ENGINE

Teaching online electronics, microcontrollers and programming in Higher Education

Hardware Implementation of Algorithms

5. Digital Clock Manager (DCM).

Lead Partner: Warsaw University of Technology

Author: Lukasz Mik

University of Applied Sciences in Tarnow

Erasmus+

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

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 Where other published and unpublished source materials have been used, these have been acknowledged.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.

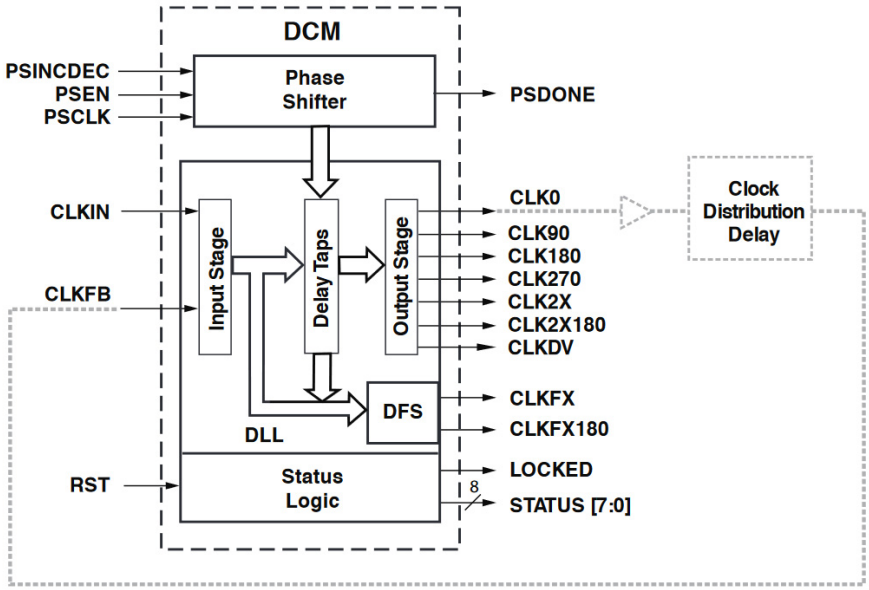
I. Distribution and management of clock signals in the Spartan-3 family.

From the point of view of the designer of a digital system using FPGAs, it is necessary to properly clock all the components of this system. Logical elements located on the surface of the semiconductor structure, due to the internal structure of the FPGA, are not clocked simultaneously. The following factors affect the signal distribution time:

- route length - distance between source and destination
- type of route - depends on the connection resources used to transport the signal
- the number of inputs clocked simultaneously in a given segment of the connection path

To prevent the risk of the synchronization effect disappearing, global clock lines are used in FPGAs marked as GCLKx, where x is the line number. Assigning the clock line to the GCLK pin informs the design program that the user wants to use the global clock line.

In addition, to minimize the impact of FPGA architecture imperfections on the quality of projects implemented in them, Xilinx equipped Spartan 3A systems with DCM blocks (their number depends on the logical resources of the FPGA).



Block diagram of DCM in Spartan 3

The DCM consists of 4 basic elements: DFS (*Digital Frequency Synthesizer*), DLL (*Delay Locked Loop*), programmable *Phase Shifter* and *Status Logic.*

Using DCM, it is possible to compensate for the phase difference of the clock signals at physically different places in the silicon structure of the chip. Thanks to the built-in DLL loop with adjustable delay line, it is possible to synthesize internal clock signals (including multiplication or division of the frequency of an externally connected clock signal). The input signal for the DLL is given to the CLKIN input. The CLKFB input is used to optionally provide a feedback signal for the feedback loop, thanks to which the DLL block can monitor the quality of the generated clock signal. The DLL block is equipped with four signal outputs that are copies of the signal from the CLKIN input, of which 3 are shifted in phase relative to it by: 90°, 180° and 270°. On these outputs it is possible to obtain 50% duty cycle signals provided that the *DUTY_CYCLE_CORRECTION* parameter is set to *TRUE*. The CLK2X and CLK2X180 outputs generate a signal with a duty cycle of 50%, a frequency twice as high as the frequency of the input signal and phases of 0° and 180°. The CLKDV output can be divided by the following values: 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 9, 10, 11, 12, 13 , 14, 15 or 16. The duty cycle of this output is 50% when the division factor is an integer.

The signals at the CLKFX and CLKFX180 outputs (shifted by 180° relative to each other) are generated by the DFS frequency synthesizer, based on the signal fed to the CLKIN input. The range of acceptable input frequencies of the synthesizer is in the range from 1 to 280 MHz. The frequency of the signal at said outputs is obtained by dividing or multiplying the frequency of the input signal according to the formula:

$$f_{CLKFX} = f_{CLKIN} \times M/D$$

where: M = {2 … 32}, D = {1 … 32}

The DCM block is also equipped with an RST reset input, which restores the default (user-defined) configuration, and a LOCKED output, used to indicate that the DLL has synchronized with the signal at the CLKIN input.

In advanced designs (at high clock frequencies), it is necessary to use a phase shifter that can be dynamically controlled. However, it will not be used at this stage of the course.

II. Increasing the frequency of a clock signal using a DCM block.

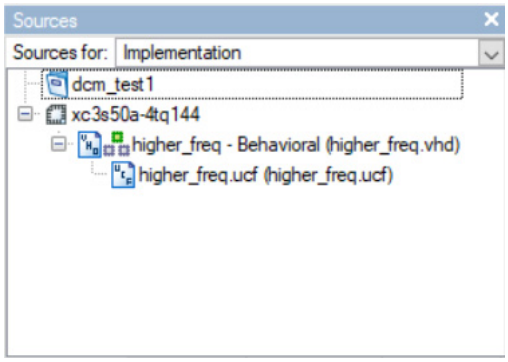
We create a new project called *dcm_test1* in ISE WebPack by entering the appropriate parameters of the target FPGA. We create a new VHDL source file (VHDL Module) named *higher_freq*. In the design unit, we add one input port named CLK and one output port named LED. After adding the ports, the design unit declaration with the header part will look like this:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity higher_freq is
    port ( CLK : in  std_logic;
           LED : out  std_logic);
end higher_freq;
```

In the next step, we add the UCF file with the assignment of unit ports to FPGA pins. When creating a new source file, select *Implementation Constraints File* and enter *higher_freq* as the file name. In the created file you need to add 2 lines with pin assignments..

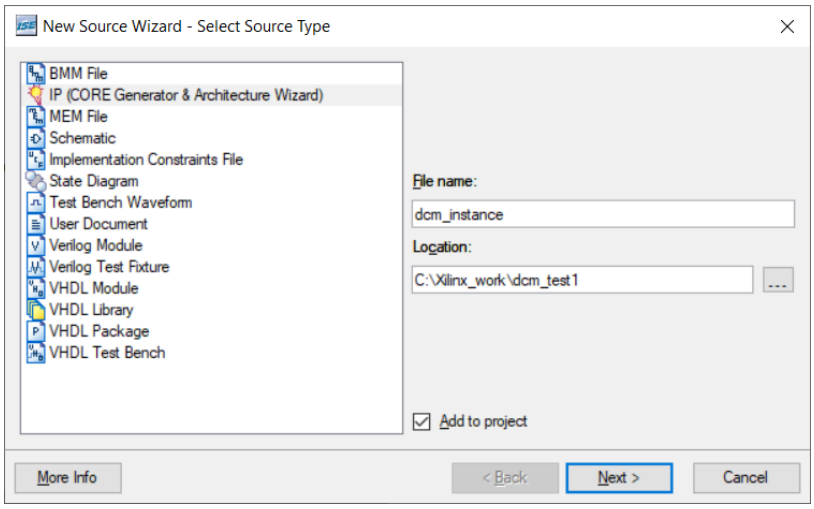
```
NET "CLK"  LOC = P129  | IOSTANDARD = LVCMOS33 | PERIOD = 12MHz;
NET "LED"  LOC = P46   | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 12;
```

After the files have been added correctly, the window with the preview of the source files should look like this:

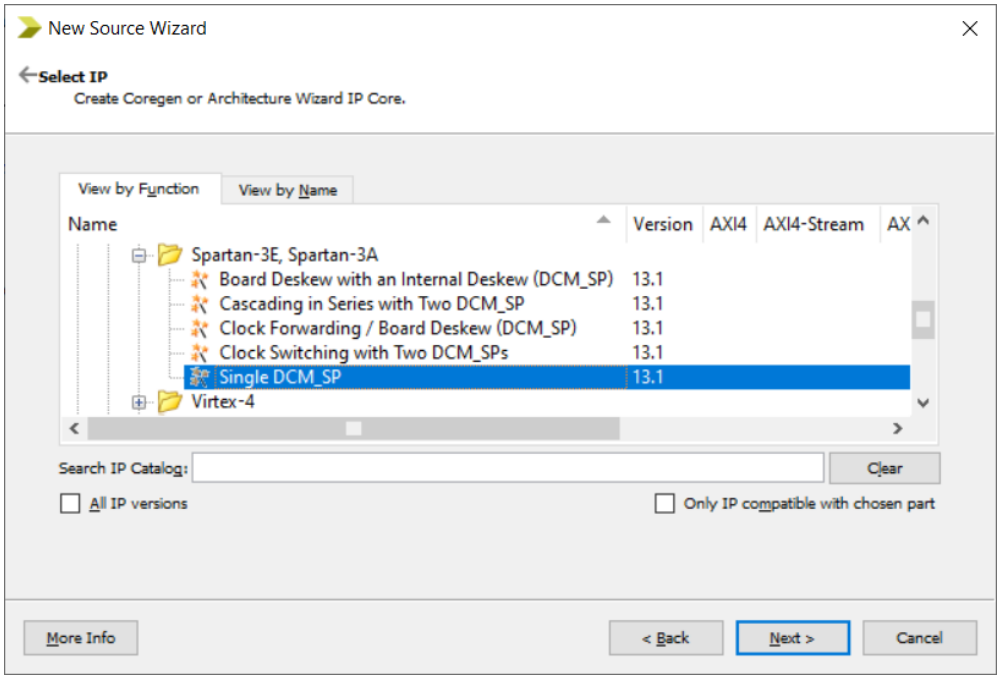


As you can see, the name of the project does not have to match the name of the source files. The names of the source files also do not have to coincide with the name of the project unit. This file naming style only makes it easier to work with projects that contain many components.

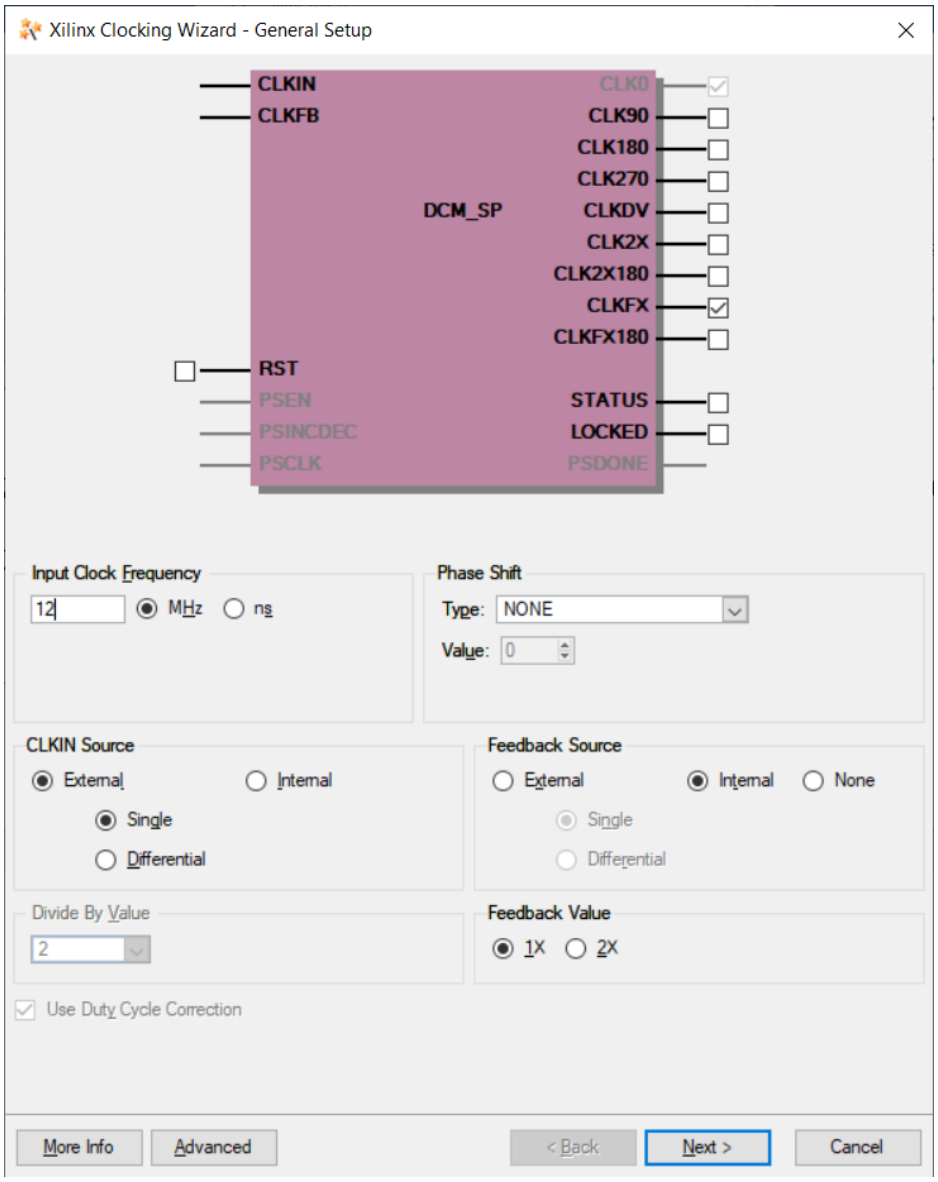
In the next step, we add a ready IP Core block to the project using the IP tool (*CORE Generator & Architecture Wizard*). By right-clicking in the Sources window or selecting the *Project → New Source* option from the horizontal menu ... Then select the IP option (*CORE Generator & Architecture Wizard*) and enter *dcm_instance* as the file name.



After clicking the *Next* button and moving to the next window, expand the *FPGA Features and Design → Clocking → Spartan 3E, Spartan 3A* group and select the *Single DCM_SP* option.

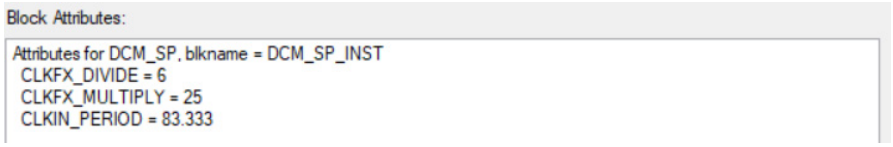


After a while, a window will open with the selection of the output file type - select VHDL and click *OK*, after which a window with DCM block settings will open.

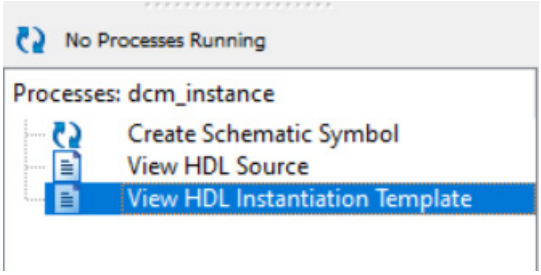


In this window, we enter the frequency of the input clock signal: 12 MHz. Uncheck the RST and LOCKED lines (we will not use them) and check the CLKFX line - clock output with a higher frequency. After confirming the settings with the Next button, in the next window we confirm the use of global buffers for the clock lines indicated in the DCM block. The frequency synthesizer (DFS) configuration will appear in the next window. We set the target output frequency at 50 MHz.

After proceeding, a window with a summary of DFS settings will appear.



The division factor of the frequency of the clock signal from the CLKIN input has been set to 6 and the multiplication factor to 25. The frequency at the CLKFX output will then be: f_{CLKFX} = 12 MHz x (25/6) = 50 MHz, i.e. as much as we specified in the configuration window. After the DCM block is successfully generated, a file named *dcm_instance.xaw* will be added to the project. By double-clicking on the name of this file we will re-enter the DCM configuration mode. If we select the file name, in the *Processes* window we can start the preview of the component template, which must be inserted into the file with the architecture description.



The content of the *dcm_instance.vhi* file from which we copy this fragment will open in the editing window:

```
COMPONENT dcm_instance
PORT(
	CLKIN_IN : IN std_logic;
	CLKFX_OUT : OUT std_logic;
	CLKIN_IBUFG_OUT : OUT std_logic;
	CLK0_OUT : OUT std_logic
	);
END COMPONENT;
```

Then we paste it inside the architecture in the declaration part, i.e. before the *begin* keyword that begins the behavioral description.

To test the correct operation of the project, we will define a signal called *clk_50MHz* inside the architecture, which we will connect to the CLKFX_OUT output of the DCM block. Then we will insert a process that will have a *clk_50MHz* signal in the sensitivity list, and the task of this process will be to generate a *clk_1Hz* signal with a frequency of 1 Hz, which we will directly connect to the LED output port.

Finally, it will be necessary to connect the DCM block to the signals inside the architecture. The connection of the inserted component is done through the **port map** keywords, followed by signals in the architecture to the ports of this component in brackets. Since we will only use the CLKIN_IN and CLKFX_OUT lines, we can leave the others unconnected as shown in the example below.

```
DCM1 : dcm_instance port map(CLKIN_IN => CLK , CLKFX_OUT => clk_50MHz);
```

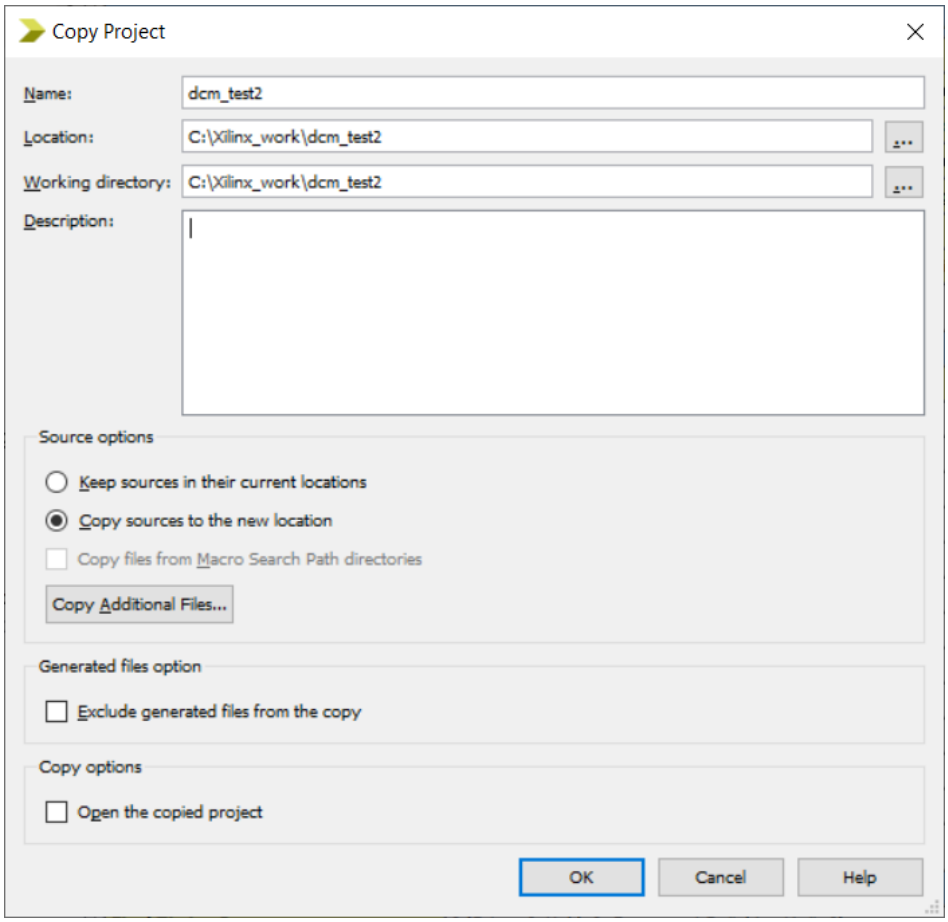
The component label, e.g. DCM1, is necessary when placing it in the project. The complete description of the architecture will take the form as in the listing below.

```
architecture Behavioral of higher_freq is

COMPONENT dcm_instance
PORT(
	CLKIN_IN : IN std_logic;
	CLKFX_OUT : OUT std_logic;
	CLKIN_IBUFG_OUT : OUT std_logic;
	CLK0_OUT : OUT std_logic
	);
END COMPONENT;

signal clk_50MHz : std_logic := '0';
signal clk_1Hz : std_logic := '0';

begin

process(clk_50MHz)
variable counter_1 : integer := 0;
begin
	if rising_edge(clk_50MHz) then
		if counter_1 < 25000000 then
			counter_1 := counter_1 + 1;
		else
			counter_1 := 0;
			clk_1Hz <= not clk_1Hz;
		end if;
	end if;
end process;

LED <= clk_1Hz;
DCM1 : dcm_instance port map(CLKIN_IN => CLK , CLKFX_OUT => clk_50MHz);

end Behavioral;
```

After compiling the project and generating the configuration file, program the FPGA and check if the LED flashes with a frequency of 1 Hz. Please note that the configuration file will be named *higher_freq.bit* (or *higher_freq.bin* if the user sets it in the file generator options).

III. Decreasing the frequency of the clock signal using a DCM block.

An example of generating a clock signal with a frequency lower than the input clock signal will be discussed in the example from the previous section. To do this, copy the project under a different name. From the *File* menu, select the *Copy Project ...* option and in the dialog box that appears, enter a new name for the project, i.e. *dcm_test2*.

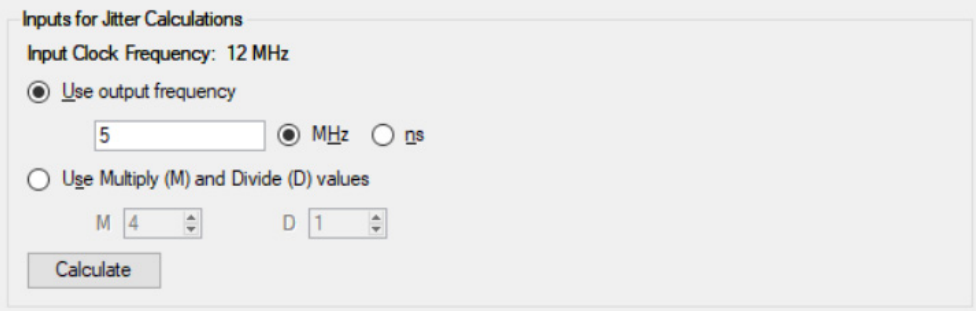


Then close the current project (*File → Close Project*) and open the *dcm_test2* project. Since the source files *.vhd and *.ucf still have the old name, they should be removed from the project. Then we rename them in the project folder to *lower_freq.vhd* and *lower_freq.ucf* and re-add them to the project with the new names. We also rename the design unit to *lower_freq*.

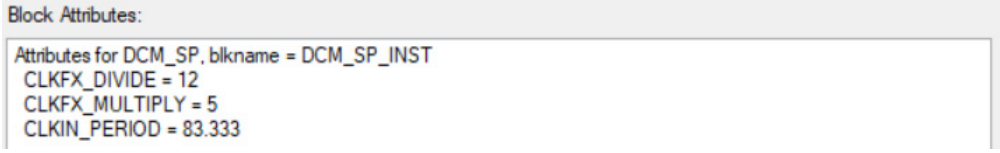
```
entity lower_freq is
    port ( CLK : in  std_logic;
           LED : out  std_logic);
end lower_freq;
```

Then double-click on the file named dcm_instance.xaw, thanks to which we will go to the dialog box with the configuration of the DCM block parameters. The only thing we change

in the configuration is the frequency of the clock signal at the CLKFX_OUT output. We set the value to 5 MHz.



The configurator will calculate the CLKFX_DIVIDE and CLKFX_MULTIPLY parameters for the frequency synthesizer itself.



In the old architecture description, we rename the clk_50MHz signal to clk_5MHz. In the process generating the clk_1Hz signal, we change the limit value of the counter from 25000000 to 2500000.

TASKS:

- Use the CLKFX and CLKFX180 outputs to drive 2 different LEDs. Remember that these diodes must be controlled by 2 different 1 Hz clock signals associated with the listed outputs of the DCM block.
- Use the CLK0, CLK90, CLK180 and CLK270 outputs to control 4 different LEDs. Remember the dependence of the 1 Hz clock signals mentioned in the previous point.

References

- J. Majewski, P. Zbysiński – *Układy FPGA w przykładach.* Wydawnictwo BTC, Legionowo 2007.

- UG331 – Spartan-3 Generation FPGA User Guide
  https://docs.xilinx.com/v/u/en-US/ds529