Laboratorij za načrtovanje integriranih vezij

Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*

Digitalni Elektronski Sistemi

Osnove jezika VHDL

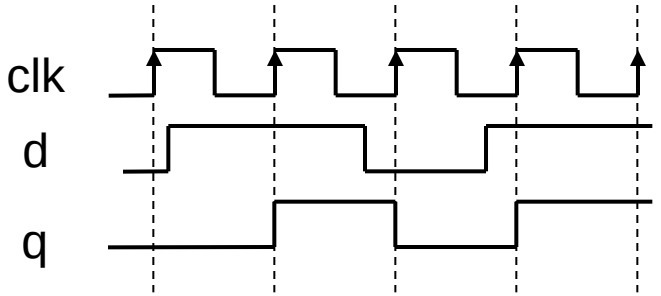
2. del: sekvenčna vezja

Opis sekvenčnih vezij

- Izhodi 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;
```

Podatkovni flip-flop



Sinteza vezja s flip-flopi

- Narediti želimo sinhrona vezja s flip-flopi (FF)
  - ne želimo asinhronih pomnilnih elementov (latch)
  - za sintezo FF uporabimo ustrezno obliko zapisa

I. oblika: popolnoma sinhrono vezje

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

II. oblika: asinhroni reset

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

Povratne zanke

- Sinhrono vezje ima lahko povratno zanko
  - na podlagi stanja izhoda 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;
```

- na podlagi stanja izhoda izračunamo novo stanje
- če je signal **q** zunanji, mora biti vrste **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;
```

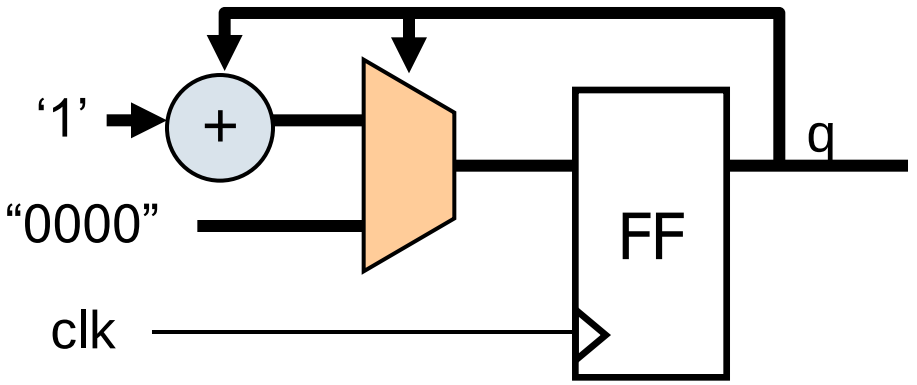
II. oblika

```
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

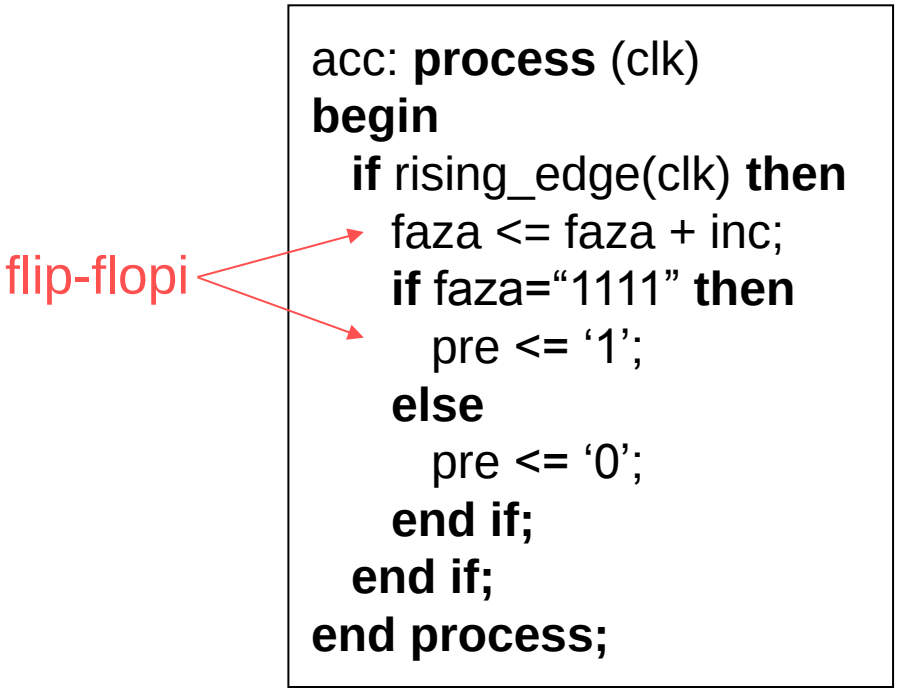
- Signal q je izhod FF, ki imajo na vhodu logiko
  - podatkovni vhod je odvisen od trenutnega izhoda – povratna zanka

```
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;
```



Kdaj dobimo flip-flope ?

- Kadar priredimo vrednost signalu ob fronti (ure), bo ta signal izhod iz flip-flopa(ov)



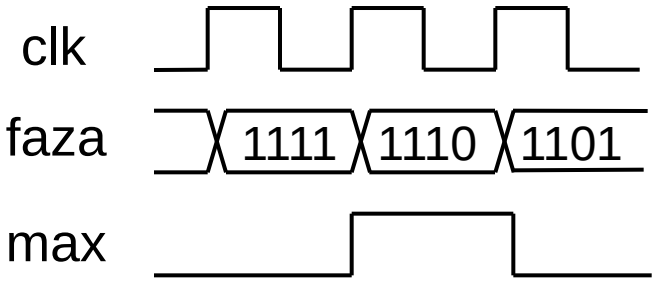
Potek simulacije

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

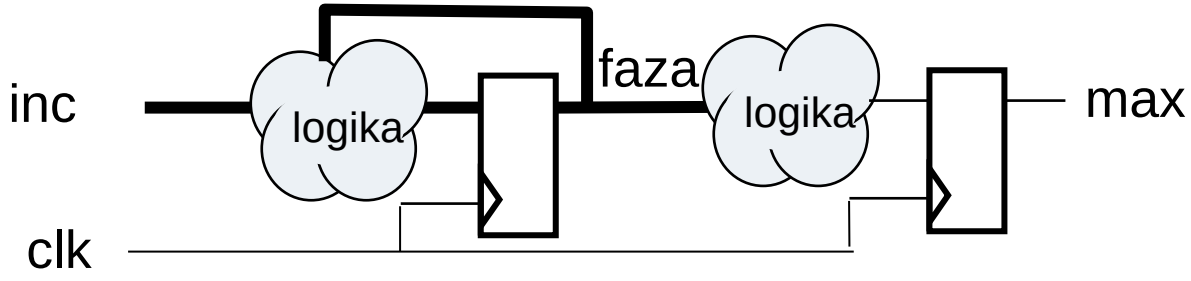
```
acc: process (clk)
begin
   if rising_edge(clk) then
      faza <= faza + inc;
      if faza="1111" then
         max <= '1';
      else
         max <= '0';
      end if;
   end if;
end process;
```

korak	**čas**	clk	inc	faza	max
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 max se postavi na '1' šele ob naslednji fronti ure
  - zakasnitev, ker je izhod iz flip-flopa !



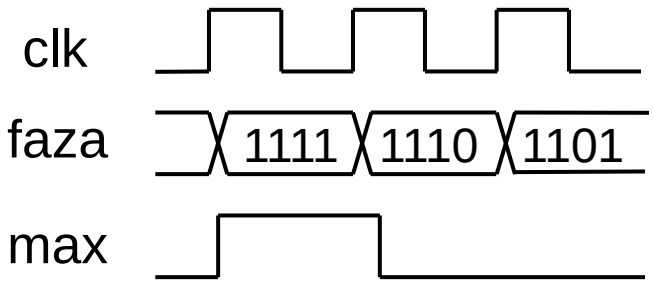
Pravila za modeliranje sekvenčnih vezij

- Sekvenčni del z izhodnimi flip-flopi opišemo s sinhronim procesom
- Kombinacijsko vezje opišemo s kombinacijskim procesom ali izven procesnega okolja

Register Transfer Level

```
acc: process (clk)
begin
   if rising_edge(clk) then
      faza <= faza + inc;
   end if;
end process;

max <= '1' when faza="1111"
           else '0';
```



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 flip-flop(e)
  - signali niso primerni za izračun vmesnih rezultatov s kombinacijsko logiko

Spremenljivke (variable)

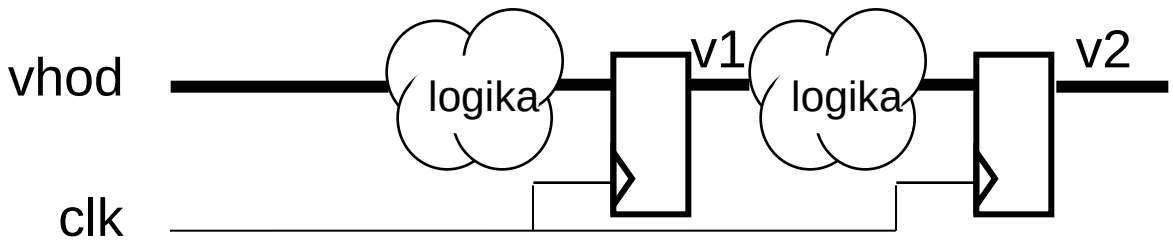
- Spremenljivke uporabljamo v procesnem okolju za izračun vmesnih rezultatov
  - ob izračunu spremenljivka takoj dobi novo vrednost
- Deklaracija spremenljivke:

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

- Prireditveni operator (:=)

```
v := vhod - “0011”;
```

Sinteza vezja s spremenljivkami



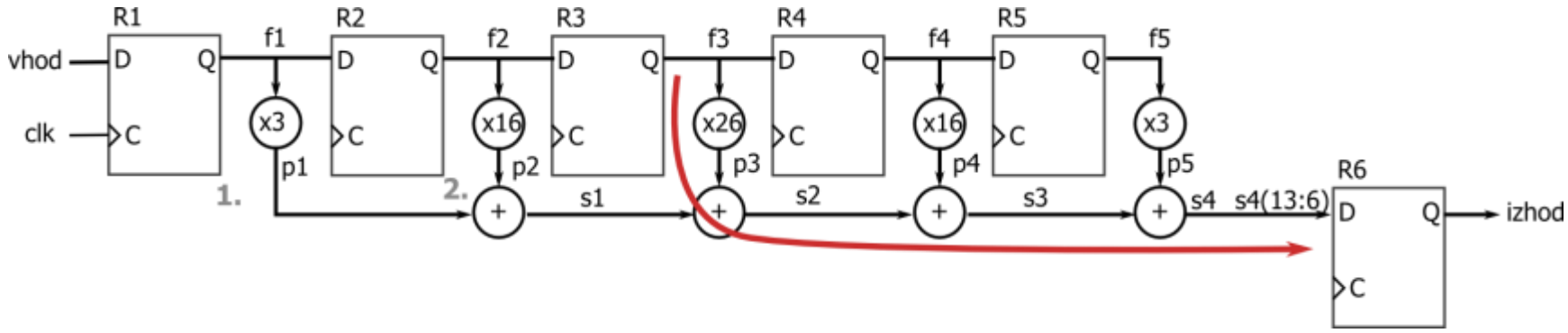

```
pipe: process (clk)
  variable v: signed (3 downto 0);
begin
   if rising_edge(clk) then
      v := vhod - “0011”;
      if v>10 then
         v1 <= v;
      else
         v1 <= “1010”;
      end if;
      v2 <= v1 xor “1010”;
   end if;
end process;
```

- prireditev spremenljivki ob fronti ure ni nujno izhod FF !
- FF dobimo šele po prireditvi spremenljivke signalu v1

Sinteza vezja za obdelavo podatkov

- Zmogljivost vezja omejujejo zakasnitve v kombinacijskem delu: prenos pri seštevanju,…
  - frekvenco ure omejuje povezava z največjo zakasnitvijo, ki se imenuje kritična pot

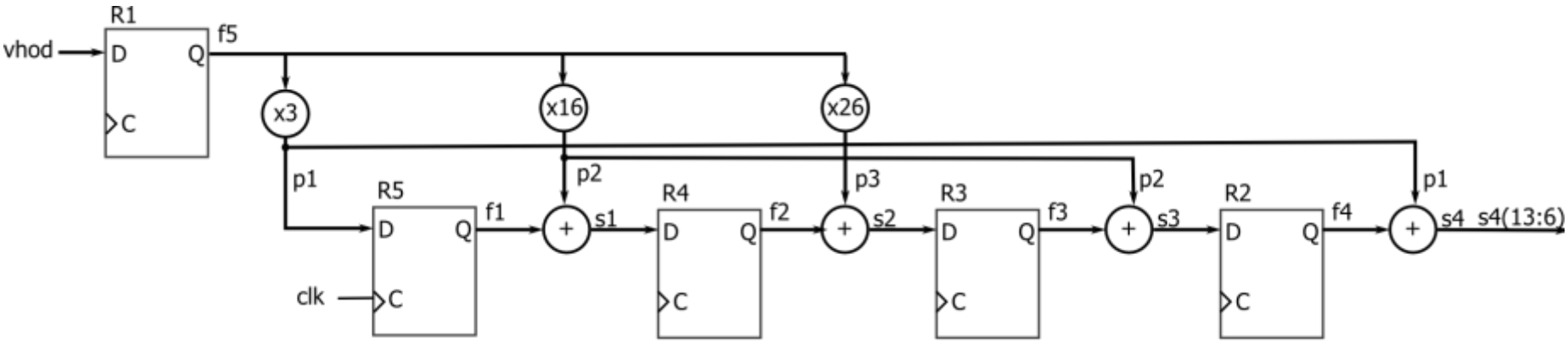
- Primer: Gaussovo sito
  - množenje z 26 = 11010 in 3 seštevanja



Optimizacija sekvenčnega vezja

- Dodamo flip-flope med kombinacijske gradnike (cevovod)
- Preuredimo operacije, da zmanjšamo kritično pot

- Optimizirano Gaussovo sito
  - množenje z 26 in eno seštevanje na kritični poti



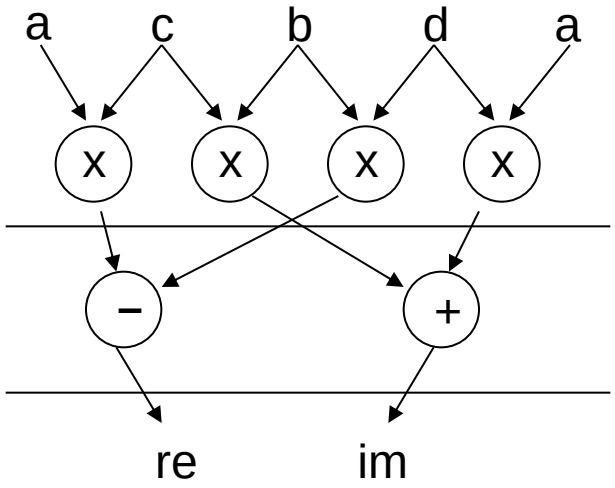
Zakasnitve odvisne od tehnologije

- Integrirana vezja ASIC: prevladujejo zakasnitve v logiki
  - zmanjšamo s topologijo vrat, obremenitvijo in načrtovanjem transistorjev
- Programirljiva vezja FPGA: velike zakasnitve v povezavah
  - na križiščih povezav so elektronska stikala, ki vnašajo zakasnitev pri prenosu signalov
  - izbira kratkih in namenskih povezav (optimizacija v programski opremi)

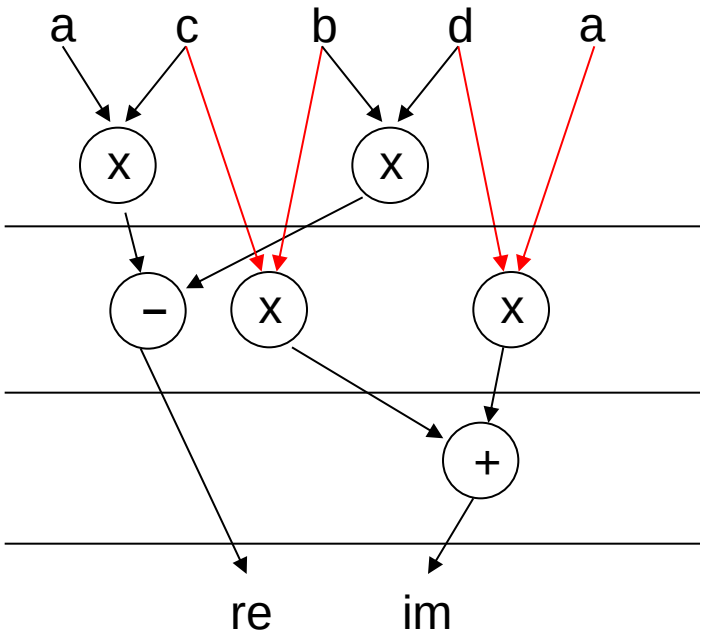
Optimizacija površine vezja

- Primer: Kompleksno množenje

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



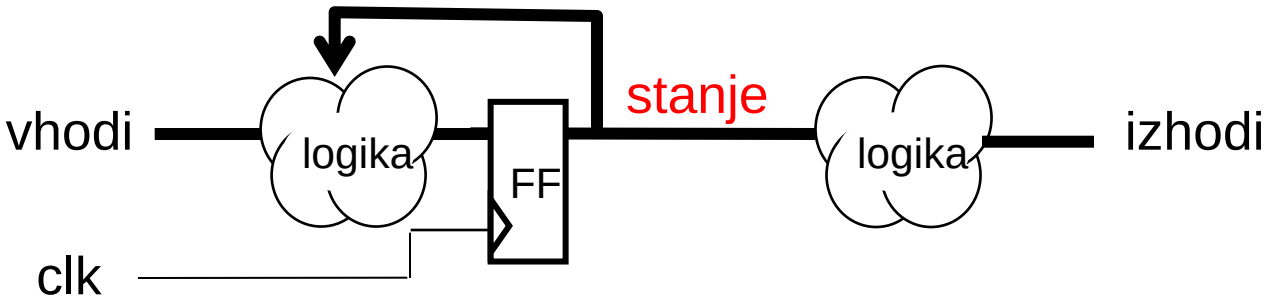
- 4 mult, 2 addsub
- 2 računska cikla



- 2 mult, 1 addsub
- 3 računski cikli

Stroj s končnim številom stanj (avtomat)

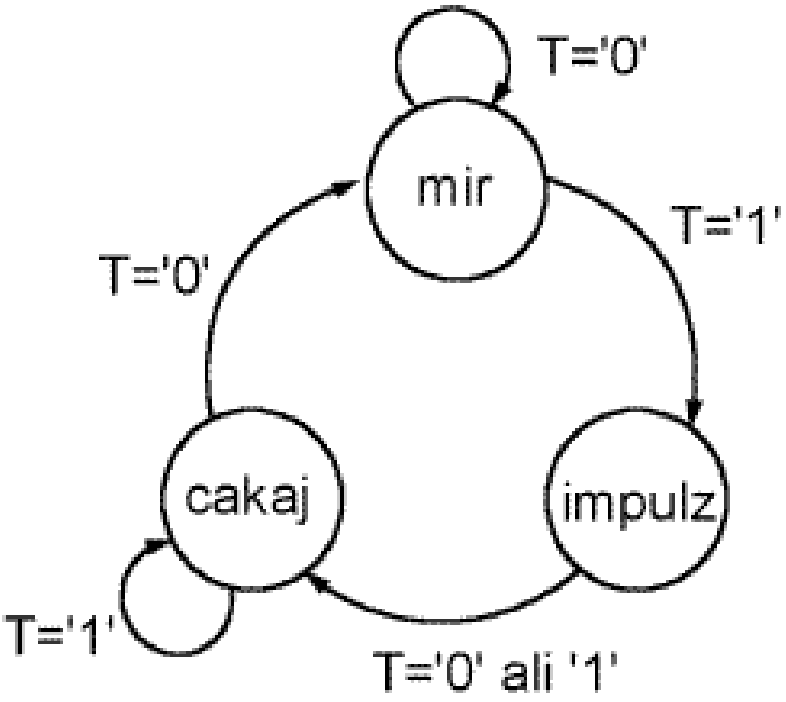
- Stroj stanj je abstrakcija iz teoretičnega računalništva
- Sekvenčno vezje, ki deluje na principu stanja in vsebuje
  - kombinacijsko logiko
  - flip-flope za shranjevanje stanja
- Npr: Moorov avtomat

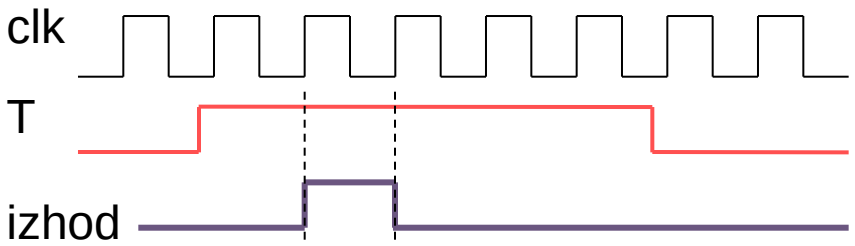


- Ob fronti ure kombinacijska logika določa izhode in naslednje stanje, ki je odvisno od vhodov in trenutnega stanja

Diagram prehajanja stanj

- Primer: vezje za detekcijo pritiska tipke
  - na izhodu naredi en impulz ob pritisku na tipko (T='1')





Opis Moorovega avtomata v jeziku VHDL

- Določimo podatkovni tip, kjer naštejemo stanja
  - sinhroni proces za shranjevanje stanj

```
architecture RTL of avtomat is
  type stanja is (mir, impulz, cakaj);
  signal stanje, naslednje: stanja;
begin

 FF: process (clk)
 begin
    if rising_edge(clk) then
       stanje <= naslednje;
    end if;
 end process;
```

prehodi med stanji:

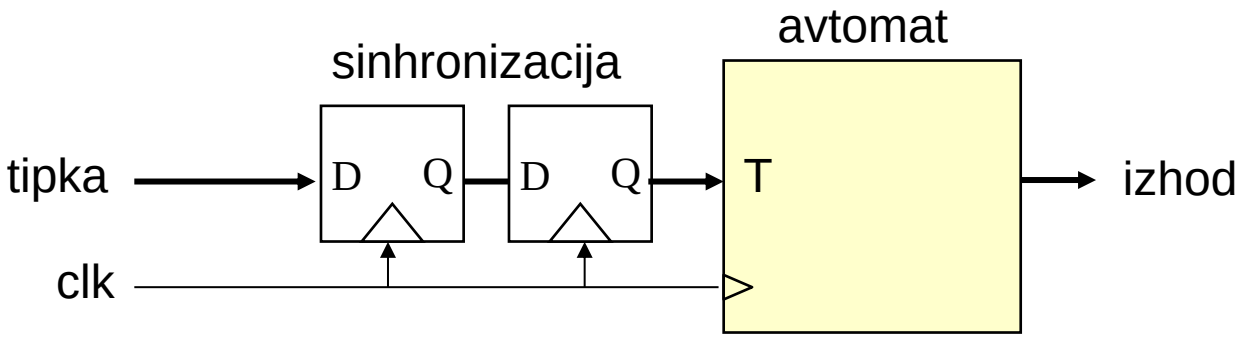
```
preh: process (stanje, T)
begin
 case stanje is
   when mir =>
    if T = '1' then
      naslednje <= impulz;
    else
      naslednje <= mir;
    end if;
   when impulz =>
      naslednje <= cakaj;
```

Izhodna logika

- Kombinacijska logika za izhode
  - kombinacijski proces ali izbirni stavek when … else

```
komb: process (stanje)
      begin
        if stanje=impulz then
          izhod <= '1';
        else
          izhod <= '0';
        end if;
end process;
```

- Vhod v avtomat je potrebno sinhronizirati !



Sinteza in kodiranje stanj

- Kako so kodirana stanja?
  - kodiranje stanj naredi program za sintezo vezja
  - nekatere kode ne predstavljajo stanja iz diagrama
    - V stavku **case** uporabimo **when others=>** kjer določimo v katero stanje naj se premakne avtomat, če pride slučajno v nedefinirano stanje

binarno
mir = 00
impulz = 01
cakaj = 10

one-hot
mir = 001
impulz = 010
cakaj = 100

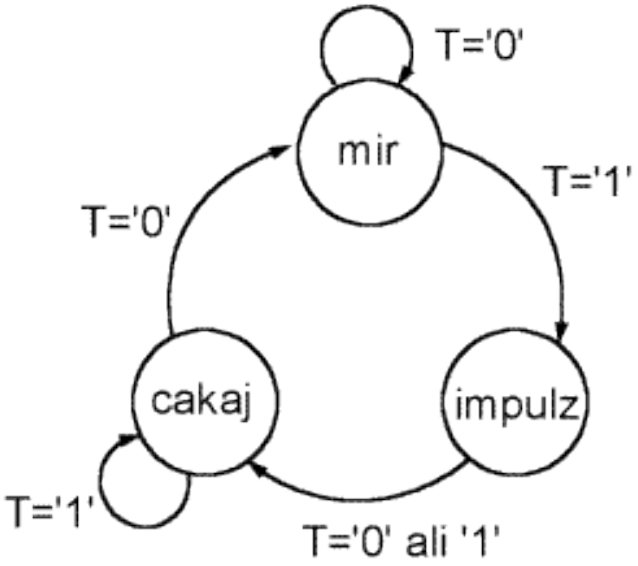
Kompakten opis v jeziku VHDL

- Kompakten opis sekvenčnega stroja stanj
  - v enem procesu opišemo flip-flope in vhodno logiko

```
sekv: process (clk)
begin
    if rising_edge(clk) then
      case stanje is
        when mir =>
          if tipka='1' then
            stanje <= impulz;
          end if;

        when impulz =>
          stanje <= cakaj;

        when others =>
          if tipka='0' then
            stanje <= mir;
          end if;
      end case;
```




```
izhod <= '1' when stanje=impulz
         else '0';
```