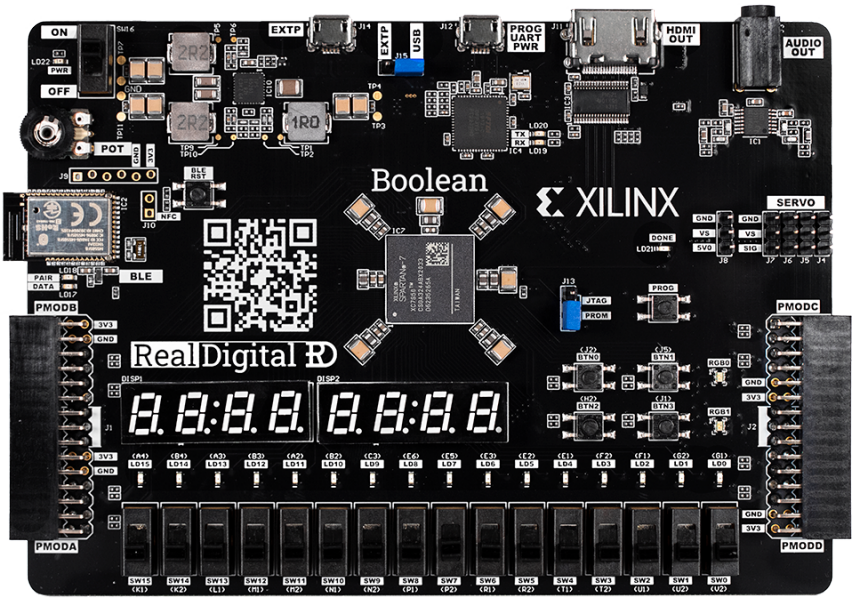
Digital Electronics – Lab 0

Tutorial

Kent Abrahamsson, Andreas Axelsson

2026-01-18



1 Revision history

Date	Author	Description
2023-03-26	Kent Abrahamsson	First edition
2023-04-01	Kent Abrahamsson	Minor typo fix
2026-01-15	Andreas Axelsson	From Altera to Xilinx

Contents

2 General

2.1 Background

This document is intended to be used as a tutorial / step by step guide on how to use AMD / Xilinx Vivado for developing a small digital design targeted for the Boolean evaluation board.

2.2 Intended outcome

The goal is to give the student a basic knowledge of all steps involved to design, build and download a small digital design into the development board (target hardware).

2.3 Development board

Through out the course we will use the AMD Boolean board sold by Real Digital as a development board. Below is a table with the key features of the board, as well as a link to their homepage with additional resources.

Boolean FPGA Board — Key Features

Core

- Xilinx Spartan-7 XC7S50-CSGA324 FPGA
- 16 MB QSPI flash

Connectivity

- USB (power, programming, UART/COM)
- Bluetooth Low Energy (BLE), **optional**
- HDMI source (up to 1080p)

User I/O

- 8-digit seven-segment display
- 16 slide switches
- 4 pushbuttons
- 16 discrete LEDs
- 2 RGB LEDs

Peripherals

- PWM audio output
- Four servo motor outputs
- On-board ADC with thumbwheel potentiometer

Expansion

- 2 Pmod+ connectors (4 Pmods)

Board documentation:
`https://www.realdigital.org/hardware/boolean`

2.4 Vivado software

Vivado is the primary development environment from AMD Xilinx for designing, implementing, and programming FPGA-based systems. It provides a complete toolchain for transforming a hardware description—written in VHDL, Verilog, or SystemVerilog—into a configured FPGA that performs a specific digital function.

In practice, Vivado takes a design through the full hardware workflow:

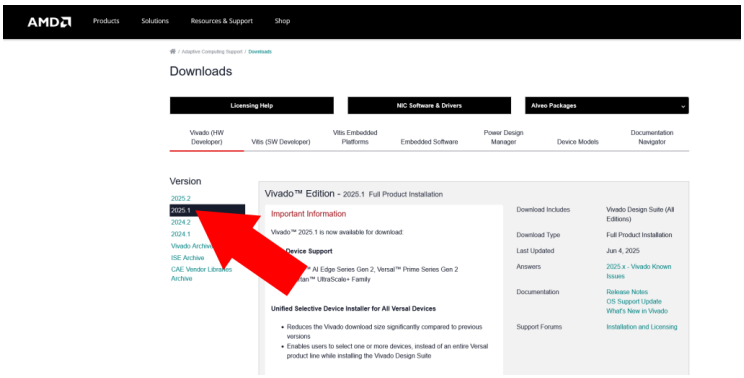
- **Design entry** — write HDL code or create block-diagram designs.
- **Simulation** — verify functionality before hardware is involved.
- **Synthesis** — translate the design into FPGA logic resources.
- **Implementation** — place and route the logic on the target device.
- **Bitstream generation** — create the configuration file for the FPGA.
- **Programming and debugging** — download the design and debug it on real hardware.

3 Tutorial instructions

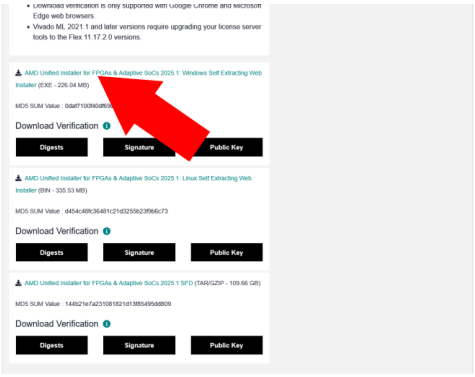
3.1 Installation of software (Windows)

The software is available for Windows 10 or later and Linux (only specific distributions and versions supported by AMD). **Note that the software is already installed on the computers in the computer lab, and the installation is only required if you intend to run on your own computer.**

Download Vivado 2025.1 from Xilinx's website: `https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2025-1.html`.

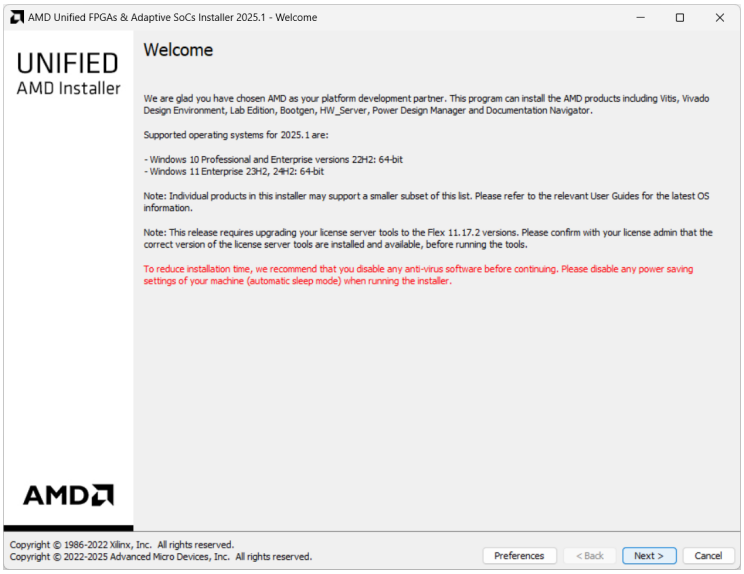


Download the "`AMD Unified Installer for FPGAs & Adaptive SoCs 2025.1: Windows Self Extracting Web Installer`".

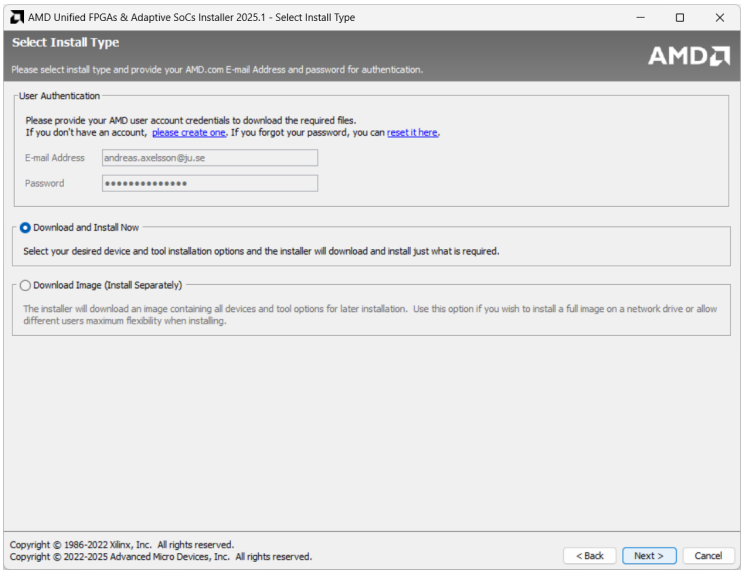


Log in or create an AMD account.

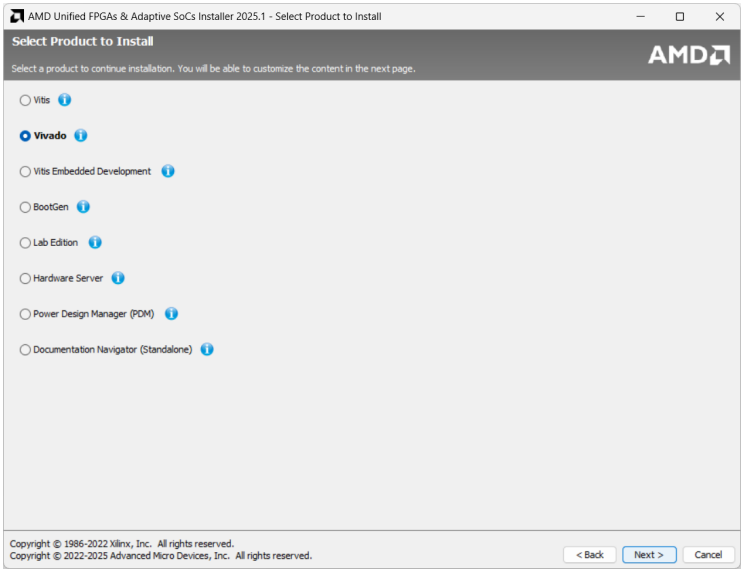
Once the file has been downloaded, run it.



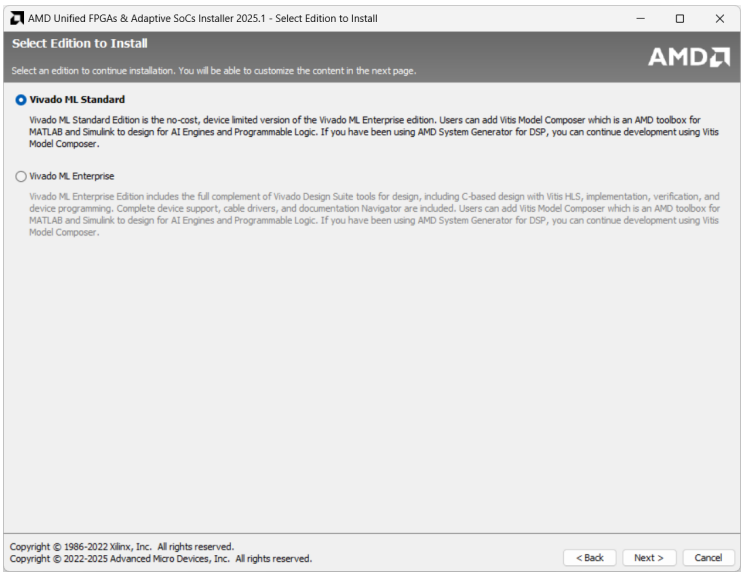
When asked, log in with your AMD account.



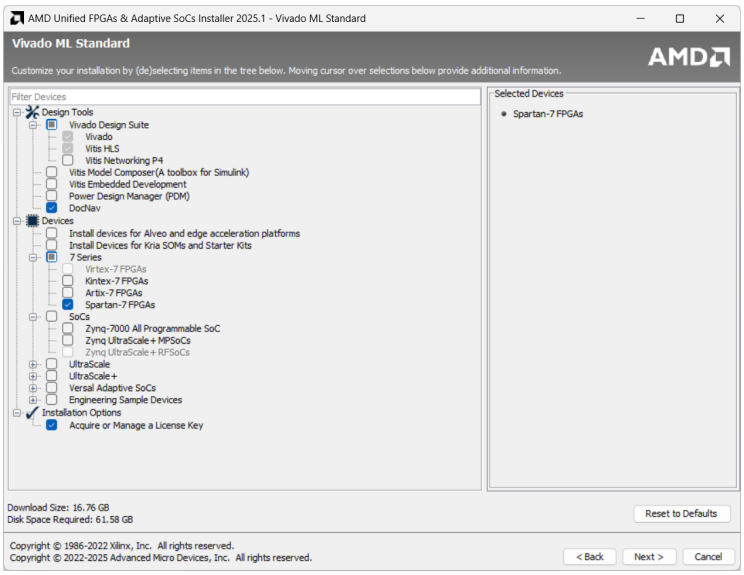
Select "`Vivado`".



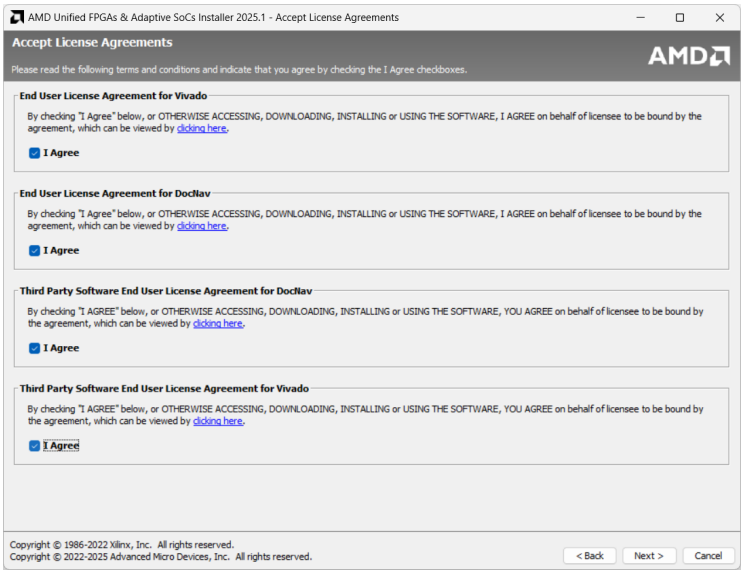
Select “Vivado ML Standard”.



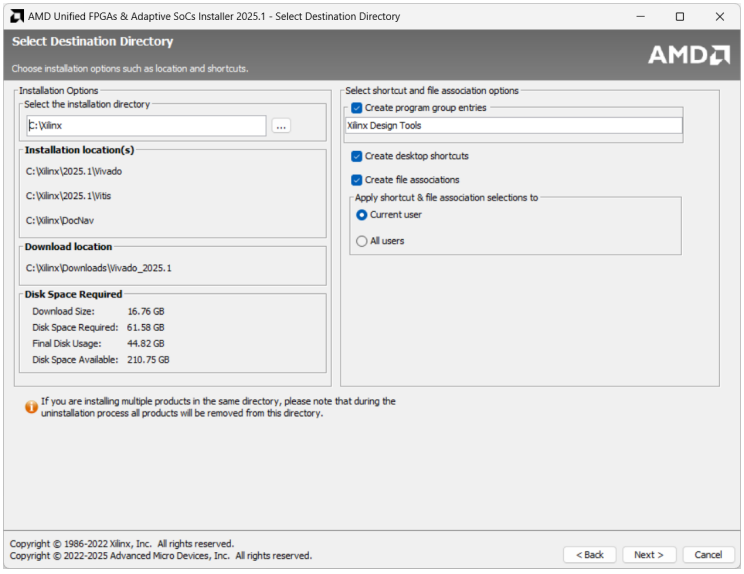
Deselect “Vitis Model Composer (A toolbox for Simulink)”, and select “Spartan-7 FPGAs” under “7 Series”.



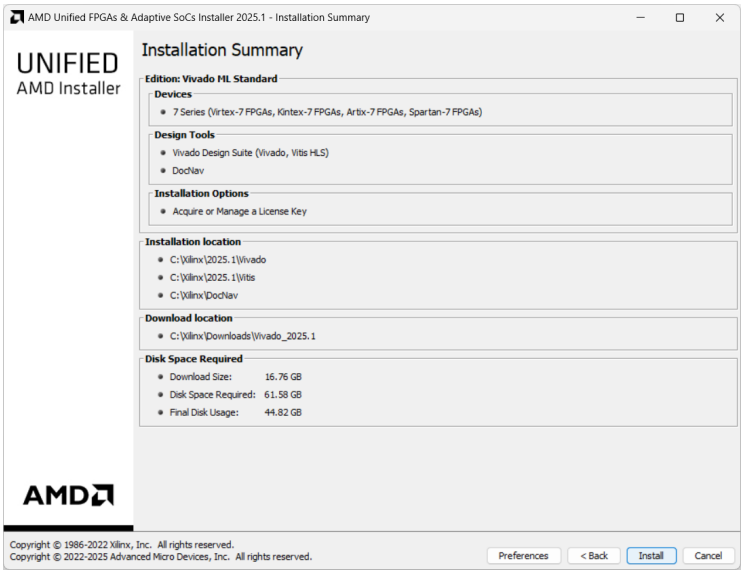
Carefully read the four agreements, and check the four boxes.



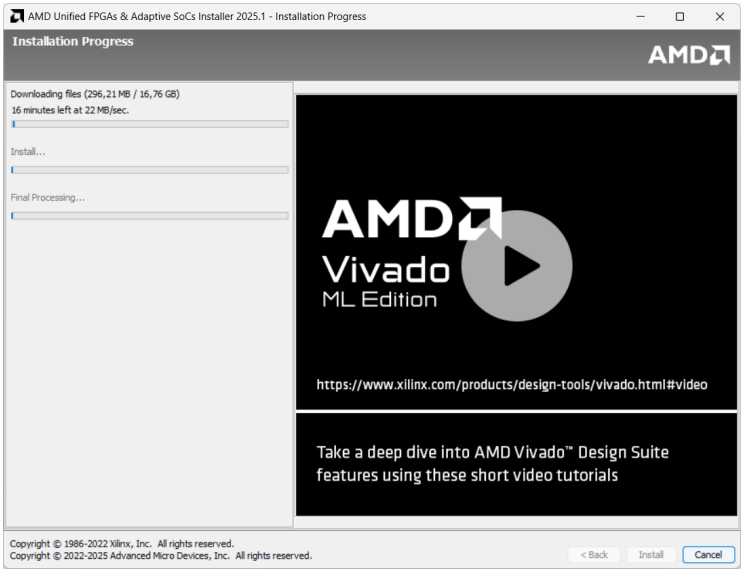
Use the default installation location.



Confirm that the installation options seem to be correct.



Wait for the download to complete. It can take a while, as it is ≈17GB of data to download.



If the License Manager appears, close it.

3.2 Start up the application

There are several ways to create a Vivado project. There is a wizard method that walk you through a process to set up a new project, and there is a script method that creates a project from a template. The script method is more general and open up for better project structure and version control, as the Vivado generated artifacts are stored in a separate folder. It is also the way it will be done for all the labs, where you will be presented a master project you can clone from the GitHub Classrom assignments. In appendix A you learn how a project can be created using the wizard.

Open start menu and start the Vivado application. As you can see in Figure 1 there are several options to choose from. You can see the *Create Project* and *Open Project* under the *Quick Start* section. If

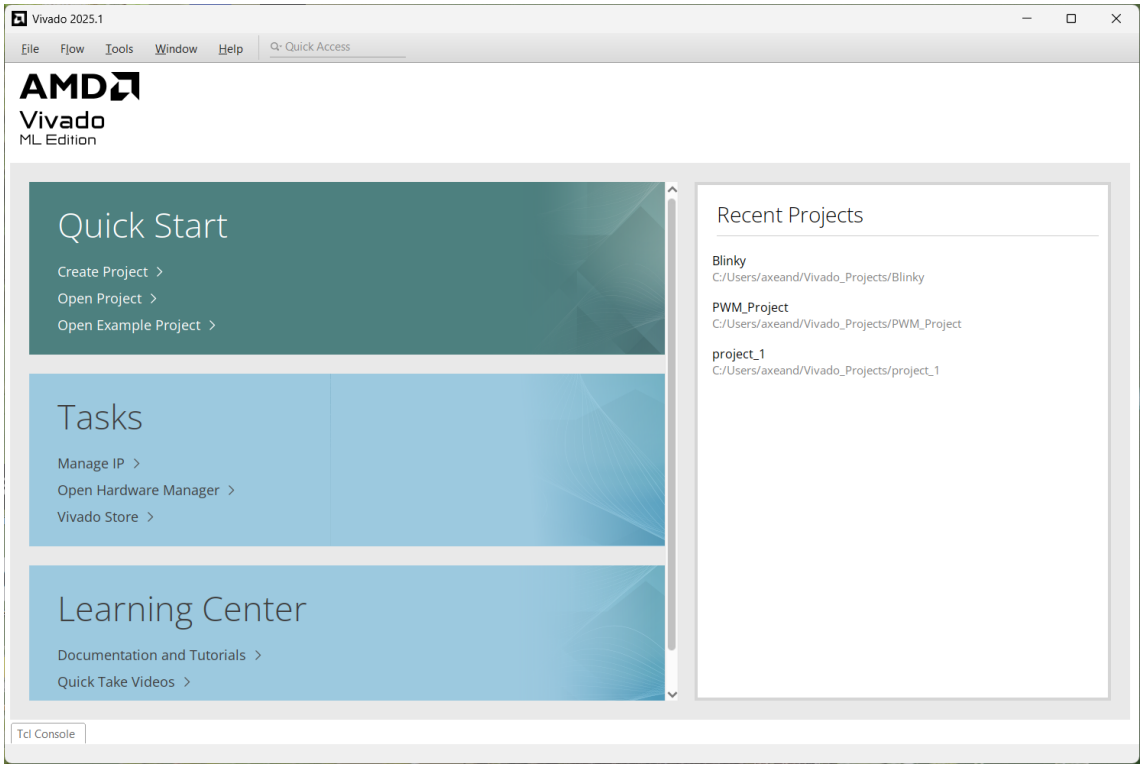


Figure 1: Start-up window.

you previously already created a project, it is likely you see it listed under *Recent Projects* as well. Then you can open it directly from there.

At the bottom of the startup window you can also see *Tcl console*. This consol will be used when we create a project from a template using scripts.

3.3 New Project using script method

One often wants to create a similar type of file structure for projects to keep your created files separated from all the Vivado generated files. This allows the project to be more easily maintained and version controlled. You typically only version control the files you create, and a script that can rebuild the Vivado project. Below is a typical project template structure with the different types of files capturing your design well separated into different sub-folder. This makes a clean structure that is easy to work with, especially when the project needs version control and/or team collaboration.

In the next sections there are two methods described on using a template project, 1) using the zip-file project template, and 2) using the repositories in GitHub Classroom. You shall use the GitHub Classroom method in the labs. Otherwise you might not be able to submit your labs correctly.

Project template structure

```
lab_template/
├── src/
│   └── top.vhd
├── tb/
│   └── tb_top.vhd
├── constraints/
│   └── board.xdc
├── scripts/
│   └── create_project.tcl
├── .gitignore
└── README.md
```

3.3.1 Zip-file project template

In Canvas there is a file `lab_template.zip` that can be downloaded and used as a template. The first step in creating a new project based on this template is to unzip the file into a suitable folder on the computer. Ensure it is not unzipped too deep into the disk folder structure, and that there is no spaces in the absolute path to the project folder as this will create problems for the tool chain used in Vivado.

Rename the template folder to the desired name before running the create-script.

3.3.2 GitHub Classroom project template

GitHub Classroom allows administration of assignments and result submissions for each assignment for groups or individuals. For each lab there is an assignment with a master repository the students can clone. It contains the in principle the same files as the template zip-file, but is customized for each assignment. The README.md is also different between the different assignments as it lists expected additional material or answers for the students to submit.

3.3.3 Run create project script

As we now have the project files on our disk, we need to initialize the Vivado project (create the Vivado specific files). This is done with the *create_project.tcl* script in the scripts folder.

To run tcl-scripts we can do this using the Tcl Console in the start-up window. See the lower left corner of Figure 1 and follow the steps below.

How to use

1. Start Vivado.
2. In the **Tcl Console**, execute:

   ```
   cd <path-to-lab_template>/scripts
   source create_project.tcl
   ```

3. The Vivado project is created in `lab_template/vivado/`.

3.4 Design and finish the project

3.4.1 General

The following actions needs to be done to finish the project design successfully.

- Add and design the VHDL description of the digital electrical design.
- Perform a logical simulation.

- Synthesize the design to create a netlist of the design.
- Make sure unused pins are set up correctly.
- Add pinout information.
- Place and Route the design.
- Create programming file and download to development board.
- Verify functionality in hardware.

Most of these actions are done using the process steps found in the *Flow Navigator* on the left side in the main window.

3.4.2 Add Sources

If you created the project using the GUI method, your project doesn't contain any VHDL source files or constraint files. Using Add Sources in the *Flow Navigator* we can bring up a wizard to either create a new source file or import an existing source file. The table below describes the different source types that can be added.

Source type	**Purpose and typical use**
Add or create design sources	Contains the main hardware design files (VHDL/Verilog). These files describe the actual logic that will be synthesized and implemented on the FPGA. This is where students write the functional behavior of their circuits.
Add or create constraints	Contains the constraint file (`.xdc`) that maps logical signals in the design to physical FPGA pins and defines timing requirements. These files ensure the design is correctly connected to the board hardware.
Add or create simulation sources	Contains testbenches and simulation-only files used to verify the design before programming the FPGA. These files are not synthesized and exist only to support functional and timing simulation.

Using the Add Sources wizard you can add a VHDL file called top.vhd. The wizard allows you to add an existing file, if you choose to copy from an existing vhd-file, or create a new file. The wizard will ask you for port inputs and outputs. This is a way to create an empty template to work from with predefined ports.

Edit the top.vhd that is created so it looks like this:

```
1  library ieee;
2      use ieee.std_logic_1164.all;
3
4  entity top_level is
5  port(
6      sw0 : in std_logic;
7      sw1 : in std_logic;
8      sw2 : in std_logic;
9      led : out std_logic
10 );
11 end entity top_level;
12
13 architecture rtl of top_level is
14 begin
15     led <= sw0 and (sw1 or sw2);
16 end architecture rtl;
```

Save the file when finished

Now use Add Sources to create a constraint file called `board.xdc` and add the following lines to it. They tie the ports to the correct pins during place and route.

```
# On-board Slide Switches
set_property -dict {PACKAGE_PIN V2 IOSTANDARD LVCMOS33} [get_ports {sw0}]
set_property -dict {PACKAGE_PIN U2 IOSTANDARD LVCMOS33} [get_ports {sw1}]
set_property -dict {PACKAGE_PIN U1 IOSTANDARD LVCMOS33} [get_ports {sw2}]

# On-board LEDs
set_property -dict {PACKAGE_PIN G1 IOSTANDARD LVCMOS33} [get_ports {led}]
```

The last thing we want to do is to create a simulation source file, or as it also is called a testbench. The testbench is standard vhdl and instantiates our device under test (DUT) and alters different signals to see how our DUT responds. Using Add Sources create the testbench tb_top.vhd and ensure the file contains the following code:

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

entity tb_top is
end entity;

architecture sim of tb_top is
  signal sw0           : std_logic;
  signal sw1           : std_logic;
  signal sw2           : std_logic;
  signal led           : std_logic;

begin

  uut: entity work.top
    port map (
      sw0  => sw0,
      sw1  => sw1,
      sw2  => sw2,
      led   => led
    );

  stim: process
  begin
    sw0 <= '0';
    sw1 <= '0';
    sw2 <= '0';
    wait for 20 ns;

    sw0 <= '1';
    wait for 20 ns;
    assert led = '0' report "LED state not as expected" severity failure;

    sw2 <= '1';
    wait for 20 ns;
    assert led = '1' report "LED state not as expected" severity failure;

    sw1 <= '1';
    wait for 20 ns;
    assert led = '1' report "LED state not as expected" severity failure;

    sw0 <= '0';
    wait for 20 ns;
    assert led = '0' report "LED state not as expected" severity failure;

    report "Simulation finished OK." severity note;
```

```
47      wait;
48    end process;
49  end architecture;
```

Note: For this example, these files are automatically included if you used the script method to create the new project.

3.5 Run Simulation

Now we have all the files we need for this example and are ready to do something with them.

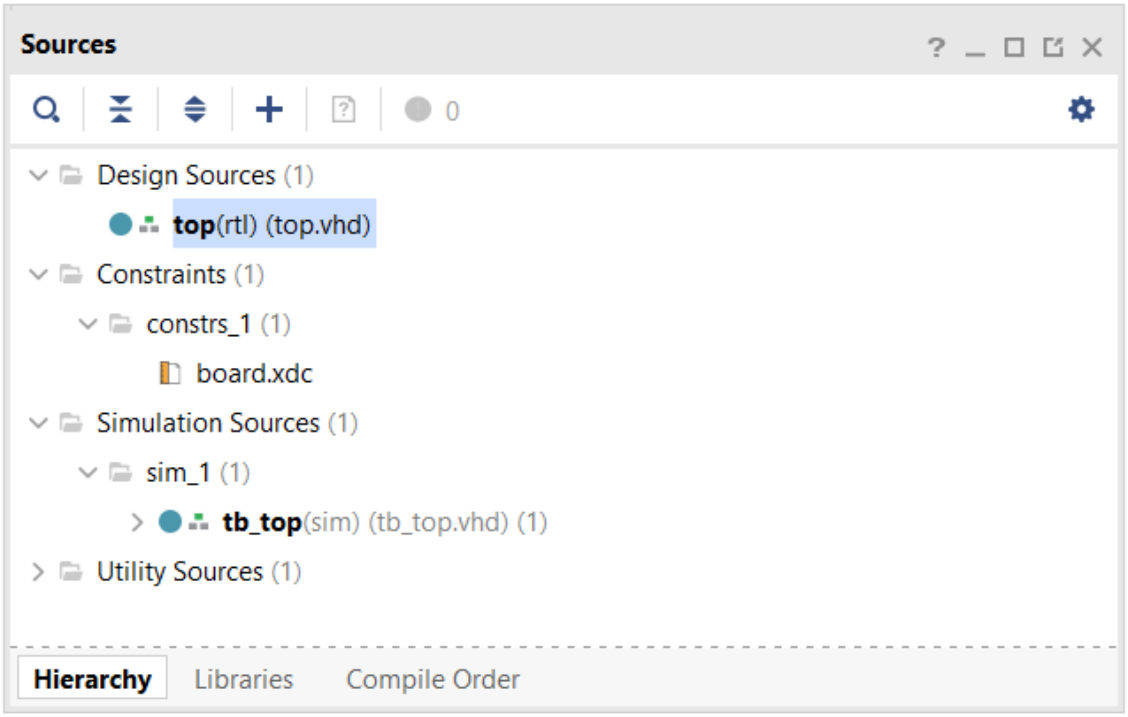
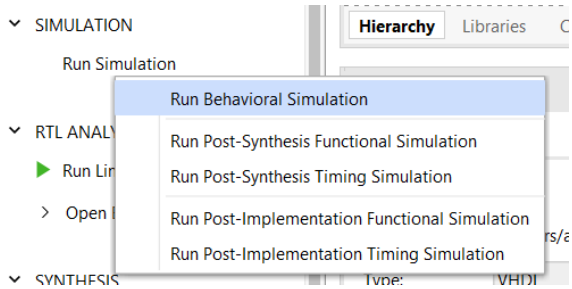


Figure 2: Source Files.

The first thing we want to do is to run a simulation. The simulation compiles the tb_top.vhd and runs a time simulation of the generated logical model.

The simulation in invoked via the *Flow Navigator* to the left.



After the simulator is launched it opens up a number of new windows. Most notably is the waveform window where the simulation result is plotted. As standard the simulation runs automatically for a

predefined period of time (typically 1 µs). In our case the runtime for the testbench is only 100 ns. To run or restart the simulation there are buttons in the toolbar. the first button is restart, followed by run all and then run for 'time'. The time and unit can be specified in the edit boxes. If we have edited the sources, there is a relaunch simulation button as well that re-compiles the sources and runs again.

After we have run the simulation the result can be seen in the simulation windows, and especially the waveform window. See Figure 3. By zooming, scrolling and clicking different signals we can examine to ensure our design does what it supposed to do. Simulation is one of the most important steps to ensure your design is working as intended.

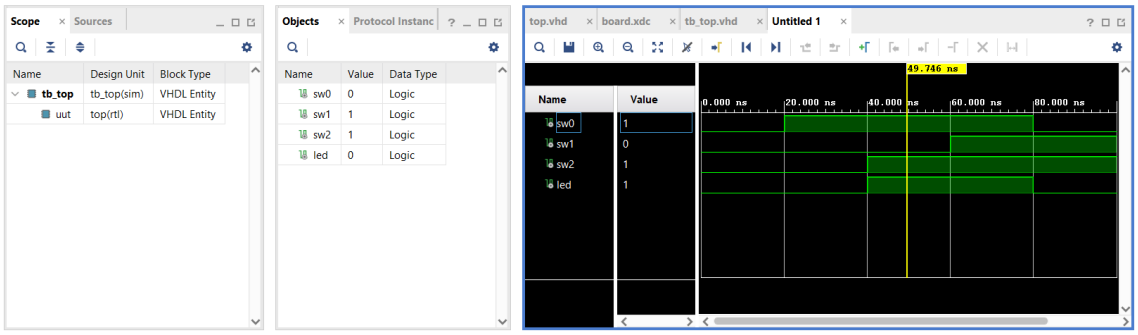


Figure 3: Simulation Windows.

IMPORTANT

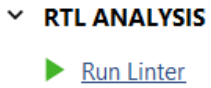
Simulation is probably the most important verification step to ensure your design works as intended. **Do not skip simulation!**

For more advanced simulation and to debugging purposes it is possible to add breakpoints to the code (both the testbench and your other code). As you learn more about testbenches you will be able to understand its strength in testing your design.

Figure 4: Simulation Windows.

3.6 RTL Analysis

By running the Linter, found under RTL ANALYSIS in the *Flow Navigator*, Vivado will compile our VHDL code into a generic logic netlist.



This is called the Elaborated Design, and by opening the schematic we can see a graphical representation of the design as seen in Figure 5. For our example we can see that the code line **led <= sw0 and (sw1 or sw2)** is indeed translated to what we expected.

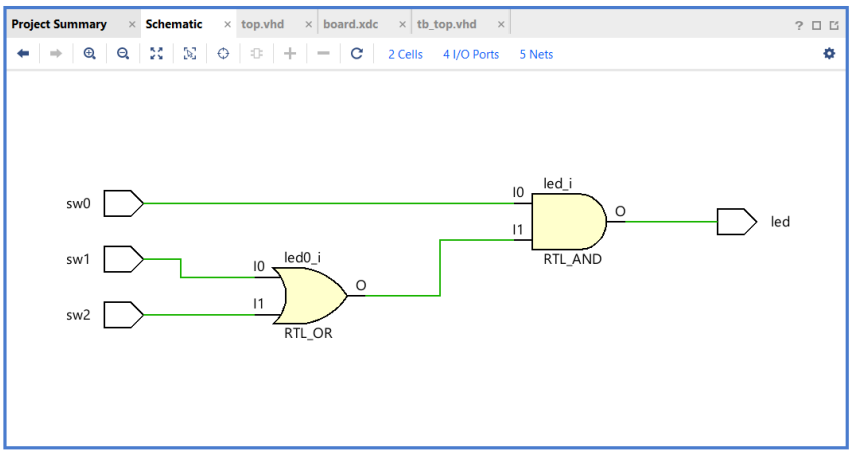


Figure 5: Elaborated Design.

3.7 Run Synthesis

Next step in our endeavour is to run synthesis. This is when the design is compiled into a device specific netlist. The difference from the elaborated design and the synthesis is that here it uses building blocks found in the specific FPGA, in our case Spartan-7. Note that the synthesis step can take considerable time (around one minute), and for very large designs it can take much longer time.

Figure 6 shows the schematic that opens up when pressing *Open Synthesized Design* → *Schematic*. As you can see it is not as before. It has input buffers IBUF after the port inputs, and an output buffer OBUF before the port output. Another thing is that there is no AND or OR gates to be seen. They are replaced with a look-up-table (LUT), LUT3. A LUT is a configurable block that can generate any output combination, and in our case it is programmed to implement the truth-table that can be seen in the pane in the lower left corner.

Another interesting thing we can inspect is the package window that also opens when we open the schematic. In Figure 7 you can see a graphical representation of all the pins on the FPGA device. If it doesn't open up automatically you can open it via the menu `Window` → `Package`. As a FPGA typically needs a lot of pins they often uses a IC package called BGA (Ball Grid Array), with a 2D arrangement of solder connections. Each connection as a name consisting of a letter for the vertical position and a number for the horizontal position. The position V2 is selected and shows some information for it. For instance is the pin connected to the net sw0 which is the name we have used in out design for one of the input ports. The connection between the physical pins and the net names is determined in the constraint file board.xdc.

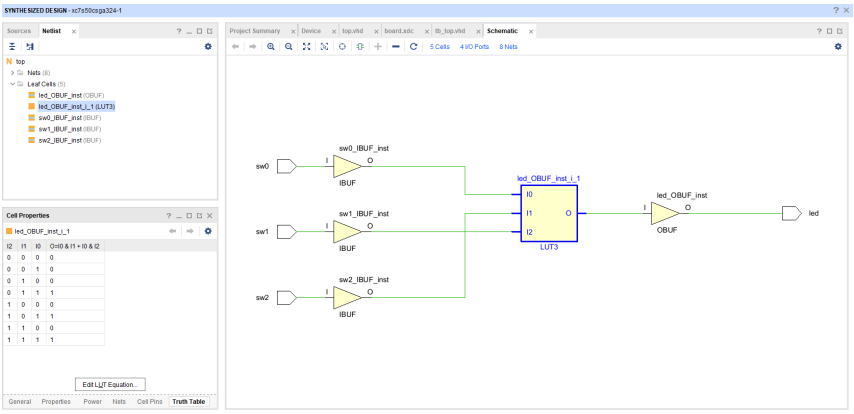


Figure 6: Synthesis Schematic.

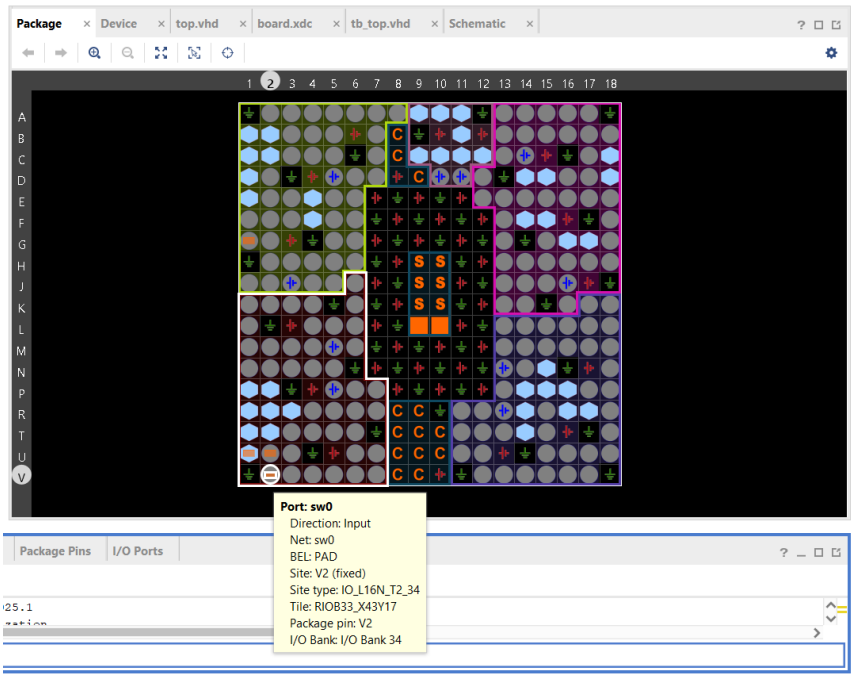


Figure 7: Package and pins.

3.8 Run Implementation

After synthesis we can run implementation. During implementation the placing and routing is performed. Placing means that Vivado decides exactly what slice and block inside the FPGA to place the synthesized logic in, and routing means that the tool finds the optimal signal route between the blocks. The placing and routing is done as an optimization process. Thus it tries to find the optimal placement and shortest routes for all the logic in the design. During this process it takes any timing constraints into consideration, and if all constraints are met we end up with a valid implemented design. For a large design the implementation can take many hours! That is why it is so important to verify as much as possible using testbenches and simulation beforehand. And we are also totaly blind of what is happening inside an FPGA once it is deployed on the board. We cannot single step and debug as we do when programming a microcontroller.

One can open the device window and zoom in and see the actual slices used. See Figure 8. The small blue-green box corresponds to the LUT and the orange boxes correspond the actual input pads and the IBUFs connected them.

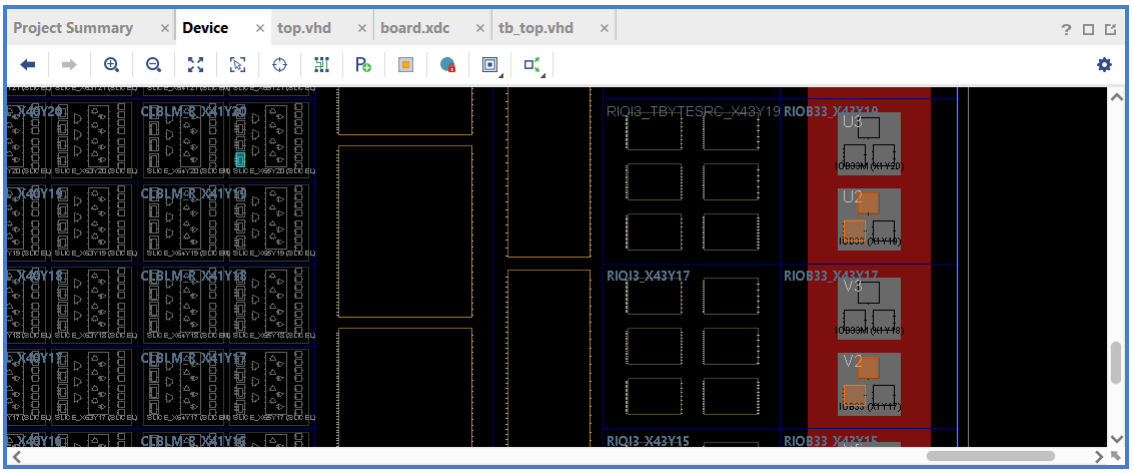


Figure 8: Package and pins.

3.9 Generate Bitstream

The final step to run the firmware on the actual Boolean board is to generate the bitstream. This create a binary file that can be downloaded to the FPGA. This binary file contains the configuration of all the LUTs, registers and routing etc and makes the device run according to our VHDL design. To download the bitstream we have to open the hardware manager. Ensure you have connected the board using the USB-cable. On the board it is the connector marked "PROG UART PWR" that should be used. After opening the hardware manager, we need to first Open Device (auto connect), and then right click the target FPGA (our Spartan-7 FPGA, xc7s50) and select "Program Device" to actually write the bitstream to the FPGA. See Figure 9.

To test your advanced device, try to toggle switches SW0-SW2 on the board and observe the led LD0. Do you see the light?

Congratulations to your first design using VHDL and a FPGA!

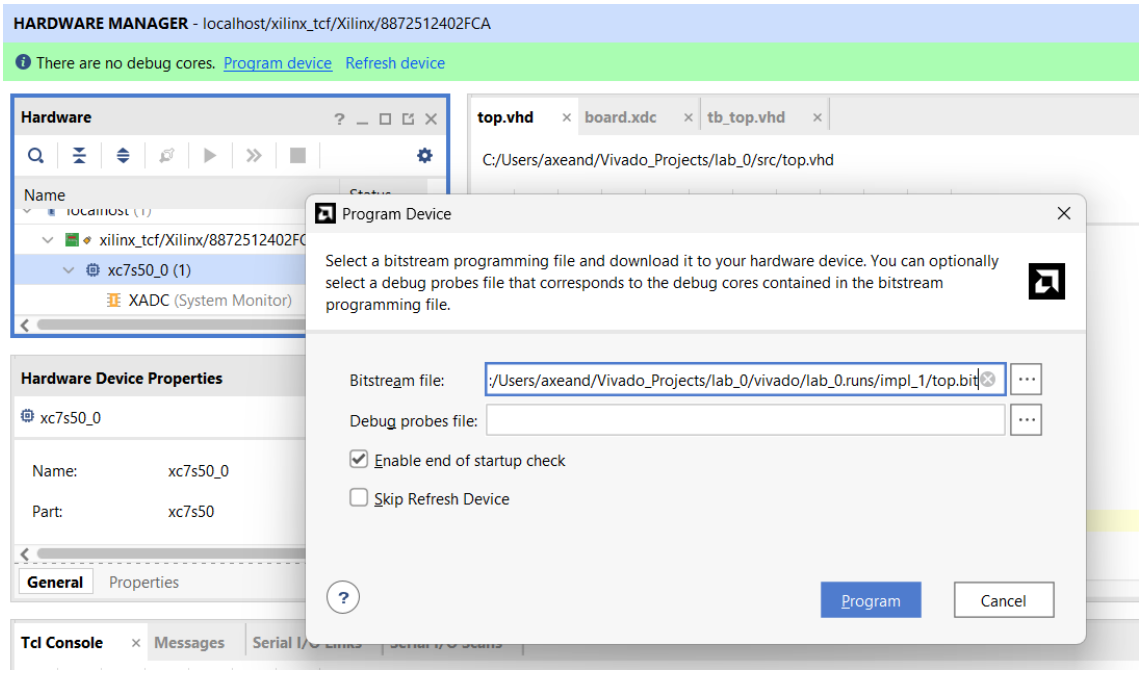


Figure 9: Program device.

A New Project using wizard method

Previously you were presented a method to create a project using a tcl-script and a template. For smaller projects and project not considered for version control the GUI method is adequate. Following below is the procedure to create the project using the wizard method.

At the startscreen when you choose *Create Project* you are presented with a Vivado project creation wizard. The wizard will guide you through a number of steps to create the new project. For instance you will need to provide a name and a location for the project. Then you will need to specify the type of flow you will be working with. Finally, you need to specify the project sources and choose a default part (the actual FPGA device you want to use).

A.0.1 Project name and location

After clicking next (on first wizard page *Create a New Vivado Project*), you should choose the the name and location of the project. See Figure 10. Make sure the path where you store your projects is not too long and **do not** contain any spaces as this can break the tool chain later on.

A.0.2 Project type

On the next page the type of project type / flow is chosen. As seen in Figure 11 there are many options to choose from. In this course we will only use the *RTL Project* as this is used to create a synthesizable project.

> **Note:** Ensure "Do not specify sources at this time" is selected.

A.0.3 Choose default part

Next step is to choose the target hardware for your project. Use Spartan-7 / `xc7s50csga324-1` for the Boolean development board. If you're using another board look at the board documentation to find the correct target FPGA circuit.

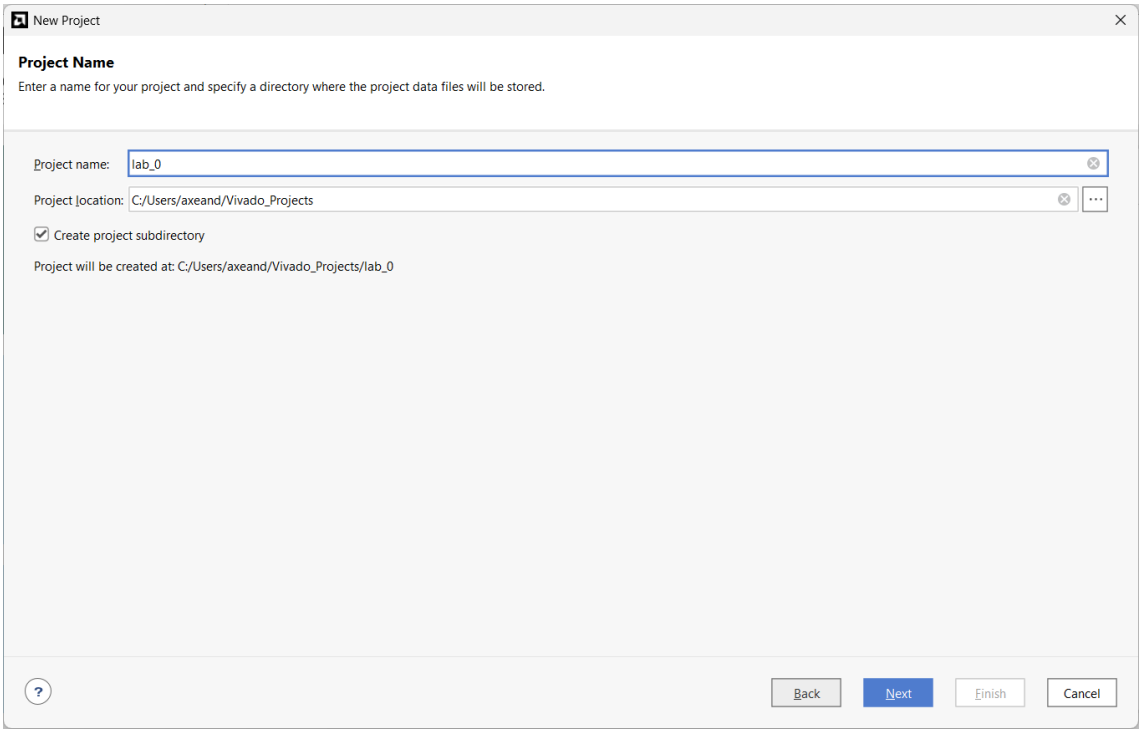
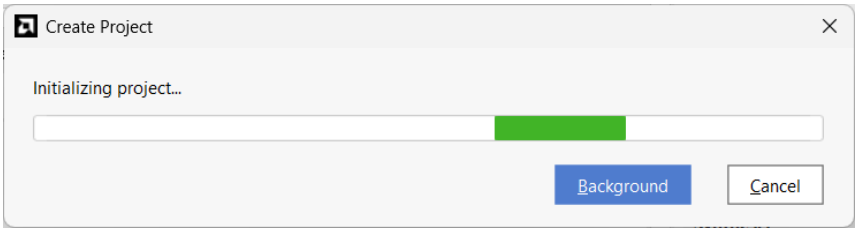


Figure 10: Project Name.

The easiest way to find the correct device is to use the search box and enter the desired device name: *xc7s50csga324-1*

A.0.4 Create project

After clicking Finish on the final page it creates the project.



When the project creation is finished the main Vivado window is presented with menus, toolbars, panes and windows. See Figure 13.

Most notably is the *Flow Navigator* to the left in the window. It contains the different process steps used during development. We will discuss the *Flow Navigator* later in this tutorial. At this stage we are ready to start designing our project.

IMPORTANT: As default the project target language is set to Verilog as seen in the Project Summary window. Click the blue Verilog link and select VHDL as the target language. Target language can also be selected in the Settings dialogue.

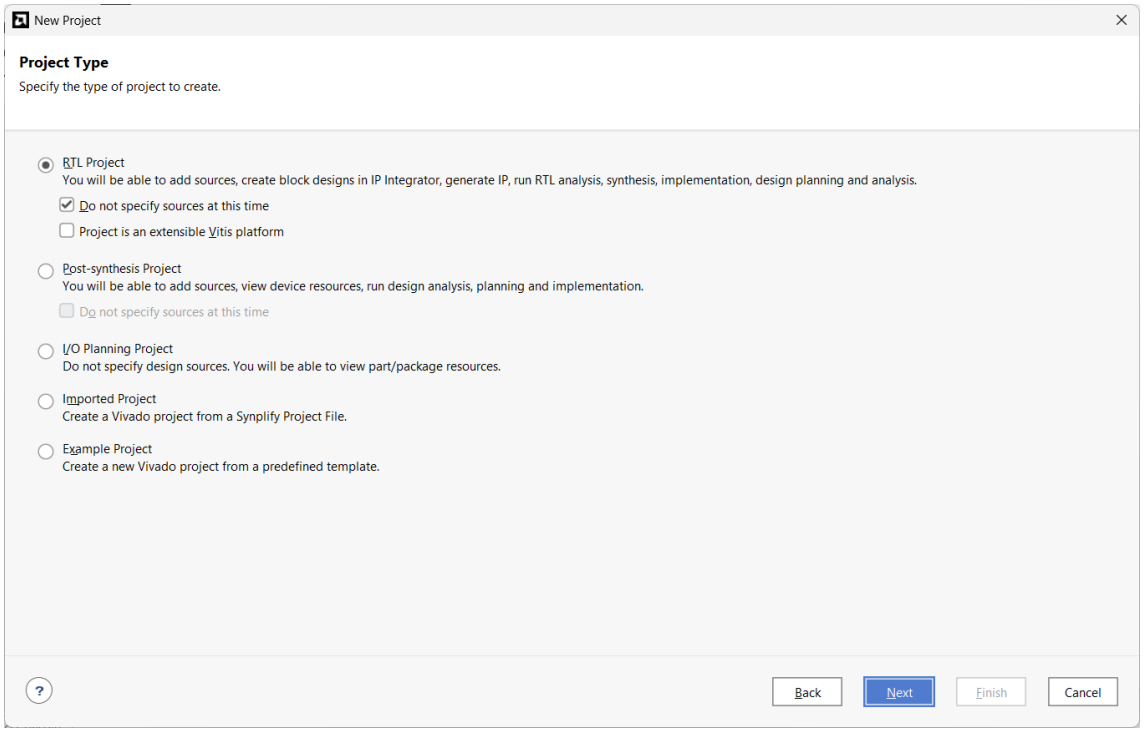


Figure 11: Project Type.

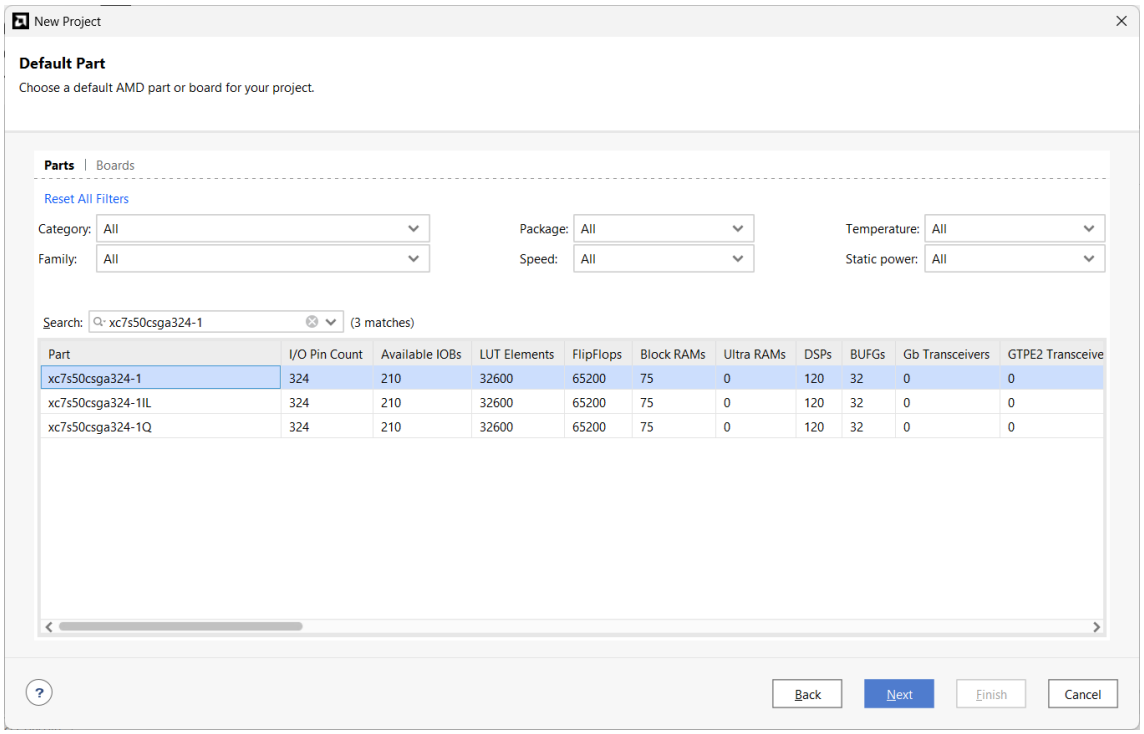


Figure 12: Default Device.

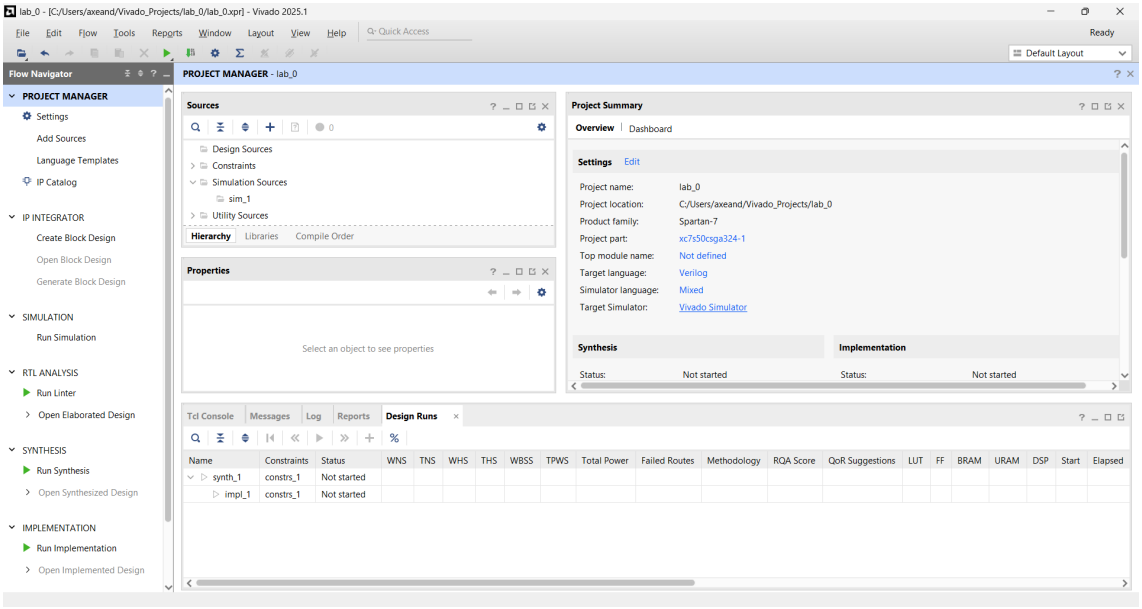


Figure 13: Main Window.