

# *Projeto Integrador* Embedded Systems

*Plate recognition with hardware  
acceleration (PrHa)*

Hugo Ribeiro – PG47241  
Ricardo Mendes - PG47612

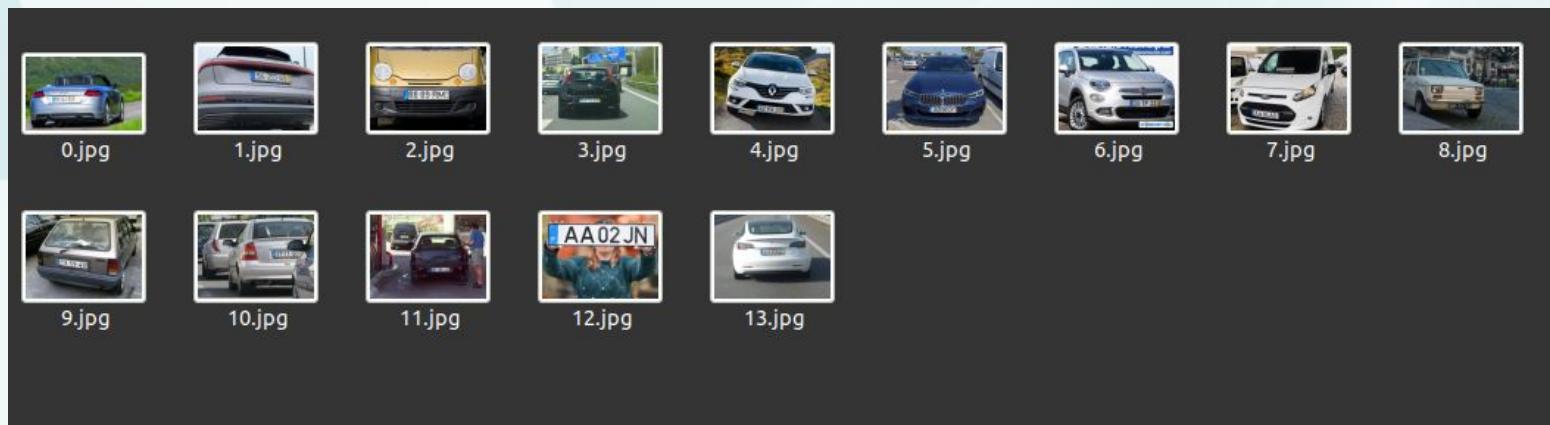
# Agenda



- Code Adaptation from Plgate
- Creating Hardware Platform
- Creating Linux Image
- Creating a Custom Vitis Platform
- Test an Hello World
- Test PrHa Code
- Bibliography

- Before working in the project itself, the Plgate code was changed so it can be reused in this new project
- The Plgate project has database connection, webapp application, camera, etc. which is not necessary for this project in a initial approach
- That features were removed, leaving only what is necessary for the PrHa project
- Plgate Repository:  
<https://github.com/HugoRibeiro-A88287-UM/Plgate>

- In a initial approach will be used photos from the memory.
- There were selected all types of plates and angles, so it's possible to check if the Plgate's plate recognition model is well trained.
- All the used images are from google



- Next part to be changed was the database one. Like said before, this part isn't necessary for now, so to remove it was only remove the database daemon.
- LedRGB and Relay device drivers aren't needed and by deleting the respective code they were excluded.
- The makefiles were also modified to this new project
- Because PIgate was constituted by modules, this adaptation was fast and easy.

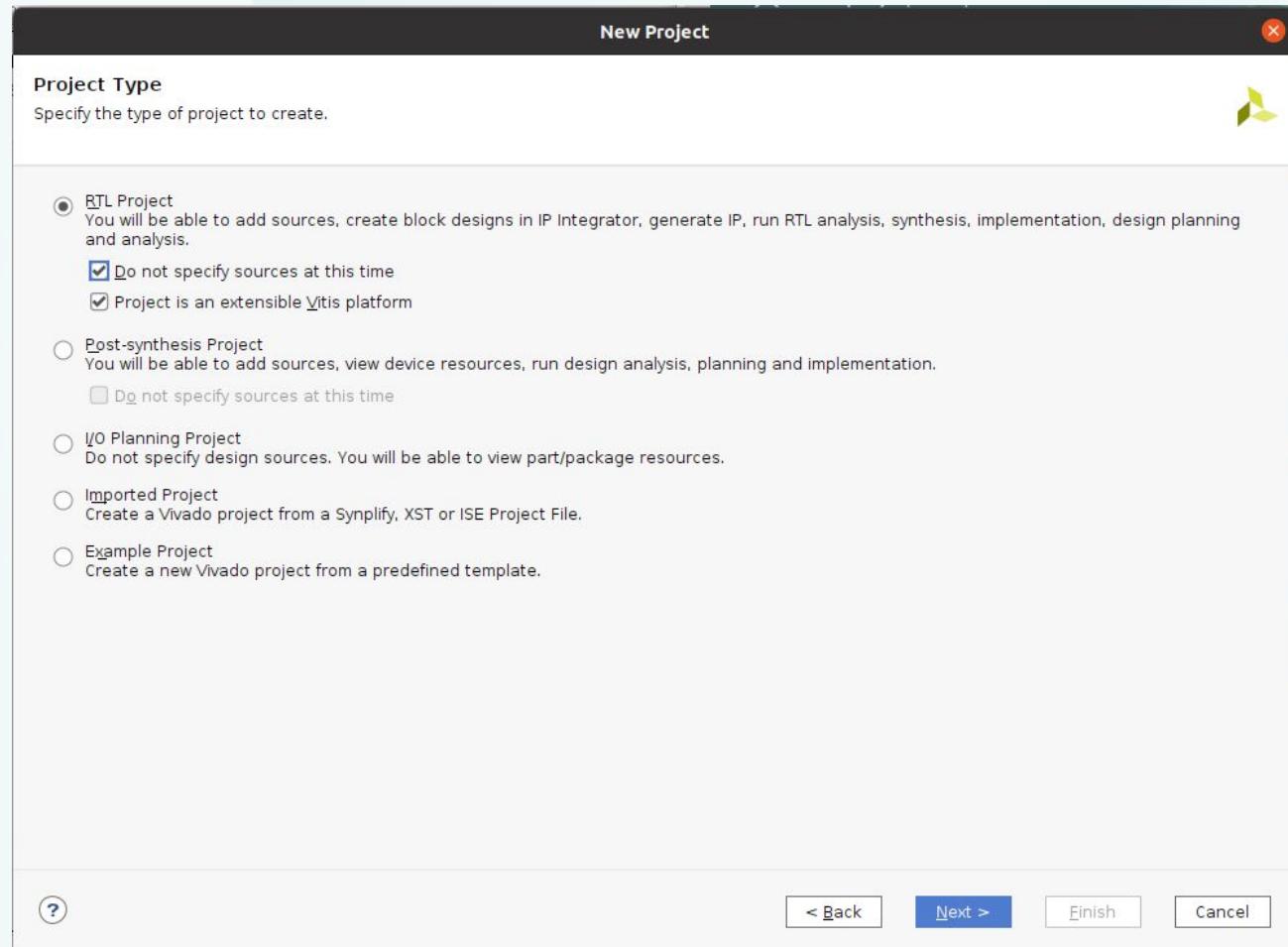
- After all changes, the project folder is the following



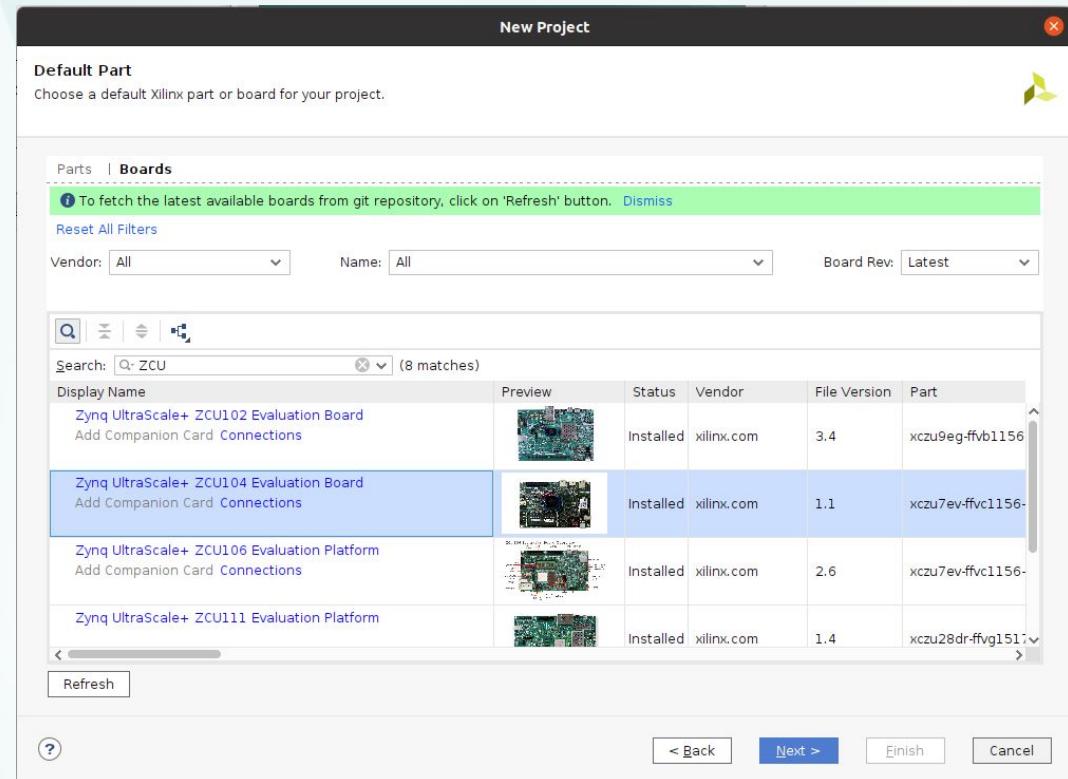
- It is possible to test it on the Laptop. For that, change the compiler in the src folder makefile.
- PrHa Repository:  
<https://github.com/HugoRibeiro-A88287-UM/PrHa>

- Before working with the zcu104 we must configure its hardware;
- It is necessary to create a hardware design including the SoC and other hardware blocks required;
- Vivado is the software used to do this.

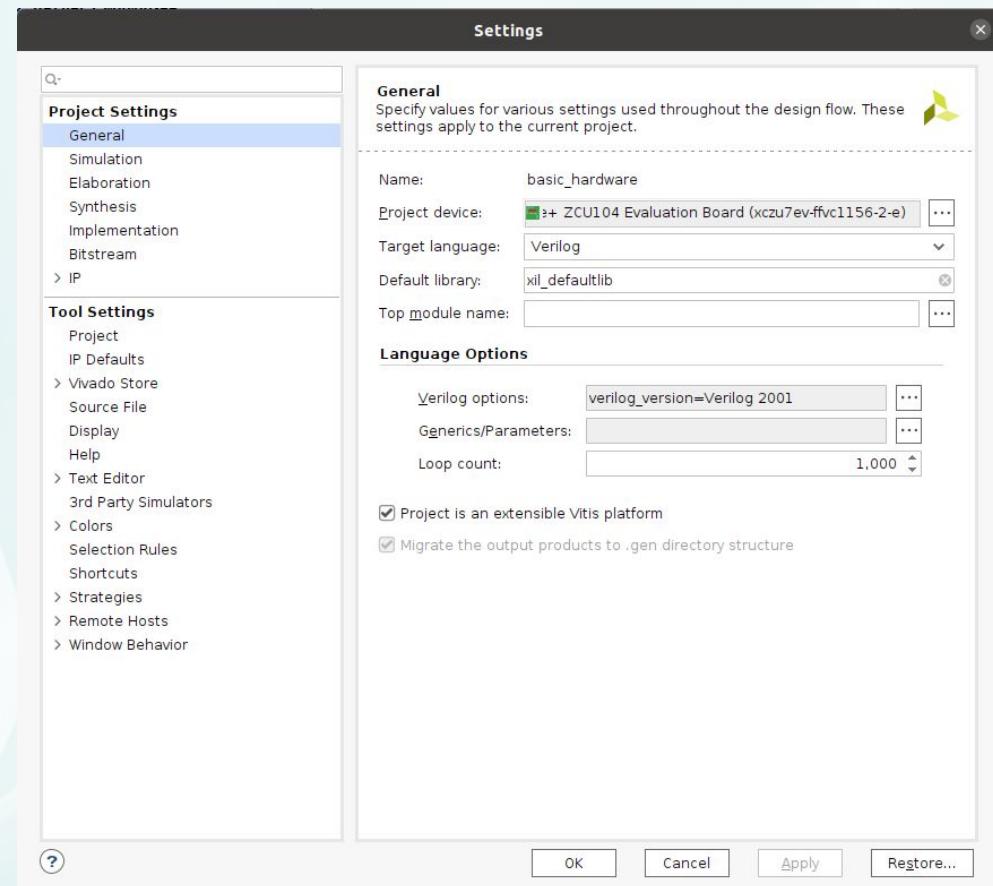
- Creating a vivado project:



- Creating a vivado project:  
-Selecting the desired board:

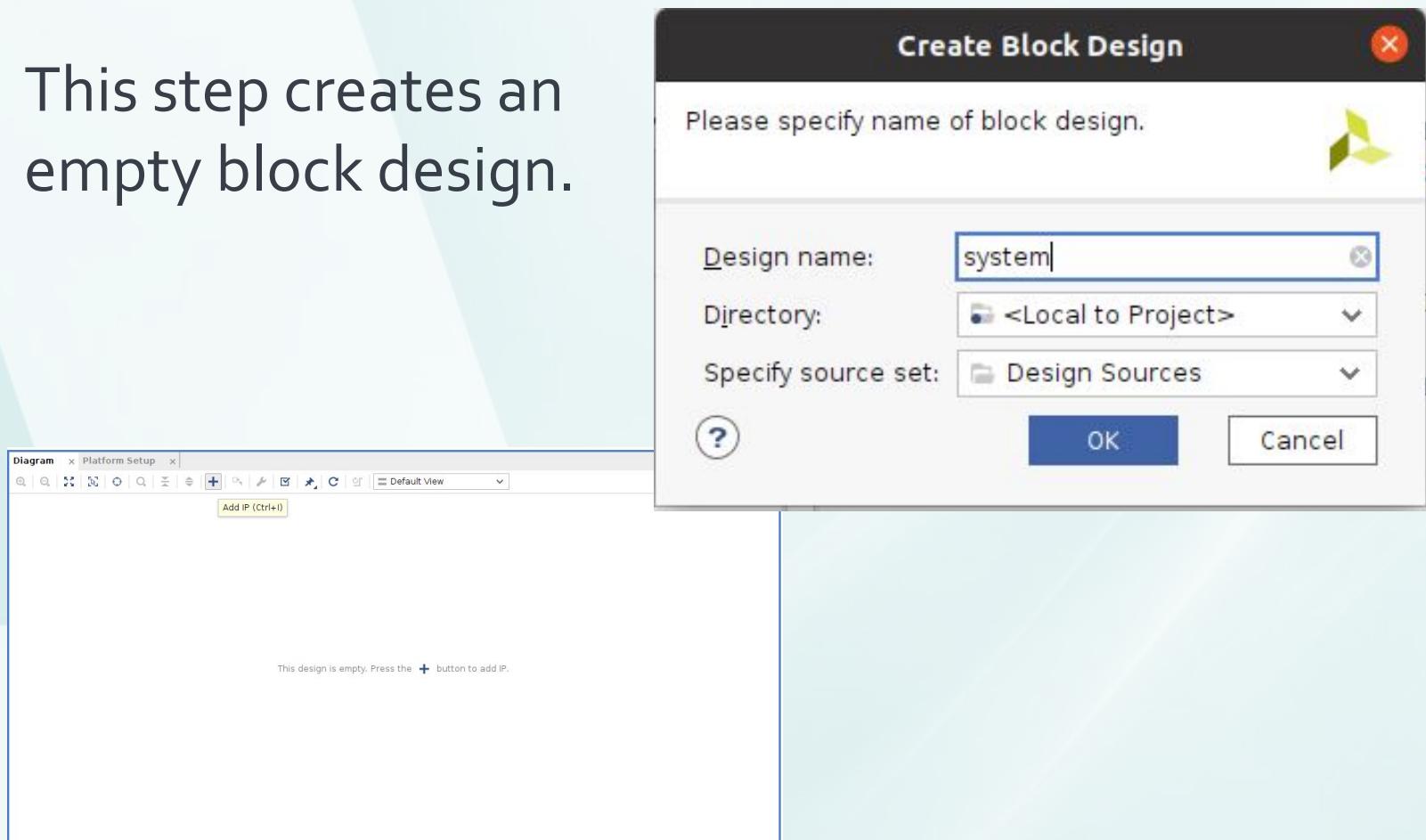


- Creating a vivado project:  
-Project settings:

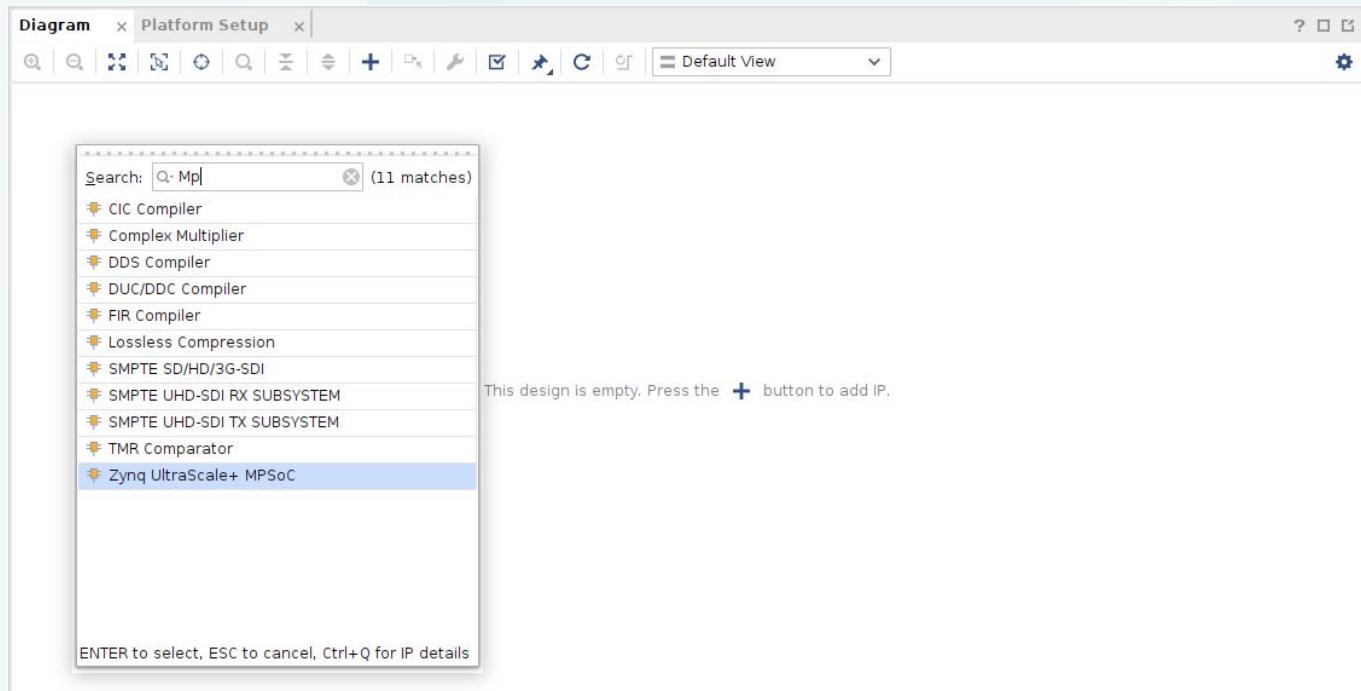


- Creating a block design:

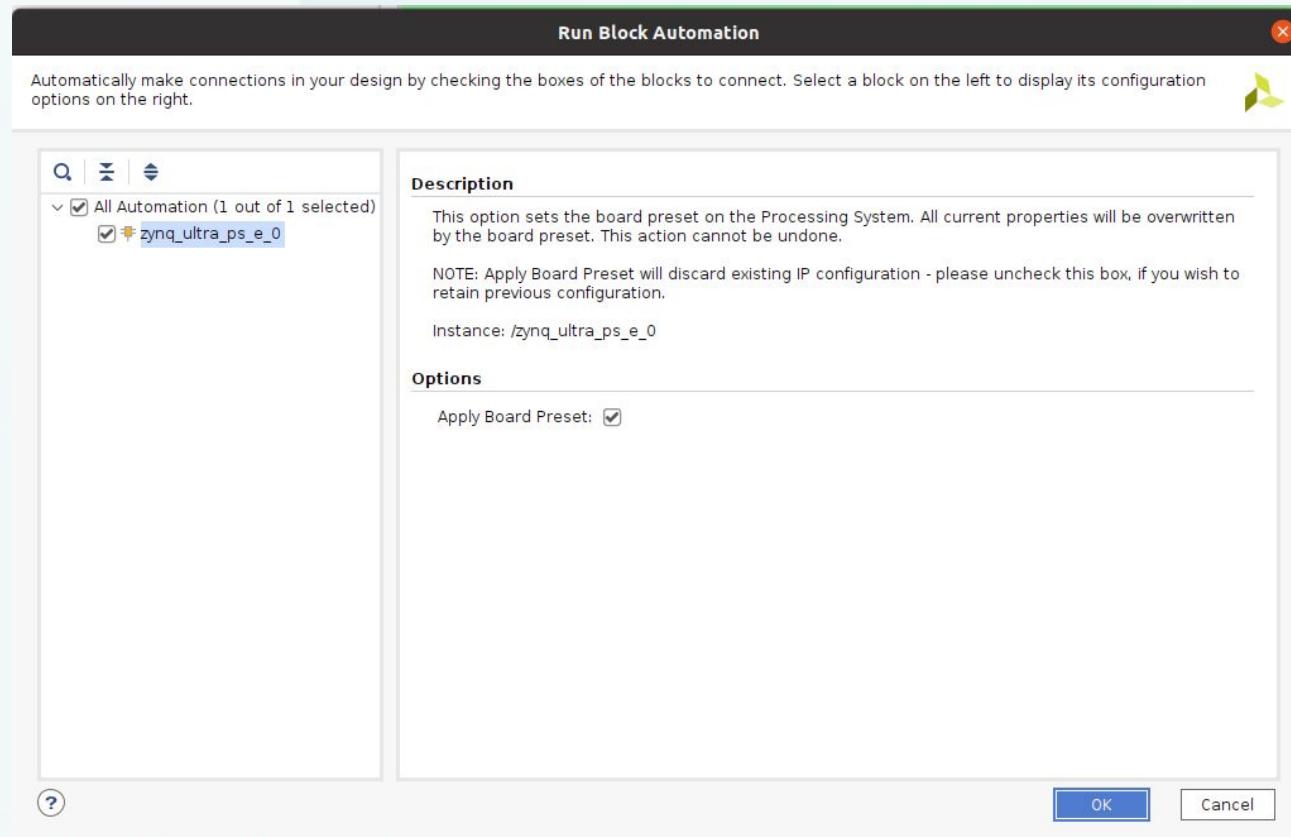
This step creates an empty block design.



- Adding an IP to the design block:  
The first block to add is the board's SoC:
  - Press add IP (+);
  - Search for Zynq Ultrascale+ MPSoC.



- Configure the SoC:
  - Run block automation;
  - Check the following options:

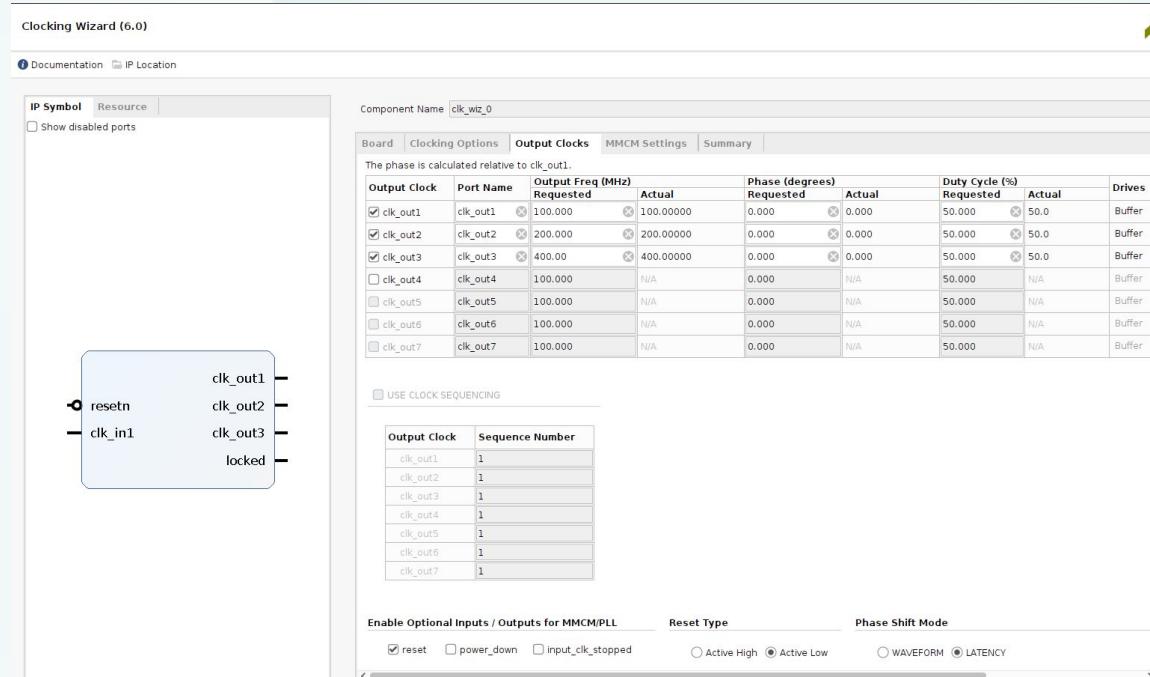


- The resulting block appears on the “Diagram” tab and should look like this:



This step has applied all board presets for the ZCU104, which include block configurations and pin assignments.

- Next, we need to add a clock wizard to the system:
  - Add the Clocking Wizard IP;
  - Enable and configure clk\_out1 -> clk\_out3.



The screenshot shows the Clocking Wizard (6.0) software interface. On the left, there is a component symbol for a clock wizard with pins: clk\_out1, clk\_out2, clk\_out3, clk\_in1, and resetn. On the right, the configuration pane shows the following details:

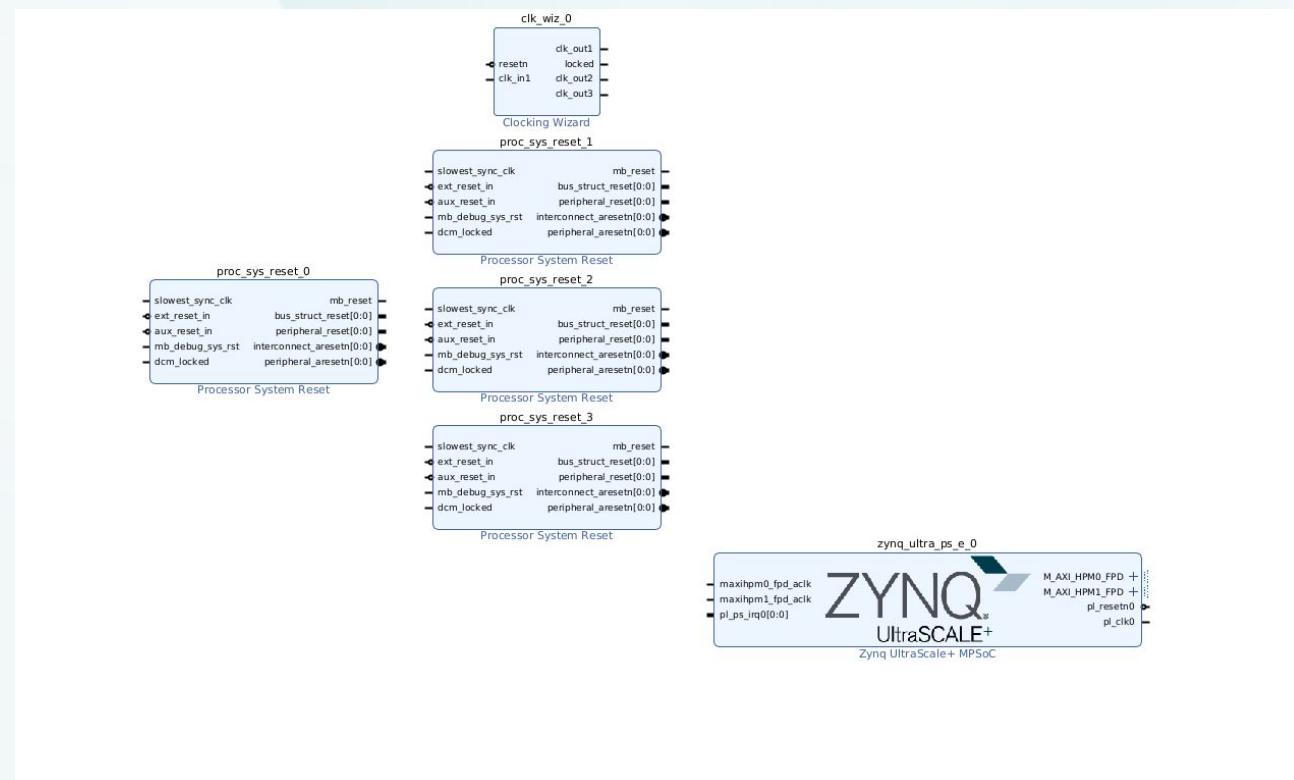
Output Clock	Port Name	Output Freq (MHz) Requested	Actual	Phase (degrees) Requested	Actual	Duty Cycle (%) Requested	Actual	Drives
clk_out1	clk_out1	100.000	100.00000	0.000	0.000	50.000	50.0	Buffer
clk_out2	clk_out2	200.000	200.00000	0.000	0.000	50.000	50.0	Buffer
clk_out3	clk_out3	400.000	400.00000	0.000	0.000	50.000	50.0	Buffer
clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.000	N/A	Buffer
clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A	Buffer
clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A	Buffer
clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A	Buffer

Below the table, there is a section for "USE CLOCK SEQUENCING" with a table:

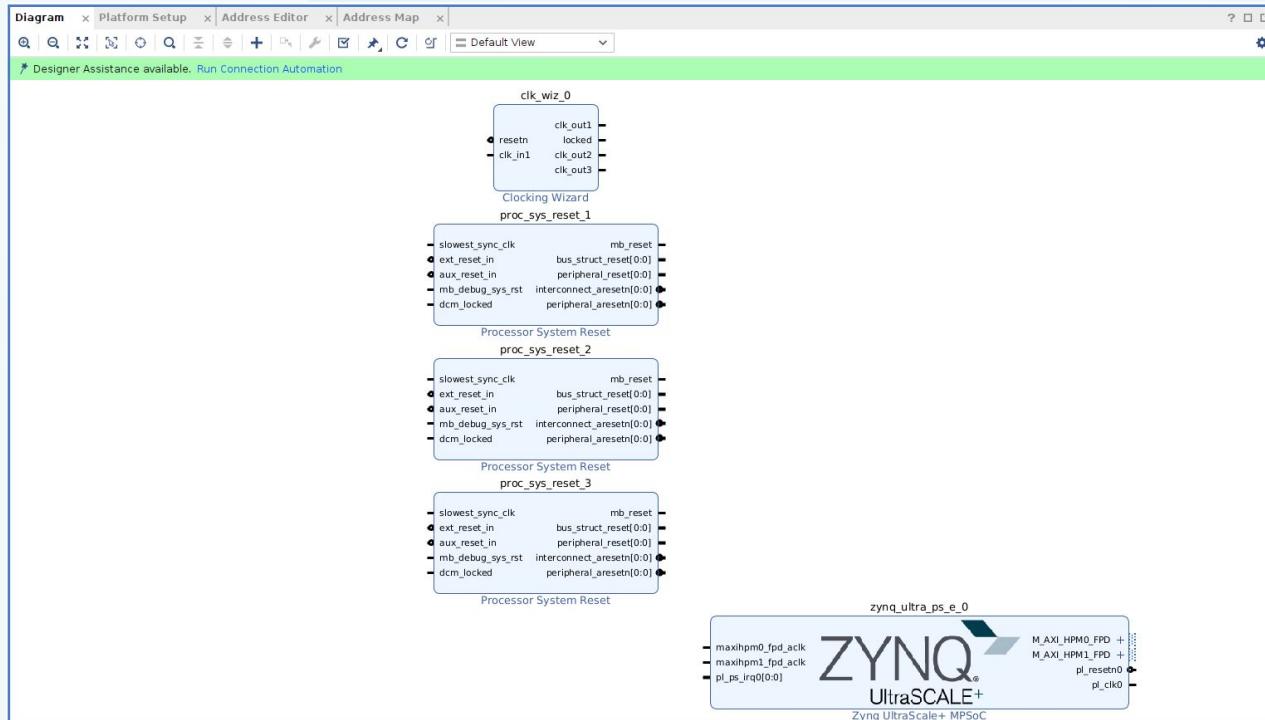
Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

At the bottom, there are sections for "Enable Optional Inputs / Outputs for MMCM/PLL", "Reset Type", and "Phase Shift Mode".

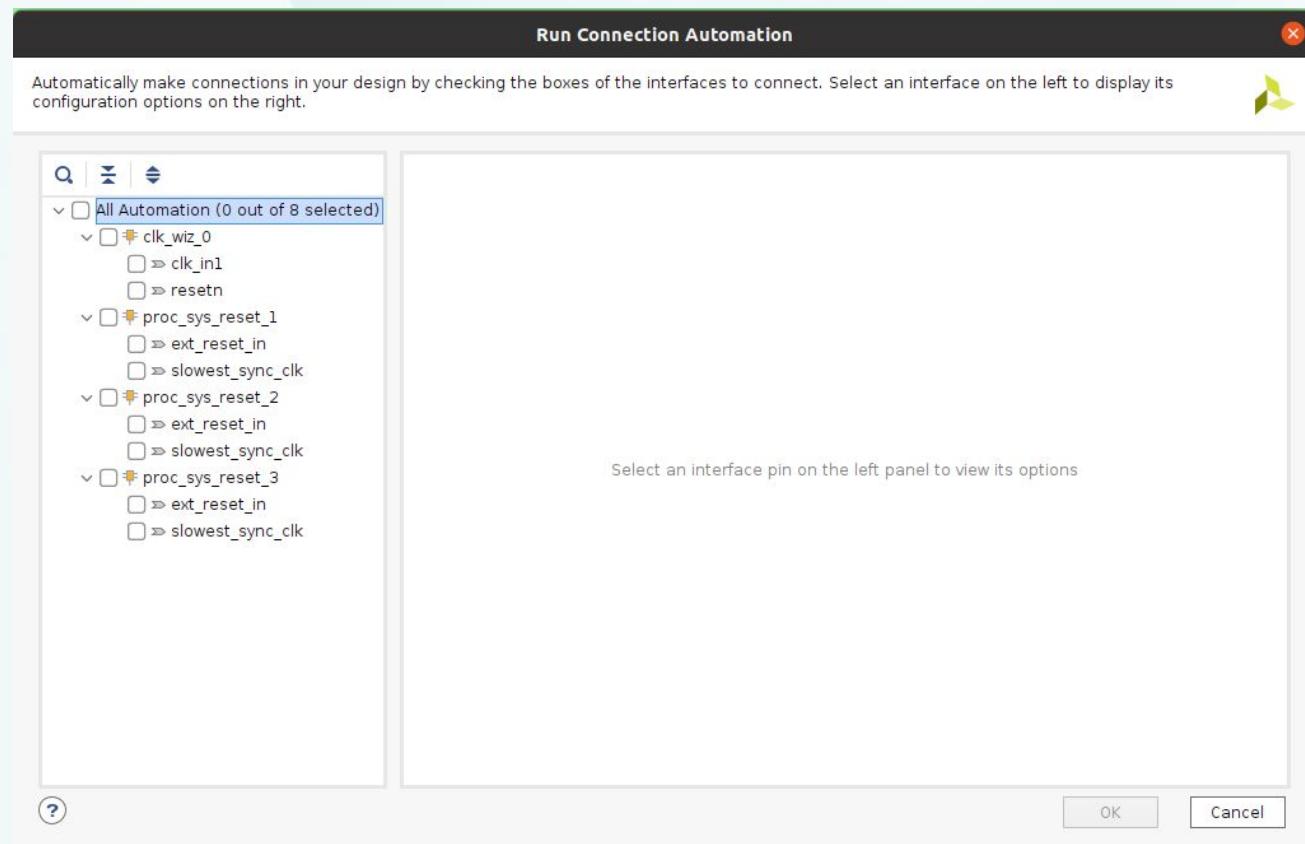
- We will also need a reset block:
  - Add 4 Processor System Reset IP blocks;



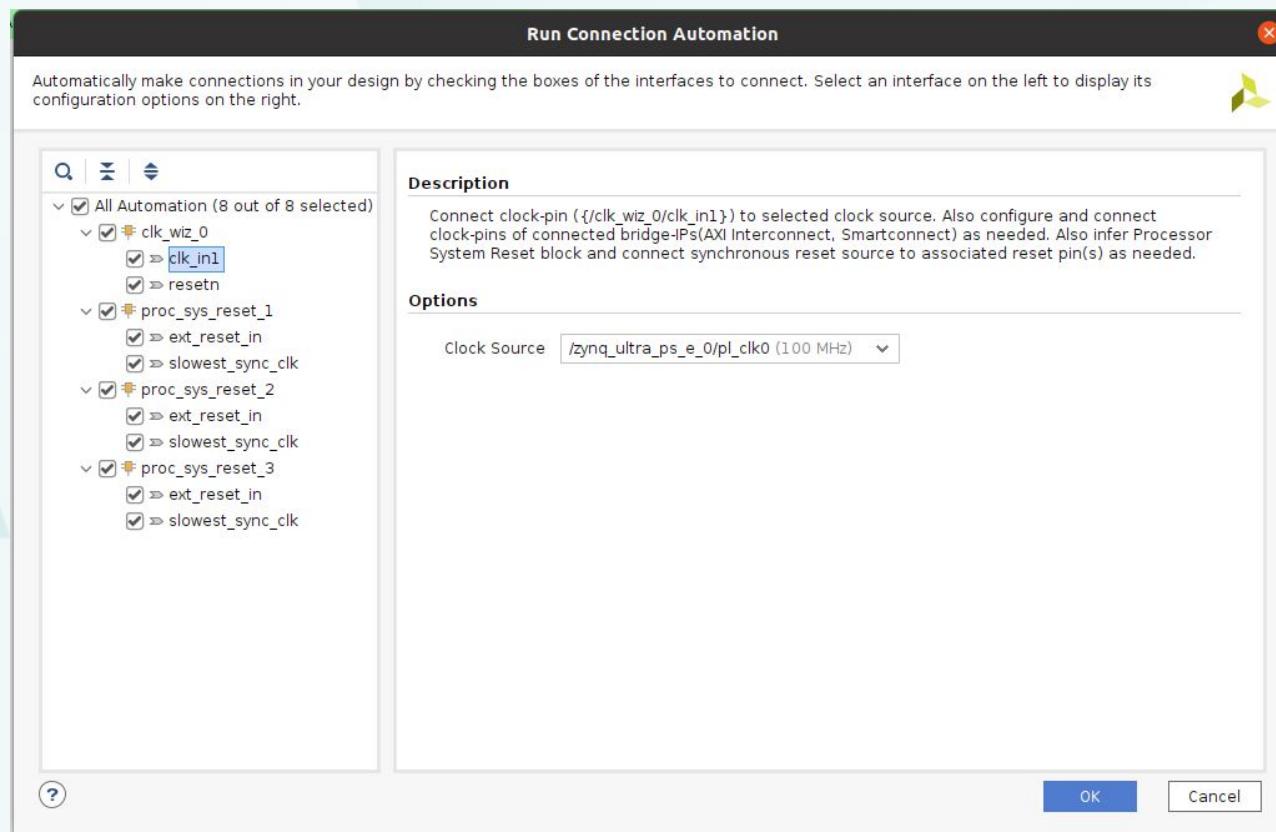
- We will also need a reset block:
  - Delete `proc_sys_reset_o`, in order to keep `proc_sys_reset_1` -> `proc_sys_reset_3`



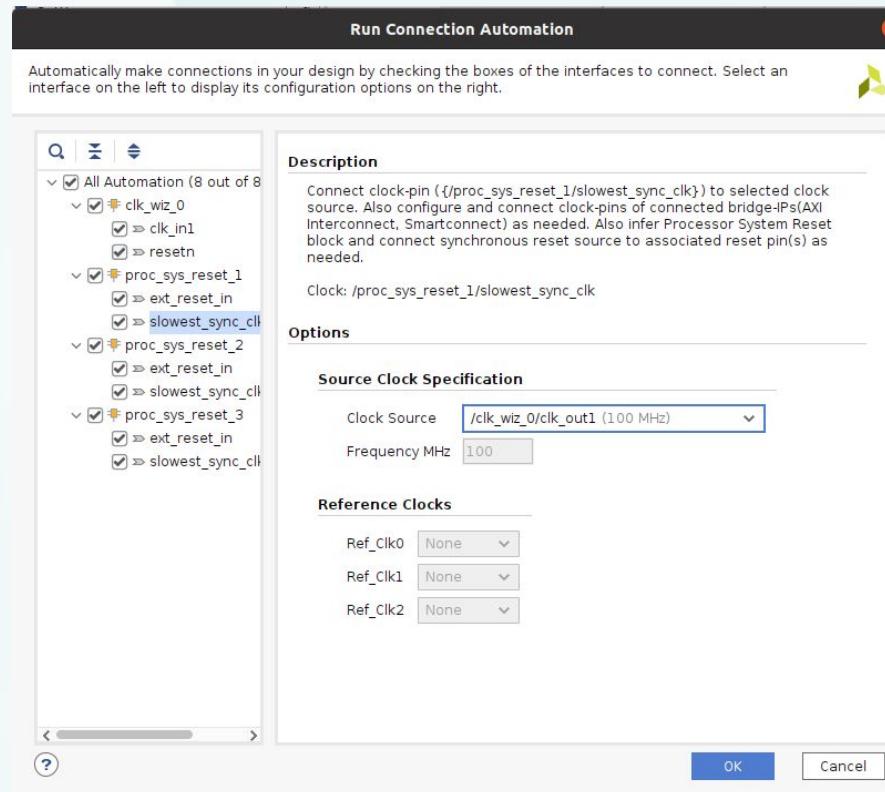
- Next, we should run the connection automation:



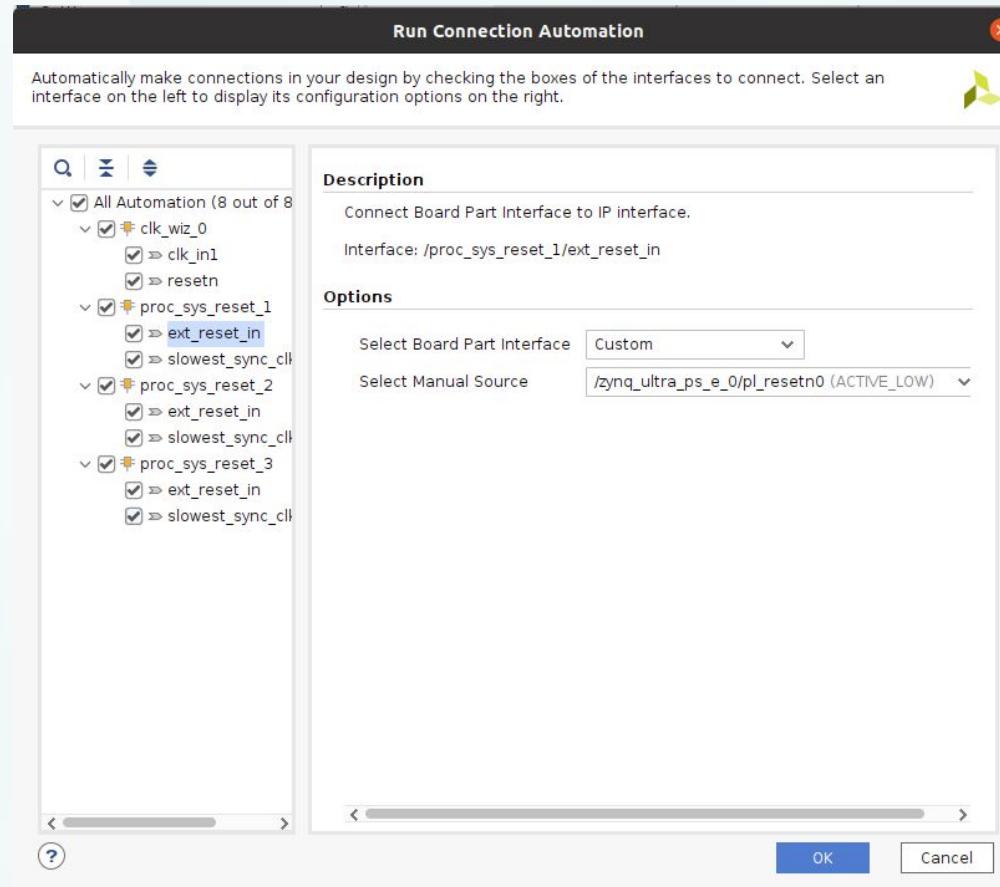
- Select all automation option;
- Under `clk_in1` set the clock source to `/zynq_ultra_ps_e_0/pl_clk0`.



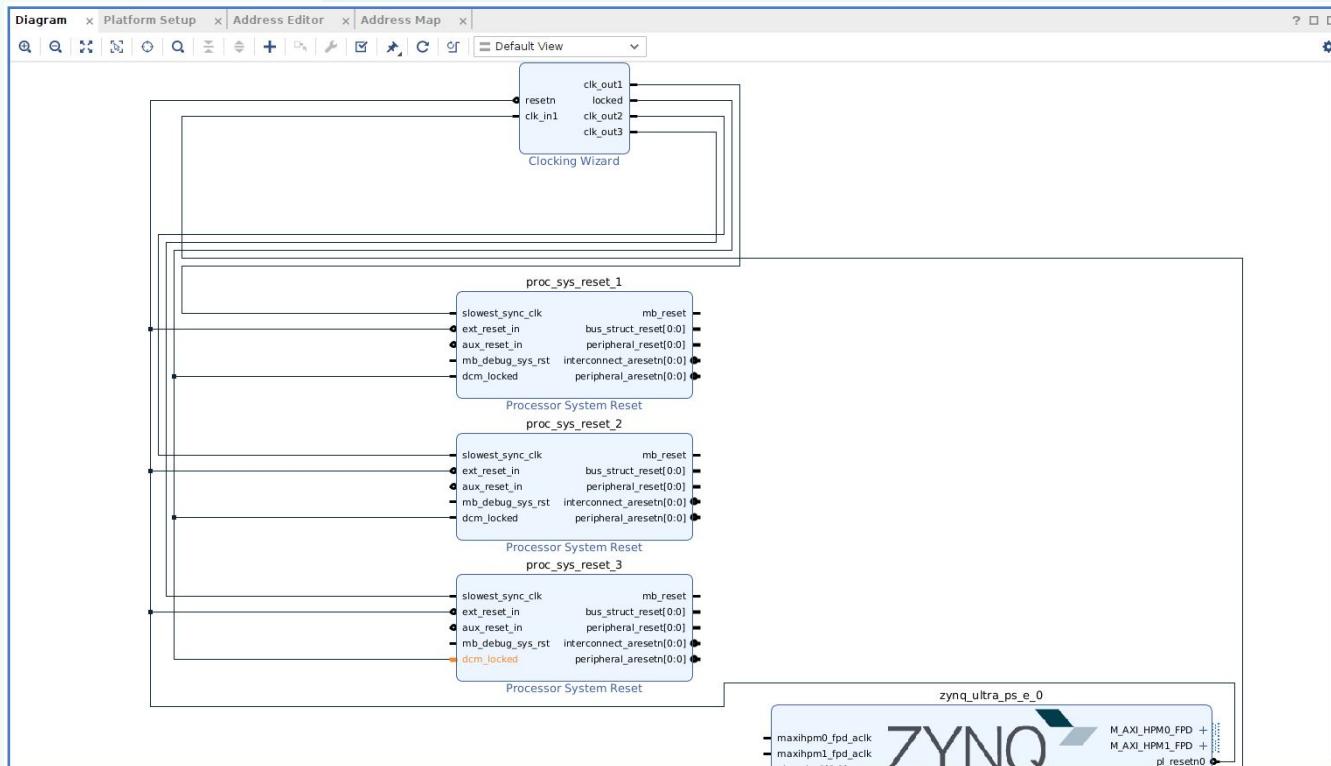
- Under “slowest\_sync\_clk” set the clock source as “/clk\_wiz\_0/clk\_out\_n”;
- Repeat for each reset block:



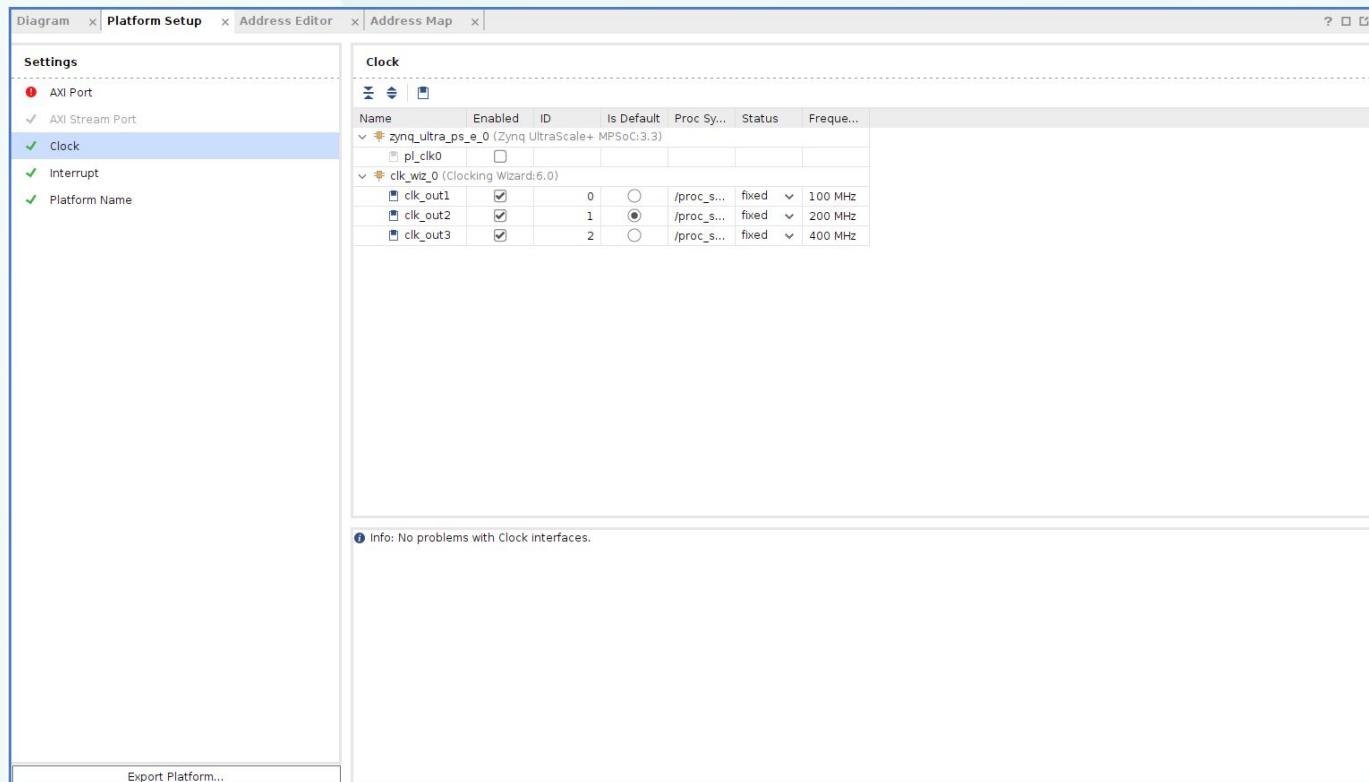
- On each proc\_sys\_reset instance set the following settings:



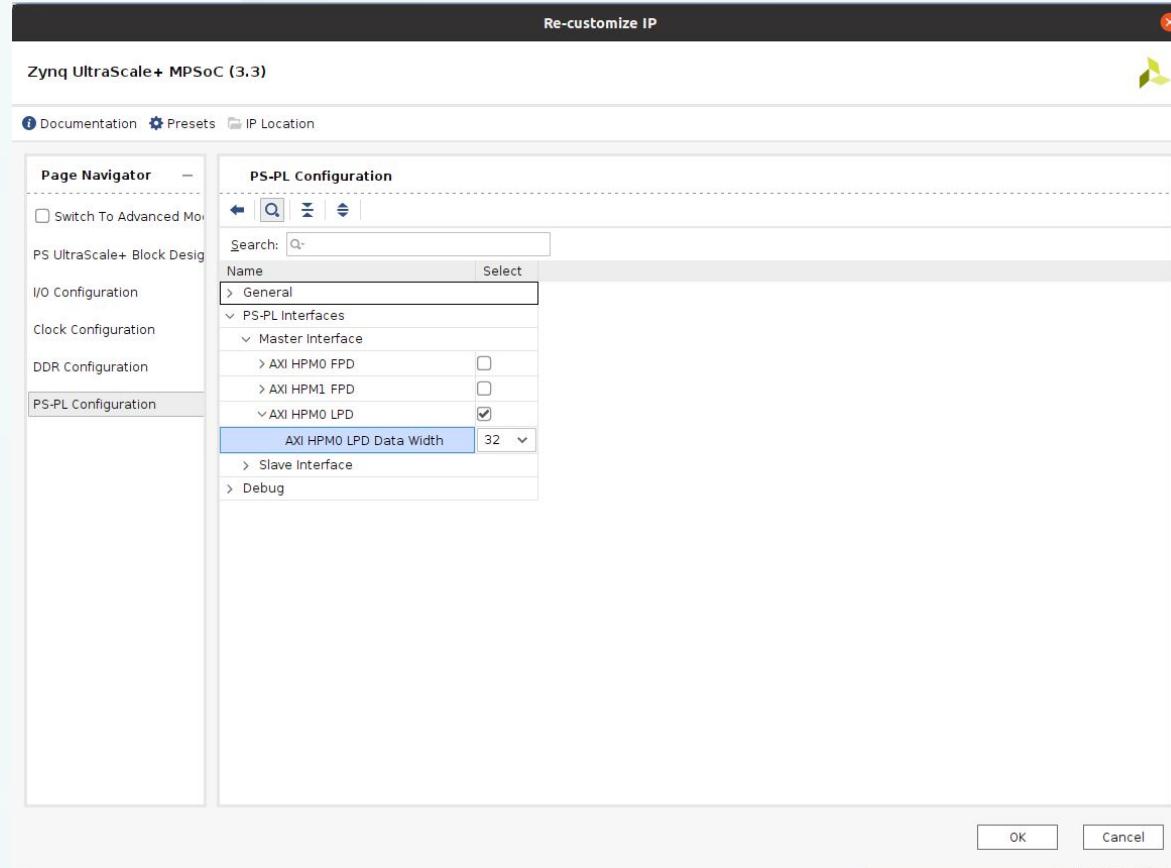
- Finally, we must connect all of the “dcm\_locked” signals to the locked signal on clk\_wiz\_0:



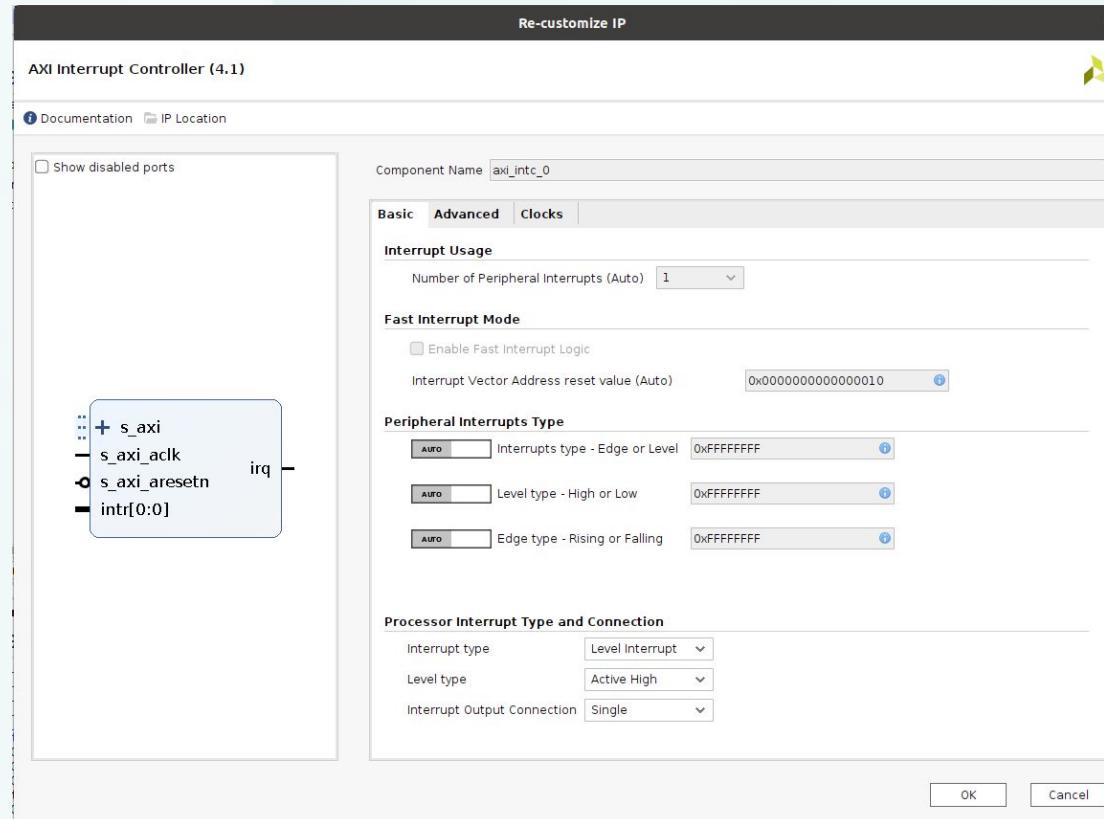
- Enable the clocks:  
In the “Platform” tab enable all clocks under `clk_wiz_0` and change their ID to 0,1,2;



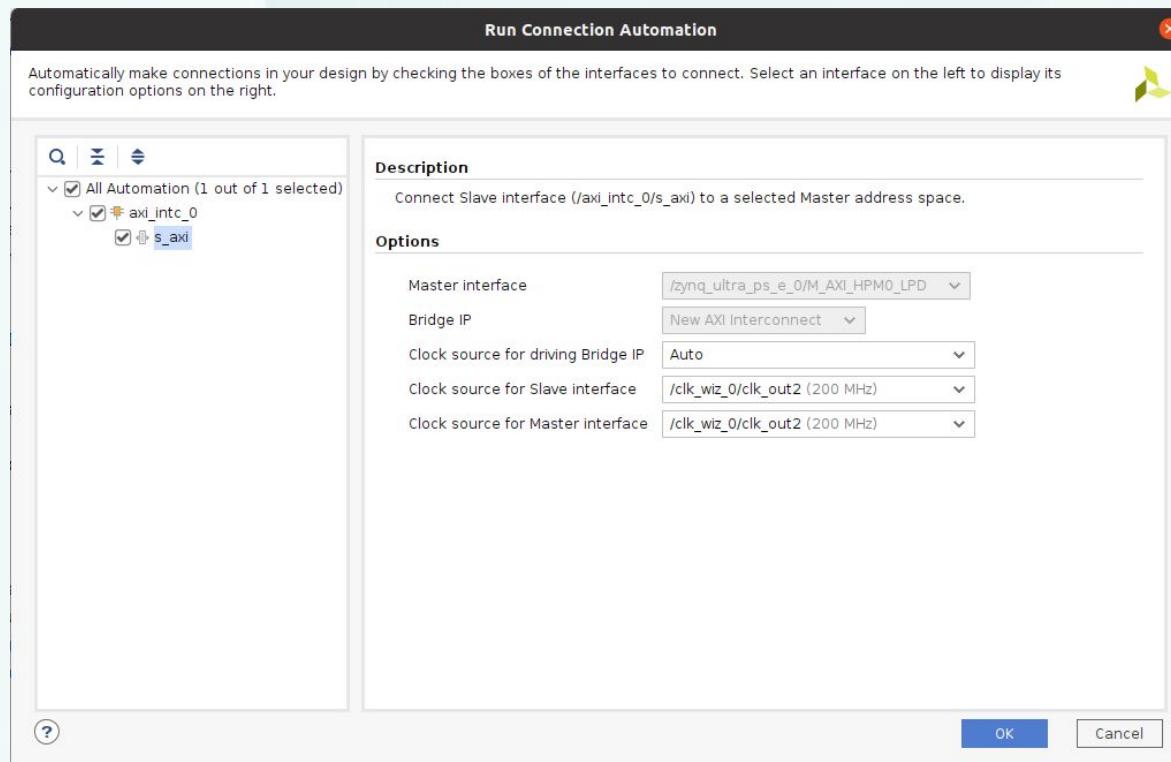
- We now should add interrupt support to our system:
  - Configure SoC block:



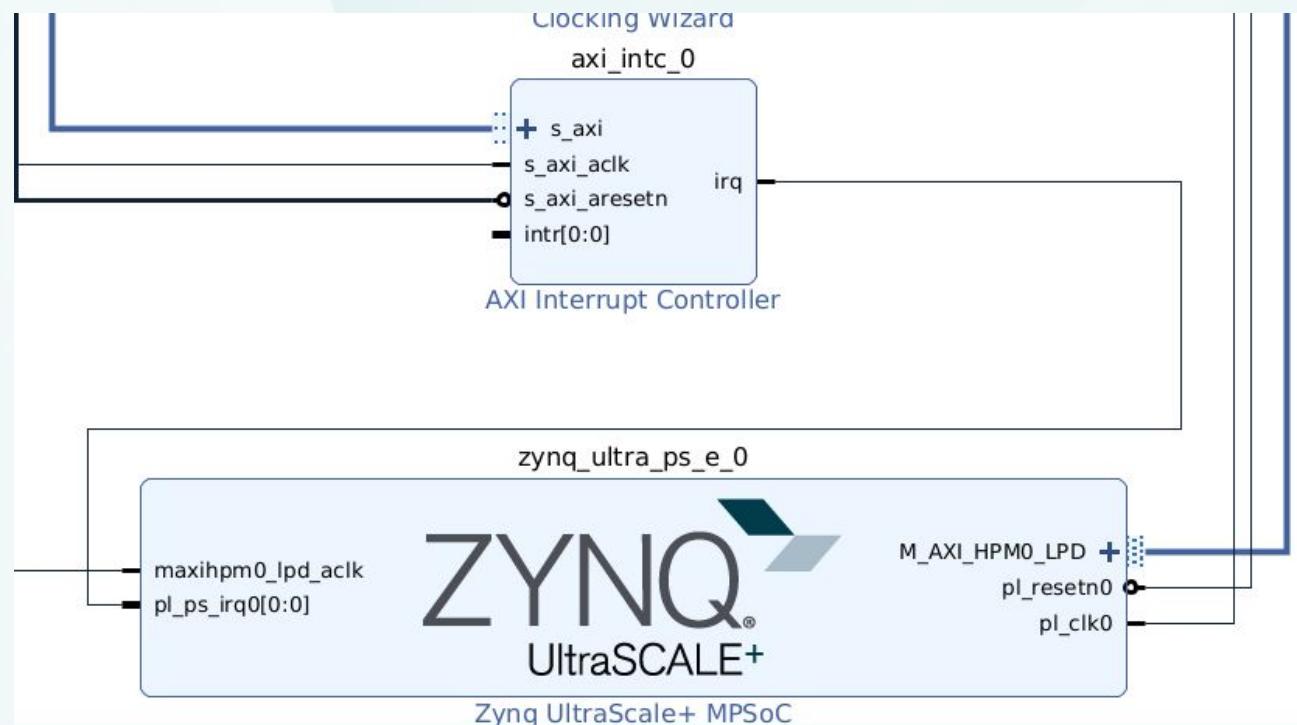
- We now should add interrupt support to our system:
  - Add AXI Interrupt controller:



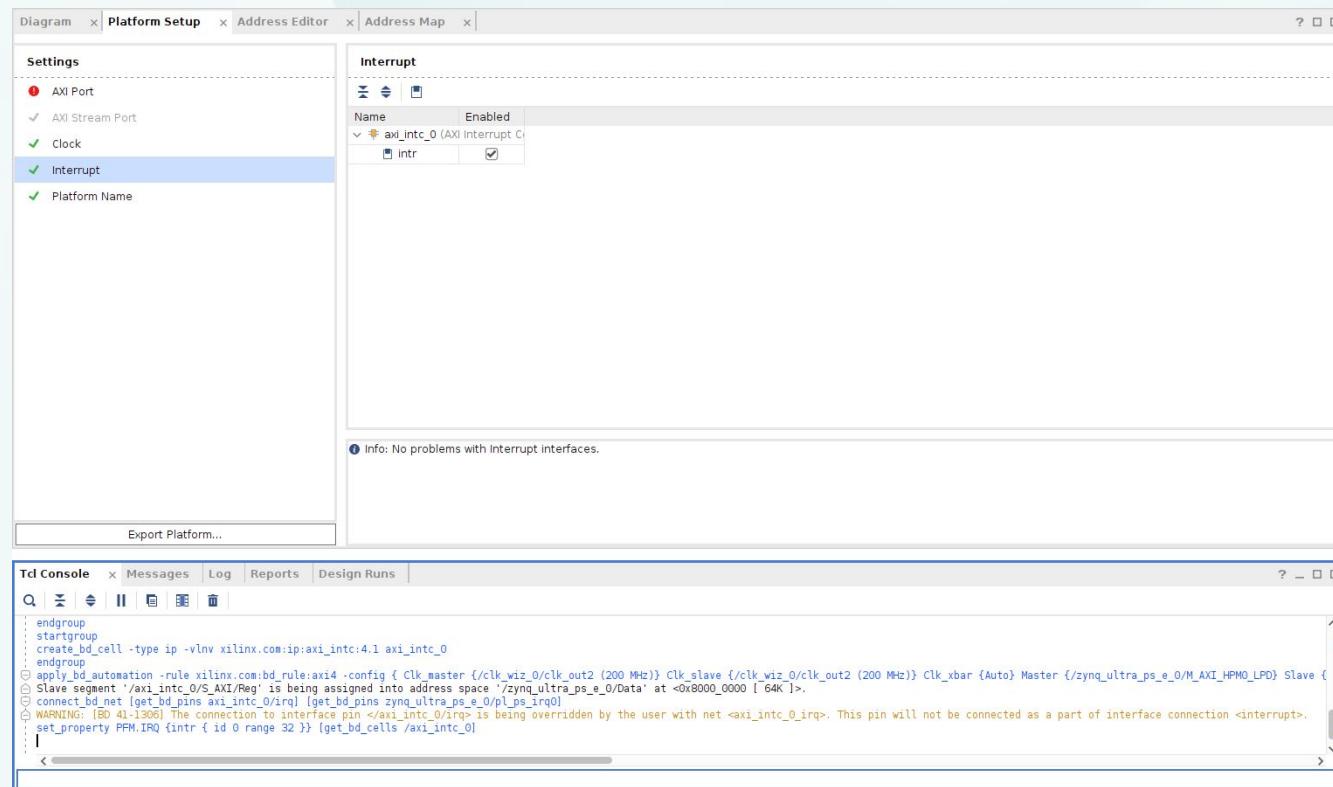
- We now should add interrupt support to our system:
  - Connect Interrupt interfaces (run connection automation):



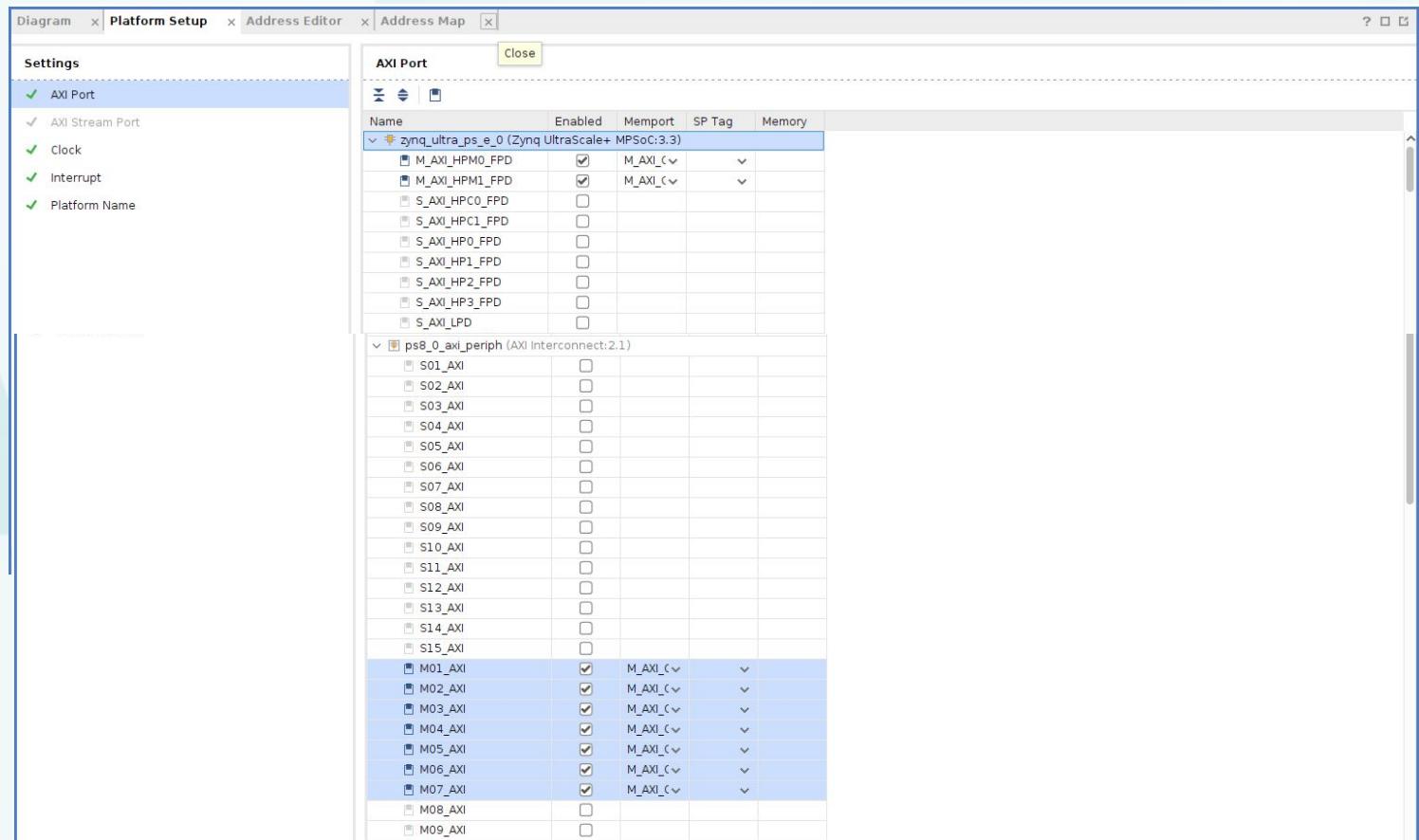
- We now should add interrupt support to our system:
  - Connect `xi_intc_o.irq` to `zynq_ultra_ps_e_o.pl_ps_irq[0:0]`:



- We now should add interrupt support to our system:
  - Enable Interrupt signals from the platform:



- It's time to enable the AXI interfaces:
  - Enable the following settings on the platform setup tab:

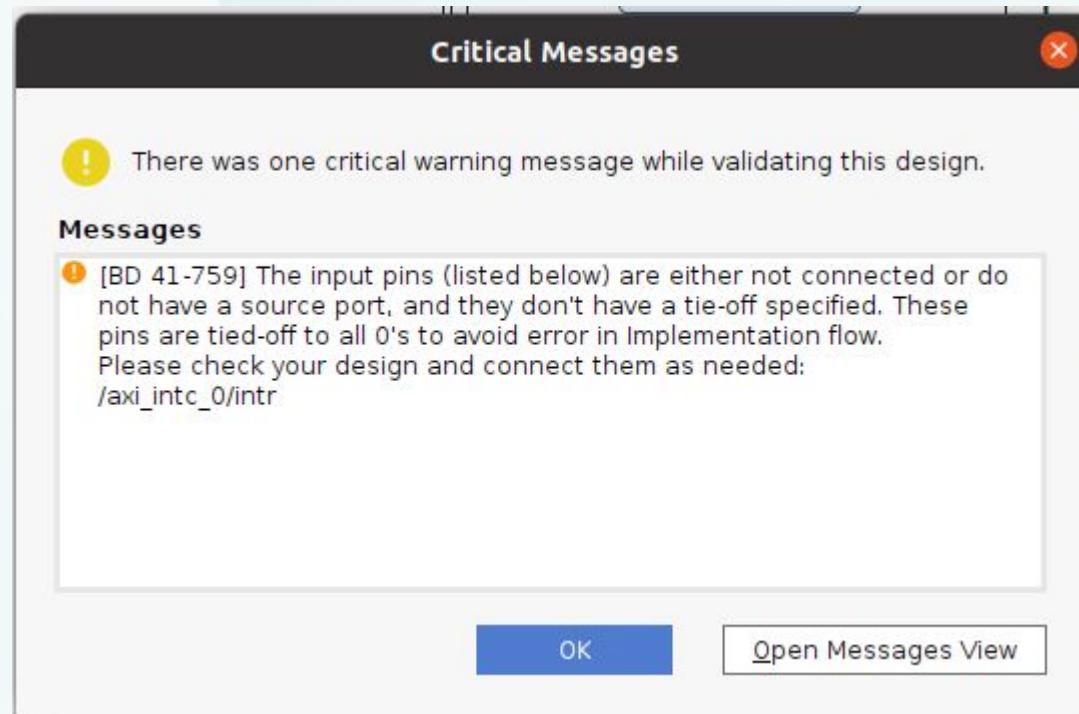


- It's time to enable the AXI interfaces:
  - Add the following SP tags:

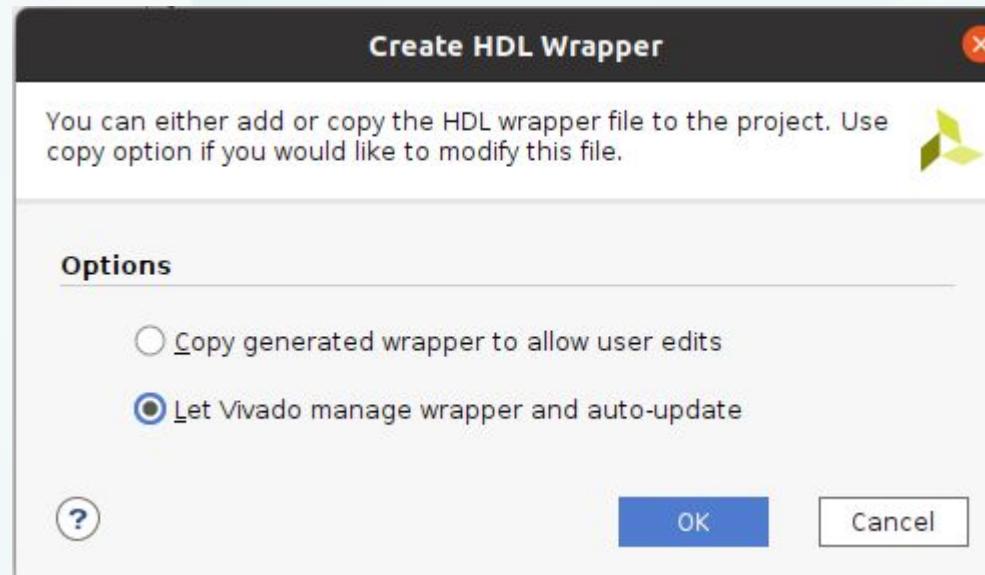
Name	Enabled	Memport	SP Tag	Memory
zynq_ultra_ps_e_0 (Zynq UltraScale+ MPSoC:3.3)				
M_AXI_HPM0_FPD	<input checked="" type="checkbox"/>	M_AXI_GP	▼	▼
M_AXI_HPM1_FPD	<input checked="" type="checkbox"/>	M_AXI_GP	▼	▼
S_AXI_HPC0_FPD	<input checked="" type="checkbox"/>	S_AXI_HP	▼	HPC0 ▼
S_AXI_HPC1_FPD	<input checked="" type="checkbox"/>	S_AXI_HP	▼	HPC1 ▼
S_AXI_HP0_FPD	<input checked="" type="checkbox"/>	S_AXI_HP	▼	HP0 ▼
S_AXI_HP1_FPD	<input checked="" type="checkbox"/>	S_AXI_HP	▼	HP1 ▼
S_AXI_HP2_FPD	<input checked="" type="checkbox"/>	S_AXI_HP	▼	HP2 ▼
S_AXI_HP3_FPD	<input checked="" type="checkbox"/>	S_AXI_HP	▼	HP3 ▼
S_AXI_LPD	<input type="checkbox"/>			

- Finally, it's time to export our design:
  - Validate the design:

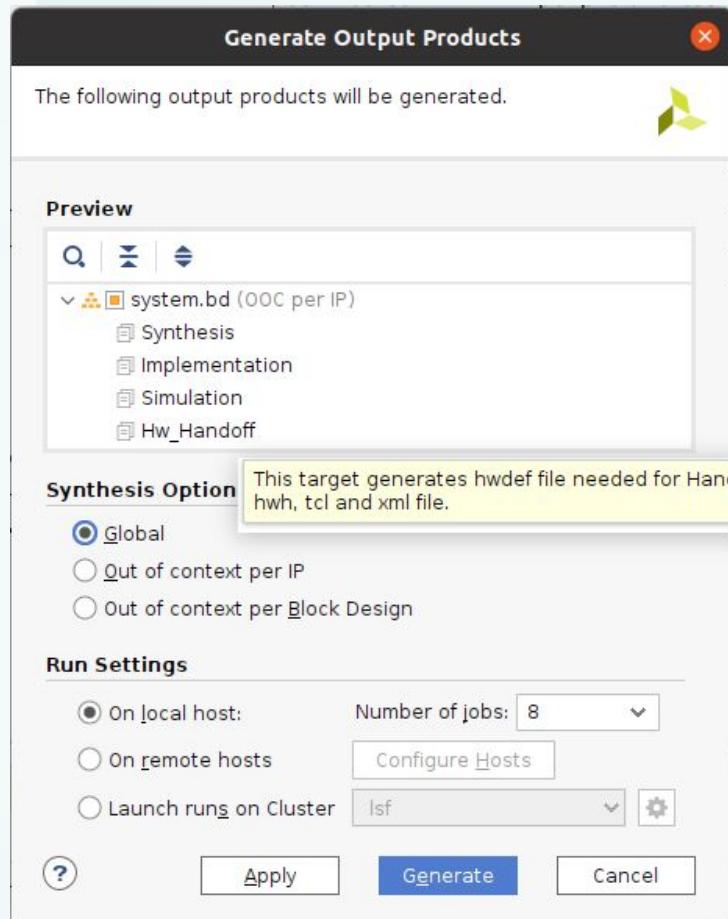
The following error message can be ignored:



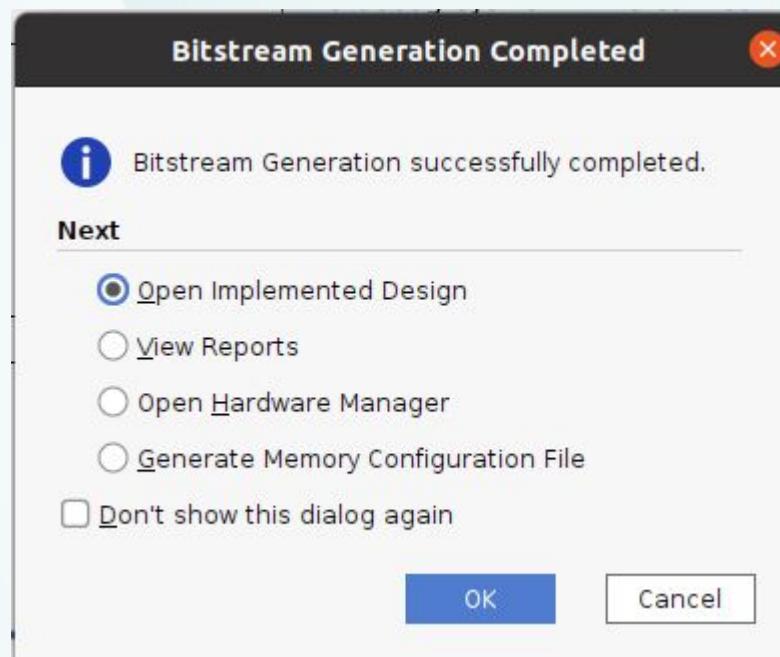
- Finally, it's time to export our design:  
-Create top module wrapper:  
Right-click system.bd and create HDL Wrapper:



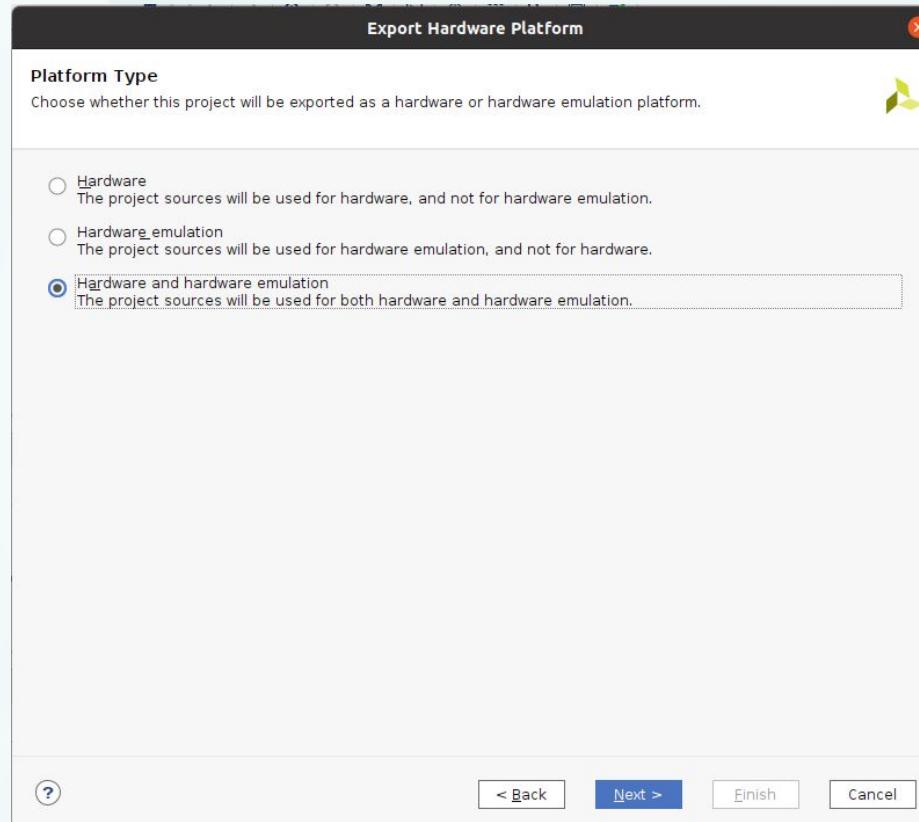
- Finally, it's time to export our design:  
**-Generate pre-synth design -> Generate Block Design**



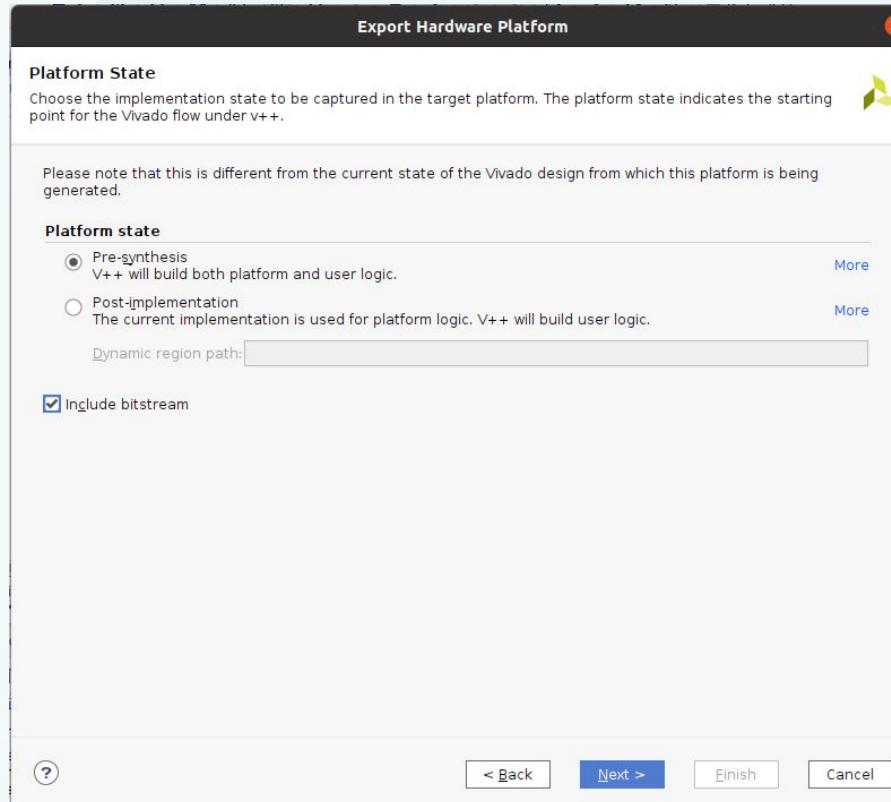
- Finally, it's time to export our design:  
-Generate Bitstream;
- We should see this:



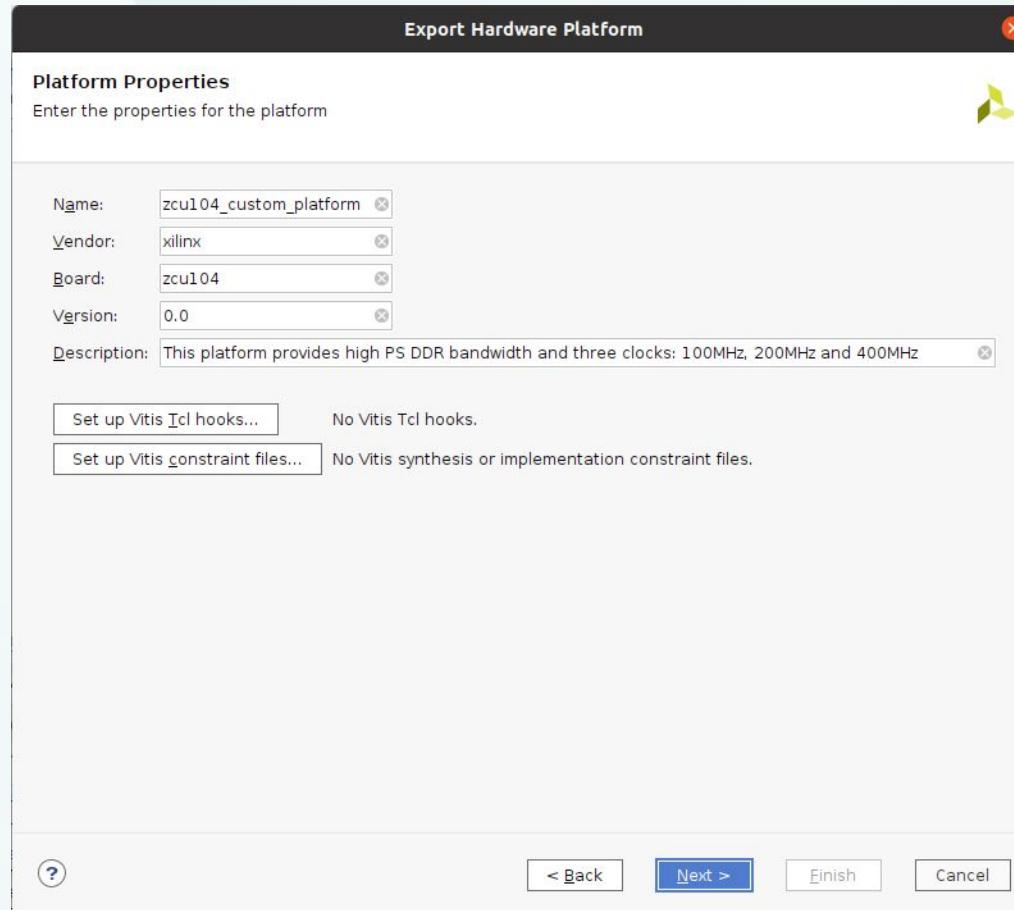
- Finally, it's time to export our design:  
-Navigate to File > Export > Export Platform



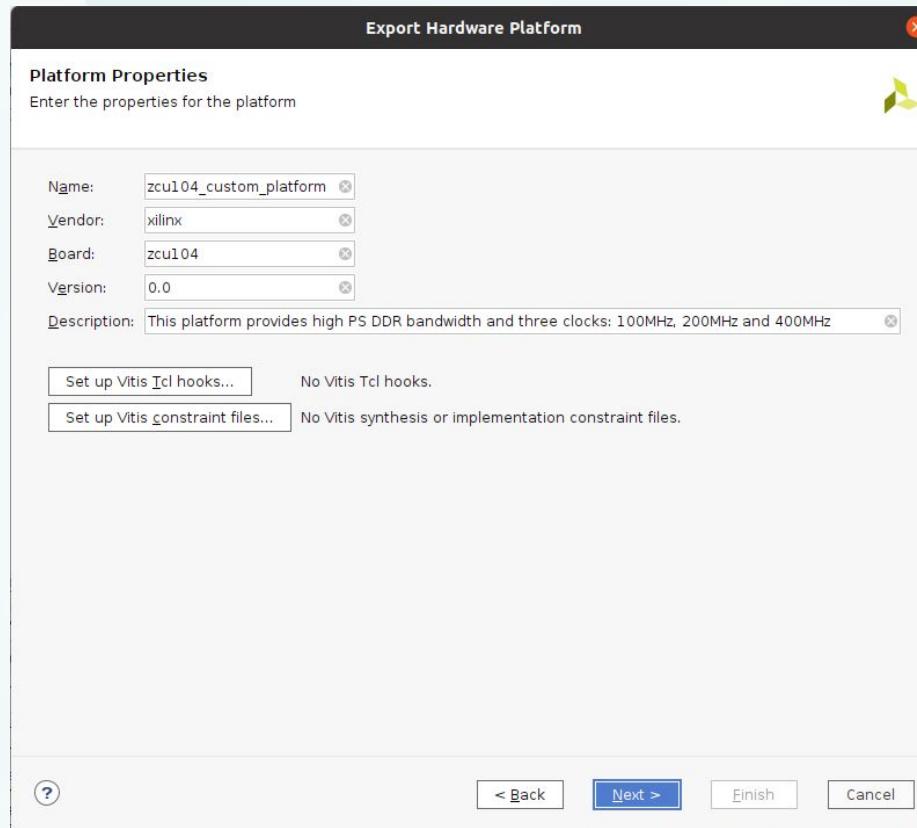
- Finally, it's time to export our design:  
-Enable "Include Bitstream":



- Finally, it's time to export our design:  
-Fill the information and click next:



- Finally, it's time to export our design:  
-Fill the information and click finish:



- For the raspberry PI 4, buildroot was used to create a custom linux image
- For the ZCU104 board, It will be used Petalinux. It is a Xilinx tool that allows to create embedded Linux systems based on your customized hardware in Xilinx FPGAs and SoCs.
- Important to mention that PetaLinux is totally free
- To install it, you must follow the next steps.

- Download Petalinux installer in the following link
  - <https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools.html>

Version

2021.2

2021.1

2020.3

Archive

PetaLinux Tools - Installer - 2021.2 Full Product Installation

**Important Information**

The PetaLinux Tools installer is downloaded using the below link. The installer checks for the required host machine package requirements followed by license acceptance from the user. It can be installed in any desired path. Note: All BSPs (located below) require the PetaLinux Tools to be installed first.

 [PetaLinux 2021.2 Installer \(TAR/GZIP - 2.1 GB\)](#)

MD5 SUM Value : fc307268822433258b006b7be02bf78e

Download Type

Full Product Installation

Last Updated

Oct 27, 2021

Answers

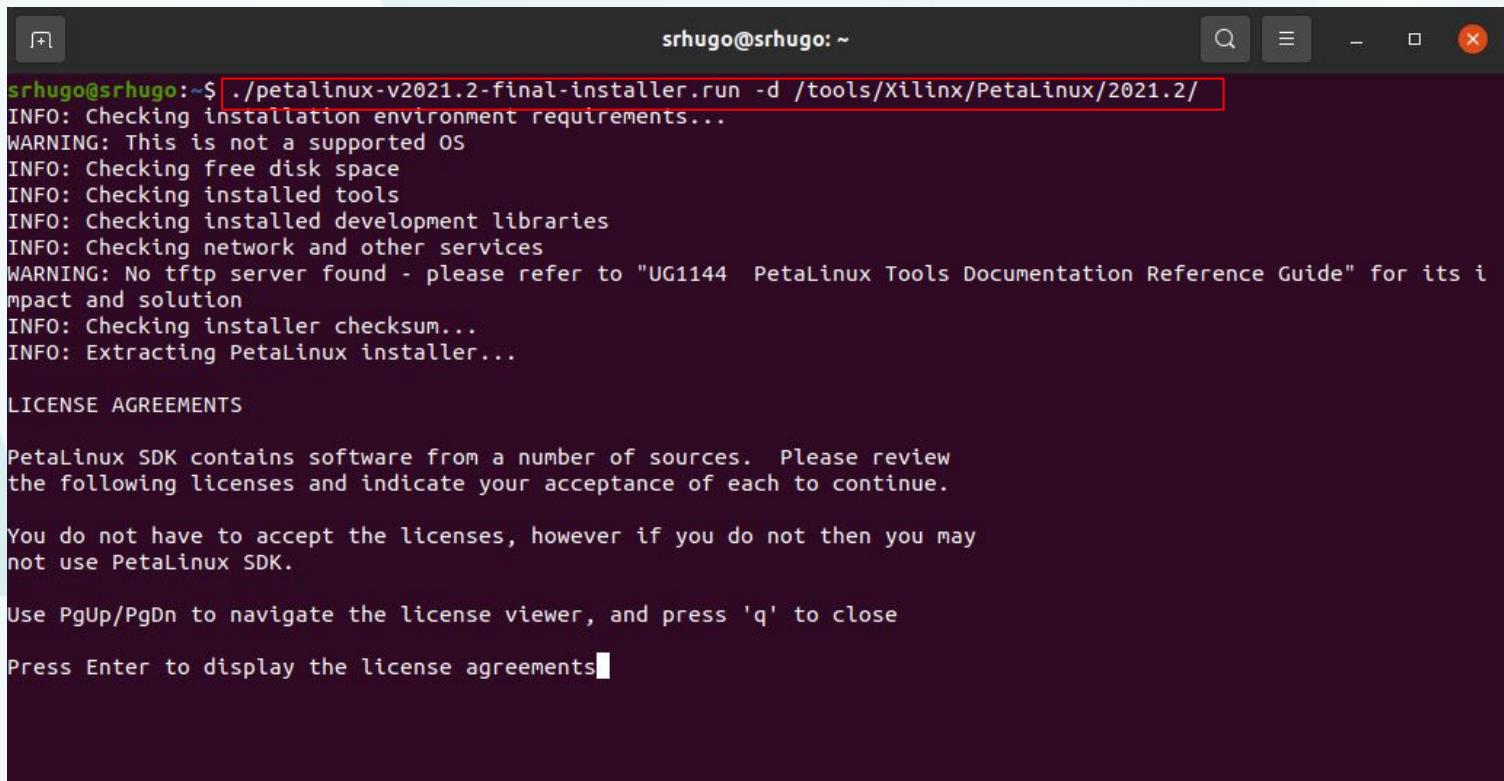
[Release Notes and Known Issues](#)

Documentation

[PetaLinux Tools Documentation](#)

[Installer Information](#)

- After download it, execute the installer by typing the following command. You should get the following output.



```
srhugo@srhugo:~$ ./petalinux-v2021.2-final-installer.run -d /tools/Xilinx/PetaLinux/2021.2/
INFO: Checking installation environment requirements...
WARNING: This is not a supported OS
INFO: Checking free disk space
INFO: Checking installed tools
INFO: Checking installed development libraries
INFO: Checking network and other services
WARNING: No tftp server found - please refer to "UG1144 PetaLinux Tools Documentation Reference Guide" for its impact and solution
INFO: Checking installer checksum...
INFO: Extracting PetaLinux installer...

LICENSE AGREEMENTS

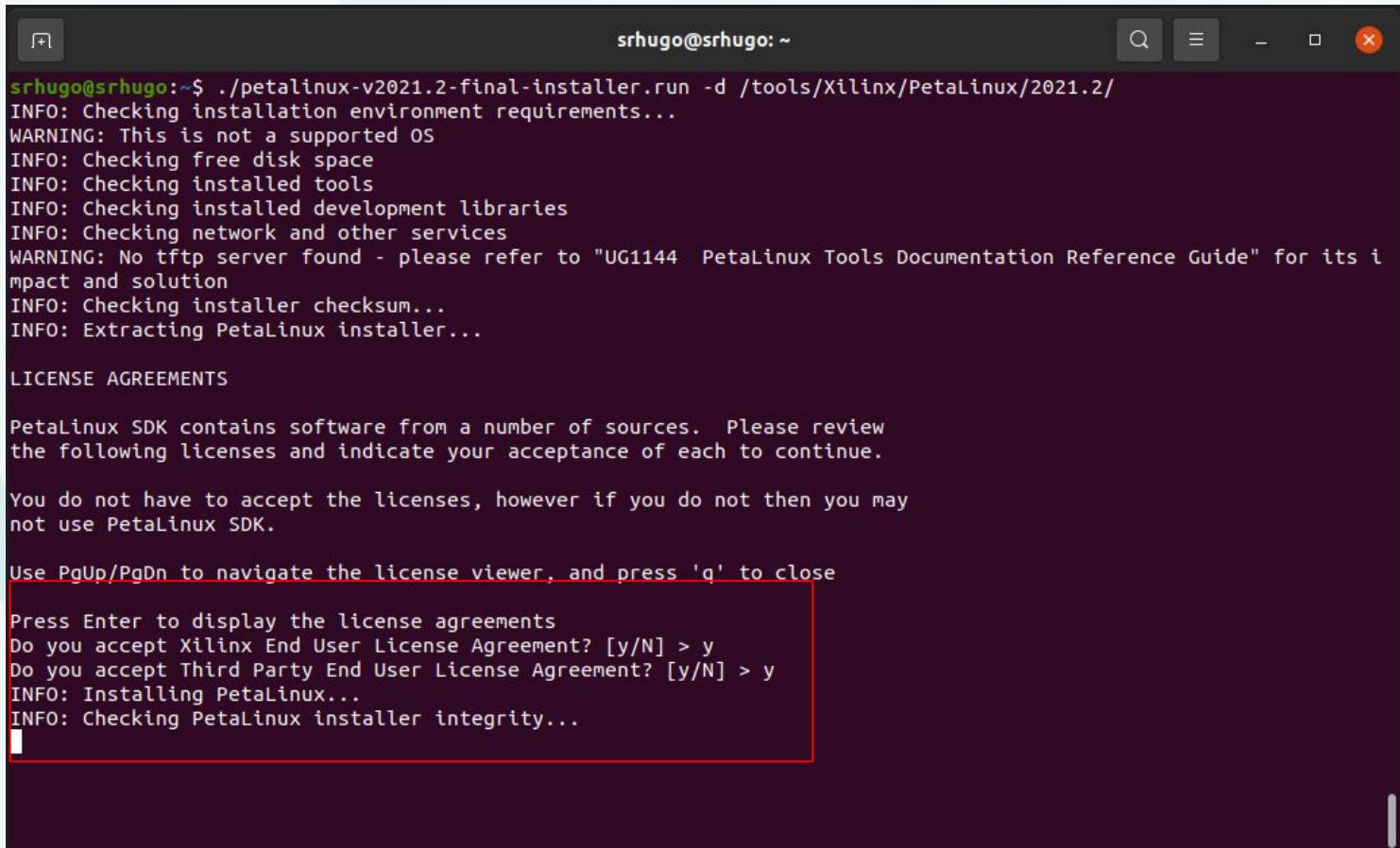
PetaLinux SDK contains software from a number of sources. Please review
the following licenses and indicate your acceptance of each to continue.

You do not have to accept the licenses, however if you do not then you may
not use PetaLinux SDK.

Use PgUp/PgDn to navigate the license viewer, and press 'q' to close

Press Enter to display the license agreements■
```

- You must accept all the license agreements. After that, Petalinux should start the installation



The screenshot shows a terminal window titled "srhugo@srhugo: ~". The window contains the following text:

```
srhugo@srhugo:~$ ./petalinux-v2021.2-final-installer.run -d /tools/Xilinx/PetaLinux/2021.2/
INFO: Checking installation environment requirements...
WARNING: This is not a supported OS
INFO: Checking free disk space
INFO: Checking installed tools
INFO: Checking installed development libraries
INFO: Checking network and other services
WARNING: No tftp server found - please refer to "UG1144 PetaLinux Tools Documentation Reference Guide" for its impact and solution
INFO: Checking installer checksum...
INFO: Extracting PetaLinux installer...

LICENSE AGREEMENTS

PetaLinux SDK contains software from a number of sources. Please review
the following licenses and indicate your acceptance of each to continue.

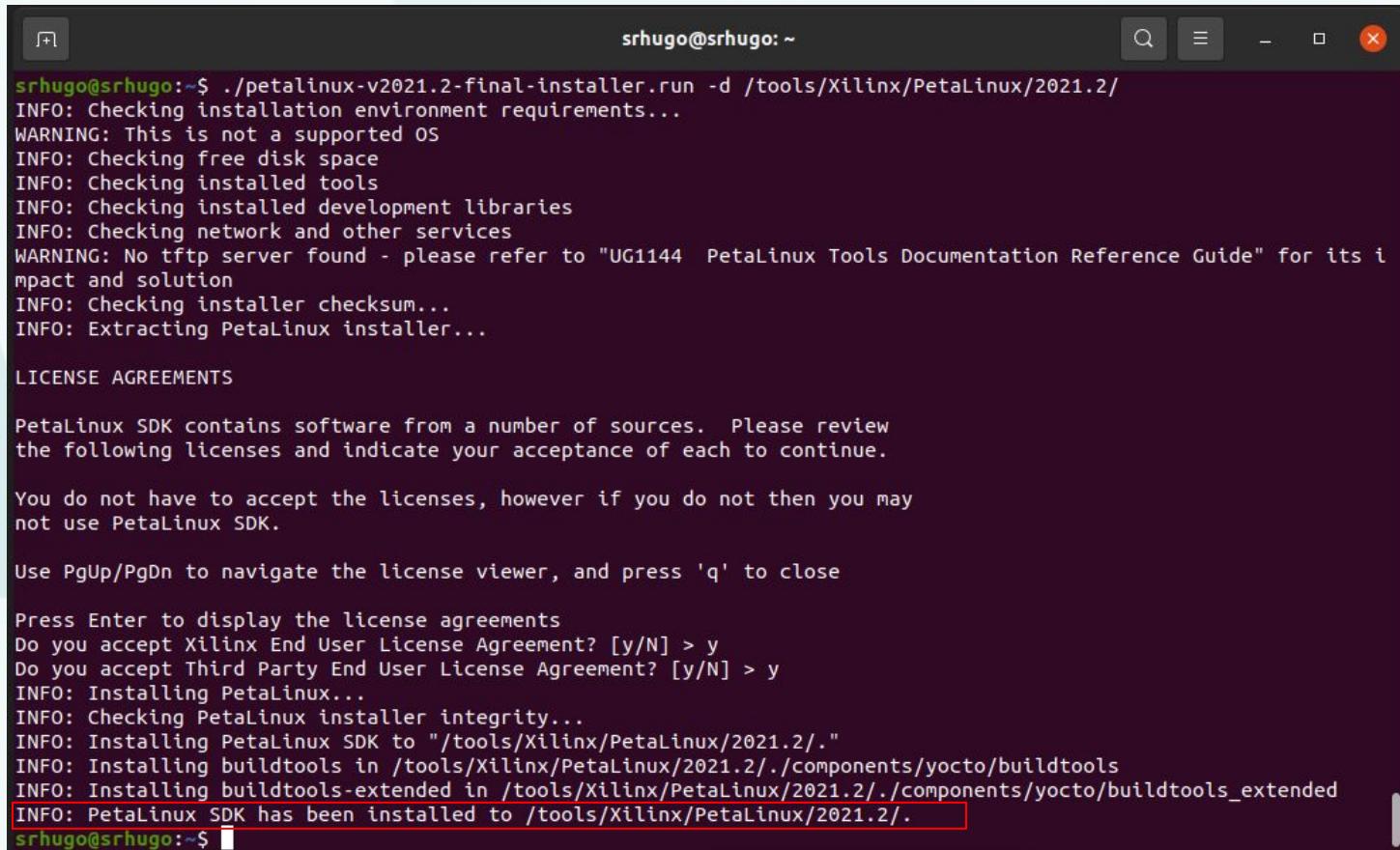
You do not have to accept the licenses, however if you do not then you may
not use PetaLinux SDK.

Use PgUp/PgDn to navigate the license viewer, and press 'q' to close

Press Enter to display the license agreements
Do you accept Xilinx End User License Agreement? [y/N] > y
Do you accept Third Party End User License Agreement? [y/N] > y
INFO: Installing PetaLinux...
INFO: Checking PetaLinux installer integrity...
```

A red rectangular box highlights the license acceptance prompt at the bottom of the terminal output.

- If everything went ok, you should receive the following output



```
srhugo@srhugo:~$ ./petalinux-v2021.2-final-installer.run -d /tools/Xilinx/PetaLinux/2021.2/
INFO: Checking installation environment requirements...
WARNING: This is not a supported OS
INFO: Checking free disk space
INFO: Checking installed tools
INFO: Checking installed development libraries
INFO: Checking network and other services
WARNING: No tftp server found - please refer to "UG1144 PetaLinux Tools Documentation Reference Guide" for its impact and solution
INFO: Checking installer checksum...
INFO: Extracting PetaLinux installer...

LICENSE AGREEMENTS

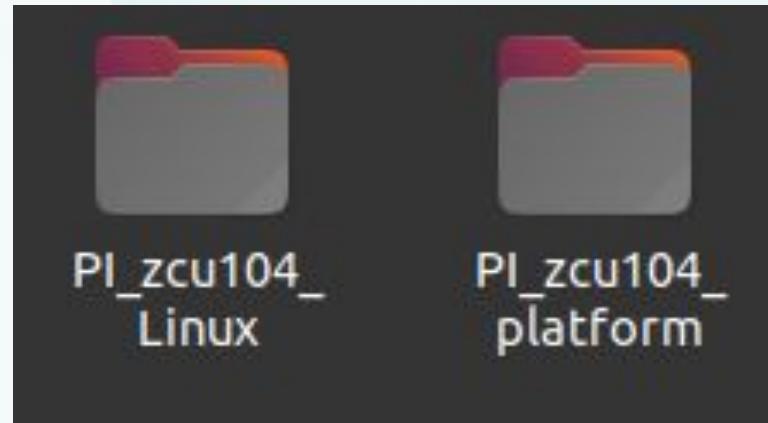
PetaLinux SDK contains software from a number of sources. Please review
the following licenses and indicate your acceptance of each to continue.

You do not have to accept the licenses, however if you do not then you may
not use PetaLinux SDK.

Use PgUp/PgDn to navigate the license viewer, and press 'q' to close

Press Enter to display the license agreements
Do you accept Xilinx End User License Agreement? [y/N] > y
Do you accept Third Party End User License Agreement? [y/N] > y
INFO: Installing PetaLinux...
INFO: Checking PetaLinux installer integrity...
INFO: Installing PetaLinux SDK to "/tools/Xilinx/PetaLinux/2021.2/."
INFO: Installing buildtools in /tools/Xilinx/PetaLinux/2021.2./components/yocto/buildtools
INFO: Installing buildtools-extended in /tools/Xilinx/PetaLinux/2021.2./components/yocto/buildtools_extended
INFO: PetaLinux SDK has been installed to /tools/Xilinx/PetaLinux/2021.2/.
srhugo@srhugo:~$
```

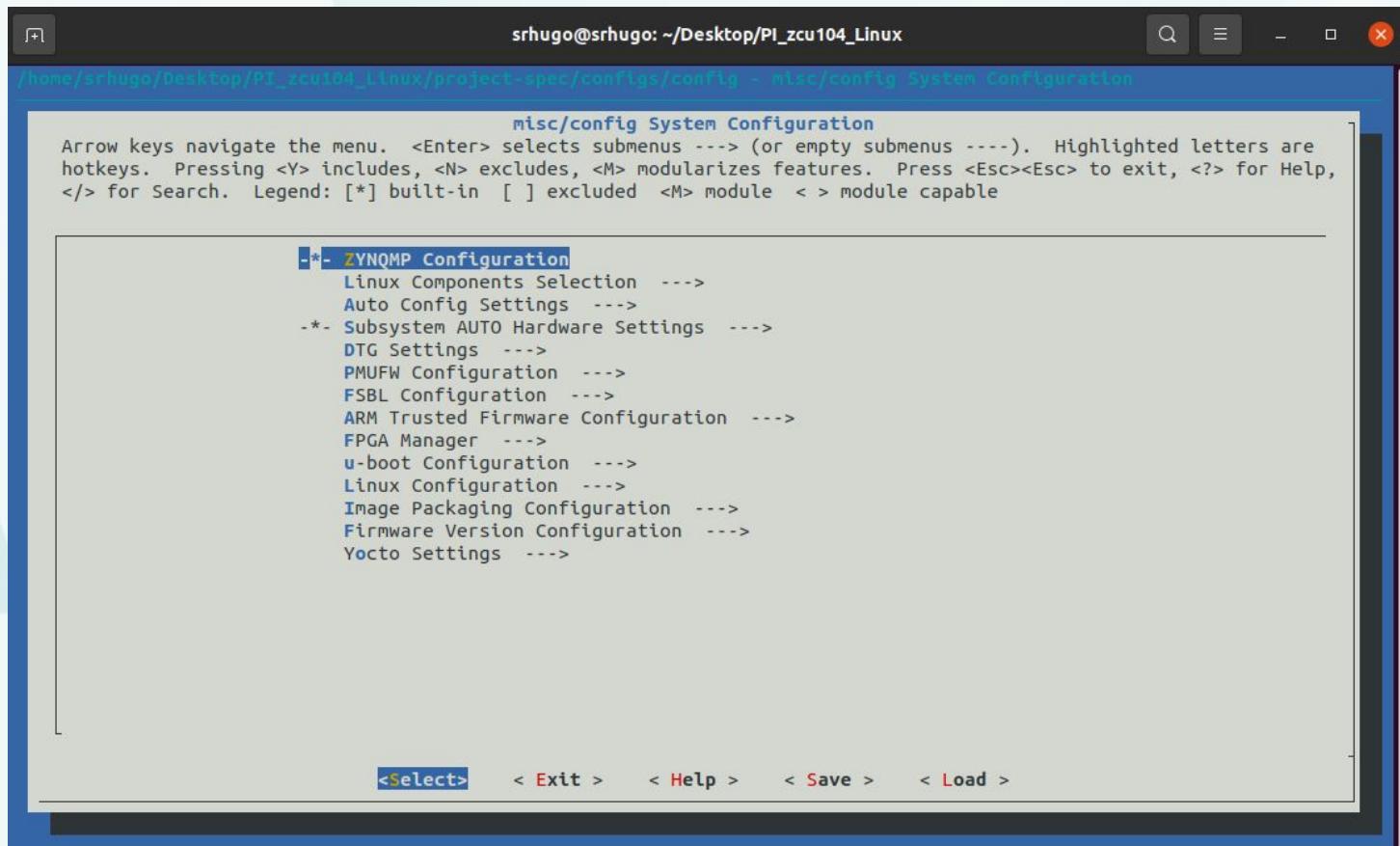
- Let's create a petalinux project
  - Create a folder to petalinux
  - Your directory hierarchy should be



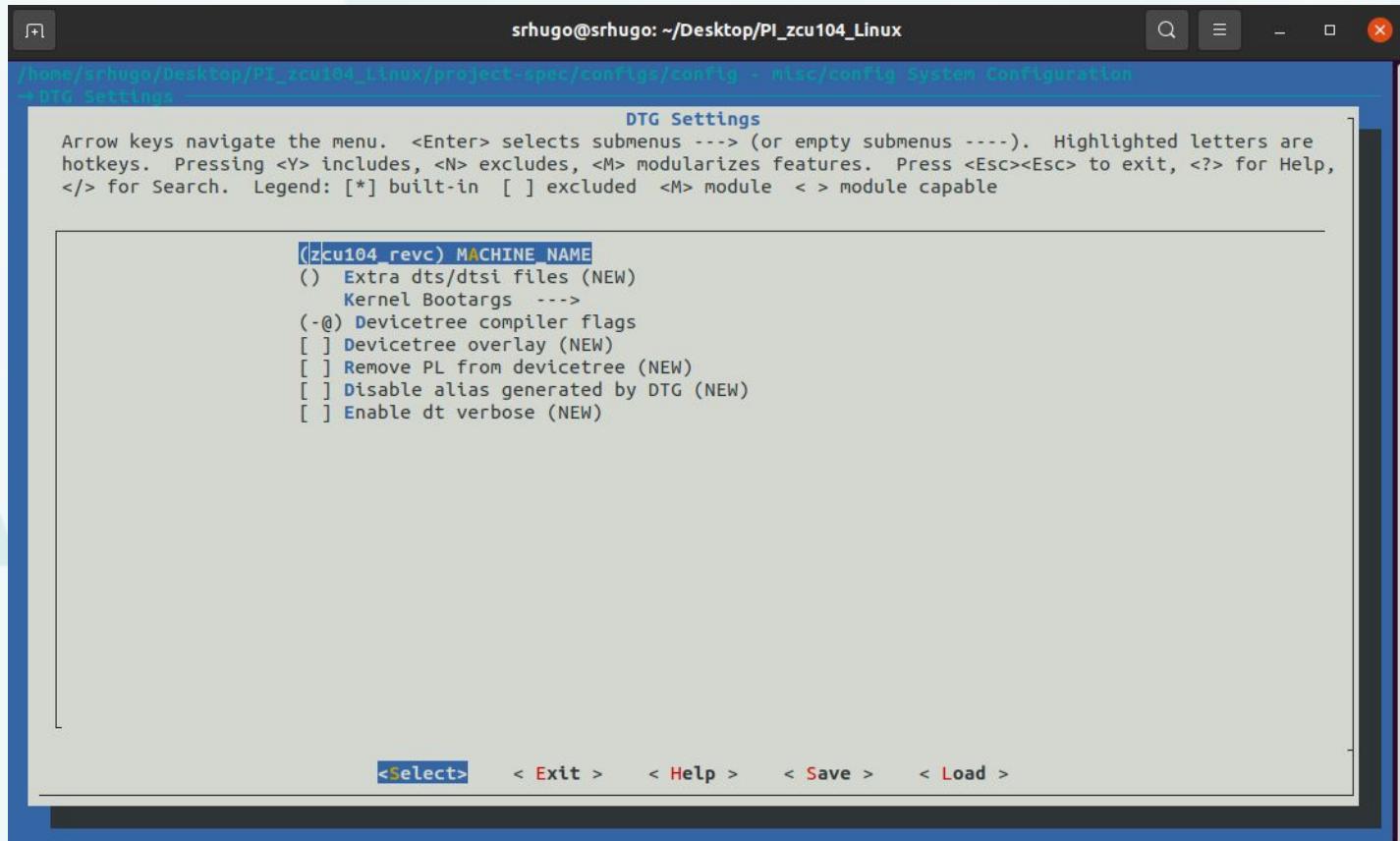
- Open the terminal in the petalinux folder
  - Execute the following commands
  - The petalinux command must receive the created .XSA file as parameter

```
srhugo@srhugo:~/Desktop$ petalinux-create --type project --template zynqMP --name PI_zcu104_Linux
INFO: Create project: PI_zcu104_Linux
INFO: New project successfully created in /home/srhugo/Desktop/PI_zcu104_Linux
srhugo@srhugo:~/Desktop$ cd PI_zcu104_Linux/
srhugo@srhugo:~/Desktop/PI_zcu104_Linux$ petalinux-config --get-hw-description=~/Desktop/Projeto_Integrador/PI_zcu104_platf
orm/PI_zcu104.xsa
[INFO] Sourcing buildtools
INFO: Getting hardware description...
INFO: Renaming PI_zcu104.xsa to system.xsa
[INFO] Generating Kconfig for project
```

- The following menu should appear

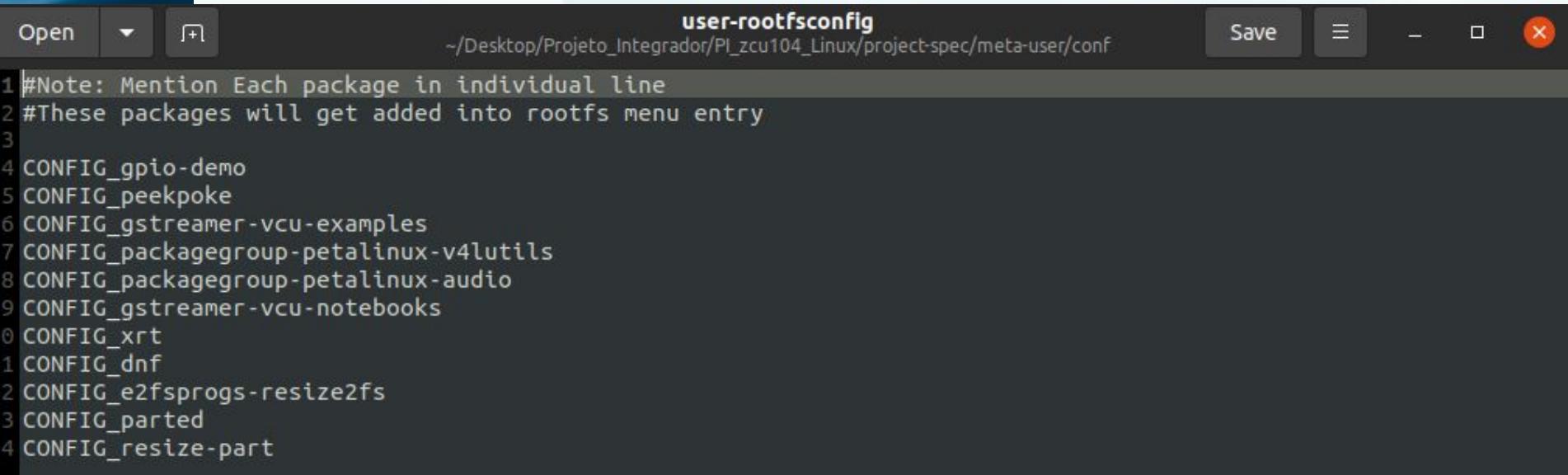


- Go to DTG Setting -> MACHINE\_NAME and change the name to **zcu104\_revC**



- Save the changes and exit
- Now let's add so packages to your petalinux image
- There are two ways
  - Add manually the packages in user-rootfsconfig file
  - Add in petalinux's menu
- Let's try the first way
- Go to  
<your\_petalinux\_project\_dir>/project-spec/meta-user/conf/user-rootfsconfig
  - CONFIG\_dnf
  - CONFIG\_e2fsprogs-resize2fs
  - CONFIG\_parted
  - CONFIG\_resize-part
  - CONFIG\_xrt

- Copy the previous topics and paste into user-rootfsconfig file

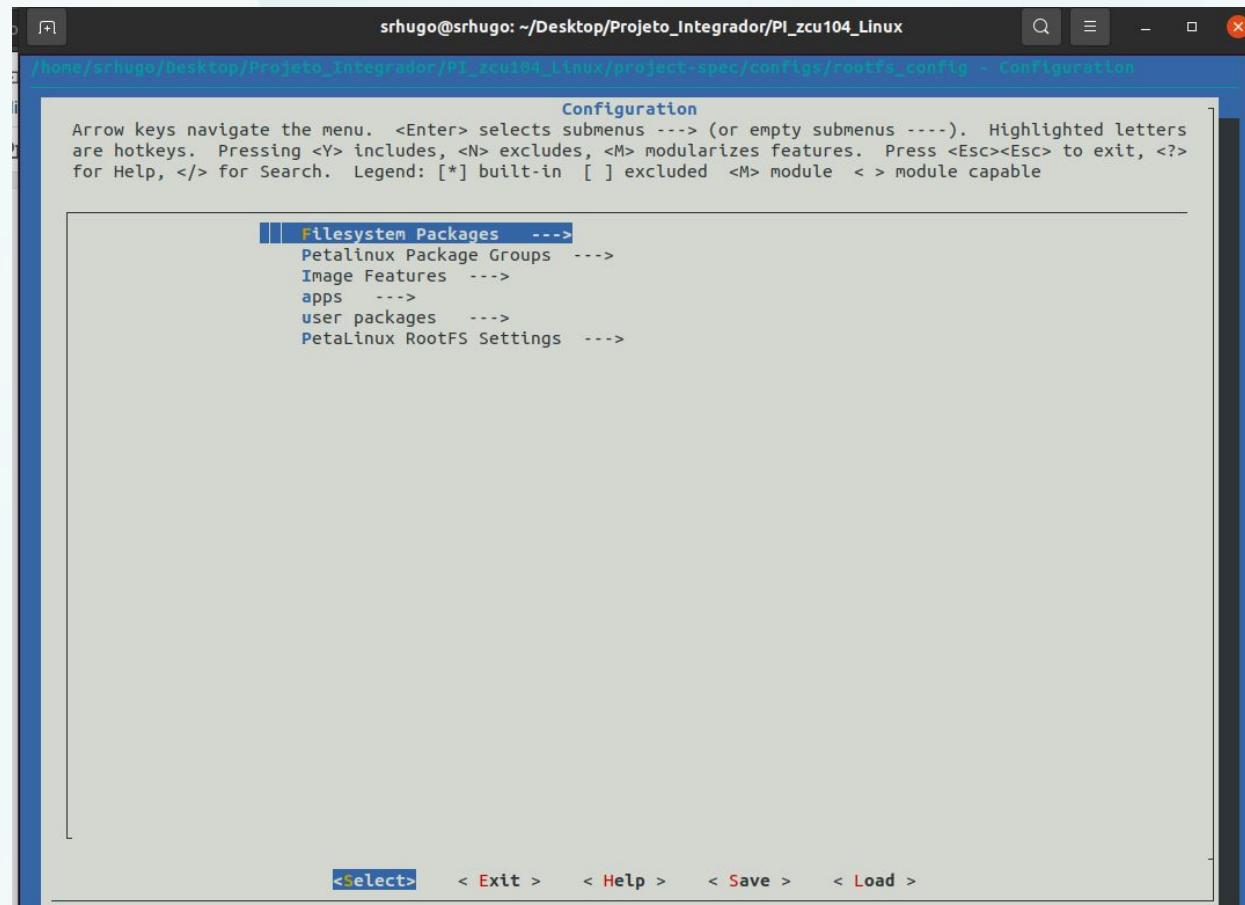


The screenshot shows a code editor window titled "user-rootfsconfig" with the file path "~/Desktop/Projeto\_Integrador/PI\_zcu104\_Linux/project-spec/meta-user/conf". The editor has standard window controls (Save, Minimize, Maximize, Close) at the top right. On the left, there are "Open" and "Save" buttons. The code itself is a configuration file with the following content:

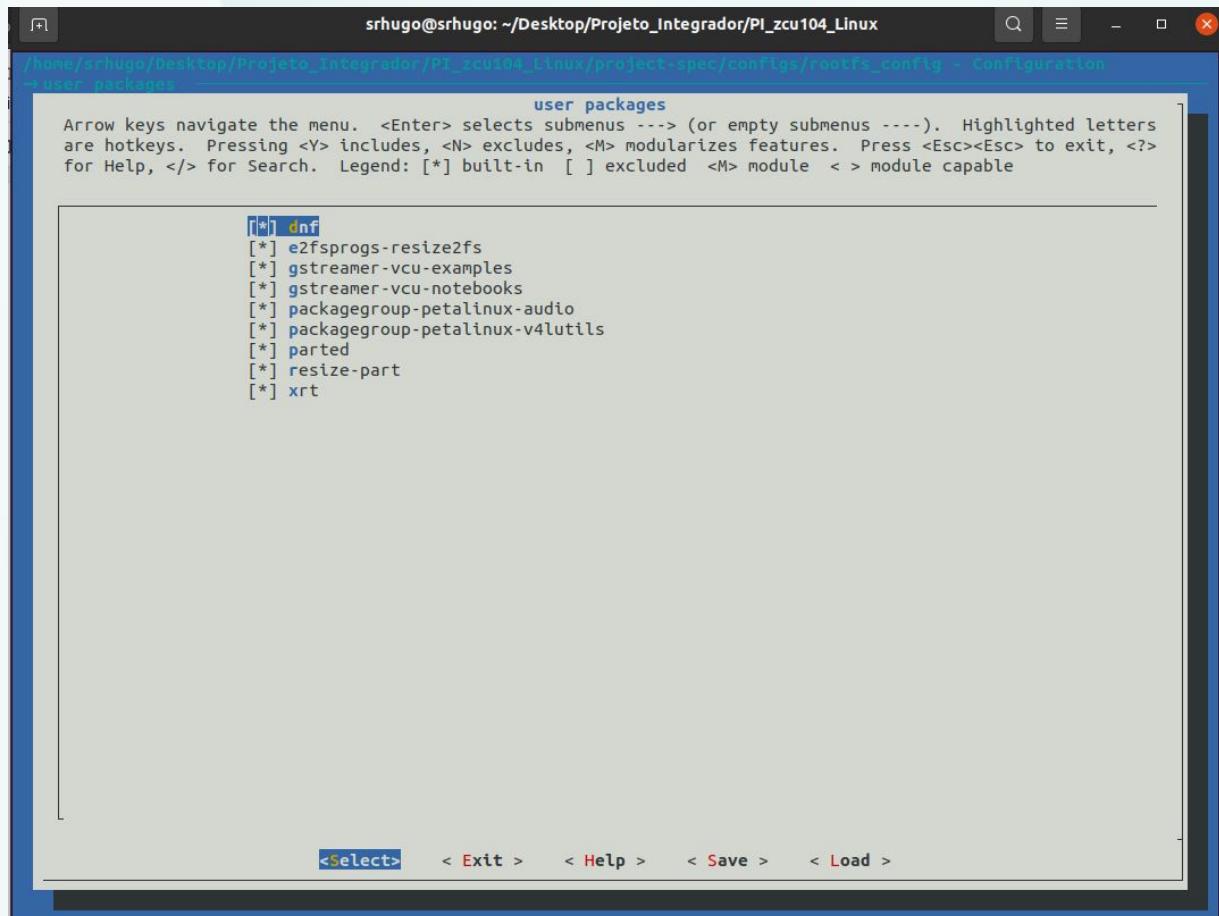
```
1 #Note: Mention Each package in individual line
2 #These packages will get added into rootfs menu entry
3
4 CONFIG_gpio-demo
5 CONFIG_peekpoke
6 CONFIG_gstreamer-vcu-examples
7 CONFIG_packagegroup-petalinux-v4lutils
8 CONFIG_packagegroup-petalinux-audio
9 CONFIG_gstreamer-vcu-notebooks
0 CONFIG_xrt
1 CONFIG_dnf
2 CONFIG_e2fsprogs-resize2fs
3 CONFIG_parted
4 CONFIG_resize-part
```

Save and close it

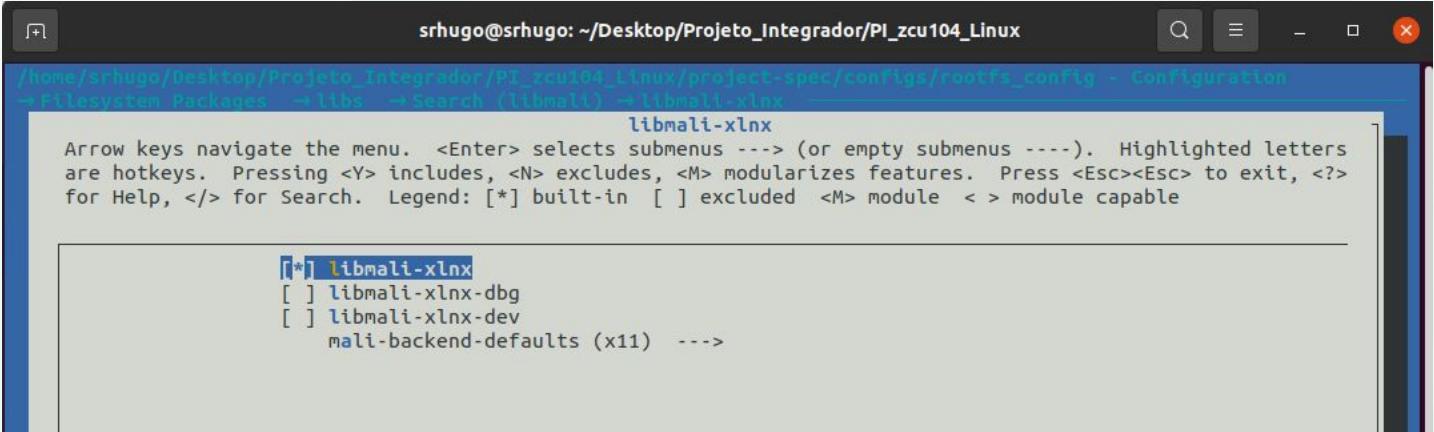
- Go back to your project folder and execute the following command
  - `petalinux-config -c rootfs`



- In user packages activate all packages.

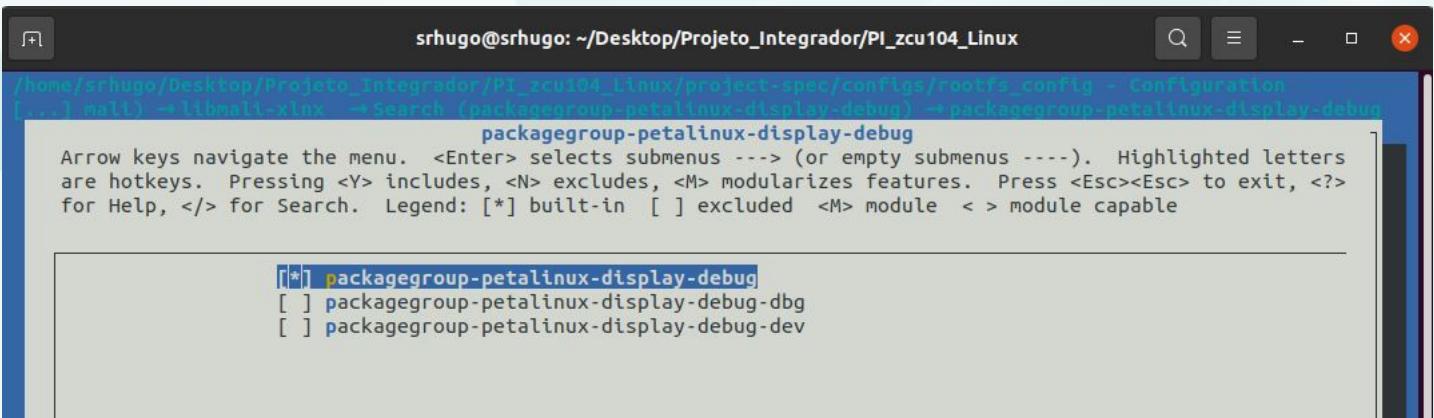


- Let's add also packages for opencv support
  - They will be used in PrHa program



```
srhugo@srhugo: ~/Desktop/Projeto_Integrador/PI_zcu104_Linux
/home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_Linux/project-spec/configs/rootfs_config - Configuration
→Filesystem Packages → libs → Search (libmali) → libmali-xlnx
libmali-xlnx
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < > module capable

[*] libmali-xlnx
[ ] libmali-xlnx-dbg
[ ] libmali-xlnx-dev
    mali-backend-defaults (x11) --->
```



```
srhugo@srhugo: ~/Desktop/Projeto_Integrador/PI_zcu104_Linux
/home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_Linux/project-spec/configs/rootfs_config - Configuration
[...] mali) → libmali-xlnx → Search (packagegroup-petalinux-display-debug) → packagegroup-petalinux-display-debug
packagegroup-petalinux-display-debug
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < > module capable

[*] packagegroup-petalinux-display-debug
[ ] packagegroup-petalinux-display-debug-dbg
[ ] packagegroup-petalinux-display-debug-dev
```

```
srhugo@srhugo: ~/Desktop/Projeto_Integrador/PI_zcu104_Linux
/home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_Linux/project-spec/configs/rootfs_config - Configuration
[...] packagegroup-petalinux-display-debug → Search (packagegroup-petalinux-opencv) → packagegroup-petalinux-opencv
    packagegroup-petalinux-opencv
    Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < > module capable

    [*] packagegroup-petalinux-opencv
    [*] packagegroup-petalinux-opencv-dev
    [ ] packagegroup-petalinux-opencv-dbg

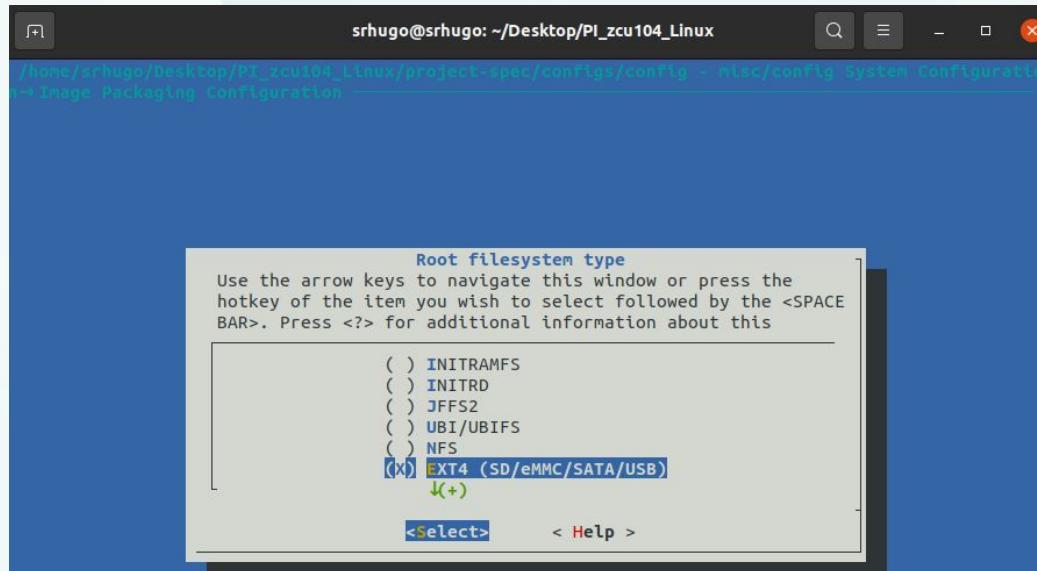
srhugo@srhugo: ~/Desktop/Projeto_Integrador/PI_zcu104_Linux
/home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_Linux/project-spec/configs/rootfs_config - Configuration
[...] packagegroup-petalinux-opencv → Search (packagegroup-petalinux-v4lutils) → packagegroup-petalinux-v4lutils
    packagegroup-petalinux-v4lutils
    Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < > module capable

    [*] packagegroup-petalinux-v4lutils
    [ ] packagegroup-petalinux-v4lutils-dbg
    [ ] packagegroup-petalinux-v4lutils-dev

srhugo@srhugo: ~/Desktop/Projeto_Integrador/PI_zcu104_Linux
/home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_Linux/project-spec/configs/rootfs_config - Configuration
[...] v4lutils → packagegroup-petalinux-v4lutils → Search (packagegroup-petalinux-x11) → packagegroup-petalinux-x11
    packagegroup-petalinux-x11
    Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < > module capable

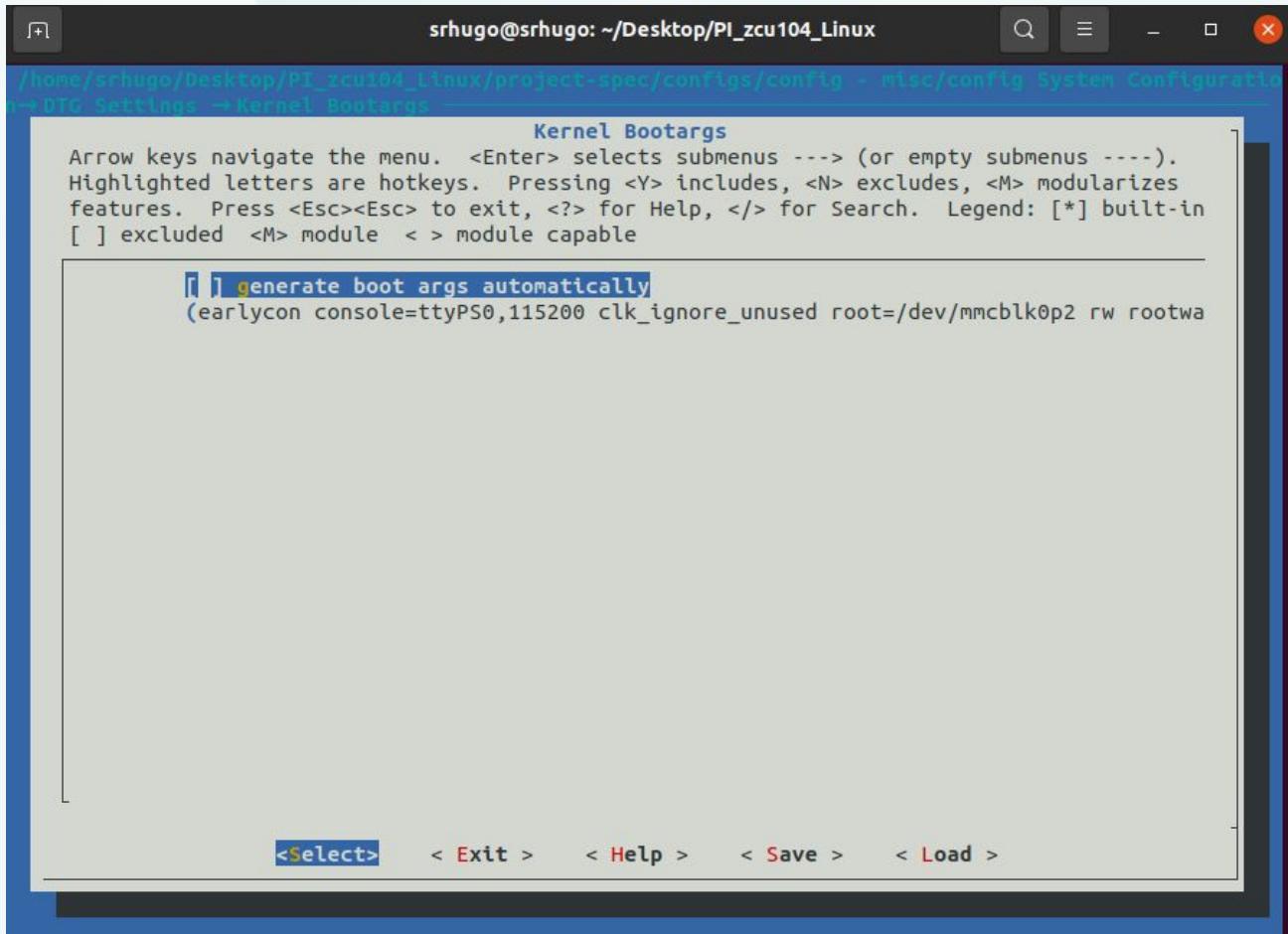
    [*] packagegroup-petalinux-x11
    [*] packagegroup-petalinux-x11-dev
    [ ] packagegroup-petalinux-x11-dbg
```

- Save the changes and exit
- Type petalinux-config on the console
- Then Image Packaging Configuration -> Root filesystem type
  - Select this option



- Go back to the first page and go to the following directory
  - DTG settings -> Kernel Bootargs
- Set the generate boot args automatically to NO
- Copy the following line to user set kernel bootargs option
  - `earlycon console=ttyPS0,115200 clk_ignore_unused root=/dev/mmcblk0p2 rw rootwait cma=512M`

- You should get the this output



The screenshot shows a terminal window titled "srhugo@srhugo: ~/Desktop/PI\_zcu104\_Linux". The window displays a configuration menu for kernel boot arguments. The title bar includes standard window controls (minimize, maximize, close) and a search bar. The main area shows a list of boot parameters with some being highlighted in blue. At the bottom, there is a footer with navigation keys: <Select>, < Exit >, < Help >, < Save >, and < Load >.

```
srhugo@srhugo: ~/Desktop/PI_zcu104_Linux
/home/srhugo/Desktop/PI_zcu104_Linux/project-spec/configs/config - Misc/config System Configuration
  ↵ DTG Settings → Kernel Bootargs
          Kernel Bootargs
Arrow keys navigate the menu. <Enter> selects submenus --- (or empty submenus ----).
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes
features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in
[ ] excluded <M> module < > module capable
[ ] generate boot args automatically
(earlycon console=ttyPS0,115200 clk_ignore_unused root=/dev/mmcblk0p2 rw rootwa
<Select>  < Exit >  < Help >  < Save >  < Load >
```

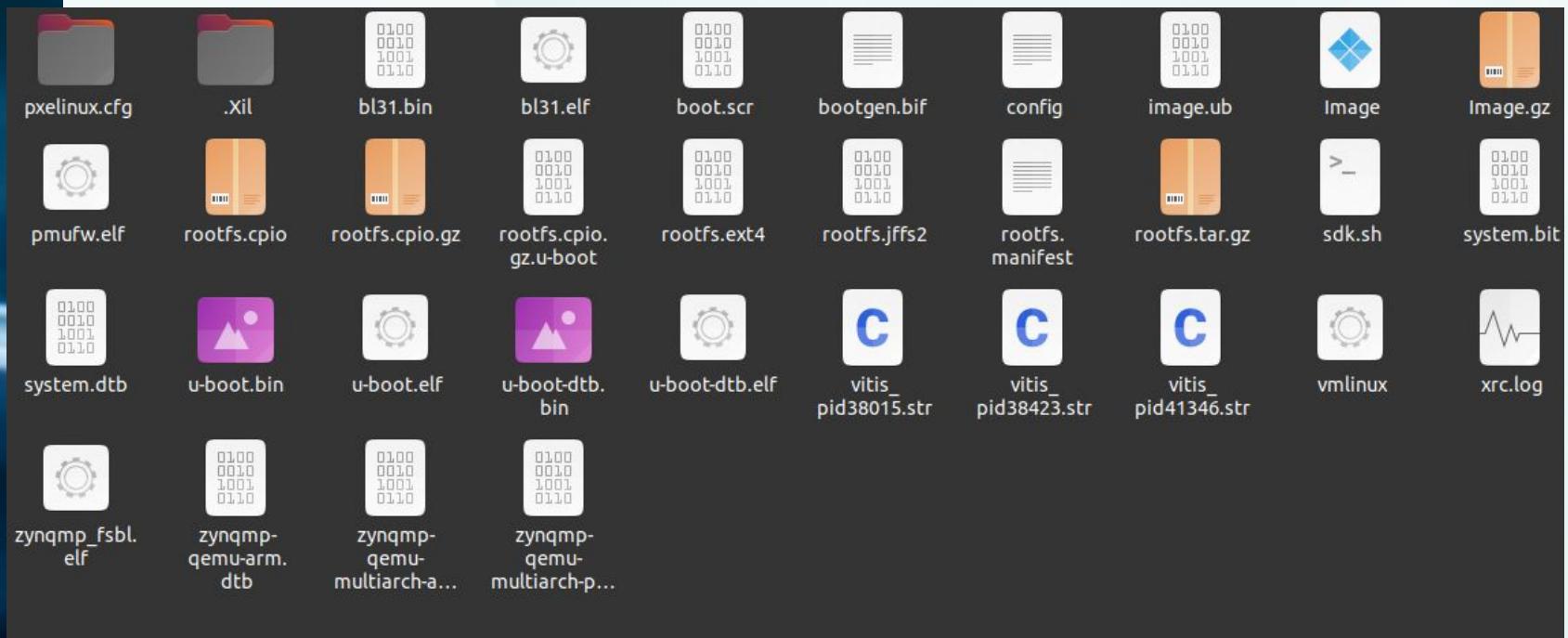
- Save all and exit
- Type petalinux-build and go take a coffee.



- After receive the build successful message, you should have the image create in images/linux
- To have the sysroot self-installer, you must run the following command
  - `petalinux-build --sdk`
- Go take another coffee :)



- After all, you should have this files in your image/linux folder



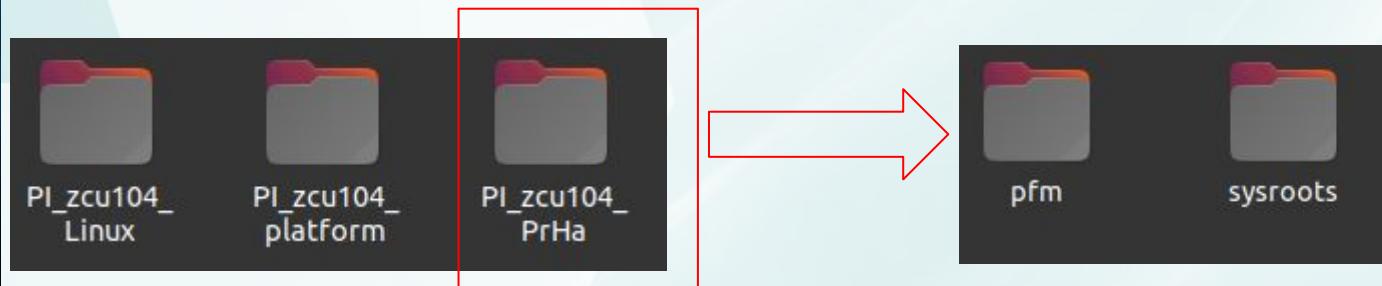
- Let's create a custom Vitis platform
- First create a folder for your platform
  - Inside that folder, create another and call it pfm
- Go to the images/linux folder and execute the sdk.sh script, with the application's directory

```
srhugo@srhugo:~/Desktop/Projeto_Integrador/PI_zcu104_PrHa$ mkdir pfm
srhugo@srhugo:~/Desktop/Projeto_Integrador/PI_zcu104_PrHa$ cd ../PI_zcu104_Linux/images/linux
srhugo@srhugo:~/Desktop/Projeto_Integrador/PI_zcu104_Linux/images/linux$ ./sdk.sh -d ../../PI_zcu104_PrHa
PetaLinux SDK installer version 2021.2
=====
You are about to install the SDK to "/home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_PrHa". Proceed [Y/n]? y
Extracting SDK.....[
```

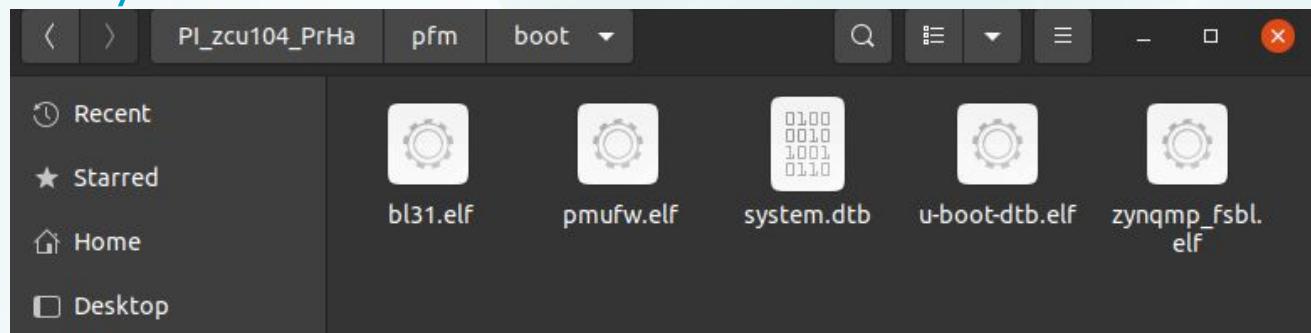
- In the end you should receive a successful message

```
srhugo@srhugo:~/Desktop/Projeto_Integrador/PI_zcu104_Linux/images/linux$ ./sdk.sh -d ../../PI_zcu104_PrHa
Petalinux SDK installer version 2021.2
=====
You are about to install the SDK to "/home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_PrHa". Proceed [Y/n]? y
Extracting SDK.....done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.
$ . /home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_PrHa/environment-setup-cortexa72-cortexa53-xilinx-linux
srhugo@srhugo:~/Desktop/Projeto_Integrador/PI_zcu104_Linux/images/linux$ █
```

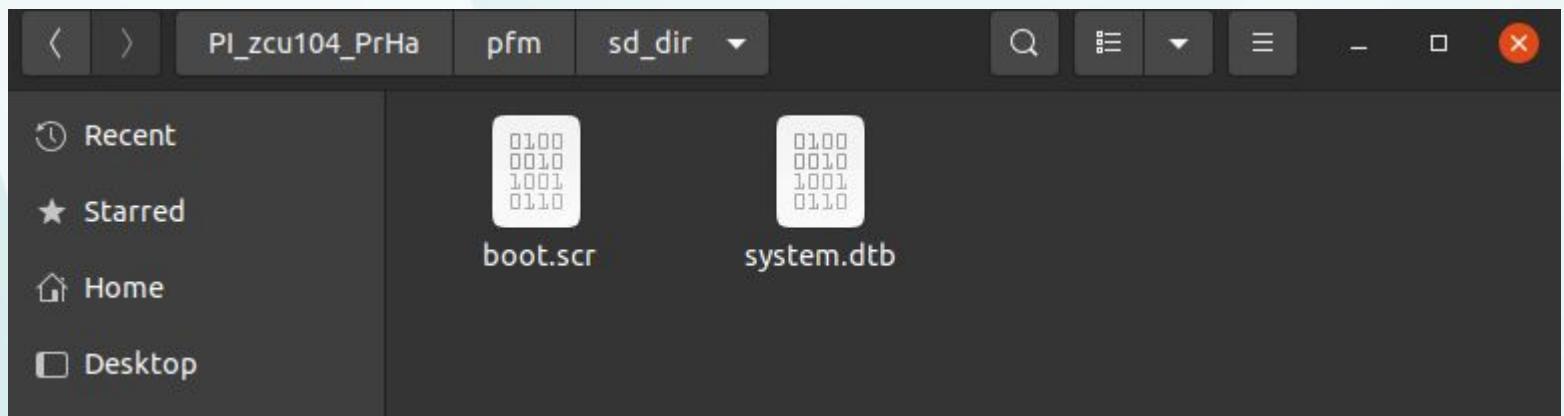
- After this you must have 3 “main” folders. The application one must have this 2 folders



- Go inside the pfm folder and create another 2 folder (boot and sd\_dir)
  - Note that these names are standard. If you want, you can name it differently.
- Inside the images/linux copy the following files to the boot folder
  - zynqmp\_fsbl.elf
  - pmufw.elf
  - bl31.elf
  - u-boot-dtb.elf: rename to u-boot.elf inside the boot folder
  - system.dtb

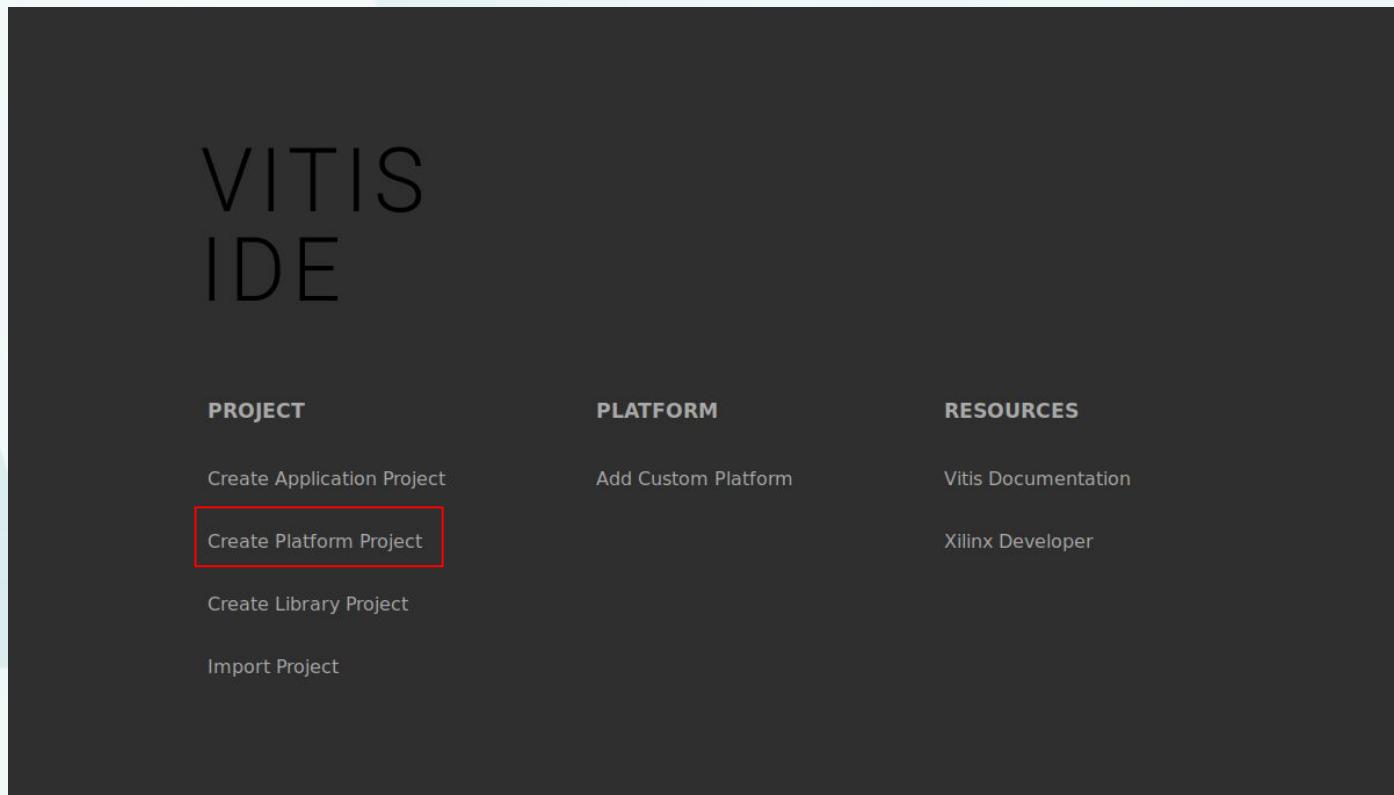


- Inside the images/linux copy the following files to the sd\_dir folder
  - boot.scr
  - system.dtb



- Now open Vitis and select the application folder as workspace directory

- Create a Vitis Platform

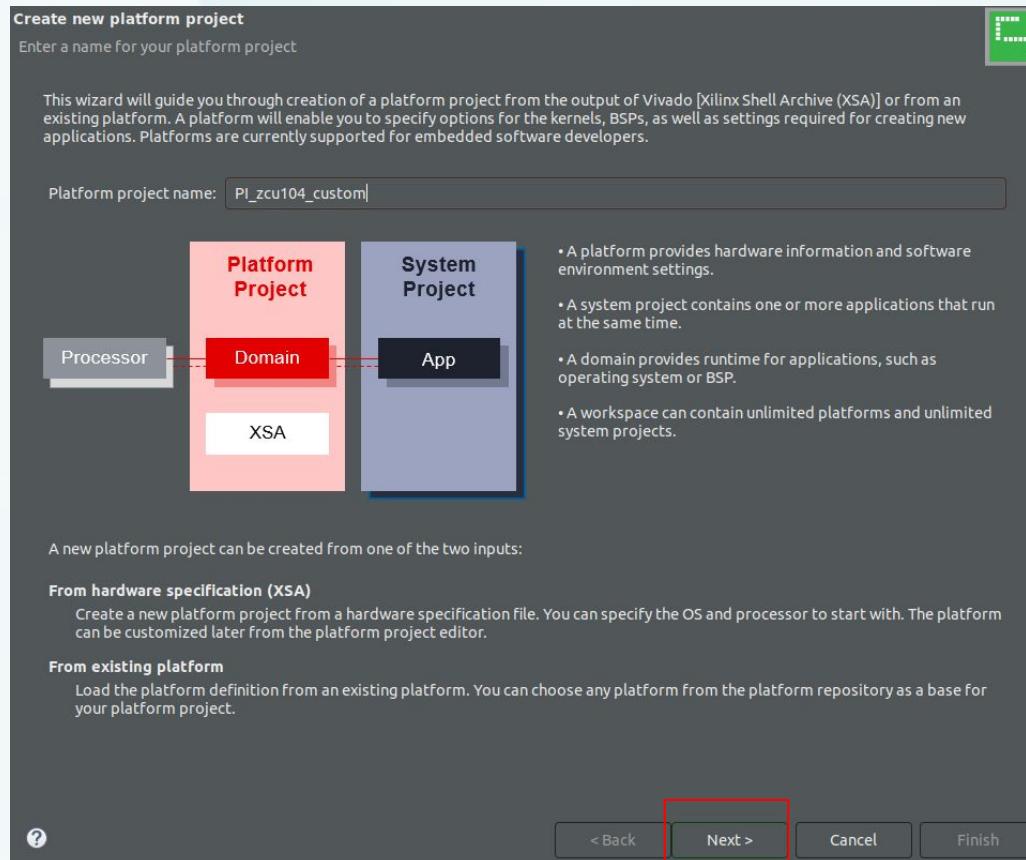


# Creating a Custom Vitis Platform

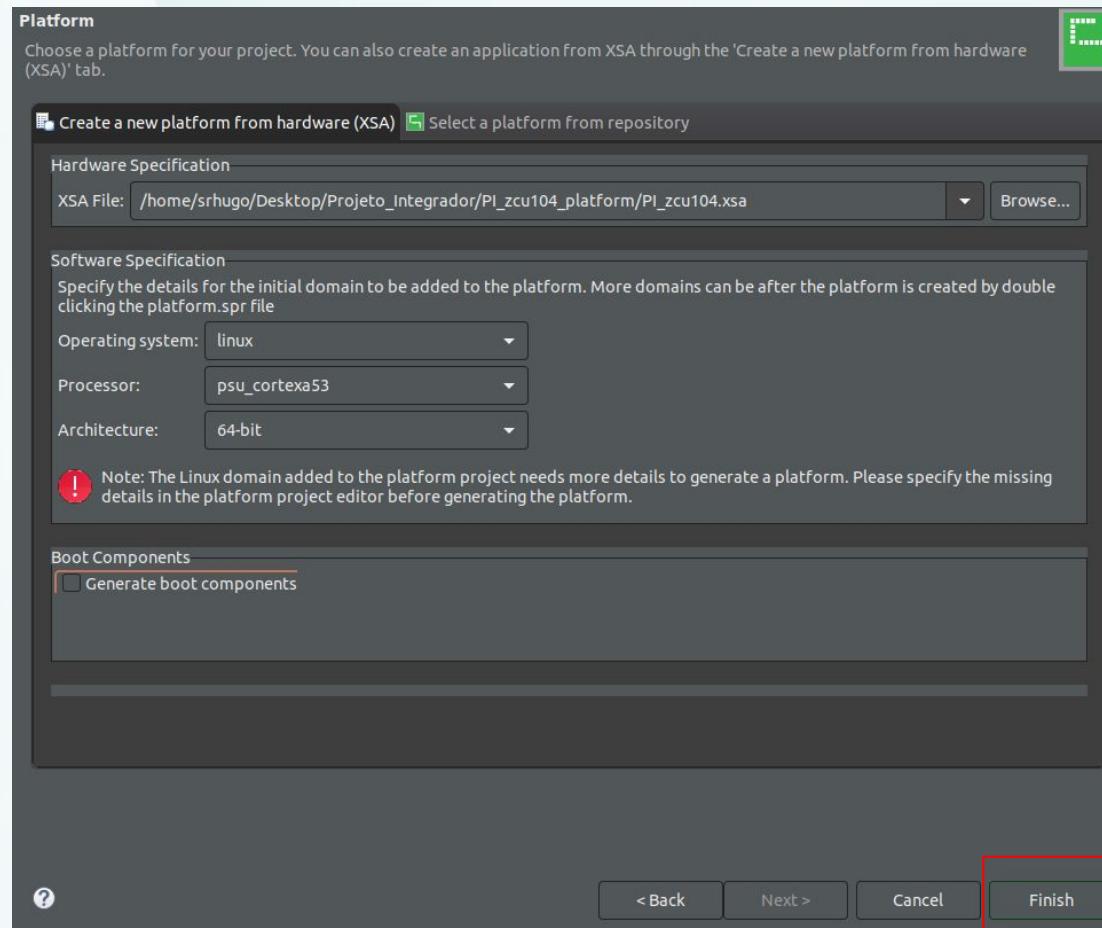


CENTROALGORITMI

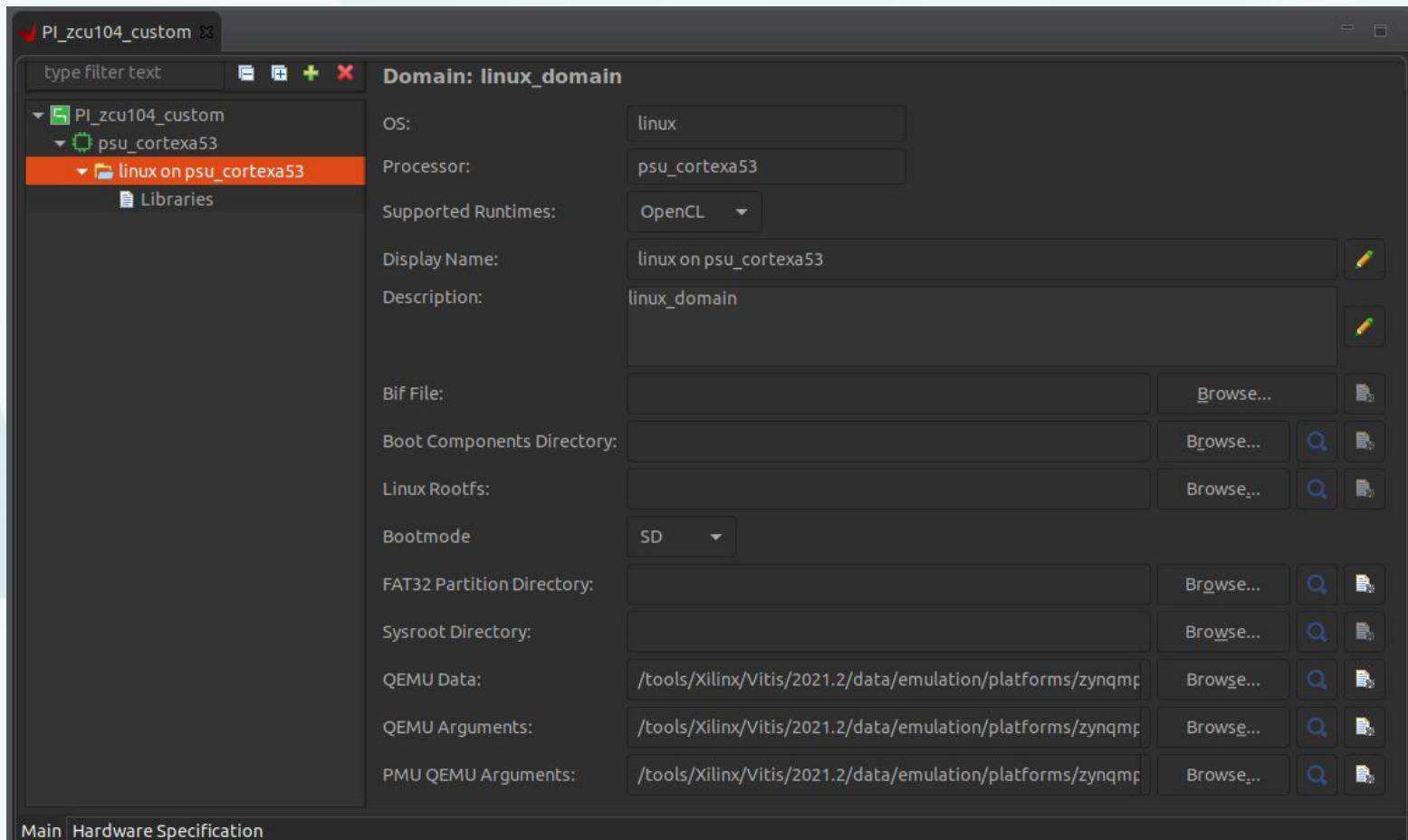
- Choose a name for your platform project
  - Do not choose a complicated name



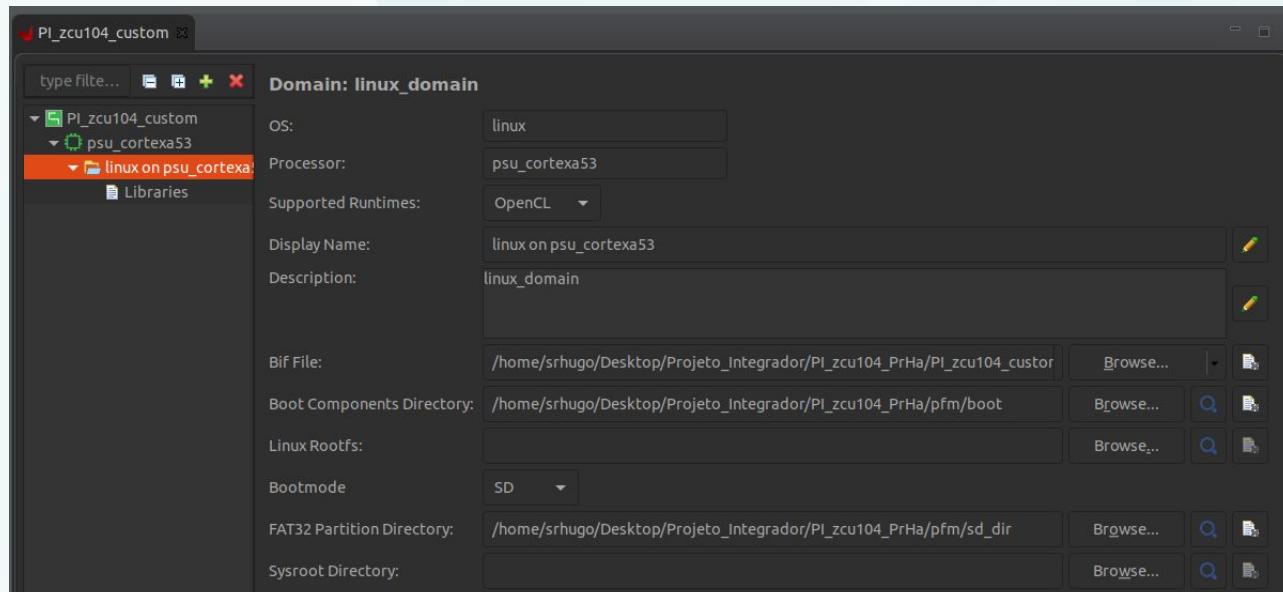
- Select the created .xsa file
- Change the operating system to **Linux**
- Remove “Generate boot components” option



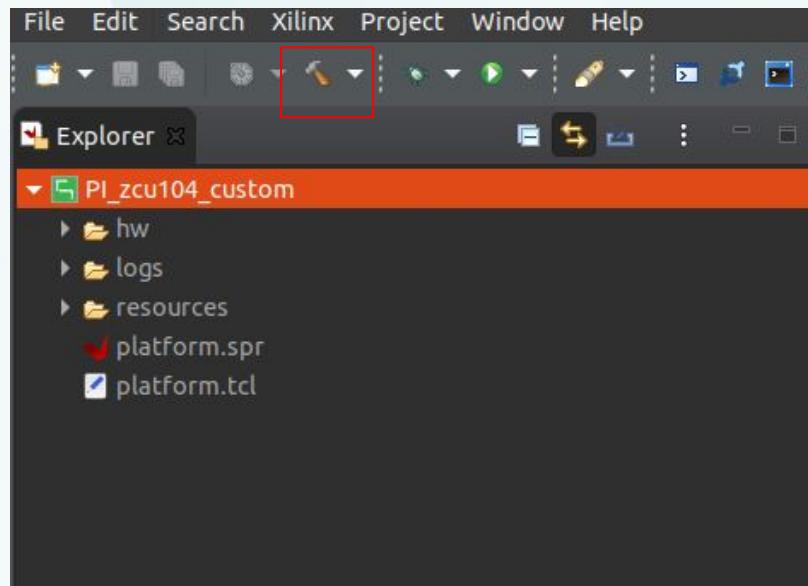
- After finished, go to **linux on psu\_cortexa53** tab



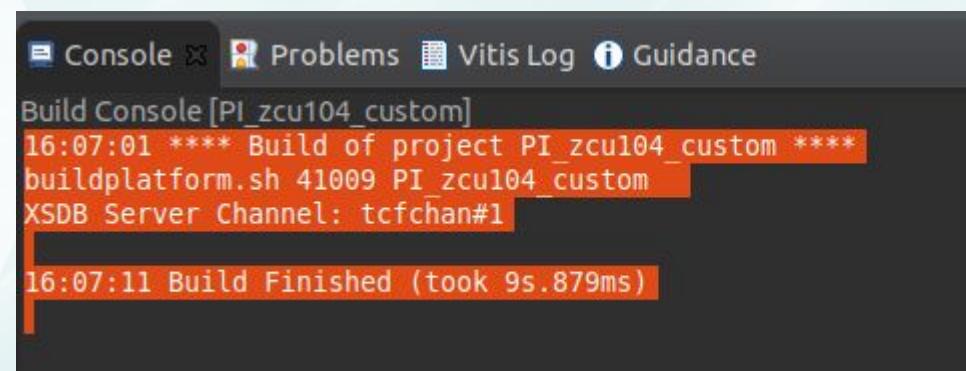
- For the Bit file, select the **Generate BIF** option
  - It will add a path automatically
- For the Boot Components Directory, select the **pfm/boot** folder
- For the FAT32 Partition Directory, select the **pfm/sd\_dir** folder
- In the end, you should have the following result



- Select the platform project and build it



- Expected Output:

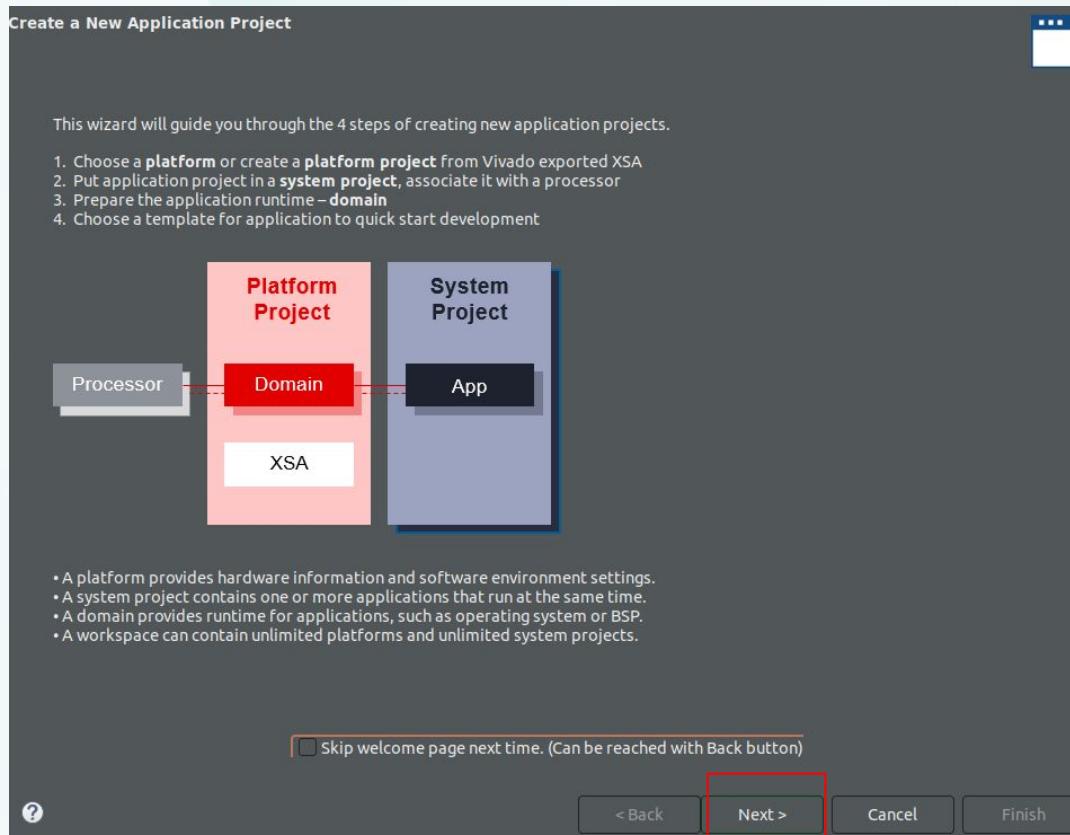


```
Console × Problems Vitis Log Guidance
Build Console [PI_zcu104_custom]
16:07:01 **** Build of project PI_zcu104_custom ****
buildplatform.sh 41009 PI_zcu104_custom
XSDB Server Channel: tcfchan#1

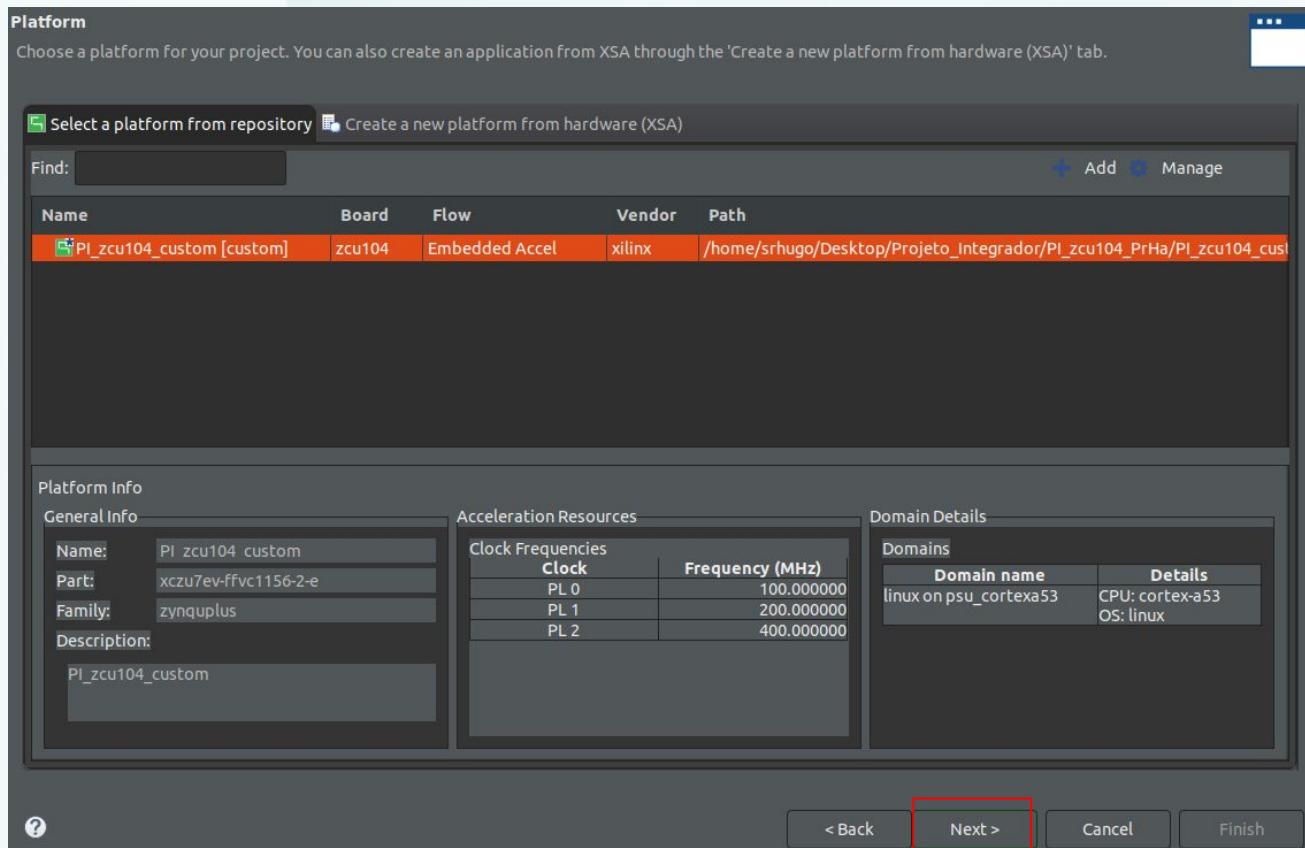
16:07:11 Build Finished (took 9s.879ms)
```

The screenshot shows the 'Console' tab of the Vitis IDE. The output window displays the build process for the 'PI\_zcu104\_custom' project. It starts with the command 'buildplatform.sh 41009 PI\_zcu104\_custom', followed by the message 'XSDB Server Channel: tcfchan#1'. The build is completed successfully at 16:07:11, with a total duration of 9s.879ms.

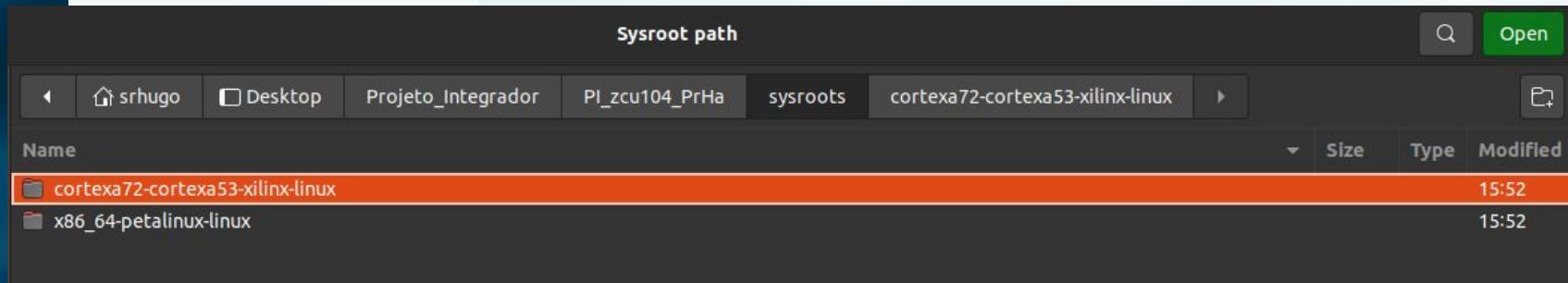
- Now that we have a custom platform let's first create a simple application (Hello World)
- To do that, go to **File -> New -> Application**



- The created platform should appear in the available platforms.
  - If not, browse it with the add option

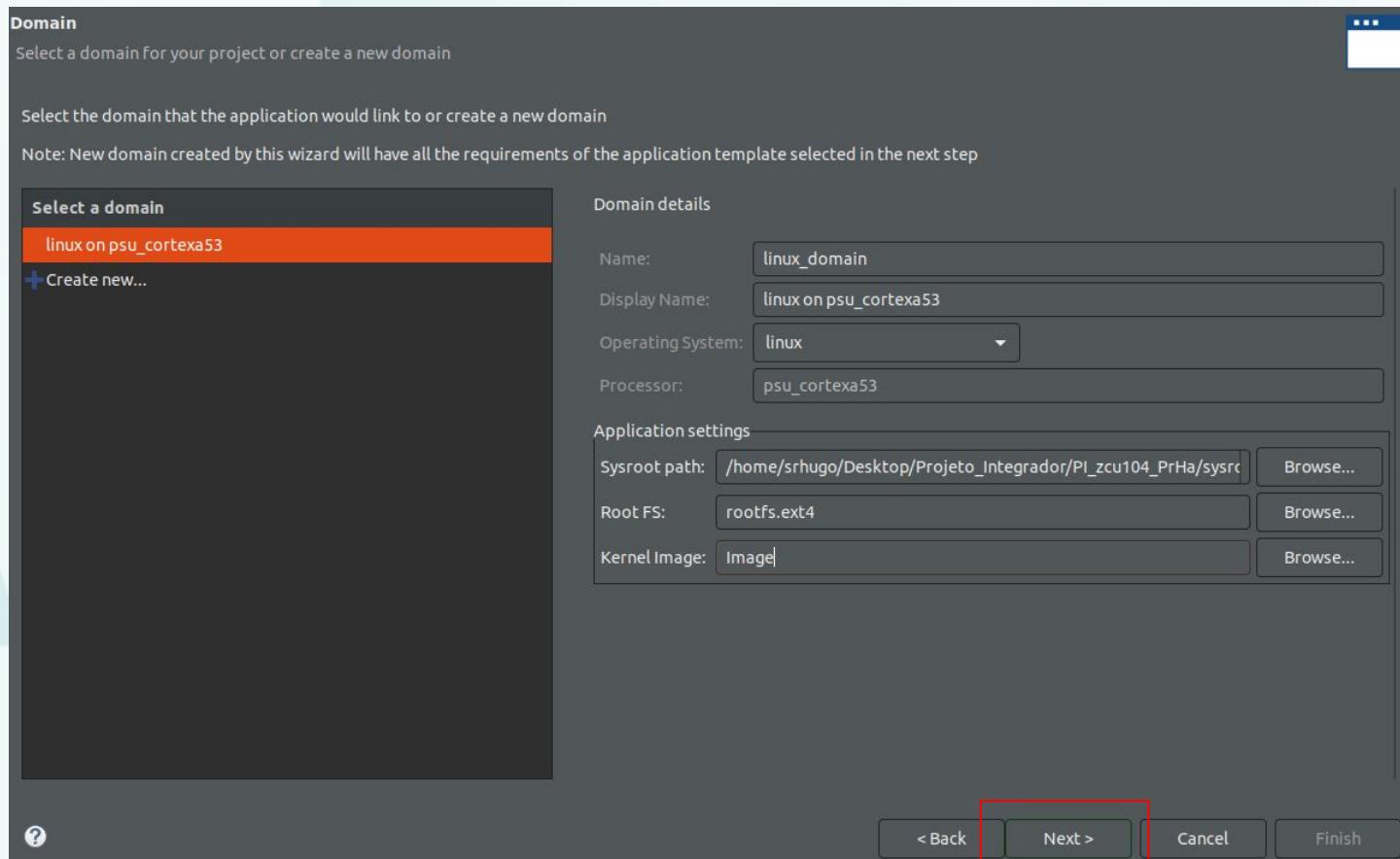


- Select the **linux on psu\_cortexa53** domain
- Then set the Sys\_root path to the following directory

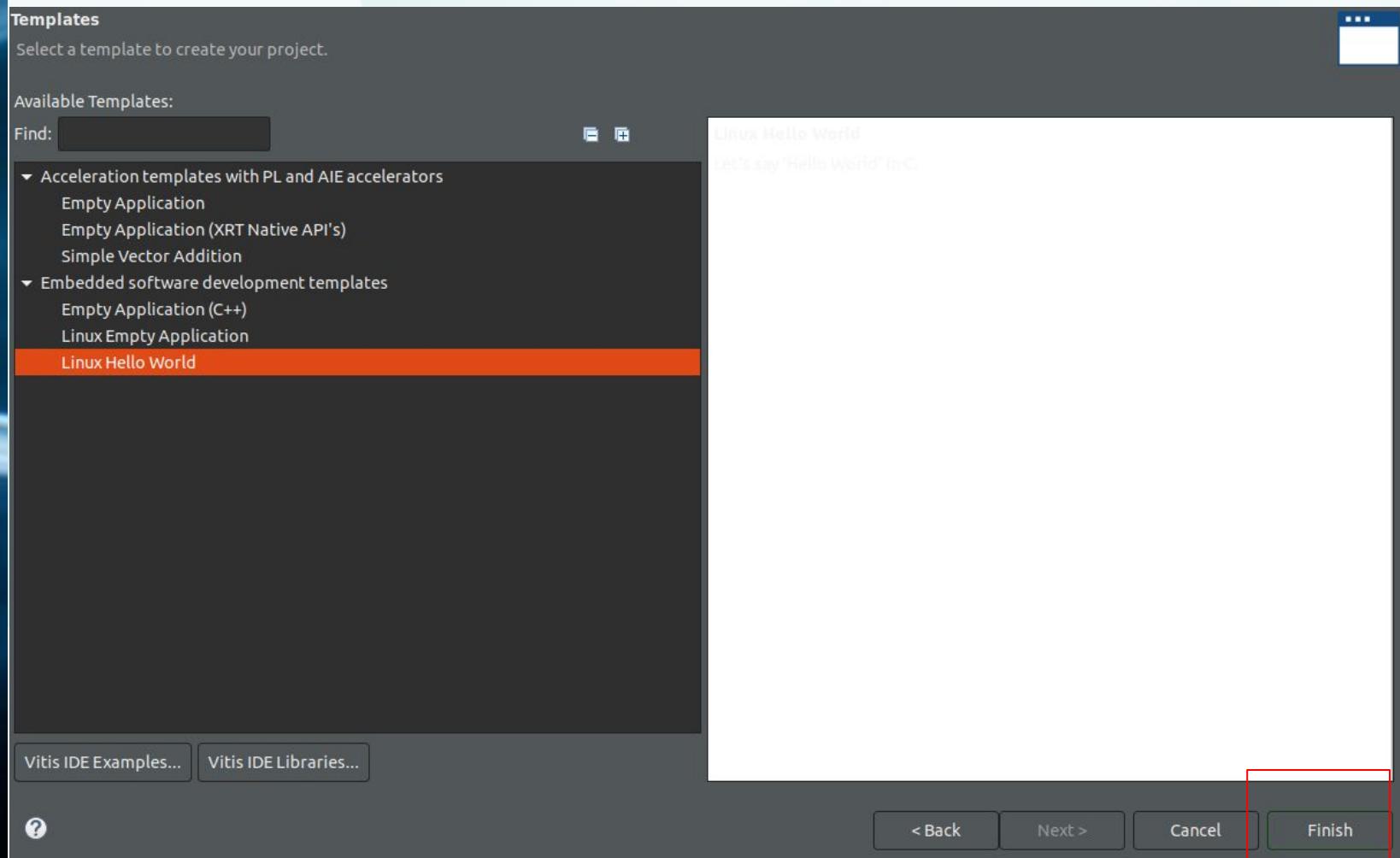


- In Root FS type rootfs.ext4
- In Kernel Image type Image

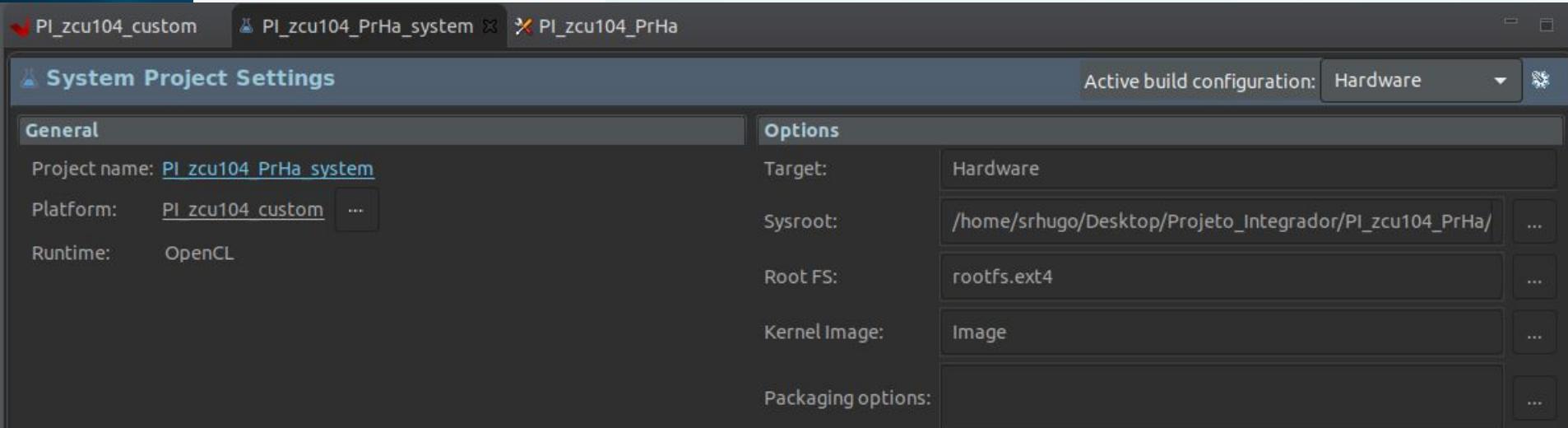
- You should have the following domain screen



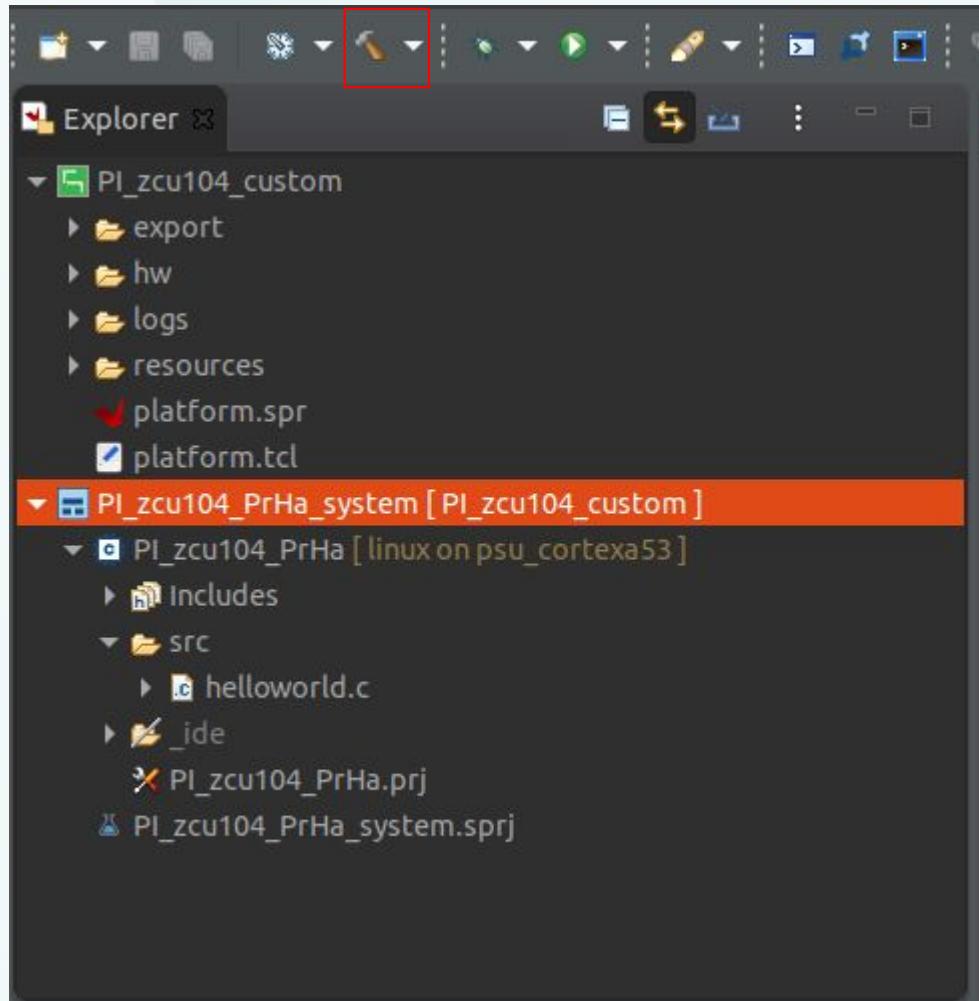
- Select the Hello World Template and hit Finish



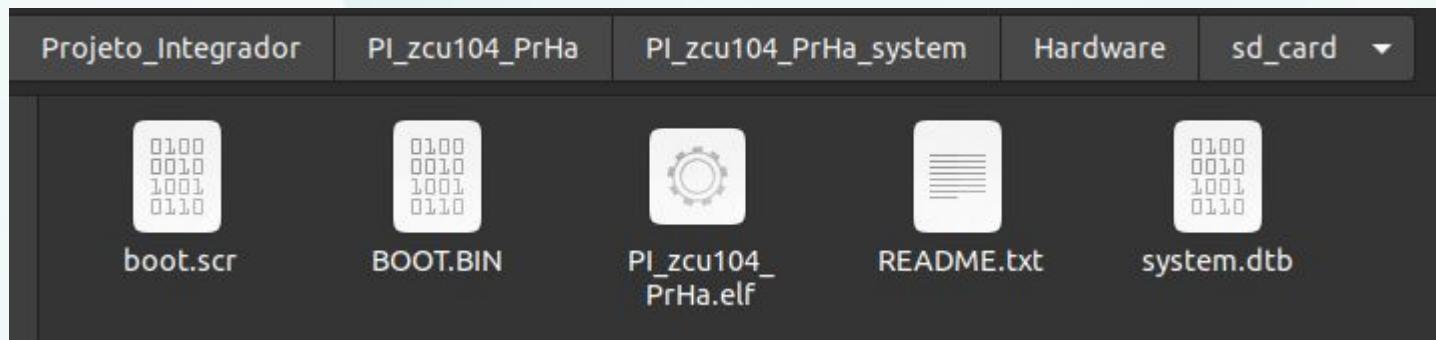
- Change the **Active build Configuration** the Hardware



- Select the Application project and build it

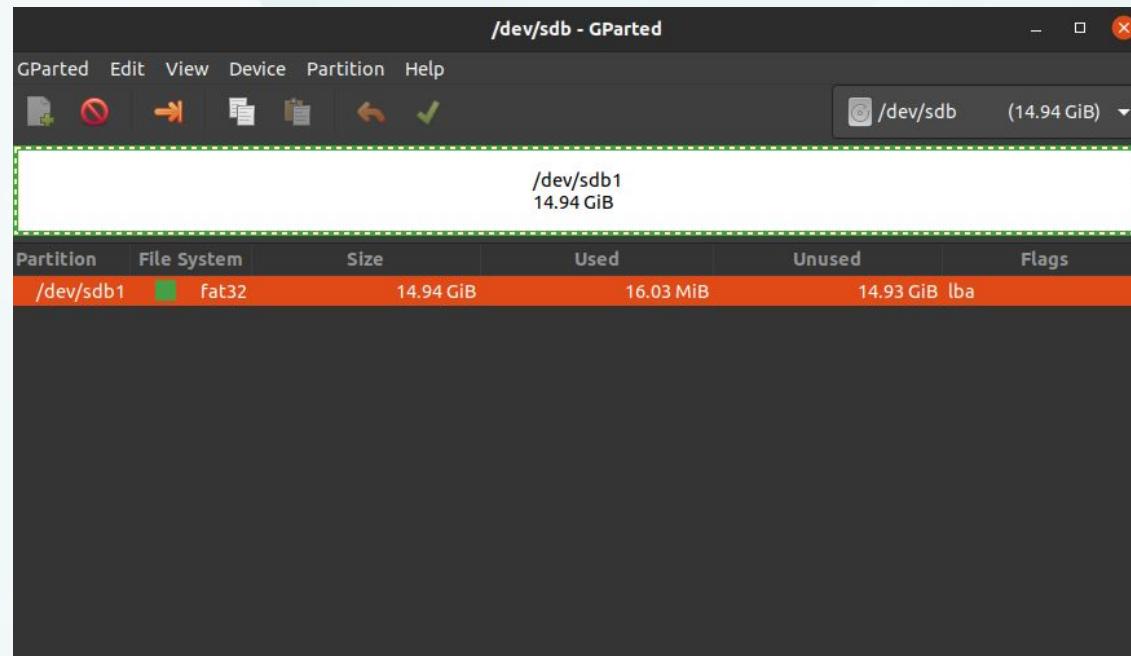


- If build was successful, you must have the following files in the <project\_name>\_system/Hardware/sd\_card folder.

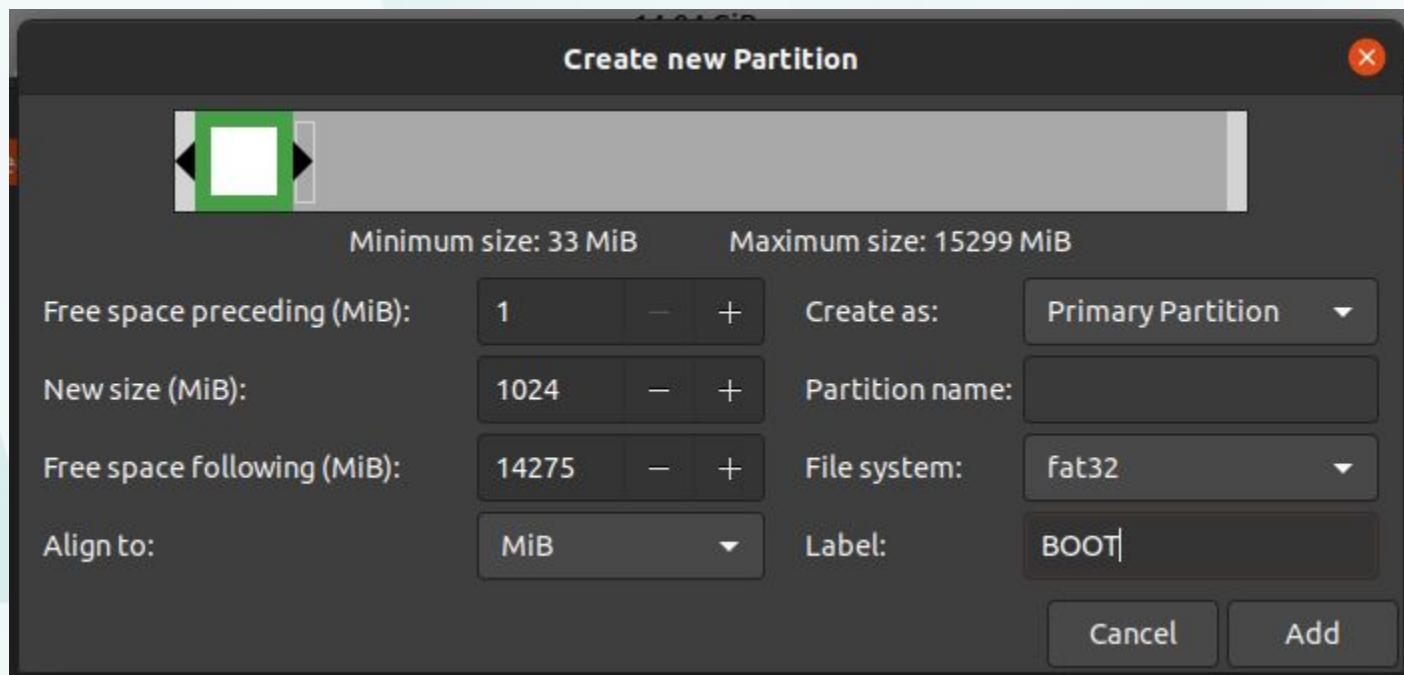


- Now we must prepare the SD card to receive the linux image

- Prepare the SD Card to receive the linux
  - In this tutorial will be used the Gparted Tool but there other ways to make the partitions
- Connect the SD card and open Gparted
  - Select the SD card

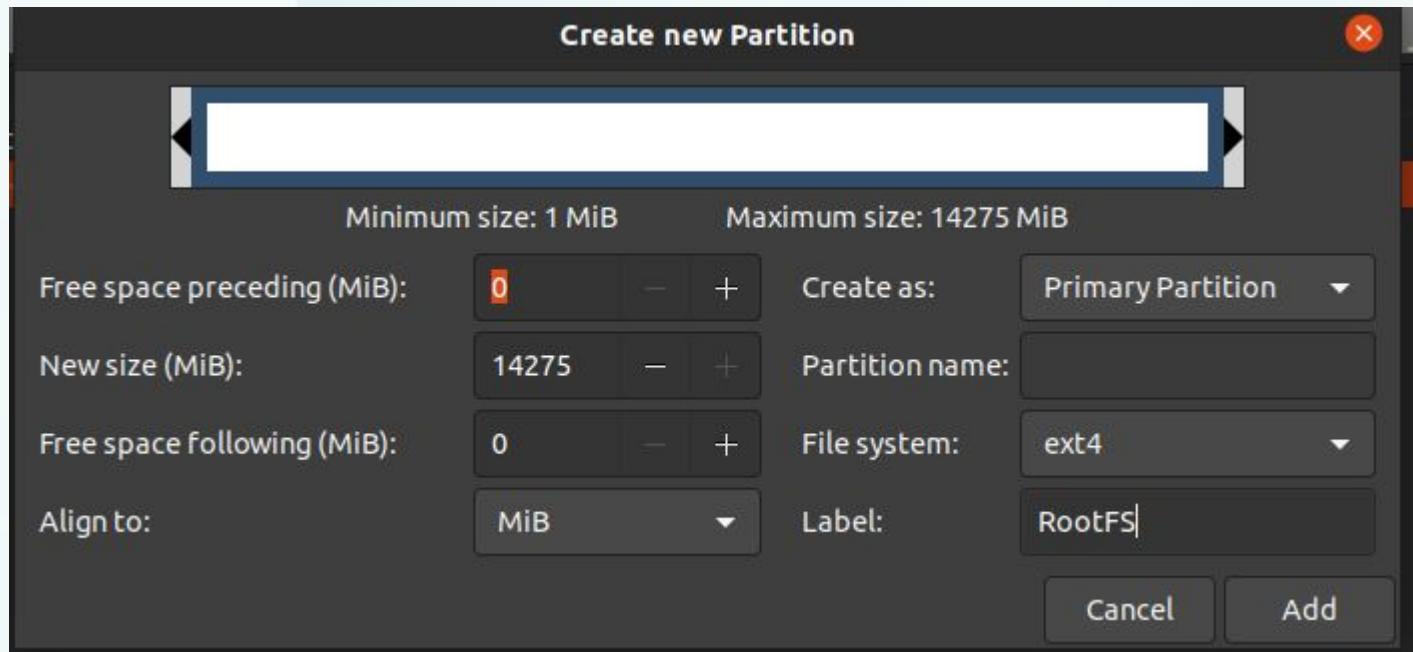


- Delete all existing partitions
- Create the BOOT partition with the following settings

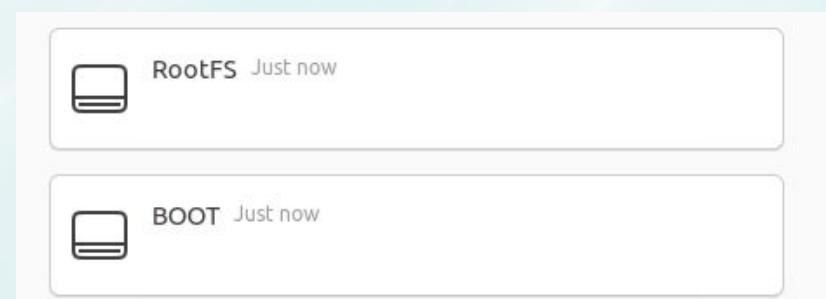


- If space is an issue, you can reduce the BOOT partition but be careful

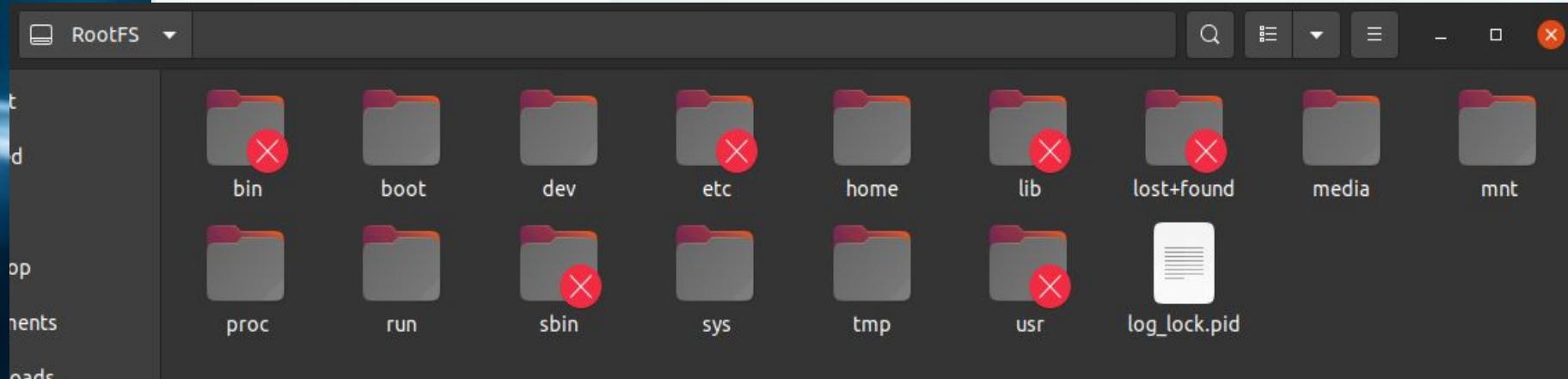
- Create the RootFS partition with the following settings



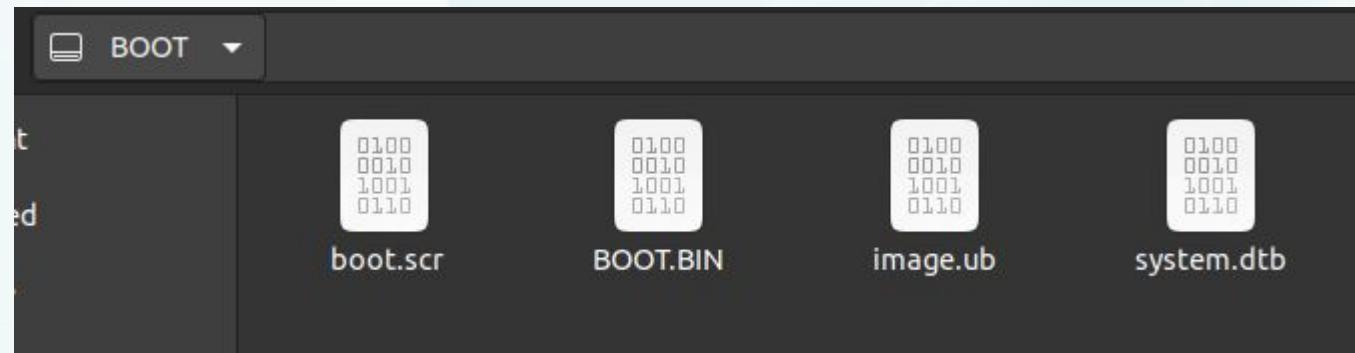
- Save the changes and verify if it was successful



- To the Root FS, go to images/linux folder and type the following command
  - `sudo tar -xvf rootfs.tar.gz -C /media/<yourName>/RootFS`
- RootFS partition after this command

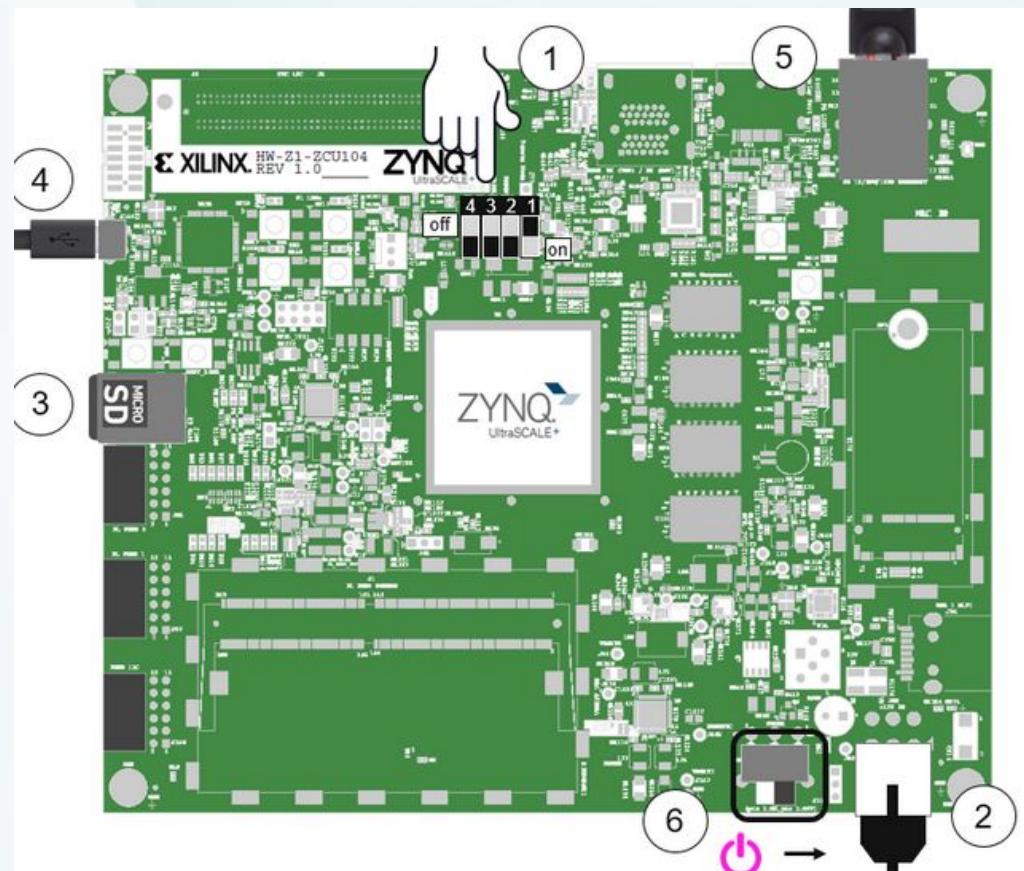


- For the BOOT partition copy the following files from the image/linux folder
  - boot.scr
  - image.ub
  - system.dtb
- Go to the generated files in the application project and copy the BOOT.BIN



- Now the sd\_card is prepared to go to the ZCU104
- You can send the program by two ways
  - Copying the .elf directly to RootFS folder
  - Sending it by a ssh connection
- We will going to do both! Choose one and keep it.
- Go to
  - <appllicationFolder>/<appllicationFolder>\_system/Hardware/sd\_card
- To copy to RootFS you must have root privileges
  - `sudo cp PI_zcu104_PrHa.elf /media/<yourName>/RootFS/etc/`

- Remove the sd\_card and put it into the ZCU104
- Select this pin configuration in ZCU104
  - This sets the board to boot from the Micro-SD card



- Connect to the board by a SSH or UART connection
  - Here will be used the UART one
    - If you don't have any serial port terminal, GTKTerm is a good choice
- After booting, you chose be have to do the following commands

```
root@xilinx-zcu104-2021_2:/etc# cd /etc
root@xilinx-zcu104-2021_2:/etc# ls
ConsoleKit           fonts          init.d          logrotate.d      profile
PI_zcu104_HelloWorld.elf formfactor    inittab        matchbox        profile.d
XII                  fstab          inittab.d     mke2fs.conf    protocols
alsa                 gconf          inputrc       modules-load.d pulse
bindresvport.blacklist group          issue         motd          rc0.d
busybox.links         gshadow        issue.net     mtab          rc1.d
ca-certificates       gtk-3.0        jupyter      netconfig      rc2.d
ca-certificates.conf host.conf      ld.so.cache   network       rc3.d
dbus-1               hostname       ld.so.conf    nsswitch.conf rc4.d
default              hosts          libreport     pam.d        rc5.d
dropbear             hotplug        login.defs   passwd       rc6.d
environment          inetd.conf    logrotate-dmesg.conf petalinux  rcS.d
```

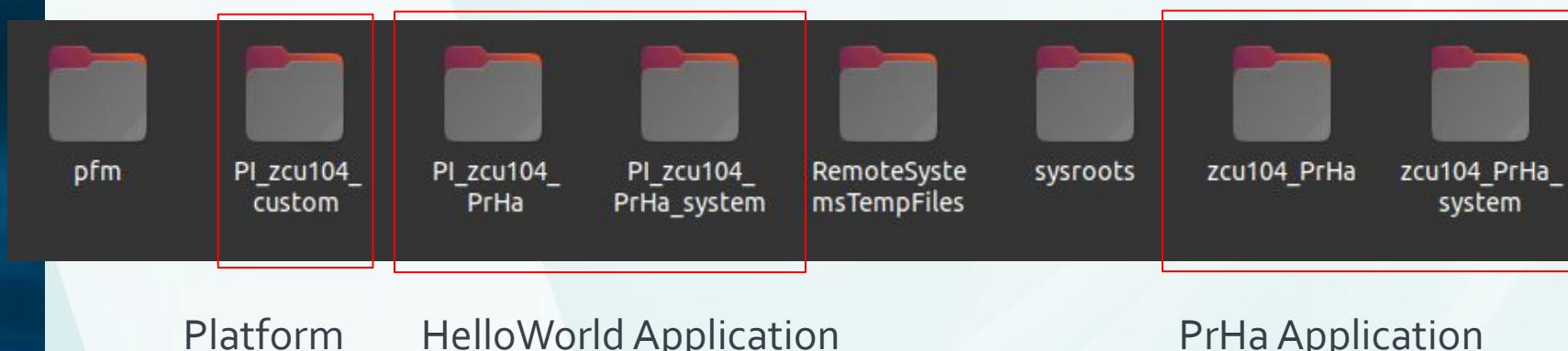
- Try to execute the HelloWorld program
- You should receive the following output :D
  - This image was cutted, so no worry about missing files

```
root@xilinx-zcu104-2021_2:~# cd /etc
root@xilinx-zcu104-2021_2:/etc# ./PI_zcu104_HelloWorld.elf
Hello World
root@xilinx-zcu104-2021_2:/etc# █

/dev/ttyUSB1 115200-8-N-1
```

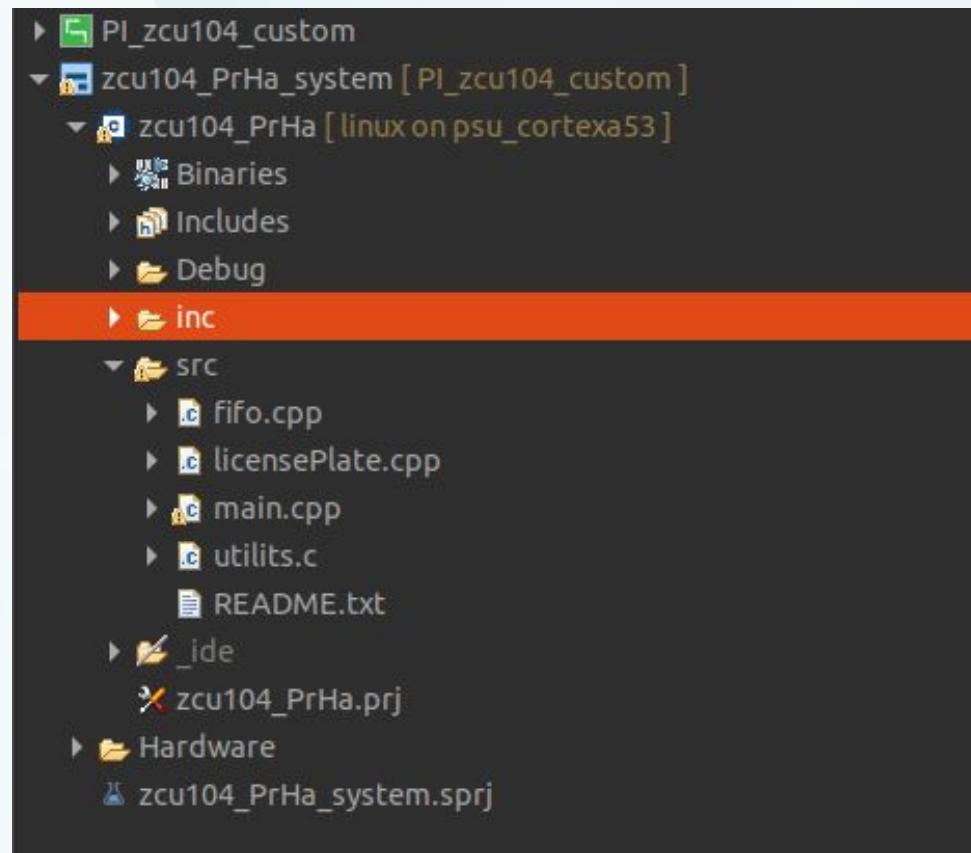
- If the HelloWorld isn't working, check all the previous steps and make sure you follow them correctly.
- Let's move to the PrHa code!
  - You can get it on GitHub
- Create another application project like before
  - This time, let's call it zcu104\_PrHa
- Follow the exact same steps, but instead of choosing the helloWorld template, choose the C++ empty project.

- After that, you should have the two projects in the same folder



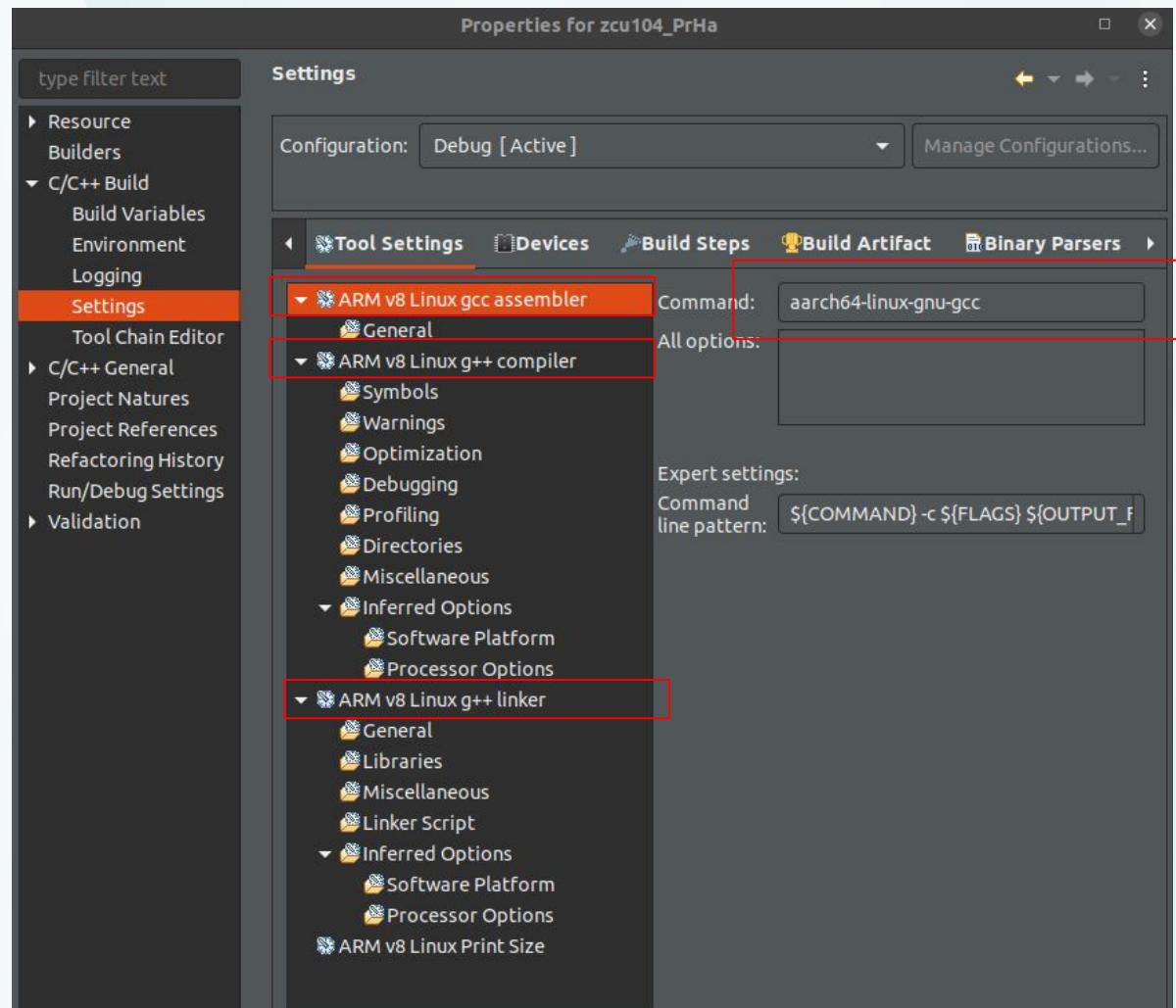
- Add the inc and src folders to zcu104\_PrHa folder

- Your Explorer window should be like this

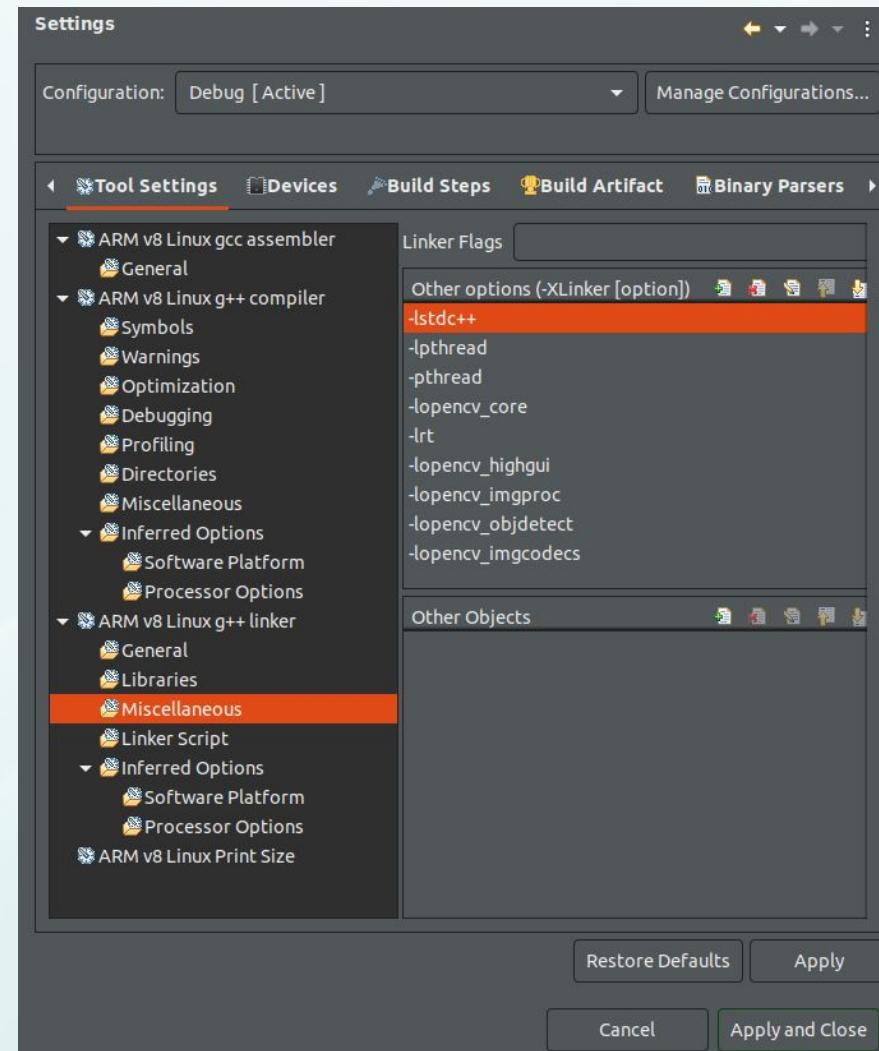


- Before build there's some changes you must do
  - Inside Vitis, select zcu104\_PrHa -> right click -> properties
  - Go to C/C++ Build -> settings
- 
- First you must change the compiler from g++ to gcc
    - This code is a mix between C and C++, so gcc compiler is a better option

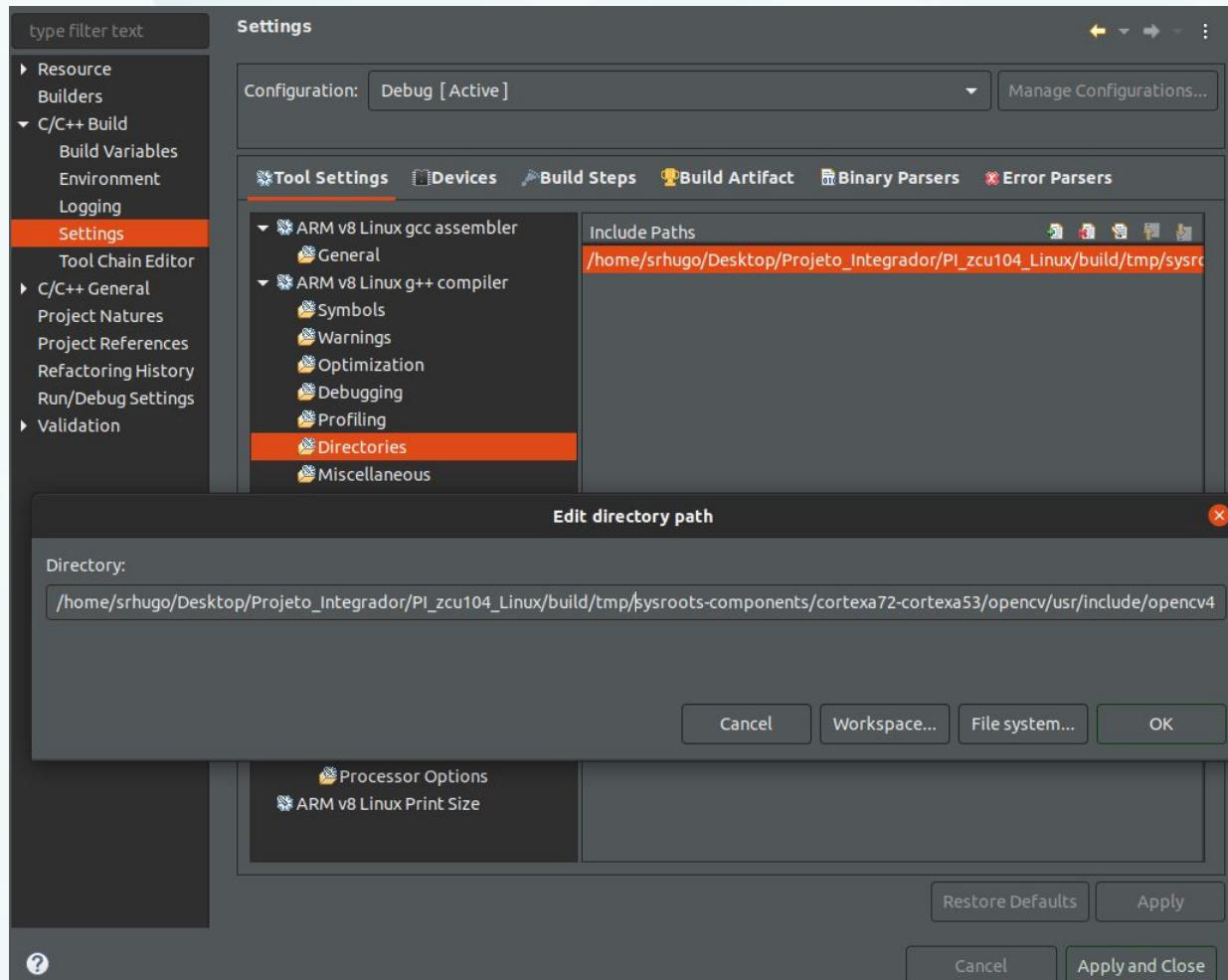
- Change g++ to gcc in this 3 tabs



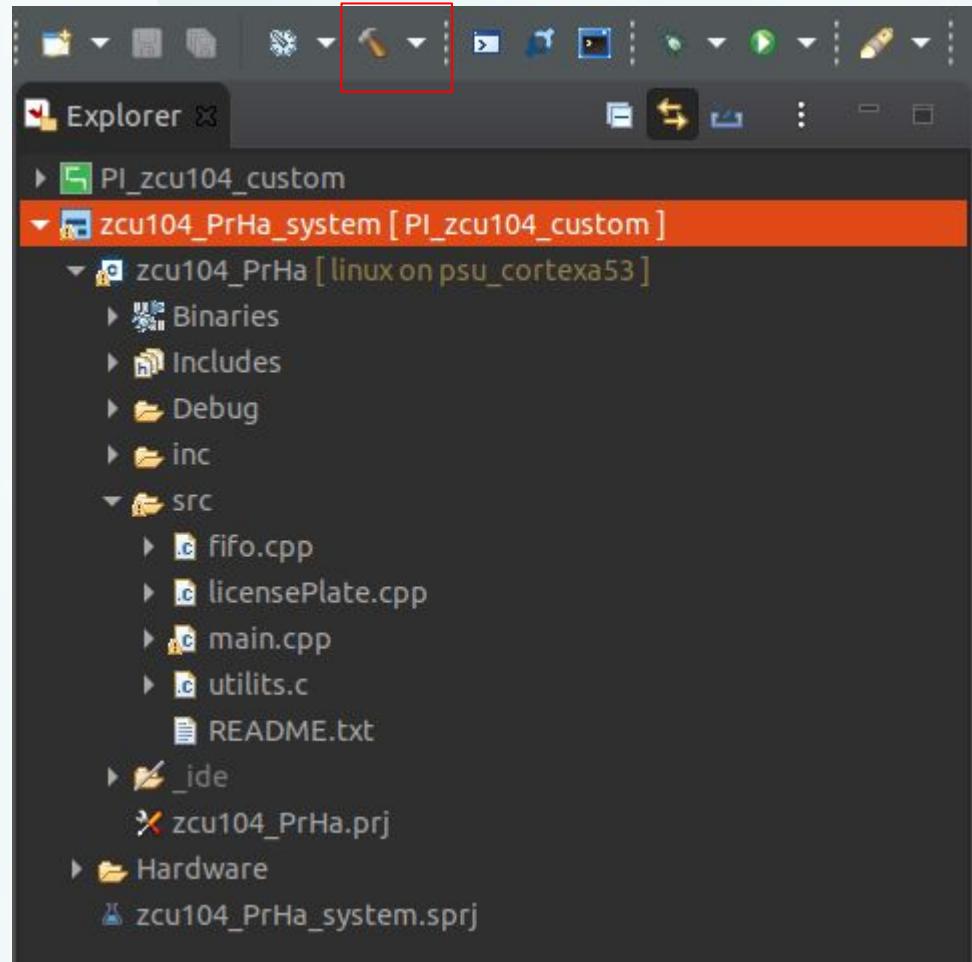
- After that, we must add the necessary flags to the linker
  - `-lstdc++`
  - `-lpthread`
  - `-pthread`
  - `-lrt`
  - `-lopencv_core`
  - `-lopencv_highgui`
  - `-lopencv_imgproc`
  - `-lopencv_objdetect`
  - `-lopencv_imgcodecs`
- `lstdc++` for c++ code
- `lpthread` for threads
- `opencv` for opencv



- Finally you must add the opencv directory to the compiler
  - It must be the opencv folder created by petalinux



- Save all changes and try to build the project



- There is a probability of vitis changing the directories when it's compiling for the first time

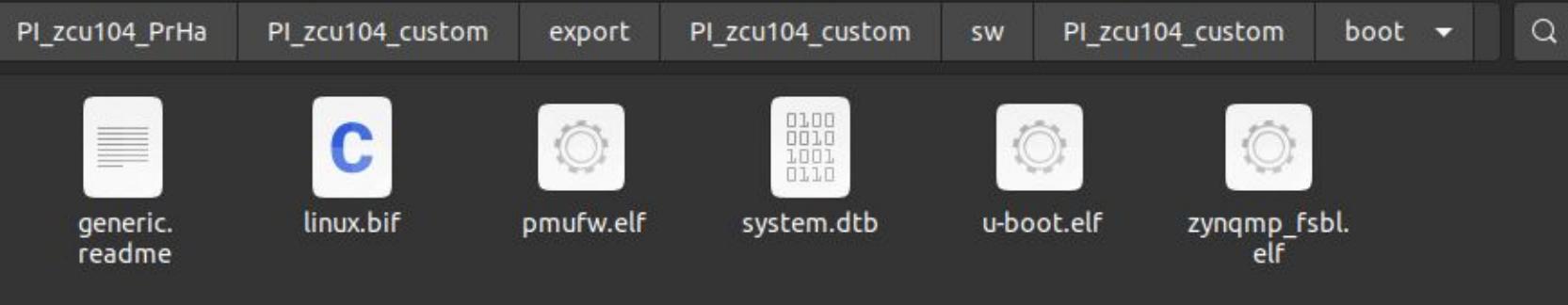
```
Build Console [zcu104_PrHa_system, Hardware]
Error: platform file not found: /home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_PrHa/PI_zcu104_custom/export/PI_zcu104_custom/sw/atf,PI_zcu104_custom/boot/bl31.elf
Error: platform file not found: /home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_PrHa/PI_zcu104_custom/export/PI_zcu104_custom/sw/dtb,PI_zcu104_custom/boot/system.dtb
Error: platform file not found: /home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_PrHa/PI_zcu104_custom/export/PI_zcu104_custom/sw/uboot,PI_zcu104_custom/boot/u-boot.elf
Running /tools/Xilinx/Vitis/2021.2/bin/bootgen -arch zynqmp -image /home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_PrHa/zcu104_PrHa_system/Hardware/sd_card_temp/boot.bif
ERROR:BootGen - syntax error
  Line #13, "/home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_PrHa/zcu104_PrHa_system/Hardware/sd_card_temp/boot.bif".
... tom/sw/atf,PI_zcu104_custom/boot/bl31.elf
^

[ERROR] : BIF file parsing failed with code 1
Error writing SD card data : Error when running '/tools/Xilinx/Vitis/2021.2/bin/bootgen -arch zynqmp -image /home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_PrHa/zcu104_PrHa_system/Hardware/sd_card_temp/boot.bif'
make: *** [makefile:50: package] Error 1

12:17:03 Build Finished (took 25s, 578ms)
```

- This was added by vitis compiler! :(

- To fix this, go to this folder and open linux.bif
  - <platformFolder>/export/PI\_zcu104\_custom/sw/PI\_zcu104\_custom/boot



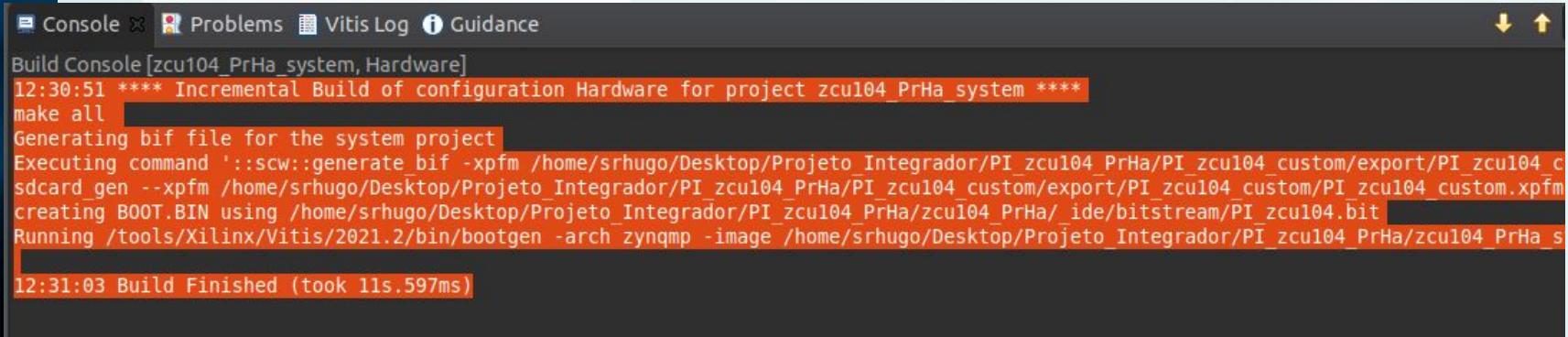
- Remove what is not correct

```
the_ROM_image:  
{  
    [fsbl_config] a53_x64  
    [bootloader] <PI_zcu104_custom/boot/zynqmp_fsbl.elf>  
    [pmufw_image] <PI_zcu104_custom/boot/pmufw.elf>  
    [destination_device=pl] <bitstream>  
    [destination_cpu=a53-0, exception_level=el-3, trustzone] <atf,PI_zcu104_custom/boot/bl31.elf>  
    [load=0x00100000] <dtb,PI_zcu104_custom/boot/system.dtb>  
    [destination_cpu=a53-0, exception_level=el-2] <uboot,PI_zcu104_custom/boot/u-boot.elf>  
}
```

- Save and compile again

```
the_ROM_image:  
{  
    [fsbl_config] a53_x64  
    [bootloader] <PI_zcu104_custom/boot/zynqmp_fsbl.elf>  
    [pmufw_image] <PI_zcu104_custom/boot/pmufw.elf>  
    [destination_device=pl] <bitstream>  
    [destination_cpu=a53-0, exception_level=el-3, trustzone] <PI_zcu104_custom/boot/bl31.elf>  
    [load=0x00100000] <PI_zcu104_custom/boot/system.dtb>  
    [destination_cpu=a53-0, exception_level=el-2] <PI_zcu104_custom/boot/u-boot.elf>  
}
```

- You should not receive no more errors



The screenshot shows a terminal window within the Vitis IDE interface. The title bar includes tabs for 'Console', 'Problems', 'Vitis Log', and 'Guidance'. The main area displays the build log for the project 'zcu104\_PrHa\_system, Hardware'.

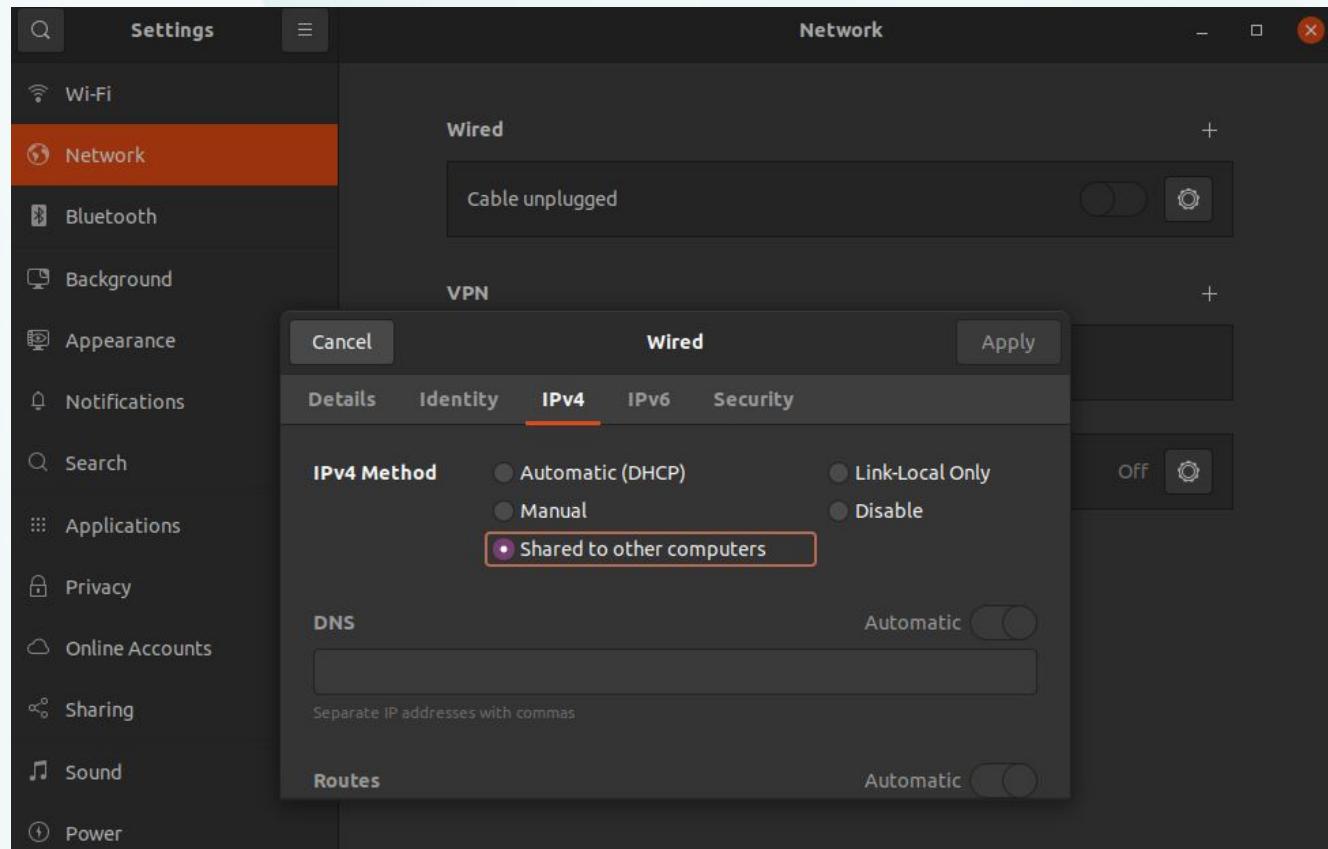
```
Build Console[zcu104_PrHa_system, Hardware]  
12:30:51 **** Incremental Build of configuration Hardware for project zcu104_PrHa_system ****  
make all  
Generating bif file for the system project  
Executing command ':::scw:::generate_bif -xpfm /home/srhugo/Desktop/Projeto Integrador/PI_zcu104_PrHa/PI_zcu104_custom/export/PI_zcu104_c  
sdcard_gen --xpfm /home/srhugo/Desktop/Projeto Integrador/PI_zcu104_PrHa/PI_zcu104_custom/export/PI_zcu104_custom/PI_zcu104_custom.xpfm  
creating BOOT.BIN using /home/srhugo/Desktop/Projeto Integrador/PI_zcu104_PrHa/zcu104_PrHa/_ide/bitstream/PI_zcu104.bit  
Running /tools/Xilinx/Vitis/2021.2/bin/bootgen -arch zynqmp -image /home/srhugo/Desktop/Projeto Integrador/PI_zcu104_PrHa/zcu104_PrHa_s  
12:31:03 Build Finished (took 11s.597ms)
```

- Go to
  - <applicationFolder>/<applicationFolder>\_system/Hardware/sd\_card
- Now you have the zcu104\_PrHa program available

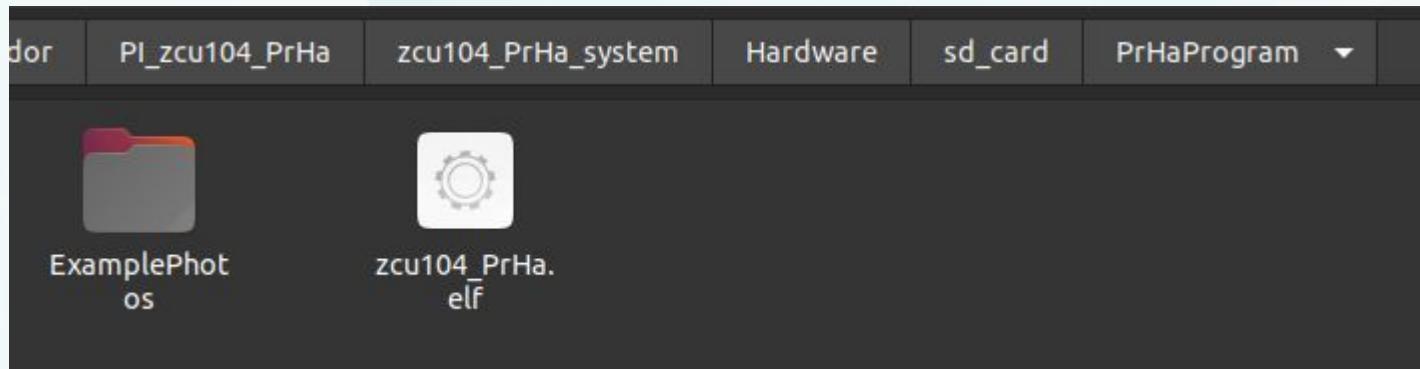


- Now let's try to send it by a ssh connection

- Make sure you have this setting in your host machine



- Create a folder, PrHaProgram, and copy the zcu104\_PrHa.elf
- Also copy the ExamplePhotos into there

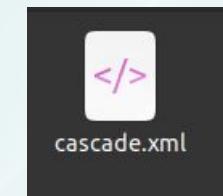


- Send the this folder to ZCU104
  - `scp PrHaProgram root@10.42.0.<number>:/etc`
    - To know the ZCU's IP, type `arp -a` in the terminal

- Before executing, you must send the cascade file to ZCU104

```
• int init_cascade()
{
    //Get the license plate cascade name
    string license_plate_cascade_name = "/usr/share/opencv4/haarcascades/cascade.xml";
    if( !plate_cascade.load( license_plate_cascade_name ) )
    {
        cout << "--(!)Error loading face cascade\n";
        return -EXIT_FAILURE;
    };
    return EXIT_SUCCESS;
}
```

- Check this code line to know the directory to send
  - Use the same command as before, but this time with this directory and the cascade file
    - It is available in PrHa/docs



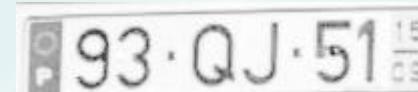
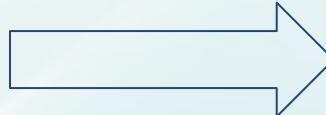
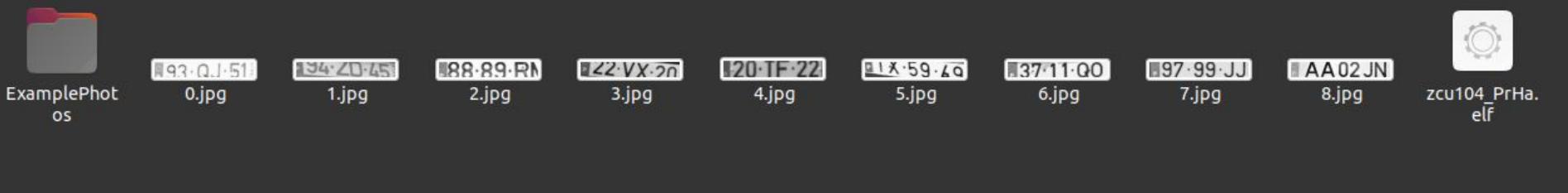
- Try to execute the program

```
root@xilinx-zcu104-2021_2:/etc/PrHaProgram# ./zcu104_PrHa.elf
t_plateRecognition is ready
t_getPhoto Thread is Ready
Plate Founded :D
The End
```

- SUCCESS! =D

- Also you can see the created outputs

```
root@xilinx-zcu104-2021_2:/etc/PrHaProgram# ls
0.jpg  2.jpg  4.jpg  6.jpg  8.jpg      zcu104_PrHa.elf
1.jpg  3.jpg  5.jpg  7.jpg  ExamplePhotos
```



# Bibliography



- [https://depositonce.tu-berlin.de/bitstream/11303/8019/3/rosli\\_etal\\_2015.pdf](https://depositonce.tu-berlin.de/bitstream/11303/8019/3/rosli_etal_2015.pdf)
- <https://uhra.herts.ac.uk/bitstream/handle/2299/14231/07167290%20Zhai%20Xiaojun%20final%20PhD%20submission.pdf?sequence=1>
- <https://ieeexplore.ieee.org/abstract/document/1639437>
- <http://www.cecs.uci.edu/~papers/ipdps06/pdfs/78-Raw-paper-1.pdf>
- [https://www.xilinx.com/support/documentation/sw\\_manuals/vitis\\_ai/2\\_0/ug1354-xilinx-ai-sdk.pdf#page=242&zoom=100,0,126](https://www.xilinx.com/support/documentation/sw_manuals/vitis_ai/2_0/ug1354-xilinx-ai-sdk.pdf#page=242&zoom=100,0,126)
- [https://www.xilinx.com/content/dam/xilinx/support/documentation/sw\\_manuals/xilinx2021\\_2/ug973-vivado-release-notes-install-license.pdf](https://www.xilinx.com/content/dam/xilinx/support/documentation/sw_manuals/xilinx2021_2/ug973-vivado-release-notes-install-license.pdf)
- <https://www.youtube.com/watch?v=3D2-OPArCiA>
- <https://www.youtube.com/watch?v=LU9hP7KLDgE>
- <https://github.com/Xilinx/Vitis-AI>
- <https://github.com/Xilinx/Vitis-AI-Tutorials>