# Vitis AI Library (VART) – Edge

Vitis AI Development Environment 2.0

## Abstract

This lab walks you through a review of the Vitis™ AI Runtime (VART) sample code and the steps to set up the host environment for cross-compilation and to run a design on the target board.

CloudShare users: You should perform this lab in your Linux environment and make sure that your Vitis environment is set up correctly.

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:

* Use the unified Vitis AI Runtime (VART) APIs in your design
* Set up the host environment for cross-compilation
* Install the Vitis AI Runtime package on the target board
* Run the application on the target board

## Introduction

The Vitis AI Runtime (VART) enables applications to use a unified high-level runtime API for both cloud and edge, making cloud-to-edge deployments seamless and efficient.

The Vitis AI Runtime API features are:

* Asynchronous submission of jobs to the accelerator
* Asynchronous collection of jobs from the accelerator
* C++ and Python implementations
* Support for multi-threading and multi-process execution

Users can use the unified Vitis AI Runtime APIs to control the DPU.

The Runtime APIs provide five interfaces:

* execute\_async()
* wait()
* get\_tensor\_format()
* get\_input\_tensors()
* get\_output\_tensors()

C++ APIs

The class name is vart::Runner. The following table lists all the functions defined in the vitis::vart::Runner class.

|  |  |  |
| --- | --- | --- |
| Return Type | Name | Arguments |
| std::unique\_ptr<Runner> | create\_runner | const xir::Subgraph\* subgraph  const std::string& mode |
| std::vector<std::unique\_ptr<Runner>> | create\_runner | const std::string& model\_directory |
| std::pair<uint32\_t, int> | execute\_async | const std::vector<TensorBuffer\*>& input  const std::vector<TensorBuffer\*>& output |
| int | wait | int jobID  int timeout |
| TensorFormat | get\_tensor\_format | – |
| std::vector<const xir::Tensor\*> | get\_input\_tensors | – |
| std::vector<const xir::Tensor\*> | get\_output\_tensors | – |

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 the Sample Code |  | Step 2:  Building  the  Application |  | Step 3:  Setting Up  the Target Board |  | Step 4:  Installing VART on the Target Board |  | Step 5:  Running on  the Target Board |

Reviewing the Sample VART Code Step

You will first clone the Vitis-AI repository from GitHub where you can find examples, reference code, and scripts.

You will then review the application sample code for the classification task provided in the sample directory.

1-1. Review the sample code (main.cc) provided in the sample directory.

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

[host]$ cd /home/xilinx/Vitis-AI/demo/VART/resnet50/src

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

[host]$ gedit main.cc

The figure below shows the main function near line no. 276.



Figure 3‑: Sample Code - main.cc file (ResNet50)

* Two arguments are required by this main function:
* <Application EXE> <MODEL>
* For example: #./resnet50 /usr/share/vitis\_ai\_library/models/  
  resnet50/resnet50.xmodel
* Create the DPURunner API:
* vart:Runner::create\_runner(<XIR\_Subgraph>, <Mode>)
* Creates an instance of the CPU/SIM/DPU runner by subgraph.
* The class name is vart::Runner. The following table lists the create\_runner function arguments.

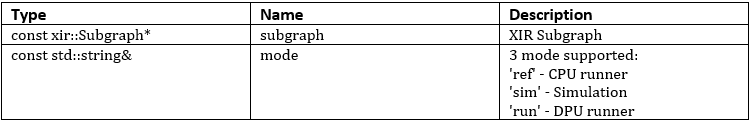


Figure 3‑: create\_runner Arguments

* Get all the input and output tensor shapes:
* get\_input\_tensors()
* Get all the input tensor of runner.
* Returns: All input tensors. A vector of raw pointer to the input tensor.
* get\_output\_tensors()
* Get all the output tensors of runner.
* Returns: All output tensors. A vector of raw pointer to the output tensor.
* Get the tensor shape:
* get\_tensor\_format()
* Get the tensor shape of the runner.
* Returns: TensorFormat: NHWC / HCHW
* Call the runResnet50() function:
* This function performs the pre-processing task, running the DPU, and the post-processing task.
* execute\_async()
* Function to execute the runner. Note that it's a block function.
* Returns: pair<jodid, status> status 0 for exit successfully, others for customized warnings or errors.
* The table lists the execute\_async function arguments.

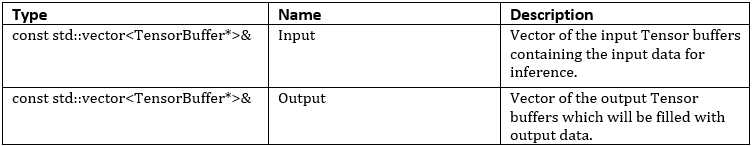


Figure 3‑: execute\_async Arguments

* wait()
* Function to wait for the end of DPU processing. It's a block function.
* Returns: Status 0 for exit successfully, others for customized warnings or errors.
* The table lists the wait function arguments.

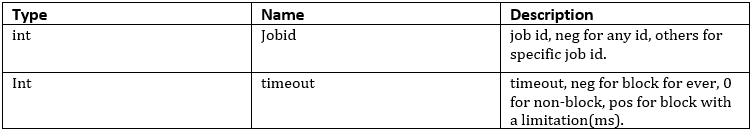


Figure 3‑: wait Arguments

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

Setting the Host Environment and Building the Application Step

You will now set up the host machine for a cross-compilation system environment to build the application. Then you will build the application, which will be copied to the target board.

2-1. Set up the host for a cross-compilation system environment.

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

[host]$ ~~cd $TRAINING\_PATH/vart/lab~~ cd Vitis-AI/setup/mpsoc

2-1-2. Install the cross-compilation system environment: host\_cross\_compiler\_setup.sh

[host]$ sudo ./sdk-2021.2.0.0.sh

Note: You may have to change the permissions for the sdk-2021.2.0.0.sh file.

If asked in the Customer Training VM for a password to execute, enter CustEd@2k.

2-1-3. Choose the path to install the cross-compilation system environment to be /opt/petalinux\_sdk\_vai.

2-1-4. Click y to install the PetaLinux SDK.

Note: This may take approximately 3-4 minutes.

PetaLinux SDK installer version 2021.2

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

Enter target directory for SDK (default: /opt/petalinux/2021.2): /opt/petalinux\_sdk\_vai

You are about to install the SDK to "/opt/petalinux\_sdk\_vai". Proceed [Y/n]? Y

Extracting SDK......................................................................................................................................................................................................................................done

Setting it up...done

SDK has been successfully set up and is ready to be used.

Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.

$ . /opt/petalinux\_sdk\_vai/  
environment-setup-cortexa72-cortexa53-xilinx-linux

training@xilinx$

2-1-5. Enter the following command to source the environment setup script:

[host]$ . /opt/petalinux\_sdk\_vai/environment-setup-cortexa72-  
cortexa53-xilinx-linux

Note: Do not forget to use the "." in the command.

2-2. Build the sample ResNet50 application, which you reviewed in the previous step.

2-2-1. Change the path to the sample directory location:

[host]$ cd /home/xilinx/Vitis-AI/demo/VART/resnet50

You will find the sample source file (main.cc) under the src directory in the above location.

2-2-2. Enter the following command to cross-compile the sample application:

[host]$ bash -x build.sh

Note: If the compilation process does not report any errors and the executable file resnet50 is generated, the host environment is installed correctly.

Preparing the SD Card and Setting Up the Target Board Step

In this step, you will prepare the SD card with the Linux image that has been already provided for you.

The following are required to perform these instructions:

* Evaluation board: ZCU104 board
* Serial port application:
* Windows OS: Tera Term application (the instructions that follow are based on Windows)
* Linux machine: GTKterm application
* Ubuntu OS: For building the application using the cross-compilation environment and copying the necessary files to the target board.

3-1. Burn the SD card with the Linux image using balenaEtcher (free, open-source software).

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

3-1-2. Launch the balenaEtcher application.

3-1-3. Locate the Linux image (name of the image) from the $TRAINING\_PATH/vart/lab directory.

Note: This Linux image (xilinx-zcu104-dpu-v2021.2-v2.0.0.img.gz) is in the Windows shared directory (C:/training/VART/lab).

3-1-4. Unzip the xilinx-zcu104-dpu-v2021.2-v2.0.0.img.gz file to extract the sd\_card.img image file.

3-1-5. Click Flash from file and select the sd\_card.img file from the $TRAINING\_PATH/  
vart/lab/xilinx-zcu104-dpu-v2021.2-v2.0.0.img directory.

3-1-6. Select the SD card drive as the target.

3-1-7. Click Flash to burn the image onto the SD card.

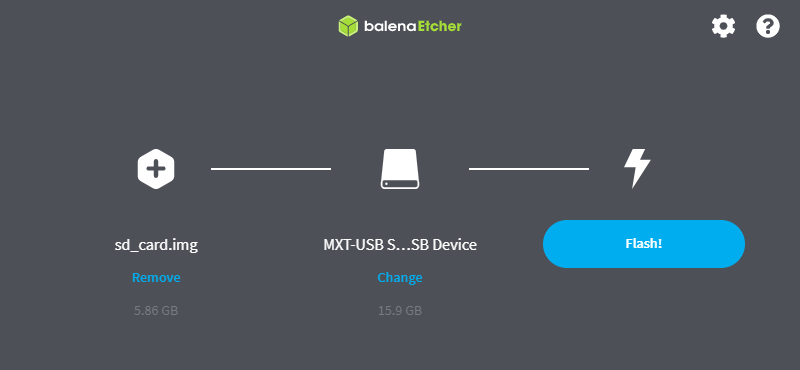


Figure 3‑: Burning the Linux Image onto the SD Card

Note: This may take approximately 5 minutes to complete.

3-1-8. After the image has been burned, click Exit from the taskbar to safely remove the card.

3-1-9. Insert the SD card into the SD card slot of the ZCU104 board.

3-2. Bring up the ZCU104 board.

3-2-1. 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.

3-2-2. Locate SW6 on the board.

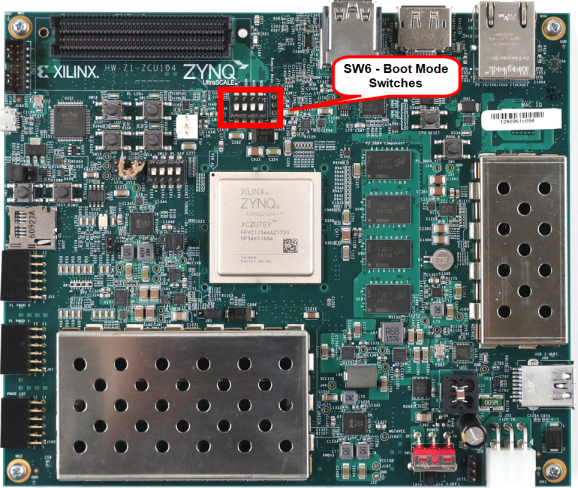


Figure 3‑: ZCU104 SW6 - Boot Mode Switches

3-2-3. 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).

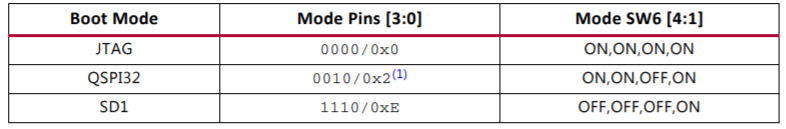


Figure 3‑: Board Configuration

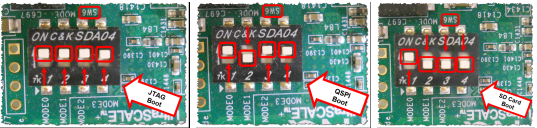


Figure 3‑: ZCU104 Boot Settings

3-2-4. Slide the power switch to the "ON" position to power on the board.

3-2-5. Connect the Ethernet cable from the target board to the host PC.

3-3. Open and configure the Tera Term terminal program.

3-3-1. From the Windows desktop, double-click the Tera Term icon to launch Tera Term.

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

3-3-2. Select File > New Connection if the Tera Term New connection window does not open automatically.

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

3-3-4. 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.

3-3-5. Select the appropriate COM # as discovered in the previous task.

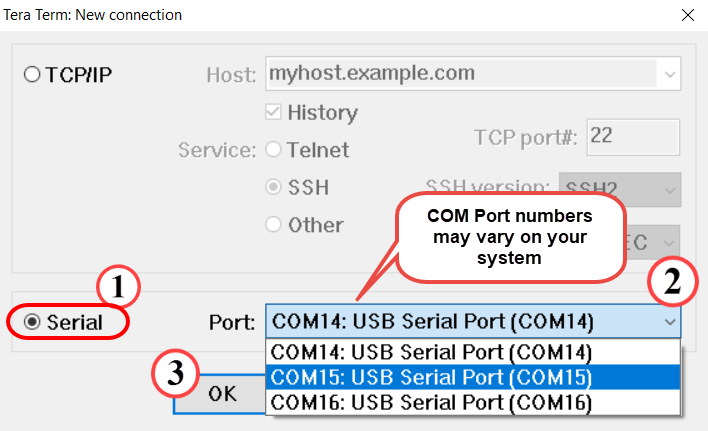


Figure 3‑: Selecting the COM Port

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.

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

The terminal console window opens.

3-3-7. Select Setup > Serial Port.

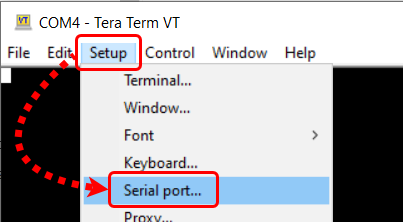


Figure 3‑: Opening the Tera Term Serial Port Setup Window

The Tera Term Serial Port Setup dialog box opens.

3-3-8. Confirm that the proper serial port has been selected (1).

3-3-9. Set the baud rate to 115200 (2).

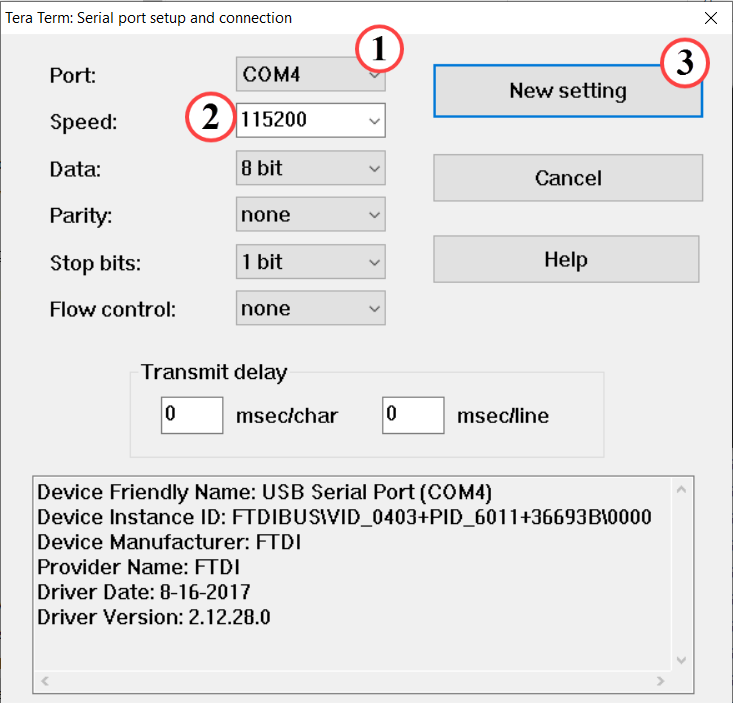


Figure 3‑: Setting the Parameters for the Serial Port

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.

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

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

3-4. Set up the IP address of the target board.

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

3-4-1. Enter the following command in the Tera Term terminal to set the IP address of the target board:

# ifconfig eth0 192.168.1.10

3-4-2. 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.

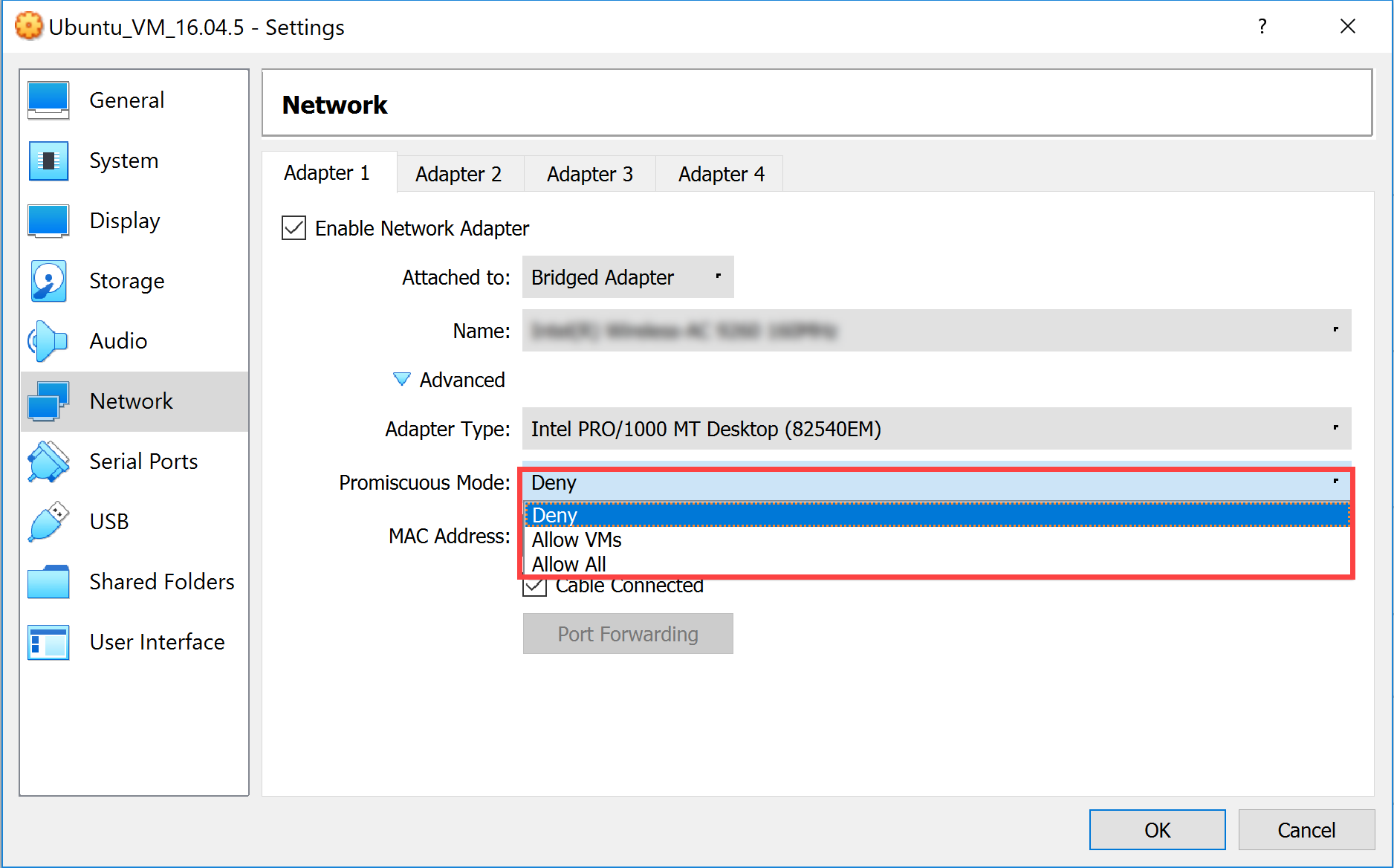
Important: There is an issue with establishing network communication in the VirtualBox VM.

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 3‑12: VM Network Settings**

Installing the Vitis AI Runtime Package on the Target Board Step

Now that your target board setup is done, install the Vitis AI Runtime package onto the target board.

4-1. 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.

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

4-1-2. 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.

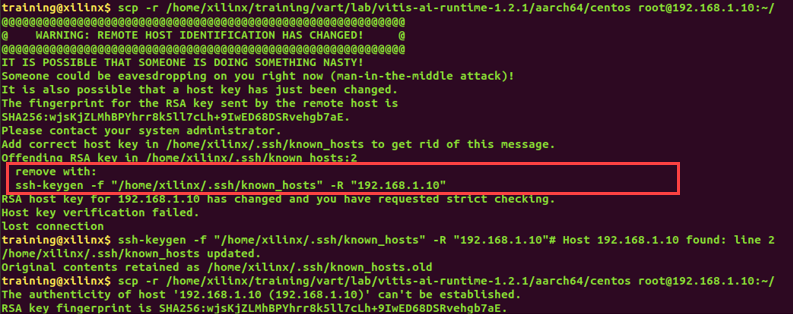


Figure 3‑: Host Key Verification Failed

4-1-3. 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#

Running the Example on the Target Board Step

Now you need to copy the sample application you have built from the host to the target board. Then run the sample design on the target board.

5-1. Copy the sample application and the sample images/videos from the host to the target board.

5-1-1. Copy the samples from the host to the target using scp with the following command:

[host]$ scp -r /home/xilinx/Vitis-AI/demo/VART/resnet50/resnet50 root@192.168.1.10:/home/root/Vitis-AI/demo/VART/resnet50

Note: Enter the password as root if necessary.

5-1-2. Copy the vitis\_ai\_runtime\_r2.0.0\_image\_video.tar.gz file from the host to the target:

[host]$ scp /home/xilinx/training/vart/lab/  
vitis\_ai\_runtime\_r2.0.0\_image\_video.tar.gz root@192.168.1.10:~/

Note: Enter the password as root if necessary.

5-1-3. Unzip the vitis\_ai\_runtime\_r2.0.0\_image\_video.tar.gz package onto the target board:

# cd ~

# tar -xzvf vitis\_ai\_runtime\_r2.0.0\_image\_video.tar.gz -C Vitis-AI/demo/VART

5-2. Run the sample application on the target board.

5-2-1. Enter the directory of the samples on the target board:

# cd ~/Vitis-AI/demo/VART/resnet50

5-2-2. Enter the following command to run the design:

# ./resnet50 /usr/share/vitis\_ai\_library/models/resnet50/  
resnet50.xmodel

5-2-3. Verify the output results.

Image : 001.jpg

top[0] prob = 0.982662 name = brain coral

top[1] prob = 0.008502 name = coral reef

top[2] prob = 0.006621 name = jackfruit, jak, jack

top[3] prob = 0.000543 name = puffer, pufferfish, blowfish, globefish

top[4] prob = 0.000330 name = eel

You can compare with the 001.jpg input image located at $vart/lab/vitis\_ai\_runtime\_r2.0.0\_image\_video.tar.gz.

Extract the vitis\_ai\_runtime\_r2.0.0\_image\_video.tar.gz file, and you will find the image located in the samples/images directory.

5-2-4. Close the Tera Term application.

5-2-5. Power off the board.

## Summary

In this lab, you have reviewed the Vitis AI Runtime sample code, set up the host environment, and run the design on the target board.

# 