Zagovor laboratorijskih vaj / NDES (VS) (projekt)

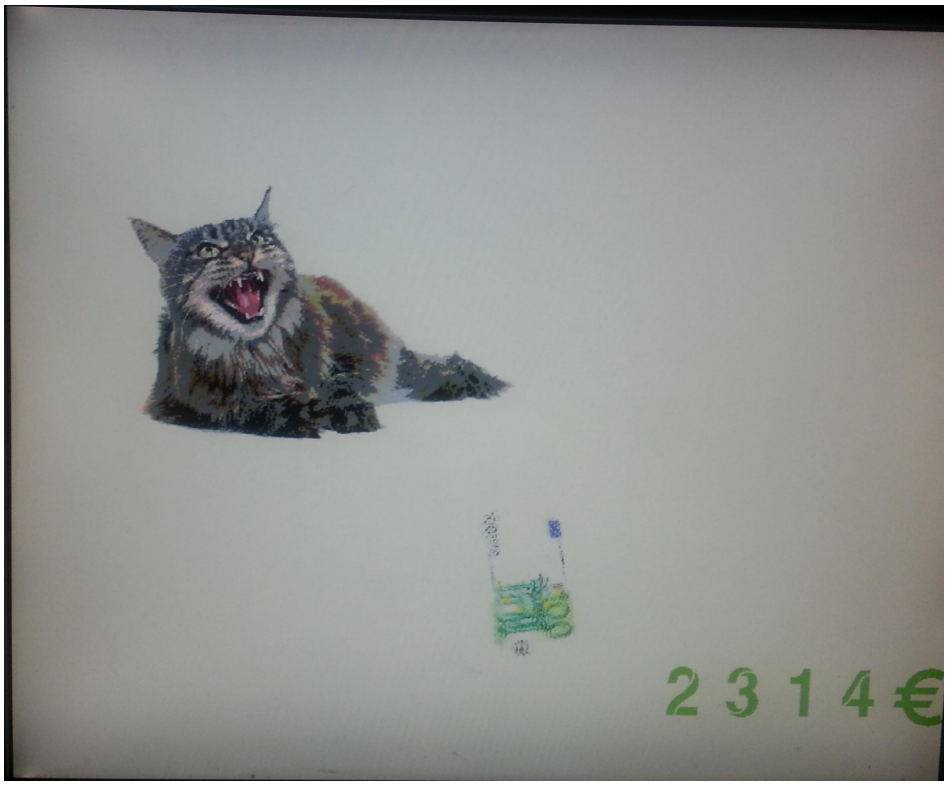
Bojan Faletič

1. februar 2019

Kazalo

1 Opis igre

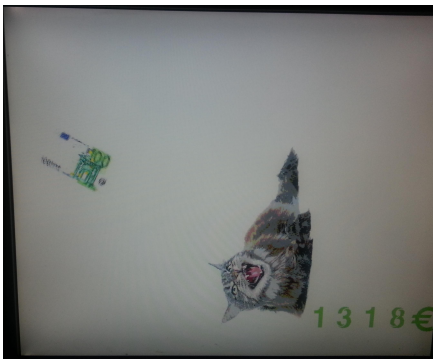
1.1 Opis



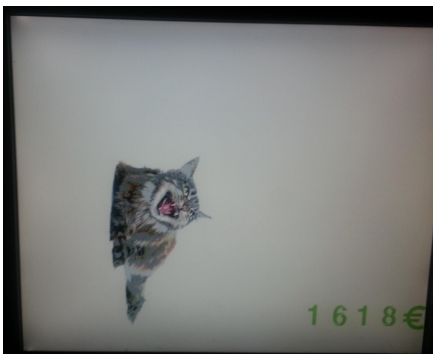
Slika 1: Slika na zaslonu mačka ki lovi denar

Namen igre je lovljenje denarja. Denar se obrača vedno hitreje dokler ne izgine. Hitrost obračanja nakazuje koliko časa bo denar čakal preden se bo pojavil drugje. Če ulovimo denar preden izgine, se nemudoma pojavi nova priložnost za dobiček.

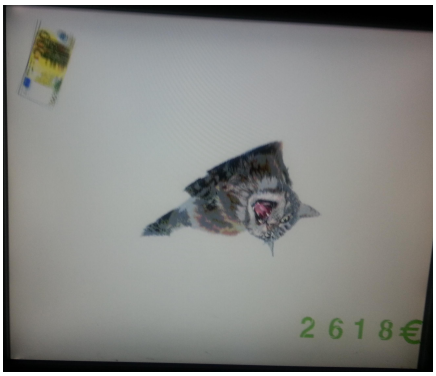
Igra vsebuje več različnih bankovcev: bankovec za 50€, 100€, in 200€.
Položaj mačka se kontrolira z tipkami na ploščici. Ko pritisnemo na tipko se maček premakne v določeno smer in se pri tem obrne v to smer. Če mačka potisnemo iz ekrana, se bo pojavil na nasprotni strani.
Igra vsebuje tudi števec denarja ki pa zaradi velikosti RAM-pomnilnika ni pravilno implementiran. Denar vedno šteje tudi če ga ne ujamemo.



Slika 2: Na ploščici držimo 1. tipko, maček se premika v levo



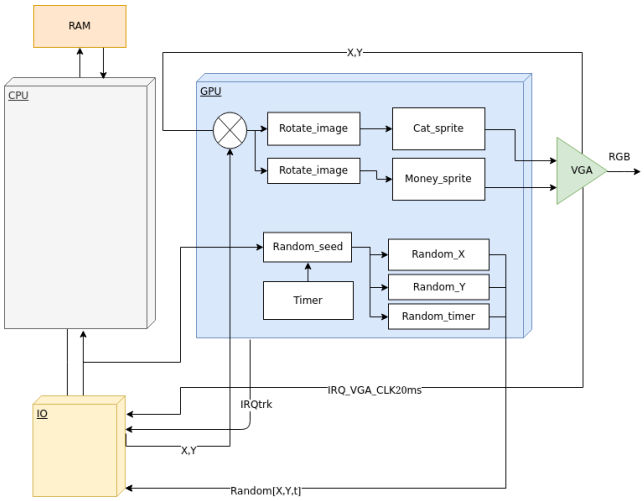
Slika 3: Na ploščici držimo 4. tipko, maček se premika v desno



Slika 4: Na ploščici držimo 3. in 4. tipko, maček se premika diagonalno

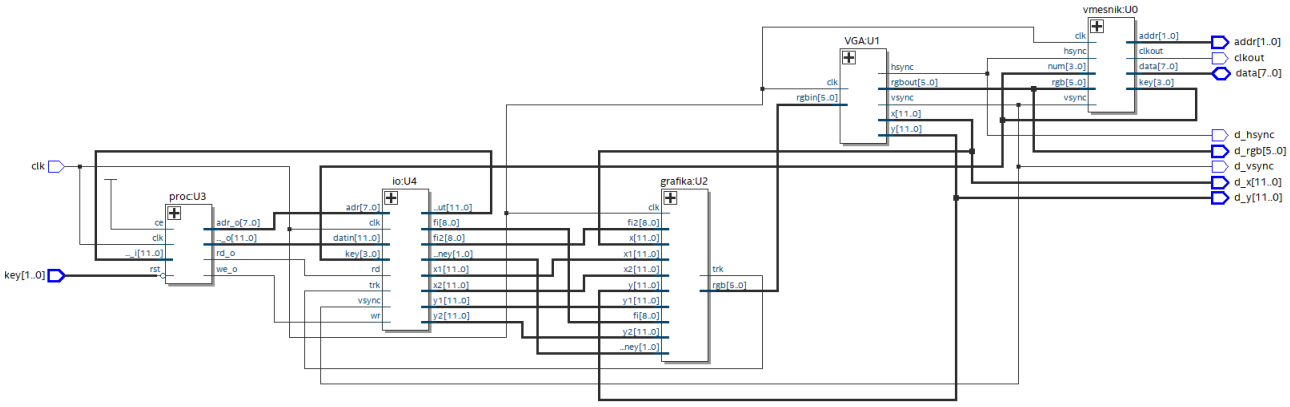
2 Opis Top modela

2.1 Opis



Slika 5: Blok shema projekta

Igro sestavlja procesor in grafičen vmesnik. Koda katera se lahko izvaja počasi se izvaja v procesorju, ki krmari grafično enoto. Celoten GPU je narejen v logiki, ker zagotavlja potrebno hitrost katero se ne da doseči z procesorjem. GPU je sestavljen iz 2 prikazovalnikov sprit-ov od tega je vsak sposoben rotacije po zaslonu. Dodatno sem dodal tri pseudo naključne generatorje in naključne seed generator. Procesor je ostal nespremenjen, z izjemo da ima RAM dodan kot IP komponento, kar omogoča nalaganje programa v procesor brez ponovnega prevajanja sistema. Dodatna sprememba je še da ima vga-core 2-bitno resolucijo na barvo.

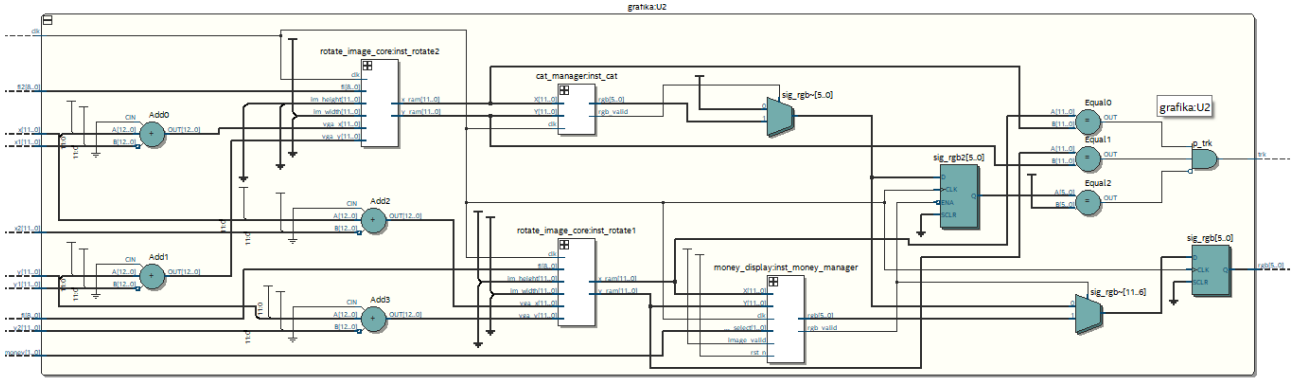


Slika 6: Top level sistem

3 Pomembna jedra

3.1 Risanje po zaslonu

Risanje izvajata dva modula money manager in cat manager. Oba bloka imata dodan dodaten core za rotacijo.



Slika 7: Shema grafike

Grafiko sestavljat dva modula za transformacijo slike: ”Add0”, ”Add1” in ”Add2”, ”Add3”. Transformirani koordinati ”vgaX”, ”vgayY” so vhodni parametri pri jedru za rotacijo slike Izsek jedra spodaj . Izhod jedri sta ”X”,”Y” koordinati, ki se uporablja za vpogled v ROM ”moneyManager”, ”catManager” , izhodni slik obeh jeder gresta v multiplekser, ki daje prioriteto risanju rezultata, denarja, mačka kasneje risanju ozadja.

3.2 Vhodi in izhodi v procesor

Glaven namen jedra je komunikacija med procesorjem in logiko. V logiki sem določil stalne naslove, na katere procesor lahko piše in bere iz njih. Na ta način procesor upravlja z logiko.

Naslov	READ	Opis
0	V_SYNC_INT	Prekinitev ki se postavi na 1 vsakih 20ms, in ostane dokler je ne preberemo
1	KEY0	Gumb na plošči premik mačka levo
2	KEY1	Gumb na plošči premik mačka navzgor
3	KEY2	Gumb na plošči premik mačka navzdol
4	KEY3	Gumb na plošči premik mačka desno
5	TRK	Prekinitev ki se postavi na 1 ko maček trči ob denar
6	RANDOM_X	Naključne kordinate za X
7	RANDOM_Y	Naključne kordinate za Y
8	RANDOM_TIME	Naključen čas življenja prikazanega denarja

Naslov	WRITE	Opis
0	X0	X kordinata mačka
1	y0	Y kordinata mačka
2	X1	X kordinata denarja
3	y0	Y kordinata denarja
4	FI	Kot zasuka denarja
5	SEL_MONEY	Izbira bankovca, ki bo prikazan
6	FI2	Kot maćka
7	Score	Ujet denar

Slika 8: Vhodi in izhodi v procesor

4 Zanimivi izseki kode

4.1 Rotacija slike

```
entity rotate_image is
    port(clk : in std_logic;
        x_im, y_im :in signed(11 downto 0) :=  to_signed(100,12);
        fi : in unsigned(8 downto 0) := (others => '0');
        x_ram, y_ram : out signed(11 downto 0));
end rotate_image;

architecture Behavioral of rotate_image is
    type t_ROM is array(0 to 90) of unsigned(11 downto 0);
    signal SIN_ROM : t_ROM := (--table of sinus);

    signal sin_angle, cos_angle : signed(11 downto 0);
    signal tmp_x_ram, tmp_y_ram : signed(23 downto 0);
begin

p_calc : process(all)
begin
        if rising_edge(clk) then
                --calculate cos(fi) and sin(fi)
                if(fi<91) then
                        sin_angle <= signed(SIN_ROM(to_integer(fi)));
                        cos_angle <= signed(SIN_ROM(90-to_integer(fi)));
                elsif((fi > 90) and (fi < 181)) then
                        sin_angle <= signed(SIN_ROM(180-to_integer(fi)));
                        cos_angle <= -signed(SIN_ROM(to_integer(fi)-90));
                elsif((fi >180) and (fi < 271)) then
                        sin_angle <= -signed(SIN_ROM(to_integer(fi)-180));
                        cos_angle <= -signed(SIN_ROM(270-to_integer(fi)));
                elsif((fi >270) and (fi < 361)) then
                        sin_angle <= -signed(SIN_ROM(360-to_integer(fi)));
                        cos_angle <= signed(SIN_ROM(to_integer(fi)-270));
                else
                        sin_angle <= signed(SIN_ROM(0));
                        cos_angle <= signed(SIN_ROM(90));
                end if;

                tmp_x_ram <= x_im*cos_angle + sin_angle*y_im;
                tmp_y_ram <= -x_im*sin_angle + y_im*cos_angle;

                x_ram <= tmp_x_ram(21 downto 10);
                y_ram <= tmp_y_ram(21 downto 10);
        end if;
end process;
end Behavioral;
```

4.2 Random number generator

```
entity lsfr is
    generic(MIN_OUT : integer := 20;
            MAX_OUT : integer := 50);
    Port (clk, rst_n : in std_logic;
          seed : in std_logic_vector(9 downto 0);
          seed_valid : in std_logic;
          random : out std_logic_vector(9 downto 0);
          random_valid : out std_logic);
end lsfr;
architecture Behavioral of lsfr is
    constant WIDTH : integer := 10;
    signal is_init : std_logic := '0';
    signal init_seed : std_logic_vector(WIDTH-1 downto 0);
    signal is_init_r0 : std_logic := '0';
    signal counter : unsigned(4 downto 0) := (others => '0');

    constant PERIOD_LSFR : integer := 17;
begin
p_delay : process(all)
begin
    if(rising_edge(clk)) then
        is_init_r0 <= is_init;
    end if;
end process;

p_random : process(all)
begin
    if (rst_n = '0') then
        is_init <= '0';
    elsif rising_edge(clk) then
        if(seed_valid = '1' or is_init = '1') then
            init_seed <= seed;
            is_init <= '1';
        end if;
        if(is_init_r0 = '1') then
            init_seed <= init_seed(WIDTH-2 downto 0) & (init_seed(9) xor init_seed(6));
            if(counter = PERIOD_LSFR-1) then
                counter <= (others => '0');
                random <= std_logic_vector((unsigned(init_seed) mod (MAX_OUT-MIN_OUT+1)) + MIN_OUT);
                random_valid <= '1';
            else
                random_valid <= '0';
                counter <= counter + 1;
            end if;
        end if;
    end if;
end process;
```

4.3 Program

```
int* p_sel_money = 5;
int* p_clk50 = 0;
int* p_key_up = 2;
int* p_key_down = 3;
int* p_key_left = 1;
int* p_key_right = 4;
int* p_rand_time = 8;
int* p_rand_x = 6;
int* p_rand_y = 7;
int* p_trk = 5;

int* p_money_x = 2;
int* p_money_y = 3;
int* p_cat_x = 0;
int* p_cat_y = 1;
int* p_fi = 4;
int* p_fi2 = 6;
int* p_score = 7;

int score = 0;
int rand_time = 0;
int money_x = 0;
int money_y = 0;
int cat_x = 100;
int cat_y = 100;
int sel_money = 0;
int fi = 0;
int fi2 = 0;

int var_fi = 0;
int prec_fi = 0;

main(){
    while(1){
        *p_money_x = money_x;
        *p_money_y = money_y;

        *p_cat_x = cat_x;
        *p_cat_y = cat_y;

        if(*p_key_up == 1){
            if(cat_y > 0){
                cat_y--;
                if(fi2 > 0){
                    fi2--;
                }
                if(fi2>180){
                    fi2++;
                }
            } else {
                cat_y = 450;
            }
        }
        if(*p_key_down == 1){
            if(cat_y < 500){
                cat_y++;
                if(fi2 < 179){
                    fi2++;
                }
                if(fi2>180){
                    fi2--;
                }
            } else {
                cat_y = 0;
            }
        }
```

```
        if(*p_key_left == 1){
            if(cat_x > 0){
                cat_x--;
                if(fi2 < 269){
                    fi2++;
                }
                if(fi2>270){
                    fi2--;
                }
            } else {
                cat_x = 650;
            }
        }
        if(*p_key_right == 1){
            if(cat_x < 700){
                cat_x++;
                if(fi2 < 89){
                    fi2++;
                }
                if(fi2>90){
                    fi2--;
                }
            } else {
                cat_x = 0;
            }
        }

            if(rand_time < 10){
                rand_time = *p_rand_time;
                sel_money++;
                money_x = *p_rand_x;
                money_y = *p_rand_y;
                *p_sel_money = sel_money;


            }
        if(sel_money > 3){
            sel_money = 1;
        }

        rand_time--;

        var_fi = 1111 - rand_time;
        var_fi  = var_fi >> 1;
        var_fi  = var_fi >> 1;
        var_fi  = var_fi >> 1;

        prec_fi = prec_fi + var_fi;
        while(prec_fi > 100){
            fi++;
            prec_fi = prec_fi - 100;
        }
        if(fi > 360){
            fi = 0;
        }

        if(*p_trk == 1){
            if(sel_money == 1){
                score = score + 50;
            }
            if(sel_money == 2){
                score = score + 100;
            }
            if(sel_money == 3){
                score = score + 200;
            }
            rand_time = *p_rand_time;
            *p_sel_money = 0;
        }
        *p_fi = fi;
        *p_fi2 = fi2;
        *p_score = score;
        while(*p_clk50 == 0){}
    }
}
```

5 Povzetek sinteze

Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins
|sistem	17932 (1)	705 (0)	168560	16	0	8	46	0
|VGA:U1|	55 (55)	23 (23)	0	0	0	0	0	0
|grafika:U2|	17225 (99)	284 (12)	165488	16	0	8	0	0
|cat_manager:inst_cat|	13952 (33)	30 (30)	0	0	0	0	0	0
|cat_360p:inst_rom|	13882 (13882)	0 (0)	0	0	0	0	0	0
|lpm_mult:Mult0|	37 (0)	0 (0)	0	0	0	0	0	0
|multcore:mult_core|	37 (21)	0 (0)	0	0	0	0	0	0
|mpar_add:padder|	16 (0)	0 (0)	0	0	0	0	0	0
|disp_score_core:inst_score|	1302 (128)	100 (38)	24368	0	0	0	0	0
|Eur_logo:inst_eur|	45 (17)	0 (0)	2968	0	0	0	0	0
|bin2bcd:inst_bcd|	68 (68)	0 (0)	0	0	0	0	0	0
|dsip_image:\gen_digit:0:inst_disp|	288 (205)	15 (15)	0	0	0	0	0	0
|lpm_mult:Mult0|	40 (0)	0 (0)	0	0	0	0	0	0
|lpm_mult:Mult1|	43 (0)	0 (0)	0	0	0	0	0	0
|dsip_image:\gen_digit:1:inst_disp|	256 (256)	15 (15)	0	0	0	0	0	0
|dsip_image:\gen_digit:2:inst_disp|	257 (257)	15 (15)	0	0	0	0	0	0
|dsip_image:\gen_digit:3:inst_disp|	257 (257)	15 (15)	0	0	0	0	0	0
|number:inst_number|	3 (0)	2 (0)	21400	0	0	0	0	0
|altsyncram:ROM_rtl_0|	3 (0)	2 (0)	21400	0	0	0	0	0
|money_display:inst_money_manager|	98 (98)	2 (2)	141120	0	0	0	0	0
|rotate_image_core:inst_rotate1|	891 (67)	70 (0)	0	8	0	4	0	0
|rotate_image_core:inst_rotate2|	883 (55)	70 (0)	0	8	0	4	0	0
|io:U4|	190 (52)	160 (93)	0	0	0	0	0	0
|lsfr:inst_lsfr_time|	74 (25)	27 (27)	0	0	0	0	0	0
|lsfr:inst_lsfr_x|	33 (12)	20 (20)	0	0	0	0	0	0
|lsfr:inst_lsfr_y|	31 (10)	20 (20)	0	0	0	0	0	0
|proc:U3|	220 (0)	97 (0)	3072	0	0	0	0	0
|CPU:c1|	157 (157)	56 (56)	0	0	0	0	0	0
|program:c0|	63 (0)	41 (0)	3072	0	0	0	0	0
|RAM:RAM_inst|	63 (0)	41 (0)	3072	0	0	0	0	0
|sld_hub:auto_hub|	121 (1)	87 (0)	0	0	0	0	0	0
|vmesnik:U0|	120 (120)	54 (54)	0	0	0	0	0	0

Slika 9: Resource Utilization by Entity

Slike denarja, in slike številk so implementirane z RAM-mom slika mačka pa je prevelika za 500k RAMa v FPGA-ju. Rešitev katero sem uporabil je, da je slika shranjena z LUT celicami. To je razvidno iz sinteze, saj je poraba LUTOV skoraj 14000.

6 Programi, za hitrejši potek

6.1 Pretvorba slike v ROM

```
from scipy import misc
import matplotlib.pyplot as plt
import numpy as np
import math
import sys
WRITE_FILE = 0
DISPL_IMAGE = 1
INPUT_FILE = sys.argv[1]
OUTPUT_FILE = INPUT_FILE.split(".")[0] + ".vhd"

if(len(sys.argv) != 2):
  print("enter input file name [image.jpg]")
  sys.exit()

  ##########Preverjanje vnosa##########
  face = misc.face()
  face = misc.imread(INPUT_FILE)

  height,width,tmp = face.shape
  pixel = np.zeros((height,width,3))
  pixel = np.copy(face)

  for h in range(height):
    for w in range(width):
      pixel[h][w] =  np.divide(pixel[h][w],64)

      ##########Deli VHDL kode potrebni za generacijo ROM datoteke ##########
  str_lib = '''library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

  entity ''' + INPUT_FILE.split(".")[0] + ''' is
  port ( clk : in std_logic;
  addr : in unsigned('''+ str(math.floor(math.log(width*height-1,2))) + ''' downto 0);
  rgb : out std_logic_vector(5 downto 0))
  ;
  end ''' + INPUT_FILE.split(".")[0] + '''; '''
  arch_str1 = '''architecture Arch of ''' + INPUT_FILE.split(".")[0] + ''' is'''
  arch_str2 = '''
  begin
  p_ROM : process(clk)
  begin
  if rising_edge(clk) then
    rgb <= ROM(to_integer(addr));
  end if;
  end process;
  end Arch;'''
  str_type = '''type t_ROM is array(0 to " + str(width)+"*"+str(height) + "-1" + ") of
    std_logic_vector(2*3-1 downto 0);'''
  str_value = "signal ROM : t_ROM := ("
  ###############Pisanje v datoteko##########
  if(WRITE_FILE):
    fp = open(OUTPUT_FILE,"w")
    fp.write(str_lib + "\n")
    fp.write(arch_str1 + "\n")
    fp.write(str_type + "\n")
    fp.write(str_value + "\n")
    for h in range(height):
      for w in range(width):
        fp.write("std_logic_vector(to_unsigned(" + str(int(pixel[h][w][0])) + ",2) &" +
        "to_unsigned(" + str(int(pixel[h][w][1])) + ",2) &" +
        "to_unsigned(" + str(int(pixel[h][w][2])) + ",2)), \n" )
        fp.write("others => (others => '0') \n")
        fp.write(');')
  fp.write(arch_str2 + "\n")
  ####prikaz 3*2bit/pixel slike###########
  if(DISPL_IMAGE):
    for h in range(height):
      for w in range(width):
        pixel[h][w] = np.add(np.multiply(pixel[h][w],85),0);
      plt.imshow(pixel)
      plt.show()
```

6.2 Primer pretvorbe enostavne slike z zgornjo kodo

Slika 10: Primer vhodne slike

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity eur_10p is
port ( clk : in std_logic;
        addr : in unsigned(2 downto 0);
        rgb : out std_logic_vector(5 downto 0))
  ;
end eur_10p;
architecture Arch of eur_10p is
type t_ROM is array(0 to 3*2-1) of std_logic_vector(2*3-1 downto 0);
signal ROM : t_ROM := (
std_logic_vector(to_unsigned(1,2) &to_unsigned(1,2) &to_unsigned(0,2)),
std_logic_vector(to_unsigned(0,2) &to_unsigned(0,2) &to_unsigned(0,2)),
std_logic_vector(to_unsigned(0,2) &to_unsigned(0,2) &to_unsigned(1,2)),
std_logic_vector(to_unsigned(2,2) &to_unsigned(2,2) &to_unsigned(0,2)),
std_logic_vector(to_unsigned(0,2) &to_unsigned(0,2) &to_unsigned(0,2)),
std_logic_vector(to_unsigned(3,2) &to_unsigned(3,2) &to_unsigned(3,2)),
others => (others => '0')
);
begin
        p_ROM : process(clk)
        begin
            if rising_edge(clk) then
                rgb <= ROM(to_integer(addr));
            end if;
        end process;
end Arch;
```

Python koda regenerira vhd datoteko ki vsebuje vsebino RGB sliko z dvema bitoma resolucije na barvo. Potrebna velikost ROMa se avtomatsko prilagodi na velikost vhodne slike, velikost je v tem primeru 2 pixla visoka in 3 pixle široka. Velikost naslova v rom se tudi sama nastavlja na minimalno dolžino, da zmore nasloviti ves ROM.

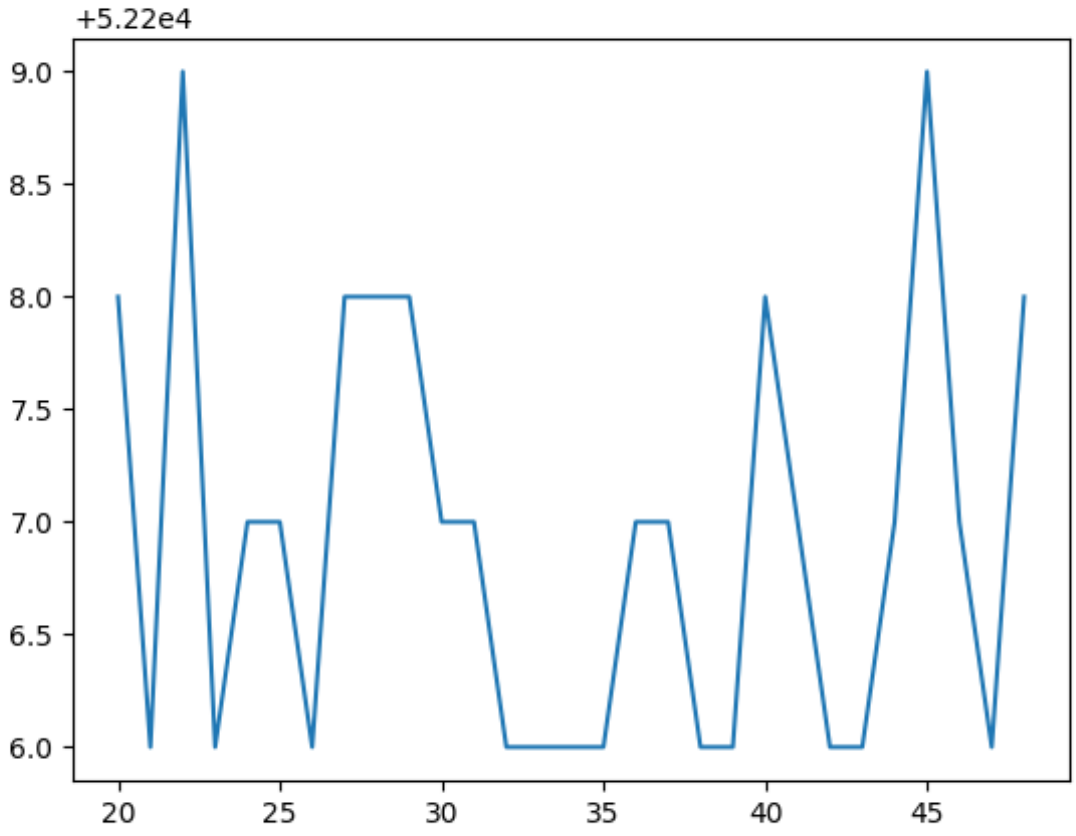
6.3 Verifikacija jeder

Primer verifikacije naključnega generatorja, jedro regenerira števila od 20 do 50. Najprej sem jedru "lfsr" dodal modul za testiranje(testbench). Namen tega je, da stimulira vhode v "lfsr" in zajema izhode iz jedra. Izhodi se zapisujejo v datoteko "output.hex".

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use STD.TEXTIO.ALL;

entity lfsr_tb is
end lfsr_tb;

architecture Behavioral of lfsr_tb is
    signal clk,rst_n : std_logic := '0';
    signal seed_valid : std_logic := '0';
    signal seed : std_logic_vector(9 downto 0) := std_logic_vector(to_unsigned(75,10));

    signal random : std_logic_vector(9 downto 0);
    signal random_valid : std_logic;
    constant CLK_PERIOD : time := 10ns;
begin

uut : entity work.lsfr
    Port map(clk => clk, rst_n => rst_n,
          seed => seed,
          seed_valid => seed_valid,
          random => random,
          random_valid => random_valid);
p_test : process
begin
    clk <= '0';
    wait for CLK_PERIOD/2;
    clk <= '1';
    wait for CLK_PERIOD/2;
end process;

p_test2 : process
begin
    wait for CLK_PERIOD*5;
    rst_n <= '1';
    seed_valid <= '1';
    wait for CLK_PERIOD;
    seed_valid <= '0';
    wait for CLK_PERIOD*10;
    wait;
end process;
p_write : process(random_valid)
    variable L : LINE;
    file outfile : text is out "output.hex";
begin
    if(random_valid = '1') then
        write(L,to_integer(unsigned(random)));
        writeline(outfile,L);
    end if;
end process;
end Behavioral;
```

Jedro simuliram toliko časa dokler ne dobim dovoli izhodnih podatkov. V tem primeru sem to počel 50ms. Z tem sem dobil 1.5 milijona naključnih števil. Za prikaz podatkov sem napisal nov program v pythonu ki sortira, prešteje in prikaže posamezne vrednosti. Te vrednosti, izrišem na graf, iz katerega lahko ocenim delovanje jedra.

```
from scipy.stats import binned_statistic
import matplotlib.pyplot as plt
import numpy as np
import sys

INPUT_FILE = sys.argv[1]

#####preveri veljaveno datoteko#######
if(len(sys.argv) != 2):
    print("enter input file name [image.jpg]")
    sys.exit()
data = np.fromfile(INPUT_FILE, dtype=int, count=-1, sep=" ")

min_value = np.amin(data)
max_value = np.amax(data)
bin_cnt = max_value-min_value

#sortiraj podatke in jih izriši
bin_means = binned_statistic(data, data, statistic='count', bins=bin_cnt, range=(min_value, max_value))
plt.plot(bin_means[1][0:bin_cnt-1],bin_means[0][0:bin_cnt-1])
plt.show()
```



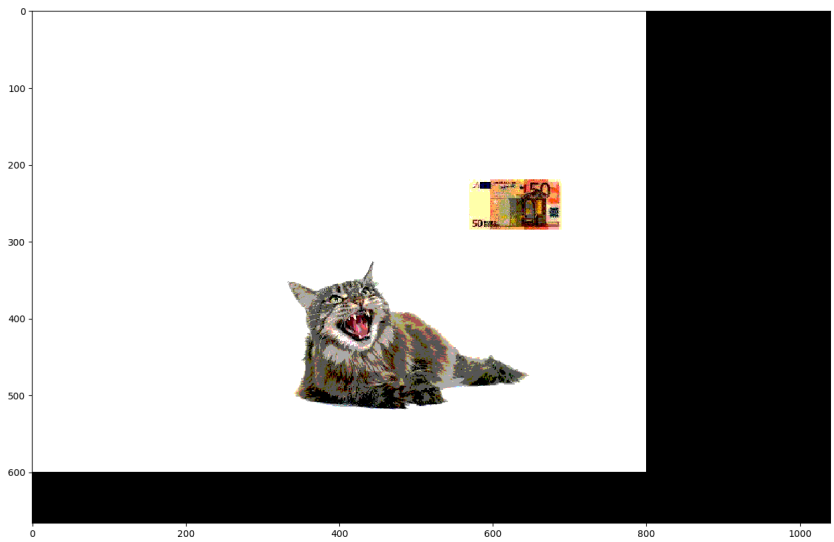
Slika 11: Pregled naključnega generatorja, na x-osi so vrednosti na y-osi je pogostost vrednosti

Ker je bil celoten projekt relativno velik je bil čas prevajanja tudi dalši kot pri manših projektih. Čeprav sem vsako jedro lahko simuliral na FPGA-ju relativno hitro, je razhroščevanje celega projekta predstavljalo težave. Rešitev katero sem uporabil je, da sem simuliral celoten sistem. Kot prej sem napisal testbench za sistem, kateremu sem dodal še nekaj dodatnih izhodov.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use STD.textio.all;
use ieee.std_logic_textio.all;

entity sistem_tb is
end sistem_tb;
architecture Behavioral of sistem_tb is
    signal clk : std_logic;
    signal key : std_logic_vector(1 downto 0) := "11";
    signal clkOut : std_logic;
    signal addr : std_logic_vector(1 downto 0);
    signal data : std_logic_vector(7 downto 0);

    --debug
    signal d_rgb : std_logic_vector(5 downto 0);
    signal d_hsync,d_vsync : std_logic;

    signal x_pos,y_pos : unsigned(11 downto 0);
    file file_out : text;
    constant CLK_PERIOD : time := 20ns;
begin
p_clk : process
begin
    clk <= '1';
    wait for CLK_PERIOD/2;
    clk <= '0';
    wait for CLK_PERIOD/2;
end process;

p_write_file : process(clk)
    variable v_OLINE : line;
    variable is_opend : integer := 0;
    variable write_cnt : integer := 0;
    variable done : integer := 0;
    variable frame_skip : integer := 2;
begin
    if(rising_edge(clk)) then
        if(x_pos = 0 and y_pos = 0 and done = 0) then
            frame_skip := frame_skip-1;
            if frame_skip=1 then
                file_open(file_out, "out.hex",write_mode);
                is_opend := 1;
            end if;
        end if;
   if(is_opend = 1) then
        write_cnt := write_cnt + 1;
        write(v_OLINE, to_integer(unsigned(d_rgb)));
        writeline(file_out, v_OLINE);
    end if;
    if(write_cnt = 1040*666) then
        file_close(file_out);
        is_opend := 0;
        done := 1;
    end if;
    end if;
end process;
uut : entity work.sistem
        Port map( clk => clk,
                  key => key,
                       clkout => clkOut,
                       addr => addr,
                       data => data,
                       d_rgb => d_rgb,
                       d_hsync => d_hsync,
                       d_vsync => d_vsync,
                       UNSIGNED(d_x) => x_pos,
                       UNSIGNED(d_y) => y_pos
                );
end Behavioral;
```

Testbench spravi RGB vrednost v izhodno datoteko "out.hex". Katero python skripta prebere in izriše sliko iz teh podatkov.

```
from scipy import misc
import matplotlib.pyplot as plt
import numpy as np
import math
import sys

INPUT_FILE = sys.argv[1]
WIDTH = 1040
HEIGHT = 666

if(len(sys.argv) != 2):
    print("enter input file name [out.hex]")
    sys.exit()
##########
pixel = np.array(np.zeros([HEIGHT,WIDTH,3]))
raw_data = np.fromfile(INPUT_FILE,dtype=int, sep='\n')
i = 0
for h in range(HEIGHT):
    for w in range(WIDTH):
        pixel[h][w] = [int((raw_data[i]>>4))*85, int(((raw_data[i] & 0b1100) >>2))*85, int((raw_data[i] & 0b11)*85)]
        i+=1
pixel.astype(float)
plt.imshow(np.divide(pixel,255))
plt.show()
```



Slika 12: Rekonstrukcija slike, generirana iz podatkov pri simulaciji

Na ta način sem lahko simuliral celoten sistem v manj kot minuti. Implementacija na FPGA-ju traja približno 30 minut, kar bi močno upočasnilo razvoj sistema.