

# **EE-334**

# **Digital System Design**

Introduction to the Vivado Design Suite

Ludovic Blanc,  
adapted from Andreas T. Kristensen

17/09/2025

Telecommunications Circuits Laboratory

# Introduction

- This tutorial describes how to **implement a VHDL design on a Xilinx FPGA**
  1. We first introduce the **FPGA design flow** and the necessary user inputs
  2. Then, a step-by-step **example illustrates the use of the Xilinx Vivado design** suite from project creation all the way to programming the FPGA device
- This tutorial is tailored to the PYNQ-Z2 evaluation board with Vivado 2020.2
  - Other versions of Vivado (slightly older/newer) should be fine although the layout and some paths may change

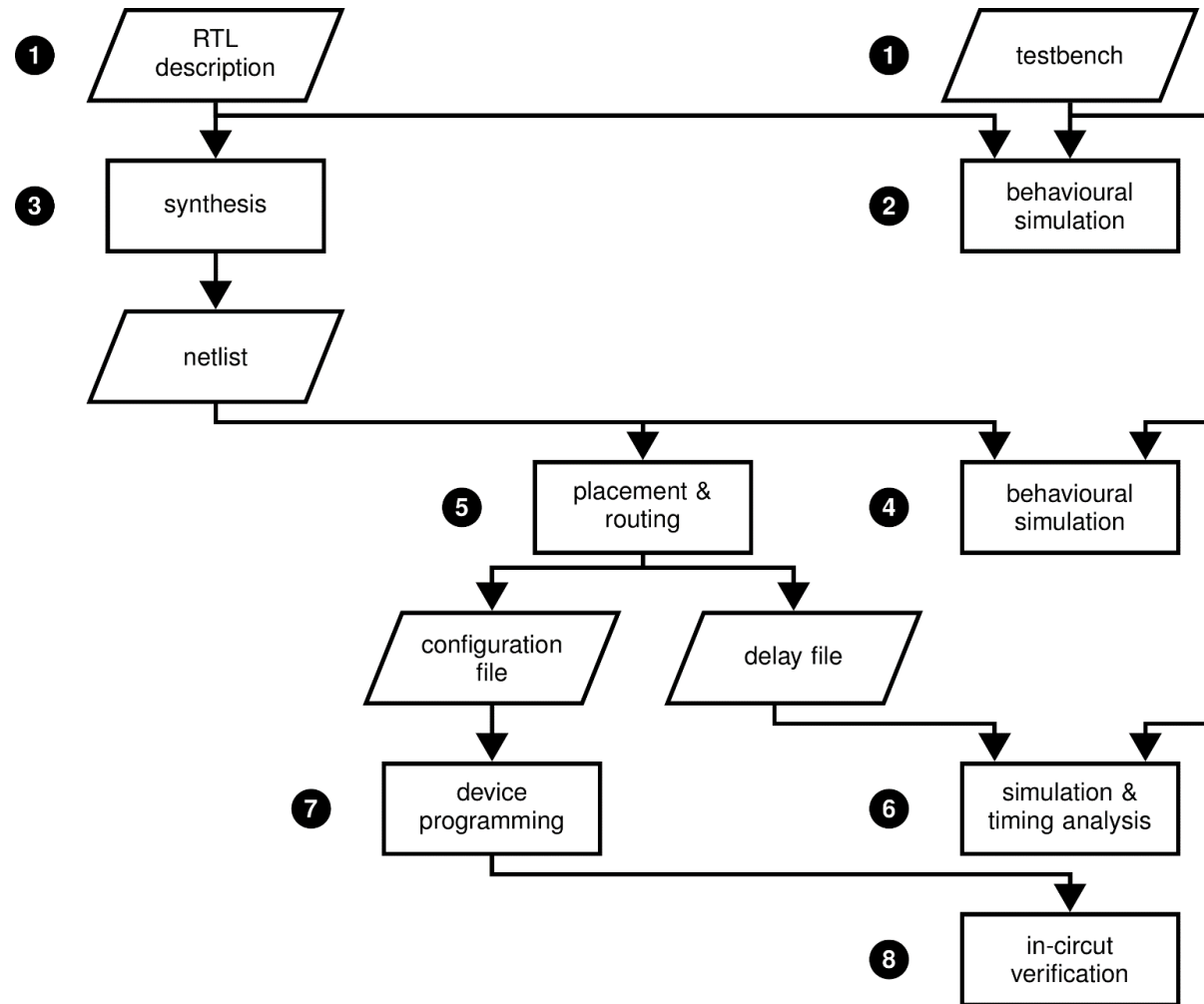
# Introduction: Learning Objectives

- After this tutorial you will have met the following learning objectives:
  - **Create a Vivado project** for a simple combinational design with a testbench for verification and constraint files for the FPGA
  - **Simulate a design** in Vivado using a testbench
  - **Synthesize a design** in Vivado to generate a netlist
  - **Implement the design and analyze** the area and the implemented circuit
  - **Configure the FPGA** with the bitstream and **verify the design on the FPGA**

# FPGA Design Flow: Overview

- The FPGA design flow comprises the following steps:
  1. **Design:** Develop the design files (RTL descriptions) and an associated testbench
  2. **Behavioral Simulation:** Use the design files as the circuit description and the testbench to simulate the circuit to verify that the design works as expected
  3. **Synthesis:** Synthesize the design to generate a netlist, described in terms of FPGA elements such as LUTs
  4. **Post-Synthesis Simulation:** Simulate to verify the correctness of the synthesized netlist
  5. **Placement and Routing (Implementation):** Components are placed and routed together on the FPGA. Some components may be remapped
  6. **Post-Implementation Simulation and Timing Analysis:** The post-implementation netlist is annotated with delays to simulate with timing information to verify the correctness of the placement and routing and to check whether the circuit meets the timing constraints
  7. **Device Programming:** Generate the configuration file for the FPGA matching the implementation result and program the device
  8. **In-Circuit Verification:** Verify operation of the circuit on the FPGA

# FPGA Design Flow: Overview



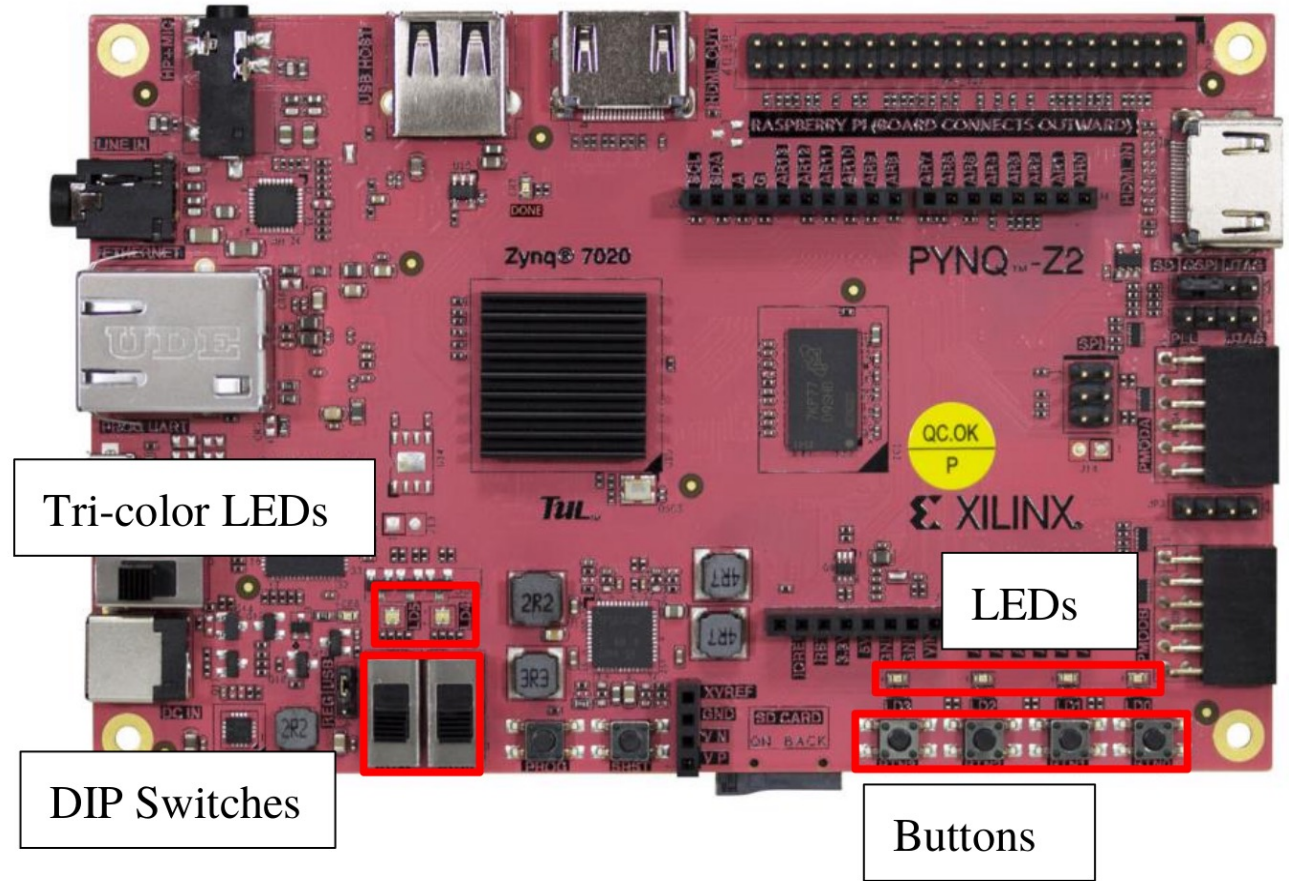
The FPGA design flow with each number matching a step on the previous slide

# FPGA Design Flow: Required Files

- The **designer must provide additional information** besides the hardware description and test-bench, as information related to the specific FPGA cannot be extracted from the RTL code:
  - **Device files** describing the specific FPGA chip for which the design flow is run
  - **Board files** describing the interfaces on the development board and the constraints required to connect the pins of these to the physical FPGA pins
  - **Information on how to connect ports** of the top-level entity to the interfaces of the board
  - **Timing information** (e.g., frequency of the clock) and timing constraints on ports
- The last two points are provided using an `.xdc` (Xilinx Design Constraints) file
- A template `.xdc` file is usually provided with an FPGA board, along with the board files as the layout of the board determines how the physical pins on the FPGA are connected to the other components on the board.

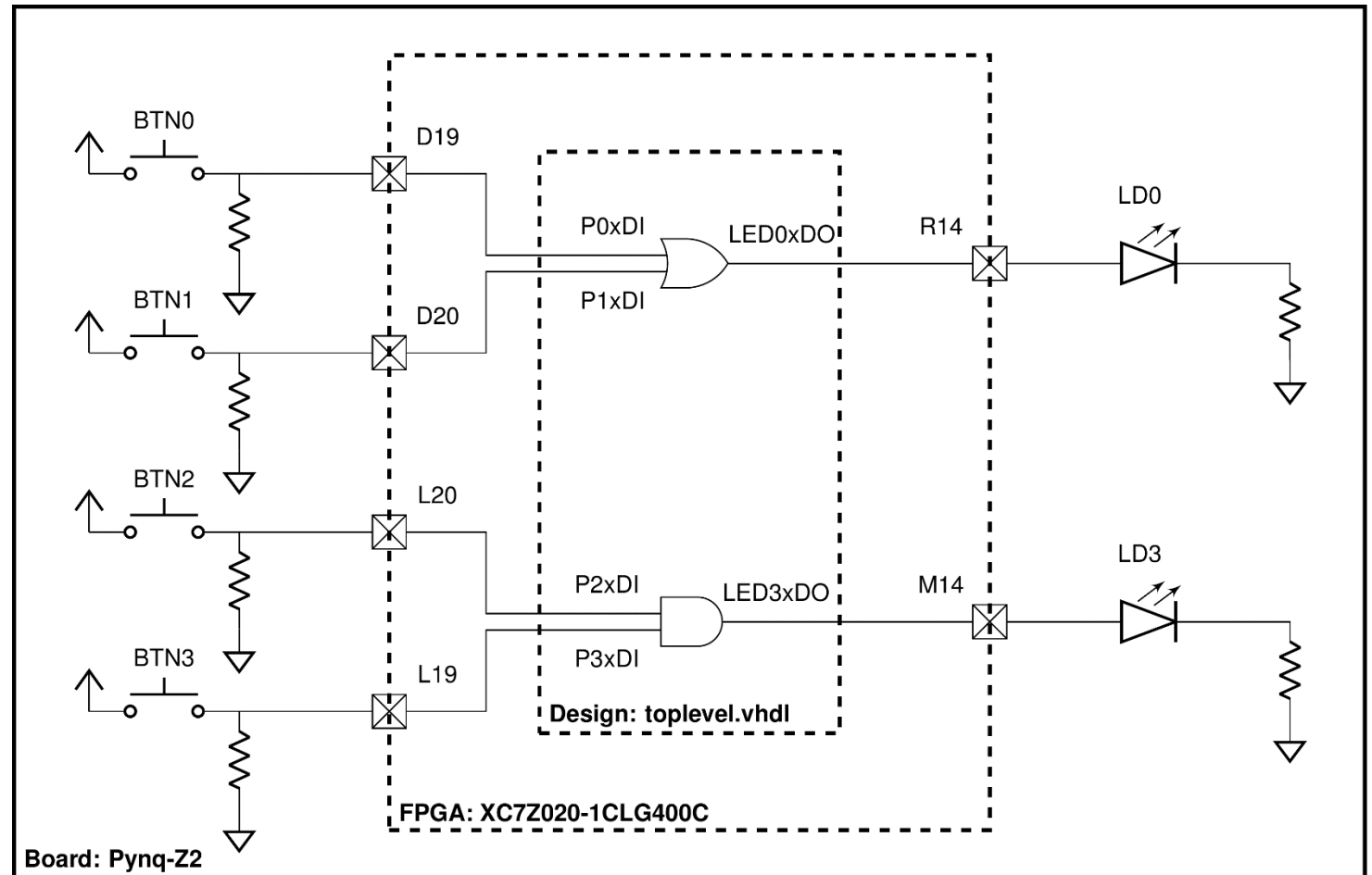
# Tutorial: FPGA Board

- For this tutorial, we use the PYNQ-Z2 development board with the ZYNQ XC7Z020-1CLG400C FPGA
- We will implement a combinational circuit connecting the buttons to the LEDs



# Tutorial: Design

- The tutorial design (toplevel.vhdl) is just 2 gates
- Buttons and LEDs on board are mapped to the pins of the FPGA which map to gate inputs and outputs



# Tutorial: Overview

- The rest of the tutorial is split up into the following parts:
  1. [Preliminaries](#)
  2. [Creating a Vivado project](#)
  3. [Adding source files](#)
  4. [Simulation](#)
  5. [Adding constraints](#)
  6. [Synthesis](#)
  7. [Implementation](#)
  8. [Bitstream generation](#)
  9. [Programming](#)

# Tutorial: Part 1 – Preliminaries

- The first step consists of creating a working environment. You have 2 options:
  1. **Local installation (recommended):** Vivado can be downloaded from [here](#). Select the 2020.2 version. During the installation, pick **Vivado** (not Vitis) and on the next page **pick the free WebPACK** version. Note that this installation (depending on version) can take up to 40 GB of space. See also [moodle](#) for the details of installation and to install the board files.
  2. **Linux remote:** You can use a Linux VM (the one called **STI-EDA-LABS-RTX**, using Ubuntu) though <https://vdi.epfl.ch/>. See also [moodle](#) for the tutorial. And [sharing file between machines](#).
- Note that for option 2, you need to either use the computers in INF3 or have a small local installation of the Vivado Lab Edition in order to program your FPGA

**Please refer to the moodle pages for more information on these options and pick one before progressing in the tutorial!** In the next slides we show how to use the Ubuntu VM and describe some of the files in the handout (so relevant for all!)



# Connecting to the Ubuntu VM

- Please proceed as follows **if you chose option 2 on the previous slide**:
  1. If not on the EPFL intranet, connect first using the VPN
  2. Login to <https://vdi.epfl.ch/> and select a Linux (Ubuntu distro) VM called **STI-EDA-LABS-RTX**
  3. Open a terminal window in the VM and login to the cluster

```
ssh -X gaspar@jed.hpc.epfl.ch
```

- Gaspar is your epfl user name
- Please note the `-X` option for redirecting the graphical terminal to your local display
- Note that this terminal window is the one you will use for the remaining part of the tutorial!
- This terminal window runs on the [epfl jed cluster](#), not the Ubuntu VM!

# Setting up the Vivado project

- Next, we will create a working environment:
  1. **Copy the lab01\_vivado\_handout.zip file (moodle)**
    1. **For Linux server users:** Copy this file to your home directory on selsrv1/2 using, e.g., WinSCP (on Windows) or scp (on Linux) from your own computer. Use selsrv1 or selsrv2 as the host name and your username as the user name for the SCP tool. Unzip with the commands in the next step.
    2. **For local installation:** Copy the zip file to whichever directory you want to use for your files and unzip.
  2. Then, execute the following commands on the linux server to create a directory :

```
mkdir -p ~/dsd22/lab01_vivado_handout && cd ~/dsd22/lab01_vivado_handout
unzip ~/lab01_vivado_handout.zip
cd lab
```

- You can then do `ls` in the terminal to see the directories of the lab, which are:
  - `constr`: Constraint files
  - `pynq`: Contains board files (use these for local installations)
  - `src`: VHDL source code
  - `work_lab01_vivado`: Vivado working directory

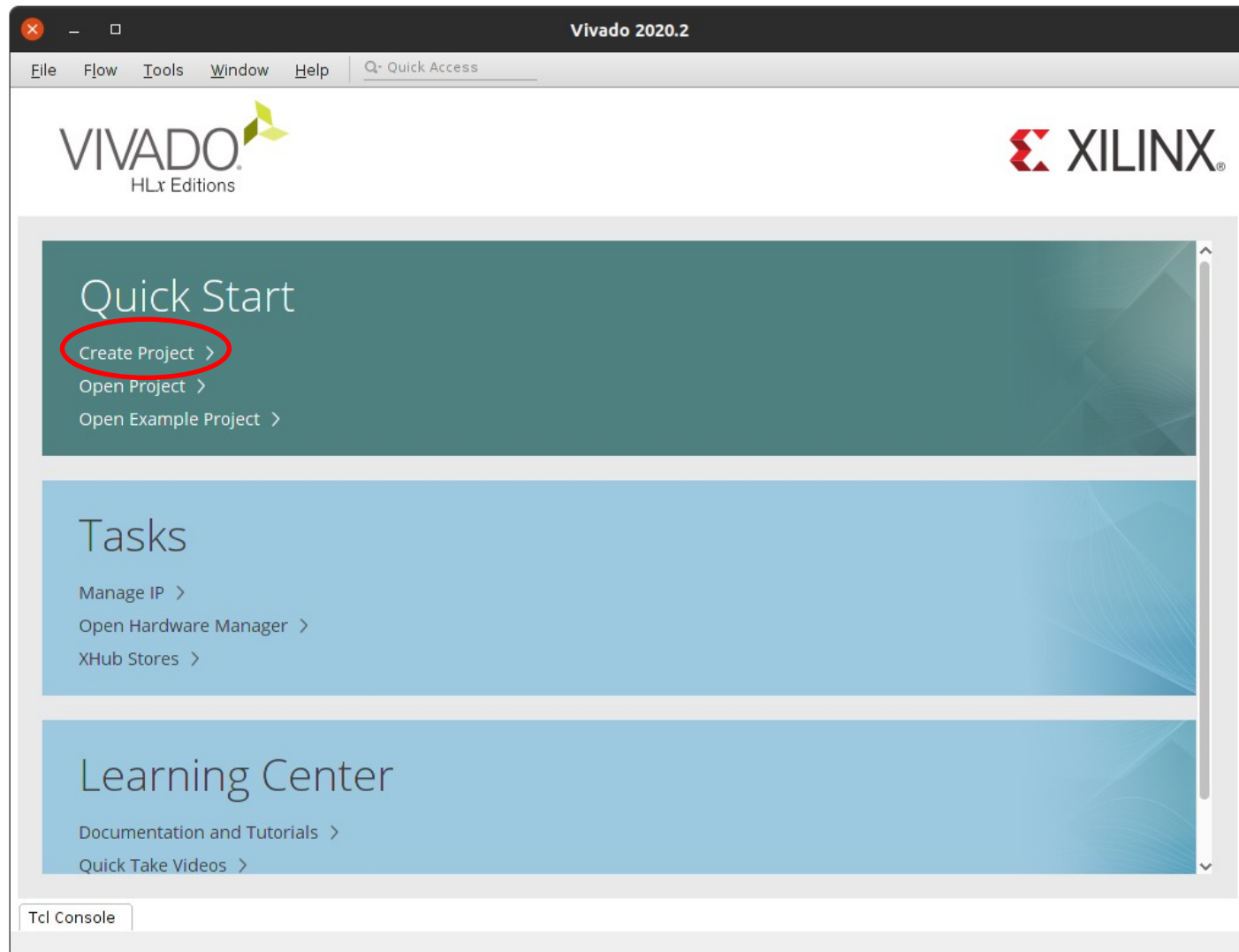
# Creating a Vivado Project

- Now, we create a Vivado project. Run the command below if you're on the Linux server (or have a personal Linux computer), otherwise, just go into the directory  

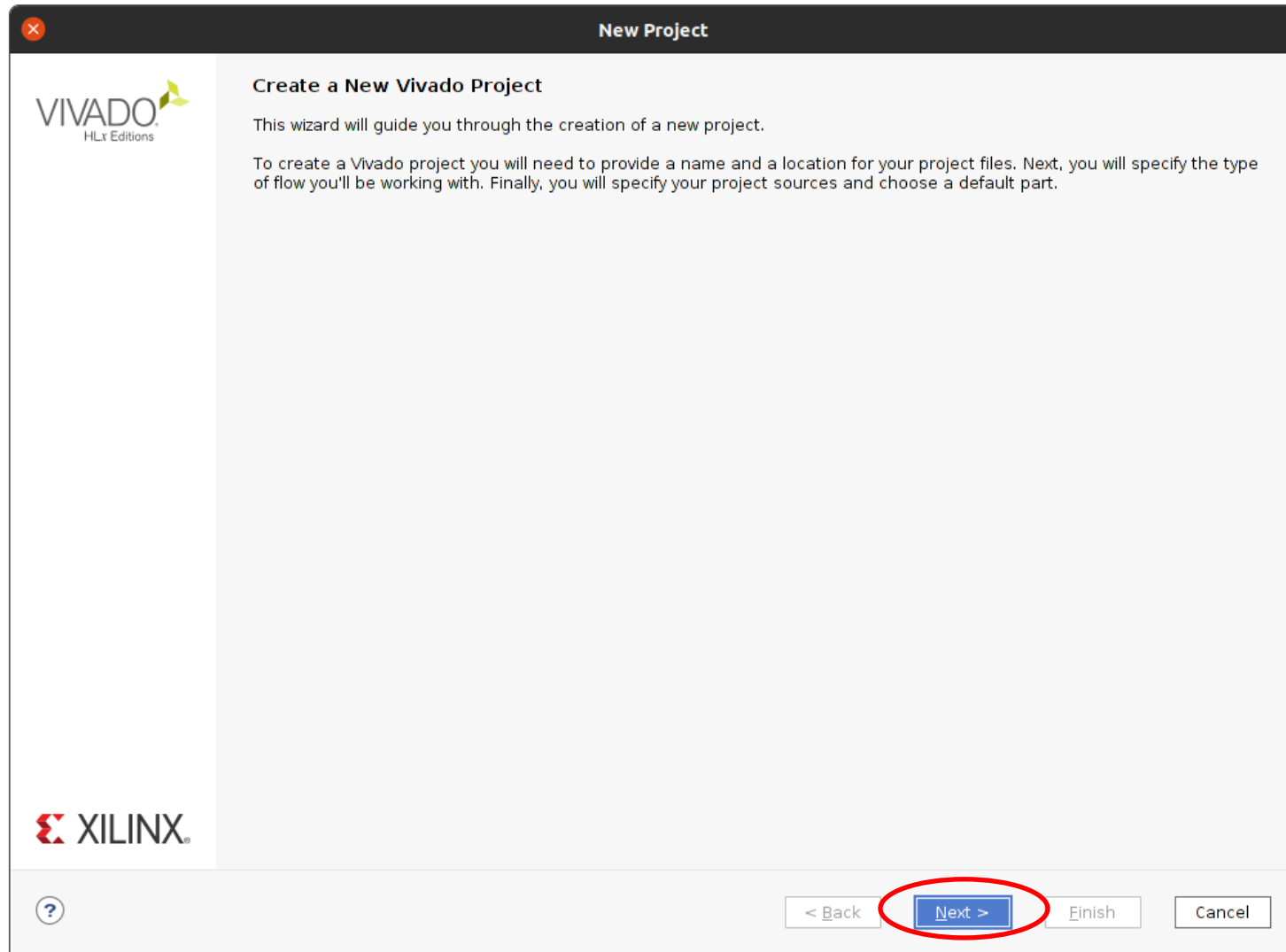
```
cd work_lab01_vivado
```
- Note that this directory contains 2 files:
  - `edatk.conf`: This file specifies the version of Vivado to load. If you have performed a local installation, this file is not necessary
  - `load_board_files.tcl`: This file loads the board files describing the development board
- We then start Vivado as follows depending on your setup
  - **Linux servers**: Start Vivado from the terminal with `vivado -source load_board_files.tcl`
  - **Linux computer**: Start Vivado from the terminal with `vivado`
  - **Windows computer**: Just open Vivado as you would any other program
- Note for students **using a local installation on a Linux machine**, you can run without the `-source` assuming you have installed the board files, otherwise use `-source`. This will use the script provided to load the board files

# Tutorial: Part 2 – Creating a Vivado Project

- We now consider how to create a Vivado project
- This part of the tutorial is identical for all the installation options



- Click on **Create Project**



- Press **Next**

**New Project**

**Project Name**  
Enter a name for your project and specify a directory where the project data files will be stored.

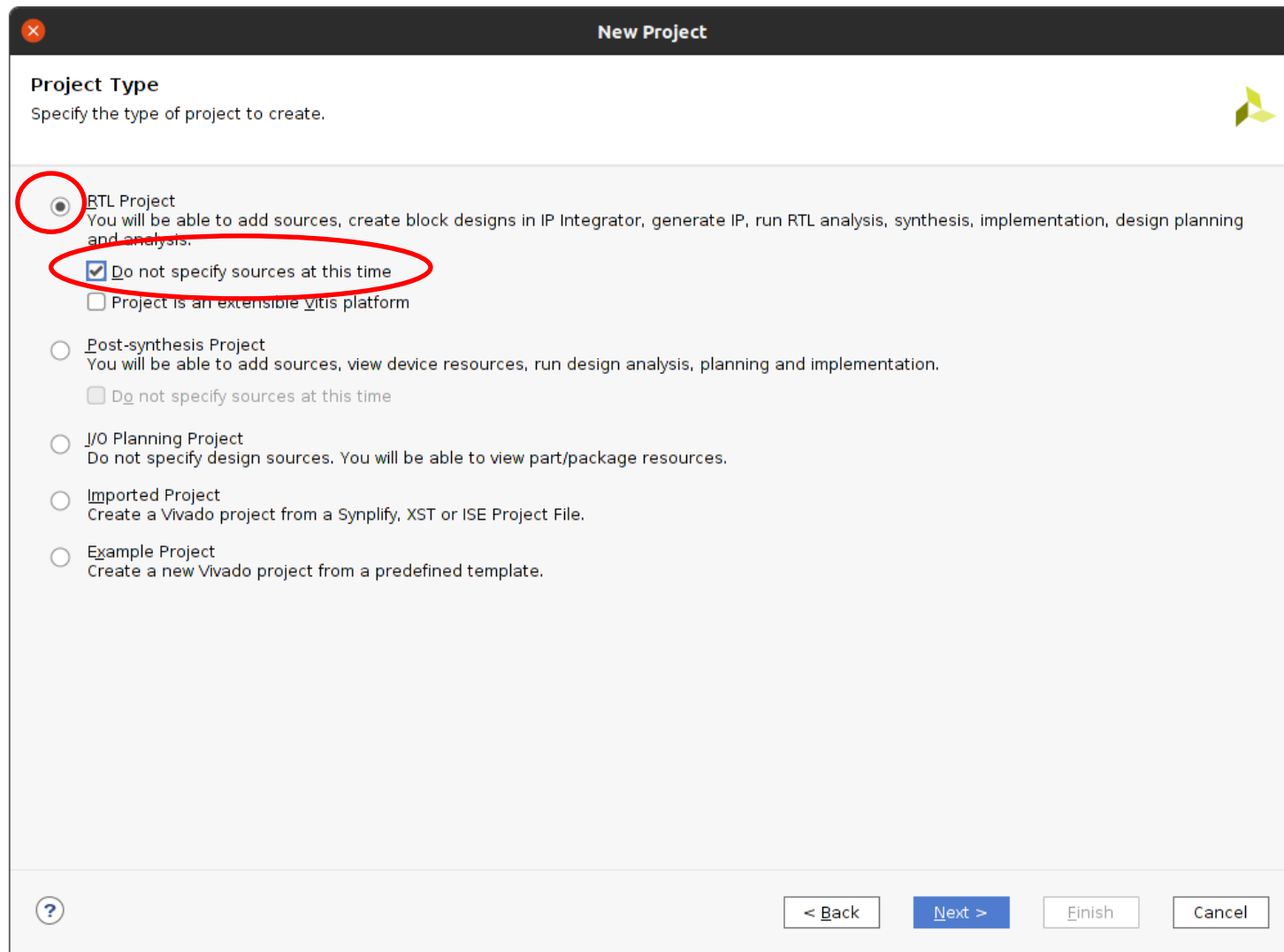
Project name:

Project location:

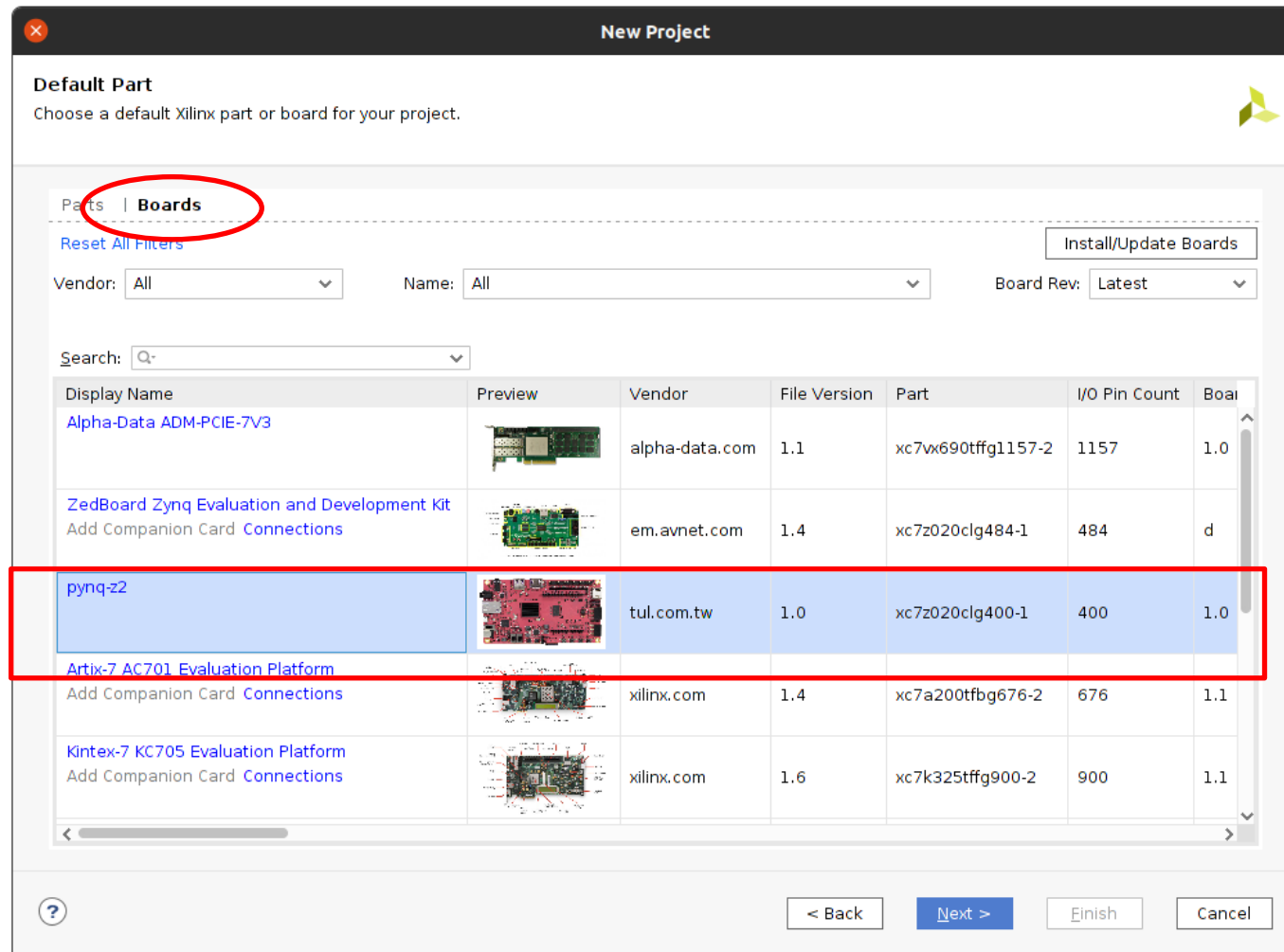
Create project subdirectory

Project will be created at: /home/andreas/gitlab/dsd/AS21/labs/lab01\_vivado/handout/lab/work\_lab01\_vivado

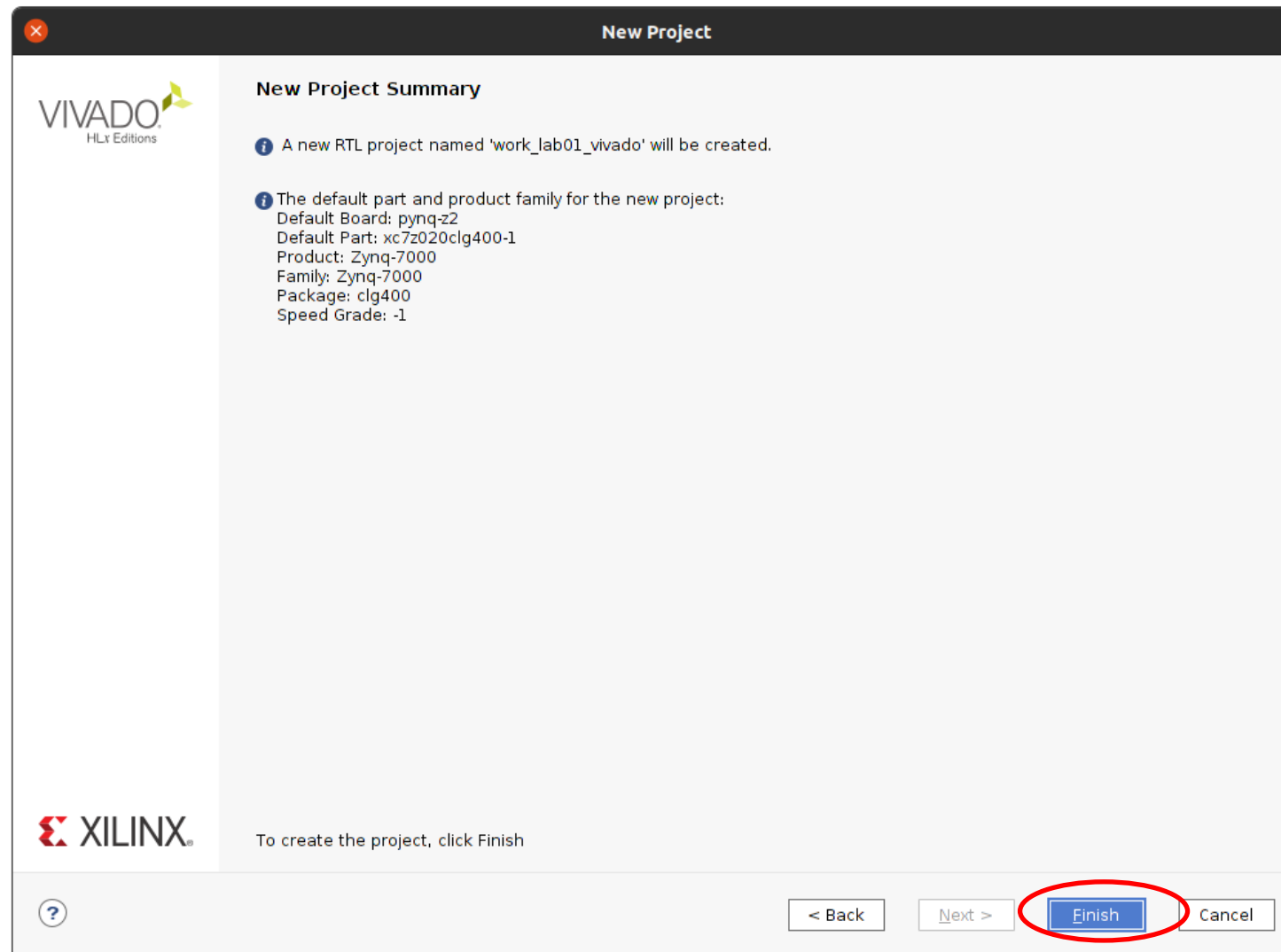
- Change project name to **work\_lab01\_vivado** and **untick Create project subdirectory**. Make sure that you create the project in work\_lab01\_vivado



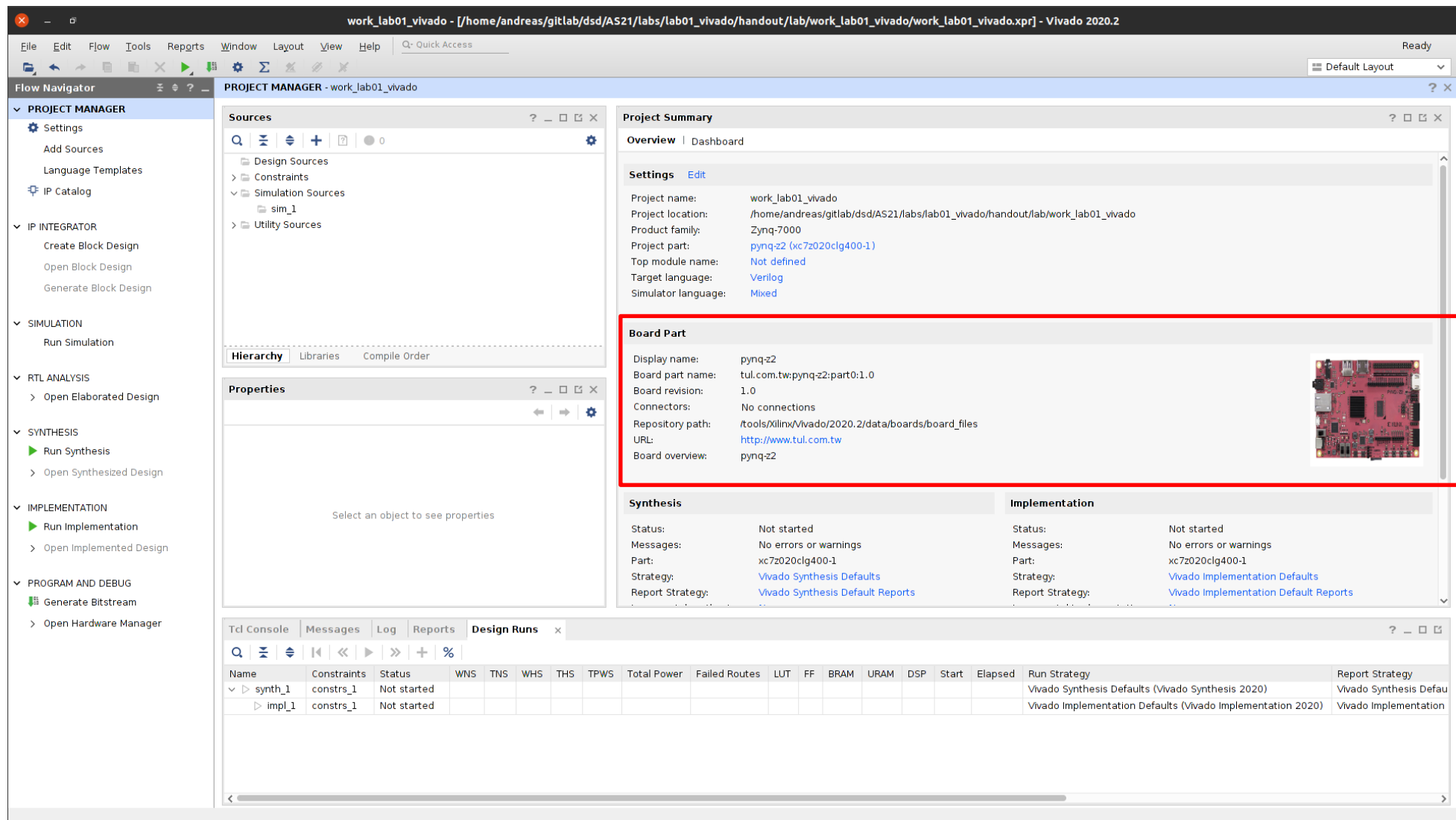
- Select **RTL project** and click **Do not specify sources at this time**, then **Next**



- **Press the Boards tab and select pynq-z2.** The board-file provides information on how to connect the FPGA to, e.g., buttons on the board and the name of the FPGA.
- Note this has to be manually installed for a local Linux/Windows installation or loaded using the .tcl script if using Linux (either local or the servers)



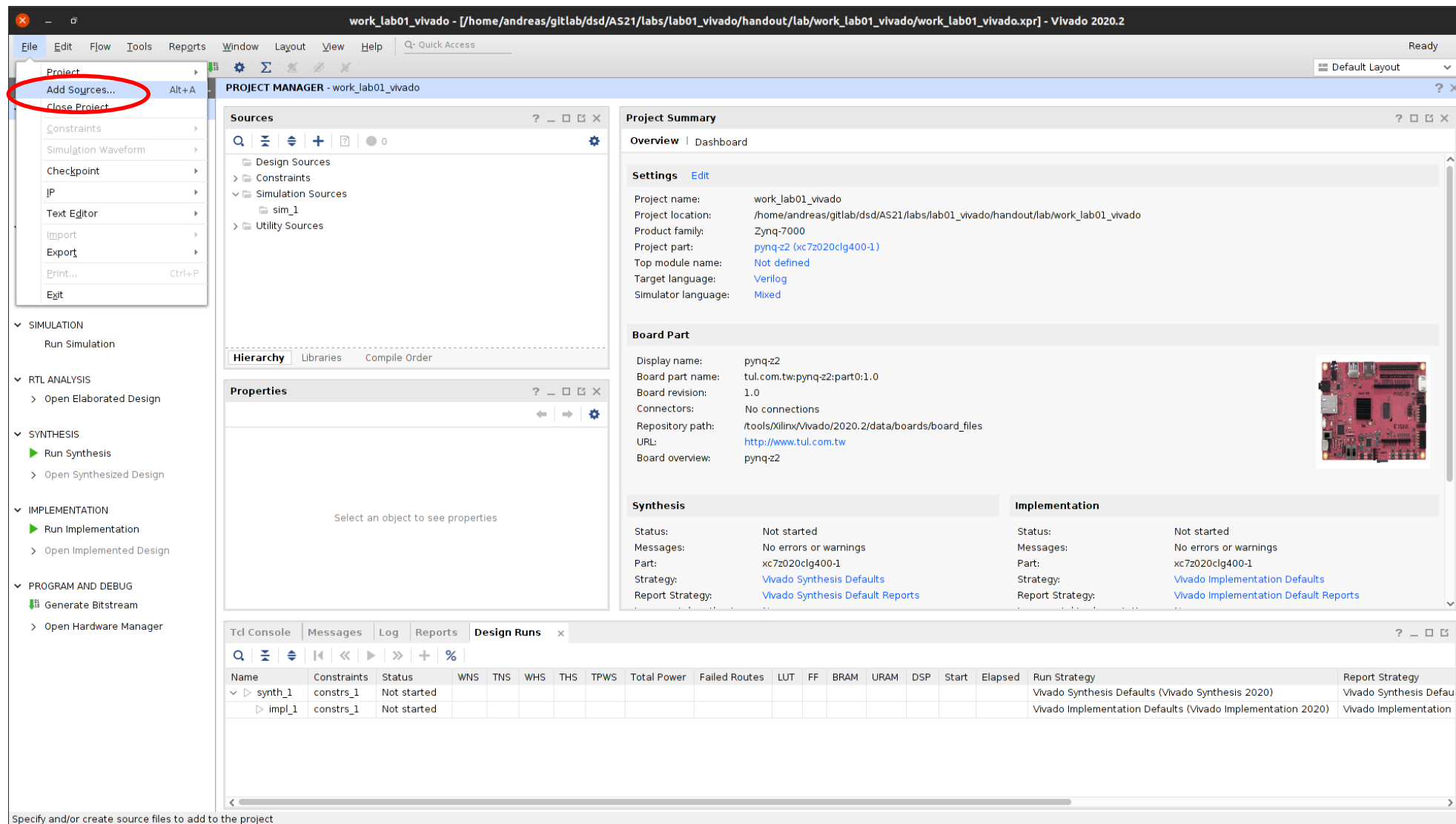
- Verify that you have the same setup, then click on **Finish**



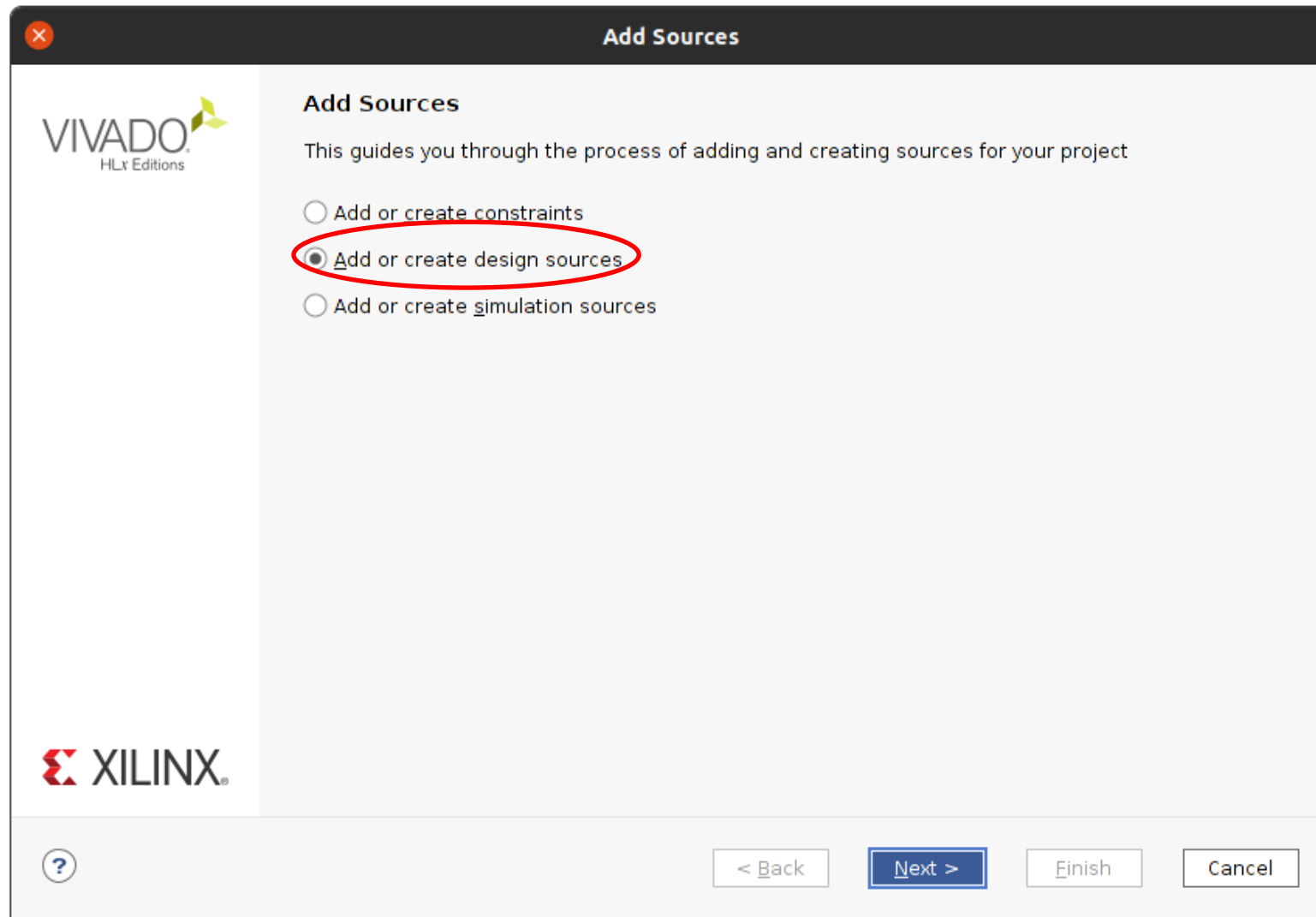
- You have now created the project and we see the Vivado tool! If you loaded the board-files correctly you should see the board part as shown above

# Tutorial: Part 3 - Adding Source Files

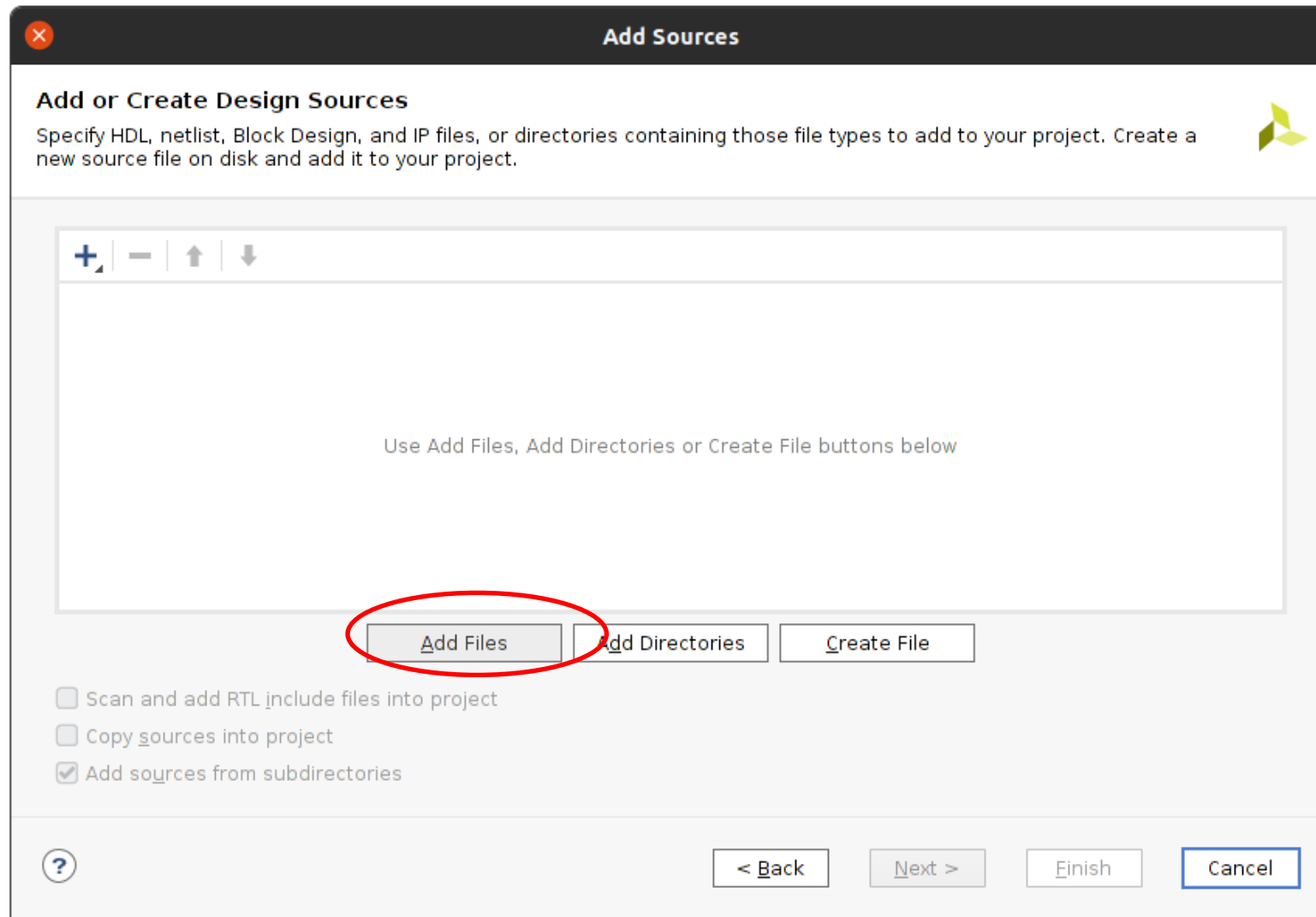
- Source files (VHDL/Verilog/SystemVerilog code) can be added during project creation, but you may add files later after project creation
- We, now show the process of adding the source files after project creation and how to edit these source files in Vivado



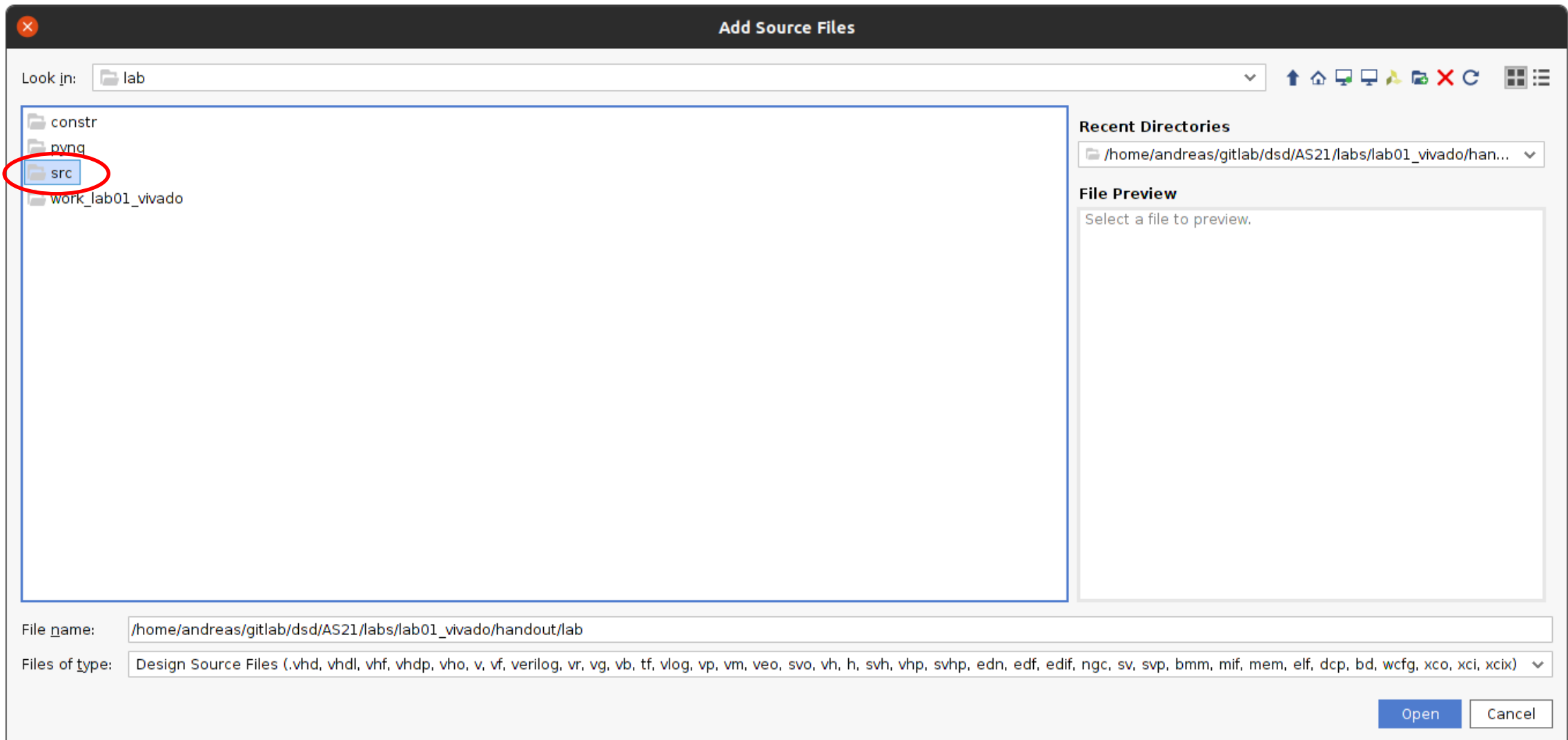
- Click on **File** and then **Add Sources** in the upper left part of Vivado



- Click on **Add or create design sources** in the window that pops up



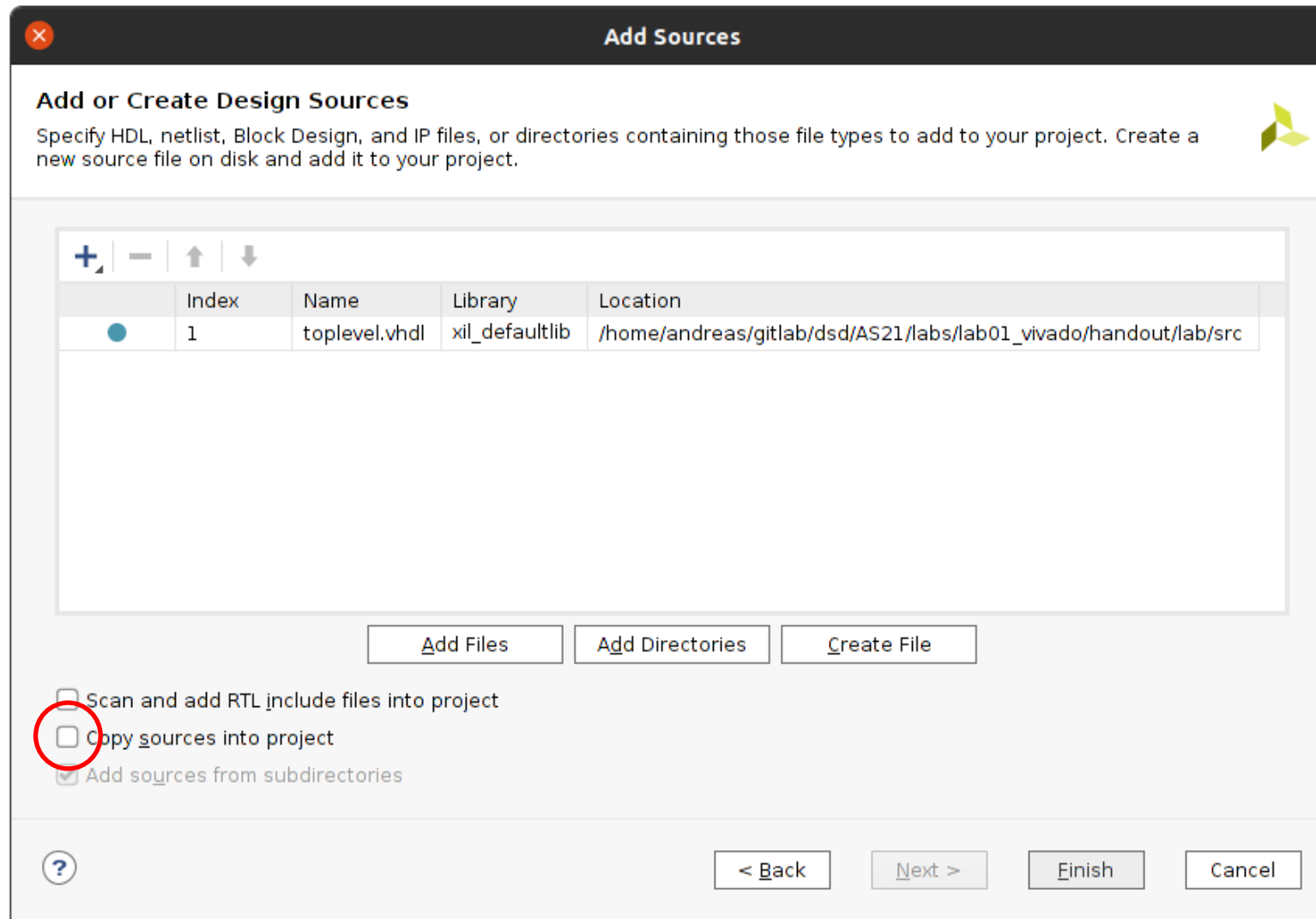
- Click on **Add Files**



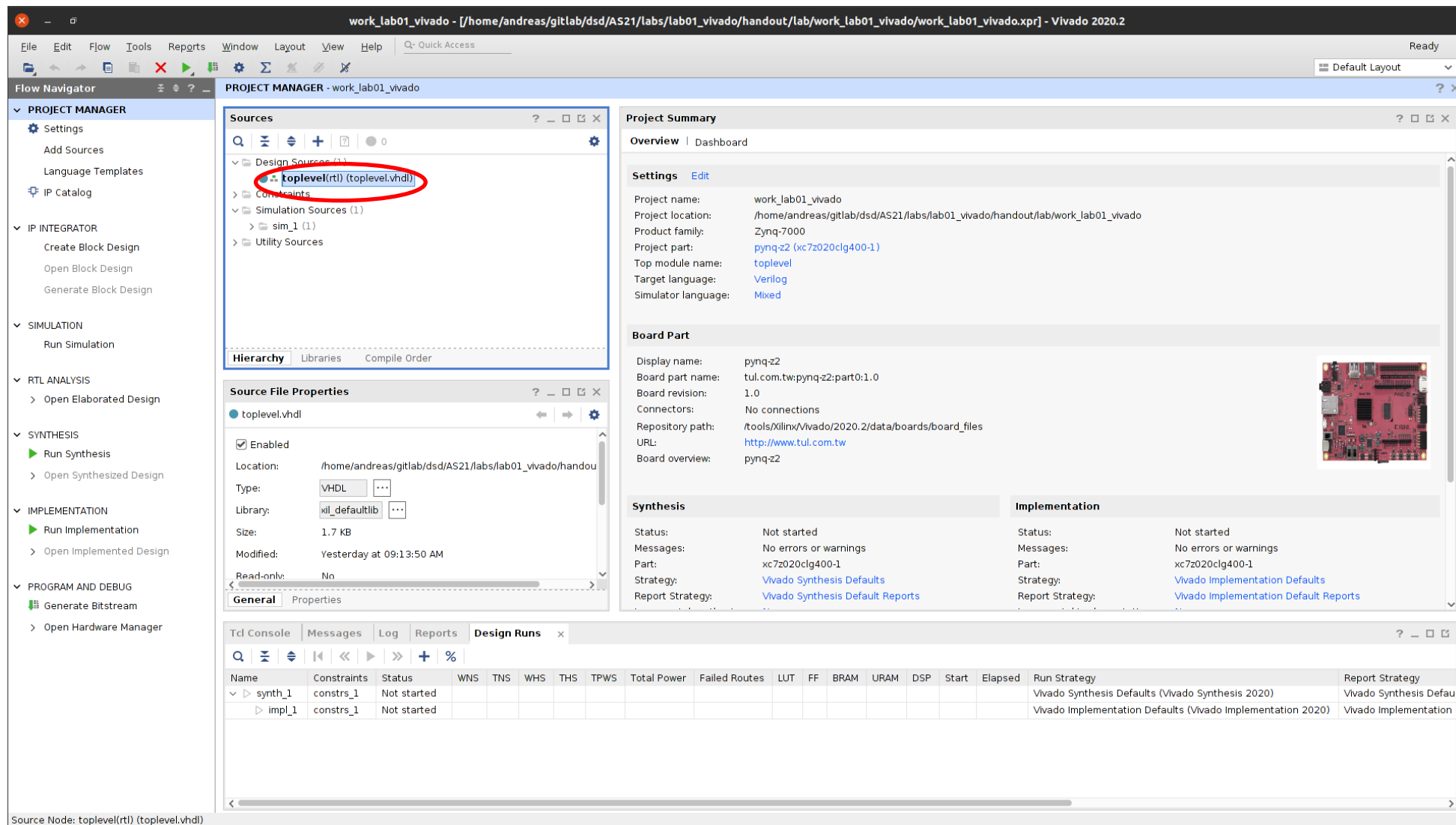
- Navigate up and then into the **src directory**



- Select the **toplevel.vhdl** file and **press OK**



- Ensure **copy sources into project** is **unticked**. It is best to never do this as your project files will otherwise be scattered in multiple locations. Then **press finish**



- You can now see the added files under **Design Sources** in the **Sources** window
- This is the window that shows you information on all the files you add to the project and by pressing on the file here you can edit it directly in Vivado. **Double click on toplevel**

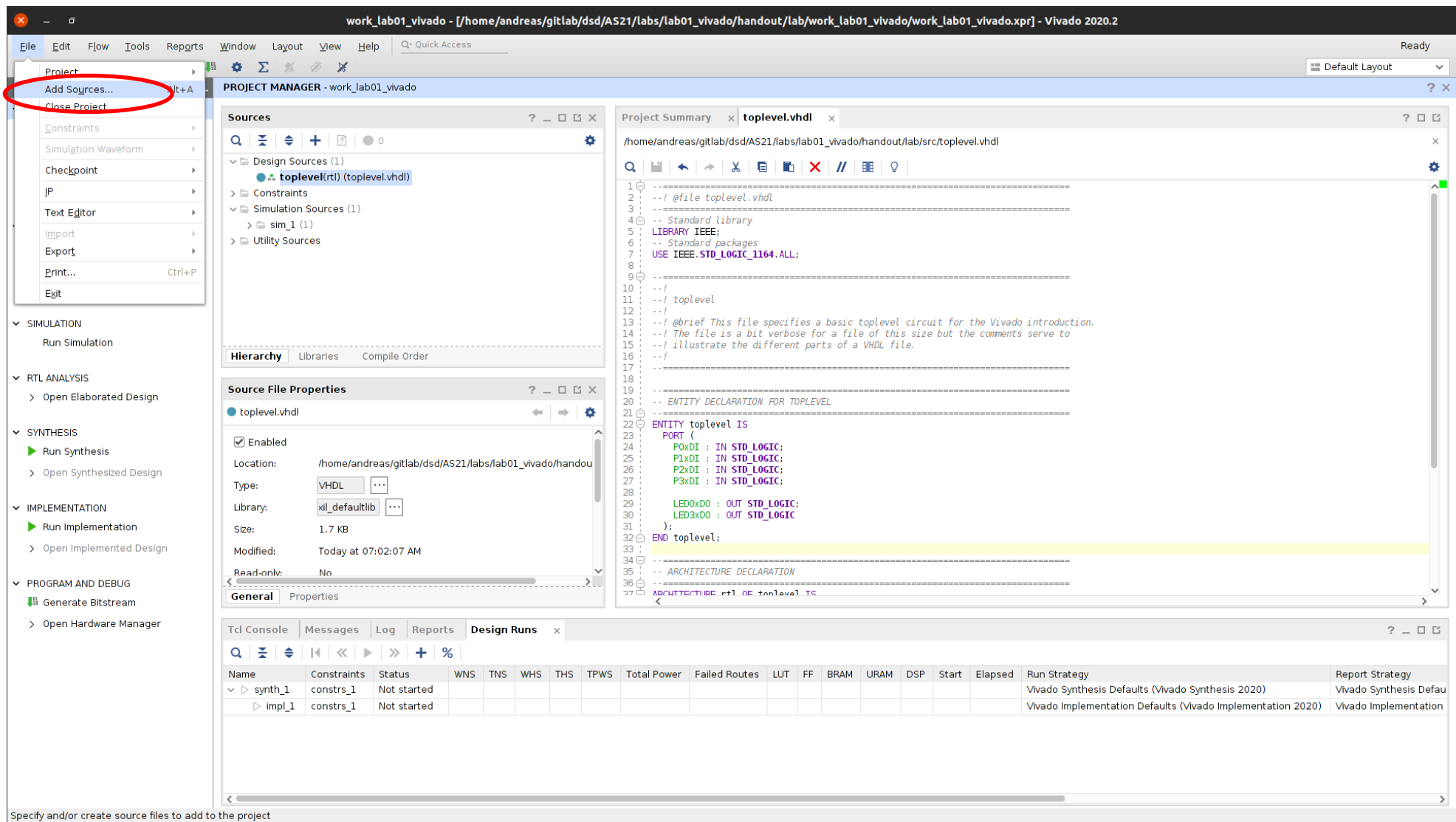
```
1 -----
2 --! @file topLevel.vhdl
3 -----
4 -- Standard library
5 LIBRARY IEEE;
6 -- Standard packages
7 USE IEEE.STD_LOGIC_1164.ALL;
8 -----
9 --!
10 --!
11 --! topLevel
12 --!
13 --! @brief This file specifies a basic topLevel circuit for the Vivado introduction.
14 --! The file is a bit verbose for a file of this size but the comments serve to
15 --! illustrate the different parts of a VHDL file.
16 --!
17 -----
18
19 -- ENTITY DECLARATION FOR TOPLEVEL
20 -----
21
22 ENTITY topLevel IS
23   PORT (
24     P0xDI : IN STD_LOGIC;
25     P1xDI : IN STD_LOGIC;
26     P2xDI : IN STD_LOGIC;
27     P3xDI : IN STD_LOGIC;
28
29     LED0xD0 : OUT STD_LOGIC;
30     LED3xD0 : OUT STD_LOGIC
31   );
32 END topLevel;
33 |
34 -----
35 -- ARCHITECTURE DECLARATION
36 -----
37 ARCHITECTURE rtl OF topLevel IS
```

- This window then shows the file content.
- It is possible to use Vivado as an editor, but most tools like emacs or VS Code will have some support for VHDL through the use of extensions

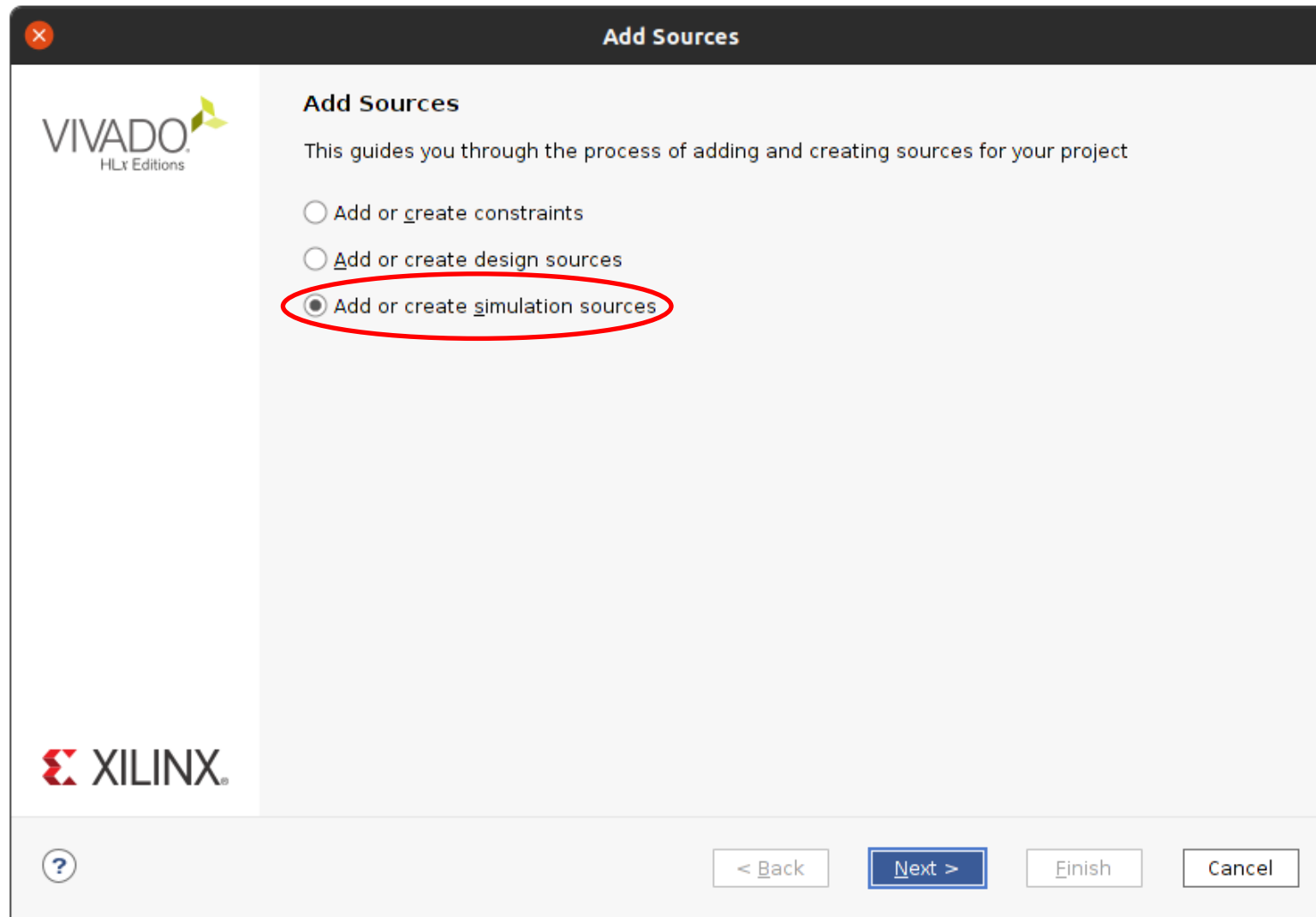
**Task:** Try to remove a semi-colon in the port declaration and see the feedback from Vivado in the text-editor and the Sources window **after saving the file**

# Tutorial: Part 4 - Simulation

- We now consider performing a basic simulation and viewing the resulting simulation output



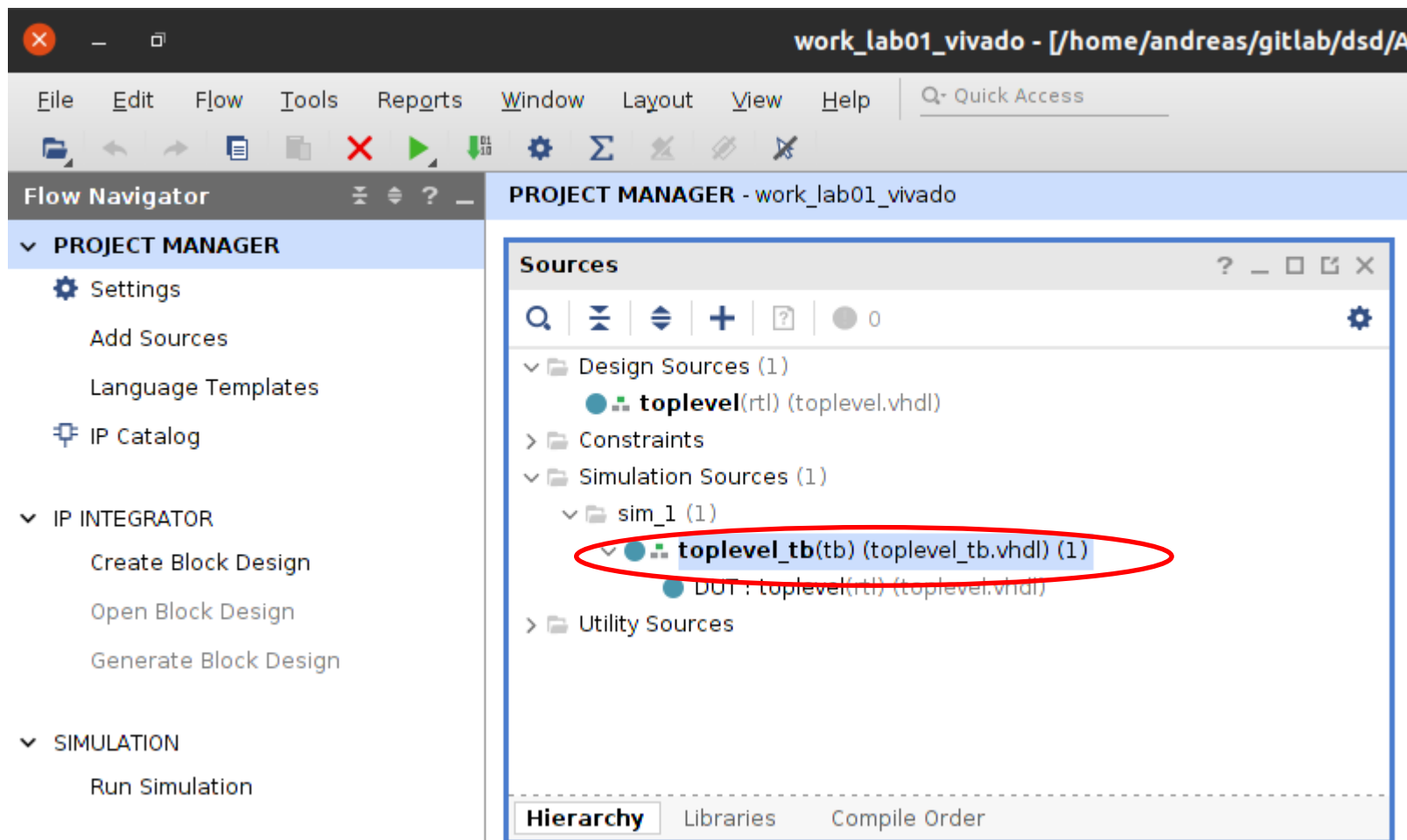
- Click on **File** and **Add Sources** in the upper left part of Vivado



- Click on **Add or create simulation sources**. Simulation files are not a part of the design flow and thus added separately.

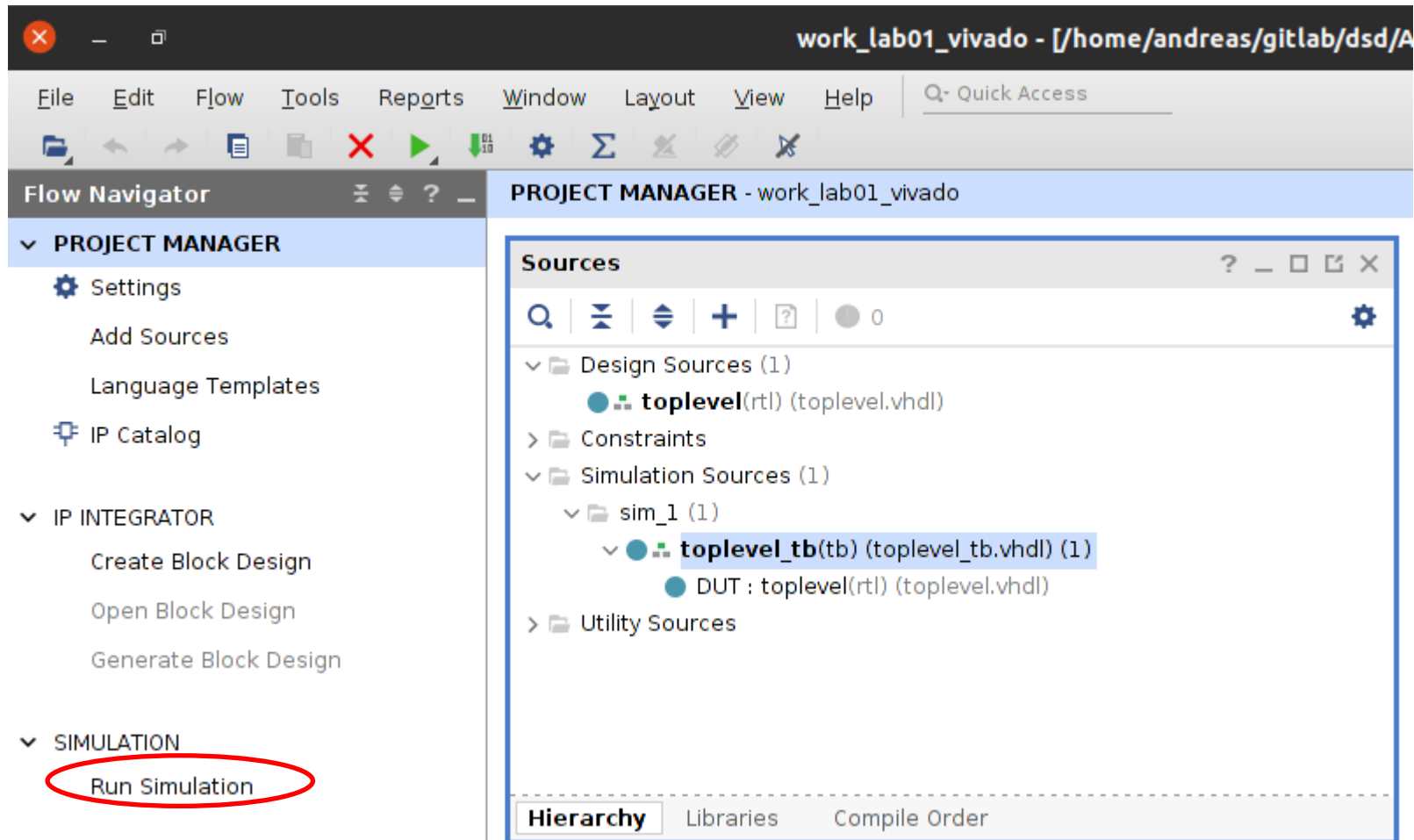


- Navigate back to the src directory and pick **toplevel\_tb.vhdl** to add. The **\_tb** suffix designates a test-bench file.

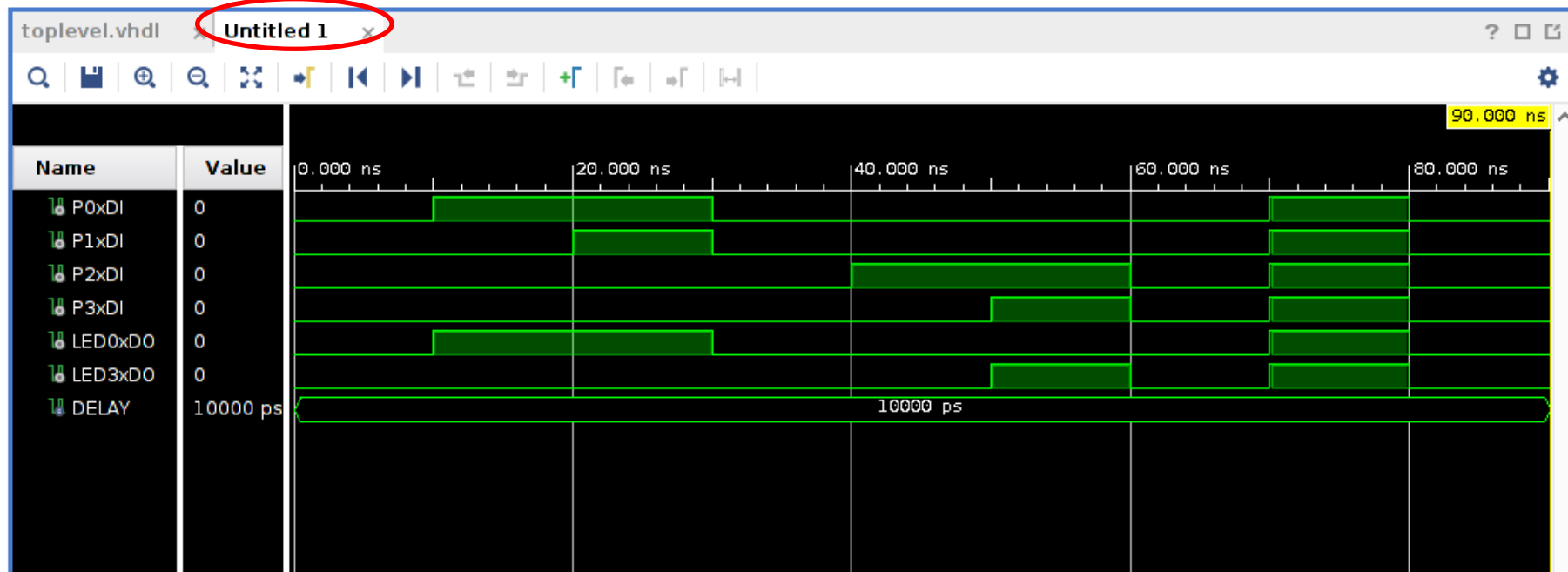


- You now see the added files under **Simulation Sources** in the **Sources** window

**Task:** Note that toplevel.vhdl is beneath toplevel\_tb.vhdl in the simulation hierarchy. Why do you think this is? Try to open the file



- Click on **Run Simulation** --> **Run Behavioural Simulation** in the Flow Navigator to start the simulation.
- **Windows users** may have to modify the firewall/anti-virus tool to run simulations

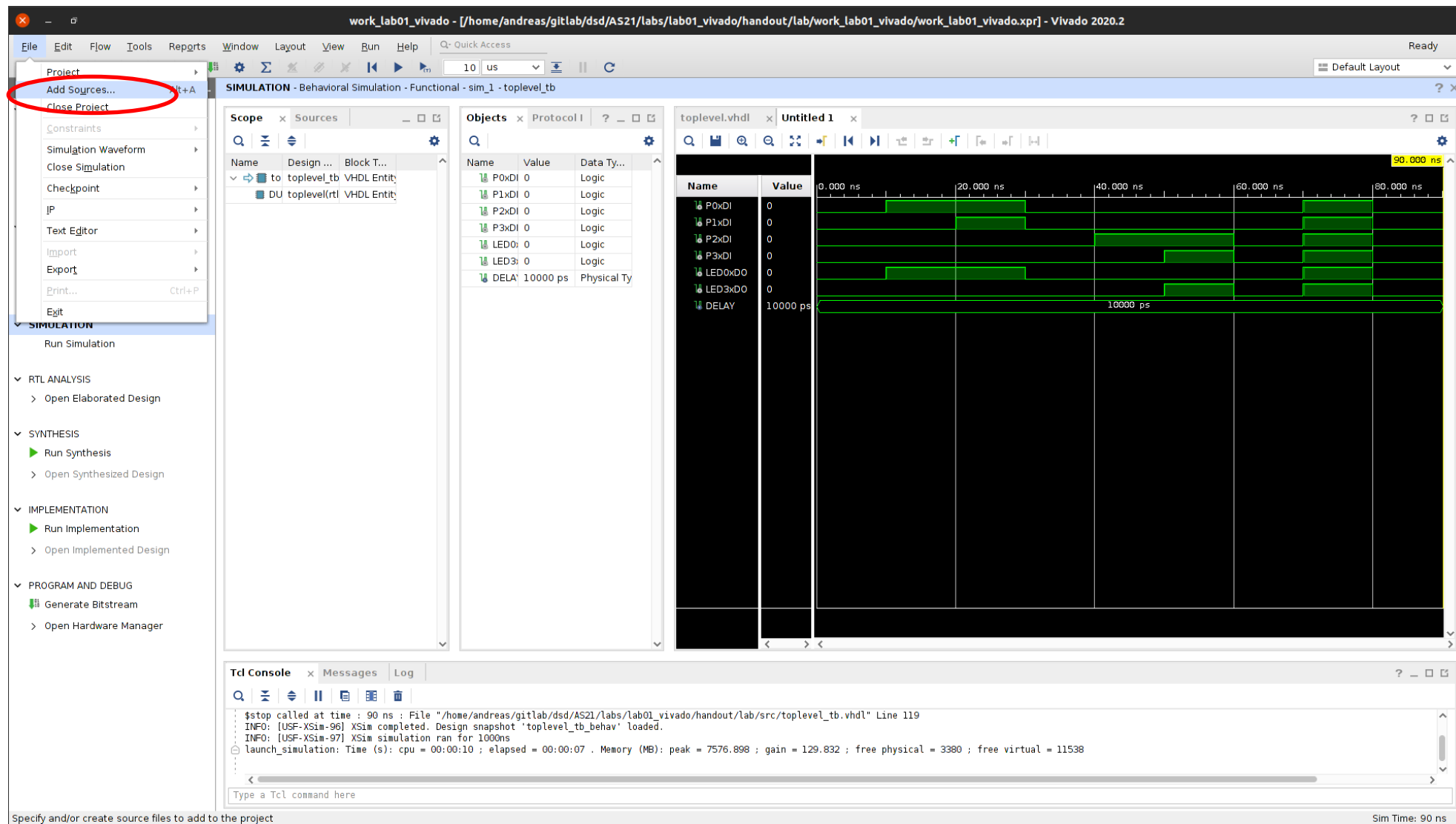


- After the simulation has completed, you should see the simulation output window, here the tab **Untitled 1**.
- You can **right-click and select Full View** inside this window to see the entire simulation

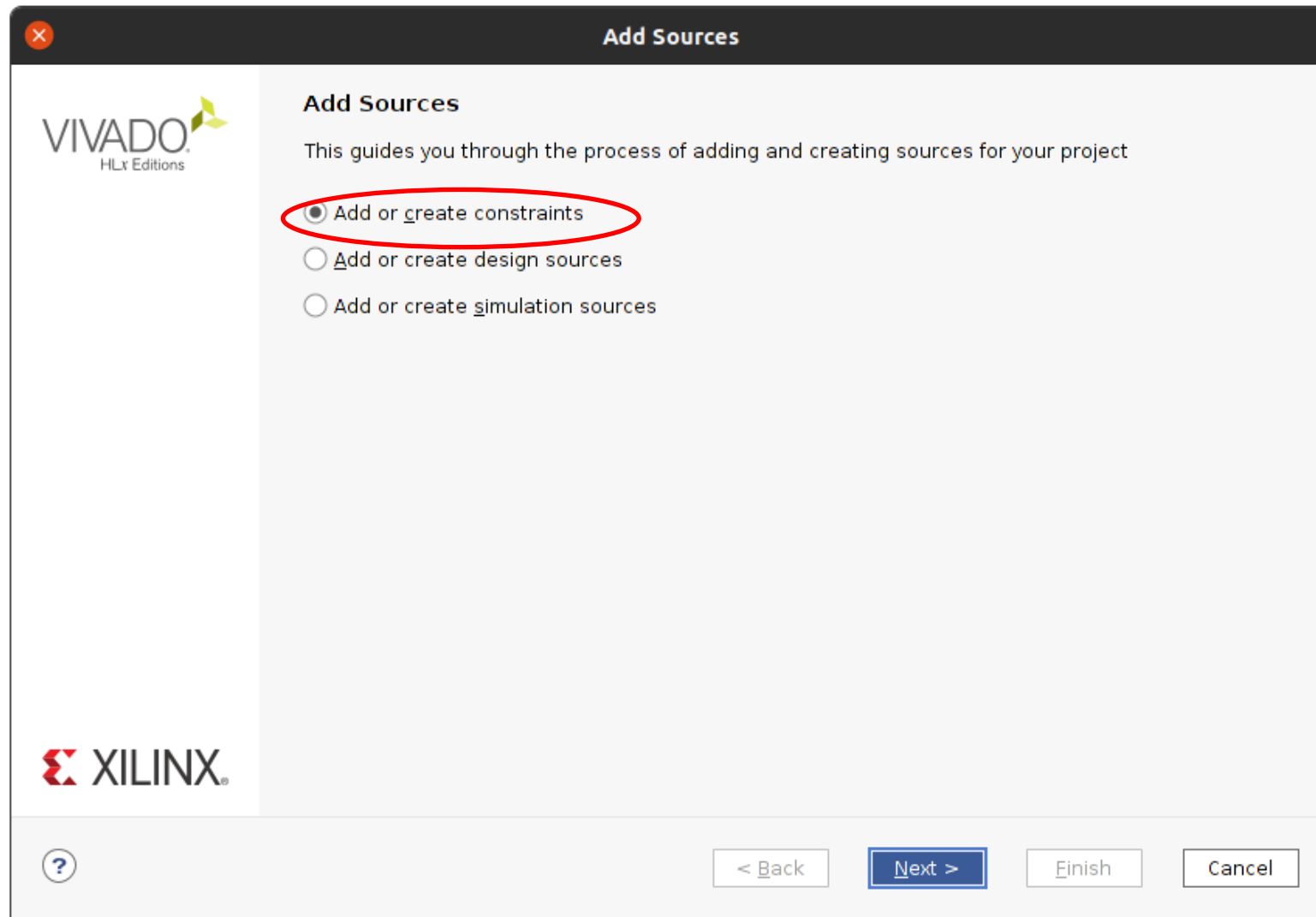
**Task:** Looking at the simulation, is this as you would expect given the previous circuit description? You can try to modify the test-bench and rerun the simulation.

# Tutorial: Part 5 – Adding Constraints

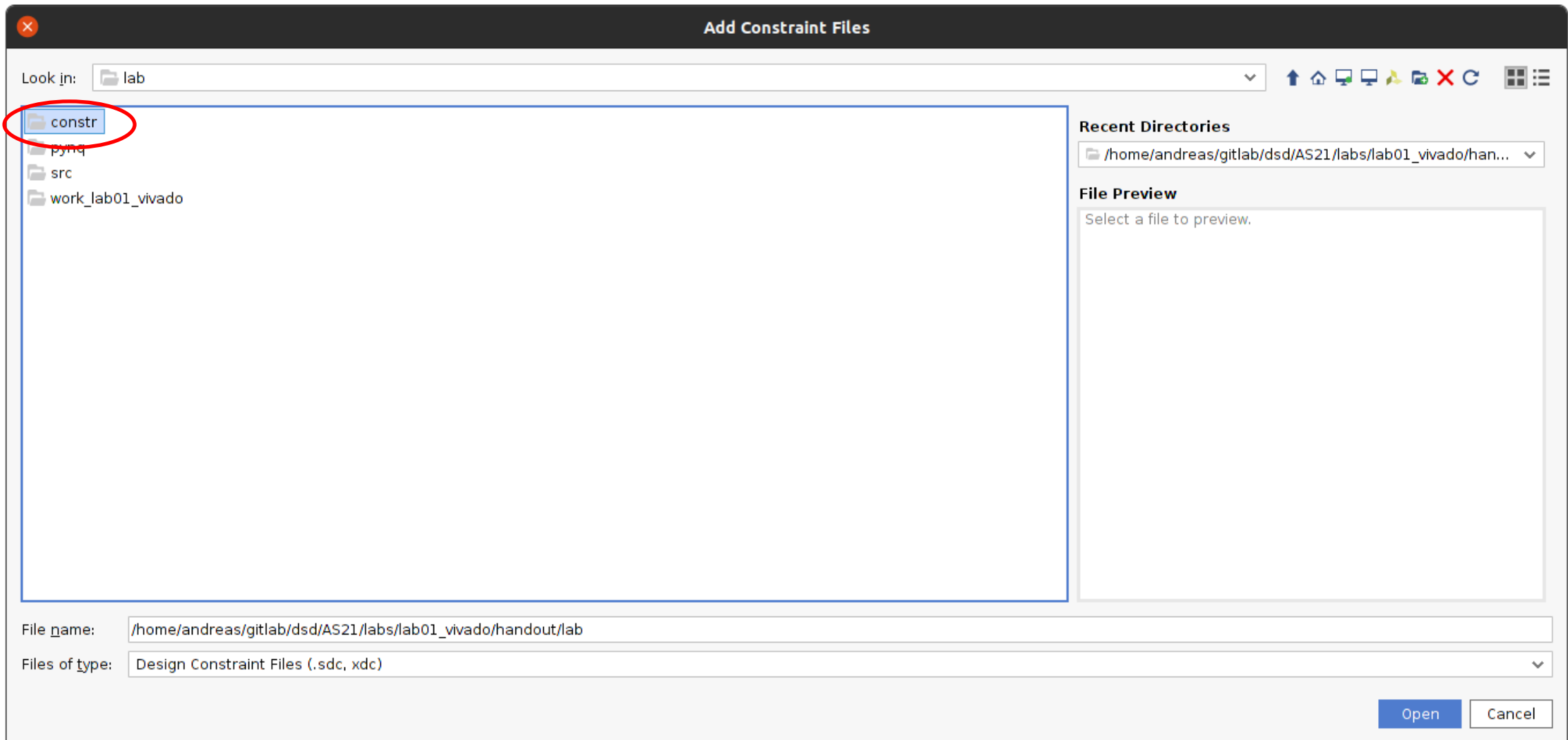
- Before we proceed with the synthesis and implementation steps, we add some constraints in the form of an `.xdc` file to the project.



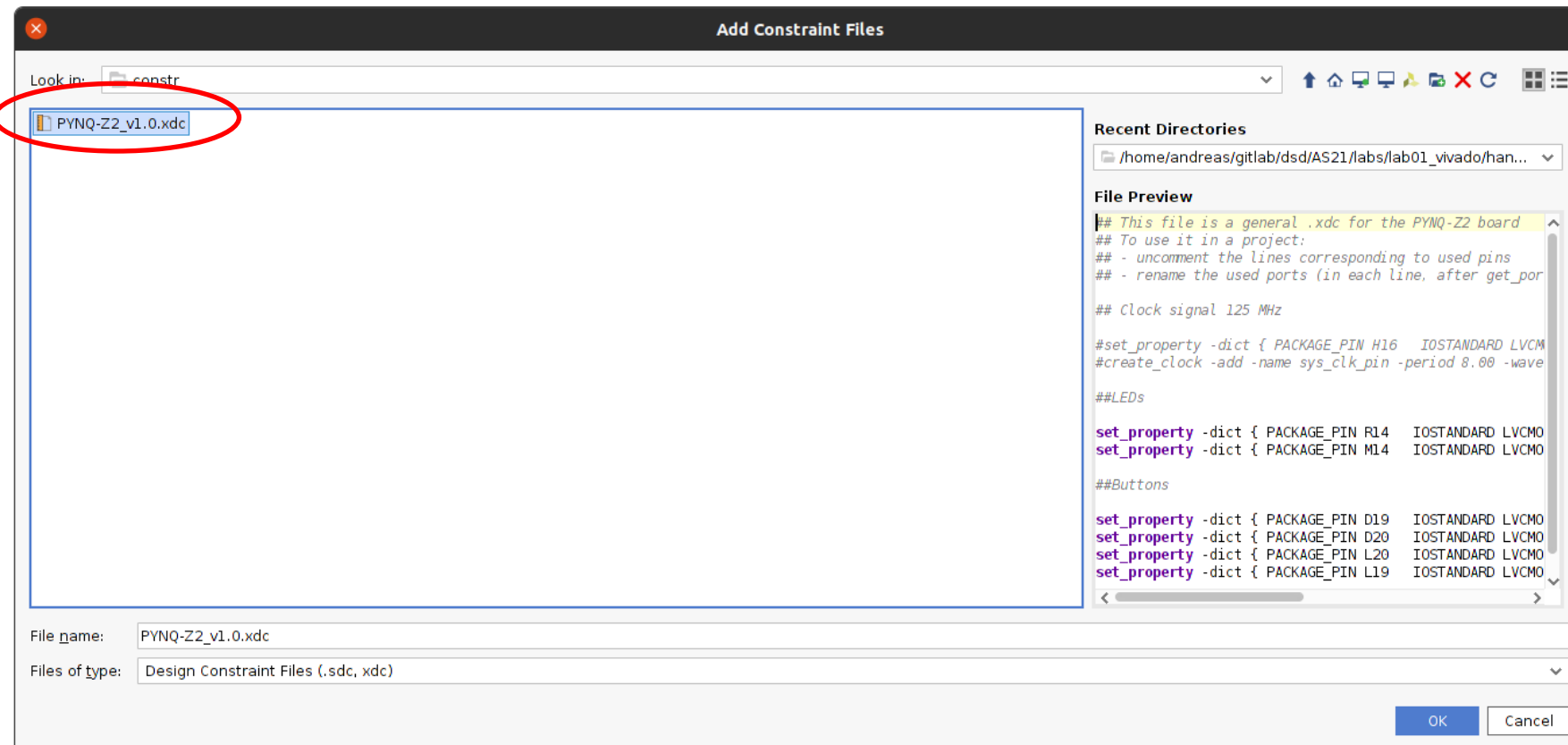
- Click on **File** and **Add Sources** in the upper left part of Vivado



- Click on **Add or create constraints**



- Navigate up and then into the **constr** directory



- Select the **PYNQ-Z2\_v1.0.xdc** file and click **OK**
- For future uses of `.xdc` files, please remember they are case-sensitive!

**Task:** Look into the `.xdc` file to try to understand the format in relation to the schematic shown in the introduction. Note the reference to a clock which are commented out. We will use this in a future tutorial

# Tutorial: Part 6 - Synthesis

- We now consider the synthesis step, which will generate a netlist describing your design in terms of FPGA resources, such as LUTs.

The screenshot shows the Vivado 2020.2 IDE interface. On the left, the Flow Navigator window is open, and the 'Run Synthesis' option under the 'SYNTHESIS' section is circled in red. The main workspace is divided into several panes: 'Scope' and 'Objects' on the left, a 'Name Value' table in the center, and a timing diagram on the right. The timing diagram shows signals P0xDI, P1xDI, P2xDI, P3xDI, LED0xD0, LED3xD0, and DELAY over a 100ns period. The Tcl Console at the bottom displays the following log messages:

```
INFO: [USF-XSim-96] XSim completed. Design snapshot 'toplevel_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:10 ; elapsed = 00:00:07 . Memory (MB): peak = 7576.898 ; gain = 129.832 ; free physical = 3380 ; free virtual = 11538
add_files -fileset constrs_1 -norecurse /home/andreas/gitlab/dsd/AS2L/labs/lab01_vivado/handout/lab/constr/PYNQ-Z2_v1.0.xdc
```

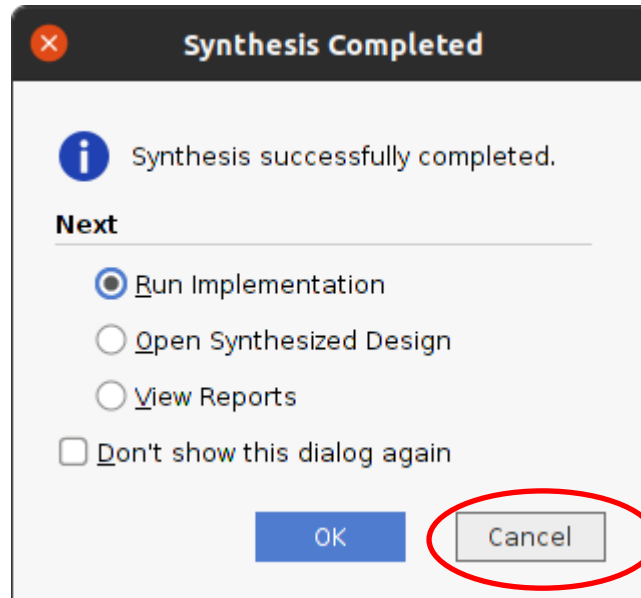
- Select **Run Synthesis** from the Flow Navigator window. In the window that opens change the number of jobs to 1 (if using servers) and press **OK**

The screenshot shows the Vivado 2020.2 interface during a behavioral simulation. The top window displays a timing diagram for signals P0xDI, P1xDI, P2xDI, P3xDI, LED0xD0, LED3xD0, and DELAY. The DELAY signal is highlighted with a 10000 ps pulse. The bottom window shows the Tcl Console with a 'Log' tab selected, displaying synthesis progress messages. The 'Synthesis' tab is also highlighted at the bottom of the console window.

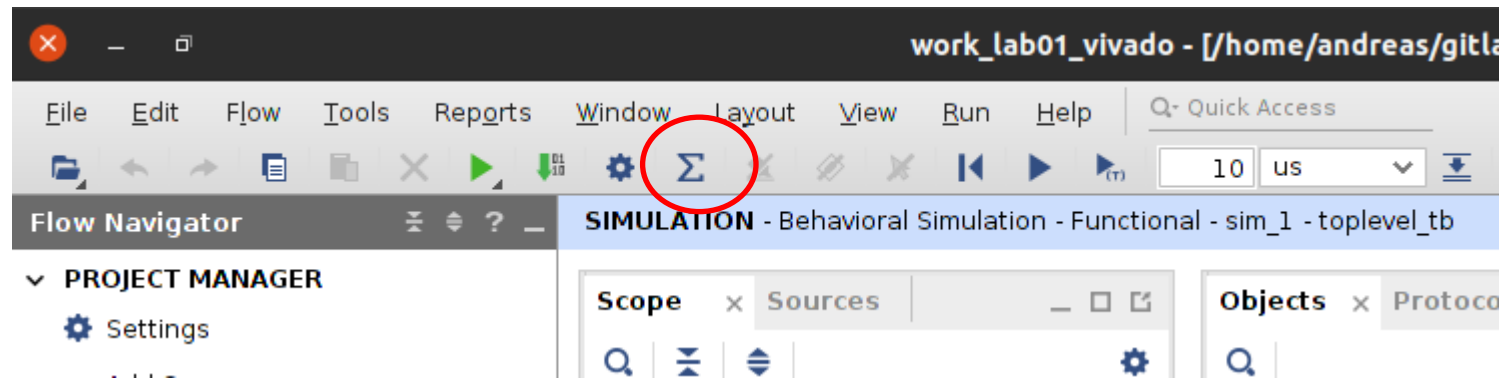
```
INFO: [Project 1-571] Translating synthesized netlist
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 . Memory (MB): peak = 2403.559 ; gain = 0.000 ; free physical = 1681 ; free virtual = 10380
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 . Memory (MB): peak = 2403.559 ; gain = 0.000 ; free physical = 1627 ; free virtual = 10323
INFO: [Project 1-111] Unisim Transformation Summary:
No Unisim elements were transformed.

INFO: [Common 17-83] Releasing license: Synthesis
15 Infos, 0 Warnings, 0 Critical Warnings and 0 Errors encountered.
synth_design completed successfully
synth_design: Time (s): cpu = 00:00:24 ; elapsed = 00:00:21 . Memory (MB): peak = 2403.559 ; gain = 91.812 ; free physical = 1765 ; free virtual = 10460
INFO: [Common 17-136] The checkpoint '/home/andreas/gitlab/dsd/AS21/labs/lab01_vivado/handout/lab/work_lab01_vivado/work_lab01_vivado.runs/synth_1/toplevel.dcp' has been generated.
INFO: [runtcl-4] Executing : report_utilization -file top_level_utilization_synth.rpt -pb top_level_utilization_synth.pb
INFO: [Common 17-206] Exiting Vivado at Mon Sep 27 07:12:49 2021...
```

- The synthesis will take about a minute. You can follow along in the process by clicking on the **Log** and **Synthesis** tabs at the bottom of the screen



- After synthesis is done you will be greeted by this window. Press **Cancel**



- We will now take a look at the project summary (highlighted above).

**Task:** Before going to the project summary, how many LUTs and I/O ports do you estimate that the design takes up and why?

The screenshot shows the Vivado 2020.2 interface during a simulation. The top bar indicates the project path and the current simulation stage: "SIMULATION - Behavioral Simulation - Functional - sim\_1 - toplevel\_tb". The "Flow Navigator" on the left shows the project structure, with "SIMULATION" selected. The main workspace is divided into several panels:

- Scope:** Shows the current design entity and its components.
- Obj Prot:** A table showing object values and delays.
- Overview | Dashboard:** Provides a summary of the board part and synthesis/implementation status.
- Utilization:** A table showing resource utilization for LUT and IO.
- Tcl Console:** Displays the output of the synthesis process.

The "Board Part" section shows details for the "pynq-z2" board, including its part number and revision. The "Synthesis" section indicates that the synthesis is complete with no errors or warnings. The "Implementation" section shows that the implementation has not yet started. The "Utilization" table is highlighted, and the "Table" link is circled in red.

Resource	Estimation	Available	Utilization %
LUT	2	53200	0.01
IO	6	125	4.80

The "Tcl Console" shows the following output:

```
INFO: [Project 1-571] Translating synthesized netlist
Netlist sorting complete. Time (s): cpu = 00:00:00 ; elapsed = 00:00:00 . Memory (MB): peak = 2403.559 ; gain = 0.000 ; free physical = 1681 ; free virtual = 10380
```

- Press on **Table** under the utilization view (after scrolling down a bit), here you see the resource utilization

# Tutorial: Part 7 - Implementation

- We now consider the implementation. This step takes the generated netlist from the synthesis step along with design constraints and implements it on the FPGA.
- This step comprises three main parts that can be carried out individually but are usually all run at the same time
  - **Linking:** The various input netlists from synthesis and design constraints are merged
  - **Placement:** Places the netlist elements onto the FPGA and optimizes the design using the constraints
  - **Routing:** Routes the various components of the design together

Note: The various warnings you will see during this example can be safely ignored. They relate to missing clock information, which is not added as the design is purely combinational

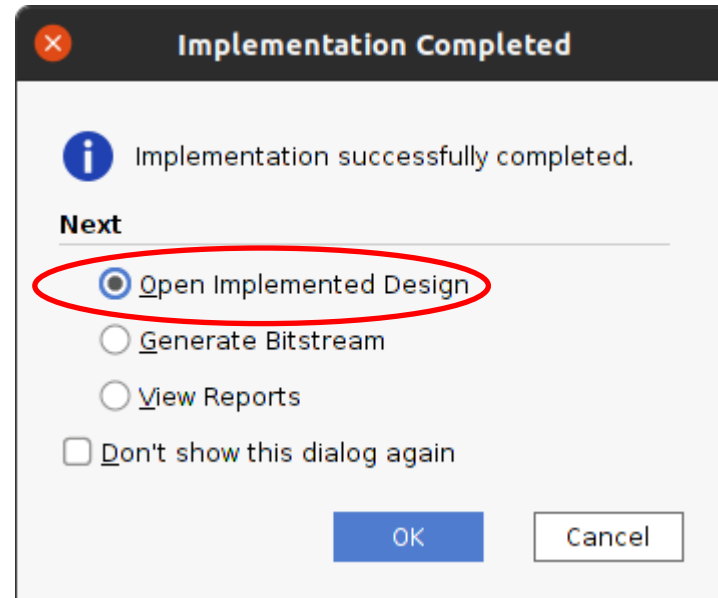
The screenshot shows the Vivado 2020.2 IDE interface. The Flow Navigator on the left has the 'Run Implementation' button highlighted with a red circle. The main window displays the 'Board Part' overview, synthesis status (Complete), and implementation status (Not started). A table shows resource utilization for LUT and IO.

Resource	Estimation	Available	Utilization %
LUT	2	53200	0.01
IO	6	125	4.80

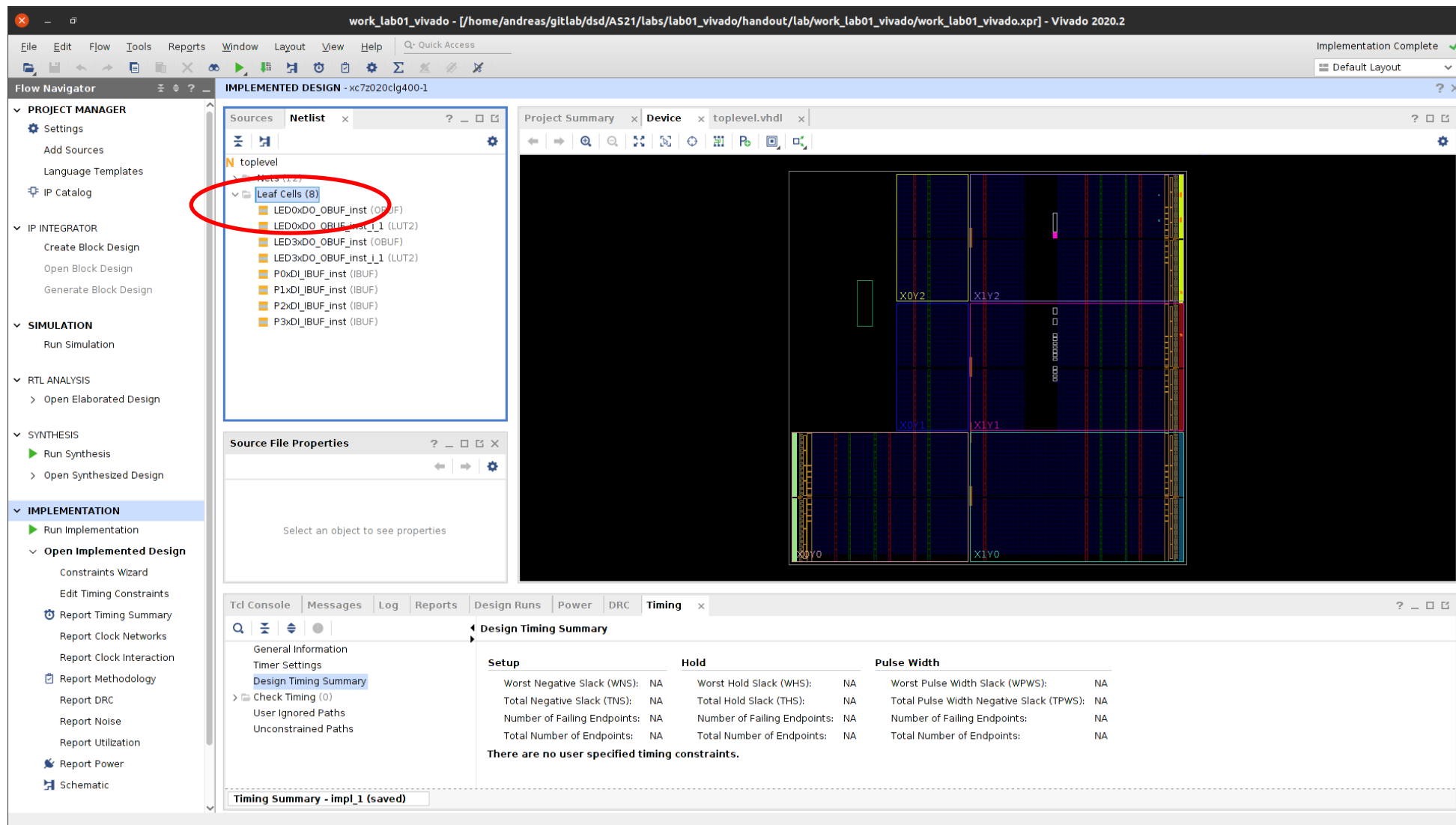
- Select **Run Implementation** from the Flow Navigator window. In the window that opens **change the number of jobs to 1** (if using servers) and press **OK**.

The screenshot shows the Vivado 2020.2 IDE interface. The top panel indicates 'Implementation Complete'. The middle panel displays the 'Board Part' details for a PYNQ-Z2 board, including the display name, board part name, revision, connectors, repository path, and URL. The bottom panel shows the 'Log' and 'Implementation' tabs, with the 'Log' tab selected and displaying the implementation command and progress. The 'Implementation' tab is also highlighted with a red circle.

- The implementation will take about a minute. You can follow along in the process by clicking on the **Log** and **Implementation** tabs at the bottom of the screen



- After the implementation finishes, press **Open Implemented Design**



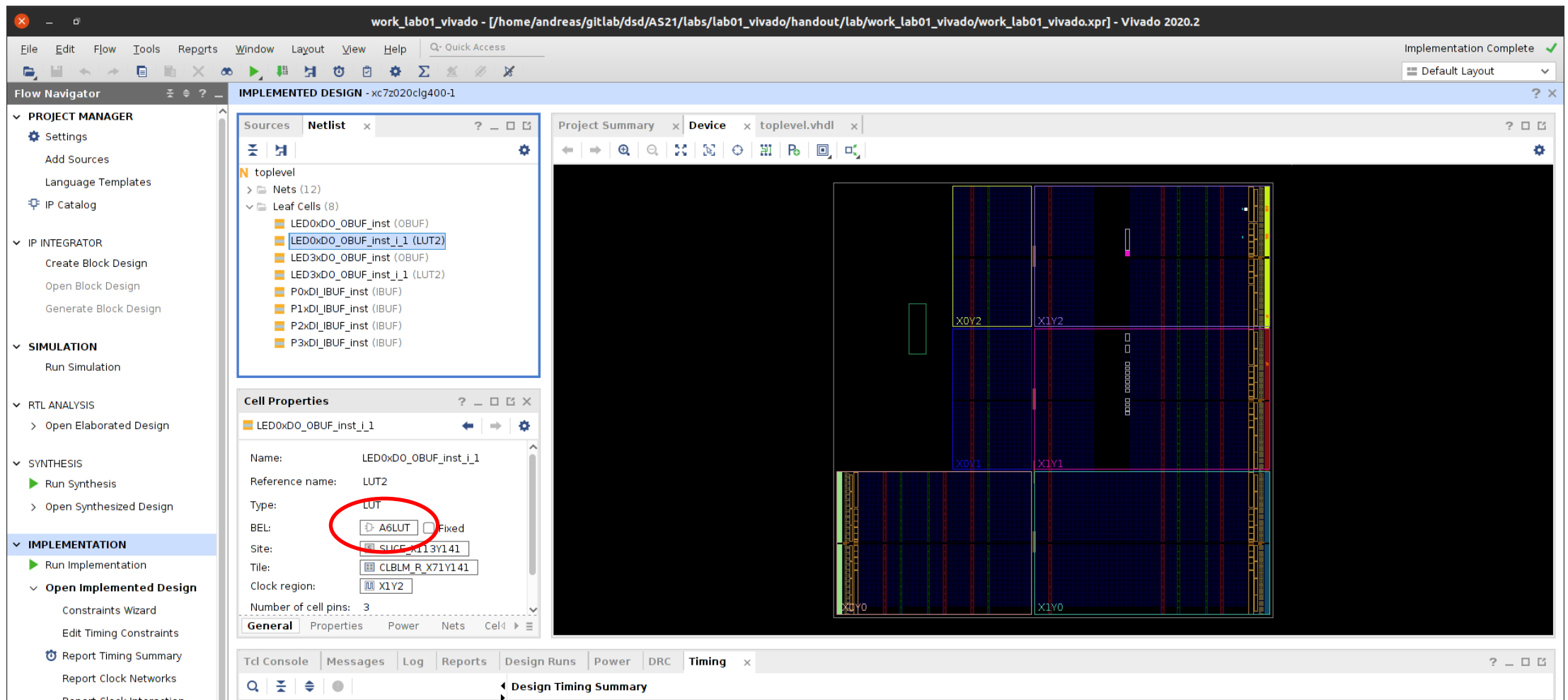
- The device view then shows how your design has been implemented on the FPGA. As this is a small design, you can only barely see it. Click on **Leaf Cells**

The screenshot displays the Vivado 2020.2 interface for an implementation design. The Netlist window shows a hierarchy of components, with the leaf cell `LED0xD0_OBUF_inst_i_1 (LUT2)` highlighted by a red circle. The Cell Properties window for this cell shows its name and reference name as `LUT2`. The Timing Summary window at the bottom indicates that there are no user-specified timing constraints.

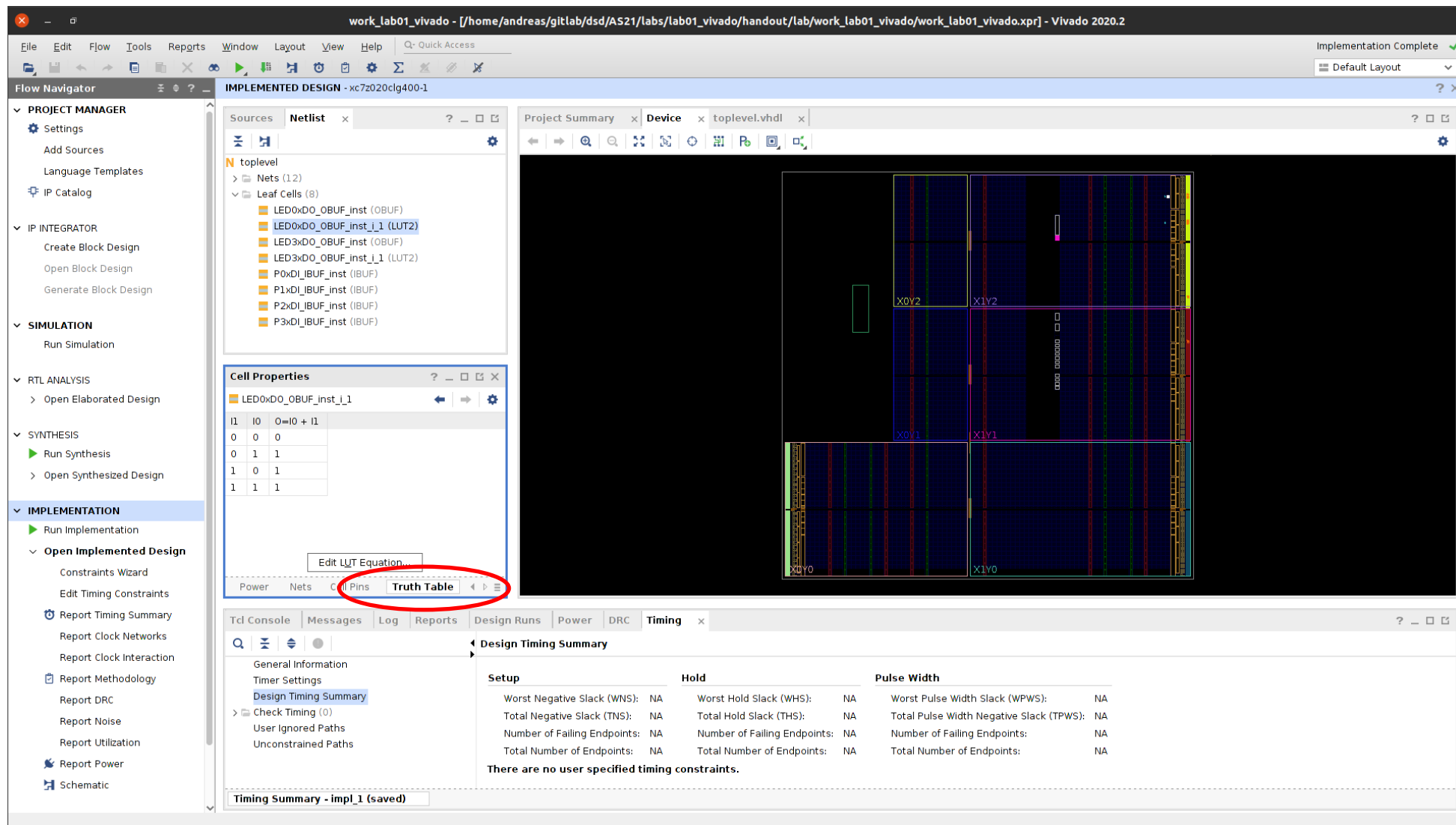
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	NA	Worst Hold Slack (WHS):	NA	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	NA	Total Hold Slack (THS):	NA	Total Pulse Width Negative Slack (TPWS):	NA
User Ignored Paths	NA	Number of Failing Endpoints:	NA	Number of Failing Endpoints:	NA
Total Number of Endpoints:	NA	Total Number of Endpoints:	NA	Total Number of Endpoints:	NA

There are no user specified timing constraints.

- Then click on the **LED0xD0\_OBUF\_inst\_i\_1 (LUT2)** leaf cell



- You can see the resource that this cell is taking up, in this case a 6-input LUT
- This is the result of the synthesis and implementation process! The OR gate has been mapped to a 6-input LUT, then placed at this site of the FPGA and its inputs/outputs routed



- As this is a LUT, we can see the truth-table implemented
- Navigate to Truth Table using the arrows in the Cell Properties window

The screenshot shows the Vivado 2020.2 IDE in the implementation stage. The Project Manager on the left is expanded to the IMPLEMENTATION section. The Netlist window shows the design hierarchy, including a LUT2 cell. The Cell Properties window displays the truth table for the selected LUT2 cell:

I1	I0	O=I0 + I1
0	0	0
0	1	1
1	0	1
1	1	1

The Device view shows a grid of logic cells with a red circle highlighting a specific cell in the upper right corner. The Design Timing Summary window at the bottom shows the following information:

Design Timing Summary			
Setup	Hold	Pulse Width	
Worst Negative Slack (WNS):	Worst Hold Slack (WHS):	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	Total Hold Slack (THS):	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	Number of Failing Endpoints:	Number of Failing Endpoints:	NA
Total Number of Endpoints:	Total Number of Endpoints:	Total Number of Endpoints:	NA

There are no user specified timing constraints.

- If we then consider the Device view again, it is possible to zoom in on your device, it will be slightly highlighted (may be hard to see)
- Here, it is in the upper right corner

The screenshot shows the Vivado 2020.2 interface. The main window displays a logic diagram of a slice, with a red circle highlighting a specific component. The left sidebar shows the Project Manager and Implementation sections. The bottom panel shows the Design Timing Summary report.

**Design Timing Summary**

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): NA	Worst Hold Slack (WHS): NA	Worst Pulse Width Slack (WPWS): NA
Total Negative Slack (TNS): NA	Total Hold Slack (THS): NA	Total Pulse Width Negative Slack (TPWS): NA
Number of Falling Endpoints: NA	Number of Falling Endpoints: NA	Number of Falling Endpoints: NA
Total Number of Endpoints: NA	Total Number of Endpoints: NA	Total Number of Endpoints: NA

- Here we then see the LUT. Note how the resources (LUTs, MUXs, FFs and carry-chains are grouped together.
- This is called a **Slice** on Xilinx FPGAs.

The screenshot shows the Vivado 2020.2 interface. On the left, the 'IMPLEMENTATION' section is expanded, and 'Schematic' is highlighted. In the center, the 'Netlist' tab is active, showing a tree view with 'toplevel' selected. The 'Netlist Properties' window shows 'Primitive Statistics' with LUT count 2 and IO count 6, and 'Net Boundary' with 6 boundary-crossing nets. The 'Design Timing Summary' window is open at the bottom, showing a table with columns for Setup, Hold, and Pulse Width, all with 'NA' values. The main workspace displays a schematic diagram of the design, showing logic components and their interconnections.

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): NA	Worst Hold Slack (WHS): NA	Worst Pulse Width Slack (WPWS): NA
Total Negative Slack (TNS): NA	Total Hold Slack (THS): NA	Total Pulse Width Negative Slack (TPWS): NA
Number of Falling Endpoints: NA	Number of Falling Endpoints: NA	Number of Falling Endpoints: NA
Total Number of Endpoints: NA	Total Number of Endpoints: NA	Total Number of Endpoints: NA

- Another option of looking at your design is to see the schematic.
- **Click on toplevel in the Netlist tab and then click on Schematic under Implementation**

The screenshot displays the Vivado 2020.2 IDE interface for an implemented design. The main window shows the Schematic view of the design, which includes several input buffers (IBUF) connected to LUT2s, which are then connected to output buffers (OBUF) for the LEDs. The Design Timing Summary window at the bottom shows that there are no user-specified timing constraints.

**Netlist Properties**

Primitive type	Count
LUT	2
IO	6

**Net Boundary**

Boundary-crossing Nets
6

**Design Timing Summary**

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): NA	Worst Hold Slack (WHS): NA	Worst Pulse Width Slack (WPWS): NA
Total Negative Slack (TNS): NA	Total Hold Slack (THS): NA	Total Pulse Width Negative Slack (TPWS): NA
Number of Falling Endpoints: NA	Number of Falling Endpoints: NA	Number of Falling Endpoints: NA
Total Number of Endpoints: NA	Total Number of Endpoints: NA	Total Number of Endpoints: NA

There are no user specified timing constraints.

- Then, we see the schematic of our design, with the LUTs

# Tutorial: Part 8 – Bitstream Generation

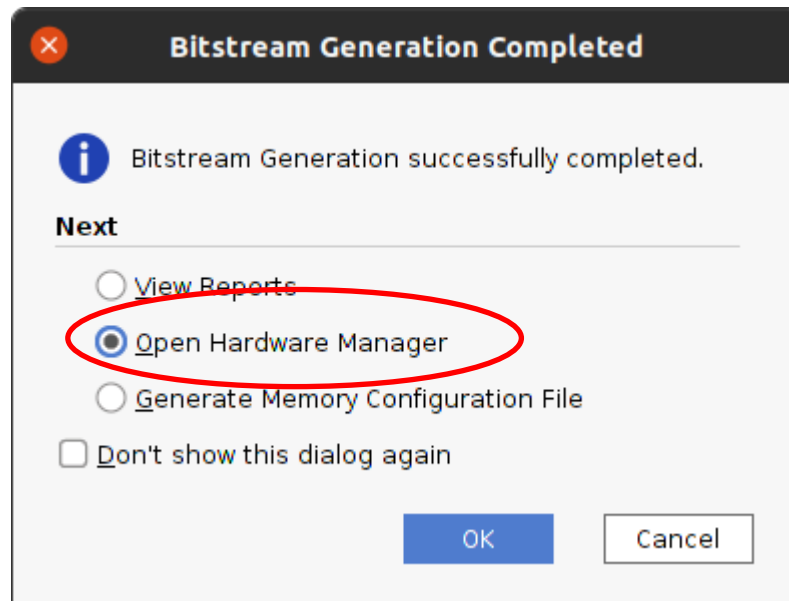
- We now consider the bitstream generation step and the programming

The screenshot shows the Vivado 2020.2 IDE interface during the implementation phase. The Flow Navigator on the left has 'Generate Bitstream' circled in red. The main window displays a schematic diagram with LUTs and IOBs. The Timing Summary window at the bottom shows a table of timing metrics.

Setup			Hold			Pulse Width		
Worst Negative Slack (WNS):	NA		Worst Hold Slack (WHS):	NA		Worst Pulse Width Slack (WPWS):	NA	
Total Negative Slack (TNS):	NA		Total Hold Slack (THS):	NA		Total Pulse Width Negative Slack (TPWS):	NA	
Number of Falling Endpoints:	NA		Number of Falling Endpoints:	NA		Number of Falling Endpoints:	NA	
Total Number of Endpoints:	NA		Total Number of Endpoints:	NA		Total Number of Endpoints:	NA	

There are no user specified timing constraints.

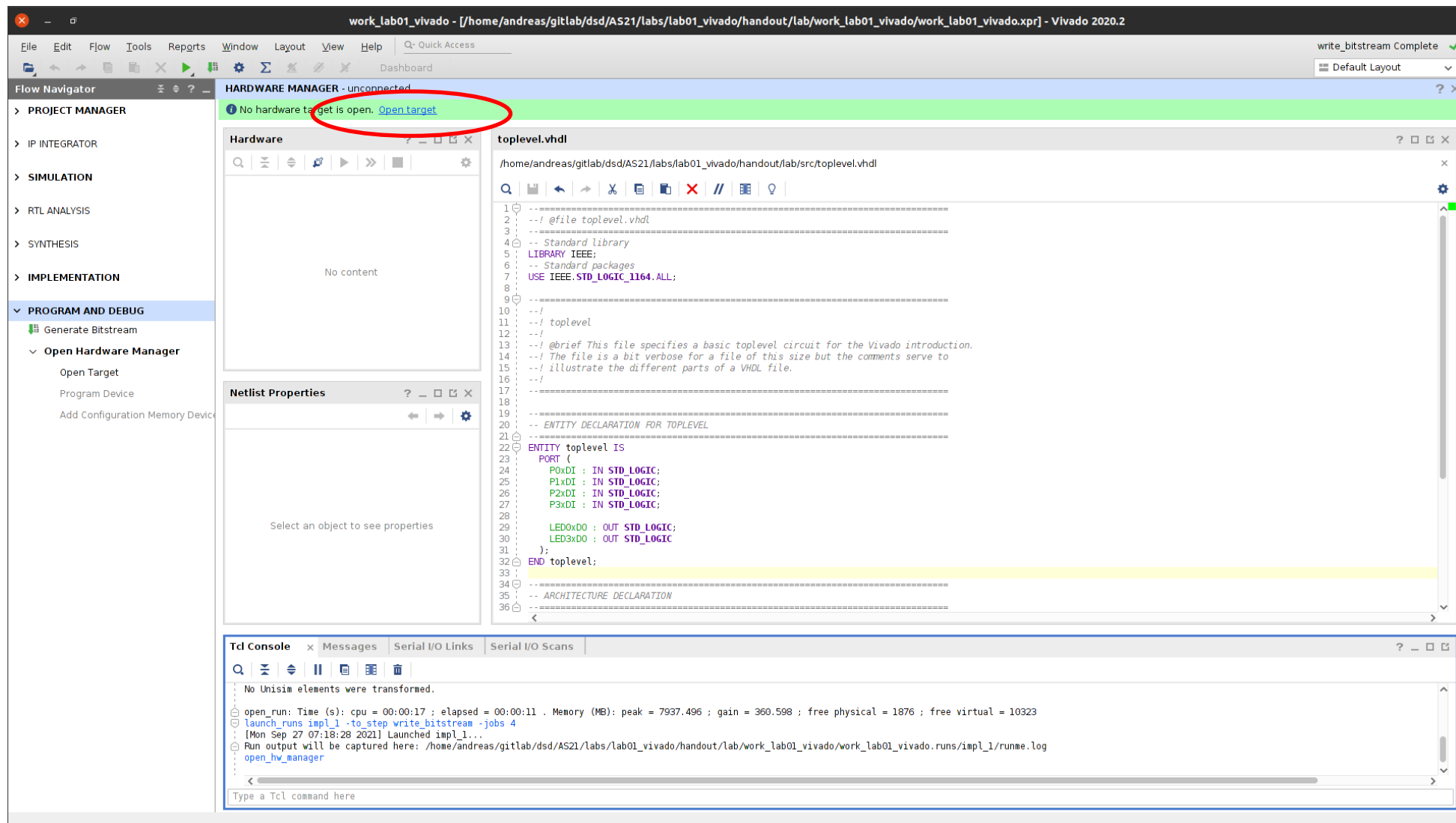
- Select **Generate Bitstream** from the Flow Navigator window. In the window that opens **change the number of jobs to 1** (if on servers) and press **OK**



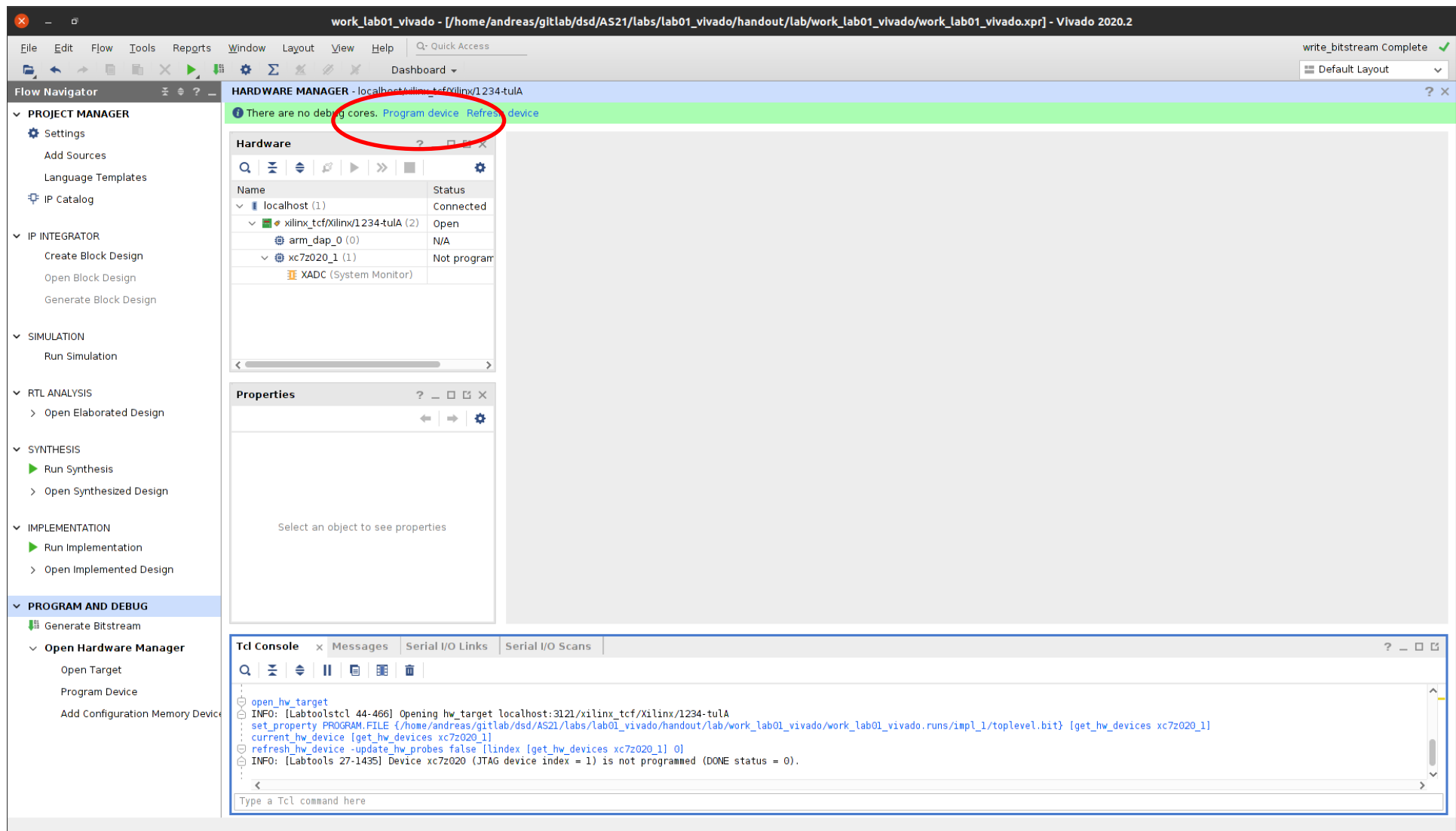
- **If you have a local installation**, you can then **press Open Hardware Manager**
- **If you do not have a local installation**, go to the next part after you have copied the bit-file:
  1. In the Ubuntu VM (not the Linux servers) or your own Linux computer open a new terminal window
  2. Use scp to copy the toplevel.bit file to the current directory of the terminal (current=dot)

```
scp edauserX@se1srv2.epfl.ch:dsd22/lab01_vivado_handout/lab/work_lab01_vivado/work_lab01_vivado.runs/impl_1/toplevel.bit .
```

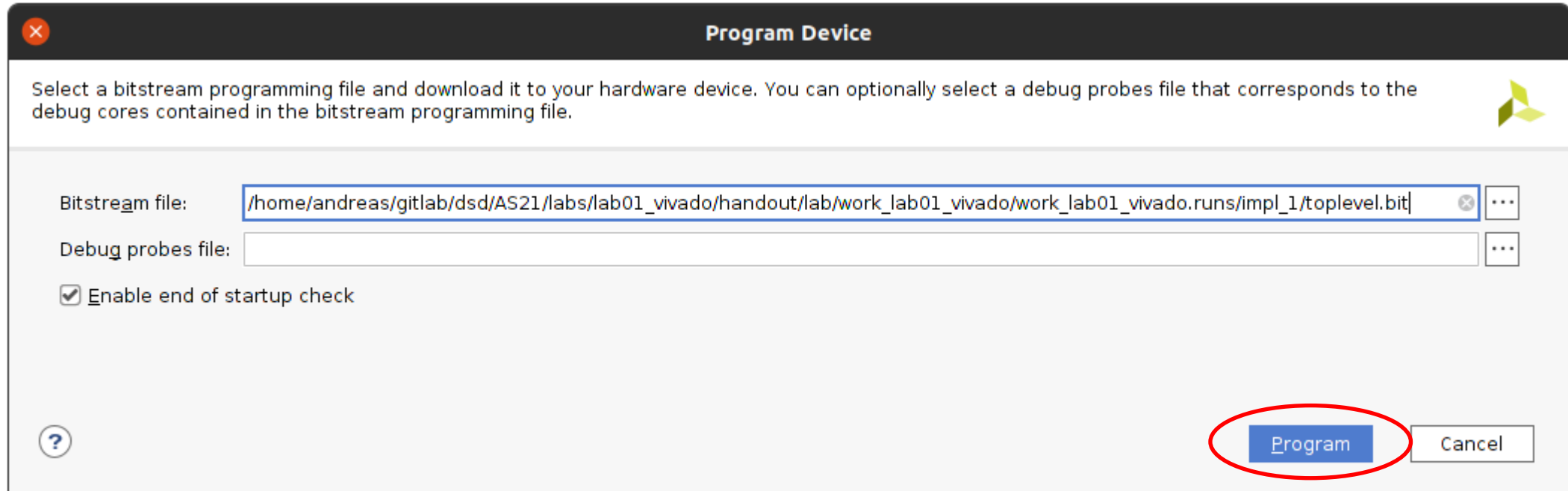
- Then put the toplevel.bit file on a computer which has Vivado installed for programming the FPGA
- You can read more about scp for Linux [here](#)
- If your host computer is a **Windows machine**, use **WinSCP** to copy the file



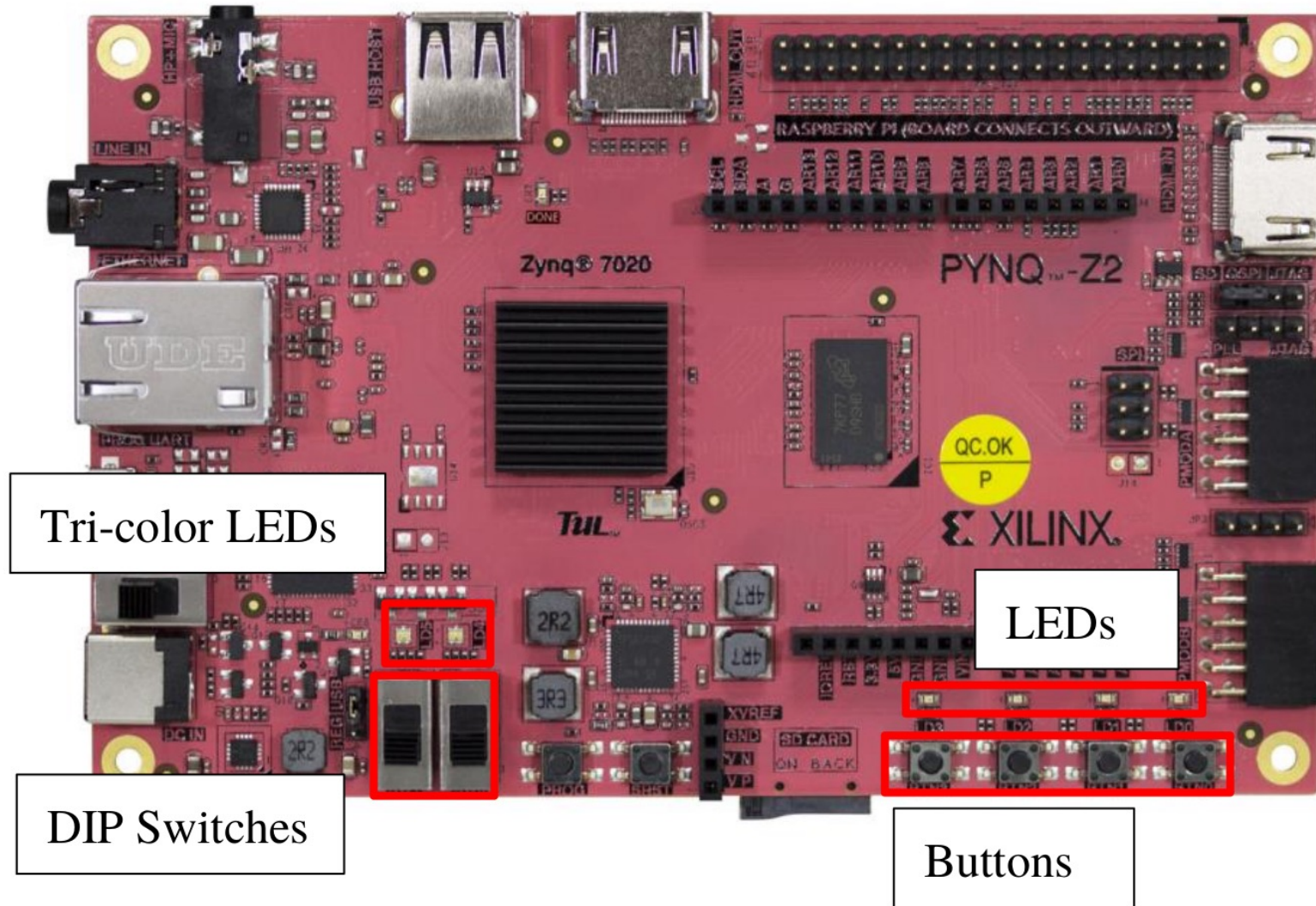
- Connect your board to your computer using the provided USB-cable
- Then, click on **Open target --> Auto Connect**



- Now that we have connected to the FPGA, we see the FPGA
- Press on **Program Device**



- Click **program**
- Note the path of the bitstream file here, in case you are running Vivado on the servers and you don't know where the bitstream file is and need to download it for locally programming the FPGA

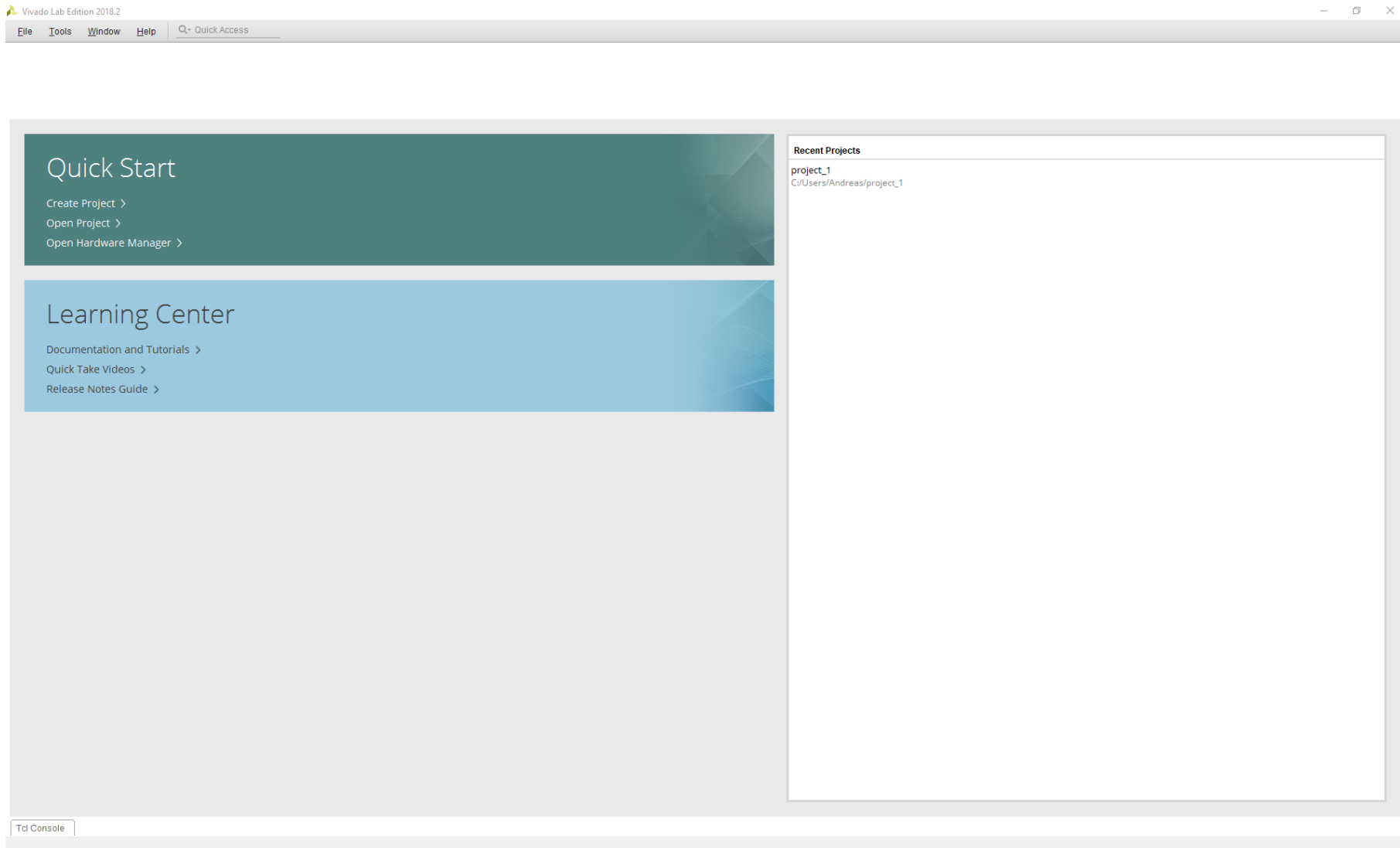


- You can then press the buttons on the board to see the LEDs lighting up based on the implemented logic

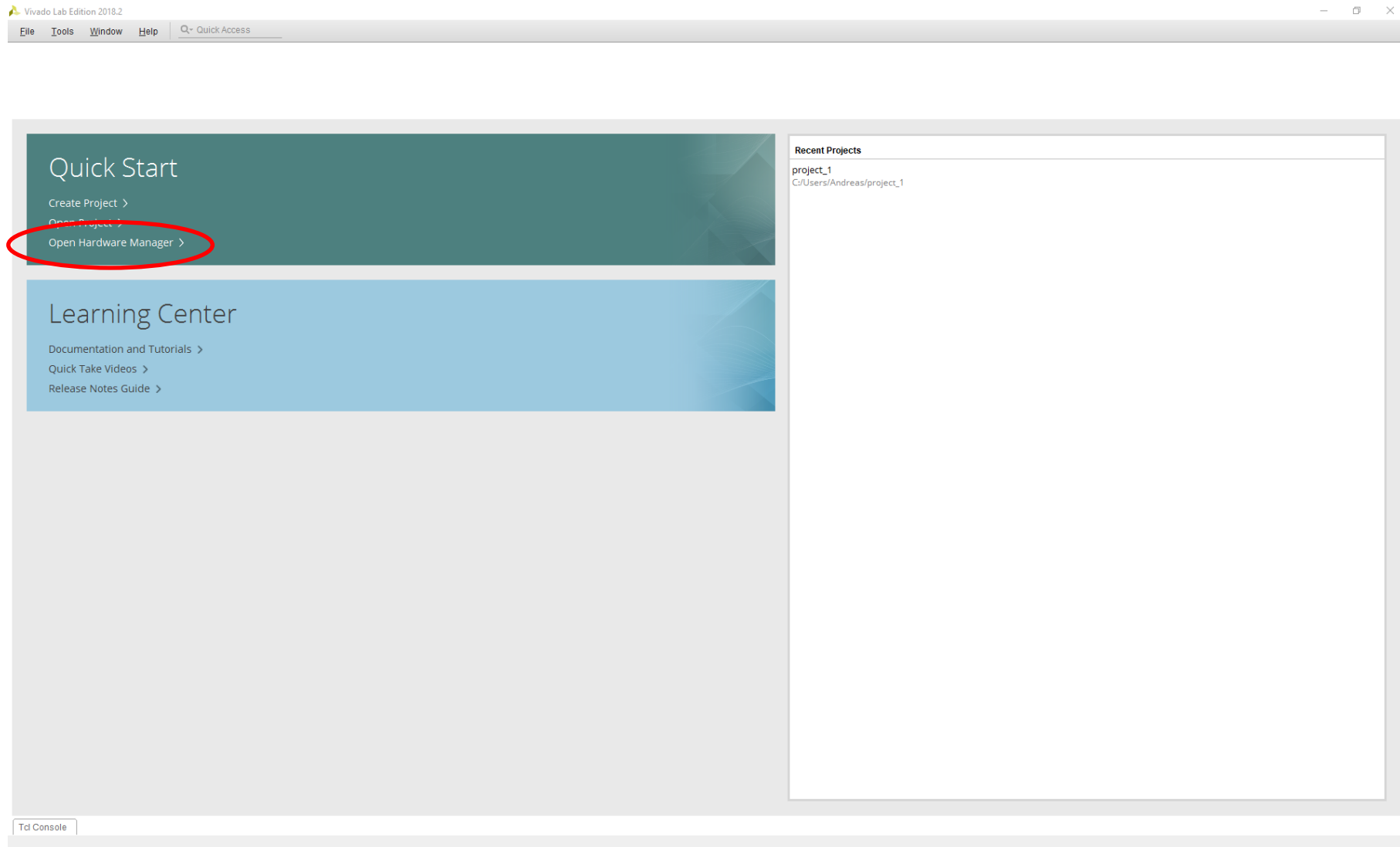
# Tutorial: Part 9 – Programming With Lab Edition

- If you do not have a local installation, you can generate the bit-file on the Linux server by following part 8 and then simply copy the bit-file to your local machine.
- This consists of the following steps:
  1. Generate the bit-file (see part 8)
  2. Copy the bit-file from the server at `work_lab01_vivado/work_lab01_vivado.runs/impl_1/toplevel.bit` to your local machine using, e.g., WinSCP (on Windows) or `scp` (on Linux). For this, you will use `selsrv1` or `selsrv2` as the host name and your username as the user name of the tool.
  3. Install the Lab Edition of the Vivado Design Suite. This version takes up considerably less space than the full edition.

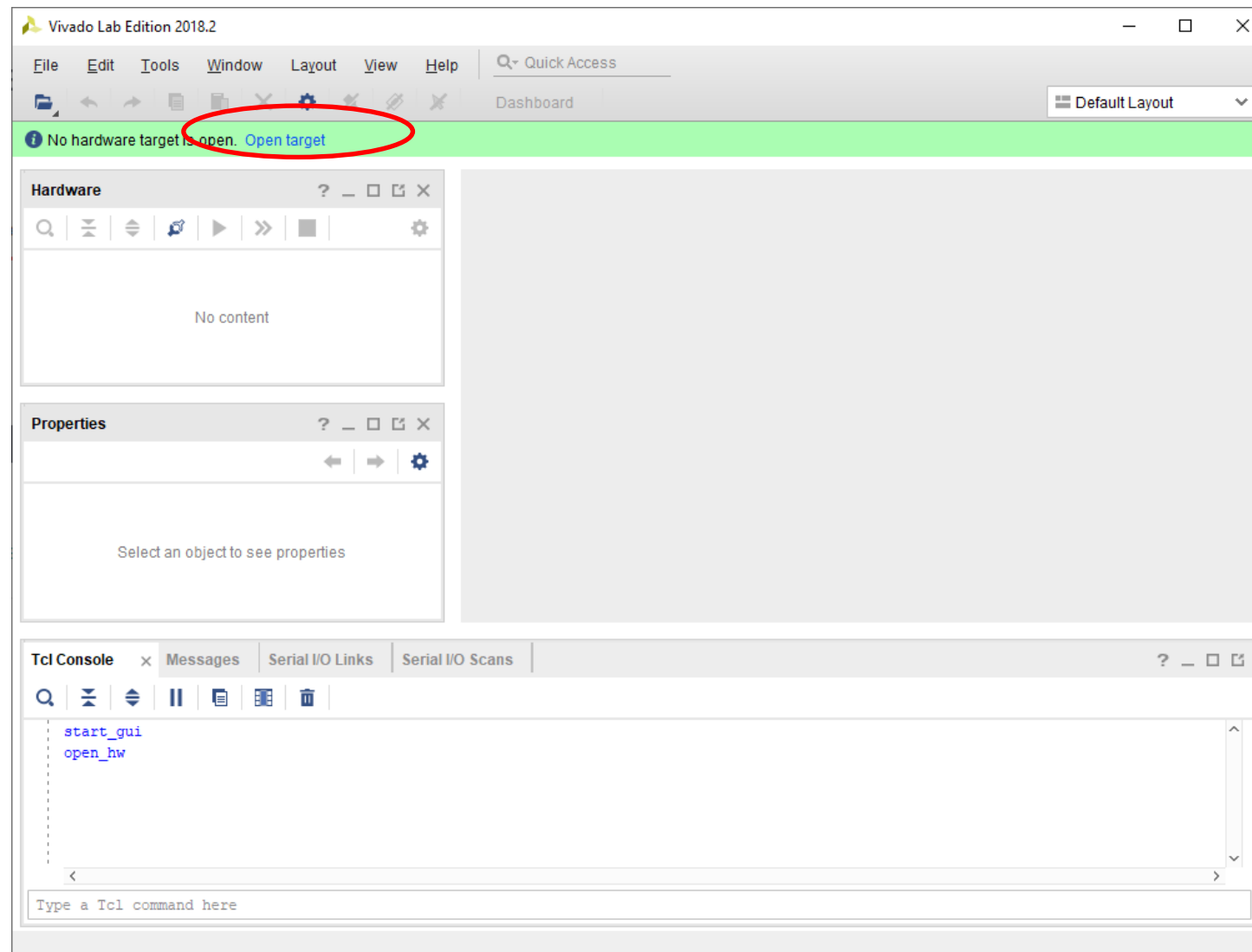
To continue to the next part of the tutorial, please first complete the installation of the Vivado Lab Edition. This part of the tutorial uses an installation of the Vivado 2018.2 lab edition on Windows as an example, but the steps are the same between Windows/Linux and for newer versions of Vivado



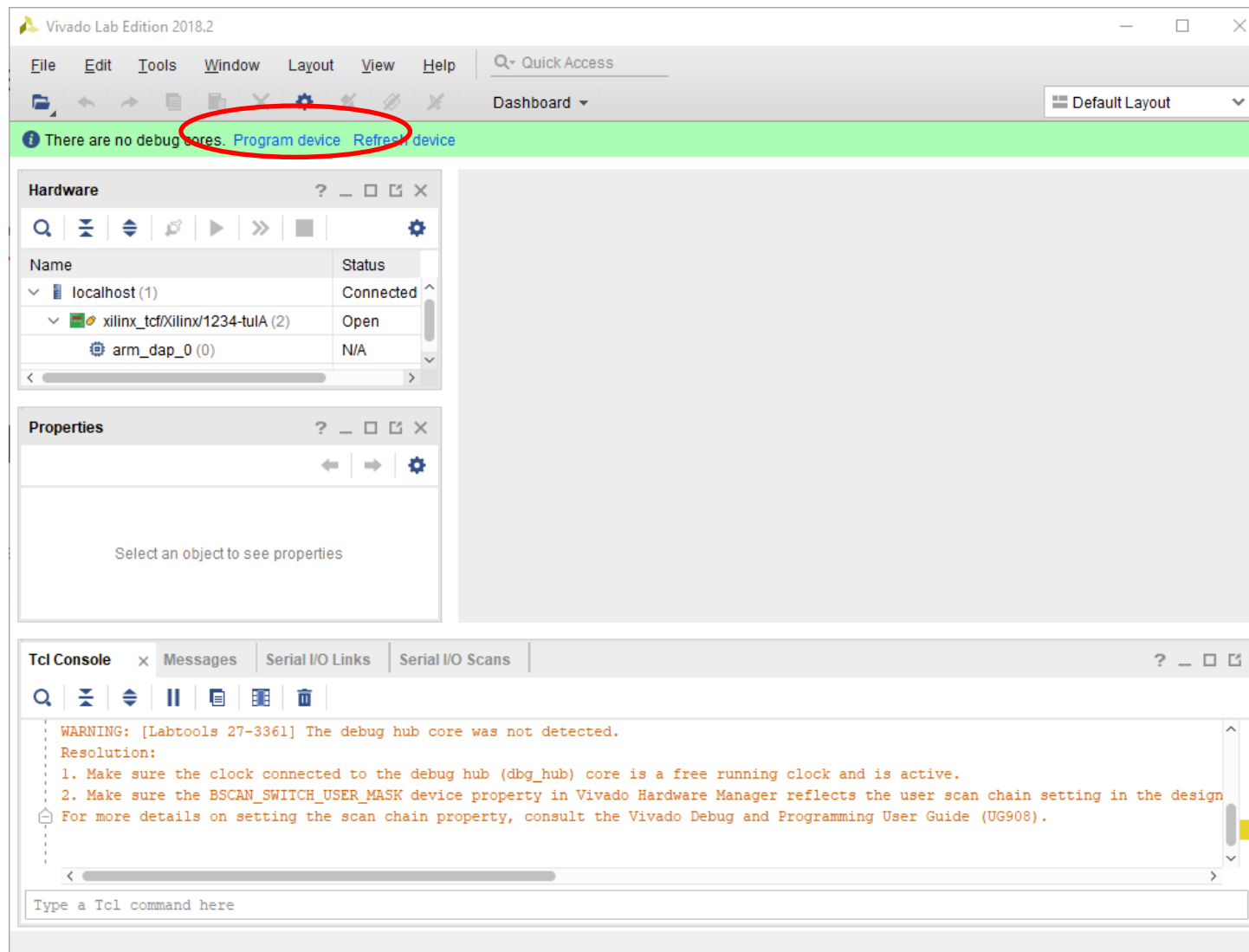
- Open Vivado Lab Edition



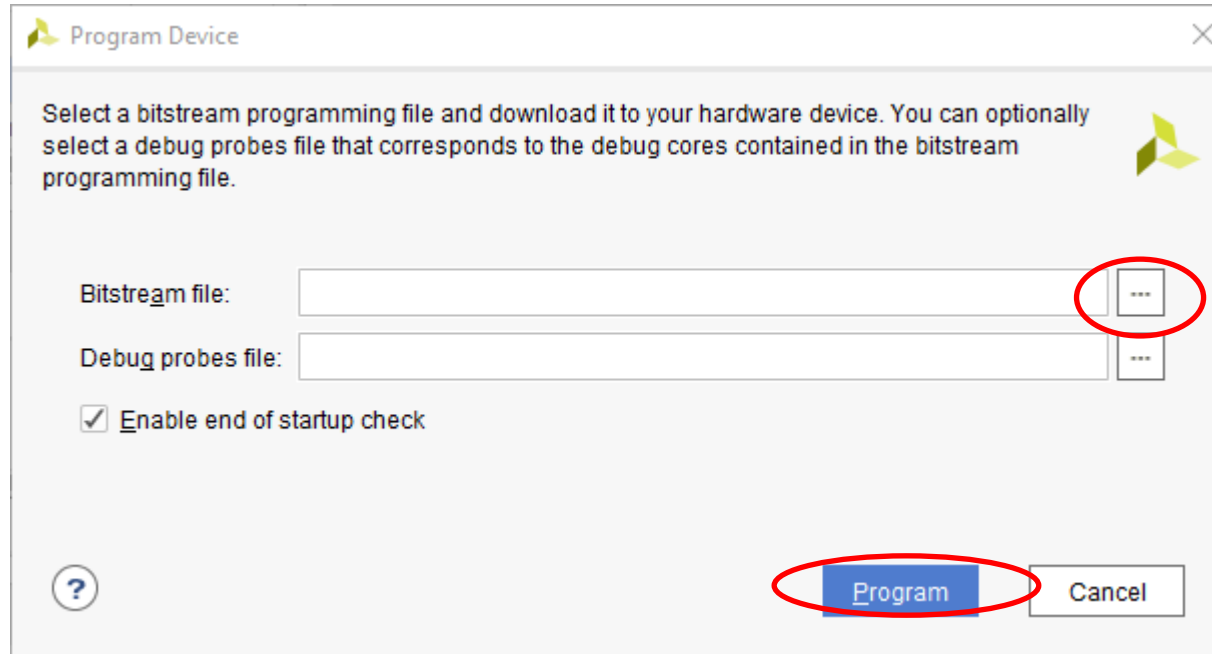
- Click on **Open Hardware Manager**



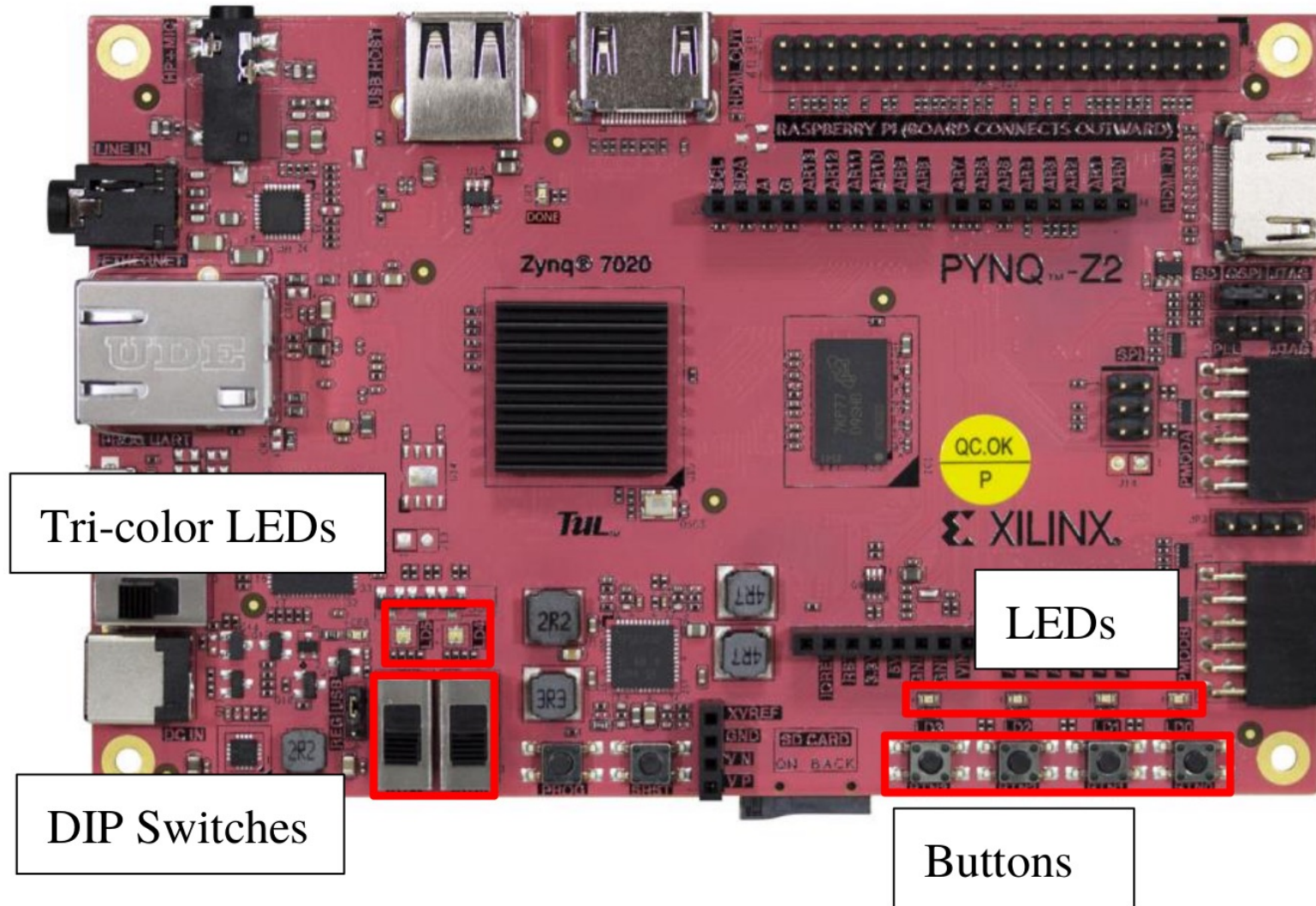
- Connect the board with the USB cable
- Then press **Open target --> autoconnect**



- Press Program Device



- In this window, you then have to **point to the bit-file that you copied before by clicking on the indicated three dots ....**
- Then, **press Program** once you have added the bit-file



- You can then press the buttons on the board to see the LEDs lighting up based on the implemented logic

# Taking care of your FPGA board

- Congratulations! You have now completed the tutorial
- Now that you're packing your FPGA board away **please always remember to**
  - **Put the FPGA board in the box** and ideally in the wrapping (if you have wrapping)
  - Please do not lose your cables or other provided equipment
- **Please do not lose the FPGA or take it with you if you leave EPFL after the semester, we check at the end of the semester if all FPGAs have been returned!**