Poročilo končnega izdelka

»Plahutajoči ptič«

Avtor: Lovro Benčina (FE_VSŠ_3_E) VŠt.: 64210022
Profesorja: prof. dr. Andrej Trost, prof. dr. Andrej Žemva

KAZALO POGLAVIJ

KAZALO SLIK

KAZALO TABEL

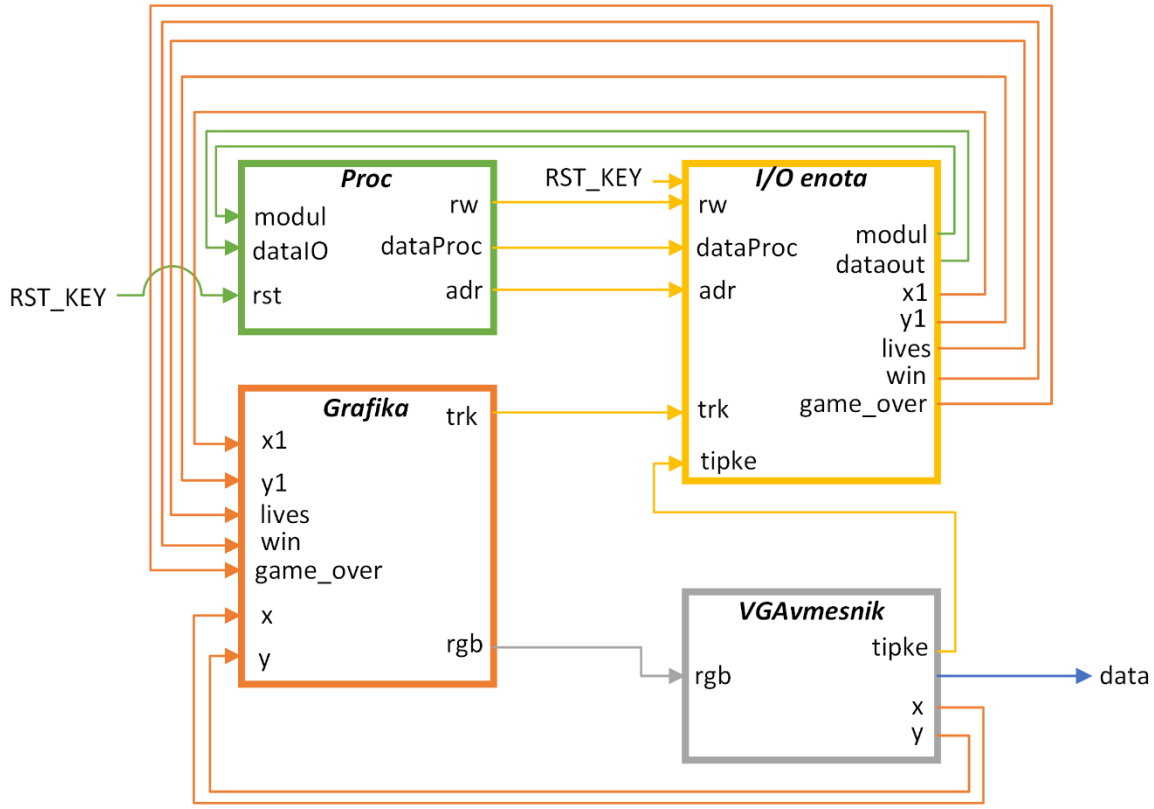
1. UVOD

Ob zaključku laboratorijskih vaj pri Modulu B smo si zadali, da izdelamo končni izdelek zgrajen na osnovi, ki smo jo pripravili tekom laboratorijskih vaj.
Ta med drugim vsebuje tudi že v preteklem poročilu omenjeno strukturo 12-bitnega učnega procesorja.
Jaz sem si za cilj izbral popularni mobilni igri podobno različico ter jo pripeljal od ideje do realizacije z dano podlago.

2. CILJI

Kot glavni cilj sem si zadal imitacijo pred časom popularne mobilne igre »Flappy bird«.
Iz omenjene igre sem želel imitirati elemente neprenehnega padanja glavnega objekta (ptiča), ovir, ki prestregajo pot napredovanja proti cilju, ki je na desni strani zaslona, izgubo življenja ob dotiku ovire ali tal ter dvig objekta (ptiča) ob pritisku na tipko.
Poleg posnemanih funkcij sem za realizacijo ciljal na nekaj svojih idej. Med drugim: možnost pomikanja objekta (ptiča) v smeri cilja s pritiskom na tipko, različne tipe ovir, animirane grafike ter koncept zmage z dotikom desnega roba zaslona.

3. BLOKOVNA SHEMA SISTEMA



Slika 1: Blokovna shema Sistema. Vir: Diagram narisan v orodju Microsoft Visio.

Sistem je zgrajen iz štirih blokov, in sicer: Procesorja, I/O bloka, Grafičnega bloka in VGAvmesnika. Glavni pa so trije: Grafični blok, Procesorski blok ter I/O blok. Procesor je enak tistemu, katerega sem obravnaval v predhodnem poročilu.

4. VGAVMESNIK

Ta blok v prvi vrsti skrbi za generacijo vseh potrebnih sinhronizacijskih signalov, ki omogočajo pravilen izris slike na zaslon preko vodila VGA.
Poleg tega skrbi za povezavo med sistemom ter razvojno ploščico. To vključuje branje tipk ter upravljanje LED diod.
Glavni izhodi tega bloka pa so signali tipke, ki opisujejo stanje tipk na ploščici, ter koordinati (x) in (y).
Bloku v vprašanju, pri izdelavi projekta, nisem naredil znatnih sprememb.

5. PROCESOR (“PROC”)

Kot že omenjeno je je procesor enak tistemu, ki sem ga obravnaval v vmesnem poročilu.
V kratkem procesor deluje v dveh korakih, in sicer v prvem zajame ukaz iz naše programske datoteke ter ga v drugem koraku izvede. Prevedeni koraki so hranjeni v datoteki »koda.mif« Assemblerska koda je bila prilagojena potrebam igre. Nekaj glavnih sprememb:

- Implementacija konstantnega padca objekta,
- Implementacija premika v desno,
- Implementacija dogodka »win« ob doseženi določeni x koordinati,
- Implementacija življenjske spremenljivke,
- Implementacija dogodka »game_over« ob prekoračitvi števila trkov oziroma ob porabi vseh »življenj«,
- Ponastavitev koordinat ter pomožnih spremenljivk ob trku.

5.1. Implementacija konstantnega padca

Naša koordinata y1 (koordinata našega objekta) se v vsakem ciklu poveča za eno enoto.

```
lda y
add 1
sta y
outp 2
```

Slika 2: Izsek kode, ki prikazuje implementacijo konstantnega padca. Vir: https://lniv.fe.uni-lj.si/cpe/cpu.html ; 16.2.2024

5.2. Implementacija pomika v desno

Ob pritisku tipko S1 se koordinata x1 (koordinata našega objekta) poveča za dve enoti in tako v rezultatu s konstantnim padcem premakne objekt navzgor za eno enoto.

```
lda x
add 1
sta x
outp 1
```

Slika 3: Izsek kode, ki prikazuje implementacijo premika v desno. Vir: https://lniv.fe.uni-lj.si/cpe/cpu.html ; 16.2.2024

5.3. Implementacija dogodka “win”

Skupaj s pomikom v desno je implementirano zaznavanje pogojev za dogodek “win”. Ob pogojih, se ta sproži in izpelje zaključek igre.

```
lda x
sbt zx
jze win_loop
```

Slika 4: Izsek kode z mehanizmom za zaznavanje pogojev za dogodek "win". Vir: https://lniv.fe.uni-lj.si/cpe/cpu.html ; 16.2.2024

```
win_loop:
        lda 1
        sta win
        outp 10
        inp t2
        nota
        sbt 4094
        jze init_loop
        jmp win_loop
```

Slika 5: Izsek kode, ki predstavlja dogodek "win" znotraj procesorja. Vir: https://lniv.fe.uni-lj.si/cpe/cpu.html ; 16.2.2024

5.4. Implementacija življenjske spremenljivke

Življenjska spremenljivka je obratno sorazmerna številu trkov, ki jih beleži procesor. Ta spremenljivka je nato vrnjena kot podatek za grafični indikator preostalih življenj.

```
lives_loop:
        lda lives
        jze reset_pos
        sbt 1
        sta lives
        outp 8
        jmp reset_pos
```

Slika 6: Izsek kode, ki prikazuje potek prilagajanja življenjske spremenljivke. Vir: https://lniv.fe.uni-lj.si/cpe/cpu.html ; 16.2.2024

5.5. Implementacija dogodka “game_over”

Dogodek »game_over« je povezano s številom trkov. Ko to prekorači določeno mejo, se izvede dogodek oziroma se igra konča.

```
lda ntrki
shl
add 1
sta ntrki
outp 6
sbt 31
jze end_pos
```

Slika 7: Mehanizem za prepoznavanje pogojev za dogodek "game_over". Vir: https://lniv.fe.uni-lj.si/cpe/cpu.html ; 16.2.2024

```
end_pos:
        lda null
        add 275
        sta y
        outp 2
        lda null
        sta x
        outp 1
        jmp end_loop
```

Slika 8: Izsek kode ponastavitve koordinat objekta. Vir: https://lniv.fe.uni-lj.si/cpe/cpu.html ; 16.2.2024

```
end_loop:
        lda 1
        sta end
        outp 9
        inp t2
        nota
        sbt 4094
        jze init_loop
        jmp end_loop
```

Slika 9: Izsek kode, ki predstavlja dogodek "game_over" v procesorju. Vir: https://lniv.fe.uni-lj.si/cpe/cpu.html ; 16.2.2024

5.6. Ponastavitev koordinat objekta ter pomožnih spremenljivk

Ob trku se ponastavijo koordinate objekta ter nekatere pomožne spremenljivke.

```
reset_pos:
        lda null
        sta x
        outp 1
        add 5
        sta y
        outp 2
        lda ntrki
        shl
        add 1
        sta ntrki
        outp 6
        sbt 31
        jze end_pos
        jmp start_loop
```

Slika 10: Izsek kode, ki izvede ponastavitev koordinat objekta ter spremembo spremenljivke, ki hrani število trkov. Vir: https://lniv.fe.uni-lj.si/cpe/cpu.html ; 16.2.2024

6. I/O ENOTA

I/O enota skrbi za prenos vrednosti v in iz procesorske enote. To izvede s sekvenčnim branjem ter pisanjem na naslovih, ki smo jih predvideli.

```
if (rw = 1) and (adr = 1) then
   x1 <= datin;
```

Slika 11: Izsek VHDL kode, ki predstavlja branje podatka iz procesorja. Vir: sistem.qpf ; 16. 2. 2024

Zanimivost na katero sem naletel je bilo reševanje problema neželene večkratne izvedbe procesov in dogodkov po prejemu signala »trk«, ki se je zgodil zaporedno zaradi časovne neusklajenosti ponastavitve spremenljivke »trk«.

```
if trk = '0' and x1 = 5 and y1 = 10 then
   int_e <= '0';
end if;
if trk = '1' and int_e = '0' then
int <= '1';
int_e <= '1';
end if;
```

Slika 12: Izsek VHDL kode z rešitvijo časovne neusklajenosti spremenljivke "trk". Vir: sistem.qpf ; 16. 2. 2024

7. GRAFIKA

Največji del igre se odvija v tem bloku. Tu se izrisujejo grafike ter ugotavlja prekrivanje določenih objektov. Tu sem implementiral različne elemente. Izraziti elementi so:

- Izris enostavne grafike,
- Izris grafike iz spomina,
- Skaliranje grafike,
- Časovno spreminjanje grafik,
- Ugotavljanje prekritja določenih objektov in proženje trka.

7.1. Izris enostavne grafike

Izris enostavne grafike je sestavljen iz nekaj elementov, in sicer so to:

- Ponastavitev koordinat izvorne točke objekta,
- Nastavitev gabaritov objekta,
- Izris objekta,
- (v primeru premikanja) dodatek časovnika oziroma števca.

7.1.1. Ponastavitev koordinat objekta

Koordinato objekta v tem primeru nastavimo tako, da od trenutne koordinate izrisa odštejemo število točk za katero želimo zamakniti izvorno točko objekta v desno oziroma v primeru koordinate y, navzdol.

```
xt3_1 <= x - 150;
yt3_1 <= y;
xt3_2 <= x - 150;
yt3_2 <= y - 325 - wall_1_cnt;
```

Slika 13: Izsek VHDL kode, ki prikazuje ponastavitev koordinat izvorne točke objekta. Vir: sistem.qpf ; 16. 2. 2024

7.1.2. Nastavitev gabaritov objekta

Ker izrisujemo enostavno grafiko v obliki pravokotnika, mu moramo podati dve meri, in sicer višino in širino. V primeru, kot je v mojem izdelku, pa je dodan še števec, ki dinamično spreminja višino objekta.
Nastavljene gabarite nato v istem koraku uporabimo tako, da pogojno upravljamo spremenljivko, ki narekuje ali je izrisujoča točka v objektu ali ne.

```
w3 <= '1' when (xt3_1<10 and yt3_1<(250+wall_1_cnt)) OR (xt3_2<10 and yt3_2<(275-wall_1_cnt)) else '0';
```

Slika 14: Izsek VHDL kode, ki določa gabarite ter določa ali je trenutna točka izrisa znotraj objekta. Vir: sistem.qpf; 16. 2. 2024

7.1.3. Izris objekta

Izris je zaradi prejšnjega koraka veliko bolj enostaven, in sicer nastavi določeno barvno točko znotraj okna 800 x 600 točk, na željeno 6-bitno barvo, v kolikor je pogoj, v tem primeru »w3«, vrednosti 1.

```
elsif w3 = '1' then
    rgb <= "011000"; --stena_1
```

Slika 15: Izsek iz VHDL kode, ki izriše objekt. Vir: sistem.qpf ; 16. 2. 2024

7.1.4. Dodatek časovnika oziroma števca

V določenih primerih, kadar sem želel, da se objekti s časom spreminjajo, sem uporabil števce kot časovnike, in sicer na način, da so šteli s hitrostjo vhodne ure, do modula in nato sami oddali pulz oziroma naredili spremembo.
Aktualni primer je tukaj parameter »wall_1_cnt«, ki ga poganja števec »sp«.
Ta števec vsebuje tudi funkcijo štetja navzgor do 75 ter navzdol do 0. S tem sem ve prej opisanem koraku izvedel pomikanje objekta.

```
sp: process(clk)
begin
 if rising_edge(clk) then
  if s = 500000 and way3 = '0' then
      wall_1_cnt <= wall_1_cnt + 1;
      s <= (others=>'0');
  elsif s = 500000 and way3 = '1' then
      wall_1_cnt <= wall_1_cnt - 1;
      s <= (others=>'0');
  else
      s <= s + 1;
  end if;
  if wall_1_cnt = 75 then
      way3 <= '1';
  elsif wall_1_cnt = 0 then
      way3 <= '0';
  end if;
 end if;
end process;
```

Slika 16: Izsek VHDL kode, kjer je opisani števec. Vir: sistem.qpf ; 16. 2. 2024

7.2. Izris grafike iz spomina

Poleg izrisa preproste grafike lahko z manjšim dodatkom kodi izrišemo grafiko, ki smo jo predhodno shranili v spomin.
Tako sem v grafičnem orodju izrisal nekaj objektov ter jih v binarnem zapisu vnesel v 8 ROM celic.
V prvem koraku sem prav tako kot pri preprosti grafiki nastavil koordinate. Temu pa so sledili nekoliko drugačni koraki:

- Nastavitev naslova branega bita v spominski celici,
- Branje iz spomina,
- Izris grafike.

7.2.1. Nastavitev naslova branega bita v spominski celici

V tem koraku sestavimo naslov bita v ROM spominu, ki ga želimo prebrati.
To storimo tako, da vzamemo prvih 5 bitov iz vsake koordinate ter jih združimo v eno spremenljivko. Tako dobimo naslov.

```
adr <= (yt(4 downto 0) & xt(4 downto 0));
```

Slika 17: Izsek VHDL kode, ki opisuje nastavitev naslova iskanega bita v pomnilniku ROM. Vir: sistem.qpf ; 16. 2. 2024

7.2.2. Branje iz spomina

Podatki v spominu so enobitni in opisujejo ali je točka na zaslonu obarvana ali ni.
Podatek sem zajel s pomočjo naslova. Ta se nato, kot pri enostavnem izrisu, shrani v enobitno spremenljivko.

```
data <= rom(to_integer(adr));
```

Slika 18: Izsek VHDL kode, ki prikazuje zajem podatka iz pomnilnika. Vir: sistem.qpf ; 16. 2. 2024

7.2.3. Izris grafike

Izris grafike iz spomina se nekoliko razlikuje od preproste, saj se naslov ob koncu branja prelije in branje se ponovno začne. To se izvaja preko celega časa risanja na zaslon. Posledica tega je matrika grafik preko celega zaslona. V kolikor to ni naš cilj, moramo omejiti izris le na željena območja.

```
elsif ((xt_trunk < 32) and (yt_trunk < 32)) and data_trunk = '1'
      rgb <= "101000";
```

Slika 19: Izsek VHDL kode, ki prikaže grafiko. Vir: sistem.qpf ; 16. 2. 2024

7.3. Skaliranje grafike

Grafike sem shranjeval v velikosti 32X32 bitov oziroma točk in s tem varčeval s spominom. Ker pa sem v nekaterih primerih želel, da je grafika večja, sem uporabil metodo skaliranja, in sicer tako, da sem pri branju datoteke vsak bit prebral večkrat. To se v kodi odraža v gradnji naslova z biti koordinat. Za primer sem uporabil bite med 6. in 1. bitom.

```
adr_1 <= (yt_b1(5 downto 1) & xt_b1(5 downto 1));
```

Slika 20: Izsek VHDL kode, ki gradi naslov s skaliranjem. Vir: sistem.qpf ; 16. 2. 2024

7.4. Časovno spreminjanje grafik

Kot že omenjeno lahko za časovno odvisnost uporabim časovnike oziroma števce. Z njimi glede na vhodno uro odmerjam časovne intervale in z njimi generiram pulze ali druge spremembe. Še en primer uporabe števca je pri animaciji trave, kjer sem z njim na določen čas spremenil prikazano grafiko. Sam števec je spremenil naslov grafike, kar se je upoštevalo pri branju iz spomina.

```
grass: process(clk)
begin
 if rising_edge(clk) then
  if grass_st=25000000 then
      grass_st <= (others=>'0');
      grass_graph <= not(grass_graph);
    else
      grass_st <= grass_st + 1;
    end if;
  end if;
end process;
```

Slika 21: Izsek VHDL kode s števcem za animacijo trave. Vir: sistem.qpf ; 16. 2. 2024

```
data_grass <= rom_2(to_integer(adr_2)) when grass_graph = '0' else rom_2_2(to_integer(adr_2));
```

Slika 22: Izsek VHDL kode s pogojnim bralnikom spomina. Vir: sistem.qpf ; 16. 2. 2024

7.5. Ugotavljanje prekrivanja in generacija trka

V določenih primerih sem želel, da je upoštevano tudi to, da se objekti prekrivajo, in sicer v primerih trka objektov. Jaz sem preverjal kadar se je glavni objekt prekril z drugimi. To sem izvedel s preverjanjem pojava bita drugega za izris objekta ob izrisu glavnega objekta.

```
elsif xt<32 and yt<32 and data = '1' and wgr = '1' then
      --rgb <= "111111";
      trk <= '1';
```

Slika 23: Izsek VHDL kode za preverjanje trkov glavnega objekta. Vir: sistem.qpf ; 16. 2. 2024

8. POVZETEK SINTEZE

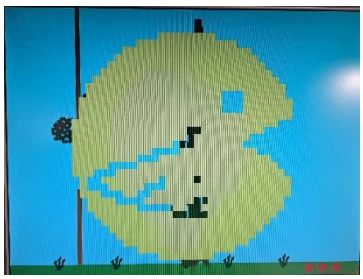
PODATEK	VREDNOST
Število vseh logičnih elementov	1791/22320 (8%)
Skupno število registrov	460
Skupno število nožic	22/154 (14%)
Skupno število spominskih bitov	3072/608256 (<1%)

Tabela 1: Povzetek sinteze. Vir: sistem.qpf ; 16. 2. 2024

9. POVZETEK IZDELKA

Slika 24: Zajem zaslona končne igre. Vir: fotografija zaslona 16. 2. 2024

Končni izdelek je igra, kjer rumeni okrogli ptiček pod našim vodstvom potuje proti desni strani zaslona. Pri tem se mora izogibati vertikalno premikajoči steni, rastočemu drevesu, oblaku ter tlom. Pomika se lahko izključno naprej (v desno) ter navzgor. Sam pa brez uporabnikovega vhoda pada proti tlom. V kolikor se s ptičkom zaletimo v kateri koli omenjeni objekt (izključujoč travo), izgubimo življenje ter se ptiček ponovno pojavi na začetni poziciji, kjer čaka na ponovni poskus. Pri tem lahko v desnem spodnjem kotu s pomočjo src, ki predstavljajo preostala življenja, sledimo tudi številu preostalih poskusov, ki so nam na voljo. Za estetiko poskrbijo še animirane travice na dnu zaslona. Ob zmagi ali porazu nas pričakata končna zaslona.



Slika 25: Zajem zaslona ob zmagi. Vir: fotografija zaslona 16. 2. 2024

Slika 26: Zajem zaslona ob porazu. Vir: fotografija zaslona 16. 2. 2024