CHALMERS
Department of Computer Science and Engineering

2018-08-20

DAT093
Introduction to Electronic System Design
Introduction to Xilinx Vivado

Introduction

Vivado is an integrated GUI based tool from the FPGA vendor Xilinx for the development of hardware applications that can be downloaded into a Xilinx FPGA. The tool can be used all the way from creating source files to downloading the bitstream to the FPGA.

The source files can be written in a number of formats and languages as

- VHDL
- Verilog
- System C

We will only use VHDL.

There are basically three steps in the development process

- *Synthesizing the design*. The source code is converted to basic design blocks, but the design is so far not target at any specific FPGA device
- *Implementing the design*. The synthesized design is converted to the building blocks that are available in the target FPGA device
- *Place and route*. The building blocks are placed within the FPGA. The port connections of the design are connected to pins on the device

Throughout the process the design can be simulated with different objectives.

- *Behavioral simulation*. Simulation of the source code without any concern for the target device. This is more or less the same thing that we have been doing in QuestaSim but since the tool is different there might me slight differences
- *Simulation of the design after synthesizing*. We are simulating the primitive building blocks
- *Simulation after implementation and place and route*. We are simulating the actual context of the FPGA

The simulation after synthesizing and after implementation can be done in two different ways

- *Functional simulation*. We are only simulating the functionality of the design and are not looking at any timing issues
- *Timing simulation*. We are also looking at the timing of the design

Including timing will make the simulation run slower.
When it comes to tools, the simulation can be done using Vivados own simulator or using a third part plugin, this includes QuestaSim. We will use QuestaSim since we have been using that before and it´s more or less a standard tool in the industry.
When the design is finished the resulting bitstream can be downloaded to the FPGA on an evaluation board from within Vivado using an USB connection.
Our description will assume that you are running Vivado on a Windows PC. It can also be run under Linux, but you will have to figure out any differences.
There is a free version of the tool called Vivado Webpack that has some limitations but is qualified enough for our designs. The webpack only supports a limited number of FPGA devices and one of these is the Artix-7 XC7A100T that is being used on the evaluation board that we are using in this course. The board comes in two versions called Nexys4 or Nexys4DDR and they come from the vendor Digilent.
Goggle for "vivado webpack" and you will find the Xilinx webpage where the tool can be downloaded. You will have to register with Xilinx to be allowed to do the download.

Document structure

This document will not be a full description of the Vivado tool, but it will hopefully include the information that you will need in this course and some more. The description is based on version 18.2.1 of Vivado although version 18.1 is installed in the labs. The differences between the versions are very small, so you might not notice any difference at all.
We do the presentation by going through the steps to implement and download one of the designs from the first lab assignment, a 4-bit ripple carry adder. There will be some detours from this that are not needed for this design, but they will make the description somewhat more complete, these things might be useful in other designs later on.

The design process

Starting a Vivado design

You start the Vivado tool by clicking on the Vivado icon  on the Desktop or by opening it from the Start Menu, *Figure 1*.

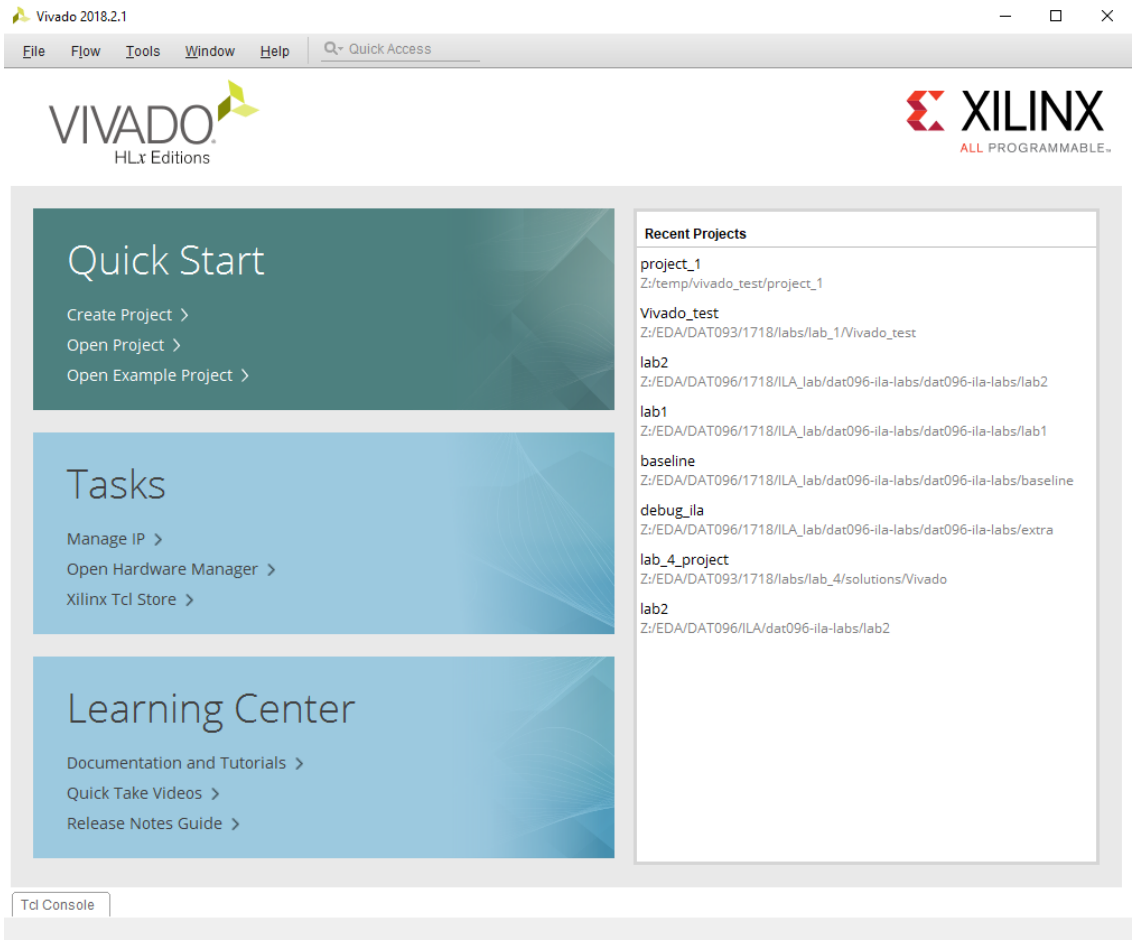


Figure 1 The Vivado GUI at start

You can see a menu at the top with some options but to get started we only need the links in the green Quick Start field. Here we can

- Create a new project
- Open an already created project
- Open an example project included in the Vivado distribution

If you have been running Vivado before you will have a list of your recently opened projects in the white Recent Projects field to the right. In *Figure 1* there are eight recent projects.
From the Tasks field, you can create and open IP blocks. You can also open the hardware manager and directly go to downloading a finished design to a FPGA. In the Tcl Store you can download third party scripts that might support your design flow.
From the Learning center field, we can get documentation and tutorials.

Opening a Vivado project

If you already have created a Vivado project, you can open it from the File/Open project… menu or from the Open Project link in the Quick Start field. In both cases a file browser will open, and you can navigate to the project, *Figure 2*.

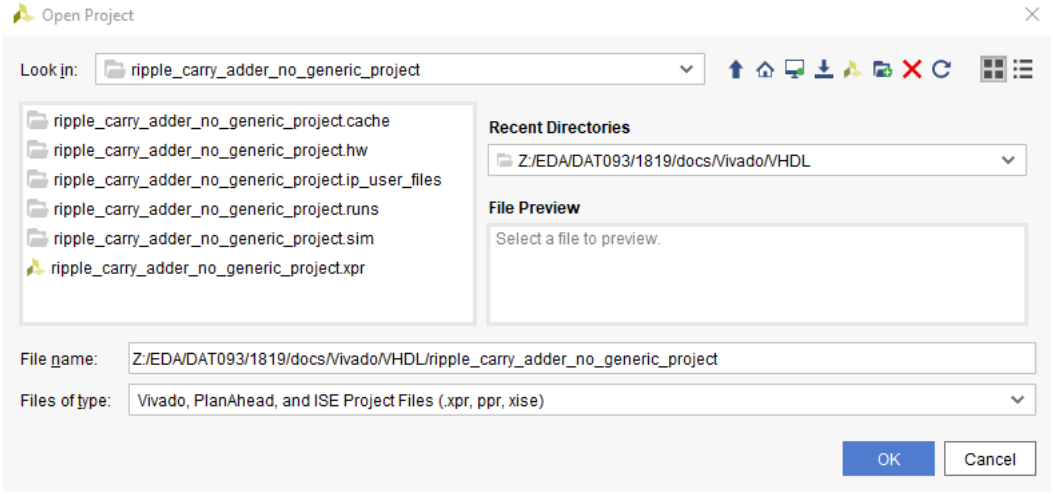


Figure 2 Open project window

In the project folder, you should look for the project file and this is called `<project name>.xpr`.

If it´s a recent project, you can also start it from the list in the white field to the right in the opening screen and in that case, you will directly start the project without having to navigate to any project file.

Creating a Vivado project

To create a new Vivado project you start the same way as when you open a project, but you select File/Create Project… or use the Quick Start link Create Project instead. In this case, there is of course no recent project.

When you click to start a new project, you will first get an information window, *Figure 3*.

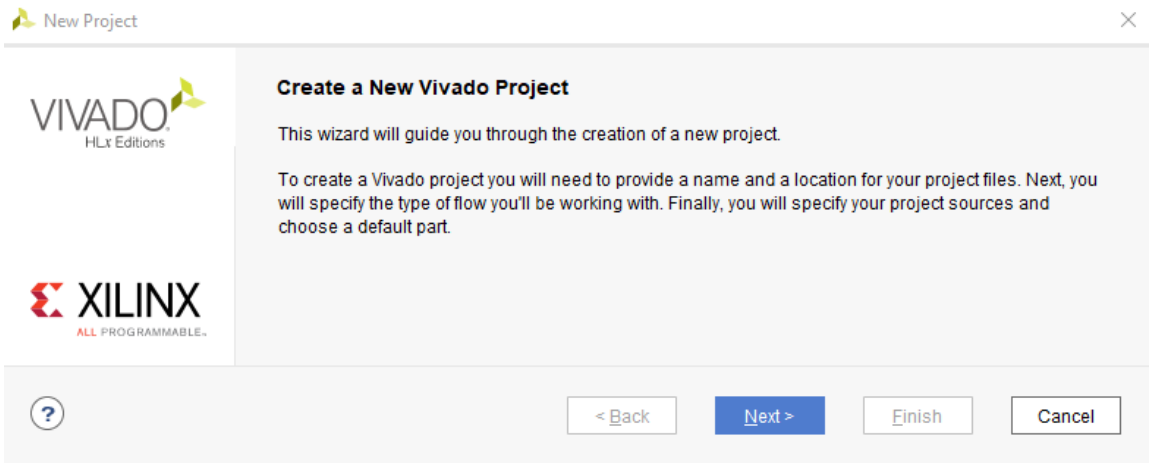


Figure 3 Create project information window

Click on Next > and you get *Figure 4*.

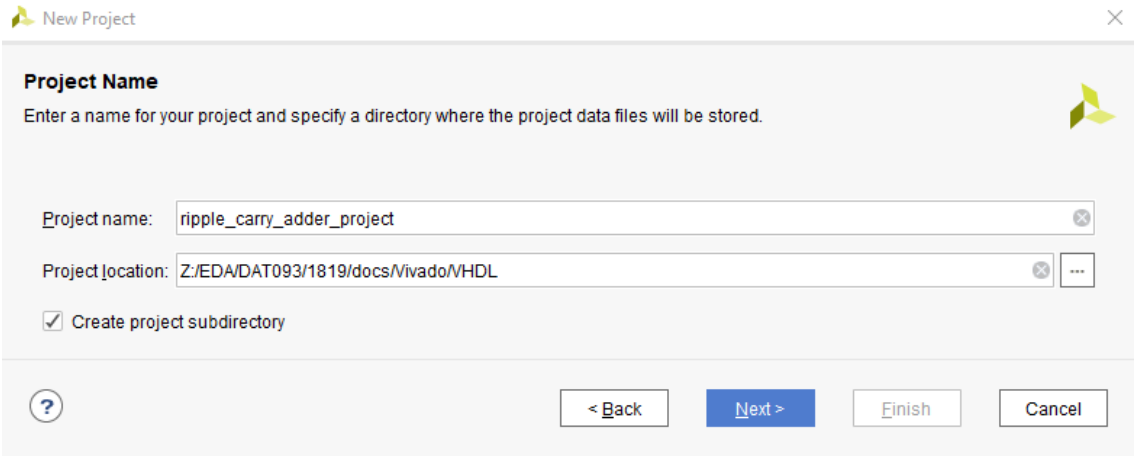


Figure 4 Create project naming window

In this window, you give a name to the project. By default, the project is called `project_1` but here I have changed that to the more informative name `ripple_carry_adder_project`.
You should define where the project should be placed within your file structure. This is easiest done by clicking on the browser tool ... and then navigate to where you want to place the project. Make sure that Create project subdirectory is checked. By doing this a new sub directory will be created where you decided to place the project and the project directories and files will be placed in this directory. When the project is created, there will be a complex structure of folders under the project folder and the structure expands as you go on with the project.
Click on Next > again and you get to *Figure 5* where you can specify the type of project.

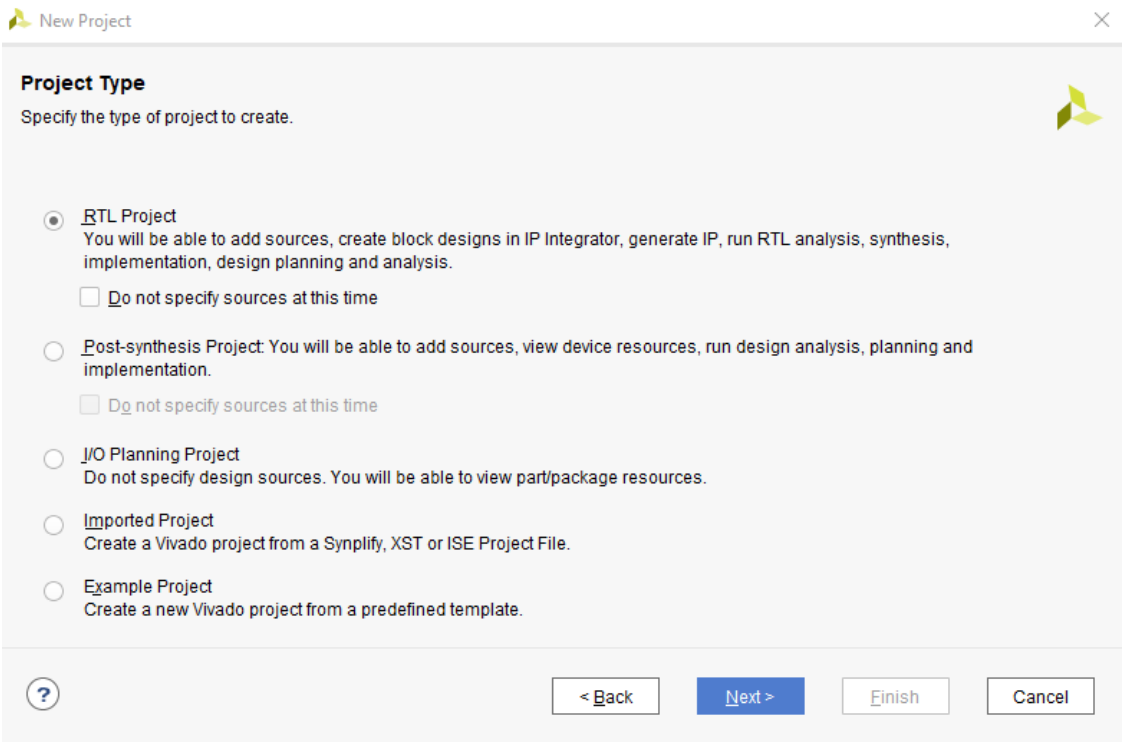
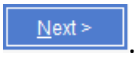


Figure 5 Create project setting project type

You have five project creation options to choose from

- RTL project (Register Transistor Logic). This is the basic project type. If Do not specify sources at this time is checked, we will skip that phase and create a project without source files. If it isn´t checked then we will move on to specify source files
- Post-synthesis project. You will create a project for an already synthesized design and use the netlists created by Vivado or some other tool as the starting point
- I/O Planning project. You start by creating a project where you only specify the ports of the design by importing a definition or by creating the ports
- Import project. You import a design project from another tool. It could be a Synopsis Synplify project or a Xilinx XST or ISE project
- Example project. You create a project from a predefined template

We will be creating RTL projects so check that option and leave Do not specify sources at this time unchecked.

Click on 

.

Setting up a Vivado project

In the next window, *Figure 6*, you can add and create source files.

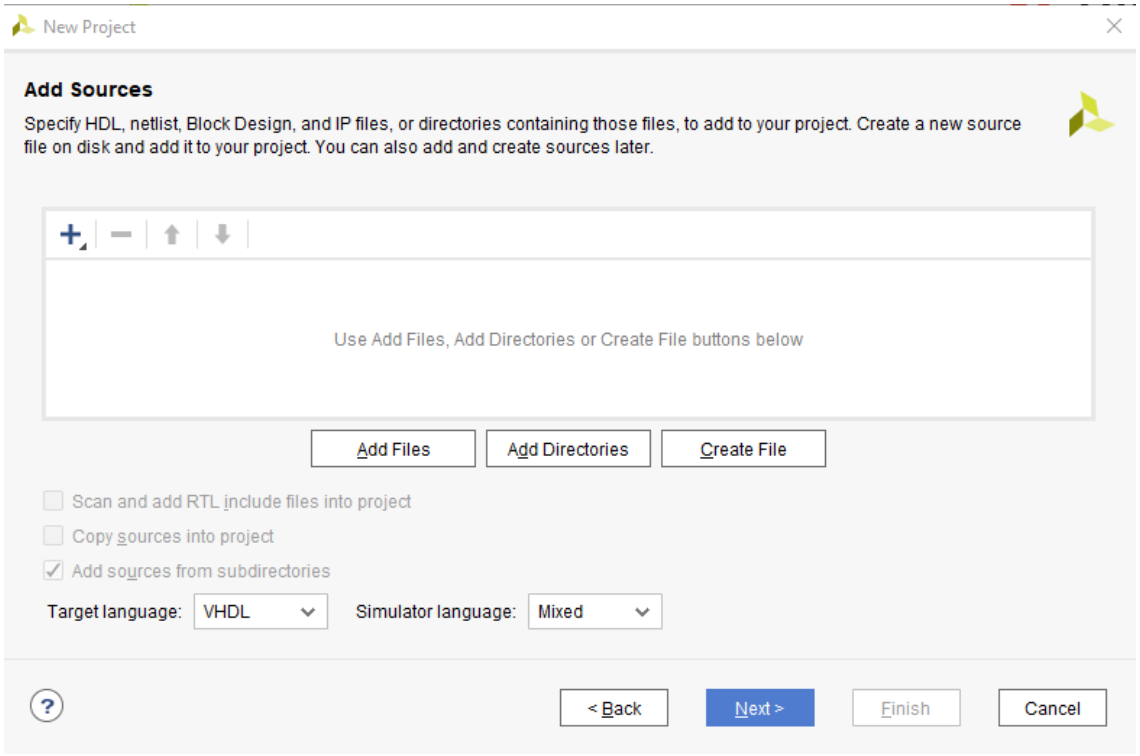


Figure 6 Create project add source files

We will in most cases have started our design in QuestaSim so there will be VHDL files from there to add to the Vivado project. Click on Add Files and navigate to the files and select the ones that you want to include in the project. You can select more than one file at the same time.
Make sure that Simulator Language is set to Mixed since some of the files that are created when simulating an implemented design are generated in Verilog.
If you check Copy Sources into project, then the original source files will be kept intact while you are using copies of the files in the project. If you use the original files they don´t have to be placed in the project folder but will be referenced from where they are placed. You decide if you use copies or not. I often find it better not to use copies since if I do there will be two versions of the file and it can be confusing what file you are using at any one time. If you want to use copies, then copy the source files to new files with new names instead to lower the confusion.
Since we will use the ripple carry adder as an example in this introduction we will add the files we used for that design in the earlier lab assignment in QuestaSim. In my case the design of the ripple carry adder with saturation consisted of three files, the 1-bit full adder, a ripple carry adder with overflow using the full adder as a component and finally a ripple carry adder with saturation using the ripple carry adder with overflow as a component, so I add these three files, *Figure 7*. There is also a testbench for the ripple adder, but this is just for simulation and not for synthesis, so we leave it out for now. We will use it later on when we simulate the synthesized design.

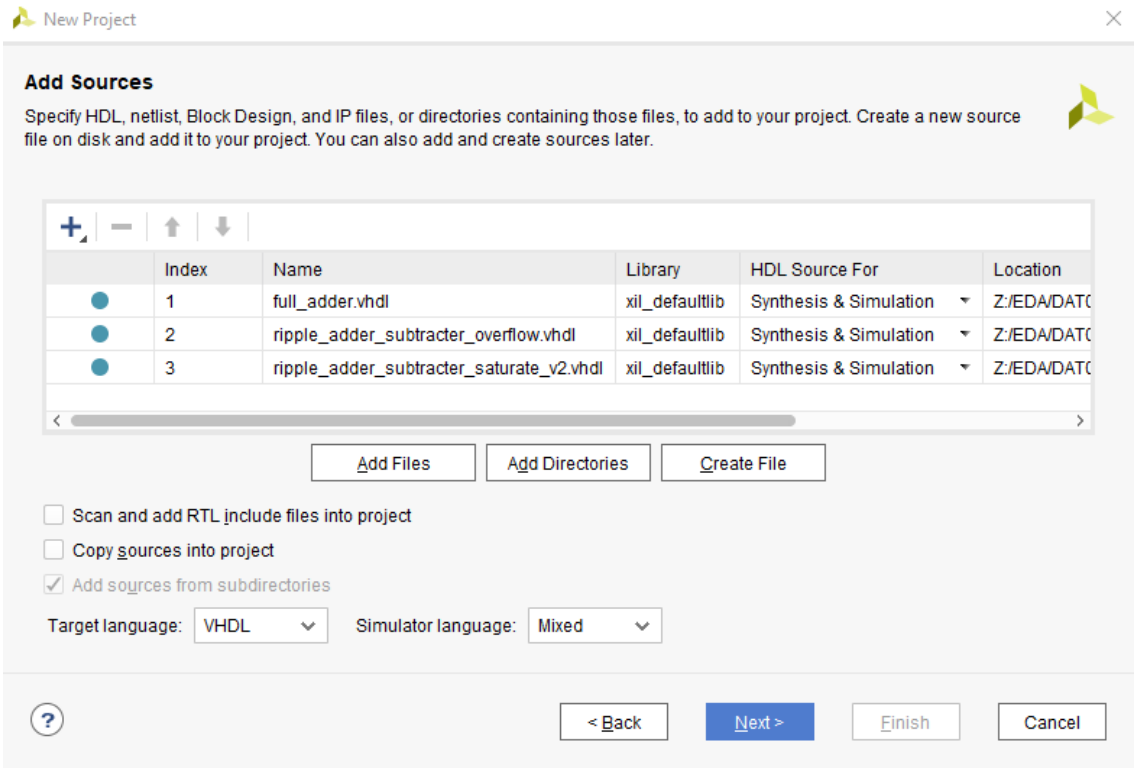


Figure 7 Create project add source files with added files

You can also create files here by clicking on Create File and then use the dialogue in *Figure 8*.
By selecting File type, you make sure that the file gets a correct structure. You can create files of the types

- Verilog
- Verilog header
- SystemVerilog
- VHDL
- Memory file

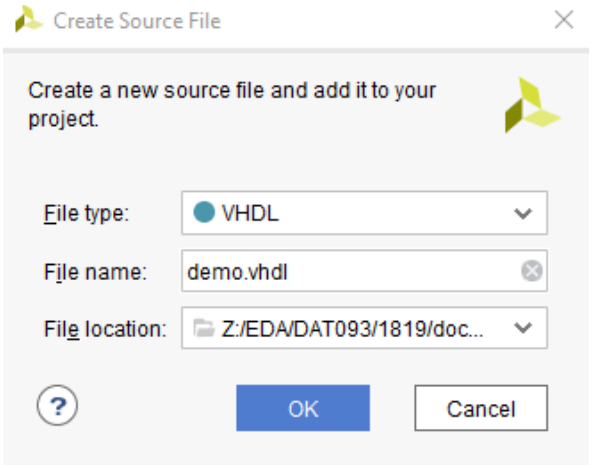


Figure 8 Create file dialogue

We will use VHDL files.
If you don´t select a File location, then the file will be placed in the project folder.
If you set the VHDL file name without an ending a VHDL file will get the ending `.vhd`. I prefer the ending `.vhdl` so I have added that to the file name.
A file with the basic VHDL structure will be created. At a later stage, you will be able to define ports for the design in the file.
Now we have all the files for our design but let´s create a new file anyway as an illustration. Since it´s not needed we will eventually remove the file from the project. Let´s call the file `demo.vhdl`. I have placed the file in a separate folder since it is just for demonstration

purposes and will be removed from the project. The file will be added to the list in the Add Sources window, *Figure 9*.

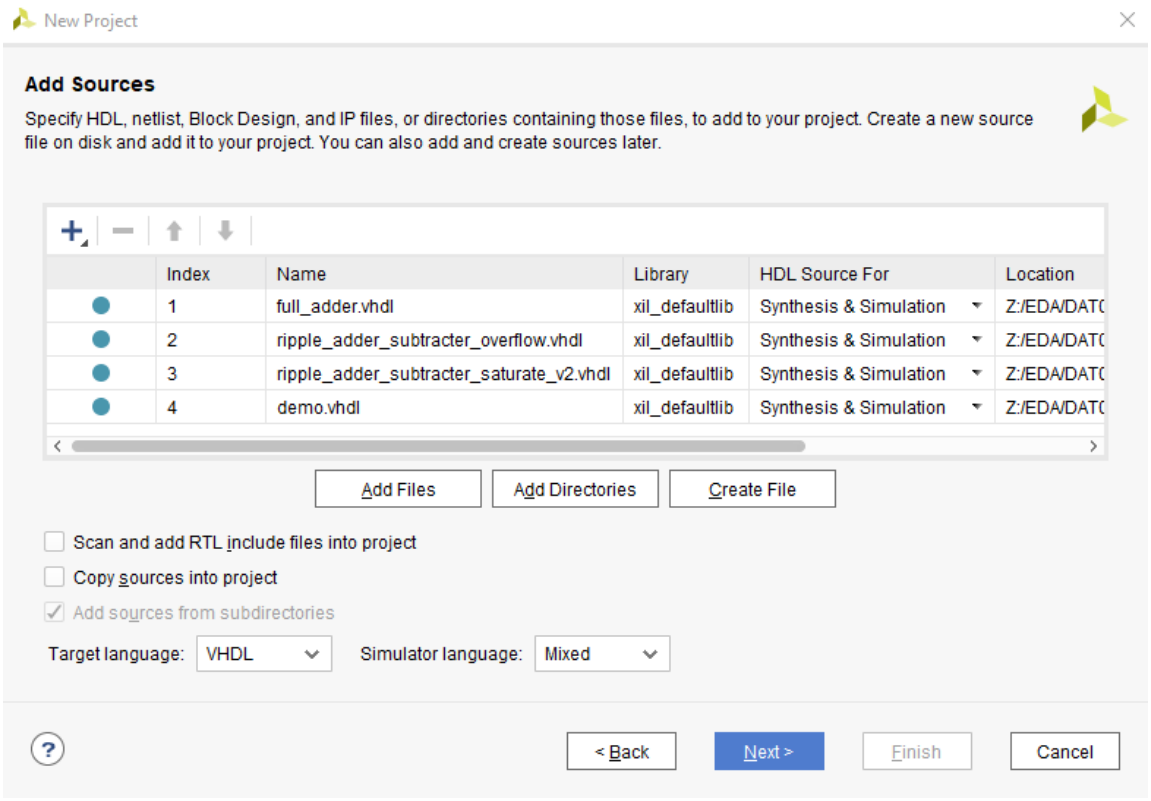


Figure 9 Create project add source files with added and created files

Click on 

. We will get a new window where we can add a constraints file, *Figure 10*.

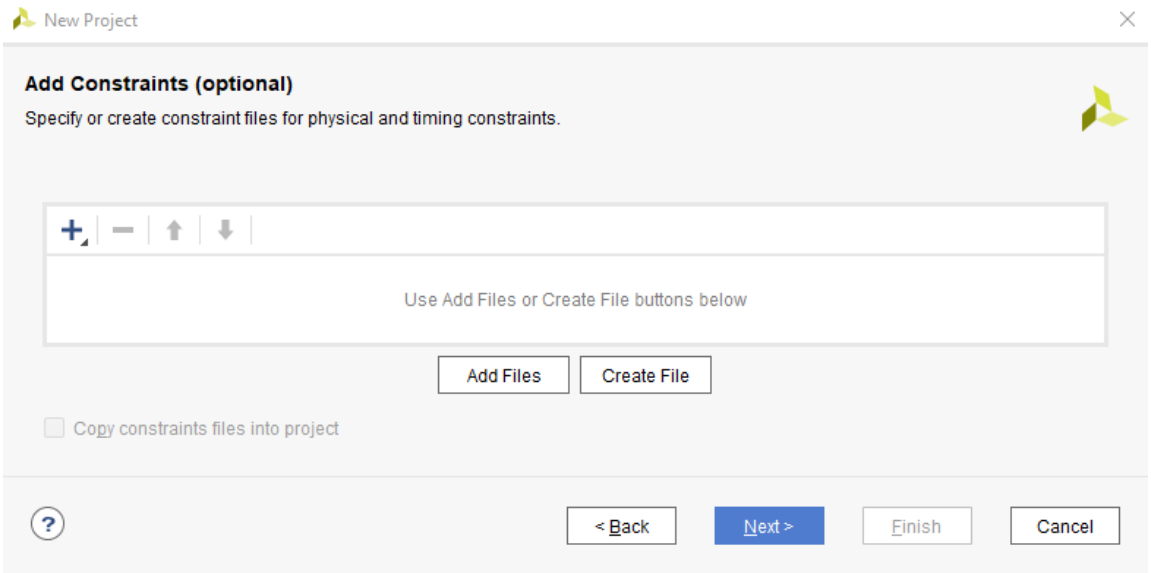


Figure 10 Create project add constraints file

In the constraints file (XDC file) we define conditions for our design. This can for example be requirements on the timing of the design. There is one type of constraint that we must include and that is the definition of to what pins of the FPGA the in- and output ports of our design should be connected. We will be using a user-board called Nexys4 or Nexys4DDR, there are two versions of the board, that besides the FPGA contains a number of peripherals like switches, push buttons and LEDs and the objective is to use these peripherals to give the input values to our adder and indicate the resulting outputs.
There are two predefined XDC files for the Nexus4 boards. There are two files since as we mentioned there are two versions of the board, one with DDR memory and the other one without that. We have a mix of both types of boards, so you must make sure to select the appropriate XDC file. Both files are uploaded to the PingPong homepage. We will get back to how to use the constraints files.
I have the Nexys4 board with the DDR memory for this demo, so I will add that XDC file. I am not adding the template file but a copy of it so the template doesn´t get corrupted when the XDC file for the project is edited, *Figure 11*. As you can see I have renamed the copy to reduce confusion.

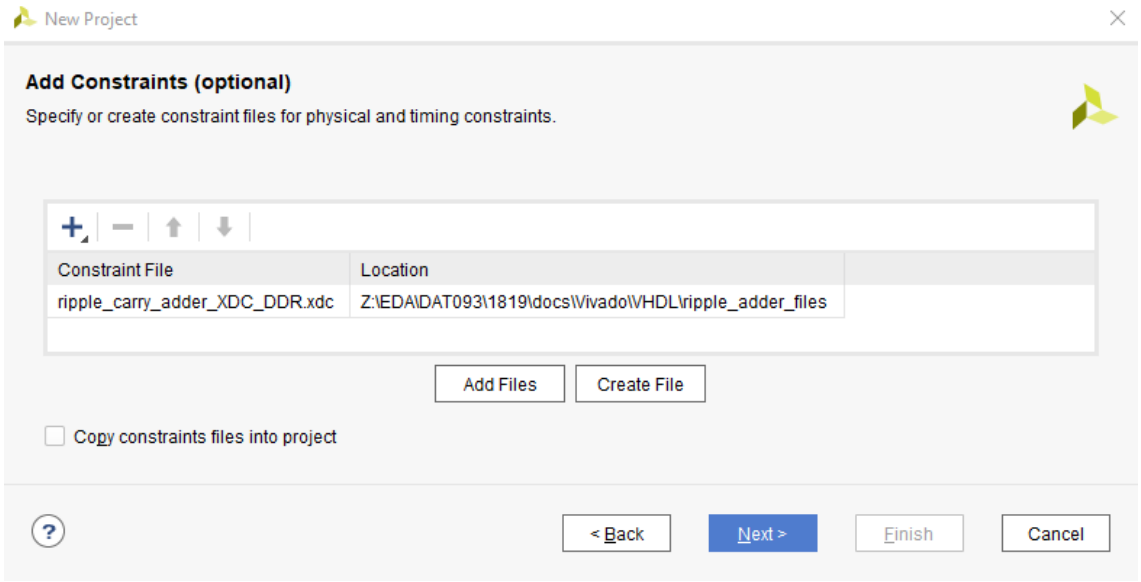


Figure 11 Create project with added constraints file

Click on 

.
We get a window where we can select the FPGA that is used. By using the filters, we can limit the number of possible choices. The Nexys4 board has a Artix-7 FPGA onboard so we select that as Family. The FPGA is placed in a csg324 package, so we select that as Package. Finally, we set the Speed grade to -1. *Figure 12*.

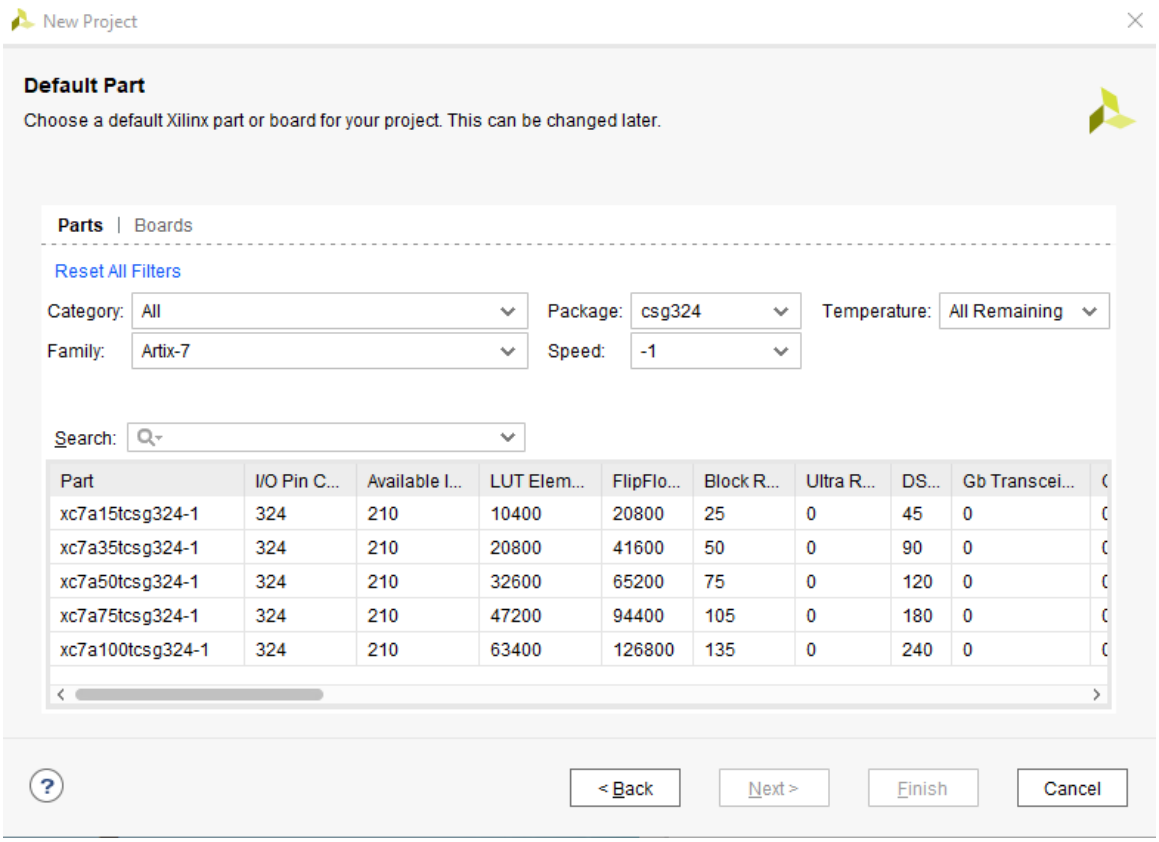


Figure 12 Create project default part

We get a list of devices that fit our description. Our actual device is the xc7a100tsg324-1 so we select that one. We can change device later one in the project if we like but then we will need a new XDC file that is adopted for that device.

Click  .

We get a summary of our configuration, *Figure 13*.

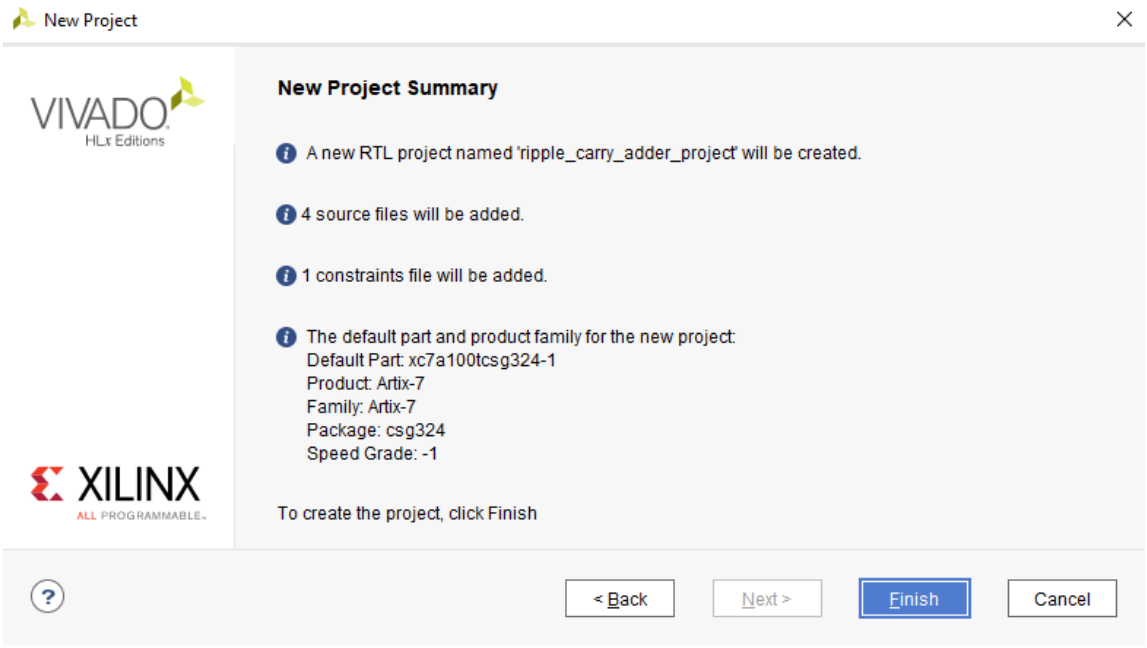


Figure 13 Project summary

The project will now be initialized. Next a window opens where we can continue setting up the VHDL file `demo.vhdl` that we just created, *Figure 14*.
Let´s say that our VHDL design in that file has one 1-bit input called `enable`, an 8-bit input called `a` and a 1-bit output called `y`, *Figure 15*.
For each in- and output port we give a Port Name. We set a Direction that can be

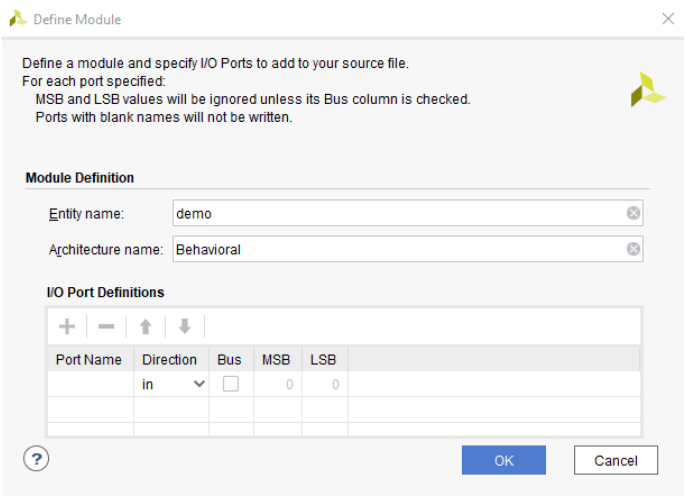


Figure 14 Module definition

- In
- Out
- InOut

where InOut should only be used if it´s absolutely necessary.

That´s all for the 1-bit ports.

We use the + sign to add a line for a new port and by selecting a port and clicking − we can remove a port.

To create the 8-bit port we must check Bus and then we select the ports bit range. By setting MSB higher than LSB we will get a vector with the indexes defined by DOWNTO. If we input them the other way around, we get indexes defined by TO.

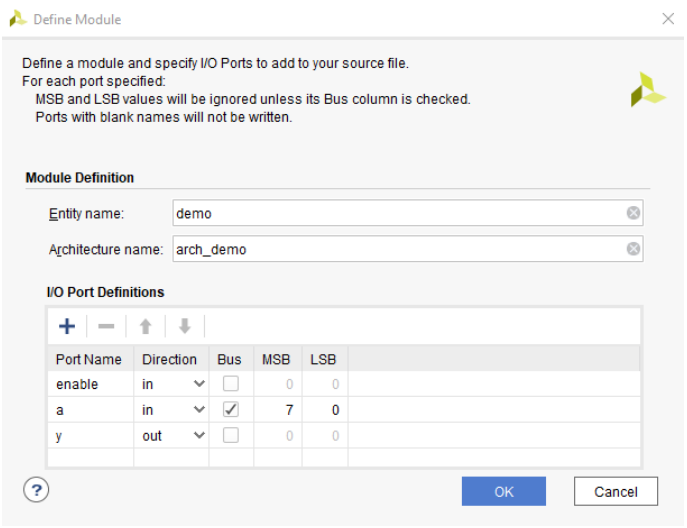


Figure 15 Port definitions for demo.vhdl

Set the name of the entity to the same name as the file name, excluding the `.vhdl` ending. The architecture will by default be called `Behavioral`, but we set the architecture name to `arch_demo` which is my preferred way of naming architectures with the same name as the entity headed by `arch_`.

The created file will look like *Table 1*. Here the file is slightly edited to fit to the page.

```
----------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 08/16/2018 11:28:40 AM
-- Design Name:
-- Module Name: demo - arch_demo
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
-- use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if instan-
-- tiating any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity demo is
    Port ( enable : in STD_LOGIC;
           a : in STD_LOGIC_VECTOR (7 downto 0);
           y : out STD_LOGIC);
end demo;

architecture arch_demo of demo is

begin

end arch_demo;
```

Table 1 demo.vhd

The file contains a template for a commented documentation part, the IEEE library we always need, a commented library that might be needed, the entity with our given name and specified ports and an architecture with the specified name. What´s missing is of course the actual functionality, the context of the architecture.
In most cases I don´t find this way of creating the file structure that practical. It´s often easier to open up an old VHDL file, delete all but the basic structure and then edit the file directly to fill it with the wanted design. Don´t forget to save the file under a new name so the old file is kept unchanged.
Most of the configuration is now done and the GUIs main window will open, *Figure 16*.

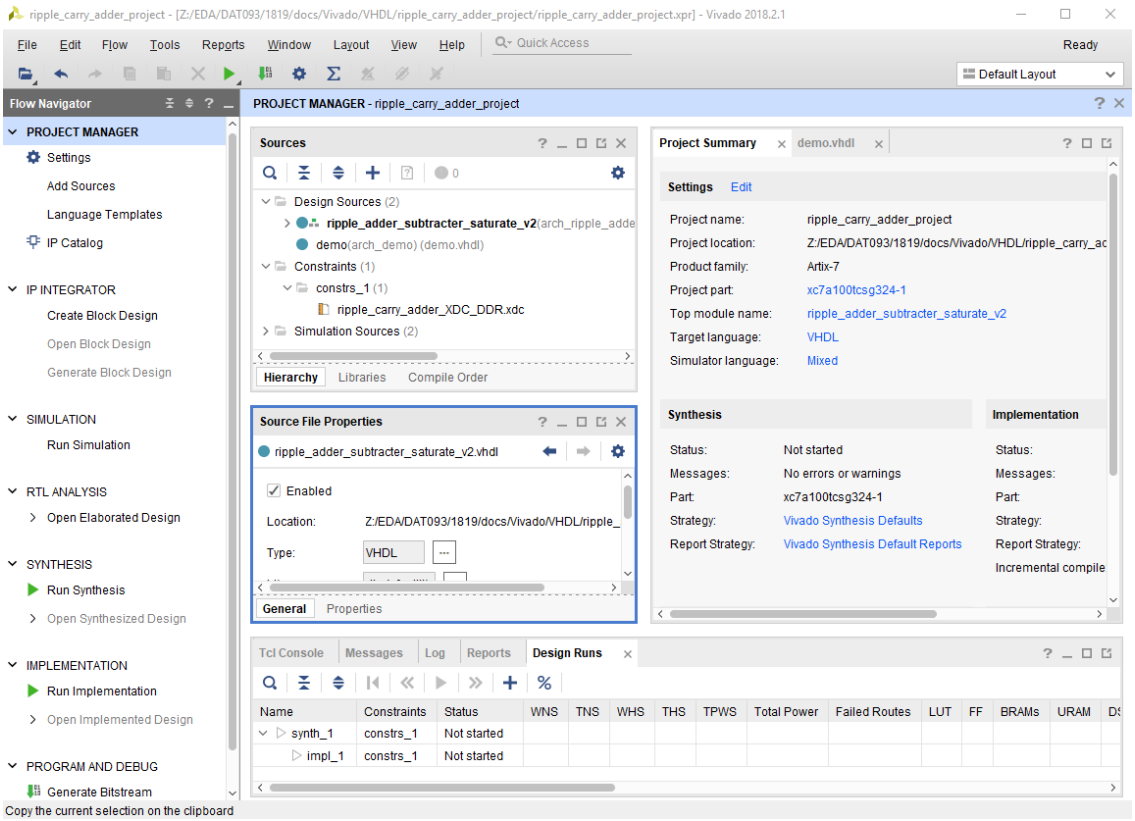


Figure 16 Main window

We can see a number of frames. So far, the only frame that is populated is the Sources frame. Let´s look a little closer at that frame and at the same time open the file structures within the frame, *Figure 17*.

Here we have turned the window into a floating window by clicking at  in the top right corner of the window. We can put it back into the GUI by clicking at  .

We have now removed the `demo.vhdl` file since it was just for demonstration purposes and should not be part of the project from now on.

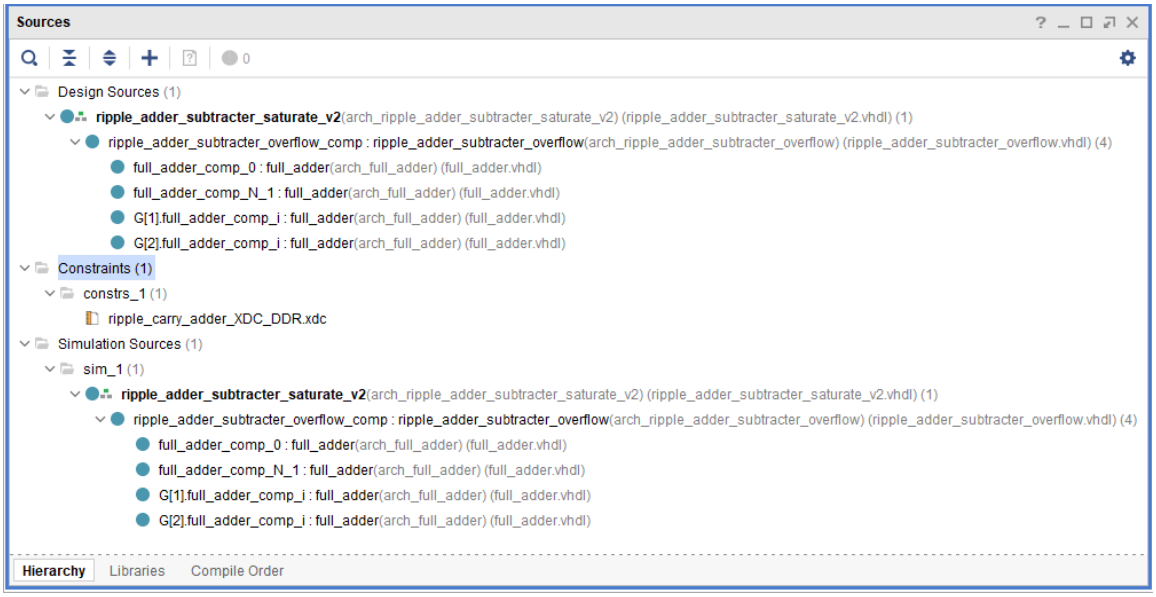


Figure 17 Sources frame

Under Design Sources we can see all the VHDL files in the project and the tool has analyzed the files, so it can display the hierarchical structure between the files. If the structure isn't there, then there is something wrong with the binding between the higher-level sources and their components. In most cases the reason is that the entity of one of the components doesn´t match the component declaration and/or instantiation at the higher level.

Under Constraints we see our constraints file (XDC).

There is also a Simulation Sources group that contains the same thing as the Sources group. In this group, we see what will be simulated if we run a simulation.

We can add a testbench to this group by right clicking on the Simulation Sources heading and select Add Sources…. We get *Figure 18*. We can do the same from the Project Manager to the left in the GUI by selecting Settings/Add Sources.

We could have added the testbench at the same time as we added the source files but then Vivado can´t separate the source files from the testbench and we need to edit the structure.

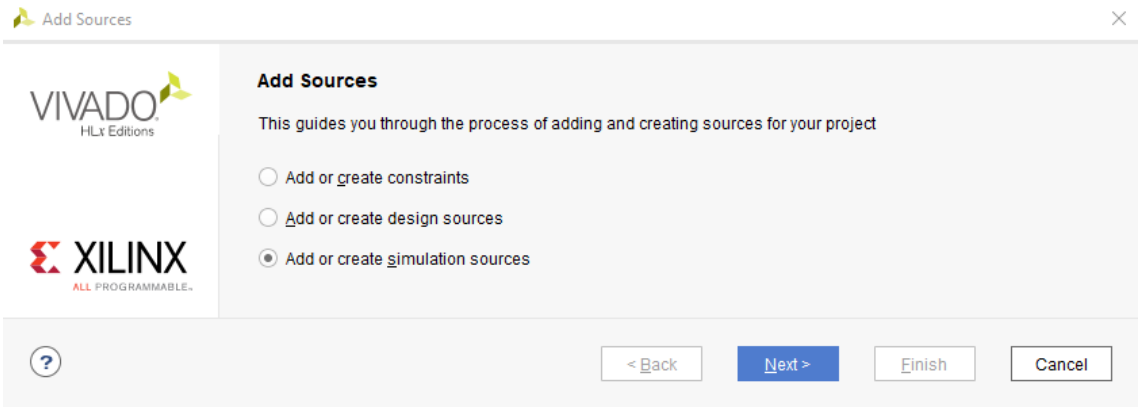


Figure 18 Add sources dialogue

As can be seen we can add source files, constraint files and simulation source files here. We check Add or create simulation sources. Click Next > to get *Figure 19*.

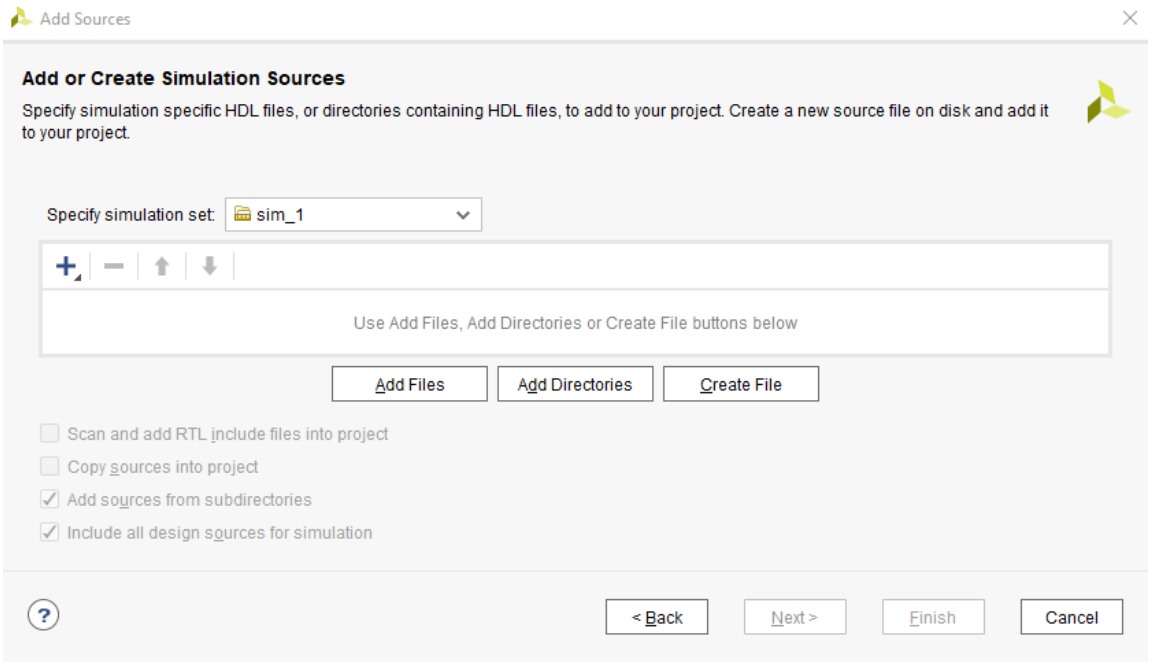


Figure 19 Add simulation sources and specify simulation set

In this dialogue, we add or create the file. We have a testbench from the earlier lab, so we add that one.
We must also declare what simulation set it belongs to, we can have several simulations with different configurations. We have only one set, so we select sim_1, *Figure 20*.

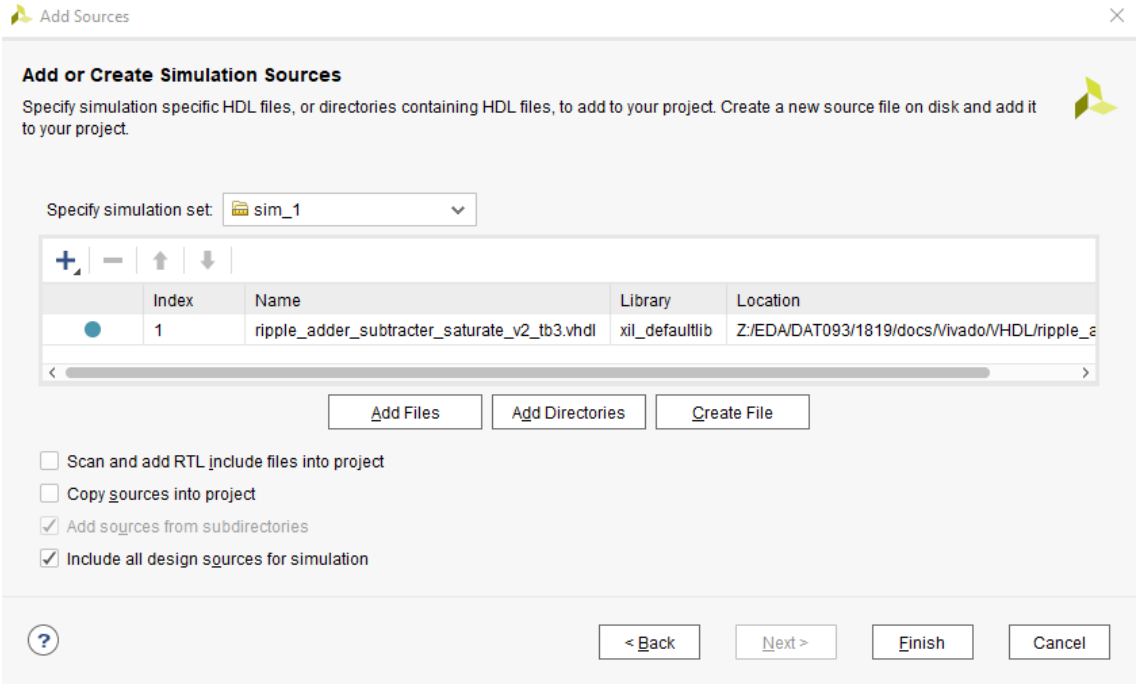


Figure 20 Simulation source added

If we look at the Sources frame again the Simulation Sources heading have changed, *Figure 21*.

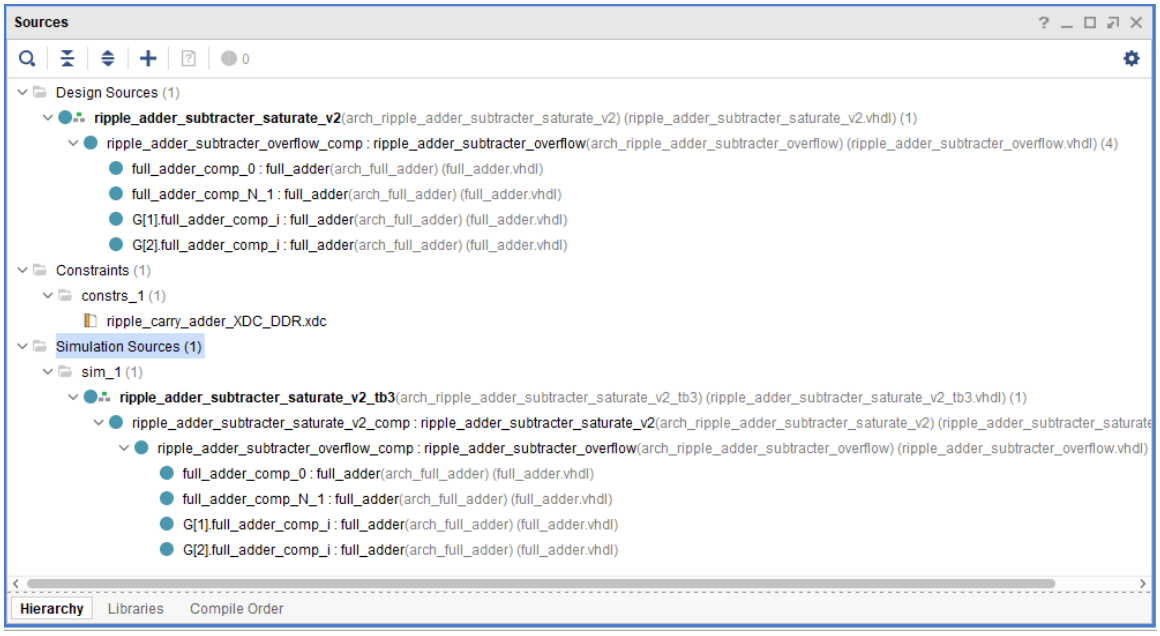


Figure 21 Updated Sources frame

Now the testbench is the top design that will be used at simulation, but it will not affect the synthesis chain.
The testbench has some limitations when it comes to simulating synthesized or implemented designs. When we do that the simulation will be done on the processed design,

not the VHDL code, and in the processed code things like the size of vectors is set meaning that we can´t simulate designs where we control these through GENERICS. We must have fixed values.
This complicates things and it is often simpler to skip the testbench and instead simulate the design directly and assign values using a do file. But if we do this we lose the possibility to do the testing of the out signals that we can do in a testbench. We will have to check them ourselves. We simulate the actual top level VHDL file, so we are back to *Figure 17*.
To keep both options without adding and removing files we can disable the testbench file by right clicking on it and select Disable File. The testbench file will now be moved to a group called Disabled sources, *Figure 22*.

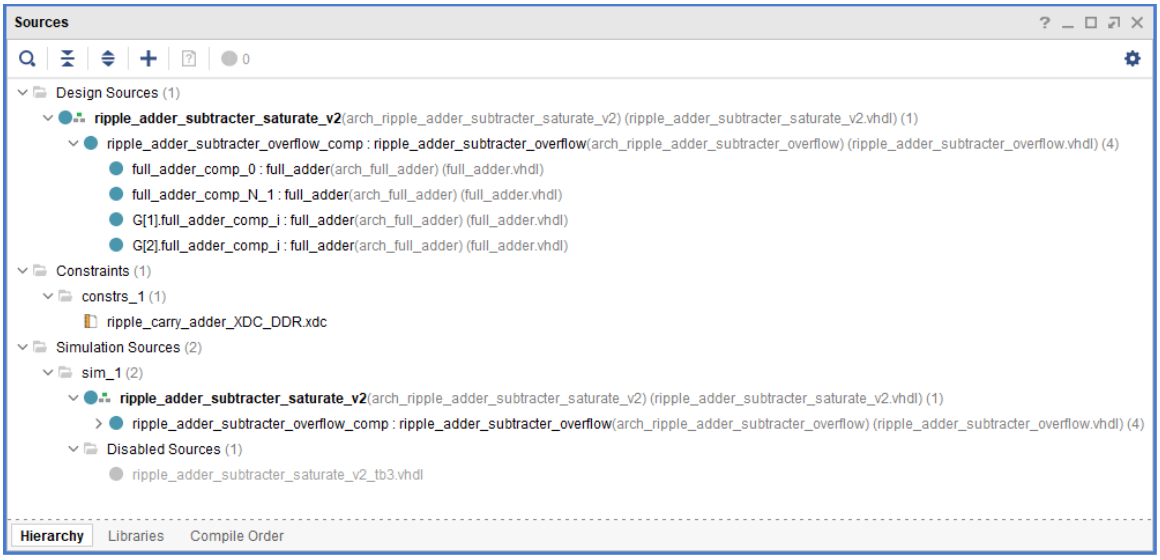


Figure 22 Updated Sources frame

We can take back the testbench into the project by right clicking on the file and select Enable File.
With this the initial setup is done and we can move on to actually do some simulation and synthesis.
Let´s look at the Flow Navigator frame, *Figure 23*. From this frame, we control the steps of the design process and we will take the items one by one staring by opening the Project Manager, *Figure 24*.

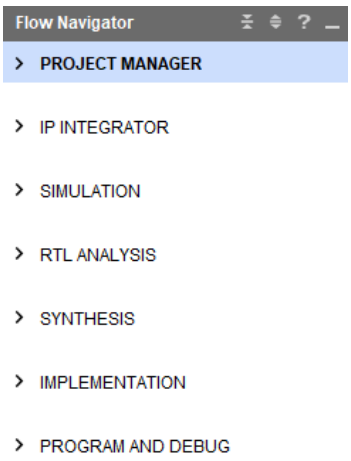


Figure 23 Flow Navigator

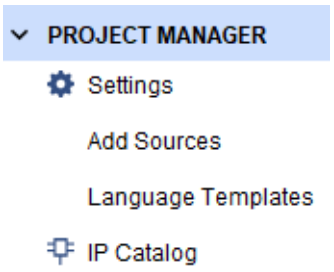


Figure 24 Project Manager

Let´s look at the Settings by clicking on that heading, *Figure 25*. We start with the General tab.

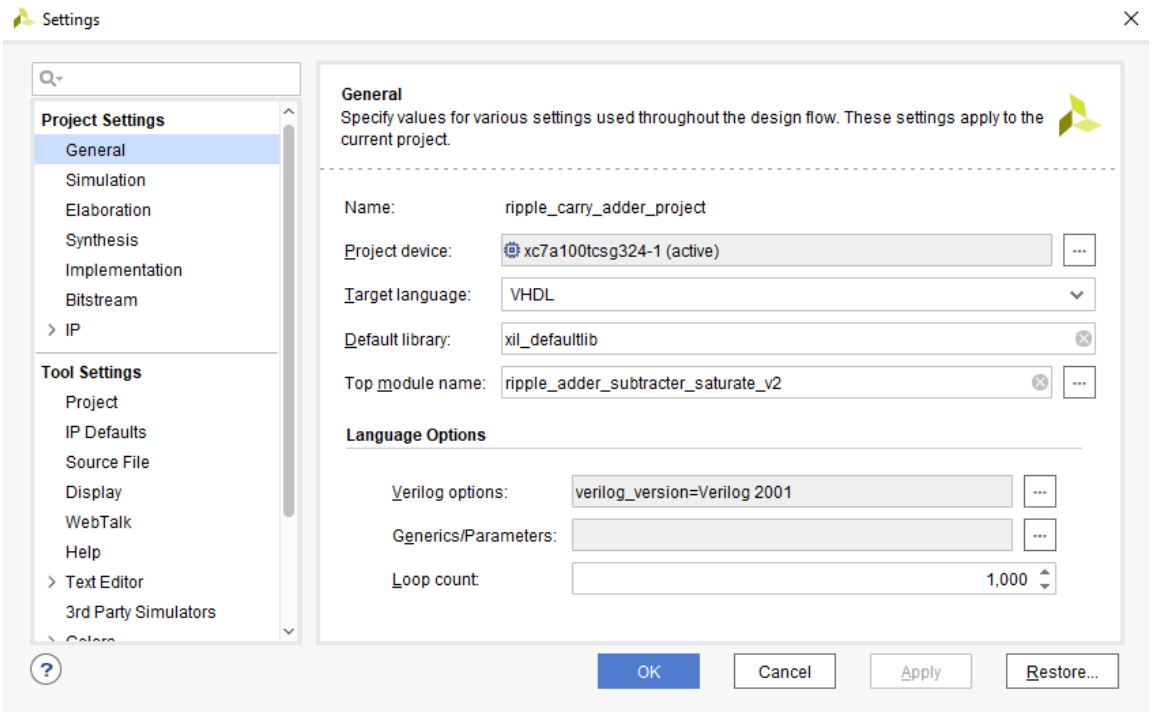


Figure 25 Project settings - General

Here we can change the target FPGA. If we click on [...] we get the same dialog as earlier, *Figure 12*.
Make sure that the Target language is set to VHDL. That´s all we need here.
We move to the Simulation tab, *Figure 26*.

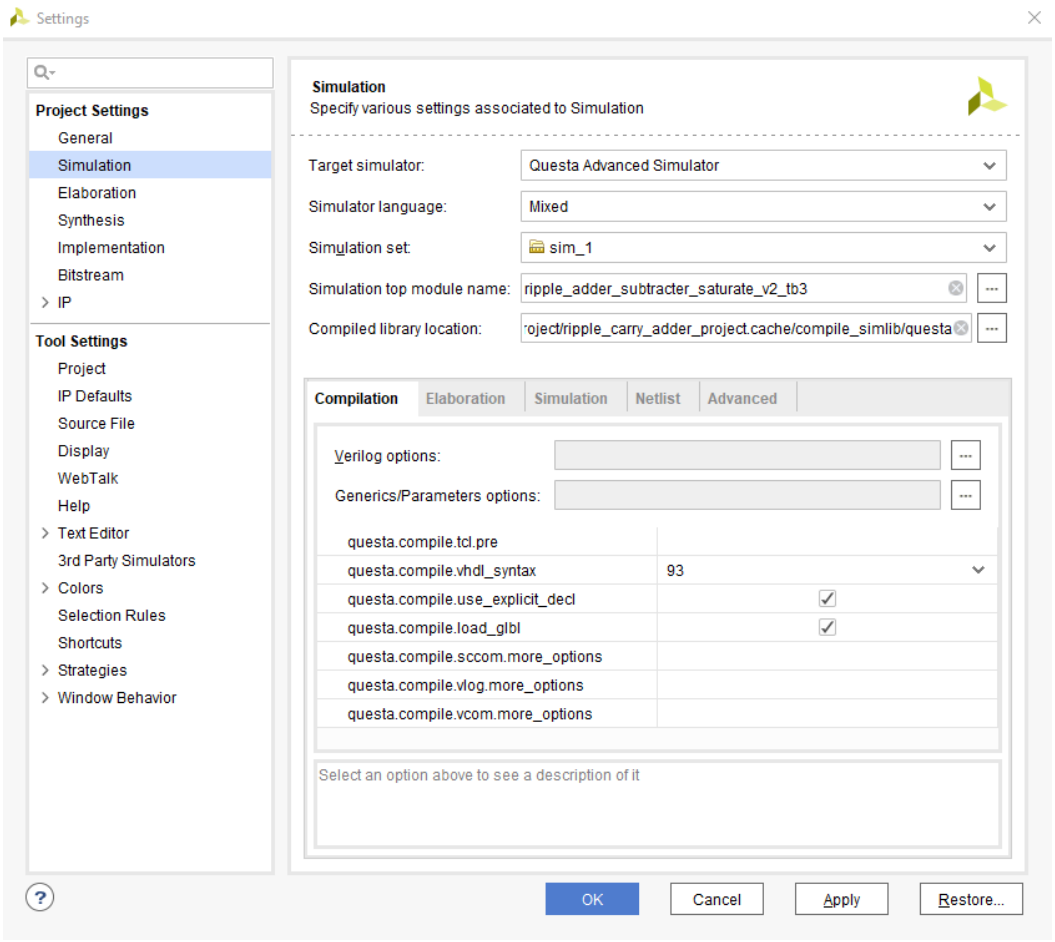


Figure 26 Project settings - Simulation

Here we should make sure that the Target simulator is Questa Advanced Simulator, if you don´t prefer to use Vivados built-in simulator. We will not give any support for this though.
As mentioned before you should set Simulation language to Mixed.
Simulation top module name should be the name of the testbench if you´re running a behavioral simulation. If you simulate the synthesized or implemented design directly without a testbench, then it should be the top-level module of your design. Remember that the two simulations require different do files.
Before we can do a simulation with QuestaSim we must compile some simulation libraries for Xilinx parts that QuestaSim will use. This is already done for the PC´s in the lab so you only need to make sure that the search path Compiled library location is correct. In the lab computers, the libraries are compiled to the folder

`C:/questasim64_10.7a/compile_simlib`

In *Figure 26* the search path used in my computer is given.
If you activate the Simulation tab in the lower part of the Settings window there are two things to update here. First, we can set the name of the do file that we use for simulation at questa.simulate.custom_udo. Use the relevant do file from the earlier assignment.

Since the simulator will be called from some subdirectory to the project it will not find the do file if it´s placed somewhere else. The best way to overcome this is to include the search path to the do file when you declare the questa.simulate.custom_udo file Remember to include the .do ending in the file name. Since we included the simulation runtime in our do files you should remove the time value from questa.simulate.runtime. We leave the rest of the Settings options as they are for now, *Figure 27*.

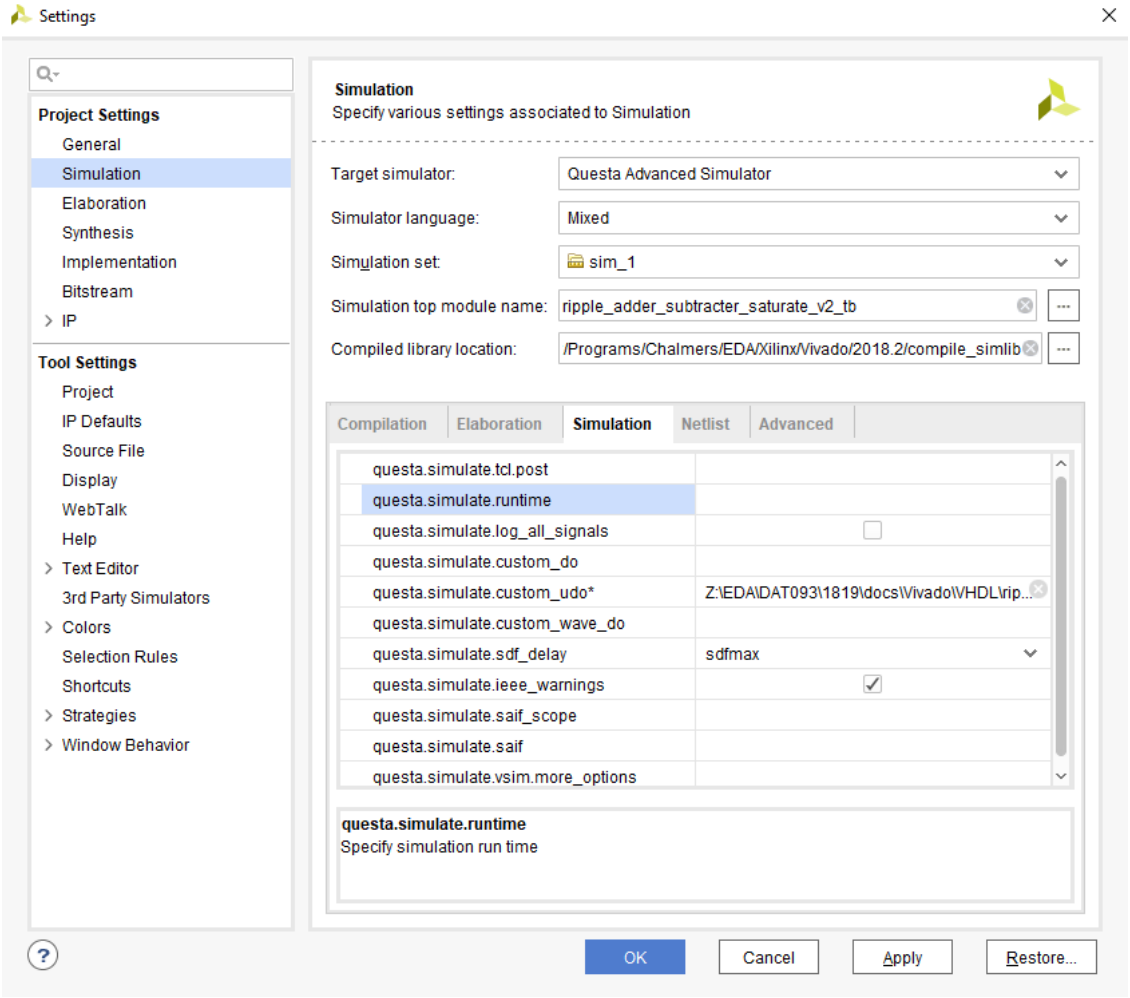


Figure 27 Project settings – Simulation settings

When we run the simulation Vivado will do some internal work and then call on QuestaSim to do the actual simulation.
Vivado has what I would call a bug and if the simulation doesn´t work out and there are some errors you will not see these during the simulation run, you will only see *Figure 28* running forever.

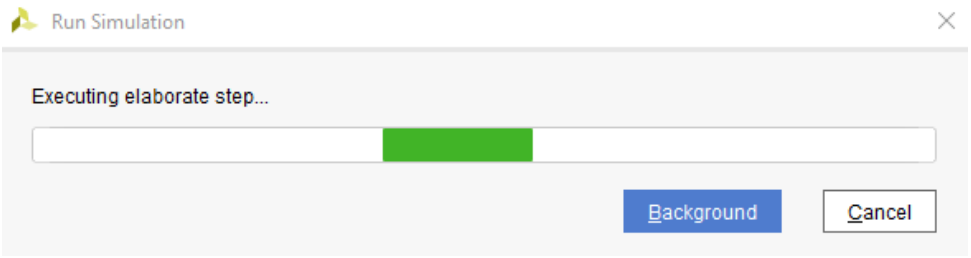


Figure 28 Simulation run

To find the errors you will have to cancel the simulation run and look for messages in the Console window.

Our next Settings option is Language Templates where we can find a number of examples of coding structures and configuration settings. In *Figure 29* the headings that are most relevant for us, VHDL and XDC, are opened.
Check out the templates and decide if you like to use some of them.

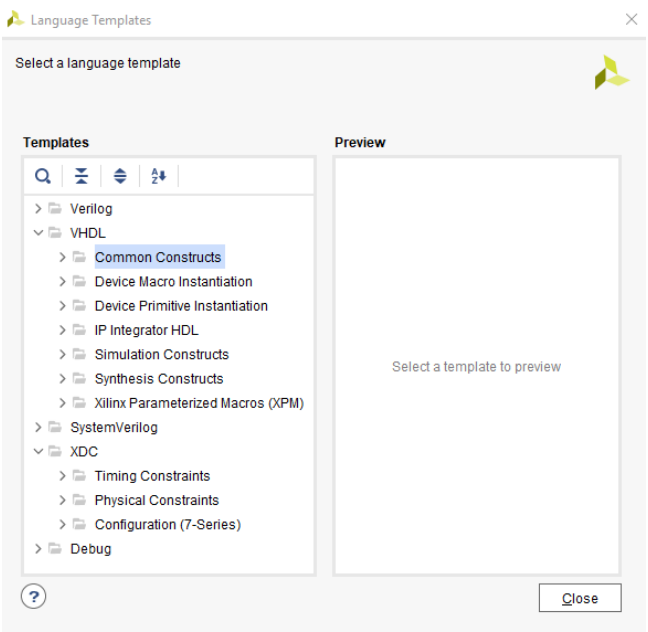


Figure 29 Language Templates

Next up is the IP Catalogue where we can find a large number of ready-made blocks, components, that we can use within our designs, *Figure 30*. Most of the components are highly configurable.
As an example, let´s look at a simple IP block, a Binary Counter placed in the group Basic Elements/Counters.

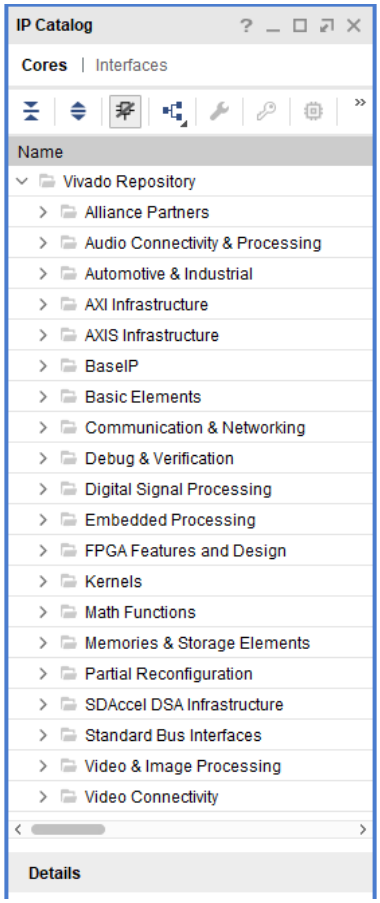


Figure 30 IP Catalog

If we double-click on the block we get *Figure 31*.
Here we can set some configuration. We can set the number of bits, the increment of the count and the direction of the count. If we check Loadable we can in parallel mode load a value into the counter. If we check Restrict Count, we can set the highest count value.

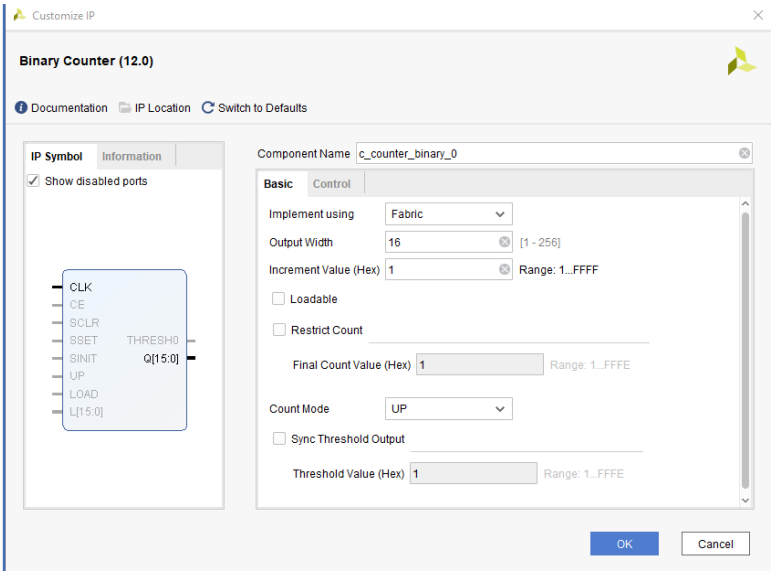


Figure 31 IP block for binary counter - Basic

Note that activating more options will activate more port on the component shown in the IP Symbol frame.
There are more settings in the Control tab, *Figure 32*. Here we can activate a enable signal and select if some signals should be synchronous or asynchronous.
There are many useful IP blocks so check them out.

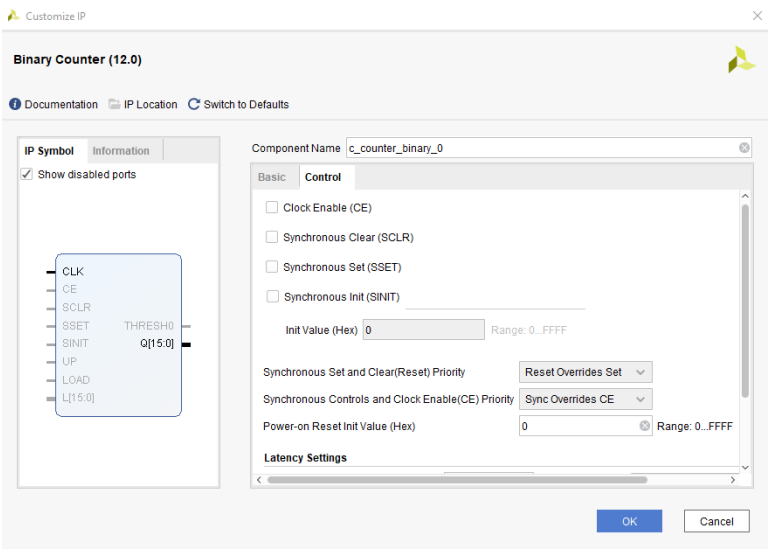


Figure 32 IP block for binary counter - Control

Let´s go back to the Flow Navigator, *Figure 23*. We´ll leave out the IP INTEGRATOR heading since we don´t need it for now.

Simulating the design

Next is the SIMULATION heading. If we right click on it we get the same simulation settings as the ones we saw under the Settings heading, *Figure 26*.
The heading has only one option Run Simulation. If we click it we get five options

- Run Behavioral Simulation
- Run Post-Synthesis Functional Simulation
- Run Post-Synthesis Timing Simulation
- Run Post-Implementation Functional Simulation
- Run Post- Implementation Timing Simulation

Since we haven´t done any synthesis yet, all option other than the behavioral simulation are dimmed out.
If you run the behavioral simulation QuestaSim should show up, run the simulation and execute the specified do file. When you start QuestaSim from Vivado it will start up from some subdirectory to the project. What subdirectory that is used is dependent on what type of simulation you are running. This makes it a little troublesome to execute do files from the command line in the Tcl Console. The do file is most likely in some other folder and it won´t be found when running the command. The easiest way to overcome this is to include the full search path to the do file when you execute the command.
In every step, when you run on of the tools, it´s essential to have the TCl Console tab open. If another tab is open, then we might miss some of the warnings and error messages and in some situations, the system gets stuck and it just continues running without information of what is actually happening until you eventually give up and press Cancel.

It´s also a good idea to right click on the Tcl Console and select Clear (Delete) before you start a tool. In this way, you are sure that all messages in the Tcl Console are from the current run of the tool and not from earlier runs.
We will not describe the handling of QuestaSim here since there is a separate document on this.
We will leave out RTL ANALYSIS, so the next step is to synthesis the design.

Synthesizing the design

Click on Run Synthesis under the SYNTHESIS heading and the synthesis will start. It will take some time to finish. It´s not that obvious that it´s running but you can see that at two places

- At the top right corner of the GUI you can see the text in *Figure 33* while the green circle is rotating.
- You can see the same thing in the Project Summery under Synthesis, *Figure 34*.

Figure 33 Synthesis is running

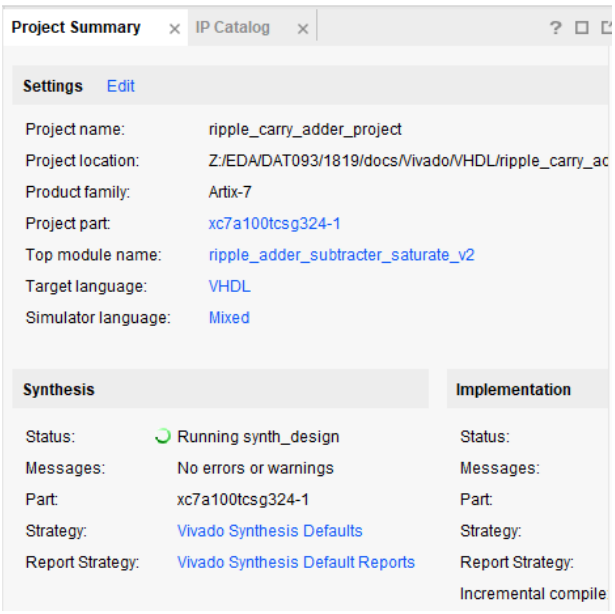


Figure 34 Project Summery while synthesis is running

When the synthesis is finished we get the dialogue in *Figure 35*.
We can move on and do the implementation, open and look at the synthesized design or look at the synthesis reports.
After doing the synthesis we have two more simulation options

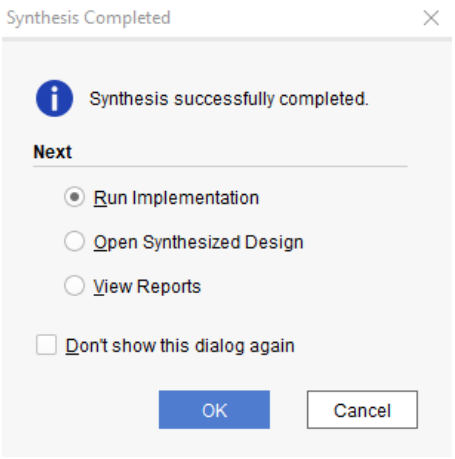


Figure 35 Successful synthesis

- Run Post-Synthesis Functional Simulation
- Run Post-Synthesis Timing Simulation

The functional simulation will show the behavior without taking any timing issues into consideration. The timing simulation will also show the timing of the synthesized design and this simulation will obviously take longer time. Some of the timing issues that might show up are not relevant for the design we´re using as an example since it´s a concurrent design and there is no clock signal.
After synthesis, we have the GUI in *Figure 36*.

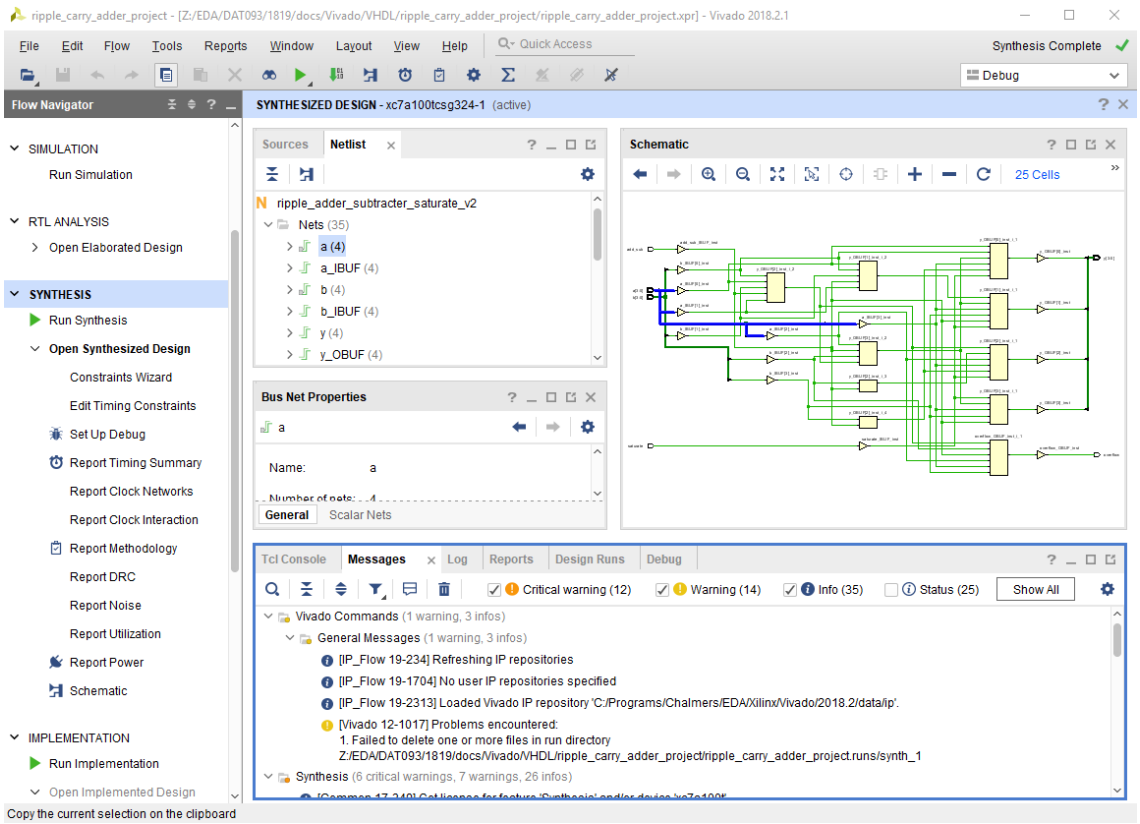


Figure 36 Main window after synthesis

Here we can see the synthesized schematic. Not that this is not yet placed in the FPGA device, so we only see primitive design blocks, not the blocks that are actually used within the FPGA. By clicking on a net in the Netlist menu we activate the connection in the schematic. In *Figure 35* we see the signal a.
We can also see a menu under the SYNTHESIS heading in the Flow Navigator, *Figure 37*.
We will not dive into this, but you should explore it on your own.

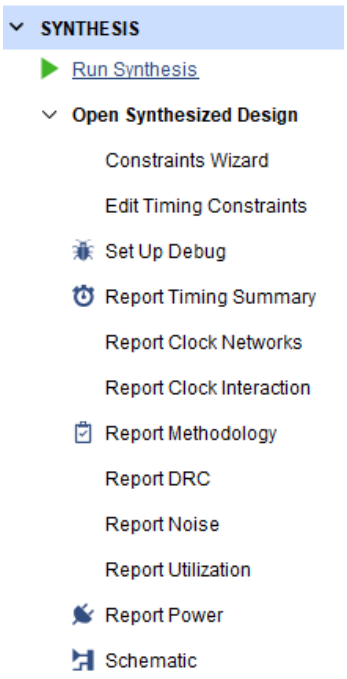


Figure 37 Flow Navigator after synthesis

Implementing the design

The next step is to implement the design. The step is visually similar to the Synthesis step. After implementation we get the dialogue in *Figure 38*.

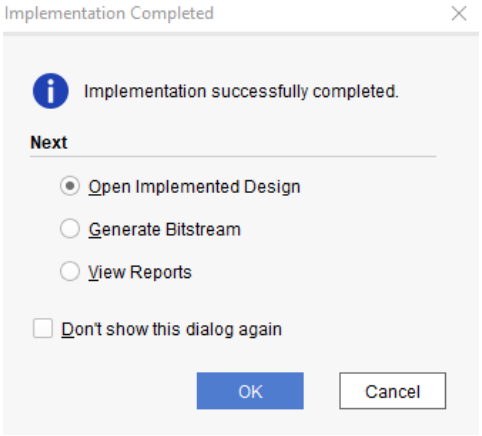


Figure 38 Successful implementation

The resulting GUI looks very similar to the synthesized one with the exception that instead of the synthesized schematic we see the internal of the FPGA device, *Figure 39*.

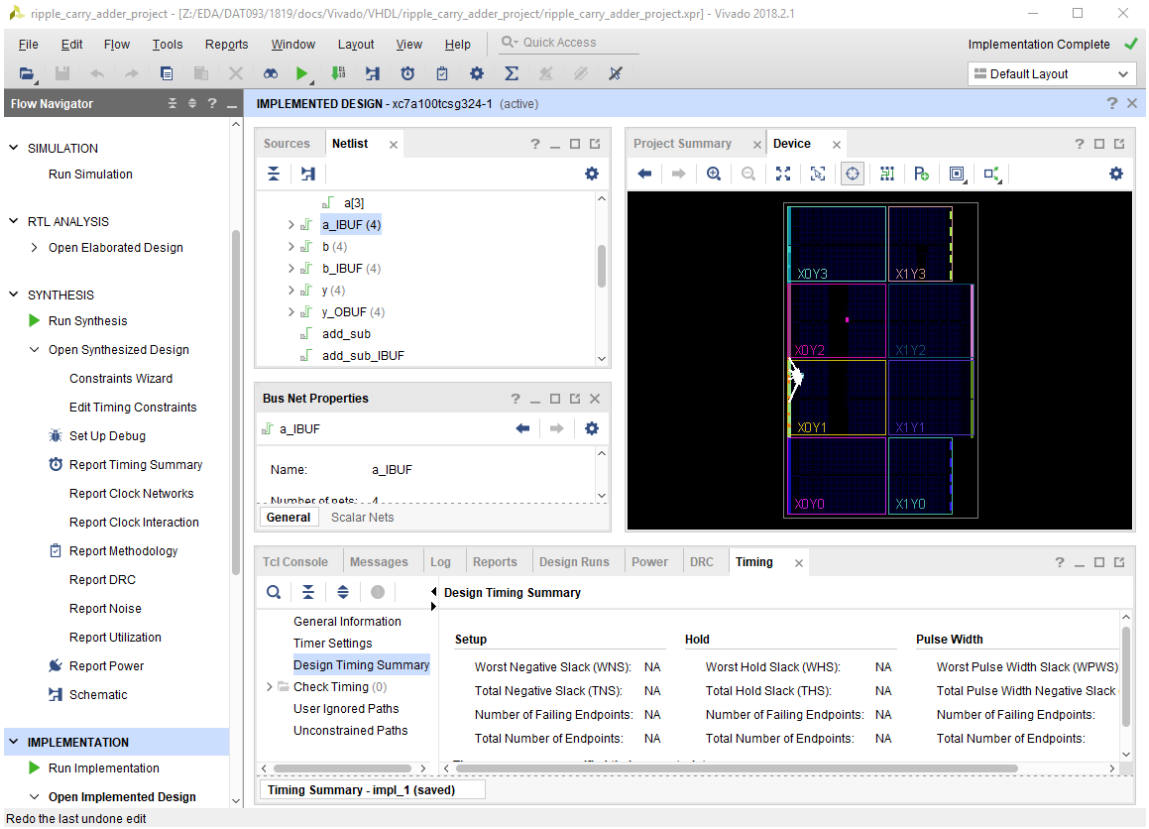


Figure 39 Main window after implementation

If we click on one of the signals in the Netlist we the connection in the device. In *Figure 39* a_IBUF is activated, this is a connection between the a ports and the internal logic.

Downloading to the FPGA

We move on in the Flow Navigator and create a bit-stream to be downloaded to the FPGA. We select PROGRAM AND DEBUG/Generate Bitstream, *Figure 40*.

As a result, we get *Figure 41*.

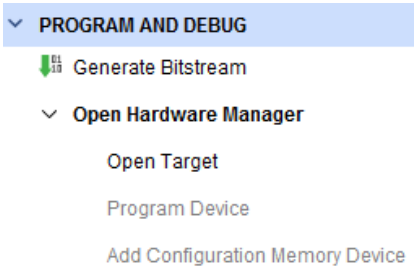


Figure 40 Program and debug

First, we open the Hardware Manager from the Flow Navigator.To use the bitstream we will have to be connected to the Nexys4 board and for this to work the board must be connected to the PC and powered using an USB connection. We click on Open Target and get *Figure 42*.

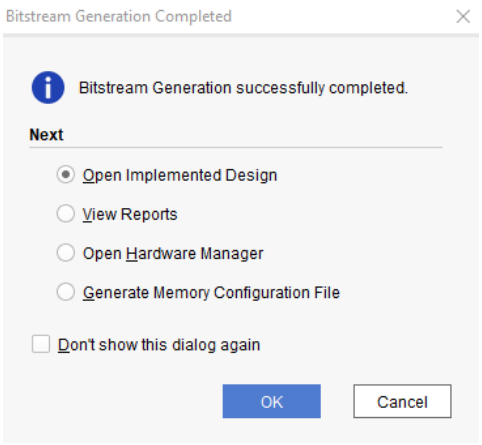


Figure 41 Successful bitstream generation

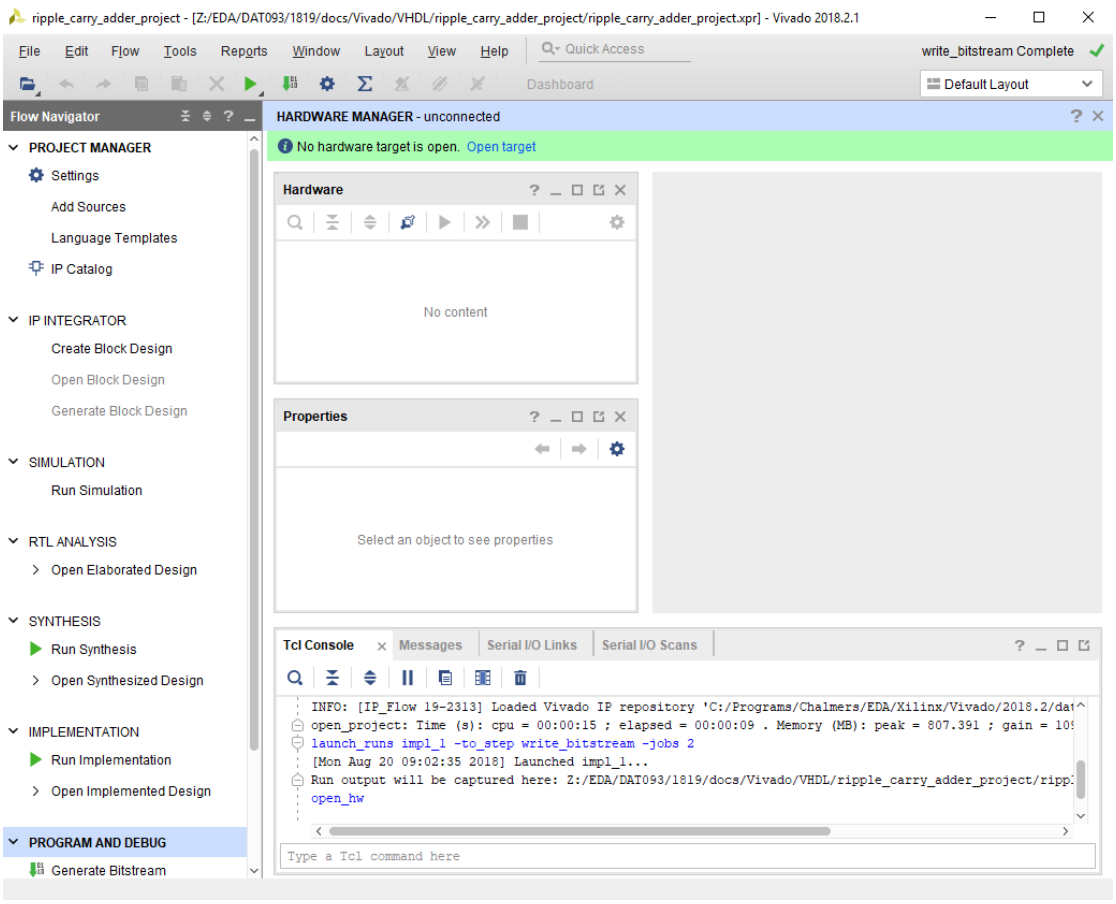


Figure 42 Hardware manager, Open target

To connect to the board, we select Open target in the green field close to the top of the window and then Auto Connect in then popup menu that appears and then the program will detect the FPGA connected, *Figure 43*.

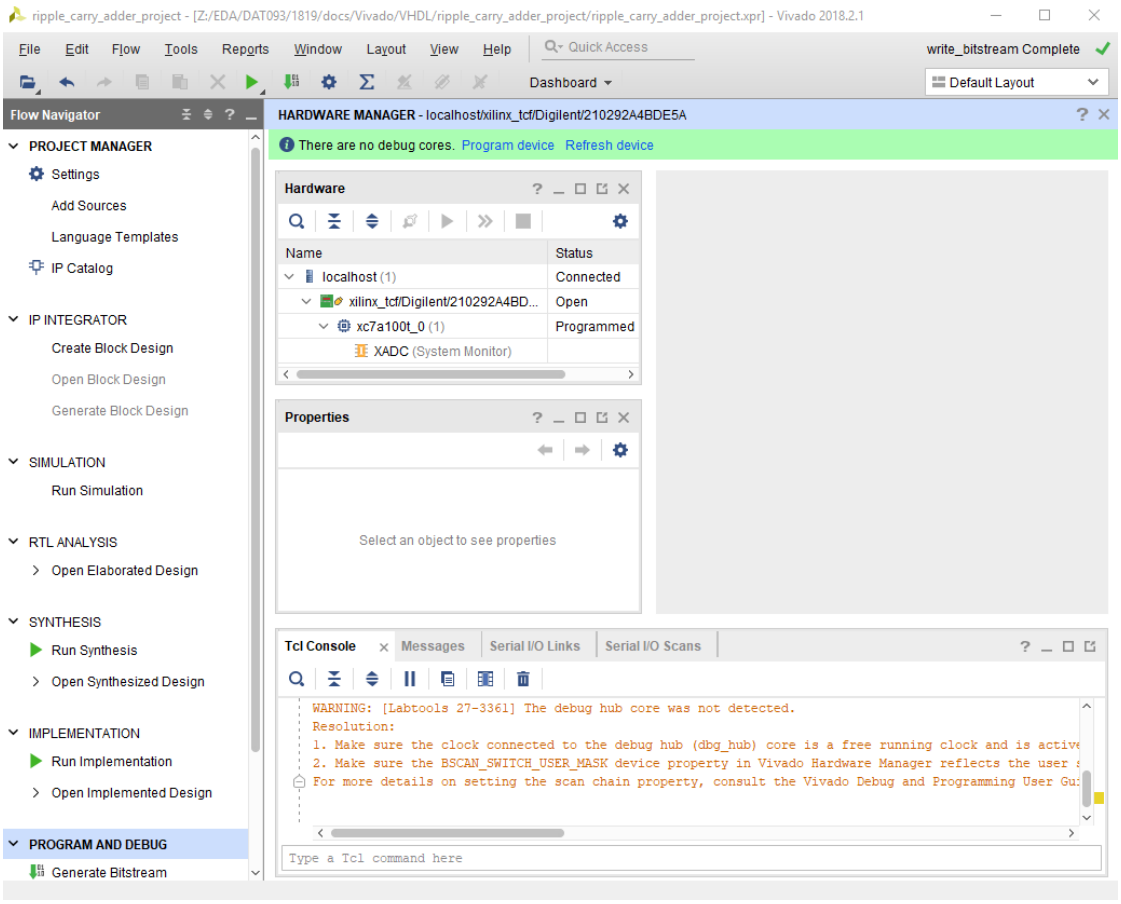


Figure 43 Hardware manager, Program evice

We will get a Hardware frame showing the detected device.
Now it´s time for programming the device so we click on Program Device in the Flow Manager.

We get a window where the bitstream file for the project is automatically selected, *Figure 44*. It will have the same name as the top-level file of the design but with the ending `.bit`.

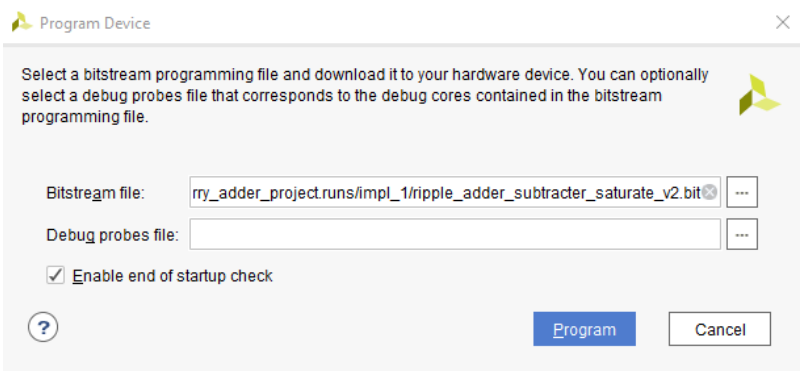


Figure 44 Programming device

Click on Program and the download starts. There is also an external memory device on the Nexys4 board that can be programmed. We will not use this memory for now.
After download the configuration is immediately implemented, and the FPGA is functional. Now we can test the physical result and not just do simulations.

The Nexys4 board

As stated earlier the Nexys4 board houses a Xilinx Artix-7 FPGA with the name the xc7a100tcs314-1. The board also contains a number of peripheral devices, *Figure 45*. You can find the reference manual for the board on the PingPong homepage.
There are two versions of the board; the Nexys4 and the Nexys4DDR. The difference being that the later has DDR memory onboard.
The setup is different in the two cases, so they will need different constraints files (XDC).
The board have the following peripheral devices

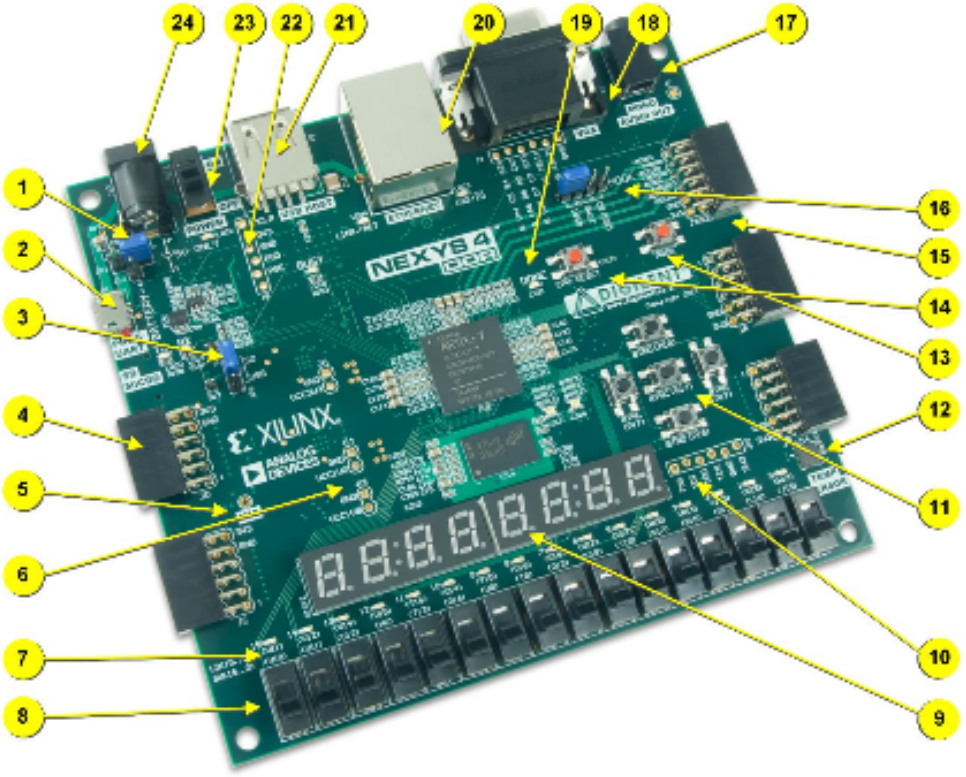


Figure 1. Nexys4 DDR board features.

Callout	Component Description	Callout	Component Description
1	Power select jumper and battery header	13	FPGA configuration reset button
2	Shared UART/ JTAG USB port	14	CPU reset button (for soft cores)
3	External configuration jumper (SD / USB)	15	Analog signal Pmod port (XADC)
4	Pmod port(s)	16	Programming mode jumper
5	Microphone	17	Audio connector
6	Power supply test point(s)	18	VGA connector
7	LEDs (16)	19	FPGA programming done LED
8	Slide switches	20	Ethernet connector
9	Eight digit 7-seg display	21	USB host connector
10	JTAG port for (optional) external cable	22	PIC24 programming port (factory use)
11	Five pushbuttons	23	Power switch
12	Temperature sensor	24	Power jack

Figure 45 The Nexys4 board

- 16 slide switches giving high and low-level signals
- 16 LEDs
- 2 tri-color LEDs
- 8 7-segment displays
- 5 non-latching bush buttons
- 4 8-bit I/O ports with ground and 3.3 Volt supply
- 1 8-bit port with connections to the ADC in the FPGA
- VGA connector
- Micro SD connector
- Accelerometer
- Temperature sensor
- Microphone
- PWM audio amplifier
- USB-RS232 interface
- USB HID
- Ethernet PHY
- External memory

For more details on the peripheral devices consult the reference manual.

The constraints file (XDC)

The constraints file is where you set up the rules for the synthesis and implementation. You can set up a lot of constraints concerning timing and placement within the FPGA. We will leave those out for now and focus on the physical external configuration, that is how the ports of our design with their peripherals are connected to the pins of the FPGA device. It´s quite complicated to write a constraints file from scratch. Luckily there are ready-made templates for constraints files for many of the evaluation boards that vendors supply, this includes the Nexys4 boards that we are using.
The Nexys4 board is no longer in production but is replaced by the Nexys4DDR board. We still have some Nexys4 boards, so we will take a look at the constraints file for both versions of the boards since they have different pinning. The two XDC files are also somewhat differently structured but they will function the same way. You can find both files on the PingPong homepage. The template constraints files are called

`Nexys4_Master.XDC`

and

`Nexys4DDR_Master.XDC`

Both the XDC files have the pins split into groups according to the peripheral list we just saw. All pins are declared in the file with default names for the connections. All lines are commented out and you should uncomment the lines for the pins you are using and give the pins the names you are using for the ports in your design. Don´t uncomment any lines for pins that you are not using.

The Nexys4 constraints file

In the constraints file for the Nexys4 board each pin is defined by two lines. The first line gives the name of the FPGA pin and the name of the port in the source code that is connected to that pin. The second line sets the signal standard for the pin and the name of the port in the source code that is connected to that pin will have to be given here to.
Let´s look at the first pin in the file, the clock pin that is connected to the system clock. This pin will have to be activated in all your designs.

```
#set_property PACKAGE_PIN E3 [get_ports clk]
     #set_property IOSTANDARD LVCMOS33 [get_ports clk]
```

The hash signs (#) indicates that the lines are commented out. `clk`, highlighted in green, is the name of the port in the source code that should be used, or the name should be changed to the one that is being used. The rest of the lines can stay as they are, but you should remove the hash signs on both lines of course. So, what you need to do to use this pin is take away the hash signs and if needed edit the green name twice.
When I do this, I keep the commented lines as they are and make copies of them just below. I edit these copies and take away the hash sign from them. The reason for doing this is that this way I still have the template for the lines if I do some mistake when I edit the lines.
Let´s look at one more line that use a port that is part of a vector

```
#set_property PACKAGE_PIN U9 [get_ports {sw[0]}]
     #set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
```

This defines the connection to switch 0 out of the 16 switches on the board. Notice that square brackets should be used for the index, not ordinary parentheses. The curly brackets around the pin name can be left out.
If you name a single pin connection, not a vector, then you name it the same way as the `clk` pin.

The Nexys4DDR constraints file

In the constraints file for the Nexys4DDR board is very similar to the constraints file for the Nexys4 board but the syntax is different. Let´s look at the same pins above. We start with the clock pin

```
#set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 }
[get_ports { CLK100MHZ }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
```

In the file this is just one line but here it´s split into three to fit on the page. In this case the only thing that might be necessary to change is the name of the port from the source code, by default it´s called `CLK100MHZ`, highlighted in green. You will also have to remove the first hash sign of course; the second hash sign should not be removed.

You can change the name `clk100mhz` at the end but since it´s just a comment it is not necessary.
Now to the line for switch 0

```
#set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 }
[get_ports { SW[0] }];
#IO_L24N_T3_RS0_15 Sch=sw[0]
```

The same kind of editing as for the clock is needed. The name to be changed is highlighted in green. Once again notice the square brackets and the curly brackets.