



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ
НАУКА У НОВОМ САДУ



Александар Комазец

**Дизајн и верификација платформе за
обраду слика**

ДИПЛОМСКИ РАД

- Основне академске студије -

Ментор: Вук Вранковић

Нови Сад, 2019



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
КАТЕДРА ЗА МИКРОРАЧУНАРСКУ
ЕЛЕКТРОНИКУ



Дизајн и верификација платформе за обраду слика

ДИПЛОМСКИ РАД

Основне академске студије

Кандидат:

Александар Комазец, ЕЕ 106-2014

Ментор:

др Вук Вранковић, доцент

Фебруар, 2019



УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
21000 НО ВИ СА Д, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА


Редни број, РБР:	
Идентификациони број, ИБР:	
Тип документације, ТД:	Монографска документација
Тип записа, ТЗ:	Текстуални штампани материјал
Врста рада, ВР:	Завршни рад
Аутор, АУ:	Александар Комазец
Ментор, МН:	др Вук Вранковић, доцент
Наслов рада, НР:	Дизајн и верификација платформе за обраду слика
Језик публикације, ЈП:	Српски
Језик извода, ЈИ:	Српски
Земља публикавања, ЗП:	Република Србија
Уже географско подручје, УГП:	Војводина
Година, ГО:	2019
Издавач, ИЗ:	Ауторски репринт
Место и адреса, МА:	Нови Сад, Трг Доситеја Обрадовића 6
Физички опис рада, ФО: (поглавља/страна/ цитата/табела/слика/графика/прилога)	(5/64/0/0/32/0/0)
Научна област, НО:	Електроника
Научна дисциплина, НД:	Системско пројектовање, дизајн и верификација дигиталног система
Предметна одредница/Кључне речи, ПО:	Системско пројектовање, дизајн и верификација дигиталног система, обрада слике, слика, универзални серијски протокол
УДК	
Чува се, ЧУ:	Библиотека Факултета Техничких Наука, 21000 Нови Сад, Трг Доситеја Обрадовића 6
Важна напомена, ВН:	Нема
Извод, ИЗ:	У раду је представљена апликација за прикупљање сликовних приказа, имплементација виртуелне платформе за обраду слике, пројектовање хардверског језгра за обраду слике као и верификација IP језгра.
Датум прихватања теме, ДП:	
Датум одбране, ДО:	
Чланови комисије, КО:	Председник: др Растислав Струхарик, ванредни професор
	Члан: маг. Андреа Ердељан, асистент
	Члан, ментор: др Вук Вранковић, доцент
	Потпис ментора



УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
21000 НО ВИ СА Д, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Accession number, ANO :	
Identification number, INO :	
Document type, DT :	Monographic publication
Type of record, TR :	Textual material, printed
Contents code, CC :	Final Thesis
Author, AU :	Александар Комазец
Mentor, MN :	dr Vuk Vranković, docent
Title, TI :	Design and verification of platform for image processing
Language of text, LT :	Serbian
Language of abstract, LA :	Serbian
Country of publication, CP :	Republic of Serbia
Locality of publication, LP :	Vojvodina
Publication year, PY :	2019
Publisher, PB :	Author's copy
Publication place, PP :	Novi Sad, Trg Dositeja Obradovića 6
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	(5/64/0/0/32/0/0)
Scientific field, SF :	Electronics
Scientific discipline, SD :	System design, digital design and functional verification
Subject/Key words, S/KW :	System design, design and verification of the digital system, photo processing, photography, universal serial protocol
UC	
Holding data, HD :	Library of the Faculty of Technical Sciences, 21000 Novi Sad, Trg Dositeja Obradovića 6
Note, N :	None.
Abstract, AB :	The paper presents an application for collecting images, implementing virtual data processing platform, designing a hardware system for image processing and verification the IP core.
Accepted by the Scientific Board on, ASB :	
Defended on, DE :	
Defended Board, DB :	President: dr Rastislav Struharik, associate professor
	Member: M.Sc. Andrea Erdeljan, assistant
	Member, Mentor: dr Vuk Vranković, docent
	Mentor's sign

	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	ЗАДАТАК ЗА ЗАВРШНИ (BACHELOR) РАД	Датум:

(Податке уноси предметни наставник - ментор)

Врста студија:	а) Основне академске студије б) Основне струковне студије
Студијски програм:	ЕМБЕДЕД СИСТЕМИ И АЛГОРИТМИ
Руководилац студијског програма:	др Милан Сечујски, ванредни професор

Студент:	Александар Комазец	Број индекса:	ЕЕ106/2014
Област:	Системско пројектовање, дизајн и верификација дигиталног система		
Ментор:	др Вук Вранковић, доцент		
НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА: <ul style="list-style-type: none"> - проблем – тема рада; - начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; - литература 			

НАСЛОВ ЗАВРШНОГ (Бечелор) РАДА:

Дизајн и верификација платформе за обраду слика

ТЕКСТ ЗАДАТКА:

- 1) Kreirati aplikaciju koja preuzima sliku sa internet kamere.
- 2) Projektovati virtualnu platform koristeći ESL metodologiju
- 3) Implementirati NNI (eng *Nearest Neighbor Interpolation*) algoritam u hardver prateći korake RT (eng *Register Transfer*) metodologije.
- 4) Verifikovati projektovano IP jezgro koristeći UV (eng *Universal Verification*) metodologiju.

Руководилац студијског програма:	Ментор рада:
др Милан Сечујски, ванредни професор	др Вук Вранковић, доцент

Примерак за: ☐ - Студента; ☐ - Студентску службу факултета

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, slika, 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

Sadržaj

Glava 1	1
Uvod	1
Glava 2	2
2.1 Uvod u embedded Linux i FPGA.....	2
2.1.2 Struktura FPGA	5
2.2 PetaLinux Tools	6
2.2.1 Konfigurisanje PetaLinux Tools.....	6
2.3 Zybo Z7-10	7
2.4 Internet kamera.....	8
2.5 Komunikacija kamere sa razvojnom pločom.....	8
Glava 3	12
3.1 Projektovanje dizajna na sistemskom nivou	12
3.2 Projektovanje Zoom2 dizajna na sistemskom nivou	15
3.2.1 Model procesora u virtuelnoj platformi	16
3.2.2 Model interkonekta u virtuelnoj platformi.....	17
3.2.3 Model dizajna za obradu slike u virtuelnoj platformi.....	19
Glava 4	22
4.1Projektovanje hardverskog IP bloka.....	22
4.1.1 Algoritam najbližeg komšije (eng <i>Nearest Neighbor Algorithm</i>)	22
4.2 Projektovanje IP modula korišćenjem RT metodologije	24
4.3 Oklopljavanje IP modula	34
Glava 5	38
5.1Verifikacija Zoom2 IP bloka	38
5.1.1 UVM metodologija i SystemVerilog	39
5.2 Projektovanje verifikacionog okruženja za Zoom2 IP blok	41
5.2.1 Projektovanje data objekta	41
5.2.2 Projektovanje stimulus slika.....	42
5.2.3 Projektovanje sekvenci	43
5.2.4 Projektovanje sekvencera.....	44
5.2.5 Projektovanje drajvera	44
5.2.6 Projektovanje monitora.....	48

5.2.7 Projektovanje agenta.....	49
5.2.8 Projektovanje skorborda	51
5.2.9 Projektovanje pokrivenosti.....	53
Zaključak	54
Dodatak	55
Literatura	56

Sadržaj slika

Slika 1 Osnovna FPGA arhitektura	4
Slika 2 Struktura FPGA kola	5
Slika 3 Zybo Z7-10	7
Slika 4 Komponente od važnosti.....	9
Slika 5 Profinjavanje modela.....	12
Slika 6 Taksonomija SystemC	13
Slika 7 TLM i RTL transakcije	15
Slika 8 ESL sistem za uveličavanje slike	16
Slika 9 Memorijsko mapiranje	18
Slika 10 Algoritam najbližeg komšije na primeru slike 2 puta 2	22
Slika 11 Struktura Controlpath-a i Datapath-a.....	26
Slika 12 Raddress registar	27
Slika 13 RaddressR registar	28
Slika 14 counter registar	28
Slika 15 Rnewaddress registar	29
Slika 16 x i y	29
Slika 17 c registar.....	30
Slika 18 r registar.....	30
Slika 19 NumberMatrix registar	31
Slika 20 mem_out registar	32
Slika 21 Zoom2_150x150 modul.....	33
Slika 22 Tipična struktura IP jezgra	35
Slika 23 Oklopljeno jezgro.....	35
Slika 24 Iskorišćenost hardvera	37
Slika 25 Konačno oklopljavanje jezgra	37
Slika 26 Verifikacioni ciklus	38

Slika 27 Unit test	41
Slika 28 Bidirekcionni model bez podrške paralelizma.....	45
Slika 29 Primer komunikacije drajvera	46
Slika 30 Komunikacija monitora.....	48
Slika 31 Agenti	49
Slika 32 Tipična struktura scoreboard-a	51

Listing koda

Listing Koda 1.....	6
Listing Koda 2.....	10
Listing Koda 3.....	16
Listing Koda 4.....	16
Listing Koda 5.....	17
Listing Koda 6.....	19
Listing Koda 7.....	19
Listing Koda 8.....	20
Listing Koda 9.....	20
Listing Koda 10.....	21
Listing Koda 11.....	21
Listing Koda 12.....	21
Listing Koda 13.....	24
Listing Koda 14.....	24
Listing Koda 15.....	42
Listing Koda 16.....	42
Listing Koda 17.....	43
Listing Koda 18.....	43
Listing Koda 19.....	43
Listing Koda 20.....	44
Listing Koda 21.....	46
Listing Koda 22.....	46
Listing Koda 23.....	47
Listing Koda 24.....	47
Listing Koda 25.....	47
Listing Koda 26.....	48
Listing Koda 27.....	50

Listing Kodā 28.....	50
Listing Kodā 29.....	50
Listing Kodā 30.....	51
Listing Kodā 31.....	52

Sadržaj tabela

Tabela1 Raspored bitova unutar memorije.....	11
Tabela2 Adrese upisanih piksela obrađene slike.....	23
Tabela3 Algoritam za generisanje adrese na kojoj će biti upisan piksel.....	23
Tabela4 Objašnjenje svih članova matematičkog iskaza	23
Tabela5 Legenda ulaza na multiplekseru	27
Tabela6 Objašnjenje portova modula za obradu slike.....	33

Glava 1

Uvod

Tema diplomskog rada obuhvata teorijski i praktični deo. U teorijskom delu su razmatrani algoritmi za obradu slike. Algoritmi su posmatrani sa aspekta kompleksnosti, resursa kao i efikasnosti. Slika spremna za obradu predstavlja rezultat aplikacije koja beleži slikovne okvire. Praktičan deo obuhvata projektovanje hardverskog bloka na RT (Register Transfer)[1] nivou kao i na sistemskom nivou. Kako u praksi prva verzija uglavnom ne funkcioniše prema očekivanjima, potrebno je sprovesti verifikaciju pomenutih projektovanih blokova. Za verifikaciju se koristi UVM(eng *Universal Verification methodology*)[2].

Poslednjih četrdeset godina, hardver je dostigao veliki napredak sa aspekta iskorišćenih tranzistora. Današnji digitalni sistemi su mnogo kompleksniji i efikasniji u poređenju sa onima iz prošlosti. Razvitak hardvera je takođe pratio i razvitak softvera. Tehnologija koja danas postoji, mnogo olakšava svakodnevni život. Efikasnost modernih sistema sa druge strane povlači i njihovu veću kompleksnost tako da projektovanje kao i provera današnjih sistema uopšte nije lak zadatak. Sa aspekta projektovanja sistema, razvija je posebna metodologija nazvana ESL (eng *Electronic System Level*)[3]. ESL metodologija omogućava razvoj softvera i hardvera u isto vreme i samim tim olakšava i ubrzava razvoj. Omogućen je relativno lak razvoj različitih modela. Neki od tih modela su:

- konceptualni - ovaj model pomaže da se istaknu važne veze između procesa,
- fizički - ovaj model ima slične karakteristike realnih sistema,
- matematički - predstavljaju jednačine koje se analiziraju,
- vizuelni - nam pomažu da zamislimo kako nešto radi u realnom svetu pomoću grafičke reprezentacije,
- logički – modeli koji nam pomažu da pretvorimo ideje u krajnja rešenja.

Pre uvpođenja ESL metodologije, često korišćen način projektovanja sistema bio je da se prvo projektuje hardver, zatim da se na projektovanom hardveru piše softverski deo. Nedostaci ovakvog pristupa su bile to što je vreme do završetka celokupnog projekta bilo dugo kao i to što su softverski inženjeri morali da čekaju završetak rada hardverskog tima kako bi počeli sa razvojem softvera.

Još jedna mana je mala fleksibilnost samog hardvera. ESL metodologija ima sasvim drukčiji pristup. Taj pristup je poznat kao profinjavanje modela. Profinjavanje modela vrši se od algoritamskog nivoa pa sve do nivoa kapija (eng *Gate Level*).

Poslednji korak predstavlja particionisanje. Particionisanje predstavlja situaciju u kojoj inženjer mora da se odluči na koji način će projektovati određeni sistem, to jeste da odluči koji deo će se projektovati kao softver, a koji deo kao hardver. Softverski deo predstavlja veću fleksibilnost u odnosu na hardverski deo zbog toga što je greške moguće otkloniti jednostavnim izbacivanjem nove verzije koda dok greške u hardveru mogu da naprave ozbiljne finansijske gubitke kompanijama. Najbolji način je sprega hardvera i softvera tako što bi se delovi koji su kritični po efikasnosti, resursima ili potrošnji implementirali u hardveru. Današnja zajednica inženjera razvila je standardizovan postupak za projektovanje hardvera nazvan RT. Ova metodologija se zasniva na određenim koracima koje je potrebno ispoštovati. Za određeni algoritam potrebno je napisati datapath model koji predstavlja podsistem za obradu podataka. Takođe potrebno je napisati i controlpath model koji predstavlja upravljački deo sistema. Controlpath model se obično predstavlja ASMD – om (eng *Algorithmic State Machine with Datapath*)[4] , dok je datapath model predstavljen dijagramom koji je obično podeljen na funkcionalne jedinice, mrežu za rutiranje i memorijske elemente.

Projektovati proizvod koji radi poštujući predviđenu funkcionalnost nakon prve verzije je praktično nemoguće tako da vrlo važan aspekt predstavlja verifikacija sistema. Od velike važnosti je pisanje efikasnog verifikacionog okruženja koje će moći ponovo da se iskoristi. Potrebno je proveriti funkcionalnost sistema pokrivajući sve moguće slučajeve. Takođe je potrebno proveriti strukturnu pokrivenost, to jest da li postoji kod koji se nikada ne izvršava. Propuštena greška u sistemu može da napravi velike finansijske troškove kompanijama kao i da uništi njihov ugled doživotno.

Glava 2 obuhvata teorijski uvod u FPGA (Field Programmable Gate Arrays)[5] , kao i teorijski uvod korišćenih alata i operativnog sistema. Predstavljena je korišćena razvojna ploča, njene osobine kao i ostale stvari potrebne za rad aplikacije koja se koristi za prikupljanje slikovnih okvira koji se šalju na dalju obradu. Aplikacija je pisana korišćenjem C programskog jezika.

Glava 3 se bavi projektovanjem celokupnog sistema uključujući i hardverski blok koje služi za obradu slike. Projektovanje je izvršeno korišćenjem SystemC jezika, koji uzgred može

da se posmatra kao biblioteka C++ jezika koja omogućava pisanje modela na sistemskom nivou. Pisanje modela na ovom nivou omogućava brz razvoj softvera, čak i pre završetka implementacije hardvera što predstavlja motivaciju za korišćenje.

Glava 4 obuhvata kratak opis odabranog algoritma, njegove prednosti i nedostatke. Pomenuta glava se takođe bavi i detaljima projektovanja hardverskog bloka korišćenjem RT (eng *Register Transfer*) metodologije. Hardverski blok je pisan korišćenjem VHDL-a (eng *VHSIC Hardware Description language*)[6].

Glava 5 bavi se projekovanjem okruženja za proveru hardverskog bloka, to jest verifikacionog okruženja koje predstavlja važan deo pre nego što proizvod dospe na tržište. Verifikaciono okruženje je pisano korišćenjem System Verilog jezika.

Glava 2

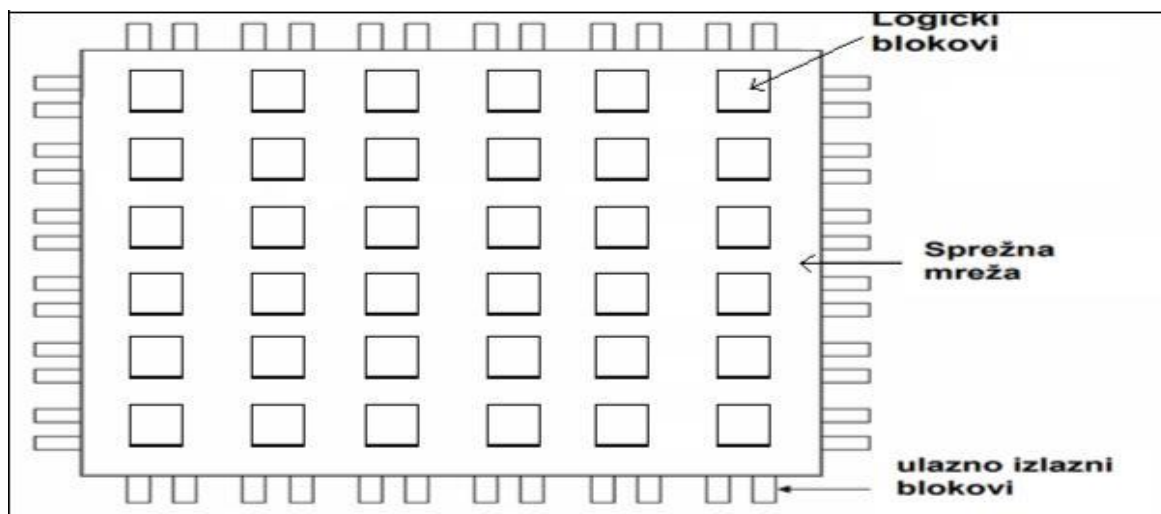
2.1 Uvod u embeded Linux i FPGA

Operativni sistemi bazirani na Linux jezgru (eng *kernel*) se koriste u ugrađenim sistemima kao što su potrošačka elektronika (tj. Set-top boksovi, video rekorderi, mrežna oprema (kao što su ruteri, bežične pristupne tačke ili bežični ruteri), softver za letelice kao i u medicini).

Operativni sistem predstavlja važan deo celokupnog sistema, takođe važan deo sistema predstavlja razvojna ploča. Danas je na tržištu moguće naći čitav spektar ploča sa različitim osobinama. U nastavku rada koristi se ploča Zybo Z7-10 kompanije Xilinx. Naime, ova ploča predstavlja FPGA (eng *Field Programmable Gate Array*). Ova ploča predstavlja jednu od mnogobrojnih koji se nalaze u familiji ploča koje je moguće ponovo programirati.

Arhitekturu ovih ploča čine sledeće komponente:

- ulazno izlazni blokovi
- logički blokovi
- sprežna mreža



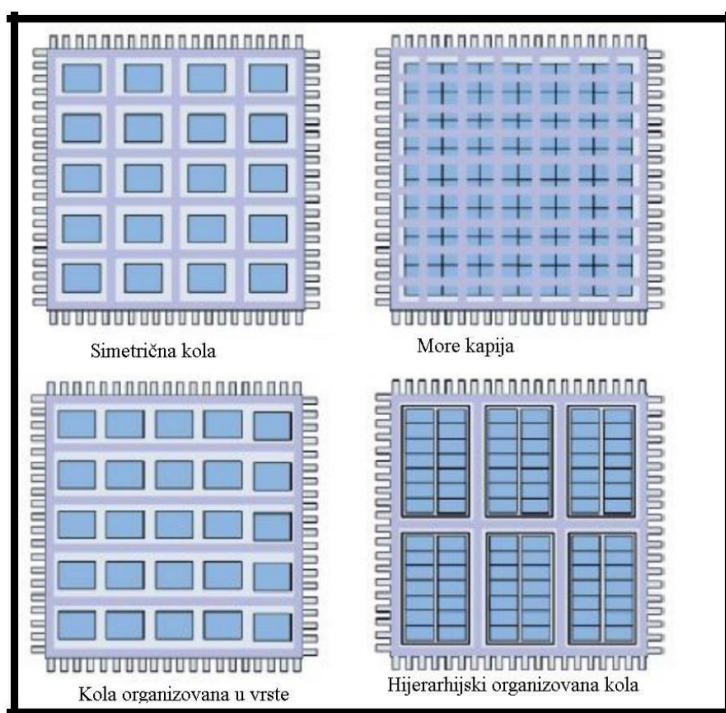
Slika 1. Osnovna FPGA arhitektura

Pored pomenutih delova koji su prikazani na slici 1, FPGA arhitektura se uglavnom sastoji još i od gradivnih blokova koji su odgovorni za distribuciju taktnog signala do svakog ločikog bloka unutar same ploče. Potrebno je operacije prevesti u određene blokove kao što su sabirači, množači, blokovi za memorisanje podataka, dekoderi i tako dalje. Vredan deo sistema su i konekcije ploče sa računarom kako bi se na jednostavan način vršila provera napisanog programa.

2.1.2 Struktura FPGA

FPGA kola se sastoje od logičkih ćelija koje su rasporede unutar kola i čine resurse za proračune. Struktura FPGA kola zavisi od proizvođača. FPGA kola se dele na četiri kategorije, a to su:

- simetrična polja (eng *symmetrical array*)
- kola organizovana u vrste (eng *row-based*)
- more gejtova (eng *sea of gates*)
- hijerarhijski organizovana kola (eng *hierarchy-based*)



Slika 2. Struktura FPGA kola (Preuzeto iz [5])

Na slici 2 se mogu videti pomenute strukture FPGA kola koje će biti opisane u nastavku.

- Simetrična polja FPGA kola sastoje se od logičkih blokova sačinjenih u dve dimenzije koji su okvireni u skup horizontalnih i vertikalnih linija.
- FPGA kola koja su organizovana u vrste sastoje se od raspoređenih vrsta logičkih blokova ili makro ćelija kao i kanala. Kanal predstavlja prostor između logičkih blokova i koristi se za rutiranje.
- FPGA kola koja su tipa more gejtova slična su FPGA-ovima pomenutim FPGA-ovima sa simetričnim poljima. Makro ćelije su organizovane u strukturu tipa dvodimenzionalno polje pri čemu ulaz u polje odgovara koordinati date makro ćelije.
- FPGA kola koja su hijerarhijski organizovana, makro ćelije su hijerarhijski raspoređene na kolu. Elementi najmanje granularnosti se nalaze na najnižem nivou[5].

2.2 PetaLinux Tools

PetaLinux Tools predstavlja sve što je potrebno za jednostavno vršenje izmena, rešavanja problema koristeći Xilinx procesorske sisteme. Namenjen je za ubrzavanje efikasnosti i prilagođen je alatima kompanije Xilinx. PetaLinux alati će kreirati paket podrške BSP (eng *Board Support Package*) za Linux, za odgovarajuću ploču. U ovom slučaju je korišćena već pomenuta ploča Zybo Z7-10 kompanije Xilinx. Tako da je potrebno koristiti BSP prilagođen toj ploči. BSP sadrži sve potrebne drajvere za ploču, gotove blokove, jezgro kao i boot loader konfiguracije.

2.2.1 Konfigurisanje PetaLinux Tools

Potrebno je instalirati PetaLinux i BSP paket za odgovarajuće verzije alata. Važna činjenica je da oba moraju biti iste verzije. U ovom projektu je korišćena verzija 2017.

```
petalinux-create -t project -s ${XIL_INSTALL}/petalinux/petalinux-
v2017.2/bsp/petalinux-zybo.bsp
cd Zybo
petalinux-config -c rootfs
petalinux-build
petalinux-package --boot --format BIN --force --fsbl ./images/linux/zynq_fsbl.elf --fpga
./images/linux/system_wrapper.bit --uboot
```

Listing Koda 1

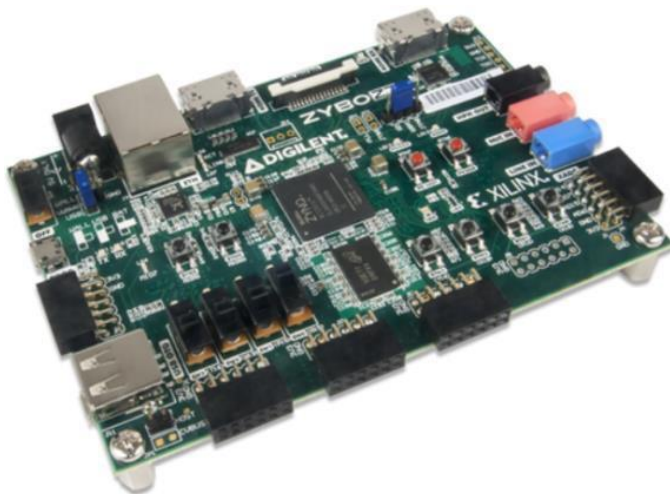
Listing koda 1, redom od gore, predstavlja:

- kreiranje projekta koristeći PetaLinux alate,
- prelazak u određeni folder,
- nameštanje konfiguracije prostora predviđenog za koren (eng *rootfs*),
- bildovanje projekta,
- kreiranje važnih fajlova koji će biti pomenuti u nastavku.

Na listingu 1 je moguće videti svega par linija sa kojima se dobija gotov proizvod, to jest fajlovi koji se kasnije prebace na određenu memoriju, recimo memorijsku karticu. Na taj način je automatizovana instalacija sistema korišćenjem gore pomenutih alata. Takođe moguće je sve raditi ručno[7].

Sledeće je potrebno kopirati BOOT.bin i ./images/linux/image.ub fajlove na memorijsku kartu, a zatim je potrebno namestiti razvojnu ploču, Zybo Z7-10 da se pokreće sa memorijske karte. U ovom slučaju potrebno je samo namestiti kratkospojnik na mestu gde je obeleženo “SD”, staviti SD kartu i uključiti ploču. Posmatranje serijske komunikacije je moguće korišćenjem nekog od za to predviđenih programa, na primer minicom.

2.3 Zybo Z7-10



Slika 3. Zybo Z7-10 (Preuzeto iz [8])

Zybo Z7-10 se nalazi na slici 3 i predstavlja član porodice Xilinx Zynq-7000. Ova ploča predstavlja nadogradnju popularne ploče objavljene 2012. godine. Prednost ove Zynq porodice je u njihovoj jedinstvenosti koje predstavlja sveobuhvatno programiranje na čipu (eng *All Programmable System-on-Chip*). Ovaj tip arhitekture sadrži dvojezgarni ARM Cortex-A9 procesor. Zybo Z7-10 okružuje Zynq sa bogatim skupom multimedijalnih periferija. U nastavku će biti prikazane neke od osobina pomenute ploče.

ZYNQ processor:

- 667 MHz dvojezgarni Cortex-A9 procesor
- DDR3L memorijski kontroler sa 8 DMA kanala i 4 AXI3 porta visokih sposobnosti
- Visoko propusne kontrolere kao što su: 1G Ethernet, USB 2.0, SDIO
- Nisko propusne kontrolere kao što su: SPI, UART, CAN, I2C
- Programabilnost, koristeći JTAG, Quad-SPI fleš, i mikro SD kartica

Memorija:

- 1 GB DDR3L sa 32-bitinom magistralom pri brzini od čak 1066 MHz
- microSD slot

USB and Ethernet:

- Gigabit Ethernet PHY
- USB-JTAG komunikacija
- USB-UART most
- USB 2.0 OTG PHY, predstavlja brzu komunikaciju

Tu su još i mnogobrojni prekidači, dugmići, ulazi za povezivanje sa video i audio uređajima [8].

2.4 Internet kamera

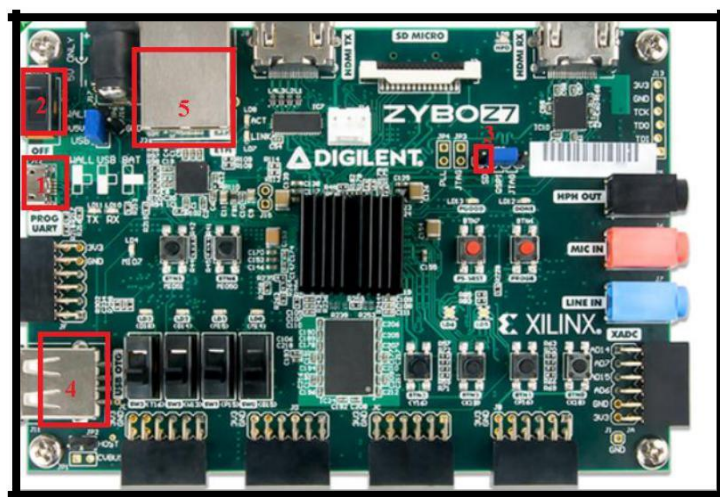
Internet kamera (eng *Webcam*) je video kamera koja sakuplja ili prenosi sliku u realnom vremenu. Slika predstavlja informaciju sastavljenu od piksela. Svaki piksel predstavlja broj u određenom opsegu. Korišćen opseg je od 0 do 255. Pikseli su poređani u matricu. Slika sa svim potrebnim informacijama predstavlja slikovni okvir (eng *Picture Frame*). Osim slika, moguće je napraviti video snimke koji predstavljaju skup slikovnih okvira. Glatkoća video snimka zavisi od količine okvira u jednoj sekundi, isto kao i od kvaliteta ekrana sa koga se posmatra određeni snimak. Komunikacioni protokol je univerzalni serijski protokol.

2.5 Komunikacija kamere sa razvojnom pločom

Nakon koraka ispraćenih u predhodnom izlaganju, ploča na sebi ima sistem koji je spreman za upotrebu. Prateći sledeće korake prikazane na slici 4, pokreće se sistem na razvojnoj ploči.

Koraci:

1. Priključiti napajanje, korišćeno je napajanje računara
2. Pokrenuti ploču pomeranjem prekidača na "ON"
3. Uveriti se da se plavi kratkospojnik nalazi na "SD"
4. Priključiti internet kameru
5. Priključiti ploču na lokalnu mrežu



Slika 4. Komponente od važnosti (Preuzeto iz [8])

Operativni sistem sa koga su kucane komande je Ubuntu operativni sistem. Ploču je potrebno spojiti na lokalnu mrežu na kojoj se nalazi računar sa koga se šalju komande. Ovo je moguće koristeći komandu: `sshpass -p "root" scp -r grabber root@ 192.168.0.xx:/home`. xx predstavlja promenljivi broj unutar adrese lokalnog rutera.

Svakako je dobra praksa proveriti tačnu adresu. Unutar gore pomenute linije, **grabber** predstavlja program koji se izvršava na ploči, koji je pozivom pomenute komande prebačen na određeno mesto na ploči. Da bi se program izvršavao na ploči potrebno je prebaciti određene biblioteke na ploču (libv4l2.so.0, libv4lconvert.so.0, libjpeg.so.8) koristeći komandu: `sshpass -p "root" scp libv4l2.so.0 libv4lconvert.so.0 libjpeg.so.8 root@192.168.0.21:/usr/lib`. Komande je moguće ručno pisati ili napraviti jednostavnu skriptu u kojoj će se nalaziti sve potrebne komande, tako da prostim pozivanjem skripte će se automatski izvršiti sve komande, jedna za drugom. U trenutku kada se prebace neophodni podaci, skripta je pauzirana i čeka se na ponovo pokretanje skripte. Sledeći korak je da se na ploči pokrene program jednostavnom komandom `./grabber`. Program će se izvršiti po algoritmu koji će biti objašnjen kasnije.

Kao rezultat, na ploči je prikupljeno nekolino slika koje su nastale fotografisanjem internet kamere. Sada je potrebno ponovo pokrenuti skriptu, pritiskom na bilo koje dugme. Skripta u nastavku prebacuje jednu od slika sa ploče na računar.

Grabber program se zasniva na Video4Linux-u[9]. Ukratko, Video4Linux je skup drajvera za uređaje i aplikacioni programabilni interfejs za snimanje videa u realnom vremenu na Linux sistemima. Podržava web kamere, TV tjunere i povezane uređaje, standardizirajući njihov izlaz, tako da programeri mogu lako dodati video podršku svojim aplikacijama. U nastavku su data obašnjenja neka od korišćenih v4l2 funkcija:

- **v4l2_open** predstavlja funkciju pomoću koje se otvara uređaj koji se nalazi na određenoj adresi, u ovom slučaju je to na adresi /dev unutar Ubuntu operativnog sistema, pod nazivom video1. **char *dev_name = "/dev/video1";**
- **v4l2_ioctl** se koristi za programiranje V4L2 uređaja. Prvi argument mora biti deskriptor koji je zadužen za otvaranje datoteke. Ioctl zahtev je kodovao u njega da li je argument ulaz, izlaz, read ili write parametar, a veličina argumenta broj dva je u bajtovima.
- Postoji jedan mehanizam za pregovaranje svih formata podataka koristeći agregatnu strukturu **v4l2_format**.
- **v4l2_buffer** predstavlja specijalan vid prostora za skladištenje podataka.
- **mmap** uklanja prethodnu mapu s mapiranim baferom **mmap()** i oslobađa je, ako je moguće.

Dobro kodiranje se takođe zasniva i na postavljanju određenih ograničenja tokom kucanja samog koda kako bi se korisnik pravovremeno informisao o pravilnom korišćenju aplikacije. Tako je i kod napisanog grabber fajla zaštićen određenim ograničenjima kao što je moguće videti na listingu koda 2. Originalni kod je izmenjen[10]

```
if (fmt.fmt.pix.pixelformat != V4L2_PIX_FMT_RGB24){  
    printf("Libv4l didn't accept RGB24 format. Can't proceed.\n");  
    exit(EXIT_FAILURE);}  
if ((fmt.fmt.pix.width != 640) || (fmt.fmt.pix.height != 480))  
    printf("Warning: driver is sending image at %dx%d\n",  
        fmt.fmt.pix.width, fmt.fmt.pix.height);
```

Listing Koda 2

U prvoj proveru se posmatra da li je ispunjen uslov što se tiče očekivanog formata. U slučaju da format nije V4L2_PIX_FMT_RGB24, javlja se greška i program se završava, dok se u drugoj proveru šalje upozorenje ako širina i dužina nisu u odgovarajućem opsegu.

Tabela1 Raspored bitova unutar memorije

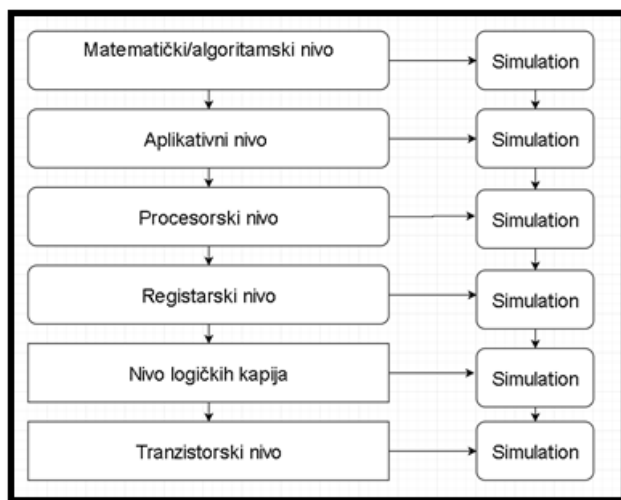
	Code	Byte 0 in memory								Byte 1								Byte 2							
V4L2 PIX FMT RGB24	„RGB3“	R	R	R	R	R	R	R	R	G	G	G	G	G	G	G	G	B	B	B	B	B	B	B	B
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Svaki od formata ima određeni raspored bitova unutar memorije. Raspored pomenutog formata se može videti u tabeli 1[11].

Glava 3

3.1 Projektovanje dizajna na sistemskom nivou

Projektovanje dizajna na ovom nivou korišćenjem određene metodologije može mnogo da olakša i ubrza čitav proces kreiranja dizajna kao i da samom inženjeru omogući da istraži prostor mogućih rešenja pre same implementacije. Postoje mnogobrojne metodologije [3] koje se u proteklom vremenskom periodu nisu pokazale kao najbolje iz više razloga, kao na primer, određeni procenat projekata je otkazan, projekti su bivali završeni sa zakašnjenjem od čak pola godine i mnogi završeni projekti nisu ispunili očekivanja. Zbog svih ovih problema, pominje se jedna prilično efikasna metodologija, a to je ESL metodologija. Ova metodologija se zasniva na sistemu profinjavanja, to jest kreće se od najapstraktnijeg modela kao što je recimo algoritamski, matematički nivo i laganim profinjavanjem se dolazi do najnižeg tranzistorskog nivoa. Jedan takav primer, prikazan je na slici 5, gde se može videti 6 nivoa, od kojih je matematički, algoritamski nivo na vrhu hijerarhije dok je tranzistorski nivo najniži.



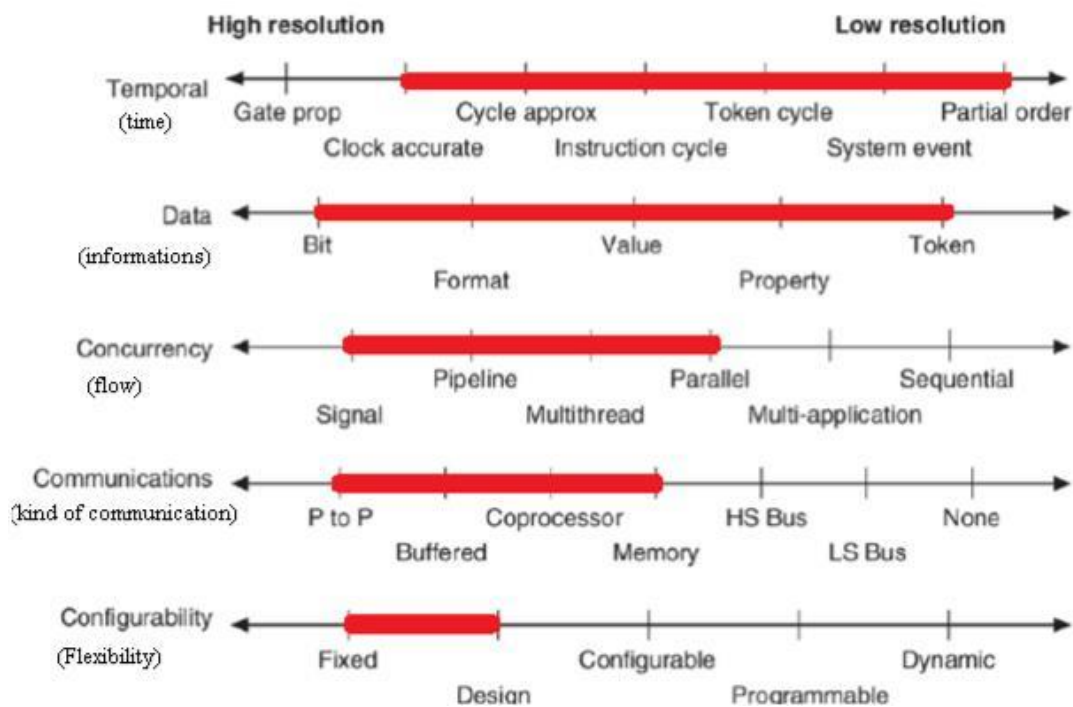
Slika 5. Profinjavanje modela

Vrši se simulacija svakog nivoa i na taj način korišćenjem pomenute metodologije moguć je rani razvoj softvera kao i procene performansi sistema u ranoj fazi razvoja. Moguće je i particionisanje sistema na određene blokove kako bi se čitava logika rada podelila na više jednostavnijih blokova i na taj način omogućila optimizacija čitavog sistema.

SistemC predstavlja most između dva domena jer sadrži fiksni model izračunavanja i prati semantiku diskretnih događaja, dok je sa druge strane zasnovan na programskom jeziku C++. SistemC pokriva sve od RTL konstrukcija pa čak sve do modela transakcija. Uvodi se pojam taksonomije koja predstavlja karakterizaciju objekata ili koncepata i bazirana je na odnosu između njih. Cilj je klasifikacija. Taksonomija može biti predstavljena u vidu hijerarhijskog grafa ili tabela atributa, gde svaki atribut određuje tačku razlike. Postoji podela ESL metodologije na ose. Ova podela je prikazana u nastavku. Ose ESL su:

- Vreme,
- Podaci,
- Konkurentnost,
- Komunikacija,
- Konfiguracija.

U nastavku, na slici 6, je data taksonomija SystemC jezika kako se uvidele prednosti pomenutog jezika.



Slika 6. Taksonomija SystemC

Prva osa, Temporal predstavlja vremensku osu na kojoj je definisano sedam tačaka. Gledajući osu, može se zaključiti da jedina tačka koja nije pokrivena ovom taksonomijom je tačka u kojoj postoji kašnjenje kroz kapije(eng *Gate propagation accurate*), koja predstavlja tajming klock periode poznate po solidnoj preciznosti. S obzirom da su sve ostale tačke pokrivena, može se zaključiti da je spisak mogućnosti prilično širok, od najapstraktnije tačke (eng *Partially ordered events*) koja predstavlja isključivo poznavanje redosleda izvršavanja događaja ali ne i njihovog početka i kraja, pa sve do sistema gde postoji precizan sinhronizacioni signal, gde je poznat tačan broj taktova (eng *Cycle-accurate*).

Druga osa, Data predstavlja osu podataka na kojoj je definisano pet tačaka. Gledajući osu, može se zaključiti da pomenuta taksonomija pokriva sve tačke, od najpovršnije tačke (eng *Token*) gde je moguće videti samo da određeni podaci prolaze kroz sistem, pa sve do informacija o podacima na nivou bita koji se smeštaju u određenu memoriju.

Treća osa, Concurrency predstavlja osu konkurentnosti na kojoj je definisano šest tačaka. Konkurentnost definiše količinu obrade ili izvršavanja neke obrade koja se može izvršavati simultano. Implementacija ne mora da iskoristi svu konkurentnost, jer to može dovesti do pogoršavanja ostalih karakteristika kao što su cena, dimenzije i potrošnja. Takođe veća konkurentnost usložnjava verifikaciju. Gledajući osu, može se zaključiti da pomenuta taksonomija pokriva četiri tačke, od tačke koja predstavlja paralelizam (eng *Parallel*) gde više funkcija može da se izvršava nezavisno, ali je potrebna saradnja kod nekih aktivnosti. Primer ovoga je sistem za obradu podataka, gde su podaci eksplicitno deljeni između svake funkcije i zaključavanje pristupa direktno utiče na rad ostalih funkcija, pa sve do tačke koja predstavlja nivo signala. Najfiniji nivo konkurentnosti tipičan za gate-level i asinhronne opise. Postoji mnogo kombinacionih puteva između tačaka sinhronizacije i podaci se prenose brzinom kojim im samo kolo ograničava. Tačke sinhronizacije upravo odvajaju delove pipeline-a.

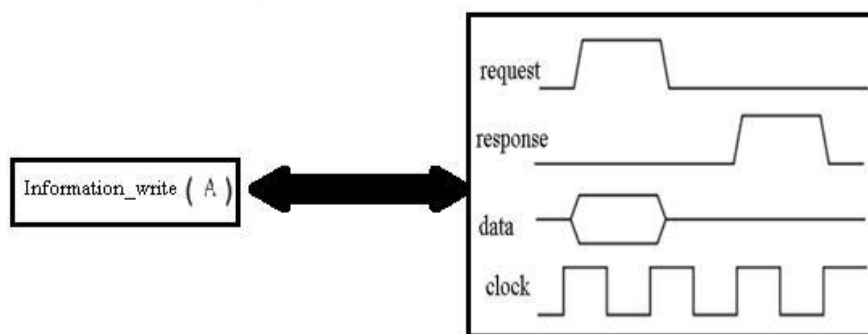
Četvrta osa, Communications predstavlja osu komunikacije na kojoj je definisano sedam tačaka. Gledajući osu, može se zaključiti da pomenuta taksonomija pokriva četiri tačke, od tačke memorije (eng *Memory*). Ova tačka je veoma čest način razmene informacija između funkcija. Podrazumeva se da svaka funkcija koja pristupa memoriji sadrži portove prema memoriji i može da upisuje i čita podatke nezavisno.

Porikvenost ide sve do tačke (eng *point to point*) koja omogućava da dve različite funkcije komuniciraju direktno bez dodatne kontrole. Ovo se nalazi samo u hardverskim rešenjima.

Peta osa, Configurability predstavlja osu konfiguracije na kojoj je definisano pet tačaka. Gledajući osu, može se zaključiti da pomenuta taksonomija pokriva dve tačke, a to su fiksna (eng *Fixed*) u kojoj nije moguće praviti promene. Može postojati na bilo kom nivou apstrakcije u hardveru ili softveru. Može da se ponaša kao crna kutija, ali to ne mora da bude slučaj. Primer za fiksni dizajn u softveru je pokretački fajl (eng *Executable file*). Druga tačka je dizajn (eng *Design*). Konfigurabilnost na nivou dizajna je moguća ako raspoložemo sa izvornim kodom. Skoro svi dizajnovi su konfigurabilni u ovoj fazi.

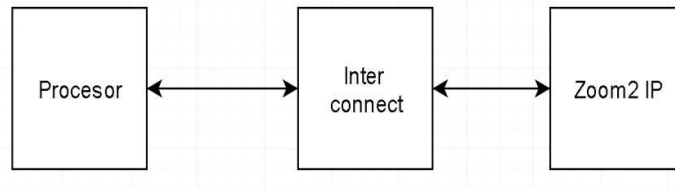
3.2 Projektovanje Zoom2 dizajna na sistemskom nivou

Kao što je već pomemuto, pomoću ESL metodologije moguć je rani razvoj samog dizajna, tako da se uvodi novi pojam, virtuelna platforma. Virtuelna platforma predstavlja softverski model koji predstavlja ranu zamenu hardverskog modela na početku projektovanja, to jest u ranoj fazi. Ovi softverski modeli se povezuju u jednu celinu kako bi u ranoj fazi verodostojno predstavljali celokupan sistem koji se razvija. Povezivanje softverskih modela se izvršavava korišćenjem metodologije na bazi transakcija (eng *Transaction Level modeling*). Modeli na nivou transakcija komuniciraju na taj način što se vrše pozivi funkcija i predstavljaju dosta jednostavniji model koji se koristi u poređenju sa RTL simulacijama gde se vrednosti postavljaju ručno na svaki signal. Jedan primer je dat u nastavku na slici 7.



Slika 7. TLM i RTL transakcije

Na levoj strani se nalazi TLM transakcija i kao što je već rečeno, predstavlja jedan poziv funkcije dok sa desne strane, isto značenje u RTL transakciji imaju četiri signala od kojih je klock sinhronizacioni signal, signal request je signal upita, data je signal za podatke i response je signal odgovora. Dolazi se do zaključka da su modeli na TLM nivou brži i jednostavniji za projektovanje. Na slici 8 je prikazan ESL sistem koji predstavlja dizajn na sistemskom nivou koji opisuje algoritam za obradu slike, to jest u ovom slučaju uveličavanje slike dva puta.



Slika 8. ESL sistem za uveličavanje slike

Pomenuti ESL sistem se sastoji od procesora, interconnect bloka i Zoom2 bloka. Ideja je da se sistem realizuje korišćenjem transakcija koje predstavljaju standard TLM metodologije.

3.2.1 Model procesora u virtuelnoj platformi

Procesor u ovom slučaju predstavlja test benč koji se sastoji od jednostavnog inicijatorskog priključa koji nosi naziv **isoc**, jednog procesa pod nazivom **test**, kao i metoda koja je zaslužna za ispisivanje transakcija, a njen naziv je **msg**. Fajl `tb_vp.cpp` je napisan test koji proverava funkcionalnost zamišljenog sistema. Konstruktor modula `tb_vp` registruje process test kao što se može videti u listingu koda 3.

```

sc_module(name){
  SC_THREAD(test);
  SC_REPORT_INFO("tb_vp", "Test bench is constructed");}
  
```

Listing Koda 3

Test proces sadrži komande koje su potrebne da se izvrši provera željene funkcionalnosti sistema.

```

//Send informations to Zoom2IP in order to read a picture
pl.set_address(VP_ADDR_MYIP);
pl.set_command(TLM_WRITE_COMMAND);
pl.set_data_length(1);
pl.set_data_ptr((unsigned char*)&val);
  
```

Listing Koda 4

Kako bi potrebne informacije bile poslate u određeni blok, koriste se već predefinisane funkcije koje zahtevaju određene argumente.

Primer se nalazi u listing koda 4 gde se šalju potrebne informacije ka Zoom2 modelu koji je opisan unutar fajla Zoom2IP.cpp i Zoom2IP.hpp, o čemu će biti reči u nastavku glave. U listing koda 4 koriste se četiri prilično jednostavne komande koje zahtevaju podatke o adresi, o tome da li je potrebo čitati ili pisati, širina podatka kao i sam podatak. Kako bi se razjasnila sama funkcionalnost, dobra praksa je koristiti poruke obaveštenja koje govore o tome koji proces je izvršen, koji proces je u toku i koji je sledeći proces koji se planira.

```
qk.set_and_sync(loct);  
loct += sc_time(5, SC_NS);
```

Listing Koda 5

set_and_sync metoda prikazana u listing koda 5, postavlja trenutni offset i proverava da li je potrebna sinhronizacija i ukoliko jeste, sinhronizuje se sa globalnim kvantom vremenom.

sc_time pošto je u pitanju sistem koji oslikava realan rad sistema, korišćeno je i dodatno kašnjenje, upotrebom pomenute funkcije kao što je prikazano u listing koda 5.

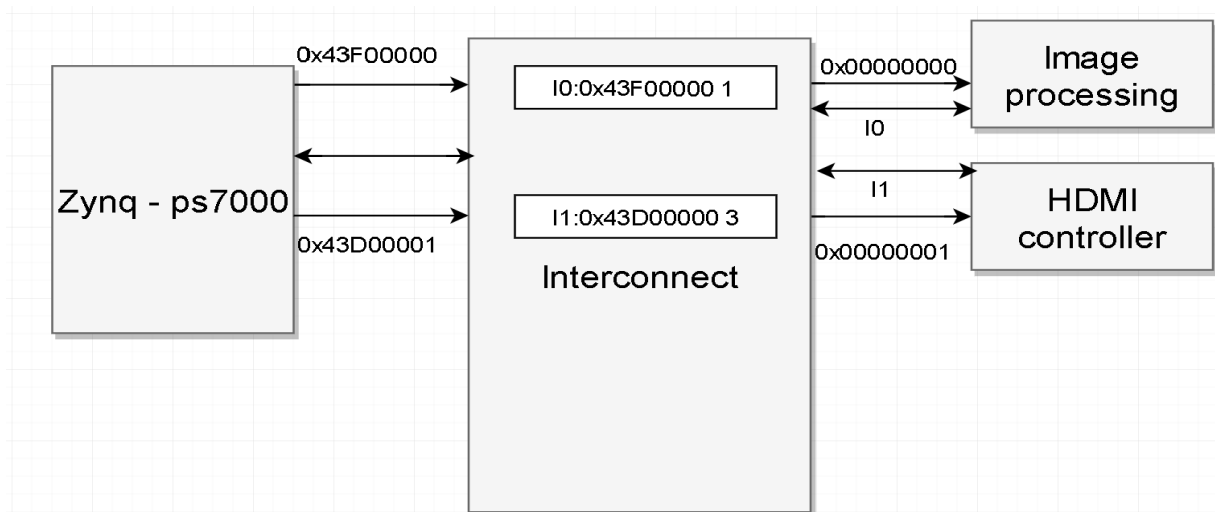
Na isti način se vrši i čitanje podataka, stim što se umesto argumenta **TLM_WRITE_COMMAND**, koristi **TLM_READ_COMMAND**.

3.2.2 Model interkonekta u virtuelnoj platformi

Pre objašnjenja interkonekta, potrebno je uvesti i razumeti pojam memorijskog mapiranja. Pomenuti blok ne treba znati ništa o blokovima koji su priključeni na njega. Ti blokovi predstavljaju adrese u odnosu na interkonekt. Ovo predstavlja pojednostavljeno objašnjenje značenje pojma memorijsko mapiranje. Na ovaj način svaka od blokova, komponenti koja je vezana na blok za unutrašnje konekcije dobija jedinstvenu adresu, kako bi se laički objasnilo, svoju ličnu kartu. Prilikom dodeljivanja adresa svakoj od komponenti, jako je važno obratiti pažnju da svaka komponenta dobije jedinstvenu adresu kako ne bi došlo do problema prilikom poziva određene komponente. Ove jedinstvene adrese nazivaju još i bazne adrese. Kako bi se bolje razumeo pojam memorijskog mapiranja, u nastavku je dat primer.

Na slici 9 je prikazan primer memorijskog mapiranja na kome se nalaze tri komponente. To su interkonekt kao i dva bloka za obradu podataka koji nose nazive Image processing i HDMI (eng *High-Definition Multimedia Interface*)[12] kontroler čija funkcionalnost trenutno nije važna tako da o tome neće biti reč. Bazna adresa bloka za obradu slike je 0x43D00000, dok je bazna adresa HDMI kontrolera 0x43F00000. Svaki blok poseduje svoj memorijski podsistem koji se sastoji od registara ili drugih potrebnih memorija. Recimo da HDMI kontroler ima 3 registra, na taj način se dolazi do informacije da je potrebno koristiti tri adrese. A to su 0x43D00000, 0x43D00001, 0x43D00002. Vrednosti 0x00000000, 0x00000001, 0x00000002 predstavljaju lokalne adrese unutar samog bloka i njihovim sabiranjem sa 0x43D00000 dobijaju se pomenute adrese koje predstavljaju adrese u jedinstvenom adresnom prostoru. Interkonekt sadrži tabelu u kojoj svaki interfejs ima svoju navedenu baznu adresu. Takođe se tu nalaze i opseg lokalnih adresa. Na primeru sa slike 9, kao što je već pomenuto postoji dva bloka koji su interfejsima I0 i I1 vezani za interkonekt. Interfejs I0 sadrži opseg od jedne adrese dok je opseg interfejsa I1 tri adrese.

Primer: Recimo da procesor želi da komunicira sa drugim registrom HDMI kontrolera, tada se od strane procesora pošalje adresa 0x43D00002 ka interkonektu. Tada se od te adrese, oduzme bazna adresa, 0x43D00002 - 0x43D00000 i kao rezultat se dobija lokalna adresa drugog registra, to jest 0x00000002.



Slika 9. Memorijsko mapiranje

Interkonekt je implementiran pod nazivom `sys_bus.hpp`. Sadrži jedan jednostavan inicijatorski priključak `s_myiproc`, kao i jedan ciljni priključak pod nazivom `s_cpu`. Inicijatorski priključak se vezuje za dizajn koji služi za obradu slike dok se ciljani priključak koristi za povezivanje sa ostatkom sistema, u ovom slučaju sa procesorom.

```
sys_bus::sys_bus(sc_module_name name) :sc_module(name){  
s_cpu.register_b_transport(this, &sys_bus::b_transport);}
```

Listing Koda 6

Konstruktor pomenutog modula registruje metodu `b_transport` na ciljani priključak što se može videti na listingu koda 6. Pomenuta metoda vrši prevođenje adrese iz jedinstvenog adresnog prostora u lokalne adrese komponenti. Kako bi se olakšalo praćenje funkcionalnosti i radi lakših izmena uglavnom se umesto brojeva koriste konstante. U ovom slučaju je to konstanta `VP_ADDR_MYIP`. Ta adresa se nalazi unutar fajla `vp_addr.hpp`. Kada je potrebno koristiti tu adresu, potrebno je samo uključiti poseban `hpp` fajl u kome se nalaze pomenute adrese. Dobra praksa je koristiti poseban `hpp` fajl za smeštanje adresa jer bi kompleksnijim projektima na ovaj način je moguće lakše snalaženje kao i izmena samih adresa na jednom mestu.

```
if(addr == VP_ADDR_MYIP) {  
taddr = addr & 0x0000000F;  
pl.set_address(taddr);}
```

Listing Koda 7

Kao što se vidi na listingu 7, jedinstvena adresa se prevede u lokalnu u slučaju da je tražena adresa ona iz opsega, to jest u ovom slučaju `VP_ADDR_MYIP`. Ako to nije slučaj, ispiše se poruka u grešci. Na kraju izvršavanja se ponovo postavlja jedinstvena adresa i na ovaj način prevođenje adresa nema uticaja na blok koji je instancirao prenos podataka.

3.2.3 Model dizajna za obradu slike u virtuelnoj platformi

Model koji se projektuje predstavlja komponentu koja služi za obradu slike, to jest uveličavanje slike dva puta. Naime korištene su već gotove funkcije tako da ovaj model može da radi bilo koju obradu slike dok će se implementacija u sledećoj glavi ograničiti samo na uveličavanju slika dva puta.

Ovaj model u svojoj biblioteci Zoom2IP.hpp sadrži događaje:

- ewrite
- eread
- can_continueWrite
- can_continueRead

Događaji su korišćeni pri komunikaciji metoda unutar samog modela. O ovome će biti reči nešto kasnije. Ovaj model takođe sadrži i dva objekta koji su klasnog tipa Image u kome se nalaze sve gotove funkcije vezane za obradu slike. Potrebno je uvezati biblioteku `#include <Magick++.h>` kao i korisnički prostor `using namespace Magick[13]`.

Takođe, nalaze se i tri metode, gde prva, `ImageResizing()` vrši obradu slike, druga `b_transport(pl_t&, sc_core::sc_time&)` je zadužena za slanje i primanje potrebnih informacija i treća `msg(const pl_t&)` služi za čitanje trenutnih transakcija.

1. Metoda zadužena za prenos podataka, `b_transport` prima određene podatke kao parametre za prvi parametar dok drugi parameter predstavlja vreme. Potrebno je pročitati koja adresa je pozvana, kao i komanda i podatak kao što je prikazano na listingu 8.

```
tlm_command_cmd = pl.get_command();
uint64 addr = pl.get_address();
unsigned char* data = pl.get_data_ptr();
```

Listing Koda 8

Potrebno je odrediti da li je komanda **TLM_WRITE_COMMAND/TLM_READ_COMMAND**. Ako je u pitanju čitanje, tada se ispituje vrednost `val`. Ako `val` ima vrednost 1. Tada se u odnosu na vrednost `rcount` upisuje slika koja je obrađena, kao što se može videti na listingu 9.

```
if(rcount>=1){
    ewrite.notify(1, sc_core::SC_NS);
    wait(can_continueWrite);};
```

Listing Koda 9

U protivnom ako vrednost `rcount` nije u datom opsegu, ispisuje se poruka "Frist, you must read the picture". U slučaju da je vrednost `val` 0, tada se vrši čitanje slike što je i prikazano na listingu koda 10.

```
case 0:
eread.notify(1, sc_core::SC_NS);
wait(can_continueRead);
```

Listing Koda 10

U slučaju da komanda ima vrednost **TLM_READ_COMMAND**, tada se vrši čitanje podataka u zavisnosti od vrednosti Irprocess.

2. Metoda zadužena za obradu slike ImageResizing() ne prima nikakve argumente, već se unutar nje koriste instancirani objekti a i b kako bi se odradila određena obrada podataka. Tok po kome funkcija izvršava svoj zadatak je prikazan na listingu koda 11.

```
b = a;
b.resize("300x300");
```

Listing Koda 11

Zbog opisanog naziva svake funkcije, nije potrebo previše govoriti o samim funkcijama. U prvoj liniji se čita slika sa određene putanje, u sledećoj se slika prikazuje. Nakon toga b preuzima vrednost a, zatim se podaci koji se nalaze u b obrađuju i nakon toga se obrađena slika zapisuje na određenu lokaciju. Unutar same metode se kao i kod već pomenute metode koriste događaji kako bi se ispratio pouzdan rad svih metoda. Kreiran je poseban fajl pod nazivom vp.cpp. Unutar ovog fajla se povezuju svi pomenuti blokovi kao što je prikazano na listingu koda 12.

```
s_cpu.register_b_transport(this, &vp::b_transport);
s_bus.bind(sb.s_cpu);
sb.s_myiproc.bind(t.soc);
```

Listing Koda 12

Korišćenjem bind metode povezuje se model za obradu slika sa interkonekt modulom.

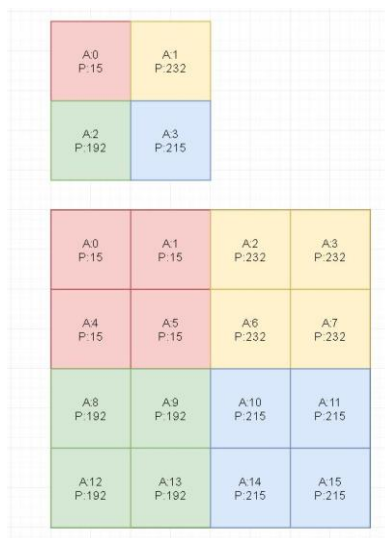
Glava 4

4.1 Projektovanje hardverskog IP bloka

U ovom poglavlju biće objašnjen implementirani algoritam upotrebnom RT metodologije. Takođe biće prikazan controlpath i datapath projektovanog bloka.

4.1.1 Algoritam najbližeg komšije (eng *Nearest Neighbor Algorithm*)

Algoritam najbližeg komšije predstavlja prilično efikasan algoritam koji se koristi pri prepoznavanju uzoraka (eng *Pattern recognition*), računarskoj geometriji (eng *Computational geometry*), pri interpolaciji podataka (eng *Interpolating data*), u grafovima, teoriji verovatnoće, pri rešavanju problema putujućeg prodavca (eng *Travelling salesman problem*). Potrebno je projektovati modul koji uvećava dobijenu sliku dva puta. Pomenuti algoritam je odabran ne samo zbog svoje efikasnosti već i zbog jednostavnosti projektovanja. Primer rada datog algoritma sledi na slici 10.



Slika 10. Algoritam najbližeg komšije na primeru slike 2 puta 2

Na ovom jednostavnom primeru vidi se da se crveni piksel koji se nalazi na adresi nula upisuje na adrese 0, 1, 4 i 5, žuti piksel koji se nalazi na adresi 1 se upisuje na adrese 2, 3, 6 i 7, zeleni piksel sa adrese 2 se upisuje na adresu 8, 9, 12 i 13, dok se plavi piksel sa adrese 3 upisuje na adrese 10, 11, 14 i 15. Kao što se vidi sa slike broj kolona i broj redova je uvećan dva puta, što je i bio cilj algoritma. A predstavlja adresu dok P podatak (od 0 do 255).

Tabela2 Adrese upisanih piksela obrađene slike

Rnewaddress1	0	2	8	10
Rnewaddress2	1	3	9	11
Rnewaddress3	4	6	12	14
Rnewaddress4	5	7	13	15

Tabela3 Algoritam za generisanje adrese na kojoj će biti upisan piksel

$Rnewaddress1 = (r * MaxRow_plus_2) + (r * r0cend) + startOffset1 + (c * 2);$
$Rnewaddress2 = (r * MaxRow_plus_2) + (r * r0cend) + startOffset2 + (c * 2);$
$Rnewaddress3 = (r * MaxRow_plus_2) + (r * r0cend) + startOffset3 + (c * 2);$
$Rnewaddress4 = (r * MaxRow_plus_2) + (r * r0cend) + startOffset4 + (c * 2);$

Tabela4 Objašnjenje svih članova matematičkog iskaza

r	Brojač redova
c	Brojač kolona
MaxRow _plus_2	Vrednost u trenutku kada je brojač redova na najvećoj vrednosti , zatim je ta vrednost sabrana sa 2
r0cend	Vrednost u trenutku kada je brojač redova na najmanjoj vrednosti i brojač kolona na maksimalnoj vrednosti
Start Offsetx	Početna adresa unutar svakog matematičkog iskaza (startOffset1=1. startOffset2=1. startOffset3=4. startOffset4=5)

U tabelama 2, 3 i 4 je prikazan matematički algoritam po kome se dobijaju adrese u kojima će biti upisan određeni piksel. Kao što se može videti, postoji četiri matematičke formule koji predstavljaju četiri matrice dva puta dva. Na ovaj način se slika dobijena kamerom uveličava dva puta. U slučaju uveličavanja četiri puta, jasno je da će postojati osam matematičkih iskaza, to jest broj kolona će se uvećati četiri puta, a broj redova će se isto uvećati četiri puta, dakle četiri plus četiri je osam.

4.2 Projektovanje IP modula korišćenjem RT metodologije

RT (eng *Register Transfer*) metodologija predstavlja metodologiju gde je predstavljen prenos podataka između registara u tačno određenim taktovima. Koristeći ovu metodu moguće je algoritme impementirati u hardver. Pre samog procesa projektovanja digitalnog sistema, potrebno je napisati pseudo kod koji predstavlja algoritam ciljanog IP modula. Pseudo kod algoritma za obradu slike se može videti na listingu koda 13.

```
for(c;c=col;c++){
  counter++;
  RWR=1;
  if(controlR='1'){
    Raddress=r*row+c;
    Rdata=mem_out;
  }
  if(counter_reg>0)
    memin=Rdata;
}
```

Listing Koda 13

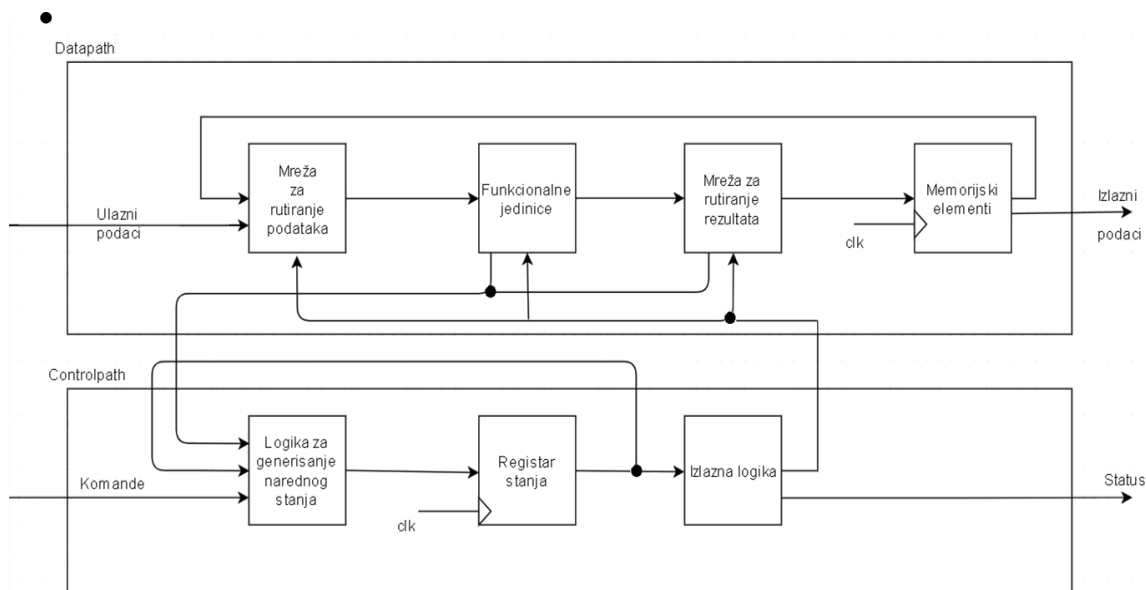
Na listingu koda 13 je prikazan deo pseudokoda koji predstavlja čitanje piksela originalne slike kao i upisivanje piksela unutar memorije koja je deo dizajna o kome će biti više reči kasnije.

```
Rnewaddress=(r*MaxRow_plus_2)+(r*r0cend)+startOffset1+(c*2);
Rnewdata= mem_out;
if(c=0&&r=0)
  RnewWR='0';
else
  RnewWR='1';
```

Listing Koda 14

Na listingu koda 14 predstavljen je deo algoritma gde se čitaju podaci o originalnoj slici koji se nalaze unutar memorije koja se nalazi unutar dizajna. Zatim se na određenim adresama upisuju podaci koji predstavljaju informacije nove, uvećane slike. To jest koristeći četiri matematička iskaza se određuje adresa na kojoj će se upisati željeni piksel. Pomenut pseudo kod je takođe napisan i kao matlab skripta. Model poređenja je korišten pri površnoj verifikaciji IP (eng *intellectual property*) modela. Detaljna verifikacija će biti pomenuta u glavi o verifikaciji upotrebom UVM (*Universal Verification methodology*). Proces projektovanja digitalnog sistema korišćenjem RT metodologije sastoji se iz sledećeg koraka:

- **Eliminacija naredbi** – To znači da je potrebno eliminisati sve naredbe ponavljanja u algoritmu (**for**, **repeat**, **while** petlje) koristeći se **if-goto** naredbama. Ovo je potrebno uraditi jer implementacija ASMD dijagrama sa povratnom petljom i početnim uslovom postaje jasnija. To jest nakon eliminacije naredbi ponavljanja inženjer ima jasan uvid u korak gde je potrebno konstruisanje ASMD dijagrama.
- **Definisanje interfejsa** – Potrebno je da se uoče ulazne i izlazne promenjive u algoritmu i takođe je potrebno odrediti potrebne širine portova. Pored promenljivih potrebno je ubaciti statusne i komandne interfejse.
- **Projektovanje *controlpath*** – na osnovu algoritma potrebno je kreirati ASMD:
 - Dijagram algoritma. Potrebno je razmotriti potencijalne optimizacije u pogledu dostupnog paralelizma.
 - Ovo je od posebnog značaja za dataflow aplikacije. Potrebno je obratiti pažnju da korišćenje paralelizma ubrzava rad sistema ali troši više resursa.
- **Projektovanje *datapath*** – sastoji se iz četiri koraka:
 - Identifikacija unutrašnjih promenljivih, alokacija i dimenzionisanje registara dodeljeni ovim promenjivama. Potrebno je kreiranje liste RT operacija.
 - Grupisanje RT operacija u odnosu na njihove odredišne registre. Sve operacije sa istim odredišnim registrima se smeštaju u istu grupu.
 - Dodavanje rutirajućih kola ispred registara.
 - Formiranje statusnih signala korišćenjem kombinacione logike.
- **Pisanje HDL modela** – U ovom koraku je potrebno napisati model dizajna koristeći jezik za opis hardvera. Za ovaj korak preporučen je dvoprocesni stil modelovanja u kom jedan proces opisuje kombinacionu logiku, a drugi sekvencijalnu. U koliko je potrebna posebna kontrola za neki blok, on se može odvojiti u poseban proces.



Slika 11. Struktura Controlpath-a i Datapath-a

Na slici 11 se nalazi tipična struktura čitavog sistema. Sistem je podeljen na dva dela, na datapath i controlpath.

Datapath sadrži tri modula:

- **Memorijski elementi** – u memorijske elemente spadaju registri koji čuvaju rezultat kao i međurezultat.
- **Funkcionalne jedinice** – pomoću funkcionalnih jedinica se realizuju potrebne operacije kao što su sabiranje, oduzimanje, množenje kao i operacije pomeranja kao i mnoge druge operacije.
- **Mreža za rutiranje** – Povezuju izlaze registara do funkcionalnih jedinica, kao i za povezivanje izlaza funkcionalnih jedinica do ulaza registara. Uglavnom su sastavljeni od multipleksera.

Datapath sadrži sledeće interfejs:

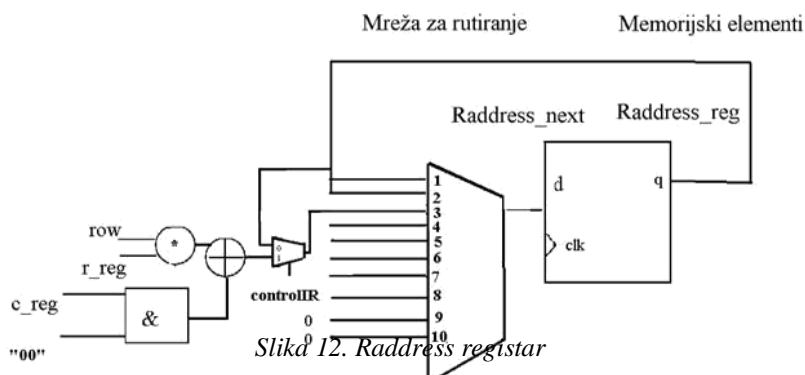
- **Ulazni interfejs podataka** – podaci se dovode do datapath-a.
- **Izlazni interfejs podataka** – rezultati izvršavanja algoritma se prosleđuju u spoljašnjem okruženju.
- **Kontrolni interfejs** – pomoću koga controlpath modul upravlja radom datapath modula.
- **Statusni interfejs** – preko koga se controlpath-u šalju informacije o trenutnom statusu datapath modula, na osnovu čega se donose odluke o narednom stanju.

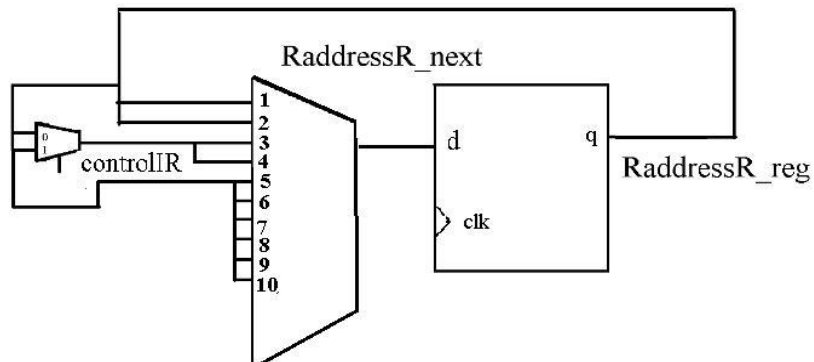
Na narednim slikama će biti prikazan datapath modula za obradu slike. Zbog kompleksnosti DataPath dela kao i veličine iste, DataPath će biti prikazan iz više slika.

Tabela5 Legenda ulaza na multiplekseru

Broj	Naziv
1	Idle
2	Prepare
3	WriteToRaddressCOL
4	WriteToRaddressROW
5	WriteToRaddressLastBit
6	WriteToRnewaddressCOL
7	additionalClock
8	WriteToRnewaddressROW
9	additionalClockRow
10	WriteToRnewaddressEmpty

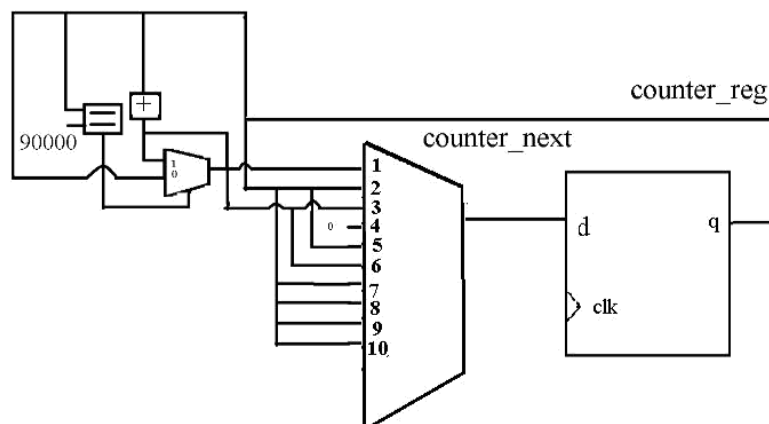
Na slici 12 je prikazan Raddress registar u koji se upisuje adresa koja predstavlja adresu originalne slike. Kako bi se ispoštovala širina magistrala, potrebno je na vrednost brojača c_reg dodati dve nule. Izvršava se matematička operacija množenja vrednosti brojača r_reg sa konstantom row. U slučaju da se controlIR 1 rezultat se prosleđuje kroz multiplekser i smešta se u registar. Po sličnom principu funkcioniše i RaddressR registar koji se prikazan na slici 13.





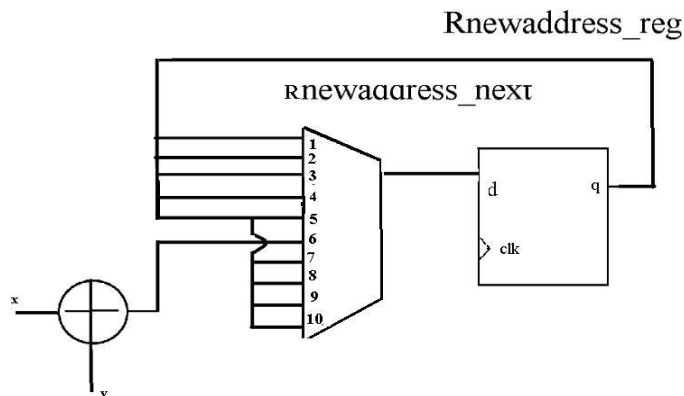
Slika 13. RaddressR registar

Counter registar prikazan na slici 14 služi isključivo kao uslovni registar, to jest obezbeđuje pravilan protok potrebnih informacija. Sasotji se iz sabirača, komparatora, dva multipleksera kao i jednog d registra.

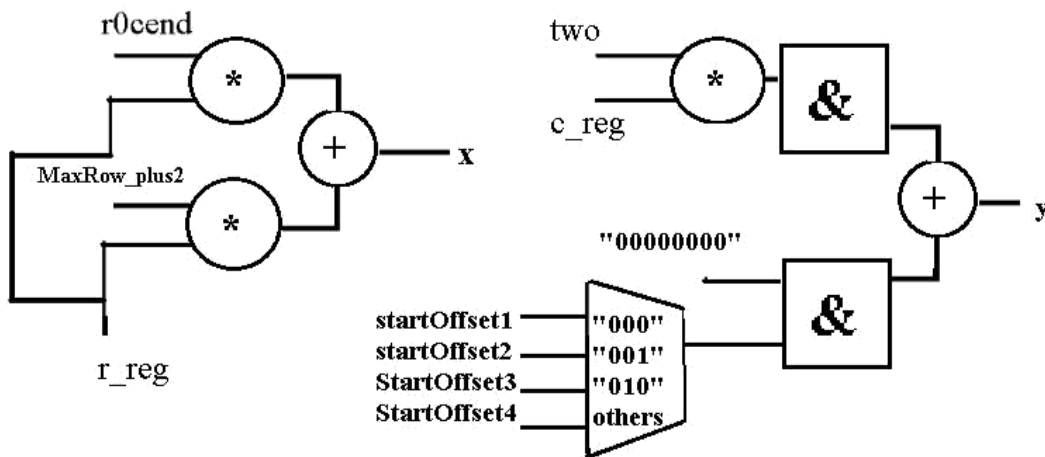


Slika 14. counter registar

Rnewaddress registar koji se nalazi na slici 15, predstavlja registar u kome se skladišti informacija o adresi koja se odnosi na adresu uveličane slike, to jest adresu polja na koji je potrebno upisati određenu vrednost koja se nalazi na izlaznom portu RnewData. Zbog obimnosti slike 15, na slici 16 se nalaze ulazi u sabirač nazvani x i y. StartOffset1,2,3,4 predstavljaju vrednosti koje se dodaju na trenutnu vrednost Rnewaddress. To zavisi od uslova koji govori koji put se petlja izvršava. Objašnjenje na pojednostavljenom primeru sa slike 10, to bi bile adrese A0, A1, A4 ili A5 za crvenu boju, respektivno i za ostale boje.

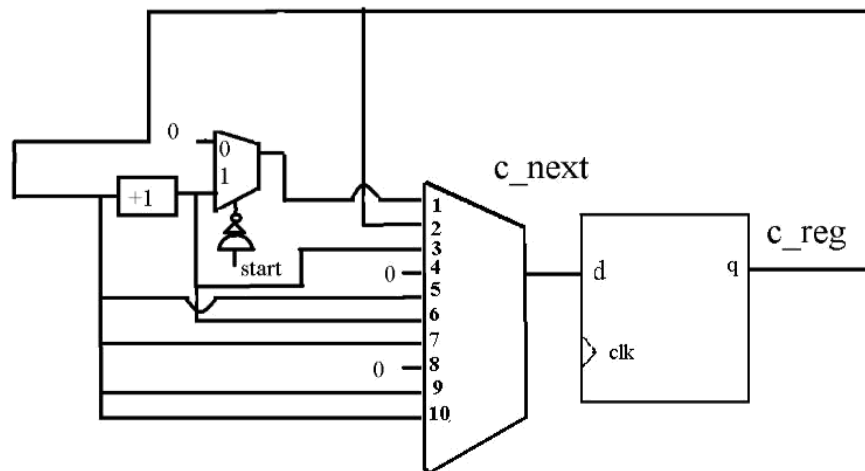


Slika 15. Rnewaddress registar

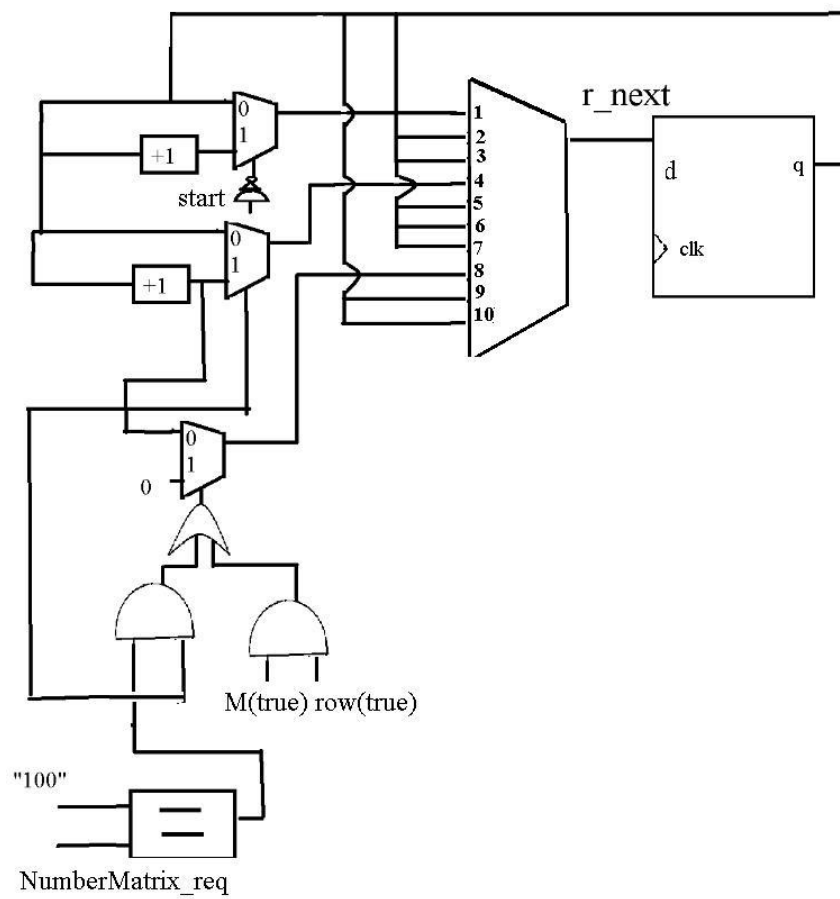


Slika 16. x i y

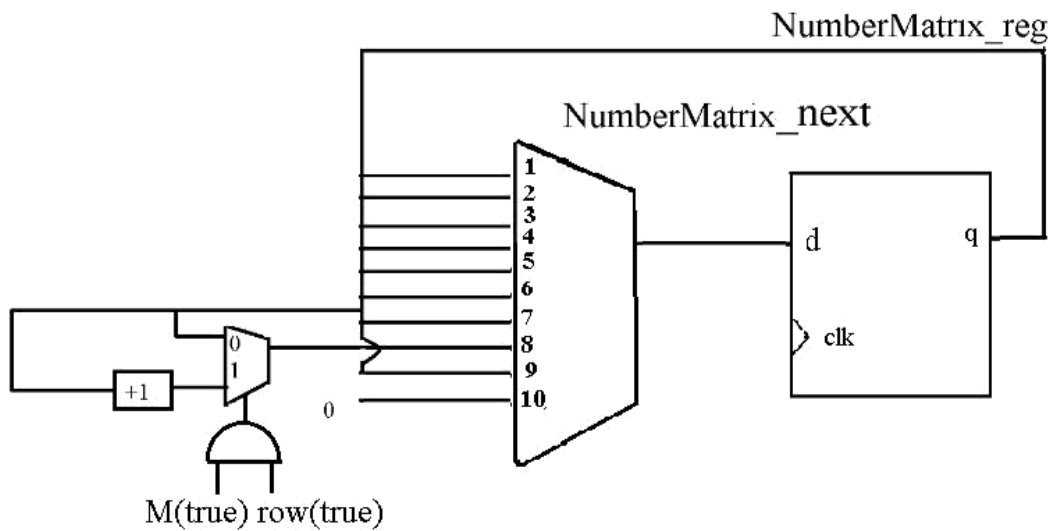
C i r registri predstavljaju kolone, to jest vrste dvodimenzionalne matrice. Kolone i vrste su implementirane kao brojači koji svojim uvećavanjem kao rezultat daju određenu adresu na kojoj će se upisati određeni podatak. Kao što se može videti na slikama 17 i 18, c i r registri se sastoje od velikog broja multipleksera pomoću kojih se određuje koji od ulaza se prosleđuje na izlaz, lokičkih kapija (I,ILI,NI), sabirača i komparatora. NumberMatrix registar prikazan na slici 19 predstavlja uslovni registar od koga zavisi koji offset će se dodati na početnu adresu, dok se na slici 20 nalazi registar mem_out. Registar mem_out predstavlja sponu pri prosleđivanju podataka iz blok memorije koja se nalazi unutar dizajna. Naime, prvi deo procesa je čitanje podataka sa adresa originalne slike i upisivanje tih podataka u pomenuti blok ram. U drugom delu procesa, unutar blok rama uslovi za upis u blok ram više nisu ispunjeni tako da se sada aktivira čitanje, to jest mem_out registar čita podatke iz blok rama i prosleđuje ih na izlaz dizajna, direktno na izlazni port za podatke, Rnewdata.



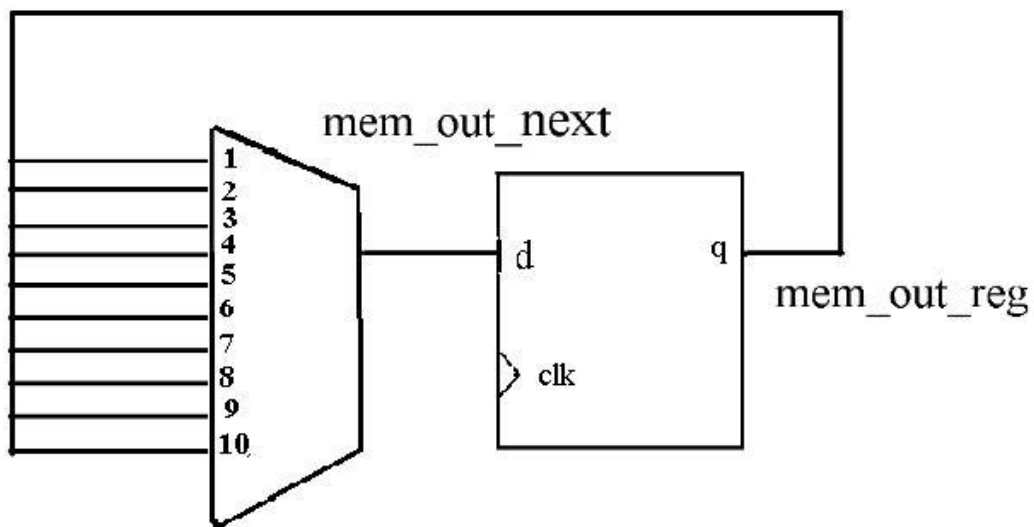
Slika 17. c registar



Slika 18. r registar



Slika 19. NumberMatrix registar



Slika 20. mem_out registar

Ove slike predstavljaju registre sa svojim funkcionalnim jedinicama i mrežom za rutiranje.

Funkcionalne jedinice predstavljaju logička kola (i,ili,ne kolo), sabirači, upoređivači i množači. Mreža za rutiranje predstavljaju multiplekseri dok D registri predstavljaju memoriju.

Controlpath modul:

- Logika za generisanje narednog stanja – Određivanje narednog stanja u FSM (eng *Finite State Machine*). U logici narednog stanja učestvuju trenutno stanje, ulazi u sistem i statusni signali iz datapath- a.
- Registar stanja – čuva trenutno stanje.
- Izlazna logika – generiše kontrolne i statusne signale. Funkcija izlaza može da bude Murovog ili Milijevog tipa.

Interfejsi Controlpath-a:

- Komandni interfejs – predstavlja ulaze preko kojih se može upravljati digitalnim sistemom.
- Izlazni statusni interfejs –sistem prosleđuje informacije o unutrašnjim stanjima preko izlaznog statusnog interfejsa.
- Kontrolni interfejs – upravljački signali se šalju prekog kontrolnog interfejsa.
- Ulazni statusni interfejs – interfejs preko kog controlpath prima informacije od datapath-a.

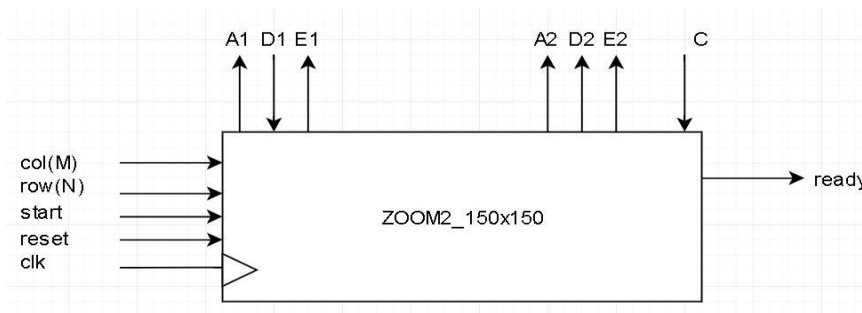
Da bi se algoritam opisao preko FSM, koriste se dve grafičke reprezentacije:

- Dijagrami stanja,
- ASM (*Algorithmic State Machine*) dijagram.

Dijagrami stanja se sastoje od čvorova, obeleženih krugovima i jednosmernim strelicama koje određuju pravac tranzicije. Svaki čvor predstavlja jedinstveno stanje i ima jedinstveno ime. Svaka strelica ima definisan uslov prelaza u sledeće stanje i po porebi Milijeve izlaze, dok su dodele Murovih izlaza uvek zapisani u sklopu stanja.

ASM dijagrami se mogu koristiti za predstavljanje kompleksnih sekvenci događaja koje uključuju ulazne komande i akcije na izlazu. Ova vrsta reprezentacije je više opsina od dijagrama stanja i lako joj se može pridružiti datapath, čime se dobija ASMD dijagram. Nakon ovoga se ASMD reprezentacija može jednoznačno transformisati u jezik za opis hardvera. Osnovni ASM blok se sastoji od polja stanja, uslova prelaza u naredno stanje, kao i od izlaznog polja u kome se opciono definišu Milijevi signali.

Kao što je navedeno u postupcima RT metodologije, sledeći korak je pisanje HDL koda. Zoom2_150x150 vhd fajl koristi utils_pkg.vhd paket koji predstavlja funkciju za određivanje minimalnog broja bita koji treba da se zauzme za određenu širinu.



Slika 21. Zoom2_150x150 modul

Na slici 21 je prikazan modul za obradu slike koji sadrži ulazne i islazne portove za komunikaciju sa okolinom koja će biti pomenuta kasnija. Objašnjenje svakog od ovih protova je prikazana u tabeli 5.

Tabela6 Objašnjenje portova modula za obradu slike

Naziv	Širina u bitima	Funkcija
col	8	Broj kolon
row	8	Broj redova
start	1	Pokretanje modula
clk	1	Sinhronizacionija modula
reset	1	Signal za dodelu poč. vrednosti
Raddress(A1)	16	Adresa piksela originalne slike
Rdata(D1)	8	Informacija, piksel (0-255)
RWR(E1)	1	Kontrolni signal
Rnewaddress(A2)	18	Adresa piksela obrađene slike
Rnewdata(D2)	8	Informacija, piksel (0-255)
RnewWR(E2)	1	Kontrolni signal
ControlR(C)	1	Komunikacioni signal
ready	1	Statusni signal spremnosti

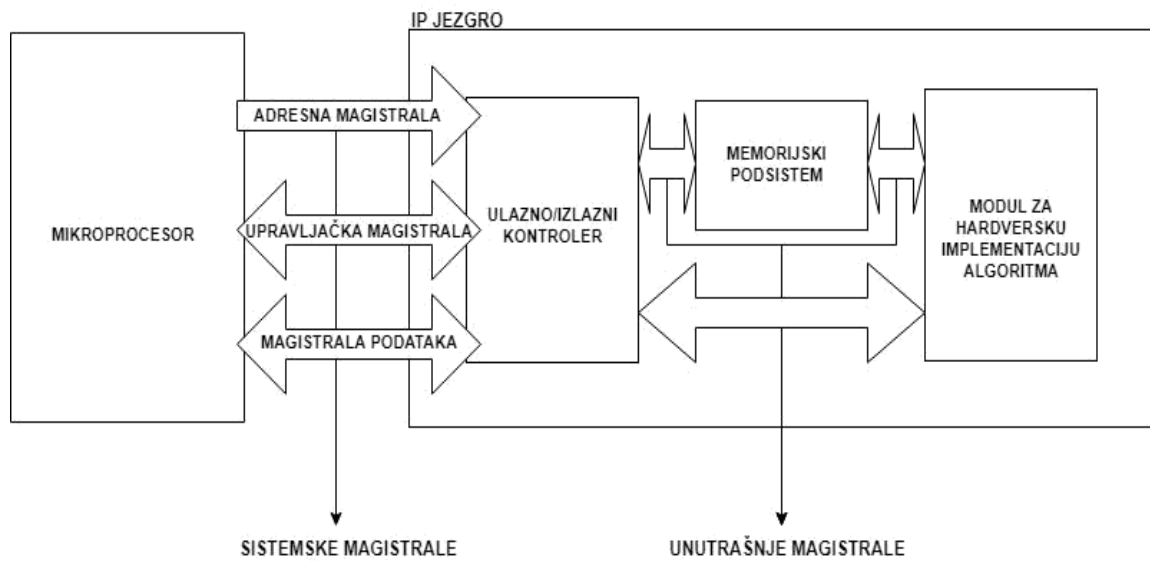
4.3 Oklopljavanje IP modula

IP modul koji je projektavan ne predstavlja najveći nivo hijerarhije već kao gradivni blok unutar nekog većeg sistema. Pomenuti složeni sistem se naziva sistem na čipu (eng *System on chip*). SoC sistemi po pravilu su bazirani na jednom ili više procesorskih jezgara, koji su putem jedne ili više sistemskih magistrala povezani sa memorijskim i komunikacionim modulima, kao i sa modulima za specijalizovanu obradu podataka. Za razmenu informacija koriste sistemske magistrale. IP modul za obradu podataka je povezan sa AXI4-Lite i AXI4-Full interfejsom[14].

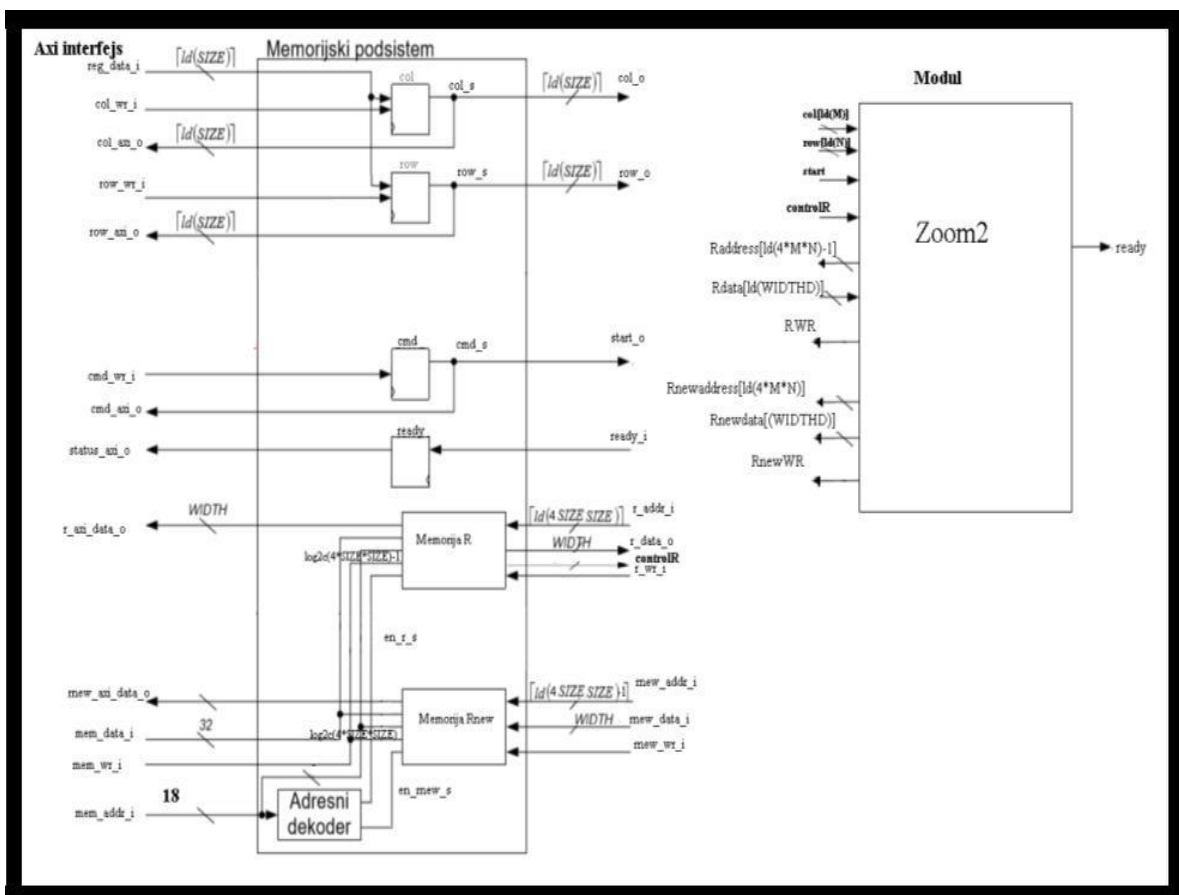
AXI4-Lite – pojednostavljena verzija AXI4-Full interfejsa, koja se takođe koristi za komunikaciju sa memorijski mapiranim modulima, ali kod kojih ne postoji mogućnost prenosa podataka u blokovima, već je moguće prenositi samo jedan podatak po transakciji. Usled toga su hardverski resursi neophodni za implementaciju AXI4-Lite kontrolera znatno manji od resursa potrebnih za implementaciju AXI4-Full kontrolera. Takođe je pojednostavljen i sam interfejs, jer je dobar deo signala iz AXI4-Full interfejsa nepotreban[14].

AXI4-Full – je AXI interfejs visokih performansi, prikladan za komunikaciju sa memorijski mapiranim modulima kod kojih je podržan prenos podataka u blokovima od maksimalno 256 transfer ciklusa[14].

Implementirani algoritam, memorijski podsistem, ulazno izlazni kontroler zajedno čine IP jezgro. Tipična struktura IP jezgra je prikazana na slici 22 i predstavlja “zamotani” blok koji može da komunicira sa mikroprocesorom na određenoj razvojnoj ploči. Na slici 23 je prikazana sprega memorijskog podsistema, AXI interfejsa i modul za hardversku implementaciju algoritma. Na slici 23 se nalazi Zoom2 modul, memorijski podsistem kao i AXI interfejs.



Slika 22. Tipična struktura IP jezgra

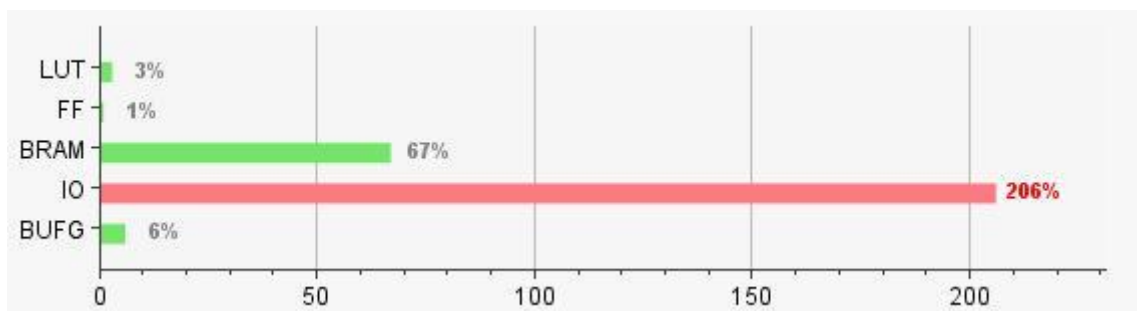


Slika 23. Oklopljeno jezgro

Testiranje funkcionalnosti oklopljenog jezgra:

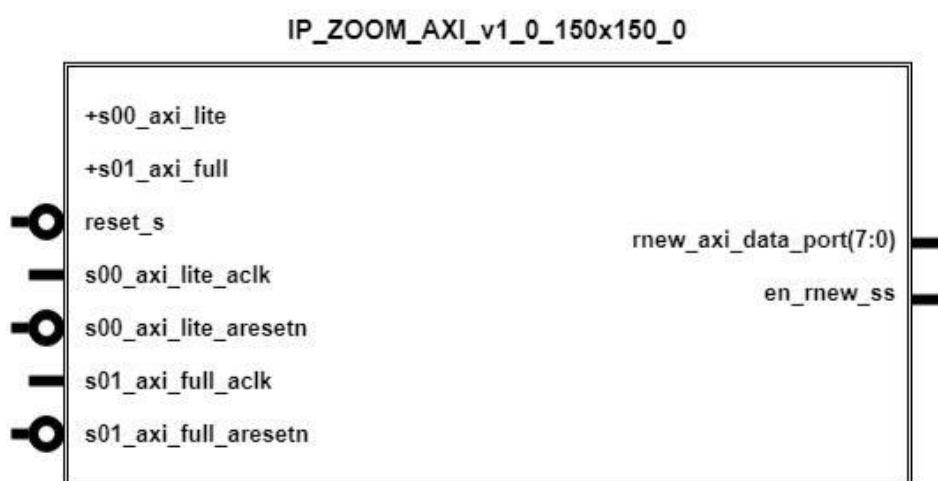
1. Koristeći matlab skriptu RecPic_TransBits.m čita se slika dobija internet kamerom. Ta slika je trodimenzionalna matrica gde su vrednosti od 0 do 255. Rezolucija slike je 150x150 to jest 22500 piksela. Ova skripta prikuplja informacije o slici na taj način što razdvaja sliku na 3 odvoje matrice veličine 150x150. Sledeći korake upisivanje pomenutih piksela u tri tekstualna fajla (FirstDim.txt, SecondDim.txt, ThirdDim.txt).
2. Zoom2_150x150TB.vhd predstavlja test vhd fajl koji prvo pročita podatke iz gore pomenutih .txt fajlova. Ti podaci su upisani u test matricu. Zatim se postave signali za upis (AXI Lite interfejs) gde se indirektno preko registara col i row unutar memorijskog podsistema upisuju podaci o broju redova i kolona unutar Zoom2_150x150.vhd modula. Zatim se postave signali za upis (AXI Full interfejs) i na taj način se napuni blok memorija R. Zatim se postave signali za upis (AXI Lite interfejs) gde modul dobija startni signal. Modul preko adresne magistrale, magistrale podataka i kontrolne magistrale prima podatke iz podsistemske memorije i te podatke o originalnoj slici šalje u blok memoriju koja se nalazi unutar modula. Nakon završetka tog procesa, modul preko adresne magistrale, magistrale podataka i kontrolne magistrale obrađene informacije u blok Memorija Rnew. Ti podaci predstavljaju novu sliku koja je dva puta veća u odnosu na originalnu sliku. Zatim se postave signali za čitanje (AXI Full interfejs) gde se informacije iz blok Memorija Rnew čitaju podaci i upisuju u test memoriju.
3. Zoom2_150x150TB.vhd upisuje podatke u .txt fajlove.
4. Matlab skripta RecMatrix_ShowNewPic.m prihvata podatke iz .txt fajlova pomenutih u koraku broj 3. Pomenuta skripta prikazuje uvećanu sliku.
5. Matlab skripta Compare_ReferentModel_IPmodel.m upoređuje rezultate dobijene korišćenjem modela za upoređivanje implementiran unutar ReferentModel.m i .txt fajlova FirstDimnew.txt, SecondDimnew.txt, ThirdDimnew.txt. Ako je sve uredi, ispisuje se poruka koja kaže da je sve uredi.

Nakon provere funkcionalnosti potrebno je proveriti i sintezu oklopljenog jezgra kako bi se oklonili određeni nedostaci i dobio uvid o količini potrošenih resursa.



Slika 24. Iskorišćenost hardvera

Na slici 24 se jasno vidi iskorišćenost BRAM-a od 67% što je jedan od glavnih nedostataka implementacije iz razloga što se unutar oklopljenog jezgra nalazi 3 blok memorije. Na ovaj način je implementacija jednostavnija ali je i iskorišćenost resursa mnogostruko veća. Nakon sinteze, potrebno je oklopiti jezgro.



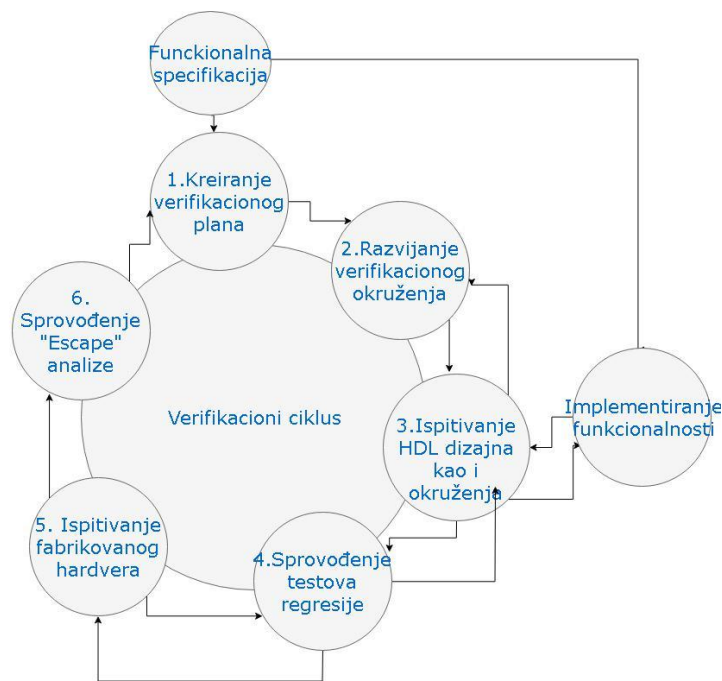
Slika 25. Konačno oklopljavanje jezgra

Na slici 25 se nalazi oklopljeni blok koje će se dalje koristiti pri implementaciji celokupnog sistema na razvojnoj ploči.

Glava 5

5.1 Verifikacija Zoom2 IP bloka

U predhodnoj glavi je projektovano hardversko jezgro koje uvećavava sliku dva puta. Za projektovano jezgro kreirana je jedinična provera (eng *test unit*) koja proverava funkcionalnost samog jezgra. Taj test svakako nije sigurnost da dizajn funkcioniše ispravno. Sam proces potpune verifikacije dizajna predstavlja ogromnu i kompleksnu nauku koja prevazilazi mogućnosti diplomskog rada. Propust tokom verifikacije može napraviti velike troškove u kompanijama, tako da je od velike važnosti da se greške pronađu u ranim tokovima verifikacije.



Slika 26. Verifikacioni ciklus

Verikacija se može podeliti u korake prikazane na slici 26. Svaki od koraka je ukratko opisan na narednoj strani.

Funkcionalna specifikacija predstavlja dokument koji dobija verifikacioni i dizajn tim. Taj dokument predstavlja zahteve mušterije. Kratko objašnjenje svakog dela u verifikacionom ciklusu:

1. Verifikacioni tim pravi verifikacioni plan na osnovu dobijene specifikacije. Tu se razmatra šta je sve i na koji način potrebno verifikovati.
2. Projektovanje verifikacionog okruženja.
3. Predstavlja pregled i otklanjanje grešaka u dizajnu kao i okruženju.
4. Pokretanje testova regresije.
5. Nakon fabrikovanja, potrebno je ponovo proveriti pomenuti dizajn.
6. Escape analiza - Jako je važno da se otklone greške, dok u slučaju ako se iz nekog razloga greška pojavljuje i nakon fabrikovanja inženjeri uče iz grešaka tako da sledeća verifikacija bude unapređena, bez pojave pomenutih grešaka.

5.1.1 UVM metodologija i SystemVerilog

UVM (eng *Universal Verification Methodology*)[15] je standardizovana metodologija za funkcionalnu verifikaciju sa pomoćnom bibliotekom u SystemVerilog jeziku. Kreirana je sa željom da se postigne lakša ponovna upotreba testbenčeva i jednostavno kreiranje univerzalnih, visoko-kvalitetnih VIPa (eng *Verification Intellectual Property*).

UVM je baziran na OVM-u (eng *Open Verification Methodology*)[16] i eRM-u (eng *e Resuse Methodology*)[17]. Jedna od glavnih karakteristika ove metodologije je UVC (eng *Universal Verification Component*), odnosno univerzalne verifikacione komponente koje imaju istu strukturu (sadrže monitore, drajvere, sekvencere) što omogućava lako korišćenje bilo kako nezavisne komponente ili kao deo većeg sistema.

Objektno-orijentisani dizajn, kao glavna karakteristika SystemVerilog jezika, omogućava lako kreiranje verifikacionih komponenti, dok veliki broj predefinisanih funkcija i taskova znatno ubrzava proces kreiranja testova. UVM obezbeđuje osnovu za verifikaciju zasnovanu na pokrivenosti, koja ne zahteva kreiranje velikog broja testova, osigurava temeljnu verifikaciju na osnovu zadatih parametara i olakšava proces pronalaženja problema. Ovo se postiže korišćenjem samoproveravajućih (eng *self-checking*) testbenčeva, automatskog generisanja testova i korišćenjem podataka o pokrivenosti.

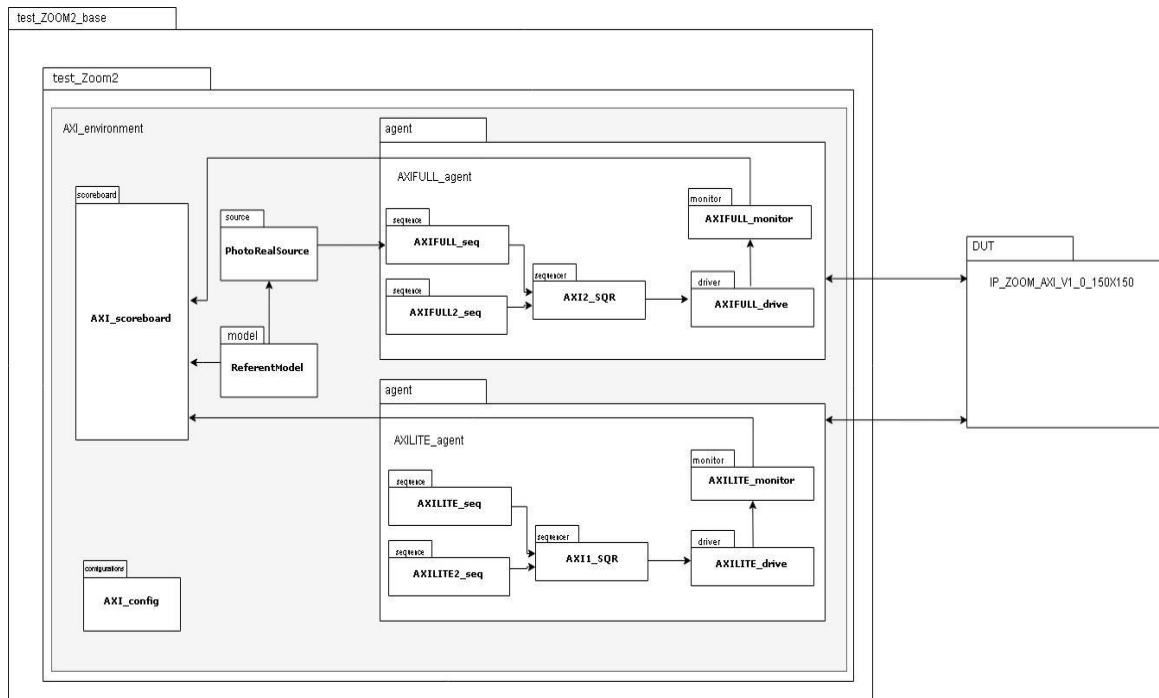
Opšta struktura verifikacionog okruženja se sastoji iz:

- Data objekat - Unutar ovih fajlova nalaze se polja verifikovanog dizajna i ako je potrebno polja za kontrolu kao i potrebna ograničenja.
- Sekvencer predstavlja stimulus generator u kome se kontrolišu podaci koji se šalju sledećoj komponenti, a to je drajver.
- Drajver je aktivna komponenta koja emulira signale, koje treba poslati dizajnu koji verifikuje. Na verifikacioni dizajn je povezan preko virtuelnog interfejsa koji se koristi u UVM metodologiji.
- Monitor je pasivna komponenta koja nadgleda signale.
- Agent obuhvata tri komponente, a to su sekvencer, drajver, monitor. Postoje aktivni agent koji obuhvata sve tri komponente dok pasivni agent obuhvata samo monitor.
- Scoreboard predstavlja komponentu koja prikuplja i upoređuje podatke.
- Konfiguracioni fajl sadrži sve potrebne informacije vezane za test bench, na primer broj potrebnih agenata, da li je pasivan ili aktivan agent i tako dalje.
- Okruženje (eng *Environment*) obuhvata sve gore pomenute delove. Predstavljeno je kao klasa koja se može jednostavno koristiti kada i gde je potrebno.
- Test

5.2 Projektovanje verifikacionog okruženja za Zoom2 IP blok

Verifikaciono okruženje se sastoji iz gore navedenih komponenti koje zajedno čine verifikaciono okruženje čija je funkcija da provere funkcionalnost dizajna za obradu slika

Verifikaciono okruženje se može videti na slici 27.



Slika 27. Unit test

5.2.1 Projektovanje data objekta

Projektovani dizajn za komunikaciju krositi AXI LITE i AXI FULL protokole tako da postoje dva fajla koji predstavljaju data objekte, a to su `Data_Sequence_AXILITE.sv` i `Data_Sequence_AXIFULL.sv`.

Data objekti sadrže polja koja su potrebna za verifikaciju projektovanog dizajna, kao i potrebna ograničenja. Na listingu broj 15 su prikazana neka od potrebnih polja unutar data objekta, gde vrednosti polja nisu randomizovani dok je treća linija randomizovano polje.

```
bit [14:0] s01_axi_awlen_s;  
bit [2:0] s01_axi_awsizs_s;  
rand bit [31:0]  
s01_axi_wdata_s;
```

Listing Koda 15

Polja rand označavaju promeljive vrednosti koja će se postaviti u koliko se pozove ugrađena metoda klase – randomize. Zajedno sa ovom metodom se pozivaju i metode pre_randomize i post_randomize u koliko su definisane. Randomizovano polje predstavlja vrednosti piksela koja su u opsegu od 0 do 255 što predstavlja širinu od 8 bita, tako da nije potrebno vršiti ograničenja datih polja. Ostala polja koja predstavljaju AXI LITE i AXI FULL polja za komunikaciju nisu randomizovana jer njihova randomizacija nema smisla iz razloga što protokoli rade po tačno određenim pravilima i jasno je da kada se ta pravila ne poštuju, dolazi do grešaka u radu.

5.2.2 Projektovanje stimulus slika

PhotoRealSource.m predstavlja matlab skriptu koja učitava slike, podešava potrebnu rezoluciju, deli vrednosti u tri matrice i zapisuje u tekstualni fajl. Skriptu je moguće videti u listing koda 16. Kao što je već rečeno, ova komponenta se koristi samo u unit testu.

```
R=uint8 (255*rand(row,col))  
Rbin=dec2base(R,2);  
dlmwrite(.\MatlabFirstDim.txt',R(),',delimiter',' ')  
dlmwrite(.\FirstDim.txt',Rbin(),'-append',',delimiter',' ','newline','pc')
```

Listing koda 16

5.2.3 Projektovanje sekvenci

Sekvenca se sastoji od transakcija koje predstavljaju važne informacije koje treba poslati preko sekvencera i drajvera u dizajn. Sekvence su objekti, a ne komponente. Pri kreiranju sekvenci, potrebno je naslediti baznu klasu **uvm_sequence#(*)**;

Unutar verifikacionog okruženja, koristi se šest sekvenci, tri za AXI LITE protokol, a tri za AXI FULL protokol. Uobičajena praksa je da se kreira bazna sekvenca koja nasleđuje gore pomenutu baznu klasu koja kao argument(*) prima data objekat.

Unutar bazne sekvence nalaze se zadaci(eng *tasks*) koji su zaduženi za podizanje i spuštanje objekata. Potrebno je registrovati sekvencu i napomenuti koji sekvencer će se koristiti kao što je prikazano na listingu 17.

```
`uvm_object_utils (AXILITE2_seq)
`uvm_declare_p_sequencer(AXI1_SQR)
```

Listing Koda 17

Ostale sekvence nasleđuju ovu baznu sekvencu i unutar njih se piše ostatak funkcionalnosti, to jest transakcije koje se šalju. Komunikacija se može izvršiti na osnovu već predefinisanih koraka ili koristeći makroe. Nekada je potrebno manipulirati pažljivo svakim korakom tako da se makroi izbegavaju. Pomenuti koraci su prikazani na listingu 18.

```
req = Data_Sequence_AXILITE::type_id::create("req");
req.s00_axi_awvalid_s=1; start_item(req);
finish_item(req);
```

Listing Koda 18

Makro pomenut u listingu 19 se koristi kada je potrebna veća kontrola. Kao što se može videti određenim poljima su dodeljene vrednosti. Ova polja predstavljaju komunikaciona polja AXI LITE protokola. Ostatak koda se može videti u priloženoj dokumentaciji. Potrebne su povratne informacije od strane drajvera, tako je potrebno zatražiti informaciju nakon završenog razgovora sa drajverom.

```
finish_item(req);
axi_read_data_v=req.s00_axi_rdata_s;
```

Listing Koda 19

5.2.4 Projektovanje sekvencera

Sekvencer koristi sekvence kako bi poslao podatke drajveru, ali i prima odgovor od drajvera ukoliko je to potrebno čime se omogućava kreiranje korisnih stimulusa. Sekvencer služi kao arbiter čiji je zadatak kontrola slanja transakcija iz jedne ili više sekvenci.

U praksi najčešće nije potrebno modifikovati sekvencer jer su funkcionalnosti koje se nalaze u **uvm_sequencer** klasi obično dovoljne. Kao i drajver, i sekvencer je moguće parametrizovati sa tipom transakcije.

Kreirana su dva sekvencera. Jedan se koristi za prenos podataka za AXI LITE protokol dok se drugi koristi za prenos podataka AXI FULL protokola.

```
class AXI1_SQR extends uvm_sequencer#(Data_Sequence_AXILITE);  
  `uvm_component_utils(AXI1_SQR)  
  function new(string name = "AXI1_SQR", uvm_component parent = null);  
    super.new(name,parent);  
  endfunction  
endclass : AXI1_SQR
```

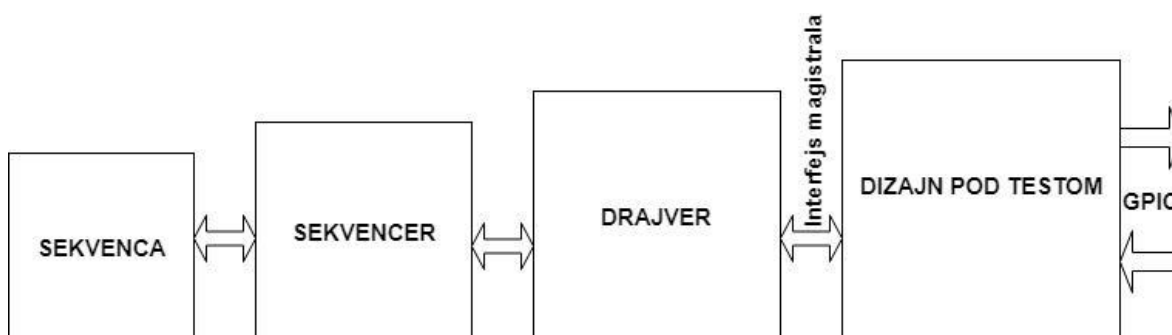
Listing Koda 20

U listing koda 20 je prikazana komponenta AXI1_SQR koja predstavlja sekvencer vezan za AXI LITE protokol. Nasleđuje se uvm klasa koja kao parametar prima data objekat koji sadrži polja potrebna za verifikaciju AXI LITE protokola.

5.2.5 Projektovanje drajvera

Drajver je aktivna komponenta koja emulira signale koji se šalju dizajnu. Na osnovu podataka iz sequence item-a i smplovanja signal na interfejsu, postavlja signale na ulaze dizajna koji se verifikuje. Transakcije koje prima su obično na višem nivou apstrakcije, pa se vrši konverzija na pin nivo apstrakcije kako bi se ispravno generisali signali. Drajver je preko TLM porta povezan na sekvencer kako bi vršio komunikaciju i sadrži virtuelni interfejs kako bi generisao signale. U UVM-u se drajver implementira nasleđujući uvm_driver #(REQ,RSP) klasu, koja je parametrizovana tipom transakcija koju de drajver zahtevati i odgovarati (eng *request, response*).

Postoji dva načina komunikacije sa sekvencerom. Korišćeni način je `get_next_item / item_done` mehanizmom. Ovaj mehanizam omogućava drajveru da uzme transakciju od sekvencera iz sekvenci, da je procesira i zatim kompletira rukovanje koristeći `item_done`. Nije preporučeno prosleđivanje argumenata uz `item_done`. Ovaj način komunikacije je preferiran i čest u praksi jer obezbeđuje jasno odvajanje između drajvera i sekvence. Kao što je već pomenuto potrebno je da drajver pošalje povratne informacije sekvenceru kako bi se moglo nastaviti sa slanjem transakcija ka verifikovanom dizajnu. Iz tog razloga se koristi dvosmerni model koji nema podršku paralelizma (eng *Bidirectional Non-Pipelined*). Blok šema pomenutog modela drajvera je prikazan na slici 28.



Slika 28. Bidirekcionni model bez podrške paralelizma

Sekvencer šalje zahteve drajveru koji ih izvršava, a zatim šalje odgovor nazad. Nova transakcija ne može biti započeta do god se ne primi odgovor i ne završi prethodna. Drajver prima transakciju, po protokolu koji se implementira pošalje podatke uz eventualno čekanje i zatim postavi polja u transakciji koja služe za odgovor i završi rukovanje sa sekvencerom. Pošto se i u drajveru i u sekvenci koristi pokazivač na isti objekat, nakon što se sekvenca odblokira (posle `finish_item`) može koristiti objekat sa novim vrednostima i vršiti dalju analizu. Primer komunikacije u vidu signala je prikazan na slici 29.



Slika 29. Primer komunikacije drajvera (Preuzeto iz [2])

Projektovana su dva drajvera na osnovu pomenutog modela. Prvi šalje dizajnu podatke potrebne za pravilnu upotrebu AXI LITE protokola dok drugi šalje podatke potrebne za upotrebu AXI FULL protokola kao i podatke koji predstavljaju vrednosti piksela. Nije moguće povezati drajver direktno sa dizajnom već je potrebno drajver i dizajn povezati na virtuelni interfejs koji će biti objašnjen u sledećem delu ovog poglavlja.

```
virtual interface IP_ZOOM_AXI_v1_0_150x150_if vif;
```

Listing Koda 21

Na listingu 21 je prikazano instanciranje gore pomenutog virtuelnog interfejsa koji će se koristiti u tasku koji je zadužen za slanje podataka ka dizajnu. Deo taska se može videti na listing 22.

```
`uvm_info("AXIFULL_drive","WRITE VALUES",UVM_HIGH)
if(req.first_or_middle_or_last_wdata==5) begin
`uvm_info("AXIFULL_drive","PRIPREMA",UVM_HIGH)
@(negedge vif.s01_axi_full_aclk);
/*1*/vif.s01_axi_full_awaddr = req.s01_axi_awaddr_s;
/*2*/vif.s01_axi_full_awlen = req.s01_axi_awlen_s;
/*3*/vif.s01_axi_full_awsz = req.s01_axi_awsz_s;
/*4*/vif.s01_axi_full_awburst = req.s01_axi_awburst_s;
/*5*/vif.s01_axi_full_awvalid = req.s01_axi_awvalid_s;
wait(vif.s01_axi_full_awready ==1'b1)
wait(vif.s01_axi_full_awready ==1'b0)
@(negedge vif.s01_axi_full_aclk);
end
```

Listing Koda 22

Prenos podataka na listingu 22 je osetljiv na rastuću ivicu sinhronizujućeg signala. Leva strana izraza predstavlja instancu virtualnog interfejsa koji prima podatke poslate koji su pristigli iz sekvencera. Ovaj task se poziva u run fazi.

```
req.s00_axi_rdata_s = vif.s00_axi_lite_rdata;
```

Listing Koda 23

Kao sto je već pomenuto potrebna je komunikacija u oba smeru, tako da su na listingu koda 23 prikazani izrazi koji predstavljaju komunikaciju od drajvera ka sekvenceru.

Pomenuti virtuelni interfejs je potrebno kreirati tako što će se u njemu nalazi sva polja koja sadrži i dizajn koji se verifikuje, to jest potrebna su polja za pravilno upravljanje AXI LITE protokola, AXI FULL protokola kao i podataka koji predstavljaju sliku.

```
interface IP_ZOOM_AXI_v1_0_150x150_if
(input s00_axi_lite_aclk,input s01_axi_full_aclk,bit reset_s, bit s00_axi_lite_aresetn, bit
s01_axi_full_aresetn);
/*1*/logic[C_S00_AXI_LITE_ADDR_WIDTH - 1 : 0]s00_axi_lite_awaddr;
/*2*/logic[2 : 0]s00_axi_lite_awprot;
.....
endinterface : IP_ZOOM_AXI_v1_0_150x150_if
```

Listing Koda 24

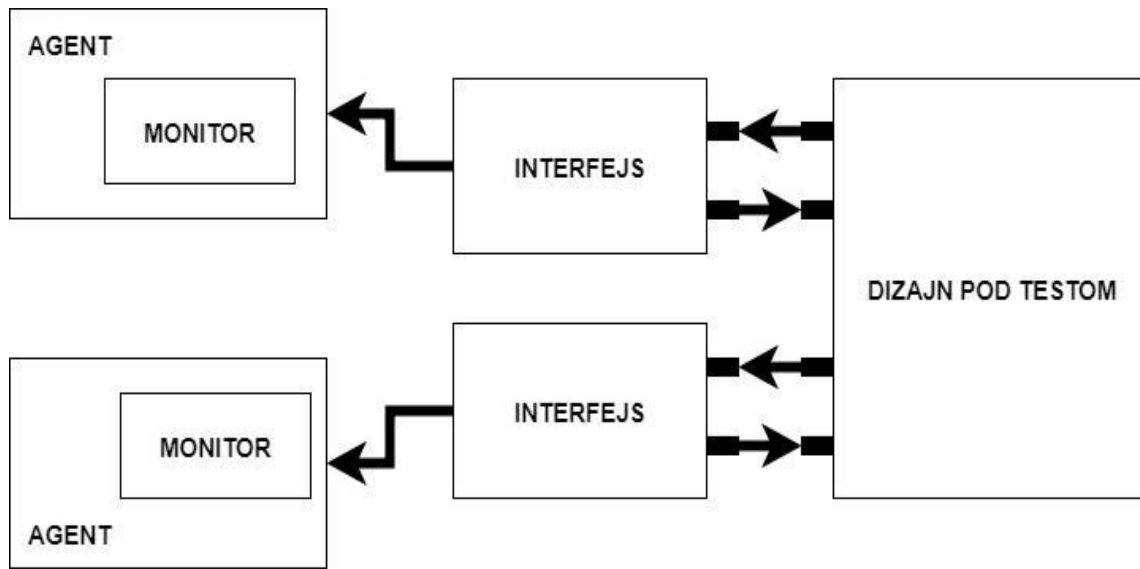
Na listingu koda 24 je prikazan deo korišćenog virtuelnog interfejsa koji je spojen sa drajverom i verifikovanim dizajnom. Veza između povezanih komponenti se uglavnom piše unutar zasebnog fajla. Deo koda u kome se povezuje virtuelni interfejs i dizajn je prikazan na listingu koda 25.

```
IP_ZOOM_AXI_v1_0_150x150_verif_top.sv.
.s00_axi_lite_awaddr      ( virtual_interface.s00_axi_lite_awaddr ),
.s00_axi_lite_awprot      ( virtual_interface.s00_axi_lite_awprot ),
.s00_axi_lite_awvalid     ( virtual_interface.s00_axi_lite_awvalid ),
.s00_axi_lite_awready     ( virtual_interface.s00_axi_lite_awready ),
```

Listing Koda 25

5.2.6 Projektovanje monitora

Korišćenje TLM (eng *Transaction Level Modeling*) interfejsa omogućava izolaciju komponenti tako da promene u okruženju ne utiču na datu komponentu. Monitor prima podatke sa interfejsa i zatim se podaci šalju ka sekvenceru. Primer monitora je prikazan na slici 30.



Slika 30. Komunikacija monitora

Monitor predstavlja pasivnu komponentu koja nadgleda signale, a nema mogućnost generisanja signala. Potrebna su dva monitora kako bismo nadgledali vrednosti pomenutih protokola .

```
uvm_analysis_port #(Data_Sequence_AXIFULL)
item_collected_port;
virtual interface IP_ZOOM_AXI_v1_0_150x150_if vif;
Data_Sequence_AXIFULL current_transaction;
```

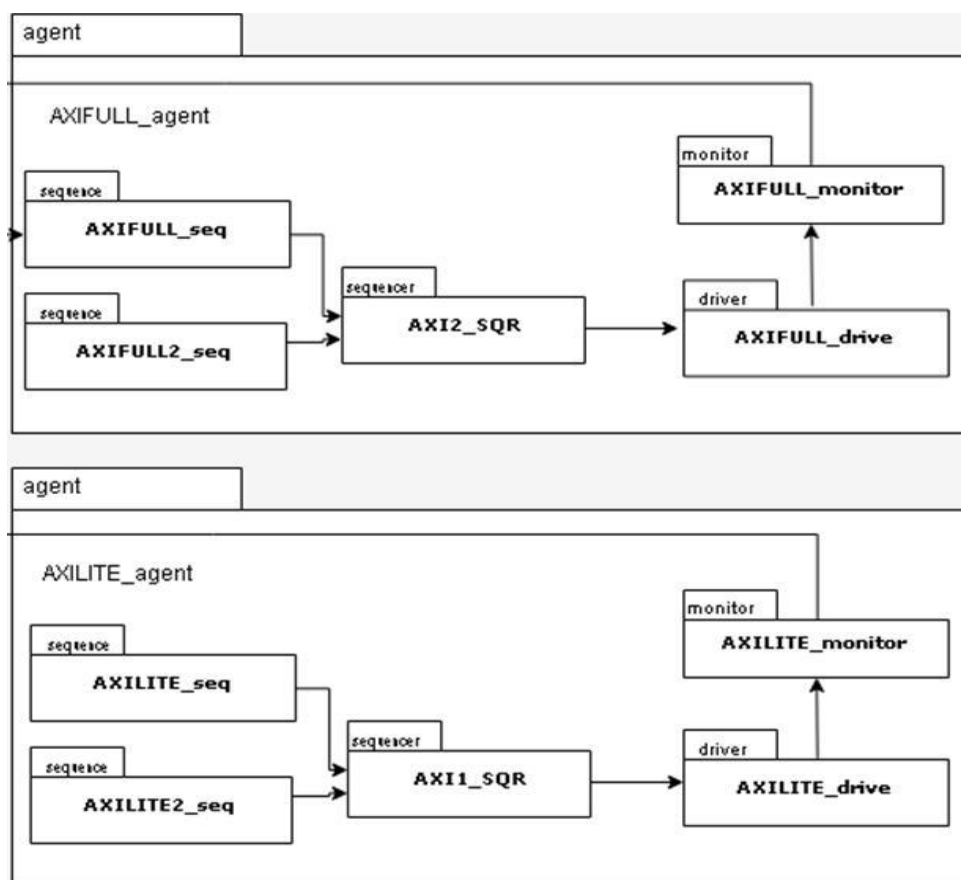
Listing Koda 26

Potrebno je napraviti instance virtuelnog interfejsa kao i data objekta kao što je prikazano na listingu koda 26.

Kao što je već rečeno, preko monitora se primaju podaci sa virtuelnog interfejsa i zatim se šalju preko porta za analizu(uvm_analysis_port#(trans) ap) dalje ka scoreboardu.

5.2.7 Projektovanje agenta

Agent obuhvata tri komponente, a to su sekvencer, drajver i monitor, kao što je prikazano u listingu koda 27. Agent predstavlja apstraktni pristup interfejsu i olakšava korišćenje ovih komponenti. Postoje aktivni i pasivni. Aktivni agent sadrži komponente koje generišu simulacione signale na interfejsu kao i one koje posmatraju signale, dok pasivni agent samo nadgleda interfejs odnosno pasivni agent sadrži samo monitor zbog čega je veoma bitno da su drajver i monitor potpuno odvojeni iako im je deo funkcionalnosti sličan. Pasivni agent se uglavnom koriste kao mali deo nekog većeg okruženja kada se stimulus generiše iz neke druge komponente.



Slika 31. Agenti

Projektovana su dva agenta kao što je prikazano na slici 31. Jedan obuhvata komponente vezane za AXI LITE protokol, dok drugi obuhvata komponente vezane za AXI FULL protokol kao i podatke vezane za sliku.

```
AXILITE_drive drv;  
AXI1_SQR seqr;  
AXILITE_monitor  
mon
```

Listing Koda 27

Potrebno je napraviti instancu kako pomenutih komponenti tako i konfiguracije. Ako je vrednost polja `is_active == UVM_ACTIVE` tada se kreiraju objekti `drv` i `seqr`, kao što je prikazano na listingu koda 28.

```
mon = AXILITE_monitor::type_id::create("mon",  
this); if(cfg.is_active == UVM_ACTIVE) begin  
drv = AXILITE_drive::type_id::create("drv", this);  
seqr = AXI1_SQR::type_id::create("seqr", this);  
end
```

Listing Koda 28

Ako je pomeunto polje aktivno to jest ima vrednost `UVM_ACTIVE` u fazi za povezivanje se povežu sve tri komponente i to predstavlja aktivni agent, u suprotnom je pasivni agent u pitanju. Pomenuti kod je prikazan na listingu 29.

```
if(cfg.is_active == UVM_ACTIVE) begin  
drv.seq_item_port.connect(seqr.seq_item_export);  
end
```

Listing Koda 29

Blok nivo predstavlja environment klasu u kojoj se nalaze sve komponente za ponovnu upotrebu. U njoj se vrši konfiguracija svih podkomponenti. Većina ponovne upotrebe se vrši na ovom nivou odnosno korisnik instancira datu `uvm_env` klasu i konfiguriše agente prema svojim potrebama.

Environment klasa je prikazana na slici slici 27 pod nazivom AXI_env. Deo koda gde se povezuju komponente je prikazanu u listingu koda 30 gde su povezani agenti na AXIScoreboard.

```
function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
LITEagent.mon.item_collected_port.connect(AXIScoreboard.port_lite);
FULLagent.mon.item_collected_port.connect(AXIScoreboard.port_full);
endfunction : connect_phase
```

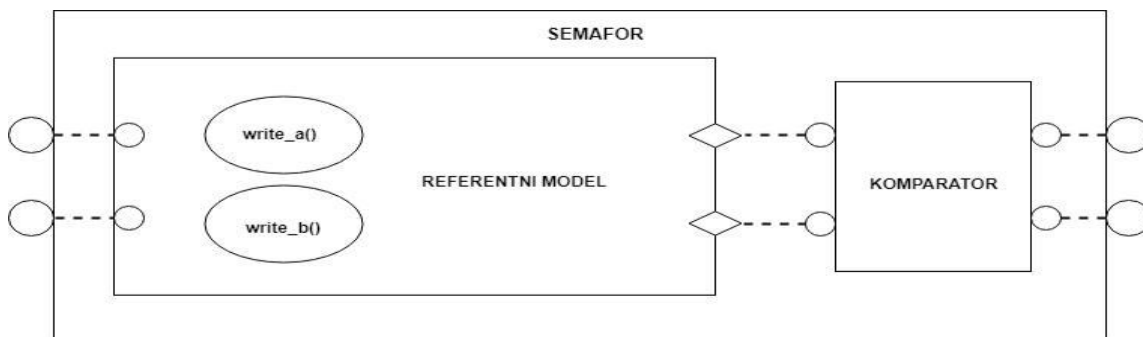
Listing Koda 30

Top-nivo okruženje sadrži potreban broj agenata. Sadrži još i ostale komponente potrebne za proveru rada dizajna, na primer scoreboard, globalni monitor, prikupljač pokrivenosti i tako dalje. Sadrži i konfiguracionu klasu. Preko konfiguracione klase se mogu podešavati neke osobine, jedan od primera je konfiguracija samih agenata.

5.2.8 Projektovanje skorborda

Glavni element okruženja je scoreboard. U scoreboardu se vrše provere funkcionalnosti sistema koji se verifikuje. Sadržaj i uloga scoreboard-a dosta zavisi od implementacije, u većini slučajeva je povezan sa ostatkom okruženja preko TLM interfejsa preko kojih prima transakcije. Transakcije se analiziraju pomoću referentnog modela ili zlatnih vektora i poredi očekivane rezultate sa dobijenim transakcijama.

Skorbord se sastoji od prediktora i evaluatora kao što je prikazano na slici 32.



Slika 32. Tipična struktura scoreboard-a

Prediktor predstavlja referenti model koji se koristi za prođenje sa rezultatima koji su dobijeni iz dizajna, dok bi zlatni vektori predstavljali gotove rezultate koji se takođe porede sa rezultatima dobijenim iz dizajna. U ovom projektu je korištena hibridna varijanta, to jest kombinacija oba. Naime komponenta koja predstavlja prediktor je napisana kao skripta u matlabu, pod nazivom ReferentModel.m.

Pomenuti prediktor funkcioniše tako što se na ulaz upišu isti podaci kao oni koji su poslali dizajnu koji se verifikuje. Zatim se određenim algoritmom obrađuju podaci i rezultat se zapisuje u tekstualni fajl koji se nakadno učitava u evaluator koji predstavlja upoređivač u pomenutom skorbordu. Deo pomenutog algoritma koji se koristi u prediktoru se nalazi na listingu 31.

```

for r = 1:row
for c = 1:col
Rnew(r+VerticalOffsetLow,c+HorisontalOffsetLow,1)= R(r,c,1);
Rnew(r+VerticalOffsetLow,c+HorisontalOffsetHigh,1)= R(r,c,1);
Rnew(r+VerticalOffsetHigh,c+HorisontalOffsetLow,1)= R(r,c,1);
Rnew(r+VerticalOffsetHigh,c+HorisontalOffsetHigh,1)=
R(r,c,1); HorisontalOffsetLow=HorisontalOffsetLow+1;
HorisontalOffsetHigh=HorisontalOffsetHigh+1;
if c==col
HorisontalOffsetLow=0;
HorisontalOffsetHigh=1;
end
end
VerticalOffsetLow=VerticalOffsetLow+1;
VerticalOffsetHigh=VerticalOffsetHigh+1;
if r==row
VerticalOffsetLow=0;
VerticalOffsetHigh=1;
end
end

```

Listing Koda 31

Kao rezultat algoritma prikazanog na listigu koda 31 dobija se matrica koja je duplo veća u odnosu na početnu, to jest ulaznu matricu.

5.2.9 Projektovanje pokrivenosti

Pokrivenost je metrika koja se koristi za merenje progressa i završetka verifikacije. Daje informacije o tome kada se neki deo dizajna aktivirao tokom simulacije i da li postoje delovi dizajna koji nikad nisu bili aktivirani. Dve najčešće korišćene coverage metrike su:

- Strukturna pokrivenost (eng *code coverage*) – implicitna , potrebno je proveriti koji procenat koda je aktiviran, to jest dali je i koliko puta određeni kod aktiviran.
- Funkcionalna pokrivenost (eng *functional coverage*) – eksplicitna, cilj funkcionalne verifikacije je utvrditi da li dizajn implementira sve osobine i funkcioniše na način opisan u funkcionalnoj specifikaciji. Međutim do zaključka o tome da li je neka funkcionalnost stvarno implemntirana i da li je verifikovana, ne možemo doći na osnovu praćenja pokrivenosti koda. Zbog toga se uvodi nova, eksplicitna metrika – funkcionalna pokrivenost. Cilj ove metrike je merenje progressa verifikacije u odnosu na funkcionalne zahteve dizajna.

Zaključak

ESL metodologija nam omogućava brz razvoj modela koji predstavlja verodostojan hardverski model. Na taj način su uviđeni potencijalni nedostaci kao i prednosti dizajna. Glavni nedostatak dizajna predstavlja implementacija blokovske memorije unutar samog IP modula kao i unutar memorijskog podsistema. Taj nedostatak se ogleda u tome što su gotovo svi blok resursi potrošeni koristeći rezoluciju slike svega 22500 (150x150) piksela kao i uvećanu sliku veličine 90000 (300x300) piksela. Veće dimenzije bi preopteretile postojeće resurse. Prednost ovakve implementacije je jednostavnost. Alternativno rešenje umesto korišćenja blok memorije, da se koristi DRAM (eng *Dynamic random access memory*) koje ima mnogostruko više ali bi u tom slučaju bilo potrebo ubacivanje DMA-a (eng. *Direct Memory Access*)[18]. Prateći RT metodologiju, NNI (*Nearest Neighbor Interpolation*) algoritam je implementiran u hardver. Nedostatak ovog algoritma se ogleda u tome što pri uvećanju slike za rezultat se dobija uvećana slika koja nije bistra, poznat naziv (eng *blocky image*). Algoritam koji odlično rešava ovaj problem je BCI (*Bicubic Interpolation*)[19]. BCI algoritam zahteva veliku količinu vremena za rešavanje polinomnijalnih jednačina N-toga reda, gde N predstavlja broj tačaka koje su potrebne da se izračunaju. Tokom pisanja dizajna, napravljeno je par propusta koji su primećeni i otklonjeni projektovanjem verifikacionog okruženja. Projektovanje dobrog verifikacionog okruženja predstavlja veoma važan deo pre slanja čipa na fabrikaciju. Greška u hardveru pravi ogromne novčane gubitke.

Dodatak

[1] Svi potrebni kodovi

<https://files.fm/u/7x92jx2y>

Literatura

- [1] Pong, P. C. (2006). *RTL Hardware Design using VHDL*. Hoboken: Wiley-IEEE Press
- [2] Gordon, A., Baird, M., Rich, E., Erickson, A., Horn, M., Peryer, M., Rose, A., Schwartz, K. (2013). *UVM Cookbook*. Wilsonville: Verification Academy
- [3] Bailey, B., Grant, M., Piziali, A. (2007). *ESL Design and Verification: A Prescription for Electronic System Level Methodology (Systems on Silicon) (1st ed.)*. San Francisco: Morgan Kaufmann
- [4] Asad, M. (2015). *Design of Controllers Finite States Machines and Algorithmic State Machine (ASM) Charts*.
- [5] Stojčev, M. (2007). *FPGA kola*.
- [6] *VHDL*. (2019).
06.02.2019 <<https://en.wikipedia.org/wiki/VHDL>>
- [7] Franz, K. (2015). *Embedded Linux Tutorial – Zybo*.
06.02.2019, <<https://www.instructables.com/id/Embedded-Linux-Tutorial-Zybo>>
- [8] *Zybo Z7: Zynq - 7000 ARM/FPGA SoC Development Board*. (2000).
06.02.2019, <<https://store.digilentinc.com/zybo-z7-zynq-7000-arm-fpga-soc-development-board>>
- [9] Ailus, S., Carvalho, C. M., Dirks, B., Schimek, M., Karicheri, M., Osciak, P., Palosaari, P., Ribalda, R., Rubli, M., Walls, A., Verkuli, H. (2002). *Video for Linux API*.
06.02.2019, <<https://linuxtv.org/downloads/v4l-dvb-apis/uapi/v4l/v4l2.html>>
- [10] Mauro, C. C. (2009). *Appendix E. Video Grabber example using libv4l*.
06.02.2019, <<https://www.linuxtv.org/downloads/v4l-dvb-apis-old/v4l2grab-example.html>>
- [11] Carvalho, C. M., Dirks, B., Schimek, M., Rubli, M., Walls, A., Verkuli, H. (2009). *Packed RGB formats 2.4 RGB Formats*.
06.02.2019,
<https://www.linuxtv.org/downloads/legacy/video4linux/API/V4L2_API/spec/re01.html>
- [12] Beal, V. (2019). *HDMI – High-Definition Multimedia Interface*.
06.02.2019, <<https://www.webopedia.com/TERM/H/HDMI.html>>
- [13] Friesenhanh, B. (2003). *ImageMagick Magick++ API*.
06.02.2019, <<https://imagemagick.org/Magick++>>

- [15] Xilinx. (2011). *AXI Reference Guide (UG761 v13.1)*. San Jose: Xilinx
- [16] *Universal Verification Metodology*. (2018).
06.02.2019, <https://en.wikipedia.org/wiki/Universal_Verification_Methodology>
- [17] *Open Verification Methodology*. (2017).
06.02.2019, <https://en.wikipedia.org/wiki/Open_Verification_Methodology>
- [18] *e Reuse Methodology*. (2018).
06.02.2019, <https://en.wikipedia.org/wiki/E_Reuse_Methodology>
- [18] Rouse, M. (2005). *Direct Memory Access (DMA)*.
06.02.2019, <<https://whatistechtarget.com/definition/Direct-Memory-Access-DMA>>
- [19] *Bicubic Interpolation*. (2017).
06.02.2019, <https://en.wikipedia.org/wiki/Bicubic_interpolation>