Laboratorij za načrtovanje integriranih vezij

Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*

Digitalni Elektronski Sistemi

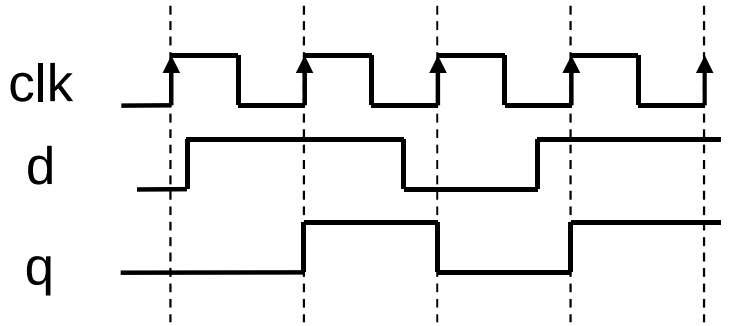
# Osnove jezika VHDL

## 3. del: sekvenčna vezja

# Opis sekvenčnih vezij

- Sinhrona sekvenčna vezja spreminjajo izhode ob fronti ure
  - prva fronta: clk'event and clk='1' ali rising_edge(clk)
  - zadnja: clk'event and clk='0' ali falling_edge(clk)

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




# Sekvenčni elementi in sinteza vezja

- Program za sintezo zna narediti register ali flip-flop, če uporabimo ustrezno obliko opisa vezja

## I. oblika: popolnoma sinhrono vezje

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

## II. oblika: asinhroni reset

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

# Povratne zanke

- Znotraj sinhronega procesa lahko uporabljamo povratno zanko
  - na podlagi stanja registra izračunamo novo stanje
  - Npr. števec, pomikalni register, sinhroni avtomat

## I. oblika

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

- signal q uporabljamo v povratni zanki
- če je q zunanji signal, mora biti buffer

# Primer: BCD števec

- BCD števec šteje od 0 do 9

## I. oblika

```
stev: process (clk)
begin
   if rising_edge(clk) then
      if q < 9 then
         q <= q + ‘1’;
      else
         q <= “0000”;
      end if;
   end if;
end process;
```

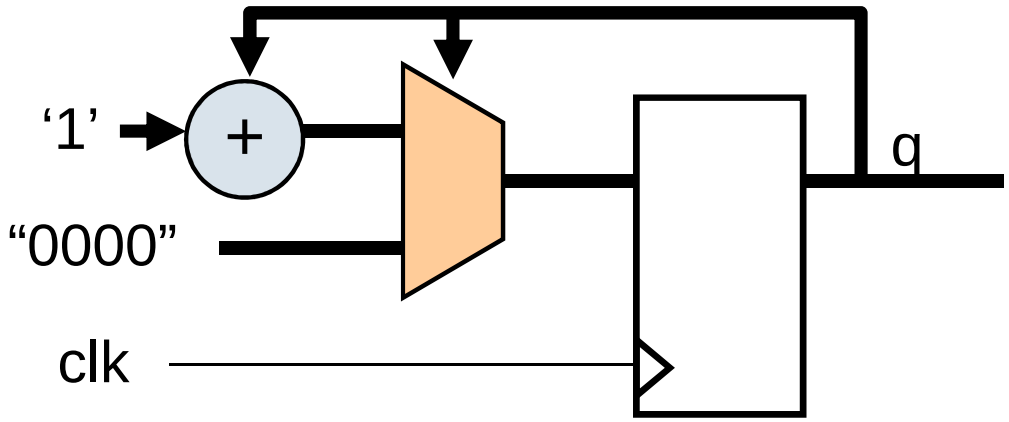
## II. oblika

```
stev: process (clk)
begin
 if reset=‘1’ then
       q <= “0000”
 elsif rising_edge(clk) then
    if q < 9 then
       q <= q + ‘1’;
    else
       q <= “0000”;
    end if;
 end if;
end process;
```

# Zgradba vezja BCD števca

- Signal q je register, ki ima na vhodu logiko
  - povratna zanka je med izhodom registra in logiko

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




# Registri in flip-flopi

- Prireditev vrednosti znotraj pogoja za fronto ure pomeni, da je signal register ali flip-flop

I. oblika

4 bitni register

flip-flop

```
acc: process (clk)
begin
   if rising_edge(clk) then
      a <= a + data;
      if a="1111" then
         full <= '1';
      else
         full <= '0';
      end if;
   end if;
end process;
```

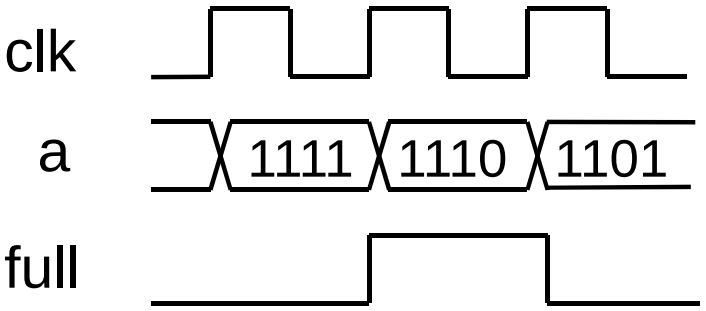
# Potek simulacije

- Procesni stavek se izvede ob spremembi clk
  - pogoj rising_edge() je izpolnjen, ko gre clk iz 0 na 1

```
acc: process (clk)
begin
   if rising_edge(clk) then
      a <= a + data;
      if a="1111" then
         full <= '1';
      else
         full <= '0';
      end if;
   end if;
end process;
```

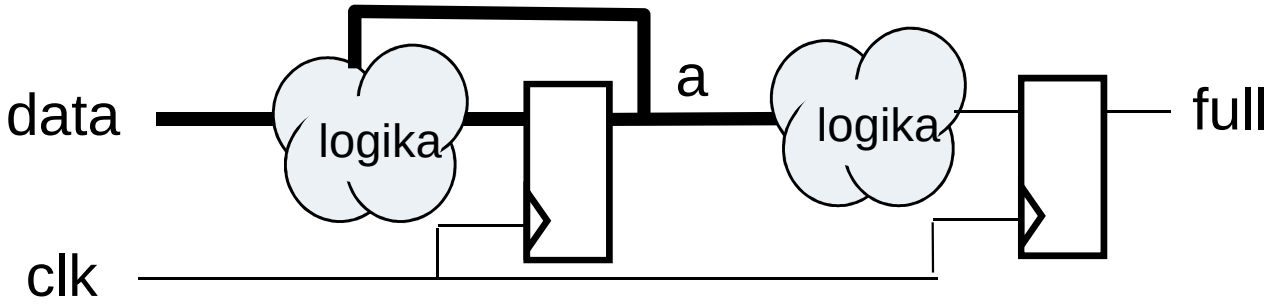
| **korak** | **čas** | clk | data | a | full |
|---|---|---|---|---|---|
| čakaj | 0 | 0, 1 | 1111 | 0000 | 0 |
| račun | T | 1 | 1111 | 1111 | 0 |
| izvrši | T+d | 1 | 1111 | 1111 | 0 |
| čakaj | T2 | 0, 1 | 1111 | 1111 | 0 |
| račun | T2+d | 1 | 1111 | 1110 | 1 |
| izvrši | T2+d | 1 | 1111 | 1110 | 1 |

# Časovni diagram in zgradba vezja

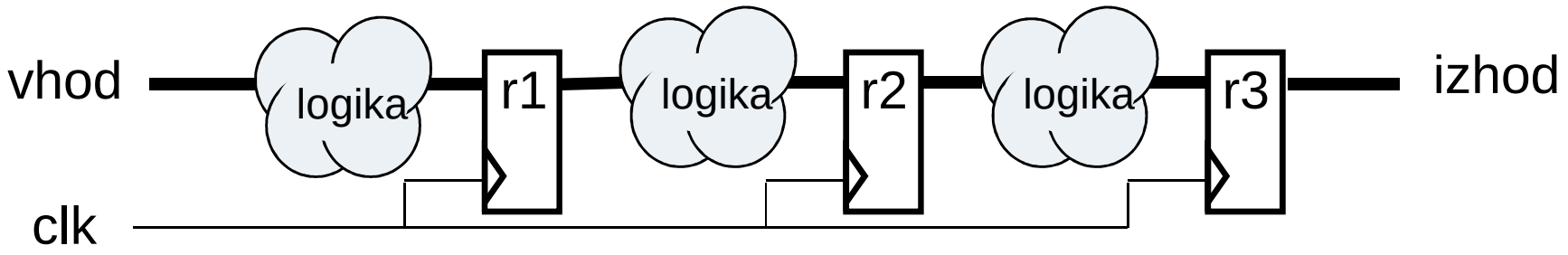



- Signal full se postavi na ‘1’ šele ob naslednji fronti ure
  - full je izhod iz flip-flopa !

- Vezje je sestavljeno iz registra (a), flip-flopa (full) in logike na vhodih obeh sekvenčnih elementov

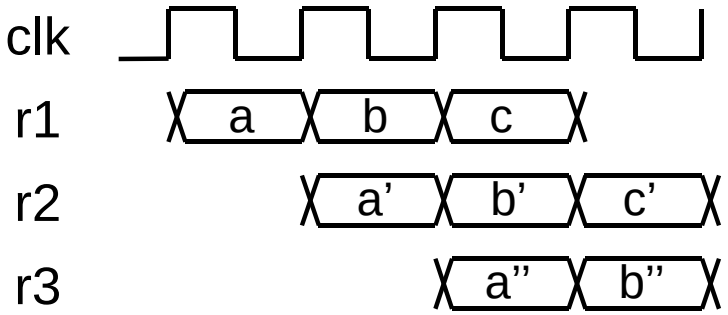


# Opis cevovodnih vezij




```
pipe: process (clk)
begin
   if rising_edge(clk) then
      r1 <= vhod - “0011”;
      if r1>10 then  r2 <= r1;
      else r2 <= “1010”;
      end if;
      r3 <= r2 xor “1010”;
   end if;
end process;
```

- Pri cevovodnih vezjih se rezultati operacij sproti shranijo v registre




# Signali v sinhronih vezjih

- S signali poimenujemo povezave v vezju
- Signali pri simulaciji:
  - ob izračunu izrazov se določi bodoča vrednost signala (dogodek)
  - vrednost se priredi, šele ko se dogodki izvršijo
- Signali pri sintezi:
  - v sinhronih procesih (I. ali II. oblika) vsak signal, ki mu prirejamo vrednost pomeni register ali flip-flop
  - signali niso primerni za izračun vmesnih rezultatov s kombinacijsko logiko

# Spremenljivke (variable)

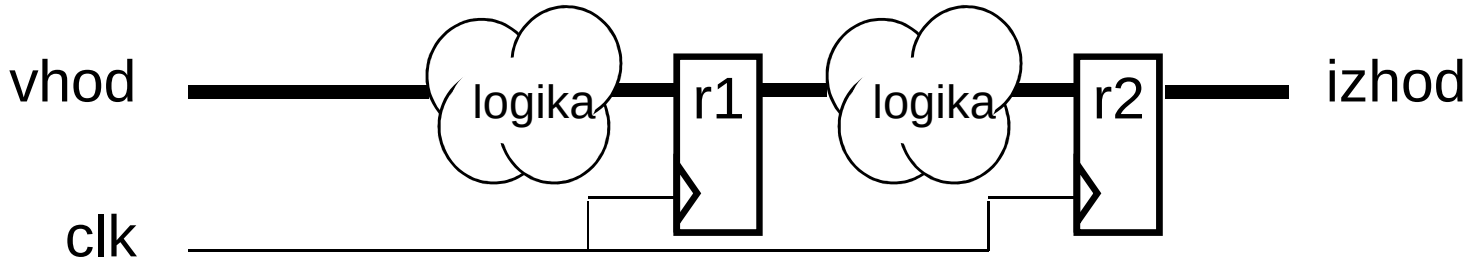
- Spremenljivke uporabljamo v procesnem okolju za izračun vmesnih rezultatov
  - ob izračunu spremenljivka takoj dobi novo vrednost
- Deklaracija spremenljivke:

```
p: process (clk)
  variable v: std_logic_vector(7 downto 0);
begin
```

- Prireditveni operator (:=)

```
v := vhod - “0011”;
```

# Sinteza vezja s spremenljivkami




```
pipe: process (clk)
  variable v: std_logic_vector (3 downto 0);
begin
   if rising_edge(clk) then
      v := vhod - “0011”;
      if v>10 then
         r1 <= v;
      else
         r1 <= “1010”;
      end if;
      r2 <= r1 xor “1010”;
   end if;
end process;
```

- spremenljivka ne predstavlja model povezave
- spremenljivka ne predstavlja registra

# Zmogljivost vezja

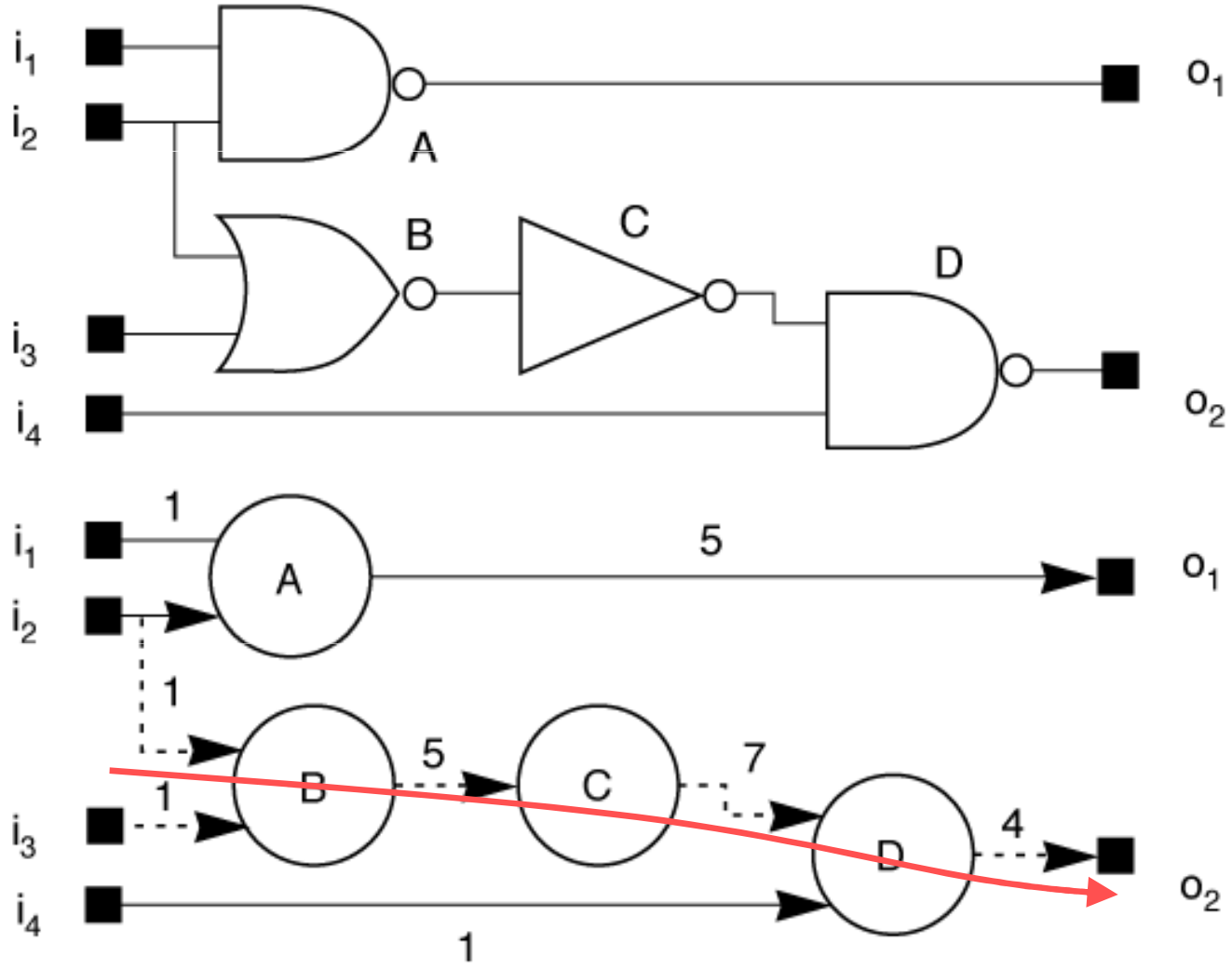
- zmogljivost vezja omejujejo zakasnitve
- kombinacijske zakasnitve
  - od spremembe na vhodu do spremembe izhodov
  - poiščemo kritično pot in jo poskusimo skrajšati
    - lahko je več enakovrednih kritičnih poti
- ASIC: prevladujejo zakasnitve v logiki
  - zmanjšamo s topologijo vrat, obremenitvijo in načrtovanjem transistorjev
- FPGA / CPLD: velike zakasnitve v povezavah
  - izbira kratkih in namenskih povezav

# Poti in zakasnitve

- kombinacijske zakasnitve merimo na poteh v vezju

vezje

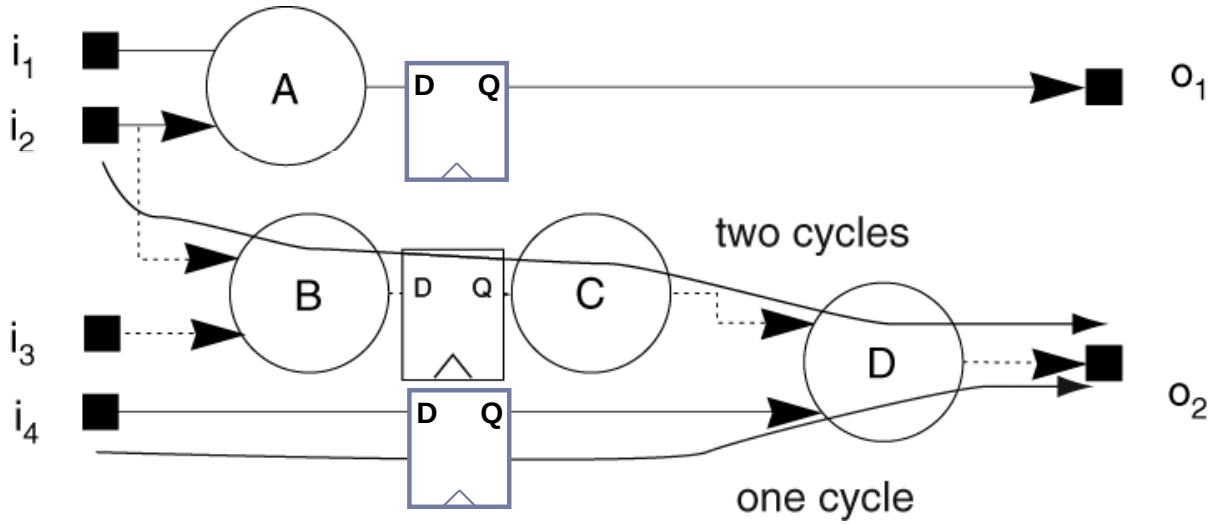
graf




- največja zakasnitev je na kritični poti

# Optimizacija sekvenčnega vezja

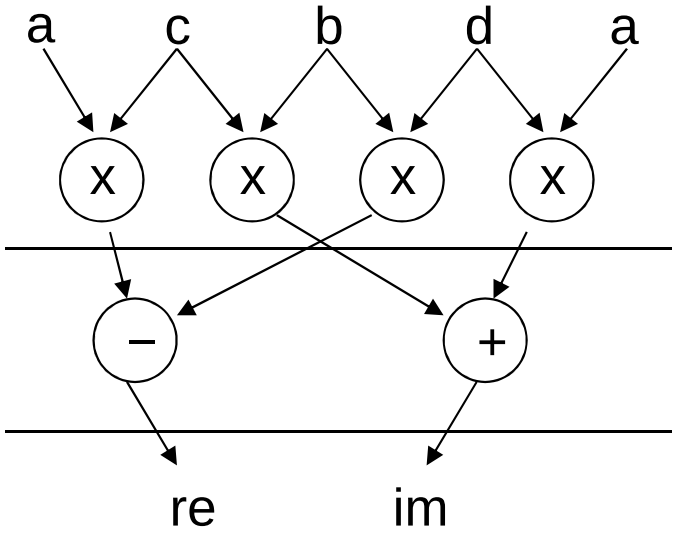
- Cevljenje (pipelining) z dodajanjem registrov
  - L : latenca (zakasnitev med vh. in izh.)
  - T : pretok (frekvenca obdelave podatkov)
- brez cevljenja
  - $L = D$
  - $T = 1 / D$
- dvostopenjski cevovod
  - $L = D + t_{FF}$
  - $T \leq 2 / D$



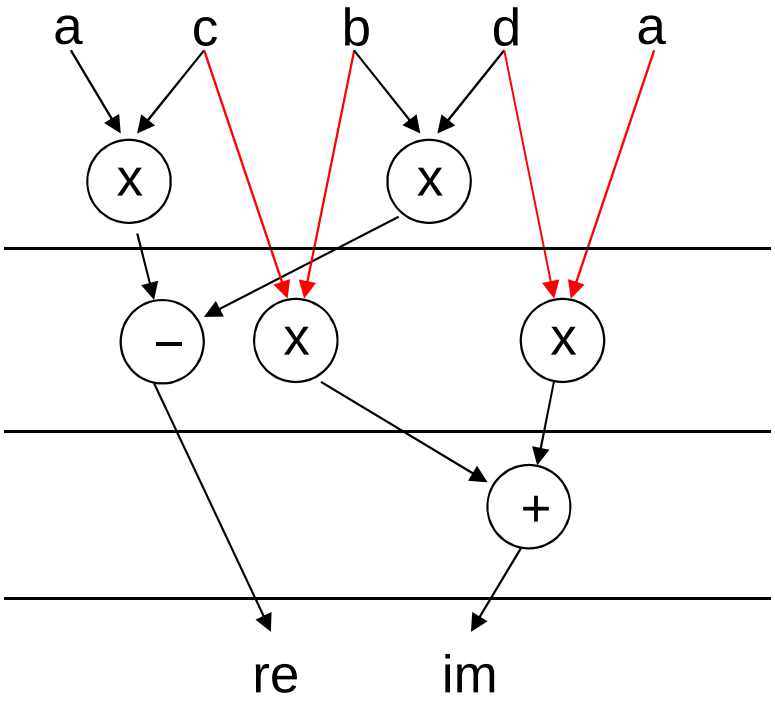

# Optimizacija površine vezja

- Kompleksno množenje

  (a + bi)(c + di) = ac – bd + (bc + ad)i




- 4 mult, 2 addsub
- 2 računska cikla




- 2 mult, 1 addsub
- 3 računski cikli