SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

Projektiranje digitalnih sustava

IZVJEŠTAJ PROJEKTA

JEDNOSTAVNA OBRADA SLIKE U VERILOGU

Ante Doko, Matea Juričić, Marija Kasalo

Split, travanj 2023.

SADRŽAJ

1. 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.

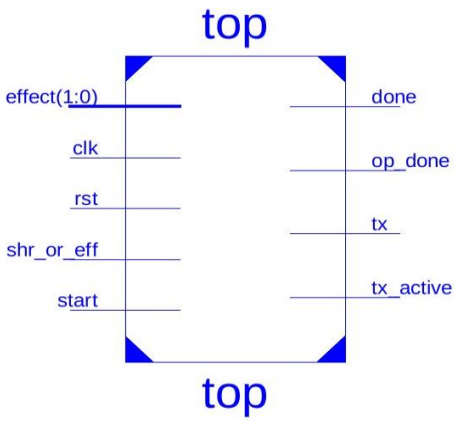
Naposljetku su dane prednosti i mane implementacije te su predložena daljnja unaprjeđenja ovog projekta.

2. 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.

2.1. Ulazi i izlazi

Na 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:

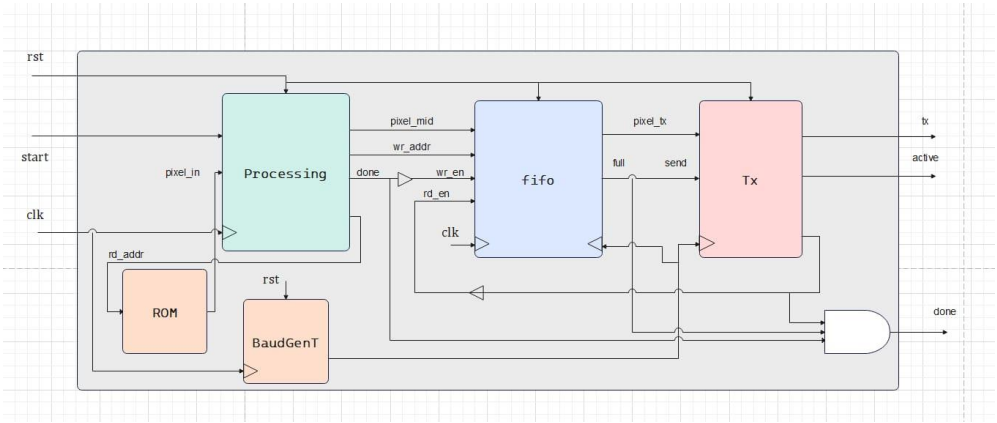
- done – gotova je prijenos obrađenih podataka putem UART-a svijetli ledica na pločici
- op_done – gotova je primjena efekta ili shrink-a
- tx – podatak koji se šalje na UART
- tx_active – aktivan je prijenos podataka putem UARTA, upaljena je jedna od ledica na pločici.

2.2. 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

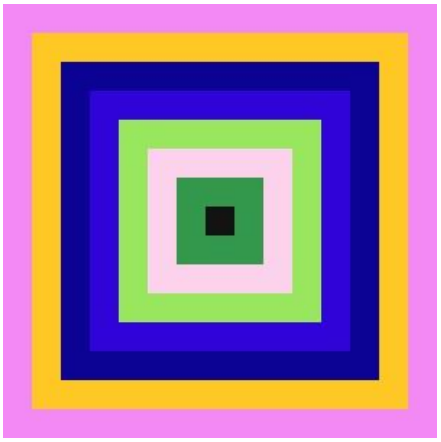
Svaki od navedenih modula s cjelovitim kodom je priložen na dnu izvještaja.



Slika 2.2. Pojednostavljena blok shema

2.2.1. Modul *rom.v*

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.



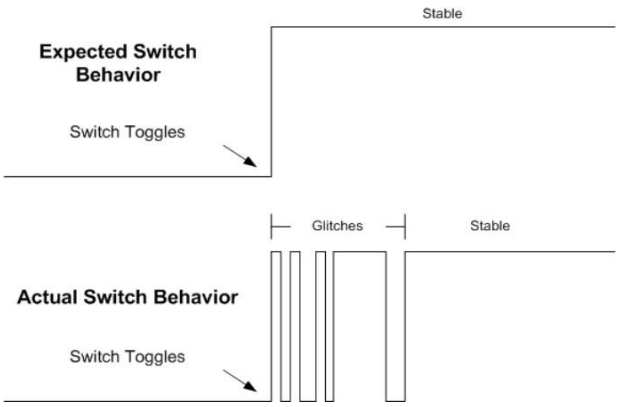
Slika 2.3. Slika korištena za obradu

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*.

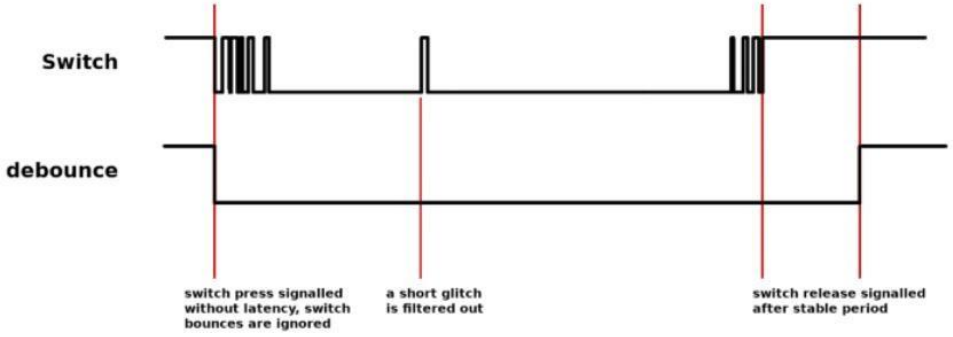
2.2.2. Modul *Debounce.v*

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 navedenoga razloga. Modul sadrži 1b ulaz *clk* i 1b izlaz *signal_i*.



Slika 2.4. „Poskakivanje“ signala tipkala ili prekidača



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]*.

2.2.3. 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.

2.2.4. 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.

2.2.5. 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:

$$\text{TICK\_PER\_HALF} = \frac{Freqency}{2 * Baudrate} = \frac{50\,000\,000}{2 * 9600} = 2604$$

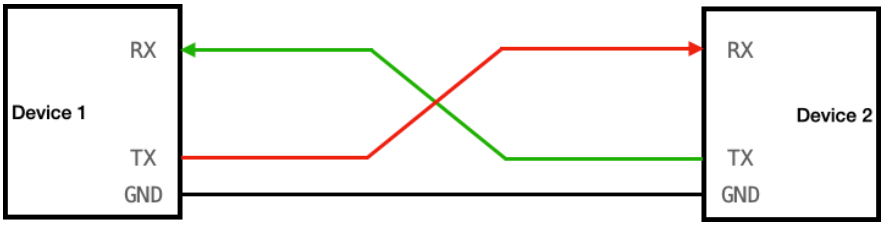
Ulazi u ovaj modul su *clock* i *rst*, a izlaz je signal *baud_clk* koji generira novi takt u skladu s postavljenom brzinom prijenosa podataka. Unutar modula interni signali su *clock_ticks* i *final_value* koji se koriste za praćenje broja taktova i konačne vrijednosti za izlazni signal *baud_clk*. 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.

2.2.6. Modul *fifo.v*

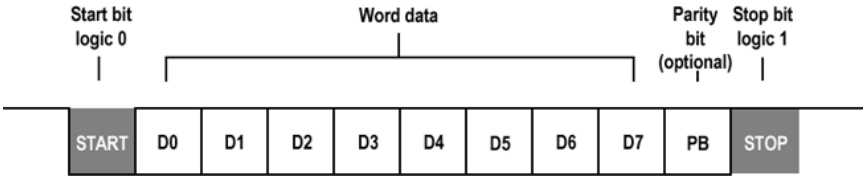
Ovim modulom implementirana je nova memorija u koju se spremaju pikseli nove slike. Parametri uključuju faktor kojim smanjujemo rezoluciju slike, visinu, širinu, broj bitova po pikselu (BPP), broj piksela (PIXELS) i broj adresa za sliku obrađenu u shrink modulu (ADDR_WR). Memorija ima 900 memorijskih 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.

2.2.7. 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

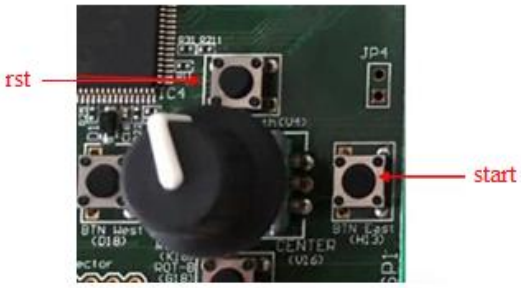


Slika 2.8. UART okvir

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.

2.3. 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.



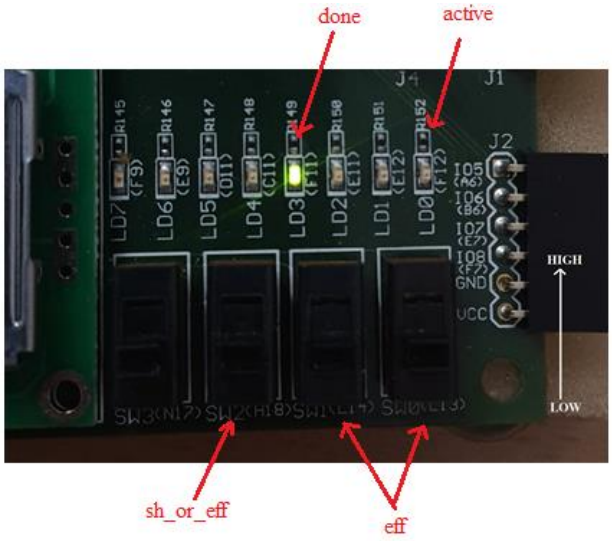
Slika 2.10. Korištena tipkala

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]).

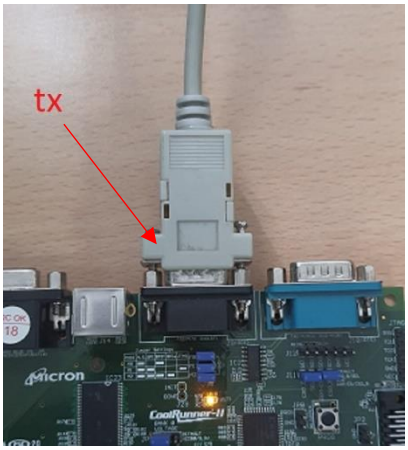
Mogući ishodi su:

- 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*.

Za 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*.

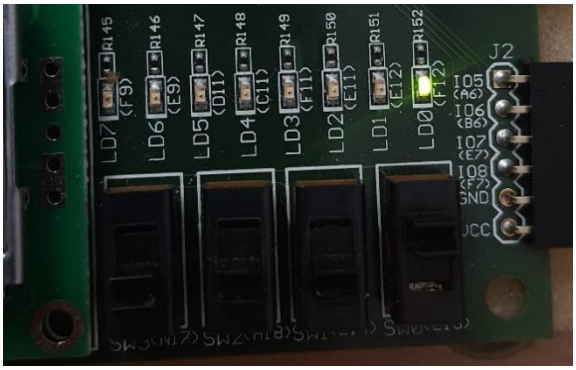


Slika 2.11. Korišteni prekidači i LED diode

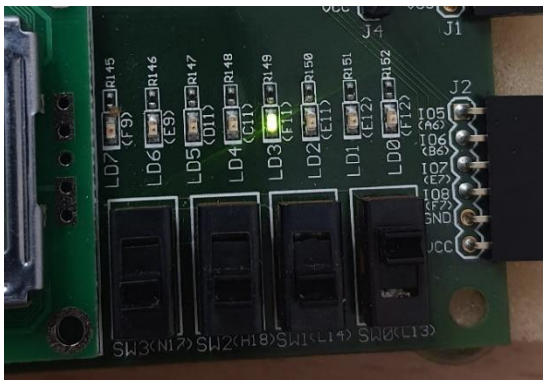


Slika 2.12. Izlaz tx (UART)

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



Slika 2.12. Vizualni prikaz dok je obrada slike u tijeku

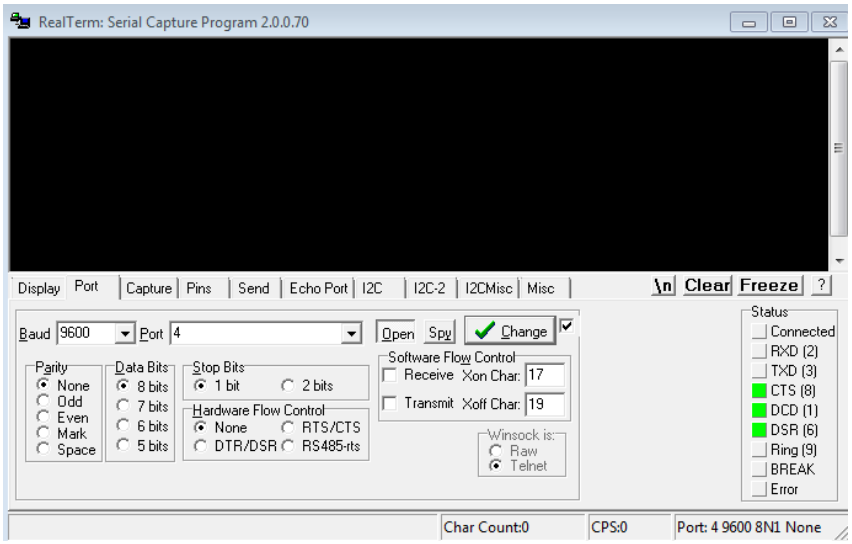


Slika 2.13. Vizualni prikaz za završenu obradu slike

3. PRIKAZ DOBIVENIH REZULTATA

3.1. RealTerm

Kako 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.

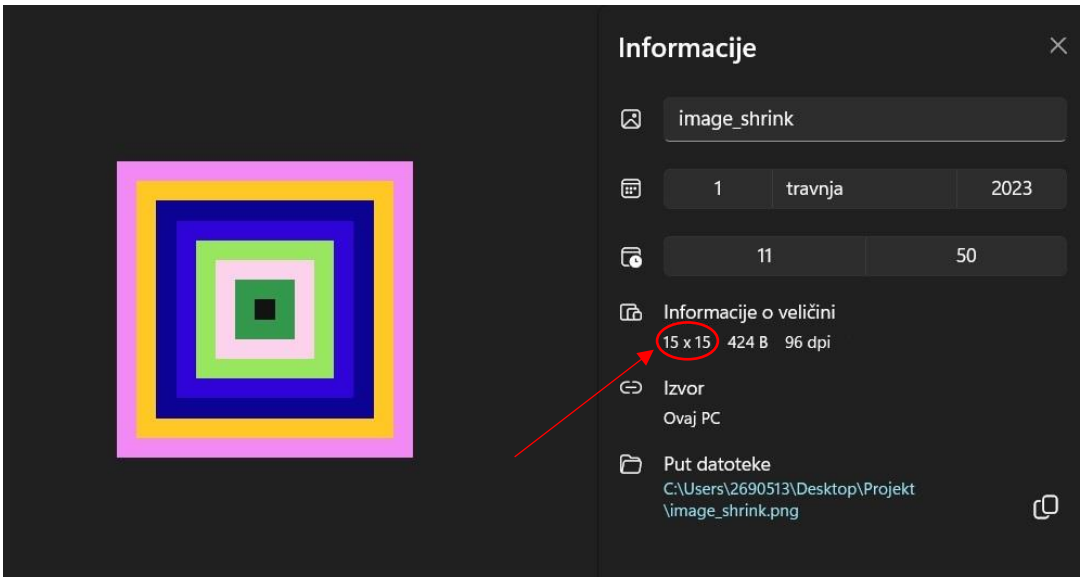


Slika 3.1. Prikaz sučelja u RealTerm-u

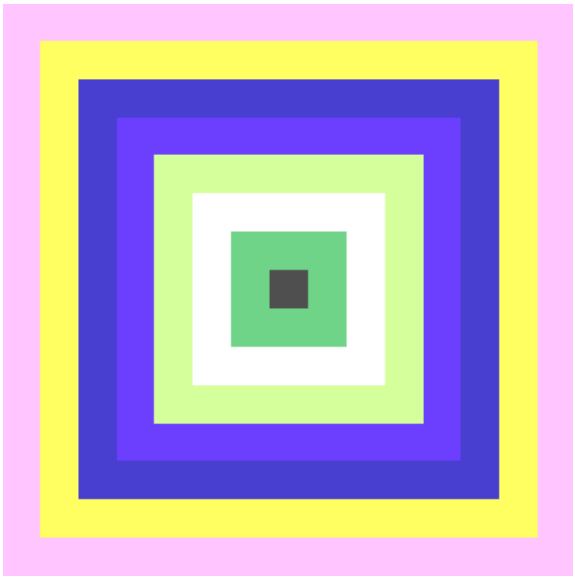
3.2. Konverzija rezultata u .png format

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.

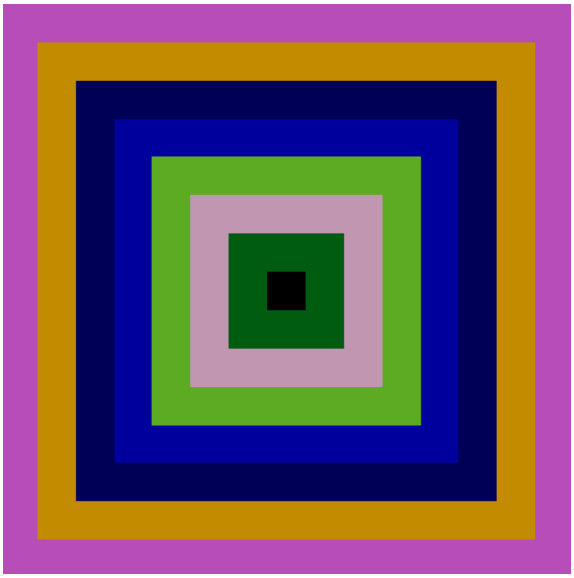
3.3. Prikaz dobivenih slika



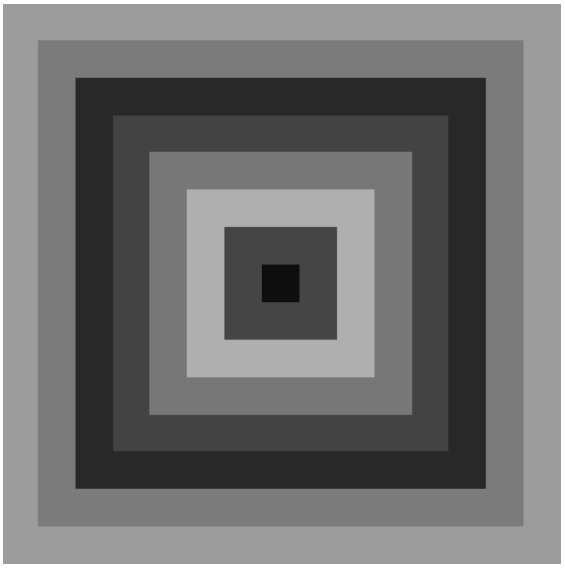
Slika 3.2. Prikaz slike dobivene downsampling metodom



Slika 3.3. Prikaz slike dobivene primjenom BRIGHTEN efekta



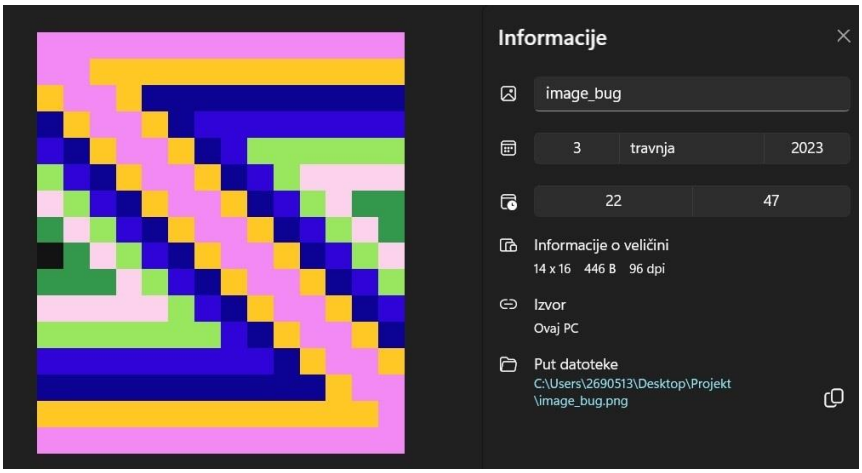
Slika 3.4. Prikaz slike primjenom DARKEN efekta



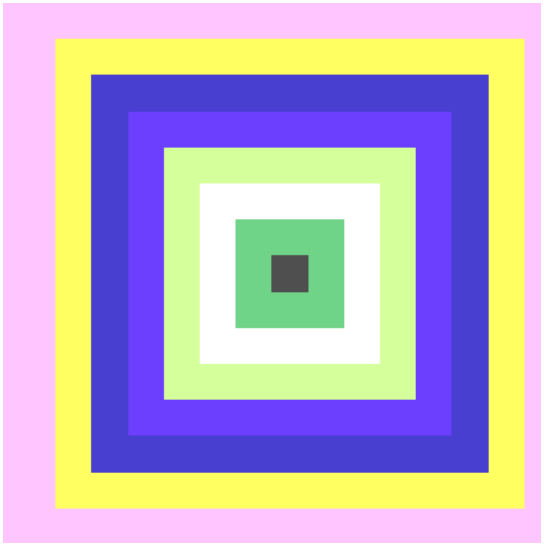
Slika 3.5. Prikaz slike primjenom GRAYSCALE efekta

4. 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 4.1. Prikaz bug-a kod downsampling metode



Slika 4.2. Prikaz bug-a kod BRIGHTEN metode

5. 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;
```