DAC-SDC Notes

Contest Introduction

DAC SDC

The DAC System Design Contest focuses on low-power object detection on an embedded FPGA system. Contestants will receive a training dataset provided by DJI, and a hidden dataset will be used to evaluate the performance of the designs in terms of accuracy and power.

https://byuccl.github.io/dac_sdc_2022/

Contesters aim to design a low-power object detection system based on the Ultra96 v2 FPGA board for a DJI dataset.

Contest Evaluation

The design is solely evaluated by the total energy consumption. However, there are minimum thresholds for accuracy and throughput. Accuracy is measured using IoU (Intersection over Union). The minimum IoU should be 0.7. The throughput should achieve at least 30 FPS. The score calculates as follows:
Score = 10^2 / log2(Energy)×Max(ReLU([1-5×ReLU(0.7-IoU)]), 0.1) × ReLU([1-ReLU(1 - FPS / 30)])

Design ideas

The idea adopted in the previous award-winning works is to design and optimize an AI model, then instantiate the model into a hardware-accelerated IP by high-level synthesis. This method can realize the Iteration Interval=1 and top-level pipeline, naturally delivering high throughput and low power consumption.

But this method is a bit difficult for us, so we decided to use DPU (Deep learning Processing Unit) to simplify the design. After the construction of the DPU-PYNQ platform, we could focus on the AI model. When it is trained, it will be compiled into a DPU executable file. That is a Vitis-AI based workflow for deploying AI on FPGA. In the future, we plan to use multiple DPUs to carry out DPU scheduling research and reconfigurable research, which create the platform for our follow-up research.

Design process

1. Burn the latest Ultra96 V2 PYNQ image

   PYNQ - Python productivity for Zynq

   for details on installing PYNQ for use with Alveo and AWS-F1. If you have a Zynq board, you need a PYNQ SD card image to get started. You can download a pre-compiled PYNQ image from the table below. If an image is not available for your board, you can build your own SD card

   http://www.pynq.io/board.html

2. Install DPU-PYNQ

   https://github.com/Xilinx/DPU-PYNQ

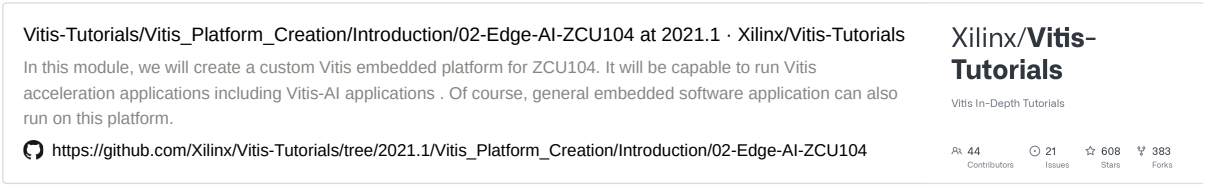
3. Train YOLOV4-tiny and deploy on board
   See below for details

Experimental results

In the end, we achieved about 30fps on ultra96, but this did not account for the cost of data post-processing. In addition, we built a petalinux system for ZCU106 without the PYNQ framework and deployed two B4096 DPUs. Yolov4 tiny model achieves 233fps and 65.7% mAP on ZCU106.

How to deploy DPUs and run NNs on ZCU106

1. Custom Embedded Platform Creation for ZCU106



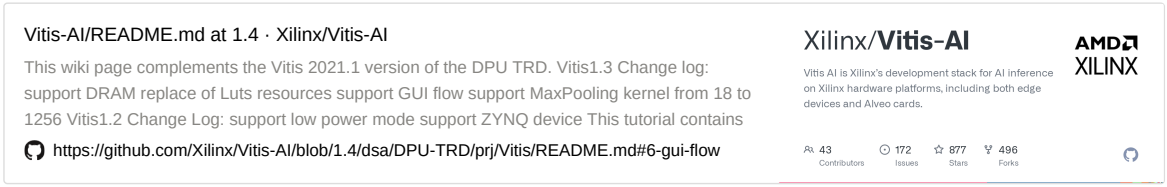
- Step 1: Create the Vivado Hardware Design and Generate XSA
  - Please choose the appropriate version of Vitis and Vivado according to the Tutorial version, we chose 2021.1
  - Please install **Y2k22 Patch** and get License
- Step 2: Create the Software Components with PetaLinux
  - Because the Petalinux development cycle is not same as vitis, please choose the 2021.2 petalinux
  - ###Modify DTG Settings->MACHINE_NAME to **zcu106-reva**
- Step 3: Create the Vitis Platform
- Step 4: Test the Platform (Vadd)

Note: Xilinx provides Embedded Base Platforms for ZCU102 and ZCU104, which are the starting point for Vitis accelerated applications. After completing the above, we get the base platform for ZCU106. Of course, our platform is simpler than Xilinx, we could refer to Xilinx github to explor the office platform source.

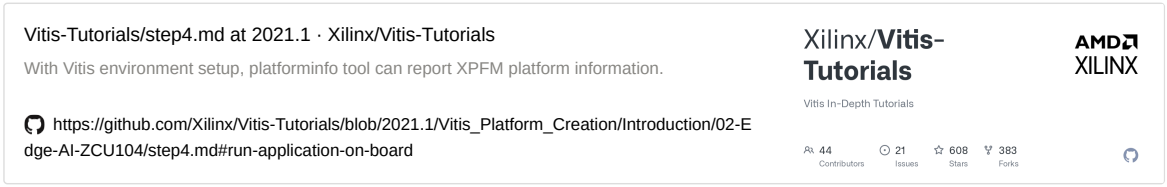
#####For DPU Creation, please refer DPU VItis GUI flow. This tutorial here misses some steps.

2. DPU Creation for ZCU106

- DPU VItis GUI flow



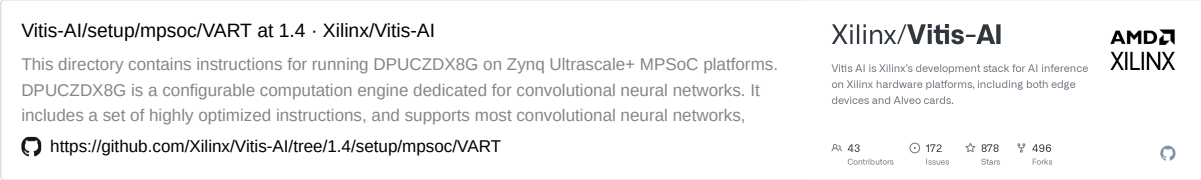
  - If Vitis IDE Library fails to download DPU Example, please store /Vitis-AI/dsa/DPU-TRD in ./xilinx/vitis/2021.1/, vitis will recognize the json file
- Resize ext4 partition and Run App



Note: The hardware fingerprint description file arch.json will be generated during the dpu compilation process. Please save it. It will be used to compile the xmodel that can run on the currently configured DPU.

#####The DPU configured by default can run model zoo zcu102 xmodel, so do not change configs.

3. VART setting and Model Zoo demo running

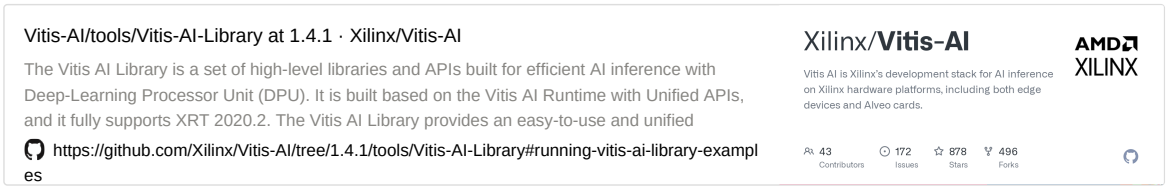


1. Setup host cross-compiler environment, which will be used to compile the executable file for zynqmp arm.
2. Setup the Target

   `dpu_sw_optimize` dnf install packagegroup-petalinux-vitisai `target_vart_setup.sh`
3. Run VART demo

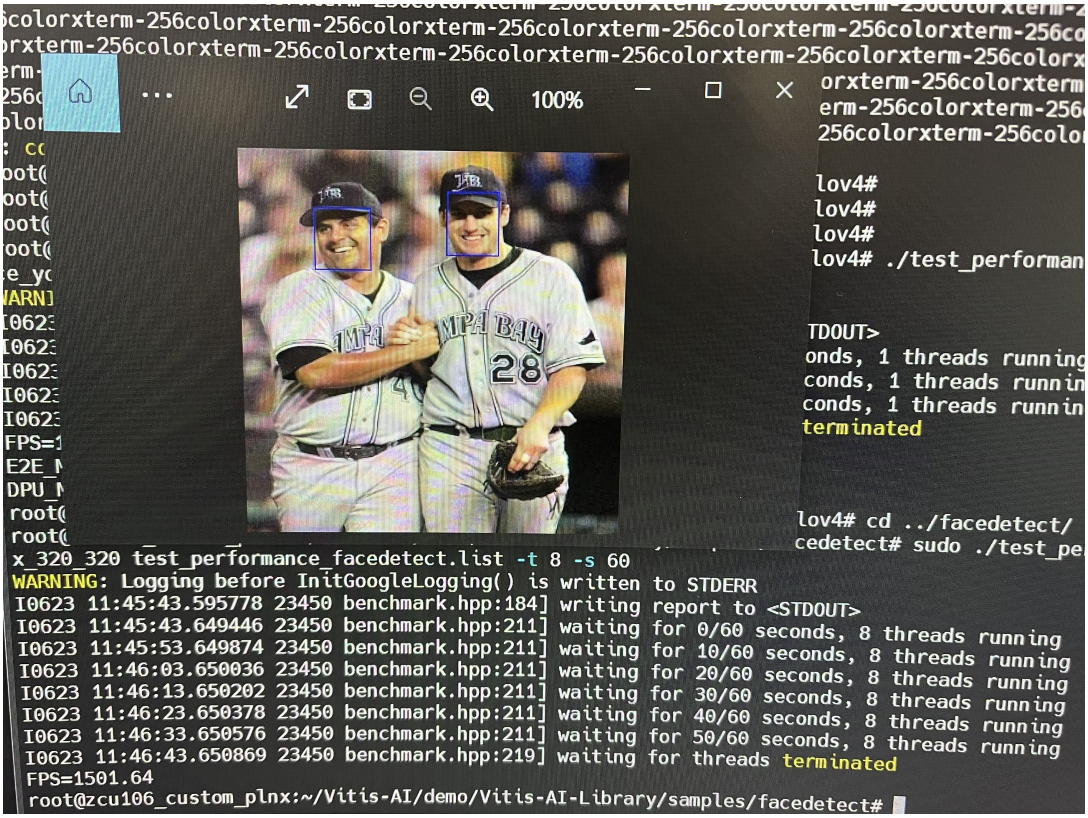
   ###work at host: <PATH to vitis ai>demo/VART

   1. Download VART demo img&video and copy to <PATH to vitis ai>demo/VART
   2. Pick a demo, Source cross-compiler environment, run build.sh, get executable file
   3. copy all files to target
   4. Refer to the readme to download the corresponding zcu102 xmodel at Xilinx AI Model Zoo and store it in the appropriate directory of the target
   5. run test application ###work at target: ~/demo/vitis-ai/
4. Run Vitis ai Library demo



An example: `facedetect densebox_320_320`

- Download vitis ai library images and video, untar, copy to `Vitis-AI/demo/Vitis-AI-Library`
- Enter the directory of `/Vitis-AI/demo/Vitis-AI-Library/samples/facedetect`
  - source cross-compiler environment (env setup before), run buid.sh, then we will get executable file
  - refer readme, the Valid model are densebox_320_320 and densebox_640_360
  - copy to target same directory
- Enter the directory of `/Vitis-AI/models/AI-Model-Zoo/model-list`
  - find densebox_320_320
  - open model.yaml, down xmodel for zcu102 & zcu104 & kv260
  - copy to `/usr/share/vitis-ai-library/models`
- cd to target `/Vitis-AI/demo/Vitis-AI-Library/samples/facedetect`
  - 1. Run the image test example `./test_jpeg_facedetect densebox_320_320 sample_facedetect.jpg`



4. Run YOLOV4-tiny on ZCU106

- Through the previous steps, we have completed the work on the hardware side and tested that the DPU can work normally. Next, we will train our own model and run it on ZCU106.
- The workflow of the Vitis-AI side is Train—(Convert)—quantize—compile, please flow the Pytorch flow tutorial to understand the workflow.

> Vitis-AI-Tutorials/README.md at 1.4 · Xilinx/Vitis-AI-Tutorials
> Tested on ZCU102, Alveo™ U50 Tools used: PyTorch 1.4 & Vitis AI™ 1.4 Dataset: MNIST handwritten digits Network: Custom CNN This tutorial introduces the user to the Vitis AI TensorFlow design process and describes how to go from a python description of the network
> https://github.com/Xilinx/Vitis-AI-Tutorials/blob/1.4/Design_Tutorials/09-mnist_pyt/README.md
> Xilinx/Vitis-AI-Tutorials
> AMD XILINX
> 2 Contributors · 42 Issues · 204 Stars · 99 Forks

- Next, we use DJI dataset to train our own yolov4-tiny model. please refer to the tutorial

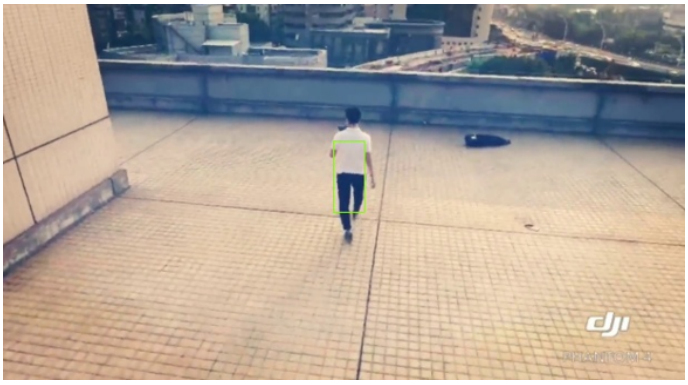
> Vitis-AI-Tutorials/Design_Tutorials/07-yolov4-tutorial at 1.4 · Xilinx/Vitis-AI-Tutorials
> YoLov4 The following tutorials cover training, evaluating, converting, quantizing, compiling, and deploying YoLov4 on the Xilinx® ZCU102 and ZCU104 evaluation boards. We modified the official YoLov4 model configuration to make it compatible with the Xilinx Zynq® UltraScale+™
> https://github.com/Xilinx/Vitis-AI-Tutorials/tree/1.4/Design_Tutorials/07-yolov4-tutorial
> Xilinx/Vitis-AI-Tutorials
> AMD XILINX
> 2 Contributors · 42 Issues · 204 Stars · 98 Forks

  - Our aim is to train a YOLOV4-tiny using the DJI dataset (provided by DAC 2022 contest)
  - preprocess DJI dataset
  - group does not work on dpu, please delete it in cfg file

```
105  [route]
106  layers=-1
107  # groups=2
108  # group_id=1
109
```

- train convert quantize compile
- modify prototxt (07-yolov4-vitis-ai-tutorial/dpu_yolov4_voc) according to the model structure
- run yolov4-tiny as the vitis-ai-library demo methods

- Experimental results
  - image test
    ./test_jpeg_yolov4 dpu_yolov4-tiny_mydata sample_person7.jpg



  - performance test
    ./test_performance_yolov4 dpu_yolov4-tiny_mydata test_mydata_lite.list -t4 -s30

```
root@zcu106_custom_plnx:~/Vitis-AI/demo/Vitis-AI-Library/samples/yolov4_mydata# ./test_performance_yolov4 dpu_yolov
4-tiny_mydata test_mydata_lite.list -t4 -s30
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0628 02:01:47.995507  4213 benchmark.hpp:184] writing report to <STDOUT>
I0628 02:01:48.076740  4213 benchmark.hpp:211] waiting for 0/30 seconds, 4 threads running
I0628 02:01:58.076936  4213 benchmark.hpp:211] waiting for 10/30 seconds, 4 threads running
I0628 02:02:08.077173  4213 benchmark.hpp:211] waiting for 20/30 seconds, 4 threads running
I0628 02:02:18.077461  4213 benchmark.hpp:219] waiting for threads terminated
FPS=233.894
```

  - accuracy test
    - get testing data result ./test_accuracy_yolov4-tiny_mydata mydata_lite.list test.txt
      - modify the dataset category name, xmodle name of test_accuracy.cpp of yolov3,
    - compare mAP between ground truth and testing data result
      - Generate a suitable ground_truth file based on description of script/evaluation.py

```
(yolov4) mavo@portlab-AB105-02:~/文档/Vitis-AI-Tutorials/Design_Tutorials/07-yolov4-tutorial$ python  ./scripts/evaluation.py -mode detect
ion          -gt_file ./yolov4-tiny_mydata_dr2tf/pred/new_mydata_gt.txt         -result_file ./yolov4-tiny_mydata_dr2tf/pred/yolov4_mydata
_pred.txt          -detection_iou 0.05 -detection_thresh 0.005;
evaluate 9353 images
car AP: 0.8656485659722262
riding AP: 0.9074693815716869
whale AP: 0.8097801585740227
person AP: 0.8137301041843397
truck AP: 0.07766155413214236
boat AP: 0.9995535236623723
group AP: 0.2203742203742204
drone AP: 0.7439432542420752
wakeboard AP: 0.17272727272727273
building AP: 0.3703703703703703
horseride AP: 0.9565217391304348
paraglider AP: 0.9503451575266628
mAP: 0.6573437752056522
```

  - power, energy test
    according to ZCU106 user guide, we need a MAXPower Tool USB cable to monitor current and voltage of PMBus.

Future work

There are many areas worth improving about the YOLO-tiny model.

1. choice an input size which adapts DJI dataset;
2. cluster the dataset then change anchor;
3. DAC score only focuses on IOU, which can be a direction to optimize the model.

In addition, we actually deployed 2 B4096 DPUs on the board, adjusting the DPU size and number to achieve multi-DPU scheduling is also an interesting exploration.