ENGINE

Teaching online electronics, microcontrollers and programming in Higher Education

# Hardware Implementation of Algorithms

1. Testbench and simulation. Frequency divider.

**Lead Partner: Warsaw University of Technology**

**Authors: Lukasz Mik**

University of Applied Sciences in Tarnow

Erasmus+

GRANT NUMBER: 2020-1-PL01-KA226-HE-095653

## Declaration

This laboratory instruction has been prepared in the context of the ENGINE project. Where other published and unpublished source materials have been used, these have been acknowledged.

## Copyright

## Funding Disclaimer

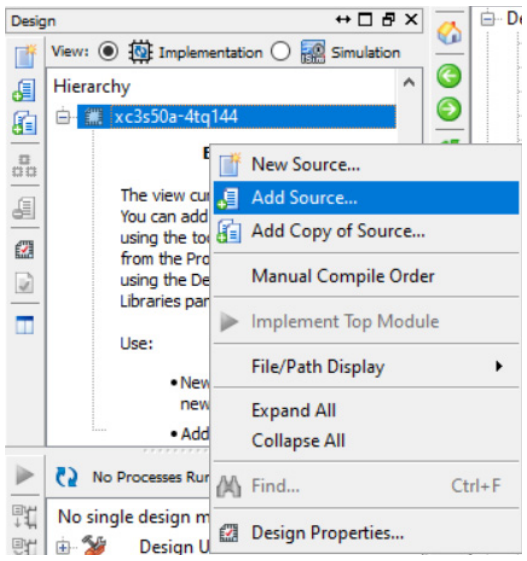
This project has been funded with support from the European Commission. This report reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

# I. Testbench

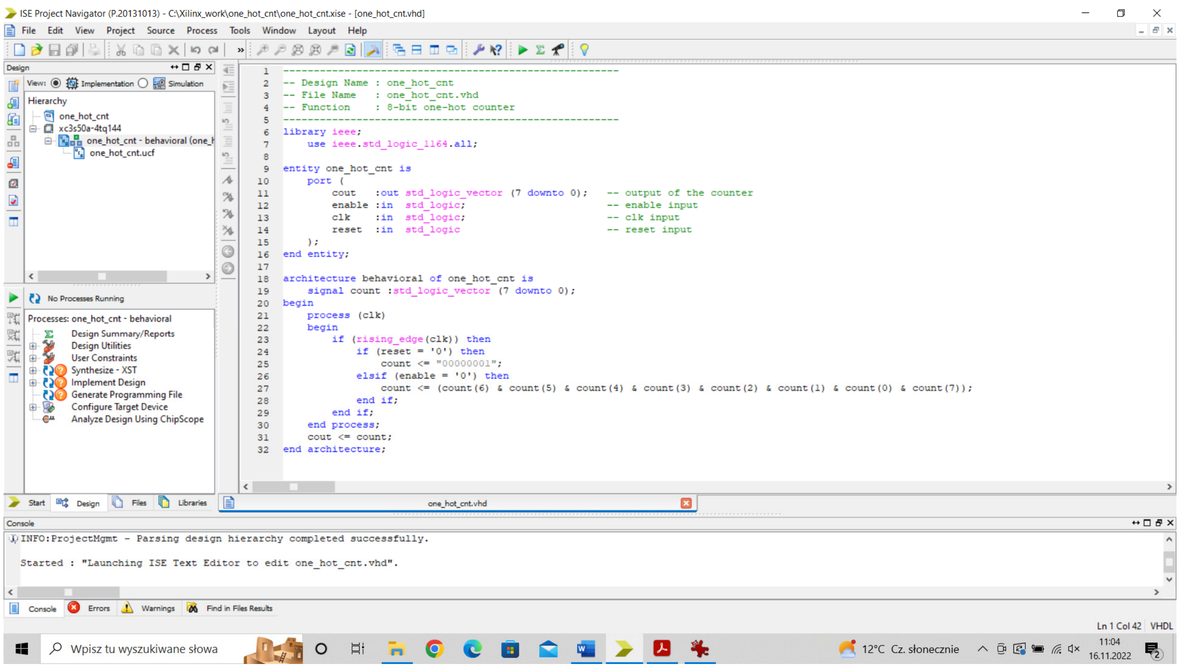
ISE Webpack software (free version of ISE Design Suite) includes tools for writing test procedures, the so-called testbench and projects simulation.  VHDL has many different ways to define the input signals of a test unit. In this class, a basic example will be presented that can be used as a template for more complex testing procedures.

**KROK 1:** Creating a new project in the program using ready modules *.vhd and *.ucf

Launch *ISE Design Suite 14.7* from the desktop shortcut or from *Start → Programs → Xilinx Design Tools → 64-bit Project Navigator*. If there are any projects open, close them all by selecting *File → Close Project*. Then create a new project named one_hot_cnt according to the lab1.pdf instruction, omitting the creation of a new module *.vhd, and only adding the sources included with this exercise: *one_hot_cnt.vhd* and *one_hot_cnt.ucf* files. The method of adding new sources is shown in the figure below.



After creating the project and importing the necessary files, the program window should look like this:


ISE Project Navigator (P.20131013) - C:\Xilinx_work\one_hot_cnt\one_hot_cnt.xise - [one_hot_cnt.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
one_hot_cnt
xc3s50a-4tq144
one_hot_cnt - behavioral (one_h
one_hot_cnt.ucf
No Processes Running
Processes: one_hot_cnt - behavioral
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
Start Design Files Libraries
one_hot_cnt.vhd
Console
Console Errors Warnings Find in Files Results
Ln 1 Col 42 VHDL

```
------------------------------------------------------
-- Design Name : one_hot_cnt
-- File Name   : one_hot_cnt.vhd
-- Function    : 8-bit one-hot counter
------------------------------------------------------
library ieee;
    use ieee.std_logic_1164.all;

entity one_hot_cnt is
    port (
        cout   :out std_logic_vector (7 downto 0);   -- output of the counter
        enable :in  std_logic;                       -- enable input
        clk    :in  std_logic;                       -- clk input
        reset  :in  std_logic                        -- reset input
    );
end entity;

architecture behavioral of one_hot_cnt is
    signal count :std_logic_vector (7 downto 0);
begin
    process (clk)
    begin
        if (rising_edge(clk)) then
            if (reset = '0') then
                count <= "00000001";
            elsif (enable = '0') then
                count <= (count(6) & count(5) & count(4) & count(3) & count(2) & count(1) & count(0) & count(7));
            end if;
        end if;
    end process;
    cout <= count;
end architecture;
```

```
INFO:ProjectMgmt - Parsing design hierarchy completed successfully.

Started : "Launching ISE Text Editor to edit one_hot_cnt.vhd".
```

The design unit configuration has one 8-bit vector output - *cout* and three 1-bit inputs: *clk*, *enable* and *reset*.

In the description of the ring counter architecture, there is one process that has only a clock signal (*clk*) in the sensitivity list. The clock edge is checked in the process. If it is rising, the status of the reset input is also checked. If there is a logical 0 on this input, then the youngest bit of the *count* vector is set to one. Otherwise, the state of the *enable* input is checked. If there is a logical 0 on it, then with each rising edge of the clock, the oldest bit (weight 7) goes to the place of the youngest (weight 0), and this in turn goes 1 position higher.

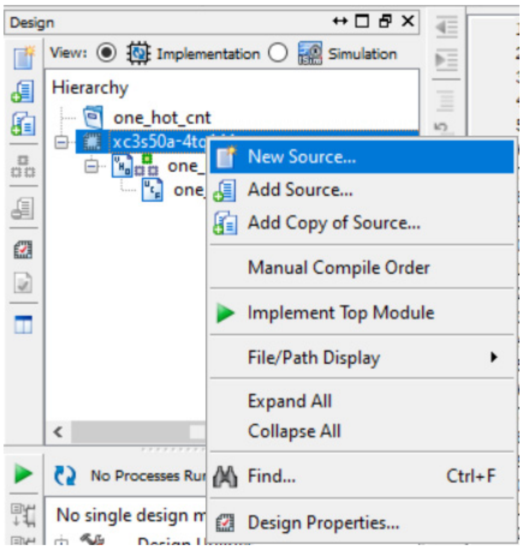
```
process (clk)
begin
if (rising_edge(clk)) then
   if (reset = '0') then
      count <= "00000001";
   elsif (enable = '0') then
       count <= (count(6) & count(5) & count(4) & count(3) & count(2) &
                 count(1) & count(0) & count(7));
   end if;
end if;
end process;
```

The '&' operator is responsible for gluing individual bits into a count vector. The count signal, defined within the architecture, is assigned to the cout output. The `cout <= count` assignment is a concurrent operation with the process (executed in parallel).

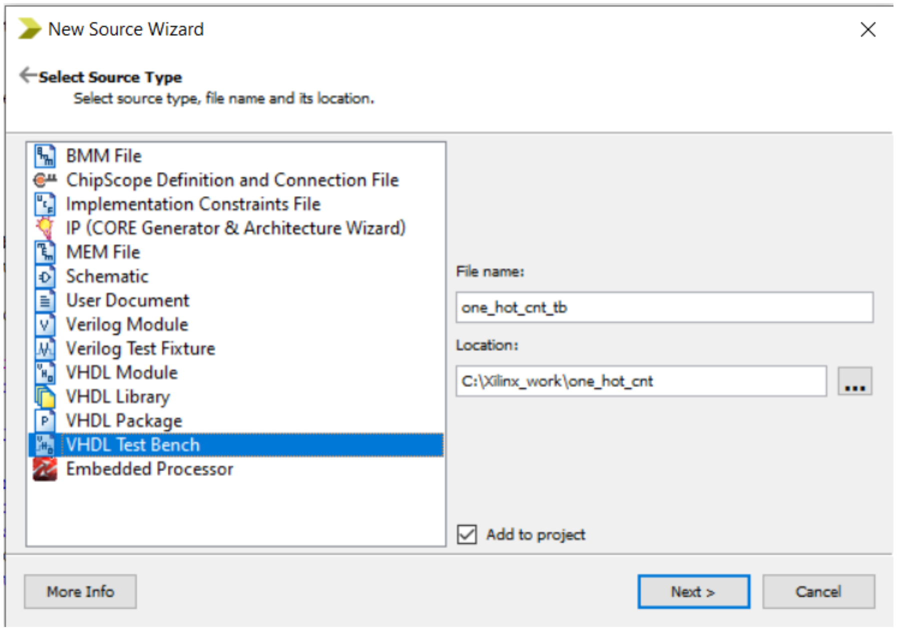
Before generating test vectors, check the project syntactically (double click on *Synthesize - XST*) and then generate the programming file (double click on *Generate Programming File*). Dwukrotne kliknięcie tylko na ostatnią opcję implikuje użycie wszystkich poprzednich tj. *Synthesize – XST* oraz *Implementing Design.*

**KROK 2:** Create a test routine (Testbench VHDL file).

Add a new file to the project by selecting the *New Source* option from the context menu.



Then, from the list of files that can be created, select *VHDL Test Bench*.

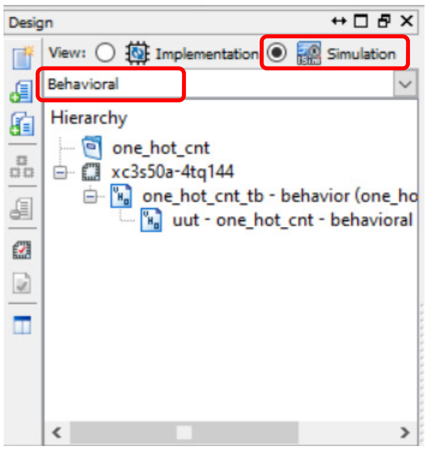


File name: one_hot_cnt_tb

Location: C:\Xilinx_work\one_hot_cnt

[✓] Add to project

In order to easily identify files in the project, it is best to end the name of the file with the test procedure with the string "*_tb*". This is an abbreviation of the word testbench, making it easy to find this file in the future. In the next dialog that appears, select "one_hot_cnt" as the source. In the project window, change the source view from *Implementation* to *Simulation*. The editor will take us to the view of files prepared for simulation.



```
Design
View: ○ Implementation  ● Simulation
Behavioral
Hierarchy
  one_hot_cnt
  xc3s50a-4tq144
    one_hot_cnt_tb - behavior (one_ho
      uut - one_hot_cnt - behavioral
```

From the drop-down list in the same window, the *Behavioral* option should be selected, which means that our design module will be simulated in terms of functionality at this stage. After double-clicking on the name of the file with the test procedure (one_hot_cnt_tb), its automatically generated content will open in the editor window.

```
ARCHITECTURE behavior OF one_hot_cnt_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT one_hot_cnt
    PORT(
         cout : OUT  std_logic_vector(7 downto 0);
         enable : IN  std_logic;
         clk : IN  std_logic;
         reset : IN  std_logic
        );
    END COMPONENT;
```

In the description of the architecture of the test unit, a component has been added, which is our project of the ring counter. It is described with the comment Unit Under Test. Inside the test unit are the signals connected to the counter and the clock period (*clk_period*), defined as a constant. The value of this parameter is automatically set to 10 ns (100 MHz frequency). In our case, the clock frequency is 12 MHz, so the period will be the reciprocal of this number and is approximately 83.3 ns. So we need to change the default value to the one that results from the frequency of the real clock in the system.

```
--Inputs
signal enable : std_logic := '0';
signal clk : std_logic := '0';
signal reset : std_logic := '0';

--Outputs
signal cout : std_logic_vector(7 downto 0);

-- Clock period definitions
constant clk_period : time := 83.3 ns;
```

The program itself automatically generates a clock signal based on a period defined by the user. The architecture of the test unit also includes a map of connections between its signals and the component under test.

```
-- Instantiate the Unit Under Test (UUT)
uut: one_hot_cnt PORT MAP (
       cout => cout,
       enable => enable,
       clk => clk,
       reset => reset
     );

-- Clock process definitions
clk_process :process
begin
   clk <= '0';
   wait for clk_period/2;
   clk <= '1';
   wait for clk_period/2;
end process;
```

In the last part of the test procedure template, a process called *stim_proc* is created, whose task is to generate input signals forcing the tested system. Generating test signals starts with a default 100-second delay. Before this delay, we can set a logic '0' on the reset signal (active state). After this time, we assign logical states '1' and '0' to the reset and enable signals, respectively. This is followed by a wait of 10 clock periods. In the place of the comment *"- - insert stimulus here*" we insert our own test signals. There is no need to generate more force signals at this stage. The code for the *stim_proc* process is shown below.

```
-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   reset <= '0';
   wait for 100 ns;
   reset <= '1';
   enable <= '0';
   wait for clk_period*10;

   -- insert stimulus here

   wait;
end process;
```

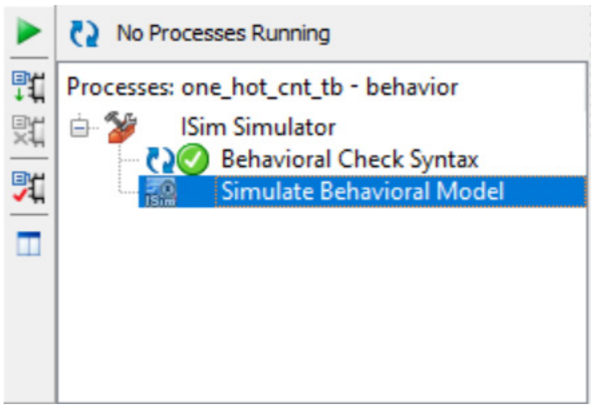
## II. Behavioral and Post-Route Simulation

In the next part of the laboratory, methods of project simulation will be discussed after synthesis, called functional or behavioral simulation, and after implementation in the target system, called post-route simulation, because it takes into account delays in the target system.

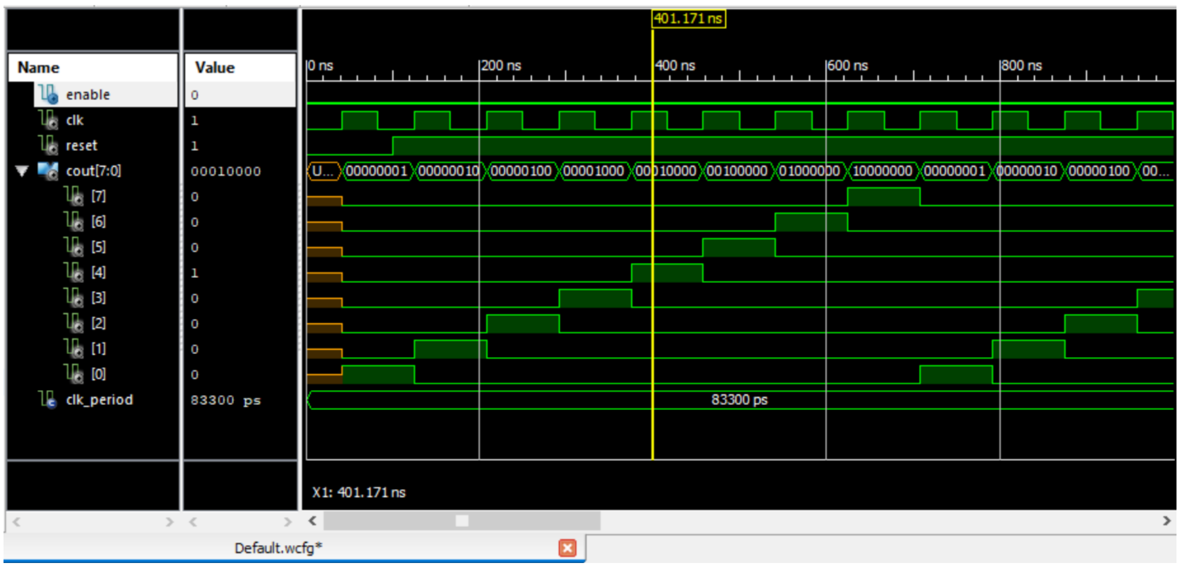
**KROK 1:** Functional (behavioral) simulation.

First, we will simulate the project in terms of functionality. For this, it is only necessary to perform the synthesis (double-clicking the *Synthesize - XST* option) in the project implementation mode.

In the simulator processes window, we first run a syntax check of the test file, and then run a behavioral simulation of the unit under test by double-clicking *Simulate Behavioral Model*.


No Processes Running
Processes: one_hot_cnt_tb - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model

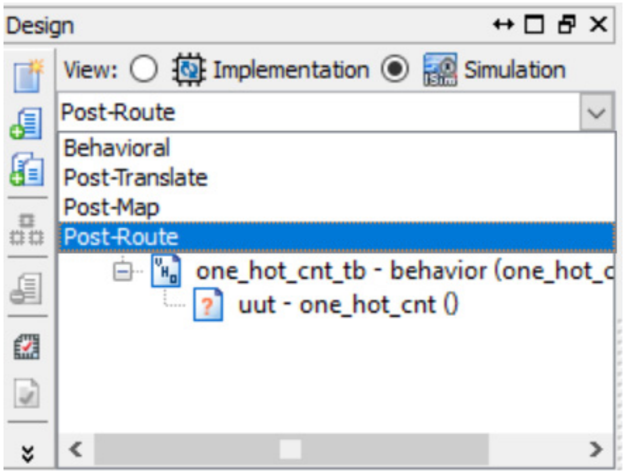
After selecting this option, the simulator window will be launched with all the signals that are the ports of the test unit.



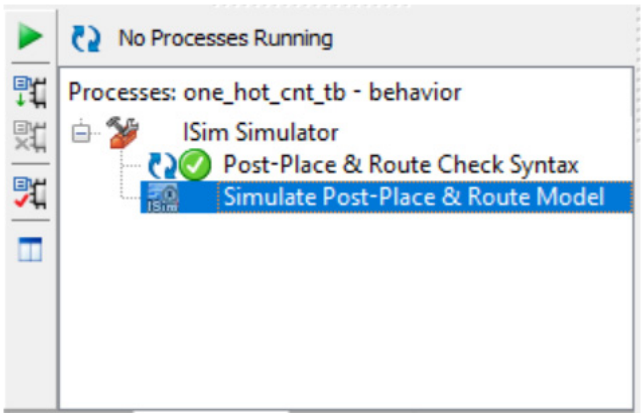
By default, the file with generated waveforms is named *Default.wcfg*. You can change it according to your needs.

**KROK 2:** Post-Route simulation, taking into account delays in the target system.

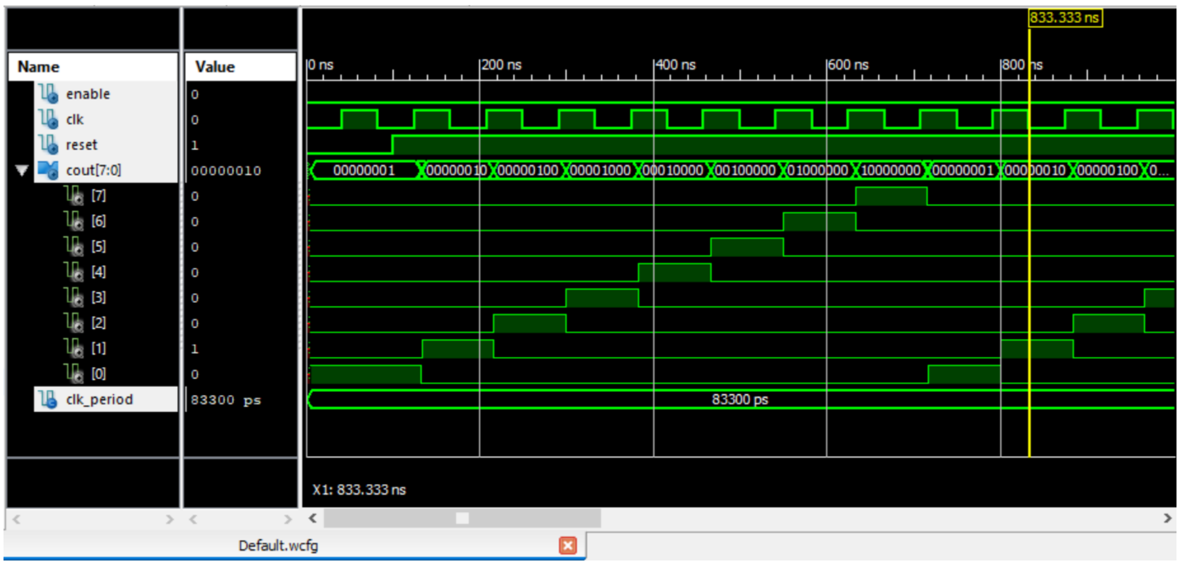
In the simulation type selection window, select the Post-Route option.



In the simulator processes window, we first run the test file syntax check, and then we run the Post-Place and Route Model simulation, which is based on the real model of the integrated circuit for which we are creating the project.


No Processes Running
Processes: one_hot_cnt_tb - behavior
ISim Simulator
Post-Place & Route Check Syntax
Simulate Post-Place & Route Model

The result of the simulation of signals included in the architecture of the design unit can be viewed in the updated window of the simulator's waveforms.



| Name | Value |
|---|---|
| enable | 0 |
| clk | 0 |
| reset | 1 |
| cout[7:0] | 00000010 |
| [7] | 0 |
| [6] | 0 |
| [5] | 0 |
| [4] | 0 |
| [3] | 0 |
| [2] | 0 |
| [1] | 1 |
| [0] | 0 |
| clk_period | 83300 ps |

## III. Frequency divider

Due to the fact that the human eye is unable to react to changes faster than 20 times per second, it is therefore necessary to lower the ring counter clock frequency so that the user can see changes on the LED bar.

Create a new project and add source files named one_hot_cnt_divider.vhd and one_hot_cnt_divider.ucf to it.

Compared to the previous project, a signal called clk_1Hz was defined in the declaration list within the architecture, with an initial value of 0. In the description of the architecture, a new process has been created whose task will be to generate a signal with a frequency of 1 Hz, based on the value of the "pulse_cnt" variable, which will be incremented according to the *clk* clock. The *clk_1Hz* clock will be changed to the opposite state each time the "pulse_cnt" variable reaches the value of 6000000, which is half the period of the clock operating at 1 Hz. In the ring counter process, the clock *clk* is changed to *clk_1Hz*. There was also a change in the name of the design unit - now it is called *one_hot_cnt_divider*. Introducing these modifications to the original design resulted in the ring counter having a bit change rate of 1 Hz.

Compile the project and program the chip with the *.bit configuration file. Check if everything works as described in the architecture, i.e. the reset button sets the logic '1' at the beginning of the LED line (D8) and the *enable* button starts the counter.

**TASK:**

On the basis of the ring counter and frequency divider projects discussed during the classes, build your own circuit that will generate a signal at the audio output (connector J2) with different frequencies, selected using the SW1 - SW6 buttons. The table below shows the frequencies assigned to specific buttons.

| **Pushbutton** | **Signal frequency** |
|---|---|
| SW1 | 396 Hz |
| SW2 | 417 Hz |
| SW3 | 528 Hz |
| SW4 | 639 Hz |
| SW5 | 741 Hz |
| SW6 | 852 Hz |

## References

- *User manual for Elbert V2 - Spartan 3A FPGA Development Board.* https://numato.com/docs/elbert-v2-spartan-3a-fpga-development-board/

- Xilinx iSim User Guide - UG660 (v14.1) https://www.xilinx.com/content/dam/xilinx/support/documents/sw_manuals/xilinx14_1/plugin_ism.pdf

## References

- *User manual for Elbert V2 - Spartan 3A FPGA Development Board.* https://numato.com/docs/elbert-v2-spartan-3a-fpga-development-board/

- *62380 - ISE Install - Installing and Running ISE 10.1 or 14.7 on a Windows 8.1 or Windows 10 machine.* https://support.xilinx.com/s/article/62380?language=en_US