# Creating a DPU Kernel Using the Vitis Environment Flow – Edge

Vitis AI Development Environment 2.0

## Abstract

This lab walks you through the steps to build a Vitis™ unified software platform project that adds the DPU (DPUCZDX8G) as the kernel (hardware accelerator). Running the design on a ZCU104 board is also illustrated in the lab.

This lab should take approximately 90 minutes.

## 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:

* Describe the Vitis command line flow options for building a custom hardware platform
* Change the configuration of the DPUCZDX8G
* Integrate the DPU into a custom platform
* Run the design on a target board

## Introduction

The figure below shows the Vitis AI development environment flow using Vitis acceleration to build hardware with a custom platform.

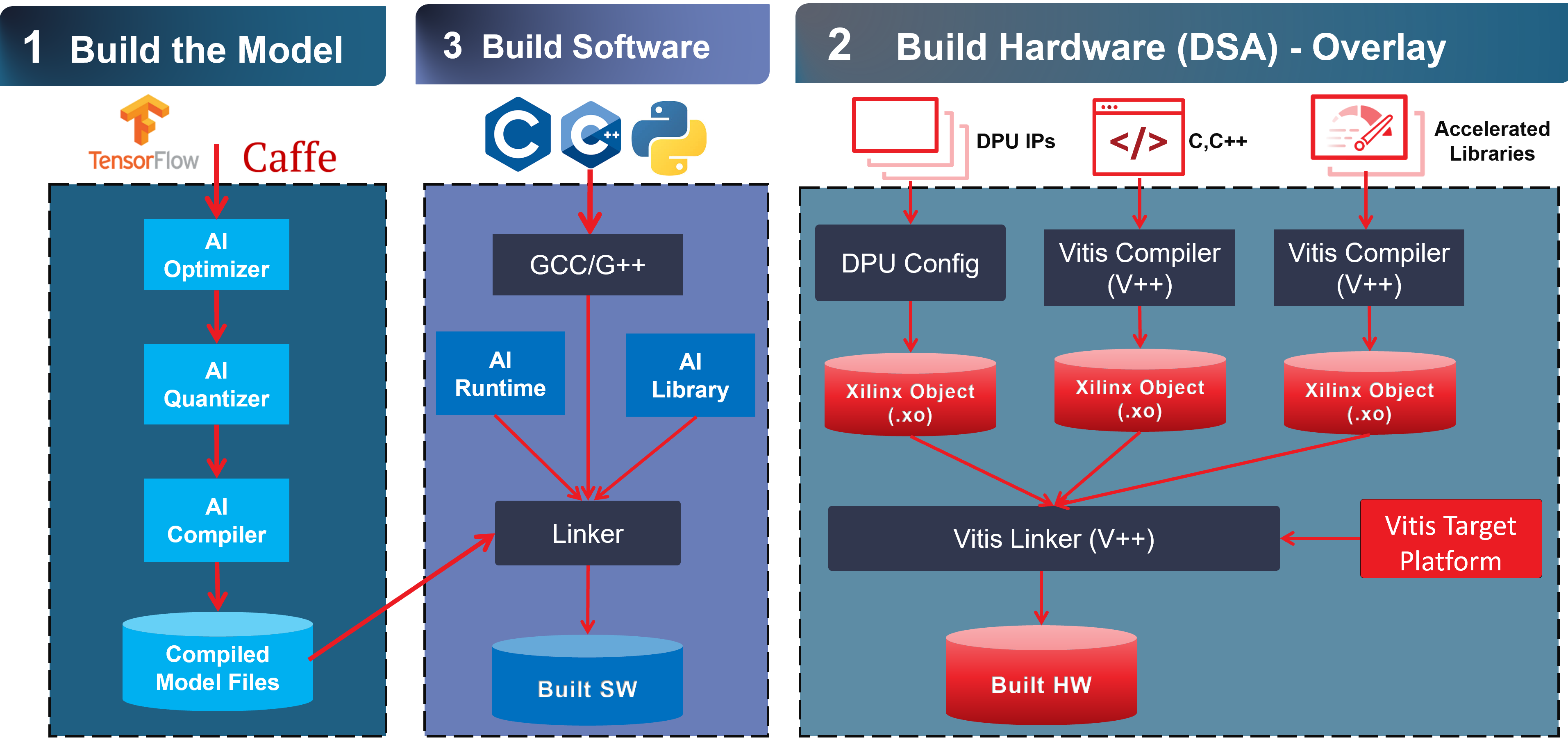


Figure 5‑: Vitis AI Development Environment Flow

You will be focusing in this lab on building the hardware in the Vitis acceleration development flow (that is, the "2 Build Hardware (DSA) - Overlay" part of the flow).

In the Vitis acceleration development flow, the hardware accelerator can be built using:

* RTL IPs
* C/C++ sources
* Vitis libraries

You will be adding the DPUCZDX8G IP (RTL) as the kernel (hardware accelerator).

The Xilinx Deep Learning Processor Unit (DPU) is a configurable computation engine dedicated for convolutional neural networks. The degree of parallelism utilized in the engine is a design parameter and application. It includes a set of highly optimized instructions and supports most convolutional neural networks, such as VGG, ResNet, GoogLeNet, YOLO, SSD, MobileNet, FPN, and others.

Features:

* One AXI slave interface for accessing configuration and status registers
* One AXI master interface for accessing instructions
* Support for a configurable AXI master interface with 64 or 128 bits for accessing data depending on the target device
* Support for individual configuration of each channel
* Support for optional interrupt request generation
* Some highlights of DPU functionality include:
* Configurable hardware architecture: B512, B800, B1024, B1152, B1600, B2304, B3136, and B4096
* Maximum of three cores
* Convolution and deconvolution
* Depthwise convolution
* Max poolling
* Average poolling
* ReLU, RELU6, and Leaky ReLU
* Concat
* Elementwise-sum
* Dilation
* Reorg
* Fully connected layer
* Softmax
* Bach normalization
* Split

Hardware Architecture

After start up, the DPU fetches instructions from off-chip memory to control the operation of the computing engine. The instructions are generated by the Vitis AI compiler where substantial optimizations have been performed.

On-chip memory is used to buffer input, intermediate, and output data to achieve high throughput and efficiency. The data is reused as much as possible to reduce memory bandwidth.

A deep pipelined design is used for the computing engine. The processing engines (PEs) take full advantage of the fine-grained building blocks, such as multipliers, adders, and accumulators in Xilinx devices.

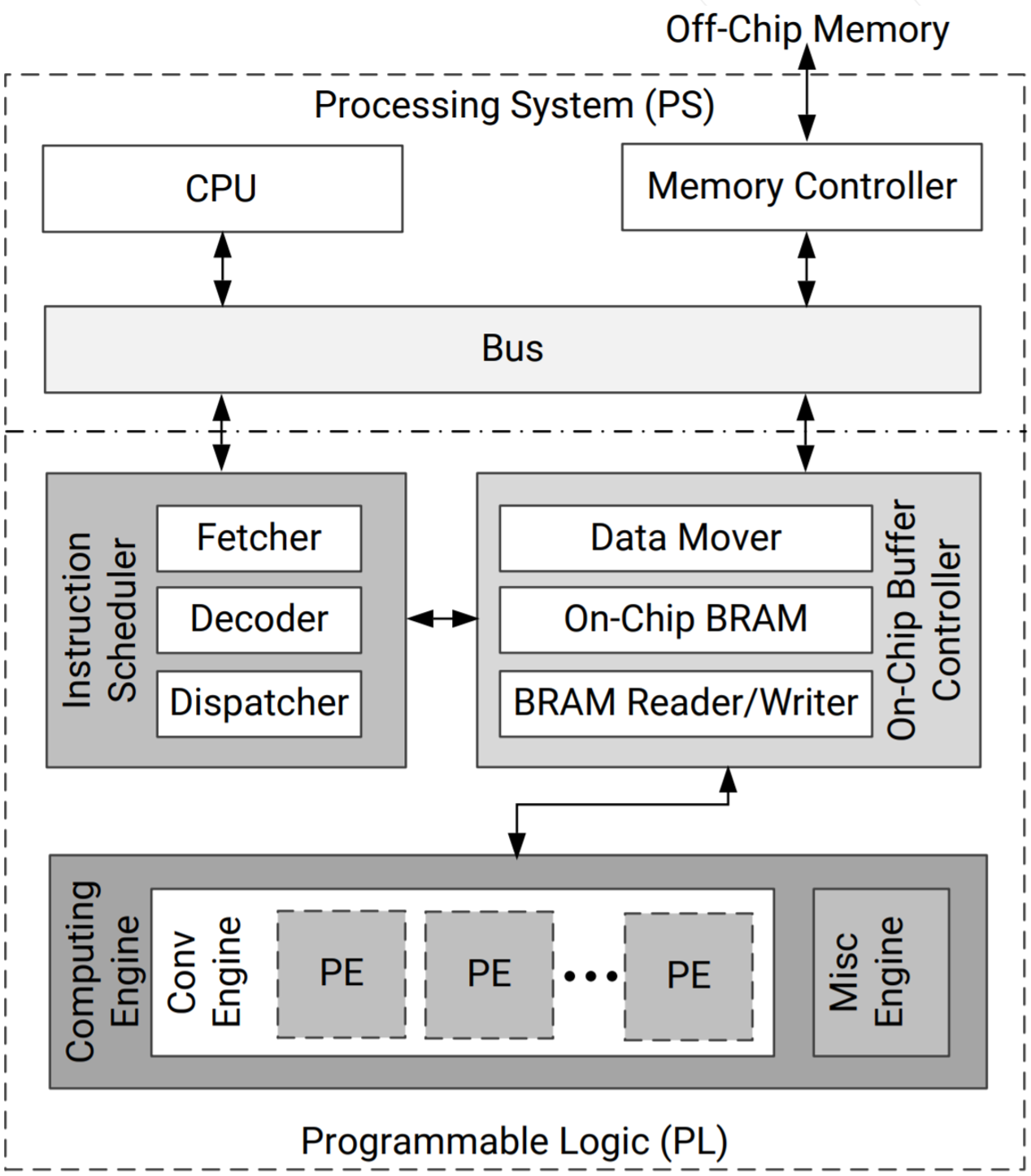


Figure 5‑: Hardware Architecture of DPUCZDX8G

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:  Reviewing  Config. Files & Makefile |  | Step 2:  Building  the  Project |  | Step 3:  Reviewing  the Output Files |  | Step 4:  Preparing  the  SD Card |  | Step 5:  Running the Design on the Board |

Reviewing the Configuration Files and Makefile Step

You will first review the configuration files for the Vitis compiler (prj\_config\_104\_2dpu) and DPU parameters (dpu\_conf.vh).

You will then review the makefile, which performs the following:

* Sets the Vivado® Design Suite and cross-compiler tool path
* Sets the TARGET build configuration, system run (hw)
* Sets the SDX\_PLATFORM path (such as a custom platform path)
* Sets the Vitis compiler options
* Provides the path for the DPU HDL sources and Softmax HDL sources
* Specifies the scripts to run package\_xo
* Specifies the XO name for the DPU and Softmax kernels
* Builds the XO package
* Builds the XCLBIN

1-1. Review the configuration files in the lab directory.

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

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

[host]$ cd $TRAINING\_PATH/custom\_hw\_platform/lab/Vitis

1-1-3. Enter the following command to see the directories and files:

[host]$ ls

Note: Observe the directories and files available in the $TRAINING\_PATH/custom\_hw\_platform/lab/Vitis directory.

1-1-4. Enter the following command to open the prj\_config\_104\_2dpu configuration file in the editor:

[host]$ gedit config\_file/prj\_config\_104\_2dpu

This file contains configuration settings for the Vivado Design Suite project, such as:

* Clock configuration
* Connectivity configuration
* Number of kernels (nk)
* Advanced tool directives for kernel compilation
* Vivado Design Suite tool strategy

1-1-5. After completing the review, close the file.

1-1-6. Enter the following command to open the dpu\_conf.vh configuration file in the editor:

[host]$ gedit dpu\_conf.vh

This file contains the configuration of the DPU parameters:

* DPU Arch: B4096
* URAM: Enabled
* DRAM: Disabled
* RAM\_USAGE: Low
* CHANNEL\_AUGMENTATION: Enabled
* DWCV: Enabled
* POOL\_AVG: Enabled
* ELEW\_MULT: Disabled
* RELU\_LEAKYRELU\_RELU6: Enabled
* DSP48\_USAGE: High
* LOWPOWER: Disabled

1-1-7. After completing the review, close the file.

1-2. Review the Makefile in the lab directory.

1-2-1. Enter the following command to open the Makefile file in the editor:

[host]$ gedit Makefile

The following describes the various steps that the makefile performs:

* Setting the Vivado Design Suite and cross-compiler tool path (near line no. 9 and 10):



Figure 5‑: Choosing the Vivado Design Suite and Cross-Compiler Tool Path

* Setting the TARGET build configuration, such as software emulation (sw\_emu), hardware emulation (hw\_emu), or system run (hw) (near line no. 11):
* The TARGET variable defines the build configuration and is set to: sw\_emu, hw\_emu, or hw.
* The default value for the TARGET variable is set to hw.



Figure 5‑: Setting the Target Build Configuration

* Setting the EDGE\_COMMON\_SW path (near line no. 13; you will be setting the common images for the embedded Vitis platforms path separately in step 2-3-1):
* This is where the common images for the embedded Vitis platforms will be pointed. You will be pointing to the common images path, which is already available in the VM.
* This image can be downloaded (ZYNQMP common image under Common images for Embedded Vitis Platforms - 2021.2) from https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-platforms.html.
* Setting the SDX\_PLATFORM path (near line no. 14, you will be setting this custom platform path separately in step 2-3-2):
* This is where the custom Vitis platform will be pointed. You will be pointing to the ZCU104\_DPU platform, which is already available in the VM.
* This platform can be downloaded (ZCU104 Base 2021.2 under Vitis Embedded Base Platforms - 2021.2) from https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-platforms.html.
* Setting the Vitis compiler options (near line no. 16):
* -t: Set the target build configuration (hw).
* --platform: Pointing to the custom platform.
* --save-temps: Directs the v++ command to save intermediate files/directories created during the compilation and link process. Use the --temp\_dir option to specify a location to write the intermediate files to.
* --config: Provide the configuration file, in this case config\_file/prj\_config\_104\_2dpu. In this configuration file, you will set the target platform and connectivity information.
* --xp: Specifies additional parameters and properties

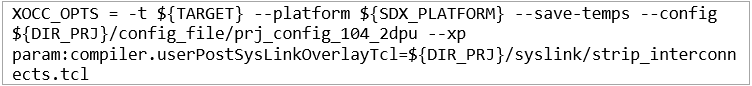


Figure 5‑: Setting the Vitis Compiler Options

* Providing the path for the DPU HDL sources and Softmax HDL sources (near line no. 17 to 34):



Figure 5‑: Providing the DPU HDL and Softmax HDL Source Paths

* Specifying the scripts to run package\_xo (near line no. 37 to 38):



Figure 5‑: Specifying the Scripts to Run package\_xo

* Setting the name of the kernels (near line no. 41 to 42 ):



Figure 5‑: Setting the Name of the Kernels

* Specifying the XO name for the DPU and Softmax kernels (near line no. 44 and 45):



Figure 5‑: Specifying the XO Name of the DPU and Softmax Kernels

* Choosing the kernel based on the provided $(KERNEL) value (near line no. 48 to 54):

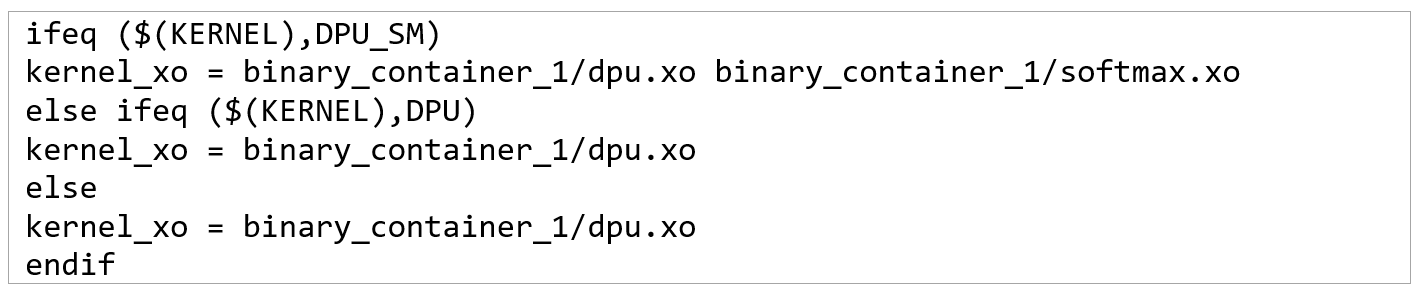


Figure 5‑: Choosing the Kernel

* Building the XO package (near line no. 60 to 68):

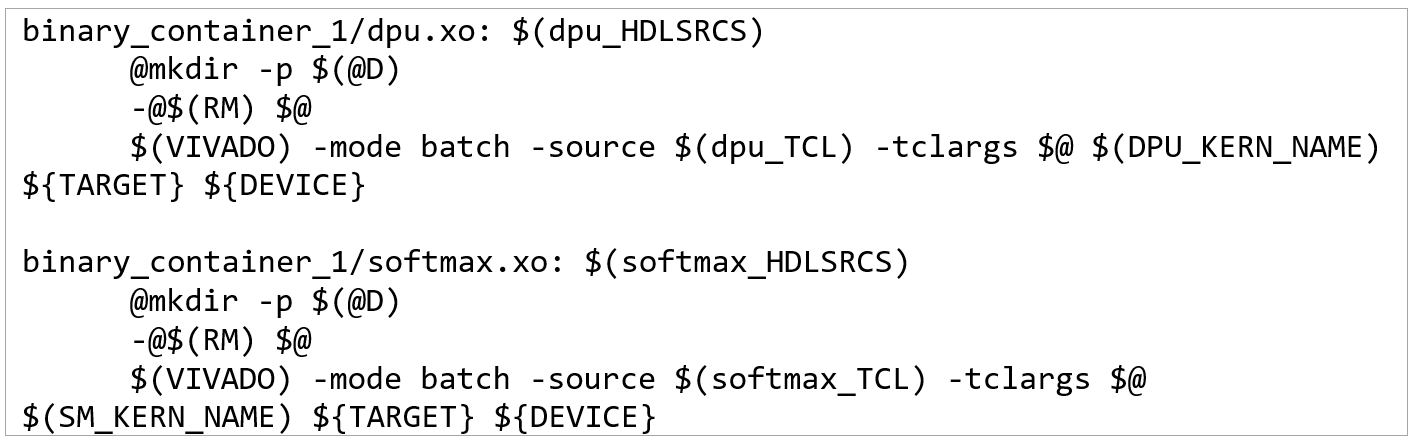


Figure 5‑: Building the XO Package

* Building the XCLBIN (near line no. 71 to 72):

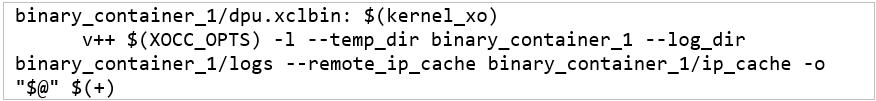


Figure 5‑: Building the XCLBIN

* Packaging (near line no. 74 to 75):
* The v++ -package, or -p step, packages the final product at the end of the Vitis compiler compile and link build process.
* Refer to UG1393: Vitis Unified Software Platform Documentation - Application Acceleration Development for more information on package and other Vitis compiler options.

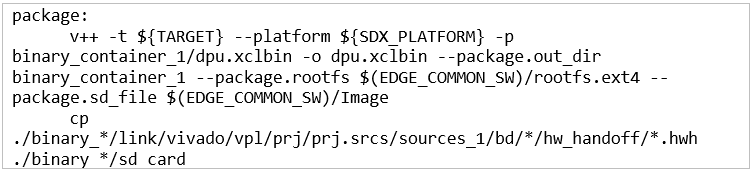


Figure 5‑: Packaging the Image

1-2-2. After completing the review, close the file.

Building the Vitis Environment Project Step

2-1. Set up the operating system's support for the Vitis environment.

2-1-1. If necessary, press <Ctrl + Alt + T> to open a new terminal window.

2-1-2. Enter the following command to source the Vitis tools:

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

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

2-2. Change the path to the lab directory.

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

[host]$ cd $TRAINING\_PATH/custom\_hw\_platform/lab/Vitis

Note: Review the instructions below on how to build the project, but do not actually run the commands. Building the project will take approximately 3–4 hours depending on your system configuration.

You will use the provided prebuilt files to create the SD card image and run the design on the target board.

2-3. Build the Vitis project for the target platform (ZCU104).

2-3-1. Enter the following command to set the path for common image for embedded platform:

[host]$ export EDGE\_COMMON\_SW=/opt/xilinx-zynqmp-common-v2021.2

For edge development: The customer training environment (CustEd\_VM) sets the ZynqMP common image to /opt/xilinx-zynqmp-common-v2021.2. Make sure that the ZynqMP common image (2021.2 version) is set in your environment and use your custom environment path..

You can choose the correct version and ZynqMP common image from: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-platforms.html.ls

2-3-2. Enter the following command to set the custom Vitis platform (ZCU104):

[host]$ export SDX\_PLATFORM=/opt/xilinx/platforms/  
xilinx\_zcu104\_base\_202120\_1/xilinx\_zcu104\_base\_202120\_1.xpfm

For edge development: The customer training environment (CustEd\_VM) sets the ZCU104 platform to /opt/xilinx/platforms. Make sure that the ZynqMP platform is set in your environment and use your custom environment path.

2-3-3. Enter the following command to build the project:

[host]$ make KERNEL=DPU DEVICE=zcu104

Note: It will take approximately 3–4 hours to build the project.

Reviewing the Output Files Step

3-1. Review the output files generated by the Vitis environment project.

3-1-1. If you built the project yourself, the files are available in the location below:

$TRAINING\_PATH/custom\_hw\_platform/lab/Vitis/binary\_container\_1/

Here are the details about the generated files:

* sd\_card.img: This will be used to flash the SD card. This has been generated by the Vitis package (-package or -p) command.
* sd\_card: This directory contains the following generated files:
* boot.scr: The u-boot configuration file.
* BOOT.BIN: Image that contains the FSBL, UBoot, and image.ub.
* dpu.xclbin: The DPU and Softmax kernels that will be programmed in the programmable logic (PL) region.
* Image: Linux kernel image.
* arch.json: Provided to the Vitis AI compiler for DPUCZDX8G configuration details.

3-1-2. If you want to use the prebuilt files, the files are available in the location below:

$TRAINING\_PATH/custom\_hw\_platform/support/Vitis/sd\_card

Preparing the SD Card Step

4-1. Burn the image file onto the SD card using Etcher software.

4-1-1. Use the image file from either of the following locations:

[Pre-built]: sd\_card.img from the $TRAINING\_PATH/custom\_hw\_platform/  
support/Vitis directory.

[User built]: sd\_card.img from the $TRAINING\_PATH/custom\_hw\_platform/  
lab/Vitis/binary\_container\_1 directory.

4-1-2. Copy the sd\_card.img to your host machine (Windows):

[VM users]: Copy the sd\_card.img file to your host machine (for example, sf\_training).

[CloudShare users]: Copy the sd\_card.img file to any cloud drive and then download from the cloud drive to your host machine.

4-1-3. Insert the SD card into your host machine.

4-1-4. Launch the balenaEtcher software.

Note: You can use any equivalent Etcher application. balenaEtcher is available from www.balena.io/etcher.

4-1-5. Click Select Image and choose the sd\_card.img file.

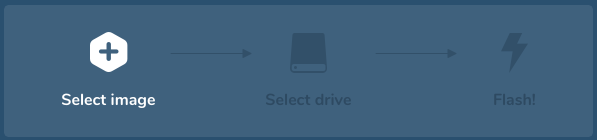


Figure 5‑: Selecting the Image

4-1-6. Select the drive and choose the SD card.

4-1-7. Click Flash to burn the image onto the SD card.

Note: This may take approximately 4-5 minutes.

This will create two partitions with FAT and ext4 formats. The naming of these partitions depends on your machine. These partitions are referred to as the BOOT partition (FAT) and ROOTFS partition (ext4) in the steps below.

4-2. Copy the application, DPU xmodel, and VART runtime library. Perform the steps below in Linux (VM).

4-2-1. Copy the app directory from the $TRAINING\_PATH/custom\_hw\_platform/  
support/Vitis directory to the SD card BOOT partition (FAT).

This app folder contains the following files:

* app > model > resnet50.xmodel: This is the DPU application generated from the trained ResNet50 model. This has been quantized by Vitis AI quantizer and compiled using the Vitis AI compiler.
* app > img: This contains the images required to be provided as input to the main application.
* app > app\_main: Main application executable.
* app > samples.tar.gz: This contains the VART library, main application source code, and main application executable. 在app\_main裡面
* samples > lib: All the necessary libraries, including VART.
* samples > include: All the necessary include files.
* samples > src > resnet50.cpp: Sample source code from which the executable has been generated.
* samples > bin > resnet50: Main application executable.

4-2-2. Remove the SD card from your host machine and insert it into the ZCU104 board.

Running the Design on the Target Board Step

Important: If you have a ZCU104 board, you can follow the instructions below. Otherwise, just review the steps to be followed.

After programming the SD card, insert the SD card into the SD card slot on the ZCU104 board.

Step 5-2 can be done on the Windows machine, which should have Tera Term application installed.

The following are required to perform these instructions:

* Evaluation board: ZCU104 board
* Serial port application: Tera Term application
* Host machine: Windows OS

5-1. Bring up the ZCU104 board.

5-1-1. 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.

5-1-2. Locate SW6 on the board.

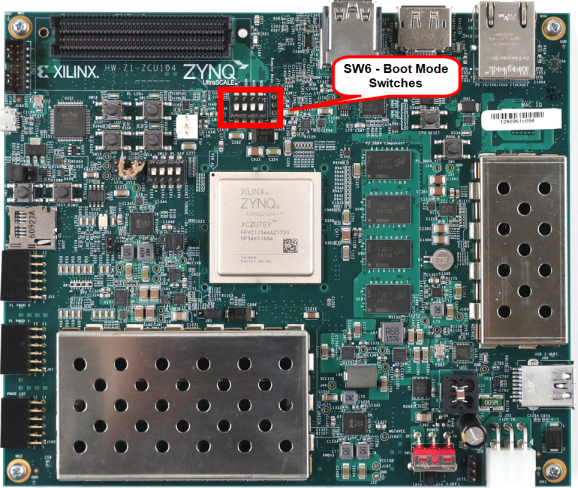


Figure 5‑: ZCU104 SW6 - Boot Mode Switches

5-1-3. 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).

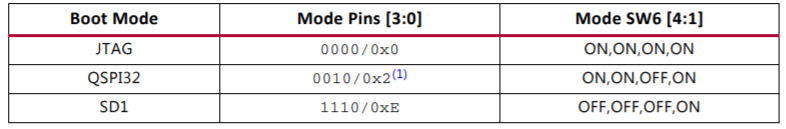


Figure 5‑: Board Configuration

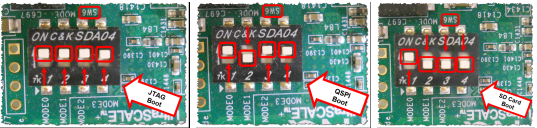


Figure 5‑: ZCU104 Boot Settings

5-1-4. Slide the power switch to the "ON" position to power on the board.

5-2. Open and configure the Tera Term terminal program.

5-2-1. From the Windows desktop, double-click the Tera Term icon to launch Tera Term.

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

5-2-2. Select File > New Connection if the Tera Term New connection window does not open automatically.

5-2-3. Select Serial as the connection (1).

5-2-4. 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.

5-2-5. Select the appropriate COM # as discovered in the previous task.

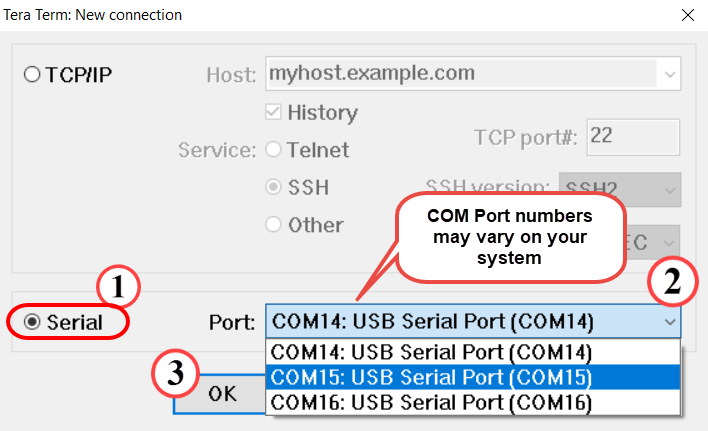


Figure 5‑: Selecting the COM Port

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.

5-2-6. Click OK (3).

The terminal console window opens.

5-2-7. Select Setup > Serial Port.

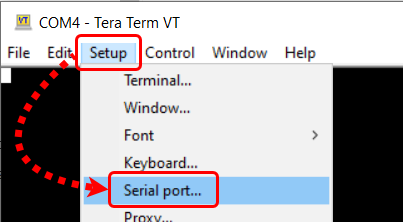


Figure 5‑: Opening the Tera Term Serial Port Setup Window

The Tera Term Serial Port Setup dialog box opens.

5-2-8. Confirm that the proper serial port has been selected (1).

5-2-9. Set the baud rate to 115200 (2).

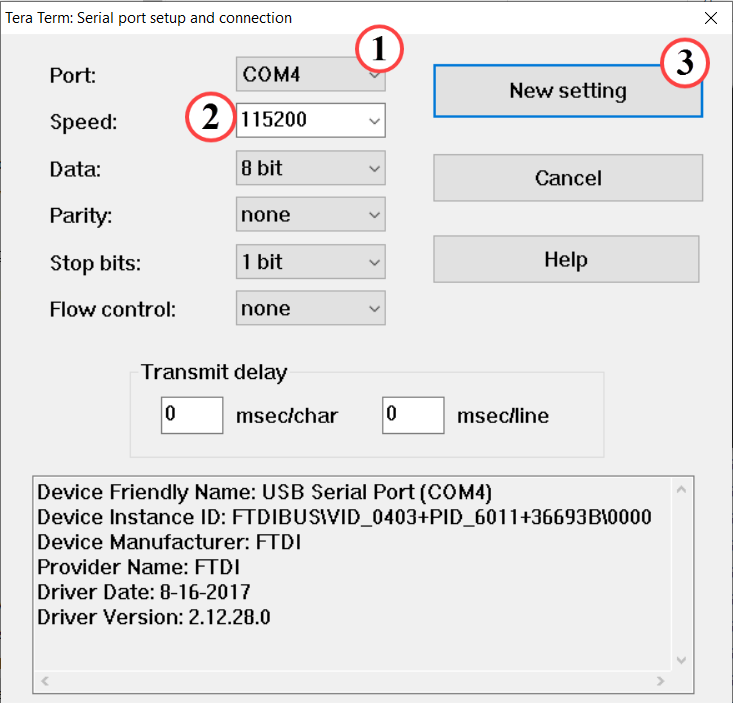


Figure 5‑: Setting the Parameters for the Serial Port

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.

5-2-10. Click OK (3).

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

5-2-11. Once booting is complete, you should see Linux boot in the terminal as shown below:

...

...

[ 6.279869] [drm] Initialized zocl 0.0.0 00000 for amba\_pl@0:zyxclmm\_drm on minor 1

[ 6.815780] FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.

[ 6.922906] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)

Starting tcf-agent: OK

PetaLinux 2021.2 zynqmp-common-2021\_1 ttyPS0

root@zynqmp-common-2021\_2:~#

5-3. Set up the IP address of the target board.

The network setup is to copy the files (such as the Vitis AI Runtime package) from the Ubuntu machine to the target board.

5-3-1. Enter the following command in the Tera Term terminal to set the IP address of the target board:

# ifconfig eth0 192.168.1.10

5-3-2. Enter the following command in the Ubuntu terminal to set the IP address of the Ubuntu VM:

[host]$ sudo ifconfig eth0 192.168.1.1

Note: Enter the password as CustEd@2k if prompted in the Customer Training VM.

To verify the connection between the Ubuntu machine and target board:

In the Tera Term terminal, enter the following command:

# ping 192.168.1.1 -c 1

You should see that the network communication works successfully.

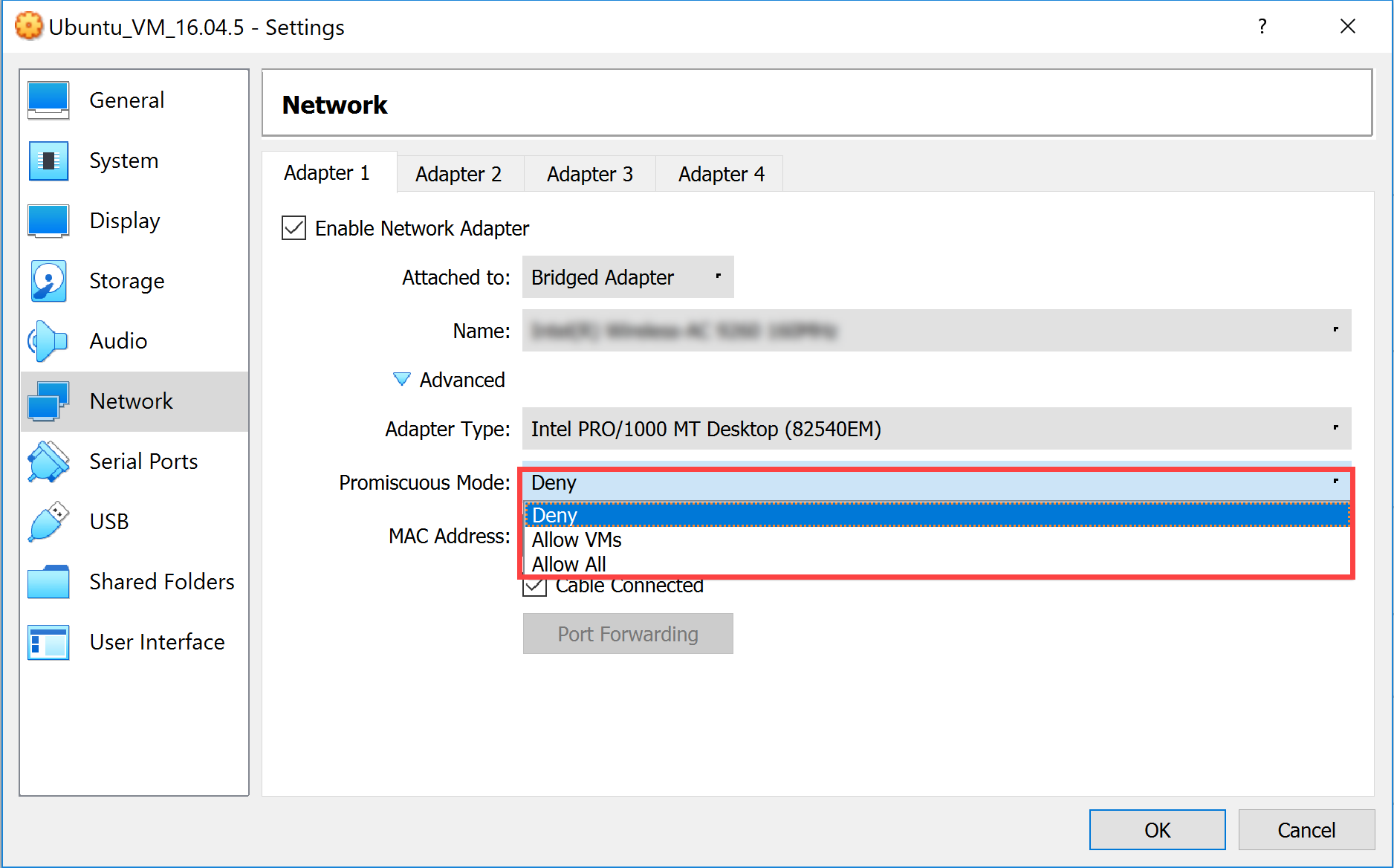
Important: There is an issue with establishing network communication in the VirtualBox VM.

Workaround: If the communication failed, follow the steps below:

1. Select Devices > Network > Network Settings.

2. Expand Advanced.

3. From the Promiscuous Mode option, change Allow ALL to DENY or vice versa.



**Figure 5‑21: VM Network Settings**

5-4. Install the Vitis AI Runtime package onto the target board.

The Vitis AI Runtime packages, VART samples, and Vitis AI library samples and models have been already built into the board image. However, you will still install the Vitis AI Runtime package to learn the procedure.

5-4-1. Press <Ctrl + Alt + T> to open a new terminal window.

5-4-2. Enter the following commands to copy the Vitis AI Runtime package to the target board by using the target board IP address:

Note: Enter the password as root if necessary.

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

[host]$ scp -r mpsoc root@192.168.1.10:~/

Note: If an error appears, indicating host key verification failure as shown below, then copy and execute the highlighted command:

ssh-keygen -f "/home/xilinx/.ssh/known\_hosts" -R "[<IP\_ADDRESS\_DISPLAYED>]"

Then run the above copy command again.

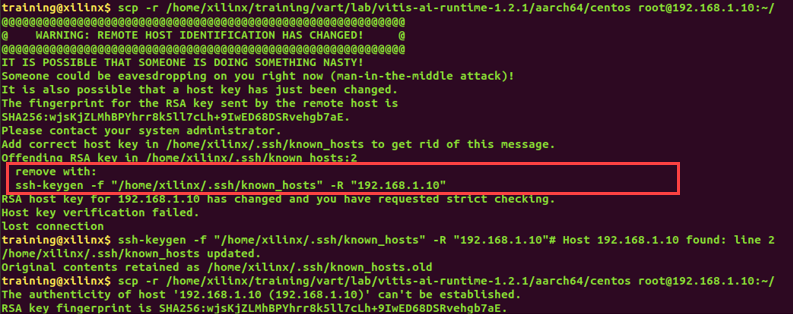


Figure 5‑: Host Key Verification Failed

5-4-3. Enter the following command in the Tera Term terminal to install the Vitis AI Runtime library:

# cd ~/mpsoc/VART

# bash target\_vart\_setup.sh

You will see the installation of Vitis AI Runtime packages as shown below:

root@xilinx-zcu104-2021\_2:~/mpsoc/VART# bash target\_vart\_setup.sh

Verifying... ################################# [100%]

%pretrans(libunilog-2.0.0-r64.aarch64): scriptlet start

%pretrans(libunilog-2.0.0-r64.aarch64): execv(/bin/sh) pid 1217

%pretrans(libunilog-2.0.0-r64.aarch64): waitpid(1217) rc 1217 status 0

Preparing... ################################# [100%]

%prein(libunilog-2.0.0-r64.aarch64): scriptlet start

%prein(libunilog-2.0.0-r64.aarch64): execv(/bin/sh) pid 1218

%prein(libunilog-2.0.0-r64.aarch64): waitpid(1218) rc 1218 status 0

Updating / installing...

...

...

1:libvitis\_ai\_library-2.0.0-r64 ################################# [100%]

%post(libvitis\_ai\_library-2.0.0-r64.aarch64): scriptlet start

%post(libvitis\_ai\_library-2.0.0-r64.aarch64): execv(/bin/sh) pid 1243

%post(libvitis\_ai\_library-2.0.0-r64.aarch64): waitpid(1243) rc 1243 status 0

%posttrans(libvitis\_ai\_library-2.0.0-r64.aarch64): scriptlet start

%posttrans(libvitis\_ai\_library-2.0.0-r64.aarch64): execv(/bin/sh) pid 1244

%posttrans(libvitis\_ai\_library-2.0.0-r64.aarch64): waitpid(1244) rc 1244 status 0

Complete VART packages installation

root@xilinx-zcu104-2021\_2:~/mpsoc/VART#

5-5. Run the sample application on the target board.

5-5-1. Enter the following commands one by one to run the application:

# cd ~

# cp /mnt/sd-mmcblk0p1/app/model/resnet50.xmodel ~

# cp -r /mnt/sd-mmcblk0p1/app/img ~

# cp /mnt/sd-mmcblk0p1/app/app\_main/resnet50 ~

# ./resnet50 img/bellpeppe-994958.JPEG

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

Expect:

score[945] = 0.993776 text: bell pepper,

score[941] = 0.00246332 text: acorn squash,

score[943] = 0.00191844 text: cucumber, cuke,

score[939] = 0.000705754 text: zucchini, courgette,

score[940] = 0.000333375 text: spaghetti squash,

The scores show that the model has detected the image as the bell pepper image with 99% accuracy (this may vary for every run).

5-5-2. Open the bellpeppe-994958.JPEG image from the $TRAINING\_PATH/  
custom\_hw\_platform/support/Vitis/app/img directory.

5-5-3. Compare the image with the results above.

You can also try using the other images available in the img directory:

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

5-5-4. After you are finished running the design, power off the board and exit Tera Term.

## Summary

In this lab, you have learned how to build a Vitis environment project with a custom platform. You also ran the design on a target board to verify results.

# 