



Laboratorij za načrtovanje integriranih vezij

Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*



Digitalni Elektronski Sistemi

Osnove jezika VHDL

komponente in testne strukture

Struktura vezja 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...

Deklaracija komponent

- ▶ Komponente deklariramo na začetku arhitekturnega stavka (pred **begin**)
- ▶ Navedemo ime komponente kot je zapisano v stavku **entity** in zunanje priključke (stavek **port**)
- ▶ Primer: v vezje sek bomo vključili register **reg**

architecture struktura of **sek** is

component **reg** is

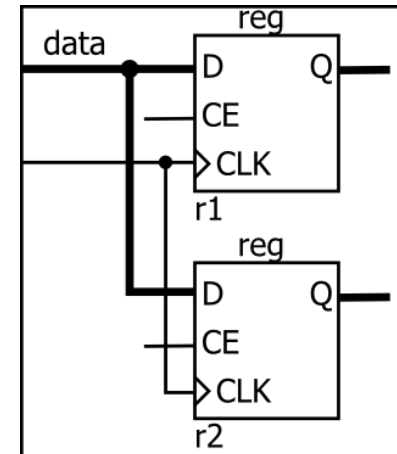
```
port ( d : in std_logic_vector(7 downto 0);
```

```
      clk, ce : in std_logic;
```

```
      q : out std_logic_vector(7 downto 0) );
```

```
end component;
```

```
begin
```



Povezovanje komponente

- ▶ Komponento povežemo s stavkom **port map**
 - ▶ vsaki komponenti damo enolično oznako
 - ▶ v oklepaju navedemo povezave (**signal komp.** => **signal vezja**)

begin

```
r1: reg port map (d=>data, clk=>clk, ce=>en1, q=>data_reg);
```

```
r2: reg port map (d=>data, clk=>clk, ce=>en2, q=>op_reg);
```

...

end struktura;

- ▶ **Krajši zapis:**

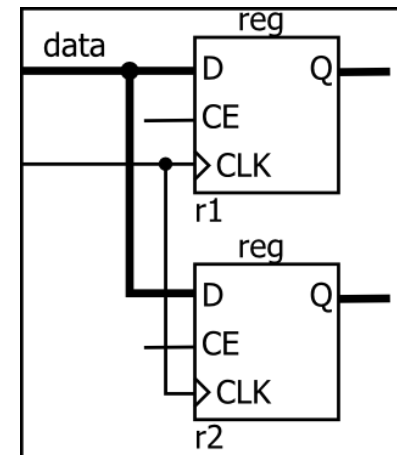
begin

```
r1: reg port map (data, clk, en1, data_reg);
```

```
r2: reg port map (data, clk, en2, op_reg);
```

...

end struktura;



Pravila povezovanja komponent

- ▶ Signale komponente povezujemo z notranjimi in zunanji signali vezja
 - ▶ omejitve pri zunanjih signalih: signal **out** je lahko vezan le na **out**, **buffer** le na **buffer** (**zato so priključki buffer nezaželeni!**)
- ▶ Povezati moramo najmanj vse vhodne signale (**in, inout**)
 - ▶ običajno povežemo vse signale, če kakšen izhod ne potrebujemo zapišemo rezervirano besedo **open**
 - ▶ pri krajšem zapisu je vrstni red pomemben

```
oznaka: ime_komp port map (sk1=>sv1, sk2=>sv2, sk3=>open);
```

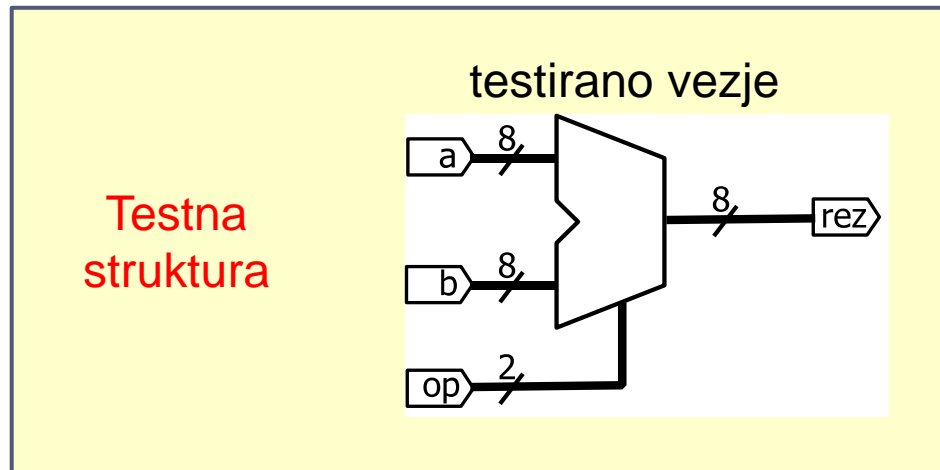
Testiranje vezij

- ▶ Testiranje strojne opreme na delovni mizi (**work bench**)
 - ▶ inštrumenti: multimeter, logični analizator, osciloskop...
- ▶ Testiranje na računalniku – simulacija

1. Izvajanje simulacije po korakih
 - ▶ nastavimo vhode, izvedemo korak, spremenimo vhode...
 - ▶ omejene možnosti simulacije, za preprosta vezja
 - ▶ **simulator v praksi nikoli ne poženemo le enkrat !**
2. Programska testna miza (testna struktura, **test bench**)
 - ▶ v testni strukturi določimo časovni potek vhodov v vezje
 - ▶ naredimo model okolice testiranega vezja
 - ▶ **ko je testna struktura narejena, jo večkrat uporabimo za izvedbo simulacije vezja**

Testna struktura

- ▶ Testna struktura je orodje, ki pomaga načrtovalcu vezja ugotavljati pravilnost delovanja vezja
- ▶ Testno strukturo povežemo s testiranim vezjem in izvedemo simulacijo



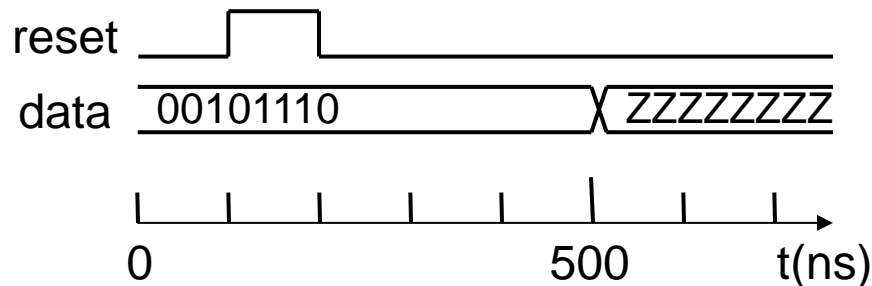
Testna struktura v jeziku VHDL

- ▶ **Veže (entity – architecture)**, ki nima zunanjih priključkov
 - ▶ vse vrednosti signalov definiramo znotraj strukture
 - ▶ testirano veže vključimo kot komponento
- ▶ **V arhitekturi definiramo časovno spreminjanje vhodnih signalov testiranega vezja**
 - ▶ uporabimo konstrukte jezika VHDL namenjene simulaciji, saj testne strukture ne bomo sintetizirali
 - ▶ **zakasnitve signalov**
 - ▶ **stavki za izpis poročil med simulacijo**
 - ▶ **funkcije za branje in zapis datotek**

Časovno spreminjanje signalov

- ▶ Signalu priredimo več dogodkov, ki so vezani na čas:

```
reset <= '0', '1' after 100 ns, '0' after 200 ns;  
data <= "00101110", "ZZZZZZZZ" after 500 ns;
```



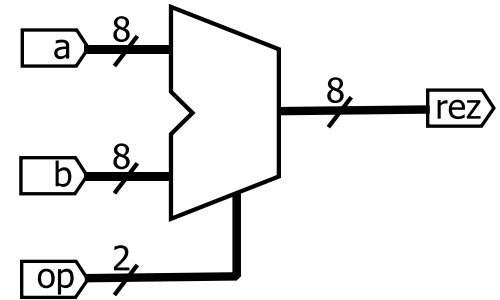
- ▶ Kadar je v testu veliko dogodkov uporabimo proces
 - ▶ prirejanje dogodkov le v najbolj preprostih primerih, npr. za enkratne dogodke kot je reset

Primer: test ALE

► Deklaracija testirane komponente in notranjih signalov

```
architecture behavior of test is
  component ale is
    port ( a, b : in std_logic_vector(7 downto 0);
          op : in std_logic_vector(1 downto 0);
          rez : out std_logic_vector(7 downto 0) );
  end component;

  signal a,b: std_logic_vector(7 downto 0);
  signal op: std_logic_vector(1 downto 0);
  signal rez: std_logic_vector(7 downto 0);
begin
```



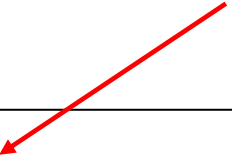
► Povežemo komponento

```
UUT: ALE port map (a=>a, b=>b, op=>op, rez=>rez);
```

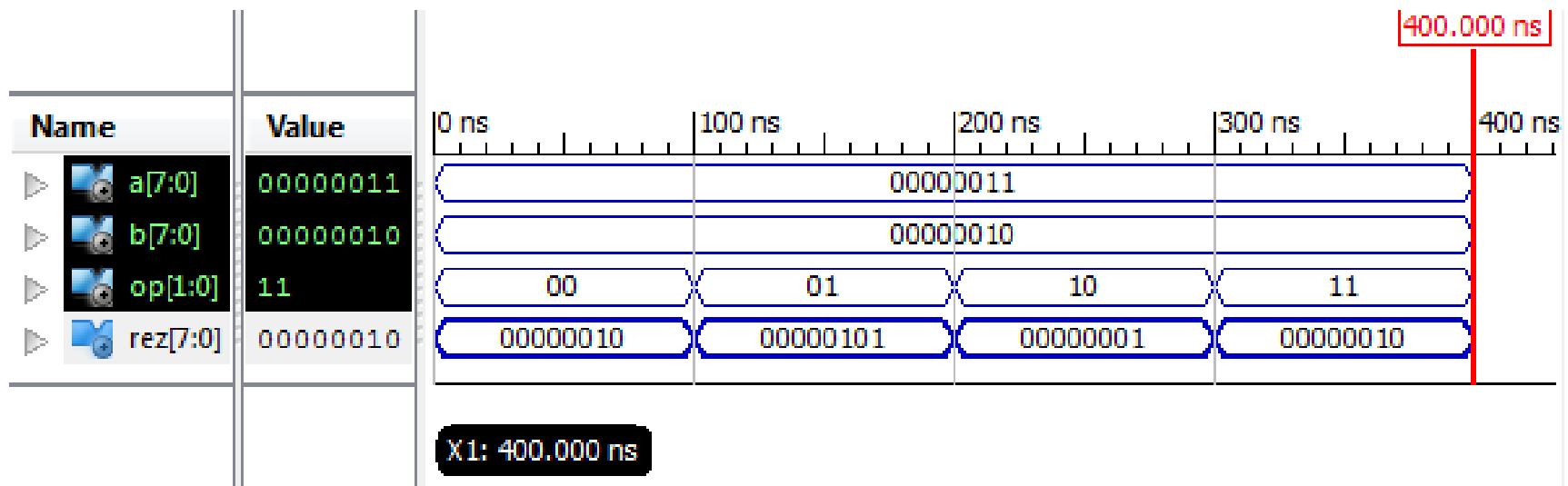
Test ALE (2): nastavljanje vhodov

- ▶ Napišemo proces v katerem spreminjamo vhode
 - ▶ časovni potek signalov določajo stavki **wait for**
 - ▶ uporabimo proces brez seznama signalov

```
stim_proc : process
begin
  a <= "00000011";
  b <= "00000010";
  op <= "00";      -- operacija AND
  wait for 100 ns;
  op <= "01";      -- vsota
  wait for 100 ns;
  op <= "10";      -- razlika
  wait for 100 ns;
  op <= "11";      -- odstej 1
  wait;           -- zamrzni proces, da se ne ponavlja
end process;
```



Test ALE(3): rezultat simulacije



- ▶ Ali lahko testna struktura samodejno preverja vezje?
 - ▶ Da, npr. preverimo ali je vsota takšna, kot jo pričakujemo:

```
op <= "01";    -- vsota
wait for 100 ns;
assert rez = 5 report "Napaka, vsota ni enaka 5 !";
...
```

Testni vektor

- ▶ Testni vektor je množica vrednosti vhodnih signalov
 - ▶ v praksi ne moremo uporabiti vseh možnih vrednosti
 - ▶ uporabimo nekaj tipičnih vrednosti, mejne vrednosti...
- ▶ Vrednosti testnega vektorja lahko zapišemo v zbirko

```
type vzorci is array (natural range <>) of
    std_logic_vector(7 downto 0);           -- deklaracija zbirke
constant data_a : vzorci := ("00000000", "00000011", "11111111");
...
process
begin
    for i in data_a'range loop -- za vse vrednosti testnega vektorja
        a <= data_a(i);
        b <= data_b(i);
        wait for 100 ns;
    end loop;
    wait;
end process;
```

Testiranje sekvenčnih vezij

- ▶ Za generiranje ure uporabimo procesno okolje brez seznama signalov in stavke **wait for**
 - ▶ proces se izvaja, dokler ga ne ustavimo z **wait**

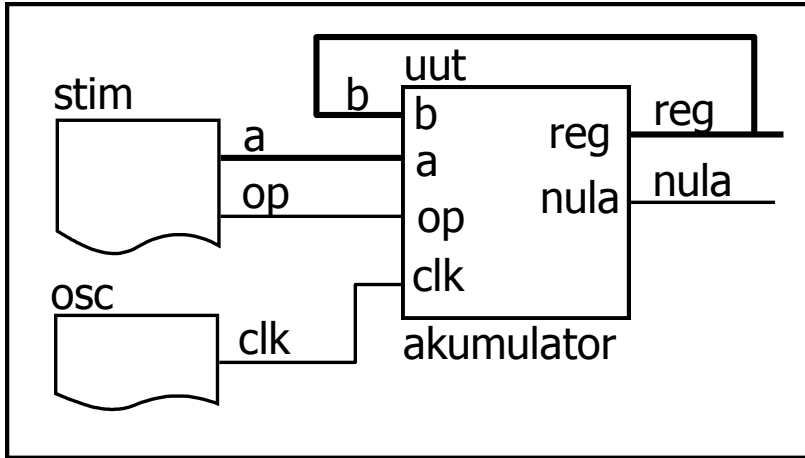
```
GEN_URE: process
begin
  clk <= '0';
  wait for 500 ns;
  clk <= '1';
  wait for 500 ns;
end process;
```

ustavi <= '0', '1' **after** 10 us;

```
GEN_URE: process
begin
  if ustavi='0' then
    clk <= '0';
    wait for 500 ns;
    clk <= '1';
    wait for 500 ns;
  else wait;
end if;
end process;
```

Primer: testiranje akumulatorja

akumulator_tb



osc: **process**

begin

if ustavi='0' **then**

clk <= '0'; **wait for** 500 ns;

clk <= '1'; **wait for** 500 ns;

else wait;

end if;

end process;

stim: **process**

begin

op <= '0'; a <= "0000";

wait for 1 us;

a <= "1010";

wait for 1 us;

op <= '1'; a <= "0001";

wait;

end process;