Laboratorij za načrtovanje integriranih vezij

Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*

Digitalni Elektronski Sistemi

Osnove jezika VHDL

Model in optimizacija vezja

Model vezja na nivoju RTL

- Algoritem določa obnašanje vezja
  - pri algoritmu ni določen časovni potek izvajanja
  - sinteza vezij iz algoritma je precej zahtevna
- Na nivoju RTL je določeno obnašanje vezja ob urinih ciklih
  - sinteza vezij iz RTL opisa je danes običajna
- Modeliramo prenos podatkov med pomnilnimi elementi in (kombinacijske) transformacije

Primer: šifriranje podatkov

- Blokovni algoritem s 4 iteracijami šifriranja

C++

```
int main(void)
{
   char a, b, a_nov, b_nov;
   char k[] = {0xff, 0x0f, 0x03, 0x30};
   cin >> a;
   cin >> b;
   for (int i=0; i<=3; i++) {
      a_nov = b;
      b_nov = (b + k[i]) ^ a;
      a = a_nov;
      b = b_nov;
   }
  cout << a << b;
}
```

VHDL

Behavioral

```
for i in 0 to 3 loop
   a_nov := b;
   b_nov := (b + k(i)) xor a;
   a := a_nov;
   b := b_nov;
end loop;
```

1. vezje: razvijemo zanko

- kombinacijska izvedba zanke
  - rezultat operacij vsakokrat shranimo v nov signal

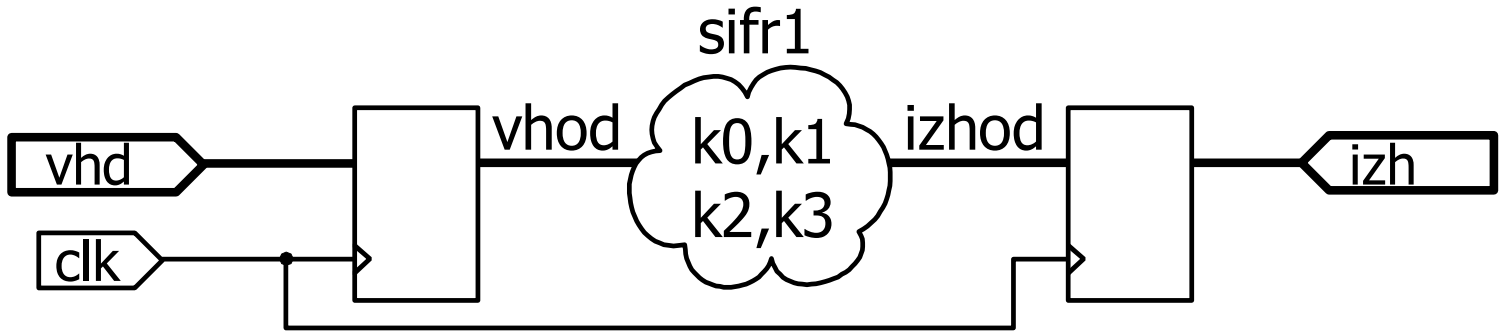
```
sifr1: process(vhod, a0, b0, a1, b1, a2, b2, a3, b3, a4, b4 )
 begin
     a0 <= vhod(15 downto 8);
     b0 <= vhod(7 downto 0);
     a1 <= b0;
     b1 <= (b0 + k(0)) xor a0;
     a2 <= b1;
     b2 <= (b1 + k(1)) xor a1;
     a3 <= b2;
     b3 <= (b2 + k(2)) xor a2;
     a4 <= b3;
     b4 <= (b3 + k(3)) xor a3;
 end process;
```

VHDL

RTL

Rezultat 1. vezja

- registri na vhodu in izhodu



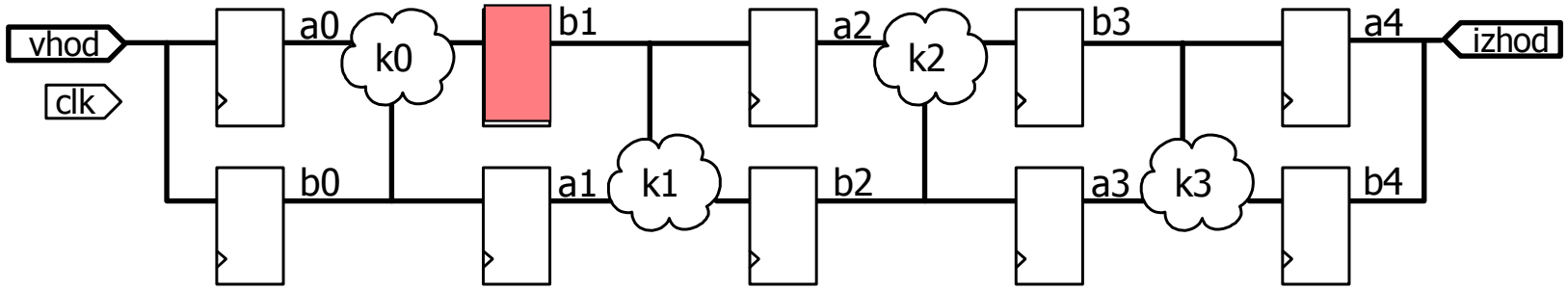
CPLD Xilinx XC95288XL-10

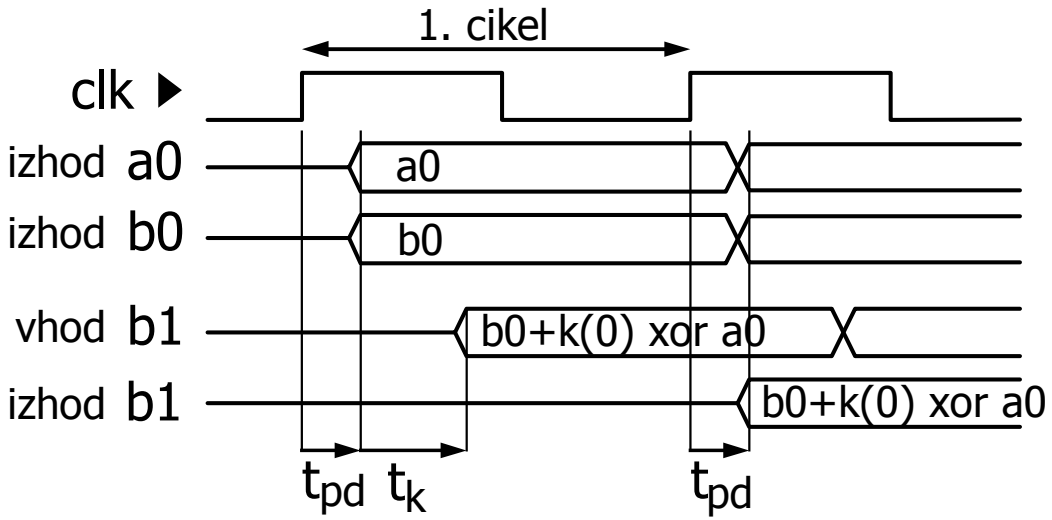
- Površina: 56 makrocelic, 32 flip-flopov
- Frekvenca ure: 26.8 MHz (53.6 MByte/s)

Kako povečati hitrost vezja?

2. vezje: polna cevovodna izvedba

- cevovodna izvedba zanke
  - rezultat vsake iteracije shranimo v register

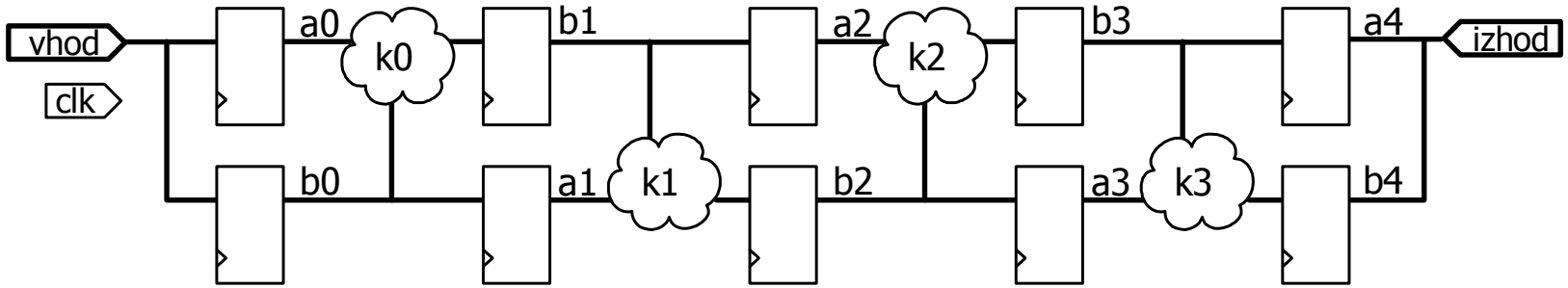




- opazujmo vhod in izhod registra b1
- frekvenca ure je omejena s $t_{pd}+t_k+t_s$
- vezje naj deluje le na eno fronto ure!

Rezultat 2. vezja

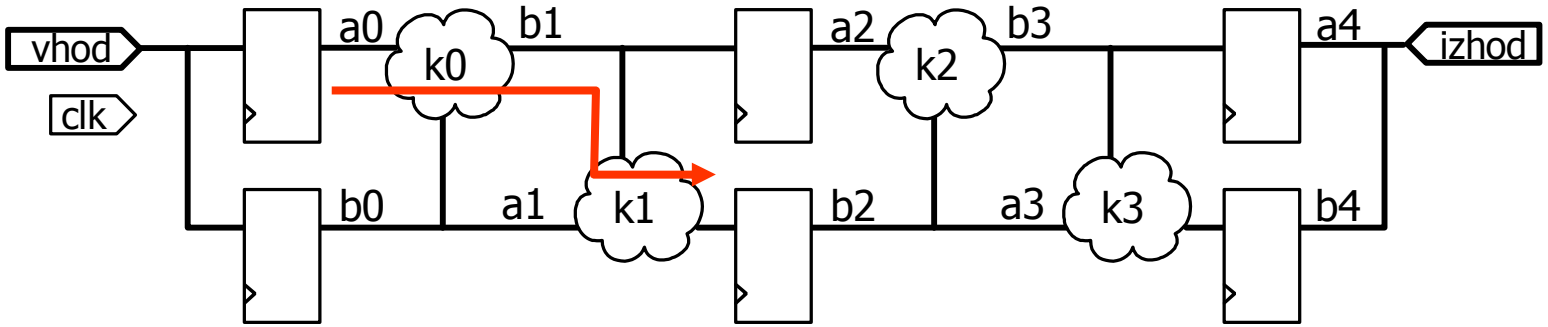
- 4-stopenjski cevovod



CPLD Xilinx XC95288XL-10

- Površina: 80 makrocelic, 80 flip-flopov
- Frekvenca ure: 90.9 MHz
- Zmogljivost: 181.8 MByte/s

3. vezje: dvostopenjski cevovod

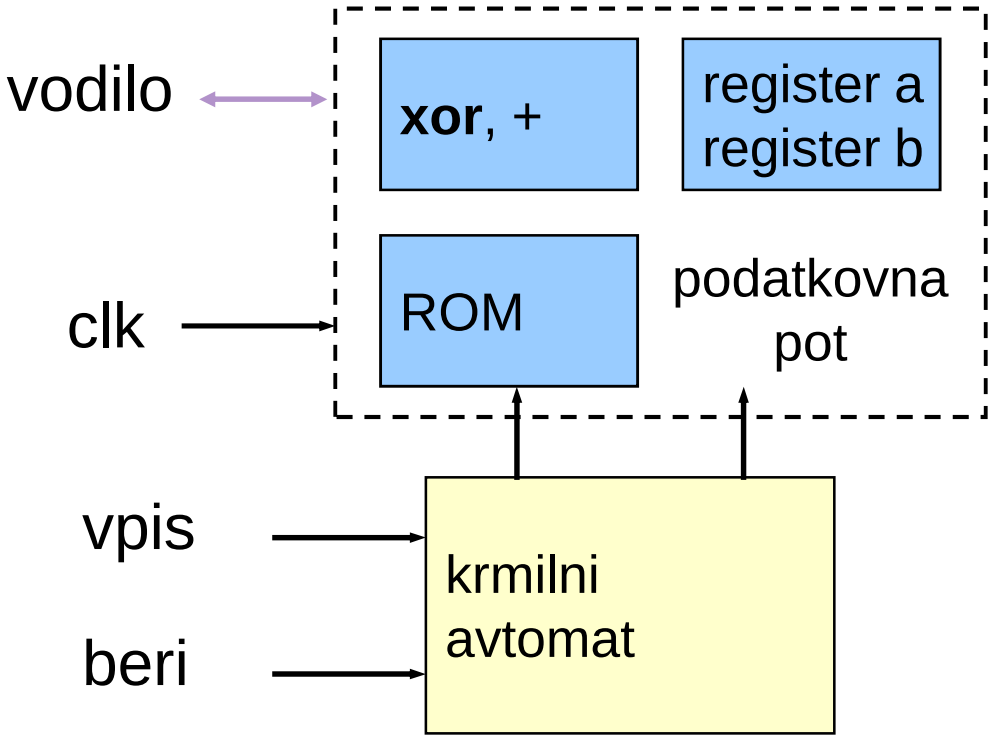


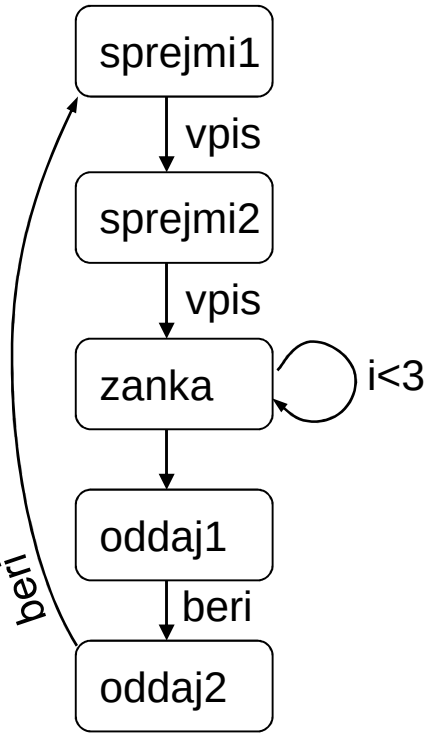
CPLD Xilinx XC95288XL-10

- Površina: 54 makrocelic, 48 flip-flopov
- Frekvenca ure: 46.1 MHz
- Zmogljivost: 92.2 MByte/s

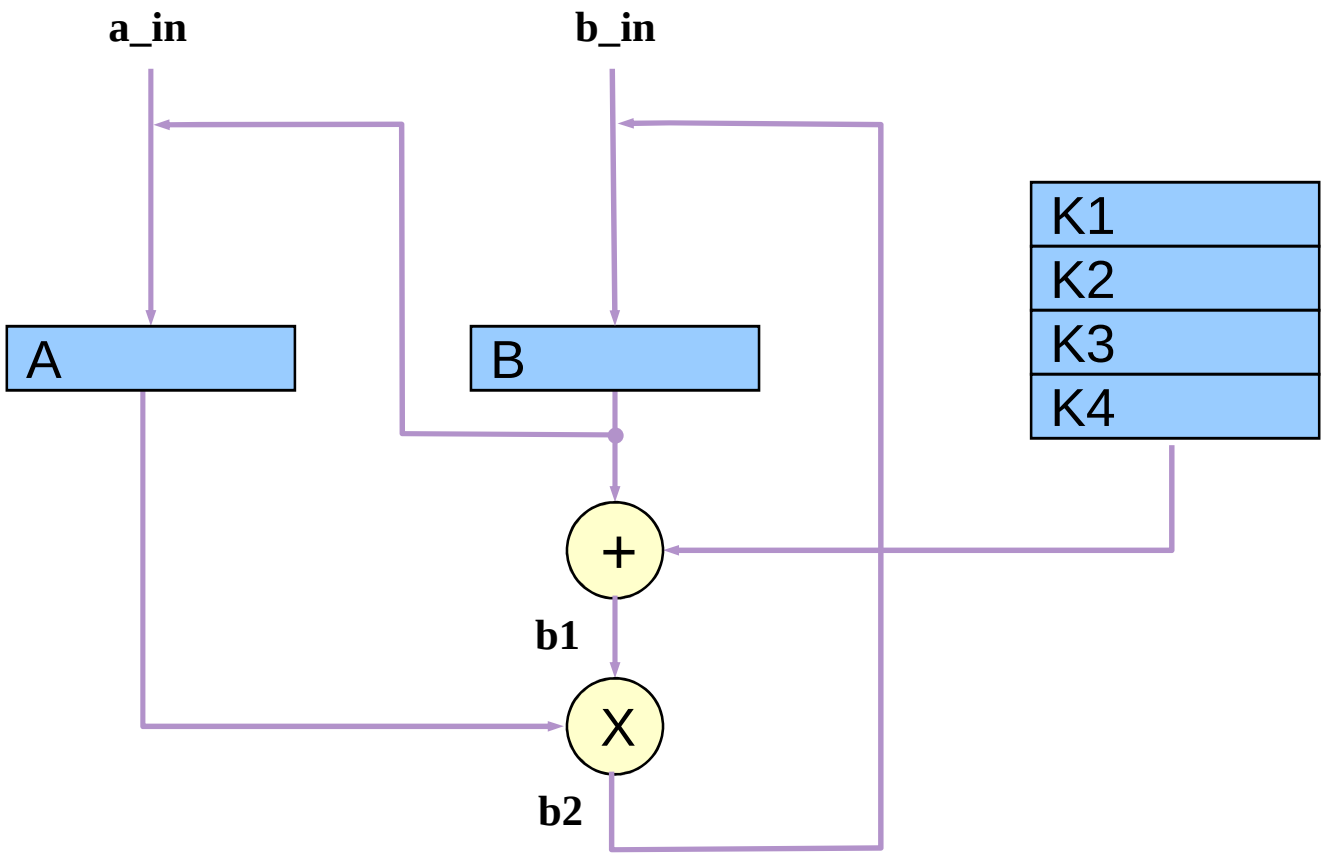
4. vezje: sekvenčni procesor

- Vezje razdelimo na podatkovni in kontrolni del

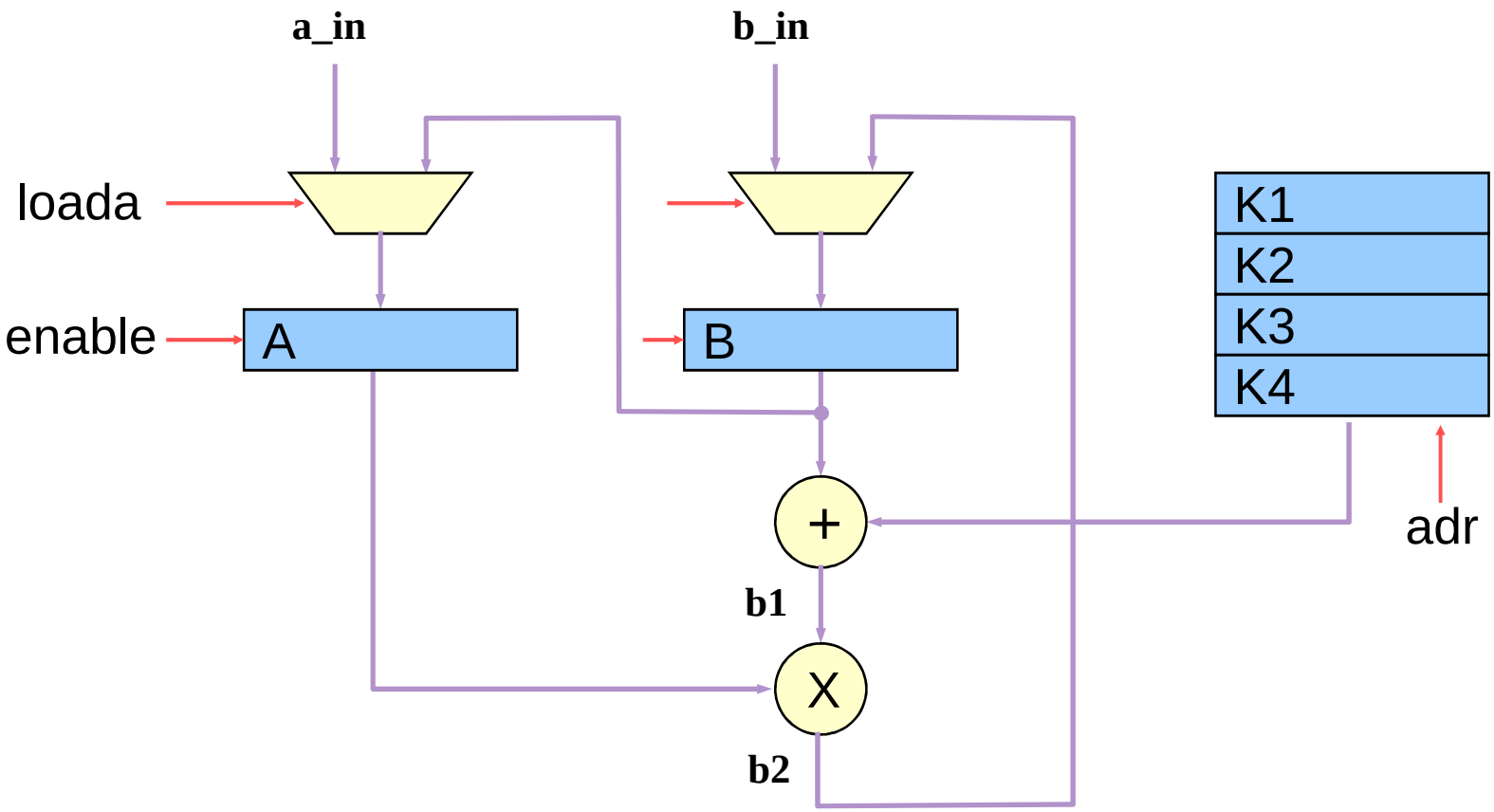




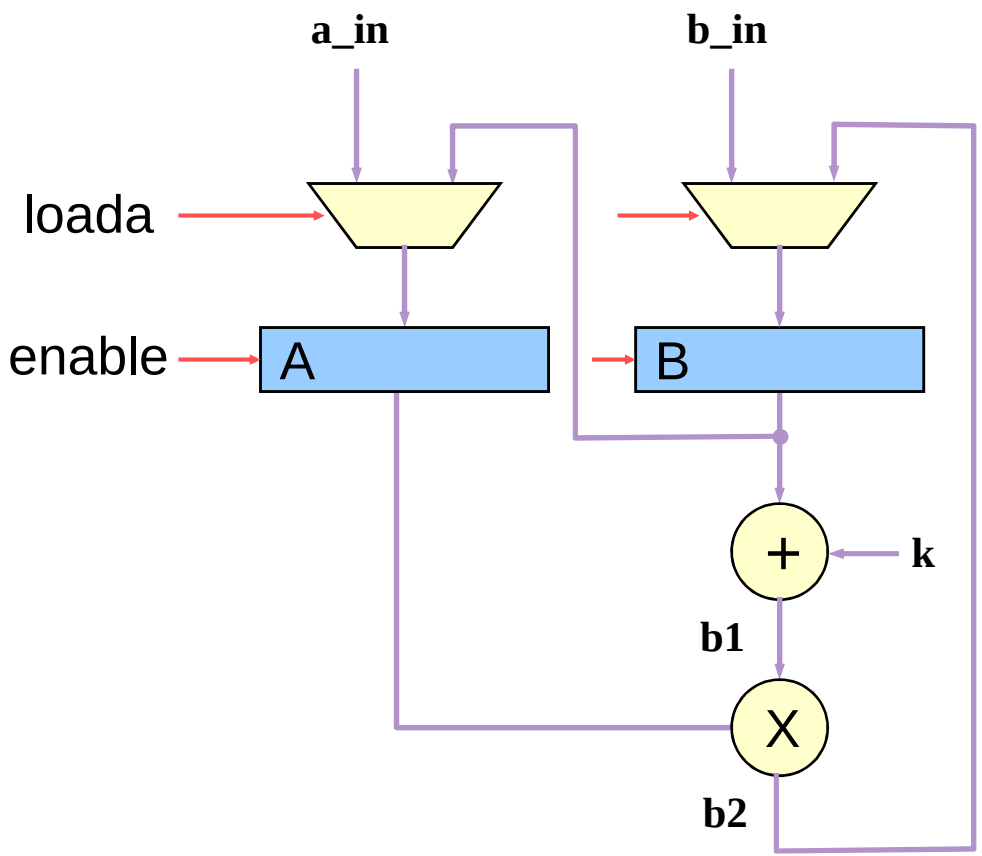
Podatkovna pot



4. Krmilni signali



4. Podatkovna pot v jeziku VHDL

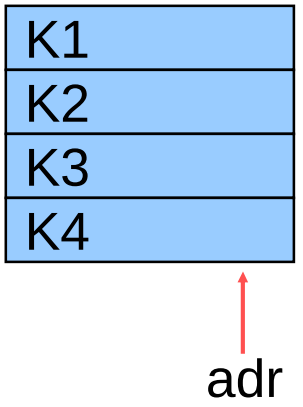



```
pod: process(clk)
begin
   if rising_edge(clk) then
      if enable = '1' then
         if loada = '1' then
            a <= a_in;
         elsif loadb = '1' then
            b <= b_in;
         else
            a <= b;
            b <= b2;
         end if;
      end if;
   end if;
end process;

b1 <= b + k;
b2 <= b1 xor a;
```

4. ROM v jeziku VHDL

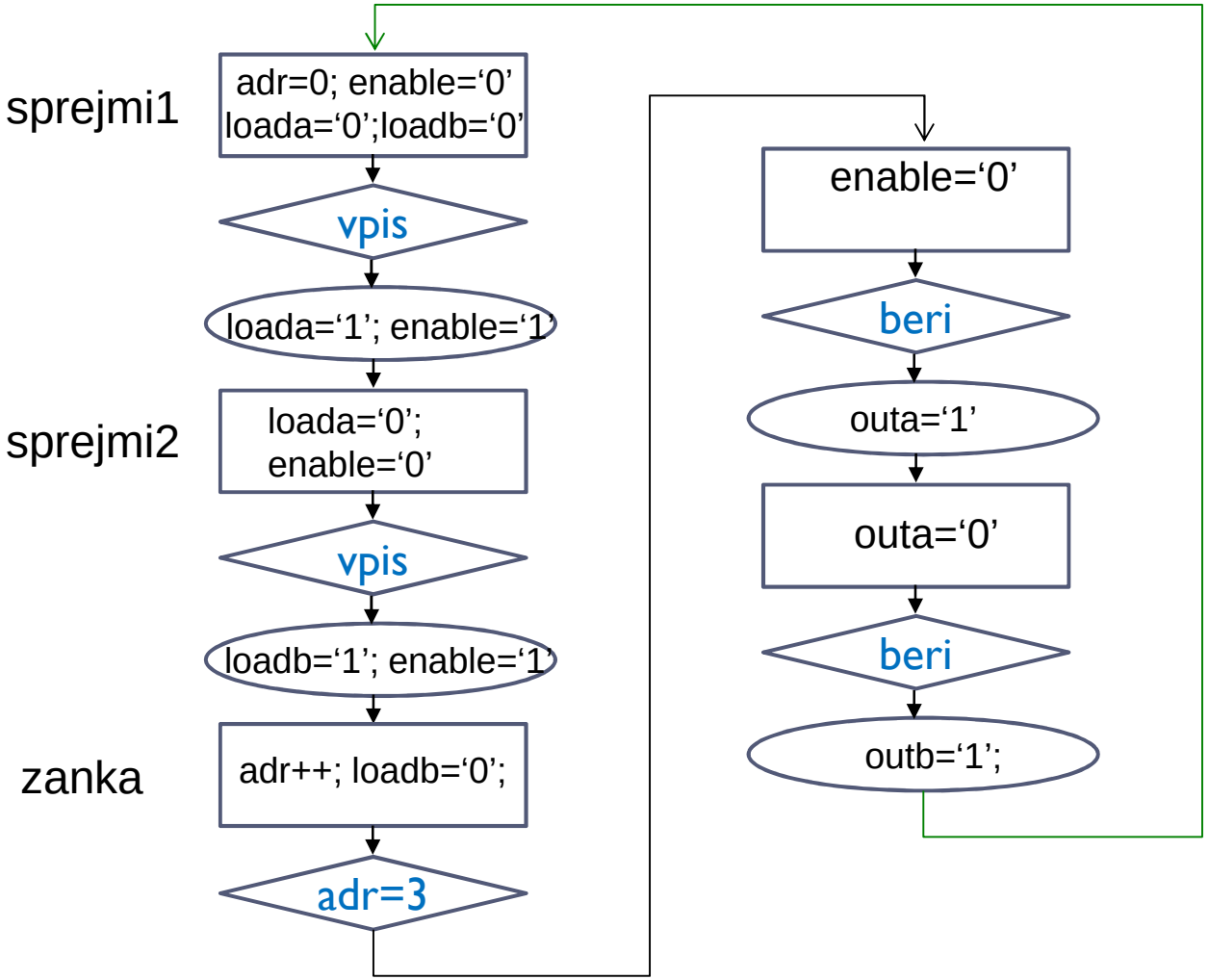
- za opis uporabimo zbirko (array)

```
architecture RTL of code is
 ...
 type rom_type is array (0 to 3) of
      std_logic_vector (15 downto 0);

 constant rom : rom_type :=
      ("0000000000000001",
      "0000000000000011",
      "0000000000000111",
      "0000000000001111");

 signal adr: integer range 0 to 3;
 ...
begin
  k <= rom(adr);
```



4. Krmilno vezje

- Algoritmični sekvenčni avtomat



Rezultat in primerjava

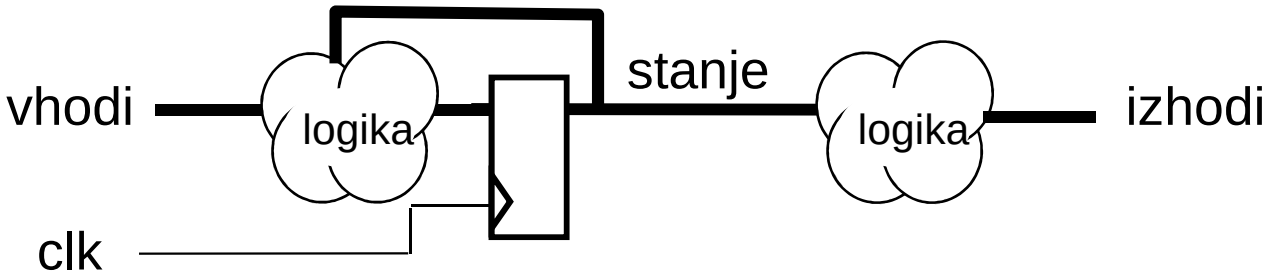
CPLD Xilinx XC95288XL-10

- Površina: 31 makrocelic, 22 flip-flopov
- Frekvenca ure: 84.7 MHz (10,6 Mbyte/s)

Vezje	Površina	f [MHz]	Mbyte/s
1. vezje	56	26.8	53,6
2. vezje	80	90.9	181,8
3. vezje	54	46.1	92,2
4. vezje	31	84.7	10,6

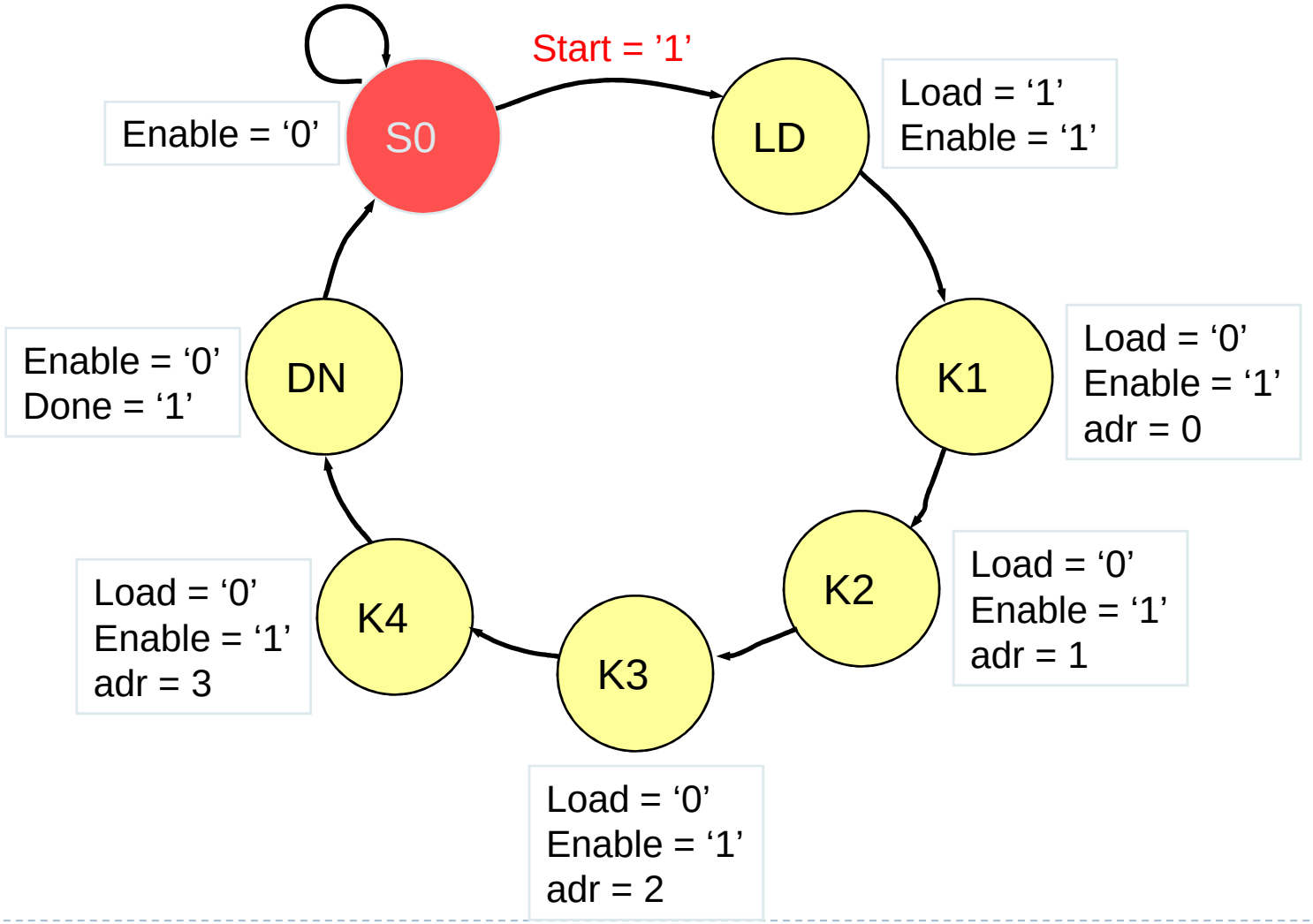
Končni avtomat (Finite State Machine)

- Sekvenčno vezje s povratno zanko
  - register stanj
  - vhodna logika
  - Izhodna logika
- Moorov avtomat:



Primer končnega avtomata

- Pri vsakem stanju določimo vrednosti izhodov



Opis avtomata v jeziku VHDL

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

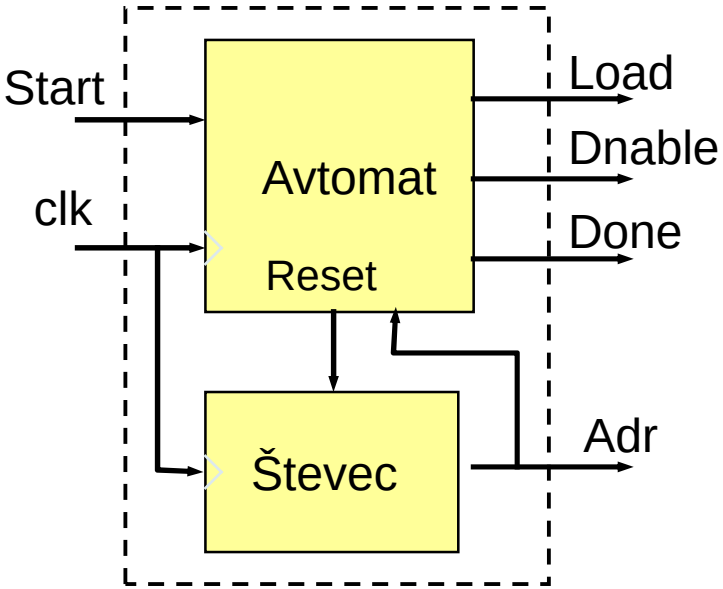
```
architecture RTL of avtomat is
 ...
 type stanja is (s0, ld, k1, k2, k3, k4, dn);
 signal stanje, naslednje: stanja;
...
begin

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

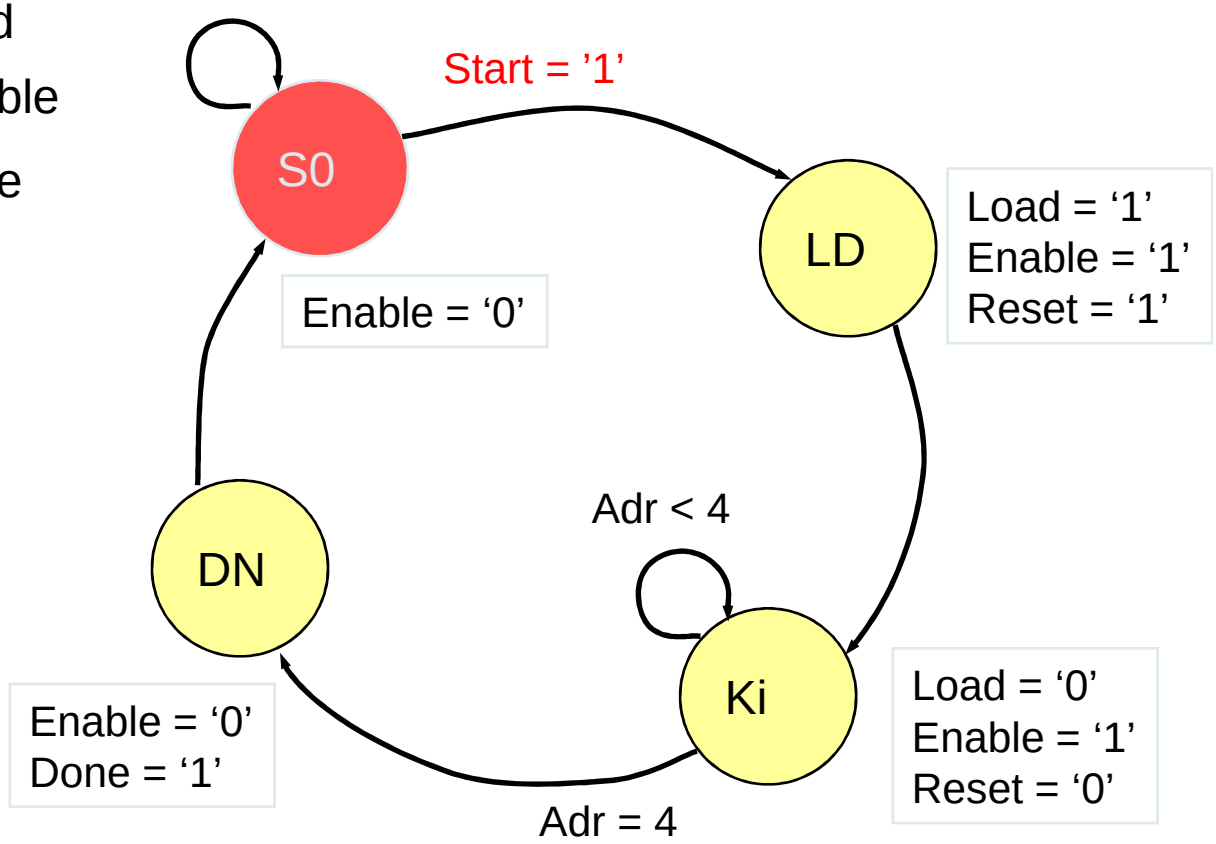
prehodi med stanji:

```
preh: process (stanje, start, adr)
begin
 case stanje is

  when s0 =>
   if start = '1' then
     naslednje <= ld;
   else
     naslednje <= s0;
   end if;

  when ld =>
     naslednje <= k1;
```

Avtomat z zunanjim števcem

- namesto stanj v katerih spreminjamo adr uporabimo zunanji števec





Izhodna logika in števec

- Kombinacijska logika za izhode (when … else)
- Sinhroni proces za zunanji števec

```
Load <= '1' when stanje=ld else '0';
Enable <= '0' when stanje=s0 or stanje=dn else '1';
Done <= '1' when stanje=dn else '0';
Reset <= '1' when stanje=ld else '0';

stev: process(clk)
begin
  if rising_edge(clk) then
    if Reset='1' then            -- mora biti sinhroni
     Adr <= 0;
    else
     Adr <= Adr + 1;
    end if;
  end if;
end process;
```

- kombinacijski izhodi imajo šum ob prehajanju stanj
- ne smemo jih uporabiti za uro ali asinhroni reset…
- šum odstranimo, če jih pošjemo čez D flip-flop

Vhodni signali

- Vhodni signali v sinhroni avtomat morajo biti sinhronizirani z uro !
  - sicer prekršimo dinamični red in avtomat gre v poljubno stanje
- 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
S0= 000
LD= 001
K1= 010
K2= 011
K3= 100
K4= 101
DN=110

one-hot
S0 = 0000001
LD = 0000010
K1= 0000100
K2= 0001000
K3= 0010000
K4= 0100000
DN= 1000000

Končni avtomat z registrskimi operacijami

- Kompakten opis sekvenčnega vezja
  - V enem procesu, opišemo register in prehajanje med stanji ter operacije (npr. nalaganje registra, povečevanje – števec…)

```
avtomat: process (clk)
 begin
   if rising_edge(clk) then
     case stanje is

     when s0 =>
       if start='1' then
         stanje <= ld;
         reg <= vhod;
       end if;

     when ld =>
       stanje <= ki;
       Adr <= 0;

     when ki =>
       if Adr<3 then
         Adr <= Adr + 1;
       else
         stanje <= dn;
       end if;

     when others =>
         stanje <= s0;
     end case;
   end if;
end process;
```