# Creating a Hardware Platform with the DPU Using the Vivado Design Suite Flow – Edge

Vivado Design Suite 2021.2

## Abstract

This lab walks you through the steps to build a custom platform with the DPUCZDX8G IP using the Vivado® Design Suite.

This lab should take approximately 60 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:

* Create a Vivado Design Suite project
* Choose the target device and add the DPUCZDX8G IP
* Configure the various parameters for the DPUCZDX8G IP
* Generate the bitstream and export the hardware

## Introduction

In this lab, a custom platform will be created using the Vivado Design Suite. The block diagram below shows the complete flow of the Vitis AI™ development environment.

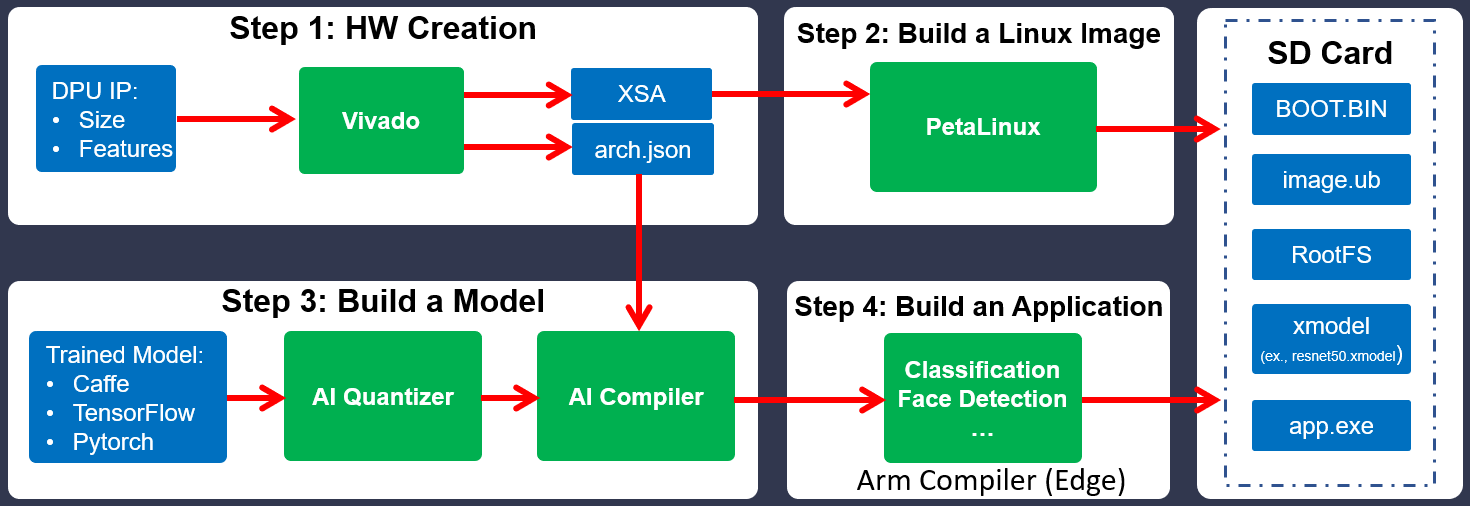


Figure 4‑: Custom Application Development Flow Using the Vivado Design Suite

When choosing the Vivado Design Suite flow to create a custom platform and AI application targeting edge devices, the following four steps are recommended:

* Step 1: Create a hardware platform using the Vivado Design Suite.
* Step 2: Build the Linux image using the PetaLinux tools.
* Step 3: Build a model using the Vitis AI quantizer and AI compiler.
* Step 4: Create an AI application.

In this lab, you will be focusing on step 1, creating a hardware platform using the Vivado Design Suite.

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 Vivado  Project |  | Step 2:  Reviewing the DPU IP  Config. |  | Step 3:  Generating  the  Bitstream |  | Step 4:  Exporting  the  Hardware |

Creating a Vivado Design Suite Project Step

You will begin the lab by creating a new Vivado Design Suite project and adding the DPU IP to the block design. These tasks will be accomplished by running a provided Tcl script.

1-1. Launch the Linux terminal and source the Vivado Design Suite.

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/Vivado

1-1-3. Source the Vivado Design Suite by entering the following command:

[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..

1-2. Run the trd\_prj.tcl script, which will perform the following tasks:

* Create a Vivado Design Suite project
* Choose the target board
* Create a block design with following IP:
* Zynq® UltraScale+™ MPSoC
* Processor System Reset
* Hierarchical DPU block
* Concatenate block (interrupt signals from the DPU IP and Softmax IP)

1-2-1. Enter the following command to source the trd\_prj.tcl script:

[host]$ vivado -source scripts/trd\_prj.tcl

Note: The script will take approximately 5-7 minutes to create the block design and the HDL wrapper in the Vivado Design Suite.

The Vivado IP integrator block design should resemble the figure below.

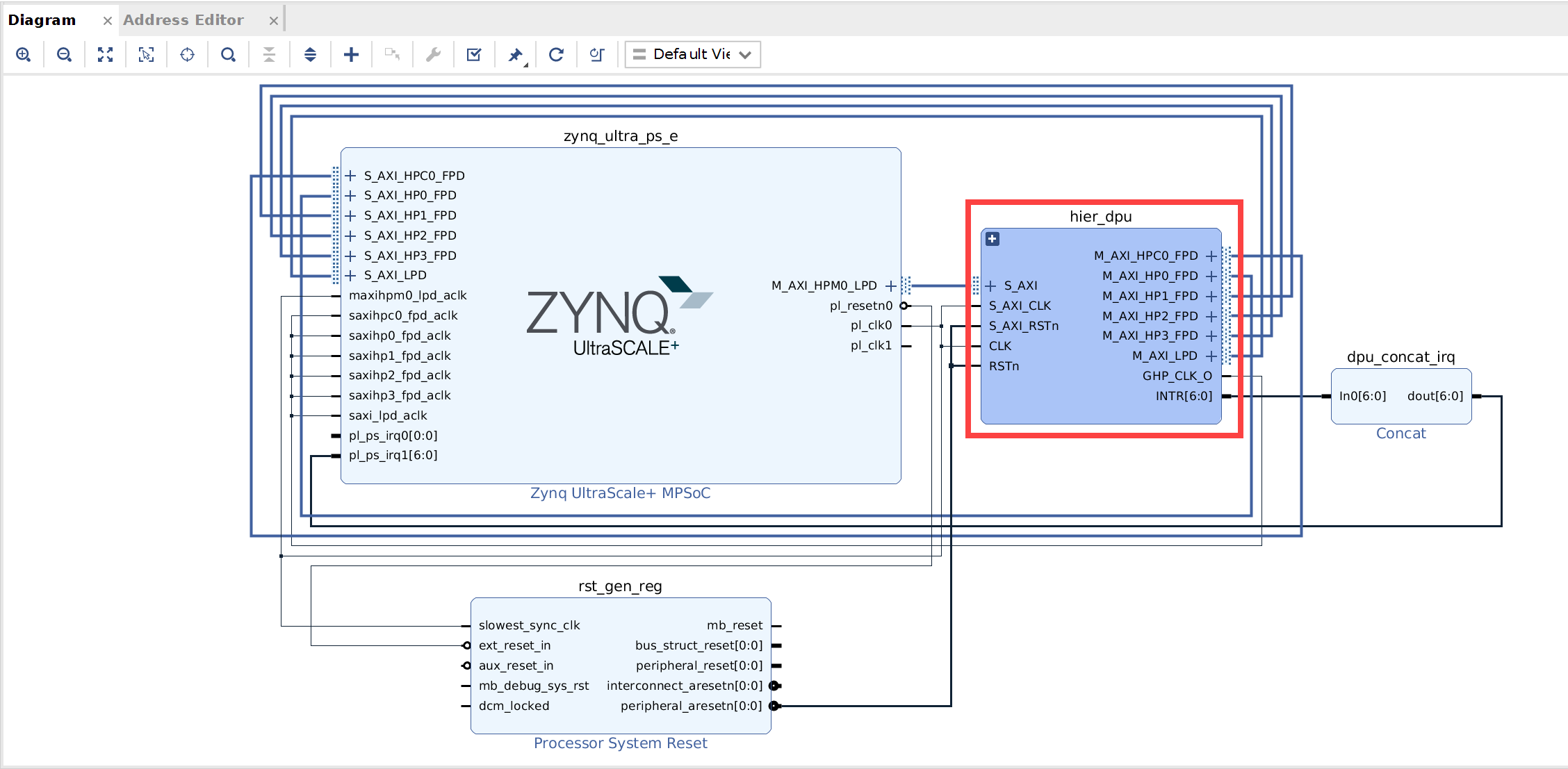


Figure 4‑: Vivado IPI Block Design with the DPU IP

1-2-2. Take note of the blocks that were added for this design.

Reviewing the DPU IP Configuration Step

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.

There are different variants of DPU IPs available.

| DPU Name | Hardware Platform |
| --- | --- |
| DPUCZDX8G | Zynq UltraScale+ MPSoC |
| DPUCAHX8H | Alveo™ U50LV, U55C Data Center accelerator cards |
| DPUCADF8H | Alveo U200, U250 Data Center accelerator cards |
| DPUCVDX8G | Versal® ACAP VCK190 (Versal AI Core Series) evaluation kit |
| DPUCVDX8H | Versal ACAP VCK5000 evaluation kit |

In this lab, you will be using the DPUCZDX8G IP.

Features:

* One AXI slave interface for accessing configuration and status registers
* One AXI master interface for accessing instructions
* Support for individual configuration of each channel
* Some highlights of DPU functionality include:
* Configurable hardware architecture: B512, B800, B1024, B1152, B1600, B2304, B3136, and B4096
* Maximum of four homogeneous cores
* Convolution and deconvolution
* Depthwise convolution and depthwise transposed convolution
* Max pooling
* Average pooling
* ReLU, RELU6, and Leaky ReLU
* Elementwise-Sum amd Elementwise-Multiply
* Dilation
* Reorg
* Fully connected layer
* Softmax
* Concat, Bach normalization: supported by the tool chain

2-1. Review the DPU IP parameters that are configured in the trd\_prj.tcl file.

2-1-1. Press <Ctrl + Alt + T> to open a new terminal window.

2-1-2. Enter the following command to open the trd\_prj.tcl file:

[host]$ gedit $TRAINING\_PATH/custom\_hw\_platform/lab/Vivado/  
scripts/trd\_prj.tcl

2-1-3. Observe the following configurations in the script:

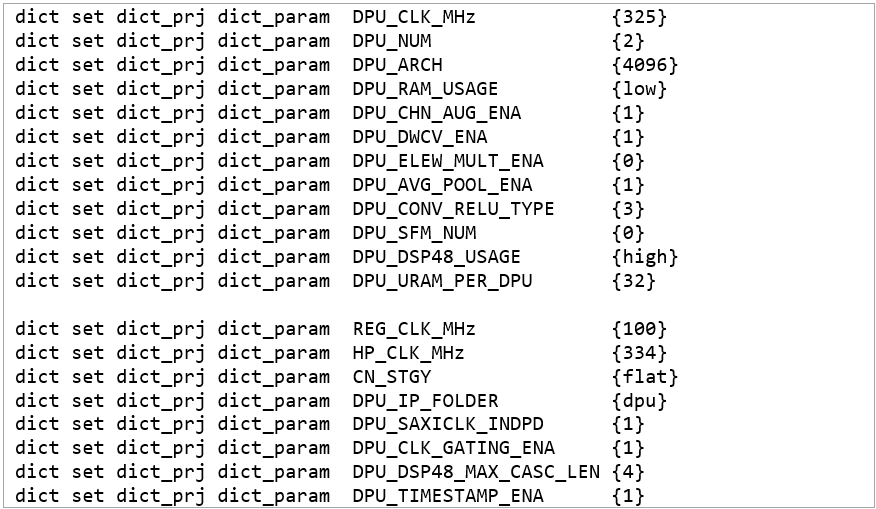


Figure 4‑: Setting the DPU IP Parameters

The table below provides detailed information on the different DPU IP configurations:

| Parameter Name (in the Script) | Value | Description |
| --- | --- | --- |
| DPU\_CLK\_MHz | 325 | DPU IP clock. |
| DPU\_NUM | 2 | Number of DPU cores. |
| DPU\_ARCH | 4096 | Architecture of the DPU: The architectures for the DPU IP include B512, B800, B1024, B1152, B1600, B2304, B3136, and B4096. |
| DPU\_RAM\_USAGE | low | RAM usage: The weights, bias, and intermediate features are buffered in the on-chip memory.  The on-chip memory consists of RAM that can be instantiated as block RAM and UltraRAM. The RAM Usage option determines the total amount of on-chip memory used in different DPU architectures, and the setting is for all the DPU cores in the DPU IP. High RAM Usage means that the on-chip memory block will be larger, allowing the DPU more flexibility to handle the intermediate data. |
| DPU\_CHN\_AUG\_ENA | 1 | Channel augmentation: Channel augmentation is an optional feature for improving the efficiency of the DPU when handling input channels much lower than the available channel parallelism. |
| DPU\_DWCV\_ENA | 1 | DepthwiseConv: In depthwise separable convolution, the operation is performed in two steps: depthwise convolution and pointwise convolution. |
| DPU\_ELEW\_MULT\_ENA | 0 | Input channel: 1 - 256 \* channel\_parallel.  Input size: Arbitrary. |
| DPU\_AVG\_POOL\_ENA | 1 | AveragePool: The AveragePool option determines whether the average pooling operation will be performed on the DPU or not. The supported sizes range from 2x2, 3x3, …, to 8x8, with only square sizes supported. |
| DPU\_CONV\_RELU\_TYPE | 3 | ReLU type: The ReLU Type option determines which kind of ReLU function can be used in the DPU. ReLU and ReLU6 are supported by default.  The option "ReLU + LeakyReLU + ReLU6" means that the DPU can also include limitations on the allowed coefficients for LeakyReLU as an activation function. |
| DPU\_SFM\_NUM | 0 | Softmax: This option allows the Softmax function to be implemented in hardware. The hardware implementation of Softmax can be 160 times faster than a software implementation.  The hardware Softmax module takes approximately 10000 LUTs, four block RAMs, and 14 DSPs. Enabling this option depends on the available hardware resources and desired throughput.  When Softmax is enabled, an AXI master interface named SFM\_M\_AXI and an interrupt port named sfm\_interrupt will appear in the DPUCZDX8G IP. The Softmax module uses m\_axi\_dpu\_aclk as the AXI clock for SFM\_M\_AXI as well as for computation. |
| DPU\_DSP48\_USAGE | high | DSP usage: This allows you to select whether DSP48E slices will be used for accumulation in the DPU convolution module. When low DSP usage is selected, the DPU IP will use DSP slices only for multiplication in the convolution. In high DSP usage mode, the DSP slice will be used for both multiplication and accumulation. |
| DPU\_URAM\_PER\_DPU | 32 | UltraRAM: There are two kinds of on-chip memory resources in Zynq UltraScale+ devices: block RAM and UltraRAM. The available amount of each memory type is device dependent. Each block RAM block consists of two block RAM 18K slices that can be configured as 9b\*4096, 18b\*2048, or 36b\*1024. UltraRAM has a fixed configuration of 72b\*4096. The DPU uses block RAM as the memory unit by default. |
| DPU\_SAXICLK\_INDPD | 1 | S-AXI clock mode: s\_axi\_aclk is the S-AXI interface clock. When Common with M-AXI Clock is selected, s\_axi\_aclk shares the same clock as m\_axi\_aclk, and the s\_axi\_aclk port is hidden. When Independent is selected, a clock different from m\_axi\_aclk is provided. |
| DPU\_CLK\_GATING\_ENA | 1 | dpu\_2x clock gating: dpu\_2x clock gating is an option for reducing the power consumption of the DPU. When the option is enabled, a port named dpu\_2x\_clk\_ce appears for each DPU core. |
| DPU\_DSP48\_MAX\_CASC\_LEN | 4 | DSP cascade: The maximum length of the DSP48E slice cascade chain can be set. Longer cascade lengths typically use fewer logic resources but might have worse timing. Shorter cascade lengths might not be suitable for small devices as they require more hardware resources. Xilinx recommends selecting the mid-value, which is 4, in the first iteration and adjusting the value if the timing is not met. |
| DPU\_TIMESTAMP\_ENA | 1 | Timestamp: When enabled, the DPU records the time that the DPU project was synthesized. When disabled, the timestamp keeps the value at the moment of the last IP update. |

2-1-4. After completing your review, close the file.

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

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

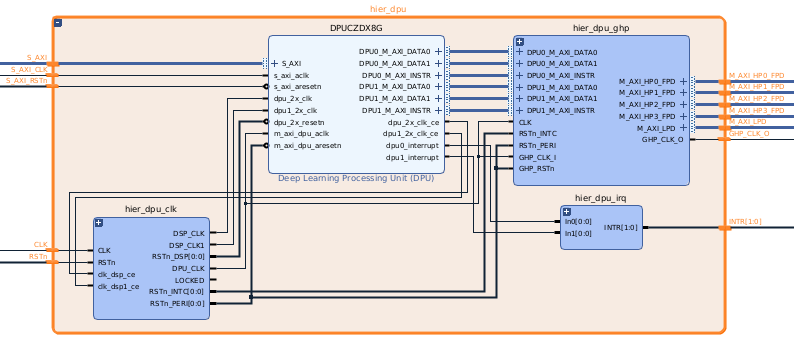


Figure 4‑: Hierarchical Block of the DPU IP

Note: If the tool is not responding, close the Vivado IDE. Launch the Vivado IDE again and open the project. In the Flow Navigator, click Settings under Project Manager, select Project Settings > IP and click Clear Cache under the IP Cache section. Then click Open Block Design in IP integrator.

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

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

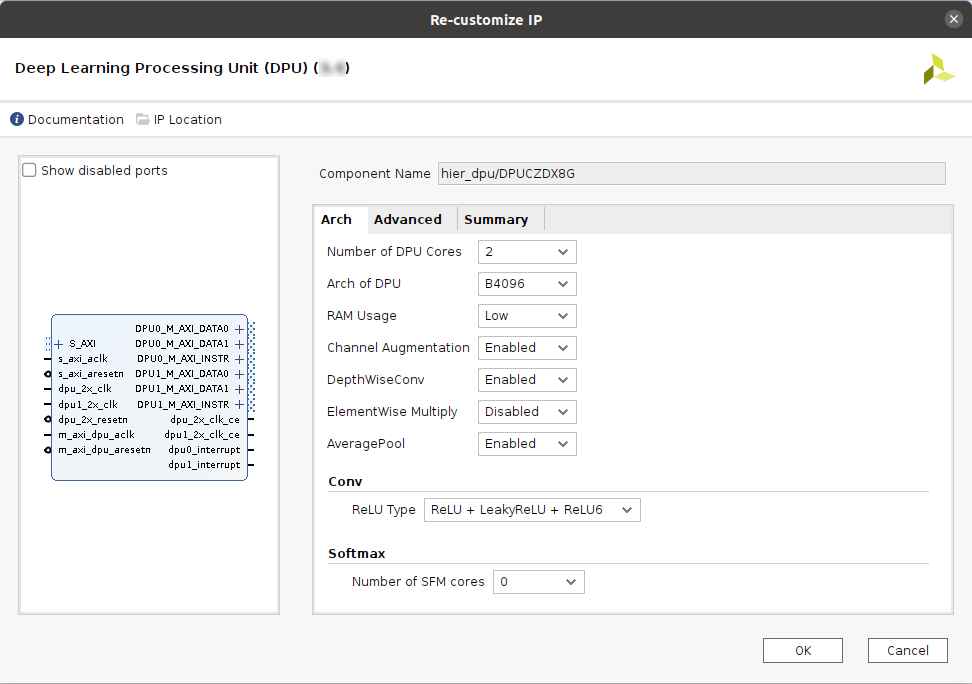


Figure 4‑: DPU IP Customization - Arch Tab

2-2-4. Review all the options.

You can see that the settings are based on the trd\_prj.tcl script.

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

Note: Do not change any settings.

2-2-6. Click the Advanced tab and review the settings.

2-2-7. Click the Summary tab to verify the selected options.

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

Generating the Bitstream Step

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

Note: Do not perform this step as generating the bitstream will take approximately 3-4 hours depending on your system configuration. The instructions below are provided for review on what the necessary actions are.

For CloudShare users, if you are going to perform this step, 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.

3-1. Generate the bitstream.

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

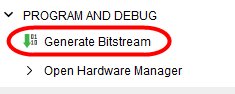


Figure 4‑: Generate Bitstream in the Flow Navigator

3-1-2. 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.

Exporting the Hardware Step

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 arch.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.

If you have performed the "Generating the Bitstream" step, proceed with this step. Otherwise, just review the instructions below on what needs to be done after bitstream generation.

4-1. Export the hardware platform.

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

4-1-2. Click Next.

4-1-3. Select Include bitstream in the Export Hardware window.

4-1-4. Click Next.

4-1-5. Leave the options in the Files section (XSA file name and export location) at their default.

4-1-6. Click Finish.

The generated files are:

* XSA file: Created at $TRAINING\_PATH/custom\_hw\_platform/lab/Vivado/  
  prj/top\_wrapper.xsa.
* The XSA file is used for software development.
* arch.json file: Created at $TRAINING\_PATH/custom\_hw\_platform/lab/  
  Vivado/srcs/top/ip/top\_dpu\_0/arch.json.
* The arch.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.

arch.json is under

custom\_hw\_platform/lab/Vivado/srcs/top/ip/top\_DPUCZDX8G\_0\_5/

Note: The prebuilt file is located at $TRAINING\_PATH/custom\_hw\_platform/  
support/Vivado/pre-built.

## Summary

In this lab, you have learned how to create custom hardware using the Vivado Design Suite and customize the DPU IP. You also generated the platform and reviewed the exported hardware files.

# 