Xilinx Answer 72076
Example design with PL-PCIe Root Port in ZCU106 and PS-PCIe Endpoint in UltraZed

Important Note: This downloadable PDF of an Answer Record is provided to enhance its usability and readability. It is important to note that Answer Records are Web-based content that are frequently updated as new information becomes available. You are reminded to visit the Xilinx Technical Support Website and review (Xilinx Answer 72076) for the latest version of this Answer.

ZCU106 Root Complex Design in Vivado

Overview

This document shows how to configure the Zynq ZCU106 as root complex with PL-PCIe using Vivado and PS-PCIe in UltraZed as an endpoint. This design will further be exported to the PetaLinux environment where the image files required to boot Linux on the Zynq ZCU106 device will be generated. There are 5 chronological steps required to generate an image file.

1. Configuring subsystems from the hardware description file for boot-up
2. Configuring the root filesystem component to enable the “lspci” command of PCIe
3. Generating the default Linux kernel component
4. Building the project from the configured components
5. Packaging the project together with the bitstream.

If successful, an image.ub file and a BOOT.BIN file which are needed to boot Linux will be generated for hardware testing.

Design Creation in IP Integrator

IPI stands for Intellectual Property Integrator. The design method for the ZCU106 root complex will utilize the “Create Block Design” tool under IPI which is located in the Flow Navigator window. This block design window allows the user to create a design using various IP blocks depending on the selected part or board.

Design Overview

Figure 1 shows the block design for the Zynq ZCU106 evaluation board with PCI Express set up as root complex

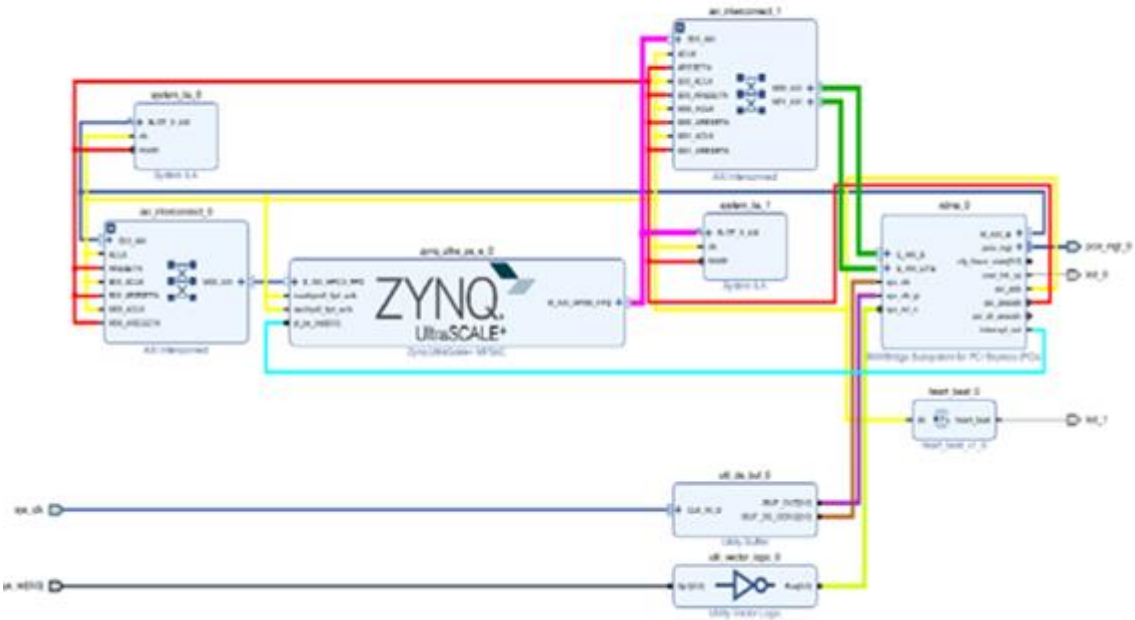


Figure 1 - Overall Vivado design

Create a new Vivado project

Open Vivado 2018.2 and from the welcome screen, click “Create Project” as shown in Figure 2 - Create project.

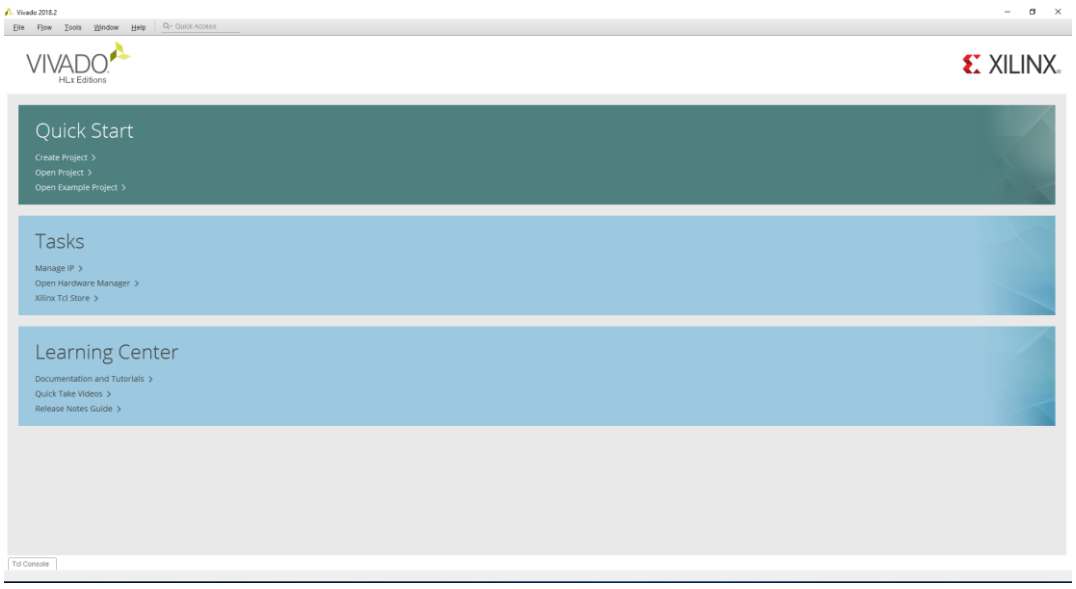


Figure 2 - Create project

Click “Next”.

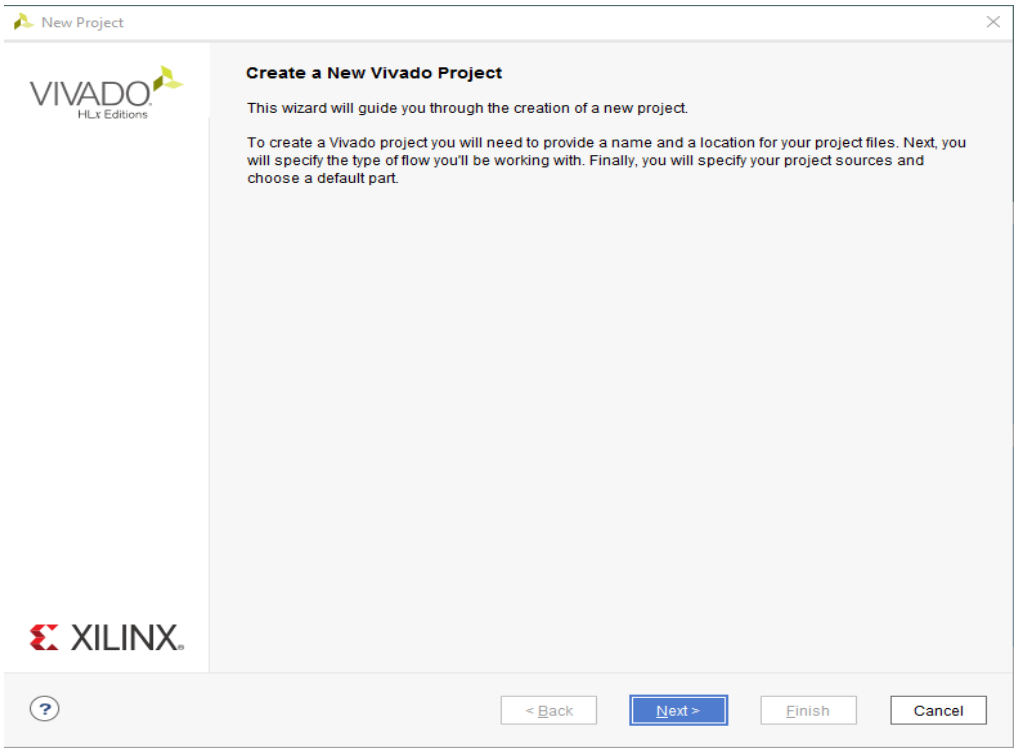


Figure 3 - Create project

Enter a desired name for the project and specify a project location where the project files will be stored. Select the “Create a project subdirectory” checkbox as shown below. Click “Next”.

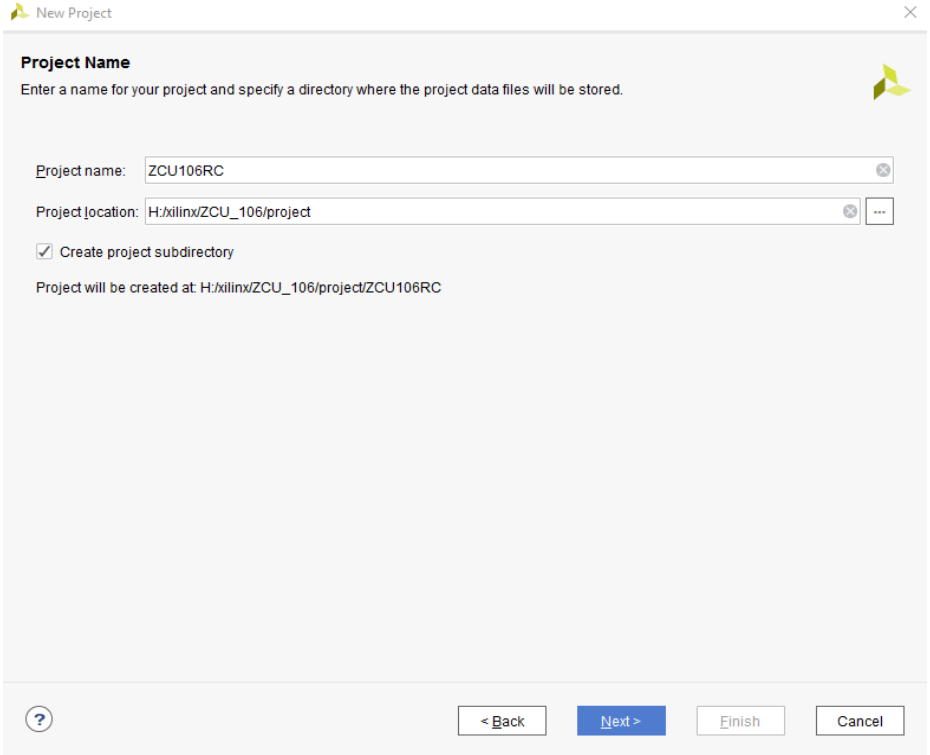


Figure 4 - Choose project name

In the Project Type window, select the “RTL Project” radio button and tick “Do not specify sources at this time”. RTL is short for Register Transfer Level. Click “Next”.

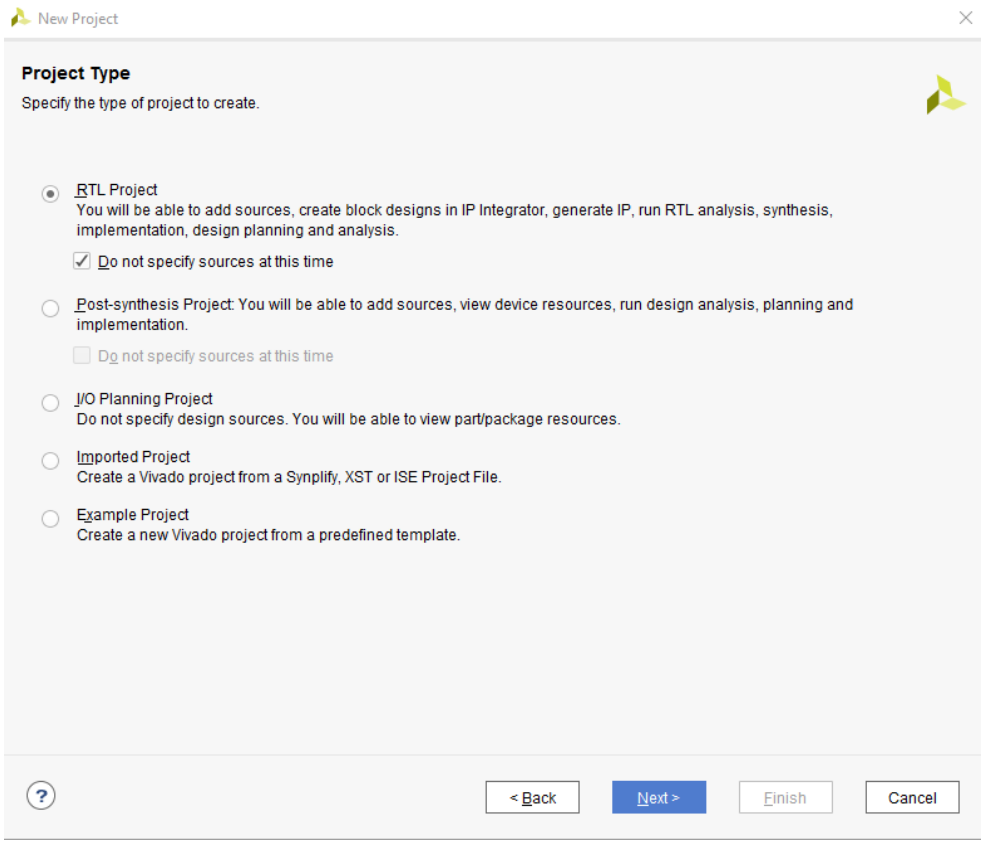


Figure 5 - Select project type

In the Default Part window, click on the “Boards” tab and type the board name, ZCU106, into the search field as shown in Figure 6. Select the board and click “Next”.

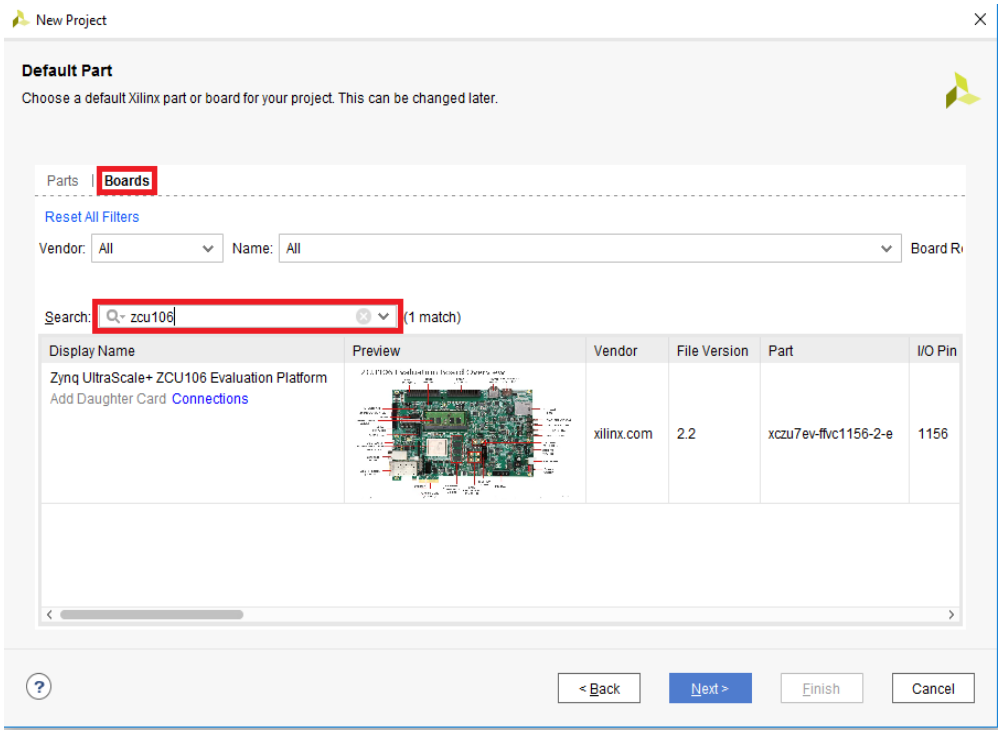


Figure 6 - Select board for the project

The resulting dialog box will display a summary of the project which includes the board information. Click “Finish” to create the project.

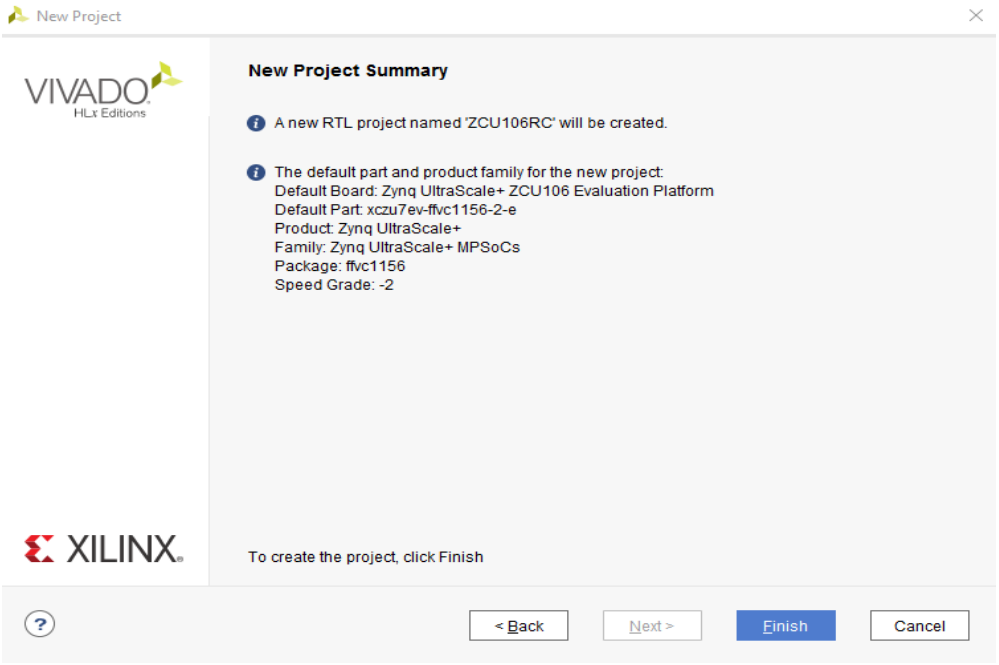


Figure 7 - Project summary

Add heartbeat Verilog module

Under "Sources" right-click on Design Sources and select "Add Sources..." from the drop-down list as shown in the figure below. Alternatively, click on "Add Sources" under the Flow Navigator pane which is located on the left part of the Vivado interface.

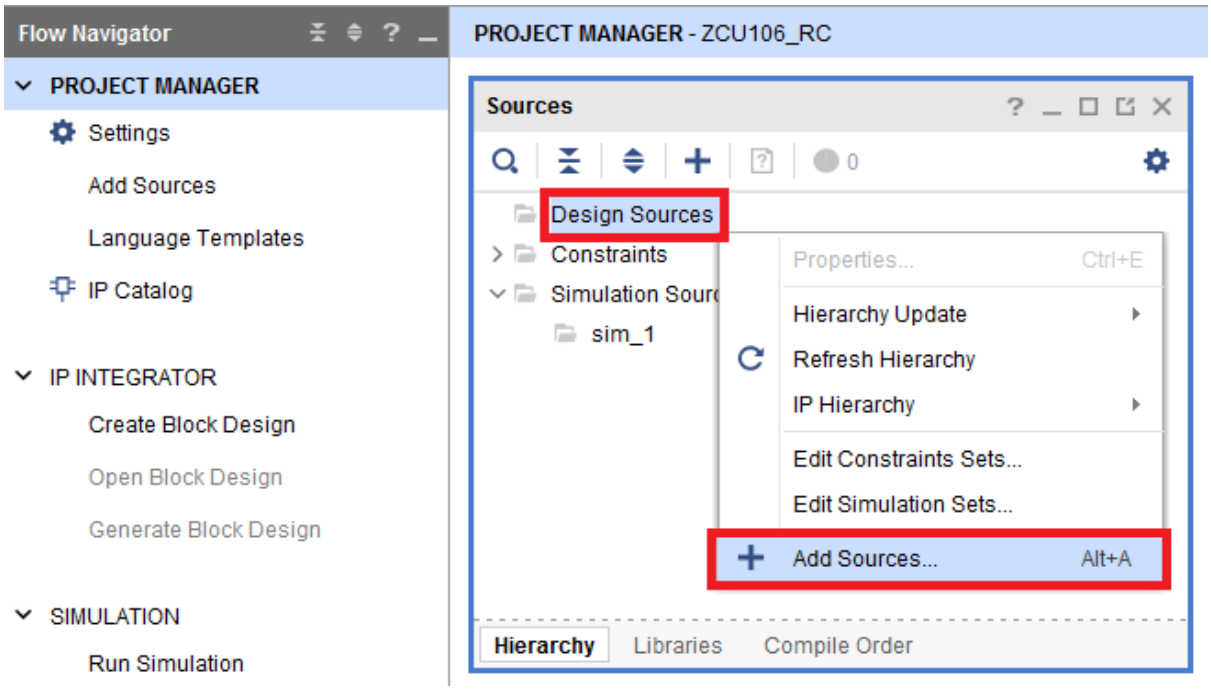


Figure 8 - Add a Verilog module

Select the “Add or create design sources” radio button and click “Next”.

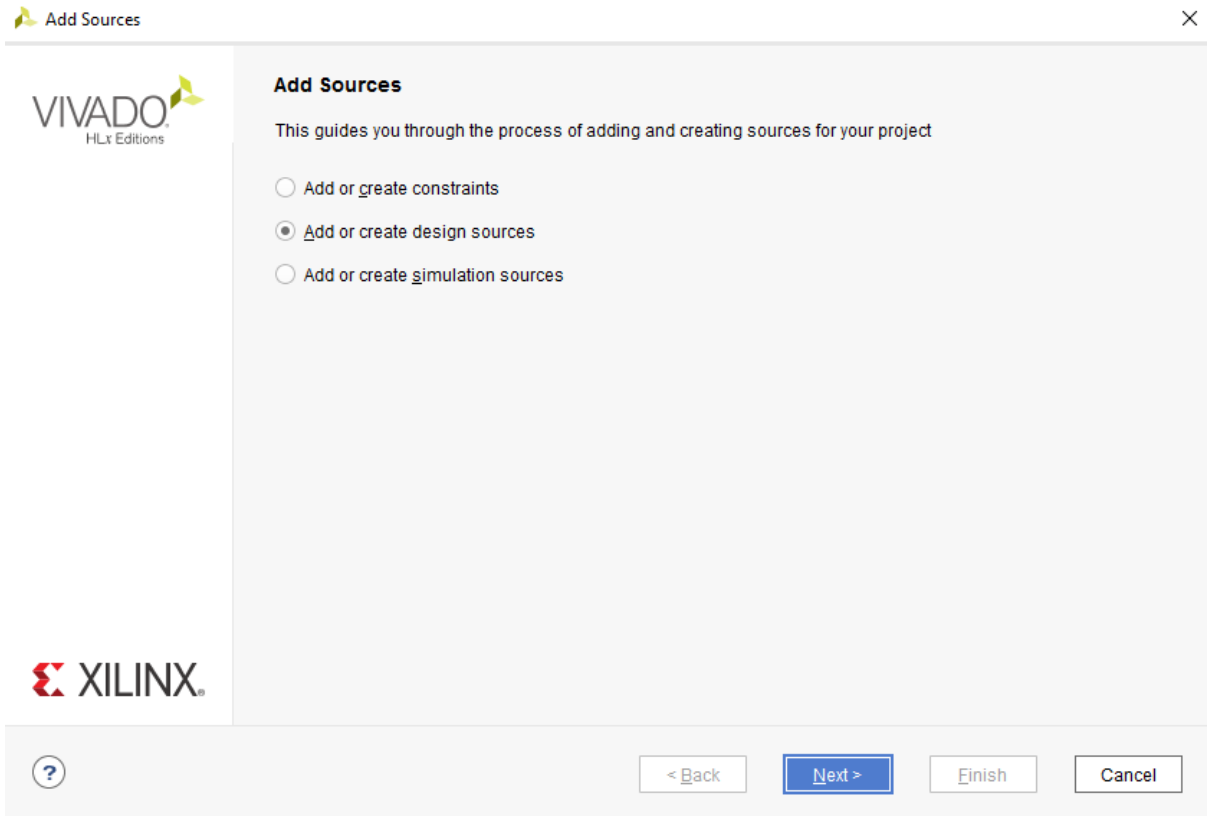


Figure 9 - Create design source

Click on “Create File”.

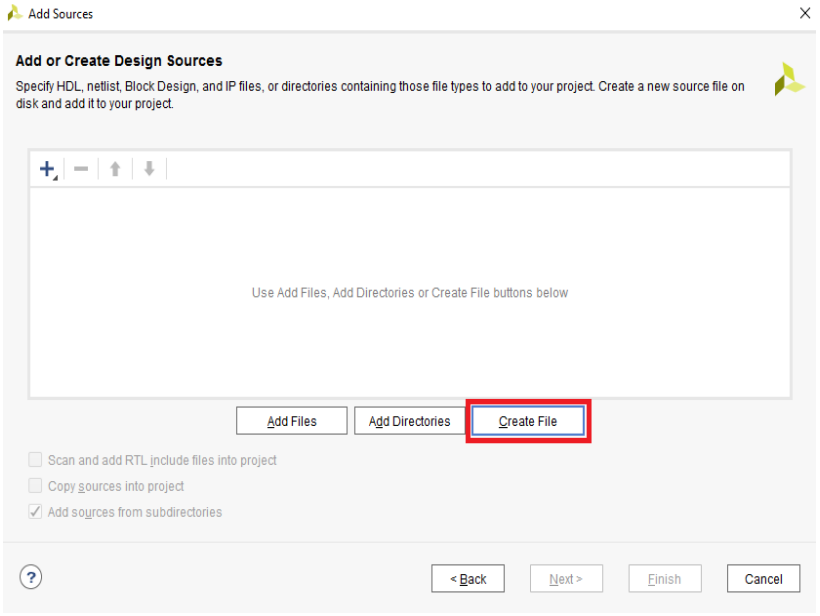


Figure 10 - Create design source

In the File name field, type heart_beat. Leave the File location local to the project. Click “OK”.

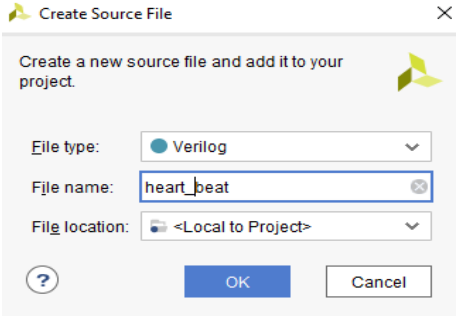


Figure 11 - Create a new source file

Click Finish to create the design source file.

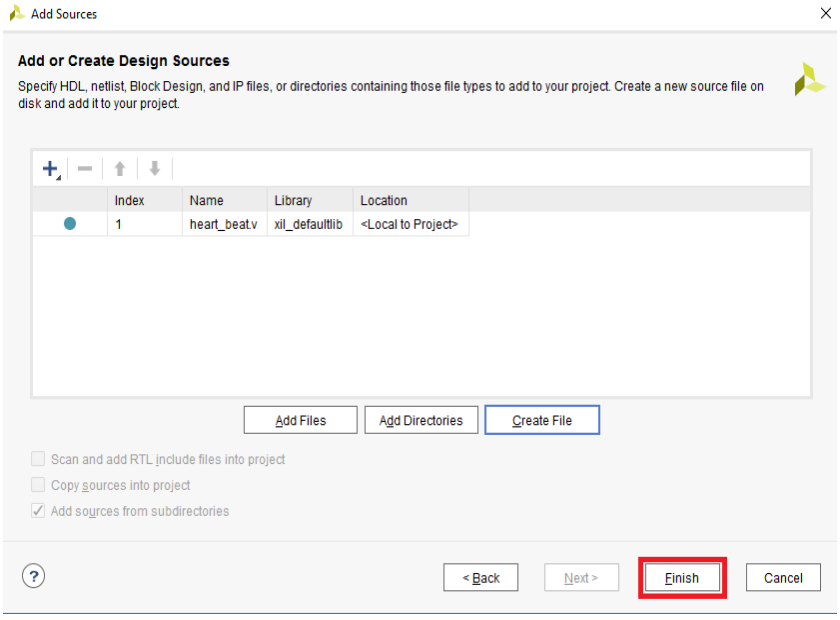


Figure 12 - Create design source

Click "Cancel" in the resulting dialog box.

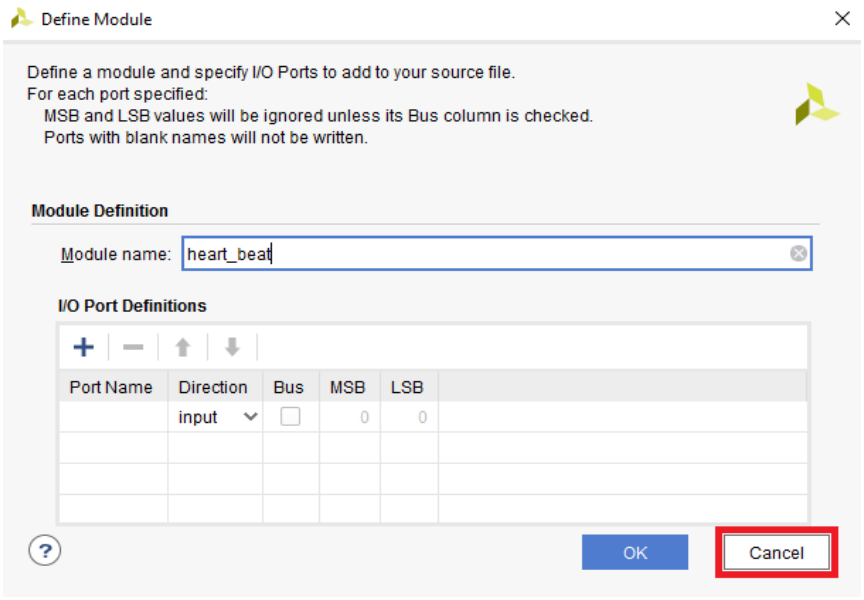


Figure 13 - Define the heart_beat module

In the warning message dialog box, click “Yes”. Module definition will be done manually in the next step.

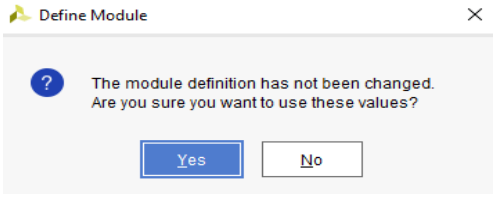


Figure 14 - Accept changes

Double-click on **heart_beat(heart_beat.v)** under "Sources" to open a code window. Enter the Verilog code as shown in Figure 15 below. Click on the “save” icon to save the Verilog file.

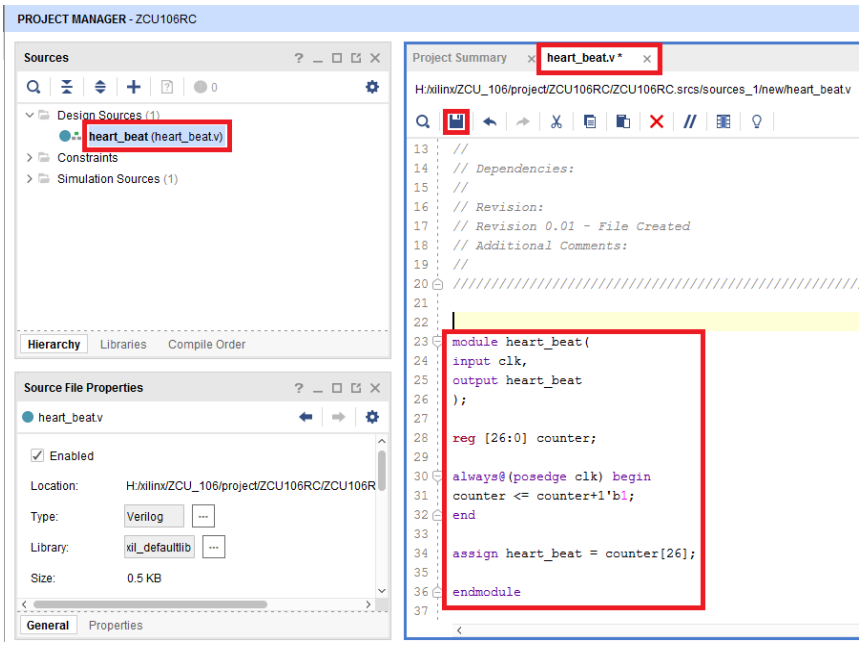


Figure 15 - Open heart_beat module

```
module heart_beat(
    input clk,
    output heart_beat
    );
    reg [26:0] counter;

    always@(posedge clk) begin
      counter <= counter+1'b1;
    end
    assign heart_beat = counter[26];
endmodule
```

Create the block design

From the Vivado Flow Navigator pane, click “Create Block Design”.

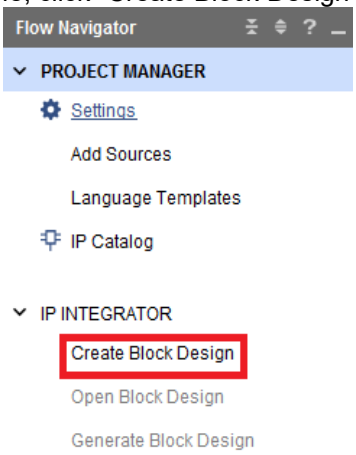


Figure 16 - Flow navigator window

Specify a name for the block design and leave the "Directory” field local to the project. Click “OK”.

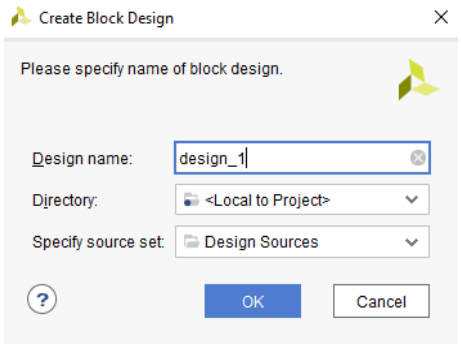


Figure 17 - Choose a name for block design

Right-click anywhere inside the Diagram window and select “Add Module...” as shown in Figure 18 below.

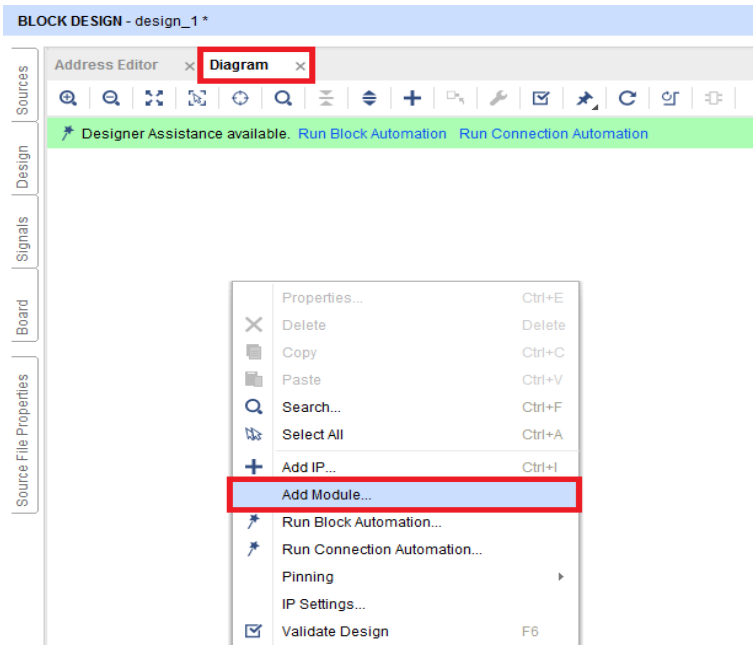


Figure 18 - Add module

Select "heart_beat.v" and click "OK". This will add a module named heart_beat to the block design.

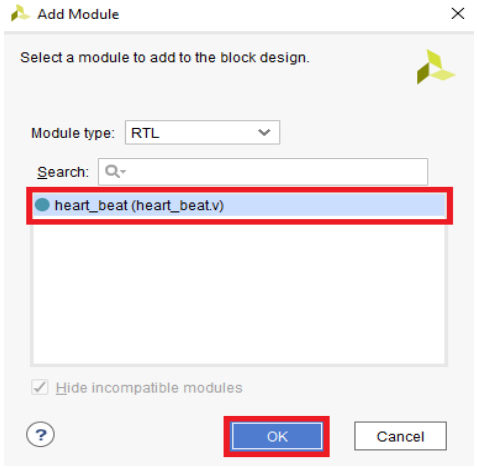


Figure 19 - Add the heart_beat module

In the Diagram window, click the “+” icon to add IP (Intellectual Property). Alternatively, use Ctrl+I or right-click anywhere inside the Diagram window and select “Add IP”.

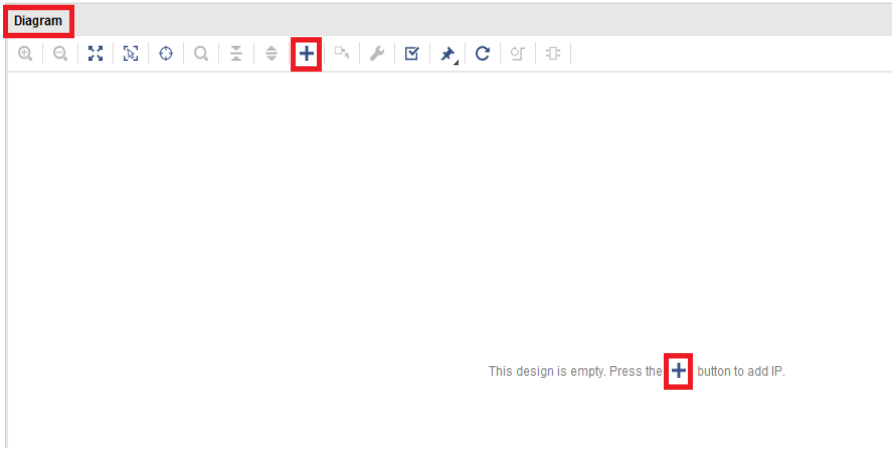


Figure 20 - Add IP via "Diagram" toolbar

The IP catalog will appear. Search for "DMA/Bridge Subsystem for PCI Express (PCIe)" by typing “PCIe” or the full name into the search field. Double-click on it to add the IP block.

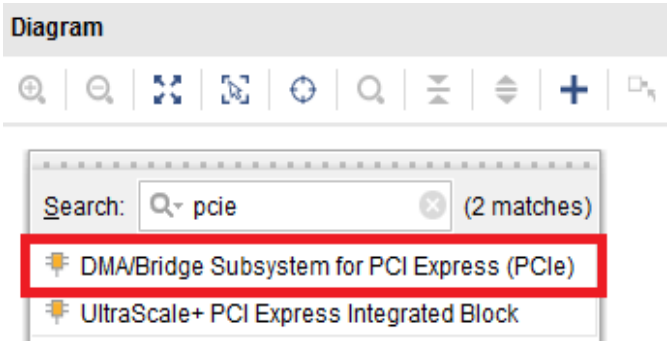


Figure 21 - Add DMA/Bridge Subsystem

Repeat the above step to search for and add the following IP blocks:

- Utility Buffer
- Utility Vector Logic
- System ILA
- AXI Interconnect (add two copies of this block)

Figure 22 below shows the complete list of IP blocks needed for the design.

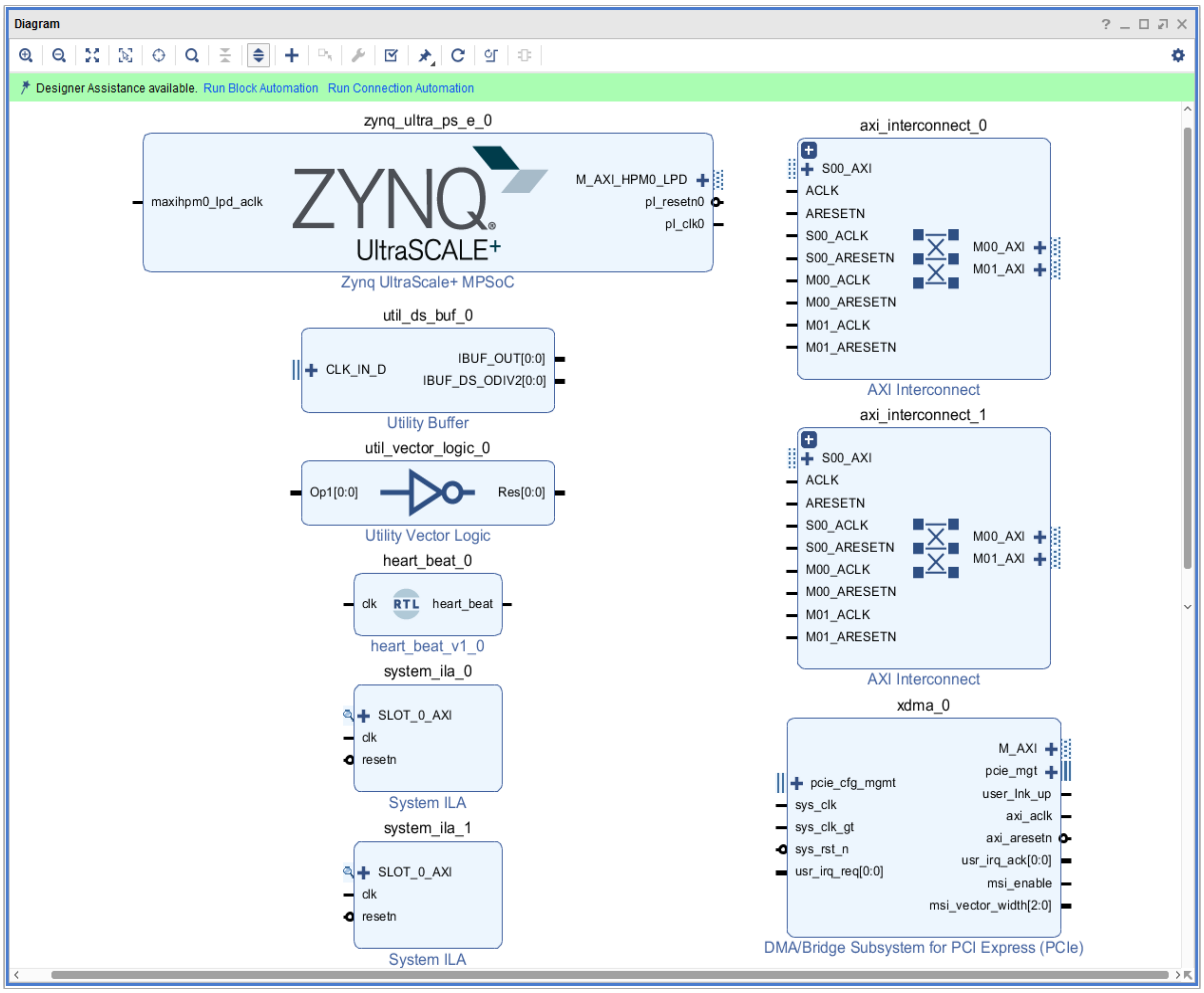


Figure 22 - The complete IP blocks

Re-customization

The next few steps will describe the steps to follow to re-customize IP blocks. To re-customize a block, double-click on the block without hitting the signal names.

Click on the Zynq PS (Zynq UltraScale+ MPSoC) block and click "Run Block Automation" to configure the Zynq PS for the target hardware.

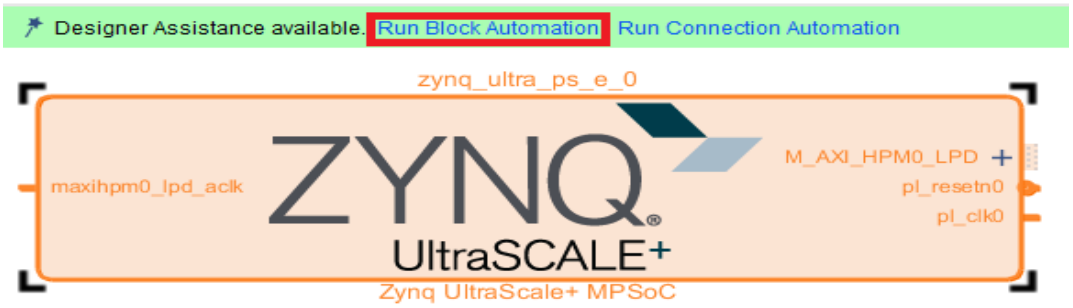


Figure 23 - Run block automation

Click OK to accept the default block automation settings.

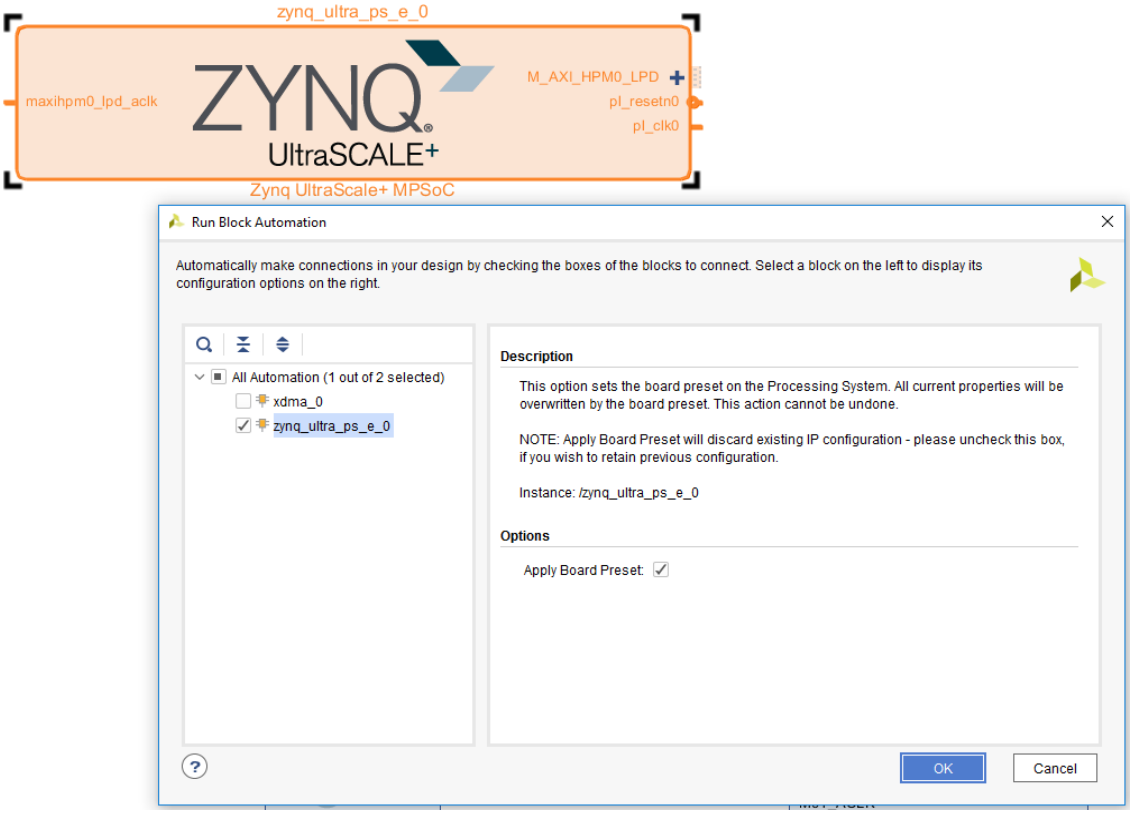


Figure 24 - Apply board preset

Double-click on the Zynq block to re-customize/configure it. Do not click on the signal names.

Figure 25 - Double click to re-customize block

Under Page Navigator select the “Switch to Advanced Mode” checkbox.

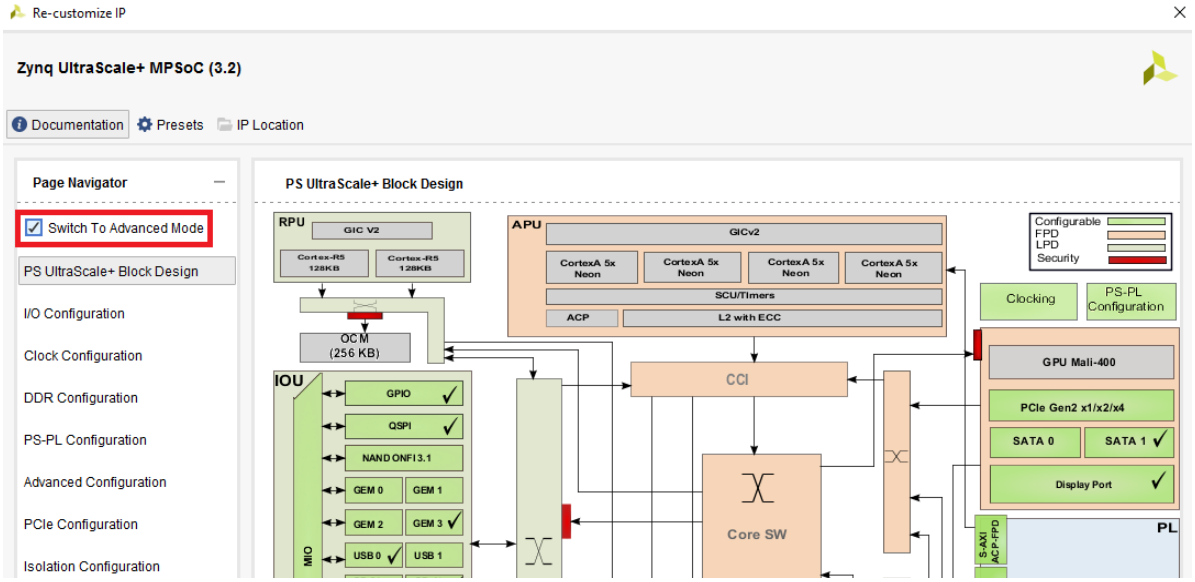


Figure 26 - Switch to advance mode

Click on the “I/O Configuration” tab and the “Expand All” icon as shown in Figure 27 below.

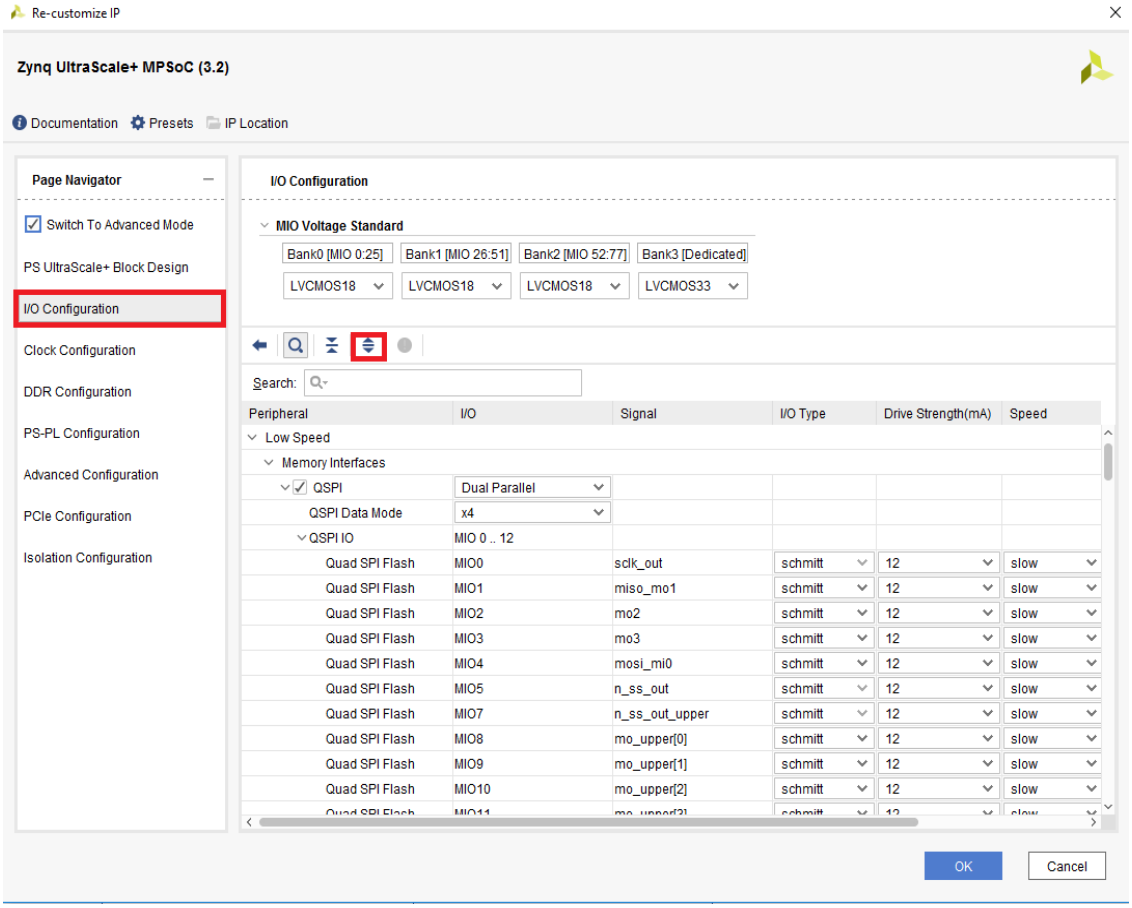


Figure 27 - I/O Configuration

Uncheck UART 1 as shown in the Figure 28.

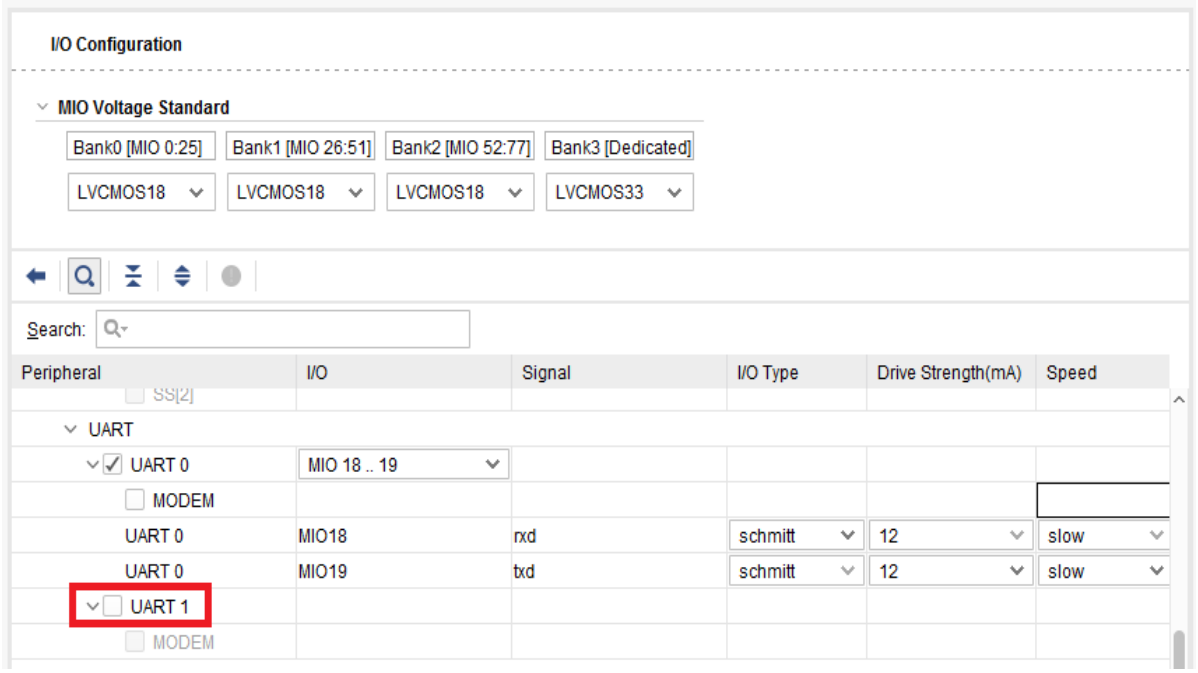


Figure 28 - I/O Configuration

Under the Page Navigator pane, click on the "Clock Configuration" tab. Select the "Output Clocks" tab and click on the "Expand All" icon.

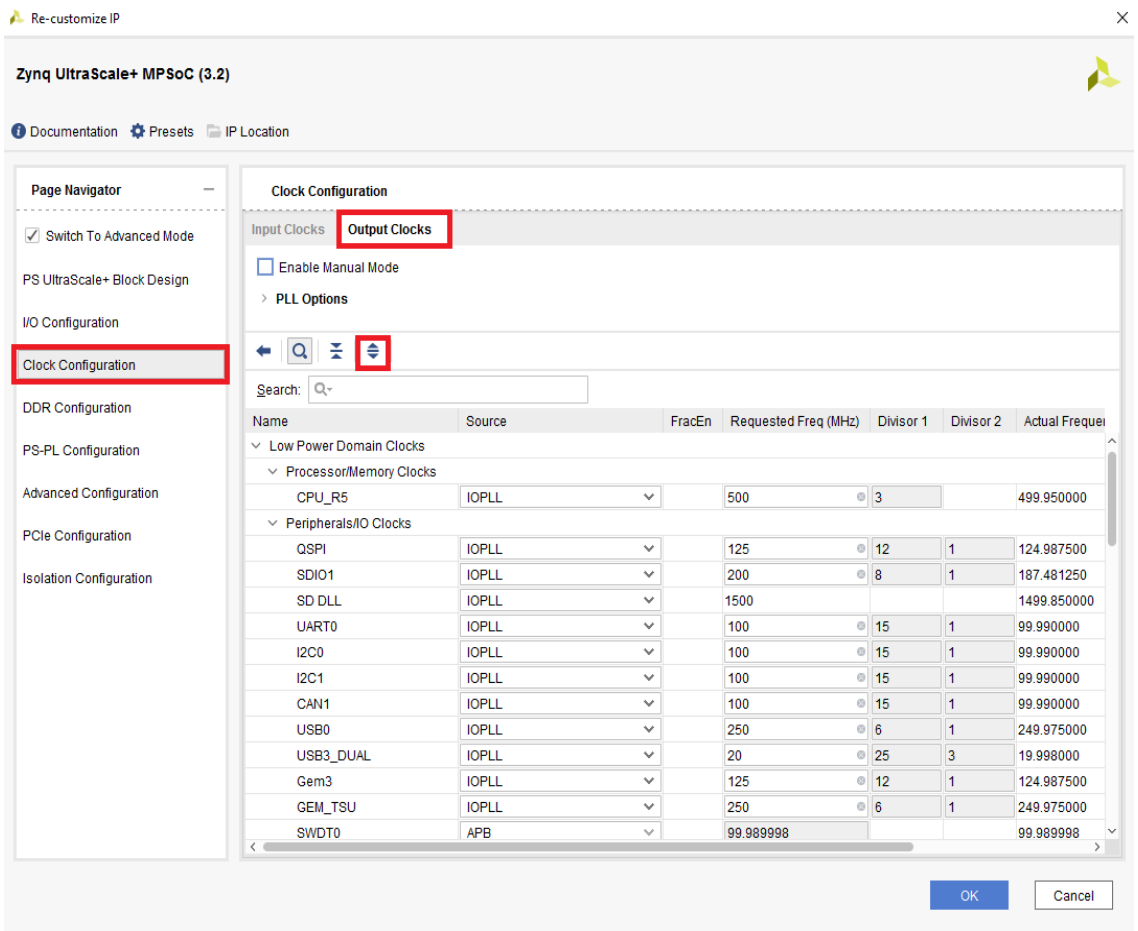


Figure 29 - Clock Configuration

Unselect the PL0 checkbox.

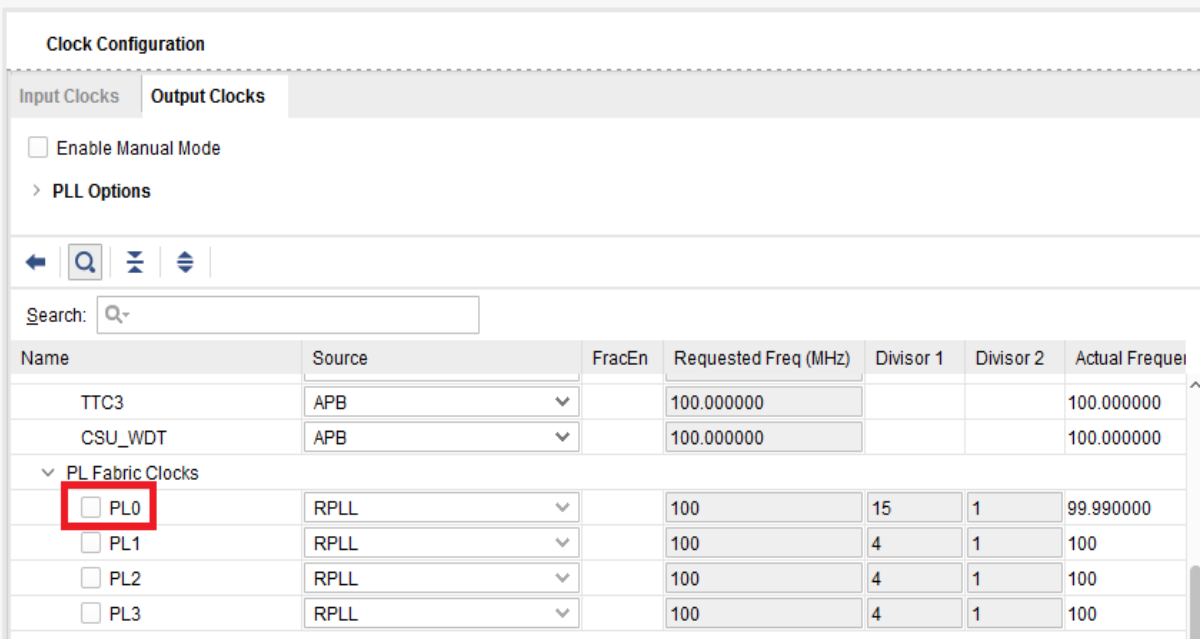


Figure 30 - Clock Configuration

Select the "PS-PL Configuration" tab under the Page Navigator pane. Expand the "General" drop-down menu.

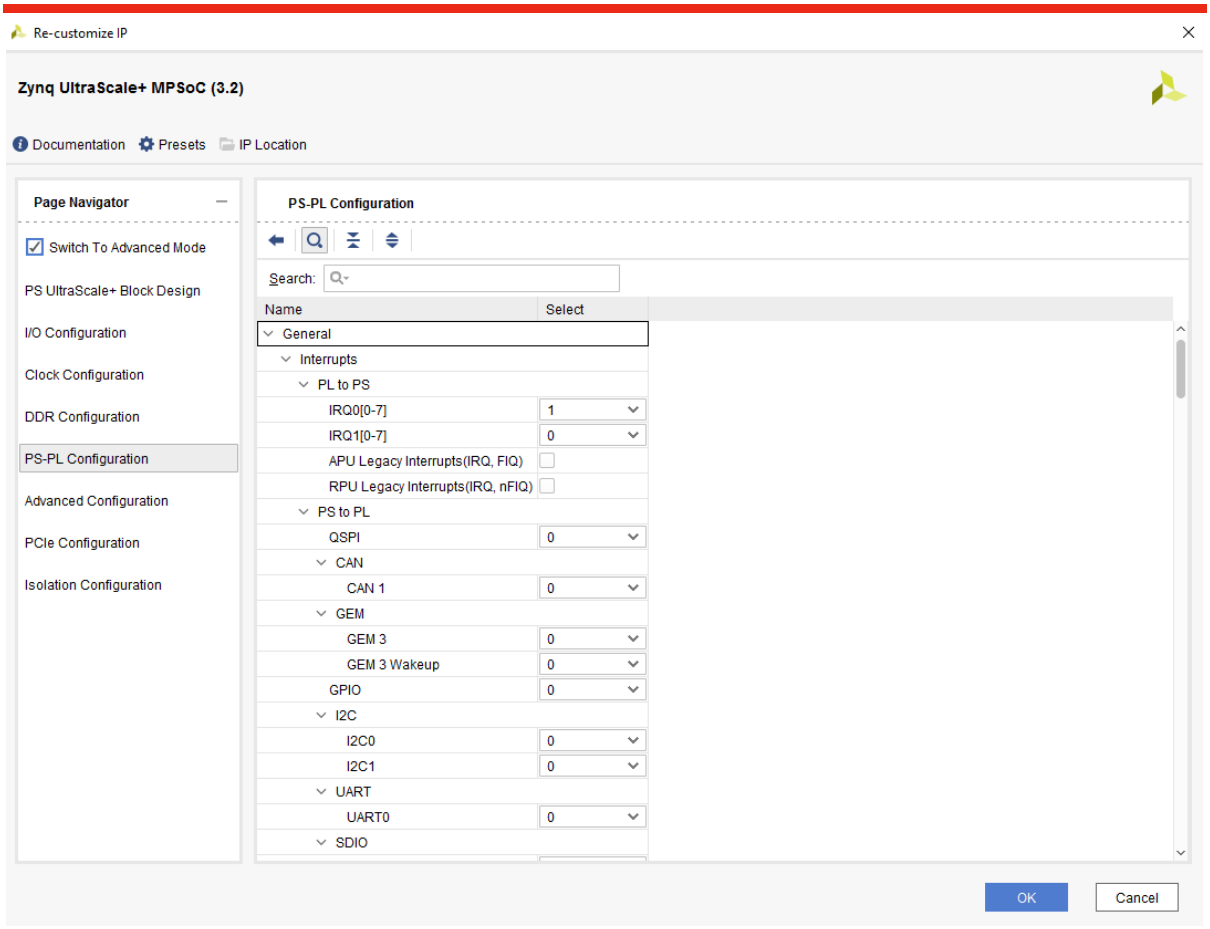


Figure 31 - PS-PL Configuration

In the PS-PL Configuration settings, uncheck "Fabric Reset Enable" and change "High Address" to 1 as shown in Figure 32.

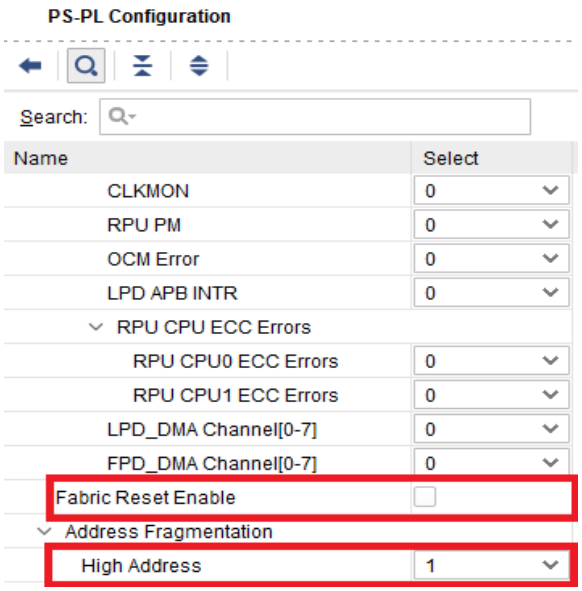


Figure 32 - PS-PL Configuration

Expand “PS-PL Interfaces”, disable/uncheck AXI HPM1 FPD and enable/check AXI HPC0 FPD.

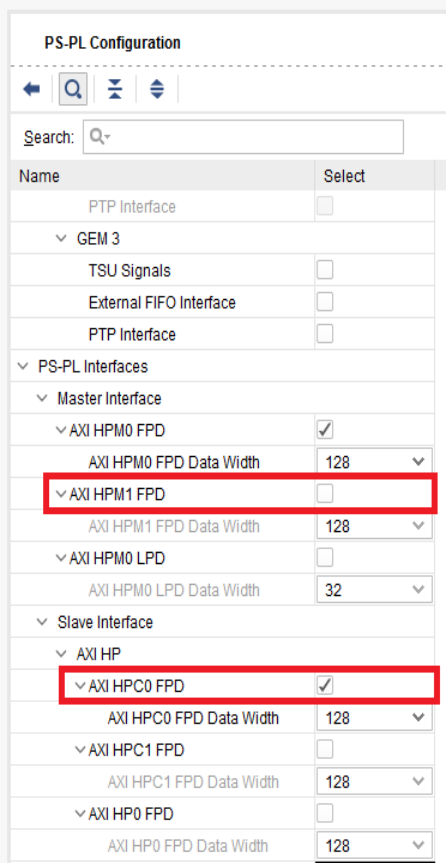


Figure 33 - PS-PL Configuration

Click OK.

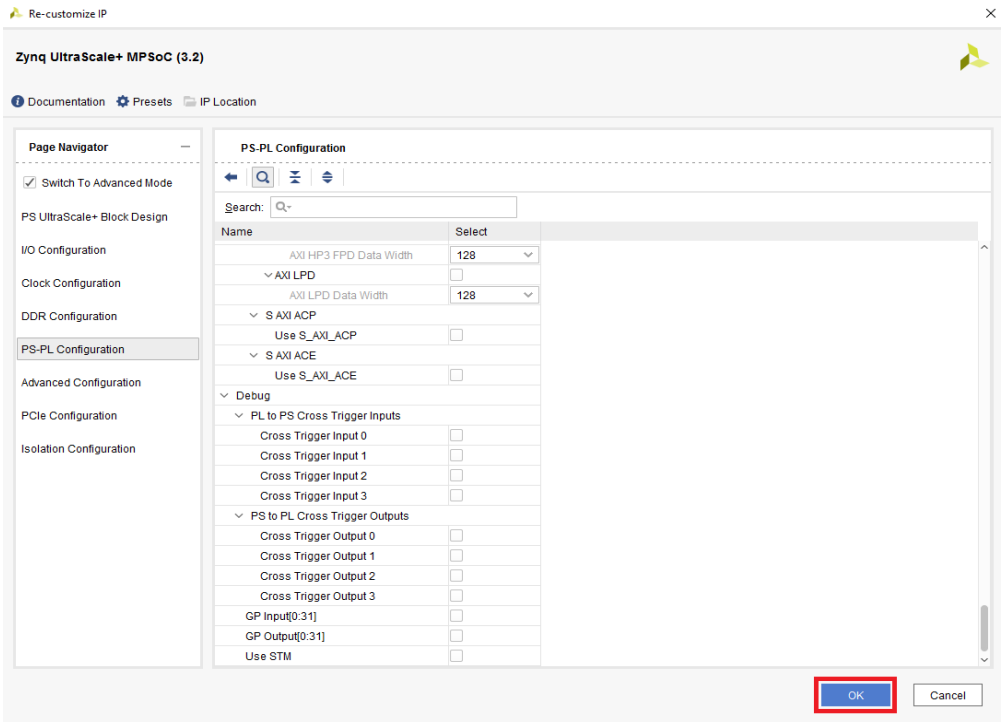


Figure 34 - PS-PL Configuration confirmation

The resulting Zynq block should look like the image in Figure 35.

Figure 35 – The customized Zynq block

Double-click on the “Utility Buffer” IP block to re-customize it.

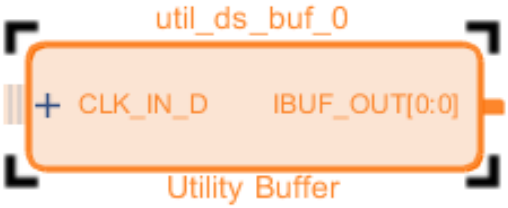


Figure 36 - Customize the utility buffer IP

Click on the "Page 0" tab and select the IBUFDSGTE radio button.

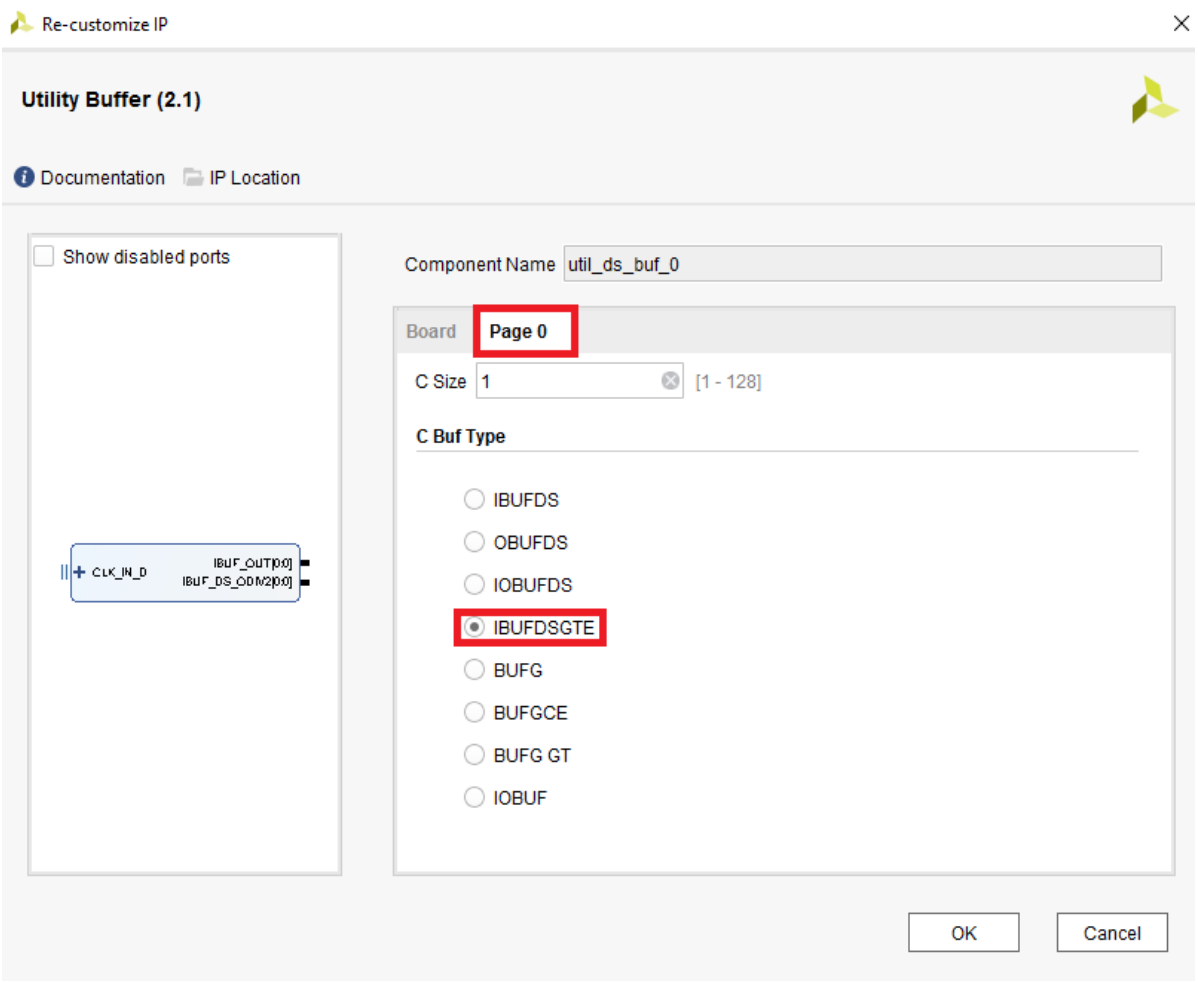


Figure 37 – Enable the IBUFDSGTE port

The resulting block should look like the image below.

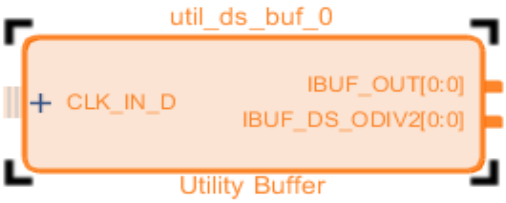


Figure 38 - The customized utility buffer

Double-click on the “Utility Vector Logic” block to re-customize it.

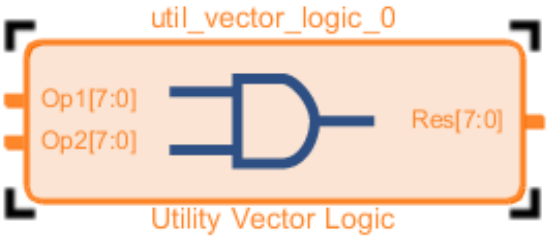


Figure 39 - Customize the utility vector logic

Under "C_OPERATION", select the "not" radio button as shown below and click OK.

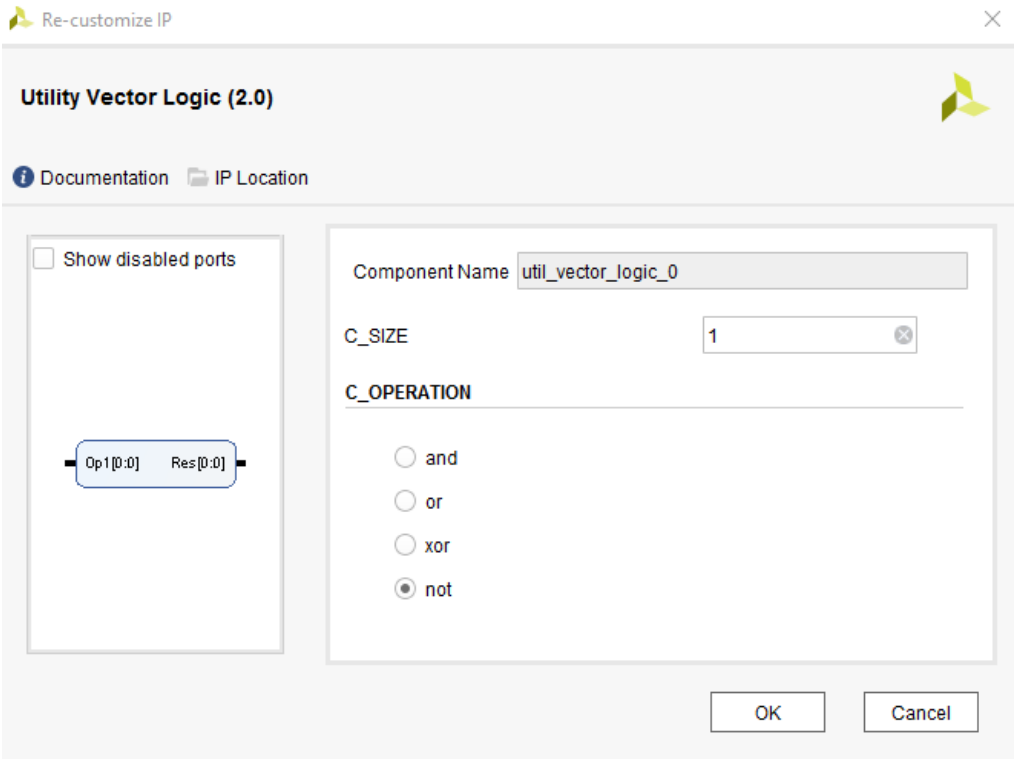


Figure 40 - Customize the utility vector logic

The customized block should look like the image below.

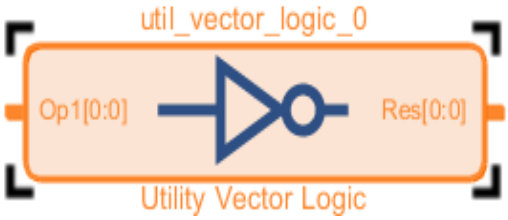


Figure 41 - The customized block

Double-click on the “DMA/Bridge Subsystem for PCI Express (PCIe)” block to re-customize it.

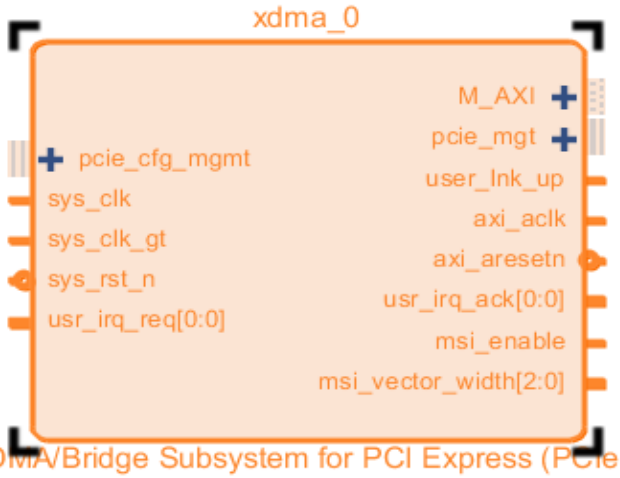


Figure 42 – Customize DMA/Bridge subsystem for PCIe

Click on the "Basic" tab and make the following changes as show in Figure 43:

Functional Mode ==> AXI Bridge
Mode ==> Advanced
Device/Port Type ==> Root Port of PCI Express Root Complex
PCIe Block Location ==> X0Y1
GT Quad ==> GTH Quad 227
Lane Width ==> X8
Maximum Link Speed ==> 8.0 GT/s
AXI Address Width ==> 64
AXI Data Width ==> 256 bit

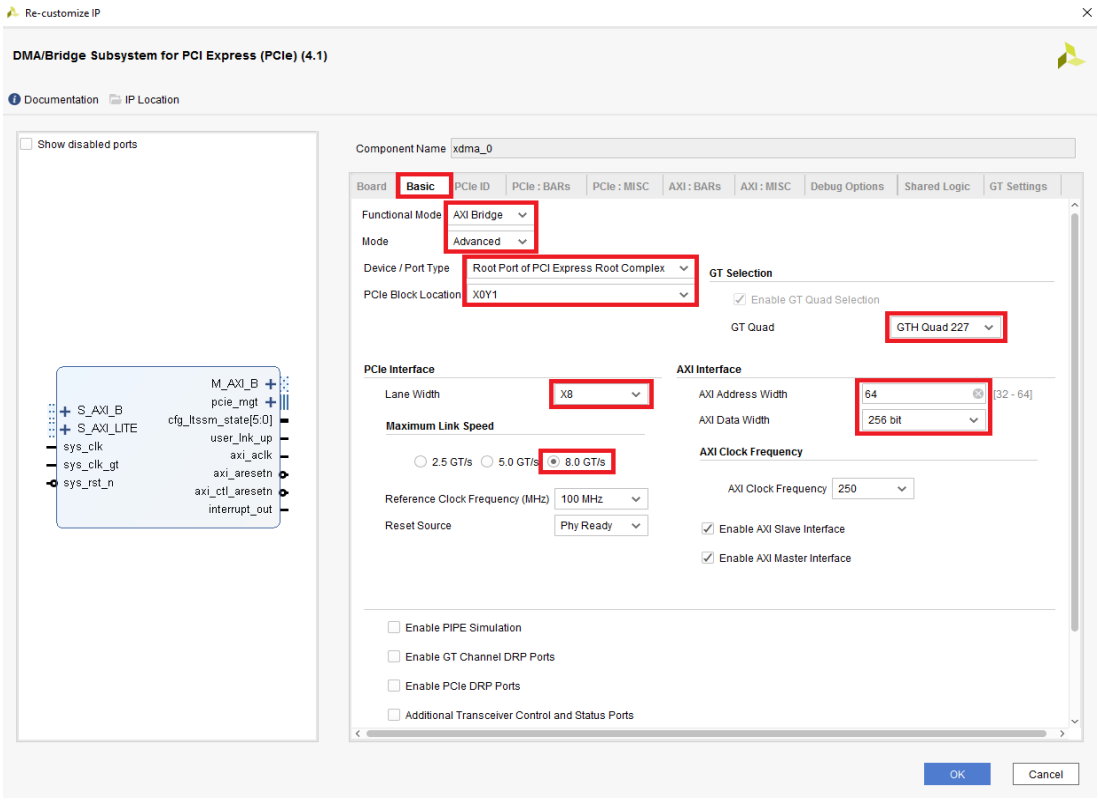


Figure 43 – Basic settings for DMA/Bridge subsystem for PCIe

Click the "PCIe ID" tab and select "PCI to PCI bridge" from the "Sub Class Interface Menu" drop-down list.

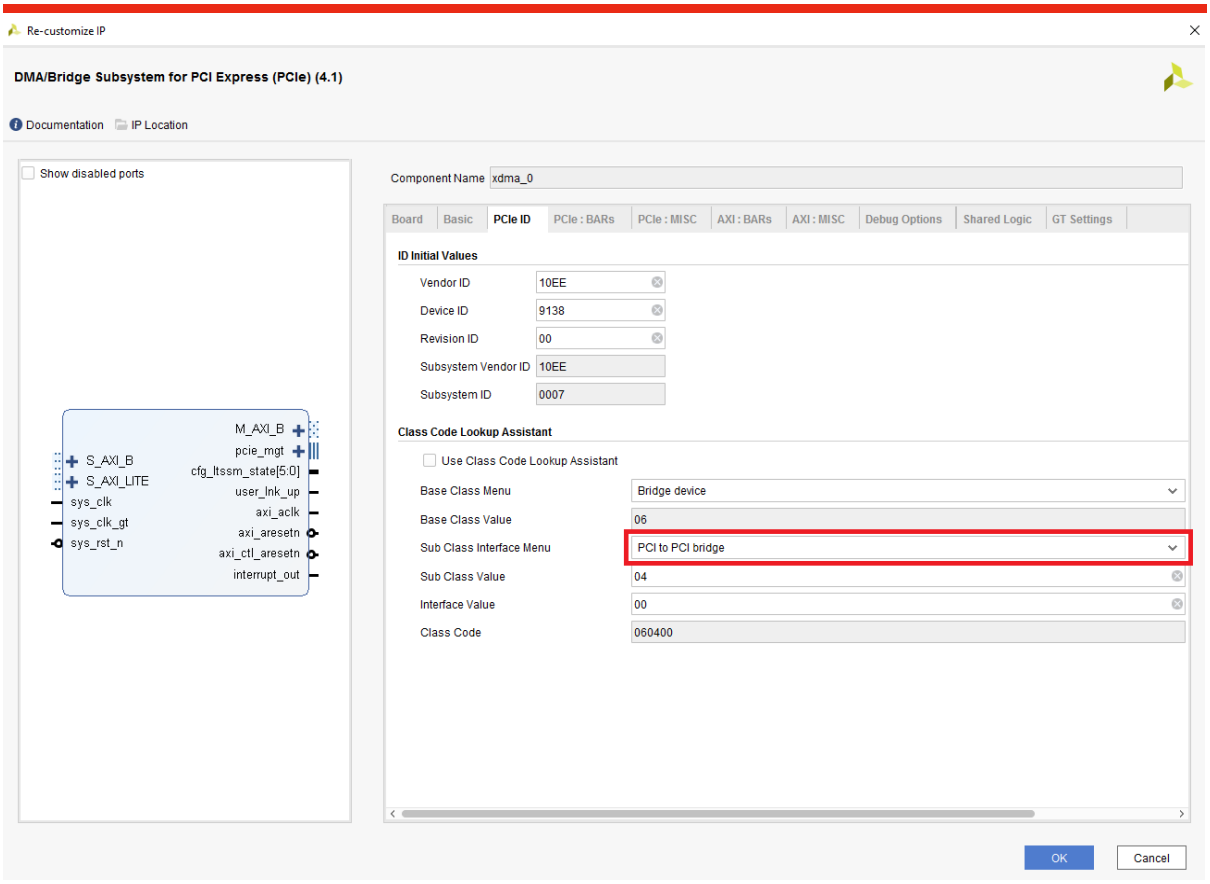


Figure 44 – Basic settings for DMA/Bridge subsystem for PCIe

Click on the "PCIe:BARs" tab and deselect PF0_BAR0.

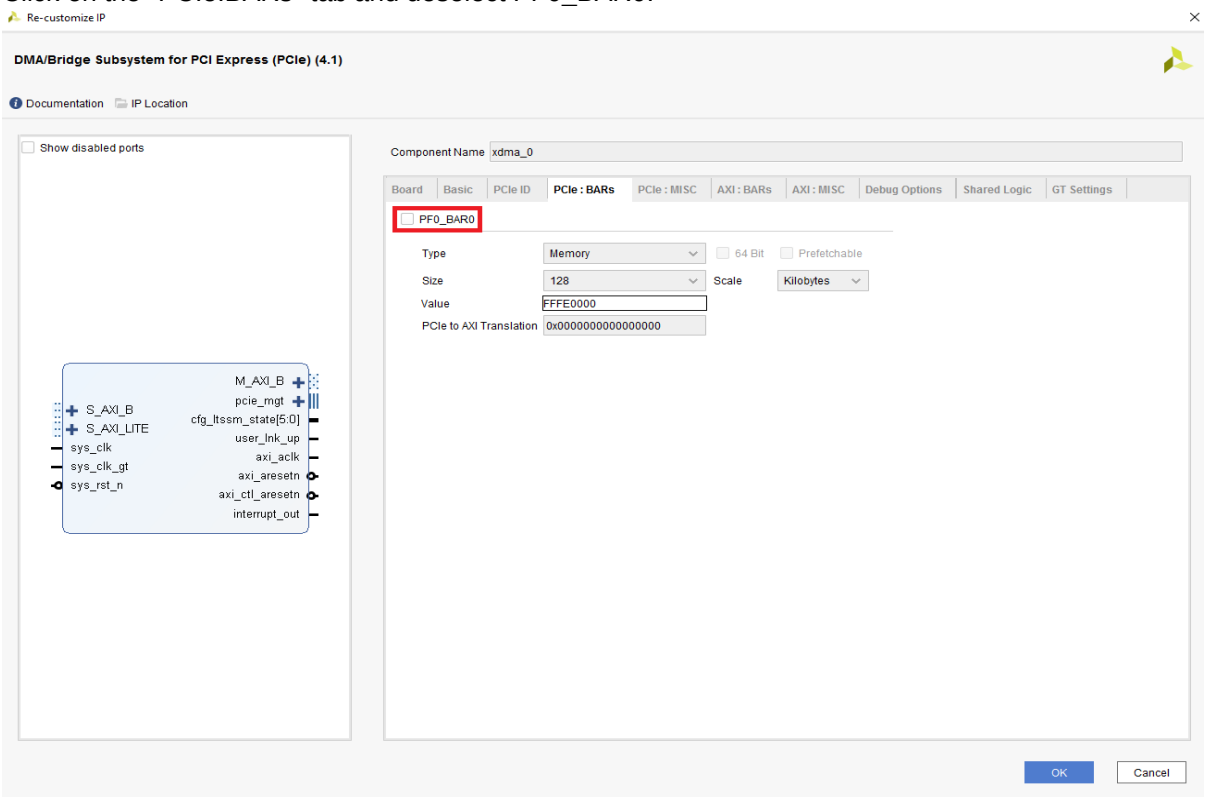


Figure 45 – PCIe:Bars settings for DMA/Bridge subsystem for PCIe

Click on the "AXI:MISC" tab and select "16" from the "AXI outstanding transactions" drop-down list.

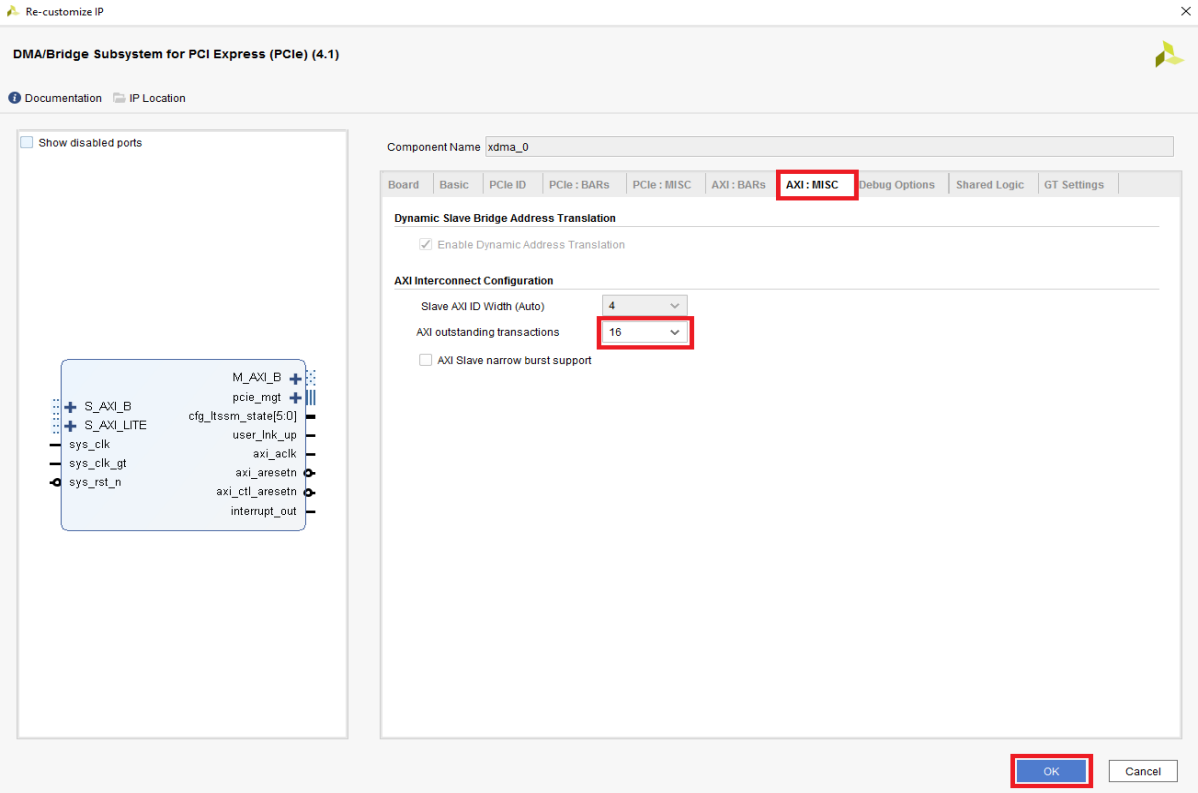


Figure 46 – AXI:MISC settings for DMA/Bridge subsystem for PCIe

The resulting IP block should be as shown in Figure 47.

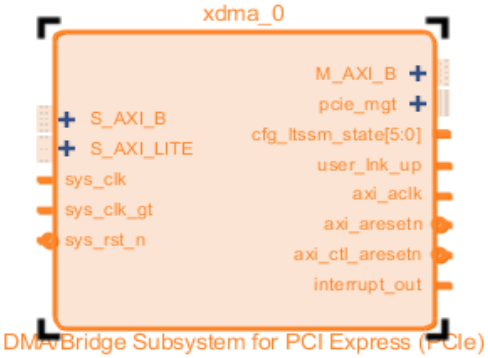


Figure 47 – The customized DMA/Bridge subsystem block

Double-click on the “AXI Interconnect” block to re-customize it.

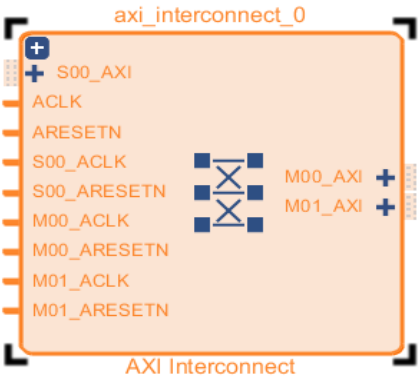


Figure 48 - Customized AXI Interconnect

Under “Top Level Settings” change "Number of Master Interfaces" to 1. Click “OK”.

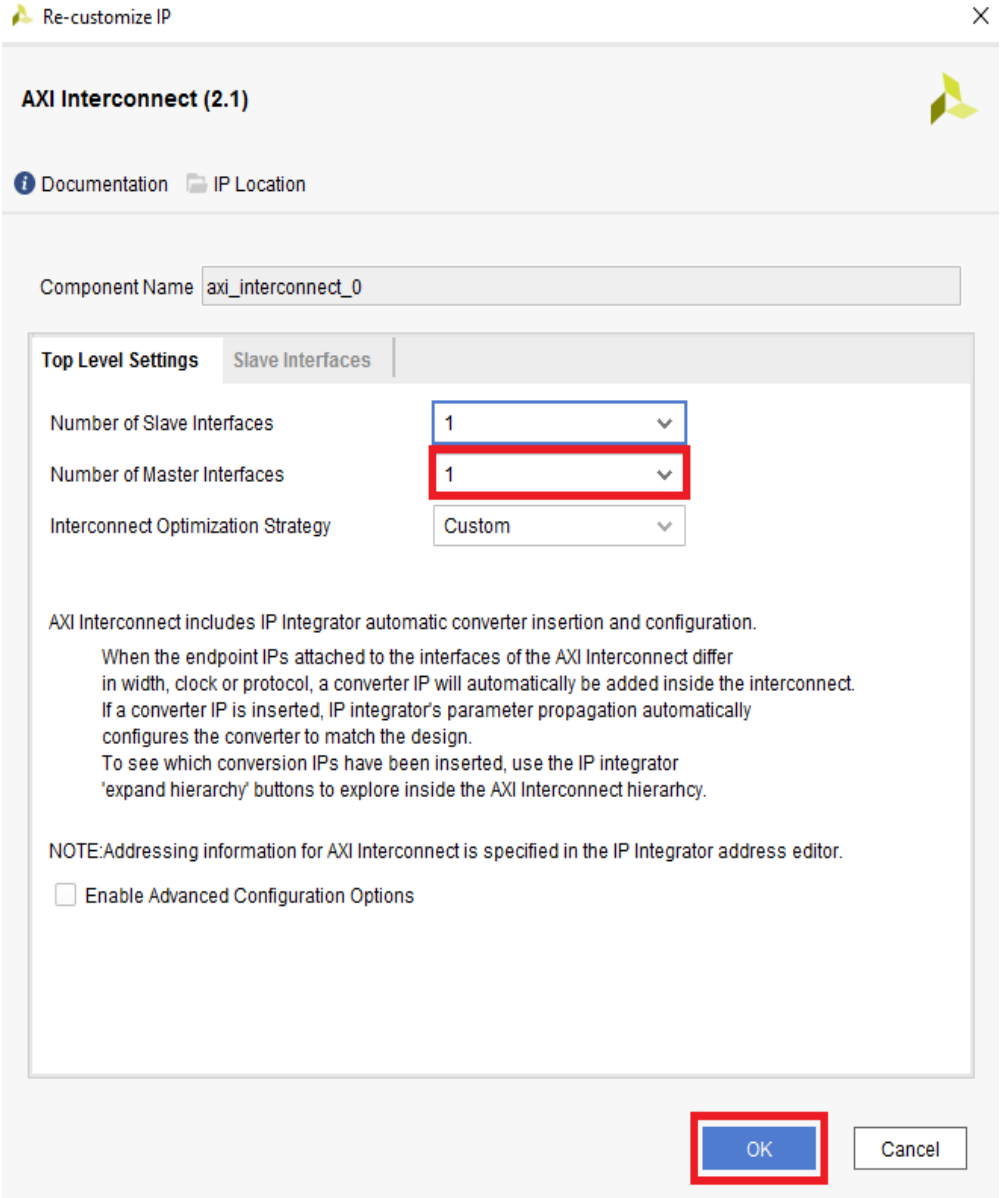


Figure 49 - Customize AXI Interconnect

The resulting block should look like the one shown in Figure 50.

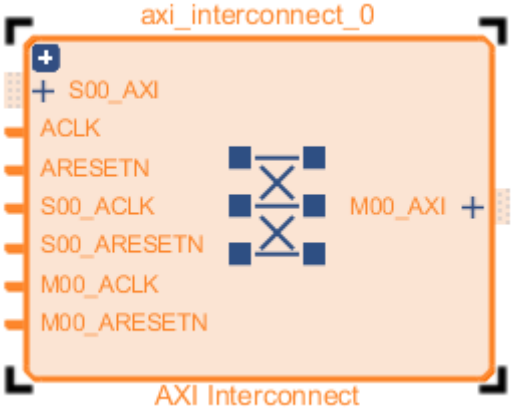


Figure 50 - Re-customized AXI IP block

Making connections

After re-customization, the complete IP blocks should look like Figure 51.

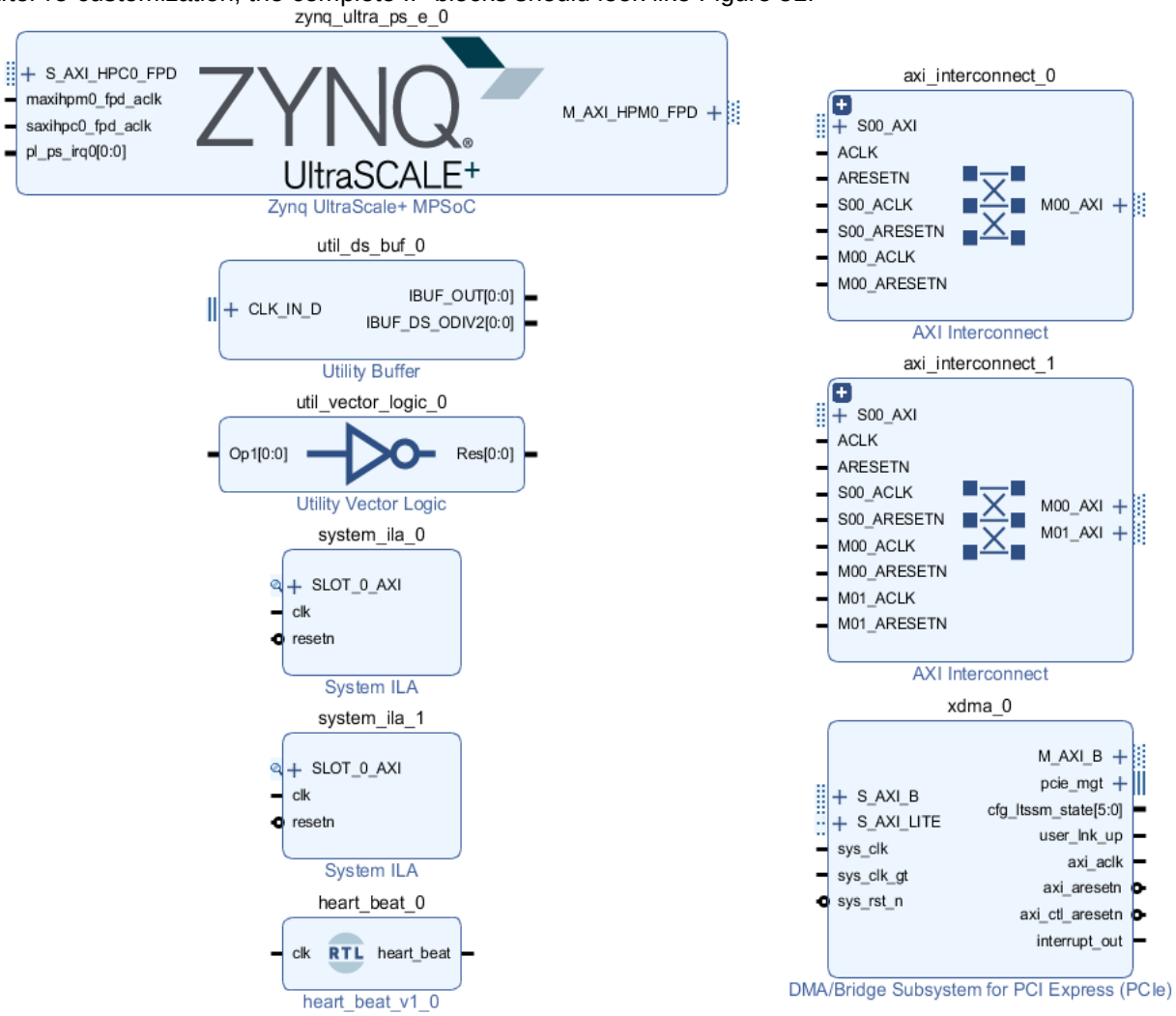


Figure 51 - The complete re-customized IP blocks

To make a connection between two points, from a particular port, click and hold the mouse button and drag to the desired connection point and then release the mouse button to establish connection. The text in blue signifies the IP block names while the text in green signifies the ports. For example, connecting [axi_interconnect_0 {M00_AXI}] to [zynq_ultra_ps_e_0 {S_AXI_HPC0_FPD}] means connecting the M00_AXI port of the axi_interconnect_0 block to the S_AXI_HPC0_FPD port of the zynq_ultra_ps_e_0 block. This connection is shown in Figure 52.

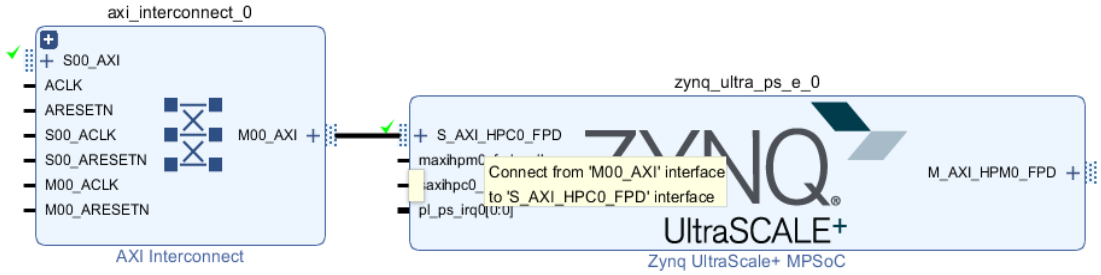


Figure 52 - Making port connections

Follow the blue net and make connections as shown in Figure 53.

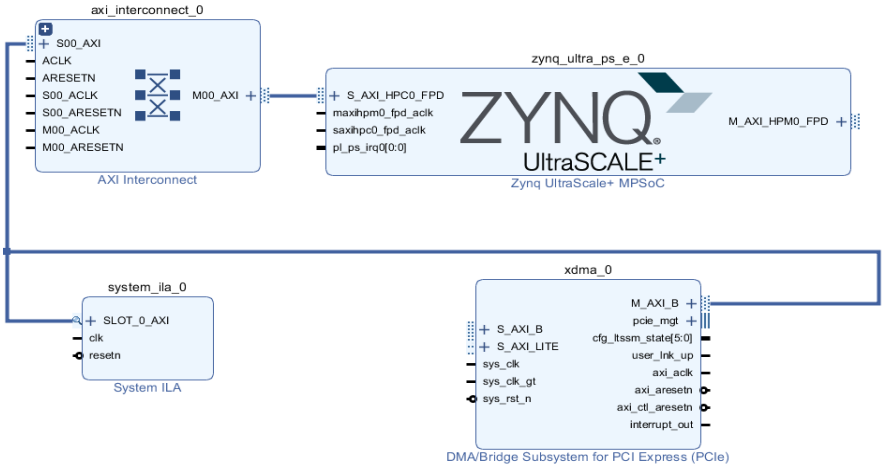


Figure 53 - Making port connections

Follow the purple net and make the following connections:
[zynq_ultra_ps_e_0 {M_AXI_HPM0_FPD}] to [axi_interconnect_1 {S00_AXI}]
[system_ila_1 {SLOT_0_AXI}] to [zynq_ultra_ps_e_0 {M_AXI_HPM0_FPD}]

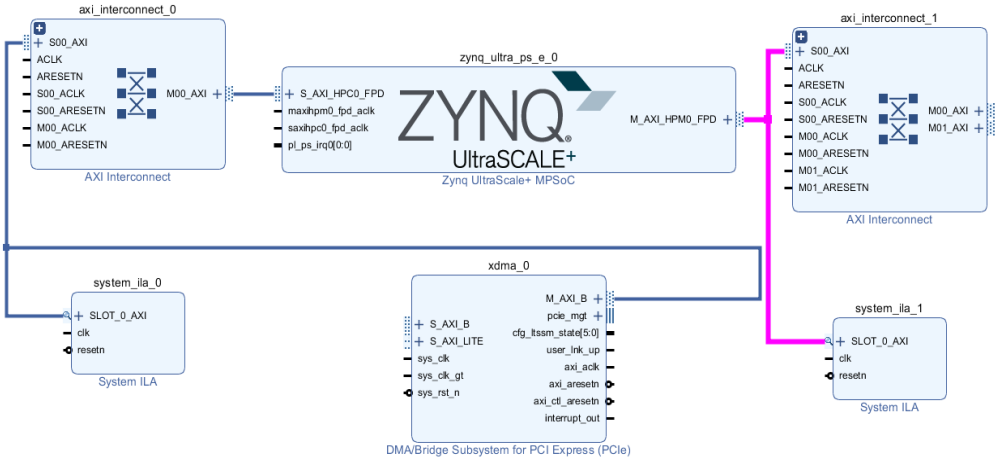


Figure 54 - Making port connections

Follow the green net and make the following connections:
[xdma_0 {S_AXI_B}] to [axi_interconnect_1 {M00_AXI}]
[xdma_0 {S_AXI_LITE}] to [axi_interconnect_1 {M01_AXI}]

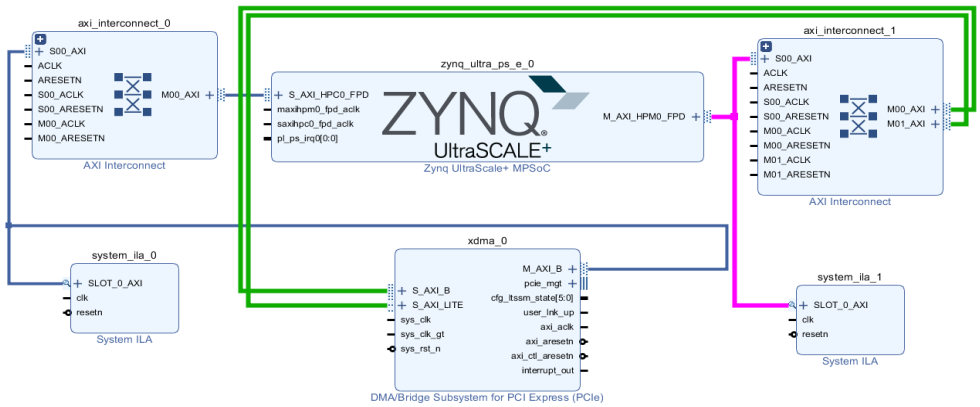


Figure 55 - Making port connections

Follow the yellow net and make the following connections for IP blocks that contain “clk” signals:
[zynq_ultra_ps_e_0 {maxihpm0_fpd_aclk}] to [zynq_ultra_ps_e_0 {saxihpc0_fpd_aclk}]
[axi_interconnect_0 and axi_interconnect_1 {*ACLK}] to [system_ila_1 {clk}]
[xdma_0 {axi_aclk}] to [system_ila_1 {clk}].
The “*ACLK” means all the ACLK signals in the IP block.

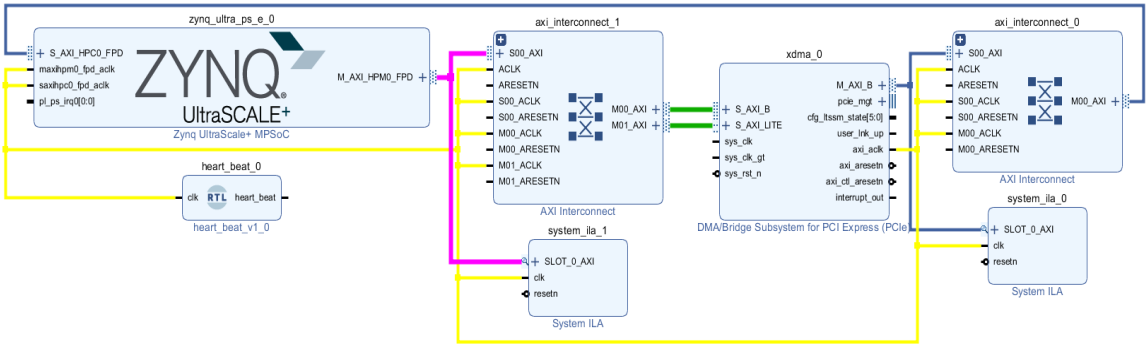


Figure 56 - Making port connections

Follow the red net and make the following connections for IP blocks that contain “reset”:
[axi_interconnect_0 and axi_interconnect_1 {*ARESETN}] to [system_ila_1 {resetn}].
The “*ARESETN” means all of the ARESETN signals in the IP block.

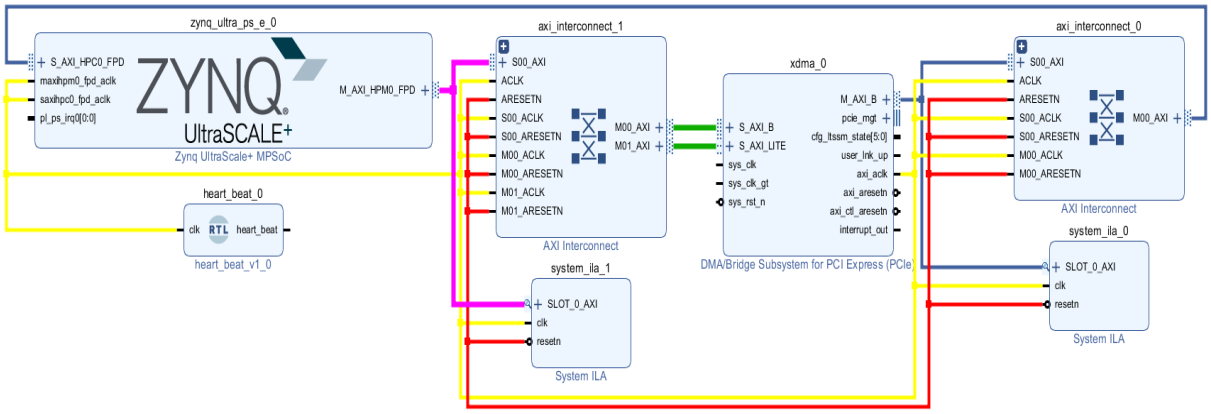


Figure 57 - Making port connections

Make a connection from [zynq_ultra_ps_e_0{pl_ps_irq}] to [xdma_0 {interrupt_out}] as shown by the cyan connection in the figure below.

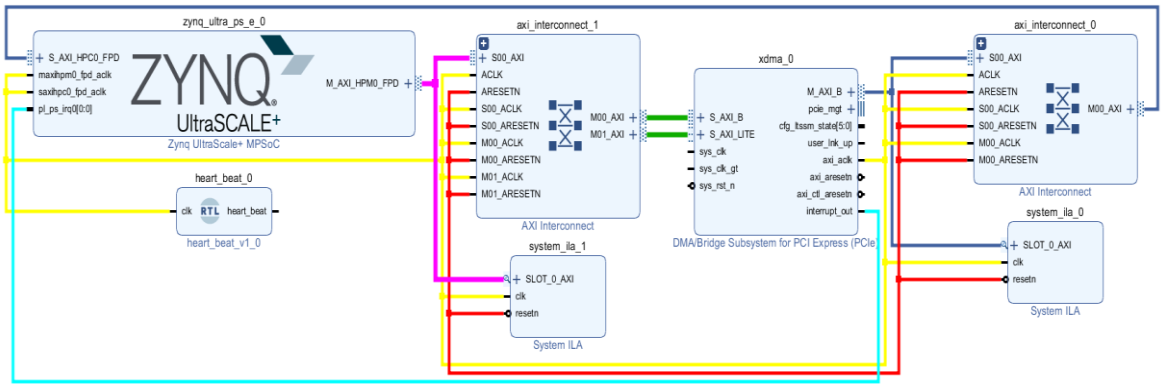


Figure 58 - Making port connections

Make a connection from [util_ds_buf_0{IBUF_OUT}] to [xdma_0 {sys_clk_gt}] and [util_ds_buf_0{IBUF_DS_ODIV2}] to [xdma_0 {sys_clk}] as shown by the purple and brown connections in Figure 59.

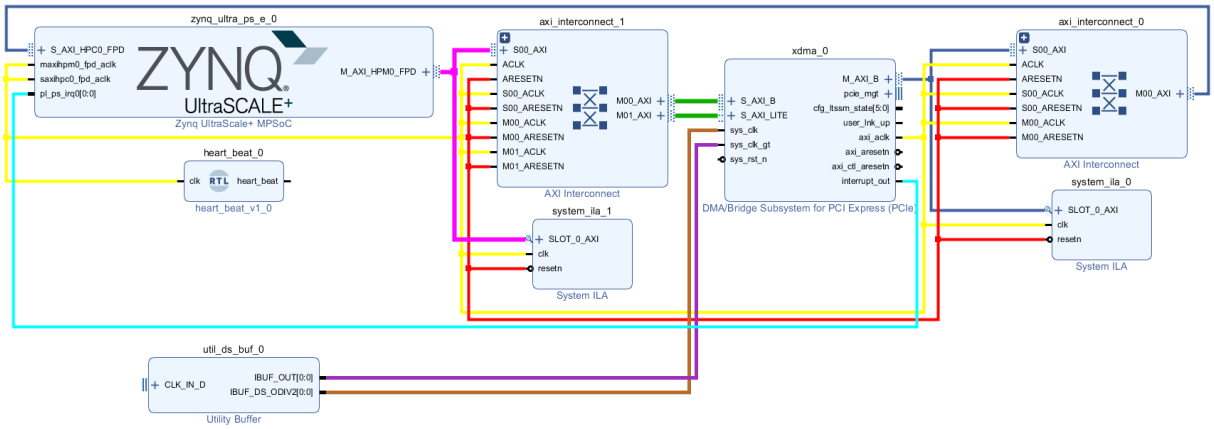


Figure 59 - Making port connections

Make a connection from [util_vector_logic_0{Res}] to [xdma_0 {sys_rst_n}] as shown by the light green connection in Figure 60.

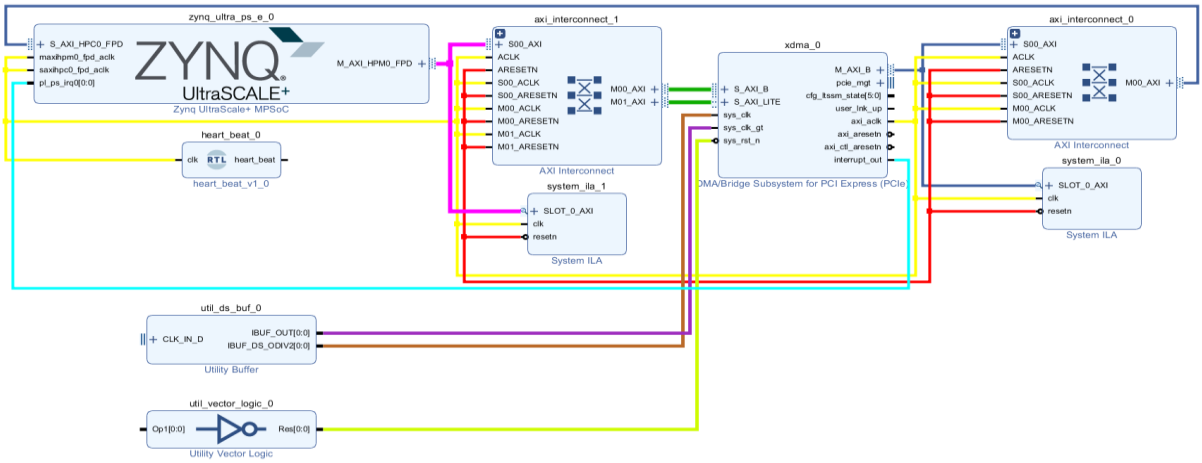


Figure 60 - Making port connections

Right-click on **heart_beat** port of the *heart_beat_0* IP block and select “Make External” from the drop-down menu as shown in the figure below. Do the same for the following ports/signals:
[util_ds_buf_0 {CLK_IN_D}]
[util_vector_logic_0 {OP1}]
[xdma_0 {pcie_mgt}]
[xdma_0 {user_lnk_up}]

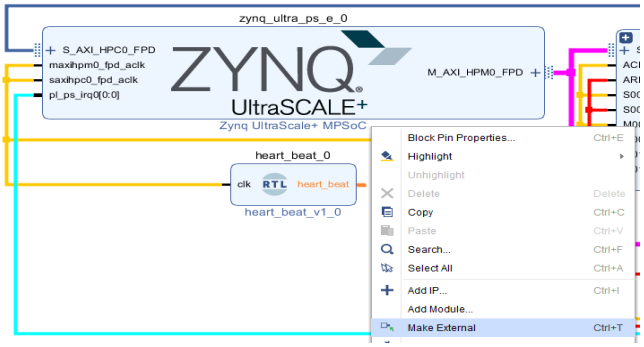


Figure 61 - Making external connections

The figure below shows the external connections that have been made.

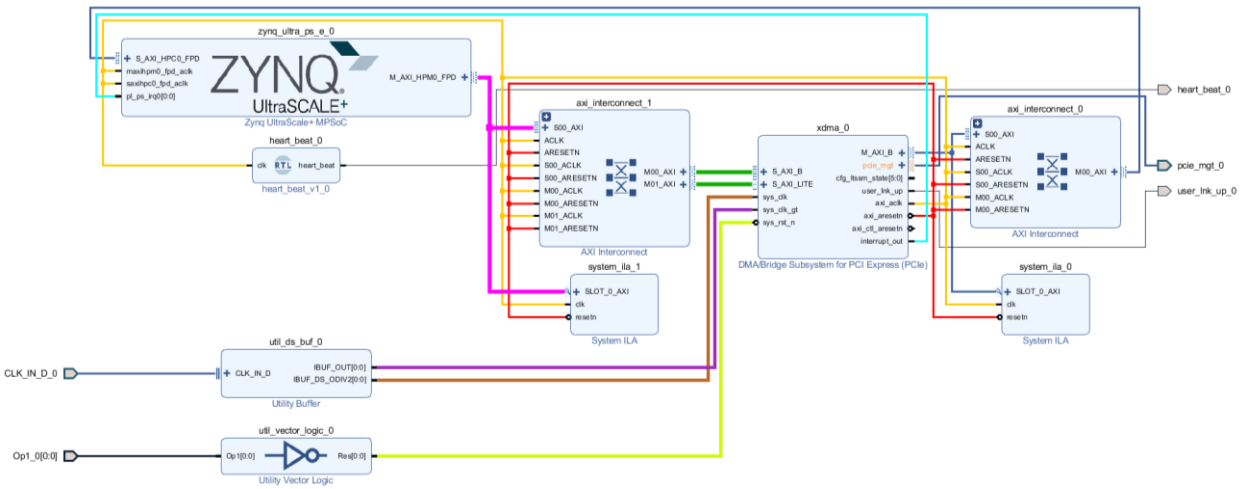


Figure 62 - Showing the external connections

Click on the “Regenerate” icon to rearrange the IP blocks.

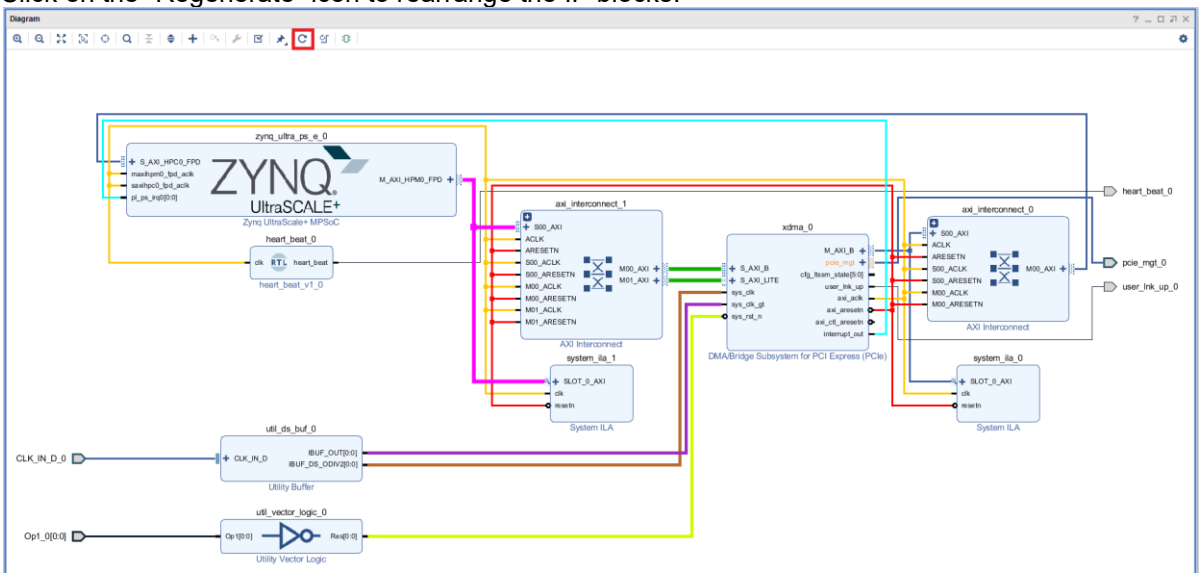


Figure 63 - Regenerate the layout

The regenerated layout is shown in Figure 64.

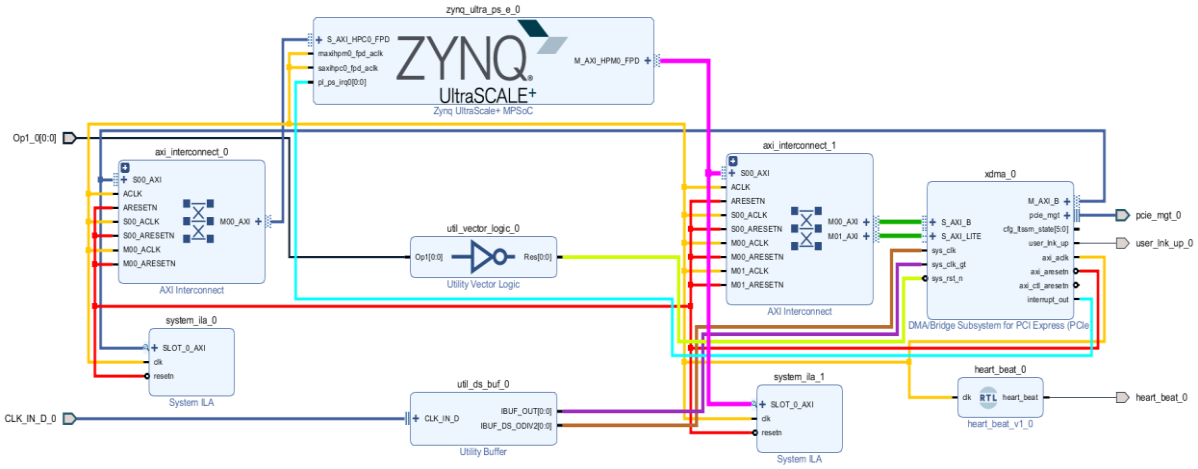


Figure 64 - Regenerated layout

Click on the “save” icon on the toolbar to save the design.

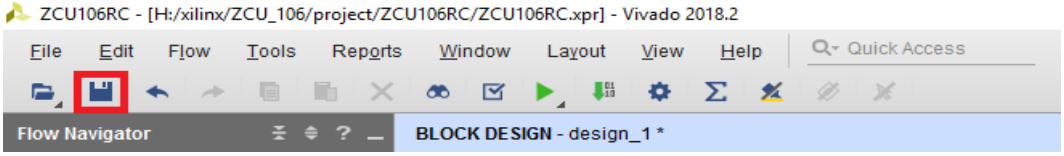


Figure 65 - Save the design

Click on the signal pin (Op1) of the util_vector_logic_0 block and rename it to **sys_rst** under External Properties as shown below.

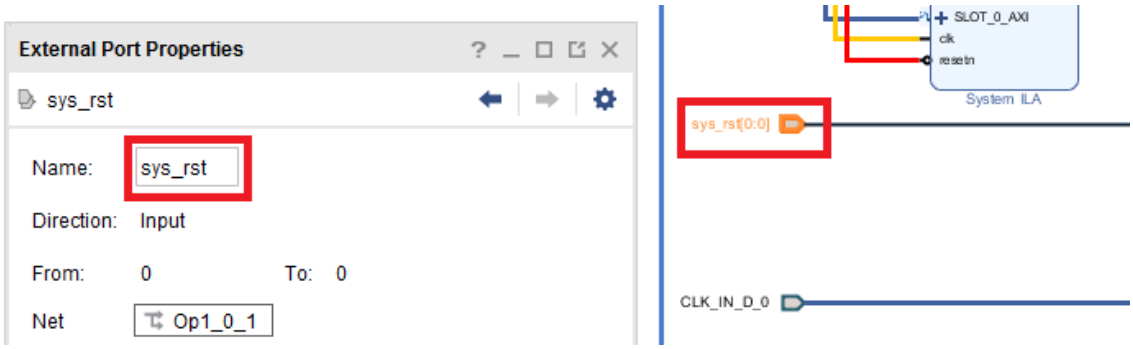


Figure 66 - Rename Op1 signal

Similarly, click on external signal pin CLK_IN_D of the util_ds_buf_0 block and rename it to **sys_clk** under External Properties as shown in Figure 67.

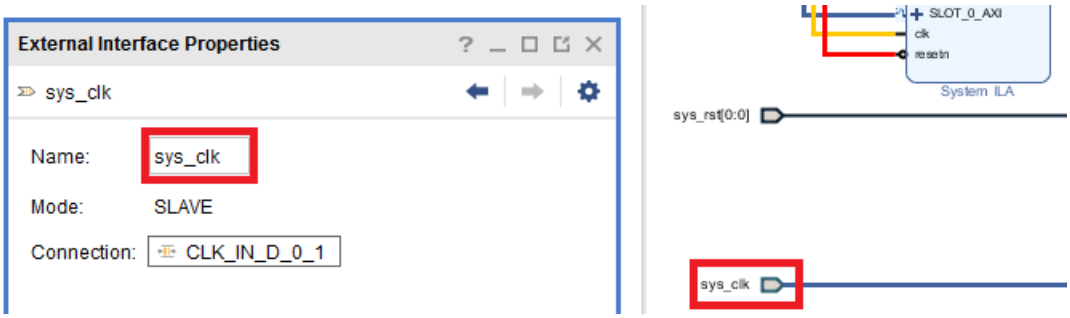


Figure 67 - Rename CLK_IN_D signal

Do the same for the following external connection signals:
Rename **user_lnk_up_0** to *led_0*
Rename **heart_beat** to *led_1*
After renaming the signals the resulting layout should look like Figure 68.

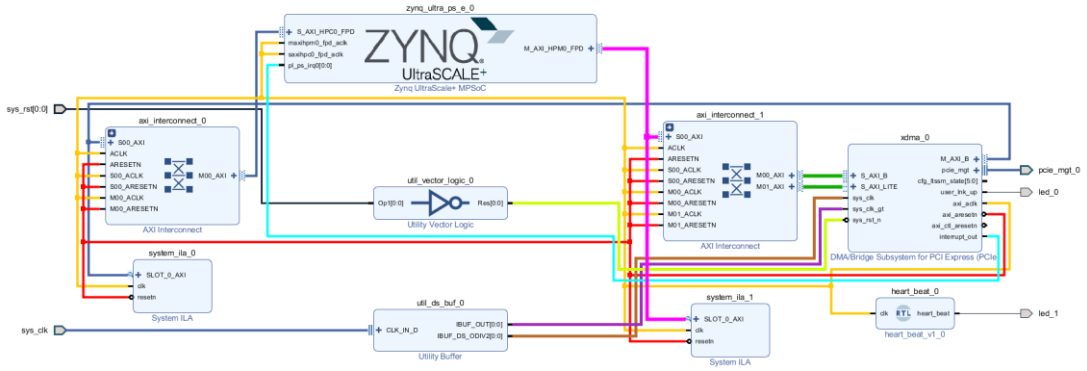


Figure 68 - Layout after renaming signals

Click on the “Address Editor” tab beside the “Diagram” tab. Click on the “Auto Assign Address” icon as shown in the red box in Figure 69.

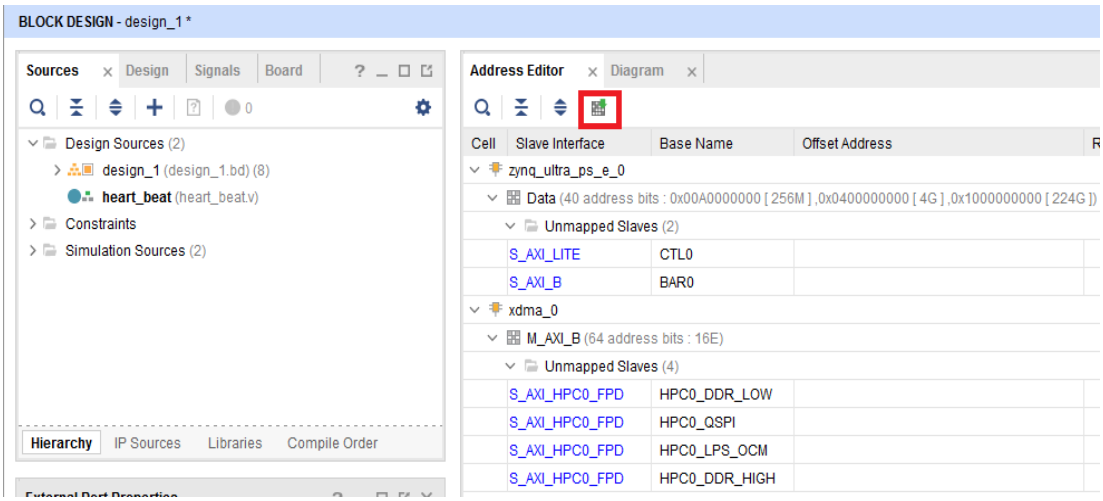


Figure 69 - Auto assign address

A message dialog box will confirm that auto-assigning the address was successful.

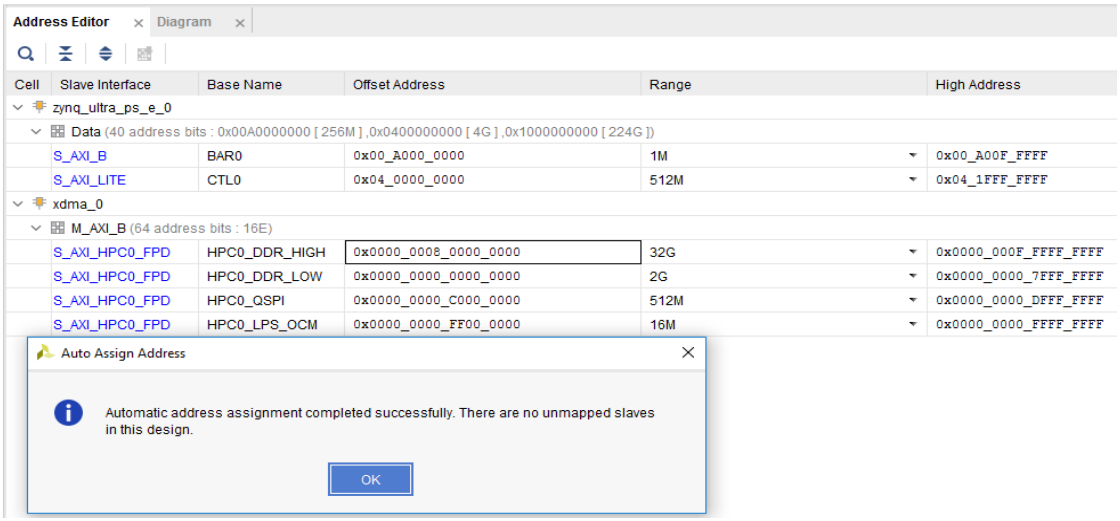


Figure 70 - Auto assign address confirmation

Change the range for [zynq_ultra_ps_e_0 {S_AXI_B}] to 256M.

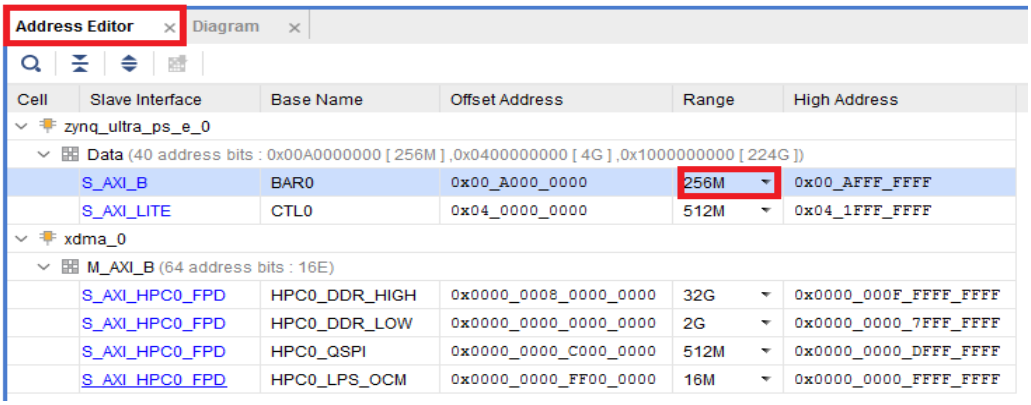


Figure 71 - Change address range

Regenerate the block design again.

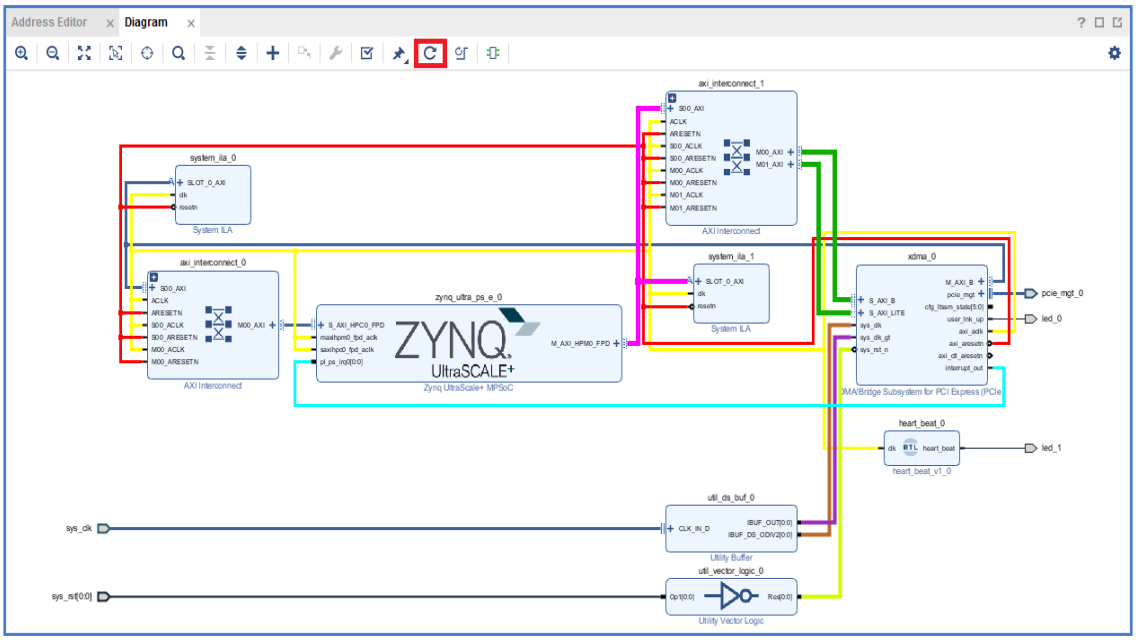


Figure 72 - Regenerate block design

Click on the “Validate Design” icon to check for any violations in IP block design and addressing.

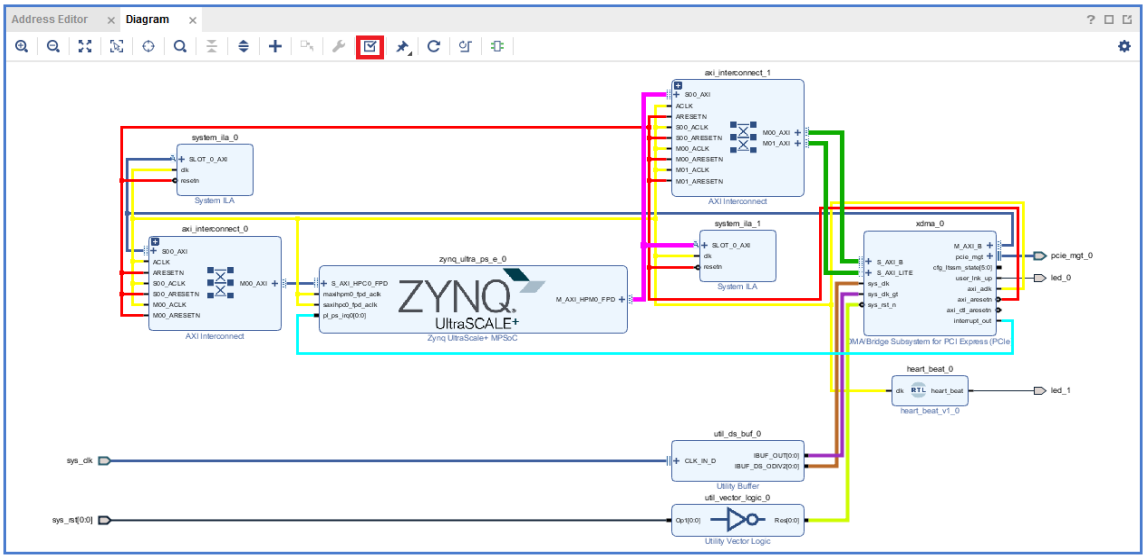


Figure 73 - Validate Design

A message dialog box will confirm that the block design and address assignments are valid.

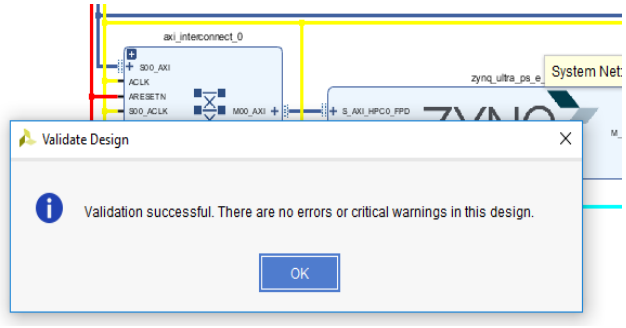


Figure 74 - Successful validation confirmation

Under "Sources" right-click on “Constraints” and select "Add Sources..." from the drop-down list as shown in the Figure 75. Alternatively, click on "Add Sources" under the Flow Navigator pane which is located on the left part of the Vivado interface.

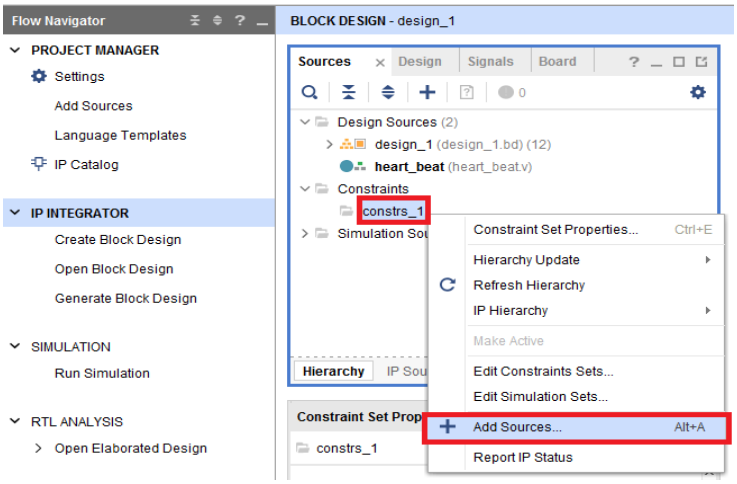


Figure 75 - Adding the constraint file

Select the “Add or create constraints” radio button and click “Next”.

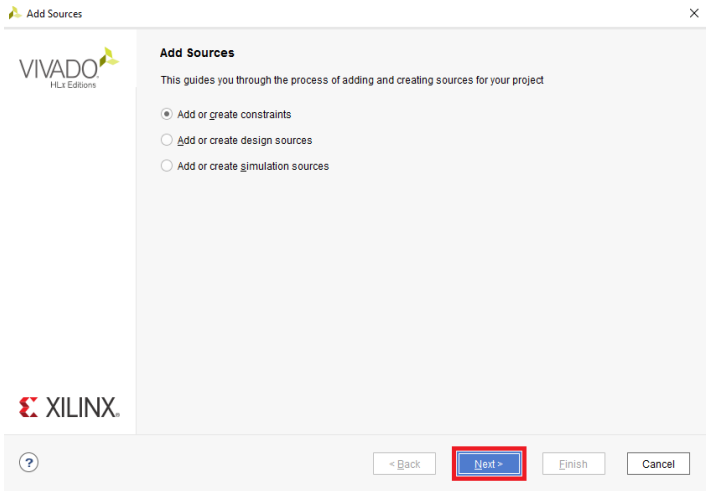


Figure 76 - Create constraint file

Click on “Create File”.

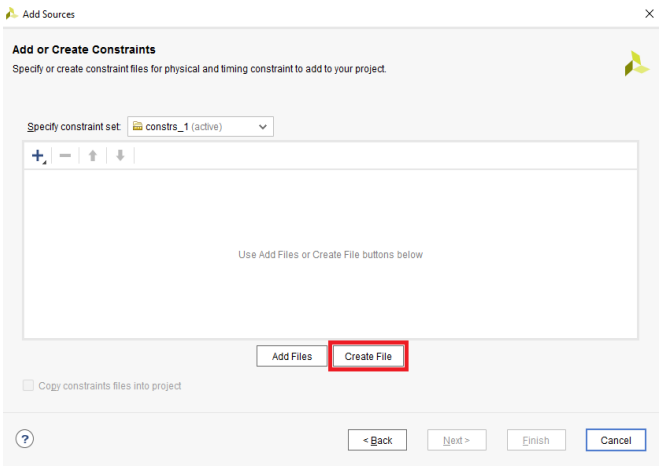


Figure 77 - Create constraint file

In the File name field, type “top”. Leave the File location as <Local to project>. Click OK.

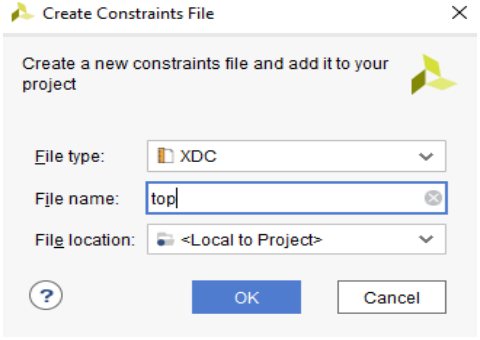


Figure 78 - Enter file name

Click Finish to create the design source file.

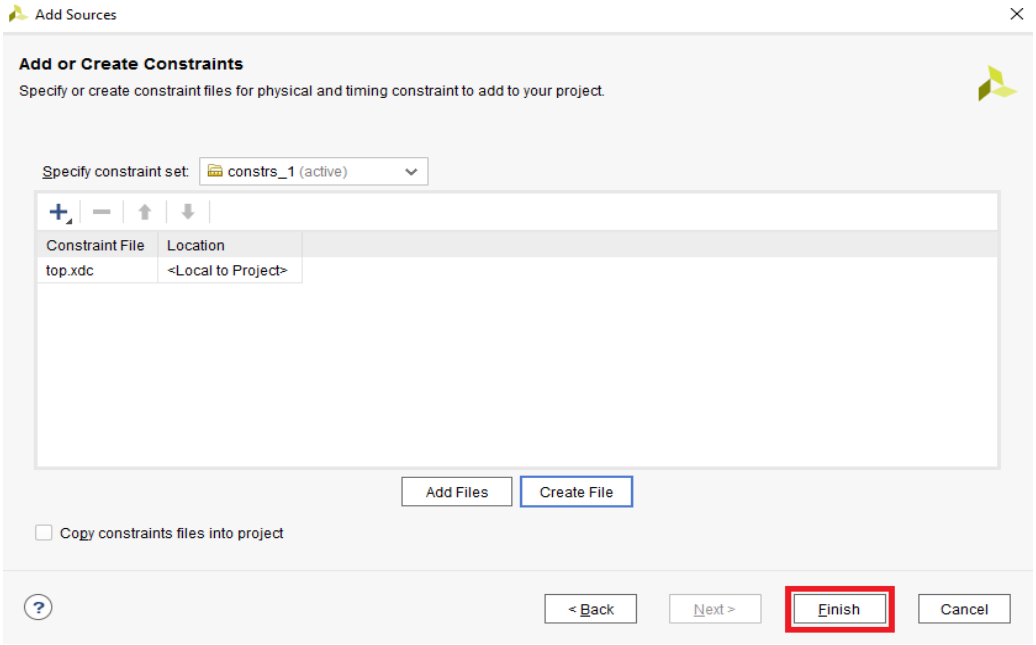


Figure 79 - Create constraint file

Under "Sources" expand “Constraints” and double-click on **top.xdc** to open a code window.

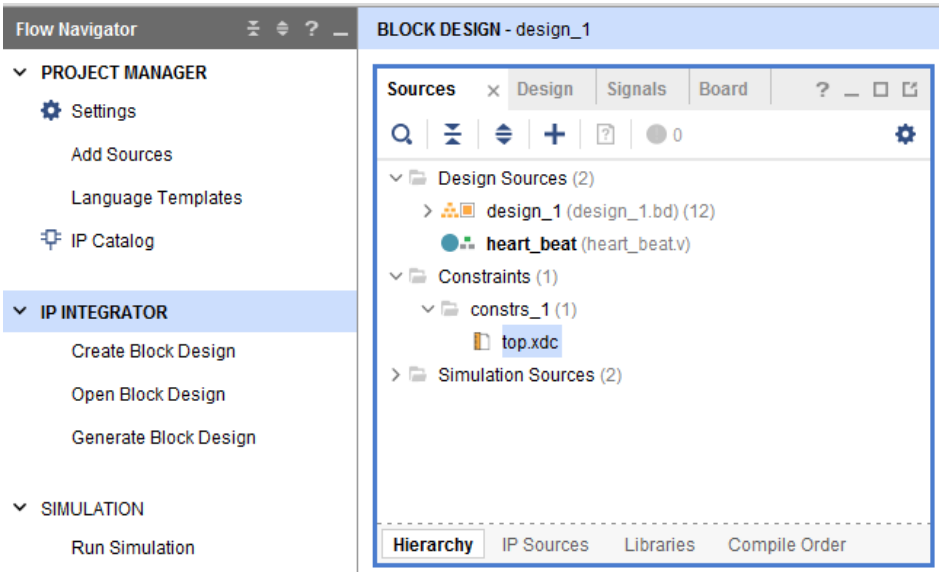


Figure 80 - Open the constraint file

Enter the constraint in “Appendix A: ZCU106 Constraints”.

```
## Project    : The Xilinx PCI Express DMA
## File       : xilinx_xdma_pcie_x0y1.xdc
## Version    : 4.0
##-----------------------------------------------------------------------------
#
# User Configuration
# Link Width   - x1
# Link Speed   - Gen2
# Family       - zynquplus
# Part         - xczu7ev
# Package      - ffvc1156
# Speed grade  - -2
##############################################################################
# User Time Names / User Time Groups / Time Specs
##############################################################################
##
## Free Running Clock is Required for IBERT/DRP operations.
##

##############################################################################################################
create_clock -name sys_clk -period 10 [get_ports sys_clk_clk_p]
#create_clock -name sys_clk_125 -period 8 [get_ports sys_clk_125_clk_p]
#
##############################################################################################################
set_false_path -from [get_ports sys_rst]
set_property PULLUP true [get_ports sys_rst]
#
#set_property CONFIG_VOLTAGE 1.8 [current_design]
#
##############################################################################################################
#set_property LOC [get_package_pins -of_objects [get_bels [get_sites -filter {NAME =~ *COMMON*} -of_objects [get_iobanks -of_objects [get_sites GTHE4_CHANNEL_X0Y19]]/REFCLK0P]] [get_ports
CLK_IN_D_0_clk_p]
#set_property LOC [get_package_pins -of_objects [get_bels [get_sites -filter {NAME =~ *COMMON*} -of_objects [get_iobanks -of_objects [get_sites GTHE4_CHANNEL_X0Y19]]/REFCLK0N]] [get_ports
CLK_IN_D_0_clk_n]
#
set_property LOC V8 [get_ports sys_clk_clk_p]
set_property LOC V7 [get_ports sys_clk_clk_n]

#set_property LOC H9 [get_ports sys_clk_125_clk_p]
#set_property IOSTANDARD  LVDS [get_ports "sys_clk_125_clk_p"] ;

##############################################################################################################
#
set_property PACKAGE_PIN AL11     [get_ports "led_0"] ;# LED 0 Bank  66 VCCO - VCC1V2   - IO_L8P_T1L_N2_AD5P_66
set_property IOSTANDARD  LVCMOS12 [get_ports "led_0"] ;# LED 0 Bank  66 VCCO - VCC1V2   - IO_L8P_T1L_N2_AD5P_66
set_property PACKAGE_PIN AL13     [get_ports "led_1"] ;# LED 1 Bank  66 VCCO - VCC1V2   - IO_L7N_T1L_N1_QBC_AD13N_66
set_property IOSTANDARD  LVCMOS12 [get_ports "led_1"] ;# LED 1 Bank  66 VCCO - VCC1V2   - IO_L7N_T1L_N1_QBC_AD13N_66
set_property PACKAGE_PIN AL10     [get_ports "sys_rst"] ;# Bank  66 VCCO - VCC1V2   - IO_L8N_T1L_N3_AD5N_66
set_property IOSTANDARD  LVCMOS12 [get_ports "sys_rst"] ;# Bank  66 VCCO - VCC1V2   - IO_L8N_T1L_N3_AD5N_66
#set_property PACKAGE_PIN F17      [get_ports "sys_reset"] ;# Bank  66 VCCO - VCC1V2   - IO_L8N_T1L_N3_AD5N_66
#set_property IOSTANDARD  LVCMOS18 [get_ports "sys_reset"] ;# Bank  66 VCCO - VCC1V2   - IO_L8N_T1L_N3_AD5N_66

#set_property PACKAGE_PIN AH17     [get_ports "UART_0_rxd"]
#set_property PACKAGE_PIN AL17     [get_ports "UART_0_txd"]
```

Figure 81 - Constraint file

Click on the “save” icon to save the constraint file.

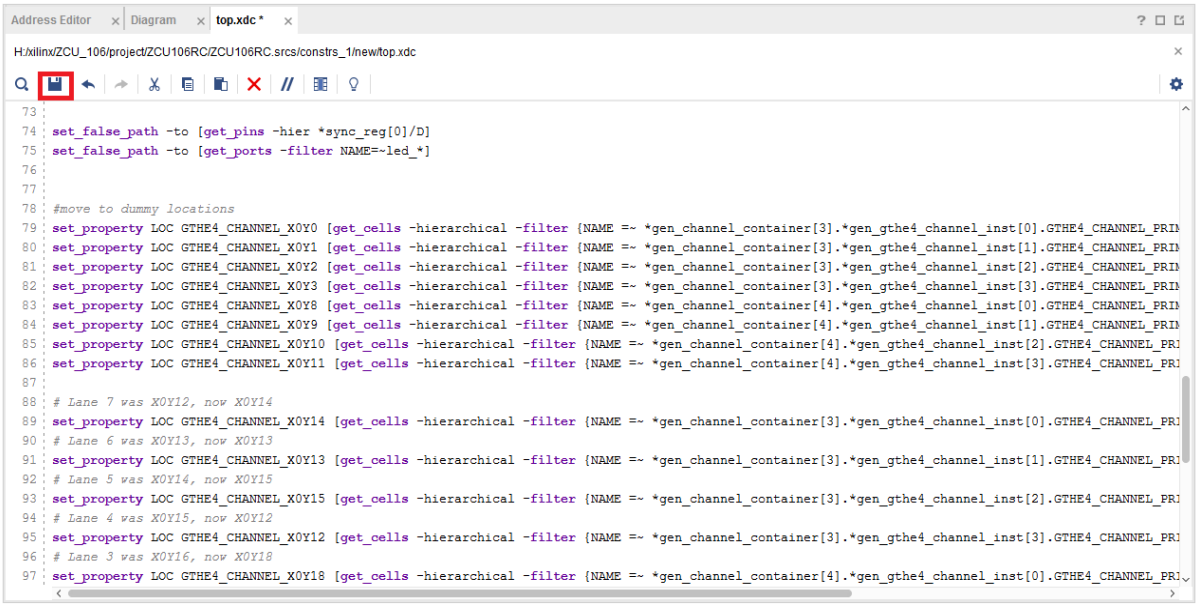


Figure 82 - Constraints

Right-click on **design_1(design_1.bd)(12)** under "Sources" and select "Create HDL Wrapper...".

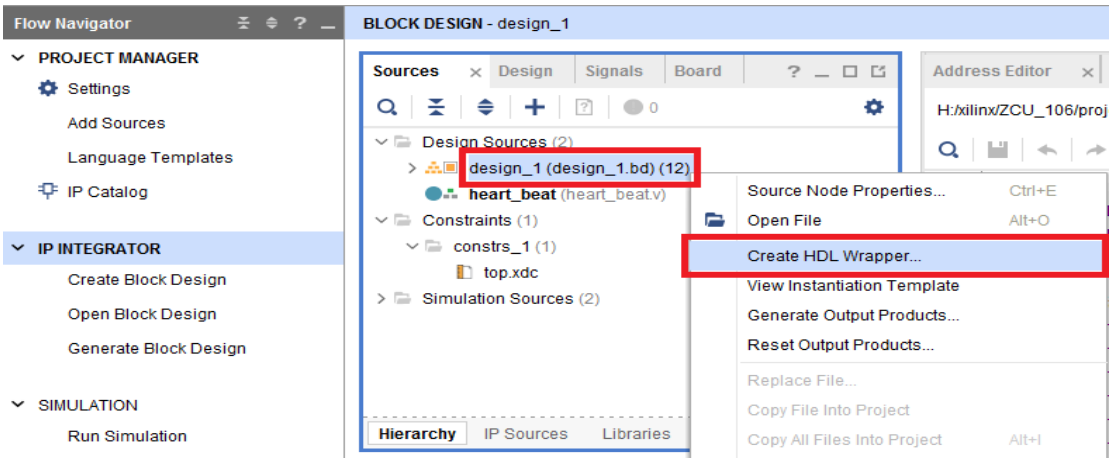


Figure 83 - HDL wrapper

Select the "Let Vivado manage wrapper and auto-update" radio button and click OK.

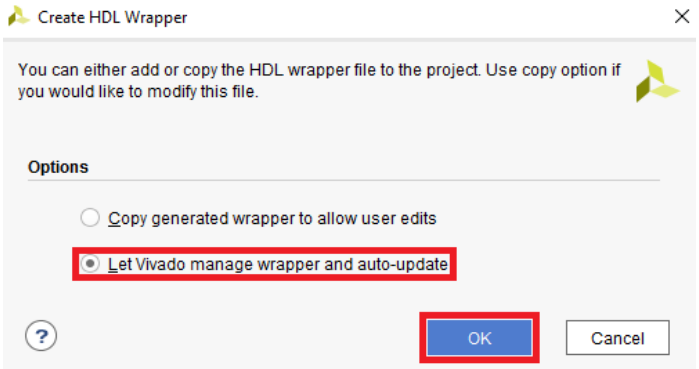


Figure 84 - HDL wrapper

Figure 85 - HDL wrapper shows that a Verilog wrapper file was successfully created.

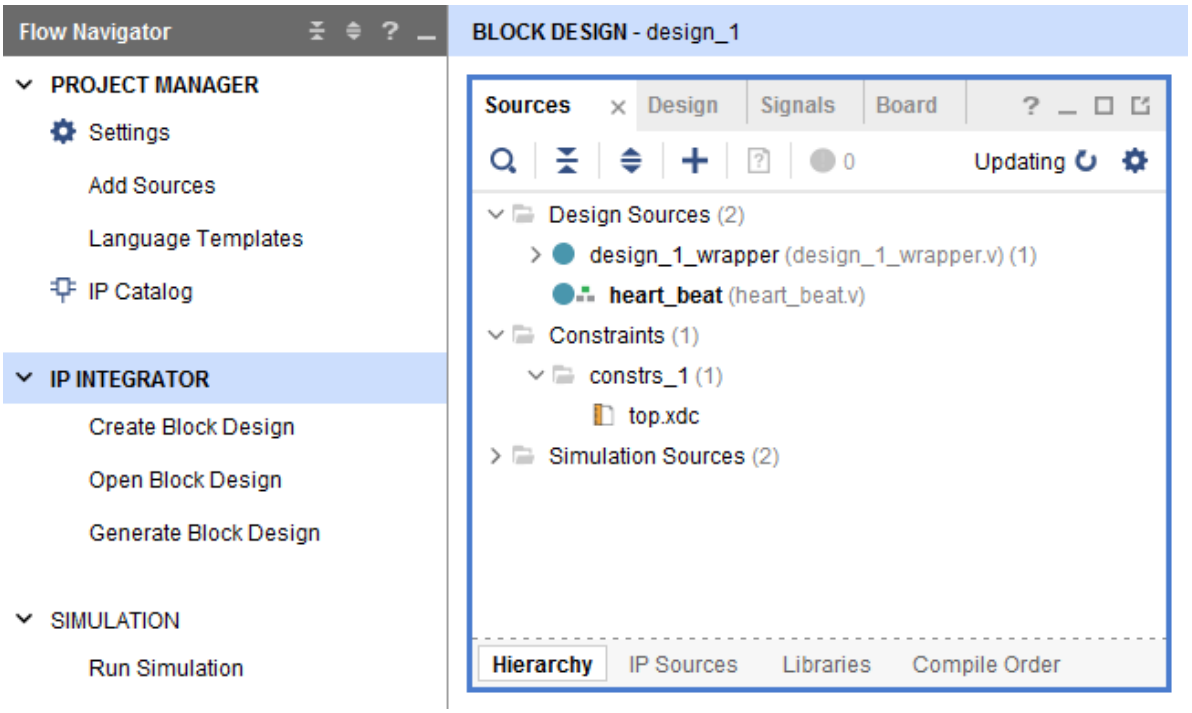


Figure 85 - HDL wrapper

Click “Run Synthesis” under Flow Navigator.

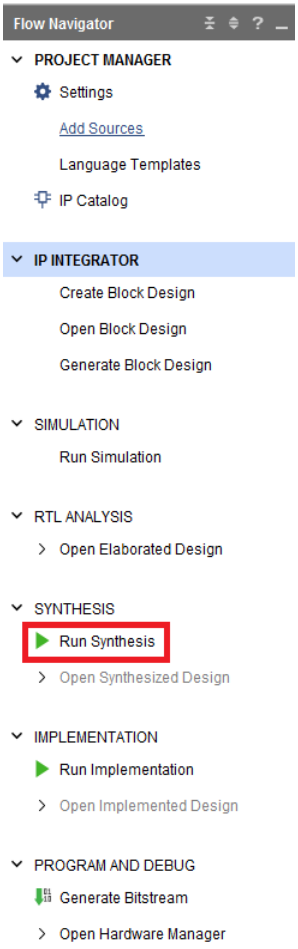


Figure 86 - Synthesis

Use the default settings for “Launch Runs”. Click “OK”

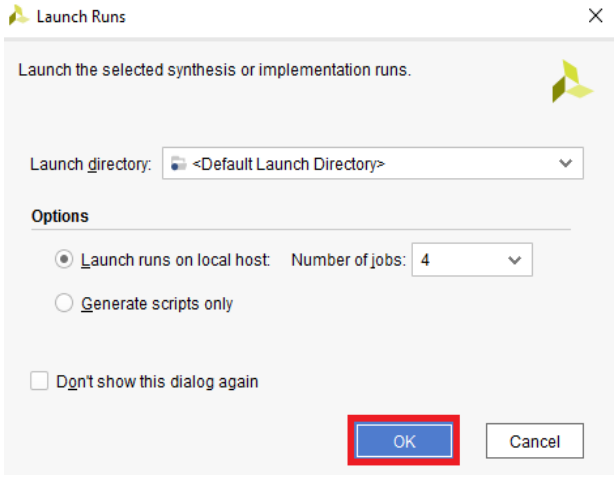


Figure 87 - Synthesis

The synthesis progress bar is shown at the top right corner of the Vivado interface. More details on the status of the synthesizing design are displayed under the “Design Runs” tab as shown in Figure 88.

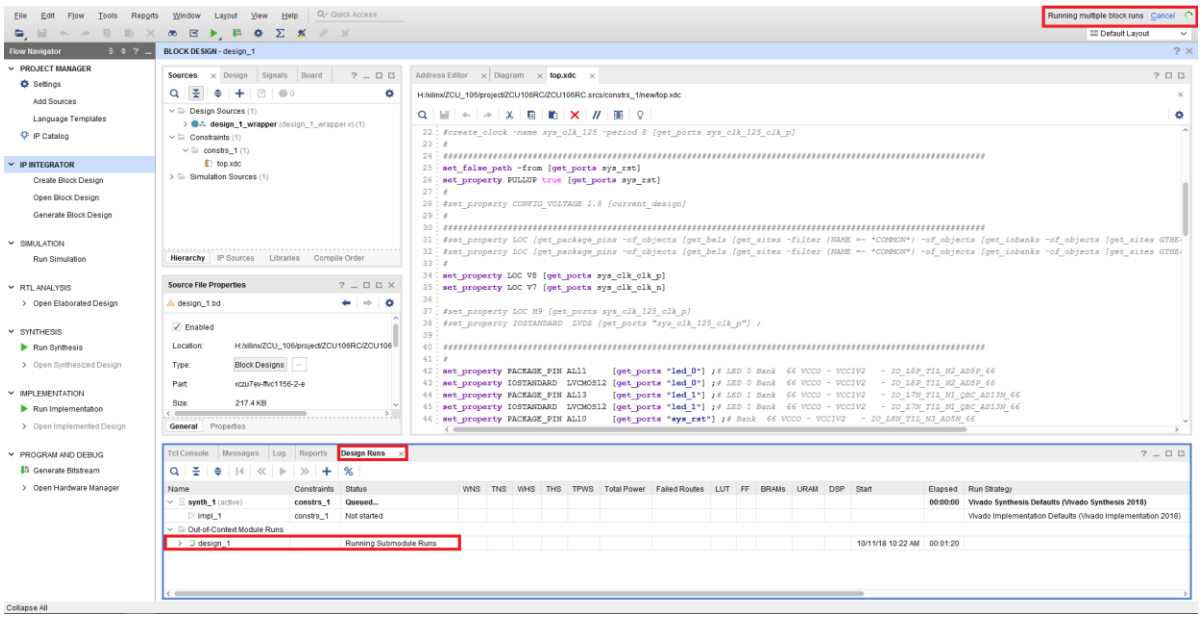


Figure 88 - Synthesis

A message dialog box will confirm that synthesis is complete. Select the “Run Implementation” radio button and click OK.

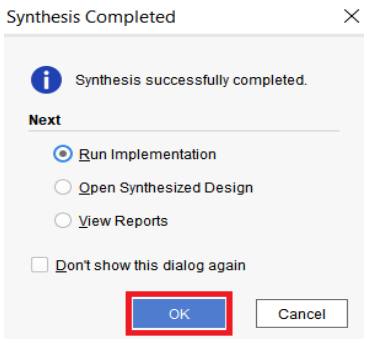


Figure 89 - Implementation

At the top right corner of the Vivado interface, you can see that the design is being implemented. The status of implementation is shown under the “Design Runs” tab.

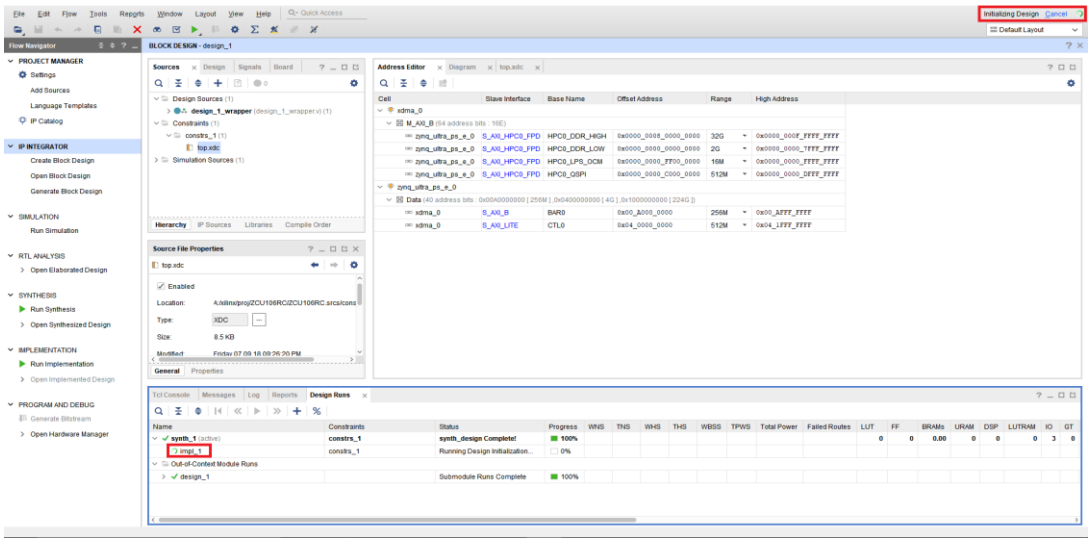


Figure 90 - Implementation

A message dialog box will appear indicating that implementation was successful. Select “Generate Bitstream” and click OK.

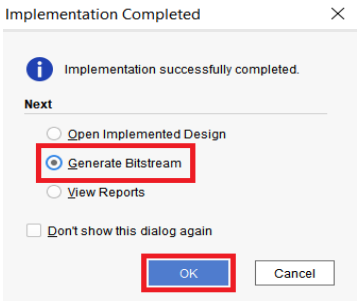


Figure 91 – Bitstream generation

At the top right corner of the Vivado interface, you can see that the design is generating a bitstream. The status of bitstream generation is shown under the “Design Runs” tab.

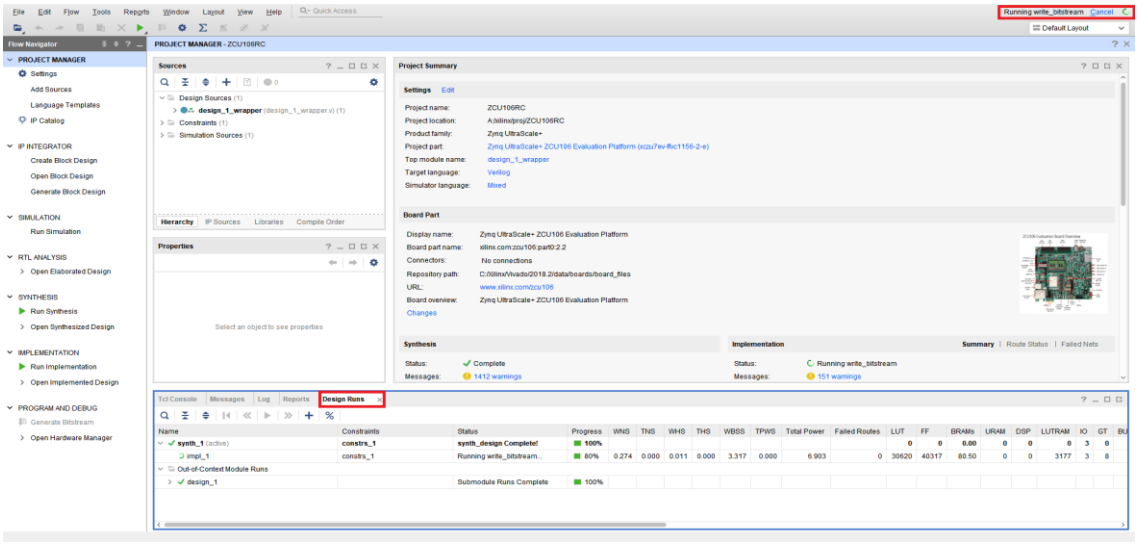


Figure 92 - Bitstream generation

A message dialog box will appear indicating that bitstream generation was successful. Click “Cancel”.

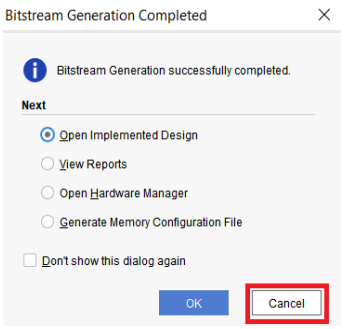


Figure 93 - Bitstream generation

Go to File and select “Export” and “Export Hardware...”

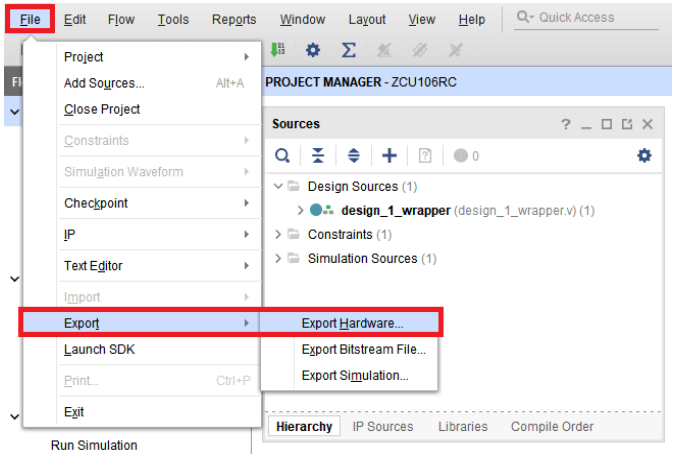


Figure 94 - Export HDF

A message dialog box will appear for exporting the hardware description file. Tick the “Include bitstream” checkbox. Select the location for the hardware description file or use <Local to Project> which will export the HDF file to the “<name of the project>.sdk” folder.

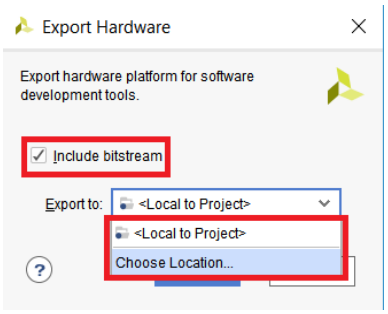


Figure 95 - Export HDF

Click OK.

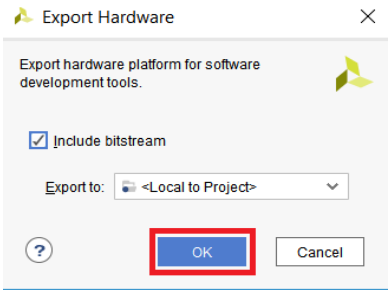


Figure 96 - Export HDF

Configuring a ZCU106 board using PetaLinux

PetaLinux is an embedded Linux development solution for Xilinx Zynq chips. PetaLinux Tools offer everything necessary to customize, build and deploy Embedded Linux solutions on Xilinx processing systems. It consists of three key elements: pre-configured binary bootable images, fully customizable Linux for the Xilinx device, and PetaLinux SDK which includes tools and utilities to automate complex tasks across configuration, build, and deployment. [7]

The primary file needed for image creation is the hardware description file (.hdf) that was exported from the Vivado design. This file will be used to configure the subsystem of the board. After it is completely configured, the root filesystem (rootfs) and kernel will be available for configuration. If these

configurations are successfully done, the project should be ready for building and packaging to create an image file.

If Vivado is not installed in the Linux OS (Operating System) and the user is using a Linux distro installed as a Virtual Machine on a virtualization software (for example, Oracle VM VirtualBox), the user may create a shared folder in the host machine (Windows or Mac) and access this folder from the guest machine.

The following steps will describe the PetaLinux image creation.

Setting up the PetaLinux environment

Note: Create a directory in the Linux OS for the Hardware Description File (.hdf) and the Bitstream (.bit) files of ZCU106 root complex design. For this project, the directory name is *hdf_zcu106rc* (yours could be different). Copy and paste the ".hdf" and ".bit" files to this directory.

Open the file directory of PetaLinux if the location is known, otherwise use the Terminal and echo the tool as described in Figure 98. Open the Terminal and run the command "echo $PETALINUX". Change directory using the syntax "cd <PETALINUX-DIRECTORY>" to the PetaLinux absolute path.

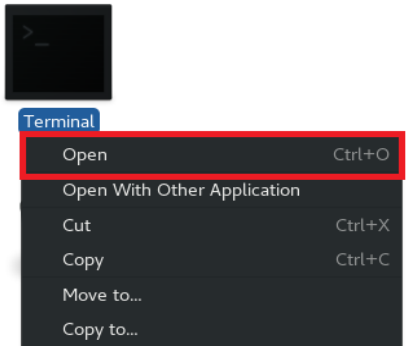


Figure 97 - Linux terminal

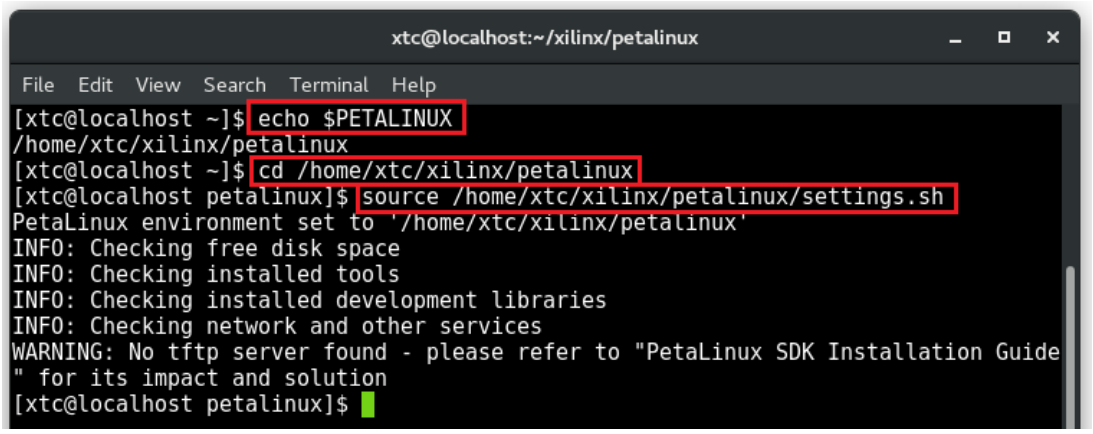


Figure 98 - Initialize image creation tool

Create a project directory using the syntax below.

```
[xtc@localhost petalinux]$ petalinux-create -t project -n ZCU106_RC --template zynqMP
```

Figure 99 - Creating the PetaLinux project

The terminal will display the following message if the project is successfully created.

```
INFO: Create project: ZCU106_RC
INFO: New project successfully created in /home/xtc/xilinx/petalinux/ZCU106_RC
[xtc@localhost petalinux]$
```

Figure 100 - Project directory

Use “cd” to change from the present working directory to the newly created project directory.

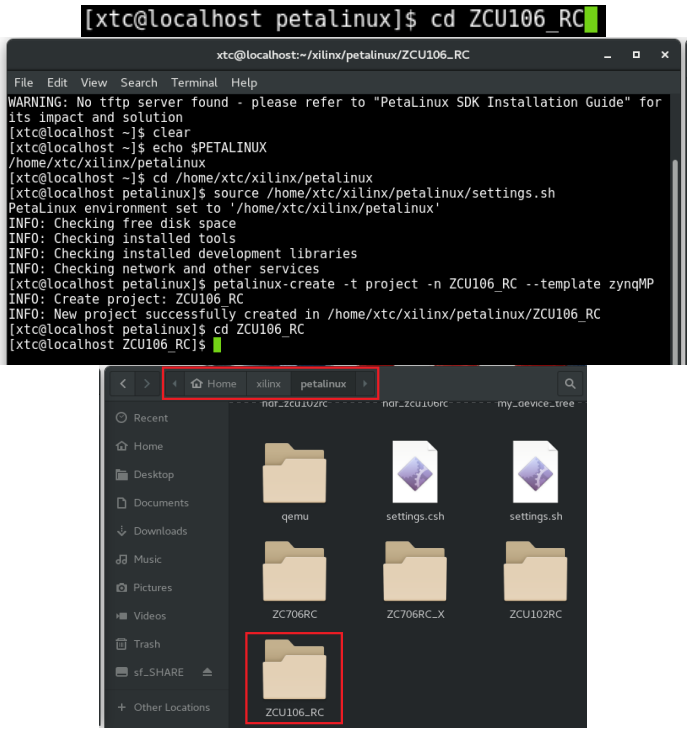


Figure 101 - Change directory

Grab the hardware description file (.hdf) from the project directory using the syntax shown in Figure 102.

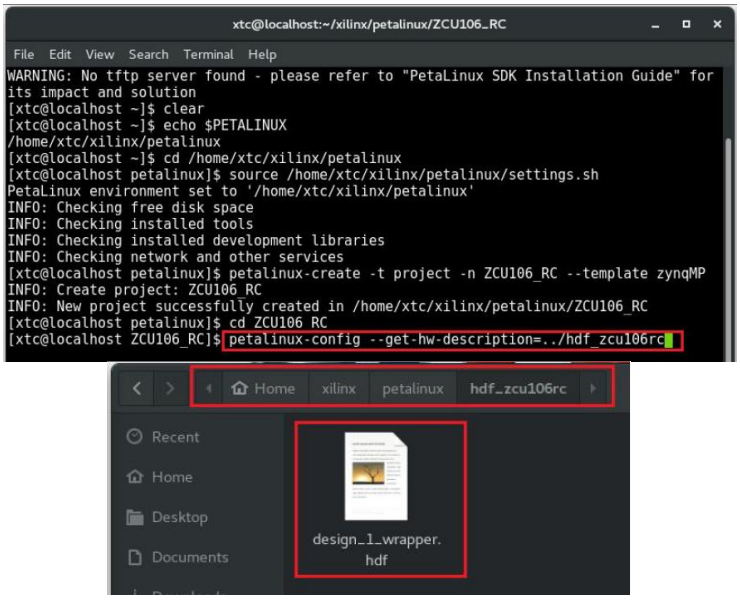


Figure 102 - Importing hardware description file

A message will appear in the Terminal to confirm the validity of the HDF.

```
INFO: Getting hardware description...
INFO: Rename design_1_wrapper.hdf to system.hdf
[INFO] generating Kconfig for project
[INFO] menuconfig project
```

Figure 103 - Information for the .hdf file

In the “misc/config System Configuration” window, select “Subsystem AUTO Hardware Settings”.

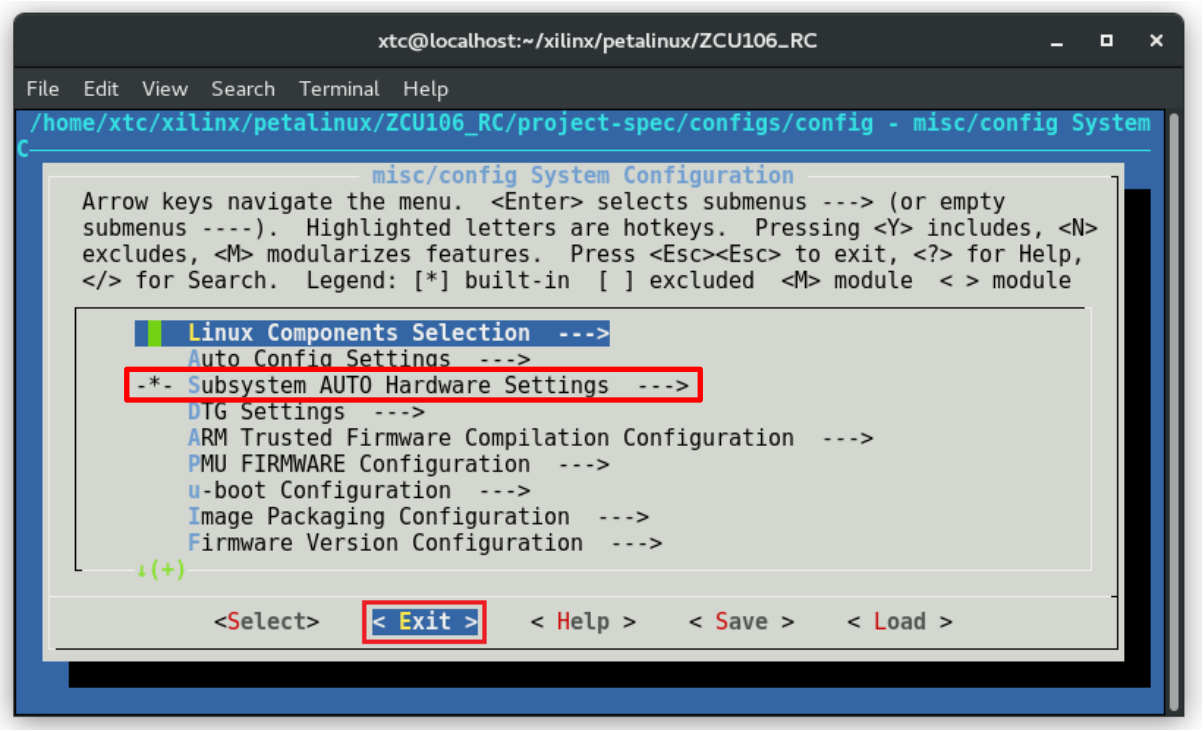


Figure 104 - System configuration

Select <Yes> to exit and save system configuration.

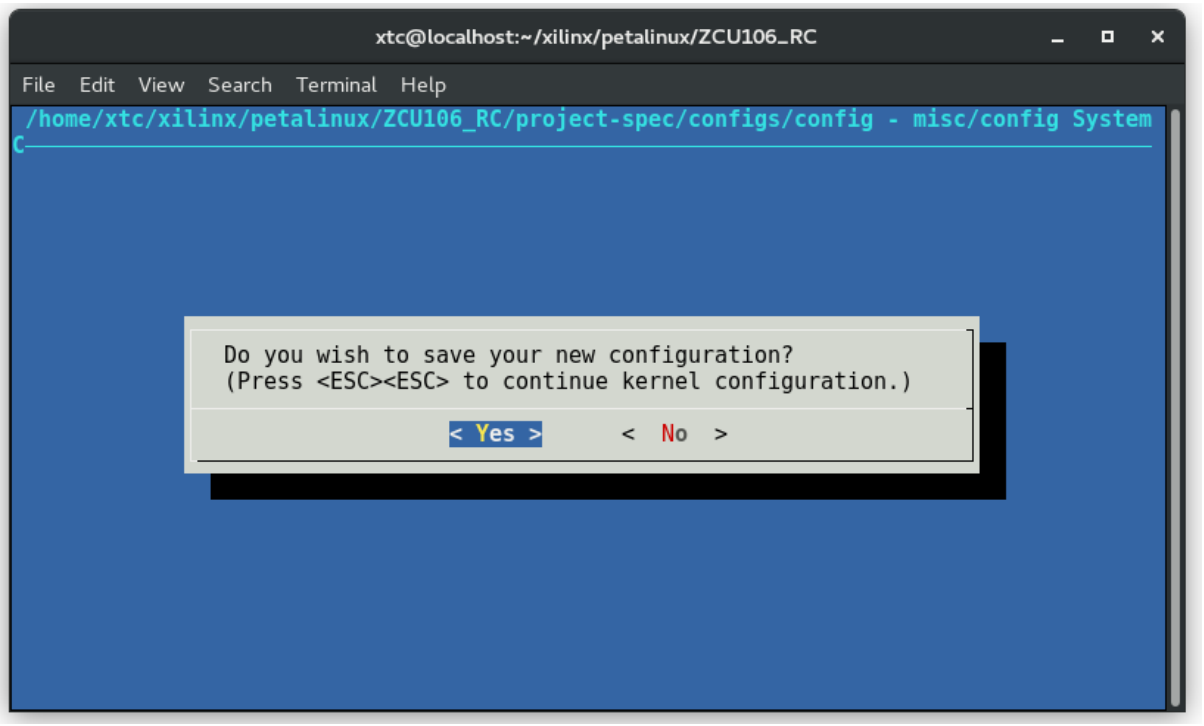


Figure 105 - Saving System configuration

A message will appear in the terminal indicating that the configuration is successful.

```
/home/xtc/xilinx/petalinux/ZCU106_RC/build/misc/config/Kconfig.syshw:30:warning: defaul
ts for choice values not supported
/home/xtc/xilinx/petalinux/ZCU106_RC/build/misc/config/Kconfig:597:warning: config symb
ol defined without type
configuration written to /home/xtc/xilinx/petalinux/ZCU106_RC/project-spec/configs/conf
ig

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

[INFO] sourcing bitbake
[INFO] generating plnxtool conf
[INFO] generating meta-plnx-generated layer
[INFO] generating machine configuration
[INFO] generating bbappends for project . This may take time !
[INFO] generating u-boot configuration files
[INFO] generating kernel configuration files
[INFO] generating kconfig for Rootfs
[INFO] oldconfig rootfs
[INFO] generating petalinux-user-image.bb
```

Figure 106 - Confirmation message for system configuration

Configure the kernel component using the syntax shown below.

```
[xtc@localhost ZCU106_RC]$ petalinux-config -c kernel
[INFO] generating Kconfig for project
[INFO] sourcing bitbake
[INFO] generating plnxtool conf
[INFO] generating meta-plnx-generated layer
[INFO] generating machine configuration
[INFO] configuring: kernel
[INFO] generating kernel configuration files
[INFO] bitbake virtual/kernel -c menuconfig
```

Figure 107 - Kernel configuration

In the “Linux/arm64 4.14.0 Kernel Configuration” window, select “Bus support”.

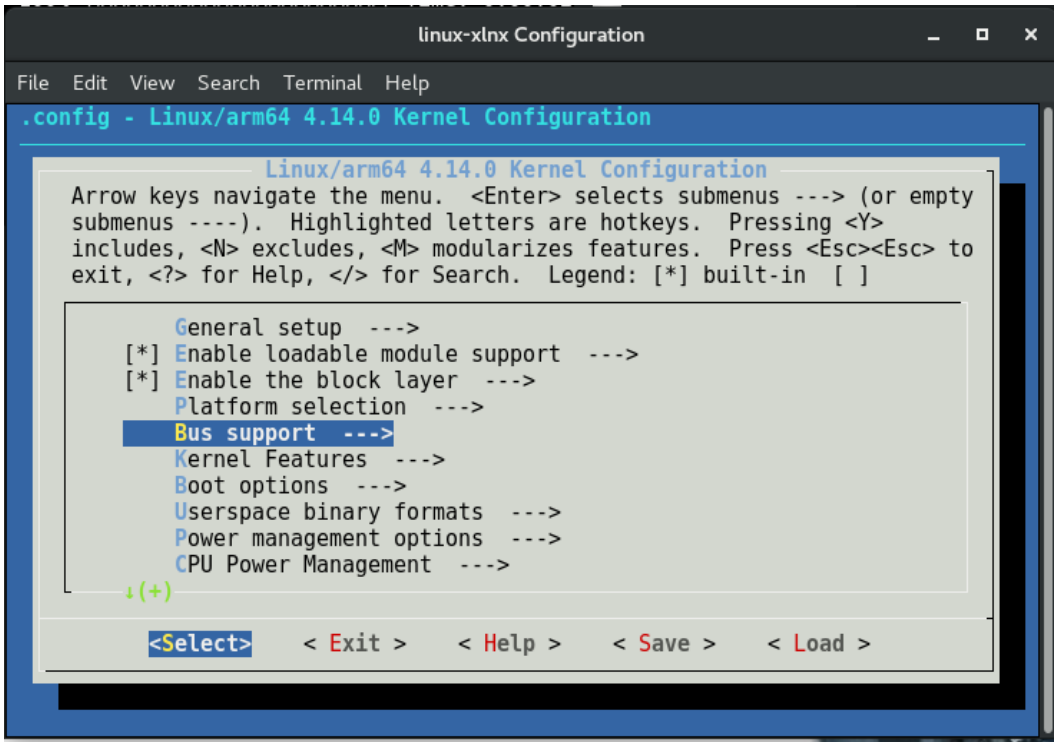


Figure 108 - Loaded kernel window

In the “Bus support” window, select “PCI host controller drivers”.

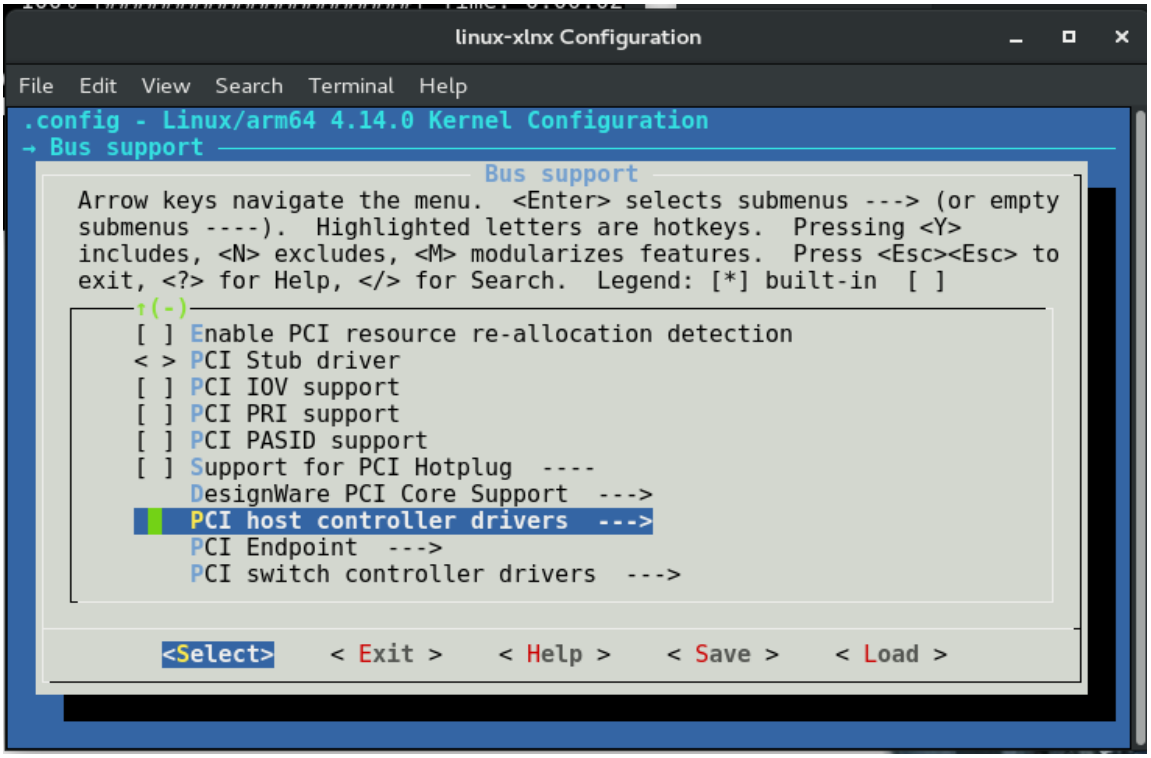


Figure 109 – Enter PCI host controller drivers

Press <Y> to include “Xilinx XDMA PL PCIe host bridge support” under “PCI host controller drivers”.

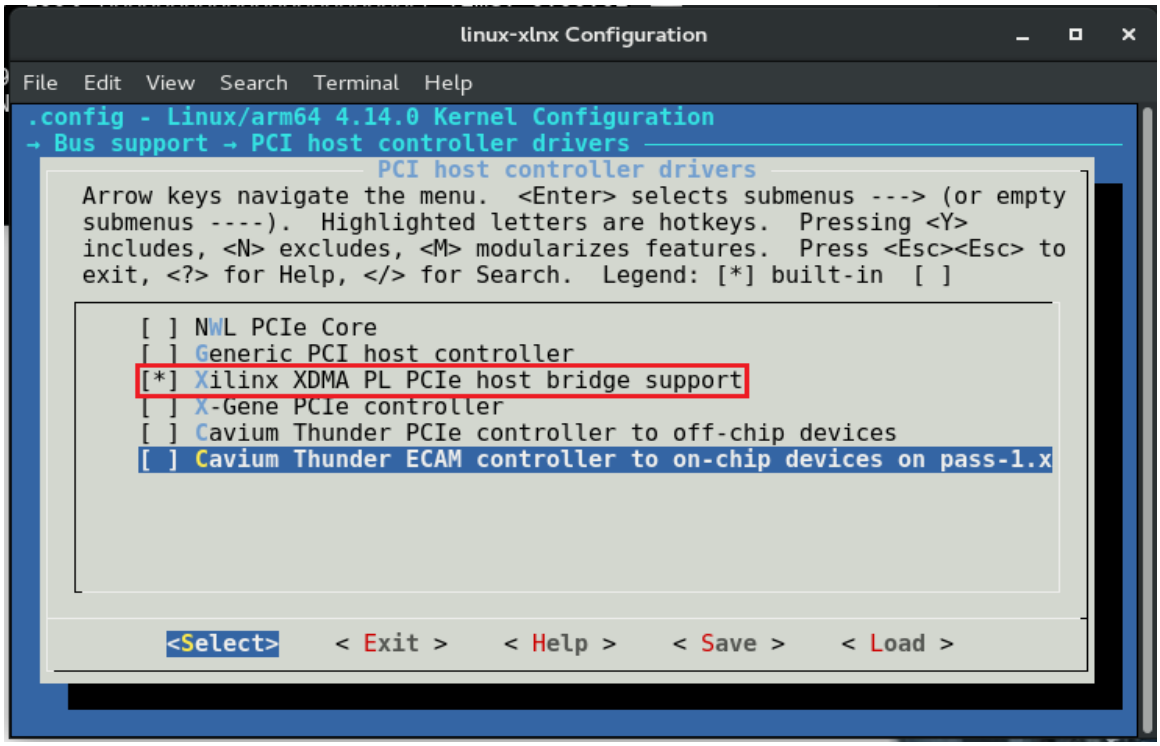


Figure 110 - Include Xilinx XDMA PL PCIe host bridge support

Select <Exit> twice to go back to the “Linux/arm64 4.14.0 Kernel Configuration” window, then select “Device Drivers”.

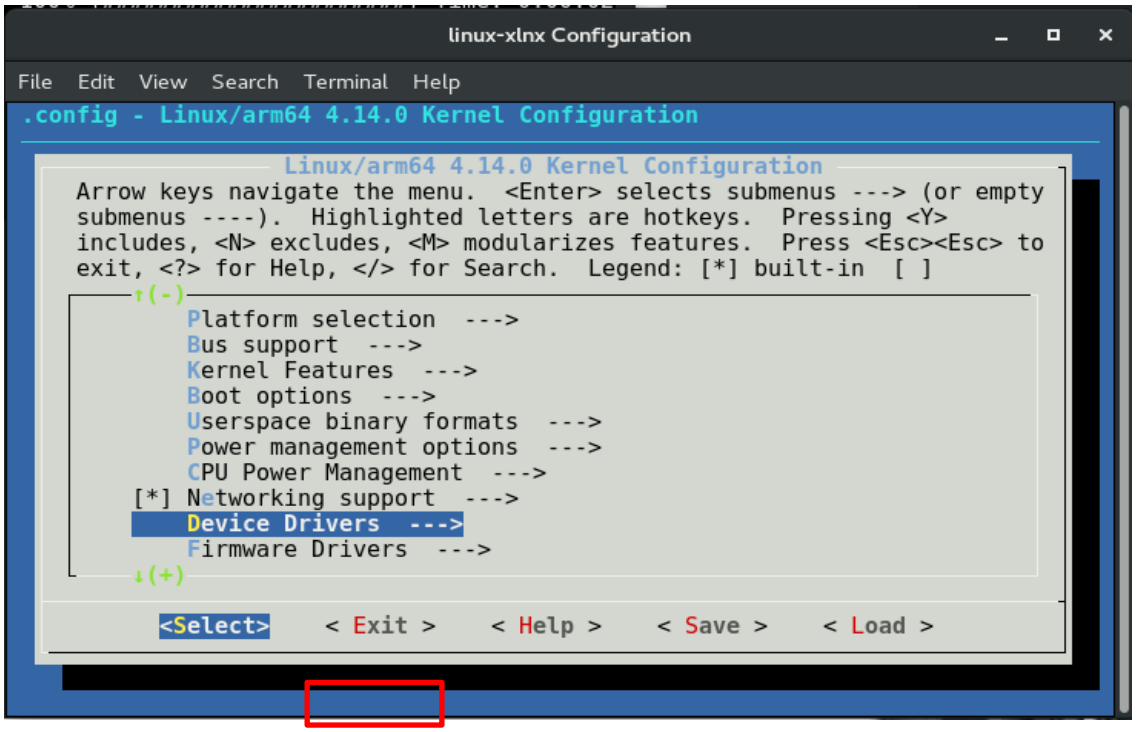


Figure 111 - Kernel configuration – Device Drivers

Press <M> to modularize the feature for “NVM Express block device”, then press <Y> to include it.

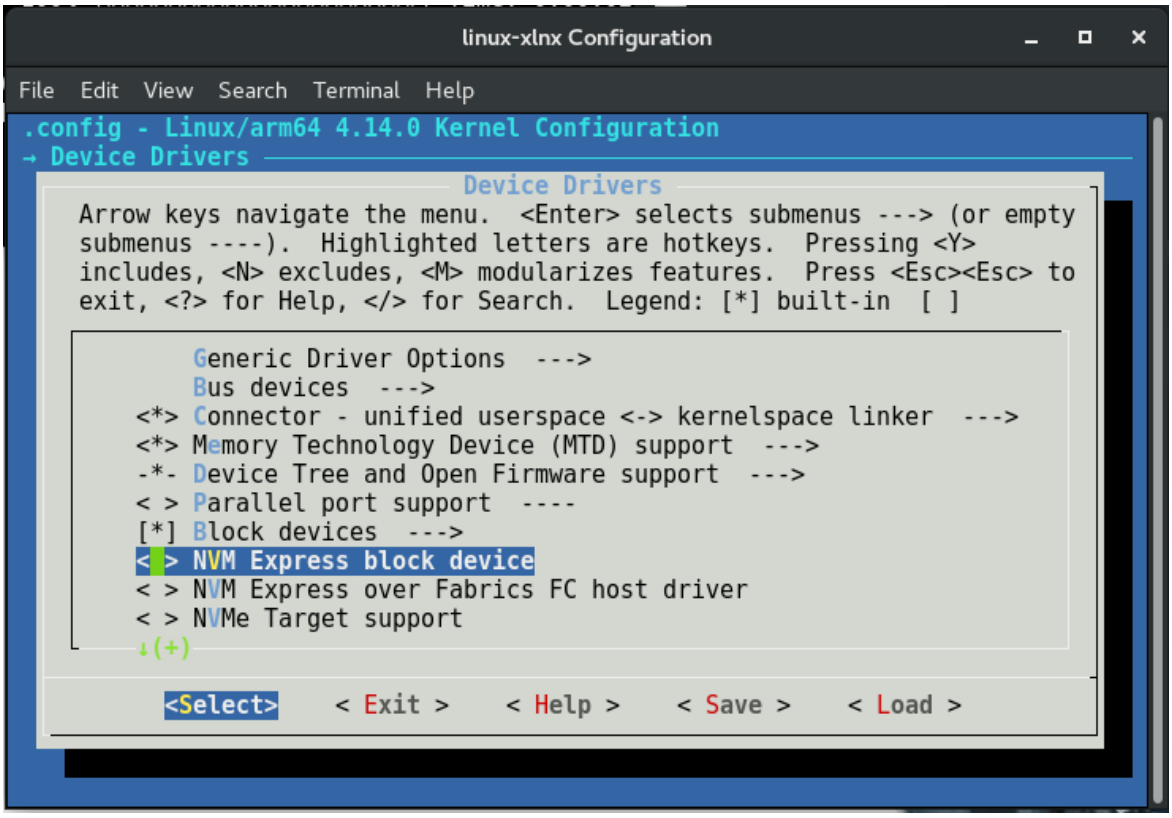


Figure 112 - Kernel configuration – NVM Express block device

Move the cursor to “DMA Engine Support” then press <Y> to include it. Press enter to go to the “DMA Engine Support” configuration window.

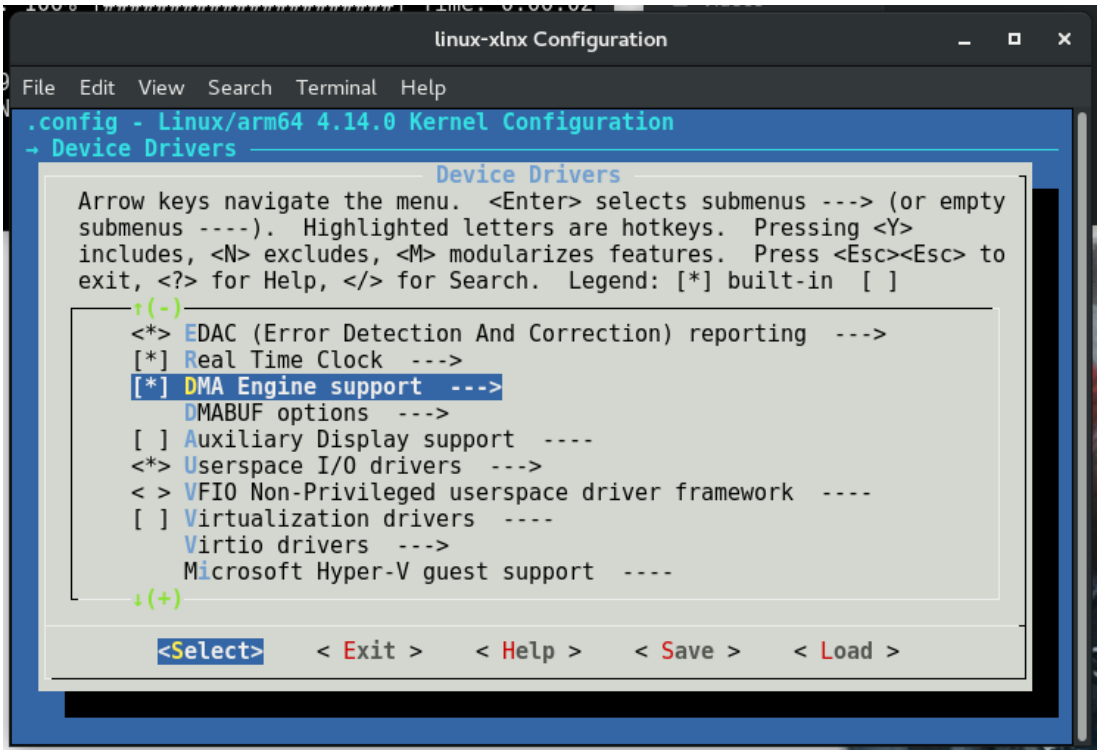


Figure 113 - Kernel configuration – DMA Engine support

Press <M> to modularize the feature for “Xilinx PS PCIe DMA support”, then press <Y> to include it.

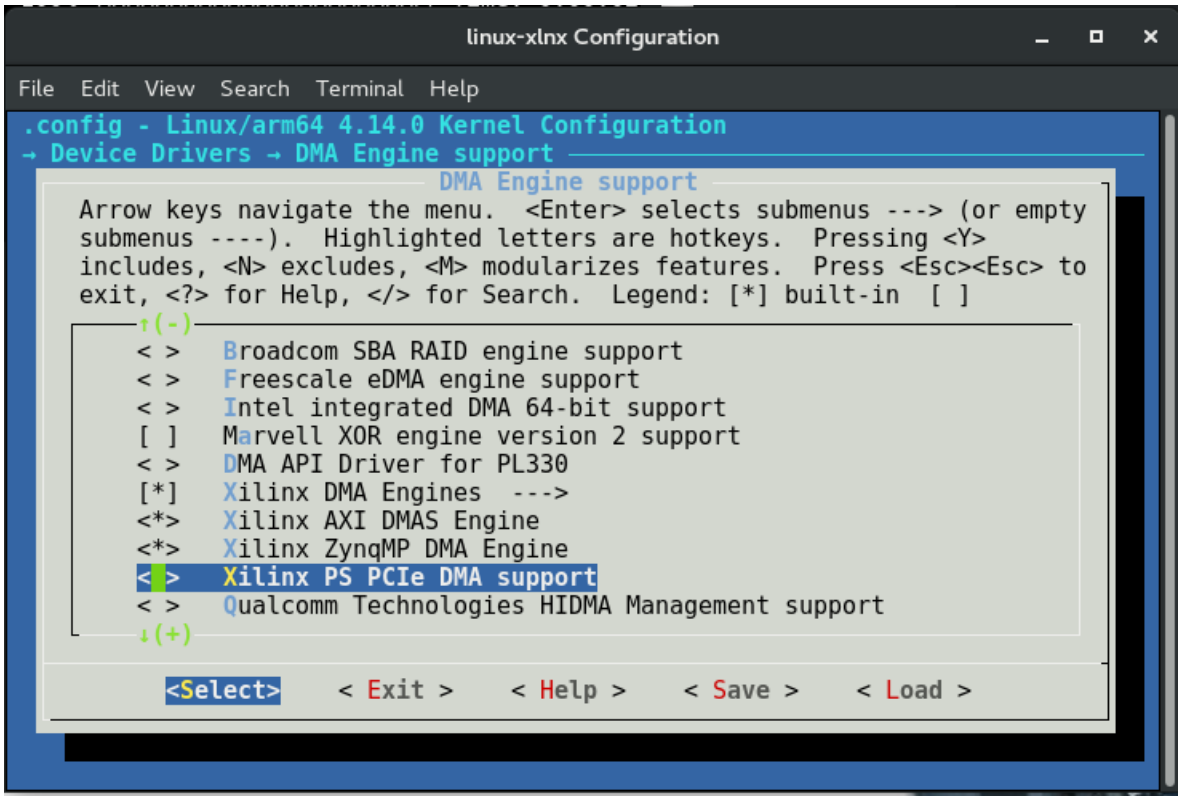


Figure 114 - Kernel configuration - Xilinx PS PCIe DMA support

Press enter to go to the “Xilinx DMA Engines” configuration window.

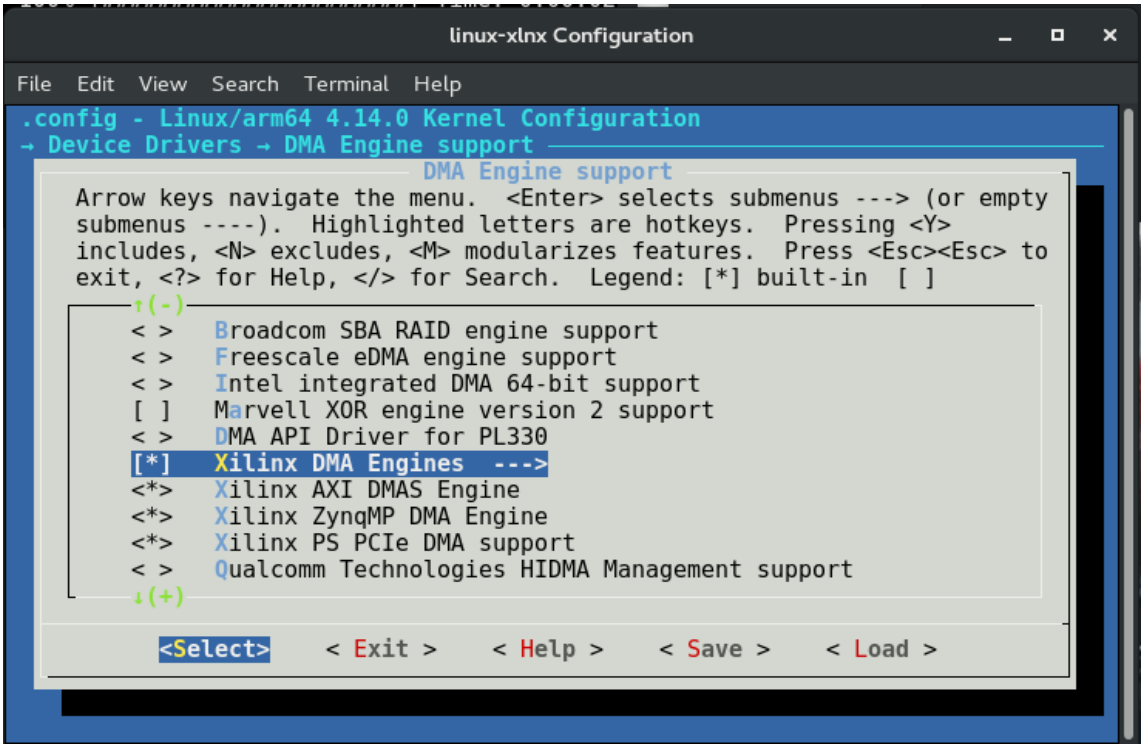


Figure 115 - Kernel configuration - Xilinx DMA Engine

Move the cursor to “Xilinx PS PCIe DMA test client (NEW)” and press <M> to modularize it then <Y> to include it.

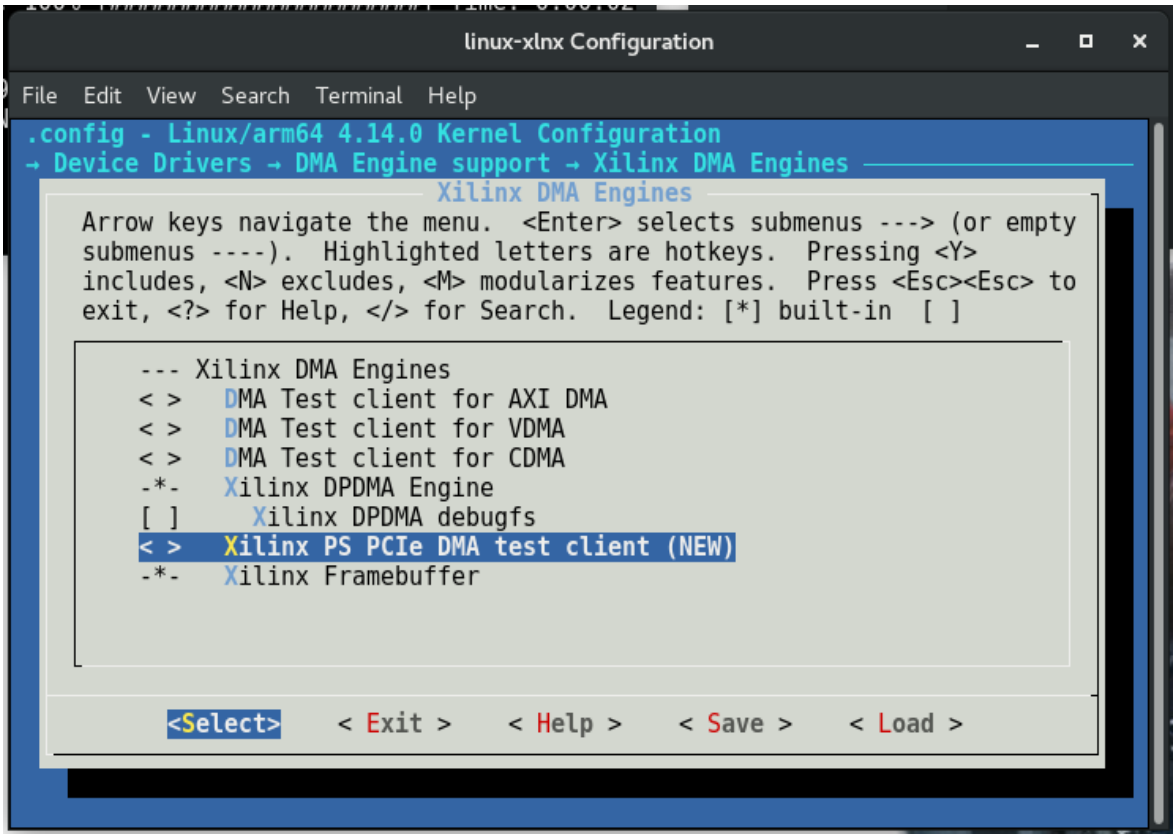


Figure 116 - Kernel configuration - Xilinx PS PCIe DMA test client (NEW)

Continue pressing <Exit> to go back to “Linux/arm64 4.14.0 Kernel Configuration” window.

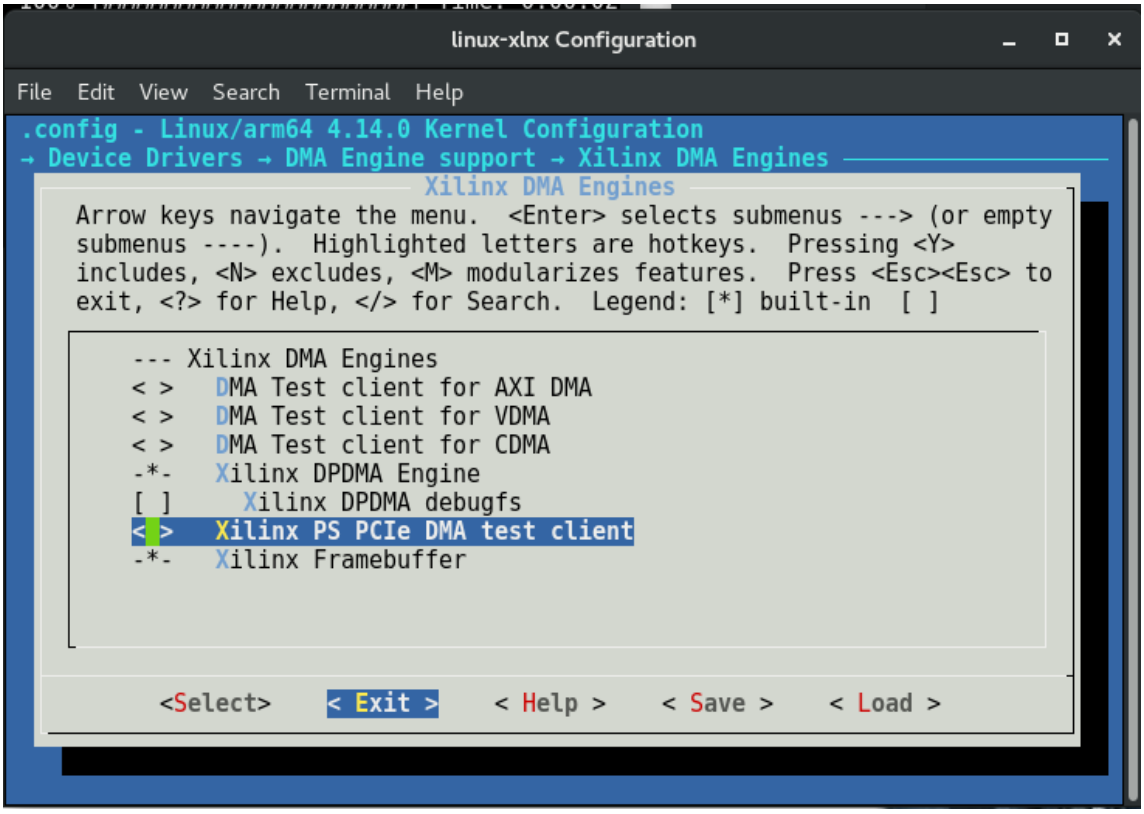


Figure 117 - Exiting Kernel configuration

Exit and save kernel configuration.

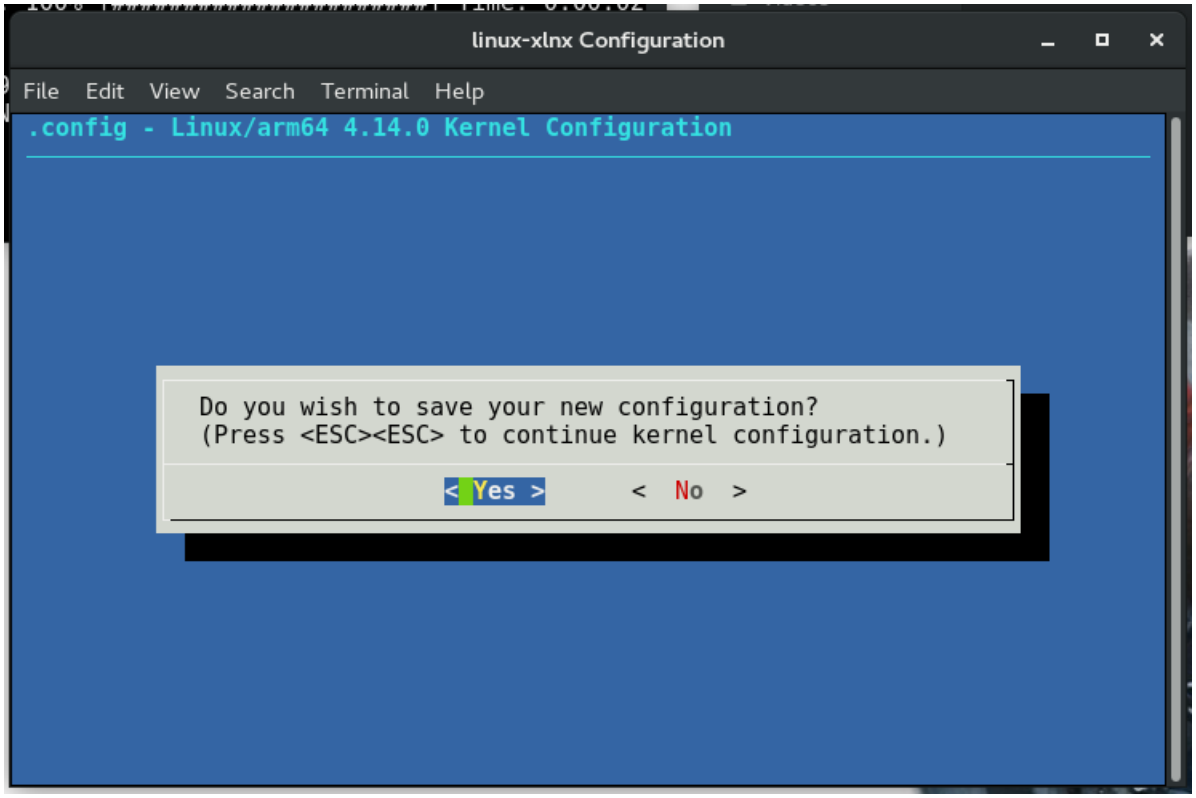


Figure 118 - Saving Kernel configuration

A message will appear indicating that the kernel configuration was successful.

```
Loading cache: 100% |################################################| Time: 0:00:01
Loaded 3437 entries from dependency cache.
Parsing recipes: 100% |##############################################| Time: 0:00:16
Parsing of 2552 .bb files complete (2512 cached, 40 parsed). 3441 targets, 139 skipped,
 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |###########################################| Time: 0:00:14
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 2 tasks of which 0 didn't need to be rerun and all succe
eded.
[INFO] successfully configured kernel
```

Figure 119 - Confirmation message for configured kernel

Configure the component root filesystem using the syntax below.

```
[xtc@localhost ZCU106_RC]$ petalinux-config -c rootfs
```

Figure 120 – Syntax for configuring root filesystem

The following configuration window will appear. Select “Filesystem Packages”.

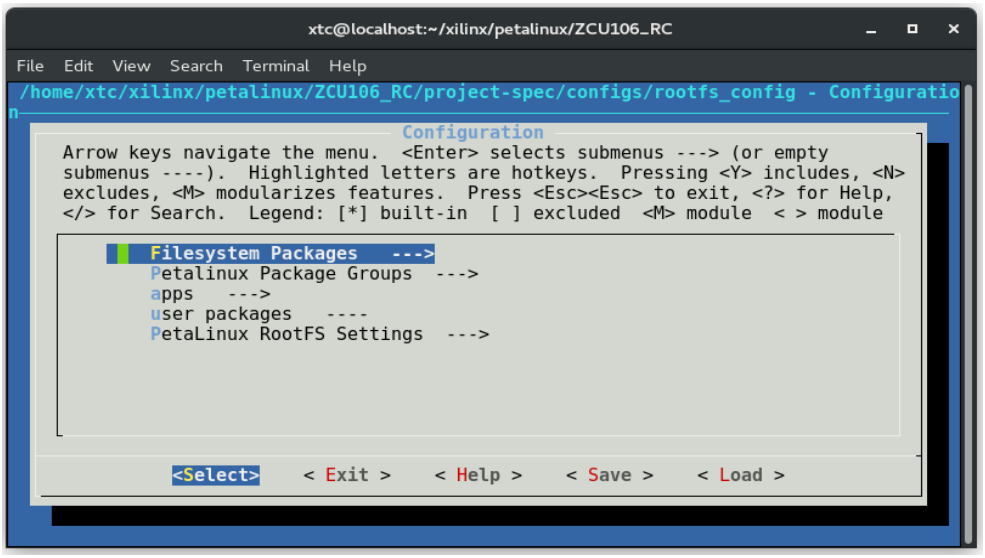


Figure 121 - Root filesystem configuration

Select “base” under “Filesystem Packages”.

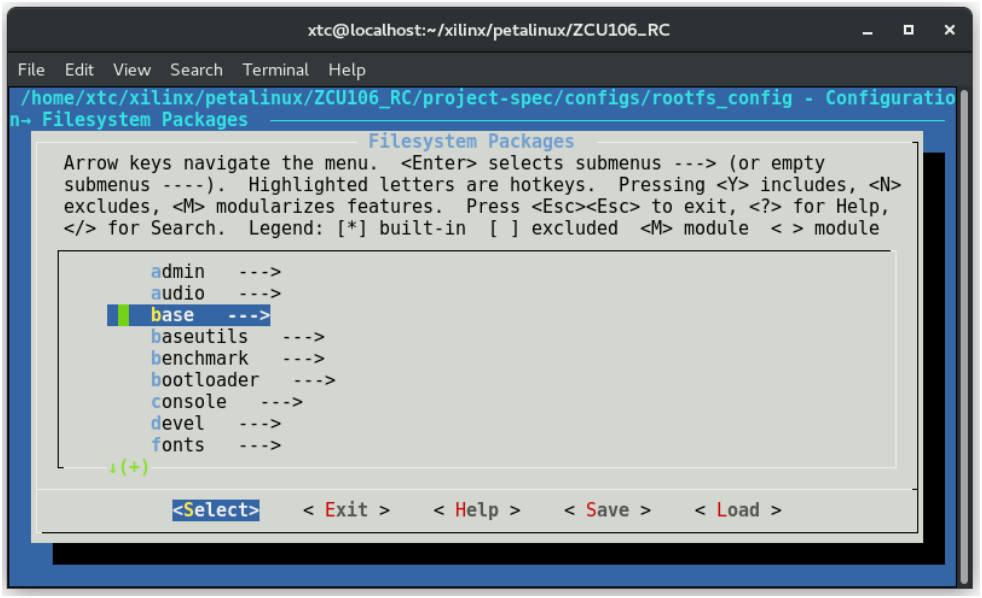


Figure 122 - Filesystem packages

Select “util-linux” under “base”.

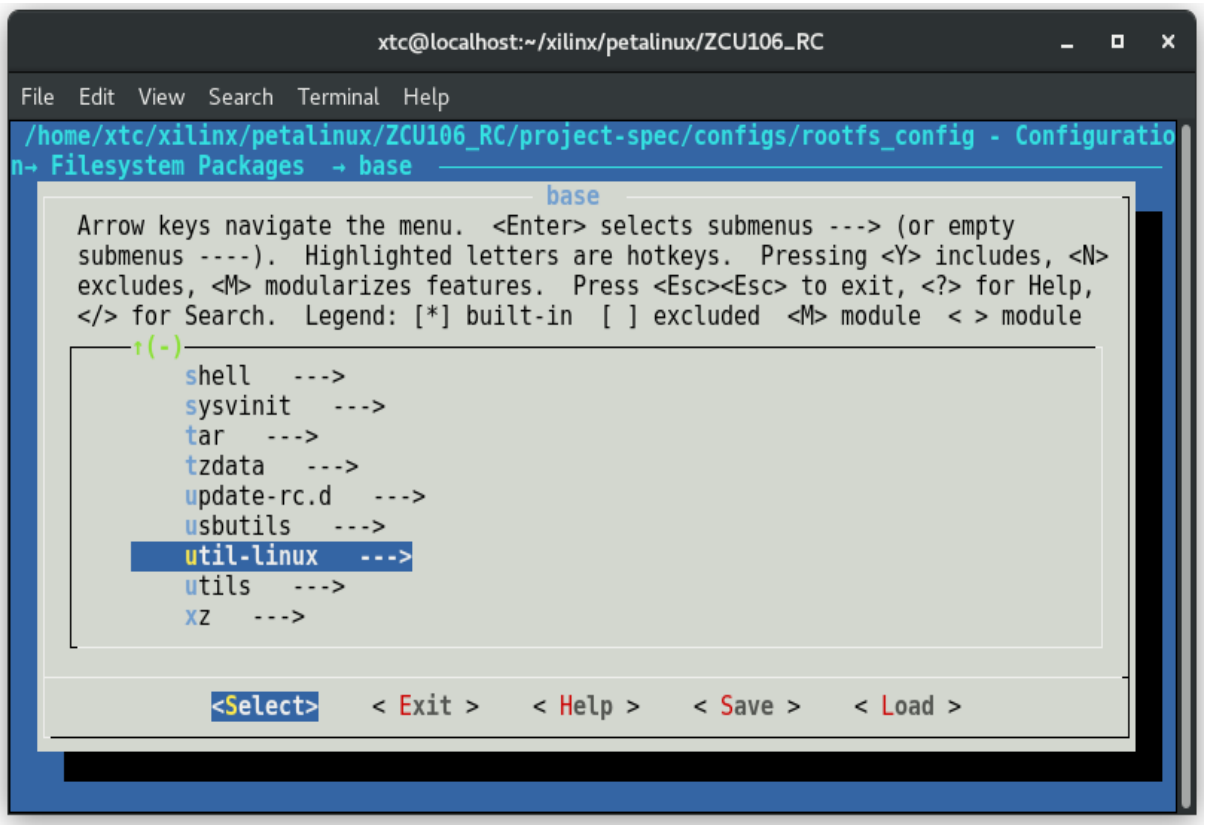


Figure 123 - Enter Util-linux

Press <Y> to include “util-linux”.

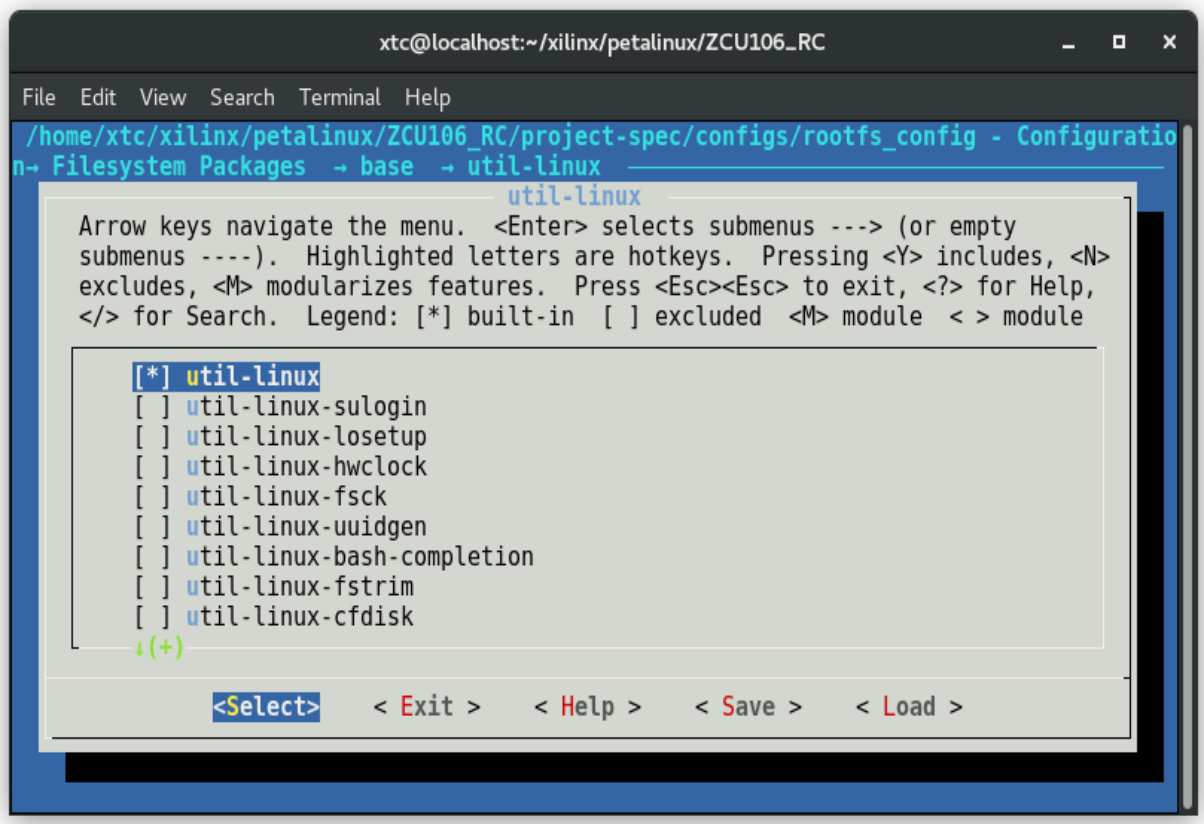


Figure 124 - Filesystem configuration – include util-linux

Press <Y> to include “util-linux mount”.

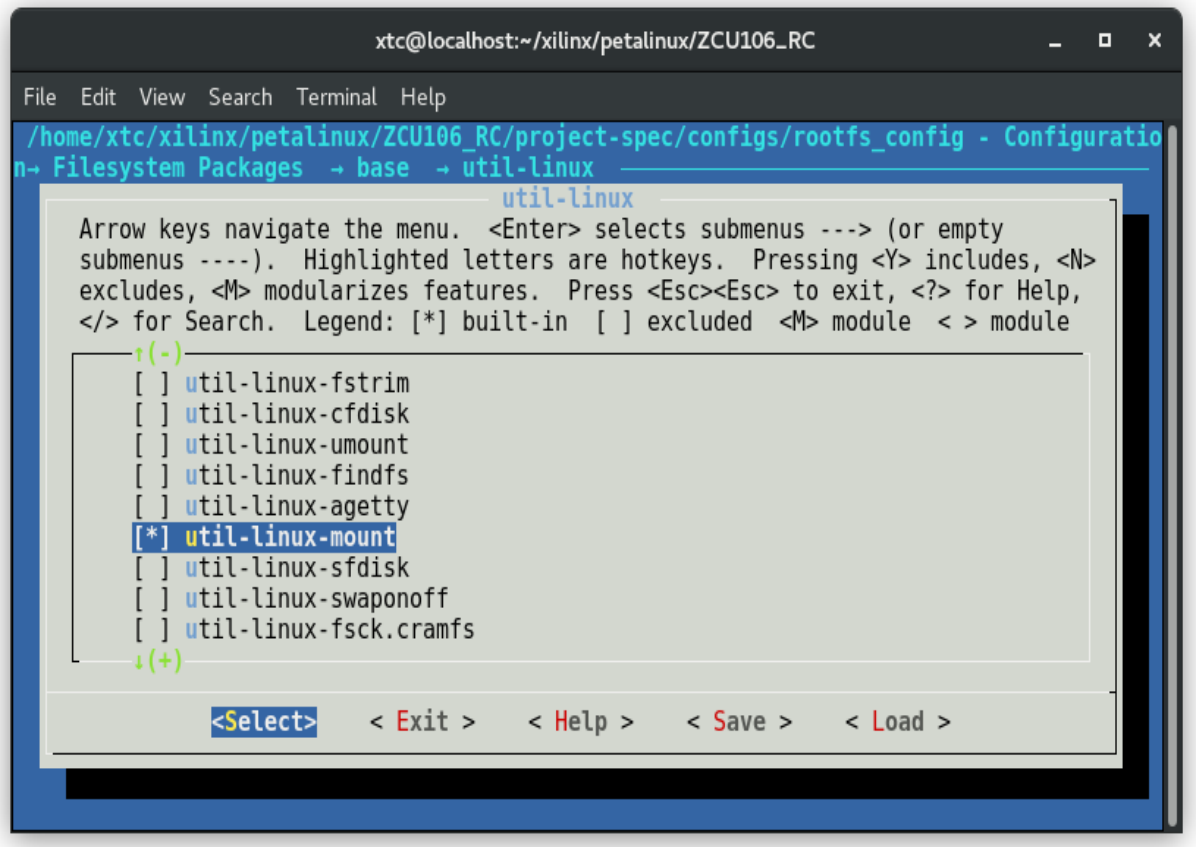


Figure 125 - Include util-linux-mount

Press <Y> to include “util-linux-blkid”.

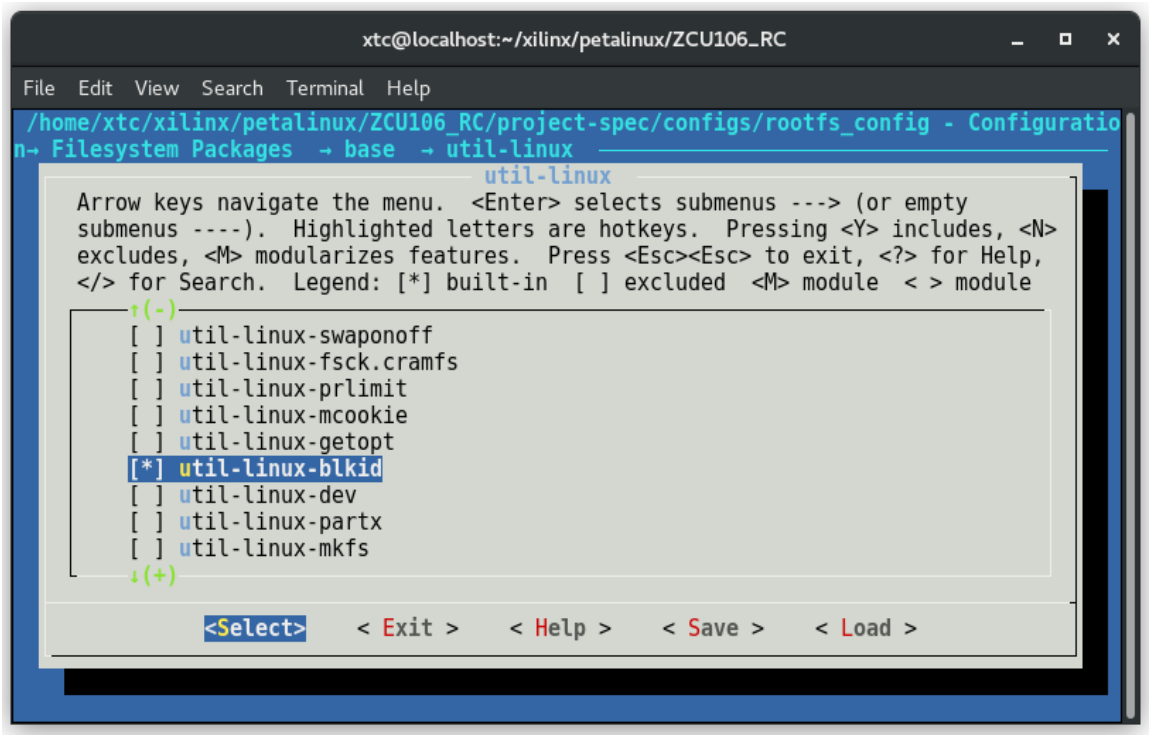


Figure 126 - Include util-linux-blkid

Press <Y> to include “util-linux mkfs”.

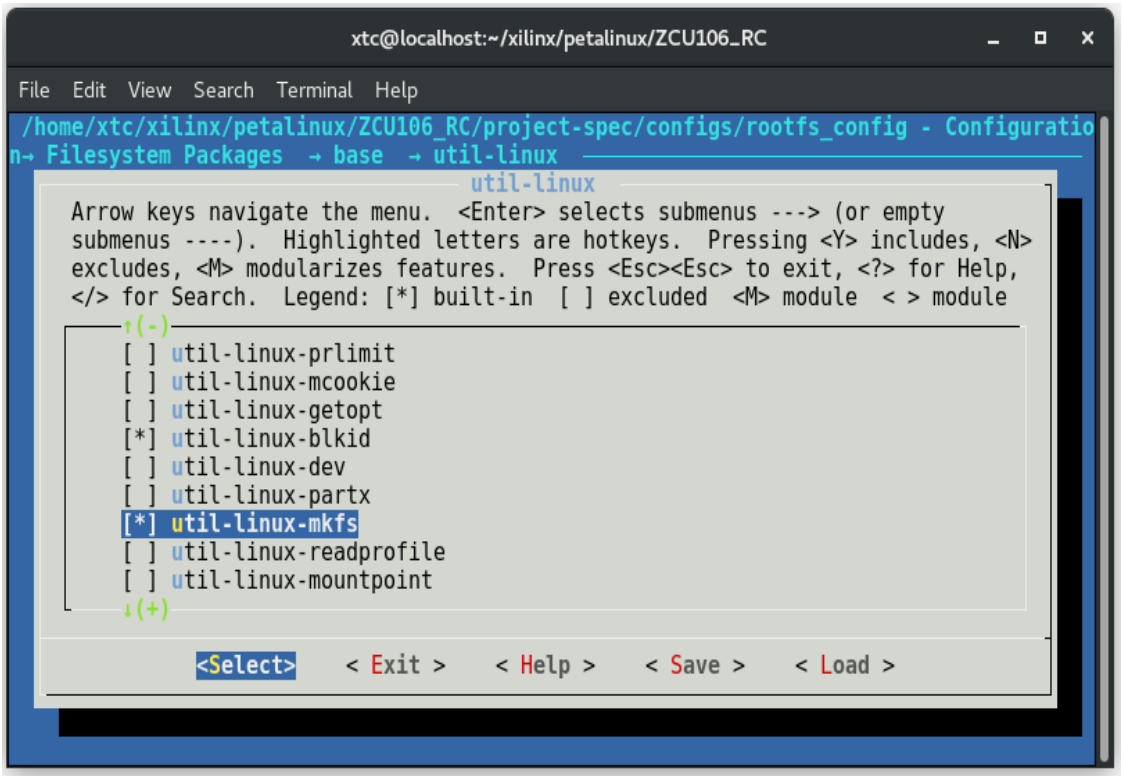


Figure 127 - Include util-linux-mkfs

Press <Y> to include “util-linux fdisk”.

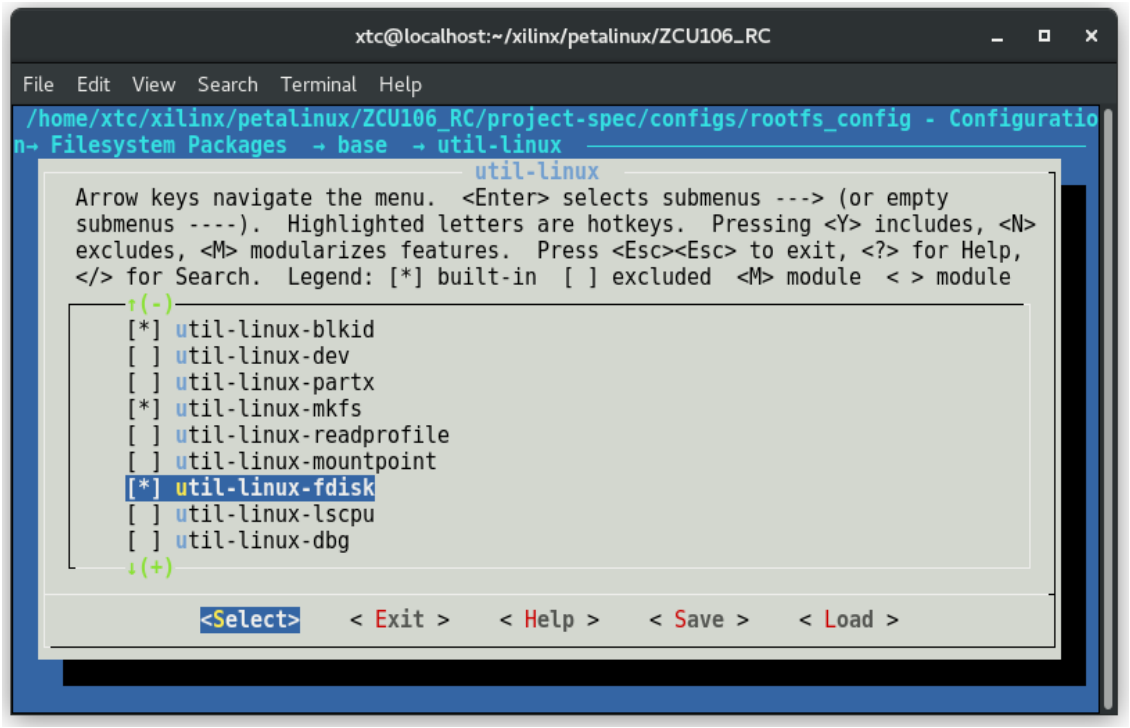


Figure 128 - Include util-linux-fdisk

Exit “util-linux” configuration window.

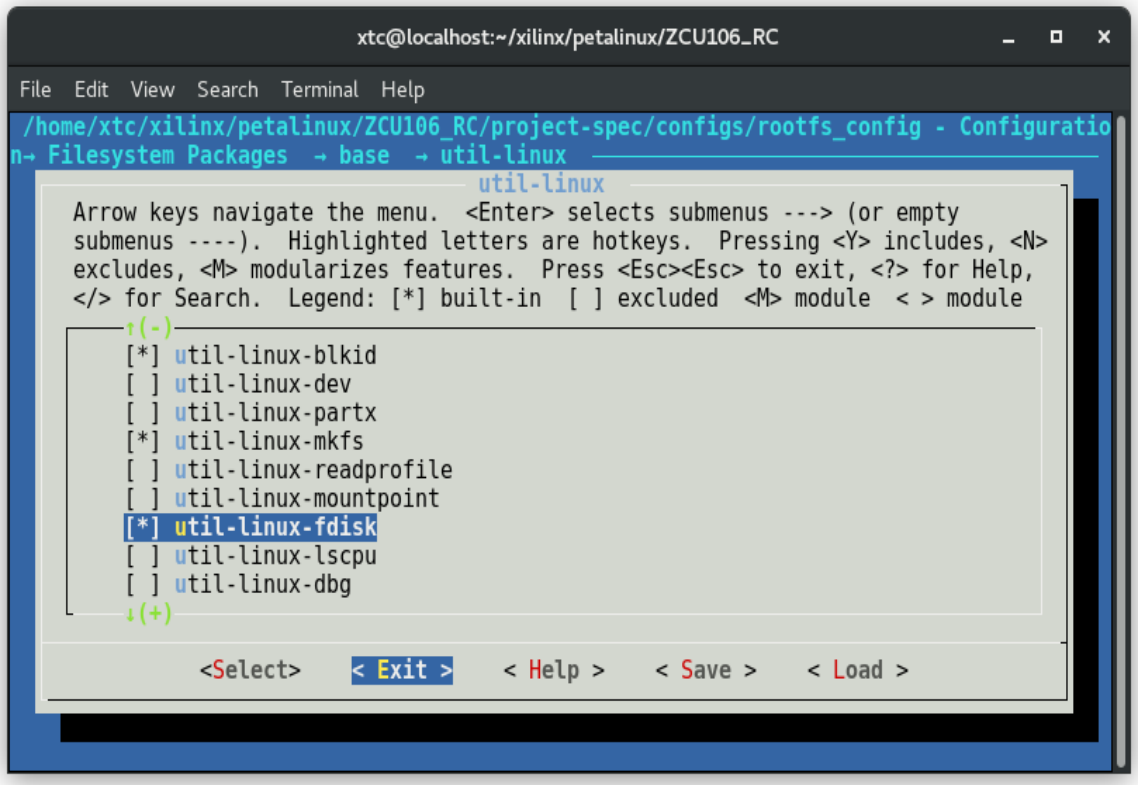


Figure 129 - Exit util-linux

Enter “e2fsprogs”.

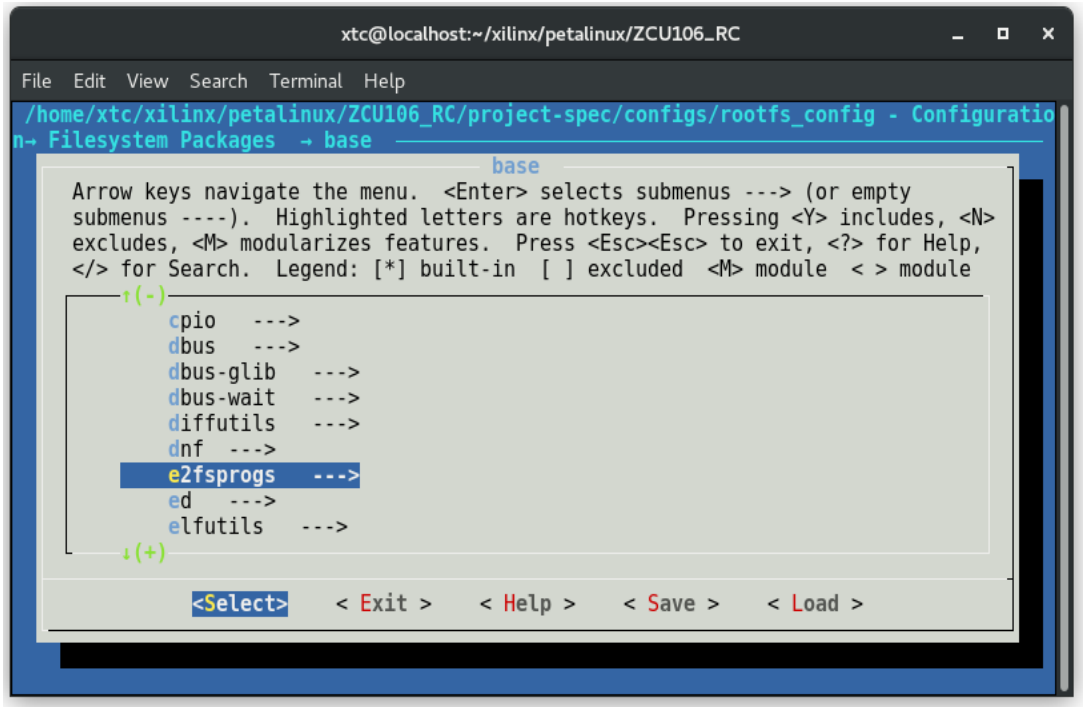


Figure 130 - Enter e2fsprogs

Press <Y> to include “e2fsprogs” and “e2fsprogs-mke2fs”.

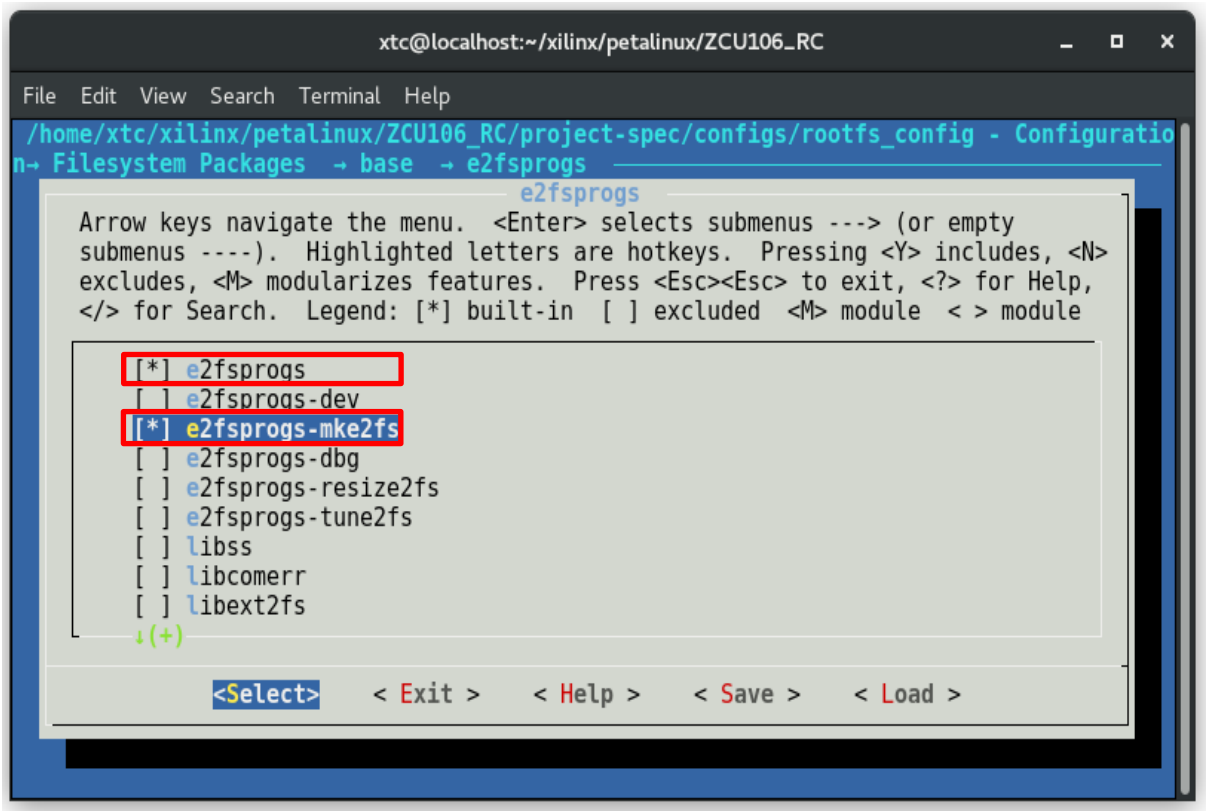


Figure 131 - Include e2fsprogs-mke2fs

Exit “e2fsprogs” configuration.

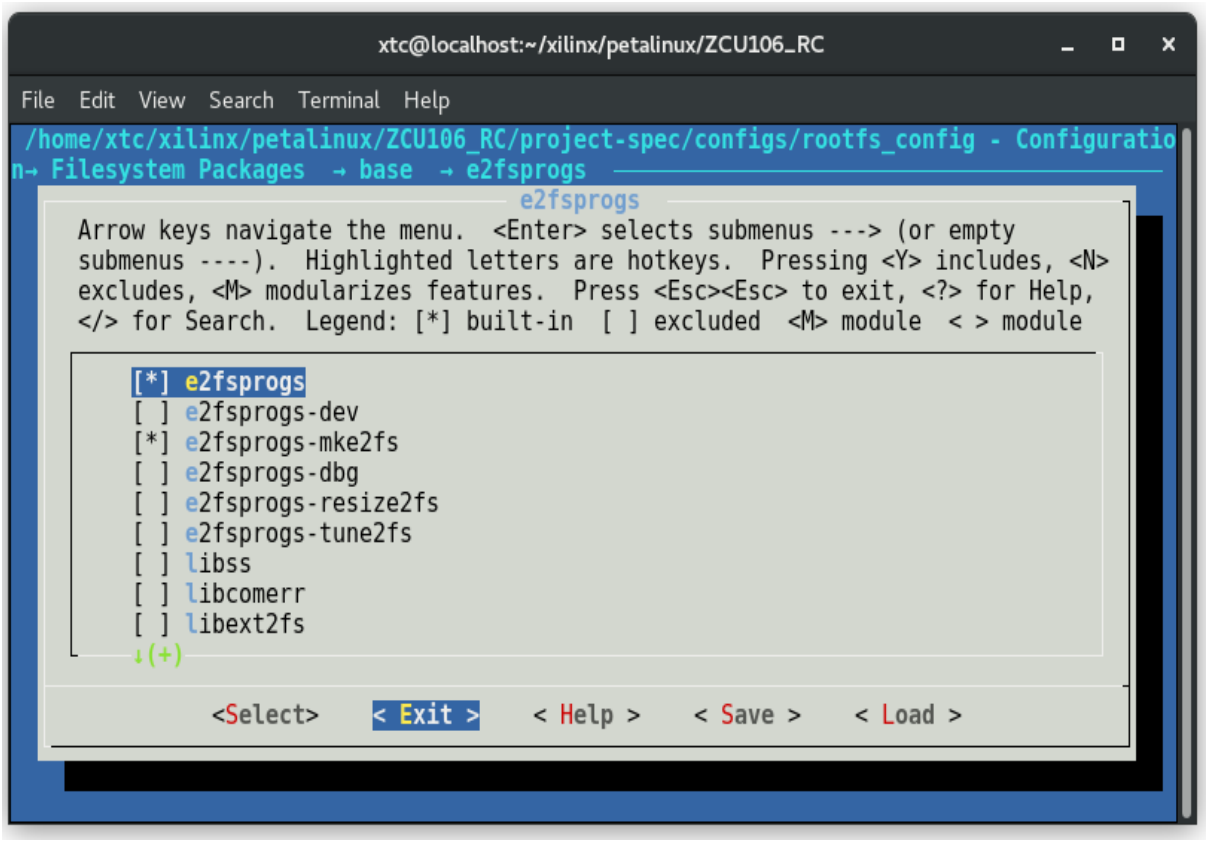


Figure 132 - Exit e2fsprogs

Exit root filesystem configuration.

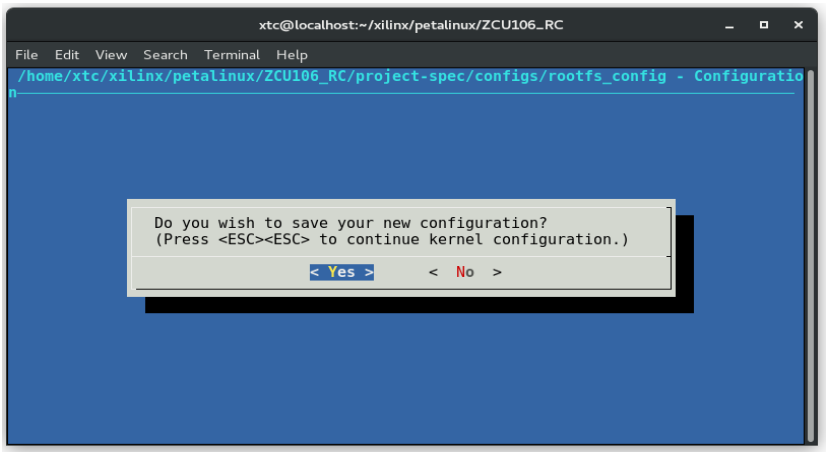


Figure 133 - Save root filesystem configuration

When the root filesystem is successfully configured a confirmation will be displayed as shown in Figure 133.

```
configuration written to /home/xtc/xilinx/petalinux/ZCU106_RC/project-spec/configs/root
fs_config

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

[INFO] generating petalinux-user-image.bb
[INFO] successfully configured rootfs
[xtc@localhost ZCU106_RC]$
```

Figure 134 - Confirmation message for configured root filesystem

Build the project using the “petalinux-build” command.

```
[xtc@localhost ZCU106_RC]$ petalinux-build
```

Figure 135 - Building the project

It will take time to build the project. If successful, a message stating that the project was successfully built will appear in the Terminal. Ignore the warning message.

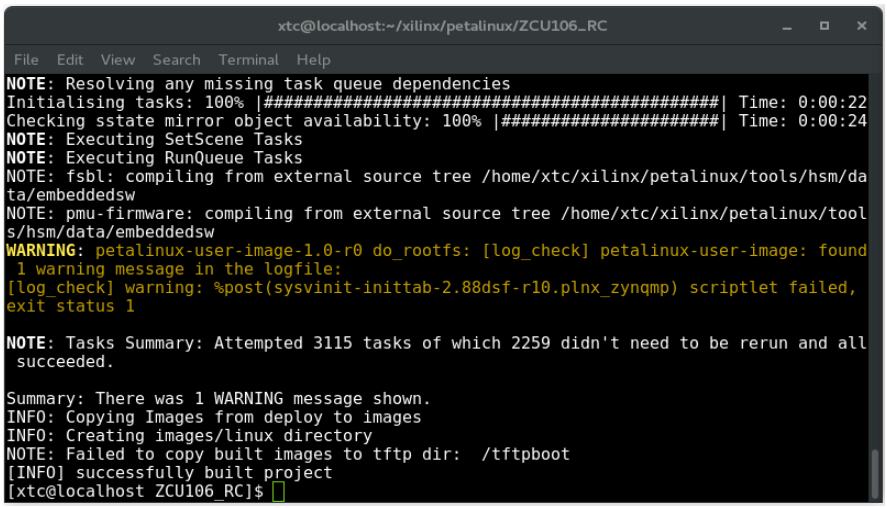
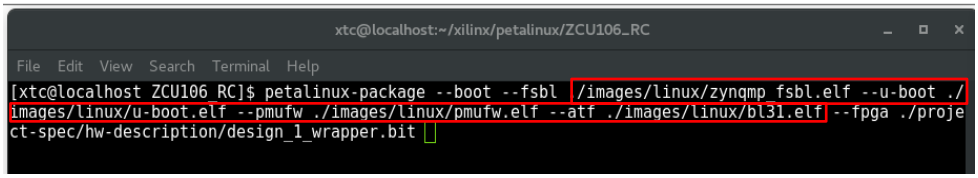


Figure 136 - Confirmation message for successful built project

Locate the file bl31.elf, pmufw.elf, u-boot.elf and zynqmp_fsbl.elf for packaging and generating the image which is usually in the file directory petalinux/<project-name>/images/linux.



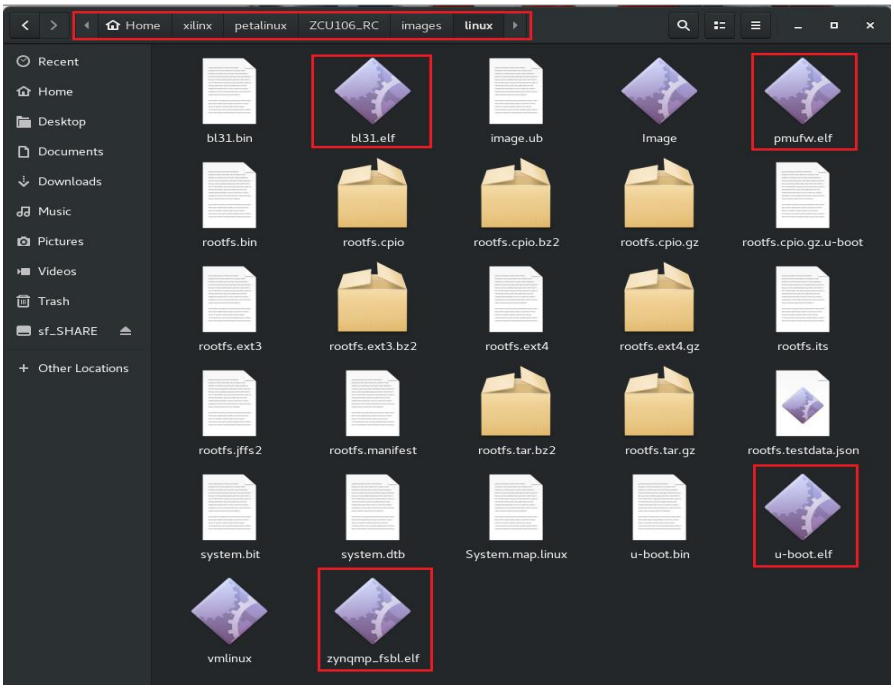


Figure 137 - Locating the .elf file for CPU codes

Locate the file “design_1_wrapper.bit” for packaging and generating the image which is usually in the file directory petalinux/<project-name>/project-spec/hw-description.

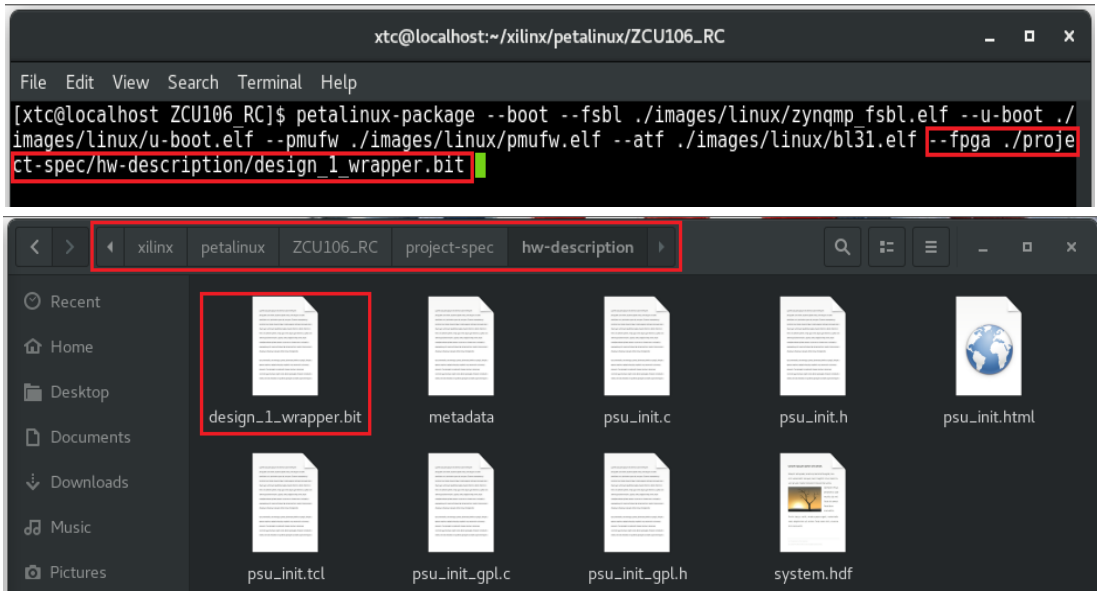


Figure 138 - Locating the binary configuration .bit file

Package these files (.elf and .bit) using the following syntax:

petalinux-package --boot --fsbl ./images/linux/zynqmp_fsbl.elf --u-boot ./images/linux/u-boot.elf --pmufw .images/linux/pmufw.elf --atf .images/linux/bl31.elf --fpga ./project-spec/hw-description/design_1_wrapper.bit

```
[xtc@localhost ZCU106_RC]$ petalinux-package --boot --fsbl ./images/linux/zynqmp_fsbl.elf --u-boot ./images/linux/u-boot.elf --pmufw ./images/linux/pmufw.elf --atf ./images/linux/bl31.elf --fpga ./project-spec/hw-description/design_1_wrapper.bit
```

Figure 139 - Packaging both CPU codes file (.elf) and binary configuration (.bit)

A “Binary is ready” message will be displayed. Ignore the TFTPBOOT warning.

```
INFO: File in BOOT BIN: "/home/xtc/xilinx/petalinux/ZCU106_RC/images/linux/zynqmp_fsbl.elf"
INFO: File in BOOT BIN: "/home/xtc/xilinx/petalinux/ZCU106_RC/images/linux/pmufw.elf"
INFO: File in BOOT BIN: "/home/xtc/xilinx/petalinux/ZCU106_RC/project-spec/hw-description/design_1_wr
apper.bit"
INFO: File in BOOT BIN: "/home/xtc/xilinx/petalinux/ZCU106_RC/images/linux/bl31.elf"
INFO: File in BOOT BIN: "/home/xtc/xilinx/petalinux/ZCU106_RC/images/linux/u-boot.elf"
INFO: Generating zynq binary package BOOT.BIN...


****** Xilinx Bootgen v2018.2
  **** Build date : Jun 14 2018-20:09:18
    ** Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.

INFO: Binary is ready.
WARNING: Unable to access the TFTPBOOT folder /tftpboot!!!
WARNING: Skip file copy to TFTPBOOT folder!!!
[xtc@localhost ZCU106_RC]$
```

Figure 140 - Confirmation message for packaged binary file

Create a simple-test application using the syntax in the Figure 141.

```
[xtc@localhost ZCU106_RC]$ petalinux-create -t apps --template c --name simple-test --enable --force
```

Figure 141 - Creating simple-test application

A message stating that the test application was successfully built will appear in the Terminal.

```
#INFO: Create apps: simple-test
INFO: New apps successfully created in /home/xtc/xilinx/petalinux/ZCU106_RC/project-spec/meta-user/re
cipes-apps/simple-test
INFO: Enabling created component...
INFO: sourcing bitbake
INFO: oldconfig rootfs
INFO: simple-test has been enabled
[xtc@localhost ZCU106_RC]$
```

Figure 142 - Confirmation message for created simple-test application

Create another test application called pio-test using the syntax below.

```
[xtc@localhost ZCU106_RC]$ petalinux-create -t apps --template c --name pio-test --enable --force
```

Figure 143 - Creating pio-test application

It will again show a confirmation message that you have successfully created a pio-test application.

```
INFO: Create apps: pio-test
INFO: New apps successfully created in /home/xtc/xilinx/petalinux/ZCU106_RC/project-spec/meta-user/re
cipes-apps/pio-test
INFO: Enabling created component...
INFO: sourcing bitbake
INFO: oldconfig rootfs
INFO: pio-test has been enabled
[xtc@localhost ZCU106_RC]$
```

Figure 144 - Confirmation message for created pio-test application

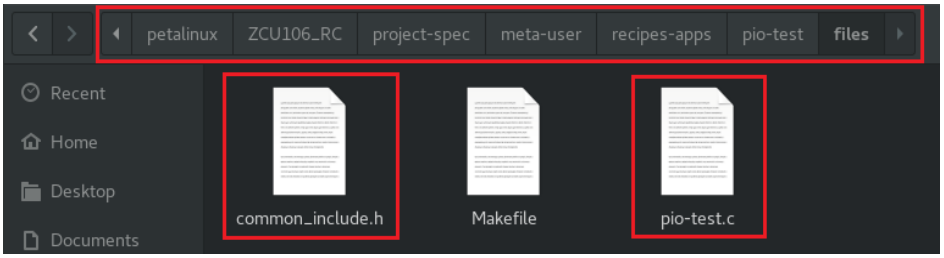
Open any text editor. Type in the following codes for the header file and save the file as “common_include.h”.

(Note: common_include.h can be downloaded from https://www.xilinx.com/support/answers/72076.html)

```
#ifndef _XLNX_PCIE_DMA_COMMON_H_
#define _XLNX_PCIE_DMA_COMMON_H_


#define MAX_NUMBER_OF_CHANNELS   4

#define MAX_ALLOWED_CHANNELS_IN_HW   4

#if MAX_NUMBER_OF_CHANNELS > MAX_ALLOWED_CHANNELS_IN_HW
#error "Please reduce number of DMA engines defined in MAX_NUMBER_OF_CHANNELS macro"
#endif

/* Each char device is associated to 1 channel.
 * write call of char device services Host to Card transfers for the channel
 * read call of char device services Card to Host transfer for the channel
 * */
#define MAX_NUMBER_OF_CHAR_DEVICES     MAX_NUMBER_OF_CHANNELS

/* This macro defines the number of Pcie devices that can be
 * supported by this driver when they are inserted parallelly
 * in different slots of the host machine
 */
#define MAX_EXP_DMA_DEVICES            5

#define CHAR_DRIVER_NAME               "ps_pcie_dmachan"

#define PIO_CHAR_DRIVER_NAME           "ps_pcie_pio"

#define EP_TRANSLATION_CHECK            0xCCCCCCCC

#define XPIO_CLIENT_MAGIC 'P'
#define IOCTL_EP_CHECK_TRANSLATION     _IO(XPIO_CLIENT_MAGIC, 0x01)


#define NUMBER_OF_BUFFER_DESCRIPTORS   1999

#define CHANNEL_COAELSE_COUNT          0

#define CHANNEL_POLL_TIMER_FREQUENCY   (HZ) //Lower value indicates frequent checks and greater
processing.

#endif
```

Figure 145 - Header file – common_include.h

Copy the common_include.h file to both the “simple-test” application and “pio-test” application file locations, as shown in Figure 146.



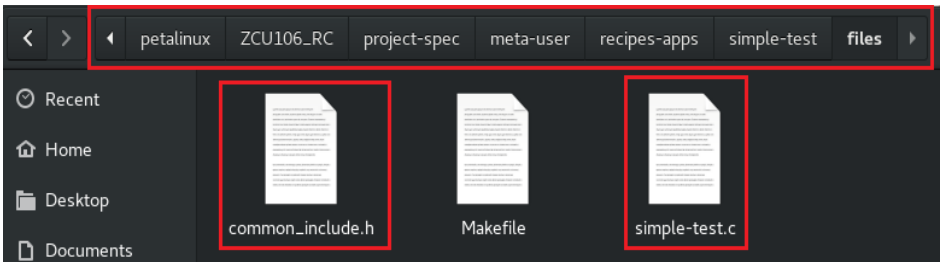


Figure 146 - Header and C file locations

Note: pio-test.c and simple-test.c can be downloaded from
https://www.xilinx.com/support/answers/72076.html

View and edit the “Makefile” file under the “simple-test” application. Add the line of code inside the red rectangular box as shown in Figure 147.

```
[xtc@localhost ZCU106_RC]$ vi project-spec/meta-user/recipes-apps/simple-test/files/Makefile
```

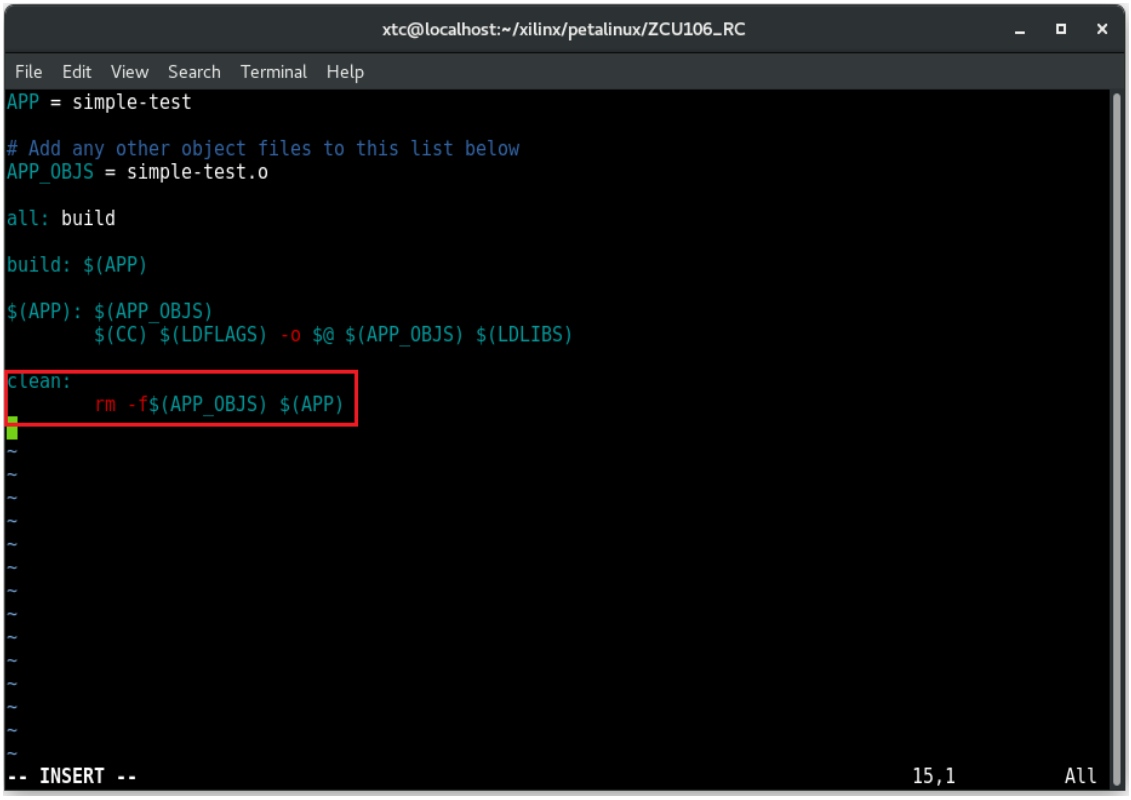


Figure 147 - View and edit Makefile for simple-test app

Press the escape key, then the “:” colon key and type “wq” to save and exit. The letters w and q mean written and quit.

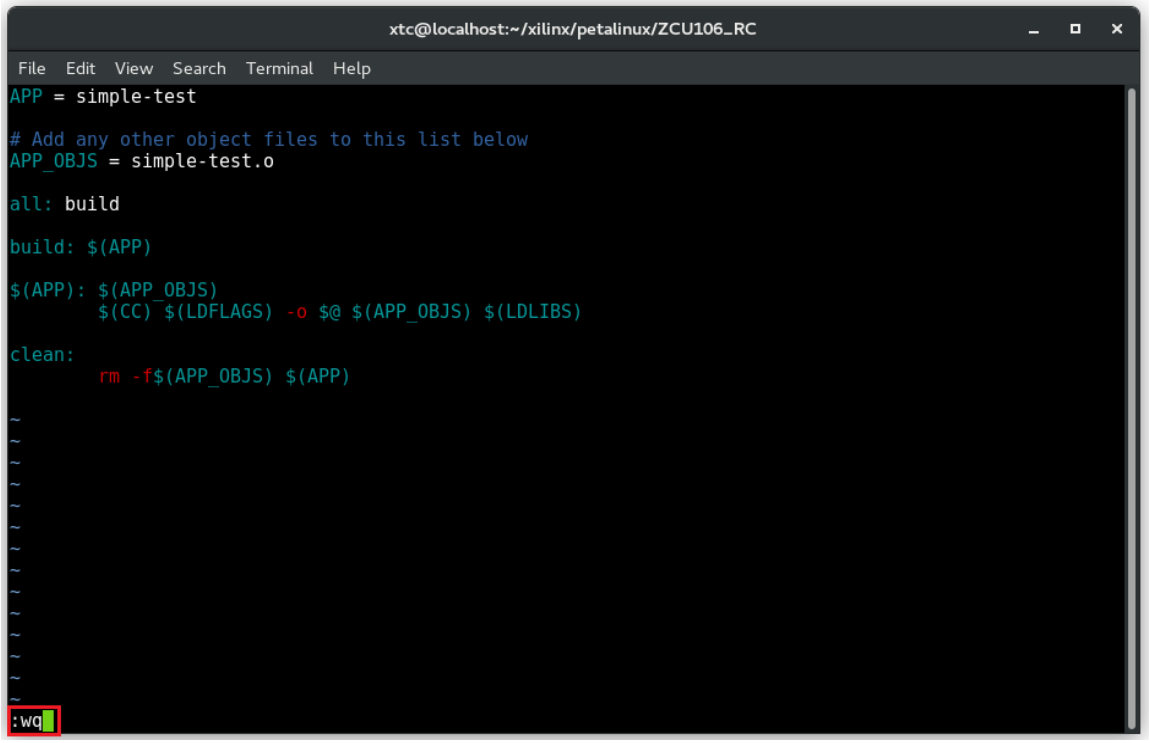


Figure 148 - Save and exit text editor for the simple-test app

View and edit the “Makefile” file under “pio-test” application. Add the line of code inside the red rectangular box as shown in Figure 149.

```
[xtc@localhost ZCU106_RC]$ vi project-spec/meta-user/recipes-apps/pio-test/files/Makefile
```

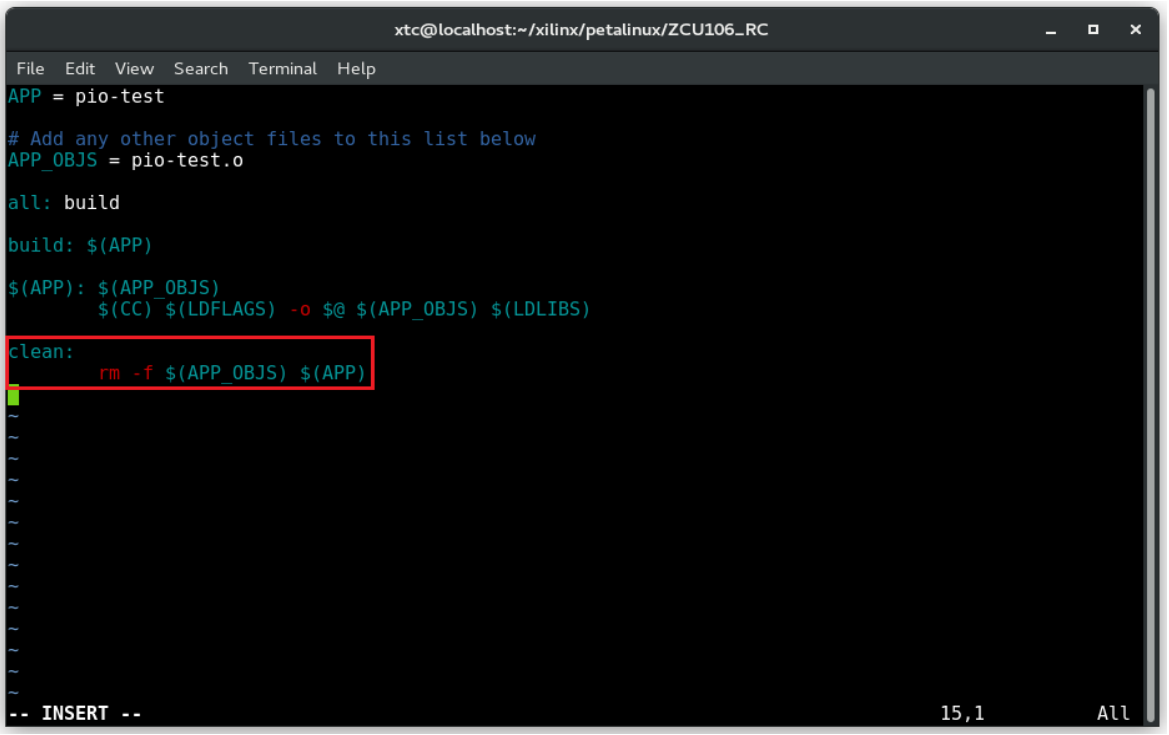


Figure 149 - View and edit Makefile for pio-test app

Press the escape key, then the ":" colon key and type "wq" to save and exit.

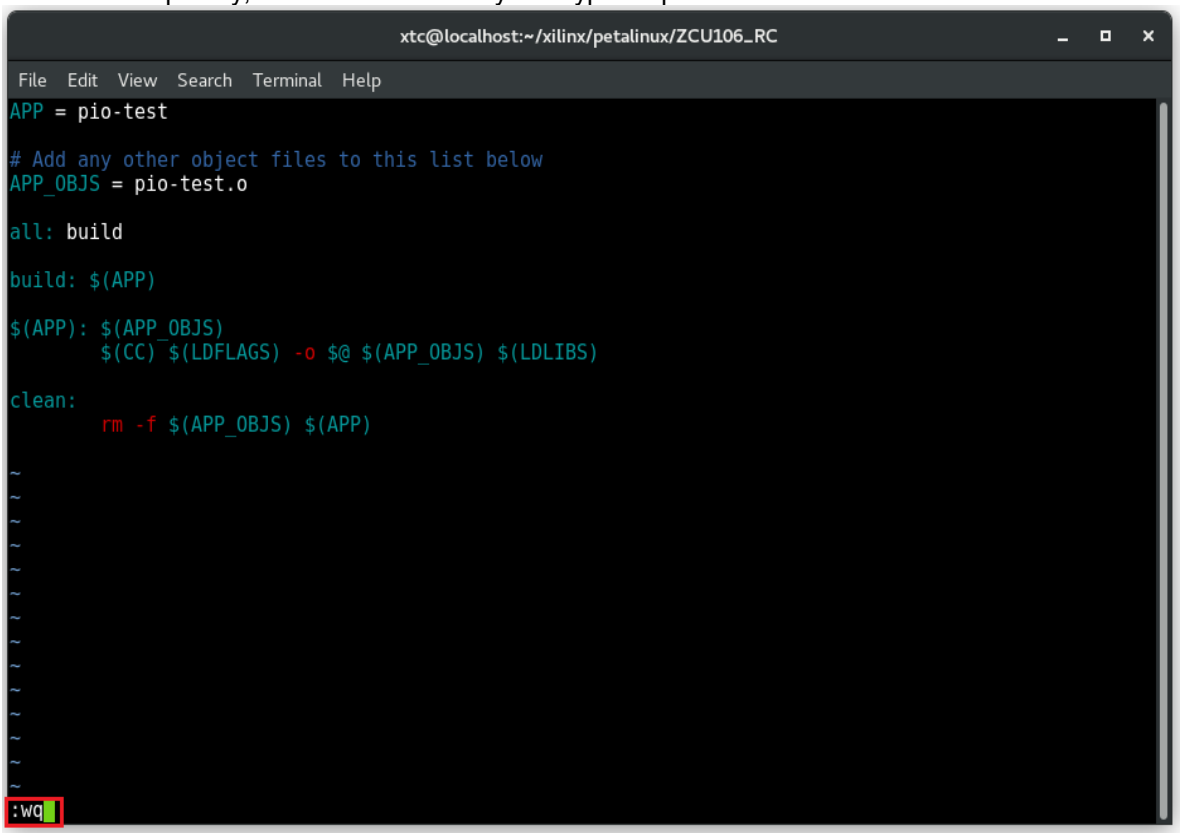


Figure 150 - Save and exit text editor for the pio-test app

View and edit the "simple-test.bb" file under the "simple-test" application. Add the line of code inside the red rectangular box as shown in Figure 151.

```
[xtc@localhost ZCU106_RC]$ vi project-spec/meta-user/recipes-apps/simple-test/simple-test.bb
```

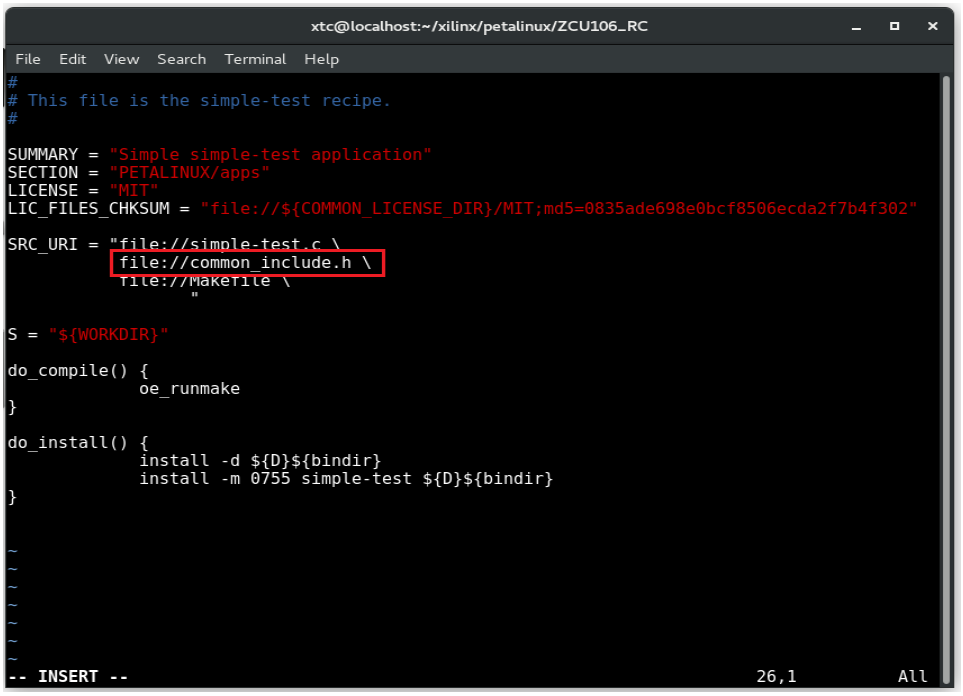


Figure 151 - View and edit simple-test.bb file

Press the escape key, then the ":" colon key and type "wq" to save and exit.

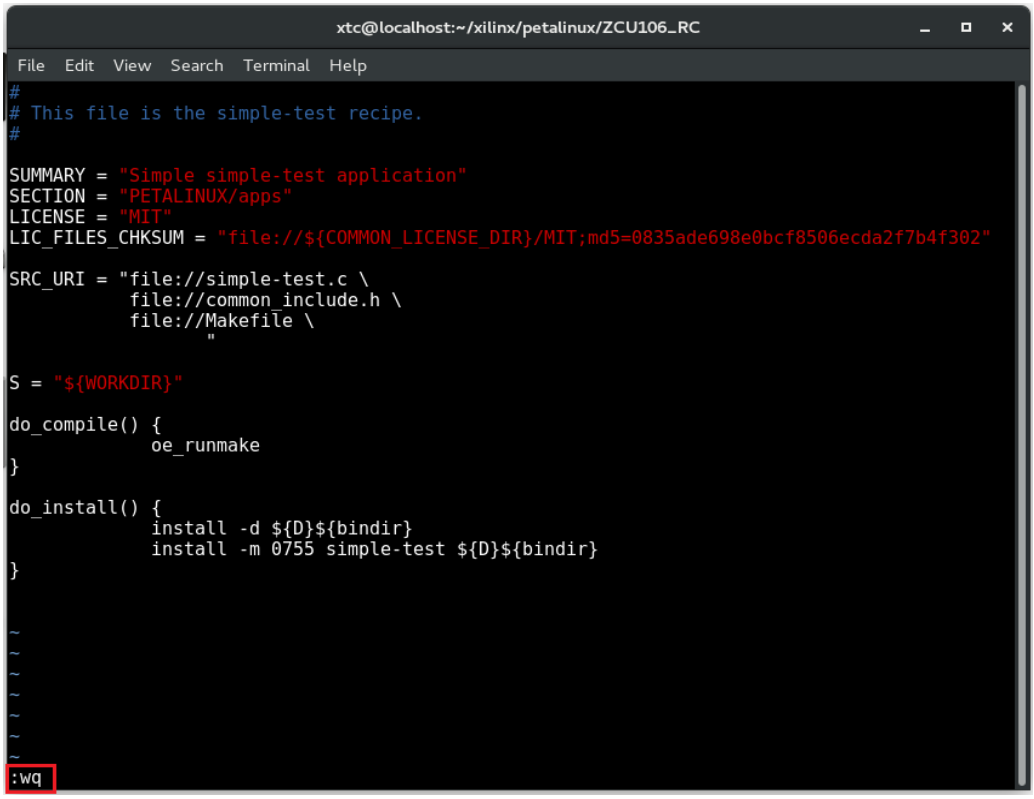


Figure 152 - Save and exit text editor for the simple-test.bb file

View and edit the "pio-test.bb" file under the "pio-test" application. Add the line of code inside the red rectangular box as shown in Figure 153.

```
[xtc@localhost ZCU106_RC]$ vi project-spec/meta-user/recipes-apps/pio-test/pio-test.bb
```

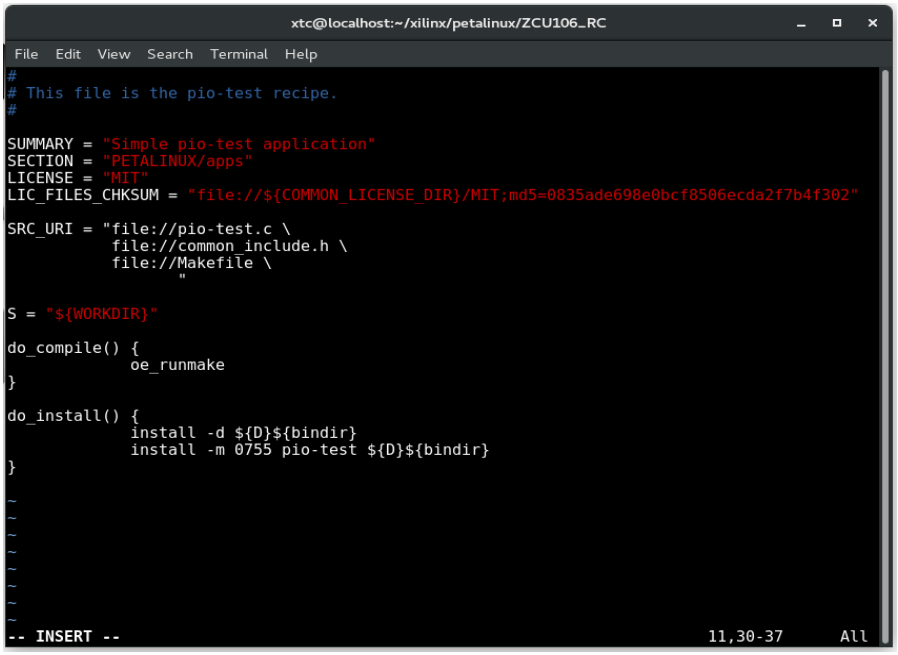


Figure 153 - View and edit pio-test.bb file

Press the escape key, then the ":" colon key and type "wq" to save and exit.

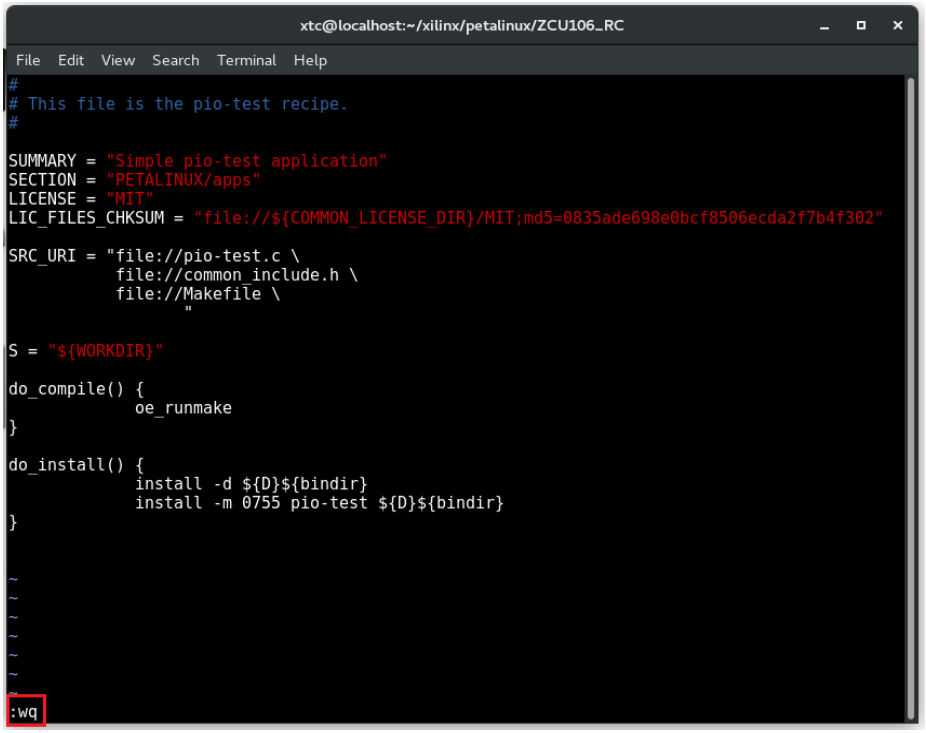


Figure 154 - Save and exit text editor for the pio-test.bb file

Rebuild the project.

```
[xtc@localhost ZCU106_RC]$ petalinux-build
```

Figure 155 - Rebuild the project

If successful, a message stating that the project was successfully built will appear in the Terminal. Ignore the warning message.

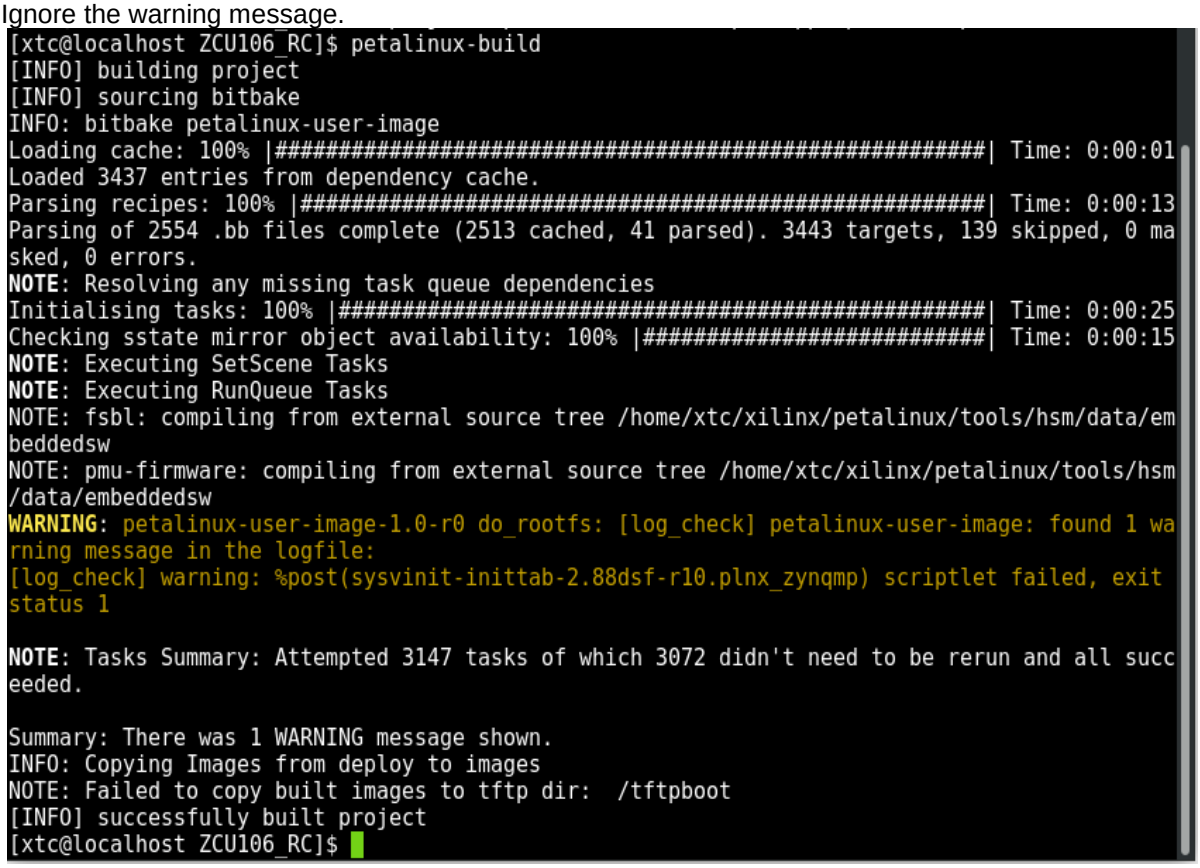


Figure 156 - Confirmation message for successfully built project

Enter the syntax shown in Figure 157 to repackage and overwrite the image file. The “--force” syntax is required to overwrite the existing image file.

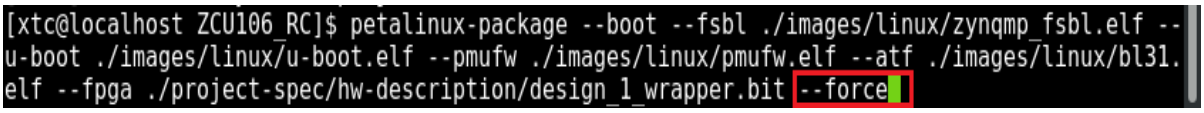


Figure 157 - Repackage the project

Another “Binary is ready” message will be displayed. Ignore the TFTPBOOT warning again.

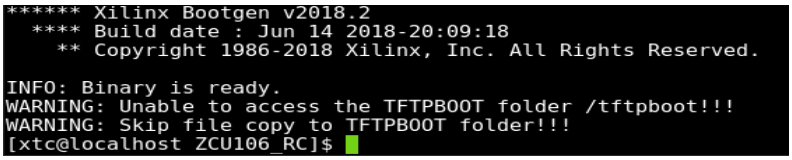


Figure 158 - Confirmation message for packaged binary file

Locate the “BOOT.BIN” and “image.ub” image file and copy them to an SD card for booting up Linux on the ZCU106 board.

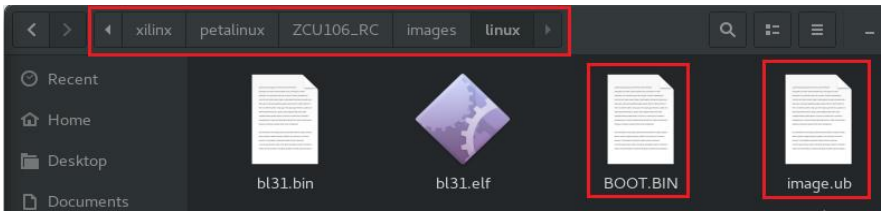


Figure 159 - Locate both image file and boot file

UltraZed Endpoint Design in Vivado

Create a new Vivado project

Launch Vivado 2018.2 and, from the welcome screen, click “Create Project” as shown in Figure 1600.

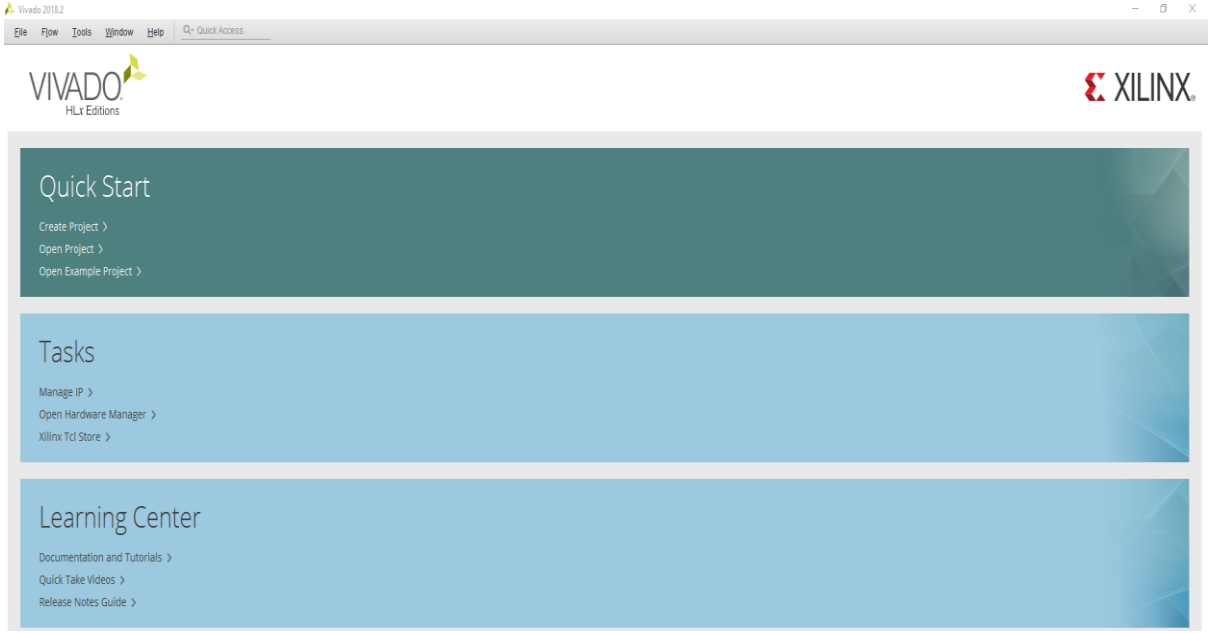


Figure 160 - Create project

Click “Next”.

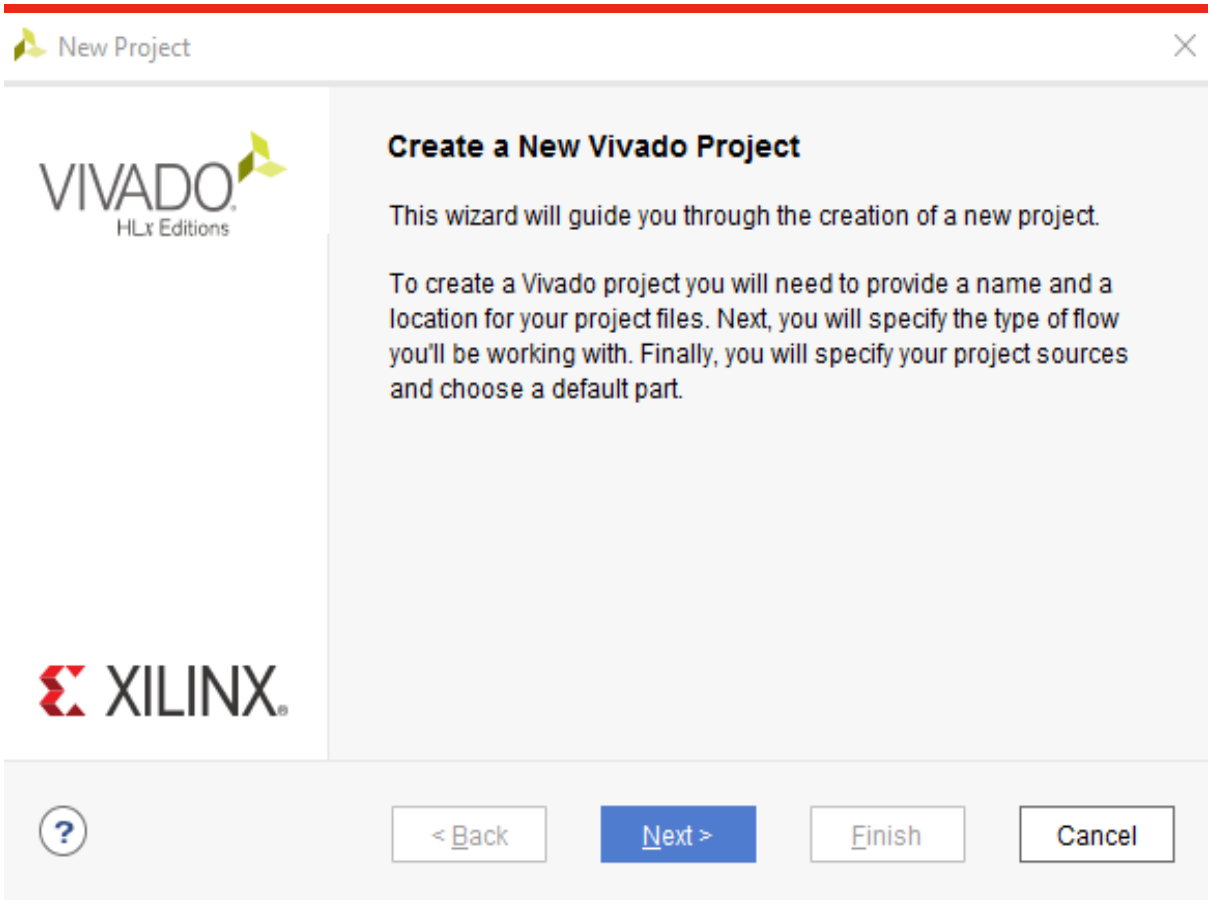


Figure 161 – Create a new Vivado project

Enter a desired name for the project and specify a project location where the project files will be stored. Select the “Create project subdirectory” checkbox as shown Figure 162. Click “Next”.

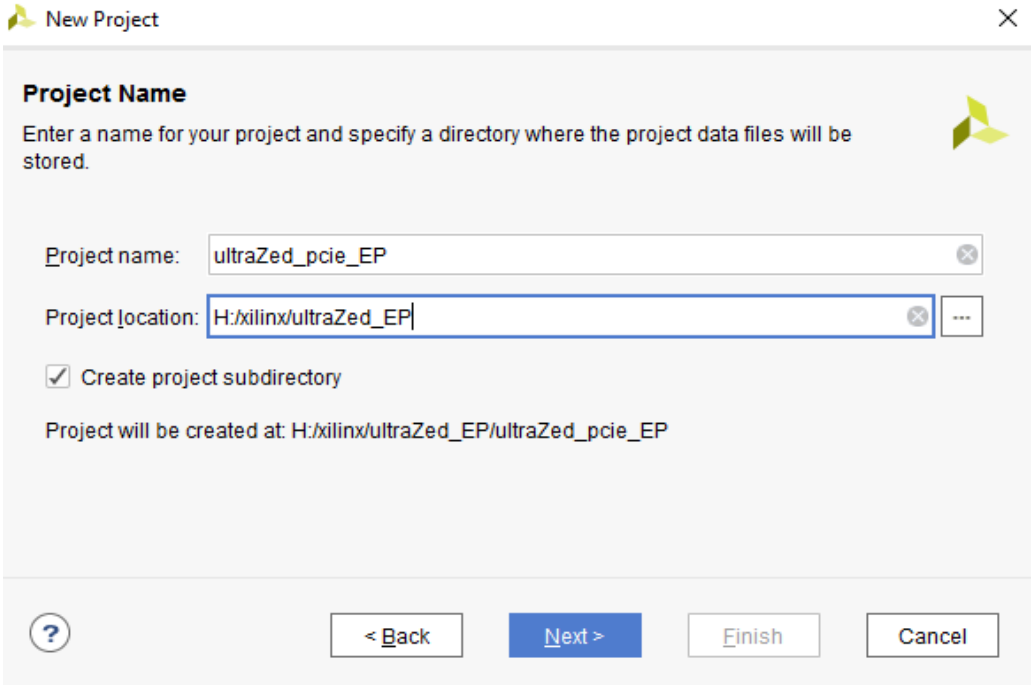


Figure 162 - Choose a project name

In the Project Type window, select the “RTL Project” radio button and tick “Do not specify sources at this time”. RTL is short for Register Transfer Level. Click “Next”.

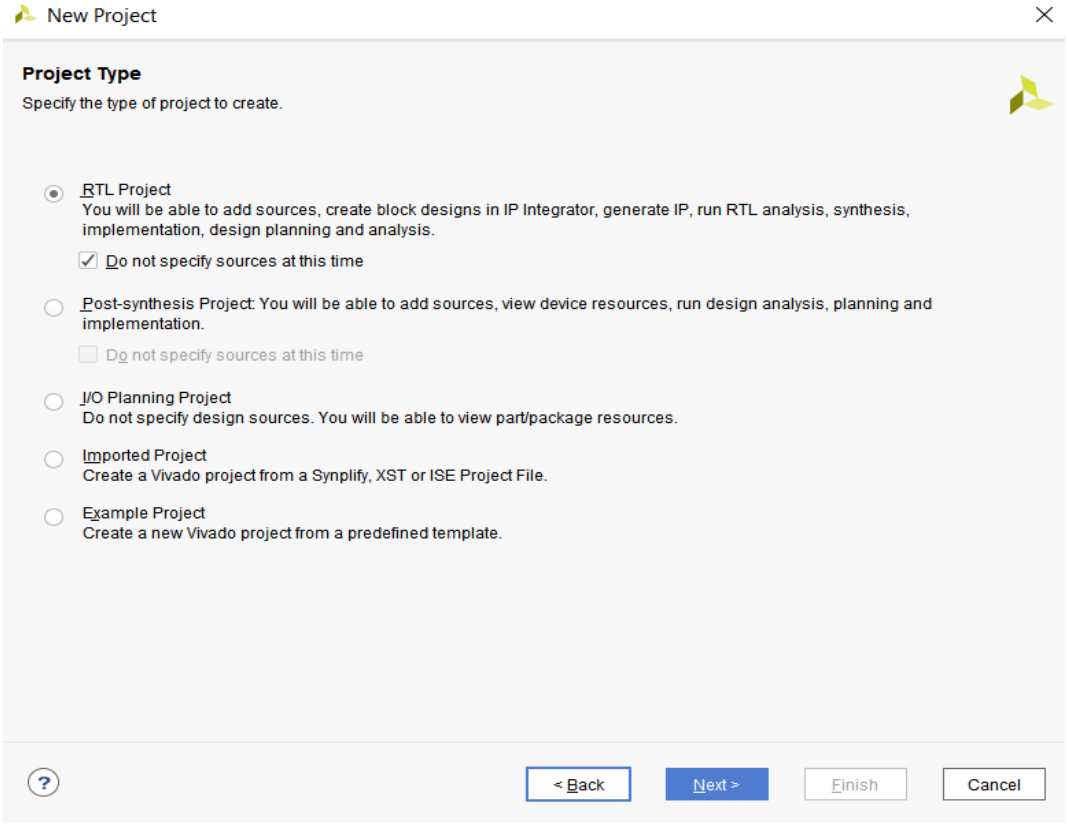


Figure 163 - Select a project type

Select “em.avnet.com” from the Vendor drop-down list and select Avnet UltraZed-3EG PCIe Carrier Card from the list as shown in Figure 1644 below. Click "Next".

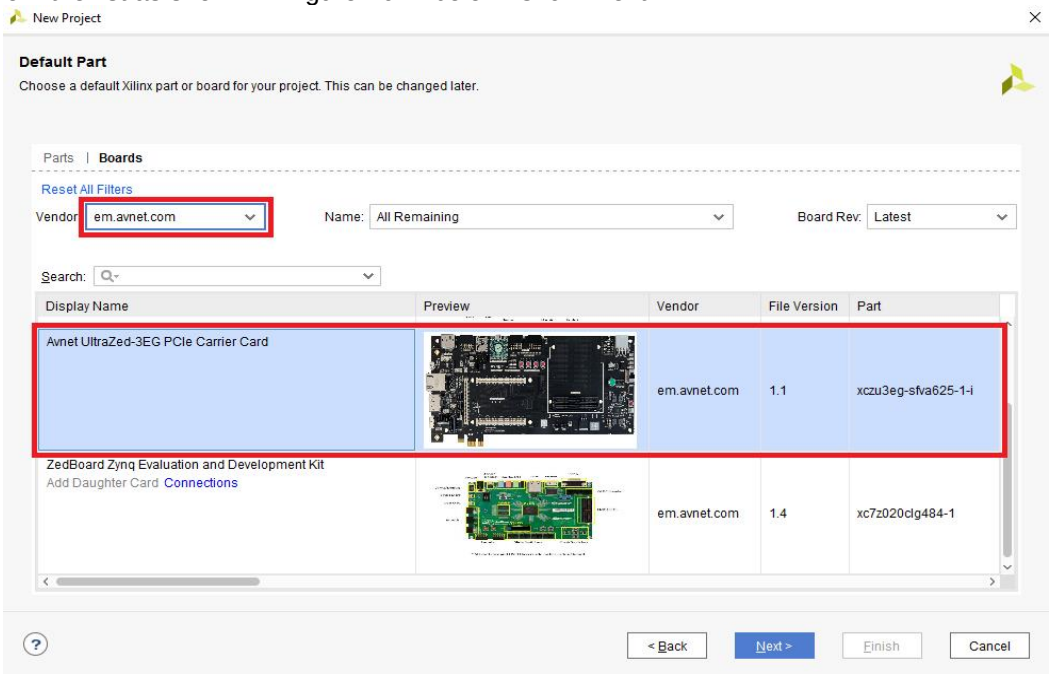


Figure 164 - Select board for the project

If the board is not listed, you can download the board files from the following GitHub repository: https://github.com/Avnet/bdf
Click on the “Clone or download” button. Download the zip file and copy ultrazed_3eg_pciecc from the list.

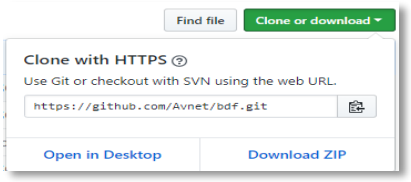


Figure 165 - Download board file

Paste this file into the following location **C:\Xilinx\Vivado\2018.2\data\boards\board_files** as shown in Figure 1666.

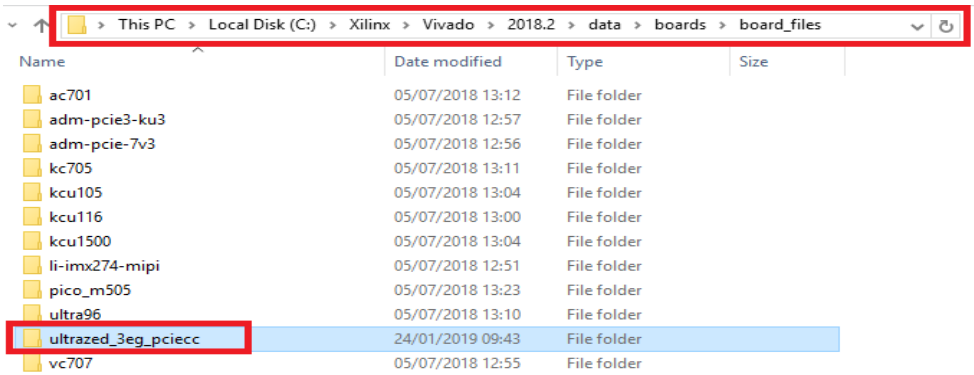


Figure 166 - Board file location

You might have to restart Vivado to see the newly added UltraZed board. If you are having problems installing the board files, you can find instructions on how to install the board files into the correct folder in the README.md file which can also be found in the GitHub repository.
The resulting dialog box will display a summary of the project which includes the board information. Click “Finish” to create the project.

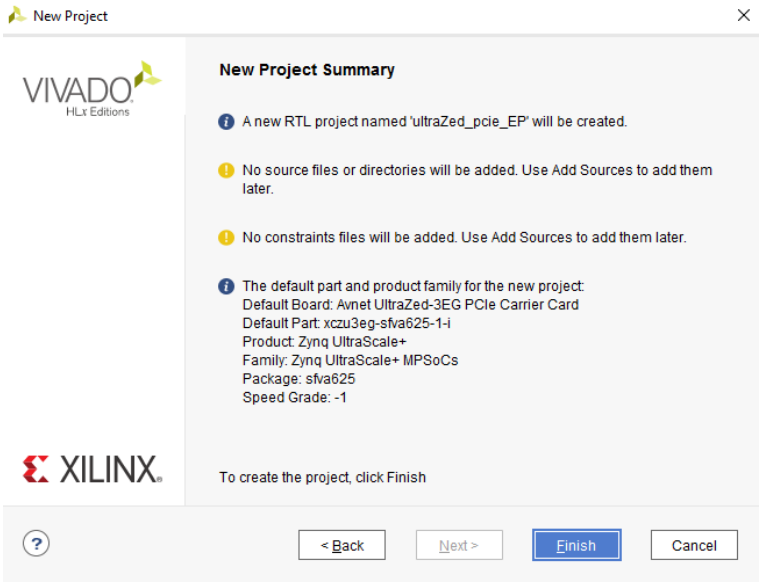


Figure 167 - Project summary

Click “Create Block Design” from the Vivado Flow Navigator pane.

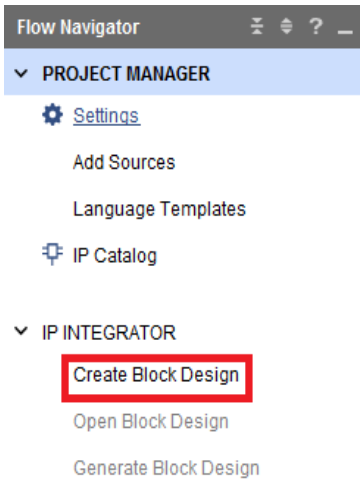


Figure 168 - Flow navigator

Specify a name for the block design and leave the "Directory" field <Local to Project>. Click OK.

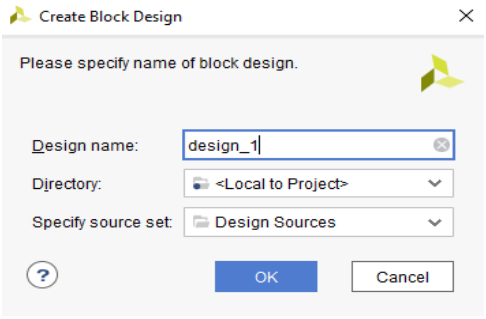


Figure 169 - Choose block design name

Click on the “+” icon on the toolbar or in the “Diagram” window as shown in Figure 1700 below.

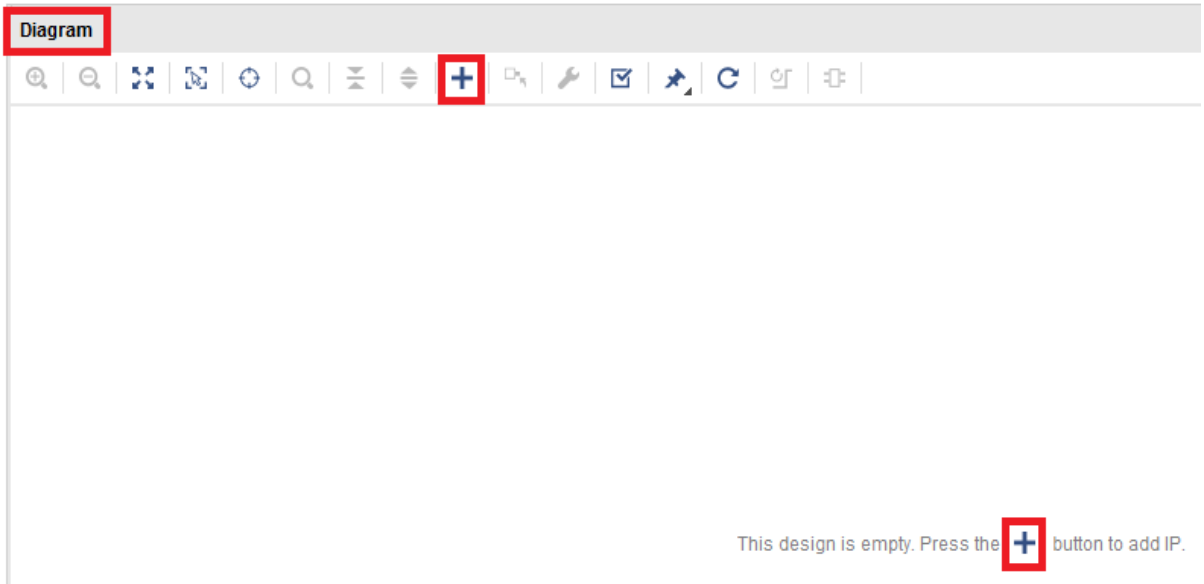


Figure 170 - Adding IP

Search for Zynq UltraScale+ MPSoC as shown in Figure 171. Double-click to add the IP block.

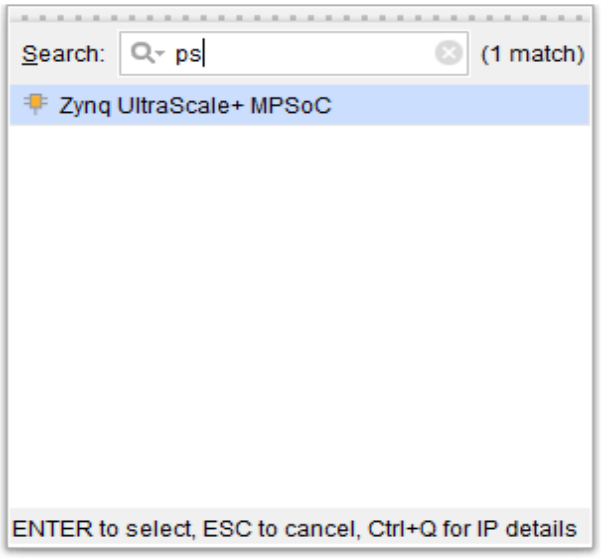


Figure 171 - Searching for the Zynq processor

The following Zynq processor block will be added to the design space.

Figure 172 - Zynq processor

Select the “Board” tab from the “Sources” window and then double-click on “LED” to open the "Connect Board Component" dialog box. Here you can select an IP block interface.

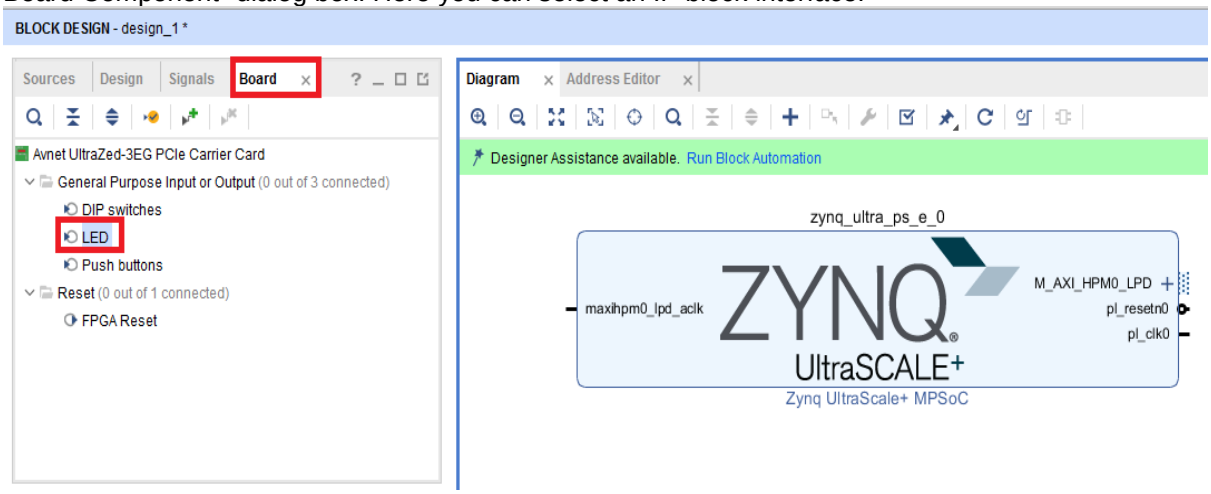


Figure 173 - Selecting GPIO-LED

Select GPIO from the list and click OK.

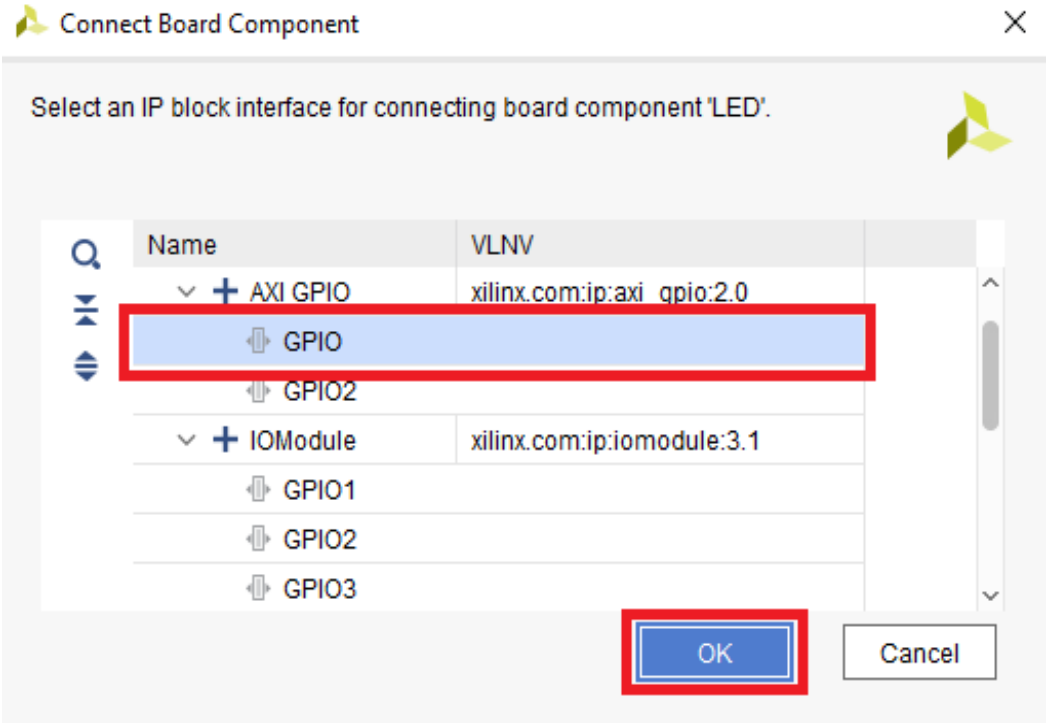


Figure 174 - Selecting GPIO interface for LED

This will add the LED “axi_gpio_0” IP block and the GPIO interface as shown in Figure 175.

Figure 175 - Established connection for LED

Similarly, double-click on "DIP switches" under “Board” to open the "Connect Board Component" dialog box.

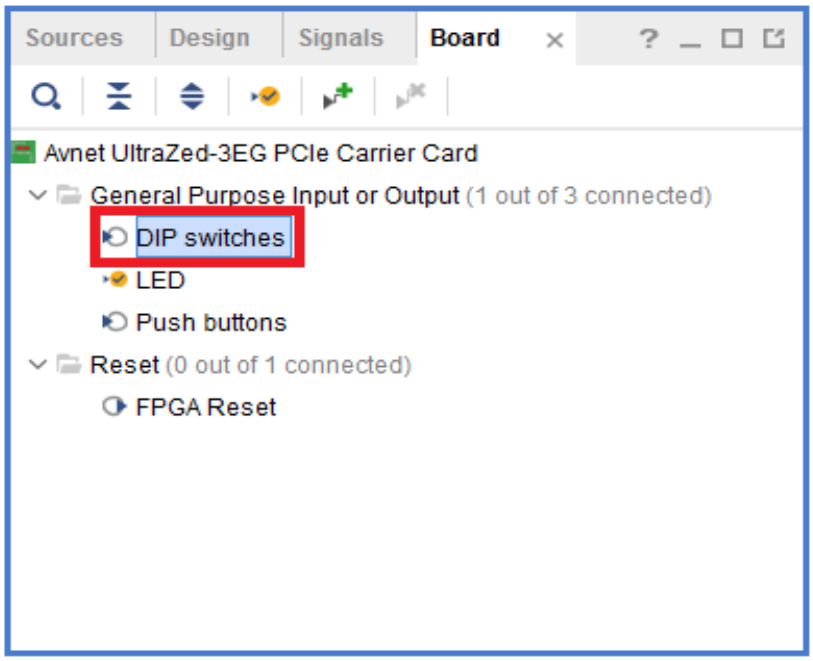


Figure 176 - Selecting GPIO-DIP Switches

Select GPIO2 and click OK to add the interface for dip_switches_8bits

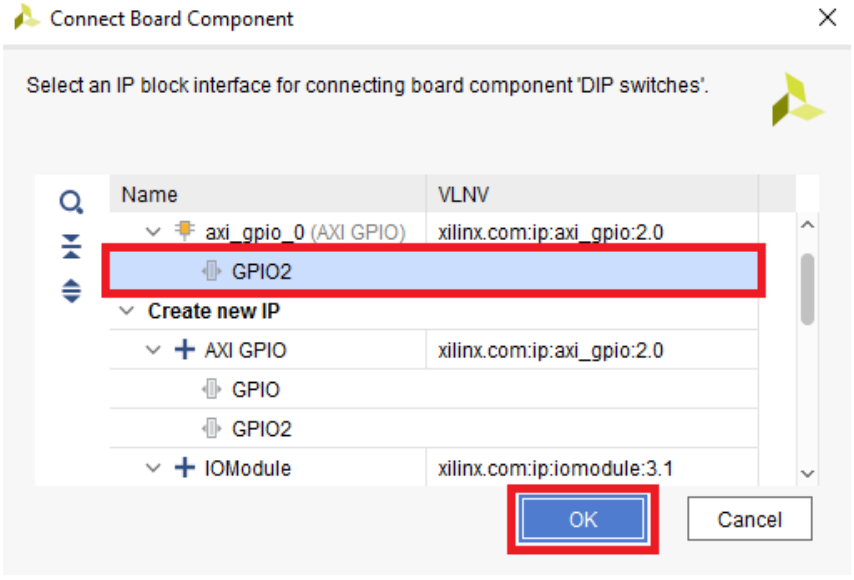


Figure 177 - Selecting the GPIO2 interface for DIP switches

This will add the “axi_gpio_0” IP block switches and the GPIO interface as shown in Figure 1788.

Figure 178 - Established connection for LED and DIP Switches

Another component should be added. Double-click on "FPGA Reset" under “Board” to open the "Connect Board Component" dialog box.

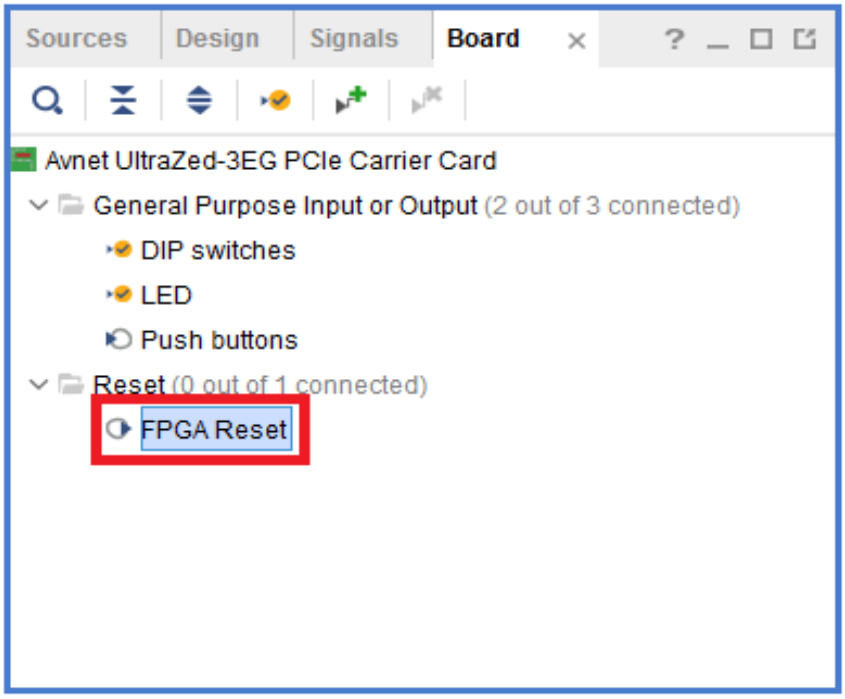


Figure 179 - Include processor system reset

Ensure that the “ext_reset” checkbox under “Processor System Reset” is selected and click OK to add the “proc_sys_reset_0” IP block and the "reset" interface.

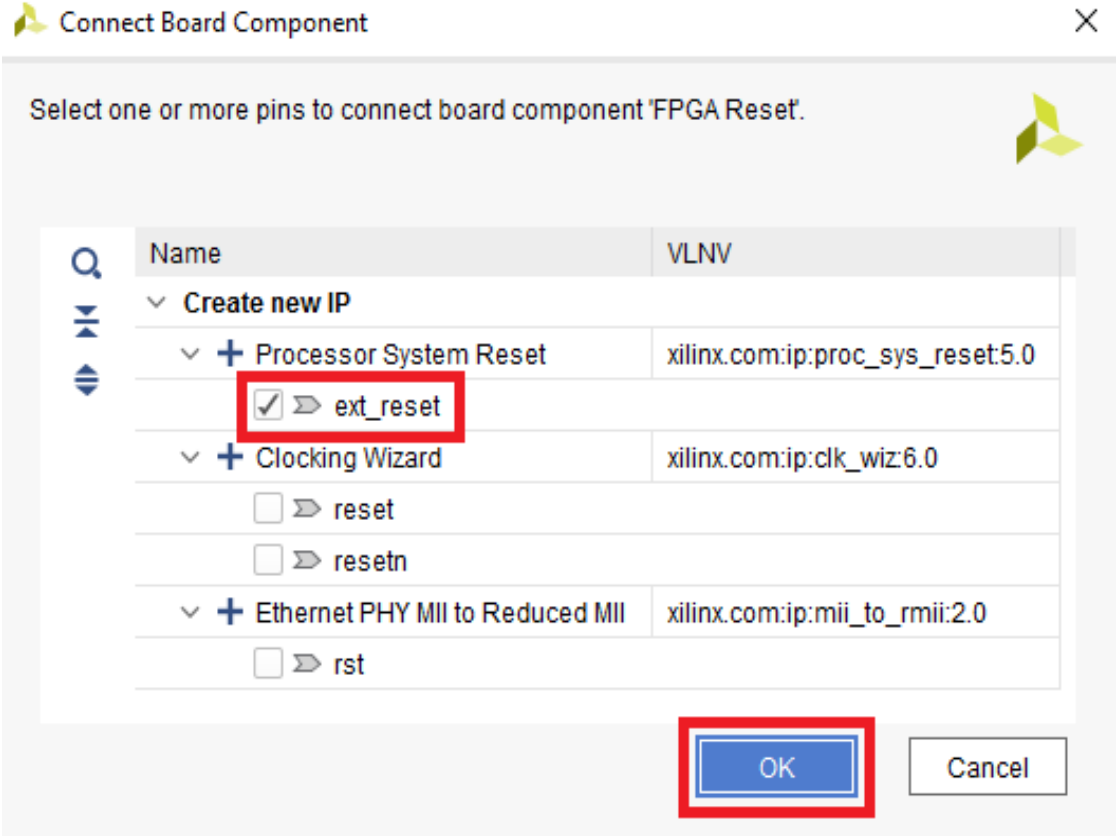


Figure 180 - Include external reset signal

Click on Run Block Automation as shown in Figure 181.

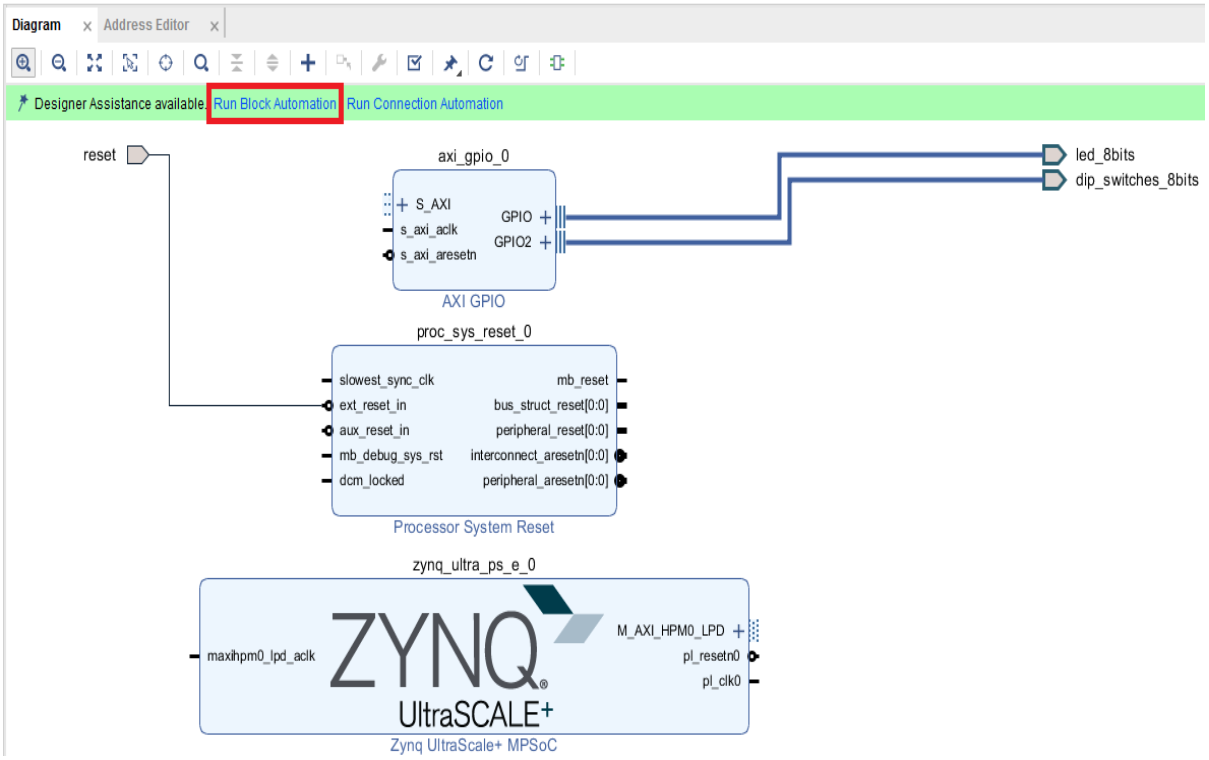


Figure 181 - Run block automation

In the resulting dialog box, ensure that the “zynq_ultra_ps_e_0” and the "Apply Board Preset" checkboxes are selected. Click OK.

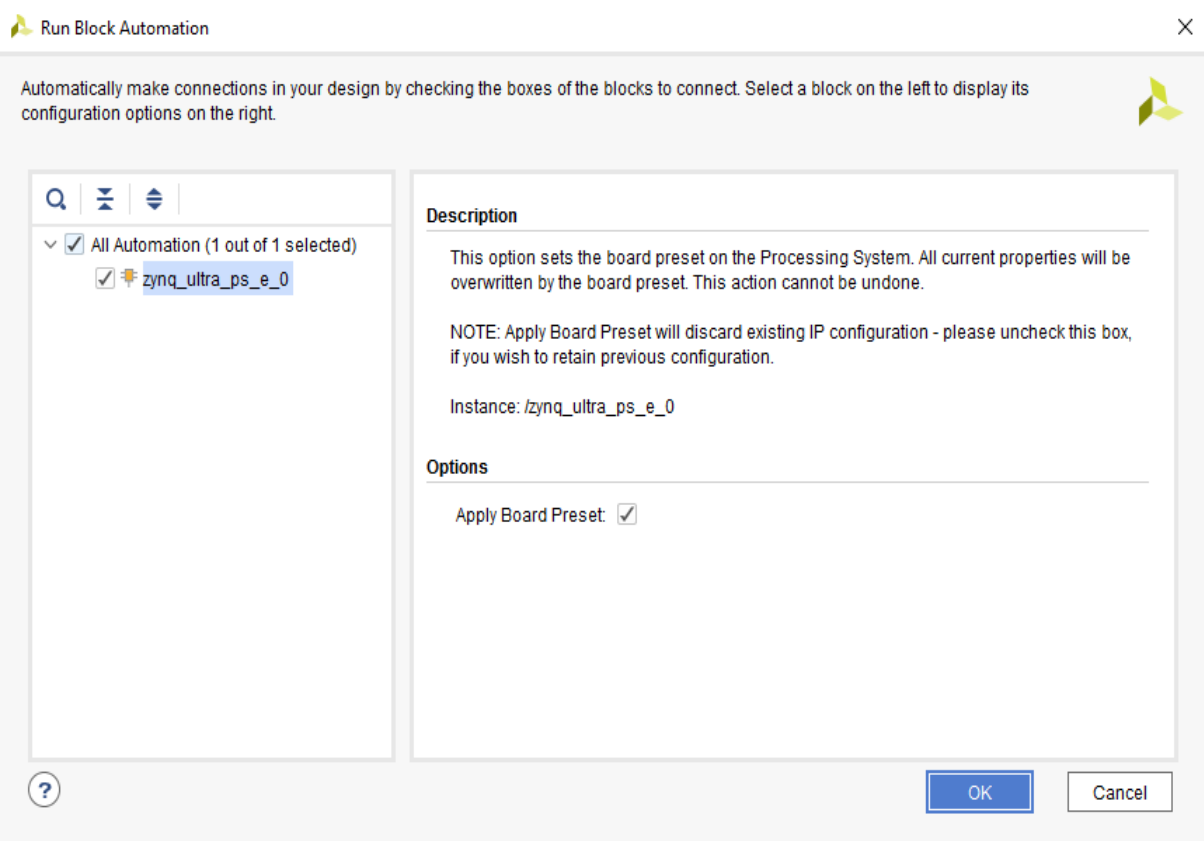


Figure 182 - Select Board Preset

Re-customize “zynq_ultra_ps_e_0” IP block

Double-click on the Zynq processor block to launch the “Re-customize IP” configuration dialog box.

Figure 183 - Customizing the Zynq IP

Select the “Switch To Advanced Mode” checkbox under the Page Navigator pane as shown in Figure 184.

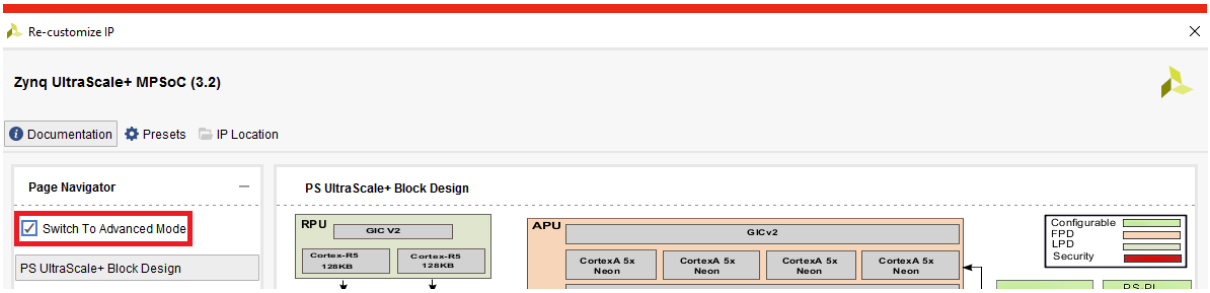


Figure 184 - Select switch to advanced mode

Click on I/O Configuration, and under Peripheral, expand "Low Speed", "I/O Peripherals" and “UART”. Uncheck “UART 1”.

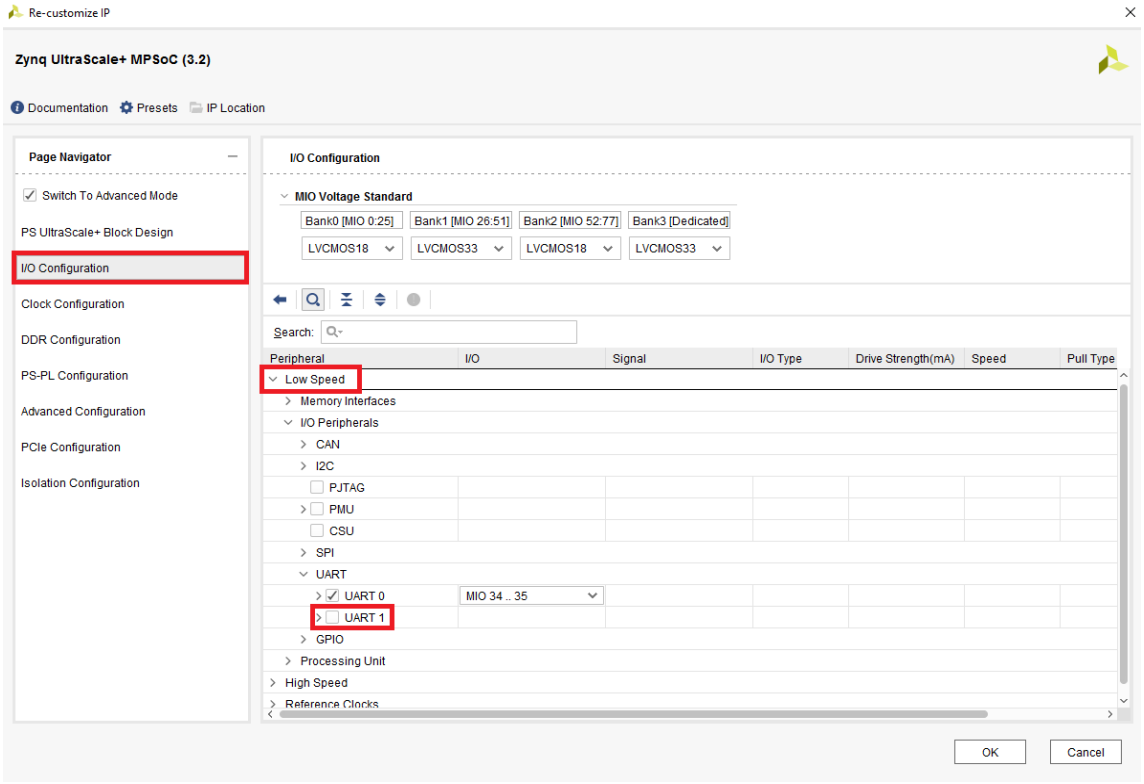


Figure 185 - Exclude UART 1

Similarly, expand "High Speed" and select the PCIe checkbox. Select “MIO 31” from the "Endpoint Mode Reset" drop-down menu.

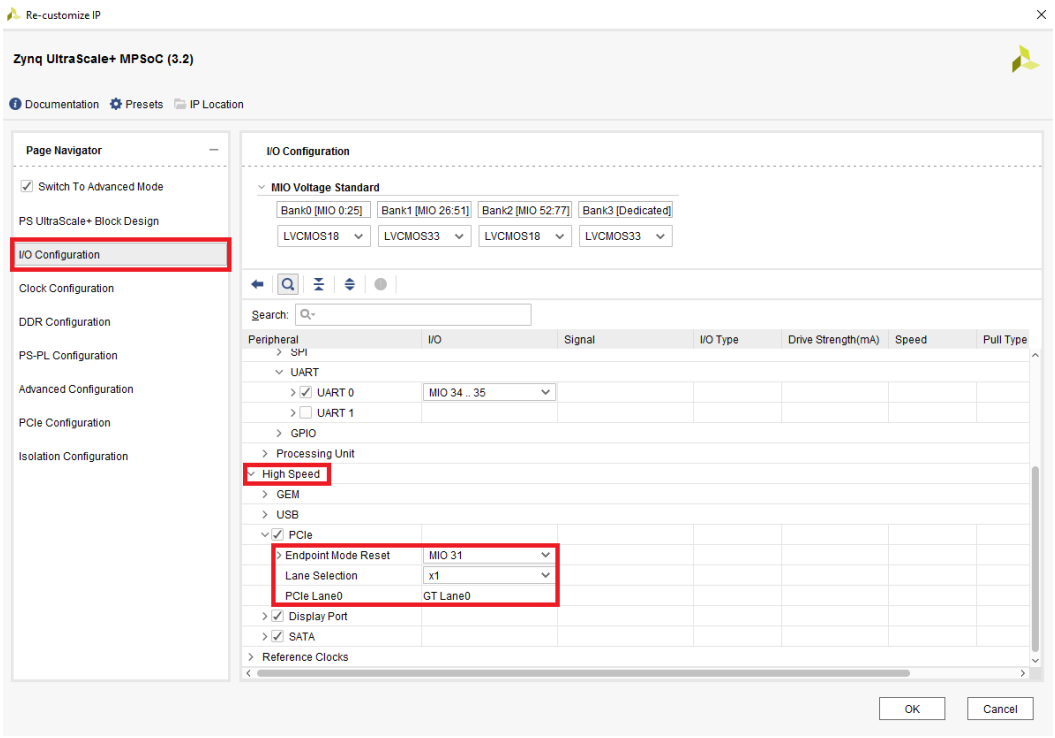


Figure 186 - Configure PCIe

Select “PCIe Configuration under the Page Navigator pane. Expand "Basic Settings" and "Device IDs" drop-down list and ensure their configuration settings are as shown in Figure 187 by editing the necessary fields.

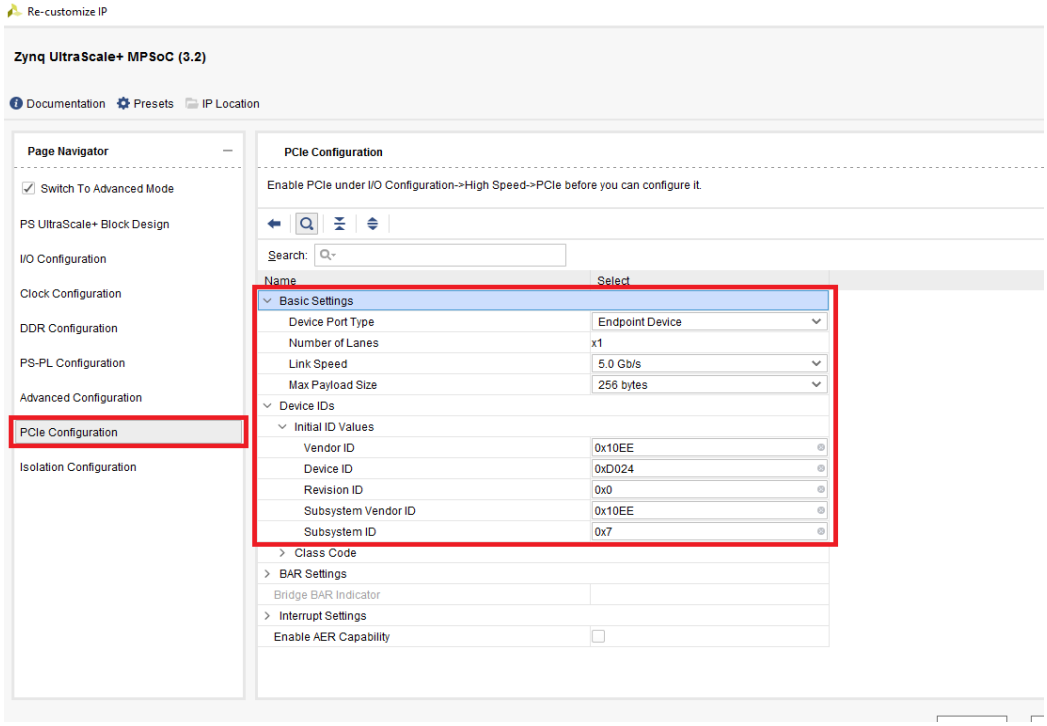


Figure 187 - Checking PCIe configuration setting

Similarly, expand the "BAR Settings" dropdown menu and select BAR0, BAR1 and BAR2 as shown in Figure 188.

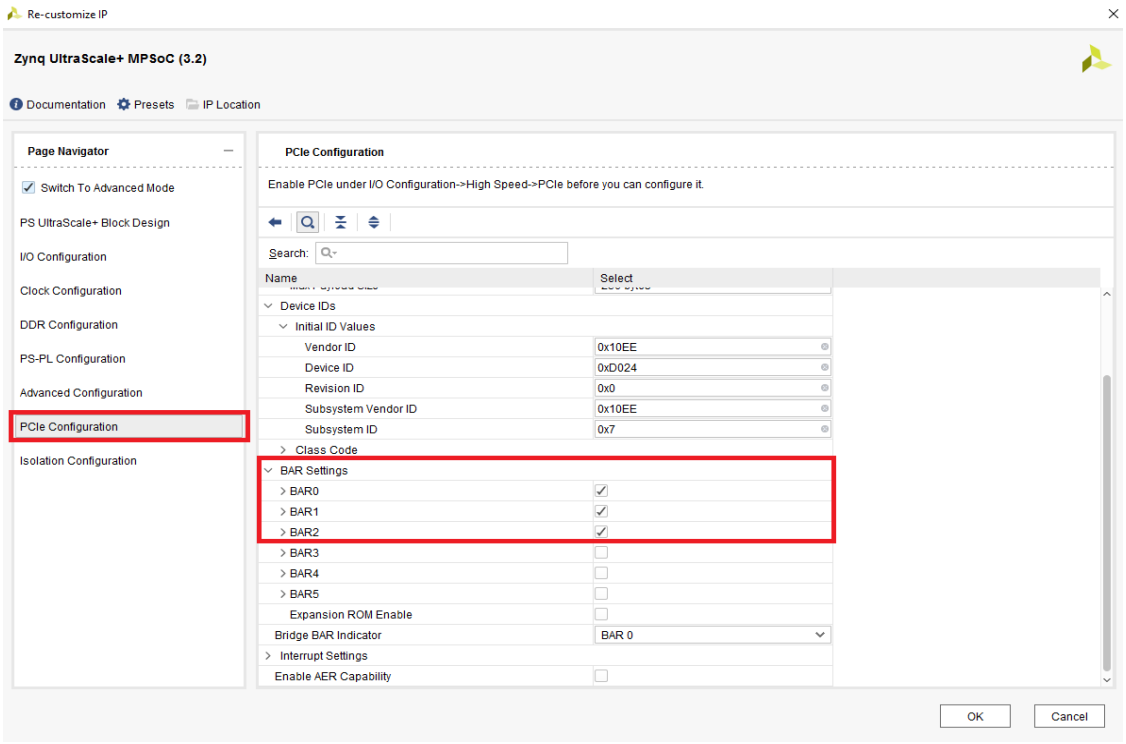


Figure 188 - Include BAR0, BAR1 and BAR2

Expand “BAR1” dropdown list and change "Size value" to 32 Kilobytes. Similarly, expand the BAR2 dropdown list and change "Size value" to 64 Kilobytes.

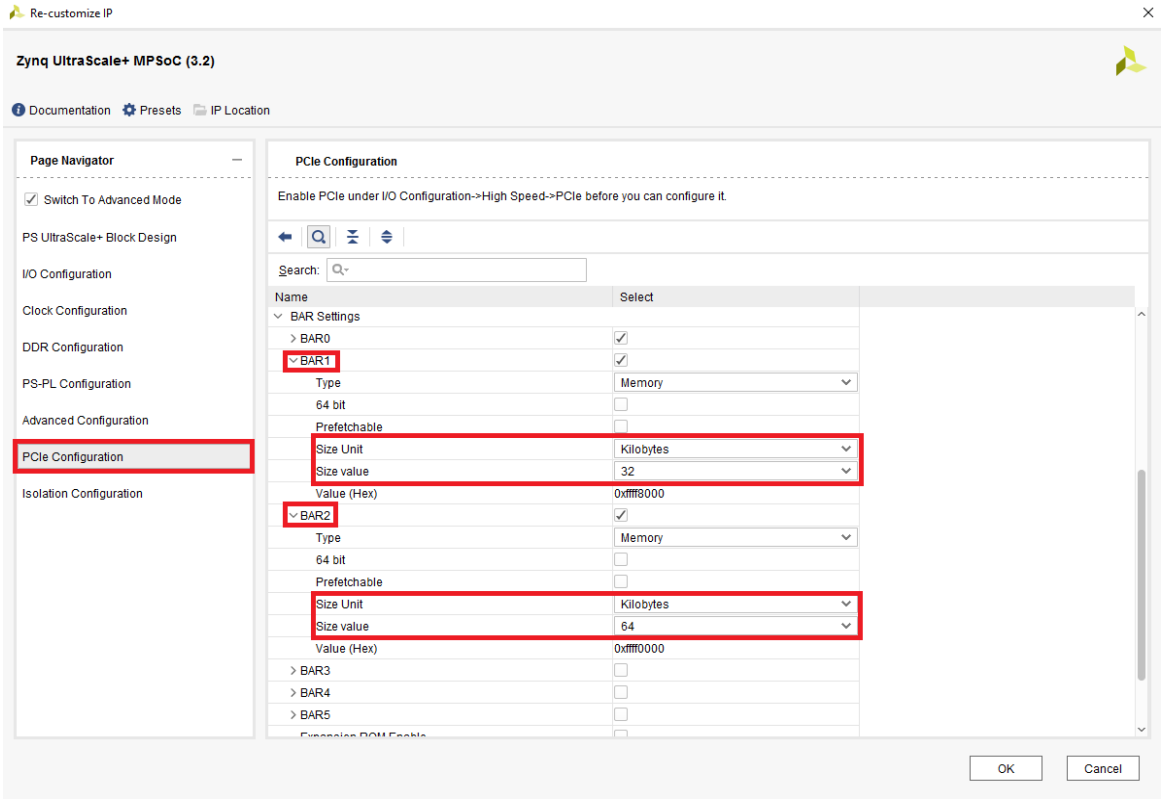


Figure 189 - Change size unit and size value for both BAR1 and BAR2

Expand the "Interrupt Settings" drop-down list and, from the list, expand the "Legacy Interrupt Settings" and "MSI Capabilities" drop-down menus. Select the "Enable MSI Capability Structure" checkbox and change "Multiple Message Capable" to "4 Vector", then click OK as shown in Figure 190.

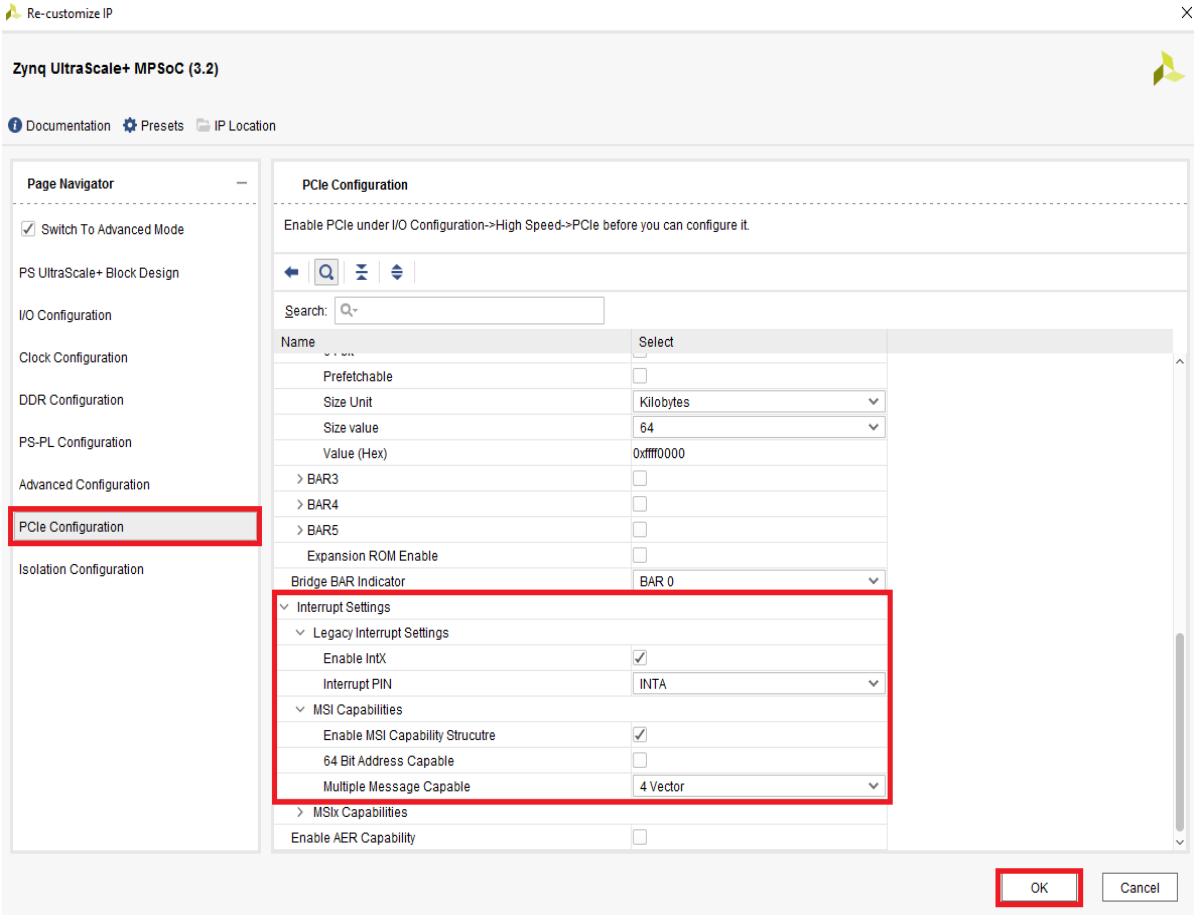


Figure 190 - Configuring interrupt settings

Click on “Run Connection Automation” as shown in Figure 191.

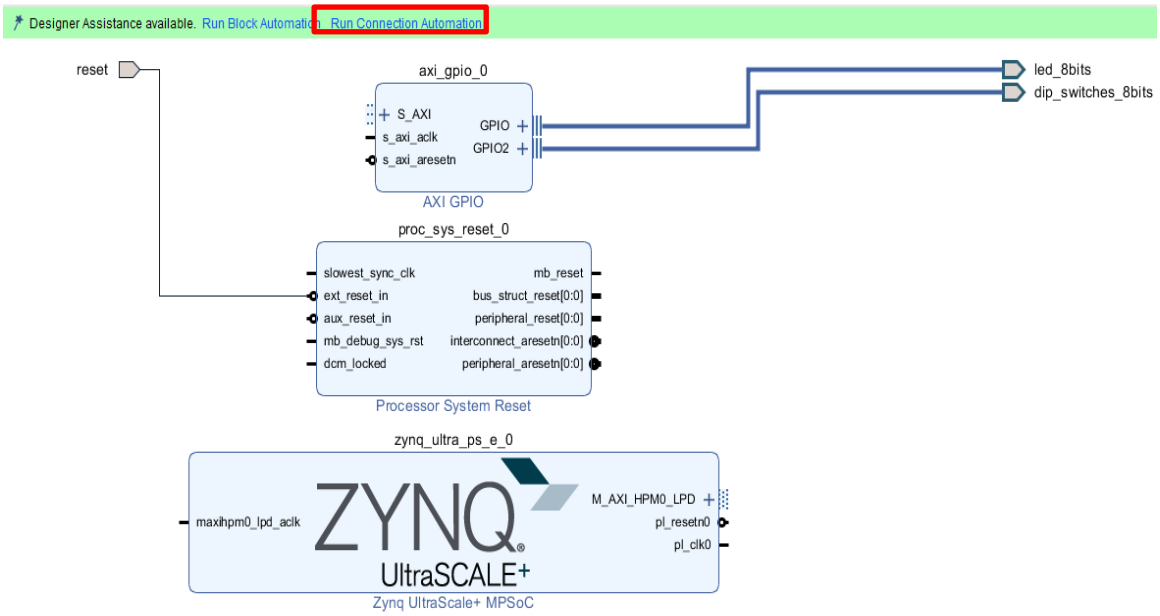


Figure 191 - Run connection automation

Tick the “All Automation” checkbox. Click “OK”.

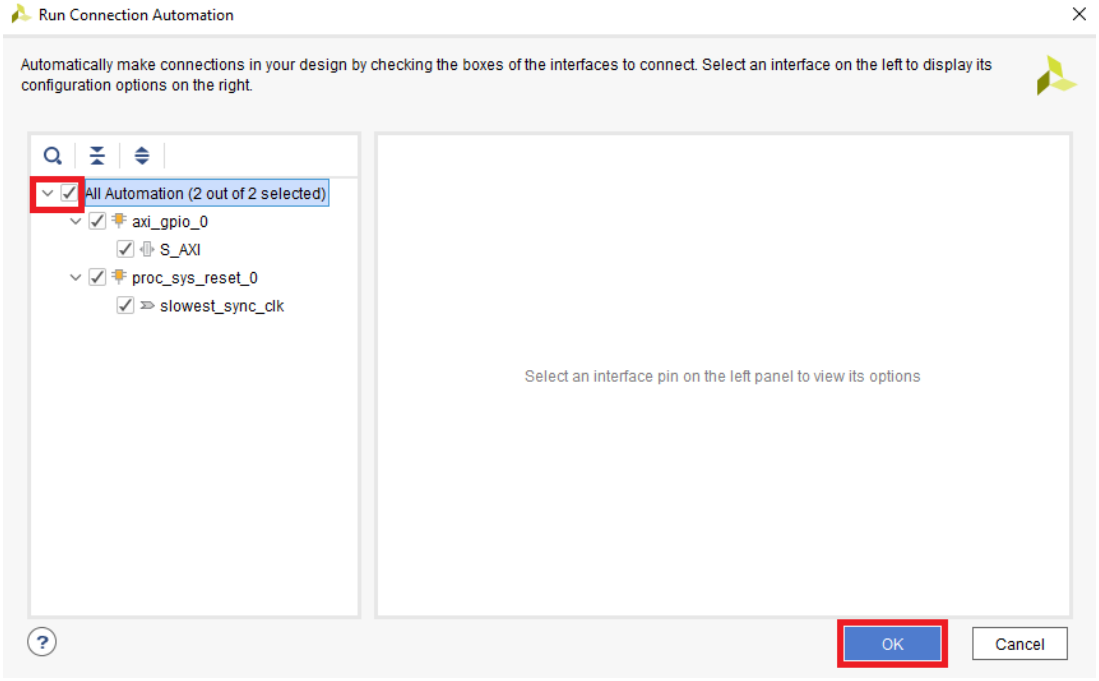


Figure 192 - Select all automation

Regenerate the layout by clicking on the “Re-generate layout” icon as shown in Figure 193 below.

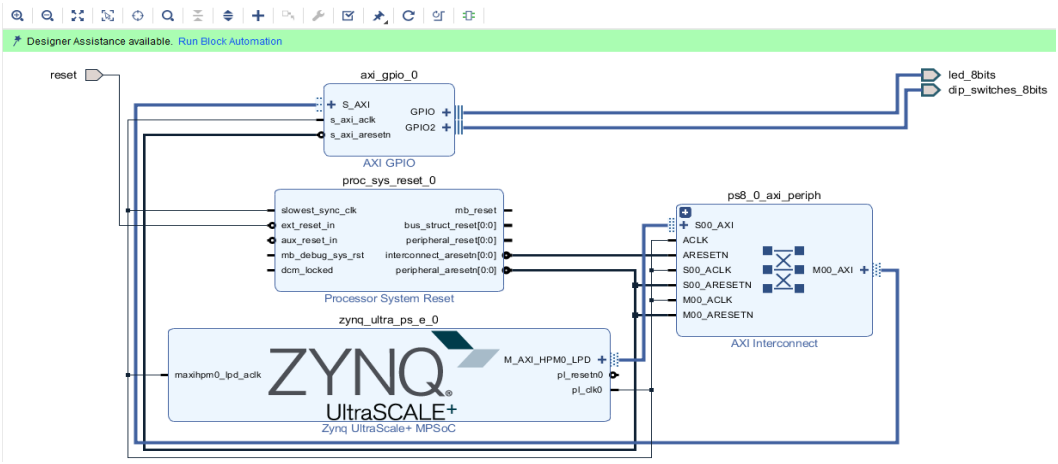


Figure 193 - Regenerate layout

Right-click on the block design window and select “Add IP”. Search for “AXI BRAM Controller”. Double-click to add the IP block.

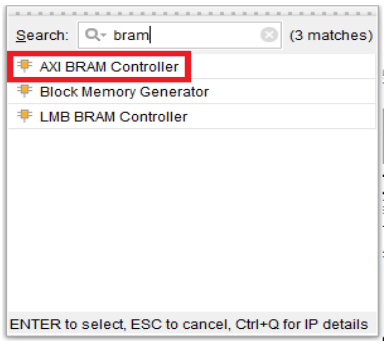


Figure 194 - Add AXI BRAM Controller

The figure below shows the added AXI BRAM Controller IP.

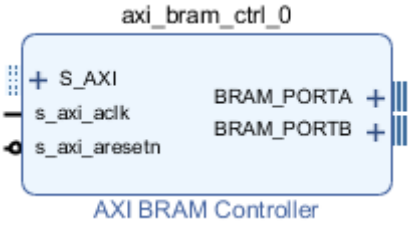


Figure 195 - AXI BRAM Controller

Click “Run Connection Automation” as shown in Figure 196.

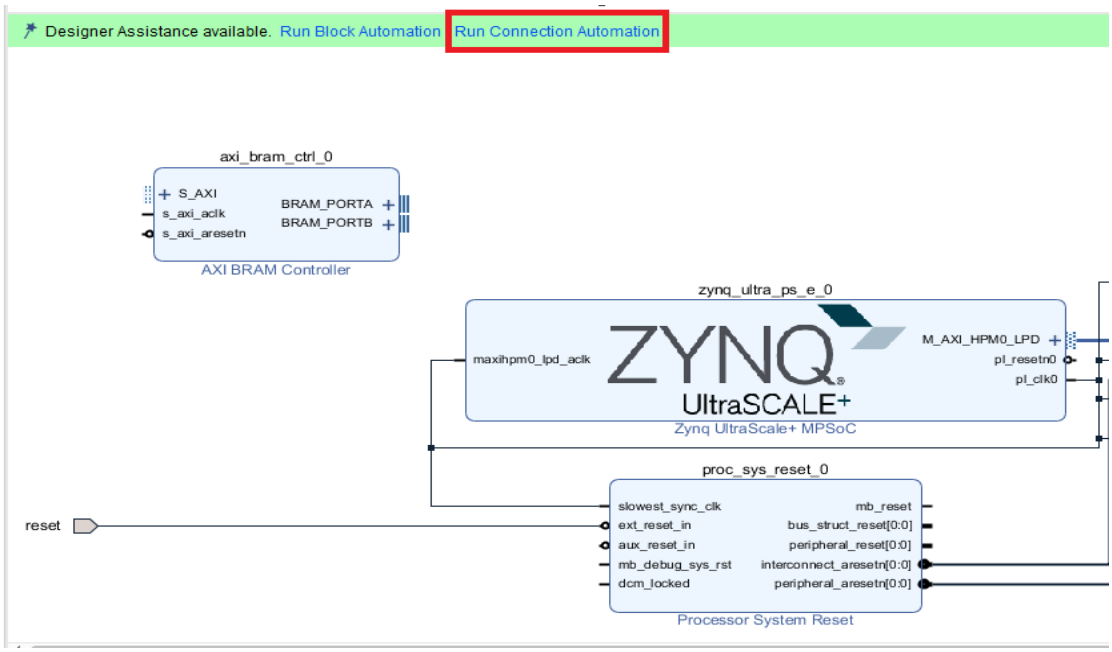


Figure 196 - Run connection automation

Select the “All Automation” checkbox. Click “OK”.

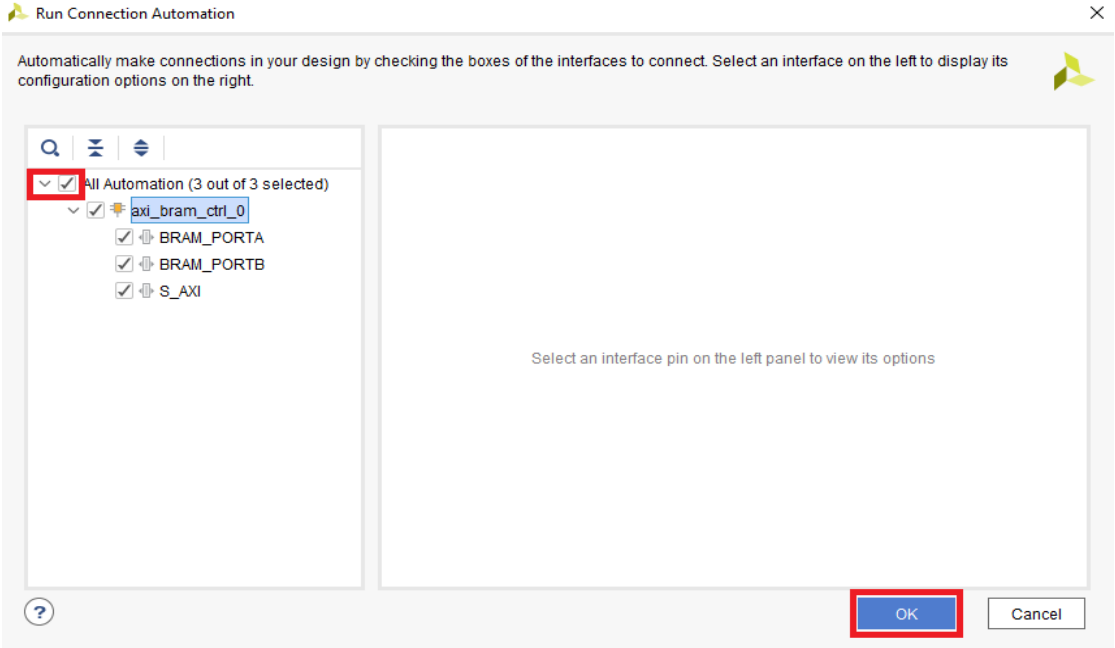


Figure 197 - Select all automation

Regenerate layout by clicking on the icon as shown in Figure 198.

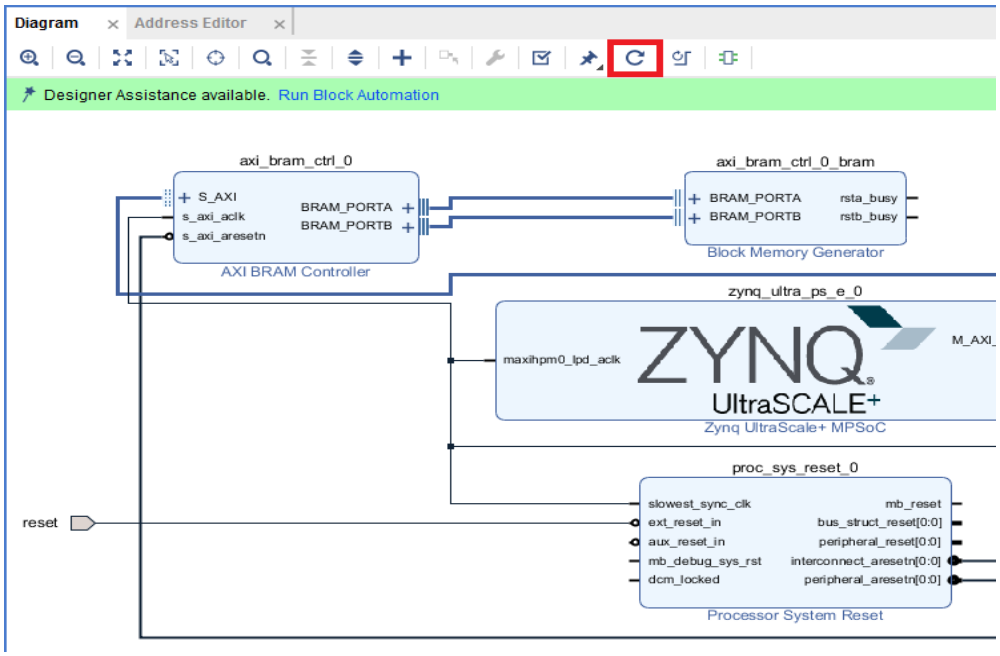


Figure 198 - Regenerate layout

Right-click on the block design window and select “Add IP”. Search for “System ILA”. Double-click on it to add the IP block.

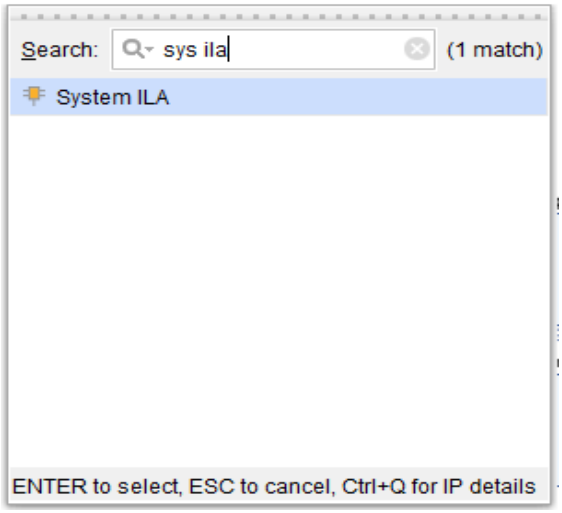


Figure 199 - System ILA

The figure below shows the added System ILA IP block.

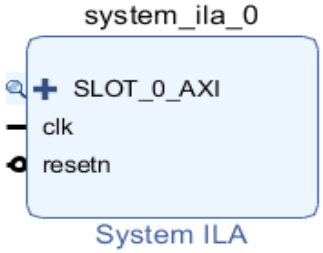


Figure 200 - System ILA IP block

Making other connections

From a particular port, click and hold the mouse button and drag to the desired connection point and then release the mouse to make connection between the two points. For example, connecting [system_ila_0{SLOT_0_AXI}] to [zynq_ultra_ps_e_0{M_AXI_HPM0_LPD}] means connecting the SLOT_0_AXI port of the system_ila_0 IP block to the M_AXI_HPM0_LPD port of the zynq_ultra_ps_e_0 IP block. This connection is shown below.

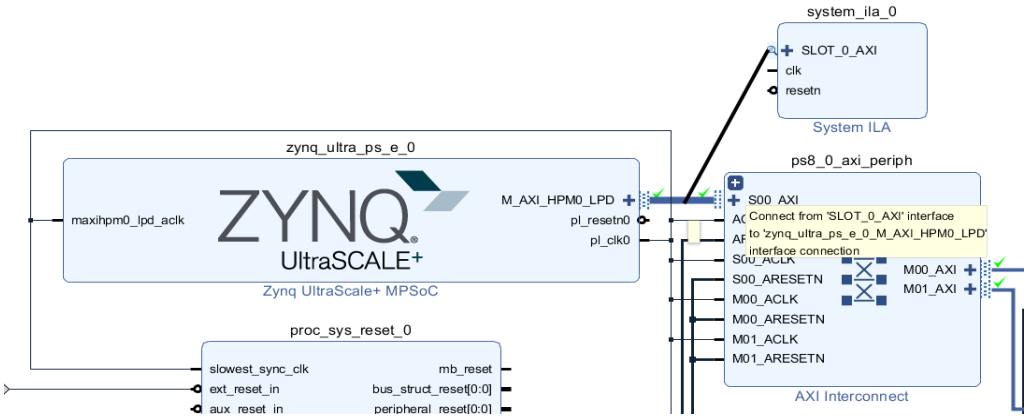


Figure 201 - Making connection

Connect the System ILA IP block to the following nets:
[system_ila_0{clk}] to [zynq_ultra_ps_e_0 {pl_clk}];
[system_ila_0{reset}] to [ps8_0_axi_periph{S00_ARESETN}];

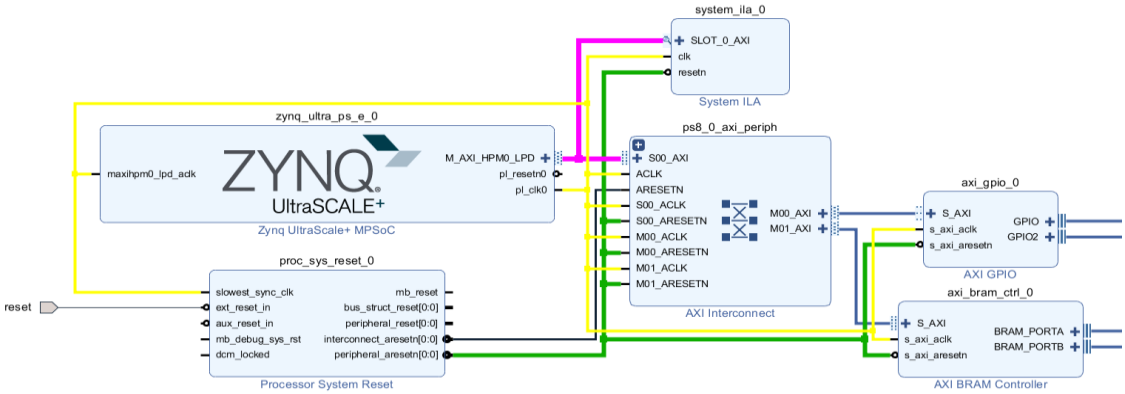


Figure 202 - Making connection

Regenerate the layout by clicking the icon as shown below.

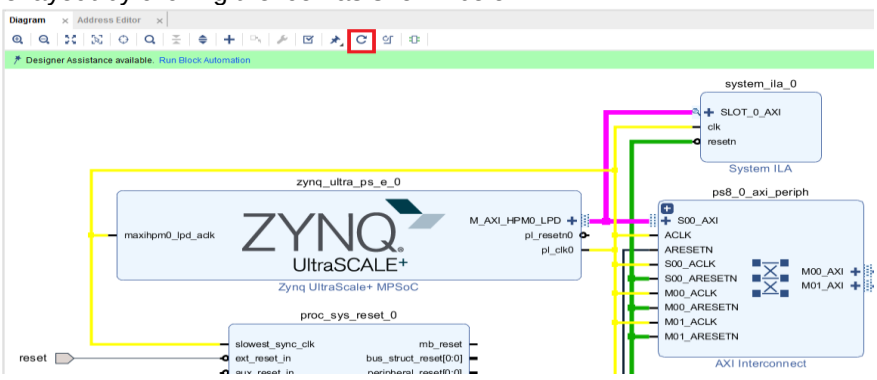


Figure 203 - Regenerate layout

Click the “Address Editor” tab and check that the “Range” is set to 4K.

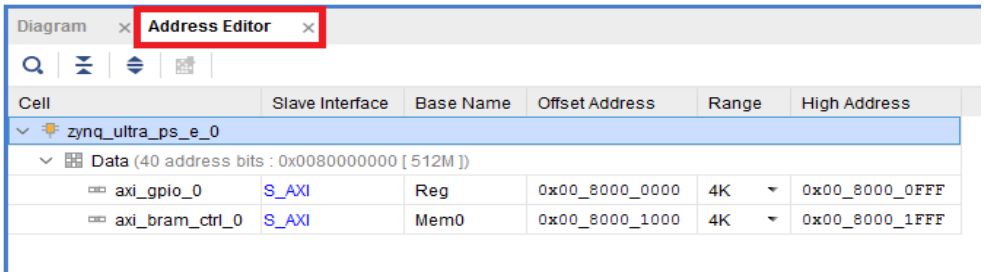


Figure 204 - Address editor

Validate the design.

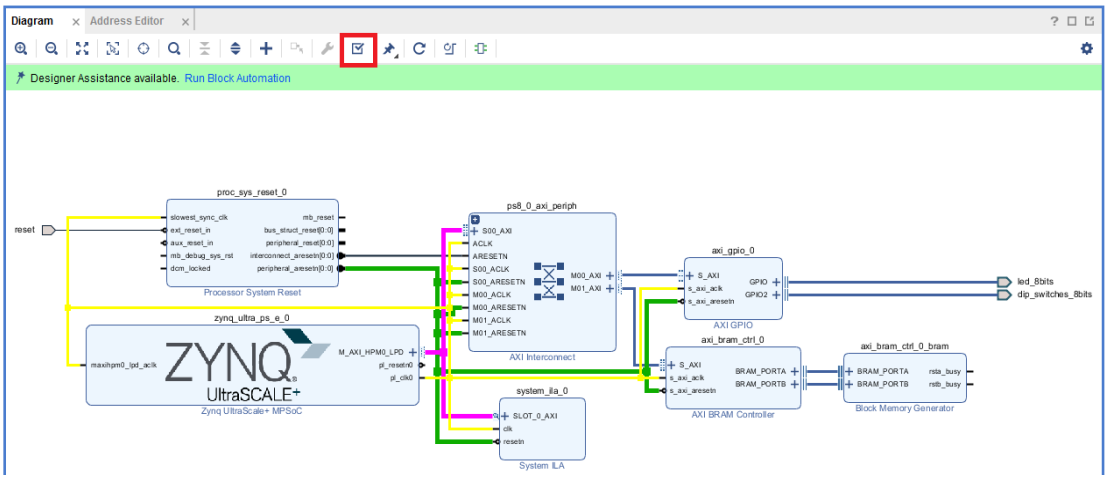


Figure 205 - Validate design

If validation is successful there should be no error as shown in the figure below. Click OK.

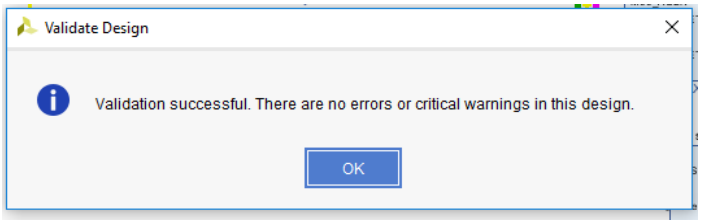


Figure 206 - Validation successful

Click on the save icon to save the block design.

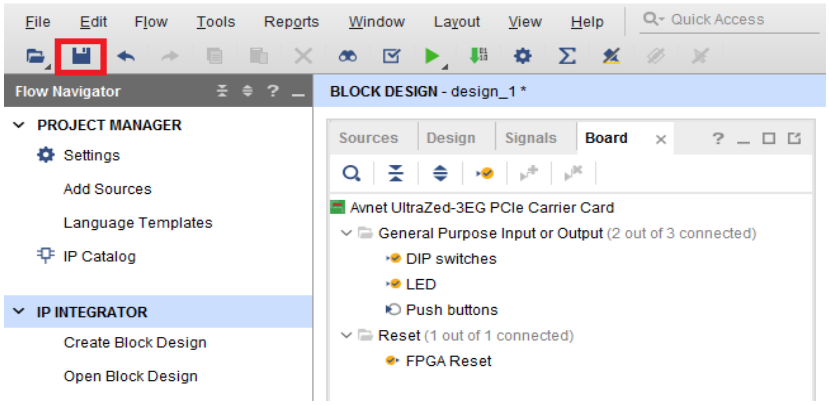


Figure 207 - Save block design

Close the block design window.

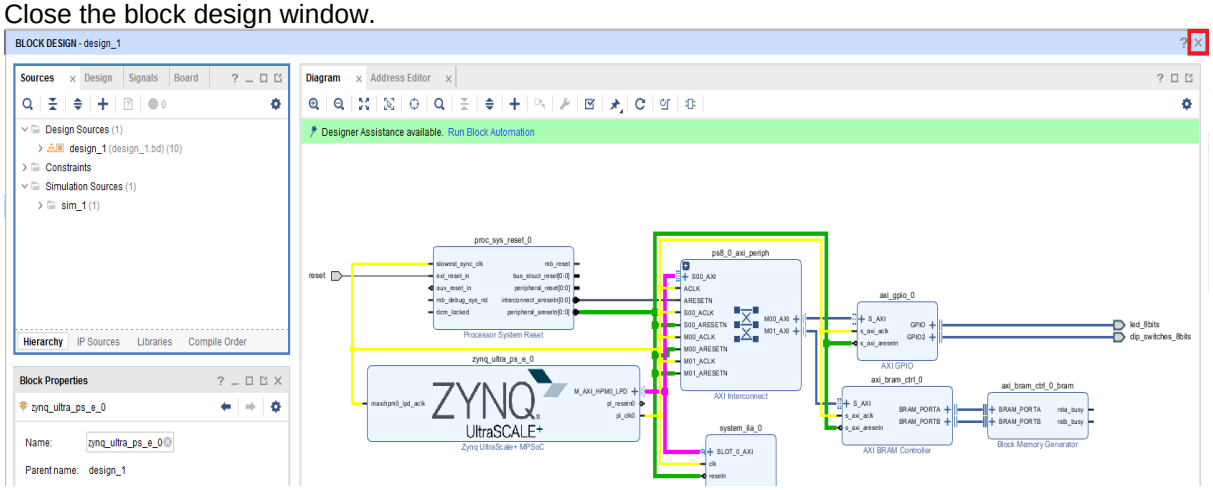


Figure 208 - Close block design

Right-click the “design_1.bd” file and select “Create HDL Wrapper...” from the drop-down list.

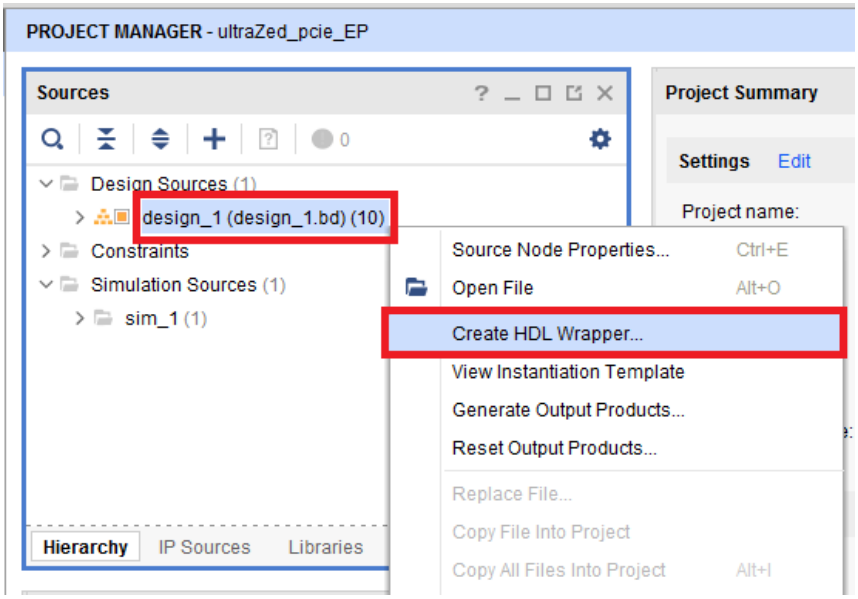


Figure 209 - Create HDL wrapper

Select “Let Vivado manage wrapper and auto-update” and click OK.

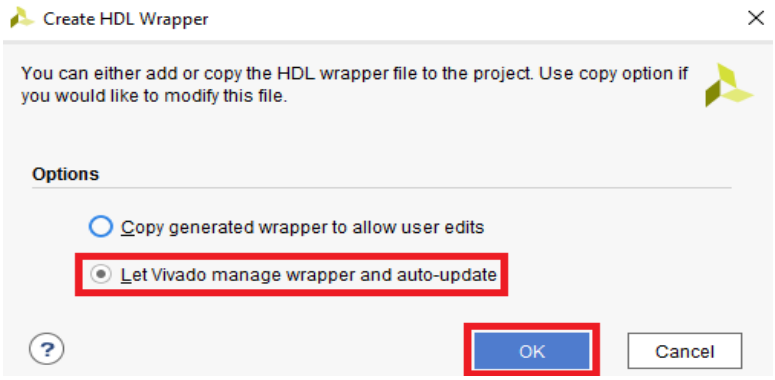


Figure 210 - Use default Vivado setting

Click “Run Synthesis” under the Flow Navigator pane.

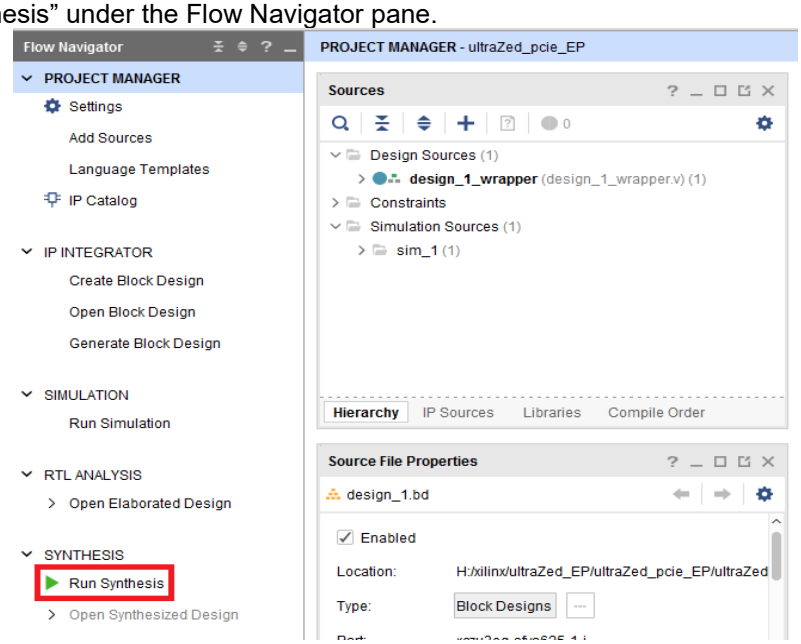


Figure 211 – Synthesizing the design

Leave the default setting as shown in Figure 212 and then click OK.

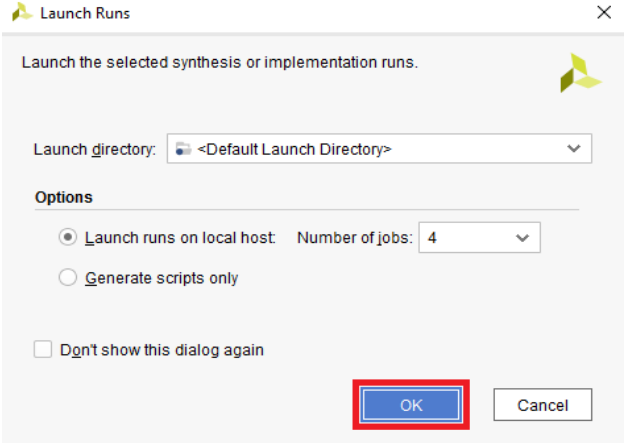


Figure 212 - Accept default settings

When Synthesis is complete, check for critical warnings and errors. Select the “Run Implementation” radio button as shown in Figure 213. Click OK to start implementation.

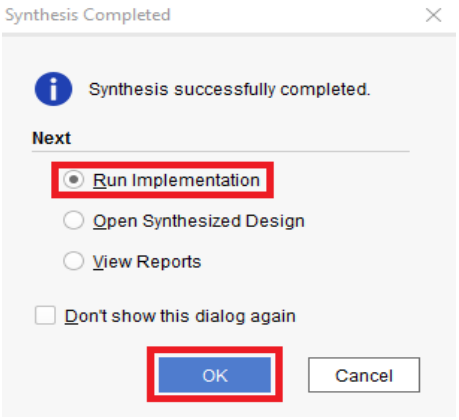


Figure 213 - Implementing the design

In the resulting dialog box leave the default setting as shown below. Click OK to continue.

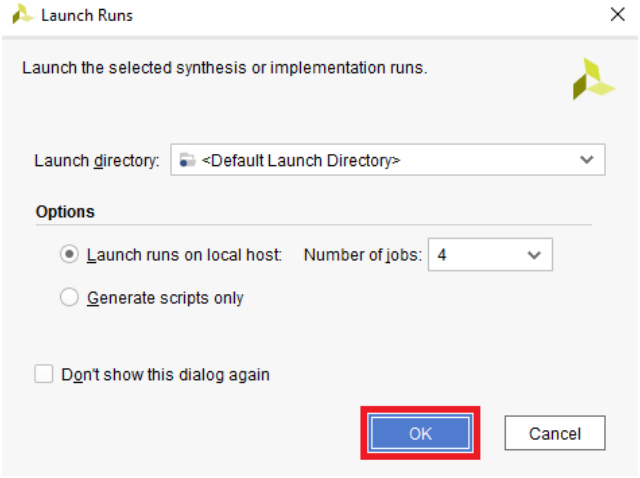


Figure 214 - Accept default settings

When Implementation is complete, select the “Generate Bitstream” radio button. Click OK.

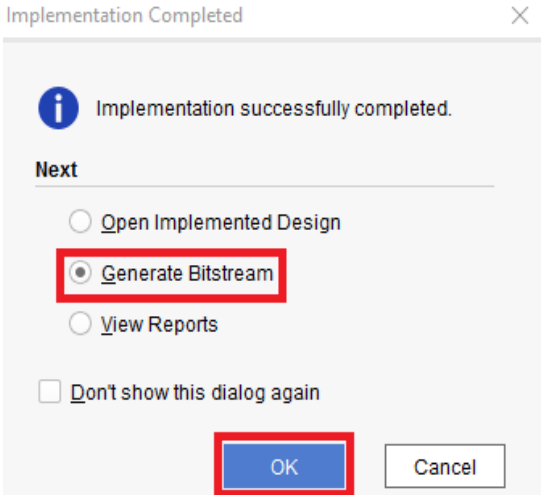


Figure 215 - Generating bitstream

Leave the default setting as shown in Figure 216 and click OK to start the bitstream generation process.

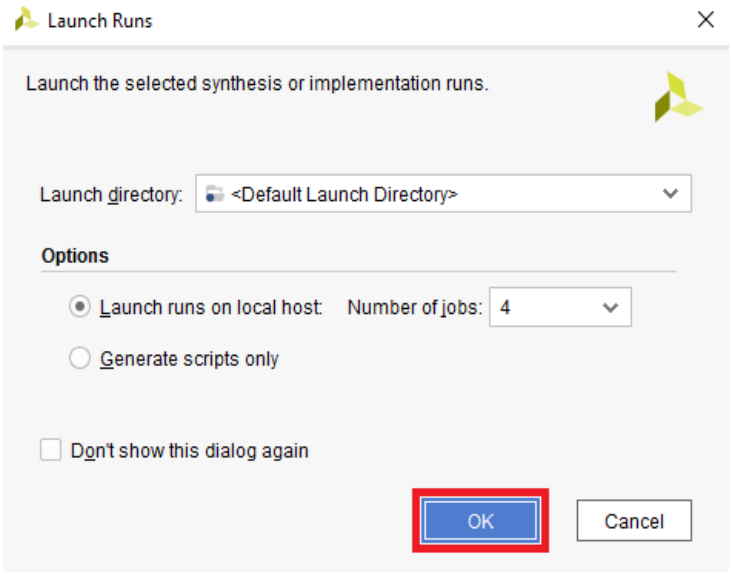


Figure 216 - Accept default settings

A message dialog box will appear indicating that bitstream generation was successful. Click “Cancel”.

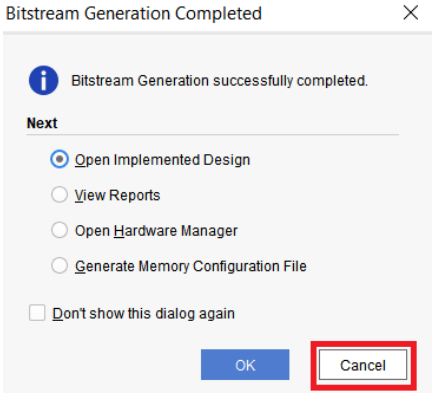


Figure 217 - Close the window of bitstream generation

Go to File > Export > Export Hardware...

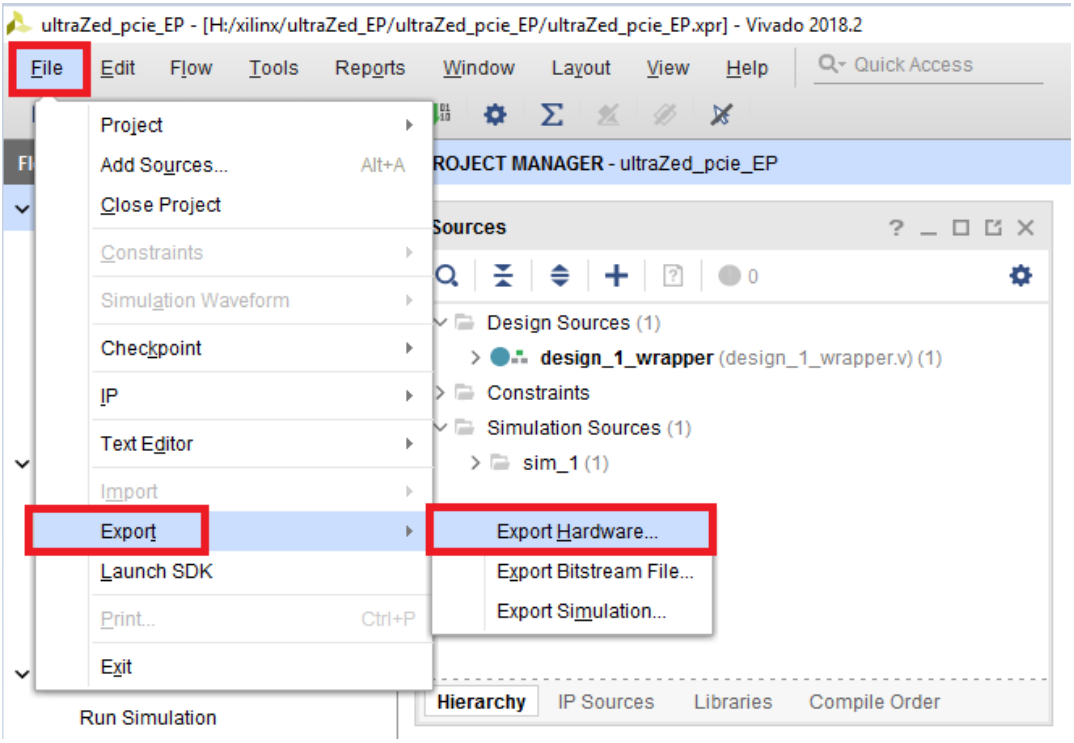


Figure 218 - Export Hardware

Select the “Include bitstream” checkbox and click OK.

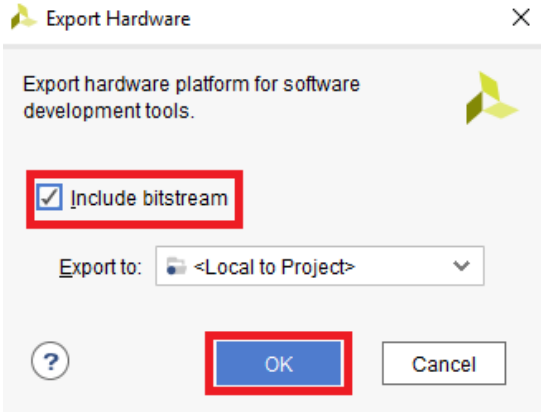


Figure 219 - Hardware file location

SDK tool

Click “File” and select “Launch SDK” from the drop-down list.

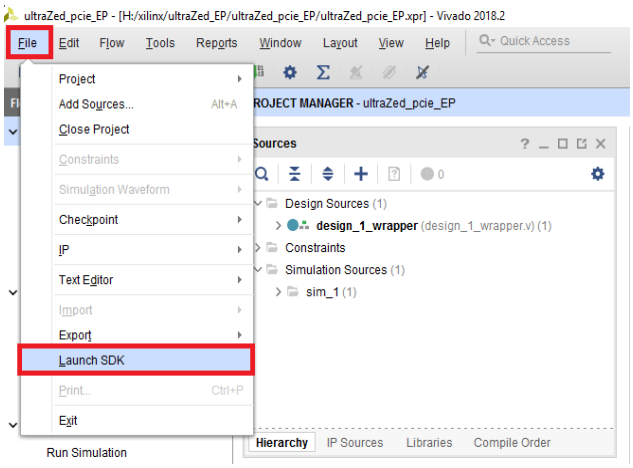


Figure 220 - Launch SDK

Leave the “Exported location” and “Workspace” fields as <Local to Project>. Click OK to launch SDK.

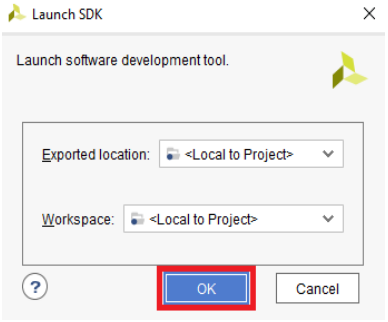


Figure 221 shows the Xilinx SDK environment.

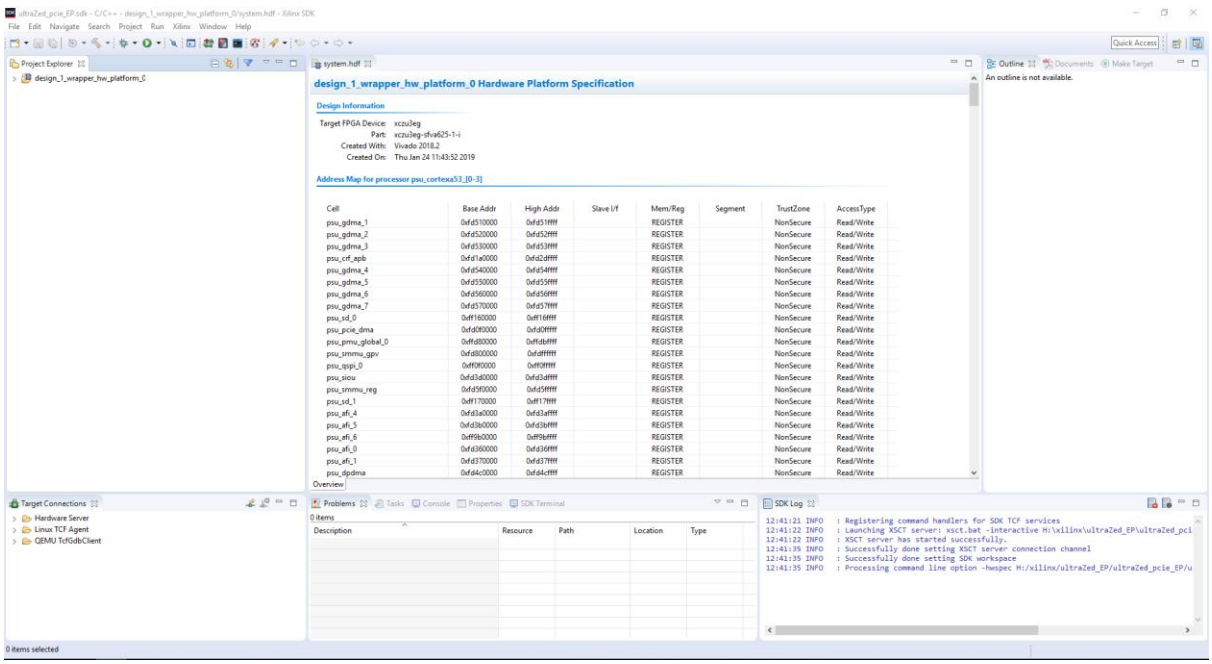


Figure 221 - SDK environment

Go to File >> New >> Application Project.

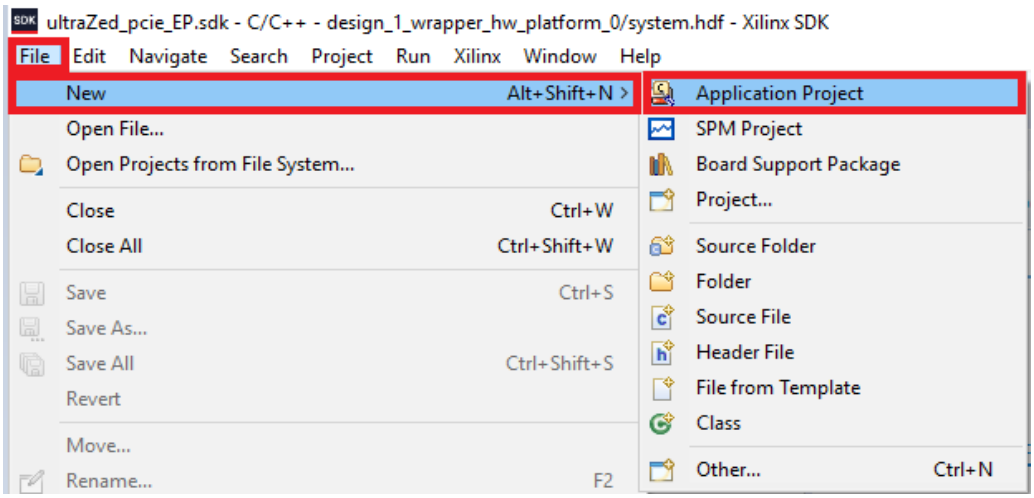


Figure 222 - Create application project

Enter a desired project name for the application project like “FSBL”. Click “Next”.

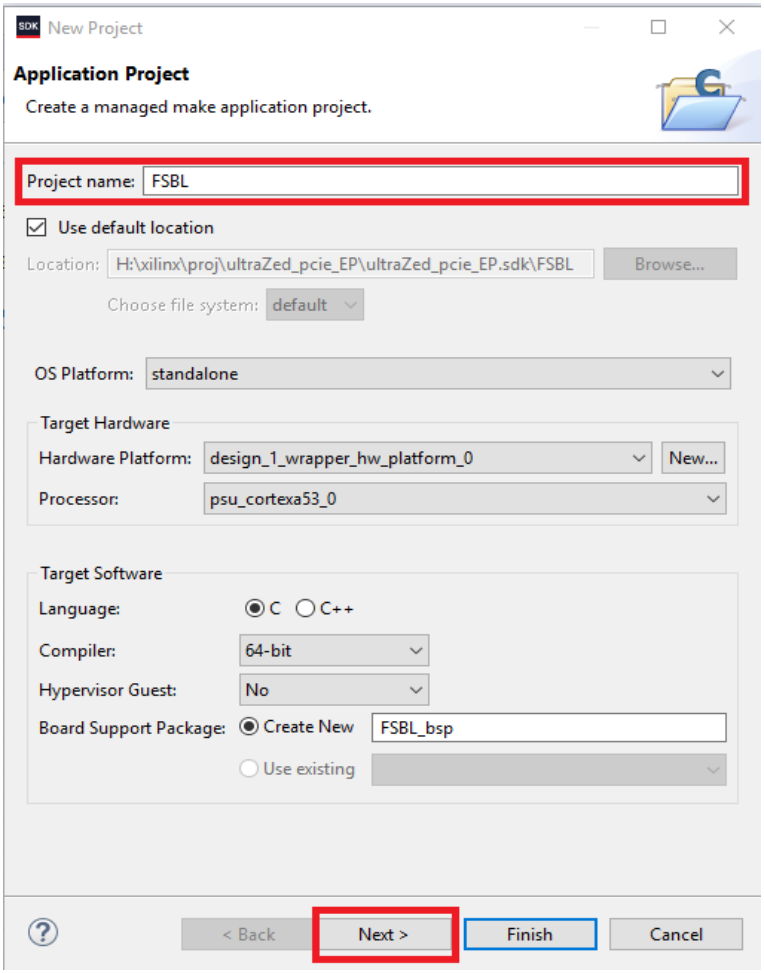


Figure 223 - Application project name

Select “Zynq MP FSBL” from the available templates and click “Finish”.

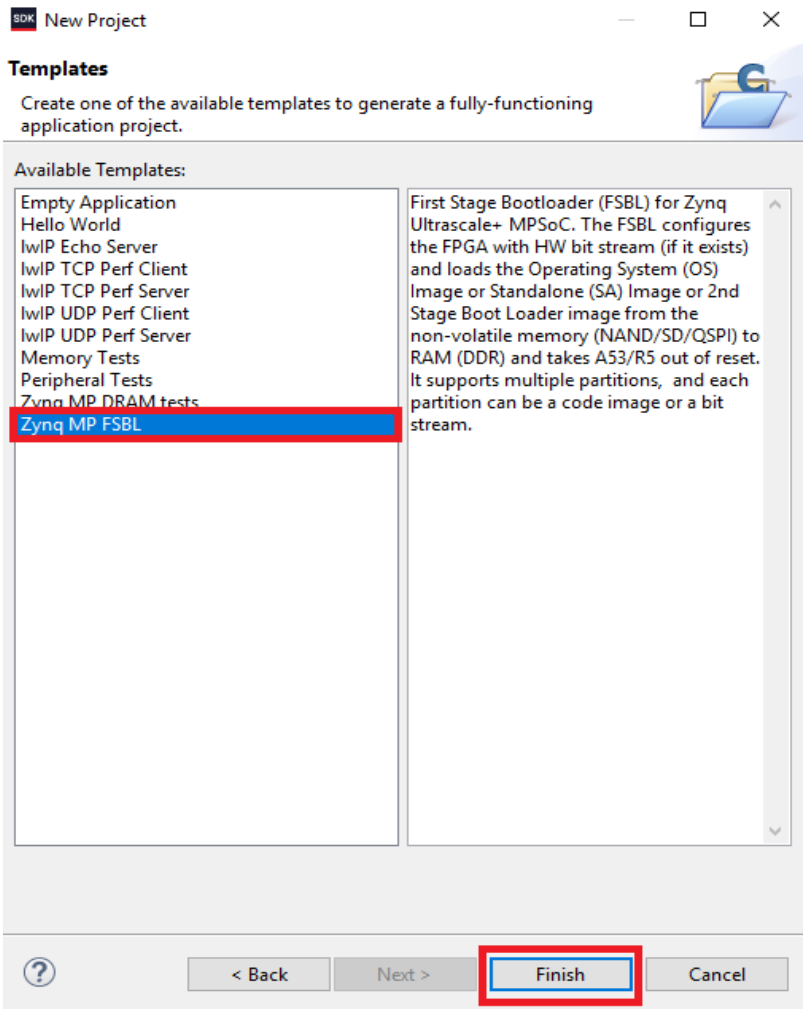


Figure 224 - Application project template

Go to File >> New >> Application Project.

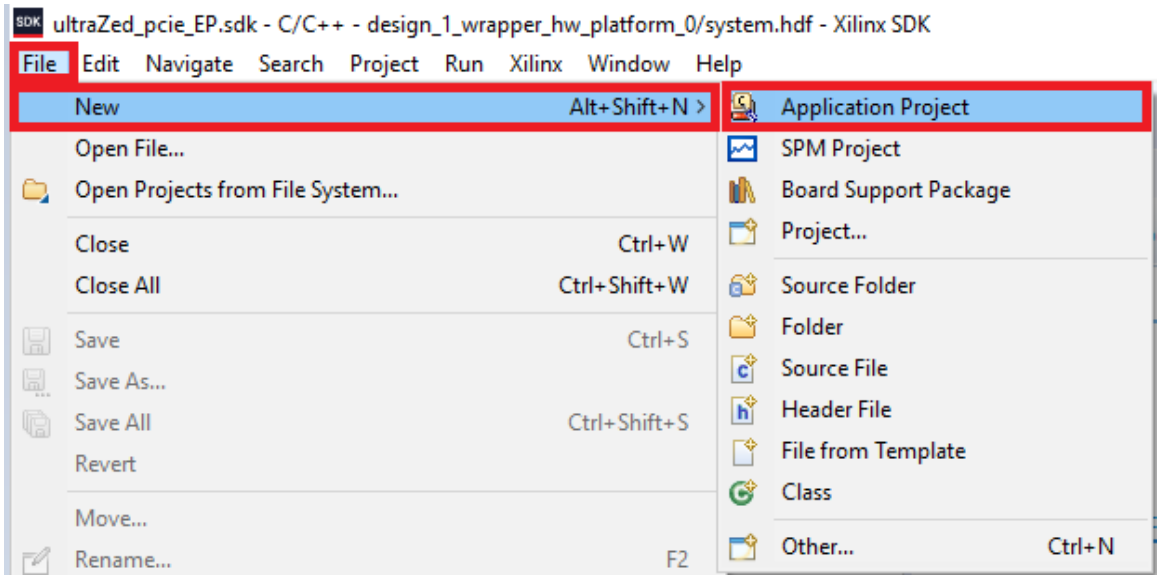


Figure 225 - Create application project

Enter “EP_Initialize” as the project name. Click “Next”.

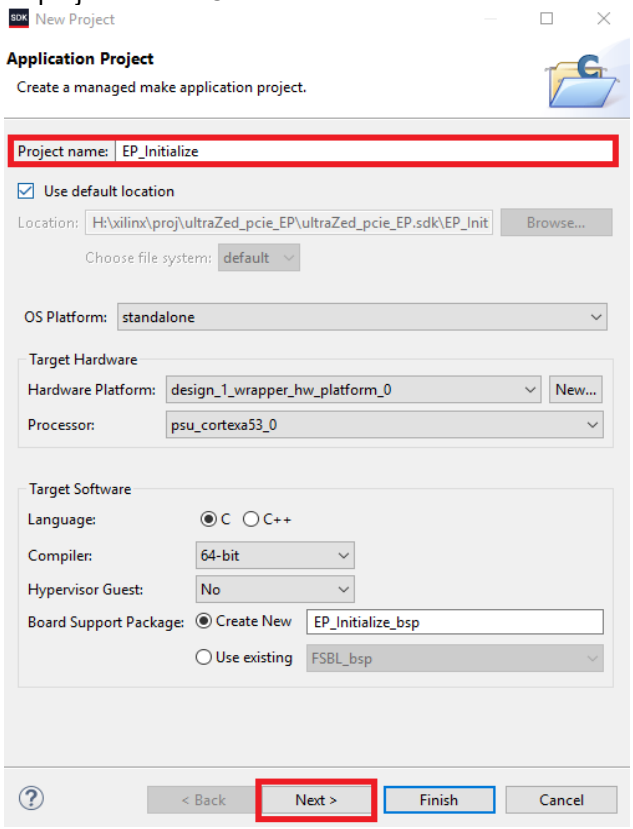


Figure 226 - Application project name

Select “Empty Application” from the “Available Templates” list. Click “Finish”.

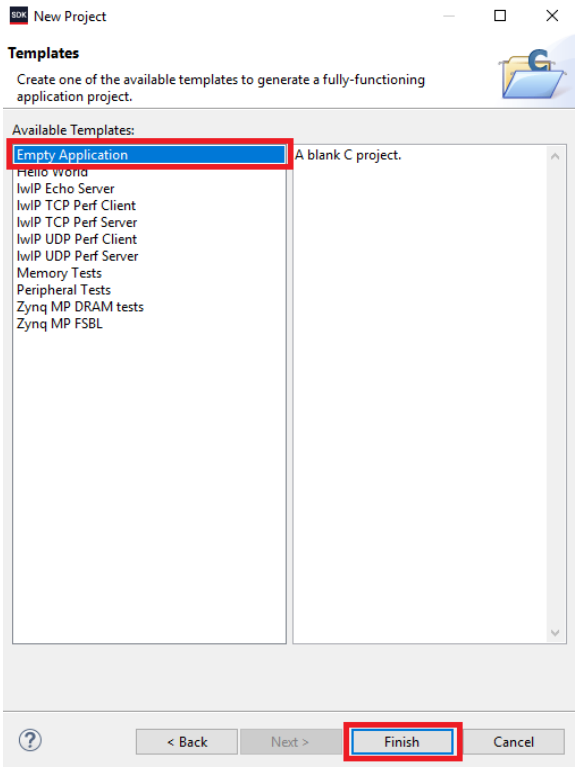


Figure 227 - Project template

Under “Project Explorer” create “pcie_ep_pio” C file under “EP_Initialze” project “src” folder. To do this, right-click on the “src” folder and select “New” and then “Source File”.

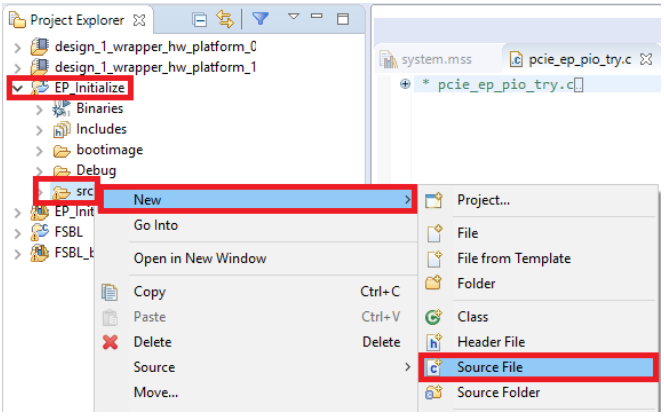


Figure 228 - Source C file

Name the source file “pcie_ep_pio” and click “Finish”.

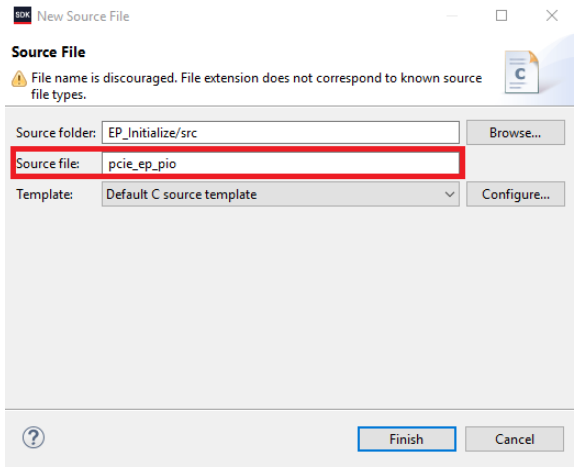


Figure 229 – Naming the C file

Under “Project Explorer” create a “pcie_ep_pio” header file under “EP_Initialze” project “src” folder. To do this, right-click on the “src” folder and select “New” and then “Header File”.

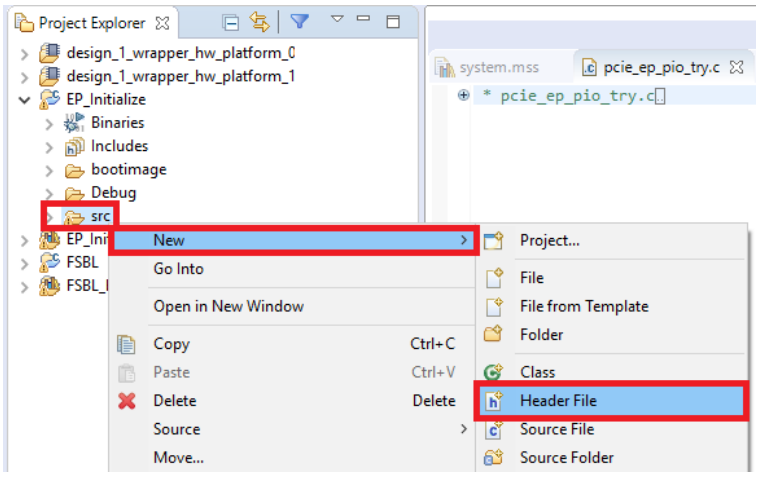


Figure 230 - Header file

Name the header file “pcie_ep_pio” and click “Finish”.

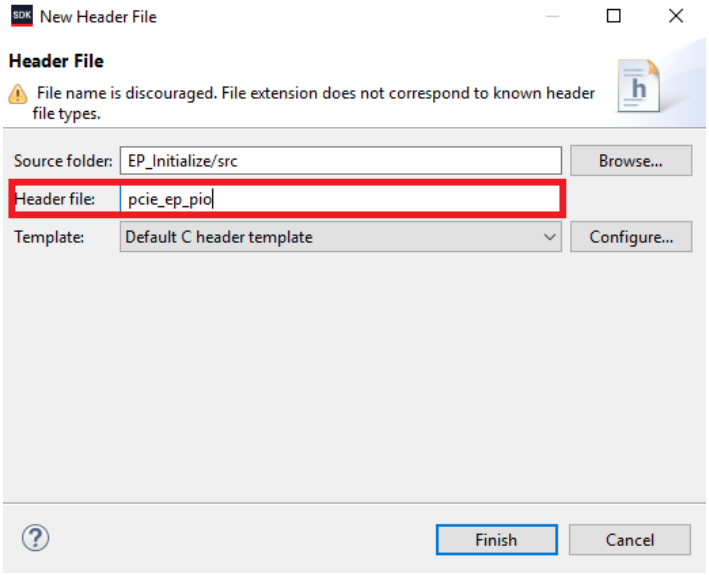


Figure 231 – Naming the header file

At this point, the C file and header file are created under the “src” folder of the “EP_Initialize” project as shown in Figure 232.

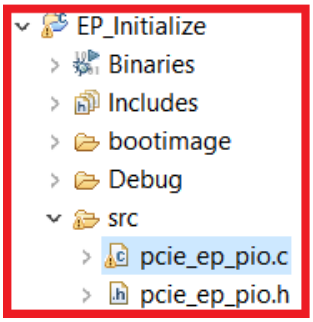


Figure 232 - PCIe header and C file for SDK

Copy the sources from the respective files (attached in http://www.xilinx.com/support/answers/72076.html) to pcie_ep_pio.h and pcie_ep_pio.c.

Click “Xilinx” and select “Create Boot Image” from the drop-down menu.

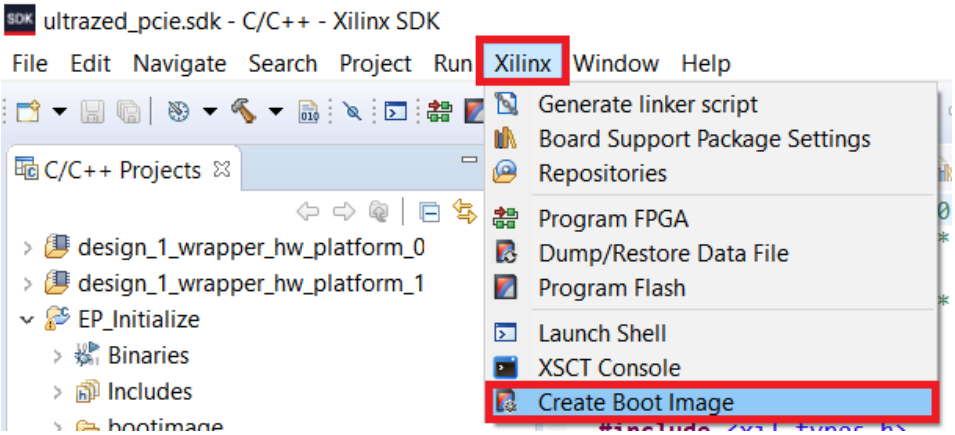


Figure 233 - Create boot image

In the resulting dialog box click on “Create Image”.

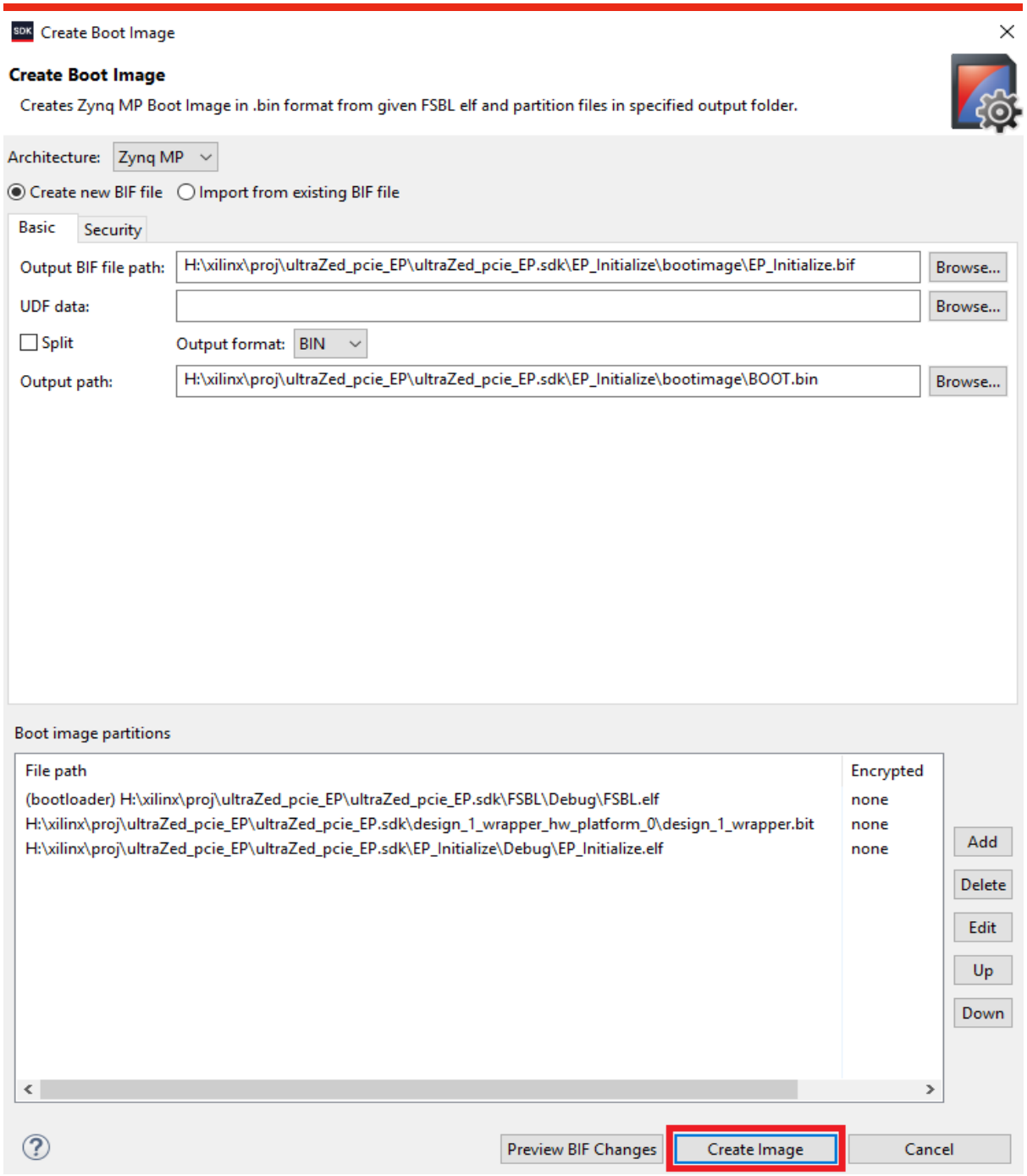


Figure 234 - Create image

Appendix A: ZCU106 Constraints

```
## Project    : The Xilinx PCI Express DMA
## File       : xilinx_xdma_pcie_x0y1.xdc
## Version    : 4.0
##-----------------------------------------------------------------------------
#
# User Configuration
# Link Width   - x1
# Link Speed   - Gen2
# Family       - zynquplus
# Part         - xczu7ev
# Package      - ffvc1156
# Speed grade  - -2
##############################################################################
# User Time Names / User Time Groups / Time Specs
##############################################################################
##
## Free Running Clock is Required for IBERT/DRP operations.
##

##############################################################################################################
##
create_clock -name sys_clk -period 10 [get_ports sys_clk_clk_p]
#create_clock -name sys_clk_125 -period 8 [get_ports sys_clk_125_clk_p]
#
##############################################################################################################
##
set_false_path -from [get_ports sys_rst]
set_property PULLUP true [get_ports sys_rst]
#
#set_property CONFIG_VOLTAGE 1.8 [current_design]
#
##############################################################################################################
##
#set_property LOC [get_package_pins -of_objects [get_bels [get_sites -filter {NAME =~ *COMMON*} -of_objects
[get_iobanks -of_objects [get_sites GTHE4_CHANNEL_X0Y19]]]/REFCLK0P]] [get_ports CLK_IN_D_0_clk_p]
#set_property LOC [get_package_pins -of_objects [get_bels [get_sites -filter {NAME =~ *COMMON*} -of_objects
[get_iobanks -of_objects [get_sites GTHE4_CHANNEL_X0Y19]]]/REFCLK0N]] [get_ports CLK_IN_D_0_clk_n]
#
set_property LOC V8 [get_ports sys_clk_clk_p]
set_property LOC V7 [get_ports sys_clk_clk_n]

#set_property LOC H9 [get_ports sys_clk_125_clk_p]
#set_property IOSTANDARD  LVDS [get_ports "sys_clk_125_clk_p"] ;

##############################################################################################################
##
#
set_property PACKAGE_PIN AL11    [get_ports "led_0"] ;# LED 0 Bank  66 VCCO - VCC1V2   - IO_L8P_T1L_N2_AD5P_66
set_property IOSTANDARD  LVCMOS12 [get_ports "led_0"] ;# LED 0 Bank  66 VCCO - VCC1V2   - IO_L8P_T1L_N2_AD5P_66
set_property  PACKAGE_PIN  AL13          [get_ports  "led_1"]  ;#  LED  1  Bank    66  VCCO  -  VCC1V2      -
IO_L7N_T1L_N1_QBC_AD13N_66
set_property  IOSTANDARD    LVCMOS12  [get_ports  "led_1"]  ;#  LED  1  Bank    66  VCCO  -  VCC1V2      -
IO_L7N_T1L_N1_QBC_AD13N_66
set_property PACKAGE_PIN AL10     [get_ports "sys_rst"] ;# Bank  66 VCCO - VCC1V2   - IO_L8N_T1L_N3_AD5N_66
set_property IOSTANDARD  LVCMOS12 [get_ports "sys_rst"] ;# Bank  66 VCCO - VCC1V2   - IO_L8N_T1L_N3_AD5N_66
#set_property PACKAGE_PIN F17     [get_ports "sys_reset"] ;# Bank  66 VCCO - VCC1V2   - IO_L8N_T1L_N3_AD5N_66
#set_property IOSTANDARD  LVCMOS18 [get_ports "sys_reset"] ;# Bank  66 VCCO - VCC1V2   - IO_L8N_T1L_N3_AD5N_66

#set_property PACKAGE_PIN AH17     [get_ports "UART_0_rxd"]
#set_property PACKAGE_PIN AL17     [get_ports "UART_0_txd"]
#set_property IOSTANDARD  LVCMOS12 [get_ports "UART_0_rxd"] ;
#set_property IOSTANDARD  LVCMOS12 [get_ports "UART_0_txd"] ;

#GPIO 8-POLE DIP SW
#set_property PACKAGE_PIN A17 [get_ports "dip_switches_8bits_tri_i[0]"] ;
#set_property IOSTANDARD LVCMOS18 [get_ports "dip_switches_8bits_tri_i[0]"] ;
#set_property PACKAGE_PIN A16 [get_ports "dip_switches_8bits_tri_i[1]"] ;
#set_property IOSTANDARD LVCMOS18 [get_ports "dip_switches_8bits_tri_i[1]"] ;
#set_property PACKAGE_PIN B16 [get_ports "dip_switches_8bits_tri_i[2]"] ;
#set_property IOSTANDARD LVCMOS18 [get_ports "dip_switches_8bits_tri_i[2]"] ;
#set_property PACKAGE_PIN B15 [get_ports "dip_switches_8bits_tri_i[3]"] ;
#set_property IOSTANDARD LVCMOS18 [get_ports "dip_switches_8bits_tri_i[3]"] ;
#set_property PACKAGE_PIN A15 [get_ports "dip_switches_8bits_tri_i[4]"] ;
#set_property IOSTANDARD LVCMOS18 [get_ports "dip_switches_8bits_tri_i[4]"] ;
#set_property PACKAGE_PIN A14 [get_ports "dip_switches_8bits_tri_i[5]"] ;
#set_property IOSTANDARD LVCMOS18 [get_ports "dip_switches_8bits_tri_i[5]"] ;
```

```
#set_property PACKAGE_PIN B14 [get_ports "dip_switches_8bits_tri_i[6]"] ;
#set_property IOSTANDARD LVCMOS18 [get_ports "dip_switches_8bits_tri_i[6]"] ;
#set_property PACKAGE_PIN B13 [get_ports "dip_switches_8bits_tri_i[7]"] ;
#set_property IOSTANDARD LVCMOS18 [get_ports "dip_switches_8bits_tri_i[7]"] ;

set_false_path -to [get_pins -hier *sync_reg[0]/D]
set_false_path -to [get_ports -filter NAME=~led_*]


#move to dummy locations
set_property     LOC     GTHE4_CHANNEL_X0Y0     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[3].*gen_gthe4_channel_inst[0].GTHE4_CHANNEL_PRIM_INST}]
set_property     LOC     GTHE4_CHANNEL_X0Y1     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[3].*gen_gthe4_channel_inst[1].GTHE4_CHANNEL_PRIM_INST}]
set_property     LOC     GTHE4_CHANNEL_X0Y2     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[3].*gen_gthe4_channel_inst[2].GTHE4_CHANNEL_PRIM_INST}]
set_property     LOC     GTHE4_CHANNEL_X0Y3     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[3].*gen_gthe4_channel_inst[3].GTHE4_CHANNEL_PRIM_INST}]
set_property     LOC     GTHE4_CHANNEL_X0Y8     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[4].*gen_gthe4_channel_inst[0].GTHE4_CHANNEL_PRIM_INST}]
set_property     LOC     GTHE4_CHANNEL_X0Y9     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[4].*gen_gthe4_channel_inst[1].GTHE4_CHANNEL_PRIM_INST}]
set_property     LOC     GTHE4_CHANNEL_X0Y10     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[4].*gen_gthe4_channel_inst[2].GTHE4_CHANNEL_PRIM_INST}]
set_property     LOC     GTHE4_CHANNEL_X0Y11     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[4].*gen_gthe4_channel_inst[3].GTHE4_CHANNEL_PRIM_INST}]

# Lane 7 was X0Y12, now X0Y14
set_property     LOC     GTHE4_CHANNEL_X0Y14     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[3].*gen_gthe4_channel_inst[0].GTHE4_CHANNEL_PRIM_INST}]
# Lane 6 was X0Y13, now X0Y13
set_property     LOC     GTHE4_CHANNEL_X0Y13     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[3].*gen_gthe4_channel_inst[1].GTHE4_CHANNEL_PRIM_INST}]
# Lane 5 was X0Y14, now X0Y15
set_property     LOC     GTHE4_CHANNEL_X0Y15     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[3].*gen_gthe4_channel_inst[2].GTHE4_CHANNEL_PRIM_INST}]
# Lane 4 was X0Y15, now X0Y12
set_property     LOC     GTHE4_CHANNEL_X0Y12     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[3].*gen_gthe4_channel_inst[3].GTHE4_CHANNEL_PRIM_INST}]
# Lane 3 was X0Y16, now X0Y18
set_property     LOC     GTHE4_CHANNEL_X0Y18     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[4].*gen_gthe4_channel_inst[0].GTHE4_CHANNEL_PRIM_INST}]
# Lane 2 was X0Y17, now X0Y19
set_property     LOC     GTHE4_CHANNEL_X0Y19     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[4].*gen_gthe4_channel_inst[1].GTHE4_CHANNEL_PRIM_INST}]
# Lane 1 was X0Y18, now X0Y16
set_property     LOC     GTHE4_CHANNEL_X0Y16     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[4].*gen_gthe4_channel_inst[2].GTHE4_CHANNEL_PRIM_INST}]
# Lane 0 was X0Y19, now X0Y17
set_property     LOC     GTHE4_CHANNEL_X0Y17     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[4].*gen_gthe4_channel_inst[3].GTHE4_CHANNEL_PRIM_INST}]

####CHECK THIS ONE!!!!!!!!!!!!!! THE CONTAINER [x]
# Lane 3 was X0Y16, now X0Y12
#set_property     LOC     GTHE4_CHANNEL_X0Y12     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[4].*gen_gthe4_channel_inst[0].GTHE4_CHANNEL_PRIM_INST}]
# Lane 2 was X0Y17, now X0Y15
#set_property     LOC     GTHE4_CHANNEL_X0Y15     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[4].*gen_gthe4_channel_inst[1].GTHE4_CHANNEL_PRIM_INST}]
# Lane 1 was X0Y18, now X0Y13
#set_property     LOC     GTHE4_CHANNEL_X0Y13     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[4].*gen_gthe4_channel_inst[2].GTHE4_CHANNEL_PRIM_INST}]
# Lane 0 was X0Y19, now X0Y14
#set_property     LOC     GTHE4_CHANNEL_X0Y14     [get_cells     -hierarchical     -filter     {NAME     =~
*gen_channel_container[4].*gen_gthe4_channel_inst[3].GTHE4_CHANNEL_PRIM_INST}]


##############################################################################################################
##
#
#
# BITFILE/BITSTREAM compress options
#
#set_property BITSTREAM.CONFIG.EXTMASTERCCLK_EN div-1 [current_design]
#set_property BITSTREAM.CONFIG.BPI_SYNC_MODE Type1 [current_design]
#set_property CONFIG_MODE BPI16 [current_design]
#set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
#set_property BITSTREAM.CONFIG.UNUSEDPIN Pulldown [current_design]
#
```