EDA234 Digital Design Lab 1

Lab Tutorial

Designed by: Anurag Negi, Angelos Arelakis, Ioannis Sourdis
Modifyed by: Torbjörn T och Arne L

In this tutorial you will familiarize yourself with two sets of tools – the first, called QuestaSim (or ModelSim – no difference between these two tools for the course requirements) lets you simulate your VHDL hardware descriptions, while the second, called Xilinx Vivado, allows you to synthesize your designs and target a variety of hardware platforms. This lab assumes that you are at least aware of what FPGAs are and where they can be used.

We will see how to use the following tools in this lab:

1. QuestaSim/ModelSim: This tool allows you to compile and simulate descriptions of digital logic in VHDL. A student edition is available free for Windows.
2. Vivado: This tool allows you to synthesize your VHDL designs and map the resulting logic onto a variety of target devices. We will target the Nexys4 board equipped with a Artix-7 (XC7A100T-1CSG324) device. Note the number in parenthesis, as it would be needed to correctly setup your development environment when using Xilinx Vivado. Again, a free student license is available for this software.

This lab will require you to complete three tasks:

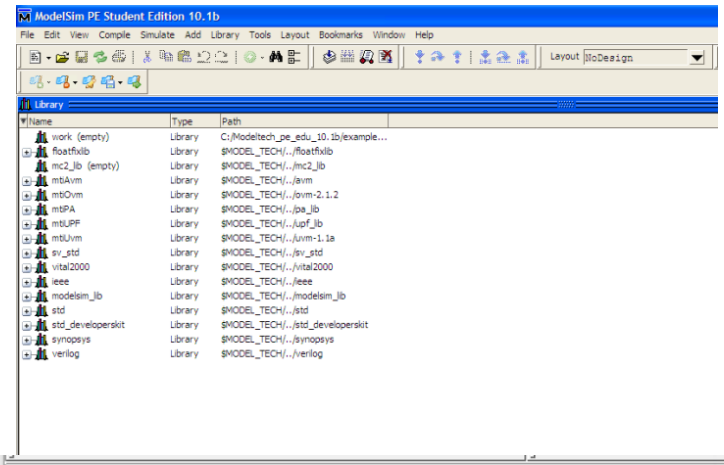
1. Learn to recognize and understand the use of several commonly occurring digital logic components.
2. Simulate a sample VHDL design in QuestaSim/ModelSim and check for correctness of operation using a "do" file.
3. Synthesize the sample VHDL design targeting a Xilinx Artix-7 FPGA platform and read off some relevant design parameters from reports.

Please download tutorial_files.zip from canvas and extract its contents in a local folder which you can access from within QuestaSim/Modelsim and Vivado. The .zip archive contains tree files: Cnt4bDec.vhdl, tst_lab0.do and Cnt4bDec_tb.vhdl.

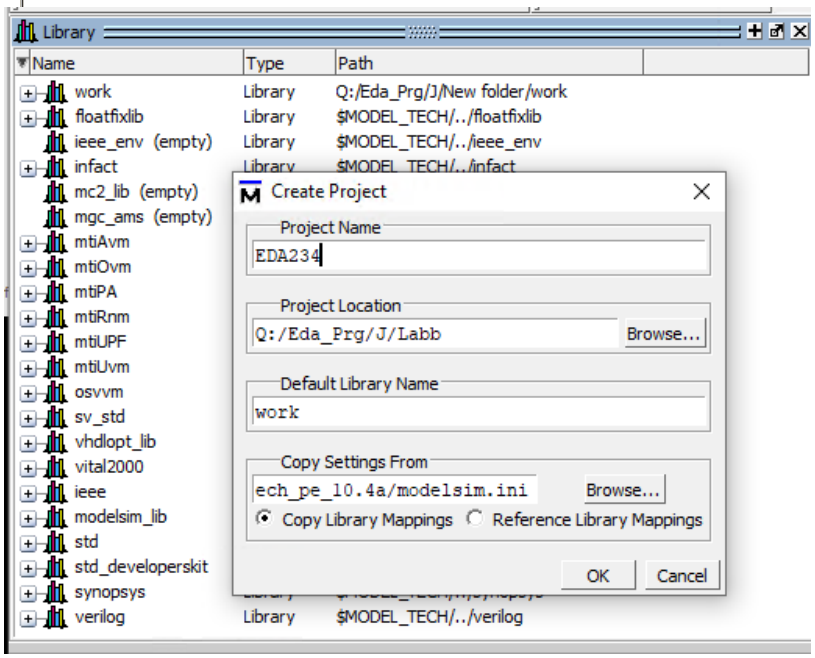
There are two hand ins that you should submit in Canvas.

Using QuestaSim/ModelSim for simulating VHDL designs – Task 1

Start QuestaSim/ModelSim.
You should see a window similar to the one shown.

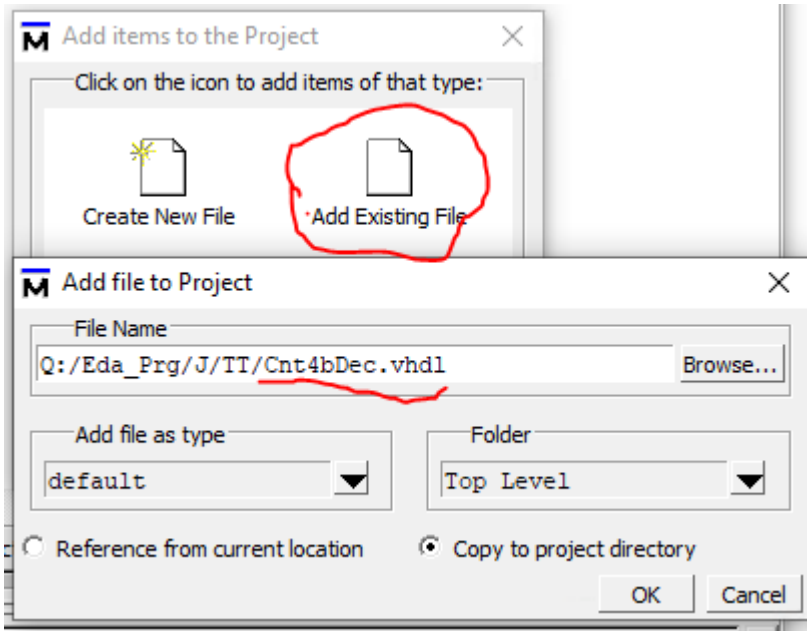


We will now create a new project, by selecting File->New->Project.
Name the project EDA234.
Choose a suitable Project Location so that your code is maintained in a secure place.

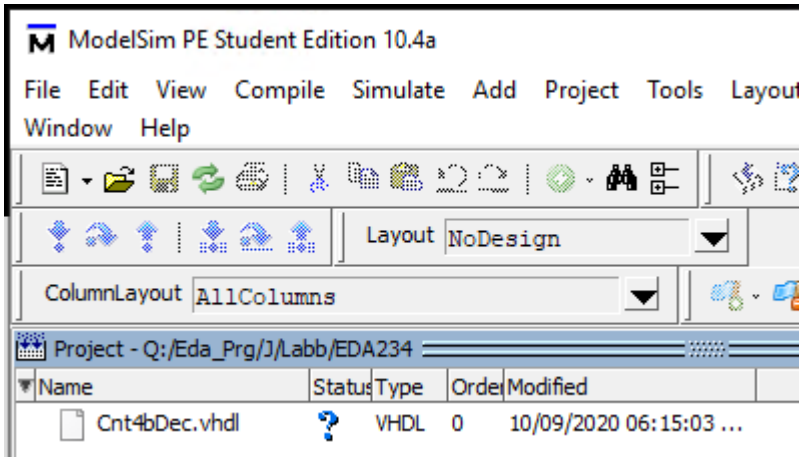


Once you click OK, the software will ask you to add items to the project.
Select "Add existing file" and add the VHDL file (with the extension ".vhd" or ".vhdl") to the project.
Add the file Cnt4bDec

You can add files to the project at a later point in time as well.

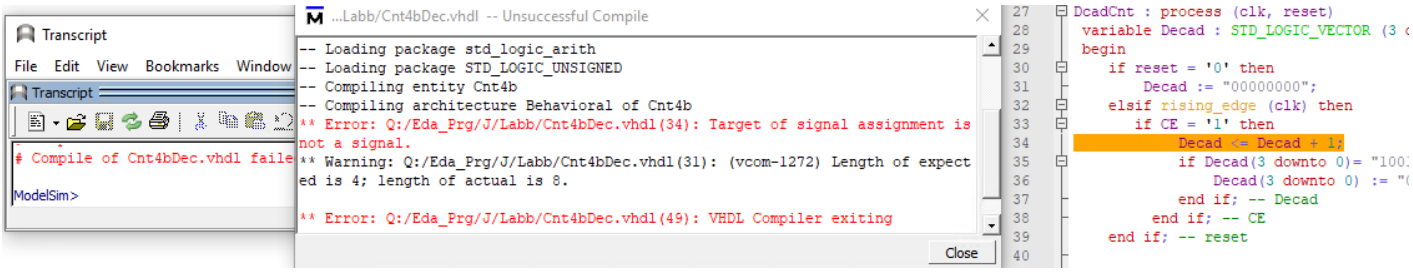


The window should now be similar to:

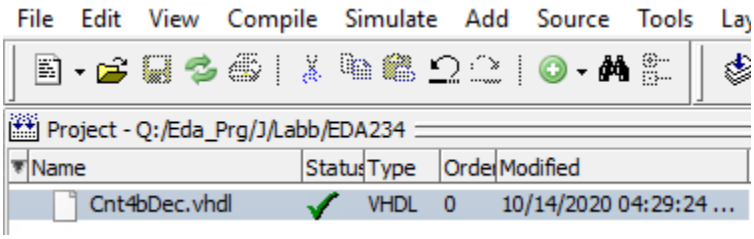


Double clicking on the file name will open the file for editing in an editing panel.
You can now take a look at how the description of a simple hardware component looks like in VHDL.

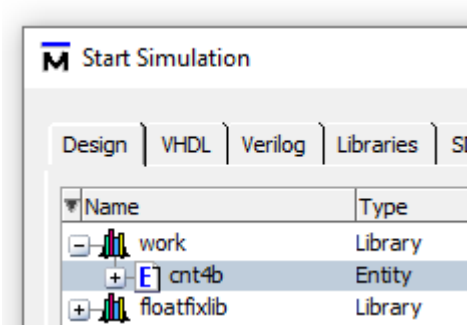
You can compile it by selecting Compile->Compile All from the menu.
If there are errors or warnings, you can see them by double clicking on the error message (appeared in red) in the Transcript window. A new window appears where the errors/warnings are listed, like in the figure below. Using this information you can fix them, save and recompile.



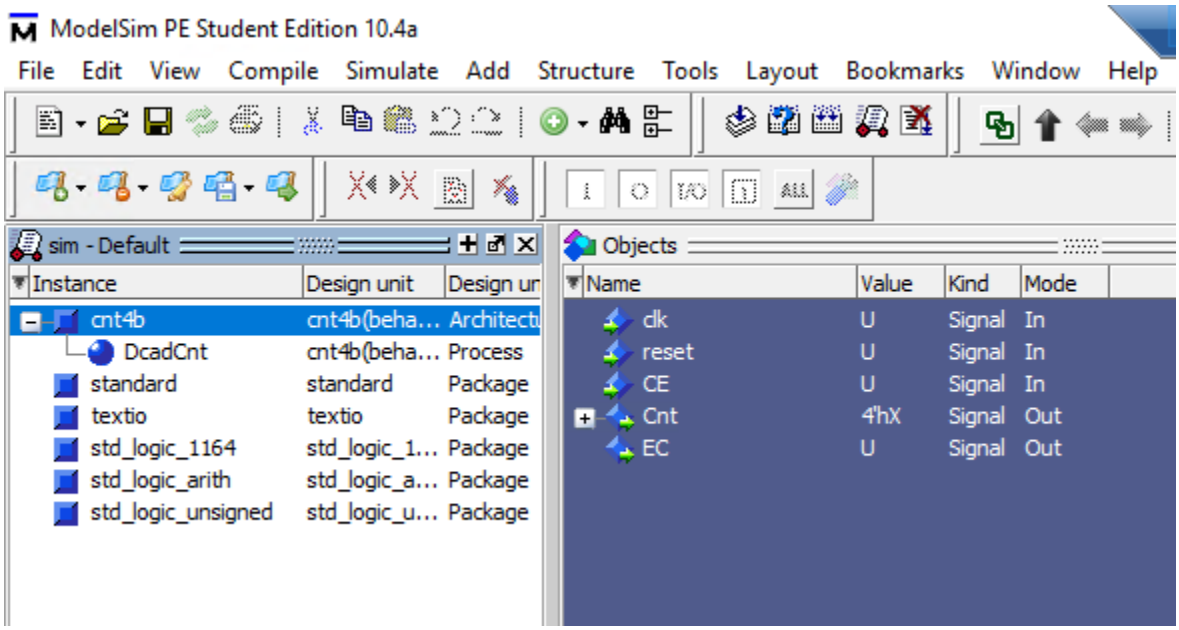
When compilation is successful, you should be able to see a green tick mark in the status field beside the file name. (If it's yellow you need to fix some warnings.)



You can simulate the design by selecting Simulate -> Start Simulation.
It will present a dialogue box asking you to choose the design entity that you want to simulate. In the "work" library, choose the name of the entity (or the top level entity if you have several entity descriptions comprising your design).
In this tutorial, the entity name "cnt4b"is the top level entity in your default "work" library.

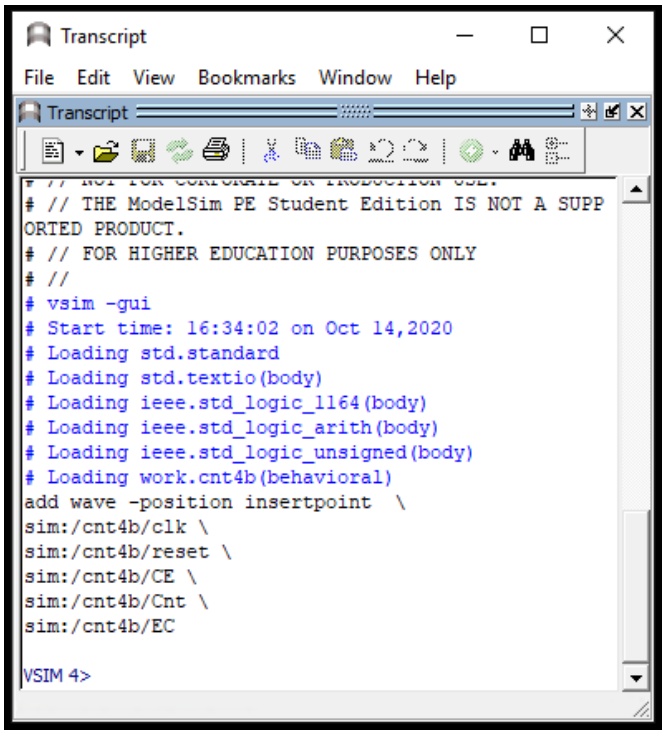


You should now see a window that looks similar to the one shown below:



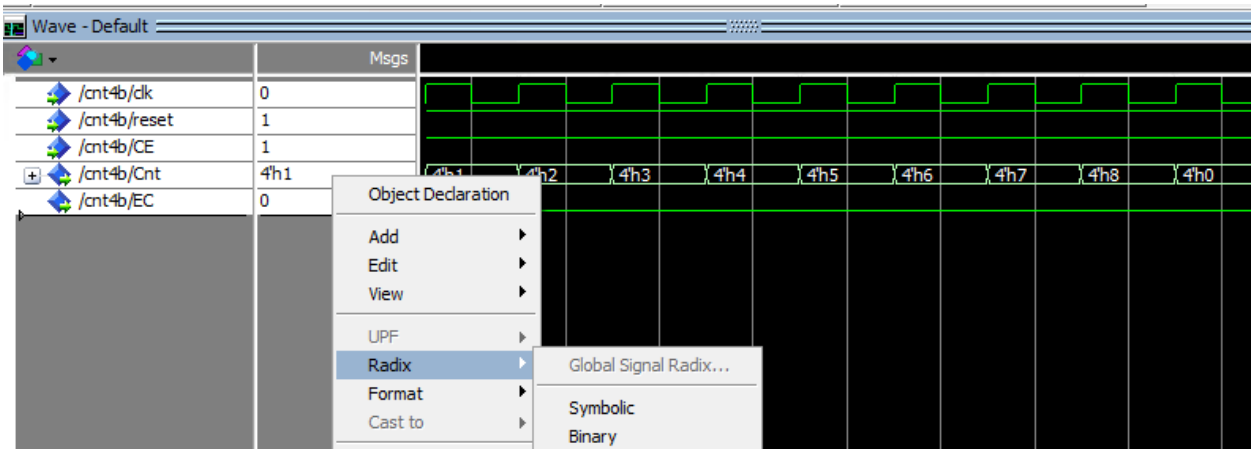
Signal names in the simulated entity appear in the Objects panel. You can right click on signal names you want to view graphically in the panel and select "Add Wave". A new waveform panel will appear with the signal(s) shown. Alternatively, you can select the signals (in Objects) you would like to add in the waveform and then drag and drop them in the signal area.

Also notice the Transcript panel at the bottom of the QuestaSim/Modelsim window, which also displays a command line.



You can now enter commands to test your module.
Type "do tst_lab0.do" to run the do file you have downloaded.
Make sure that the "do" file is placed in the working directory for the project.

You can see that values in the waveform window are updated when the run command is executed. The figure below shows various signal transitions when a VHDL module is simulated. You can zoom in/out using the respective "magnifying glass" tools. You can also change the radix which the signals appear in by right clicking on a signal (or a set of them) and selecting radix. A signal can be added a number of times using different radixes (can be done through the graphic interface or inside the "do" file), which can be practical.



The "do" file contains a sequence of commands that can also be entered at the console. A quick guide and a command reference have been provided for your guidance on Canvas.

Take some time at this point to familiarize yourselves with the syntax of the *.do* file and with the features available in the simulator -- like adding/removing signals from the waveform window, zooming in and out, cursor control, changing the way values are represented in the window (radix). Getting familiar with these controls will help you test and debug code more effectively – a skill which will be valuable throughout the lab assignment.

Try to answer the following questions:

1. Do the code work?

Testbenches

A more efficient way of testing the design is by using VHDL and write a testbench.

End the simulation, Simulate→ End Simulation.

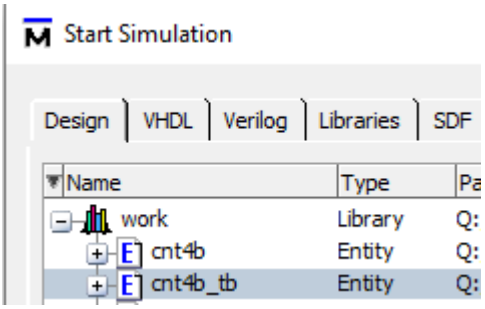
And add a testbench.
Project→Add to Project→ Existing File, and add "Cnt4bDec_tb.vhdl"

Create an .do file that restart the simulator and add all internal signals in the testbench.

```
restart -f -nowave
view signals wave
add wave reset_n_tb_signal clk_tb_signal count_tb_signal ce_tb_signal
ec_tb_signal
run 2550ns
```

Compile and simulate using the testbench.

Run the do file and check the result.



Task:
Fix the counter design for the errors:

```
# ** Error: Error for reset_n=1 and count=1001 and EC=1
#    Time: 1250 ns  Iteration: 0  Instance: /cnt4b_tb
# ** Error: 1350 ns Error for reset_n=1 and count=0000 and EC=0
#    Time: 1350 ns  Iteration: 0  Instance: /cnt4b_tb
# ** Error: 1450ns Error for reset_n=1 and count=0000 and EC=0
#    Time: 1450 ns  Iteration: 0  Instance: /cnt4b_tb
```

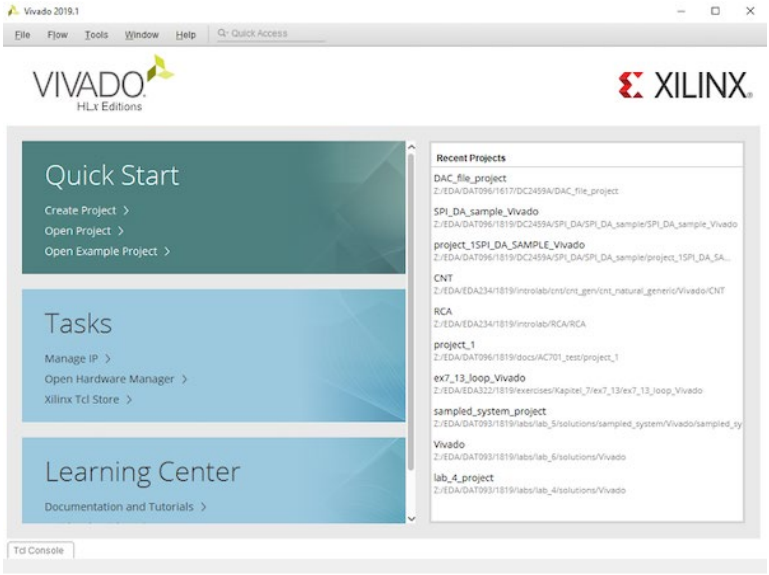
You can also update the "Cnt4bDec_tb.vhdl" to minimize the numbers of warnings.

Using Vivado for Synthesizing a design – Task 1

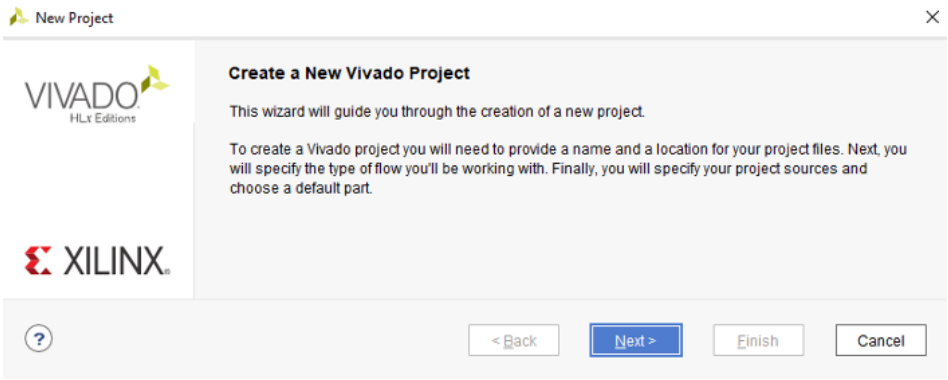
We will use Vivado to synthesize an design. Synthesis converts a VHDL description into a netlist of components that can be mapped onto a target device like an FPGA. For this lab we will target a Xilinx Artix-7 FPGA (**XC7A100**) in a **CSG324** package. The design tool also does the mapping onto the FPGA and applies board constraints (which will mapping top level design ports onto pins on the FPGA board).

Start the Vivado by clicking

on the icon  on the Desktop
and click on Create Project in Quick Start …

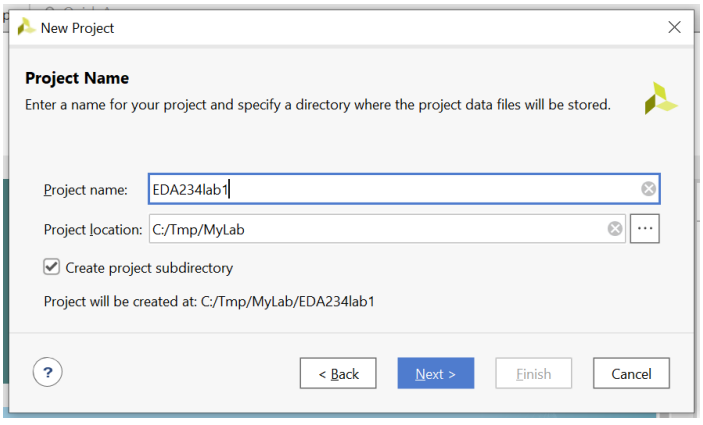


When you click to create a new project,
you will first get an information window

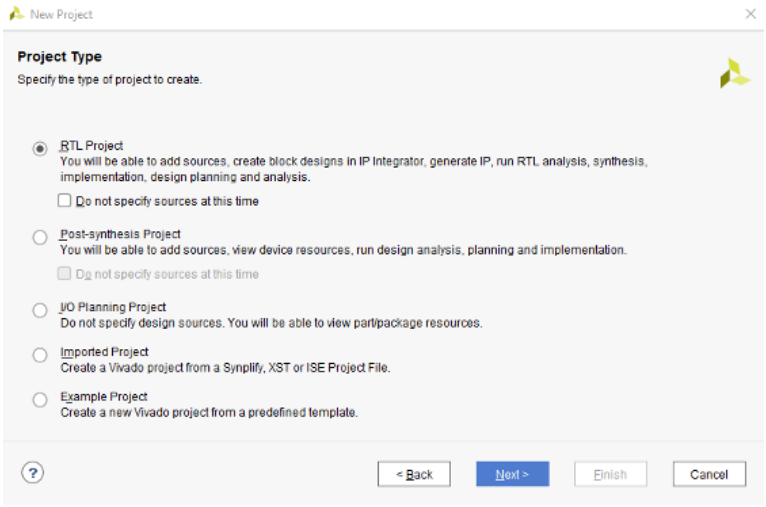


Click on  and you get a dialogue box, as shown below, should open up. Give your project a name and provide a location and a working directory.

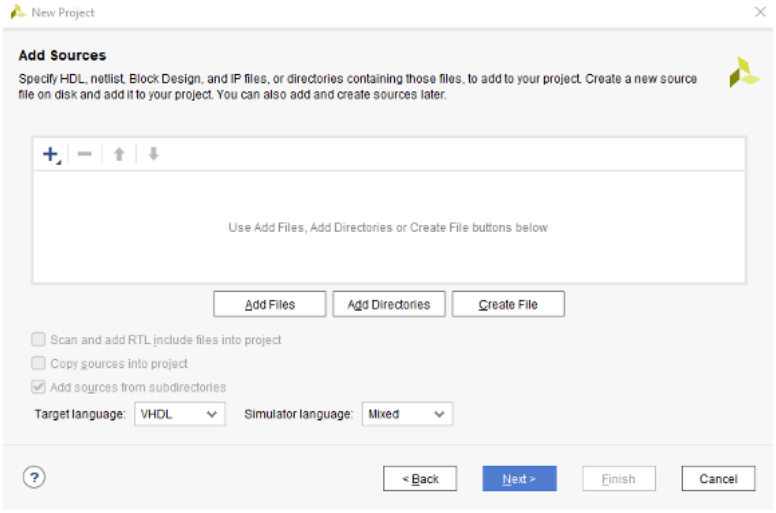
For this example, C:/it's used but it's recommended to use your own network drive Z:/ instead and create a project folder for the lab and all its files.



After choosing the project name and location, then click on Next > and choose RTL as type of project.

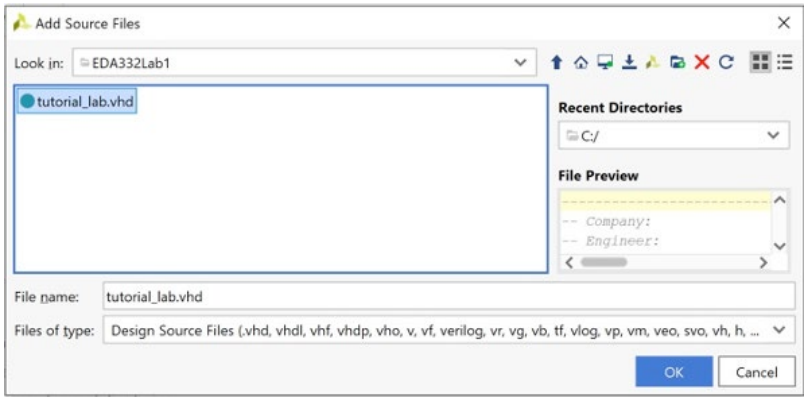


Click on Next > again to **Add Source** and setup the Vivado project.

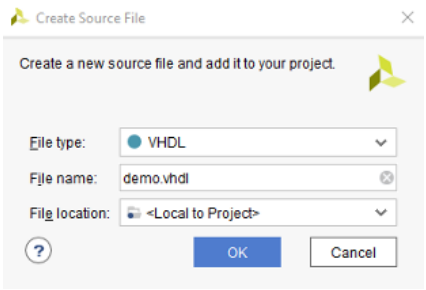


Choose "Target language" as VHDL since the design will be written VHDL. Note that when you create a project using Vivado, a new folder will be created at the location you have chosen.

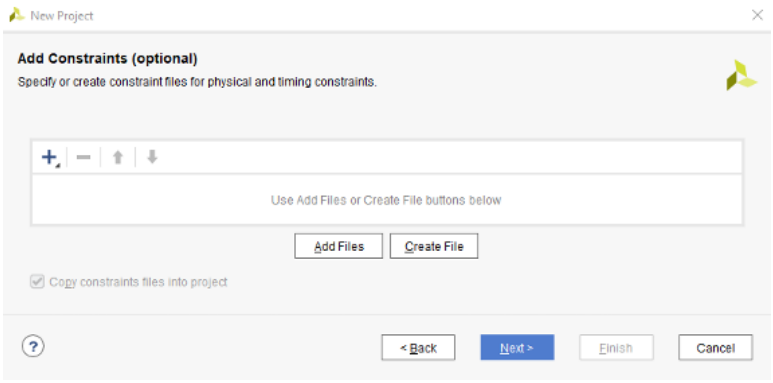
If all VHDL code is already made in QuestaSim (or ModelSim) then just click on Add Files to add source files into Vivado project and then select the file and click <Ok>.



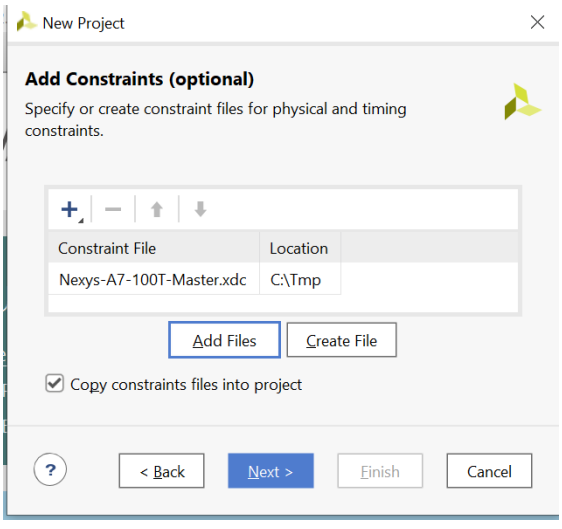
Autogenerate new VHDL file by click on Create File in **Add Source** window.
Don't forget to choose VHDL as file type and click <OK>.
This file will be configurated later on.



Back in the **Add Source** window for project setting.
When all necessarily VHDL-file are added or created into the project then click on Next > to add the constraints file (How to add more VHDL file into the project will be described in end of this section).



A constraints file (.xdc) containing mapping information between the top file's entity and the FPGA's actually pins, i.e. each type of board has its own constraints file.
Therefor it's very important to have the right constraints file and you must verify that the file is for our board. In the course we are going to use the board Nexys4DDR or newer version Nexys A7-100T.

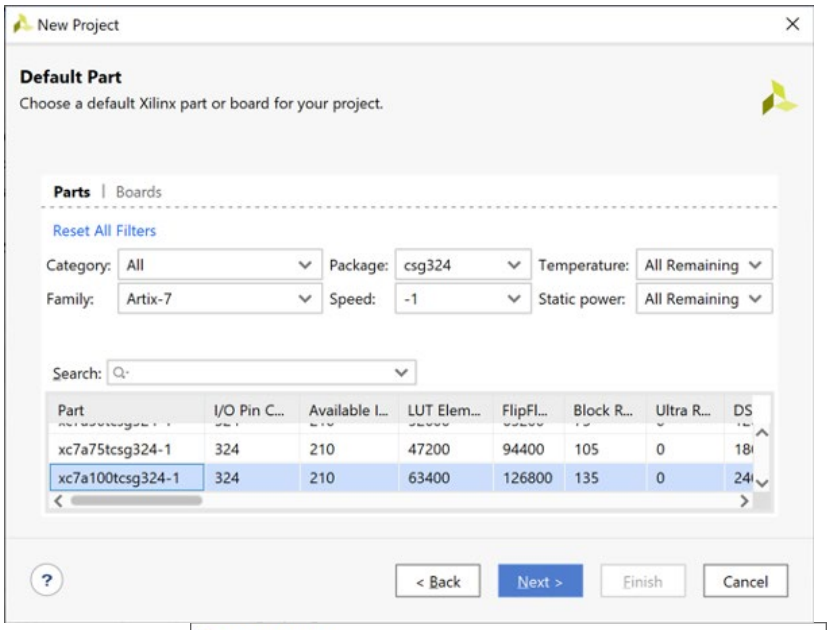


Click on Next > to open the chip selection, called **Default Part**.

Select family: Artix-7,
package: csg324,
Speed: -1 and
chose **xc7a100tsg324-1.**

Click on Next > and

then Finish to create the project.



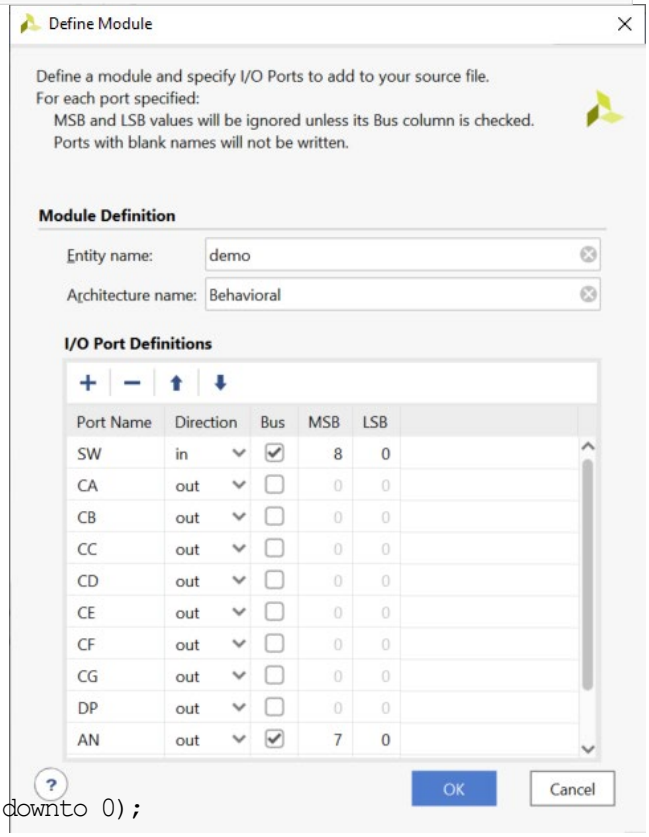
When autogenerated VHDL file was created during **Add source** then a **Define Module** will appear to configurated the files entity, see below.

Add the ports:

SW [8:0] as input

CA as output
CB as output
CC as output
CD as output
CE as output
CF as output
CG as output
DP as output

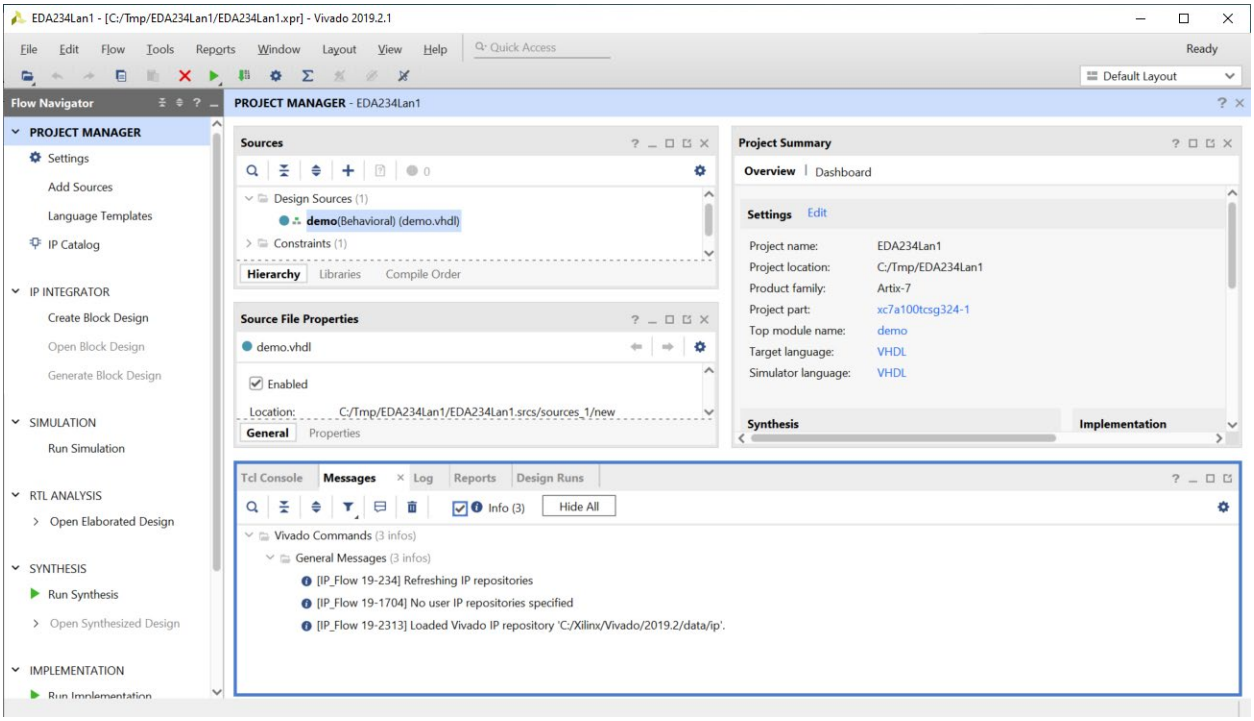
AN [7:0] as output



This module will generate the named VHDL-file with entity from the selection off **I/O Port Definition**, see the result below. The entity can be changed afterwards in the code editor.

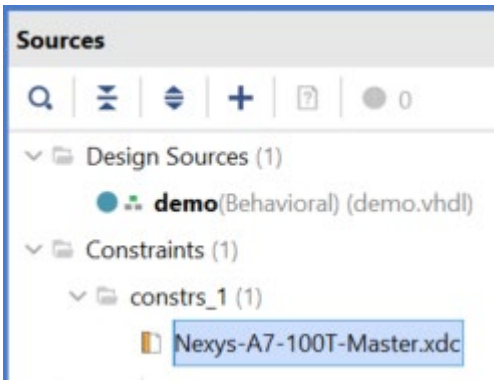
```
entity demo is
    Port ( SW : in STD_LOGIC_VECTOR (8 downto 0);
           CA : out STD_LOGIC;
           CB : out STD_LOGIC;
           CC : out STD_LOGIC;
           CD : out STD_LOGIC;
           CE : out STD_LOGIC;
           CF : out STD_LOGIC;
           CG : out STD_LOGIC;
           DP : out STD_LOGIC;
           AN : out STD_LOGIC_VECTOR (7 downto 0));
end demo;
```

Now when everything is configurated it's time to add some simple code.



Click on the demo file and add in the file:

```
begin

  AN <= SW(0) & "0000011"; -- This
   CA <= SW(1); --              is
   CB <= SW(2); --              the
   CC <= SW(3); --              code
   CD <= SW(4); --              you
   CE  <= SW(5); --             should
   CF  <= SW(6); --             add
   CG  <= SW(7); --
   DP <= SW(8);  --             !

end Behavioral;
```



Now we have the code for testing the 7-seg display om the lab system.
To get it working we need to assign the right pins in the constraint file.

Add the "Constraints" file:
File --> Add Sourcse --> [Add or create constraints], Next--> Add Files --> [Nexys-A7-100T-Master.xdc], Next --> Finish

Click on "Constraints" and open "Nexys-A7-100T-Master.xdc"

Found each of the signals you use, uncommit them, remove "#" in the beginning of the line.

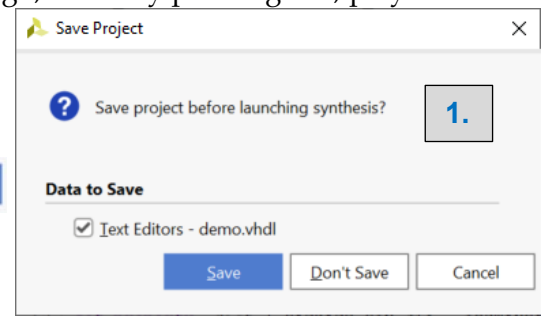
```
##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { SW[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
```

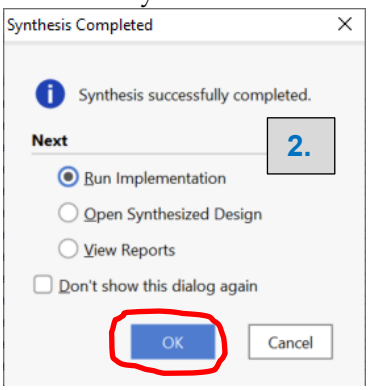
```
##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { SW[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]

##7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { CA }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { CB }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { CC }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { CD }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { CE }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { CF }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { CG }]; #IO_L4P_T0_D04_14 Sch=cg
set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { DP }]; #IO_L19N_T3_A21_VREF_15 Sch=dp
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { AN[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { AN[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { AN[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { AN[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { AN[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { AN[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { AN[6] }]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { AN[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
```

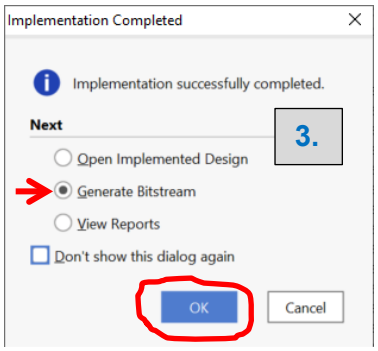
Now synthesis and run the design, either by pressing F11, play button  . or Run Synthesis ▶ Run Synthesis , see above.

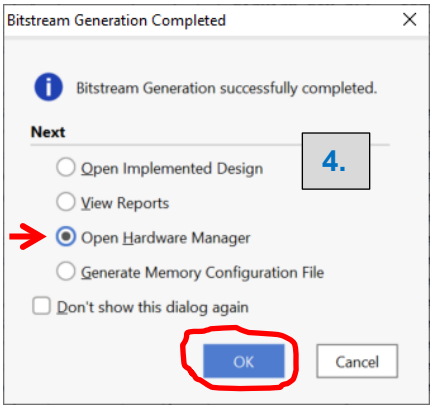
1. Save the project files:

2. Run Implementation.
3. Generate Bitstream.
4. Open Hardware Manager
5. **Connect the development board** to the commuter's USB-port.
   Use the "Program Uart" on the lab-system.
   Power on the lab-system.

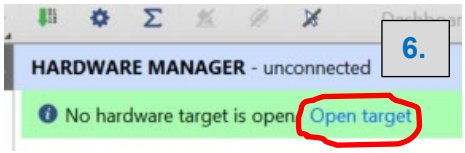






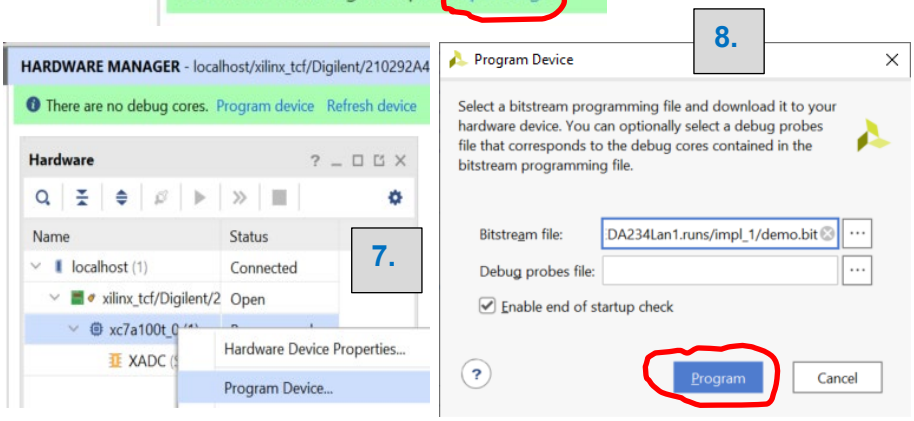


6. Open target → Auto Connect.



7. Program Device.

8. Program



Try to answer the following questions:

1. What polarity do the switch have?
2. Do an "1" ore an "0" activate the segment display?
3. Do an "1" ore an "0" activate an LED in the display?

Task 1

Modify the constraint and VHDL-file so that the signals CA, CB, CC, CD, CE, CF, CG and DP are on the bus
Seg : out std_logic_vector (7 downto 0);

CA bit 7 and DP bit 0.

Task 2

Use SW 3-0 to select the number and SW 6-4 to select display.

```
Hex     Binary  Segment Code

  0     "0000"  "11000000"
  1     "0001"  "11111001"
  2     "0010"  "10100100"
  3     "0011"  "10110000"
  4     "0100"  "10011001"
  5     "0101"  "10010010"
  6     "0110"  "10000010"
  7     "0111"  "11111000"
  8     "1000"  "10000000"
  9     "1001"  "10010000"
  A     "1010"  "10001000"
  B     "1011"  "10000011"
  C     "1100"  "11000110"
  D     "1101"  "10100001"
  E     "1110"  "10000110"
  F     "1111"  "10001110"
```

Task 3

Use SW 3-0 to select the number to display on the first segment display, display (SW 3-0)+1 on the second display and so on.

First hand in:

Make a second counter.

On the two first 7-segment displays show the seconds (0 to 59)

Make it in two steps, in QuestaSim/ModelSim implement the design.

Use shorter delays so that it becomes reasonable to simulate.

Write a Testbenches and verify the design.

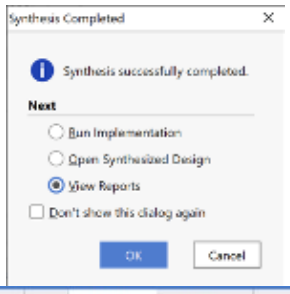
Add the correct delays and Synthesize the design to targeting a Xilinx Artix-7 FPGA platform.

Download to the FPGA-card and verify.

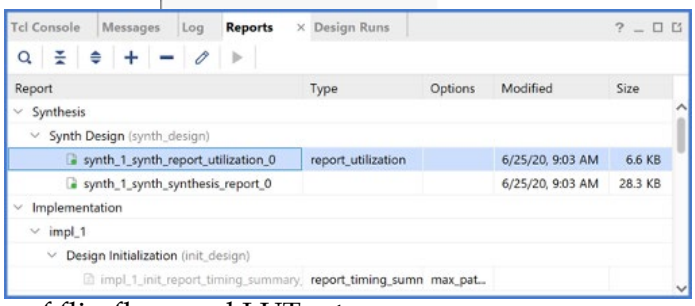
Submit: The VHDL-code in the first step and the Testbenches.

Utilization of the FPGA

We can study the utilization of the FPGA by select view reports, click <OK>

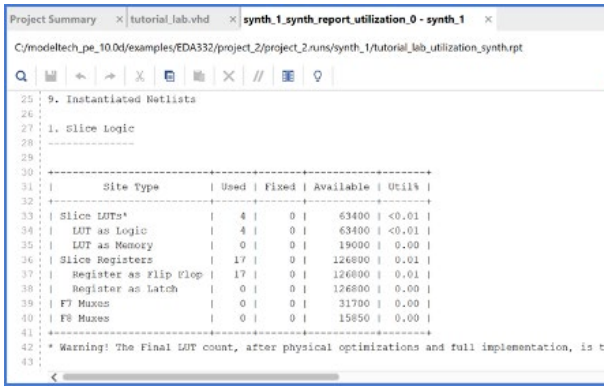


Select the tab **Reports** and then double-click on **synth_1_synth_report_utilization_0**

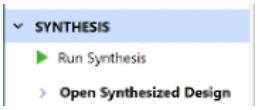


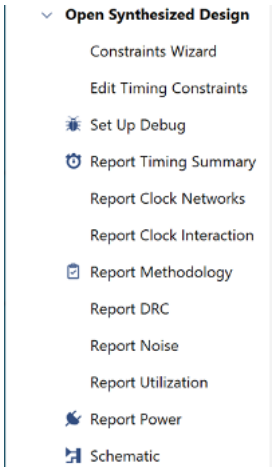
Now we can see reports about e.g. the number of flip-flops and LUTs etc.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	47	0	63400	0.07
LUT as Logic	47	0	63400	0.07
LUT as Memory	0	0	19000	0.00
Slice Registers	44	0	126800	0.03
Register as Flip Flop	44	0	126800	0.03
Register as Latch	0	0	126800	0.00
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00

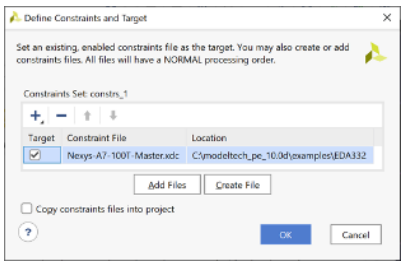


To more deeply investigate the design, we can open up the Synthesized design by first click on **Open Synthesized Design** on the left side of the main window.





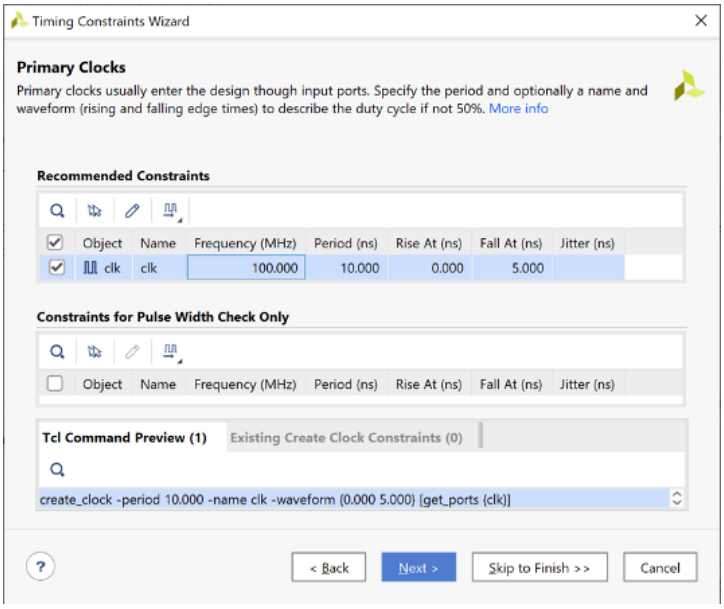
From the collection of functions, we can e.g. generate a timing report by first setting up timing constraints by click on the **Constants Wizard**. Since we only have pin mapping, i.e. no timing for this board we use that file, see below.



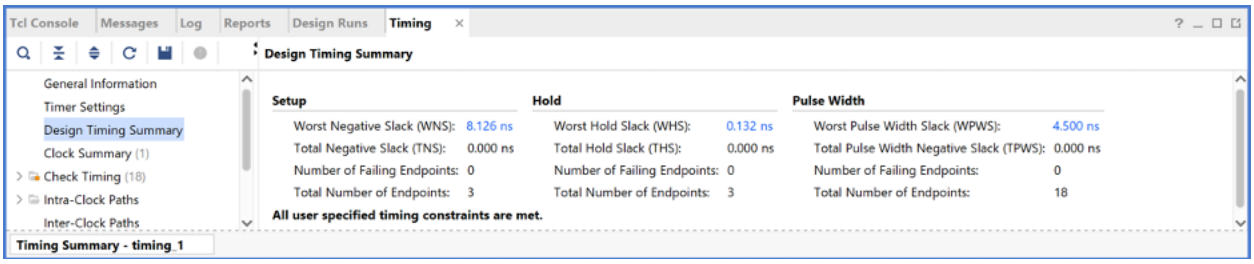
After click on <OK> the Constraints wizard will start with an information window.

Click on Next > to go to the timing-constraint window.
Since the only timing-constraint, we know, is the clock frequency, we will type in 100 as the frequency

and then click on Skip to Finish >>.



Now we can look into the timing by click on **Report Timing Summary** on the left side of the main window and in the **Options window** just click <Ok> then you can see **Design Timing Summary**.



Now we have both **synth_1_synth_report_utilization** and **Design Timing Summary** and other useful information. Try to understand the meaning and implications of various terms and numbers you see in these reports.

Answer the following questions and check your answers with an assistant**:**

1. How many flip-flops were used to implement the design?
2. How many LUTs (look-up tables) were used to implement the design?
3. What is the maximum net delay post place and route?

Second hand in:

You should be able to use:

```
|   Register as Flip Flop |   44 |
```

Or less.

How many Registers do you use?

If you use more then 50 "`Register as Flip Flop`", optimize the code to reduce the number of Registers so you have below 50.

Hand in the new code. (with correct delays).
Write a short reflection on what you have done.

Hints and Tips:

Installing ModelSim and Xilinx on your personal device

The student version of *ModelSim* [1]and *Vivado Design Suite Evaluation and WebPACK* [2] are sufficient for the this course. You can download and use these tools on your own device from the links provided at the end of this document.

Forcing values in the .do file

Depending on the version of the ModelSim you use, you can utilize different formats for assigning a value to a signal. But, it is strongly recommended that you use the generally accepted explicit format where # symbol should be used to define the radix of a value. For instance:

- force mySignal 2#101

where binary value "101" is assigned to mySignal, or

- force mySignal 10#240

where decimal value "240" is assigned to mysignal.

References

[1] ModelSim PE Student Edition. [Online]. Available:
http://www.mentor.com/company/higher_ed/modelsim-student-edition

[2] Vivado Design Suite Evaluation and WebPACK. [Online]. Available:
https://www.xilinx.com/products/design-tools/vivado/vivado-webpack.html