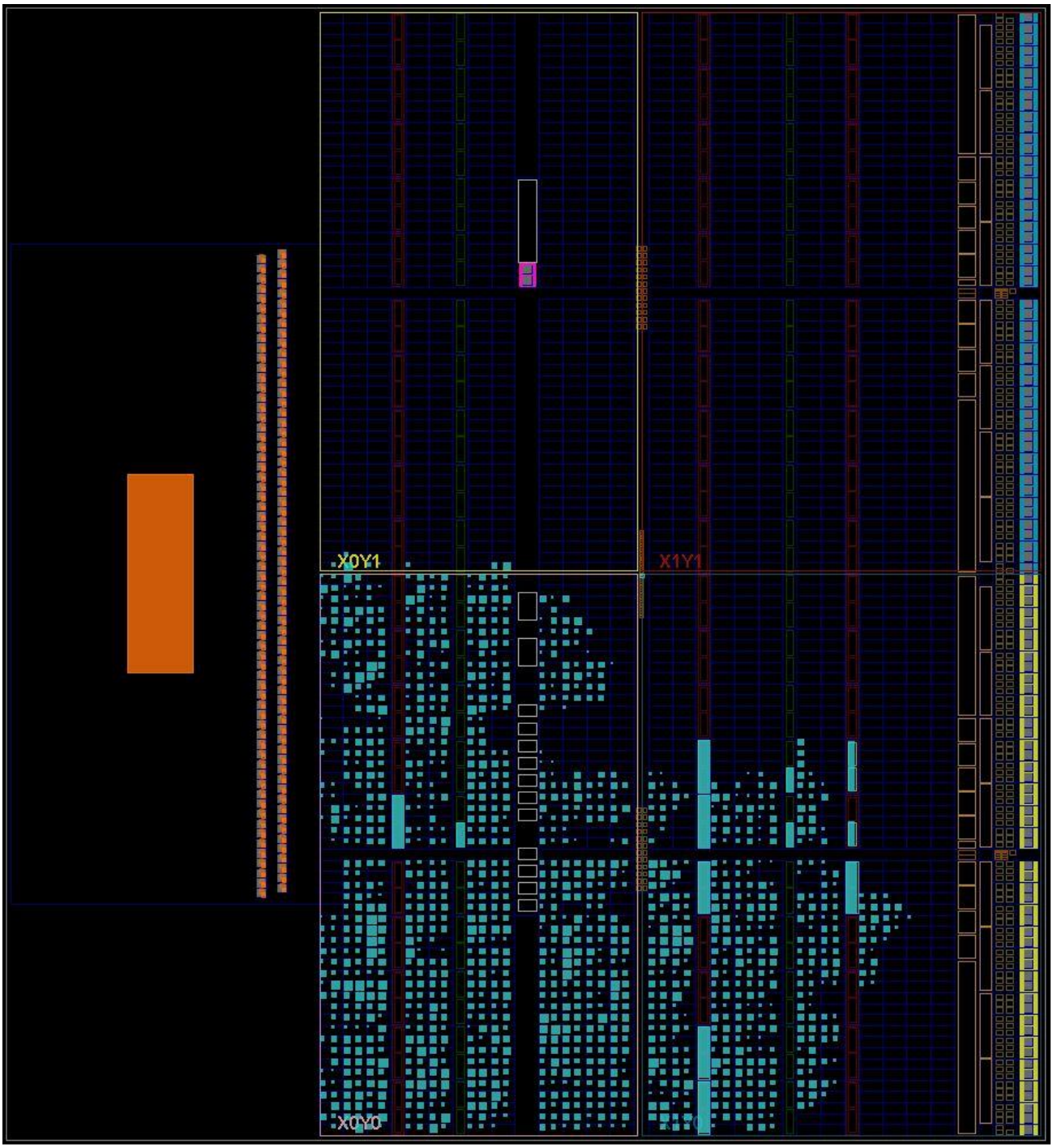
Prilog 3. Izlazna slika filtrirana na FPGA, σ = 5

P.4.1.



Prilog 4. Raspoređenost resursa na FPGA čipu za sustav izveden s BRAM-om

P.4.2.



Prilog 5. Raspoređenost resursa na FPGA čipu za sustav izveden s RAM-om