Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*

Poročilo pri predmetu načrtovanje digitalnih elektronskih sistemov

Pac-man v FPGA-ju

Avtor: Tilen Tinta
Program: Aplikativna elektrotehnika
3. letnik – Avtomatika
Vpisna številka: 64190463

Ložice, januar 2023

Povzetek

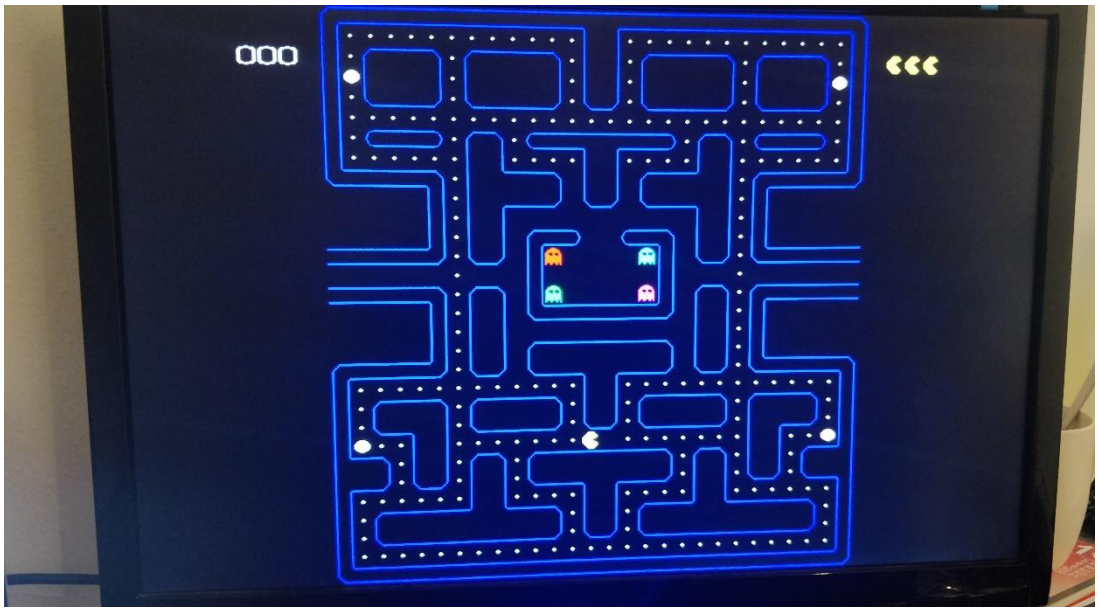
V končnem projektu je bila naša naloga, da nadgradimo sistem za FPGA v programskem okolju Intel Quartus v jeziku VHDL. Sistem smo skozi vaje dopolnjevali z različnimi funkcijami. Večinoma smo se ukvarjali z kodo v grafičnem delu ter signali do procesorja, da smo lahko s tipkami spremenili potek programa. Moj cilj je bil poustvariti igro PAC-MAN. Poskušal sem se ji čim bolj približati in hkrati nekaj stvari spremenil, da so bolj ustrezale mojim zahtevam.

Kazalo

Kazalo slik

1. Uvod

Cilj projekta je bil dograditi in dopolniti celotni sistem na FPGA-ju. Dodati povezave med komponentami, dodati različne funkcije v naše komponente in preko narejenih povezav komunicirati z drugimi komponentami, dopolniti program v zbirniku, ki ga uporablja procesor in temu dodati še kakšen ukaz za izvajanje.
Za ta projekt sem si zastavil, znano igro PAC-MAN. Poskušal sem se ji kar se, da približati s funkcijami in izgledom krati pa je bilo zanjo potrebno implementirati vse zahtevane dodatke. Z to igro sem lahko uporabil vse tipke, ki so nam bile na voljo na razširitveni ploščici. To mi je omogočalo dopolniti zbirnik, ki pa se je kasneje izkazal da še zdaleč ni bila edina izpopolnitev. Da sem dosegel pravi izgled igre je bilo potrebno narediti veliko grafike ter jo pravilno uporabiti.



Slika 1: Izgled igre

2. Delovanje sistema

Sistem sem dograjeval v treh blokih. To so: grafika, io in sistem.

V datoteki sistem sem vzpostavil dodatne signale med blokoma grafika in io. Ti so bili potrebni za pravilno delovanje tipk ob vodenju pacmana skozi labirint. Podrobnejše delovanje za nastavljanje vrednosti teh spremenljivk bom opisal v nadaljevanju.

Ravno tako sem te iste signale, ki vplivajo na premikanje junaka po labirintu dodal v blok io. Te štiri signale, sem dodal v kodo bo znal naš procesor brati in glede na njihovo vrednost pravilno izvajati program v zbirniku.

Glavni del programa se dogaja v bloku grafika.

2.1. Grafični del

Ker mi je bila pomembno kako igra zgleda je bila moja prva naloga kako narisati še kar zahteven labirint. Način kot smo uporabili pri vajah mi ni bil v pomoč da bi narisal celotno sliko saj bi to pomenilo 800x600 veliko tabelo. To bi bilo zelo dolgotrajno risanje in hkrati bi najverjetneje zasedla velik del celic na FPGA-ju. Po podrobnejšem pregledu labirinta iz igre in štetju »kvadratkov« sem prišel na idejo da bi z manjšo prilagoditvijo lahko labirint razdelil v dele. Sestavljen je iz le nekaj osnovnih elementov kateri se čez celoten labirint ponavljajo. Primer so dve ravni črti/steni, ki sta na celotni spodnji in zgornji stranici. Enak element se pojavi tudi na levi in desni strani čez celo stranico. Ideja je bila, da če eno takšno sliko narišem lahko za drugo stranico enako grafiko samo s kodo rotiram za 90 stopinj. Enako sem storil za vse vogale, kotnike, črte…

```
--//// Data za labirint ////
p_Block_data: process(y_maze_location, x_maze_location, y_block_cnt, x_block_cnt, data_labirint, Osnova, tocke_data)
begin

   data_labirint <= Osnova(to_integer(y_maze_location), to_integer(x_maze_location)); -- Lokacija labirinta

   case data_labirint is
      when 0 =>
         data_block <= Blok_Blank(to_integer(y_block_cnt(4 downto 0) * "10100" + x_block_cnt(4 downto 0))); -- Crno / prazno
      when 1 =>
         data_block <= Blok_Lines(to_integer(y_block_cnt(4 downto 0) * "10100" + x_block_cnt(4 downto 0))); -- Horizontalno dve crti
      when 2 =>
         data_block <= Blok_Lines(to_integer(x_block_cnt(4 downto 0) * "10100" + y_block_cnt(4 downto 0))); -- Vertikalno dve crti
      when 3 =>
```

Slika 2: Nastavljanje podatkov za izris grafike

Labirint iz originalne igre Pacman bi lahko razdelili na bloke 28 po x koordinati in 31 po y koordinati. Ker pa to ni deljivo z našo resolucijo sem jo spremenil tako, da ima po y koordinati 30 blokov, kar nam da velikost enega bloka 20 pikslov.
S tem načinom sem dobil tabelo 40x30 in vsak blok velikosti 20x20 pikslov.

```
-- /// LABIRINT (Bloki in osnova) ///
type deklaracija is array (0 to 29, 0 to 39) of integer range 0 to 30;
signal Osnova: deklaracija:=(
```

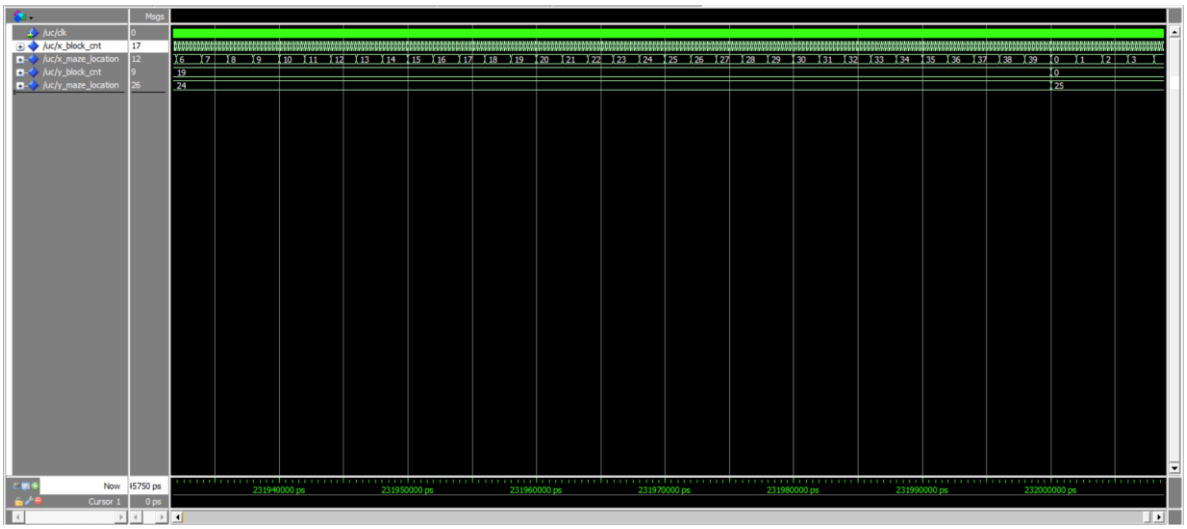
Slika 3: Deklaracija glavne tabele za labirint

Z logiko štirih števcev, ki se med seboj ustrezno povečujejo in ponastavljajo sem izvedel sekvenco, ki izriše labirint.

```
-- ** Ozadje (labirint) - logika **
if x_block_cnt = 19 then -- reset x blok stevca
   x_block_cnt <= (others=>'0');
   if x_maze_location = 39 then -- konec vrstice (labirint)
      x_maze_location <= (others=>'0'); -- reset vrstice
      if y_block_cnt = 19 then
         y_block_cnt <= (others=>'0');
         if y_maze_location = 29 then
            y_maze_location <= (others=>'0'); -- reset y
         else
            y_maze_location <= y_maze_location + 1;
         end if;
      else
         y_block_cnt <= y_block_cnt + 1;
      end if;
   else
      x_maze_location <= x_maze_location + 1; -- zamik na naslednjo x lokacijo labirinta
   end if;
else
   x_block_cnt <= x_block_cnt + 1; -- x premikanje po bloku
end if;
```

Slika 4: Logika izrisa labirinta

Primer simulacije v ModelSimu za del kode, ki nadzoruje števce za izris labirinta. Vhod v simulacijo je takt procesorja kateri je tudi vhodni pogoja za ta proces. V logiki imamo štiri spremenljivke. Ob vsakem taktu se poveča x_block_cnt za ena. Ko ta doseže vrednost 19 se ponastavi na 0 in istočasno poveča za ena spremenljivko x_maze_location. To se ponavlja do konca vrstice. Takrat je x_maze_location enak 40. Nato se ta ponastavi na 0 in poveča y_block_cnt za ena. To nam začne izrisovati naslednjo vrstico. Ta cikel se ponavlja dokler y_block_cnt ne doseže 19. Takrat se ta ponastavi na 0 in poveča y_maze_location za ena. Tako se ta celoten cikel ponavlja dokler y_maze_location ne doseže vrednosti 30. Vse te vrednosti iz cikla podajamo kot podatke za lokacijo pri izrisu labirinta. Ta cikel se začne ob priključitvi napajanja na ploščico in se stalno ponavlja.



Slika 5: Simulacija v ModelSimu

Sam pacman je narisan s štirimi slikami, ki se med seboj preko števca menjajo. To nam omogoča animacijo ust. Z enako veliko sliko so narisani tudi duhci, le da ti niso animirani. Njihov izris se izvede na enak način kot smo ga uporabljali že na vajah.
V zgornjem desnem kotu imamo prikazana življenja v igri. Zanje je uporabljena ena od slik pacmana iz animacije. Glede na vrednost spremenljivke lives pa se te prikažejo ali ne.

Z naborom slik števil prikazujemo točke v zgornjem levem kotu. Izpišemo jih glede na vrednosti treh števcev. Vsak prikazuje svoj del števil (enice, desetice, stotice), ki se povečujejo glede pobranih točk v igri. Majhna pika nam prišteje eno točko, velika pa deset. Ob pobrani veliki piki dobimo bonus, ki nam omogoča pojesti duhca. V času trajanja bonusa se barva duhcev spremeni v belo. Ko časovnik poteče se spremenijo nazaj v svojo barvo in jih ne moremo več pojest. Če duhca v času bonusa pojemo se ta ne izrisuje več.

2.2. Logika igre

Za igranje igre se uporablja vse štiri tipke na razširitveni ploščici. Vsaka je namenjena premikanju v svojo smer. Tipka na sami ploščici s samim FPGA-jem ki je namenjena za ponastavitev pa poleg ostalih komponent ponastavi tudi grafiko.

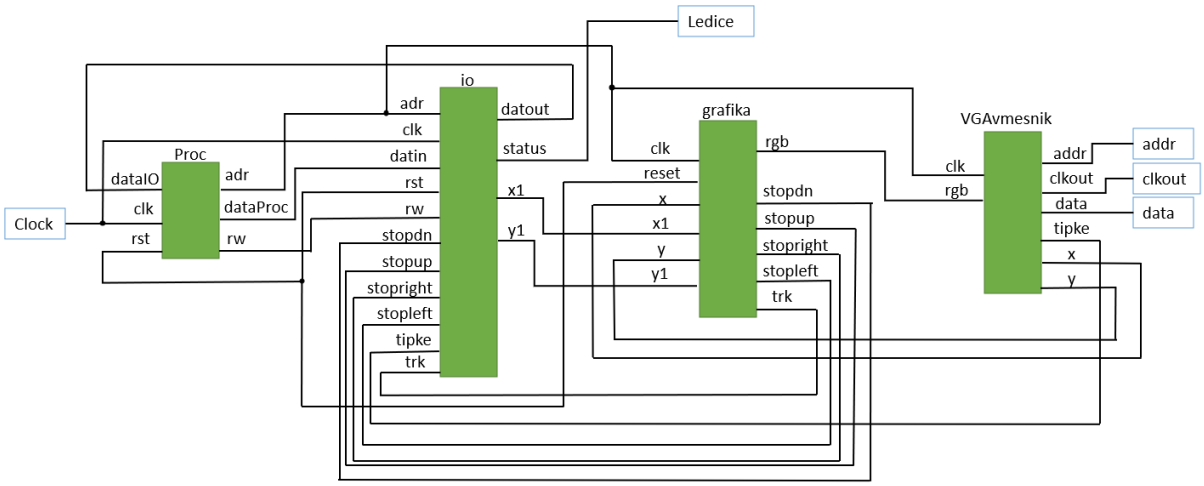
Kot že zgoraj omenjeno se tudi celotna igra ko pride do premikanja naslanja na osnovno tabelo 40x30. Ob začetku igre se pacman pojavi na določenih koordinatah. V ozadju mu sledita dva števca ki sta vedno usklajena s lokacijo pacmana. Glede na to lokacijo lahko gledamo okolico našega junaka in določamo dovoljeno smer premikanja. Kam se lahko premikamo zapišemo v vrednost štirih spremenljivk. Vsaka je namenjena za svojo smer. Te so preko bloka sistem speljane v blok io, kjer se njihova vrednost zapiše v spremenljivke do katerih dostopa procesor.
Koda v zbirniku, ki teče na našem procesorju te prebere in glede na njihovo vrednost spreminja pot izvajanja programa. Če zazna steno je vrednost spremenljivke enaka nič. V tem primeru pride do skoka na značko kjer se bere naslednjo tipko. Del kode, ki smo preskočili ni spremenil vrednosti spremenljivke x ali y in s tem je ostala lokacija našega pacmana enaka.
Tudi vsak duhec ima svoje lastne koordinate. Ker vemo velikost slike, ki jih izriše lahko z malo matematike zaznamo trk med njimi in pacmanom. V tem primeru je izvedejo vsi ukazi za trk v vhdl kodi, hkrati pa tudi pošlje signal v naš procesor, ki preko zbirnika ponastavi koordinate pacmana.
Za nabiranje točk se moramo s pacmanom premikati po blokih kateri vsebujejo vrednosti, ki nam izrisujejo pike. Če pridemo na to polje se vrednosti le tega postavi na 0 kar nariše samo prazen kvadrat nam pa prišteje točko. Enako se zgodi z večjimi pikami ki so bonus. Te sprožijo časovnik in spremenijo barvo duhcev. Trk pacmana z duhcem deluje enako kot prej, le da nam v tem primeru ne pobere življenja in zmanjša točk ampak spremeni status spremenljivke posameznega duhca, ki nam pove ali je ta izrisan in delujoč ali ne.
Premikanje duhcev je avtonomno. Njihov izhod iz »škatle« na začetku je zapisan v kodi, od tam naprej pa nanje vpliva okolica. Vsak ima neko osnovno logiko kam se bo premaknil in ta se razlikuje med njimi, da gibanje ni popolnoma enako. Ravno tako kot spremljam okolico pacmana se spremlja tudi okolica vsakega duhca. Ta vpliva na spremenljivke, ki duhcu povejo smer.

```
   -- preverjanje kje je pot
case (Osnova(to_integer(D1_y_maze),to_integer(D1_x_maze + 1))) is
   when 00 =>
      D1_right <= '0';
   when 28 =>
      D1_right <= '0';
   when 29 =>
      D1_right <= '0';
   when others =>
      D1_right <= '1';
end case;
```

Slika 6: Zaznavanje ovir x smeri za duhca

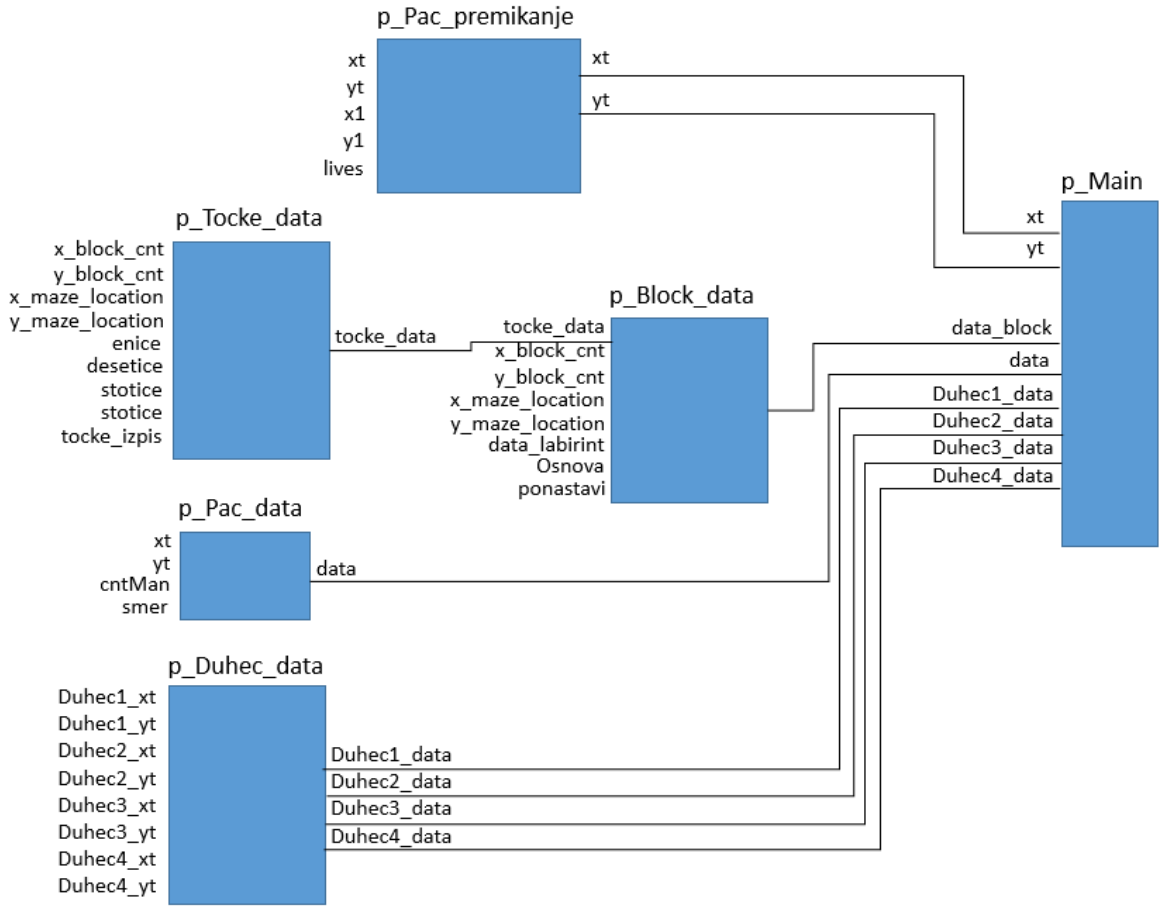
Logika za duhce deluje na način, da ne glede na labirint se bo duhec znal premikati po njem.

2. Blokovna shema sistema



Slika 7: Shema pomembnejših blokov in povezav

Povezave med procesi so najbolj razvidne v bloku grafika saj jih je tam največ oziroma sem jih tam zaradi velike količine kode dodajal.



Slika 8: Povezave med procesi v bloku Grafika

3. Bloki v sistemu

5.1. procpak

Blok vsebuje transformacijo med binarnimi kodami, ki jih procesor razume in nam poznanimi oz. človeku berljivimi ukazi v zbirniku. To man omogoča pisanje programov v klasičnem jeziku kar zelo pohitri in poenostavi programiranje. Komponento smo dobili kot že pred narejeno in se z njo nisem ukvarjal v mojem programu.

5.2. proc

Je blok, ki služi za povezavo signalov med komponentama io in cpu ter signali, ki so bolj splošni v sistemu kot je na primer reset signal. Ravno tako nam je bil tudi ta blok podan in ga nisem nič spreminjal ali dopolnjeval.

5.3. cpu

V bloku cpu najdemo mikroprocesor katerega smo programirali že v miniprojektu. Tudi ta nam je bil že podan. To pa zato, ker vsebuje vse ukaze, ki smo jih imeli napisane kot možnosti implementacije v naš miniprojekt, še posebej pa ker ni imel napak v strukturi programa. Ravno tako je tudi ta blok ostal nespremenjen.

5.4. VGA vmesnik

Ravno tako kot ostali bloki zgoraj že opisani je tudi ta nam bil že podan. Vsebuje logiko, ki upravlja z VGA priključkom za izris slike na ekranu. Tudi ta je ostal nespremenjen.

5.5. sistem

Blok je prišel z predlogo projekta vendar sem ga za svojo nalogo moral dopolniti s štirimi signali. To so stopup, stopdn, stopright, stopleft. Omenjeni signali so iz grafike, ki nam omejujejo premikanje junaka glede na njegovo lokacijo v labirintu povezani v io, kjer jih predamo procesorju, ki z njimi operira.

```
U3: entity work.Grafika port map (
    clk => clk,
    x => x,
    y => y,
    x1 => x1,
    y1 => y1,
    rgb => rgb,
    reset => rst, -- dodano za reset
    stopup => stopup,
    stopdn => stopdn,
    stopleft => stopleft,
    stopright => stopright,
    debug => debug,
    trk => trk);
```

```
U4: entity work.io port map(
    clk => clk,
    rw => rw,
    adr => adr,
    rst => rst,
    datin => dataProc,
    datout => dataIO,
    tipke => tipke,
    x1 => x1,
    y1 => y1,
    modul => delilnik,
    trk => trk,
    status => led(0),
    stopup => stopup,
    stopdn => stopdn,
    stopleft => stopleft,
    debug => debug,
    stopright => stopright
);
```

Slika 9: Povezava signalov med bloki

5.6. io

Blok io je ravno tako prišel s predlogo in sem ga v mojem primeru ravno tako moral dopolniti za željeno delovanje. Iz bloka sistem vanj pripeljemo signale, ki izvirajo iz bloka grafika. Te signale tukaj pravilno usmerimo na prave naslove iz katerih nato s procesorjem beremo vrednosti.

```
elsif (adr = 5) and (rw = 2) then -- trk
   datout <= (0 => int, others => '0');
   int <= '0';
elsif (adr = 6) and (rw = 2) then -- dodano za stop na gor
   datout <= (0 => not(stop_up), others => '0');
   stop_up <= '0';
elsif (adr = 7) and (rw = 2) then -- dodano za stop na dol
   datout <= (0 => not(stop_dn), others => '0');
   stop_dn <= '0';
elsif (adr = 8) and (rw = 2) then -- dodano za stop na levo
   datout <= (0 => not(stop_left), others => '0');
   stop_left <= '0';
elsif (adr = 9) and (rw = 2) then -- dodano za stop na desno
   datout <= (0 => not(stop_right), others => '0');
   stop_right <= '0';
elsif rw = 2 then
   datout <= (0 => tipke(to_integer(adr)), others => '0');
end if;
```

Slika 10: Razvrščanje signalov po naslovih

5.7. grafika

Je blok s katerim sem se največ ukvarjal. V njem najdemo celotno kodo za izris grafike na zaslonu in logiko za igranje igre.

6. Program za procesor v zbirniku

Prikazan je začetek programa v zbirniku do druge tipke. Program se štirikrat ponovi z enako logiko le drugimi spremenljivkami.

```
    lda 4000        ;naloži vrednost ki bo uporabljena za delilnik takta
    outp  do0

start: lda 380      ;naloži vrednost x koordinate pacmana
    sta x           ;shrani naloženo vrednost v spremenljivko x
    outp 1
    lda 440         ;naloži vrednost y koordinate pacmana
    sta y           ;shrani naloženo vrednost v spremenljivko y
    outp 2

trk: inp rst        ;trk - preveri vrednost spremenljivke in če je 1 se pacman resetira
    jze loop0       ;če trk ni izpolnjen, skok na preverjanje tipk
    jmp start       ;če je trk izpolnjen, skok na start značko za reset koordinat

loop0: inp t0       ;preverjanje prve tipke (levo)
    jze loop1       ;če ni pritisnjena skok na značko loop1 - ne premeni vrednosti x
    inp stopleft    ;preverjanje ali se lahko pacman premakne v to smer (stene labirinta)
    jze loop1       ; če je 0 pomeni da je ob steni in v to smer se ne da premaknit, x ostane enak
    lda x
    sbt 1
    sta x
    outp 1
    jmp trk
```

```
loop1:  inp t1          ;enaka sintaksa kot zgoraj le za drugo tipko
    .
    .
    .
```

7. Rezultati sinteze

	Hierarhija	Kombinaciske ALUTs	Dodeljeni logični registri	Spomin (biti)
1	-Sistem			3070
1	VGAvmesnik	60(60)	48(48)	0
2	Grafika	19744(19275)	1052(1052)	0
3	io	43(43)	42(42)	0
4	proc	210(0)	97(0)	3070
5	sld_hub:auto_hub	118(8)	86(0)	0

8. Zaključek

Izdelava zaključnega projekta mi je zelo dobra ideja saj nam pusti prosto pot za programiranje in izdelavo svojih idej. Z reševanjem problemov, ki so se mi sproti pojavili sem se veliko naučil saj sem lahko na različne načine prišel do rešitve. Lahko po daljših a enostavnejših poteh ali krajših in zahtevnejših pri katerih sem velikokrat spoznaval novo sintakso. Ker sistem deluje kot celota v uporabo več komponent nam je dal dobro predstavo nekega resničnejšega sistema, ki bi ga lahko srečali v resničnem svetu. Projekt mi je nedvomno dal veliko zanimanje za FPGA-je in njihove aplikacije.

5. Viri

1. PDF-ji vaj Načrtovanje digitalnih elektronskih sistemov