

# RISC-V picorv32 processor porting to KV260 + PYNQ

Luca Durante, Last update: August 6<sup>th</sup>, 2023

Host PC: Windows 10 Professional + Vivado 2022.1

KV260 board: UBUNTU 2022.04.2 + PYNQ 3.0.1

You can find all files related to this document at:

<https://github.com/luca-64/kv260-RISC-V-On-PYNQ>

I started from: <https://github.com/drichmond/RISC-V-On-PYNQ> based on PYNQ 2.1+ Vivado 2017.4.

This repository demonstrates a library for evaluating RISC-V Projects from the paper "Everyone's a Critic: A Tool for Evaluating RISC-V Projects":

[https://kastner.ucsd.edu/wp-content/uploads/2013/08/admin/fpl18-everyones\\_critic.pdf](https://kastner.ucsd.edu/wp-content/uploads/2013/08/admin/fpl18-everyones_critic.pdf)

But I found a more up to date fork from <https://github.com/drichmond/RISC-V-On-PYNQ>:

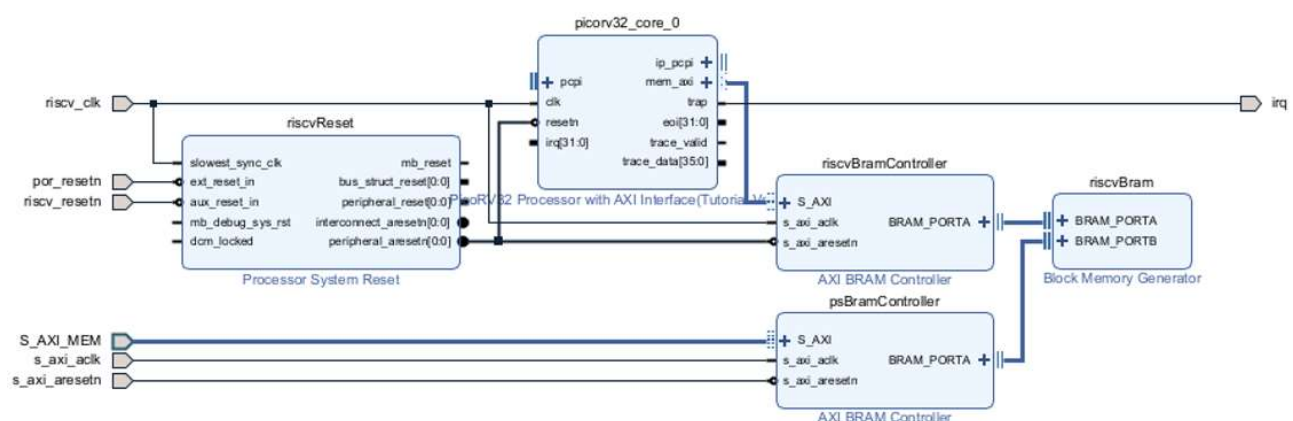
<https://github.com/Siudya/RISC-V-On-PYNQ>

I used the last one, based on PYNQ 2.4 Vivado 2018.3, as real starting point for the port to KV260 + PYNQ.

TCL file I manually modified with Vivado 2022.1 to obtain a working design for KV260 board:

<https://github.com/Siudya/RISC-V-On-PYNQ/blob/master/riscvonpyng/picorv32/tut/build.tcl>

It uses cliffordwolf:ip:picorv32\_axi:1.0 component (versus picorv32\_bram one):



I followed these steps:

- 1) Created a new directory C:\work\kv260-RISC-V-On-PYNQ\tutorial and copied <https://RISC-V-On-PYNQ/blob/master/riscvonpyng/picorv32/tut/build.tcl> in it. Renamed build.tcl in tutorial.tcl
- 2) Modified C:\work\kv260-RISC-V-On-PYNQ\tutorial\tutorial.tcl as follows:

```
#####
```

# Check if script is running in correct Vivado version.

#####

set scripts\_vivado\_version 2022.1

- 3) From Vivado TCL shell, recreated the original design based on the PYNQ-Z2 board:

```
C:\% Vivado 2022.1 Tcl Shell - C:\Xilinx\Vivado\2022.1\bin\vivado.bat -mode tcl
Vivado% cd c:/work/kv260-RISC-V-On-PYNQ/tutorial
Vivado%
```

cd c:/work/kv260-RISC-V-On-PYNQ/tutorial


source tutorial.tcl

From Vivado IDE, opened the project and the Block Diagram and manually updated the design as follows:

- 1) Replaced board part with the following one:

**Board Part**

Display name: Kria KV260 Vision AI Starter Kit  
Board part name: xilinx.com:kv260\_som\_som240\_1\_connector\_kv260\_carrier\_som240\_1\_connector:part0:1.3  
Board revision: Rev\_B01  
Connectors: Connector 1 on kv260 > Vision AI Starter Kit carrier card  
Repository path: C:/Xilinx/Vivado/2022.1/data/xhub/boards  
URL: [www.xilinx.com](http://www.xilinx.com)  
Board overview: Kria KV260 Vision AI starter Kit  
[Changes](#)



- 2) Replaced the Zynq-7000 SoC with the UltraSCALE+ MPSoC;

- 3) Added axi\_gpio\_0 for managing picorv32 reset and configured resetSlice for using axi\_gpio\_0 bit 0 output as riscv\_resetrn:

Re-customize IP

**Slice (1.0)**

Documentation IP Location

☐ Show disabled ports

Component Name: resetSlice

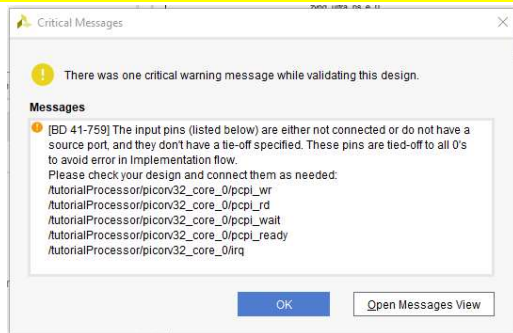
Din Width: 2 [2 - 4096]  
Din From: 0 [0 - 1]  
Din Down To: 0 [0 - 1]  
Dout Width: 1

OK Cancel

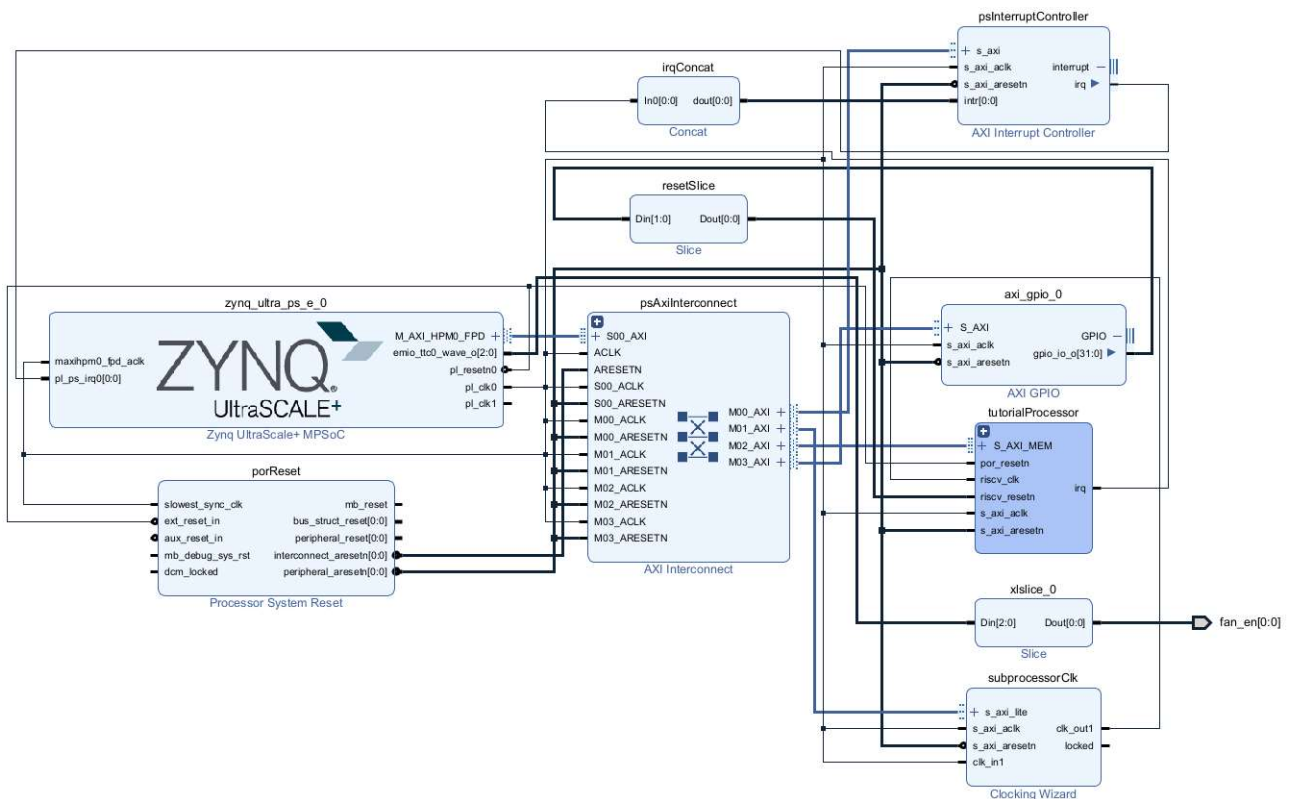
- 4) Updated memory addresses as follows:

Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/tutorialProcessor/picorv32_core_0					
/tutorialProcessor/picorv32_core_0/mem_axi (32 address bits : 4G)					
/tutorialProcessor/riscvBramController/S_AXI	S_AXI	Mem0	0x0000_0000	64K	0x0000_FFFF
Network 1					
/zynq_ultra_ps_e_0					
/zynq_ultra_ps_e_0/Data (40 address bits : 0x00A0000000 [ 256M ], 0x0400000000 [ 4G ], 0x1000000000 [ 224G ])					
/axi_gpio_0/S_AXI	S_AXI	Reg	0x00_A003_0000	64K	0x00_A003_FFFF
/psInterruptController/S_AXI	s_axi	Reg	0x00_A001_0000	64K	0x00_A001_FFFF
/subprocessorClk/s_axi_lite	s_axi_lite	Reg	0x00_A002_0000	64K	0x00_A002_FFFF
/tutorialProcessor/psBramController/S_AXI	S_AXI	Mem0	0x00_A000_0000	64K	0x00_A000_FFFF

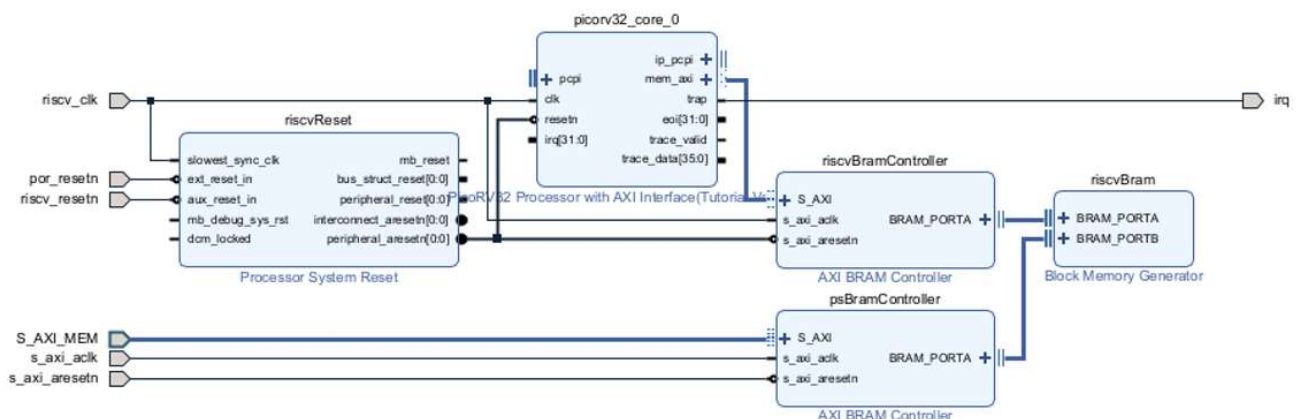
- 5) Added KV260 fan management (see constraints.xdc file and digital output fan\_en[0:0]) to reduce fan noise. Fan rotation speed is automatically managed by recent versions of KV260 UBUNTU OS;
- 6) Saved and validated the Design. Ignored the following warning messages:



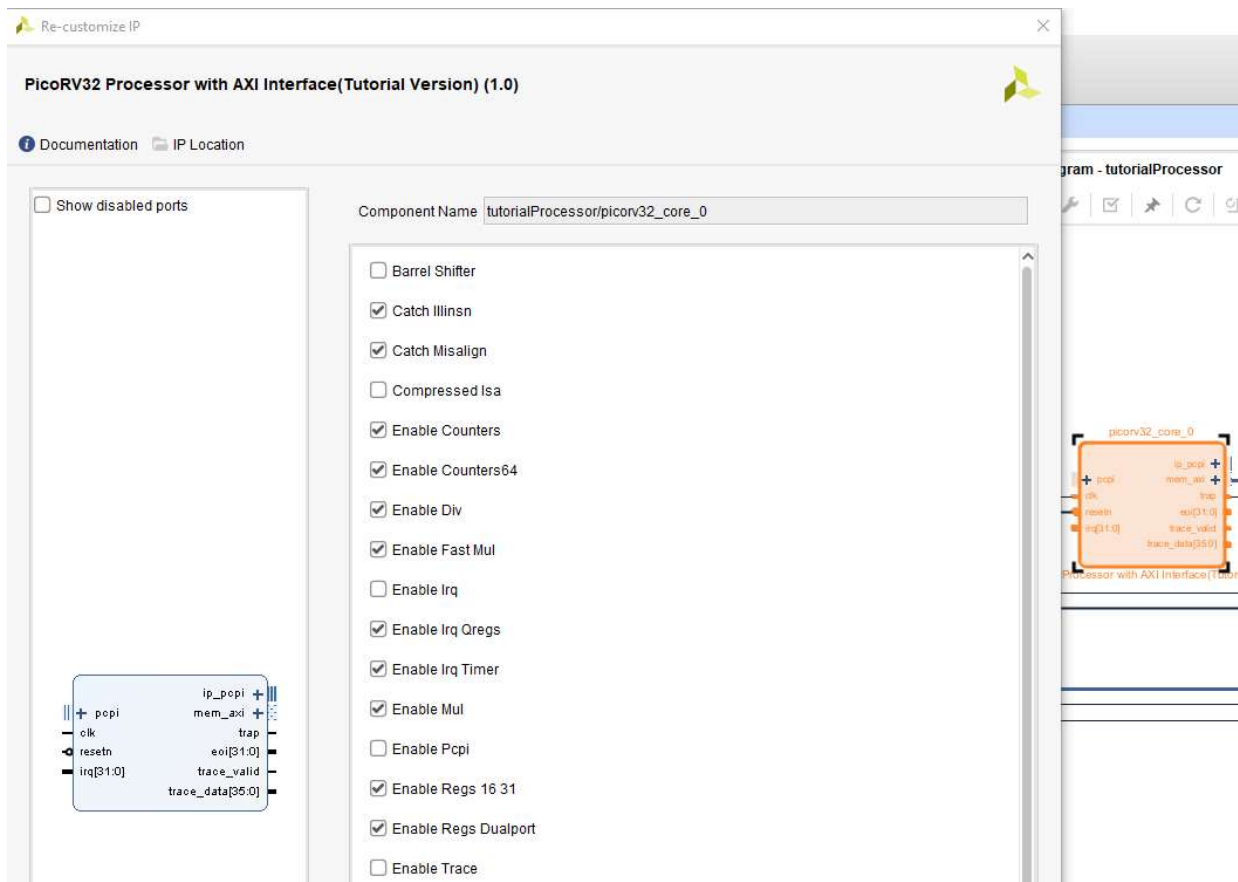
Here is the final design:



A last step: selected tutorialProcessor and, with a double click on it, opened its internal Diagram:



Selected PicoRV32 Processor, opened it with a double click on it and checked on Enable Div check box:



Generated Bitstream: Flow Navigator → Generate Bitstream:

- ▼ PROGRAM AND DEBUG
  - Generate Bitstream
  - > Open Hardware Manager

Ignored further warning windows.

Created the 3 files (tutorial.bit, tutorial.hwh and tutorial.tcl), necessary to run the design on KV260 board, with the following command from Vivado IDE Tcl Console:

```
source C:/work/kv260-RISC-V-On-PYNQ/tutorial/write_files_for_pynq.tcl
```

These files are stored in:

C:\work\kv260-RISC-V-On-PYNQ\tutorial

They are automatically overwritten if they already exist.

Just for reference - here is the content of the file: write\_files\_for\_pynq.tcl

```
cd C:/work/kv260-RISC-V-On-PYNQ/tutorial
```

```
file copy -force ./tutorial/tutorial.runs/impl_1/tutorial_wrapper.bit tutorial.bit
```

```
file copy -force ./tutorial/tutorial.gen/sources_1/bd/tutorial/hw_handoff/tutorial.hwh tutorial.hwh
```

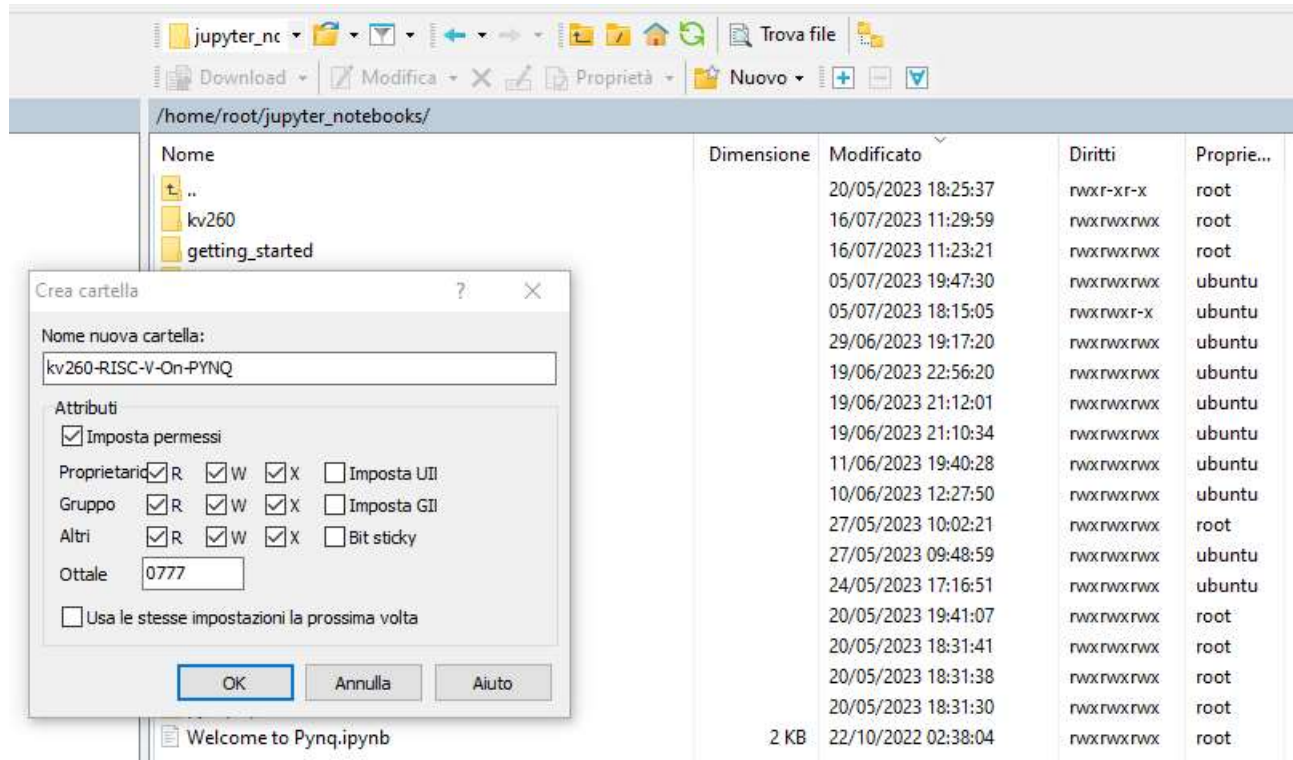
```
write_bd_tcl -force tutorial.tcl
```

Just for reference – I manually crated the Tutorial.ipynb Jupyter notebook stored in:

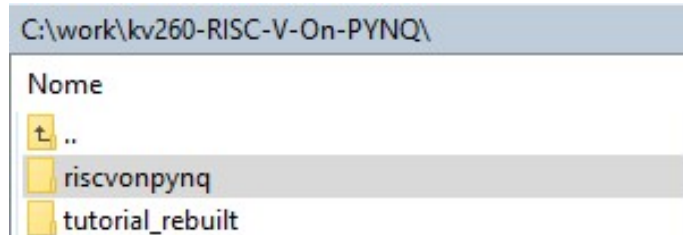
```
C:\work\kv260-RISC-V-On-PYNQ\riskonpynq\picorv32\tut
```

# Transfer files on KV260 board and run Tutorial.ipynb

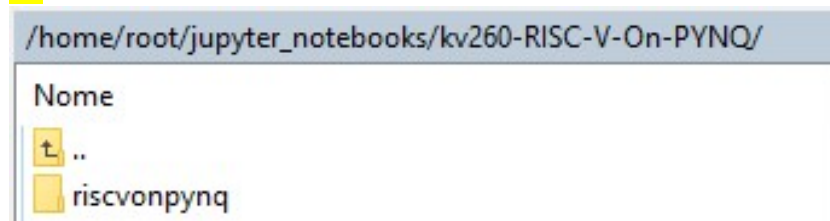
Using WinSCP, create the new directory kv260-RISC-V-On-PYNQ on KV260 board in /home/root/jupyter\_notebooks/ as follows:



Copy the directory riscvonpynq from:



to:





Copy the following files from:

C:\work\kv260-RISC-V-On-PYNQ\tutorial_rebuilt\			
Nome	Dimensi...	Tipo	Modificato
..		Cartella superiore	30/07/2023 10:48:11
tutorial		Cartella di file	30/07/2023 09:55:09
tutorial.tcl	27 KB	File TCL	30/07/2023 10:29:52
tutorial.bit	7.616 KB	File BIT	29/07/2023 16:02:45
tutorial.hwh	473 KB	File HWH	29/07/2023 15:22:04

to:

/home/root/jupyter_notebooks/kv260-RISC-V-On-PYNQ/riscvonpynq/picorv32/tut/		
Nome	Dimensione	Modificato
..		30/07/2023 11:09:05
build		30/07/2023 11:15:40
__pycache__		30/07/2023 11:09:03
Tutorial.ipynb	17 KB	30/07/2023 12:24:40
tutorial.tcl	27 KB	30/07/2023 10:29:52
tutorial.bit	7.616 KB	29/07/2023 16:02:45
tutorial.hwh	473 KB	29/07/2023 15:22:04
test.bin	1 KB	17/07/2023 16:07:54
test.elf	5 KB	17/07/2023 16:07:36
test.c	2 KB	17/07/2023 16:07:28
tutorial.py	3 KB	10/04/2020 07:29:05
__init__.py	1 KB	10/04/2020 07:29:05

Just for reference - I modified C:\work\kv260-RISC-V-On-PYNQ\riscvonpynq\Processor.py:

commented the 2 lines containing Xlnk references before transferring the project to KV260 board.

From PYNQ 2.7 Xlnk allocator and libraries for edge devices were removed, replaced by XRT allocation.

This was the reported error:

```
from tut.tutorial import TutorialOverlay
```

```
File /home/xilinx/RISC-V-On-PYNQ/riscvonpynq/Processor.py:36, in <module>
```

```
1 # -----
2 # Copyright (c) 2018, The Regents of the University of California All
3 # rights reserved.
(...)
33 # DAMAGE.
34 # -----
35 import pynq, os, enum, numpy as np, time
--> 36 from pynq import Xlnk
37 __author__ = "Dustin Richmond"
38 __copyright__ = "Copyright 2018, The Regents of the University of California"
```

```
ImportError: cannot import name 'Xlnk' from 'pynq' (/usr/local/share/pynq-venv/lib/python3.10/site-packages/pynq/__init__.py)
```

You can find more details on XRT allocation here:

[https://github.com/Xilinx/PYNQ\\_Workshop/blob/master/Session\\_4/4\\_basic\\_allocate\\_example.ipynb](https://github.com/Xilinx/PYNQ_Workshop/blob/master/Session_4/4_basic_allocate_example.ipynb)

# Run test.bin on picoRV32im

Open <http://kria:9090/lab> from Firefox web browser and open the Tutorial.ipynb notebook in kv260-RISC-V-On-PYNQ/riscvpyng/picorv32/tut:

File Edit View Run Kernel Tabs Settings Help

Filter files by name

/ ... / picorv32 / tut /

Name	Last Modified
build	a day ago
_init_.py	3 years ago
_test.bin	14 days ago
test.bin	a day ago
test.c	14 days ago
test.elf	14 days ago
tutorial.bit	a day ago
tutorial.hwh	a day ago
Tutorial.ipynb	a day ago
tutorial.py	3 years ago
tutorial.tcl	a day ago

## KV260 picoRV32 Risc-V Tutorial

```
[1]: import sys
import os
os.chdir("/home/root/jupyter_notebooks/kv260-RISC-V-On-PYNQ/riscvpyng/picorv32/tut")
sys.path.insert(0, '/home/root/jupyter_notebooks/kv260-RISC-V-On-PYNQ/riscvpyng/picorv32')
sys.path.insert(0, '/home/root/jupyter_notebooks/kv260-RISC-V-On-PYNQ')
from tut.tutorial import TutorialOverlay
overlay = TutorialOverlay("tutorial.bit")
```

Javascript Error: require is not defined  
Javascript Error: require is not defined  
Javascript Error: require is not defined

```
[2]: # Optional step
help(overlay)
```

Help on TutorialOverlay in module tut.tutorial:

<tut.tutorial.TutorialOverlay object>  
Default documentation for overlay tutorial.bit. The following attributes are available on this overlay:

IP Blocks  
-----  
psInterruptController : pynq.overlay.DefaultIP  
subprocessorClk : pynq.overlay.DefaultIP  
axi\_gpio\_0 : pynq.lib.axigpio.AxiGPIO  
zynq\_ultra\_ps\_e\_0 : pynq.overlay.DefaultIP

Run it step by step following instructions inside comments. Ignore Javascript Error: require is not defined.

If you see something similar to the following hardcopy (only Memory Index 1028 value can be different), everything is working well:

```
[10]: # Display memory locations written by the test program

# Execute this cell more than once to see Memory Index 1028 fast increments

# Memory Index 1029 is not written by the test program

print(f'Memory Index {1024:3}: {arr[1024]:#0{10}x}')
print(f'Memory Index {1025:3}: {arr[1025]:#0{10}x}')
print(f'Memory Index {1026:3}: {arr[1026]:#0{10}x}')
print(f'Memory Index {1027:3}: {arr[1027]:#0{10}x}')
print(f'Memory Index {1028:3}: {arr[1028]:#0{10}x}')
print(f'Memory Index {1029:3}: {arr[1029]:#0{10}x}')

Memory Index 1024: 0x000000ff
Memory Index 1025: 0x0000ffff
Memory Index 1026: 0x00ffffff
Memory Index 1027: 0xffffffff
Memory Index 1028: 0x0012a089
Memory Index 1029: 0x00000000
```



# Rebuild the design from: tutorial\_orig.tcl

Host PC: Windows 10 Professional + Vivado 2022.1 KV260 board: UBUNTU 2022.04.2 + PYNQ 3.0.1

If you want to completely rebuild the design, you can start from the TCL file: tutorial\_orig.tcl

Working directory: C:\work\kv260-RISC-V-On-PYNQ\tutorial\_rebuilt

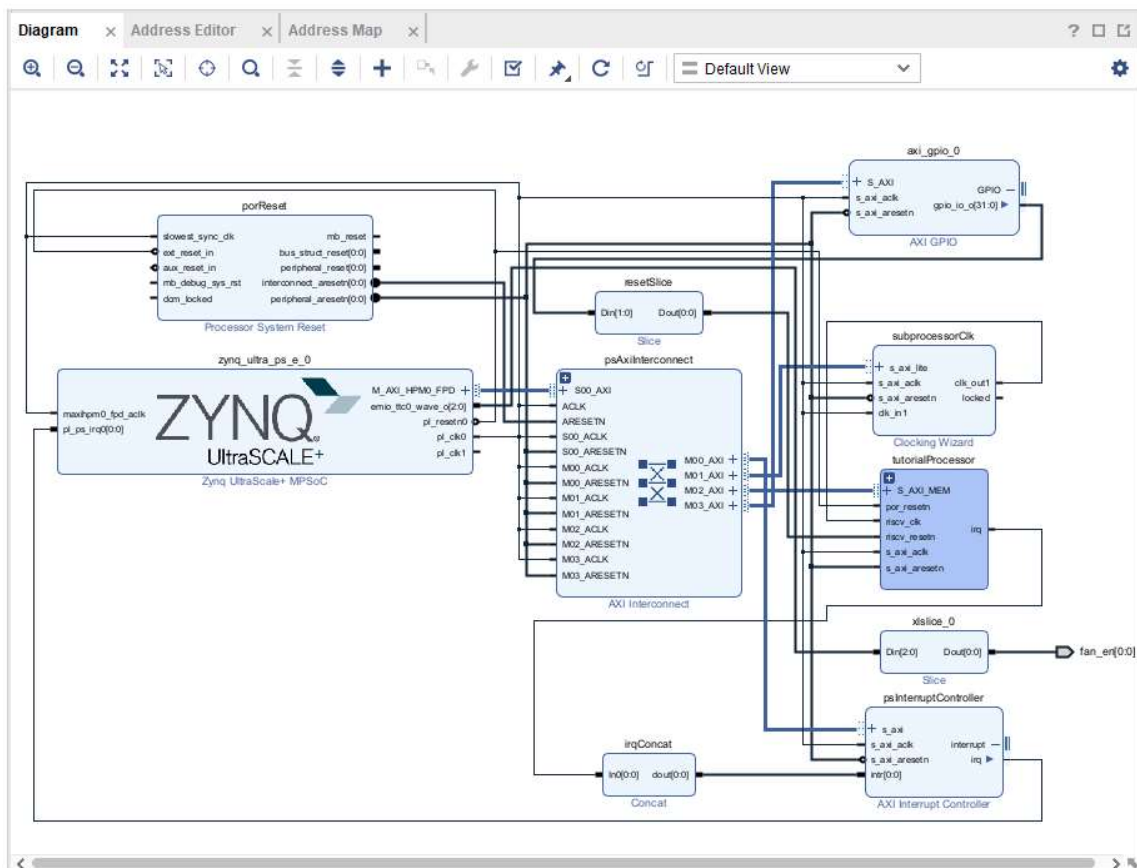
From Vivado IDE 2022.1 Tcl Console:



`cd /work/kv260-RISC-V-On-PYNQ/tutorial_rebuilt`

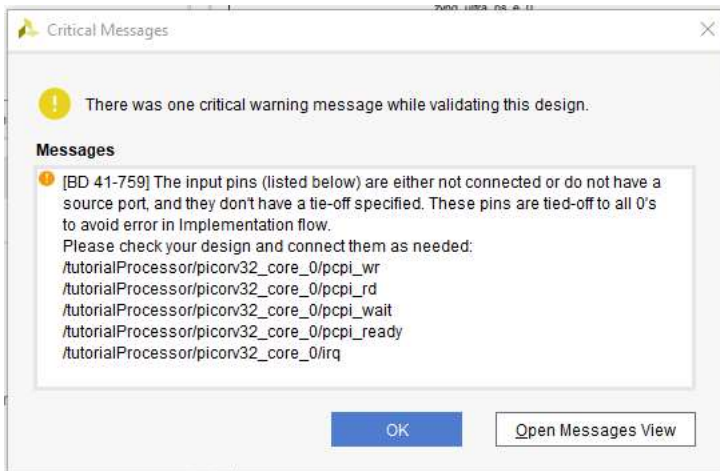
`source tutorial_orig.tcl`

This is the generated Block Diagram:

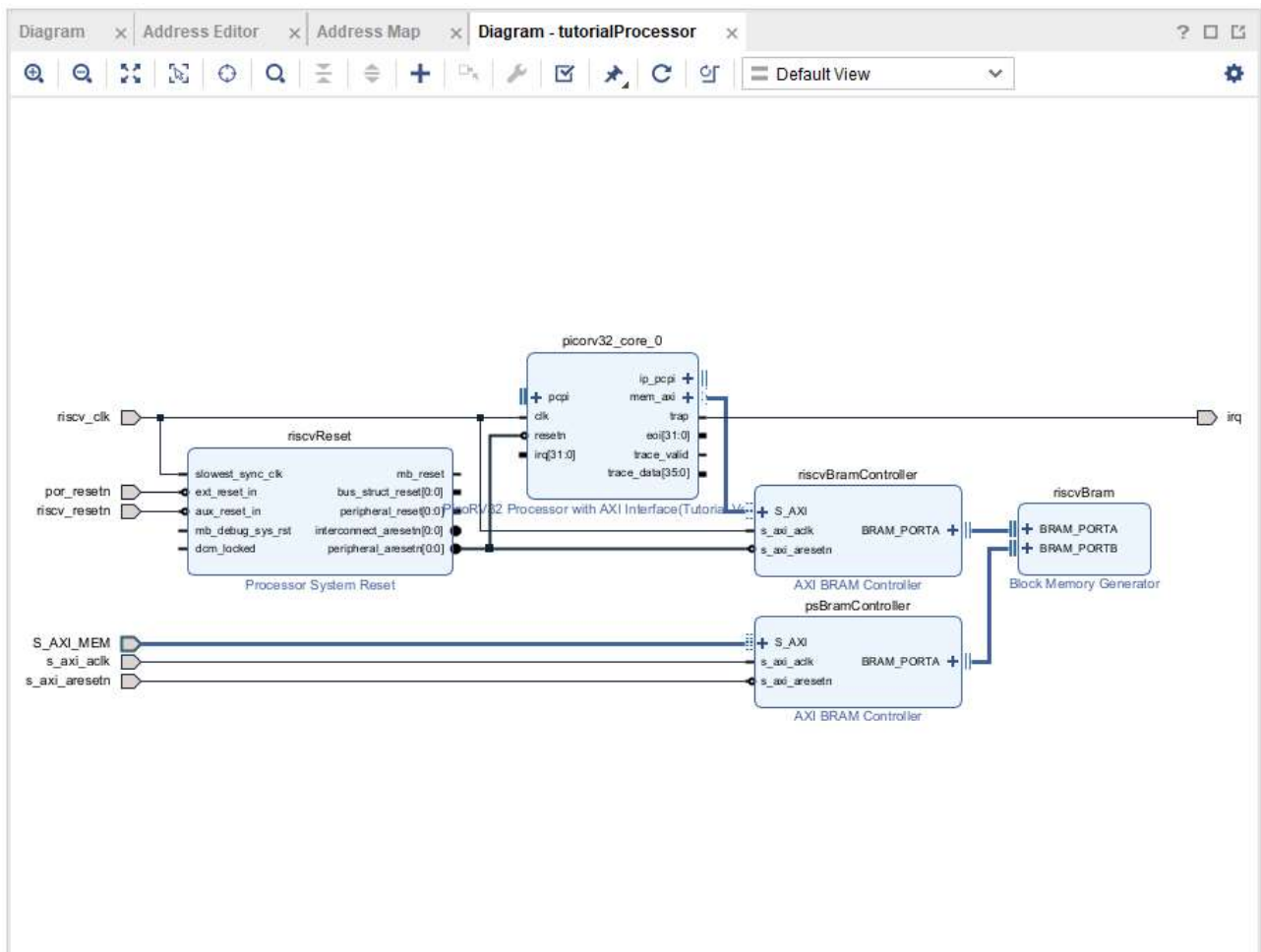


Right click inside a free space in Diagram and run the command: Validate Design.

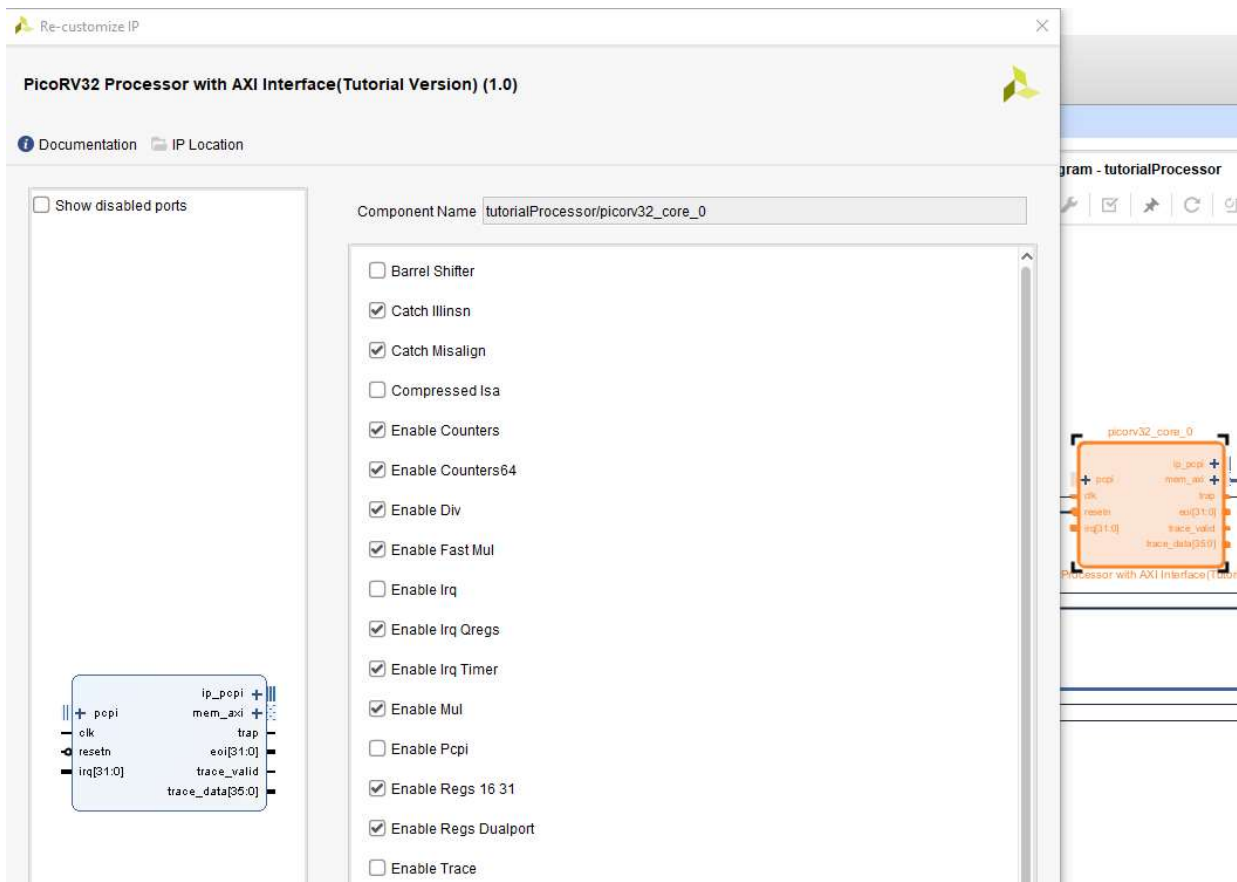
Ignore the following messages:



Select and double click on tutorialProcessor to open its Diagram:

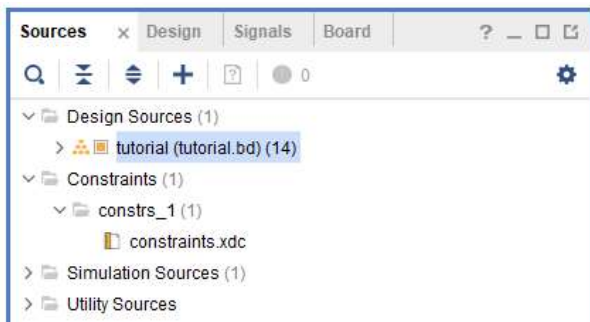


Select and double click on picoRV32 Processor to check or modify its configuration:

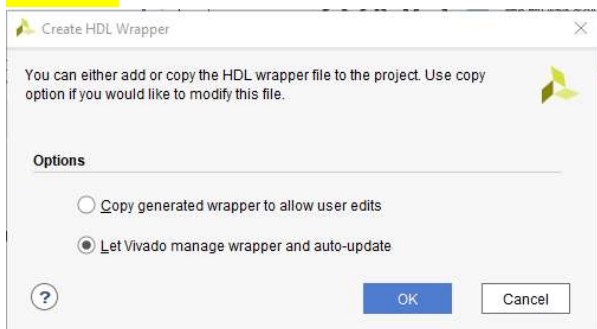


Check Enable Div box if not already checked.

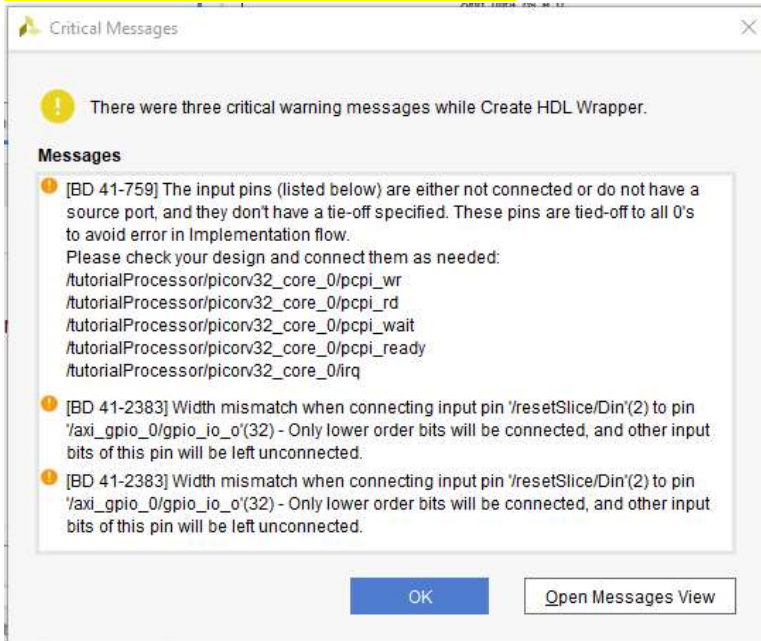
Select tutorial (tutorial.bd) and, with right mouse button, issue the command "Create HDL Wrapper..."



Press OK:



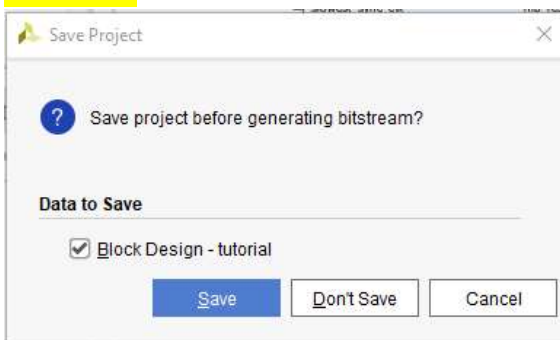
Press OK to ignore the following messages:



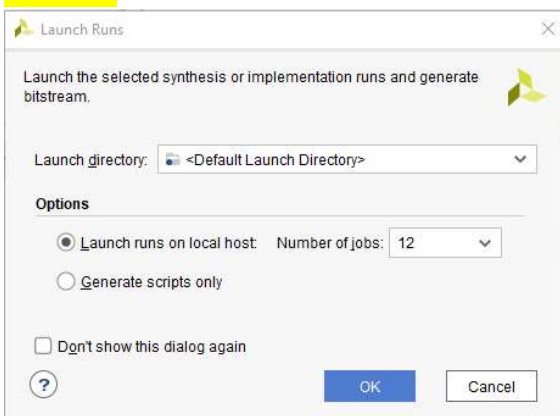
In Flow Navigator → Generate Bitstream:

- ▼ PROGRAM AND DEBUG
  - Generate Bitstream
  - > Open Hardware Manager

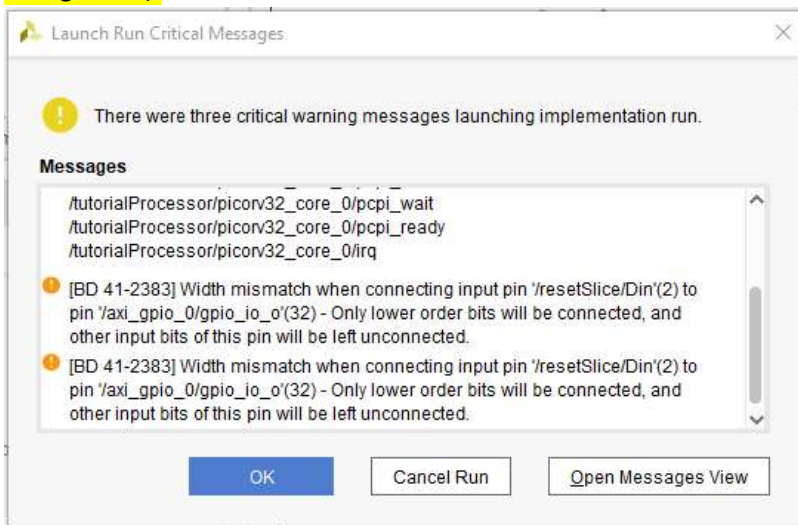
Press Save:



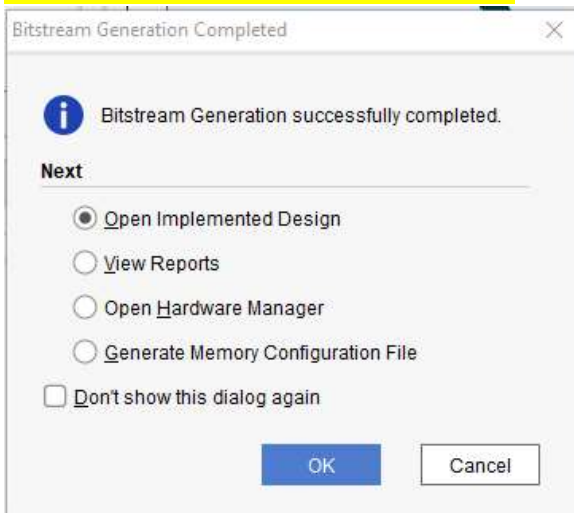
Press OK:



Press OK to ignore the following messages and wait for end of bitstream generation (it is running in background):



End of Bitstream Generation. Press Cancel:



Create the 3 files (tutorial.bit, tutorial.hwh and tutorial.tcl), necessary to run the design on KV260 board, with the following command from Vivado IDE Tcl Console:

```
source C:/work/kv260-RISC-V-On-PYNQ/tutorial_rebuilt/write_files_for_pynq.tcl
```

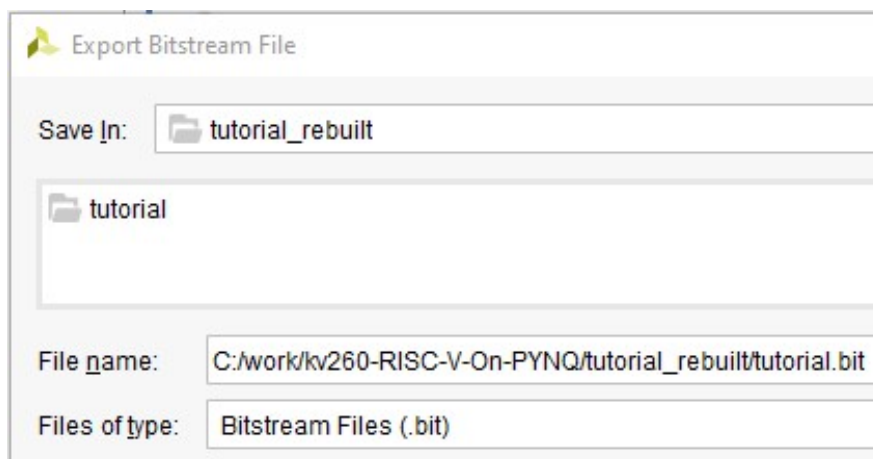
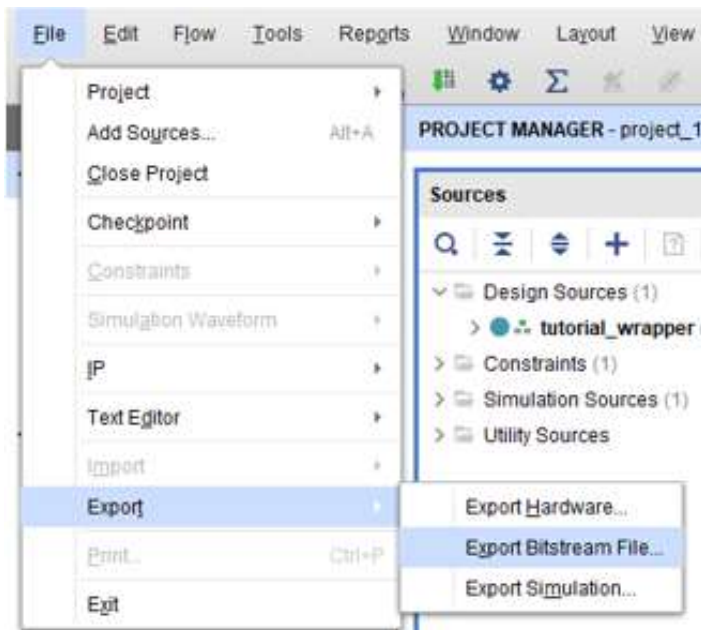
These files are stored in:

C:\work\kv260-RISC-V-On-PYNQ\tutorial\_rebuilt

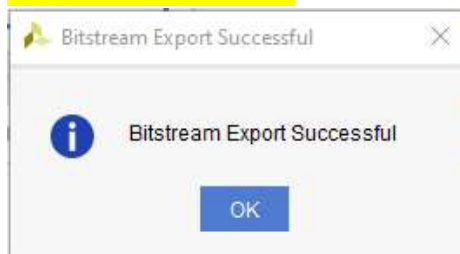
They are automatically overwritten if they already exist.

Just for reference – manual procedure for creating the 3 files to be transferred on KV260 board:

1) Export Bitstream File as tutorial.bit in C:\work\kv260-RISC-V-On-PYNQ\tutorial\_rebuilt:

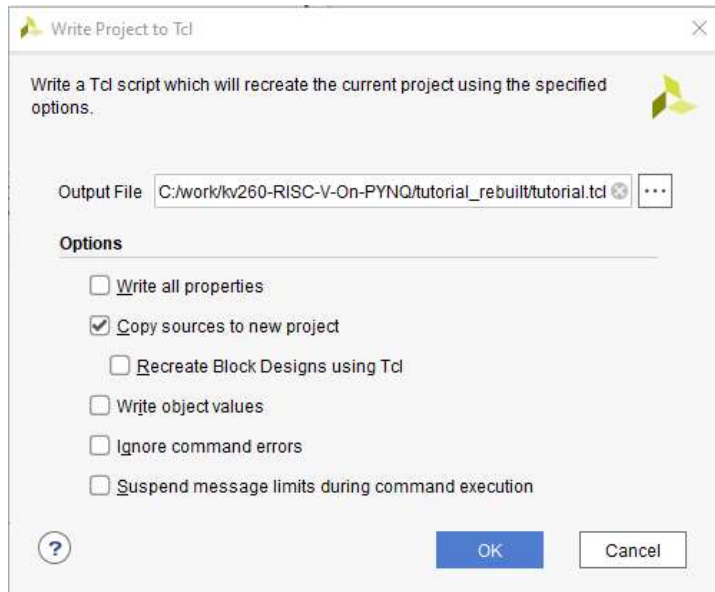


Press Save and then OK:

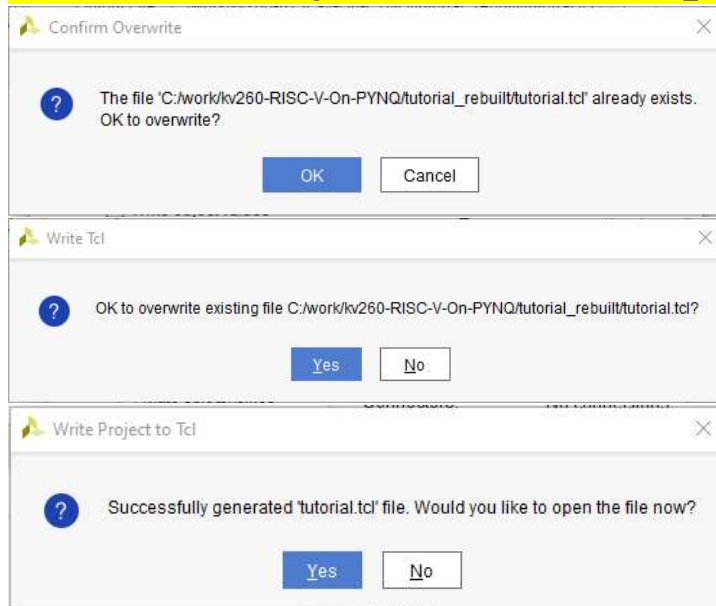




- 2) Export the TCL file as tutorial.tcl in C:\work\kv260-RISC-V-On-PYNQ\tutorial\_rebuilt:  
File → Project → Write Tcl...



Press OK, Yes, No (the original TCL file is saved in tutorial\_orig.tcl and is not overwritten):



- 3) Copy: C:\work\kv260-RISC-V-On-PYNQ\tutorial\_rebuilt\tutorial\project\_1.gen\sources\_1\bd\tutorial\hw\_handoff\tutorial.hwh  
to: C:\work\kv260-RISC-V-On-PYNQ\tutorial\_rebuilt\tutorial.hwh

Just for reference - this is the current Board Part. Connectors is configured as No connections:

Board Part	
Display name:	Kria KV260 Vision AI Starter Kit
Board part name:	xilinx.com:kv260_som:part0:1.3
Board revision:	Rev_B01
Connectors:	No connections
Repository path:	C:/Xilinx/Vivado/2022.1/data/xhub/boards
URL:	<a href="http://www.xilinx.com">www.xilinx.com</a>
Board overview:	Kria KV260 Vision AI starter Kit
<a href="#">Changes</a>	

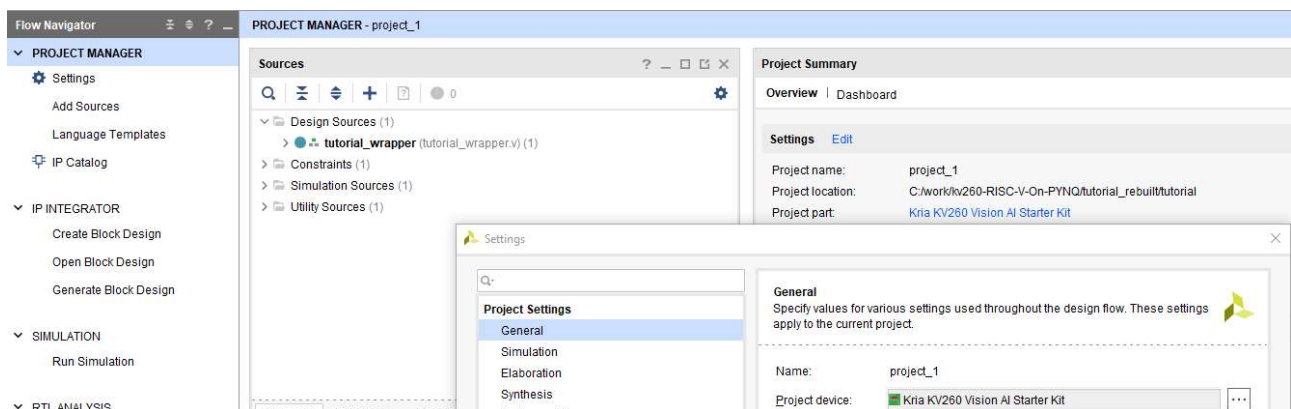


This was a workaround to solve the BOARD\_PART TCL managing error in Vivado 2022.1, due to the fact that the KV260 part name, with carrier, was not present into the repository or simply that its name is too long:

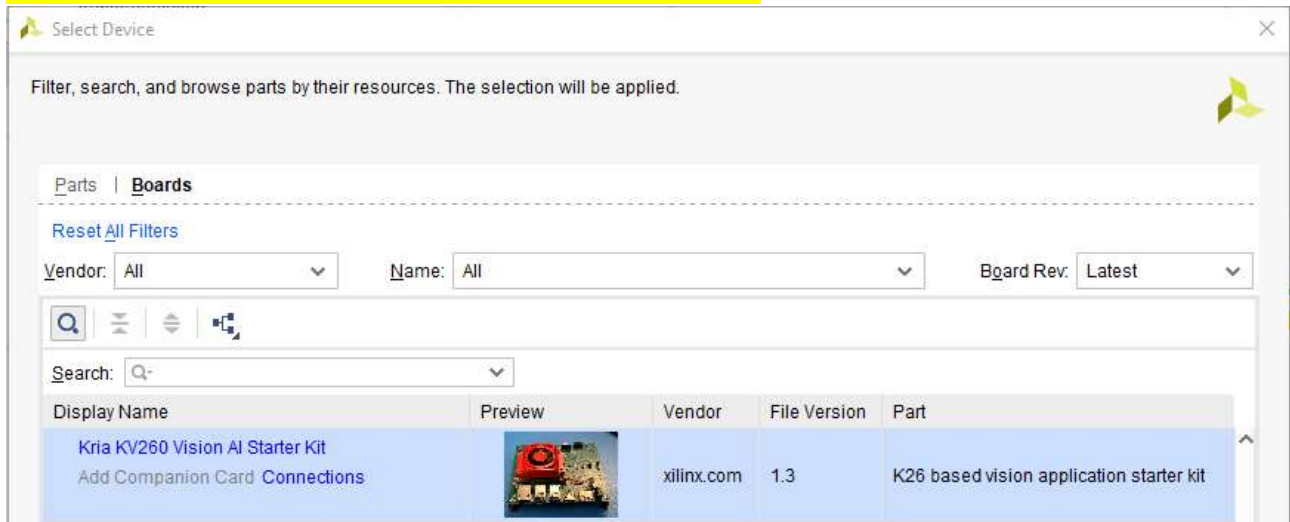
```
Tcl Console x Messages Log Reports Design Runs
# create_project project_1 myproj -part xck26-sfvc784-2LV-c
# set_property BOARD_PART xilinx.com:kv260_som_som240_1_connector_kv260_carrier_som240_1_connector:part0:1.3 [current_project]
#
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository 'C:/Xilinx/Vivado/2022.1/data/ip'.
ERROR: [Board 49-71] The board_part definition was not found for xilinx.com:kv260_som_som240_1_connector_kv260_carrier_som240_1_connector:part0:1.3.
Type a Tcl command here
```

You can manually change it as follows and then rebuild everything from Vivado:

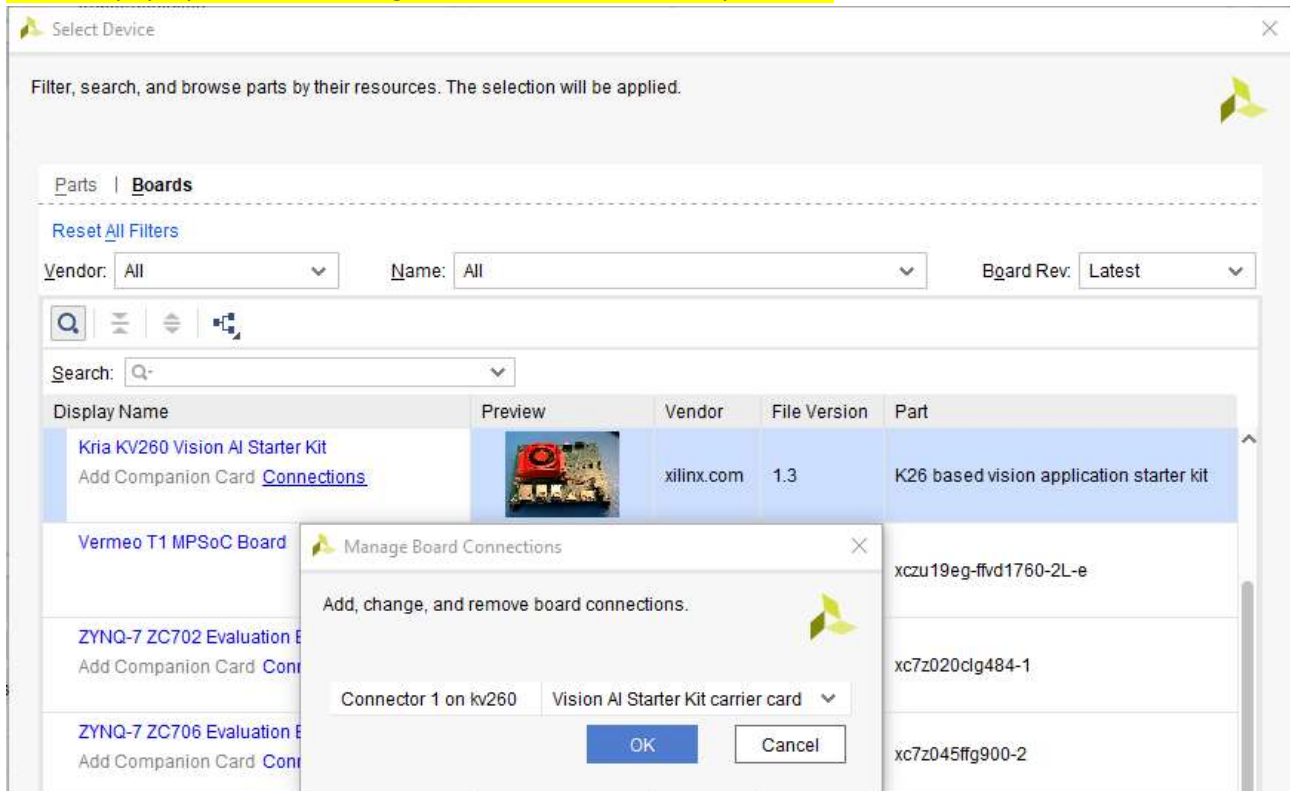
Double click on Kria KV260 Vision AI Starter Kit in the right Project Summary panel. In the settings window select ... on the right side of Kria KV260 Vision AI Starter Kit:



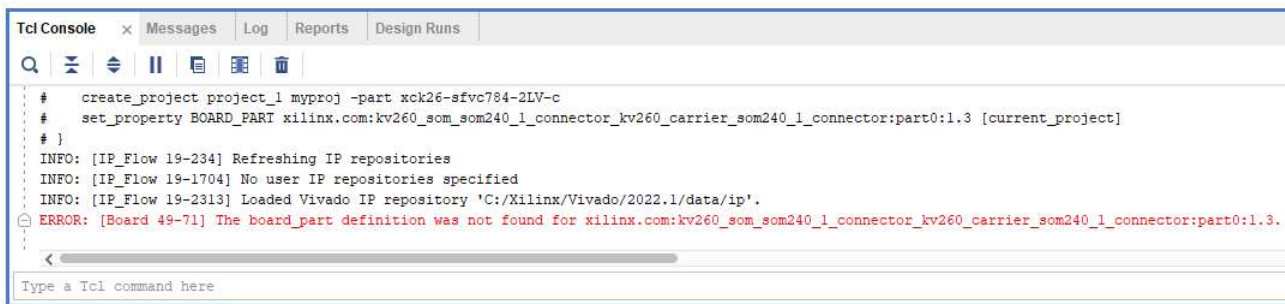
Select Kria KV260 Vision AI Starter Kit and click on Connections:



Select "Vision AI Starter Kit carrier card" from the drop-down menu on the right of "Connector 1 on kv260" inside the pop-up window "Manage Board Connections" and press OK:



Just for reference - steps to build tutorial\_orig.tcl file starting from a tested exported tutorial.tcl file.



```
Tcl Console x Messages Log Reports Design Runs
# create_project project_1 myproj -part xck26-sfvc784-2LV-c
# set_property BOARD_PART xilinx.com:kv260_som_som240_1_connector_kv260_carrier_som240_1_connector:part0:1.3 [current_project]
#
INFO: [IP_Flow 19-234] Refreshing IP repositories
INFO: [IP_Flow 19-1704] No user IP repositories specified
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository 'C:/Xilinx/Vivado/2022.1/data/ip'.
ERROR: [Board 49-71] The board_part definition was not found for xilinx.com:kv260_som_som240_1_connector_kv260_carrier_som240_1_connector:part0:1.3.
Type a Tcl command here
```

Copy a tested exported tutorial.tcl in C:\work\kv260-RISC-V-On-PYNQ\tutorial\_rebuilt\tutorial\_orig.tcl

Modify the following rows highlighted in green inside tutorial\_orig.tcl:

```
#####
# START
#####
```

# To test this script, run the following commands from Vivado Tcl console:

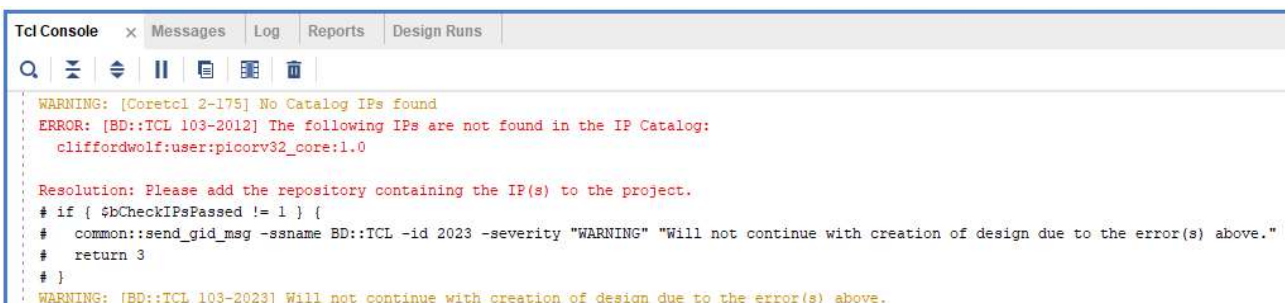
# source tutorial\_orig.tcl

# If there is no project opened, this script will create a  
# project, but make sure you do not have an existing project  
# in the current working folder.

```
set list_projs [get_projects -quiet]
if { $list_projs eq "" } {
    create_project tutorial tutorial -part xck26-sfvc784-2LV-c
    set_property BOARD_PART xilinx.com:kv260_som:part0:1.3 [current_project]
}
```

# CHANGE DESIGN NAME HERE

```
variable design_name
set design_name tutorial
```



```
Tcl Console x Messages Log Reports Design Runs
WARNING: [Coretool 2-175] No Catalog IPs found
ERROR: [BD::TCL 103-2012] The following IPs are not found in the IP Catalog:
    cliffordwolf:user:picorv32_core:1.0
Resolution: Please add the repository containing the IP(s) to the project.
# if { $bCheckIPsPassed != 1 } {
#     common::send_gid_msg -ssname BD::TCL -id 2023 -severity "WARNING" "Will not continue with creation of design due to the error(s) above."
#     return 3
# }
WARNING: [BD::TCL 103-2023] Will not continue with creation of design due to the error(s) above.
```

Add the following 4 rows, highlighted in green, inside tutorial\_orig.tcl before not highlighted ones:

```
set_property ip_repo_paths ../ip [current_project]
```

update\_ip\_catalog

add\_files -fileset constrs\_1 -norecurse constraints.xdc

import\_files -fileset constrs\_1 constraints.xdc

```
set bCheckIPsPassed 1
#####
# CHECK IPs
#####
set bCheckIPs 1
if { $bCheckIPs == 1 } {
    set list_check_ips "\
xilinx.com:ip:axi_gpio:2.0\
xilinx.com:ip:xlconcat:2.1\
xilinx.com:ip:proc_sys_reset:5.0\
xilinx.com:ip:axi_intc:4.1\
xilinx.com:ip:xlslice:1.0\
xilinx.com:ip:clk_wiz:6.0\
xilinx.com:ip:zynq_ultra_ps_e:3.4\
cliffordwolf:user:picorv32_core:1.0\
xilinx.com:ip:axi_bram_ctrl:4.1\
xilinx.com:ip:blk_mem_gen:8.4\
"
```

Build the design from Vivado IDE Tcl Console:

source tutorial\_orig.tcl

In case of error when building the design from Vivado IDE Tcl Console:

Close Project inside Vivado: File → Close project

Using Windows file explorer, delete the directory: C:\work\kv260-RISC-V-On-PYNQ\tutorial\_rebuilt\tutorial

Modify the tutorial\_orig.tcl file and rebuild the design from Vivado IDE Tcl Console:

source tutorial\_orig.tcl

# Install GNU RISC-V Compiler on KV260

Open a Linux terminal window on KV260 (point 1 or 2) or follow instructions from notebooks (point 3):

- 1) Using PuTTY serial console terminal from Windows 10;
- 2) Using KV260 Ubuntu GUI, if you have a monitor, a keyboard and a mouse connected to KV260;
- 3) Opening JupyterLab from Firefox web browser: <http://kria:9090/lab> and following instructions from notebooks: <https://github.com/drichmond/RISC-V-On-PYNQ/blob/master/notebooks/tutorial/1-Downloading-And-Configuring.ipynb> and <https://github.com/drichmond/RISC-V-On-PYNQ/blob/master/notebooks/tutorial/3-Compiling-RISC-V-GCC-Toolchain.ipynb>

The following instructions apply to Linux terminal window only (point 1 or 2).

Download dependencies:

```
sudo apt-get -y install autoconf automake autotools-dev curl libmpc-dev libmpfr-dev libgmp-dev gawk  
build-essential bison flex texinfo gperf libtool patchutils bc zlib1g-dev git
```

Download compiler source code:

```
git clone --recursive https://github.com/riscv/riscv-gnu-toolchain /home/xilinx/riscv-gnu-toolchain  
cd /home/xilinx/riscv-gnu-toolchain
```

Configure compiler source code for managing RISC-V 32im instruction set:

```
./configure --prefix=/opt/riscv32im --with-arch=rv32im
```

Build and install the compiler – approximative required time = 6 hours:

```
sudo make
```

The compiler has been automatically installed in /opt/riscv32im

Add /opt/riscv32im/bin to PATH in /etc/profile.d/pynq\_venv.sh:

```
sudo nano /etc/profile.d/pynq_venv.sh
```

Modify PATH as follows:

```
export PATH=$PATH:/usr/local/share/pynq-venv/bin/microblazeel-xilinx-elf/bin:/opt/riscv32im/bin
```

Save, reboot and verify that PATH has been updated correctly:

```
cat $PATH
```

(Ignore the message: No such file or directory)

Do a fast check to verify that the compiler has been installed correctly:

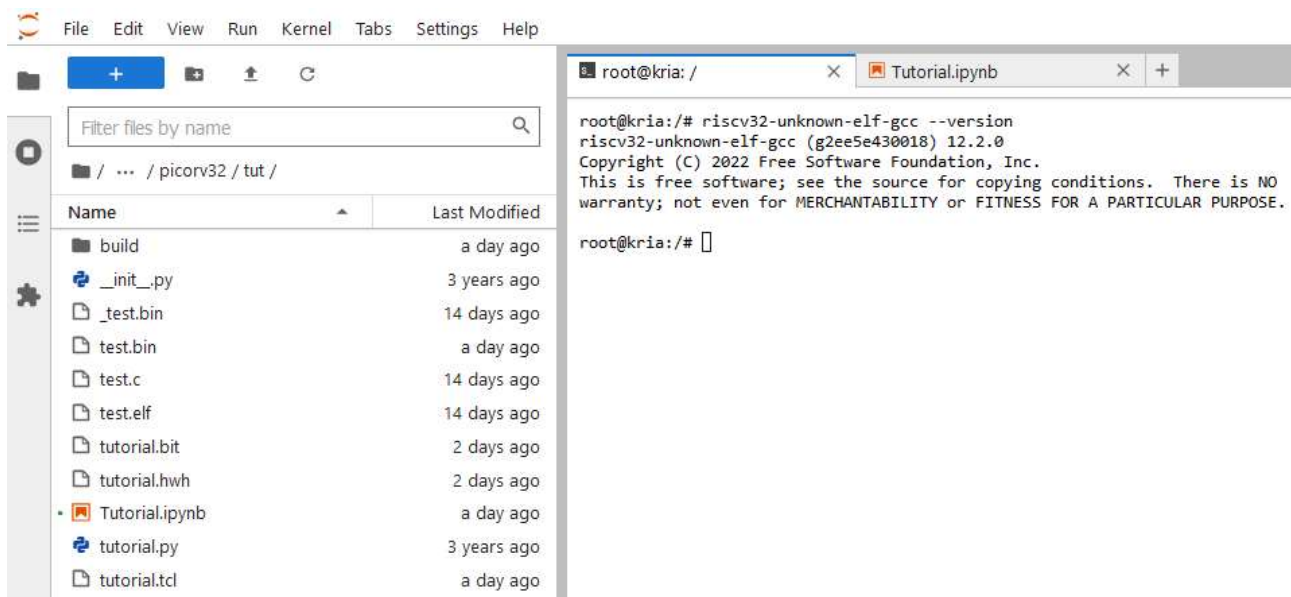
```
ubuntu@kria:~$ riscv32-unknown-elf-gcc --version
```

```
riscv32-unknown-elf-gcc (g2ee5e430018) 12.2.0
```

Copyright (C) 2022 Free Software Foundation, Inc.



Or with Firefox, <http://kria:9090/lab> using local root terminal:



## Compile test.c with RISC-V Compiler on KV260

Open <http://kria:9090/lab> from Firefox and go to the root Terminal tab.

Compile test.c to obtain the executable test.bin:

```
root@kria:/# cd /home/root/jupyter_notebooks/kv260-RISC-V-On-PYNQ/riscvonpynq
```

```
root@kria:/home/root/jupyter_notebooks/kv260-RISC-V-On-PYNQ/riscvonpynq/picorv32/tut/build# ls
__init__.py __pycache__ init.S init.o makefile picorv32.ld test.c test_prg.bin test_prg.c test_prg.elf
test_prg.o
```

```
root@kria:/home/root/jupyter_notebooks/kv260-RISC-V-On-PYNQ/riscvonpynq/picorv32/tut/build#
PATH=/opt/riscv32im/bin:$PATH
```

```
root@kria:/home/root/jupyter_notebooks/kv260-RISC-V-On-PYNQ/riscvonpynq/picorv32/tut/build# make
test.bin
```

Building object file test.o for test.c

```
riscv32-unknown-elf-gcc -c -Qn -march=rv32im -o test.o -Os --std=c99 test.c
```

Combining object files init.o to produce test.elf

```
riscv32-unknown-elf-gcc -Os -ffreestanding -nostdlib -o test.elf \
-Wl,-Bstatic,-T,picorv32.ld \
init.o test.o -lgcc -march=rv32im
```

```
/opt/riscv32im/lib/gcc/riscv32-unknown-elf/12.2.0/../../../../riscv32-unknown-elf/bin/ld: warning: test.elf
has a LOAD segment with RWX permissions
```

Converting .elf file test.elf into test.bin

```
riscv32-unknown-elf-objcopy -O binary test.elf test.bin
```

```
root@kria:/home/root/jupyter_notebooks/kv260-RISC-V-On-PYNQ/riscvonpynq/picorv32/tut/build#
```

Copy test.bin in /home/root/jupyter\_notebooks/kv260-RISC-V-On-PYNQ/riscvonnpyng/picorv32/tut and repeat 'Run test.bin on picoRV32im' steps.

test.c can be compiled inside a cell (see Tutorial.ipynb):

```
# Optional step
# Compile test.c program to test.bin
!pwd
!make test.bin
!ls -lh /home/root/jupyter_notebooks/kv260-RISC-V-On-PYNQ/riscvonnpyng/picorv32/tut

/home/root/jupyter_notebooks/kv260-RISC-V-On-PYNQ/riscvonnpyng/picorv32/tut
Building object file test.o for test.c
riscv32-unknown-elf-gcc -c -Qn -march=rv32im -o test.o -Os --std=c99 test.c
Combining object files init.o to produce test.elf
riscv32-unknown-elf-gcc -Os -ffreestanding -nostdlib -o test.elf \
    -Wl,-Bstatic,-T,picorv32.ld \
    init.o test.o -lgcc -march=rv32im
/opt/riscv32im/lib/gcc/riscv32-unknown-elf/12.2.0/../../../../riscv32-unknown-elf/bin/ld: warning: test.elf has a LOAD segment with RWX permissions
Converting .elf file test.elf into test.bin
riscv32-unknown-elf-objcopy -O binary test.elf test.bin
total 8.1M
-rw-rw-r-- 1 ubuntu ubuntu 22K Aug  2 14:12 Tutorial.ipynb
-rw-rw-r-- 1 ubuntu ubuntu  43 Apr 10 2020 __init__.py
drwxr-xr-x 2 root  root  4.0K Jul 30 09:09 __pycache__
drwxr-xr-x 4 ubuntu ubuntu 4.0K Aug  2 13:45 build
-rw-r--r-- 1 root  root  2.2K Aug  2 14:02 init.S
-rw-r--r-- 1 root  root  864 Aug  2 14:04 init.o
-rw-r--r-- 1 root  root  988 Aug  2 14:01 makefile
-rw-r--r-- 1 root  root  2.0K Aug  2 14:02 picorv32.ld
-rwxr-xr-x 1 root  root  322 Aug  2 14:14 test.bin
-rw-rw-r-- 1 ubuntu ubuntu 1.2K Jul 17 14:07 test.c
-rwxr-xr-x 1 root  root  4.8K Aug  2 14:14 test.elf
-rw-r--r-- 1 root  root  904 Aug  2 14:14 test.o
-rw-rw-r-- 1 ubuntu ubuntu 7.5M Jul 30 17:21 tutorial.bit
-rw-rw-r-- 1 ubuntu ubuntu 468K Jul 30 17:16 tutorial.hwh
-rw-rw-r-- 1 ubuntu ubuntu 2.3K Apr 10 2020 tutorial.py
-rw-rw-r-- 1 ubuntu ubuntu  63K Jul 30 17:24 tutorial.tcl
```

# Compile and simulate the test.c program on a Windows Host using Ripes Risc-V Simulator

Risc-V Simulator - Ripes-v2.2.6-win-x86\_64.zip

<https://github.com/mortbopet/Ripes/releases>

Risc-V Compiler: <https://xpack.github.io/blog/2022/05/14/riscv-none-elf-gcc-v11-3-0-1-released/>

<https://github.com/xpack-dev-tools/riscv-none-elf-gcc-xpack/releases/tag/v11.3.0-1/>

Unzip Ripes in a directory of your choice. Unzip the compiler in C:\xpack-riscv-none-elf-gcc-11.3.0-1

Add to your PATH: C:\xpack-riscv-none-elf-gcc-11.3.0-1\bin

Copy memory\_map.ld and test.c in your work directory and issue the following commands from a Windows command prompt:

```
riscv-none-elf-gcc -march=rv32im -mabi=ilp32 -nostdlib -ffreestanding -Tmemory_map.ld -o test.elf test.c
```

```
riscv-none-elf-objcopy -O binary test.elf test.bin
```

memory\_map.ld

ENTRY(main)

MEMORY

```
{
    /* program counter (PC) is initially set to 0x00000000 */
    rom (rx): ORIGIN = 0x00000000, LENGTH = 1K

    ram (rw): ORIGIN = 0x00001000, LENGTH = 1K
}
```

SECTIONS

```
{
    .text : {
        /*
            entry point is expected to be the first function here
            --> we are assuming there's only a single function in the .text.boot segment and by convention that is
            "main"
        */
        KEEP(*( .text.boot))

        /*
            all other code follows
        */
        *(.text*)
    } > rom

    .rodata : { *(.rodata*) } > rom
}
```

```
.bss : { *(.bss*) } > ram
}
```

test.c

```
/*
```

```
*** test.c ***
```

```
*** It can be simulated with Ripes. To inspect data memory in Ripes set RAM address to 0x00001000 ***
```

You need to make an ELF file without headers, which starts at the address 0x00000000, with the following command:

```
riscv-none-elf-gcc -march=rv32im -mabi=ilp32 -nostdlib -ffreestanding -Tmemory_map.ld -o test.elf test.c
```

As we are using a bare metal system without an elf loader, we get rid of the elf part and use the binary only:

```
riscv-none-elf-objcopy -O binary test.elf test.bin
```

```
*/
```

```
#include <stdint.h>
```

```
// RAM data segment address set at the beginning of the second Kbyte.
```

```
// The first Kbyte contains the program.
```

```
#define START_DATA_ADDRESS 0x00001000
```

```
int main(void)
```

```
{
```

```
    volatile uint32_t *a = (uint32_t*)(START_DATA_ADDRESS);
    volatile uint32_t *b = (uint32_t*)(START_DATA_ADDRESS + 0x4);
    volatile uint32_t *c = (uint32_t*)(START_DATA_ADDRESS + 0x8);
    volatile uint32_t *d = (uint32_t*)(START_DATA_ADDRESS + 0xC);
    volatile uint32_t *e = (uint32_t*)(START_DATA_ADDRESS + 0x10);
```

```
    volatile uint32_t x = 0;
```

```
    *a = 0xFF;
```

```
    *b = 0xFFFF;
```

```
    *c = 0xFFFFFFFF;
```

```
    *d = 0xFFFFFFFF;
```

```
    while (1) {
```

```
        *e = x;
```

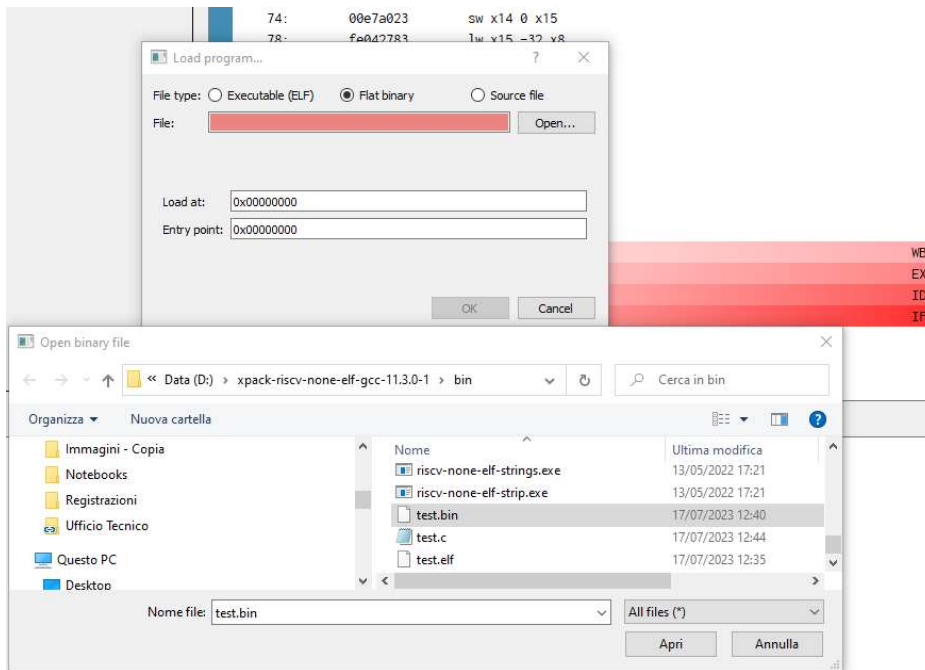
```
        x++;
```

```
        if (x == 0xFFFFFFFF) x = 0;
```

```
    }
```

```
}
```

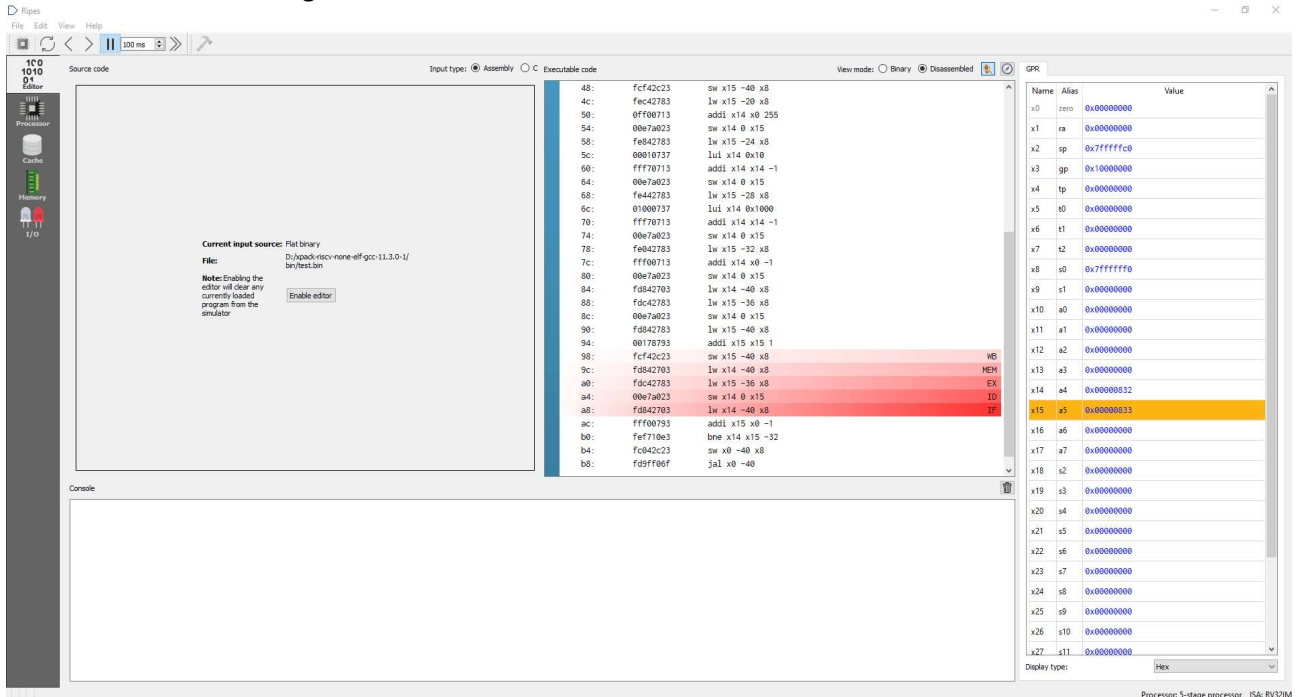
Load test.bin file in Ripes. You can alternatively load the test.elf file:



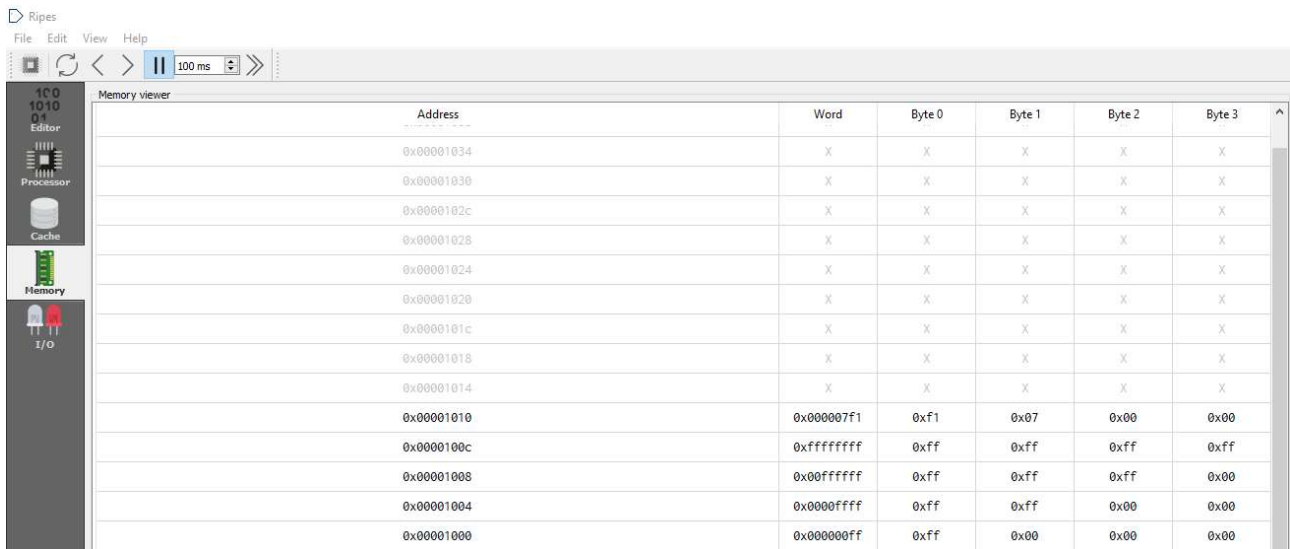
Press the green triangular button to start simulation:



Simulation is now running:



Set RAM memory display starting address at 0x00001000:



The screenshot shows the Ripes Memory viewer interface. On the left is a sidebar with icons for Editor, Processor, Cache, Memory, and I/O. The main window displays a table of memory data. The table has columns for Address, Word, Byte 0, Byte 1, Byte 2, and Byte 3. The data is organized into two sections: the first section shows addresses from 0x00001034 down to 0x00001014, all containing 'X' values; the second section starts at address 0x00001010 and contains specific hexadecimal data.

Address	Word	Byte 0	Byte 1	Byte 2	Byte 3
0x00001034	X	X	X	X	X
0x00001030	X	X	X	X	X
0x0000102c	X	X	X	X	X
0x00001028	X	X	X	X	X
0x00001024	X	X	X	X	X
0x00001020	X	X	X	X	X
0x0000101c	X	X	X	X	X
0x00001018	X	X	X	X	X
0x00001014	X	X	X	X	X
0x00001010	0x000007f1	0xf1	0x07	0x00	0x00
0x0000100c	0xffffffff	0xff	0xff	0xff	0xff
0x00001008	0x00ffffff	0xff	0xff	0xff	0x00
0x00001004	0x0000ffff	0xff	0xff	0x00	0x00
0x00001000	0x000000ff	0xff	0x00	0x00	0x00

The test.c program writes the following 4 fixed 32 bit data in RAM starting from address 0x00001000:

```
*a = 0xFF;  
*b = 0xFFFF;  
*c = 0xFFFFFFFF;  
*d = 0xFFFFFFFF;
```

You will see the 'e' counter at address 0x00001010 which continuously increments at a fast rate...



# Packaging RISC-V Source as Vivado IP

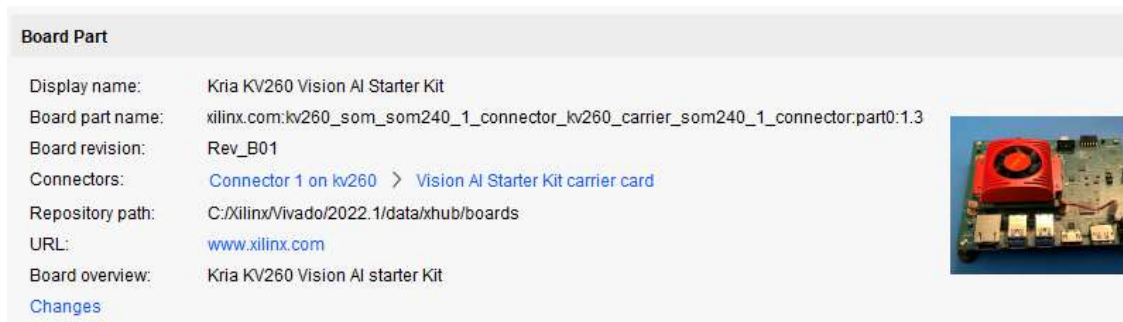
Host PC: Windows 10 Professional + Vivado 2022.1 KV260 board: UBUNTU 2022.04.2 + PYNQ 3.0.1

Main reference:

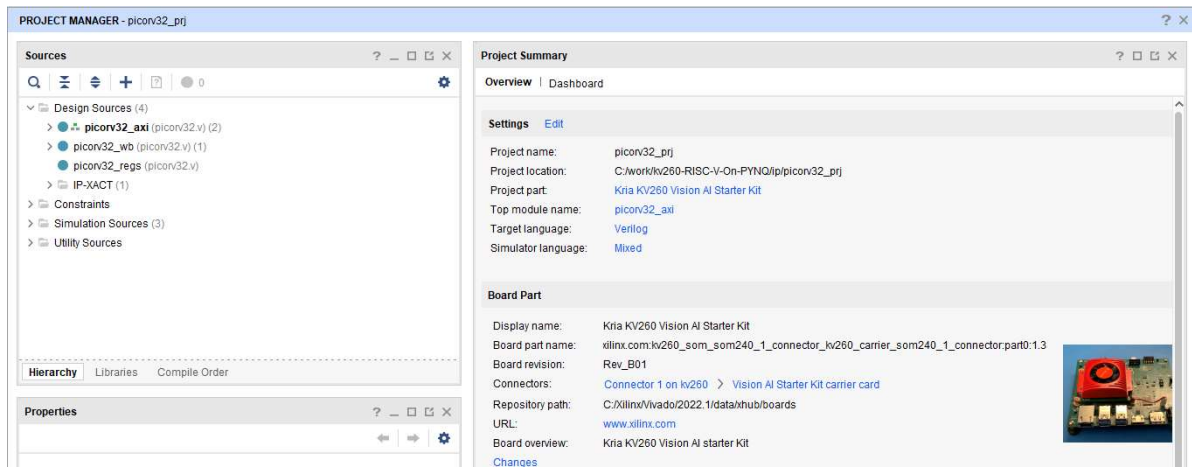
<https://github.com/drichmond/RISC-V-On-PYNQ/blob/master/notebooks/tutorial/2-Creating-A-Bitstream.ipynb>

Steps:

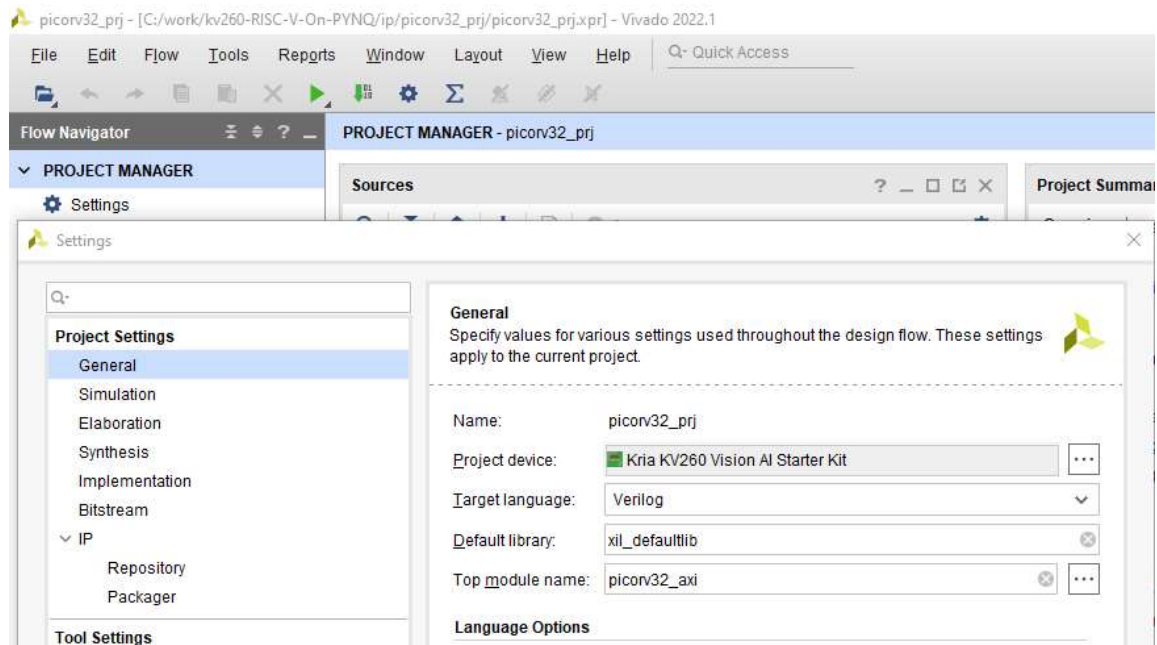
- 1) Open Vivado IDE and create the following project in: **C:/work/kv260-RISC-V-On-PYNQ/ip**  
Project name: picorv32\_prj.  
Check "Create project subdirectory" check box.
- 2) Add picorv32.v source file from: **C:/work/kv260-RISC-V-On-PYNQ/picorv32**
- 3) Select:



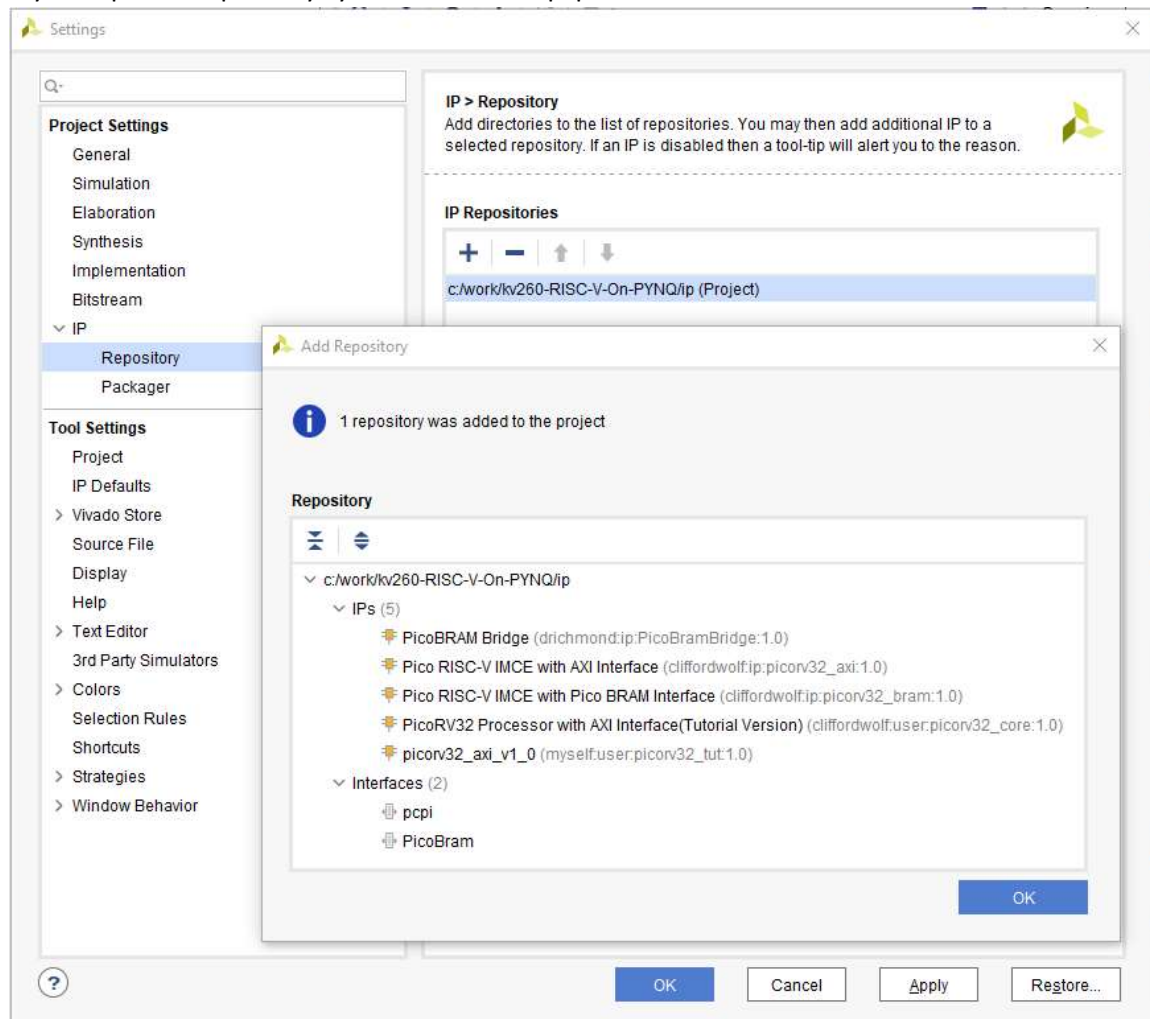
- 4) This should be the final result:



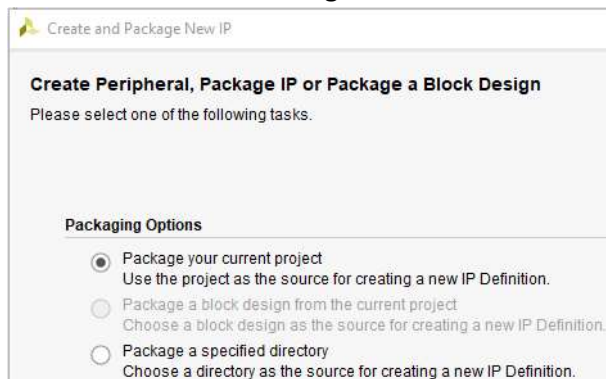
- 5) Add an IP Interface that will encapsulate the PCPI interface provided by the picorv32.  
Under Flow Navigator→PROJECT MANAGER click on Settings:



Select Repository, press + and add c:/work/kv260-RISC-V-On-PYNQ/ip  
If you expand “Repository” you will find the “pcpi” interface:



6) **Tools → Create and Package New IP...** Press **Next**.

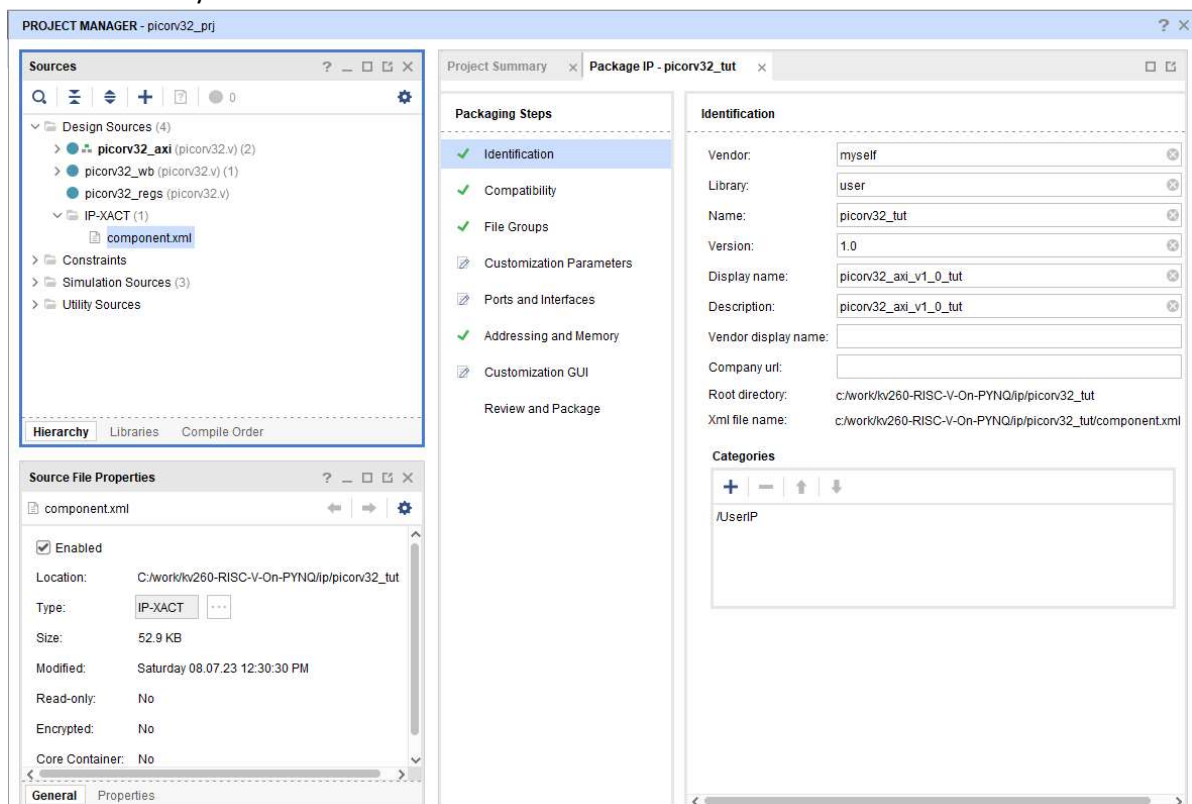


Press **Next**.



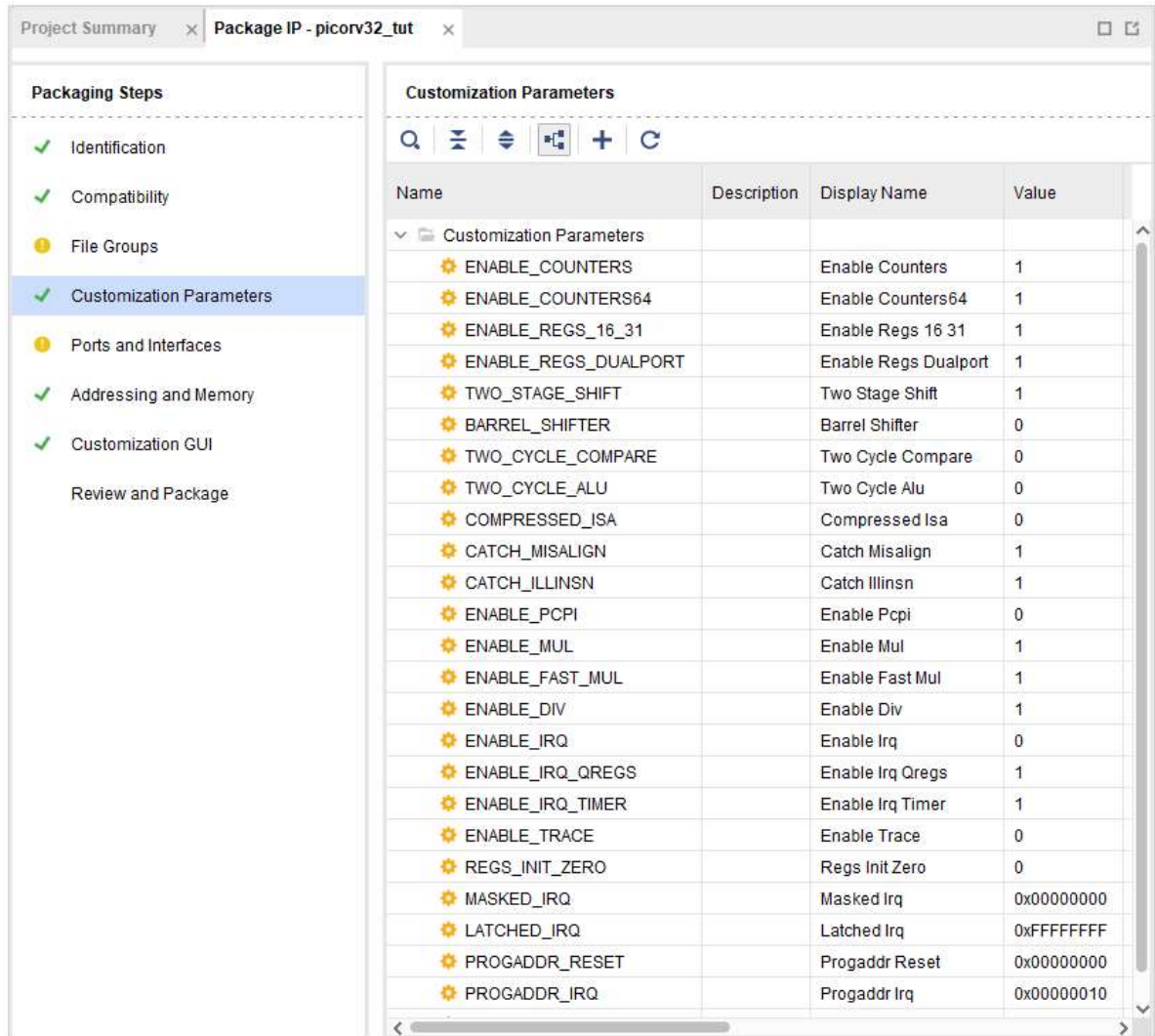
Press **Next** and confirm that you want to create a folder for your New IP.

7) Click **Next** until you arrive at the screen shown below:



The same window can be opened at any time with a double click on **component.xml** inside Sources panel.

8) Configure parameters as follows:



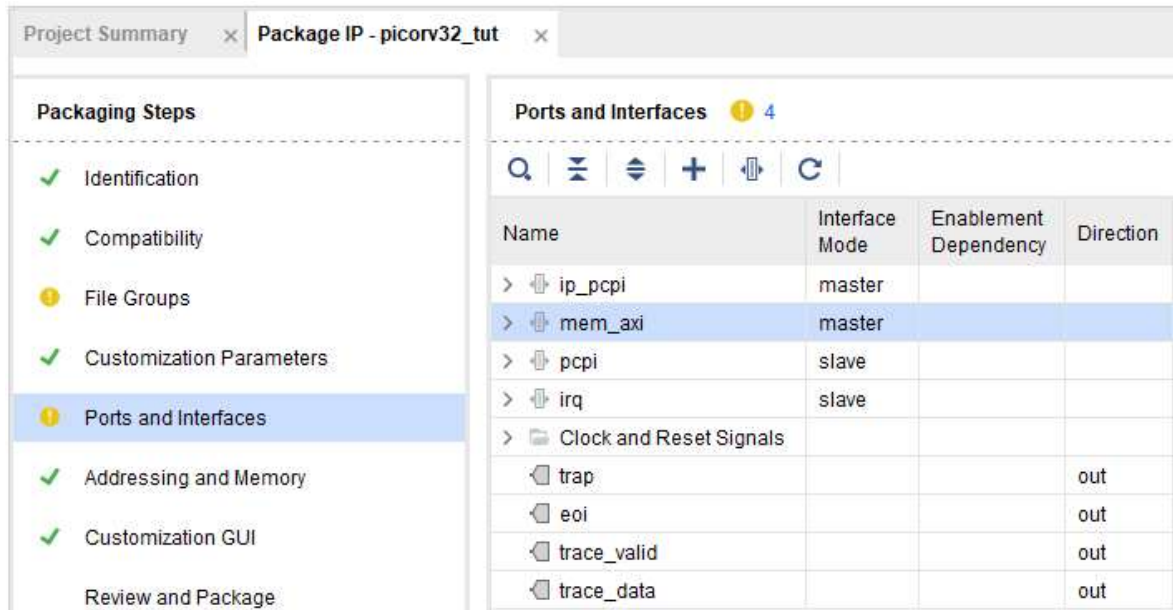
With respect to default configuration that you will find, you have to enable in addition:

ENABLE\_MUL Value: 1

ENABLE\_FAST\_MUL Value: 1

ENABLE\_DIV: 1

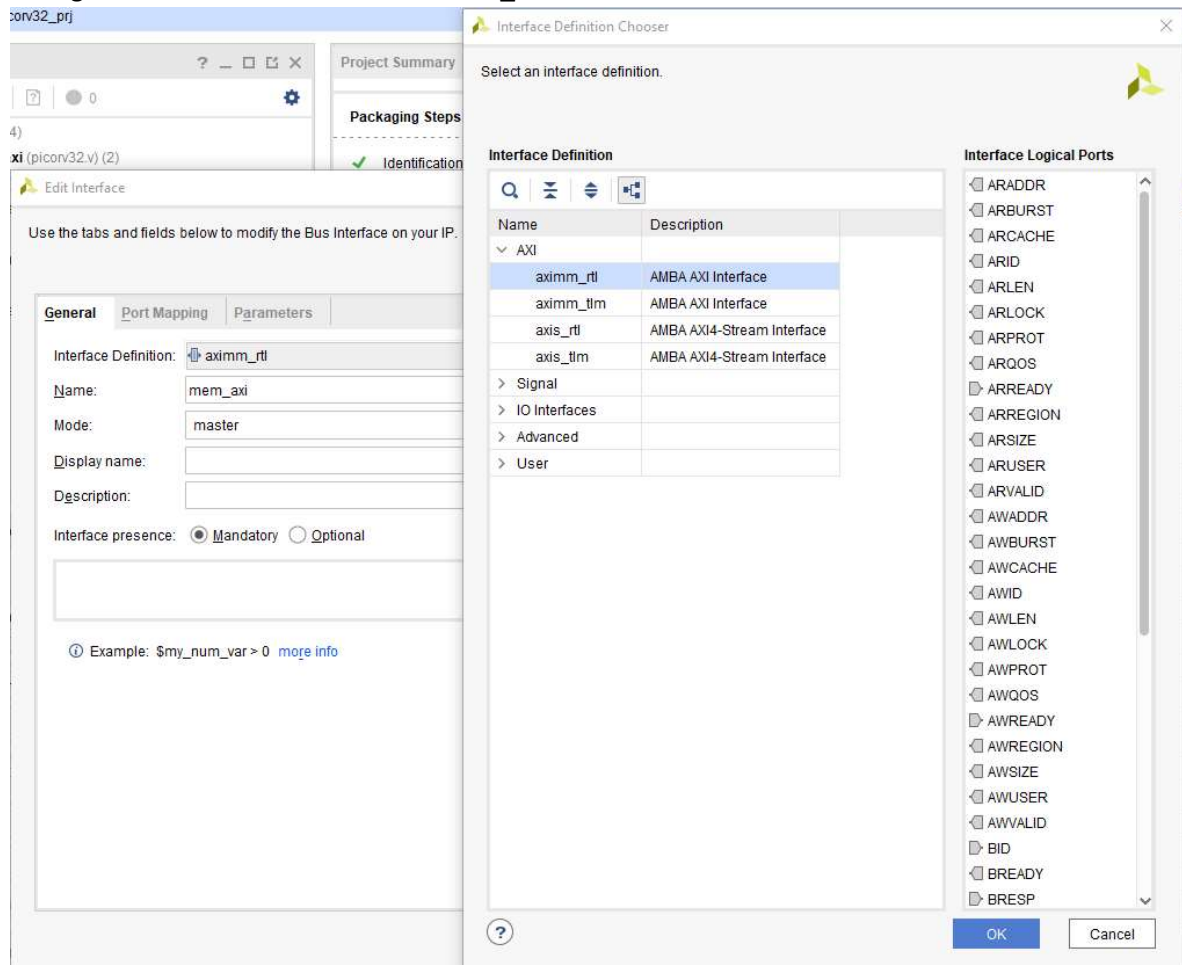
- 9) Right click on mem\_axi, open the “Edit interface...” window and select “General” tab:



The screenshot shows the 'Package IP - picorv32\_tut' window with the 'Ports and Interfaces' tab selected. The 'Packing Steps' on the left include Identification, Compatibility, File Groups, Customization Parameters, Ports and Interfaces (selected), Addressing and Memory, Customization GUI, and Review and Package. The 'Ports and Interfaces' table lists the following interfaces:

Name	Interface Mode	Enablement Dependency	Direction
> ip_pcp	master		
> mem_axi	master		
> pcp	slave		
> irq	slave		
> Clock and Reset Signals			
trap			out
eoi			out
trace_valid			out
trace_data			out

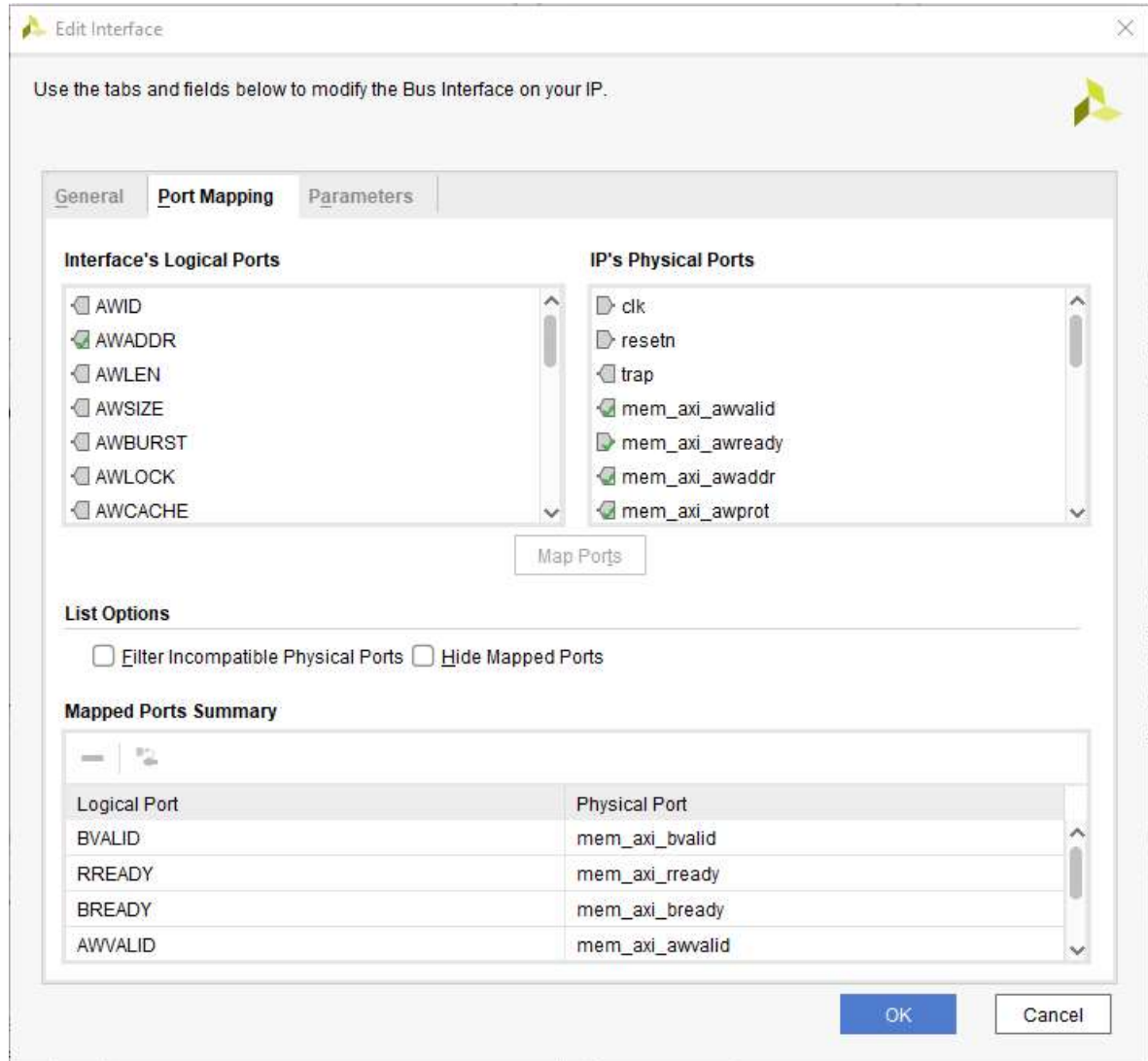
Change the interface definition to aximm\_rtl:



The screenshot shows the 'Interface Definition Chooser' window. The 'Interface Definition' list on the left includes 'aximm\_rtl' (selected), 'aximm\_tlm', 'axis\_rtl', and 'axis\_tlm'. The 'Interface Logical Ports' list on the right includes a long list of ports such as ARADDR, ARBURST, ARCACHE, ARID, ARLEN, ARLOCK, ARPROT, ARQOS, ARREADY, ARREGION, ARSIZE, ARUSER, ARVALID, AWADDR, AWBURST, AWCACHE, AWID, AWLEN, AWLOCK, AWPROT, AWQOS, AWREADY, AWREGION, AWSIZE, AWUSER, AWVALID, BID, BREADY, and BRESP. The 'OK' button is highlighted in blue.



10) Right click on mem\_axi, open the “Edit interface...” window and select “Port Mapping” tab:

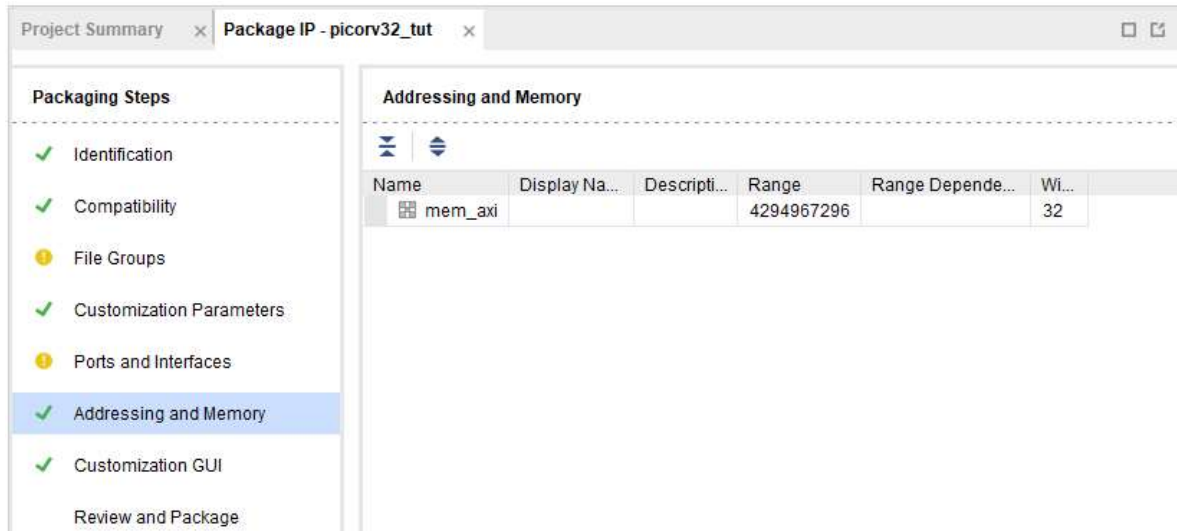


Correct Port Mapping bindings:

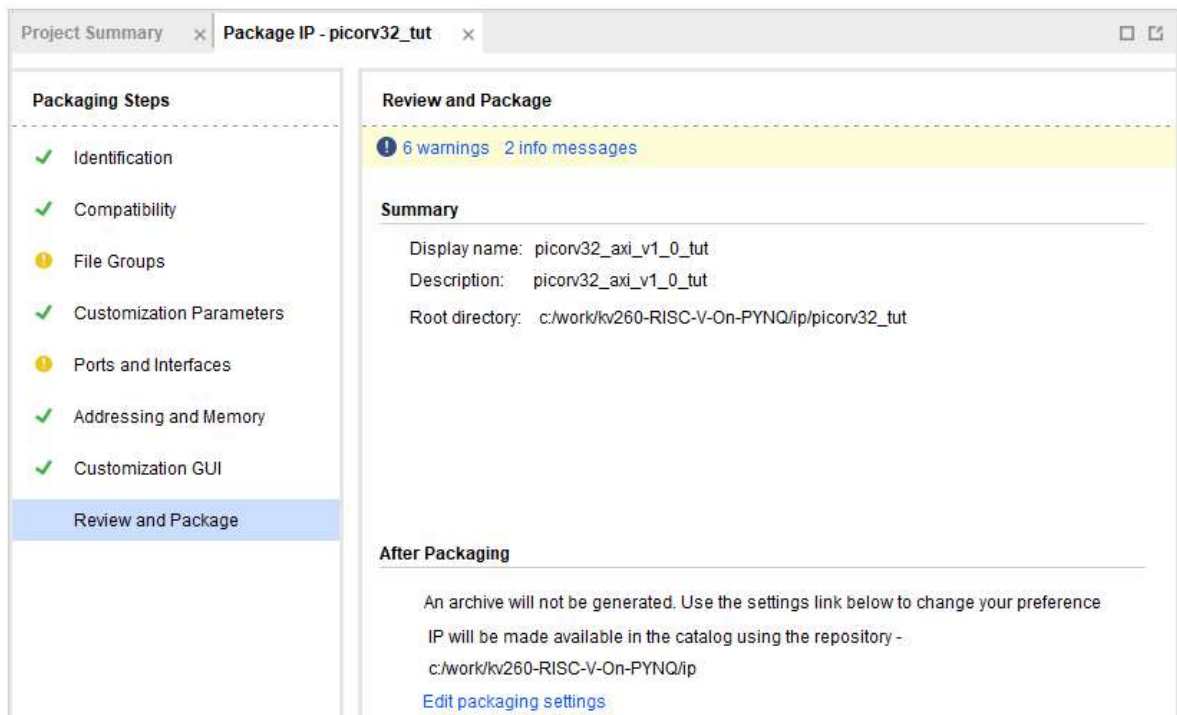
AWADDR - mem\_axi\_awaddr  
 AWPROT - mem\_axi\_awprot  
 AWVALID - mem\_axi\_awvalid  
 AWREADY - mem\_axi\_awready  
 WDATA - mem\_axi\_wdata  
 WSTRB - mem\_axi\_wstrb  
 WVALID - mem\_axi\_wvalid  
 WREADY - mem\_axi\_wready  
 BVALID - mem\_axi\_bvalid  
 BREADY - mem\_axi\_bready  
 ARADDR - mem\_axi\_araddr  
 ARPROT - mem\_axi\_arprot  
 ARVALID - mem\_axi\_arvalid  
 ARREADY - mem\_axi\_arready  
 RDATA - mem\_axi\_rdata  
 RVALID - mem\_axi\_rvalid  
 RREADY - mem\_axi\_rready



## 11) Addressing and Memory configuration:



## 12) Last step: Review and Package:



The IP has been generated in: C:/work/kv260-RISC-V-On-PYNQ/ip/picorv32\_tut

Ignore the following warnings and info:

