Univerza v Ljubljani
Fakulteta za elektrotehniko

Aleš Voda

Digitalni sistem za obdelavo video slike z morfološkimi operacijami

Magistrsko delo

Mentor: izr. prof. dr. Andrej Trost

Ljubljana, 2016

Zahvala

V prvi vrsti bi se zahvalil svojemu mentorju izr. prof. dr. Andreju Trostu, ki je z veseljem sprejel predlagano temo. Zahvalil bi se mu za vso izkazano pomoč, komentarje, priporočila in smernice pri izdelavi magistrskega dela.

Zahvalil bi se tudi podjetju Emsiso, d.o.o., iz Maribora za pomoč pri spajkanju ključnih SMD komponent na tiskano vezje, kakor tudi mojemu dobremu prijatelju Mateju Vesenjaku in njegovemu očetu Antonu Vesenjaku za pomoč pri izdelavi dodatnega tiskanega vezja.

Posebej bi se zahvalil mami, ki mi je ves čas študija stala ob strani in me vzpodbujala tudi v trenutkih, ko mi je bilo najtežje. Vsekakor ne smem izpustiti vseh svojih prijateljev, ki so me prav tako bodrili in vlivali upanje.

Vsebina

1 Uvod....5
1.1 Predstavitev problema....5
1.2 Struktura digitalnega sistema....6
2 Osnove morfoloških filtrov....7
2.1 Binarni morfološki filtri....8
2.1.1 Osnovne morfološke operacije....8
2.1.1.1 Erozija (angl. erosion)....8
2.1.1.2 Razteg (angl. dilation)....9
2.1.2 Sestavljene morfološke operacije....10
2.1.2.1 Odpiranje (angl. opening)....10
2.1.2.2 Zapiranje (angl. closing)....12
2.1.2.3 Gradient....12
2.1.3 Lastnosti....14
2.2 Uporaba nelinearnih morfoloških operacij nad sivinsko sliko....14
2.2.1 Erozija....14
2.2.2 Razteg....15
2.2.3 Odpiranje....15
2.2.4 Zapiranje....15
2.2.5 Gradient....16
3 Načrtovanje in izdelava strojne opreme....17
3.1 Določitev osnovnih funkcij strojne opreme....17
3.2 Izbira komponent....18
3.3 Izdelava električne sheme....19
3.4 Podrobnejši opis glavnih sklopov digitalnega sistema....20
3.4.1 Napajalni del....21
3.4.2 CMOS digitalni slikovni senzor....24
3.4.3 Programirljivo vezje FPGA (Xilinx XC6SLX45)....33
3.4.3.1 Enota za upravljanje ure (CMT)....34
3.4.3.2 Konfigurabilni logični bloki (CLB)....35
3.4.3.3 Vhodno-izhodni bloki (IOB)....36
3.4.3.4 Matrika povezav....36

3.4.3.5 Pomnilniški krmilni blok (MCB)....37
3.4.3.6 Notranji pomnilniki....37
3.4.4 Zunanji pomnilnik....38
3.4.5 Asinhroni serijski komunikacijski vmesnik....39
3.4.6 Izhodni digitalno-analogni pretvornik....39
3.5 Določitev velikosti in števila slojev tiskanega vezja....42
3.6 Razporeditev in povezovanje komponent na tiskanem vezju....42
3.7 Izdelava tiskanine....45
3.8 Opremljanje tiskanega vezja s komponentami....46
3.9 Verifikacije ustreznega delovanja strojne opreme....47
3.10 Modifikacije strojne opreme....48
4 Implementacija digitalnega sistema v programirljivem vezju FPGA....49
4.1 Vhodna komponenta za zajem slike iz CMOS senzorja....50
4.2 RGB Bayerjev filter....51
4.3 Pretvorba slike v RGB formatu v sivinsko sliko....53
4.4 Komunikacija z zunanjim pomnilnikom....54
4.5 Sinhronizacija pisanja in branja video vsebine iz zunanjega pomnilnika....55
4.6 VGA krmilna enota....59
4.7 Implementacija morfološkega filtra....61
4.7.1 Prepust....64
4.7.2 Mediana....64
4.7.3 Razteg....65
4.7.4 Erozija....65
4.7.5 Implementacija morfološkega filtra v visokonivojskem strojno opisnem jeziku VHDL....65
4.7.5.1 Dvovhodni 8-bitni sinhroni primerjalnik....66
4.7.5.2 Osemvhodni urejevalnik z združevanjem 8-bitnih elementov lihih in sodih indeksov....67
4.7.5.3 Devetvhodni urejevalnik 8-bitnih elementov....68
4.7.5.4 Devetvhodni urejevalnik 8-bitnih elementov z uporabo morfološkega strukturnega elementa....70
4.7.5.5 Morfološki filter s cevovodno strukturo....74
5 Rezultati....78
6 Sklep....83

7 Literatura.....84
8 Dodatek.....88
8.1 Izvorna koda komponente morfološkega filtra napisana v visokonivojskem strojno opisnem jeziku VHDL.....88

Seznam slik

Slika 1.1: Poenostavljena blokovna shema digitalnega sistema.....6
Slika 2.1: Strukturni element 3x3 v obliki križa.....8
Slika 2.2: Strukturni element 3x3 v obliki kvadrata.....8
Slika 2.3: Morfološka operacija erozije binarne slike.....9
Slika 2.4: Morfološka operacija raztega binarne slike.....10
Slika 2.5: Morfološka operacija odpiranja binarne slike.....11
Slika 2.6: Morfološka operacija zapiranja binarne slike.....12
Slika 2.7: Morfološka operacija osnovnega gradienta binarne slike.....13
Slika 3.1: Podrobnejša blokovna shema digitalnega sistema.....20
Slika 3.2: Električna shema napajalnikov digitalnega sistema.....22
Slika 3.3: Povezava napajalnikov na tiskanem vezju ter prikaz delitve napajalnih površin na notranjem sloju.....23
Slika 3.4: Električna shema povezav CMOS digitalnega slikovnega senzorja MT9E001 s programirljivim vezjem FPGA.....25
Slika 3.5: Povezava CMOS digitalnega slikovnega senzorja MT9E001 s programirljivim vezjem FPGA na tiskanem vezju.....26
Slika 3.6: Električna shema dodatne tiskanine s CMOS digitalnim slikovnim senzorjem MT9P031.....26
Slika 3.7: Dodatna tiskanina s CMOS digitalnim slikovnim senzorjem MT9P031.....27
Slika 3.8: Časovni potek podatkovnih in statusnih signalov sinhronega paralelnega vodila CMOS digitalnega slikovnega senzorja [25].....27
Slika 3.9: Aktivno območje CMOS digitalnega slikovnega senzorja [25].....28
Slika 3.10: Razporeditev slikovnih elementov znotraj aktivnega območja CMOS digitalnega slikovnega senzorja v Bayerjev vzorec [25].....28
Slika 3.11: Centralni element zelene barve RGB Bayerjevega vzorca za liho vrstico.....30
Slika 3.12: Centralni element rdeče barve RGB Bayerjevega vzorca za lihe vrstice.....30
Slika 3.13: Centralni element modre barve RGB Bayerjevega vzorca za sode vrstice.....31
Slika 3.14: Centralni element zelene barve RGB Bayerjevega vzorca za sode vrstice.....31
Slika 3.15: Časovni diagram pisanja vrednosti 0x0284 v register z naslovom 0x09 CMOS slikovnega senzorja preko I2C vodila [25].....32
Slika 3.16: Časovni diagram branja vrednosti 0x0284 iz registra z naslovom 0x09 CMOS slikovnega senzorja preko I2C vodila [25].....33

Slika 3.17: Vrstična in stolpična odvisnost med CLB in SLICE [9].....35
Slika 3.18: Povezovalni kanali blokov CLB [9].....37
Slika 3.19: Električna shema digitalno-analognega pretvornika in povezava na programirljivo vezje FPGA.....40
Slika 3.20: Zgornji signalni sloj tiskanega vezja (1. sloj).....43
Slika 3.21: Notranji napajalni sloj tiskanega vezja (2. sloj).....44
Slika 3.22: Notranji sloj mase (3. sloj).....44
Slika 3.23: Spodnji signalni sloj tiskanega vezja (4. sloj).....45
Slika 3.24: Zgornja stran neopremljenega izdelanega tiskanega vezja.....46
Slika 3.25: Spodnja stran neopremljenega izdelanega tiskanega vezja.....46
Slika 4.1: Poenostavljena zgradba RGB Bayerjevega filtra znotraj programirljivega vezja FPGA.....53
Slika 4.2: Poenostavljen diagram prehajanja stanj avtomata komponente buf_sched.....58
Slika 4.3: Časovni potek uporabe medpomnilnikov komponente buf_sched za sinhronizacijo branja in pisanja video vsebine v zunanji pomnilnik.....58
Slika 4.4: Časovni potek vertikalnega sinhronizacijskega signala VGA krmilnika.....60
Slika 4.5: Časovni potek horizontalnega sinhronizacijskega signala VGA krmilnika.....60
Slika 4.6: Poenostavljana zgradba morfološkega filtra.....61
Slika 4.7: Dvovhodni 8-bitni primerjalnik.....66
Slika 4.8: Osemvhodni urejevalnik 8-bitnih elementov.....68
Slika 4.9: Batcherjev algoritem urejanja na primeru osmih elementov z združevanjem elementov lihih in sodih (angl. odd-even merge sort) [23].....68
Slika 4.10: Prikaz 6-stopenjske cevovodne strukture in mreže povezav primerjalnikov za urejanje z združevanjem elementov lihih in sodih indeksov.....68
Slika 4.11: Devetvhodni urejevalnik 8-bitnih elementov.....69
Slika 4.12: Devetvhodni urejevalnik z uporabo morfološkega strukturnega elementa.....69
Slika 4.13: Prikaz 13-stopenjske cevovodne strukture devetvhodnega urejevalnika 8-bitnih elementov.....70
Slika 4.14: Morfološki filter s cevovodno strukturo.....75
Slika 5.1: Časovni potek operacije notranjega gradienta morfološkega filtra.....80
Slika 5.2: Vhodna testna sivinska slika velikosti 5x5 slikovnih elementov.....81
Slika 5.3: Erozija vhodne testne sivinske slike.....81
Slika 5.4: Notranji gradient vhodne sivinske slike.....81
Slika 5.5: Prikaz sivinske slike, zajete z izdelanim digitalnim sistemom.....81

Slika 5.6: Prikaz obdelane sivinske slike z morfološkim operatorjem osnovnega gradienta z izdelanim digitalnim sistemom.....82
Slika 5.7: Izdelan digitalni sistem za obdelavo video slike z morfološkimi operacijami.....82

Seznam tabel

Tabela 3.1: Izhodna napetost napajalnika v odvisnosti od stanja krmilnih vhodov [26].....21
Tabela 3.2: Družina programirljivih vezij FPGA Spartan-6 [9].....34
Tabela 4.1: Prikaz izbire operacij morfološkega filtra.....76
Tabela 4.2: Izbira naprednih funkcij morfološkega filtra.....76
Tabela 4.3: Zamenjava izhodov douta in doutb morfološkega filtra.....76
Tabela 4.4: Zasedenost programirljivega vezja XC6SLX45 po sintezi samo ene komponente morfološkega filtra.....77
Tabela 5.1: Zasedenost programirljivega vezja XC6SLX45 po sintezi digitalnega sistema…78

Seznam uporabljenih simbolov

FPGA	polje programirljivih logičnih vrat (angl. Field Programmable Gate Array)
VHDL	visokonivojski strojno opisni jezik, namenjen opisovanju obnašanja in strukture digitalnih vezij (angl. Hardware Description Language)
JTAG	protokol, ki se uporablja za nalaganje konfigurcije oziroma programa v programirljiva vezja (angl. Joint Test Action Group)
DFF	sinhroni zakasnilni element (angl. Delay Flip-Flop)
LUT	vpogledna tabela (angl. Look Up Table)
SLICE	gradnik programirljivega vezja FPGA
CLB	konfigurabilni logični blok – gradnik znotraj programirljivega vezja (angl. Configurable Logic Block)
IOB	vhodno izhodni blok – gradnik programirljivega vezja FPGA (angl. Input/Output Block)
CMT	enota za upravljanje ure v programirljivem vezju FPGA
DCM	digitalni upravljalnik ure (angl. Digital Clock Manager)
MCB	strojna krmilna enota znotraj programirljivega vezja FPGA, namenjena komunikaciji z zunanjim pomnilnikom (angl. Memory Controller Block)
MPMC	večvhodni krmilnik pomnilnika (angl. Multiport Memory Controller)
IP	intelektualna lastnina (angl. Intelektual Property)
FIFO	oznaka za tip medpomnilnika prvi not, prvi ven (angl. First In-First Out)
USB	univerzalno serijsko vodilo (angl. Universal Serial Bus)
RS-232	standard za asinhrono serijsko komunikacijo
I2C	sinhrono serijsko vodilo za povezavo ingeriranih vezij med seboj (angl. Inter Integrated Circuit)
VGA	video grafično polje (angl. Video Graphics Array)

RGB	trojica osnovnih barvnih komponent: rdeča, zelena in modra (angl. Red, Green, Blue)
PIXCLK	ura slikovnega elementa (angl. Pixel Clock)
PLL	fazno sklenjena zanka (angl. Phase Locked Loop)
THT	tehnologije pritrditve elektronskih komponent na tiskano vezje skozi luknje (angl. Through Hole Type)
SMT	tehnologija za površinsko pritrjevanje komponent (angl. Surface-Mount Technology)
SMD	elektronske komponente, namenjene za površinsko pritrjevanje na tiskano vezje (angl. Surface-Mount Device)
DRC	preverjanje napak v dizajnu (angl. Design Rule Check)
LED	polprevodniška svetleča dioda (angl. Light Emitting Diode)
CMOS	tehnologija izdelave digitalnih integriranih vezij z uporabo komplementarnih kovina-oksid-polprevodnikov (angl. Complementary Metal-Oxide-Semiconductor)
SDRAM	sinhroni dinamični pomnilnik z naključnim dostopom (angl. Synchronous Dynamic Random Access Memory)
RAM	pomnilnik z naključnim dostopom (angl. Random Access Memory)
DDR3	oznaka tipa SDRAM pomnilnika z dvojno podatkovno hitrostjo (angl. Double Data Rate 3)
D/A	digitalno-analogni pretvornik
LAN	lokalno omrežje (angl. Local Area Netowrk)

Seznam uporabljenih enot

Ime	**Simbol**
bit	**b**
bajt	**B**
sekunda	**s**
hertz	**Hz**
ohm	**Ω**
volt	**V**
amper	**A**

Povzetek

V magistrskem delu je predstavljen razvoj digitalnega sistema, ki omogoča obdelavo video slike z morfološkimi operacijami. V začetnem delu je predstavljena zasnova digitalnega sistema v smislu osnovnih funkcij, ki so potrebne za izvedbo. Sledi opis osnov morfoloških operacij. Opisani sta osnovni morfološki operaciji raztega in erozije nad binarno sliko, kakor tudi sestavljene morfološke operacije odpiranja, zapiranja in gradienta s pripadajočimi lastnostmi morfoloških operacij. V nadaljevanju je predstavljena razširitev uporabe morfoloških operacij na sivinskih slikah.

V poglavju, ki sledi, je opisan razvoj strojne opreme digitalnega sistema, osnovne funkcije, izbira komponent, načrtovanje tiskanega vezja in spajkanje SMD elementov ter glavni gradniki. Prikazani so deli električnih shem in povezav pomembnejših gradnikov digitalnega sistema na tiskanem vezju. Podrobneje so opisani glavni gradniki digitalnega sistema, in sicer CMOS digitalni slikovni senzor, napajalniki, programirljivo vezje FPGA, zunanji pomnilnik, asinhroni serijski komunikacijski vmesnik, analogno-digitalni pretvornik.

Strojnemu načrtovanju sledi opis izvedbe potrebnih gradnikov znotraj programirljivega vezja FPGA. Za nekatere izmed komponent, implementiranih znotraj programirljivega vezja FPGA, je podana tudi izvorna koda, napisana v visokonivojskem strojno opisnem jeziku VHDL, kakor tudi pripadajoči rezultati simulacij.

Ključne besede: digitalni sistem, morfološke operacije, erozija, razteg, gradient, sivinska slika, obdelava video slike, CMOS digitalni slikovni senzor, načrtovanje strojne oprema, tiskano vezje, spajkanje SMD elementov, FPGA, VHDL, zunanji pomnilnik

Abstract

This master thesis describes development of digital system for video image processing using morphological operations. The first part describes initial digital system with respect to main functionality desired. Basic principles of binary morphological operation such as erosion and dilation including some combined operations like opening, closing and gradient together with their main properties are explained. Afterwards extension of morphological operations on grayscale video images is presented as well.

In the following chapter development of hardware is described in terms of basic functionalities, printed circuit board design and SMD soldering of main components. Some partial electrical and routing schemes are presented for main components of digital system. Key components of digital system such as CMOS digital image sensor, power supplies, FPGA, external memory, asynchronous serial interface, analog-digital converter are described in more detail.

The part following hardware design is software design of components implemented inside FPGA. Along description of main components imeplemented inside FPGA there is also some part of the source code written in VHDL language available together with the simulation results.

Keywords: digital system, morphological operations, erosion, dilation, gradient, opening, closing, CMOS digital image sensor, grayscale image, video image processing, hardware design, printed circuit board, soldering SMD components, FPGA, VHDL, external memory

1 Uvod

1.1 Predstavitev problema

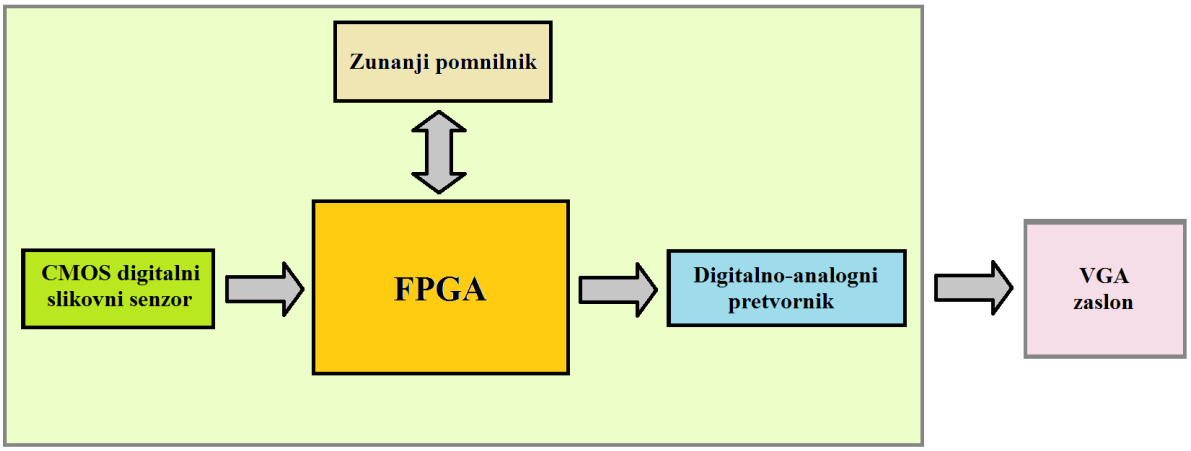
Cilj naloge je bil zasnovati digitalni sistem, ki bo nudil možnost obdelave video slik z morfološkimi operacijami. Potrebujemo izvor video signala, ki ga nato obdelamo z morfološkimi operacijami erozije, raztega, gradienta, odpiranja, zapiranja ali kombinacije le-teh. Rezultat obdelave je potrebno prikazati kot izhodni video signal. Osnovne probleme, s katerimi se je potrebno soočiti, lahko v grobem razdelimo na sledeče sklope:

- načrtovanje in izdelava strojne opreme
- izbira izvora video signala
- shranjevanje zajetega videa
- obdelava videa z morfološkimi operacijami
- prikaz obdelane video vsebine

Sistem mora delovati z dovolj visoko hitrostjo, ki bo omogočala zajemanje, obdelavo in prikaz obdelanega videa v realnem času. V nadaljevanju je predstavljena izdelava digitalnega sistema za obdelavo sivinskih slik oziroma videa z morfološkimi operacijami.

1.2 Struktura digitalnega sistema

Za izdelavo digitalnega sistema, ki bo kos zadani nalogi, sem kot osnovni gradnik izbral programirljivo vezje FPGA, ki zaradi svoje notranje zgradbe nudi veliko možnosti v smislu paralelnega izvajanja operacij, s čimer lažje dosežemo zahtevo po obdelavi videa v realnem času v primerjavi s splošno namenskim procesorjem. Za izvor videa sem uporabil CMOS digitalni slikovni senzor. Sistemska ura programirljivega vezja FPGA mora biti dovolj visoka, da lahko podatke iz CMOS digitalnega slikovnega senzorja nemoteno beremo ter sočasno izvajamo morfološke operacije in prikazujemo obdelan video v realnem času. Če bi na primer za implementacijo sistema izbral splošno namenski procesor, bi zahteve po obdelavi slike z morfološkimi operacijami v realnem času lahko izvršil le ob uporabi precej višje sistemske ure procesorja v primerjavi s programirljivim vezjem FPGA. Nižja frekvenca sistemske ure je neposredno povezana z nižjo porabo celotnega digitalnega sistema. Kot prikazuje poenostavljena blokovna shema (slika 1.1), imamo na vhodu CMOS digitalni slikovni senzor za generiranje izvornega videa. Video signal je sestavljen iz niza zajetih slik CMOS digitalnega slikovnega senzorja. Za video signal je značilno, da če slike zajemamo s frekvenco večjo ali enako 25 Hz, človeško oko zaradi svoje vztrajnosti sestavi niz slik v povezano celoto. S programirljivim vezjem FPGA beremo sliko in jo obdelamo z morfološkim filtrom. Rezultat shranimo v zunanji pomnilnik in ga nato prikažemo na VGA zaslonu. To je le poenostavljen opis delovanja digitalnega sistema, katerega podrobnejša struktura je opisana v nadaljevanju.



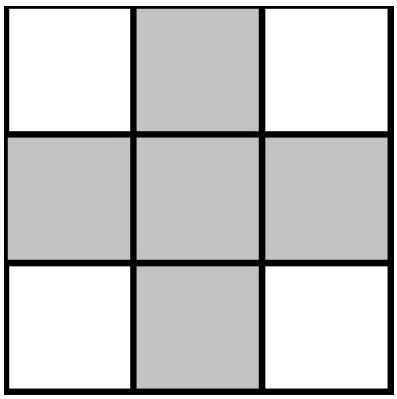
Slika 1.1: Poenostavljena blokovna shema digitalnega sistema

2 Osnove morfoloških filtrov

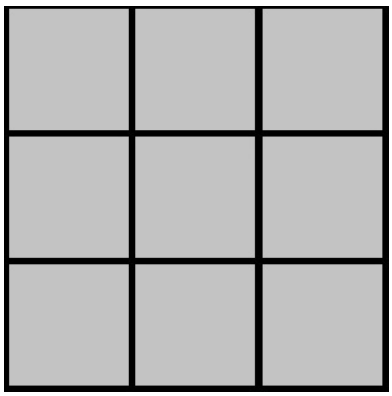
Matematična morfologija se je razvila v šestdesetih letih prejšnjega stoletja. Njena uporaba je bila sprva omejena le na binarne slike. Informacija o vrednosti posameznega slikovnega elementa na binarni sliki je predstavljena z enim samim bitom, pri čemer z logično ničlo predstavimo ozadje binarne slike, medtem ko z logično enico predstavimo objekte. Kasneje se je njena uporaba razširila še na sivinske slike, pri katerih lahko posamezni slikovni element zavzame eno izmed 256 vrednosti, saj je informacija o vrednosti slikovnega elementa pri sivinskih slikah predstavljena z osmimi biti [1] [2].

Morfološke filtre uvrščamo v skupino nelinearnih filtrov. Najosnovnejši morfološki operaciji sta erozija in raztezanje. Osnovo morfološkega filtra predstavlja strukturni element oziroma okno, ki ga postopoma pomikamo preko celotne binarne slike. Za vsak slikovni element izvršimo želeno morfološko operacijo, pri čemer dobimo novo vrednost slikovnega elementa filtrirane slike. Ko postopek pomikanja strukturnega elementa preko celotne slike zaključimo, dobimo kot rezultat morfološke operacije za vsak vhodni slikovni element binarne slike pripadajoče vrednosti izhodnih slikovnih elementov, ki predstavljajo filtrirano sliko. Z uporabo strukturnih elementov različnih velikosti in oblik ter uporabo različnih morfoloških operacij oziroma kombinacije le-teh, lahko iz slike pridobimo informacije na primer o velikosti, obliki, povezanosti in orientaciji objektov ter iz slike odstranimo informacije, ki niso predmet obravnave [3].

Na slikah 2.1 in 2.2 sta prikazana dva najpogosteje uporabljena strukturna elementa morfoloških operacij velikosti 3x3 slikovnih elementov. Prvi strukturni element ima obliko križa in ga uporabljamo na primer, kadar želimo iz binarne slike pridobiti informacije o štiritočkovno povezanih objektih na sliki. Drugi strukturni element v obliki kvadrata je namenjen operacijam, kjer nas zanimajo osemtočkovno povezani objekti na sliki.



Slika 2.1: Strukturni element 3x3 v obliki križa



Slika 2.2: Strukturni element 3x3 v obliki kvadrata

2.1 Binarni morfološki filtri

Kot samo ime pove, so binarni morfološki filtri namenjeni obdelavi binarnih slik. Obliko strukturnega elementa binarnega morfološkega filtra določajo slikovni elementi znotraj okna poljubne velikosti, katerih vrednosti zavzamejo vrednost logične enice. Strukturni elementi morfoloških filtrov so večinoma simetrične oblike, v splošnem pa so lahko tudi nesimetrični. V strukturnem elementu imamo centralni element, za katerega računamo izhodno vrednost morfološkega filtra. Centralni element strukturnega elementa položimo na vsakega izmed slikovnih elementov vhodne binarne slike in tako izračunamo pripadajoče izhodne vrednosti slikovnih elementov filtra za celotno binarno sliko. Vrednost izhodnega slikovnega elementa za dani vhodni slikovni element izračunamo glede na morfološko operacijo, ki jo izvajamo nad binarno sliko.

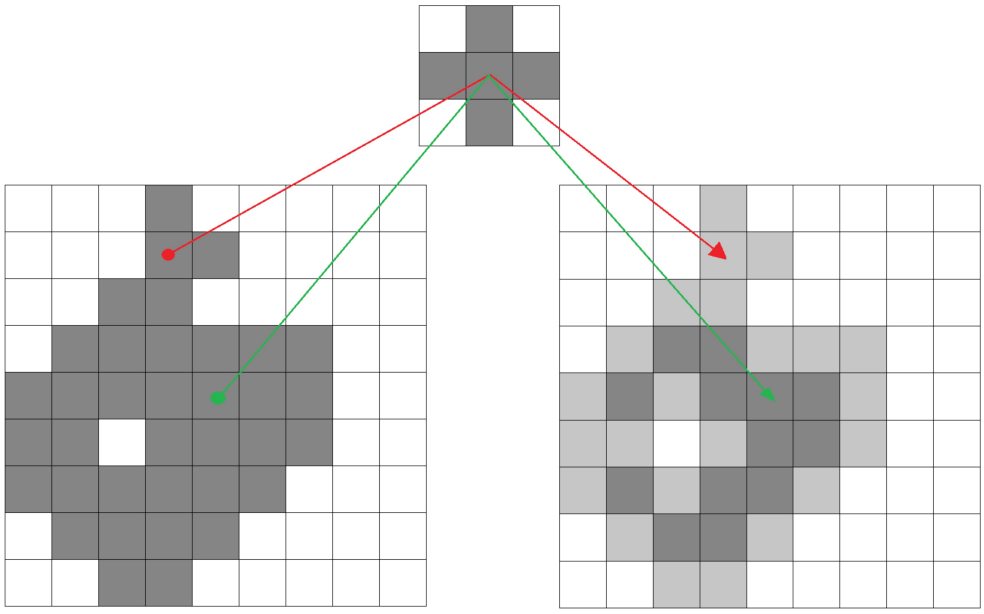
2.1.1 Osnovne morfološke operacije

2.1.1.1 Erozija (angl. erosion)

Z morfološko operacijo erozije objekte na binarni sliki skrčimo. V odvisnosti od oblike in velikosti strukturnega elementa lahko iz binarne slike odstranimo za obravnavo nepomembne informacije (razdruževanje predhodno povezanih objektov na binarni sliki, povečevanje lukenj v objektu). Vrednost centralnega slikovnega elementa je določena kot presek množice elementov B, ki tvorijo obliko strukturnega elementa in podmnožice slikovnih elementov A binarne slike, ki ga strukturni element prekrije. Če je presek med množicama polna množica, je vrednost izhodnega slikovnega elementa za trenutni centralni

slikovni element enaka ena, sicer je vrednost enaka nič. Izhodni slikovni element je enak ena torej samo v primeru, ko strukturni element za opazovani slikovni element binarne slike v celoti prekrije objekt na sliki, sicer je enak nič. Matematično opišemo morfološko operacijo erozije kot presek med množico strukturnega elementa B in podmnožico binarne slike A (2.1), ki ga za opazovani slikovni element strukturni element v celoti prekriva.

$$A \ominus B = \{ \; z \; \in R^2 \; | \; B_Z \; \subseteq A \; \} \tag{2.1}$$



Slika 2.3: Morfološka operacija erozije binarne slike

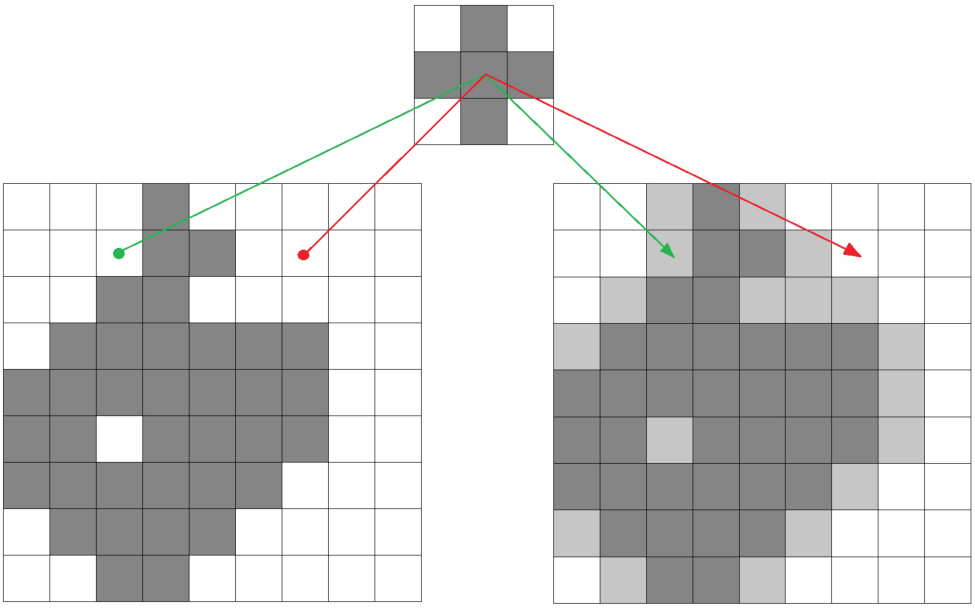
Slika 2.3 prikazuje primer morfološke operacije erozije nad binarno sliko z uporabljenim strukturnim elementom velikosti 3x3 v obliki križa, ki je eden izmed najpogosteje uporabljenih strukturnih elementov. S svetlo sivo barvo so označeni slikovni elementi, ki jih z operacijo erozije, za dani primer strukturnega elementa v obliki križa, iz prvotne slike odstranimo.

2.1.1.2 Razteg (angl. dilation)

Morfološka operacija raztega je morfološki operaciji erozije dualna operacija, ki objekte na binarni sliki raztegne. V binarno sliko tako v odvisnosti od oblike in velikosti strukturnega elementa dodajamo za obravnavo pomembne informacije (povezovanje predhodno

nepovezanih objektov na binarni sliki, daljšanje krakov objekta, zmanjševanje lukenj v objektu). Vrednost slikovnih elementov izhodne slike je določena kot presek množice slikovnih elementov B, ki predstavlja obliko strukturnega elementa in podmnožice slikovnih elementov A binarne slike, ki ga strukturni element prekrije (2.2). Če je presek med množicama prazna množica, je vrednost izhodnega slikovnega elementa za trenutni centralni slikovni element enaka nič, sicer je vrednost enaka ena. Izhodni slikovni element je enak nič samo v primeru, ko strukturni element za opazovani vhodni slikovni element binarne slike ne prekrije objekta na sliki niti v enem samem slikovnem elementu.

$$A \oplus B = \{ \; z \; \in R^2 \; | \; A \cap B_z \neq 0 \} \tag{2.2}$$



Slika 2.4: Morfološka operacija raztega binarne slike

Slika 2.4 prikazuje primer morfološke operacije raztega nad binarno sliko z uporabljenim strukturnim elementom velikosti 3x3 slikovnih elementov v obliki križa. S svetlo sivo barvo so na sliki označeni slikovni elementi, ki jih po morfološki operaciji raztega dodamo v prvotno binarno sliko.

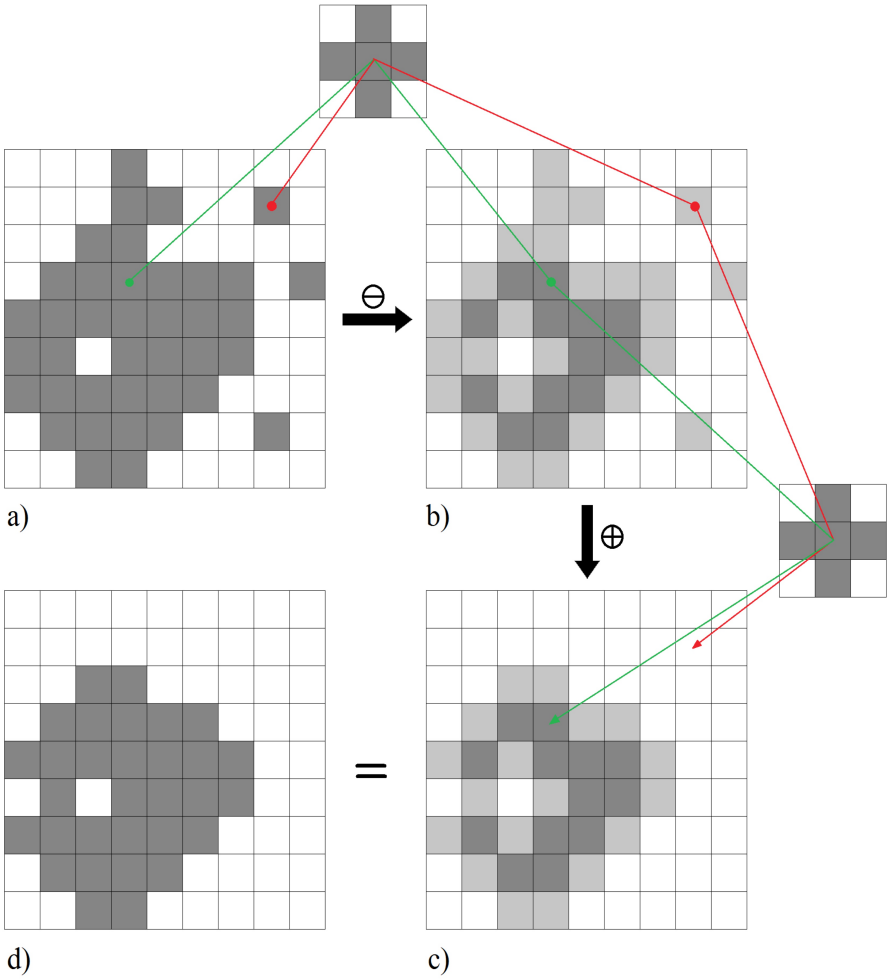
2.1.2 Sestavljene morfološke operacije

2.1.2.1 Odpiranje (angl. opening)

Odpiranje je sestavljena morfološka operacija, kjer binarno sliko najprej erodiramo (izvedemo morfološko operacijo erozije), nato pa nad erodirano sliko izvršimo še morfološko

operacijo raztega. Z morfološko operacijo erozije najprej povečamo luknje v objektu, nato pa z raztegom popravimo oziroma zapolnimo praznine v objektu (2.3). Končni rezultat morfološke transformacije je rahlo erodirana slika, kjer po eroziji z raztegom delno zgladimo oziroma zakrpamo ustvarjene odprtine oziroma skrajšane robove v objektih.

$$A \circ B = (A \ominus B) \oplus B \tag{2.3}$$



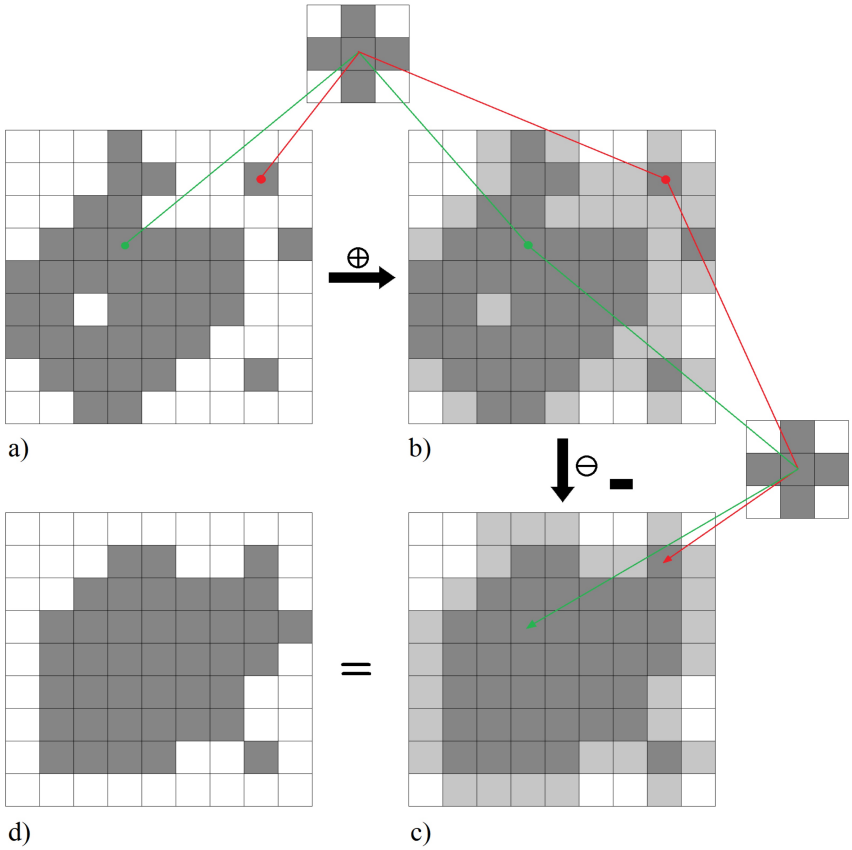
Slika 2.5: Morfološka operacija odpiranja binarne slike

Sliki 2.5b in 2.5c prikazujeta vmesna rezultata morfološke operacije erozije in raztega, kjer so s svetlo sivo barvo označeni slikovni elementi, ki jih z morfološko operacijo erozije odstranimo (slika 2.5b) iz izvorne slike (slika 2.5a), kakor tudi elementi, ki jih dodamo z morfološko operacijo raztega (slika 2.5c) predhodno erodirani sliki (slika 2.5b). Slika 2.5d prikazuje končni rezultat morfološkega odpiranja vhodne binarne slike (slika 2.5a).

2.1.2.2 *Zapiranje (angl. closing)*

Tudi morfološka operacija zapiranja je sestavljena morfološka operacija, kjer pa je v primerjavi z morfološkim odpiranjem vrstni red morfoloških operacij erozije in raztega zamenjan. Najprej z morfološko operacijo raztega zapolnimo luknje v objektih, nato pa z morfološko operacijo erozije zgladimo predhodno zapolnjene luknje oziroma razširjene robove objektov (2.4).

$$A \bullet B = (A \oplus B) \ominus B \tag{2.4}$$



Slika 2.6: Morfološka operacija zapiranja binarne slike

2.1.2.3 *Gradient*

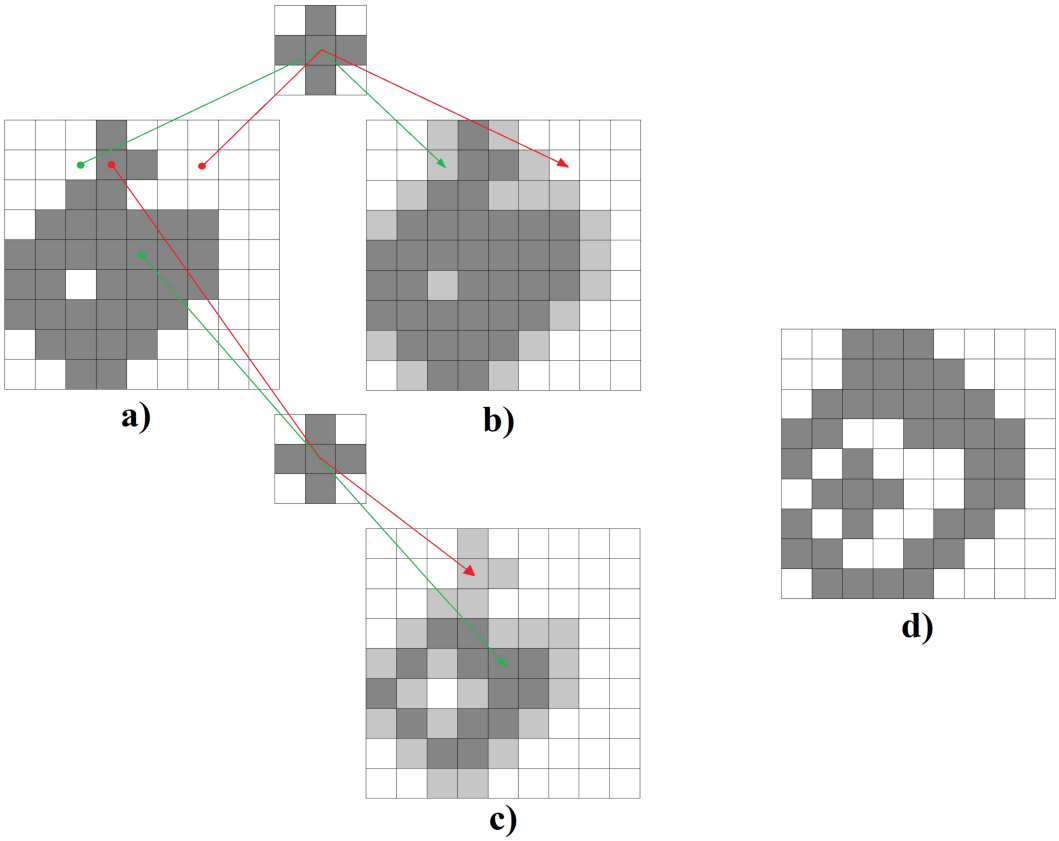
Osnovni morfološki gradient je enak razliki med raztegom in erozijo binarne slike. Ta morfološki filter poudari tako notranje kot zunanje robove. Če nad binarno sliko izvršimo morfološko operacijo erozije in rezultat odštejemo od izvorne binarne slike, dobimo prav tako

gradient, vendar v sliki poudarimo le notranje robove. Kadar pa nad binarno sliko izvršimo morfološko operacijo raztega in od rezultata odštejemo izvorno binarno sliko, dobimo gradient, kjer na sliki poudarimo zunanje robove. Rezultat vseh treh morfoloških operacij gradienta je torej robna slika, razlika pa je le v tem, katere izmed robov na sliki poudarimo: notranje in zunanje (2.5), samo notranje (2.6), samo zunanje (2.7).

osnovni gradient $$\nabla_A = (A \oplus B) - (A \ominus B) \quad (2.5)$$

notranji gradient $$\nabla_A = A - (A \ominus B) \quad (2.6)$$

zunanji gradient $$\nabla_A = (A \oplus B) - A \quad (2.7)$$



Slika 2.7: Morfološka operacija osnovnega gradienta binarne slike

Slika 2.7a prikazuje vhodno binarno sliko, nad katero smo izvedli morfološki operaciji erozije (slika 2.7c) in raztega (slika 2.7b). Na sliki 2.7d vidimo rezultat operacije osnovnega morfološkega gradienta, ki je enak razliki med morfološkima operacijama raztega in erozije vhodne binarne slike. Robovi slike so predvsem zanimivi pri obravnavi lastnosti slike s stališča prepoznavanja objektov na sliki.

2.1.3 Lastnosti

Morfološki operaciji raztega in erozije imata kar nekaj zanimivih lastnosti. Za binarne objekte A, B, C in D veljajo sledeče lastnosti:

Komutativnost: $$A \oplus B = B \oplus A \tag{2.8}$$

Dualnost: $$(A \ominus B)^C = A^C \oplus B \tag{2.9}$$

Asociativnost: $$A \oplus (B \oplus C) = (A \oplus B) \oplus C \tag{2.10}$$

Translacijska invariantnost $$A_Z \oplus B = (A \oplus B)_Z \tag{2.11}$$

Naraščanje: $$A \subseteq B \Rightarrow A \oplus D \subseteq B \oplus D \tag{2.12}$$

Distributivnost: $$(A \cup B) \oplus C = (A \oplus C) \cup (B \oplus C) \tag{2.13}$$

$$A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C) \tag{2.14}$$

2.2 Uporaba nelinearnih morfoloških operacij nad sivinsko sliko

Morfološki filtri so v samem začetku razvoja morfologije služili le za obdelavo binarnih slik, kasneje se je njihova uporaba razširila tudi na sivinske slike. Slikovne elemente sivinske slike v primerjavi z binarnimi slikami predstavimo z osmimi podatkovnimi biti, ki lahko zavzamejo eno izmed vrednosti iz množice [0, 255]. Kot pri binarni sliki, imamo tudi pri sivinski sliki strukturni element določene oblike, ki ga pomikamo preko celotne sivinske slike, da dobimo vrednosti slikovnih elementov glede na operacijo, ki jo nad sliko izvajamo (erozija, razteg). Pri obravnavi sivinskih slik se razširita le osnovni morfološki operaciji raztega in erozije, vse ostale morfološke operacije pa so nespremenjene, saj je njihova osnova morfološka operacija raztega, erozije ali kombinacije obeh.

2.2.1 Erozija

Pri eroziji je vrednost slikovnega elementa, ki sovpada s središčem strukturnega elementa, enaka minimumu izmed vrednosti množice slikovnih elementov, ki jih strukturni element poljubne oblike znotraj okna pokrije. To pomeni, da pri morfološki operaciji erozije sivinske slike iščemo najmanjšo vrednost slikovnih elementov, ki jih za dani središčni slikovni element, strukturni element prekrije. Matematično opišemo morfološko operacijo erozije sivinske slike f v katerikoli točki kot funkcijo iskanja minimuma znotraj strukturnega elementa b s središčem v točki (x, y) z naslednjo enačbo (2.15):

$$[f \ominus b](x, y) = min\, f(x+s, y+t); (s, t) \in b \tag{2.15}$$

2.2.2 Razteg

Operacija raztega slikovnemu elementu, ki sovpada s središčem okna strukturnega elementa, priredi maksimalno vrednost iz množice vrednosti slikovnih elementov, ki jih strukturni element poljubne oblike pokrije. Morfološko operacijo raztega si lahko tako zamislimo kot iskanje lokalnega maksimuma v množici vrednosti slikovnih elementov sivinske slike, ki jih pokrije strukturni element. Matematično opišemo morfološko operacijo raztega sivinske slike f v katerikoli točki (x, y) kot funkcijo iskanja maksimuma znotraj strukturnega elementa b s središčem v točki (x, y) (2.16):

$$[f \oplus b](x, y) = max\, f(x+s, y+t); (s, t) \in b \tag{2.16}$$

2.2.3 Odpiranje

Odpiranje sivinske slike f s strukturnim elementom b je morfološki operator, ki ga sestavljata osnovni morfološki operaciji erozije in raztega sivinske slike. Nad sivinsko sliko najprej izvršimo morfološko operacijo erozije, nato pa nad dobljeno sivinsko sliko izvršimo še morfološko operacijo raztega (2.17). S pomočjo morfološke operacije odpiranja lahko iz sivinske slike izločimo svetli šum.

$$f \circ b = (f \ominus b) \oplus b \tag{2.17}$$

2.2.4 Zapiranje

Zapiranje sivinske slike f s strukturnim elementom je morfološki operator, ki ga sestavljata osnovni morfološki operaciji raztega in erozije. Najprej nad sivinsko sliko f izvršimo morfološko operacijo raztega, nato pa nad dobljeno sivinsko sliko izvršimo še morfološko operacijo erozije (2.18). S pomočjo morfološke operacije zapiranja lahko iz sivinske slike izločimo temni šum.

$$f \bullet b = (f \oplus b) \ominus b \tag{2.18}$$

2.2.5 Gradient

Gradient sivinske slike dobimo tako, da nad sivinsko sliko izvršimo morfološko operacijo raztega. Nad izvorno sivinsko sliko izvršimo tudi morfološko operacijo erozije. Dobljena rezultata med seboj odštejemo, in sicer tako da od sivinske slike, ki smo jo dobili po morfološki operaciji raztega, odštejemo sivinsko sliko, ki smo jo dobili kot rezultat morfološke operacije erozije. Rezultat je osnovni gradient oziroma robna slika izvorne sivinske slike (2.19).

osnovni gradient sivinske slike $$\nabla_f = (f \oplus b) - (f \ominus b) \quad (2.19)$$

notranji gradient sivinske slike $$\nabla_f = f - (f \ominus B) \quad (2.20)$$

zunanji gradient sivinske slike $$\nabla_f = (f \oplus B) - f \quad (2.21)$$

3 Načrtovanje in izdelava strojne opreme

Strojno opremo digitalnega sistema uporabljenega za realizacijo oziroma implementacijo nelinearnih morfoloških operacij sem v celoti načrtal samostojno. Na tržišču je sicer veliko razvojnih platform, a v trenutku načrtovanja nisem uspel najti takšne, ki bi zadostila potrebam in želenim zmožnostim. Poudariti je potrebno tudi to, da so razvojne platforme precej drage, kar je bil eden izmed razlogov za odločitev, da sam izdelam razvojno platformo, na kateri bom razvijal. Načrtovanje strojne opreme digitalnega sistema bi lahko strnil v sledeče korake:

- določitev osnovnih funkcij strojne opreme
- izbira potrebnih komponent
- izdelava električne sheme
- določitev velikosti in števila slojev tiskanega vezja
- razporeditev komponent na tiskanem vezju
- povezovanje komponent na tiskanem vezju
- izdelava tiskanega vezja
- opremljanje tiskanega vezja
- verifikacije ustreznega delovanja strojne opreme
- modifikacije strojne opreme

3.1 Določitev osnovnih funkcij strojne opreme

Osnovne funkcije, katerim je morala zadostiti strojna oprema, so bile zmožnost zajemanja videa, obdelava znotraj programirljivega vezja FPGA, shranjevanje videa v zunanjem pomnilniku, komunikacija z zunanjimi napravami (RS-232) ter prikaz obdelanega videa. Razvojno ploščo sem zasnoval širše v smislu funkcionalnosti, saj sem poleg potrebnih funkcij strojne opreme dodal še dva komunikacijska vmesnika za povezavo z zunanjimi napravami (LAN, USB), stereo avdio kodek ter zunanji trajni pomnilnik NAND. Ker pri implementaciji morfoloških operacij nad video sliko določenih segmentov tiskanega vezja

nisem potreboval, jih nisem opremljal (LAN, USB, NAND, avdio kodek).

Poleg osnovnih komponent sem dodal še osem svetlečih diod in štiri tipke, ki jih v času samega razvoja in kasneje tudi v končni aplikaciji lahko s pridom uporabimo. Svetleče diode nam na primer lahko označujejo trenutni način delovanja sistema ali nas opozarjajo na nepravilnosti v delovanju sistema, tipke pa uporabniku nudijo možnost izbire med različnimi načini delovanja digitalnega sistema.

3.2 Izbira komponent

Glede na zahtevano funkcionalnost strojne opreme sem izbral sledeče glavne komponente digitalnega sistema:

- CMOS digitalni slikovni senzor 8 MP proizvajalca Aptina MT9E001 [24]
- CMOS digitalni slikovni senzor 5 MP proizvajalca Aptina MT9P031 [25]
- Zbirna leča DSL377 proizvajalca Sunex [30]
- zunanji pomnilnik 128 MB proizvajalca Micron MT41J64M16JT-15E:G [28]
- programirljivo vezje FPGA proizvajalca Xilinx XC6SLX45-2FGG484C [9]
- Serijski PROM pomnilnik proizvajalca Xilinx XCF32PF [27]
- DC-DC napajalniki digitalnega sistema proizvajalca Enpirion EN5322QI [26]
- integrirano vezje za prilagoditev CMOS RS-232 napetostnih nivojev MAX232 [29] proizvajalca Texas Instruments za asinhroni serijski vmesnik
- konektor DB-9 za povezavo zunanje naprave na digitalni sistem
- konektor DB-15 za povezavo zaslona na digitalni sistem
- napajalni konektor za priklop digitalnega sistema na zunanji napajalni vir
- 50-pinski razširitveni konektor

3.3 Izdelava električne sheme

Za izdelavo električne sheme sem uporabil razvojno orodje Altium Designer. V prvi fazi sem moral pregledati, ali obstajajo znotraj razvojnega orodja električni simboli in pripadajoča ohišja za izbrane komponente. Za večino komponent digitalnega sistema je bilo potrebno izdelati tako električne simbole elementov, kakor tudi pripadajoča podnožja ohišij. Pri izdelavi električnih simbolov bi izpostavil programirljivo vezje FPGA, ki ima kar 484 priključnih pinov [10] in je bilo zato potrebno samo komponento smiselno razdeliti na manjše podsklope. Tako sem v en električni simbol združil vse napajalne pine, v drugi sklop sem dal vse pine mas. Tretji sklop se nanaša na pine, ki služijo nalaganju programa v programirljiva vezja FPGA s pomočjo JTAG vmesnika in pine za konfiguracijski PROM pomnilnik. Preostale pine komponente sem razdelil na štiri podsklope, od katerih vsak predstavlja eno izmed štirih tako imenovanih bank programirljivega vezja FPGA. Programirljiva vezja nudijo namreč možnost direktnega priklopa posamezne banke na več različnih vhodno-izhodnih standardov (LVTTL, LVCMOS33, LVCMOS25, LVCMOS18, LVCMOS12, PCI33_3/PCI66_3, I2C, SMBus…) [14]. Vse, kar je potrebno, je to, da pripadajoče napajalne pine banke povežemo z ustrezno napetostjo vhodno-izhodnega standarda, ki ga želimo uporabiti za določeno banko. Za načrtovanje električne sheme sem porabil približno dva meseca.

Drugi korak je bila razdelitev celotne električne sheme na podsklope. Električno shemo digitalnega sistema sem tako razdelil na sledeče podsklope:

- napajalni del
- video vhod (CMOS digitalni slikovni senzor) in izhod (D/A pretvornik)
- zunanji pomnilnik (DDR3)
- zunanji trajni pomnilnik (NAND)
- programirljivo vezje FPGA
- vhodno-izhodni del (RS-232, svetleče diode, tipke, razširitveni konektor)
- avdio
- USB

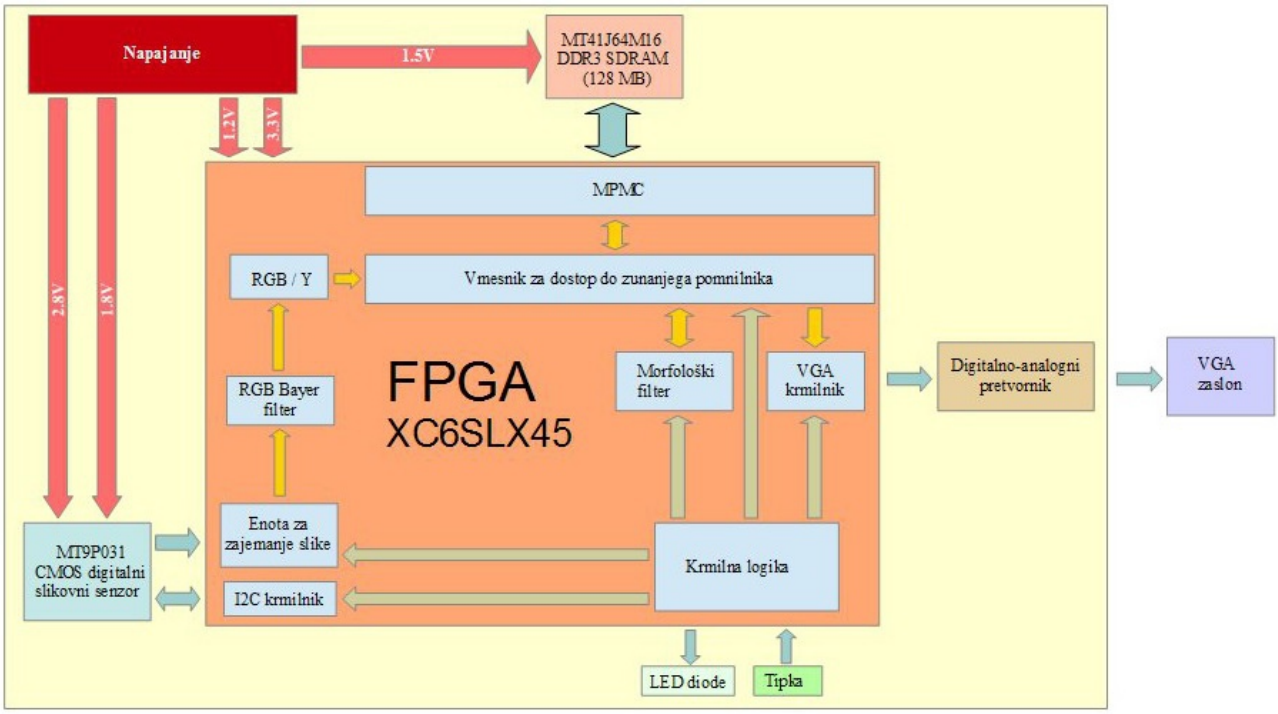
– LAN

Sledilo je nameščanje in povezovanje komponent znotraj podsklopov električne sheme. Omeniti je potrebno, da vse ključne aktivne komponente digitalnega sistema, omenjene v prejšnjem poglavju, potrebujejo za normalno delovanje še pripadajoče pasivne komponente (upori, kondenzatorji…).

Po končanem povezovanju komponent v vseh sklopih električne sheme sem izvršil avtomatsko preverjanje napak v sklopu razvojnega orodja (angl. Design Rule Check - DRC), ki se nanašajo na nepravilne povezave v smislu izhodov in vhodov komponent ter napajalnih in nepovezanih pinov komponent v celotni električni shemi. Če programsko orodje ne zazna napak, lahko v naslednjem koraku kreiramo seznam vseh komponent in njihovih medsebojnih povezav, ki so potrebne v fazi povezovanja komponent na tiskanem vezju.

3.4 Podrobnejši opis glavnih sklopov digitalnega sistema

V nadaljevanju so opisani le sklopi digitalnega sistema, ki sem jih uporabil pri izdelavi naloge.



Slika 3.1: Podrobnejša blokovna shema digitalnega sistema

3.4.1 Napajalni del

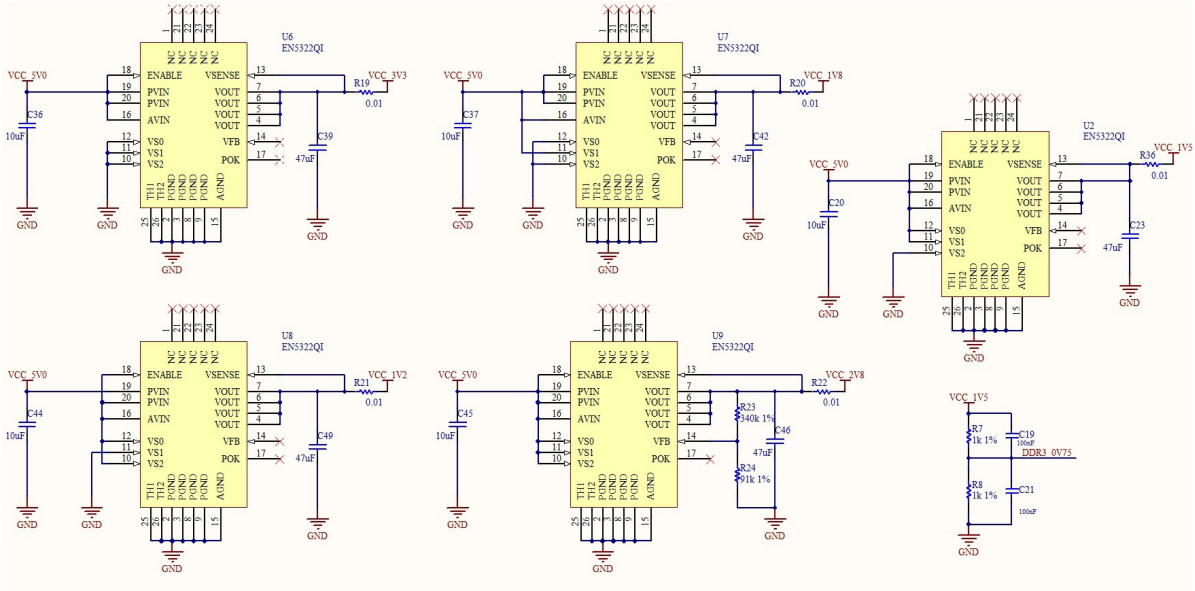
Vhodna napajalna napetost digitalnega sistema je 5 V. Za napajanje posameznih sklopov digitalnega sistema potrebujemo več različnih napetosti, ki sem jih realiziral s pomočjo petih DC-DC sinhronih stikalnih napajalnikov EN5322QI proizvajalca Enpirion [26]. Izhodna tuljava in MOSFET tranzistorji so integrirani že v sam čip, kar precej poenostavi načrtovanje. Napajalni čip EN5322QI deluje s fiksno frekvenco 4 MHz. Izgube pri pretvorbi vhodne enosmerne napetosti v izhodno enosmerno napetost so za ta čip res majhne, saj lahko dosežemo tudi do 95 % učinkovitost delovanja čipa.

Ohišje napajalnega čipa je velikosti 4 mm x 6 mm x 1.1 mm, kar pomeni, da na tiskanem vezju ne zasede veliko prostora. Z uporabo tovrstnih napajalnih čipov sem tako zmanjšal površino dela tiskanine, ki jo zasedejo napajalniki. Za delovanje napajalnega čipa sta potrebna le dva kondenzatorja, in sicer eden na vhodu, drugi pa na izhodu. Napajalnik ima tri krmilne vhode, s pomočjo katerih na enostaven način izberemo eno izmed sedmih izhodnih napetosti, ki jih napajalnik omogoča, kot prikazuje spodnja tabela. Izhodno napetost lahko spreminjamo med samim delovanjem. Če potrebujemo izhodno napetost vrednosti, ki ni podprta z izbiro ene izmed kombinacij na krmilnih vhodih, enostavno povežemo vse tri krmilne vhode VS0, VS1 in VS2 na vhodno napajalno napetost, vrednost izhodne napetosti pa določimo z uporovnim delilnikom na izhodu.

2.4 V <= Vvh <= 5 V, Iizh = 0 ~ 2 A -40°C <= Tokolice <= +85°C			
VS2	VS1	VS0	Uizh
0	0	0	3.3 V
0	0	1	2.5 V
0	1	0	1.8 V
0	1	1	1.5 V
1	0	0	1.25 V
1	0	1	1.2 V
1	1	0	0.8 V
1	1	1	Določi uporabnik

Tabela 3.1: Izhodna napetost napajalnika v odvisnosti od stanja krmilnih vhodov [26]

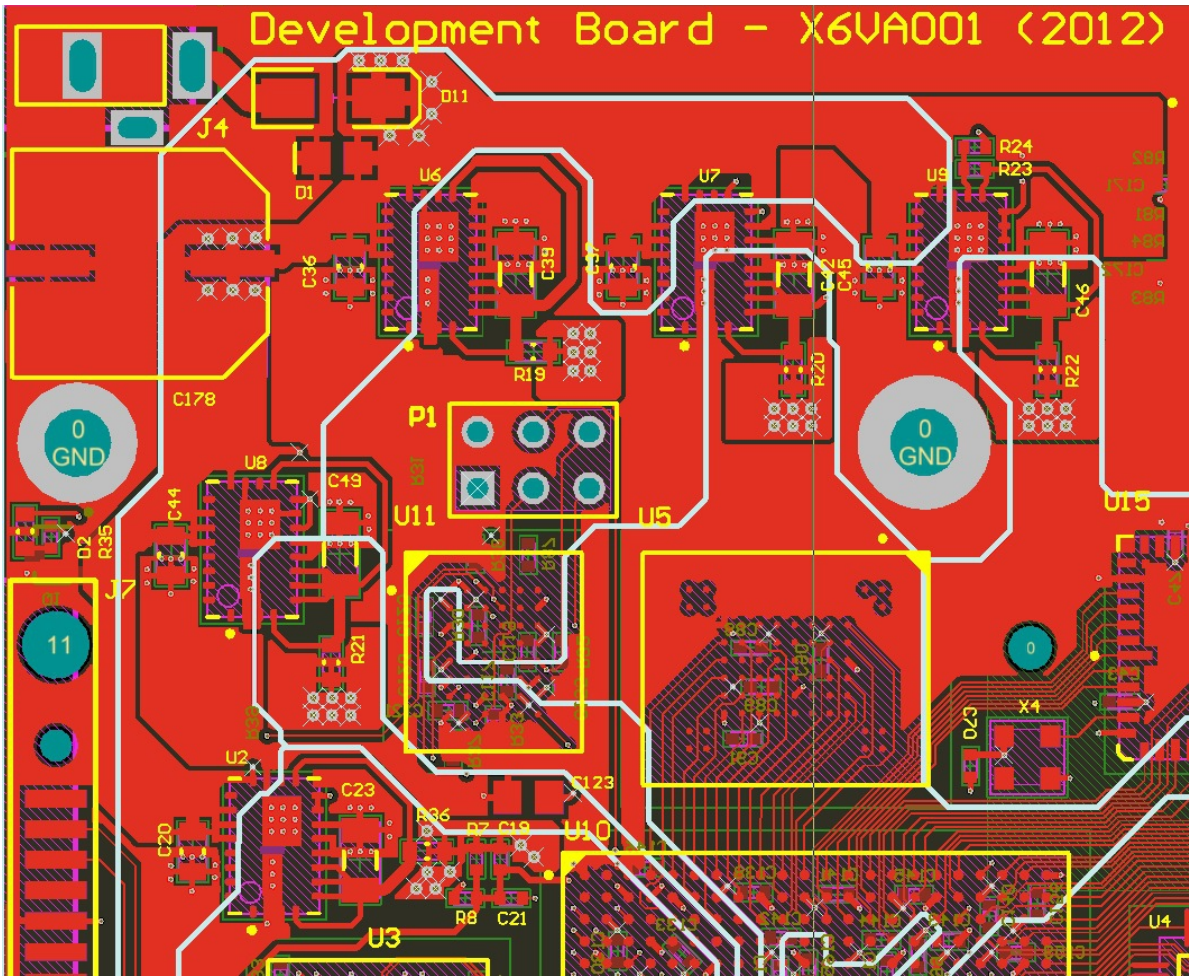
Izhodne napetosti, ki jih napajalnik omogoča brez uporabe uporovnega delilnika na izhodu, so standardne napetosti, ki se največkrat pojavljajo v digitalnih sistemih.



Slika 3.2: Električna shema napajalnikov digitalnega sistema

Kot je razvidno iz napajalnega dela električne sheme (slika 3.2), sem vse potrebne napetosti razen ene generiral z ustrezno kombinacijo vhodnih krmilnih signalov napajalnega čipa VS0, VS1 in VS3. CMOS digitalni slikovni senzor potrebuje za svoj analogni del napajalno napetost 2.8 V, ki sem jo generiral z uporabo uporovnega delilnika na izhodu napajalnega čipa. Vrednost enega izmed uporov je vnaprej določena in znaša 340 kΩ z natančnostjo 1 %, vrednost drugega upora pa za želeno izhodno napetost določimo s pomočjo enačbe (3.1):

$$R = \frac{204}{(V_{IZH} - 0.6)} k\Omega \tag{3.1}$$



Slika 3.3: Povezava napajalnikov na tiskanem vezju ter prikaz delitve napajalnih površin na notranjem sloju

Nekaj najpomembnejših lastnosti napajalnega čipa EN5322QI:

- zelo majhno ohišje QFN, 4 mm x 6 mm x 1.1 mm
- fiksna obratovalna frekvenca, 4 MHz
- visoka učinkovitost, do 95 %
- majhna valovitost izhodne napetosti, tipično 8 mVp-p
- razpon vhodne napetosti od 2.4 V do 5 V
- zmožnost kontinuiranega izhodnega toka je 2 A
- hiter odzivni čas
- tipični tok v stanju izklopa je 17 µA
- zaščita pred prenizko vhodno napetostjo, prevelikim izhodnim tokom, kratkim stikom

ter temperaturna zaščita

- dinamično spreminjanje izhodne napetosti z uporabo krmilnih vhodov

3.4.2 CMOS digitalni slikovni senzor

Tiskano vezje sem prvotno opremil z 1/2.5-palčnim CMOS digitalnim slikovnim senzorjem MT9E001 ločljivosti osem milijonov slikovnih elementov [24]. Ker sem v času razvoja naletel na težave in po več kot dveh mesecih poizkusov nisem uspel vzpostaviti normalnega delovanja senzorja, sem bil primoran poiskati alternativno rešitev. Iz senzorja mi je uspelo dobiti le statusna signala ter uro podatkovnega vodila. Na podlagi teh treh signalov sem lahko določil, da sta nastavljena ločljivost, kakor tudi frekvenca osveževanja bili ustrezni. Problemi so se pojavili pri podatkovnih signalih, ki nikakor niso dali pravih vrednosti, tako da nisem nikoli uspel zajeti slike s pravilno vsebino. Vzroka za nedelovanje nisem uspel odkriti, tako da je težko reči, ali so bile za neustrezno delovanje senzorja krive le nastavitve registrov, slabi spoji senzorja na tiskanini ali pa je bil morda sam senzor celo v okvari.

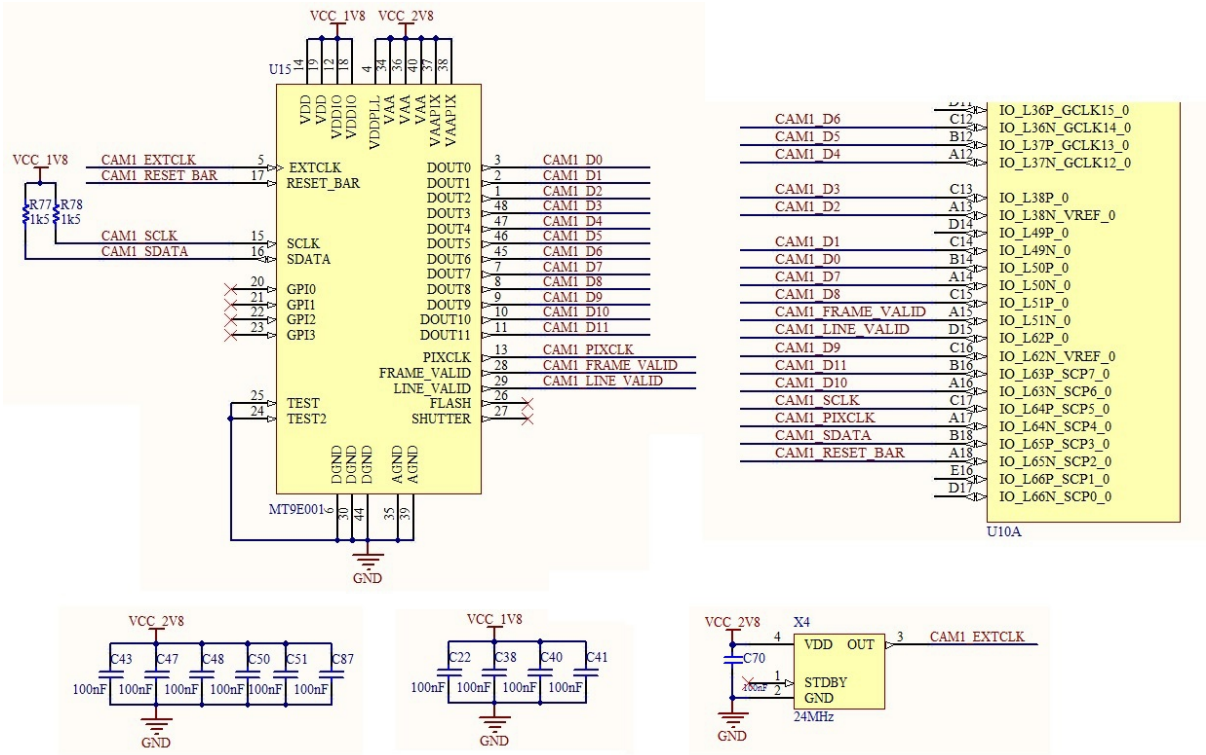
Ker je bilo potrebno poiskati alternativno rešitev, sem se odločil, da izberem drug CMOS digitalni slikovni senzor ter naredim novo dodatno tiskano vezje in ga povežem s programirljivim vezjem FPGA preko 50-pinskega razširitvenega konektorja. Izbral sem CMOS digitalni slikovni senzor proizvajalca Aptina MT9P031 z ločljivostjo pet milijonov slikovnih elementov [25]. Za razvoj dodatnega enoslojnega tiskanega vezja z novim CMOS digitalnim slikovnim senzorjem sem potreboval le dobre tri ure. Ohišje novo izbranega senzorja je bilo namreč popolnoma enako prvotno uporabljenemu senzorju (ILCC-48). Za izdelavo električne sheme sem moral narisati le nove električne simbole senzorja in dveh napajalnikov. Zaradi enostavnejšega povezovanja senzorja s konektorji na enem sloju tiskanega vezja sem se odločil, da uporabim raje dva konektorja in ju nato s posebej v ta namen izdelanim kablom povežem s 50-pinskim razširitvenim konektorjem razvojne plošče. Senzor mi je po izdelavi tiskanega vezja in namestitvi ter priklopu na že obstoječo razvojno ploščo uspelo usposobiti za normalno delovanje le v nekaj urah. Slika 3.6 prikazuje električno shemo, slika 3.7 pa povezave na dodatnem tiskanem vezju.

CMOS digitalni slikovni senzor ima dve vodili, in sicer sinhrono paralelno podatkovno vodilo, preko katerega se prenaša video vsebina, ter sinhrono serijsko komunikacijsko vodilo I2C, preko katerega nastavljamo vrednosti registrov in s tem način delovanja senzorja. Poleg

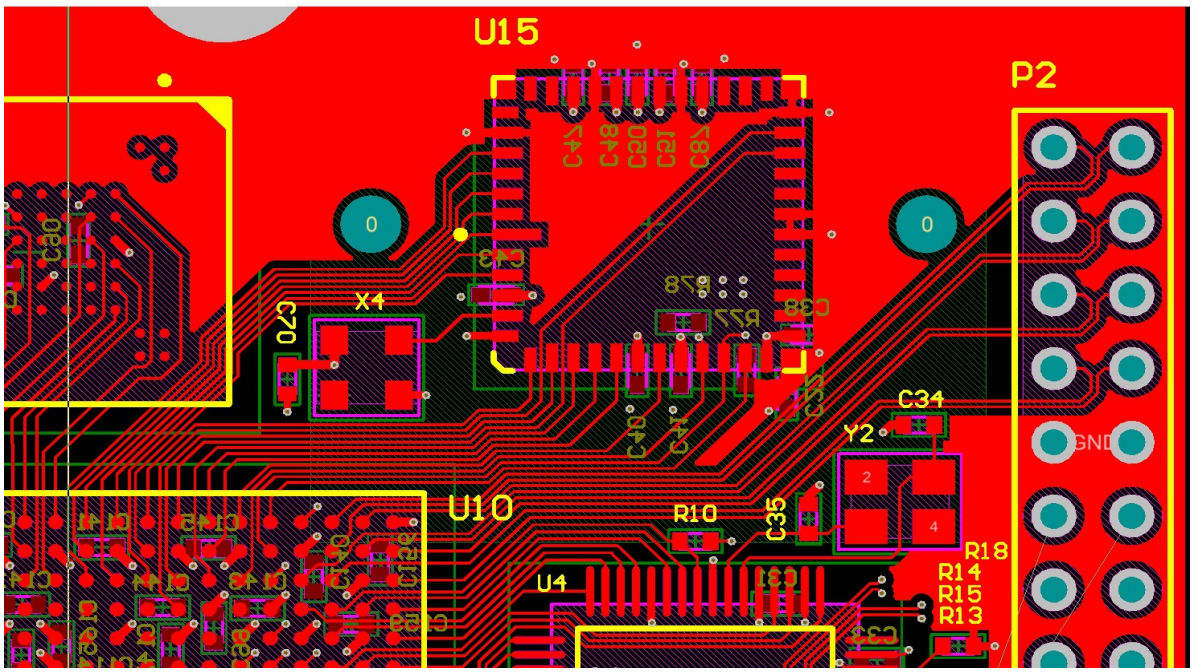
tega ima senzor še vhod sistemske ure, krmilne signale, izhod ure, na katerega je sinhronizirano podatkovno vodilo, ter statusna signala LV in FV, ki določata veljavnost podatkov na podatkovnem vodilu. Podatki so veljavni le, ko sta oba signala hkrati aktivna, torej v stanju logične enice, sicer so neveljavni. Z nastavitvami registrov določimo režim delovanja CMOS digitalnega slikovnega senzorja ter ostale parametre, kot so [25]:

- upravljanje notranje ure senzorja (PLL)
- izbira ločljivosti videa (od 640x480 do 2592x1944)
- hitrost osveževanja (od 14 do 123 slik na sekundo – odvisno od izbrane ločljivosti)
- način delovanja statusnih signalov sinhronega paralelnega podatkovnega vodila
- začetek ali zaustavitev zajemanja slike znotraj senzorja in s tem toka podatkov senzorja preko sinhronega paralelnega podatkovnega vodila

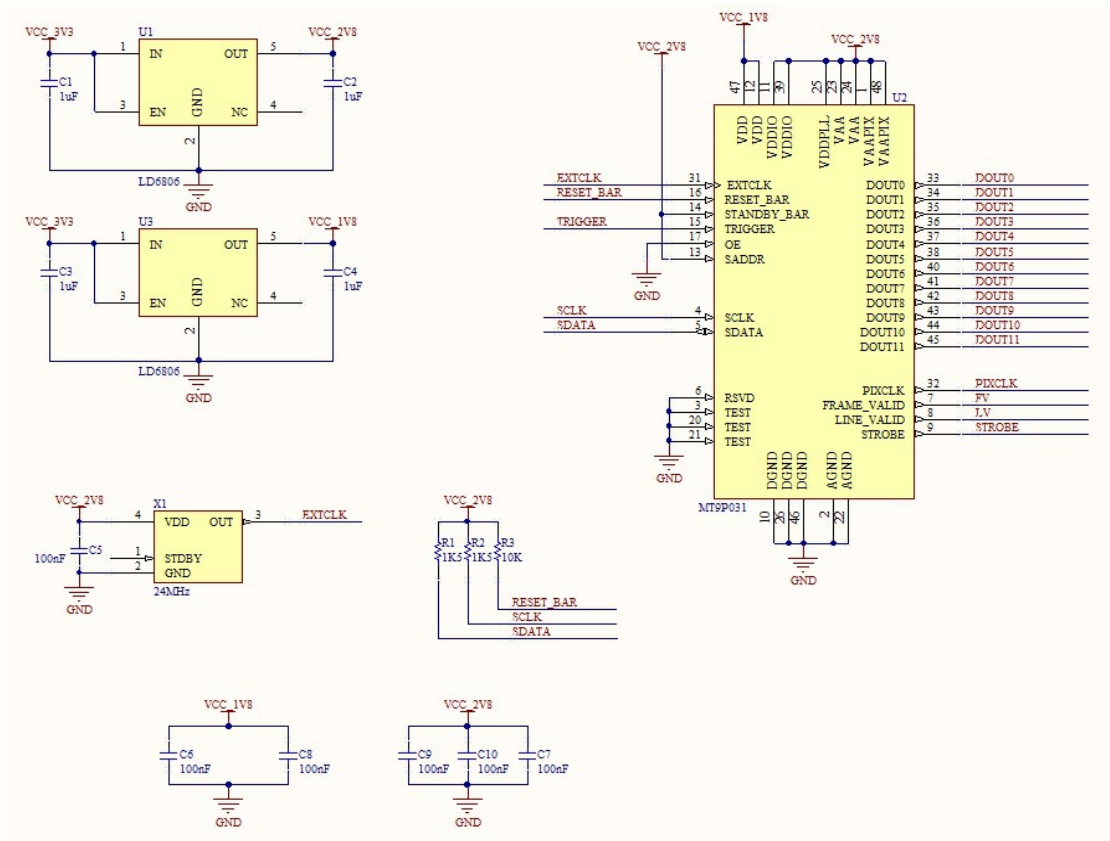
Slika 3.4 prikazuje povezavo CMOS digitalnega slikovnega senzorja MT9E001 s programirljivim vezjem FPGA, torej prvotno zasnovo digitalnega sistema, slika 3.5 pa pripadajoči izsek povezav senzorja na zgornjem signalnem sloju tiskanega vezja.



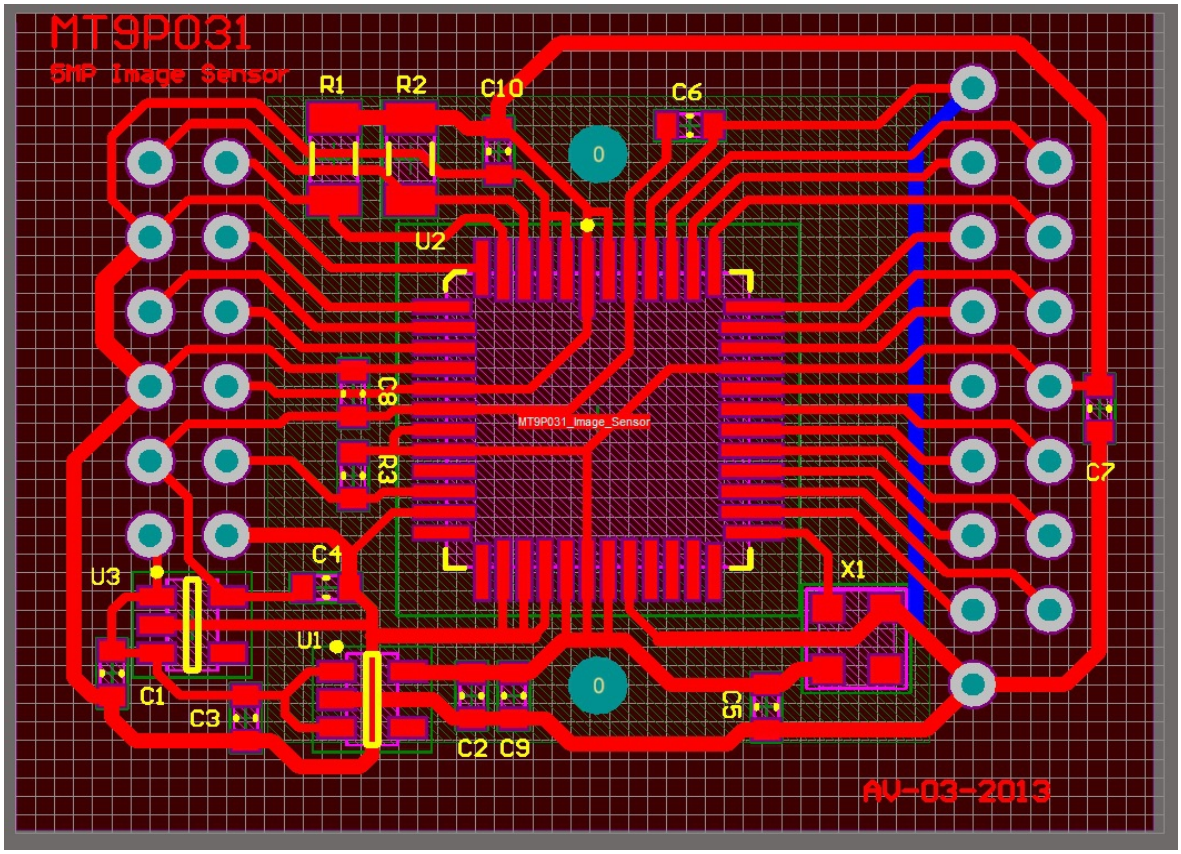
Slika 3.4: Električna shema povezav CMOS digitalnega slikovnega senzorja MT9E001 s programirljivim vezjem FPGA



Slika 3.5: Povezava CMOS digitalnega slikovnega senzorja MT9E001 s programirljivim vezjem FPGA na tiskanem vezju

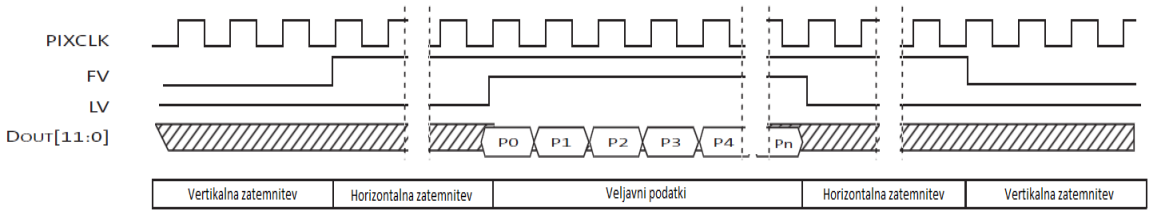


Slika 3.6: Električna shema dodatne tiskanine s CMOS digitalnim slikovnim senzorjem MT9P031



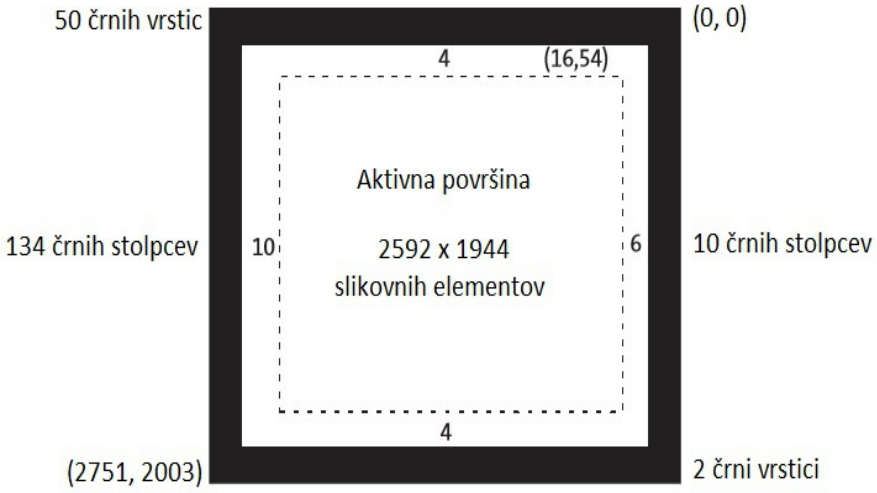
Slika 3.7: Dodatna tiskanina s CMOS digitalnim slikovnim senzorjem MT9P031

Zunanja ura senzorja mora za normalno delovanje biti v mejah od 6 do 27 MHz. Za izvor ure senzorja sem izbral generator ure v integrirani izvedbi frekvence 20 MHz. Spodnja slika 3.8 prikazuje časovni diagram in spreminjanje statusnih ter podatkovnih signalov sinhronega paralelnega podatkovnega vodila senzorja.

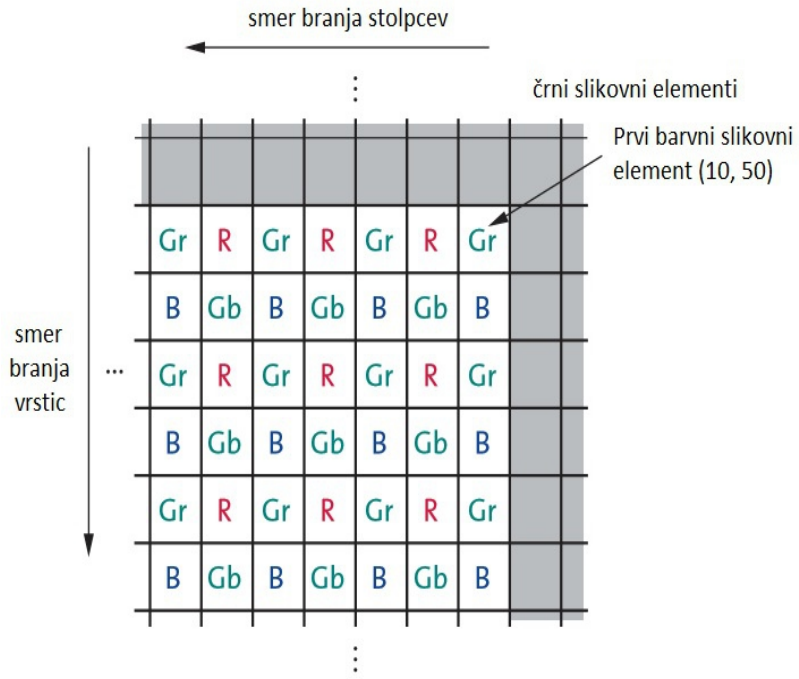


Slika 3.8: Časovni potek podatkovnih in statusnih signalov sinhronega paralelnega vodila CMOS digitalnega slikovnega senzorja [25]

Največja ločljivost CMOS digitalnega slikovnega senzorja MT9P031 je pet milijonov točk ob frekvenci osveževanja slike štirinajst slik na sekundo. Aktivno površino senzorja sestavlja 2592x1944 slikovnih elementov (slika 3.9), razporejenih v RGB Bayerjev vzorec, kot je prikazano na sliki 3.10.



Slika 3.9: Aktivno območje CMOS digitalnega slikovnega senzorja [25]



Slika 3.10: Razporeditev slikovnih elementov znotraj aktivnega območja CMOS digitalnega slikovnega senzorja v Bayerjev vzorec [25]

Ločljivost senzorja lahko zmanjšamo z ustreznimi nastavitvami vrednosti pripadajočih registrov. Ob zmanjšani ločljivosti senzorja se vidno polje ne zmanjša, saj senzor nudi možnost izpuščanja slikovnih elementov tako po stolpcih, kakor tudi po vrsticah pri branju vrednosti slikovnih elementov aktivne površine senzorja. Izpuščanje vrstic in stolpcev med branjem slikovnih elementov lahko nastavljamo neodvisno. Pri branju slikovnih elementov lahko izpustimo eno ali tri vrstice. Enako velja za stolpce aktivnega področja senzorja. Za predstavljen digitalni sistem sem ločljivost slike zmanjšal na 640x480 slikovnih elementov.

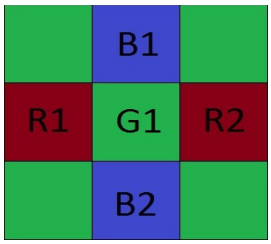
Časovni in krmilni del senzorja skrbita za to, da senzor absorbira vpadno svetlobo točno določen čas med resetom in odčitavanjem pripadajočih analognih vrednosti posamezne vrstice v barvnem polju Bayerjevega vzorca. Čas izpostavljenosti vpadni svetlobi spreminjamo z nastavljanjem intervala med resetom in odčitavanjem vrstice. Odčitane analogne vrednosti so speljane preko analognih ojačevalnikov do 12-bitnega analogno-digitalnega pretvornika. Ojačanje analognih ojačevalnikov lahko spreminjamo s spreminjanjem vrednosti pripadajočih registrov. Podobno lahko spreminjamo tudi digitalno ojačanje. Nastavitev analognega in digitalnega ojačevalnika je potrebno prilagoditi jakosti vpadne svetlobe, ki ji je senzor izpostavljen. Če so ojačanja prevelika, je slika presvetla in obratno. Če so nastavljena ojačanja premajhna, je slika pretemna. Vzorčenje barvnih komponent senzorja je izvedeno z 12-bitnim analogno-digitalnim pretvornikom, zato je tudi podatkovno vodilo CMOS digitalnega slikovnega senzorja 12-bitno.

Izhodnih vrednosti slikovnih elementov senzorja ne moremo direktno uporabiti za prikaz slike, ampak je potrebno predhodno slikovne elemente senzorja znotraj vezja FPGA filtrirati z RGB Bayerjevim filtrom, katerega okno je velikosti 3x3 slikovnih elementov. Po filtriranju dobimo za vsakega izmed slikovnih elementov v RGB Bayerjevem vzorcu pripadajočo vrednost slikovnega elementa v RGB888 formatu. RGB vrednost posameznega slikovnega elementa dobimo s pomikanjem okna filtra preko celotne slike z RGB Bayerjevim vzorcem. Posamezne barvne komponente RGB dobimo z interpolacijo posameznih barv znotraj okna. Iz RGB Bayerjevega vzorca na sliki 3.10 je razvidno, da imamo v vzorcu dva tipa vrstic. Vse lihe vrstice so sestavljene iz množice slikovnih elementov zelene in rdeče barve, ki se izmenjujeta za vsak slikovni element v vrstici senzorja. Podobno imajo vse sode vrstice strukturo sestavljeno samo iz modrih in zelenih slikovnih elementov, ki se prav tako izmenjujeta v vzorcu znotraj ene vrstice. Za določitev barvne komponente opazovanega slikovnega elementa, ki sovpada s središčem okna RGB Bayerjevega filtra za vsako izmed

vrstic ne uporabimo linearne interpolacije. Za centralne elemente znotraj pozicije okna vzamemo kar vrednost pripadajoče barvne komponente prebrane iz CMOS digitalnega slikovnega senzorja. Preostali dve barvni komponenti dobimo z linearno interpolacijo RGB komponent znotraj okna filtra. Iz indeksov barvnih komponent v izračunih (3.2 – 3.13) je razvidno, da interpoliramo le barvni komponenti, ki ne sovpadata s središčem okna Bayerjevega filtra.

Izračun RGB barvnih komponent slikovnih elementov za lihe vrstice:

a) lihi slikovni elementi lihih vrstic



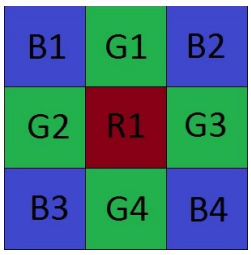
Slika 3.11: Centralni element zelene barve RGB Bayerjevega vzorca za lihe vrstice

$$R = \frac{(R1 + R2)}{2} \tag{3.2}$$

$$G = G1 \tag{3.3}$$

$$B = \frac{(B1 + B2)}{2} \tag{3.4}$$

b) sodi slikovni elementi lihih vrstic



Slika 3.12: Centralni element rdeče barve RGB Bayerjevega vzorca za lihe vrstice

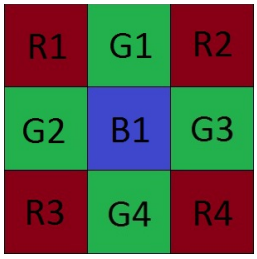
$$R = R_1 \tag{3.5}$$

$$G = \frac{(G1 + G2 + G3 + G4)}{4} \tag{3.6}$$

$$B = \frac{(B1 + B2 + B3 + B4)}{2} \tag{3.7}$$

Izračun RGB barvnih komponent slikovnih elementov za sode vrstice:

a) lihi slikovni elementi sodih vrstic



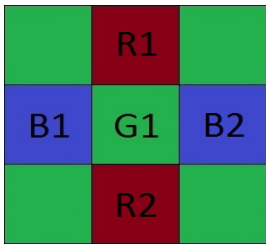
Slika 3.13: Centralni element modre barve RGB Bayerjevega vzorca za sode vrstice

$$R = \frac{(R1 + R2 + R3 + R4)}{4} \tag{3.8}$$

$$G = \frac{(G1 + G2 + G3 + G4)}{4} \tag{3.9}$$

$$B = B1 \tag{3.10}$$

b) sodi slikovni elementi sodih vrstic



Slika 3.14: Centralni element zelene barve RGB Bayerjevega vzorca za sode vrstice

$$R = \frac{(R1 + R2)}{2} \tag{3.11}$$

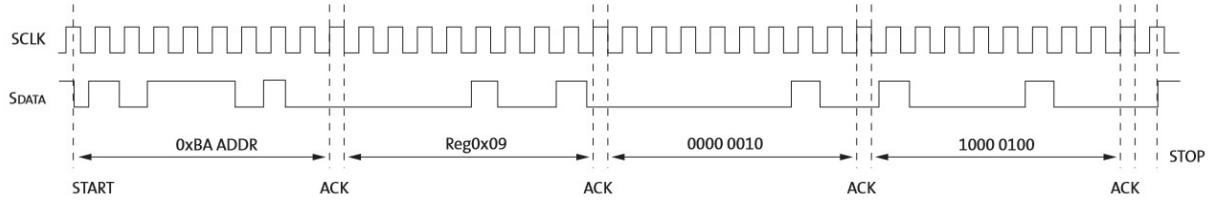
$$G = G1 \tag{3.12}$$

$$B = \frac{(B1 + B2)}{2} \tag{3.13}$$

Kot sem že predhodno omenil, dostopamo do registrov senzorja preko sinhronega serijskega komunikacijskega vodila I2C, ki ga sestavljata ura in dvosmerni podatkovni signal, ki je tipa odprti kolektor. Hitrost prenosa podatkov po I2C vodilu je odvisna od frekvence ure vodila, ki za počasne naprave znaša 100 kHz, za hitre naprave pa 400 kHz. Komunikacija na vodilu I2C poteka po protokolu. Vsaka naprava na vodilu I2C ima svoj naslov. Na vodilu I2C je zmeraj le ena tako imenovana glavna naprava (angl. master), ki sproži začetek prenosa podatkov na I2C vodilu, vse ostale naprave so podrejene naprave (angl. slave). V digitalnem sistemu sem glavno napravo implementiral znotraj programirljivega vezja FPGA. Sliki 3.15 in 3.16 prikazujeta, kako preko I2C komunikacijskega vodila iz senzorja beremo vrednosti registrov ter kako v registre zapisujemo vrednosti. Naslovi registrov znotraj CMOS

digitalnega slikovnega senzorja so 8-bitni, vrednosti registrov na teh naslovih pa so 16-bitne.

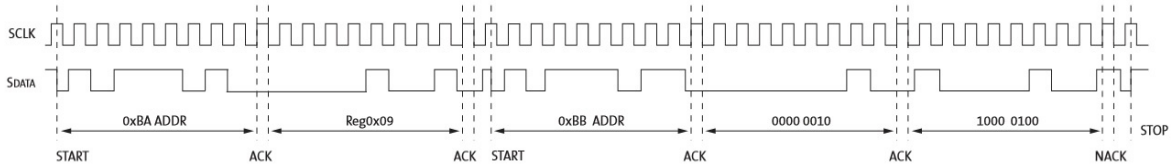
Tako bralni kot tudi pisalni cikel se na vodilu I2C pričneta z začetnim bitom (START), kateremu sledi 7-bitni naslov naprave, s katero želimo komunicirati. Osmi bit označuje v splošnem operacijo, ki jo želimo izvršiti na vodilu I2C. Logična ničla pomeni pisanje, logična enica pa branje. Ker v tej fazi podrejeni napravi pošiljamo naslov registra, v katerega bomo pisali ali pa iz njega brali, mora biti osmi bit enak nič. Deveti bit je potrditveni bit (ACK), s katerim podrejena naprava potrdi prisotnost na vodilu tako, da povleče podatkovno linijo proti masi. O negativni potrditvi (NACK) govorimo, kadar ostane podatkovna linija v fazi potrjevanja na nivoju logične enice. V vseh primerih nepotrditve se komunikacija na vodilu takoj prekine z napako. V primeru, da podrejena naprava potrdi prisotnost na vodilu I2C s potrditvenim bitom (ACK), se komunikacija nadaljuje z 8-bitnim naslovom registra, ki ga želimo prebrati oziroma vanj pisati. Če je operacija, ki jo izvajamo na vodilu I2C na primer pisanje v register, potem mora podrejena naprava po prejetem 8-bitnem naslovu registra potrditi s potrditvenim bitom (ACK), da je uspešno prejela zahtevo po pisanju v register. V nasprotnem primeru se komunikacija prekine z napako. Po uspešni potrditvi (ACK) se preko I2C vodila prenese najprej podatkovna beseda z višjo utežno vrednostjo, ki ji ponovno sledi potrditev (ACK) ali pa tudi nepotrditev (NACK) s strani podrejene naprave. V primeru potrditve (ACK) se pošlje še podatkovna beseda z nižjo utežno vrednostjo, kateri pa ne sledi nepotrditev s strani podrejene naprave (NACK). Komunikacija se nato zaključi z zaključnim bitom (STOP), ki označuje uspešen konec pisalnega cikla na I2C vodilu.



Slika 3.15: Časovni diagram pisanja vrednosti 0x0284 v register z naslovom 0x09 CMOS slikovnega senzorja preko I2C vodila [25]

Prvi dve fazi, in sicer naslavljanje podrejene naprave in pošiljanje 8-bitnega naslova registra, katerega vsebino želimo tokrat prebrati, sta popolnoma identični pisalnemu ciklu. Bralni cikel se od pisalnega cikla razlikuje po tem, da se po prejetem potrditvenem bitu (ACK) in s tem poslani podatkovni besedi, ki določa naslov registra, katerega želimo prebrati, še enkrat pošlje začetni bit. Sledi ponovno naslavljanje podrejene naprave, a tokrat je osmi bit v fazi naslavljanja enak logični enici, ki označuje zahtevo po branju. Če podrejena naprava pošlje

potrditveni bit (ACK), potem glavna naprava prebere najprej 8-bitno podatkovno besedo z višjo utežno vrednostjo. Uspešen sprejem mora tokrat glavna naprava potrditi s potrditvenim bitom (ACK) podrejeni napravi, s čimer ji signalizira uspešen prenos podatkovne besede. Sledi še branje podatkovne besede z nižjo utežno vrednostjo. Komunikacija se tudi tokrat zaključi podobno kot v primeru pisalnega cikla z nepotrditvenim signalom (NACK) ter zaključnim bitom (STOP), ki označuje konec komunikacije.



Slika 3.16: Časovni diagram branja vrednosti 0x0284 iz registra z naslovom 0x09 CMOS slikovnega senzorja preko I2C vodila [25]

Za normalno uporabo CMOS digitalnega slikovnega senzorja je potrebno senzorju dodati zbirno lečo, ki vpadno svetlobo usmeri v aktivno področje senzorja. V ta namen sem uporabil zbirno lečo DSL377 proizvajalca Sunex.

Nekaj glavnih karakteristik uporabljene zbirne leče DSL377 [30]:

- miniaturna širokokotna leča s prilagojenim popačenjem
- format senzorja: 1/2.5-palca and 1/2.3-palca
- goriščna razdalja: 2.5 mm
- premer senzorja: 7.7 mm
- IR filter: premaz na površini leče, mejna valovna dolžina je 650 nm +/- 10 nm

3.4.3 Programirljivo vezje FPGA (Xilinx XC6SLX45)

Na tržišču je kar nekaj proizvajalcev programirljivih vezij FPGA (Xilinx, Lattice, Altera…). Za implementacijo sem uporabil programirljivo vezje serije Spartan-6 XC6SLX45 proizvajalca Xilinx [9]. Izvorna koda za vezje FPGA je napisana v visokonivojskem strojno opisnem jeziku VHDL. Uporabljeno programirljivo vezje ima že integriran krmilnik za zunanji pomnilnik, kar je neprimerno olajšalo sam razvoj. V splošnem imajo programirljiva vezja FPGA precej regularno notranjo strukturo v primerjavi z na primer splošno-namenskimi

procesorji. Osnovni gradniki programirljivih vezij FPGA so logične celice, razporejene v matriko po celotni površini vezja. Za povezavo med posameznimi logičnimi celicami je na voljo matrika povezav. Poleg teh povezav imamo še posebne globalne povezovalne linije, ki se uporabljajo za distribucijo ure programirljivega vezja FPGA do vseh sekvenčnih gradnikov. Posebnost globalnih povezav je v tem, da je razlika v zakasnitvi ure znotraj programirljivega vezja FPGA do sekvenčnih gradnikov minimalna. Vsak izmed priključnih pinov programirljivih vezij FPGA ima pripadajoči IOB blok. V tabeli 3.2 najdemo podatke o programirljivih vezjih FPGA družine Spartan-6 proizvajalca Xilinx. FPGA vezja znotraj družine se med seboj razlikujejo po številu posameznih gradnikov, ki so na voljo, velikosti ohišja čipa in številu priključnih pinov, ki so na voljo uporabniku [9].

Naprava	Logične celice[1]	Konfigurabilni logični bloki (CLB)			DSP48A1 Rezine[3]	Blokovni RAM bloki		CMTs[5]	Pomnilniški krmilni bloki (Max)[6]	PCI Express bloki	GTP bloki	Število I/O bank	Max I/O na voljo
		Rezine[2]	Flip-Flopi	MAX Porazdeljeni RAM (kb)		18 Kb[4]	Max (Kb)						
XC6SLX4	3,840	600	4,800	75	8	12	216	2	0	0	0	4	132
XC6SLX9	9,152	1,430	11,440	90	16	32	576	2	2	0	0	4	200
XC6SLX16	14,579	2,278	18,224	136	32	32	576	2	2	0	0	4	232
XC6SLX25	24,051	3,758	30,064	229	38	52	936	2	2	0	0	4	266
XC6SLX45	43,661	6,822	54,576	401	58	116	2,088	4	2	0	0	4	358
XC6SLX75	74,637	11,662	93,296	692	132	172	3,096	6	4	0	0	6	408
XC6SLX100	101,261	15,822	126,576	976	180	268	4,824	6	4	0	0	6	480
XC6SLX150	147,443	23,038	184,304	1,355	180	268	4,824	6	4	0	0	6	576
XC6SLX25T	24,051	3,758	30,064	229	38	52	936	2	2	1	2	4	250
XC6SLX45T	43,661	6,822	54,576	401	58	116	2,088	4	2	1	4	4	296
XC6SLX75T	74,637	11,662	93,296	692	132	172	3,096	6	4	1	8	6	348
XC6SLX100T	101,261	15,822	126,576	976	180	268	4,824	6	4	1	8	6	498
XC6SLX150T	147,443	23,038	184,304	1,355	180	268	4,824	6	4	1	8	6	540

Opombe:
1. **Spartan 6 logične celice omogočajo večjo zmogljivost zardi 6 vhodne LUT arhitekture.**
2. **Vsaka rezina Spartan-6 FPGA vsebuje 4 LUT-e in osem filp-flopov.**
3. **Vsaka rezina DSP48A1 vsebuje množilnik 18 x 18, seštevalnik in akumulator.**
4. **Blokovni RAM bloki so v osnovi velikosti 18 kb, a jih lahko uporabimo kot dva neodvisna 9 kb bloka.**
5. **Vsak CMT vsebuje dva DCM gradnika in en PLL.**
6. **Pomnilniški krmilni bloki niso podprti v hitrostnem razredu -3N.**

Tabela 3.2: Družina programirljivih vezij FPGA Spartan-6 [9]

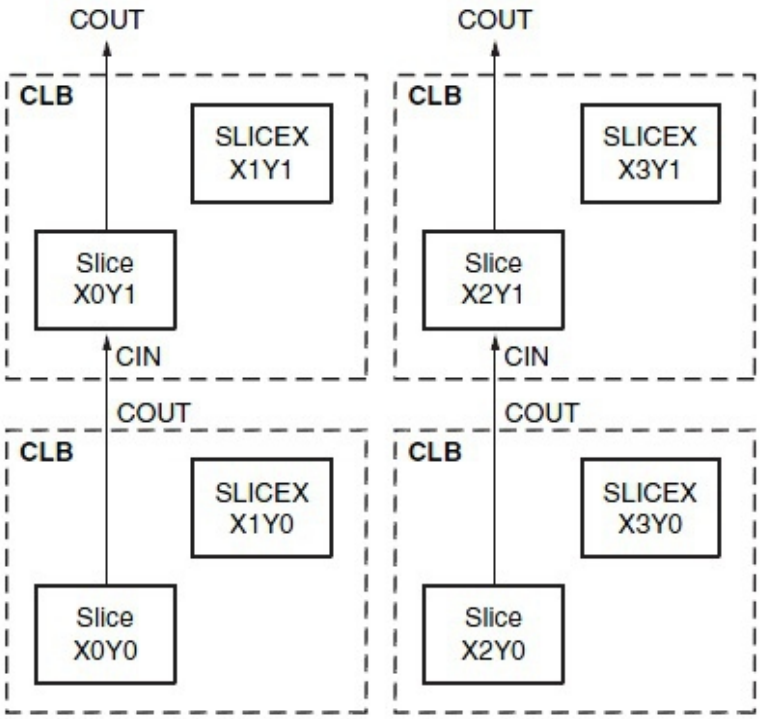
3.4.3.1 Enota za upravljanje ure (CMT)

Vsako vezje FPGA iz družine Spartan-6 lahko vsebuje do dvanajst DCM gradnikov. Vezje XC6SLX45 ima na voljo 6 CMT blokov. Vsak CMT sestavljata dva DCM gradnika in en PLL gradnik. PLL služi kot frekvenčni sintetizator, katerega razpon sega od 400 MHz pa vse tja do 1080 MHz. Srce PLL gradnika je napetostno krmiljen oscilator. Programirljivo

vezje FPGA potrebuje za generator notranje ure vezja vsaj en zunanji vir ure. Z gradniki DCM lahko frekvenco vhodne ure povečamo na dvakratno vrednost, ali na primer ustvarimo fazno zamaknjene signale za 0°, 90°, 180°, ali 270°. DCM gradnik omogoča tudi sintezo ure določene frekvence, pri čemer lahko izbiramo med različnimi vrednostmi množilnega faktorja (od 2 do 32) in vrednostmi delilnega faktorja (od 1 do 32) [14].

3.4.3.2 Konfigurabilni logični bloki (CLB)

Konfigurabilni logični bloki so v programirljivem vezju FPGA zloženi v matrično strukturo. V družini programirljivih vezij FPGA serije Spartan-6 so sestavljeni iz dveh rezin (SLICE), ki se nahajata druga ob drugi v dveh zaporednih stolpcih matrične strukture. Ločimo med tremi različnimi tipi rezin: SLICEM, SLICEL in SLICEX. Vsaka izmed rezin je sestavljena iz štirih gradnikov LUT, osmih spominskih celic DFF in ostalih logičnih



Slika 3.17: Vrstična in stolpična odvisnost med CLB in SLICE [9]

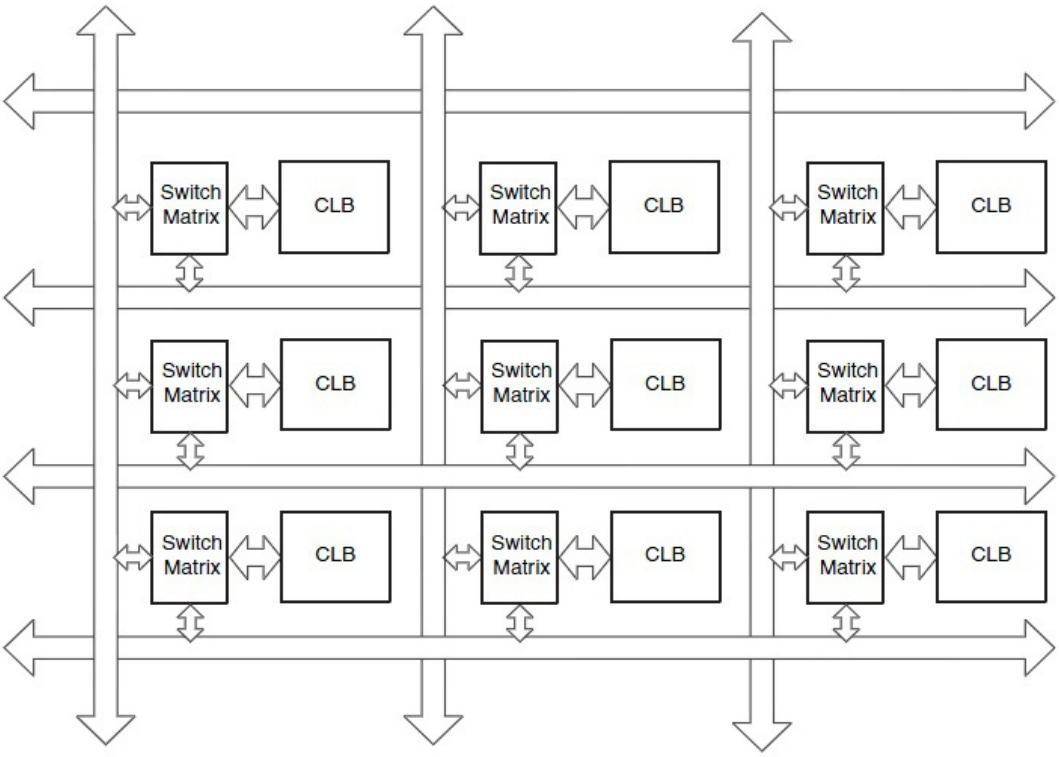
gradnikov, kot so logična vrata, izbiralniki, notranja matrika povezav. Gradniki LUT so v osnovi namenjeni implementaciji kombinacijske ter sekvenčne logike, lahko pa jih uporabimo tudi kot porazdeljeni 64-bitni RAM pomnilnik [11].

3.4.3.3 Vhodno-izhodni bloki (IOB)

Vhodno-izhodni bloki so v programirljivem vezju FPGA družine Spartan-6 razdeljeni v štiri ali šest bank. Vsako izmed bank lahko, če je to potrebno, napajamo z drugo napajalno napetostjo, kar pomeni, da na tiskanem vezju ni potrebno uporabiti pretvornikov napetostnih nivojev med različnimi I/O standardi za priključitev na vezje FPGA, le z ustrezno napajalno napetostjo glede na uporabljen I/O standard gradnikov napajamo želeno banko. Vsakemu priključnemu pinu vezja FPGA pripada po en vhodno-izhodni blok. Vhodno-izhodni bloki so razporejeni tako, da so fizično postavljeni kar se da blizu dotičnega priključnega pina. IOB vsebuje dve spominski celici DFF, ki služita za sinhronizacijo vhodnih in izhodnih signalov na sistemsko uro vezja FPGA. Zaradi same postavitve IOB bloka je zakasnitev med priključnim pinom in spominsko celico IOB bloka minimalna.

3.4.3.4 Matrika povezav

Matrika povezav znotraj vezja FPGA je sestavljena iz povezav, ki omogočajo povezave med konfigurabilnimi logičnimi bloki, vhodno-izhodnimi bloki (slika 3.18). V matriki povezav najdemo še globalne povezave, namenjene povezovanju sistemske ure z vsemi sinhronimi gradniki znotraj vezja FPGA. Povezovalne linije, ki služijo distribuciji ure znotraj vezja FPGA so izvedena tako, da so razlike v zakasnitvah ure do vseh sinhronih gradnikov v vezju FPGA minimalne.



Slika 3.18: Povezovalni kanali blokov CLB [9]

3.4.3.5 Pomnilniški krmilni blok (MCB)

Skoraj vsako vezje FPGA iz družine Spartan-6 vsebuje namenski pomnilniški krmilni blok MCB (angl. Memory Controller Block). Vsak pomnilniški krmilni blok lahko povežemo z enim zunanjim pomnilnikom SDRAM (DDR, DDR2, DDR3 ali LPDDR). Največja hitrost, ki jo lahko dosežemo, je 800 Mb/s. Za zunanji pomnilnik s 16-bitnim podatkovnim vodilom to pomeni, da je največja teoretična zmogljivost prenosa podatkov enaka:

pretok podatkov = 2 * f * 16 = 2 * 400 Mb/s × 16 = 12.8 Gb/s / 8 = 1.6 GB/s

Na MCB lahko povežemo zunanji pomnilnik s 4, 8 ali 16-bitnim podatkovnim vodilom.

3.4.3.6 Notranji pomnilniki

Vezja FPGA družine Spartan-6 vsebujejo dva tipa notranjega pomnilnika. Prvi je porazdeljeni RAM pomnilnik, ki je realiziran z gradniki LUT, drugi pa je blokovni dvovratni RAM pomnilnik. Količina porazdeljenega RAM pomnilnika vezja FPGA je odvisna od števila rezin (SLICE), ki jih vezje vsebuje. Vezja FPGA lahko imajo od 12 pa vse do 268 blokovnih

dvovratnih RAM pomnilnikov, točno število pa je odvisno od izbranega FPGA vezja iz družine. Širino podatkovnega vodila blokovnega RAM pomnilnika lahko konfiguriramo na naslednje velikosti: 16 k × 1, 8 k × 2, 4 k × 4, 2 k × 9 (ali 8), 1 k × 18 (ali 16) ali 512 × 36 (ali 32). Vsak blokovni RAM pomnilnik lahko razdelimo v dva popolnoma neodvisna 9 kb pomnilnika. Vsakega posebej lahko uporabimo v eni izmed konfiguracij: 8 k x 1, 512 x 18, 256 x 36. Količino notranjega RAM pomnilnika v programirljivih vezjih iz družine Spartan-6 prikazuje tabela 3.2 .

3.4.4 Zunanji pomnilnik

Notranji pomnilnik programirljivega vezja FPGA je premajhen za shranjevanje zajete slike iz CMOS digitalnega slikovnega senzorja in vmesnih rezultatov filtriranja, zato je bilo potrebno sistemu dodati zunanji pomnilnik. Izbral sem DDR3 SDRAM pomnilnik kapacitete 128 MB MT41J64M16 proizvajalca Micron [28]. Programirljiva vezja FPGA iz družine Spartan-6 vsebujejo vgrajeni namenski krmilnik MPMC (angl. Mutliport Memory Controller) [9] za direktno priključitev zunanjega pomnilnika. MPMC omogoča uporabo več različnih tipov zunanjih pomnilnikov, kot na primer DDR, DDR2, DDR3 ter LPDDR različnih kapacitet in bitnih širin podatkovnega vodila. Podatkovno vodilo krmilnika je omejeno na največ 16 podatkovnih bitov, naslovno vodilo pa na največ 15 bitov. Največja frekvenca ure zunanjega pomnilnika, ki jo vgrajeni namenski krmilnik zunanjega pomnilnika omogoča, je 400 MHz. Za pomnilnike tipa DDR (angl. dual data rate) to pomeni, da je frekvenca dejansko enaka dvakratniku frekvence ure, saj se podatki prenašajo tako ob pozitivni, kot tudi ob negativni fronti ure. Največji teoretični pretok podatkov, ki ga lahko dosežemo na primer z uporabo zunanjega pomnilnika s šestnajst bitnim podatkovnim vodilom na sekundo, je enak:

$$\text{pretok podatkov} = 2 * f * (16\ b / 8) = 2 * 400\ \text{MHz} \times (16\ b / 8) = 1600\ \text{MB/s} \qquad (3.14)$$

Frekvenca ure, ki sem jo izbral za zunanji pomnilnik znaša 300 MHz, kar zadosti zahtevam po izvajanju obdelave video slike v realnem času. Z izbrano frekvenco znaša največji možni pretok podatkov:

$$\text{pretok podatkov} = 2 * f * (16\ b / 8) = 2 * 300\ \text{MHz} \times (16\ b / 8) = 1200\ \text{MB/s} \qquad (3.15)$$

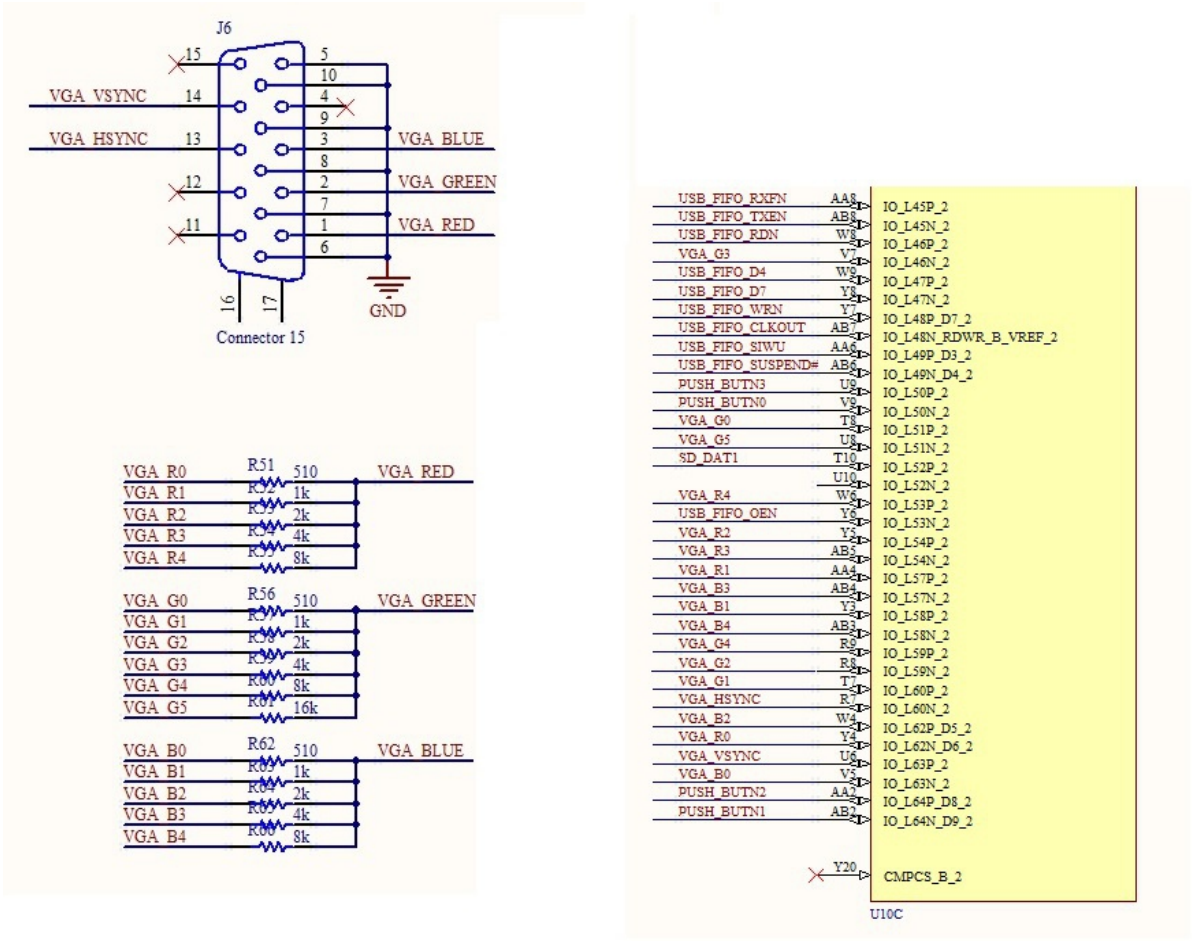
3.4.5 Asinhroni serijski komunikacijski vmesnik

Asinhroni serijski komunikacijski vmesnik sem uporabil za verifikacijo delovanja morfoloških operacij znotraj vezja FPGA. V ta namen sem v vezju FPGA implementiral asinhroni serijski sprejemnik, katerega hitrost prenosa je fiksna in znaša 921600 b/s. Za pretvorbo CMOS v RS-232 napetostne nivoje sem uporabil integrirano vezje MAX232 proizvajalca Texas Instruments [29]. Tiskanino sem opremil z ženskim konektorjem DB-9, tako da je priklop digitalnega sistema na osebni računalnik precej enostaven. Za namene testiranja sem na osebnem računalniku razvil aplikacijo, s pomočjo katere lahko nalagamo sivinske slike v zunanji pomnilnik digitalnega sistema. Izvor videa oziroma slike krmilimo z uporabo tipke na tiskanini digitalnega sistema. Zaradi poenostavitve nisem uporabil nobenega posebnega protokola za prenos slike iz osebnega računalnika v digitalni sistem. Prenašajo se le 8-bitne vrednosti slikovnih elementov, ki se sproti shranjujejo v zunanji pomnilnik sistema. Sistem nato iz zunanjega pomnilnika neprestano bere slikovne elemente prenesene slike in izvaja trenutno izbrano kombinacijo morfoloških operacij. Obdelana slika se shranjuje nazaj v zunanji pomnilnik sistema, od koder jo nato VGA krmilna enota, implementirana znotraj programirljivega vezja FPGA, bere ter prikazuje na zaslonu.

3.4.6 Izhodni digitalno-analogni pretvornik

Za prikaz slike na VGA zaslonu je potrebno digitalne vrednosti slikovnih elementov, shranjenih v zunanjem pomnilniku, pretvoriti v ustrezne analogne vrednosti napetosti. Za vsako izmed treh osnovnih barv (rdeča, zelena in modra) sem uporabil enostavni uporovni digitalno-analogni pretvornik. To je zelo enostavna in cenovno ugodna rešitev, ki nudi zadovoljive rezultate za prikaz slike na VGA zaslonu.

Izhode programirljivega vezja FPGA, ki so speljani na vhod preprostega uporovnega digitalno-analognega pretvornika (slika 3.19), je bilo potrebno prilagoditi napetostnim nivojem zunanjega VGA zaslona. Napetost na izhodnih pinih programirljivega vezja FPGA je 3.3 V, maksimalna vhodna napetost VGA zaslona pa je 0.7 V za posamezno barvno komponento. Vhodna upornost VGA zaslona je 75 Ω. Najprej sem izračunal potrebno vrednost upornosti za primer enega samega bita za vsako izmed treh barvnih komponent. Osnova za izračun izhodne upornosti je enačba uporovnega delilnika (3.16):



Slika 3.19: Električna shema digitalno-analognega pretvornika in povezava na programirljivo vezje FPGA

$$Uvh = Uizh * Rvh / (Rvh + Rizh) \qquad (3.16)$$

pri čemer je z Uvh označena maksimalna vhodna napetost VGA zaslona, Uizh je maksimalna napetost digitalnega izhoda programirljivega vezja FPGA, Rvh pa predstavlja vhodno upornost VGA zaslona. Ker nas zanima vrednost izhodne upornosti Rizh, ki predstavlja potrebno serijsko upornost na izhodu iz programirljivega vezja FPGA, moramo gornjo enačbo uporovnega delilnika ustrezno obrniti, tako da izpostavimo Rizh:

$$(Rvh + Rizh) * Uvh = Uizh * Rvh$$

$$Rizh * Uvh = Uizh * Rvh – Uvh * Rvh$$

$$Rizh * Uvh = Rvh * (Uizh – Uvh)$$

$$Rizh = Rvh * (Uizh – Uvh) / Uvh$$

Podatki za izračun ustrezne vrednosti upornosti so:

$$Rvh = 75\ \Omega,\quad Uizh = 3.3\ V,\quad Uvh = 0.7\ V$$

$$Rizh = 75\ \Omega * (3.3\ V – 0.7\ V) / 0.7\ V$$

$$Rizh = 278\ \Omega$$

Za izdelavo N-bitnega enostavnega uporovnega digitalno-analognega pretvornika je potrebno vrednosti posameznih upornosti v vejah izbrati tako, da je skupna upornost vseh vzporednih N vej za posamezno barvno komponento približno enaka 278 Ω ter da je tok veje za bite z nižjo utežno vrednostjo polovica toka sosednjega bita z višjo utežno vrednostjo. Zaradi enostavnejše izdelave sem izbral začetno vrednost upornosti 510 Ω za bit največje utežne vrednosti, vse ostale upornosti pa lahko izračunamo rekurzivno, saj mora biti upornost vsake naslednje veje za bit nižje utežne vrednosti dvakratnik predhodne upornosti veje bita z višjo utežno vrednostjo. Od tod sledi, da so vrednosti upornosti v vejah bitov nižjih utežnih vrednosti enake: 1 kΩ, 2 kΩ, 4 kΩ, 8 kΩ in 16 kΩ. Nadomestna upornost vzporedne vezave tako izbranih upornosti je približno enaka 255 Ω, kar sicer odstopa od izračunane vrednosti 278 Ω, a storjena napaka glede na namen uporabe ni kritična in jo lahko zanemarimo.

3.5 Določitev velikosti in števila slojev tiskanega vezja

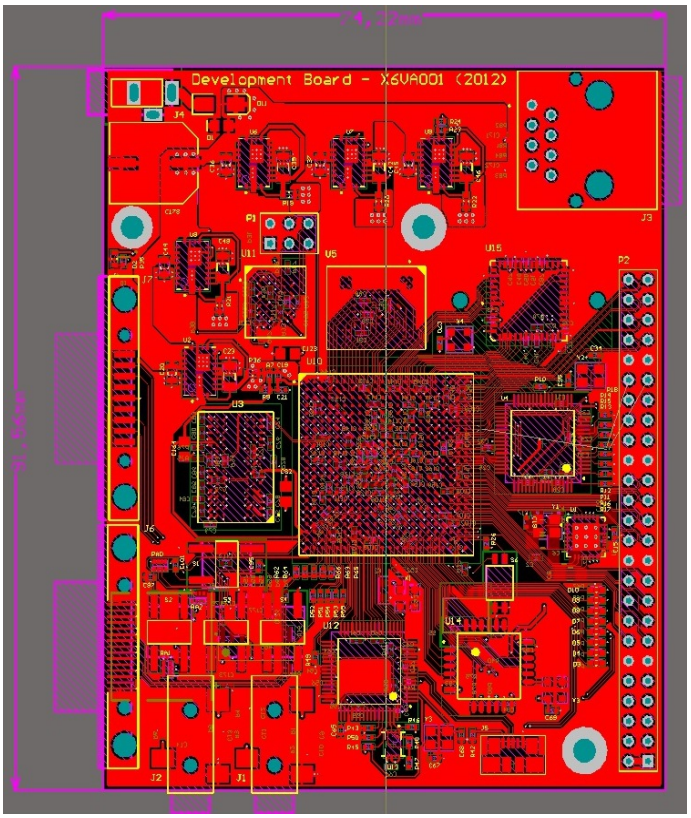
Ker se stroški izdelave večslojnih tiskanih vezij večajo s številom slojev, sem se odločil, da izdelam štirislojno tiskano vezje. Izbira števila slojev lahko kaj hitro poveča zahtevnost povezovanja komponent med seboj, saj za povezavo signalnih linij na štirislojni tiskanini na voljo v glavnem le dva sloja tiskanega vezja. Pri štirislojnih tiskaninah se praviloma notranja dva sloja uporabljata za porazdeljene napajalne površine in neprekinjeno površino mase, zunanja dva sloja pa se uporabljata za povezovanje signalnih linij. Načeloma lahko za povezovanje signalnih linij uporabimo tudi notranja napajalna sloja, vendar moramo biti pri tem previdni, da te linije ne prekinejo napajalnih površin na delih, kjer potekajo visokofrekvenčne linije na zunanjih signalnih slojih. Tak primer so signalne linije, ki povezujejo programirljivo vezje FPGA z zunanjim pomnilnikom. Za primer zunanjega pomnilnika, ki je na tiskanem vezju zagotovo kritična komponenta z vidika povezovanja, to dosežemo tako, da imajo vse signalne linije med priključnimi pini zunanjega pomnilnika in pini programirljivega vezja FPGA na sosednjem notranjem sloju neprekinjeno površino enakega potenciala vzdolž linij. Impedanca vzdolž visokofrekvenčnih linij mora biti čim bolj konstantna. Za prehode signalnih linij iz enega na drugi sloj sem uporabil vije, ki omogočajo prehod iz enega sloja tiskanega vezja na kateregakoli izmed preostalih treh slojev.

3.6 Razporeditev in povezovanje komponent na tiskanem vezju

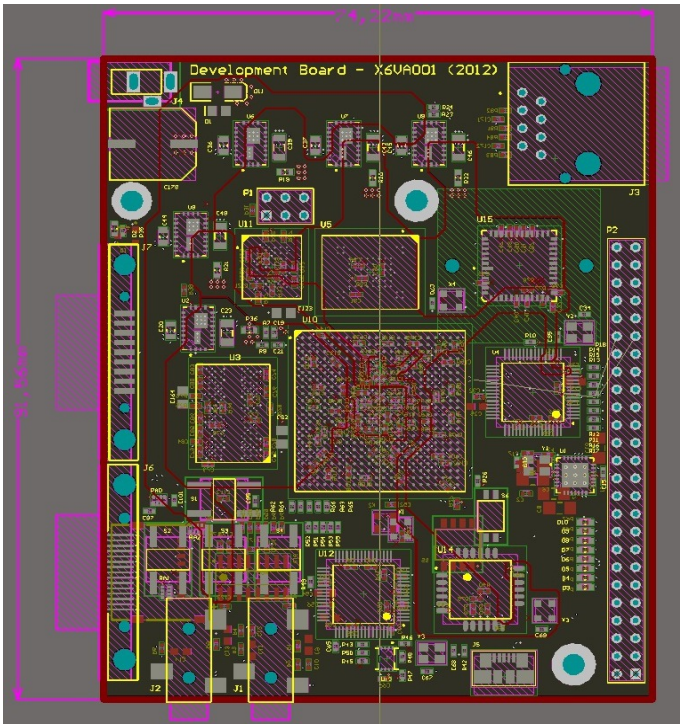
Pri razporeditvi komponent je bilo potrebno paziti, da so komponente na tiskanem vezju razporejene tako, da so povezave med komponentami čim krajše. Še posebej velja to za komponente, katerih pravilno delovanje je odvisno od dolžine linij na tiskanem vezju. Višja kot je frekvenca, s katero poteka komunikacija, toliko bolj je pomembno, da so linije impedančno prilagojene ter da je razlika med dolžinami linij, ki povezujejo med seboj dve komponenti na tiskanem vezju, zanemarljivo majhna. Izpostavil bi predvsem povezovanje zunanjega SDRAM DDR3 pomnilnika s programirljivim vezjem FPGA. Ura zunanjega pomnilnika znaša 300 MHz. Vse krmilne, podatkovne in naslovne signale zunanjega pomnilnika sem povezal na enem samem sloju tiskanega vezja, in sicer na spodnjem signalnem sloju, le za povezavo ure sem uporabil zgornji signalni sloj tiskanega vezja.

Za povezovanje zunanjega pomnilnika s programirljivim vezjem FPGA na tiskanem

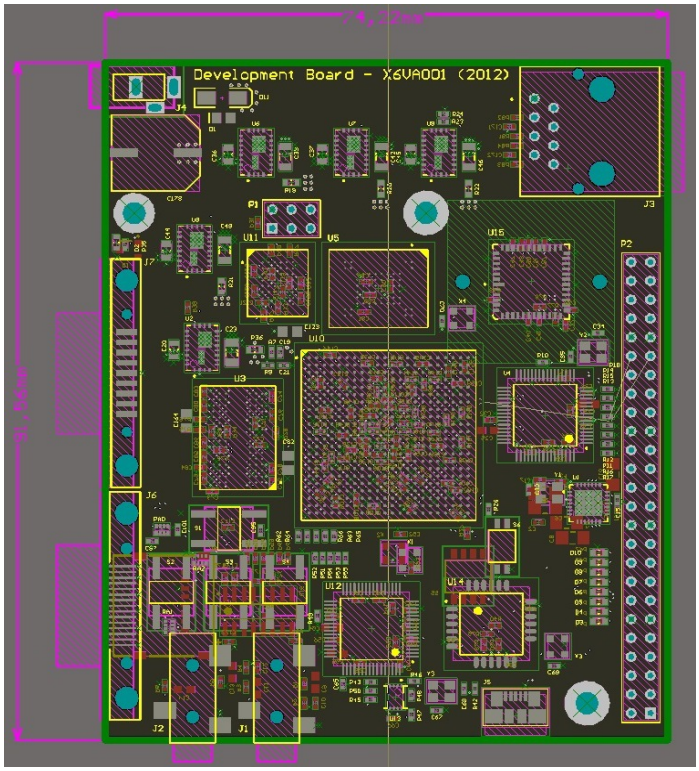
vezju, kot tudi za večino preostalih komponent, sem uporabil linije širine 0.1016 mm ter razmik med linijami 0.1016 mm. Za zagotovitev površine enakega potenciala vzdolž vseh signalnih linij zunanjega pomnilnika sem za ves tretji notranji sloj tiskanega vezja uporabil neprekinjeno površino mase. Slike 3.20, 3.21, 3.22 in 3.23 prikazujejo vse štiri sloje tiskanega vezja, kot jih lahko vidimo med samim načrtovanjem s programskim orodjem za načrtovanje tiskanih vezij Altium Designer.



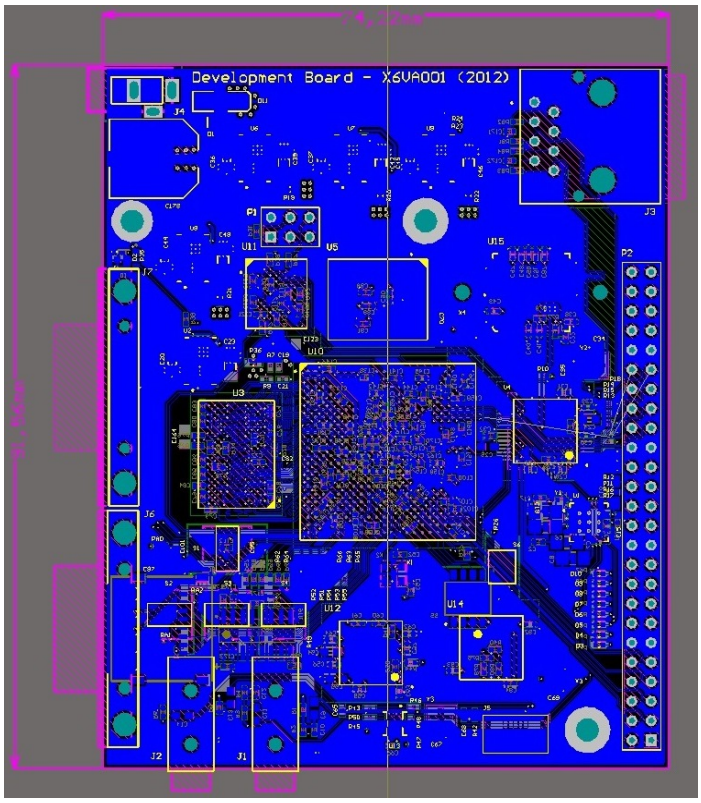
Slika 3.20: Zgornji signalni sloj tiskanega vezja (1. sloj)



Slika 3.21: Notranji napajalni sloj tiskanega vezja (2. sloj)



Slika 3.22: Notranji sloj mase (3. sloj)



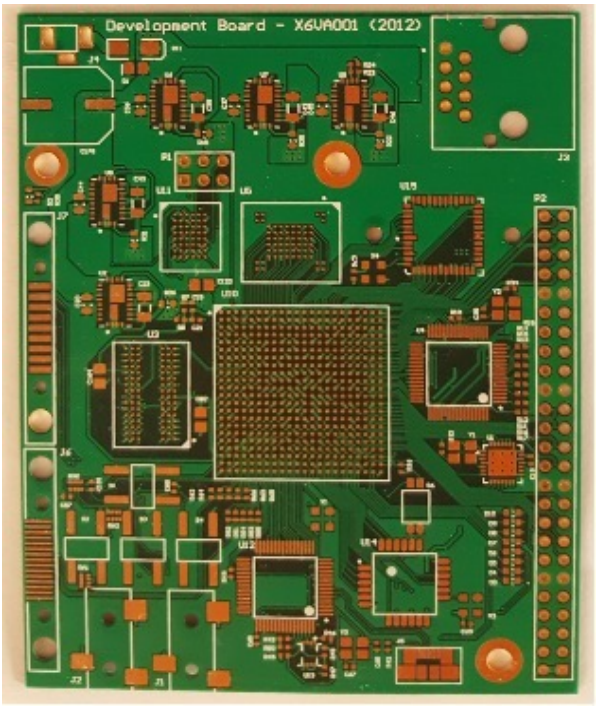
Slika 3.23: Spodnji signalni sloj tiskanega vezja (4. sloj)

Orodje Altium Designer sicer omogoča popolnoma avtomatsko povezovanje komponent, ki pa zaradi kompleksnosti samega vezja, kot se je izkazalo v času razvoja, enostavno odpove. Celotno tiskano vezje sem tako povezal ročno. Uporabna funkcija načrtovalskega orodja pa je vsekakor računalniško vodeno povezovanje podatkovnih vodil in diferencialnih signalov, pri čemer orodje samodejno išče možne kanale za vse izmed izbranih linij, ki jih želimo speljati v določeni smeri, in avtomatsko vzdržuje želeni razmik med izbranimi linijami, razmik med priključki in linijami, razmik do drugih linij ter tako upošteva vsa pravila povezovanja, ki jih orodju nastavimo pred pričetkom ali pa tudi med samim povezovanjem komponent na tiskanem vezju.

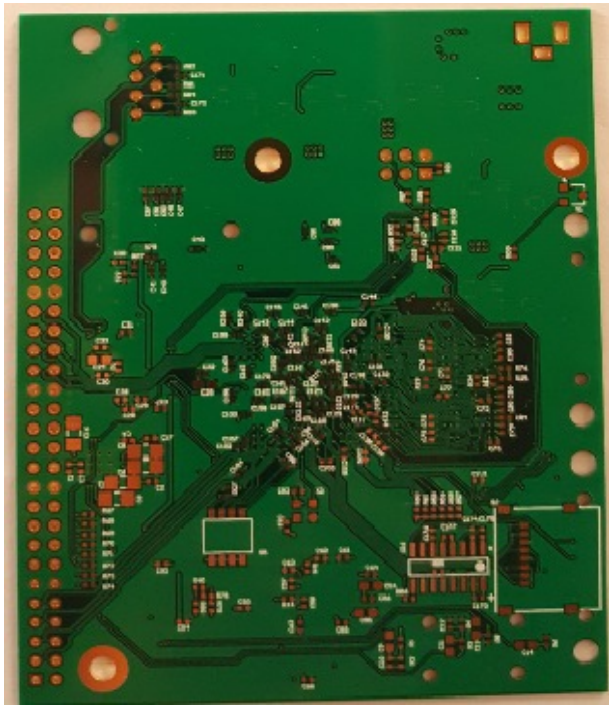
3.7 Izdelava tiskanine

Ker bi stroški izdelave prototipnih tiskanin v Sloveniji presegli 1000 €, sem dal tiskano vezje izdelati na Kitajsko in tako stroške izdelave oklestil na slabo tretjino za 50 tiskanih vezij. Prvotni strošek izdelave je precejšen, saj cena izdelave tiskanine vključuje tudi izdelavo

potrebnih mask. Vsaka nadaljnja izdelava tiskanin je zato cenejša za strošek izdelave mask, ki je v mojem primeru znašal skoraj polovico celotnih stroškov izdelave. Sliki 3.24 in 3.25 prikazujeta izdelano tiskanino in sicer iz spodnje in zgornje strani.



Slika 3.24: Zgornja stran neopremljenega izdelanega tiskanega vezja



Slika 3.25: Spodnja stran neopremljenega izdelanega tiskanega vezja

3.8 Opremljanje tiskanega vezja s komponentami

Skoraj vse komponente na izdelanem tiskanem vezju so SMD. V primerjavi s klasičnimi komponentami THT, pri katerih priključne pine komponent namestimo na tiskano vezje v odprtine na tiskanem vezju, se SMD komponente spajkajo na samo površino tiskanega vezja. Spajkanje SMD komponent z velikim številom priključnih pinov, ki se nahajajo pod samim ohišjem komponente, je z uporabo klasičnega spajkalnika praktično nemogoče. V takšnih primerih je potrebno uporabiti spajkalnik na vroči zrak, veliko bolje pa je, če imamo na voljo spajkalno postajo na vroči zrak, opremljeno z mikroskopom, vakuumsko prijemalno glavo za komponento ter mehanizmom za fino nameščanje komponente na tiskano vezje pred spajkanjem. Pasivne komponente kot so SMD upori, kondenzatorji, svetleče diode, tipke, integrirana veja s tipi ohišij, kjer so priključni pini enostavno dostopni, ter konektorje, sem na

tiskano vezje prispajkal z uporabo klasičnega spajkalnika. Programirljivo vezje FPGA, zunanji pomnilnik, konfiguracijski PROM pomnilnik za programirljivo vezje FPGA in napajalne čipe so mi pomagali prispajkati na tiskano vezje v podjetju Emsiso, d.o.o., iz Maribora, kjer imajo na voljo spajkalno postajo, namenjeno spajkanju tovrstnih zahtevnih SMD komponent.

3.9 Verifikacije ustreznega delovanja strojne opreme

Začetni korak je vključeval preverjanje ustreznega delovanja napajalnikov in obeh generatorjev ure za vezje FPGA in CMOS digitalni slikovni senzor. Pri prvi priključitvi strojne opreme na napajalni vir sem imel serijske upore na napajalnih linijah odstranjene, tako da so pod električno napetostjo bili le glavni napajalniki tiskanega vezja. Ko sem z meritvami preveril, da so izhodne napetosti neobremenjenih napajalnih čipov pravilne, sem na vezje prispajkal serijske upore v vse glavne napajalne linije. Po opravljenih meritvah napajalnih virov sem tako prvič priklopil celoten digitalni sistem na napajanje. Ta korak je zelo pomemben, saj če so v tiskanem vezju kakšne napake v smislu kratkih stikov, ki so lahko posledica napak že v sami električni shemi ali tudi posledica napak pri spajkanju, lahko določene komponente celo uničimo. Ker so glavne komponente v ohišjih BGA (angl. Ball Grid Array) in QFN (angl. Quad Flat No-leads package), ki imajo priključne pine pod samim ohišjem, je zelo težko preveriti, ali so vsi spoji med komponento in tiskanim vezjem dobri. Zato sem na zunanjem napajalnem viru nastavil tokovno zaščito na 100 mA.

Po uspešnem prvem priklopu napajanja sem se z osebnim računalnikom preko USB JATG programatorja HS2 proizvajalca Xilinx uspešno povezal s programirljivim vezjem FPGA in pripadajočim konfiguracijskim PROM pomnilnikom.

Sledilo je testiranje delovanja zunanjega DDR3 SDRAM pomnilnika. V ta namen sem uporabil procesor Microblaze proizvajalca Xilinx in testne funkcije za detekcijo pravilnega delovanja naslovnega in podatkovnega vodila ter krmilnih signalov, ki so sestavni del knjižnic programskega razvojnega orodja SDK (angl. Software Development Kit) za programiranje v okolju C.

Verifikacija strojne opreme je vključevala še ostale bloke digitalnega sistema, kot so preverjanje delovanja tipk, vklop/izklop vseh osmih LED diod. V ta namen sem moral v

visokonivojskem strojno opisnem jeziku VHDL napisati manjše testne komponente, s pomočjo katerih sem preveril delovanje omenjenih komponent na tiskanem vezju. Ker v tej fazi razvoja še nisem imel implementiranih komponent znotraj programirljivega vezja FPGA, kot sta I2C krmilnik in asinhroni serijski vmesnik, sem v sklopu verifikacije strojne opreme lahko preveril le, ali so povezave med priključnimi pini programirljivega vezja FPGA in pripadajočimi komponentami dobre. To sem preveril tako, da teh komponent v začetni fazi nisem opremil na tiskanem vezju. Tako sem lahko v vezju FPGA generiral signale in jih na priključnih pinih še neopremljenih komponent izmeril z osciloskopom.

3.10 Modifikacije strojne opreme

Zaradi težav pri verifikaciji delovanja CMOS digitalnega slikovnega senzorja MT9E001 na tiskanem vezju sem moral izdelati dodatno enoslojno tiskano vezje s CMOS digitalnim slikovnim senzorjem MT9P031, ki sem ga na osnovno tiskano vezje priklopil preko razširitvenega konektorja.

4 Implementacija digitalnega sistema v programirljivem vezju FPGA

Za opis digitalnega sistema sem uporabil visokonivojski strojno opisni jezik VHDL (angl. hardware description language). Njemu sorodni jezik je Verilog, katerega sintaksa je nekoliko drugačna. Programska razvojna orodja proizvajalcev programirljivih vezij, kot so na primer Xilinx, Altera, Lattice in drugi, nudijo poleg načrtovanja digitalnega sistema z uporabo opisnega jezika še možnost shematskega oziroma blokovnega načrtovanja sistema. Pri tovrstnem načrtovanju s pomočjo grafičnega vmesnika povezujemo komponente med seboj v funkcionalno zaključeno celoto. V določenih primerih samo programiranje sploh ni potrebno. Veliko namenskih komponent, kot so na primer filtri, množilniki, komponente za zajemanje, obdelavo in prikaz videa ter veliko drugih, lahko najdemo v knjižnicah omenjenih programskih orodij. Obstajajo tudi prevajalniki za objektni jezik C++, kjer ob uporabi namenskih knjižnic lahko opišemo strukturo in obnašanje digitalnega sistema kar v objektnem jeziku C++.

Če primerjam razvoj aplikacije v programskem jeziku C za na primer nek aplikacijski procesor ali mikrokrmilnik ter razvoj v visokonivojskem strojno opisnem jeziku VHDL, lahko rečem, da poteka razvoj za aplikacijski procesor precej hitreje. Iskanje napak v času razvoja aplikacije je enostavnejše, saj lahko program z uporabo razhroščevalnika med samim izvajanjem ustavljamo ter preverjamo določeno stanje. Pri načrtovanju v visokonivojskem strojno opisnem jeziku VHDL si večinoma pomagamo s simulacijo na osebnem računalniku, pri čemer opazujemo delovanje digitalnega sistema na način, da za izbrane vhodne vrednosti digitalnega sistema spremljamo notranje in zunanje signale v obliki grafičnega časovnega poteka. Za namene verifikacije ustrezne funkcionalnosti sistema znotraj programirljivih vezij si lahko v visokonivojskem strojno opisnem jeziku VHDL napišemo testne komponente, ki berejo vhodne podatke iz vnaprej pripravljenih datotek in iz njih generirajo vhodne testne signale v digitalni sistem, obenem pa rezultate izhodov digitalnega sistema zapisujejo v izhodne datoteke. Ker za dani digitalni sistem poznamo odziv, lahko glede na generirane vhodne vrednosti sproti preverjamo, ali je obnašanje digitalnega sistema na izhodu pravilno ali ne.

Obstaja bistvena razlika izvajanja programa med na primer mikrokrmilnikom in programirljivim vezjem FPGA. Pri mikrokrmilniku se ukazi izvajajo sekvenčno, procesi pa se

med seboj izvajajo le navidezno vzporedno. Če imamo na primer dva procesa, se bo nek določen časovni interval izvajal en proces, v drugem časovnem intervalu pa drug proces. Izjema so procesorji z več jedri, kjer se seveda določeni deli programa lahko izvajajo paralelno. Ker se preklop med procesi, ki se izvajajo znotraj enega procesorskega jedra, ki mu pravimo tudi časovno rezinjenje, izvaja dovolj hitro, je z vidika uporabnika tovrstno izvajanje paralelno. Paralelizem je prisoten predvsem pri namenskih procesorjih ali ASIC, kjer je arhitektura posebej izdelana za namene aplikacije. V programirljivem vezju FPGA lahko procese v resnici izvajamo sočasno oziroma vzporedno, saj opisujemo strukturo in obnašanje digitalnega vezja in ne pišemo ukazov, ki se bi izvajali sekvenčno, kot je to primer pri mikrokrmilniku.

4.1 Vhodna komponenta za zajem slike iz CMOS senzorja

CMOS digitalni slikovni senzor pošilja podatke o zajeti sliki sinhrono z lastno uro preko 12-bitnega podatkovnega vodila. Senzor lahko uporablja zunanjo uro direktno ali pa iz zunanje ure generira sistemsko uro s pomočjo interne fazno sklenjene zanke (PLL). Za sinhronizacijo slike se uporabljata statusna signala LV in FV, ki določata veljavnost podatkovnih signalov na vodilu. Ločljivost CMOS slikovnega digitalnega senzorja sem nastavil na 640x480 slikovnih elementov. Za dosego osveževanja slike s frekvenco 30 Hz sem uporabil notranjo fazno sklenjeno zanko (PLL). Z ustreznimi nastavitvami registrov CMOS digitalnega senzorja sem iz zunanjega signala ure, ki znaša 20 MHz, generiral notranjo uro senzorja, ki znaša približno 58 MHz.

Največji problem vezij FPGA je sinhronizacija zunanjih signalov na notranjo sistemsko uro. Najosnovnejšo obliko sinhronizacije zunanjih signalov ponavadi izvedemo z dvema v kaskado vezanima spominskima celicama DFF. Prva spominska celica spreminja svoj izhod glede na zunanji vhodni podatek skladno z notranjo sistemsko uro vezja FPGA. Ker lahko zunanji vhodni signal spremeni svojo vrednost ravno v trenutku fronte sistemske ure programirljivega vezja FPGA, obstaja verjetnost, da postane izhod spominske celice za nek določen čas nestabilen. Da bi to potencialno nestabilnost prve vhodne spominske celice minimizirali, uporabimo v seriji še drugo sinhronizacijsko spominsko celico, ki tako zmanjša verjetnost, da bi nestabilni izhod prve spominske celice DFF propagiral v notranjost vezja. Takšen dogodek bi namreč lahko imel za posledico nepravilno delovanje dela vezja, katerega

delovanje je odvisno od stanja tega vhodnega signala. Sinhronizacijo zunanjih signalov na sistemsko uro programirljivega vezja FPGA lahko realiziramo tudi s pomočjo asinhronih medpomnilnikov (FIFO), katerih značilnost je, da se pisanje in branje izvajata z različnima urama – prehod med različnimi domenami ure (angl. clock domain crossing). V primeru CMOS digitalnega slikovnega senzorja imamo opravka z zunanjim sinhronim izvorom toka podatkov, zato se pisanje v asinhroni medpomnilnik vrši sinhrono z zunanjo uro podatkovnega vodila, branje pa se izvaja sinhrono z notranjo sistemsko uro programirljivega vezja FPGA.

Za sinhronizacijo zunanjih signalov CMOS digitalnega slikovnega senzorja opisanega digitalnega sistema na notranjo sistemsko uro sem uporabil knjižnično IP komponento proizvajalca Xilinx Video Input AXI4 Lite [13], ki za sinhronizacijo signalov iz CMOS digitalnega slikovnega senzorja uporablja prav asinhroni medpomnilnik, v katerega se sinhrono z zunanjo uro senzorja zapisujejo 12-bitne vrednosti slikovnih elementov v odvisnosti od veljavnosti statusnih signalov LV in FV. Branje slikovnih elementov iz knjižnične komponente za zajemanje vhodnega videa je zelo enostavno, saj je praktično enako branju iz medpomnilnika (FIFO).

Ker CMOS digitalni slikovni senzor omogoča spreminjanje nastavitev registrov le preko I2C sinhronega serijskega komunikacijskega vodila, sem v ta namen znotraj programirljivega vezja implementiral še I2C krmilno enoto, s pomočjo katere v začetni fazi po priklopu napajanja ali po resetu vezja nastavim ustrezne vrednosti registrov.

4.2 RGB Bayerjev filter

Slikovne elemente zajete iz CMOS digitalnega slikovnega senzorja je potrebno filtrirati z Bayerjevim filtrom, katerega izhodna vrednost je podana v RGB888 formatu, kjer sem za vsako izmed treh barvnih komponent uporabil osem bitov. Velikost okna Bayerjevega filtra je 3x3 slikovne elemente. Vrednost posamezne barvne komponente (rdeča, zelena in modra) za opazovani centralni element Bayerjevega filtra dobimo z linearno interpolacijo ustreznih slikovnih elementov znotraj okna filtra.

Za pričetek postopka filtriranja slike potrebujemo zadostno količino podatkov o vhodni sliki. Ker je okno filtra velikosti 3x3, moramo začasno shraniti podatke o slikovnih elementih

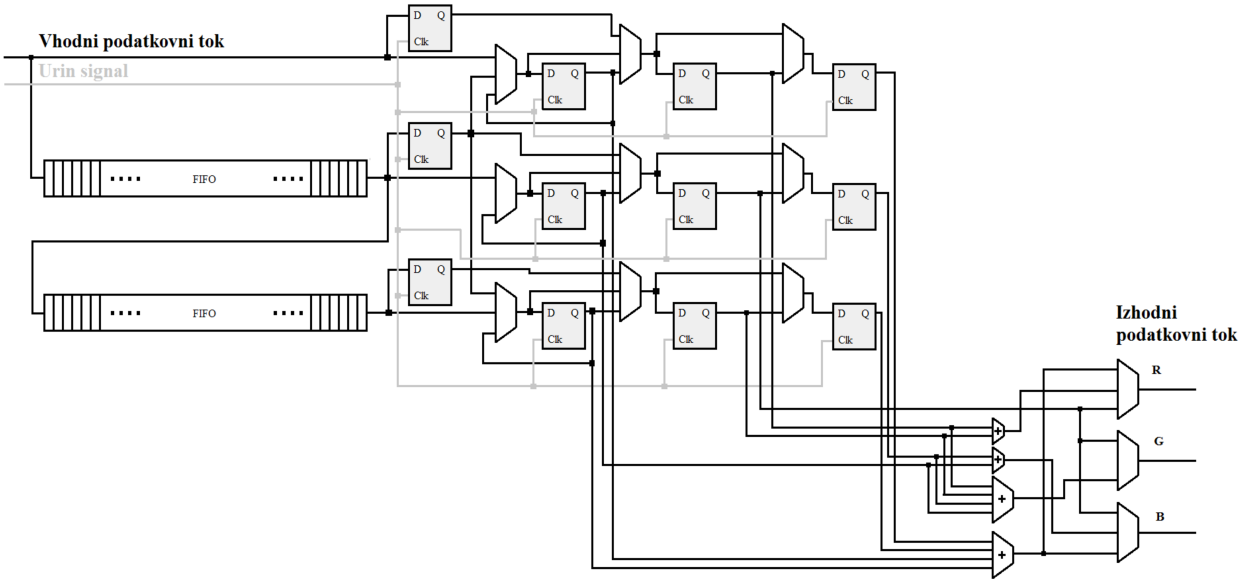
slike za dve vrstici. V ta namen sem uporabil že ustaljen pristop z uporabo dveh medpomnilnikov FIFO, po enega za vsako izmed vrstic. Vhodne podatke lahko začnemo obdelovati, ko zapolnimo prvega izmed medpomnilnikov. Poseben primer so zunanji robovi zajete slike. Za ta poseben primer obstaja več pristopov. Izbral sem metodo raztega slike pri robnih pogojih, kjer informacijo o manjkajočih podatkih, potrebnih za filtriranje, nadomestimo s podvajanjem robnih slikovnih elementov. Obstajajo še drugi pristopi, kot so na primer zrcaljenje notranjih slikovnih elementov preko robov slike, zapolnitev slikovnih elementov zunaj veljavnega območja slike z logičnimi ničlami ali z logičnimi enicami. Nobena izmed naštetih rešitev ni ravno optimalna, saj v vseh primerih nadomeščamo manjkajočo informacijo za del slike, ki v resnici ne obstaja. Posledica kateregakoli izmed uporabljenih raztegov slike je ta, da bodo vrednosti slikovnih elementov roba slike po filtriranju ob uporabi različnih metod nadomeščanja manjkajoče informacije o sliki zavzele različne vrednosti. Iz tega sledi, da se je potrebno te napake zavedati. Ker gre zgolj za robne slikovne elemente opazovane slike, lahko pri obravnavi napako zanemarimo.

Z uporabo metode z medpomnilniki lahko razdelimo filtriranje ene slike v tri faze. Prva faza je zajemanje podatkov in polnjenje medpomnilnikov, dokler ne zajamemo zadostne količine podatkov o sliki za pričetek filtriranja. Zapolniti je potrebno prvega izmed medpomnilnikov. Sledi faza, ko imamo na voljo vse podatke in filtriranje izvajamo vse dokler ne pridobimo informacije o zadnjem slikovnem elementu slike. V tem trenutku se prične zadnja faza, ko mora zadnji slikovni element vhodne slike prispeti do sredine okna filtra. Ta del imenujemo praznjenje medpomnilnikov. V vseh robnih primerih mora krmilna logika filtra poskrbeti, da se na ustrezna mesta v filtru preslikajo podatki o manjkajočih slikovnih elementih izven roba vhodne slike. Nadomeščanje manjkajočih slikovnih elementov poteka pri štirih robnih pogojih in kombinacijah le-teh:

- obdelava prve vrstice slike
- obdelava levega roba slike
- obdelava desnega roba slike
- obdelava zadnje vrstice slike

Za preslikavo oziroma nadomeščanje manjkajočih slikovnih elementov izven roba slike sem uporabil izbiralnike, ki jih krmilim s končnim avtomatom. Na izhodu iz filtra potrebujemo

poleg izhodne vrednosti še informacijo o veljavnosti, ki nam služi kot vhodni podatek v nadaljnji obdelavi podatkov slike (shranjevanje v zunanji pomnilnik, obdelava z naslednjim filtrom v kaskadi).



Slika 4.1: Poenostavljena zgradba RGB Bayerjevega filtra znotraj programirljivega vezja FPGA

4.3 Pretvorba slike v RGB formatu v sivinsko sliko

Za pretvorbo barvnega slikovnega elementa, podanega v RGB formatu, v slikovni element sivinske slike je potrebno najprej vsako izmed barvnih komponent pomnožiti z ustrezno utežno vrednostjo. Vrednost slikovnega elementa sivinske slike je enaka vsoti uteženih vrednosti treh barvnih komponent (rdeča, zelena in modra). Utežne vrednosti, s katerimi je potrebno barvne komponente pomnožiti, so decimalna števila. Ker je množenje z decimalnimi števili zahtevna operacija, se ji lahko izognemo tako, da namesto množenja z decimalnim številom uporabimo množenje s celim številom. Utežne decimalne vrednosti vsako posebej najprej pomnožimo z 2^N, da dobimo utežne vrednosti, predstavljene s celim številom, kjer decimalni del števila zanemarimo. Vsota treh uteženih barvnih komponent predstavlja vmesni rezultat, ki ga je potrebno le še deliti z 2^N. Deljenje s številom 2^N je enako pomiku v desno za N bitov. Sama implementacija deljenja z 2^N je še enostavnejša, saj vrednosti sploh ni potrebno pomikati v desno za N bitov, ampak je potrebno le prebrati

ustrezne bite višjih utežnih vrednosti. Takšen pristop sam proces deljenja in množenja decimalnih števil precej poenostavi, vnaša pa tudi napako, ki pa je v večini primerov v praksi zanemarljivo majhna. Za N sem izbral vrednost 15. Uporabil sem Intelovo enačbo za pretvorbo slikovnih elementov iz RGB formata v svetilnost Y, kjer se pretvorba utežnih vrednosti v decimalnem zapisu pretvori v celo število tako, da vrednosti pomnožimo z 2^{15}. Vrednost sivinskega slikovnega elementa dobimo tako, da delimo uteženo vsoto barvnih komponent z 2^{15} (4.1).

$$Y = (9798 * Rdeča + 18235 * Zelena + 3736 * Modra) / 2^{15} \tag{4.1}$$

Za operacijo množenja sem znotraj programirljivega vezja FPGA uporabil namenske gradnike DSP48A, s čimer sem zmanjšal potrebno število rezin, ki bi sicer bile uporabljene za izvedbo množenja. Uporaba namenskih gradnikov DSP48A za realizacijo različnih matematičnih operacij, kot so seštevanje, odštevanje in množenje, je predvsem primerna za cevovodne strukture, saj se operacije izvršijo v vnaprej določenem številu ciklov sistemske ure. Pri diskretni izvedbi matematičnih operacij znotraj gradnikov SLICE se lahko kaj hitro pripeti, da se zaradi večnivojske logike, s pomočjo katere so realizirane operacije seštevanja, odštevanja in množenja, zakasnitve v vezju povečajo za toliko, da vezje ne deluje pravilno.

4.4 Komunikacija z zunanjim pomnilnikom

Za komunikacijo z zunanjim pomnilnikom sem uporabil knjižnično IP komponento MPMC (angl. Multiport Memory Controller) proizvajalca Xilinx [12]. Zaradi lažje uporabe knjižnične komponente sem napisal lasten uporabniški vmesnik, ki dodatno poenostavi komunikacijski protokol, katerega knjižnična komponenta uporablja za izvajanje operacij z zunanjim pomnilnikom (branje, pisanje). Ker je sam komunikacijski protokol med komponento MPMC in zunanjim pomnilnikom dokaj zahteven, sem z namenom povečanja zmogljivosti pretoka podatkov namesto najmanjšega možnega prenosa, ki je enak eni 32-bitni besedi, uporabil blokovni prenos. Za vsako zahtevo branja iz zunanjega pomnilnika ali pisanja v zunanji pomnilnik sem uporabil blokovni prenos dolžine osmih 32-bitnih besed. MPMC komponenta ima štiri 32-bitna podatkovna vrata, od katerih je dvoje vrat namenjenih branju, preostanek pa pisanju v zunanji pomnilnik. V uporabniški vmesnik sem dodal še dodatne strukture FIFO za pretvorbo bitne širine 8 na 32 ter 32 na 8 bitov. Zajeta slika iz CMOS

slikovnega senzorja se iz RGB formata pretvori v sivinsko sliko, kjer je slikovni element predstavljen z osmimi podatkovnimi biti. Uporabniški vmesnik sem napisal tako, da je branje in pisanje v zunanji pomnilnik za uporabnika precej poenostavljeno in podobno uporabi FIFO struktur. Vmesnik ima dvoje vrat, namenjenih branju, ter dvoje pisanju. Prva izmed vrat, namenjenih pisanju, se uporabljajo za shranjevanje slike v zunanji pomnilnik, ki jo pred samim shranjevanjem filtriramo z Bayerjevim filtrom, nato pa pretvorimo v sivinsko sliko. Prva izmed bralnih vrat so namenjena branju sivinske slike iz zunanjega pomnilnika, ki jo nato filtriramo z uporabo v kaskado vezanih morfoloških filtrov. Sliko filtriramo z morfološkimi filtri in jo nato shranimo nazaj v zunanji pomnilnik s pomočjo drugih izmed pisalnih vrat uporabniškega vmesnika. Za prikaz filtrirane slike na zaslonu je potrebno predhodno obdelano sliko prebrati iz zunanjega pomnilnika in jo prikazati s pomočjo v ta namen izdelanega VGA krmilnika.

4.5 Sinhronizacija pisanja in branja video vsebine iz zunanjega pomnilnika

Ker se hitrosti pisanja in branja v zunanji pomnilnik razlikujeta, je potrebna sinhronizacija. Brez sinhronizacije se namreč zgodi, da se na primer med prikazovanjem slike vsebina pomnilnika lahko spremeni, kar pa na zaslonu opazimo kot popačeno sliko. Na zaslonu je prikazan del predhodne slike, v drugem delu zaslona pa je prikazan že del nove slike. Za sinhronizacijo se ponavadi uporabita dva medpomnilnika (angl. double buffering). To pomeni, da se slika zapisuje v en medpomnilnik, medtem ko se slika bere iz drugega medpomnilnika. Ločimo med tremi možnimi scenariji:

- Ko se operacija branja iz medpomnilnika zaključi, pisanje v drug medpomnilnik pa še ni zaključeno, se novi cikel branja izvrši iz istega medpomnilnika. V primeru, da se je pisanje že izvršilo pred koncem branja, novi cikel pisanja pa se še ni pričel, potem se vlogi medpomnilnikov med seboj zamenjata.
- Ko se operacija pisanja v medpomnilnik zaključi, branje v drug medpomnilnik pa še ni zaključeno, se novi cikel pisanja izvrši ponovno v isti medpomnilnik. Če se je operacija branja že zaključila, novo branje pa se še ni pričelo, se medpomnilnika med seboj zamenjata.
- Če se operaciji pisanja v medpomnilnik in branja iz medpomnilnika zaključita

istočasno, se v naslednjem ciklu medpomnilnika zamenjata. Pisanje novih podatkov se izvrši v medpomnilnik, iz katerega smo prej vršili branje in obratno. Branje novih podatkov se izvrši iz medpomnilnika, v katerega smo prej izvršili pisanje podatkov.

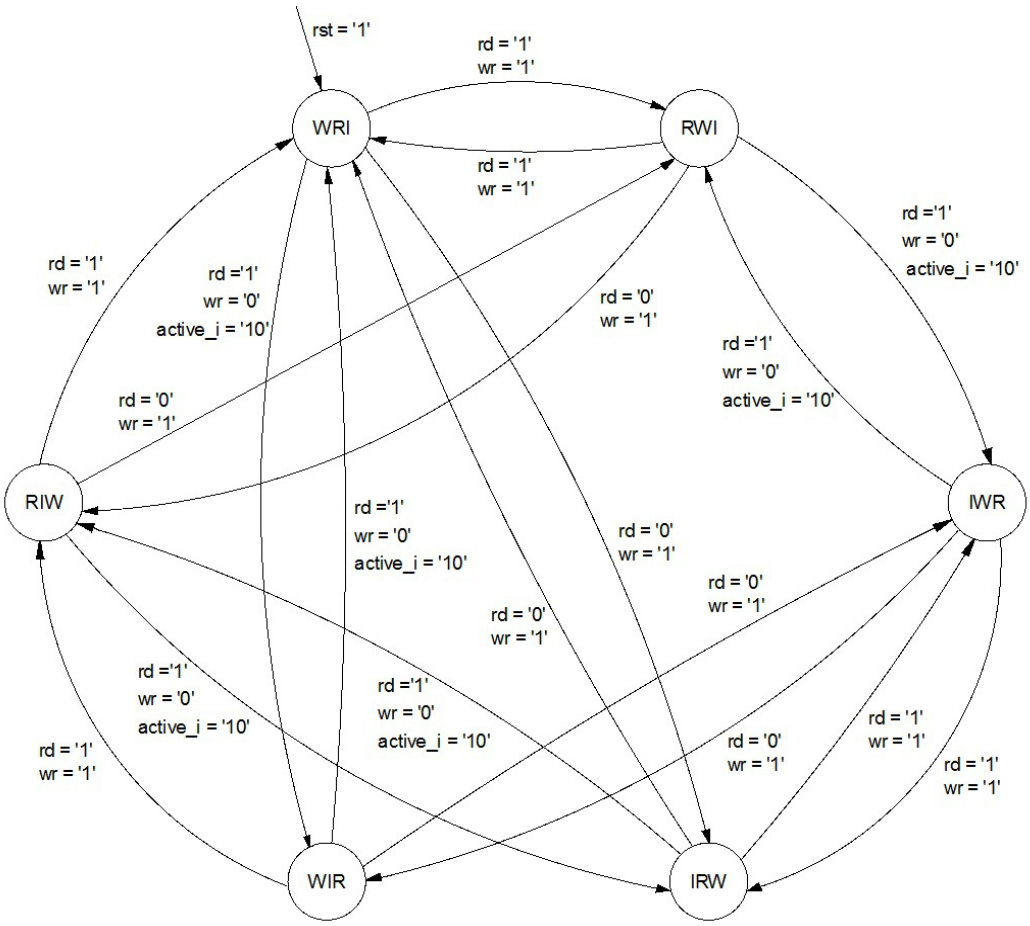
Tovrstna sinhronizacija ni najpopolnejša, saj je težko uskladiti hitrosti pisanja in branja, vendar se efekt popačenja slike oziroma verjetnost za njen pojav zmanjša. Izboljšana metoda sinhronizacije uporablja še tretji medpomnilnik (angl. triple buffering). Pri tem načinu je eden izmed treh pomnilnikov zmeraj na voljo za pisanje, branje pa se vrši iz zadnjega pomnilnika, v katerega je bila slika uspešno zapisana. Na tak način lahko popolnoma uspešno sinhroniziramo različne hitrosti med branjem in pisanjem. Metodo trojnega medpomnjenja sem kot sinhronizacijski mehanizem uporabil tako na vhodu kot tudi izhodu digitalnega sistema. Pri metodi trojnega medpomnjenja ločimo med tremi scenariji:

- Branje iz medpomnilnika se zaključi, pisanje pa še traja. Če je zadnji medpomnilnik, ki je bil uporabljen za zapis slike prav ta, iz katerega smo pravkar prebrali celotno sliko, potem se naslednji cikel branja izvrši iz istega medpomnilnika. Če je med operacijo branja bila nova slika zapisana v drugega izmed medpomnilnikov, ki ga nismo brali v zadnjem ciklu, potem se bralni medpomnilnik spremeni. Predhodno uporabljen medpomnilnik se tako sprosti in je na voljo za pisanje, branje pa se vrši iz medpomnilnika, v katerega smo nazadnje pisali.
- Pisanje v medpomnilnik se zaključi, branje pa še traja. Vedno imamo na voljo en prosti medpomnilnik, ki ga uporabimo za zapis nove slike. Predhodno uporabljen medpomnilnik označimo kot medpomnilnik, v katerega je bila v celoti shranjena zadnja slika. Ta podatek je potreben ob zaključku bralnega cikla, kjer se odločitev o zamenjavi bralnega pomnilnika navezuje prav na to, ali obstaja v katerem izmed medpomnilnikov novejša slika.
- Operaciji pisanje in branja se zaključita istočasno. V tem primeru se bralni in pisalni medpomnilnik pravkar zaključenih operacij enostavno le zamenjata.

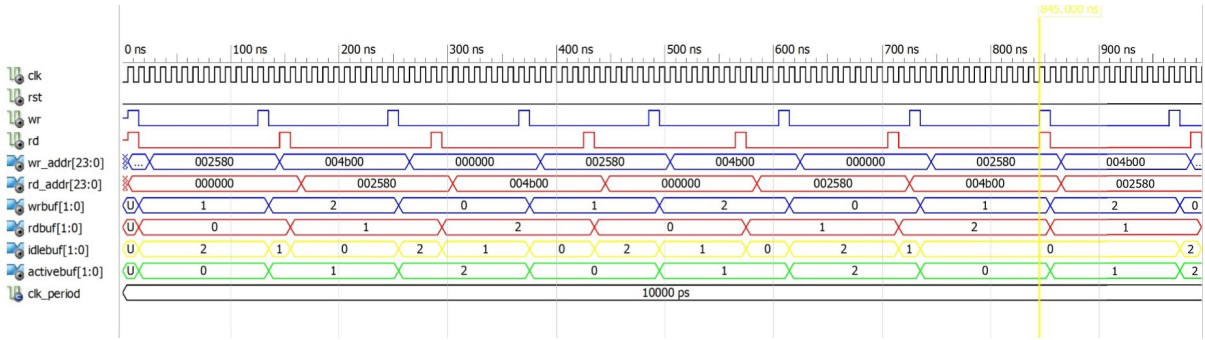
Za izvedbo ustreznega sinhronizacijskega mehanizma časovnega preklapljanja med bralnim, pisalnim in prostim medpomnilnikom sem uporabil končni avtomat prehajanja stanj (slika 4.2). Vsako izmed stanj nam pove, kateri izmed treh medpomnilnikov se trenutno uporablja za pisanje, kateri za branje ter kateri je trenutno prost. Tako na primer stanje WRI pomeni, da je prvi medpomnilnik namenjen pisanju (angl. Write), durgi branju (angl. Read), tretji pa je

prost (angl. Idle). Vhodna krmilna signala wr in rd krmilita preklope med stanji avtomata. Aktivno stanje vhodnega krmilnega signala wr je logična enica in pomeni, da se je pisanje v trenutni medpomnilnik zaključilo. Prav tako je aktivno stanje vhodnega krmilnega signala rd stanje logične enice in ponazarja zaključek bralnega cikla iz medpomnilnika. Prehod v nova stanja končnega avtomata je v določenih stanjih avtomata odvisen tudi od zadnjega aktivnega medpomnilnika. To je medpomnilnik, v katerega je bila nazadnje uspešno zapisana slika. V nadaljevanju je prikazan poenostavljen diagram prehajanja stanj avtomata (slika 4.2) in deklaracija komponente v visokonivojskem strojno opisnem jeziku VHDL, ki prikazuje vhodne in izhodne signale komponente. Podobno lahko izluščimo informacijo o stanju določenega medpomnilnika še za preostlih pet stanj avtomata. Vsaki instanci komponente buf_sched lahko s pomočjo treh generičnih konstant določimo naslove treh video medpomnilnikov, ki jih želimo uporabiti. Naslovi medpomnilnikov morajo biti poravnani na osem 32-bitnih besed, torej na 32 bajtov. Pri izdelavi digitalnega sistema sem uporabil eno komponento za sinhronizacijo na vhodu, drugo pa na izhodu. V ta namen sem uporabil šest medpomnilnikov, po tri za vsako komponento. Privzeti naslovi treh medpomnilnikov prve sinhronizacijske komponente so prednastavljeni v izseku izvorne kode, napisane v visokonivojskem strojno opisnem jeziku VHDL (deklaracija komponente). Velikost enega medpomnilnika je enaka velikosti sivinske slike ločljivosti 640x480 slikovnih elementov (307200 bajtov). Za medpomnjenje zajetih in obdelanih slik sem tako v zunanjem pomnilniku porabil le slaba 2 MB od 128 MB, ki so na voljo.

```
entity buf_sched is
        generic (
                C_BUF0_ADDR : std_logic_vector (23 downto 0) := x"000000";
                C_BUF1_ADDR : std_logic_vector (23 downto 0) := x"002580";
                C_BUF2_ADDR : std_logic_vector (23 downto 0) := x"004b00"
        );
        port (
                clk         : in  STD_LOGIC;
                rst         : in  STD_LOGIC;

                wr          : in  STD_LOGIC;
                rd          : in  STD_LOGIC;

                wr_addr     : out  STD_LOGIC_VECTOR (23 downto 0);
                rd_addr     : out  STD_LOGIC_VECTOR (23 downto 0);
                wrbuf       : out  STD_LOGIC_VECTOR (1 downto 0);
                rdbuf       : out  STD_LOGIC_VECTOR (1 downto 0);
                idlebuf     : out  STD_LOGIC_VECTOR (1 downto 0);
                activebuf : out  STD_LOGIC_VECTOR (1 downto 0)
        );
end buf_sched;
```



Slika 4.2: Poenostavljen diagram prehajanja stanj avtomata komponente buf_sched



Slika 4.3: Časovni potek uporabe medpomnilnikov komponente buf_sched za sinhronizacijo branja in pisanja video vsebine v zunanji pomnilnik

Slika 4.3 prikazuje rezultate simulacije, v kateri sem uporabil periodo bralnega cikla 140 ns, periodo pisalnega cikla pa 120 ns. Frekvenca pisalnega cikla za izbrano simulacijo je nekoliko večja od frekvence bralnega cikla. V simulaciji (slika 4.3) lahko vidimo tudi zahtevo po sočasnem preklopu med bralnim in pisalnim medpomnilnikom ob času 845 ns (slika 4.3). Z modro barvo so na sliki 4.3 označeni vhodni krmilni signal (wr), ki označuje konec pisalnega cikla, naslov medpomnilnika, v katerega se vrši pisanje (wr_addrs) ter indeks trenutno

uporabljenega medpomnilnika (wrbuf). Z rdečo barvo so označeni pripadajoči signali bralnega cikla rd, rd_addr in rdbuf. Z rumeno barvo je označen signal idlebuf, ki predstavlja indeks trenutno neuporabljenega izmed treh medpomnilnikov. Iz časovnega poteka je razvidno, kako se ob vsakem koncu bralnega cikla, pisalnega cikla ali obeh hkrati spremeni indeks medpomnilnika, ki je prost za naslednji pisalni cikel. To seveda velja le v primeru, da le-ta ni bil uporabljen v zadnjem pisalnem ciklu, torej ni označen kot aktiven medpomnilnik. Indeks medpomnilnika activebuf, ki je bil uporabljen v zadnjem pisalnem ciklu, je na sliki 4.3 označen z zeleno barvo.

4.6 VGA krmilna enota

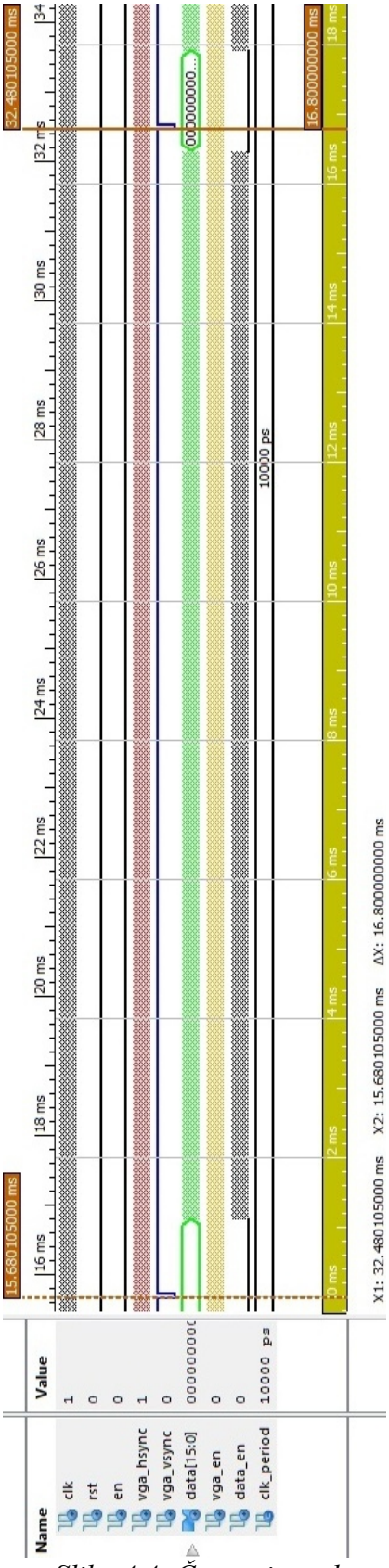
VGA krmilna enota generira potrebni vertikalni in horizontalni sinhronizacijski signal (VSYNC, HSYNC) ter bere vrednosti slikovnih elementov iz zunanjega pomnilnika za vsako izhodno sliko videa. Izhodi treh barvnih komponent so speljani na tri uporovne digitalno-analogne pretvornike. Za rdečo in modro barvno komponento sem uporabil dva 5-bitna uporovna digitalno-analogna pretvornika, za zeleno barvno komponento pa en 6-bitni uporovni digitalno-analogni pretvornik. Ker je video vsebina, ki jo obdelujemo z morfološkimi operacijami sivinska slika, je potrebno znotraj VGA krmilne enote 8-bitne sivinske vrednosti pretvoriti v RGB565 format. Izhodni del digitalnega sistema za prikaz na zunanjem zaslonu sem načrtal tako, da omogoča prikaz barvne video vsebine. Sama pretvorba 8-bitnih vrednosti slikovnih elementov v RGB565 format je precej enostavna in poteka tako, da opustimo vse bite nižjih utežnih vrednosti. Pri pretvorbi svetilnosti Y v rdečo in modro barvno komponento sem zanemaril spodnje tri bite, pri pretvorbi v zeleno barvno komponento pa spodnja dva bita.

$$R(4:0) = Y(7:3)$$

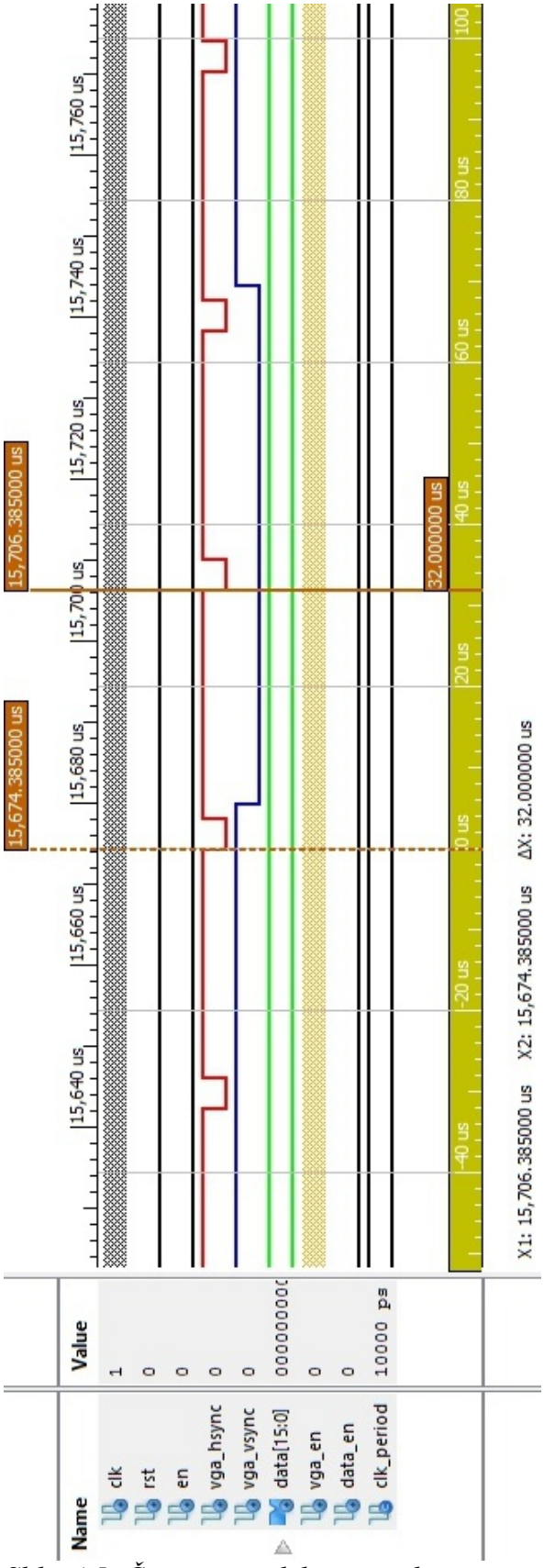
$$G(5:0) = Y(7:2)$$

$$B(4:0) = Y(7:3)$$

Iz rezultatov simulacije VGA krmilne enote za vertikalni sinhronizacijski signal vga_vsync je iz pozicije kontrolnikov na sliki 4.4 razvidno, da je perioda osveževanja zaslona 16.8 ms, kar znaša približno 59.52 Hz. Iz rezultatov simulacije VGA krmilne enote za horizontalni sinhronizacijski signal vga_hsync je iz pozicije kontrolnikov na sliki 4.5 razvidno, da je perioda horizontalnega sinhronizacijskega signala 32 μs, kar je enako 31.25 kHz.



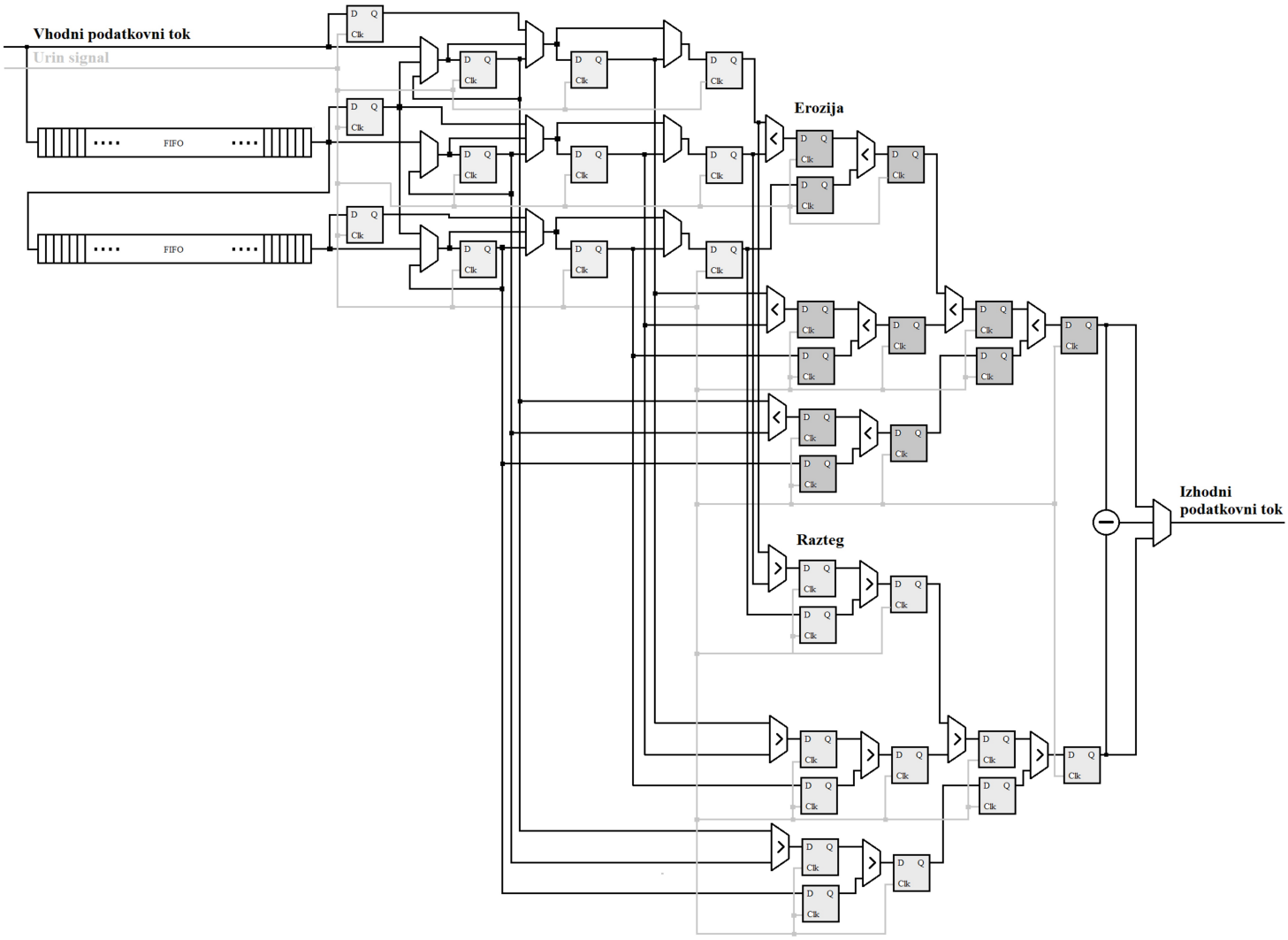
Slika 4.4: Časovni potek vertikalnega sinhronizacijskega signala VGA krmilnika



Slika 4.5: Časovni potek horizontalnega sinhronizacijskega signala VGA krmilnika

4.7 Implementacija morfološkega filtra

Spodnja slika 4.6 predstavlja nekoliko poenostavljeno zasnovo morfološkega filtra znotraj programirljivega vezja FPGA. Na vhodu sta dva medpomnilnika FIFO, ki sta namenjena začasnemu shranjevanju podatkovnega toka vhodne slike, potrebna za izvajanje filtriranja. Medpomnilnikoma sledi devet 8-bitnih registrov, ki predstavljajo okno velikosti 3x3 slikovnih elementov morfološkega filtra. Trije dodatni registri pred oknom služijo za začasno shranjevanje elementov iz medpomnilnika za cikel ure, v katerem obdelujemo desni rob slike, da nam vhodnih podatkov nove vrstice slike ni potrebno zaustavljati. Vrednosti na izhodu morfološkega filtra so glede na vhod zakasnjene za čas, ki je potreben za prehod slikovnih elementov skozi prvega izmed medpomnilnikov, kateremu je potrebno prišteti še zakasnitev cevovodne strukture primerjalnikov. Slika 4.6 je le idejna zasnova.



Slika 4.6: Poenostavljana zgradba morfološkega filtra

Za izvedbo osnovnih morfoloških operacij raztega in erozije je potrebno poiskati ali lokalni maksimum ali lokalni minimum znotraj okna, ki določa velikost strukturnega elementa. V splošnem je lahko okno in s tem posledično strukturni element poljubne velikosti. Za dano izvedbo filtra sem uporabil strukturni element velikosti 3x3 slikovnih elementov. Ker iščemo lokalni minimum ali lokalni maksimum ali v določenih primerih oboje, pomeni, da je potrebno vhodne slikovne elemente urediti po velikosti od najmanjšega do največjega. Problem iskanja lokalnega minimuma ali maksimuma znotraj okna lahko torej rešimo z urejanjem vhodnih slikovnih elementov po velikosti. Obstaja več različnih algoritmov, s pomočjo katerih lahko med seboj po velikosti uredimo množico elementov. Nekaj najpogosteje uporabljenih algoritmov urejanja:

- mehurčno urejanje (angl. bubble sort)
- urejanje z združevanjem elementov sodih in lihih indeksov (angl. even-odd merge sort)
- urejanje z združevanjem elementov lihih in sodih indeksov (angl. odd-even merge sort)
- bitonično urejanje z združevanjem (angl. bitonic merge sort)

Metode urejanja elementov se med seboj razlikujejo predvsem po številu operacij, ki so potrebne, da dano množico vhodnih elementov uredimo po velikosti od najmanjšega do največjega. Za podano izvedbo morfološkega filtra velja, da ko je množica vhodnih elementov urejena, imamo na voljo tako lokalni minimum, lokalni maksimum, kakor tudi mediano znotraj opazovanega okna. Poudariti je treba, da median filter ne sodi med morfološke filtre, ampak sem ga dodal kot eno izmed funkcij v sam filter zaradi lažje povezljivosti filtrov med seboj ter izvajanja operacij med filtri. Delno urejanje elementov, kjer bi iskali le minimum, maksimum ali oboje, bi lahko precej poenostavil, vendar bi s tem izgubil generičnost v smislu kombiniranja filtrov in povezovanja le-teh v kaskado ali vzporedno več kaskad zaradi različnih zakasnitev skozi posamezne filtre. V ta namen sem izdelal komponento, ki uredi množico devetih vhodnih elementov. Ker ima lahko strukturni element poljubno obliko znotraj danega okna, je bila potrebna rešitev, s pomočjo katere primerjamo le slikovne elemente, ki jih strukturni element morfološke operacije znotraj okna pokrije. Prva izvedba morfološkega filtra je poleg osmih podatkovnih bitov vsebovala še en podatkovni bit, ki je bil enak logični enici za vsak element, ki ga je okno strukturnega

elementa za trenutno opazovani vhodni slikovni element pokrilo in logični ničli za preostale slikovne elemente. Tako sem po sortiranju vhodnih elementov dobil na izhodu filtra vrednosti, ki so bile razvrščene od najmanjšega do največjega elementa. Nato je bilo potrebno ugotoviti, katere izmed vrednosti so veljavne glede na obliko strukturnega elementa. Z opisom v visokonivojskem strojno opisnem jeziku VHDL, ki bi iz množice veljavnih slikovnih elementov določil minimum, maksimum in mediano, je bilo precej težav. Vsi preizkušeni zapisi so privedli do prioritetnega dekodirnika, ki je zaradi uporabe večnivojske logike v vezje vnesel neželene časovne zakasnitve. Kot rezultat sem na izhodu filtra dobil precejšen šum za veliko število slikovnih elementov. Potrebno je bilo poiskati alternativno rešitev za dani problem. V naslednji implementaciji sem uporabil tri namesto ene same sortirne komponente. Pri iskanju lokalnega minimuma sem vhodne vrednosti, ki jih okno filtra ni pokrivalo in zato niso bili del vhodne množice, preslikal v maksimalno vrednost 8-bitnega števila, torej v 255. Na ta način sem zagotovil, da so na izhodu dotične sortirne komponente med najmanjšimi urejenimi elementi le tisti, ki jih je strukturni element morfološkega filtra znotraj okna pokril. Pri iskanju lokalnega maksimuma sem uporabil podobno rešitev, le da sem tokrat vse vhodne elemente, ki jih strukturni element morfološkega filtra znotraj okna ni pokril, preslikal v najmanjši element iz množice 8-bitnih vrednosti, torej v 0. Na izhodu te druge sortirne komponente sem tako zagotovil, da je bil maksimum zmeraj element iz množice, ki jih je strukturni element morfološkega filtra znotraj okna pokril. Tretja sortirna komponenta je bila namenjena iskanju mediane med vsemi devetimi vhodnimi elementi, zato modifikacija vhodnih vrednosti ni bila potrebna. Uporabljena rešitev sicer zavzame več prostora znotraj programirljivega vezja FPGA, vendar zagotavlja časovno neodvisnost izhoda filtra od oblike strukturnega elementa.

Da bi bil morfološki filter čim bolj uporaben v smislu enostavnega povezovanja več filtrov med seboj v kaskado ali paralelno, sem izdelal komponento, ki nudi štiri osnovne funkcije filtra:

- prepust
- mediana
- razteg
- erozija

Poleg zgoraj naštetih osnovnih funkcij nudi filter možnost izvajanja matematičnih operacij med zgoraj naštetimi funkcijami. Ker se vse štiri osnovne funkcije morfološkega filtra izvajajo paralelno in neodvisno od izbrane končne funkcije morfološkega filtra, lahko z enim samim filtrom realiziramo morfološki operator gradienta. S stališča porabe prostora znotraj programirljivega vezja FPGA zasnova filtra ni najbolj optimalna, vendar nudi večje možnosti v smislu spreminjanja izhodne funkcije filtra v realnem času, še posebej v primerih, če filtre vežemo v kaskado ali uporabimo paralelno več kaskad filtrov za izvedbo želenih sestavljenih morfoloških operatorjev. Poleg podatkovnih vhodov in krmilnih vhodov za izbiro izhodne funkcije morfološkega filtra vsebuje komponenta dodaten podatkovni vhod, s pomočjo katerega določimo obliko strukturnega elementa morfološkega filtra.

4.7.1 Prepust

V načinu prepusta filter ne spreminja vhodnih vrednosti, kar pomeni, da je izhodna vrednost filtra enaka kar središčnemu slikovnemu elementu znotraj opazovanega okna morfološkega filtra. Središčni slikovni element je namreč tisti, za katerega se vrši funkcija filtriranja. Namen funkcije prepusta je le zakasnitev vhodnih slikovnih elementov, kar omogoča, da na izhodu filtra izvršimo želeno operacijo med neobdelanimi slikovnimi elementi ter elementi, ki smo jih z izbrano operacijo modificirali. Ker za prepustni filter ne potrebujemo primerjalnikov, je bilo potrebno slikovne elemente le zakasniti z uporabo registrov za število ciklov, ki so potrebni za izvršitev ene izmed preostalih funkcij filtra: median, razteg, erozija. Slikovni elementi iz medpomnilnika FIFO na vhodu propagirajo skozi 8-bitne registre, s pomočjo katerih sem realiziral ustrezno zakasnitev na izhodu, ki je ekvivalentna zakasnitvi elementov skozi urejevalnik. Vhodna vrednost na ta način prehaja skozi filter z enako zakasnitvijo, kot urejena množica vhodnih elementov v vseh treh urejevalnih komponentah morfološkega filtra.

4.7.2 Mediana

Median filter je namenjen odpravljanju šuma v sliki. Uporabljamo ga največkrat na samem vhodu, torej pred uporabo ostalih filtrov. Mediano dobimo tako, da vhodne elemente uredimo po velikosti od najmanjšega pa do največjega. Mediana je enaka središčnemu izmed devetih po velikosti urejenih elementov na izhodu urejevalne komponente pri velikosti okna morfološkega filtra 3x3 slikovnih elementov. Za izvršitev funkcije sem uporabil eno sortirno

komponento.

4.7.3 Razteg

Funkcija raztega med vhodnimi elementi da na izhodu maksimalno izmed vhodnih vrednosti, ki jih strukturni element znotraj okna morfološkega filtra pokrije. Za izvedbo je bila potrebna ena sortirna komponenta.

4.7.4 Erozija

Funkcija erozije nam da na izhodu minimalno vrednost iz množice vhodnih elementov, ki jih strukturni element znotraj okna morfološkega filtra pokrije. Kot pri raztegu je bila tudi tukaj potrebna ena sortirna komponenta.

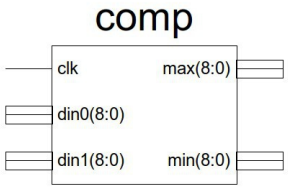
4.7.5 Implementacija morfološkega filtra v visokonivojskem strojno opisnem jeziku VHDL

Osnovo morfološkega filtra predstavlja komponenta sorter_pipeline_3x3 ter kombinacijska logika, ki omogoča izbiro ene izmed osnovnih ali kombinirano izhodno funkcijo morfološkega filtra. Komponento sorter_pipeline_3x3 sestavljajo tri komponente sorter_3x3, vsaka izmed njih pa je nadalje sestavljena iz dveh komponent oem_sort in ene primerjalne komponente. Kot osnovo urejanja vhodnih elementov sem uporabil Batcherjev algoritem [21] urejanja z združevanjem elementov lihih in sodih indeksov (angl. odd-even merge sort). Ker algoritem deluje nad sodim številom vhodnih elementov 2^N, sem ga priredil problemu urejanja lihega števila vhodnih elementov in v ta namen razdelil na tri dele (slika 4.13). V prvem delu komponenta oem_sort uredi osem od devetih vhodnih elementov. Nato se izvrši primerjava med najmanjšim izmed urejenih elementov urejevalnika in devetim vhodnim elementom. Če je deveti element večji, se izvrši zamenjava osmega in devetega elementa znotraj primerjalnika ob fronti ure. Na mesto najmanjšega elementa že urejene množice osmih vhodnih elementov pride večji izmed elementov dodatne primerjave, izvedene z dvovhodnim primerjalnikom. Ker pravkar zamenjani element ni nujno najmanjši glede na predhodno že po velikosti urejenih sedem elementov, izvršimo še eno dodatno urejanje s pomočjo druge komponente oem_sorter. Po končanem urejanju je vseh devet vhodnih elementov urejenih od najmanjšega do največjega. V primeru, ko je deveti vhodni element že manjši od najmanjšega iz množice osmih urejenih vhodnih elementov, je vso nadaljnje

urejanje redundantno. Redundanco bi lahko odpravil z določeno logiko, vendar bi v določenih primerih izgubil generičnost filtra predvsem, ko so filtri vezani vzporedno. Izhod enega izmed filtrov bi lahko bil zaradi že urejene množice po prvem urejanju časovno zamaknjen glede na drug izhod filtra. Izvajanje kombiniranih operacij obeh filtrov bi bilo v takšnem primeru precej oteženo. Prav zaradi tega sem se odločil, da je redundanca in s tem nekaj večja zasedenost vezja FPGA vseeno upravičena, saj je medsebojno povezovanje filtrov enostavno in ni potrebna dodatna sinhronizacijska logika. Osnovni gradniki komponente oem_sorter so v mrežo povezani dvovhodni primerjalniki, ki imajo po dva izhoda. Na enem izhodu se po primerjavi sinhrono s ciklom ure pojavi večja izmed primerjanih vhodnih vrednosti, na drugem izhodu pa manjša izmed vhodnih vrednosti. Komponenta torej na svojih izhodih da kot rezultat primerjave na en izhod večjega ter na drugi izhod manjšega izmed elementov. Na ta način se znotraj komponente oem_sorter izvrši primerjava komponent, elementa na izhodu pa sta po velikosti urejena. Slika 4.10 prikazuje, kako je za uporabljeni algoritem urejanja 8-bitnih elementov z združevanjem lihih in sodih indeksov potrebno povezati vhodne elemente ter elemente vseh preostalih notranjih primerjav, ki se v komponenti vršijo sinhrono s sistemsko uro, da na izhodu komponente dobimo po šestih ciklih ure po velikosti od najmanjšega pa do največjega elementa urejeno množico vhodnih elementov. Iz množice urejenih elementov so na izhod povezani le trije izmed devetih elementov, in sicer: minimum, median ter maksimum. To so namreč vrednosti, ki jih potrebujemo za realizacijo želene funkcije morfološkega filtra.

4.7.5.1 Dvovhodni 8-bitni sinhroni primerjalnik

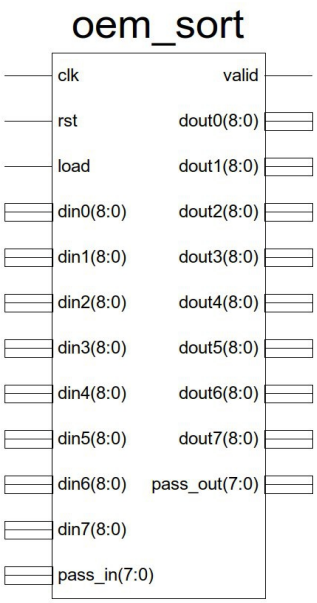
Dvovhodni primerjalnik je najprimitivnejši gradnik morfološkega filtra. Ob vsaki pozitivni fronti sistemske ure se rezultat primerjave med vhodnima elementoma din0 in din1 shrani v dva izhodna registra. Manjšega izmed vhodnih elementov shranimo v izhodni register min, večjega pa v izhodni register max.



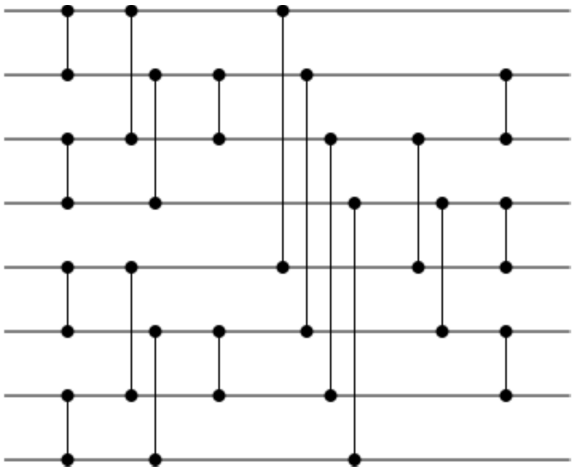
Slika 4.7: Dvovhodni 8-bitni primerjalnik

4.7.5.2 Osemvhodni urejevalnik z združevanjem 8-bitnih elementov lihih in sodih indeksov

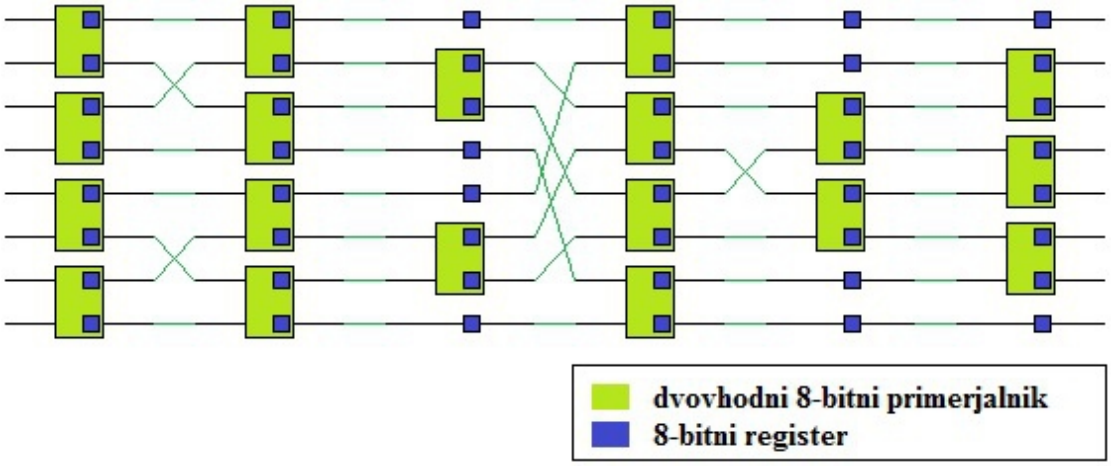
Osemvhodni urejevalnik z združevanjem elementov lihih in sodih indeksov (angl. odd-even merge sorter) ob vsaki pozitivni fronti sistemske ure shranjuje rezultate primerjave dvovhodnih primerjalnikov, ki so osnovni gradniki komponente. Komponenta ima cevovodno strukturo (slika 4.10), kar omogoča delovanje tudi pri višjih frekvencah sistemske ure, saj so zakasnitve med registri, v katere se shranjujejo rezultati primerjav s tovrstnim načinom, minimalne. Med registri se namreč nahajajo le dvovhodni 8-bitni primerjalniki. Signal load služi za indikacijo veljavnosti vhodnih signalov oziroma elementov. Tako kot vhodni elementi, tudi signal load napreduje po cevovodni strukturi sinhrono s sistemsko uro skozi vmesne registre, ki signal časovno uskladijo z notranjimi primerjalniki. S tem dosežemo, da ko signal load prepotuje skozi vse vmesne registre, služi kot statusni signal, ki označuje veljavnost izhoda urejevalnika. Komponento sestavlja 6-stopenjski cevovod, kar pomeni, da je za urejanje osmih elementov potrebnih šest ciklov ure, kar je tudi enako časovni zakasnitvi vhodnih elementov pri prehodu skozi urejevalnik. Poleg osmih podatkovnih vhodov ima komponenta še dodaten vhod pass_in, kamor lahko pripeljemo kakršenkoli signal, ki ga želimo sinhronizirati z izhodom urejevalnika. Vhodni element pass_in se namreč časovno zakasnjen pojavi na izhodu sinhrono z urejeno množico vhodnih elementov. Veljavnost izhoda primerjalnika določa izhodni signal valid. Stanje logične enice na izhodu valid pomeni, da je množica elementov na izhodih dout0–dout7 ter pass_out veljavna. Ta signal je potreben za povezavo izhoda primerjalnika in na primer naslednje komponente.



Slika 4.8: Osemvhodni urejevalnik 8-bitnih elementov



Slika 4.9: Batcherjev algoritem urejanja na primeru osmih elementov z združevanjem elementov lihih in sodih (angl. odd-even merge sort) [23]

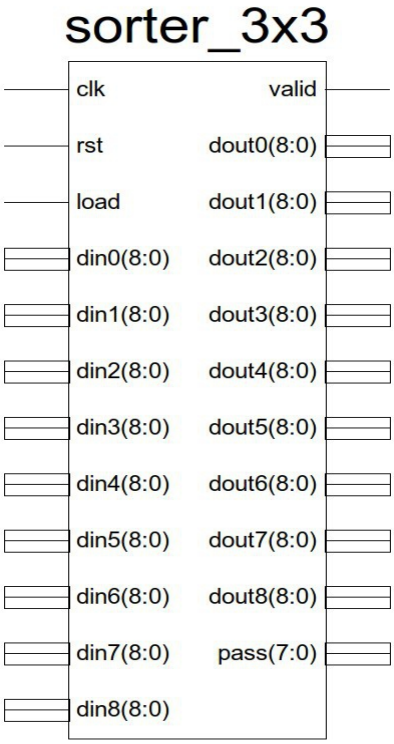


Slika 4.10: Prikaz 6-stopenjske cevovodne strukture in mreže povezav primerjalnikov za urejanje z združevanjem elementov lihih in sodih indeksov

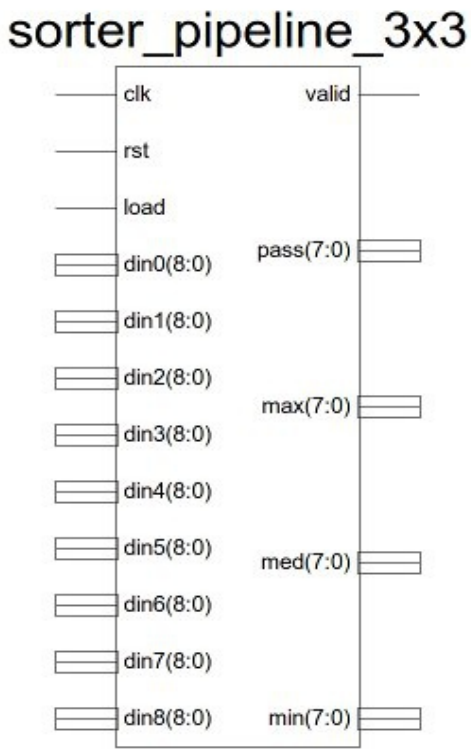
4.7.5.3 Devetvhodni urejevalnik 8-bitnih elementov

Urejevalnik sestavljata dve komponenti oem_sorter in dvovhodni primerjalnik comp. Vhodni signali v urejevalnik so 9-bitni, čeprav so vrednosti, ki jih želimo urediti med seboj 8-bitne. Deveti bit služi kot indikator in pove, ali za dano okno velikosti 3x3 strukturni element pokrije dani vhodni element ali ne. Če strukturni element morfološkega filtra vhodnega

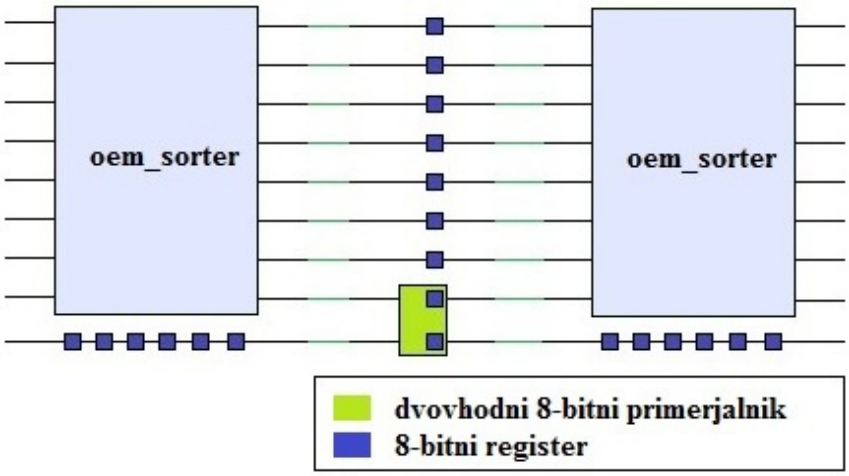
elementa ne pokrije, je deveti bit enak 0 in pomeni, da ta element ne bo upoštevan pri rezultatu primerjave na izhodu filtra, ko bo izbrana funkcija filtra samostojna, ali v kombinaciji s funkcijo lokalnega minimuma ali lokalnega maksimuma znotraj opazovanega okna. V nasprotnem primeru, ko je deveti bit enak ena, se element pri primerjavi upošteva. V končni izvedbi filtra zaradi že omenjenih težav deveti bit ni uporabljen in je v resnici redundanten. Komponente nisem spreminjal z namenom, da bi lahko v okviru nadaljnjega dela filter modificiral tako, da bi deveti bit uporabil kot indikator veljavnosti pri primerjavi.



Slika 4.11: Devetvhodni urejevalnik 8-bitnih elementov



Slika 4.12: Devetvhodni urejevalnik z uporabo morfološkega strukturnega elementa



Slika 4.13: Prikaz 13-stopenjske cevovodne strukture devetvhodnega urejevalnika 8-bitnih elementov

4.7.5.4 Devetvhodni urejevalnik 8-bitnih elementov z uporabo morfološkega strukturnega elementa

Ob poskusu implementacije morfološkega filtra le z eno komponento sorter_3x3 v visokonivojskem strojno opisnem jeziku VHDL sem naletel na težavo pri izločanju izhodnih elementov urejevalnika, ki zaradi oblike strukturnega elementa morfološkega filtra znotraj okna 3x3 ne sodijo v rezultat primerjave. Določitev izhodnih vrednosti min, max ter median z uporabo izbiralnikov ni dala želenih rezultatov. S pomočjo devetega bita znotraj komponente sorter_3x3 nisem uspel implementirati filtra, ki bi deloval pravilno, saj so vsi preizkušeni dekodirniki, ki bi izločili iz urejene množice le želene elemente, vnesli v vezje prevelike časovne zakasnitve, ki so se v sliki manifestirale kot močan temen šum v različnih področjih slike. Izhod filtra je bil zaradi časovne odvisnosti večnivojske kombinacijske logike dekodirnika zato neuporaben za obdelavo slike. Najti je bilo potrebno alternativno rešitev.

Komponenta sorter_pipeline_3x3 na sliki 4.12 je alternativa prvotni implementaciji. V komponento sorter_pipeline_3x3 sem združil vzporedno tri komponente sorter_3x3. Dejstvo je, da pri iskanju lokalnega maksimuma znotraj okna 3x3 lahko vhodne vrednosti, ki jih strukturni element morfološkega filtra ne pokrije, preslikamo tako, da njihove vrednosti nadomestimo z najmanjšo možno vrednostjo 8-bitnega števila, torej z nič, preden jih pričnemo urejati. Na ta način dobimo na izhodu urejevalnika sorter_3x3 urejeno množico

elementov, od katerih je element z največjo vrednostjo zagotovo tak, da sovpada s strukturnim elementom morfološkega filtra. Podobno velja za operacijo iskanja lokalnega minimuma, ko podmnožico vhodnih elementov, ki jih strukturni element ne pokrije, preslikamo v maksimalno vrednost 8-bitnega števila, torej v 255, pred urejanjem. Na izhodu tako dobimo za minimalno vrednost zagotovo element, ki sovpada s strukturnim elementom morfološkega filtra.

V nadaljevanju je prikazana izvorna koda, napisana v visokonivojskem strojno opisnem jeziku VHDL za devetvhodni sortirnik v cevovodni strukturi, ki v sklopu primerjav uporablja strukturni element. Vhodne slikovne elemente, ki niso podmnožica strukturnega elementa s predhodno opisanima postopkoma preslikave glede na izbrano operacijo, izločimo iz postopka sortiranja. Tako podmnožico vhodnih vrednosti elementov, ki jih strukturni element ne prekriva za dani centralni vhodni element, postavimo na 0 v postopku iskanja maksimuma ter na maksimalno vrednost 255 v postopku iskanja minimuma.

```
library IEEE;
use IEEE.std_logic_1164.ALL;

entity sorter_pipeline_3x3 is
	generic (
		C_DATA_WIDTH : integer := 8
	);
	port (
		clk	: in  std_logic;
		 rst	: in  std_logic;

		load	: in  std_logic;
		din0	: in  std_logic_vector (C_DATA_WIDTH downto 0);
		din1	: in  std_logic_vector (C_DATA_WIDTH downto 0);
		din2	: in  std_logic_vector (C_DATA_WIDTH downto 0);
		din3	: in  std_logic_vector (C_DATA_WIDTH downto 0);
		din4	: in  std_logic_vector (C_DATA_WIDTH downto 0);
		din5	: in  std_logic_vector (C_DATA_WIDTH downto 0);
		din6	: in  std_logic_vector (C_DATA_WIDTH downto 0);
		din7	: in  std_logic_vector (C_DATA_WIDTH downto 0);
		din8	: in  std_logic_vector (C_DATA_WIDTH downto 0);

		pass	: out std_logic_vector ((C_DATA_WIDTH - 1) downto 0);
		max	: out std_logic_vector ((C_DATA_WIDTH - 1) downto 0);
		med	: out std_logic_vector ((C_DATA_WIDTH - 1) downto 0);
		min	: out std_logic_vector ((C_DATA_WIDTH - 1) downto 0);
		valid : out std_logic
	);
end sorter_pipeline_3x3;
```

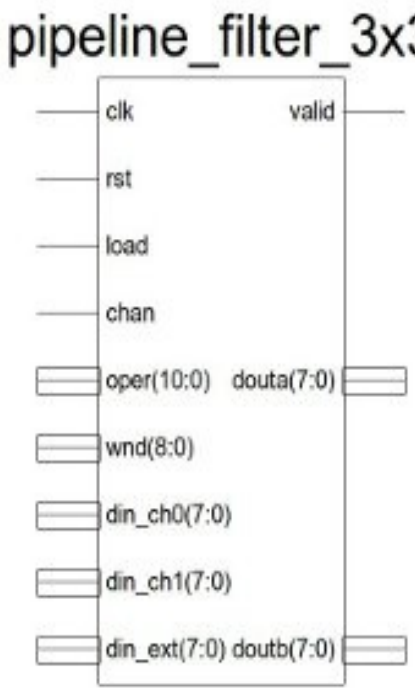
```
architecture Behavioral of sorter_pipeline_3x3 is
	component sorter_3x3
		generic (
			C_DATA_WIDTH : integer := 8
		);
		port (
			clk	: in  std_logic;
			rst	: in  std_logic;
			load	: in  std_logic;
			din0	: in  std_logic_vector (C_DATA_WIDTH downto 0);
			din1	: in  std_logic_vector (C_DATA_WIDTH downto 0);
			din2	: in  std_logic_vector (C_DATA_WIDTH downto 0);
			din3	: in  std_logic_vector (C_DATA_WIDTH downto 0);
			din4	: in  std_logic_vector (C_DATA_WIDTH downto 0);
			din5	: in  std_logic_vector (C_DATA_WIDTH downto 0);
			din6	: in  std_logic_vector (C_DATA_WIDTH downto 0);
			din7	: in  std_logic_vector (C_DATA_WIDTH downto 0);
			din8	: in  std_logic_vector (C_DATA_WIDTH downto 0);
			dout0	: out std_logic_vector (C_DATA_WIDTH downto 0);
			dout1	: out std_logic_vector (C_DATA_WIDTH downto 0);
			dout2	: out std_logic_vector (C_DATA_WIDTH downto 0);
			dout3	: out std_logic_vector (C_DATA_WIDTH downto 0);
			dout4	: out std_logic_vector (C_DATA_WIDTH downto 0);
			dout5	: out std_logic_vector (C_DATA_WIDTH downto 0);
			dout6	: out std_logic_vector (C_DATA_WIDTH downto 0);
			dout7	: out std_logic_vector (C_DATA_WIDTH downto 0);
			dout8	: out std_logic_vector (C_DATA_WIDTH downto 0);
			pass	: out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
			valid	: out std_logic
		);
	end component;

	signal pass_i : std_logic_vector (C_DATA_WIDTH-1 downto 0);
	signal min_i, med_i, max_i : std_logic_vector (C_DATA_WIDTH downto 0);
	signal min0_i, min1_i, min2_i, min3_i, min4_i : std_logic_vector (C_DATA_WIDTH downto 0);
	signal min5_i, min6_i, min7_i, min8_i : std_logic_vector (C_DATA_WIDTH downto 0);
	signal max0_i, max1_i, max2_i, max3_i, max4_i : std_logic_vector (C_DATA_WIDTH downto 0);
	signal max5_i, max6_i, max7_i, max8_i : std_logic_vector (C_DATA_WIDTH downto 0);
	signal load_i, valid_i : std_logic;
	signal din0_i, din1_i, din2_i, din3_i, din4_i : std_logic_vector (C_DATA_WIDTH downto 0);
	signal din5_i, din6_i, din7_i, din8_i : std_logic_vector (C_DATA_WIDTH downto 0);
begin
	max0_i <= (others => '0') when din0 (C_DATA_WIDTH) = '0' else din0;
	max1_i <= (others => '0') when din1 (C_DATA_WIDTH) = '0' else din1;
	max2_i <= (others => '0') when din2 (C_DATA_WIDTH) = '0' else din2;
	max3_i <= (others => '0') when din3 (C_DATA_WIDTH) = '0' else din3;
	max4_i <= (others => '0') when din4 (C_DATA_WIDTH) = '0' else din4;
	max5_i <= (others => '0') when din5 (C_DATA_WIDTH) = '0' else din5;
	max6_i <= (others => '0') when din6 (C_DATA_WIDTH) = '0' else din6;
	max7_i <= (others => '0') when din7 (C_DATA_WIDTH) = '0' else din7;
	max8_i <= (others => '0') when din8 (C_DATA_WIDTH) = '0' else din8;

	min0_i <= (others => '1') when din0 (C_DATA_WIDTH) = '0' else din0;
	min1_i <= (others => '1') when din1 (C_DATA_WIDTH) = '0' else din1;
	min2_i <= (others => '1') when din2 (C_DATA_WIDTH) = '0' else din2;
	min3_i <= (others => '1') when din3 (C_DATA_WIDTH) = '0' else din3;
	min4_i <= (others => '1') when din4 (C_DATA_WIDTH) = '0' else din4;
	min5_i <= (others => '1') when din5 (C_DATA_WIDTH) = '0' else din5;
```

```
min6_i <= (others => '1') when din6 (C_DATA_WIDTH) = '0' else din6;
min7_i <= (others => '1') when din7 (C_DATA_WIDTH) = '0' else din7;
min8_i <= (others => '1') when din8 (C_DATA_WIDTH) = '0' else din8;

sorter_max : sorter_3x3 PORT MAP (
            clk  => clk,
            rst  => rst,
            load => load,
            din0 => max0_i,
            din1 => max1_i,
            din2 => max2_i,
            din3 => max3_i,
            din4 => max4_i,
            din5 => max5_i,
            din6 => max6_i,
            din7 => max7_i,
            din8 => max8_i,
            dout0 => max_i,
            dout1 => open,
            dout2 => open,
            dout3 => open,
            dout4 => open,
            dout5 => open,
            dout6 => open,
            dout7 => open,
            dout8 => open,
             pass  => pass_i,
            valid => valid_i
);

sorter_med : sorter_3x3 PORT MAP (
            clk => clk,
            rst  => rst,
            load => load,
            din0 => din0,
            din1 => din1,
            din2 => din2,
            din3 => din3,
            din4 => din4,
            din5 => din5,
            din6 => din6,
            din7 => din7,
            din8 => din8,
            dout0 => open,
            dout1 => open,
            dout2 => open,
            dout3 => open,
            dout4 => med_i,
            dout5 => open,
            dout6 => open,
            dout7 => open,
            dout8 => open,
            pass  => open,
            valid => open
);
```

```
        sorter_min : sorter_3x3 PORT MAP (
                clk => clk,
                rst  => rst,
                load => load,

                din0 => min0_i,
                din1 => min1_i,
                din2 => min2_i,
                din3 => min3_i,
                din4 => min4_i,
                din5 => min5_i,
                din6 => min6_i,
                din7 => min7_i,
                din8 => min8_i,

                dout0 => open,
                dout1 => open,
                dout2 => open,
                dout3 => open,
                dout4 => open,
                dout5 => open,
                dout6 => open,
                dout7 => open,
                dout8 => min_i,

                pass  => open,
                valid => open
        );

        min <= min_i ((C_DATA_WIDTH-1) downto 0);
        med <= med_i ((C_DATA_WIDTH-1) downto 0);
        max <= max_i ((C_DATA_WIDTH-1) downto 0);

        pass  <= pass_i;
        valid <= valid_i;

end Behavioral;
```

4.7.5.5 Morfološki filter s cevovodno strukturo

Komponenta pipeline_filter_3x3 je tista, ki združuje predhodno opisane komponente v funkcionalno celoto morfološkega filtra. Filter ima dva podatkovna vhodna, od katerih s krmilnim signalom chan izberemo tistega, ki ga želimo uporabiti za obdelavo z morfološkim filtrom. Z logično ničlo na krmilnem vhodu chan izberemo vhodna vrata chan0, z logično enico pa izberemo vhodna vrata chan1. Poleg krmilnega signala load, ki v stanju logične enice služi kot indikator, da je vhodni slikovni element veljaven, sem komponenti dodal še 11-bitni krmilni vhod oper, s pomočjo katerega določimo osnovno ali kombinirano funkcijo filtra.



Slika 4.14: Morfološki filter s cevovodno strukturo

Filter ima dva podatkovna izhoda, katerih funkcijo določimo s pomočjo vhodnega krmilnega signala oper. Kombinacija bitov krmilnega signala oper določa, kakšno funkcijo pripeljemo na posameznega izmed dveh izhodov douta in doutb. Stanje logične enice na izhodnem signalu valid označuje veljavnost izhodov douta in doutb morfološkega filtra. Operacije se glede na izbrano operacijo izvajajo med izhodi notranje komponente sorter_3x3: min, max, med in pass ter dodatnim vhodnim podatkovnim vhodom din_ext. Z bitoma b4 in b5 izberemo povezavo enega izmed izhodov komponente sorter_ 3x3 na notranji signal a_i, bita b2 in b3 določita povezavo enega izmed izhodov komponente sorter_3x3 na notranji signal b_i, bita b0 in b1, pa povezavo na notranji signal c_i. Izjema je izbira povezave na notranji signal c_i signalov a_i in b_i. Kot lahko razberemo iz spodnje tabele, je funkcijo median možno pripeljati le na notranji signal c_i ter nato z ustrezno kombinacijo bitov 9 in 10 krmilnega vhoda oper na želenega izmed izhodov morfološkega filtra. Z biti od b6 do b8 določimo operacijo, ki jo želimo izvesti med notranjimi signali a_i, b_i in c_i. Obliko strukturnega elementa morfološkega filtra velikosti 3x3 določimo s krmilnim vhodom wnd, ki ga lahko spreminjamo tudi med samim delovanjem sistema.

oper(5)	oper(4)	a_i(7:0)
0	0	pass
0	1	max
1	0	min
1	1	din_ext

oper(3)	oper(2)	b_i(7:0)
0	0	pass
0	1	max
1	0	min
1	1	din_ext

oper(1)	oper(0)	c_i(7:0)
0	0	pass
0	1	max
1	0	med
1	1	min

Tabela 4.1: Prikaz izbire operacij morfološkega filtra

oper(8)	oper(7)	oper(6)	oper_out
0	0	0	a_i
0	0	1	b_i
0	1	0	a_i + b_i
0	1	1	a_i - b_i
1	0	0	b_i - a_i
1	0	1	max(a_i, b_i)
1	1	0	min(a_i, b_i)
1	1	1	a_i == b_i

Tabela 4.2: Izbira naprednih funkcij morfološkega filtra

oper(10)	doutb		oper(9)	douta
0	c_i		0	oper_out
1	oper_out		1	c_i

Tabela 4.3: Zamenjava izhodov douta in doutb morfološkega filtra

Rezultati sinteze ene same komponente morfološkega filtra prikazujejo zasedenost programirljivega vezja FPGA XC6SLX45 (tabela 4.4), kjer je uporabljenih le 1 % vseh registrov znotraj gradnikov SLICE, 2 % LUT gradnikov, 1 % distribuiranih pomnilnikov, 2 % za kombinacijsko logiko ter le 1 % vseh blokovnih pomnilnikov. V tabeli so navedeni le nekateri glavni gradniki.

Gradniki	***Razpoložljivi***	***Uporabljeni***	***Zasedenost***
LUT registri	54576	761	1 %
LUT	27288	646	2 %
LUT logika	27288	609	2 %
LUT pomnilniki	6408	17	1 %
RAMB8BWER	232	2	1 %

Tabela 4.4: Zasedenost programirljivega vezja XC6SLX45 po sintezi samo ene komponente morfološkega filtra

Izvorna koda morfološkega filtra s cevovodno strukturo v visokonivojskem strojno opisnem jeziku VHDL se nahaja v dodatku.

5 Rezultati

Rezultati sinteze digitalnega sistema implementiranega znotraj programirljivega vezja FPGA XC6SLX45 (tabela 5.1) prikazujejo količino uporabljenih pomembnejših gradnikov, in sicer je uporabljenih le 14 % vseh registrov, 29 % LUT gradnikov, ki pa niso v celoti izkoriščeni, ter 17 % blokovnih pomnilnih elementov. Sicer je zasedenih 41 % vseh gradnikov SLICE, ki pa niso v celoti izkoriščeni. Uporabljeno programirljivo vezje FPGA XC6SLX45 iz družine Spartan-6 z vidika še neizkoriščenih gradnikov znotraj vezja vsekakor nudi možnosti nadgradnje oziroma dopolnitve digitalnega sistema.

Gradniki	***Razpoložljivi***	***Uporabljeni***	***Zasedenost***
LUT registri	54576	7676	14 %
SLICE	27288	8044	29 %
SLICE logika	27288	7380	27 %
SLICE pomnilniki	6408	402	6 %
SLICE	6822	2863	41 %
Vh/Izh pini	105	306	33 %
RAMB16BWER	19	116	16 %
RAMB8BWER	2	232	1 %
BUFIO2/BUFIO2_2CLKs	1	32	3 %
BUFPLL_MCB	1	4	25 %
DSP48A1	3	58	5 %
MCB	1	2	50 %
PLL_ADV	1	4	2 %

Tabela 5.1: Zasedenost programirljivega vezja XC6SLX45 po sintezi digitalnega sistema

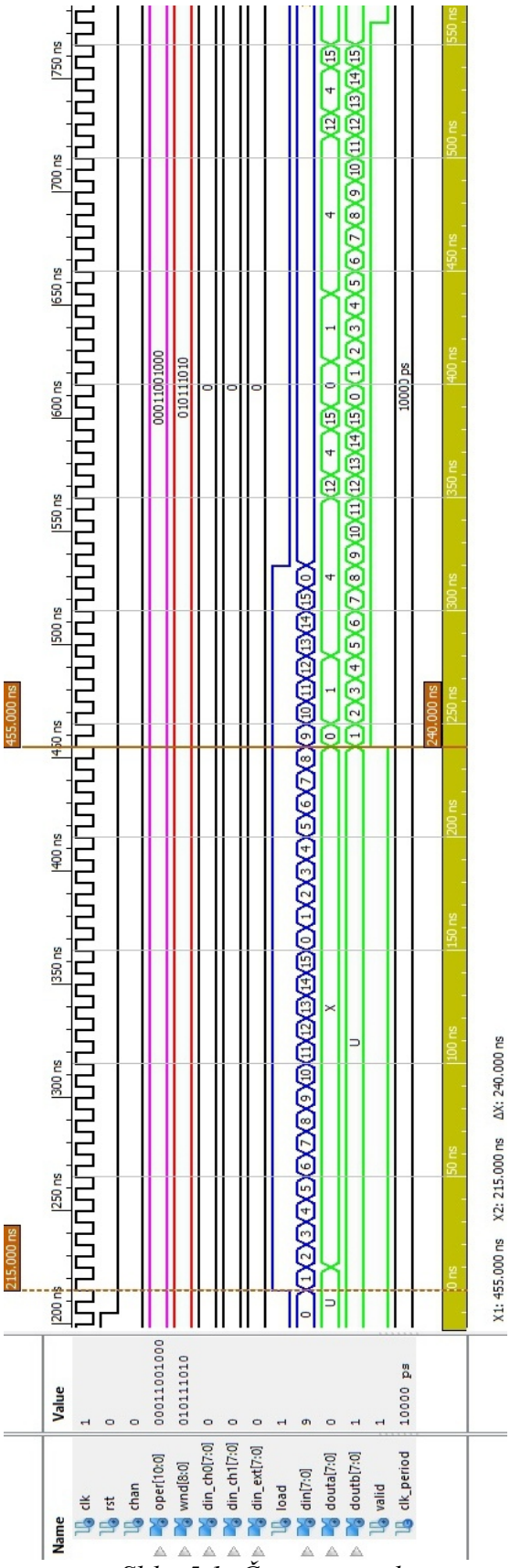
Slika 5.1 prikazuje rezultate simulacije morfološkega filtra za primer dveh zaporednih prehodov vhodnih testnih sivinskih slik velikosti 4x4 slikovnih elementov skozi morfološki filter. Slika 5.2 prikazuje vrednosti slikovnih elementov vhodne sivinske slike, uporabljene za simulacijo delovanja morfološkega filtra. Z modro barvo so na sliki označeni vhodni slikovni elementi v morfološki filter in pripadajoči krmilni signal, ki označuje veljavnost vhodnih podatkov v filter. Z zeleno barvo so na sliki 5.1 označene vrednosti izhodnih slikovnih elementov iz filtra in pripadajoči statusni signal valid, katerega aktivno stanje logične enice

označuje veljavnost izhodnih podatkov morfološkega filtra. Za primer simulacije sem na krmilnem vhodu oper izbral kot osnovno funkcijo morfološkega filtra erozijo, dodatno pa še aritmetično operacijo odštevanja erozije od osnovne vhodne slike. Rezultat takšne operacije je notranji gradient. Strukturni element morfološkega filtra v prikazani simulaciji je velikosti 3x3 slikovne elemente in ima obliko križa. Uporabil sem uro frekvence 100 MHz, kar ustreza dejansko uporabljeni sistemski uri programirljivega vezja FPGA za predstavljen digitalni sistem.

Glede na izbrano operacijo morfološkega filtra dobimo na izhodu douta notranji gradient, na izhodu doutb pa nespremenjeno vhodno sivinsko sliko. Kontrolnika na sliki 5.1 označujeta časovno zakasnitev izhoda morfološkega filtra glede na vhod. Zakasnitev je posledica več dejavnikov.

Za začetek obdelave vhodne slike z morfološkim filtrom potrebujemo zadostno količino podatkov, ki je enaka eni vrstici vhodne slike oziroma velikosti vhodne FIFO strukture. K temu je potrebno prišteti še štiri cikle ure zaradi vhodnih in izhodnih registrov ter sinhronega branja iz FIFO strukture. Od tod izhaja prvi del zakasnitve izhoda filtra, ki je konstanten za izbrano širino vhodne slike, ki za dani primer znaša 4 cikle ure, in neodvisni del zakasnitve, ki znaša 4 cikle ure. Drugi del zakasnitve izhoda morfološkega filtra se nanaša na zakasnitev sortirne komponente sorter_3x3 znotraj filtra, katere cevovodna struktura je 13-stopenjska, kar vnese zakasnitev 13 ciklov ure in je prav tako konstantna in neodvisna od velikosti vhodne slike. Tretji in zadnji del zakasnitev je prav tako konstanten in zaradi 3-stopenjske cevovodne strukture, uporabljene za izvrševanje aritmetičnih operacij filtra nad obdelanimi vhodnimi slikovnimi elementi, znaša 3 cikle ure.

Konstantni del zakasnitev morfološkega filtra znaša 20 ciklov ure in je neodvisna od ločljivosti oziroma širine vhodne slike, preostala zakasnitev pa je konstantna za izbrano širino slike. Tako velja, da je zakasnitev morfološkega filtra za vhodno sliko ločljivosti 640x480 slikovnih elementov enaka 660 ciklom ure, kar pri frekvenci sistemske ure programirljivega vezja FPGA 100 MHz pomeni zakasnitev 6.6 μs.



Slika 5.1: Časovni potek operacije notranjega gradienta morfološkega filtra

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

Slika 5.2: Vhodna testna sivinska slika velikosti 5x5 slikovnih elementov

1	1	2	3
1	2	3	4
5	6	7	0
9	10	0	0

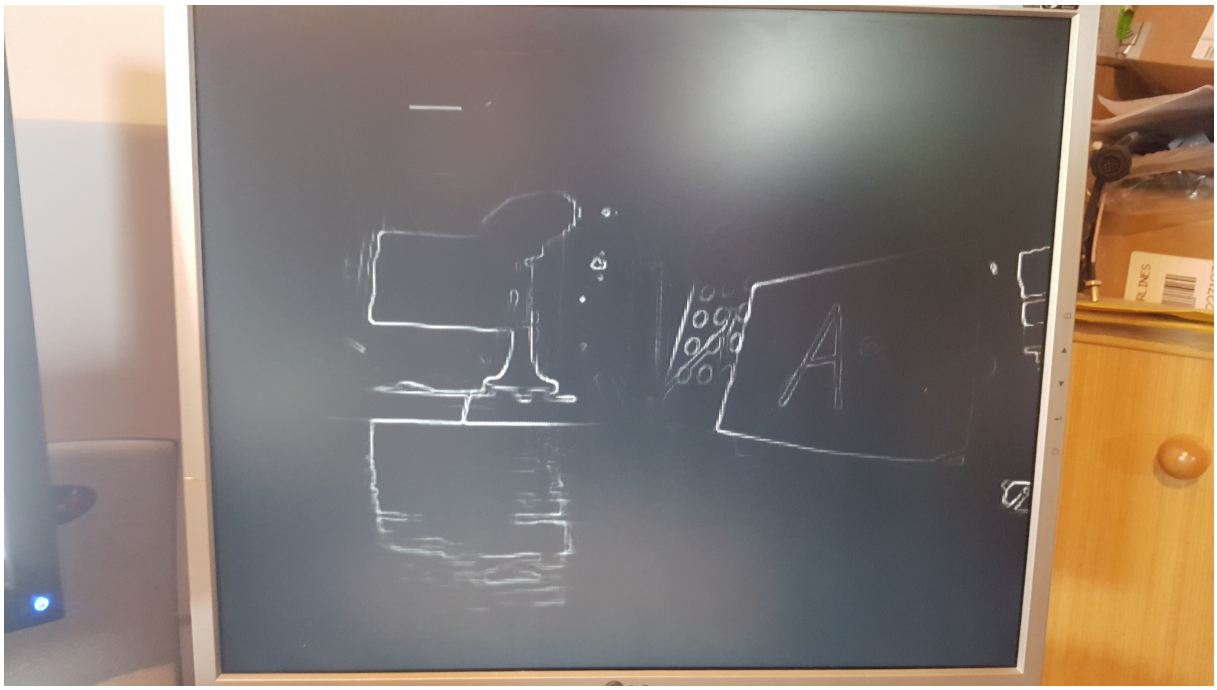
Slika 5.3: Erozija vhodne testne sivinske slike

0	1	1	1
4	4	4	4
4	4	4	12
4	4	15	0

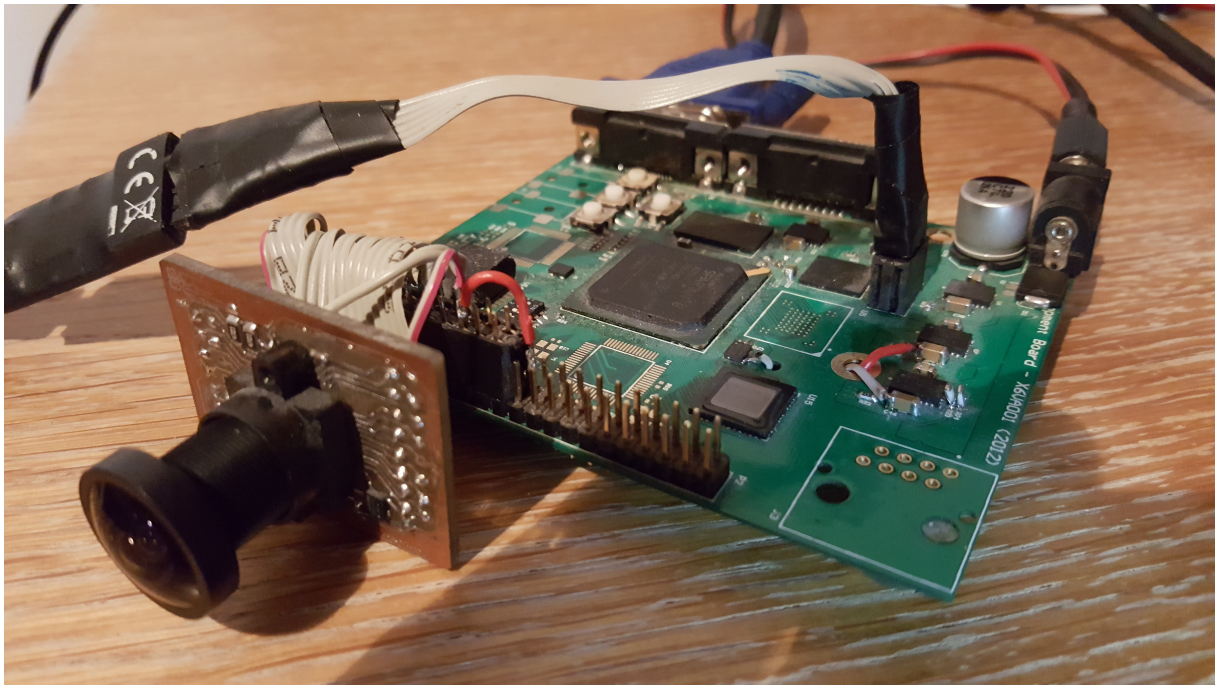
Slika 5.4: Notranji gradient vhodne sivinske slike

Rezultate simulacije morfološke operacije notranjega gradienta iz slike 5.1 sem potrdil še z ročno opravljeno operacijo morfološke erozije (slika 5.3) za vhodno testno sivinsko sliko velikosti 4x4 slikovne elemente (slika 5.2) ter izračunom pripadajočega notranjega gradienta. Notranji gradient dobimo tako, da izračunamo razliko med istoležnimi slikovnimi elementi vhodne testne slike 5.2 in erodirane slike 5.3.

Slika 5.5: Prikaz sivinske slike, zajete z izdelanim digitalnim sistemom



Slika 5.6: Prikaz obdelane sivinske slike z morfološkim operatorjem osnovnega gradienta z izdelanim digitalnim sistemom



Slika 5.7: Izdelan digitalni sistem za obdelavo video slike z morfološkimi operacijami

6 Sklep

Z izdelano strojno opremo, katere jedro predstavljajo CMOS digitalni slikovni senzor, programirljivo vezje FPGA in zunanji pomnilnik, sem uspel izdelati digitalni sistem, ki omogoča obdelavo video sivinskih slik ločljivosti 640x480 slikovnih elementov ter frekvence osveževanja 30 Hz v realnem času. Ustrezno delovanje morfoloških operacij nad sivinsko sliko sem potrdil tako z rezultati simulacije morfološkega filtra na osebnem računalniku, kot tudi z uporabo testnih sivinskih slik, ki sem jih v digitalni sistem naložil iz osebnega računalnika preko asinhronega serijskega komunikacijskega vmesnika (RS-232).

Digitalni sistem nudi možnosti razširitve, saj je v programirljivem vezju FPGA še precej prostora oziroma še veliko neizkoriščenih gradnikov. Trenutna izvedba z vidika hitrosti delovanja morfološkega filtra oziroma celotnega digitalnega sistema vsekakor presega zahteve po realni obdelavi video slike, saj je sistem zmožen obdelati približno 325 video slik v sekundi. Frekvenca zajetih video slik iz CMOS digitalnega slikovnega senzorja je približno 30 Hz, kar pomeni, da je digitalni sistem v času med dvema zaporednima slikama zmožen obdelati približno 10 slik ločljivosti 640x480 slikovnih elementov. Prepustnost sistema bi lahko še dodatno povečal s paralelno izvedbo morfološkega filtra v smislu obdelave štirih slikovnih elementov v enem ciklu ure. S tem bi prepustnost sistema povečal kar za štirikrat.

Glede na trenutno porabo gradnikov znotraj programirljivega vezja FPGA bi lahko znotraj digitalnega sistema implementiral na primer operacijo skeletonizacije, ki je iterativna metoda, pri kateri bi vsekakor bilo potrebno povečati zmogljivost sistema oziroma število obdelanih video slik na sekundo, da bi lahko izvršil potrebno število iteracij morfoloških operacij v realnem času. Ker lahko operacijo skeletonizacije izvajamo le nad binarno sliko in ne nad sivinsko, bi moral najprej izvesti pretvorno sivinske slike v binarno s pomočjo ene izmed metod, kot je na primer Otsu algoritem [22] za določanje mejne vrednosti. Mejna vrednost določa, kateri izmed slikovnih elementov sivinske slike po pretvorbi v binarno sliko zavzamejo vrednost nič in kateri vrednost ena. Zmogljivost sistema za iterativne metode morfoloških operatorjev bi lahko povečal tako, da bi v enem ciklu ure obdelal kar 32 slikovnih elementov, poleg tega bi morfološke operacije, ki so potrebne za eno iteracijo povezal v kaskado na primer vsaj štirih sklopov, kar bi pomenilo, da bi v času obdelave ene same binarne slike lahko izvršil kar štiri iteracije.

7 Literatura

[1] J. Serra, Image Analysis and Mathematical Morphology, Academic Press New York, 1982.

[2] J. Serra, Image Analysis and Mathematical Morphology, Part II: Theoretical Advances, Academic Press, London, 1988.

[3] J. Goutsias, J. Mathematical Morphology, IOS Press, Amsterdam, Berlin, Oxford, Tokyo, Washington DC, 2000.

[4] P. Soille, Morphological Image Analysis, Springer-Verlag, 1999.

[5] C. R. Gonzalez, R. E. Woods, Digital Image Processing, Prentice Hall, 2002.

[6] B. Sharma, V.K. Padney, A. Khan, Implementation of Morphological Image Processing on FPGAs, IJIRCCE Vol. I, Issue 4, 2013.

[7] R. Arunmozhi, G. Mohan, Implementation of Digital Image Morphological Algorithm on FPGA using Hardware Description Language, IJCA Volume 57 – No. 5, 2012.

[8] J. Bartovsky, P. Dokladal, E. Dokladova, M. Bilodeau, M. Akil, Real-Time Implementation of Morphological Filters with Polygonal Structuring Elements, Journal of Real-Time Image Processing, 2013.

[9] Spartan-6 Family Overview (DS160), Xilinx Inc., October 25 2011, http://*www.**xilinx**.com/support/documentation/data_sheets/ds160.pdf* [Dostopano: junij 2016].

[10] Spartan-6 FPGA Packaging and Pinouts (UG385), May 12 2014, http://*www.**xilinx**.com/support/documentation/user_guides/ug385.pdf* [Dostopano: junij 2016].

[11] Spartan-6 FPGA SelectIO Resources User Guide (UG381), Xilinx Inc., October 21 2015, http://www.xilinx.com/support/documentation/user_guides/ug381.pdf, [Dostopano: junij 2016].

[12] LogiCORE IP Multi-Port Memory Controller (MPMC) (v6.03a), Xilinx Inc., 2011, http://www.xilinx.com/support/documentation/ip_documentation/mpmc.pdf [Dostopano: junij 2016].

[13] LogiCORE IP Video In to AXI4-Stream v3.0, Product Guide (PG043), Xilinx Inc., April 2014, http://www.xilinx.com/support/documentation/ip_documentation/v_vid_in_axi4s/v3_0/pg043_v_vid_in_axi4s.pdf [Dostopano: junij 2016].

[14] Spartan-6 FPGA Datasheet: DC and Switching Characteristics (DS162), Xilinx Inc., January 30, 2015, http://*www.**xilinx**.com/support/documentation/data_sheets/ds162.pdf* [Dostopano: junij 2016].

[15] TN-41-13: DDR3 Point-to-Point Design Support, Micron Technology Inc., March 2013, https://www.micron.com/~/media/documents/products/technical-note/dram/tn4113_ddr3_point_to_point_design.pdf [Dostopano: junij 2016].

[16] DDR3 Design Considerations for PCB Applications, Freescale Semiconductor Inc., July 2009, http://www.nxp.com/files/training_pdf/VFTF09_AN111.pdf [Dostopano: junij 2016].

[17] B Olney, DDRx Application Note – Tips for DDR, DDR2 & DDR3, In-Circuit Design Pty Ltd, 2010, http://hands.com/~lkcl/ddr3/DDRx_AN2010.pdf [Dostopano: junij 2016].

[18] MB86R12 Application Note DDR3 Interface PCB Design Guideline, Fujitsu Semiconductor, 2011, https://www.fujitsu.com/us/Images/an-mb86r12-ddr3-pcbdesignguide-rev1-0.pdf [Dostopano: junij 2016].

[19] Hardware and Layout Design Considerations for DDR Memory Interfaces, Freescale Semiconductors, 2007, http://cache.freescale.com/files/32bit/doc/app_note/AN2582.pdf [Dostopano: junij 2016].

[20] V. Sklyarov, J. Skliarova, A. Sudnitson, FPGA-based Accelerators for Parallel Data Sort, Applied Computer Systems, Vol. 16, Issue 1, December 2014.

[21] J. Munson, F. Cox, Video Circuit Collection, Application Note 57, Linear Technology Inc., January1994, http://cds.linear.com/docs/en/application-note/an57fa.pdf [Dostopano: junij 2016].

[22] N. Otsu, A Threshold Selection Metod from Gray-Level Histogram, IEEE Transactions on Systems, man and cybernetics, Vol. SMC-9, No. 1 , January 1979.

[23] K. E. Batcher, Sorting networks and their applications, In Proceedings of the April 30{May 2, 1968, Spring Joint Computer Conference, AFIPS'68 (Spring), pages 307{314, New York, NY, USA, 1968.

[24] 1/2.5-Inch CMOS Digital Image Sensor MT9E001, Aptina, Rev. E, 2006, https://webcache.googleusercontent.com/search?q=cache:al-VdnYE40UJ:https://aptina.atlassian.net/wiki/download/attachments/10551342/MT9E001_DS.pdf%3Fapi%3Dv2+&cd=1&hl=en&ct=clnk&gl=si [Dostopano: junij 2016].

[25] 1/2.5-Inch 5 Mp CMOS Digital Image Sensor, MT9P031 Datasheet, Rev. J, ON Semiconductor, 2015, http://www.onsemi.com/pub_link/Collateral/MT9P031-D.PDF [Dostopano: junij 2016].

[26] Enpirion Power Datasheet, EN5322QI 2A Power SoC Synchronous Buck DC-DC Converter with Integrated Inductor, February 25 2014, https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ds/en5322qi_03454.pdf [Dostopano: junij 2016].

[27] Platform Flash XL High-Density Configuration and Storage Device (DS617), Xilinx Inc., January 07 2010, http://www.farnell.com/datasheets/1498252.pdf [Dostopano: junij 2016].

[28] 1Gb DDR3 SDRAM, Micron, November 2014, http://www.micron.com/~/media/documents/products/data-sheet/dram/ddr3/1gb_ddr3_sdram.pdf [Dostopano: junij 2016].

[29] MAX232x Dual EIA-232 Drivers/Receivers, Texas Instruments, November 2014, http://*www.**ti**.com/lit/ds/symlink/**max232**.pdf*
[Dostopano: junij 2016].

[30] Miniature Wide-angle Lens DSL377, Sunex, *www.optics-online.com/OOL/DSL/**DSL377**.PDF*
[Dostopano: junij 2016].

8 Dodatek

8.1 Izvorna koda komponente morfološkega filtra napisana v visokonivojskem strojno opisnem jeziku VHDL

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity pipeline_filter_3x3 is
	generic (
		C_DATA_WIDTH	: integer := 8;
		C_IMAGE_WIDTH	: integer := 4;
		C_IMAGE_HEIGHT	: integer := 4;
		C_STRUCT_ELEM	: std_logic_vector (8 downto 0) := "010111010"
	);
	port (
		clk	: in std_logic;
		rst	: in std_logic;

		load	: in std_logic;
		chan	: in std_logic;
		oper	: in std_logic_vector (10 downto 0);
		wnd	: in std_logic_vector (8 downto 0);
		din_ch0 : in std_logic_vector ((C_DATA_WIDTH-1) downto 0);
		din_ch1 : in std_logic_vector ((C_DATA_WIDTH-1) downto 0);
		din_ext : in std_logic_vector ((C_DATA_WIDTH-1) downto 0);

		douta	: out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
		doutb	: out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
		valid	: out std_logic
	);
end pipeline_filter_3x3;

architecture Behavioral of pipeline_filter_3x3 is

	component pipe_sm_bram
		generic (
			C_DATA_WIDTH	: integer := 8;
			C_IMAGE_WIDTH	: integer := 4;
			C_IMAGE_HEIGHT : integer := 4
		);
		port (
			clk	: in  std_logic;
			rst	: in  std_logic;

			load	: in std_logic;
			din	: in std_logic_vector ((C_DATA_WIDTH-1) downto 0);

			dout0	: out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
			dout1	: out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
			dout2	: out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
			dout3	: out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
			dout4	: out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
```

```
                dout5   : out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
                dout6   : out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
                dout7   : out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
                dout8   : out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
                valid   : out std_logic
        );
end component;


component sorter_pipeline_3x3
        generic (
                C_DATA_WIDTH : integer := 8
        );
        port (
                clk     : in  std_logic;
                rst     : in  std_logic;

                load    : in  std_logic;
                din0    : in  std_logic_vector (C_DATA_WIDTH downto 0);
                din1    : in  std_logic_vector (C_DATA_WIDTH downto 0);
                din2    : in  std_logic_vector (C_DATA_WIDTH downto 0);
                din3    : in  std_logic_vector (C_DATA_WIDTH downto 0);
                din4    : in  std_logic_vector (C_DATA_WIDTH downto 0);
                din5    : in  std_logic_vector (C_DATA_WIDTH downto 0);
                din6    : in  std_logic_vector (C_DATA_WIDTH downto 0);
                din7    : in  std_logic_vector (C_DATA_WIDTH downto 0);
                din8    : in  std_logic_vector (C_DATA_WIDTH downto 0);

                pass    : out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
                max     : out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
                med     : out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
                min     : out std_logic_vector ((C_DATA_WIDTH-1) downto 0);
                valid   : out std_logic
        );
end component;

signal valid_i : std_logic;
signal din0_i, din1_i, din2_i, din3_i, din4_i     : std_logic_vector ((C_DATA_WIDTH - 1) downto 0);
signal din5_i, din6_i, din7_i, din8_i             : std_logic_vector ((C_DATA_WIDTH - 1) downto 0);
signal din0_ii, din1_ii, din2_ii, din3_ii, din4_ii : std_logic_vector (C_DATA_WIDTH downto 0);
signal din5_ii, din6_ii, din7_ii, din8_ii          : std_logic_vector (C_DATA_WIDTH downto 0);

signal din_ch_i, data_in_i                  : std_logic_vector ((C_DATA_WIDTH - 1) downto 0);
signal a_i, a_ii, b_i, b_ii, c_i, c_ii      : std_logic_vector ((C_DATA_WIDTH - 1) downto 0);

signal douta_i, douta_ii, doutb_i, doutb_ii  : std_logic_vector ((C_DATA_WIDTH - 1) downto 0);
signal pass_i, max_i, med_i, min_i          : std_logic_vector ((C_DATA_WIDTH - 1) downto 0);
signal oper_out_i                           : std_logic_vector ((C_DATA_WIDTH - 1) downto 0);

signal oper_i           : std_logic_vector (10 downto 0);
signal load_i           : std_logic := '0';
signal valid_out_i      : std_logic_vector (1 downto 0);

signal struct_elem_i    : std_logic_vector (8 downto 0) := C_STRUCT_ELEM;

signal valid_bram_i     : std_logic;
signal valid_sorter_i   : std_logic;
```

```
begin

        -- Input channel multiplexor
        din_ch_i <= din_ch0 when (chan = '0') else din_ch1;

        in_mux : process (clk)
        begin
                if (rising_edge(clk)) then
                        if (rst = '1') then
                                data_in_i <= (others => '0');
                                oper_i    <= (others => '0');
                                struct_elem_i <= "010111010";
                                load_i <= '0';
                        else
                                data_in_i <= din_ch_i;
                                oper_i <= oper;
                                struct_elem_i <= wnd;
                                load_i <= load;
                        end if;
                end if;
        end process;

        pipeline_buf : pipe_sm_bram
                generic map (
                        C_DATA_WIDTH   => C_DATA_WIDTH,
                        C_IMAGE_WIDTH  => C_IMAGE_WIDTH,
                        C_IMAGE_HEIGHT => C_IMAGE_HEIGHT
                )
                port map (
                        clk       => clk,
                        rst       => rst,

                        load      => load_i,
                        din       => data_in_i,

                        dout0     => din0_i,
                        dout1     => din1_i,
                        dout2     => din2_i,
                        dout3     => din3_i,
                        dout4     => din4_i,
                        dout5     => din5_i,
                        dout6     => din6_i,
                        dout7     => din7_i,
                        dout8     => din8_i,

                        valid => valid_bram_i
                );

        din0_ii <= struct_elem_i(0) & din0_i;
        din1_ii <= struct_elem_i(1) & din1_i;
        din2_ii <= struct_elem_i(2) & din2_i;
        din3_ii <= struct_elem_i(3) & din3_i;
        din4_ii <= struct_elem_i(4) & din4_i;
        din5_ii <= struct_elem_i(5) & din5_i;
        din6_ii <= struct_elem_i(6) & din6_i;
        din7_ii <= struct_elem_i(7) & din7_i;
        din8_ii <= struct_elem_i(8) & din8_i;
```

```
sorter_3x3 : sorter_pipeline_3x3
        generic map (
                C_DATA_WIDTH => C_DATA_WIDTH
        )
        port map (
                clk     => clk,
                rst     => rst,

                load    => valid_bram_i,
                din0    => din0_ii,
                din1    => din1_ii,
                din2    => din2_ii,
                din3    => din3_ii,
                din4    => din4_ii,
                din5    => din5_ii,
                din6    => din6_ii,
                din7    => din7_ii,
                din8    => din8_ii,

                pass    => pass_i,
                max     => max_i,
                med     => med_i,
                min     => min_i,
                valid => valid_sorter_i
        );


--      Operand A assignment of input data (bits 4 to 5)
a_i <=  pass_i  when (oper_i(5 downto 4) = "00") else
        max_i   when (oper_i(5 downto 4) = "01") else
        min_i   when (oper_i(5 downto 4) = "10") else
        din_ext when (oper_i(5 downto 4) = "11");

--      Operand B assignment of input data (bits 3 to 2)
b_i <=  pass_i  when (oper_i(3 downto 2) = "00") else
        max_i   when (oper_i(3 downto 2) = "01") else
        min_i   when (oper_i(3 downto 2) = "10") else
        din_ext when (oper_i(3 downto 2) = "11");

--      Operand C assignment of input data (bits 0 to 1)
c_i <=  pass_i when (oper_i(1 downto 0) = "00") else
        max_i  when (oper_i(1 downto 0) = "01") else
        med_i  when (oper_i(1 downto 0) = "10") else
        min_i  when (oper_i(1 downto 0) = "11");
```

```
	proc_oper : process (clk)
	begin
		if (rising_edge(clk)) then
			if (rst = '1') then
				valid_out_i(0) <= '0';
				valid_out_i(1) <= '0';
				valid <= '0';
			else
				a_ii <= a_i;
				b_ii <= b_i;
				c_ii <= c_i;

				valid_out_i(0) <= valid_sorter_i;
				valid_out_i(1) <= valid_out_i(0);
				valid <= valid_out_i(1);

				douta_i <= oper_out_i;
				doutb_i <= c_ii;

				douta <= douta_ii;
				doutb <= doutb_ii;

			end if;
		end if;
	end process;

	--	Operation carried out between operand A and B depends on operation bits 6 to 8
	oper_out_i <=	a_ii when (oper_i(8 downto 6) = "000") else
			b_ii when (oper_i(8 downto 6) = "001") else
			(a_ii + b_ii) when (oper_i(8 downto 6) = "010") else
			(a_ii - b_ii) when (oper_i(8 downto 6) = "011") else
			(b_ii - a_ii) when (oper_i(8 downto 6) = "100") else
			a_ii when (oper_i(8 downto 6) = "101" and (a_ii > b_ii)) else
			b_ii when (oper_i(8 downto 6) = "101" and (a_ii < b_ii)) else
			b_ii when (oper_i(8 downto 6) = "110" and (a_ii > b_ii)) else
			a_ii when (oper_i(8 downto 6) = "110" and (a_ii < b_ii)) else
			(others => '1') when (oper_i(8 downto 6) = "111" and (a_ii = b_ii)) else
			(others => '0') when (oper_i(8 downto 6) = "111" and (a_ii /= b_ii));

	--	Data output routing depends on operation bit 9 and 10
	douta_ii <= douta_i when (oper_i(9) = '0') else
		    doutb_i;

	doutb_ii <= doutb_i when (oper_i(10) = '0') else
		    douta_i;

end Behavioral;
```