**SVEUČILIŠTE U SPLITU**

**FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE**

Projektiranje digitalnih sustava

**IZVJEŠTAJ PROJEKTA**

*JEDNOSTAVNA OBRADA SLIKE U VERILOGU*

Ante Doko, Matea Juričić, Marija Kasalo

**SADRŽAJ**

[1. UVOD 3](#_Toc131621908)

[2. IMPLEMENTACIJA 4](#_Toc131621909)

[2.1. Ulazi i izlazi 4](#_Toc131621910)

[2.2. Korišteni moduli 5](#_Toc131621911)

[2.2.1. Modul *rom.v* 6](#_Toc131621912)

[2.2.2. Modul *Debounce.v* 6](#_Toc131621913)

[2.2.3. Modul *shrink.v* 7](#_Toc131621914)

[2.2.4. Modul *effects.v* 8](#_Toc131621915)

[2.2.5. Modul *BaudGenT.v* 10](#_Toc131621916)

[2.2.6. Modul *fifo.v* 11](#_Toc131621917)

[2.2.7. Modul *Tx.v* 11](#_Toc131621918)

[2.3. Korišteni ulazi i izlazi na Spartan-3E pločici 12](#_Toc131621919)

[3. PRIKAZ DOBIVENIH REZULTATA 14](#_Toc131621920)

[3.1. RealTerm 14](#_Toc131621921)

[3.2. Konverzija rezultata u .png format 14](#_Toc131621922)

[3.3. Prikaz dobivenih slika 15](#_Toc131621923)

[4. UOČENI „BUG - ovi“ 17](#_Toc131621924)

[5. ZAKLJUČAK 18](#_Toc131621925)

[MOGUĆA POBOLJŠANJA 19](#_Toc131621926)

[LITERATURA 20](#_Toc131621927)

[PRILOG A – top.v 21](#_Toc131621928)

[PRILOG B – rom.v 24](#_Toc131621929)

[PRILOG C – Debounce.v 25](#_Toc131621930)

[PRILOG D – shrink.v 28](#_Toc131621931)

[PRILOG E – effects.v 31](#_Toc131621932)

[PRILOG F – BaudGenT.v 35](#_Toc131621933)

[PRILOG G – fifo.v 36](#_Toc131621934)

[PRILOG H – Tx.v 38](#_Toc131621935)

[PRILOG I – convert\_u\_hex.py 40](#_Toc131621936)

[PRILOG J – output\_txt.py 41](#_Toc131621937)

[PRILOG K – last\_first.py 42](#_Toc131621938)

[PRILOG L – convert\_ u\_sliku.py 43](#_Toc131621939)

[PRILOG M – spartan.ucf 44](#_Toc131621940)

# UVOD

Zadatak ovog projekta je realizacija jednostavne obrade slike na Xilinx Spartan-3E pločici u Verilogu. Naime, korisnik pomoću prekidača na razvojnoj pločici odabire željeni efekt koji se primjenjuje na sliku rezolucije 30x30 piksela koja je učitana u FPGA (engl. Field Programmable Gate Arrays). Krajnji rezultat obrade se preko UART-a (engl. Universal Asynchronous Receiver Transmitter) prenosi na računalo, a uz pomoć Python skripti obrađenu sliku je zapravo moguće vidjeti.

U nastavku izvještaja dan je pregled mplementacije obrade slike na pločici. Objašnjeni su pojedini dijelovi Verilog koda, korišteni moduli, definirani ulazi i izlazi te prikaz rezultata razvijenog sustava. Cjeloviti kod (kao i .ucf datoteka i Python skripte) je priložen na kraju izvještaja.

Naposljetku su dane prednosti i mane implementacije te su predložena daljnja unaprjeđenja ovog projekta.

# IMPLEMENTACIJA

U nastavku je objašnjeno koji se ulazi i izlazi koriste, svi potrebni moduli i komponente Spartan-3E pločice. Pojašnjeni su najbitniji dijelovi koda nekih modula. Cjeloviti kod se nalazi na kraju kao prilog.

## Ulazi i izlazi

Slika na kojoj se prikazuje grafikon

Opis je automatski generiranNa slici 2.1. prikazan je top modul sa svim njegovim ulazima i izlazima.

Slika 2.1. TOP modul

Ulazi modula:

* clk – takt sklopa – korišten je interni takt Spartan-3E pločice (50 MHz oscilator)
* rst – vraća sklop u početno stajne (korištena je gumb s pločice)
* shr\_or\_eff – sklopka kojom biramo hoćemo li na slici smanjiti rezoluciju ili joj dodati neki od efekata
* start – pokreće obradu slike
* eff [1:0] – dvije sklopke s pločice čijim vrijednostima biramo efekt koji će se primijeniti na sliku. Ponuđeni efekti su opisani su u narednom dijelu izvještaja. Ako se u tijeku serijskog prijenosa podataka promjeni vrijednost ulaza *eff* neće doći do promjene podatak koji se prenose (npr. ako se šalju podaci *BRIGHTEN* operacije, a tijekom prijenosa promijenimo *eff* na 2'b11 tako da se obavlja *GRAYSCALE* operacija, neće se u trenutku promjene ulaza prenositi podaci *GRAYSCALE* operacije već će se dovršiti prijenos podataka *BRIGHTEN* operacije).

Izlazi modula:

* done – gotova je prijenos obrađenih podataka putem UART-a svijetli ledica na pločici
* op\_done – gotova je primjena efekta ili shrink-a
* tx – podatak koji se šalje na UART
* tx\_active – aktivan je prijenos podataka putem UARTA, upaljena je jedna od ledica na pločici.

## Korišteni moduli

Na slici 2.2. prikazana je pojednostavljena blok shema osnovnih modula. Top modul sadrži module:

* *rom.v* – upisivanje podataka slike u memoriju
* *Debounce.v* – stabilizacija signala korištenih sklopki i tipki na pločici
* *shrink.v* – implementacija algoritma za smanjenje rezolucije slike
* *effects.v* – dodavanje efekta na sliku
* *BaudGenT.v* – generiranje takta koji odgovara brzini slanja podataka
* *fifo.v* – memorija u koju se upisuju podaci obrađene slike i iz koje se šalju podaci s pločice na računalo
* *Tx.v* – modul koji implementira slanje podataka na računalo

Slika na kojoj se prikazuje dijagram

Opis je automatski generiranSvaki od navedenih modula s cjelovitim kodom je priložen na dnu izvještaja.

Slika 2.2. Pojednostavljena blok shema

### Modul *rom.v*

Slika na kojoj se prikazuje tekst, okvir za sliku, galerija, slikanje

Opis je automatski generiran  
Parametri *WIDTH, HEIGHT,BPP, PIXELS* definiraju parametre slike. Korištena slika je veličine 30x30 odnosno 900 piksela. Svaki piksel je 24-bitna binarna kodna riječ. Svaki piksel ima R,G i B komponentu. Svakoj komponenti pripada 1 bajt. Slika koja je korištena za obradu prikazana je na Slici 2.3.

Slika 2.3. Slika korištena za obradu

Definirana memorija *data\_mem* ima 900 memorijskih lokacija, po jednu za svaki 24-bitni piksel. Naredba *$readmemh* koristi se za čitanje podataka iz ulazne .txt datoteke. Kako bi podatke iz tekstualne datoteke mogli učitati u memoriju koristeći naredbu readmemh prvo je potrebno sliku konvertirati u hex format. Za to je korištena python skripta *convert\_u\_hex.py* koja je priložena na kraju ovoga izvještaja. Na izlaz *read\_data* se šalje podatak na svaku promjenu ulaza *rd\_addr*.

### Modul *Debounce.v*

Modul *Debounce* predstavlja *debounce* modul za tipkala i prekidače s pločice. Instancirana su 4 *debouncer* modula za svaki od ulaza potrebnih za obradu slike (*start*, *shr\_or\_eff*, *eff*[0] i *eff*[1]). Prilikom korištenja tipkala ili prekidača s pločice dolazi do „poskakivanja“ signala uslijed brzog spajanja i odvajanja metalnih kontakata prije nego imaju vremena da se slegnu. Kao što je prikazano na *slici 2.4.*, prilikom pritiska nekog od tipkala očekuje se čista tranzicija signala iz logičke nule u jedinicu (ili obrnuto). Međutim, stvarni signal ne izgleda tako zbog prethodno Slika na kojoj se prikazuje dijagram

Opis je automatski generirannavedenoga razloga. Modul sadrži 1b ulaz *clk* i 1b izlaz *signal\_i*.

Slika 2.4. „Poskakivanje“ signala tipkala ili prekidača

Slika na kojoj se prikazuje grafikon

Opis je automatski generiran

Slika 2.5. Ulazni i izlazni signal

Na *slici 2.5.* prikazan je ulazni i izlazni signal modula *debouncer*. U glavnom modulu *top* signali bez poskakivanja nazvani su shr\_or\_eff\_stbl, *start\_stbl, effect\_stbl[0]* i *effect\_stbl[1].*

### Modul *shrink.v*

Ovo je modul za smanjivanje (eng. *down-sampling*) dimenzija slike. Ulazi u modul su clk, rst, start i pixel\_in. Izlazi iz modula su pixel\_out, done, write\_adrr i read\_adrr. Modul ima nekoliko parametara *uključujući FACTOR, BYTE\_PER\_PIXEL, HIEGHT, WIDTH, PIXELS* i *ADDR\_WR*. *PIXELS* je ukupan broj piksela u ulaznoj slici, a *ADDR\_WR* je broj adresa potrebnih za pisanje smanjene slike. Modul ima nekoliko internih registara i parametara, uključujući *COL\_LIM, IDLE, GET, SKIP* i *JUMP*. *COL\_LIM* je broj stupaca u smanjenoj slici. *IDLE, GET, SKIP* i *JUMP* su stanja automata koji se koristi za kontrolu algoritma smanjivanja slike. Modul koristi dva "always" bloka da bi se implementirao automat stanja. Prvi "*always*" blok ažurira interne registre na uzlaznu brid takta (eng. *clock*). Drugi "*always*" blok izračunava sljedeće stanje automata i sljedeće adrese za čitanje i pisanje na svaku promjenu ulaznih priključaka ili internih registara.

Ako je stanje :

* IDLE - čeka se signal za pokretanje procesa smanjivanja slike. Kada se primi signal za pokretanje, modul prelazi u stanje GET.
* GET - modul čita piksel iz ulazne slike i zapisuje ga u izlaznu sliku. Zatim modul prelazi u stanje SKIP.
* SKIP- modul preskače FAKTOR-1 piksela u ulaznoj slici i zapisuje smanjeni broj piksela u izlaznu sliku. Ako se dosegne kraj reda, automat prelazi u stanje JUMP. Inače, automat se vraća u stanje GET.
* JUMP - modul preskače FAKTOR-1 redova u ulaznoj slici i zapisuje smanjeni broj piksela u izlaznu sliku. Ako se dosegne kraj slike, automat prelazi u stanje IDLE, inače prelazi u stanje GET.

Kada se završi obrada slike zastavica *done* postavlja se u visoko stanje, a na izlaz se šalju sljedeća adresa za čitanje i sljedeća adresa za pisanje za ulaznu i izlaznu sliku.

*Napomena*: Korišteni algoritam nije najpreciznija metoda smanjivanja slike zbog zanemarivanja velikog broja piksela originalne slike. Postoje algoritmi koji puno bolje implementiraju smanjenje rezolucije slike, kao npr. *Box Filter algorithm [1]*, ali su puno složeniji za implementirati. Ovaj algoritam može se koristiti za slike koje ne sadrže puno detalja.

### Modul *effects.v*

Modul ima parametar *VALUE* koji definira vrijednost koja će se oduzimati ili dodavati pojedinom pikselu kod primjene pojedinih efekata. Modul kao ulaze prima signale *clk, rst, start, eff, pixel\_in*. *eff* je 2-bitni ulaz koji je zapravo ulaz čija se vrijednost definira sklopkama na pločici. U ovisnosti o ulaznoj vrijednosti *eff* na sliku će se primijeniti neki efekt.

Unutar *effects.v* modula definirani su lokalni parametri *NONE, BRIGHTEN, DARKEN* i *GRAYSCALE* koji predstavljaju efekte koji se mogu primijeniti na sliku. Također, unutar modula je implementiran automat stanja s dva stanja *IDLE* i *ACTIVE*.

Unutar prvog „*always*“ bloka provjerava se stanje ulaza *rst*. Ako je ulaz rst postavljen u visoko stanje sve se vrijednosti internih registara modula postavljaju u nulu, a automat je u stanju *IDLE*. Inače, prelazi se u stanje *ACTIVE*. Drugi “*always*“ blok reagira na svaku promjenu nekih od internih registara. Ako je automat stanja u *ACTIVE* stanju na sliku se ovisno o ulazu eff može primijeniti jedan od 4 implementirana efekta.

Ako ulaz *eff* ima vrijednost 2'b**00** onda se na sliku ne primjenjuje efekt već se originalna slika prosljeđuje na izlaz. Ako je ulaz *eff* jednak 2'b**01** onda se na sliku primjenjuje *BRIGHTEN* efekt. Odnosno R,G i B komponenti se dodaje *VALUE* čime se posvjetljuje sam piksel . Ako je vrijednost pojedine komponente piksela veća od 255 (maksimalna vrijednost) onda se komponenti dodjeljuje vrijednost 255.

1. BRIGHTEN : begin
2. temp\_b = blue  + VALUE;
3. temp\_g = green + VALUE;
4. temp\_r = red   + VALUE;
5. **if**(temp\_b > 255) pixel\_out[7:0]   = 8'd255;
6. **else**             pixel\_out[7:0]   = temp\_b;
7. **if**(temp\_g > 255) pixel\_out[15:8]  = 8'd255;
8. **else**             pixel\_out[15:8]  = temp\_g;
9. **if**(temp\_r > 255) pixel\_out[23:16] = 8'd255;
10. **else**             pixel\_out[23:16] = temp\_r;
11. end

Ako ulaz *eff* ima vrijednost 2'b**10** na sliku se primjenjuje DARKEN efekt. Od svake komponente piksela se oduzima vrijednost VALUE. U slučaju da vrijednost pojedine komponente bude manja od nule onda se komponenti dodjeljuje vrijednost nula.

1. DARKEN : begin
2. temp\_b = blue  - VALUE;
3. temp\_g = green - VALUE;
4. temp\_r = red   - VALUE;
5. **if**(temp\_b < 0) pixel\_out[7:0]   = 8'd0;
6. **else**           pixel\_out[7:0]   = temp\_b;
7. **if**(temp\_g < 0) pixel\_out[15:8]  = 8'd0;
8. **else**           pixel\_out[15:8]  = temp\_g;
9. **if**(temp\_r < 0) pixel\_out[23:16] = 8'd0;
10. **else**           pixel\_out[23:16] = temp\_r;
11. end

Ako je ulaz eff jednak 2'b**11** na sliku se primjenjuje algoritam koji pretvara RGB sliku u Grayscale sliku. Korišteni algoritam ne implementira pretvorbu RGB u Grayscale najpreciznije, ali je jako jednostavna metoda i dobra zamjena za bolje,a puno složenije algoritme. Točnije bi bilo koristiti algoritam kojim se zbroje R,G i B komponenta piksela te se taj zbroj podjeli sa 3. Kako operaciju dijeljenja nije moguće direktno primijeniti u sintezi potrebno je implementirati logiku koja bi zamijenila operaciju dijeljenja.

1. GRAYSCALE : begin
2. temp = ((red + green + blue + 1) >> 2);
3. pixel\_out = {temp[7:0],temp[7:0],temp[7:0]};
4. end

Kao izlaz iz modula šalju se *wr\_addr* i *rd\_addr*. *wr\_addr* je izlaz za novu memoriju za sljedeću adresu piksela kojeg treba upisati. *rd\_addr* je izlaz za staru memoriju za sljedeći piksel koji treba pročitati iz nje.

### Modul *BaudGenT.v*

Ovaj modul se koristi za generiranje takta za serijsku komunikaciju pri određenoj brzini prijenosa podataka (eng. *baudrate*). U ovom projektu korištena brzina prijenosa podataka je 9600, te je prema tome parametar *TICK\_PER\_HALF* ovog modula postavljen na vrijednost 2604. Ta vrijednost dobivena je iz formule:

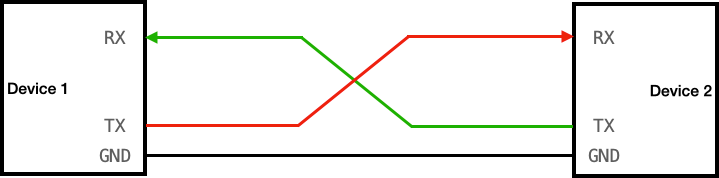
Ulazi u ovaj modul su *clock* i *rst*, a izlaz je signal *baud\_clk* koji generira novi takt u skladu s postavljenom brzinom prijenosa podataka. Unutar modula interni signali su *clock\_ticks* i *final\_value* koji se koriste za praćenje broja taktova i konačne vrijednosti za izlazni signal *baud\_clk*. Prvi „*always*“ blok obrađuje ulazne signale na pozitivnom bridu takta clock. Ako je signal reset aktivan (*rst =* 1), modul postavlja *clock\_ticks* na nulu i *baud\_clk* na 0. U suprotnom, ako je trenutni broj taktova jednak *TICK\_PER\_HALF*, modul ponovno postavlja *clock\_ticks* na nulu i invertira izlazni signal *baud\_clk*. Ako trenutni broj taktova nije jednak *TICK\_PER\_HALF*, modul samo inkrementira brojač *clock\_ticks* za jedan i *baud\_clk* ostaje nepromijenjen.

### Modul *fifo.v*

Ovim modulom implementirana je nova memorija u koju se spremaju pikseli nove slike. Parametri uključuju faktor kojim smanjujemo rezoluciju slike, visinu, širinu, broj bitova po pikselu (BPP), broj piksela (PIXELS) i broj adresa za sliku obrađenu u shrink modulu (ADDR\_WR). Memorija ima 900 memorijskih lokacija i prima 24 bitne podatke. Podaci se u memoriju upisuju na svaki pozitivni brid clk signala kada je omogućeno upisivanje u memoriju odnosno kada je wr\_en=1. Kada se završi upisivanje u memoriju wr\_en se postavlja u nisko stanje, a rd\_en u visoko te se podaci iz memorije šalju na izlaz modula read\_data pri svakom pozitivnom bridu baud\_clk signala.

### Modul *Tx.v*

Prije opisivanja modula Tx, potrebno je objasniti kako funkcionira UART potokol. Na razini UART-a komunikacija je dvosmjerna (full duplex) što znači da je moguć istovremeni prijem i predaja podataka (prikazano na slici 2.7.). S obzirom da je komunikacija asinkrona, da bi prijemnik mogao ispravno dekodirati poslani niz bitova predajnika potrebno je zadovoljiti još nekoliko osnovnih zahtjeva. Prema UART standardu, neaktivno stanje RX/TX linije je digitalna jedinica „1“. Podaci se šalju u okvirima od deset bitova, kako je prikazano na slici 2.8. START bit je uvijek digitalna nula „0“, zatim slijedi osam bitova podataka te nakon slanja podataka slijedi STOP bit koji je uvijek „1“. Ponekad se iza bitova podatka šalje i paritetni bit pomoću kojeg se utvrđuje ispravnost prijema podatka.



Slika 2.7. Povezanost uređaja preko UART-a

Slika na kojoj se prikazuje tekst, sat

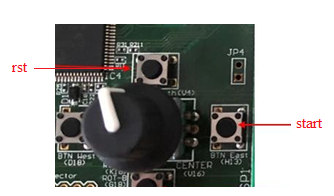
Opis je automatski generiran

Slika 2.8. UART okvir

Trajanje svakog bita definirano je brzinom komunikacije u broju bitova u sekundi i mora biti jednako podešeno na oba kontrolera kao i broj bitova podataka, paritetni bit i broj stop bitova.

Modul kao ulaz prima izlazni podatak iz FIFO memorije. Unutar modula definiran je automat s dva stanja. Ako se automat nalazi u početnom *IDLE* stanju onda se u interni registar *frame\_r* pohranjuje ulazni piksel. *frame\_r* je 28-birni registar jer osim podataka o pikselu sadrži i start i stop bitove. Okvir se na UART šalje u 3 dijela po 8 bitova. Tj. Sklop šalje 8 bitova, zatim pošalje stop bit jednak 0, nakon tog stop bita start bit jednak 1, pa sljedećih 8 bitova podataka, pa opet stop bit jednak 0, pa ponovno start bit jednak 1 i zadnjih 8 bitova piksela i ponovno stop bit jednak 0. U stanju *ACTIVE* na izlaz *data\_tx* šalje se najmanje značajan bit te se primjenom shift-anja u desno za 1 bit prelazi kroz čitav okvir do najznačajnijeg bita. Kada se prenesu svi podatci *done\_flag* se postavlja u 1.

## Korišteni ulazi i izlazi na Spartan-3E pločici

Kao što je i prethodno spomenuto, za ulaze *start* i *rst* korištena su 2 tipkala koja se nalaze na pločici. Kako su korišteni prikazano je na slici 2.10.

Slika 2.10. Korištena tipkala

Za ulaz *sh\_or\_eff* korišten je jedan prekidač, a za *eff* su korištena 2 prekidača. Ukoliko je *sh\_or\_eff* u „1“, izabran je *shrink* kojim se smanjuje rezolucija slike, a ukoliko je u „0“ na sliku se primjenjuje neki od efekata. Kombinacijom 2 prekidača (*eff*[0] i *eff*[1]).

Mogući ishodi su:

* oba prekidača u „0“ – odabran efekt *NONE*,
* *eff*[0] u „0“ i *eff*[1] u „1“ – odabran efekt *DARKEN*,
* *eff*[0] u „1“ i *eff*[1] u „0“ – odabran efekt *BRIGHTEN* i
* oba prekidača u „1“ – odabran efekt *GRAYSCALE*.

Slika na kojoj se prikazuje tekst, elektronika

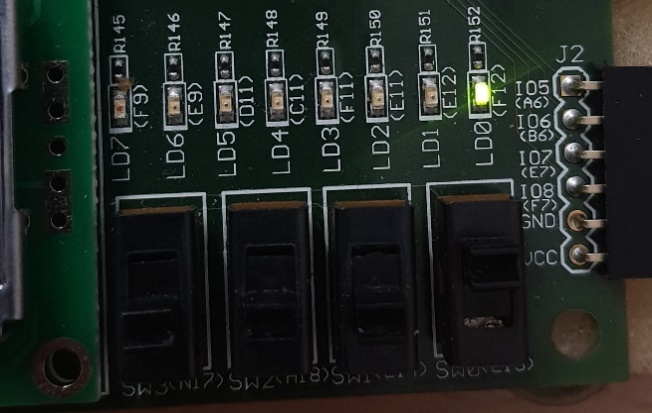
Opis je automatski generiranSlika na kojoj se prikazuje tekst, elektronika

Opis je automatski generiranZa izlaze *done* i *active* korištene su LED diode. Izlaz *active* svijetli sve dok je u tijeku obrada slike, a kad se obrada završi *active* se gasi te se pali *done* kojim se inicira da je slika obrađena. (prikazano na slici 2.11.). Na slici 2.12. prikazan je izlaz *tx*.

Slika 2.12. Izlaz tx (UART)

Slika 2.11. Korišteni prekidači i LED diode

Na slikama 2.12. i 2.13. prikazan je konkretan primjer obrade slike ukoliko se na sliku primijeni efekt *BRIGHTNESS*.

Slika na kojoj se prikazuje tekst, elektronika

Opis je automatski generiran

Slika 2.12. Vizualni prikaz dok je obrada slike u tijeku

Slika 2.13. Vizualni prikaz za završenu obradu slike

# PRIKAZ DOBIVENIH REZULTATA

## RealTerm

Slika na kojoj se prikazuje tekst

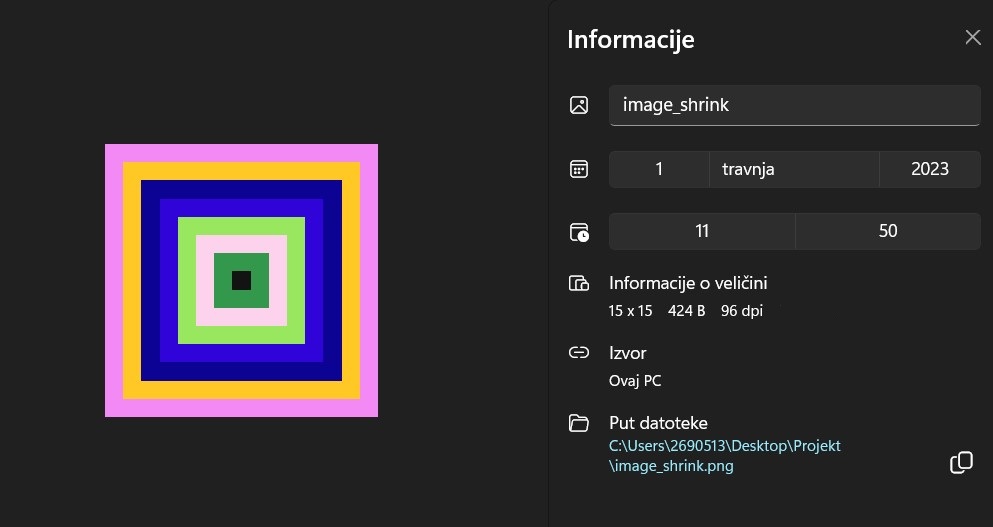
Opis je automatski generiranKako bi se prikazali i pratili izlazni podaci iz pločice, koristi se **RealTerm** - besplatni softver otvorenog koda za upravljanje serijskim portovima na Windows operacijskom sustavu. (slika 2.9.). U Realterm-u je potrebno odabrati ispravni serijski port, postaviti brzinu prijenosa na 9600 (ako je modul *Tx* modificiran, onda postaviti na tu vrijednost), postaviti Display (engl. *prikaz*) na *bin* kako bi se u terminalu ispisivali bitovi u binarnom zapisu. Kako bi dobiveni podaci bili spremljeni na računalu, potrebno je u *Capture* prozoru odabrati mjesto na kojem se tekstualna datoteka sprema te njen naziv. Također, odabire se *Capture as hex* za spremanje podataka u heksadecimalnom brojevnom sustavu.

Slika 3.1. Prikaz sučelja u RealTerm-u

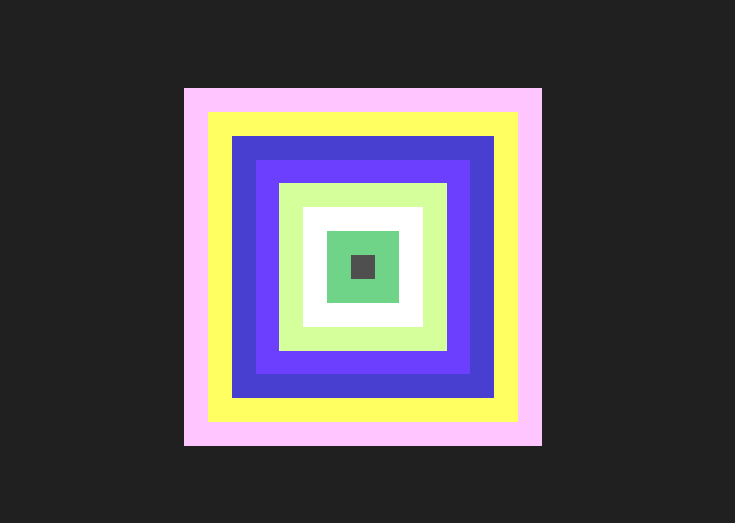
## Konverzija rezultata u .png format

Nova tekstualna datoteka kreirana je na način da python skripta *output\_txt.py* čita *.hex* tekstualnu datoteku dobivenu korištenjem realterma od kraja prema početku datoteke tako da grupira dva znaka kojima zamijeni mjesta. Zatim je potrebno grupirati parove tako da u jednom retku nove tekstualne datoteke bude po 6 heksadecimalnih znakova. Jedan redak predstavlja jedan piksel. Kreiranu datoteku potrebno je ispisati od kraja prema vrhu kako slika ne bi bila okrenuta. Za to je korištena python skripta *last\_first.py*. Korištenjem python skripte *convert\_u\_sliku.py* koja učitava tekstualnu datoteku dobije se *.png* format slike.

## Prikaz dobivenih slika



Slika 3.2. Prikaz slike dobivene downsampling metodom



Slika 3.3. Prikaz slike dobivene primjenom BRIGHTEN efekta

Slika na kojoj se prikazuje tekst, monitor, ukrašeno, okvir za sliku

Opis je automatski generiran

Slika 3.4. Prikaz slike primjenom DARKEN efekta

Slika na kojoj se prikazuje galerija, u dvorani, soba, okvir za sliku

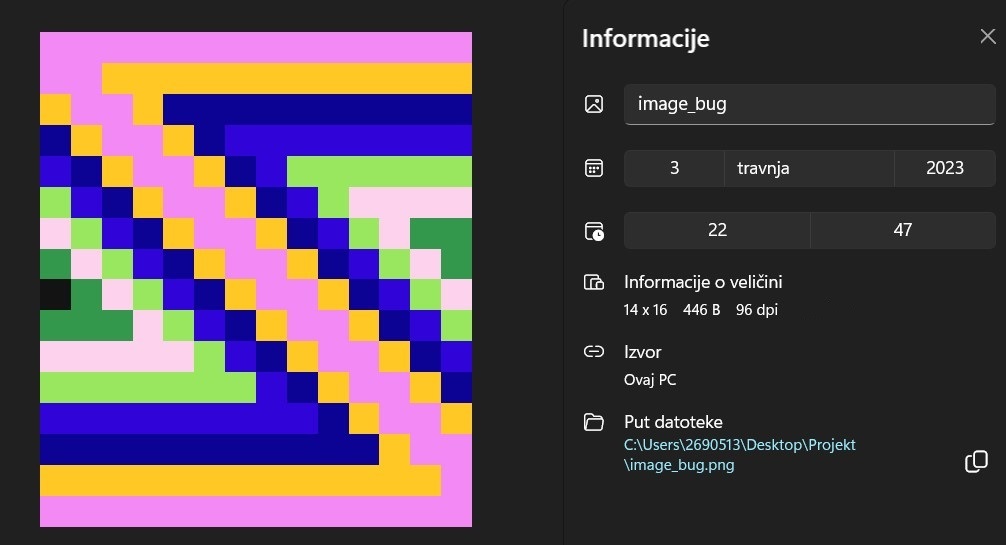
Opis je automatski generiran

Slika 3.5. Prikaz slike primjenom GRAYSCALE efekta

# UOČENI „BUG - ovi“

Prilikom korištenja Realterm-a dalo se primjetiti da se na display prozoru ispisuju 8 bitni binarni podaci. Okvir podataka koji se šalje veličine je 24 bita i to na način da se prvo šalje najmanje značajan bit pa sve do najznačajnijeg. Realterm ispisuje okvir u obrnutom redoslijedu. Prilikom snimanja u .hex datoteku dolazi do gubitka jednog piksela primjenom downsampling algoritma. Kod snimanja podataka u tekstualnu datoteku primjenom nekih od efekata pojavljuje se jedan piksel viška ili točan broj piksela, ali je posljednji piksel na prvom mjestu. Na slikama ispod prikazani su rezultati kada se navedeni bug-ovi ne isprave ručno.

Slika na kojoj se prikazuje tekst, okvir za sliku

Opis je automatski generiran

Slika 4.1. Prikaz bug-a kod downsampling metode

Slika 4.2. Prikaz bug-a kod BRIGHTEN metode

# ZAKLJUČAK

Jednostavna obrada slike je uspješno realizirana i testirana na Spartan 3E pločici. Mana korištenog koda je korištenje slike fiksne rezolucije (30x30 piksela). Slika ove rezolucije korištena je s ciljem brze obrade i manje opterećenosti hardvera. Također, algoritmi smanjenja rezolucije slike i pretvorbe iz RGB formata u GRAYSCALE format nisu najprecizniji. Mogu se koristiti puno bolje metode koje su dosta složenije za implementaciju. UART protokol korišten je za primanje podataka s FPGA na računalo. Umjesto upisivanja podataka o slici direktno u memoriju korištenjem *$readmemh* funkcije moguće je koristiti UART protokol za slanje podataka na pločicu. Spartan 3E podržava korištenje DDR SDRAM memorije koje bi omogućilo upravljanje većim podacima i puno bržom obradom tih podataka. Ako se tijekom snimanja rezultata obrade primjenom nekog efekta npr. *BRIGHTEN* efekta promijeni vrijednost ulaza *eff* na npr. *GRAYSCALE* efekt, dovršit će se prijenos podataka *BRIGHTEN* operacije, tj. neće se u trenutku promjene ulaza promijeniti podaci koji se prenose. Ako se pogleda na snimljene rezultate može se dogoditi da se spremi višak podataka čemu uzrok može biti remećenje sustava u aktivnom prijenosu te RealTerm snimalica koja može dati drukčiji ispis od prethodnog iako je ponovljen identičan postupak. Realizacijom sustava proučeno je spremanje podataka u memoriju FPGA-a, upravljanje tim podacima te njihovom modifikacijom i slanjem putem UART-a.

# MOGUĆA POBOLJŠANJA

* Korekcija koda za učitavanje slika različiti dimenzija
* Korekcija koda kako ne bi došlo do gubitka jednog podatka prilikom smanjenja rezolucije slike
* Implementacija preciznijeg algoritma smanjenja rezolucije slike
* Implementacija preciznijeg algoritma pretvorbe iz RGB u Grayscale
* Korištenje brze memorije DDR SDRAM koju podržava pločica Spartan 3E kako bi se puno brže mogle obrađivati slike veće rezolucije
* Korištenje UART protokola za slanje podataka s računala na FPGA

# LITERATURA

1. <https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=10864&context=theses>
2. <https://www.micromatic.hr/download/Serijska%20komunikacija%20mikrokontrolera.pdf>
3. <https://docs.xilinx.com/v/u/en-US/ug230>
4. <https://www.fpga4student.com/2016/11/image-processing-on-fpga-verilog.html>
5. <https://github.com/SafaKucukkomurler/verilog-button-debouncer/blob/master/sources_1/new/debouncer.v>
6. <https://nandland.com/debounce-a-switch/>
7. <https://www.ijitee.org/wp-content/uploads/papers/v8i7/G5497058719.pdf>

# PRILOG A – top.v

//-----------------Compiler Directives-----------------\\

`timescale 1ns/1ps

module top

//-----------------Parameters-----------------\\

#(

    parameter integer FACTOR = 2,  // Down-sampling factor

    parameter integer VALUE  = 60,

    parameter integer BPP = 3,

    parameter integer HIEGHT = 30,

    parameter integer WIDTH = 30,

    parameter integer PEXILS = HIEGHT\*WIDTH,

    // parameter integer ADDR\_WR = (PEXILS/(FACTOR\*\*2)),

    parameter integer TICK\_PER\_HALF = 2604,    // Fsys/(2\*baudrate)

    parameter integer SZ = (8\*BPP)

)

//-----------------Ports-----------------\\

(

    input clk,                  // System Clock.

    input wire rst,             // Active high synchronous reset, Push button.

    input wire start,           // Push button.

    input wire shr\_or\_eff,      // High for shrink operation, Low for effect operation, switch.

    input wire [1:0] effect,    // Effect selector, switch.

    output wire tx,        // Serial output.

    output wire tx\_active, // Transmittiong is Active, LED.

    output wire done,      // Transmittiong is Done, LED.

    output wire op\_done

);

//-----------------interconnects-----------------\\

wire [SZ-1:0] pixel\_mid, pixel\_mid\_out, ram\_out, rom\_out, eff\_out, shrink\_out;

wire [9:0] rd\_adrr\_w, rd\_adrr\_eff, rd\_adrr\_sh;

wire [9:0] wr\_adrr\_w, wr\_adrr\_eff, wr\_adrr\_sh;

wire ram\_done, b\_clk, effect\_done, shrink\_done, done\_tx, start\_shrink\_w, start\_effects\_w, ram\_done\_wr, ram\_done\_rd;

wire start\_stbl, shr\_or\_eff\_stbl;

wire [1:0] effect\_stbl;

//-----------------Effect Selection-----------------\\

assign start\_shrink\_w  = (shr\_or\_eff\_stbl  && start\_stbl);

assign start\_effects\_w = (!shr\_or\_eff\_stbl && start\_stbl);

assign rd\_adrr\_w       = shr\_or\_eff\_stbl? rd\_adrr\_sh  : rd\_adrr\_eff;

assign wr\_adrr\_w       = shr\_or\_eff\_stbl? wr\_adrr\_sh  : wr\_adrr\_eff;

assign pixel\_mid       = shr\_or\_eff\_stbl? shrink\_out  : eff\_out;

assign apply\_done      = shr\_or\_eff\_stbl? shrink\_done : effect\_done;

//-----------------Debouncing-----------------\\

Debounce  dpb\_rst (

    .clk(clk),

    .signal\_i(start),

    .signal\_o(start\_stbl)

);

Debounce  dpb\_strt (

    .clk(clk),

    .signal\_i(shr\_or\_eff),

    .signal\_o(shr\_or\_eff\_stbl)

);

Debounce  dpb (

    .clk(clk),

    .signal\_i(effect[0]),

    .signal\_o(effect\_stbl[0])

);

Debounce  dpb1 (

    .clk(clk),

    .signal\_i(effect[1]),

    .signal\_o(effect\_stbl[1])

);

//-----------------ROM-----------------\\

rom #(.HIEGHT(HIEGHT), .WIDTH(WIDTH), .BPP(BPP)) ROM (

    .rd\_addr(rd\_adrr\_w),

    .read\_data(rom\_out)

);

//-----------------Processing Unit-----------------\\

shrink #(.FACTOR(FACTOR), .BPP(BPP), .HIEGHT(HIEGHT), .WIDTH(WIDTH)) dwnsamp(

    .clk(clk),

    .rst(rst),

    .start(start\_shrink\_w),

    .pixel\_in(rom\_out),

    .rd\_adrr(rd\_adrr\_sh),

    .wr\_adrr(wr\_adrr\_sh),

    .pixel\_out(shrink\_out),

    .done(shrink\_done)

);

//-----------------Effects module-----------------\\

effects #(.VALUE(VALUE)) effu(

    .clk(clk),

    .rst(rst),

    .start(start\_effects\_w),

    .eff(effect\_stbl),

    .pixel\_in(rom\_out),

    .rd\_adrr(rd\_adrr\_eff),

    .wr\_adrr(wr\_adrr\_eff),

    .pixel\_out(eff\_out),

    .done(effect\_done)

);

//-----------------Baud Clock Generator-----------------\\

BaudGenT #(.TICK\_PER\_HALF(TICK\_PER\_HALF)) baudgen(

    .clock(clk),

    .rst(rst),

    .baud\_clk(b\_clk)

);

//-----------------FIFO-----------------\\

fifo #(.HIEGHT(HIEGHT), .WIDTH(WIDTH), .BPP(BPP)) fi\_fo (

    .wr\_clk(clk),

    .rd\_clk(b\_clk),

    .rst(rst),

    .rd\_en(done\_tx),

    .sh\_en(shr\_or\_eff\_stbl),

    .wr\_en(!apply\_done),

    .wr\_addr(wr\_adrr\_w),

    .write\_data(pixel\_mid),

    .read\_data(ram\_out),

    .rd\_done(ram\_done\_rd),

    .wr\_done(ram\_done\_wr)

);

assign ram\_done = (ram\_done\_rd)? 1'b0 : ram\_done\_wr;

//-----------------UART-Tx-----------------\\

Tx txu(

    .baud\_clk(b\_clk),

    .rst(rst),

    .send(ram\_done),

    .data\_in(ram\_out),

    .data\_tx(tx),

    .active\_flag(tx\_active),

    .done\_flag(done\_tx)

);

assign done = (!ram\_done && done\_tx && apply\_done);

assign op\_done = apply\_done;

endmodule

# PRILOG B – rom.v

module rom

//-----------------Parameters-----------------\\

#(

    parameter HIEGHT = 30,

    parameter WIDTH = 30,

    parameter BPP = 3,

    parameter PEXILS = HIEGHT\*WIDTH

)

//-----------------Ports-----------------\\

(

    input wire [9:0] rd\_addr,

    output reg  [(8\*BPP)-1:0]  read\_data

);

//-----------------Memory Declaration-----------------\\

reg [(8\*BPP)-1:0] data\_mem [0:PEXILS-1];

//-----------------Memory Initialization-----------------\\

initial begin

    $readmemh("input.txt", data\_mem);

end

//-----------------Reading combinationally-----------------\\

always @(rd\_addr) begin

    read\_data  = data\_mem[rd\_addr];

end

endmodule

# PRILOG C – Debounce.v

`timescale 1ns / 1ps

//`define SIMULATION

module Debounce(

input clk,

input signal\_i,

output reg signal\_o

    );

    parameter clock\_freq = 100000000,

              debounce\_time = 1000,

              initial\_value = 1'b0;

    localparam timerlim = clock\_freq / debounce\_time;

    `ifdef SIMULATION

        localparam s\_initial = "s\_initial",

                   s\_zero = "s\_zero",

                   s\_zero\_to\_one = "s\_zero\_to\_one",

                   s\_one = "s\_one",

                   s\_one\_to\_zero = "s\_one\_to\_zero";

        reg [13\*8-1 : 0] state = s\_initial;

    `else

        localparam s\_initial = 3'b000,

                   s\_zero = 3'b001,

                   s\_zero\_to\_one = 3'b010,

                   s\_one = 3'b011,

                   s\_one\_to\_zero = 3'b100;

        reg [2:0] state = s\_initial;

    `endif

    reg [16:0] timer = 17'b0;

    reg timer\_en = 1'b0, timer\_tick = 1'b0;

    always@ (posedge clk) begin

        case (state)

            s\_initial: begin

                if (initial\_value == 1'b0)

                    state <= s\_zero;

                else

                    state <= s\_one;

            end

            s\_zero: begin

                signal\_o <= 1'b0;

                if (signal\_i == 1'b1)

                    state <= s\_zero\_to\_one;

            end

            s\_zero\_to\_one: begin

                signal\_o <= 1'b0;

                timer\_en <= 1'b1;

                if (timer\_tick == 1'b1) begin

                    state <= s\_one;

                    timer\_en <= 1'b0;

                end

                if (signal\_i == 1'b0) begin

                    state <= s\_zero;

                    timer\_en <= 1'b0;

                end

            end

            s\_one: begin

                signal\_o <= 1'b1;

                if (signal\_i == 1'b0)

                    state <= s\_one\_to\_zero;

            end

            s\_one\_to\_zero: begin

                signal\_o <= 1'b1;

                timer\_en <= 1'b1;

                if (timer\_tick == 1'b1) begin

                    state <= s\_zero;

                    timer\_en <= 1'b0;

                end

                if (signal\_i == 1'b1) begin

                    state <= s\_one;

                    timer\_en <= 1'b0;

                end

            end

        endcase

        if (timer\_en == 1'b1) begin

            if (timer == (timerlim - 1)) begin

                timer\_tick <= 1'b1;

                timer <= 17'b0;

            end

            else begin

                timer\_tick <= 1'b0;

                timer <= timer + 17'b1;

            end

        end

        else begin

            timer <= 17'b0;

            timer\_tick <= 1'b0;

        end

    end

endmodule

# PRILOG D – shrink.v

//-----------------Compiler Directives-----------------\\

`timescale 1ns/1ps

module shrink

//-----------------Parameters-----------------\\

#(

    parameter FACTOR = 2,  // Down-sampling factor

    parameter BPP = 3,

    parameter HIEGHT = 30,

    parameter WIDTH = 30,

    parameter integer PEXILS = HIEGHT\*WIDTH

    // parameter PEXILS = (PEXILS/(FACTOR\*\*2))

)

//-----------------Ports-----------------\\

(

    input        clk,

    input        rst,

    input        start,

    input  [(8\*BPP)-1:0] pixel\_in,       // Input image data

    output reg [(8\*BPP)-1:0] pixel\_out,  // Output image data

    output reg   done,

    output [9:0] wr\_adrr, // output to the new data memory for the next pixel address to write

    output [9:0]  rd\_adrr   // output to the old data memory for the next pixel address to read

);

//-----------------Local Parameters-----------------\\

localparam integer COL\_LIM = ((WIDTH+FACTOR-1)/FACTOR);

localparam IDLE = 2'b00,

           GET  = 2'b01,

           SKIP = 2'b10,

           JUMP = 2'b11;

//-----------------Interconnects-----------------\\

reg [1:0] crnt\_st, nxt\_st;

reg [9:0] rd\_address, rd\_address\_r;

reg [9:0] wr\_address, wr\_address\_r;

reg [3:0] row\_count\_r, row\_count, col\_count, col\_count\_r;

//-----------------Shrink FSM-----------------\\

always @(posedge clk) begin

    if (rst) begin

        crnt\_st      <= IDLE;

        rd\_address\_r <= 0;

        wr\_address\_r <= 0;

        row\_count\_r  <= 0;

        col\_count\_r  <= 0;

    end

    else begin

        crnt\_st      <= nxt\_st;

        rd\_address\_r <= rd\_address;

        wr\_address\_r <= wr\_address;

        row\_count\_r  <= row\_count;

        col\_count\_r  <= col\_count;

    end

end

always @(\*) begin

    // default values

    nxt\_st      = crnt\_st;

    rd\_address  = rd\_address\_r;

    wr\_address  = wr\_address\_r;

    nxt\_st      = crnt\_st;

    row\_count   = row\_count\_r;

    col\_count   = col\_count\_r;

    done        = 1'b0;

    case (crnt\_st)

        IDLE: begin

            row\_count         = 0;

            col\_count         = 0;

            if (start) begin

                nxt\_st        = GET;

                rd\_address    = 0;

                wr\_address    = 0;

                done          = 1'b0;

            end

            else begin

                nxt\_st        = IDLE;

                done          = 1'b1;

            end

        end

        GET: begin

            pixel\_out = pixel\_in;

            nxt\_st    = SKIP;

        end

        SKIP: begin

            if (col\_count\_r == COL\_LIM-1) begin

                col\_count  = 0;

                nxt\_st     = JUMP;

            end

            else begin

                rd\_address = rd\_address\_r + (FACTOR);

                wr\_address = wr\_address\_r + 1'b1;

                col\_count  = col\_count\_r + 1'b1;

                nxt\_st     = GET;

            end

        end

        JUMP: begin

            if (row\_count\_r == COL\_LIM-1) begin

                row\_count  = 0;

                nxt\_st     = IDLE;

                done       = 1'b1;

            end

            else begin

                rd\_address     = rd\_address\_r + (((FACTOR-1)\*HIEGHT)+2);

                wr\_address     = wr\_address\_r + 1'b1;

                row\_count      = row\_count\_r + 1'b1;

                nxt\_st         = GET;

            end

        end

        default: nxt\_st    = IDLE;

    endcase

end

//-----------------Output-----------------\\

assign wr\_adrr = wr\_address\_r;

assign rd\_adrr = rd\_address\_r;

endmodule

# PRILOG E – effects.v

module effects

//-----------------Parameters-----------------\\

#(

    parameter VALUE = 50  // Brighten/Darken value

)

//-----------------Ports-----------------\\

(

    input         clk,

    input         rst,

    input         start,

    input  [1:0]  eff,            // effect selector

    input  [23:0] pixel\_in,       // Input image data

    output reg [23:0] pixel\_out,  // Output image data

    output reg        done,

    output [9:0] wr\_adrr,  // output to the old data memory for the next pixel address to write

    output [9:0] rd\_adrr   // output to the old data memory for the next pixel address to read

);

//-----------------Interconnects-----------------\\

reg crnt\_st, nxt\_st;

reg  [9:0] rd\_address, rd\_address\_r;

reg  [9:0] wr\_address, wr\_address\_r;

reg  [9:0] count, count\_r;

reg  [7:0] temp;

wire [7:0] red, blue, green;

integer temp\_r, temp\_b, temp\_g;

localparam NONE      = 2'b00,

           BRIGHTEN  = 2'b01,

           DARKEN    = 2'b10,

           GRAYSCALE = 2'b11;

localparam IDLE   = 1'b0,

           ACTIVE = 1'b1;

assign blue  = pixel\_in[7:0];

assign green = pixel\_in[15:8];

assign red   = pixel\_in[23:16];

//-----------------Intializing Registers-----------------\\

always @(posedge clk) begin

    if (rst) begin

        rd\_address\_r <= 0;

        wr\_address\_r <= 0;

        count\_r      <= 0;

        crnt\_st      <= IDLE;

    end

    else begin

        rd\_address\_r <= rd\_address;

        wr\_address\_r <= wr\_address;

        count\_r      <= count;

        crnt\_st      <= nxt\_st;

    end

end

//-----------------Effect Logic-----------------\\

always @\* begin

    // default values

    rd\_address  = rd\_address\_r;

    wr\_address  = wr\_address\_r;

    count       = count\_r;

    nxt\_st      = crnt\_st;

    done        = 1'b0;

    temp\_r      = 0;

    temp\_b      = 0;

    temp\_g      = 0;

    temp        = 0;

    case (crnt\_st)

        IDLE : begin

            if (start) begin

                nxt\_st        = ACTIVE;

                rd\_address    = 0;

                wr\_address    = 0;

                done          = 1'b0;

            end

            else begin

                nxt\_st        = IDLE;

                done          = 1'b1;

            end

        end

        ACTIVE : begin

            case (eff)

                NONE : begin

                    pixel\_out = pixel\_in;

                end

                BRIGHTEN : begin

                    temp\_b = blue  + VALUE;

                    temp\_g = green + VALUE;

                    temp\_r = red   + VALUE;

                    if(temp\_b > 255) pixel\_out[7:0]   = 8'd255;

                    else             pixel\_out[7:0]   = temp\_b;

                    if(temp\_g > 255) pixel\_out[15:8]  = 8'd255;

                    else             pixel\_out[15:8]  = temp\_g;

                    if(temp\_r > 255) pixel\_out[23:16] = 8'd255;

                    else             pixel\_out[23:16] = temp\_r;

                end

                DARKEN : begin

                    temp\_b = blue  - VALUE;

                    temp\_g = green - VALUE;

                    temp\_r = red   - VALUE;

                    if(temp\_b < 0) pixel\_out[7:0]   = 8'd0;

                    else           pixel\_out[7:0]   = temp\_b;

                    if(temp\_g < 0) pixel\_out[15:8]  = 8'd0;

                    else           pixel\_out[15:8]  = temp\_g;

                    if(temp\_r < 0) pixel\_out[23:16] = 8'd0;

                    else           pixel\_out[23:16] = temp\_r;

                end

                GRAYSCALE : begin

                    temp = ((red + green + blue + 1) >> 2);

                    pixel\_out = {temp[7:0],temp[7:0],temp[7:0]};

                end

                default: pixel\_out = 24'bx;

            endcase

            if(count\_r == 899) begin

                count  = 0;

                done   = 1'b1;

                nxt\_st = IDLE;

            end

            else begin

                nxt\_st     = ACTIVE;

                rd\_address = rd\_address\_r + 1;

                wr\_address = wr\_address\_r + 1;

                count      = count\_r + 1;

                done       = 1'b0;

            end

            end

    endcase

end

//-----------------Output-----------------\\

assign wr\_adrr  = wr\_address\_r;

assign rd\_adrr  = rd\_address\_r;

endmodule

# PRILOG F – BaudGenT.v

//-----------------Compiler Directives-----------------\\

`timescale 1ns/1ps

module BaudGenT

//-----------------Parameters-----------------\\

#(

    parameter TICK\_PER\_HALF = 2604 // BR = 115200    // Fsys/(2\*baudrate)

)

//-----------------Ports-----------------\\

(

    input              clock,

    input wire         rst,

    output reg         baud\_clk

);

//-----------------interconnects-----------------\\

reg [11:0] clock\_ticks, final\_value;

//-----------------Timer logic-----------------\\

always @(posedge clock)

begin

    if(rst)

    begin

        clock\_ticks <= 0;

        baud\_clk    <= 1'b0;

    end

    else

    begin

        if (clock\_ticks == TICK\_PER\_HALF)

        begin

            clock\_ticks <= 0;

            baud\_clk    <= ~baud\_clk;

        end

        else

        begin

            clock\_ticks <= clock\_ticks + 1'd1;

            baud\_clk    <= baud\_clk;

        end

    end

end

endmodule

# PRILOG G – fifo.v

//-----------------Compiler Directives-----------------\\

`timescale 1ns/1ps

module fifo

//-----------------Parameters-----------------\\

#(

    parameter FACTOR = 2,

    parameter HIEGHT = 30,

    parameter WIDTH = 30,

    parameter BPP = 3,

    parameter PEXILS = HIEGHT\*WIDTH,

    parameter ADDR\_WR = (PEXILS/(FACTOR\*\*2))

)

//-----------------Ports-----------------\\

(

    input  wr\_clk, rd\_clk,

    input  wire        rst,

    input  wire        sh\_en,

    input  wire        rd\_en,

    input  wire        wr\_en,

    input  wire [9:0] wr\_addr,

    input  wire [(8\*BPP)-1:0]  write\_data,

    output reg         wr\_done,

    output reg         rd\_done,

    output reg  [(8\*BPP)-1:0]  read\_data

);

//-----------------Memory Declaration-----------------\\

reg [(8\*BPP)-1:0] data\_mem [0:PEXILS-1];

reg [9:0] rd\_addr = 0;

// reg wr\_done, rd\_done;

//-----------------writing @ posedge of the system clock-----------------\\

always @(posedge wr\_clk) begin

    if (rst) begin

        wr\_done        <= 1'b0;

    end

    else begin

        if (wr\_en) data\_mem[wr\_addr] <= write\_data;

        if(sh\_en) begin

            if (wr\_addr == (ADDR\_WR-1)) wr\_done     <= 1'b1;

            else                       wr\_done     <= 1'b0;

        end

        else begin

            if (wr\_addr == (PEXILS-1)) wr\_done     <= 1'b1;

            else                       wr\_done     <= 1'b0;

        end

    end

end

//-----------------Reading @ posedge of the baud clock-----------------\\

always @(posedge rd\_clk) begin

    if(rd\_en && !wr\_en) begin

        if (wr\_addr == rd\_addr) begin

            rd\_done     <= 1'b1;

            rd\_addr     <= rd\_addr;

        end

        else begin

            rd\_done     <= 1'b0;

            rd\_addr     <= rd\_addr + 1;

        end

        read\_data       <= data\_mem[rd\_addr];

    end

    // if(rd\_done) done    <= 1'b0;

    // else        done    <= wr\_done;

end

endmodule

# PRILOG H – Tx.v

//-----------------Compiler Directives-----------------\\

`timescale 1ns/1ps

module Tx

(

    input wire           rst,

    input wire           send,

    input wire           baud\_clk,

    input wire [23:0]  data\_in,

    output reg           data\_tx,

    output reg           active\_flag,

    output reg           done\_flag

);

//-----------------Interconnects-----------------\\

reg [28:0]  frame\_r, frame\_man;

reg [4:0]   stop\_count, stop\_count\_r;

reg         crnt\_st, nxt\_st;

//-----------------Encoding the states-----------------\\

localparam IDLE   = 1'b0,

           ACTIVE = 1'b1;

//-----------------Frame Generation-----------------\\

always @(posedge baud\_clk) begin

    // if (crnt\_st == IDLE)  frame\_r <= {1'b0,data\_in[23:16],1'b1,1'b0,data\_in[15:8],1'b1,1'b0,data\_in[7:0]};

    if (crnt\_st == IDLE)  frame\_r <= {data\_in[23:16],1'b0,1'b1,data\_in[15:8],1'b0,1'b1,data\_in[7:0],1'b0};

    else                  frame\_r <= frame\_man;

end

//-----------------Counter Logic-----------------\\

always @(posedge baud\_clk) begin

    if (rst) stop\_count\_r <= 0;

    else     stop\_count\_r <= stop\_count;

end

//-----------------Transmission Logic FSM-----------------\\

always @(posedge baud\_clk, posedge rst) begin

    if (rst) crnt\_st   <= IDLE;

    else     crnt\_st   <= nxt\_st;

end

always @(\*) begin

    // default values

    nxt\_st       = crnt\_st;

    frame\_man    = frame\_r;

    stop\_count   = stop\_count\_r;

    done\_flag    = 1'b1;

    active\_flag  = 1'b0;

    data\_tx      = 1'b1;

    case (crnt\_st)

        IDLE: begin

            stop\_count         = 0;

            active\_flag    = 1'b0;

            done\_flag      = 1'b1;

            if (send) begin

                nxt\_st         = ACTIVE;

                active\_flag    = 1'b1;

                done\_flag      = 1'b0;

            end

            else      nxt\_st   = IDLE;

        end

        ACTIVE: begin

            active\_flag  = 1'b1;

            done\_flag    = 1'b0;

            data\_tx      = frame\_man[0];

            frame\_man    = frame\_r >> 1;

            stop\_count   = stop\_count\_r + 1;

            if(stop\_count == 29) begin

                nxt\_st  = IDLE;

                active\_flag    = 1'b0;

                done\_flag      = 1'b1;

            end

            else                 nxt\_st  = ACTIVE;

        end

        default: nxt\_st = IDLE;

    endcase

end

endmodule

# PRILOG I – convert\_u\_hex.py

import cv2

import numpy as np

# ----------->>> Conversion from the png to the hex file

# Read the input image

img = cv2.imread('slika\_original.png', cv2.IMREAD\_COLOR)

# Convert the image to a 1-dimensional array of RGB pixels

data = img.reshape((-1, 3))

# Convert the RGB values to hex strings

hex\_data = [format(int(x), '02x') for x in data.flatten()]

# Join the hex strings into a single string with line breaks after every 3 bytes

hex\_data\_str = '\n'.join([''.join(hex\_data[i:i+3])

                         for i in range(0, len(hex\_data), 3)])

# Write the hex data to a text file

with open('input\_original.txt', 'w') as file:

    file.write(hex\_data\_str)

# PRILOG J – output\_txt.py

with open('grayscale\_nasa.txt', 'r', encoding='utf-8') as file:

    letters = file.read().strip()

# Reverse the order of the letters and group them in pairs

reversed\_letters = ''.join(

    reversed([letters[i:i+2] for i in range(0, len(letters), 2)]))

# Write the reversed letters to a new file in groups of 6 on each line

with open('grayscale.txt', 'w', encoding='utf-8') as file:

    for i in range(0, len(reversed\_letters), 6):

        file.write(reversed\_letters[i:i+6]+'\n')

# PRILOG K – last\_first.py

with open("grayscale.txt", "r") as f\_in:

    # Read in the contents of the input file

    contents = f\_in.read()

# Split the contents into individual lines

lines = contents.splitlines()

# Reverse the order of the lines

lines.reverse()

# Join the reversed lines back together into a single string

new\_contents = "\n".join(lines)

with open("grayscale.txt", "w") as f\_out:

    # Write the reversed contents to the output file

    f\_out.write(new\_contents)

# PRILOG L – convert\_ u\_sliku.py

# ----------->>> Conversion from the hex to the png

import cv2

import numpy as np

# Input to define the nuber of bytes per pexil

BPP = 3

# Load the hex data from the text file

with open('shrink.txt', 'r') as f:

    hex\_data = f.read().replace('\n', '')

if (BPP == 3):

    dvsr = 6

else:

    dvsr = 1

num\_pixels = len(hex\_data) // 6

width = int(num\_pixels \*\* 0.5)

height = num\_pixels // width

# Convert the hex data to a NumPy array of uint8 values

data = np.frombuffer(bytes.fromhex(hex\_data), dtype=np.uint8)

# Reshape the data into an array of RGB pixels

data = data.reshape((height, width, 3))

# Convert the array to an RGB image

img = cv2.cvtColor(data, cv2.COLOR\_RGB2RGBA)

cv2.imwrite('image\_bug.png', img)

# PRILOG M – spartan.ucf

#####################################################

### SPARTAN-3E STARTER KIT BOARD CONSTRAINTS FILE

#####################################################

# ==== Pushbuttons (BTN) ====

NET "start" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN;

NET "rst" LOC = "V4" | IOSTANDARD = LVTTL | PULLDOWN;

#NET "BTN\_SOUTH" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN;

#NET "BTN\_WEST" LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN;

# ==== Clock inputs (CLK) ====

NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33;

#Define clock period for 50 MHz oscillator (40%/60% duty-cycle)

NET "clk" PERIOD = 20.0ns HIGH 50%;

#NET "CLK\_AUX" LOC = "B8" | IOSTANDARD = LVCMOS33;

#NET "CLK\_SMA" LOC = "A10" | IOSTANDARD = LVCMOS33;

# ==== Discrete LEDs (LED) ====

# These are shared connections with the FX2 connector

NET "tx\_active" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;

# NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;

# NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;

NET "done" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;

# NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;

# NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;

# NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;

# NET "LED<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;

# ==== RS-232 Serial Ports (RS232) ====

NET "tx" LOC = "M14" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = SLOW; #

#NET "tx" LOC = "M13" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = SLOW; #

# NET "RS232\_DCE\_RXD" LOC = "R7" | IOSTANDARD = LVTTL;

#NET "RS232\_DTE\_RXD" LOC = "U8" | IOSTANDARD = LVTTL;

# ==== Slide Switches (SW) ====

NET "effect[0]" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;

NET "effect[1]" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP;

NET "shr\_or\_eff" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP;

#NET "SW<3>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP;