FPGA tutorial

Lecture 3

11.11.2020

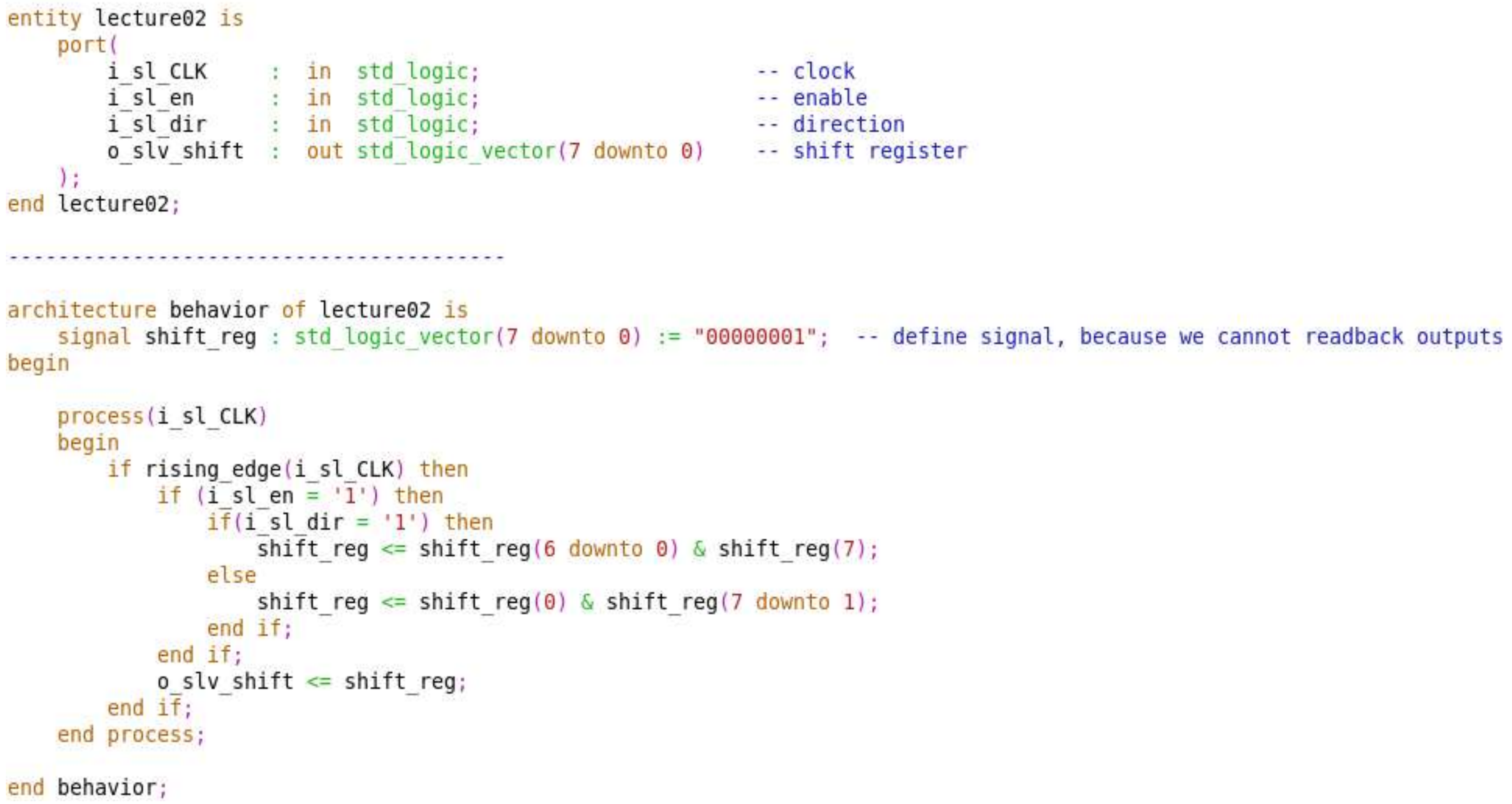
Jochen Steinmann

How does the exam look like?

- Writing some VHDL code / just code inside the module
  - e.g. a counter etc.
- „Manual simulation“ of VHDL code
  - e.g. what does a module do?
- For sure not: Find errors in VHDL code.

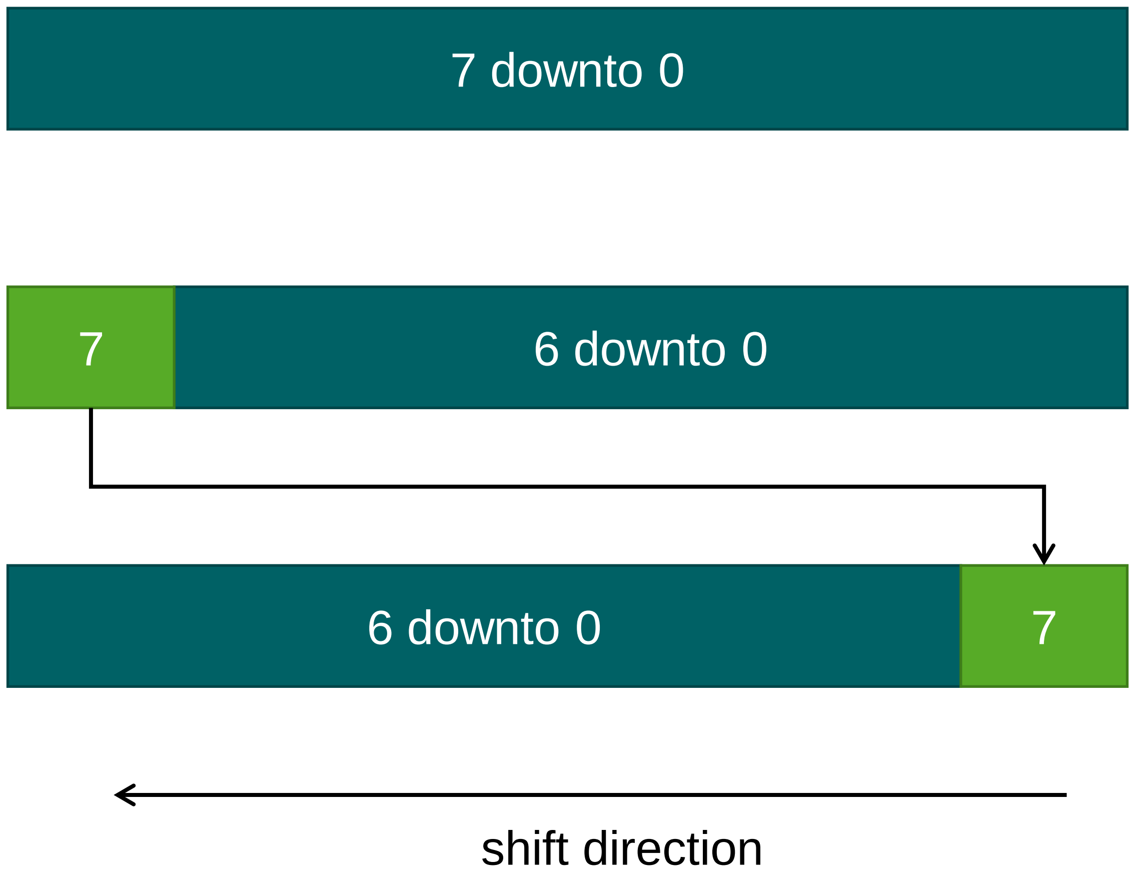
Solution

Solution 2a



```vhdl
entity lecture02 is
    port(
        i_sl_CLK     :  in  std_logic;                       -- clock
        i_sl_en      :  in  std_logic;                       -- enable
        i_sl_dir     :  in  std_logic;                       -- direction
        o_slv_shift  :  out std_logic_vector(7 downto 0)     -- shift register
    );
end lecture02;

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

architecture behavior of lecture02 is
    signal shift_reg : std_logic_vector(7 downto 0) := "00000001";  -- define signal, because we cannot readback outputs
begin

    process(i_sl_CLK)
    begin
        if rising_edge(i_sl_CLK) then
            if (i_sl_en = '1') then
                if(i_sl_dir = '1') then
                    shift_reg <= shift_reg(6 downto 0) & shift_reg(7);
                else
                    shift_reg <= shift_reg(0) & shift_reg(7 downto 1);
                end if;
            end if;
            o_slv_shift <= shift_reg;
        end if;
    end process;

end behavior;
```

Explanation of shift operation

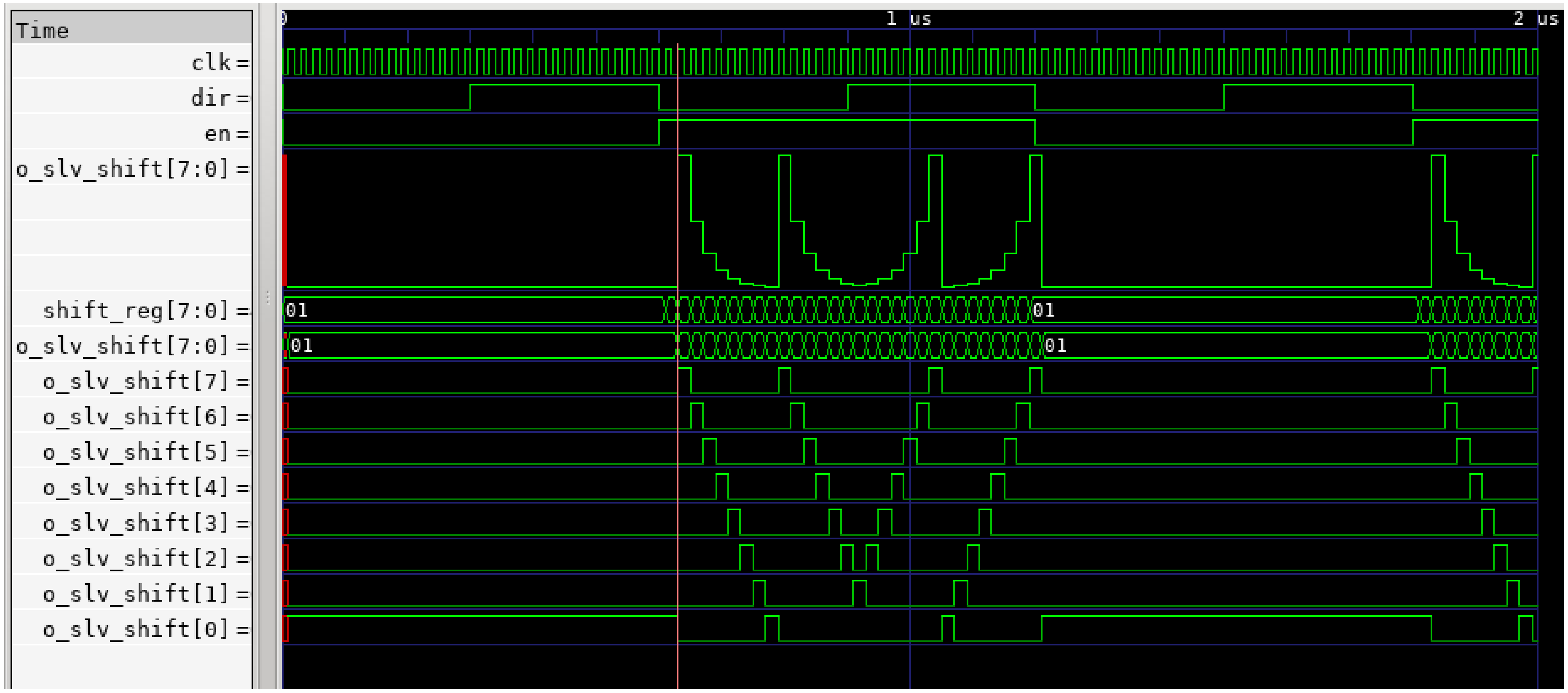


split bus in bit 7 and rest

move bit 7 to right position

Solution 2a

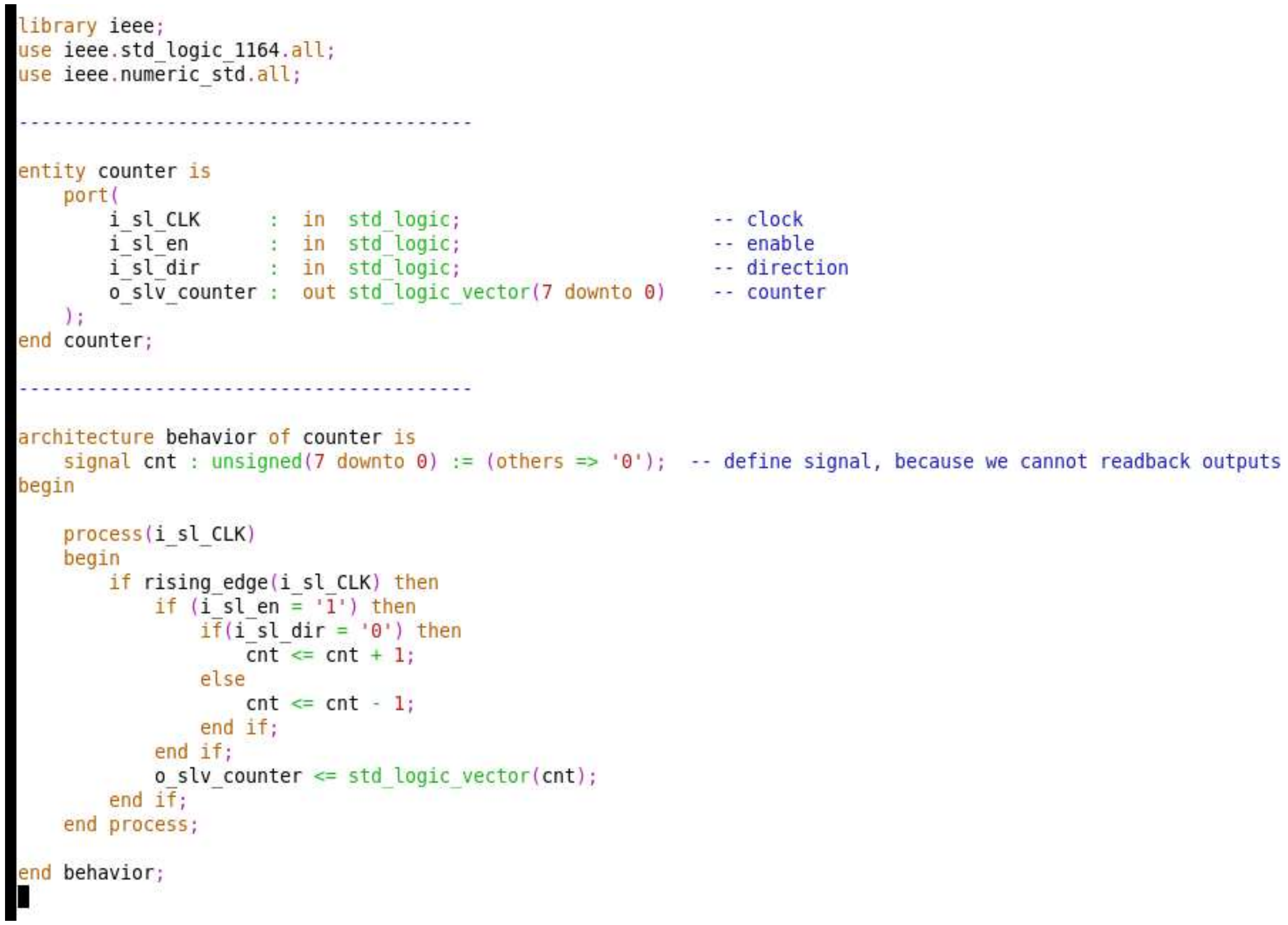
Simulation output



III. Physikalisches Institut B
RWTH AACHEN UNIVERSITY

Solution 2b

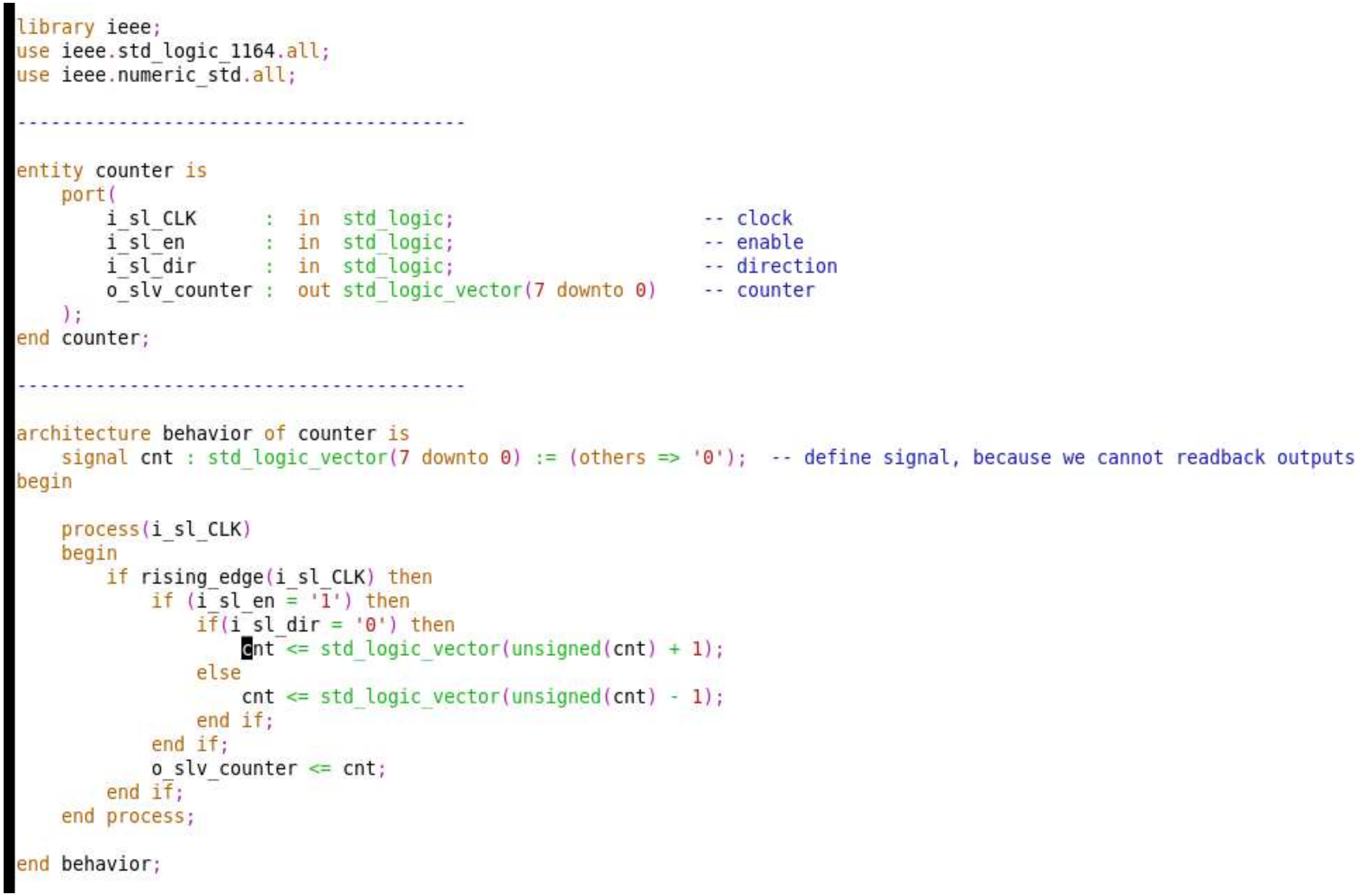
Counter



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

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

entity counter is
    port(
        i_sl_CLK      :  in  std_logic;                     -- clock
        i_sl_en       :  in  std_logic;                     -- enable
        i_sl_dir      :  in  std_logic;                     -- direction
        o_slv_counter :  out std_logic_vector(7 downto 0)   -- counter
    );
end counter;

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

architecture behavior of counter is
    signal cnt : unsigned(7 downto 0) := (others => '0');  -- define signal, because we cannot readback outputs
begin

    process(i_sl_CLK)
    begin
        if rising_edge(i_sl_CLK) then
            if (i_sl_en = '1') then
                if(i_sl_dir = '0') then
                    cnt <= cnt + 1;
                else
                    cnt <= cnt - 1;
                end if;
            end if;
            o_slv_counter <= std_logic_vector(cnt);
        end if;
    end process;

end behavior;
```

Solution 2b

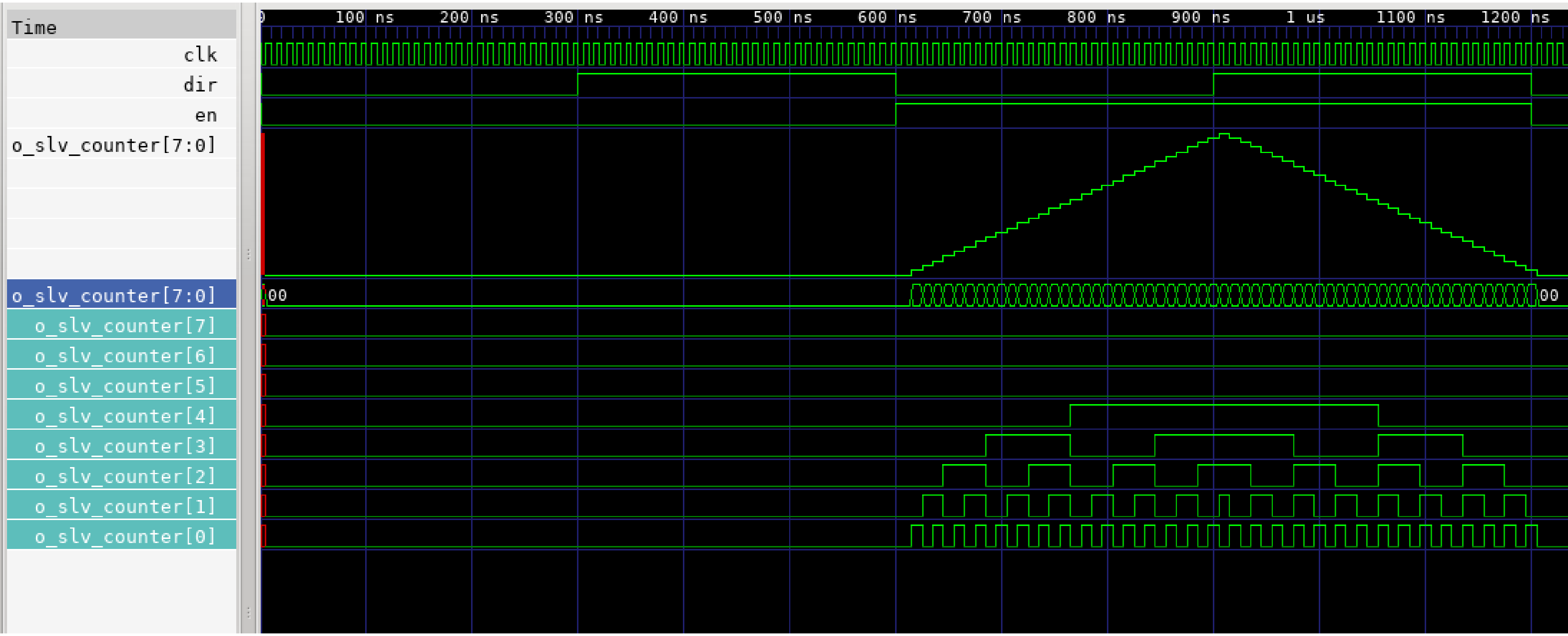
Alternative solution – same result



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

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

entity counter is
    port(
        i_sl_CLK      :  in  std_logic;                       -- clock
        i_sl_en       :  in  std_logic;                       -- enable
        i_sl_dir      :  in  std_logic;                       -- direction
        o_slv_counter :  out std_logic_vector(7 downto 0)     -- counter
    );
end counter;

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

architecture behavior of counter is
    signal cnt : std_logic_vector(7 downto 0) := (others => '0');  -- define signal, because we cannot readback outputs
begin

    process(i_sl_CLK)
    begin
        if rising_edge(i_sl_CLK) then
            if (i_sl_en = '1') then
                if(i_sl_dir = '0') then
                    cnt <= std_logic_vector(unsigned(cnt) + 1);
                else
                    cnt <= std_logic_vector(unsigned(cnt) - 1);
                end if;
            end if;
            o_slv_counter <= cnt;
        end if;
    end process;

end behavior;
```

Solution 2b

Simulation output

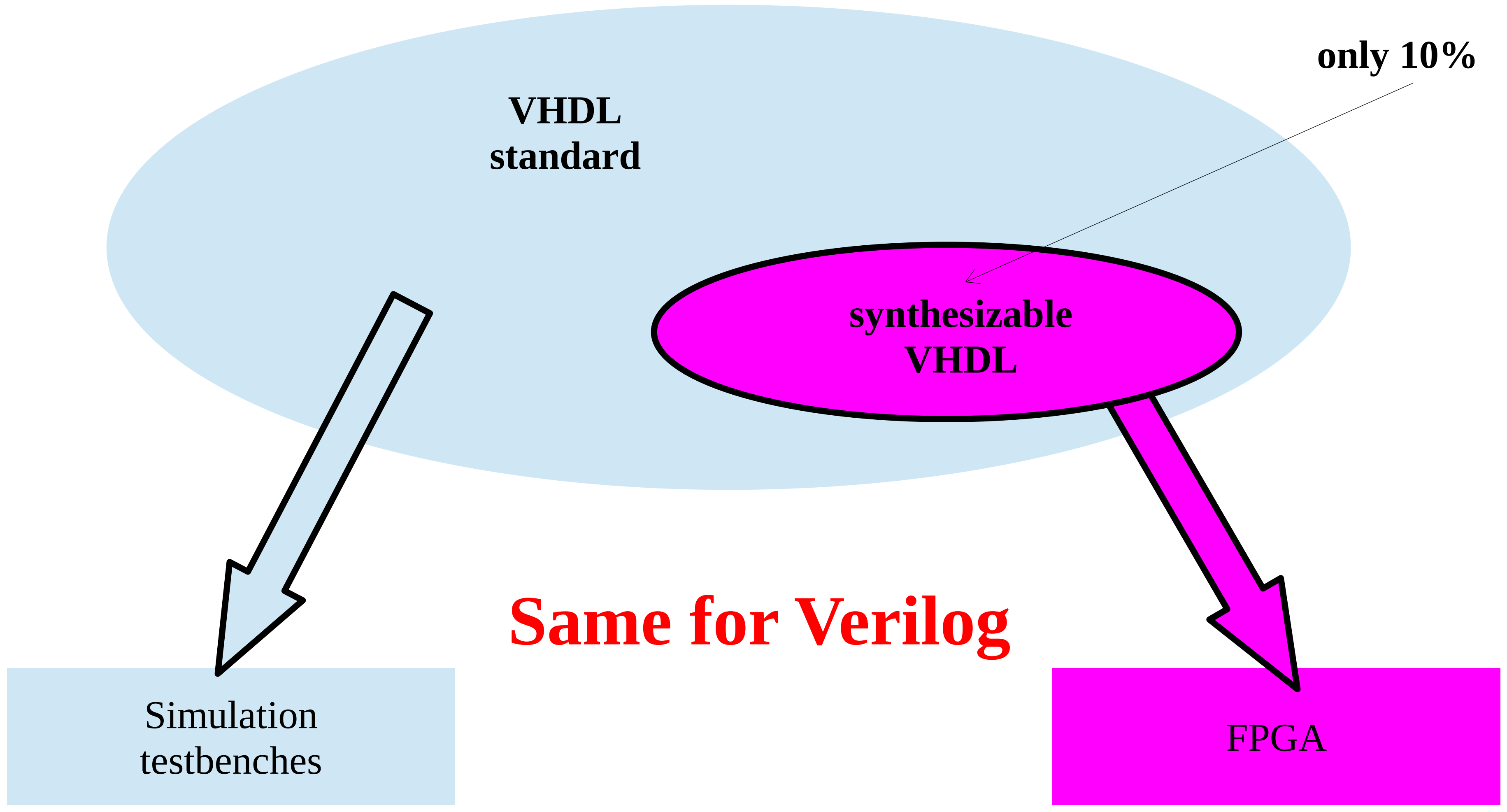


NEW

Basics

- Inside the FPGA everything is executed in parallel!
- It does not matter in which line the definition is written.
  - Order just matters inside one Block

Scope of VHDL



e.g. wait for 5 ns;

Reusing VHDL code

- We have written an 8 bit counter
  - What, if we want a different bit size?
    - rewrite the whole code?
    - Create counter_8bit, counter_9bit, counter_32bit, counter_3bit?
  - Use **generic**

integer is just a number

```
entity  counter is
            generic (
                        data_width : integer := 8
            );
            port (
                        i_sl_clk      : in   std_logic;
                        o_slv_counter : out  std_logic_vector(data_width-1 downto 0)
            );
end counter;
```

  - Generics have to appear in the component and generic map

Changing generic

```
COMPONENT counter is
    generic(
        DATA_WIDTH    :  integer := 8
    );
    port(
        i_sl_CLK      :  in  std_logic;                     -- clock
        i_sl_en       :  in  std_logic;                     -- enable
        i_sl_dir      :  in  std_logic;                     -- direction
        i_sl_rst      :  in  std_logic;                                -- reset
        o_slv_counter :  out std_logic_vector(DATA_WIDTH-1 downto 0)    -- counter
    );
end COMPONENT;
```

```
-- Instance of unit under test.
uut: counter
PORT MAP (
    i_sl_CLK    => CLK,
    i_sl_en     => en,
    i_sl_dir    => dir,
    i_sl_rst    => rst,
    o_slv_counter => o_slv_counter
);

uut2: counter
GENERIC MAP(
    DATA_WIDTH => 9
)
PORT MAP (
    i_sl_CLK    => CLK,
    i_sl_en     => en,
    i_sl_dir    => dir,
    i_sl_rst    => rst,
    o_slv_counter => o_slv_counter2
);
```

Comparing two values

- If we want to compare two values, we have to use ‚signed‘ or ‚unsigned‘

```
if unsigned(slv_pwm_cnt) >= unsigned(i_slv_val) then
```

Works just in if clause, means, we cannot assign the values to a signal directly

Constants

Sometimes, things are really constant

- For example the data_width in a testbench

```
constant DATA_WIDTH : integer := 4;
```

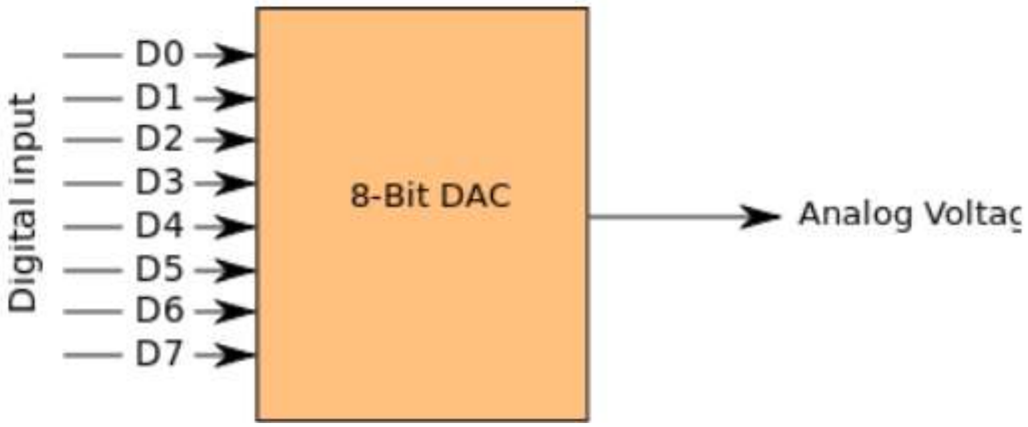
like #define in C / C++

Resetting a module

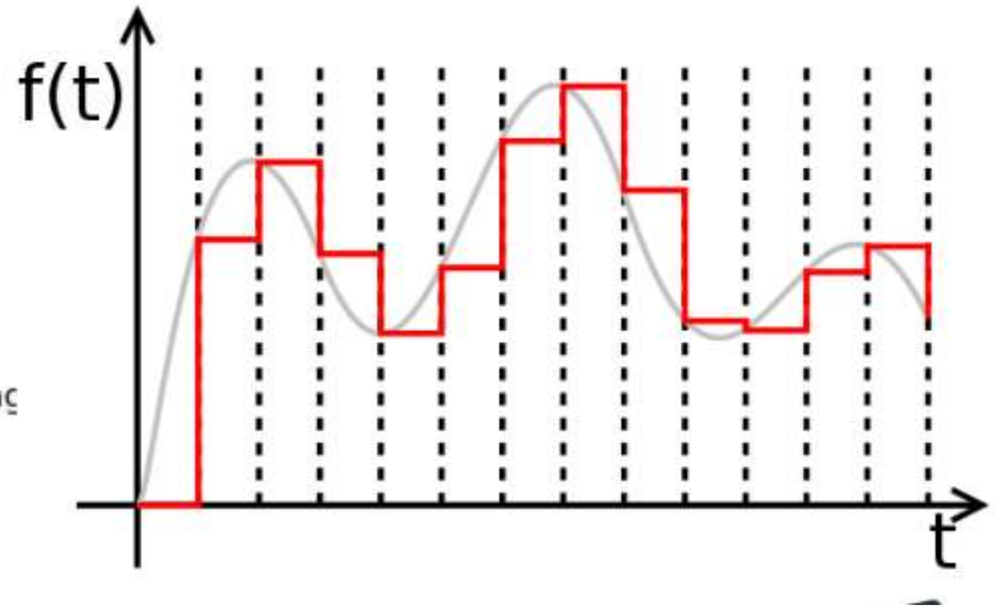
- There are a two ways how to reset a module:
  - **Asynchronous reset**
    - Reset signal can be raised without any relation to the clock
  - **Synchronous reset**
    - Reset signal is evaluated on clock event
  - A FPGA is made for a synchronous reset
    - Relates on the internal structure of the logic blocks

Creating analog outputs

- Needed for
  - thresholds
  - stimulus waveforms
  - testpulses

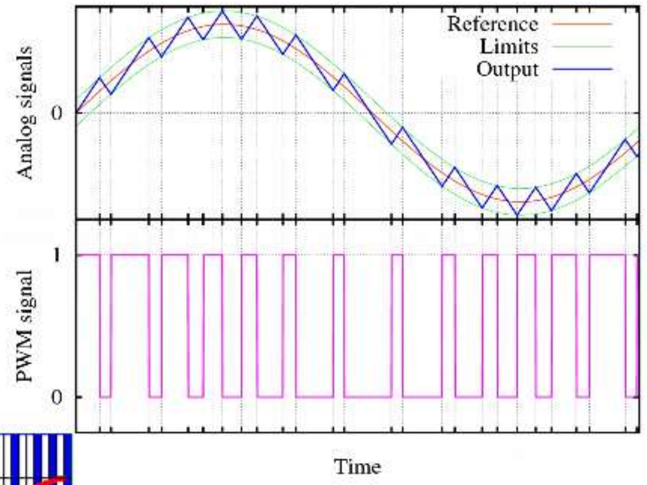


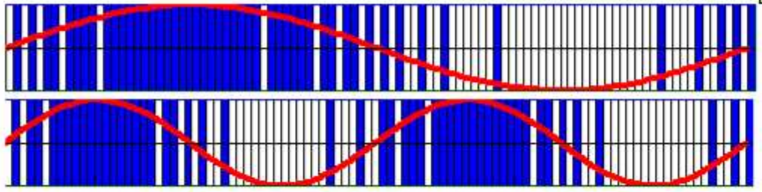
output of DAC



1-Pin DAC

- Pulse-Width-Modulation
  PWM
- Pulse Density Modulation
  PDM
  - Delta-Sigma-DAC

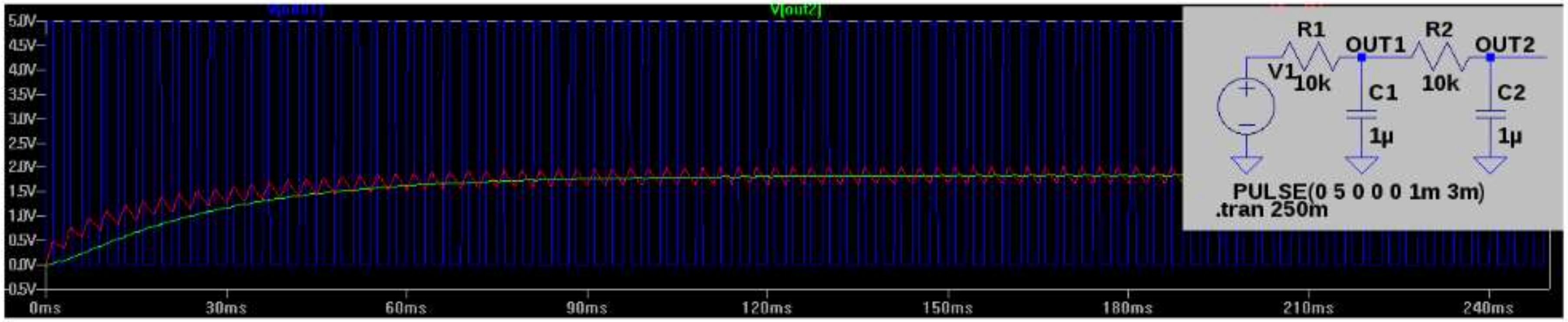


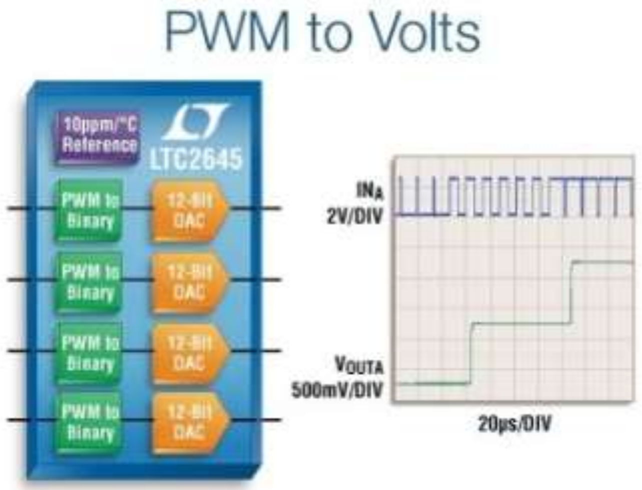


Example for PDM Sigma-Delta-DAC
same Pulsewidth but different Pulses per time!

Real analogue output

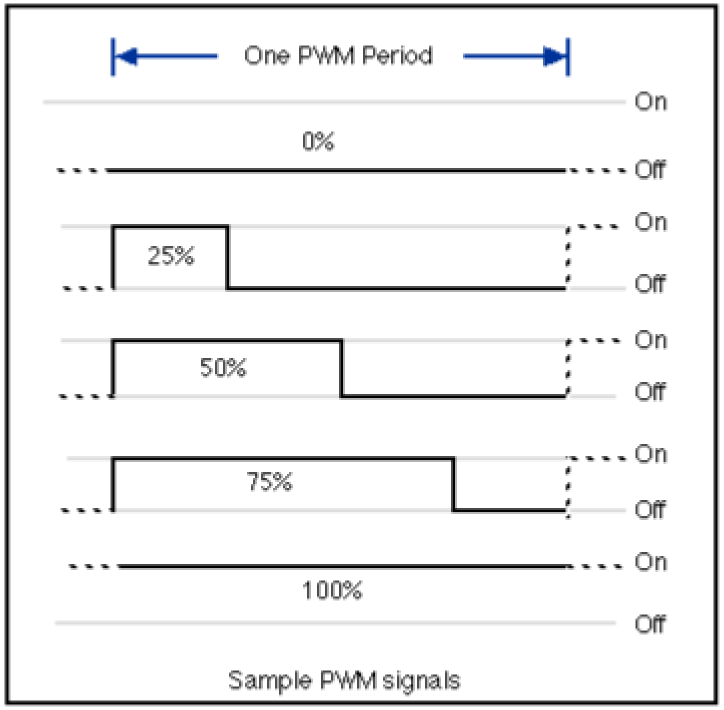
- PWM needs some smoothing
  - for normal low-pass filter is sufficient (here 2nd order)

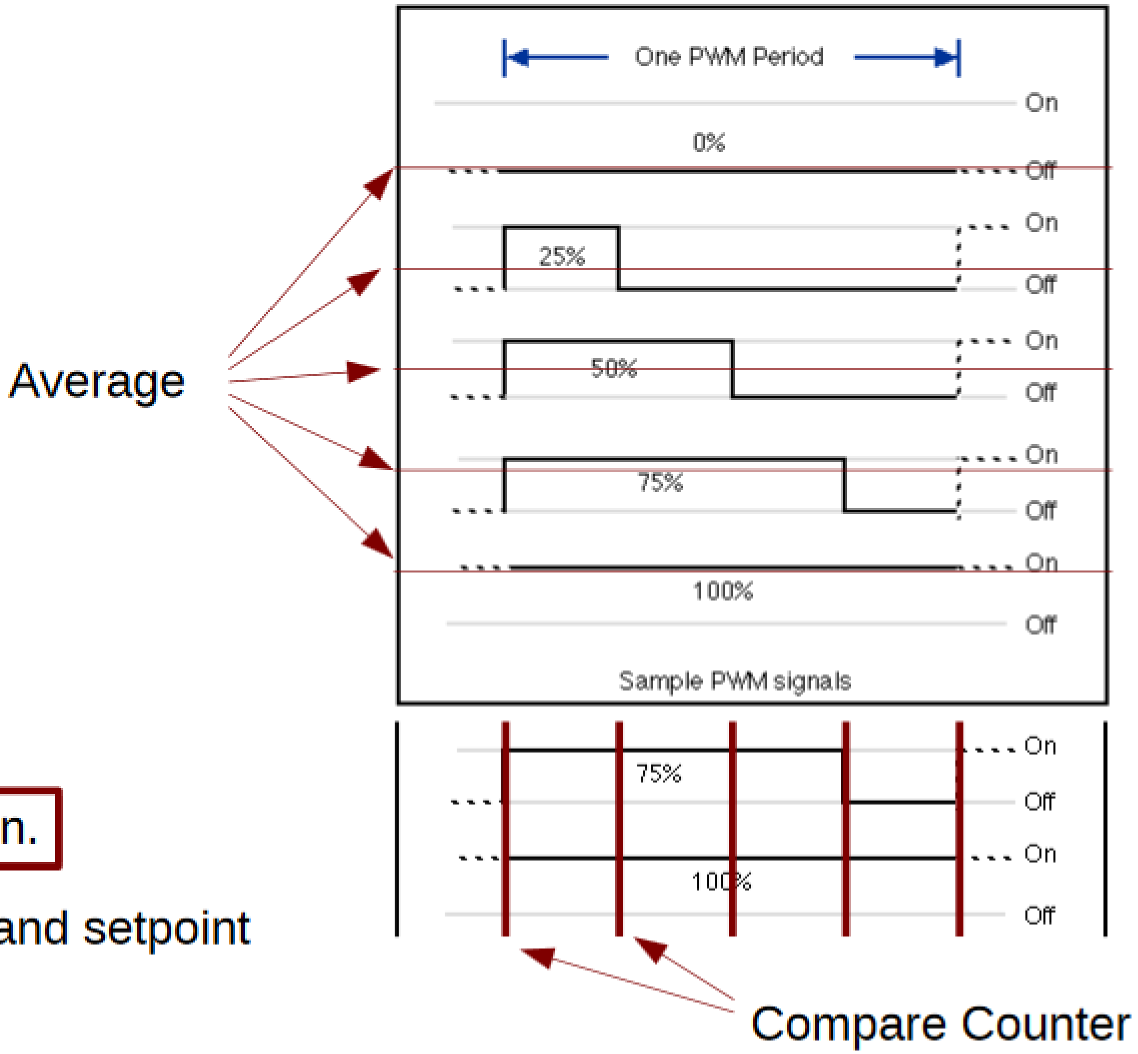




  - not very linear
    - but this can be solved by calibration

Creation of different pulse width



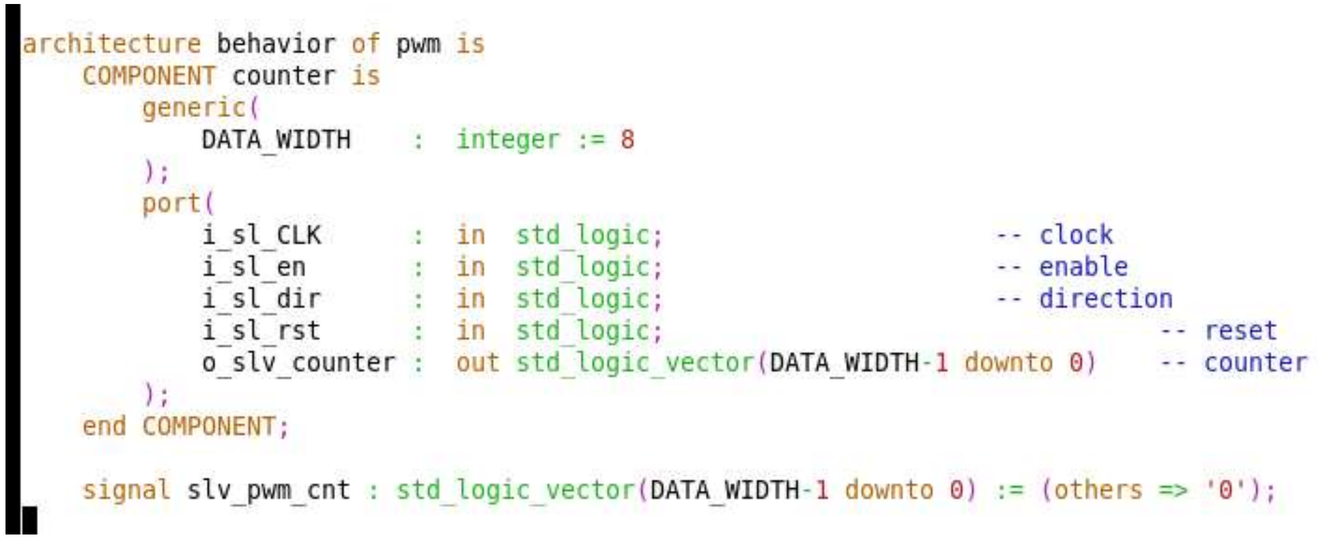


PWM Period: Time until counter overrun.

Output: Comparison between counter and setpoint

How to use a module in a module?

- COMPONENT direct after acrhitecture



```
architecture behavior of pwm is
    COMPONENT counter is
        generic(
            DATA_WIDTH     :  integer := 8
        );
        port(
            i_sl_CLK       :  in  std_logic;                          -- clock
            i_sl_en        :  in  std_logic;                          -- enable
            i_sl_dir       :  in  std_logic;                          -- direction
            i_sl_rst       :  in  std_logic;                                     -- reset
            o_slv_counter  :  out std_logic_vector(DATA_WIDTH-1 downto 0)       -- counter
        );
    end COMPONENT;

    signal slv_pwm_cnt : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
```

- PORT MAP as done in testbench

ToDo for today…

- **3a:** Implement generics in your counter
  - one 8 bit and one 9 bit counter

starting point, solution 2b

- **3b:** Implement a reset in your counter

- **3c:** Implement a first version of the PWM by using counter
  - Use your counter and skeleton3c.zip from MOODLE