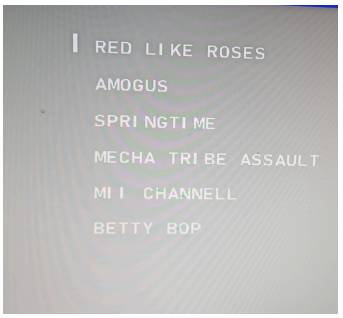
NDES POROČILO VGA IGRE
FPGAMANIA

Jure Sivec
Ljubljana, 2024

UVOD:
FPGAMANIA je poenostavljena verzija nekdaj popularne igre DanceDanceRevolution, kjer igralec s pritiski na gumbe zadane dvigajoče se puščice ravno takrat, ko ti dosežejo njihov “okvir” na vrhu zaslona. Igra si pri tem beleži točnost zadetka in igralcu za vsako puščico (ki je bila vsaj blizu) dodeli 30, 15 ali 10 točk, rezultat pa se v odstotkih prikazuje na vrhu zaslona in v stolpičnem diagramu na desni strani. Vsako igro lahko tudi ustavimo, s tem se izriše tudi meni pavze z opcijami nadaljuj, resetiraj, izhod. Če kliknemo na izhod, pridemo na pozdravni zaslon z napisom FPGAMANIA. Po pritisku desne tipke se prikaže meni s šestimi različnimi pesmimi, od katerih je vsaka različno dolga in vsaka ima svoj tempo in s tem težavnost. Takoj po pritisku na desno tipko se izbrana pesem začne.

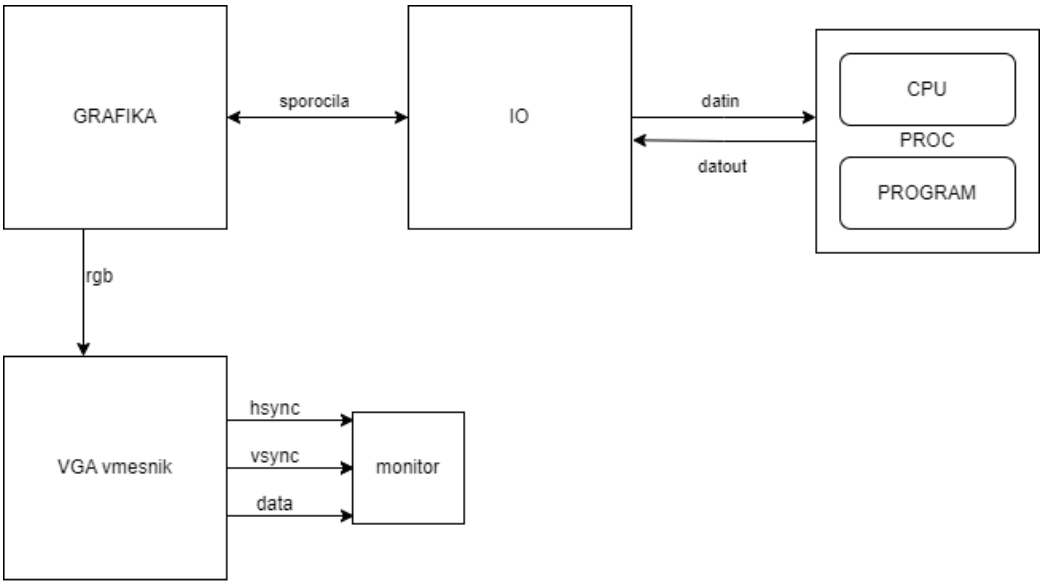
Pozdravni zaslon

Menu pavze

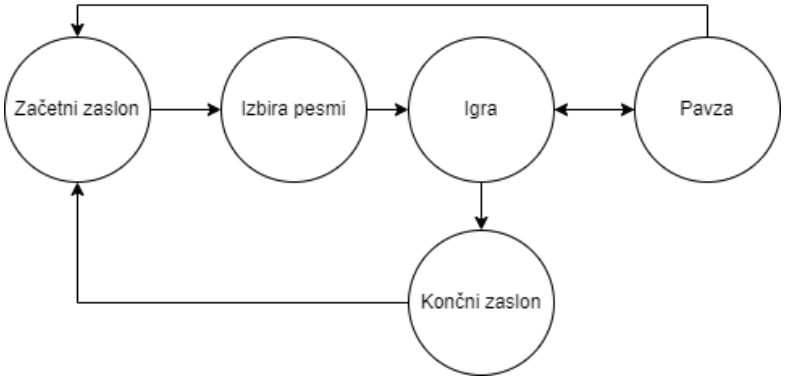


Izbira pesmi

OPIS:
Večina logike igre je napisana v grafika.vhd. V njej se izračunajo vsi objekti, to je puščice, stolpci, rezultat, okvirji za puščice, indikator za pavzin menu, meniji in pisave in selektor za meni, welcome screen…
Proc pa se uporablja za detekcijo pritiska gumba, nato debounce filtra, računanja rezultata za posamezno puščico. Proc prejme podatke o števcu, trenutnih puščicah in tipkah preko io. Cpu nazaj kodirane podatke pošilja po enem vodilu.
Io komponenta deluje kot vmesnik med grafiko in proc-om, VGA vmesnik pa vzame generiran rgb signal, in ga skupaj s hsyncom in vsyncom pošlje na monitor.
Vse komponente so zvezane v komponenti sistem, kjer lahko nastavimo tudi hitrost ure.



Glavni del igre je avtomat stanj, s pomočjo katerega igra opravlja različne funkcije in glede na stanja avtomata priredi različne signale:



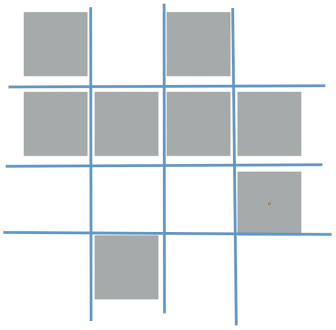
GLAVNE FUNKCIJE IGRE:

Števci (signali):
clk: glavni clock.
cntr_128_scaledown: clk[Mhz] / 128 = 390625 @1BPM. Uporabimo ga, da dobimo counter, ki šteje od 0 do 128 v eni sekundi
cntr_128: šteje od 0 do 128 v eni sekundi (odvisno od tempa)
beat_cntr: poveča se pri vsakem “beatu,” tj. vsakič ko se cntr_128 prelije (odvisno od tempa)
BPM: beats per minute je tempo igre, določamo ga s cntr_128_scaledown= 390625/BPM

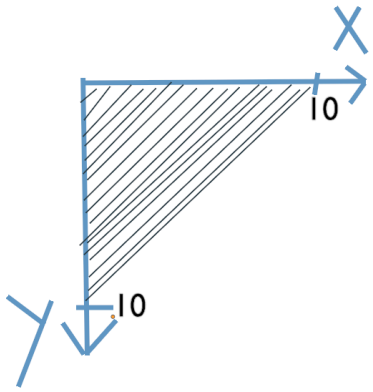
Sporočila (signal):
Ob zaznani spremembi signala sporocila, ki prihaja na grafiko z mikroprocesorja, ga dekodiramo:

sporocila = (d)17	+ 10
sporocila = (d)18	+ 15 točk
sporocila = (d)19	+ 30 točk
sporocila < 16	informacija o tipki
sporocila = 25	NULL

Prikaz puščic:
Za vsak piksel na sliki se ugotavlja ali je ta na območju katerega zaseda puščica. Primer:

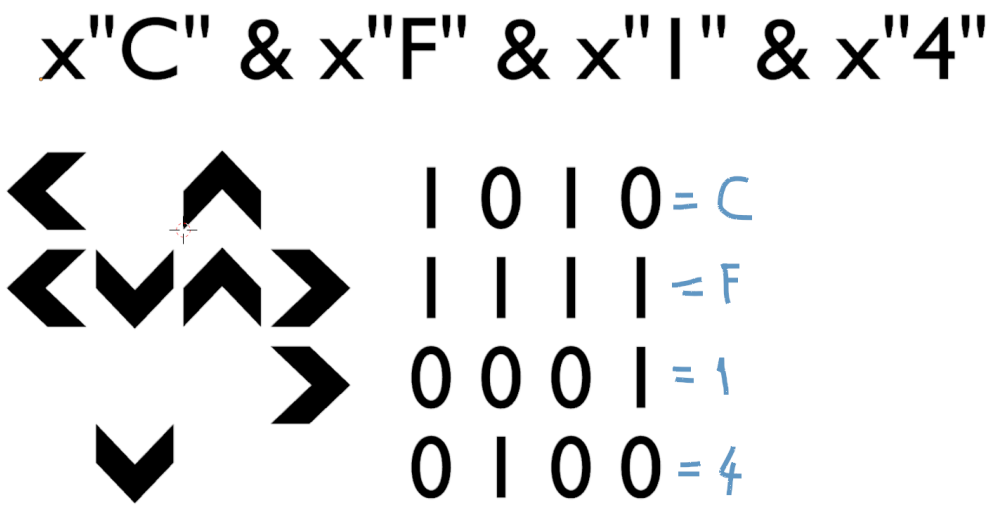
	Višina enega kvadrata je 128 pikslov. Največje število puščic, ki so lahko hkrati na zaslonu je torej 20

Nato se na vsakem mestu, kjer je logična 1, oblikuje puščica s pomočjo > in < operatorjev. Krivino pod kotom 45 stopinj se realizira s funkcijo x+y>a, npr. x+y<10 bi izgledal takole:



Vsa mesta za enke so zapisane v 4352 bitov dolgi konstanti, naenkrat pa gledamo 16 bitov, za vsako vrsto 4. Zato da se puščice premikajo navzgor, spreminjamo okvir 16 bitov navzgor:

```
constant song : songie := (x"C"& x"F"& x"1"& x"4"& x"8"& x"1"& x"8"& x"1"& x"
x"8"& x"1"& x"4"& x"1"& x"4"& x"0"& x"1"& x"0"& x"4"& x"2"& x"8"& x"2"& x"4"&
x"0"& x"0"& x"1"& x"0"& x"0"& x"0"& x"0"& x"0"& x"2"& x"0"& x"0"& x"2"& x"0"&
x"0"& x"0"& x"1"& x"0"& x"0"& x"0"& x"0"& x"0"& x"4"& x"0"& x"0"& x"8"& x"0"&
```



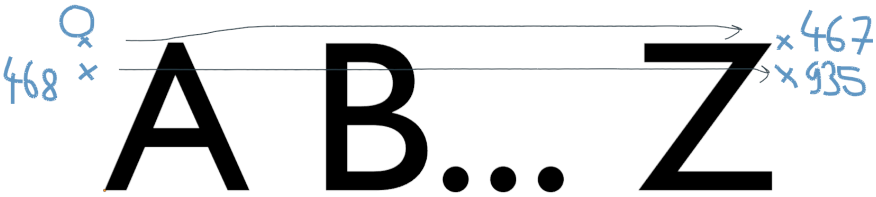
Da pa dobimo tekoče premikanje puščic, dodamo še offset, ki je enako visok kot puščica. in ga v enem ciklu potisnemo za celo njegovo višino navzgor. V nizu imamo 6 “pesmi,” vse iste dolžine. Indeksiranje trenutnih puščic torej izgleda takole:

```
song(to_integer(song_select))(to_integer(x(8 downto 7)) +
to_integer(y_pos(9 downto 7))*4 + to_integer(beat_cntr)*4),
```

kjer je song_select ena izmed 6 pesmi, x in y sta obe koordinati, cntr_128 pa števec.

Izris teksta: *if abeceda(y_koord) = '1' then …*

Abeceda se izrisuje čez cel zaslon, uporablja se ena 8892 bitov dolga konstanta, ki se lahko izrisuje čez cel zaslon.



Abecedo spremenim v tekst tako, da števec, ki ga uporabljam za indeks tabele potisnem za debelino ali več debelin črk naprej ali nazaj in s tem izrišem neko drugo črko. Primer: Če števec pošljem za 1 črko naprej tik pred črko C in nato eno mesto nazaj tik pred črko F, dobim:

Tu je scilab skripta, ki prevede tekst v vhdl konstanto z vsemi odmiki za izris stavka:

```
clc
//vpiši besedo v beseda= "..."
beseda="hello";
//če je dolžina 25 (ali več), potem ne bo delalo
//sprejema samo črke, ki so v abecedi spodaj
beseda_v_vhdl=ones(1:10)*30;
indexi_besede=zeros(1:length(beseda));
zadnji_element=0;
array=['0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0'];
abeceda =['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','x','y','w','z'] ;//dolžina je 26

for i=1:length(beseda)
array(i)=part(beseda,i);
end
for j=1:length(beseda)
   for i=1:26
      if array(j) == abeceda(i)
         indexi_besede(j)=i;
      end

   end
end
for i=1:(length(beseda))
   if i==1
      beseda_v_vhdl(i)=indexi_besede(i)-1;
      zadnji_element=beseda_v_vhdl(i);
   else
      beseda_v_vhdl(i)=indexi_besede(i)-(indexi_besede(i-1)+1);
      zadnji_element=zadnji_element+beseda_v_vhdl(i);
   end
end
 beseda_v_vhdl(length(beseda)+1) = -zadnji_element;
printf("constant %s_zapis: sestindvajsetcrk  :=(", beseda);
for i=1:length(beseda)+1
   printf("%d,", beseda_v_vhdl(i));
end
for i=1:25-length(beseda)
   printf("0");
   if i < 25-length(beseda)
      printf(",")
   end
end
printf(");");
```

Diagram kode mikroprocesorja (desno):

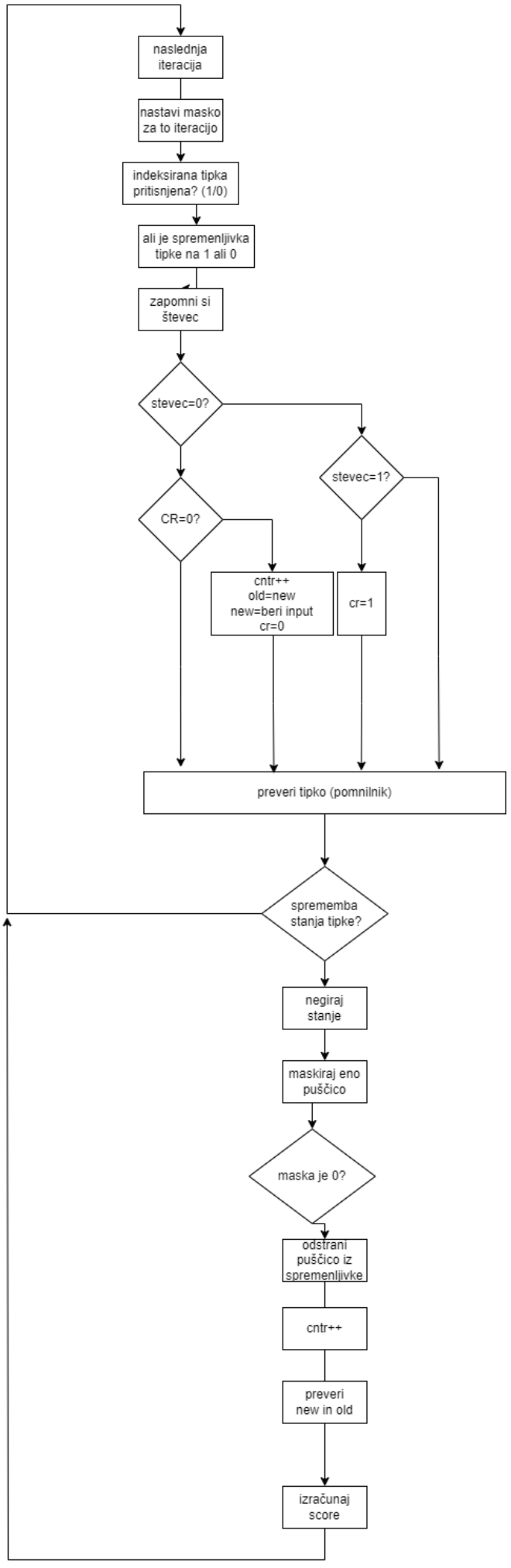
Zanimivejši izseki kode:

```
findID:                 anda maska
                        jze return_function
                         lda 1
return_function:  sta rezultat_findID
                         ret
```

ko pokličemo findID, nam ta vrne 1, če je bit na mestu, določenim z masko enak 1 in 0, če je enak 0.

```
twotothe: sta tempvar
jze load1

sbt 1
jze load2

sbt 1
jze load4

lda 8
sta tempvar
jmp return

load1:   lda 1
sta tempvar
jmp return

load2:   lda 2
sta tempvar
jmp return

load4:   lda 4
sta tempvar
jmp return

return:  lda kamFunk
jze ret1
jmp ret2
```

ko pokličemo twotothe, nam vrne 2^x, x pa je lahko 0,1,2,3



Rezultati sinteze:
Skupni logični elementi: 9 103 od 22 320 (41%)
Skupni registri: 597
Skupaj pomnilniških bitov: 3 071 od 608 256 (<1%)

Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	9,103 / 22,320 (41 %)
Total registers	597
Total pins	22 / 154 (14 %)
Total virtual pins	0
Total memory bits	3,072 / 608,256 (< 1 %)
Embedded Multiplier 9-bit elements	3 / 132 (2 %)
Total PLLs	0 / 4 (0 %)