Načrtovanje digitalnih sistemov

Digitalni elektronski sistemi 2024/25

Andrej Trost

UNIVERZA V LJUBLJANI
FE
Fakulteta za elektrotehniko

Načrtovanje digitalnih sistemov

- delovanje digitalnega sistema postopoma razdelimo na podrobneje določene naloge: od zgoraj navzdol (top-down)
- vezje razdelimo na bloke (komponente), ki so vnaprej pripravljene ali pa jih načrtamo, ko imamo določene njihove naloge
- (pod)bloke modeliramo in preizkušamo s simulacijo

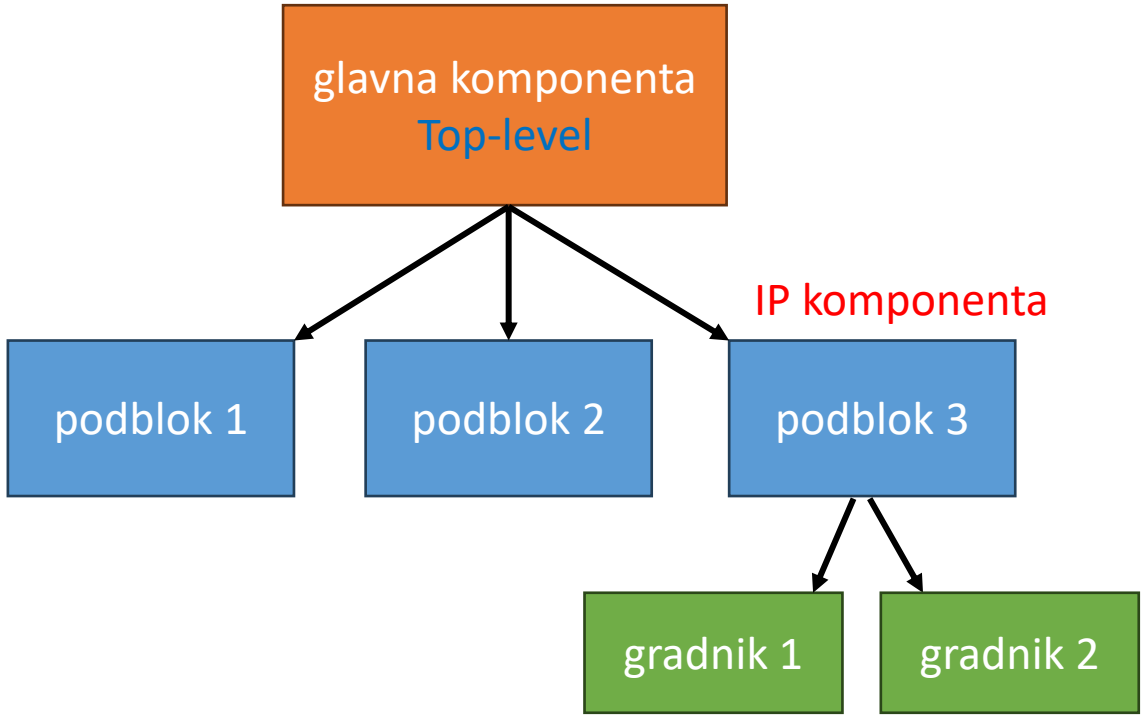
- top-down je uporaben postopek za realne sisteme
  - zahteva znanje in izkušnje
- bottom-up: iz gradnikov sestavljamo vezje ali sistem
  - običajno v procesu učenja

Načrtovanje od zgoraj navzdol

POTEK

1. specifikacija
   sistem
2. arhitektura
   algoritem
3. implementacija
   RTL, logika

RAZDELITEV



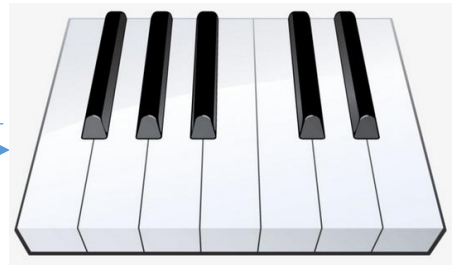
METODE

model sistema
(programski, matematični)

registrske operacije, FSM
podatkovna pot in krmilnik
strojno-opisna koda (HDL)
sintetizirana vezja (HLS, AI)

logika, števci, registri

Primer: digitalni sintetizator



vmesnik

generator

generator

mešalnik

+

X

PWM

delilnik

en

Vezja s števci

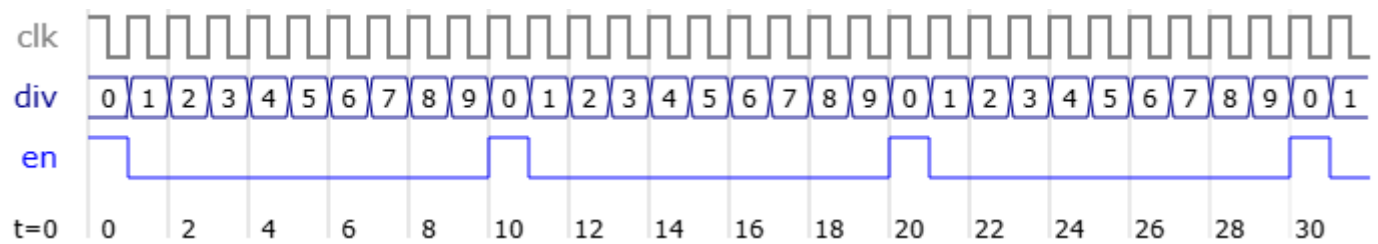
- števec po modulu je lahko delilnik frekvence

Primer: števec po modulu 250 šteje od 0 do 249,
ob prelivu števca negiramo signal na izhodu flip-flopa,
ki ima 500x nižjo frekvenco (50 MHz -> 100 kHz)

```
if div=249 then
  div <= 0
  clk1 <= not clk1
else
  div <= div + 1
end
```

- pri sinhronih vezjih (ena sama ura) uporabimo števec za omogočanje dogodkov z nižjo frekvenco

```
if div=9 then
  div <= 0
else
  div <= div + 1
end
en = 1 when div=0 else 0
```



Načrtovanje VHDL z umetno inteligenco (AI)

- podamo čim bolj natančno zahtevo

> **Design VHDL model of a modulo 10 counter with clock input and overflow output.**

Specifications

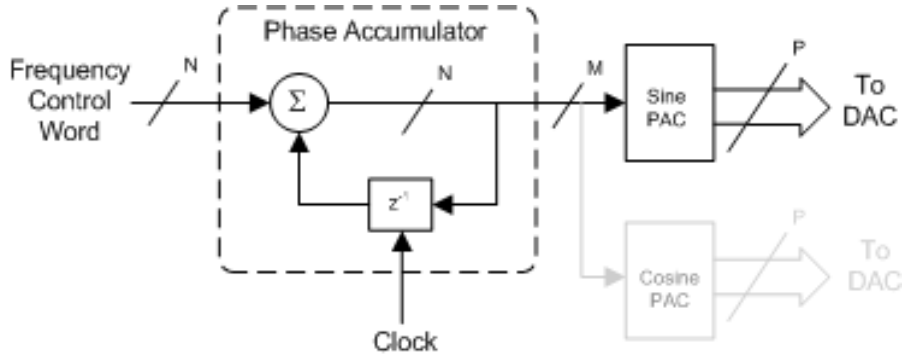
- Clock input: `clk`
- Synchronous counting
- Counts 0 to 9 → wraps to 0
- Overflow output: `overflow` goes high **only** when counter wraps from 9 to 0
- No reset

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Modulo10_Counter is
    Port (
        clk      : in  STD_LOGIC;
        count    : out STD_LOGIC_VECTOR(3 downto 0);
        overflow : out STD_LOGIC
    );
end Modulo10_Counter;
```

```vhdl
architecture Behavioral of Modulo10_Counter is
    signal cnt       : unsigned(3 downto 0) := (others => '0');
    signal oflow_reg : STD_LOGIC := '0';
begin

    process(clk)
    begin
        if rising_edge(clk) then
            if cnt = 9 then
                cnt       <= (others => '0');
                oflow_reg <= '1';
            else
                cnt       <= cnt + 1;
                oflow_reg <= '0';
            end if;
        end if;
    end process;

    count    <= std_logic_vector(cnt);
    overflow <= oflow_reg;

end Behavioral;
```

Numerično krmiljen oscilator

- NCO: N-bitni števec, ki se povečuje za FCW
- frekvenco določa perioda obračanja števca, uporabimo najvišji(e) bit(e) števca

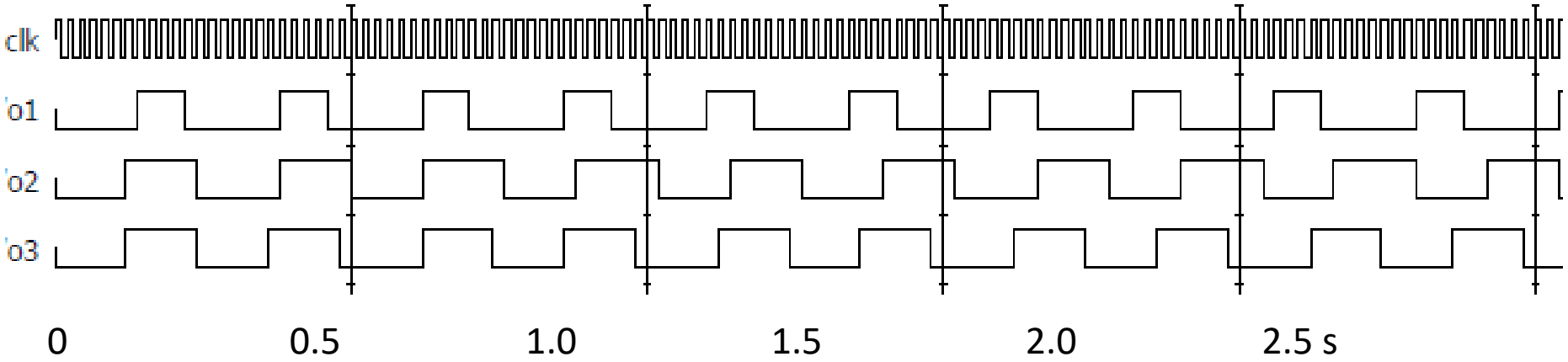


https://en.wikipedia.org/wiki/Numerically_controlled_oscillator

Primerjava delilnika in NCO

Vhod: 50 Hz, izhod: 4 Hz (razmerje 12.5)

- delilnik: 12 => 4.16 Hz ali 13 => 3.84 Hz
- NCO: 6-bit, FCW=5 => 3.906 Hz
  10-bit, FCW=82 => 4.004 Hz



Načrtovanje NCO z AI

> **Design in VHDL an NCO with 8-bit sine LUT, 32 samples, and 24-bit phase counter.**

How It Works:

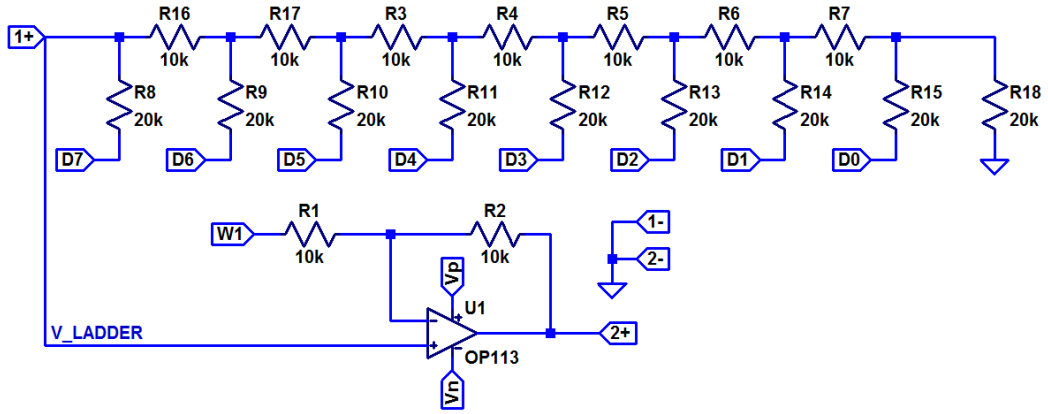
1. A **24-bit phase accumulator** adds a **phase increment** every clock cycle.
2. The **upper 5 bits** of the accumulator select the **address** in a 32-entry **sine lookup table (LUT)**.
3. The LUT returns the corresponding **8-bit sine value**.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity NCO is
    Port (
        clk         : in  STD_LOGIC;
        reset       : in  STD_LOGIC;
        phase_inc   : in  STD_LOGIC_VECTOR(23 downto 0); -- Tuning word
        sine_out    : out STD_LOGIC_VECTOR(7 downto 0)
    );
end NCO;
```

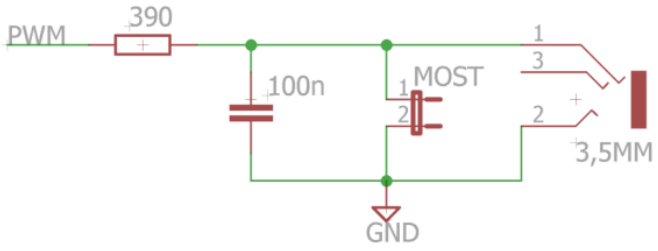
```vhdl
architecture Behavioral of NCO is
    signal phase_acc : unsigned(23 downto 0) := (others => '0');
    signal addr      : integer range 0 to 31 := 0;

    type sine_lut_type is array (0 to 31) of STD_LOGIC_VECTOR(7 downto 0);
    constant sine_lut : sine_lut_type := (
        -- 8-bit unsigned sine values (0 to 255), centered at 128
        -- Values scaled from sin(2πn/32) * 127 + 128
        x"80", x"98", x"AF", x"C3", x"D3", x"DF", x"E7", x"EA",
        x"E7", x"DF", x"D3", x"C3", x"AF", x"98", x"80", x"67",
        x"50", x"3C", x"2C", x"20", x"18", x"15", x"18", x"20",
        x"2C", x"3C", x"50", x"67", x"80", x"98", x"AF", x"C3"
    );
begin

    process(clk, reset)
    begin
        if reset = '1' then
            phase_acc <= (others => '0');
        elsif rising_edge(clk) then
            phase_acc <= phase_acc + unsigned(phase_inc);
        end if;
    end process;

    -- Use upper 5 bits to index 32-sample LUT
    addr <= to_integer(phase_acc(23 downto 19));
    sine_out <= sine_lut(addr);

end Behavioral;
```

Digitalno-analogna pretvorba

- NCO izhod pošljemo na paralelni D/A pretvornik, npr:



- ker deluje FPGA na dosti višjih frekvencah, kot je frekvenca zvoka, uporabimo 1-bitni pretvornik
  - pulzno-širinsko moduliran signal (PWM) pošljemo skozi RC sito na zvočnik



Načrtovanje PWM z AI

> **Design a VHDL PWM for the NCO with an 8-bit signed data and without reset input.**

Key Requirements

- Input: `sine_in` – 8-bit **signed** (range: -128 to +127, `std_logic_vector`)
- Output: `pwm_out` – 1-bit PWM signal
- No reset input
- Uses an internal 8-bit unsigned counter (0–255) as PWM ramp
- Maps signed sine input to unsigned threshold for PWM comparison

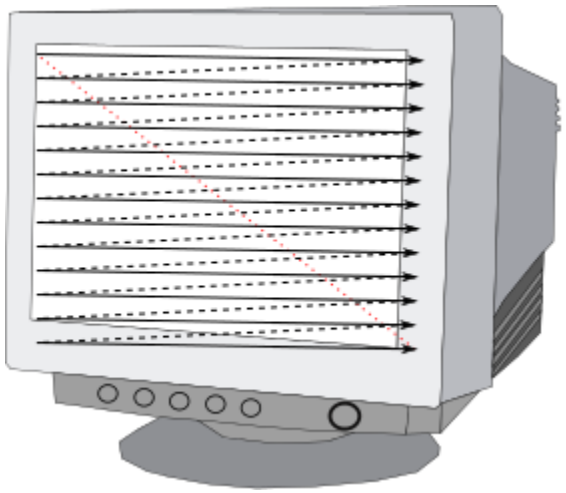
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity PWM_Signed is
    Port (
        clk      : in  STD_LOGIC;
        sine_in  : in  STD_LOGIC_VECTOR(7 downto 0); -- signed input (-128 to +127)
        pwm_out  : out STD_LOGIC
    );
end PWM_Signed;

architecture Behavioral of PWM_Signed is
    signal counter   : unsigned(7 downto 0) := (others => '0');
    signal sine_val  : signed(7 downto 0);
    signal threshold : unsigned(7 downto 0);
begin

    -- Cast input to signed
    sine_val <= signed(sine_in);

    -- Map signed range (-128 to +127) to unsigned (0 to 255)
    threshold <= unsigned(sine_val + 128);

    process(clk)
    begin
        if rising_edge(clk) then
            counter <= counter + 1;
        end if;
    end process;

    -- Generate PWM signal
    pwm_out <= '1' when counter < threshold else '0';

end Behavioral;
```

Izvedba PWM v projektu sint25

- uporabimo števec za deljenje ure
- namesto +128 naredimo **xor** 128

```
-- deljenje sistemske ure 50 MHz na 100 kHz (clk1)
process(clk)
begin
 if rising_edge(clk) then
  if div = 249 then
    div <= (others=>'0');
    clk1 <= not clk1;

    -- premakni zvok v pozitivne vrednosti
    zvokn <= unsigned(zvok xor "10000000");
  else
    div <= div + 1;
  end if;
 end if;
end process;

-- generiraj PWM izhod
pwm <= '1' when div < zvokn else '0';
```

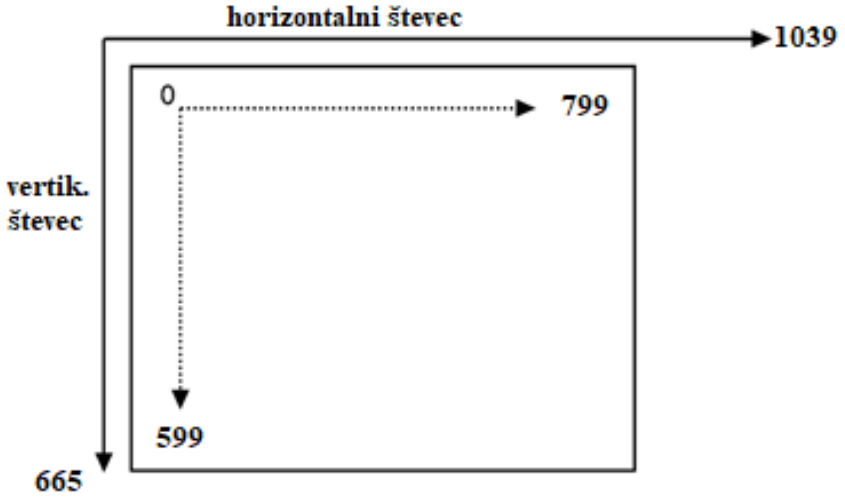
Prikazovanje slike na monitorju VGA



skeniranje vrstice (horizontala)

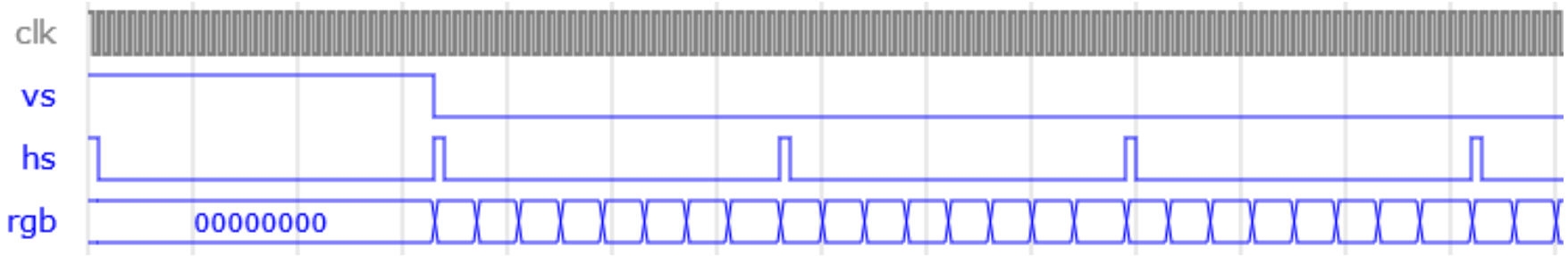
horizontalno vračanje žarka (zatemnitev)

vertikalno vračanje žarka (zatemnitev)



SVGA 800 x 600 vidnih točk @72 Hz

- standard določa časovni potek sinhronizacije: hs, vs
- slika: $7 \cdot 10^5$ ciklov (14 ms)



VGA krmilnik

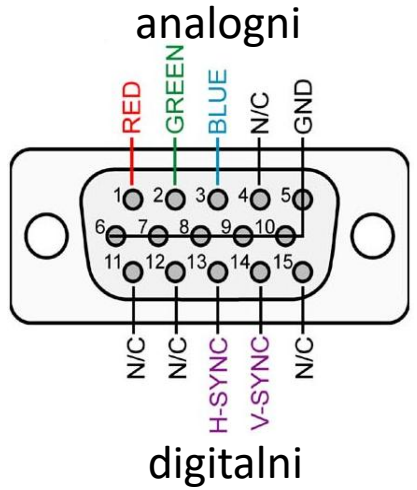
- z dvema števcema določamo koordinati trenutne točke in generiramo sinhronizacijska impulza
- časovni potek določa standard, npr. http://tinyvga.com/vga-timing/800x600@72Hz

```
entity vga
 x,y: out u11;  -- koordinati
 hs,vs: out u1; -- sinhronizacija
begin
 if x<1039 then
   x<=x+1
 else
   x<=0
   if y<665 then
     y<=y+1
   else
     y<=0
   end
 end
 hs = 1 when x>=856 and x<976 else 0
 vs = 1 when y>=637 and y<643 else 0
end
```

ura (MHz)	horizontala (period)	start hsync	trajanje hsync	veritikala (period)	start vsync	trajanje vsync
50	1040	856	120	666	637	6

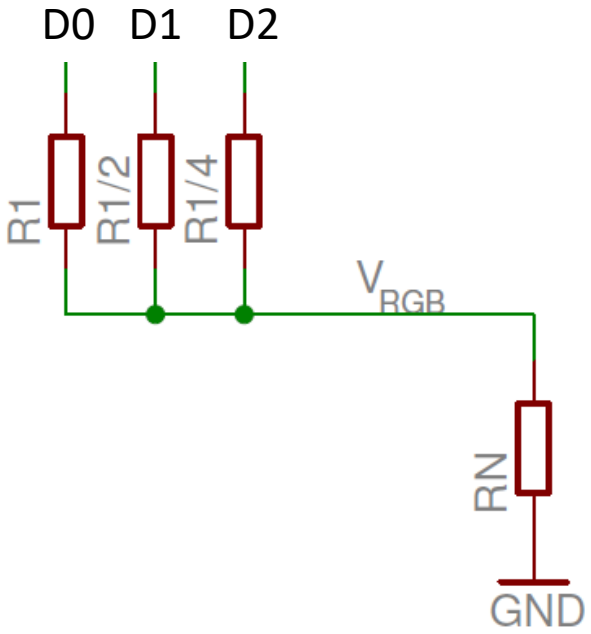
Prikaz slike iz na monitorju

- VGA priključek ima 3 analogne signale za barvo
  - na vezju so trije 2-bitni uporovni D/A pretvorniki
- dodamo logiko, ki nastavlja barvo točk (**rgb**) na določenih koordinatah v vidnem delu slike
  - kadar smo izven vidnega dela, pošljemo 0



RN = 75 Ω Vcc=3,3 V

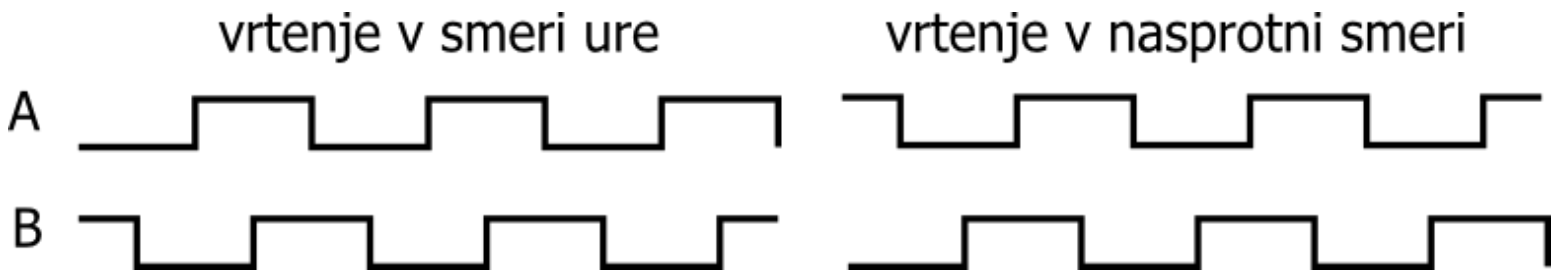
V_{RGB} = 0,7 V



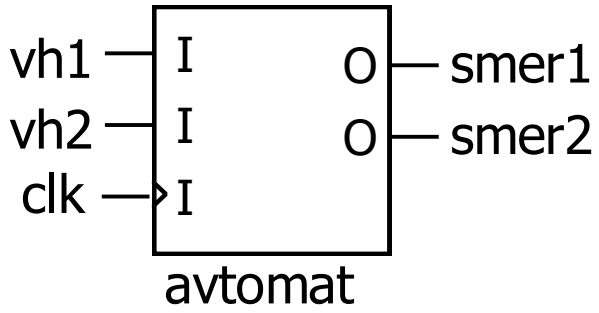

```
rgb <= "000011" when (x>150 and x<550) and (y>200 and y<300) else
       "010101" when (x<800 and y<600) else
       "000000";
```

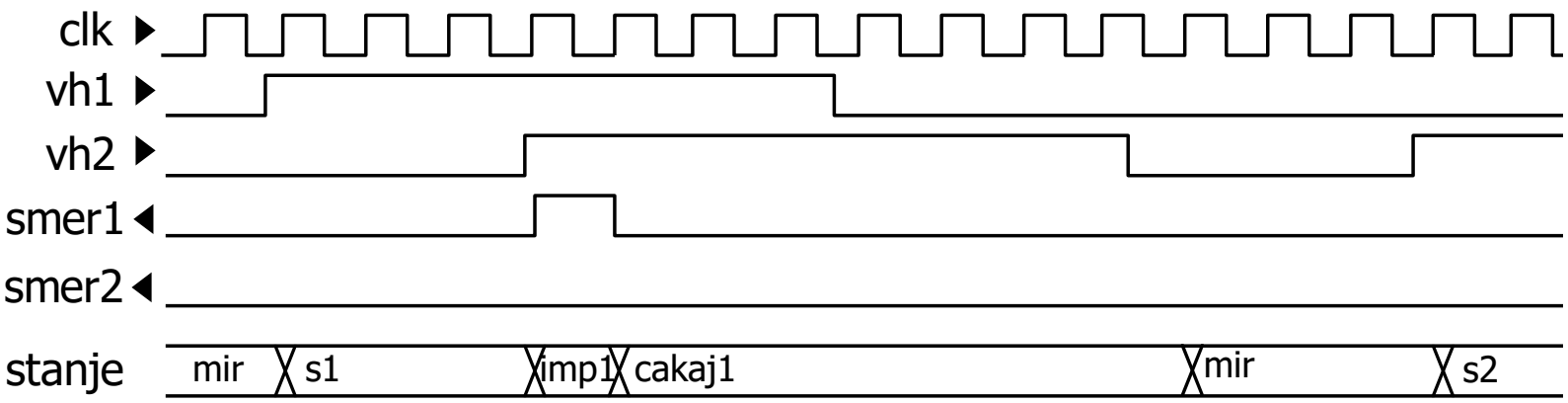
Rotacijski inkrementalni kodirnik

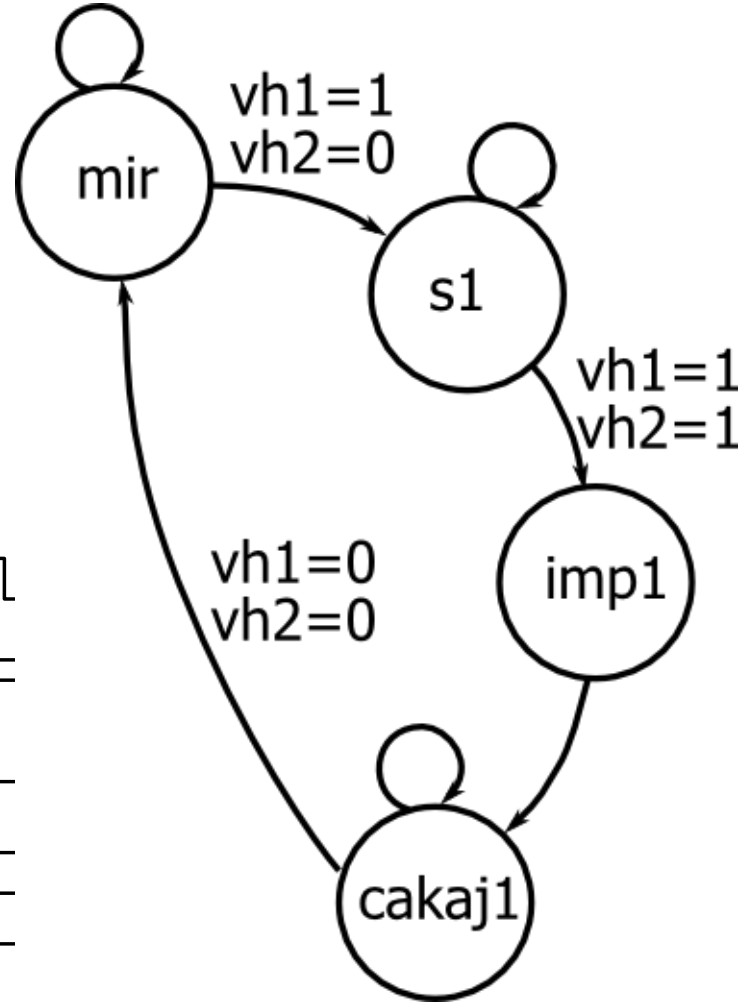
- kodirnik ima dva izhoda, ki ob vrtenju generirata impulze



Sekvenčno vezje za detekcijo vrtenja

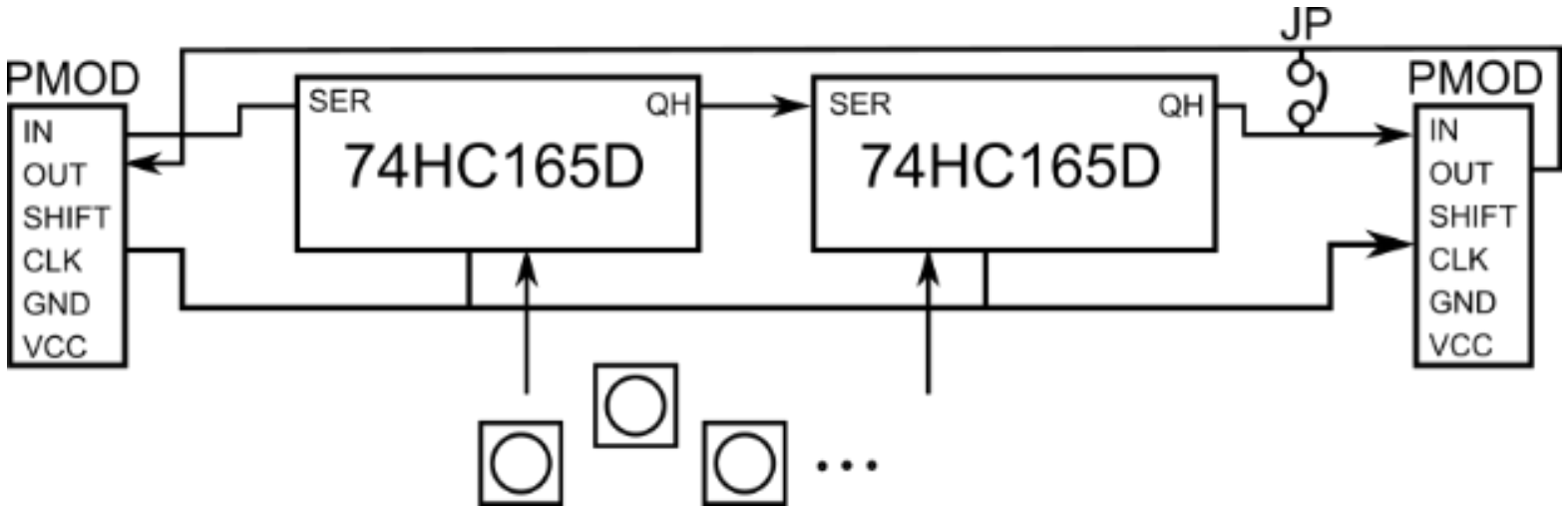






Prototipna klaviatura

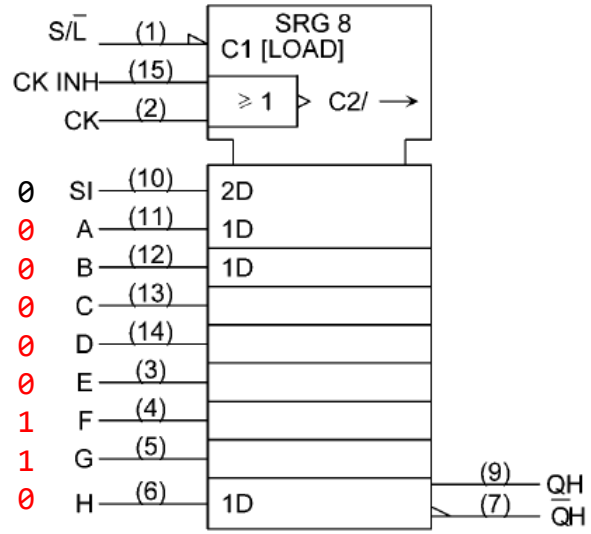
- 12 tipk povezanih na dva 8-bitna pomikalna registra

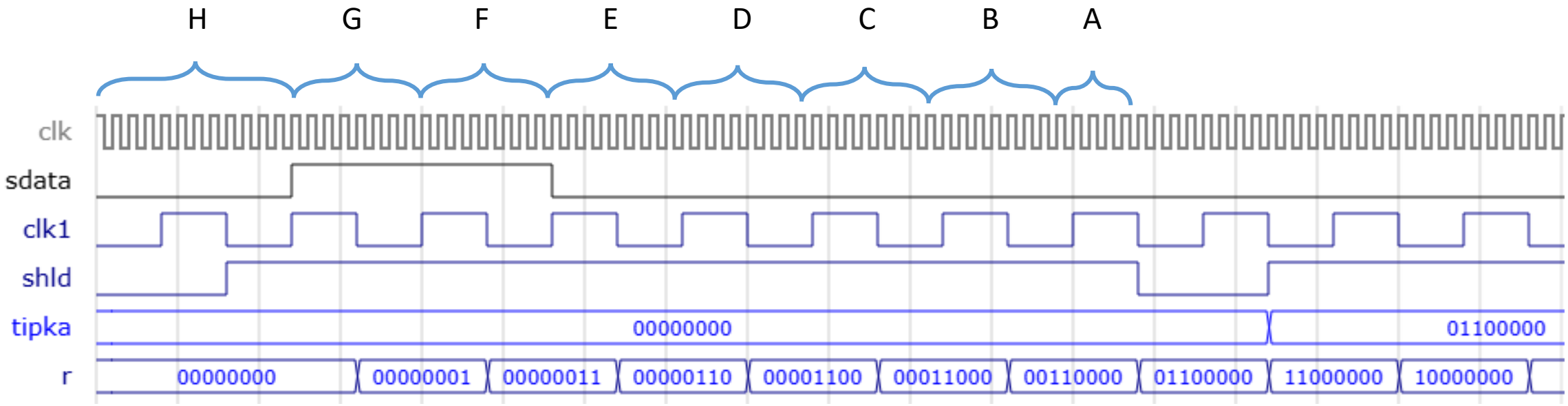


- Kako narediti krmilnik za branje pomikalnega registra?

Načrtovanje krmilnika

- ob **shld=0** dobimo MSB (**sdata** = H)
- **shld=1** in 7x **clk1**, pomikanje bitov G-A na **sdata**
- ob padajoči fronti **clk1** pomikamo register **r**





Krmilnik v SHDL

- kako narediti krmilnik za branje dveh 74165?

```
entity vmes
 sdata: in u1;
 clk1,shld: out u1;
 tipka: out u8;
 del:  u2;
 nbit:  u3;
 r:  u8;
begin
 del<=del+1  -- deljenje ure s 4
 if del=3 then
   clk1 <= not clk1 -- ura za 74165
   if clk1=1 then   -- stej bite in pomikaj
     nbit <= nbit + 1
     r <= r(6 downto 0) & sdata
     if shld=0 then -- prepis na izhod
       tipka <= r
     end
   end
 end
 shld = 0 when nbit=0 else 1
end
```

Krmilnik z AI

> Design VHDL model to control shift register 74165 and read serial data.

74LS165 Pin Summary (Relevant Pins):

- PL (Parallel Load): Active LOW, loads data from input pins.
- CP (Clock): Shifts bits on rising edge.
- CE (Clock Enable): Active LOW.
- QH: Serial output (connected to FPGA/microcontroller).
- D0-D7: Parallel data inputs.

VHDL Overview:

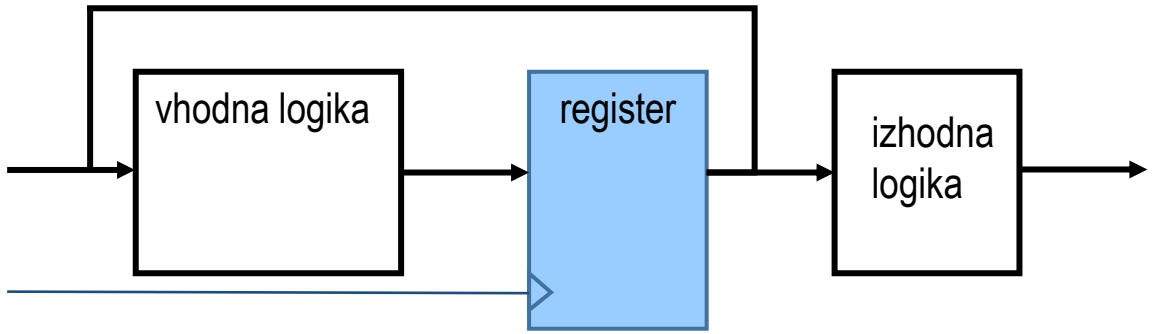
The VHDL code will:

1. Load data using the `PL` signal.
2. Pulse the `CP` signal 8 times (or however many bits you want to read).
3. Sample the `QH` pin on each clock cycle.

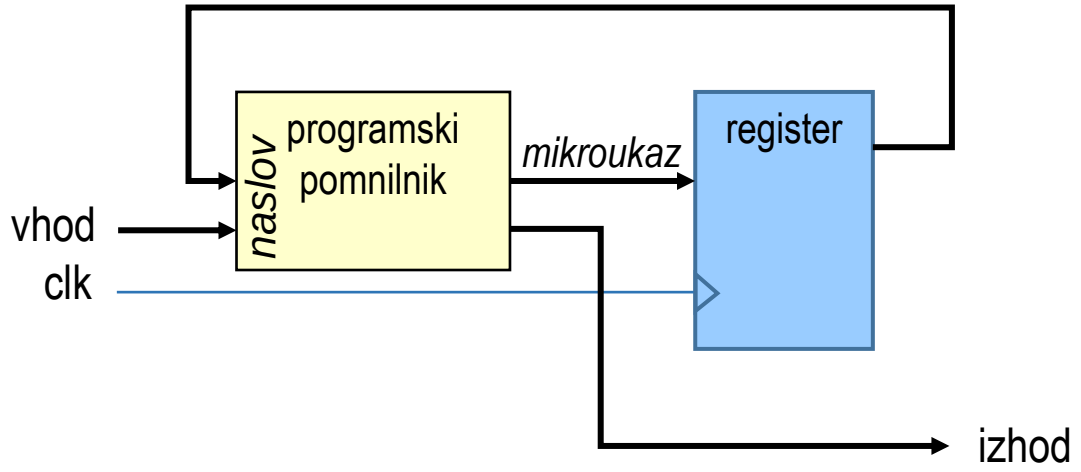
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity shift_74165_ctrl is
    Port (
        clk       : in  STD_LOGIC;
        start     : in  STD_LOGIC;
        QH_in     : in  STD_LOGIC;
        PL        : out STD_LOGIC;
        CP        : out STD_LOGIC;
        CE        : out STD_LOGIC;
        data_out  : out STD_LOGIC_VECTOR(7 downto 0);
        done      : out STD_LOGIC
    );
end shift_74165_ctrl;

architecture Behavioral of shift_74165_ctrl is
    type state_type is (IDLE, LOAD, SHIFT, DONE);
    signal state       : state_type := IDLE;
    signal bit_count   : integer range 0 to 8 := 0;
    signal shift_reg   : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
    signal cp_pulse    : STD_LOGIC := '0';
begin
```

```vhdl
            when LOAD =>
                PL    <= '0';  -- load parallel data
                CE    <= '1';  -- keep clock disabled
                CP    <= '0';
                bit_count <= 0;
                state <= SHIFT;

            when SHIFT =>
                PL    <= '1';  -- PL inactive
                CE    <= '0';  -- enable clock
                -- create one pulse on CP per cycle
                if cp_pulse = '0' then
                    CP        <= '1';
                    cp_pulse  <= '1';
                else
                    CP        <= '0';
                    cp_pulse  <= '0';
                    shift_reg <= shift_reg(6 downto 0) & QH_in;
                    bit_count <= bit_count + 1;
                    if bit_count = 7 then
                        state <= DONE;
                    end if;
                end if;

            when DONE =>
                CE       <= '1'; -- disable clock
                CP       <= '0';
                data_out <= shift_reg;
```

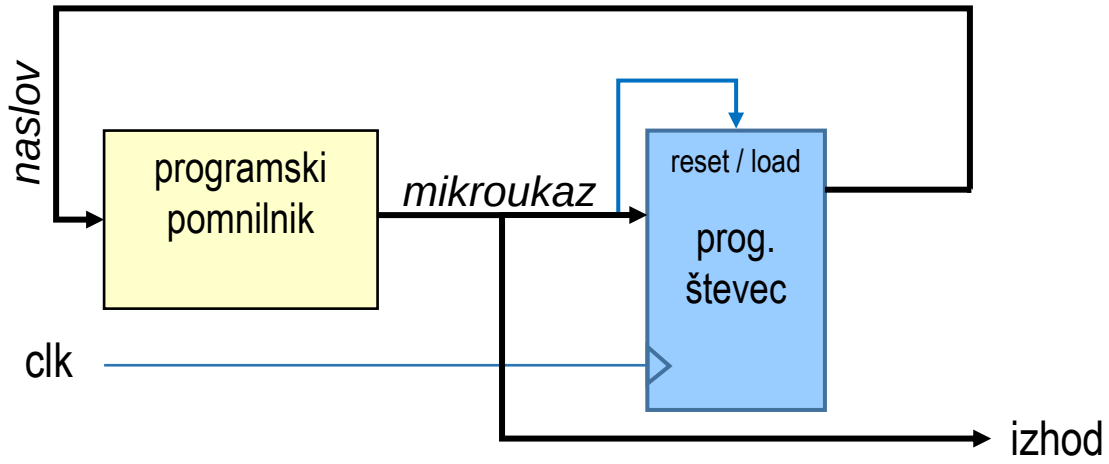
- stroj končnih stanj (FSM)



- (mikro)programiran stroj



Mikrosekvenčnik



Npr. izhod: 100, 110, 001, 001, 010

- najvišji bit mikroukaza je reset

naslov	mikroukaz
000	0 100
001	0 110
010	0 001
011	0 001
100	1 010
101	x xxx
111	x xxx

Programski števec določa zaporedje

- reset za skok na začetek
- load za skok na poljubno mesto (vejitev)

Generator zaporedja z mikrosekvenčnikom

- Npr. ob start oddaj Morsejevo kodo — ...

```
entity morseB
 start: in u1;
 izh: out u1;
 en: u1
 rom: 16u2=1,1,1,0,1,0,1,0,1,2;
 data: u2;
 n: u4
begin
 data=rom(n) -- branje iz ROM
 izh=data(0) -- LSB je izhod

 if start=1 then -- start in stop
   en<=1; n<=0
 elsif data(1) then
   en<=0
 end
 if en then  -- naslov pomnilnika
   n<=n+1
 end
end
```