

Lab 1: Zynq-SoC Design Flow Tutorial



Description

- This Lab serves as the back-bone of all the steps/processes that you will need through out the semester. Please make sure to do it fully and completely understand it.
- Through out the following Labs, specific parts of this Lab might be mentioned for your reference.
- In this Lab, you will learn about:
 - Importing user-created IP Repository
 - Creating your own IP Packages from VHDL designs that you write
 - Adding AXI hardware to your design to utilize software testing
 - Testing your hardware system using C testing code and VHDL testbench (as in the Pre-Lab)
 - Connecting a serial terminal to the PYNQ-Z1 board for communication.
 - Using Integrated Logic Analyzer (ILA) to probe different system signals during runtime.



Xilinx Zynq SoC



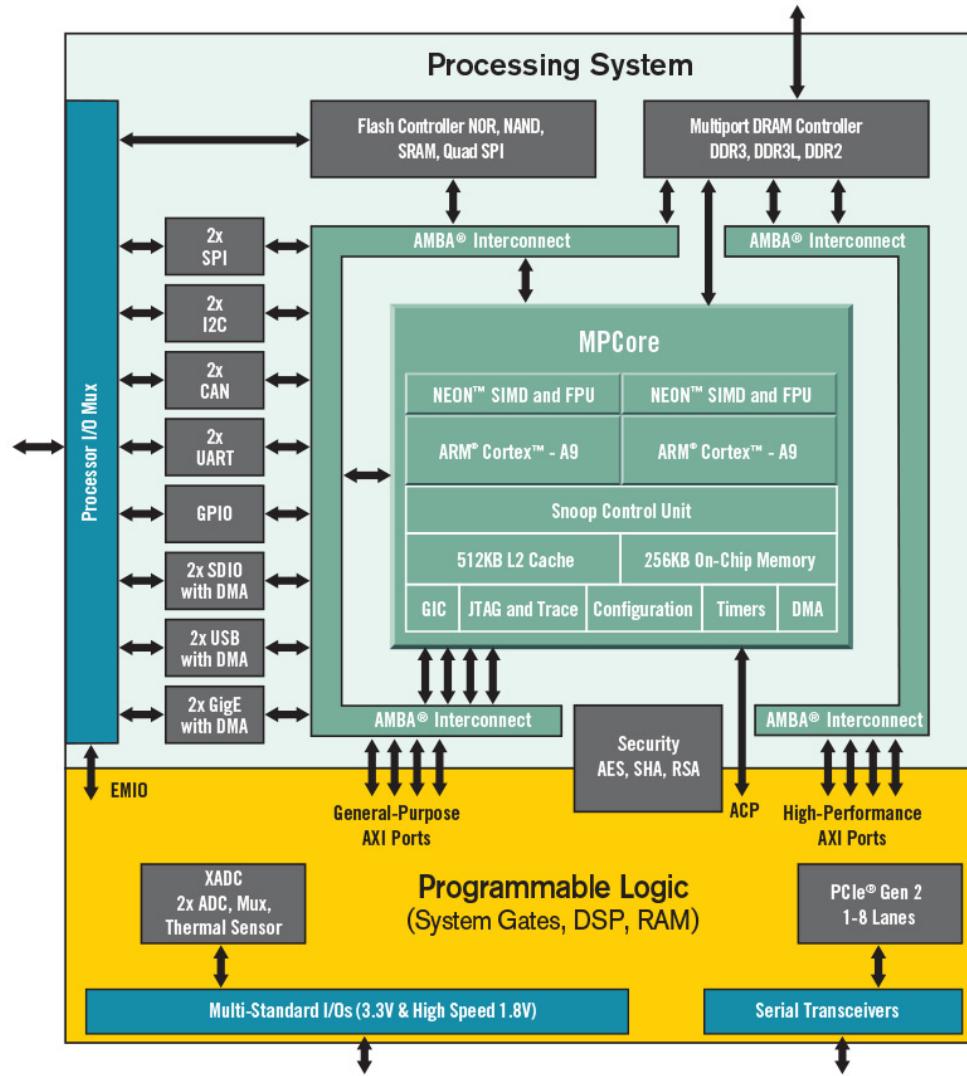
Documentation Resources

- Zynq SoC [TRM - UG585]
- Vivado
 - Synthesis [UG901]
 - Constraints [UG903]
 - Implementation [UG904]
 - Simulation [UG900]
 - ILA [PG172]
 - Libraries [UG953]
- Embedded System Tools [UG1043]
- Petalinux [UG1144]

Note: These are just for reference. No need to look any of them now!

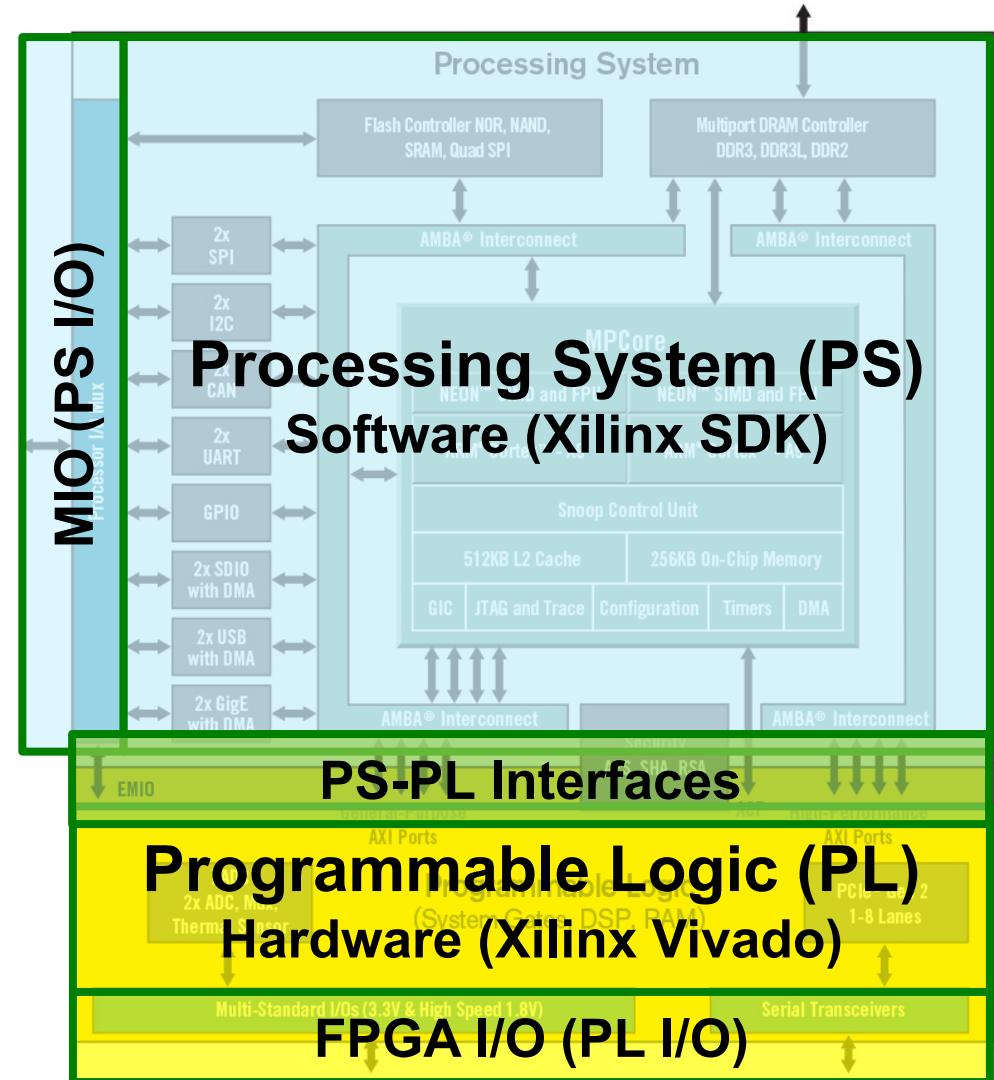


Zynq-7000 SoC Architecture Overview (1/2)



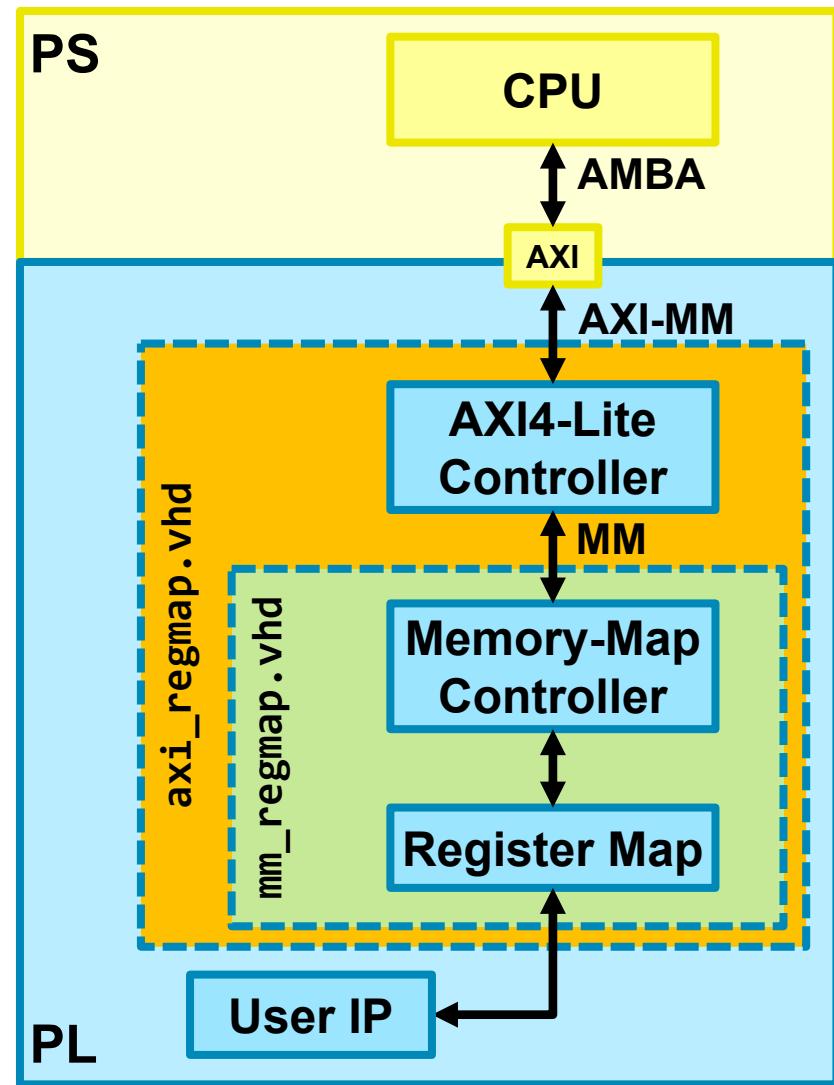
Zynq-7000 SoC Architecture Overview (2/2)

- Processing System (PS)
 - Contains general-purpose AXI ports with address space
 - Handles C programs to test Hardware (PL) Design
- Programmable Logic (PL)
 - FPGA Hardware
 - This is where the Configure Logic Blocks (CLBs) are



Memory-Mapped IP Cores

- PS
 - Contains general-purpose AXI ports with address space
 - CPU issues read/writes through AXI ports to peripherals in FPGA
- PL
 - Attach AXI-compatible IP core to AXI port to interface with CPU
 - AXI and memory-map controllers convert AXI transactions into register reads/writes
 - regmap IP enables users to connect custom IP with register map for general-purpose reads/writes

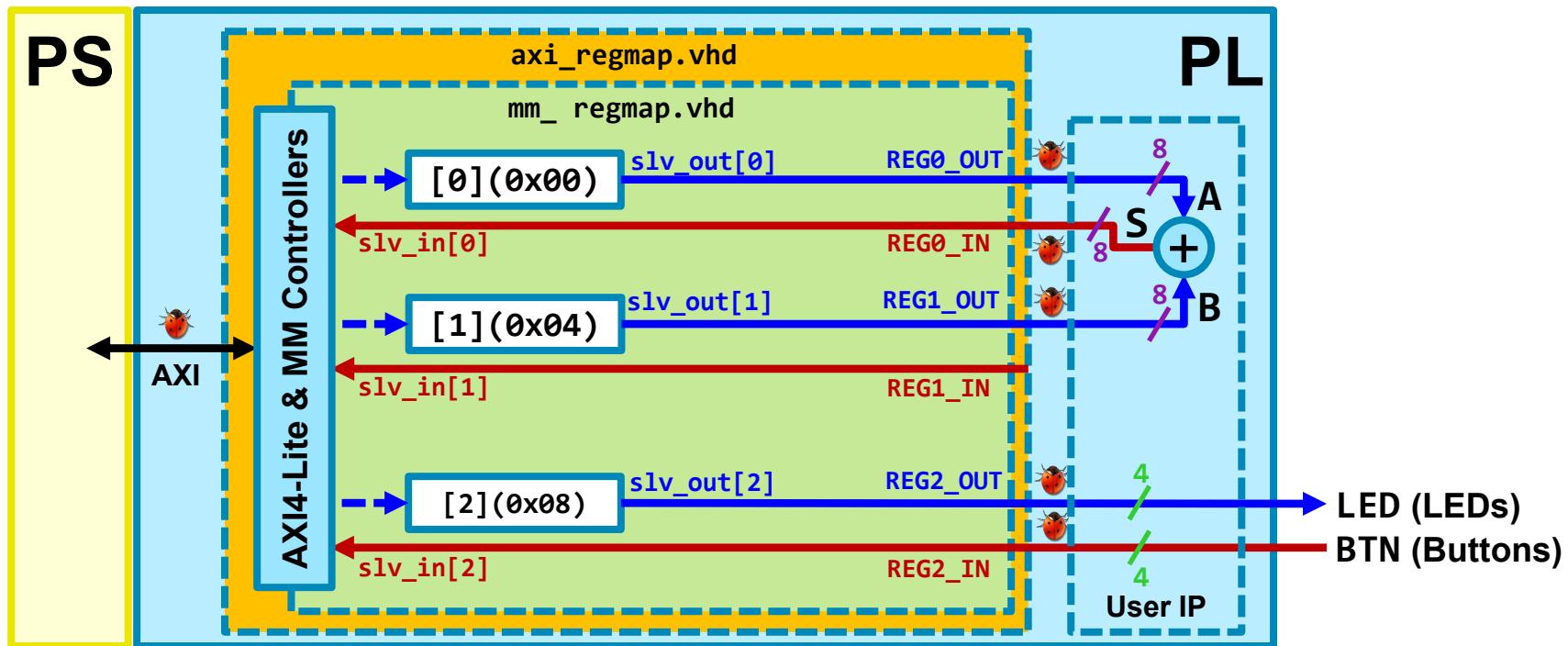


Example Design



Integrated Logic Analyzer (ILA)
Signals that will be probed

Register Map			
Index	Offset	Read Access	Write Access
[0]	0x00	S (adder output)	A (adder input)
[1]	0x04	Zero	B (adder input)
[2]	0x08	BTN	LED



Xilinx Vivado Tutorial



Part 1: Creating a new Vivado Project



Part 1: Create a new Vivado Project

- Create a new project as per the Pre-Lab
 - The project should be named “Lab_1”.
Note: Neither the project name, nor the project path should have spaces!
 - Project Type: RTL
 - Target Language: VHDL
 - Constraints File: “PYNQ-Z1_C.xdc” (from Pre-Lab files)
 - Boards: PYNQ-Z1

Note: These are just for reference. No need to look any of them now!

Part 2: Adding an IP Repository

An IP repository contains all packaged IP that can be used for system integration. We will add an IP repository that already contains the **axi_regmap** IP.



Click **Settings**.

The screenshot shows the Vivado Project Manager interface. On the left, the Project Manager sidebar is open, with the 'Settings' option under 'PROJECT MANAGER' highlighted with a red box. The main area displays the 'example' project summary, board part details, synthesis, and implementation sections. At the bottom, the 'Design Runs' tab is active, showing a table of build runs for 'synth_1' and 'impl_1'.

Project Summary

Settings

Board Part

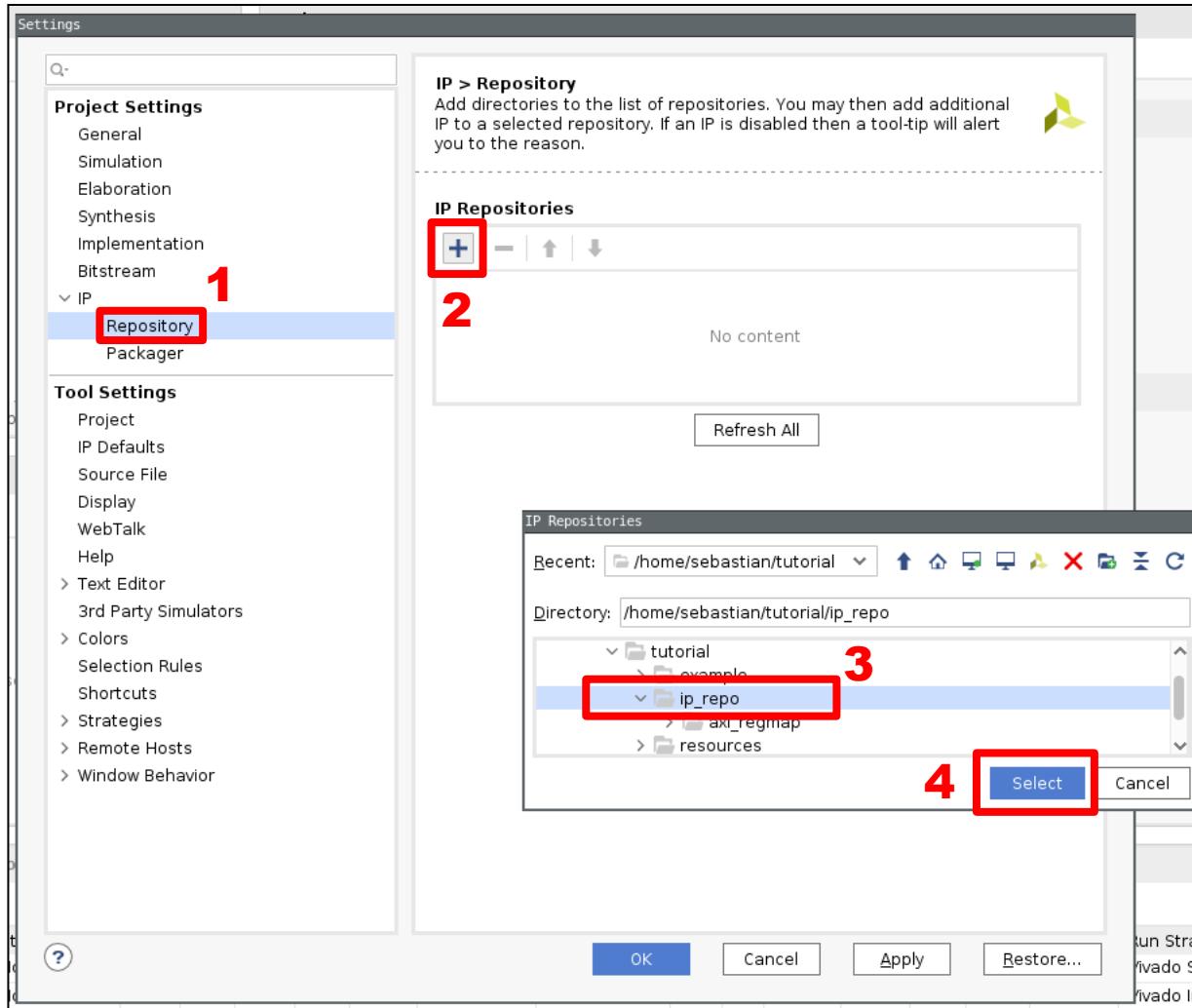
Synthesis

Implementation

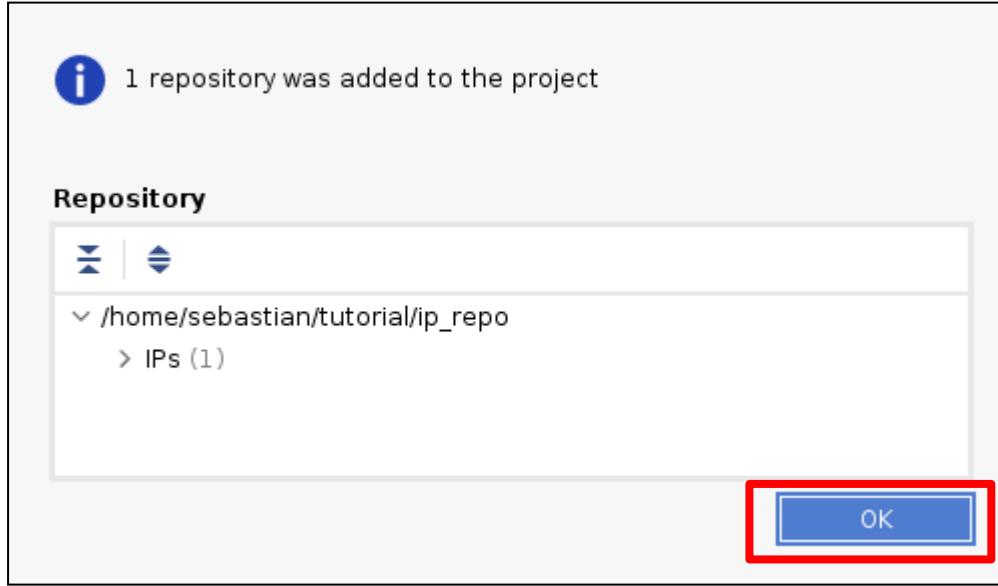
Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy	Report Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2018)	Vivado Synthesis Default Reports (Vivado Synthesis 2018)
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2018)	Vivado Implementation Default Reports (Vivado Implementation 2018)

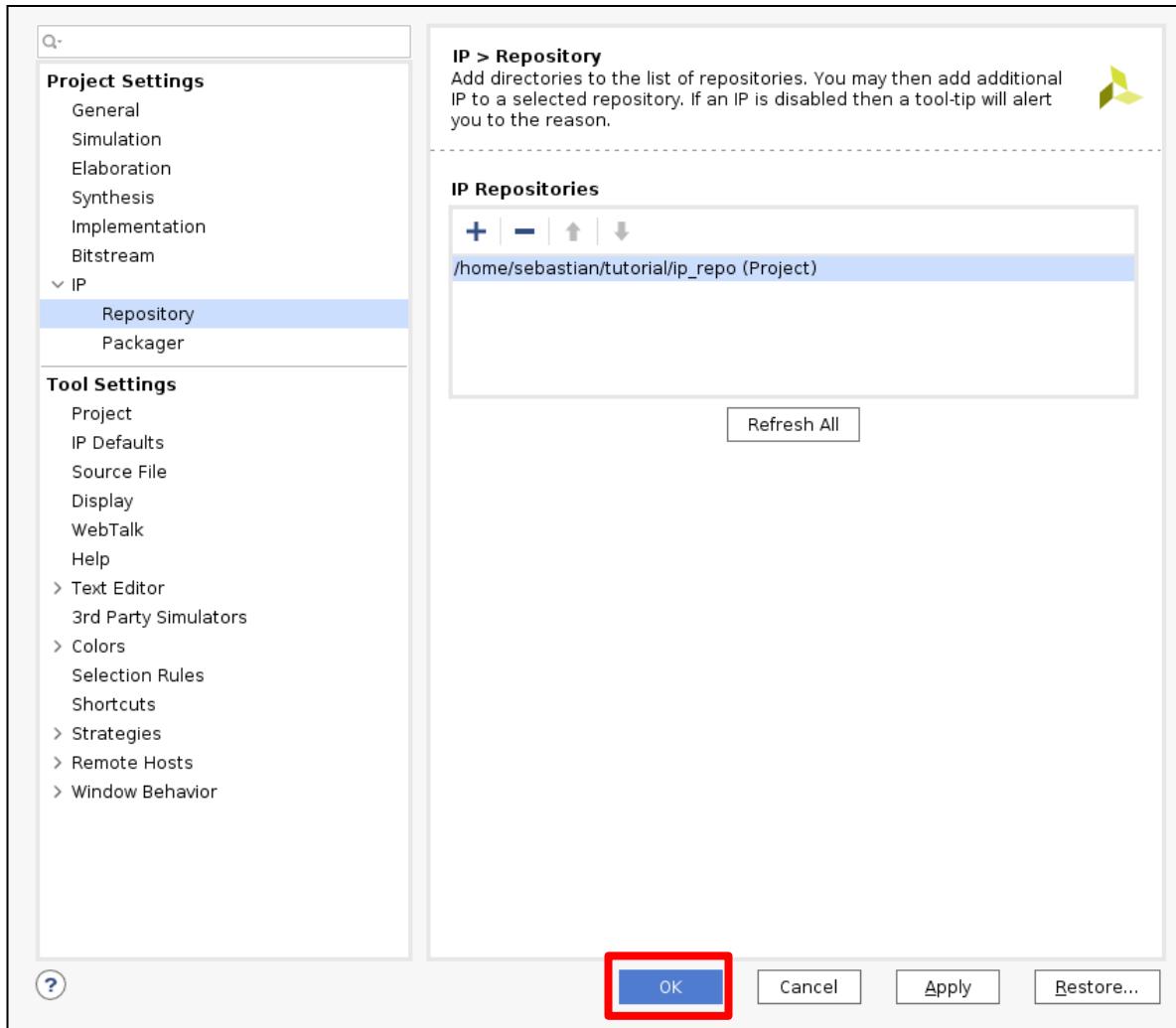
Click **Repository**. Click **+ (Add)**. Navigate to **Lab_1_repo/ip_repo**. Click **Select**.



Click **OK**.



Click **OK**.

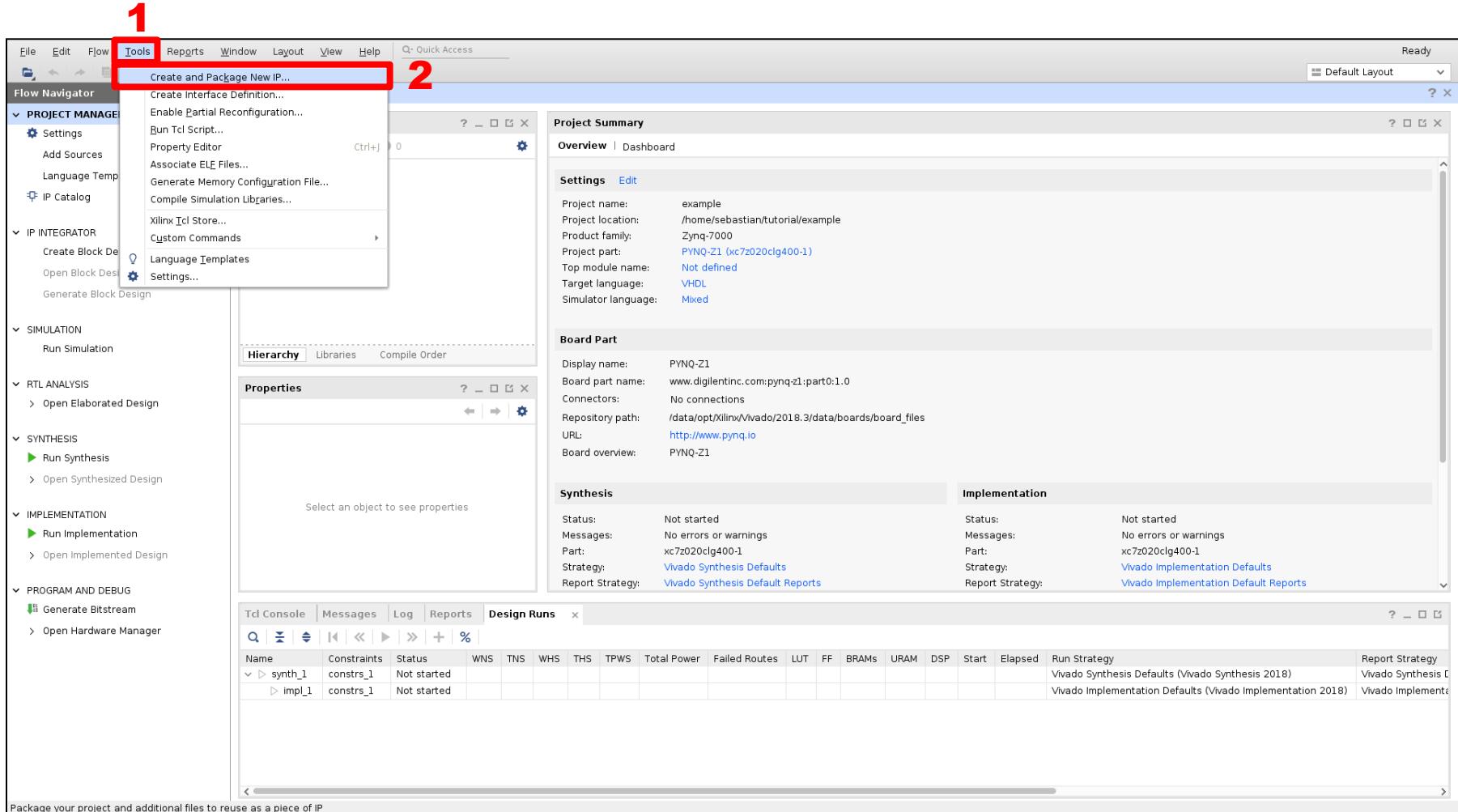


Part 3: Creating a new IP

Next, we will create an **adder** IP and package it using the Vivado IP Packager.



Click **Tools**. Click **Create and Package New IP....**



Click Next.

VIVADO
HLx Editions

Create and Package New IP

This wizard can be used to accomplish following tasks:

Package a new IP for the Vivado IP Catalog
This wizard will guide you through the process of creating a new Vivado IP using source files and information from your current project, block design or specified directory.

Create a new AXI4 Peripheral
This wizard will guide you through the process of creating a new AXI4 peripheral which includes HDL, driver, software test application, IP Integrator VIP simulation and debug demonstration design.

XILINX

Click Next to continue

< Back **Next >** Finish Cancel



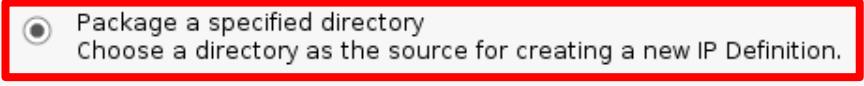
Select **Package a specified directory**. Click **Next**.

Create Peripheral, Package IP or Package a Block Design
Please select one of the following tasks.



Packaging Options

- Package your current project
Use the project as the source for creating a new IP Definition.
- Package a block design from the current project
Choose a block design as the source for creating a new IP Definition.
- Package a specified directory**
Choose a directory as the source for creating a new IP Definition.

1 

Create AXI4 Peripheral

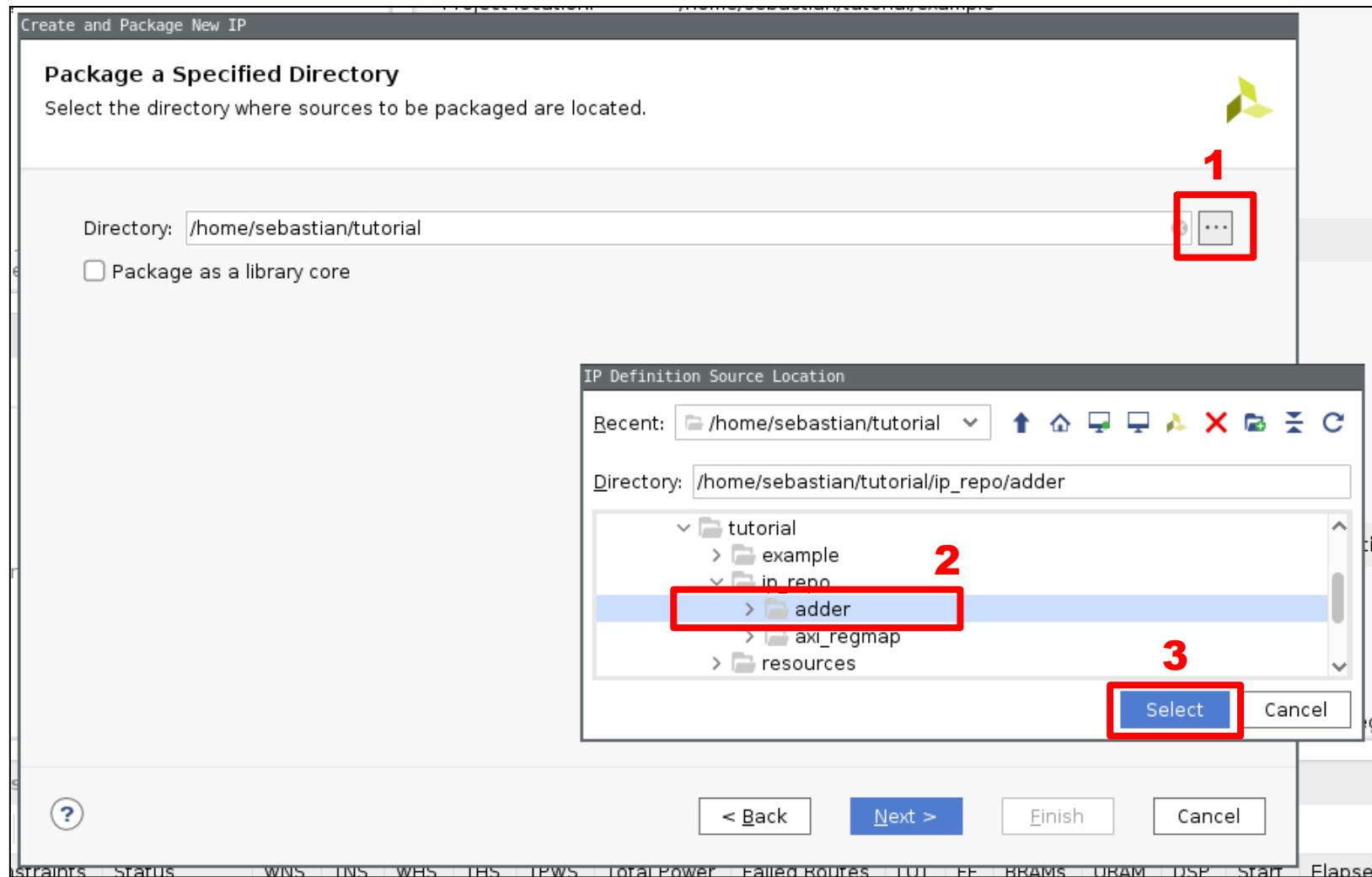
- Create a new AXI4 peripheral
Create an AXI4 IP, driver, software test application, IP Integrator AXI4 VIP simulation and debug demonstration design.

2 

 [? < Back](#) [Next >](#) [Finish](#) [Cancel](#)

Click Navigate to **Lab_1_repo/resources/adder**. Click **Select**.

Note: The above directory is different from the one in the picture.



Click Next.

Edit in IP Packager Project Name

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

Project name: edit_ip_project

Project location: /home/sebastian/tutorial/example/example.tmp

Edit IP project will be created at: /home/sebastian/tutorial/example/example.tmp/edit_ip_project

< Back **Next >** Finish Cancel



Click ***Finish***.

VIVADO
HLx Editions

New IP Creation

The following pieces of information will be gathered:

- Identification information based on top module name
- Family compatibility based on part in the project
- File(s) from Synthesis and Simulation file sets
- Ports from the file containing the top module
- Parameters from the file containing the top module
- Bus Interfaces based on port names
- Address Spaces and Memory Maps based on inferred bus interfaces

Following file will be created on disk along with corresponding customization files:
`/home/sebastian/tutorial/ip_repo/adder/component.xml`

XILINX

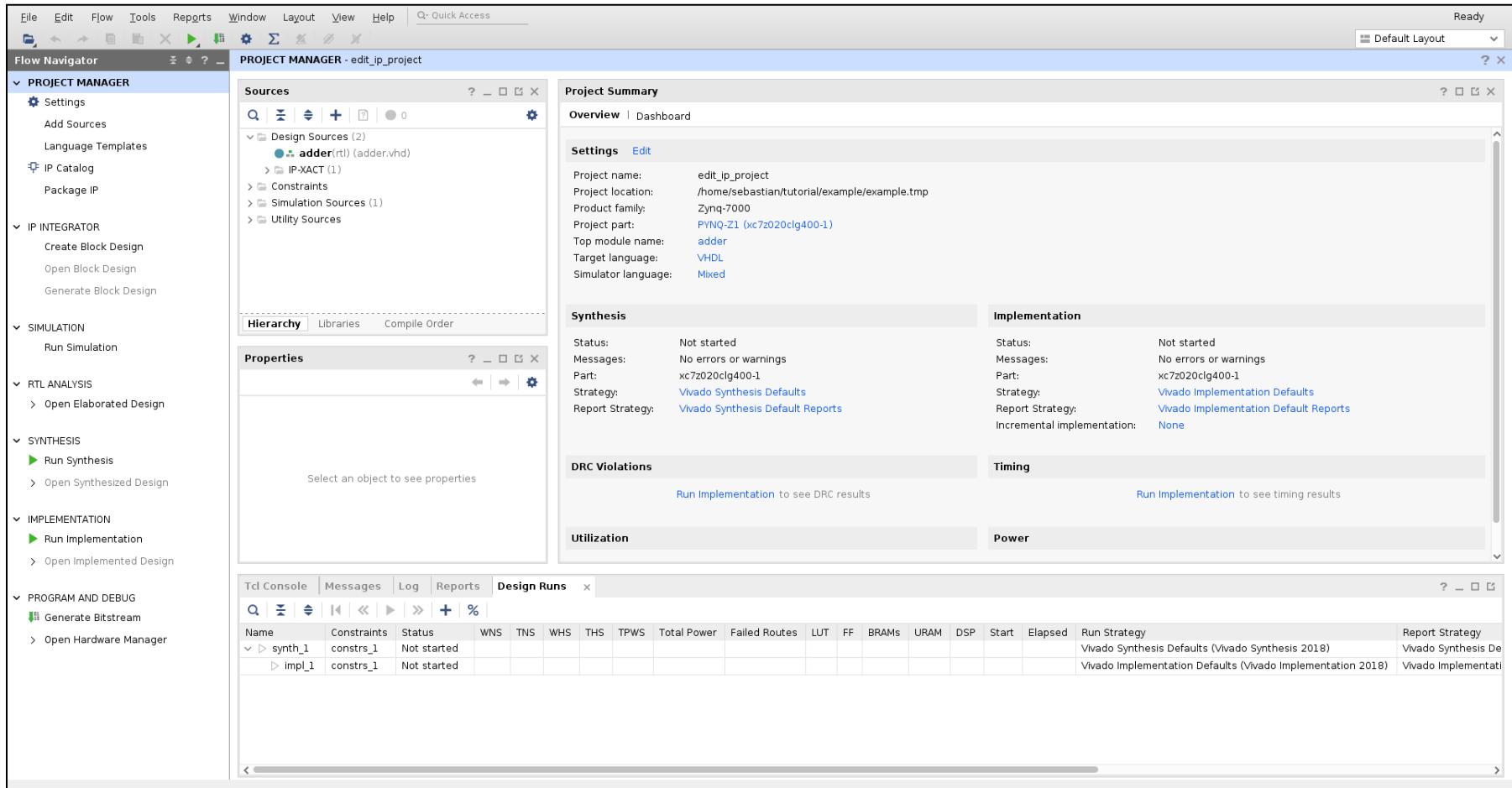
Click Finish to continue

[?](#) [Back](#) [Next >](#) [Finish](#) [Cancel](#)



A new Vivado instance is created specifically for the **adder** IP. Use this instance to develop and package the **adder** IP. Continue.

Note: for future projects, you might want to leave this instance opened in case you need to edit and re-package your design.



adder.vhd. This RTL adds two inputs (**A** and **B**) represented as vectors of width **WIDTH** and outputs the sum (**S**) represented as a vector of width **WIDTH**. Continue.

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

entity adder is
    generic (
        WIDTH : positive := 8
    );
    port (
        A      : in  std_logic_vector(WIDTH-1 downto 0);
        B      : in  std_logic_vector(WIDTH-1 downto 0);
        S      : out std_logic_vector(WIDTH-1 downto 0)
    );
end entity;

architecture rtl of adder is
begin
    S <= std_logic_vector(unsigned(A) + unsigned(B));
end architecture;
```

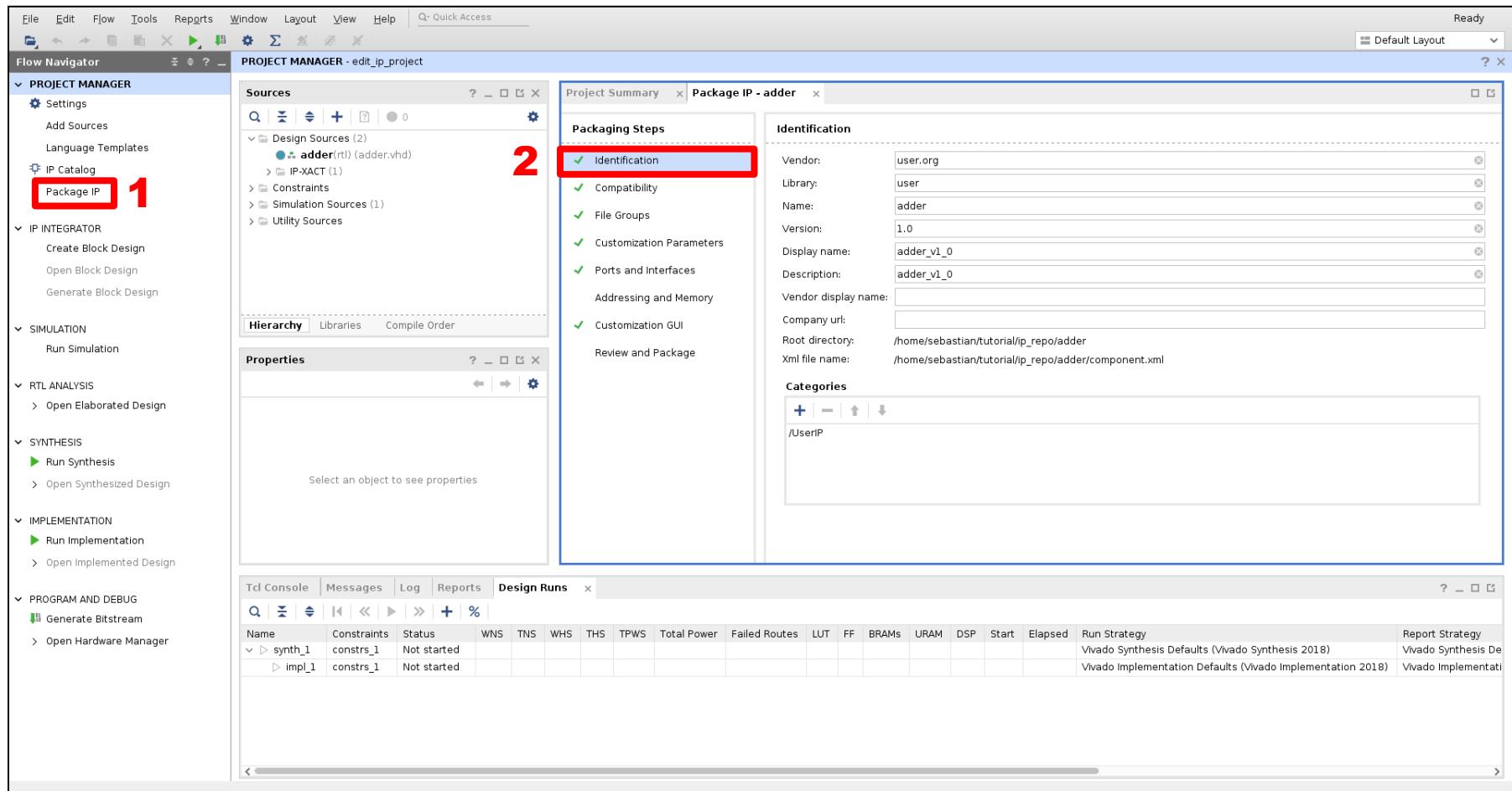
Generics in VHDL become
Customization Parameters in Vivado

Ports in VHDL become
Ports and Interfaces in Vivado



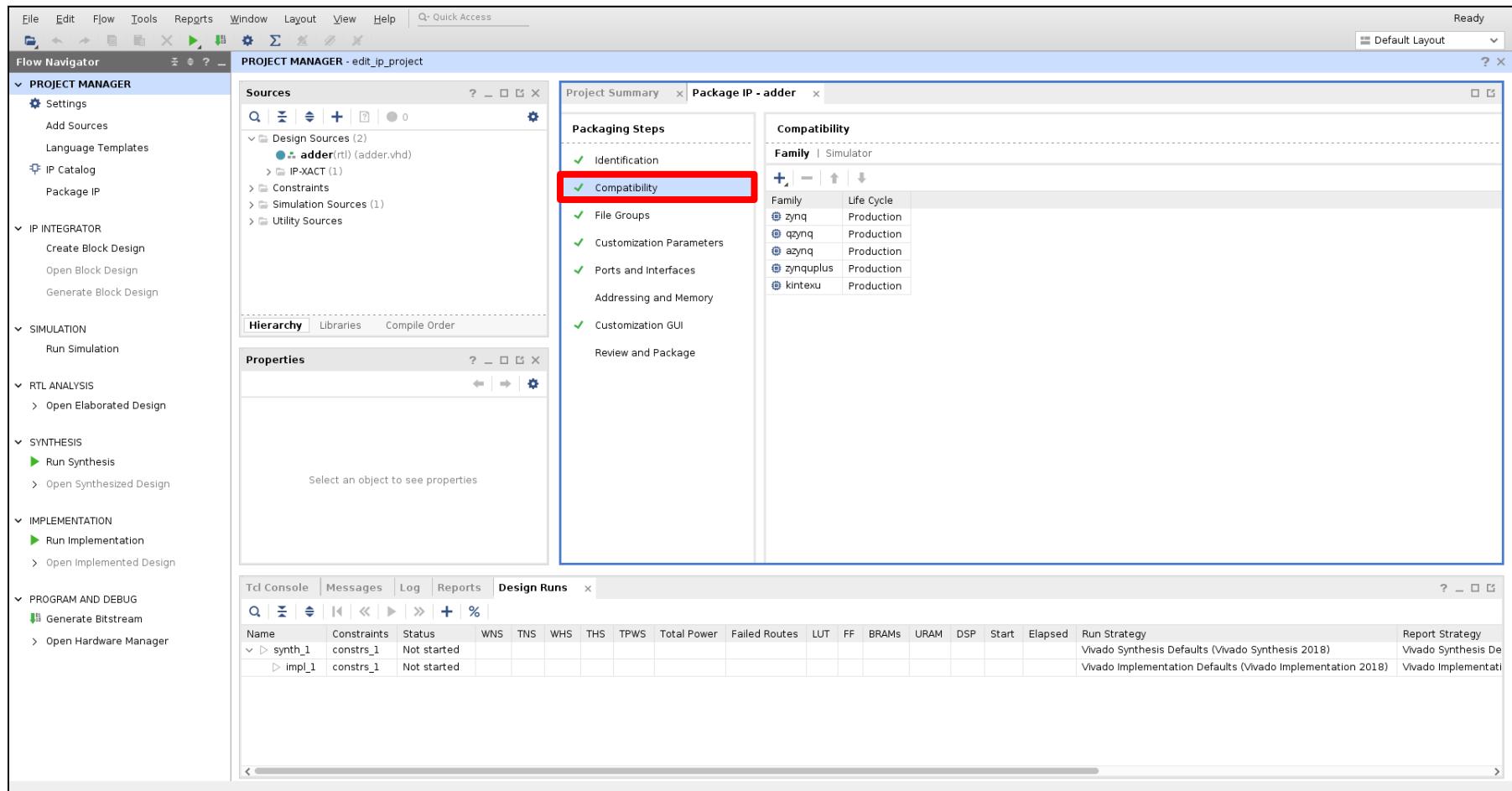
Click **Package IP**. Click **Identification**.

NOTE: Here you can specify the vendor, library, name, version, and description of your IP.



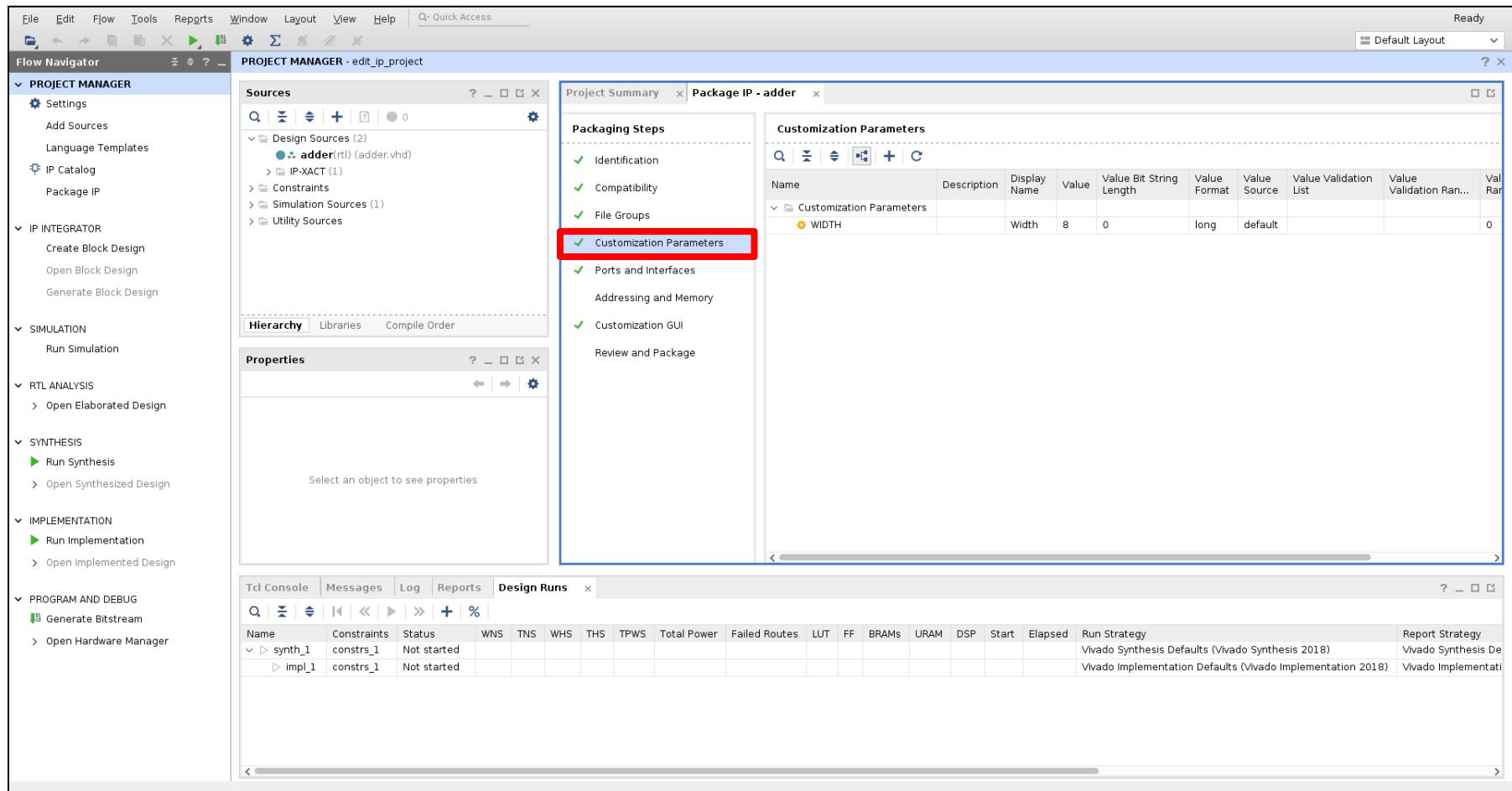
Click **Compatibility**.

NOTE: Here you can specify SoC/FPGA devices supported by your IP.



Click **Customization Parameters**.

NOTE: Generics in VHDL are inferred as customization parameters (e.g., **WIDTH**).



Click Ports and Interfaces.

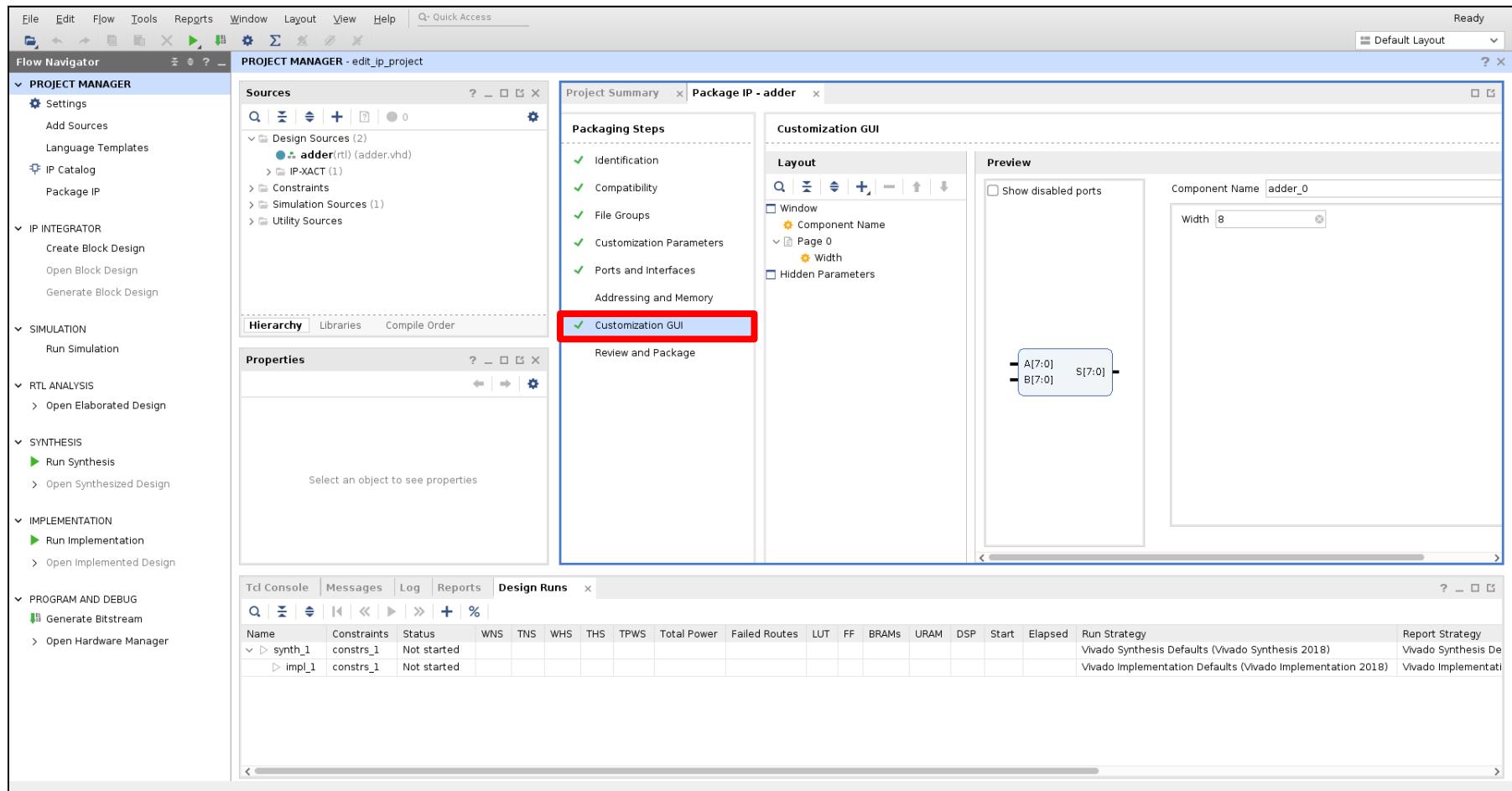
NOTE: Ports in VHDL are inferred as ports and interfaces (e.g., **A**, **B**, and **S**).

The screenshot shows the Vivado Project Manager interface. The left sidebar contains project navigation sections like Flow Navigator, Project Manager, IP Catalog, IP Integrator, Simulation, RTL Analysis, Synthesis, Implementation, Program and Debug. The main area displays the 'PROJECT MANAGER - edit_ip_project' window. In the center, there's a 'Sources' browser showing a design source 'adder(rtl) (adder.vhd)' under 'Design Sources (2)'. To the right, the 'Project Summary' and 'Package IP - adder' tabs are open. Under 'Project Summary', the 'Packaging Steps' section lists 'Identification', 'Compatibility', 'File Groups', 'Customization Parameters', and 'Ports and Interfaces' (which is highlighted with a red box). Below this is an 'Addressing and Memory' section. The 'Ports and Interfaces' table shows three ports: A, B, and S. The table columns include Name, Interface Mode, Enablement Dependency, Is Declaration, Access Handle, Access Type, Direction, Driver Value, Size Left, Size Right, Size Left Dependency, Size Right Dependency, and Type Name. Port A is an input ref port, B is an input ref port, and S is an output ref port, all with width 7. The 'Properties' panel on the left is empty, and the 'Design Runs' panel at the bottom shows a table of synthesis and implementation runs.

Name	Interface Mode	Enablement Dependency	Is Declaration	Access Handle	Access Type	Direction	Driver Value	Size Left	Size Right	Size Left Dependency	Size Right Dependency	Type Name
A					ref	in		7	0	(WIDTH - 1)		std_logic
B					ref	in		7	0	(WIDTH - 1)		std_logic
S					ref	out		7	0	(WIDTH - 1)		std_logic

Click **Customization GUI**.

NOTE: This tab illustrates the customization GUI, which allows you to customize parameters of your IP in a block design.



Time to simulate your design. Click **Add Sources**.

The screenshot shows the Vivado Project Manager interface. On the left, the Project Navigator pane is open, displaying various project management options. The 'Add Sources' button under the 'PROJECT MANAGER' section is highlighted with a red box. The main workspace shows the 'PROJECT MANAGER - edit_ip_project' window. This window includes tabs for 'Sources', 'Properties', and 'Design Runs'. The 'Sources' tab displays a tree view of design sources, including an 'adder' item under 'Design Sources (2)'. The 'Properties' tab shows a message to 'Select an object to see properties'. The 'Design Runs' tab lists two runs: 'synth_1' and 'impl_1', both of which are currently 'Not started'. The central part of the window is the 'Project Summary' panel for the 'Package IP - adder' project. It contains sections for 'Packaging Steps' (with 'Identification' selected), 'Identification' (filling in vendor, library, name, version, display name, and description), and 'Categories' (listing '/UserIP').



Select **Add or create simulation sources**. Click **Next**.

VIVADO
HLS Editions

Add Sources

This guides you through the process of adding and creating sources for your project

Add or create constraints

Add or create design sources

Add or create simulation sources

1

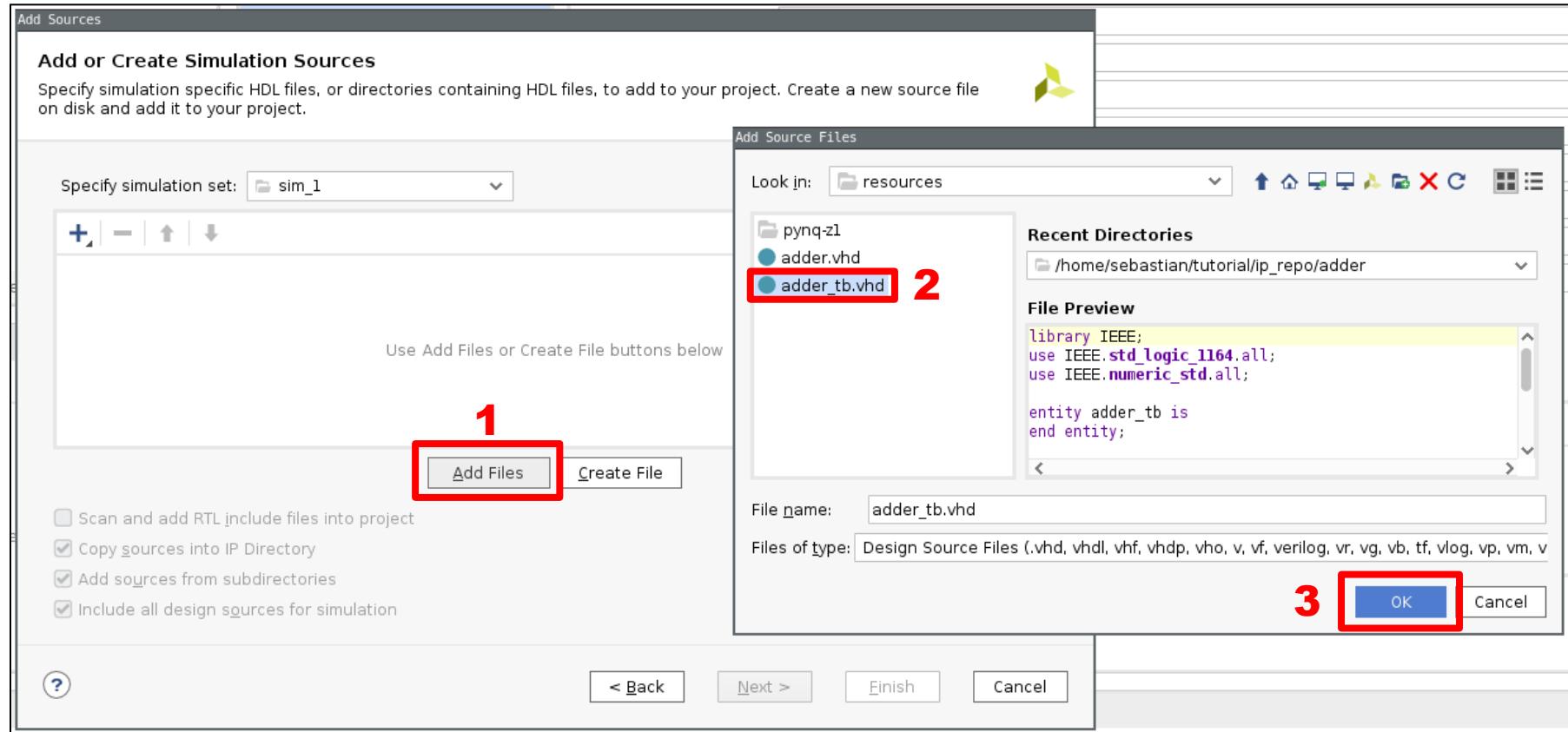
XILINX

? < Back **Next >** Finish Cancel

2



Click **Add Files**. Navigate to `Lab_1_repo/resources/adder_tb.vhd`. Click **OK**.



Click **Finish**. Note the checked/unchecked boxes

Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

Specify simulation set: sim_1

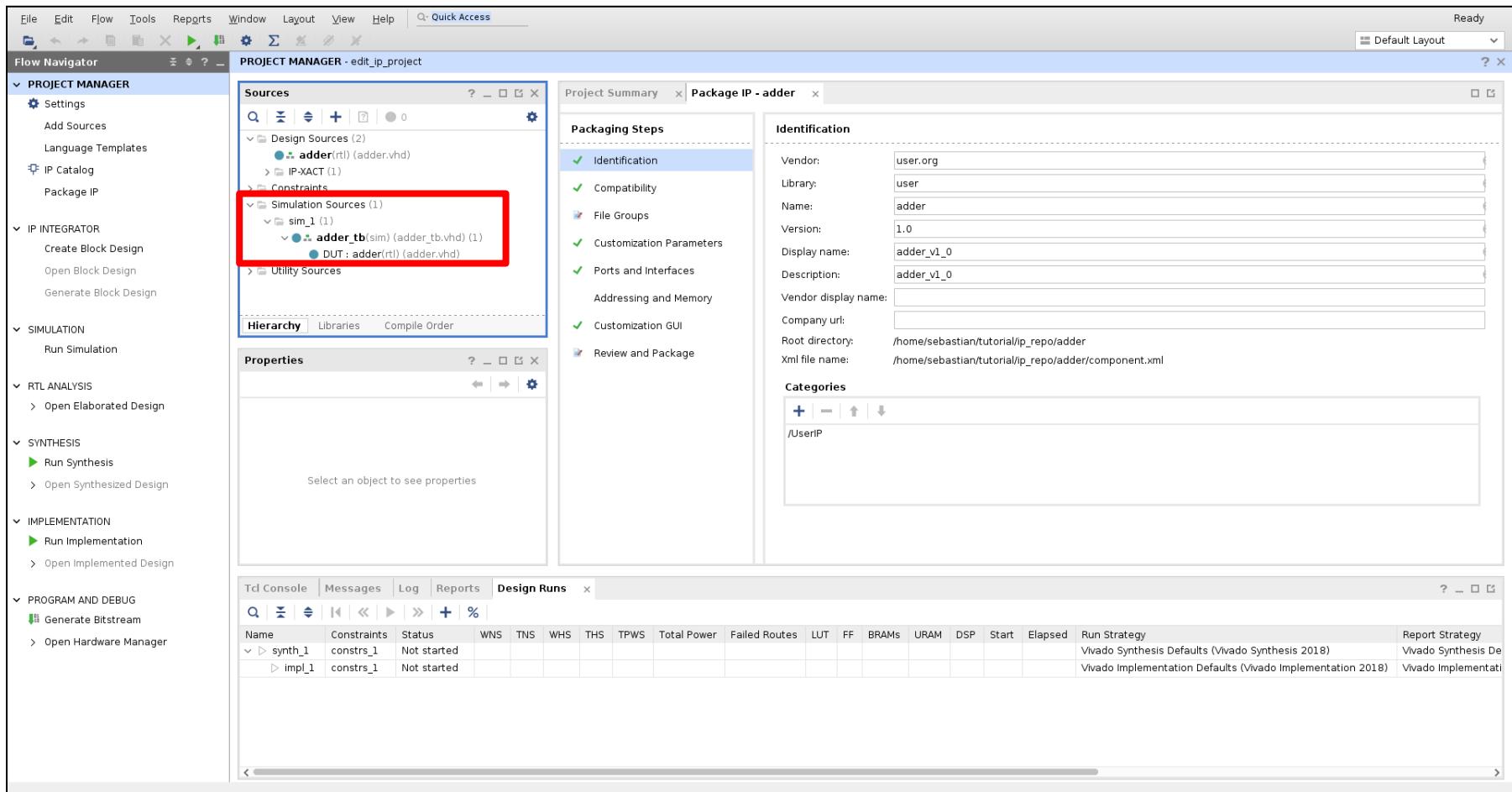
	Index	Name	Library	Location
●	1	adder_tb.vhd	xil_defaultlib	/home/sebastian/tutorial/resources

Add Files **Create File**

Scan and add RTL include files into project
 Copy sources into IP Directory
 Add sources from subdirectories
 Include all design sources for simulation

Finish (Red box)
Cancel

`adder_tb.vhd` has been added to the project as a simulation source.



adder_tb.vhd. The main process assigns loop iterators to the adder inputs every 10 ns. Continue.

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

entity adder_tb is
end entity;

architecture sim of adder_tb is
component adder is
generic (
    WIDTH : positive := 8
);
port (
    A      : in  std_logic_vector(WIDTH-1 downto 0);
    B      : in  std_logic_vector(WIDTH-1 downto 0);
    S      : out std_logic_vector(WIDTH-1 downto 0)
);
end component;

constant WIDTH : positive := 4;
signal A      : std_logic_vector(WIDTH-1 downto 0);
signal B      : std_logic_vector(WIDTH-1 downto 0);
signal S      : std_logic_vector(WIDTH-1 downto 0);

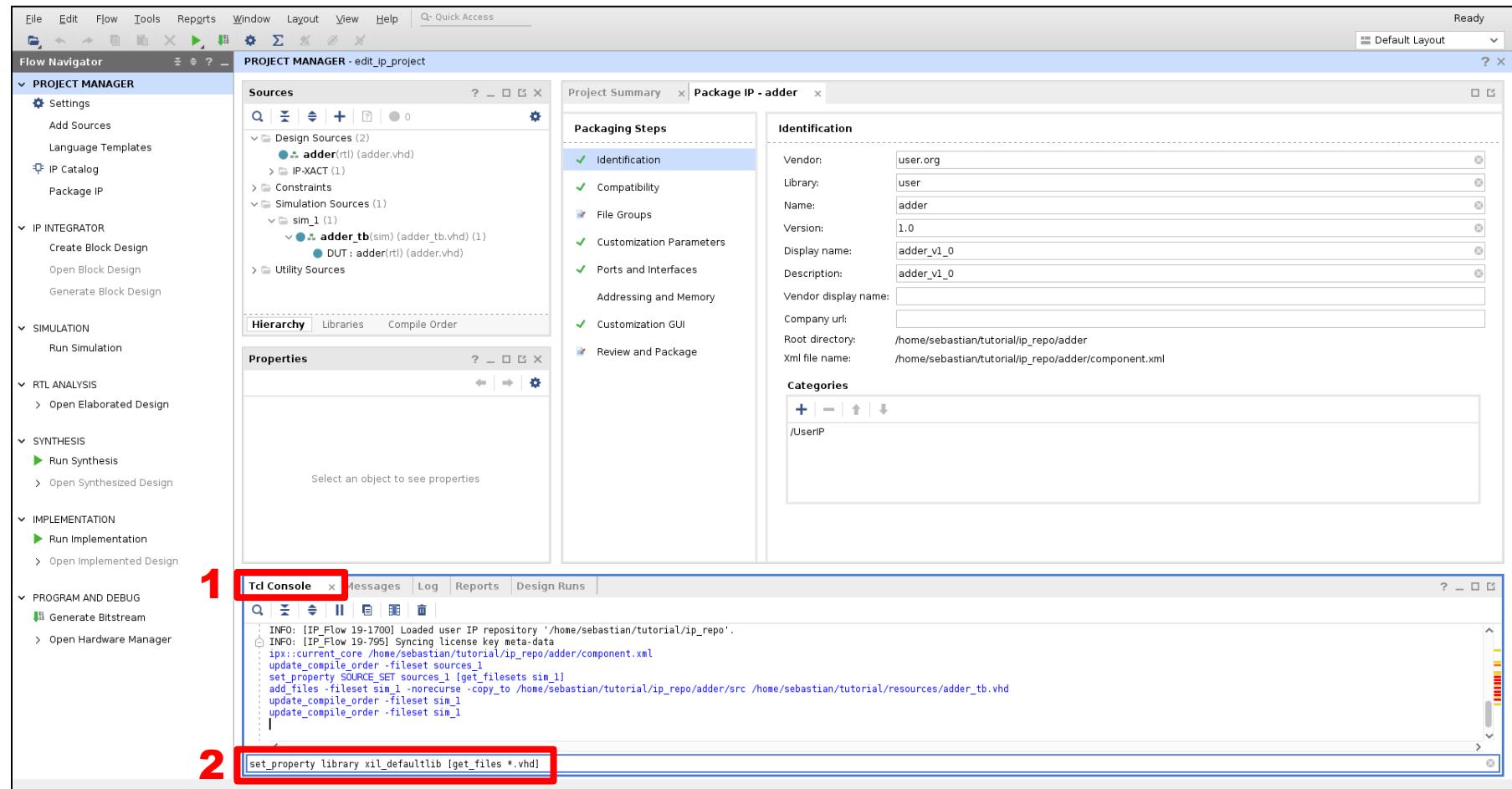
begin
DUT: adder
generic map (
    WIDTH => WIDTH
)
port map (
    A      => A,
    B      => B,
    S      => S
);

process
begin
    for i in 0 to 2**WIDTH-1 loop
        for j in 0 to 2**WIDTH-1 loop
            A <= std_logic_vector(to_unsigned(i, WIDTH));
            B <= std_logic_vector(to_unsigned(j, WIDTH));
            wait for 10 ns;
        end loop;
    end loop;
    wait;
end process;
end architecture;
```

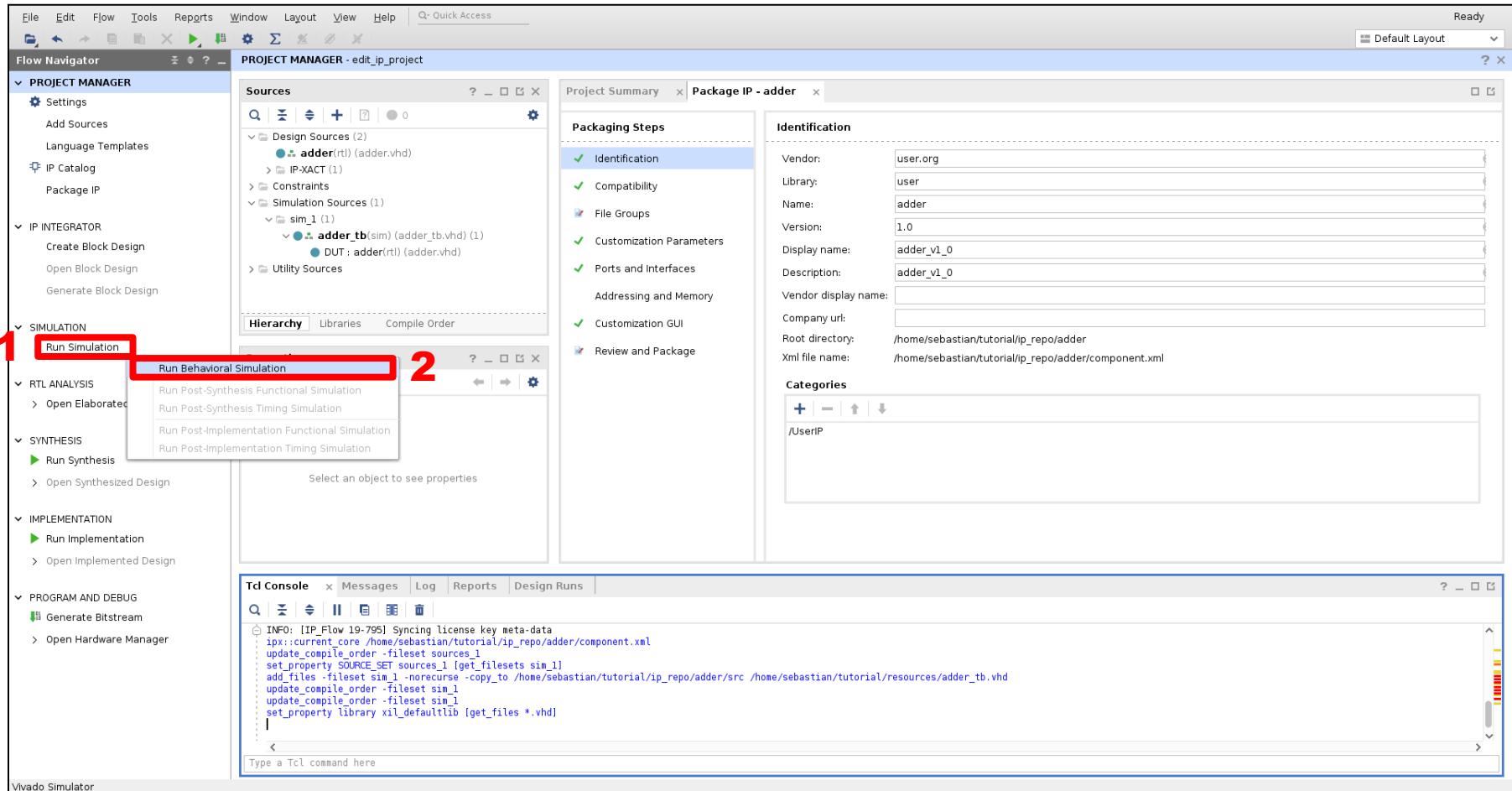


Click **Tcl Console** tab.

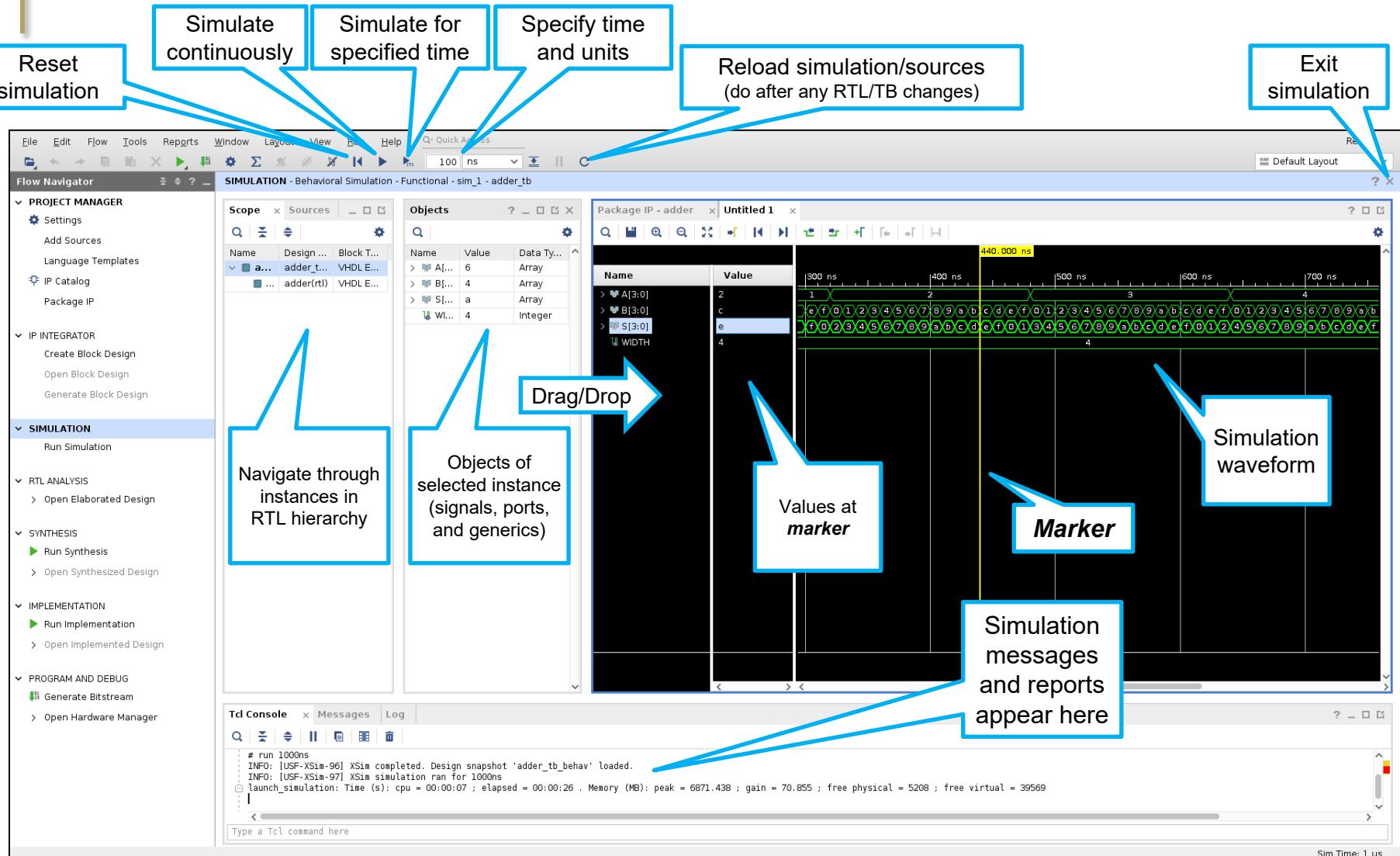
Enter Tcl command **set_property library xil_defaultlib [get_files *.vhd]**.



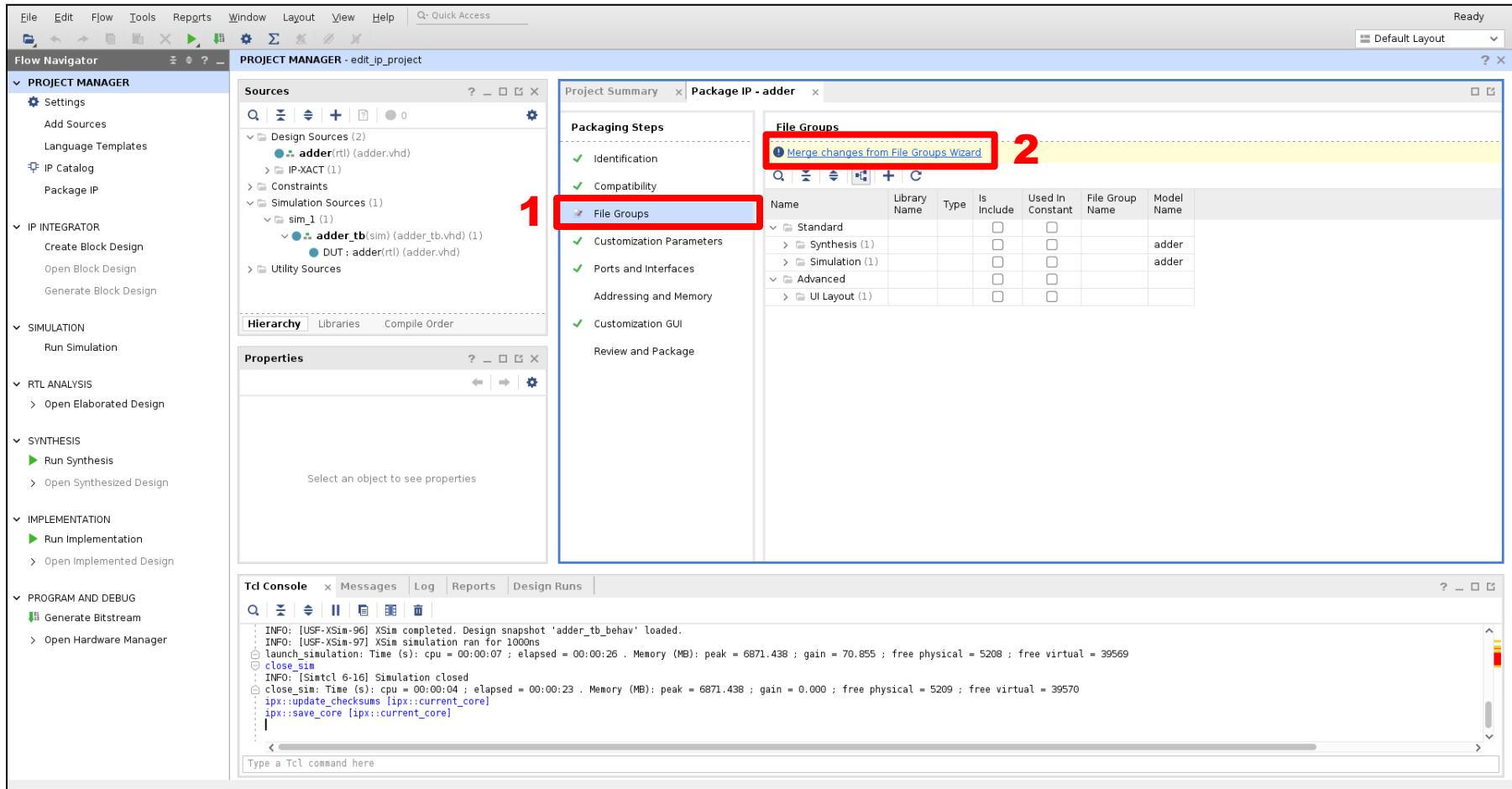
Click **Run Simulation**. Click **Behavioral Simulation**.



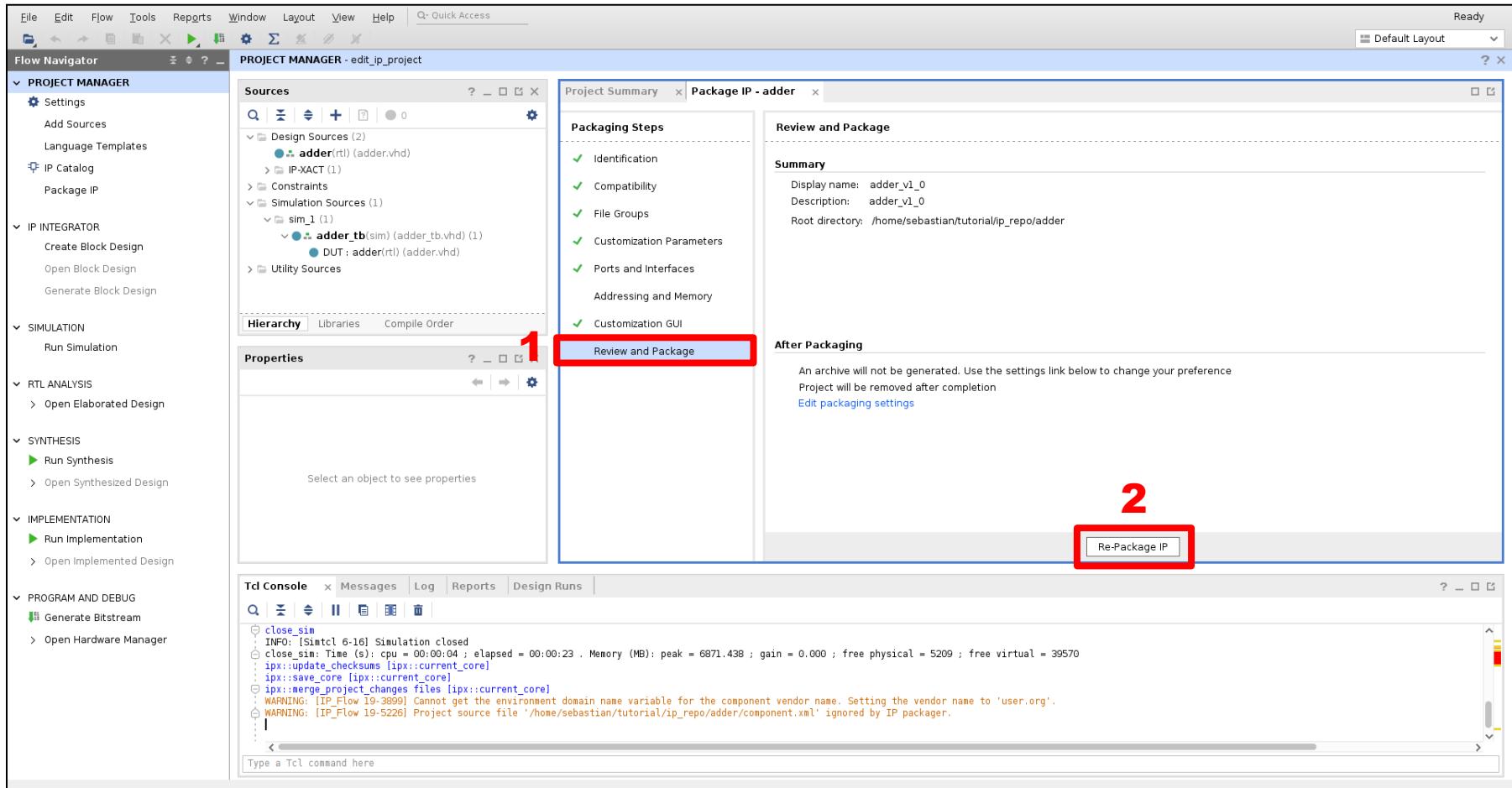
This is the Simulation GUI. Continue.



Click **File Groups**. Click **Merge changes from File Groups Wizards**.

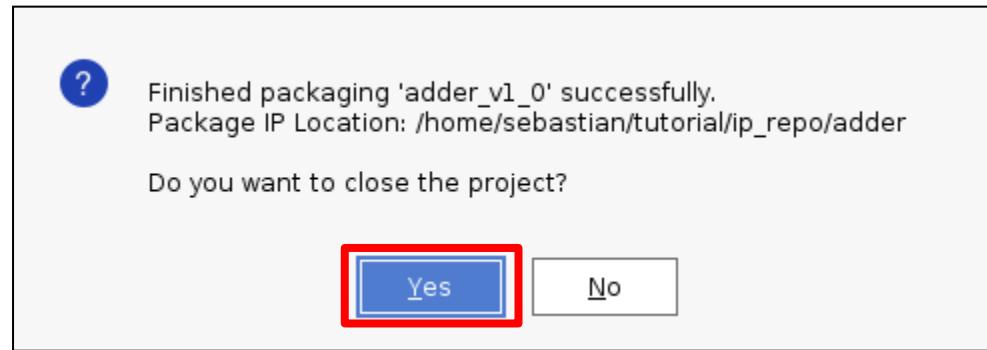


Click **Review and Package**. Click **Re-Package IP**.



Click Yes.

Note: for future projects, you might want to leave this instance opened in case you need to edit and re-package your design.



We return to the main Vivado instance. Continue.

The screenshot shows the Vivado 2018.3 software interface. The left sidebar contains a tree view of project management, IP integrator, simulation, RTL analysis, synthesis, implementation, and program/debug options. The main area is divided into several panels:

- PROJECT MANAGER - example**: Shows sources (Design Sources, Constraints, Simulation Sources containing sim_1, Utility Sources), Hierarchy, Libraries, and Compile Order.
- Properties**: A panel where no object is selected, displaying placeholder text "Select an object to see properties".
- Project Summary**: Overview and Dashboard sections. **Settings** tab shows:
 - Project name: example
 - Project location: /home/sebastian/tutorial/example
 - Product family: Zynq-7000
 - Project part: PYNQ-Z1 (xc7z020clg400-1)
 - Top module name: Not defined
 - Target language: VHDL
 - Simulator language: Mixed
- Board Part**:
 - Display name: PYNQ-Z1
 - Board part name: www.digilentinc.com:pynq-z1:part0:1.0
 - Connectors: No connections
 - Repository path: /data/opt/Xilinx/Vivado/2018.3/data/boards/board_files
 - URL: <http://www.pynq.io>
 - Board overview: PYNQ-Z1
- Synthesis** and **Implementation** sections: Both show "Not started" status, no errors or warnings, and default strategies.
- Design Runs**: A table showing two runs:

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy	Report Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2018)	Vivado Synthesis Default Reports
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2018)	Vivado Implementation Default Reports

Part 4: Integrating a System

Next, we will integrate a system interfacing the **axi_regmap** and **adder** IP to the *Zynq PS*.

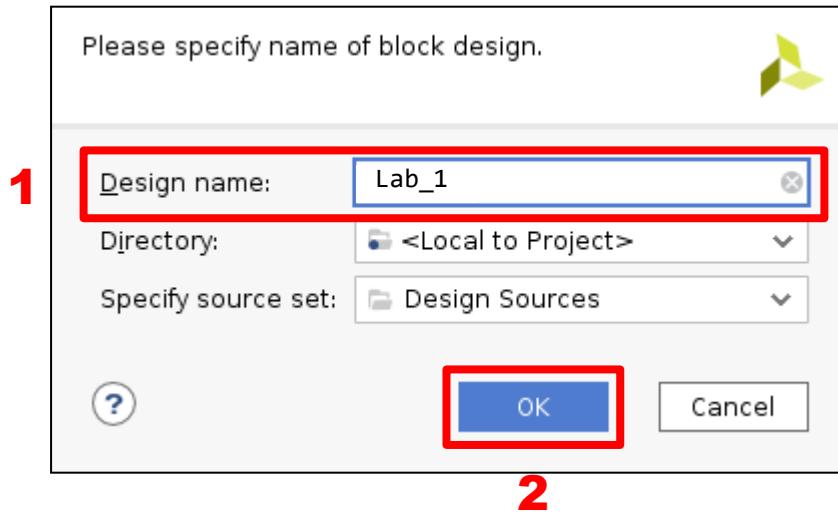


Click *Create Block Design*.

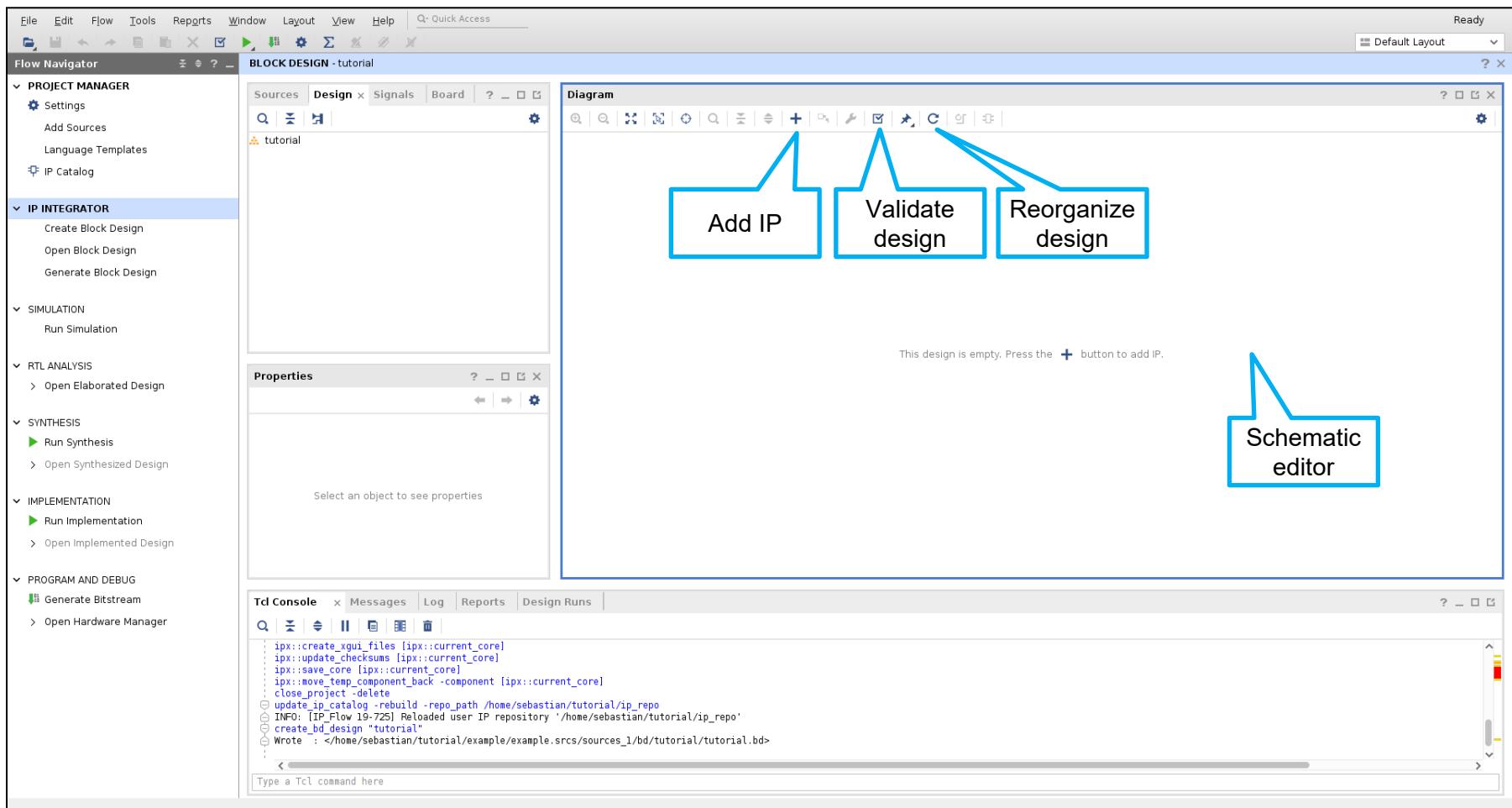
The screenshot shows the Vivado 2018.3 graphical user interface. The left sidebar contains a tree view of project management, IP integrator, simulation, RTL analysis, synthesis, implementation, and program/debug options. A red box highlights the 'Create Block Design' option under the IP Integrator section. The main workspace is divided into several panes: 'Sources' (listing design sources, constraints, simulation sources like 'sim_1', and utility sources), 'Project Summary' (containing an 'Overview' tab and a 'Settings' tab with detailed project information such as name, location, family, part, module, language, and simulator), 'Board Part' (listing display name, board part name, connectors, repository path, URL, and board overview), 'Synthesis' (status: Not started, messages: No errors or warnings, part: xc7z020clg400-1, strategy: Vivado Synthesis Defaults, report strategy: Vivado Synthesis Default Reports), and 'Implementation' (status: Not started, messages: No errors or warnings, part: xc7z020clg400-1, strategy: Vivado Implementation Defaults, report strategy: Vivado Implementation Default Reports). At the bottom, the 'Design Runs' tab is active, showing a table of runs: 'synth_1' (constraints: 'constrs_1', status: Not started) and 'impl_1' (constraints: 'constrs_1', status: Not started). The table includes columns for Name, Constraints, Status, WNS, TNS, WHS, THS, TPWS, Total Power, Failed Routes, LUT, FF, BRAMs, URAM, DSP, Start, Elapsed, Run Strategy, and Report Strategy.

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy	Report Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2018)	Vivado Synthesis Default Reports (Vivado Synthesis 2018)
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2018)	Vivado Implementation Default Reports (Vivado Implementation 2018)

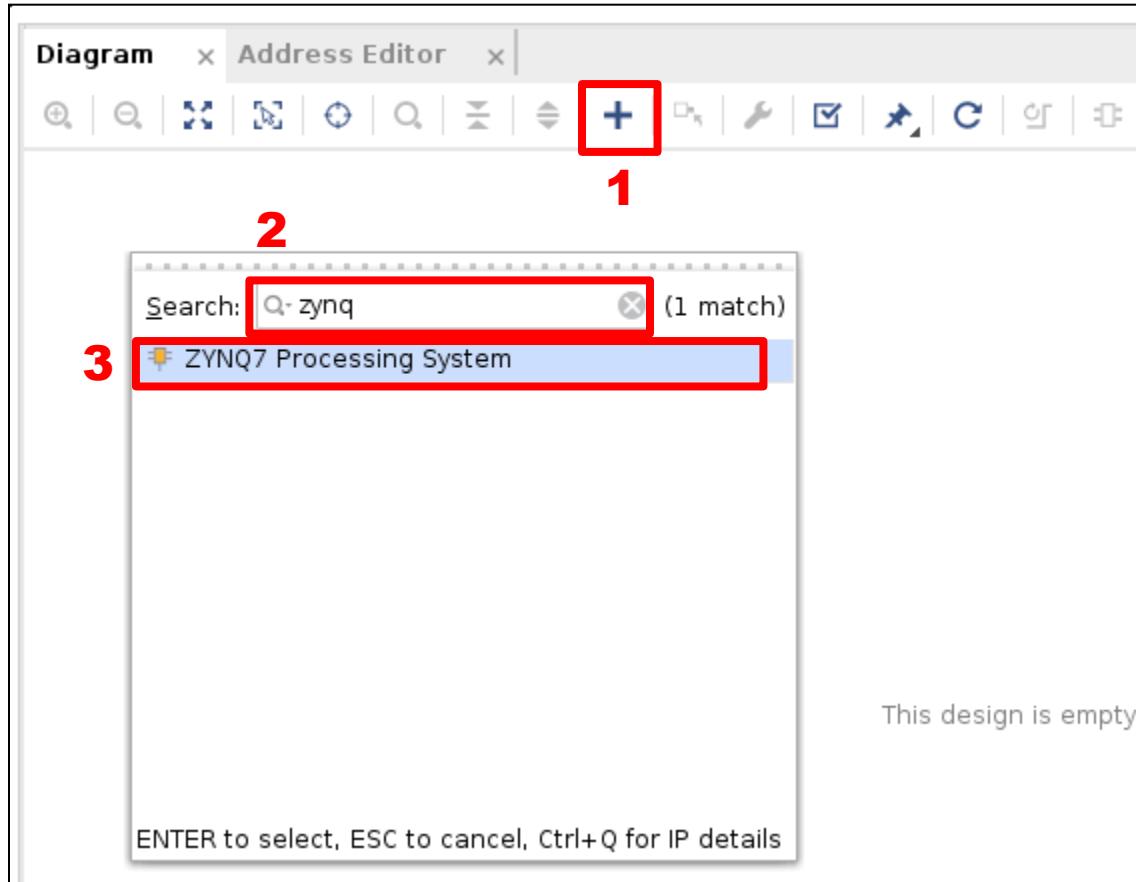
Enter design name **Lab_1**. Click **OK**.



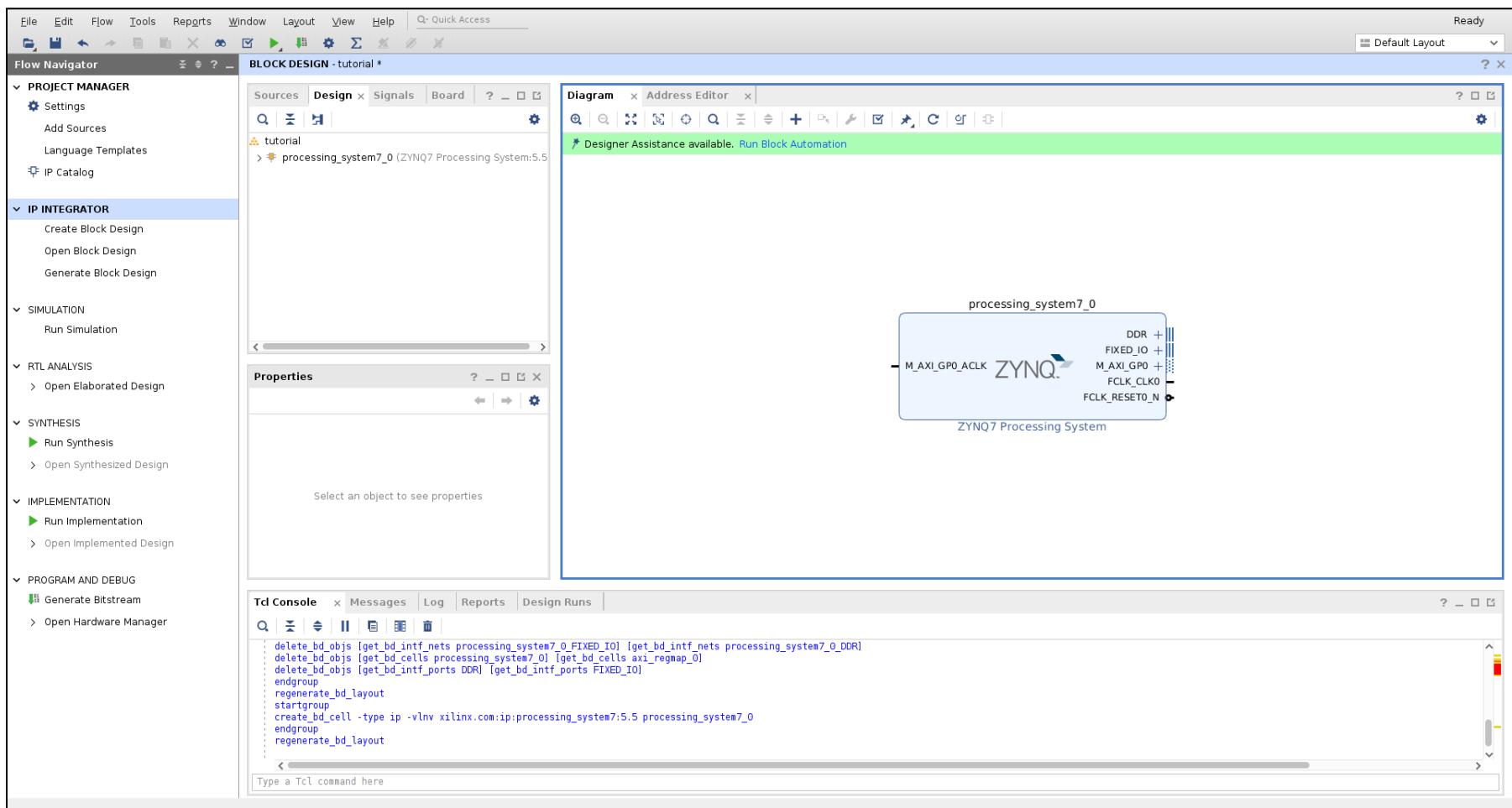
The block design **Lab_1** was created. The diagram (schematic editor) is shown. Continue.



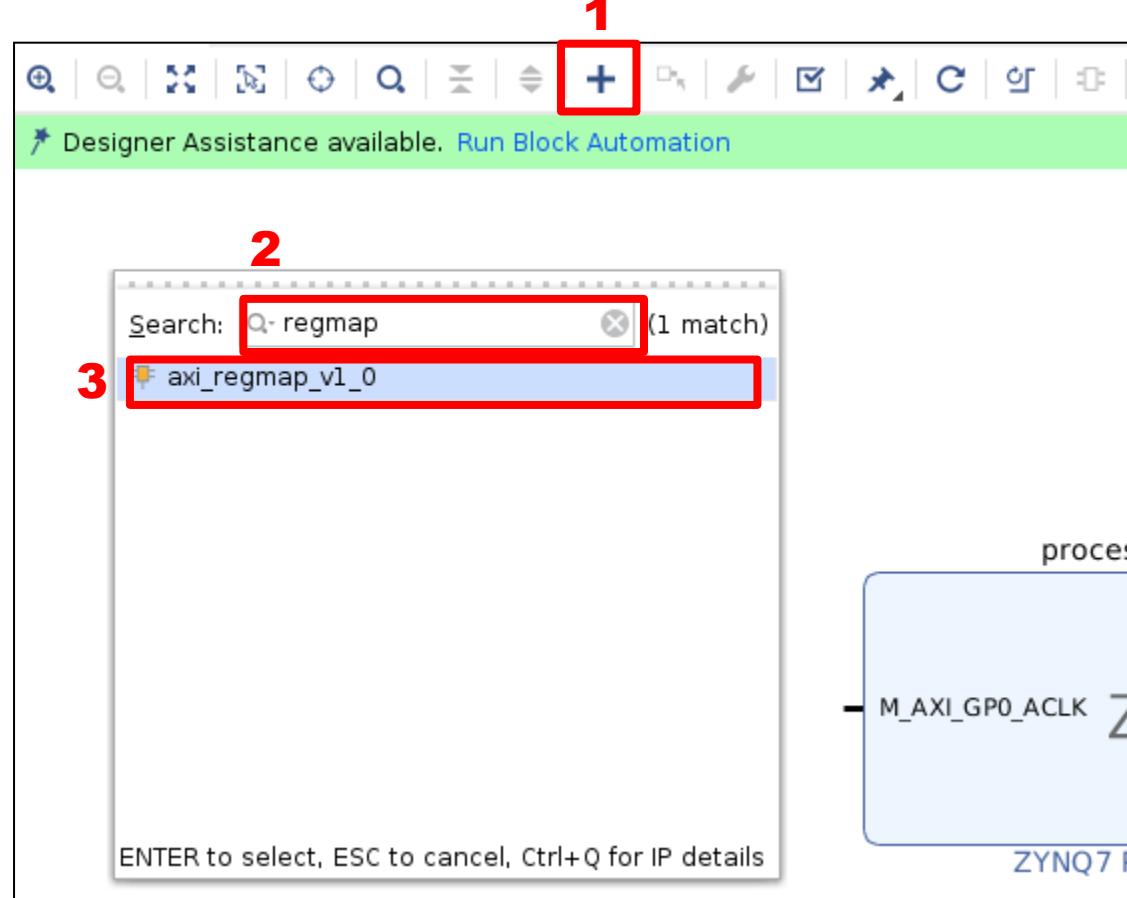
Click + (Add IP). Search **zynq**. Double-click **ZYNQ7 Processing System**.



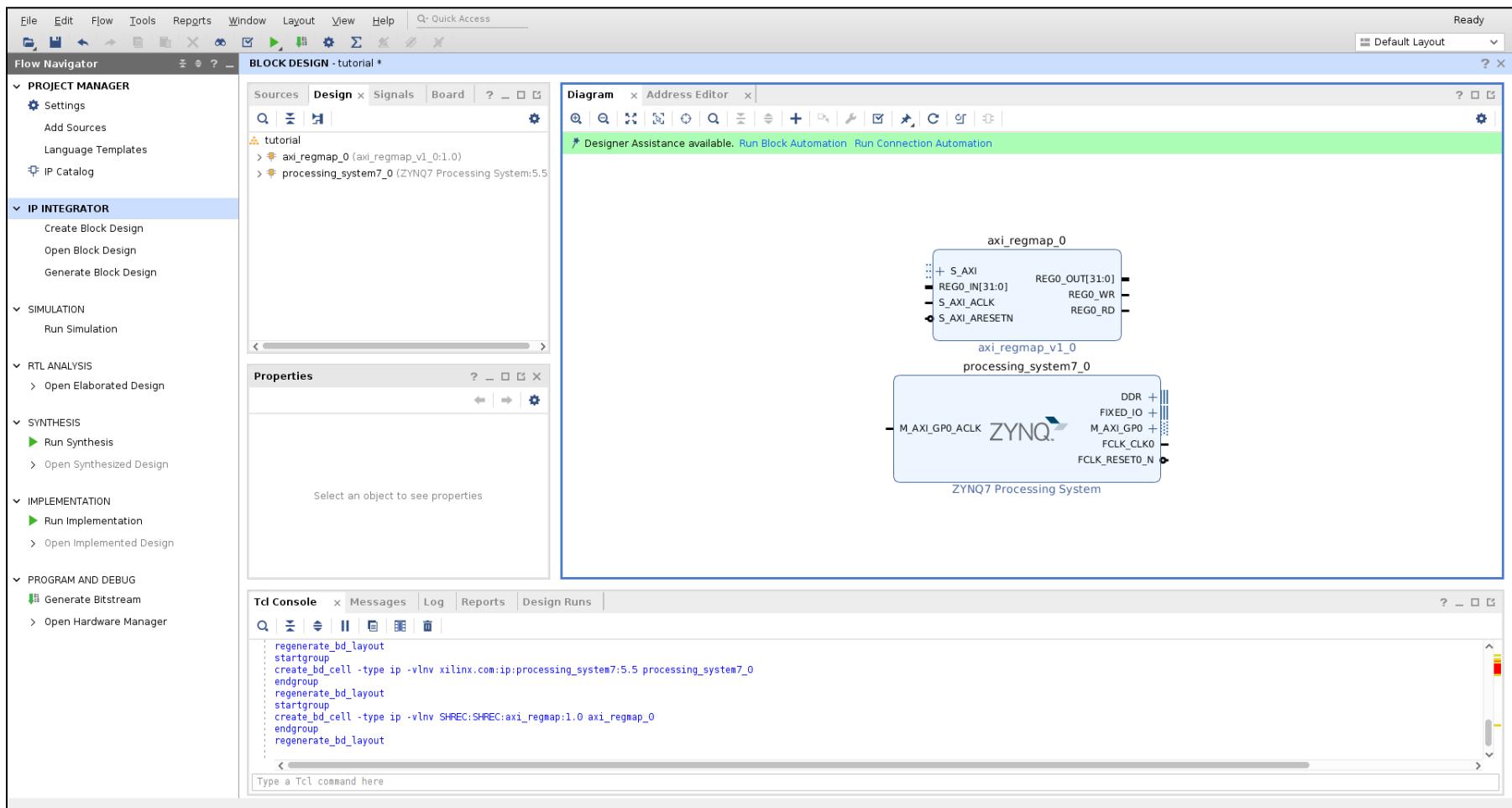
Wild **ZYNQ7 Processing System** appeared! The instance name is **processing_system_0**. Continue.



Click + (Add IP). Search *regmap*. Double-click *axi_regmap_v1_0*.



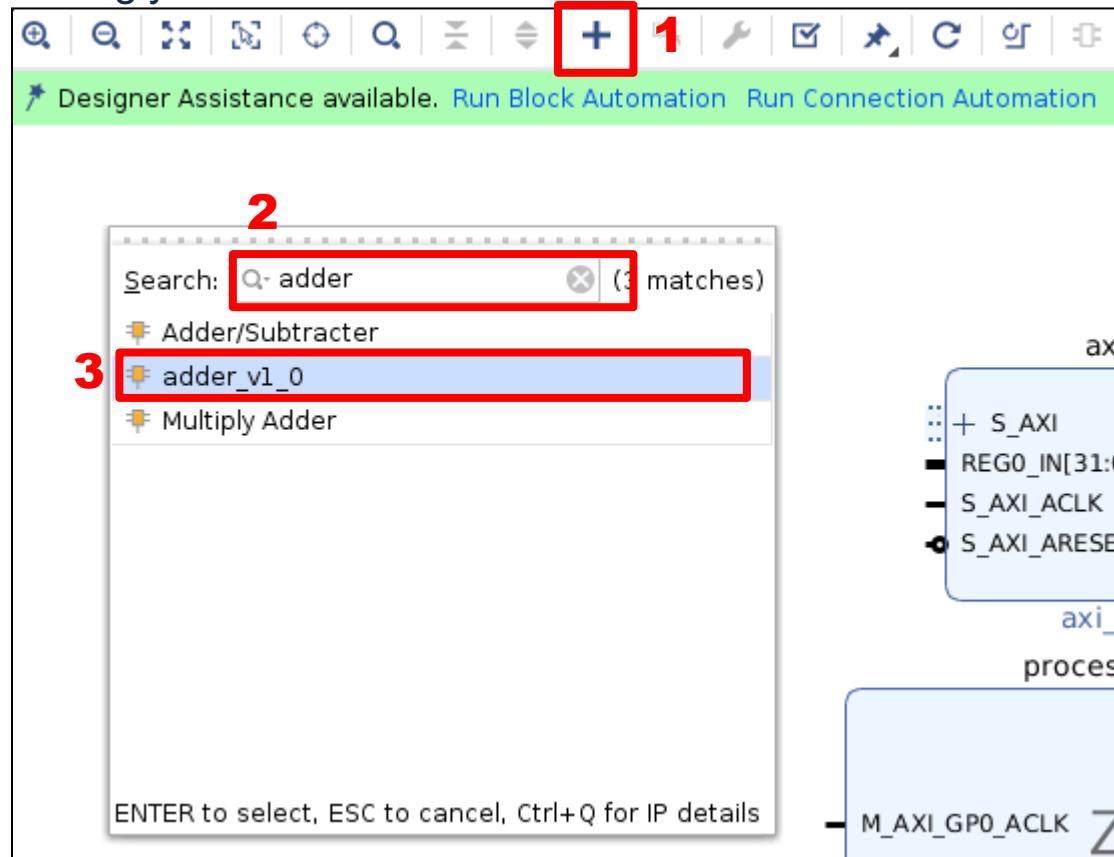
Wild ***axi_regmap_v1_0*** appeared! The instance name is ***axi_regmap_0***. Continue.



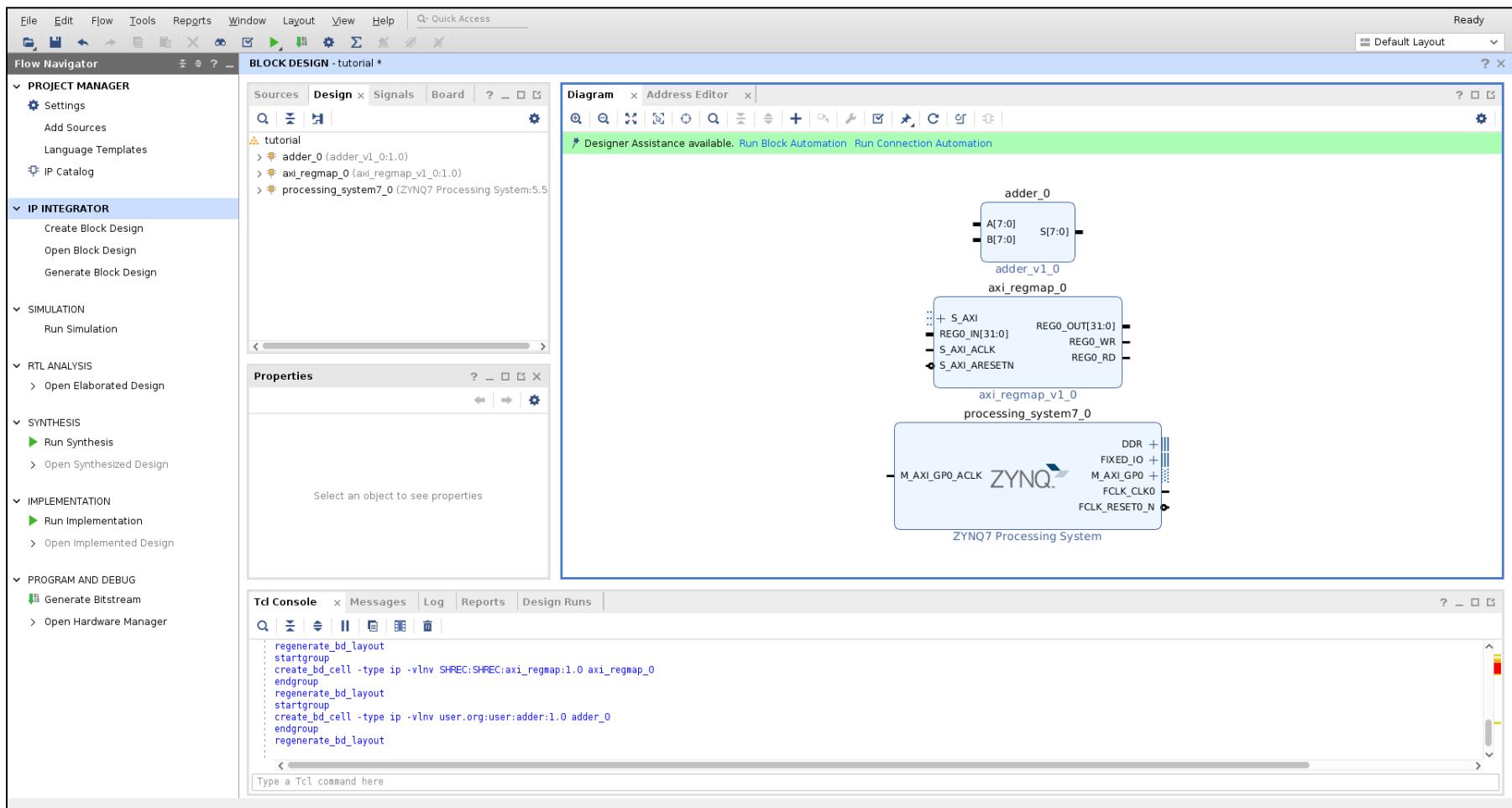
Click **+** (Add IP). Search **adder**. Double-click **adder_v1_0**.

Note 1: if you can't find it, click IP Catalog -> right click in the opened window -> refresh

Note 2: you can also switch to the **Sources** tab, click on **adder.vhd** and drag it to the Diagram window. (if you did it this way, you can skip slides 83-91. These are to edit the VHDL source of an IP added through **+** (Add IP) or **Note 1**. To edit a module added via **Note 2**, you just double click on the design file (.vhd), edit it, then the block diagram will be updated accordingly.

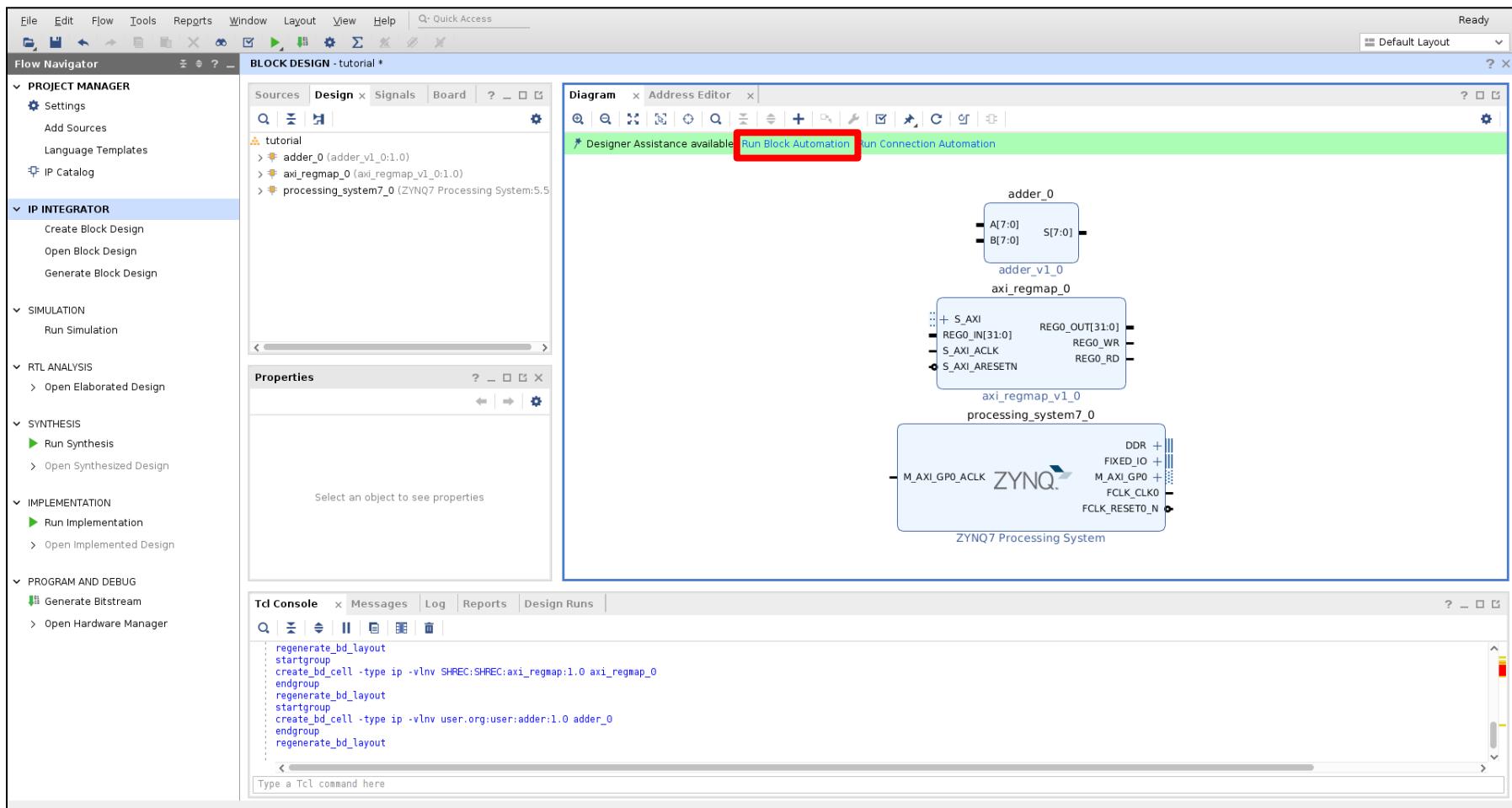


Wild **adder_v1_0** appeared! The instance name is **adder_0**. Continue.



Click *Run Block Automation*.

NOTE: This process will configure the **processing_system_0** to the board configuration specified by the PYNQ-Z1 board files.



Click **OK**.

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

Description

This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset. This action cannot be undone. Zynq7 block automation applies current board preset and generates external connections for FIXED_IO, Trigger and DDR interfaces.

NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration.

Instance: /processing_system7_0

Options

Make Interface External: FIXED_IO, DDR

Apply Board Preset:

Cross Trigger In:

Cross Trigger Out:

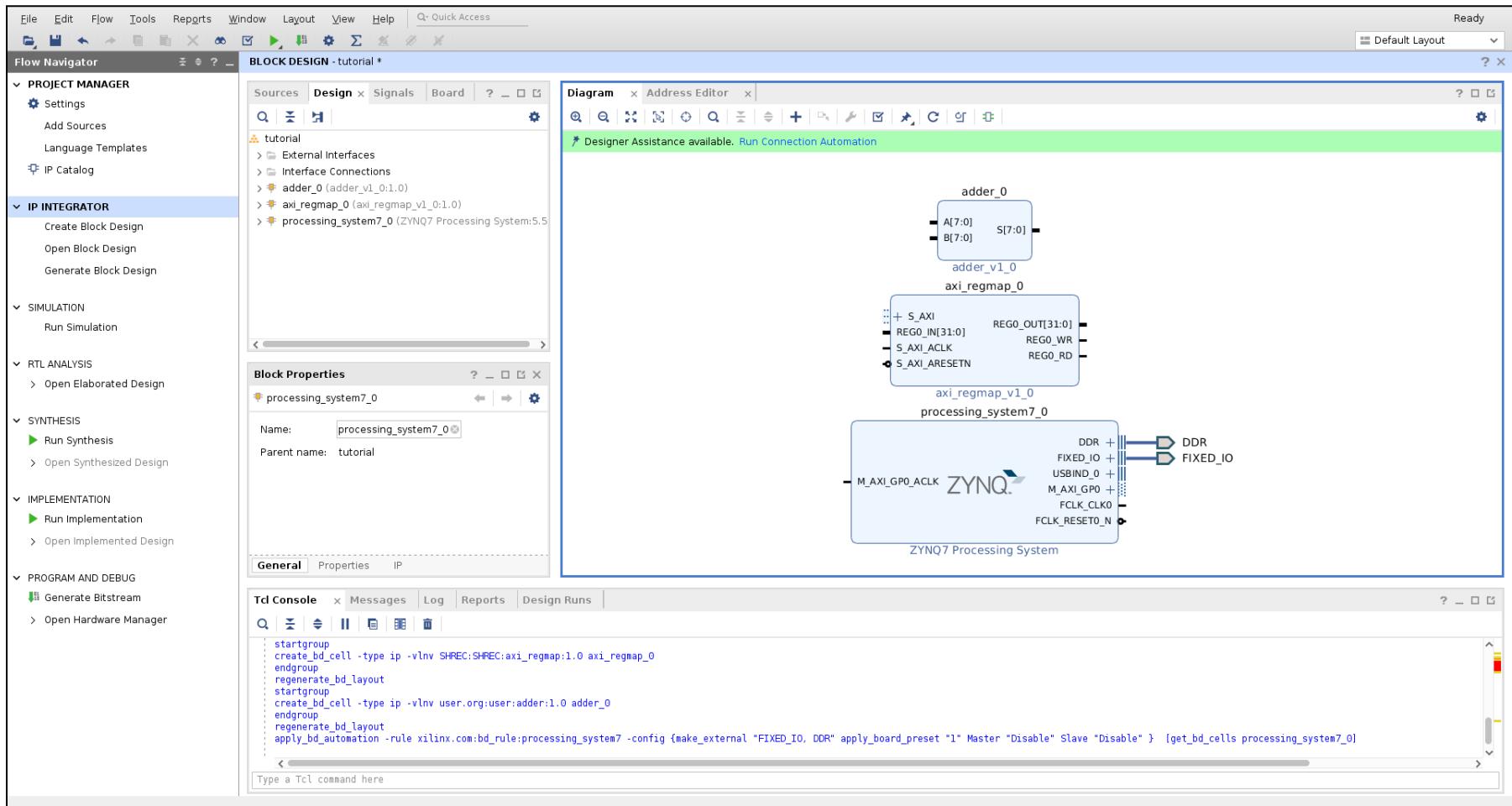
?

OK

Cancel

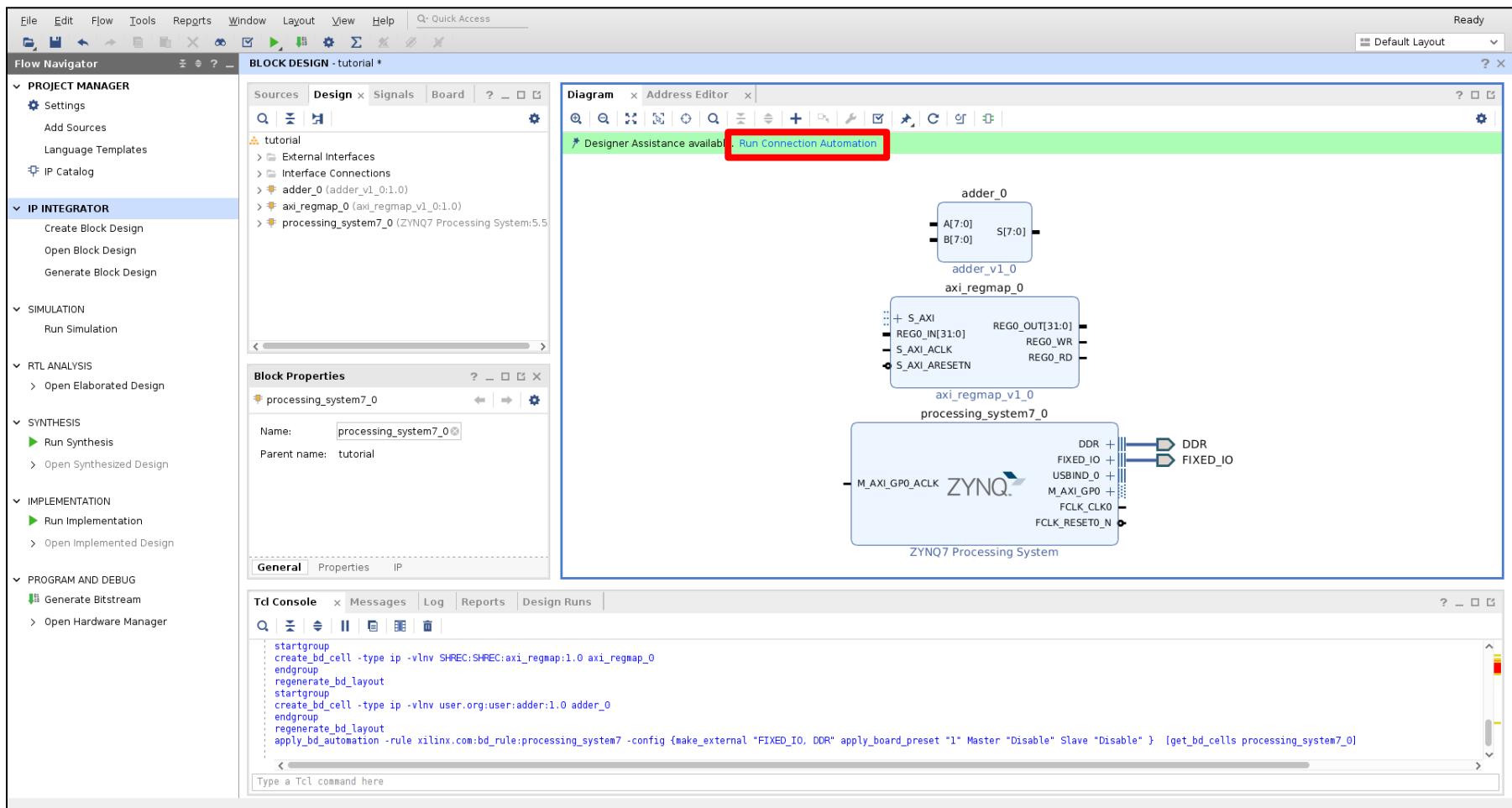


The processing_system_0 IP has been configured. Continue.



Click **Run Connection Automation**.

NOTE: This process will automatically instantiate an AXI connect between the **processing_system_0** and **axi_regmap_0** IP.



Click **OK**.

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

Description

Connect Slave interface (/axi_regmap_0/S_AXI) to a selected Master address space.

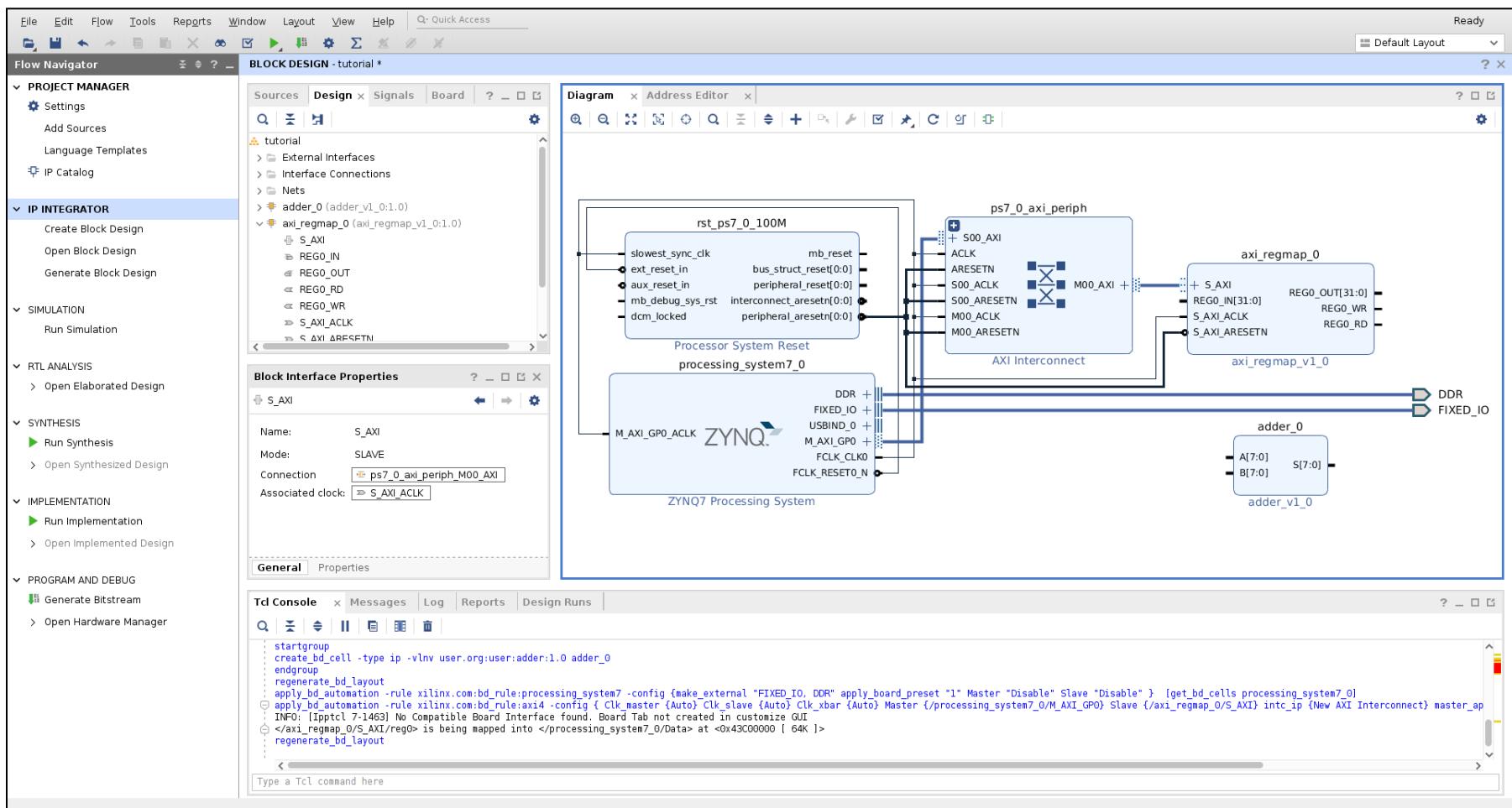
Options

Master	/processing_system7_0/M_AXI_0
Bridge IP	New AXI Interconnect
Clock source for driving Interconnect IP	Auto
Clock source for Master interface	Auto
Clock source for Slave interface	Auto

?

OK Cancel

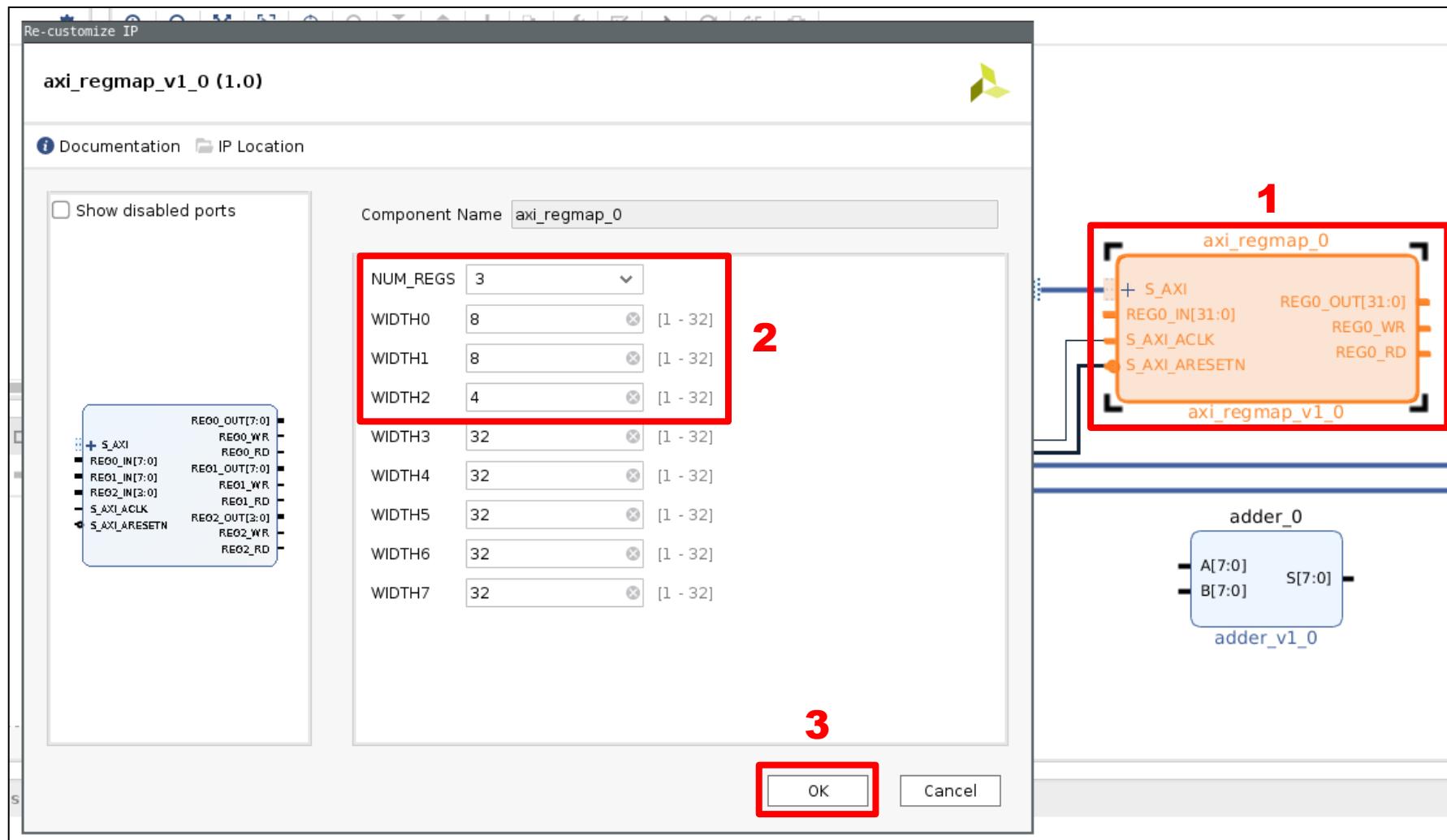
An AXI connection between the **processing_system_0** and **axi_regmap_0** IP was made. Supporting IP (**AXI Interconnect** and **Processor System Reset**) were also instantiated. Continue.



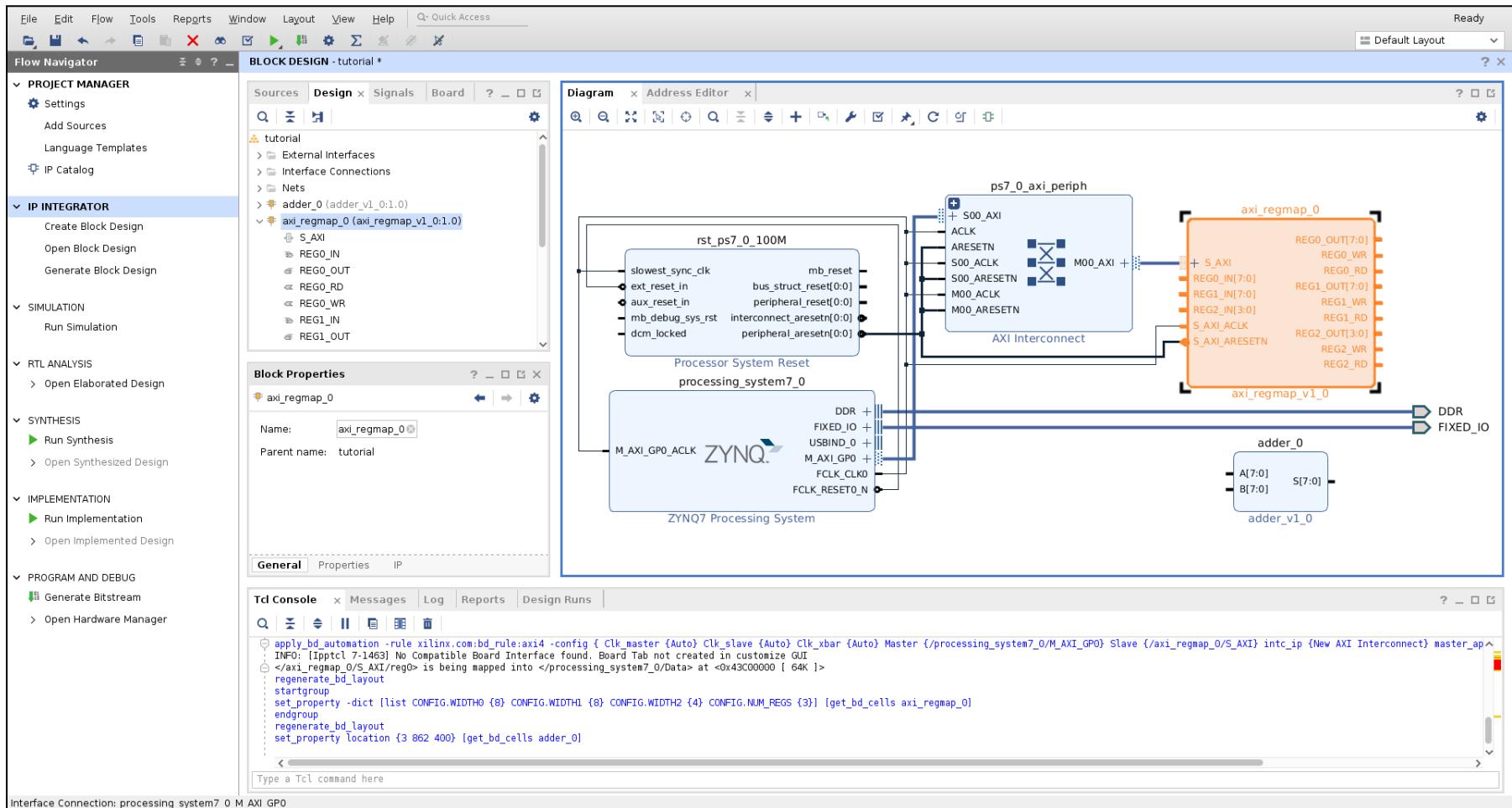
Double-click **axi_regmap_0**.

Set parameters **NUM_REGS** → 3, **WIDTH0** → 8, **WIDTH1** → 8, and **WIDTH2** → 4.

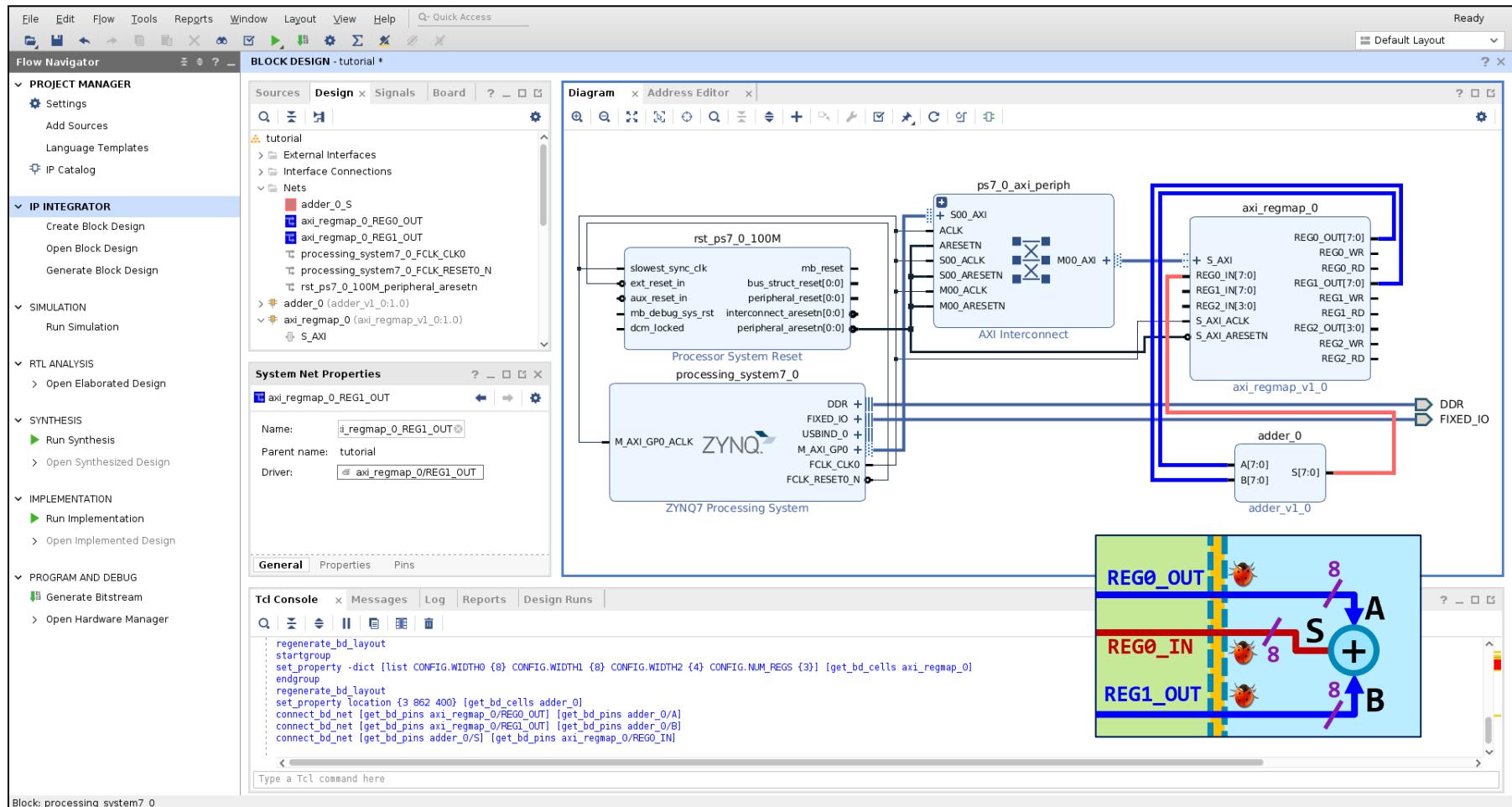
Click **OK**.



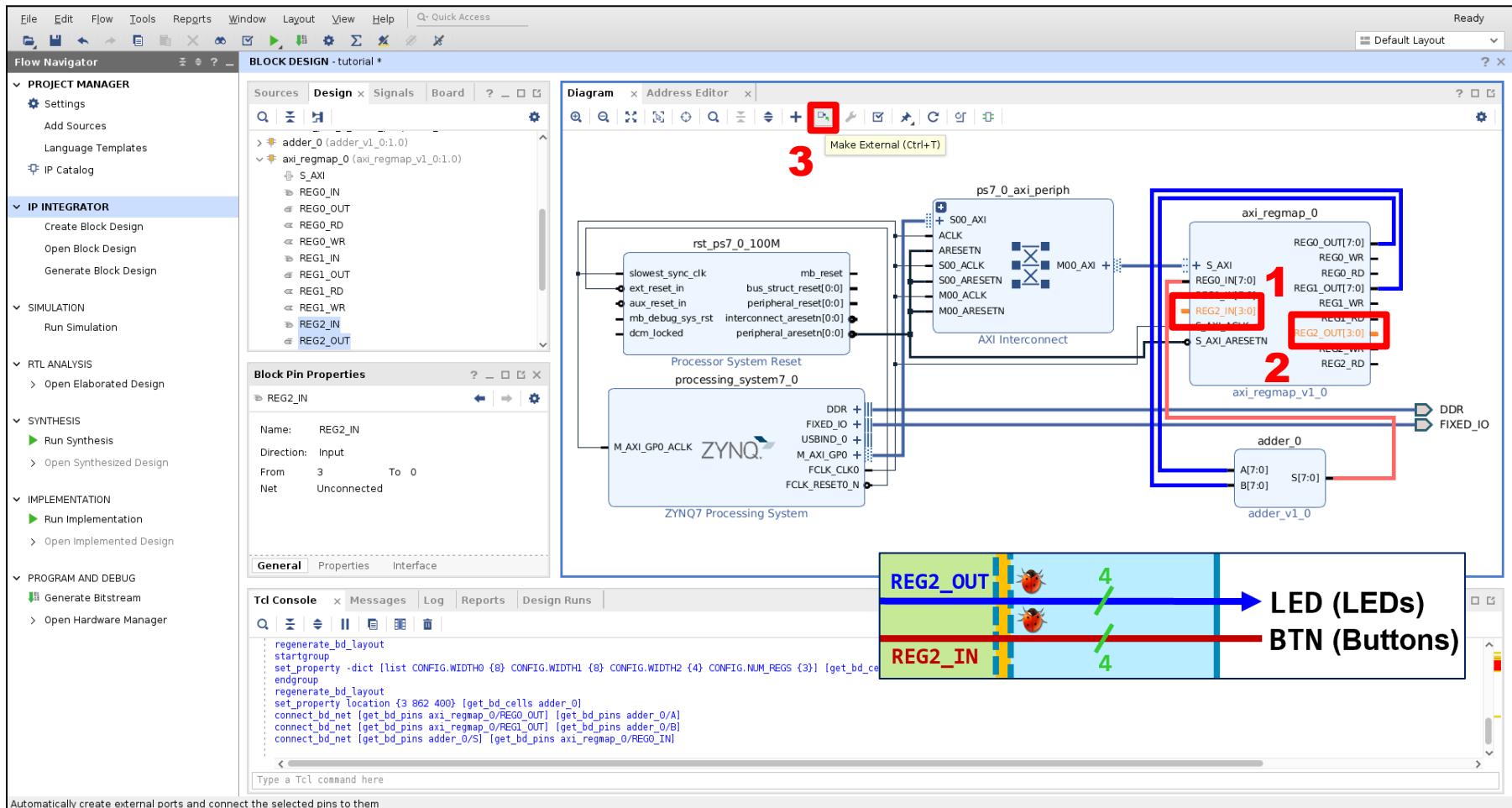
The axi_regmap_0 IP block has changed. Continue.



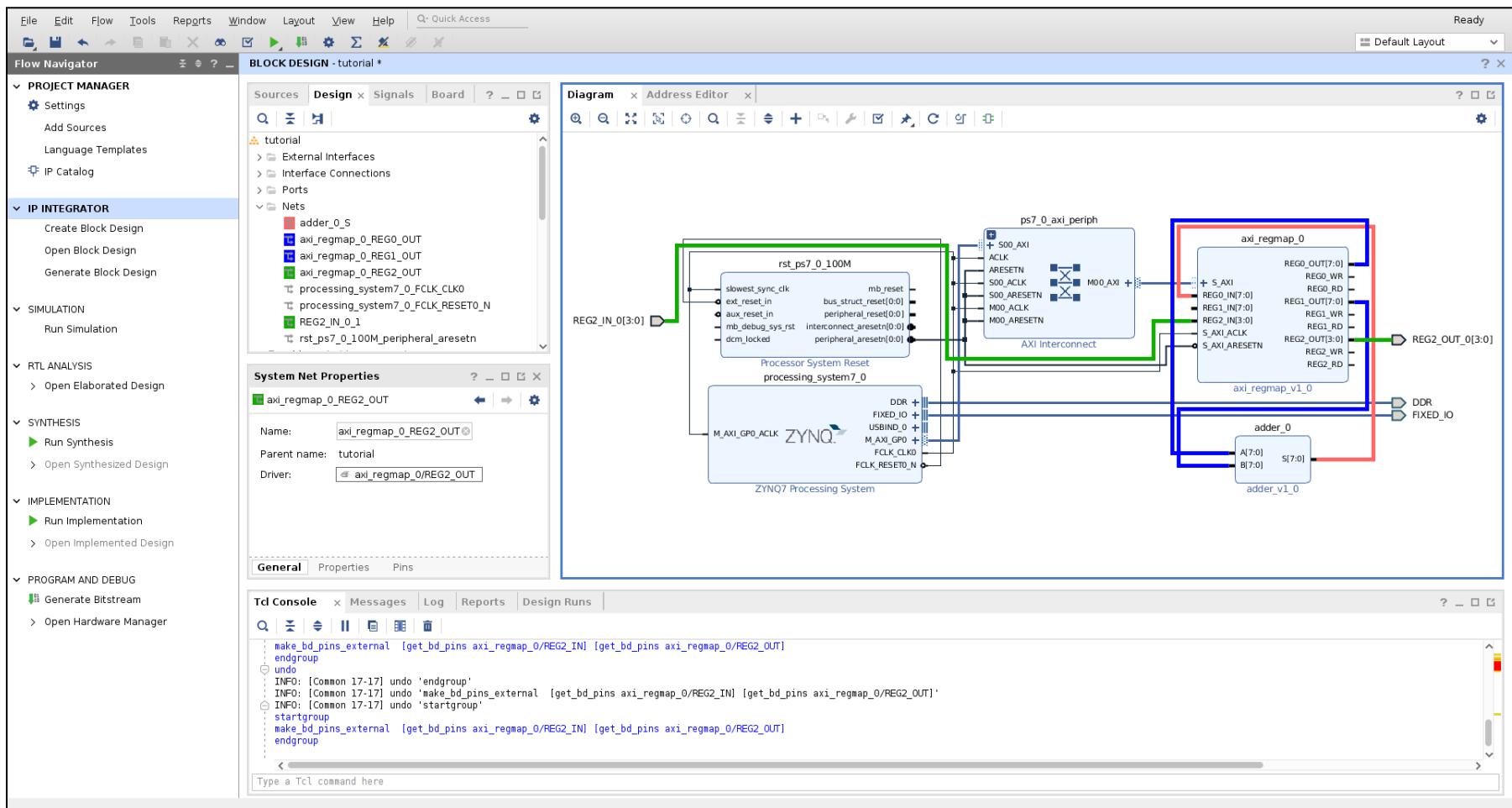
Make the following connections (Hold-click from one pin to another).
REG0_OUT → A (adder), REG1_OUT → B (adder), and S (adder) → REG0_IN



Control-click REG2_IN and REG2_OUT. Click *Make External*.

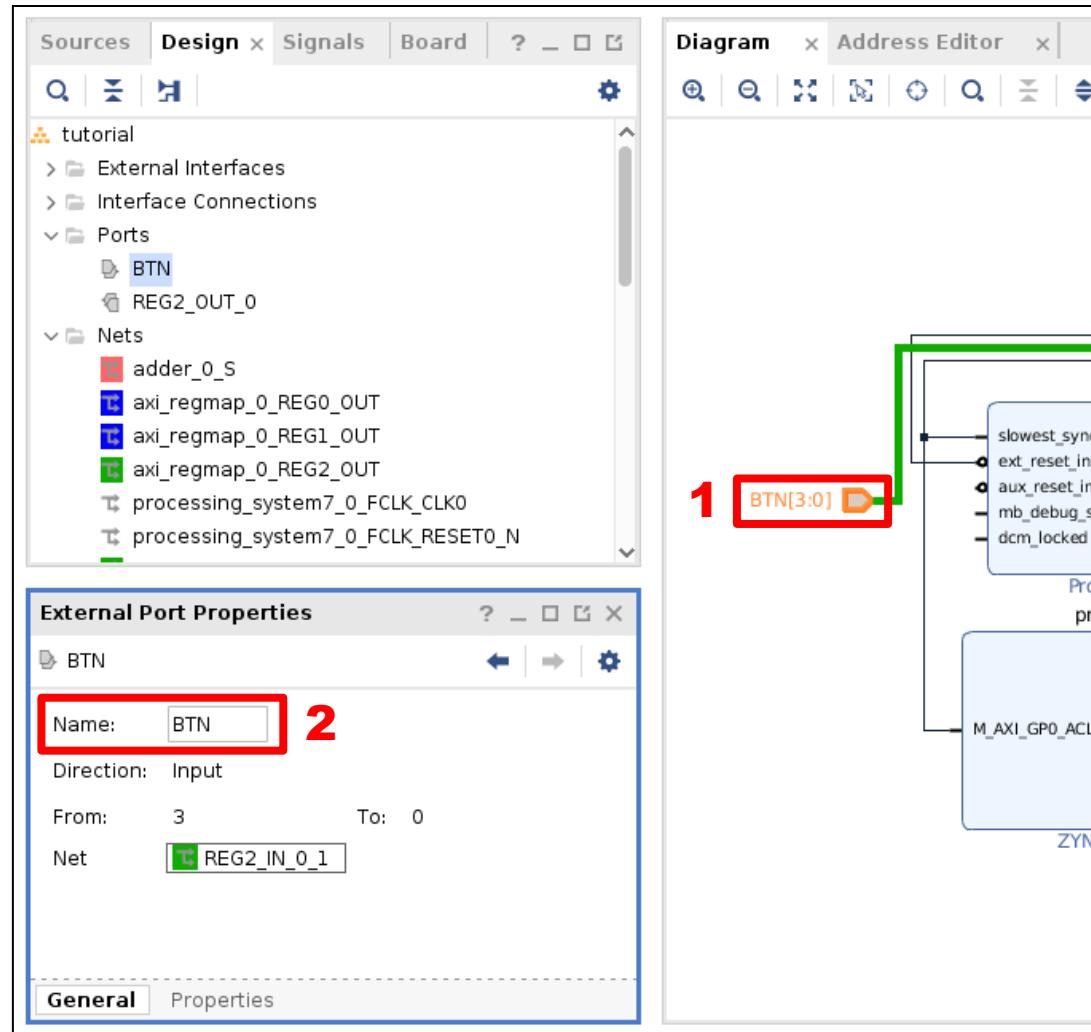


This is the current design. Continue.

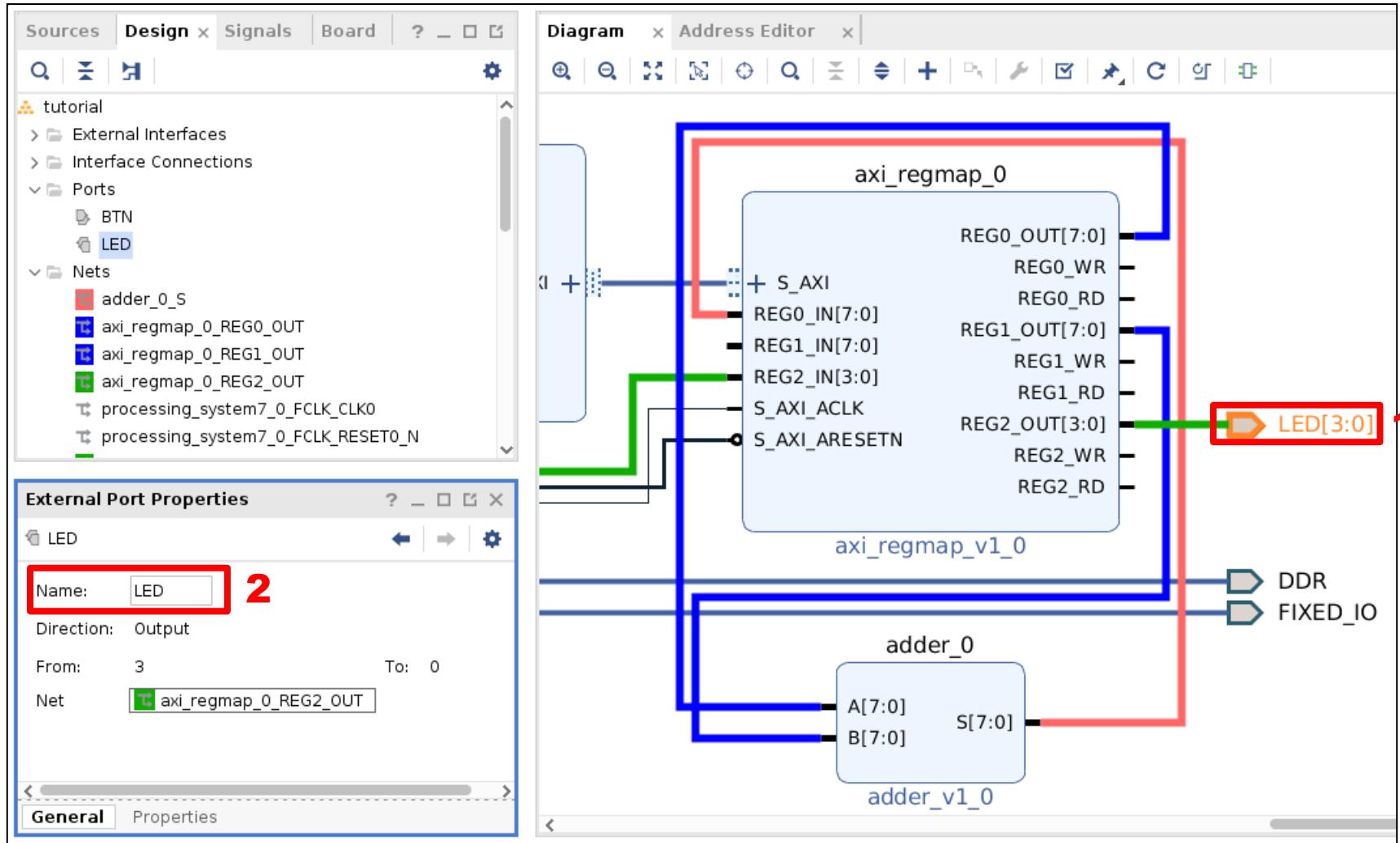


Click **REG2_IN_0[3:0]** (appears as **BTN[3:0]** in the fig). Rename external port to **btn**.

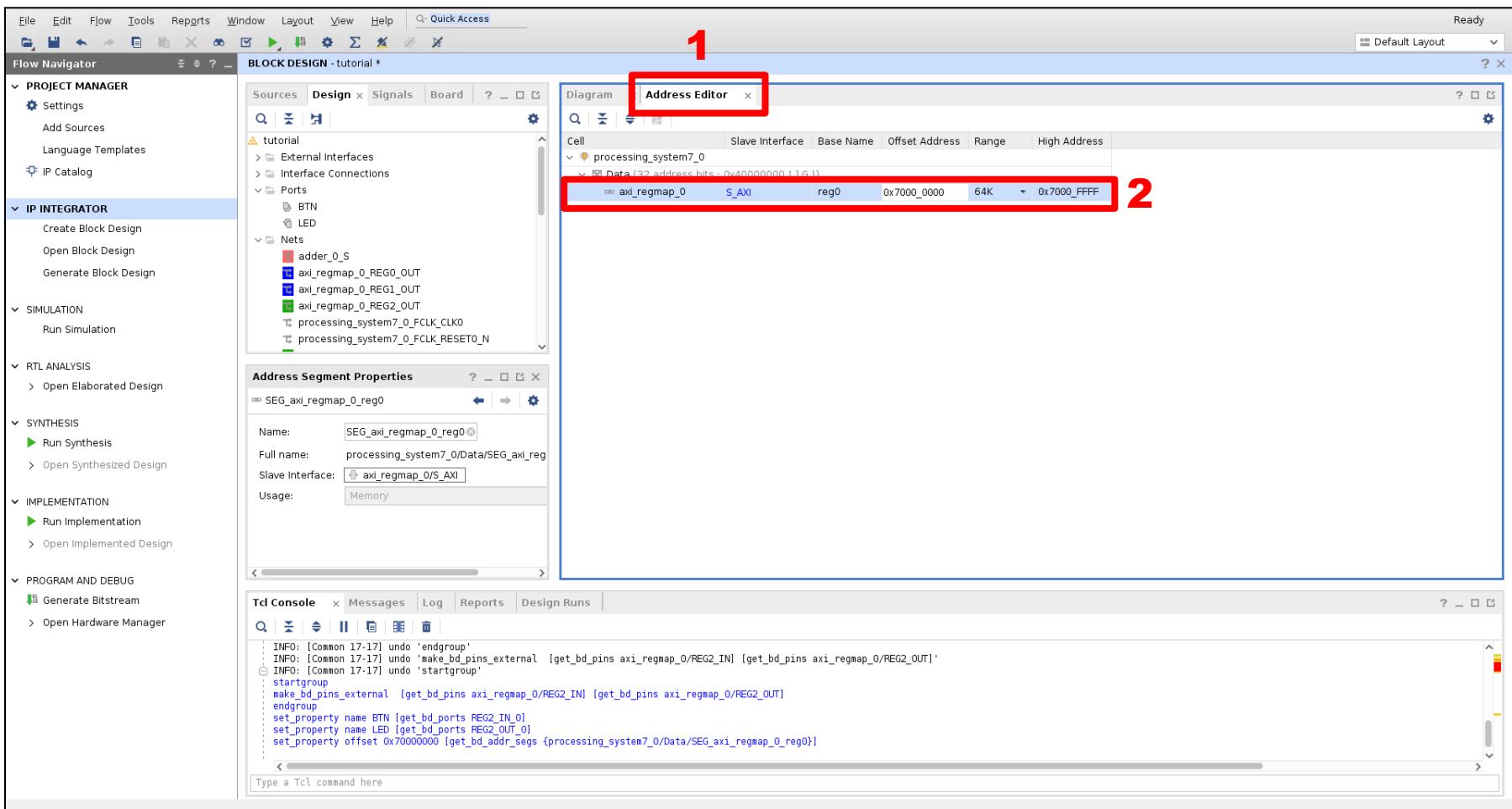
NOTE: Although VHDL is case insensitive, the XDC (constraints file) is. Make sure that the port name here matches the case in your XDC file.



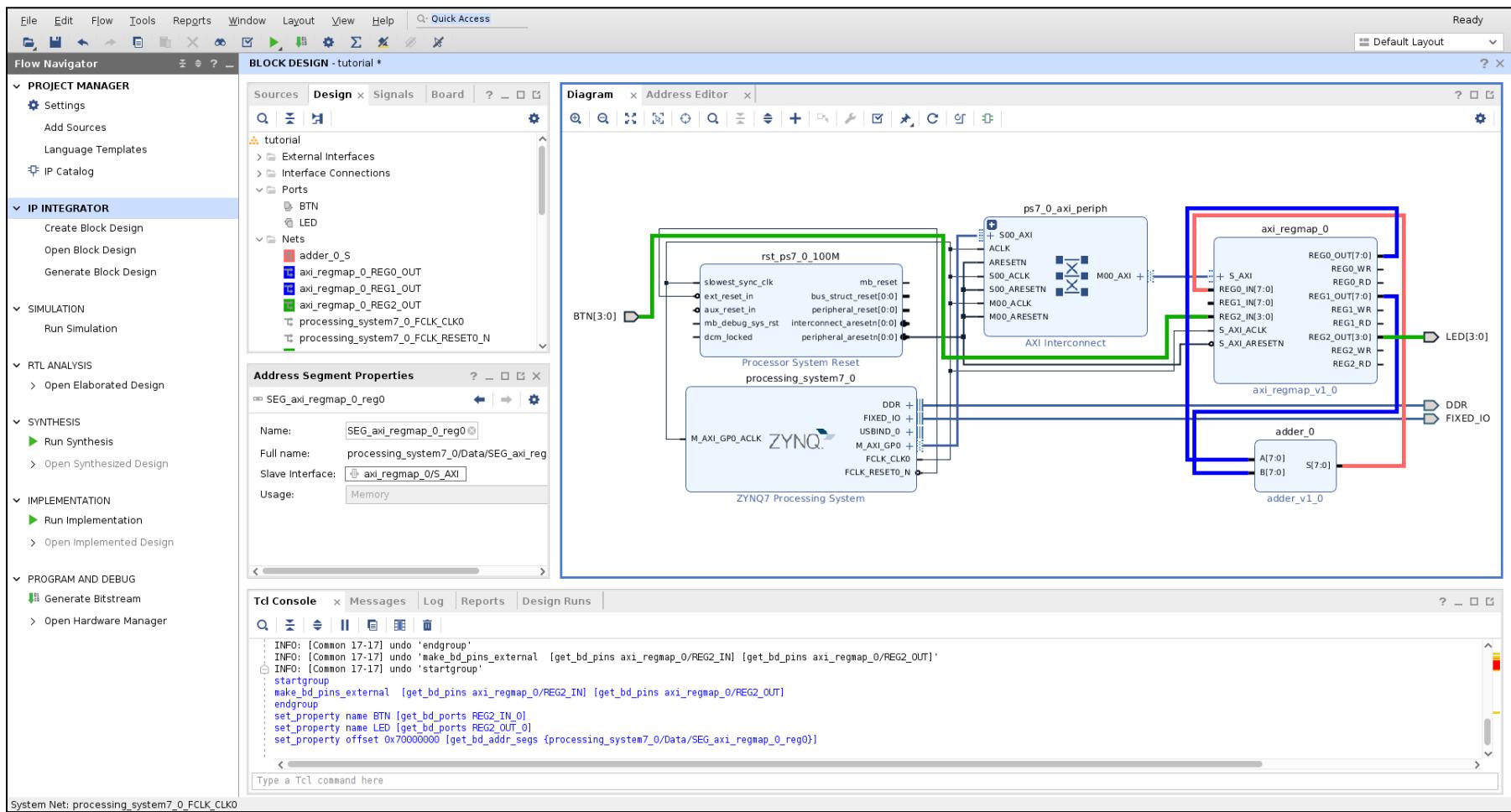
Click **REG2_OUT_0[3:0]** (appears as **LED[3:0]** in the pic). Rename external port to **led**.
NOTE: Although VHDL is case insensitive, the XDC (constraints file) is. Make sure that the port name here matches the case in your XDC file.



Click **Address Editor**. Enter offset address **0x7000_0000** for **axi_regmap_0**.



This is the current design. Continue.



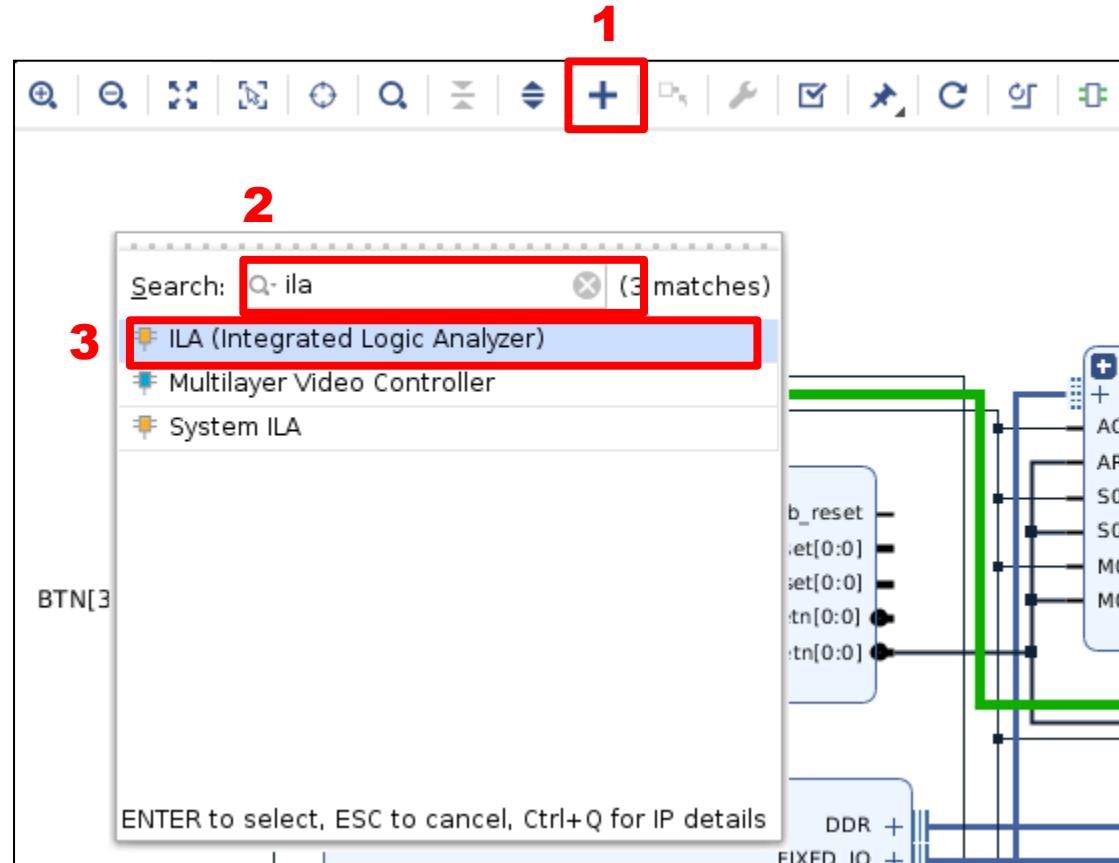
Part 5: Adding ILA

Next, we will add an Integrated Logic Analyzer (ILA) to probe the signals needed.

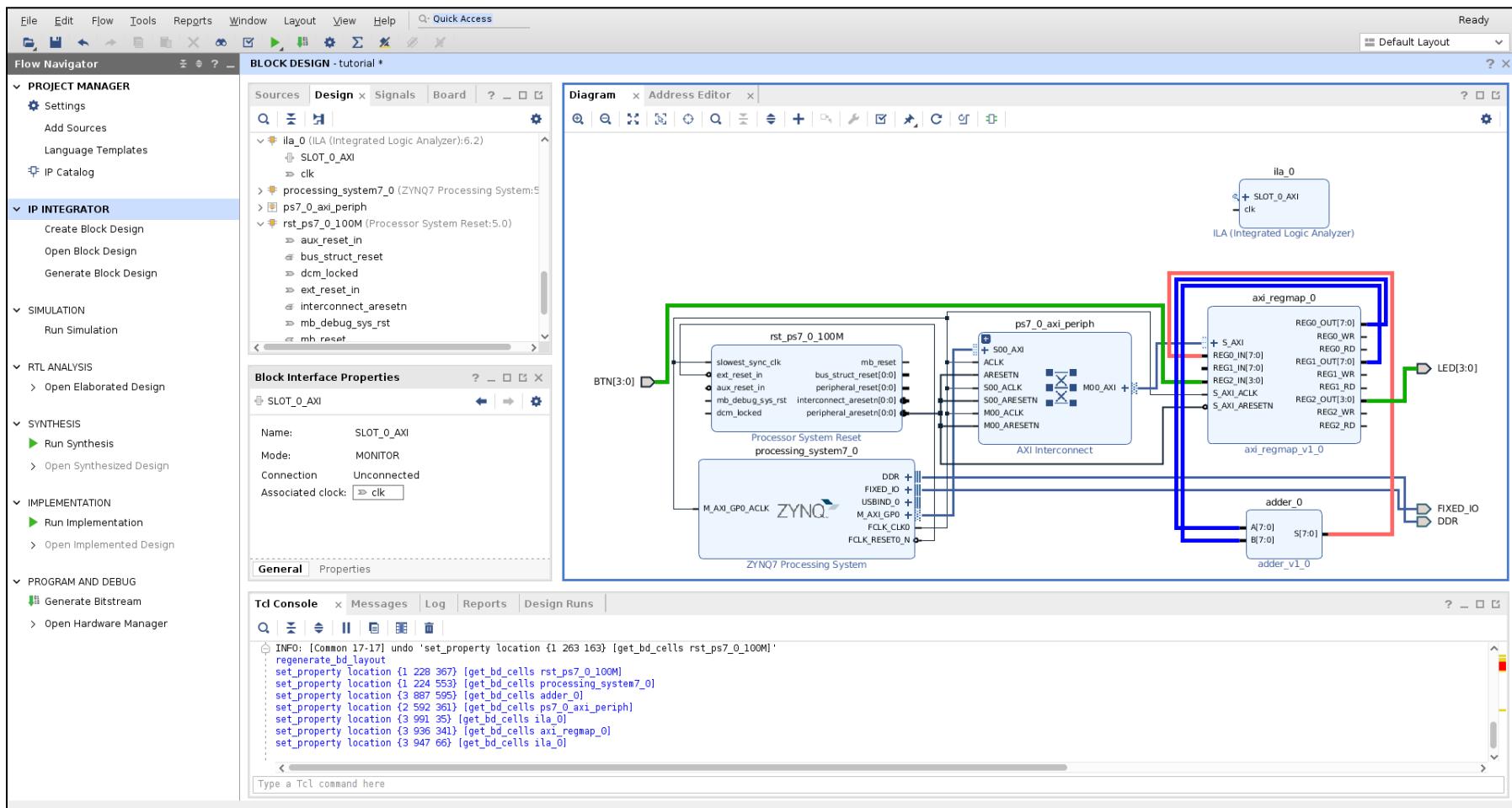


Click + (Add IP). Search *ila*. Double-click **ILA (Integrated Logic Analyzer)**.

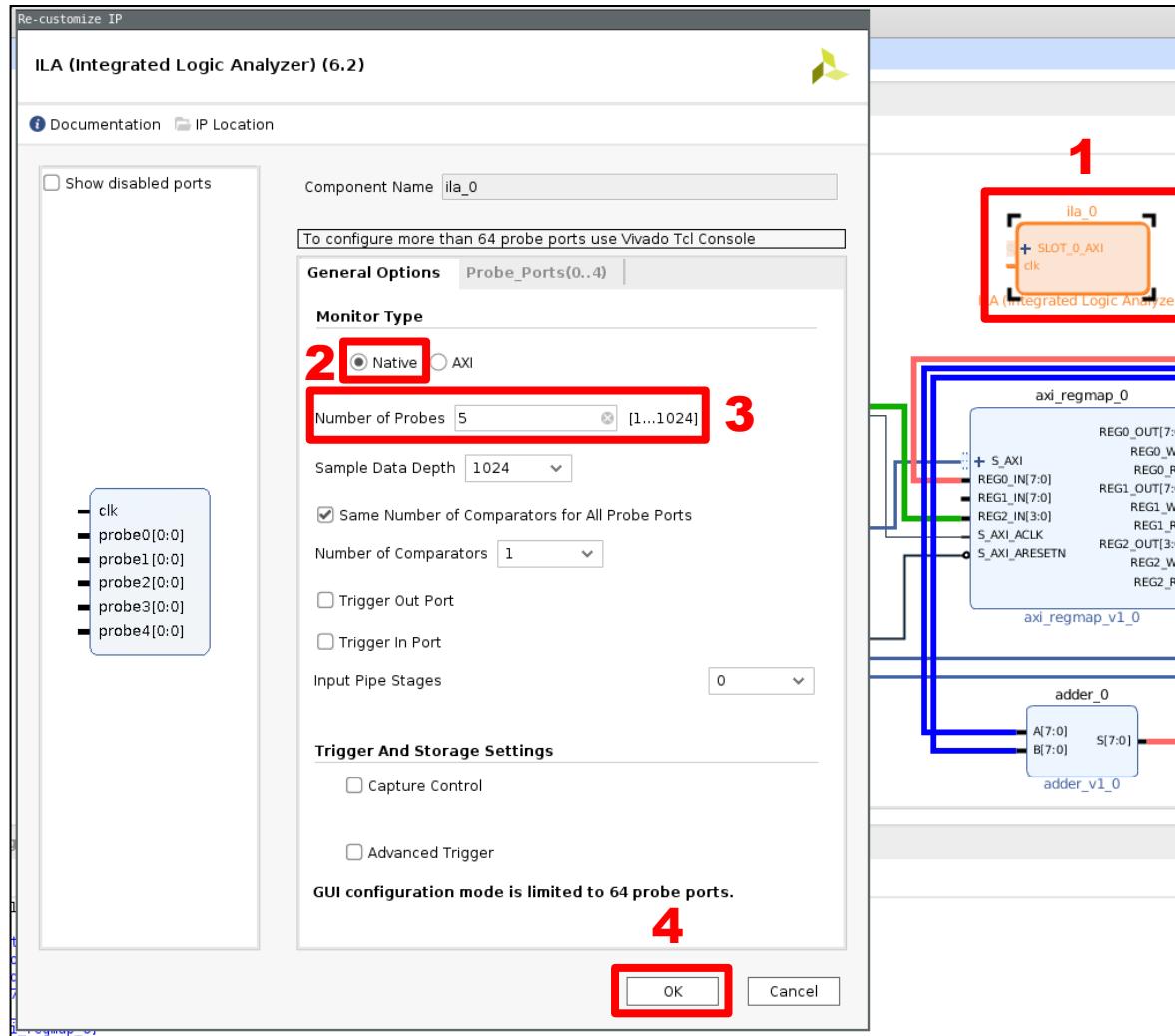
NOTE: The ILA is a useful debug tool for analyzing internal signals while the device and design are running.



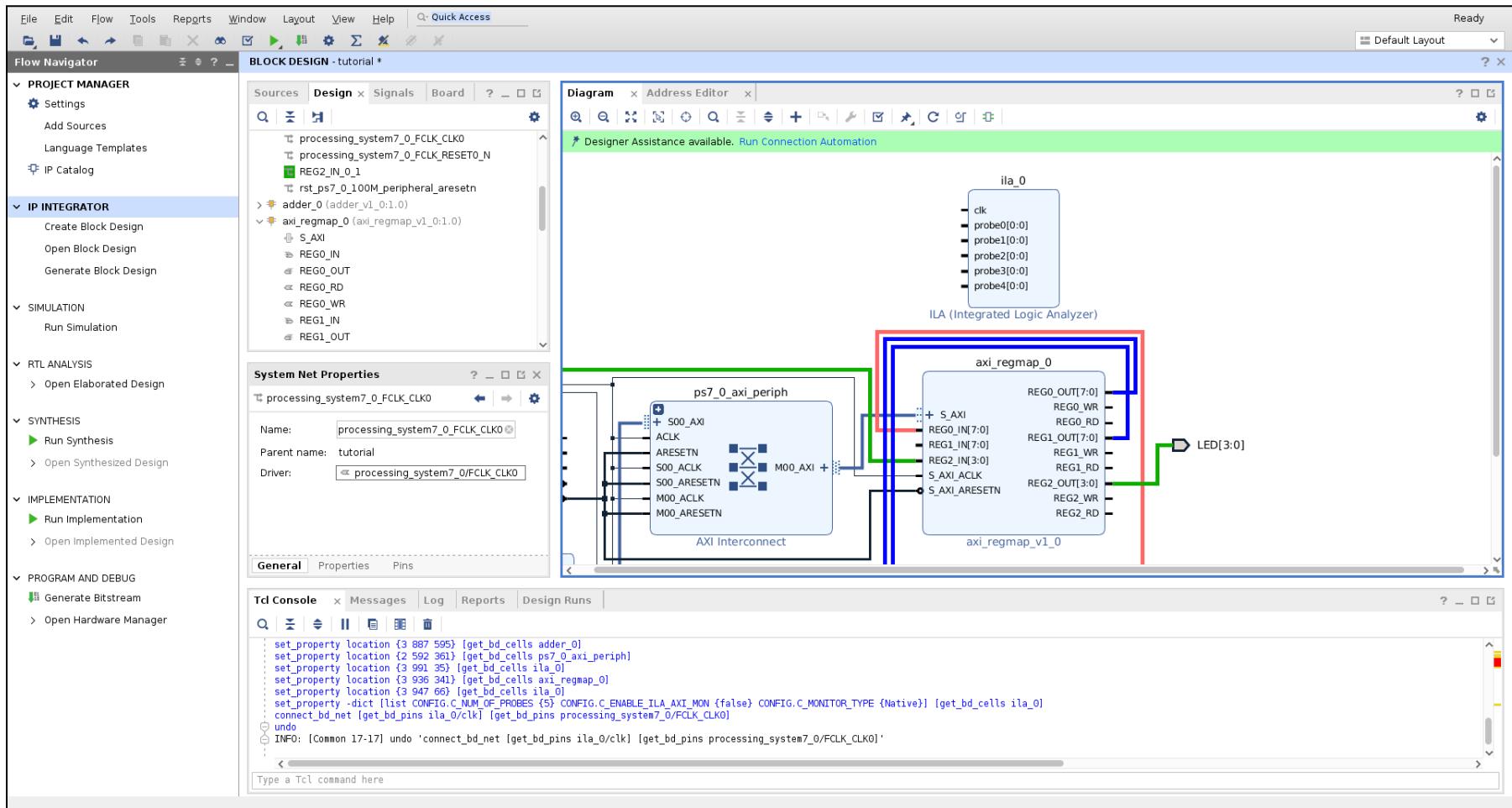
Wild **ILA** (*Integrated Logic Analyzer*) appeared! The instance name is **ila_0**. Continue.



Double-click **ila_0**. Set parameters **Native** and **Number of Probes**→5. Click **OK**.

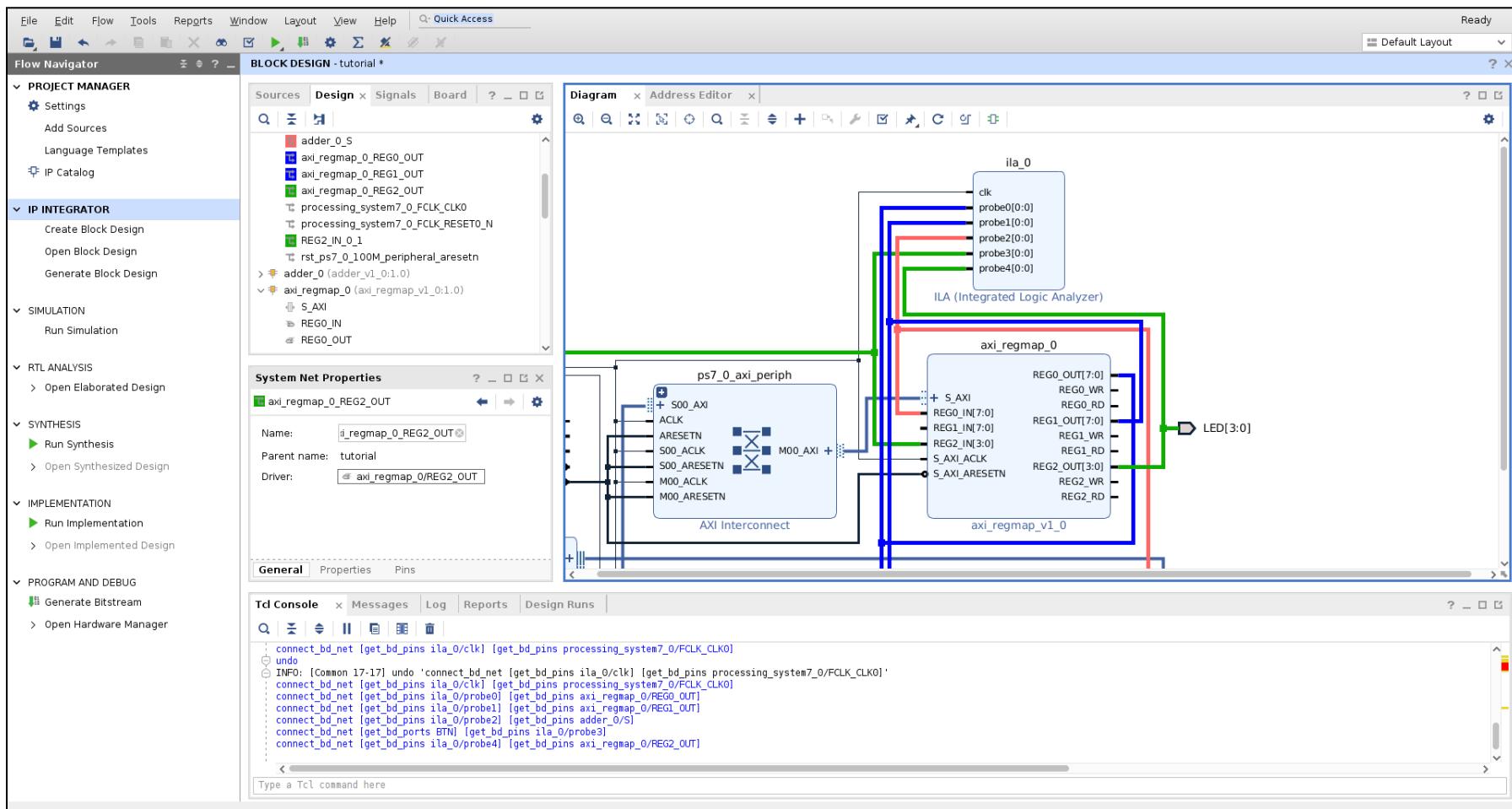


The ila_0 IP block has changed. Continue.



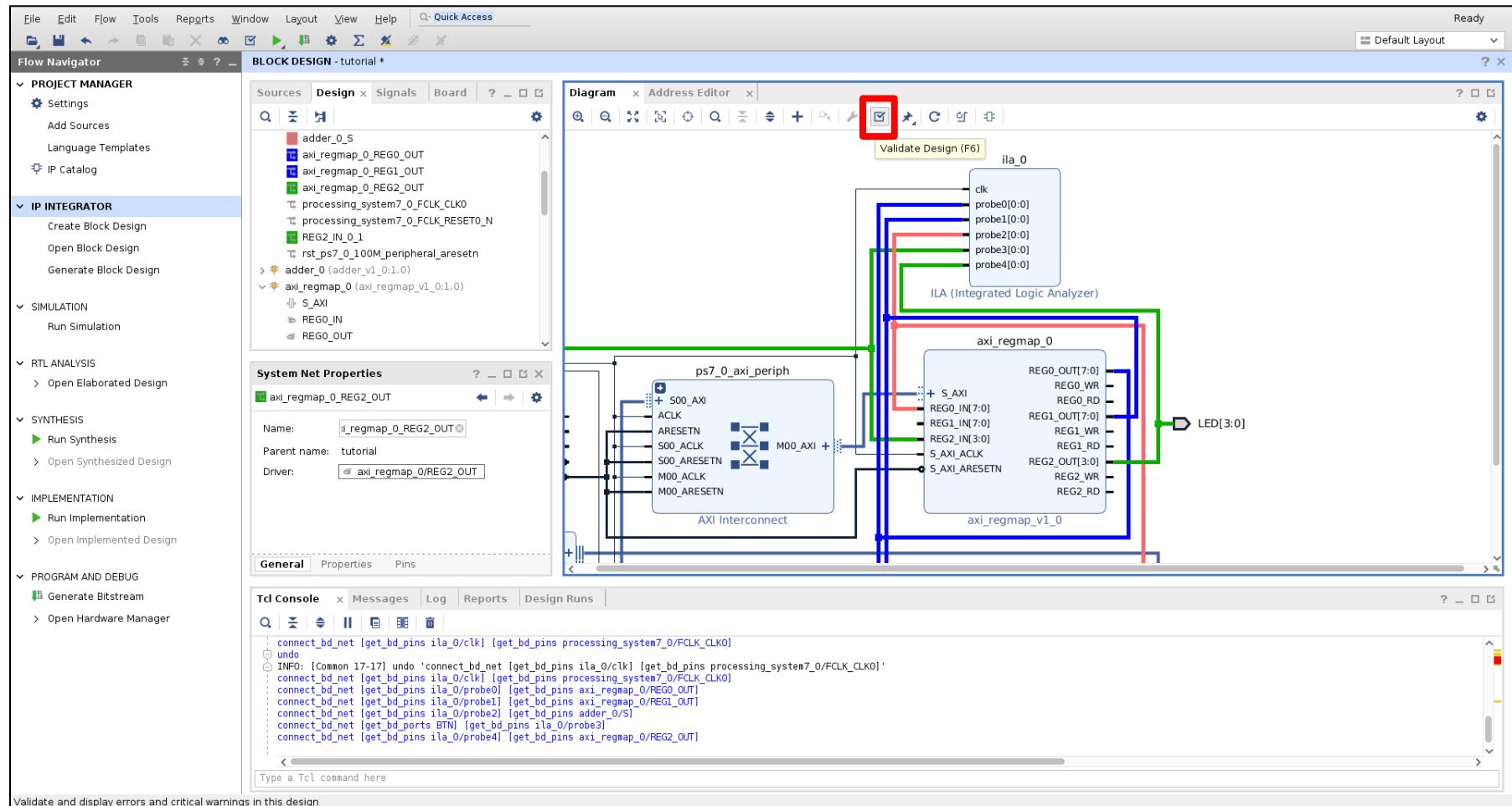
Connect the **ila_0 clk** port to the global clock signal (same as **S_AXI_ACLK** on **axi_regmap_0**). Connect the **ila_0** probes to the target nets.

NOTE: The order of probes does not matter; the probes are renamed to the connected net name.



Click **Validate Design**.

NOTE: This will report design errors and propagate parameters throughout the design.



Validate and display errors and critical warnings in this design

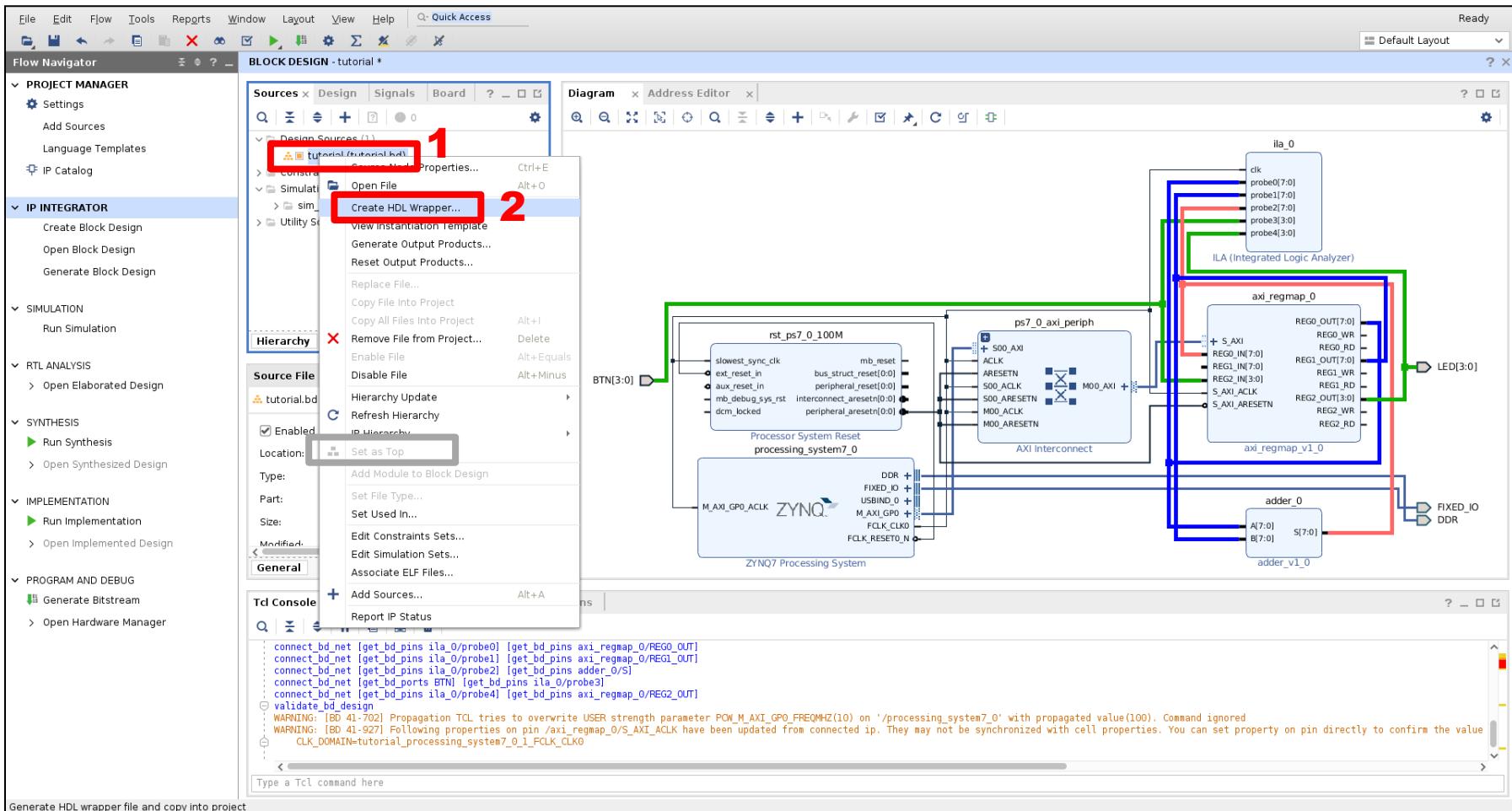
Click **OK**.



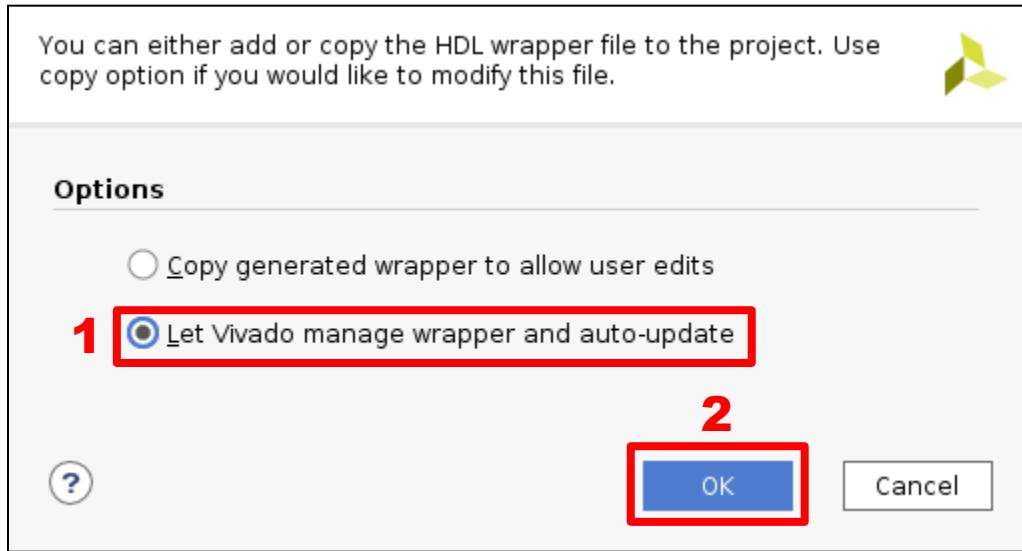
Right-click the block design **Lab_1**. Click **Create HDL Wrapper....**

NOTE: This process will generate a VHDL wrapper for the block design. You can think of this as the top design that we will be synthesizing and downloading on the FPGA.

NOTE: make sure the block design is set as Top (the grey box in the picture below).



Select ***Let Vivado manage wrapper and auto-update***. Click ***OK***.

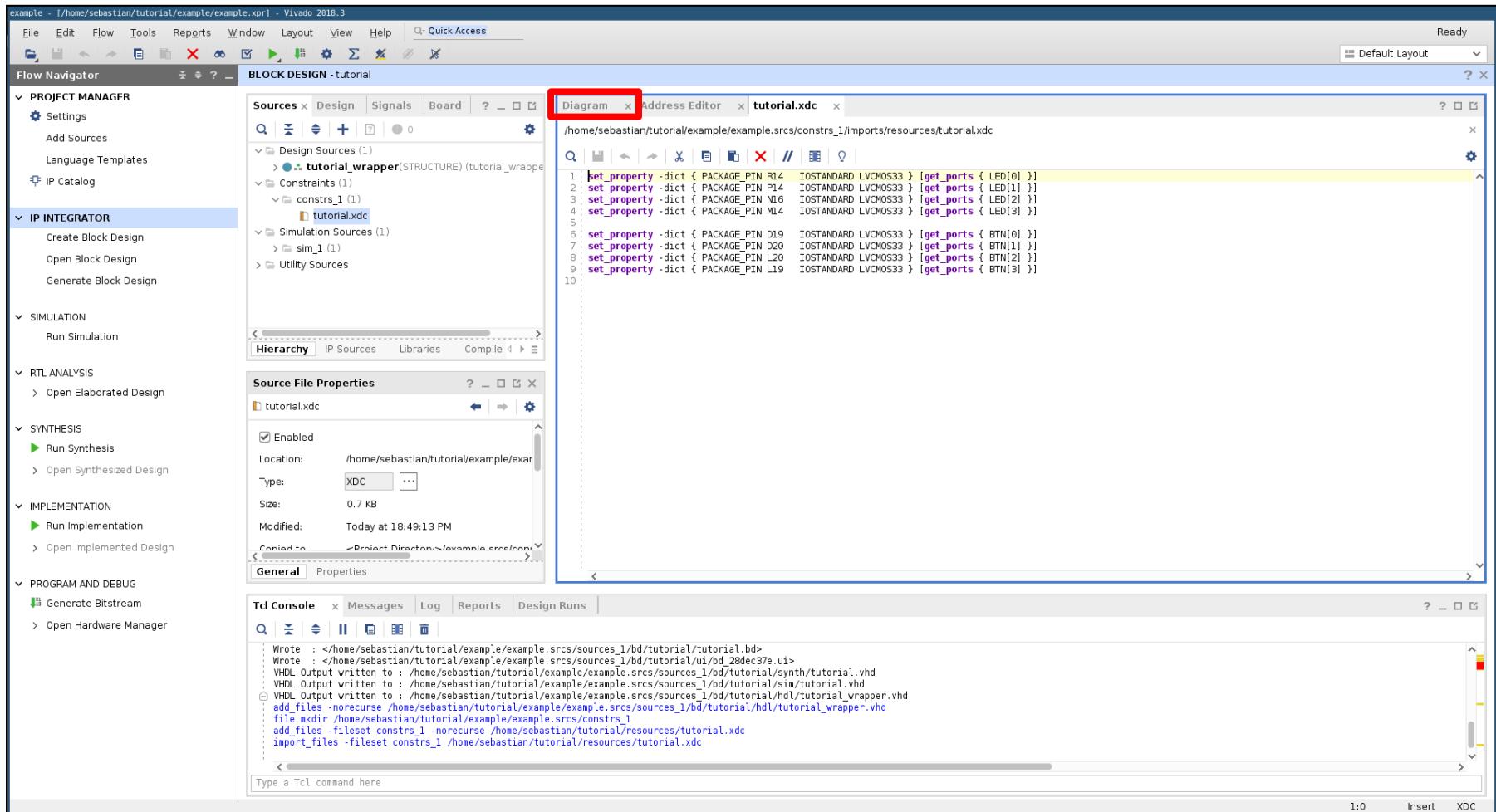


PYNQ-Z1_C.xdc. FPGA **package pins** and **IO standard (type and voltage)** are assigned to indexed bits of the **BTN and LED port vectors**. Continue.
Note: port names must match external ports of the block design.

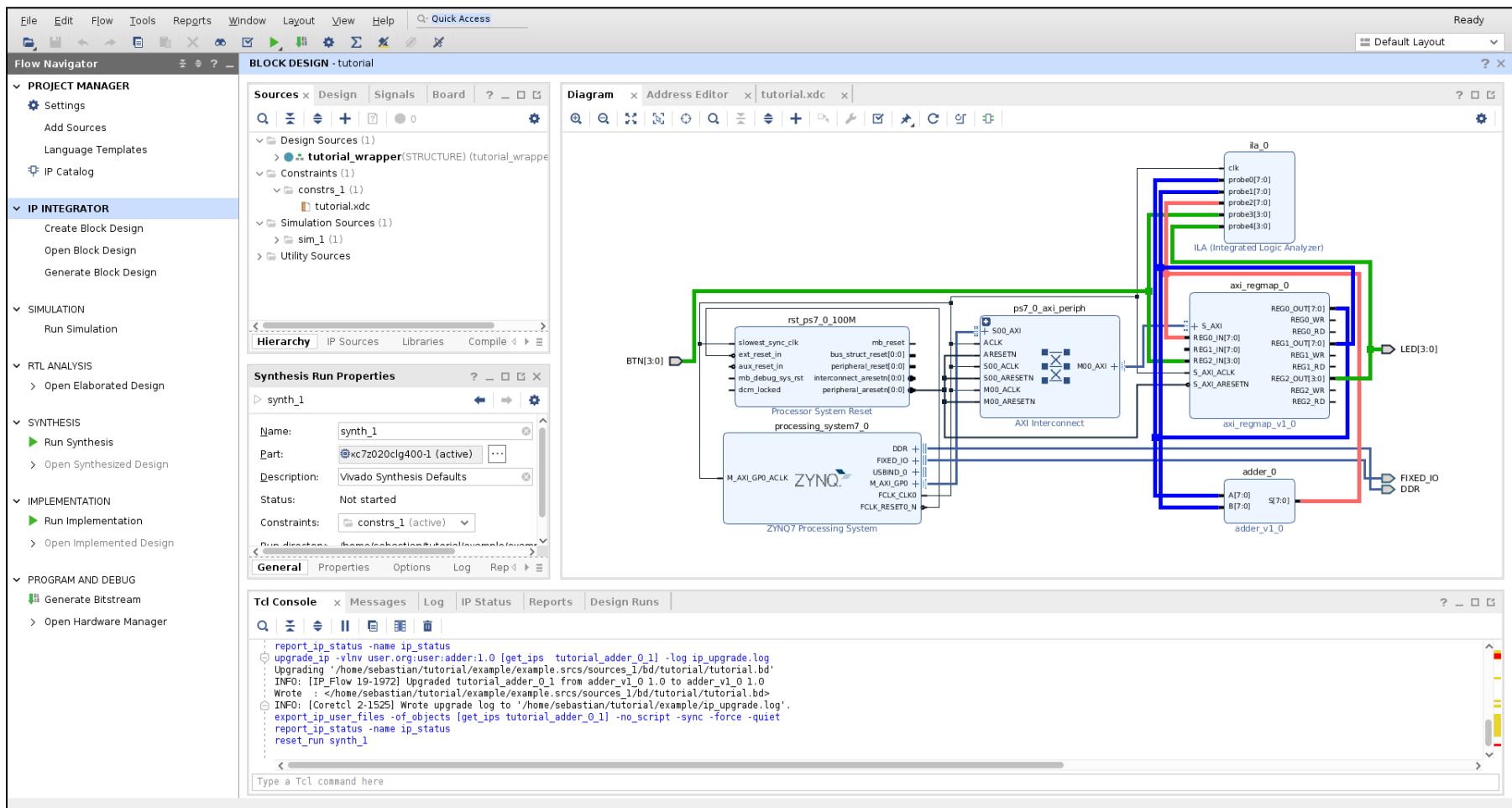
```
set_property -dict { PACKAGE_PIN R14    IOSTANDARD LVCMOS33 } [get_ports { LED[0] }]
set_property -dict { PACKAGE_PIN P14    IOSTANDARD LVCMOS33 } [get_ports { LED[1] }]
set_property -dict { PACKAGE_PIN N16    IOSTANDARD LVCMOS33 } [get_ports { LED[2] }]
set_property -dict { PACKAGE_PIN M14    IOSTANDARD LVCMOS33 } [get_ports { LED[3] }]

set_property -dict { PACKAGE_PIN D19    IOSTANDARD LVCMOS33 } [get_ports { BTN[0] }]
set_property -dict { PACKAGE_PIN D20    IOSTANDARD LVCMOS33 } [get_ports { BTN[1] }]
set_property -dict { PACKAGE_PIN L20    IOSTANDARD LVCMOS33 } [get_ports { BTN[2] }]
set_property -dict { PACKAGE_PIN L19    IOSTANDARD LVCMOS33 } [get_ports { BTN[3] }]
```

Click the **Diagram** tab.



This is the current design. Continue.

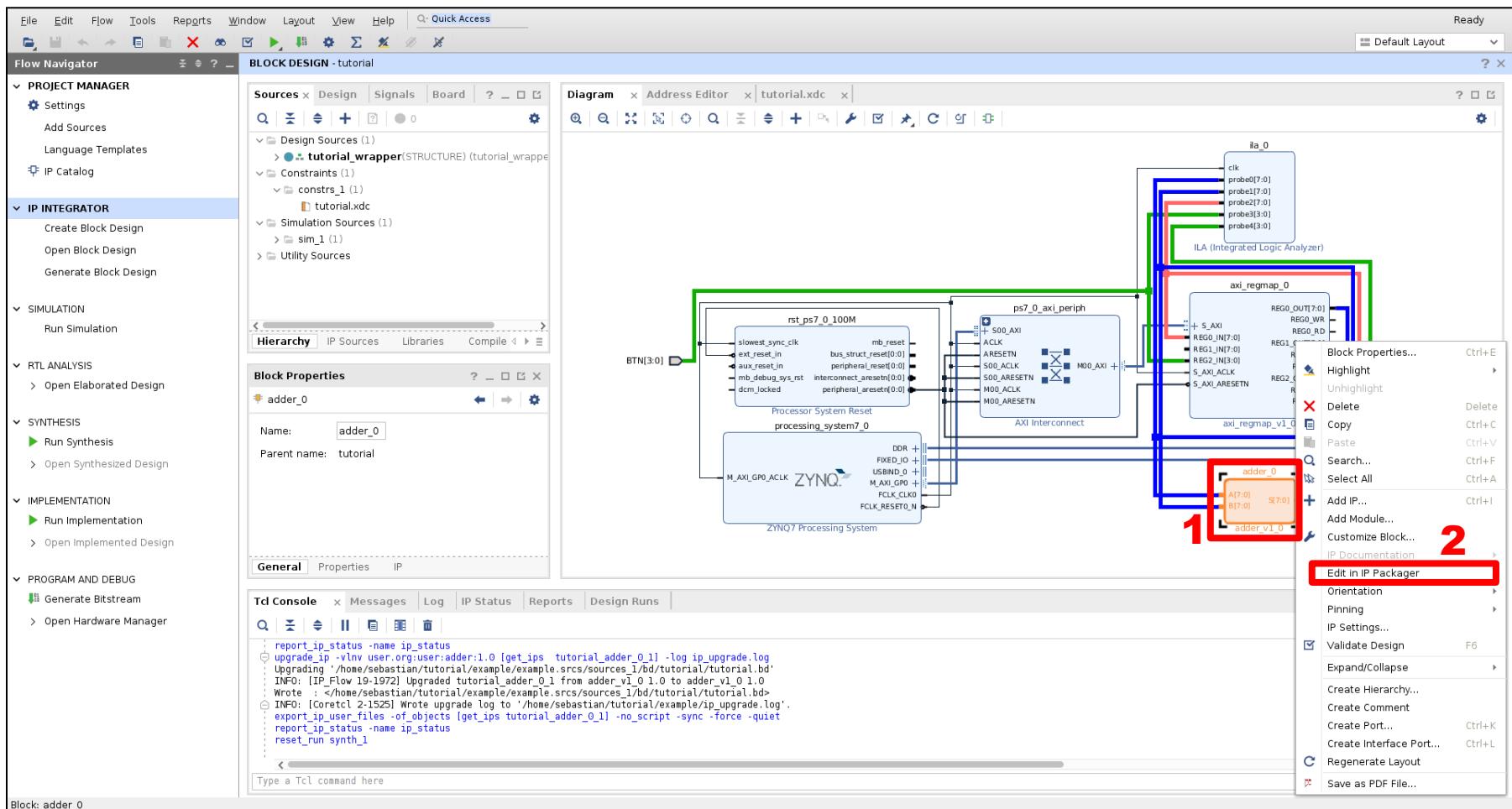


Part 5: Editing an IP

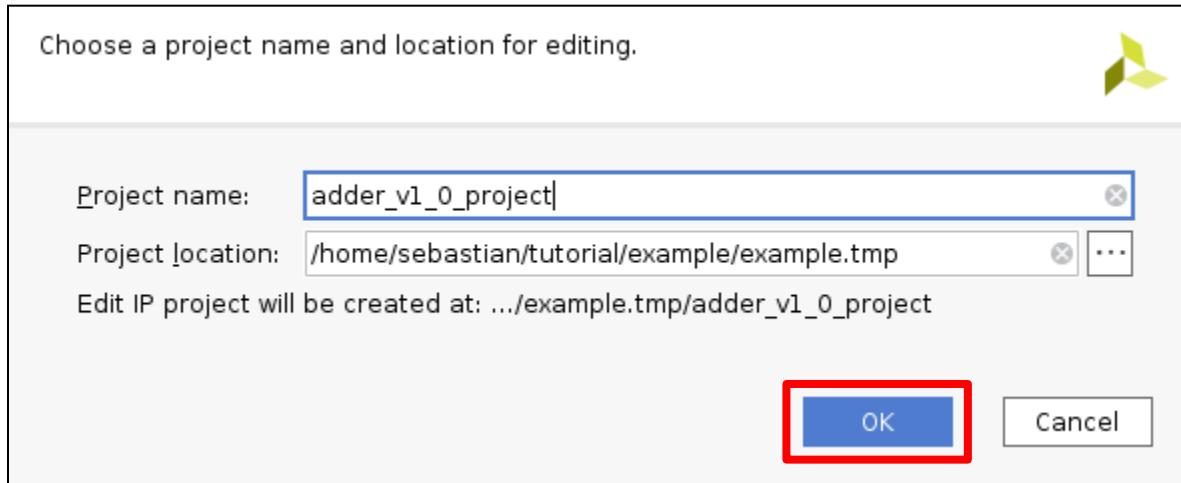
If the RTL (e.g., VHDL) of an IP must be modified, the Vivado IP packager must be used to re-package the IP.
Let us suppose that the **adder** IP must be edited.



Right-click the **adder_0** IP block. Click **Edit in IP Packager**.



Click **OK**.



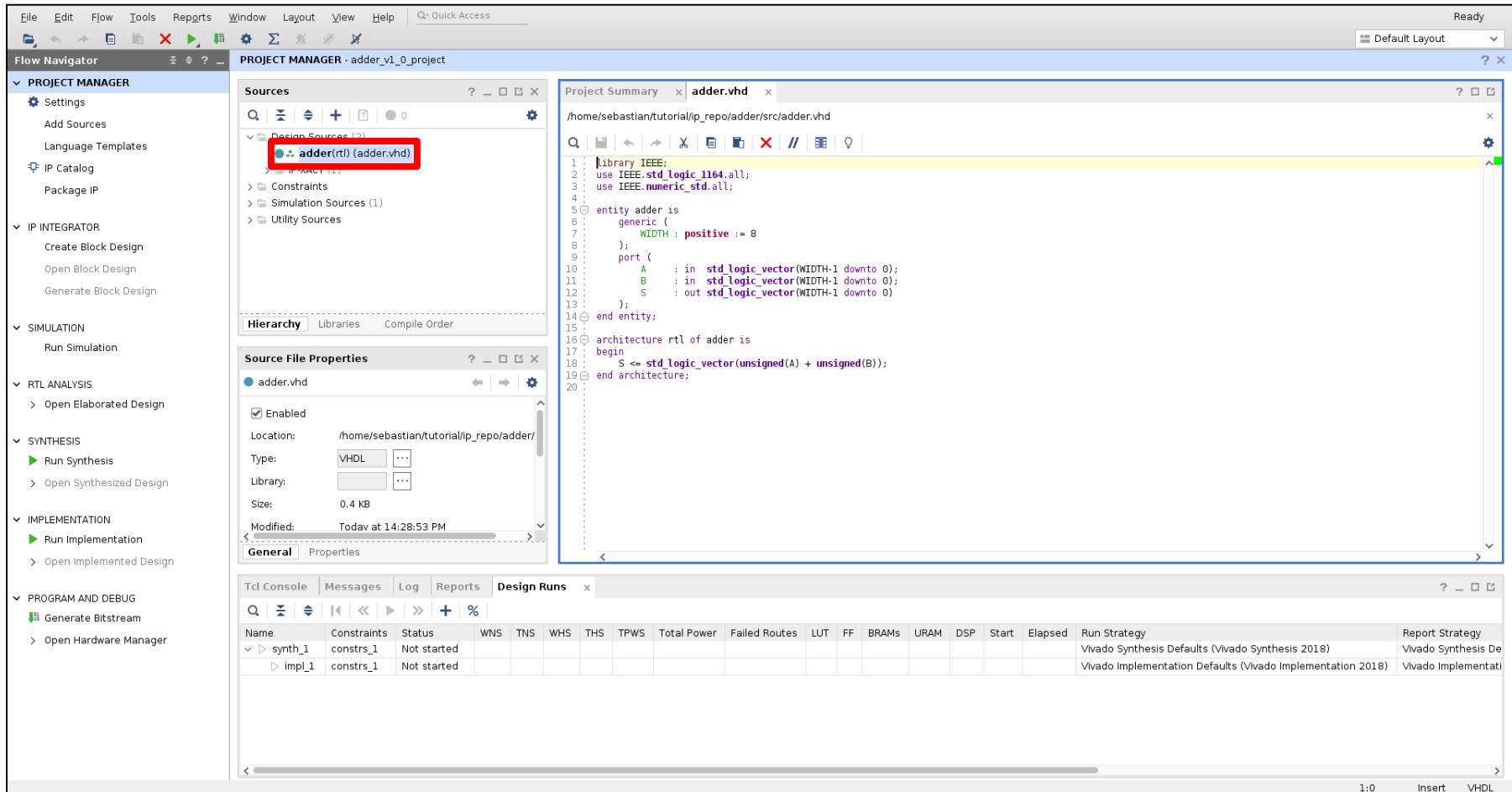
A new Vivado instance is created specifically for the **adder** IP. Use this instance to develop and package the **adder** IP. Continue.

The screenshot shows the Vivado Project Manager interface with the following details:

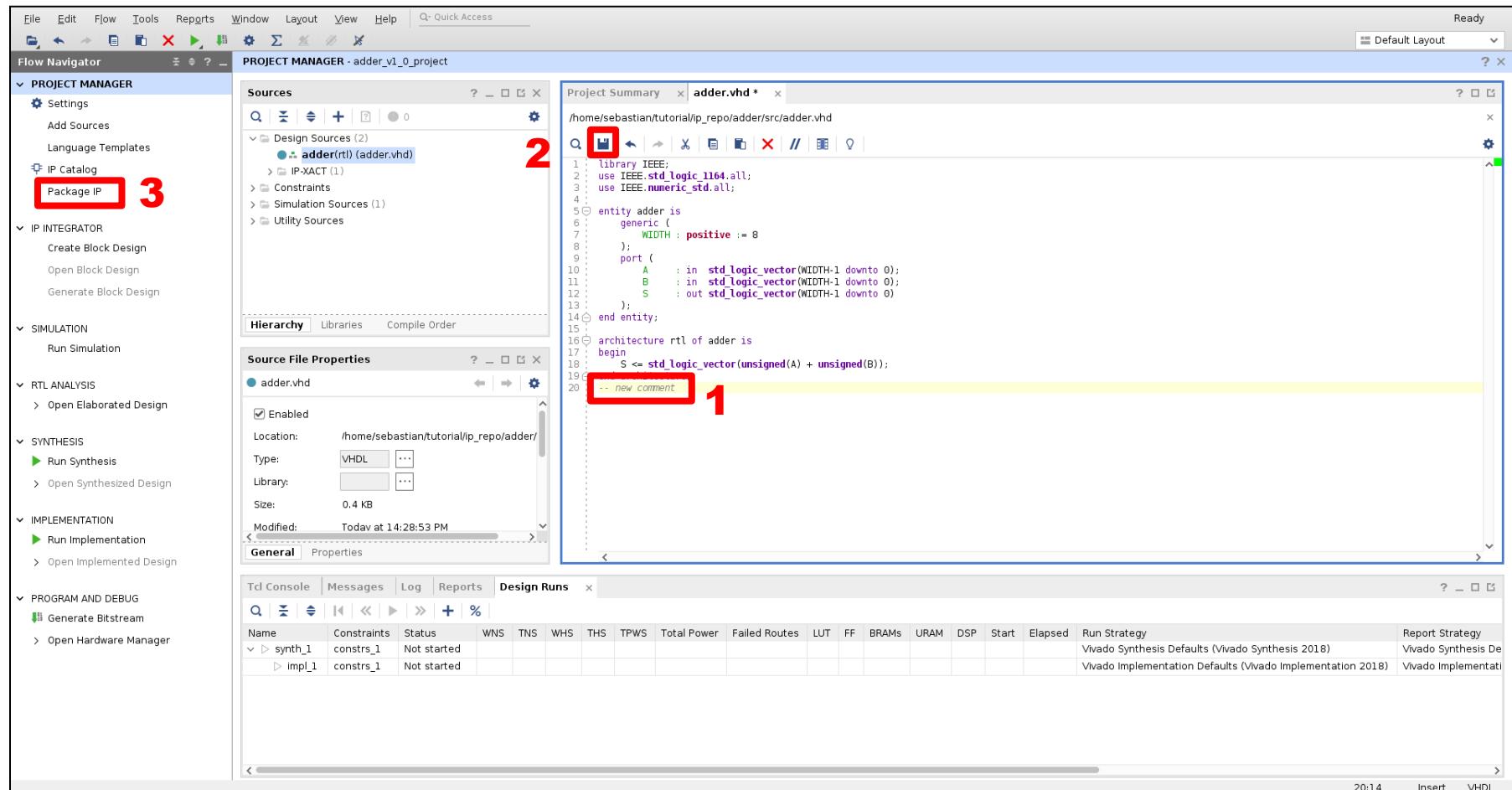
- Project Summary:**
 - Project name: adder_v1_0_project
 - Project location: /home/sebastian/tutorial/example/example.tmp/adder_v1_0_project
 - Product family: Zynq-7000
 - Project part: PYNQ-Z1 (xc7z020clg400-1)
 - Top module name: adder
 - Target language: VHDL
 - Simulator language: Mixed
- Synthesis:**
 - Status: Not started
 - Messages: No errors or warnings
 - Part: xc7z020clg400-1
 - Strategy: Vivado Synthesis Defaults
 - Report Strategy: Vivado Synthesis Default Reports
- Implementation:**
 - Status: Not started
 - Messages: No errors or warnings
 - Part: xc7z020clg400-1
 - Strategy: Vivado Implementation Defaults
 - Report Strategy: Vivado Implementation Default Reports
 - Incremental implementation: None
- DRC Violations:** Run Implementation to see DRC results
- Timing:** Run Implementation to see timing results
- Utilization:**
- Power:**
- Design Runs:**

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy	Report Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2018)	Vivado Synthesis Defaults (Vivado Synthesis 2018)
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2018)	Vivado Implementation Defaults (Vivado Implementation 2018)

Double-click adder.vhd to open it in the editor.

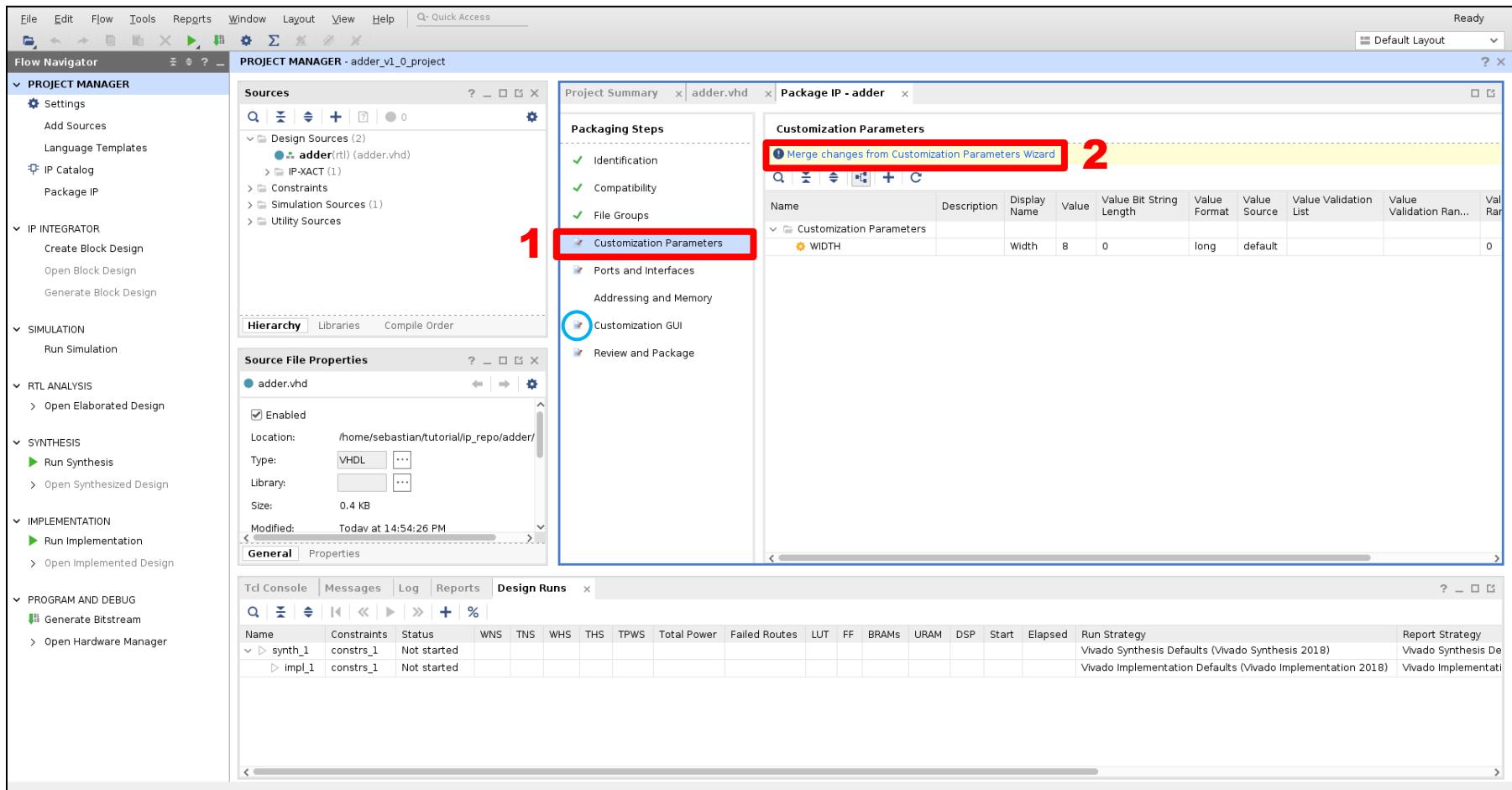


Add a commented line. Click **Save**. Click **Package IP**.

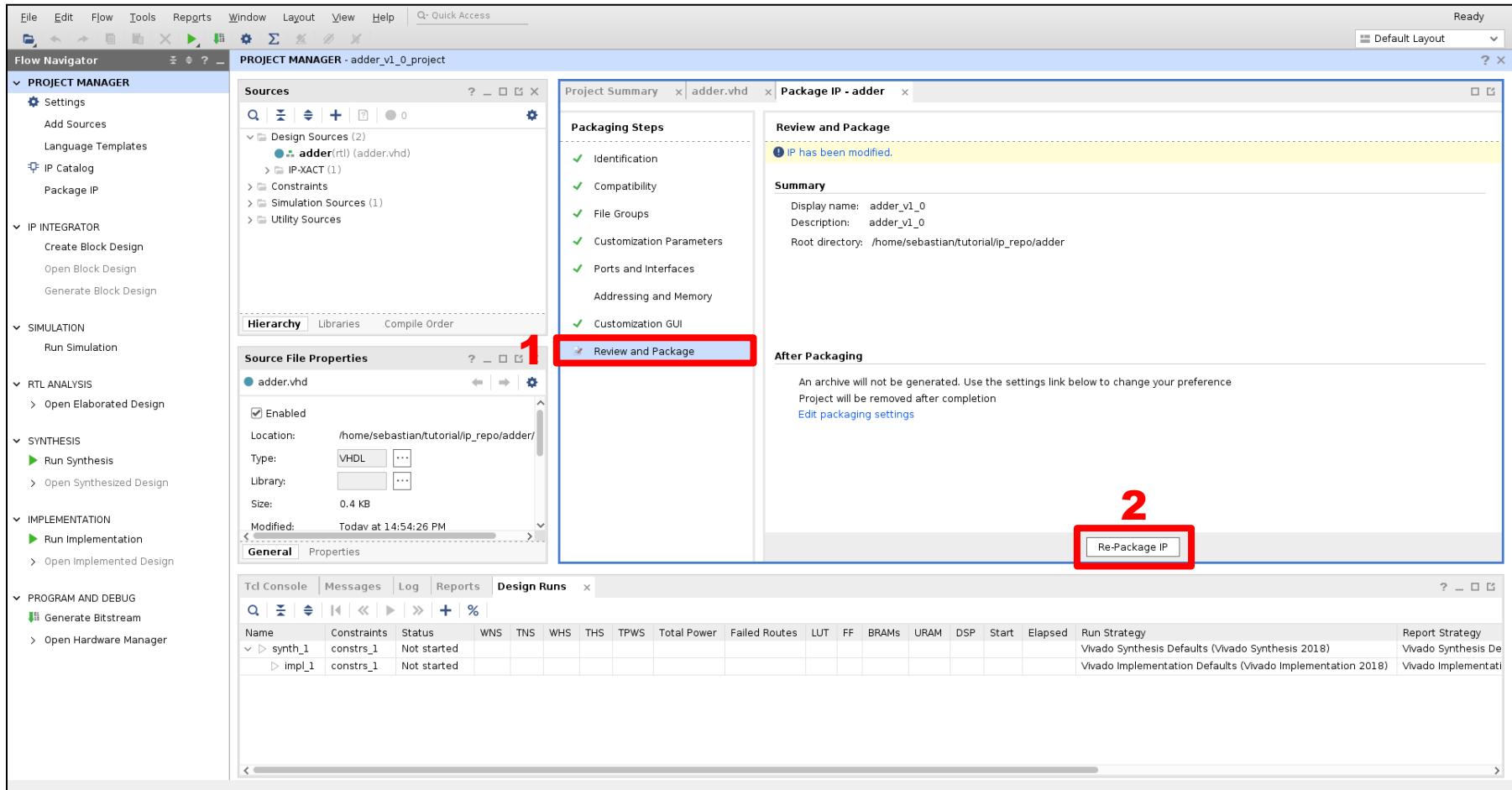


Click on any tabs containing the modified **symbol**. Click the yellow-highlighted link to merge changes.

NOTE: Often, clicking one link will merge all changes.

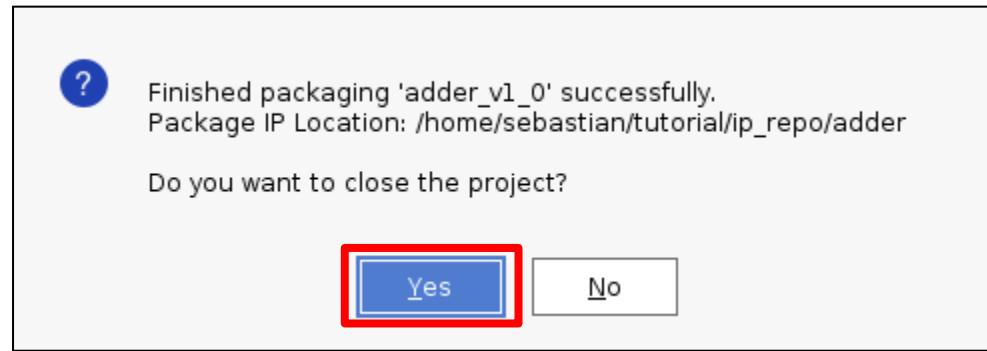


Click **Review and Package**. Click **Re-Package IP**.

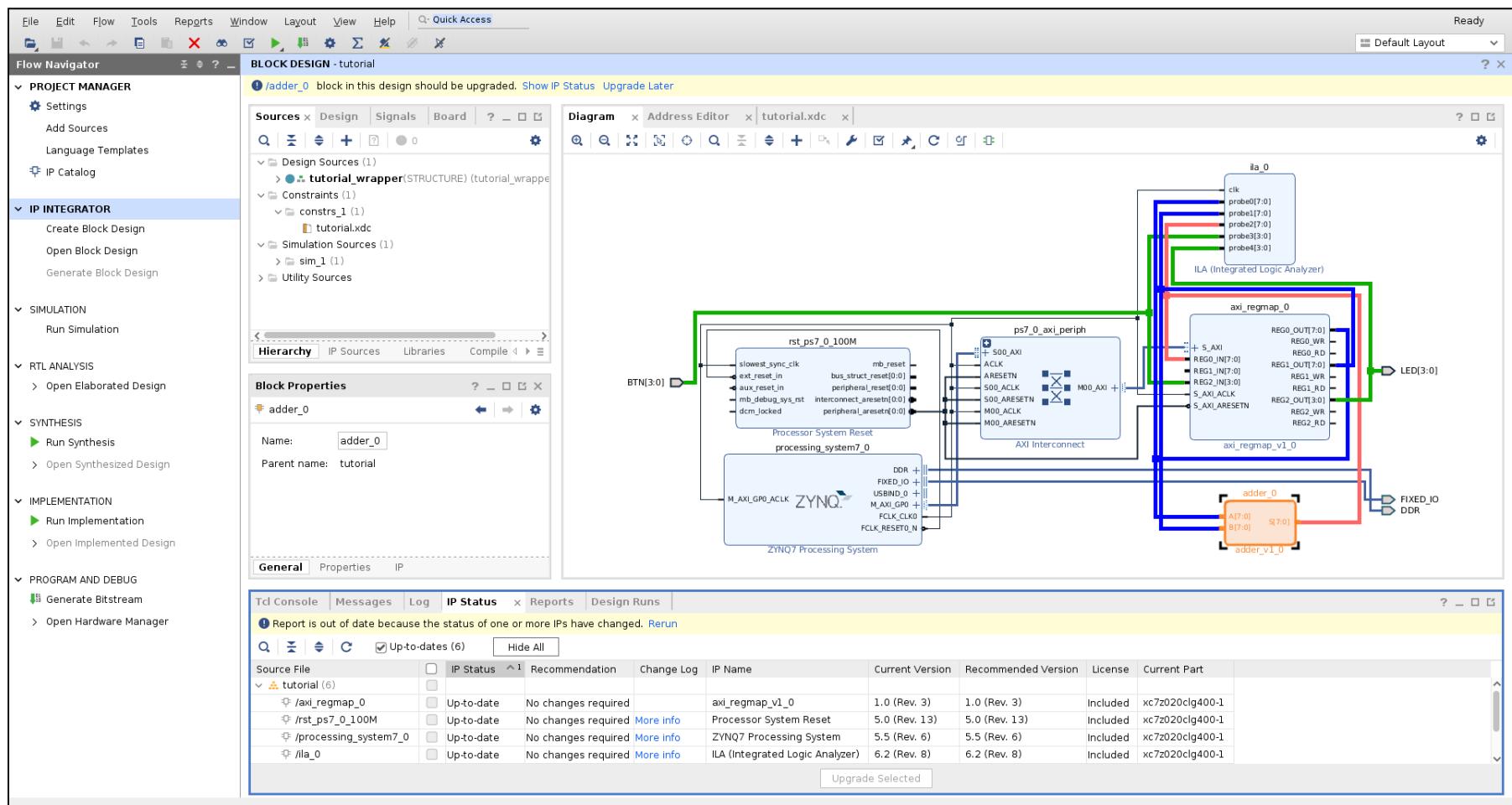


Click Yes.

NOTE: You can optionally keep this Vivado instance open to continue developing your IP.



We return to the main Vivado instance. Continue.



Click on the **adder_0** instance, then **Show IP Status**. Click **Rerun**.
Note: if you didn't see the **Rerun** option, continue to the next slide.

Block Design - tutorial

1 /adder_0 block in this design should be upgraded Show IP Status Upgrade Later

Sources x Design x Signals x Board x ? - Diagram x Address Editor x tutorial.xdc x

Design Sources (1)

- > tutorial_wrapper(STRUCTURE) (tutorial_wrapper)

Constraints (1)

- constrs_1 (1)
 - tutorial.xdc

Simulation Sources (1)

- > sim_1 (1)

Utility Sources

Hierarchy IP Sources Libraries Compile < >

Block Properties

adder_0

Name: adder_0

Parent name: tutorial

General Properties IP

Diagram

Block Diagram showing the Zynq SoC architecture. It includes the ZYNQ Processing System, AXI Interconnect, ps7_0_axi_periph, axi_remap_0, and ila_0. The adder_0 instance is highlighted with a red box. A green box highlights the 'adder_0' instance in the Block Properties panel.

2

Tcl Console Messages Log IP Status x Reports Design Runs

Report is out of date because the status of one or more IPs have changed Rerun

Up-to-dates (6) Hide All

Source File	IP Status	Recommendation	Change Log	IP Name	Current Version	Recommended Version	License	Current Part
tutorial (6)								
axi_remap_0	Up-to-date	No changes required		axi_remap_v1_0	1.0 (Rev. 3)	1.0 (Rev. 3)	Included	xc7z020clg400-1
rst_ps7_0_100M	Up-to-date	No changes required	More Info	Processor System Reset	5.0 (Rev. 13)	5.0 (Rev. 13)	Included	xc7z020clg400-1
processing_system7_0	Up-to-date	No changes required	More Info	ZYNQ7 Processing System	5.5 (Rev. 6)	5.5 (Rev. 6)	Included	xc7z020clg400-1
ila_0	Up-to-date	No changes required	More Info	ILA (Integrated Logic Analyzer)	6.2 (Rev. 8)	6.2 (Rev. 8)	Included	xc7z020clg400-1

Upgrade Selected

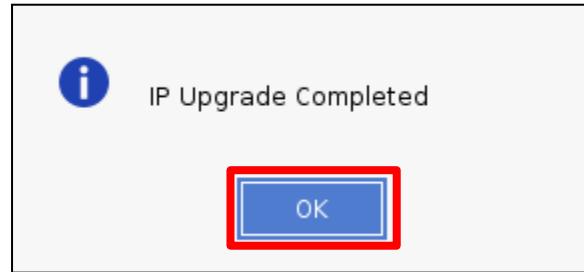
Click *Upgrade Selected*.

NOTE: All upgradeable IP are listed. All instances of a modified IP are upgraded.

The screenshot displays the Vivado Design Suite interface for a Zynq SoC design. The top section shows the 'BLOCK DESIGN - tutorial' window, which includes a diagram of the system architecture. The architecture consists of a ZYNQ Processing System connected to various peripherals like DDR, USB, and AXI components. An 'adder_0' IP instance is highlighted in orange. The bottom section shows the 'IP Status' window, which lists all IPs in the design. The 'adder_0' IP is selected, and a red box surrounds the 'Upgrade Selected' button at the bottom of the table.

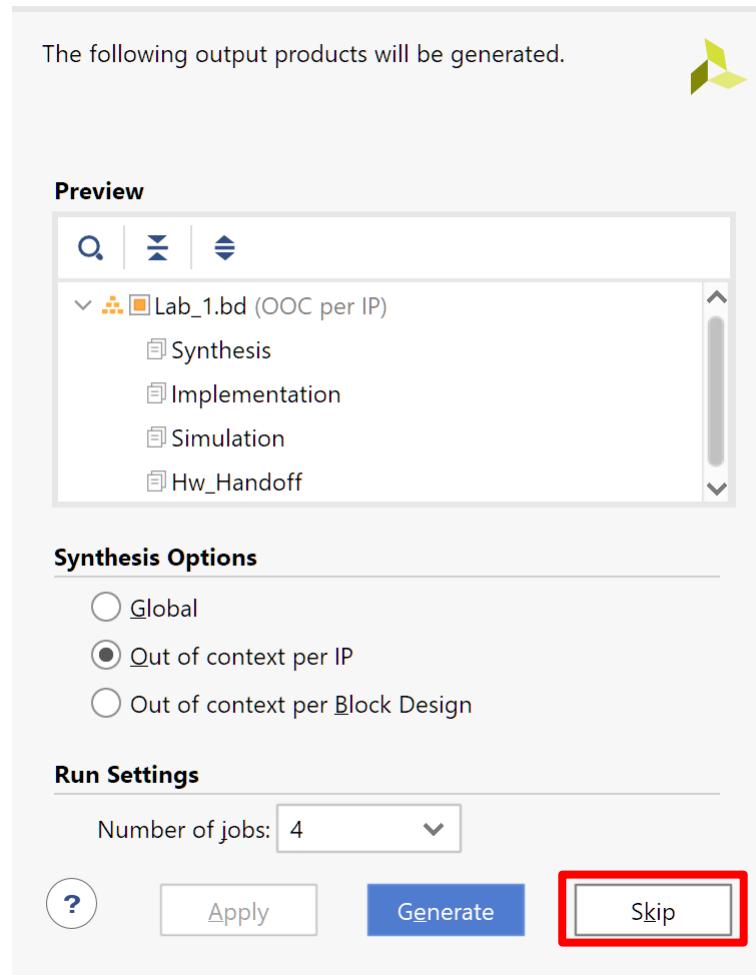
Source File	IP Status	Recommendation	Change Log	IP Name	Current Version	Recommended Version	License	Current Part
/adder_0	<input checked="" type="checkbox"/> IP revision change. IP definition 'adder_v1_0 (1.0)' changed on disk. Upgrade IP			adder_v1_0	1.0 (Rev. 2)	1.0 (Rev. 3)	Included	xc7z020cig400-1
/axi_regmap_0	<input type="checkbox"/> Up-to-date	No changes required	More info	axi_regmap_v1_0	1.0 (Rev. 3)	1.0 (Rev. 3)	Included	xc7z020cig400-1
/rst_ps7_0_100M	<input type="checkbox"/> Up-to-date	No changes required	More info	Processor System Reset	5.0 (Rev. 13)	5.0 (Rev. 13)	Included	xc7z020cig400-1
/processing_system7_0	<input type="checkbox"/> Up-to-date	No changes required	More info	ZYNQ7 Processing System	5.5 (Rev. 6)	5.5 (Rev. 6)	Included	xc7z020cig400-1
/ila_0	<input type="checkbox"/> Up-to-date	No changes required	More info	ILA (Integrated Logic Analyzer)	6.2 (Rev. 8)	6.2 (Rev. 8)	Included	xc7z020cig400-1

Click **OK**.



Click **Skip**.

NOTE: This process is automatically performed during bitstream generation.



Click Rerun.

File Edit Flow Tools Reports Window Layout View Help Quick Access

Flow Navigator

PROJECT MANAGER

- Settings
- Add Sources
- Language Templates
- IP Catalog

IP INTEGRATOR

- Create Block Design
- Open Block Design
- Generate Block Design

SIMULATION

- Run Simulation

RTL ANALYSIS

- Open Elaborated Design

SYNTHESIS

- Run Synthesis
- Open Synthesized Design

IMPLEMENTATION

- Run Implementation
- Open Implemented Design

PROGRAM AND DEBUG

- Generate Bitstream
- Open Hardware Manager

BLOCK DESIGN - tutorial

Sources

- Design Sources (1)
 - > tutorial_wrapper(STRUCTURE) (tutorial_wrapper)
- Constraints (1)
 - > constrs_1 (1)
 - tutorial.xdc
- Simulation Sources (1)
 - > sim_1 (1)
 - > Utility Sources

Hierarchy

Synthesis Run Properties

Name: synth_1
Part: xc7z020clg400-1 (active)
Description: Vivado Synthesis Defaults
Status: Not started
Constraints: constrs_1 (active)

Tcl Console

Report is out of date because the status of one or more IPs have changed. **Rerun**

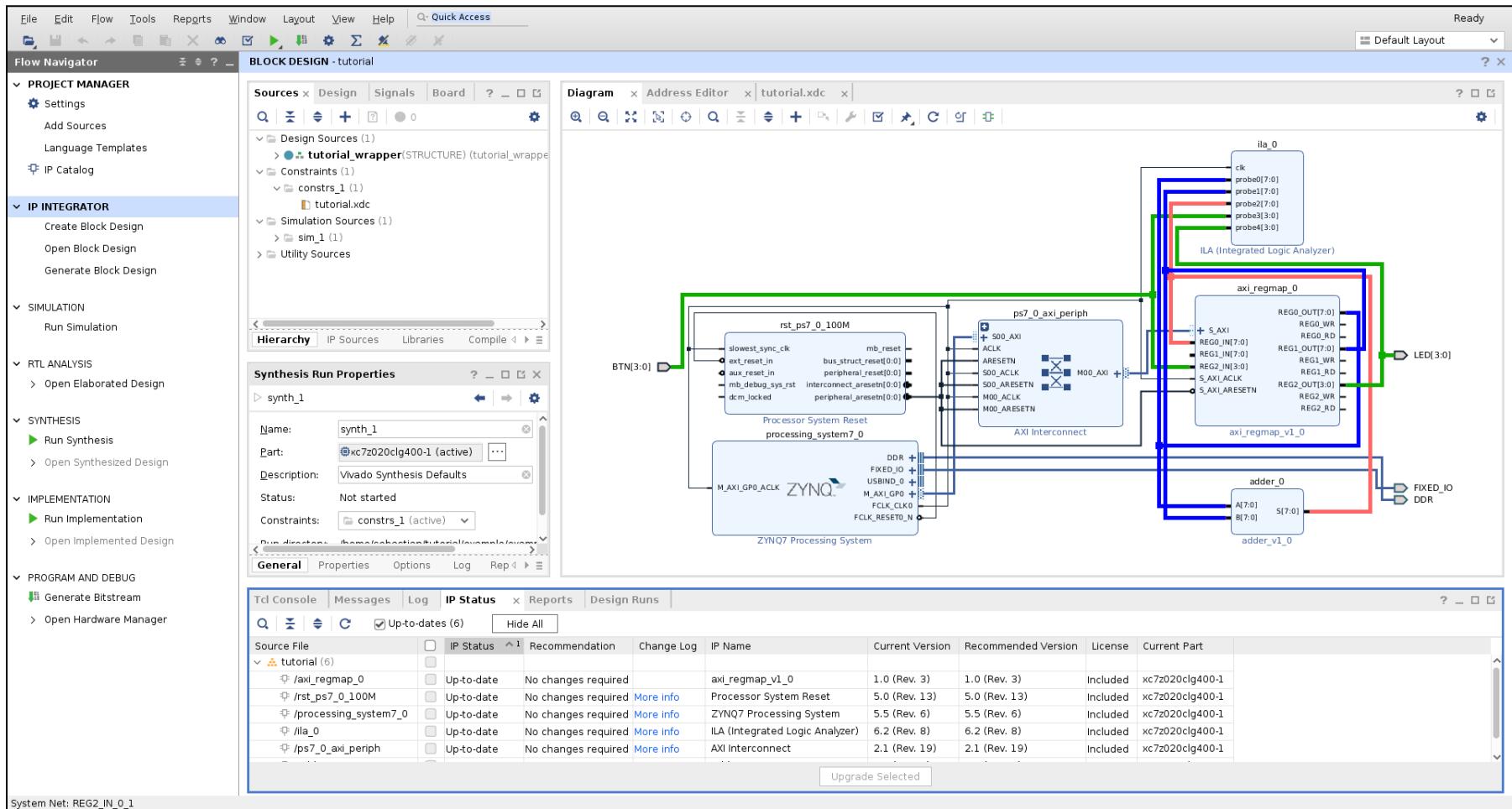
IP Status

Source File	IP Status	Recommendation	Change Log	IP Name	Current Version	Recommended Version	License	Current Part
tutorial (6)								
&adder_0	<input checked="" type="checkbox"/>	IP revision change. IP definition 'adder_v1_0 (1.0)' changed on disk	Upgrade IP	adder_v1_0	1.0 (Rev. 2)	1.0 (Rev. 3)	Included	xc7z020clg400-1
&/axi_regmap_0	<input type="checkbox"/>	Up-to-date		axi_regmap_v1_0	1.0 (Rev. 3)	1.0 (Rev. 3)	Included	xc7z020clg400-1
&/rst_ps7_0_100M	<input type="checkbox"/>	Up-to-date		Processor System Reset	5.0 (Rev. 13)	5.0 (Rev. 13)	Included	xc7z020clg400-1
&/processing_system7_0	<input type="checkbox"/>	Up-to-date		ZYNQ Processing System	5.5 (Rev. 6)	5.5 (Rev. 6)	Included	xc7z020clg400-1

Diagram

The diagram shows the ZYNQ SoC architecture. The ZYNQ Processing System contains several IP blocks: rst_ps7_0_100M, ps7_0_axi_periph, axi_regmap_0, axi_regmap_v1_0, adder_0, and DDR. The rst_ps7_0_100M block provides system reset signals (ext_reset_in, aux_reset_in, mb_debug_sys_rst, dcm_locked) and bus reset signals (bus_struct_reset[0:0], peripheral_reset[0:0]). The ps7_0_axi_periph block is connected to the AXI Interconnect. The axi_regmap_0 and axi_regmap_v1_0 blocks provide memory-mapped I/O (MMIO) access. The adder_0 block performs arithmetic operations on A[7:0] and B[7:0] to produce S[7:0]. The DDR block provides memory interface. External connections include a BTN[3:0] button, an ILA (Integrated Logic Analyzer) for monitoring signals like clk and probe[0:4], and an LED[3:0] output.

Vivado did not find any more IP to upgrade. Continue.



System Net: REG2_IN_0_1

Part 6: Generating a Design Bitstream

Next, we will generate a bitstream and analyze the implemented design. The generated outputs will be transferred to a new SDK project for software development.



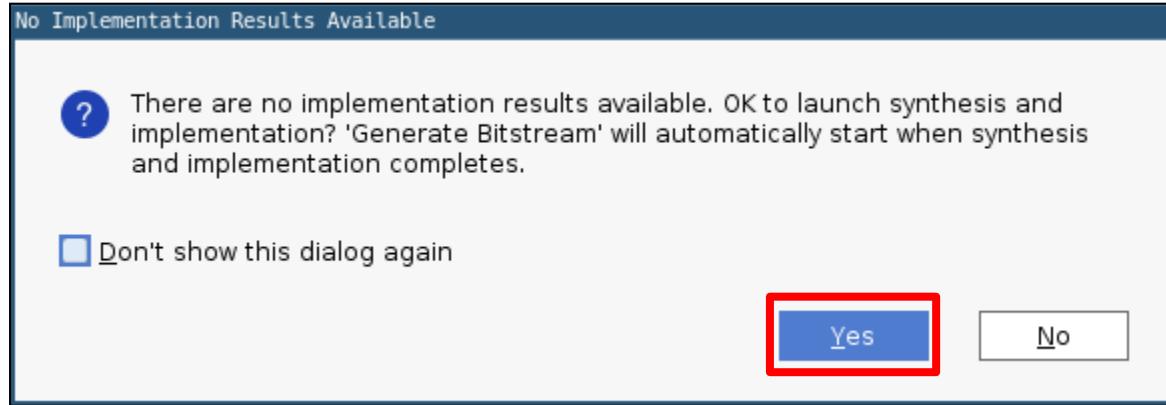
Click **Generate Bitstream**.

NOTE: This process will generate the design bitstream (including synthesis and implementation of the design).

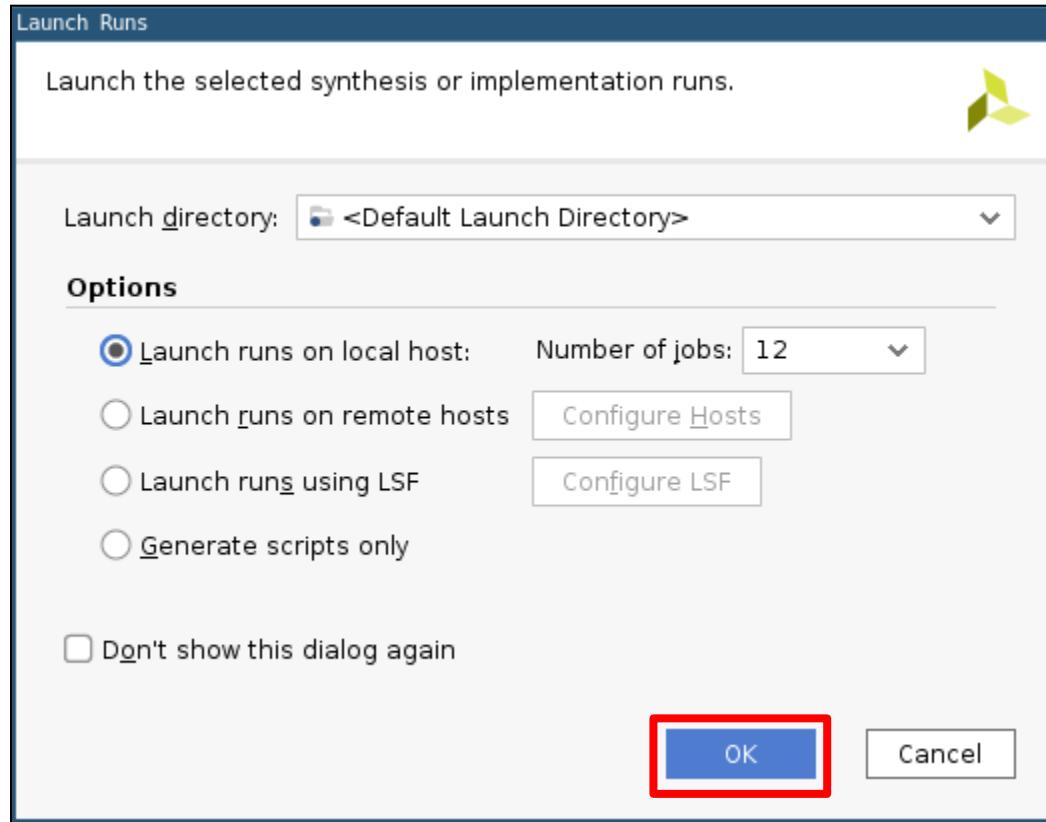
The screenshot shows the Vivado 2017.4 interface for a ZYNQ SoC design. The left sidebar contains project management, IP integrator, simulation, RTL analysis, synthesis, implementation, and program/debug options. The 'Generate Bitstream' button is highlighted with a red box. The main area displays the 'BLOCK DESIGN - tutorial' window, showing the system architecture with various IP blocks and their connections. The 'Diagram' tab is active, showing the physical layout of the ZYNQ Processing System. The bottom section shows the 'IP Status' tab of the Tcl Console, listing all IP components and their status.

Source File	IP Status	Recommendation	Change Log	IP Name	Current Version	Recommended Version	License	Current Part
tutorial (6)								
/rst_ps7_0_100M	Up-to-date	No changes required		Processor System Reset	5.0 (Rev. 13)	5.0 (Rev. 13)	Included	xc7z020clg400-1
/processing_system7_0	Up-to-date	No changes required	More Info	ZYNQ7 Processing System	5.5 (Rev. 6)	5.5 (Rev. 6)	Included	xc7z020clg400-1
/ila_0	Up-to-date	No changes required	More Info	ILA (Integrated Logic Analyzer)	6.2 (Rev. 8)	6.2 (Rev. 8)	Included	xc7z020clg400-1
/ps7_0_axi_periph	Up-to-date	No changes required	More Info	AXI Interconnect	2.1 (Rev. 19)	2.1 (Rev. 19)	Included	xc7z020clg400-1

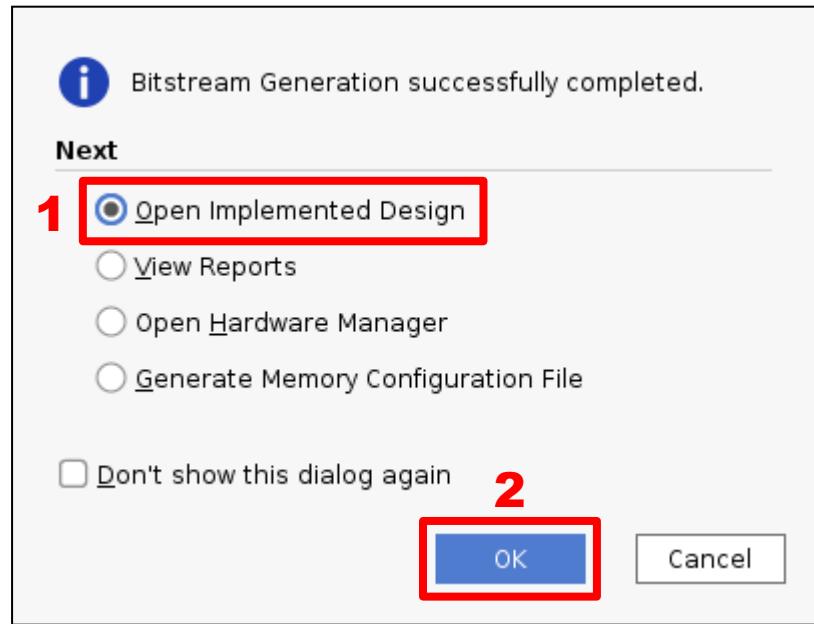
Click Yes.



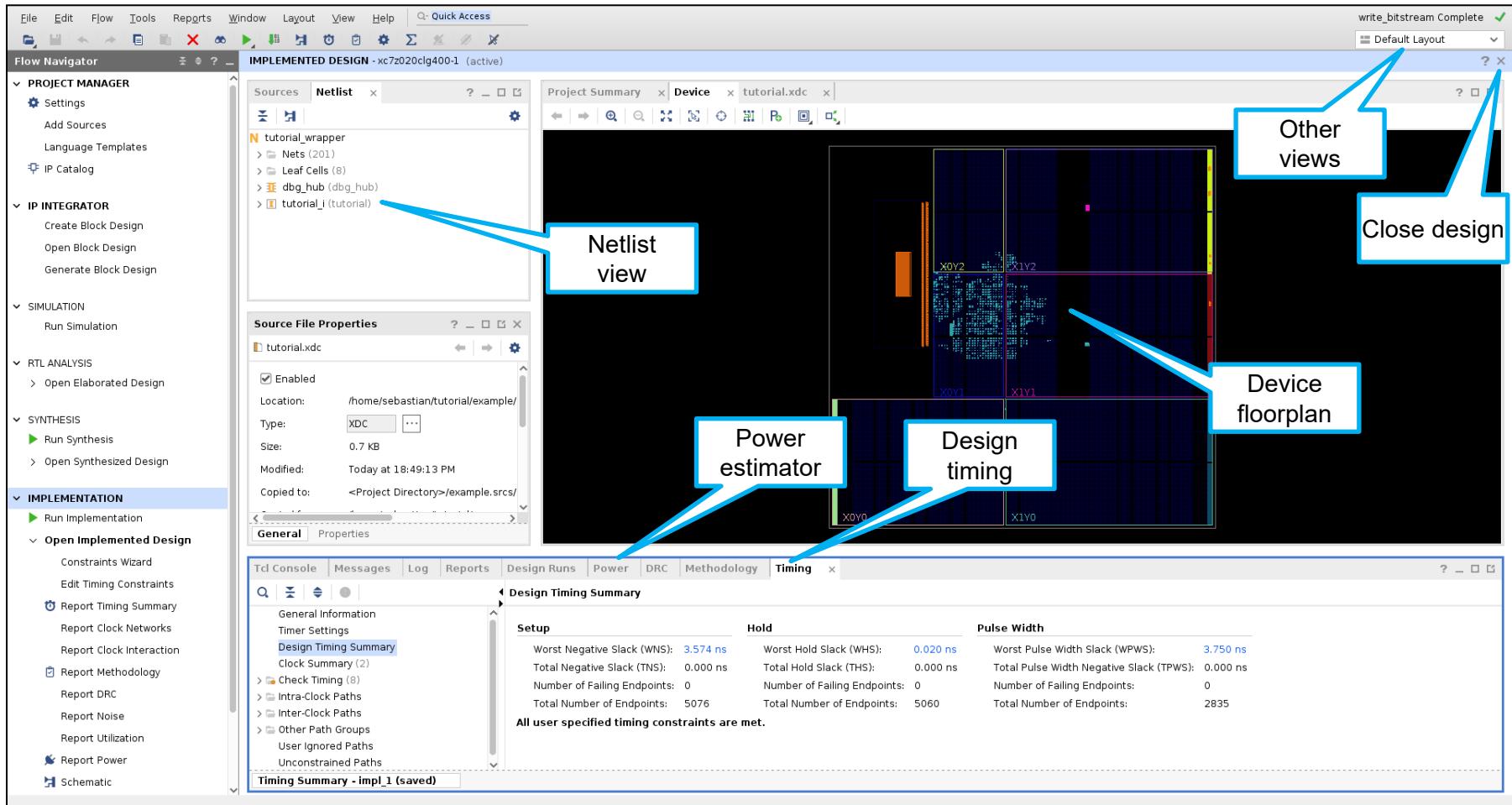
Click **OK**.



Wait until bitstream generation has completed. Select ***Open Implemented Design***. Click **OK**.



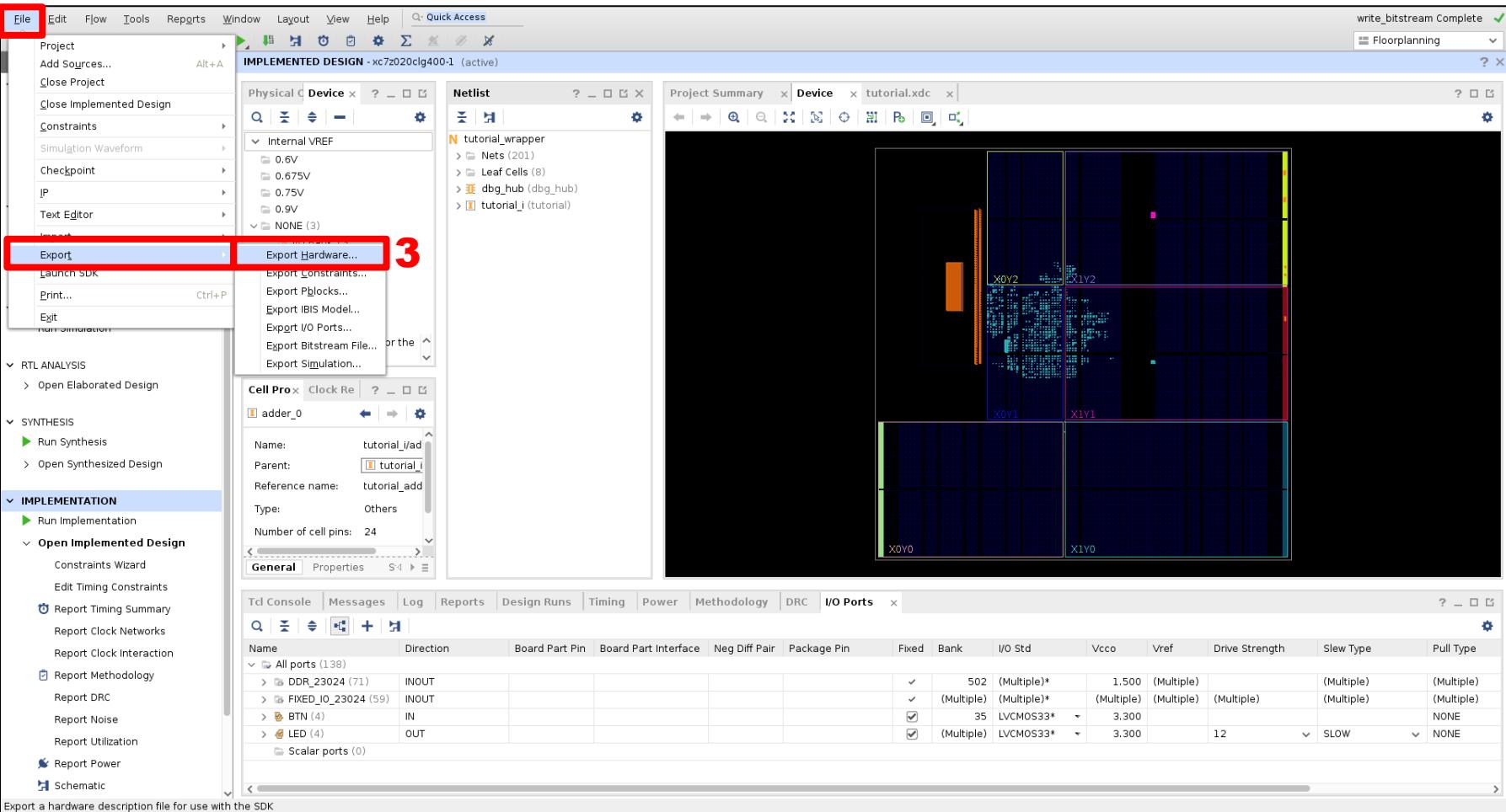
This is the Implemented Design GUI. Continue.



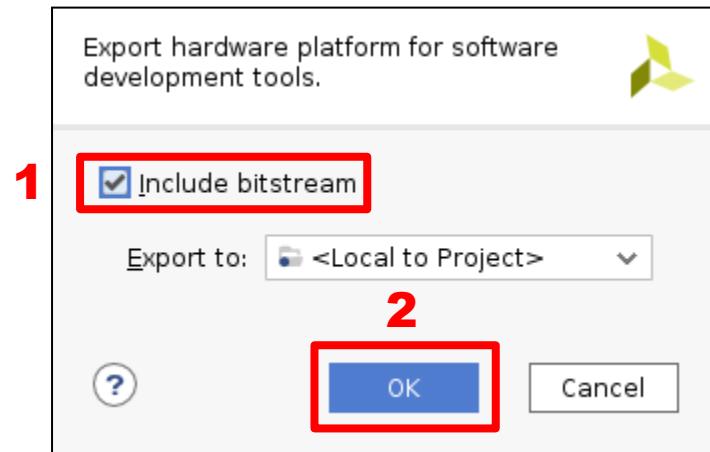
Click *File*. Click *Export*. Click *Export Hardware....*

NOTE: This process will export the design and launch SDK.

1

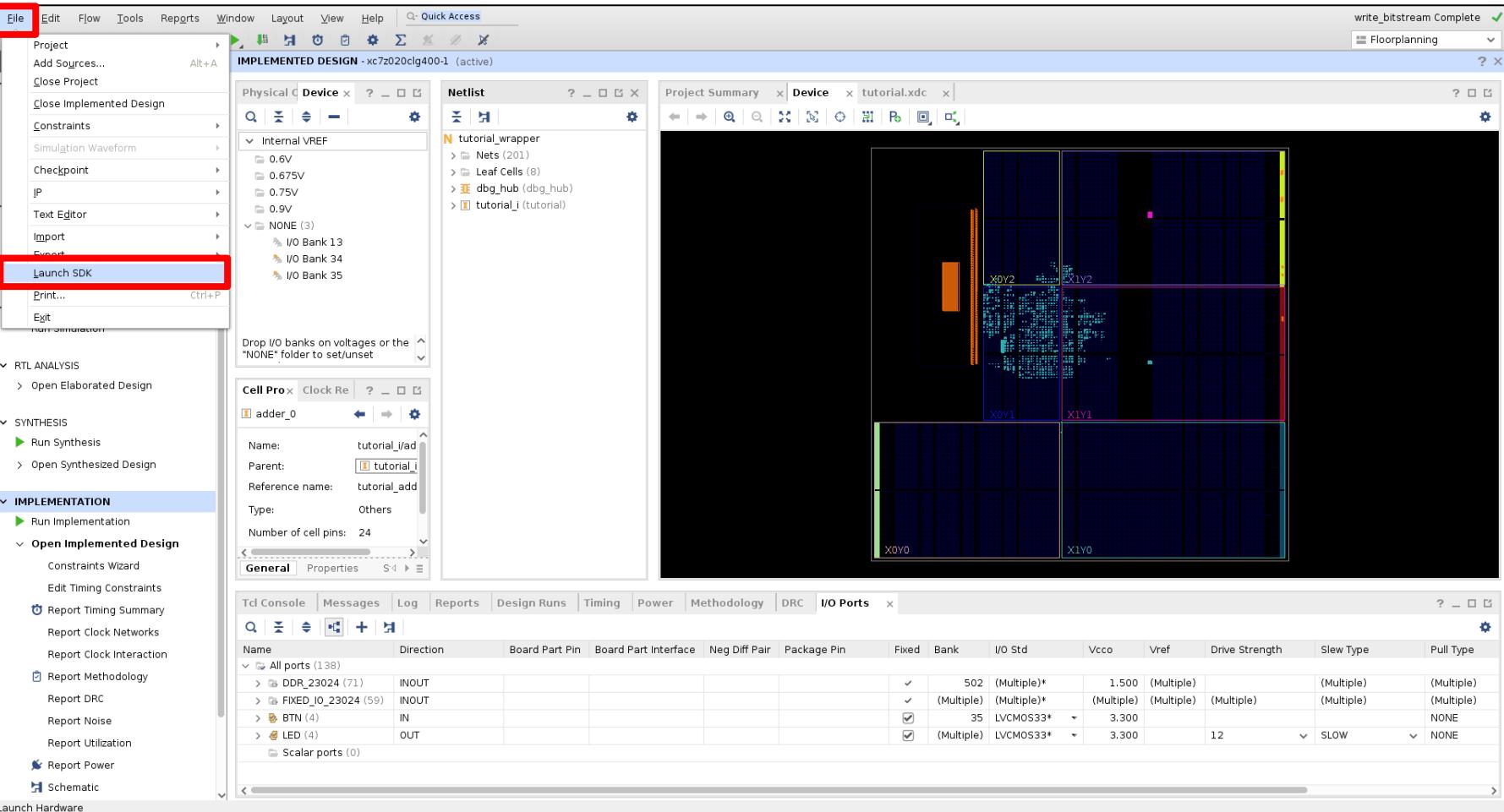


Check ***Include bitstream***. Click ***OK***.

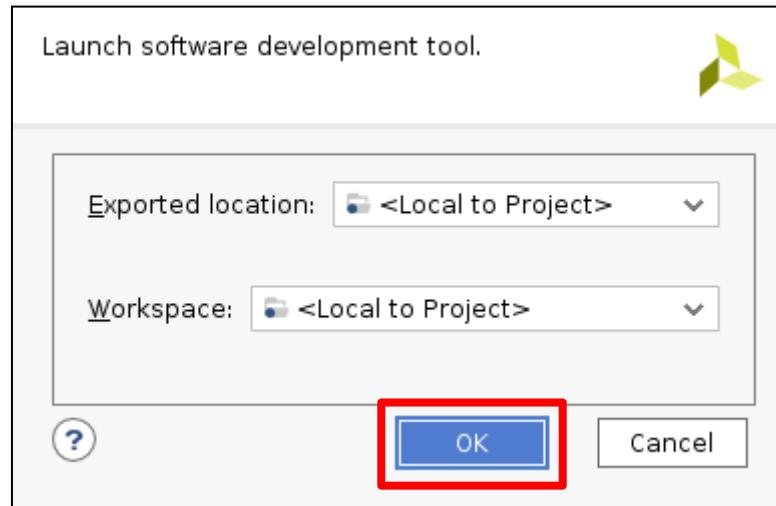


Click **File**. Click **Launch SDK**.

1



Click **OK**.



Xilinx SDK Tutorial



Part 1: Creating Application Projects

We will create two application projects: **adder** and **io**.
The **adder** app will test the **adder** IP, and the **io** app will echo button inputs to LED outputs.



This is the SDK GUI. Continue.

The screenshot shows the Xilinx SDK GUI interface. The main window displays the "tutorial_wrapper_hw_platform_0" hardware platform specification. The "Address Map for processor ps7_cortexa9 [0-1]" table lists memory components and their addresses:

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
ps7_intc_dist_0	0xf8f01000	0xf8f01fff		REGISTER
axi_regmap_0	0x70000000	0x7000ffff	S_AXI	REGISTER
ps7_gpio_0	0xe000a000	0xe000afff		REGISTER
ps7_scutimer_0	0xf8f00600	0xf8f0061f		REGISTER
ps7_sclcr_0	0xf8000000	0xf8000fff		REGISTER
ps7_scuwdt_0	0xf8f00620	0xf8f006ff		REGISTER
ps7_l2cachec_0	0xf8f02000	0xf8f02fff		REGISTER
ps7_scuc_0	0xf8f00000	0xf8f000fc		REGISTER
ps7_qspi_linear_0	0xfc000000	0xfcfffff		FLASH
ps7_pmu_0	0xf8893000	0xf8893fff		REGISTER
ps7_afi_1	0xf8009000	0xf8009fff		REGISTER
ps7_afi_0	0xf8008000	0xf8008fff		REGISTER
ps7_qspi_0	0xe000d000	0xe000dff		REGISTER
ps7_usb_0	0xe0002000	0xe0002fff		REGISTER
ps7_afi_3	0xf800b000	0xf800bfff		REGISTER
ps7_afi_2	0x80000000	0x8000afff		REGISTER

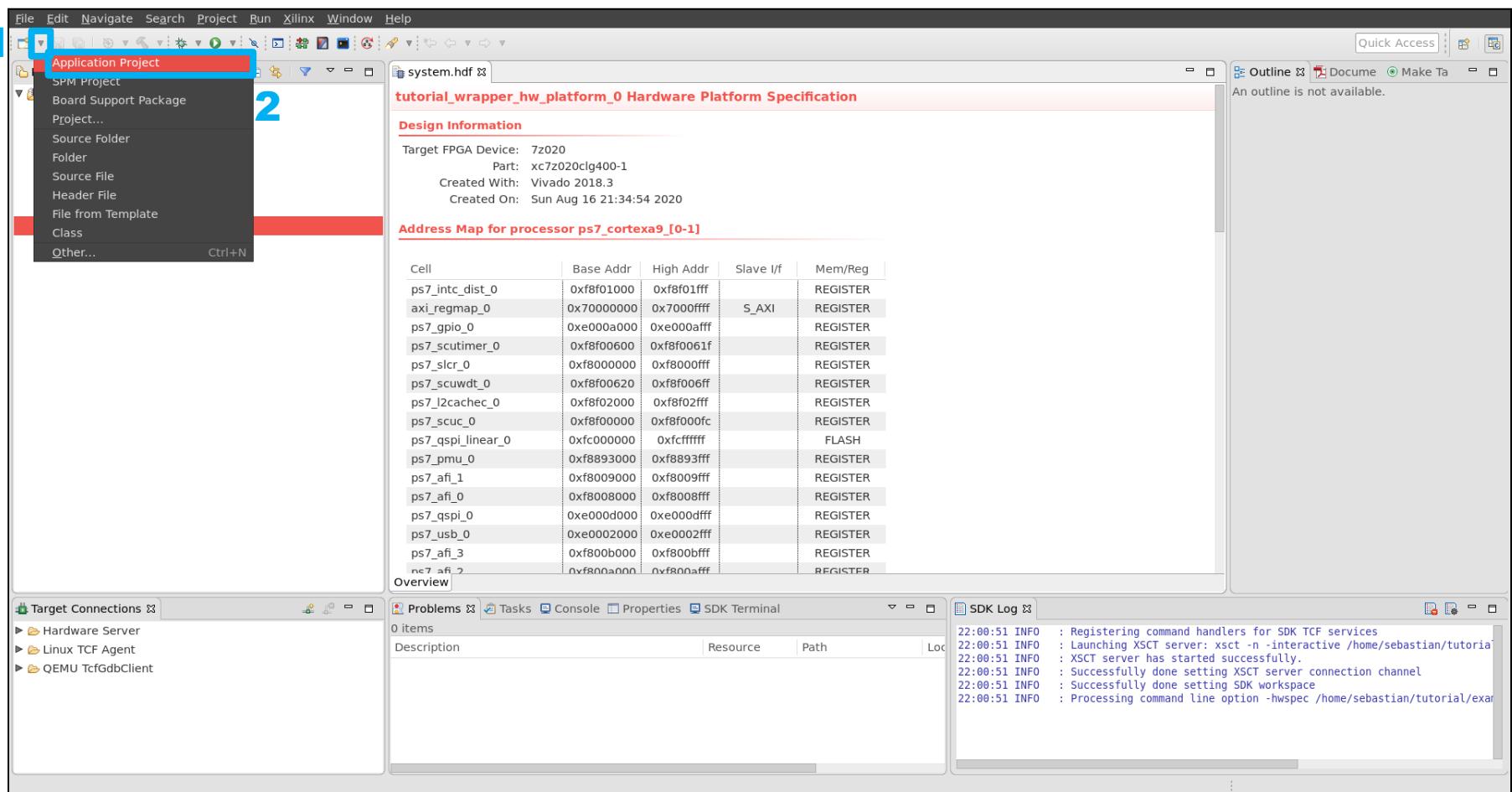
The "SDK Log" panel shows the following log entries:

```
22:00:51 INFO : Registering command handlers for SDK TCF services
22:00:51 INFO : Launching XSCT server: xsct -n -interactive /home/sebastian/tutorial/exam
22:00:51 INFO : XSCT server has started successfully.
22:00:51 INFO : Successfully done setting XSCT server connection channel
22:00:51 INFO : Successfully done setting SDK workspace
22:00:51 INFO : Processing command line option -hwspec /home/sebastian/tutorial/exam
```

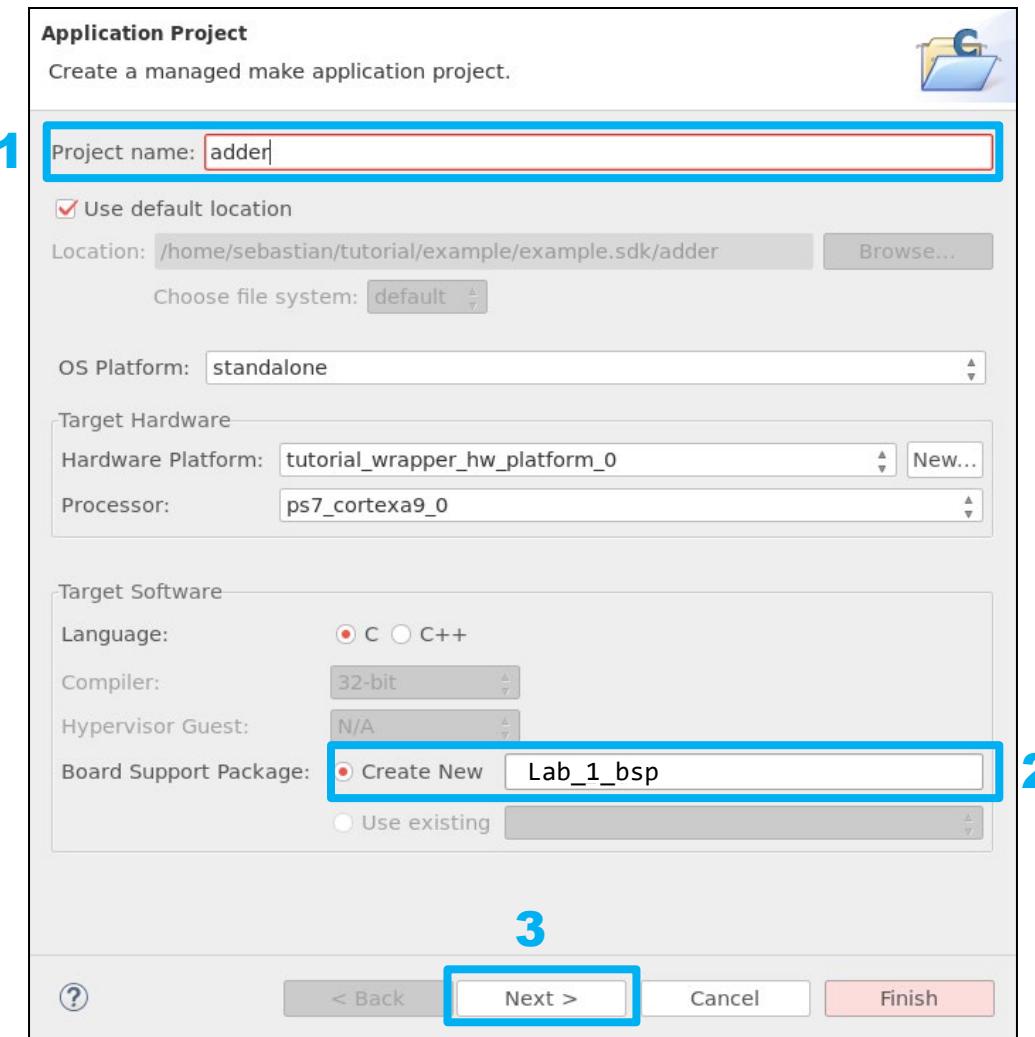
Click the **New (drop-down)**. Click **Application Project**.

1

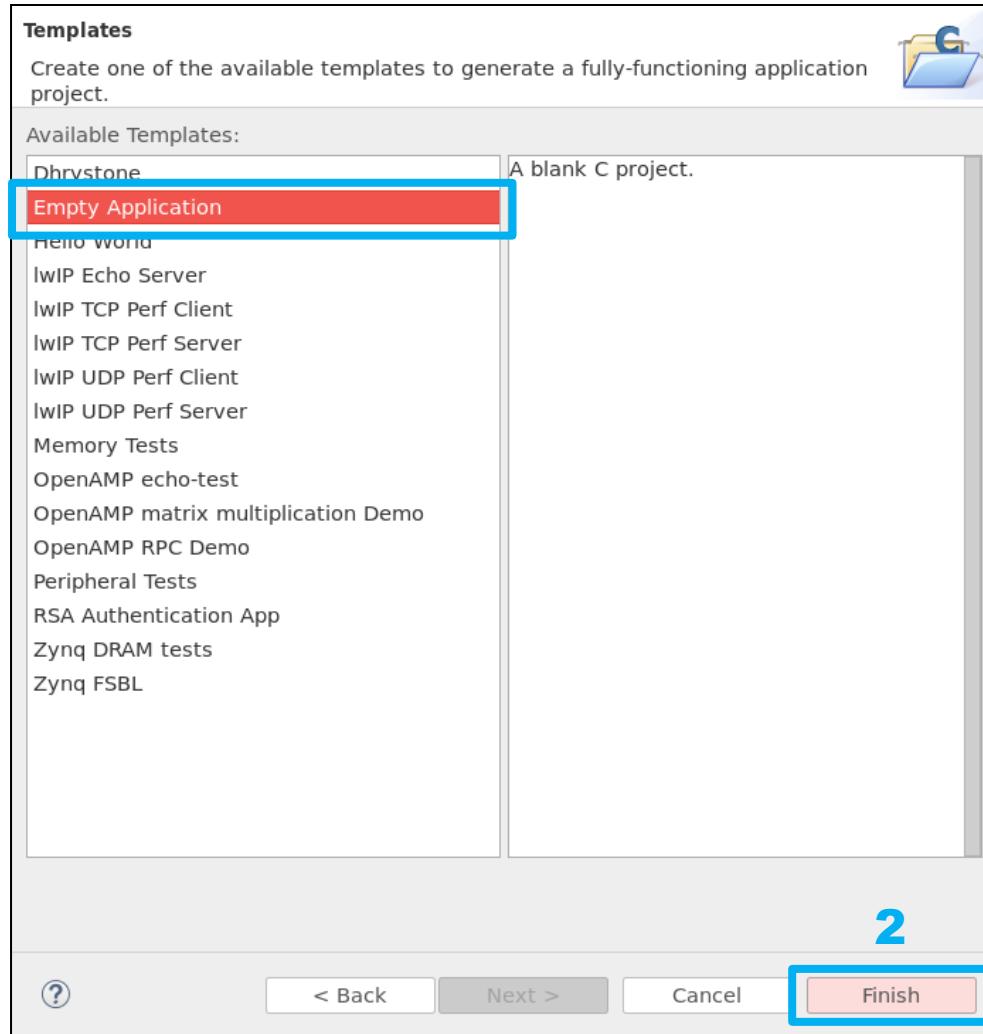
2



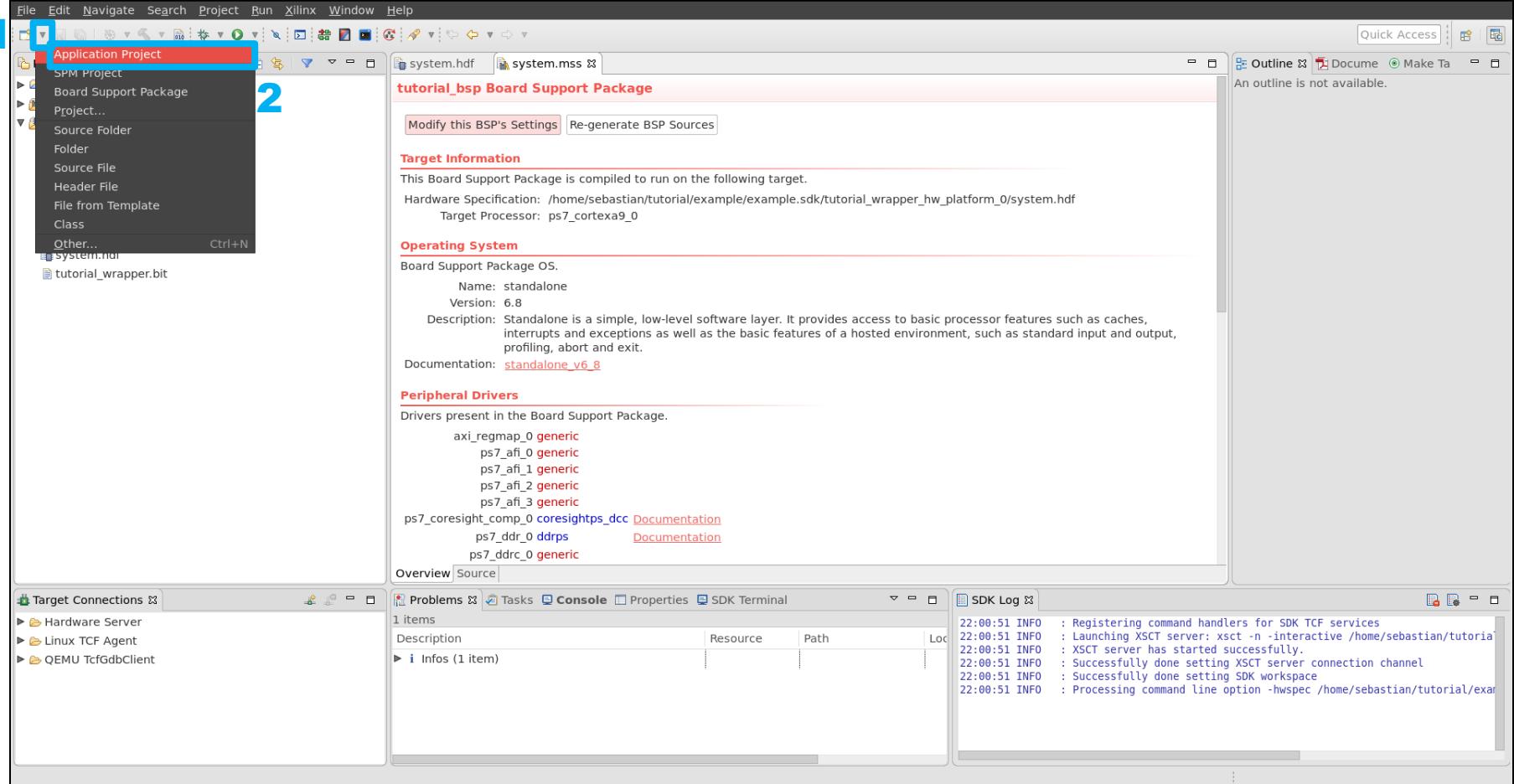
Enter project name **adder**. Select to **Create New BSP** and enter **Lab_1_bsp**. Click **Next**. Note "naming might be different than the picture



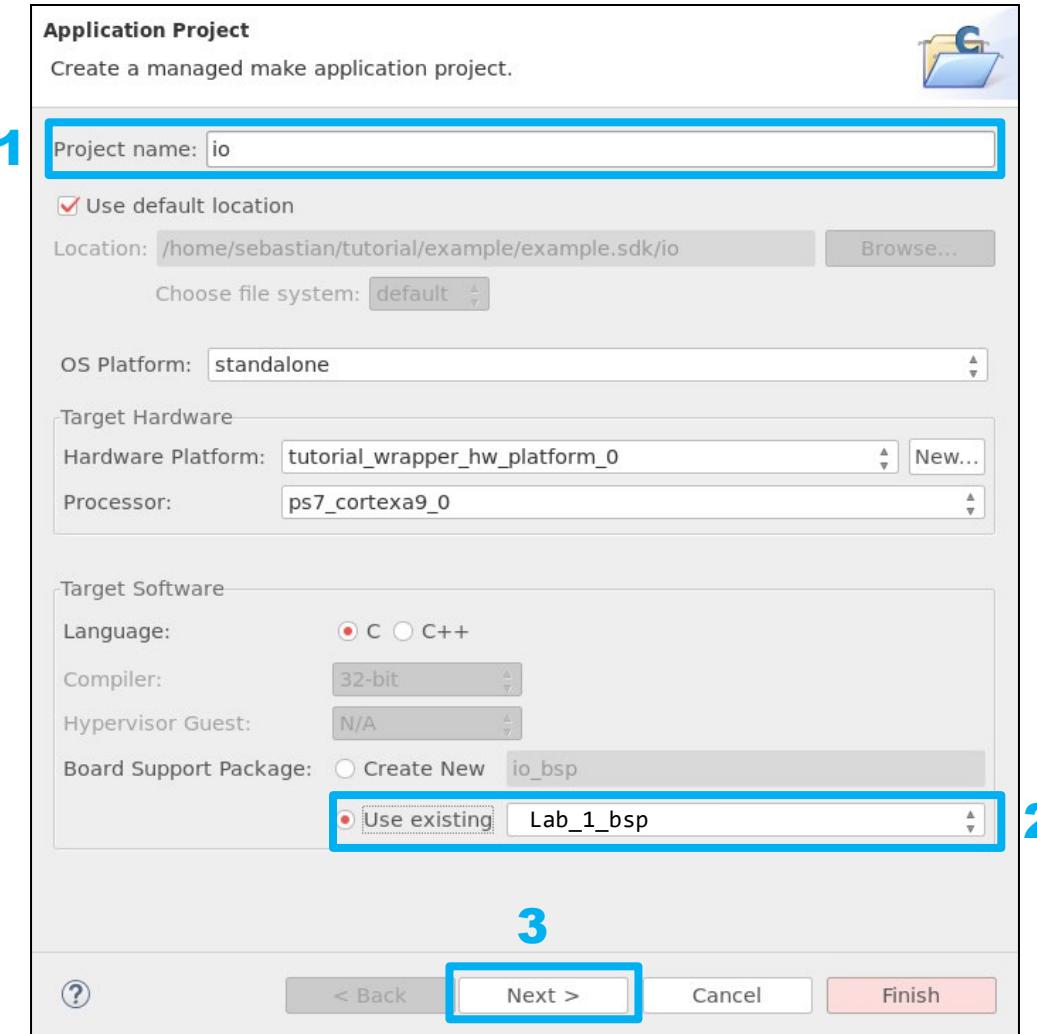
Select **Empty Application**. Click **Finish**.



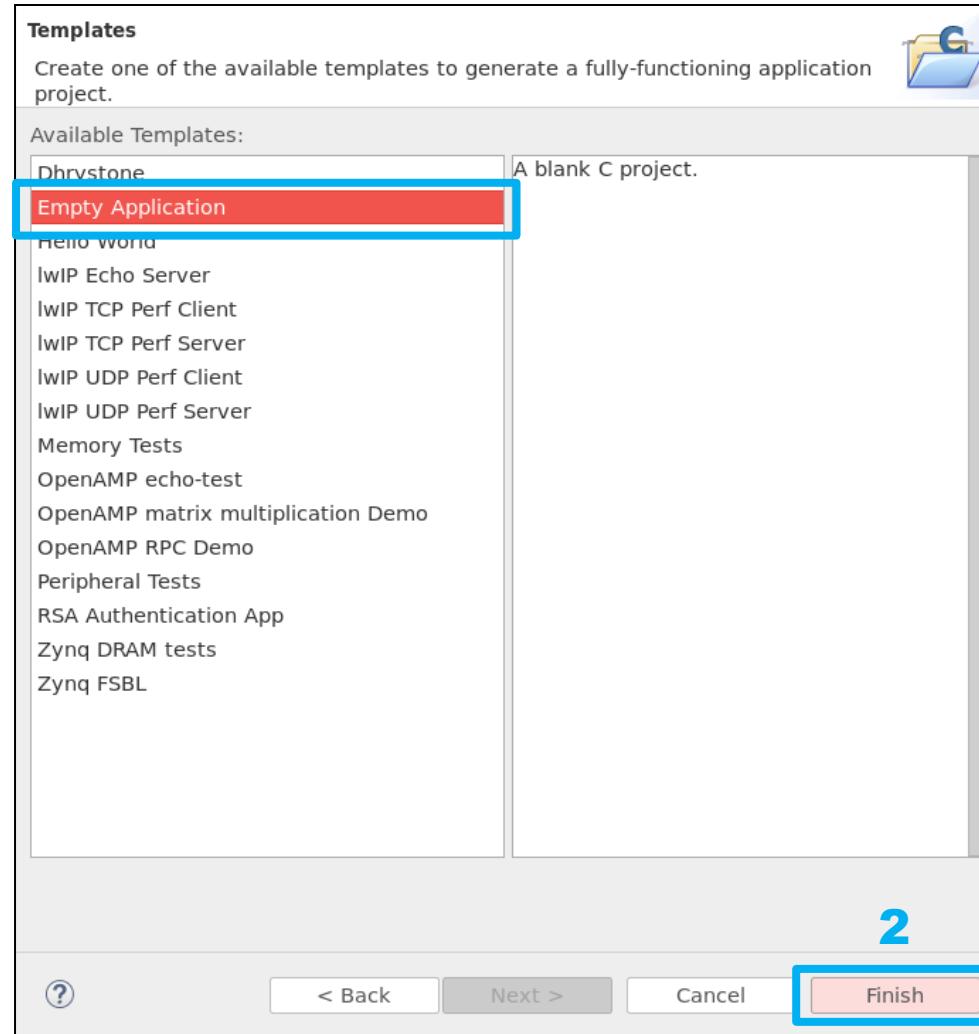
Click the **New (drop-down)**. Click **Application Project**.



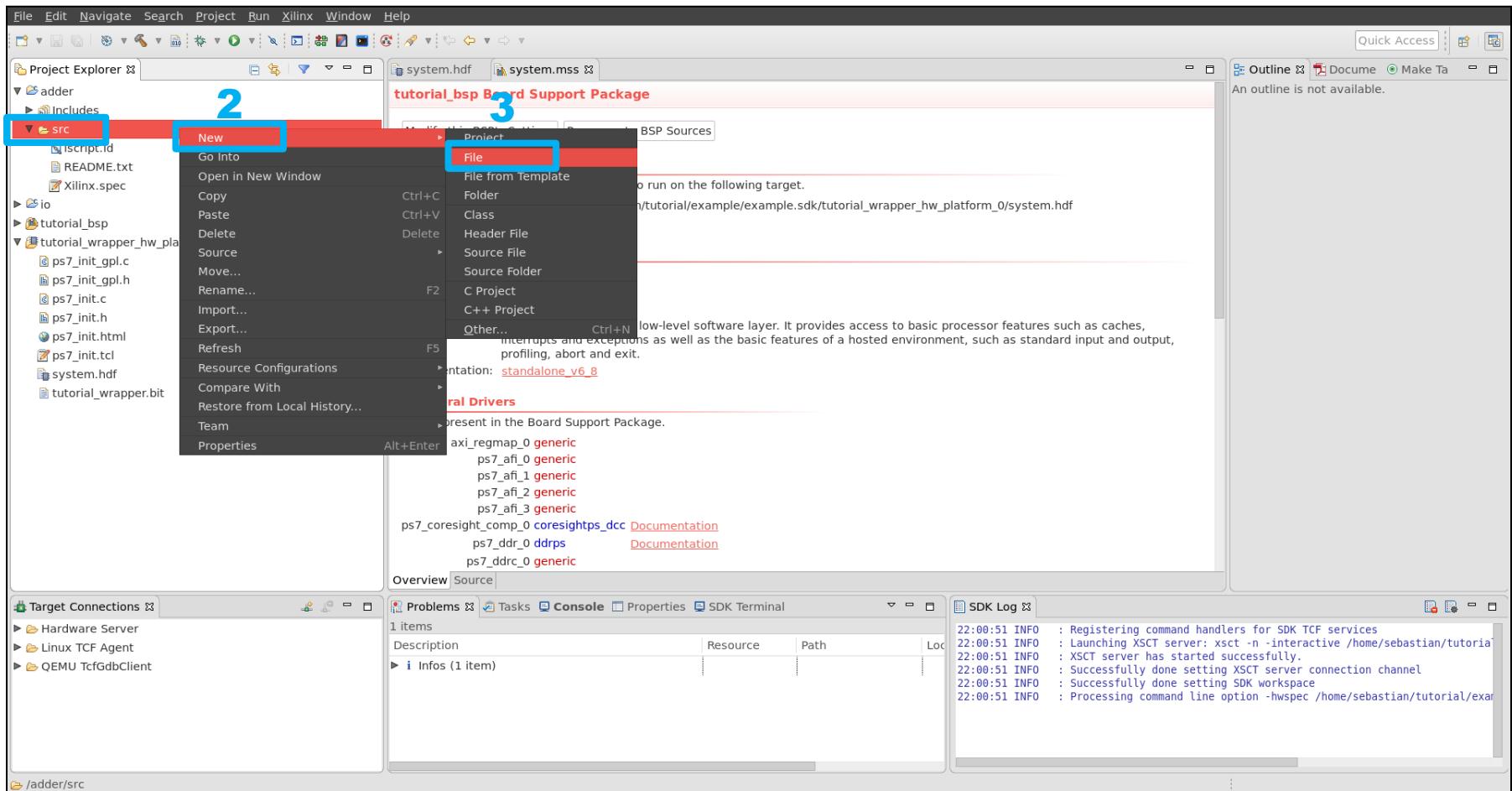
Enter project name **io**. Select to **Use existing** BSP and select **Lab_1_bsp**. Click **Next**.



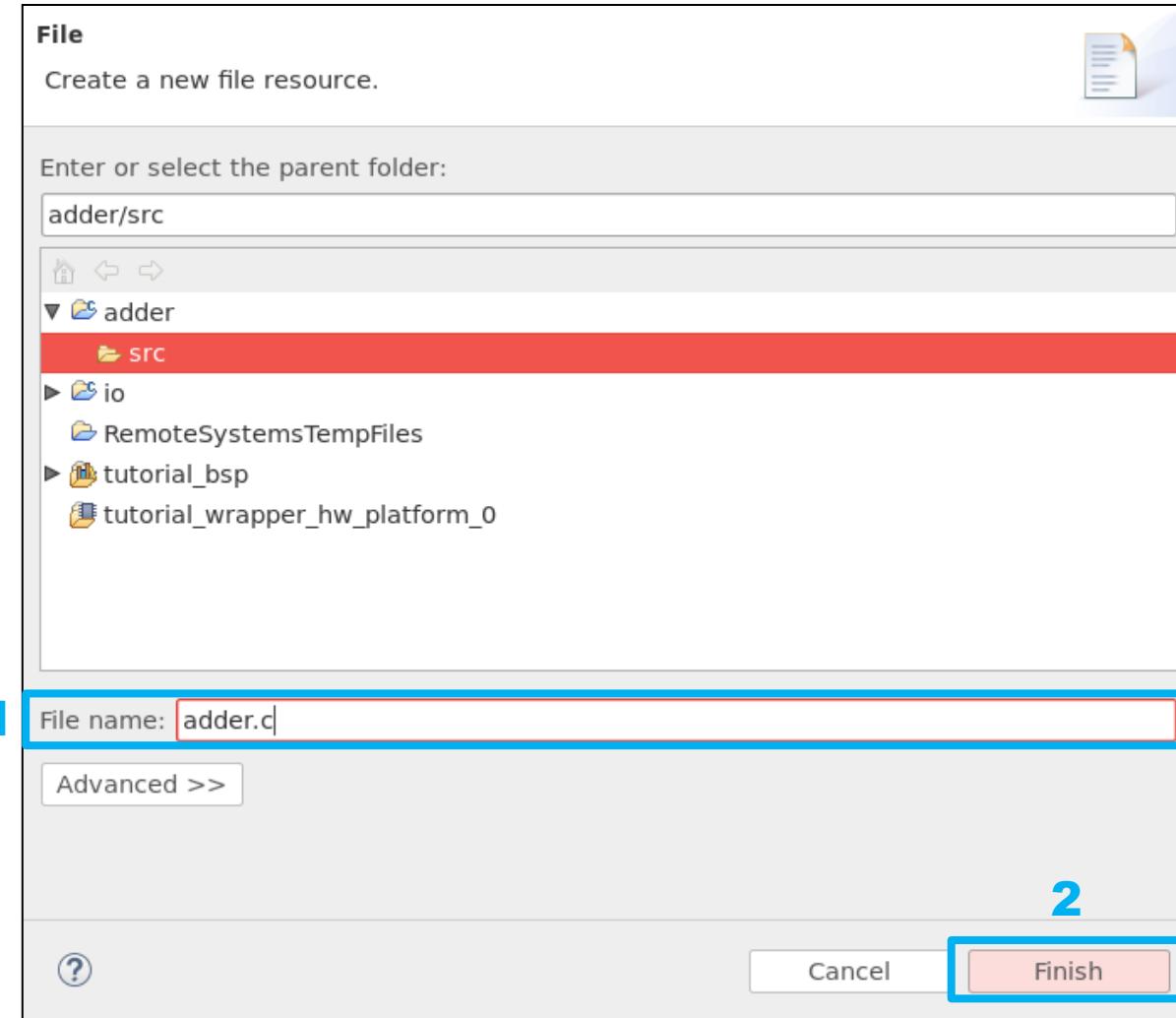
Select **Empty Application**. Click **Finish**.



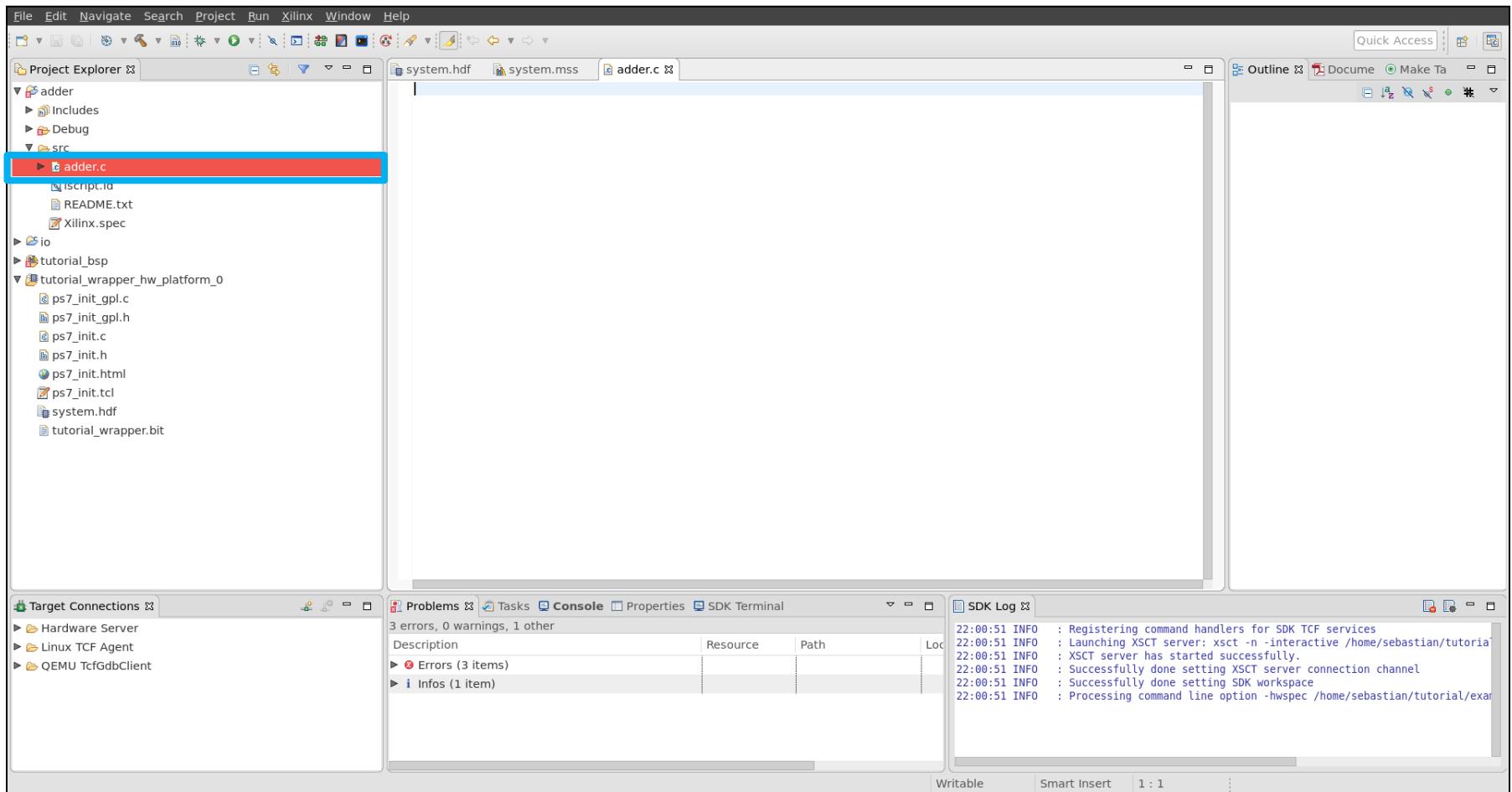
Right-click **src** directory of the **adder** application project. Click **New**. Click **File**.



Enter file name **adder.c**. Click **Finish**.



Double-click `src/adder.c` file of the **adder** application project to open it in the editor.



Copy-paste contents of **Lab_1_repo/resources/adder.c** to **adder.c** in the editor.

The screenshot shows the Vivado IDE interface with the following components:

- Project Explorer:** Shows the project structure under "adder". The file "adder.c" is selected and highlighted with a red border.
- Code Editor:** Displays the C code for "adder.c". The code initializes a register map at address 0x70000000, writes to registers 0 and 1, reads from register 0, and prints the result. It includes comments explaining the Vivado address editor setup.
- Outline:** Shows the declaration of the main function in stdio.h.
- Target Connections:** Lists "Hardware Server", "Linux TCF Agent", and "QEMU TcfGdbClient".
- SDK Log:** Displays log messages from the Xilinx Software Configuration Tool (XSCT) server, indicating successful registration of command handlers and launching of the XSCT server.

adder.c. The main loop assigns iterators to adder inputs via writes to registers 0 and 1, respectively, and retrieves adder result via reads from register 0. Continue.

NOTE: The base address **0x7000_0000** matches the base address set in Vivado.

```
#include <stdio.h>

int main(void)
{
    // base address of axi_regmap as set in the Vivado address editor
    uint32_t *regmap = (uint32_t *) 0x70000000;

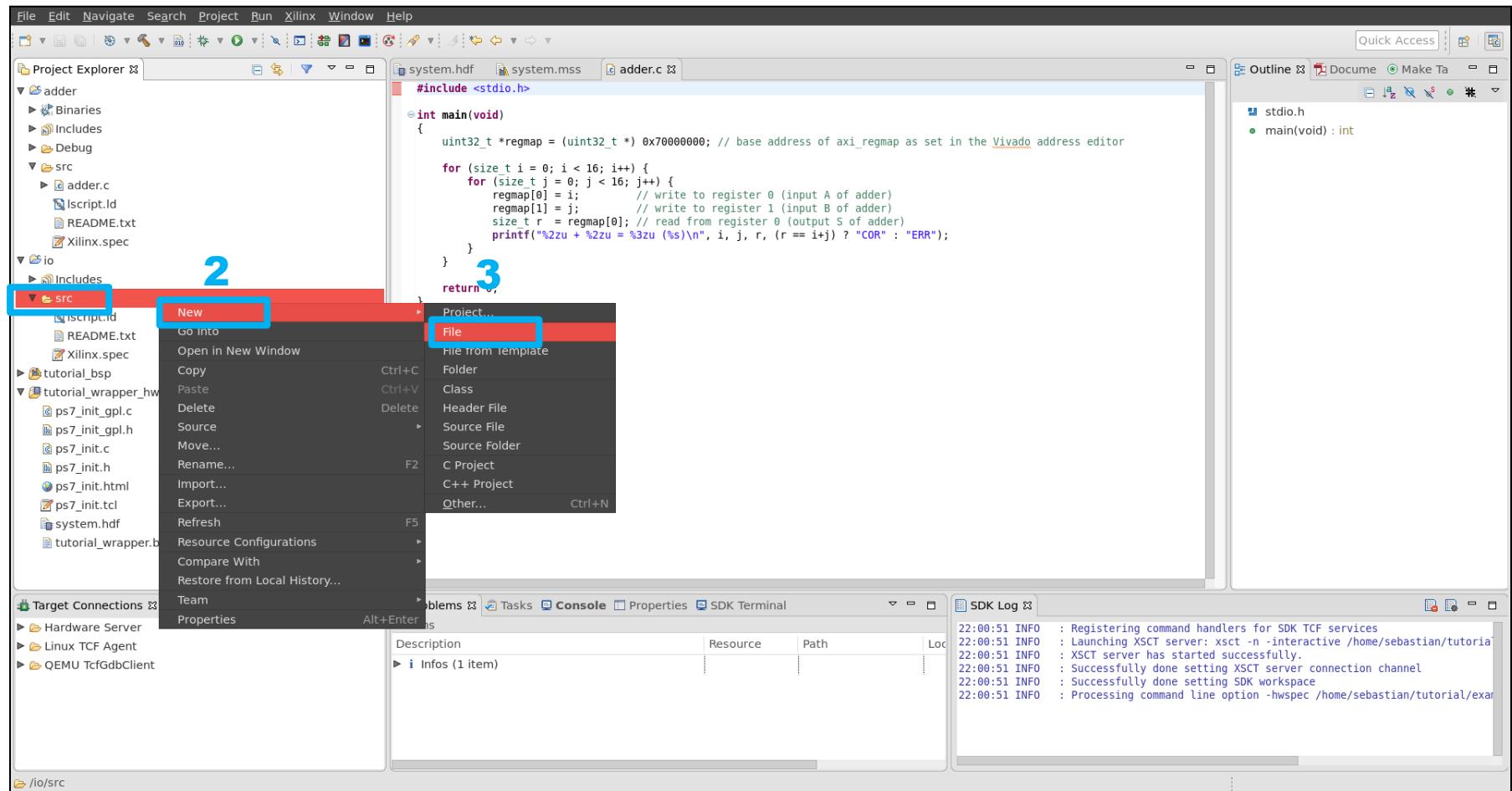
    for (size_t i = 0; i < 16; i++) {
        for (size_t j = 0; j < 16; j++) {
            regmap[0] = i;           // write to register 0 (REG0_OUT, input A of adder)
            regmap[1] = j;           // write to register 1 (REG1_OUT, input B of adder)
            size_t r = regmap[0];   // read from register 0 (REG0_IN, output S of adder)
            printf("%zu + %zu = %zu (%s)\n", i, j, r, (r == i+j) ? "COR" : "ERR");
        }
    }

    return 0;
}
```

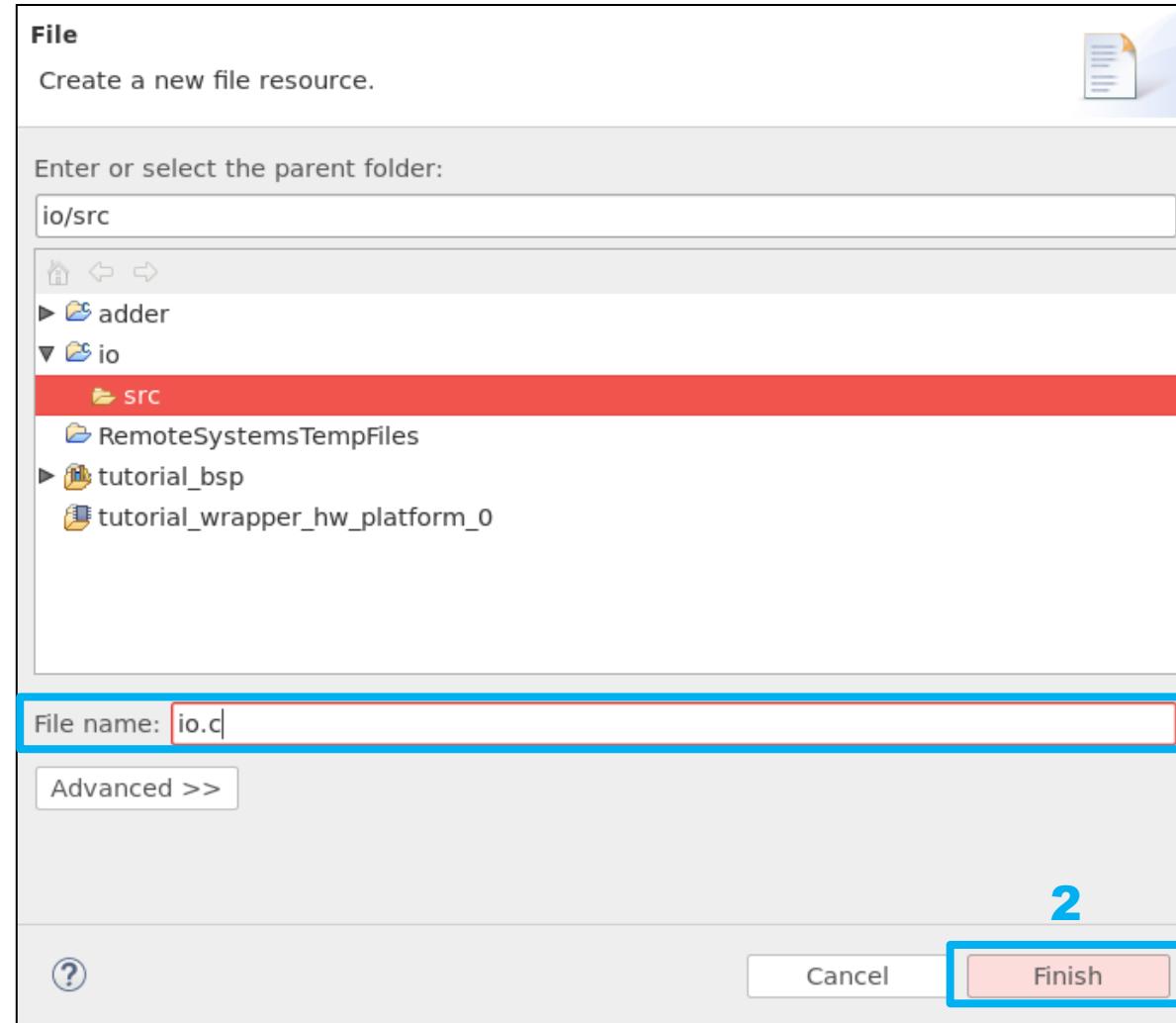
Register Map			
Index	Offset	Read Access	Write Access
[0]	0x00	S (adder output)	A (adder input)
[1]	0x04	Zero	B (adder input)
[2]	0x08	BTN	LED



Right-click **src** directory of the **io** application project. Click **New**. Click **File**.



Enter file name **io.c**. Click **Finish**.



Double-click **src/io.c** file of the **io** application project to open it in the editor. Copy-paste contents of **Lab_1_repo/resources/io.c** to **io.c** in the editor.

```
#include <stdio.h>

int main(void)
{
    uint32_t *regmap = (uint32_t *) 0x70000000; // base address of axi_regmap as set in the Vivado address editor

    while (1) {
        size_t io = regmap[2]; // read from register 2 (input BTN)
        regmap[2] = io;         // write to register 2 (output LED)
    }

    return 0;
}
```

io.c. The main loop inputs from buttons via reads from register 2 and outputs to LEDs via writes to register 2. Continue.

```
#include <stdio.h>

int main(void)
{
    // base address of axi_regmap as set in the Vivado address editor
    uint32_t *regmap = (uint32_t *) 0x70000000;

    while (1) {
        size_t io = regmap[2]; // read from register 2 (REG2_IN, input BTN)
        regmap[2] = io;         // write to register 2 (REG2_OUT, output LED)
    }

    return 0;
}
```

Register Map			
Index	Offset	Read Access	Write Access
[0]	0x00	S (adder output)	A (adder input)
[1]	0x04	Zero	B (adder input)
[2]	0x08	BTN	LED



Tutorial Demo



Part 1: Board Setup



Digilent PYNQ-Z1

Boot Mode

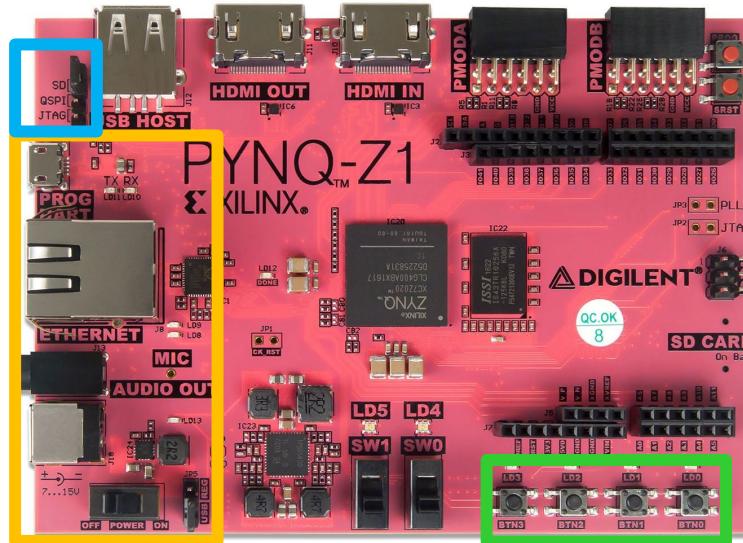


JTAG/UART



Power
Switch

Power Source
Select



Buttons and LEDs



Go to the following link to download PuTTY. It will be used for opening a serial session with the target board.

<https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>



The screenshot shows a web browser window with the URL [chiark.greenend.org.uk/~sgtatham/putty/latest.html](https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html) in the address bar. The page title is "Download PuTTY: latest release (0.74)". Below the title are links to Home, FAQ, Feedback, Licence, Updates, Mirrors, Keys, Links, Team, and a download section for Stable, Snapshot, Docs, Changes, and Wishlist. A note states that the page contains download links for the latest released version of PuTTY (0.74, released on 2020-06-27). It also mentions that new releases will update the page. A link to a permanent link to the 0.74 release is provided. A section titled "Package files" lists download options: MSI ("Windows Installer") for 32-bit and 64-bit, and a Unix source archive (.tar.gz). Each download link includes an alternative link (or by FTP) and a signature link.

Download PuTTY: latest release (0.74)

[Home](#) | [FAQ](#) | [Feedback](#) | [Licence](#) | [Updates](#) | [Mirrors](#) | [Keys](#) | [Links](#) | [Team](#)
Download: **Stable** · [Snapshot](#) | [Docs](#) | [Changes](#) | [Wishlist](#)

This page contains download links for the latest released version of PuTTY. Currently this is 0.74, released on 2020-06-27.

When new releases come out, this page will update to contain the latest, so this is a good page to bookmark or link to. Alternatively, here is a [permanent link to the 0.74 release](#).

Release versions of PuTTY are versions we think are reasonably likely to work well. However, they are often not the most up-to-date version of the code available. If you have a problem with this release, then it might be worth trying out the [development snapshots](#), to see if the problem has already been fixed in those versions.

Package files

You probably want one of these. They include versions of all the PuTTY utilities.

(Not sure whether you want the 32-bit or the 64-bit version? Read the [FAQ entry](#).)

MSI ("Windows Installer")

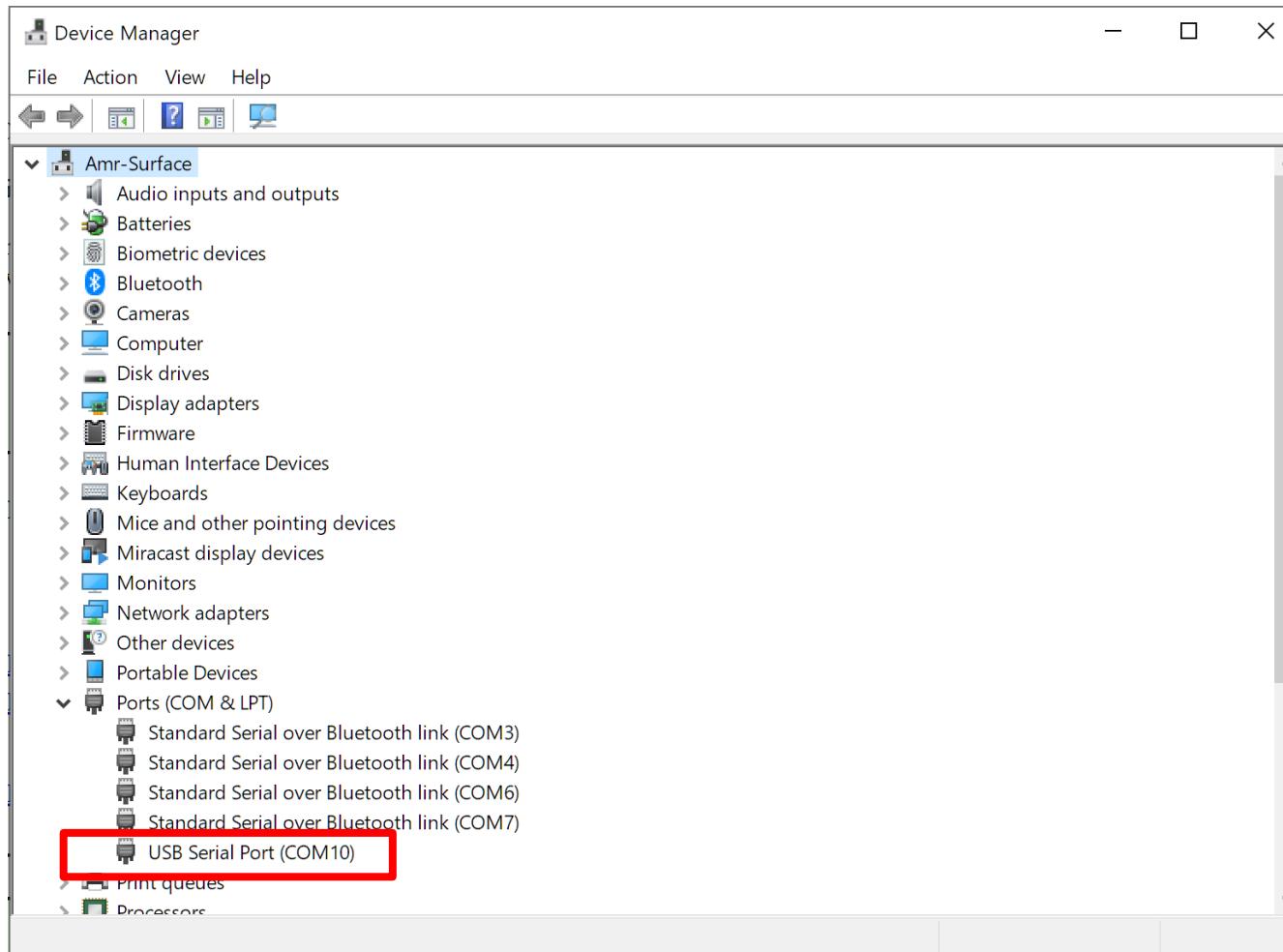
32-bit:	putty-0.74-installer.msi	(or by FTP)	(signature)
64-bit:	putty-64bit-0.74-installer.msi	(or by FTP)	(signature)

Unix source archive

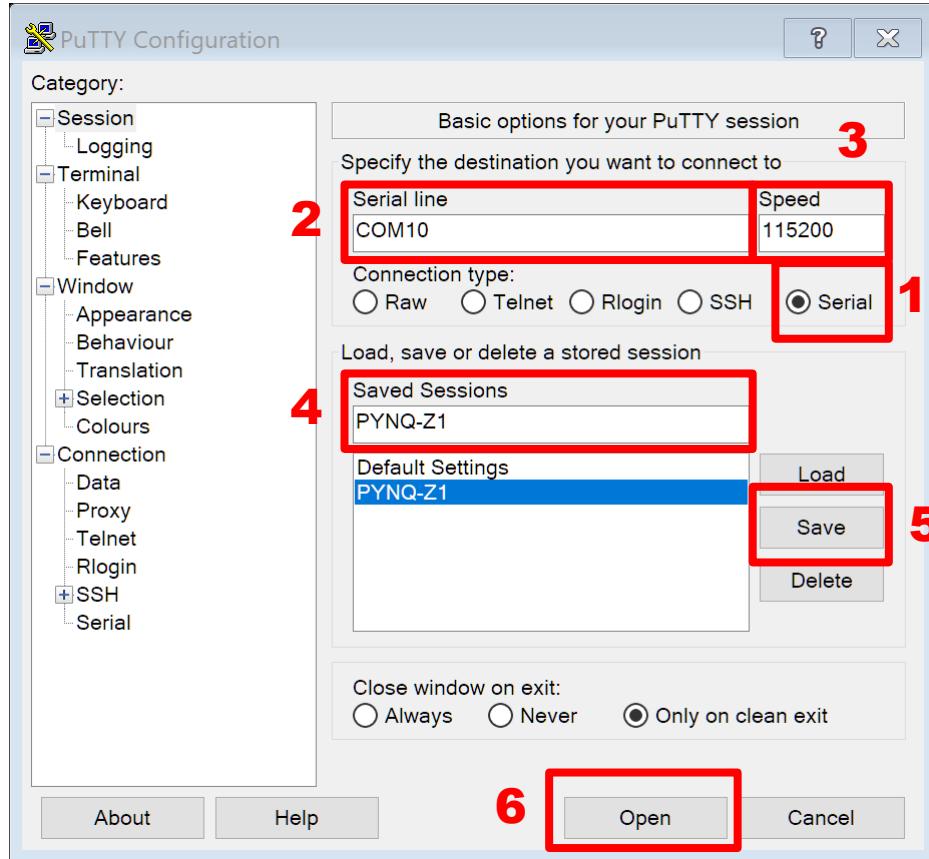
.tar.gz:	putty-0.74.tar.gz	(or by FTP)	(signature)
----------	-----------------------------------	-------------------------------	-------------------------------

Connect the board via USB cable and turn it on. Open the device manager and record the COM port that corresponds to the board.

Note: if you can't figure it out, while the device manager window is opened, turn off the board and back on again. Note the COM# added as "USB Serial Port"



Open PuTTY. Select **Serial**, add **COM#** under **Serial Line**, change **Speed** to **115200**. In order to save those settings, Enter **PYNQ-Z1** under **saved sessions** and click **save**. Click **Open** to open a terminal connected to the board.

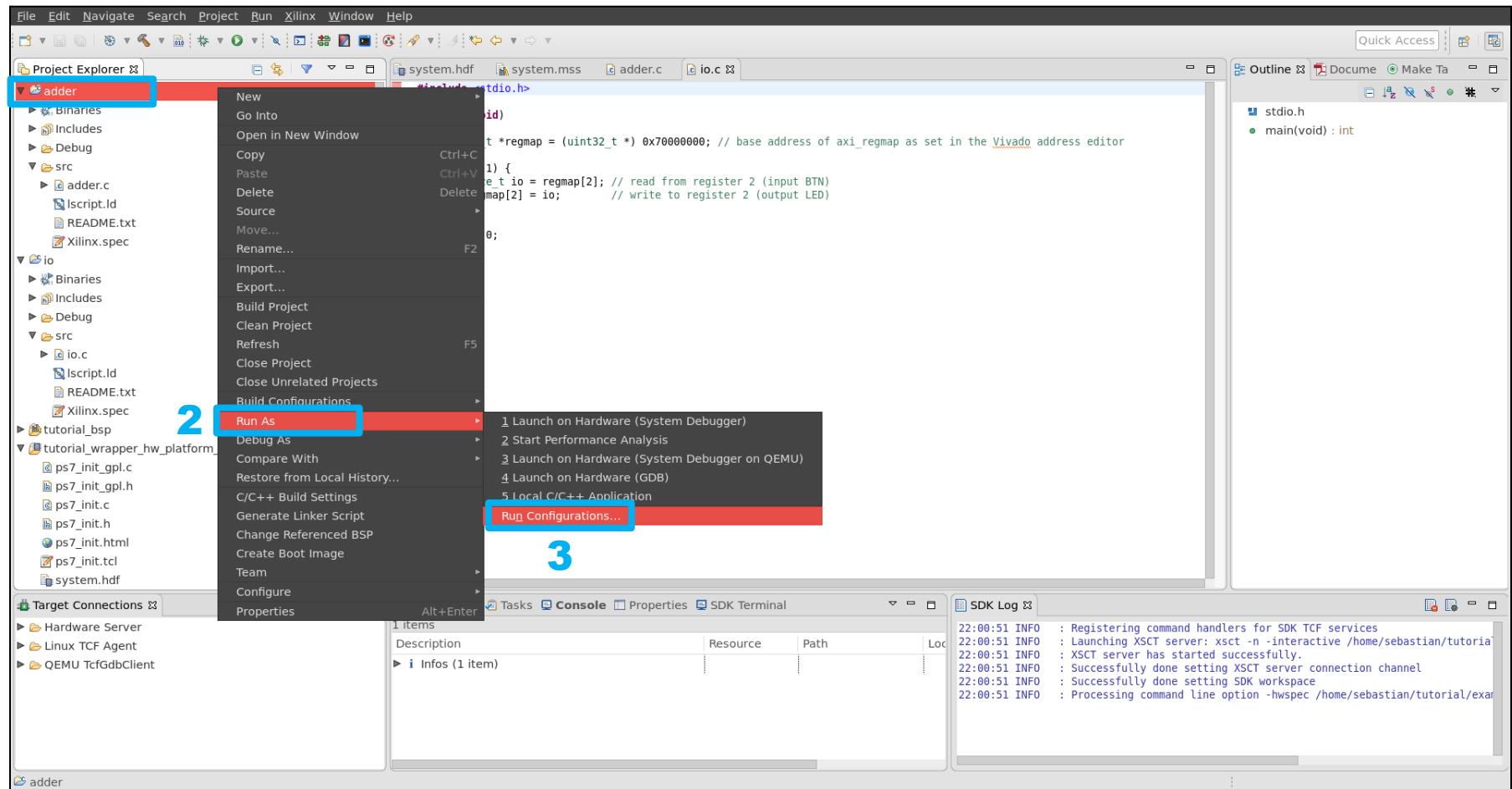


Part 2: Creating a Run Configuration

A run configuration is a profile for how an application project is programmed into the board. We will create one run configuration for each application.



Right-click the **adder** application project. Click **Run As**. Click **Run Configurations....**



Double-click Xilinx C/C++ application (System Debugger).

Create, manage, and run configurations



Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the ['Perspectives'](#) preference page.

Filter matched 5 of 8 items

- Performance Analysis
- Target Communication Framework
- Xilinx C/C++ Application (GDB)
- Xilinx C/C++ application (System Debugger on QEMU)
- Xilinx C/C++ application (System Debugger)

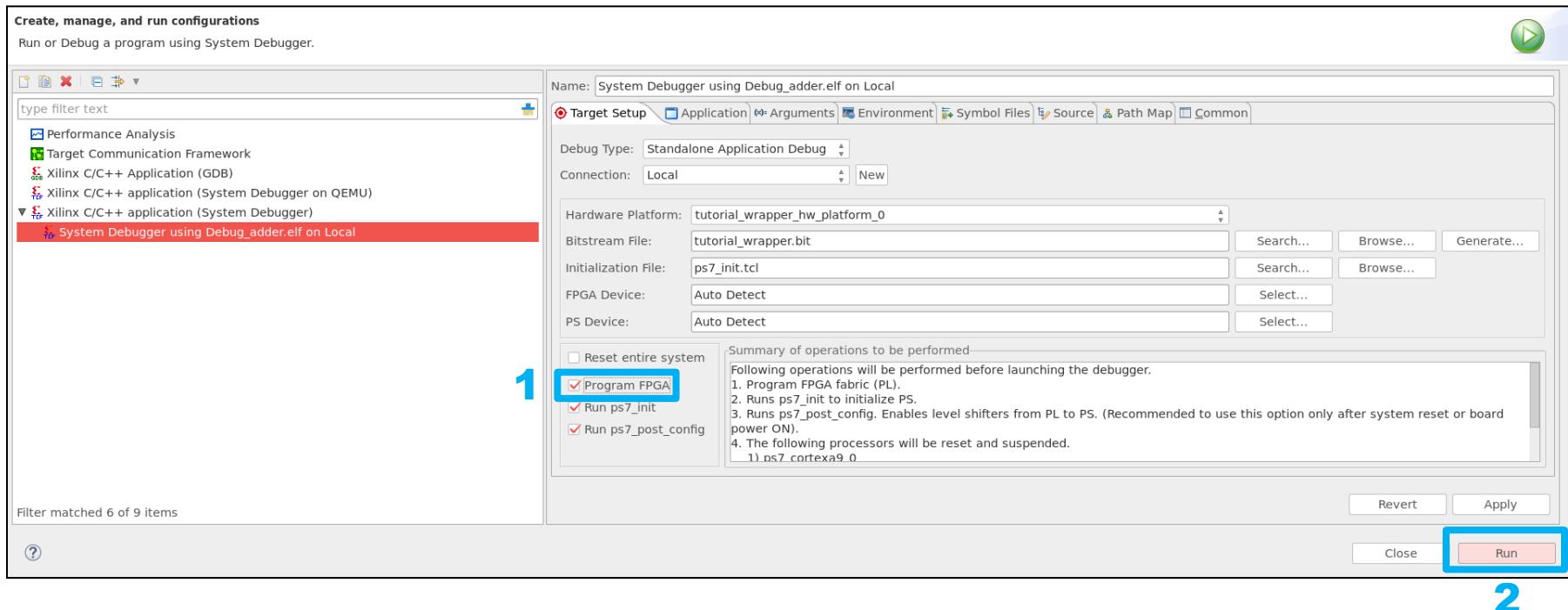


Close

Run



A profile is generated for the adder application. Check **Program FPGA**. Click **Run**.
NOTE: This will program the FPGA, then load and run the binary application.

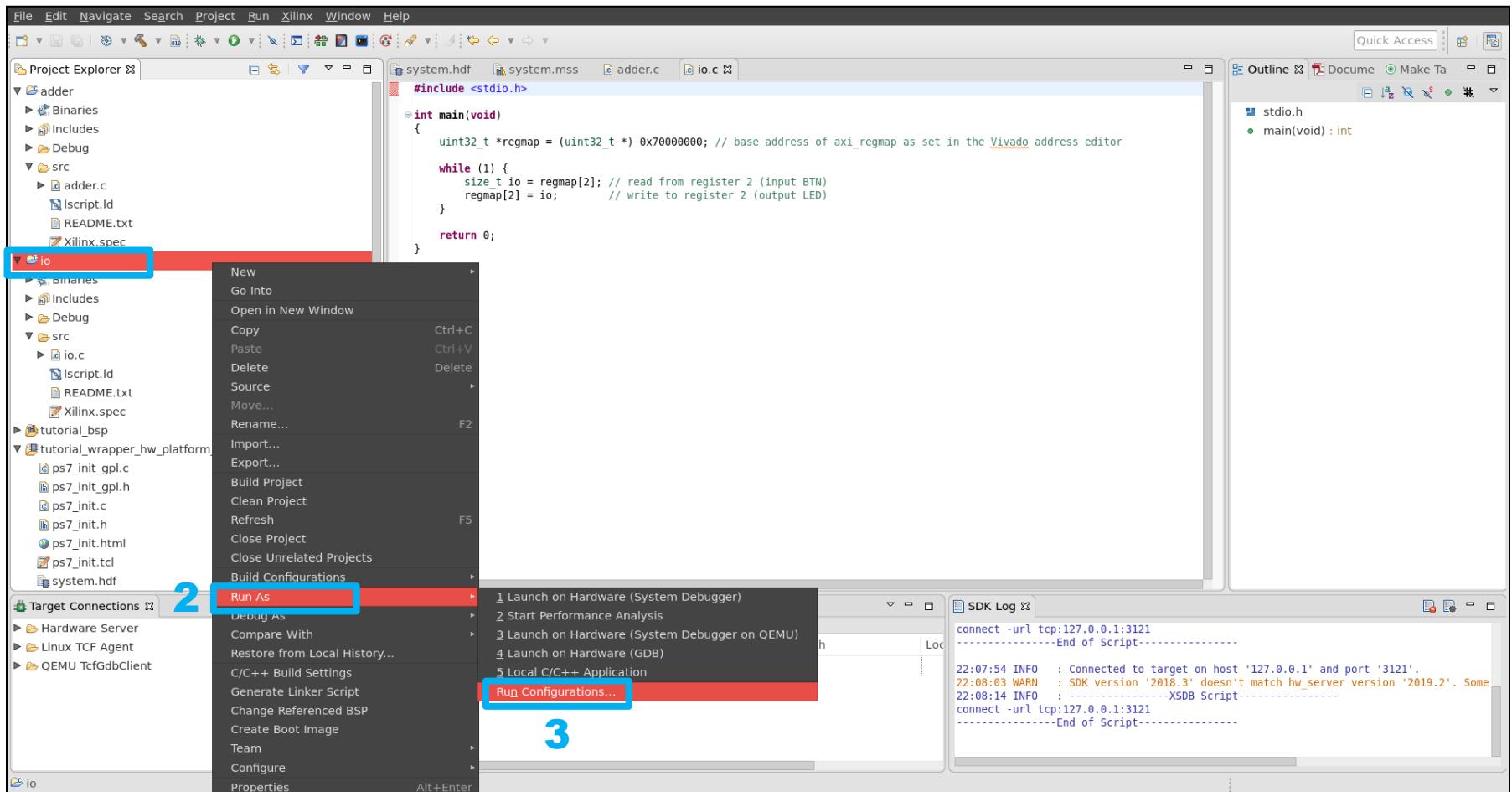


This is a snippet of the printout of the **adder** application. Continue.

```
0 +  0 =  0 (COR)
0 +  1 =  1 (COR)
0 +  2 =  2 (COR)
0 +  3 =  3 (COR)
0 +  4 =  4 (COR)
0 +  5 =  5 (COR)
0 +  6 =  6 (COR)
0 +  7 =  7 (COR)
0 +  8 =  8 (COR)
0 +  9 =  9 (COR)
0 + 10 = 10 (COR)
0 + 11 = 11 (COR)
0 + 12 = 12 (COR)
0 + 13 = 13 (COR)
0 + 14 = 14 (COR)
0 + 15 = 15 (COR)
1 +  0 =  1 (COR)
1 +  1 =  2 (COR)
1 +  2 =  3 (COR)
1 +  3 =  4 (COR)
1 +  4 =  5 (COR)
1 +  5 =  6 (COR)
1 +  6 =  7 (COR)
1 +  7 =  8 (COR)
1 +  8 =  9 (COR)
1 +  9 = 10 (COR)
```

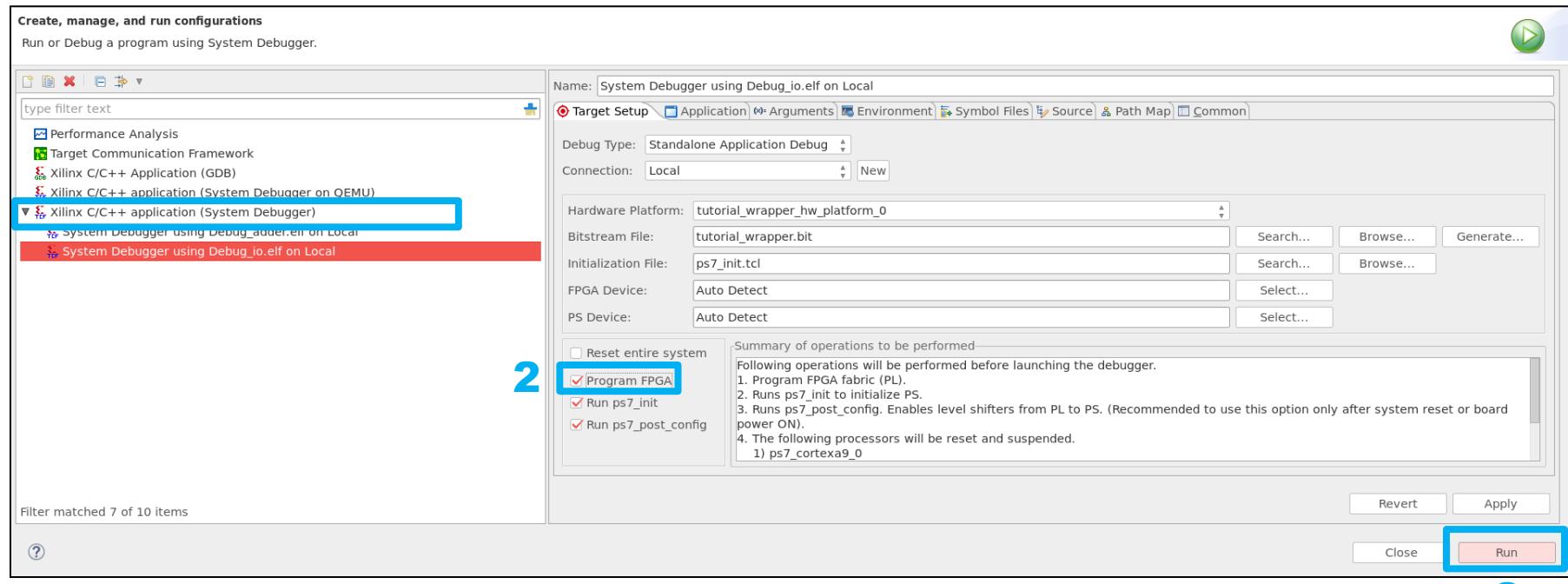


Right-click the **io** application project. Click **Run As**. Click **Run Configurations....**



Double-click Xilinx C/C++ application (System Debugger). A profile is generated for the adder application. Check **Program FPGA**. Click **Run**.

NOTE: This will program the FPGA, then load and run the binary application.



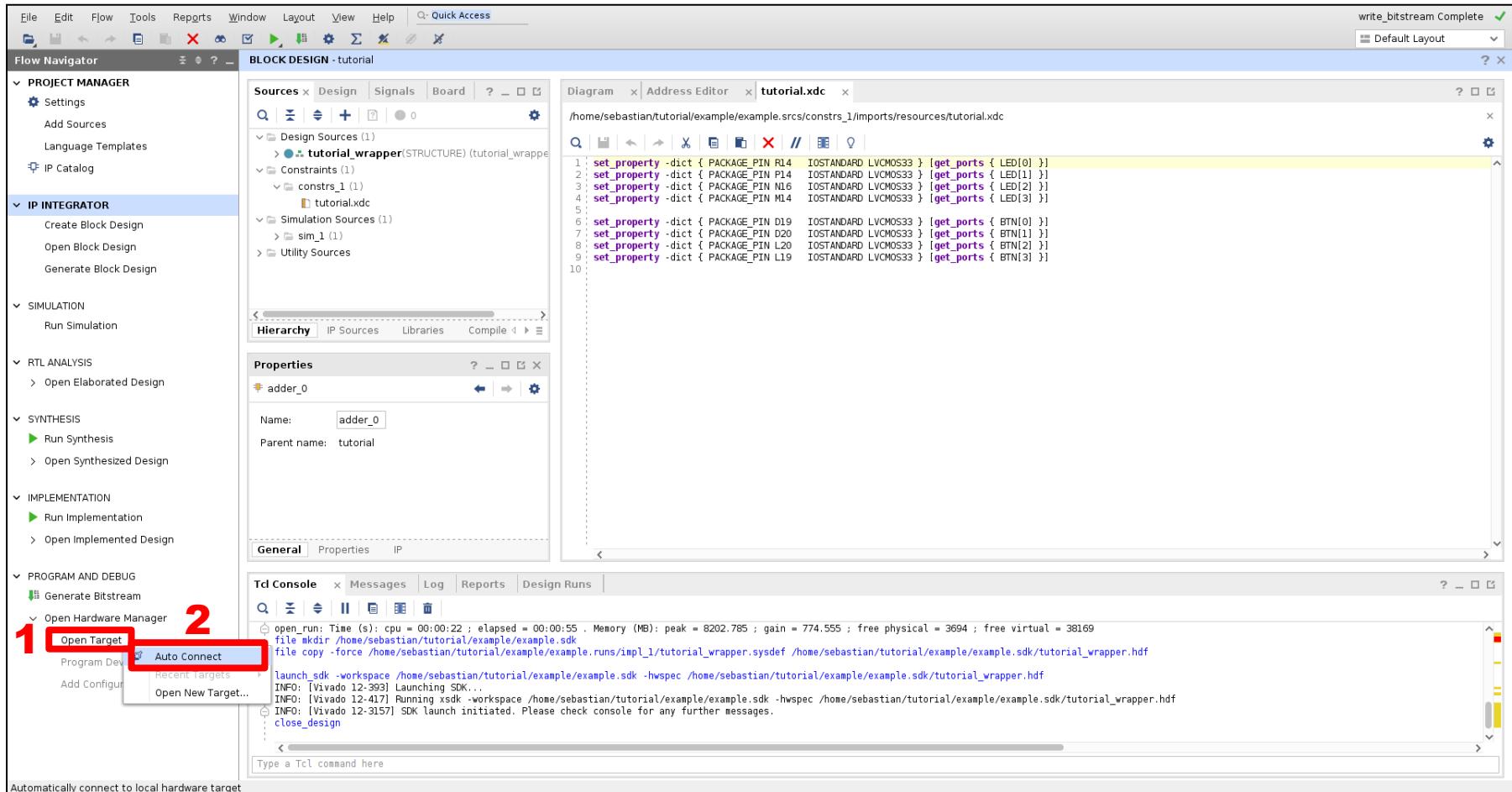
Part 3: Using the Integrated Logic Analyzer

The ILA is a powerful debugging tool that allows internal signals to be probed and analyzed at runtime.

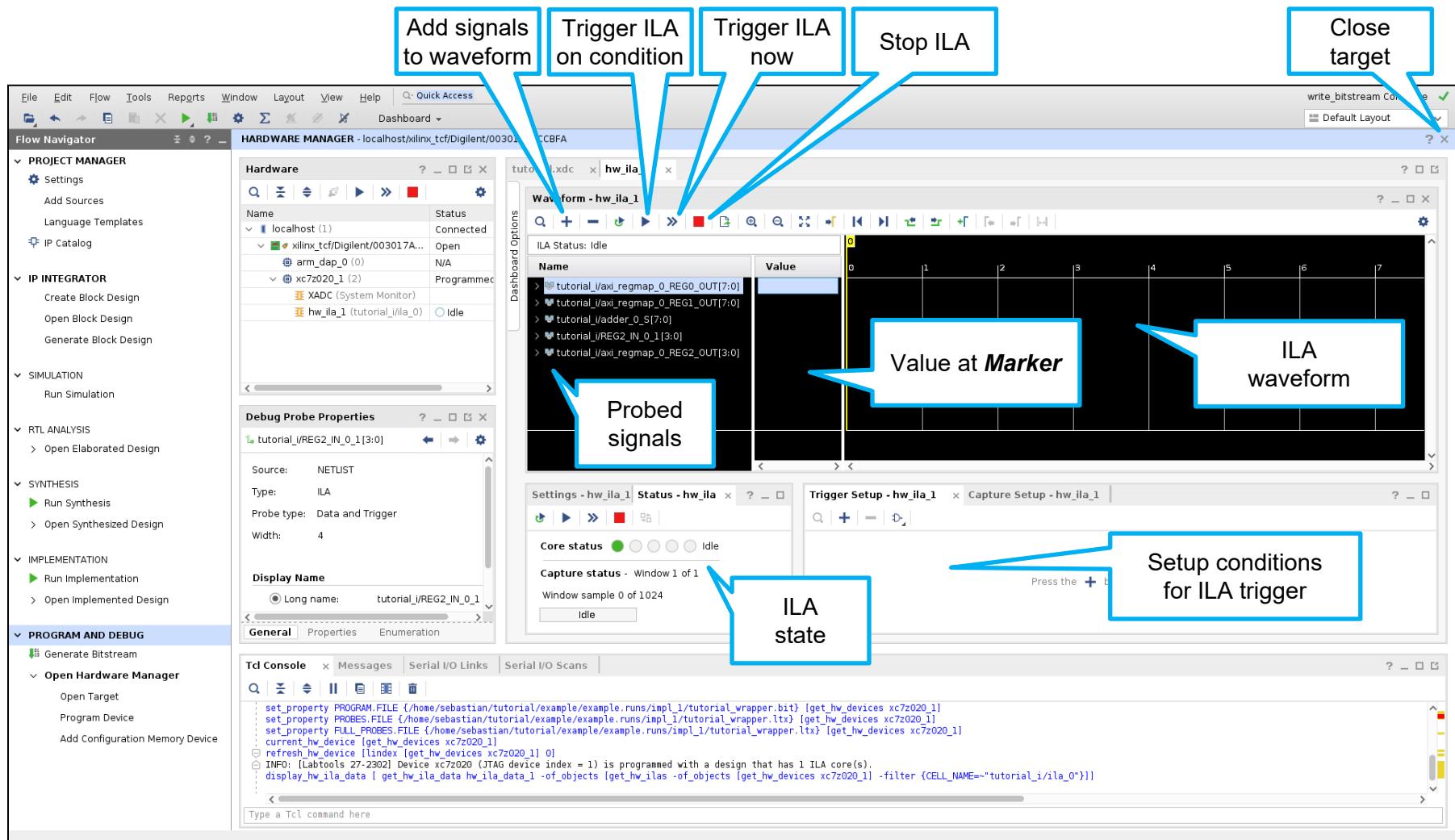
We will demonstrate the ILA for the **io** app (the app assumed to be running from the previous step).



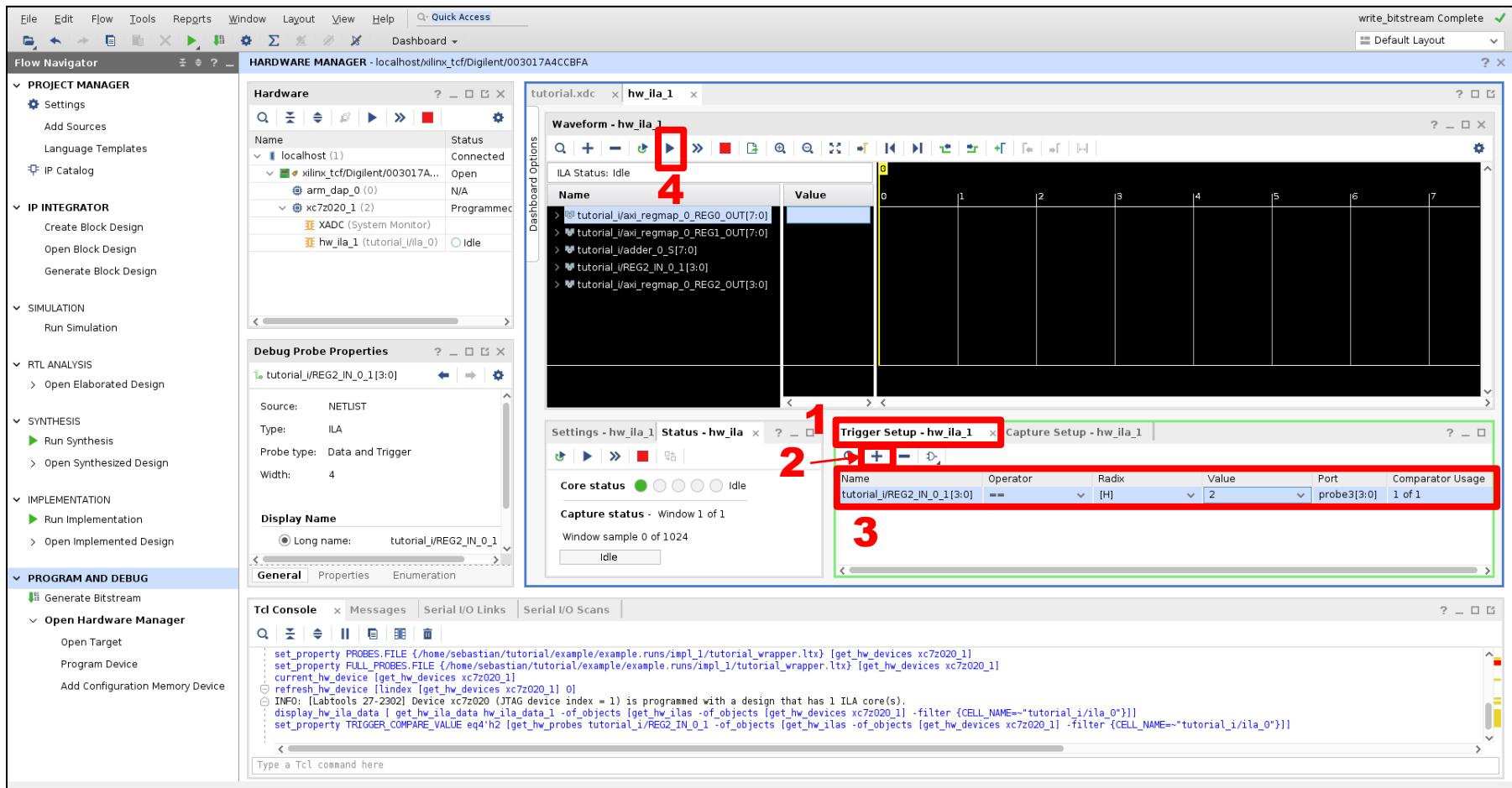
Go back to **Lab_1** project Instance. Click **Open Target**. Click **Auto Connect**.



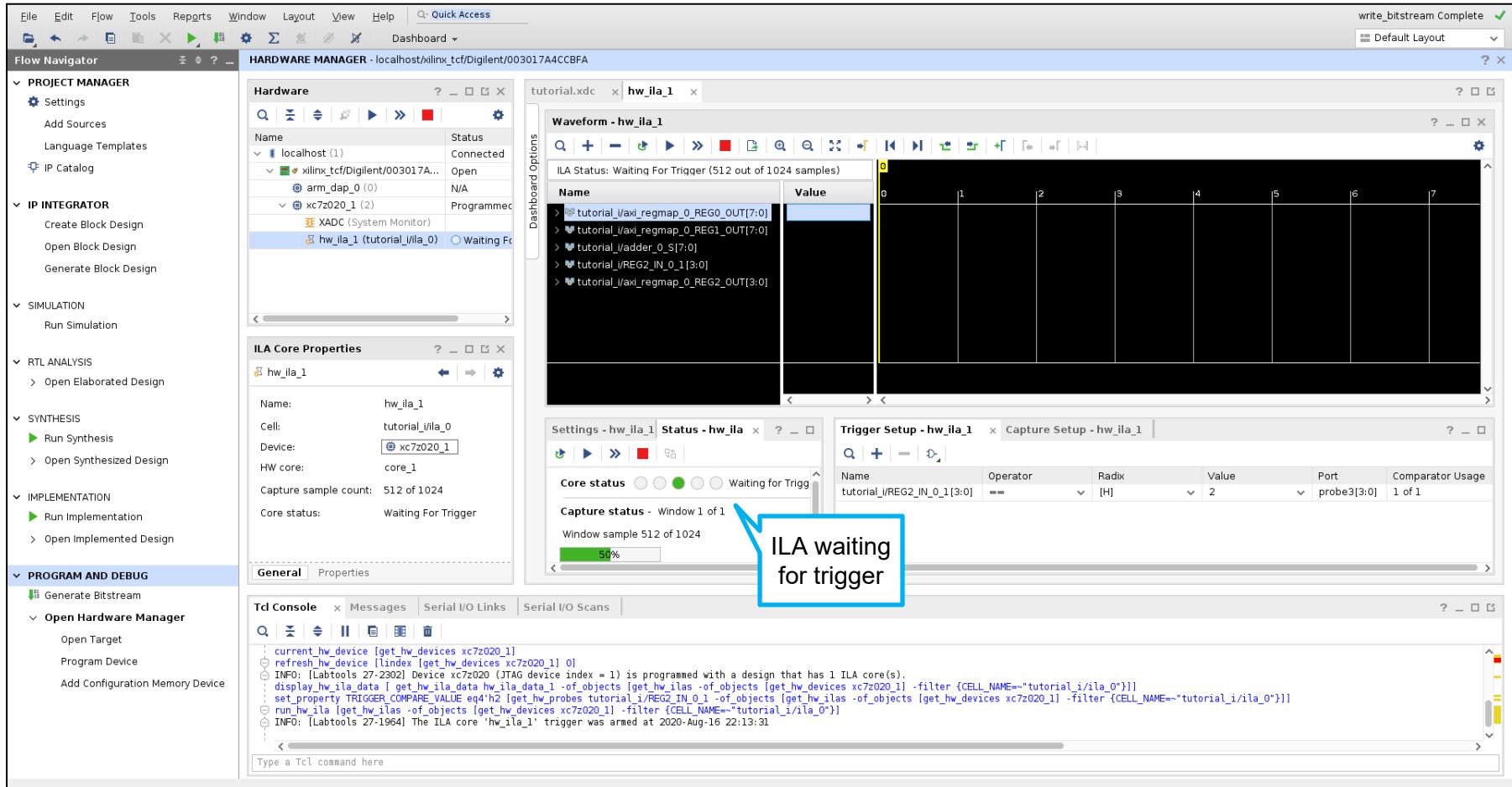
This is the ILA GUI. Continue.



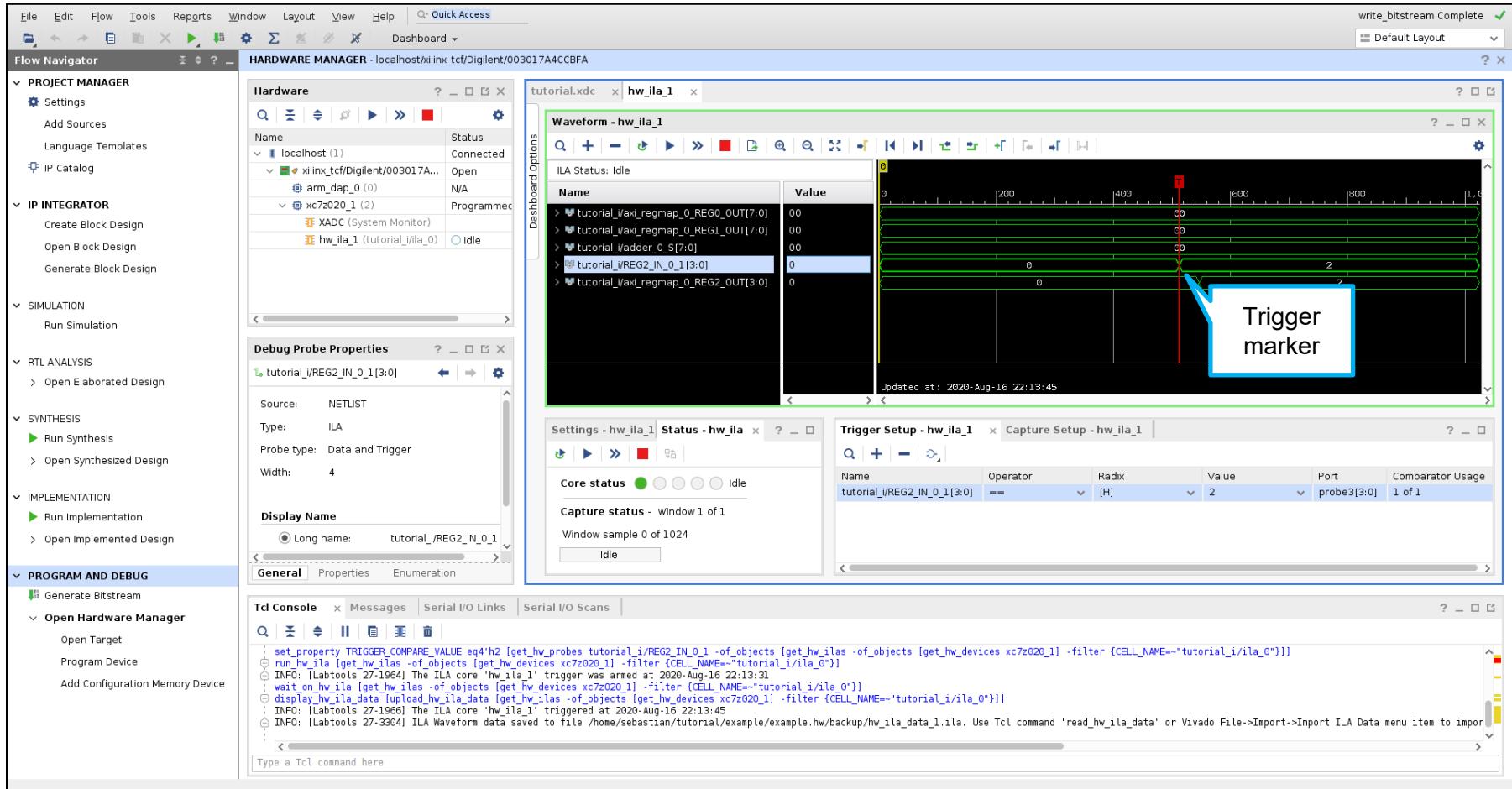
Click the **Trigger Setup – hw_il_1** tab. Click **+ (Add probe(s))** and add the net ***REG2_IN***. Set the trigger parameters **Operator → ==** and **Value → 2**. Click **▶**. This means that the ILA will start capturing the probed signals once the trigger condition happens.



ILA is waiting for trigger condition. Continue.



Set board **BTN[3:0] → 0010** (Value = 2) to trigger ILA. (Click on **BTN1**)



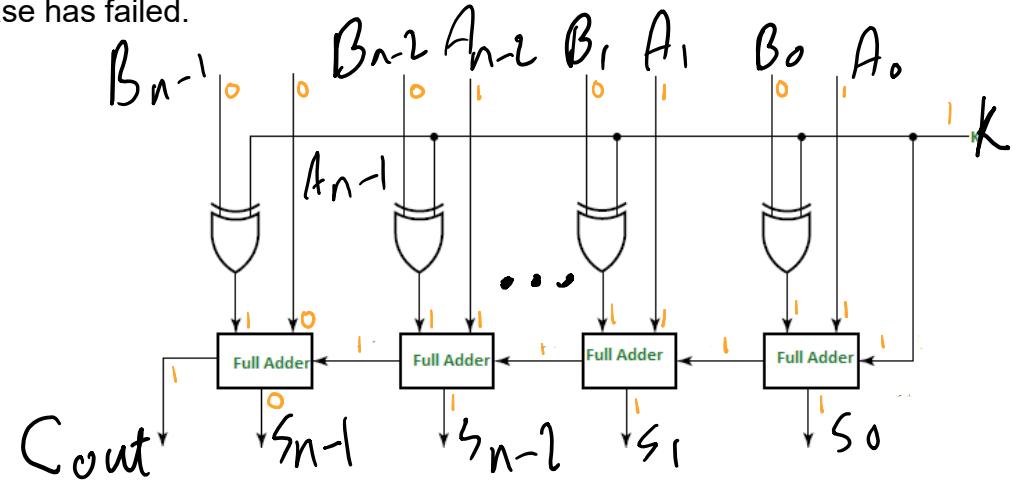
What to do

$$\begin{aligned} -0x80000000 &= 2^{n-1} \\ 0x7FFF\text{ FFFF} &= 2^{n-1}-1 \end{aligned}$$

- Design a generic 32 bit adder/subtractor using:
 - Generic statement
 - For-Generate statement
- Write a testbench to fully test your VHDL design, verify the functionality using the behavioral simulation. (refer to assert_tb.vhd)
- Write C code to fully test your design. Make sure to cover both the addition and subtraction cases. (use random numbers. Refer to this [blog](#) for more information)

Note: no need to print to the terminal here, as it will keep printing for a very long time. Instead, only print statement, when you pass all the test cases, or if any test case has failed.

$$\begin{array}{r} 01111111 \\ - 00000000 \\ \hline 01111111 \\ + 11111111 \\ \hline 01111111 \\ \text{Cout} = 1 \end{array}$$



<https://www.geeksforgeeks.org/4-bit-binary-adder-subtractor/>

Deliverables

- You will be delivering all the VHDL and C codes that you have written, including the design files, testbenches, and the C codes. The submission will on Canvas Assignment for Lab 1.
- If you Want, you can also just Zip the top Vivado project folder that has all the files that was mentioned above. (please make sure that it has all the files required).
- Check offs will be done on the due date for this assignment. The rubric for the check-offs will be provided later, but you should expect that we would:
 - Look at your design files and make sure that you used the required design method.
 - Check the VHDL testbench simulation.
 - Check the C code and make sure it has enough test cases.
 - Check that you can generate a bitstream and configure the design on the board.



End of Lab 1

