Univerza v Ljubljani
Fakulteta
za računalništvo
in informatiko

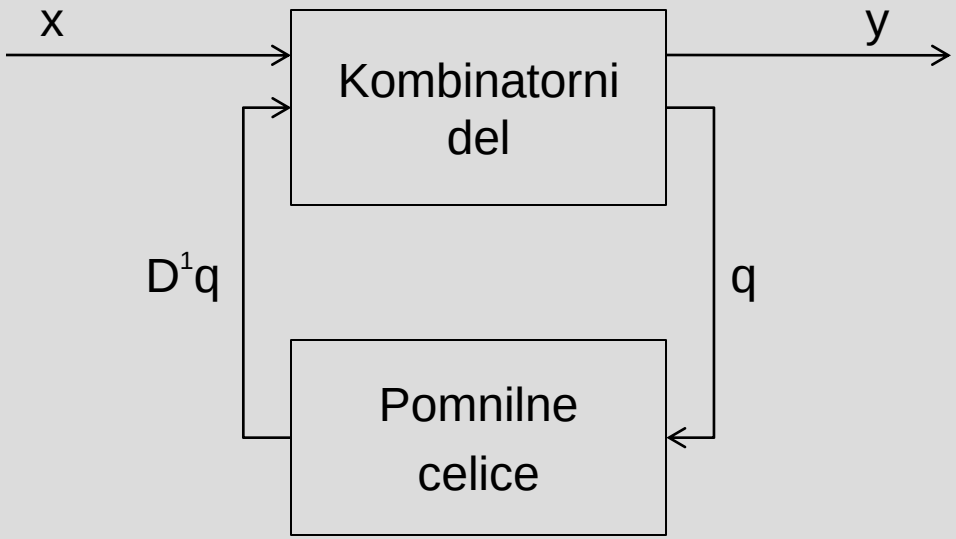
Metode logičnega snovanja

Sekvenčna vezja v jeziku VHDL

Miha Moškon

Sekvenčna vezja

Sekvenčna vezja so **časovno odvisna** vezja; sestavljena so iz **kombinatornega dela** in iz **pomnilnih celic:**



Sekvenčna vezja

Večinoma se uporabljajo **sinhronska** sekvenčna vezja – stanja pomnilnih celic se spreminjajo ob **pozitivni** (**negativni**) **fronti urinega signala**.

V GAL 16V8 so uporabljene D celice.

Enačba D celice: $D^1q = d$

Zaporedni stavki

Sinhrona sekvenčna vezja v jeziku VHDL vedno podajamo z **zaporednimi stavki**.

Zaporedni stavki se na videz izvajajo **zaporedno**, združujemo jih v **procese**.

Procesi

Znotraj arhitekture definiramo procese:

```
[ime_procesa:] process (prožilni_signal1, ..., 
  prožilni_signalN)
  begin
       -- zaporedni stavki
  end process [ime_procesa];
```

Posamezen proces se izvede samo, če se spremeni kakšen od prožilnih signalov v **sensitivity listi**.

Procesi

Ob spremembi vsaj enega signala v sensitivity listi, se proces sproži. Pri prireditvah signalov v procesu, se upoštevajo vrednosti, ki so jih imeli signali na desni strani prireditvenega stavka **pred izvedbo** procesa.

Vzporedno se lahko izvaja več procesov naenkrat (isti signal lahko spreminjamo **le v enem** od vzporednih procesov).

Zaporedni stavki (sintaksa)

- prireditev:

```
izhod1 <= vhod1;
```

- `if-then-else` stavek:

```
if pogoj1 then stavki1;
elsif pogoj2 then stavki2;
...
else stavkiN;
end if;
```

OPOZORILO: `if` stavek mora vedno imeti tudi `else` vejo, razen če želimo realizirati flip-flop (latch).

Zaporedni stavki (sintaksa)

- `case` stavek:

```
case vhod is
   when vrednost1 =>
          stavki1;
   when vrednost2 =>
          stavki2;
          ...
   when vrednostN =>
          stavkiN;
end case;
```

Proženje procesov

Stanje sinhronega sekvenčnega vezja se ponavadi spremeni ob **prehodu urinega signala**.

Zaznavanje urine fronte znotraj procesa:

- z atributom `'event`:

  primer proženja ob pozitivni fronti:

  ```
  ime_ure'event AND ime_ure = '1'
  ```

- s funkcijo `rising_edge(), falling_edge()`:

  primer proženja ob pozitivni fronti:

  ```
  rising_edge(ime_ure);
  ```

Zaznavanje urine fronte

- z atributom `'event`:

```
process (clk_in)
begin
  if clk_in'event and clk_in = '1' then -- pozitivna fronta
     ...
  end if;
end process;
```

- s funkcijo `rising_edge(), falling_edge()`:

```
process (clk_in)
begin
  if rising_edge(clk_in)then -- pozitivna fronta
     ...
  end if;
end process;
```

Implementacija Reset signala

- **Sinhroni**: aktivnost Reset signala preverjamo samo znotraj procesa:

```
process (clk_in)
begin
  if rising_edge(clk_in) then -- pozitivna fronta
      if rst_in = '1' then
       ...
      else
       ...
      end if;
  end if;
end process;
```

Implementacija Reset signala

- **Asinhroni**: Reset signal vključimo v **sensitivity listo**. Njegovo aktivnost preverjamo pred preverjanjem fronte:

```
process (clk_in, rst_in)
begin
  if rst_in = '1' then
     ...
  elsif rising_edge(clk_in) then -- pozitivna fronta
     ...
  end if;
end process;
```

Primer implementacije T celice

```
-- knjiznica, ki jo uporabljamo
library IEEE;
use IEEE.STD_LOGIC_1164.all;

-- entiteta
entity T_cell is
   port (t_in   : in std_logic; -- t vhod
         clk_in  : in std_logic; -- urin signal
         rst_in  : in std_logic; -- reset signal
         q_out   : out std_logic); -- izhod iz T celice
end T_cell;
```

Primer implementacije T celice

```
architecture behavioral of T_cell is
   signal q : std_logic; -- pomozni signal
begin
   process (clk_in, rst_in) -- asinhroni rst
   begin
        if (rst_in = '1') then
                q <= '0';
        elsif (rising_edge(clk_in)) -- ob pozitivni fronti
                q <= (NOT t_in AND q) OR
                        (t_in AND NOT q); -- enacba T celice
        end if;
   end process;
   q_out <= q;
end behavioral;
```

Naloga

1. Implementirajte JK pomnilni celici, ki imata sinhrona RESET in SET (skupna). RESET naj ima prednost pred SET. Prvo celico opišite z logičnimi enačbami, drugo pa s stavkom `if-then-else`.
2. Pomnilnima celicama dodajte vhodni signal enable. Ko je enable enak 0, naj celica deluje normano, ko pa je enak 1 naj se njen izhod postavi v visoko impendančno stanje (‘Z’).

Naloga

3. Realizirajte pomnilno celico, ki deluje v skladu s sledečo pravilnostno tabelo in asinhronim resetom:

A	B	C	Q(t+1)
0	0	0	Q
0	0	1	!Q
0	1	0	1
0	1	1	0
1	0	0	Q
1	0	1	!Q
1	1	0	1
1	1	1	0