



Laboratorij za načrtovanje integriranih vezij

Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*



Digitalni Elektronski Sistemi

Osnove jezika VHDL

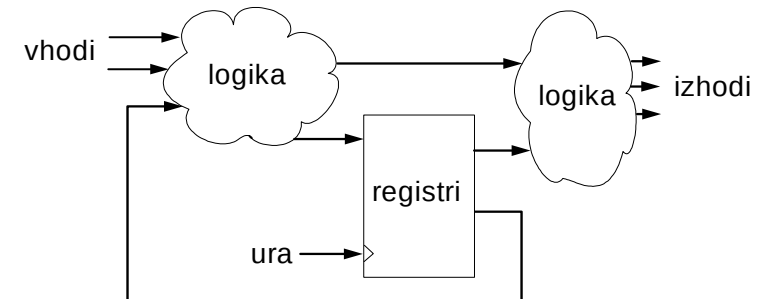
1. del: signali, operacije in simulacija

Visokonivojski jeziki za opis vezij

- ▶ Standardizirani jeziki (IEEE standard)
 - ▶ VHDL (Very high-speed IC Hardware Description Language)
 - ▶ Verilog

- ▶ Modeliranje na različnih nivojih

- ▶ logična vrata, RTL (Register Transfer Level)
 - ▶ nivo obnašanja (zapišemo algoritem)



- ▶ Funkcionalna in časovna simulacija vezja

- ▶ s simulacijo preverimo delovanje vezja

- ▶ Sinteza vezja

- ▶ pretvorba vezja na nivo logičnih vrat in flip-flopov



Osnovni koncepti jezika VHDL

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

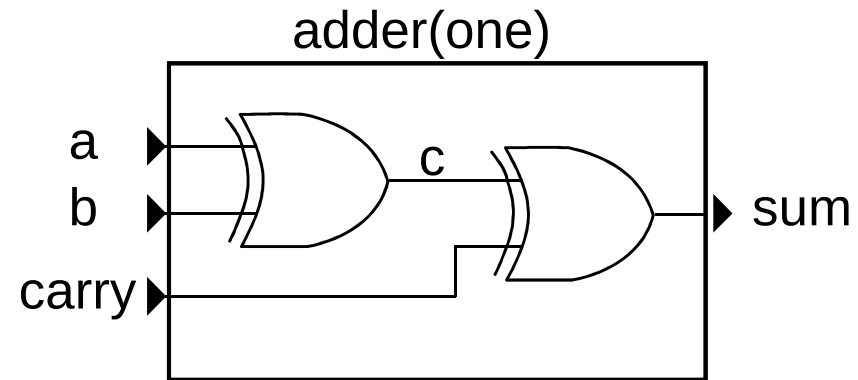
```
entity adder is
  port ( a, b : in bit;
         carry : in bit;
         sum : out bit );
end adder;
```



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;
```

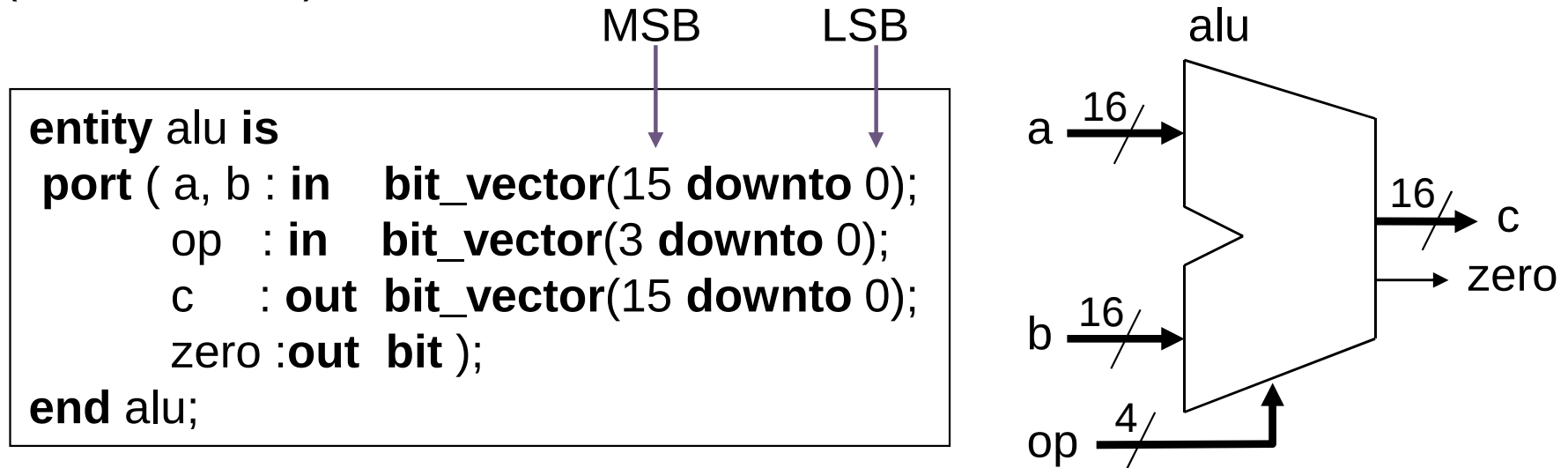


dekleracija notranjega signala

- ▶ V arhitekturnem stavku določimo notranje in izhodne signale

Zunanji signali

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



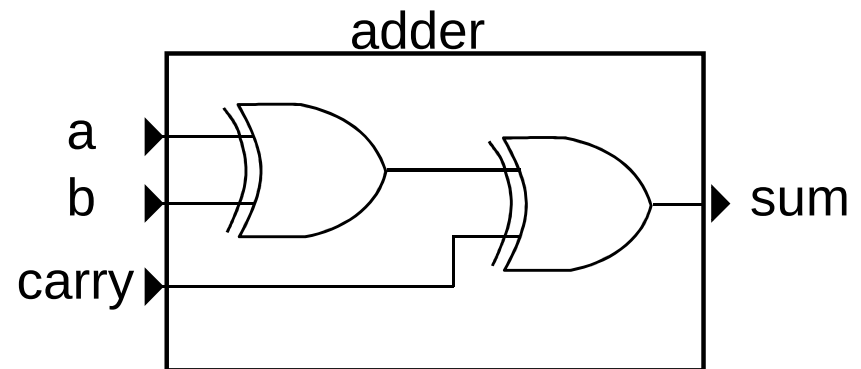
- ▶ Podatkovni tip **bit** pozna dve vrednosti: 0 ali 1
 - ▶ za bolj realistično simulacijo potrebujemo več-vrednostno logiko ('Z', 'U', 'X')
 - ▶ potrebujemo tudi funkcije za razrešitev
 - ▶ npr. ko signalu vsilimo vrednost '0' in 'Z', prevlada vrednost '0'

Večvrednostna logika: std_logic

- ▶ V knjižnici IEEE: `std_logic` in `std_logic_vector`
 - ▶ 'U' nedefinirano stanje
 - ▶ 'X' kratek stik
 - ▶ 'Z' visoka impedanca (odprte sponke)
 - ▶ 'H' pasivno visoko stanje (pullup)
 - ▶ ...

```
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;
```

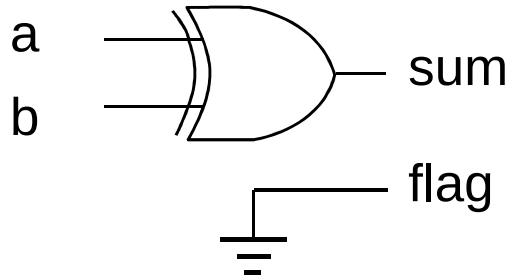
vkjučimo knjižnico



Prireditveni stavek

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

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



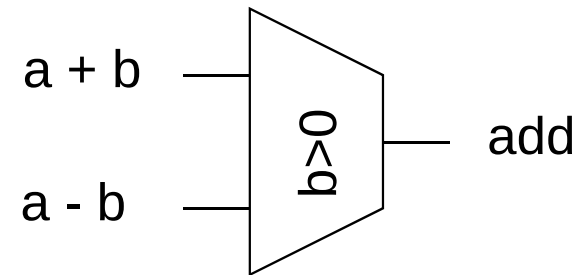
- ▶ Prireditveni stavek predstavlja logiko, katere izhod je signal, ki mu prirejamo vrednost

Za isti signal ne moremo imeti več sočasnih prireditvenih stavkov, ker bi opisali kratek stik!

Pogojni prireditveni stavek

- ▶ Signalu priredimo vrednost ali izraz ob določenem pogoju

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



- ▶ Pogojni prireditveni stavek v osnovni obliki predstavlja izbiralnik (multiplekser)
 - ▶ izrazi predstavljajo logiko na vhodu izbiralnika, pogoj pa logiko za izbiro vhoda
- ▶ Relacijski operatorji:

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

Konstantne vrednosti

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

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

- ▶ Vektorske konstante (`bit_vector` ali `std_logic_vector`) imajo 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**, notranji pa v **arhitekturnem** stavku
 - ▶ signalu lahko ob deklaraciji priredimo začetno vrednost
- ▶ Konstante deklariramo v **arhitekturnem** stavku

```
architecture one of test is
```

```
  signal flag: std_logic;
```

```
  signal reg: std_logic_vector(3 downto 0) := "0000";
```

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

```
begin
```



Deklaracija zunanjih signalov

- ▶ Zunanji signali imajo poleg podatkovnega tipa določeno tudi smer

in ▶ vhodni signali, ki se lahko pojavljajo le na desni strani prireditvenih stavkov in v pogojih

out ▶ izhodni signali, ki se lahko pojavljajo le na levi strani prireditvenih stavkov

inout ▶ dvosmerni signali (npr. dvosmerna vodila), ki se lahko pojavljajo na obeh straneh prireditev

buffer ▶ izhodni signali, ki pa jih lahko uporabljamo tudi na desni strani prireditev in v pogojih



Primer deklaracije: buffer

```
entity increment is  
  port ( a: in std_logic_vector(3 downto 0);  
         incr_a : buffer std_logic_vector(3 downto 0);  
         flag : out std_logic );  
end increment;  
  
architecture one of increment is  
begin  
  incr_a <= a + '1';  
  flag <= '1' when incr_a="1111" else '0';  
end one;
```

- ▶ Signal incr_a mora biti deklariran kot **buffer**, ker se pojavlja na levi strani prireditve (kot izhod) in v pogoju (kot vhod)



Podatkovni tipi: vektorji

- ▶ Vektorje uporabljamo za modeliranje večbitnih signalov (vodil)
- ▶ Območje indeksov običajno deklariramo od MSB proti LSB

```
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
```



Operacije z vektorji

- ▶ Logične operacije: **and, or, not, xor, xnor...**
- ▶ Sestavljanje vektorjev: **&**

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

- Pomikanje vektorjev
 - pomik naredimo s sestavljanjem in podvektorjem

```
b <= a(6 downto 0) & '0';    -- pomik v levo  
c <= '0' & a(7 downto 1);    -- pomik v desno  
d <= a(6 downto 0) & a(7);  -- rotacija v levo
```



Aritmetične operacije z vektorji

- Aritmetične operacije: $+$, $-$, $*$
 - vključimo knjižnjico: `std_logic_unsigned` (nepredznačena) ali `std_logic_signed` (predznačena števila)

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

architecture one of test is
    signal a, b, sum, inc, m: std_logic_vector(7 downto 0);
    signal d, e: std_logic_vector(3 downto 0);
begin
    sum <= a + b;      -- 8 bitni seštevalnik
    inc <= a + '1';    -- '1' se razširi na "00000001" in prišteje
    m <= d * e;        -- 4 x 4 biti = 8 bitni rezultat
```



Podatkovni tipi: signed, unsigned

- ▶ Predznačeni ali nepredznačeni vektorji
- ▶ Vključimo knjižnico: IEEE.numeric_std
- ▶ Definirane osnovne aritmetične operacije

```
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';    -- '1' se razširi na "00000001" in prišteje
    m <= d * e;        -- 4 x 4 biti = 8 bitni rezultat
```



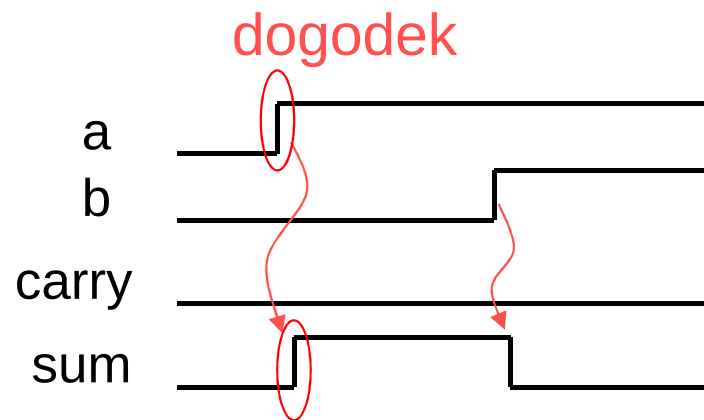
Obravnava VHDL modelov vezij

► Simulacija modela

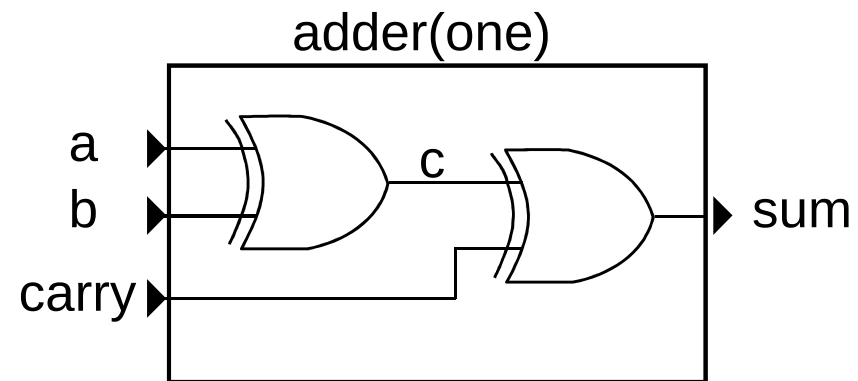
- na simulatorju določimo spreminjaje vhodnih signalov in opazujemo izhode

► Sinteza modela

- program za sintezo določi zgradbo vezja, ki izhaja iz VHDL modela



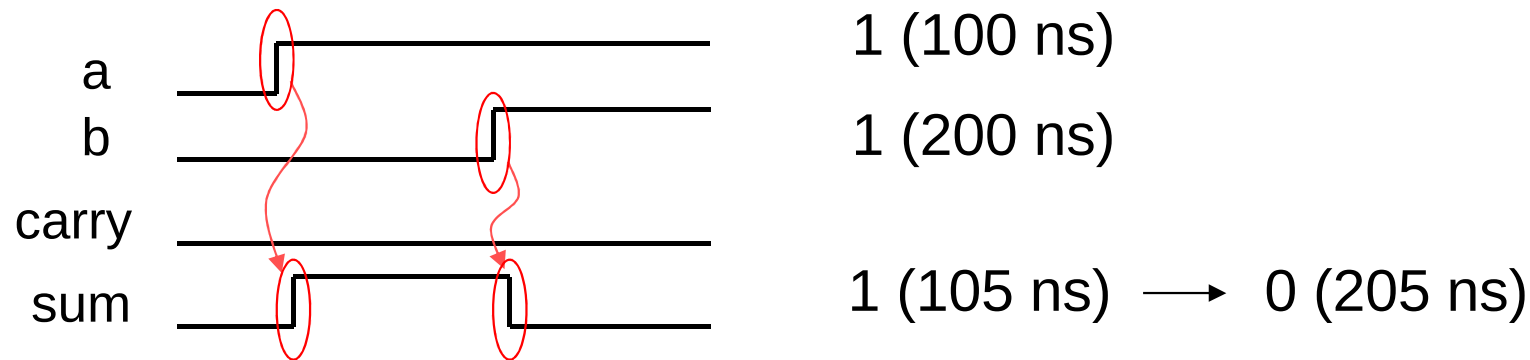
Graf simulacije (waveform)



Sintetizirano vezje

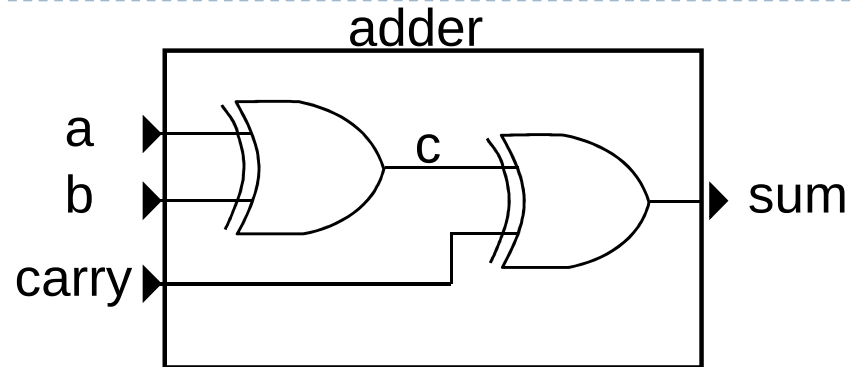
Simulator diskretnih dogodkov

- ▶ Dogodki so opisani z vrednostjo signala in časom ob katerem se zgodijo



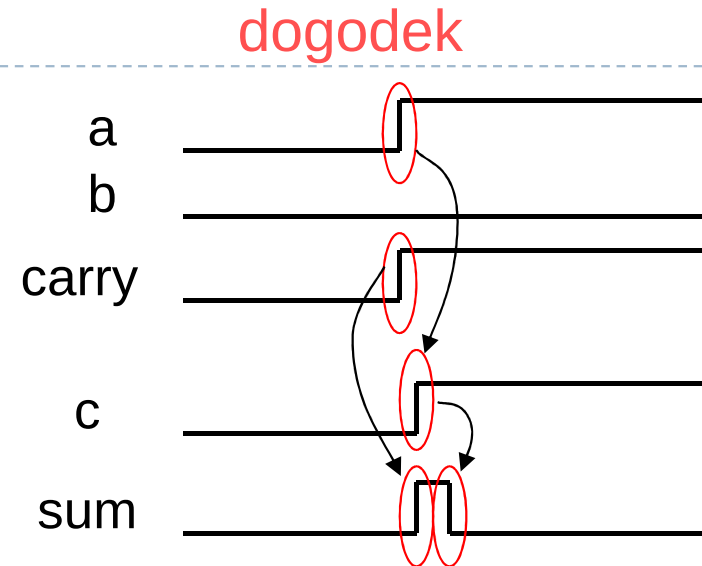
- ▶ Ciklus simulatorja
 1. simulator izračunava dogodke na signalih in jih uvršča na seznam dogodkov
 2. po vseh izračunih se poveča simulacijski čas in
 3. izvrši dogodke, ki spremenijo vrednosti signalom

Potek simulacije



Seznam dogodkov:

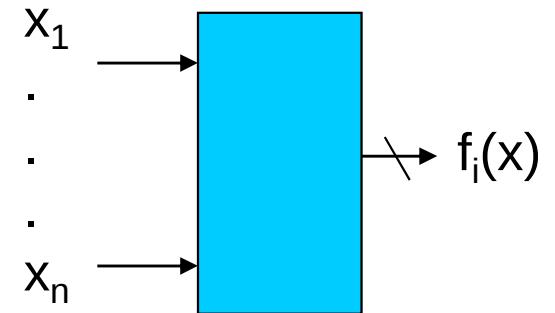
a: 0, 1 (T) carry: 0, 1 (T)



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

Lastnosti realnih kombinacijskih vezij

- ▶ Izhod se spremeni z zakasnitvijo ($\Delta > 0$)
- ▶ Večina vezij je večnivojskih
 - ▶ zaporedno vezana logična vrata
 - ▶ v času spremembe so na izhodu motnje!



- ▶ Primer: seštevalnik in primerjalnik

```
add <= a + 1;  
n <= '1' when add=0 else '0';
```

