Osnove strojno-opisnega jezika

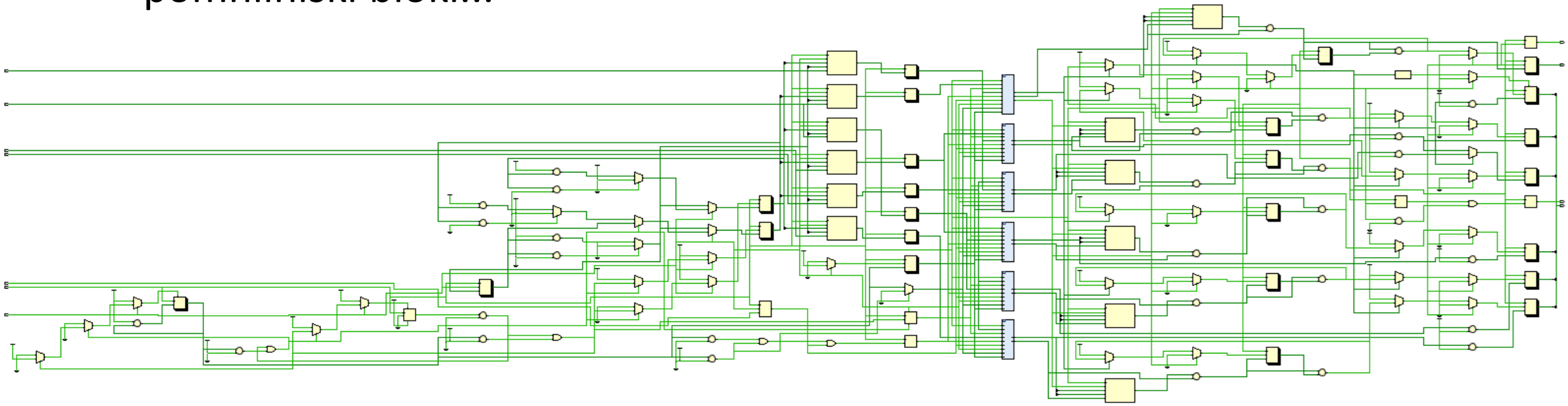
Digitalni elektronski sistemi 2024/25

Andrej Trost

UNIVERZA V LJUBLJANI | FE Fakulteta za elektrotehniko

Logično vezje

- gradniki vezja: izbiralniki, seštevalniki, primerjalniki, flip-flopi, registri, pomnilniški bloki...



Modeli digitalnih elektronskih naprav

Enostavni (abstraktni) in podrobni modeli

- algoritem
  - poenostavljen opis delovanja računalniške naprave
  - npr. opis zaporedja operacij
- logično vezje
  - poenostavljen opis digitalnega elektronskega vezja
  - npr. opis transformacij logičnih vrednosti
- analogno vezje
  - poenostavljen opis elektronske naprave
  - idealizirane komponente vezja (npr. upor ima le upornost)
- fizikalni model
  - opis naprave s fizikalnimi zakoni
  - npr. opis dogajanje v plasteh integriranega vezja

Zakaj potrebujemo strojno-opisni jezik

HDL – Hardware Description Language

- učinkovito načrtovanje in simuliranje digitalnih vezij
- bolj abstrakten v primerjavi z Boolovo logiko (manj podrobnosti)
  - hitrejši razvoj kompleksnih vezij
  - hitrejše testiranje vezij
- standarden opis podpira različna orodja in tehnologije
- avtomatska pretvorba v shemo in nižji nivo opisa
  - sinteza vezja

Verilog

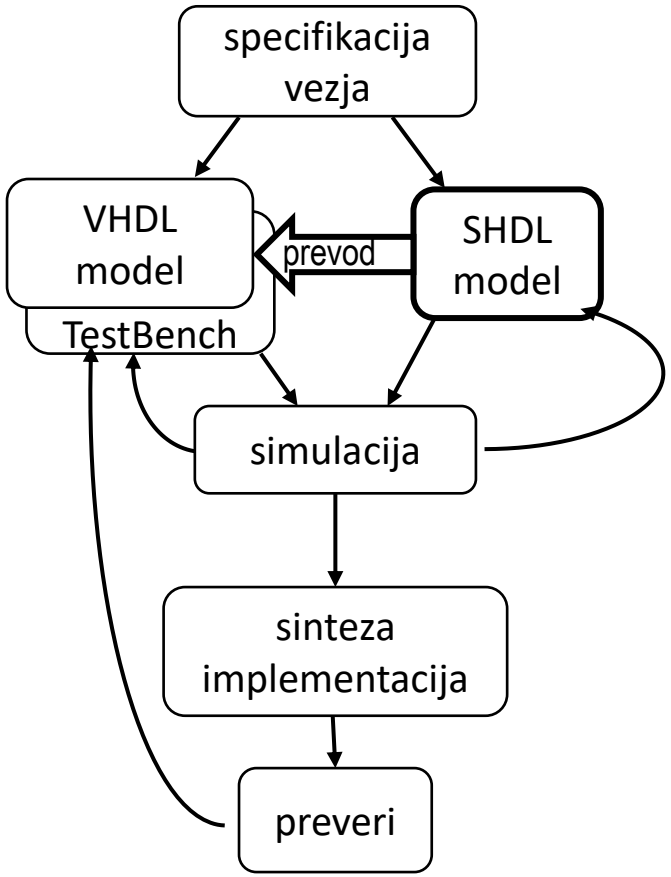
SystemVerilog

VHDL

Very high-speed IC
Hardware
Description
Language

SHDL

Small HDL



IEEE standard: Verilog, SystemVerilog, in VHDL

SHDL je poenostavljen VHDL za učne namene
razvit v Laboratoriju za načrtovanje integriranih vezij

Lastnosti jezikov Verilog in VHDL

- Verilog in VHDL opisujeta splošna logična vezja, omogočata simulacijo vezij in sintezo v izbrano tehnologijo

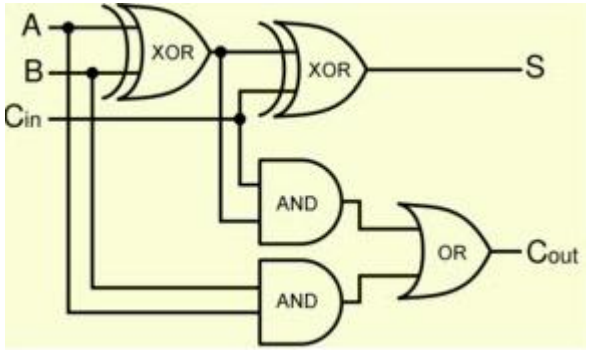
lastnost	Verilog	VHDL
standard	IEEE 1364 ('95, 2001)	IEEE 1076 ('93, 2008)
sintaksa	jedrnata, C šibka pravila	gostobesedna, ADA stroga pravila
podatkovni tipi	malo	veliko
sočasnost	slabše določena	dobro določena
podpora simulacije	osnovna (močna: SystemVerilog)	močna
uporaba	ASIC/FPGA (ZDA) hitra digitalna vezja	FPGA (Evropa) kritični, kompleksni modeli

Načini modeliranja vezja

Strukturni opis

- opis sheme
- primer v jeziku Verilog

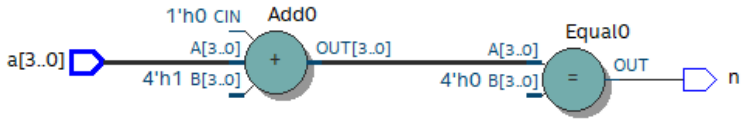
```
module FA(a,b,cin,s,cout);
 input a, b, cin;
 output s, cout;

 xor(x, a, b);
 xor(s, x, cin);
 and(c1, a, b);
 and(c2, cin, x);
 or(cout, c1, c2);

endmodule
```



Funkcijski opis (dataflow)

- prireditveni stavki opisujejo gradnike
- opis sledi zgradbi vezja, primer v SHDL

```
entity logika
 a: in u4;
 add:  u4;
 n: out u1;
begin
 add = a + 1
 n = 1 when add=0 else 0
end
```

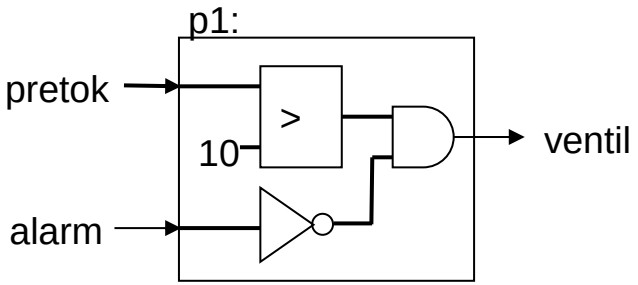



```
assign add = a + 1;
assign n = (add == 0)? 1 : 0;
```

Postopkovni opis

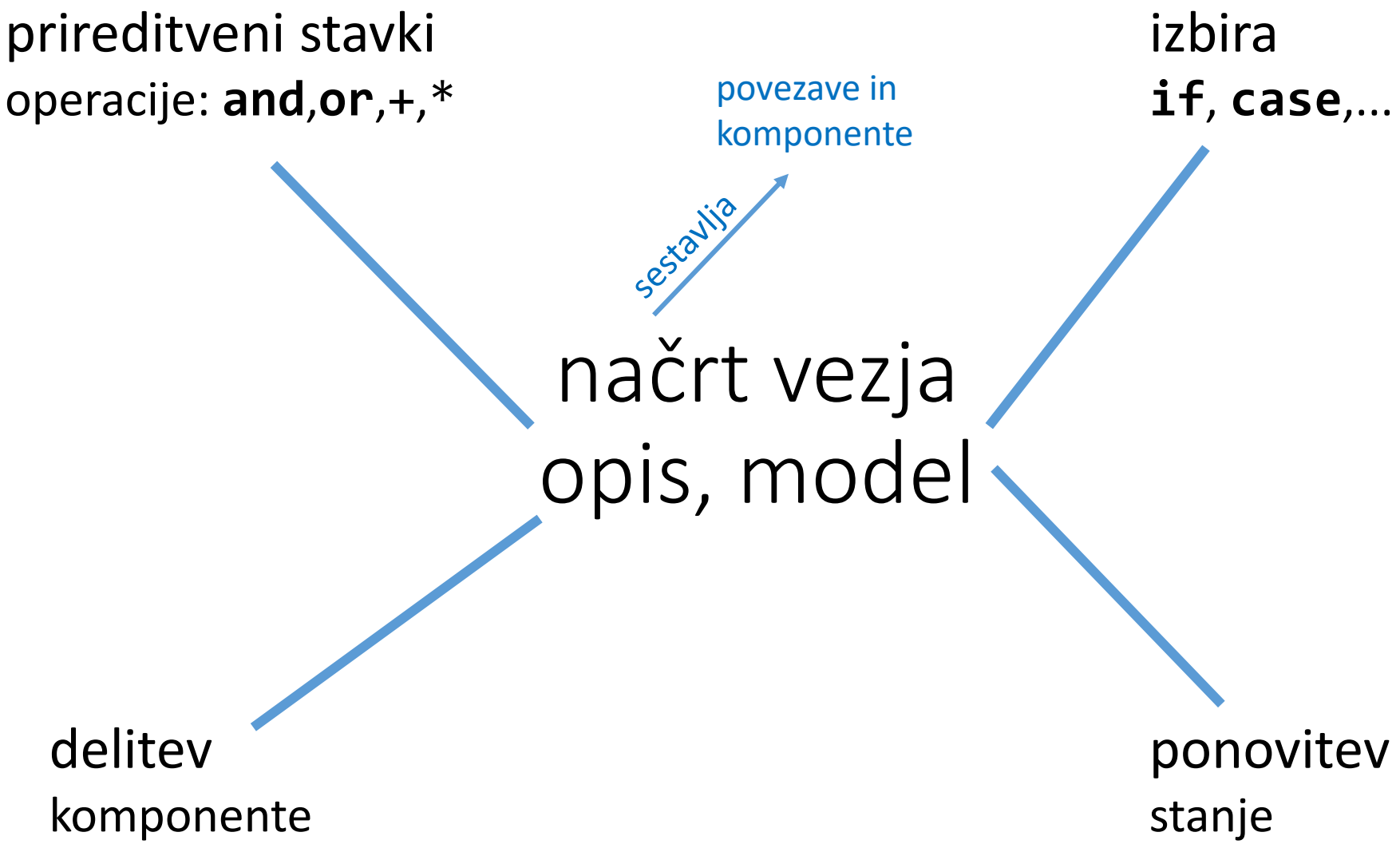
- zgradbo vezja določi program za sintezo
- primer procesa v jeziku VHDL

```
p1: process(pretok, alarm)
begin
   ventil <= '0';
   if pretok > 10 then
      ventil <= '1';
   end if;
   if alarm = '1' then
     ventil <= '0';
   end if;
end process;
```

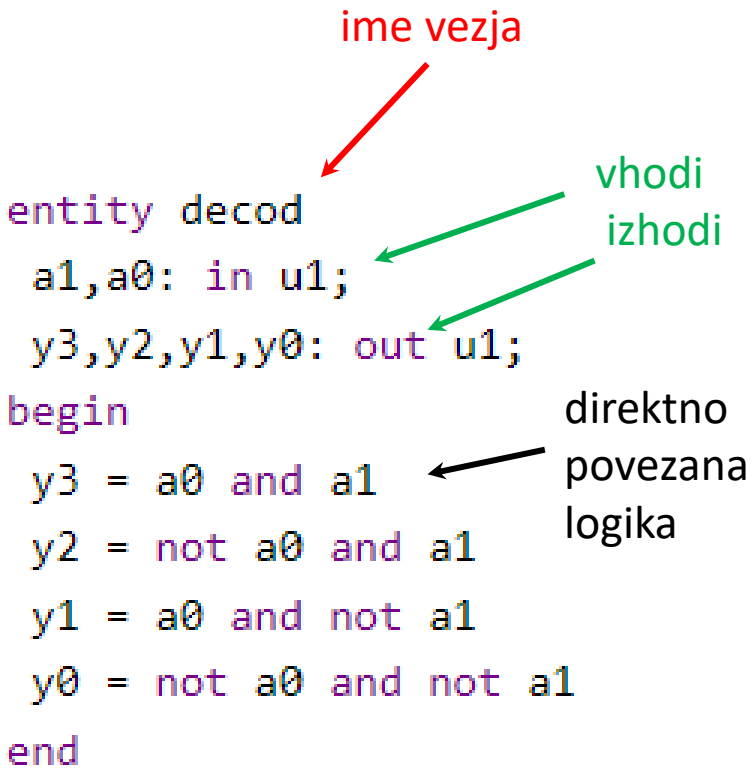


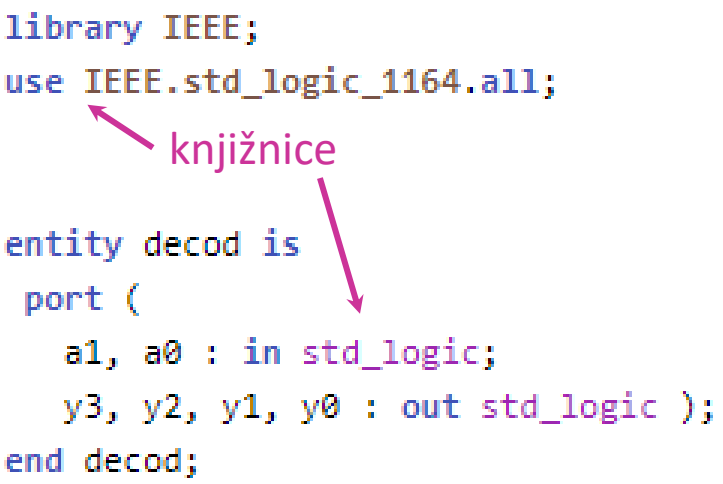

```
always @* begin
  ventil = 0;
  if (pretok > 10) ventil = 1;
  if (alarm == 1) ventil = 0;
end
```

Kaj delamo v strojno-opisnem jeziku?



Povezave vezja: SHDL / VHDL

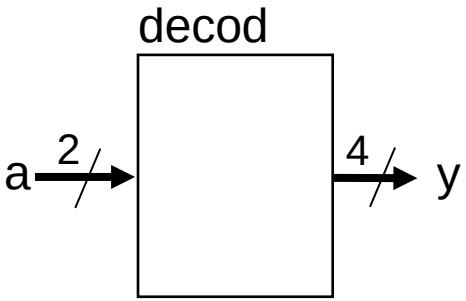




u1: 1-bitna povezava, 0 ali 1

std_logic: '0','1','Z','U','X','H','L'

Večbitni (vektorski) priključki



SHDL

```
entity decod
 a: in u2
 y: out u4
```

VHDL

MSB LSB

```
entity decod is
port (a :  in  std_logic_vector (1 downto 0);
        y : out std_logic_vector (3 downto 0)
 );
end decod;
```

Omejitev: u64

Omejitev: 2,147,483,647 bitov?

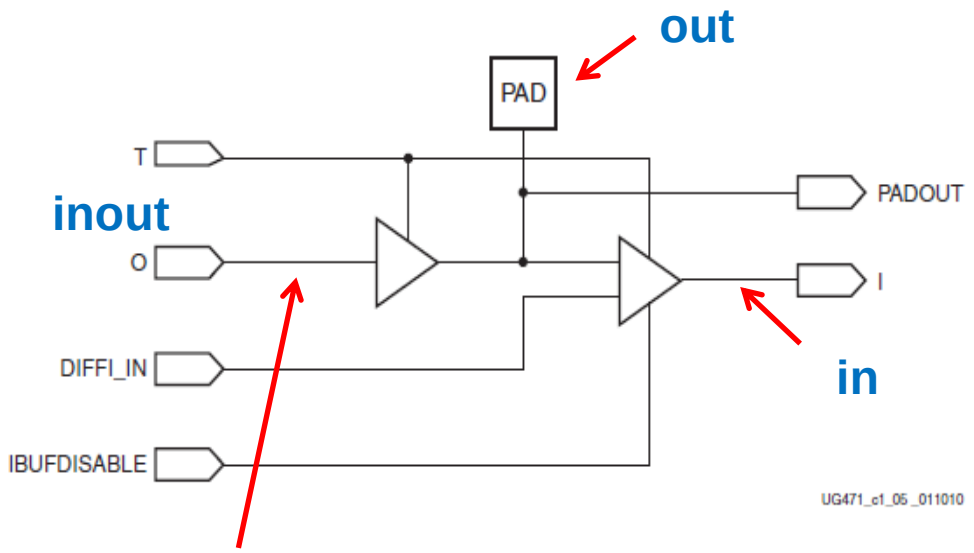
v praksi 1-8M

*VHDL priključki

- VHDL opisuje poleg vhodov (smer: **in**) in izhodov (**out**) tudi prave dvosmerne priključne (**inout**) ter izhode z notranjo povezavo (**buffer**)

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

smer: in, out, inout, buffer



buffer (notranji in izhodni signal)

Podatkovni tip vektorjev

SHDL

- nepredznačen, npr. u4
  - vrednost: "1101" ali 13
- predznačen, npr. s4
  - "1101" ali -3

VHDL

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

- dvojiški: **std_logic_vector**, "0101"
- celoštevilski: **integer**, 5
- nepredznačen: **unsigned**, "1101" (13)
- predznačen: **signed**, "1101" (-3)

Signali in konstante

```
entity test

not_sig: u4
konst0: u1 = 0

begin
```

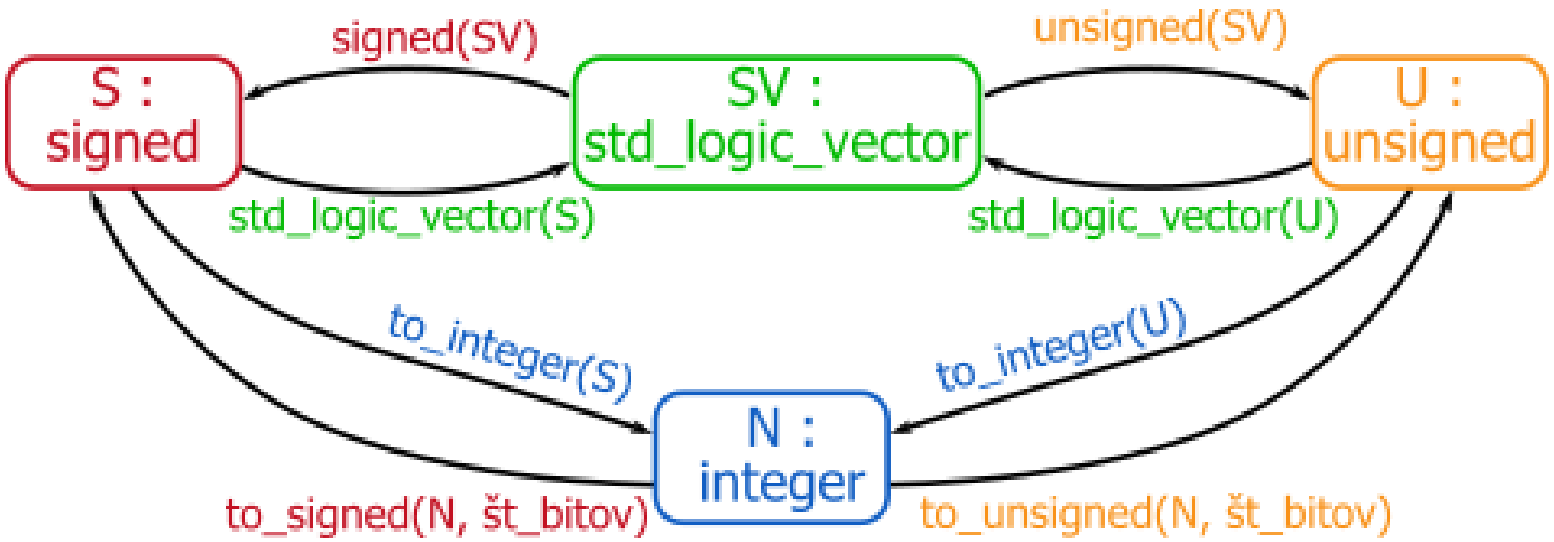
```
architecture one of test is

 signal not_sig: unsigned(3 downto 0);

 constant konst0: std_logic := '0';

begin
```

- enobitne: '0','1' ali 0,1
- dvojiški vektor: "011", 3, 0x3

- enobitne: '0' ali '1'
- vektorske: "011"
- šestnajstiške: X"3"
- cel vector na 0: (**others** => '0');

*VHDL zahteva eksplicitno pretvarjanje tipov

```
signal a, b: std_logic_vector(3 downto 0);
signal an: unsigned(3 downto 0);

begin

an <= unsigned(a);
b <= std_logic_vector(an);
```



HDL prireditveni stavek

V jeziku C/C++

a = 1;

a = b;

V jeziku VHDL

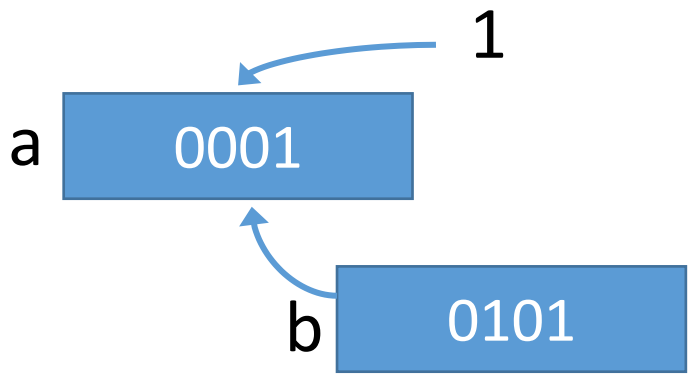
a <= '1';

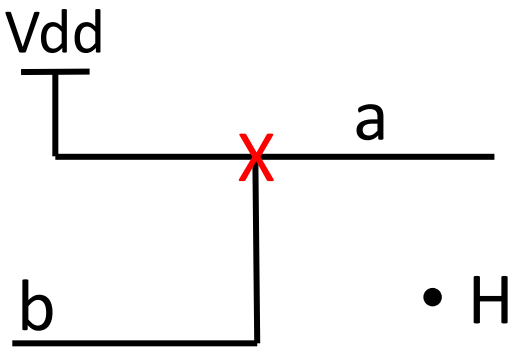
a <= b;

V jeziku SHDL

a = 1

a = b

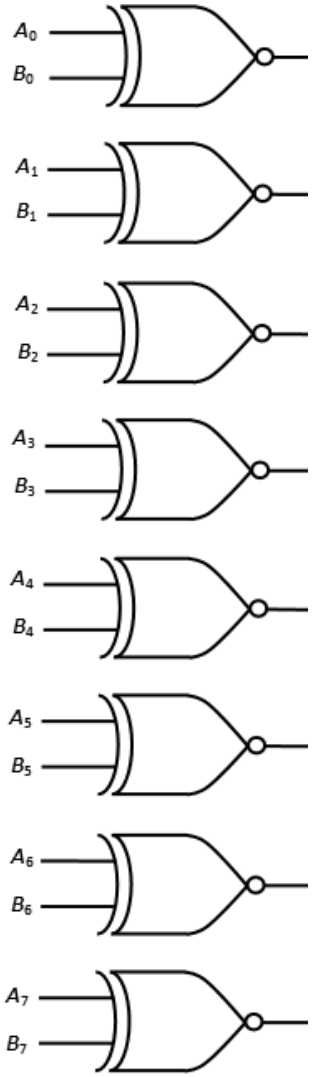




- HDL opisuje kratek stik!

Vektorske operacije

```
a,b,c: u8;


c = not (a xor b);
```



signed, unsigned

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

c <= not (a xor b);
```

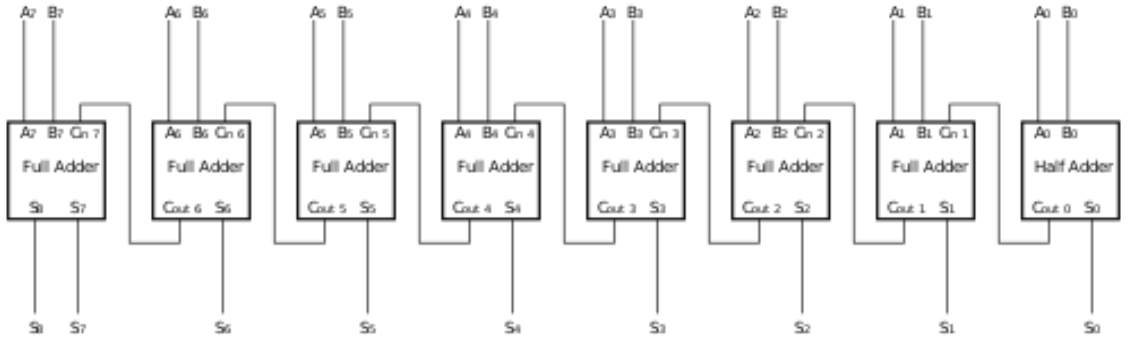
- **and, or, not, xor, xnor...**
- Podvektor in sestavljanje: **&**

```
high = a(7 downto 4)
low = a(3 downto 0)
sign = a(7)

b = high & low
c = sign & "000" & low
```

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

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

Seštevalnik

- v jeziku VHDL uporabimo tip **signed** ali **unsigned**




```
entity add
 a,b: in u8;
 sum,inc: out u8;
begin
 sum = a+b
 inc = a+1
end
```

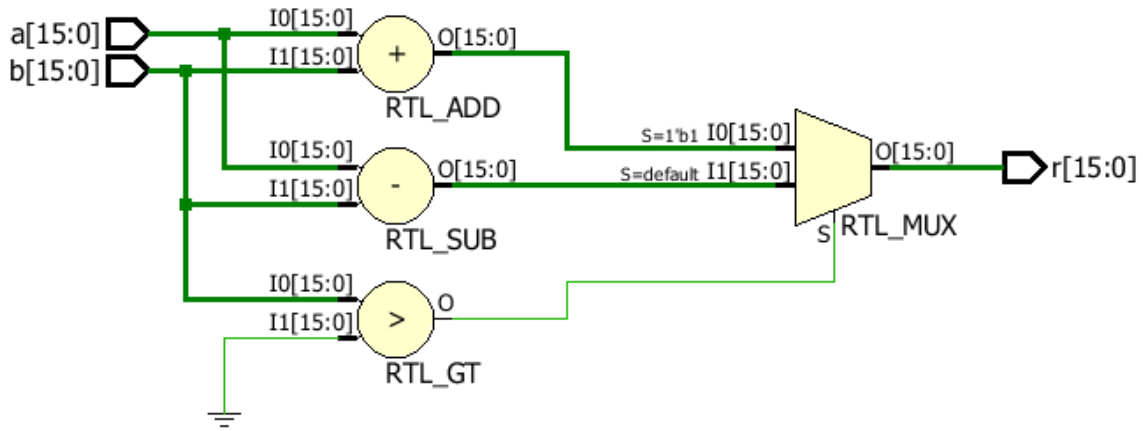
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity add is
 port (
   a, b : in unsigned(7 downto 0);
   sum, inc : out unsigned(7 downto 0) );
end add;

architecture RTL of add is
begin
sum <= a + b;
inc <= a + 1;

end RTL;
```

- Kaj pa vsota s prenosom?

Pogojni prireditveni stavek

```
mux = a when sel=0 else b  -- izbiralnik
flag = 1 when a>b else 0  -- primerjalnik
add = a+b when b>0 else a-b
```

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



- relacijski operatorji:

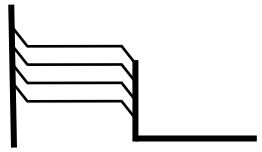
=	/=	>	>=	<	<=
enako	ni enako	večje	večje ali enako	manjše	manjše ali enako

Funkcijski opis kombinacijske logike

- premik bitov

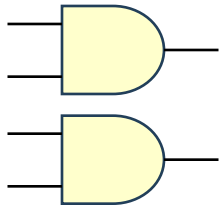
z5 = a4 & '0'

z4 = a4 **sll** 1



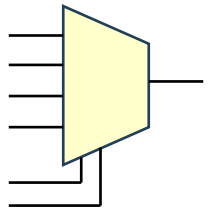
- logična vrata

z2 = a2 **and** b2



- izbiralnik

z1 = a4(n2)



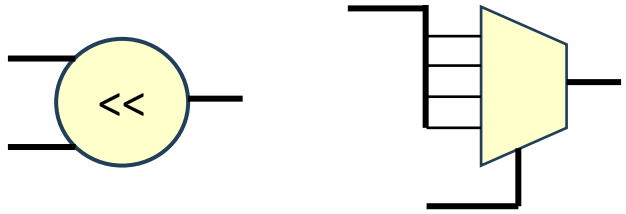
- ROM

c: u4 = "1001"

z1 = c(n2)

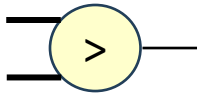
- pomikalnik (splošni)

z8 = a8 **sll** n3



- primerjalnik

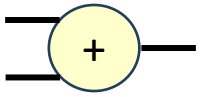
z1 = 1 **when** a4>b4 **else** 0



- seštevalnik

z5 = a4 + b4

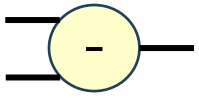
z4 = a4 + b4



- odštevalnik

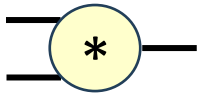
z5 = a4 - b4

z4 = a4 - b4

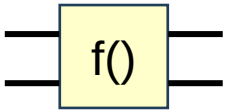


- množilnik

z8 = a4 * b4



- delilnik, ... ?



Postopkovni opis: VHDL proces

Znotraj procesnega okolja so sekvenčni stavki

- vrstni red stavkov je v procesu lahko pomemben
- med sekvenčne stavke spada pogojni (if) stavek
- dovoljeno je narediti več prireditev enemu signalu, dejansko se izvrši le zadnja prireditev

```
primerjava: process (a, b)
begin
   enako <= '0';
   if a=b then
      enako <= '1';
   end if;
end process;
```

```
enako = 0
if a=b then
   enako =  1
end
```

SHDL: pri prevodu v VHDL naredi procesno okolje

*VHDL: zgradba procesnega okolja

```
oznaka: process (seznam signalov)
begin
.....

end process;
```

- z oznako poimenujemo del vezja, ki ga predstavlja proces
- simulacija procesa se izvrši ob spremembi enega izmed signalov iz seznama
  - v kombinacijskih vezjih navedemo vse signale, ki predstavljajo vhode v proces
  - VHDL-2008 pozna skrajšano obliko: **process (all)**

Pogojni stavek

- opcijski **else**, če pogoj ni izpolnjen
- **elsif** za povezane pogoje

```
if pogoj then
   stavki(1);
else
   stavki(2);
end if;
```

```
if pogoj then
   stavki(1)
else
   stavki(2)
end
```

```
if pogoj1 then
   stavki(1);
elsif pogoj2 then
   stavki(2);
else
   stavki(3);
end if;
```

```
if pogoj1 then
   stavki(1)
elsif pogoj2 then
   stavki(2)
else
   stavki(3)
end
```

```
p_max: process(a, b)
begin
  if a>b then
     max <= a;
  else
     max <= b;
  end if;
end process;
```

Primer: 4-bitna binarna koda v BCD

- postopkovni opis s pogojnim stavkom

```
if bin>9 then
   enice = bin – 10
   desetice = 1
else
   enice = bin
   desetice = 0
end
```

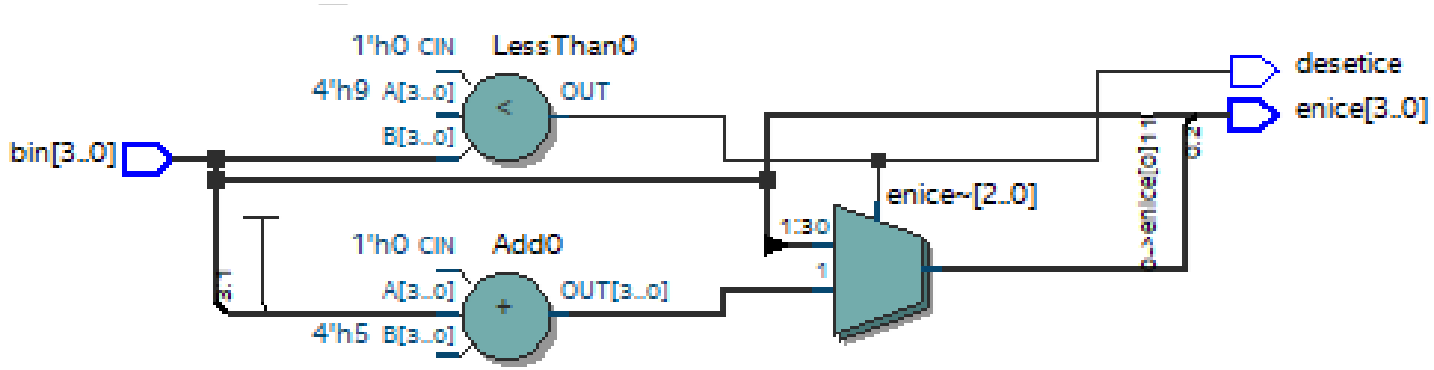
=

- ...ali pa funkcijski opis z dvema stavkoma

```
enice = bin - 10  when bin>9 else bin
desetice = 1 when bin>9 else 0
```

VHDL:

```
p: process(bin)
begin
 if bin>9 then
    enice <= bin – 10;
    desetice <= '1';
  else
    enice <= bin;
    desetice <= '0';
  end
end process;
```



Zaporedje pogojev: prioriteta

```
p: process(int0, int1, int2)
begin
  if int0='1' then
     v <= "01"
  elsif int1='1' then
     v <= "10";
  elsif int2='1' then
     v <= "11";
  else
     v <= "00";
  end if;
end process;
```

=

SHDL V3.0 razširjene oblike when…else ne pozna !

```
v <= "01"  when int0='1' else
     "10"  when int1='1' else
     "11"  when int2='1' else
     "00";
```

int0 ima prednost pred int1, ki ima prednost pred int2 ...

Zaporedje pogojev in stavek case*

```
if (int0) then v=1
elsif (int1) then v=2
elsif (int2) then v=3
else v=0
end
```

```
process(int0,int1,int2)
begin
  if int0 = '1' then
     v <= to_unsigned(1, 2);
  elsif int1 = '1' then
     v <= to_unsigned(2, 2);
  elsif int2 = '1' then
     v <= to_unsigned(3, 2);
  else
     v <= to_unsigned(0, 2);
  end if;
end process;
```

```
if digit=0 then display="00111111"
elsif digit=1 then display= "00000110"
else display= "11111001"
end
```

```
process(digit)
begin
 case digit is
   when "00" =>
     display <= "00111111";
   when "01" =>
     display <= "00000110";
   when others =>
     display <= "11111001";
  end case;
end process;
```

Primer: večbitni izbiralnik 4-1

```
if mode = "00" then
    mux = a
elsif mode = "01" then
    mux = b
elsif mode = "10" then
    mux = c
else
    mux = d
end
```

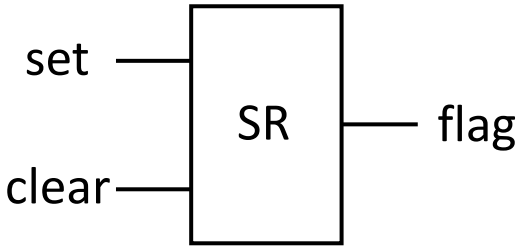
```
m: process(mode,a,b,c,d)
begin
  case mode is
    when "00" =>
      mux <= a;
    when "01" =>
      mux <= b;
    when "10" =>
      mux <= c;
    when others  =>
      mux <= d;
  end case;
end process;
```

Sekvenčna vezja

- proces omogoča prirejanje vrednosti signalom na več mestih
- če signalu ne določimo vrednosti, se bo ohranjala zadnja vrednost
  - dobimo sekvenčno vezje: nesinhronizirani zapah

```
p: process(set, clear)
begin
  if set='1' then
     flag <= '1';
  elsif clear='1' then
     flag <= '0';
  end if;
end process;
```

```
if set = 1 then
     flag = 1
elsif clear = 1 then
     flag = 0
end
```



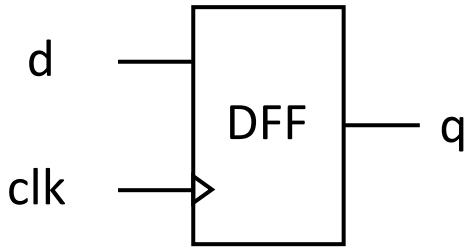
SHDL parser javi: Warning: Uncomplete assignment creates latch!

Sinhrona sekvenčna vezja

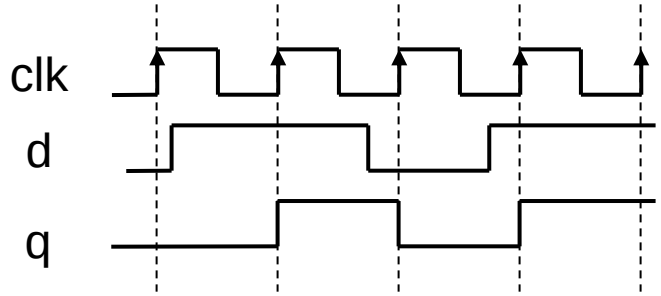
- stanja sinhronih vezij se spreminjajo ob uri
  - prva fronta: clk'event and clk='1' ali rising_edge(clk)
  - zadnja: clk'event and clk='0' ali falling_edge(clk)

```
FF: process (clk)
begin
   if rising_edge(clk) then
      q <= d;
   end if;
end process;
```

```
q <= d
```



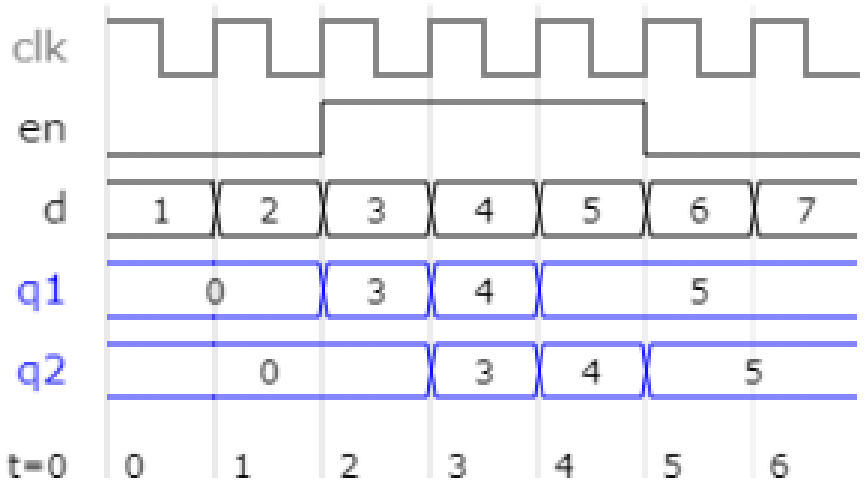
Podatkovni flip-flop



SHDL prireditev <= se izvede ob prvi fronti ure clk,
parser avtomatsko naredi procesno okolje s pogojem za fronto ure

Zapah in register s flip-flopi

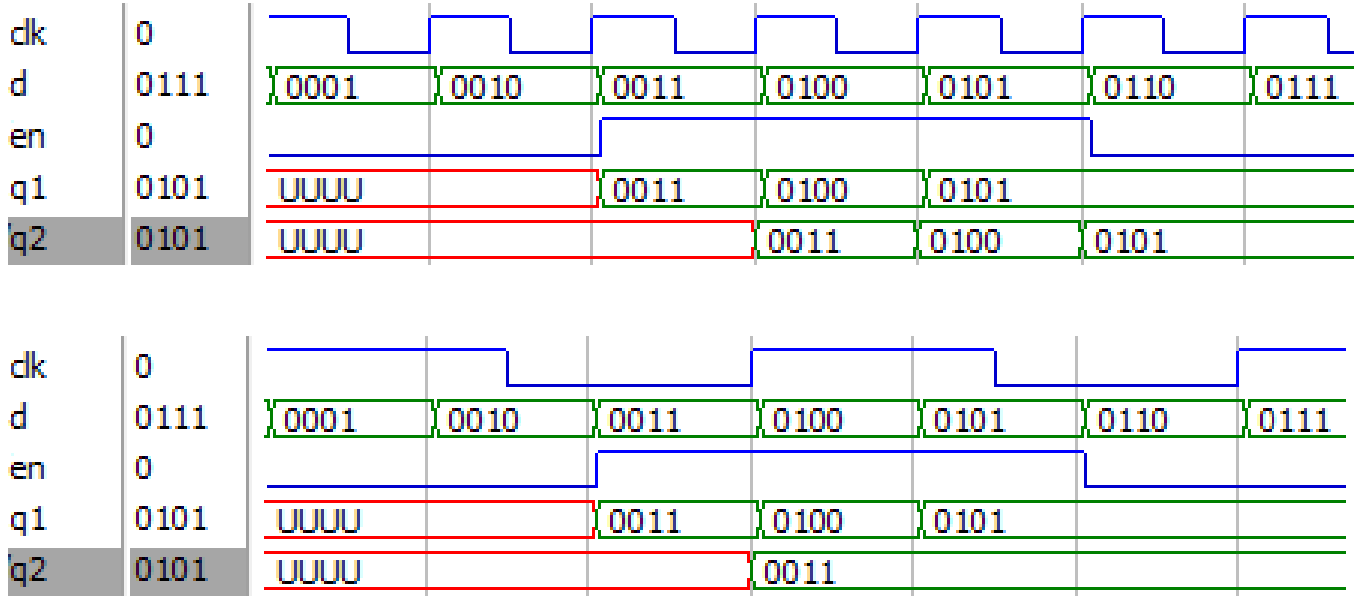
```
entity zapah
 d: in u4
 en: in u1
 q1, q2: out u4
begin
 if en=1 then
  q1 = d
  q2 <= d
 end
end
```



- flip-flop zakasni vhod za en cikel ure, zapah je transparenten

Zapah in register s flip-flopi

```
entity zapah
 d: in u4
 en: in u1
 q1, q2: out u4
begin
 if en=1 then
  q1 = d
  q2 <= d
 end
end
```



- flip-flop izloči motnje, če se podatek spreminja znotraj cikla ure

Opis sinhronih vezij

- ne želimo asinhronih pomnilnih elementov (latch)
- za sintezo flip-flopov uporabimo ustrezno obliko zapisa

I. oblika: samo ura

```
FF: process (clk)
begin
   if rising_edge(clk) then
      q <= d;
   end if;
end process;
```

```
q <= d
```

II. oblika: ura in asinhroni reset

```
FF: process (clk, reset)
begin
   if reset='1' then
      q <= "00000000";
   elsif rising_edge(clk) then
      q <= d;
   end if;
end process;
```

SHDL opisuje sinhrona vezja I. oblike, pri prevodu v VHDL lahko nastavimo tudi asinhroni reset

Sekvenčna vezja in povratne zanke

- sekvenčno vezje ima lahko povratno zanko
  - trenutno stanje registra določa kakšno bo novo stanje
- primer: sinhroni števec

I. oblika

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

```
st <= st + 1;
```

- na podlagi stanja izhoda izračunamo novo stanje
- če je signal st zunanji priključek, mora biti deklariran kot buffer

Sinhroni števec v jeziku VHDL

- signal st je hkrati vhod in izhod (deklaracija buffer) ali pa deklariramo notranji signal, ki ga priredimo izhodnemu signalu

```
entity stev is
 port ( clk : in std_logic;
        st  : buffer unsigned(3 downto 0) );
end stev;

architecture RTL of stev is
begin
 process(clk)
 begin
  if rising_edge(clk) then
    st <= st + 1;
  end if;
 end process;
end RTL;
```

```
entity stev is
 port ( clk : in std_logic;
        st : out unsigned(3 downto 0) );
end stev;

architecture RTL of stev is
 signal st_sig : unsigned(3 downto 0) := "0000";
begin
 process(clk)
 begin
  if rising_edge(clk) then
    st_sig <= st_sig + 1;
  end if;
 end process;

 st <= st_sig;

end RTL;
```

Števec z razponom od 0 do 9 (BCD)

```
bcd: process (clk)
begin
   if rising_edge(clk) then
      if q < 9 then
         q <= q + 1;
      else
         q <= "0000";
      end if;
   end if;
end process;
```

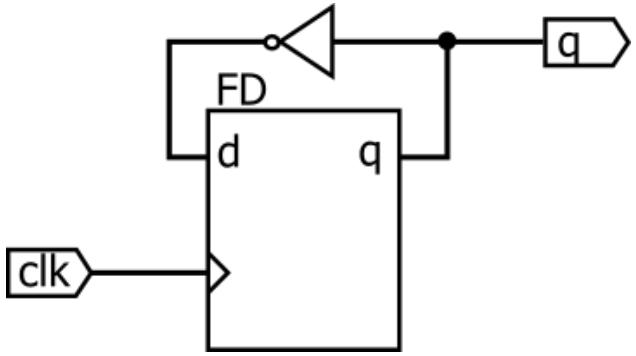
s sinhronim signalom reset

```
bcd: process (clk)
begin
 if rising_edge(clk) then
   if reset='1' then q <= "0000";
   else
      if q < 9 then
         q <= q + 1;
      else
         q <= "0000";
      end if;
   end if;
 end if;
end process;
```

```
if (reset) then q <= 0
else
 if q<9 then q <= q+1
 else q <= 0 end
end
```

Flip-flop z negatorjem

- sinhrono vezje s povratno zanko
  - opis negatorja
  - opis flip-flopa

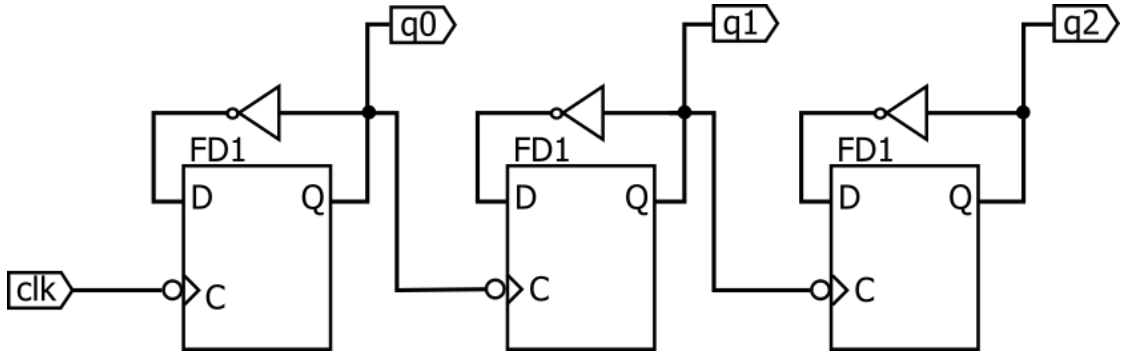



```
entity FFneg
 q: out u1;
 d: u1;
begin
 d = not q;
 q <= d;
end
```

- kompakten opis z enim prireditvenim stavkom:

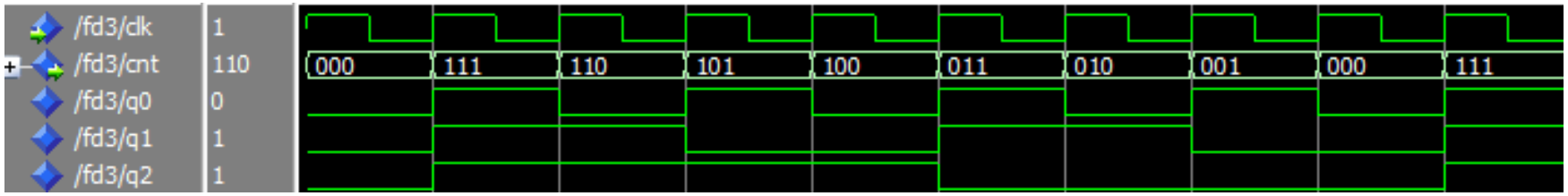
```
entity FFneg
 q: out u1;
begin
 q <= not q;
end
```

Serijski števec

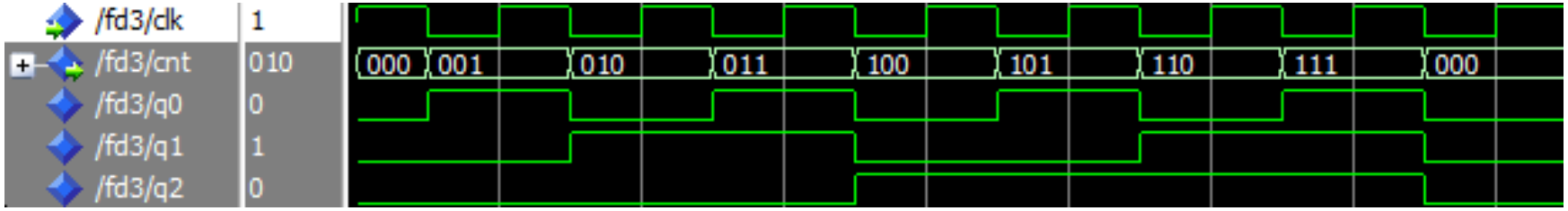


- npr. 3 zaporedni flip-flopi z negatorjem

  cnt = q2 & q1 & q0

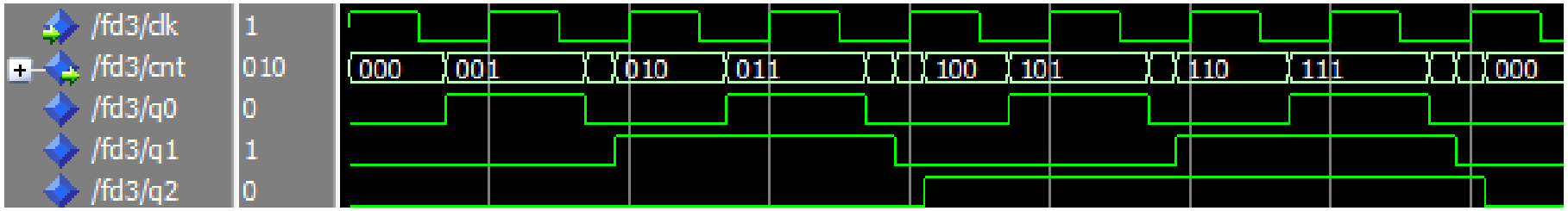


- flip-flopi proženi na zadnjo fronto ure



Serijski števec

- serijski števec ni sinhrono vezje (vsak flip-flop ima svojo uro)
- zakasnitve...



- če je zakasnitev števca zelo majhna v primerjavi s periodo ure bi bil tak števec uporaben, sicer pa raje naredimo sinhroni števec!

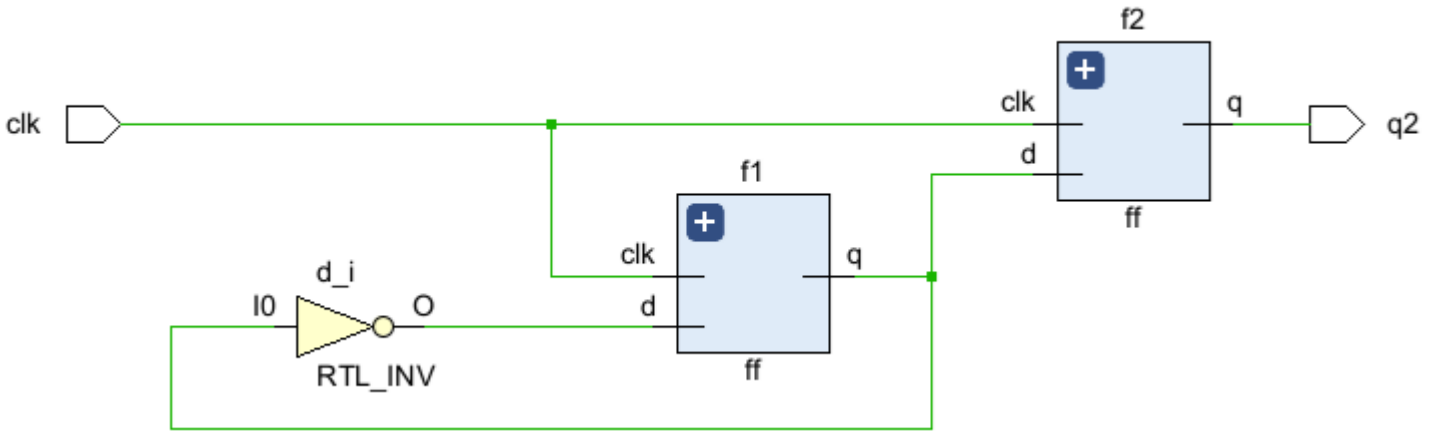
Strukturno načrtovanje s komponentami

- model vezja je sestavljen iz para entity-architecture
- posamezne modele povezujemo v večje vezje, tako da jih vključimo v opis vezja kot komponente
  - podobno risanju sheme vezja

Strukturno načrtovanje

- struktura (zgradba) vezja, ki temelji na povezavi komponent
- hierarhična struktura vezja
  - celotno vezje je sestavljeno iz komponent, ki so zgrajene iz enostavnejših komponent...

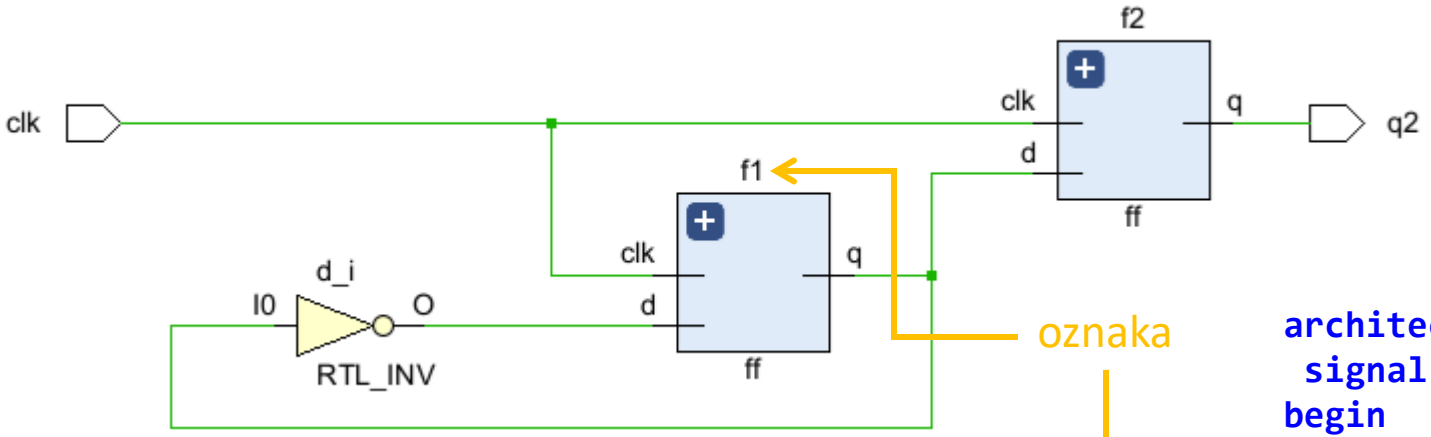
Model vezja s komponentami



- komponenta je opisana s samostojnim modelom, ki je preveden v skupno knjižnico
- v vezje lahko vključimo poljubno število komponent z enakim modelom

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

entity ff is
 port (
   clk : in std_logic;
   d : in std_logic;
   q : out std_logic );
end ff;

architecture RTL of ff is
begin
 process(clk)
 begin
  if rising_edge(clk) then
   q <= d;
  end if;
 end process;
end RTL;
```

Model vezja s komponentami



- stavek port map opiše instanco komponente
- določi unikatno oznako, ime in povezave

oznaka

```
architecture strukt of vezje is
 signal d,q,q1 : std_logic;
begin
 d <= not q;

 f1: entity work.ff port map(clk=>clk, d=>d, q=>q);

 f2: entity work.ff port map(clk=>clk, d=>q, q=>q2);

end strukt;
```

knjižnica in
ime komponente

povezave

Deklaracija komponente

- če ni navedena knjižnica, mora biti posamezna vrsta komponente deklarirana

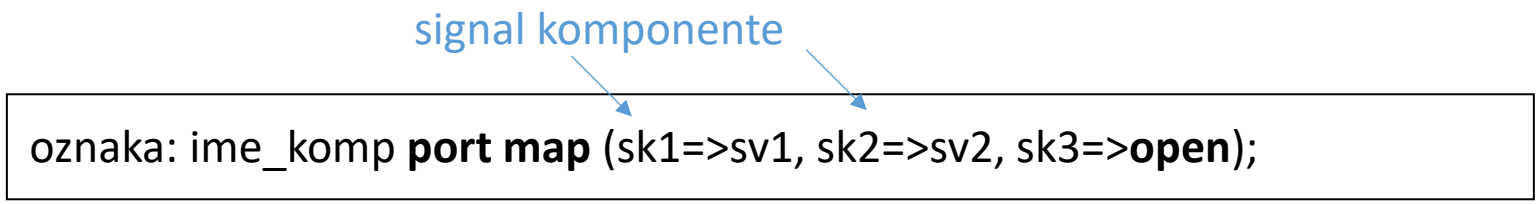
deklaracija

```
architecture strukt of vezje is
 component ff is
  port (
    clk : in std_logic;
    d : in std_logic;
    q : out std_logic );
 end component;

 signal d,q,q1 : std_logic;
begin
 d <= not q;

 f1: ff port map(clk=>clk, d=>d, q=>q);

 f2: ff port map(clk=>clk, d=>q, q=>q2);

end strukt;
```

samo ime komponente
brez "entity" in knjižnice

Pravila povezovanja

- priključke komponente vežemo na notranje ali zunanje signale vezja
  - omejitve: out povezan le na out, in le na in, buffer le na buffer ...



- vsi vhodni signali morajo biti povezani
- kakšen izhod je lahko nepovezan (open)
- pri krajšem zapisu izpustimo signale komponente (vrstni red je pomemben!)

```
f1: ff port map(clk, d, q);

f2: ff port map(clk, q, q2);
```

Povzetek

- modeli logičnega vezja
- strojno-opisni jeziki
- signali in priključki v SHDL/VHDL
- funkcijski opis: operacije, prireditev
- postopkovni opis in pogojni stavek
- opis sekvenčnih vezij
- povratna zanka (števci)
- strukturno načrtovanje