Laboratorij za načrtovanje integriranih vezij

Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*

Digitalni Elektronski Sistemi

Osnove jezika VHDL

1. del: signali, izrazi in osnovni stavki

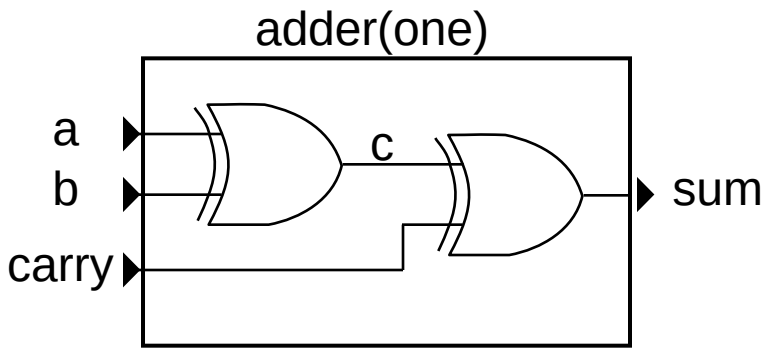
Osnovni koncepti jezika VHDL

- Model vezja je sestavljen iz:
  - opisa vmesnika (entity)
  - 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;
```

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

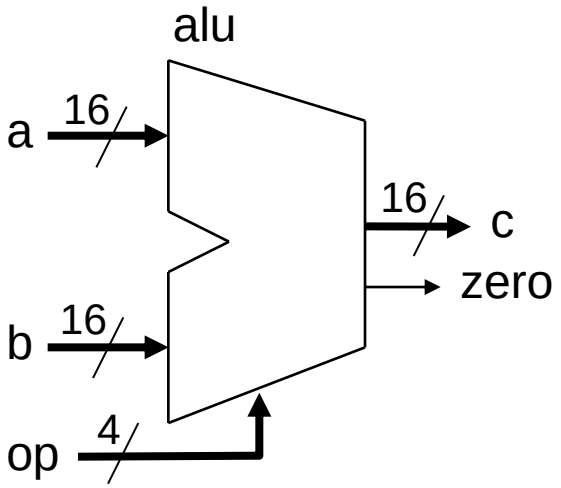
- V arhitekturnem stavku deklariramo notranje in izhodne signale

Zunanji signali

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

MSB LSB

```
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;
```



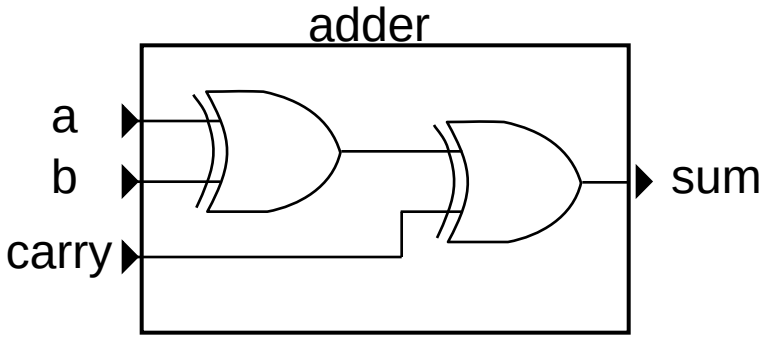
- Podatkovni tip bit pozna dve vrednosti: 0 ali 1
  - za bolj realistično simulacijo potrebujemo več-vrednostno logiko ('Z', 'U', 'X')
  - potrebujemo tudi funkcije za razrešitev
    - npr. ko signalu vsilimo vrednost '0' in 'Z', prevlada vrednost '0'

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', '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

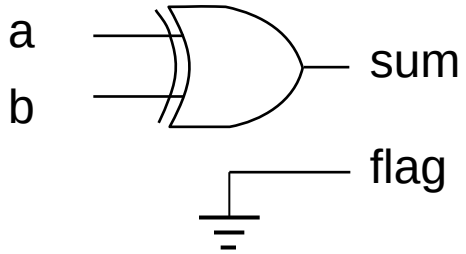
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

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!

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
```

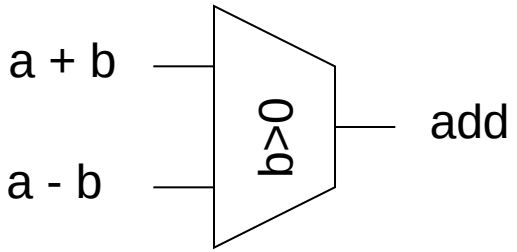
- Logične operacije: **and, or, not, xor, xnor...**
- Sestavljanje vektorjev: **&**

```
a <= high & low;
b <= sign & “000” & low;
```

Pogojni prireditveni stavek

- 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

Deklaracija zunanjih signalov

- Določimo podatkovni tip in smer (mode)

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 unsigned(3 downto 0);
         incr_a : buffer unsigned(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)

Procesno okolje

- 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;
```

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
  - pri opisu kombinacijskih vezij moramo navesti vse signale, ki predstavljajo vhode v proces

Pogojni stavek (if)

- Ob izpolnjenem pogoju se izvrši en ali več stavkov, ki so zapisani za "then"
- Opcija: če pogoj ni izpolnjen se izvršijo stavki za "else"
- Pogojni stavek zaključimo z "end if "

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

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

Primerjava s sočasnimi stavki

- Pogojni stavek spada med sekvenčne stavke
  - uporabljamo ga lahko le znotraj procesa
- Pogojni prireditveni stavek je alterativa med sočasnimi stavki

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

=

```
max <= a when a>b else b;
```

Primerjava (2)

```
bin2bcd: process(bin)
begin
  if bin>9 then
     enice <= bin – “1010”;
     desetice <= ‘1’;
  else
     enice <= bin;
     desetice <= ‘0’;
  end if;
end process;
```

=

```
enice <= bin - ”1010”  when bin>9
                 else bin;

desetice <= ‘1’ when bin>9
                      else ‘0’;
```

- V pogojnem stavku imamo lahko več prireditev

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;
```

=

```
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 ...

- Zaporedni pogoji se ovrednotijo s prioriteto

Sekvenčni izbirni stavek (case)

- Odločamo se glede na vrednost enega signala
- Izbirni stavek nima prioritete

```
case ime_signala is
    when vrednost1 =>
       stavki(1);
    when vrednost2 =>
       stavki(2);
    ...
    when others  =>
       stavki(n);
  end case;
```

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

Primer: izbiralnik (multipleksor)

- Izbiralnik s štirimi vhodi (a, b, c in d)

```
m: process(mode,a,b,c,d)
begin
  if mode="00" then
     mux <= a;
  elsif mode="01" then
     mux <= b;
  elsif mode="10" then
     mux <= c;
  else
     mux <= d;
  end if;
end process;
```

```
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;
```

Sočasni izbirni stavek (with…select)

- Na podlagi izbire priredimo signalu različne vrednosti (ali izraze)

```
with izbira select
signal <= izraz(1) when vrednost1,
          izraz(2) when vrednost1,
          ...
          izraz(n) when others;
```

- Za razliko od case stavka lahko prirejamo vrednost le enemu signalu

```
with digit select
display <= “00111111” when “00”,
           “00000110”  when “01”,
           “11111001”  when others;
```

Sekvenčni stavki in sinteza vezja

- Prirejanje vrednosti signalom na več mestih
- Če signalu pod kakšnim pogojem ne določimo vrednosti, se bo ohranjala zadnja vrednost
  - dobimo sekvenčno vezje !

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

Naredili smo asinhroni zapah za signal flag !

Opis kombinacijskih vezij

- Definiramo vrednosti pri vseh pogojih:
  - vrednost priredimo pri vsakem "if" in "else" ali
  - vrednost priredimo na začetku procesa in jo spremenimo v "if" stavkih

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

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