Univerza v Ljubljani
Fakulteta za elektrotehniko

Dejan Banovec

# PROJEKT VHDL

Poročilo

Izr. prof. dr. Andrej Trost

Ljubljana, 14.1.2018

# KAZALO

# 1 UVOD

Vezje simulira sistem, ki z uporabo CPU, RAM, VGA in vmesnika na zaslonu izriše aplikacijo z določeno logiko. V našem primeru smo naredili igro z 30 tarčami, žogo in ploščkom. Cilj igre je z žogo zadeti vseh trideset tarč. Igra je sestavljena iz štirih stanj. V prvem stanju se žoga še ne premika po y osi. Premikamo jo skupaj z ploščkom po x osi, kar storimo s tipkama key(0) in key(3). Igra gre v drugo stanje, ko pritisnemo tipko key(1). Takrat se začne žoga sama premikati po x in y osi. Ko z žogo zadenemo vseh trideset tarč gre igra v tretje stanje v katerem se elementi ponovno izrišejo na zaslonu na začetnih pozicijah in igra gre nazaj v prvo stanje. Kadar pa žoga štirikrat zgreši plošček, takrat gre igra v četrto stanje v katerem se izvedejo iste stvari kot v tretjem.

# 2 OPIS VEZJA

## 2.1 Vhodni in izhodni signali sistema

### 2.1.1 Clk

Vhodni signal, ki nosi frekvenco delovanja de0_nano. Uporabimo ga v funkcijah, za katere želimo, da se izvajajo po isti frekvenci.

### 2.1.2 Key

Vhodni signal, ki nosi podatke o zunanjih tipkah na razvojni plošči de0_nano. To sta tipki key(0) in key(1)

### 2.1.3 Addr

Izhodni signal, ki je namenjen naslovnemu vodilu na razširitveni plošči. Pošilja podatek o tem kaj v danem trenutku pošiljamo preko data izhoda.

### 2.1.4 Clkout

Izhodni signal, z katerim nastavimo frekvenco delovanja razširitveni plošči. V našem primeru smo jo povezali v vhodno frekvenco CLK.

### 2.1.5 Data

Vhodno izhodni signal, preko katerega pošiljamo in beremo podatke iz razširitvene plošče.

## 2.2 Sistem

V sistem smo pisali logiko in grafično postavitev projekta. Ta bere in pošilja podatke na VGA, proc in vmesnik. VGA se uporablja za definiranje resolucije in postavitev grafike na zaslonu. Preko vmesnika pa dostopa do zunanje plošče.

Postavitev grafike:

Grafične elemente smo definirali na dva načina. Prvi način je bil, da smo sistemu povedali kateri piksli so se uporabili za element. To smo naredili s ploščkom, ki smo ga uporabljali za odboj žoge, in sicer smo to naredili z naslednjo kodo:

```
color <= '1' when x>xp and x<xp+64 and y>580 and y<595 else  '0';
```

S signalom color smo povedali kdaj naj se piksel na zaslonu pobarva. Color se je postavil na 1, le kadar je bil znotraj določenih koordinat x in y. Signal xp smo uporabili, zato da smo z njim spreminjali x pozicijo ploščka.

Drugi način risanja elementov na zaslon je bil, da smo iz knjižnice sprites poklicali želen element, ki smo ga predhodno v knjižnico narisali kot array ničel in enic. Enica pove kateri piksel se bo pobarval. Eden izmet takšnih elementov je bilo srce:

```
 type srce10x16 is array (0 to 9) of unsigned(15 downto 0);
constant srce: srce10x16:= (
"0011100000111100",
"0111111011111111",
"1111111111111111",
"1111111111111111",
"0111111111111100",
"0011111111111000",
"0001111111110000",
"0000111111100000",
"0000011111000000",
"0000001110000000");
```

Nato smo v sistemu naredili štiri nove signale, ki smo jih uporabili za postavitev srca. Signala xsrce in ysrce sta podajala začetno x in y pozicijo srca, signal colorsrce1 smo uporabili za branje x osi iz arraya, srce1 pa za branje y osi.

```
signal ysrce: unsigned (11 downto 0):=to_unsigned(10,12);
signal xsrce: unsigned (11 downto 0):=to_unsigned(10,12);
```

*signal colorsrce1: std_logic:='0';*
*signal srce1: unsigned (15 downto 0);*

*srce1 <= srce(to_integer(y-ysrce)) when y>=ysrce and y<ysrce+10 else (others=>'0');*
*colorsrce1 <= srce1(to_integer(x-xsrce)) when x>=xsrce and x<xsrce+16 else '0';*

Za določanje barve smo uporabili naslednjo kodo:
*if x<800 and y<600 and colorsrce1='1' then rgb <="110000"; end if;*

Kadar se je x in y pozicija nahajala znotraj resolucije 800x600 in je bila na colorsrce1 postavljena na 1 smo na rgb signal poslali podatek za rdečo barvo.

Barve rgb signala:

- Rdeča: rgb <="110000";
- Zelena: rgb <="001100";
- Modra: rgb <="000011";
- Bela: rgb <="111111";
- Črna: rgb <="000000";

Za premik žoge gor in dol smo uporabili kodo:
*if smerz = '0' and en ='1' then yz <= yz+1; elsif smerz = '1' and en = '1' and yz > 5 then yz <= yz-1; end if;*

S signalom smerz smo povedali, kako se žoga premika, in sicer ali se premika gor ali dol. Signal smerz smo vedno spremenili ob trkih v tarče, zgornjo steno ali v plošček.

Pozicijo x ploščka smo spreminjali na naslednji način:
*keys(0)='1' and en = '1' then xp <= xp-1; elsif keys(3)='1' and en = '1' then xp <= xp+1; end if;*

Ploščku smo tudi nastavili koordinate do kje se lahko premika:
*if xp >= 746 then xp <= xp-1; elsif xp <= 0 then xp <= xp+1; end if;*

za zgrešitev ploščka smo uporabili naslednjo kodo:

*if yz = 600 then interrupt<='1'; trksrc<=1+trksrc; xz<=to_unsigned(13, 12); yz<=to_unsigned(550, 12);*

*xp <= to_unsigned(5, 12); stanje<="01"; end if;*

Kadar je žoga prišla do pozicije yz = 600 se je stanje postavilo na 01, poziciji ploščka in žoge sta se postavili v prvotno vrednost v spodnji levi kot. Interrupt se je pa postavil na 1, kar smo uporabili v programu, da smo ga postavili na začetno pozicijo. Nato smo interrupt postavili na 0 v stanju 01.

Da smo tarčo ob trku žoge zbrisali, smo najprej definirali signal trk, ki se je postavil na 0 kadar, sta bili koordinati žoge in ploščka enaki. Izbris tarče smo naredili na sledeč način:

*if colorp1 = '1' and colorz = '1' and trk = '0' then trk <= '1'; smerz<='0'; trkst<=trkst+1; end if;*

Signal trkst se je povečal za 1 ob vsakem trku žoge s tarčo in ko se je trideset krat povečal, se je stanje postavilo na 11, kar je pomenilo zmago. Nato se vsi elementi postavijo na začetne pozicije na zaslonu.

## 2.3 VGA

V vga smo definirali resolucijo, ki smo jo potem uporabljali za prikaz na zaslonu. V našem primeru je bila resolucija 800x600.

Resolucijo smo uredili z naslednjo kodo:

```
if rising_edge(clk) then
  if hst < "010000001111" then                    -- hst < 1039
    hst <= hst + 1;                               -- hst++
  else
    hst <= (others=>'0');                         -- hst = 0
                  if vst < "001010011001" then    -- vst < 665
    vst <= vst + 1;                               -- vst++
                  else
```

```
vst <= (others=>'0');                    -- vst = 0
end if;
```

Hsync in vsync smo uporabili, kadar smo bili izven resolucije:

```
hsync <= '1' when hst>=856 and hst<856+120 else '0';     -- hsync=1 kadar je hst med 856 in 976
vsync <= '1' when vst>=637 and vst<637+6 else '0';       -- vsync=1 kadat je vst med 637 in 643
```

## 2.4 Vmesnik

Vmesnik služi povezavi med razširitveno ploščo in sistemom. V našem primeru pošiljamo na vmesnik podatke ki se bodo izrisali na zaslonu in led matriki, z njega pa beremo tipke na razširitveni plošči. Vse to nastavljamo z addr signalom. Beremo in pišemo podatke na naslovno ploščo preko vhodno izhodnega signala data. Izhod key vmesnik pošiljamo na vhod num. Ta potem podatke preko number_counter, message_addr, addr_room, data_room in data_led pošlje podatke na data. Izhod clkout služi za frekvenco delovanja izhodne plošče. Vhod cpu_pc uporabimo za debug, ki ga dobimo iz cpu-ja.

Kaj se bo pošiljalo na data izhod in bralo iz njega, naredimo z naslednjo kodo:

```
if rising_edge(clk) then
  hs1 <= hsync;
        hs2 <= '0';
        if hs1='1' and hsync='0' then
          addr <= "01"; data <= "ZZZZZZZZ";      -- address = 01, data = ZZZZZZZZ
          hs2 <= '1';
        elsif hs2='1' then
          addr <= "10"; data<=data_led;          -- address = 10, data = data_led (podatki za ledice)
          key <= unsigned(data(3 downto 0));     -- key
        else
          addr <= "00";
          data(7) <= hsync;
```

*data(6) <= vsync;*

*data(5 downto 0) <= std_logic_vector(rgb(1 downto 0) & rgb(3 downto 2) & rgb(5 downto 4));*

*end if;*

## 2.5 Proc

Proc povezuje program, cpu in sistem med seboj. Iz cpu dobiva podatke preko dat_o signala, preko dat_i pa daje podatke na cpu.

Podatke, ki smo jih želeli poslati iz sistema na cpu, smo jih naložili na dat_i. Tako smo na primer naredili za začetno pozicijo žoge. X pozicijo žoge smo dobili na vhod xzoge, ki smo jo nato postavili na dat_i, vendar samo kadar smo dobili ukaz inp 2, kar pomeni da se je adr_o postavil na x"02":

*dat_i <= xzoge when adr_o=x"02"else pin;*

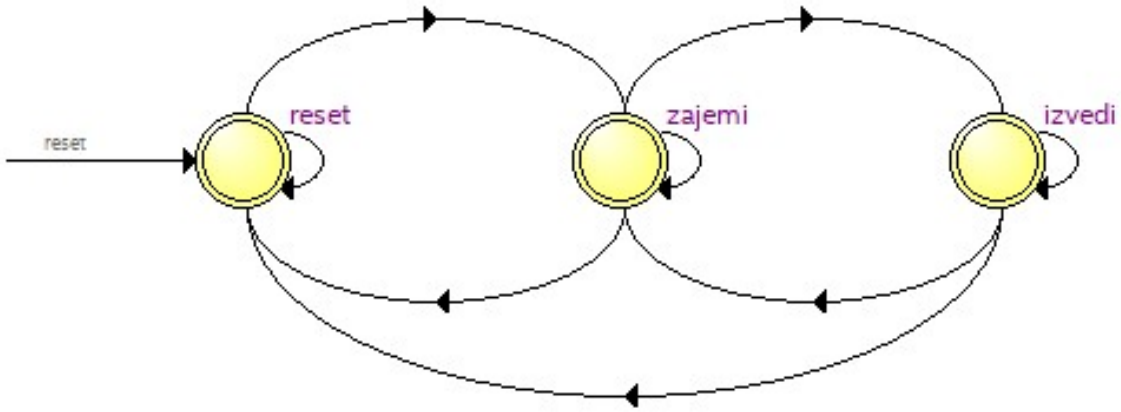
Podatke, ki jih je pa cpu pošiljal nazaj na proc, smo brali iz izhoda dat_o na cpu. Na primer za branje naslednje pozicije žoge smo uporabili kodo:

*if ce = '1' and we_o = '1' and adr_o = 1 then pout1 <= dat_o; end if;*

Dat_o smo postavili na pout1, ki je šel na sistem, samo kadar je bil ce na 1, we_o na 1 in adr_o na 1, kar pomeni da je bil izveden ukaz outp 1. Signala ce in we_o smo uporabljali samo za uro, ki pove kdaj se bo koda izvedla.

### 2.5.1 Cpu

CPU ali centralna procesna enota izvaja ukaze, ki jih dobi iz program-a. CPU vsebuje tri stanja, to so reset, zajemi in izvedi.




V stanju reset postavi akum na 0 in adr-reg na 0 in gre na stanje zajemi. V stanju zajemi, zajame podatke, ki mu jih pošilja program iz izhoda q na vhod data. Ko dobi podatke iz programa se postavi v stanje izvedi. V tem stanju glede na dobljene podatke izvaja kodo.

Stanje reset:

```
if st = reset then
        akum <= x"000";
        adr_reg <= x"00";
        st <=zajemi;
```

stanje zajemi:

```
if st = zajemi then
        pc <= adr_reg + 1;
        busy<='0';
        if (data(11 downto 8)=jmp) or
          (data (11 downto 8)=jze  and akum= 0) or
          (data(11 downto 8)=jcs and carry = '1') then
              st <= zajemi;
        elsif (data(11 downto 8)=call) then -- skok na podprogram
              st <= zajemi;
              pc1 <= adr_reg + 1; -- shrani naslednji pc(tj. adr+1) in akumulator
              akum1 <= akum;
```

```
else
        st <=izvedi;
        busy<='1';
end if;
if data(11 downto 8)=outp then
        dat_o <= akum;
end if;
if data(11 downto 8)=inp or data(11 downto 8)=outp then
        adr_o <= data (7 downto 0);
end if;
inst_code <= data(11 downto 8);
```

stanje izvedi:

```
st <=zajemi;
case inst_code is
 when lda =>
        akum <= data;
        carry <= '0';
 when inp =>
        akum <= dat_i;
        carry <= '0';
 when add =>
        rez := ('0' & akum) + ('0' & data);
        akum <= rez(11 downto 0);
        carry <= rez(12);
 when sbt =>
        rez := ('0' & akum) - ('0' & data);
        akum <= rez(11 downto 0);
        carry <= rez(12);
 when nota =>
        akum <= not akum;
```

```
            carry <= '0';
        when anda =>
            akum <= akum and data;
            carry <= '0';
        when ora =>
            akum <= akum or data;
            carry <= '0';
        when shl =>
            akum <= akum sll 1;
            carry <= akum(11);
        when shr =>
            akum <= akum srl 1;
            carry <= akum(0);
        when ret =>
            akum <= akum1;
        when outp =>
            we_o <= '1';
```

konstante lda, sta, jmp, add, abt, nota, inp, jze, jcs, outp, anda, ora, call, ret, shl in shr kličemo iz knjižnice procpak.
Konstante v procpak so zapisane z naslednjo kodo:

```
-- osnovni nabor operacij
 constant lda: koda := "0001"; -- nalozi iz pomnilnika v akum
 constant sta: koda := "0010"; -- shrani iz akum v pomnilnik
 constant jmp: koda := "0100"; -- skok na novo pomnilnisko lokacijo
 constant add: koda := "1000"; -- pristej k akum vrednost iz pomnilnika
 constant sbt: koda := "1001"; -- odstej vrednost iz pomnilnika

 -- razsiritve
 constant nota: koda := "0000"; -- logicna negacija
```

```
constant inp: koda := "0011"; -- nalozi iz inport v akum
constant jze: koda := "0101";  -- skok, ce je akum=0
constant jcs: koda := "0110";  -- skok, ce je carry=1
constant outp: koda := "0111"; -- shrani iz akum v outport
constant anda: koda := "1100"; -- logicna IN
constant ora:  koda := "1101"; -- logicna ALI
constant call: koda := "1010"; -- skok na podprogram, shrani PC in akum
constant ret: koda := "1011";  -- vrni se, ponastavi PC in akum
constant shl: koda := "1110"; -- pomik akum za eno v levo (mnozenje z 2)
constant shr: koda := "1111"; -- pomik akum za eno v desno (deljenje z 2)
```

nastavljanje adr-ja(naslovnega signala):

```
-- kombinacijsko določi naslov
adr_next <=  x"00" when st=reset else                              -- reset
             pc1 when st=izvedi and inst_code=ret else   -- dodatek za return
             pc when st=izvedi else                      -- naslednji ukaz
             data(7 downto 0);                           -- parameter
adr <= adr_next; -- naslovni signal
```

### 2.5.2 Program

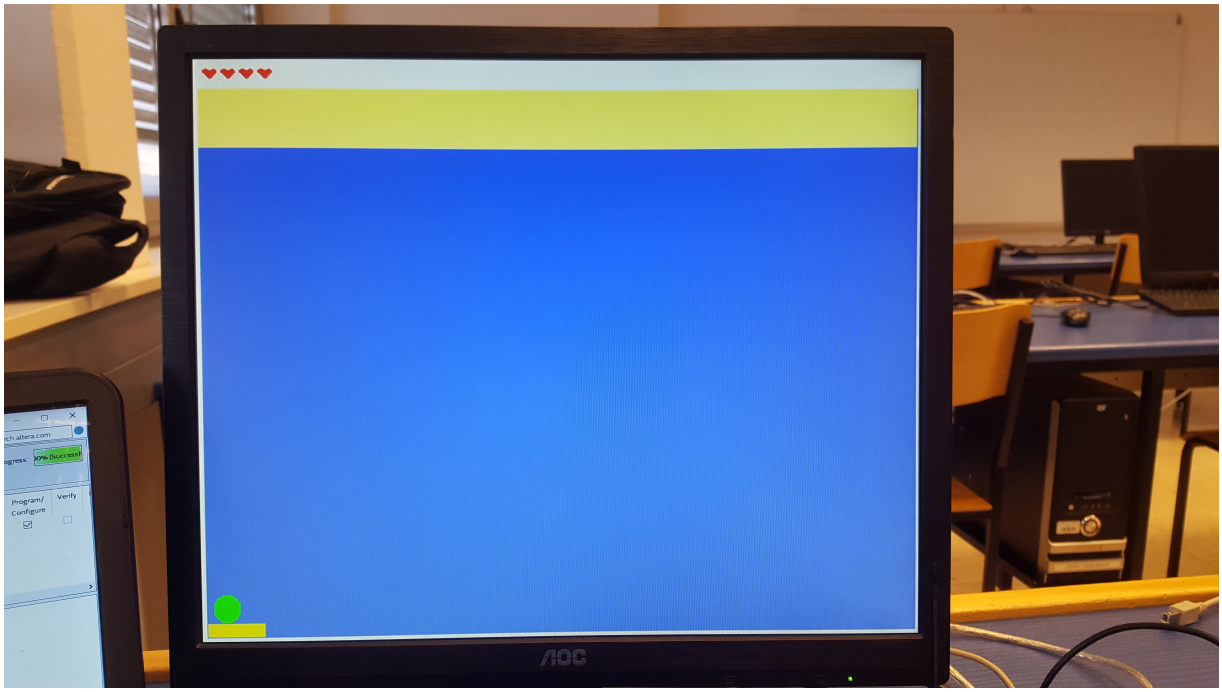
Predstavlja RAM, ki vsebuje ukaze v zbirnem jeziku, ki jih zapišemo v program.mif in vrednosti, ki jih shranimo na določeno lokacijo. Preko izhoda q na vhod data cpu-ja pošilja podatke o tem, kateri ukaz se bo izvedel in vrednost, ki je shranjena na določeni lokaciji v ram-u. Ko cpu podatek glede na ukaz predela, ga pošlje preko izhoda dout na vhod data v program-u, kateri ga na določeno lokacijo shrani. Preko vhoda address dobivamo iz cpu-ja naslovne signale.

Ukazi zbirnega jezika so:

- NOTA - negiraj AKUMULATOR
- LDA n - naloži RAM(n) v AKUMULATOR
- STA n - shrani iz AKUMULATORJA v RAM(n)
- INP n - naloži dat_i(n) v AKUMULATOR

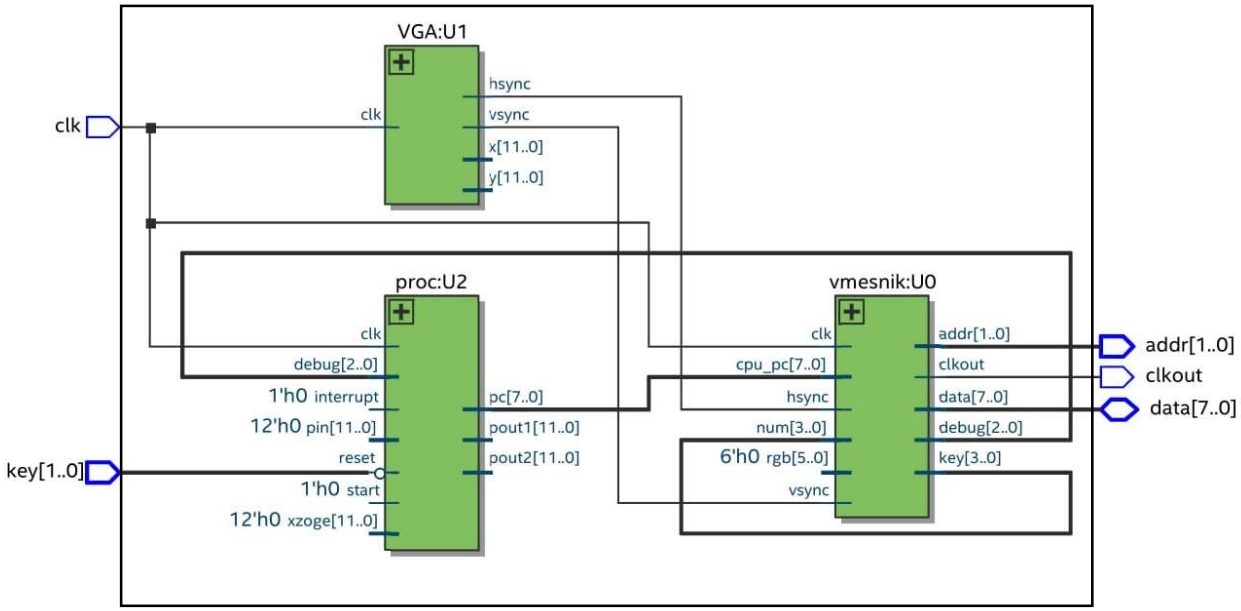
- JMP n - skok na naslov n
- JZE n - skoči na n če je AKUM= 0
- JCS n - skoči na n če je carry = 1
- OUTP n - shrani iz AKUMULATORJA na dat_o(n)
- ADD n - prištej RAM(n) k AKUMULATORJU
- SBT n - odštej RAM(n) od AKUMULATORJA
- CALL n - skoči na n in shrani PC in AKUM
- RET - skoči nazaj, uporabi shranjen PC in AKUM
- ANDA n - AKUM = AKUM AND RAM(n)
- ORA n - AKUM = AKUM OR RAM(n)
- SHL - pomakni AKUMULATOR v levo (množenje z 2)
- SHR - pomakni AKUMULATOR v desno (deljenje z 2)

# 3 SLIKA IGRE

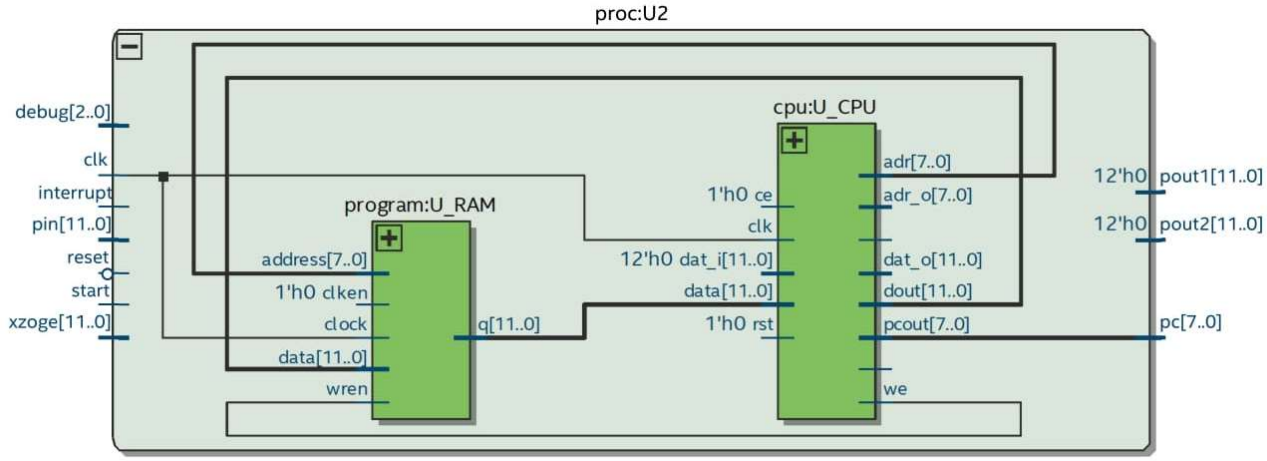


# 4 BLOK SHEMA

## 4.1 Sistem.vhd




## 4.2 Proc.vhd



# 5 PROGRAM.MIF

| ZBIRNI JEZIK: | | | IZHODI: |
|---|---|---|---|
| | lda 20; | hitrost delovanja | 00 : 128; |
| | outp 0; | piši ob adr_o=0 | 01 : 700; |
| start: | inp tipka; | beri ob adr_o=1 | 02 : 301; |
| | sta t; | shrani na t | 03 : 227; |
| | outp 3 | reset int, start | 04 : 703; |
| | inp xzoge; | beri ob adr_o=2 | 05 : 302; |
| | sta a; | shrani na a | 06 : 224; |
| | lda s; | odpri s | 07 : 126; |
| | sbt t; | s-t | 08 : 927; |
| | jze desno; | s=0 -> desno | 09 : 50b; |
| | jmp start; | skoč na start | 0a : 402; |
| desno: | lda a; | naloži xz | 0b : 124; |
| | add 1; | xz+1 | 0c : 829; |
| | sta a; | shrani novi xz na a | 0d : 224; |
| | outp xz; | izpisi xz | 0e : 701; |
| | sbt 765; | a-765 | 0f : 92a; |
| | jze levo; | a=0 -> levo | 10 : 517; |
| | inp int; | beri trk | 11 : 300; |
| | sta int; | shrani pod trk | 12 : 223; |
| | lda s; | odpri s | 13 : 126; |
| | sbt int; | s-trk | 14 : 923; |
| | jze start; | a=0 -> start | 15 : 502; |
| | jmp desno; | skoci na desno | 16 : 40b; |
| levo: | lda a; | naloži xz | 17 : 124; |
| | sbt 1; | a-1 | 18 : 929; |
| | sta a; | shrani novi a | 19 : 224; |
| | outp xz; | izpisi xz | 1a : 701; |
| | sbt 5; | a-5 | 1b : 92b; |
| | jze desno; | a=0 -> desno | 1c : 50b; |
| | inp int; | beri trk | 1d : 300; |
| | sta int; | shrani v trk | 1e : 223; |
| | lda s; | odpri s | 1f : 126; |
| | sbt int; | s-trk | 20 : 923; |
| | jze start; | int=0 -> start | 21 : 502; |
| | jmp levo; | skoci na levo | 22 : 417; |
| int | db 0 | | 23 : 000; |
| a | db 0 | | 24 : 000; |
| n | db 0 | | 25 : 000; |
| s | db 1 | | 26 : 001; |
| t | db 0 | | 27 : 000; |
| xz | do 1 | | 28 : 014; |
| int | di 0 | | 29 : 001; |
| tipka | di 1 | | 2a : 2fd; |
| xzoge | di 2 | | 2b : 005; |

# 6 ZASEDENOST VEZJA

Total logic elements: 1,443 / 22,320 ( 6 % )

Total registers: 545