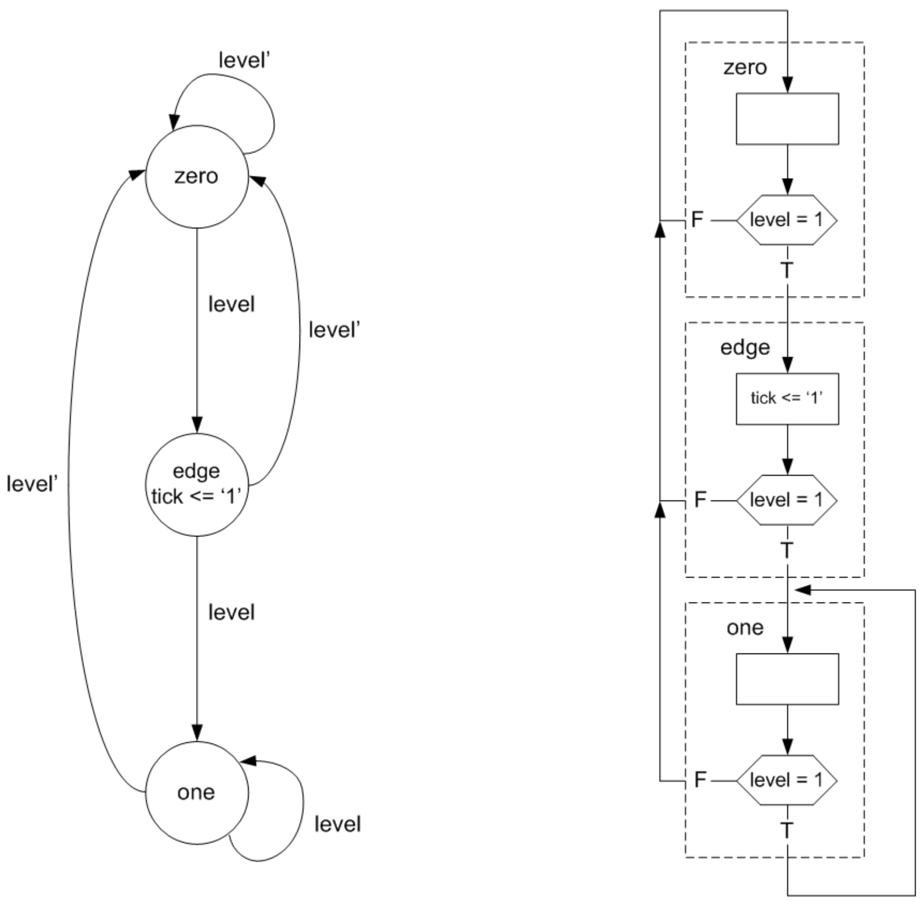
Primjer: **Rising-edge detector**

Projektovati kolo koje obavlja funkciju detektora rastuće ivice signala na ulazu. Po detektovanju rastuće ivice, kolo generiše kratak impuls (**tick**). Realizovati isto kolo na bazi *Moore*-ovog i na bazi *Mealy*-jevog automata.

Rješenje:

Realizacija rising-edge detector-a na bazi *Moore*-ovog automata

Dijagram stanja i ASM dijagram kola dat je na slici 1.



Slika 1 Dijagram stanja i ASM dijagram kola za detekciju rastuće ivice ulaznog signala na bazi *Moore*-ovog automata

VHDL kod moguće realizacije **rising-edge detector**-a na bazi *Moore*-ovog automata dat je u listingu 1.

Listing 1 – Rising-edge detector na bazi Moore-ovog automata

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

entity egde_detector is
      port
             (
                    clk, rst : in std_logic;
                    level : in std_logic;
                    tick : out std_logic
             );
end egde_detector;

architecture arch of egde_detector is
       type state_type is (zero, edge, one);
       signal state_reg, state_next : state_type;
begin
       process(clk, rst)
       begin
             if(rst = '1') then
                    state_reg <= zero;
             elsif(clk'event and clk = '1') then
                    state_reg <= state_next;
             end if;
       end process;

       process(state_reg, level)
       begin
             state_next <= state_reg;
             tick <= '0';
             case state_reg is
                    when zero =>
                           if (level = '1') then
                                  state_next <= edge;
                           end if;
                    when edge =>
                           tick <= '1';
                           if (level = '1') then
                                  state_next <= one;
                           else
                                  state_next <= zero;
                           end if;
                    when one =>
                           if (level = '0') then
                                  state_next <= zero;
                           end if;
             end case;
       end process;
end arch;
```

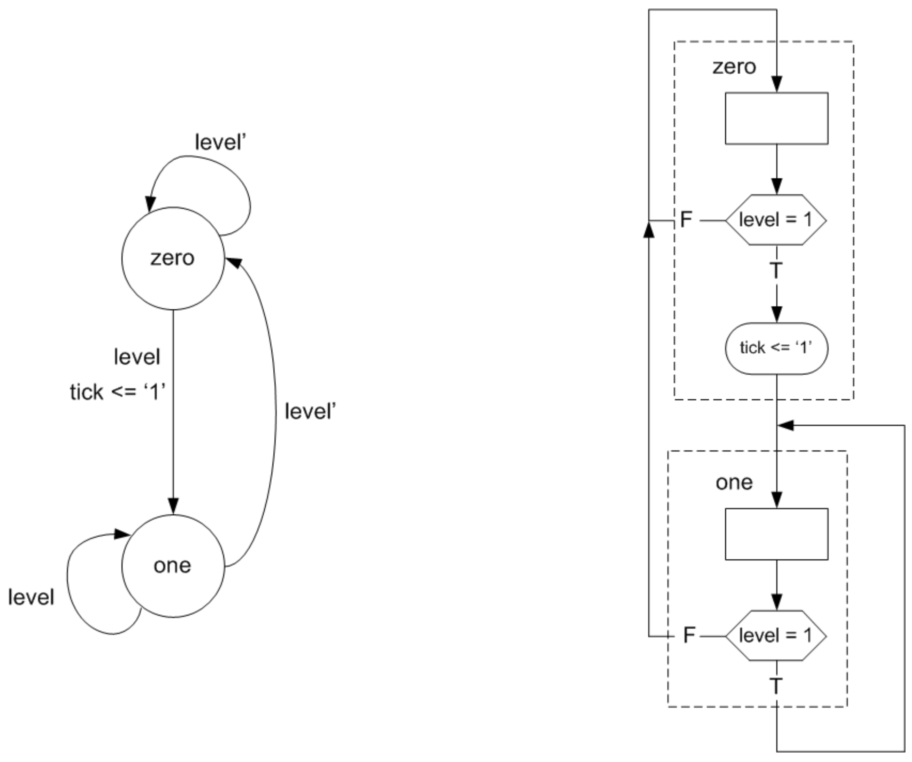
Kreirati VHDL **testbench** (pogledati vježbe 4). Izvršiti simulaciju rada kola. Prikazati rezultate simulacije.

<u>Realizacija rising-edge detector-a na bazi *Mealy*-ovog automata</u>

Dijagram stanja i ASM dijagram kola dat je na slici 2.

VHDL kod moguće realizacije **rising-edge detector**-a na bazi *Mealy*-ovog automata dat je u listingu 2.

Kreirati VHDL **testbench** (pogledati vježbe 4). Izvršiti simulaciju rada kola. Prikazati rezultate simulacije.



Slika 2 Dijagram stanja i ASM dijagram kola za detekciju rastuće ivice ulaznog signala na bazi *Moore*-ovog automata

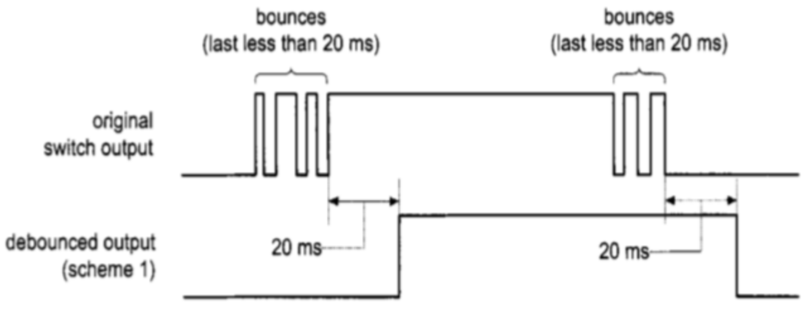
Listing 2 – Rising-edge detector na bazi Mealy-ovog automata

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

entity edge_detector is
       port
              (
                    clk, rst : in std_logic;
                    level : in std_logic;
                    tick : out std_logic
              );
end edge_detector;

architecture arch of edge_detector is
       type state_type is (zero, one);
       signal state_reg, state_next : state_type;
begin
       process(clk, rst)
              begin
              if (rst = '1') then
                    state_reg <= zero;
              elsif (clk'event and clk = '1') then
                    state_reg <= state_next;
              end if;
       end process;

       process(state_reg, level)
       begin
              state_next <= state_reg;
              tick <= '0';
              case state_reg is
                    when zero =>
                           if (level = '1') then
```

```
                                  state_next <= one;
                                  tick <= '1' ;
                           end if;
                    when one =>
                           if (level = '0') then
                                  state_next <= zero;
                           end if;
             end case;
      end process;
end arch;
```

Primjer: **Debouncing circuit**

Projektovati **debouncing** kolo koje filtrira **glitch**-eve koji se generišu prilikom aktivacije mehaničkih prekidača i tastera na razvojnoj ploči, slika 3.



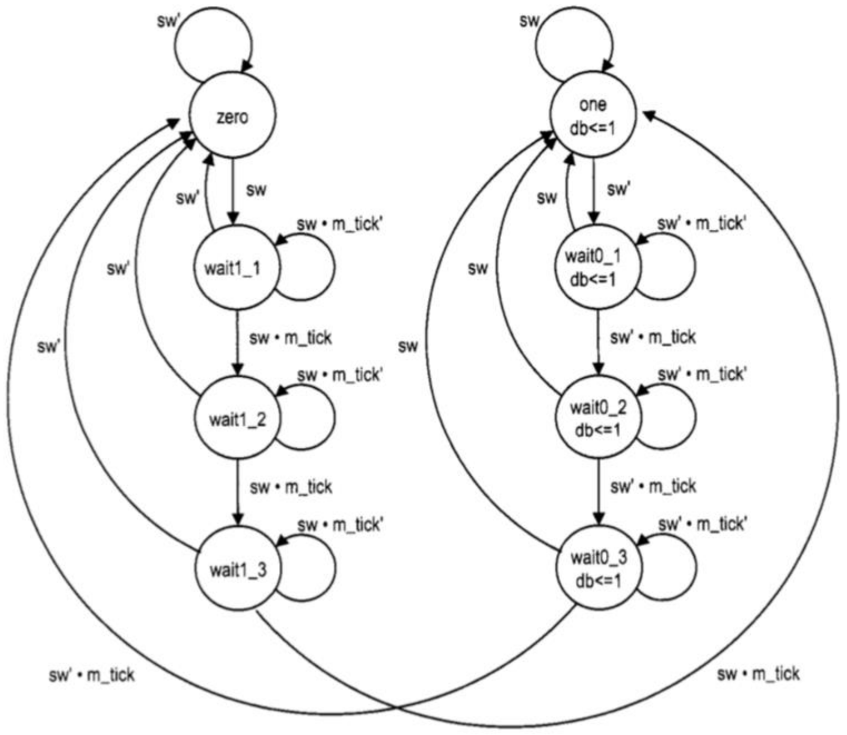
Slika 3

Rješenje:

Dijagram stanja i ASM dijagram kola dat je na slici 4.

VHDL kod moguće realizacije **debouncing** kola dat je u listingu 3.

Kreirati VHDL **testbench** (pogledati vježbe 4). Izvršiti simulaciju rada kola. Prikazati rezultate simulacije.



Slika 4 Dijagram stanja **debouncing** kola

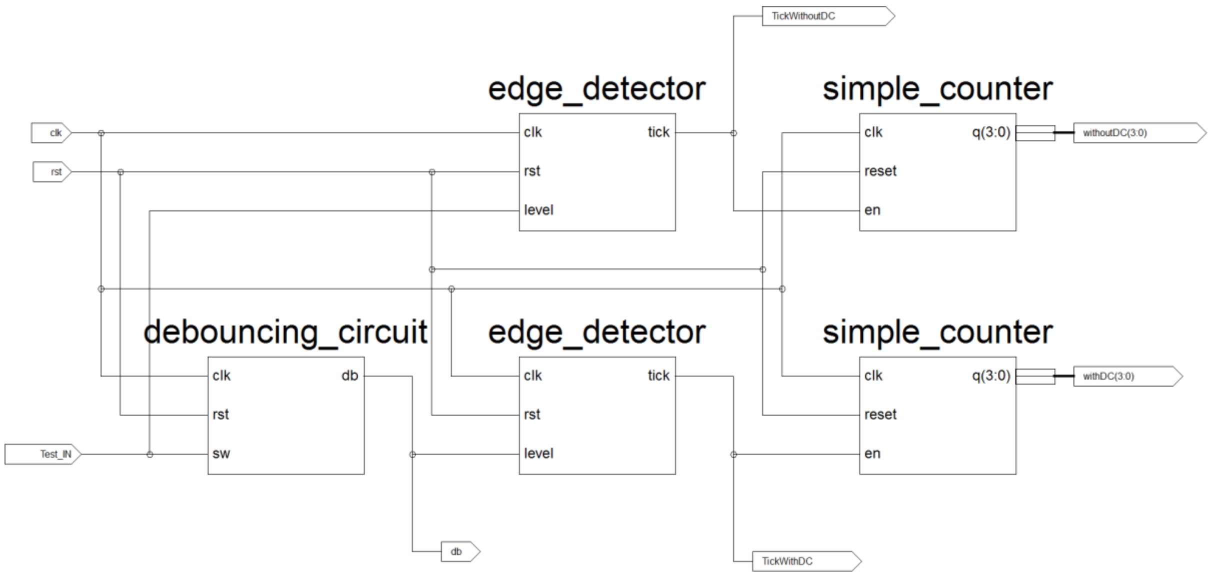
Listing 3 – Debouncing circuit

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

entity debouncing_circuit is
	port
		(
			clk, rst : in std_logic;
			sw : in std_logic;
			db : out std_logic
		);
end debouncing_circuit;

architecture arch of debouncing_circuit is

	constant N : integer := 19;
	signal q_reg, q_next : unsigned(N-1 downto 0);
	signal m_tick : std_logic;
	type eg_state_type is (zero, wait1_1, wait1_2, wait1_3, one, wait0_1,
wait0_2, wait0_3);
	signal state_reg, state_next : eg_state_type;
begin
-------  counter to generate 10 ms tick -------
	process(clk, rst)
	begin
		if (rst = '1') then
			q_reg <= (others=>'0');
		elsif (clk'event and clk = '1') then
```

```
                    q_reg <= q_next;
            end if;
    end process;
    -- next-state logic --
    q_next <= q_reg + 1;
    -- output logic --
    m_tick <= '1' when q_reg = 0 else
                        '0';
-------  debouncing circuit FSM -------
    process(clk, rst)
    -- state register --
    begin
            if(rst = '1') then
                    state_reg <= zero;
            elsif(clk'event and clk = '1') then
                    state_reg <= state_next;
            end if;
    end process;
    -- next-state / output logic --
    process(state_reg, sw, m_tick)
    begin
            state_next <= state_reg;
            db <= '0';
            case state_reg is
                    when zero =>
                            if(sw = '1') then
                                    state_next <= wait1_1;
                            end if;
                    when wait1_1 =>
                            if(sw = '0') then
                                    state_next <= zero;
                            else
                                    if(m_tick = '1') then
                                            state_next <= wait1_2;
                                    end if;
                            end if;
                    when wait1_2 =>
                            if(sw = '0') then
                                    state_next <= zero;
                            else
                                    if(m_tick ='1') then
                                            state_next <= wait1_3;
                                    end if;
                            end if;
                    when wait1_3 =>
                            if(sw = '0') then
                                    state_next <= zero;
                            else
                                    if(m_tick = '1') then
                                            state_next <= one;
                                    end if;
                            end if;
                    when one =>
                            db <= '1';
                            if(sw = '0') then
                                    state_next <= wait0_1;
                            end if;
                    when wait0_1 =>
                            db <= '1';
                            if(sw = '1') then
                                    state_next <= one;
                            else
                                    if(m_tick = '1') then
                                            state_next <= wait0_2;
                                    end if;
```

```
                end if;
            when wait0_2 =>
                db <= '1';
                if(sw = '1') then
                    state_next <= one;
                else
                    if(m_tick = '1') then
                        state_next <= wait0_3;
                    end if;
                end if;
            when wait0_3 =>
                db <= '1';
                if(sw = '1') then
                    state_next <= one;
                else
                    if(m_tick = '1') then
                        state_next <= zero;
                    end if;
                end if;
            end case;
        end process;
end arch;
```

U cilju verifikacije rada sistema, razviti testno kolo kako je prikazano na slici 5 (pogledati vježbe 4). **Test_IN** ulazni signal je izlaz tastera, izlaze **withoutDC** i **withDC** prikazati pomoću odgovarajućih dioda na razvojnoj FPGA ploči, izlaze **TickWithoutDC**, **TickWithDC** i **db** posmatrati pomoću osciloskopa. VHDL kod **simple_counter** kola dat je u listingu 4.



Slika 5

Izvršiti procese **synthesize**, **translate**, **map** i **place** & **route**.

Izvršiti implementaciju kola uz pomoć **Spartan-3E Starter Kit** razvojne platforme (pogledati uputstvo u okviru vježbi 2) i verifikovati rad kola.

Listing 4 – Simple counter

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

entity simple_counter is
generic(N: integer := 4);
       port
       (
              clk, reset: in std_logic;
              en : in std_logic;
              q: out std_logic_vector(N-1 downto 0)
       );
       end simple_counter;
architecture arch of simple_counter is
       signal r_reg: unsigned(N-1 downto 0);
       signal r_next: unsigned(N-1 downto 0);
begin
       process(clk, reset)
       begin
              if (reset = '1') then
                     r_reg <= (others=>'0');
              elsif (clk' event and clk = '1') then
                     r_reg <= r_next;
              end if;
       end process;
       r_next <=
                            r_reg + 1 when en = '1' else
                            r_reg;
       q <= std_logic_vector(r_reg);
end arch;
```