Tutorial Vivado 2020.2

coralie.allioux

May 2022

1 Tutorial scope

This tutorial explains how to use the graphical interface of Vivado 2020.2 to:

- Create a new IP with VHDL sources and package it
- Create a project including the ZYNQ process unit and the custom IP
- Synthesize, implement and generate bitstream and useful files

2 Download and installation

The link where to download Vivado is the following: https://www.xilinx.com/support/download.html

This tutorial is done for **Vivado 2020.2**, which is already considered as an archive.
This version was chosen because of compatibility with **PYNQ 2.7**.

Note: *Vivado is available for Windows and Linux. Be careful, it asks a lot of space and the installation requires some hours.*

3 Setup

The PYNQ-Z1 board description is not inside the default database of Vivado: it needs to be added manually.

First, download the description file of the board at the following link, in the section **Vivado board files**: https://pynq.readthedocs.io/en/v2.0/overlay_design_methodology/board_settings.html

3.1 For version 2020

Once the download is done, put the unzip directory at location
`<Xilinx installation directory>/Vivado/<version>/data/boards/board_files/`.

3.2 For version 2021

On Windows, in `%APPDATA%/Xilinx/Vivado/`, create the file `init.tcl` if does not already exist. Then, write the following line inside that file:

```
set_param board.repoPaths [list "<your_full_path_to_board_files>]".
```

Note: *Do not use back-slashes in your path, forward-slashes ONLY.*

4 Create a project

This section shows how to create and setup a new project, in order to then be ready to add a personalized IP.

4.1 Create RTL project on PYNQ-Z1 board

- STEP 1: After opening Vivado 2020.2, Create a *RTL project* without specifying any sources. Then click *Next*
- STEP 2: Select the *PYNQ-Z1* board: that board should appear in the list, since the board files have been added in section 3
- STEP 3: Click on *Next* until *Finish*.

The project should be now created, as shown on Figure 3.

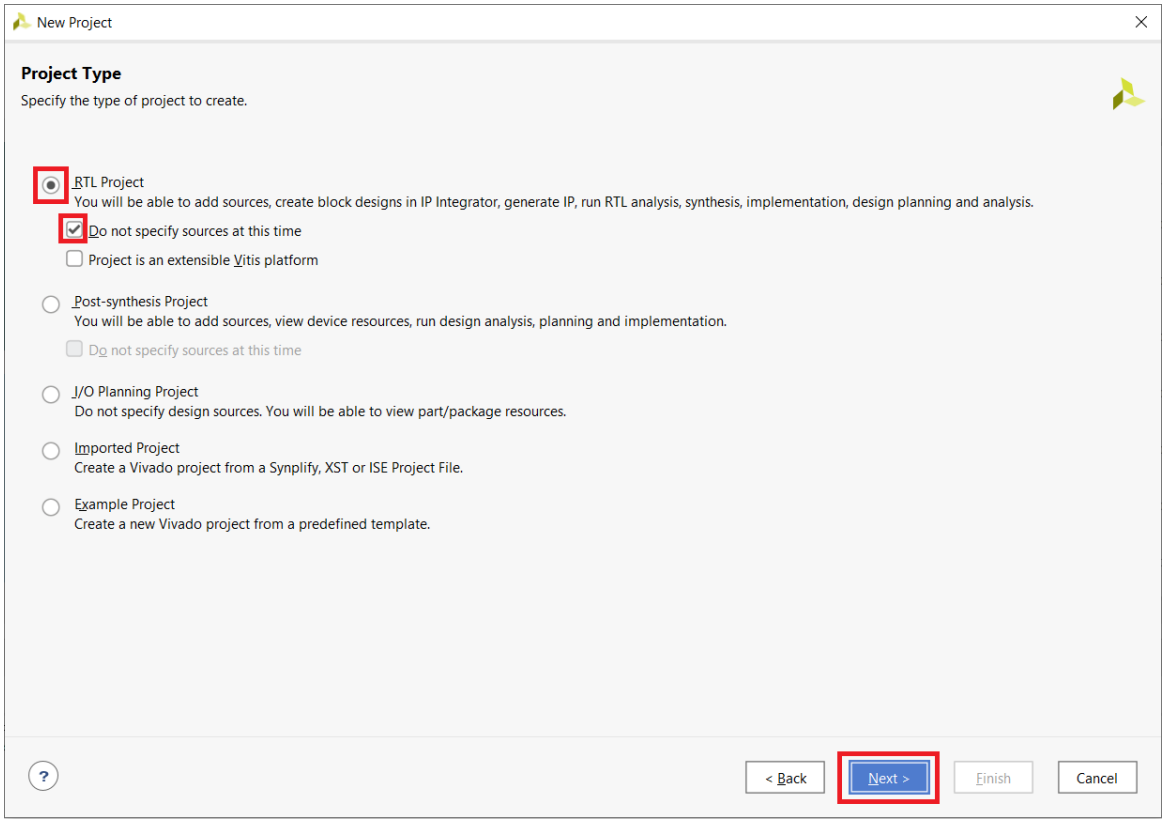

Figure 1: Create a *RTL project* without specifying any sources. Then click *Next*

4.2 Create Block Design

In order to connect IPs, Vivado uses a **Block Diagram**. The entire diagram will be filled in section 6, after having built the personalized IP, as described in section 5.

Note that in Figure 3, the current RTL language used by the project is *Verilog*: it will be replaced by *VHDL* in section 4.3.

- STEP 4: Create a block diagram by clicking on *Create Block Design*, located on the left menu under **IP Integrator**.

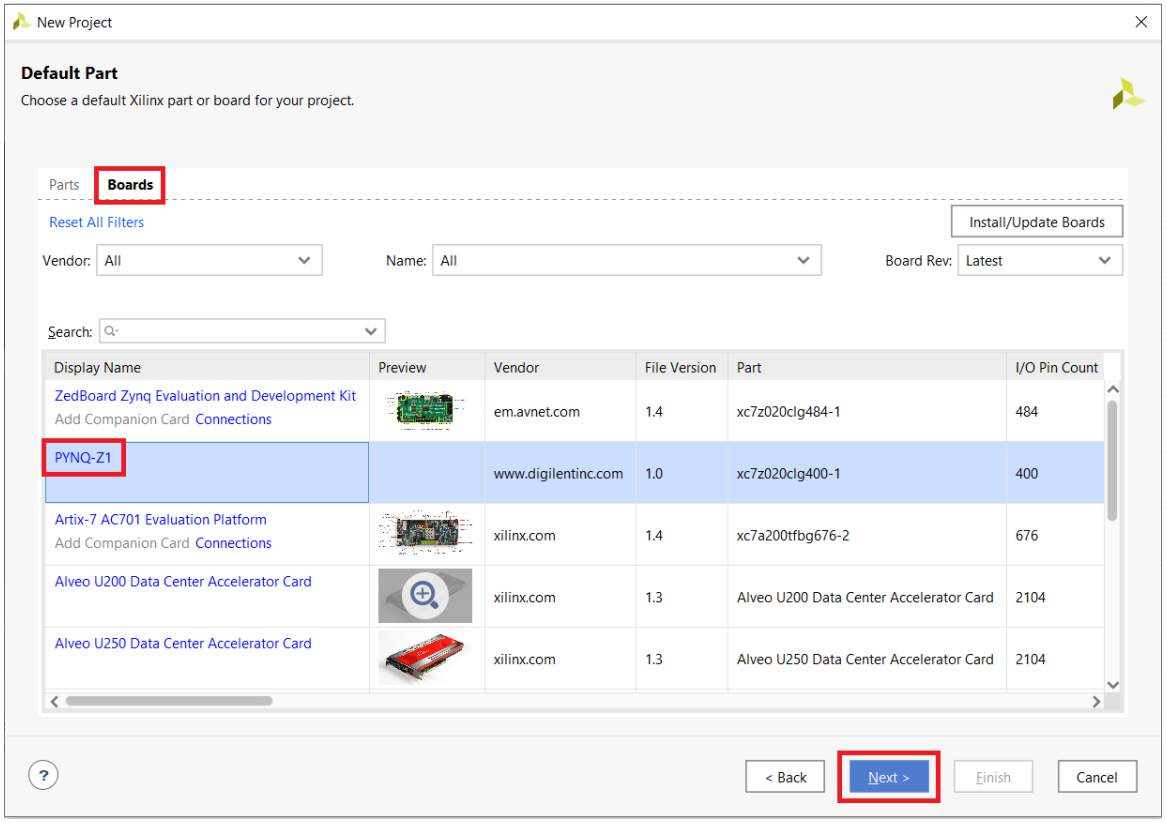

Figure 2: Select the *PYNQ-Z1* board

As shown on Figure 4, the current view of Vivado is now focused on the created blank block design.

4.3 Change RTL language from Verilog to VHDL

This change allows to use sources in VHDL, ie using VHDL description to create a personalized IP.

- STEP 5: to change it, Go to *Tools* menu and then *Settings*
- STEP 6: Click on *General* section and select **VHDL** as target language instead of **Verilog**

5 Create a new IP

The IP will package the challenge that needs to be embedded on the Pynq-Z1 board. This section describes:

- How to create a personalized IP by using a AXI4 peripheral
- How to import VHDL sources
- How to integrate those sources inside the AXI4 peripheral

This part is based on a specific example: an IP implementing a simple synchronous adder component. This component makes the sum of two 32-bit inputs `a` and `b`, given by the 32-bit output `q`. Inputs are sampled with the signal `clk`. If `rst` is up, then `c` keeps the value 0.

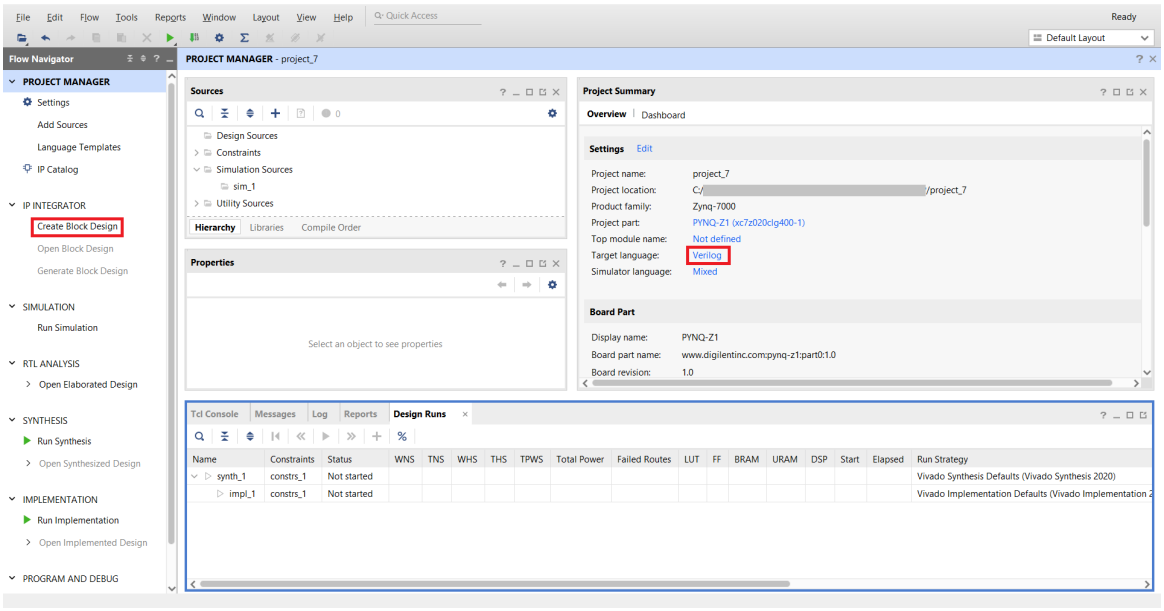

Figure 3: Create a block diagram by clicking on *Create Block Design*, located on the left menu under **IP Integrator**.

This component must then be included inside a *AXI4 peripheral*: the interconnection between the two is depicted on Figure 7. The AXI4 peripheral is used as an interface between the adder component and the Zynq processing unit of the board. For more details on the entire block diagram, please refer to section 6.

The interconnection between the component and the AXI4 peripheral is done by mapping the component signals to the peripheral registers. Each signal of the adder component must be mapped to a 32-bit register of the AXI4 peripheral. The only exception is the clock, which will be directly mapped to the internal clock of the peripheral, and not to a register.
In this particular example, only four registers are needed to fully map the adder component.

Moreover, each register of the AXI4 peripheral can be renamed. Better to do so because those labels will be visible by the python library. Indeed, instead of interacting with `slvregX`, it is more friendly to associate it to a known label.
The chosen mapping is as followed, renaming each register as done on Figure 7:

- input `a`: register `a`
- input `b`: register `b`
- output `q`: register `c` *(note that the names can be different)*
- reset signal `rst`: bit 0 of register `rst` *(note that the size can differ, ie map a signal to bits and not to an entire register)*
- clock signal `clk`: internal clock `S_AXI_ACLK`

Note: *The peripheral characteristics can be modified, as the number of registers available. When possible, it will be noticed during the following sections.*

5.1 Create a new IP project

- STEP 1: Got to *Tools* menu and then *Create and Package New IP*

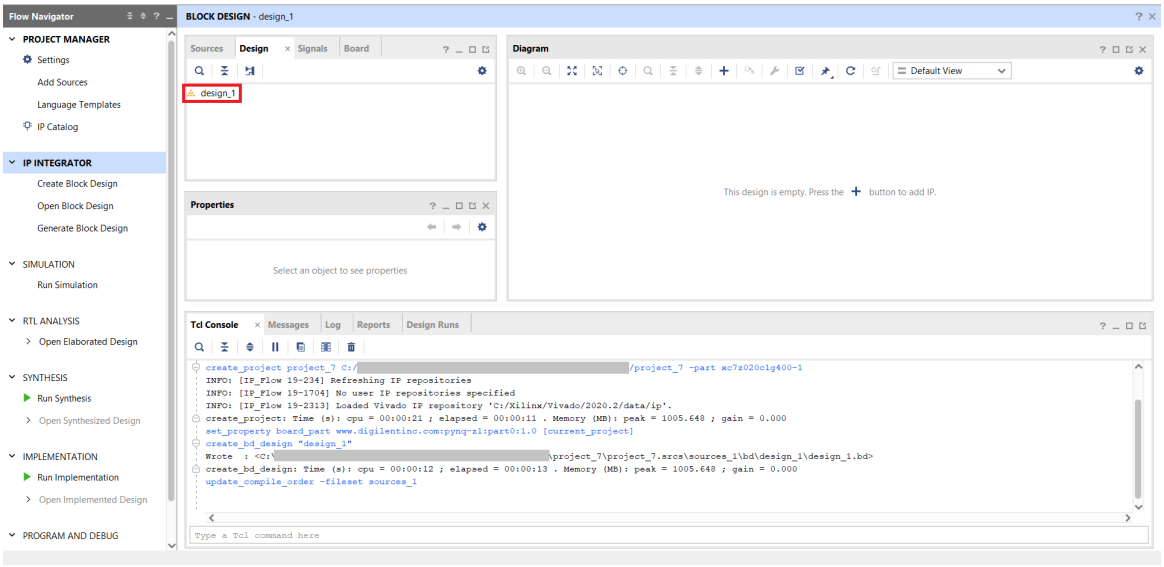

Figure 4: Blank block design

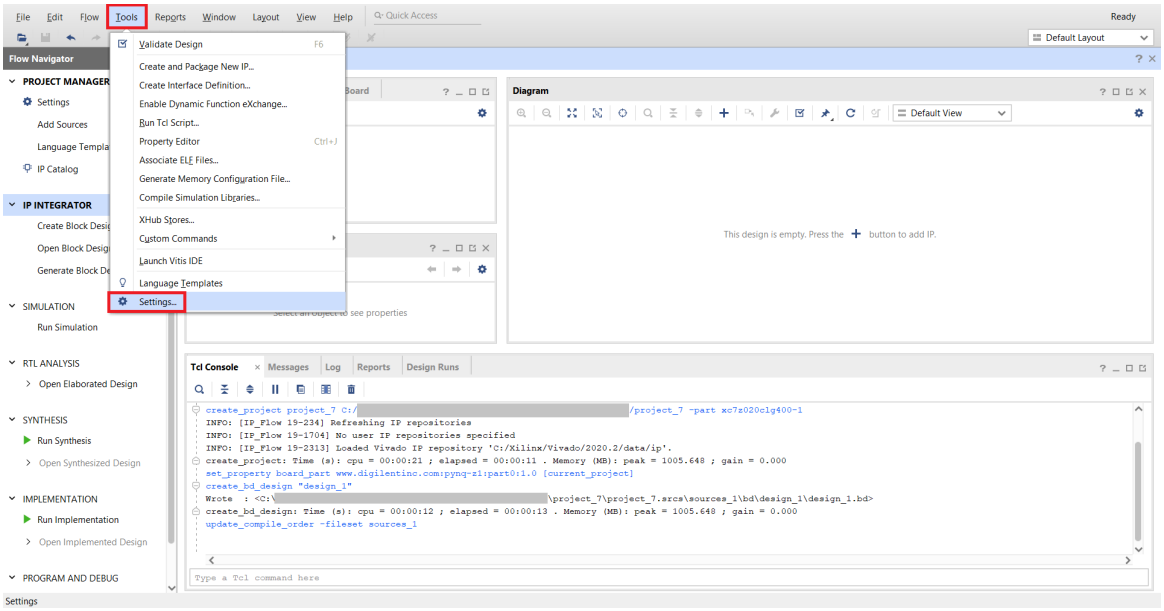

Figure 5: Go to *Tools* menu and then *Settings*

- STEP 2: Select the option *Create a new AXI4 peripheral* and then click *Next*
- STEP 3: Let the default interface for the AXI4 and click *Next*. In this window, the characteristics of the peripheral can be modified, in particular: *size of registers*; *number of registers*. **Adapt it to your design!**
- STEP 4: Select *Edit IP* in order to define it right away and finally click *Finish*

As shown on Figure 12, the blank IP should be correctly created.

Note: *You can name this IP as you want. Here the chosen name is* `adder_example`.

5.2 Adding vhdl sources

In those steps, the VHDL sources of the *adder component* will be inserted inside the IP project. Of course, if your design is made of several components and files, **add all of them** inside the

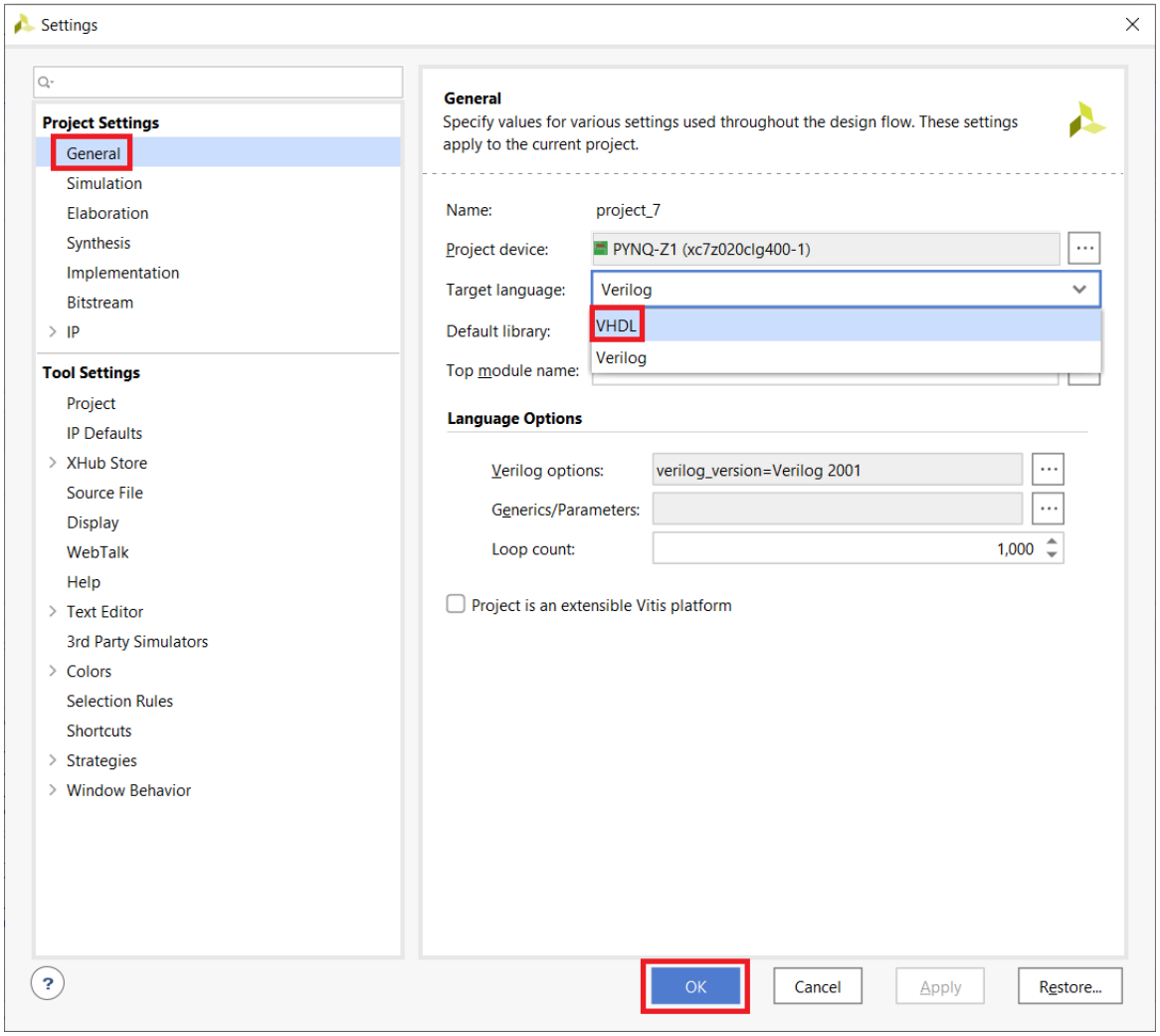

Figure 6: Click on *General* section and select **VHDL** as target language instead of **Verilog**

project.

- STEP 5: Click on the *plus* button inside the *Sources* box
- STEP 6: Select *Add or create design sources* and then click *Next*
- STEP 7: Click on *Add Files*, search for your VHDL sources (here *adder.vhd*) and click *Finish*

Now looking on Figure 15, inside the *Sources* box should appear several files:

- `adder.vhd`: source file of the adder component, which was just added
- `adder_example_v1_0.vhd`: main file which describes the overall IP
- `adder_example_v1_0_S00_AXI.vhd`: file which describes the AXI4 peripheral

The last cited file will be modified in section 5.3 to integrate the adder component inside the AXI4 peripheral.

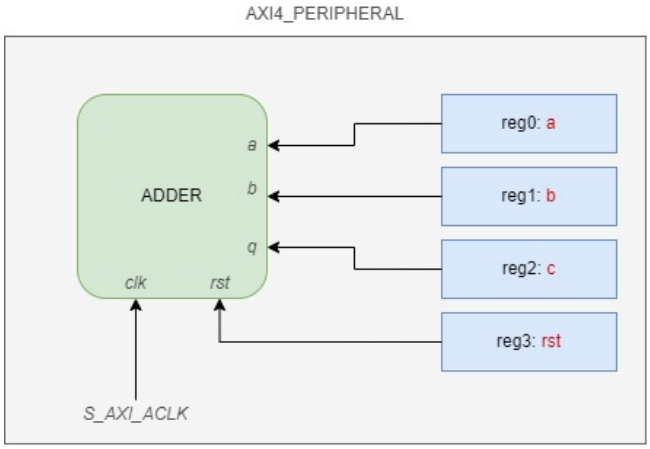

Figure 7: Connection of the adder component within the AXI4 peripheral

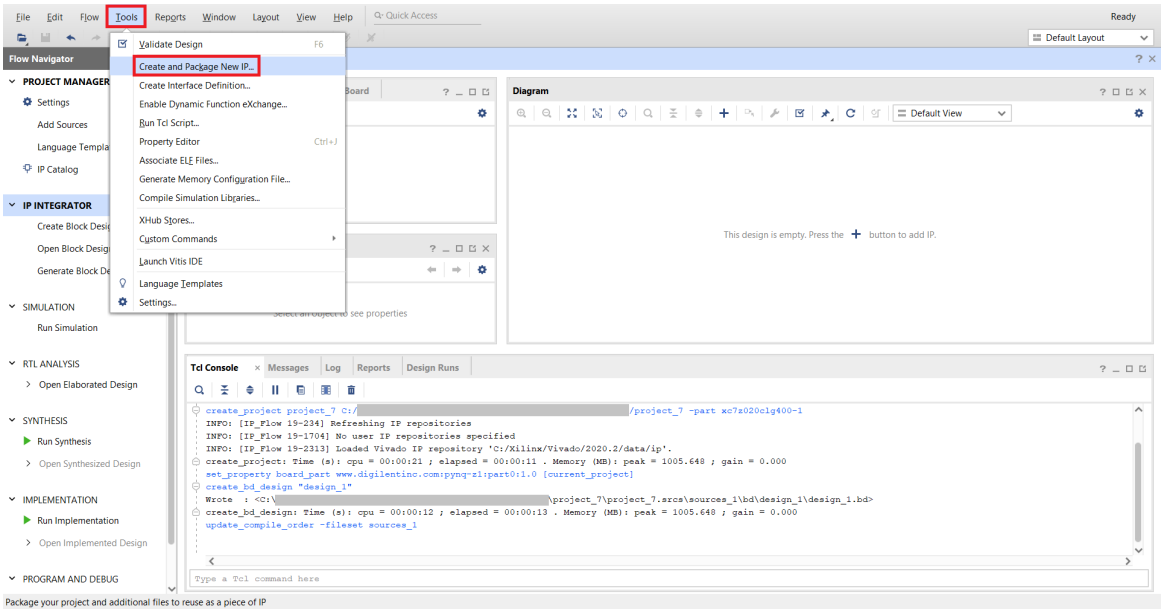

Figure 8: Got to *Tools* menu and then *Create and Package New IP*

5.3 Integrating adder component inside IP

Here the source code of the AXI4 peripheral is modified in order to integrate the adder component. The mapping is the same depicted on Figure 7, explained earlier in section 5.

- STEP 8: Open the file *adder_example_v1_0_S00_AXI.vhd* by clicking on its name in the *Sources* box: it appears on the editer on the right
- STEP 9: Declare the *adder* component by copying it before the keyword *begin*
- STEP 10: Rename the slave registers with more friendly names
- STEP 11: Change the outputs depending on the behavior of your circuit
- STEP 12: Add the associated logic of the component at the end of the file
- STEP 13: Save the modified file and go to menu *Package IP - adder_example*, still in the right editer.

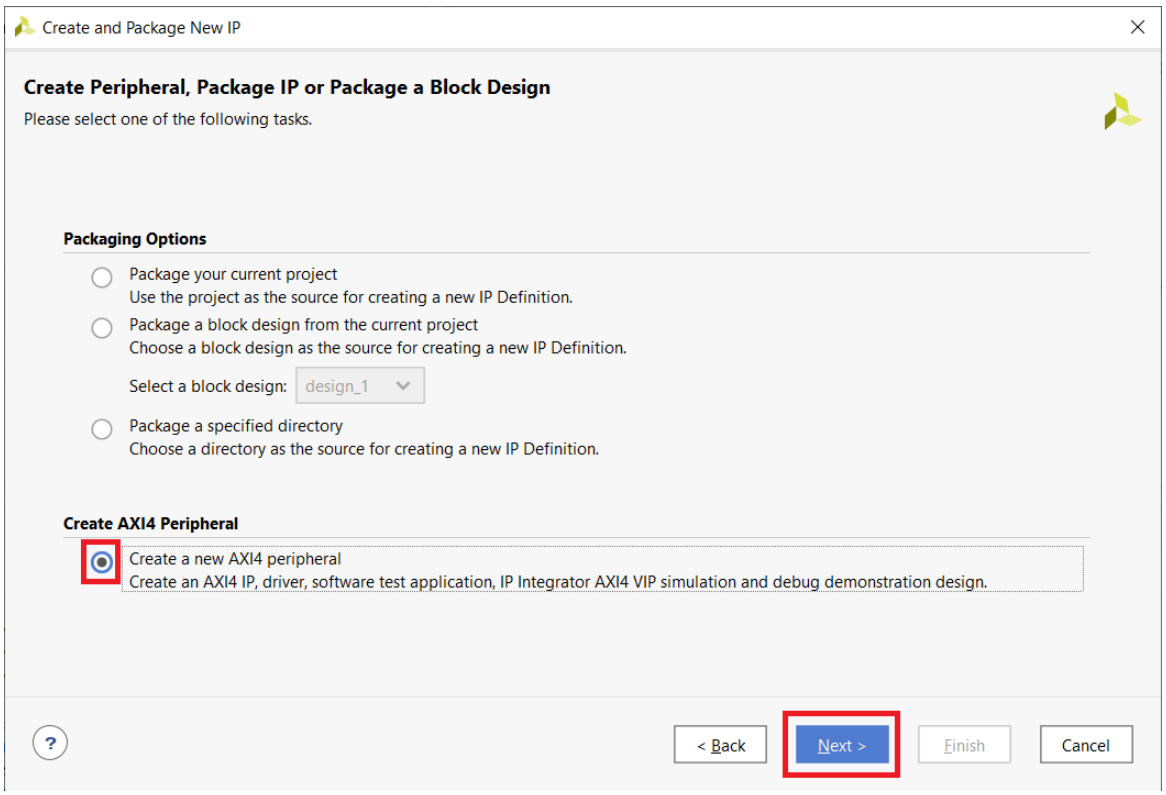

Figure 9: Select the option *Create a new AXI4 peripheral* and then click *Next*

5.4 Update IP files

Now that the source file for the peripheral is ready, the package must be updated.

- STEP 14: While in *Package IP - adder_example*, click on *File Groups*; then click on *Merge changes from File Groups Wizard*; to take into account the changes

5.5 Adressing IP registers

The used registers within the peripheral must now be addressed, so that to be accessible by the python interface.

- STEP 15: Go to *Addressing and Memory*, and add a register to *S00_AXI_reg* by right clicking on it and select *Add Register...*
- STEP 16: Rename the added register, update the address offset and the associated size. *Be careful, use the names you have chosen in STEP 10.*
- Repeat STEP 15 and 16 for all registers of the peripheral (ie, the ones renamed in the previous vhdl file)
- While done for every register, the final configuration should be the same as the one shown on Figure 22

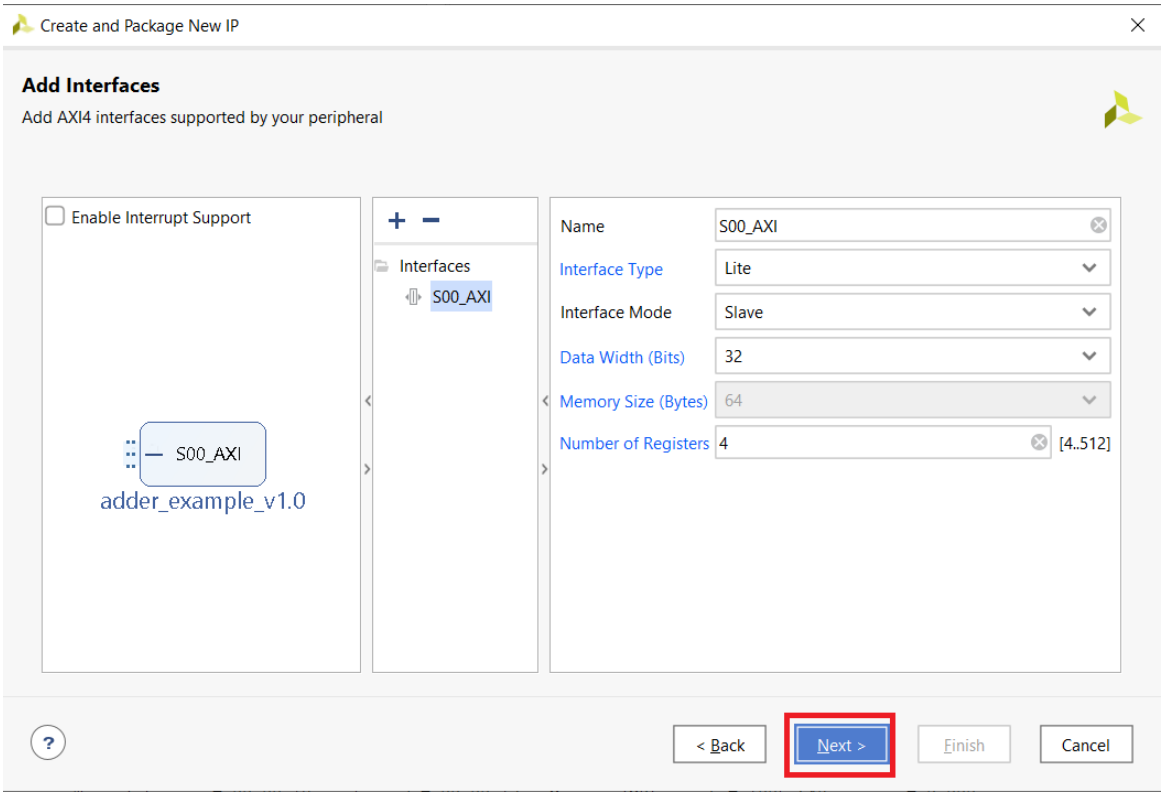

Figure 10: Let the default interface for the AXI4 and click *Next*

5.6 Package IP

Finally to conclude the overall configuration of the IP:

- STEP 17: Got to *Review and Package* and click on the *Re-Package IP* button
- STEP 18: Close the IP project and go back the the blank block design created in section 4.2

6 Modify Block Design

In this section, the Block Design is fully implemented. It connects the previously created IP *adder_example*, in section 5, to the Zynq processing system.

- STEP 1: Click on the *plus* button in *Diagram* box and search for *Zynq7 Processing System*; add this IP by double clicking on it. The IP block should appear as in Figure 25
- STEP 2: Click on *Run Block Automation* to add automatically useful configuration. A pop-up window will appear, let the default configuration and click *OK*. The result of this operation is shown by Figure 26
- STEP 3: Click on the *plus* button in *Diagram* box and search for *adder_example_v1.0*; add this IP by double clicking on it. *If the name of your IP is different, search for that one.* The IP block should appear as in Figure 28
- STEP 4: Click on *Run Connection Automation* to automatically connect the Adder IP and the Processing System. A pop-up window will appear, let the default configuration and click *OK*. The result of this operation is shown by Figure 29

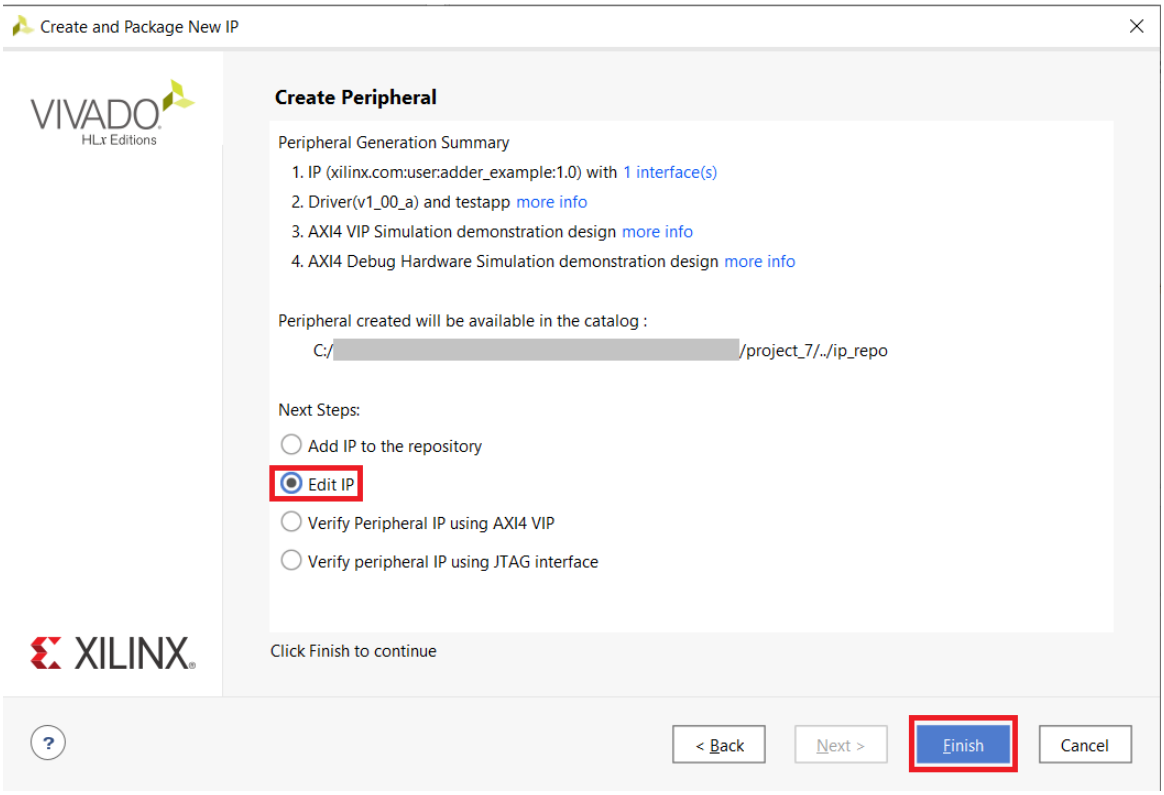

Figure 11: Select *Edit IP* in order to define it right away and finally click *Finish*

The Block Design is finally ready!

7 Add HDL wrapper

The entire block design needs to be wrapped into a **HDL wrapper**, in order to be synthesized:

- STEP 1: In the *Sources* menu, right click on your design (here called *design_1*) and select *Create HDL Wrapper*.
- STEP 2: A window appears: Select the option *Let Vivado manage wrapper and auto-update* and click *OK*
- STEP 3: Wait until the operation is done. Note then that the wrapper is now visible as shown on Figure 32

8 Synthesis, implementation and bitstream

At that step, the overall design is ready to be synthesized.

- STEP 1: On the left menu, under *Synthesis*, click on *Run Synthesis*
- STEP 2: A dialog window appears, Click *OK* to run the synthesis
- STEP 3: While the synthesis is successfully completed, a new dialog window ask for running the implementation. Click on *Run Implementation* and *OK*
- STEP 4: While the implementation is done, a new dialog window appears. Select on *Generate Bitstream* and *OK*

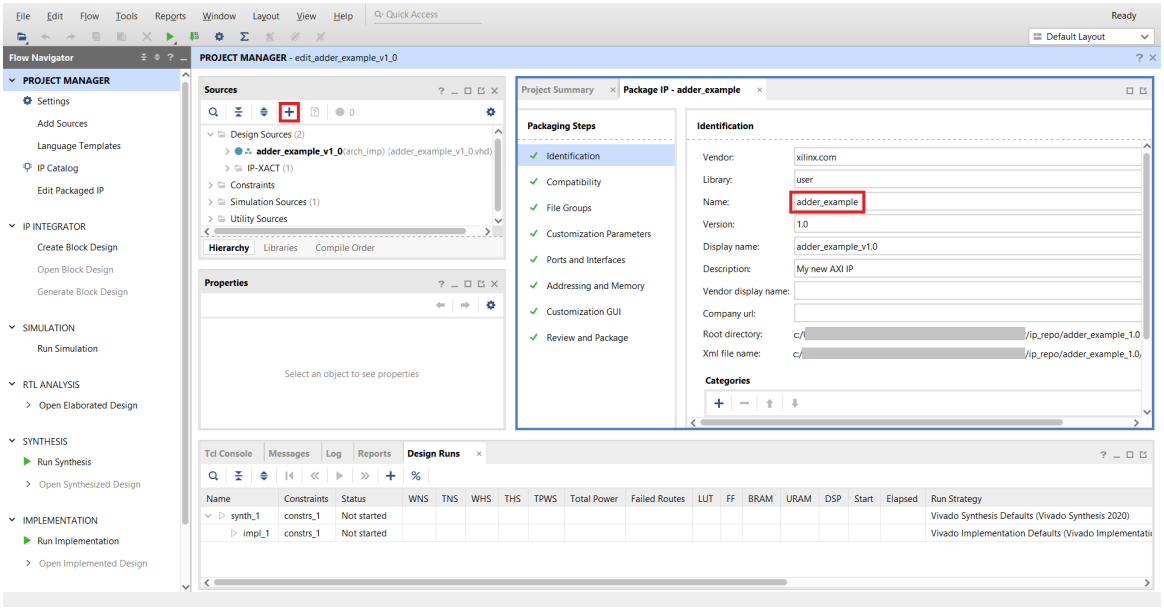

Figure 12: Click on the *plus* button inside the *Sources* box

9 Generate needed files for PYNQ-Z1 board

In order to import your design as an Overlay on the Pynq-Z1 board, three files are needed. The following sections explain how to generate those files.

Once generated, they can be copied on the Pynq-Z1 board and the design is ready to be imported and used.

9.1 .tcl

An easy way to generate this file is to export the Block Design. To do so, do the following:

- STEP 1: In *File* menu, select *Export* and then *Export Block Design...*
- STEP 2: Choose the destination and click *OK*

9.2 .hwh

After the steps done in section 8, the file named `<design_name>.hwh`, should be already correctly generated.

It can be found at the following location:
`<project_path>\<project_name>.gen\sources_1\bd\<design_name>\hw_handoff\`.

9.3 .bit

As for `.hwh` file, the `<design_name>_wrapper.bit` file has already been created in section 8.

The location is the following:
`<project_path>\<project_name>.runs\impl_1`

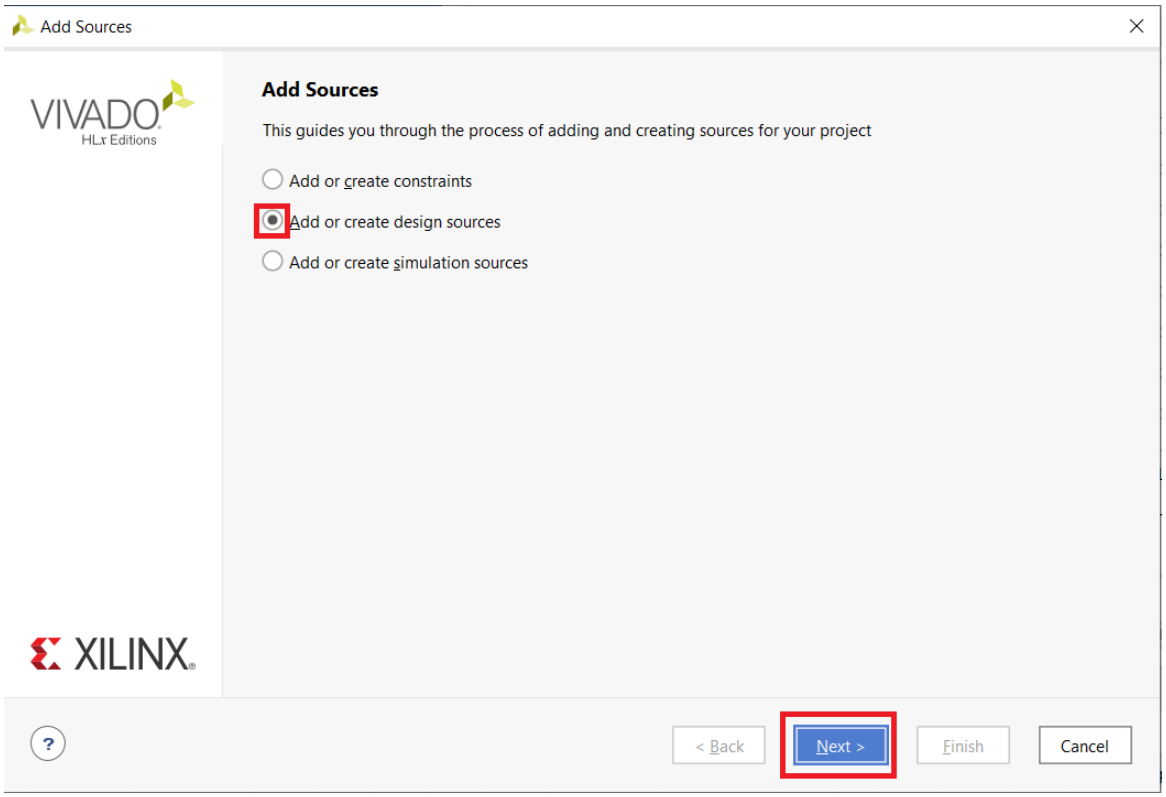

Figure 13: Select *Add or create design sources* and then click *Next*

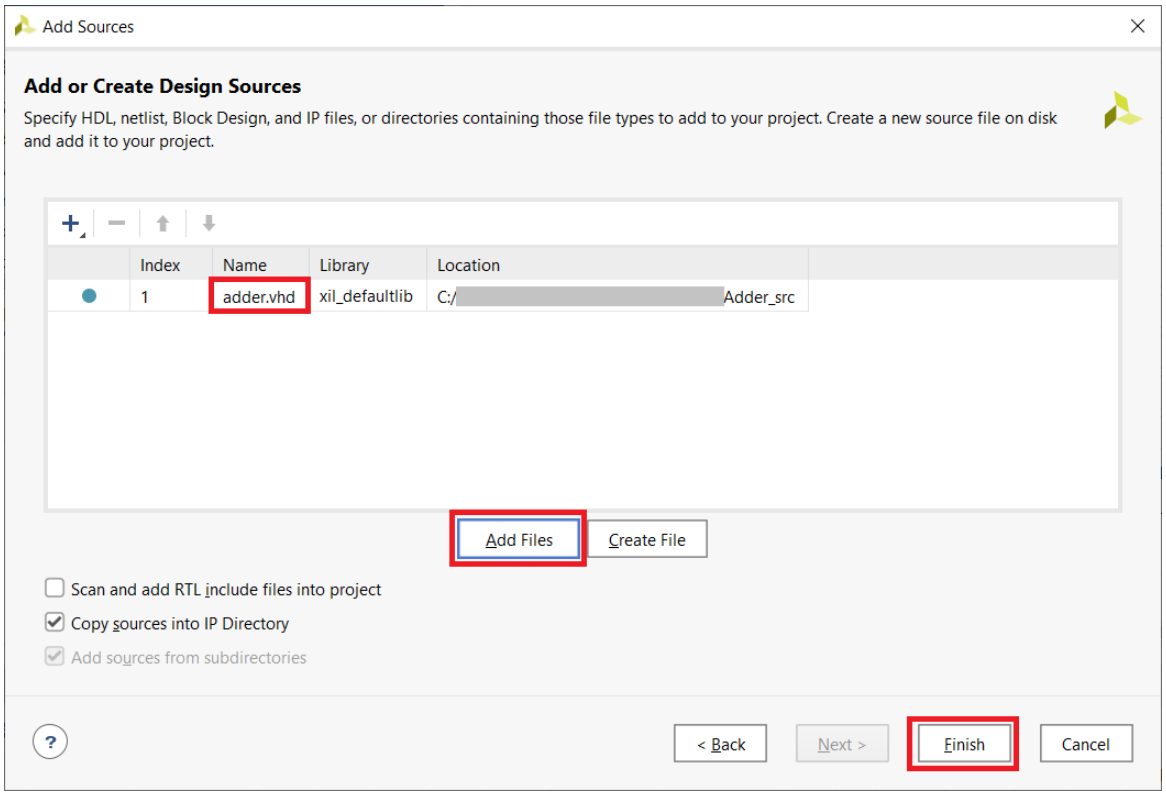

Figure 14: Click on *Add Files*, search for your VHDL sources (here *adder.vhd*) and click *Finish*

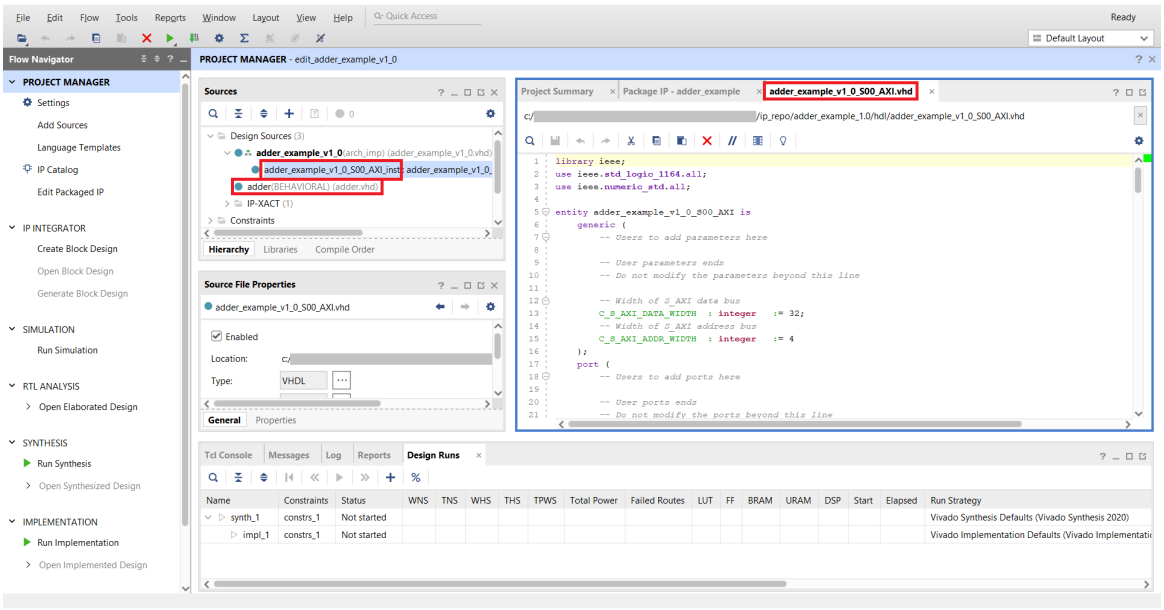

Figure 15: Open the file *adder_example_v1_0_S00_AXI.vhd* by clicking on its name in the *Sources* box: it appears on the editer on the right

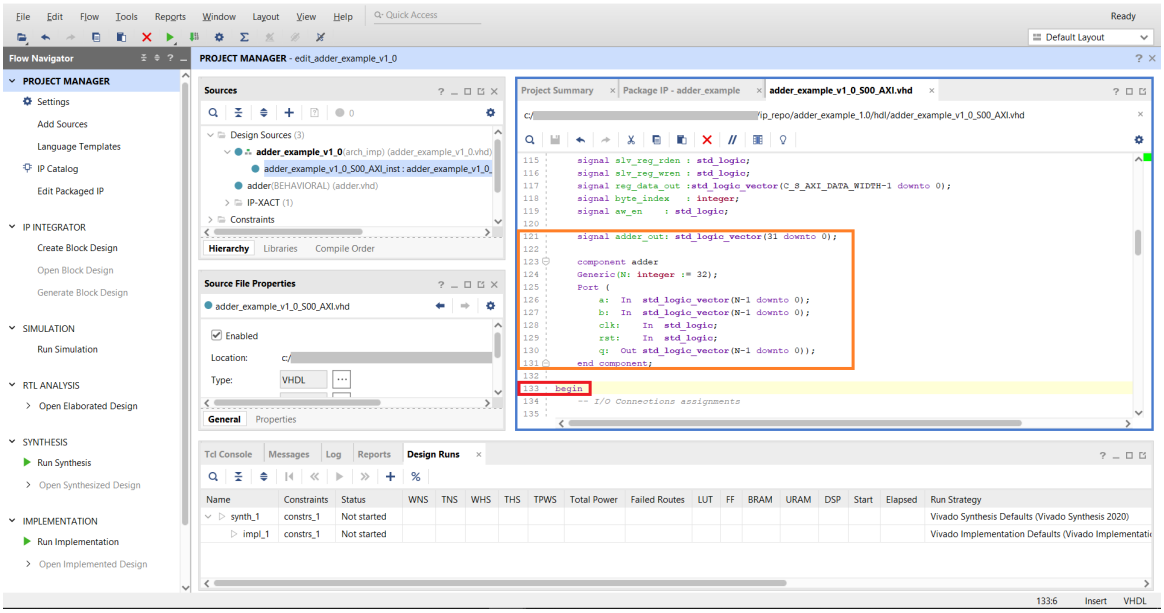

Figure 16: Declare the *adder* component by copying it before the keyword *begin*

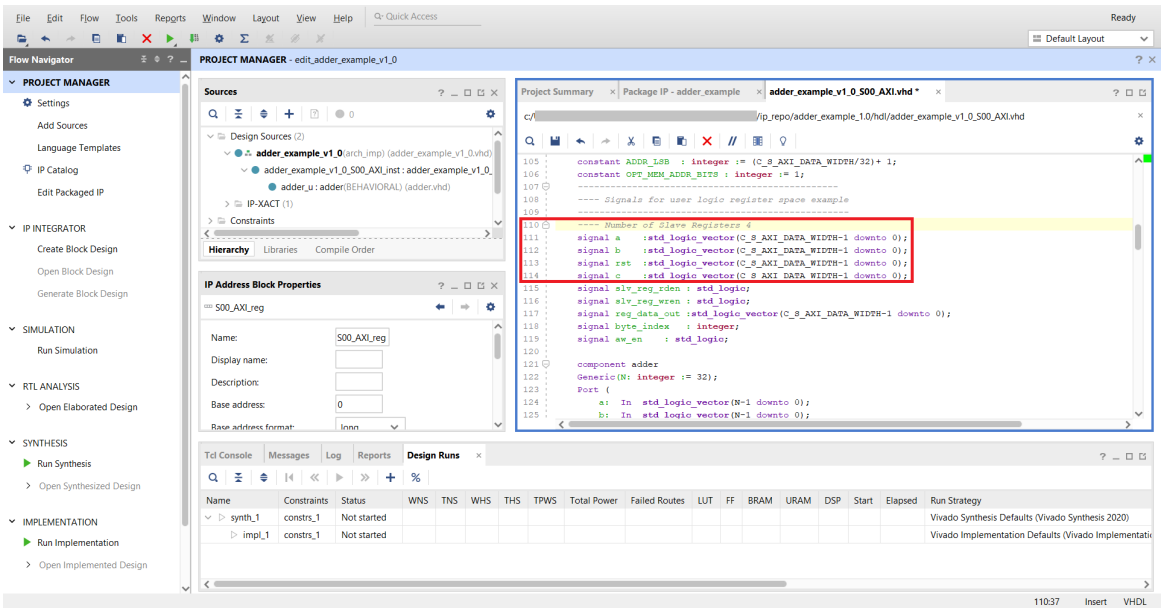

Figure 17: Rename the slave registers with more friendly names

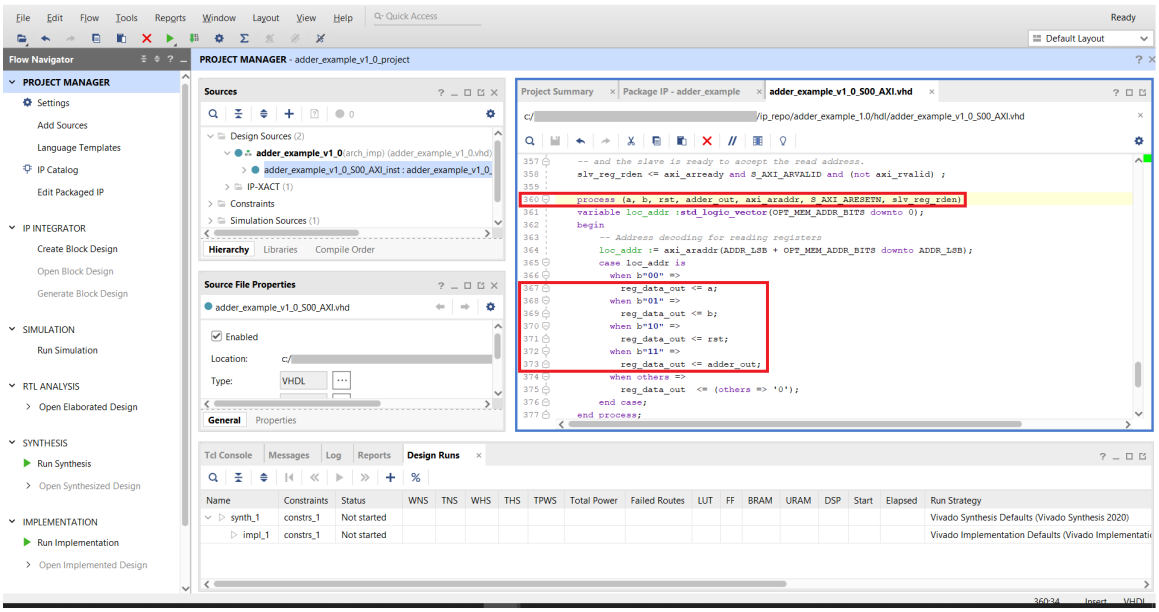

Figure 18: Change the outputs depending on the behavior of your circuit

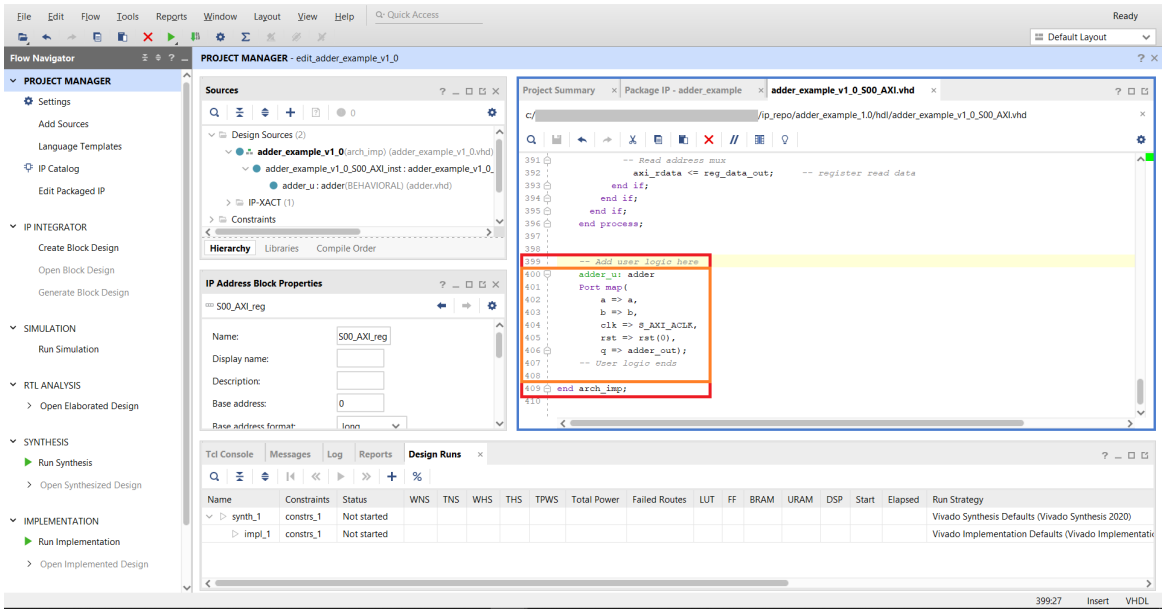

Figure 19: Add the associated logic of the component at the end of the file

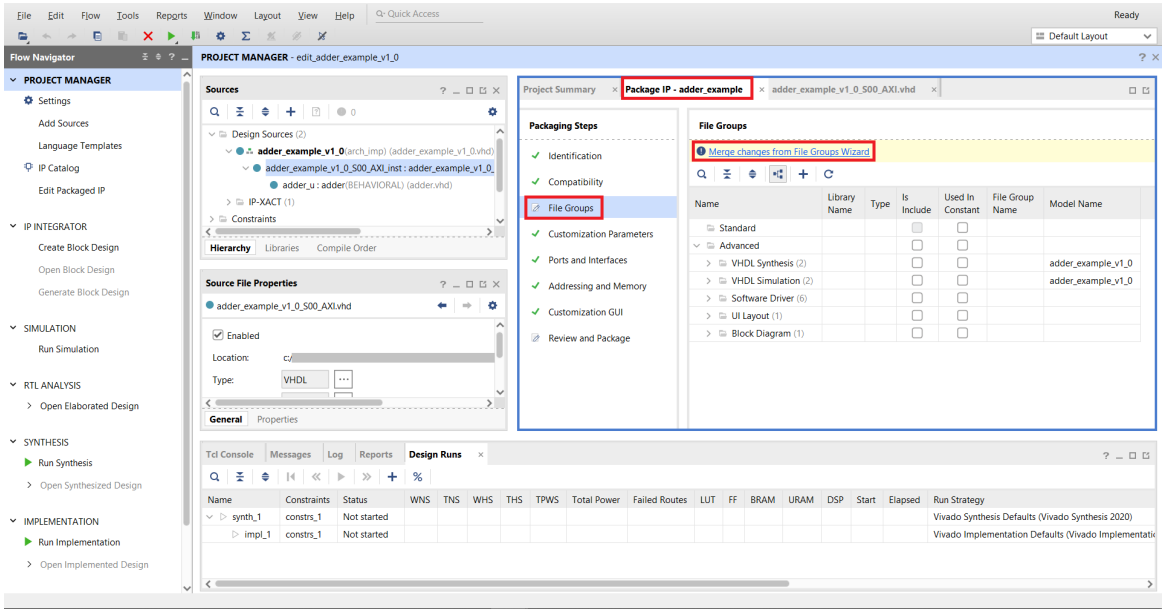

Figure 20: While in *Package IP - adder_example*, click on *File Groups*; then click on *Merge changes from File Groups Wizard*; to take into account the changes

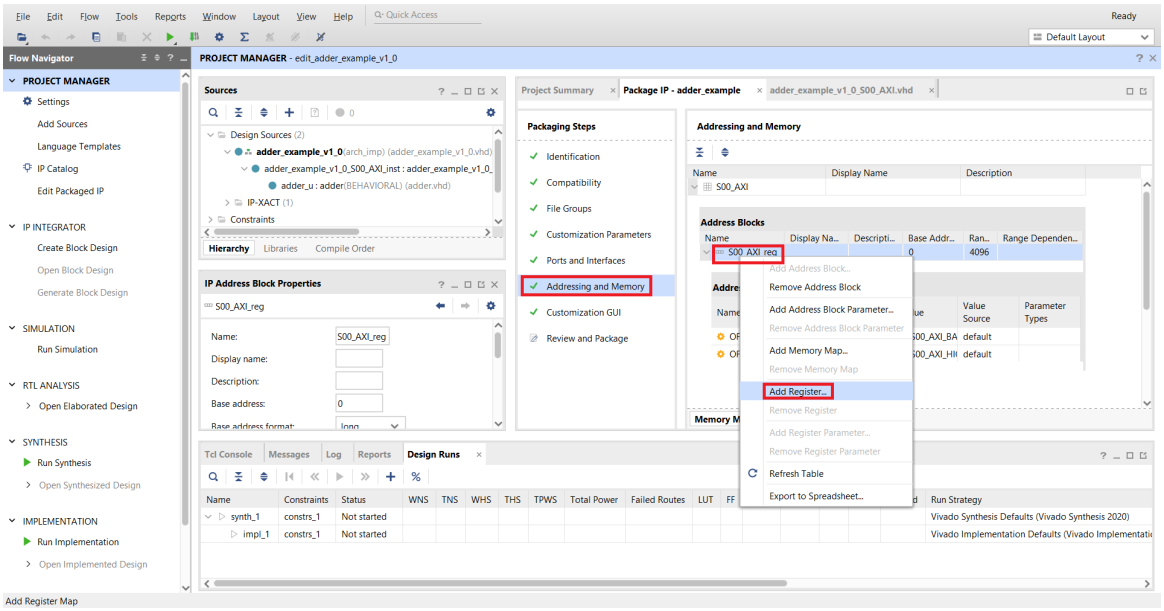

Figure 21: Go to *Addressing and Memory*, and add a register to *S00_AXI_reg* by right clicking on it and select *Add Register...*

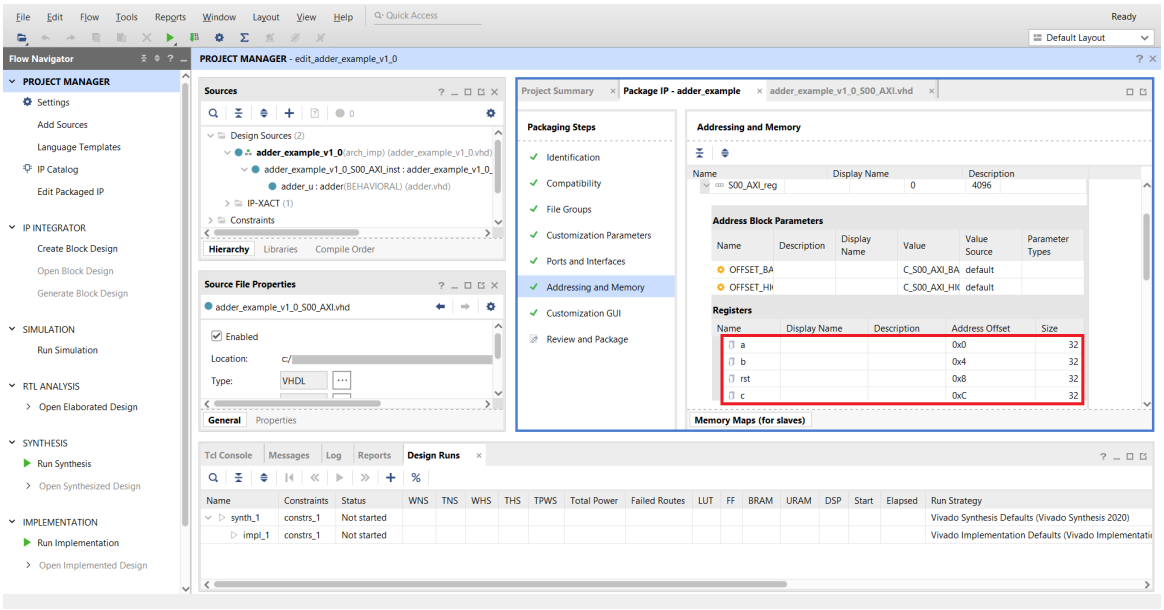

Figure 22: Final configuration of addressing for the AXI4 peripheral registers

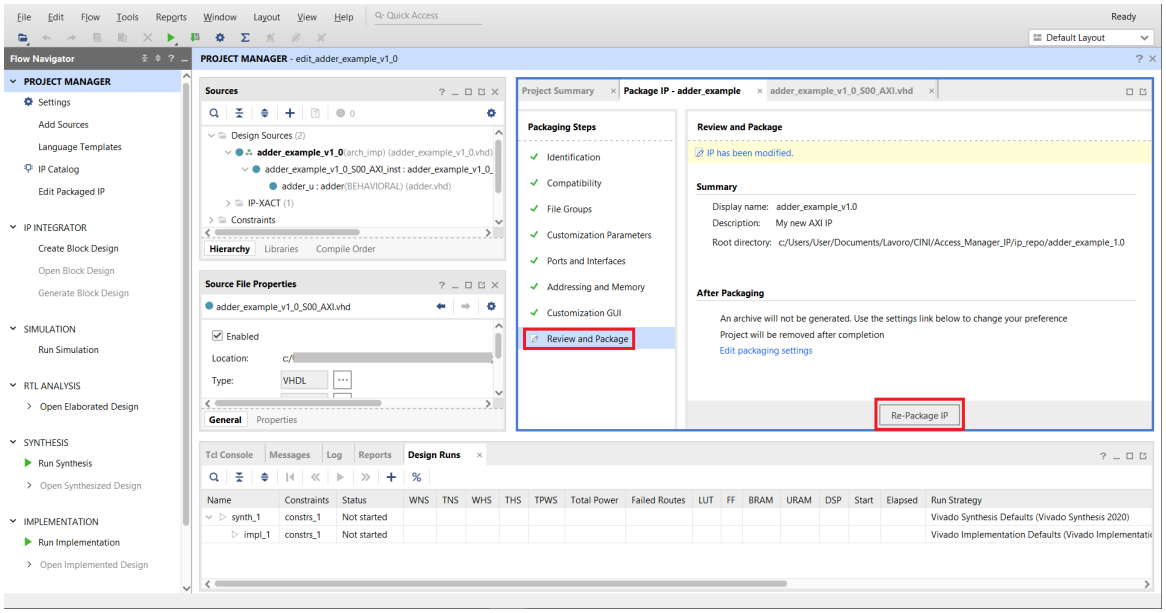

Figure 23: Got to *Review and Package* and click on the *Re-Package IP* button

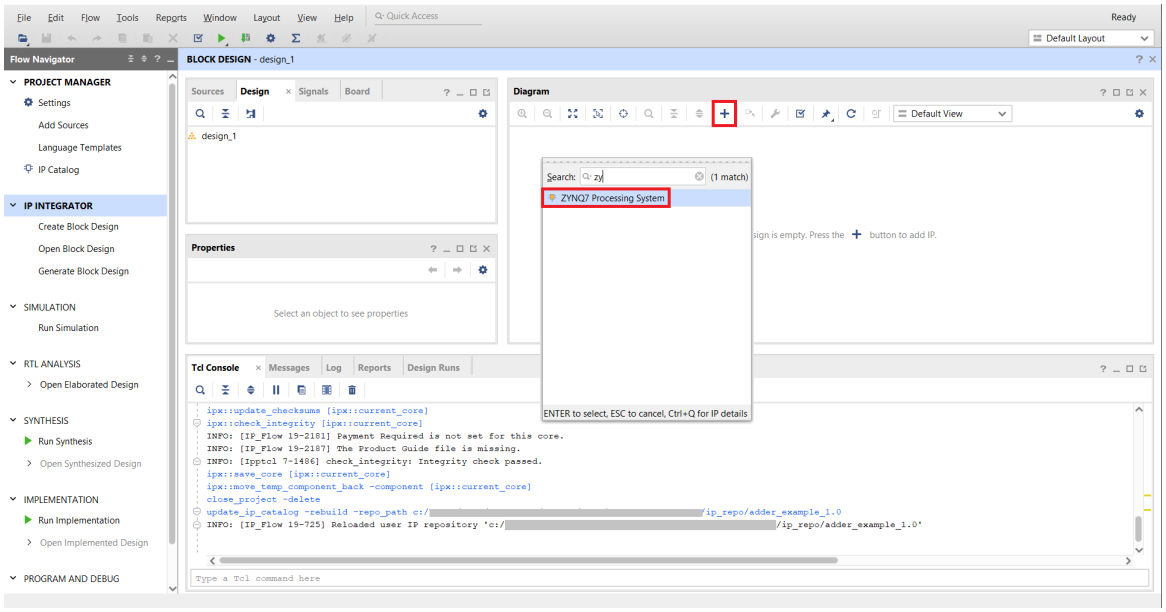

Figure 24: Click on the *plus* button in *Diagram* box and search for *Zynq7 Processing System*; add this IP by double clicking on it

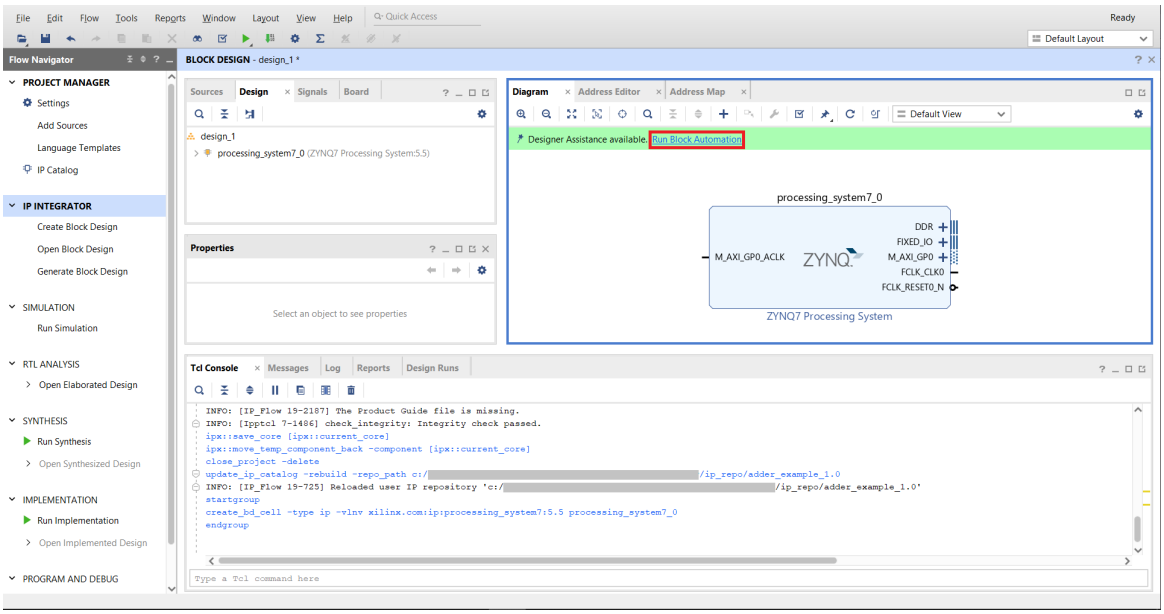

Figure 25: Click on *Run Block Automation* to add automatically useful configuration

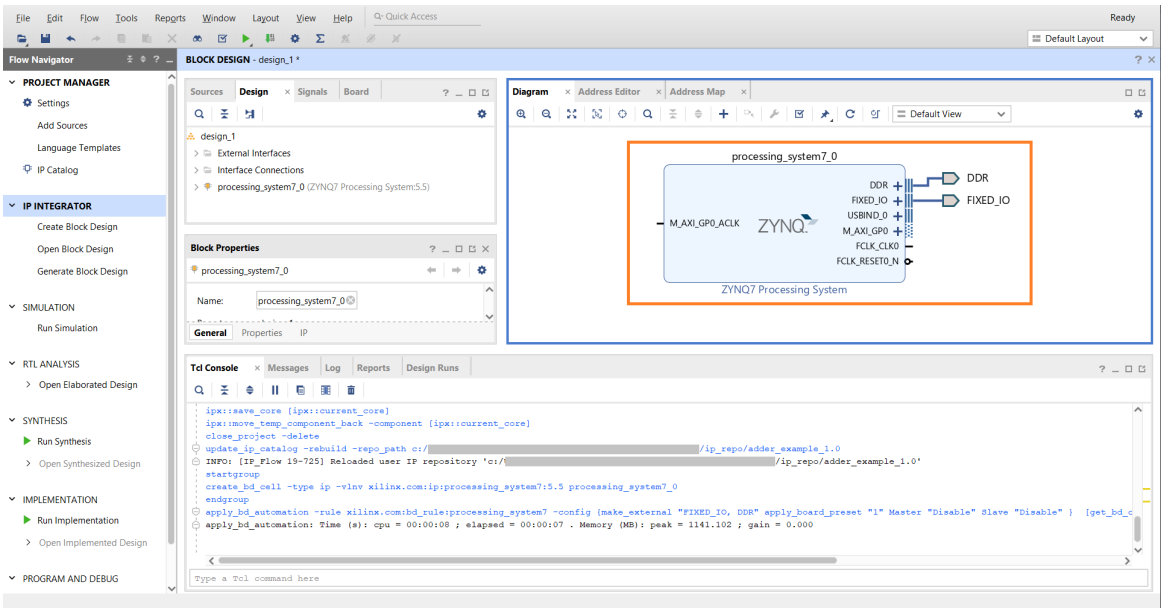

Figure 26: Fully configured Zynq processing system, after running block automation

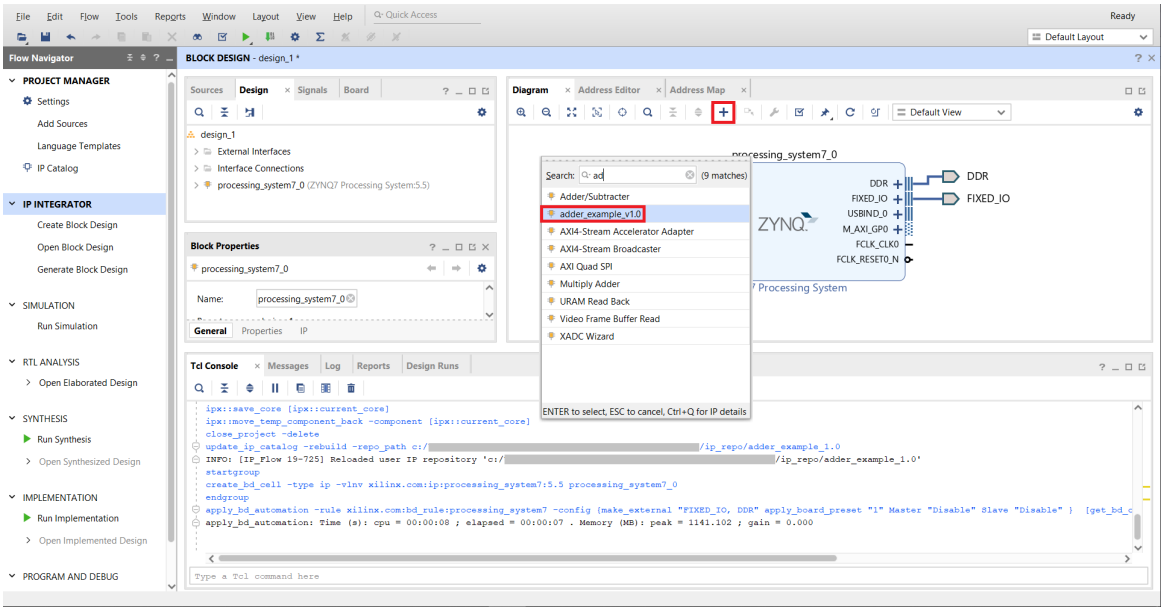

Figure 27: Click on the *plus* button in *Diagram* box and search for *adder_example_v1.0*; add this IP by double clicking on it

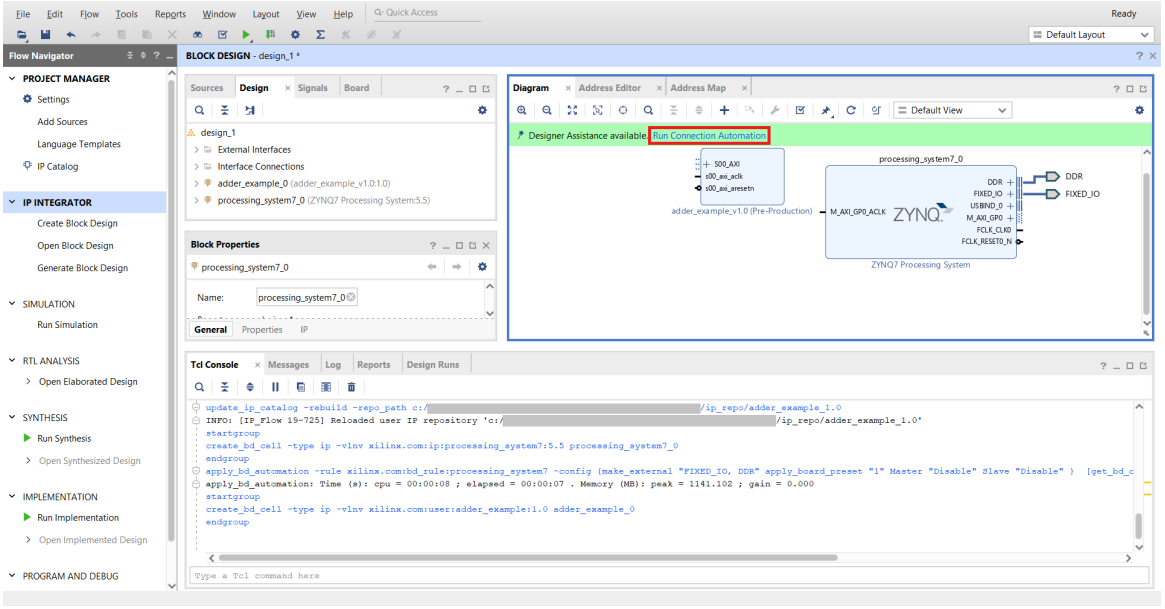

Figure 28: Click on *Run Connection Automation* to automatically connect the Adder IP and the Processing System

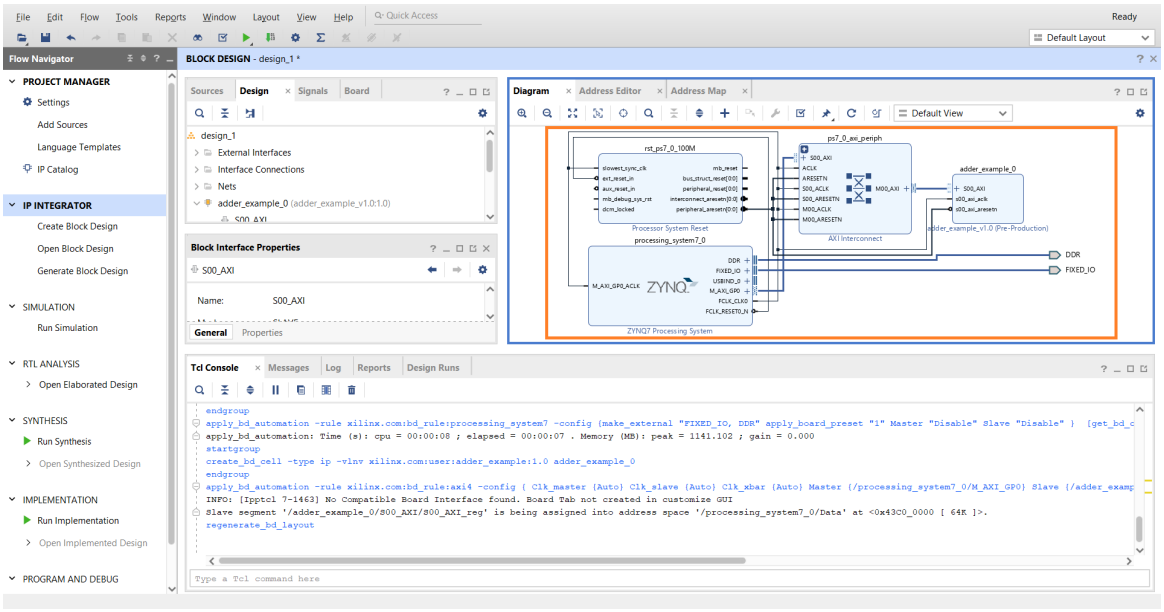

Figure 29: Fully connected block design

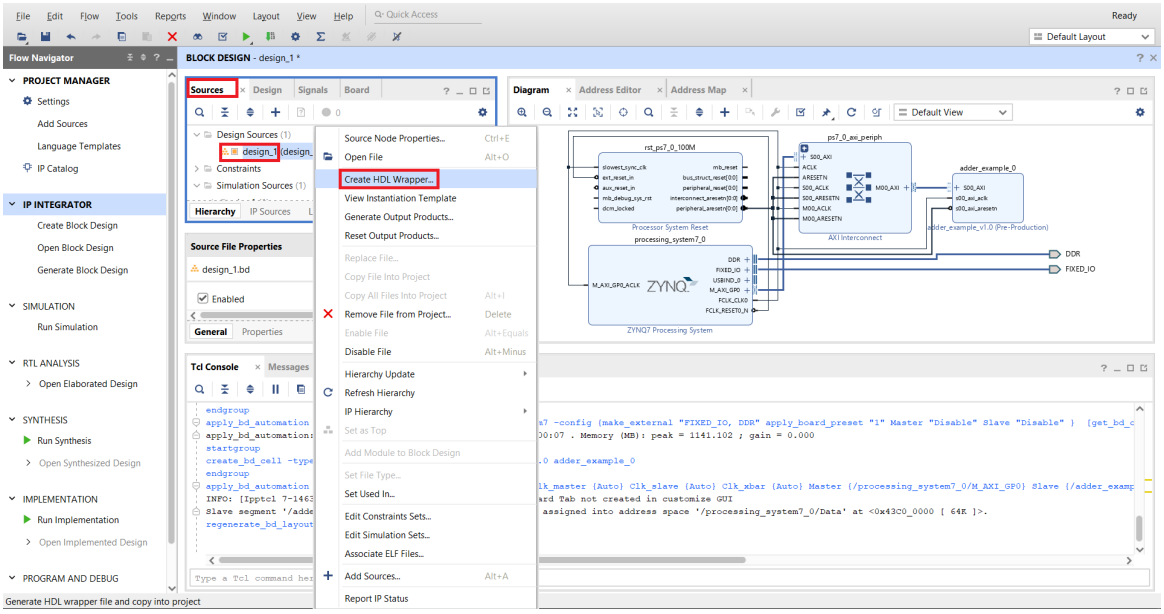

Figure 30: In the *Sources* menu, right click on your design (here called *design_1*) and select *Create HDL Wrapper*

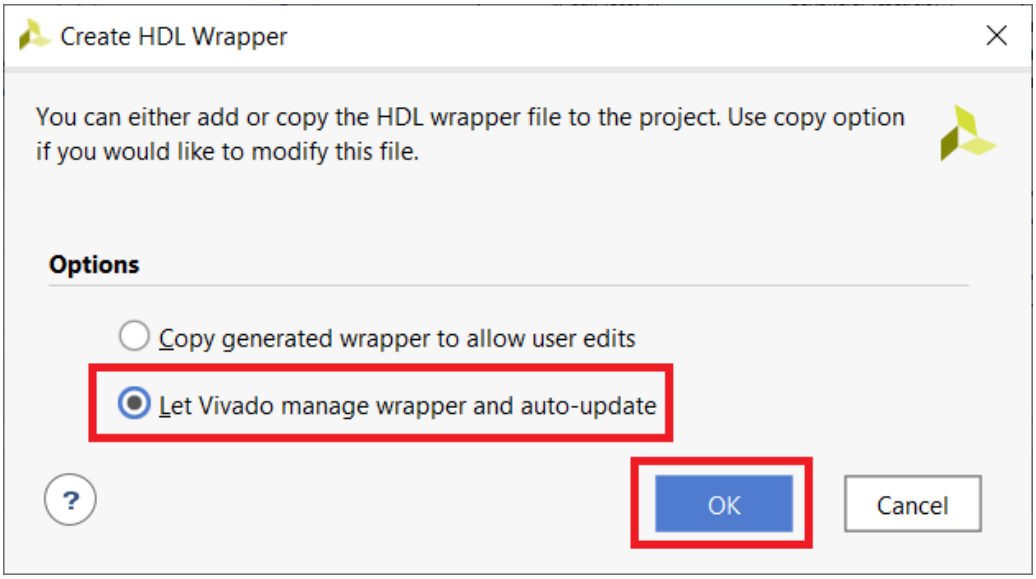

Figure 31: Select the option *Let Vivado manage wrapper and auto-update* and click *OK*

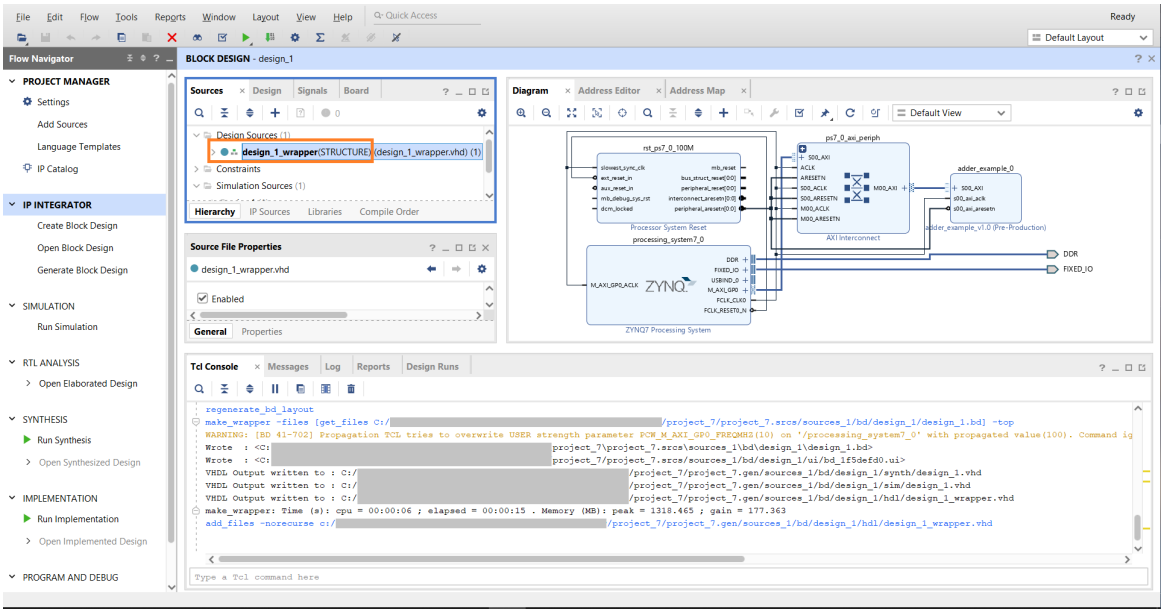

Figure 32: Added HDL wrapper on current block design

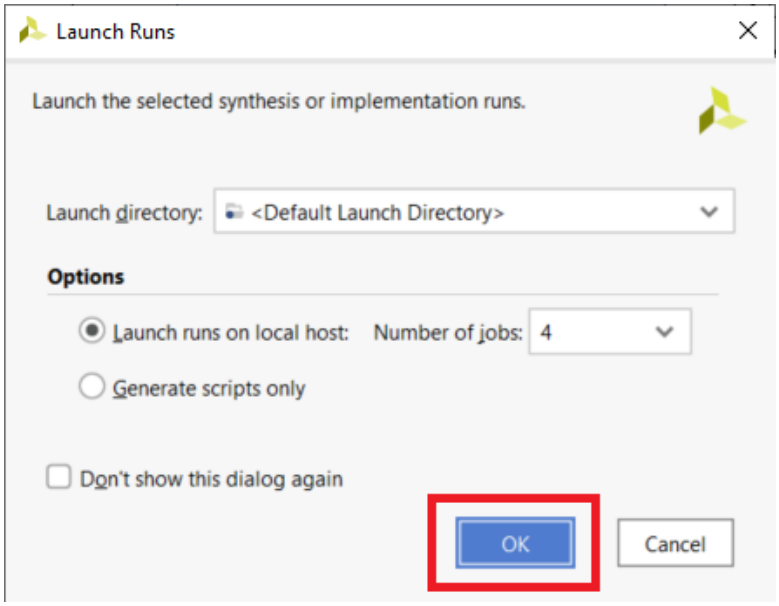

Figure 33: Click *OK* to run the synthesis

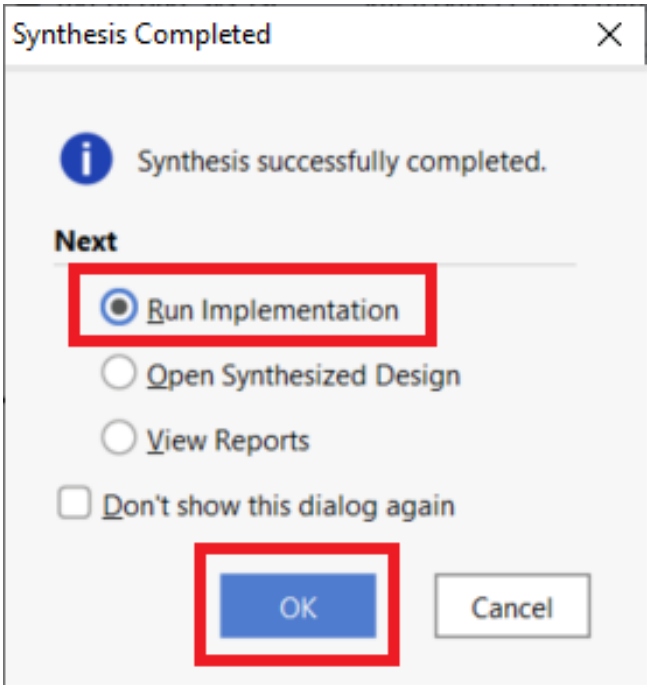

Figure 34: Click on *Run Implementation* and *OK*

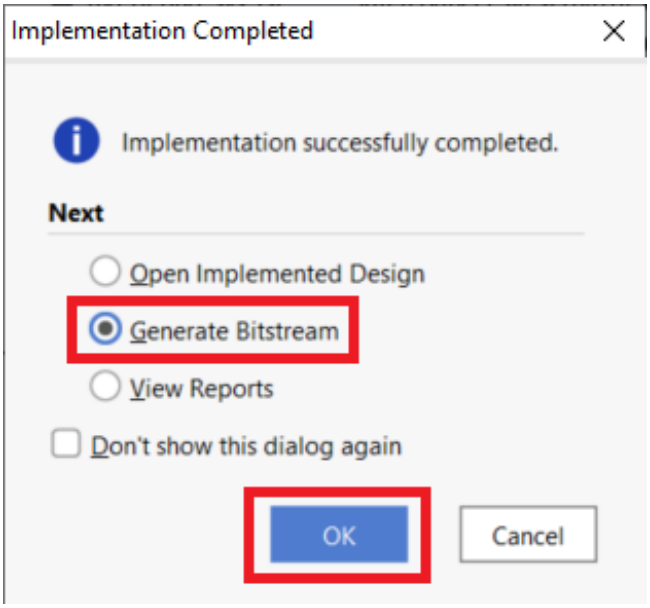

Figure 35: Select on *Generate Bitstream* and *OK*

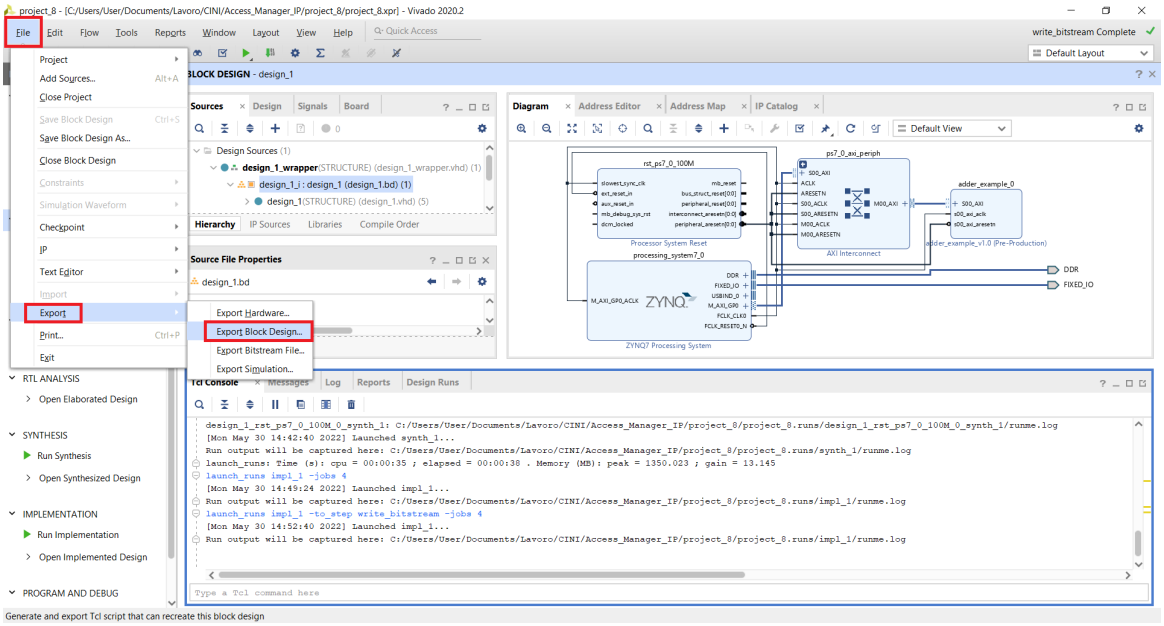

Figure 36: In *File* menu, select *Export* and then *Export Block Design...*

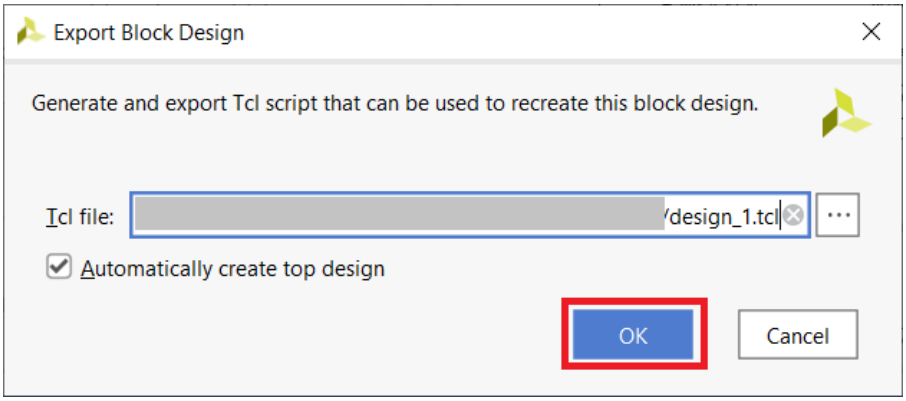

Figure 37: Choose the destination and click *OK*