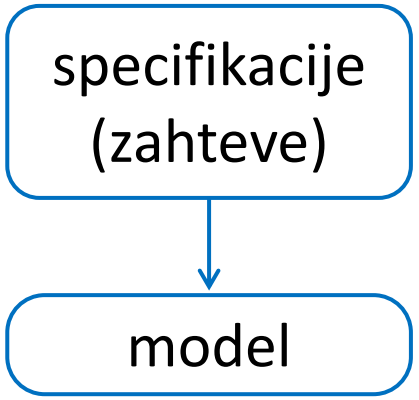
Sinteza in simulacija

Digitalni elektronski sistemi 2024/25

Andrej Trost

UNIVERZA V LJUBLJANI
FE
Fakulteta za elektrotehniko

Načrtovanje digitalnega vezja



Analiza in simulacija

- logične in električne lastnosti
- preveri delovanje vezja
  - ali vezje izvaja predvidene operacije

Logična sinteza

- pretvorba opis vezja v tehnološko obliko
- iz abstraktnega v podroben model
- rezultat je vezje iz logičnih gradnikov, ki so primerni za fizično izvedbo

Sinteza vezja iz strojno-opisnega jezika

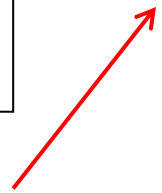
- sintetizator (synthesis tool) pretvori HDL v vezalno shemo
  - preveri sintakso in morebitne napake
  - iz opisa izloči gradnike vezja in povezave med njimi
  - izvede optimizacijo
- strukturni opis
  - najenostavnejši, ker že modelira gradnike in povezave
- funkcijski opis
  - vsaj stavek predstavlja logiko povezano na signal, ki mu prirejamo vrednost
- postopkovni opis
  - zahteva analizo opisa in iskanje vzorcev
  - delitev na kombinacijski in sekvenčni del vezja

Sinteza kombinacijskih vezij

- v postopkovnem opisu definiramo vrednosti pri vseh pogojih:
  - vrednost priredimo pri vsakem “if” in “else” ali
  - vrednost priredimo na začetku procesa in jo spremenimo v “if” stavkih
- npr: primerjalnik

```
primerjava: process (a, b)
begin
   if a=b then
      enako <= '1';
   else
      enako <= '0';
   end if;
end process;
```

```
primerjava: process (a, b)
begin
1.    enako <= '0';
      if a=b then
2.       enako <= '1';
      end if;
end process;
```



- vrstni red stavkov je pomemben !

Popravi model vezja

- Kaj je narobe z modelom izbiralnika s signalom za omogočanje (en)?

```
entity demo
 a, en, d0, d1: in u1
 y: out u1;
begin
 if en=0 then
   y = 0
 end
 if a=0 then
   y = d0
 else
   y = d1
 end
end
```

1.
2.

- prvi stavek se ne izvede, ker drugi stavek v vsakem primeru (if-else) nastavi izhod
- potrebno je zamenjati vrstni red stavkov ali pa združiti pogoje v en stavek s prioriteto (if-elsif-else)

Vrstni red stavkov

- Ali je vrstni red označenih stavkov pomemben?

1. 2.
```
if c=1 then
  y = x
  x = 0
end
```

1. 2.
```
z = x
if sel=1 then
  x = a
else
  x = b
end
```

1. 2.
```
if s=1 then
  a = 1
end
a = 0
b = a
```

  - NE, kadar prirejamo vrednost različnim signalom, sicer DA

Zapah

- Ali je v kodi opisan zapah?

```
if c=1 then
  a = d
end
```

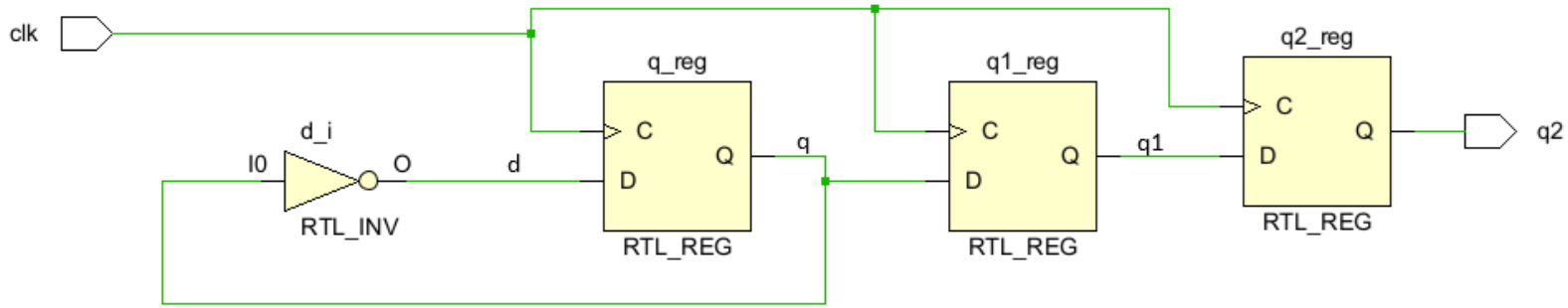
```
if c=1 then
  a = 1
elsif c=2 then
  a = 2
end
```

```
if e=1 then
  if c=1 then
    a = 1
  else
    a = 0
  end
end
```

- DA, ker izhodni signal ni definiran v vseh primerih

Sinteza sekvenčnih vezij

```
entity Logic
 d:  u1;
 q,q1:  u1;
 q2: out u1;
begin
 d = not q
 q <= d
 q1 <= q
 q2 <= q1
end
```



- prireditev ob uri opisuje flip-flop ali register
- vrstni prireditvenih stavkov z različnimi izhodi ni pomemben!

```
process(clk)
begin
 if rising_edge(clk) then
  q <= d;
  q1 <= q;
  q2 <= q1;
 end if;
end process;
```

```
process(clk)
begin
 if rising_edge(clk) then
  q2 <= q1;
  q <= d;
  q1 <= q;
 end if;
end process;
```

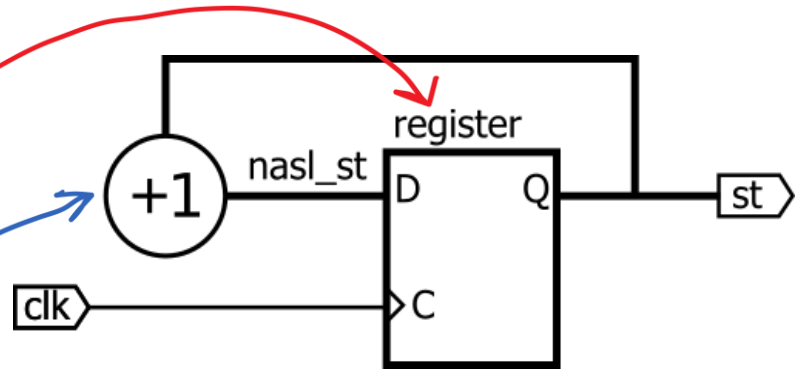
```
process(clk)
begin
 if rising_edge(clk) then
  q2 <= q1;
  q1 <= q;
  q <= d;
end if;
end process;
```

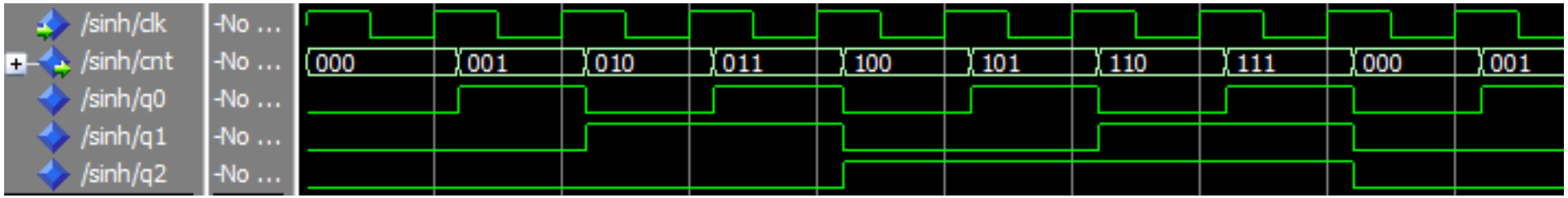
Sinteza vezij s povratno zanko

- model vezja razdelimo na kombinacijski in sekvenčni del
- primer: sinhroni števec

```
entity stev
 st: out u3
begin
 st <= st + 1
end
```

```
entity stev
 st: out u3;
 nasl_st: u3;
begin
 st <= nasl_st
 nasl_st = st + 1
end
```

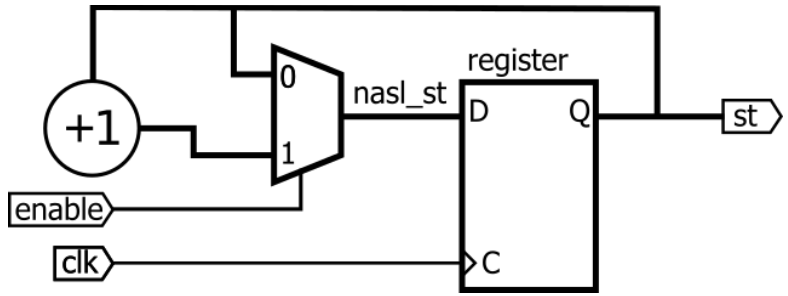


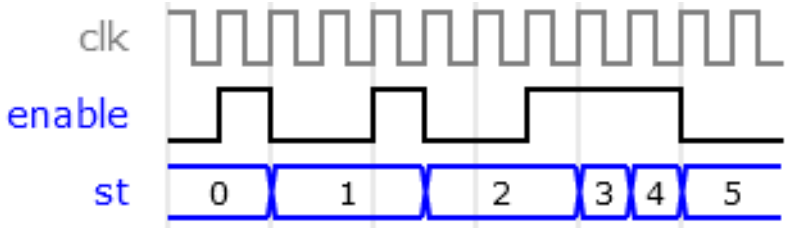


Sinhroni števec z omogočanjem štetja

- števec ob enable=0 ohranja zadnje stanje, ob enable=1 pa se povečuje
- v vezju je izbiralnik za naslednje stanje

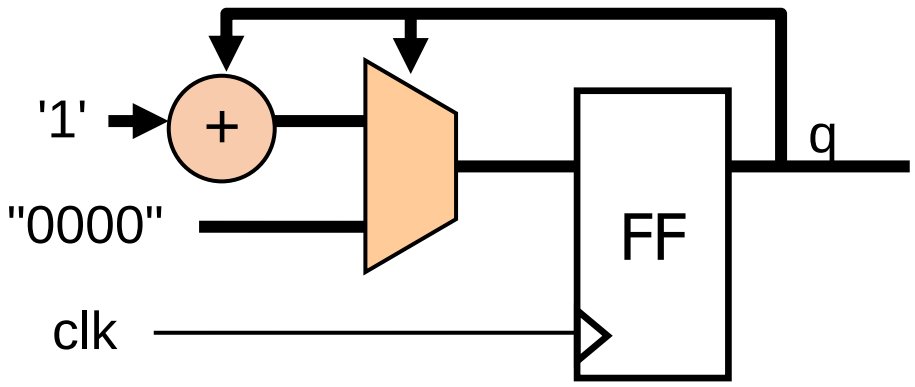
```
entity stev
 st: out u3;
 enable: in u1;
 nasl_st: u3;
begin
 if (enable) then
   nasl_st = st + 1
 else
   nasl_st = st
 end
 st <= nasl_st
end
```





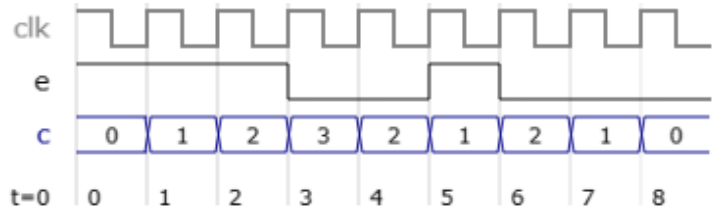
Sinteza vezja BCD števca

```
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;
```

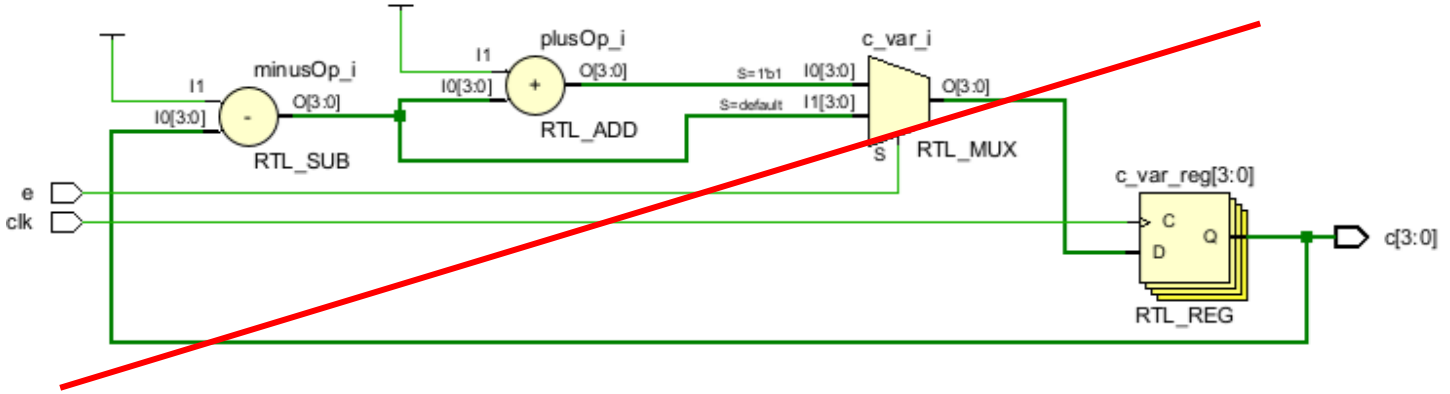


Sinteza postopkovnega opisa

- izvede se le zadnja prireditev istemu signalu
  - primer: vrednost signala c ob naslednjem ciklu, če je e=1

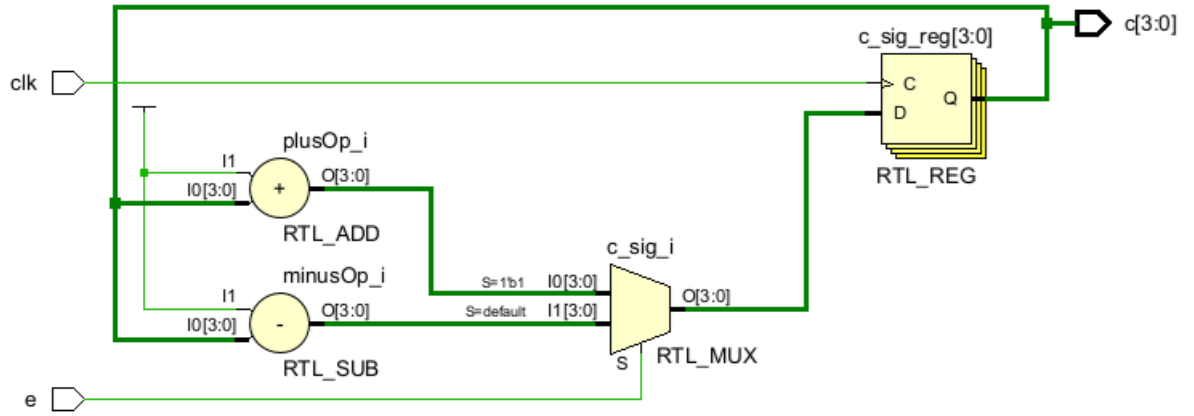



```
entity test
 e: in u1;
 c: out u4;
begin
 c <= c - 1
 if e=1 then
   c <= c + 1
 end
end
```



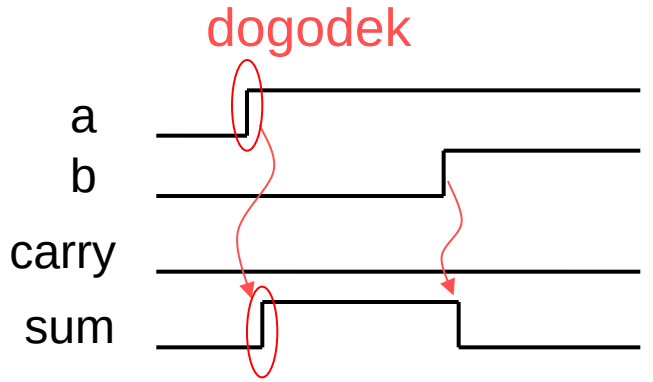
- enako vezje

```
 if e=1 then
   c <= c + 1
 else
   c <= c - 1
 end
```



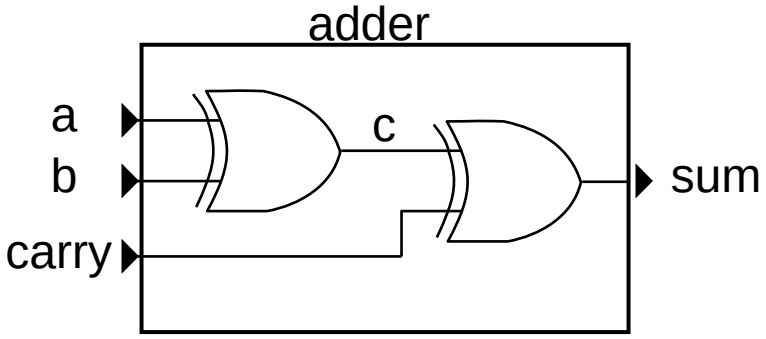
Simulator diskretnih dogodkov

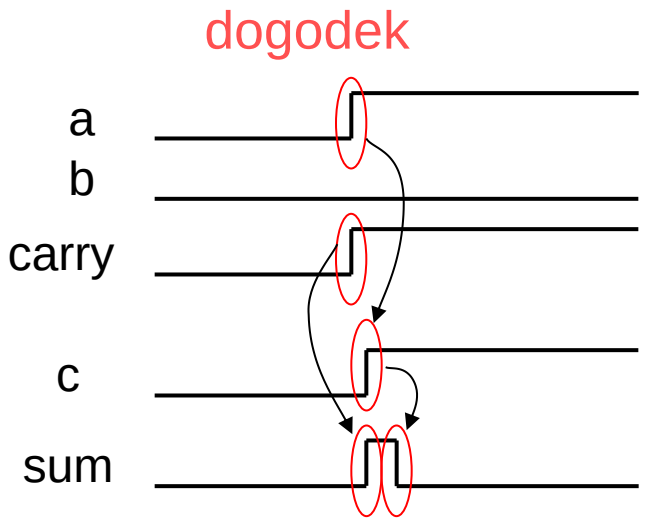
- simulacija vezja v strojno-opisnem jeziku
  - vrstni red stavkov ni pomemben
- opazuje stanje modela vezja in obravnava dogodke, ki se spreminjajo s časom
- simulacijska zanka
  1. izračunaj in uvrsti dogodek na seznam
  2. povečaj čas
  3. posodobi stanje



Graf simulacije (waveform)

Simulacija seštevalnika





seznam dogodkov:
a: 0, 1 (T) carry: 0, 1 (T)

sum = c **xor** carry
c = a **xor** b

korak	**čas**	a	b	carry	sum	c
seznam	0	0, 1(T)	0	0, 1(T)	0	0
posodobi	T	1	0	1	0	0
izračunaj	T	1	0	1	1(T+ Δ)	1(T+ Δ)
posodobi	T+ Δ	1	0	1	1	1
izračunaj	T+ Δ	1	0	1	0(T+2 Δ)	1
posodobi	T+2 Δ	1	0	1	0	1

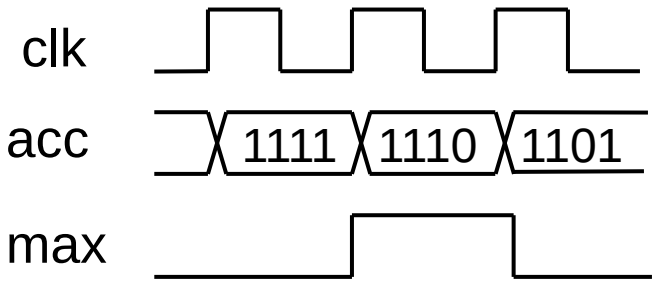
Simulacija sekvenčnega vezja

- primer: akumulator in zastavica max
- simulacija procesa se izvede ob prehodu ure (clk) iz 0 na 1

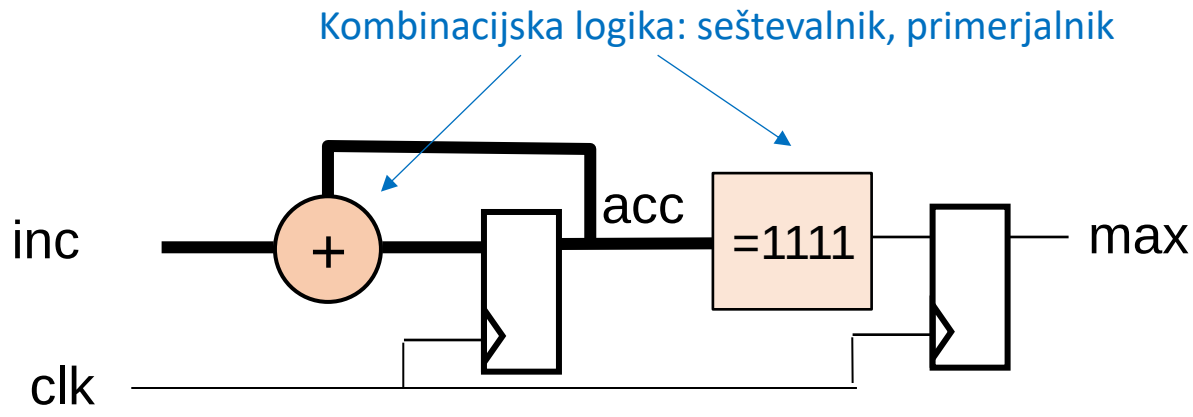
```
acc <= acc + inc
if acc="1111" then
   max <= 1
else
   max <= 0
end
```

korak	**čas**	clk	inc	acc	max
stanje	0	0, 1	F	0	0
izračunaj	T	1	F	F(T+Δ)	0
posodobi	T+Δ	1	F	F	0
čas	2T	0, 1	F	F	0
izračunaj	2T+Δ	1	F	E(2T+Δ)	1
posodobi	2T+Δ	1	F	E	1

Časovni diagram in zgradba vezja



- signal max se postavi na 1 šele ob naslednji fronti ure
  - zakasnitev, ker je izhod iz flip-flopa !



Kdaj opisuje jezik VHDL flip-flope ?

VHDL: prirejanje znotraj pogoja za fronto ure

SHDL: operator <=

flip-flopi

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

```
acc <= acc + inc
if acc="1111" then
   max <= 1
else
   max <= 0
end
```

V SHDL flip-flope opisujejo signali, ki jim priredimo vrednost z <=

Za vsak signal konsistentno uporabljamo le eno vrsto prireditve!

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

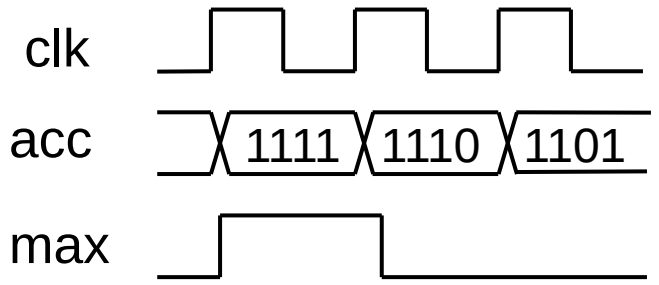
Register

Transfer

Level

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

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

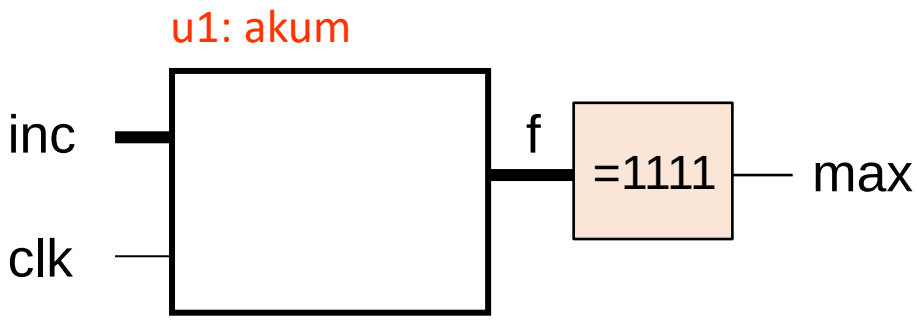


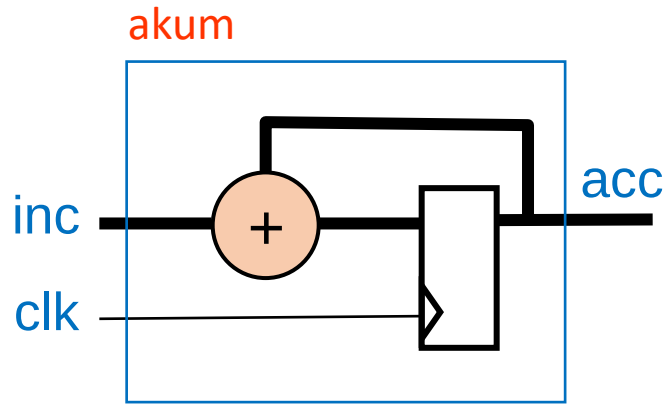

```
acc <= acc + inc;

max = 1 when acc="1111" else 0
```

Strukturni opis vezja

- v vezje vključimo drugo vezje in povežemo priključke (port map)
  - npr. akumulator, ki inkrementira izhod je v ločenem opisu (datoteki)






```
u1: entity work.akum port map (
  clk => clk,
  inc => inc,
  acc => f );

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

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

Sekvečni stroj (avtomat)

stroj s končnim številom stanj (Finite State Machine)

- Moorov avtomat: register stanj, vhodna in izhodna logika

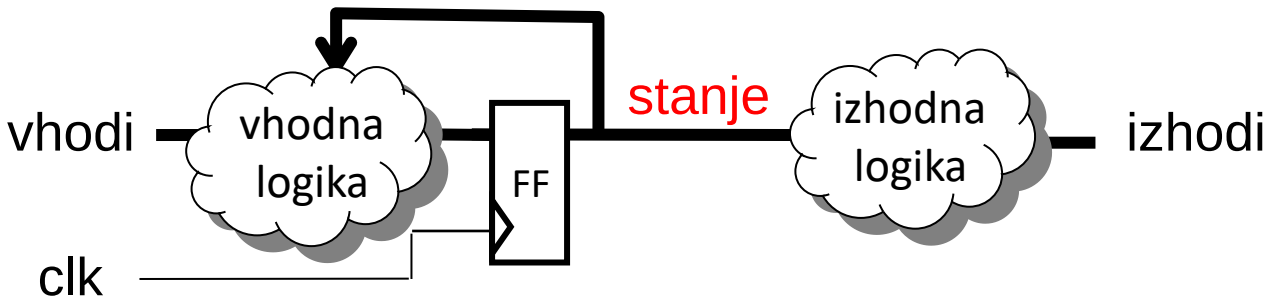
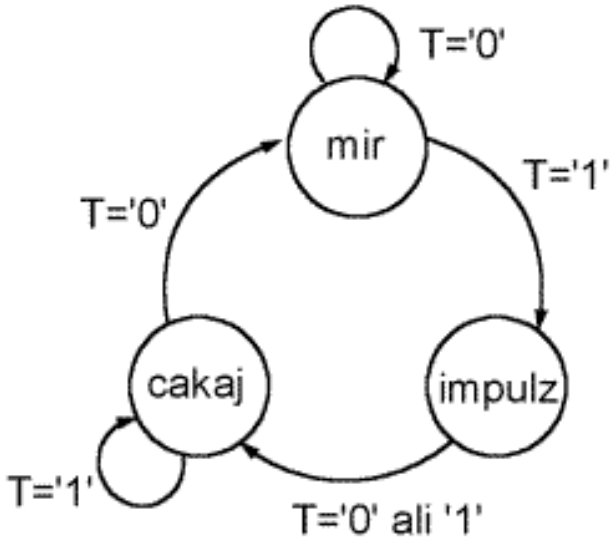
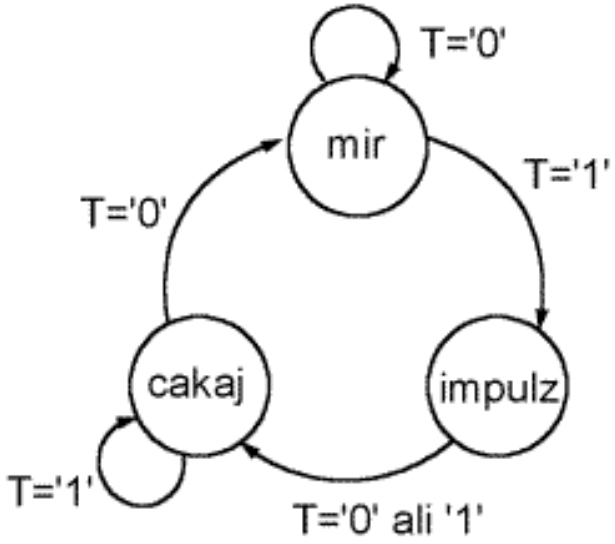


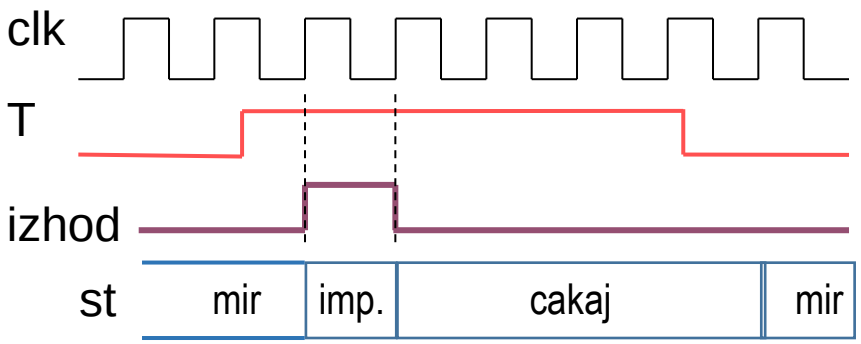
Diagram stanj



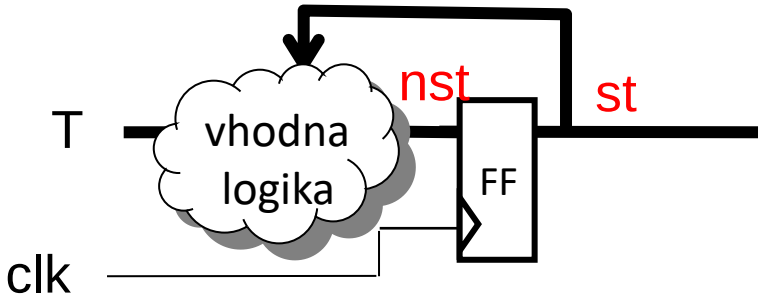
- ob fronti ure kombinacijska logika določa izhode in naslednje stanje, ki je odvisno od vhodov in trenutnega stanja

Diagram stanj in sekvenčno vezje





- sekvenčno vezje s povratno zanko določa naslednje stanje (nst) v odvisnosti od trenutnega stanja (st) in vhoda (T)



Naštevni podatkovni tip

- V jeziku VHDL in SHDL lahko stanja naštejemo s posebnim podatkovnim tipom

```
type stanja is (mir, impulz, cakaj);
signal st, nst: stanja;
```

VHDL **type**

SHDL

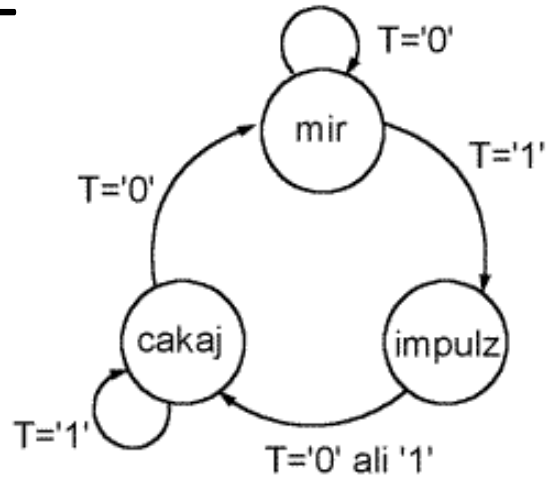
```
st, nst: (mir, cakaj, impulz)
```

- Kodiranja stanj ni potrebno določiti
  - kodiranje izbere programsko orodje za sintezo logike

binarno
mir = 00
impulz = 01
cakaj = 10

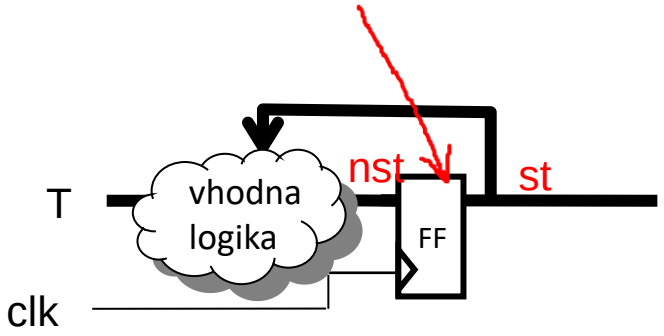
one-hot
mir = 001
impulz = 010
cakaj = 100

*Prehajanje stanj v jeziku VHDL




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

1.
```
FF: process (clk)
begin
  if rising_edge(clk) then
    st <= nst;
  end if;
end process;
```

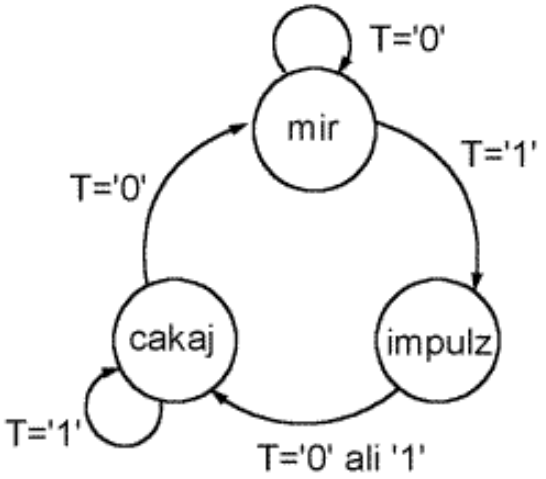


2.
```
preh: process (st, T)
begin
 case st is
  when mir =>
   if T = '1' then
     nst <= impulz;
   else
     nst <= mir;
   end if;
  when impulz =>
     nst <= cakaj;
```

Opis prehajanja stanj v SHDL

1.
```
st <= nst
```

2.
```
if st=mir then
  if (T=1) nst=impulz else nst=mir
elsif st=impulz then
  nst=cakaj
elsif st=cakaj then
  if (T=0) nst=mir else nst=cakaj
else
  nst=mir
end
```



- kompakten opis
  - sekvenčno prirejamo st le takrat, ko se spremeni
  - **else** namesto **if** st=cakaj **then**

```
if st=mir then
  if T=1 then st <= impulz end
elsif st=impulz then
  st <= cakaj
else
  if T=0 then st <= mir end
end
```

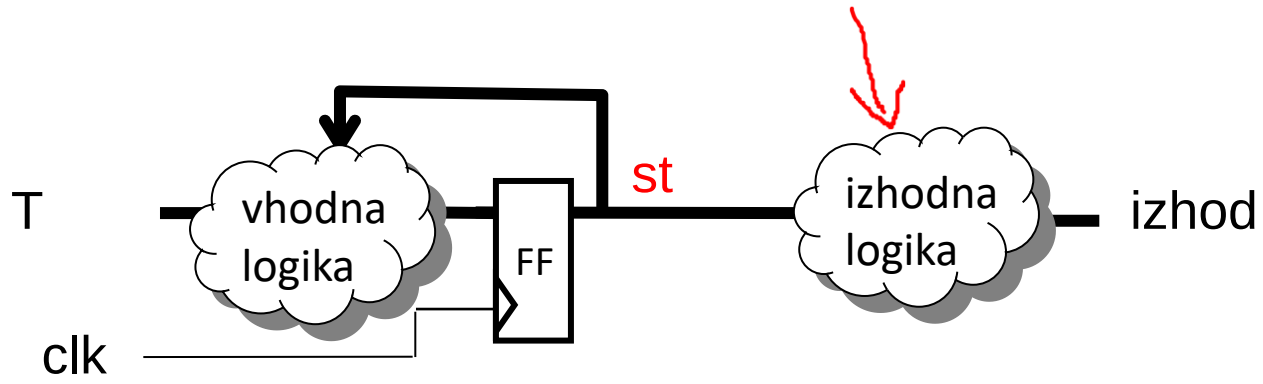
Izhodna logika

Kombinacijska logika za izhode (proces ali izbirni stavek)

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

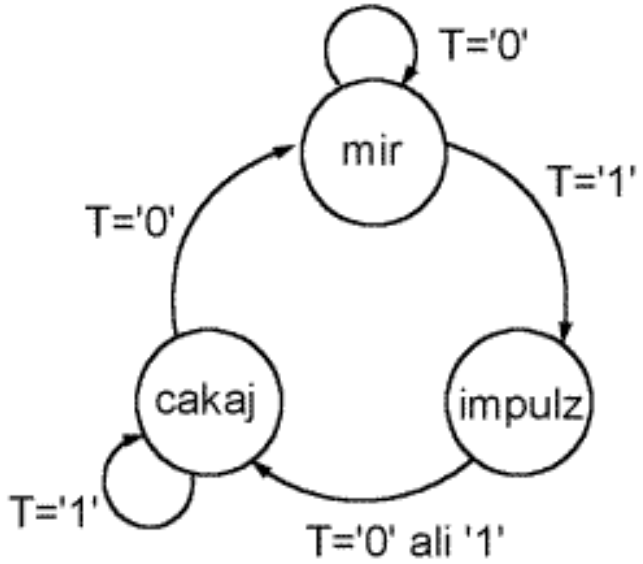
ali

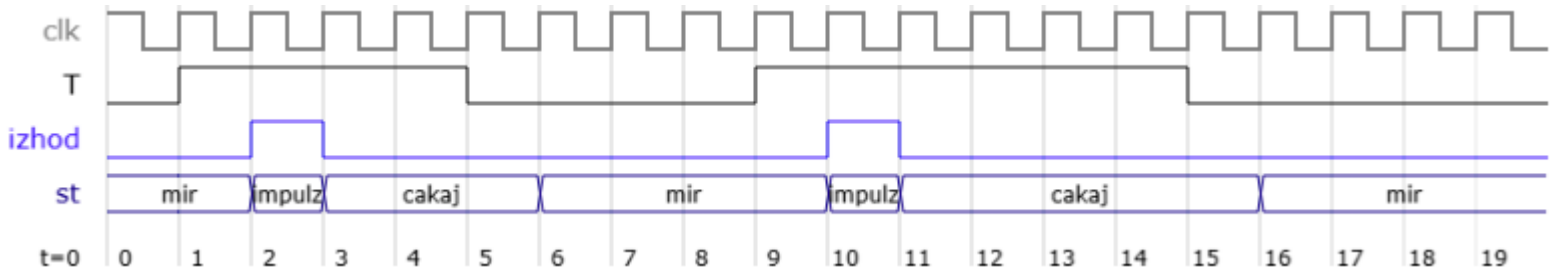
```
izhod <= '1' when st=impulz  else '0';
```



SHDL opis avtomata za tipko

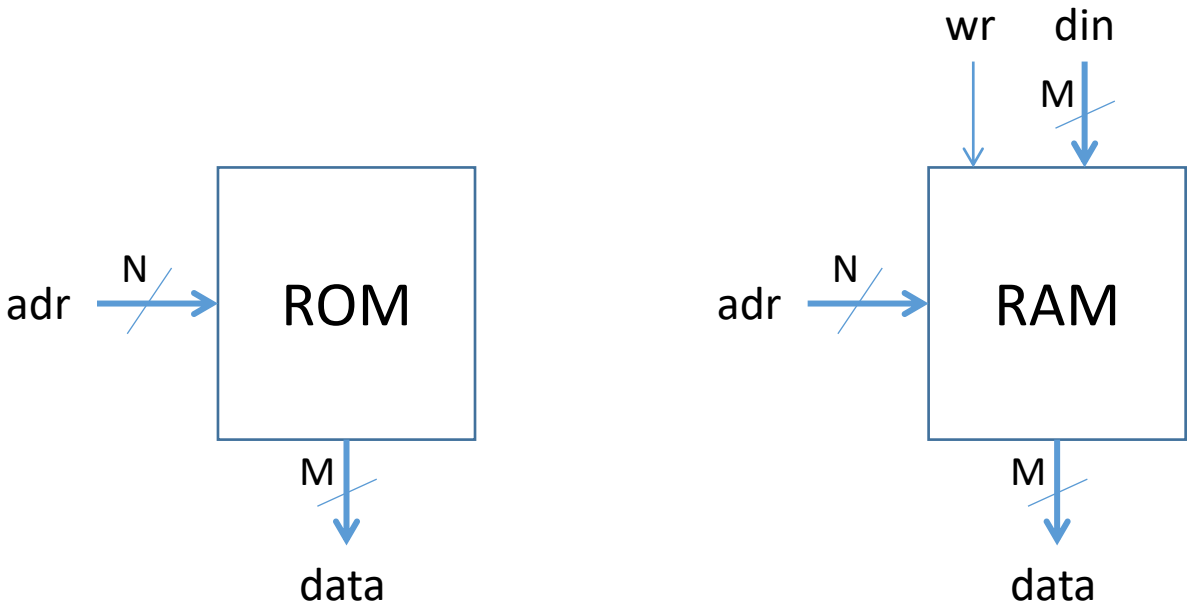
```
entity fsm
 T: in u1;
 izhod: out u1;
 st: (mir, impulz, cakaj)
begin
 if st=mir then
   if T=1 then st <= impulz end
 elsif st=impulz then
   st <= cakaj
 else
   if T=0 then st <= mir end
 end
 izhod = 1 when st=impulz else 0
end
```





Pomnilnik

- Pomnilnik je 2D zbirka celic, ki shranjujejo bite
- N naslovnih bitov (address) in M podatkovnih (data)



Model pomnilnika

deklariramo zbirko

```
type tab is array (0 to 15) of signed(7 downto 0);
signal z: tab;
```

```
z: 16s8
```

opis bralnega pomnilnika (ROM):

```
data <= z(to_integer(address));
```

16x8

```
data = z(address)
```

Program za sintezo generira pomnilnik s sklepanjem (inference)

- signal mora biti zbirka (bitov = vektor, besed ...)
- indeks ne sme biti konstanta
  - če je konstanten, potem predstavlja stavek le eno povezavo, npr. `y=z(1)`

```
y = z(x)
```

Sinteza pomnilnika ROM

Kaj pa predstavlja model:

```
y = z(x)
if p1 then
  r = z(x1)
else
  r = z(x2)
end
```

Pomnilnik z 2 ali 3 vrati ali pa 3 enostavne pomnilnike...

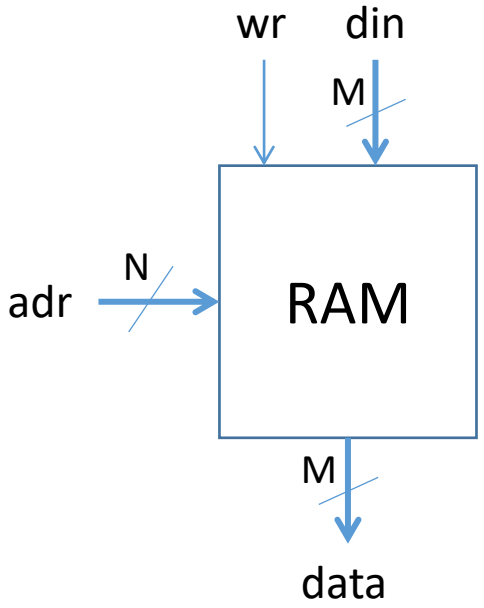
Še en trd oreh:

```
y = z(x) + z(x+1) + z(x+2)
```

- Ali res želimo iz pomnilnika istočasno prebrati 3 vrednosti?
- pomnilnik s 3 vrati, če ni na voljo v zbirki gradnikov, bo moral sintetizator narediti 3 pomnilnike...

Pomnilnik RAM

```
data = m(adr)
if wr=1 then
 m(adr) <= din
end
```



- branje in pisanje v pomnilnik na naslovu adr
  - FPGA BRAM potrebuje uro (sinhroni <=)

Dvo-vratni pomnilnik RAM:

```
dout = m(a1)
if p then
 m(a2) <= din
end
```

Preizkušanje vezij

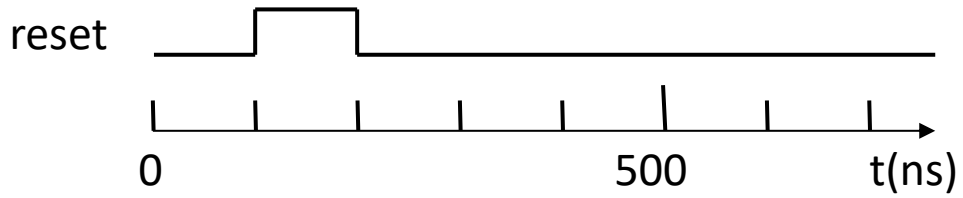
- Preizkušanje na delovni mizi (work bench)
  - inštrumenti: multimeter, logični analizator, osciloskop…
- Testiranje na računalniku – simulacija

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 v jeziku VHDL

- Vezje (entity – architecture), ki nima zunanjih priključkov
  - vse vrednosti signalov definiramo znotraj strukture
  - testirano vezje 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, izpis vrednosti, branje datotek…

```
reset  <= '0', '1' after 100 ns, '0' after 200 ns;
```

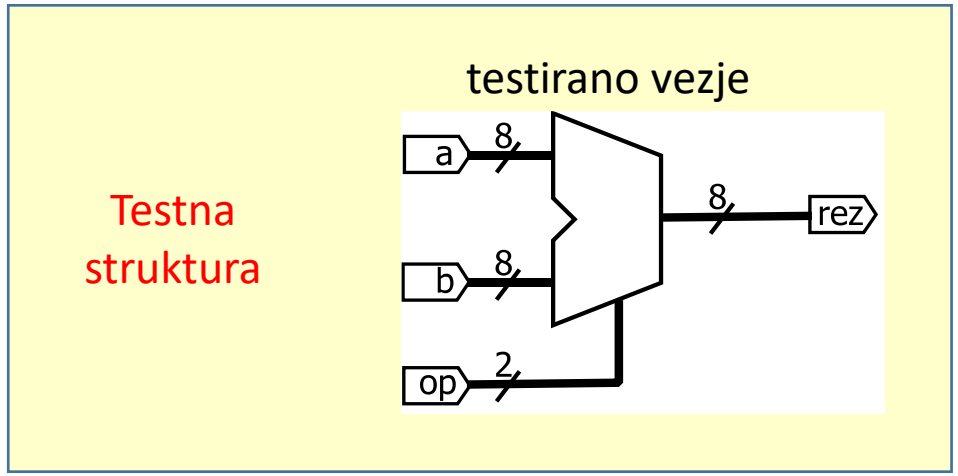


Primer: testna struktura za ALE

- 8-bitna Aritmetično Logična Enota
  - 2-bitni op določa vrsto operacije

```
if op=1 then      rez = a + b
elsif op=2 then   rez = a - b
elsif op=3 then   rez = a and b
else              rez = a – 1
end
```

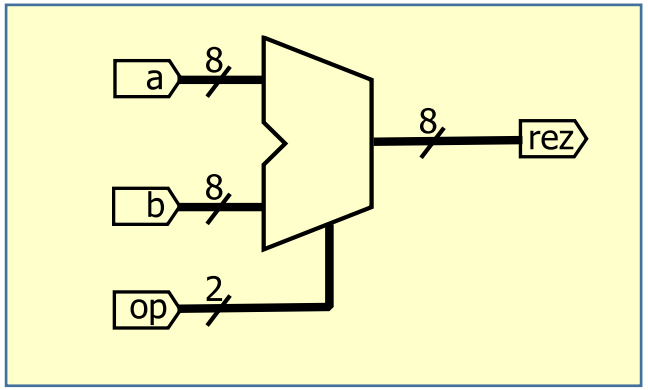
- Testno strukturo povežemo s testiranim vezjem
  - nastavljali bomo a, b in op



Testna struktura za ALE

```
entity ale_tb is
end ale_tb;

architecture opis of ALE_test is
  signal a,b: unsigned(7 downto 0);
  signal op: unsigned(1 downto 0);
  signal rez: unsigned(7 downto 0);
begin
```



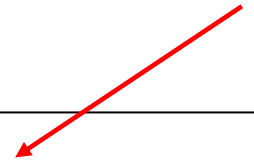
- deklariramo signale in povežemo komponento

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

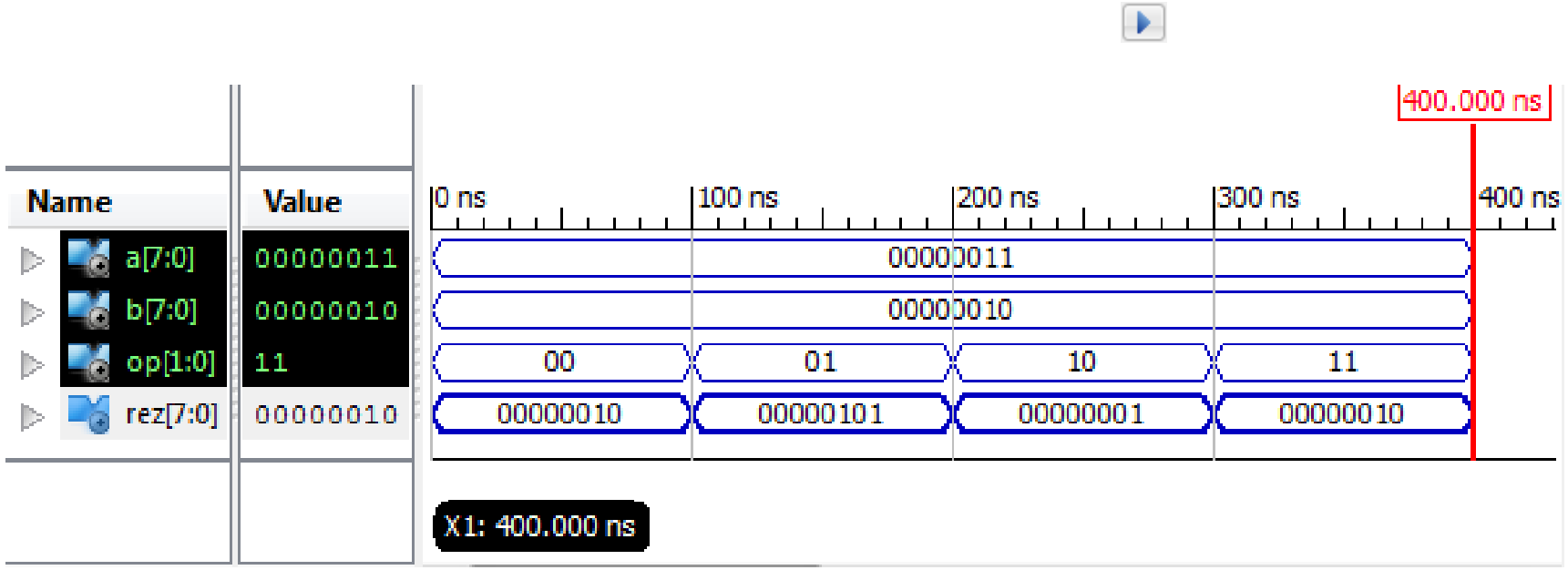
oznaka, Unit Under Test

Testna struktura za ALE (2)

- 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";       -- odstej 1
    wait for 100 ns;
    op <= "01";       -- vsota
    wait for 100 ns;
    op <= "10";       -- razlika
    wait for 100 ns;
    op <= "11";       -- operacija AND
    wait;     -- zamrzni proces, da se ne ponavlja
end process;
```



- 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 !";
...
```

Ura sekvenčnih vezij

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

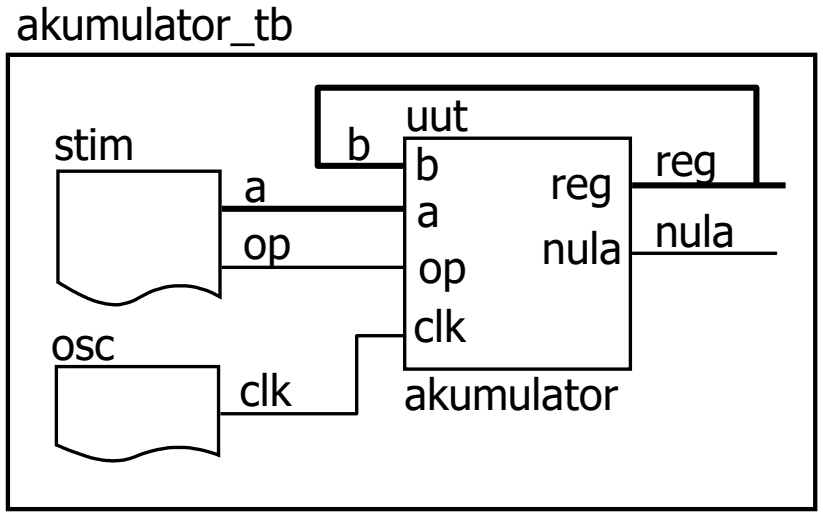
```
GEN_URE: process
  begin
    clk <= '0';
    wait for 500 ns;
    clk <= '1';
    wait for 500 ns;
end process;
```

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 = ALE in izhodni register




```
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;
```

Povzetek

- sinteza vezja
- simulator diskretnih dogodkov
- sekvenčni stroj
- pomnilnik
- testne strukture