# vaja 03

# Generični moduli, komponente, simulacija

Digitalno načrtovanje – laboratorijske vaje

asistent: Nejc Ilc

# Splošni gradniki - generic

- Stavek `generic` uporabimo, ko želimo opisati splošne, parametrizirane gradnike, recimo n-bitni števec.
- Ob deklaraciji navedemo privzeto vrednost, ob instanciaciji pa dokončno.

```
entity counter is

port (
    clock: in  STD_LOGIC;
    reset: in  STD_LOGIC;
    value: out
         STD_LOGIC_VECTOR (3 downto 0);
 );
end counter;
```

```
entity counter is
generic (n: integer := 8);
port (
    clock: in  STD_LOGIC;
    reset: in  STD_LOGIC;
    value: out
         STD_LOGIC_VECTOR (n-1 downto 0);
 );
end counter;
```

# Komponente

- Želimo, da ima naš projekt pregledno in modularno strukturo.
- Želimo večkratno uporabo že izdelanih (splošnih) gradnikov.
  - Primer: opis n-bitnega števca uporabimo enkrat za 4-bitni, drugič za 8-bitni števec.
- Izkoristimo koncept komponent, ki omogoča vstavljanje in povezovanje že definiranih gradnikov v drugih gradnikih.

# Primer

- Zgradimo števec, ki se bo povečeval vsako sekundo.
- Opis bomo razdelili v tri datoteke (module):
  - `prescaler`
    - vhodi: `clock`, `reset`, `limit`
    - izhod: `clock_enable`
  - `counter`
    - vhodi: `clock`, `reset`, `clock_enable`
    - izhod: `value`
  - `top`
    - povezuje komponenti `prescaler` in `counter` ter definira zunanji vmesnik

# Modul prescaler

```
entity prescaler is
  generic (
    width: integer := 8; -- the width of a prescaler counter
  );
  port (
    clock:        in  std_logic;
    reset:        in  std_logic;
    limit:        in  integer;
    clock_enable: out std_logic;
  );
end prescaler;
```

# Modul counter

```
entity counter is
  generic (
    width: integer := 4;
  );
  port (
    clock:        in  std_logic;
    reset:        in  std_logic;
    clock_enable: in  std_logic;
    count_up:     in  std_logic;
    count_down:   in  std_logic;
    value:        out signed(width-1 downto 0);
  );
end counter;
```

# Modul top

- Zunanji modul ne more biti generičen.

```
entity top is
  port (
    CLK100MHZ: in  std_logic;
    BTNC:      in  std_logic;
    SW_0:      in  std_logic;
    SW_1:      in  std_logic;
    LED:       out signed(3 downto 0);
  );
end top;
```

# Povezovanje modulov

- Gradnik, ki ga želimo uporabiti znotraj drugega gradnika, imenujemo komponenta.
- Deklaracija (pred begin v arhitekturi)

```
component ime_komponente
    port ( ime: smer tip_signala ...);
end component;
```

- Povezovanje

```
<oznaka>: ime_komponente
port map (
    ime_signala_kom1 => ime_signala_vezje1,
    ime_signala_kom2 => ime_signala_vezje2,
    ...
);
```

# Primer: deklaracije

```
-- Constants
constant f_clk_sys: integer := 100e6; -- 100 MHz
constant f_clk:     integer := 1; -- 1 Hz
constant cnt_width: integer := 4;
constant pr_width:  integer := 27;
constant pr_limit:  integer := f_clk_sys / f_clk - 1;

-- Internal signals
signal CE: std_logic := '0';
```

```
-- Components
component counter is generic (width: integer := 4);
    port (
        clock:        in  std_logic;
        reset :       in  std_logic;
        clock_enable: in  std_logic;
        count_up:     in  std_logic;
        count_down:   in  std_logic;
        value:        out signed (width-1 downto 0));
end component;
component prescaler is generic (width : integer := 27);
     port (
        clock:        in  std_logic;
        reset:        in  std_logic;
        limit:        in  integer;
        clock_enable: out std_logic);
end component;
```

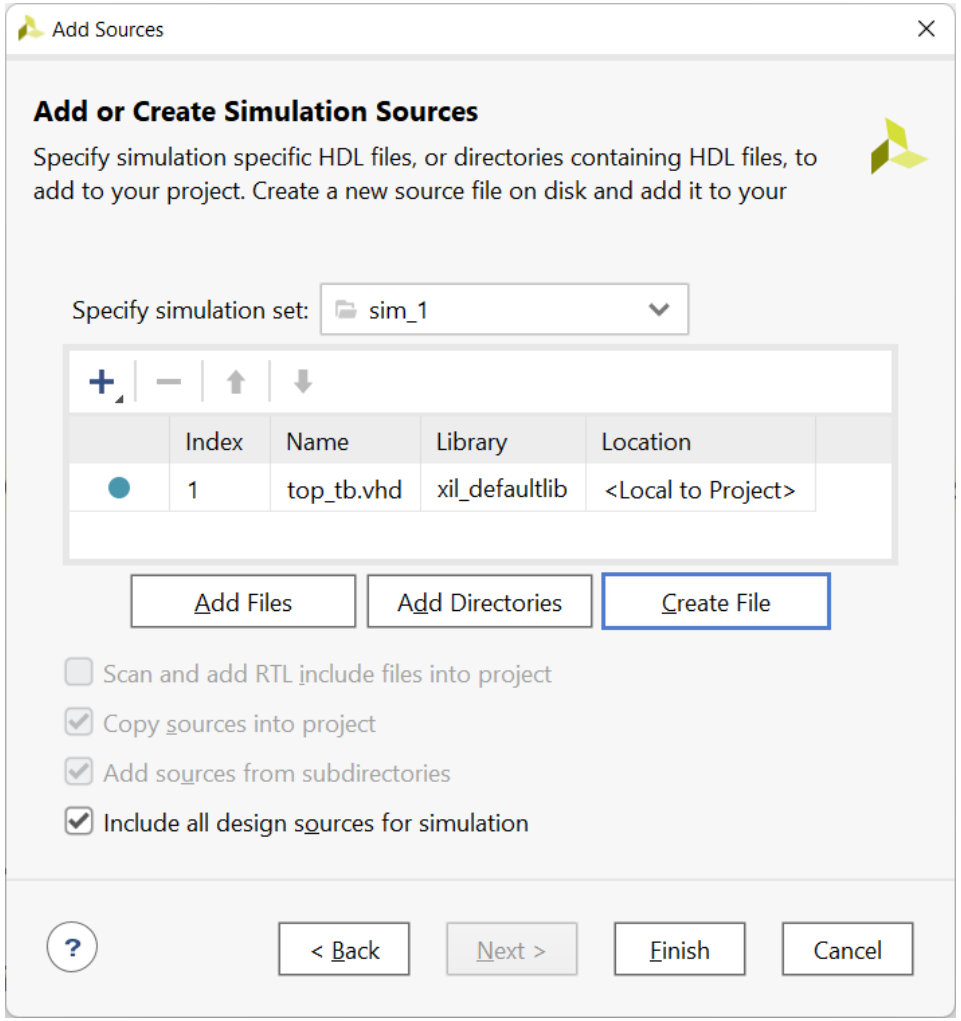
# Primer: instanciacija

```
pr: prescaler
generic map (
  width => pr_width)
port map (
  clock        => CLK100MHZ,
  reset        => BTNC,
  clock_enable => CE,
  limit        => pr_limit
);
```

```
cnt: counter
generic map (
  width => cnt_width)
port map (
  clock        => CLK100MHZ,
  reset        => BTNC,
  clock_enable => CE,
  count_up     => SW_0,
  count_down   => SW_1,
  value        => LED
);
```

# Simulacija

- Pred sintezo preverimo obnašanje vezja: "Behavioral simulation"
- Napisali bomo svoj scenarij testiranja, t.i. "test bench".
- Definiramo dražljaje na vhodu v vezje in opazujemo potek izhodov.
- File → Add Sources … → Add or Create Simulation Sources → Create File → <ime_modula>_tb.vhd



# Simulacija: "test bench"

- Modul za simulacijo nima zunanjih vhodov ali izhodov.
- V arhitekturi:
  - deklariramo komponento, ki predstavlja testirani modul (UUT – unit under test),
  - instanciramo komponento,
  - opišemo dražljaje, pri tem uporabimo procese
    - proces za generiranje ure,
    - proces za ostale dražljaje (stimuluse).

# Simulacija: "test bench" (2)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity top_tb is
end entity;

architecture Behavioral of top_tb is
    constant clock_period: time := 10 ns;
    signal clock:         std_logic := '0';
    signal reset:         std_logic := '0';
    signal count_up:      std_logic := '0';
    signal count_down:    std_logic := '0';
    signal counter_value: signed (3 downto 0);
```

```
component top is
  port ( CLK100MHZ: in  std_logic;
         BTNC:      in  std_logic;
         SW_0:      in  std_logic;
         SW_1:      in  std_logic;
         LED:       out signed (3 downto 0));
end component;

begin
  uut: top
  port map( CLK100MHZ => clock,
            BTNC      => reset,
            SW_0      => count_up,
            SW_1      => count_down,
            LED       => counter_value);
```

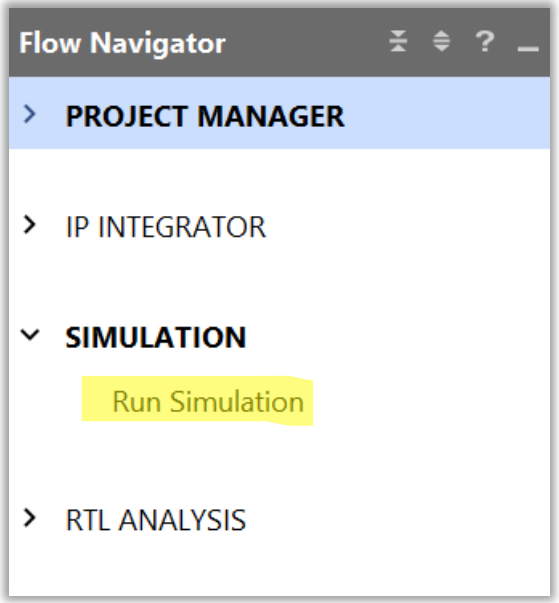
# Simulacija: "test bench" (3)

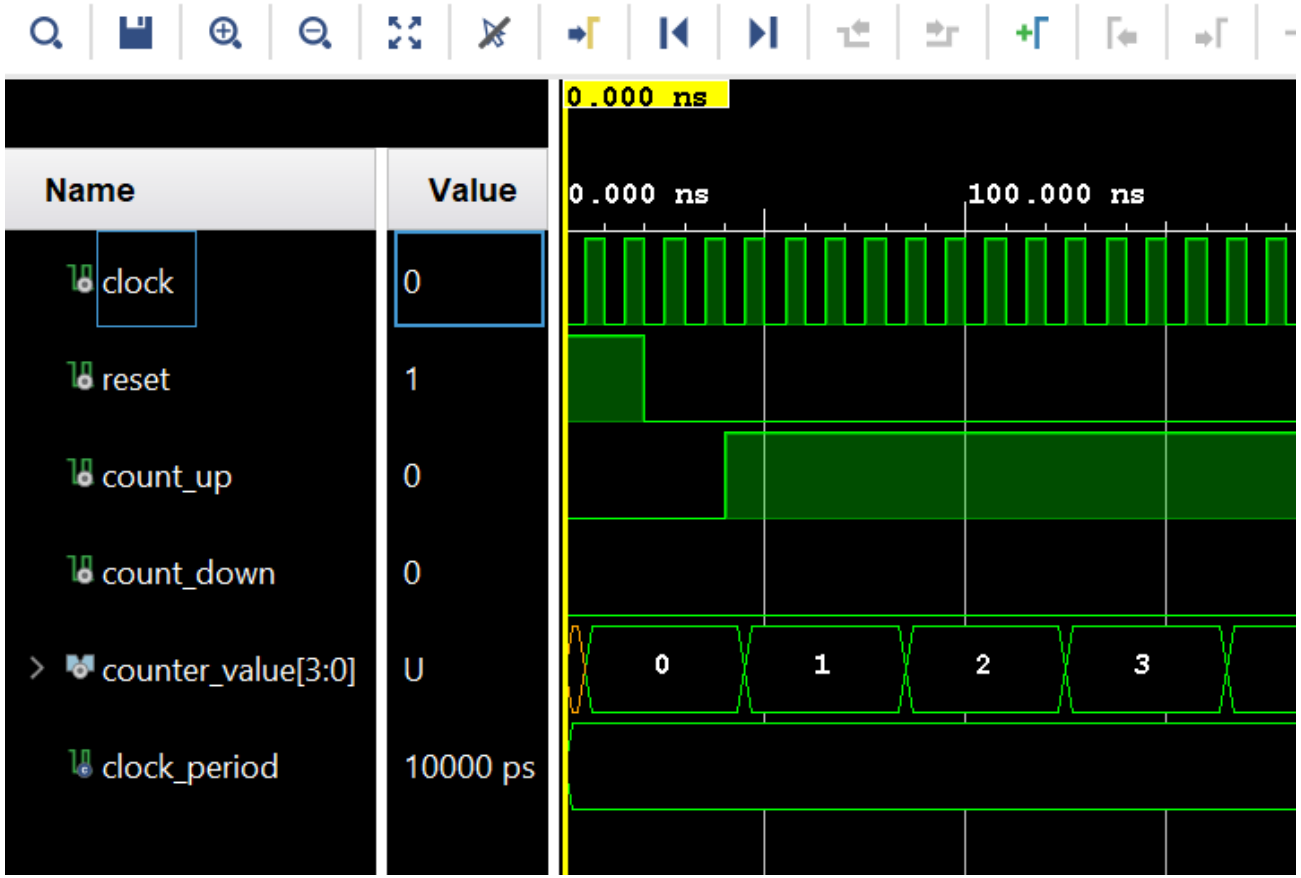
```
clk: process
    begin
        wait for clock_period/2;
        clock <= not clock;
    end process;
```

```
stimuli: process
    begin
        reset <= '1';
        wait for 2*clock_period;
        reset <= '0';
        wait for 2*clock_period;
        count_up <= '1';
        wait for 15*clock_period;
        count_up <= '0';
        count_down <= '1';
        wait; -- wait forever ...
    end process;
end Behavioral;
```

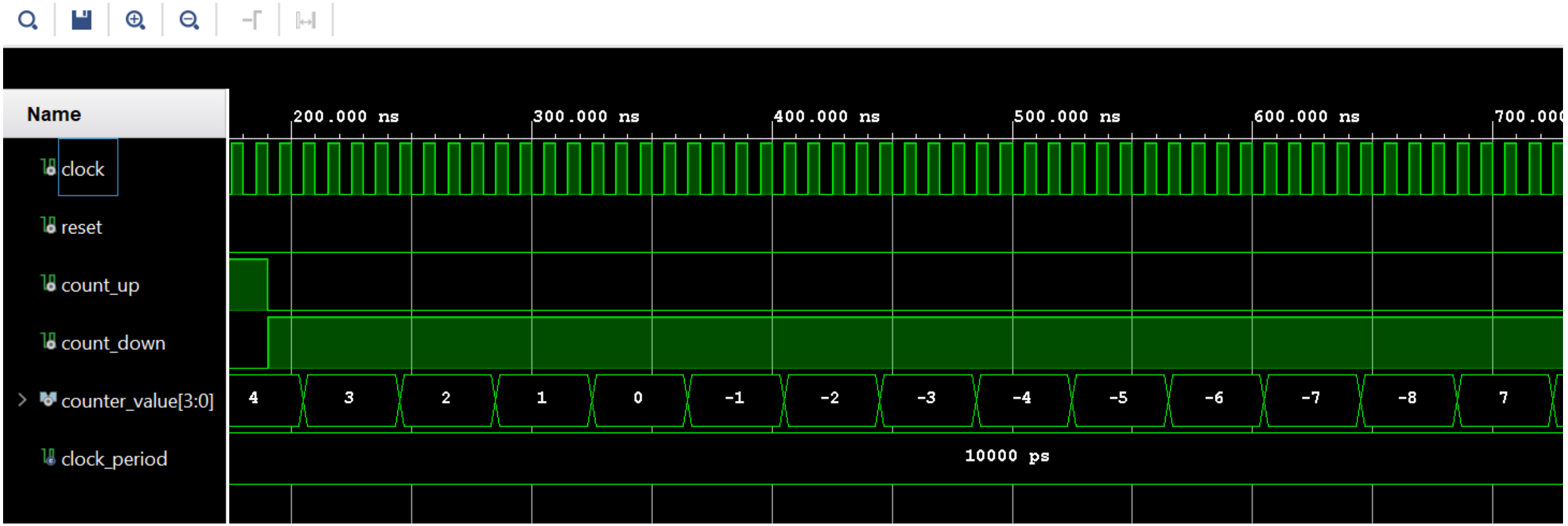
# Simulacija: časovni potek signalov

- Zagon simulacije:
  - Flow → Run Simulation → Run Behavioral Simulation



# Simulacija: časovni potek signalov (2)




Opomba: signal `counter_value` prikazujemo kot predznačeno desetiško vrednost (Radix → Signed Decimal)

# Izziv

V prejšnjem izzivu ste opisali modul za pomično prižiganje LEDic. Isti izziv rešite z uporabo komponent:

- `prescaler`
  - gradnik, ki "upočasni" uro
  - naj ima generično širino registra za števec
- `scroller`
  - gradnik, ki glede na signal iz modula `prescaler` prižiga/ugaša LEDice in ustvari efekt pomikanja (glej opis prejšnjega izziva)
  - naj ima generično širino registra za stanja LEDic; posledično to pomeni število LEDic, ki jih krmili.
- `top`
  - glavni modul, ki v katerem povežete komponenti `prescaler` in `scroller` in opišete zunanji vmesnik.