Porčilo projekta pri predmetu

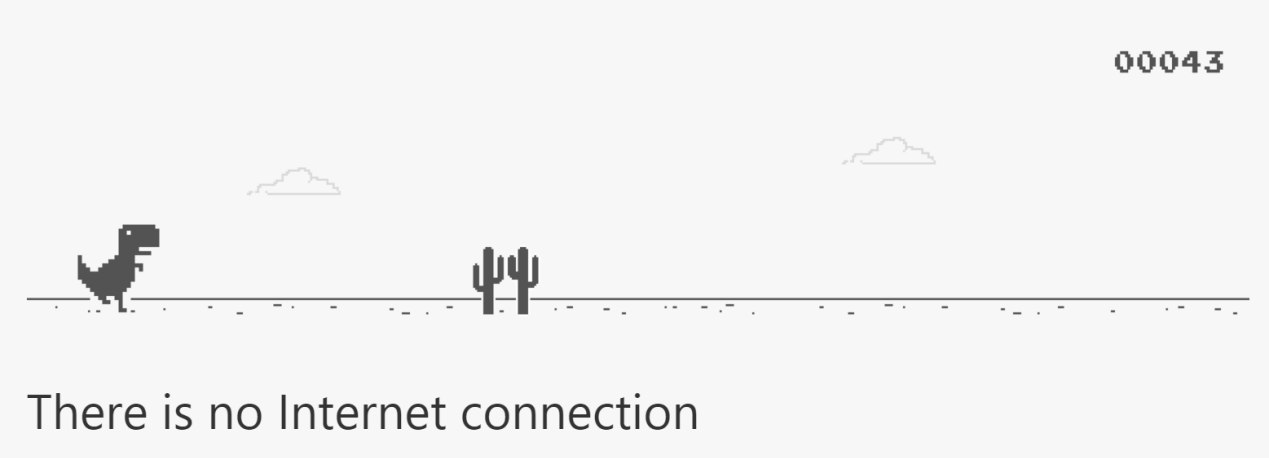
# Digitalna elektronska vezja in sistemi

Tadej Skuber

14. 11. 2016, Ljubljana

# Cilj

Namen te projektne naloge je bil narediti arkadno igro podobno Googlovi Chrome Dino game. V igri tekajoči igralec s pritiskom na presledek preskakuje ovire ter si povečuje število točk. Ko oviro zadane se igra začne od začetka.




Primer igranja igre.

Do igre lahko pridemo, če izklopimo internet ter v Chromu poskušamo odpreti kakšno spletno stran. Ali pa alternativno če v Google vpišemo “google dino game” ter kliknemo prvi zadetek.

# Osnovni princip

Projekt je razdeljen na VHDL ter C del. V VHDL delu projekta je poskrbljeno za izris na zaslon, v C delu pa za logiko igre. Za nadzor skakanja se uporablja ena od tipk na razvojni plošči.

Osnovni koncet izrisa na zaslon je podoben kot pri 4. laboratorijski vaji. V naši VHDL komponenti imamo pomnilnik v katerega preko AXI vodila vpisujemo podatke o lokaciji posamezne sličice na zaslonu.

Pomnilnik je zaradi preprostosti igre (samo 3 ali 4 različne slikice) razdeljen na manjše. In sicer lokacija v pomnilniku od 0 do 99 je namenjena tlom, do 100 do 199 oblakom, … Po vodilu tako samo pošljemo kje na zaslonu se mora pojaviti nek objekt (v C-ju v določen register zapišemo neko vrednost), VHDL del pa ga tam izriše. Če pa nekaterih objektov ne želimo na zaslonu pa damo koordinate izven zaslona.
Izris slikic na zaslon poteka tako, da za tekoči piksel preverimo, ali se nahaja znotraj katere slikice. Če se, preverimo vrednost ustreznega bit te slike ter rgb-ju priredimo drugačno bravo.

# VHDL

VHDL je zgrajen na osnovi predloge VGARAMtest (4. laboratorjiska vaja) ter vsebuje dodano IP komponento. IP komponenta je zgrajena is treh datotek, in sicer iz vga_bram_draw (dodana komponenta) ter dveh malce prilagojenih (AXI_VGA_v1_0_S00_AXI ter AXI_VGA_v1_0 oz. vmesnik ter komponenta, ki vse skupaj povezuje).

## AXI_VGA_v1_0 ter AXI_VGA_v1_0_S00_AXI

Ti dve komponenti nam naredi že Vivado (če sledimo navodilom 4. vaje). Potrebno ju je le malce spremeniti.

AXI_VGA_v1_0_S00_AXI:
In sicer dodati je potrebno kontrolni signal **wren** (to naredimo kot je opisano v 4. laboratorijski vaji). Nato dodamo še signale **reg** (ta signal nam služi kot naslov pomnilnika slike, ki ji hočemo spremeniti koordinate), **xy** (x in y koordinate slike na naslovu, ki ga posreduje reg) ter **num** (katera verzija slike naj se izriše na zaslon; v primeru če hočemo doseči učinek gibanja).

```
reg: out std_logic_vector(8 downto 0);
xy: out std_logic_vector(19 downto 0);
num: out std_logic_vector(2 downto 0);
wren: out std_logic;
```

Signal xy je 8-biten, saj potrebujemo za vsako koordinato (x in y) po 10 bitov. Num je 3 biten, da lahko prikažemo zadostno število ustreznih slik posameznega objekta. Tako nam za reg ostane zadostnih 9 bitov za naslove (skupaj 32 bitov). Tako potrebujemo za premikanje slik samo en naslov z odmikom (po naključju slv_reg1). Lako bi uporabili tudi bolj zahteven način, vendar je ta čisto dovolj za to preprosto igro.

```
--priključki signalov ki gredo naprej
--na vga_bram_draw
reg <= slv_reg1(31 downto 23);
xy <= slv_reg1(22 downto 3);
num <= slv_reg1(2 downto 0);
```

Zgornjih 9 bitov je tako namenjenih naslovu, srednjih 10 za x in y ter spodnji trije za num .

Primer postavitev sličice igralca na x koordinato 50, y koordinato 50 ter prikaz prve slikice igralca:

Xil_Out32(0x43c00004, (500 << 23) | (50 << 13) | (50 << 3) | 1);

AXI_VGA_v1_0_S00_AXI:
V to komponento vključimo vga_bram_draw ter kot v 4. vaji rutinsko skupaj povežemo potrebne signale.

## vga_bram_draw

Ta komponenta skrbi za izris slik na zaslon. Na začetku imamo tako deklarirane dvodimenzionalne zbirke tipa basic_block. V njih shranjujemo slikice za različne objekte.

```
type basic_block is array(0 to 19) of std_logic_vector(0 to 19);
--1. slika zmajčka (igralca)
signal player_block_0: basic_block := (
"00000000000111111000",
"00000000000001111110",
"00000000001111101111",
"00000000000001111111",
"00000000000000110000",
"00001001001001111000",
"00111111011111111111",
"01110111111111111111",
"11100011111111111111",
"00000111111111111110",
"00001111111111111000",
"00011111111111111100",
"00111111111111101110",
"00011111111111100100",
"00011111100111000000",
"00011111000011100000",
"00011111000001110000",
"00001111100000111000",
"00001111111000011000",
"00000111111100011110");
```

Del te komponente je pomnilnik, ki skrbi za shranjevanje x in y koordinate posamezne slike.

```
mem_bram: process(clk50)
begin
    if rising_edge(clk50) then
        if wren = '1' then
            bram(to_integer(unsigned(axireg_reg))) <= axireg_xy & axireg_num;
        end if;
    end if;
end process;
```

Podobno kot v 4. vaji.

Del pa skrbi za preverjanje, ali se trenutni piksel nahaja znotraj koorinat željene slike. Če se, pripišemo ustrezen bit slike enemu od signalov za shranjevanje.

```
--imamo neko število oblakov, v tem primeru 12
--(lahko bi jih dali 100, vendar bi jih morali 88 na začetku postaviti izven zaslona)
for i in 0 to 11 loop
    --prevermo če se trenutni piksel nahaja znotraj katerega od oblakov
    if (((tx >= unsigned(bram(i + 100)(22 downto 13))) and (tx <= 19 + unsigned(bram(i + 100)(22 downto 13)))) and
    ((ty >= unsigned(bram(i + 100)(12 downto 3))) and (ty <= 19 + unsigned(bram(i + 100)(12 downto 3))))) then
        --če se mu pripišemo ustrezen bit slike
        --oblaki se za?nejo na naslovu 100
        cloud_block_pixel(i) <= cloud_block(to_integer(ty-unsigned(bram(i+100)(12 downto 3))))(to_integer(tx-unsigned(bram(i+100)(22 downto 13))));
    else
        cloud_block_pixel(i) <= '0';
    end if;
end loop;
```

Kratka razlaga je v komentarijh.

```
--pogledamo če je kateri od signalov za shranjevanje '1', ter če je to zapišemo
ground_block_pix <= '0' when ground_block_pixel = (x"0000000000" & "00") else '1';
cloud_block_pix <= '0' when cloud_block_pixel = "000000000000" else '1';
obstacle_block_pix <= '0' when obstacle_block_pixel = "0000" else '1';

--pogledamo če se karkoli nahaja na trenutnem pikslu
pixel <= ground_block_pix or cloud_block_pix or obstacle_block_pix or player_block_pixel;

--barva piksla je bela če je ozadje ter temno modra če je piksel obarvan
rgb <= "00000000" when en='0' else
       "01001010" when pixel='1' else
       "11111111" when pixel='0';
```

Nato preverimo bite za shranjevanje ter temu ustrezno spremenimo bravo.

## Simulacija vga_bram_draw

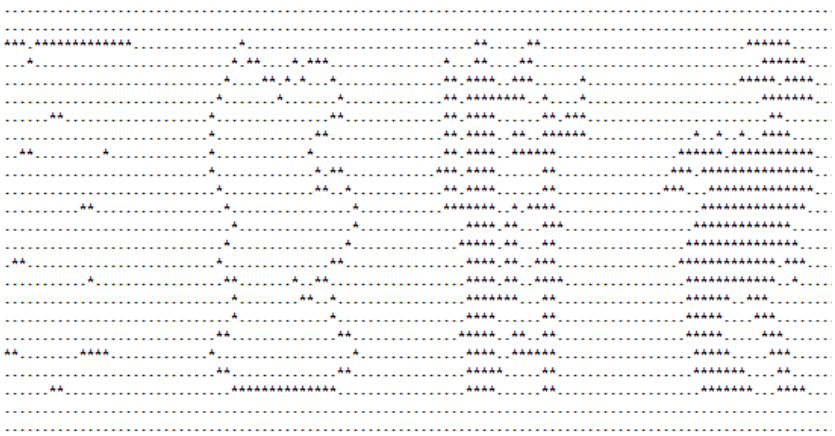
Preden komponento vga_bram_draw kjerkoli vključimo jo moramo še pretestirati. Pripravimo simulacijo.

```
--nastavimo objekte na neke lokacije na zaslonu
--tla
axireg_reg <= "000000000";
axireg_xy  <= "0000000000"&"0111110100";
axireg_num <= "000";
wait until rising_edge(clk50);
--oblak
axireg_reg <= "001100100";
axireg_xy  <= "0000011110"&"0111110100";
axireg_num <= "000";
wait until rising_edge(clk50);
--ovira
axireg_reg <= "011001000";
axireg_xy  <= "0000111100"&"0111110100";
axireg_num <= "000";
wait until rising_edge(clk50);
--igralec
axireg_reg <= "111110100";
axireg_xy  <= "0001011010"&"0111110100";
axireg_num <= "000";
wait until rising_edge(clk50);
wren <= '0';

--posimuliramo celoten zaslon
--ter zapišemo v txt datoteko
for y in 0 to 601 loop
    for x in 0 to 1039 loop
        wait until rising_edge(clk50);
        if rgb=x"4A" then
            write(vrstica, '*');
        elsif rgb=x"FF" then
            write(vrstica, '.');
        end if;
    end loop;
    writeline(dat, vrstica);
end loop;
```

Kratka razlaga v komentarjih.

Simulacija nam v txt datoteko zapiše spodnje rezultate.

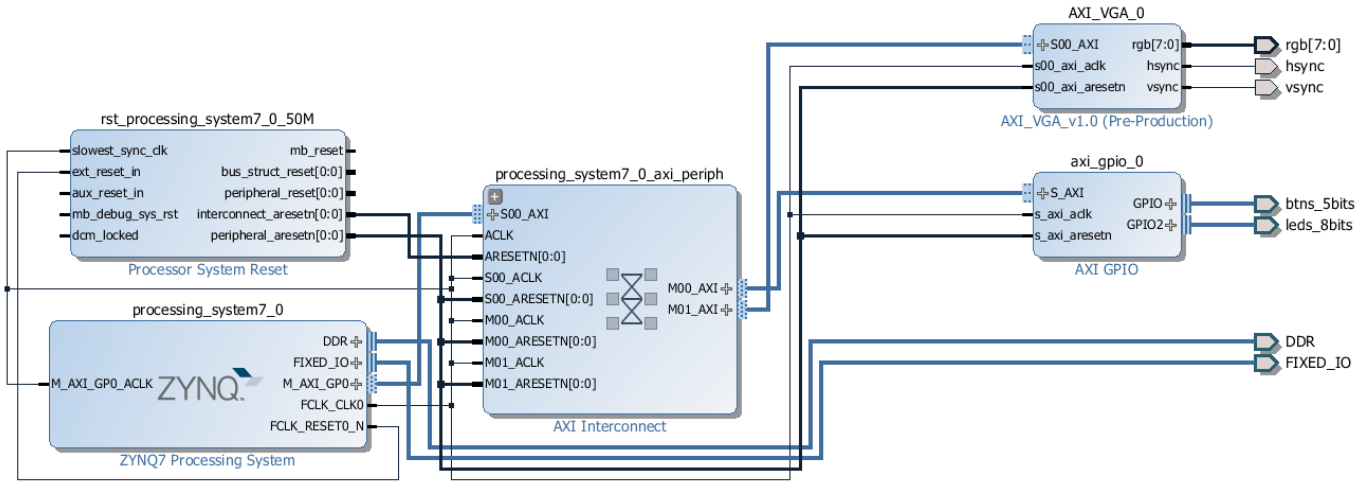


Del zaslona.

Ker se na ustrezna mesta izpišejo ustrezni znaki zaključimo, da komponenta deluje pravilno.

## Celoten sistem

Na koncu IP komponento izvozimo, ter jo vključimo v VGAtest. Postopamo kot v laboratorijskih vajah.




Celoten system izgleda nekako takole.

# C

V tem delu je poskrbljeno za logko igre. Program je sestavljen iz treh glavnih funkcij, ki se kličejo v while zanki. In sicer najprej z funkcijo make() poskrbimo, da se vse postavi na svoje začetno mesto. Nato vstopimo v while zanko ter pokličemo update(). Ta funkcija poskrbi za tipko ter posodobi pozicije ter lastnosti objektov (oblakov, tal, ovir, …). Nato na novih

lokacijah narišemo objekte (funkcija draw()). Sledi še funcija usleep(), ki poskrbi da igra ne teče prehitro.

Funkcija update z kratkimi komentarji:

```
void update(){
	//preverimo gumb
	button = XGpio_DiscreteRead(&gp, 1) & 0x0001;

	//premaknemo oblake
	int i = 0;
	for(i; i<12; i++){
		clouds[0][i] -= cloud_speed[i];
		//če je oblak prišel do konca ga damo na začetek
		//ter mu damo neko naključno hitrost
		if(clouds[0][i] < 0){
			clouds[0][i] = 801;
			cloud_speed[i] = rand()%5+1;
		}
	}

	//premakneko ovire
	//podobno kot pri oblakih
	i = 0;
	for(i; i<4; i++){
		obstacles[0][i] -= obstacle_speed[i];
		if(obstacles[0][i] <= 0){
			if(i == 0){
				obstacles[0][i] = 800;
				//če je ovira prišla do konca jo damo na začetek
				//z odmiki poskrbimo da so ovire različno narazen
			}else{
				obstacles[0][i] = obstacles[0][i-1] + 100 + rand()%100;
			}
		}
	}

	//igralca premikamo
	if(player[1] < 560){
		//če je igralec v zraku simuliramo skok
		player[1] += gravity - player_jump_speed;
		player_jump_speed -= 1;
		if(player_jump_speed < 0){
			player_jump_speed = 0;
		}
		if(player[1] > 560){
			player[1] = 560;
		}
	}else{
		if(button == 1){
			//če smo pritisnili na gumb in
			//smo na tleh skočimo
			player_jump_speed = 19;
			player[1] += gravity - player_jump_speed;
			player_jump_speed -= 1;
		}
	}
```

```
	//preverimo, če smo zadeli katero do ovir
	i = 0;
	for(i; i<4; i++){
		if(intersects_obstacles(i) == 1){
			//če smo jo igro začnemo znova
			make();
		}
	}
}
```

Izris s komentarji:

```
void draw(){
	//narišemo tla
	int i = 0;
	for(i; i<42; i++){
		Xil_Out32(AXIPIX, (i << 23) | (ground[0][i] << 13) | (ground[1][i] <<
	3) | 0);
	}
	//oblake
	i = 0;
	for(i; i<12; i++){
		Xil_Out32(AXIPIX, (i + 100 << 23) | (clouds[0][i] << 13) |
(clouds[1][i] << 3) | 0);
	}
	//ovire
	i = 0;
	for(i; i<4; i++){
		if(obstacles[0][i] <= 800){
			Xil_Out32(AXIPIX, (i + 200 << 23) | (obstacles[0][i] << 13) |
(obstacles[1][i] << 3) | 0);
		}else{
		Xil_Out32(AXIPIX, (i + 200 << 23) | (800 << 13) | (600 << 3) | 0);
		}
	}

	//ta del skrbi za spreminjanje slike igralca,
	//da izgleda kot tek
	frame -= 1;

	if(frame <= 0){
		frame = 5;
		if(player_frame == 0){
			player_frame = 1;
		}else{
			player_frame = 0;
		}
	}

	//narišemo igralca
	Xil_Out32(AXIPIX, (500 << 23) | (player[0] << 13) | (player[1] << 3) |
player_frame);
}
```

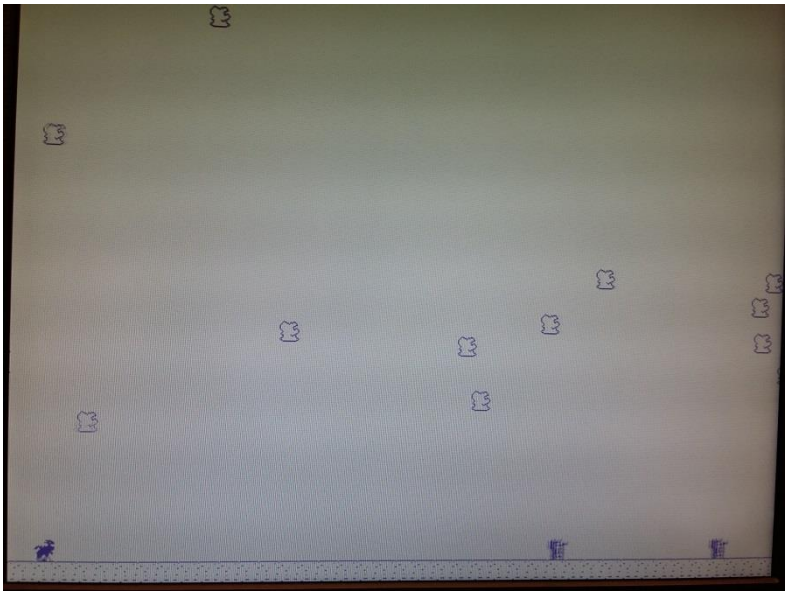
Main z zanko:

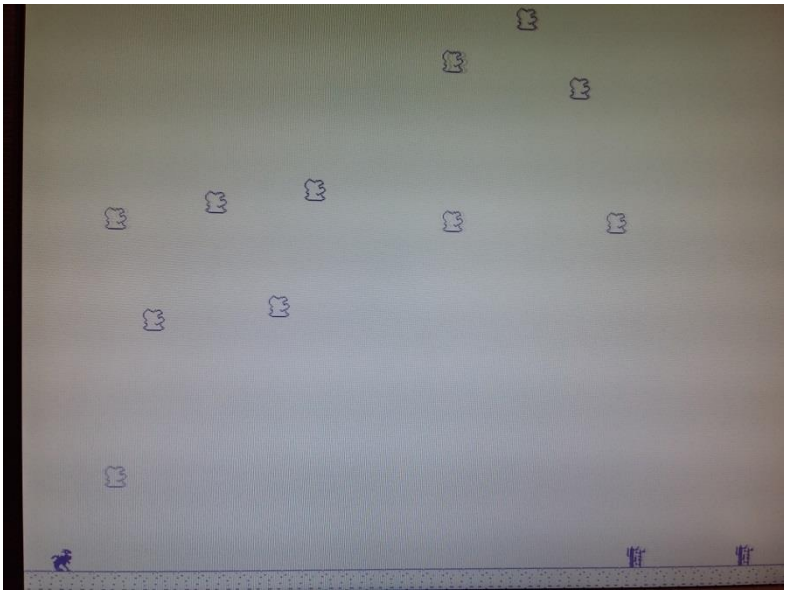
```
...
    make();

    while(1){
       update();
       draw();
       control();
    }
...
```

# REZULTATI

Imamo igrico, kjer se more igralec z skakanjem izogibati oviram. Moženo izboljšavo bi lahko dodali še kakšno drugo oviro (letečo ali pa luknjo v tleh). Možno bi bilo tudi izboljšati animacije. Manjka pa sevada še števec točk (za bolj zanimivo ter tekmovalno igranje).





Na koncu vse skupaj izgleda nekako takole. Slike slabše kvalitete, saj so slikane z telefonom.