ESRG
Embedded Systems
Research Group

CENTROALGORITMI

Projeto Integrador

Embedded Systems

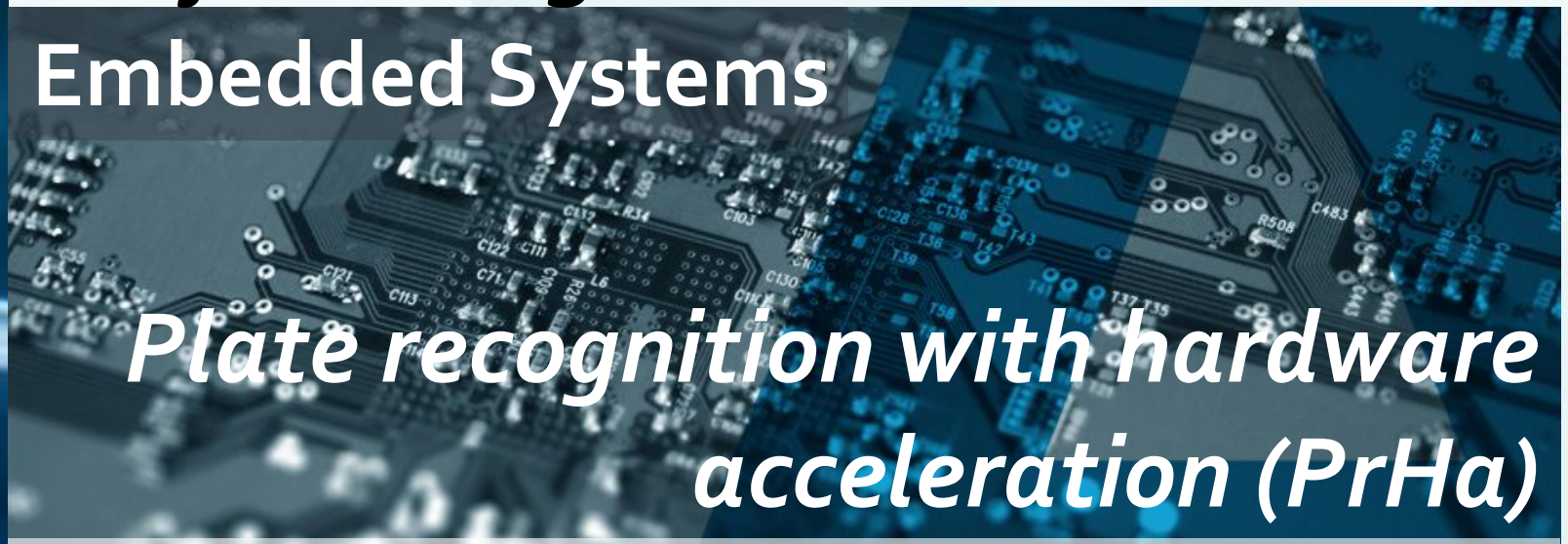


Plate recognition with hardware acceleration (PrHa)

Hugo Ribeiro – PG47241
Ricardo Mendes - PG47612

Universidade do Minho
Escola de Engenharia

COMPETE

QREN Quadro de Referência Estratégico Nacional

União Europeia
Fundo Europeu de Desenvolvimento Regional

FCT Fundação para a Ciência e a Tecnologia
Ministério da Ciência, Tecnologia e Ensino Superior

Agenda

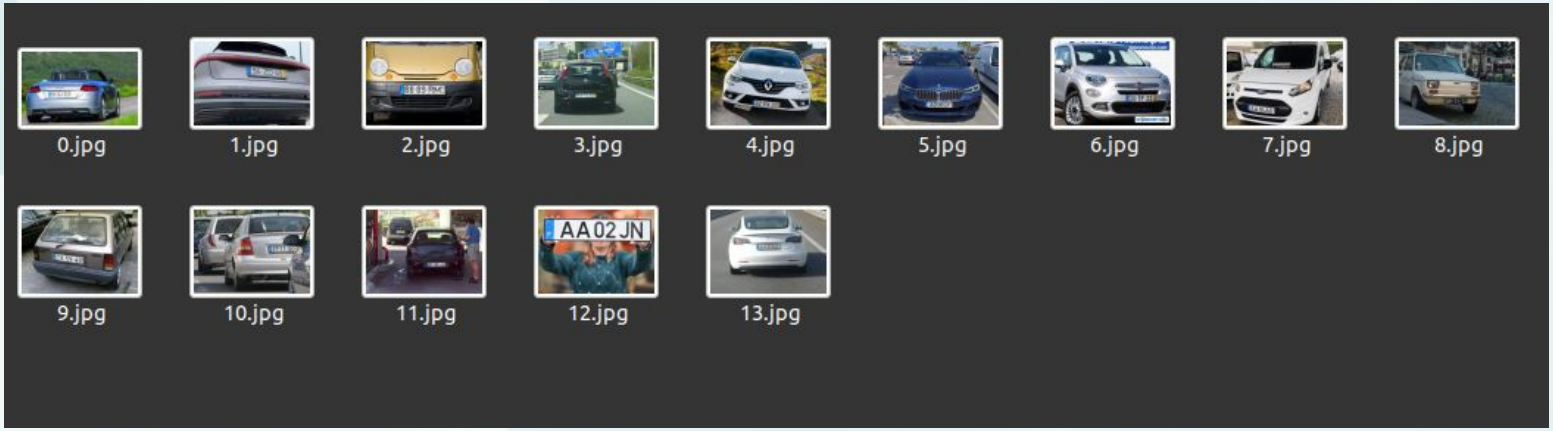
- Code Adaptation from PIgate
- Creating Hardware Platform
- Creating Linux Image
- Creating a Custom Vitis Platform
- Test an Hello World
- Test PrHa Code
- Bibliography

Code Adaptation from PIgate

- Before working in the project itself, the PIgate code was changed so it can be reused in this new project
- The PIgate 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
- PIgate Repository: https://github.com/HugoRibeiro-A88287-UM/PIgate

Code Adaptation from PIgate

- 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 PIgate's plate recognition model is well trained.
- All the used images are from google



Code Adaptation from PIgate

- Next parted 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.

Code Adaptation from PIgate

- After all changes, the project folder is the following

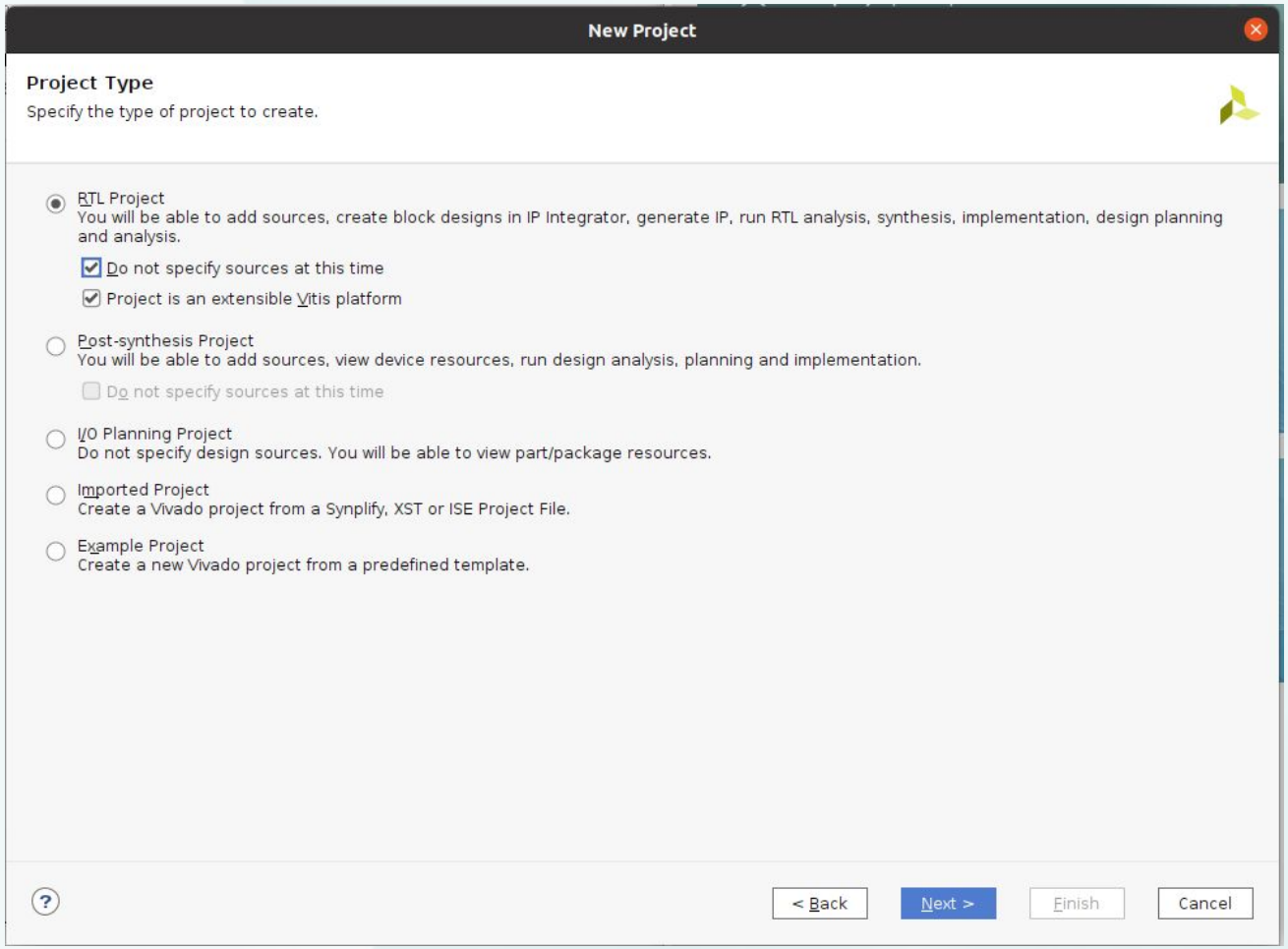
- It is possible to test it on the Laptop. For that, change the compiler in the src folder makefile.
- PrHa Repository: https://github.com/HugoRibeiro-A88287-UM/PrHa

Creating the Hardware Platform

- 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.

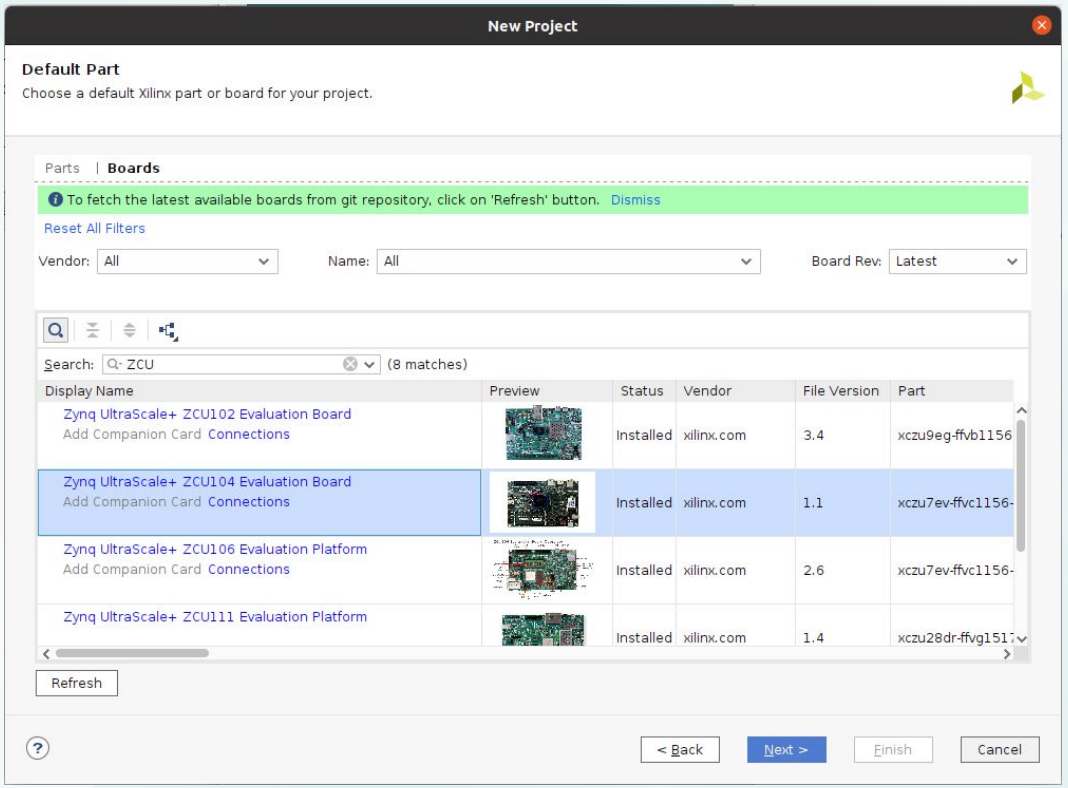
Creating the Hardware Platform

- Creating a vivado project:



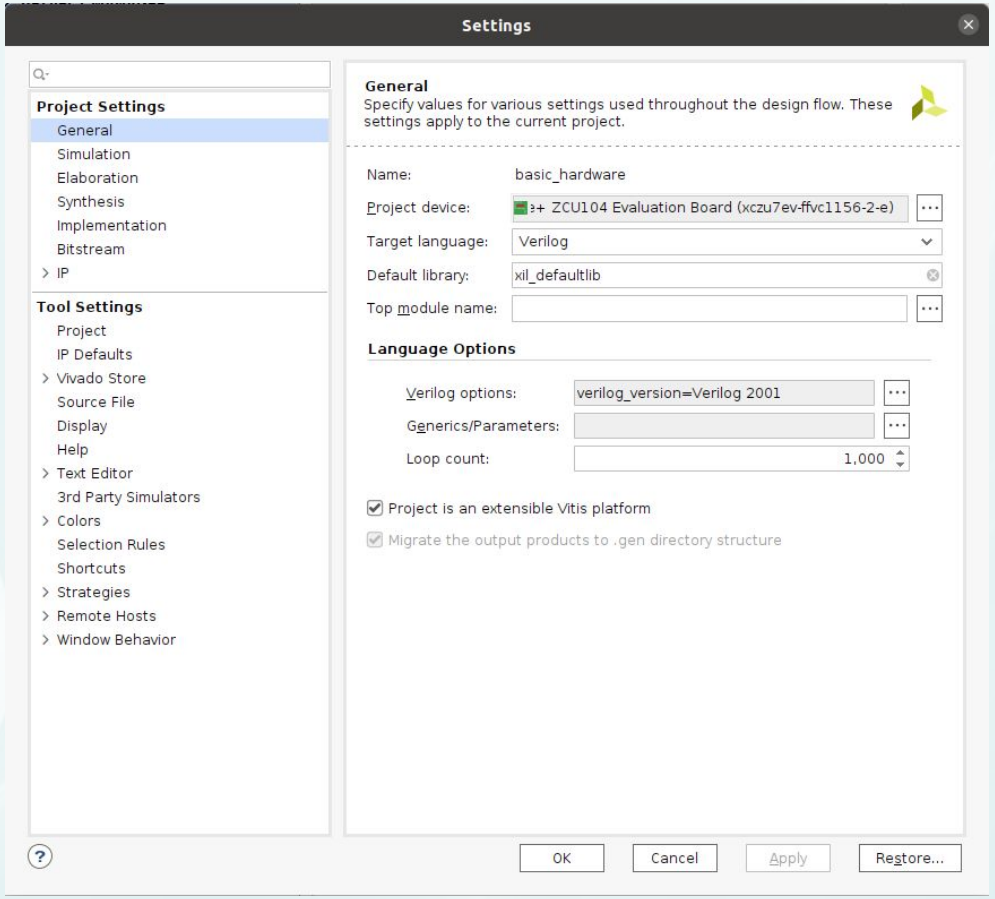
Creating the Hardware Platform

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



Creating the Hardware Platform

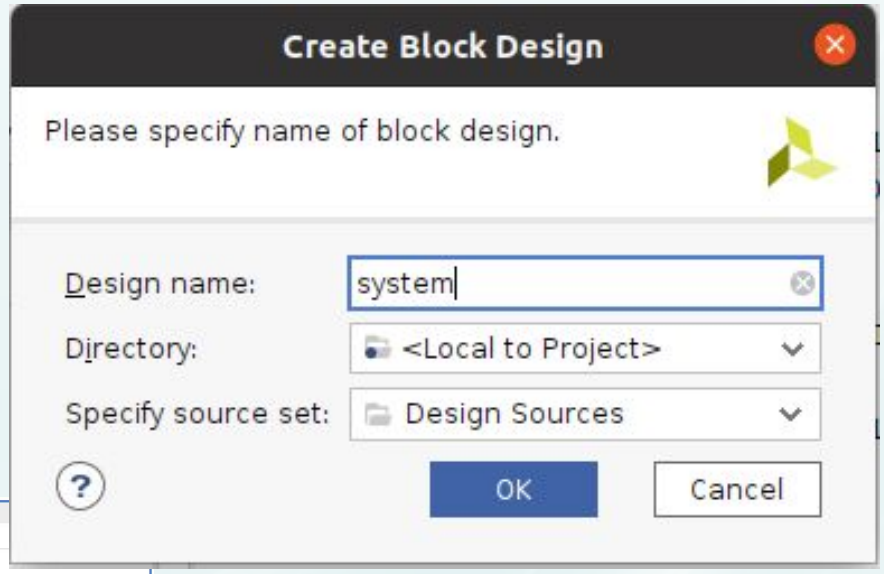
- Creating a vivado project:
  -Project settings:

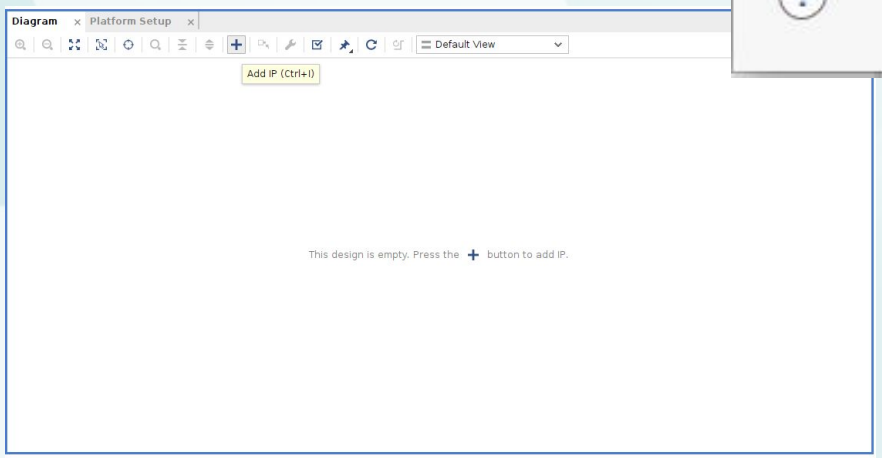


Creating the Hardware Platform

- Creating a block design:

This step creates an empty block design.





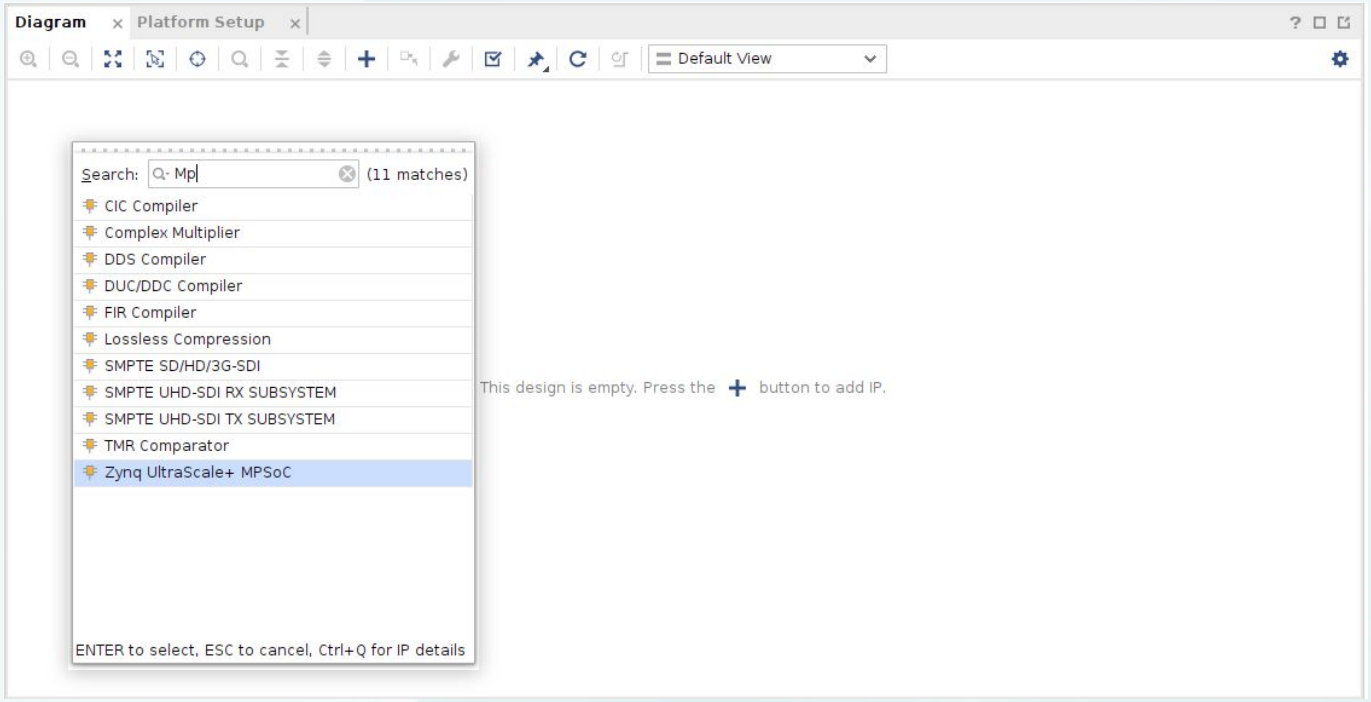
Creating the Hardware Platform

- Adding an IP to the design block:

The first block to add is the board's SoC:

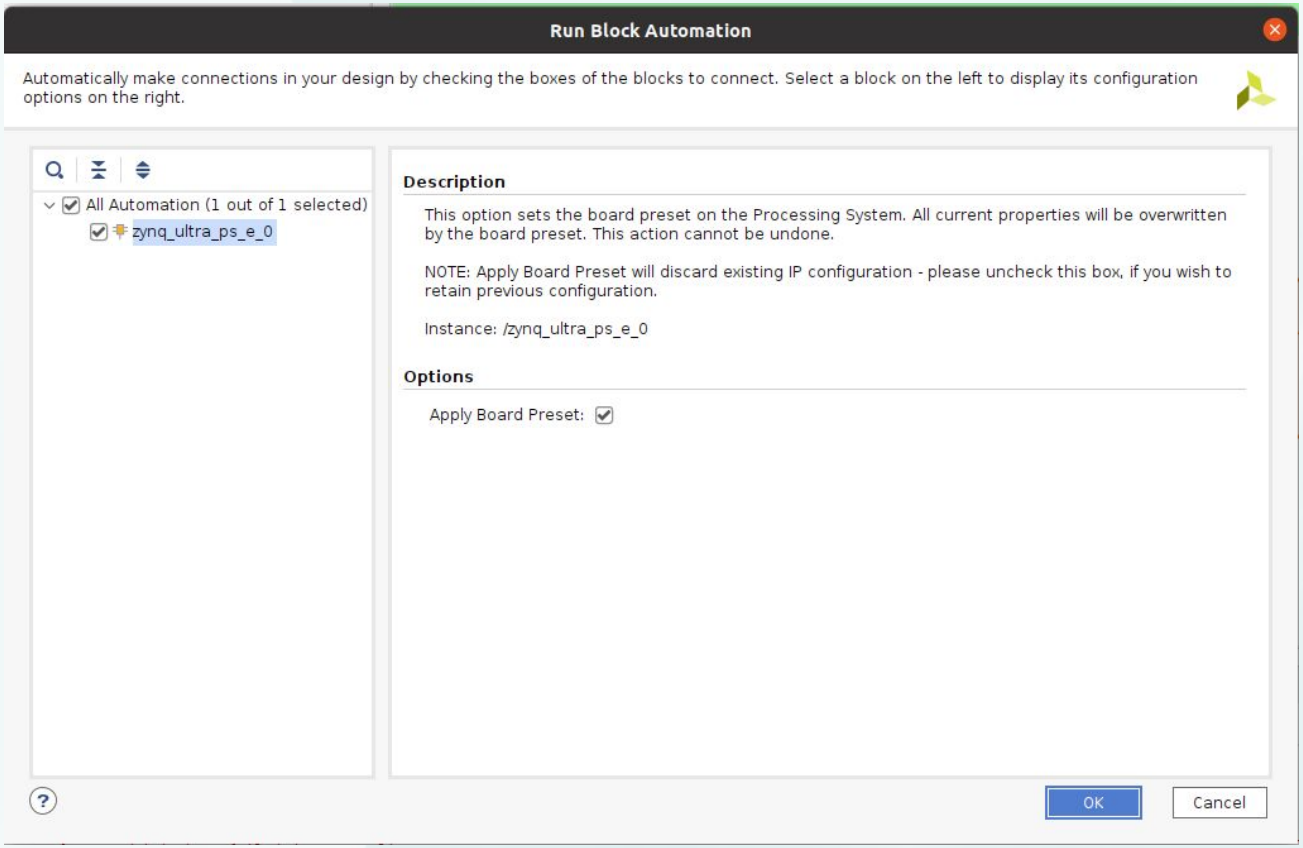
-Press add IP (+);

-Search for Zynq Ultrascale+ MPSoC.



Creating the Hardware Platform

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



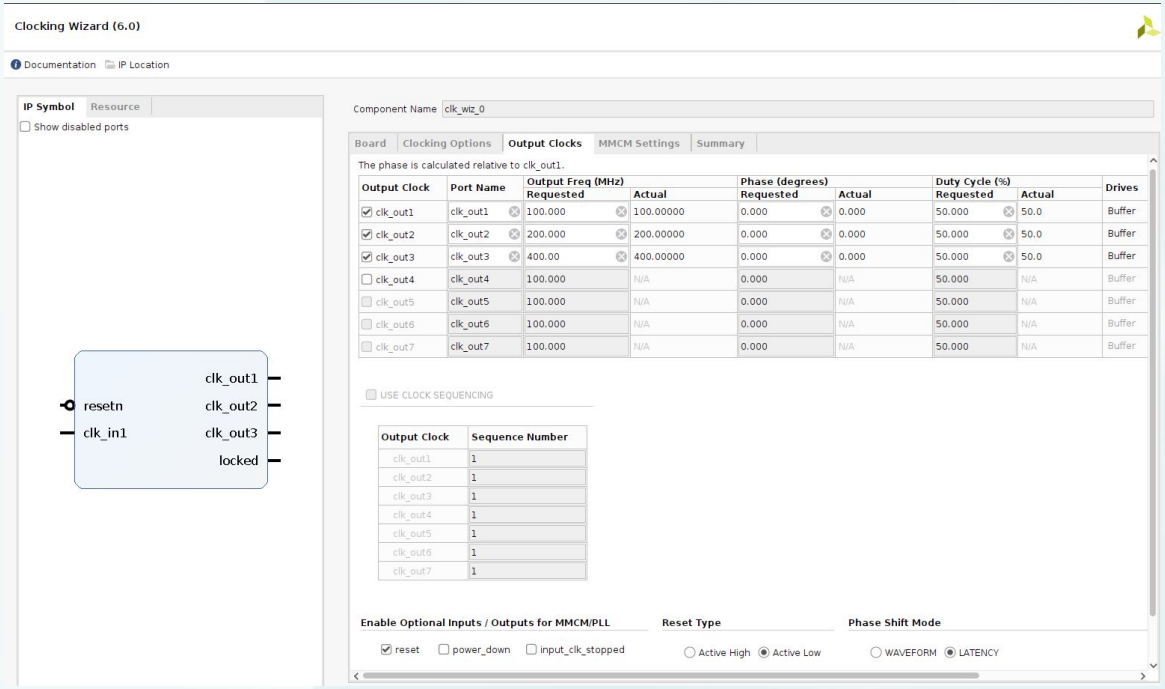
Creating the Hardware Platform

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

This step has applied all board presets for thZCU104, which include block configurations and pin assignments.

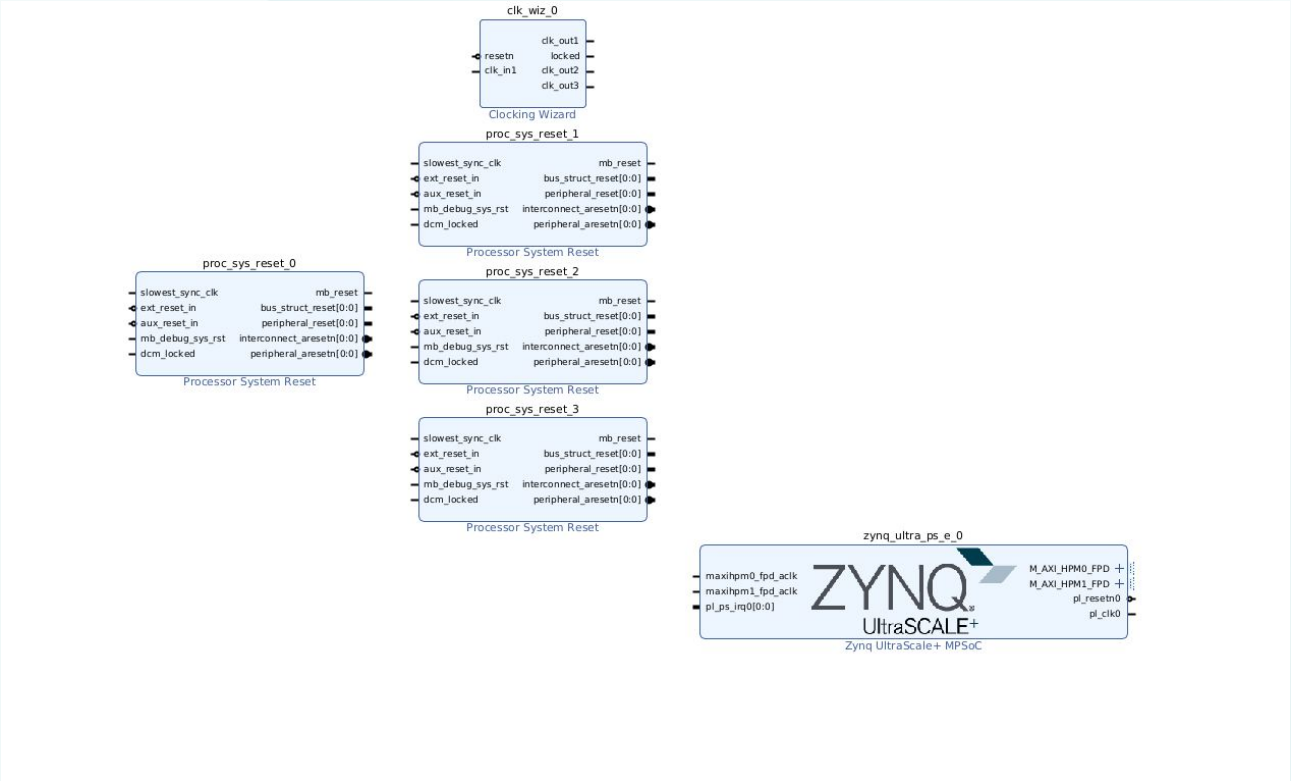
Creating the Hardware Platform

- Next, we need to add a clock wizard to the system:
  - Add the Clocking Wizard IP;
  - Enable and configure clk_out1 -> clk_out3.



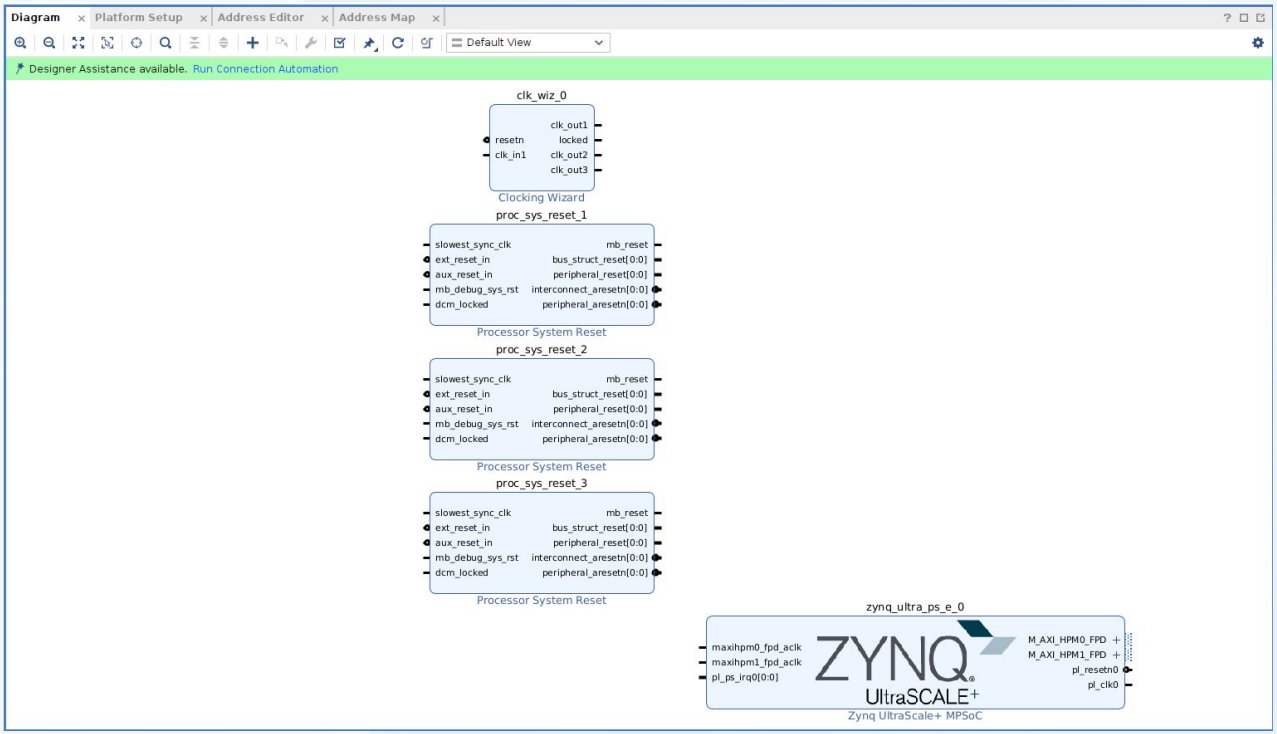
Creating the Hardware Platform

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



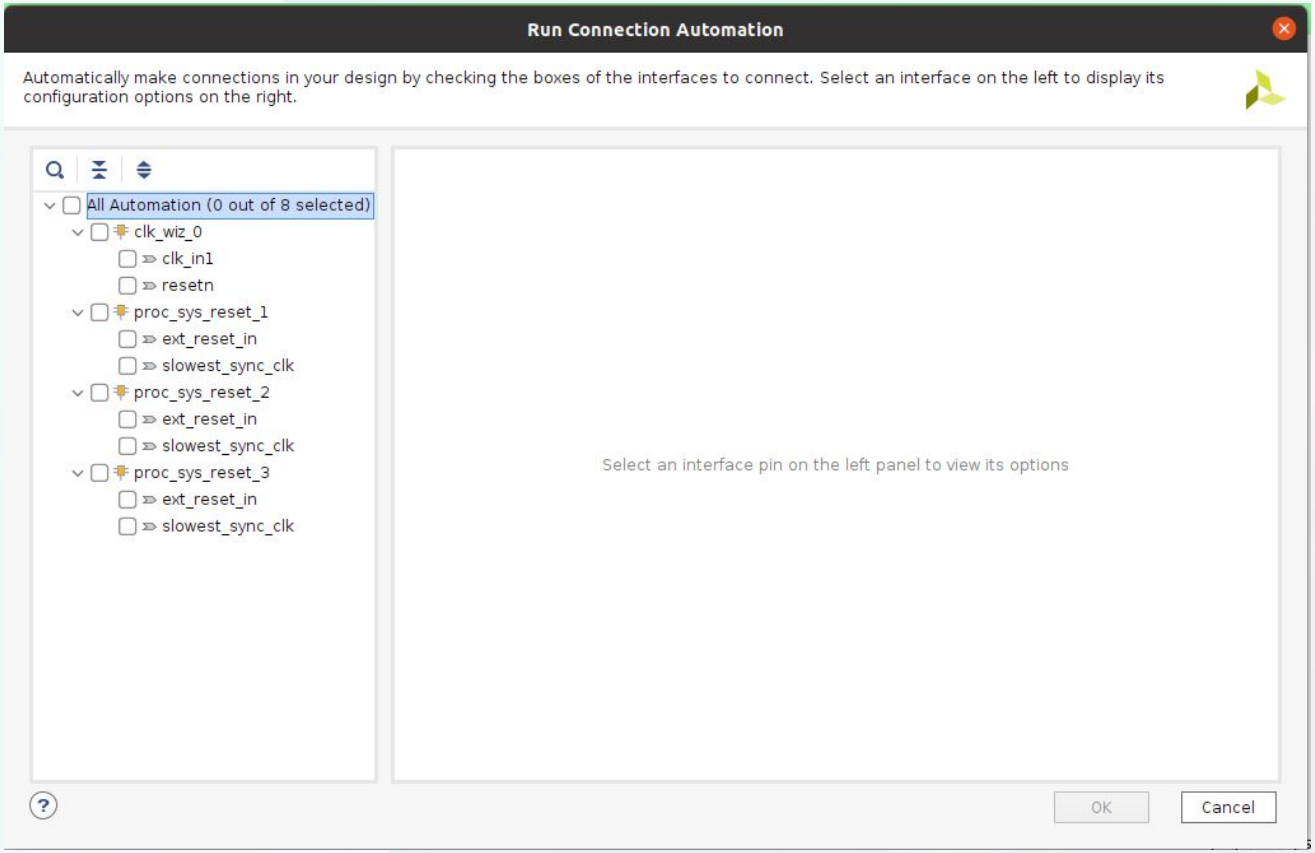
Creating the Hardware Platform

- We will also need a reset block:
  - Delete proc_sys_reset_0, in order to keep proc_sys_reset_1 -> proc_sys_reset_3



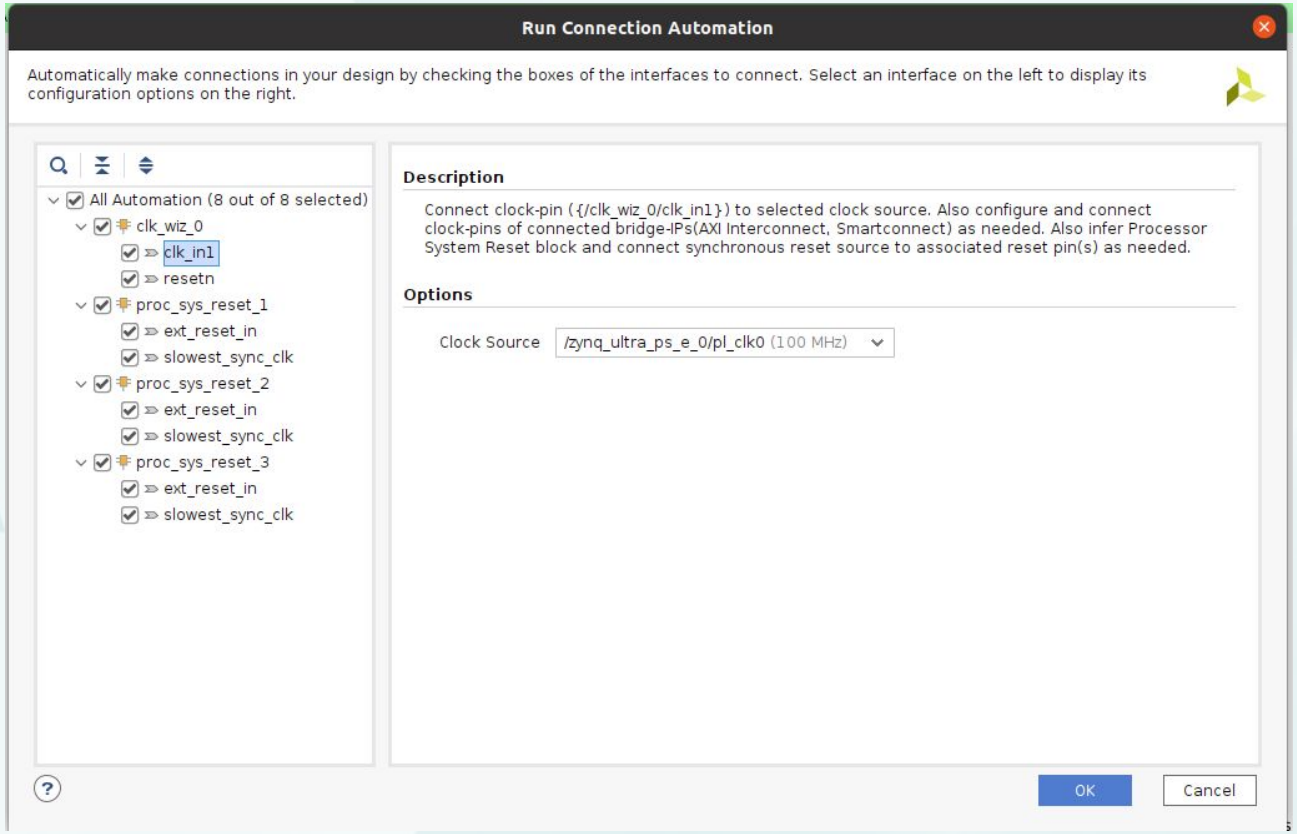
Creating the Hardware Platform

- Next, we should run the connection automation:



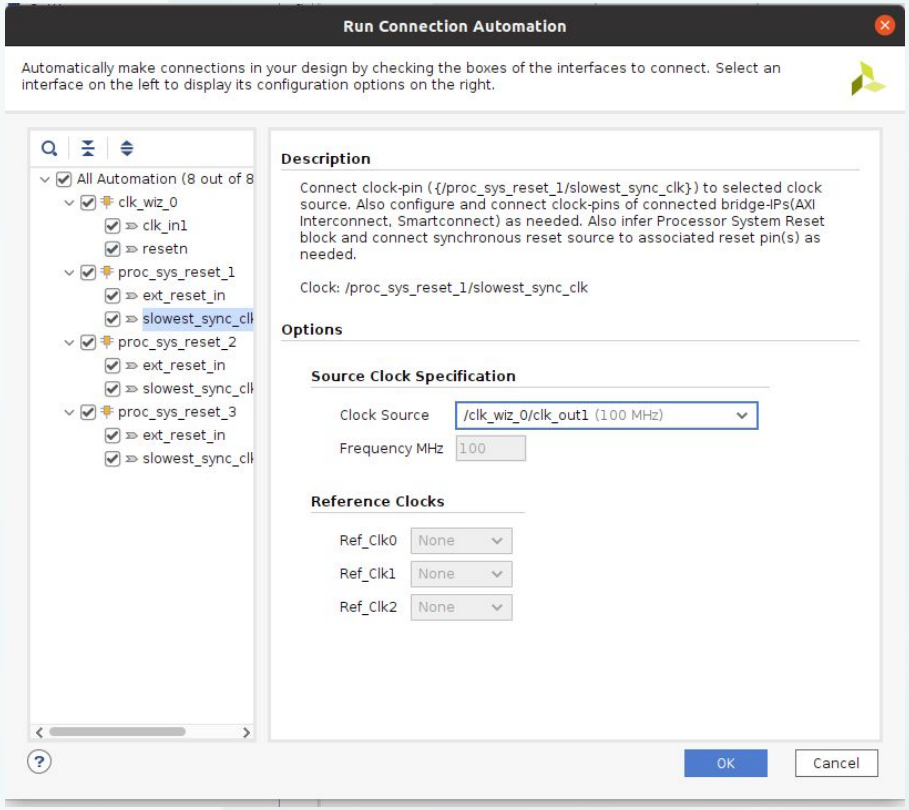
Creating the Hardware Platform

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



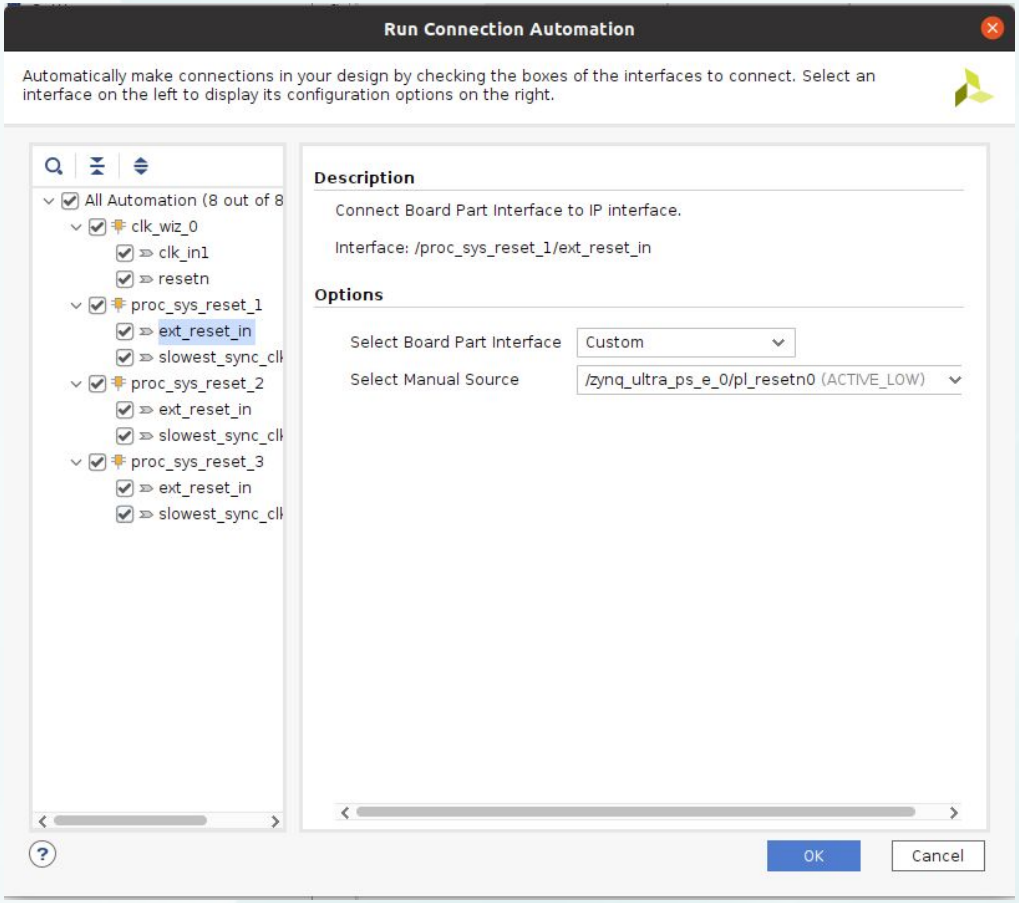
Creating the Hardware Platform

- Under “slowest_sync_clk” set the clock source as “/clk_wiz_0/clk_out_n”;
- Repeat for each reset block:



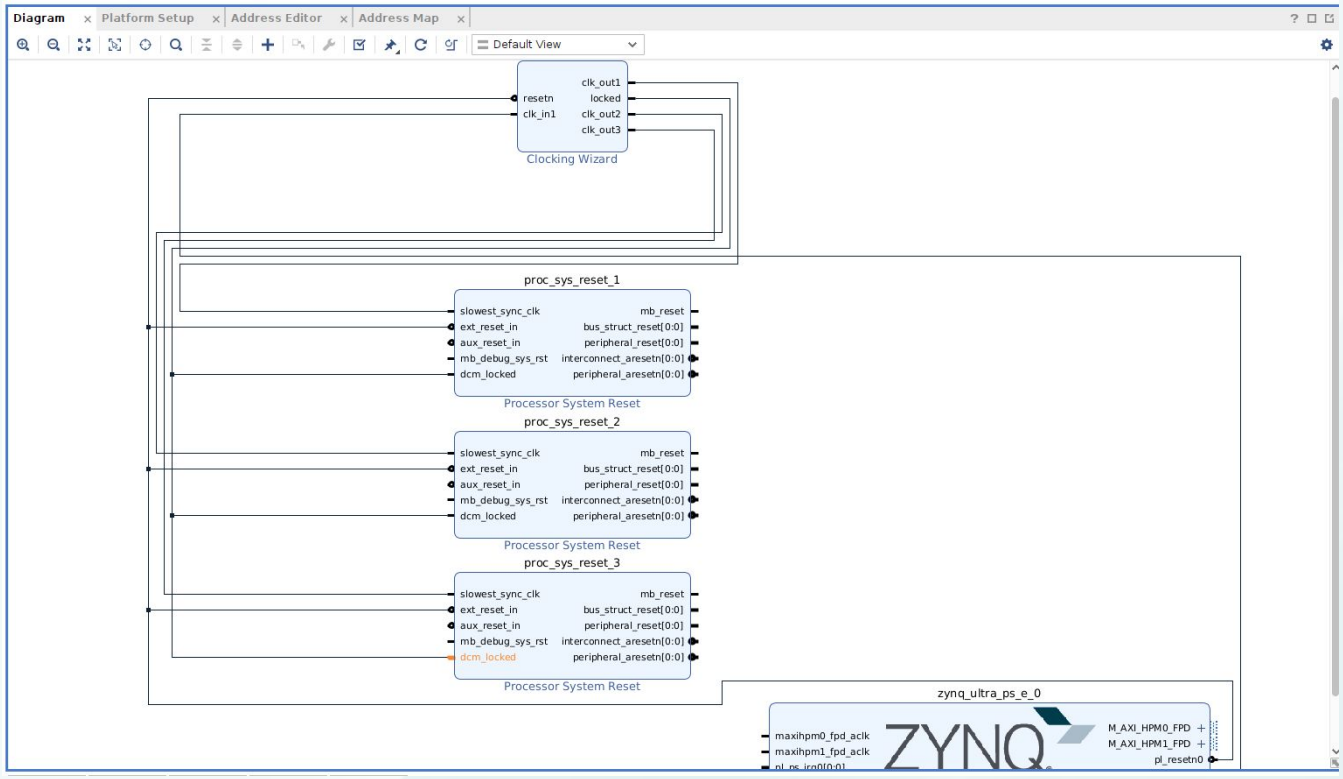
Creating the Hardware Platform

- On each proc_sys_reset instance set the following settings:



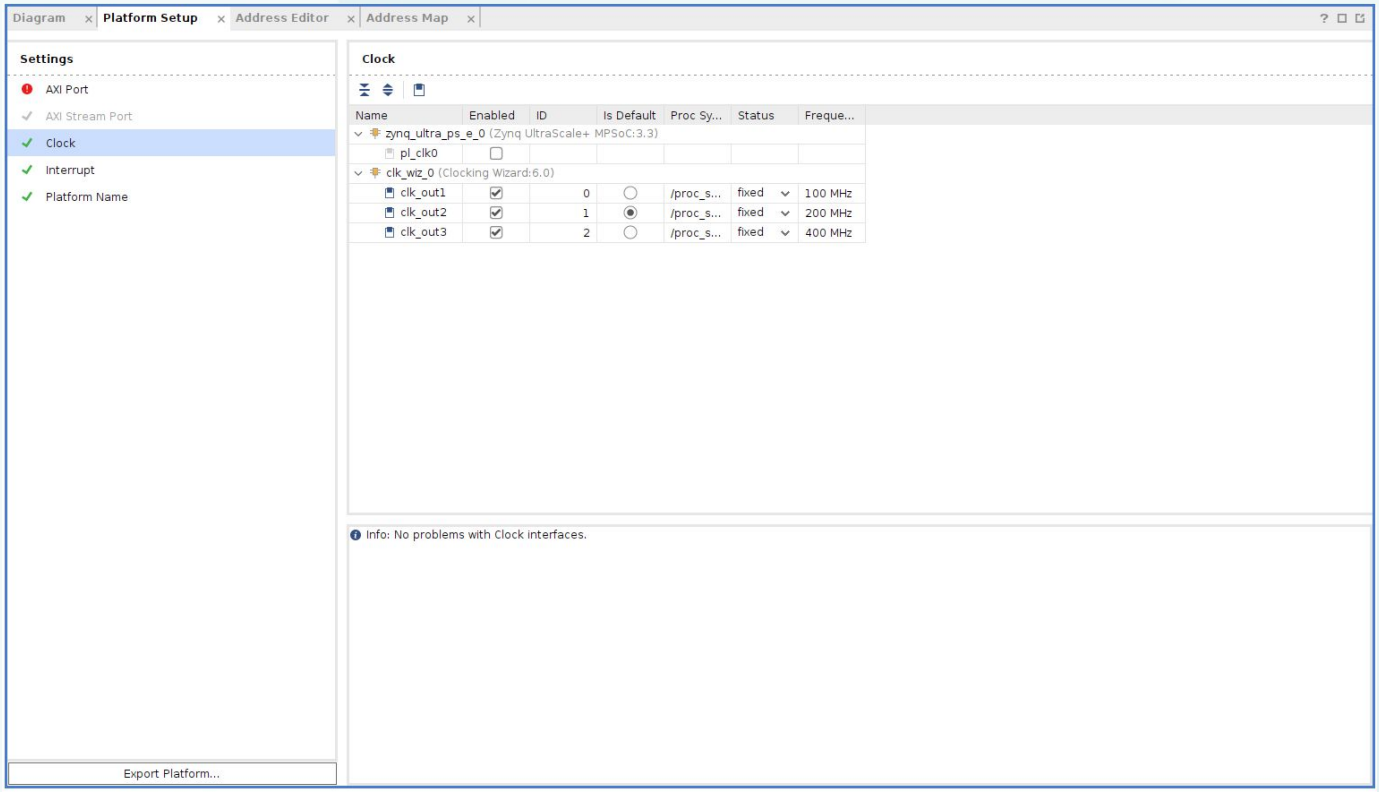
Creating the Hardware Platform

- Finally, we must connect all of the "dcm_locked" signals to the locked signal on clk_wiz_0:



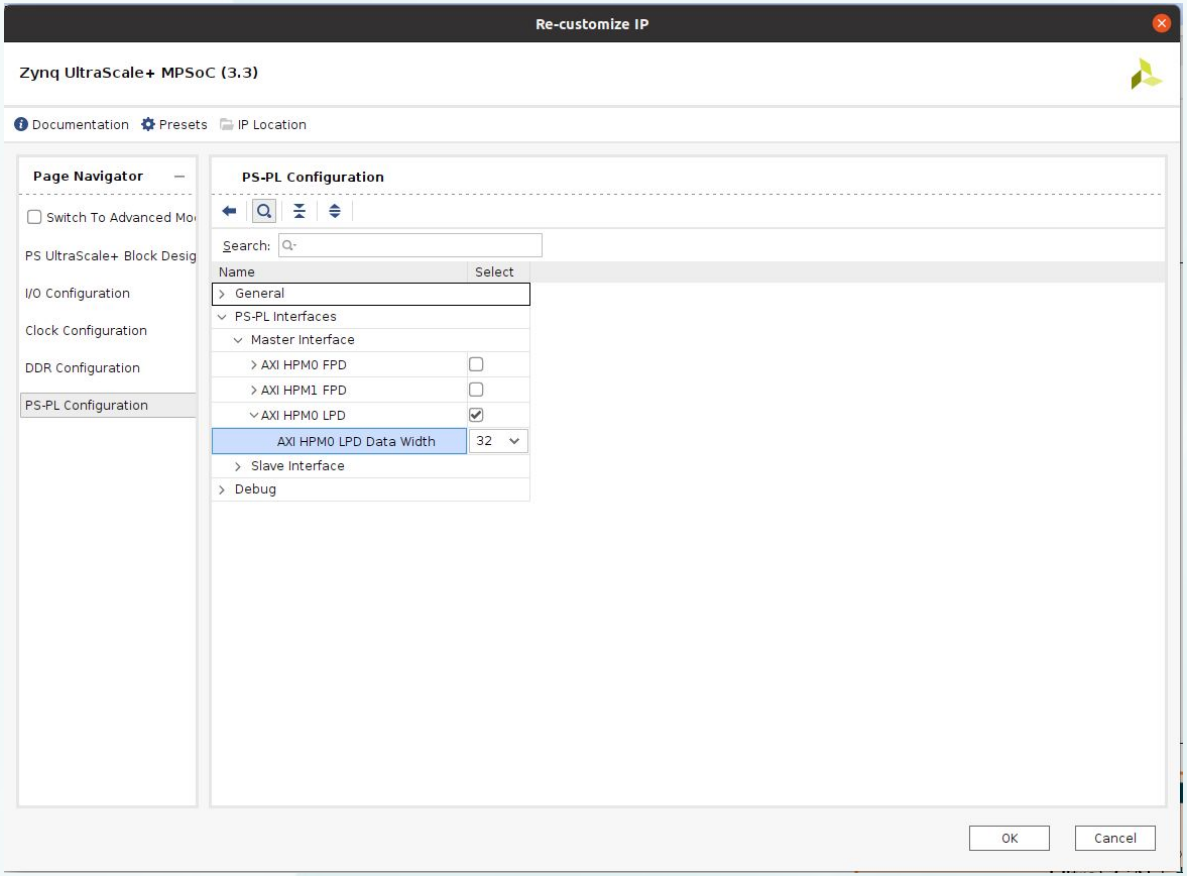
Creating the Hardware Platform

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



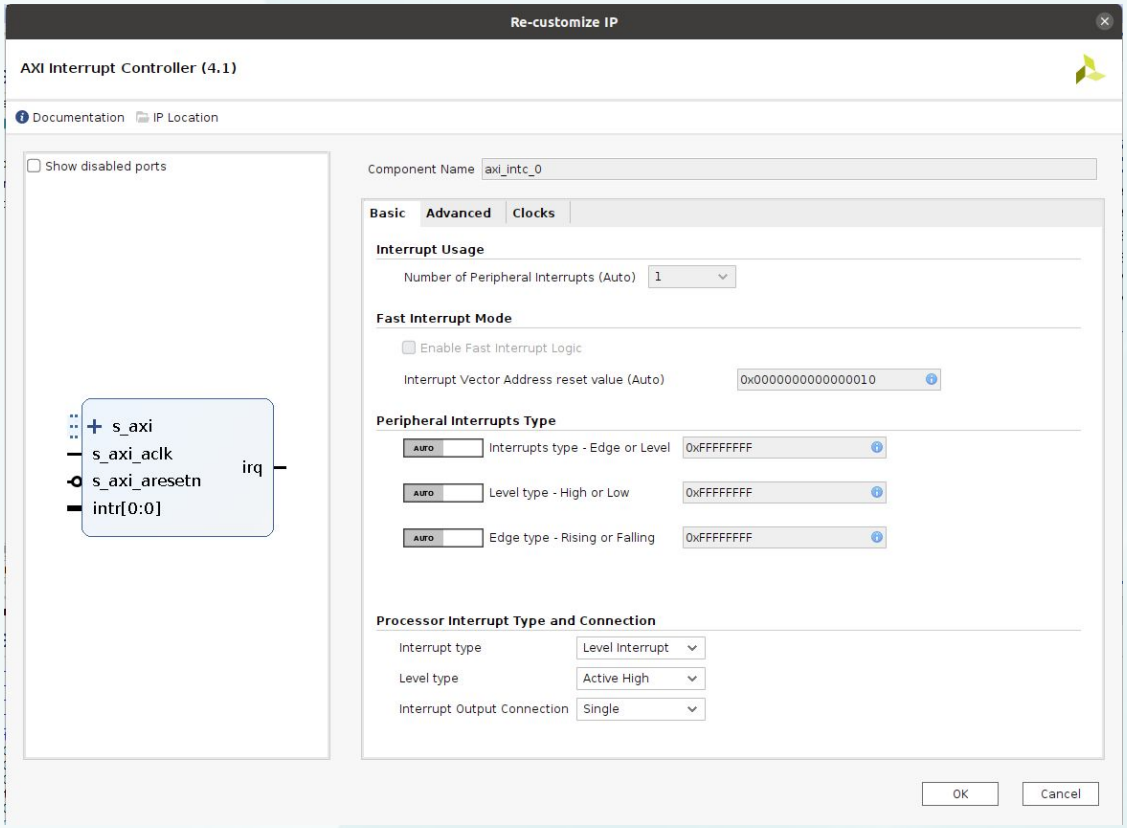
Creating the Hardware Platform

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



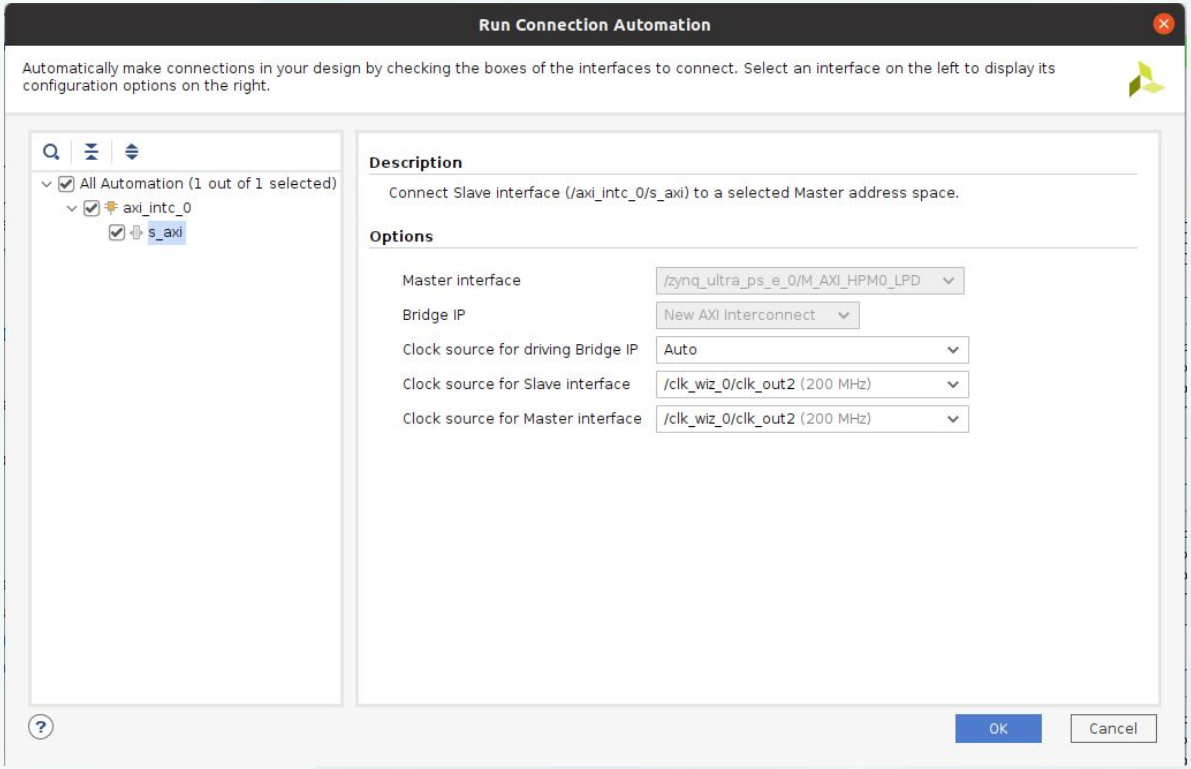
Creating the Hardware Platform

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



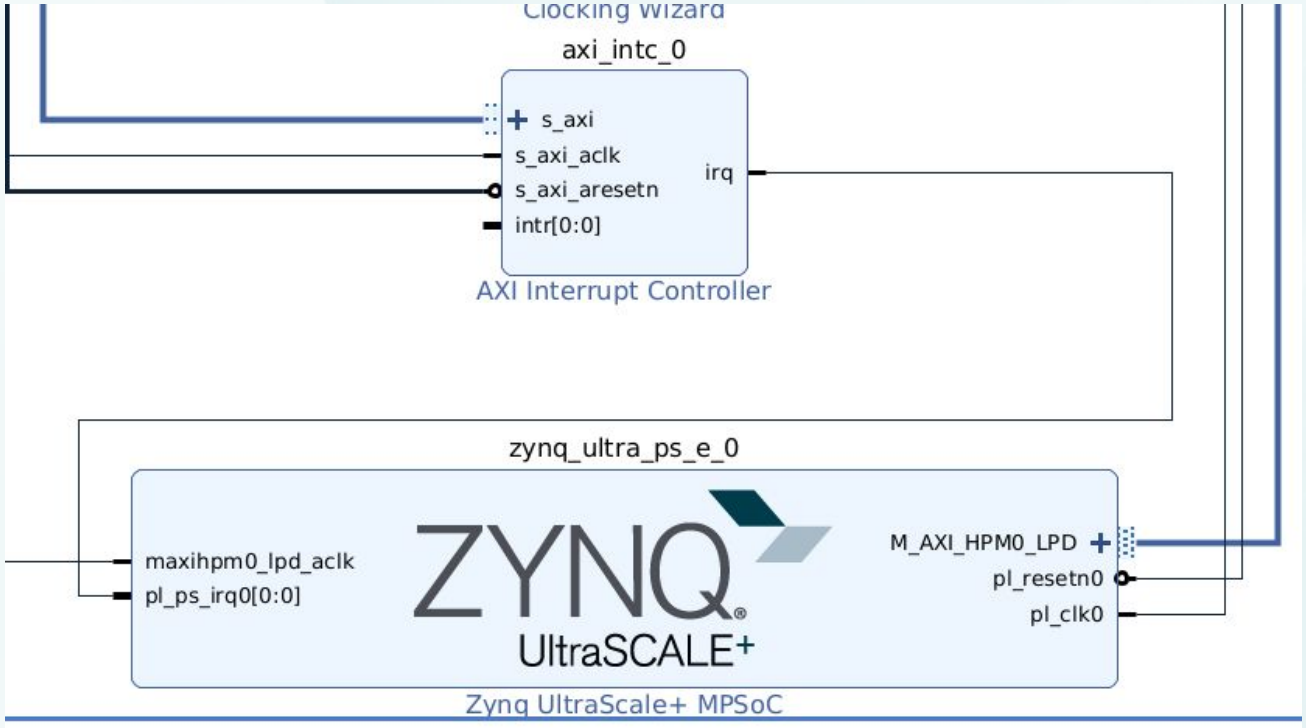
Creating the Hardware Platform

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



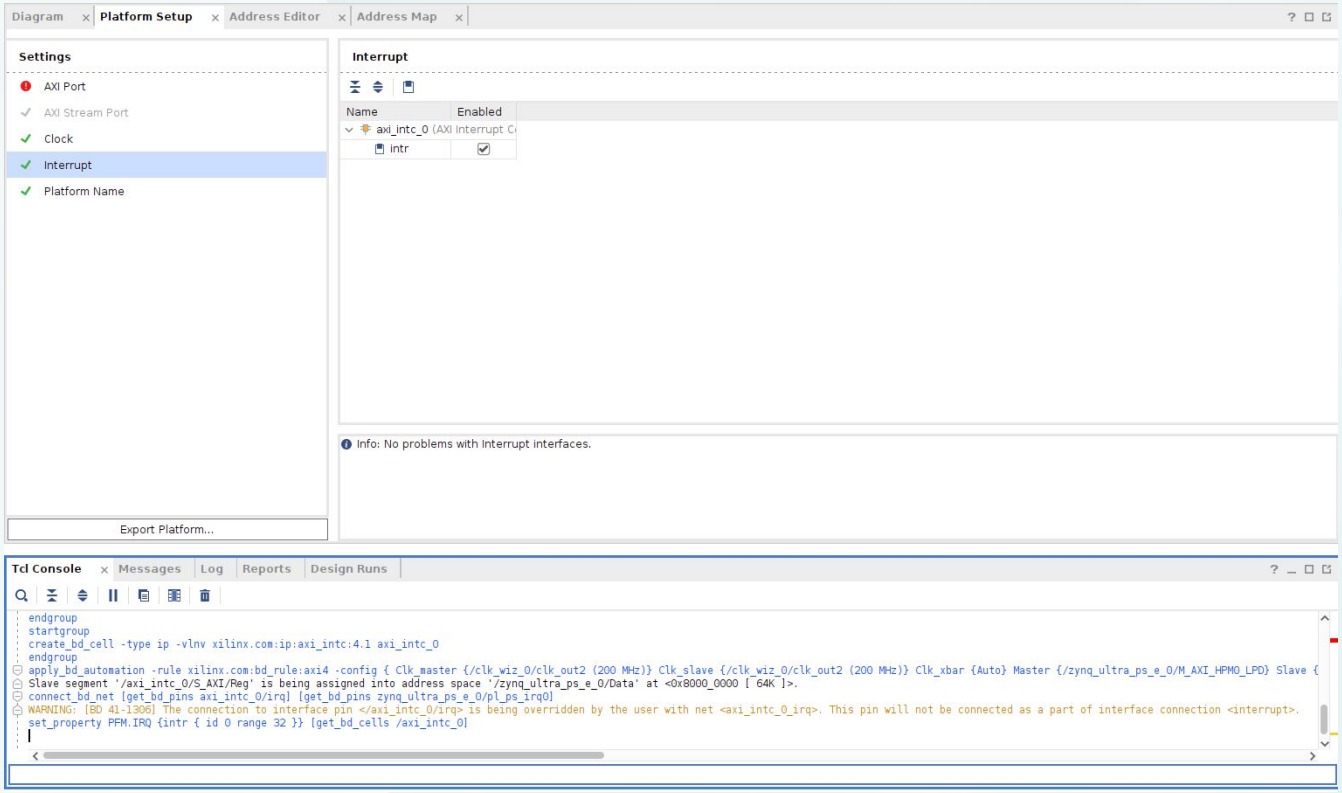
Creating the Hardware Platform

- We now should add interrupt support to our system:
  - Connect xi_intc_0.irq to zynq_ultra_ps_e_0.pl_ps_irq[0:0]:



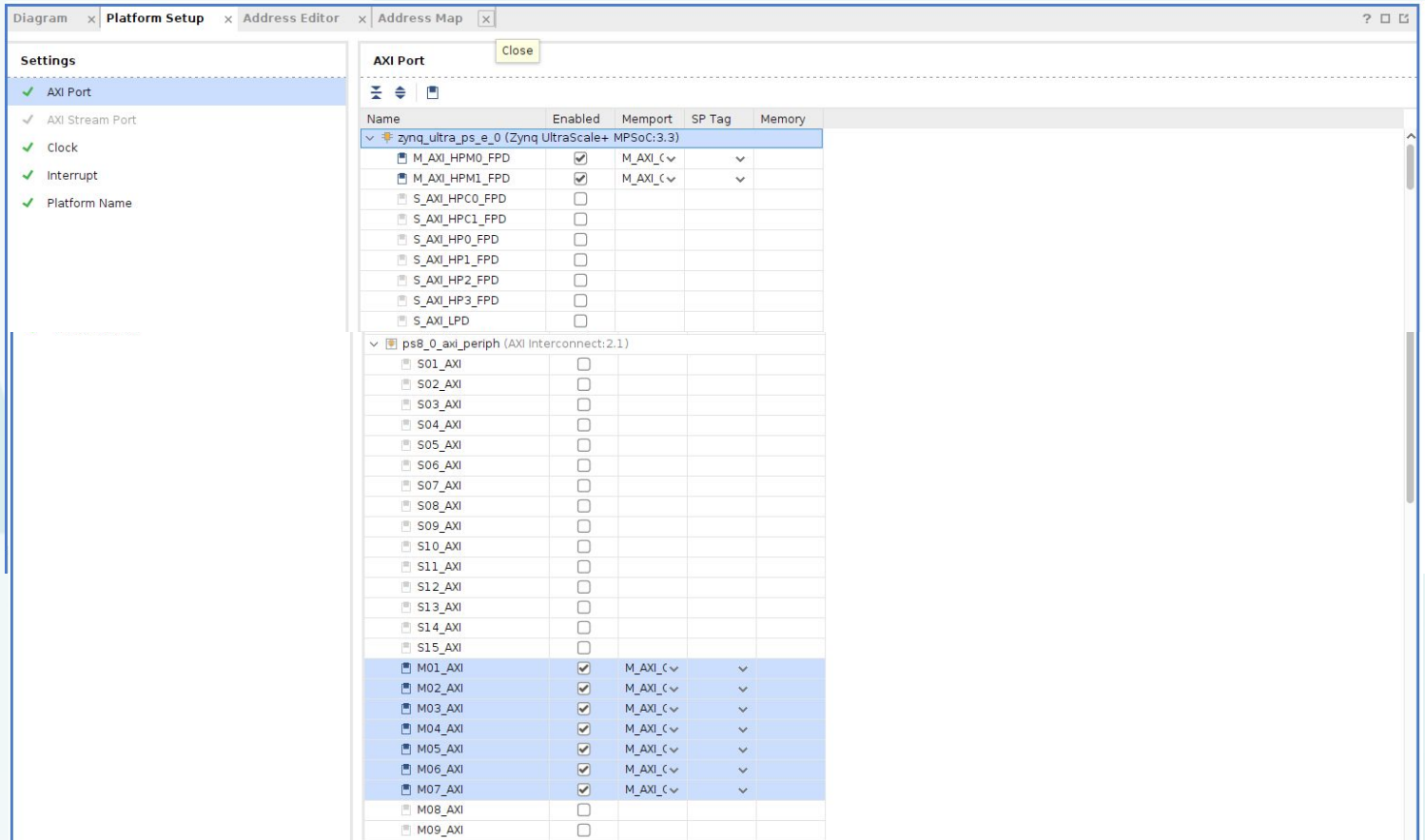
Creating the Hardware Platform

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



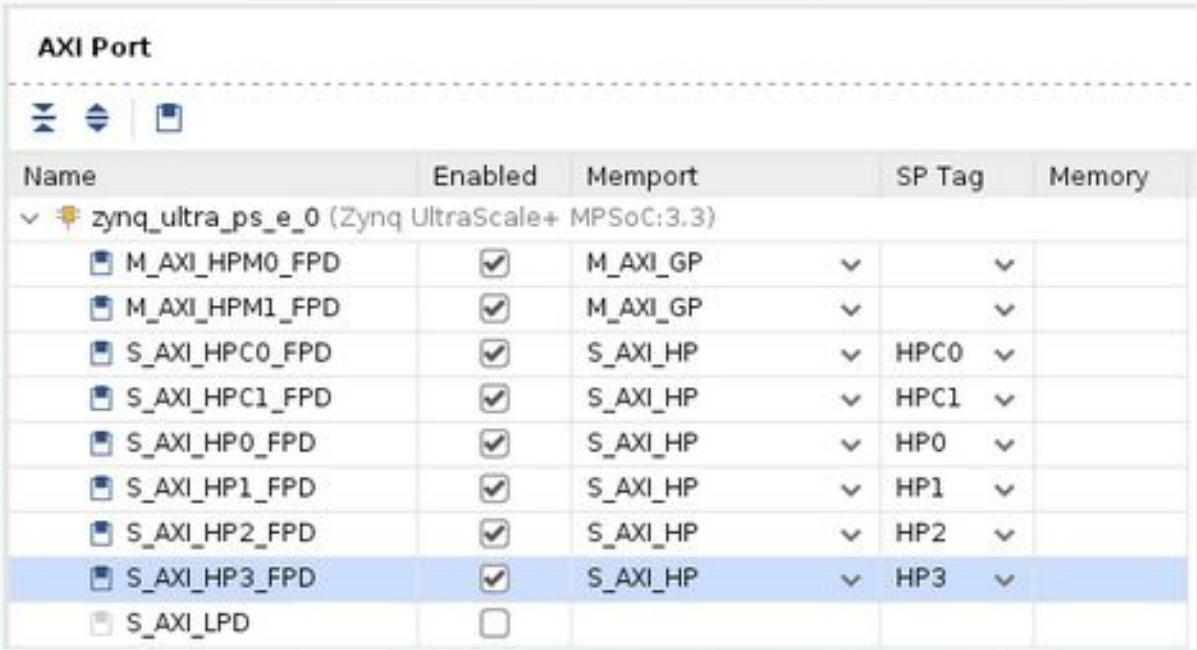
Creating the Hardware Platform

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



Creating the Hardware Platform

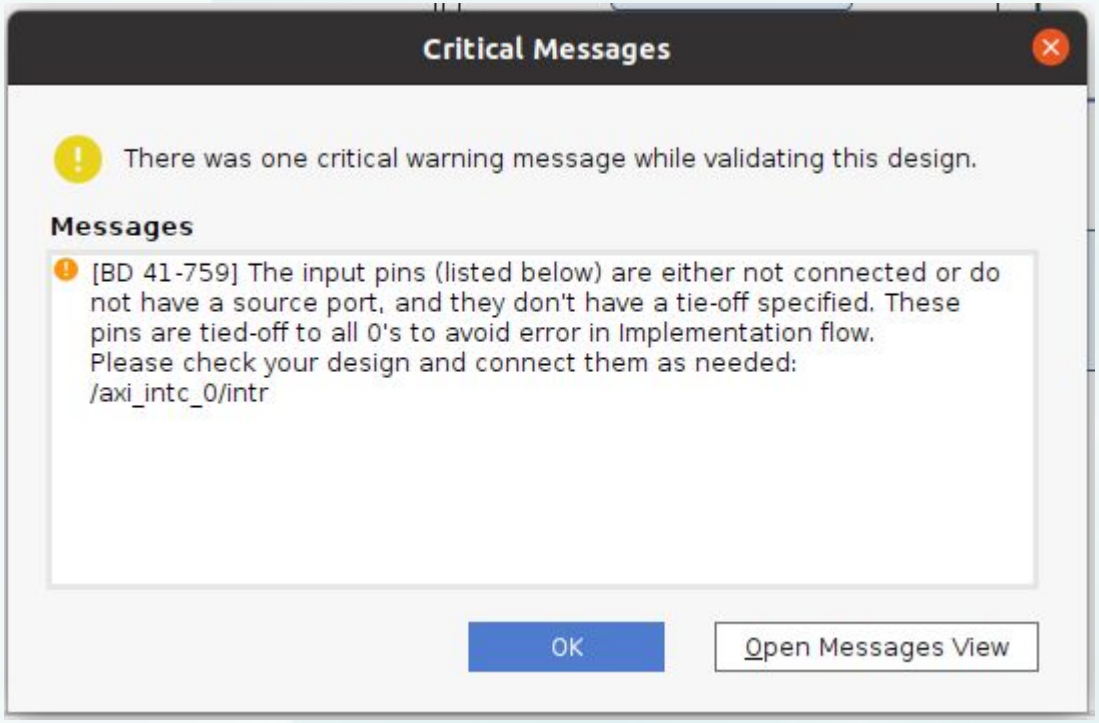
- It's time to enable the AXI interfaces:
  - Add the following SP tags:



Creating the Hardware Platform

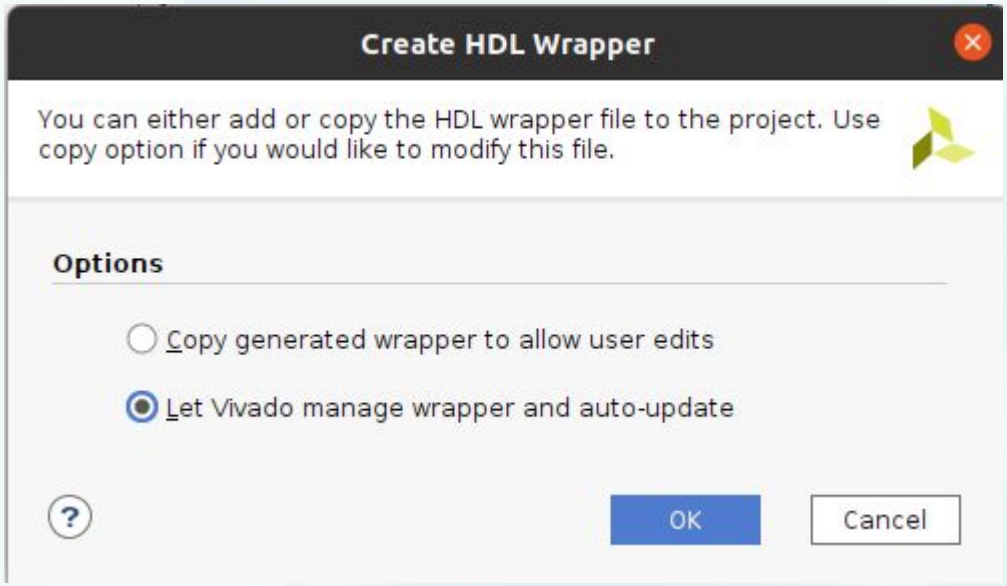
- Finally, it's time to export our design:
  - Validate the design:

The following error message can be ignored:



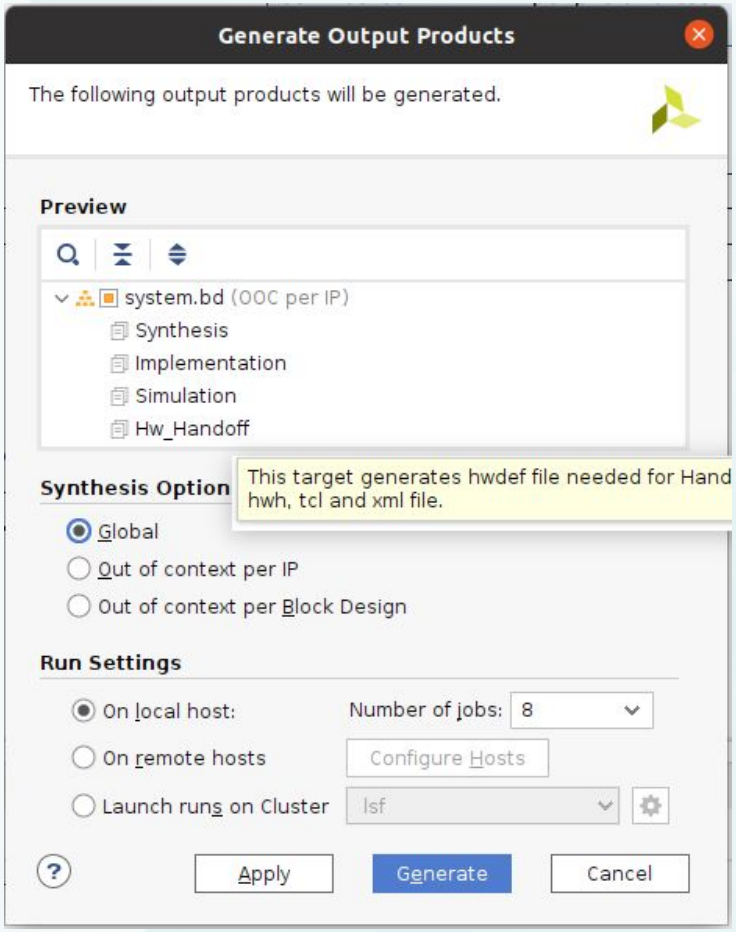
Creating the Hardware Platform

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



Creating the Hardware Platform

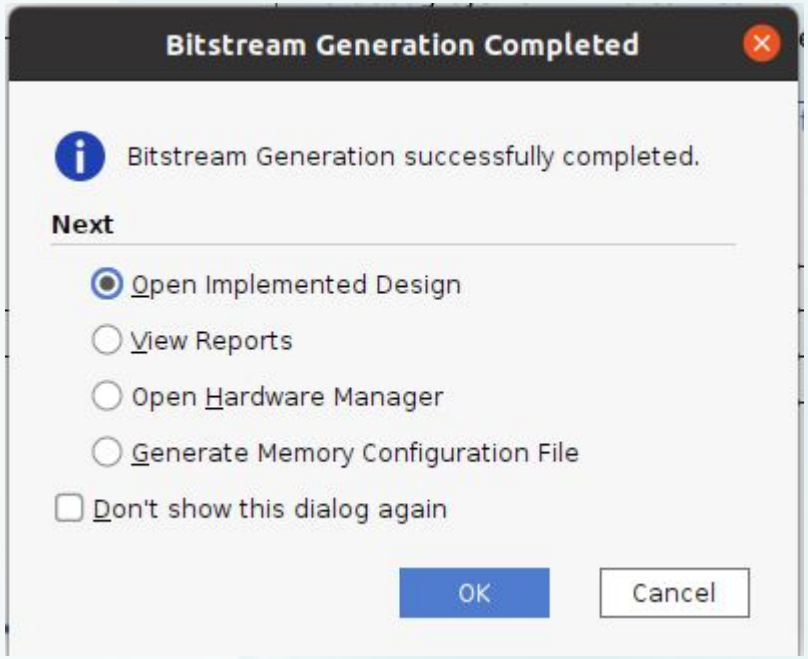
- Finally, it's time to export our design:
  -Generate pre-synth design -> Generate Block Design



Creating the Hardware Platform

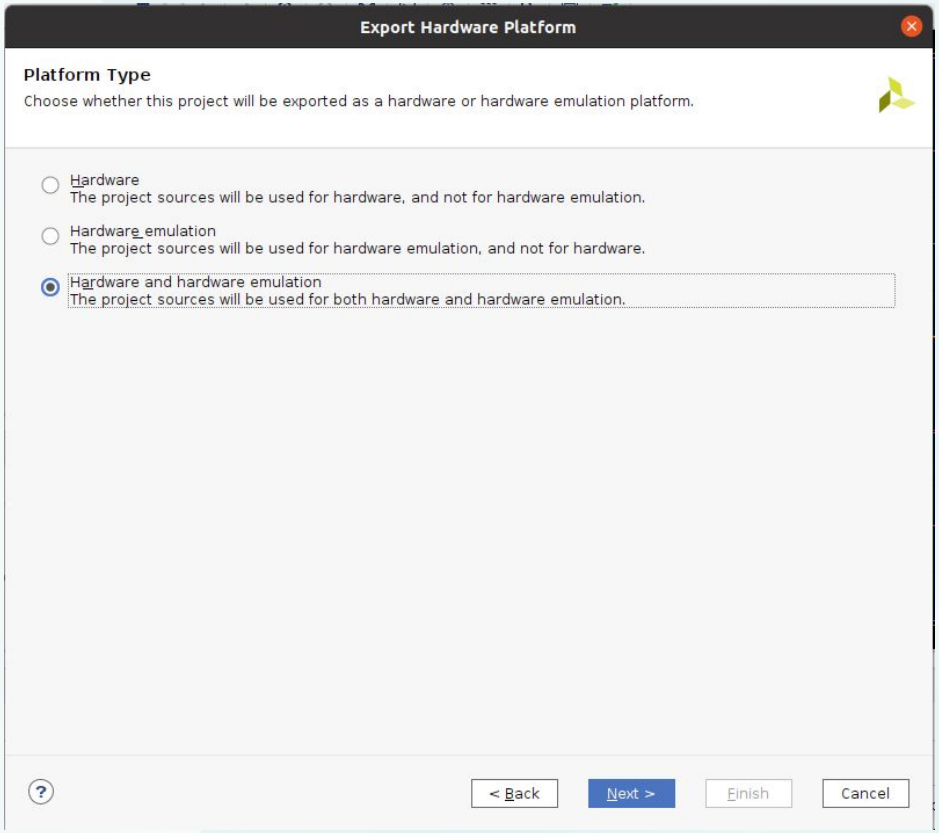
- Finally, it's time to export our design:
  -Generate Bitstream;

We should see this:



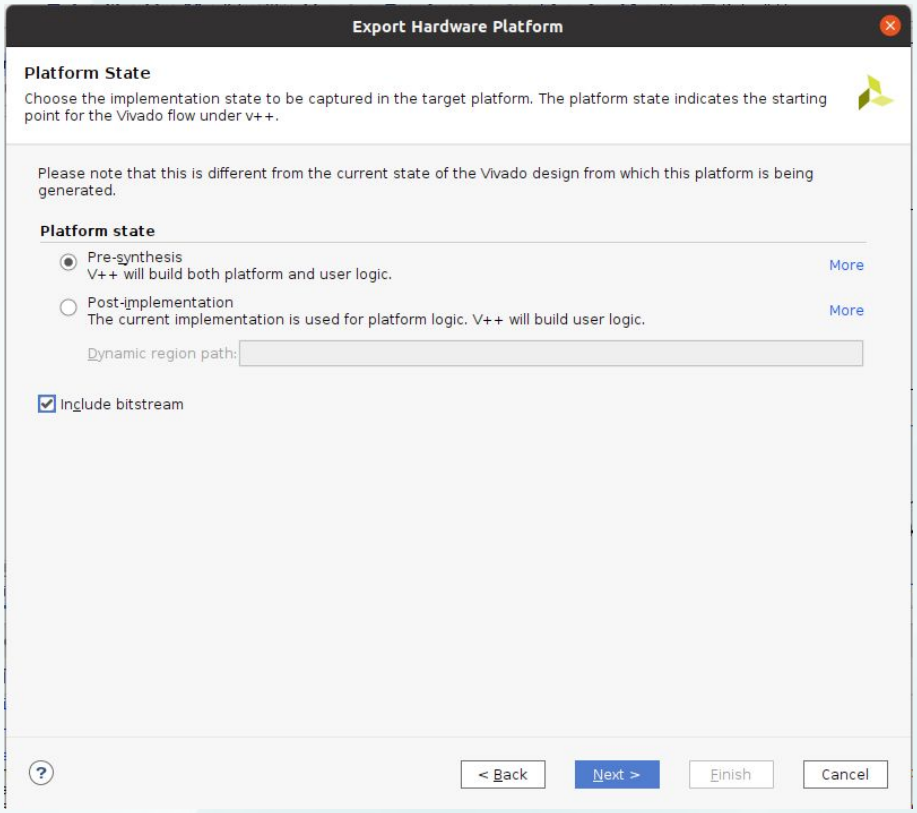
Creating the Hardware Platform

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



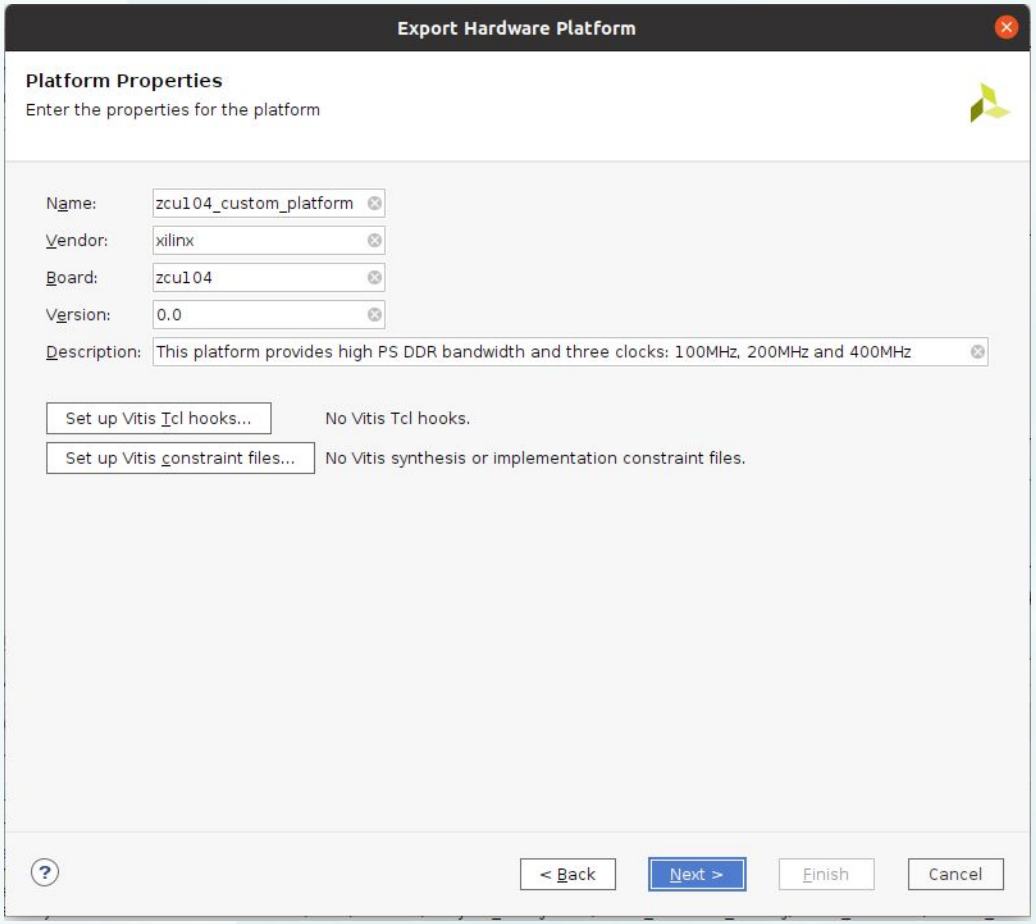
Creating the Hardware Platform

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



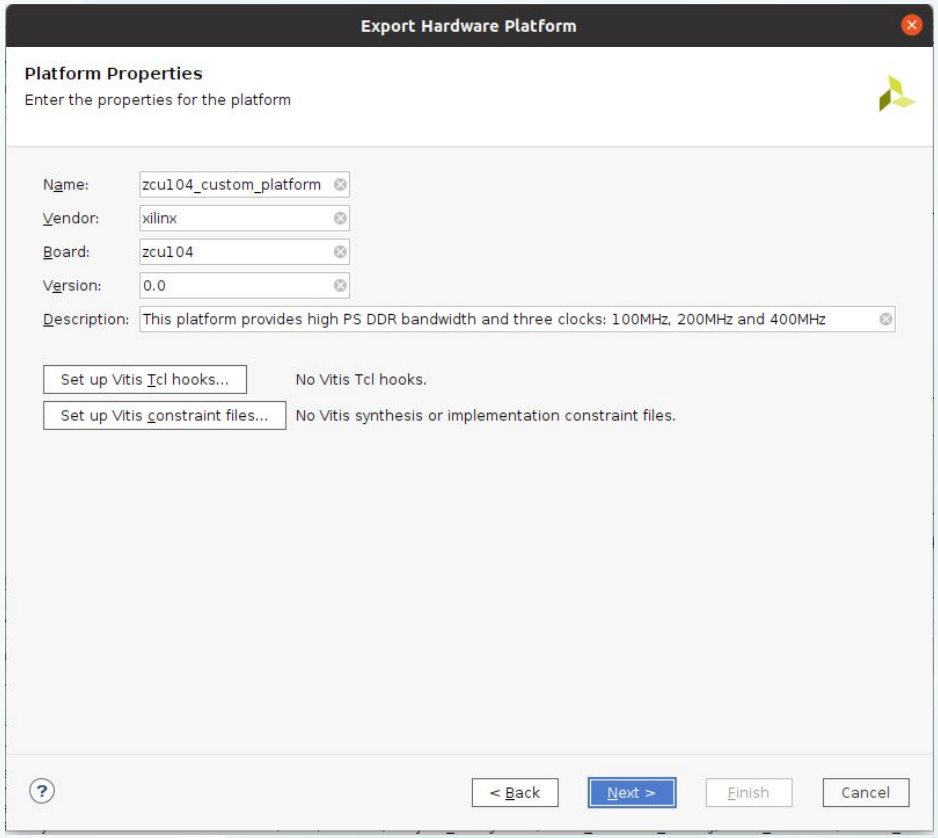
Creating the Hardware Platform

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



Creating the Hardware Platform

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

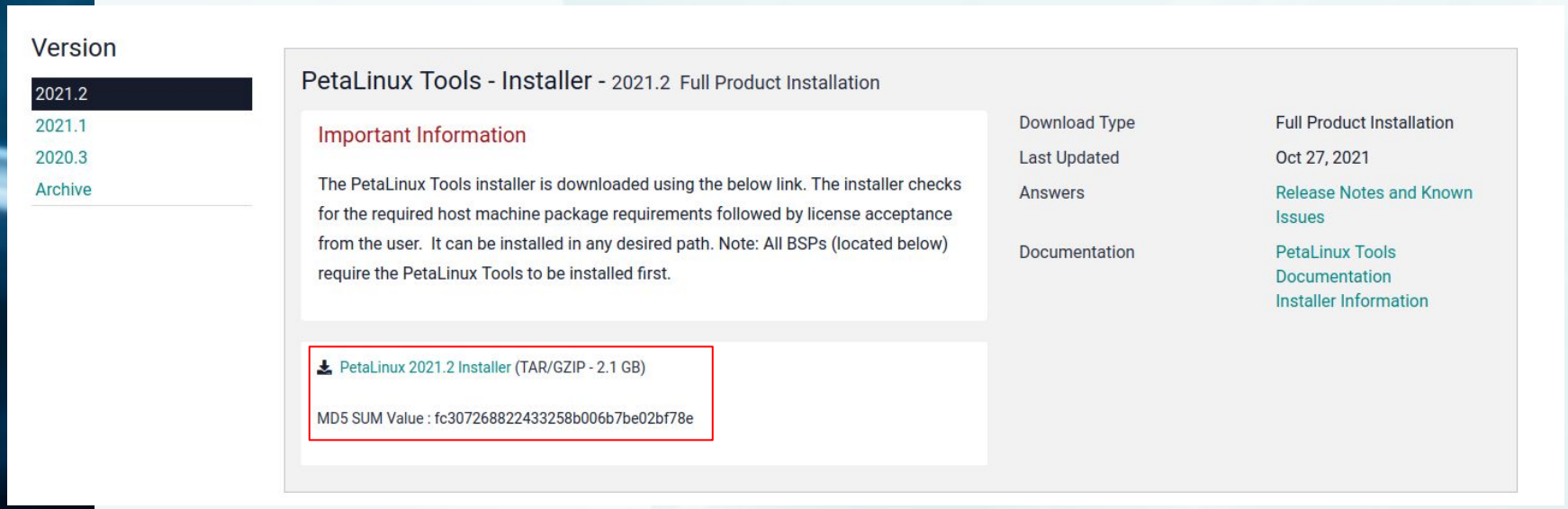


Creating the Linux Image

- 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 Xilinxs 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.

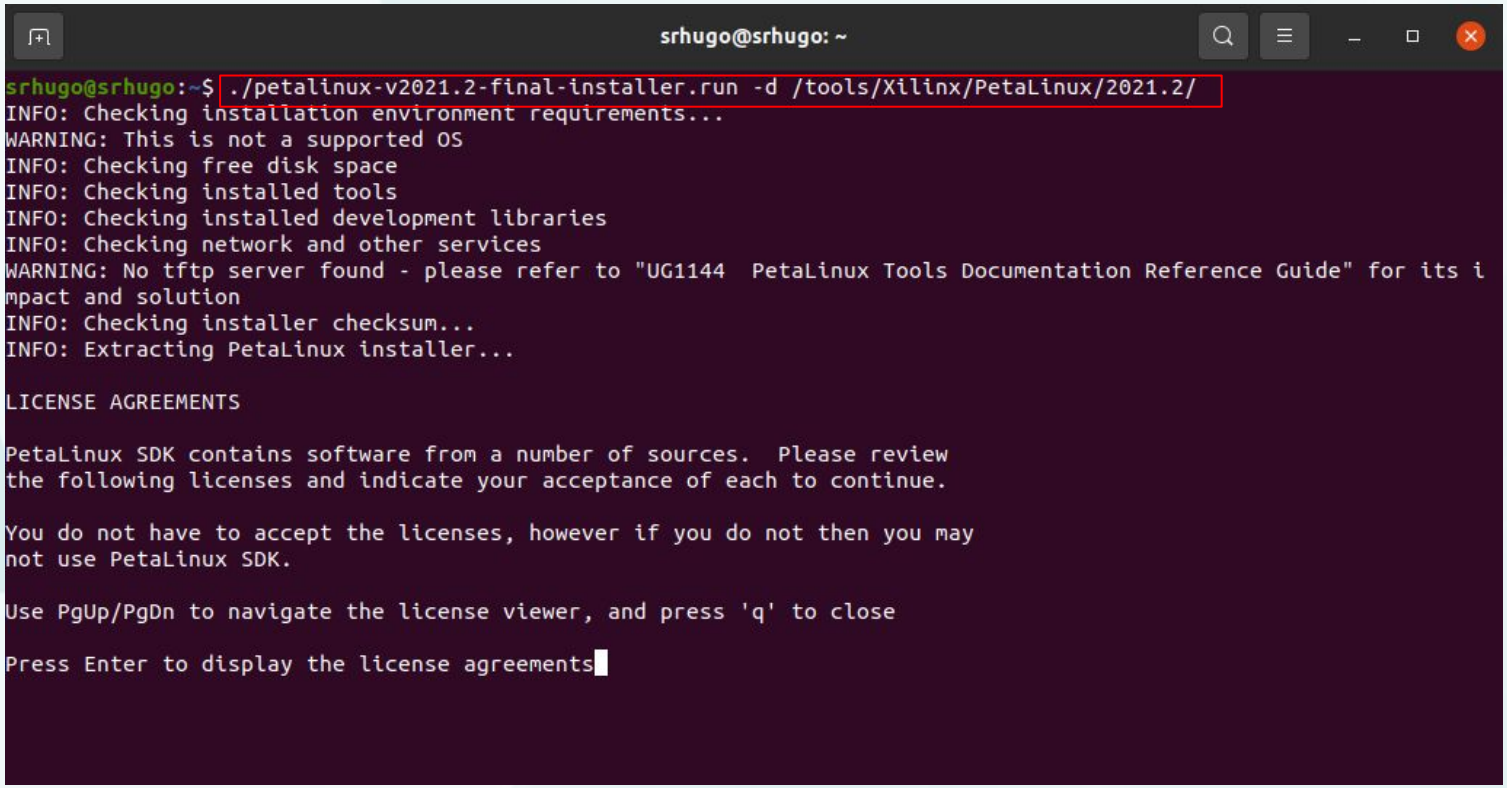
Creating the Linux Image

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



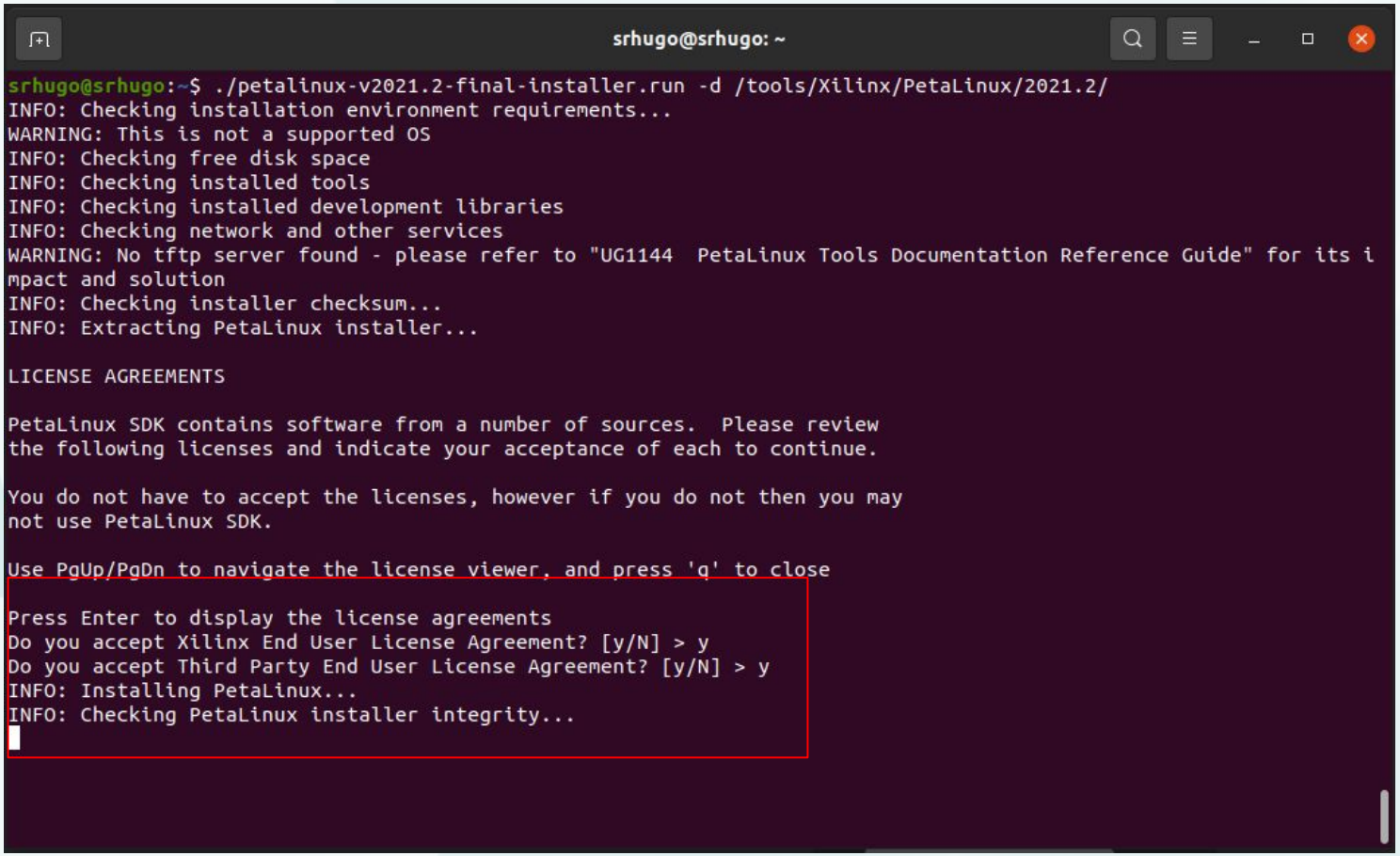
Creating the Linux Image

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



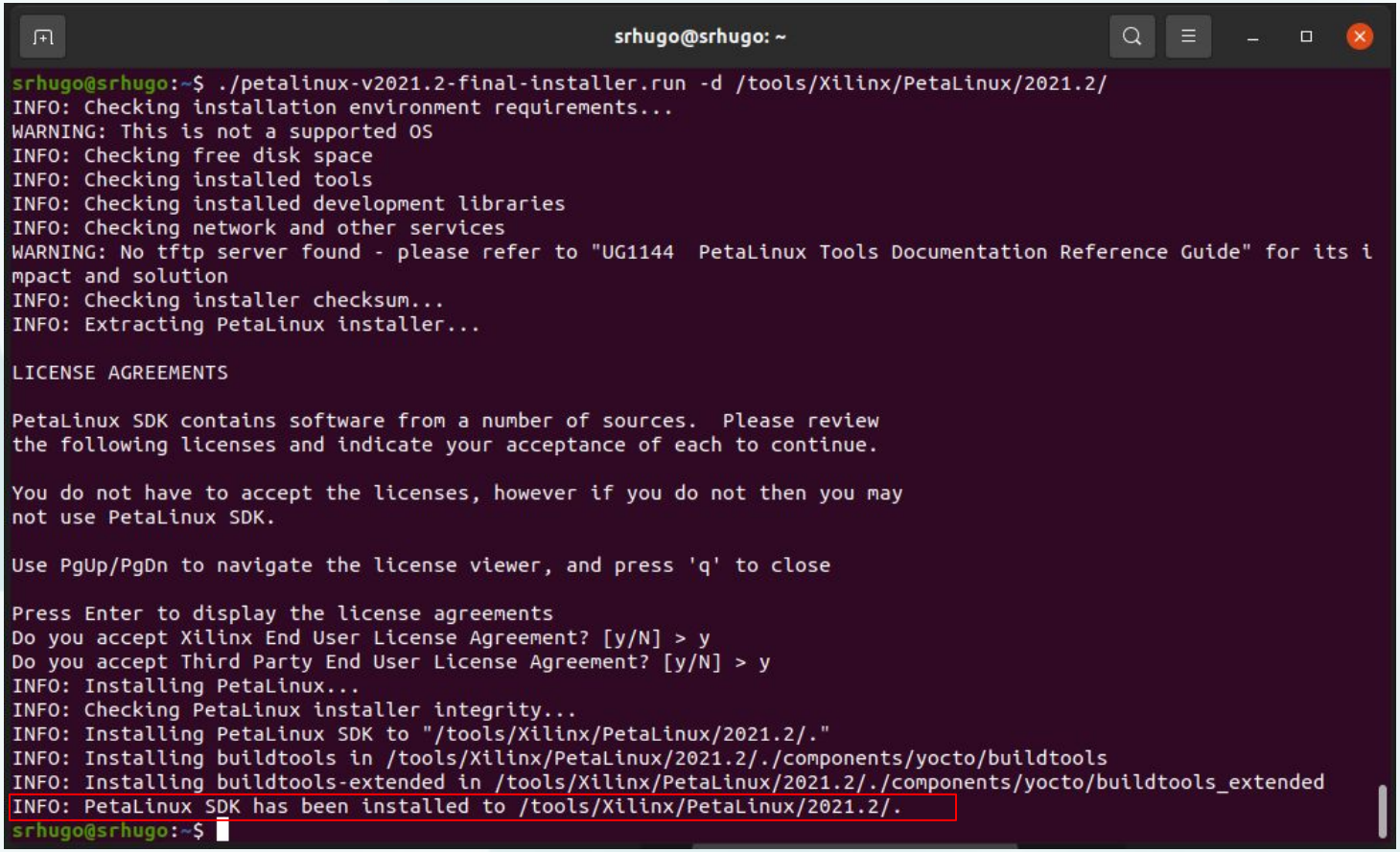
Creating the Linux Image

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



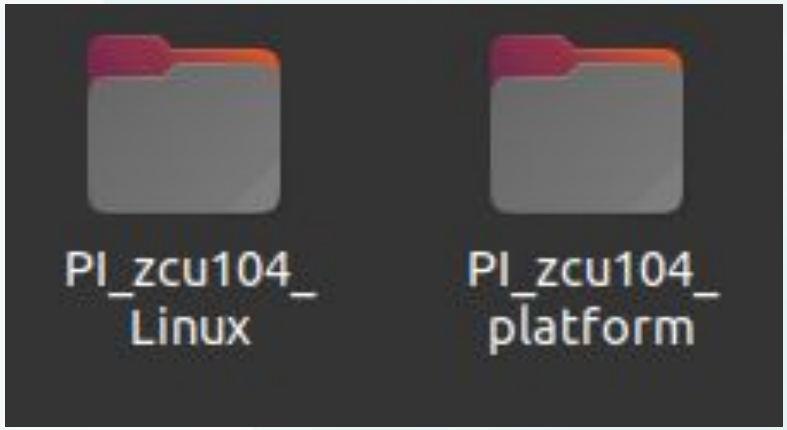
Creating the Linux Image

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



Creating the Linux Image

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



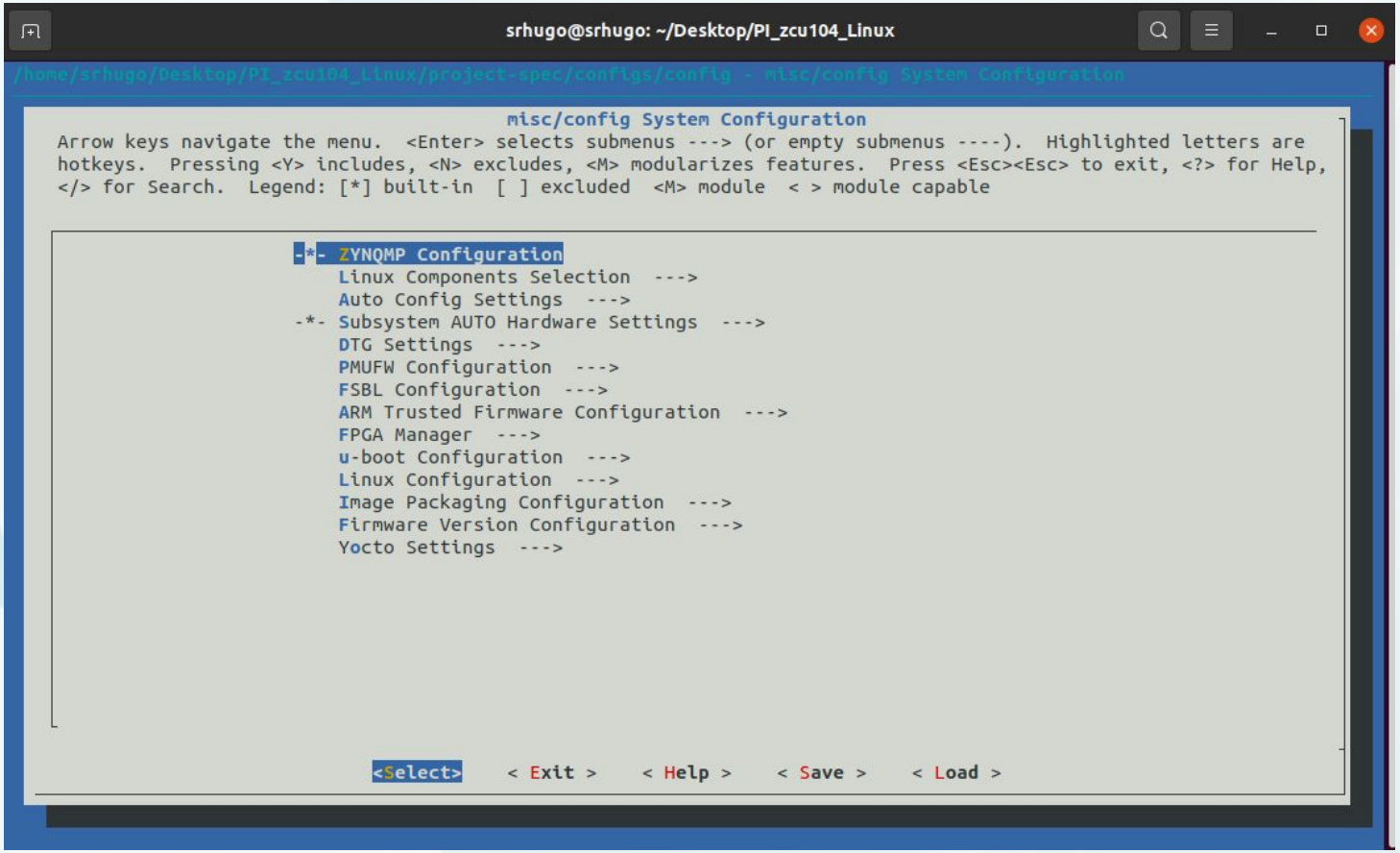
Creating the Linux Image

- 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
```

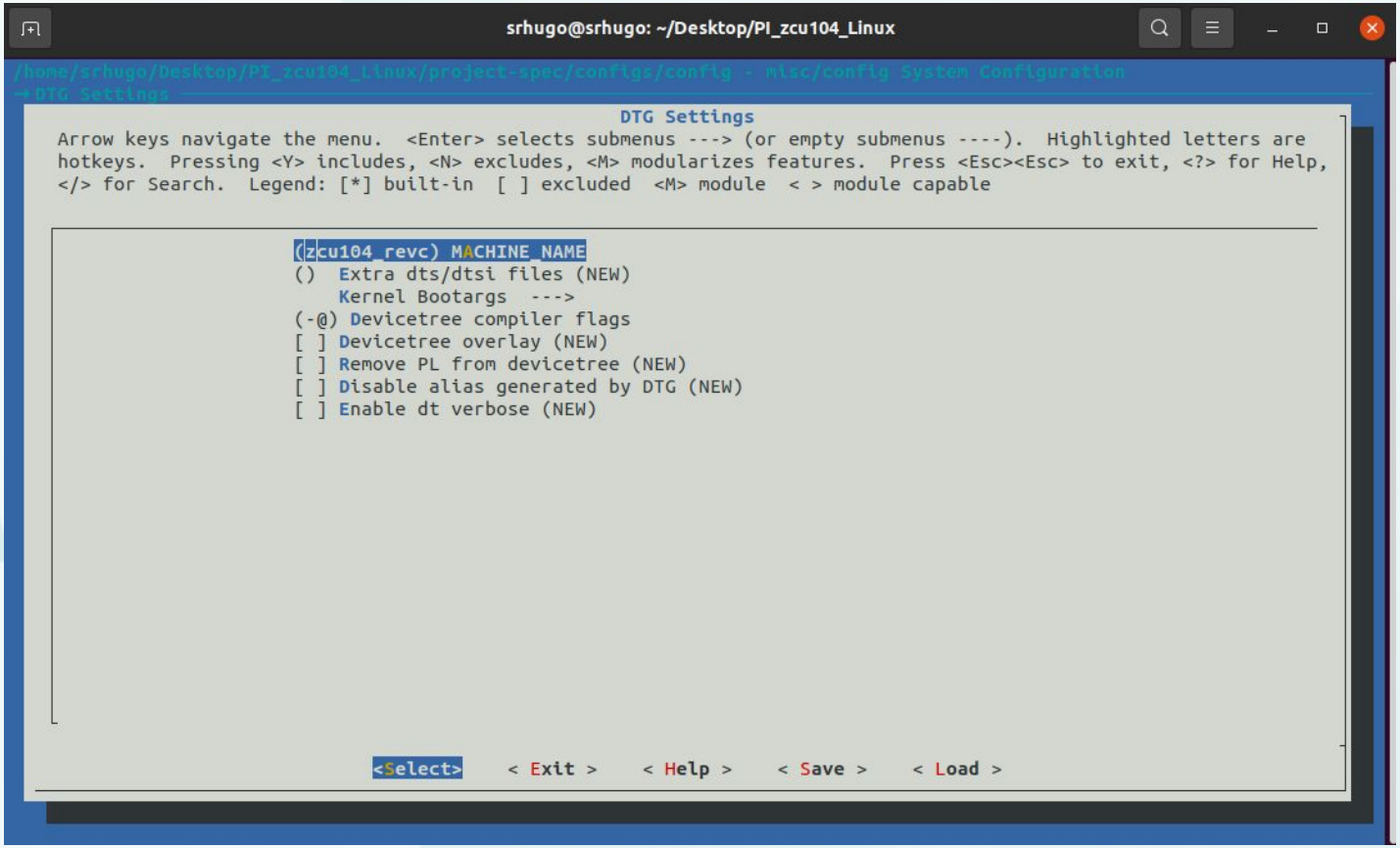
Creating the Linux Image

- The following menu should appear



Creating the Linux Image

- Go to DTG Setting -> MACHINE_NAME and change the name to zcu104_revc



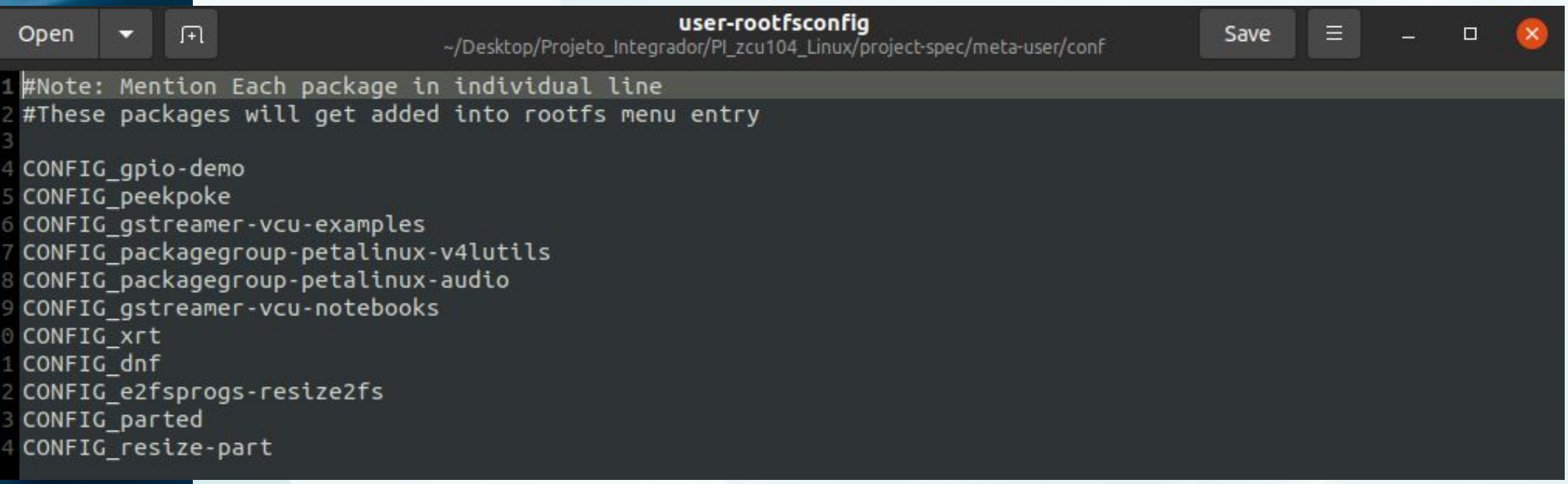
Creating the Linux Image

- 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

Creating the Linux Image

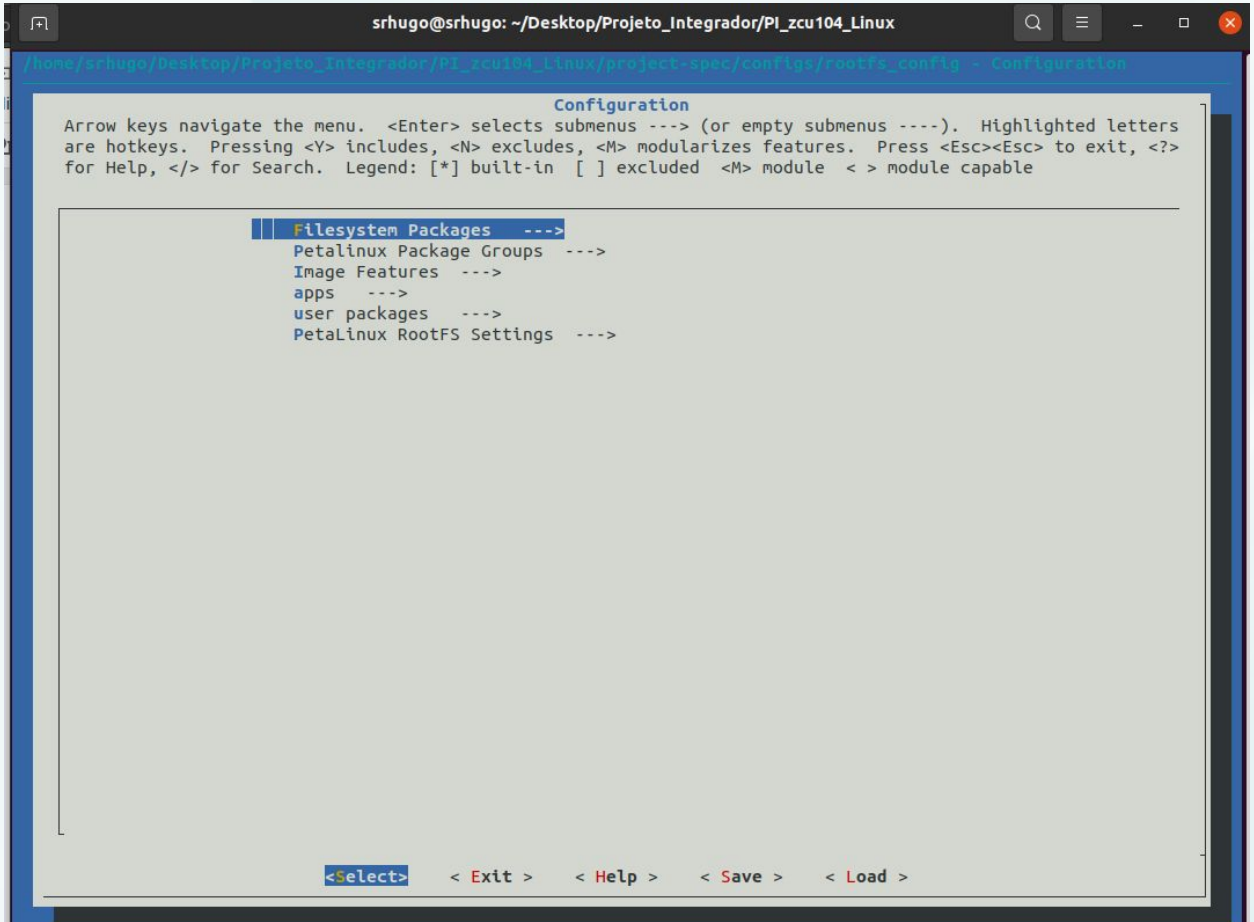
- Copy the previous topics and paste into user-rootfsconfig file



Save and close it

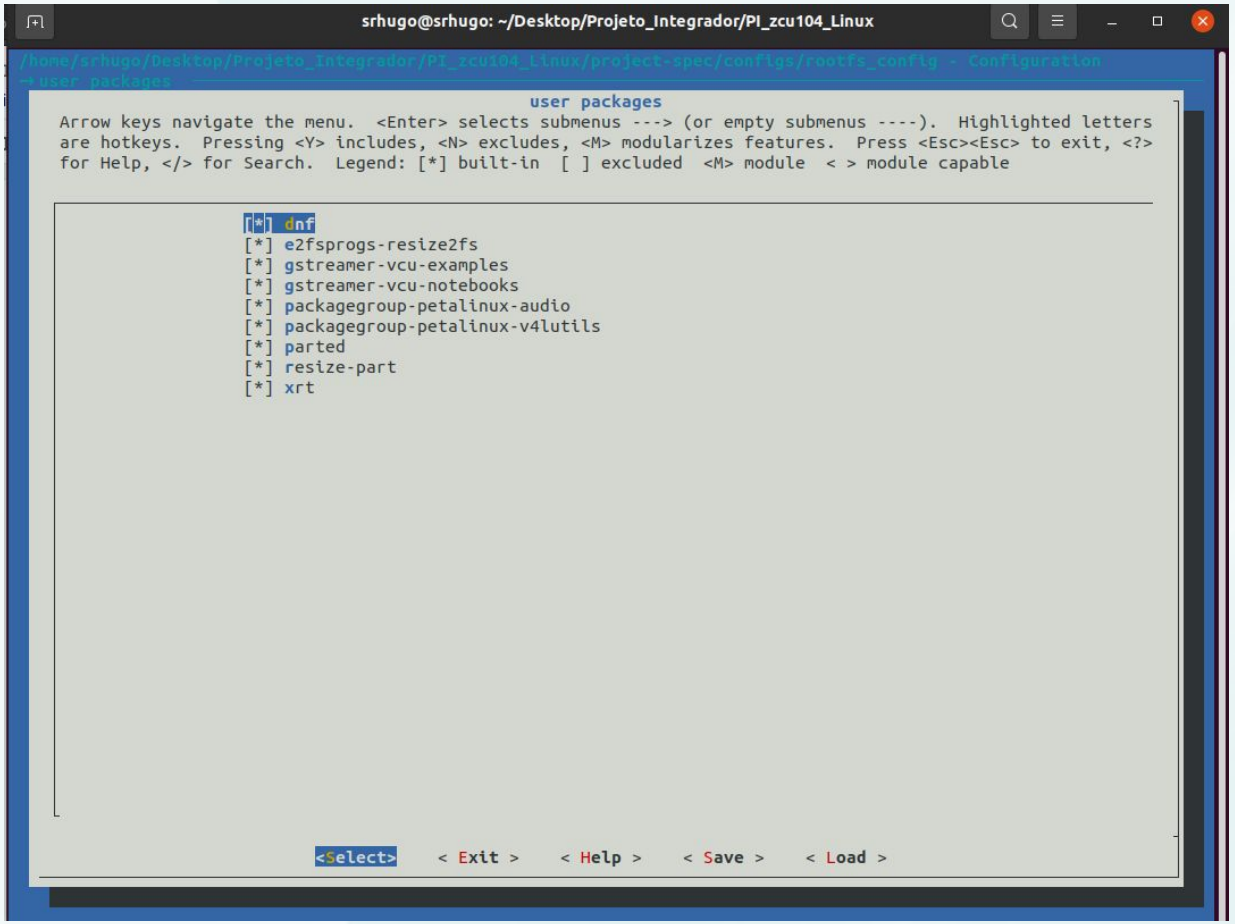
Creating the Linux Image

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



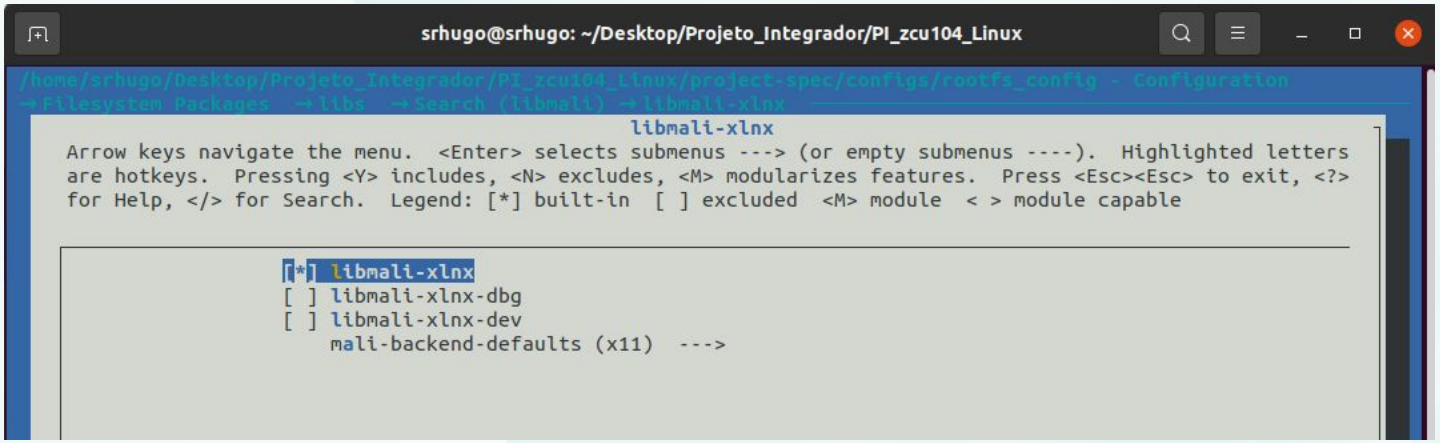
Creating the Linux Image

- In user packages activate all packages.

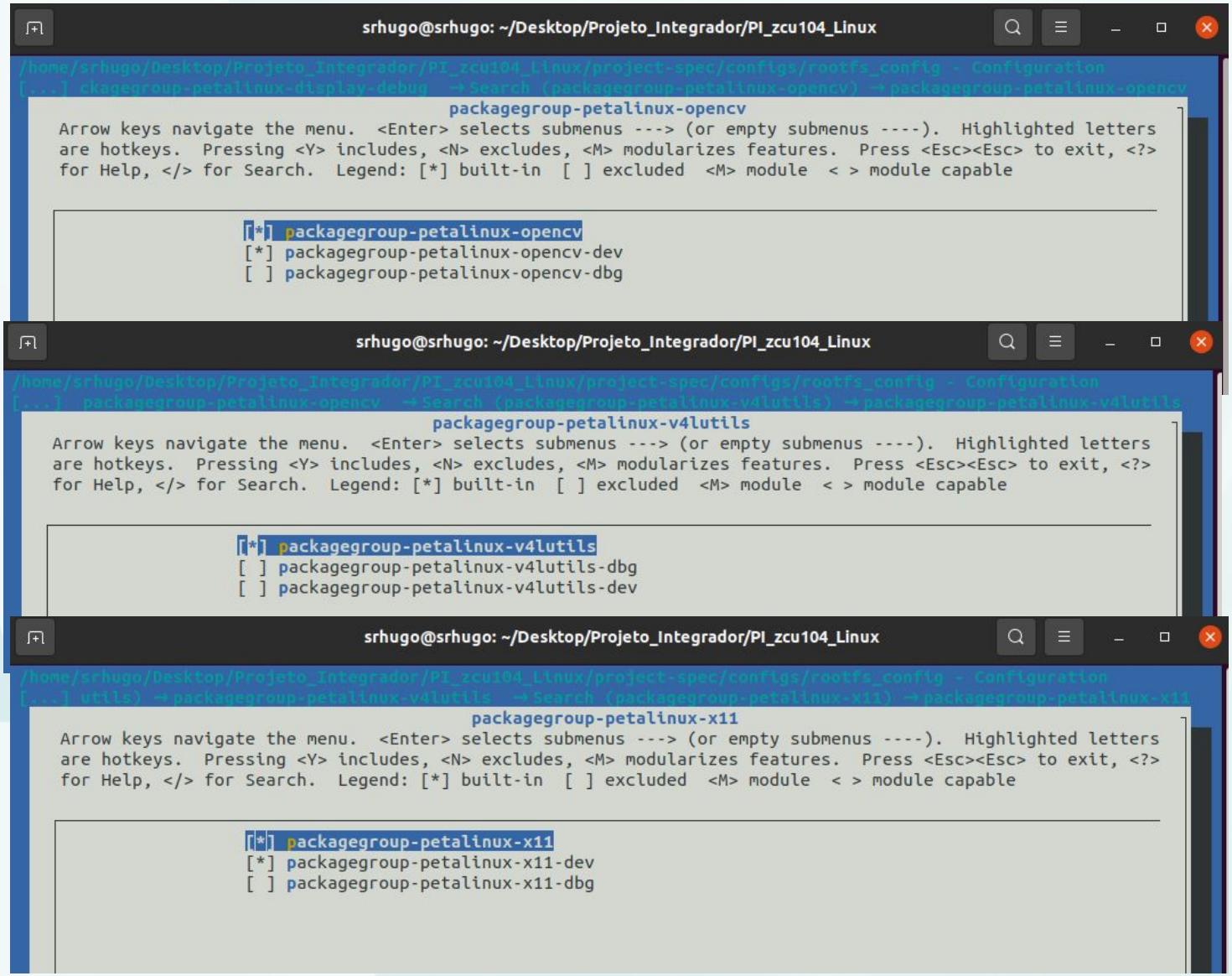


Creating the Linux Image

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

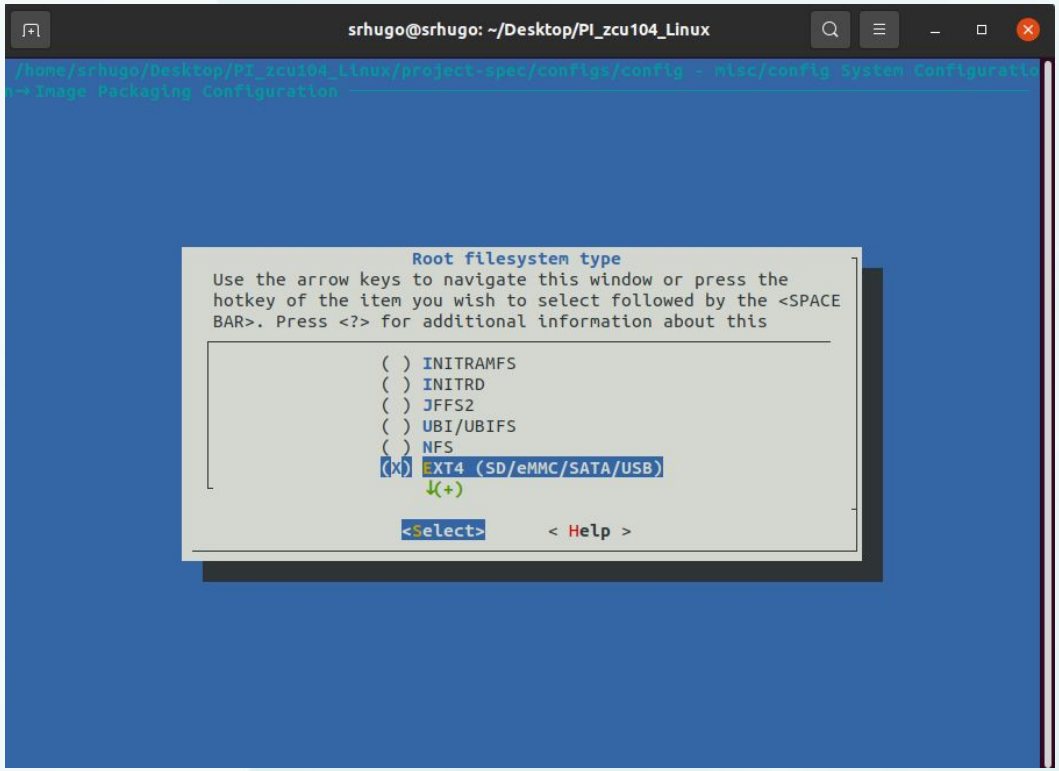


Creating the Linux Image



Creating the Linux Image

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

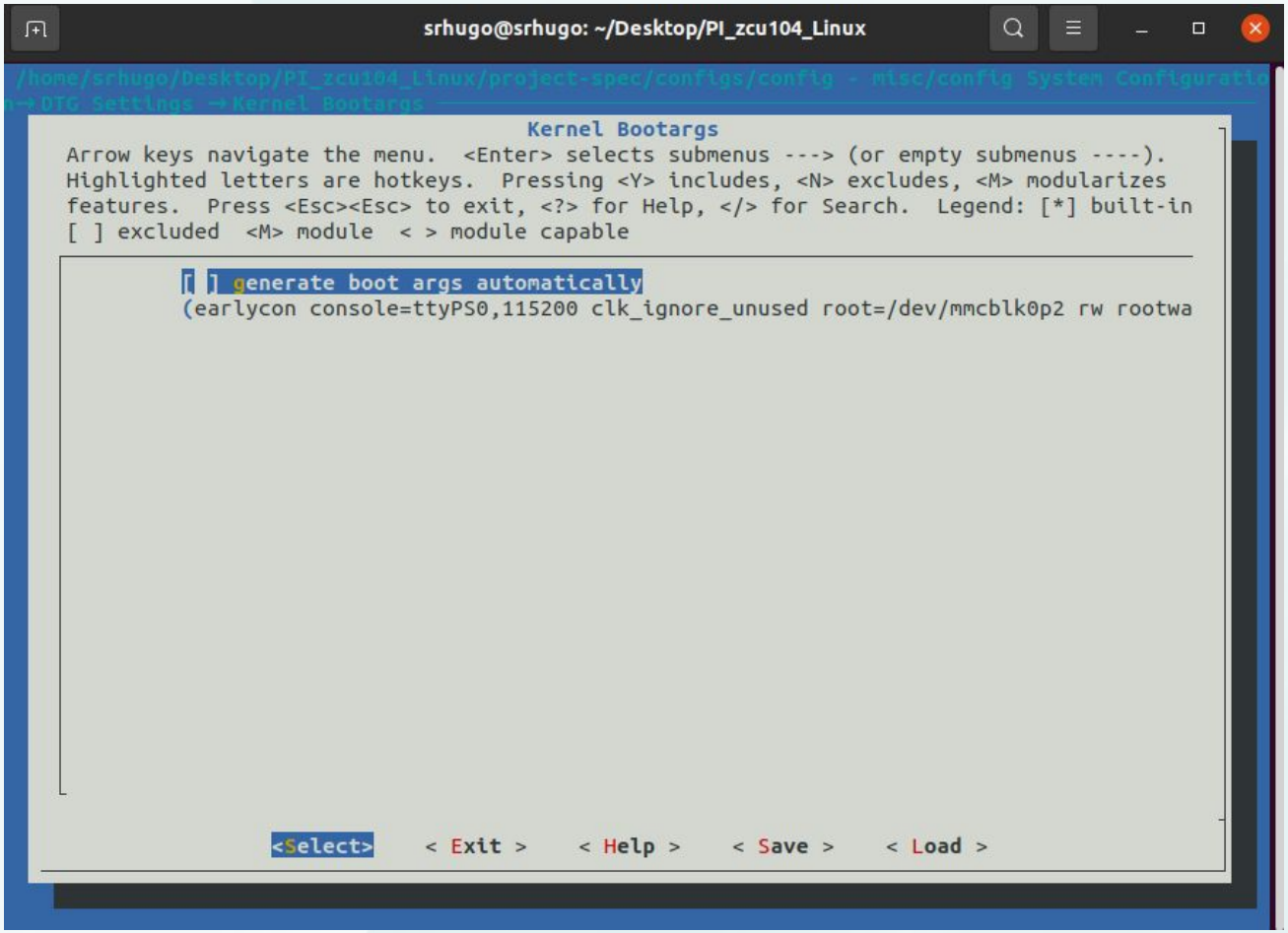


Creating the Linux Image

- 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/mmcblkop2 rw rootwait cma=512M

Creating the Linux Image

- You should get the this output



Creating the Linux Image

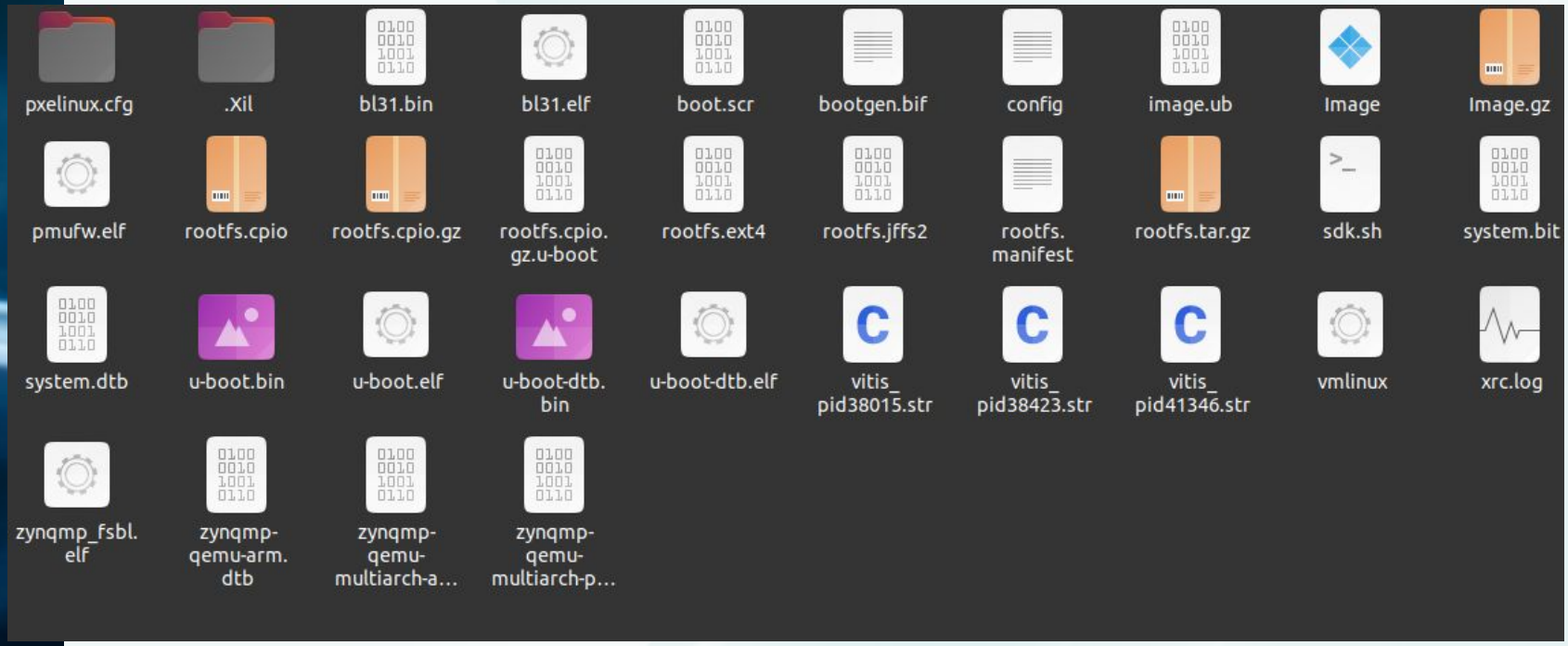
- Save all and exit
- Type petalinux-build and go take a coffee.

Creating the Linux Image

- 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 :)

Creating the Linux Image

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



Creating a Custom Vitis Platform

- 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.....................................
```

Creating a Custom Vitis Platform

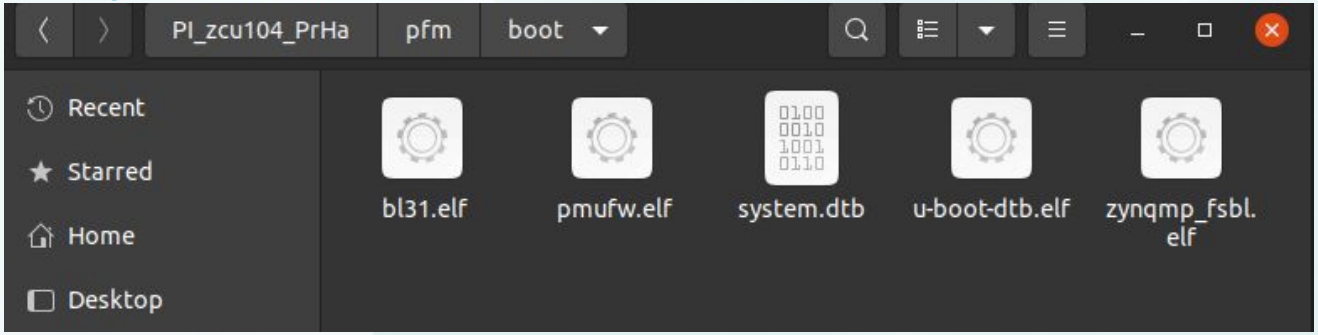
- In the end you should receive a successful message

```
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

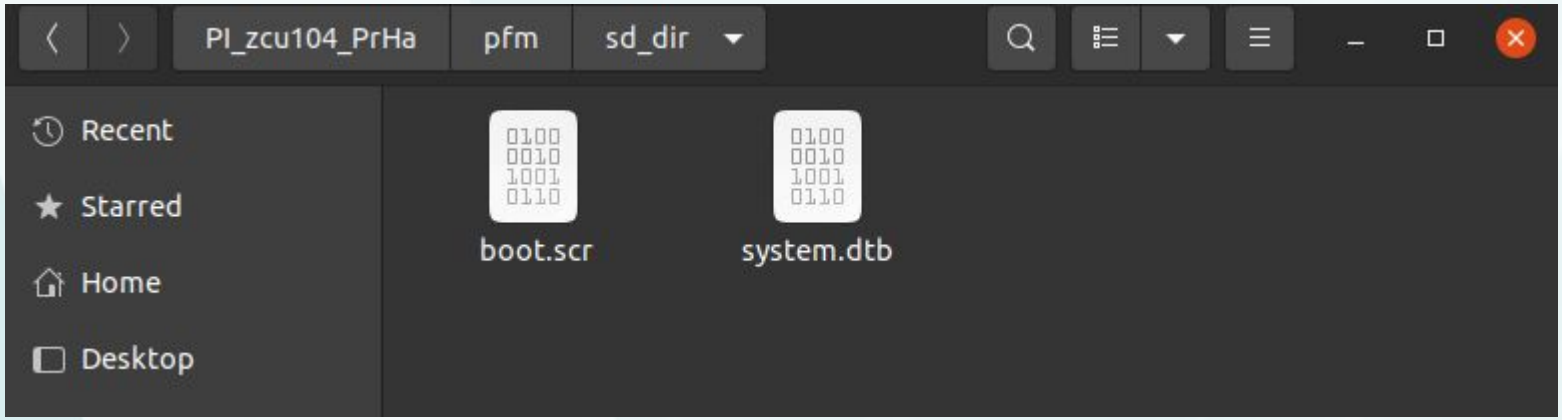
Creating a Custom Vitis Platform

- Go inside the pfm folder and create another 2 folder (boot and sd_dir)
  - Note that this names are standart. 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



Creating a Custom Vitis Platform

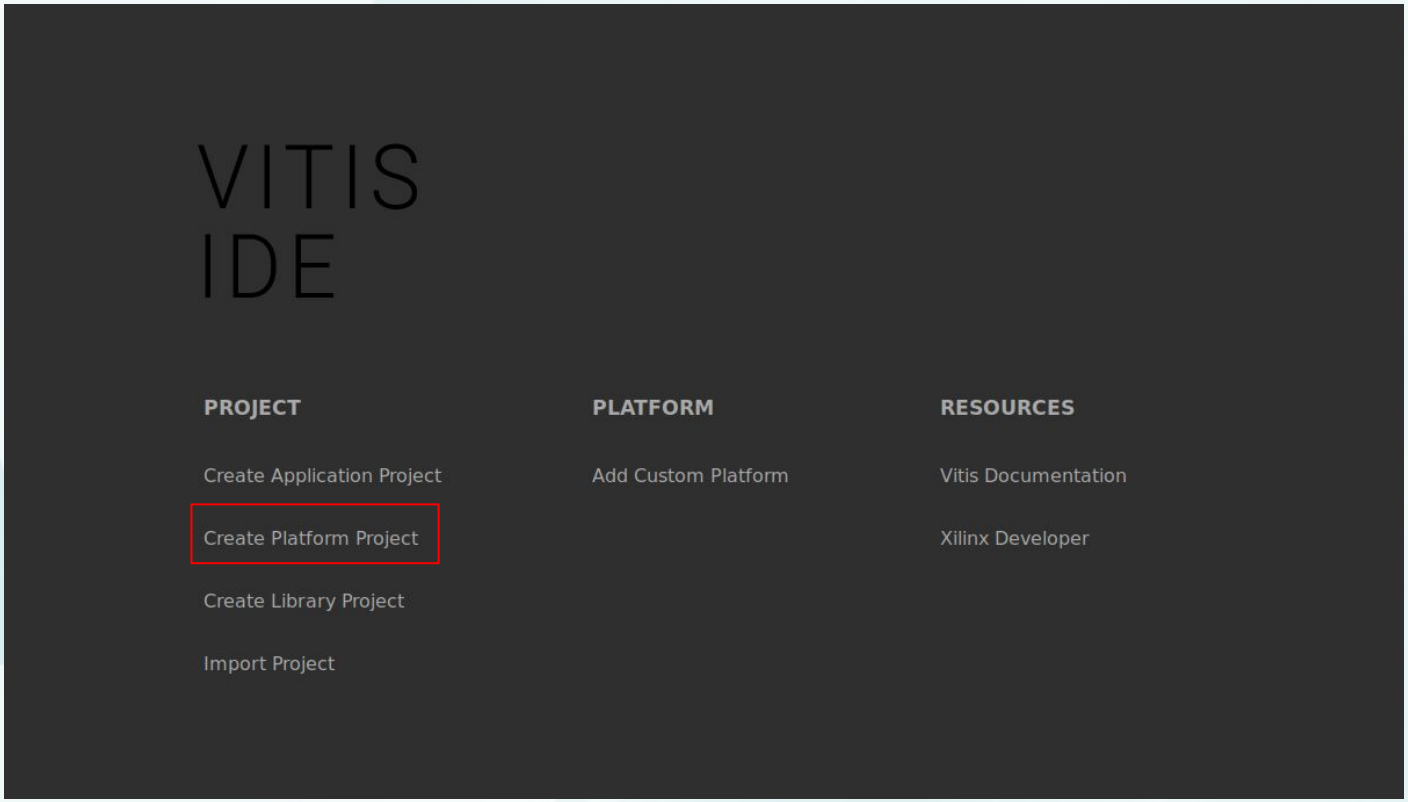
- Inside the images/linux copy the following files to the sd_dir folder
  - boot.src
  - system.dtb



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

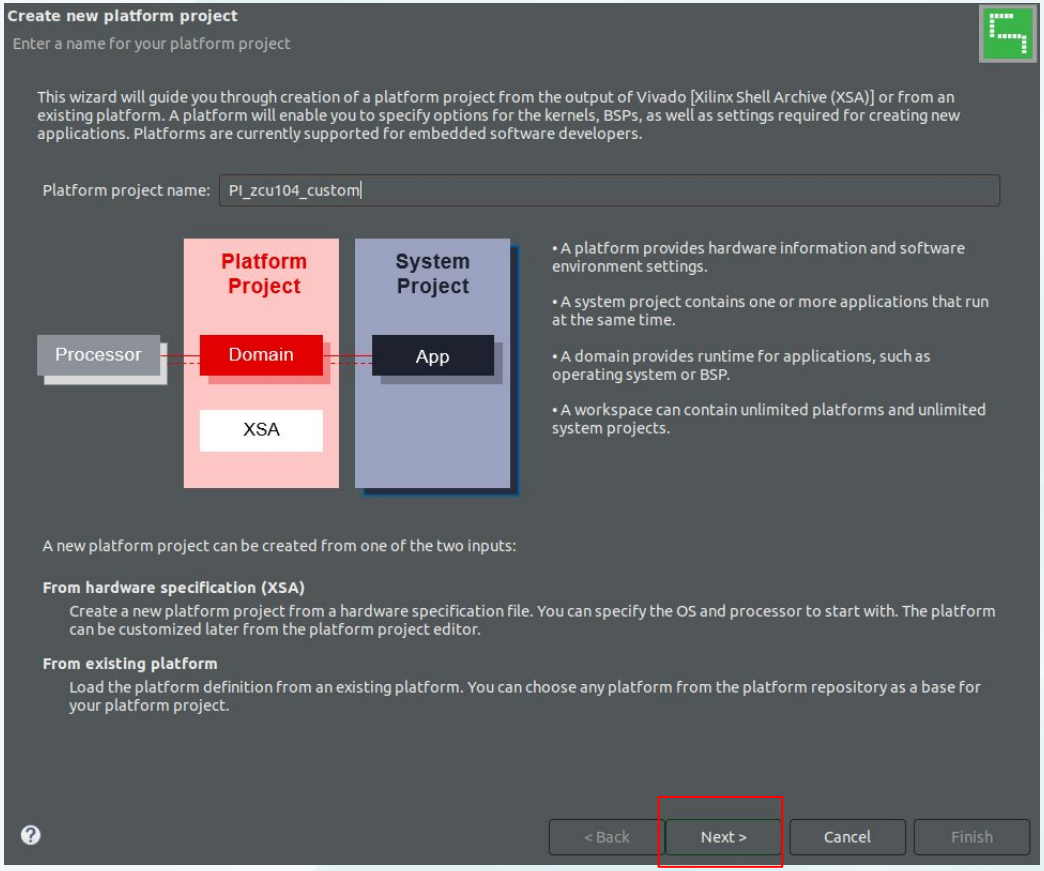
Creating a Custom Vitis Platform

- Create a Vitis Platform



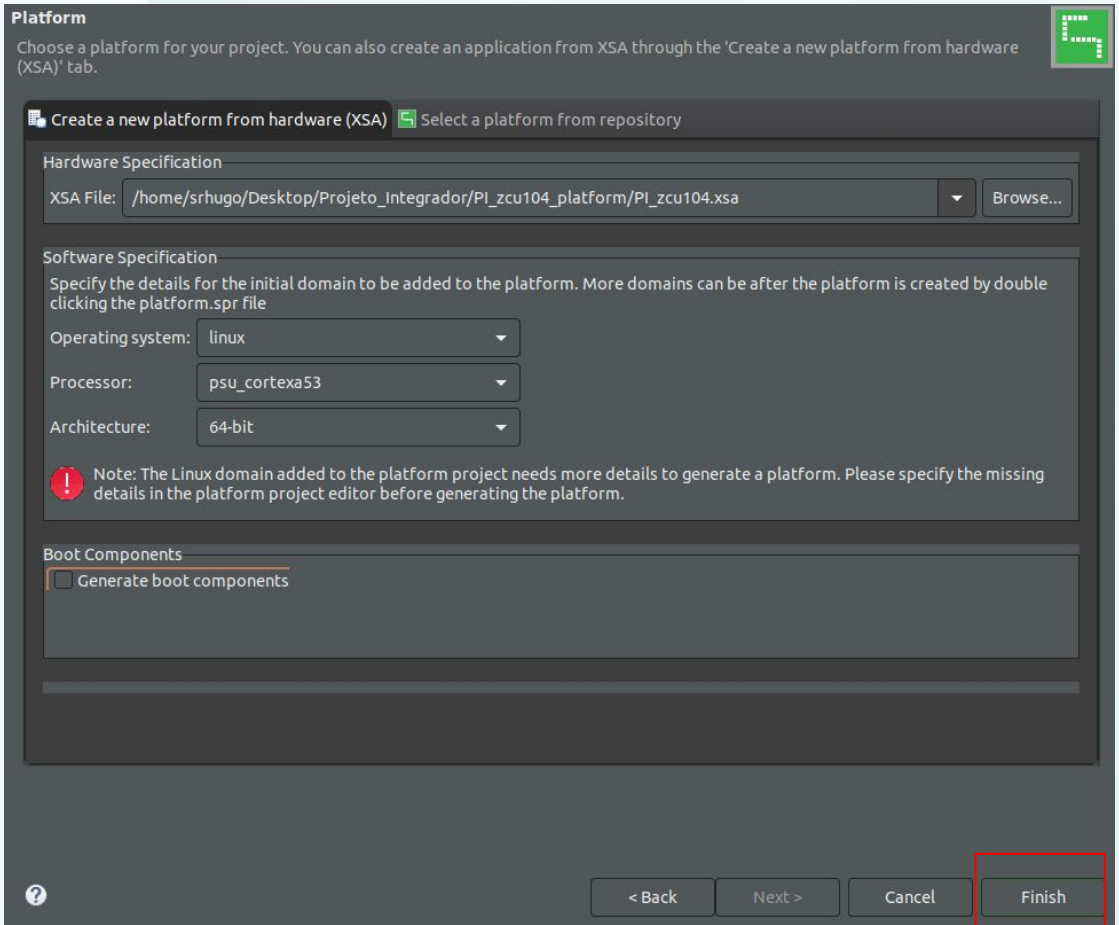
Creating a Custom Vitis Platform

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



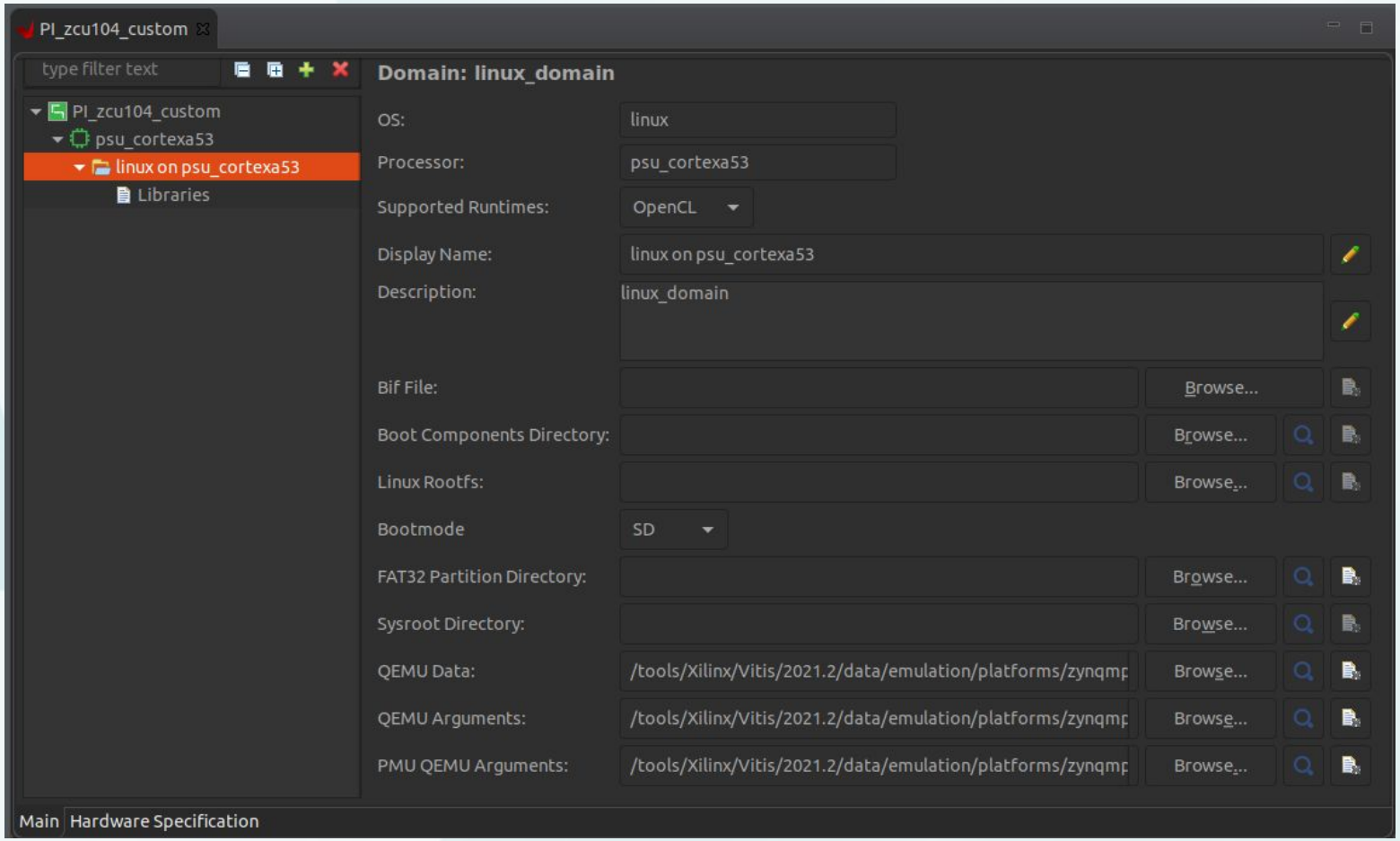
Creating a Custom Vitis Platform

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



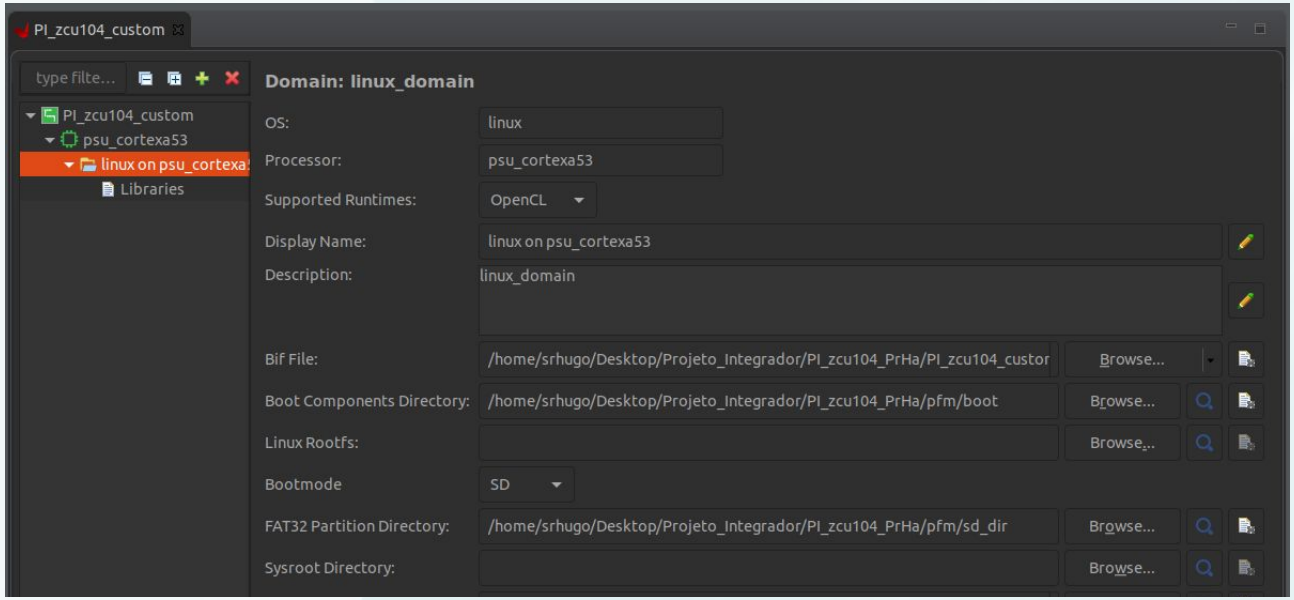
Creating a Custom Vitis Platform

- After finished, go to **linux on psu_cortexa53** tab



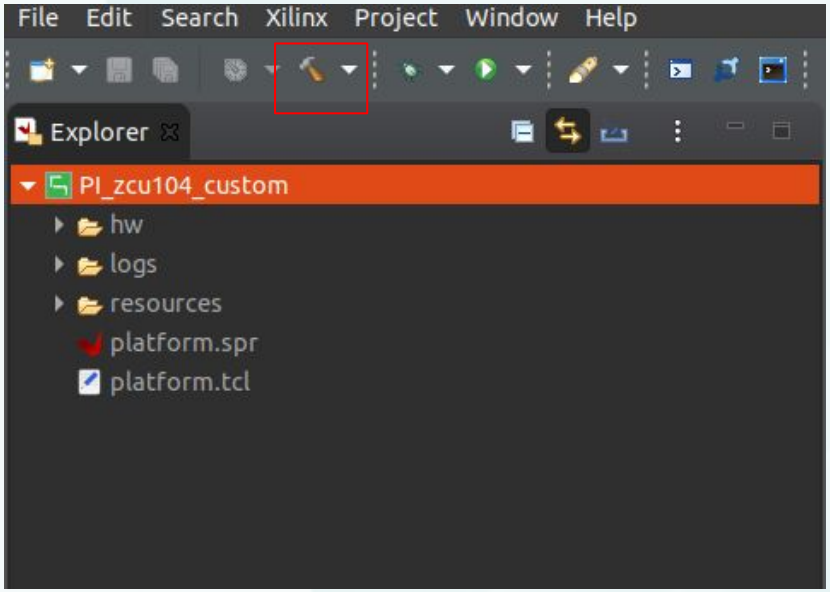
Creating a Custom Vitis Platform

- 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

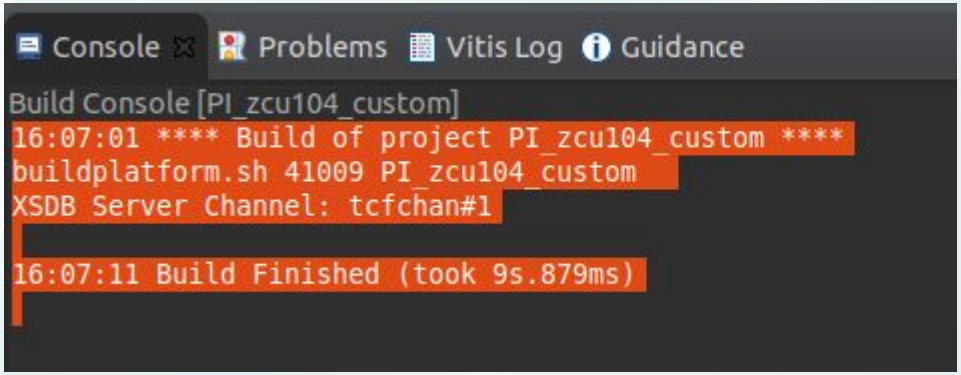


Creating a Custom Vitis Platform

- Select the platform project and build it

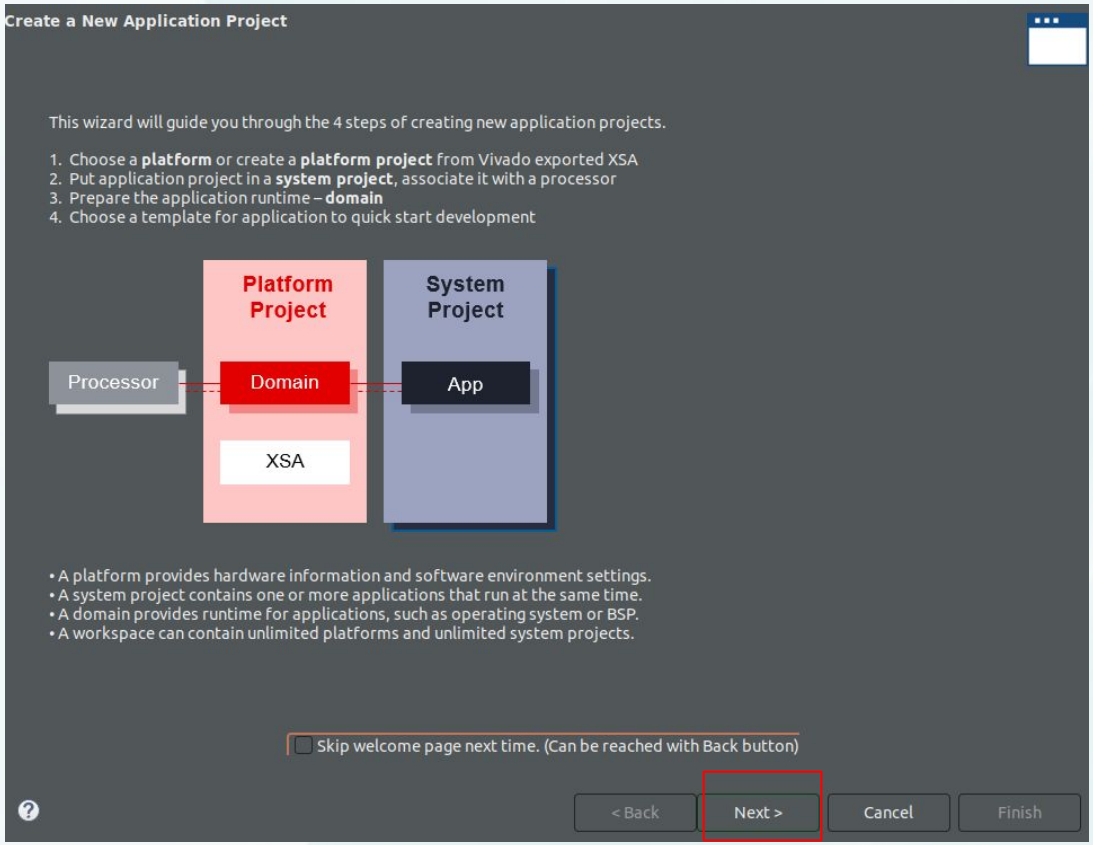


- Expected Output:



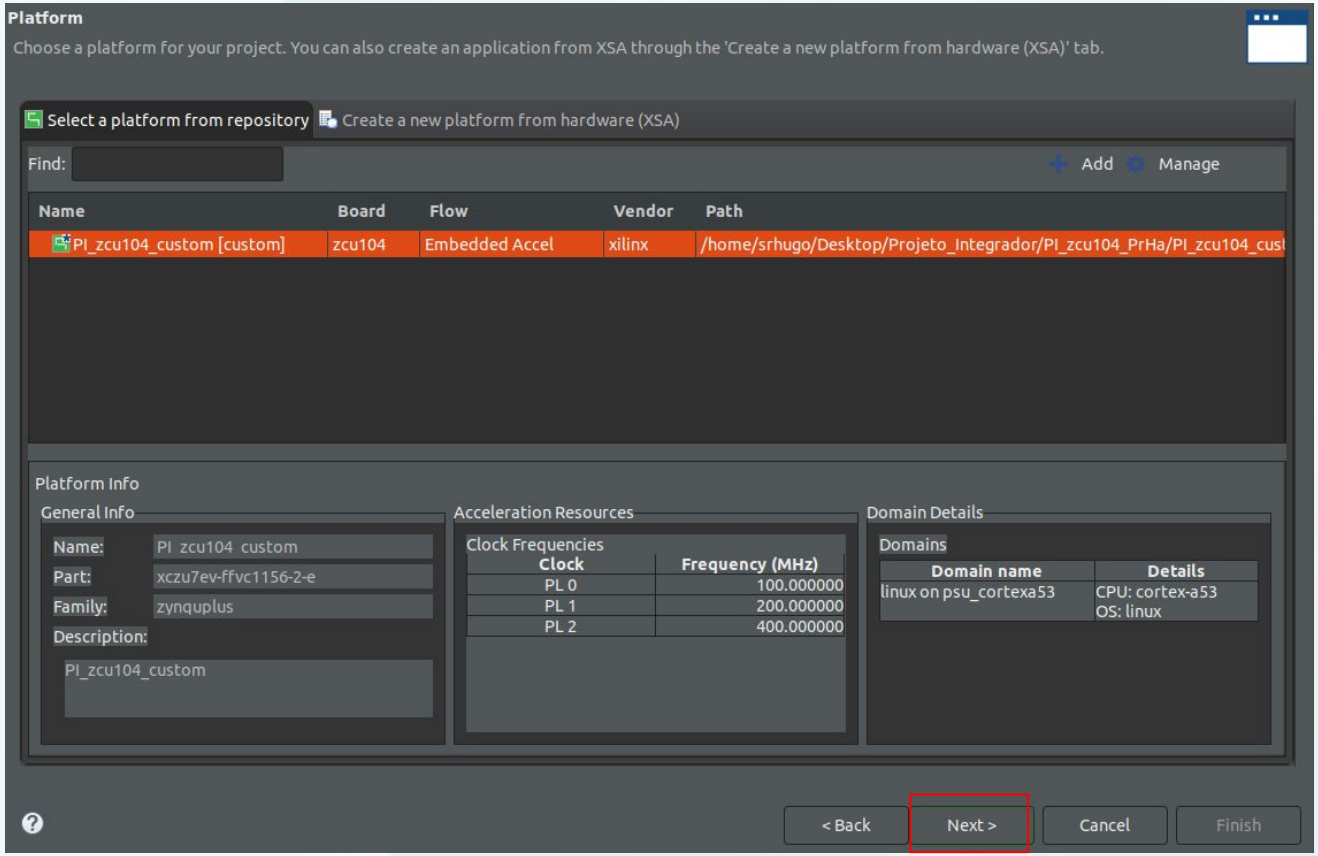
Test an Hello World

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



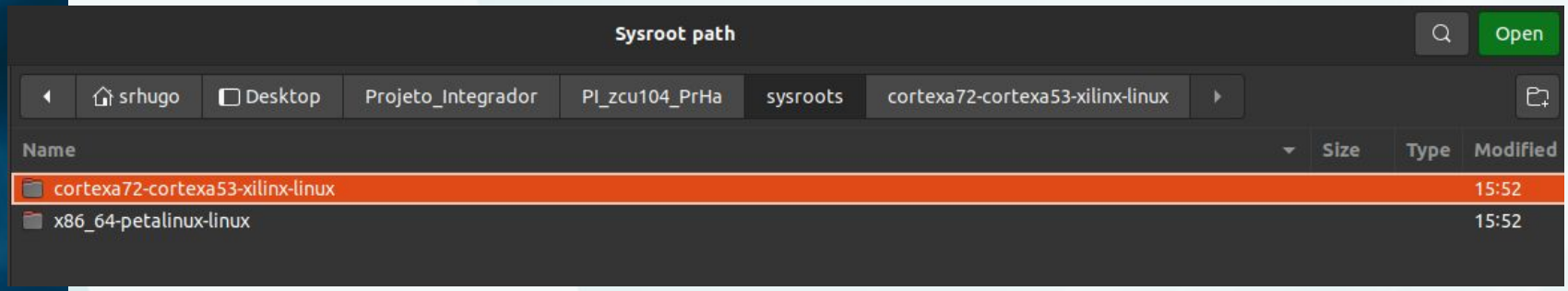
Test an Hello World

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



Test an Hello World

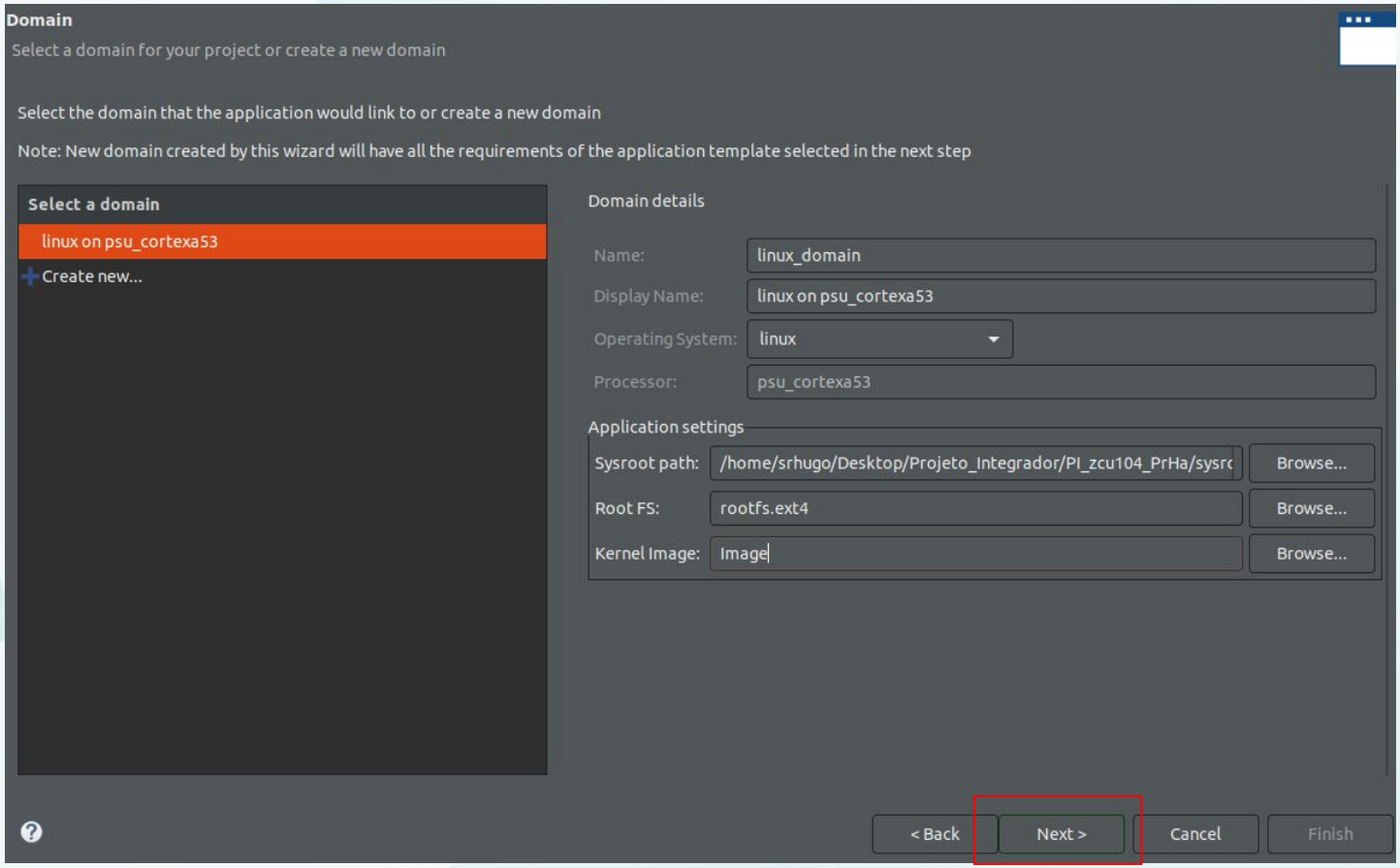
- Select the **linux on psu_cortexa53** domain
- Then set the Sys_root path to the following directory



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

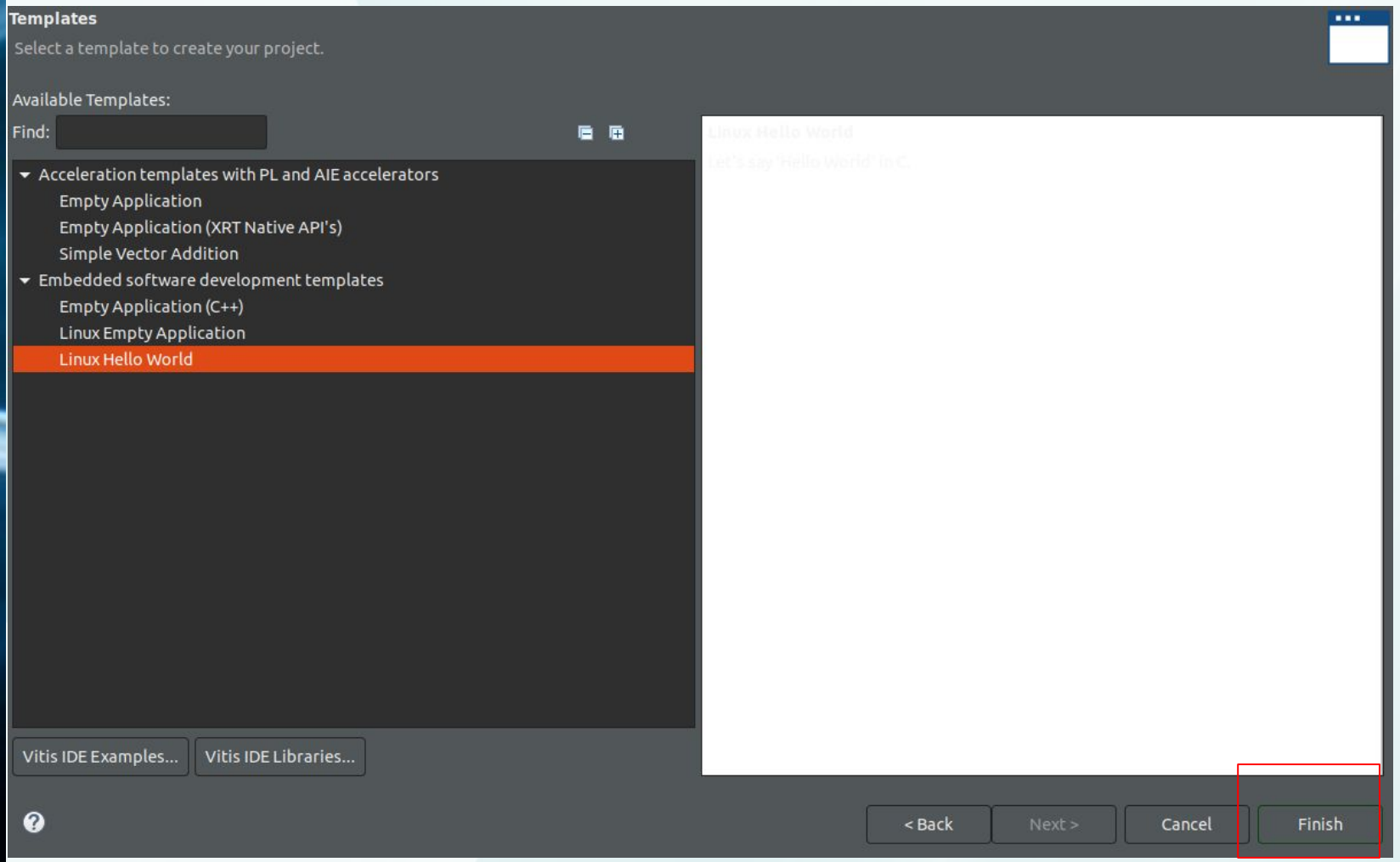
Test an Hello World

- You should have the following domain screen



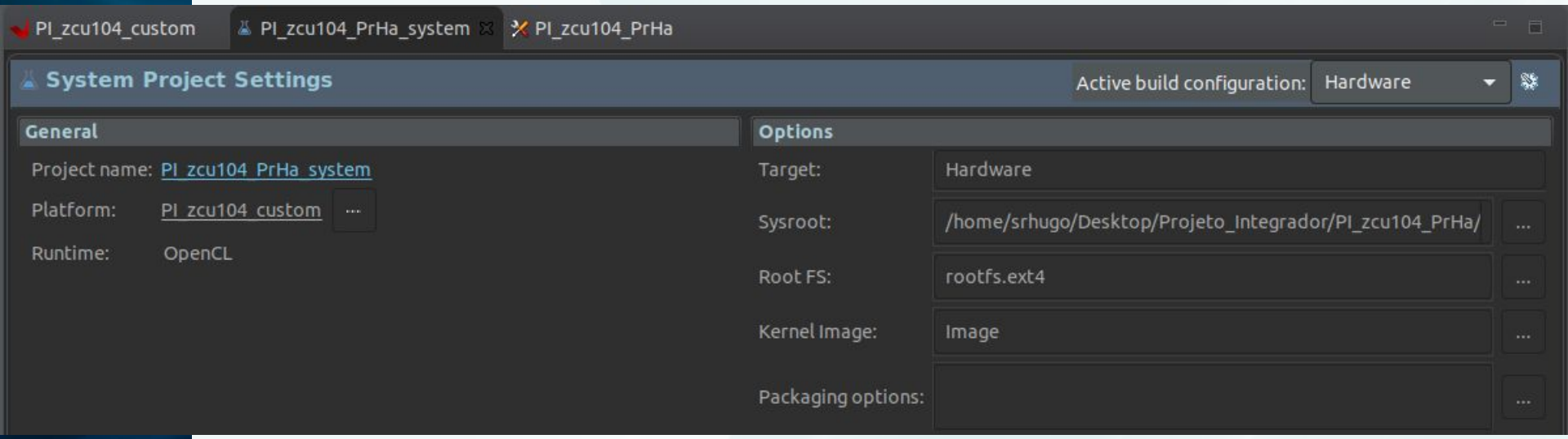
Test an Hello World

- Select the Hello World Template and hit Finish



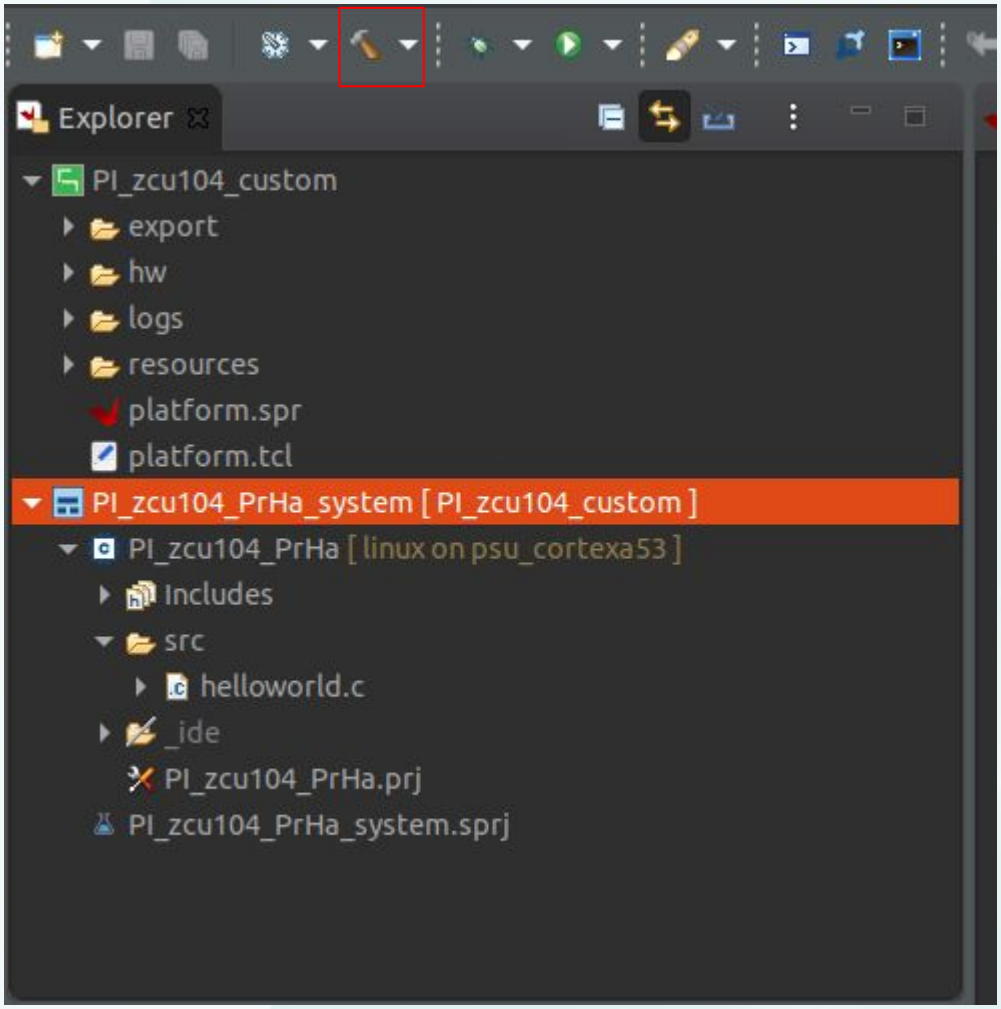
Test an Hello World

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



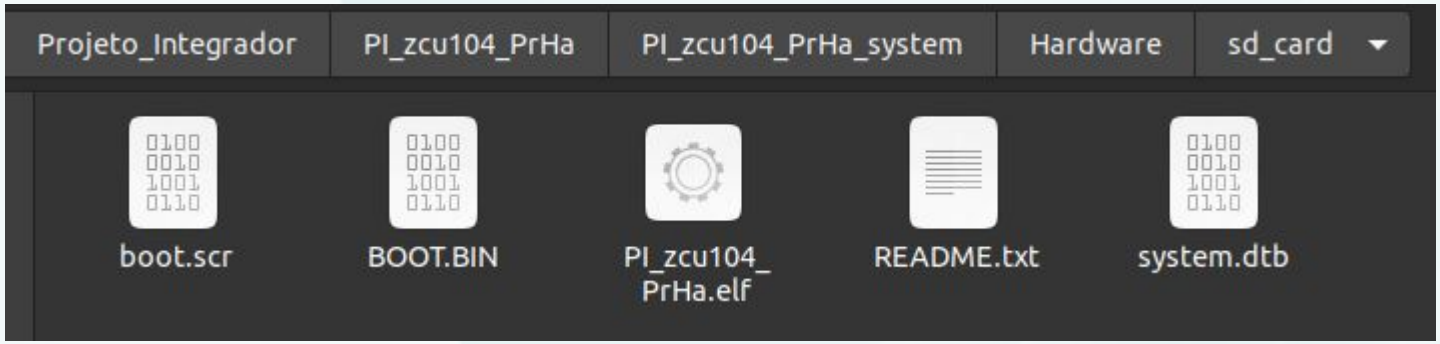
Test an Hello World

- Select the Application project and build it



Test an Hello World

- If build was successful, you must have the following files in the <project_name>_system/Hardware/sd_card folder.

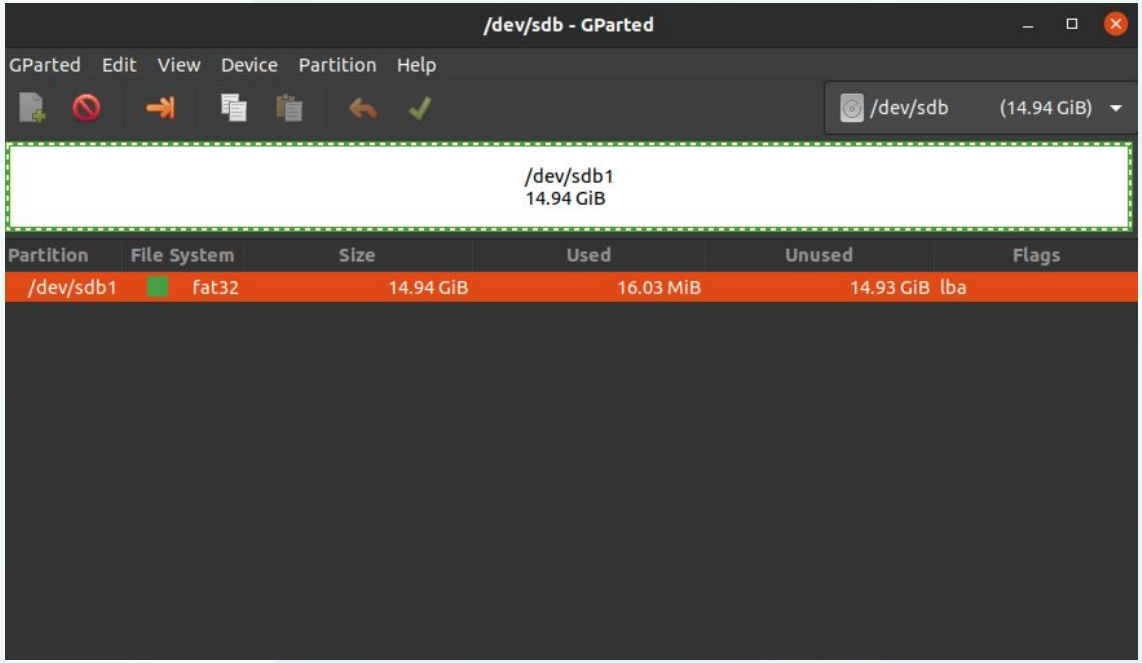


- Now we must prepare the SD card to receive the linux image

Test an Hello World

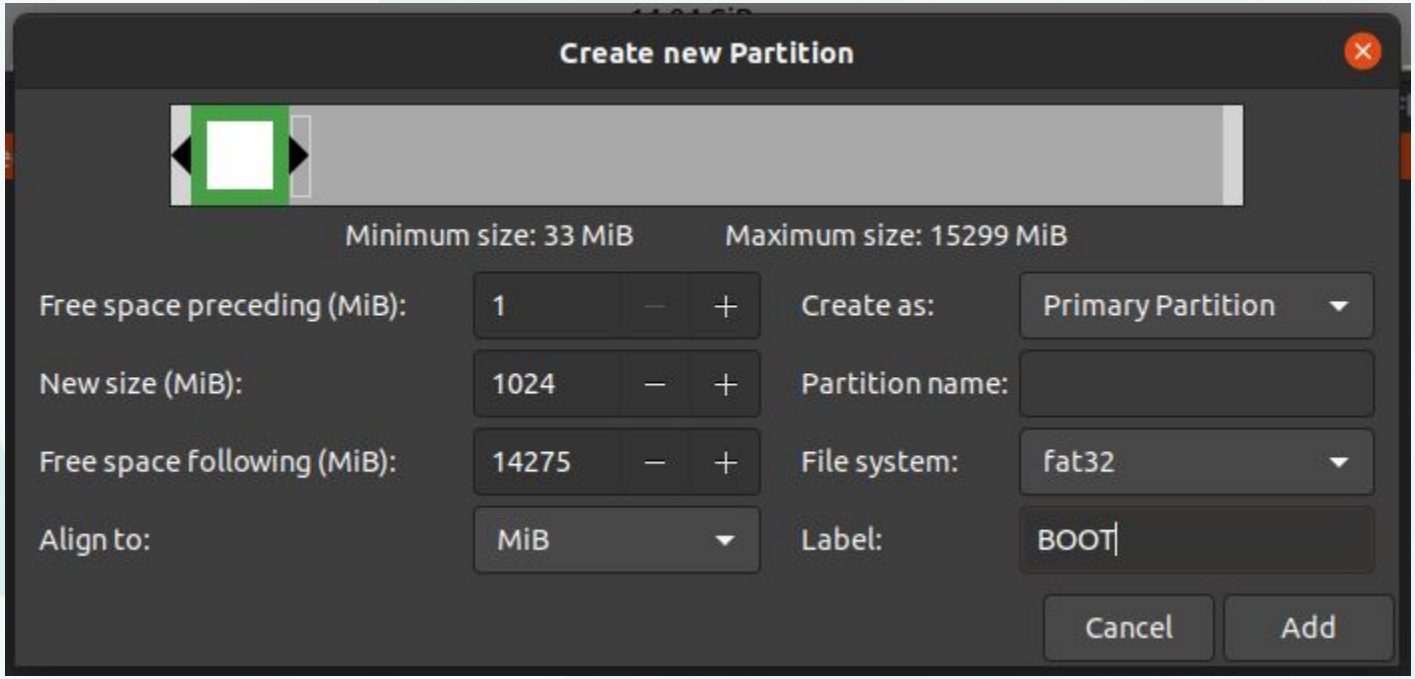
- Prepare the SD Card to receive the linux
  - In this tutorial will be used the Gparted Tool but there other ways to make the partitions

- Connect the SD card and open Gparted
  - Select the SD card



Test an Hello World

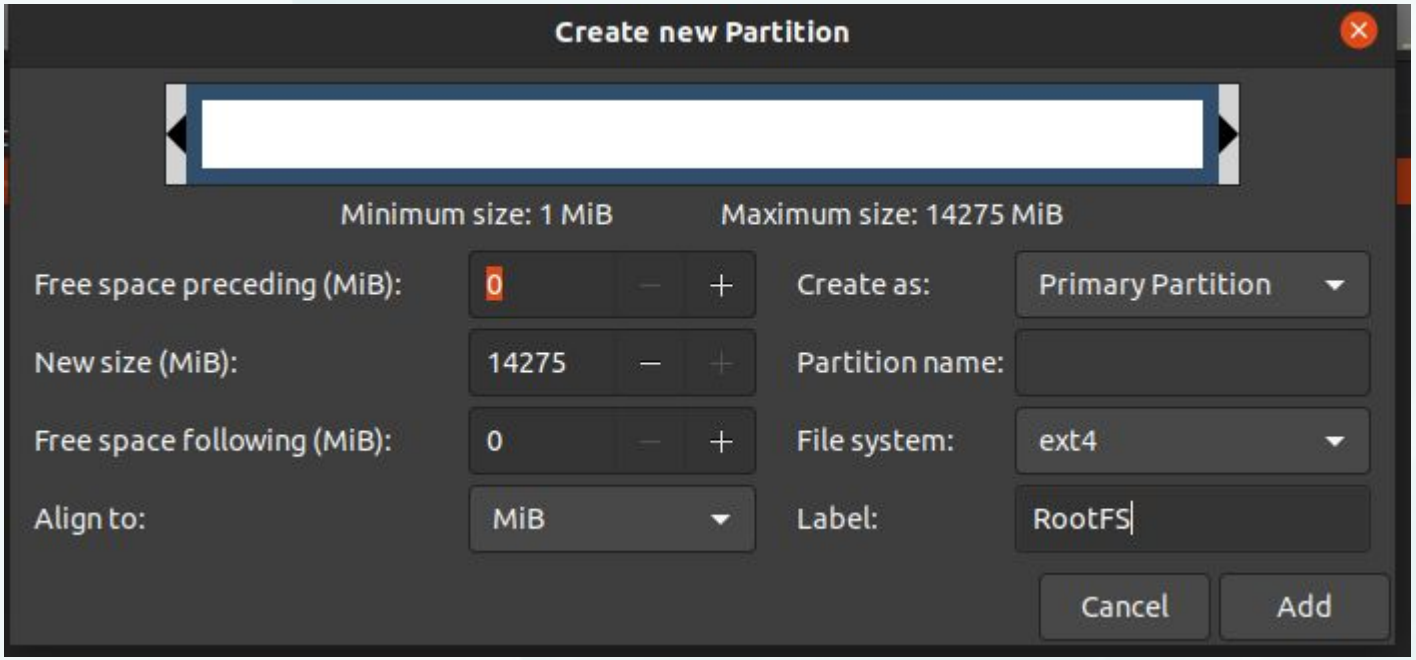
- Delete all existing partitions
- Create the BOOT partition with the following settings



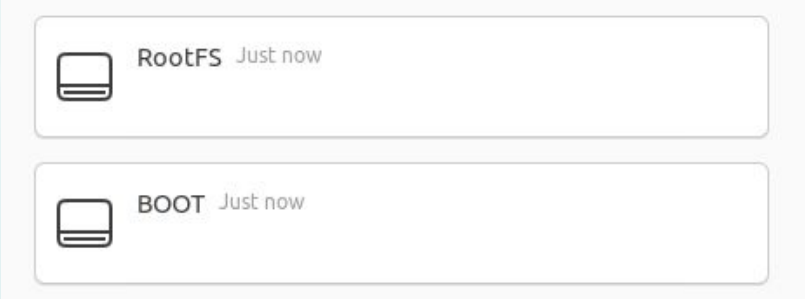
- If space is an issue, you can reduce the BOOT partition but be careful

Test an Hello World

- Create the RootFS partition with the following settings



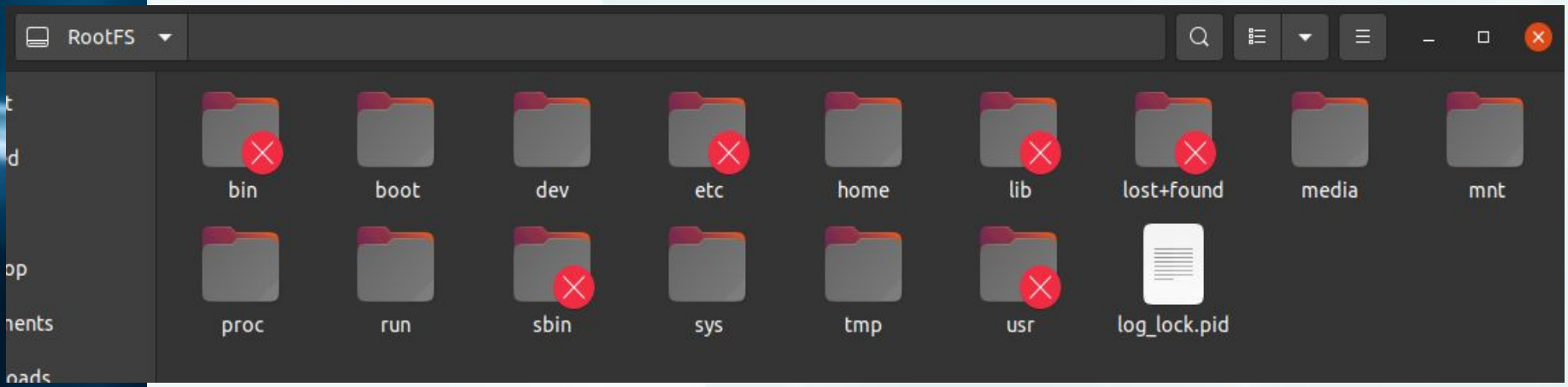
- Save the changes and verify if it was successful



Test an Hello World

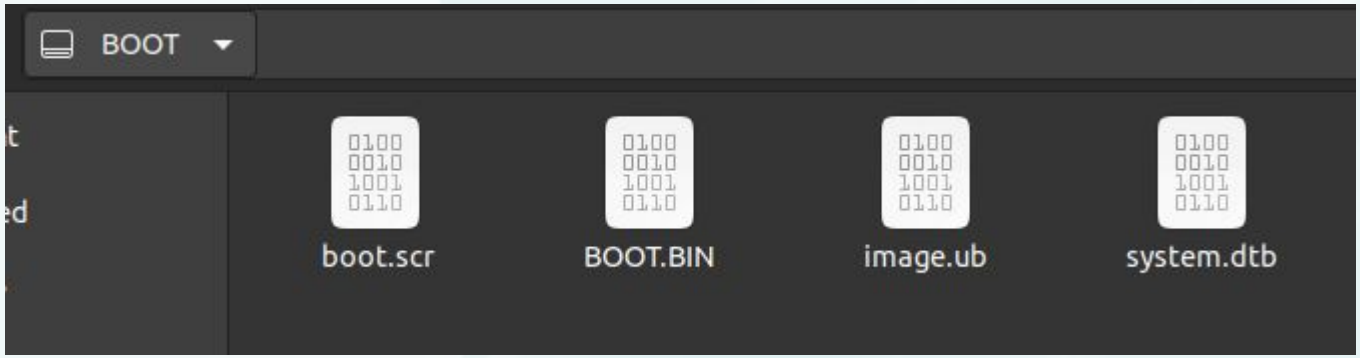
- To the Root FS, go to images/linux folder and type the following command
  - sudo tar -xvf rootfs.tar.gz -C /media/<yourName>/RootFS

- RootFS partition after this command



Test an Hello World

- 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



Test an Hello World

- 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

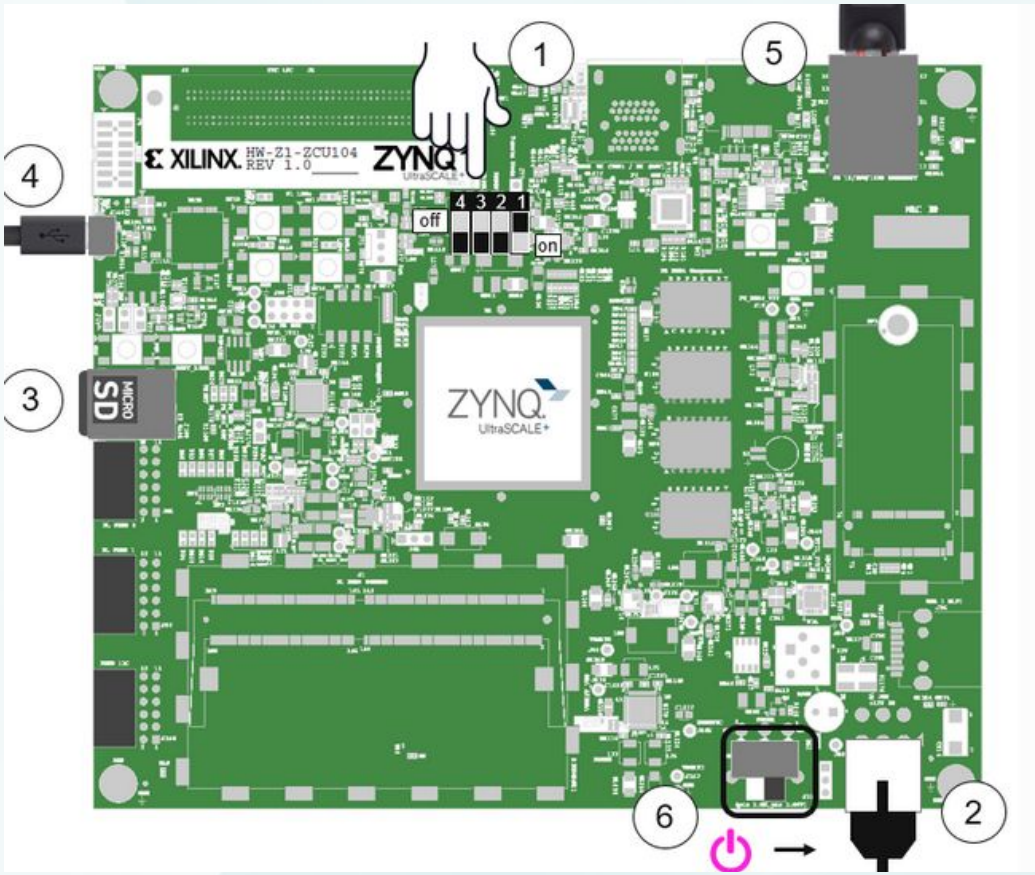
- We will going to do both! Choose one and keep it.

- 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/

Test an Hello World

- 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



Test an Hello World

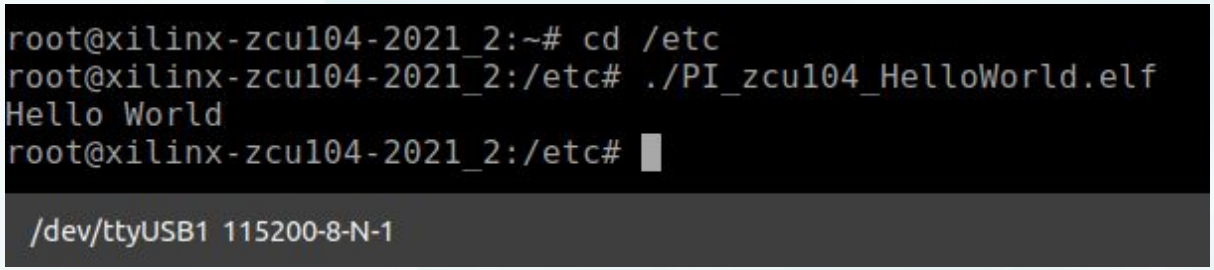
- Connect to the board by a SSH or UART connection
  - Here will be used the UART one
    - If you don't have any serial port terminal, GTKTerm is a good choice

- After botting, 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
X11                       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
```

Test an Hello World

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

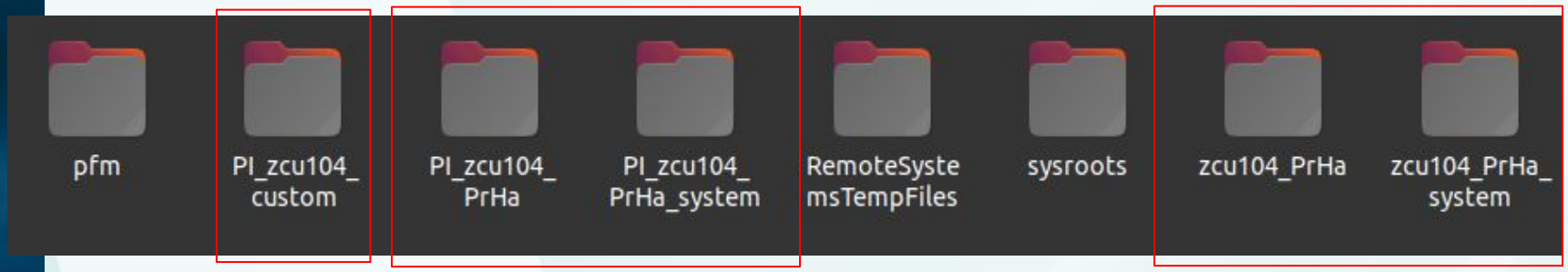


Test PrHa Code

- If the HelloWorld isn't working, check all the previous steps and make sure you follow then 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**.

Test PrHa Code

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

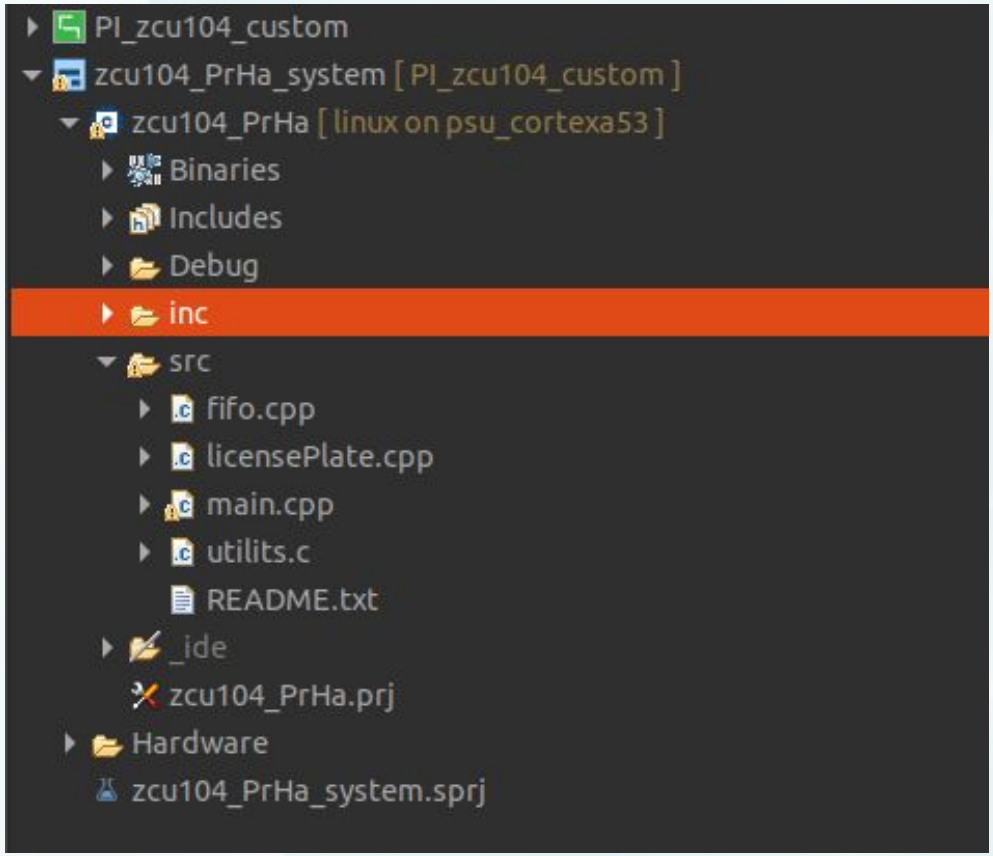


Platform HelloWorld Application PrHa Application

- Add the inc and src folders to zcu104_PrHa folder

Test PrHa Code

- Your Explorer window should be like this



Test PrHa Code

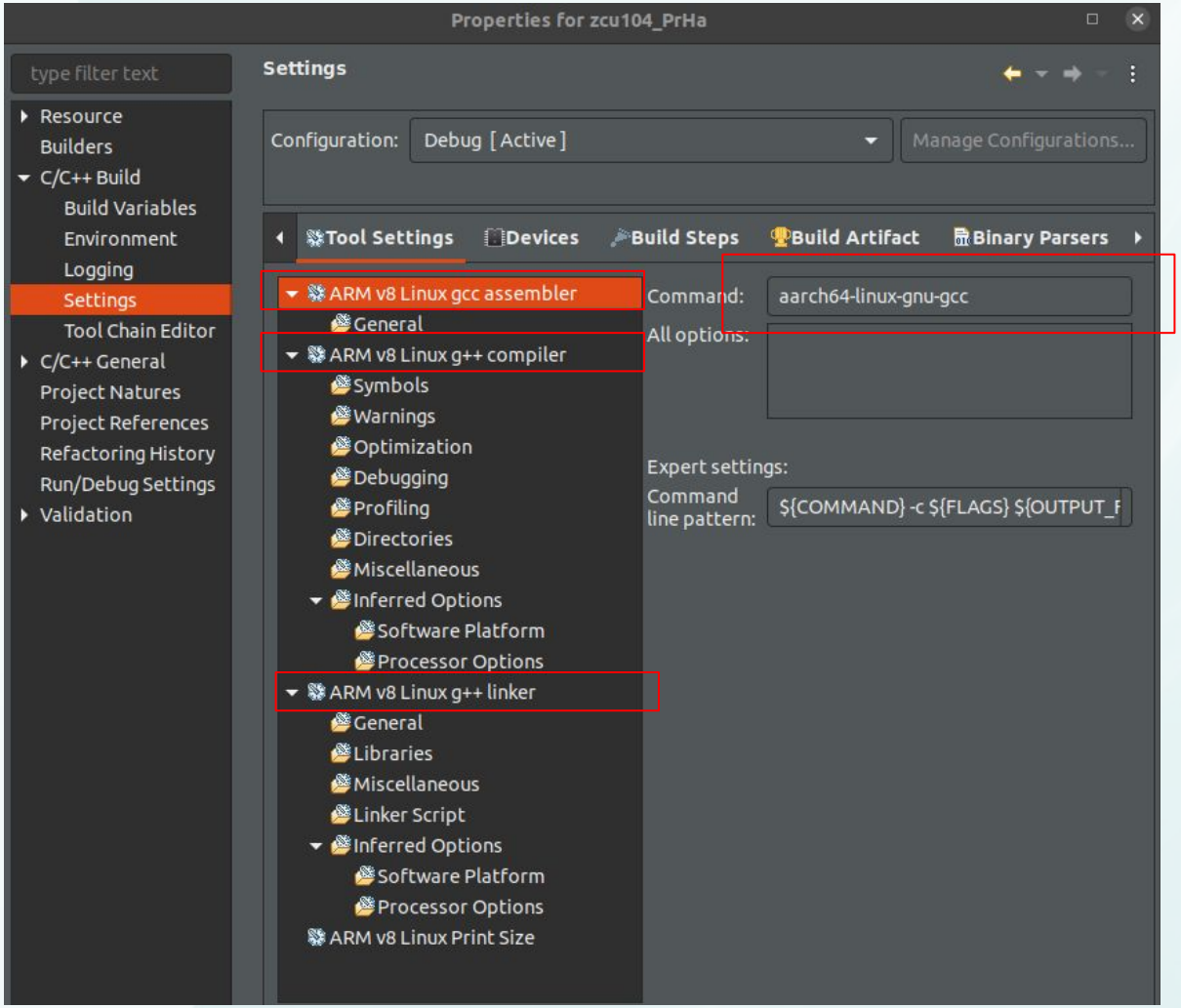
- Before build there's some changes you must do

- 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

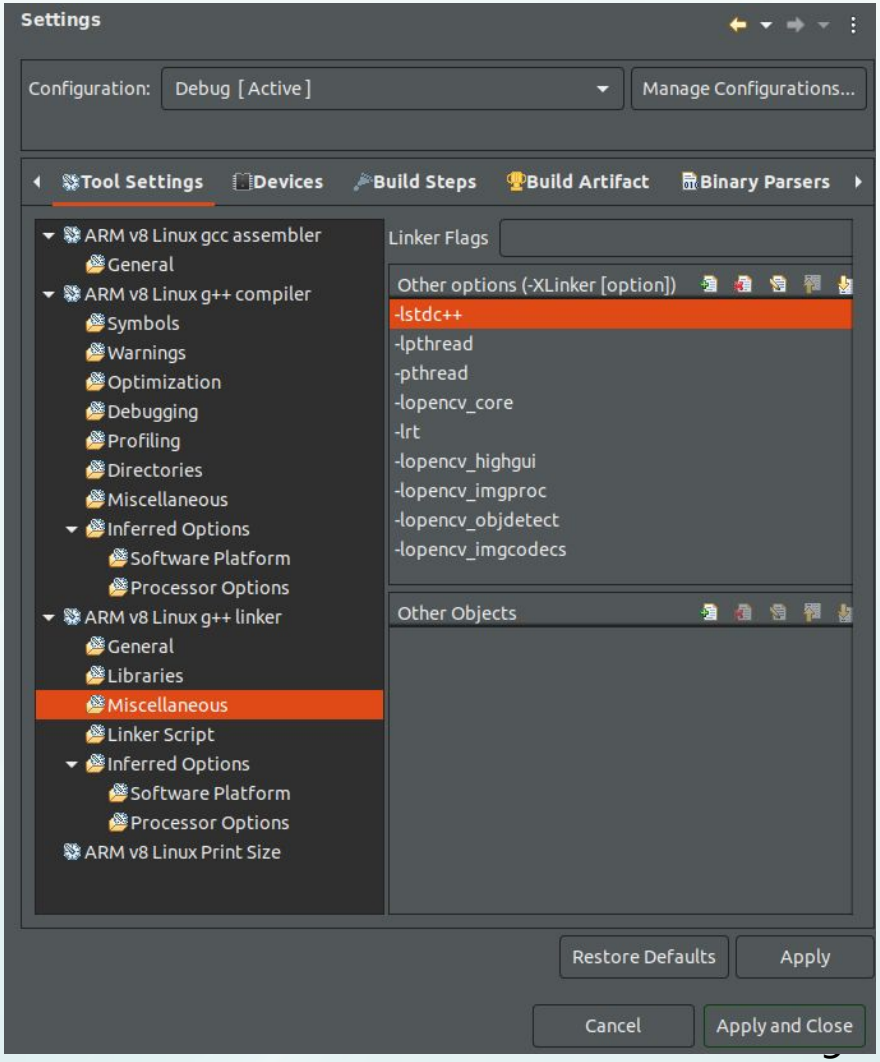
Test PrHa Code

- Change g++ to gcc in this 3 tabs



Test PrHa Code

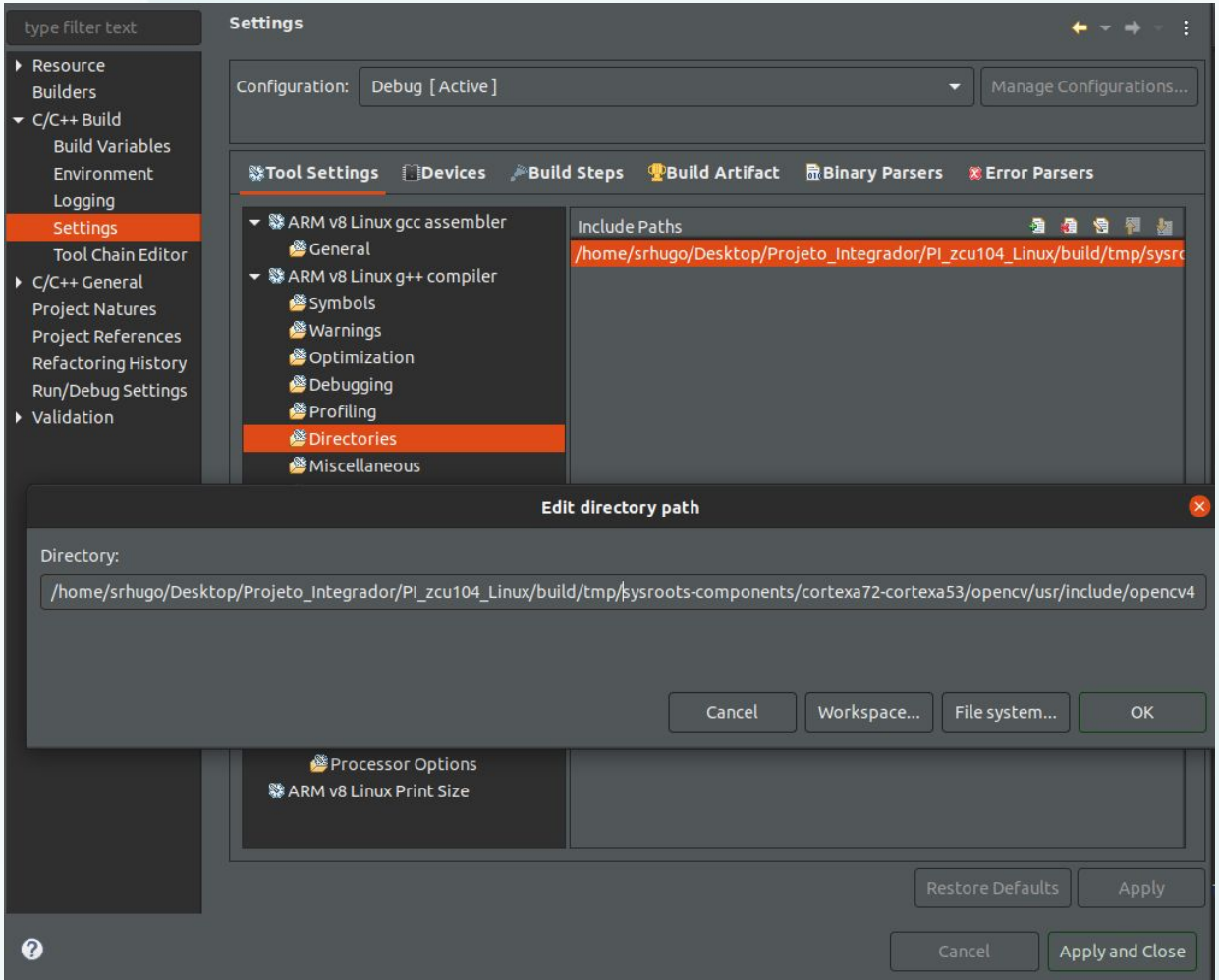
- After that, we must add the necessary flags to the linker
  - -lstdc++
  - -lpthread
  - -pthread
  - -lrt
  - -lopencv_core
  - -lopencv_highgui
  - -lopencv_imgproc
  - -lopencv_objdetect
  - -lopencv_imgcodecs



- lstdc++ for c++ code
- lpthread for threads
- lopencv for opencv

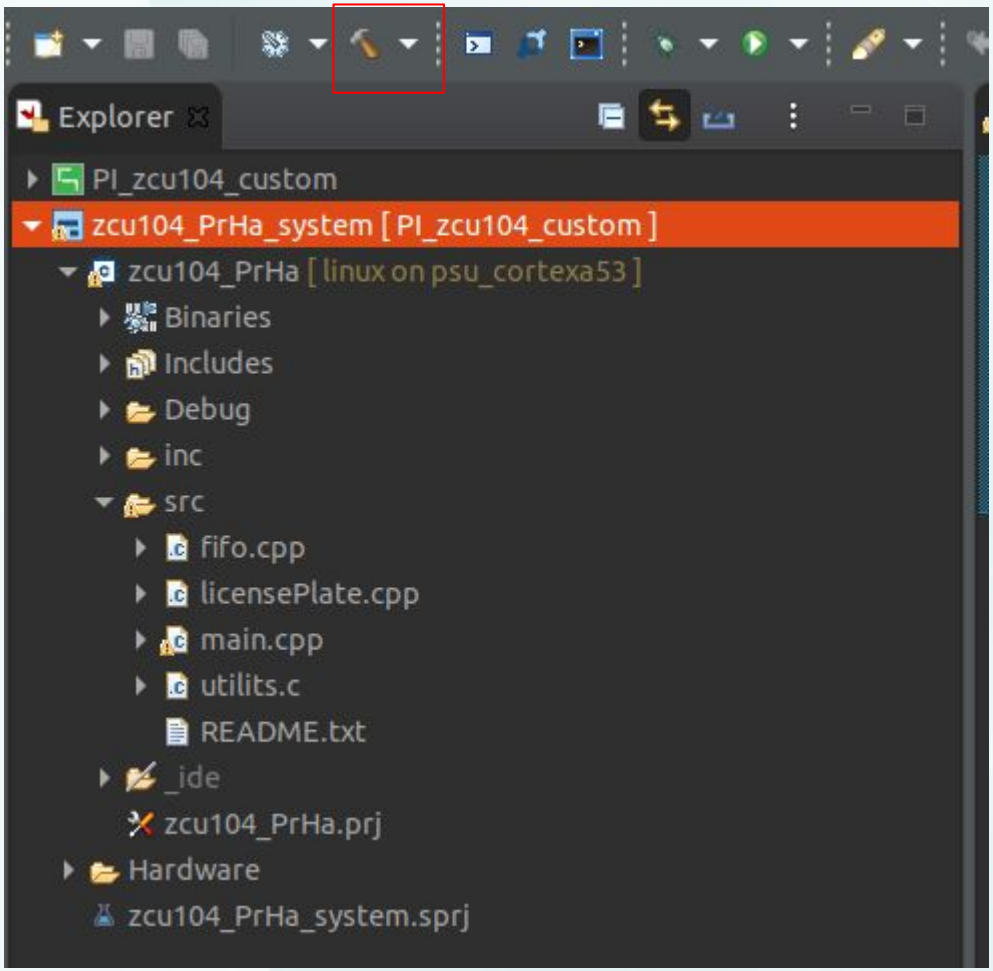
Test PrHa Code

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



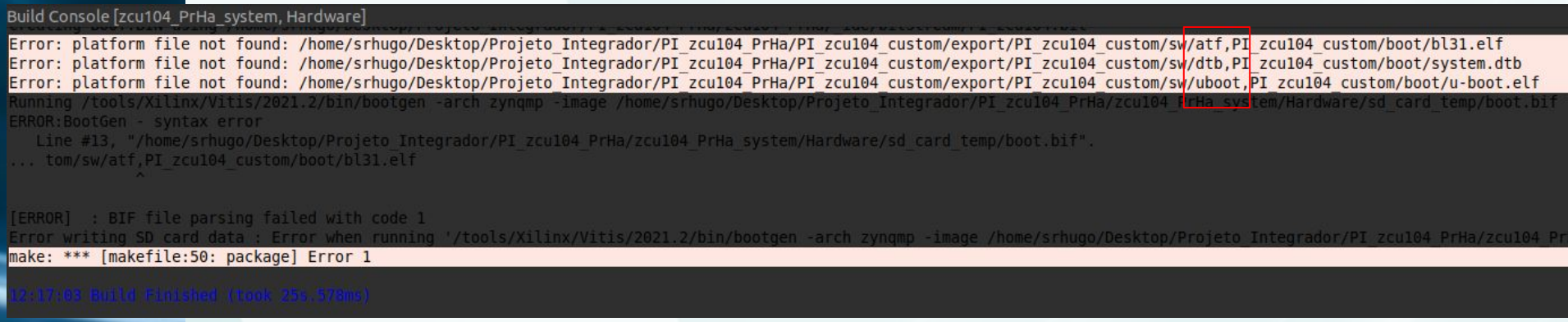
Test PrHa Code

- Save all changes and try to build the project



Test PrHa Code

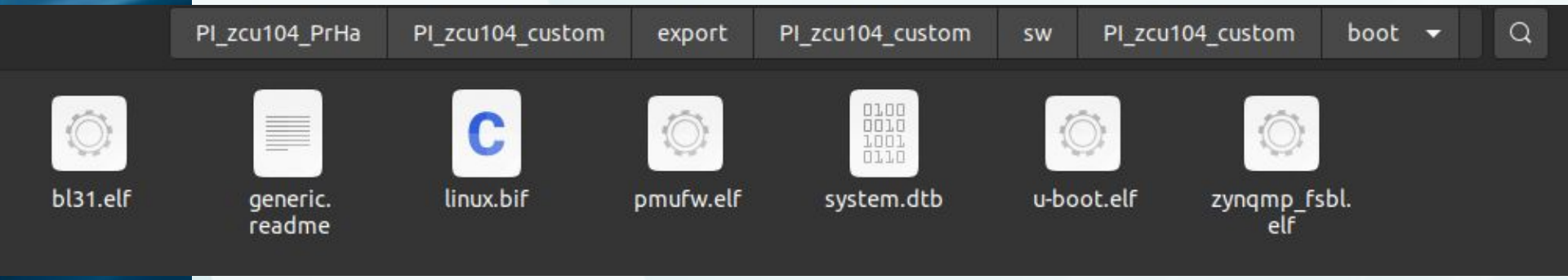
- There is a probability of vitis changing the directories when it's compiling for the first time



- This was added by vitis compiler! :(

Test PrHa Code

- 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>
}
```

Test PrHa Code

- 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

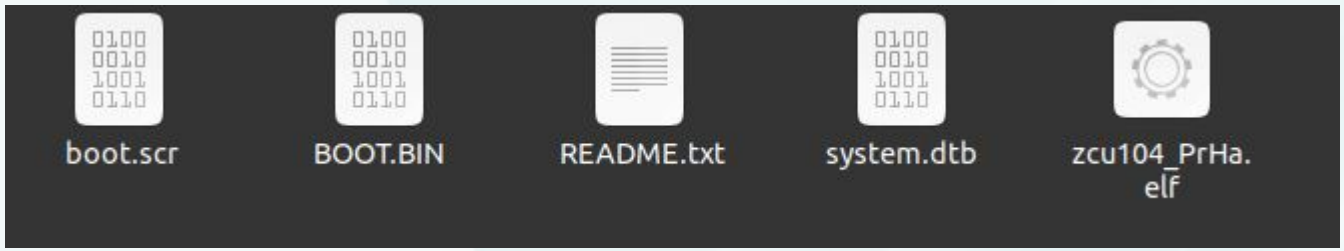
```
Console  Problems  Vitis Log  Guidance
Build Console [zcu104_PrHa_system, Hardware]
12:30:51 **** Incremental Build of configuration Hardware for project zcu104_PrHa_system ****
make all
Generating bif file for the system project
Executing command '::scw::generate_bif -xpfm /home/srhugo/Desktop/Projeto_Integrador/PI_zcu104_PrHa/PI_zcu104_custom/export/PI_zcu104_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)
```

Test PrHa Code

- Go to
  - <appllicationFolder>/<appllicationFolder>_system/Hardware/sd_card

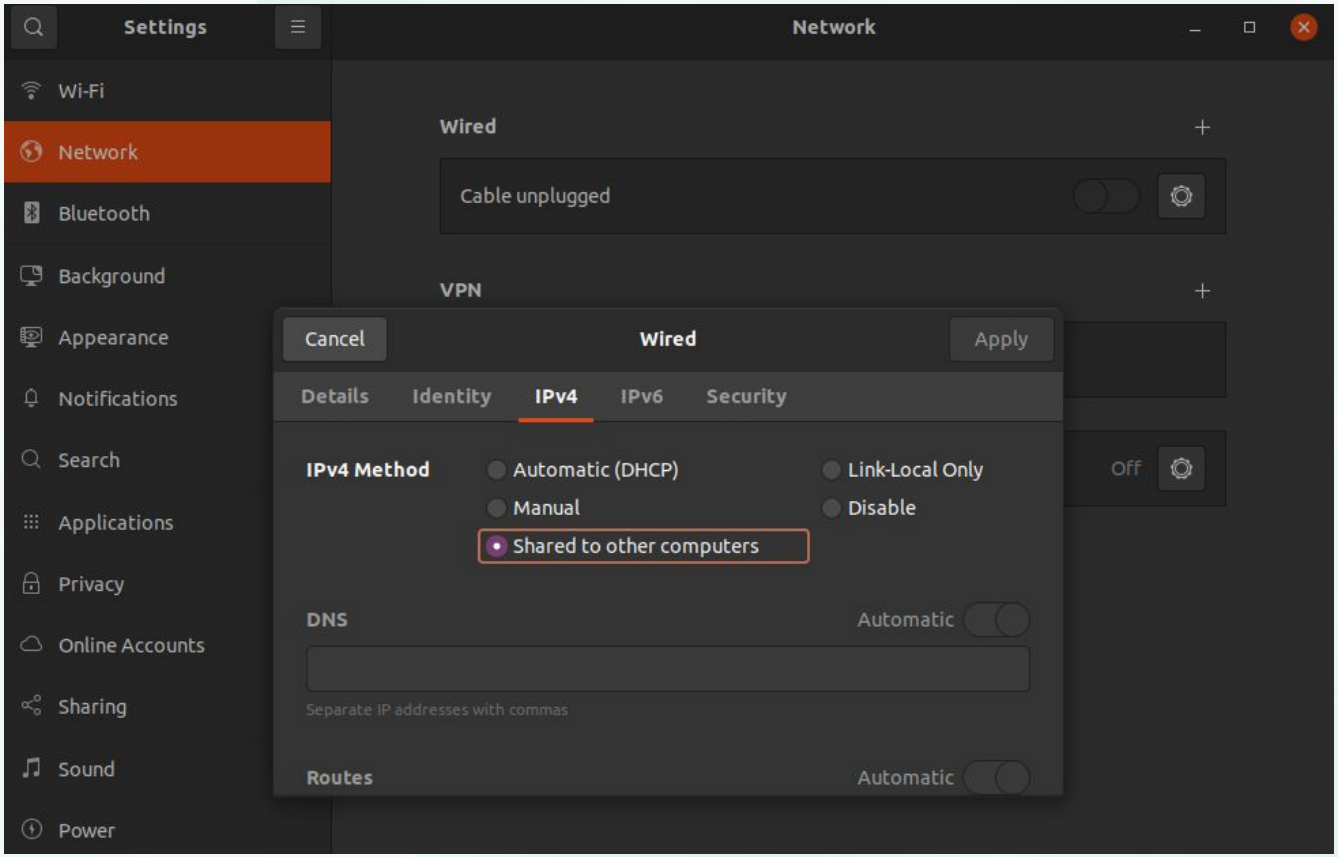
- Now you have the zcu104_PrHa program available



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

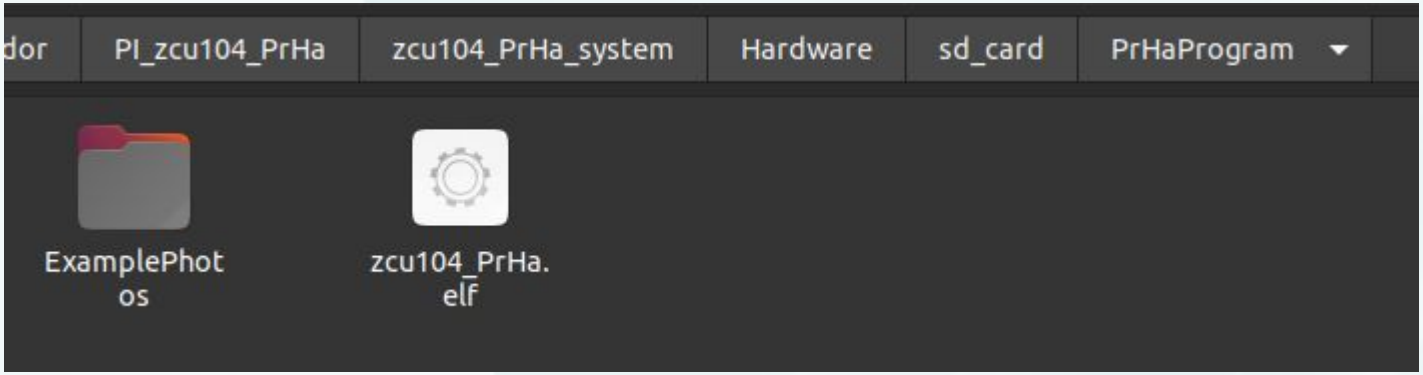
Test PrHa Code

- Make sure you have this setting in your host machine



Test PrHa Code

- 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

Test PrHa Code

- 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

Test PrHa Code

- Try to execute the program

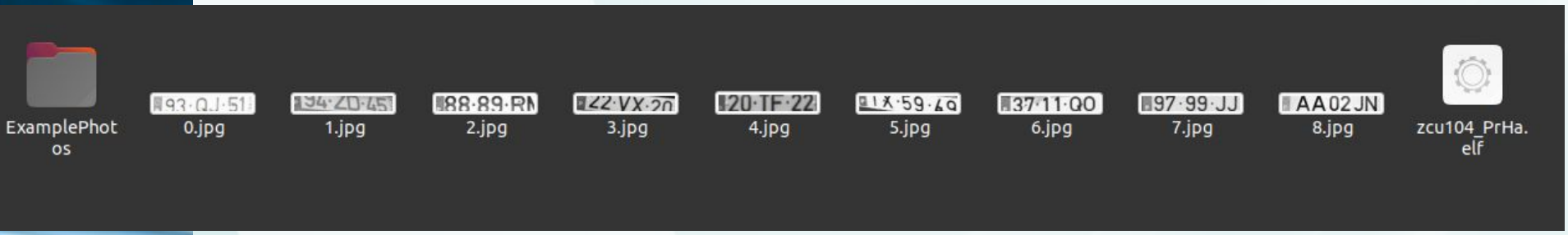
```
root@xilinx-zcu104-2021_2:/etc/PrHaProgram# ./zcu104_PrHa.elf
t_plateRecognition is ready
t_getPhoto Thread is Ready
Plate Founded :D
Plate Founded :D
Plate Founded :D
Plate Founded :D
Plate Founded :D
Plate Founded :D
Plate Founded :D
Plate Founded :D
Plate Founded :D
The End
```

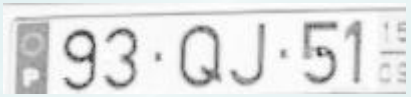
- SUCCESS! =D

Test PrHa Code

- 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://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/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