VHDL Language Review (Vivado)

VHDL Testbenches

Digilent Nexys-4 kit – basic I/O

LECTURE 2

IOULIIA SKLIAROVA

VHDL

Very high speed integrated circuits **H**ardware **D**escription **L**anguage (IEEE std 1076)

- Modeling, simulation and synthesis of digital systems
- Allows to describe the behavior and structure of digital hardware

Vivado synthesis supports a synthesizable subset of:

- VHDL: IEEE Standard for VHDL Language (IEEE Std 1076-2002)
- VHDL 2008
- …

Supported VHDL Data Types

- Predefined enumerated types
  - bit (std)
  - boolean (std)
  - std_logic (std_logic_1164)
    - ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-')
- User-defined enumerated types
  - **type** STATES **is** (START, IDLE, RUN) ;
- Bit vector types
  - std_logic_vector (std_logic_1164)
- Overloaded types
  - unsigned (numeric_std)
  - signed (numeric_std)
- Integer types (32 bits by default)
- Multi-dimensional array types (no restriction but limit to 3)
- Record types

VHDL Code Structure

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

Inclusion of **libraries**

```
entity ??? is
  port(sel    : in  std_logic;
       input0 : in  std_logic;
       input1 : in  std_logic;
       mOut   : out std_logic);
end ???;
```

Entity - definition of module interface

Is VHDL case-sensitive?

Identifiers are subject to certain restrictions

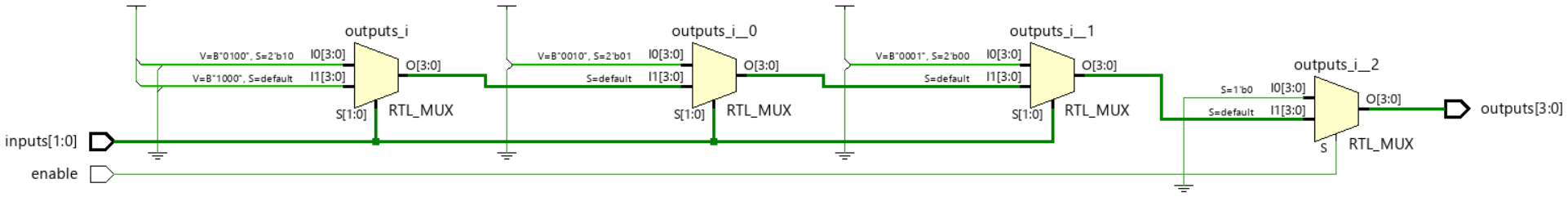
```
architecture Equations of ??? is

  signal s_and0Out, s_and1Out : std_logic;

begin
  s_and0Out <= not sel and input0;
  s_and1Out <=     sel and input1;
  mOut      <= s_and0Out or s_and1Out;
end Equations;
```

Signal and constant declarations

Architecture - definition of module implementation

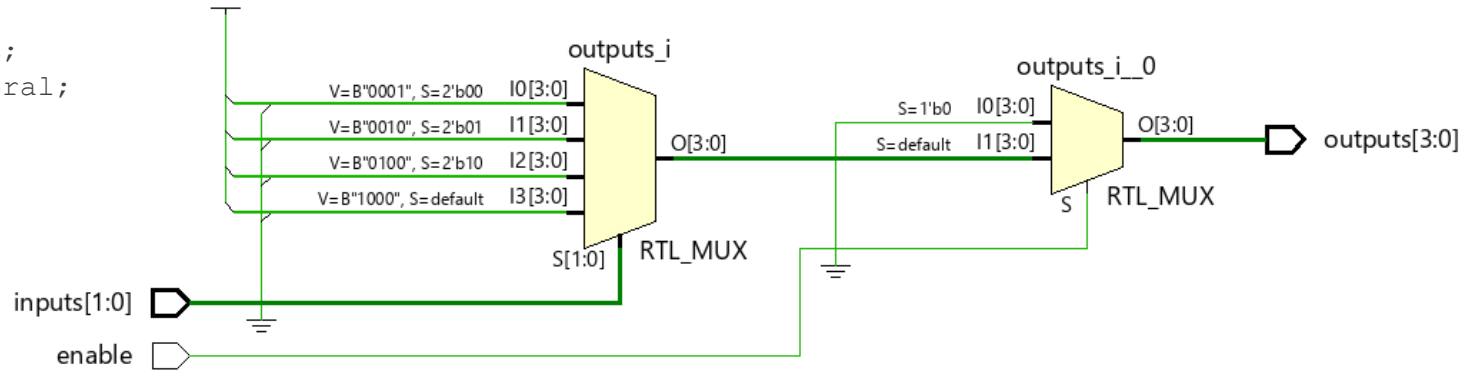
VHDL Module? Hardware Inferred?

```
entity ??? is
  port(enable  : in  std_logic;
       inputs  : in  std_logic_vector (1 downto 0);
       outputs : out std_logic_vector (3 downto 0));
end ???;

architecture Behavioral of ??? is
begin
    process(enable, inputs)
    begin
        if (enable = '0') then
            outputs <= "0000";
        else
            if (inputs = "00")     then     outputs <= "0001";
            elsif (inputs = "01") then      outputs <= "0010";
            elsif (inputs = "10") then      outputs <= "0100";
            else                            outputs <= "1000";
            end if;
        end if;
    end process;
end Behavioral;
```



VHDL Module? Hardware Inferred?

```
entity ??? is
  port(enable  : in  std_logic;
       inputs  : in  std_logic_vector (1 downto 0);
       outputs : out std_logic_vector (3 downto 0));
end ???;

architecture Behavioral of ??? is
begin
    process(enable, inputs)
begin
    if (enable = '0') then outputs <= (others => '0');
    else
        case(inputs) is
            when "00"   => outputs <= x"1";
            when "01"   => outputs <= x"2";
            when "10"   => outputs <= x"4";
            when others => outputs <= x"8";
        end case;
    end if;
end process;
end Behavioral;
```



VHDL Module? Hardware Inferred?

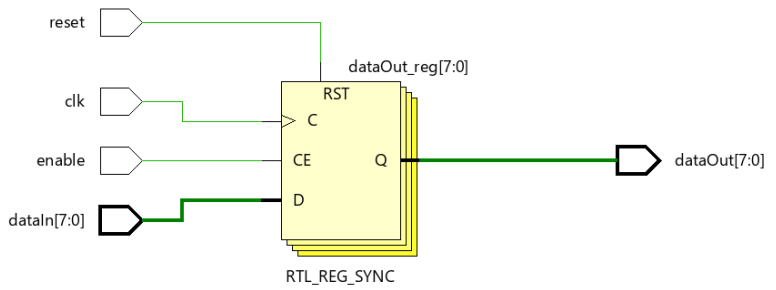
```
entity ??? is
  port(reset   : in  std_logic;
       clk     : in  std_logic;
       enable  : in  std_logic;
       dataIn  : in  std_logic;
       dataOut : out std_logic);
end ???;
```

```
architecture Behavioral1 of ??? is
begin
  process(reset, clk)
  begin
    if (reset = '1') then
      dataOut <= '0';
    elsif (rising_edge(clk)) then
      if (enable = '1') then
        dataOut <= dataIn;
      end if;
    end if;
  end process;
end Behavioral1;
```

```
architecture Behavioral2 of ??? is
begin
  process(clk)
  begin
    if (rising_edge(clk)) then
      if (reset = '1') then
        dataOut <= '0';
      elsif (enable = '1') then
        dataOut <= dataIn;
      end if;
    end if;
  end process;
end Behavioral2;
```

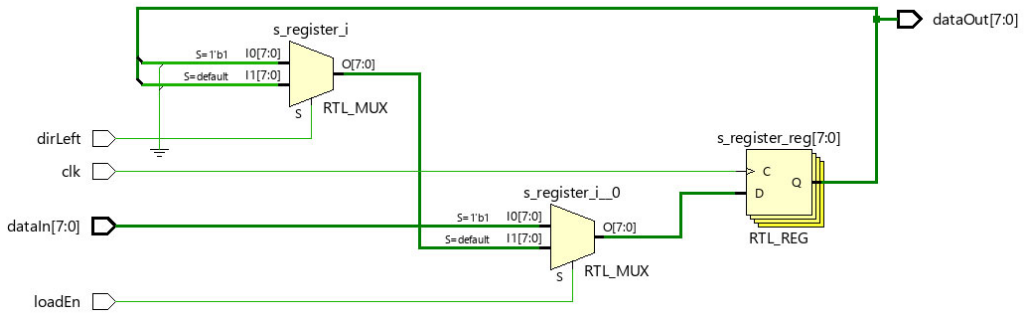
VHDL Module?

```
entity ??? is
  generic(N     : positive := 8);
  port(reset    : in  std_logic;
       clk      : in  std_logic;
       enable   : in  std_logic;
       dataIn   : in  std_logic_vector((N-1) downto 0);
       dataOut  : out std_logic_vector((N-1) downto 0));
end ???;

architecture Behavioral of ??? is
begin
  process(clk)
  begin
    if (rising_edge(clk)) then
      if (reset = '1') then
        dataOut <=  (others => '0');
      elsif (enable = '1') then
        dataOut <= dataIn;
      end if;
    end if;
  end process;
end Behavioral;
```



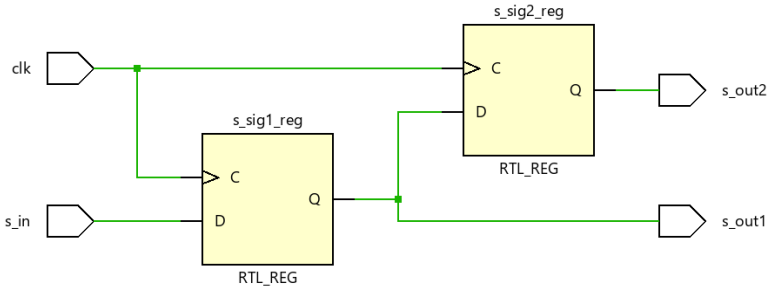
VHDL Module? Hardware Inferred?

```
entity ??? is
  port(clk       : in  std_logic;
       loadEn    : in  std_logic;
       dataIn    : in  std_logic_vector(7 downto 0);
       dirLeft   : in  std_logic;
       dataOut   : out std_logic_vector(7 downto 0));
end ???;

architecture Behavioral of ??? is
  signal s_register : std_logic_vector(7 downto 0);
begin
  process(clk)
  begin
    if (rising_edge(clk)) then
      if (loadEn = '1') then
        s_register <= dataIn;
      elsif (dirLeft = '1') then
        s_register <= s_register(6 downto 0) & '0';
      else
        s_register <= '0' & s_register(7 downto 1);
      end if;
    end if;
  end process;

  dataOut <= s_register;

end Behavioral;
```



Hardware Inferred?

```
entity ??? is
    Port ( clk    : in STD_LOGIC;
           s_in   : in STD_LOGIC;
           s_out1 : out STD_LOGIC;
           s_out2 : out STD_LOGIC);
end ???;

architecture Behavioral of ??? is
    signal s_sig1, s_sig2 : std_logic;
begin

process(clk)
begin
    if (rising_edge(clk)) then
        s_sig1 <= s_in;
        s_sig2 <= s_sig1;
    end if;
end process;

    s_out1 <= s_sig1;
    s_out2 <= s_sig2;

end Behavioral;
```



```
process(clk)
begin
    if (rising_edge(clk)) then
        s_sig2 <= s_sig1;
        s_sig1 <= s_in;
    end if;
end process;
```

VHDL Module? Hardware Inferred?

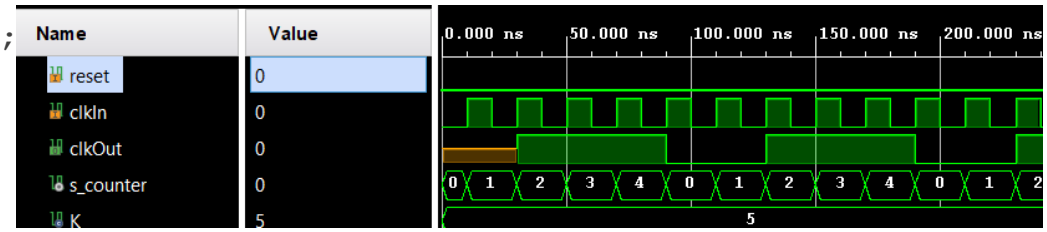
```
entity ??? is
  generic(K    : positive := 4);
  port(reset   : in  std_logic;
       clkIn   : in  std_logic;
       clkOut  : out std_logic);
end ???;

architecture Behavioral of ??? is
  signal s_counter : natural;
begin
  process(clkIn)
  begin
    if rising_edge(clkIn) then
      if ((reset = '1') or (s_counter = K - 1)) then
        clkOut    <= '0';
        s_counter <= 0;
      else
        if (s_counter = K/2 - 1) then
          clkOut    <= '1';
        end if;
        s_counter <= s_counter + 1;
      end if;
    end if;
  end process;
end Behavioral;
```

Clock divider
Modulo K free running counter
`clkOut <=` `'1'` at the "middle" of the count
`clkOut <=` `'0'` at the end of the count

Assuming
f_{clkIn} = 50 MHz

K=10
f_{clkOut} ?
Duty cycle?

K=5
f_{clkOut} ?
Duty cycle?



Concurrent Statements

- Concurrent statements define logic that is inherently parallel.
- Concurrent statements are evaluated independently of the order in which they appear.
- Signals pass values between concurrent statements, much as wires connect components on a schematic.
- Concurrent statements include:
  - Signal assignments (simple, selected and conditional)
  - Process statements
  - Component instantiations
  - Generate statements
  - Procedure and function calls

Signal Assignments

- Simple signal assignment:

```
a <= b and c;
```

- Conditional signal assignment:

```
out <= in1 when s = '0' else in2;
```

- Selected signal assignment:

```
with inputs select outputs <=
   x"1" when "00",
   x"2" when "01",
   x"4" when "10",
   x"8" when others;
```

Process Statements

A process includes **sequential statements**, so called because they are executed in sequence.

The process statement includes a **sensitivity list** - a list of signals to which the process is sensitive. When any of these signals changes value, the process resumes and executes the sequential statements.

After it has executed the last statement, the process suspends again.

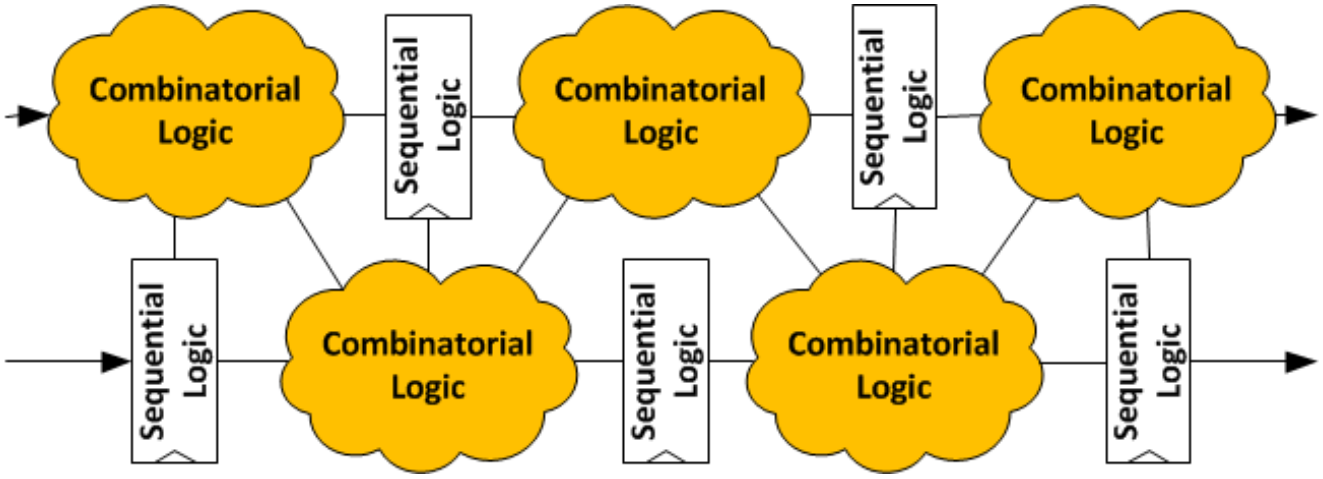
Signals' values in a process are updates when the process suspends.

Sequential statements:

- if statements
- conditional assignments (VHDL-2008)
- case statements
- selected assignments (VHDL-2008)
- …

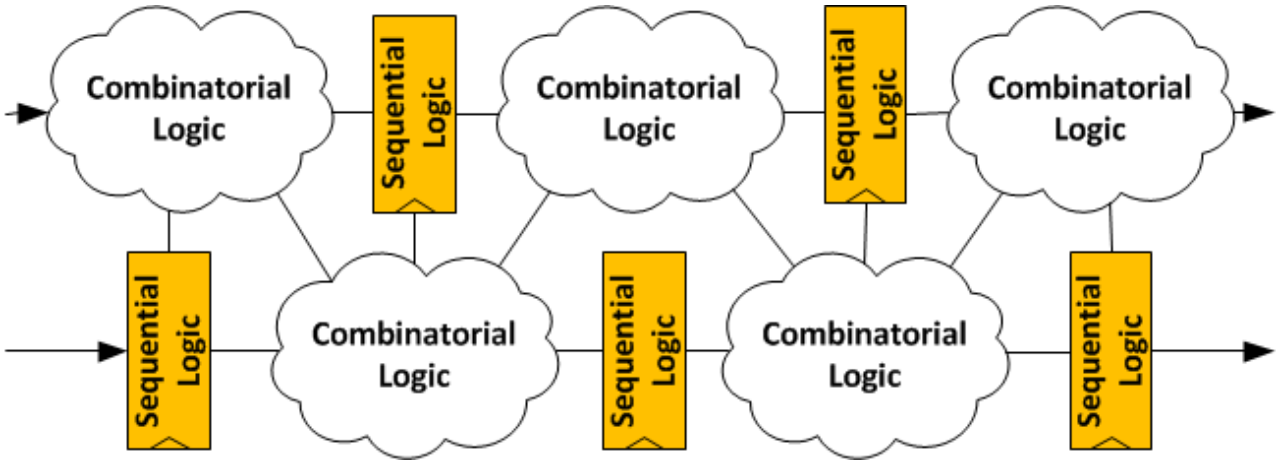
Typical VHDL Process Template for a Combinational Component

```
process(<all inputs>)
begin
   <assignments to signals/ports – outputs must be specified
for all combinations of input signals – even if don't care
(to avoid latches)>
end process;
```



Typical VHDL Process Template for a Sequential Component

```
process(<clock and asynchronous sets/resets>)
begin
   <test of asynchronous signals>
      <asynchronous assignments>
   <test of active clock edge>
      <test of synchronous signals>
         <synchronous assignments>
end process;
```



Components

You know how to write entity declarations and architecture bodies that describe the structure of a system.

Within an architecture body, we can write entity instantiation statements that describe instances of an entity and connect signals to the ports of the instances.

```
bit0 : entity work.d_ff(basic)
       port map (d0, int_clk, q0);
```

This simple approach to building a hierarchical design works well if we know in advance all the details of the entities we want to use.

However, that is not always the case, especially in a large design project.

Components are an alternative way of describing the hierarchical structure of a design that affords significantly more flexibility at the cost of a little more effort in managing the design.

Components

```
16  Entity tutorial_tb Is
17  end tutorial_tb;
18
19  Architecture behavior of tutorial_tb Is
20      Component tutorial
21      port (
22           sw  : in STD_LOGIC_VECTOR(7 downto 0);
23           led : out STD_LOGIC_VECTOR(7 downto 0)
24           );
25      End Component;
26
27      Signal switch : STD_LOGIC_VECTOR(7 downto 0) := X"00";
28      Signal led_out : STD_LOGIC_VECTOR(7 downto 0) := X"00";
29      Signal led_exp_out : STD_LOGIC_VECTOR(7 downto 0) := X"00";
30
31      Signal count_int_2 : STD_LOGIC_VECTOR(7 downto 0) := X"00";
32
33      procedure expected_led (...
49
50  begin
51      uut:  tutorial PORT MAP (
52              sw  => switch,
53              led => led_out
54           );
```

Component declaration: specifies the external interface to the component in terms of generic constants and ports

Component instantiation: specifies a usage of the module in a design

Generate Statements

Generate statement is a concurrent statement containing further concurrent statements that are to be replicated during elaboration of a design.

```
entity ??? is
    port ( code_in  : in STD_LOGIC_VECTOR (3 downto 0);
           en       : in STD_LOGIC;
           code_out : out STD_LOGIC_VECTOR (15 downto 0));
end ???;

architecture Behavioral of ??? is
begin
ger_out: for i in code_out'range generate
  code_out(i) <= en when (i = to_integer(unsigned(code_in)))
                 else '0';
end generate;

end Behavioral;
```

Typical Specification Errors

When the value of a signal/port is not specified for one or more input combinations the synthesis tool infers a memory element for that signal/port (why?):

- Flip-flop
- Latch

This situation could be either absolutely Ok (for sequential circuits) or undesirable (for combinational circuits).

Typical Specification Errors

```
process(enable, dataIn)
begin
  if (enable = '1') then
    dataOut <= dataIn;
  end if;
end process;
```

Sequential (latch)

```
process(clk)
begin
  if (clk'event and clk = '1') then
    dataOut <= dataIn;
  end if;
end process;
```

Sequential (flip-flop)

```
process(decodIn)
begin
    if (decodIn(1) = '1') then
        validOut <= '1';
        encodOut <= "1";
    elsif (decodIn(0) = '1') then
        validOut <= '1';
        encodOut <= "0";
    else
        validOut <= '0';
        encodOut <= "-";
    end if;
end process;
```

Combinational (2:1 priority encoder)
If the line **encodOut<="-"** is removed, the signal **encodOut** is not specified for **decodIn="00"**, leading the tools to infer a latch for this signal!

Multiple Assignments to a Signal

If multiple assignments are made to a signal in a process, according to VHDL semantics, the last one prevails.

This facility permits to make code more compact.

```
process(decodIn)
begin
    validOut <= '1';
    if (decodIn(1) = '1') then
       encodOut <= "1";
    elsif (decodIn(0) = '1') then
       encodOut <= "0";
    else
        validOut <= '0';
        encodOut <= "-";
    end if;
end process;
```

However, only one concurrent statement may control a signal (exception: tri-state multi-driver signals).

Simulation in VHDL - Testbenches

COMBINATIONAL MODULES

Entity without ports

Architecture:

- Declaration of the UUT (**Unit Under Test**) in the declarative part of the architecture
- Declaration of signals to be connected to UUT ports in the declarative part of the architecture
- Instantiation of UUT in the architecture body
- Defining a process generating the simulation vectors over time
  - In more complex systems, more than one process can be used for this purpose

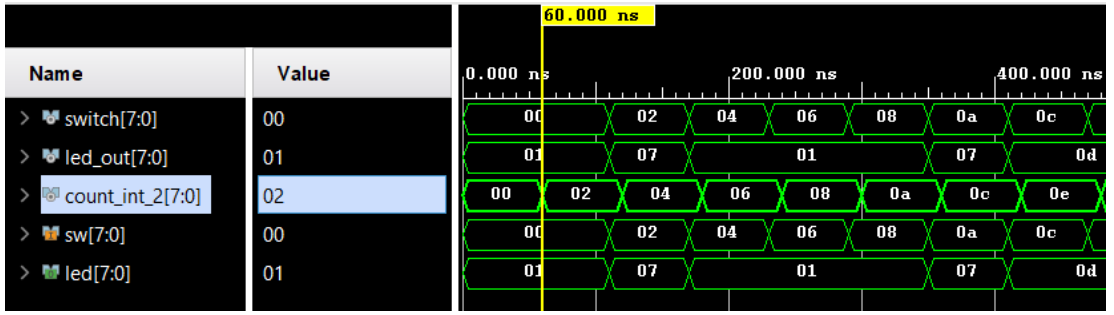
SEQUENTIAL MODULES

Entity without ports

Architecture:

- Declaration of the UUT (**Unit Under Test**) in the declarative part of the architecture
- Declaration of signals to be connected to UUT ports in the declarative part of the architecture
- Instantiation of UUT in the architecture body
- Defining a process for generating the clock signal
- Defining a process to apply the simulation vectors over time
  - In more complex systems, more than one process can be used for this purpose

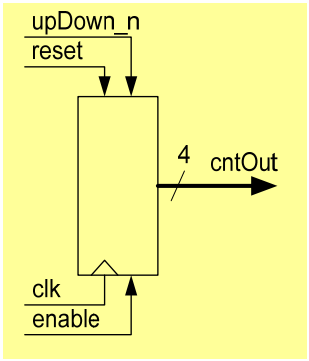
VHDL Testbench Example (CS)




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

entity tutorial_tb is
end tutorial_tb;

architecture behavior of tutorial_tb is
        component tutorial
        port (     sw  : in STD_LOGIC_VECTOR(7 downto 0);
                   led : out STD_LOGIC_VECTOR(7 downto 0));
        end component;

        signal switch : STD_LOGIC_VECTOR(7 downto 0) := X"00";
        signal led_out : STD_LOGIC_VECTOR(7 downto 0) := X"00";
        signal count_int_2 : unsigned(7 downto 0) := X"00";
begin
        uut:  tutorial PORT MAP ( sw  => switch,
                                  led => led_out);

        comb_process: process
        begin
                    wait for 50 ns;
                    switch <= std_logic_vector(count_int_2);
                    wait for 10 ns;
                    count_int_2 <= count_int_2 + x"02";
        end process;
end behavior;
```

VHDL Testbench Example (SS)

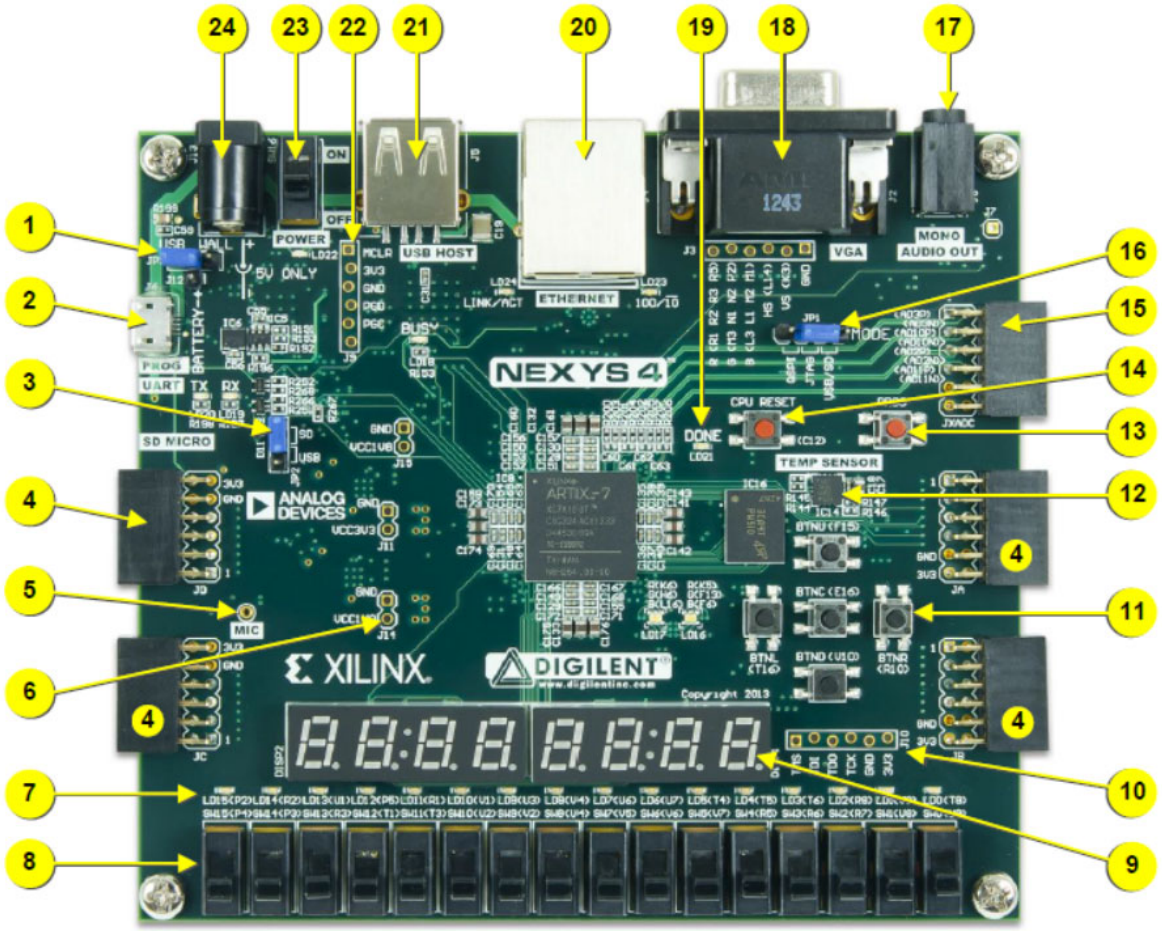
```
entity BinUDCntEnRst8Tb is
end BinUDCntEnRst8Tb;

architecture Stimulus of BinUDCntEnRst8Tb is
  signal s_reset, s_clk       : std_logic;
  signal s_enable, s_upDown_n : std_logic;
  signal s_cntOut : std_logic_vector(3 downto 0);

  <BinUDCntEnRst4 component declaration>

begin
  uut : BinUDCntEnRst4
        port map(reset    => s_reset,
                 clk      => s_clk,
                 enable   => s_enable,
                 upDown_n => s_upDown_n,
                 cntOut   => s_cntOut);

  clock_proc : process
  begin
    s_clk <= '0'; wait for 100 ns;
    s_clk <= '1'; wait for 100 ns;
  end process;
```




```
stim_proc : process
  begin
    s_reset    <= '1';
    s_enable   <= '0';

    s_upDown_n <= '1';
    wait for 325 ns;

    s_reset    <= '0';
    wait for 25 ns;

    s_enable   <= '1';
    wait for 925 ns;
    s_enable   <= '0';
    wait for 375 ns;

    s_upDown_n <= '0';
    s_enable   <= '1';
    wait for 975 ns;

    s_enable   <= '0';
    wait for 125 ns;
  end process;
end Stimulus;
```

Nexys-4 Development Board

- 16 user switches
- 16 user LEDs
- 2 tri-color LEDs
- 6 push buttons
- 100 MHz oscillator
- …



FPGA: xc7a100Tcsg324-1

Nexys-4 Basic I/O

- 2 tri-color LEDs
- 16 slide switches
- 6 push buttons
- 16 individual LEDs

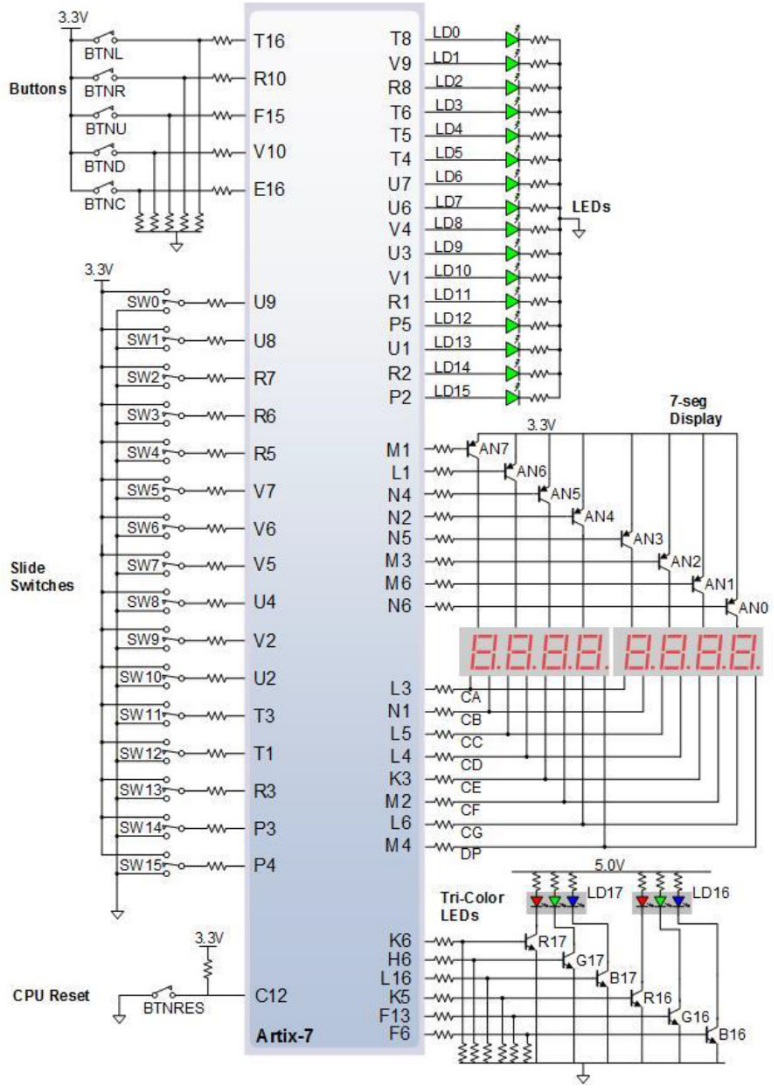
The five pushbuttons generate a low output when they are at rest, and a high output only when they are pressed.

The red pushbutton labeled “CPU RESET” generates a high output when at rest and a low output when pressed.

The CPU RESET button is intended to be used in Vitis designs to reset the processor, but you can also use it as a general purpose pushbutton.

Slide switches generate constant high or low inputs depending on their position.

The sixteen individual LEDs are anode-connected to the FPGA via 330-ohm resistors, so they will turn on when a logic high voltage is applied to their respective I/O pin.



Tri-color LEDs

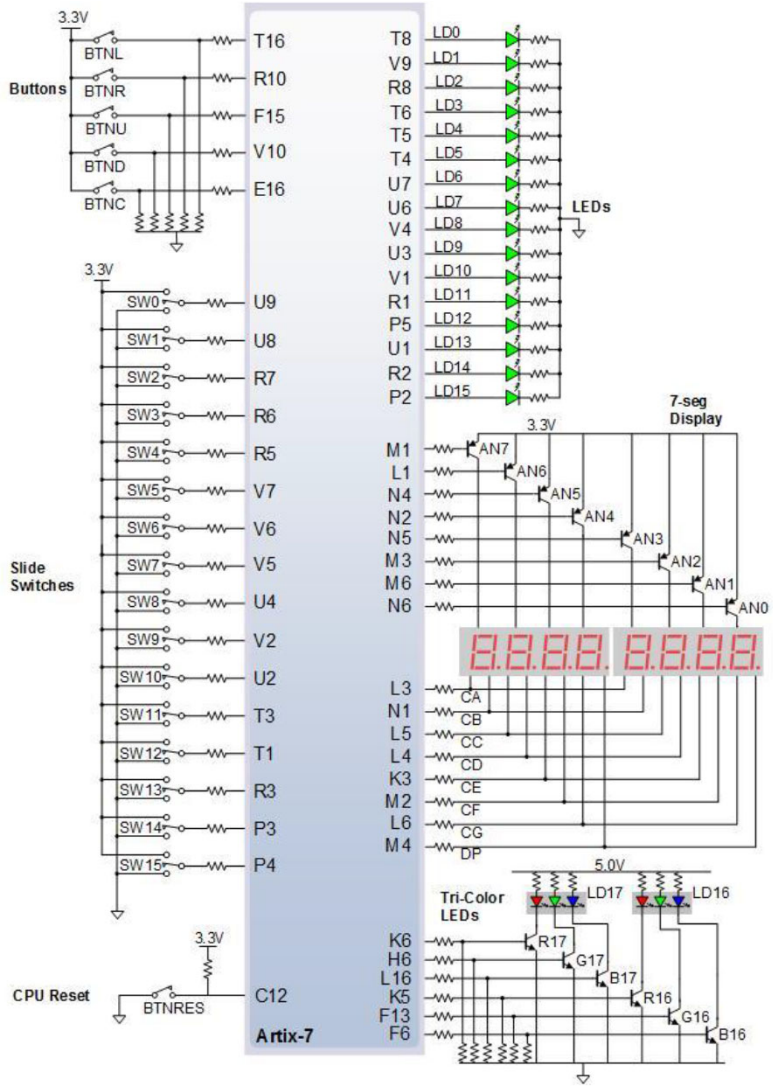
Each tri-color LED has three input signals that drive the cathodes of three smaller internal LEDs: one red, one blue, and one green.

Driving the signal corresponding to one of these colors high will illuminate the internal LED.

The input signals are driven by the FPGA through a transistor, which inverts the signals. Therefore, to light up the tri-color LED, the corresponding signals need to be driven high.

The tri-color LED will emit a color dependent on the combination of internal LEDs that are currently being illuminated.

Note: Driving any of the inputs to a steady logic '1' will result in the LED being illuminated at an uncomfortably bright level. You can avoid this by ensuring that none of the tri-color signals are driven with more than a 50% duty cycle.



Clock Oscillator

The Nexys-4 board includes a single **100MHz** crystal oscillator.

XDC:

```
# Clock signal
#Bank = 35, Pin name = IO_L12P_T1_MRCC_35,
        Sch name = CLK100MHZ
set_property PACKAGE_PIN E3 [get_ports clk]
        set_property IOSTANDARD LVCMOS33 [get_ports clk]
        create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports clk]
```

The input clock can drive MMCMs or PLLs to generate clocks of various frequencies and with known phase relationships.

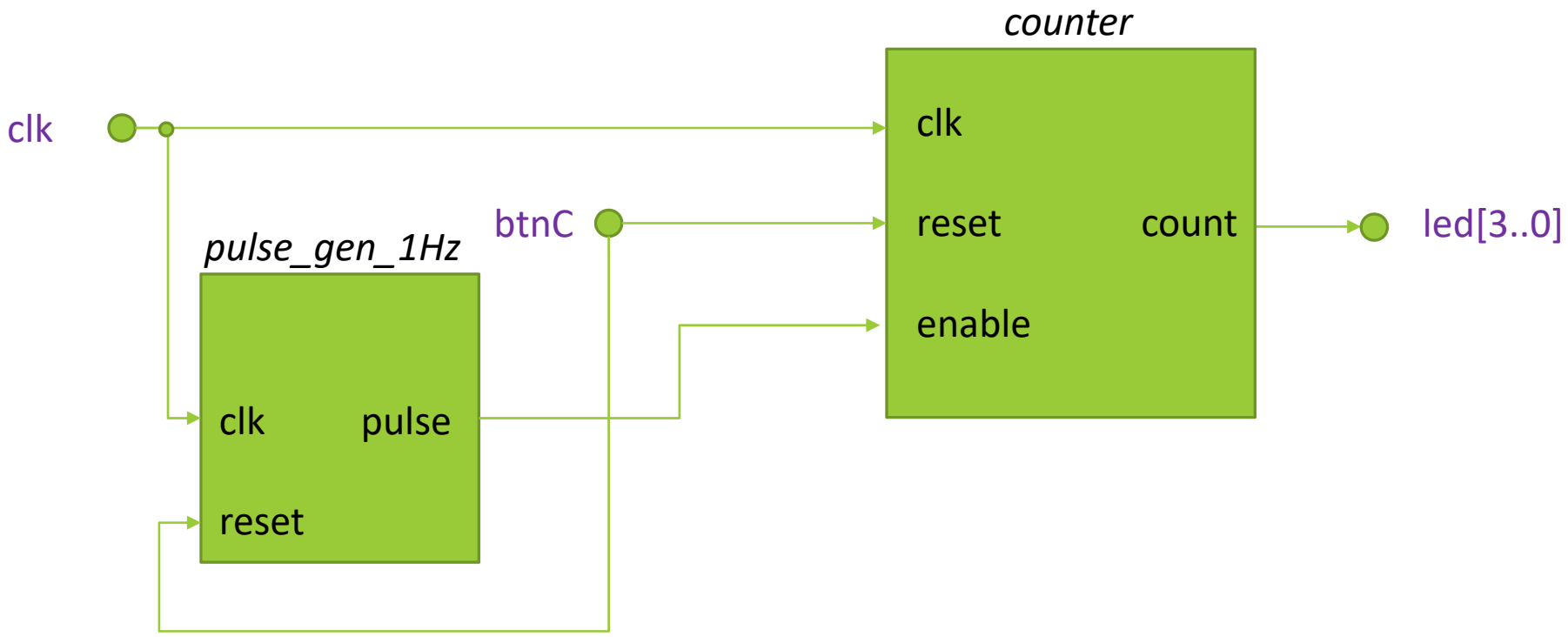
Xilinx offers the Clocking Wizard IP core to help users generate the different clocks required for a specific design.

Artix-7's **clock management tiles (CMT)** provide clock frequency synthesis, deskew, and jitter filtering functionality. CMTs, each containing one **mixed-mode clock manager (MMCM)** and one **phase-locked loop (PLL)**, reside in the CMT column next to the I/O column.

The FPGA xc7a100Tcsg324-1 includes 6 CMTs.

Project Example

Binary up 4-bit counter, updated with frequency of 1Hz, with display of the count value on the board's LEDs.



Final Remarks

At the end of this lecture you should be able to:

- Record the known synthesizable VHDL constructs
- Choose the correct coding style to describe simple combinational and sequential components
- Identify concurrent VHDL statements
- Declare and instantiate components
- Design and use VHDL test benches
- Avoid typical VHDL coding errors
- Create, synthesize, implement, analyze and test simple VHDL projects for Nexys-4 kit in Vivado

To do:

- Complete lab 2 exercises and test them on Nexys-4 kit
- Submit task 4 by March 21
- Decide about April 6