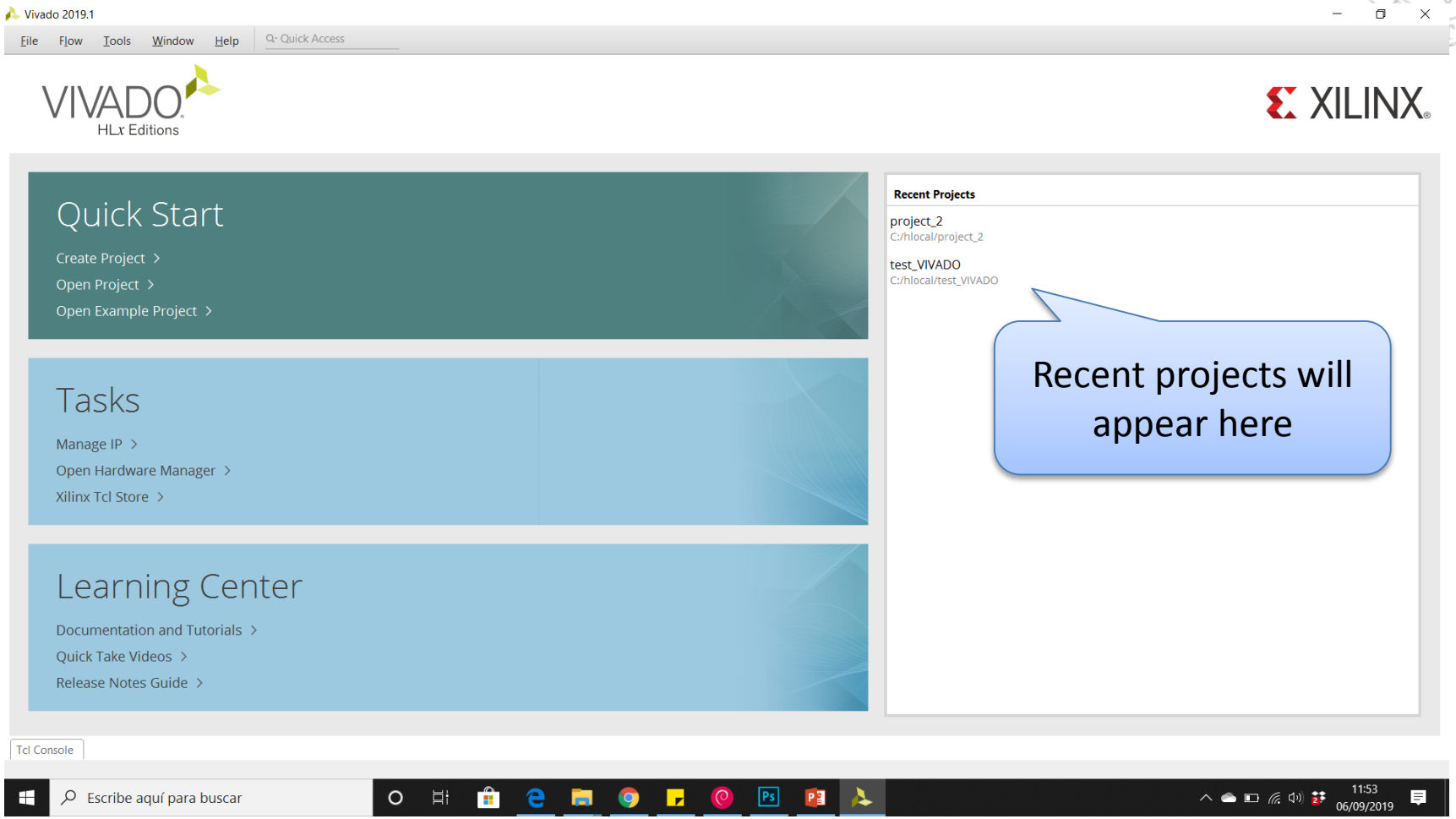
Lab 1

- How to use the Vivado tool to program with VHDL
- Design and simulation of simple circuits

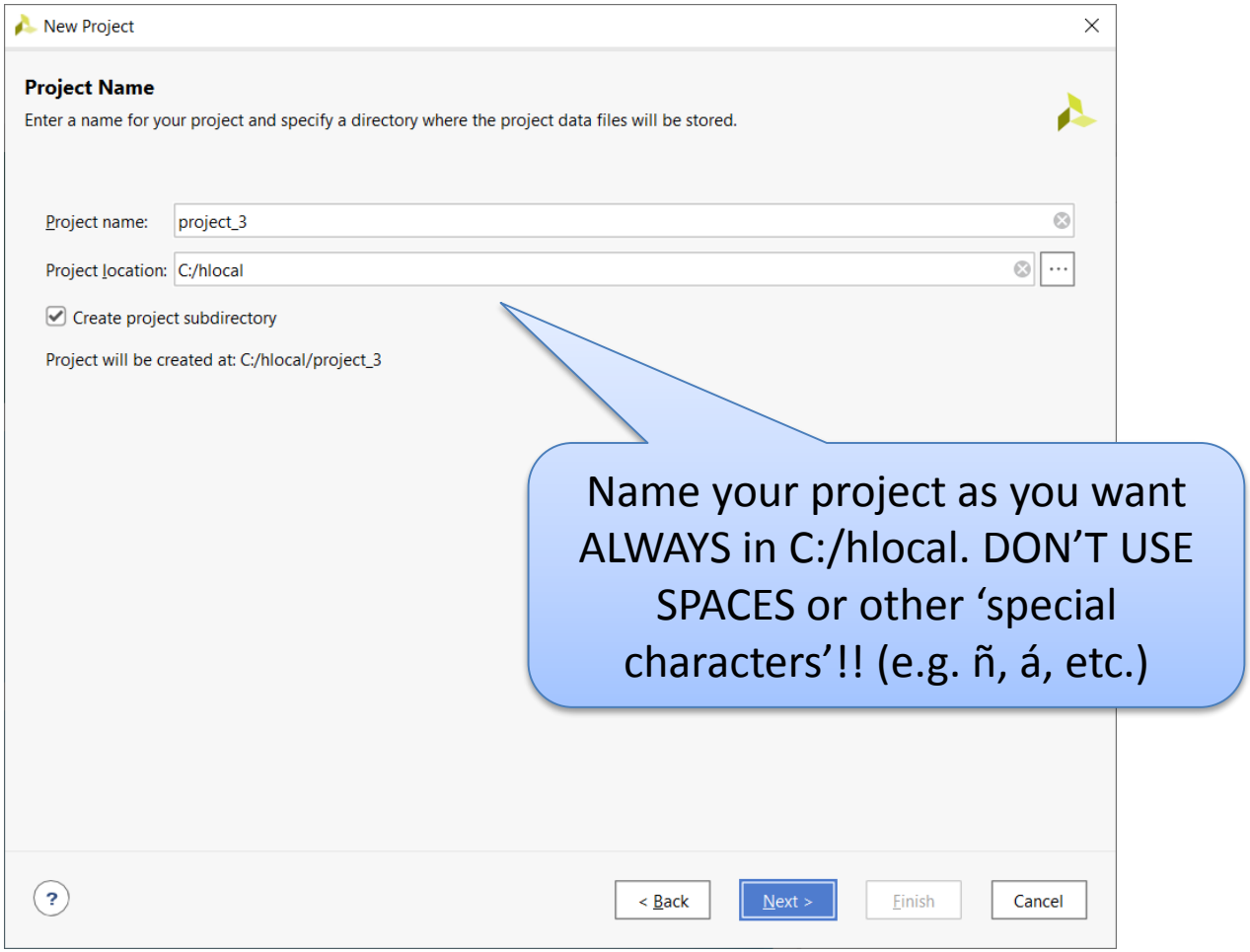
Prof. Juan Antonio Clemente

Welcome screen

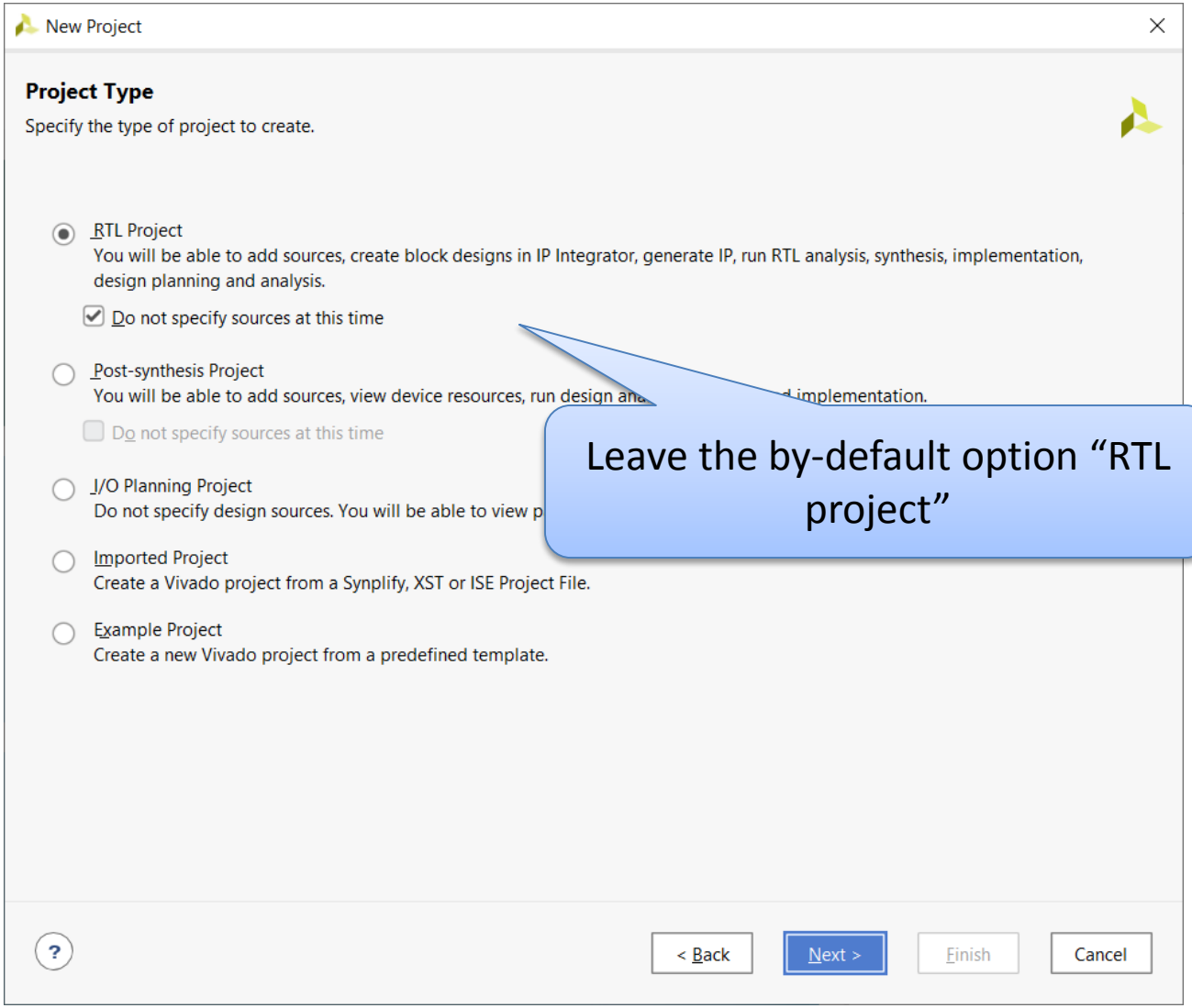


How to create a project

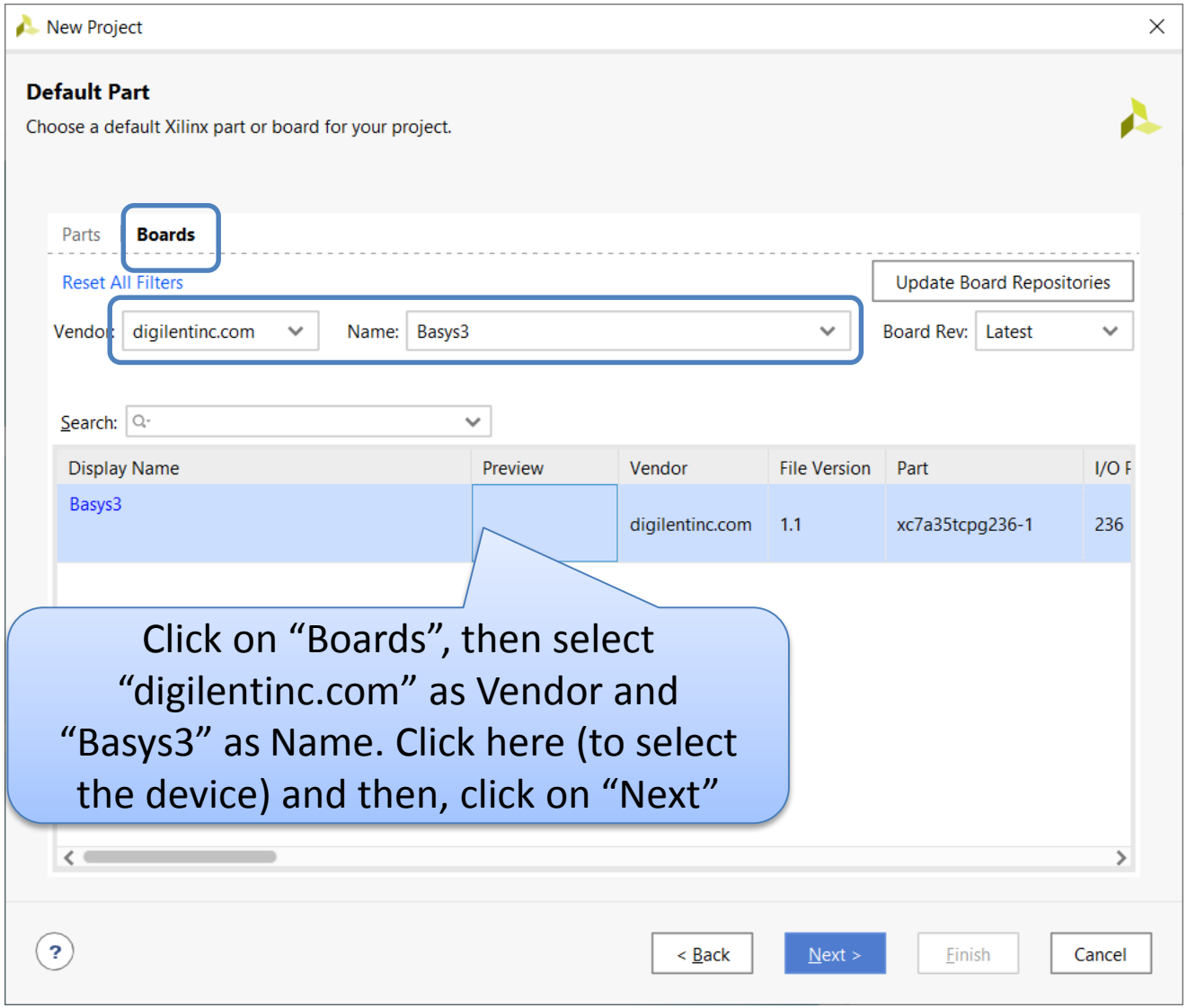
- Click on File->Project->New and then, Next. The following screen will appear



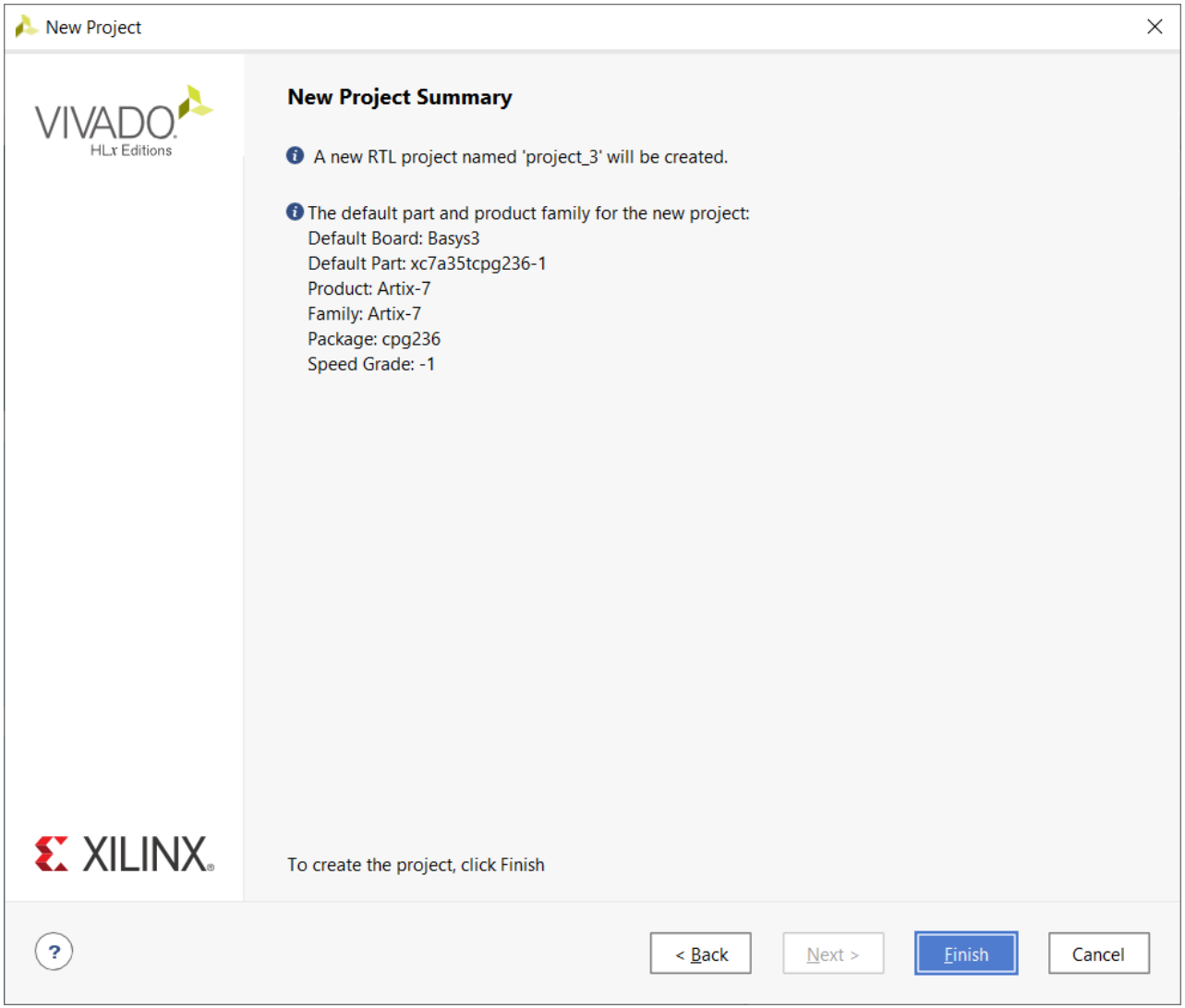
How to create a project



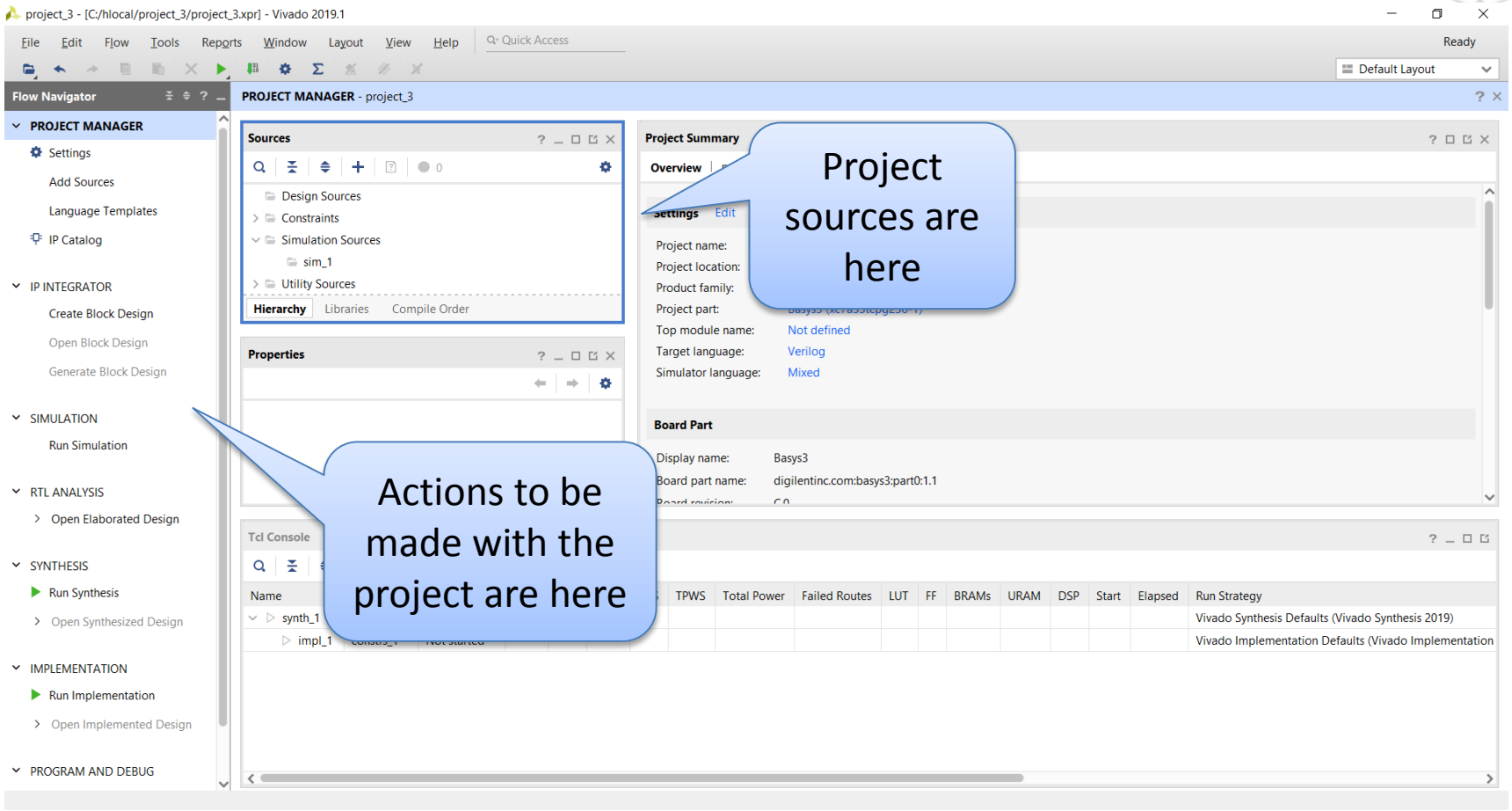
Select the target FPGA



Project summary

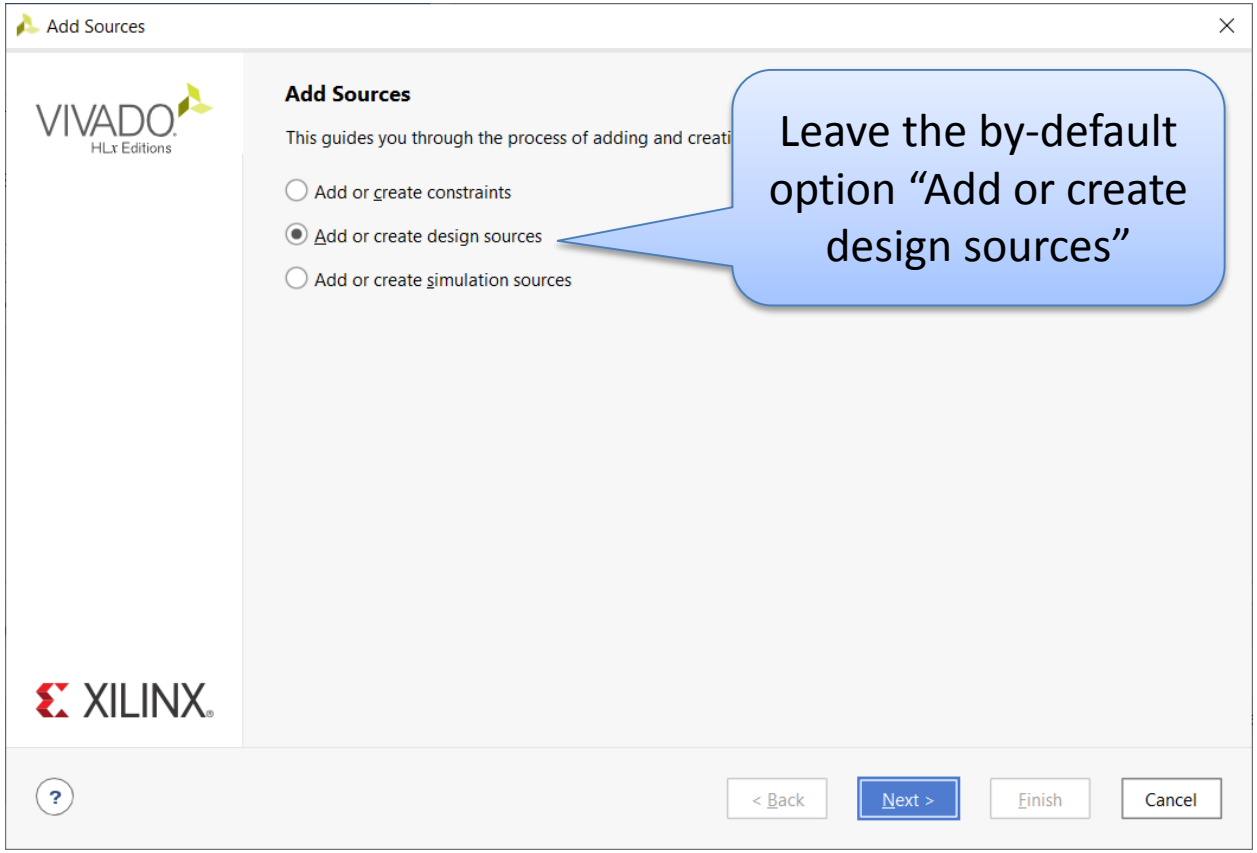


Voilà the main GUI of VIVADO

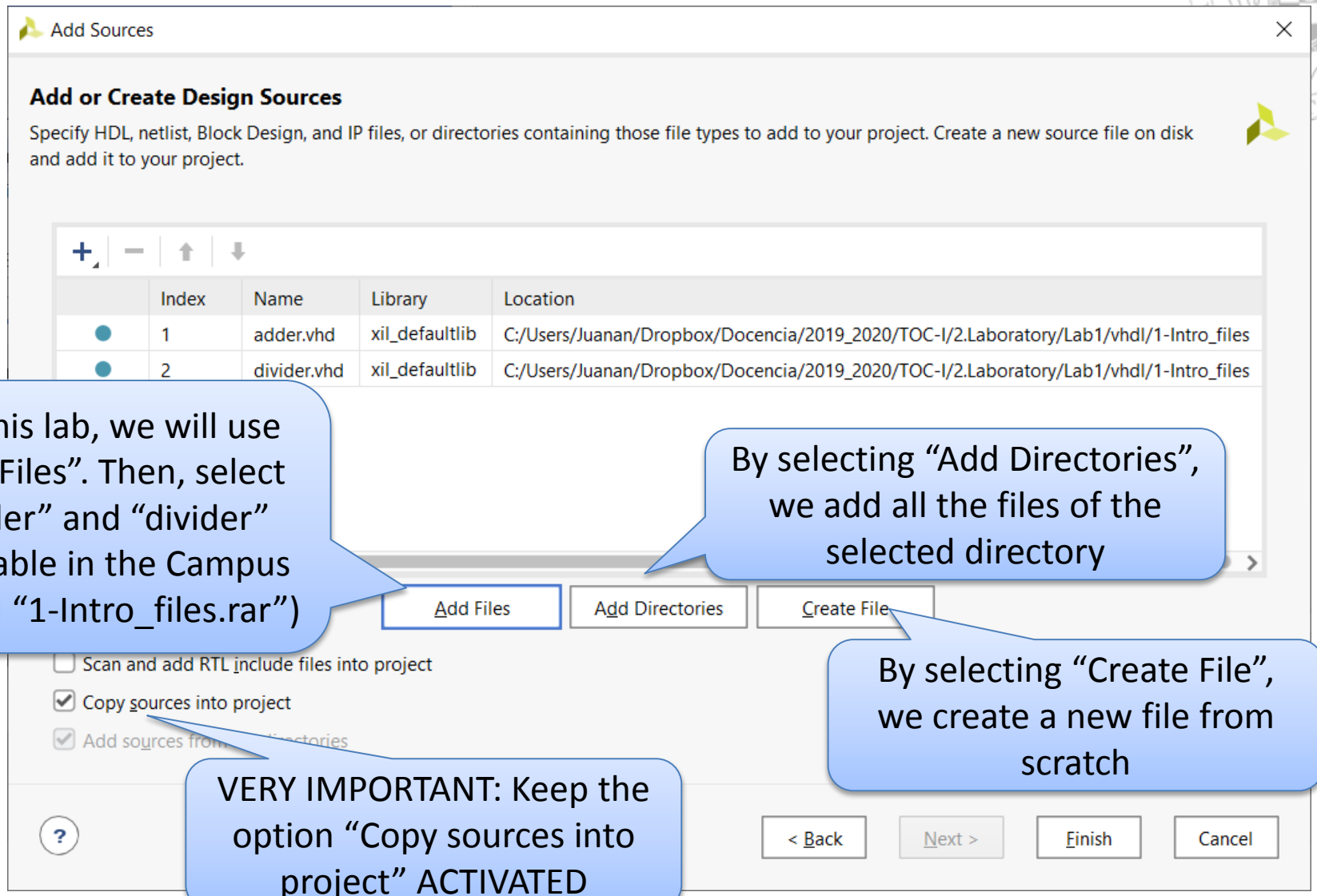


Add or create new source files

- Click on “Add source” (top left), then…



Add or create new source files



For this lab, we will use "Add Files". Then, select "adder" and "divider" (available in the Campus Virtual "1-Intro_files.rar")

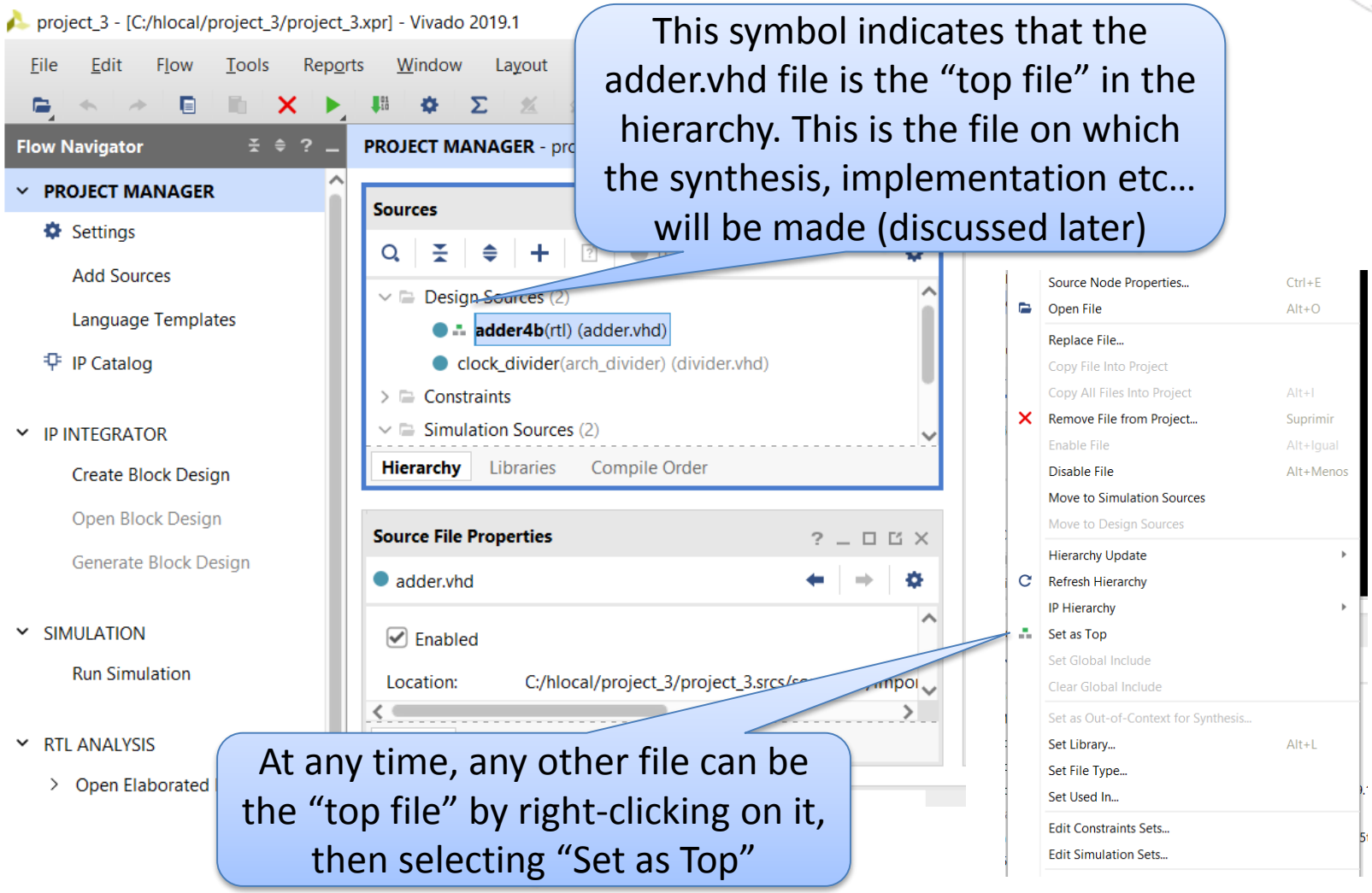
By selecting "Add Directories", we add all the files of the selected directory

By selecting "Create File", we create a new file from scratch

VERY IMPORTANT: Keep the option "Copy sources into project" ACTIVATED

Add existing source files to our project

- The added files will be now visible in the Sources part:

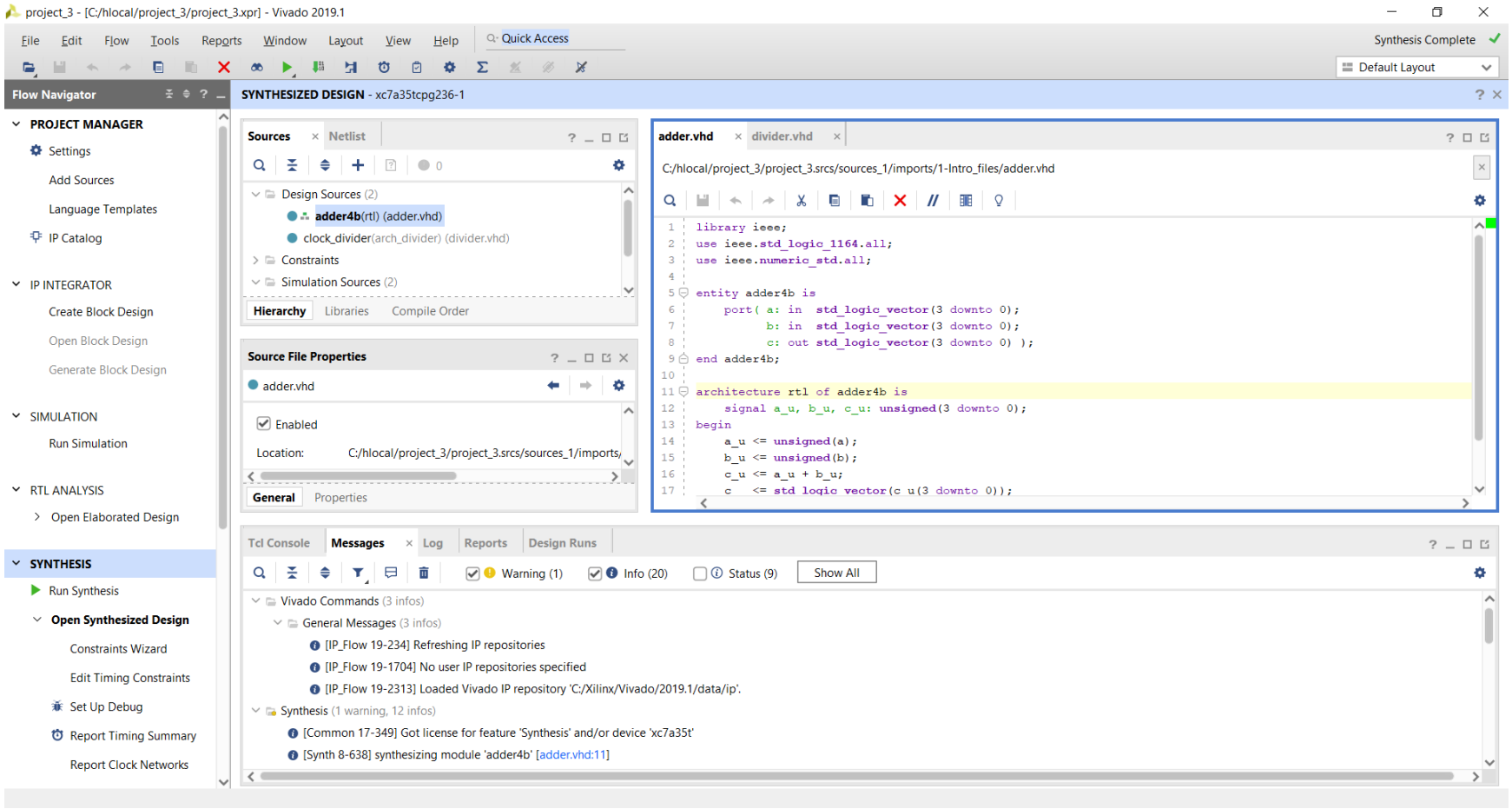


This symbol indicates that the adder.vhd file is the “top file” in the hierarchy. This is the file on which the synthesis, implementation etc… will be made (discussed later)

At any time, any other file can be the “top file” by right-clicking on it, then selecting “Set as Top”

Add existing source files to our project

- By double-clicking on these files, one can see and edit the source code

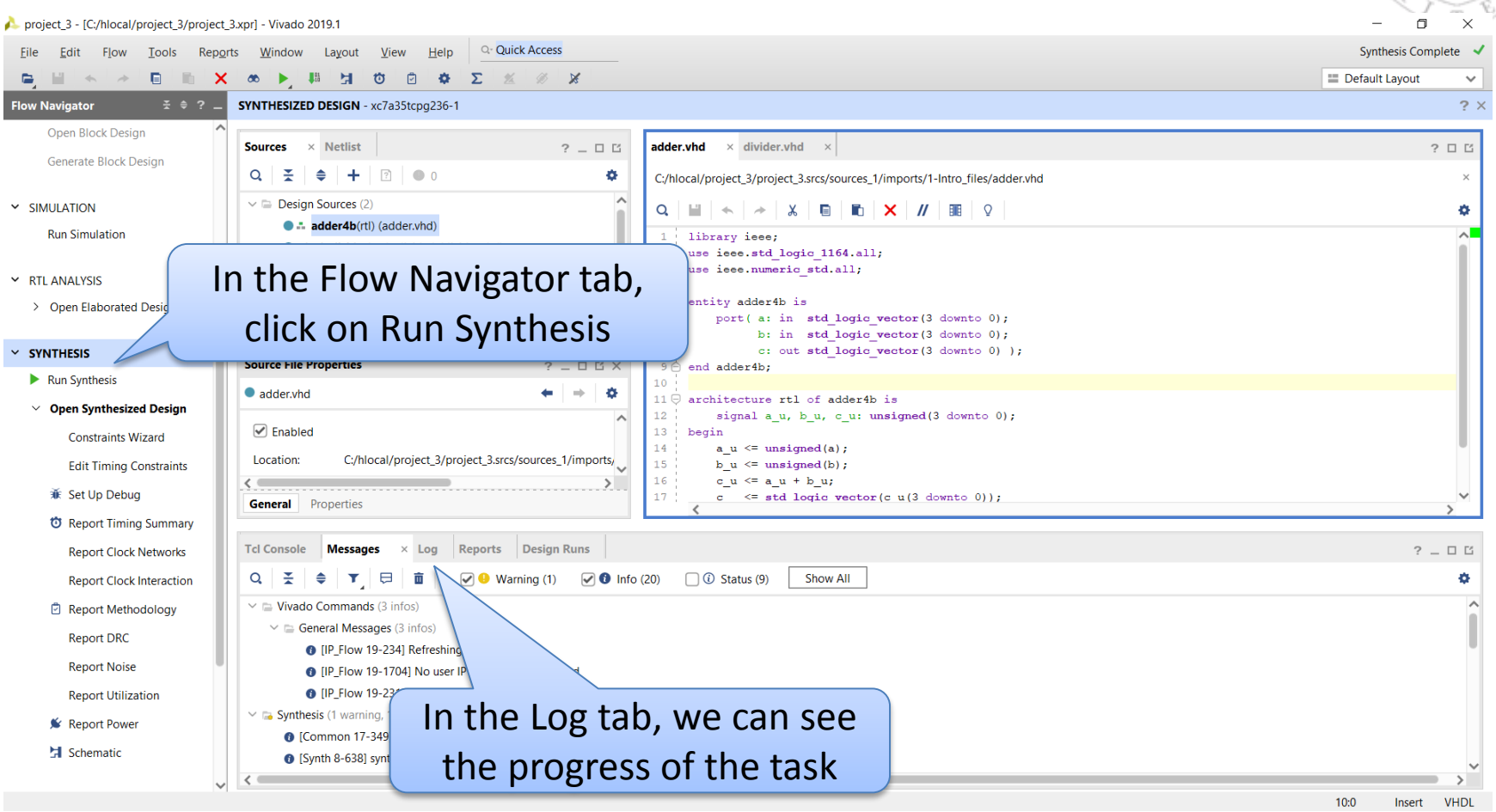


Lab 1.a

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

entity adder4b is
    port( a: in  std_logic_vector(3 downto 0);
          b: in  std_logic_vector(3 downto 0);
          c: out std_logic_vector(3 downto 0) );
end adder4b;

architecture rtl of adder4b is
    signal a_u, b_u, c_u: unsigned(3 downto 0);
begin
    a_u <= unsigned(a);
    b_u <= unsigned(b);
    c_u <= a_u + b_u;
    c   <= std_logic_vector(c_u(3 downto 0));
end rtl;
```

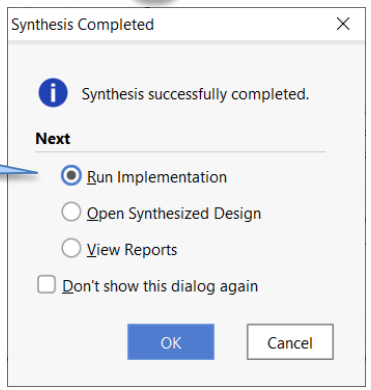
This is a 4-bit adder (we will discuss this in detail in class)

Synthesize the design

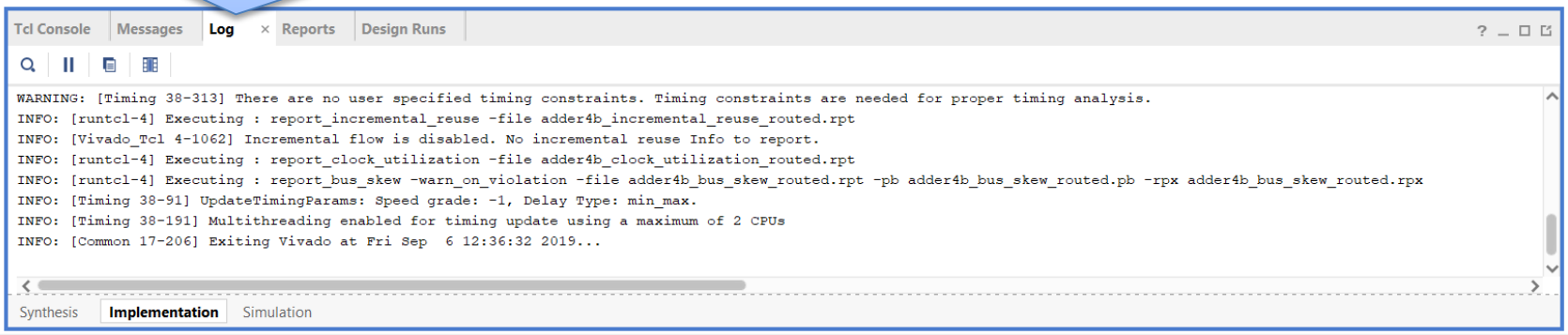


Synthesize the design

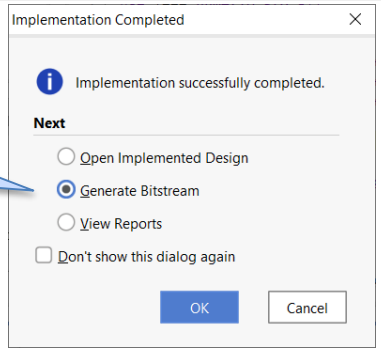
Once Synthesis is completed, the following window appears. We click on Run Implementation, then Next.



In the Log tab, you will see that there are separate messages for the Synthesis, Implementation and Simulation processes!

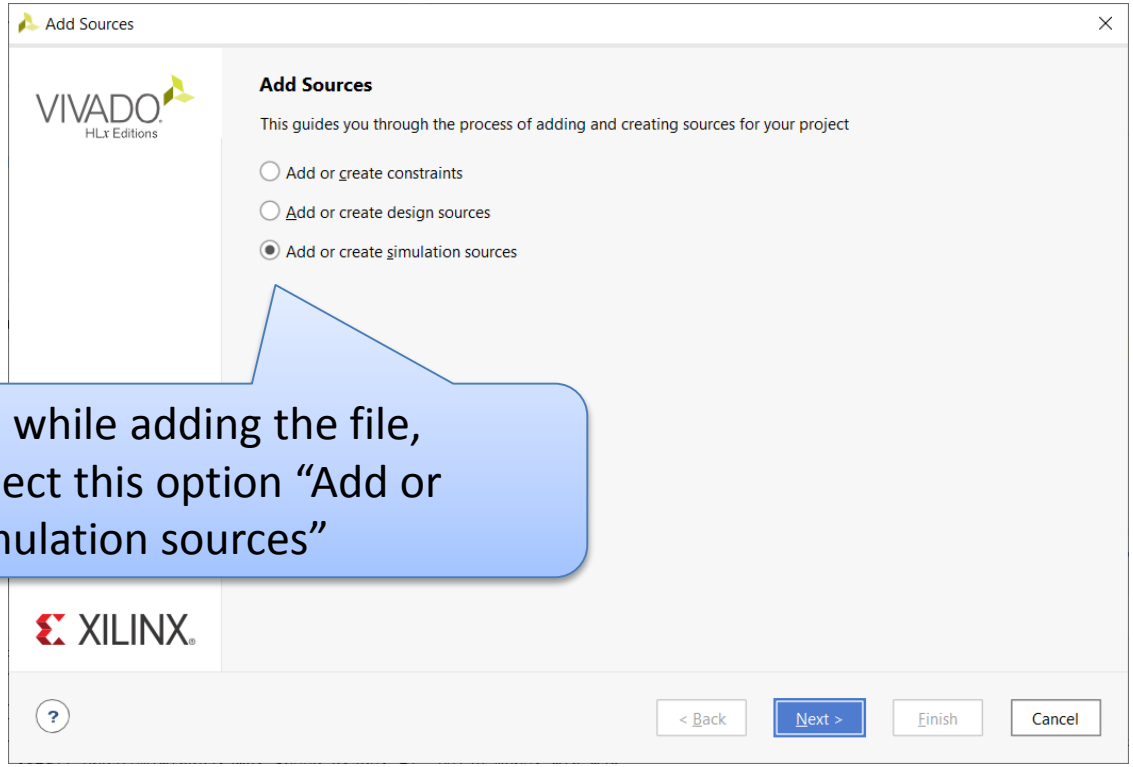


Once Implementation is completed, the following window appears. If we click on Generate Bitstream, IT WILL FAIL (a constraints file is needed). We just open implemented design



Check that the design is correct

- In order to check if a design is correct, you must simulate the circuit:
  - Add the testbench that is available in the Campus Virtual “simu_1a.vhd”.

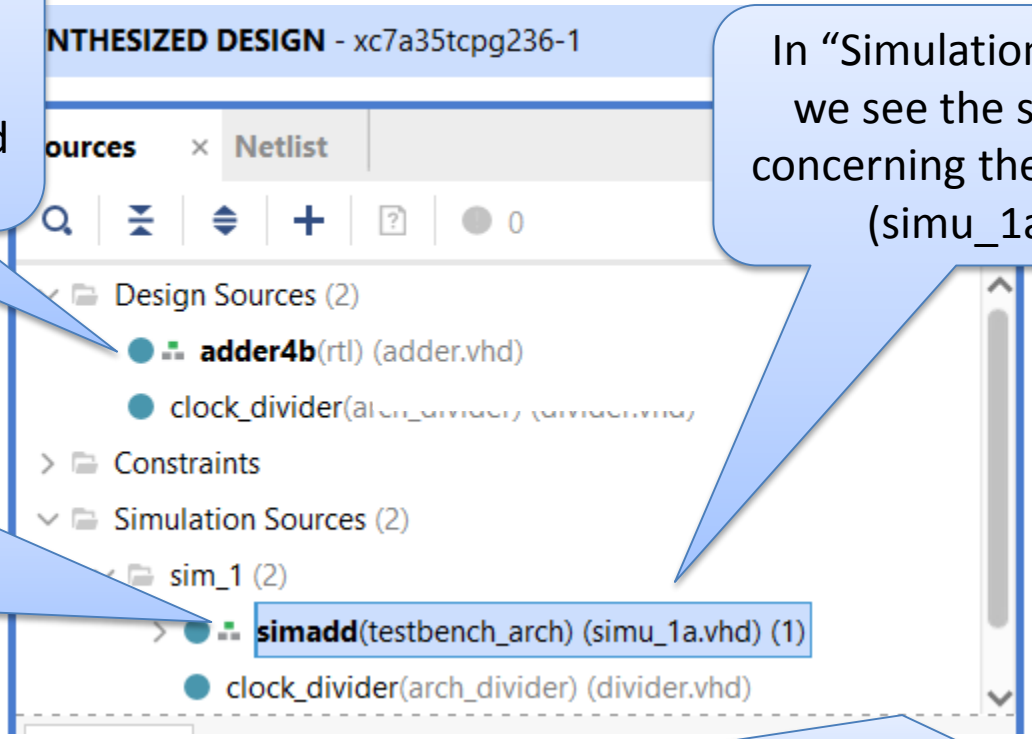


This time, while adding the file, please select this option “Add or créate simulation sources”

Simulation testbench

- Once added, we can see it in the main VIVADO GUI

In “Design Sources”, we see the source files concerning the design (adder.vhd and divider.vhd)



In “Simulation Sources”, we see the source file concerning the simulation (simu_1a.vhd)

This symbol also appears in one of the simulation files. The file with the symbol will be the one that is simulated

divider.vhd appears in “Design Sources” and “Simulation Sources” because it is still not part of the design hierarchy (it is not used from adder.vhd). **We will fix this later.**

Simulation testbench

- POSSIBLE PROBLEM: You have screwed it up and a simulation source has been included as a design source, or viceversa.
  - SOLUTION: At any time, you can right-click on the involved file and select “Move to Simulation Sources” or “Move to Design Sources” to fix this problem.

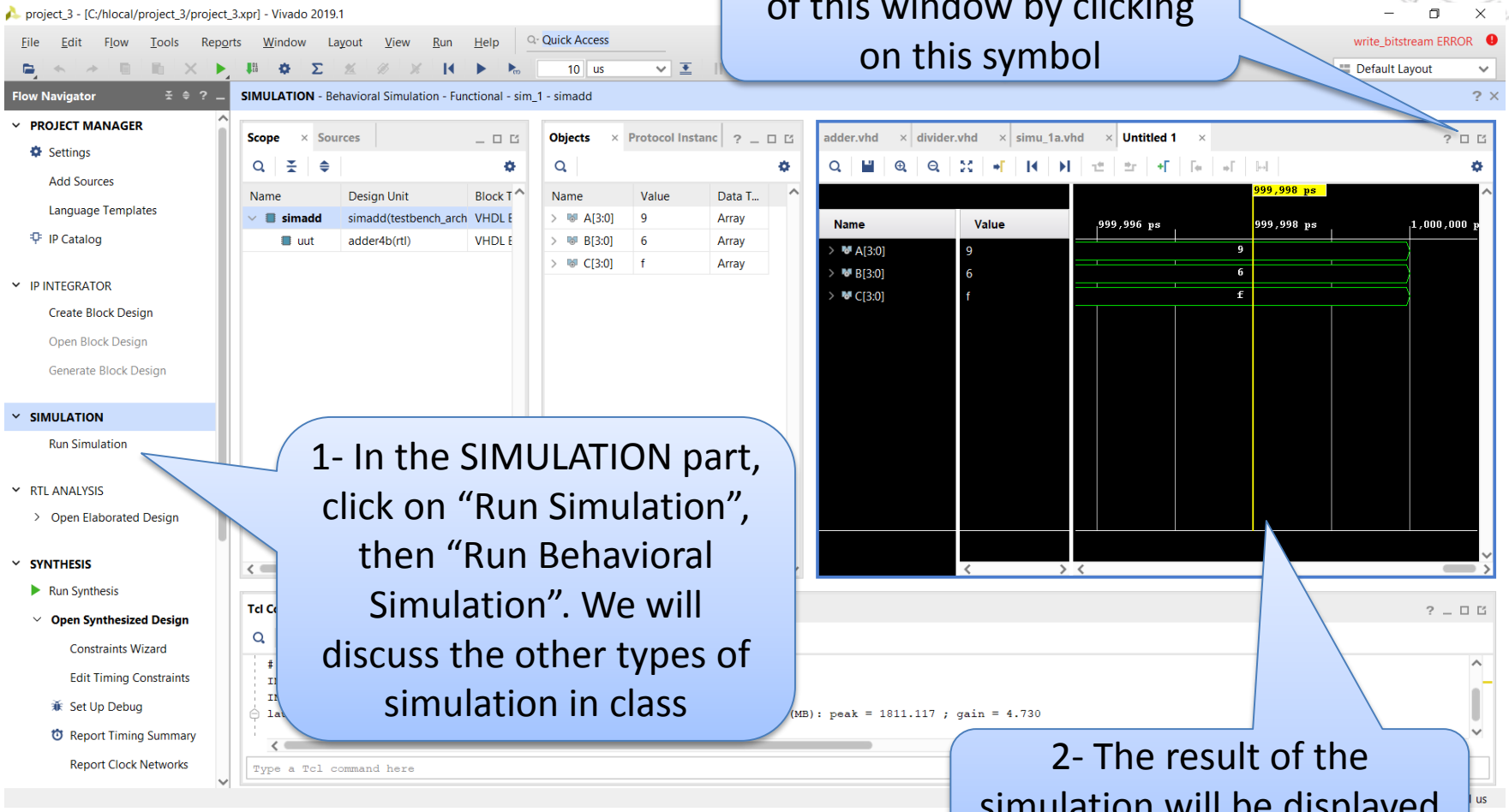
Simulation testbench (check the Campus Virtual)

```
-- We add the libraries needed
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Entity declaration
ENTITY simadd IS
END simadd;

-- Architecture
ARCHITECTURE testbench_arch OF simadd IS
    -- Component declaration
    COMPONENT adder4b
    PORT(
         A : IN  std_logic_vector(3 downto 0);
         B : IN  std_logic_vector(3 downto 0);
         C : OUT std_logic_vector(3 downto 0)
        );
    END COMPONENT;

   -- Inputs
   signal A : std_logic_vector(3 downto 0) := (others => '0');
   signal B : std_logic_vector(3 downto 0) := (others => '0');

   -- Outputs
   signal C : std_logic_vector(3 downto 0);
...
```

Simulation testbench (check the Campus Virtual)

```
BEGIN
  -- Instantiation of the unit under test
   uut: adder4b PORT MAP (
          A => A,
          B => B,
          C => C );
  -- Stimuli process
stim_proc: process
    begin
        A <= "0000";        B <= "0000";
        wait for 100 ns;
        A <= "0101";        B <= "0100";
        wait for 100 ns;
        A <= "0000";        B <= "0111";
        wait for 100 ns;
        A <= "0011";        B <= "1000";
        wait for 100 ns;
        A <= "1011";        B <= "1111";
        wait for 100 ns;
        A <= "1001";        B <= "0110";
        wait;
end process;

END testbench_arch;
```

Simulation



3- We can expand the size of this window by clicking on this symbol

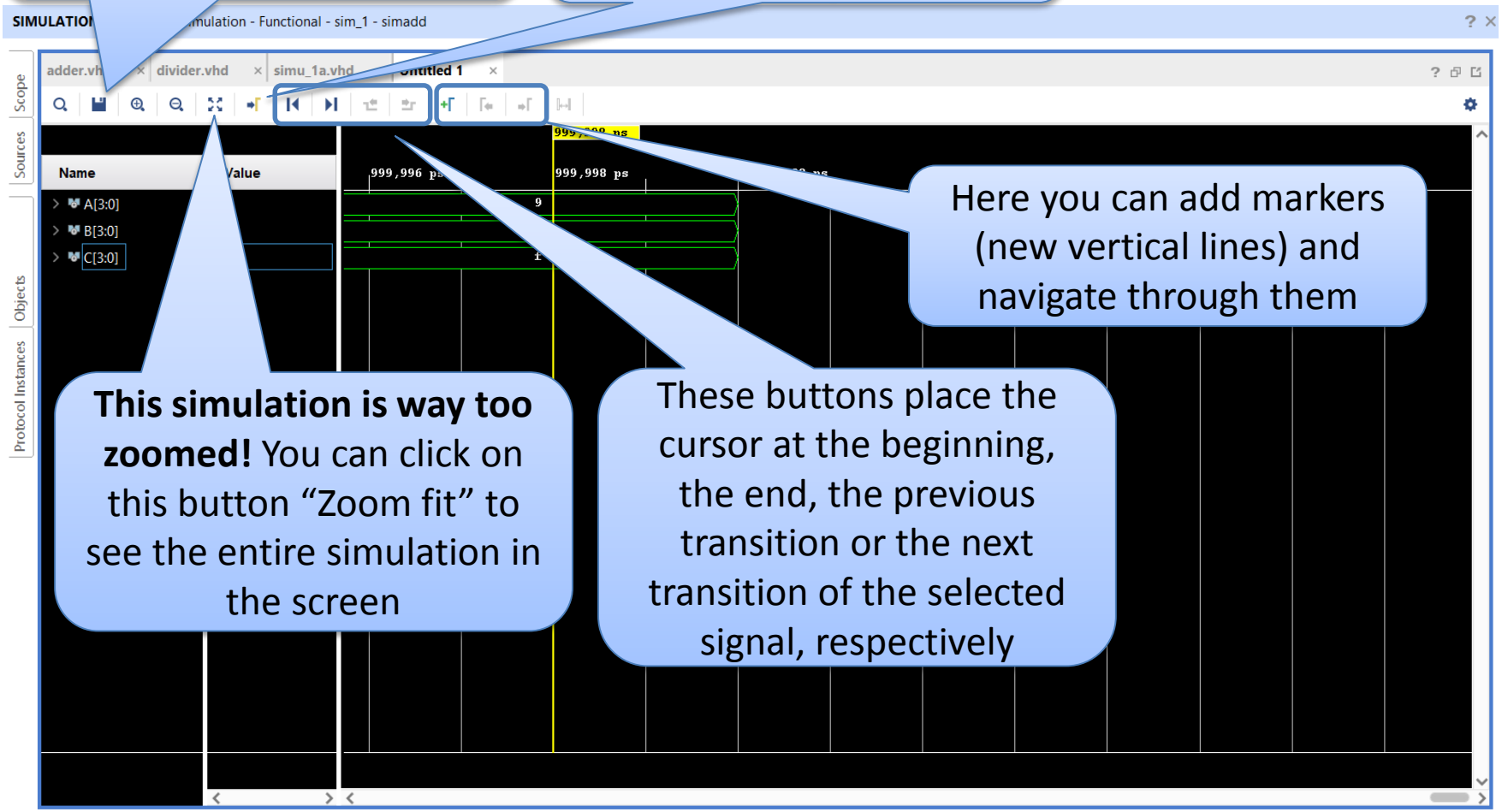
1- In the SIMULATION part, click on “Run Simulation”, then “Run Behavioral Simulation”. We will discuss the other types of simulation in class

2- The result of the simulation will be displayed in this window

Simulation window

The waveform can be saved as a .wcfg file

This centers the view on the "cursor" (yellow vertical line)

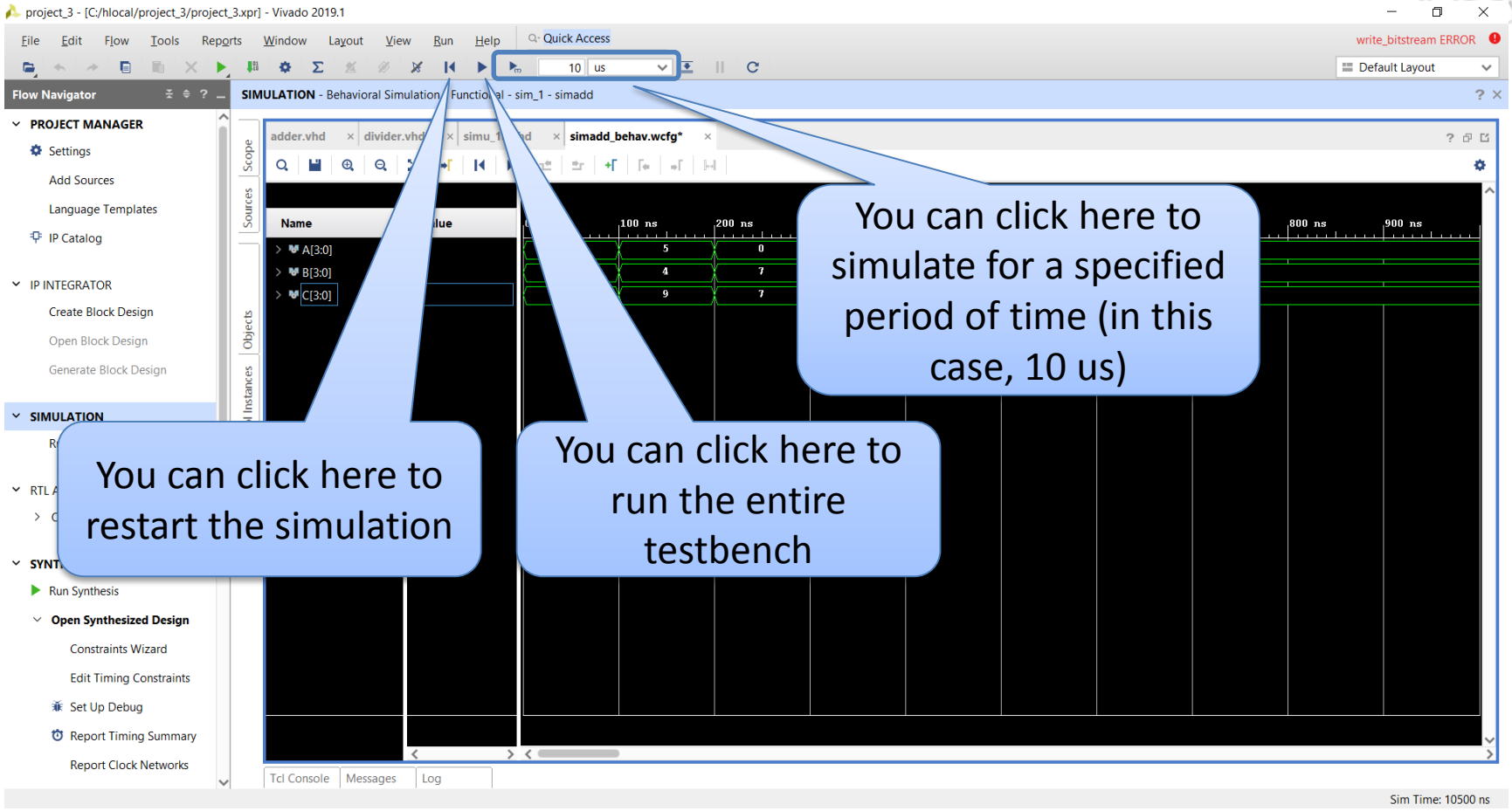


Here you can add markers (new vertical lines) and navigate through them

This simulation is way too zoomed! You can click on this button "Zoom fit" to see the entire simulation in the screen

These buttons place the cursor at the beginning, the end, the previous transition or the next transition of the selected signal, respectively

Simulation

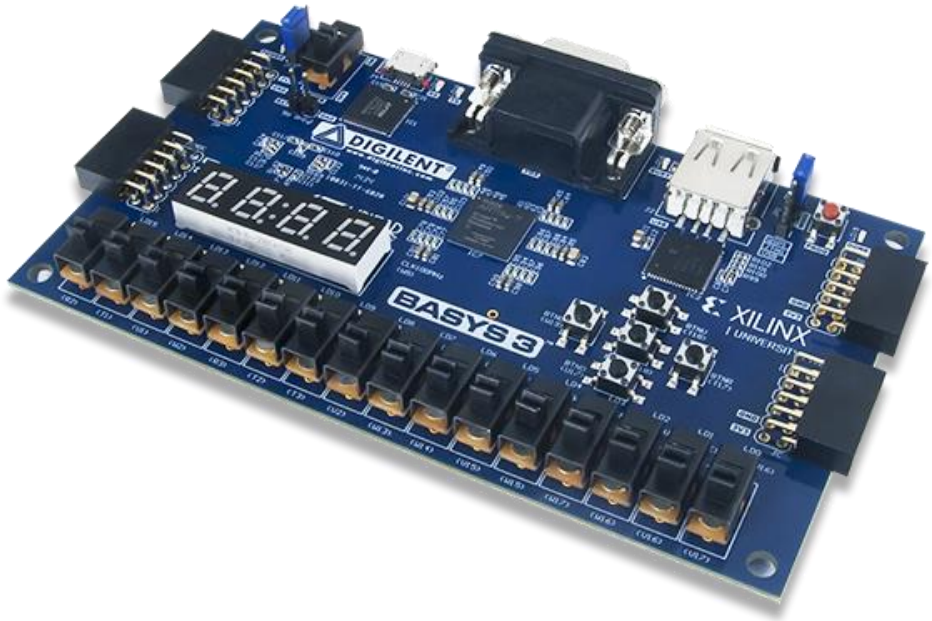

project_3 - [C:/hlocal/project_3/project_3.xpr] - Vivado 2019.1
File Edit Flow Tools Reports Window Layout View Run Help
Quick Access
write_bitstream ERROR
10 us
Default Layout
Flow Navigator
SIMULATION - Behavioral Simulation - Functional - sim_1 - simadd
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
RTL A
SYNT
Run Synthesis
Open Synthesized Design
Constraints Wizard
Edit Timing Constraints
Set Up Debug
Report Timing Summary
Report Clock Networks
adder.vhd
divider.vhd
simadd_behav.wcfg*
Scope
Sources
Objects
Instances
Name
A[3:0]
B[3:0]
C[3:0]
100 ns
200 ns
800 ns
900 ns
Tcl Console
Messages
Log
Sim Time: 10500 ns
You can click here to restart the simulation
You can click here to run the entire testbench
You can click here to simulate for a specified period of time (in this case, 10 us)

Implementation

- We can check that the circuit work the *real world*
  - The circuit will be implemented on an FPGA
  - The implementation tool needs to know where the inputs come from (push buttons or switches) and where to display the outputs (LEDs or 7-segment displays).
    - A constraints file (with extension .xdc) must be added with all this information.

Implementation

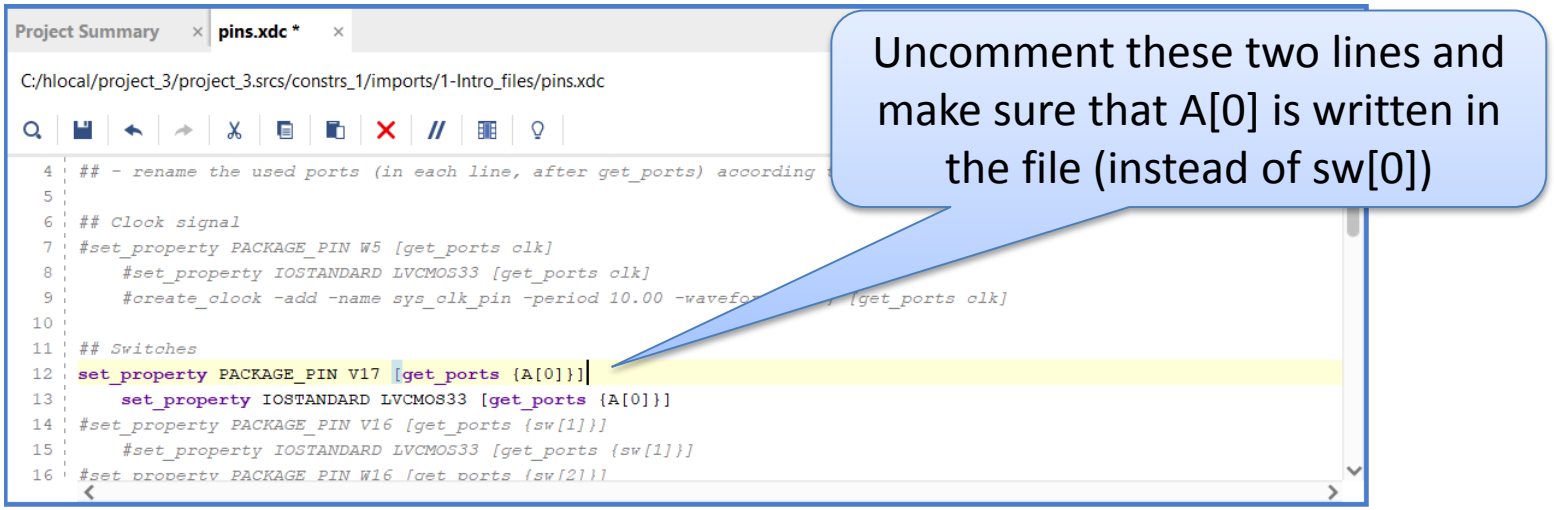
- This is the FPGA that we will use in all the labs:



https://reference.digilentinc.com/reference/programmable-logic/basys-3/start

Implementation

- .xdc file
  - You can find a complete .xdc file in the Campus Virtual of this particular board (downloaded from https://reference.digilentinc.com/reference/programmable-logic/basys-3/start → Master XDC files).
- This is a general file that allows associating your inputs and outputs with the physical IO devices of the board (LEDs, buttons, etc…).
- All the code in it is written on comments. You can uncomment the desired lines to use the IO devices that you need.
  - Each IO device of the board is associated with 2 lines starting with *set_property*.
- For instance, to use a switch for input pin A[0] (*adder.vhd*), it would be as follows:



Implementation

A and B (2 4-bit inputs) are associated to 8 switches like this

C (a 4-bit input) is associated to 4 LEDs like this

Project Summary | pins.xdc *

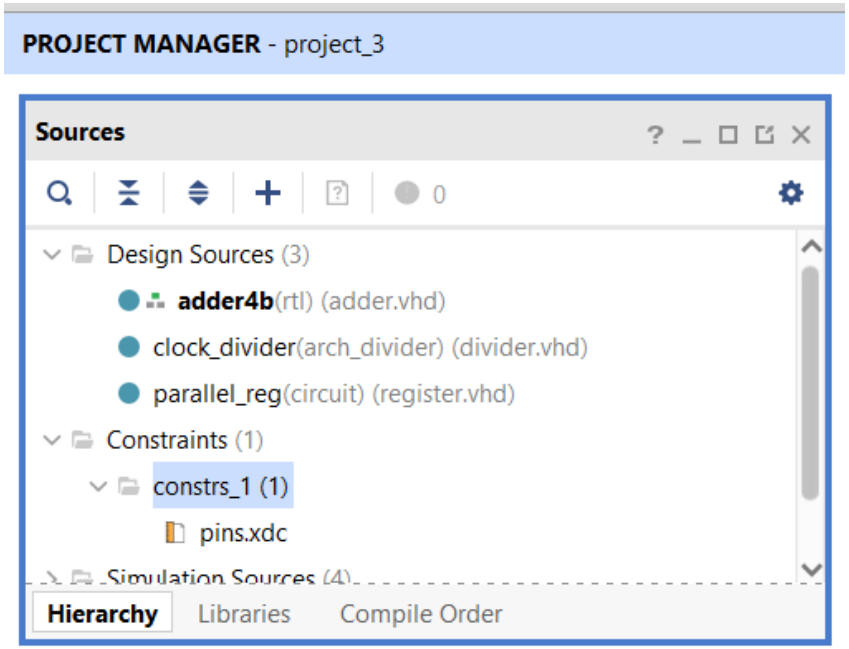
C:/hlocal/project_3/project_3.srcs/constrs_1/imports/1-Intro_files/pins.xdc

```
7  #set_property PACKAGE_PIN W5 [get_ports clk]
8      #set_property IOSTANDARD LVCMOS33 [get_ports clk]
9      #create_clock -add -name sys_clk_pin -period 10.00 -wav
10
11 ## Switches
12 set_property PACKAGE_PIN V17 [get_ports {A[0]}]
13     set_property IOSTANDARD LVCMOS33 [get_ports {A[0]}]
14 set_property PACKAGE_PIN V16 [get_ports {A[1]}]
15     set_property IOSTANDARD LVCMOS33 [get_ports {A[1]}]
16 set_property PACKAGE_PIN W16 [get_ports {A[2]}]
17     set_property IOSTANDARD LVCMOS33 [get_ports {A[2]}]
18 set_property PACKAGE_PIN W17 [get_ports {A[3]}]
19     set_property IOSTANDARD LVCMOS33 [get_ports {A[3]}]
20 set_property PACKAGE_PIN W15 [get_ports {B[4]}]
21     set_property IOSTANDARD LVCMOS33 [get_ports {B[4]}]
22 set_property PACKAGE_PIN V15 [get_ports {B[5]}]
23     set_property IOSTANDARD LVCMOS33 [get_ports {B[5]}]
24 set_property PACKAGE_PIN W14 [get_ports {B[6]}]
25     set_property IOSTANDARD LVCMOS33 [get_ports {B[6]}]
26 set_property PACKAGE_PIN W13 [get_ports {B[7]}]
27     set_property IOSTANDARD LVCMOS33 [get_ports {B[7]}]
28 #set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
29     #set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
30 #set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
```

```
## LEDs
set_property PACKAGE_PIN U16 [get_ports {C[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {C[0]}]
set_property PACKAGE_PIN E19 [get_ports {C[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {C[1]}]
set_property PACKAGE_PIN U19 [get_ports {C[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {C[2]}]
set_property PACKAGE_PIN V19 [get_ports {C[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {C[3]}]
```

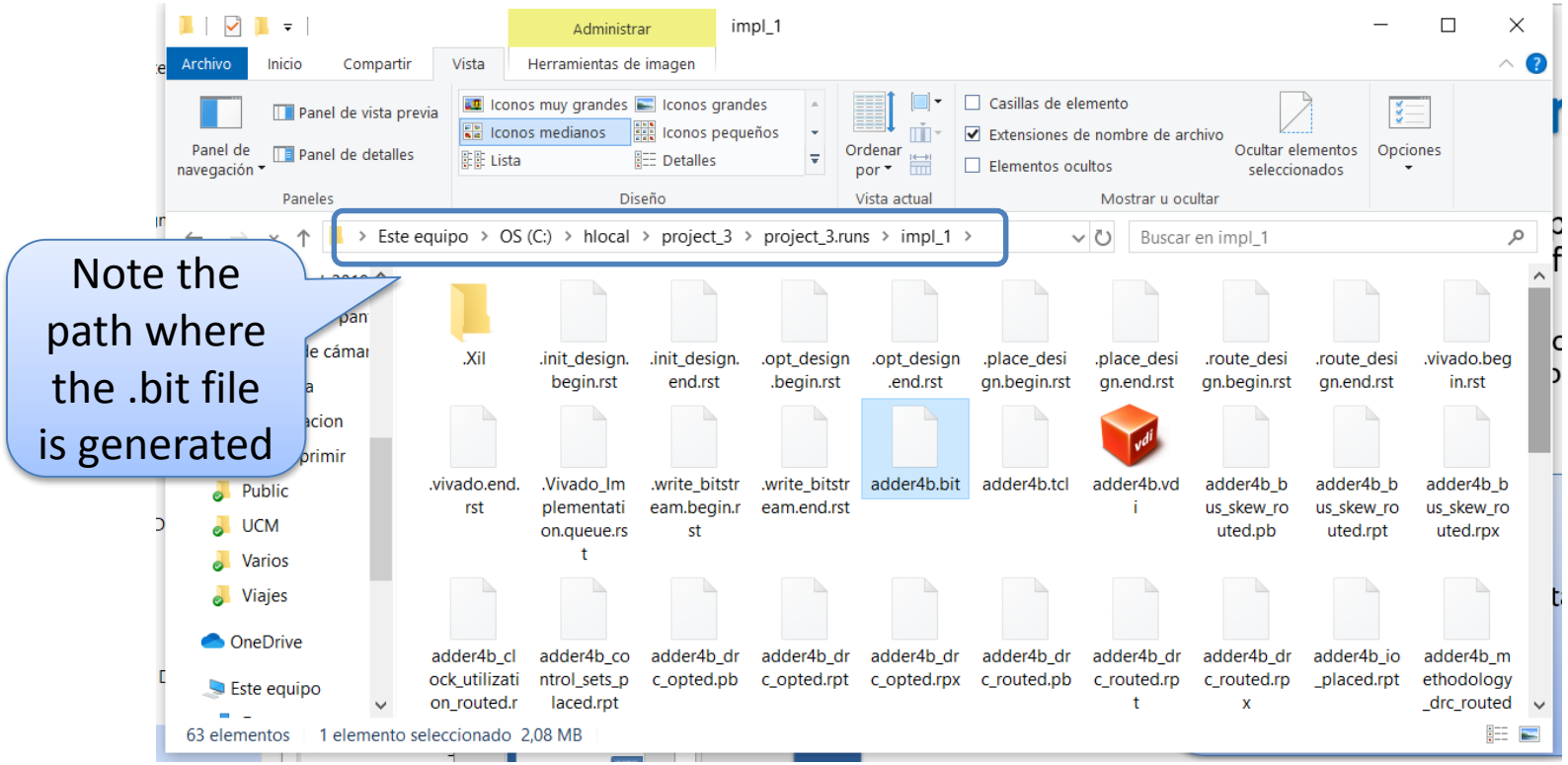
Implementation

- The *pins.xdc* file is added to the project by clicking in "Add Sources" → "Add or Create Constraints"
- The file will be visible in the sources panel of the VIVADO GUI.



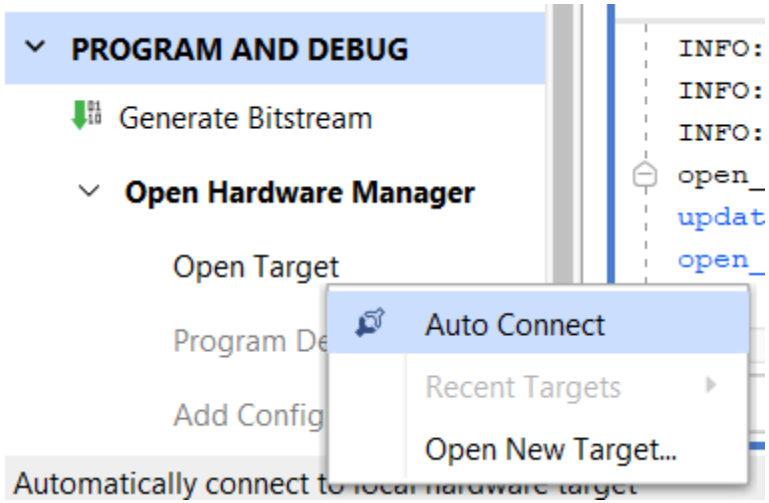
Implementation

- The last steps are clicking on the “Run Implementation” option (bottom left of the main GUI) and finally, “Generate bitstream”. When it is done, click on “Open Hardware manager”
  - This last step will generate an implementation file (*top_entity_name.bit*), which can be used to program the FPGA.

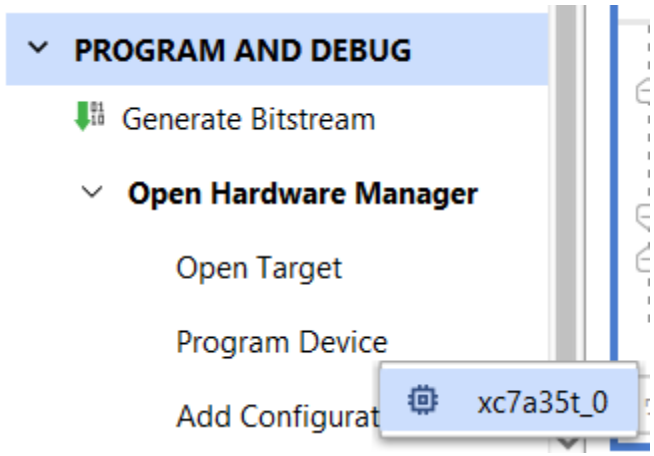


Download .bit on the FPGA

1. Borrow the FPGA to the lab assistants.
2. Connect the USB cable to one of the USB ports of the PC and make sure that the POWER switch is in position ON.
   - The device will be programmed with a by-default .bit file (which makes the 7-segment displays to show numbers from 0 to 9).
   - DON'T STEAL THE USB CABLE (other students are going to need it).
3. If you didn't open the "Open Hardware Manager", click on Target → Auto Connect

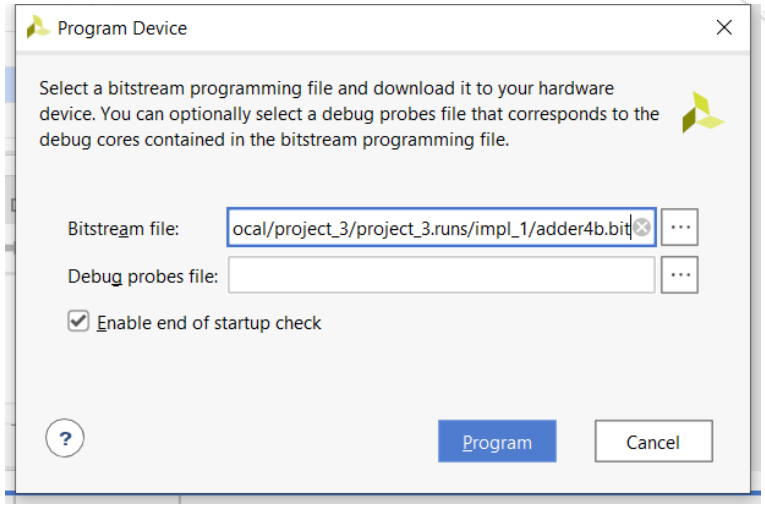


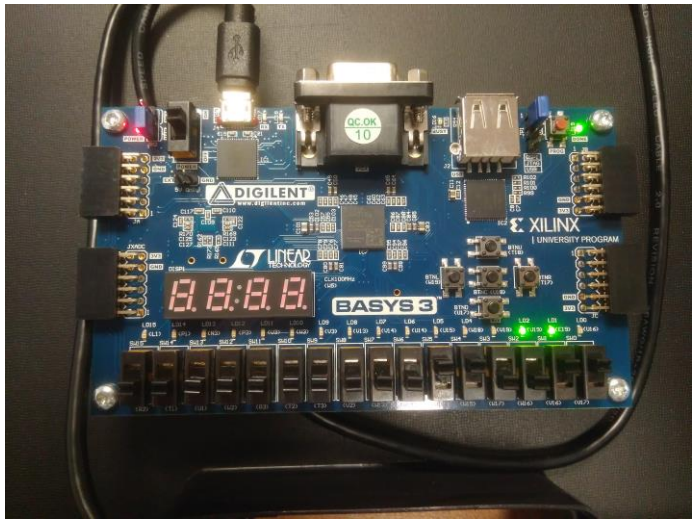
4. Anyway, click on Program Device → xc7a35t_0



Download .bit on the FPGA

- Finally, a .bit file must be provided (the .bit file of the open project is selected by default). Click on “Program”.
- The bitstream will be downloaded to the FPGA and you can use it as expected (try to move SW7-SW0 to see how the result of the addition is made visible on LD3-LD0).
- To go back to edit a source file, click on “PROJECT MANAGER” (top left of the main VIVADO window).





Lab 1.b

- Simulate and implement a register with parallel input / parallel output.
- We are going to program two versions of it:
  - WITHOUT frequency divider (simulation).
  - WITH frequency divider (implementation).

Lab 1.b

```
------------------------------------------------------------
-- register with parallel input / parallel output
------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity parallel_reg is
    port( rst, clk, load: in  std_logic;
          I:              in  std_logic_vector(3 downto 0);
          O:              out std_logic_vector(3 downto 0) );
end parallel_reg;
```

File register.vhd

Lab 1.b

```
architecture circuit of parallel_reg is
begin

    process(rst, clk)
    begin
        if clk'event and clk = '1' then
            if rst = '1' then
                O <= "0000";
            elsif load = '1' then
                O <= I;
            end if;
        end if;
    end process;
end circuit;
```

File register.vhd

Implementation Lab 1.b

- Clock pin:

Uncomment these three lines (by deleting the #'s)

Make sure that your clock signal is named exactly as it appears here (in this case, clk)

```
## Clock signal
set_property PACKAGE_PIN W5 [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]
```

This line indicates that clk is a clock signal with a period of 10.00 ns (the oscillator in pin W5 generates such a clock signal). The CAD tool will place and route it accordingly to optimize the Static Timing Analysis (STA, this concept is discussed in Lesson 2)

- Use the clock divider to feed a 1HZ clock to the register in order to be aware (as humans) of the transitions.
- In addition, make sure that you assign 6 switches (1 for reset, 1 for load, and 4 for input I) and 4 LEDs (for output O).

Grading

- Follow the instructions of the slides to simulate and implement the designs of Labs 1.a and 1.b.
- You have to show me these two parts working on the FPGA (don't forget the clock divider).
- There will be an EXTRA part (0.15 pts).