AMD Vitis AI:

Getting started with KRIA KV260

Step by Step Guide VAI3.0

Contents

Prerequisites for Vitis-AI

Vitis AI Host (Developer) Machine Requirements — Vitis™ AI 3.0 documentation (xilinx.github.io)

- OS: Ubuntu 20.04, CentOS 7.8, 7.9, 8.1, RHEL 8.3, 8.4
- CPU: Intel i3/i5/i7/i9/Xeon 64-bit CPU, AMD EPYC 7F52 64-bit CPU
- CUDA GPU: NVIDIA GPUs supporting CUDA 11.3 or higher,
- CUDA Driver: NVIDIA-465.19.01 or higher for CUDA 11.3
- Docker: 19.03 or higher, nvidia-docker2

Vitis-AI Docker Installation

1. **Clone Vitis-AI GitHub repository**

```
cd ~
git clone https://github.com/Xilinx/Vitis-AI
cd Vitis-AI/docker/
git checkout v3.0
```

2. **Build Vitis-AI Docker**

   Host Installation Instructions — Vitis™ AI 3.0 documentation (xilinx.github.io)

   In Vitis-AI v3.0, you can now build a docker for specific target framework (cpu, gpu or rocm) and desired conda environment (tf1, tf2, pytorch). In this use case, we will build tf2 with GPU acceleration:

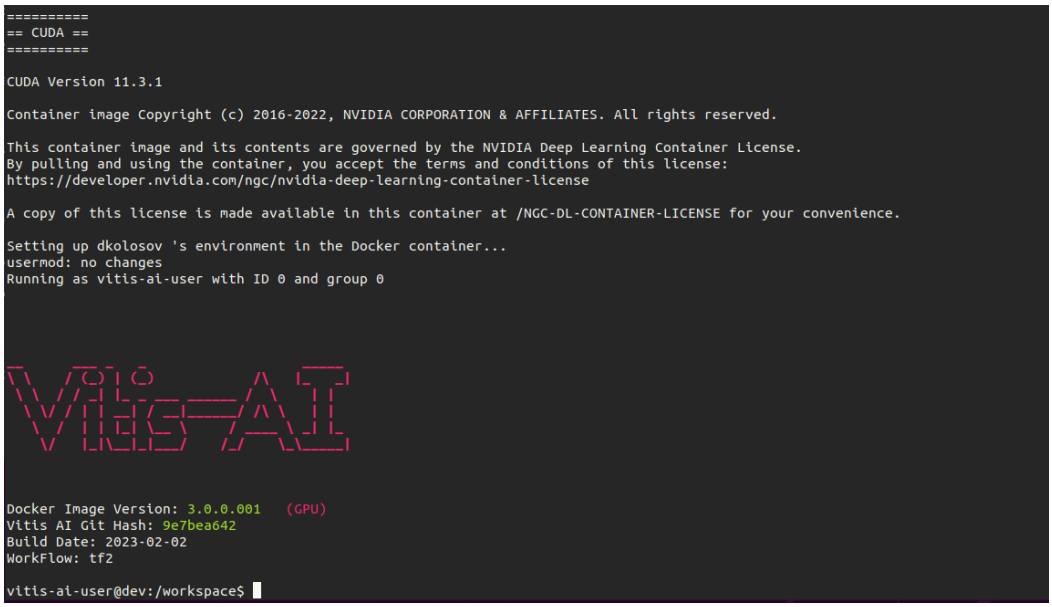
```
./docker_build.sh -t gpu -f tf2
```

Launching Vitis-AI Docker

1. **Launch Vitis-AI Docker we just built**

   Previously you required to activate conda environment after launching docker, where with v3.0 that isn't required.

```
cd Vitis-AI/
./docker_run.sh xilinx/vitis-ai-tensorflow2-gpu:latest
```



End-to-end Model Zoo Example

1. **Navigate to Model Zoo**

```
cd model_zoo
```

2. **Download pre-trained models, for TF2**
   By running python script downloader.py, you can choose which framework, model and target edge device to download. In our example, we will download TF2, ResNet50 model, GPU version. Note, the numbering might different.

```
python3 downloader.py

tf2     # tensorflow2.x
2       # tf2_resnet50_imagenet_224_224_0.78G_3.0
1       # GPU
```

```
vitis-ai-user@dev:/workspace$ cd model_zoo/
vitis-ai-user@dev:/workspace/model_zoo$ python3 downloader.py
Tip:
you need to input framework and model name, use space divide such as tf vgg16
tf:tensorflow1.x  tf2:tensorflow2.x  cf:caffe  dk:darknet  pt:pytorch  all: list all model
input:tf2
chose model
0 : all
1 : tf2_efficientnet-b0_imagenet_224_224_0.78G_3.0
2 : tf2_resnet50_imagenet_224_224_7.76G_3.0
3 : tf2_erfnet_cityscapes_512_1024_54G_3.0
4 : tf2_efficientnet-lite_imagenet_224_224_0.77G_3.0
5 : tf2_2d-unet_nuclei_128_128_5.31G_3.0
6 : tf2_yolov3_coco_416_416_65.9G_3.0
7 : tf2_inceptionv3_imagenet_299_299_11.5G_3.0
8 : tf2_mobilenetv3_imagenet_224_224_132M_3.0
9 : tf2_mobilenetv1_imagenet_224_224_1.15G_3.0
input num:2
chose model type
0: all
1 : GPU
2 : zcu102 & zcu104 & kv260
3 : vck190
4 : vck5000-DPUCVDX8H-4pe
5 : vck5000-DPUCVDX8H-6pe-aieDWC
6 : vck5000-DPUCVDX8H-6pe-aieMISC
7 : vck5000-DPUCVDX8H-8pe
input num:1
tf2_resnet50_imagenet_224_224_7.76G_3.0.zip
                                          100.0%|100%
done
vitis-ai-user@dev:/workspace/model_zoo$
```

3. **Unzip the downloaded model**

```
unzip tf2_resnet50_imagenet_224_224_7.76G_3.0.zip
```

4. **Download validation dataset**
   Need to download dataset from [ImageNet (image-net.org)](), which does require account registration. Once registered, download validation dataset 2012 from ILSVRC-2012

   File name: ILSVRC2012_img_val.tar
   Size: 6.7GB

5. **Prepare Dataset**
   Place tar file in tf2_resnet50_imagenet_224_224_7.76G_3.0/data/
   Extract and you will see folder “ILSVRC2012_img_val” with the images inside
   rename “ILSVRC2012_img_val” folder to “validation”
   Copy and run provided (by Avnet) script that sorts the images in their corresponding folders.
   Must place script in folder:
   model_zoo/tf2_resnet50_imagenet_224_224_7.76G_3.0/code/gen_data

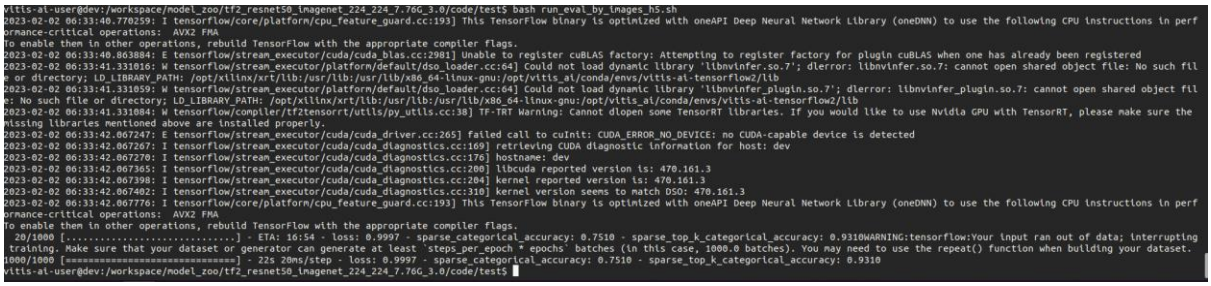
```
cd tf2_resnet50_imagenet_224_224_7.76G_3.0/code/gen_data
bash pt_images_process.sh
```

6. **Pre-Process Dataset**

```
bash get_dataset.sh
```

7. **Evaluate model**

```
cd ../test
bash run_eval_by_images.sh
```



8. **Quantize model**

```
cd ../quantize
bash run_quantize_by_images_h5.sh
```

9. **Evaluate quantized model**
   By modifying script /code/test/run_eval_by_images.sh, we can evaluate the quantized model.
   make the following changes:

   [Line 18]: --model ../../~~float~~/~~resnet_50~~.h5 \ → --model ../../quantized/quantized.h5 \
   [Line 24]: --gpus 1 → --gpus 1 \
   [Line 25]: --quantize_eval=true

```
cd ../../test
bash run_eval_by_images.sh
```

10. **Compare Results**

	Accuracy %	Accuracy Top5 %

(FP32) float h5	75.10	93.10
(INT8) quant h5	75.30	93.10

11. Compile quantized model for DPU

Next step is to compile model (*.xmodel) for the target architecture.

First create a json file, targeting the architecture of DPU. For the pre-compiled PetaLinux image, the fingerprint ID is as follows:

```
cd ../../quantized
echo '{"fingerprint":"0x101000056010407"} > arch_kv260.json
```

And then compile using vai_c_tensorflow2:

```
vai_c_tensorflow2- \
--model quantized.h5 \
--arch arch_kv260.json \
--output_dir ./ \
--net_name resnet50
```

Output model in same directory named “resnet50.xmodel” defined by “net_name”.

12. Required files for following steps:

a. resnet50.xmodel
   Model we just compiled in Vitis-AI for on-board testing on KV260.

b. Test image/video
   You can download images (488MB) or videos (1.2GB) examples from UG1354, that can be used with the Vitis AI Library examples.

c. **Pre-build PetaLinux** SD Card image for KV260
   Setting Up the ZCU102/ZCU104/KV260/VCK190 Evaluation Board • Vitis AI User Guide (UG1414) • Reader • Documentation Portal (xilinx.com)

13. Flash SD Card

Using BalenaEtcher, flash SD Card with a pre-build PetaLinux image from 12.b.

14. **Power KV260**

    After powering KV260, connect to it via serial terminal, e.g. PuTTy or TeraTerm

15. **Copy files to KV260**

    Copy model (12.a) and test images/video (12.b) to KV260.

16. **Untar image/video examples:**

    Untar test data on KV260 via serial terminal as follows:

```
cd ~
tar -xzvf vitis_ai_library_r3.0*_images.tar.gz -C Vitis-AI/examples/vai_library
tar -xzvf vitis_ai_library_r3.0*_video.tar.gz -C Vitis-AI/examples/vai_library
```

17. **Quick DPU check**

```
show dpu
```

```
root@xilinx-kv260-starterkit-20222:~# show_dpu
device_core_id=0 device= 0 core = 0 fingerprint = 0x101000056010407 batch = 1 full_cu_name=DPUCZDX8G:DPUCZDX8G_1

root@xilinx-kv260-starterkit-20222:~#
```

```
xdputil query
```

```
root@xilinx-kv260-starterkit-20222:~# xdputil query
{
    "DPU IP Spec":{
        "DPU Core Count":1,
        "IP version":"v4.1.0",
        "generation timestamp":"2022-11-30 19-15-00",
        "git commit id":"ce8dd1",
        "git commit time":2022113019,
        "regmap":"1to1 version"
    },
    "VAI Version":{
        "libvaip-core.so":"Xilinx vaip Version: 1.0.0-a176db67b19f94b0a31f9d24ef80322efe4494ad  2022-12-27-01:24:22 ",
        "libvart-runner.so":"Xilinx vart-runner Version: 3.0.0-2efa5fe1e56c2b2c8a7e71e9fc1636242dd50a9f  2022-12-27-00:47:05 ",
        "libvitis_ai_library-dpu_task.so":"Xilinx vitis_ai_library dpu_task Version: 3.0.0-1cccff04dc341c4a6287226828f90aed56005f4f  2022-12-20 10:29:01 [UTC] ",
        "libxir.so":"Xilinx xir Version: xir-9204ac72103092a7b253a0c23ec7471481656940 2022-12-27-00:46:16",
        "target_factory":"target-factory.3.0.0 860ed0499ab009084e2df3004eeb9ae710c26351"
    },
    "kernels":[
        {
            "DPU Arch":"DPUCZDX8G_ISA1_B4096",
            "DPU Frequency (MHz)":300,
            "IP Type":"DPU",
            "Load Parallel":2,
            "Load augmentation":"enable",
            "Load minus mean":"disable",
            "Save Parallel":2,
            "XRT Frequency (MHz)":300,
            "cu_addr":"0xa0010000",
            "cu_handle":"0xaaaacfe29490",
            "cu_idx":0,
            "cu_mask":1,
            "cu_name":"DPUCZDX8G:DPUCZDX8G_1",
            "device_id":0,
            "fingerprint":"0x101000056010407",
            "name":"DPU Core 0"
        }
    ]
}
root@xilinx-kv260-starterkit-20222:~#
```

We can see, B4096 running at 300MHz, with 2x DPU cores

18. **Quick Benchmarking Test**

We can use xdputil utility and perform quick benchmarks (with dummy data) and derive throughput as follows:

```
xdputil benchmark <model.xmodel> <num of threads>
```

```
xdputil benchmark resnet50.xmodel 1
```

```
root@xilinx-kv260-starterkit-20222:~# xdputil benchmark resnet50.xmodel 1
Nov 19 09:27:25 xilinx-kv260-starterkit-20222 kernel: [drm] ERT_EXEC_WRITE is obsoleted, use ERT_START_KEY_VAL
WARNING: Logging before InitGoogleLogging() is written to STDERR
I1119 09:27:25.233786  2201 test_dpu_runner_mt.cpp:474] shuffle results for batch...
I1119 09:27:25.234745  2201 performance_test.hpp:73] 0% ...
I1119 09:27:31.234948  2201 performance_test.hpp:76] 10% ...
I1119 09:27:37.235215  2201 performance_test.hpp:76] 20% ...
I1119 09:27:43.235513  2201 performance_test.hpp:76] 30% ...
I1119 09:27:49.235724  2201 performance_test.hpp:76] 40% ...
I1119 09:27:55.235949  2201 performance_test.hpp:76] 50% ...
I1119 09:28:01.236171  2201 performance_test.hpp:76] 60% ...
I1119 09:28:07.236421  2201 performance_test.hpp:76] 70% ...
I1119 09:28:13.236657  2201 performance_test.hpp:76] 80% ...
I1119 09:28:19.236877  2201 performance_test.hpp:76] 90% ...
I1119 09:28:25.237103  2201 performance_test.hpp:76] 100% ...
I1119 09:28:25.237231  2201 performance_test.hpp:79] stop and waiting for all threads terminated....
I1119 09:28:25.243063  2201 performance_test.hpp:85] thread-0 processes 5938 frames
I1119 09:28:25.243116  2201 performance_test.hpp:93] it takes 5851 us for shutdown
I1119 09:28:25.243145  2201 performance_test.hpp:94] FPS= 98.9528 number_of_frames= 5938 time= 60.0084 seconds.
I1119 09:28:25.243213  2201 performance_test.hpp:96] BYEBYE
Test PASS.
root@xilinx-kv260-starterkit-20222:~#
```

FPS: 98.95

19. Run ResNet50 demo

```
cp -R ~/Vitis-AI/examples/vai_library/samples_onnx/resnet50_pt/images/ ~/Vitis-AI/examples/vai_runtime/images/

cd ~/Vitis-AI/examples/vai_runtime/resnet50/
./resnet50 ~/resnet50.xmodel
```

```
root@xilinx-kv260-starterkit-20222:~/Vitis-AI/examples/vai_runtime/resnet50# ./resnet50 ~/resnet50.xmodel
WARNING: Logging before InitGoogleLogging() is written to STDERR
I1119 09:58:56.712509  4104 main.cc:292] create running for subgraph: subgraph_quant_avg_pool_fix

Image : 036.JPEG
top[0] prob = 0.822969  name = kite
top[1] prob = 0.111377  name = coucal
top[2] prob = 0.019354  name = spoonbill
top[3] prob = 0.015073  name = goose
top[4] prob = 0.007120  name = vulture
terminate called after throwing an instance of 'cv::Exception'
  what():  OpenCV(4.5.2) /usr/src/debug/opencv/4.5.2-r0/git/modules/highgui/src/window_gtk.cpp:624: error: (-2:Unspecified error) Can't initialize GTK backe
nd in function 'cvInitSystem'

Aborted
root@xilinx-kv260-starterkit-20222:~/Vitis-AI/examples/vai_runtime/resnet50#
```

Vivado Flow for KV260 (Vivado v2022.2 / Vitis-AI v3.0 / DPU v4.1)

This step-by-step example starts with building a Vivado v2022.2 project for KV260. By utilising current material/scripts in the Vitis-AI GitHub, configure, build and export an XSA file. Next steps include building a PetaLinux image based on the XSA file and creating an overlay. Lastly, boot and load the custom overlay, in order to test and verify the operation of configured DPU.

DPU Configuration Target:

- DPU Clock: 325MHz
- DSP Clock: 650MHz
- DPU Variant: B4096
- DPU Cores: 1
- UltraRAM: 50/64

Prerequisites:

- Vivado 2022.2 (Windows or Linux OS)

- Petalinux 2022.2 (Linux OS)
- KV260 Starter Kit BSP v2022.2 from PetaLinux Download page (link)
- DPUCZDX8G_VAI_v3.0 (v4.1) source files from Vitis-AI GitHub (link)

1. Configure Vivado build script with custom settings

Download DPUCZDX8G from Vitis-AI/dpu at v3.0 · Xilinx/Vitis-AI · GitHub or via the Terminal to a known location, and untar:

```
wget https://www.xilinx.com/bin/public/openDownload?filename=DPUCZDX8G_VAI_v3.0.tar.gz -O DPUCZDX8G_VAI_v3.0.tar.gz
tar -cvzf DPUCZDX8G_VAI_v3.0.tar.gz
```

Navigate to directory "DPUCZDX8G_VAI_v3.0/prj/Vivado/hw/scripts/" and edit trd_prj.tcl as per below:

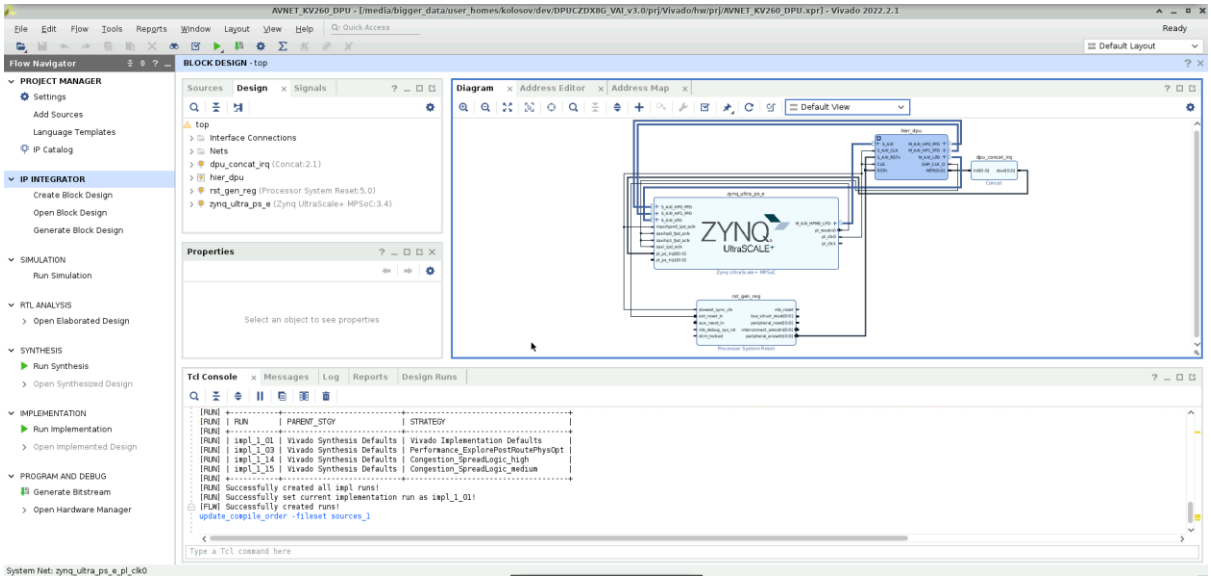
```
22 #****************************************************************
23 # set project
24 #****************************************************************
25 dict set dict_prj dict_sys prj_name                {AVNET_KV260_DPU}
26 dict set dict_prj dict_sys prj_part                {xck26-sfvc784-2LV-c}
27 dict set dict_prj dict_sys prj_board               {KV260}
28
29 #****************************************************************
30 # set bd
31 #   for bd_ooc: None for global, Hierarchical for ooc per IP
32 #****************************************************************
33 dict set dict_prj dict_sys bd_name                  top
34 dict set dict_prj dict_sys bd_ooc                   None
35
36 #****************************************************************
37 # set param
38 #****************************************************************
39 dict set dict_prj dict_param  DPU_CLK_MHz           {325}
40 dict set dict_prj dict_param  REG_CLK_MHz           {100}
41
42 #The following parameters correspond to Arch Tab of the IP GUI
43 dict set dict_prj dict_param  DPU_NUM               {1}
44 dict set dict_prj dict_param  DPU_ARCH              {4096}
45 dict set dict_prj dict_param  DPU_RAM_USAGE         {low}
46 dict set dict_prj dict_param  DPU_CHN_AUG_ENA       {1}
47 dict set dict_prj dict_param  DPU_SAVE_ARGMAX_ENA   {1}
48 dict set dict_prj dict_param  DPU_CONV_RELU_TYPE    {3}
49 dict set dict_prj dict_param  DPU_ALU_PARALLEL_USER {4}
50 dict set dict_prj dict_param  DPU_ALU_LEAKYRELU     {0}
51 dict set dict_prj dict_param  DPU_SFM_NUM          {0}
52
53 #The following parameters correspond to Advanced Tab of the IP GUI
54 dict set dict_prj dict_param  DPU_SAXICLK_INDPD     {1}
55 dict set dict_prj dict_param  DPU_CLK_GATING_ENA    {1}
56 dict set dict_prj dict_param  DPU_DSP48_MAX_CASC_LEN {4}
57 dict set dict_prj dict_param  DPU_DSP48_USAGE       {high}
58 dict set dict_prj dict_param  DPU_URAM_PER_DPU      {50}
59
```

[Line 25] dict set dict_prj dict_sys prj_name {AVNET_KV260_DPU}
[Line 26] dict set dict_prj dict_sys prj_part {xck26-sfvc784-2LV-c}
[Line 27] dict set dict_prj dict_sys prj_board {KV260}
[Line 43] dict set dict_prj dict_param DPU_NUM {1}
[Line 51] dict set dict_prj dict_param DPU_SFM_NUM {0}
[Line 50] dict set dict_prj dict_param DPU_URAM_PER_DPU {50}

Save and open terminal in directory "/DPUCZDX8G_VAI_v3.0/prj/Vivado/hw/" and then drun tcl to build the Vivado Project (source envs for the right version of Vivado):

```
/opt/Xilinx/Vivado/2022.2/bin/vivado -source
scripts/trd_prj.tcl
```

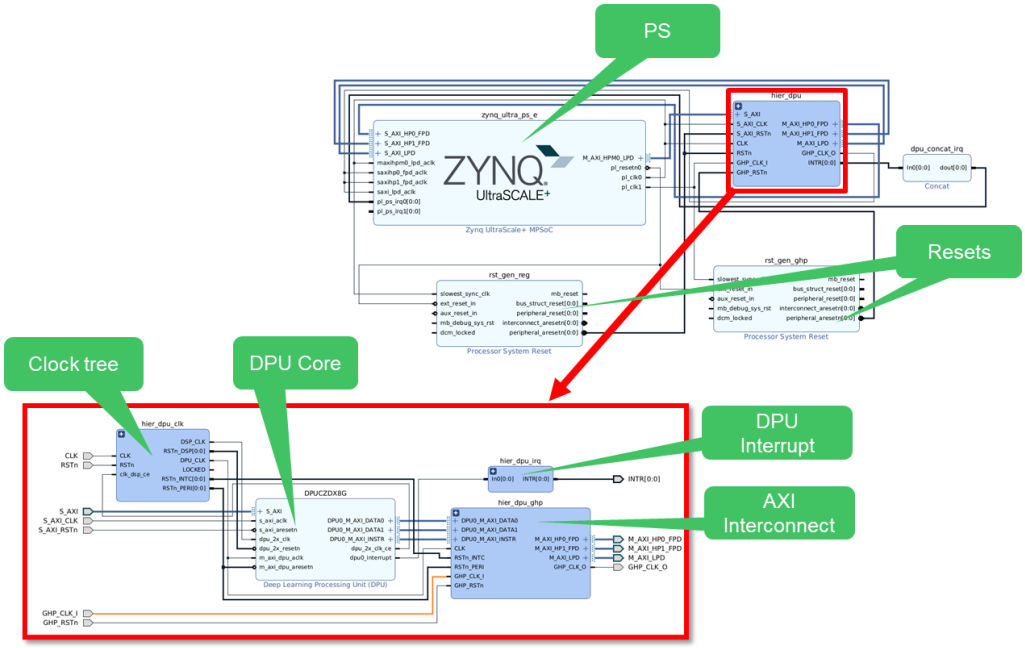
The script will create a Vivado project and open it:



2. Overview of created Vivado Project

Core Components :

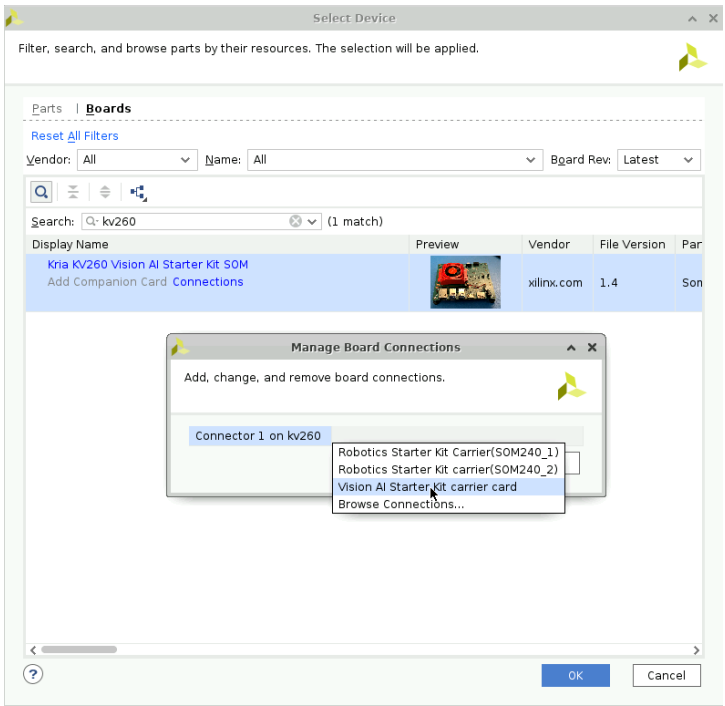
- PS
- DPU (DPUCZDX8G)
- Clocking Wizard (AXI, DPU, 2x_DPU)
- Processor System Resets
- AXI Interconnect



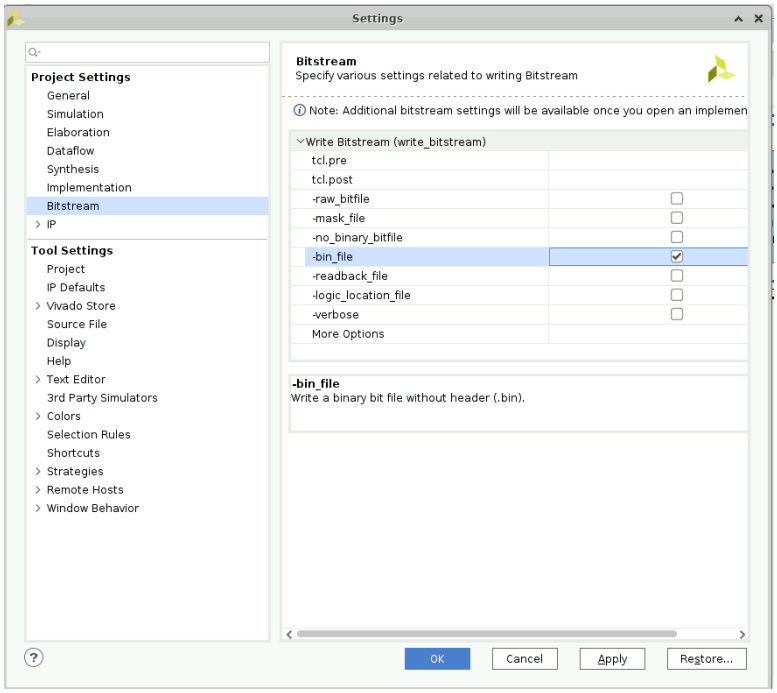
3. Modify Vivado Project

There are a few modifications to the project to do before we compile for KV260 Starter Kit.

From Flow Navigator on the left, open Settings. Under Project settings → General, choose Project Device as per below, Kria KV260 Vision AI Starter Kit SOM with adding the right connector card.

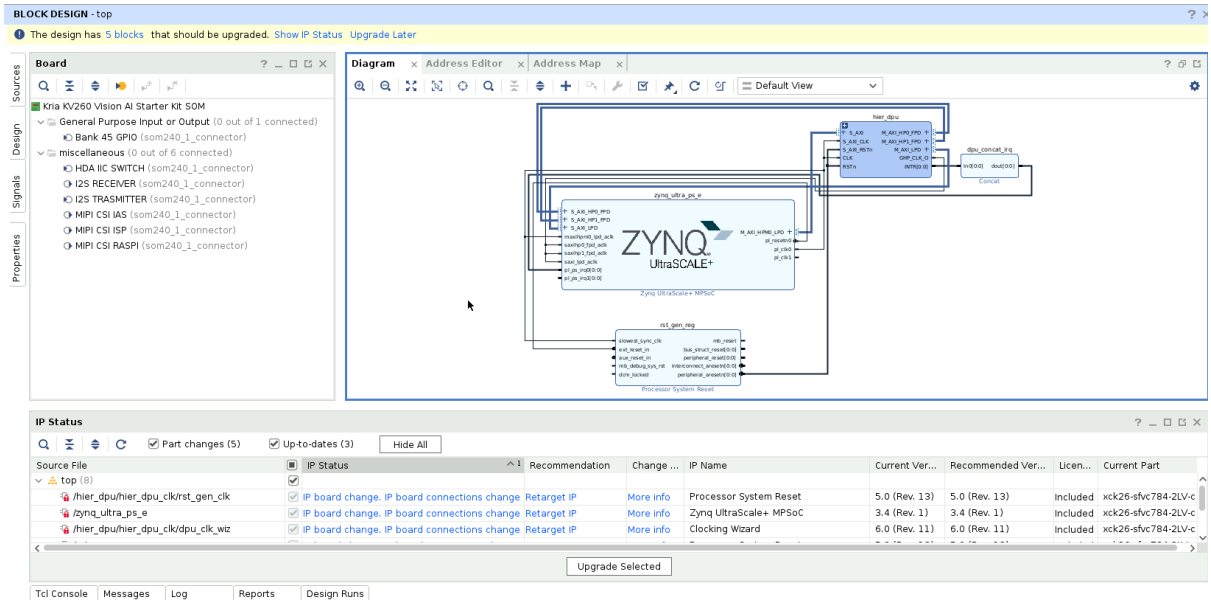


Then, under Project Settings → Bitstream, enable -bin_file option as per below:

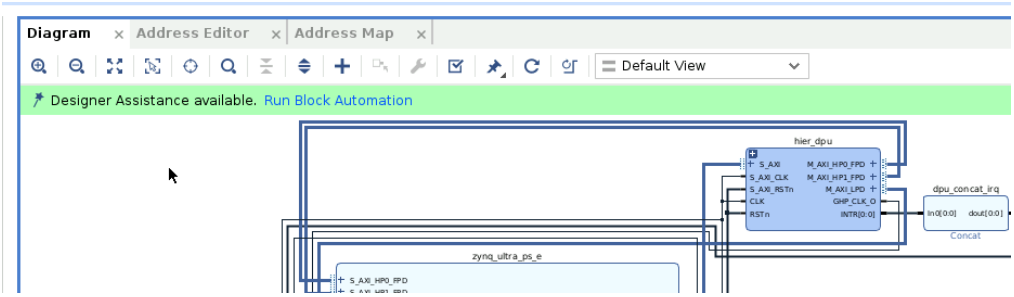


Click apply and ok.

A warning will pop regarding updating design IP blocks. After clicking Show IP Status, a new window at the bottom will open up. Click Upgrade Selected to upgrade all IPs.

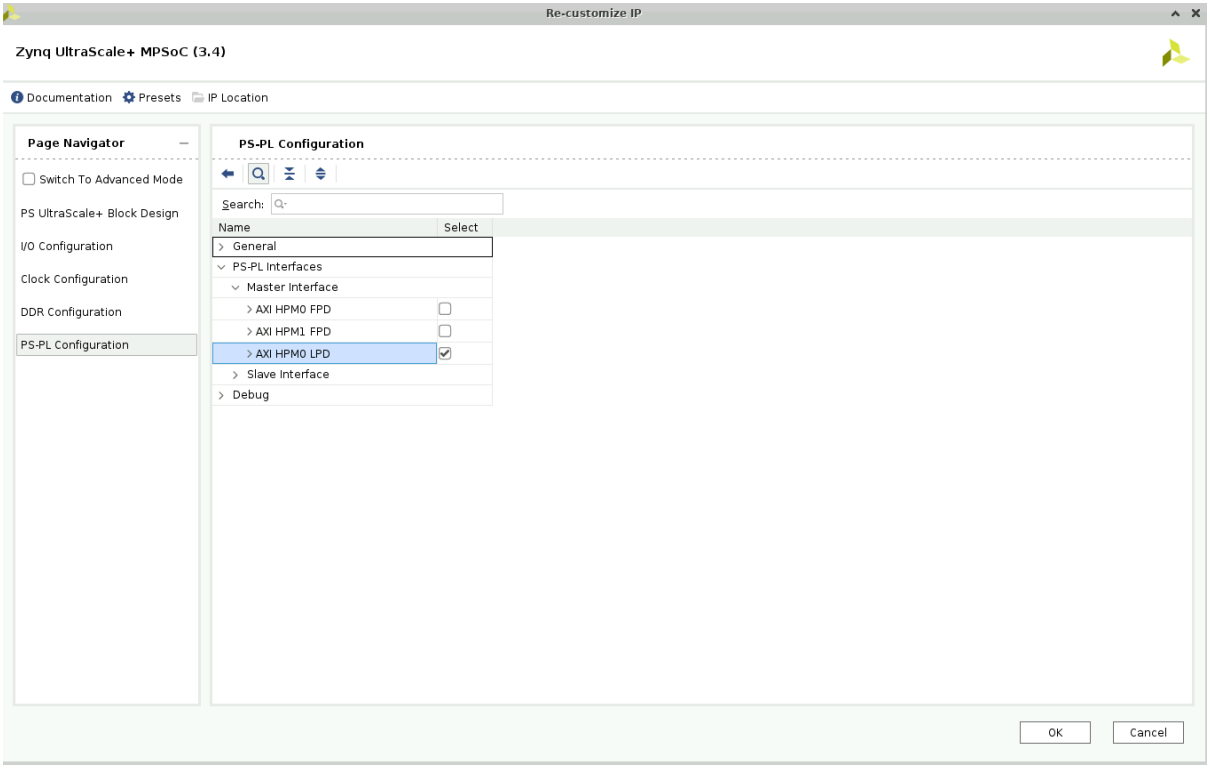


If Generate Output Product Window opens, keep defaults and press generate.
Next step is to Run block automation, which we will apply specific board presets



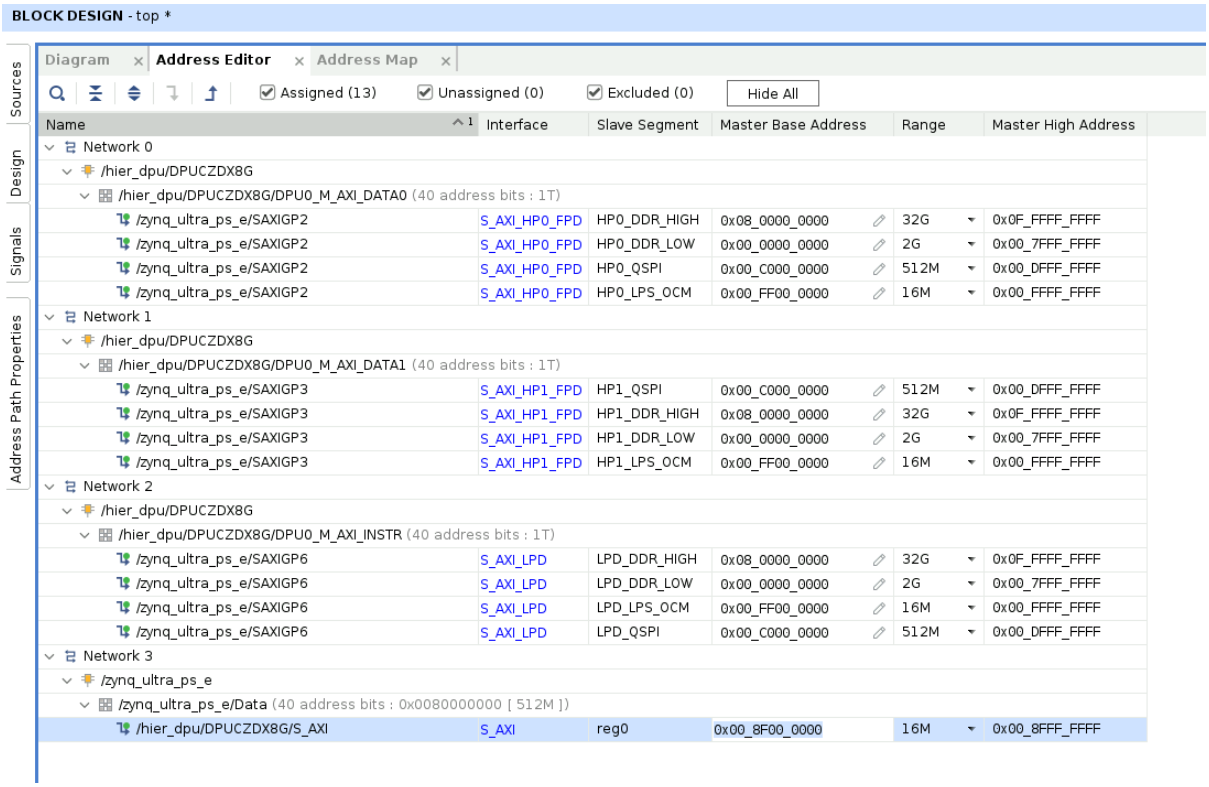
After applying board preset, design will change slightly and you will get some critical warnings where we will resolve in the following steps.

Open Zynq Ultrascale+ MPSOC block and navigate to PS-PL Configuration → PS-PL Interfaces → Master Interfaces and disable AXI HPM0 FPD and AXI HP1 FPD, while enabling AXI HPM0 LPD.

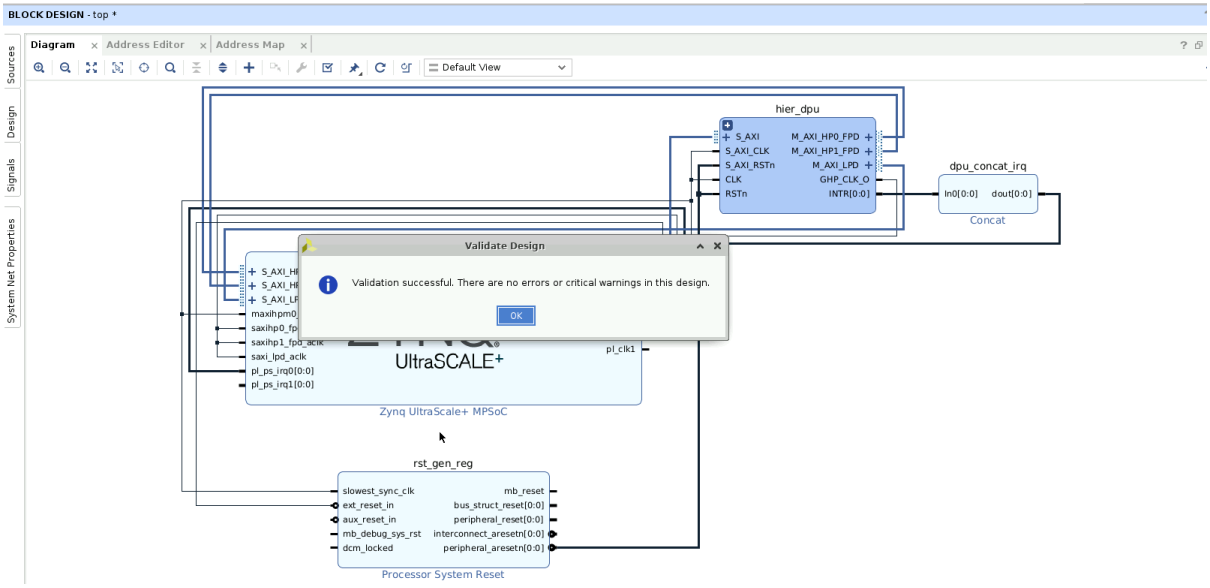


Then make the following connections:
M_AXI_HPM0_LPD (zynq_ultra_ps_e) → S_AXI (hier_dpu)
maxihpm0_lpd_aclk (zynq_ultra_ps_e) → pl_clk0 (zynq_ultra_ps_e)

In address editor tab, there will be unassigned address for the slave AXI bus.
Make automatic assignment and manually modify Master base Address to: 0x00_8F00_0000



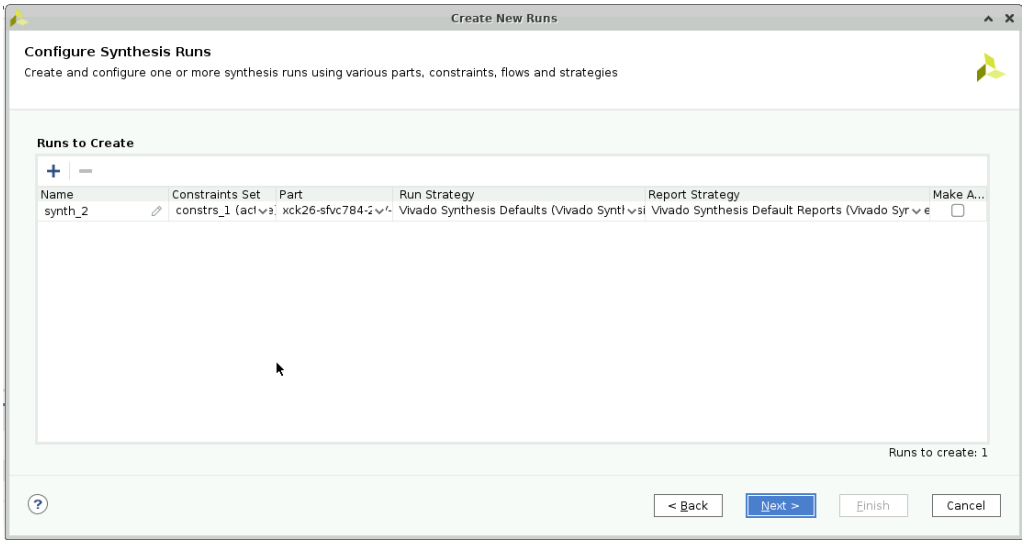
Save (Control+S) and Validate (fn+F6) Design.



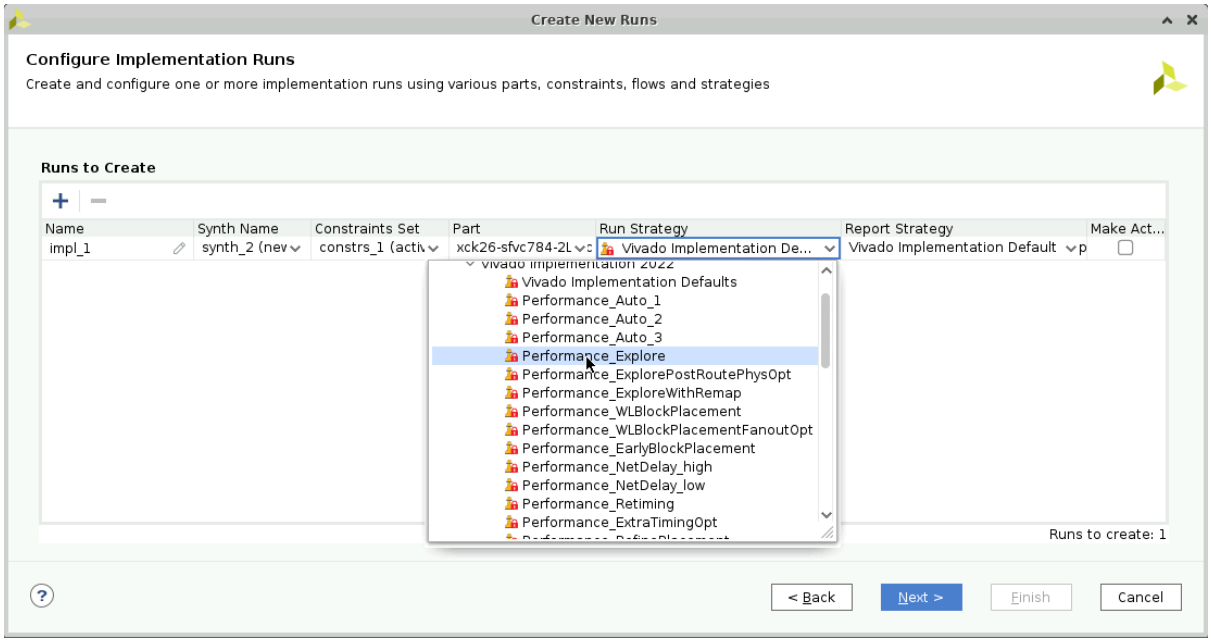
4. Compile

First we will need to update implementation strategy, so we wont get timing failures. In tab Flows, open Create Runs… and choose both.

Leave Synthesis as defaults:



For implementation strategy, choose Performance Explore and tick Make Active checkbox.

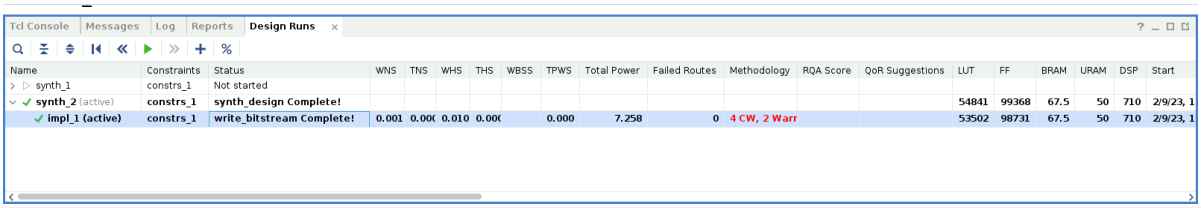


In the last section, you can leave defaults, which will launch the run (synthesis + implementation).

When finished (around 60 minutes), select Generate Bitstream option. It is not required to open implemented design, unless you want to explore various reports.

5. Vivado Project Overview

As seen in picture below, timing has been passed. You are able to explore various reports regarding power consumption, resource utilisation, etc.



To move to the next stage 2 important files are required.

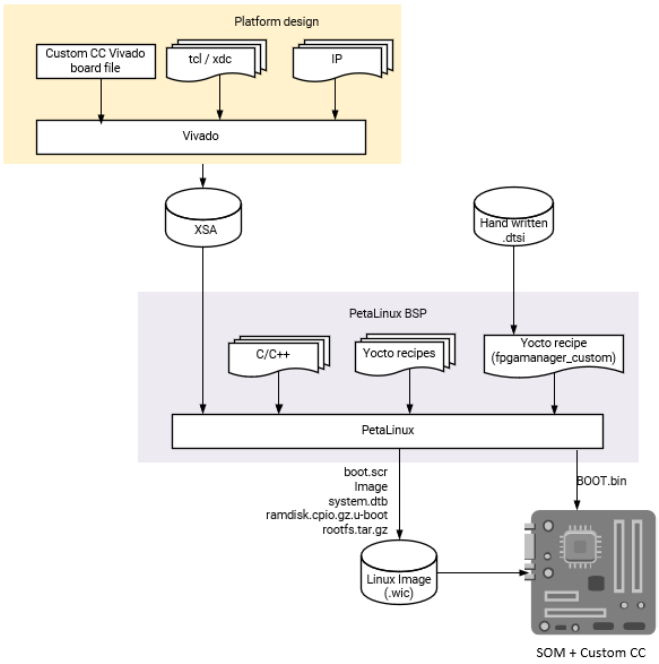
1st it is required to generate an XSA file, which will be need for next steps in building petalinux project. You can generate one by opening tab File → Export →Export Hardware. In the output setting, select “include bitstream” and leave default names.
XSA file is found in: DPUCZDX8G_VAI_v3.0/prj/Vivado/hw/prj/top_wrapper.xsa

2nd is the DPU architecture fingerprint, which will be required by VitisAI if we are to compile any models for our customer hardware architecture. This is found in the following directory:

DPUCZDX8G_VAI_v3.0/prj/Vivado/hw/srcs/top/ip/top_DPUCZDX8G_0/arch.json

Which for our use case is: **{"fingerprint":"0x101000056010407"}**

6. Create PetaLinux Project



Requirements:

- XSA file (DPUCZDX8G_VAI_v3.0/prj/Vivado/hw/prj/top_wrapper.xsa)
- KV260 Starter Kit BSP v2022.2 from PetaLinux Download page (link)

With VitisAI v3.0.0 (v2022.2 tools), a bash script is provided for helping build and configure petalinux project, which is found in DPUCZDX8G_VAI_v3.0/prj/Vivado/sw/helper_build_bsp.sh

An updated version script has been provided for reference, but we will do go through the flow step by step.

1. Navigate to directory DPUCZDX8G_VAI_v3.0/prj/Vivado/sw/ in the terminal
2. Source Petalinux tools

```
source /opt/Xilinx/petalinux/2022.2/settings.sh
```

3. Create PetaLinux project based on BSP
   Move xilinx-kv260-starterkit-v2022.2-10141622.bsp to this folder

```
prj_name=kv260_dpu
bsp=xilinx-kv260-starterkit-v2022.2-10141622.bsp
petalinux-create -t project -s $bsp -n $prj_name
```

4. Update project with custom XSA

```
cd $prj_name
xsadir=../../hw/pre-built
petalinux-config --get-hw-description=${xsadir} --
silentconfig
```

7. Configure PetaLinux settings
   1. **Copy recommended recipes into current directory**

```
recipesdir=../meta-vitis/
cp -arf ${recipesdir}/recipes-apps project-spec/meta-user/
cp -arf ${recipesdir}/recipes-core project-spec/meta-user/
cp -arf ${recipesdir}/recipes-vitis-ai project-spec/meta-
user/
```

   2. **Append packages to petalinux BSP**

```
echo "IMAGE_INSTALL:append=\" vitis-ai-library \""   >> project-
spec/meta-user/conf/petalinuxbsp.conf
```

   3. **Petalinux kernel Configuration**

- Enable DPU driver
- Enable NFS server (to remove some errors during boot)

These can be achieved by running the cmd “petalinux-config -c kernel” and applying settings manually or updating the recipes as per below cmd command:

```
cp -arf ${recipesdir}/recipes-kernel project-spec/meta-user/

echo -e 'CONFIG_NFSD=y
# CONFIG_NFSD_V3 is not set
# CONFIG_NFSD_V4 is not set' >> project-spec/meta-
user/recipes-kernel/linux/linux-xlnx/bsp.cfg
```

4. **PetaLinux rootfs configuration**

- Disable zocl and xrt
- Enable auto-login
- Enable package management (for install rootfs packages via dnf)

These can be achieved by running the cmd “petalinux-config -c rootfs” and applying settings manually or updating the recipes as per below cmd command:

```
sed -i 's/CONFIG_xrt=y/\# CONFIG_xrt is not set/' project-
spec/configs/rootfs_config

sed -i 's/CONFIG_xrt-dev=y/\# CONFIG_xrt-dev is not set/'
project-spec/configs/rootfs_config

sed -i 's/CONFIG_zocl=y/\# CONFIG_zocl is not set/' project-
spec/configs/rootfs_config

sed -i '/# CONFIG_auto-login is not set/c CONFIG_auto-
login\=y' project-spec/configs/rootfs_config.

sed -i '/# CONFIG_imagefeature-package-management is not
set/c CONFIG_imagefeature-package-management\=y' project-
spec/configs/rootfs_config
```

5. **Other configurations**

- Enable FPGA Manager
- Disable TFTPBOOT
- Update host name

These can be achieved by running the cmd “petalinux-config” and applying settings manually or updating the recipes as per below cmd command:

```
sed -i '/# CONFIG_SUBSYSTEM_FPGA_MANAGER is not set/c
CONFIG_SUBSYSTEM_FPGA_MANAGER\=y' project-spec/configs/config

sed -i 's/CONFIG_SUBSYSTEM_COPY_TO_TFTPBOOT=y/\# #
CONFIG_SUBSYSTEM_COPY_TO_TFTPBOOT is not set/' project-
spec/configs/config

sed -i '/CONFIG_SUBSYSTEM_HOSTNAME=/c
CONFIG_SUBSYSTEM_HOSTNAME\="avnet-kv260-dpu"' project-
spec/configs/config
```

8. Build Petalinux Project

```
petalinux-config -c kernel --silentconfig
petalinux-config -c rootfs --silentconfig
petalinux-build
```

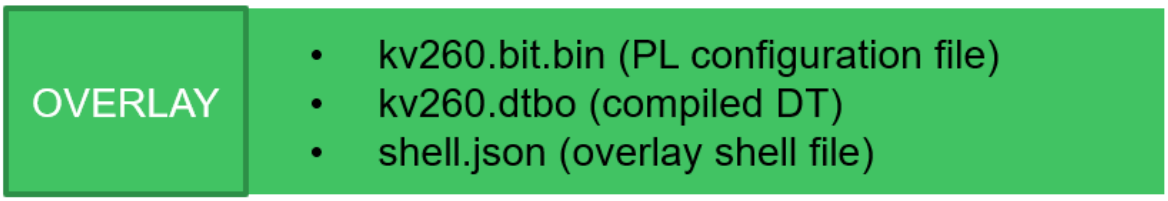
9. Create SD card

```
petalinux-package --wic --images-dir images/linux/ --
bootfiles "ramdisk.cpio.gz.u-
boot,boot.scr,Image,system.dtb,system-zynqmp-sck-kv-g-
revB.dtb" --disk-name "mmcblk1" --wic-extra-args "-c gzip"
```

Under directory DPUCZDX8G_VAI_v3.0/prj/Vivado/hw/prj/kv260_dpu/images/linux a file named "petalinux-sdimage.wic.gz" is present. This can be used to flash an SDCARD using programs like BalenaEtcher.

10. Create overlay

Next step is to create an overlay that can be loaded during runtime, with our custom hardware image, which contains the customised DPU.



################## Prepare files

```
mkdir workspace_overlay
cp ../../hw/prj/top_wrapper.xsa ./workspace_overlay
cd workspace_overlay
source /opt/Xilinx/Vitis/2022.2/settings64.sh
```

################## CREATE DT

```
xsct

hsi::open_hw_design top_wrapper.xsa

createdts -hw top_wrapper.xsa -zocl -platform-name
kv260_hw_platform -git-branch xlnx_rel_v2022.2 -overlay -
compile -out ./KV260_dt

exit
```

################## COMPILE DT

```
dtc -@ -O dtb -o ./kv260.dtbo
./KV260_dt/KV260_dt/kv260_hw_platform/psu_cortexa53_0/device_
tree_domain/bsp/pl.dtsi
```

################## PREPARE REQUIRED FILES

```
echo '{ "shell_type" : "XRT_FLAT", "num_slots": "1" }' >
shell.json

mkdir KV260_DPU
```

```
cp shell.json ./KV260_DPU
cp top_wrapper.bit ./KV260_DPU/kv260_dpu.bit.bin
cp kv260.dtbo ./KV260_DPU/kv260_dpu.dtbo
```

11. KV260 Boot

Required files:

- SD Card image: **petalinux-sdimage.wic.gz**
  Found in directory: DPUCZDX8G_VAI_v3.0/prj/Vivado/hw/prj/kv260_dpu/images/linux
- Overlay folder: **KV260_DPU**
  Found in directory: DPUCZDX8G_VAI_v3.0/prj/Vivado/sw/kv260_dpu/workspace_overlay

1. Flash SD Card image with balenaEtcher - Flash OS images to SD cards & USB drives 9
2. Boot KV260 Starter Kit
3. Copy KV260_DPU folder to directory /lib/firmware/xilinx/

```
sudo mv ./KV260_DPU /lib/firmware/xilinx/
```

4. Check available apps and make sure KV260_DPU is present

```
sudo xmutil listapps
```

5. Unload current app

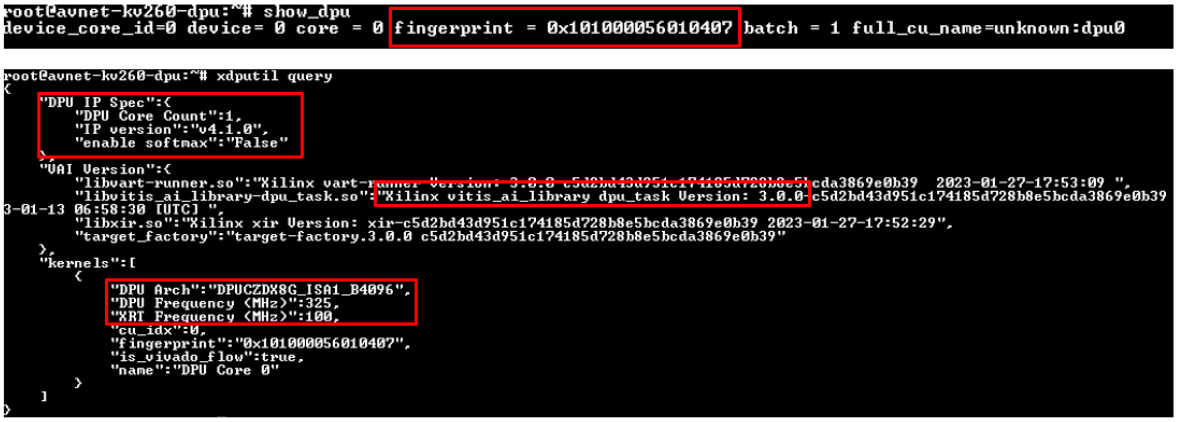
```
sudo xmutil unloadapp
```

6. Load custom app

```
sudo xmutil loadapp KV260_DPU
```

7. Check DPU settings

```
show_dpu
xdputil query
```



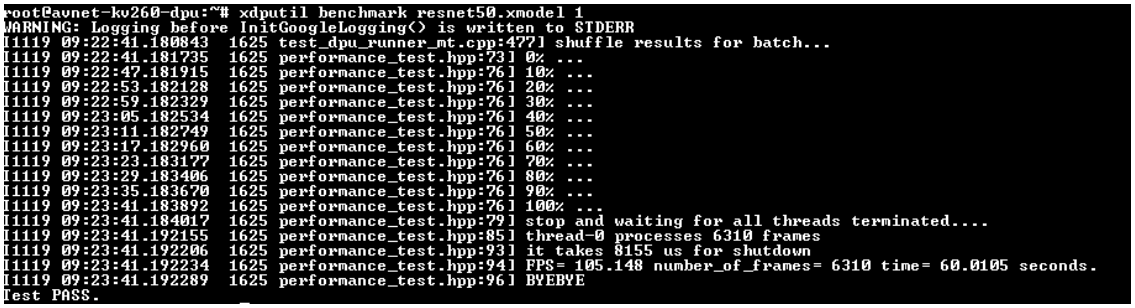
8. Benchmark previous resnet model

Going back to vitis-ai environment, you can recompile same model, but targeting a different architecture, as now the fingerprint id has changed due to changes we made to the DPU. Run again the compilation of the model with the updated ID:

```
vai_c_tensorflow2- \
--model quantized.h5 \
--arch arch_kv260.json \
--output_dir ./ \
--net_name resnet50
```

Copy the updated model to KV260, and quickly benchmark its performance in terms of throughout (FPS). Note, you might need to load custom overlay again, if you have restarted the board:

```
xdputil benchmark resnet50.xmodel 1
```



Comparing results, there was an increase of ~6% of FPS performance by increasing clock rate by ~8% (25MHz).

DPU Clock / DSP Clock (MHz)	FPS
300 / 600	98.95
325 / 650	105.1

Resources

- GitHub - Xilinx/Vitis-AI: Vitis AI is Xilinx's development stack for AI inference on Xilinx hardware platforms, including both edge devices and Alveo cards.
- Vitis AI — Vitis™ AI 3.0 documentation (xilinx.github.io)
- Vitis AI Overview • Vitis AI User Guide (UG1414) • Reader • AMD Adaptive Computing Documentation Portal (xilinx.com)
- Introduction • Vitis AI Library User Guide (UG1354) • Reader • AMD Adaptive Computing Documentation Portal (xilinx.com)
- Introduction • DPUCZDX8G for Zynq UltraScale+ MPSoCs Product Guide (PG338) • Reader • AMD Adaptive Computing Documentation Portal (xilinx.com)