VHDL kviz: števec

- Kakšno je stanje števca, ko je reset='1' ?

```
s1: process (clk)
begin
 if rising_edge(clk) then
    stev <= stev + 1;
    if reset='1' then
      stev <= "0000";
    end if;
 end if;
end process;
```

```
s2: process (clk)
begin
 if rising_edge(clk) then
    if reset='1' then
      stev <= "0000";
    end if;
    stev <= stev + 1;
 end if;
end process;
```

a) "0000"
b) "0001"
c) stev+1

Celoštevilski podatkovni tip (integer)

- Tip integer: 32-bitna cela števila
  - pri deklaraciji lahko omejimo obseg (range)
- Uporaba: števci, konstante, indeksi vektorjev

```
architecture one of test is
  signal n: integer := 0;
  signal s: integer range 0 to 255;
begin

s1: process (clk)
begin
 if rising_edge(clk) then
    n <= n + 1;   -- 32 bitni števec
    s <= s + 1;   -- 8 bitni števec
```

Izbira podatkovnih tipov

- Zunanji signali realnega modela so vektorji
- Notranji sig. so lahko vektorji ali pa cela števila
- Za pretvorbo uporabimo ustrezne funkcije:

```
use IEEE.std_logic_arith.all;  -- conv_std_logic_vector()
use IEEE.std_logic_unsigned.all;  -- conv_integer()

architecture one of test is
  signal i, n: integer;
  signal a, b: std_logic_vector(7 downto 0);
begin
  i <= conv_integer(a);                  -- vektor v integer
  b <= conv_std_logic_vector(n, 8); -- integer v vektor
```

Pravila glede podatkovnih tipov

- V prireditvenih stavkih velja:
  tip in velikost signala = tip in velikost izraza

```
signal x, y: std_logic;
signal a: std_logic_vector(32 downto 0);
signal b,c: std_logic_vector(7 downto 0);
```

Pravilno	Napačno

x <= '0';
y <= x;
a<="00001111000011110000111100001111";
a <= (others =>'0'); -- vse bite na '0'
b <= X"A5"; -- šestnajstiška vrednost
c <= b;
	x <= "0"; -- x ni vektorski signal
	y <= a; -- y ni 32 bitni vektor
	a <= '0'; -- a ni enobitni signal
	a <= 0; -- a ni celo število
	b <= "0"; -- napačno število bitov
	c <= a; -- napačno število bitov

Obnašanje operatorjev

- Operatorji so dodatno definirani v knjižnicah!
  - angl. operator overloading
- Knjižnica IEEE in paketi:
  - std_logic_arith oz. numeric_std
  - std_logic_signed oz. std_logic_unsigned
- Primer:
  - kombiniranje std_logic_vector in integer

```
if stev < 9 then          -- rezultat odvisen od knjižnice!
    stev <= stev + 1;  -- lahko seštevamo vektor in celo število
```

Podatkovna struktura zbirka

- Z zbirko (array) modeliramo LUT, ROM, RAM
- Definiramo nov podatkovni tip (type), ki ga uporabimo pri deklaraciji signalov

```
type beseda is array (15 downto 0) of std_logic;
type tabela1 is array (0 to 1023) of std_logic_vector(15 downto 0);
type tabela2 is array (0 to 1023) of beseda;

signal b: beseda;
signal t1: tabela1;
signal t2: tabela2;
```

Tip std_logic_vector je narejen kot zbirka std_logic!

Opis ROM pomnilnika

- Definiramo zbirko in deklariramo signal
  - ob deklaraciji nastavimo začetne vrednosti signala

```
type tabela is array (0 to 3) of std_logic_vector(7 downto 0);
constant rom : tabela := (x"08", x"09", x"0A", x"00");
```

- Uporabimo indeks za posamezni element
  - če je indeks tipa std_logic_vector, ga pretvorimo v integer

```
podatek <= rom(conv_integer(naslov));
```

Opis RAM pomnilnika

- RAM deklariramo in beremo tako kot ROM
- Za pisanje v RAM zapišemo proces:

```
P: process(adr, data, wr)
begin
  if wr='0' then
    ram(conv_integer(adr)) <= data;
  end if;
end process;
```

model asinhronega SRAM

- Sinhroni RAM opišemo s sinhronim procesom