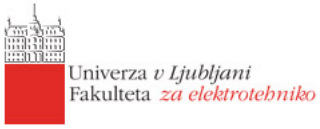


Univerza v Ljubljani

Projektna naloga pri predmetu

Digitalna integrirana vezja in sistemi

Avtor:
Blaž Modic

Mentor:
prof. dr. Andrej Trost

Ljubljana, januar 2015

Kazalo

Slike

1 Uvod

Pri predmetu Digitalna integrirana vezja in sistemi smo projekte delali na razvojni ploščici, ki je vsebovala čip Zynq-7000. Zynq-7000 je sistem na čipu (SoC), ki vsebuje vezje FPGA, dvojederni ARM Cortex A-9 procesor in veliko perifernih enot. Na vajah smo v jeziku VHDL načrtali strojni del aplikacije, ki je vsebovala gonilnik za 8 bitni VGA vmesnik z lastnim delom RAM pomnilnika. Projekt sem nadgradil z aplikacijo v programskem jeziku C, ki je tekla na ARM procesorju. Končni izdelek je bila video igra, ki je nekako spominjala na klasično arkadno igro PacMan.

2 Načrtovanje vezja v VHDL

V programu Vivado smo načrtali vezje v programskem jeziku VHDL, pri čemer smo uporabljali IP komponente. V sledečih poglavjih so predstavljene glavne komponente sistema.

2.1 VGA sinhronizacija

IP blok VGAsinh skrbi za sinhronizacijske signale po VGA standardu. Kontorlirali smo signal hsync za sinhronizacijo vrstic ter signal vsync za osveževanje slike. Točke na zaslonu pošiljamo za vsako vrstico posebaj - signal na DA pretvorniku se za vsako točko v vrstici spremeni po enem urinem ciklu. Ko pridemo do konca vrstice malo počakamo, nato s signalom hsync signaliziramo, da želimo pošiljati podatke za naslednjo vrstico. Ko pošljemo celotno sliko s signalom vsync signaliziramo, da želimo sliko osvežiti na zaslonu. Spodaj je prikazan del VHDL kode, ki skrbi za generacijo signalov hsync in vsync:

```
process(clk50)
begin
if rising_edge(clk50) then
  if hst < H-1 then
    hst <= hst + 1;
  else
    hst <= 0;
    if vst < V-1 then
      vst <= vst + 1;
    else
      vst <= 0;
    end if;
  end if;
end if;
end process;

hsync <= '0' when hst>=Hf and hst<Hf+Hs else '1';
vsync <= '0' when vst>=Vf and vst<Vf+Vs else '1';
```

2.2 VGA izhod

IP komponenta VGAm skrbi za izris slike. Omogoča nam tudi izris kurzorja na zaslonu. Ali se bo kurzor izrisal je odvisno od vhodnega signala ctr ter signala en_i. Signal ctrl vsebuje koordinate, kjer se bo kurzor izrisal ter zastavico, ali naj se kurzor sploh izriše. Za izris kurzorja mora biti postavljen tudi signal en_i. Če oba pogoja nista izpolnjena se na izhod pošlje enaka barva, kakor jo dobimo na vhod IP komponente, le da se signal zakasni za en urin cikel. Funkcionalnosti kurzorja v moji aplikaciji nisem uporabljal.

2.3 VGA RAM

IP komponenta VGAram nam omogoča, da na zaslonu lako prikažemo sliko, ki jo imamo shranjeno v pomnilniku. V komponenti nastavimo signale, ki so potrebni za komunikacijo z BRAM enoto. Prikazan je ukaz, s katerim nastavimo naslov v BRAMu, iz katerega želimo prebrati podatek:

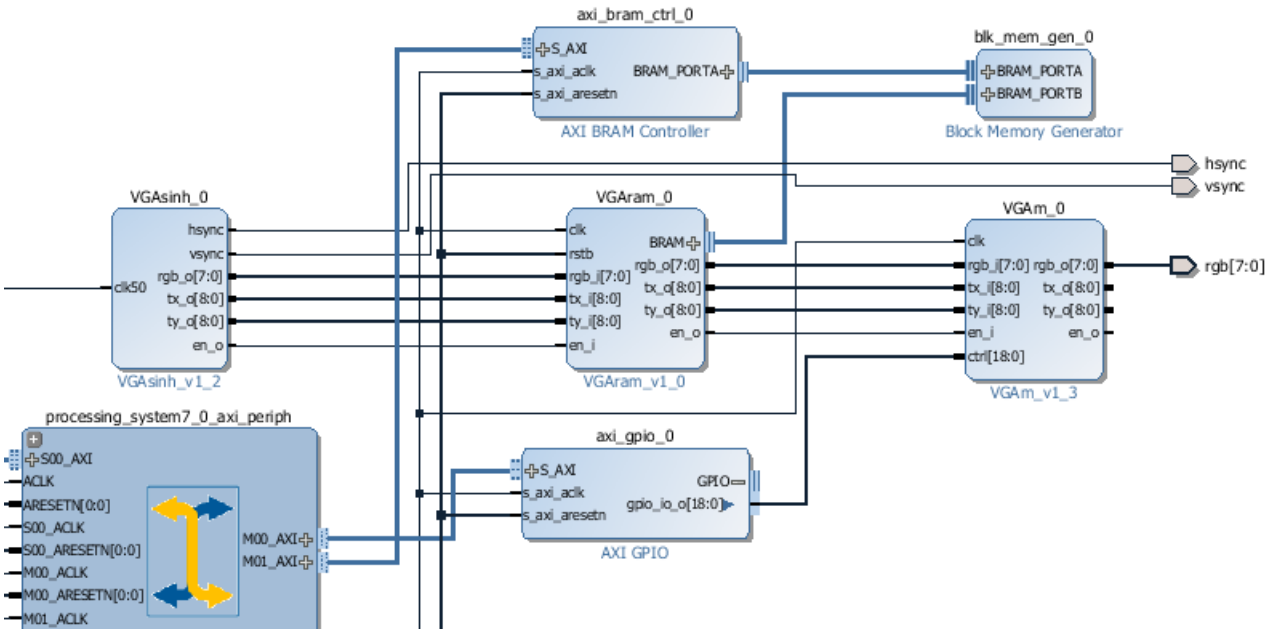
```
BRAM_addrb <= "00000000000000" & ty_i & tx_i;
```

S signaloma ty_i in tx_i torej skrbimo za naslavljanje. Ko poskrbimo za pravilen naslov se nam na 32 bitnem vhodu BRAM_dinb pojavi željeni podatek. Ker je naš rgb izhod samo 8 bitni se potem glede na 2 LSB bita signala tx_i odločamo, kateri del 32 bitne besede BRAM_dinb bomo posalli na izhod rgb_o.

```
process(clk)
begin
    if rising_edge(clk) then
        if en_i = '1' then

            if(tx_i(1 downto 0) = "01") then
                rgb_o <= BRAM_dinb(7 downto 0);
            elsif(tx_i(1 downto 0) = "10") then
                rgb_o <= BRAM_dinb(15 downto 8);
            elsif(tx_i(1 downto 0) = "11") then
                rgb_o <= BRAM_dinb(23 downto 16);
            else
                rgb_o <= BRAM_dinb(31 downto 24);
            end if;
        else
            rgb_o <= rgb_i;
        end if;


    en_o <= en_i;
    tx_o <= tx_i;
    ty_o <= ty_i;

    end if;

end process;
```

2.4 Končni sistem

Ko smo naredili vse 3 IP komponente, ki nam omogočajo prikaz slike preko VGA vmesnika, smo jih povezali v končni sistem, ki vsebuje tudi ARM procesor in vse potrebno periferijo. Sistem je prikazan na sliki 1.



Slika 1: Celoten sistem

3 Aplikacija v jeziku C

Ko smo imeli strojni del aplikacije pripravljeno smo v okolju Eclipse načrtali še aplikacijo v jeziku C, ki je tekla na ARM procesorju.

3.1 Osnovni sistem

Celoten program se izvaja v neskončni while zanki. Ker je procesor zelo hiter sem izvajanje programa razdelil v časovne rezile 100 ms. Vsakih 100 ms sem torej izvedel en cikel programa, nato pa počakal ponovnih 100 ms in izvedel naslednji cikel. Potek igre se upravlja s tipkami, zato sem na začetku vsakega cikla najprej prebral stanje tipk. Če je bila katera koli izmed tipk pritisnjena si je sistem to zapomnil, da se bo pri kasnejšem izvajanju lahko odzval na pritisk. Po branju tipk se je izvajanje programa odvijalo v odvisnosti stanja aplikacije.

Aplikacija je bila lahko v sledečih stanjih:

- INIT - inicializacija igralnega polja. Stanje je aktualno samo na začetku vsake igre.
- PLAY - igranje igre. V tem stanju se skrbi za algoritme ter izris grafike pri igranju igre.
- WIN - zmaga. Po zmagi se program nahaja v tem stanju, ki prikaže zmagovalno sliko in osveži števec zmag ter čaka na pritisk tipke. Po pritisku tipke se zažene nova igra.
- LOOSE - poraz. Po porazu se program nahaja v tem stanju, ki prikaže sliko poraza in čaka na pritisk tipke. Po pritisku tipke se zažene nova igra.

3.2 Princip igre

Izgled igre je prikazan na sliki 2.

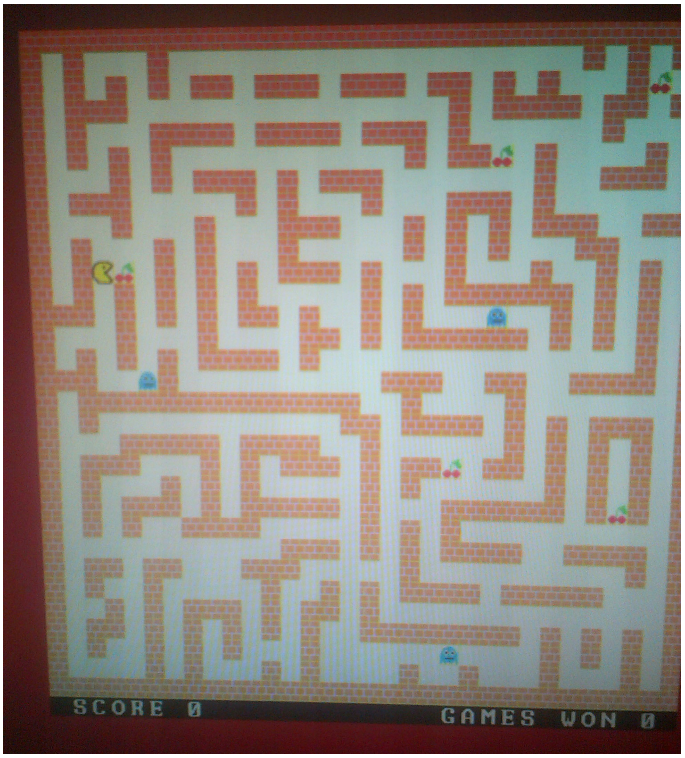
S PacMan-om se pomikamo po labirintu ter poskušamo pojesti vse češnje. Če se zaletimo v modro pošast je igra izgubljena. Postavitev labirinta je v naprej definirana, pošasti in češnje pa se vsako igro generirajo na psevdo naključnih lokacijah. Češnje so celoten potek igre pri miru, pošasti pa se premikajo po labirintu. Algoritem za premikanje pošasti je preprost - ko pošast naleti na oviro se naključno odloči med vsemi možnimi orientacijami in nato nadaljuje premikanje v novi smer, dokler ne naleti na novo oviro. Z vsako pobrano češnjo se nam poveča trenutni seštevek točk (slika 3), ko pojemo vse češnje pa igro dobimo in poveča se nam število zmag (slika 4).

3.3 Opis nekaterih funkcij

Poleg glavnega programa aplikacijo sestavlja še 13 ostalih funkciji. V tem poglavju bom opisal samo nekatere izmed njih.

3.3.1 Funkcija za izris črk

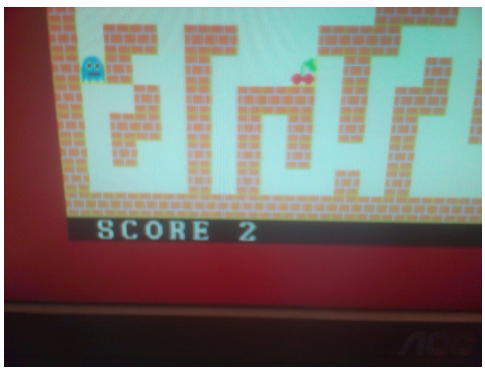
```
void WriteChar(int row, int col, char ch)
{
  int i;
  short *val = (short*)&BigFont[(ch - 65 + 18)*32];
  for (i = 0; i < 16; ++i)
  {
    WriteLine(row, col, i, *val);
    val++;
  }
```



Slika 2: Izgled igre

}

Funkciji podamo v kateri del zaslona (zaslon imam v aplikaciji razdeljen na 32 x 32 kvadratkov) naj izpiše željeni znak ter kateri znak naj se izpiše. Znaki so sestavljeni iz matrike 16 x 16 pixlov. Funkcija vsako vrstico prebere iz tabele znakov (zbirka BigFont) in jo zapiše s pomočjo funkcije WriteLine.



Slika 3: Točke v trenutni igri

Slika 4: Število zmaganih iger

3.3.2 Funkcija za generiranje hrane

```
void GenerateFood(FoodDataType *food)
{
  food->eaten = 0;
  while(!food->eaten)
  {
    food->x = rand()%29 + 2;
    food->y = rand()%29 + 2;
    if(SPACE == world[food->y][food->x])
    {
      world[food->y][food->x] = FOOD;
      food->eaten = 1;
    }
  }
  food->eaten = 0;
  return;
}
```

Funkciji podamo kazalec na podatkovno strukturo hrane. Ta vsebuje koordinate hrane ter zastavico, če jo je glavni junak igre že pojedel (food->eaten). To zastavico uporabim tudi pri generaciji hrane, najprej jo postavim na nič in začnem iskati primerne lokacije za hrano s funkcijo rand(). Če je izbrani kvadratek prazen (SPACE), potem postavim lokacijo hrane vpišem v trenutni labirint in končam funkcijo.

3.3.3 Funkcija za izris celozaslonske slike

```
void WriteFullImage(char *img)
{
  int vrste, stolpci;
  char *p = img;
  for(vrste = 0; vrste < 512; vrste++)
  {
    for(stolpci = 0; stolpci < 512; stolpci++)
    {
      p = (char *) XPAR_AXI_BRAM_CTRL_0_S_AXI_BASEADDR + stolpci + 512 *
          vrste;
      *p = *(img + 512 * vrste + stolpci);
    }
  }
}
```

Funkciji podam kazalec na zbirko točk slike. To zbirko sem generiral s pomočjo dodatnega programa, ki sem ga napisal v programskem jeziku Python. Primer zbirke za manjšo sliko (16x16) je predstavljen v poglavju 3.4. Funkcija se nato sprehodi po vseh točkah na zaslonu in vanje vpiš barvo, ki jo prebere iz zbirke. Primer izrisane celozaslonske slike je na sliki 5.

Slika 5: Celozaslonska slika, ki se izriše ob zmagi

3.4 Zbirka podatkov za izris slike

Spodaj je predstavljena zbirka podatkov za izris slike s 16 x 16 točkami. Vsak element zbirke predstavlja barvni zapis posamezne točke, kjer so barve zakodirane, kot je predstavljeno na sliki 6.

Bit	7	6	5	4	3	2	1	0
Barva	R	R	R	G	G	G	B	B

Slika 6: Kodiranje barv

```
static const char img_pacman[16][16] = {
{0xff, 0xff, 0xff, 0xff, 0x49, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0xff, 0xfd},
{0xff, 0xff, 0x49, 0x00, 0x00, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0x00, 0x48, 0x00, 0xfd},
{0xff, 0x49, 0x00, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0x00},
{0xff, 0x00, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0x48, 0x44, 0xfc, 0xfc, 0xfc, 0xfc, 0x00, 0x6d},
{0x49, 0x00, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0x00, 0x00, 0xfc, 0xfc, 0xfc, 0x25, 0x00, 0xfd},
{0x00, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0x49, 0x00, 0xff, 0xfd},
{0x00, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xf8, 0x49, 0x00, 0xff, 0xff, 0xfd},
{0x00, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0x00, 0x00, 0x00, 0xff, 0xff, 0xff, 0xfd},
{0x00, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0x00, 0x49, 0xff, 0xff, 0xff, 0xff, 0xfd},
{0x00, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0x00, 0x49, 0xff, 0xff, 0xff, 0xfd},
{0x00, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0x90, 0x49, 0xff, 0xff, 0xfd},
{0x00, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0x00, 0x49, 0xff, 0xfd},
{0xdb, 0x00, 0x44, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0x00, 0x49, 0xfd},
{0xff, 0xff, 0x00, 0x44, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0x00, 0x48},
{0xff, 0xff, 0xff, 0x00, 0x44, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0xfc, 0x44, 0x00, 0xfd},
{0xfd, 0xfd, 0xfd, 0xfd, 0x00, 0x48, 0x00, 0x48, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfd, 0xfd}};
```