



Laboratorij za načrtovanje integriranih vezij

Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*



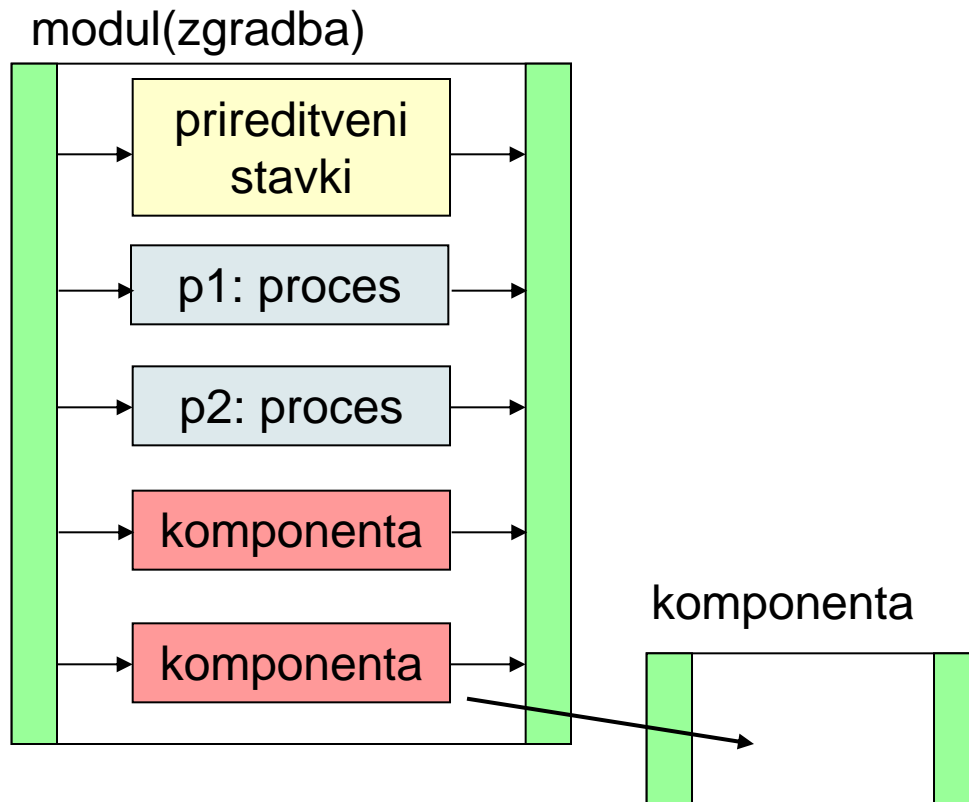
Digitalni Elektronski Sistemi

Osnove jezika VHDL

signali, izrazi in osnovni stavki

VHDL opisuje:

- ▶ operacije nad digitalnimi signali
- ▶ registre in avtomate
- ▶ komponente vezja



VHDL

Very high-speed IC

Hardware

Description

Language

```
q <= n;
```

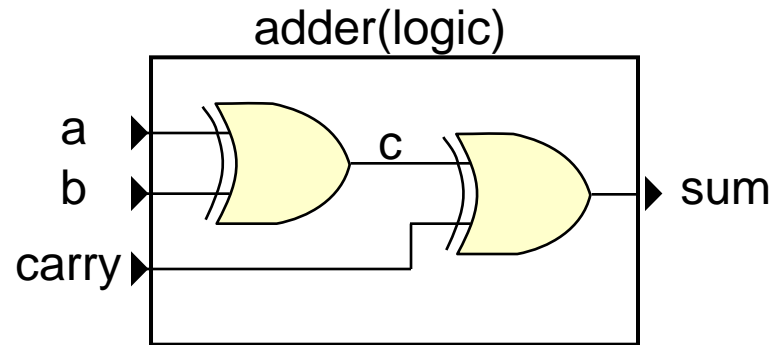
```
p1: process(clk)
begin
  if rising_edge(clk) then
    n <= n + 1;
  end if;
end process;
```

Funkcijski opis vezja v jeziku VHDL

- ▶ stavki opisujejo gradnike vezja
 - ▶ vrstni red stavkov ni pomemben (sočasni stavki)

```
entity adder is  
  port ( a, b : in std_logic;  
         carry : in std_logic;  
         sum : out std_logic);  
end adder;
```

```
architecture logic of adder is  
  signal c : std_logic;  
begin  
  sum <= c xor carry;  
  c <= a xor b;  
end one;
```

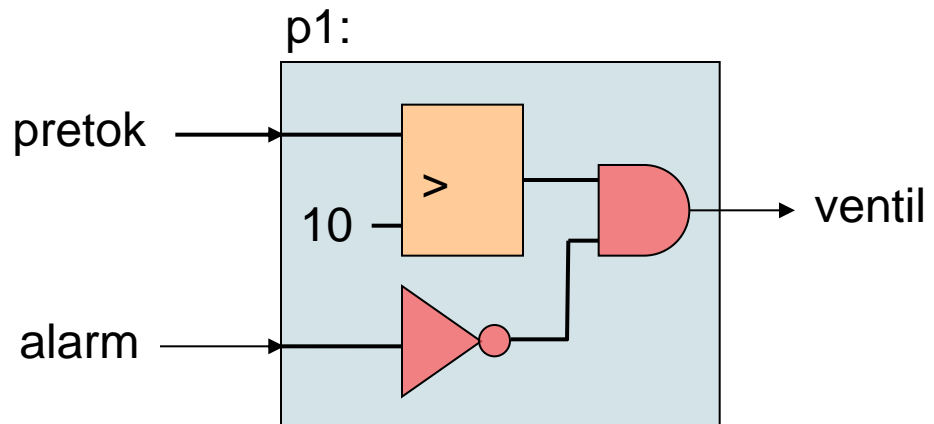


deklaracija notranjega signala

Postopkovni opis vezja v jeziku VHDL

- ▶ v procesu opišemo delovanje vezja
 - ▶ zgradbo vezja določi program za sintezo vezij
 - ▶ vrstni red sorodnih prireditev je pomemben

```
arhitektura
p1: process
ventil <= '0';
if pretok > 10 then
    ventil <= '1';
end if;
if alarm = '1' then
    ventil <= '0';
end if;
end process;
```



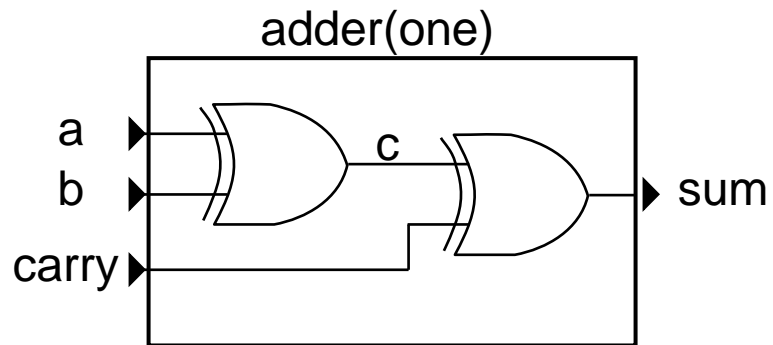
Osnovni koncepti jezika VHDL

- ▶ Model vezja je sestavljen iz:
 - ▶ opisa vmesnika (**entity**), kjer določimo priključke in parametre
 - ▶ opisa delovanja (**architecture**)
- ▶ Osnovni elementi so **signali**, ki predstavljajo povezave
- ▶ Pri opisu vmesnika določimo zunanje signale (**port**) in parametre vezja
 - ▶ signalu določimo smer (**in, out, inout, buffer**)
 - ▶ in podatkovni tip (npr. **bit**)

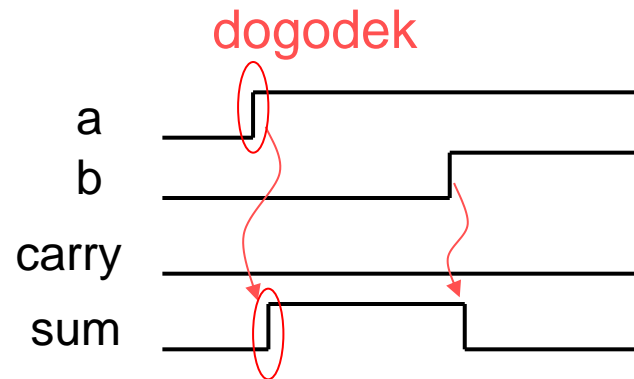
```
entity adder is  
  port ( a, b : in bit;  
         carry : in bit;  
         sum : out bit );  
end adder;
```

Obravnava VHDL modelov vezij

```
architecture logic of adder is  
  signal c : std_logic;  
begin  
  sum <= c xor carry;  
  c <= a xor b;  
end one;
```

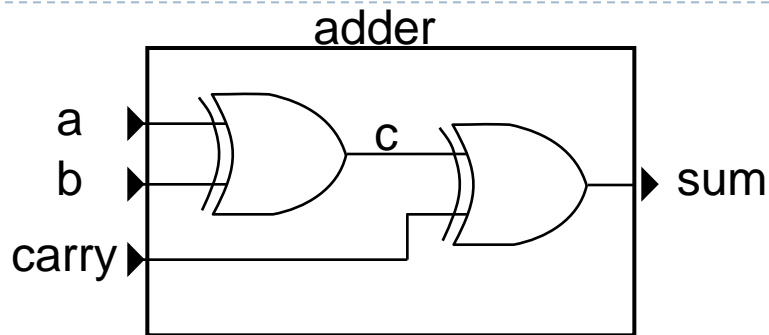


Sintetizirano vezje

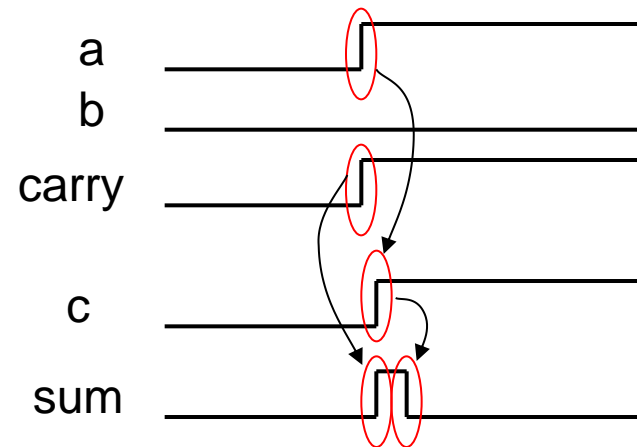


Graf simulacije (waveform)

Potek simulacije



dogodek



seznam dogodkov:

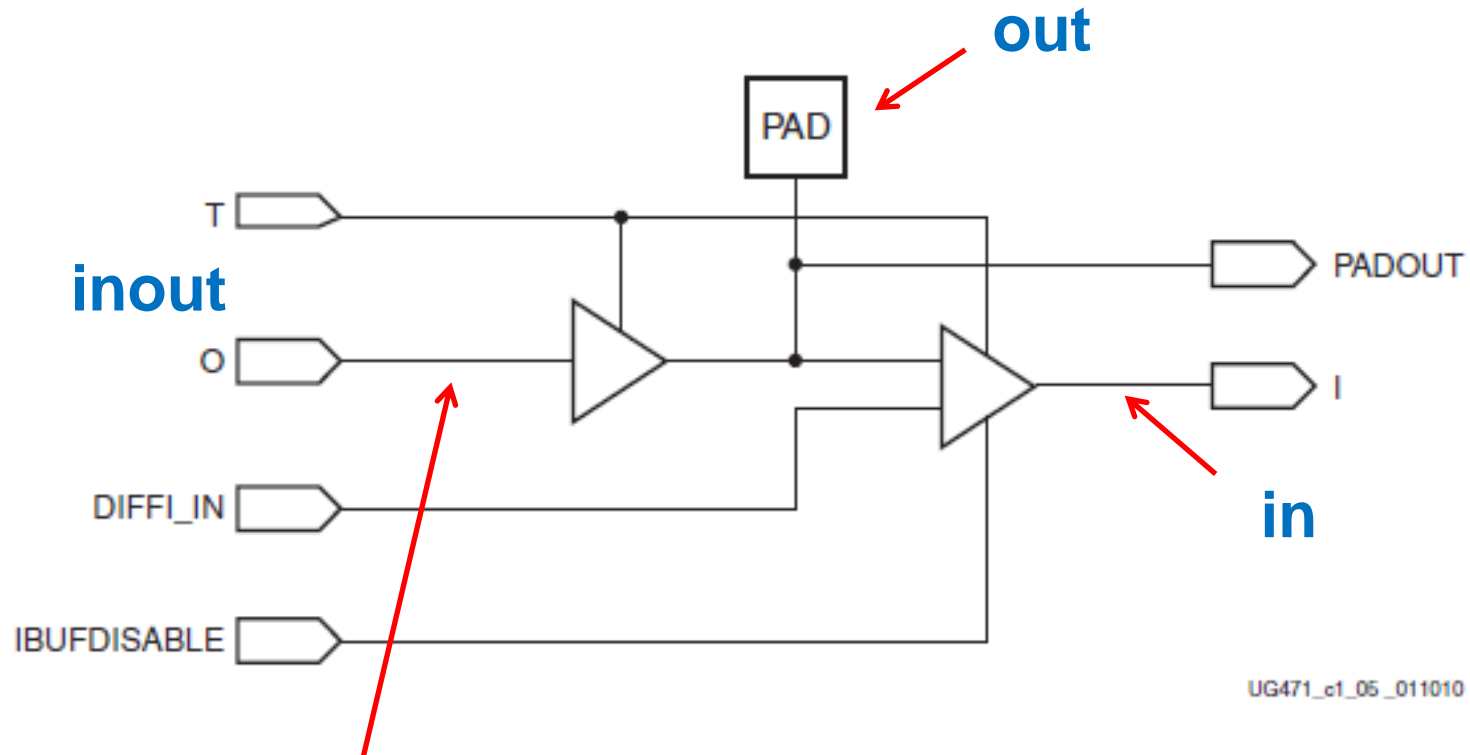
a: 0, 1 (T) carry: 0, 1 (T)

korak	čas	a	b	carry	sum	c
seznam	0	0, 1(T)	0	0, 1(T)	0	0
izvrši	T	1	0	1	0	0
izračunaj	T	1	0	1	1(T+Δ)	1(T+Δ)
izvrši	T+Δ	1	0	1	1	1
izračunaj	T+Δ	1	0	1	0(T+2Δ)	1
izvrši	T+2Δ	1	0	1	0	1

Deklaracija zunanjih signalov

```
entity vezje is  
  port ( ime : način tip; ...
```

▶ način: in, out, inout, buffer



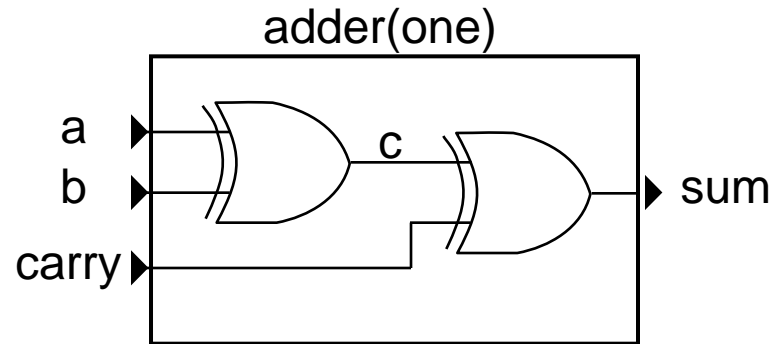
UG471_c1_05_011010

buffer (notranji in izhodni signal)

Opis delovanja (architecture)

```
entity adder is  
  port ( a, b : in bit;  
         carry : in bit;  
         sum : out bit );  
end adder;
```

```
architecture one of adder is  
  signal c : bit;  
begin  
  sum <= c xor carry;  
  c <= a xor b;  
end one;
```



deklaracija notranjega signala

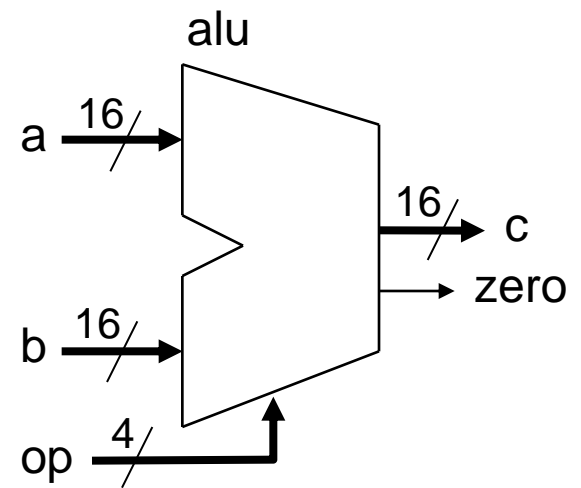
- ▶ V arhitekturnem stavku deklariramo notranje in izhodne signale

Zunanji signali

- ▶ Zunanji signali so enobitni (**bit**) ali vektorski signali (**bit_vector**)

```
entity alu is
  port ( a, b : in  bit_vector(15 downto 0);
        op  : in  bit_vector(3  downto 0);
        c   : out bit_vector(15 downto 0);
        zero :out bit );
end alu;
```

MSB LSB



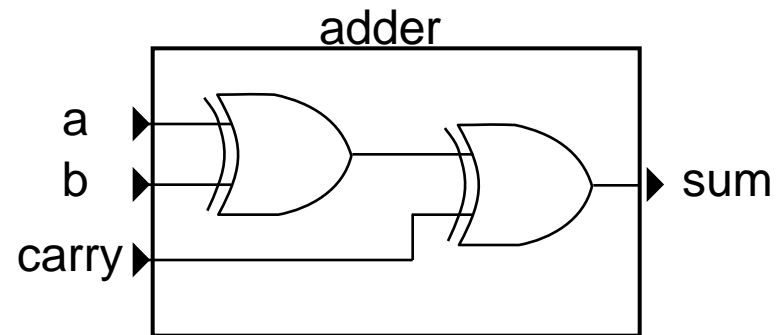
- ▶ Podatkovni tip **bit** pozna dve vrednosti: 0 ali 1
 - ▶ za bolj realistično simulacijo potrebujemo več-vrednostno logiko ('Z', 'U', 'X')
 - ▶ potrebujemo tudi funkcije za razrešitev
 - ▶ npr. ko signalu vsilimo vrednost '0' in 'Z', prevlada vrednost '0'

Večvrednostna logika: std_logic

- ▶ V knjižnici IEEE: `std_logic` in `std_logic_vector`
 - ▶ 'U' nedefinirano stanje
 - ▶ 'X' kratak stik
 - ▶ 'Z' visoka impedanca (odprte sponke)
 - ▶ 'H', 'L' šibko visoko, nizko stanje (pullup, pulldown)
 - ▶ ...

vključimo knjižnico

```
library IEEE;  
use IEEE.std_logic_1164.all;  
  
entity adder is  
  port ( a, b : in std_logic;  
        carry : in std_logic;  
        sum : out std_logic );  
end adder;
```



Numerični vektorji: signed in unsigned

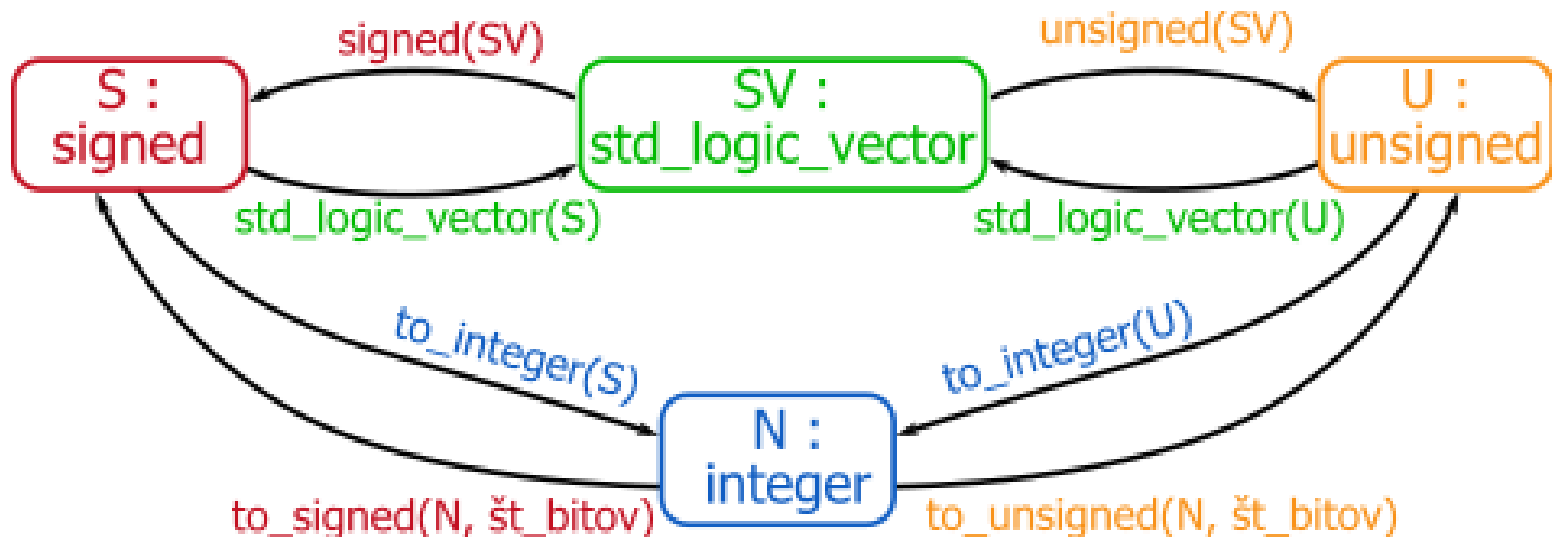
- ▶ Kako interpretiramo vrednost dvojiškega vektorja ?
- ▶ V knjižnici IEEE: **numeric_std** sta definirana dva numerična podatkovna tipa: **signed** in **unsigned**
 - ▶ v stavkih jezika VHDL je potrebno zapisati pretvorbo med standardnimi in numeričnimi vektorji

```
library IEEE;  
use IEEE.std_logic_1164.all;  
use IEEE.numeric_std.all;  
  
architecture one of test is  
  
    signal a, b: std_logic_vector(3 downto 0);  
    signal an: unsigned(3 downto 0);  
  
begin  
  
    an <= unsigned(a);  
    b <= std_logic_vector(an);
```

V izrazih jezika VHDL je potrebna eksplicitna pretvorba podat. tipov!

Pretvorba podatkovnih tipov

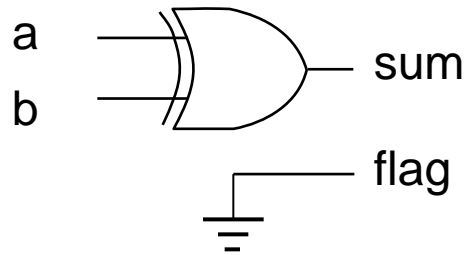
- ▶ Funkcije za pretvorbo podatkovnih tipov
 - ▶ pretvorbo med integer in std_logic_vector delamo preko podatkovnih tipov signed ali unsigned



Prireditveni stavek

- ▶ S prireditvenim stavkom signalu priredimo konstantno vrednost ali izraz:

```
flag <= '0';  
sum <= a xor b;
```



- ▶ Prireditveni stavek predstavlja logiko, katere izhod je signal, ki mu prirejamo vrednost

Signalu ne smemo hkrati (v več stavkih) prirejati različne vrednosti, ker bi opisali kratek stik!

Konstantne vrednosti

- ▶ Konstantno vrednost tipa `bit` ali `std_logic` zapišemo med enojne narekovaje:

```
one <= '1';  
tristate <= 'Z';
```

- ▶ Vektorske konstante (`std_logic_vector`, `unsigned`) pišemo med dvojne narekovaje:

```
a <= "11110000";  
b <= X"3A";  
c <= (others => '0');
```

šestnajstiška konstanta

agregat

postavi vse bite vektorja na 0

Deklaracija signalov in konstant

- ▶ Zunanji signali so deklarirani v stavku **port**
- ▶ Notranje signale in konstante deklariramo v **arhitekturnem** stavku
 - ▶ signalu lahko ob deklaraciji priredimo začetno vrednost
 - ▶ konstantam moramo prirediti vrednost !

```
architecture one of test is
```

```
    signal count: unsigned(3 downto 0) := "0011";
```

```
    constant zero: std_logic := '0';
```

```
begin
```


Aritmetični izrazi

- ▶ Osnovne računske operacije: **+**, **-**, *****
 - ▶ definirane so v IEEE.numeric_std za izraze, kjer nastopajo numerični vektorji **signed** ali **unsigned**

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

architecture one of test is
    signal a, b, sum, inc, m: unsigned(7 downto 0);
    signal d, e: unsigned(3 downto 0);
begin
    sum <= a + b;      -- 8 bitni seštevalnik
    inc <= a + 1;     -- prišteje celoštevilsko konstanto (integer)
    m <= d * e;       -- 4 x 4 biti = 8 bitni rezultat
```

Operacije z vektorji

```
signal a, b, c: std_logic_vector(7 downto 0);  
signal high, low: std_logic_vector(3 downto 0);
```

▶ Podvektor:

```
high <= a(7 downto 4);  -- 4 bitni podvektor  
low <= a(3 downto 0);  -- 4 bitni podvektor  
sign <= a(7);          -- sign je tipa std_logic
```

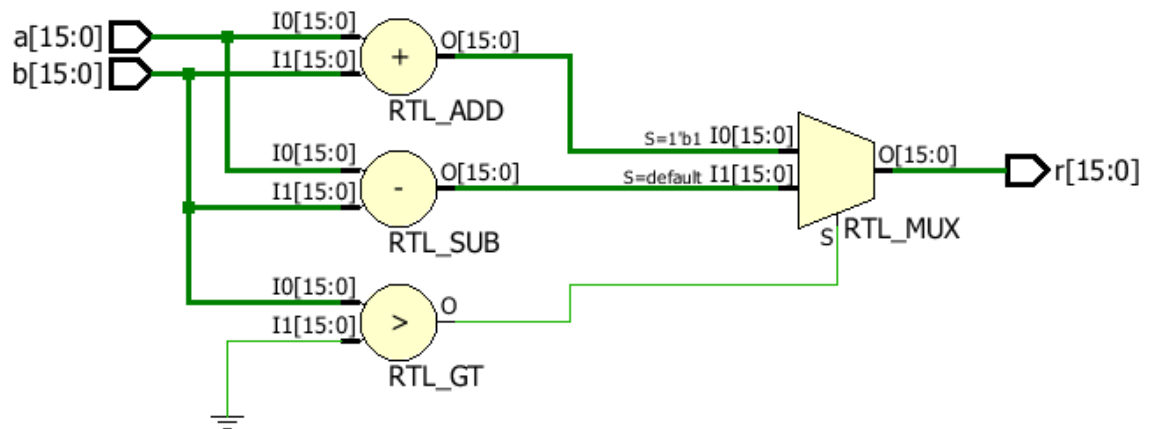
▶ Logične operacije: **and, or, not, xor, xnor...**

▶ Sestavljanje vektorjev: **&**

```
a <= high & low;  
b <= sign & "000" & low;
```

Pogojni prireditveni stavek

```
mux <= a when select='0' else b;  
flag <= '1' when a>b else '0';  
add <= a+b when b>0 else a-b;
```



► relacijski operatorji:

=	/=	>	>=	<	<=
enako	ni enako	večje	večje ali enako	manjše	manjše ali enako