Deo 3: Sekvencijalni blokovi

SR lec

```
library ieee;
use ieee.std_logic_1164.all;

entity SR_latch1 is
  port(S,R :in std_ulogic;
       Q,Qbar:buffer std_ulogic);
end entity SR_latch1;

architecture dataflow of SR_latch1 is
begin
  Q<='1' when R='0' else
      '0' when S='0' else  Q;
  Qbar<='1' when S='0' else
         '0' when R='0' else Qbar;
end architecture dataflow;

--------------------------

library ieee;
use ieee.std_logic_1164.all;

entity SR_latch2 is
    port(S,R :in std_ulogic;
         Q,Qbar:out  std_ulogic);
end entity SR_latch2;

architecture behavioural of SR_latch2 is
begin
  po:process (R,S) is
  begin
     case std_ulogic_vector(R,S) is
        when "00" =>
         Q<='1';
         Qbar<='1';

        when "01" =>
         Q<='1';
         Qbar<='0';

        when "10" =>
         Q<='0';
         Qbar<='1';

        when others =>
           null;
     end case;
 end process po;
end architecture behavioural;
----------------------------------

architecture dataflow of SR_latch2 is
begin
  Q<='1' when R='0' else
      '0' when S='0' else
                       unaffected;
```

```
  Qbar<='1' when S='0' else
        '0' when R='0' else
                      unaffected;
end architecture dataflow;
```

D lec

```
library ieee;
use ieee.std_logic_1164.all;

entity D_latch is
  port( D,Enable :in std_ulogic;
                Q:out std_ulogic);
end entity D_latch;

architecture beh of D_latch is
begin
  po:process(D, Enable) is
   begin
    if (Enable='1') then
      Q<=D;
    end if;
  end process po;
end architecture beh;

--------------------------
```

D flipflop

```
library ieee;
use ieee.std_logic_1164.all;

entity D_FF is
  port( D,Clock :in std_ulogic;
              Q :out std_ulogic);
end entity D_FF;

architecture beh of D_FF is
begin
  po : process is
   begin
    wait until (Clock='1');
     Q<=D;
  end process po;
end architecture beh;


--------------------------

architecture beh2 of D_FF is
begin
  po : process(Clock) is
   begin
    if  (Clock='1') then
     Q<=D;
    end if;
```

```
  end process po;
end architecture beh2;

architecture beh3 of D_FF is
begin
  po : process is
   begin
    if  (Clock='1') then
     Q<=D;
    end if;
  wait on Clock;
  end process po;
end architecture beh3;

architecture neg_egde of D_FF is
begin
  po : process is
   begin
    wait until (Clock='0');
     Q<=D;
  end process po;
end architecture neg_edge;
```

Asinhroni set i reset

```
library ieee;
use ieee.std_logic_1164.all;

entity D_FF_R is
  port( D,Clock,Reset :in std_ulogic;
              Q :out std_ulogic);
end entity D_FF_R;

architecture beh of D_FF_R is
begin
  po : process(Clock,Reset) is
   begin
     if (Reset='0') then
        Q<='0';
     elsif (Clock='1' and Clock'event) then
        Q<=D;
     end if;
   end process po;
end architecture beh;


library ieee;
use ieee.std_logic_1164.all;

entity D_FF_RS is
  port( D,Clock,Reset,Set :in std_ulogic;
              Q :out std_ulogic);
end entity D_FF_RS;

architecture beh of D_FF_RS is
begin
  po : process(Clock,Reset,Set) is
   begin
     if (Set='0') then
```

```
        Q<='1';
     elsif  (Reset='0') then
        Q<='0';
     elsif (Clock='1' and Clock'event) then
        Q<=D;
     end if;
   end process po;
end architecture beh;
```

Rising_edge I Falling_edge

```
architecture true_edge of D_FF_R is
begin
  po : process(Clock,Reset) is
   begin
     if (Reset='0') then
        Q<='0';
     elsif (Clock='1' and Clock'last_value='0' and Clock'event) then
        Q<=D;
     end if;
   end process po;
end architecture true_edge;


architecture r_edge of D_FF_R is
begin
  po : process(Clock,Reset) is
   begin
     if (Reset='0') then
        Q<='0';
     elsif rising_edge(Clock) then
        Q<=D;
     end if;
   end process po;
end architecture r_edge;
```

Sinhroni set i reset I clock enable

```
architecture syn_reset of D_FF_R is
begin
  po : process(Clock) is
   begin
     if rising_edge(Clock) then
        if (Reset='0') then
         Q<='0';
        else
         Q<=D;
        end if;
     end if;
   end process po;
end architecture syn_reset;



library ieee;
use ieee.std_logic_1164.all;
```

```
entity D_FF_E is
  port( D,Clock,Enable :in std_ulogic;
                     Q :out std_ulogic);
end entity D_FF_E;

architecture beh of D_FF_E is
begin
  po : process(Clock) is
   begin
     if Rising_edge(Clock) then
         if (Enable='1') then
          Q<=D;
         end if;
     end if;
   end process po;
end architecture beh;


architecture gated_clock of D_FF_E is
signal ce : std_ulogic;
begin
  ce<=Enable and Clock;
  po : process(ce) is
   begin
     if Rising_edge(ce) then
         Q<=D;
     end if;
   end process po;
end architecture beh;
```

Vremenske I logicke provere

```
Assert condition
Report message
Severity level

Assert (Set='1' or Reset='1')
Report "Set and reset are both asserted"
Severity warning;


Assert (not(Set='0' and Reset='0'))



Assert (not(Clk='1' and D'event and not Clk'stable(3 ns)))
Report "Hold time violation"
Severity warning;

---------------------------------------
```

```
library ieee;
use ieee.std_logic_1164.all;

entity D_FF is
 generic(CQ_Delay,SQ_delay,RQ_delay:time:=5 ns;
                              Setup:time:=3ns  );
  port( D,Clock,Reset,Set,Enable :in std_ulogic;
                              Q :out std_ulogic);
Begin
  Assert(not(rising_edge(Clock) and not D'stable(Setup)))
  Report "Setup time violation"
  Severity warning;
end entity D_FF;

architecture beh of D_FF is
begin
  po : process(Clock,Reset,Set) is
   begin
      assert (not(Set='0' and Reset='0'))
      report "Set and Reset are both asserted"
      severity error;
     if (Set='0') then
        Q<='1' after  SQ_delay;
     elsif  (Reset='0') then
        Q<='0'after RQ_delay;
     elsif rising_edge(Clock) then
       if Enable='1' then
          Q<=D after  CQ_delay;
       end if;
     end if;
   end process po;
end architecture beh;
```

JK I T flipflopovi

```
library ieee;
use ieee.std_logic_1164.all;
entity D_FF_R is
  port( D,Clock,Reset :in std_ulogic;
                    Q :out std_ulogic);
end entity D_FF_R;

architecture sig of D_FF_R is
  signal state:std_ulogic;
begin
  po : process(Clock,Reset) is
   begin
     if (Reset='0') then
        state<='0';
     elsif (Clock='1' and Clock'event) then
        state<=D;
     end if;
   end process po;
  Q<=state;
  Qbar<= not state;
end architecture sig;
```

```
architecture var of D_FF_R is
begin
  po : process(Clock,Reset) is
   variable state : std_ulogic;
   begin
     if (Reset=’0’) then
        state:=’0’;
     elsif (Clock=’1’ and Clock’event) then
        state:=D;
     end if;
   end process po;
  Q<=state;
  Qbar<= not state;
end architecture var;

----------------------------
library ieee;
use ieee.std_logic_1164.all;

entity JK_FF is
  port( J,K,Clock,Reset :in std_ulogic;
                    Q :out std_ulogic);
end entity D_FF_R;

architecture sig of JK_FF is
  signal state:std_ulogic;
begin
  po : process(Clock,Reset) is
   begin
     if (Reset=’0’) then
        state<=’0’;
     elsif rising_edge(Clock) then
       case std_ulogic_vector(J,K)  is
          when “11” => state<=not state;
          when “10” => state<=’1’;
          when “01” => state<=’0’;
          when others => null;
       end case;
     end if;

   end process po;
  Q<=state;
  Qbar<= not state;
end architecture sig;

architecture var of T_FF is
begin
  po : process(Clock,Reset) is
   variable state : std_ulogic;
   begin
     if (Reset=’0’) then
        state:=’0’;
     elsif (Clock=’1’ and Clock’event) then
        if T=’1’ then
             state:=not state;
        end if;
     end if;
   end process po;
  Q<=state;
  Qbar<= not state;
```

```
end architecture var;
```

Registri i SHIFT registri

```
--Registar sa n bitova
library ieee;
use ieee.std_logic_1164.all;

entity reg is
  generic(n:natural:=4);
  port(   D:  in std_logic_vector(n-1 downto 0);
          Clock,Reset : in std_logic;
          Q : out std_logic_vector(n-1 downto 0));
end entity reg;

architecture beh of reg is
begin
  po : process(Clock,Reset) is
   begin
     if (Reset='0') then
        Q<=(others=>'0');
     elsif rising_edge(Clock) then
        Q<=D;
     end if;
   end process po;
end architecture beh;


-- shift registar
library ieee;
use ieee.std_logic_1164.all;

entity sipo is
  generic(n:natural:=8);
  port(   A,Clock : in std_logic;
          Q : out std_logic_vector(n-1 downto 0));
end entity sipo;

architecture beh of sipo is
begin
  po : process(Clock) is
   variable reg: std_logic_vector(n-1 downto 0));
   begin
     if rising_edge(Clock) then
        reg:=reg(n-2 downto 0)&A;
     end if;
     Q<=reg;
   end process po;
end architecture beh;
```

```
-- Sinhroni binarni brojac

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric.std.all;

entity counter is
  generic(n:natural:=4);
  port(   Reset,Clock : in std_logic;
          Q : out std_logic_vector(n-1 downto 0));
end entity counter;

architecture beh of counter is
begin
  po : process(Clock,Reset) is
   variable cnt: std_logic_vector(n-1 downto 0));
   begin
     if Reset='1' then
         cnt:=(others=>'0');
     elsif rising_edge(Clock) then
         cnt:=cnt+1;
     end if;
     Q<=cnt;
   end process po;
end architecture beh;


-- Opis asinhronog brojaca kao niza T flip flopova


architecture ripple of counter is
 component T_FF is
   port(  T,Clock,Reset:in std_logic;
               Q, Qbar :out std_logic);
 end component;

 signal carry:std_logic_vector(n downto 0);

begin
  carry(0)<=Clock;
  g0: for i in 0 to n-1 generate
     ti: T_FF port map ('1',carry(i),reset,count(i),carry(i+1));
  end generate g0;
end architecture ripple;

-- Opis T flip flopa sa kasnjenjem

architecture delayed of T_FF is
begin
  po : process(Clock,Reset) is
   variable state : std_ulogic;
   begin
     if (Reset='0') then
        state:='0';
     elsif (Clock='1' and Clock'event) then
        if T='1' then
             state:=not state;
```

```
        end if;
     end if;
   end process po;
  Q <= state after 5ns;
  Qbar<= not state after 5ns;
end architecture var;
```

ROM memorija

```
library ieee;
use ieee.std_logic_1164.all;

entity rom16x7 is
    port(   address : in range 0 to 15;
               data : out std_logic_vector(6 downto 0));
end entity rom16x7;

architecture beh of rom16x7 is
  type rom_array is array(0 to 15) of std_logic_vector(6 downto 0);
  constant rom: rom_array:=(
         "1110111" ,
         "0010010",
         "1011101",
         "1011011",
         "0111010",
         "1101011",
         "1101111",
         "1010010",
         "1111111",
         "1111011",
         "1101101",
         "1101101",
         "1101101",
         "1101101",
         "1101101",
         "1101101",
         "1101101");
begin
    data<=rom(address);
end architecture beh;
```

staticka RAM memorija

```
library ieee;
use ieee.std_logic_1164.all;

entity RAM16x8 is
        port(address  : in range 0 to 15;
                data  : inout std_logic_vector(7 downto 0);
             cs,we,oe : in std_logic );
end entity RAM16x8;

architecture beh of RAM16x8 is
begin
  po : process(address,cs,we,oe) is
    type ram_array is array(0 to 15) of std_logic_vector(7 downto 0);
    variable mem: ram_array;
```

```
  begin

  data<=(others=>'Z');
  if cs='0' then
      if oe='0' then
        Data<=mem(adress);
      elsif we='0' then
        mem(address):=data;
      end if;
  end if;
end process po;

end architecture beh;
```

Opisivanje SEKVENCERA

```
library ieee;
use ieee.std_logic_1164.all;
entity traffic is
        port( clock,timed,car: in std_logic;
              start_timer,minor_green,major_green : out std_logic));
end entity traffic;

architecture asm1 of traffic is
  type state_type is (G,R);
  signal present_state,next_state: state_type;
begin
 proc1: process(clock) is
 begin
   if rising_edge(clock) then
         present_state<=next_state;
   end if;
 end process proc1;

 proc2: process(car,timed,present_state) is
 begin
  start_timer<='0';
  case present_state is
    when G =>
      major_green<='1';
      minor_green<='0';
      if (car='1') then
         start_timer<='1';
         next_state<=R;
      else
         next_state<=G;
      end if;
    when R=>
      major_green<='0';
      minor_green<='1';
      if (timed='1') then
        next_state<=G;
      else
        next_state<=R;
      end if;
  end case;
end process proc2;
```

```
end architrcture asm1;

architecture asm2 of traffic is
  type state_type is (G,R);
  signal present_state,next_state: state_type;
begin
 proc1: process(clock) is
 begin
   if rising_edge(clock) then
         present_state<=next_state;
   end if;
 end process proc1;

 proc2: process(car,timed,present_state) is
 begin
    case present_state is
    when G =>
      if (car='1') then
         next_state<=R;
      else
         next_state<=G;
      end if;
    when R=>
      if (timed='1') then
        next_state<=G;
      else
        next_state<=R;
      end if;
  end case;
end process proc2;

proc3: process(car, present_state) is
 begin
  start_timer<='0';
  if (present_state=G) then
      major_green<='1';
      minor_green<='0';
      if (car='1') then
        start_timer<='1';
      end if;
  else
      major_green<='0';
      minor_green<='1';
  end if;
end process proc3;

end architrcture asm2;
```