Univerza v Ljubljani
Fakulteta za elektrotehniko

Aljaž Podobnik

Projektna naloga

Poročilo pri predmetu Digitalna integrirana vezja in sistemi

Mentor: izr. prof. dr. Andrej Trost

Ljubljana, 2016

Uvod

Za projektno nalogo sem se odločil, da na razvojnem sistemu ZedBoard izdelam igro Pong. Pong je ena izmed najstarejših arkadnih video iger in prva športna arkadna video igra. Gre za simulacijo tenisa s preprosto dvodimenzijsko grafiko.

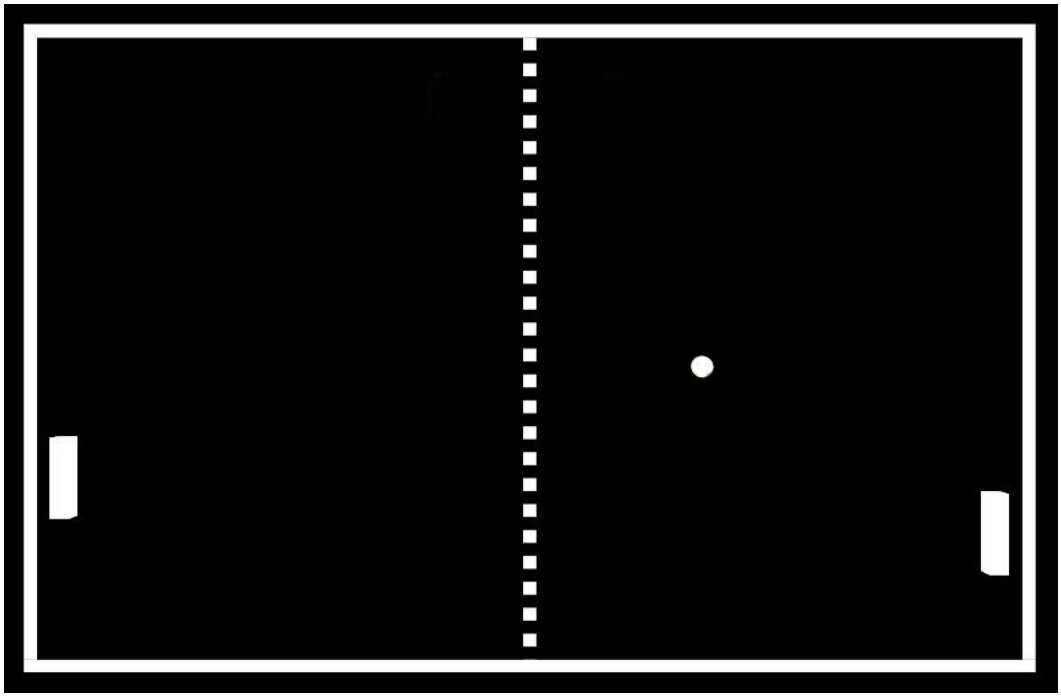
Igra je na razvojnem sistemu ZedBoard zasnovana tako, da za celoten izris slike na VGA monitorju, tako fiksnih kot tudi gibajočih objektov in ozadja skrbi FPGA, procesor pa sproti računa pozicije gibajočih objektov glede na pritisnjene tipke in potek same igre ter jih vpisuje v prave registre.

Strojni del aplikacije, torej del, ki se nanaša na FPGA, je sprogramiran v strojno opisnem jeziku VHDL v programskem okolju Vivado, programski del aplikacije, ki računa pozicije gibajočih objektov pa je sprogramiran v programskem jeziku C v okolju Software Development Kit ali SDK.

Podrobnosti o izdelavi projektne naloge bodo opisane po posameznih poglavjih v nadaljevanju.

Opis komponente Objects

Za igro Pong lahko na spletu najdemo mnogo različnih dizajnov dvodimenzijske grafike, vsi pa imajo podobno osnovo. Imamo levi in desni igralni plošček, enega premika igralec, drugega premika računalnik ali drugi igralec. Zgornji in spodnji rob sta vedno prisotna, saj se žoga od njiju odbija z upoštevanjem vpadnega kota. Levi in desni rob sta lahko prisotna ni pa nujno, saj se žoga od njiju, v primeru, da je igralec ne uspe prestreči, ne odbije. Na sliki Slika *1* je prikazan eden izmed možnih dizajnov igre Pong, osnovni princip je vedno enak.



Slika 1:Eden izmed dizajnov igre Pong

Da bi dosegel takšen dizajn, sem v programskem okolju Vivado ustvaril novo komponento. Komponento Objects sem izdelal po vzoru komponente Krog, ki smo jo izdelali pri eni izmed laboratorijskih vaj pri tem predmetu. Komponenta Objects ima poleg vodila AXI še nekaj zunanjih priključkov in sicer eno biten vhod en, 6 biten vhod rgbin, 18 biten vhod xy in 6 biten izhod rgb. Z vhodom en lahko delovanje komponente vklopimo ali izklopimo. V primeru, da je en postavljen na logično ničlo, je funkcija komponente Objects izklopljena, vhod rgbin pa se direktno preslika na izhod rgb. Če je en postavljen na logično enko, se najprej izračunajo pogoji za izris objektov izhod rgb pa se določi glede na izpolnjenost teh pogojev. Zgoraj opisano delovanje komponente sem dosegel z uporabo enega procesnega stavka, v katerem se preverijo vsi pogoji in se nato glede na veljavnost teh pogojev izhod rgb postavi na določeno vrednost. Na sliki Slika *2* se nahaja izsek kode VHDL, ki vsebuje procesni stavek za določitev izhoda rgb.

```
174 p1: process(s0_axi_aclk) --*************** PROCESS ********************************
175     begin
176         if rising_edge(s0_axi_aclk) then
177             --izračunamo kvadratno vrednost koordinat
178             x2 <= (x_signed-xc_signed) * (x_signed-xc_signed);
179             y2 <= (y_signed-yc_signed) * (y_signed-yc_signed);
180 
181             if  (
182                     (
183                         (
184                             --pogoj za sredinsko črto
185                             (x_signed>=-5 and x_signed<=4)
186                         ) or
187                         (
188                             --pogoj za levo in desno črto
189                             (x_signed>=-256 and x_signed<=-254) or
190                             (x_signed>=253 and x_signed<=255)
191                         ) or
192                         (
193                             --pogoj za zgornjo in spodnjo črto
194                             (y_signed>=-256 and y_signed<=-247) or
195                             (y_signed>=246 and y_signed<=255)
196                         ) or
197                         (
198                             --pogoj za levi pravokotnik
199                             (x_signed>=-256 and x_signed<=-247) and
200                             (y_signed>=(-46+yc1_signed) and y_signed<=(45+yc1_signed))
201                         ) or
202                         (
203                             --pogoj za desni pravokotnik
204                             (x_signed>=246 and x_signed<=255) and
205                             (y_signed>=(-46+yc2_signed) and y_signed<=(45+yc2_signed))
206                         ) or
207                         (
208                             --pogoj za krog
209                             (x2 + y2 <= r2)
210                         )
211                     ) and en='1'
212                 ) then
213                 rgb <= "111111";
214             else
215                 rgb <= rgbin;
216             end if;
217         end if;
218     end process;--******************** END PROCESS ********************************
```

Slika 2: Izsek VHDL kode s procesnim stavkom za izris objektov

V procesnem stavku najprej izračunamo kvadratno vrednost koordinat z upoštevanjem koordinat centra kroga kar potrebujemo kasneje pri računanju enačbe kroga za izris žoge. Nato sledi pogojni stavek kjer so zapisani pogoji za izris vseh objektov. Iz komentarjev v kodi je razvidno kateri pogoj je odgovoren za izris določenega objekta. Pri računanju enačbe kroga upoštevamo, da je njegovo središče lahko izmaknjeno izven koordinatnega izhodišča, ki se nahaja v centru VGA zaslona, saj se žoga lahko premika po celotnem igralnem polju. Koordinate centra žoge se računajo v programu, ki je spisan v jeziku C, in se vpisujejo v register do katerega lahko dostopamo preko vodila AXI. Iz tega registra nato izluščimo x in y koordinati centra žoge ter pretvorimo podatkovni tip iz tipa std_logic_vector v signed, da se koordinate centra žoge pri izrisu pravilno upoštevajo. To naredimo z nekaj vrsticami kode.

```
xc<=xcyc(17 downto 9);

yc<=xcyc(8 downto 0);

xc_signed<=signed(xc);

yc_signed<=signed(yc);
```

Seveda so bili vsi signali že vnaprej deklarirani kot notranji signali ustreznega podatkovnega tipa.

Pogoja za izris levega in desnega pravokotnika, torej levega in desnega igralnega ploščka pa sta odvisna le od centralne pozicije po y osi, saj se igralna ploščka vzdolž x osi ne premikata. Centralni y poziciji obeh ploščkov se prav tako sproti računata v C programu in preko AXI vodila vpisujeta v notranji register. Centralni poziciji obeh ploščkov izluščimo na enak način kot prej.

```
yc1<=yc1yc2(17 downto 9);

yc2<=yc1yc2(8 downto 0);

yc1_signed<=signed(yc1);

yc2_signed<=signed(yc2);
```

Signale za koordinate točke x in y dobimo iz vhodnega vektorja xy in sicer zgornjih 9 bitov za koordinato x in spodnjih 9 bitov za koordinato y. Nato spremenimo še podatkovni tip za pravilno upoštevanje pri določanju pogojev za izris objektov.

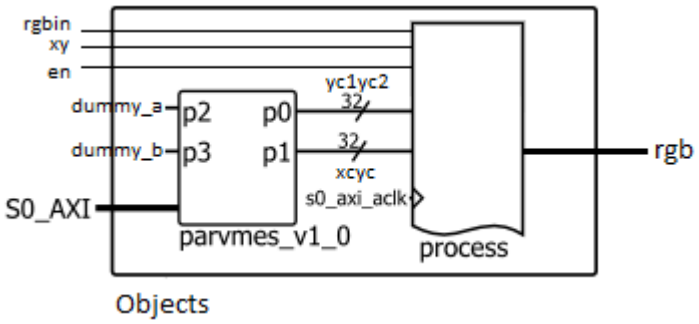
```
x<=xy(17 downto 9);

y<=xy(8 downto 0);

x_signed<=signed(x);

y_signed<=signed(y);
```

Komponenta Objects torej deluje tako, da izhod rgb postavi na vrednost "111111" če je signal en postavljen na '1' in če je izpolnjen katerikoli izmed pogojev za izris objekta, sicer dobi izhod rgb vrednost vhoda rgbin.

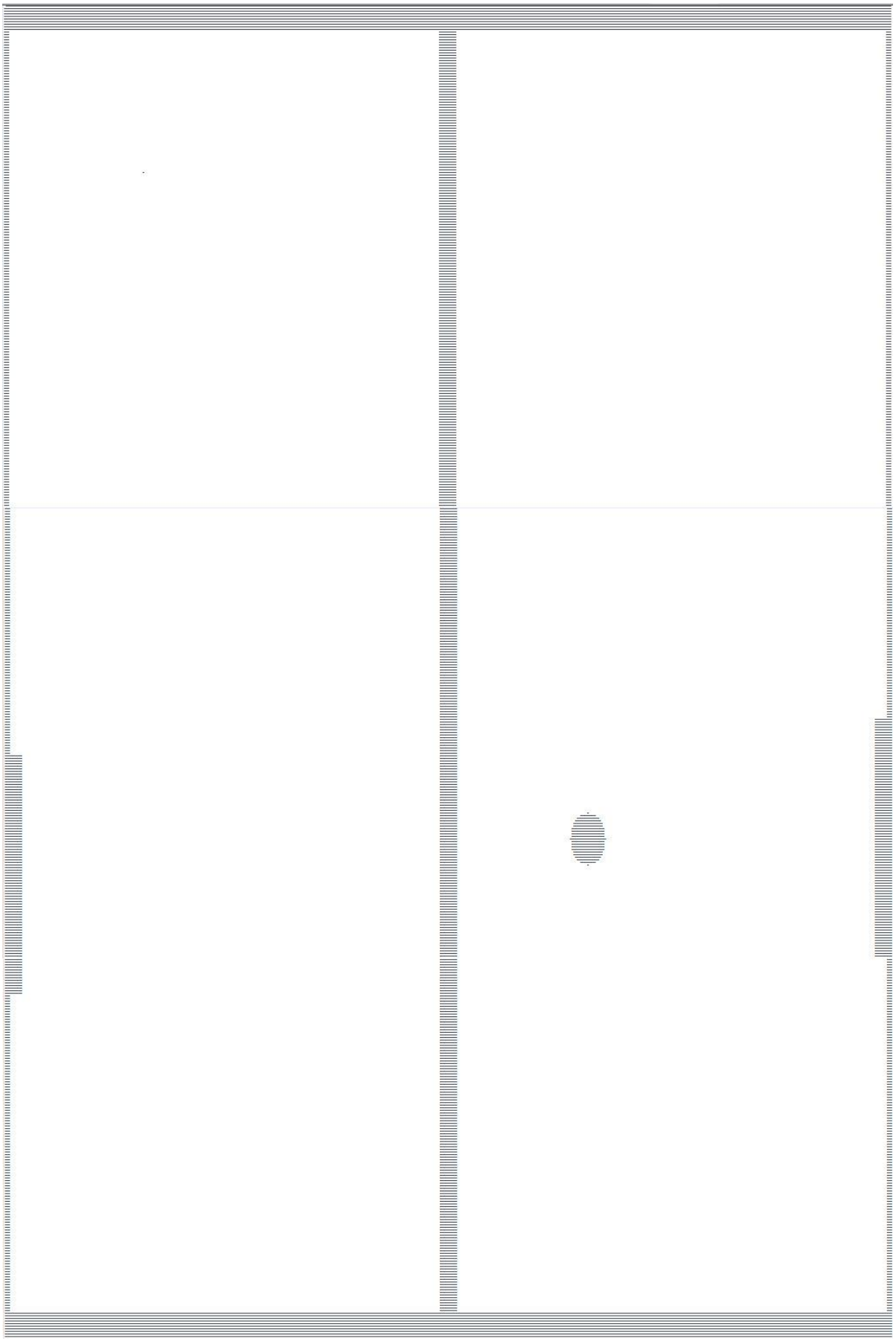


Slika 3: Shema komponente Objects

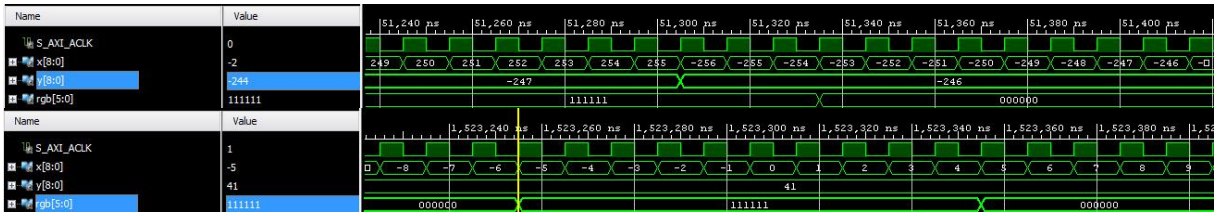
Na sliki Slika *3* je prikazana shema komponente Objects. Iz sheme lahko vidimo da smo na vhoda p2 in p3 paralelnega vmesnika vezali signala dummy_a in dummy_b. Ta dva signala v komponenti Objects nimata nobene funkcije in sta na vhod notranjih registrov vodila AXI vezana zgolj zato, da vhoda p2 in p3 ne ostaneta neprikljućena. Na izhod p0 je vezan notranji signal yc1yc2 iz katerega izluščimo y centralni poziciji obeh igralnih ploščkov, na izhod p1 pa je vezan notranji signal xcyc iz katerega izluščimo x in y koordinati za center žoge. Izhod rgb gre nato direktno na VGA zaslon, kot bomo to videli kasneje pri opisu celotnega sistema.

Delovanje komponente Objects

Delovanje komponente Objects sem preveril s simulacijo v programskem okolju Vivado. Vzel sem testno strukturo test_krog.vhd, ki smo jo uporabili pri testiranju komponente krog pri eni izmed laboratorijskih vaj in jo predelal tako, da sem preveril delovanje svoje komponente Objects. Popraviti sem moral nastavitve določenih parametrov in sicer dimenzije okvirja sem nastavil kar na dimenzije VGA zaslona na katerem sem kasneje izrisoval sliko. Koordinati x in y sta obe potekali od -256 do 255 torej je bila ločljivost slike 512 krat 512 točk. Center žoge in centra igralnih ploščkov sem za potrebe simulacije nekoliko izmaknil izven središčne lege, da sem lahko preveril pravilnost delovanja komponente tudi pri branju vrednosti centralnih pozicij gibajočih objektov iz registrov. Perioda ure je bila nastavljena na 10 ns vhodni signal rgbin pa je imel vrednost "000000". Simulacija ustvari tudi tekstovno datoteko slika.txt, ki prikazuje dejansko sliko velikosti 512 krat 512 točk z vsemi izrisanimi objekti. Datoteka je ustvarjena tako, da na mesto, kjer je izhod rgb enak "111111" zapiše * sicer postavi presledek. Vsebina tekstovne datoteke je prikazana na sliki Slika 4. Pravilnost delovanja komponente sem nato še natančneje preveril z ogledom simulacije na kritičnih mestih. Izsek simulacije je prikazan na sliki Slika 5. Zgornji del slike prikazuje del, ko se zaključi zadnja vrstica zgornjega roba in je potem izhod rgb še vedno postavljen na "111111" prve tri točke v novi vrstici, saj so te točke del levega roba. Spodnji del slike pa prikazuje kako se izhod rgb postavi na "111111", ko pride do točk od -5 do 4, ki predstavljajo sredinsko črto.



Slika 4: Vsebina tekstovne datoteke slika.txt



Slika 5: Izsek simulacije komponente Objects

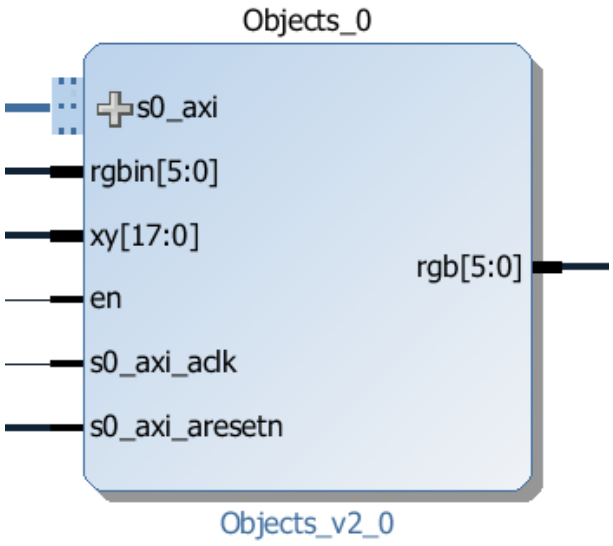
Zasedenost vezja si lahko ogledamo pot Utilization Report, ki se ustvari po tem, ko izvedemo sintezo vezja. Tabela zasedenosti je prikazana na sliki Slika 6.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	274	0	53200	0.52
LUT as Logic	274	0	53200	0.52
LUT as Memory	0	0	17400	0.00
Slice Registers	145	0	106400	0.14
Register as Flip Flop	145	0	106400	0.14
Register as Latch	0	0	106400	0.00
F7 Muxes	0	0	26600	0.00
F8 Muxes	0	0	13300	0.00

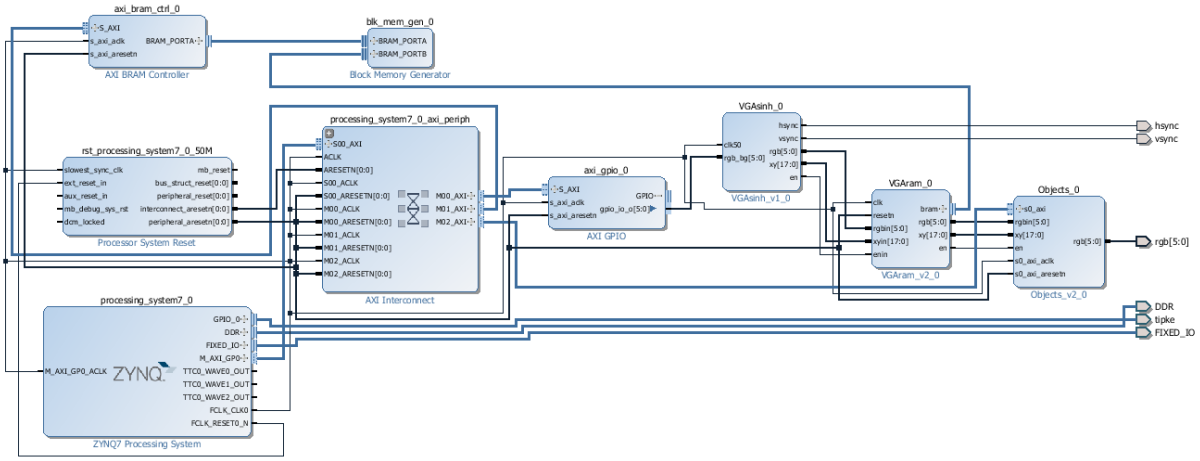
Slika 6: Tabela zasedenosti vezja po sintezi

Zasnova celotnega sistema

Ko sem izvedel simulacijo in se prepričal, da VHDL koda komponente Objects deluje tako, kot sem si zamislil sem vezje zapakiral v IP komponento. Nato sem ustvaril nov projekt za razvojno ploščo ZedBoard in nov blokovni diagram ter nanj postavil vse potrebne komponente in jih med sabo povezal. Izgled komponente Objects v blokovnem diagramu je prikazan na sliki Slika 7, shema celotnega sistema pa je predstavljena na sliki Slika 8.



Slika 7: IP komponenta Objects



Slika 8: Blokovni diagram celotnega sistema

Komponente, ki sem jih uporabil pri izdelavi celotnega sistema so procesorski sistem Zynq ter komponenta za sistemsko ponastavitev vezja, komponenta AXI GPIO in AXI Interconnect, ki skrbi za povezavo med master in slave napravami, komponenta VGAsinh, ki smo jo izdelali na eni izmed laboratorijskih vaj pri tem predmetu in deluje kot VGA krmilnik. Sistem vsebuje še blokovni pomnilnik Block Memory Generator, komponento za nadzor pomnilnika AXI BRAM Controller in komponento za branje pomnilnika VGAram. Te tri komponente omogočajo prikaz slike iz notranjega pomnilnika BRAM. Kot zadnjo sem vključil še svojo komponento Objects.

Procesorski sistem Zynq narekuje uro s frekvenco 50 MHz ostalim komponentam. Komponenta AXI GPIO ima preko vodila AXI dostop do registra, kjer je shranjena barva ozadja, to vrednost pa se potem posreduje na vhod rgb_bg komponente VGAsinh.

Izhodni signali komponente VGAsinh rgb, xy in en so najprej vezani na vhod komponente VGAram. Ta na izhod rgb zapiše sliko iz pomnilnika, signala xy in en pa posreduje naprej. Izhodni signali komponente VGAram rgb, xy in en so naprej vezani na vhode komponente Objects, ki čez sliko, ki je bila zapisana v pomnilniku prepiše dele, kjer se nahajajo grafični objekti igre, izhod rgb komponente Objects pa se nato direktno izrisuje na VGA zaslonu. Sistem lahko še bere stanja štirih tipk na plošči ZedBoard. Sam sem potreboval dve tipki za premikanje levega igralnega ploščka.

Zasedenost vezja po sintezi celotnega sistema si lahko ponovno ogledamo pod Utilization Report. Tabela zasedenosti je prikazana na sliki Slika 9.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	2487	0	53200	4.67
LUT as Logic	2333	0	53200	4.39
LUT as Memory	154	0	17400	0.89
LUT as Distributed RAM	0	0		
LUT as Shift Register	154	0		
Slice Registers	2817	0	106400	2.65
Register as Flip Flop	2817	0	106400	2.65
Register as Latch	0	0	106400	0.00
F7 Muxes	1	0	26600	<0.01
F8 Muxes	0	0	13300	0.00

Slika 9: Zasedenost vezja po sintezi celotnega sistema

Delovanje na razvojnem sistemu ZedBoard

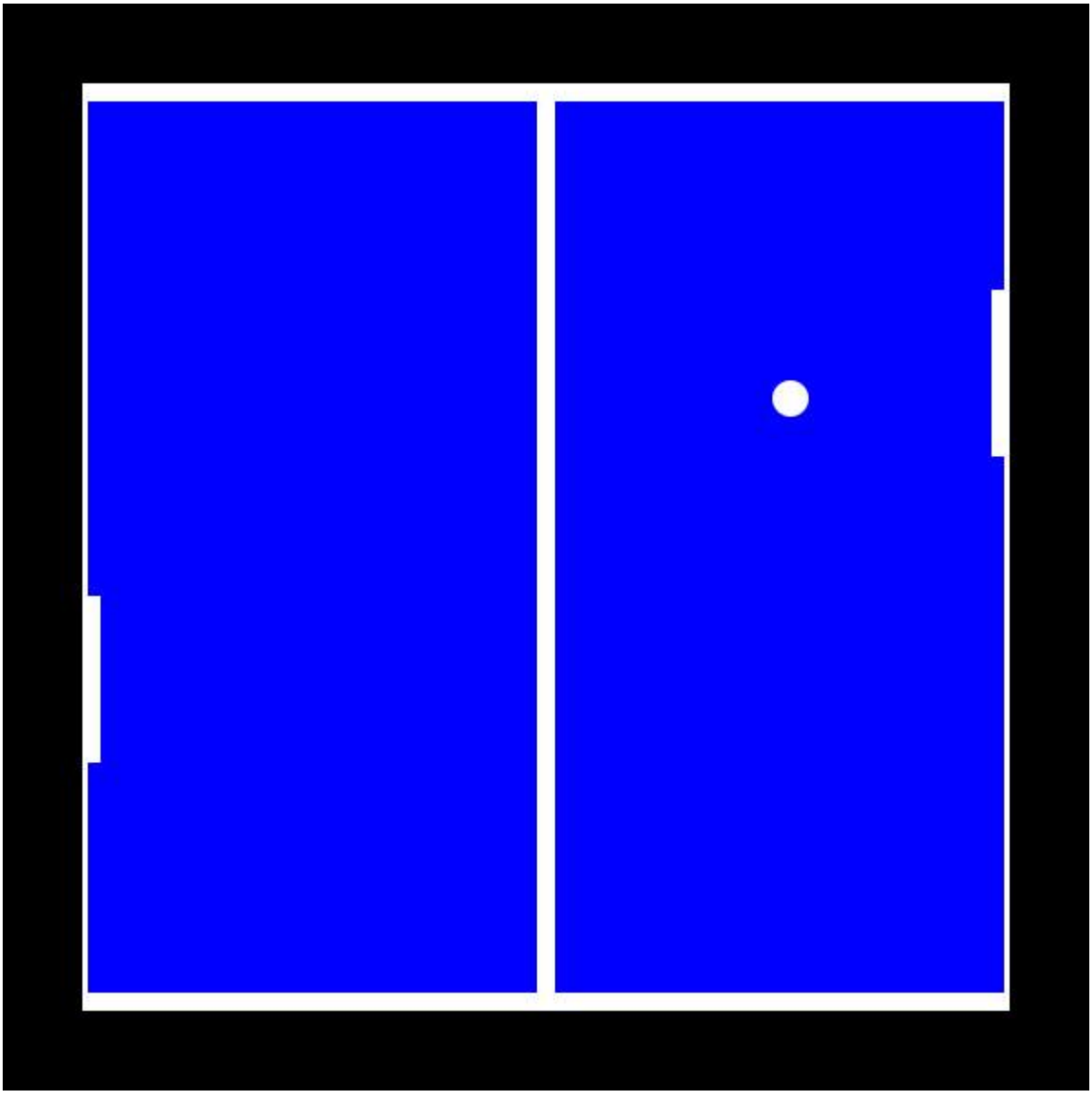
Ko sem preveril blokovni diagram in izvedel implementacijo sem se nato lotil še pisanja programa za računanje pozicij gibajočih objektov in nastavitev drugih parametrov igre.

V program sem vključil knjižnice xgpiops.h, kjer se nahaja gonilnik za PS GPIO (tipke), xil_io.h, ki omogoča neposreden dostop do registrov, sleep.h, ki omogoča nastavljanje poljubne časovne zakasnitve in time.h ter stdlib.h za uporabo funkcije rand(), ki generira neko naključno število. Sledila je deklaracija globalnih spremenljivk ter nato glavna funkcija main().

V glavni funkciji je bilo najprej potrebno deklarirati vse potrebne lokalne spremenljivke in inicializirati PS GPIO. Sledijo nastavitve igre kot so barva ozadja, ki je nastavljena na črno in je nato zapisana v pravi register, barva igralnega polja, ki se zapiše v pomnilnik za vsak piksel posebej in je nastavljena na modro, nastavitve začetnih pozicij obeh igralnih ploščkov in žoge, nastavitev začetne smeri ter kota potovanja žoge in nastavitev zakasnitve glavne zanke s čimer lahko nadziramo hitrost igre. Seveda bi lahko kateregakoli izmed teh parametrov poljubno nastavili v okvirju smiselnih vrednosti. Nastavili bi lahko na primer drugačno barvo ozadja ali igralnega polja ali pa spremenili hitrost poteka igre.

Nato se v glavni zanki preveri, če je bila pritisnjena katerakoli tipka, centralna pozicija levega igralnega ploščka pa se ustrezno poveča oziroma zmanjša če je bila pritisnjena zgornja ali spodnja tipka. Premik desnega igralca se računa glede na trenutno pozicijo žoge, njegova odzivnost pa je nekoliko zmanjšana s pomočjo funkcije rand(). Na ta način dosežemo, da je nasprotnika (računalnik) dejansko možno premagati. Centralni poziciji obeh igralcev nato združimo in zapišemo v pravi register. Sledi računanje premika žoge oziroma centralne pozicije žoge, x in y koordinati se računata posebej na koncu pa ju združimo in zapišemo v register preko vodila AXI. Na koncu sledi še zakasnitev zanke, ki poskrbi, da se igra ne izvaja prehitro.

Končen izgled igre je prikazan na sliki Slika 10.



Slika 10: Končen izgled igre Pong

Priloga 1 – VHDL koda

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Objects is --********************* ENTITY ***********************************
   Port (
         rgbin         : in std_logic_vector(5 downto 0);
         xy            : in std_logic_vector(17 downto 0);
         en            : in std_logic;
         rgb           : out std_logic_vector(5 downto 0);
         s0_axi_aclk    : in std_logic;
         s0_axi_aresetn : in std_logic;
         s0_axi_awaddr  : in std_logic_vector(3 downto 0);
         s0_axi_awprot  : in std_logic_vector(2 downto 0);
         s0_axi_awvalid : in std_logic;
         s0_axi_awready : out std_logic;
         s0_axi_wdata   : in std_logic_vector(31 downto 0);
         s0_axi_wstrb   : in std_logic_vector(3 downto 0);
         s0_axi_wvalid  : in std_logic;
         s0_axi_wready  : out std_logic;
         s0_axi_bresp   : out std_logic_vector(1 downto 0);
         s0_axi_bvalid  : out std_logic;
         s0_axi_bready  : in std_logic;
         s0_axi_araddr  : in std_logic_vector(3 downto 0);
         s0_axi_arprot  : in std_logic_vector(2 downto 0);
         s0_axi_arvalid : in std_logic;
         s0_axi_arready : out std_logic;
         s0_axi_rdata   : out std_logic_vector(31 downto 0);
         s0_axi_rresp   : out std_logic_vector(1 downto 0);
         s0_axi_rvalid  : out std_logic;
         s0_axi_rready  : in std_logic
       );
end Objects; --********************** END ENTITY *********************************

architecture Behavioral of Objects is --********* ARCHITECTURE **********************

   --definiramo AXI paralelni vmesnik kot komponento
   component parvmes_v1_0_S0_AXI is
      generic (
               C_S_AXI_DATA_WIDTH    : integer    := 32;
               C_S_AXI_ADDR_WIDTH    : integer    := 4
            );
```

```
    port (
          p0_dataout, p1_dataout : out std_logic_vector(31 downto 0);
          p2_datain, p3_datain : in std_logic_vector(31 downto 0);
          S_AXI_ACLK     : in std_logic;
          S_AXI_ARESETN  : in std_logic;
          S_AXI_AWADDR   : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
          S_AXI_AWPROT   : in std_logic_vector(2 downto 0);
          S_AXI_AWVALID  : in std_logic;
          S_AXI_AWREADY  : out std_logic;
          S_AXI_WDATA    : in std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
          S_AXI_WSTRB    : in std_logic_vector((C_S_AXI_DATA_WIDTH/8)-1 downto 0);
          S_AXI_WVALID   : in std_logic;
          S_AXI_WREADY   : out std_logic;
          S_AXI_BRESP    : out std_logic_vector(1 downto 0);
          S_AXI_BVALID   : out std_logic;
          S_AXI_BREADY   : in std_logic;
          S_AXI_ARADDR   : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
          S_AXI_ARPROT   : in std_logic_vector(2 downto 0);
          S_AXI_ARVALID  : in std_logic;
          S_AXI_ARREADY  : out std_logic;
          S_AXI_RDATA    : out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
          S_AXI_RRESP    : out std_logic_vector(1 downto 0);
          S_AXI_RVALID   : out std_logic;
          S_AXI_RREADY   : in std_logic
        );
  end component parvmes_v1_0_S0_AXI;

  ------------------------------- definiramo signale ---------------------------
  signal dummy_a, dummy_b: std_logic_vector (31 downto 0);

  signal x, y: std_logic_vector (8 downto 0);
  signal x_signed, y_signed: signed (8 downto 0);

  --vektor s centralnima y pozicijama pravokotnikov
  signal yc1yc2: std_logic_vector (31 downto 0);
  signal yc1, yc2: std_logic_vector (8 downto 0);
  signal yc1_signed, yc2_signed: signed (8 downto 0);

  --vektor s centralno pozicijo kroga
  signal xcyc: std_logic_vector (31 downto 0);
  signal xc, yc: std_logic_vector (8 downto 0);
  signal xc_signed, yc_signed: signed (8 downto 0);
  signal x2, y2: signed (17 downto 0);

  --polmer kroga
  constant r: integer :=10;
  constant r2: integer :=r*r;

begin --****************************************** BEGIN *************************

  --povežemo priključke AXI paralelnega vmesnika z priključki komponente Objects
parvmes: parvmes_v1_0_S0_AXI
```

```
    generic map (
                C_S_AXI_DATA_WIDTH    => 32,
                C_S_AXI_ADDR_WIDTH    => 4
            )
    port map (
            p2_datain => dummy_a,
            p3_datain => dummy_b,
            p0_dataout => yc1yc2,
            p1_dataout => xcyc,
            S_AXI_ACLK => s0_axi_aclk,
            S_AXI_ARESETN => s0_axi_aresetn,
            S_AXI_AWADDR =>  s0_axi_awaddr,
            S_AXI_AWPROT =>  s0_axi_awprot,
            S_AXI_AWVALID => s0_axi_awvalid,
            S_AXI_AWREADY => s0_axi_awready,
            S_AXI_WDATA =>   s0_axi_wdata,
            S_AXI_WSTRB =>   s0_axi_wstrb,
            S_AXI_WVALID =>  s0_axi_wvalid,
            S_AXI_WREADY =>  s0_axi_wready,
            S_AXI_BRESP =>   s0_axi_bresp,
            S_AXI_BVALID =>  s0_axi_bvalid,
            S_AXI_BREADY =>  s0_axi_bready,
            S_AXI_ARADDR =>  s0_axi_araddr,
            S_AXI_ARPROT =>  s0_axi_arprot,
            S_AXI_ARVALID => s0_axi_arvalid,
            S_AXI_ARREADY => s0_axi_arready,
            S_AXI_RDATA =>   s0_axi_rdata,
            S_AXI_RRESP =>   s0_axi_rresp,
            S_AXI_RVALID =>  s0_axi_rvalid,
            S_AXI_RREADY =>  s0_axi_rready
          );

    --izkuščimo x in y koordinati iz vektorja xy
    x<=xy(17 downto 9);
    y<=xy(8 downto 0);
    x_signed<=signed(x);
    y_signed<=signed(y);

    --izluščimo yc1 in yc2 koordinati iz vektorja yc1yc2
    yc1<=yc1yc2(17 downto 9);
    yc2<=yc1yc2(8 downto 0);
    yc1_signed<=signed(yc1);
    yc2_signed<=signed(yc2);

    --izluščimo xc in yc koordinati iz vektorja xcyc
    xc<=xcyc(17 downto 9);
    yc<=xcyc(8 downto 0);
    xc_signed<=signed(xc);
    yc_signed<=signed(yc);

p1: process(s0_axi_aclk) --**************** PROCESS ********************************
    begin
```

```
    if rising_edge(s0_axi_aclk) then
      --izračunamo kvadratno vrednost koordinat
      x2 <= (x_signed-xc_signed) * (x_signed-xc_signed);
      y2 <= (y_signed-yc_signed) * (y_signed-yc_signed);

      if  (
            (
              (
                --pogoj za sredinsko črto
                (x_signed>=-6 and x_signed<=3)
              ) or
              (
                --pogoj za levo in desno črto
                (x_signed>=255 or x_signed<=-255) or
                (x_signed>=252 and x_signed<=254)
              ) or
              (
                --pogoj za zgornjo in spodnjo črto
                (y_signed>=-256 and y_signed<=-247) or
                (y_signed>=246 and y_signed<=255)
              ) or
              (
                --pogoj za levi pravokotnik
                (x_signed>=255 or x_signed<=-248) and
                (y_signed>=(-46+yc1_signed) and y_signed<=(45+yc1_signed))
              ) or
              (
                --pogoj za desni pravokotnik
                (x_signed>=245 and x_signed<=254) and
                (y_signed>=(-46+yc2_signed) and y_signed<=(45+yc2_signed))
              ) or
              (
                --pogoj za krog
                (x2 + y2 <= r2)
              )
            ) and en='1'
          ) then
          rgb <= "111111";
      else
          rgb <= rgbin;
      end if;
    end if;
  end process;--********************** END PROCESS ******************************

end Behavioral; --************************** END ARCHITECTURE **********************
```

Priloga 2 – C koda

```
#include <stdio.h>
#include "platform.h"
#include "xgpiops.h" // gonilnik za PS GPIO (tipke)
#include "xil_io.h" // neposreden dostop do registrov
#include "sleep.h" // časovna zakasnitev
#include <time.h>
#include <stdlib.h>

//********************************** GLOBALNE SPREMENLJIVKE ************************
//pozicije igralcev
int yc1;
int yc2;

//pozicija žoge
int xc;
int yc;

//vektorja pozicij
int yc1yc2;
int xcyc;

//vertikalna in horizontalna smer žoge
int ballVertical; //0=dol, 1=gor
int ballHorizontal; //0=desno, 1=levo

//časovna zakasnitev zanke
int delay;

//kot odboja
int x;
int y;

//spremenljivke za nastavljanje kota odboja
int i=0;
int j=0;

//naključna številka
int r=0;

//********************************** FUNKCIJE **************************************
int intMerge(int x, int y) //prepare coordinates for component Objects
{
	int result;
	if(y>=0)
	{
		x=x<<9;
		result=x+y;
	}
	else
	{
```

```
		x=x<<9;
		result=x+(512+y);
	}
	return result;
}

/*void speed(int tipke)
{
	static int i=0;
	if(tipke==0x1) //pritisnjena desna tipka
	{
		if(i==0)
		{
			delay=delay+1000; //zmanjšaj hitrost
			i=1;
		}
	}
	else if(tipke==0x4) //pritisnjena leva tipka
	{
		if(i==0)
		{
			delay=delay-1000; //povečaj hitrost
			i=1;
		}
	}
	else i=0;
	if(delay>10000) delay=10000; //zgornja meja
	if(delay<1000) delay=1000; //spodnja meja
}*/

int main() //************************** MAIN ****************************************
{
	int t;
	int i;
	char *p; //kazalec na pomnilnik za prikaz grafike
	XGpioPs gp;
	XGpioPs_Config *gpCfg;

	//---------------------------------- inicializacija PS GPIO ---------------------------------
	gpCfg = XGpioPs_LookupConfig(XPAR_PS7_GPIO_0_DEVICE_ID);
	XGpioPs_CfgInitialize(&gp, gpCfg, gpCfg->BaseAddr);

	//---------------------------- nastavitve igre ------------------------------------------------
	//nastavitev barve ozadja (RGB 6 bit)
	Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR, 0x00);

	//nastavitev barve polja
	for(i=0;i<512*512;i++)
	{
		p= (char *)XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + i;
		*p = 0x03; //RGB 6 bit
	}
```

```
//nastavitev začetne pozicije igralcev in žoge
yc1=0;
yc2=0;
xc=50;
yc=110;

//nastavitev začetne smeri žoge
ballVertical=0;
ballHorizontal=1;

//nastavitev začetne hitrosti
delay=1500;

//nastavitev začetnega kota žoge
x=1;
y=5;

//------------------------------- glavna zanka ----------------------------------------------
while(1)
{
	//branje tipk
	t = XGpioPs_Read(&gp, 2);

	//generiranje naključne številke
	srand(yc);
	r = rand()%2;

	//---------------------------------- premik igralcev -------------------------------------
	//premik prvega igralcev
	if(t==0x2) //pritisnjena tipka gor
	{
		yc1--;
	}
	else if(t==0x8) //pritisnjena tipka dol
	{
		yc1++;
	}
	else yc1=yc1;
	if(yc1>200) yc1=200; //zgornja meja
	if(yc1<-200) yc1=-200; //spodnja meja

	//premik drugega igralca
	if(r!=0)
	{
		if((yc2-yc)>0) yc2--;
		if((yc2-yc)<0) yc2++;
	}
	if(yc2>200) yc2=200; //zgornja meja
	if(yc2<-200) yc2=-200; //spodnja meja

	//združitev premikov obeh igralcev
```

```
yc1yc2=intMerge(yc1, yc2);
Xil_Out32(XPAR_OBJECTS_0_BASEADDR, yc1yc2);

//----------------------- premik žoge ----------------------------------------------------
i++;
j++;

if(i>=x)
{
	i=0;
	if(ballHorizontal==0) xc++;
	if(ballHorizontal==1) xc--;
}

if(j>=y)
{
	j=0;
	if(ballVertical==0) yc++;
	if(ballVertical==1) yc--;
}

//robni pogoji za odboj žoge
if(xc<=-237)
{
	if(((yc1-yc)<=55) && ((yc1-yc)>=-55))
	{
		ballHorizontal=0;
		//določitev kota odboja žoge glede na vpadno pozicijo na ploščo
		if((yc1-yc)>0)
		{
			y=(56-(yc1-yc))/10+1;
			ballVertical=1;
		}
		else
		{
			y=(56+(yc1-yc))/10+1;
			ballVertical=0;
		}
	}
	else
	{
		xc=0;
		yc=0;
	}
}
if(xc>=236)
{
	if(((yc2-yc)<=55) && ((yc2-yc)>=-55))
	{
		ballHorizontal=1;
		//določitev kota odboja žoge glede na vpadno pozicijo na ploščo
		if((yc2-yc)>0)
```

```
                                {
                                        y=(56-(yc2-yc))/10+1;
                                        ballVertical=1;
                                }
                                else
                                {
                                        y=(56+(yc2-yc))/10+1;
                                        ballVertical=0;
                                }
                        }
                        else
                        {
                                xc=0;
                                yc=0;
                                ballHorizontal=1;
                        }
                }
                if(yc<=-237) ballVertical=0;
                if(yc>=236) ballVertical=1;

                //vpis koordinat žoge v register
                xcyc=intMerge(xc, yc);
                Xil_Out32(XPAR_OBJECTS_0_BASEADDR+4, xcyc); //xcyc

                //nastavitev zakasnitve
                //speed(t);
                usleep(delay);
        }
        return 0;
} //********************************** END MAIN ***********************************
```