# Creating a Custom Application – Edge

Vitis AI Development Environment 2.0

## Abstract

This lab walks you through the steps to create a hardware design using the Vivado® Design Suite, build a Linux image using the PetaLinux tool, and use a trained Resnet50 model that will be optimized using the Vitis™ AI quantizer and will be compiled by the AI compiler. In the last step, you will run the application on a target ZCU104 board.

This lab should take approximately 120 minutes.

CloudShare users: You can perform all the steps in this lab except for the last two ("Preparing the SD Card" and "Running the Design on the Target Board") in the CloudShare environment. For the last two steps, you will need to perform them in your Linux environment locally. You will also need to have a ZCU104 board to perform these last two steps.

CloudShare user requirements (to perform the last two steps):

* Zynq® UltraScale+™ MPSoC ZCU104 evaluation kit
* Linux environment

## CloudShare Users Only

You are provided three attempts to access a lab, and the time allotted to complete each lab is 2X the time expected to complete the lab. Once the timer starts, you cannot pause the timer. Also, each lab attempt will reset the previous attempt—that is, your work from a previous attempt is not saved.

## Objectives

After completing this lab, you will be able to:

* Create a hardware design
* Build a Linux image using the PetaLinux tool
* Configure the rootfs
* Build a model (Resnet50)
* Run a design on a target board

## Introduction

Here is the complete Vitis AI environment development flow using the Vivado Design Suite to build a custom application.

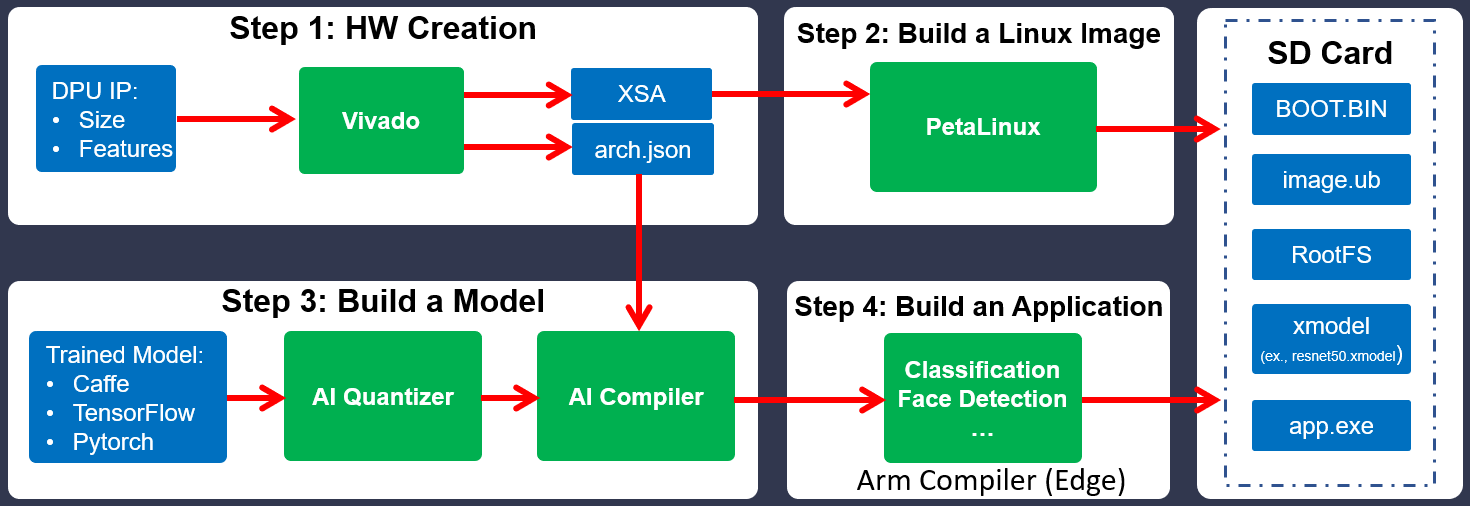


Figure 6‑: Custom Application Development Flow Using the Vivado Design Suite

In this lab, you will be focusing on "Hardware Creation" (Step 1: HW Platform Creation), "Building a Linux Image" (Step 2: Build a Linux OS), "Build a Model" (Step 3), and running the design on a board with the custom application.

You will be using the following PetaLinux tool commands in this lab:

petalinux-create: Creates objects that are part of a PetaLinux project.

Syntax: $petalinux-create -t <TYPE> --template <CPU\_TYPE> --name <PROJECT NAME> -s <PATH TO BSP>

petalinux-config: Allows you to use a standard menuconfig interface to control how the embedded Linux system is built.

Example for petalinux-config --get-hw-description:

Initialize a PetaLinux project within the project directory with an external HDF:

$petalinux-config --get-hw-description=<PATH-TO-HDF-DIRECTORY>

Initialize a PetaLinux project from within the directory containing an HDF:

$ petalinux-config --get-hw-description -p <PATH-TO-PETALINUX-PROJECT>

petalinux-build: Builds the entire embedded Linux system or a specified component of a Linux system.

Understanding the Lab Environment

Customizable environment variables enable you to tailor your environment for specific machine configurations. The only environment variable (shown below) used in the customer training environment (CustEd\_VM) points to the training directory where all the lab files are located.

This environment variable can be customized according to your specific location and can be set for Linux systems in the /etc/profile file.

The following is the environment variable used in the customer training VM:

| Environment Variable Name | Description |
| --- | --- |
| $TRAINING\_PATH | Points to the space allocated for students to work through their labs. This directory includes prebuilt images and starting points for the labs and demos. In the customer training VM, $TRAINING\_PATH sets to the /home/xilinx/training directory. |

Note: Environment variables are not supported from the Vitis IDE GUI. When using this tool, you must manually replace $TRAINING\_PATH with the value of the variable, which in the customer training virtual machine, is /home/xilinx/training.

## General Flow

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Step 1:  Creating  a Hardware Platform |  | Step 2:  Creating a PetaLinux Project |  | Step 3:  Configuring and Building the Image |  | Step 4:  Building  the  Model |  | Step 5:  Reviewing  the  Application |

|  |  |  |
| --- | --- | --- |
| Step 6:  Preparing  the  SD Card |  | Step 7:  Running the Design on  the Board |

Creating a Hardware Platform Using the Vivado Design Suite Step 1

You will build the hardware design with the DPU IP.

The default settings of the DPU include the following:

* B4096 with RAM\_USAGE\_LOW
* CHANNEL\_AUGMENTATION\_ENABLE
* DWCV\_ENABLE
* POOL\_AVG\_ENABLE
* RELU\_LEAKYRELU\_RELU6

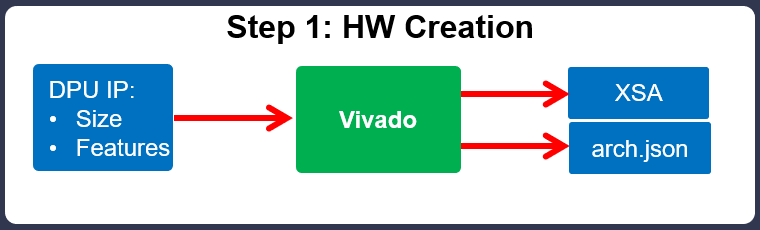


Figure 6‑: Step 1: Hardware Creation

1-1. Source the Vivado Design Suite and run a TCL script to build the block design.

1-1-1. Click the Terminal icon in the taskbar () to open a new terminal window.



Alternatively, press <Ctrl + Alt + T> to open a new terminal.

1-1-2. Enter the following command to source the Vivado Design Suite:

[host]$ source /opt/Xilinx/Vivado/2021.2/settings64.sh

Note: The customer training environment (CustEd\_VM) sets the Vivado Design Suite install path to /opt/Xilinx/Vivado. If the Vivado Design Suite is installed in a different location in your environment, use that install path.

1-1-3. Enter the following command to change the path to the project directory:

[host]$ cd $TRAINING\_PATH/custom\_app\_dev/lab/hw

1-1-4. Enter the following command to build the block design:

[host]$ vivado -source ../../support/custom\_zcu104\_hw.tcl

Note: It may take approximately 3-5 minutes to build the block design.

After the script completes, the generated Vivado IPI block design should be as shown below.

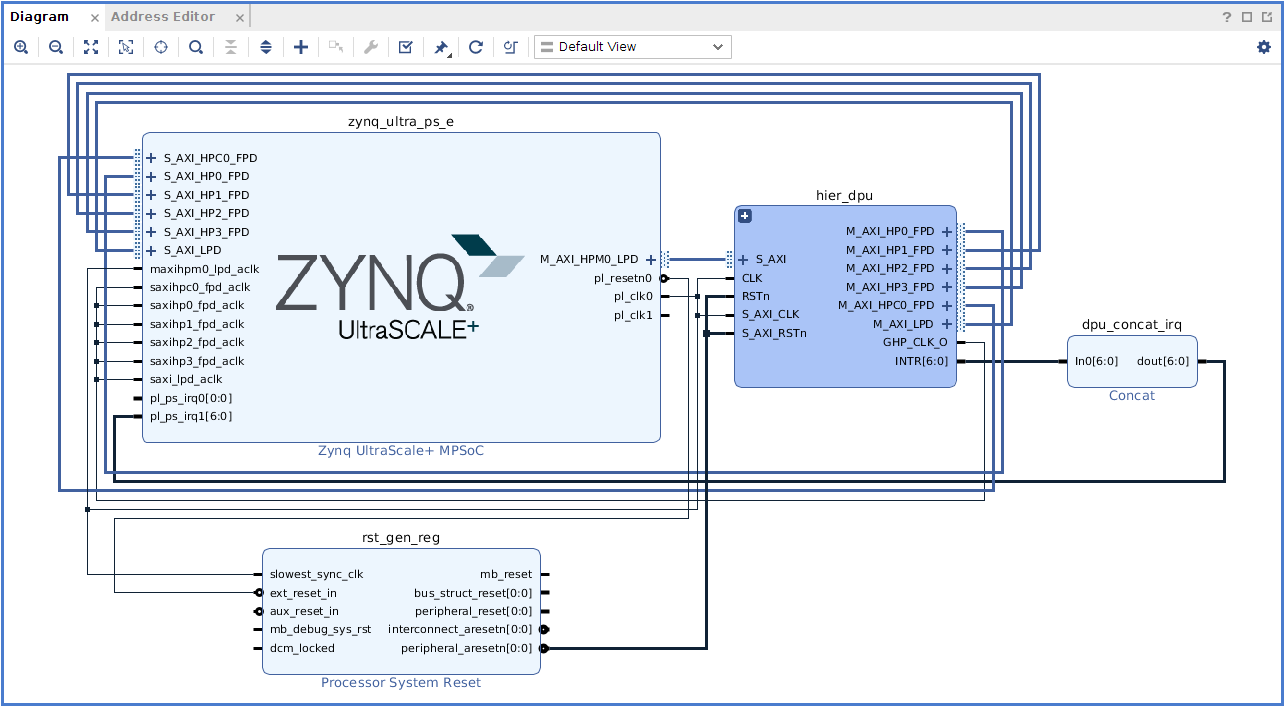


Figure 6‑: Custom Hardware - Block Design

1-2. Review the DPU IP re-customize block.

1-2-1. In the Vivado Design Suite, click the '+' in the hier\_dpu sub-block in the Diagram tab.

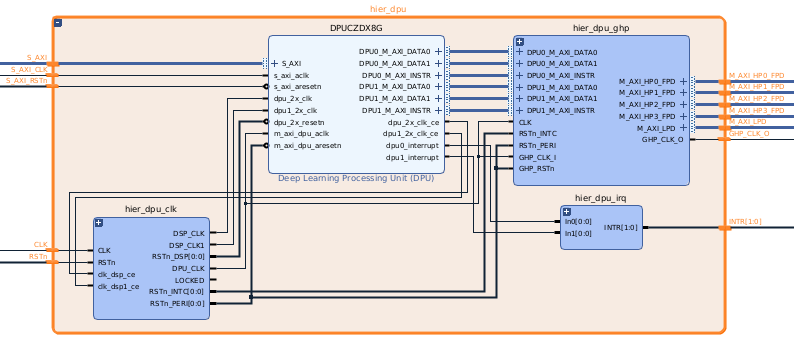


Figure 6‑: Hierarchical Block of the DPU IP

1-2-2. Observe all the blocks in hier\_dpu.

1-2-3. Double-click the Deep Learning Processing Unit (DPU) IP to open the Re-customize IP window.

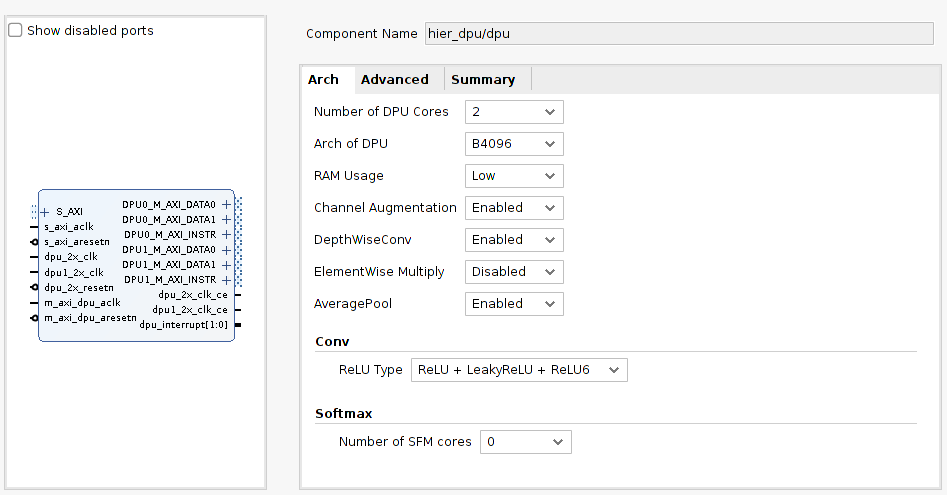


Figure 6‑: DPU IP Customization - Arch Tab

1-2-4. Review all the options.

You can see that the settings are based on the custom\_zcu104\_hw.tcl script.

1-2-5. Verify the drop-down options for all the fields.

Note: Do not change any settings.

1-2-6. Select the Advanced tab and review the settings.

1-2-7. Select the Summary tab to verify the selected options.

1-2-8. Click Cancel to keep the original settings.

1-2-9. Select the IP Sources tab in the Sources view.

1-2-10. Select top under Block Designs, right-click, and select Create HDL Wrapper.

1-2-11. Select Let Vivado manage wrapper and auto-update.

1-2-12. Click OK.

The DPU configuration is done and all the connections were made. Now it is time to generate the bitstream.

Note: Do not perform the following steps as generating the bitstream and exporting the hardware will take approximately 3-4 hours depending on your system configuration. The instructions below are provided for review on what the necessary actions are. After reviewing the instructions that follow, you can proceed to the "Creating a PetaLinux Project" step.

For CloudShare users, if you are going to perform these steps, you should do so in your local environment, not in CloudShare. This is because the memory available in CloudShare is 12 GB, and 16 GB is the minimum requirement. Generating the bitstream will still take approximately 3-4 hours.

1-3. Generate the bitstream.

1-3-1. Locate the Generate Bitstream entry under Program and Debug in the Flow Navigator.

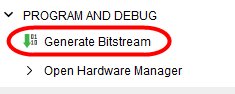


Figure 6‑: Generate Bitstream in the Flow Navigator

1-3-2. Click Generate Bitstream to start the bitstream generation.

Note: If a window appears with “No Implementation Results available”, click Yes. Then if any window appears with “Launch Runs”, click OK.

The status indicator in the upper-right corner of the workspace, as well as in the design runs console, will indicate when bitstream generation is complete.

After the bitstream generation is completed, export the hardware platform for software development (in this case, input to the PetaLinux tool) to build the Linux image.

Note that the JSON file is an important file that is needed by the Vitis AI compiler and will be generated at the time of exporting the hardware platform.

1-4. Export the hardware.

1-4-1. Select File > Export > Export Hardware.

1-4-2. Click Next.

1-4-3. Select Include bitstream and click Finish.

This will export the XSA file.

The JSON file is an important file that is needed by the Vitis AI compiler. This file was created during compilation by the Vivado Design Suite. It works together with the Vitis AI compiler to support model compilation under various DPU configurations. If you have generated the bitstream, you will find the arch.json file in the $TRAINING\_PATH/  
custom\_app\_dev/lab/hw/srcs/top/ip/top\_dpu\_0 directory.

Note: The prebuilt file is located at $TRAINING\_PATH/custom\_app\_dev/support/  
pre-built/hw.

1-5. Close the Vivado Design Suite.

1-5-1. Select File > Exit.

The Exit Vivado dialog box opens.

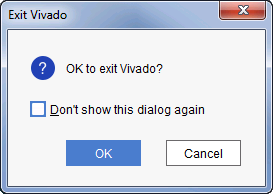


Figure 6‑: Exit Vivado Dialog Box

1-5-2. If you are asked to save the project or a portion of the project, select whichever elements of the project you want to save, then click Save to save the selected elements; otherwise, click Don't Save.

1-5-3. Click OK when you are asked to exit the Vivado Design Suite.

Note: You can choose to select the Don't show this dialog again option to avoid being asked for confirmation when exiting the Vivado Design Suite.

1-5-4. Close the terminal.

Creating a PetaLinux Project Step 2

You will now create a PetaLinux project and configure the project with the hardware platform that was generated from the Vivado Design Suite.

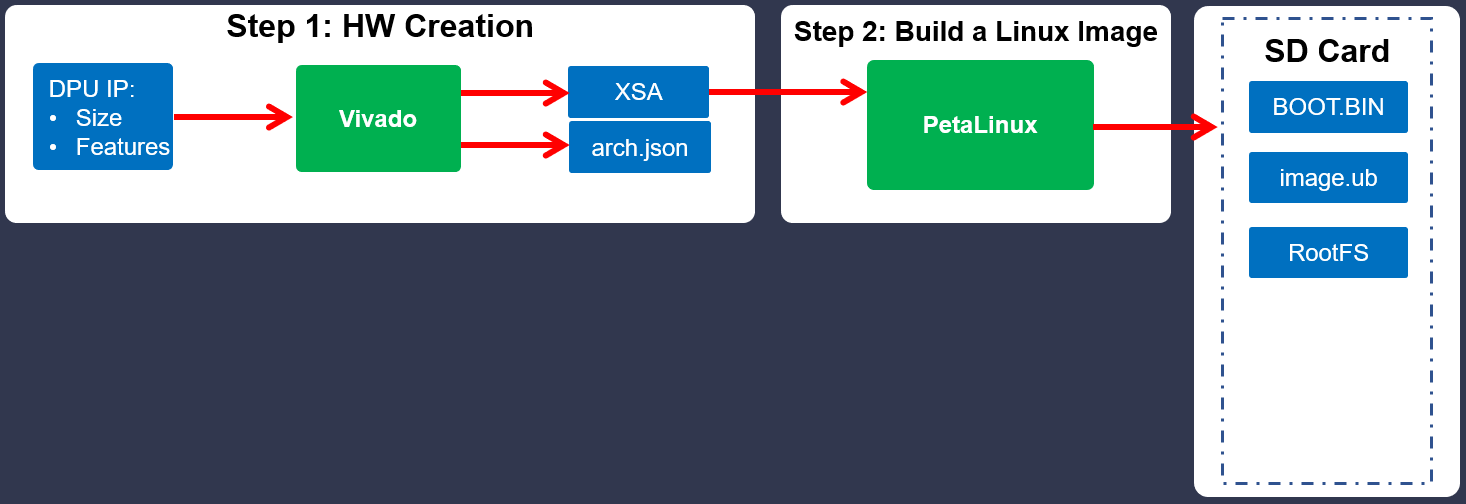


Figure 6‑: Step 2: Build a Linux Image

2-1. Set up the PetaLinux environment.

2-1-1. Click the Terminal icon in the taskbar () to open a new terminal window.



Alternatively, press <Ctrl + Alt + T> to open a new terminal.

2-1-2. Enter the following command to set up the PetaLinux environment:

[host] $ source /opt/Xilinx/PetaLinux/2021.2/tool/settings.sh

Note: The path provided here is valid for the customer training (CustEd\_VM) and the CloudShare environments. If you are running on another platform, you will need to replace /opt/Xilinx with the install path for your machine.

2-2. Use the petalinux-create command to create a new embedded Linux platform and choose the platform.

2-2-1. Enter the following command to change directory:

[host]$ cd $TRAINING\_PATH/custom\_app\_dev/lab/sw

2-2-2. Enter the following command to create a new PetaLinux project:

[host]$ petalinux-create -t project -s ../../support/  
custom-app-zcu104-2021.2.bsp

You should see the project directory at $TRAINING\_PATH/custom\_app\_dev/lab/sw: custom-app-zcu104-2021.2.

This BSP has been synced with the same hardware created in the previous step. However, for an understanding of the complete flow, the instructions for importing the hardware configuration with the custom hardware platform that was generated from the Vivado Design Suite has been provided below.

2-2-3. Enter the following command to change to the PetaLinux project directory:

[host]$ cd custom-app-zcu104-2021.2

Configuring the PetaLinux Project and Building the Image Step 3

This section explains the process of updating an existing or newly created PetaLinux project with a new hardware configuration. This enables you to make the PetaLinux tools software platform ready for building a Linux system, customized to your new hardware platform.

3-1. Import the hardware configuration with the custom hardware platform that was generated from the Vivado Design Suite.

3-1-1. Use the petalinux-config command to import the hardware configuration:

[Using the XSA file you built yourself]:

[host]$ petalinux-config --get-hw-description=$TRAINING\_PATH/  
custom\_app\_dev/lab/<<EXPORT\_PATH\_OF\_XSA>>

[Using the prebuilt XSA file]:

[host]$ petalinux-config --get-hw-description=$TRAINING\_PATH/  
custom\_app\_dev/support/pre-built/hw

After initialization, the tool displays the system-level menuconfig interface. This automatic launch of the system-level menuconfig interface occurs after the first time PetaLinux initializes a project. To return to this menuconfig later, execute petalinux-config from within the PetaLinux project directory.

3-1-2. Select Subsystem AUTO Hardware Settings > SD/SDIO Settings > Primary SD/SDIO (psu\_sd\_1).

Note: Press <Enter> to go into the sub-menu.

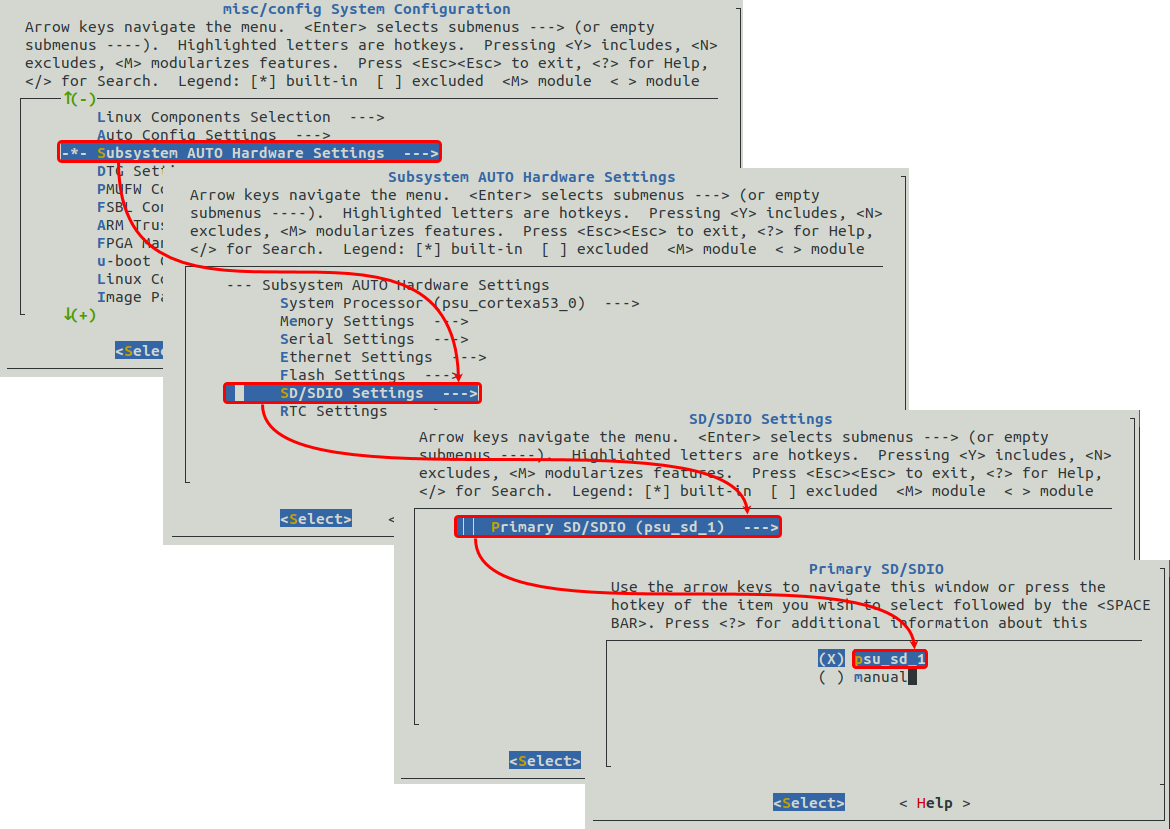


Figure 6‑: Selecting the Image Storage Media

3-1-3. Verify that the default image storage media is primary sd (i.e., psu\_sd\_1).

3-1-4. Exit 3 times to come to the main menu misc/config System Configuration.

3-1-5. Select Image Packaging Configuration > Root filesystem type (EXT4 (SD/eMMC/SATA/USB)).

3-1-6. Verify that EXT4 (SD/eMMC/SATA/USB) is enabled.

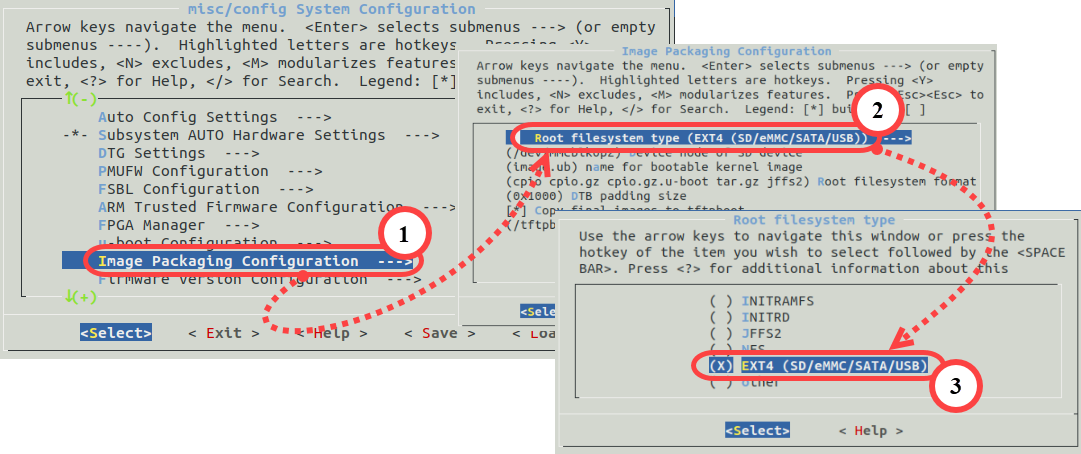


Figure 6‑: Reviewing the Image Packaging Configuration

3-1-7. Go to the main menu misc/config System Configuration.

3-1-8. Exit the configuration.

There is no need to save any changes.

3-2. Review the modules that are added.

3-2-1. Enter the following command in the terminal:

[host]$ petalinux-config -c rootfs

3-2-2. Go to the main menu Configuration > modules and observe that there is one module dpu added.

3-2-3. Exit the rootfs configuration without saving changes.

3-3. Review the device tree file.

3-3-1. Enter the following command to review the device tree file:

[host]$ gedit project-spec/meta-user/recipes-bsp/device-tree/  
files/system-user.dtsi

Note that the required changes are already done based on the DPU configuration made in the design from the Vivado Design Suite.

/include/ "system-conf.dtsi"

/ {

dpu: dpu@80000000 {

compatible = "deephi, dpu";

reg = <0x0 0x80000000 0x0 0x700>;

interrupts = <0x0 106 0x1 0x0 107 0x1>;

interrupt-parent = <&gic>;

core-num = <0x2>;

};

dpcma: dpcma {

compatible = "deephi,cma";

};

};

Note that core-num is 2. This is because two cores were configured in the Vivado Design Suite project.

3-3-2. Close the editor.

Note: Do not perform the following step as it will take approximately 7-8 hours to generate the Linux image based on your system configuration. In a native Linux environment, building the image takes approximately an hour.

The instructions below are provided for review on what the necessary actions are. After reviewing the instructions that follow, you can proceed to the "Building the Model" step.

The prebuilt files with which you can continue the lab are provided in the $TRAINING\_PATH/custom\_app\_dev/support/pre-built/sd\_card directory.

3-4. Build the Linux image.

3-4-1. Enter the following command to build the Linux image:

[host]$ petalinux-build

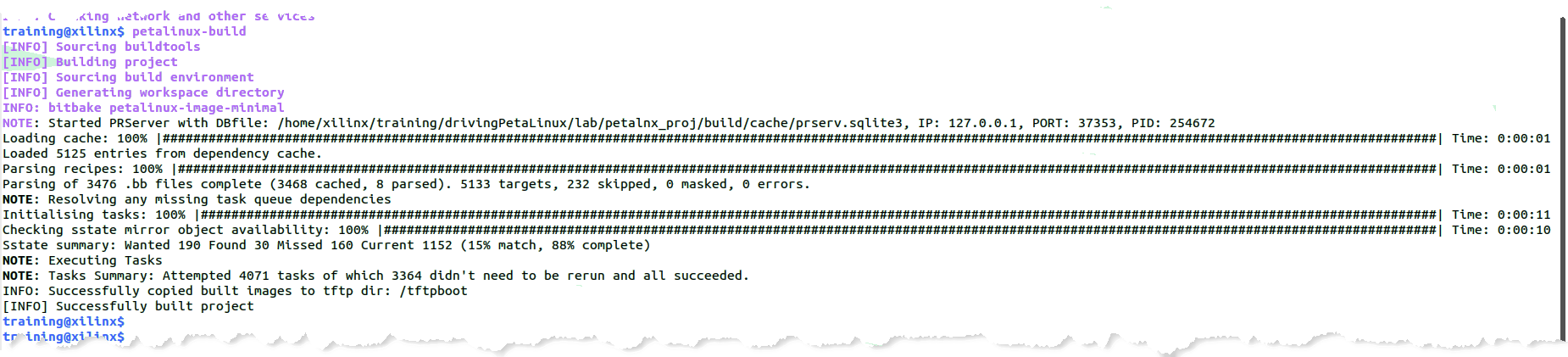


Figure 6‑: Building the Linux Image

Note: It will take approximately 7-8 hours to build the image.

This step will generate a device tree DTB file, a First Stage Boot Loader (if selected), U-Boot (if selected), the Linux kernel, and a rootfile system image. Finally, it will generate boot images.

The petalinux-build tool always generates a FIT image as the kernel image.

<project-root>/images is also automatically generated. Files in this directory will get updated when you run petalinux-build.

During this time, the following will occur:

* Cross-compiling and linking of the Linux kernel (linux-x.x/\*)
* Cross-compiling and linking of the default user libs and applications (lib/\* and user/\*)
* Building of a local copy of the Arm® Cortex™-A53 processor Linux root file system (rootfs/\*)
* Assembling of the kernel and root file system into a single downloadable binary image file (images/\*)
* Copying of the image files from images/ to /tftpboot

The build log is saved in the $custom\_app\_dev/lab/custom\_app\_dev/build/build.log file.

3-4-2. Look at the contents in the images/linux subdirectory once compilation completes by executing the following commands from the project directory:

[host]$ cd images/linux

[host]$ ls -ll

3-5. Create a boot image (BOOT.BIN) including the FSBL, ATF, bitstream, and U-Boot.

3-5-1. Enter the following command to create a boot image:

[host]$ petalinux-package --boot --fsbl zynqmp\_fsbl.elf --u-boot u-boot.elf --pmufw pmufw.elf --fpga system.bit

This will create a BOOT.BIN file.

3-5-2. [Prebuilt users]: You can find the same content in the pre-built directory. Enter the following command to see the files located in the pre-built directory:

[host]$ cd $TRAINING\_PATH/custom\_app\_dev/support/pre-built/sd\_card

[host]$ ls -ll

Review the files located in this directory.

* BOOT.BIN: This is the boot image, which includes:
* zynqmp\_fsbl.elf: First Stage Boot Loader (FSBL)
* u-boot.elf: Second Stage Boot Loader (U-Boot)
* pmufw.elf: PMU firmware
* system.bit: Bitstream of the custom hardware platform
* image.ub: Linux kernel image
* rootfs.tar.gz: Root file system

3-5-3. Close the terminal.

Building the Model Step 4

The pre-trained Resnet50 model from Caffe will be quantized using the Vitis AI quantizer. Then the output from the quantizer will be fed to the AI compiler, which generates the DPU xmodel.

Note: You will be using the completed quantized model from the "AI Quantizer and AI Compiler" lab.

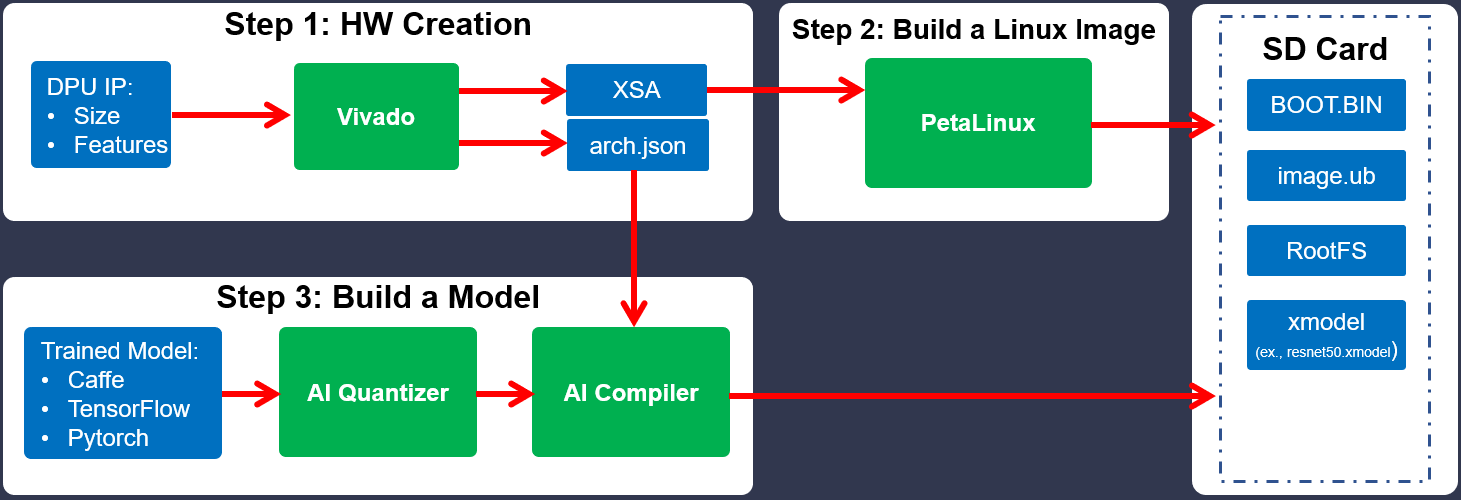


Figure 6‑: Step 3: Build a Model

4-1. Copy the arch.json file generated from the Vivado Design Suite, the JSON file required for the AI compiler, and the AI compiler script.

4-1-1. Press <Ctrl + Alt + T> to open a new terminal window.

4-1-2. Enter the following commands to create a new folder named custom\_app\_dev:

[host]$ cd /home/xilinx/Vitis-AI

[host]$ mkdir custom\_app\_dev

[host]$ cd custom\_app\_dev

4-1-3. Enter the following commands to copy the JSON, quantized model and AI compiler script files:

[host]$ cp $TRAINING\_PATH/custom\_app\_dev/support/pre-built/hw/  
arch.json .

[host]$ cp $TRAINING\_PATH/custom\_app\_dev/support/xmodel/  
2\_caffe\_compile\_for\_edge.sh .

[host]$ cp -r $TRAINING\_PATH/custom\_app\_dev/support/  
quantized\_model .

4-2. Review the 2\_caffe\_compile\_for\_edge.sh script file.

4-2-1. Enter the following command to review the script:

[host]$ gedit $TRAINING\_PATH/custom\_app\_dev/lab/model/  
2\_caffe\_compile\_for\_edge.sh

Observe the arch.json file path and how it points to the vai caffe compiler argument.

4-2-2. Close the editor.

4-3. Load the CPU image from the Docker hub.

4-3-1. Press <Ctrl + Alt + T> to open a new terminal window.

4-3-2. Enter the following commands to load the CPU image from the Docker hub:

[host]$ cd Vitis-AI

[host]$ ./docker\_run.sh xilinx/vitis-ai-cpu:latest

If there is a newer version of the Vitis AI tools that has been released, entering the above command will download the latest version (that is, not VAI 2.0). In order to use the Vitis AI 2.0 environment, enter the command with the docker image tag. You can find the tag from https://hub.docker.com/r/xilinx/vitis-ai-cpu/tags?page=1&ordering=last\_updated.

For example: [host]$ ./docker\_run.sh xilinx/vitis-ai-cpu:<Docker\_tag>

To use 2.0: [host]$ ./docker\_run.sh xilinx/vitis-ai-cpu:2.0

Note: Keep clicking to accept the terms and agreements and then enter 'y'.

The terminal output should be similar to the following:

==========================================

\_\_ \_\_\_ \_ \_ \_\_\_\_\_

\ \ / (\_) | (\_) /\ |\_ \_|

\ \ / / \_| |\_ \_ \_\_\_ \_\_\_\_\_\_ / \ | |

\ \/ / | | \_\_| / \_\_|\_\_\_\_\_\_/ /\ \ | |

\ / | | |\_| \\_\_ \ / \_\_\_\_ \ \_| |\_

\/ |\_|\\_\_|\_|\_\_\_/ /\_/ \\_\\_\_\_\_\_|

==========================================

Docker Image Version: 2.0.0.1103 (CPU)

Vitis AI Git Hash: 06d7cbb

Build Date: 2022-01-12

For TensorFlow 1.15 Workflows do:

conda activate vitis-ai-tensorflow

For Caffe Workflows do:

conda activate vitis-ai-caffe

For PyTorch Workflows do:

conda activate vitis-ai-pytorch

For TensorFlow 2.6 Workflows do:

conda activate vitis-ai-tensorflow2

Vitis-AI /workspace >

4-4. Activate the Caffe conda environment.

4-4-1. Enter the following command to activate the Caffe conda environment:

Vitis-AI /workspace > conda activate vitis-ai-caffe

The terminal output should be similar to the following:

(vitis-ai-caffe) Vitis-AI /workspace >

In this lab, you will be using the cf\_resnet50\_imagenet\_224\_224\_7.7G pre-trained model from the AI model zoo. The quantized model has been provided and can be compiled using the AI compiler directly.

You will find the quantized model in the $TRAINING\_PATH/custom\_app\_dev/support/quantized\_model directory.

4-5. Run the AI compiler tool by executing the 2\_caffe\_compile\_for\_edge.sh script.

4-5-1. Enter the following command to change the path:

(vitis-ai-caffe) Vitis-AI /workspace > cd custom\_app\_dev

4-5-2. Enter the following command to run the Vitis AI compiler for Caffe:

(vitis-ai-caffe) Vitis-AI /workspace/custom\_app\_dev > sh 2\_caffe\_compile\_for\_edge.sh

Note: This will take approximately 1-2 minutes to complete.

When a network model is compiled, the required options should be specified to the Vitis AI compiler. Once compilation is successful, the Vitis AI compiler will generate the xmodel file for deployment. This file is located under the folder specified by output\_dir.

You will see the following output after the compilation has completed.

(vitis-ai-caffe) Vitis-AI /workspace/custom\_app\_dev > sh 2\_caffe\_compile\_for\_edge.sh

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* VITIS\_AI Compilation - Xilinx Inc.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[INFO] Namespace(batchsize=1, inputs\_shape=None, layout='NCHW', model\_files=['quantized\_model/deploy.caffemodel'], model\_type='caffe', named\_inputs\_shape=None, out\_filename='/tmp/resnet50\_org.xmodel', proto='quantized\_model/deploy.prototxt')

[INFO] caffe model: /workspace/custom\_app\_dev/quantized\_model/deploy.caffemodel

[INFO] caffe model: /workspace/custom\_app\_dev/quantized\_model/deploy.prototxt

[INFO] parse raw model :100%|█| 194/194 [00:23<00:00, 8.14it/s]

[INFO] infer shape (NCHW) :100%|█| 194/194 [00:00<00:00, 452.12it/s]

[INFO] infer shape (NHWC) :100%|█| 194/194 [00:00<00:00, 1150.67it/s]

[INFO] perform level-1 opt :100%|█| 3/3 [00:00<00:00, 90.38it/s]

[INFO] infer shape (NHWC) :100%|█| 196/196 [00:00<00:00, 3310.31it/s]

[INFO] generate xmodel :100%|█| 196/196 [00:00<00:00, 396.83it/s]

[INFO] dump xmodel: /tmp/resnet50\_org.xmodel

[UNILOG][INFO] Compile mode: dpu

[UNILOG][INFO] Debug mode: function

[UNILOG][INFO] Target architecture: DPUCZDX8G\_ISA0\_B4096\_MAX\_BG2

[UNILOG][INFO] Graph name: deploy, with op num: 412

[UNILOG][INFO] Begin to compile...

[UNILOG][INFO] Total device subgraph number 3, DPU subgraph number 1

[UNILOG][INFO] Compile done.

[UNILOG][INFO] The meta json is saved to "/workspace/custom\_app\_dev/quantized\_model/vai\_c\_output\_custom\_app\_ZCU104/meta.json"

[UNILOG][INFO] The compiled xmodel is saved to "/workspace/custom\_app\_dev/quantized\_model/vai\_c\_output\_custom\_app\_ZCU104//resnet50.xmodel"

[UNILOG][INFO] The compiled xmodel's md5sum is 97155d13e8ee9aa8a5b4bca3d71304ce, and has been saved to "/workspace/custom\_app\_dev/quantized\_model/vai\_c\_output\_custom\_app\_ZCU104/md5sum.txt"

(vitis-ai-caffe) Vitis-AI /workspace/custom\_app\_dev >

You can also find this information under the folder specified by output\_dir. In this case, this is the /workspace/custom\_app\_dev/quantized\_model/vai\_c\_output\_custom\_app\_ZCU104 directory.

The generated files are:

* Compiled xmodel: resnet50.xmodel
* For run time: meta.json

4-5-3. Enter the following command to exit from the Vitis AI environment:

(vitis-ai-caffe) Vitis-AI /workspace/custom\_app\_dev > exit

4-6. Copy the generated xmodel file to the support directory.

4-6-1. Enter the following command to the copy the generated ELF file:

[host]$ cp /home/xilinx/Vitis-AI/custom\_app\_dev/quantized\_model/  
vai\_c\_output\_custom\_app\_ZCU104/resnet50.xmodel $TRAINING\_PATH/custom\_app\_dev/support

Reviewing the Application Step 5

You have built the hardware, Linux images, and the ResNet50 model. Now, you need to build the application. In this step, you will review the sample application and then in the next step you will use the executable that is already available to run the design on a board.

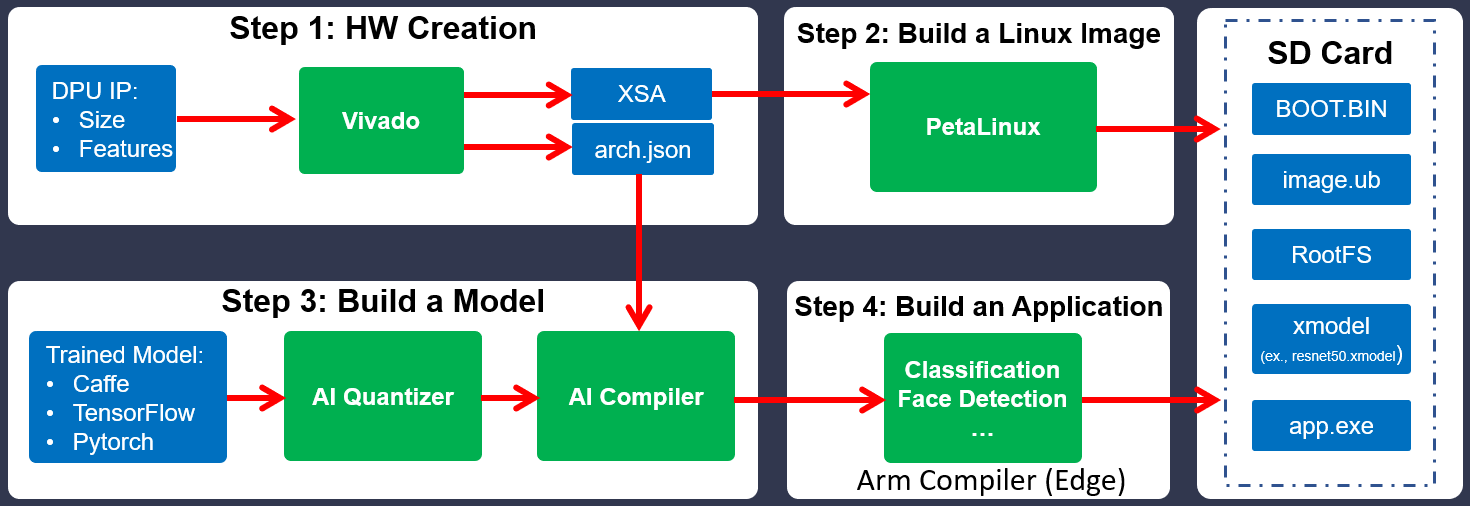


Figure 6‑: Step 4: Application Development

5-1. Review the sample code (resnet50.cpp) provided in the app directory.

5-1-1. Enter the following command to change the path to the sample directory:

[host]$ cd $TRAINING\_PATH/custom\_app\_dev/lab/app

5-1-2. Enter the following command to open and review the sample file:

[host]$ gedit resnet50.cpp

The figure below shows the main function near line no. 95.

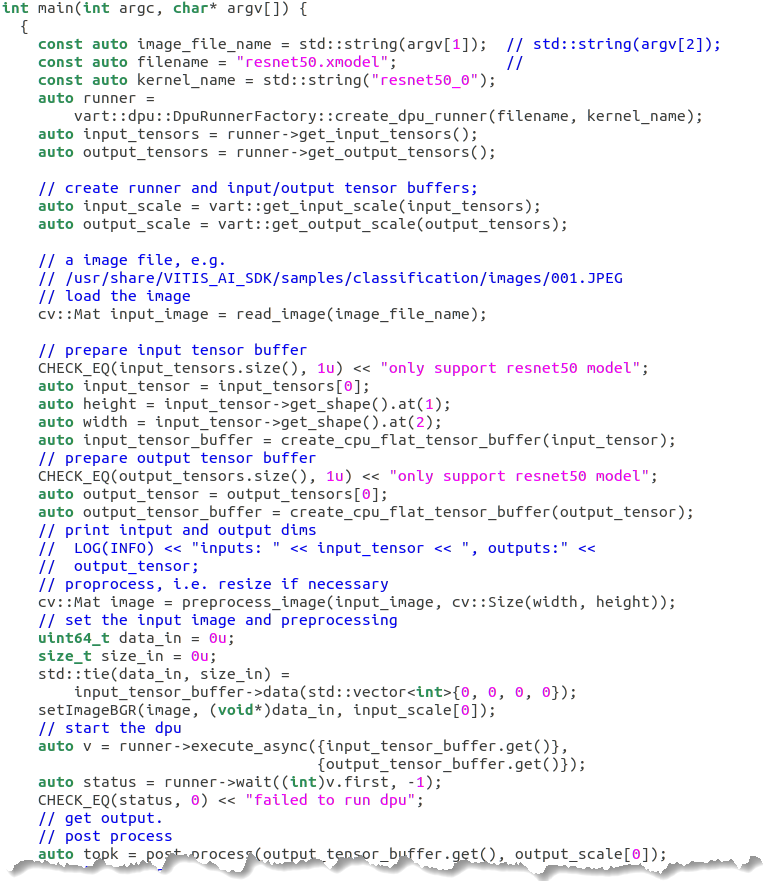


Figure 6‑: Main Application (resnet50.cpp)

* argv[1]: Receives the input image.
* filename: Points to the xmodel generated in the previous step (named as resnet50.xmodel).
* kernel\_name: Name of the kernel.
* create\_dpu\_runner(): Creates a task, and after DPU processing is complete, the post-processing function can be invoked.
* get\_input\_tensors(): Queries DpuRunner for the tensor format it expects. Returns DpuRunner::TensorFormat::NCHW or returns DpuRunner::TensorFormat::NHWC.
* get\_output\_tensors(): Queries DpuRunner for the shape and name of the output tensors it expects. Returns all output tensors.
* execute\_async(): Function to execute the runner.
* wait(): Function to wait for the end of DPU processing.

You can also review the pre-processing and post-processing functions.

5-1-3. After completing the review of the sample code, close the editor.

Preparing the SD Card Step 6

CloudShare users: You can perform all the steps in this lab except for the last two ("Preparing the SD Card" and "Running the Design on the Target Board") in the CloudShare environment. For the last two steps, you will need to perform them in your Linux environment. You will also need to have a ZCU104 board to perform these last two steps.

CloudShare user requirements (to perform the last two steps):

* Zynq UltraScale+ MPSoC ZCU104 evaluation kit
* Linux environment

6-1. Configure the SD card with an ext file system boot.

6-1-1. Format the SD card with two partitions using a partition tool such as gparted.

First partition:

* Size: Should be at least 500 MB in size
* File system: Format as the FAT32 file system
* Label: Label this partition as BOOT
* Files to be copied to this partition from $TRAINING\_PATH/custom\_app\_dev/support/pre-built/sd\_card:
* Copy the BOOT.BIN, image.ub, and boot.scr files (not rootfs.tar.gz):
* BOOT.BIN
* image.ub
* boot.scr
* Copy the generated resnet50.xmodel file from $TRAINING\_PATH/custom\_app\_dev/support:
* resnet50.xmodel
* Copy the application resnet50 file from $TRAINING\_PATH/custom\_app\_dev/support/app:
* resnet50
* Copy the image folder img from $TRAINING\_PATH/custom\_app\_dev/support:
* img

Second partition:

* Size: Remaining size of the SD card
* File system: Format as the ext4 file system
* Label: Label this partition as RootFS
* Files to be copied to this partition:
* Extract the rootfs.tar.gz and copy to the RootFS partition:
* rootfs.tar.gz file located in $TRAINING\_PATH/custom\_app\_dev/support/pre-built/sd\_card

To extract the rootfs.tar.gz file in the RootFS partition:

Step 1: [host]$ sudo tar -xvf rootfs.tar.gz -C <ROOTFS\_PATH>

For example:

[host]$ sudo tar -xvf rootfs.tar.gz -C /media/xilinx/RootFS

Step 2: [host]$ sync && sync

Running the Design on the Target Board Step 7

After the files are copied to the BOOT and RootFS partitions on the SD card, now insert the SD card into the SD card slot of the ZCU104 board.

7-1. Bring up the ZCU104 board.

7-1-1. Ensure that the board is powered off and that the board is connected to the host with both USB and Ethernet cables.

Note: This lab will have you boot using SD Card (SD1) mode, which requires the mode pins to be set as described below.

7-1-2. Locate SW6 on the board.

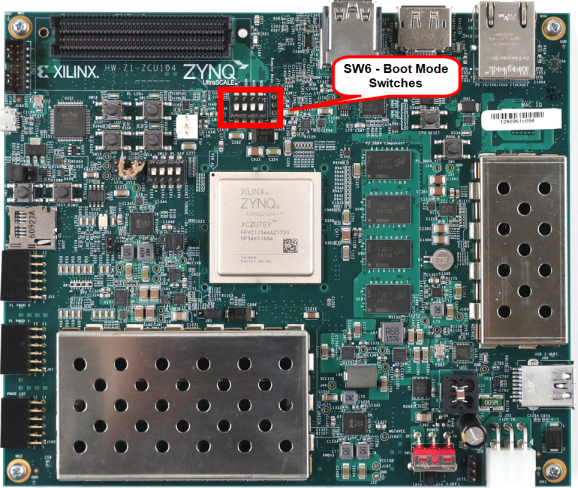


Figure 6‑: ZCU104 SW6 - Boot Mode Switches

7-1-3. Set SW6 as shown below to ensure that the board is configured to boot from SD Card (SD1).

Note: Settings are shown to illustrate different boot mode settings. Make sure you select SD Card (SD1).

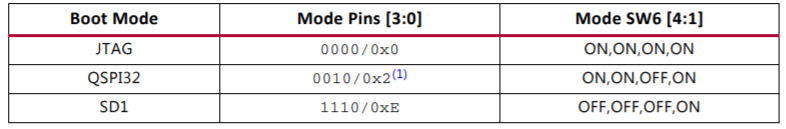


Figure 6‑: Board Configuration

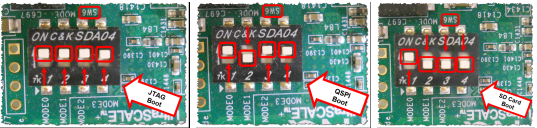


Figure 6‑: ZCU104 Boot Settings

7-1-4. Slide the power switch to the "ON" position to power on the board.

7-2. Open the Tera Term terminal program.

7-2-1. From the Windows desktop, double-click the Tera Term icon to launch Tera Term.

Alternatively, you select Start > All Programs > Tera Term > Tera Term.

7-2-2. Select File > New Connection.

7-2-3. Select Serial as the connection (1).

7-2-4. Click the Port drop-down list to view the available COM ports (2).

Note: If your port is not listed, exit Tera Term, power cycle your board and restart this step.

7-2-5. Select the COM # discovered previously (3).

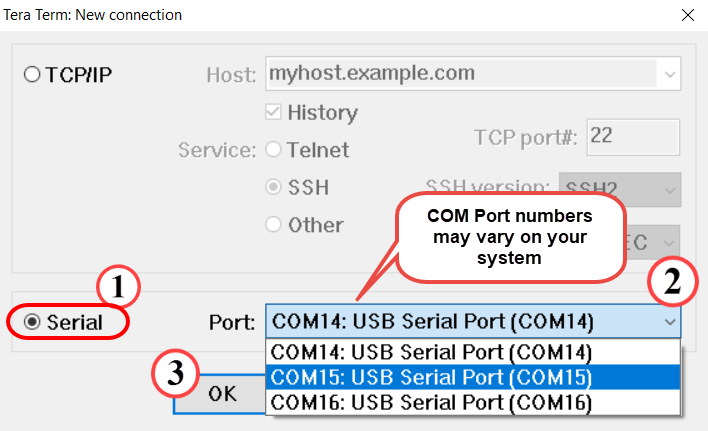


Figure 6‑: Selecting the COM Port

Note: The COM port setting is specific to the computer being used and may need to be different than shown. Use the COM port # that was discovered previously.

7-2-6. Click OK.

The terminal console window opens.

7-2-7. Select Setup > Serial Port.

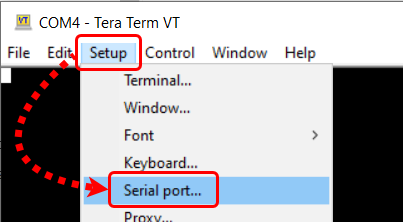


Figure 6‑: Opening the Tera Term Serial Port Setup Window

The Tera Term Serial Port Setup dialog box opens.

7-2-8. Confirm that the proper serial port has been selected (1).

7-2-9. Set the baud rate to 115200 (2).

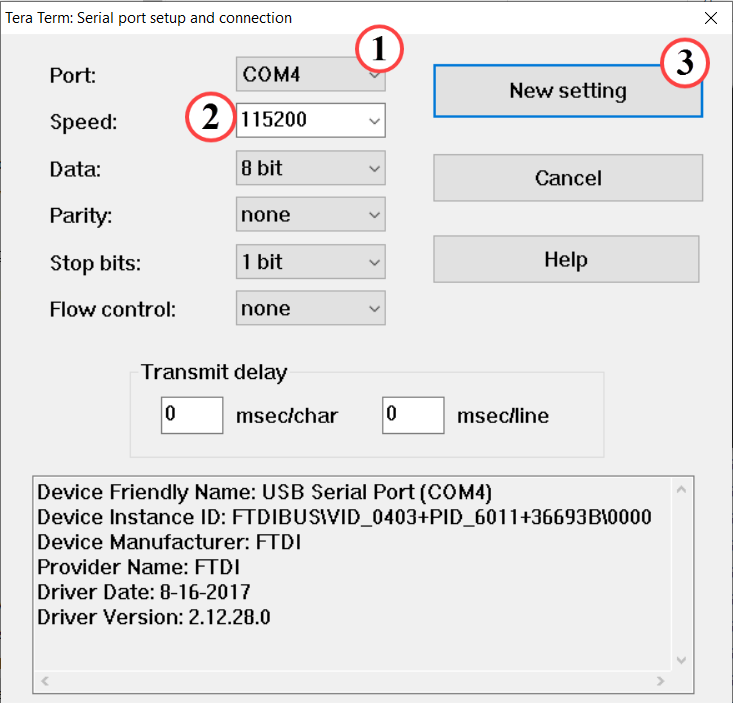


Figure 6‑: Setting the Parameters for the Serial Port

Note: The COM port setting is specific to the computer being used and may need to be different than shown. Use the COM port # that was discovered previously.

7-2-10. Click OK (3).

Tera Term is now configured to receive and transmit serial information to/from the evaluation board.

7-3. Run the design on the target board.

7-3-1. Observe the booting messages in the Tera Term terminal.

You should see the Linux image boot and the terminal as below after a few seconds:

...

Starting Xserver

Starting Dropbear SSH server:

Generating 2048 bit rsa key, this may take a while...

...

Fingerprint: sha1!! b9:ac:da:27:4c:e8:87:d0:f1:2f:ee:02:2a:fd:b4:f6:e1:08:72:48

dropbear.

Starting internet superserver: inetd.

Starting syslogd/klogd: done

Starting tcf-agent: OK

mount: /mnt: /dev/mmcblk0p1 already mounted on /run/media/mmcblk0p1.

attempting to run /mnt/init.sh

/mnt

root@custom-app-zcu104-2021:~#

7-3-2. Enter the following commands one by one to run the application:

% cd ~

% cp /mnt/sd-mmcblk0p1/resnet50.xmodel .

% cp /mnt/sd-mmcblk0p1/resnet50 .

% cp -r /mnt/sd-mmcblk0p1/img .

% ./resnet50 img/bellpeppe-994958.JPEG

You should see the output as shown below (with the XRT messages):

Expect:

score[945] = 0.992235 text: bell pepper,

score[941] = 0.00315807 text: acorn squash,

score[943] = 0.00191546 text: cucumber, cuke,

score[939] = 0.000904801 text: zucchini, courgette,

score[949] = 0.00054879 text: strawberry,

Note: The scores show that the model has detected the image as the bell pepper image with ~99% accuracy.

7-3-3. Verify and compare the results above with the bellpeppe-994958.JPEG image from the $custom\_app\_dev/support/img directory.

You can also experiment with the other images available in the img folder.

% ./resnet50 img/<<OTHER\_IMAGES>>.JPEG

7-3-4. After finishing running the design, power off the board and exit Tera Term.

## Summary

In this lab, you have learned how to build a hardware design using the Vivado Design Suite, build a Linux image using the PetaLinux tool, build a model, run the design on a target board, and verify the results.