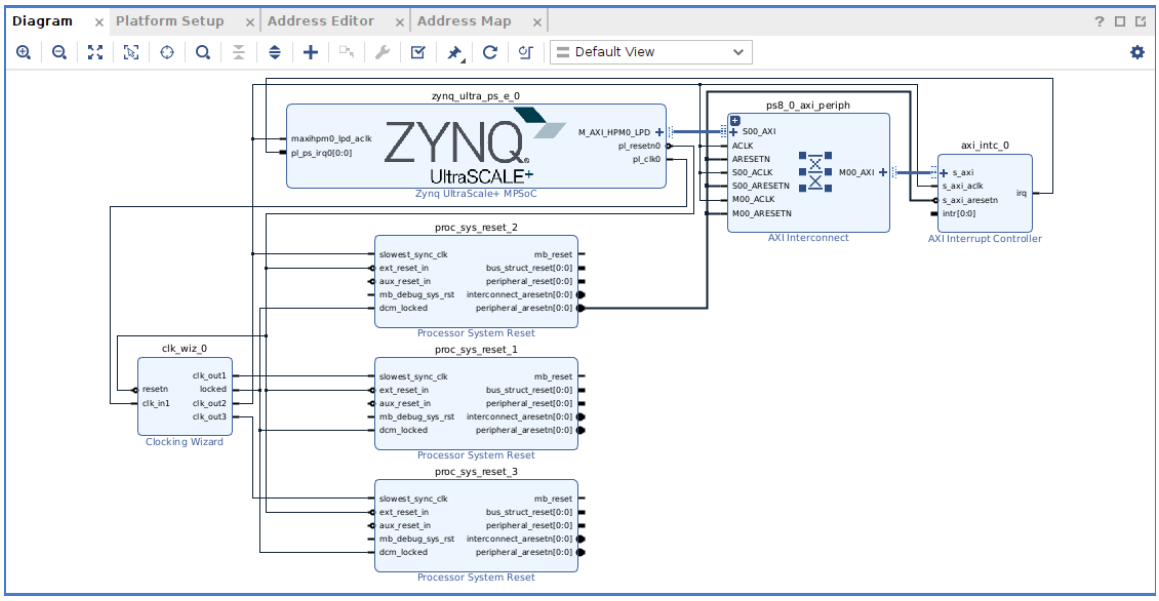
Custom Platform Creation for Vitis AI on MPSoc Using Vitis Flow

This is a recreation of Xilinx XD101 Vitis Platform Creation on ZCU102.
In this tutorial we will create a custom Vitis embedded platform for ZCU102. This platform will be targeted to be for using Vitis AI applications.
The platform creating process can be divided into three sections as described below
Section 1: Creating Base Hardware Platform in Vivado
Section 2: Creating Petalinux Build
Section 3: Adding DPU kernel onto the base Platform

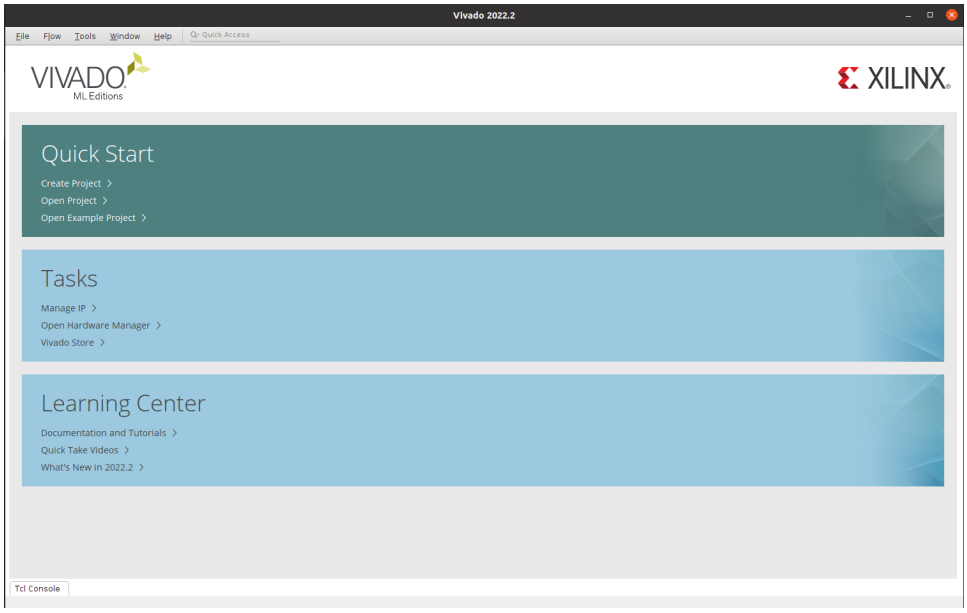
Section 1: Creating Base Hardware Platform in Vivado

In this section, we will use Vivado to create the base platform for ZCU102.
The final base platform will be as shown below.

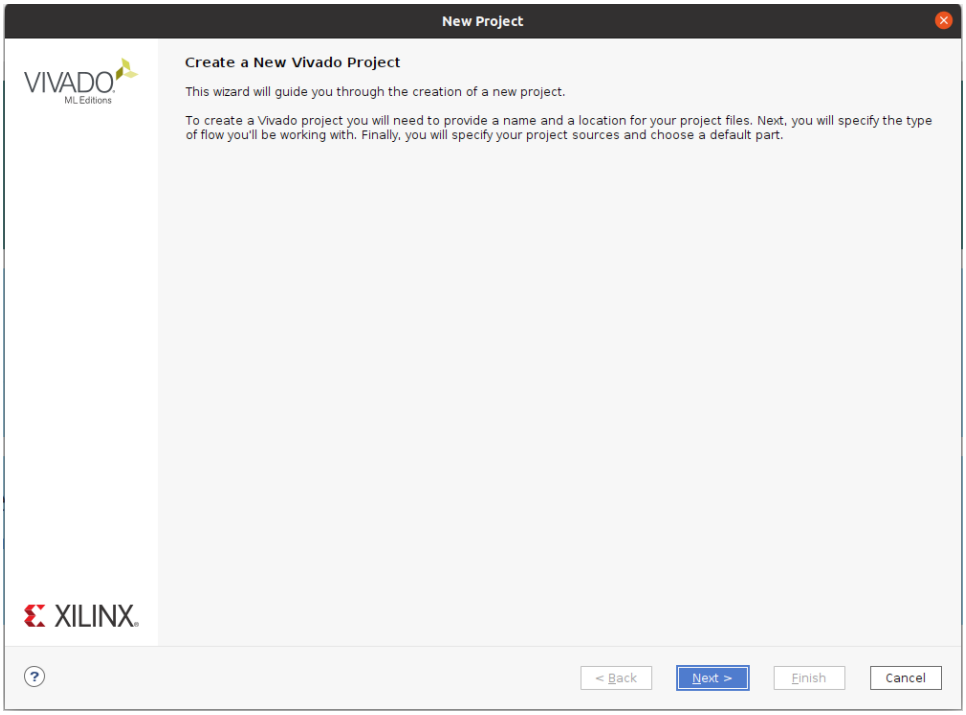


Building a Base Vivado Project form a Preset

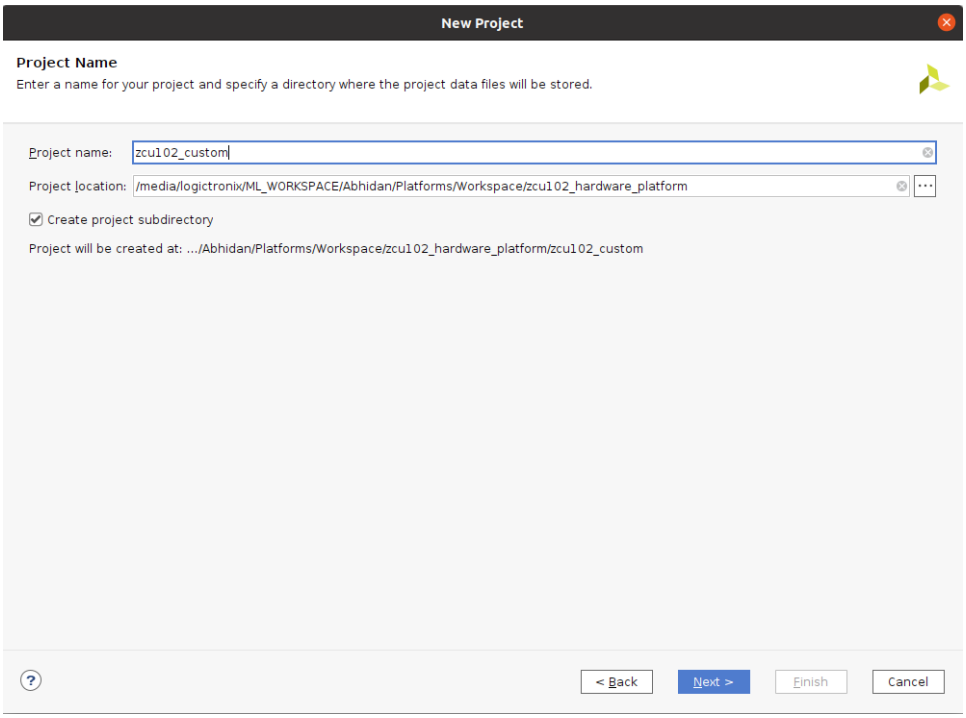
1. Start by creating a workspace and opening Vivado
2. Enter following commands into the Linux console
   source <vivado-installation-directory>/settings64.sh
   mkdir workspace
   cd workspace
   mkdir zcu102_hardware_platform
   cd zcu102_hardware_platform
   vivado &



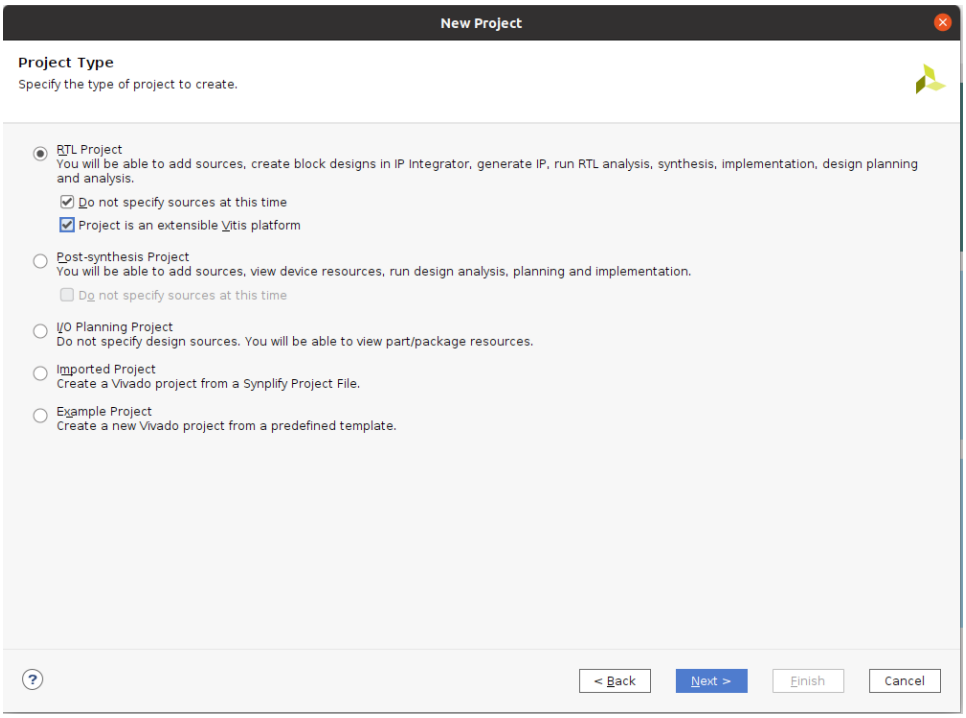
3. Create a Vivado project named zcu102_custom_platform
    a. Select File->Project->New, Click Next.



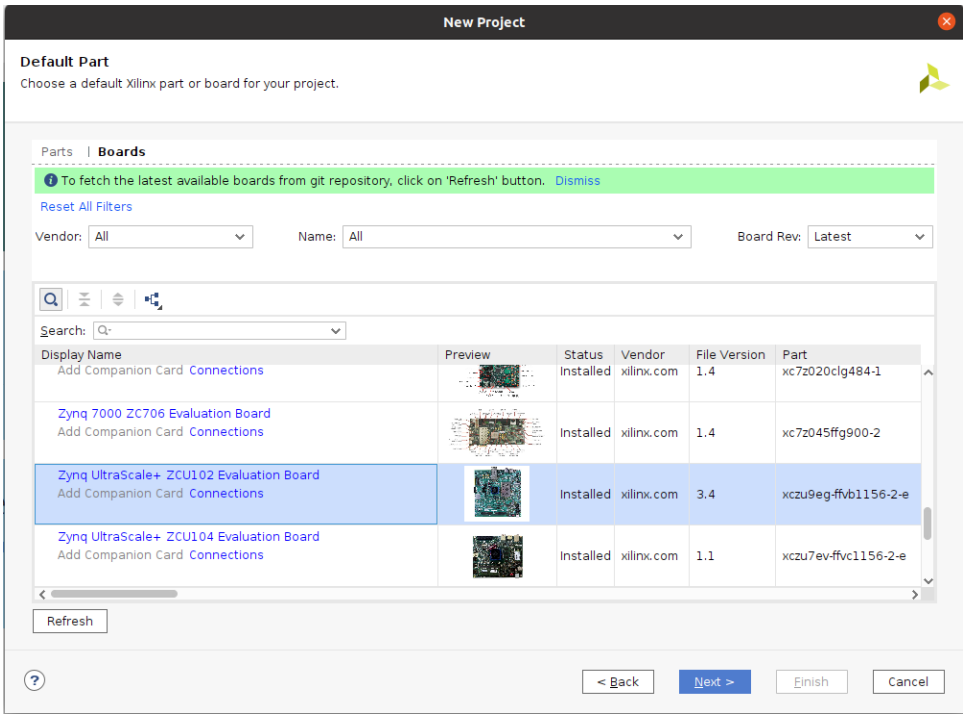
b. In Project Name dialog set Project name to zcu102_custom_platform. Uncheck Create project subdirectory and click Next.



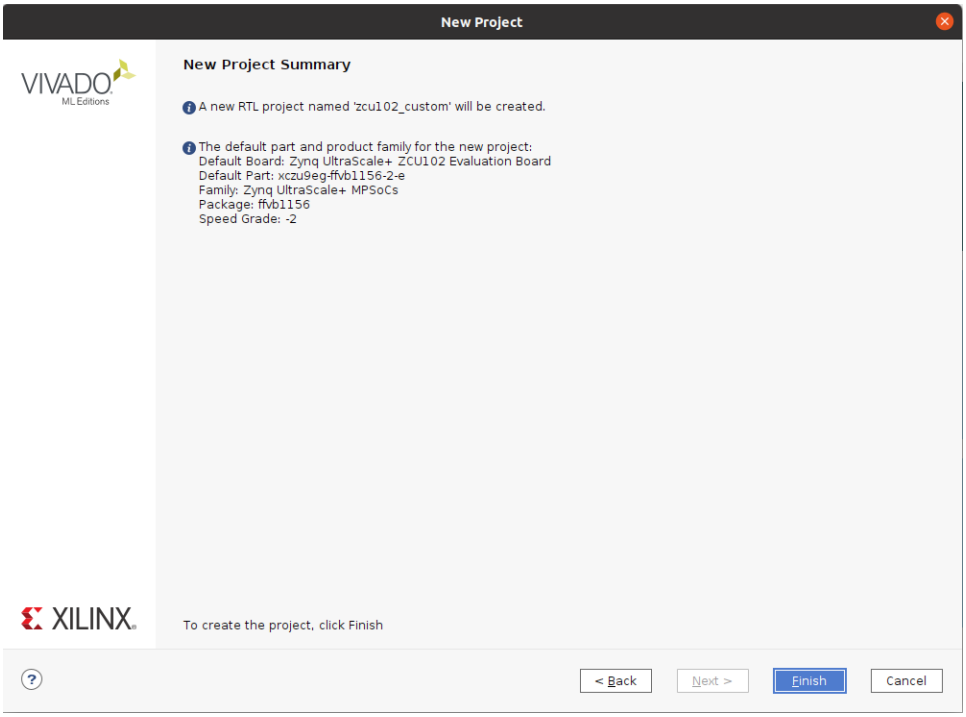
c. Enable Project is an extensible Vitis platform. Click Next.

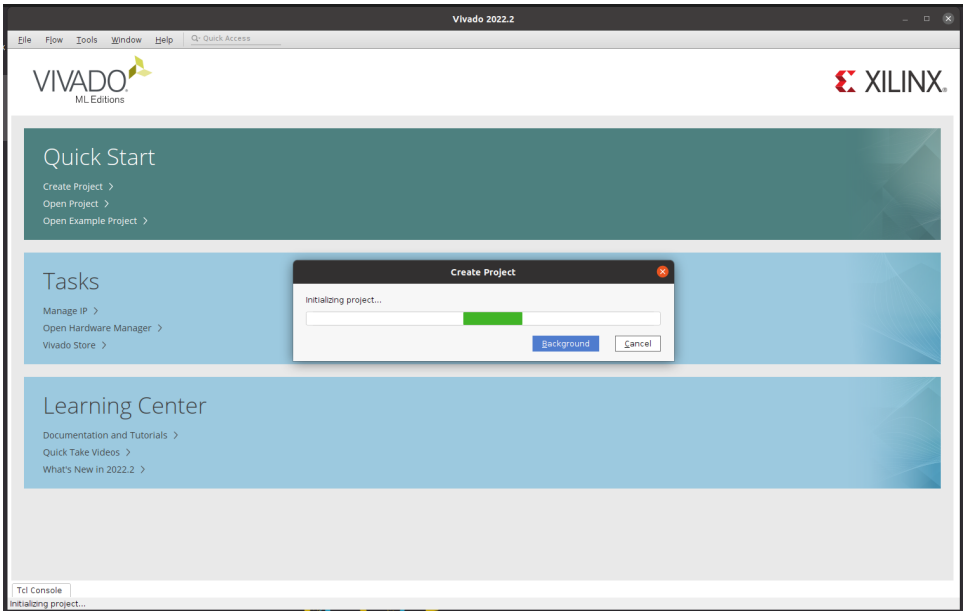


d. Select Boards tab and then select Zynq UltraScale+ZCU102 Evaluation Board. Click Next.

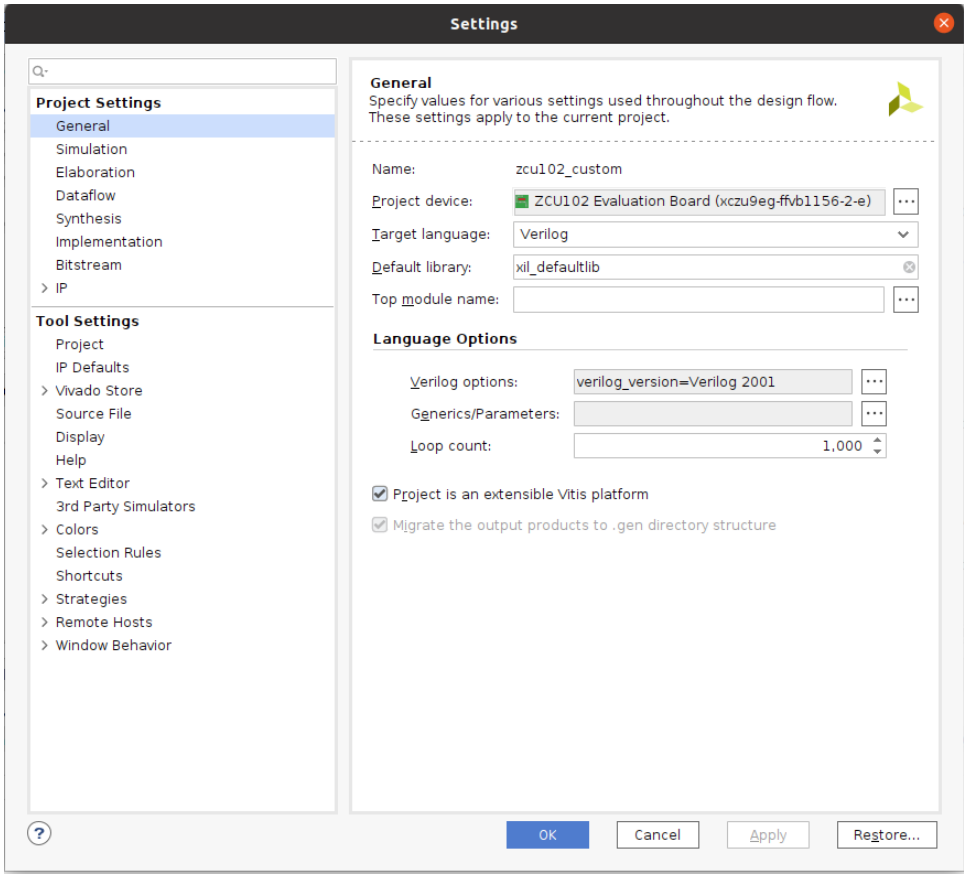


e. Review project summary and click Finish

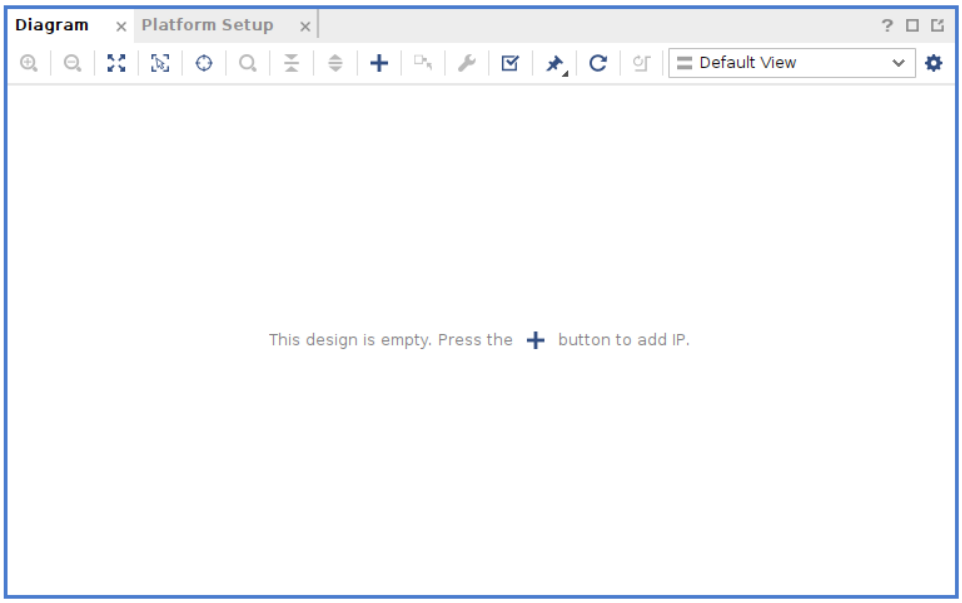




Note: If you need to change an existing Vivado project to an extensible Vitis platform project, you can goto Settings in Flow Navigator in an opened Vivado design, go to General and Enable project is an extensible Vitis Platform. To know more about platforms: https://docs.xilinx.com/r/2022.2-English/ug1393-vitis-application-acceleration/Platform-Types

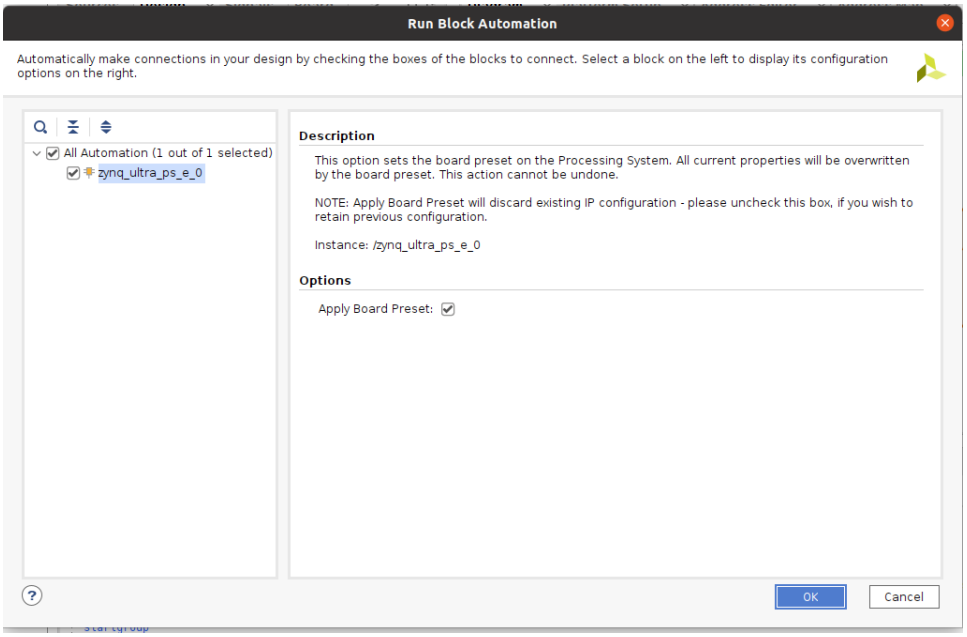


4. Create a block design
    a. In Project Manager, under IP INTEGRATOR, select Create Block Design
    b. Change the design name to your choice.
    c. Click OK.

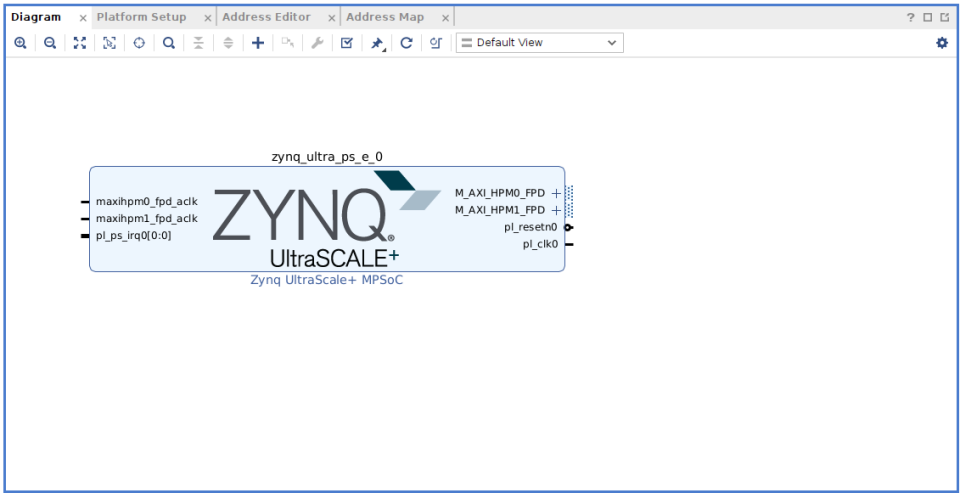


5. Add MPSoC IP and run block automation to configure it.

a. Right click Diagram view and select ADD IP
b. Search for zynq and then double-click the Zynq UltraScale+MPSoC from the IP search results.
c. Click the Run Block Automation link to apply the board presets. In the Run Block Automation dialog, ensure the following is check marked:
    i. All Automation
    ii. Zynq_ultra_ps_e_0
    iii. Apply Board Presets



d. Click OK. You should get MPSoC block configure like below.



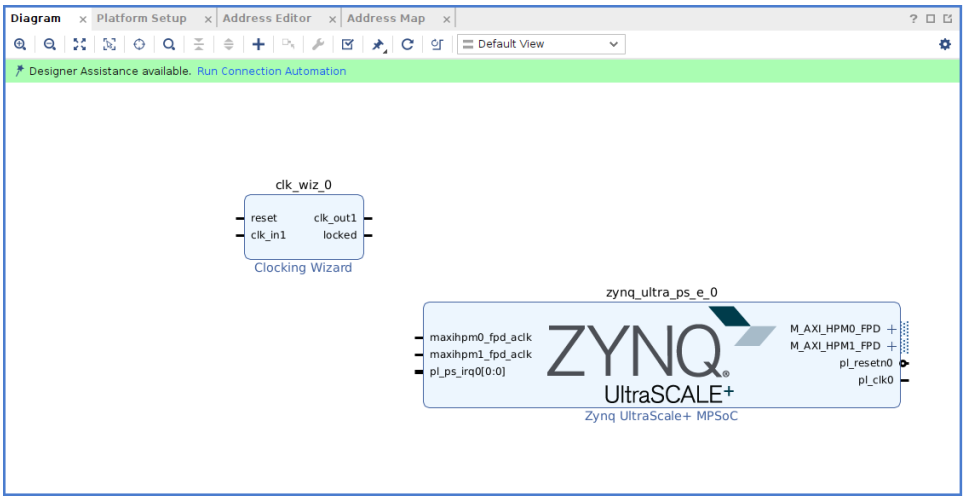
Note: At this stage, the Vivado block automation has added a Zynq UltraScale+ MPSoC block and applied all board presets for the ZCU102. The presets includes MPSoC PS side block configurations and pin assignments. TRM: https://docs.xilinx.com/r/en-US/ug1085-zynq-ultrascale-trm/Zynq-UltraScale-Device-Technical-Reference-Manual

Customize System Design for Clock and Reset:

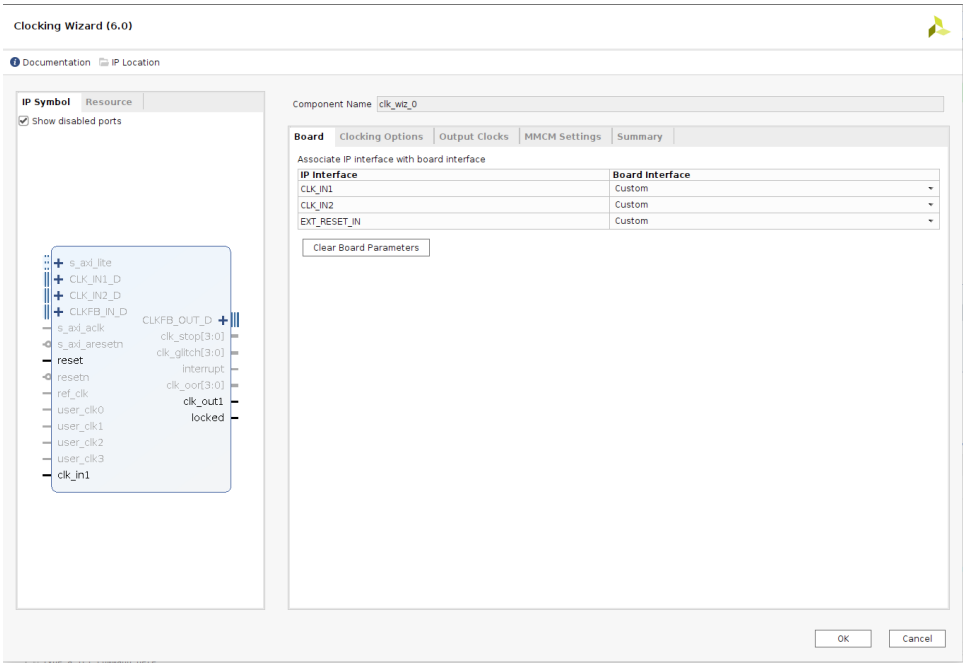
V++ linker can automatically link the clock signals between kernel and platform. The available clock signals in the platform are exported by PFM.CLK property.
For simple designs, interrupt signals can be sourced by processor's pl_clk. The limitation is that the processor has maximum 4 pl_clks and their phase is not aligned.

To access more interrupt signals or to have phase-aligned clocks, we can use a block called Clocking Wizard.
We will include the Clocking Wizard in our block diagram and turn on the clock signals for the platform. Here are the steps.

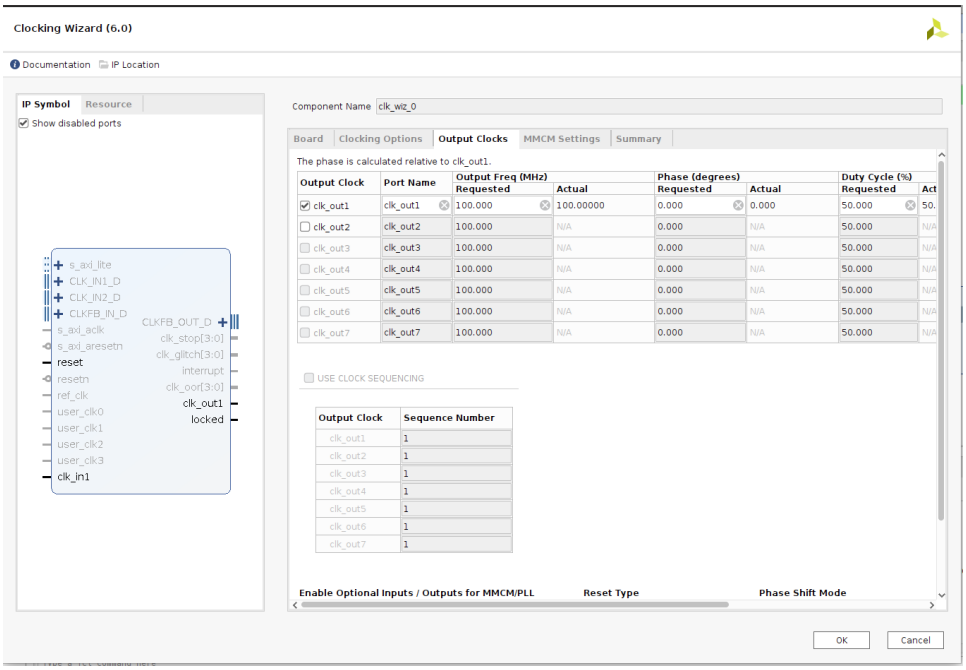
1. Adding the Clock Wizard Block to Generate Three Clocks:
    a. Right click Diagram view and select Add IP.
    b. Search for and add a Clocking Wizard from the IP Search dialog.



c. Double-click the clk_wiz_0 IP block to open the Re-Customize IP dialog box.

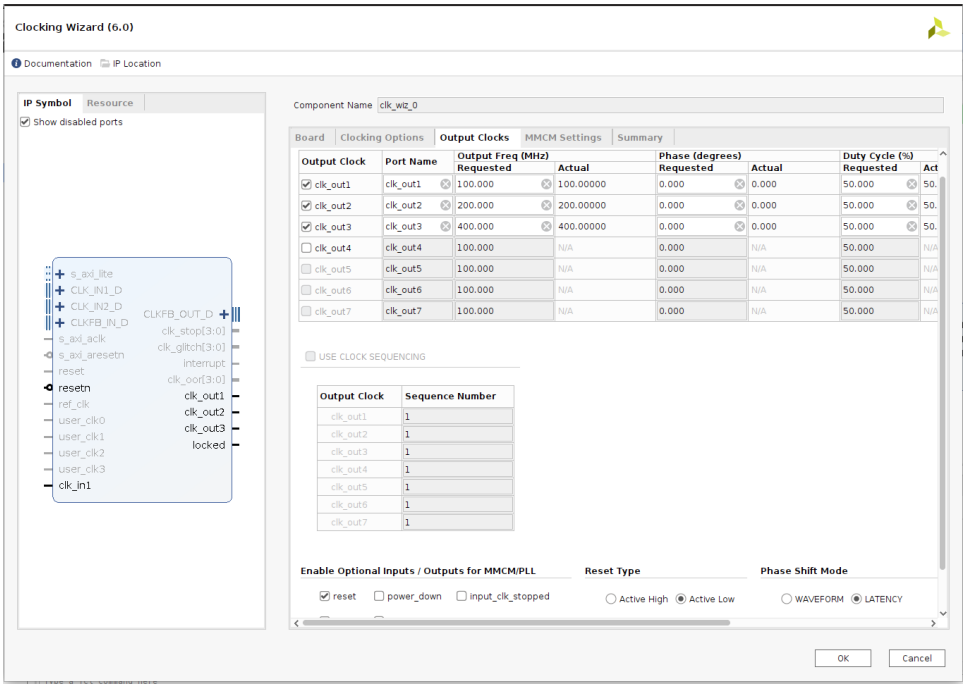


d. Click the Output Clocks tab.



e. Enable clk_out1 through clk_out3 in the Output Clock column. Set the Requested Output Freq as follows:

   i. clk_out1 to 100 MHz.
   ii. clk_out2 to 200 MHz.
   iii. clk_out3 to 400 MHz.

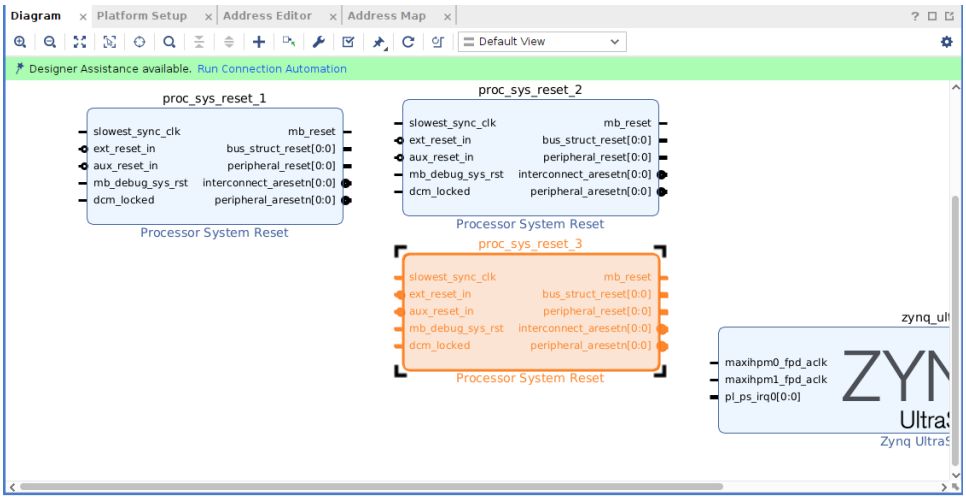


f. At the bottom of the dialog box set the Reset Type to Active Low.
g. Click OK to close the dailog box

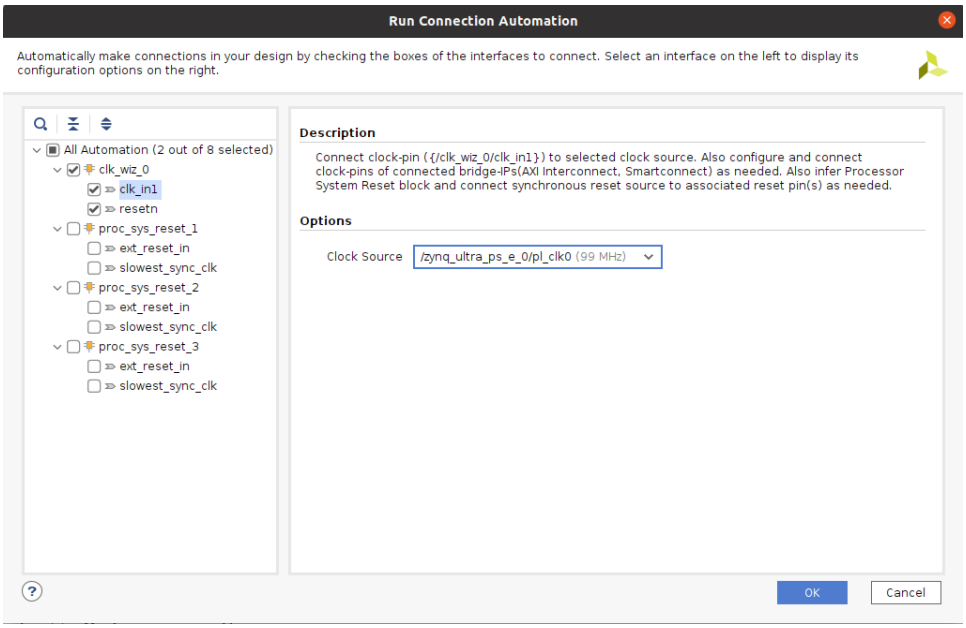
Note: So now we have set up the clock system for our design. This clock wizard uses the pl_clk as input clock and generates clocks needed for the whole logic design. In this simple design, we would use 100 MHz clock as the zi_lite control bus clock. 200MHz and 400MHz clocks are reserved for DPU AXI interface clock and DPU core clock during design linking phase. You are free to modify the clock quantities and frequency to fit your target design. We'll setup the clock export in future steps. Before that, we need to create reset signals for each clock because they are needed in clock export setup.

2. Add three Processor System Reset blocks corresponding to the three clocks:
    a. Right click Diagram view and select Add IP.
    b. Search for and add a Processor System Reset from the IP Search dialog
    c. Rename the reset block to proc_sys_reset_1 so that it's easy to understand the relationship between reset modules and the clock signals.

d. Select the proc_sys_reset_1 block and duplicate it twice. The newly created modules will be named as proc_sys_reset_2 and proc_sys_reset_3 by default.

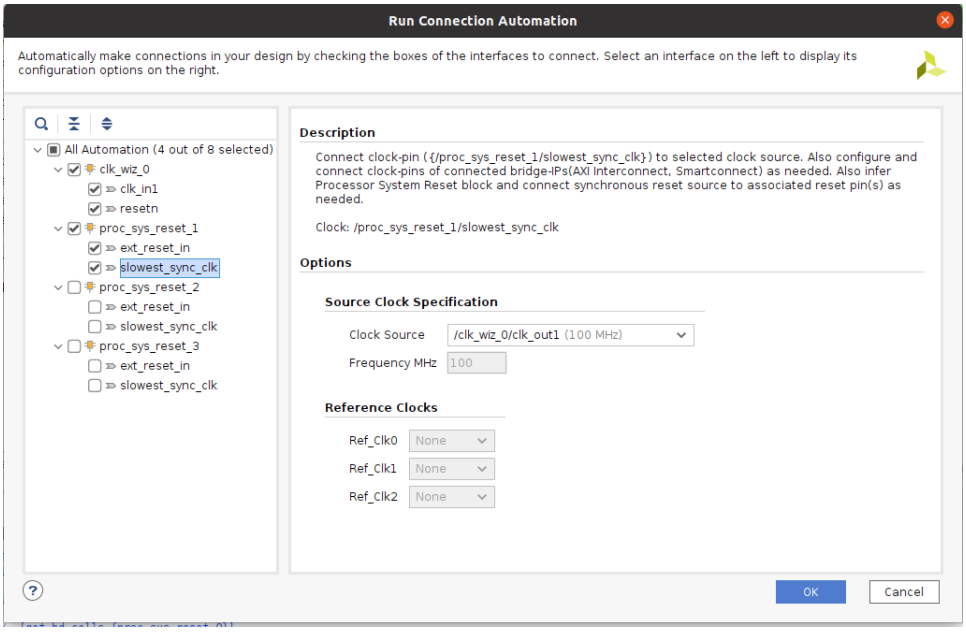


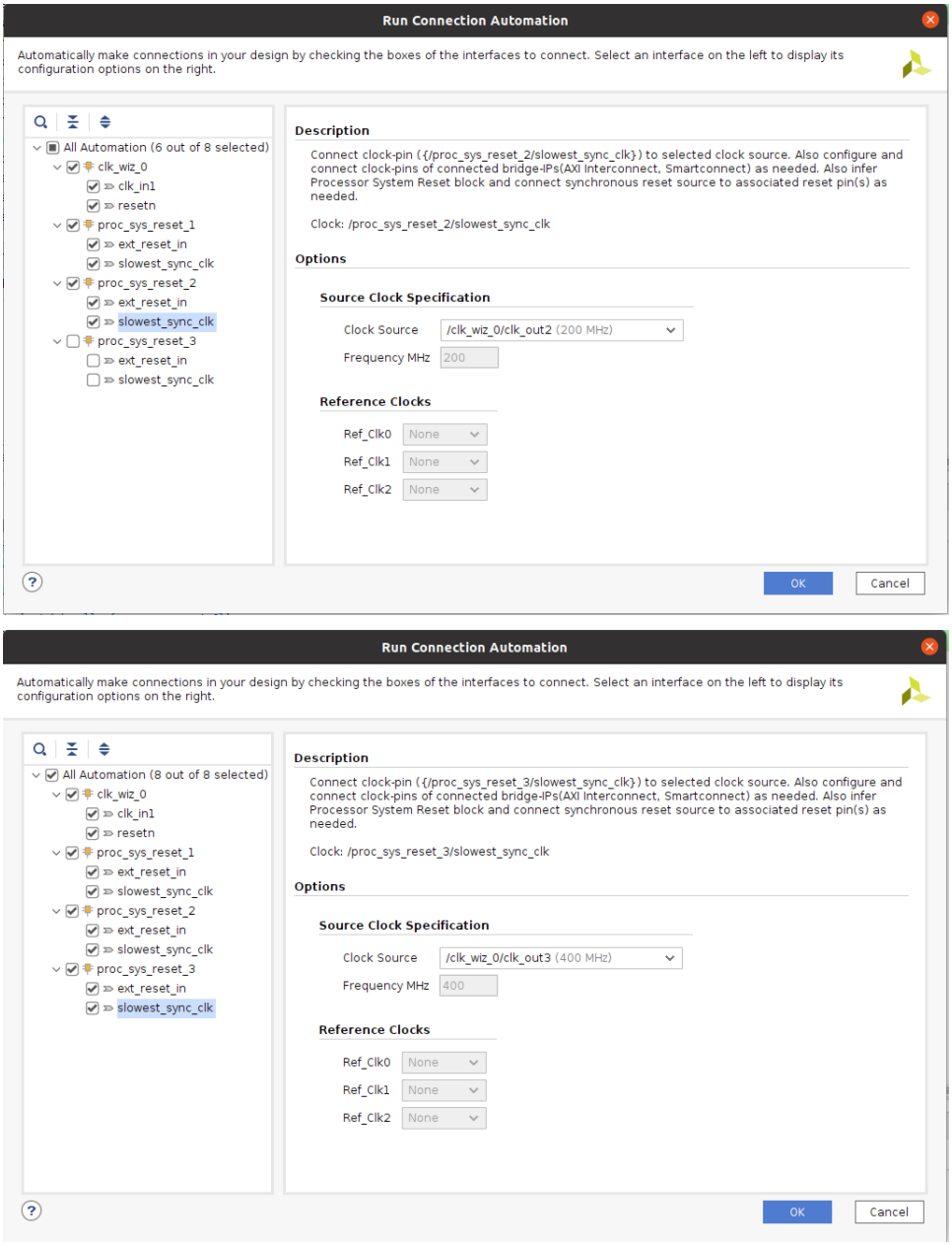
3. Connect Clocks and Resets:
   a. Click Run Connection Automation, which will open a dialog that will help connect the proc_sys_reset blocks to the clocking wizard clock output.
   b. Enable All Automation on the left side of the Run Connection Automation dialog box.
   c. Select clk_in1 on clk_wiz_0, and set the Clock Source to /zynq_ultra_ps_e_0/pl_clk0.



   d. For each proc_sys_reset instance, select the slowest_sync_clk, and set the Clock Source as follows:
      i. proc_sys_reset_1 with /clk_wiz_0/clk_out1
      ii. proc_sys_reset_2 with /clk_wiz_0/clk_out2

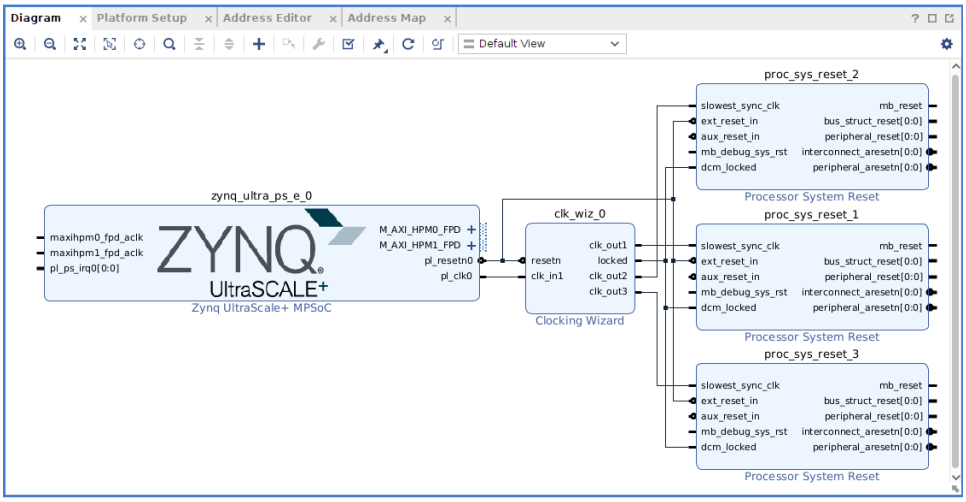
iii. proc_sys_reset_3 with /clk_wiz_0/clk_out3



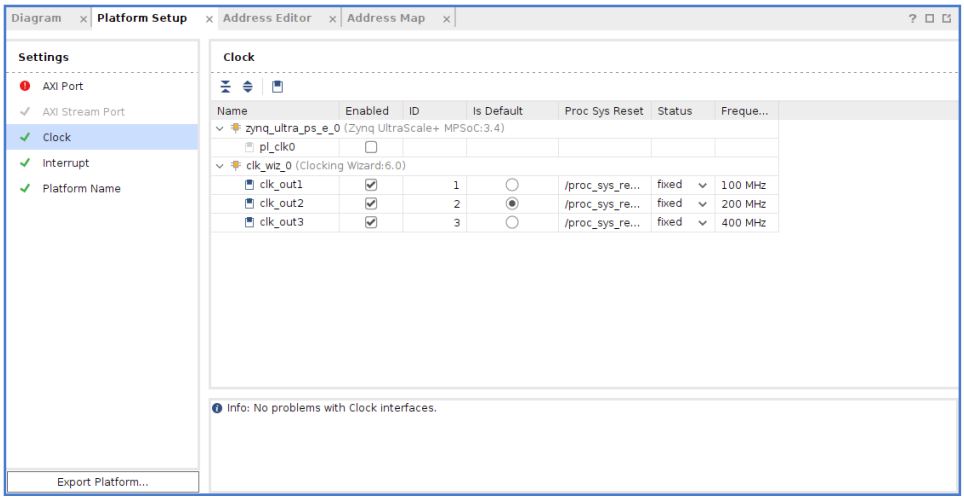


e. On each proc_sys_reset instances, select the ext_reset_in, set Board Part Interface to Custom and set the Select Manual Source to /zynq_ultra_ps_e_0/pl_resetn0.
f. Make sure all check boxes are enabled, and click OK to close the dialog and create the connections.

g. Connect all the dcm_locked signals on each proc_sys_reset instance to the locked signal on clk_wiz_0.



4. Enable clocks for the platform.
   a. Goto Platform Setup tab.
   b. If it's not opened yet, use menu Window-> Platform Setup to open it.
   c. Click the Clock tab.
   d. Enable all clocks under clk_wiz_0: clk_out1, clk_out2, clk_out3
   e. Change their ID to 1,2 and 3
   f. Set a default clock: click Is Default for clk_out2



g. After everything is setup, it should report Info: **No problem with Clock interface**

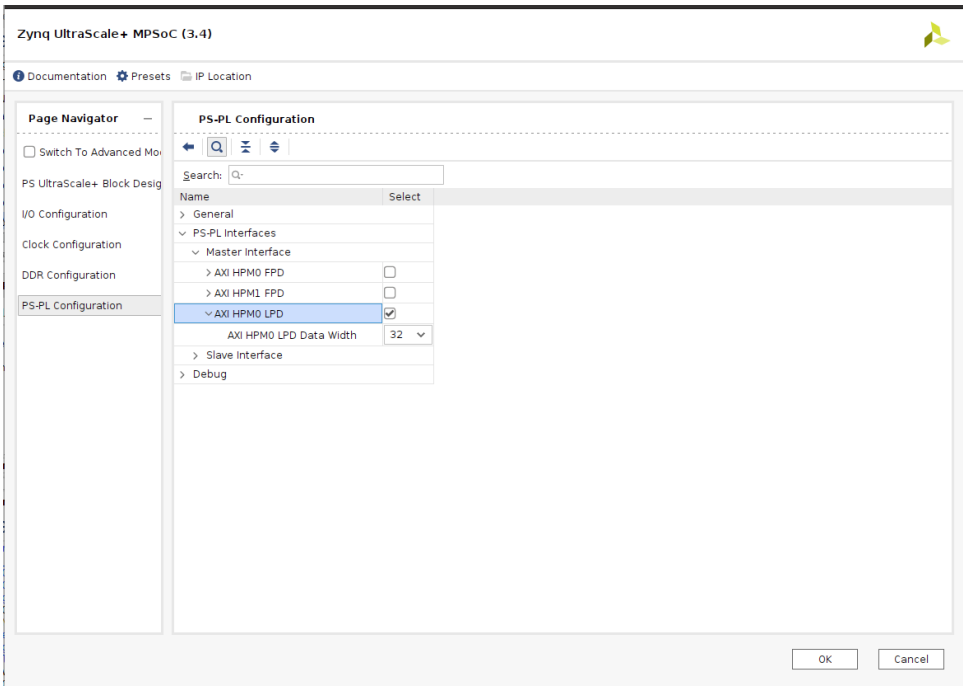
Note: There should be one and only one default clock in the platform. During v++ linking stage linker will use the default clock to connect the IP blocks if there are no user assignments for link configuration.

Add Interrupt Support

V++ linker can automatically link the interrupt signals between kernel and platform. The available interrupt signals in the platform are exported by PFM.IRQ property.

For simple designs, interrupt signals can be sourced by processor's pl_ps_irq. The limitation is that it can only provide maximum 16 interrupt signals. To provide more interrupt signals, We can use AXI Interrupt Controller. We'll enable AXI HPM0 LPD to control the AXI Interrupt Controller. The next we will add the AXI Interrupt Conroller and enable interrupt signals for PFM.IRQ. Here are the detailed steps.

1. Enable AXI HPM0 LPD to control the AXI Interrupt Controller
    - In the block diagram, double-click the Zynq UltraScale+ MPSoC block.
    - Select PS-PL Configurations -> PS-PL interfaces -> Master interface.
    - Enable the AXI HPM0 LPD option.
    - Expand the arrow before AXI HPM0 LPD. Check the AXI HPM0 LPD Data width settings and keep it as default 32.
    - Disable AXI HPM0 FPD and AXI HPM1 FPD



    - Click OK to finish the configuration.

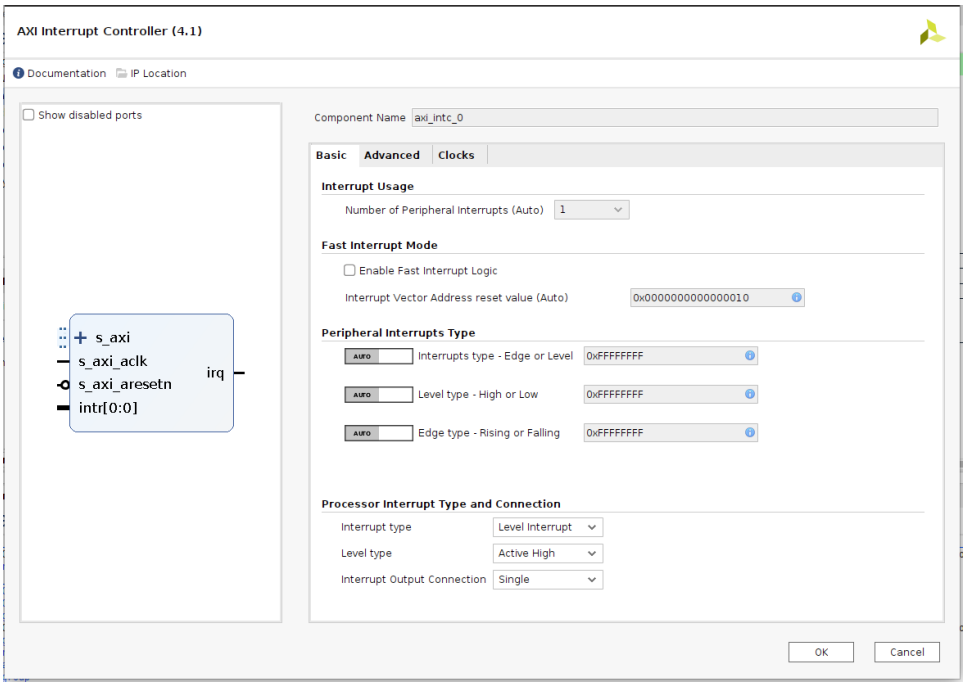
    Note:

    a. We use AXI HPM0 LPD mainly for controlling purpose. It would read and write 32 bit control registers. If the interface is more than 32, AXI Interconnect or SmartConnect will do AXI bus width conversion using PL logic. It would cost logic resources and introduce unnecessary latency.

    b. We reserve AXI HPM0 FPD and AXI HPM1 FPD for kernel usage. Disabling them from the block diagram can prevent auto connection to use it by accident. We can export the unused AXI Interfaces in Platform Setup, no matter it's visible in the block diagram or not.

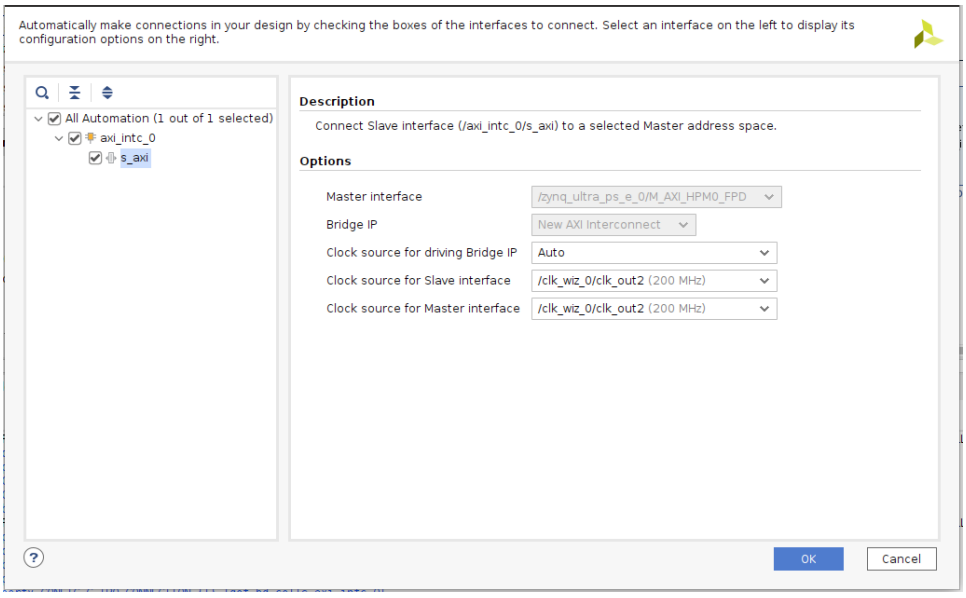
2. Add the AXI Interrupt Controller and configure it
    - Right click Diagram view and select Add IP, search and add AXI Interrupt Controller IP. It's instantiated as axi_intc_0.

- Double click the AXI Interrupt Controller block, change Interrupt Output Connection to Single so that it can be connected to PS IRQ interface.



- Click OK

3. Connect AXI Interfaces of axi_intc_0 to AXI HPM0 LPD of PS
   - Click Run Connection Automation
   - Review the settings (axi_intc_0 is enabled, s_axi is to be connect to /zynq_ultra_ps_e_0/M_AXI_HPM0_LPD)
   - Set Clock Source for Slave Interface and Clock Source for Master Interface to /clk_wiz_0/clk_out2(200MHz)

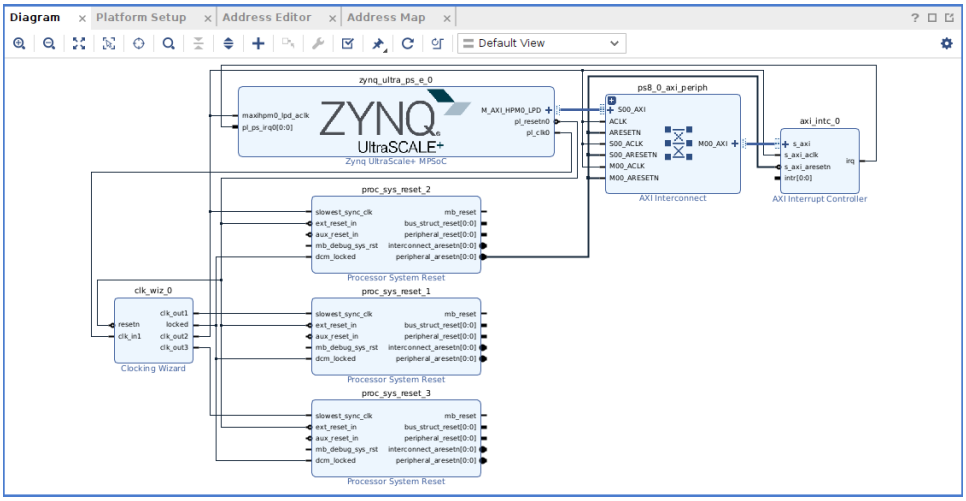


   - Click OK

Note:

- We wish interrupt controller and most kernel IRQ signals are synchronous to one clock. It's best for stability. But don't worry about the asynchronous IRQ if kernels are running at different clocks. The interrupt controller can handle asynchronous IRQ with level interrupt signals as well.

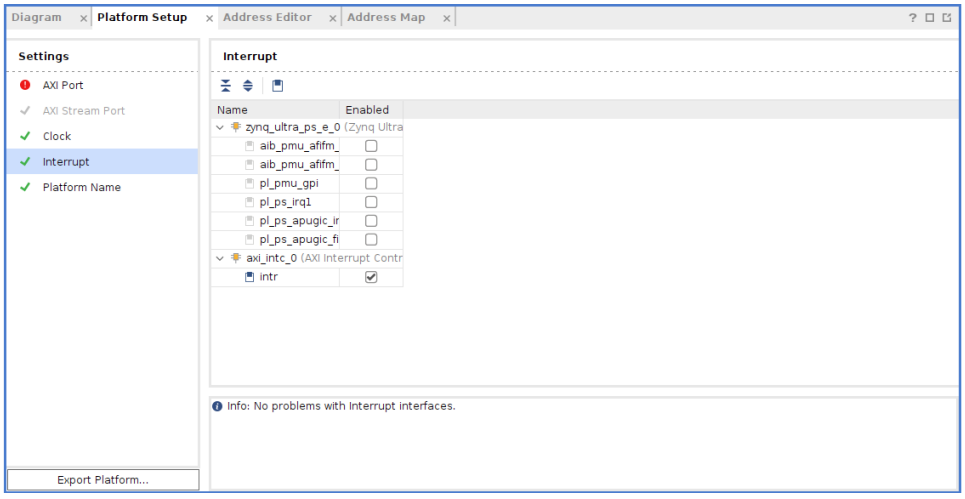
4. Connect irq of the Interrupt Controller
    - Connect **axi_intc_0.irq** to **zynq_ultra_ps_e_0.pl_ps_irq[0:0]**



Note:

- If you have more than one irq signals to connect to pl_ps_irq of PS, use a concat IP to concatenate them to a bus and then connect the bus to pl_ps_irq.

5. Enable interrupt signals for the platform.
    - Go to Platform Setup tab
    - Go to Interrupt tab
    - Enable intr under axi_intc_0



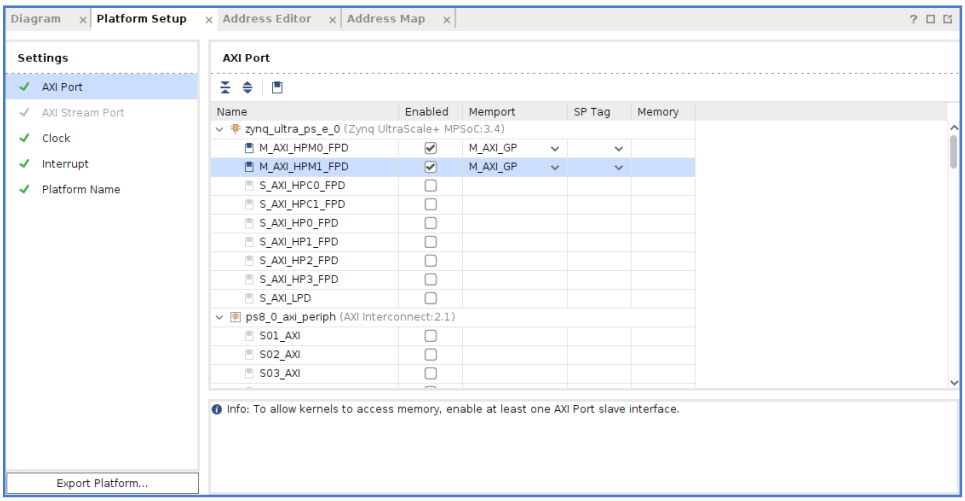
    - Tcl console shows the corresponding tcl command for this setup

```
set_property PFM.IRQ {intr { id 0 range 32 }}
[get_bd_cells /axi_intc_0]
```

The IPI design connection would look like below till now:

Enable AXI Interfaces for the Platform:

1. Enable AXI Master interfaces from PS
    - Go to Platform Setup tab
    - Go to AXI Port tab in Platform Setup
    - Under zynq_ultra_ps_e_0, enable **M_AXI_HPM0_FPD** and **M_AXI_HPM1_FPD**. Keep the Memport and sptag default to M_AXI_GP and empty.

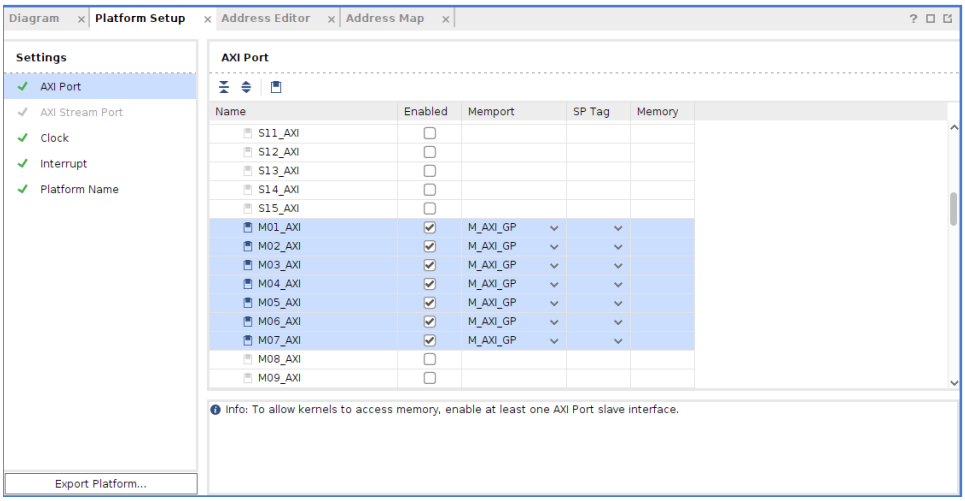


Note:

- M_AXI_GP means general purpose AXI Master interface
- sptag is only applicable to AXI slave interfaces.
- V++ linker will instantiate AXI Interconnect automatically to connect between PS AXI Master interfaces and slave interfaces of acceleration kernels. One AXI Master Interface will connect upto 16 kernels.

2. Enable AXI Master interfaces from AXI Interconnect
    - Under ps8_0_axi_periph, click M01_AXI, press Shift and click M07_AXI to multi-select master interfaces from M01_AXI to M07_AXI.
    - Right click the selection and click on Enable.

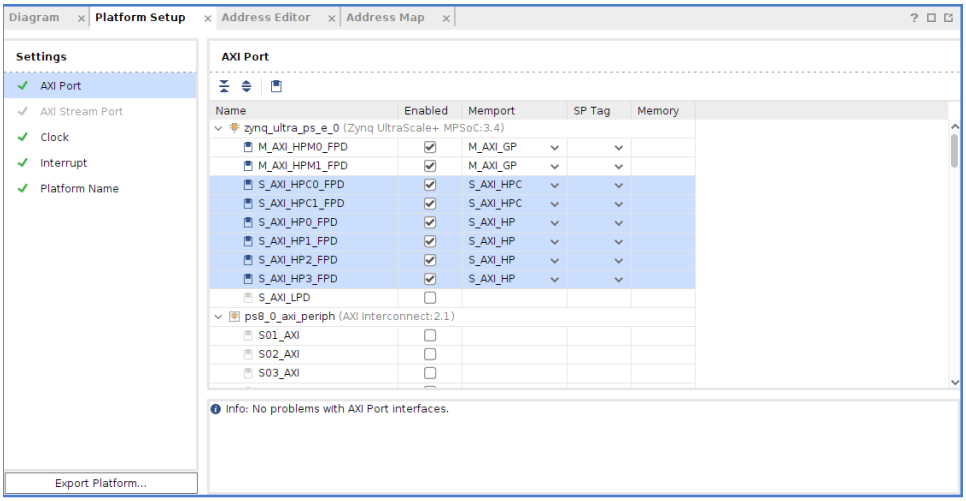
- Keep the Memport and sptag default to M_AXI_GP and empty.



Note:

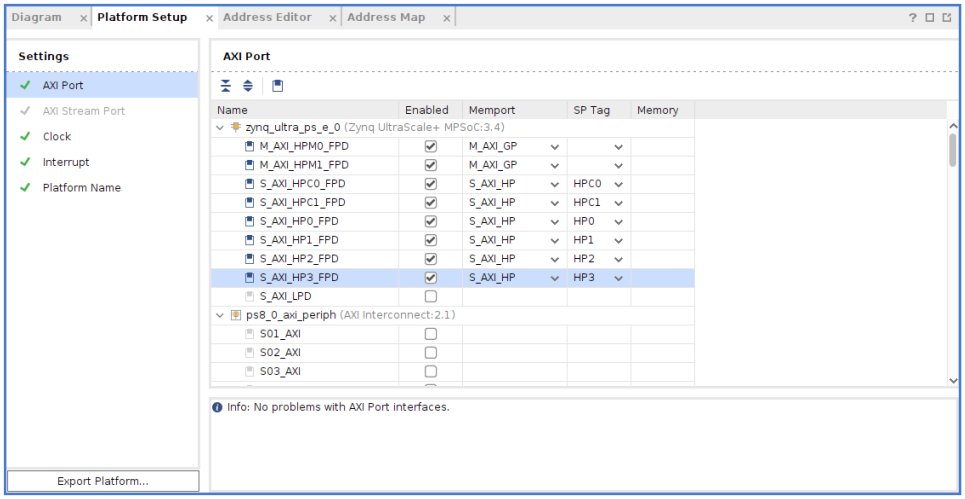
- V++ will not cascade another level of AXI Interconnect if the AXI master interface is exported from AXI Interconnect IP.
- AXI Master interfaces from PS and AXI Interconnect are functionally equivalent to the platform.
- In general, platform designer should export as many as AXI interfaces to the platform. Application developer should decide which interface to use.

3. Enable AXI Slave interfaces from PS to allow kernels access DDR memory
   - Under zynq_ultra_ps_e_0, multi-select all AXI slave interfaces: press Ctrl and click S_AXI_HPC0_FPD, S_AXI_HPC1_FPD, S_AXI_HP0_FPD, S_AXI_HP2_FPD, S_AXI_HP3_FPD.
   - Right click the selections and select enable.



   - Change Memport of S_AXI_HPC0_FPD and S_AXI_HPC1_FPD to S_AXI_HP because we won't’ use any coherent features for these interfaces.

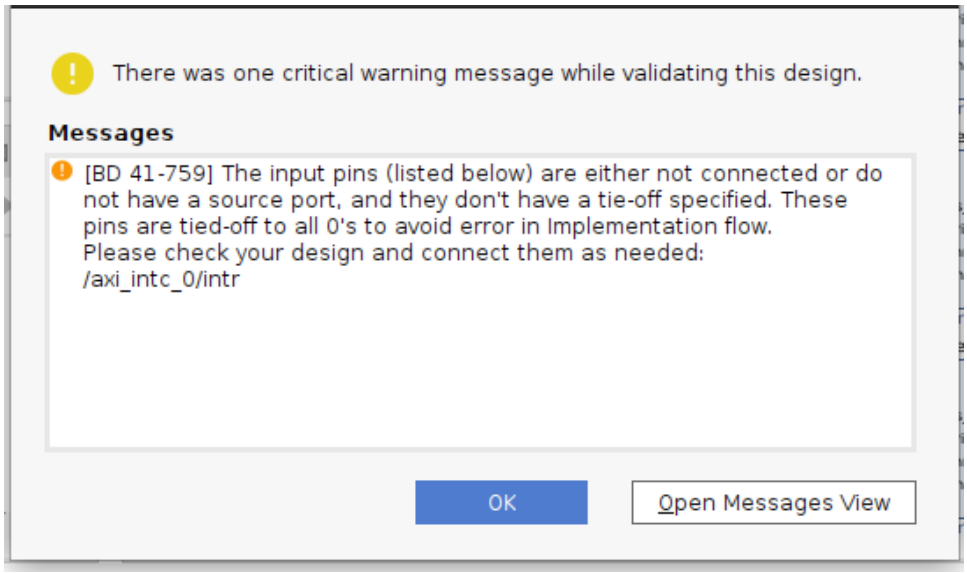
- Type in simple sptag names for these interfaces so that they can be selected by v++ configuration during linking phase. **HPC0, HPC1, HP0, HP1,HP2,HP3.**



Export Hardware XSA:

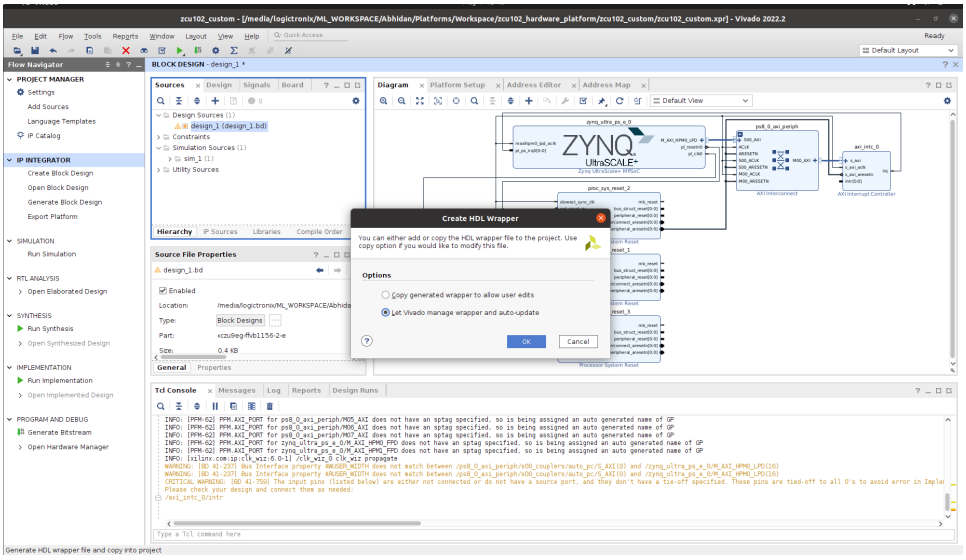
1. Validate the block design
   - Click the **Validate Design** button in the block design Diagram window

Note: During validation, Vivado reports a critical warning that /axi_intc_0/intr is not connected. This warning can be safely ignored because v++ linker will link kernel interrupt signals to this floating intr signal.

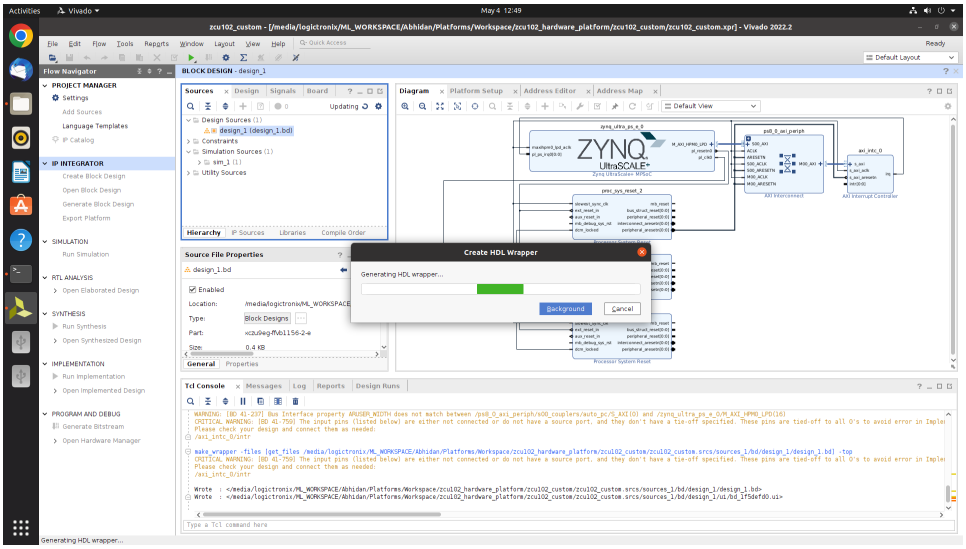


2. Create a top module wrapper for the block design
   - In Source tab, right click system.bd in Design Sources group
   - Select Create HDL Wrapper…

- Select Let Vivado manage wrapper and auto-update.

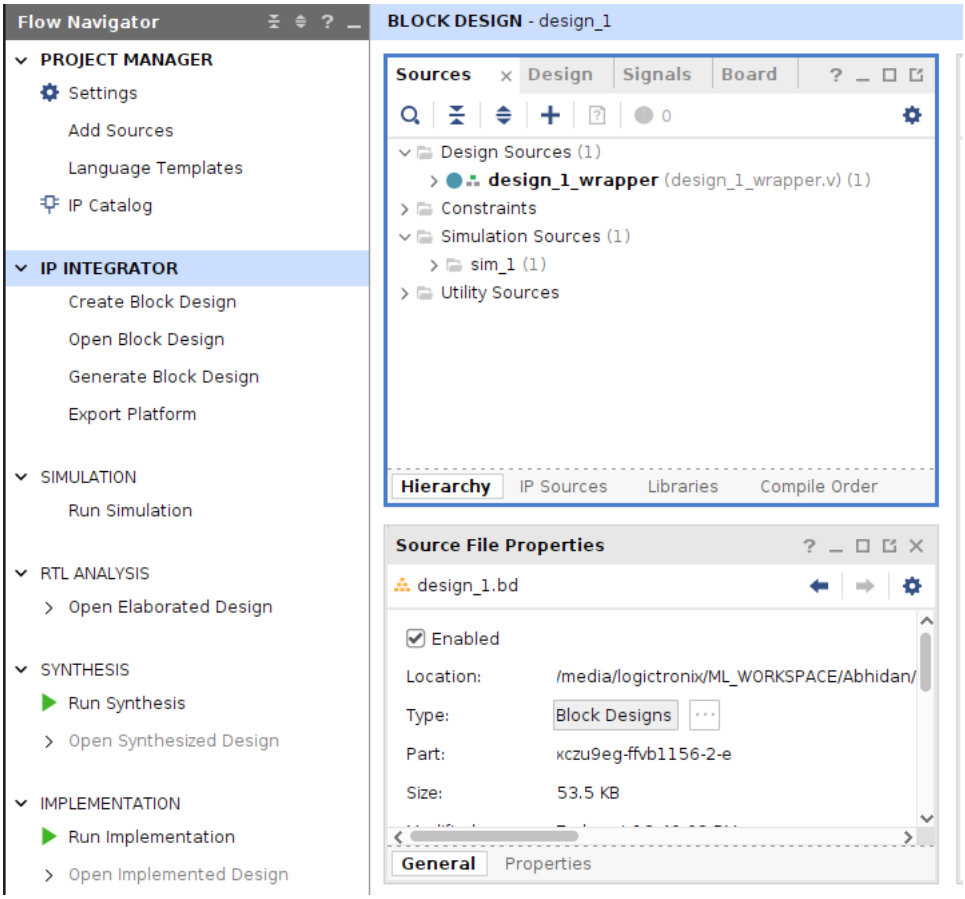


- Click OK to generate wrapper for block design.

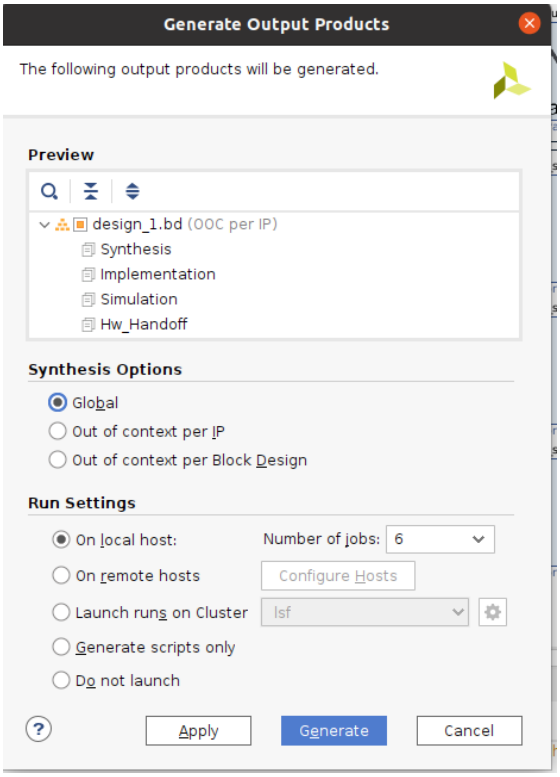


3. Generate pre-synth design

- Select Generate Block Design from Flow Navigator



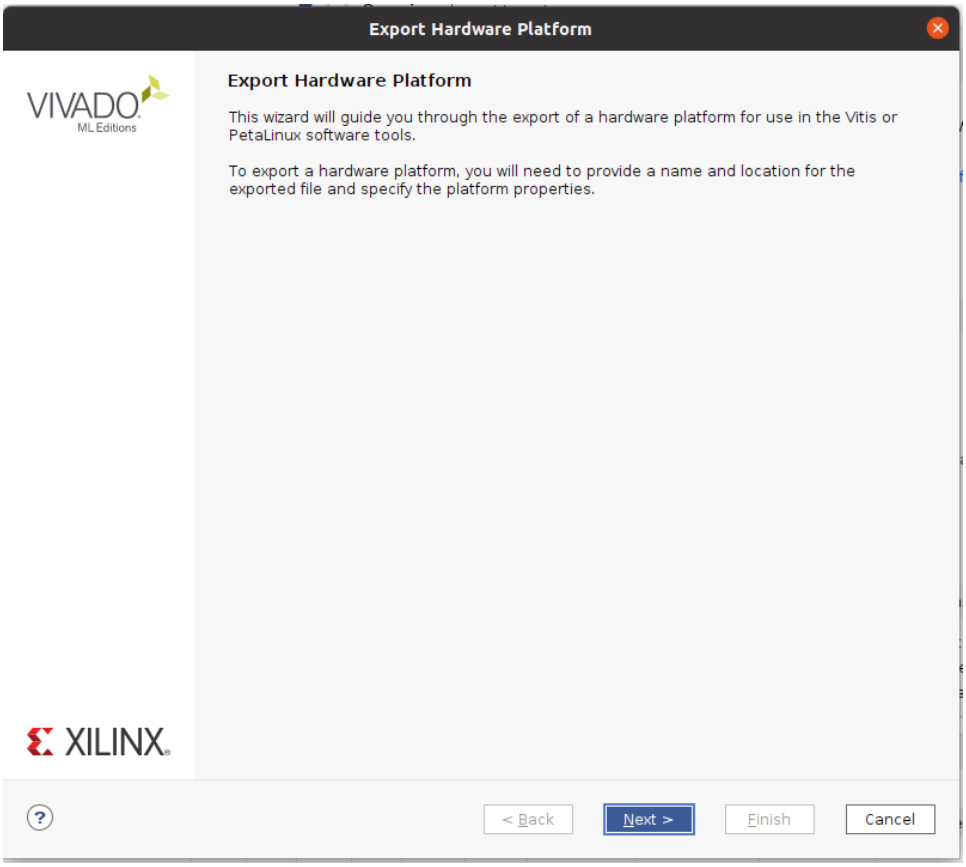
- Select Synthesis Options to Global. It will skip IP synthesis during generation



- Click Generate.

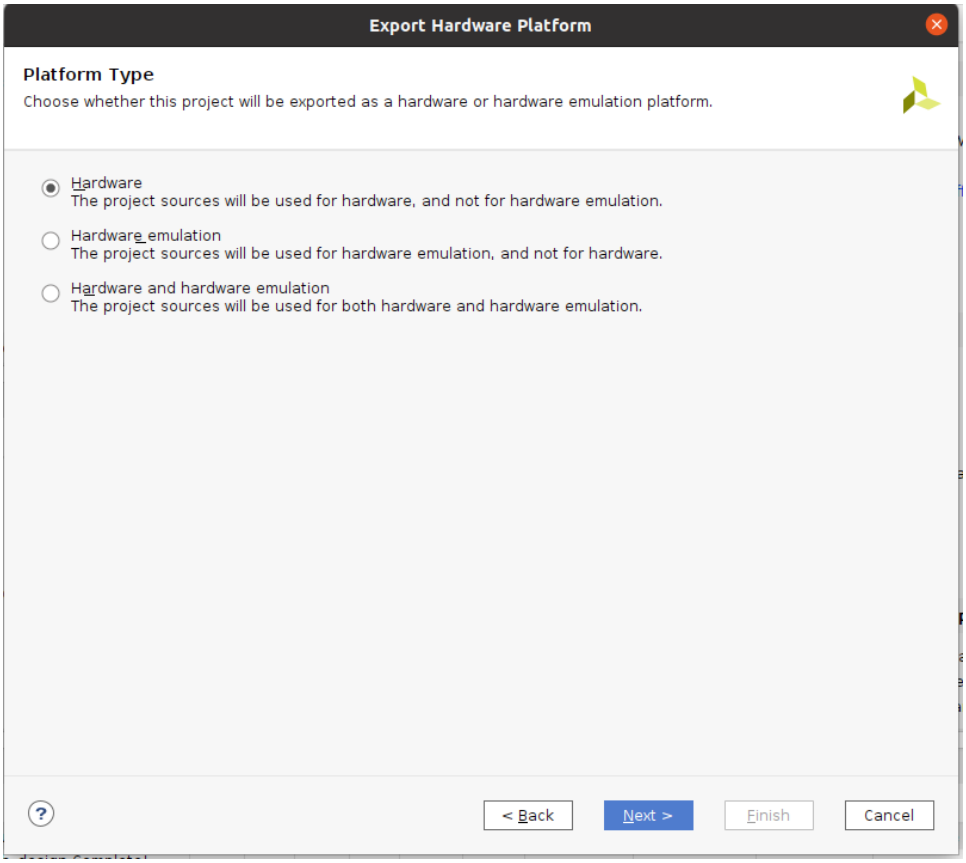
4. Export the platform

- Click menu File -> Export -> Export Platform to launch the Export Hardware

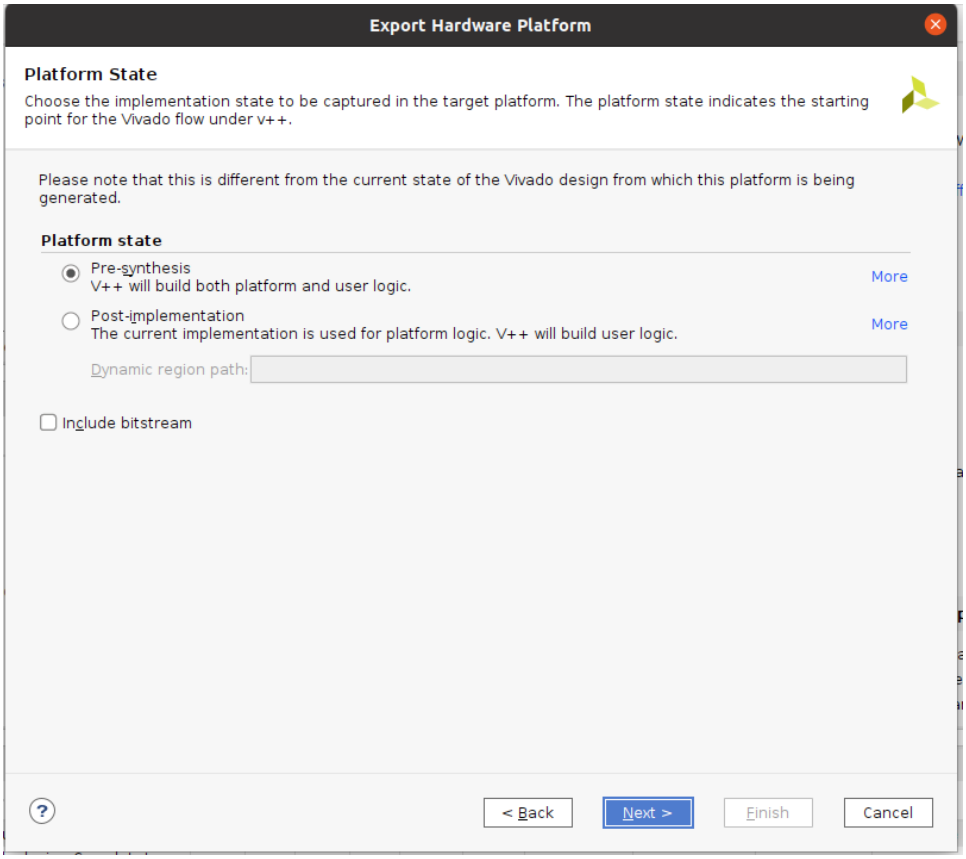


  Platform wizard. This wizard can also be launched by Export Platform button in Flow Navigator or Platform Setup window.
- Click Next in the first information page.

- Select Platform Type: Hardware, click Next. Select **Hardware** here.

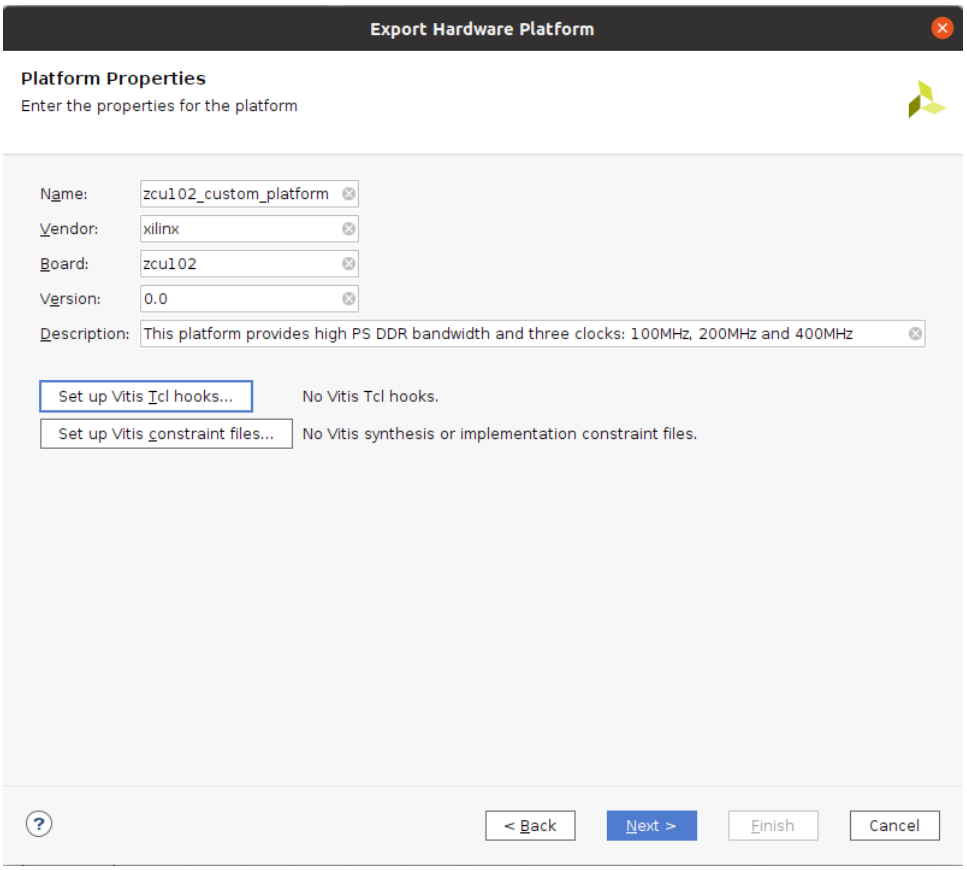


- Select Platform State: **Pre-synthesis**

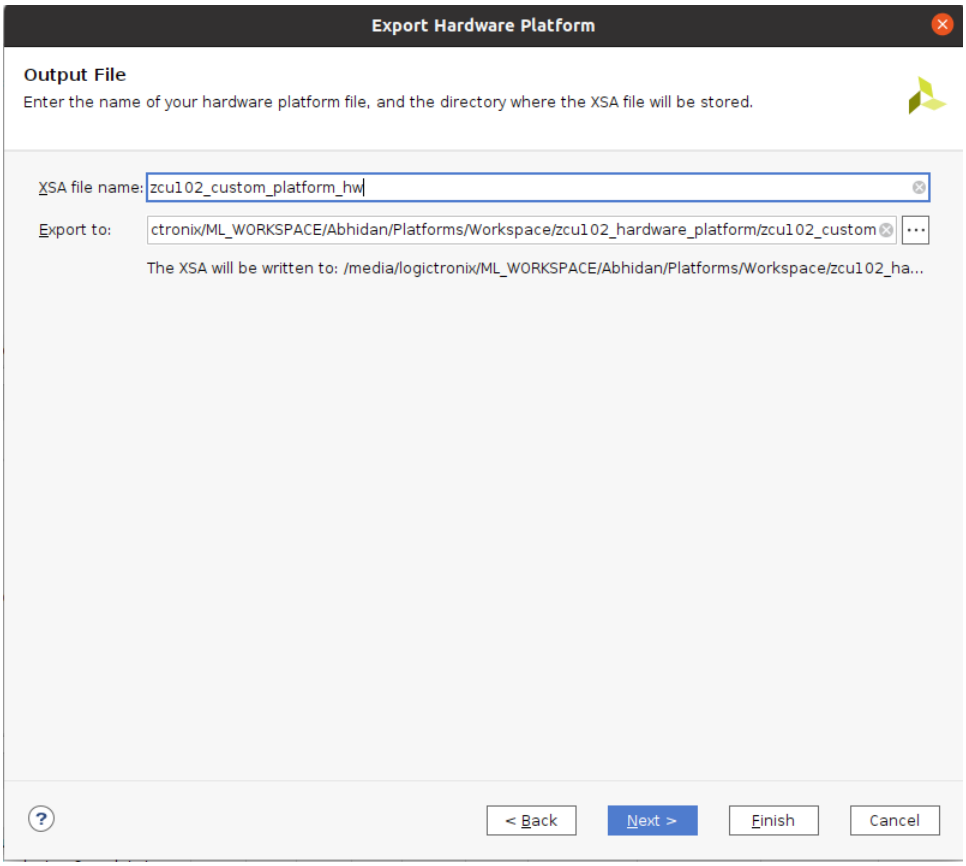


- Input Platform Properties and click Next. For example,
  - Name: zcu102_custom_platform
  - Vendor: xilinx
  - Board: zcu102
  - Version: 0.0

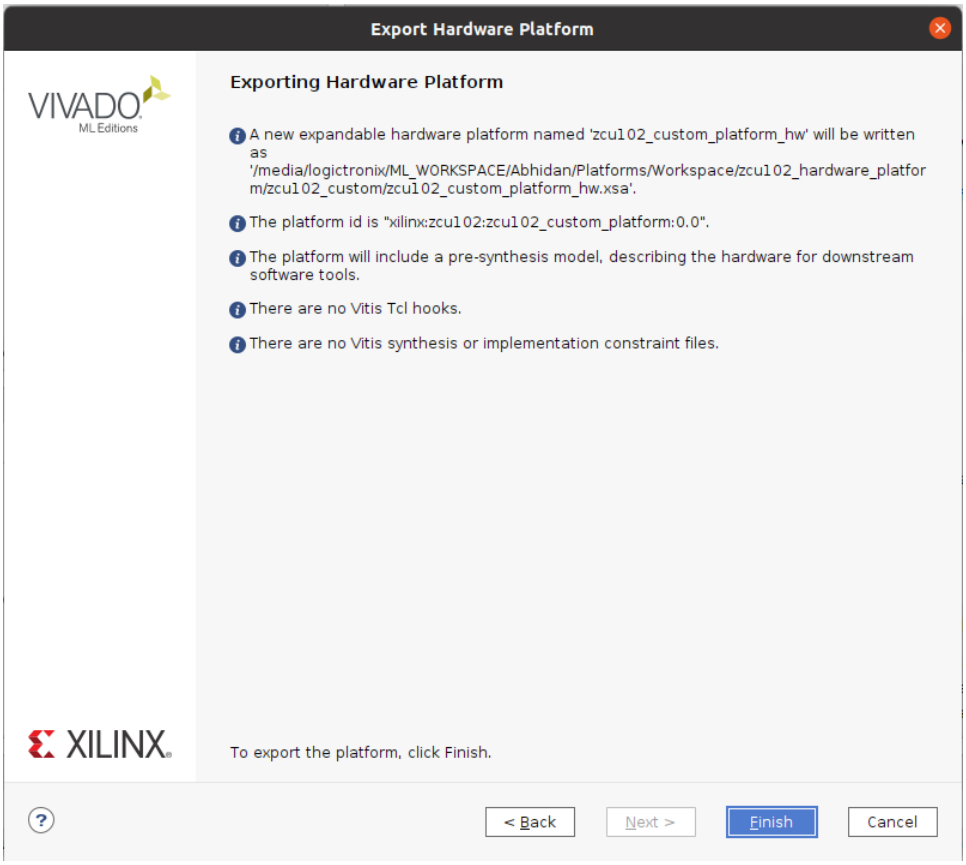
- Description: This platform provides high PS DDR bandwidth and three clocks: 100 MHz, 200MHz and 400 MHz.



- Fill in XSA file name: zcu102_custom_platform_hw and keep the export directory as default.



- Click Finish.



- zcu102_custom_platform _hw.xsa will be generated. The export path is reported in the TCL console.

Section 2: Creating Petalinux Build

In this section we will build platform software components. We'll use the PetaLinux tools to create the Linux image and sysroot with XRT support, together with some more advanced tweaks. For more information on PetaLinux Check: [UG1144]

Prepare the base platform

1. Create a workspace
    a. mkdir workspace
    b. cd workspace

Create a PetaLinux Project

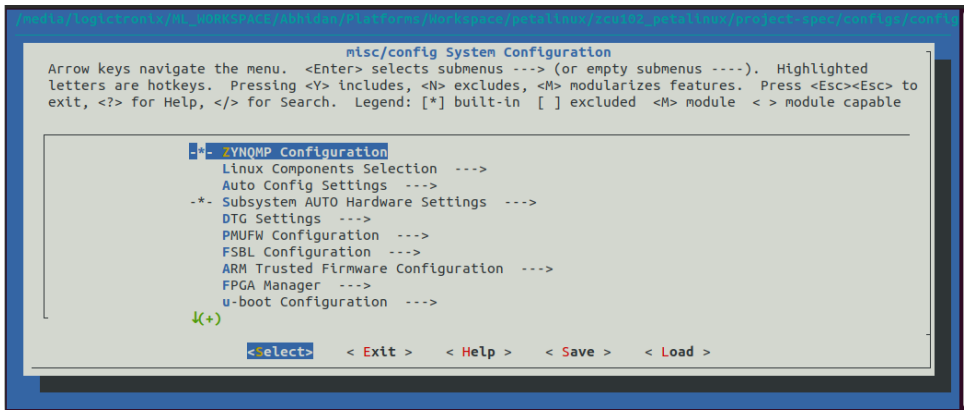
1. Setup PetaLinux environment
   Source <petalinux_tool_install_directory>/settings.sh
2. Create a PetaLinux project named zcu102_petalinux and configure the hw option with XSA file:

   cd workspace
   petalinux-create - -type project - - template zynqMP - -name zcu102_petalinux
   cd zcu102_petalinux
   petalinux-config –get-hw-description=xilinx_zcu102_base/hw/hw.xsa

   Note: - - template option specifies the chipset. zcu102 board adopts the ZYNQMP series chip. Therefore we specify this option as zynqMP.

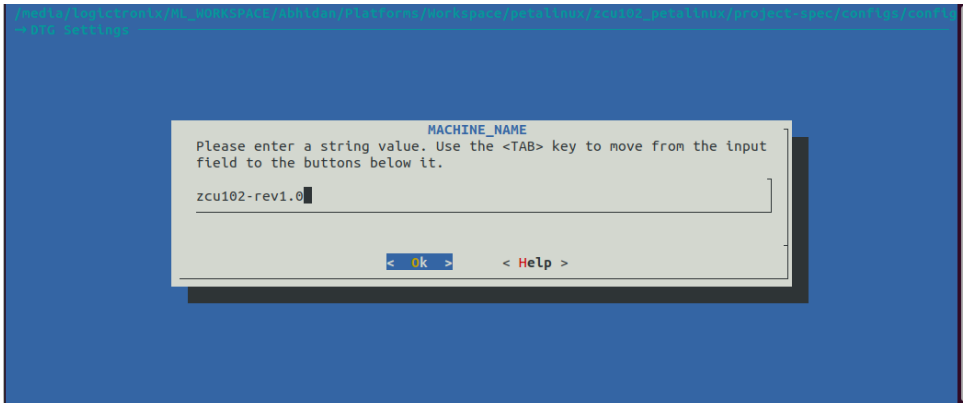
   Note: PetaLinux will use XSA to generate the device tree. Since hardware XSA and hardware emulation XSA have identical peripherals, giving either of them to PetaLinux makes no difference. When simplifying the hardware design for hardware emulation, it's recommended to keep all the peripherals that need device tree and drivers so that the auto-generated device tree can be reused. If the two design has different address-able peripherals, you will need to create two sets of device trees for hardware running and hardware emulation separately.

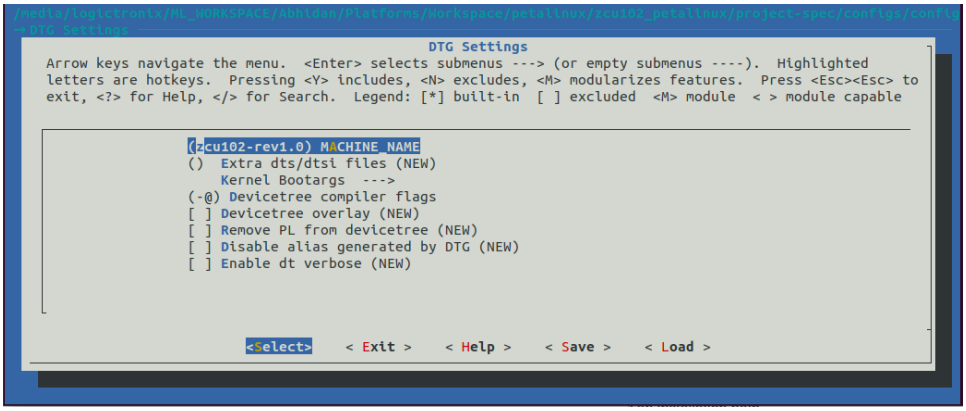
3. A petalinux-config menu would be launched, Set to use ZCU102 device tree in this configuration window.



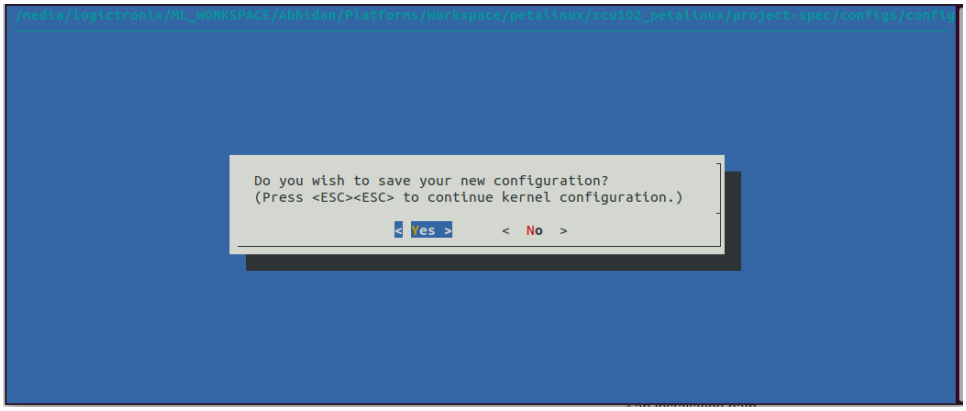
    ○ Select DTG Settings -> MACHINE_NAME

- Modify it to zcu102-rev1.0





- Select OK -> Exit -> Exit -> Yes to close this window.

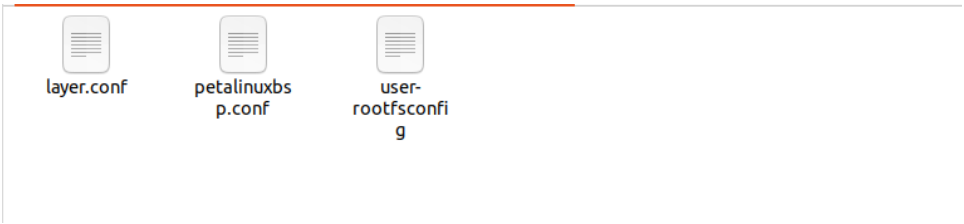


Note:

- If you are using a Xilinx development board it is recommended to modify the machine name so that the board configurations would be involved in the DTS auto-generation. Check UG1144 for corresponding machine name.

Customize Root File System, Kernel, Device Tree and U-boot

1. Add user packages
    - Append the CONFIG_x lines below to the <your_petalinux_project_dir>/project-spec/meta-user/conf/user-rootfsconfig file.



Packages for base XRT support:
CONFIG_xrt

- xrt is required for Vitis acceleration flow. The dependency packages such as ZOCL driver module will be added automatically.

Recommended Packages for easy system management

CONFIG_dnf
CONFIG_e2fsprogs-resize2fs
CONFIG_parted
CONFIG_resize-part

- dnf is for package management
- Parted, e2fsprogs-resize2fs and resize-part can be used for ext4 partition resize. We will use it to expand the ext4 partition to make full use of SD card when running Vitis-AI test case.

Packages for Vitis-AI dependencies support:
CONFIG_packagegroup-petalinux-vitisai

Optional Packages for natively building Vitis AI applications on target board:
CONFIG_packagegroup-petalinux-self-hosted
CONFIG_cmake
CONFIG_packagegroup-petalinux-vitisai-dev
CONFIG_xrt-dev
CONFIG_opencl-clhpp-dev
CONFIG_opencl-headers-dev
CONFIG_packagegroup-petalinux-opencv
CONFIG_packagegroup-petalinux-opencv-dev

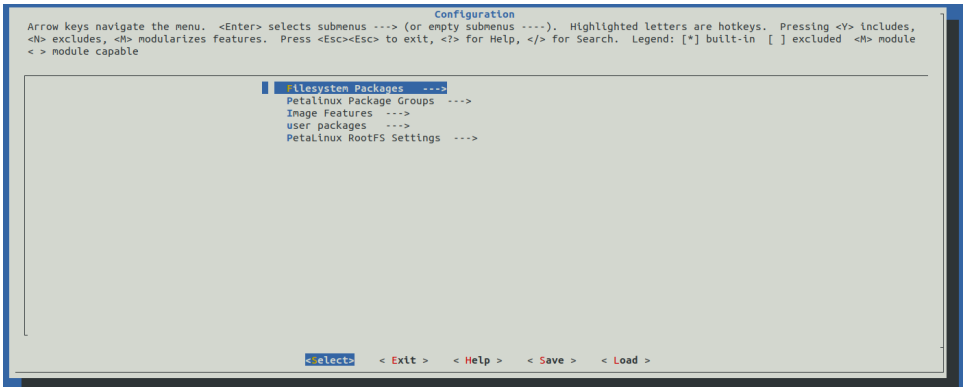
Optional Packages for running Vitis-AI demo applications with GUI
CONFIG_mesa-megadriver
CONFIG_packagegroup-petalinux-x11
CONFIG_packagegroup-petalinux-v4lutils
CONFIG_packagegroup-petalinux-matchbox

The user-rootfsconfig should contain the following packages in the end:

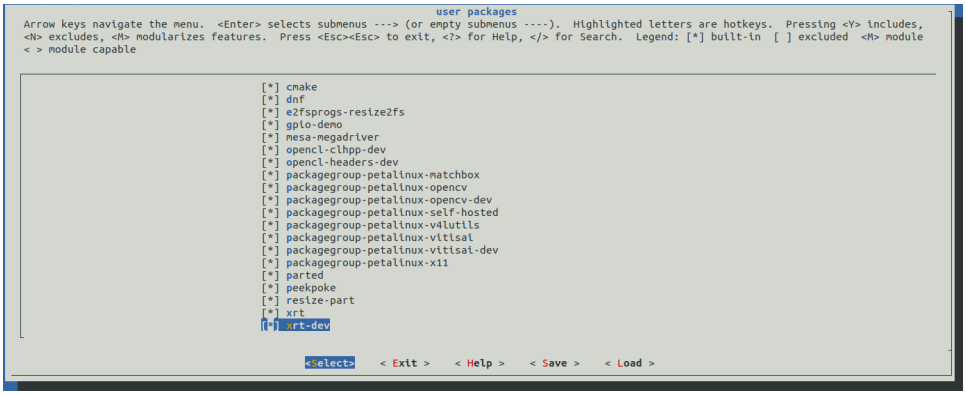
```
#Note: Mention Each package in individual line
#These packages will get added into rootfs menu entry

CONFIG_gpio-demo
CONFIG_peekpoke
CONFIG_xrt
CONFIG_dnf
CONFIG_e2fsprogs-resize2fs
CONFIG_parted
CONFIG_resize-part
CONFIG_packagegroup-petalinux-vitisai
CONFIG_packagegroup-petalinux-self-hosted
CONFIG_cmake
CONFIG_packagegroup-petalinux-vitisai-dev
CONFIG_xrt-dev
CONFIG_opencl-clhpp-dev
CONFIG_opencl-headers-dev
CONFIG_packagegroup-petalinux-opencv
CONFIG_packagegroup-petalinux-opencv-dev
CONFIG_mesa-megadriver
CONFIG_packagegroup-petalinux-x11
CONFIG_packagegroup-petalinux-v4lutils
CONFIG_packagegroup-petalinux-matchbox
```

2. Enable selected rootfs packages

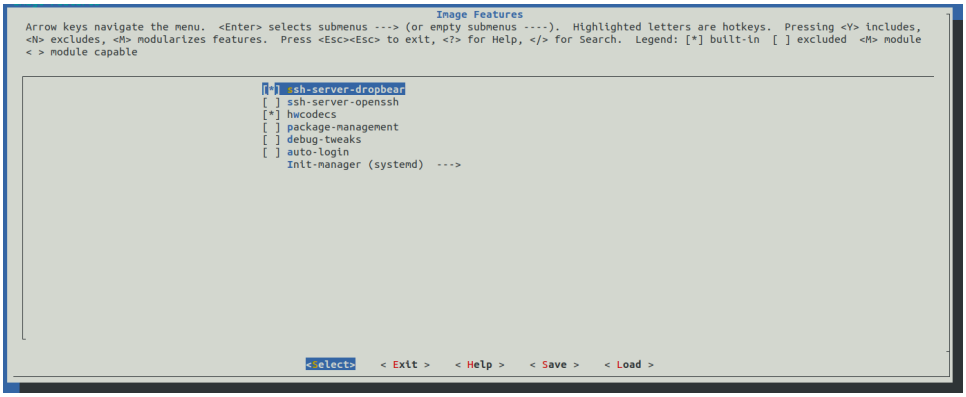
- Run petalinux-config -c rootfs



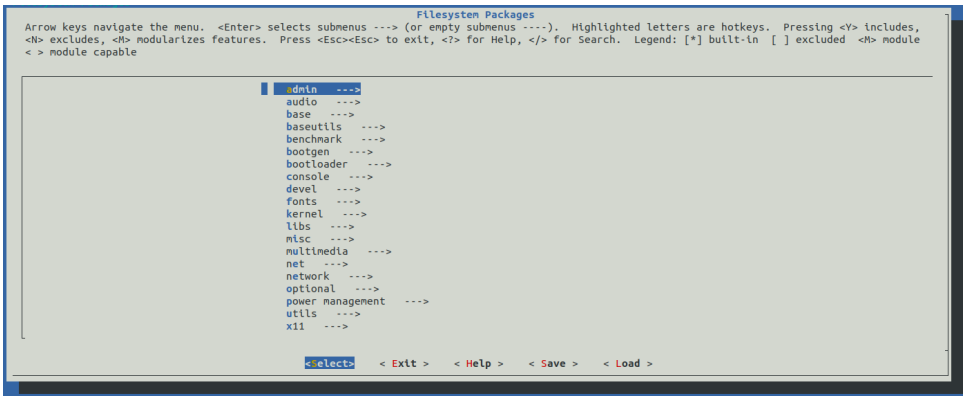
- Select User Packages
- Select name of rootfs all the libraries listed above.

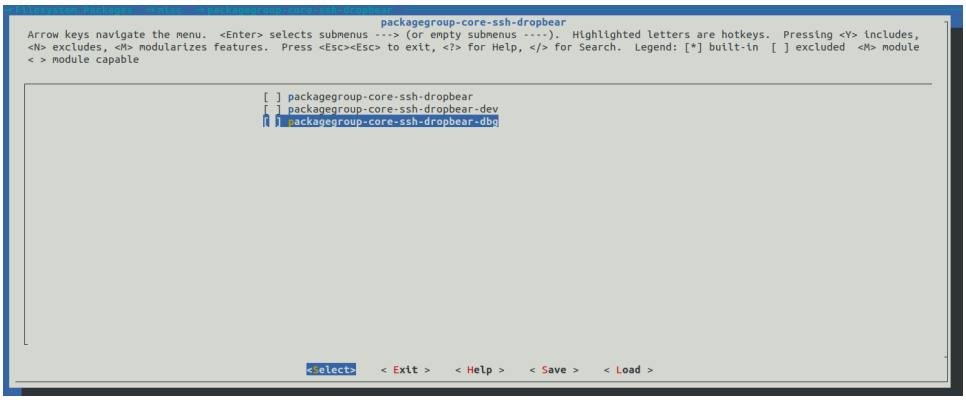


3. Enable OpenSSH and disable dropbear (this is an optional step)
   Dropbear is the default SSH tool in Vitis Base Embedded Platform. If OpenSSH is used to replace Dropbear, the system could achieve 4x times faster data transmission speed over ssh (tested on 1Gbps Ethernet environment). Since Vitis-AI applications may use remote display feature to show machine learning results, using OpenSSH can improve the display experience.
   - Still in the RootFS configuration window, go to root directory by select Exit once.
   - Go to Image Features
   - Disable ssh-server-dropbear and enable ssh-server-openssh and click Exit.

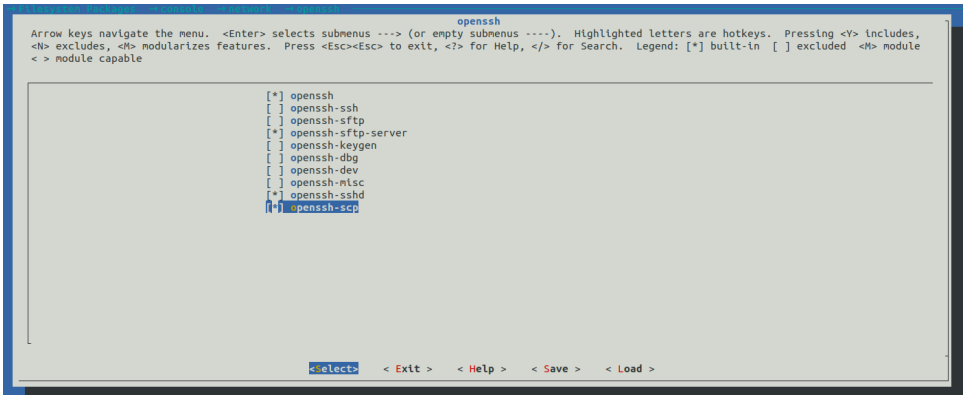


- Go to Filesystem Packages -> misc -> packagegroup-core-ssh-dropbear and disable packagegroup-core-ssh-dropbear.





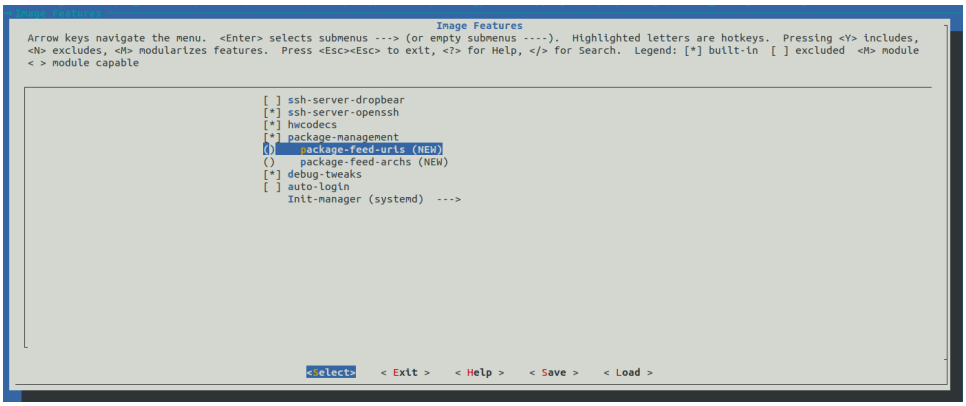
- Go to Filesystem Packages level by Exit twice
- Go to console->network->openssh and enable openssh, openssh-sftp-server, openssh-sshd, openssh-scp



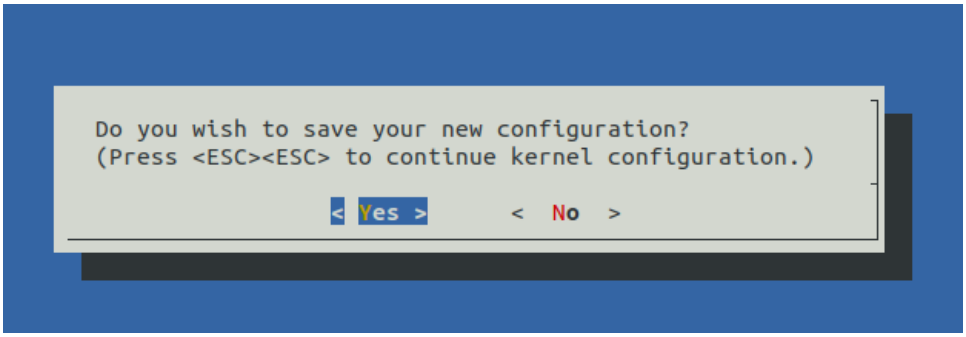
- Go to root level by Exit four times

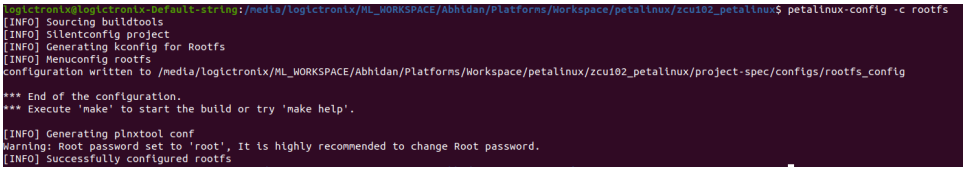
4. Enable Package Management
   Package management feature can allow the board to install and upgrade software packages on the fly.

- In rootfs config go to Image Features and enable package-managment and debug_tweaks option



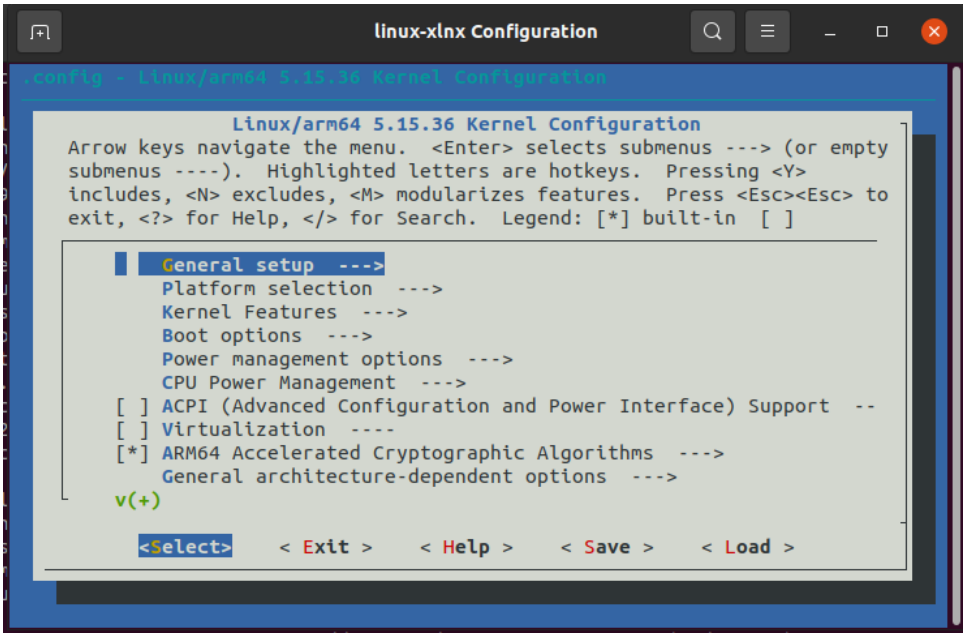
- Click OK, Exit twice and select Yes to save the changes.



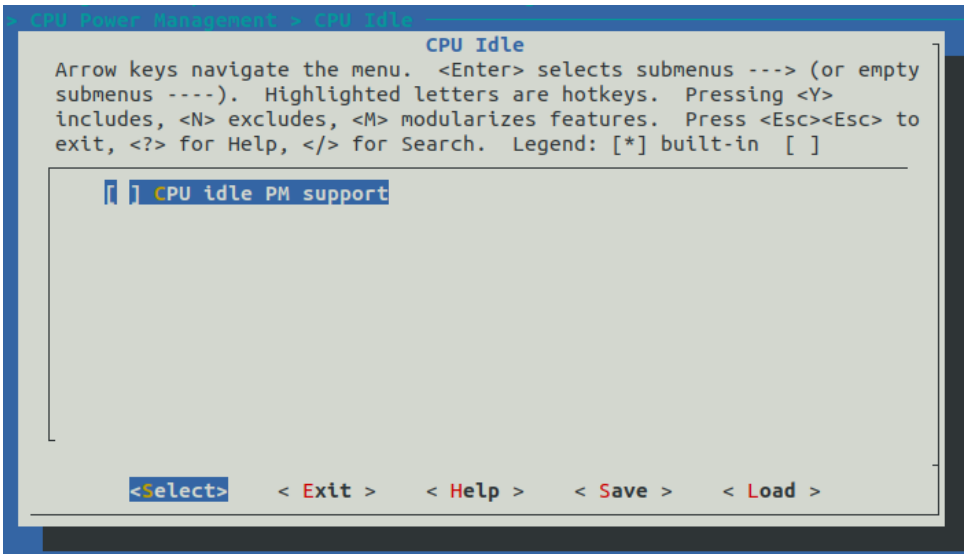


5. Disable CPU IDLE in kernel config (Recommended during debugging).
   CPU IDLE would cause processors get into IDLE state(WFI) when the processor is not in use. When JTAG is connect, the hardware server on host machine talks to the processor regularly. If it talks to a processor in IDLE status, the system will hang because of incomplete AXI transactions. So it is recommend to disable the CPU IDLE feature during project development phase. It can be re-enabled after the design has completed to save power in final products.

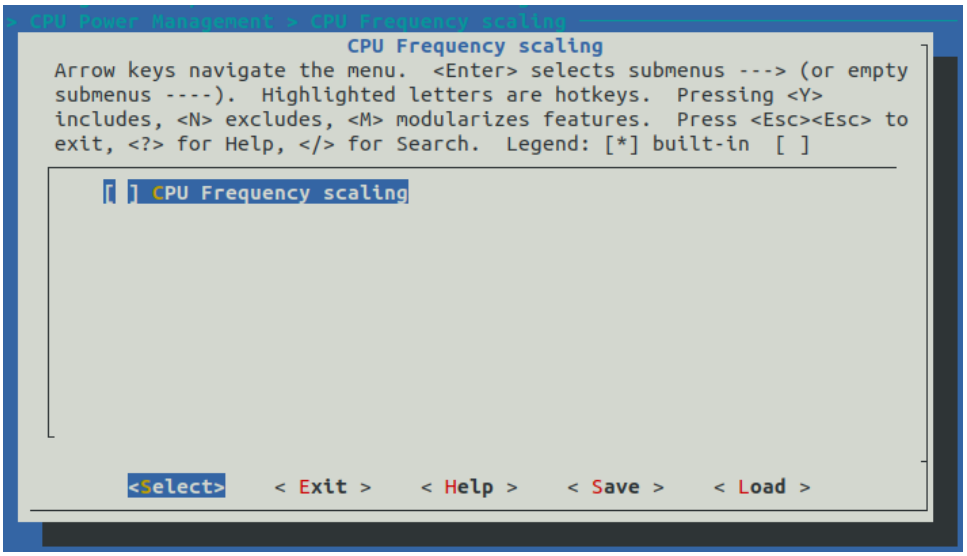
- Launch kernel config : petalinux-config -c kernel



- Ensure the following items are TURNED OFF by entering ‘n’ in the [ ] menu selection:
  - CPU Power Management > CPU Idle > CPU idle PM Support



- CPU Power Management > CPU Frequency scaling > CPU Frequency scaling



- Exit and Save.

Update the Device tree

Device tree describes the hardware components of the system. Xilinx device tree generator (DTG) can generate the device tree according to hardware configurations from XSA file. User needs to ad customization settings in system-user.dtsi for PetaLinux to consume if there are any settings not available in XSA, for example, any driver nodes that don't have a corresponding hardware, or if user need to override any DTG auto-generated configurations.

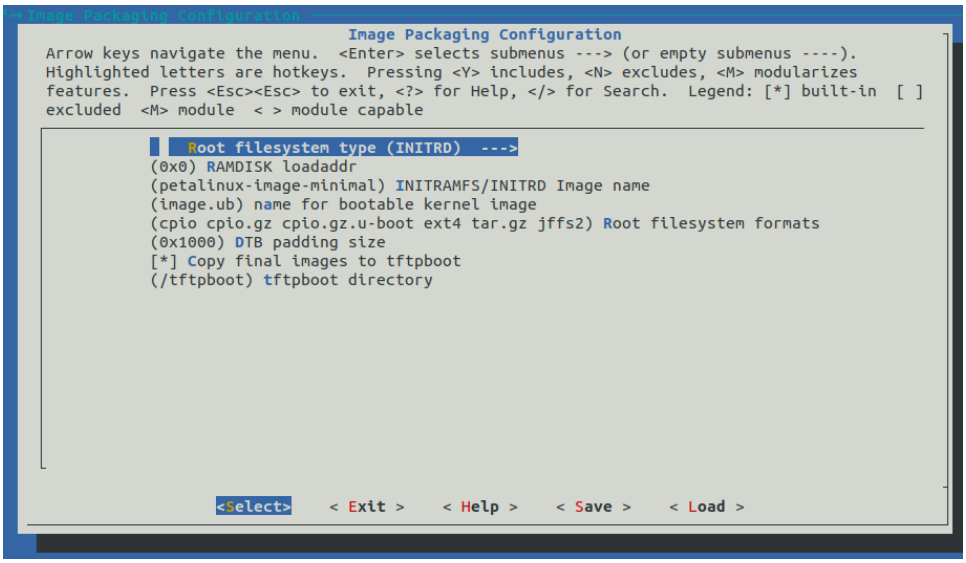
ZOCL driver module has no associated hardware, but it's required by XRT and Vitis acceleration flow. It requires a device tree node to describe the interrupt signal relationship. In previous Vitis and PetaLinux versions, users need to add ZOCL device tree node manually. From 2021.1, Petalinux can add ZOCL device tree node automatically if the XSA is a Vitis extensible platform project.

Device Tree Generator(DTG) also overrides the interrupt controller (axi_intc_0) input numbers parameter from 0 to 32 because in the platform XSA the interrupt controller inputs have not been connected but they will be connected after v++ links the acceleration kernels.

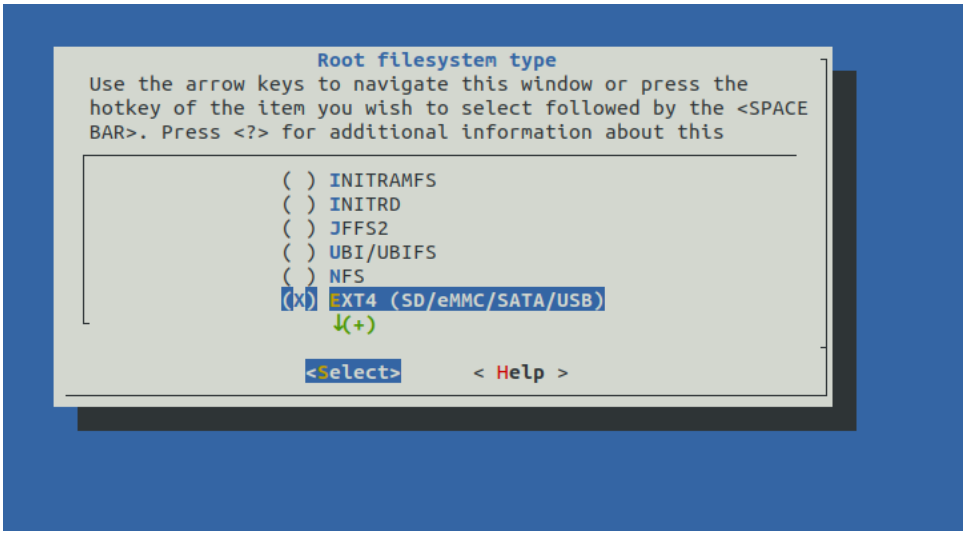
Add EXT4 rootfs support

It's recommended to use EXT4 for Vitis Acceleration designs. PetaLinux uses initramfs format for rootfs by default. It can't retain the rootfs changes in run time. Initramfs keeps rootfs contents in DDR, which makes user usable DDR memory reduced. To make the root file system retain changes and to enable maximum usage of available DDR memory, we'll use EXT4 format for rootfs in second partition while keep the first partition FAT32 to store the boot files.

1. Let PetaLinux generate EXT4 rootfs
    - Run petalinux-config
    - Go to Image Packaging Configuration

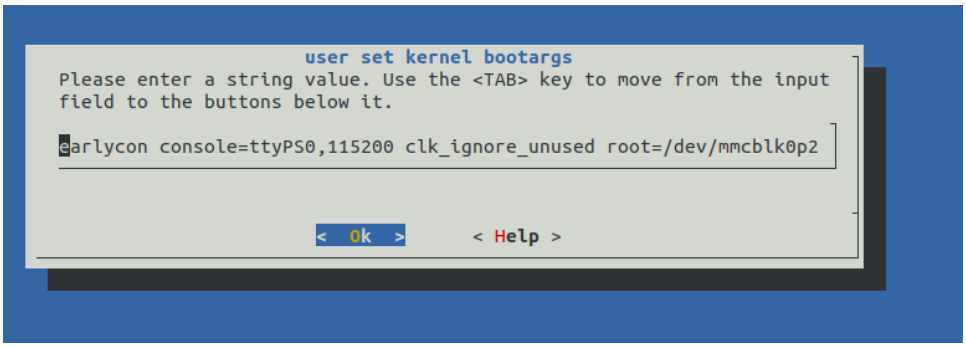


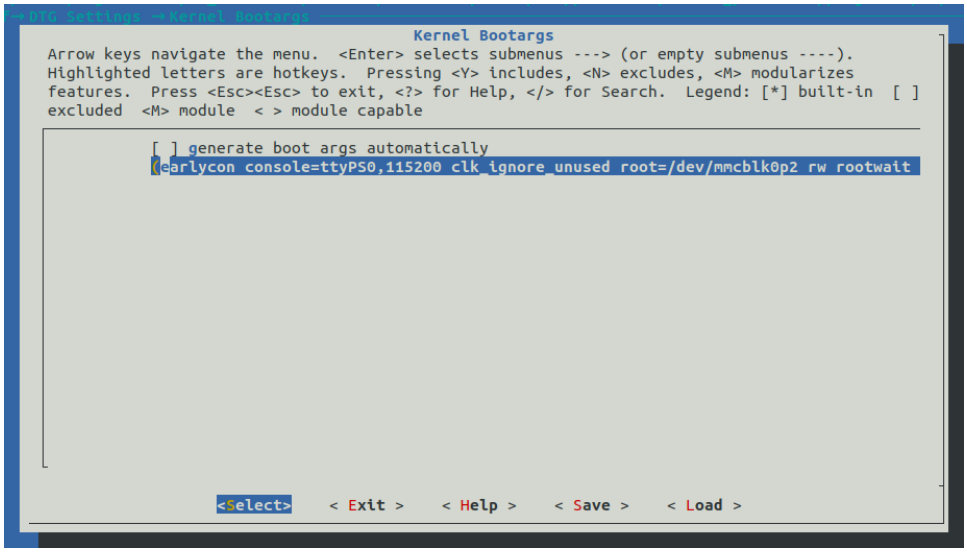
- Enter into Root File System Type
- Select Root File System Type as EXT4



- Exit and Save

2. Let Linux use EXT4 rootfs during boot
   The setting of which rootfs to use during boot is controlled by bootargs. We would change bootargs settings to allow Linux to boot from EXT4 partition.
   - Run petalinux-config
   - Change **DTG settings -> Kernel Bootargs -> generate boot argos automatically** to NO and update **User Set Kernel Bootargs** to *earlycon console=ttyPS0, 115200 clk_ignore_unused root=/dev/mmcblk0p2 rw rootwait cma=512M.* Click OK, Exit three time and Save.





Note:

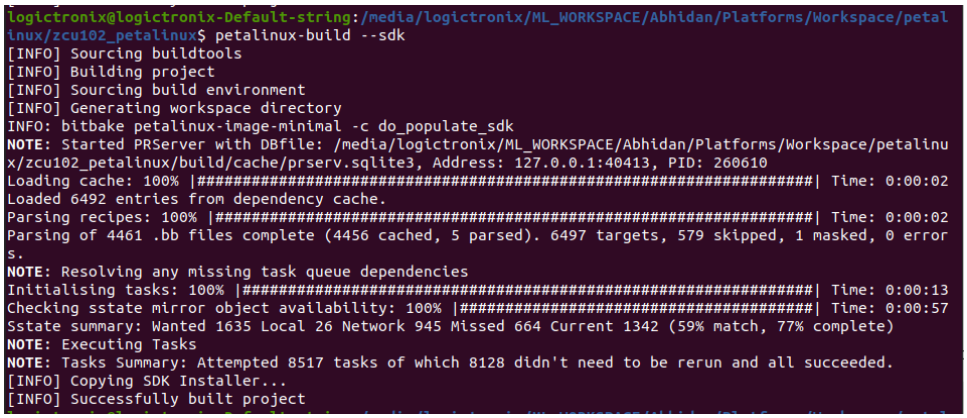
- root=/dev/mmcblk0p2 means to use second partition of SD card, which is the EXT4 partition.
- Please note that we also set these options in bootargs:
    - clk_ignore _unused: it tells Linux kernel don't turn off clocks if this clock is not used. It's useful clocks that only drives PL kernels are not represented in device tree.
    - cma=512M: CMA is used to exchange data between PS and PL kernel. The size for CMA is determined by PL kernel requirements. Vitis-AI/DPU needs at least 512MB CMA.

Build Petalinux Images

1. From any directory within the PetaLinux Project, build the PetaLinux project.
   petalinux-build
   The Petalinux Image files will be generated in /images/linux directory.

2. Create a sysroot self-installer for the target Linux system
   petalinux-build - - sdk

The generated sysroot package sdk.sh will be located in /images/linux directory. You can extract it when you need.



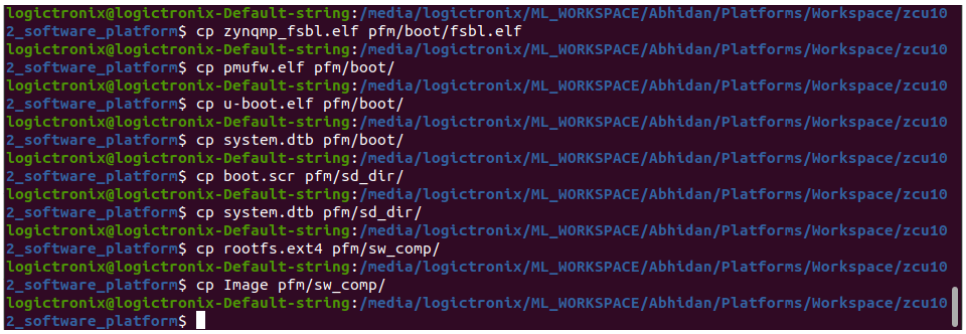
Section 3: Adding DPU kernel onto the base Platform

1. First we will create four directories: pfm, boot, sd_dir, sw_comp to store the components and copy files to these directories.
   cd workspace/
   mkdir zcu102_software_platform
   cd zcu102_software_platform
   mkdir pfm
   mkdir pfm/boot
   mkdir pfm/sd_dir
   mkdir pfm/sw_comp

   Copy the following files from <petalinux-project>/images/linux directory to zcu102_software_platform
   - bl31.elf
   - boot.scr
   - fsbl.elf
   - Image
   - pmufw.elf
   - rootfs.ext4
   - rootfs.manifest
   - rootfs.tar.gz
   - system.dtb
   - u-boot.elf

   Copy fsbl.elf, pmufw.elf, bl31.elf, u-boot.elf and system.dtb to pfm/boot/ directory
   Copy boot.scr and system.dtb to pfm/sd_dir/ directory

Copy rootfs.ext4 and Image to pfm/sw_comp directory



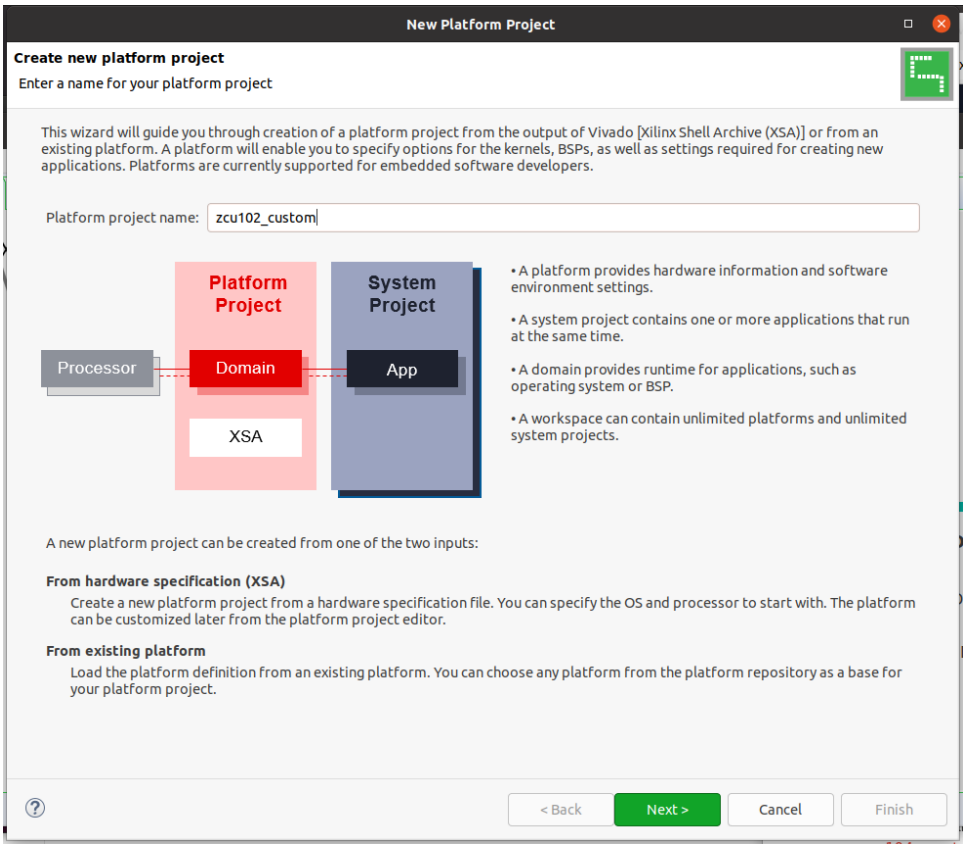
Note: The files fsbl.elf, pmufw.elf, bl31.elf, u-boot.elf, and system.dtb located in the boot directory serve as the source for creating the BOOT.BIN image. The files boot.scr and system.dtb in the sd_dir directory are used for the initialization of u-boot and the booting up of Linux. These files will be packaged into the FAT32 partition by the v++ packaging tool. The files Image and rootfs.ext4 are the Linux kernel and root filesystem, respectively. These will also be packaged into the sd.img file by the v++ tool.

2. Install the sysroot
    - In our previous step, we executed the command `petalinux-build - - sdk` to create a file named `sdk.sh`. This file is located within the `<petalinux-project>/images/linux` directory.
    - To install the PetaLinux SDK, use the command `./sdk.sh-d`. The `-d` option allows you to specify a full pathname for the output directory. Make sure to confirm your input.
    - Please remember that when running this command, the environment variable `LD_LIBRARY_PATH` should not be set.

3. Create Vitis Platform
    - cd workspace
    - source <Vitis_tool_install_dir>/settings64.sh
    - Launch Vitis by typing ***vitis &*** in the console.

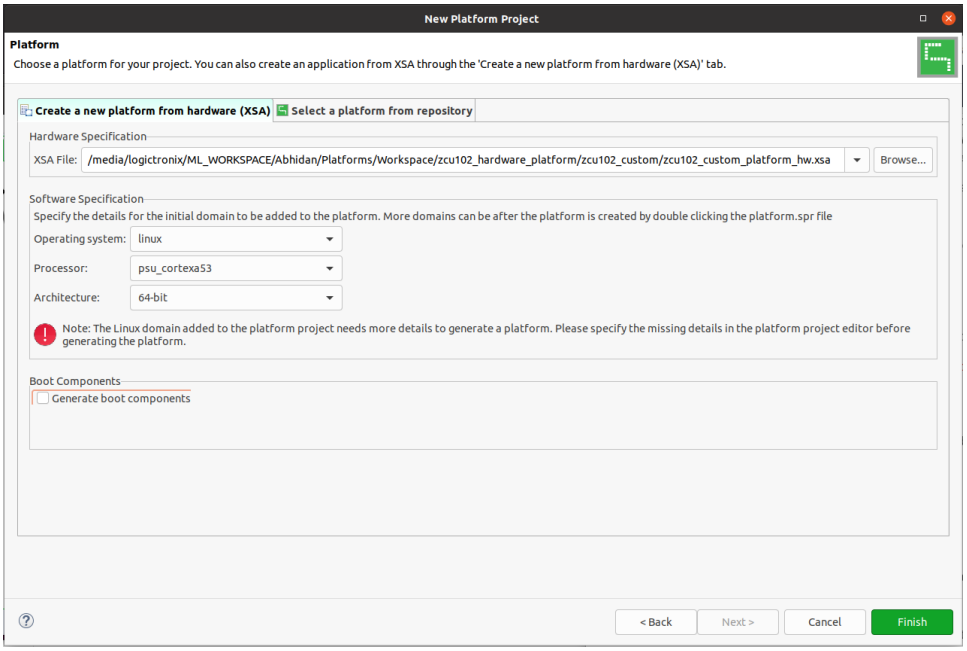
```
2_dpu_vitis$ vitis &
[1] 639903
logictronix@logictronix-Default-string:/media/logictronix/ML_WORKSPACE/Abhidan/Platforms/Workspace/zcu10
2_dpu_vitis$
****** Xilinx Vitis Development Environment
****** Vitis v2022.2 (64-bit)
  **** SW Build 3671529 on 2022-10-13-17:52:08
    ** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.

Launching Vitis with command /media/logictronix/ML_WORKSPACE/Xilinx/Vitis/2022.2/eclipse/lnx64.o/eclipse
 -vmargs -Xms64m -Xmx1024m -Dorg.eclipse.swt.internal.gtk.cairoGraphics=false -Dosgi.configuration.area=
@user.home/.Xilinx/Vitis/2022.2 --add-modules=ALL-SYSTEM --add-opens=java.base/java.nio=ALL-UNNAMED --ad
d-opens=java.desktop/sun.swing=ALL-UNNAMED --add-opens=java.desktop/javax.swing=ALL-UNNAMED --add-opens=
java.desktop/javax.swing.tree=ALL-UNNAMED --add-opens=java.desktop/javax.swing.plaf.basic=ALL-UNNAMED --
add-opens=java.desktop/javax.swing.plaf.synth=ALL-UNNAMED --add-opens=java.desktop/com.sun.awt=ALL-UNNAM
ED --add-opens=java.desktop/sun.awt.X11=ALL-UNNAMED &

[1]+  Done                    vitis
logictronix@logictronix-Default-string:/media/logictronix/ML_WORKSPACE/Abhidan/Platforms/Workspace/zcu10
2_dpu_vitis$
```

- Create a zcu102_dpu_vitis folder as workspace directory
- Select menu File -> New -> Platform Project to create a platform project.
- Enter the project name. For this example, type zcu102_custom. Click Next.



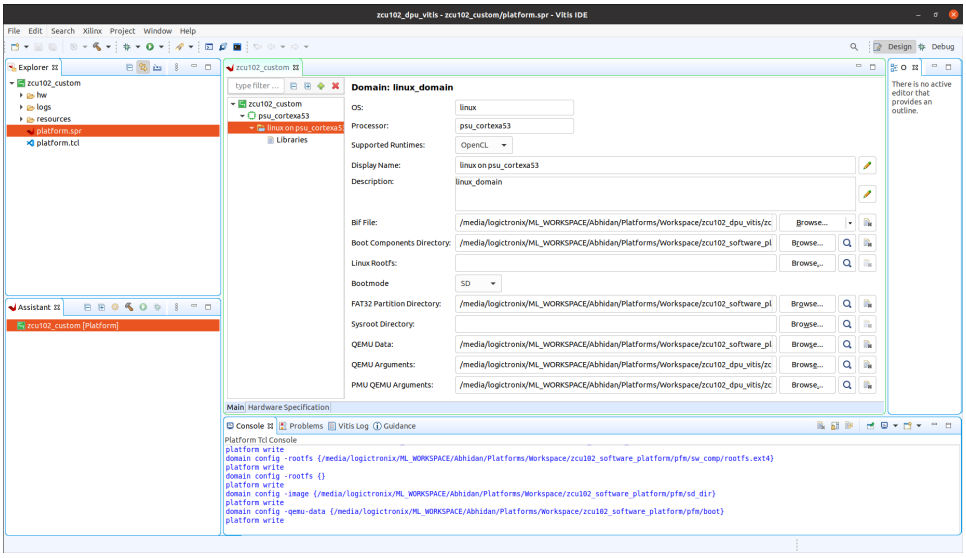
- In the Platform page,
  - Click Browse button, select XSA file generated by the Vivado. In our case, it is zcu102_custom_platform_hw.xsa
  - Set the operating system to linux.
  - Set the processor to psu_cortexa53
  - Architecture: 64-bit
  - uncheck option *Generate boot components*, because already have fsbl and pmu from petalinux project.

- Click Finish.

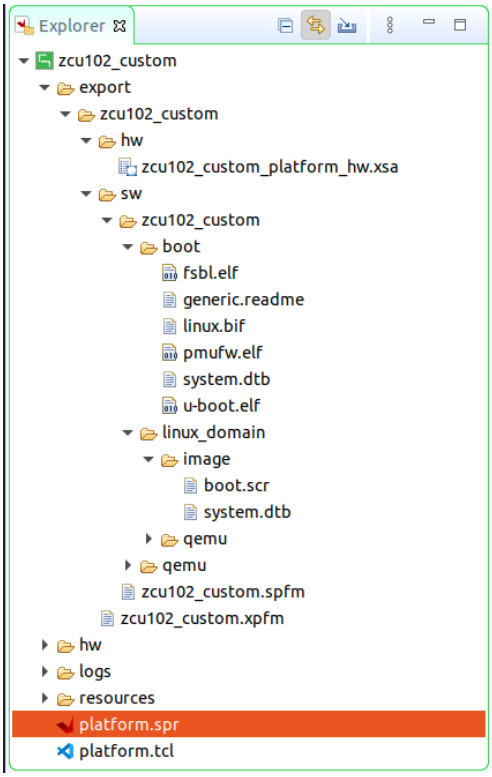


4. Set up the software settings in Platform Settings view
    - Click the linux on psu_cortexa53 domain, browse to the locations and select the directory or file needed to complete the dialog box for the following:
    - Bif file: Click the drop-down icon and select Generate BIF
        - Note: The file names in <> are placeholders in bif file. Vitis will replace the placeholders with the relative path to platform during platform packaging. V++ packager, which runs when building the final application would expand it further to the full path during image packaging. Filename placeholders point to the files in boot components directory. The filenames in boot directory need to match with placeholders in BIF file. <bitstream> is reserved keyword. V++ packager will replace it with the final system bit file.
    - Boot Components Directory: Browse to zcu102_software_platform/pfm/boot and click OK.
    - FAT32 Partition Directory: Browse to zcu102_software_platform/pfm/sd_dir and click OK.

- QEMU Data: Browse to zcu102_software_platform/pfm/boot and click OK.



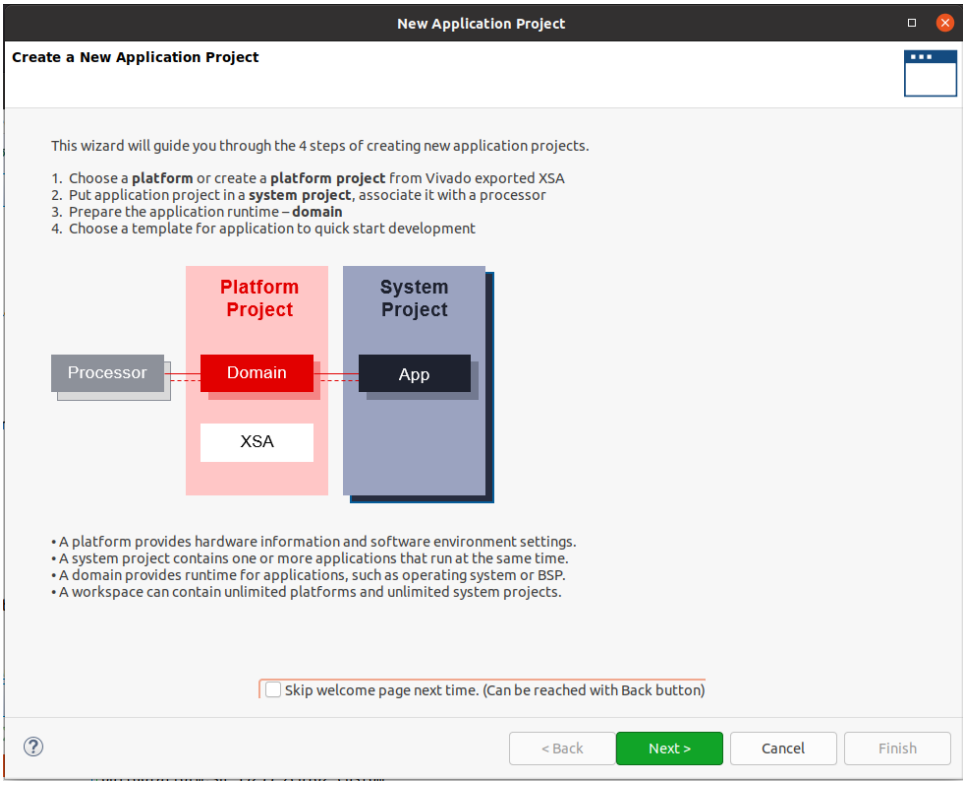
5. Click zcu102_custom project in the Vitis Explorer view, click the Build button to build the platform.
    - Note: The generated platform is placed in the export directory. BSP and source files are also provided for rebuilding the FSBL and PMU if desired and are associated with the platform. The platform is ready to be used for application development.



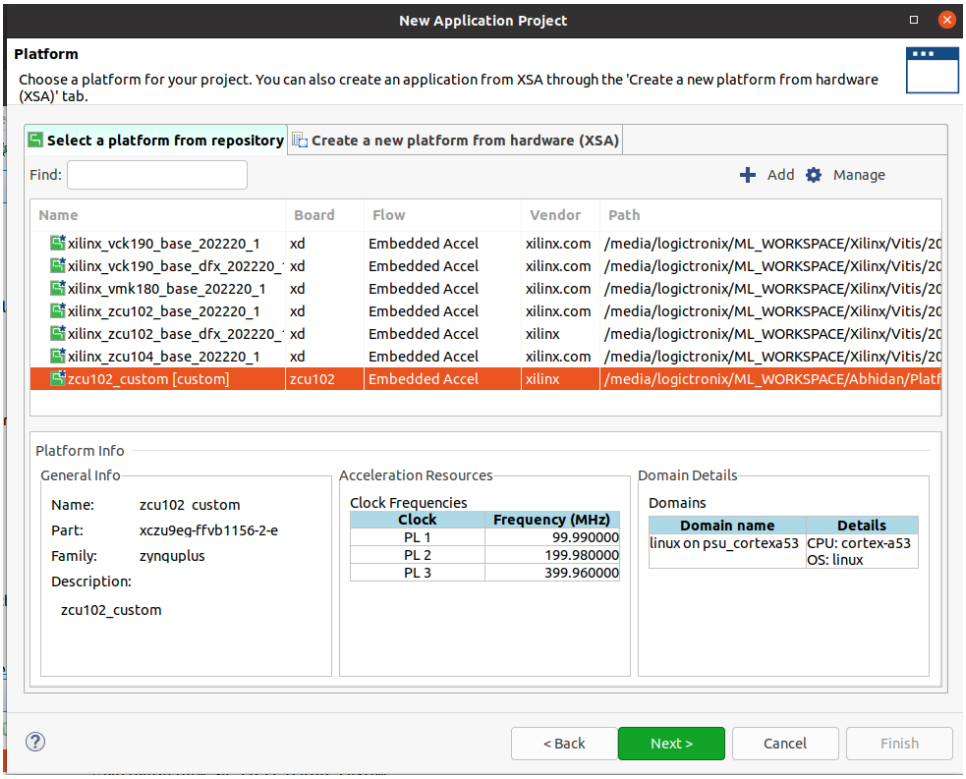
6. Add Vitis-AI repository into Vitis IDE
    - Open menu Window -> Preferences

- Goto Library Repository tab
- Add Vitis-AI
  - Click Add button
  - Input ID: vitis-ai
  - Name: Vitis AI
  - Location: assign a target download directory or keep empty. Vitis will use default path ~/.Xilinx if this field is empty.
  - Git URL: https://github.com/Xilinx/Vitis-AI.git
  - Branch: The branch you'd like to verify with your platform. Use master for the Vitis-AI version that matches Vitis 2022.2. You can use master for the latest patched version. Please note that the master branch will move forward. At some point master branch will point to a new release that will not be compatible with Vitis 2022.2

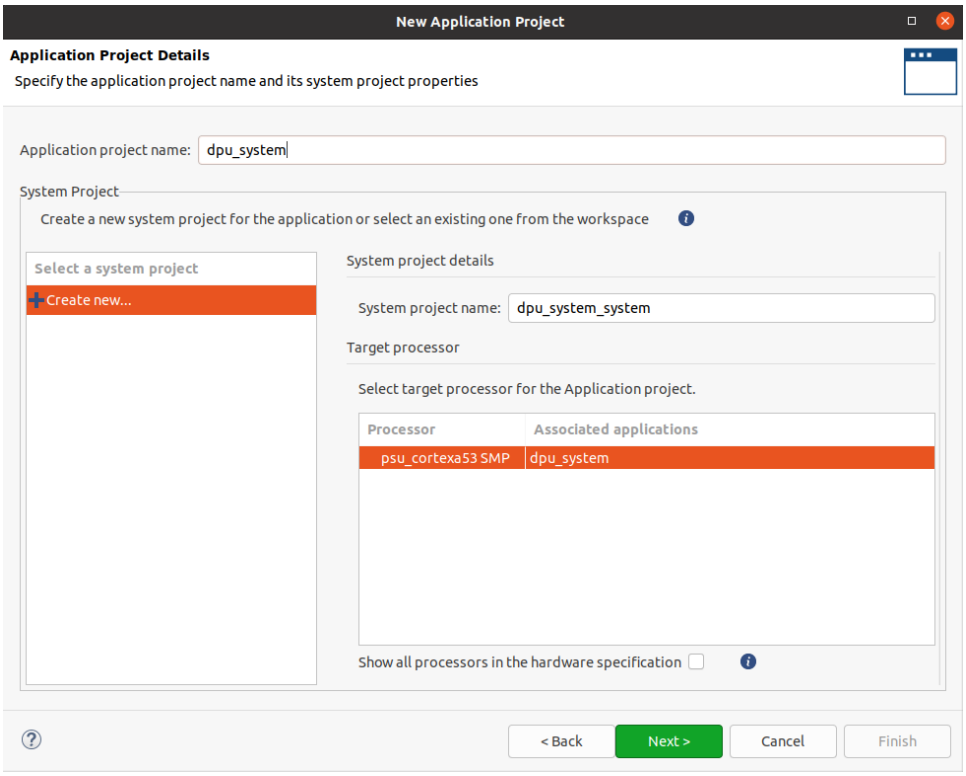
7. Download the Vitis-AI Library
   - Open menu Xilinx -> Libraries
   - Find the Vitis-AI entry we just added. Click the Download button on it.
   - Wait until the download of Vitis-AI repository completes
   - Click OK to close this window.
   - Vitis IDE will check the upstream status of each repository. If there are updates, it will allow users to download the updates if the source URL is a remote Git repository.
8. Create a Vitis-AI design on our zcu102_custom platform.
   - Go to menu File -> New -> Application Project
   - Click Next in Welcome Page



- Select platform zcu102_custom. Click Next.

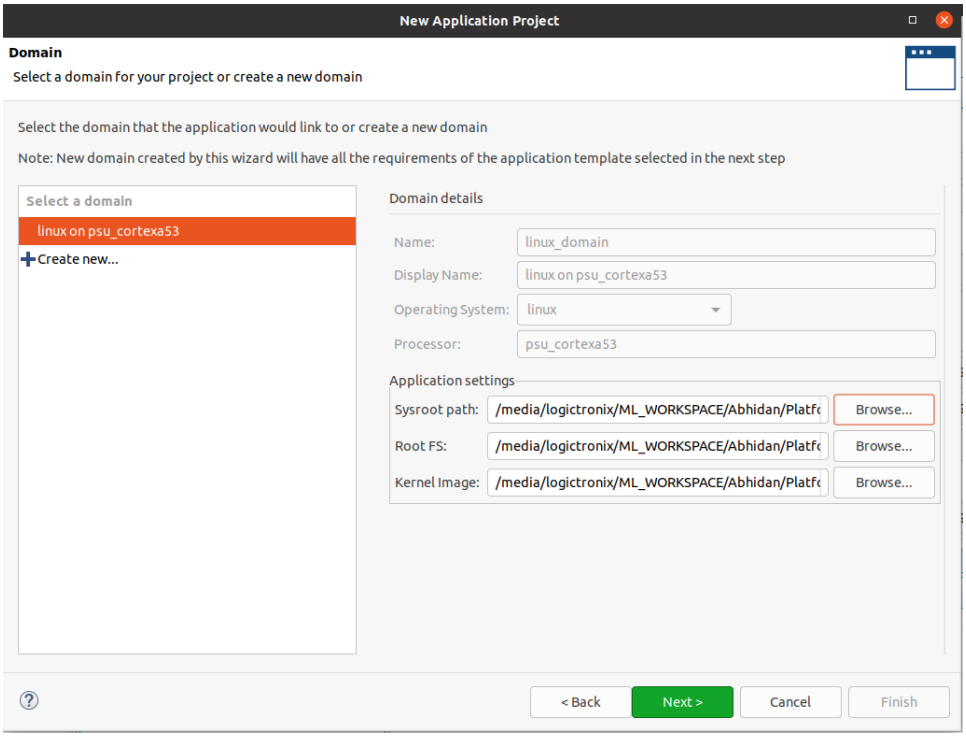


- Name the project dpu_system, click next.

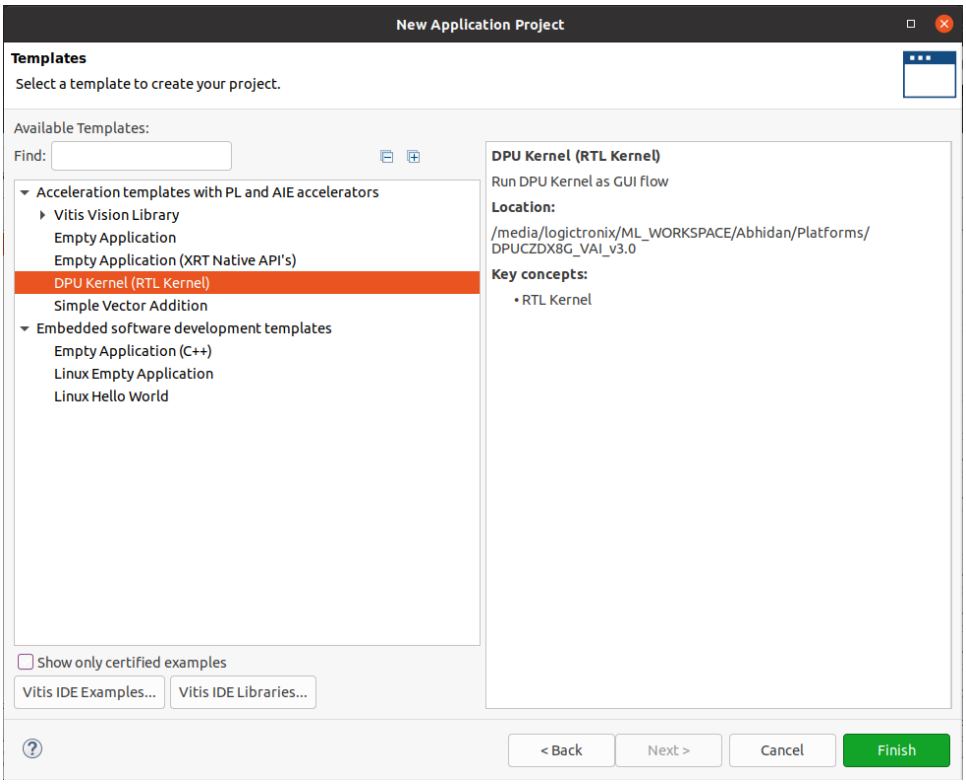


- Set Domain to linux on psu_cortex53, set Sysroot path to sysroot installation path in previous step.

- Set the Root FS to rootfs.ext4 and Kernel Image to Image. These files located in zcu102_software_platform/sw_comp directory, which are generated in previous section. Click Next.

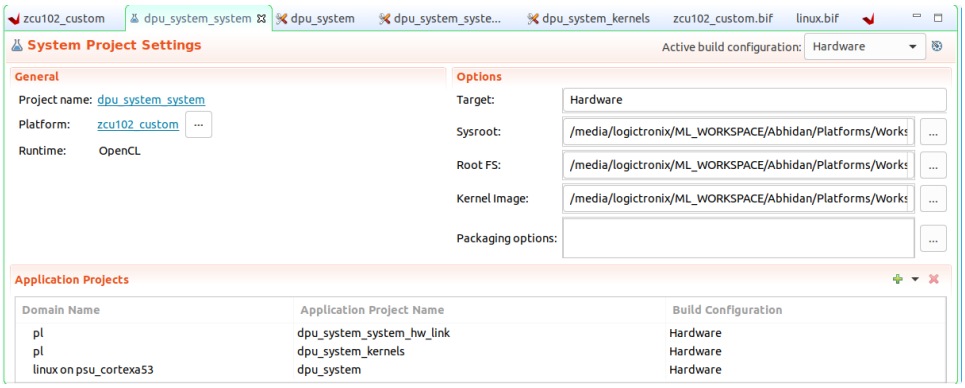


- Select dsa -> DPU Kernel(RTL kernel) click Finish to generate the application



9. Update Build Target

- Double click the system project file dpu_system_system.sprj
- Change Active Build Configuration to Hardware



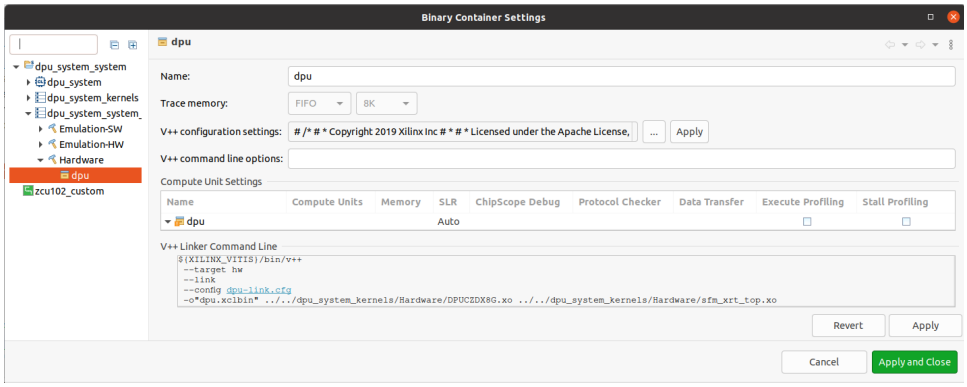
You can make changes to the DPU settings for your specific requirement or else no changes are required. The default created design has the DPU settings for ZCU102

10. Review system_hw_link v++ for proper kernel instantiation

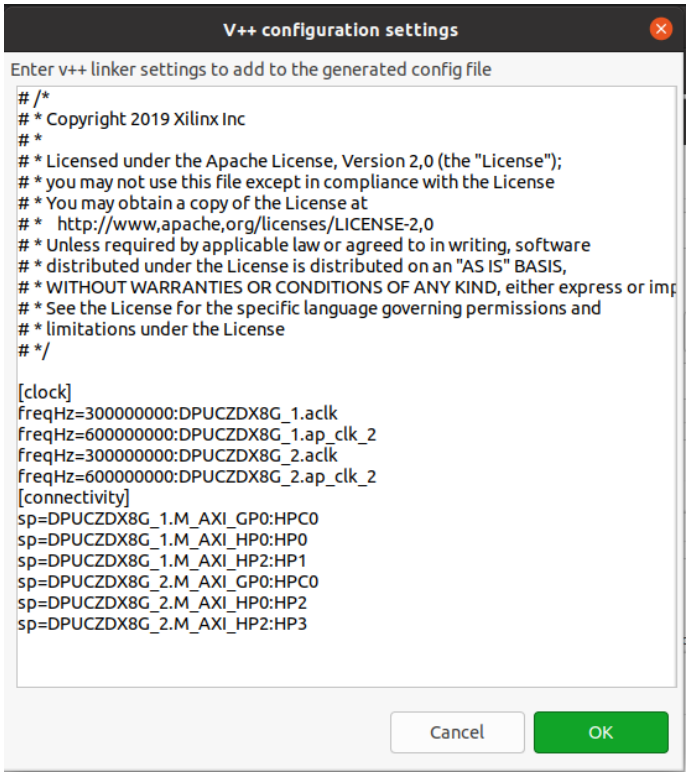
    The DPU kernel requires two phase aligned clocks, 1x clock and 2x clock. The configuration is stored in the example design. It sets up clock and AXI interface connections between the DPU kernel to the platform.

    Here's how to review the setup in the project.

    - Go to **Assistant View**
    - Double click **dpu_system_system [System]**
    - Expand the left tree panel and find dpu_system_system -> dpu_system_system_hw_link -> Hardware -> dpu



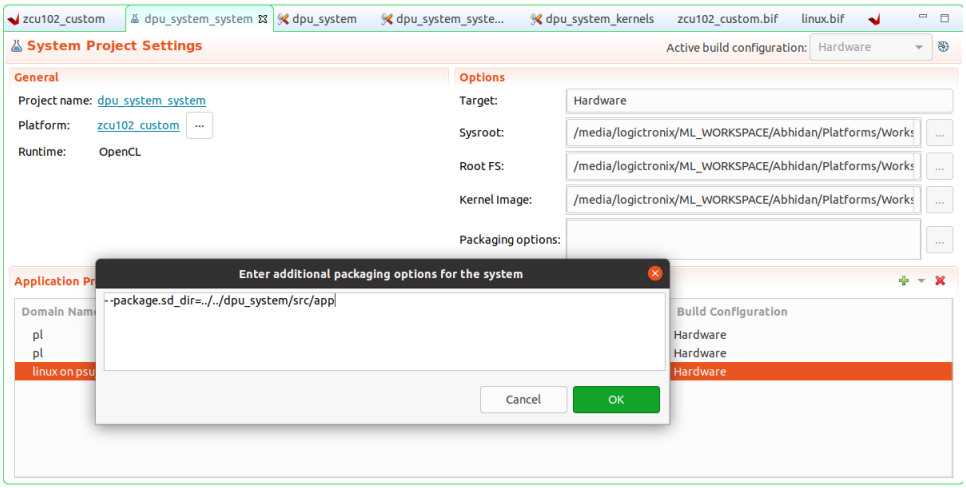
11. Click . . . button on the line of v++ configuration settings, it shows the configuration like this:



Note: the contents will be written to dpu-link.cfg during build time and pass to v++ Linker command line.

Note: To customize the v++ link configuration, you can add contents in v++ configuration settings, or create your own configuration file and add - -config <your_config_file.cfg> to v++ Command Line Options field. If you need to use relative path for the configuration file, the base location is dpu_system_system_hw_link/Hardware directory.

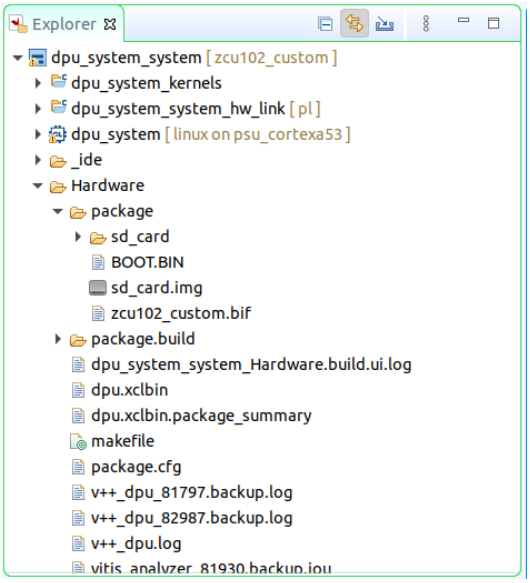
12. Update package options to add dependency models into SD Card
    - Double click dpu_system_system.sprj
    - Click . . . button on Package options
    - Input - -package.sd_dir=../../dpu_system/src/app
    - Click OK



All contents in the - -package.sd_dir assigned directory will be added to the FAT32 partition of the sd_card.img. We package samples and models for verification.

The dpu_trd in the path name is the application project name in this example. If your project name is different, please update the project name accordingly.

1. Build the hardware design
    - Select the dpu_system_system system project
    - Click the hammer button to build the system project
    - The generated SD card image is located at dpu_system_system/Hardware/package/sd_card.img.



This completes the build. We can burn the sd_card.img onto the sdcard and bootup.