# DAC 2023 Design Contest

For full contest details, please see the [2023 DAC System Design Contest](https://dac-sdc.github.io/2023/) page.

For general questions regarding this contest, please use the Slack workspace: <https://join.slack.com/t/dac-sdc/shared_invite/zt-1rrtmgjad-NCYE2leBfOw8xTOp52KR7w>

## ABOUT

Goal : Object detection and classification on a FPGA system

### Requirements : bits (32/64)

#### ACCURACY

Accuracy is measured using the [F1-score](https://en.wikipedia.org/wiki/F-score), where:

F1-score = (2 \* precision \* recall) / (precision + recall)

precision = global\_true\_positives / (global\_true\_positives + global\_false\_positives)

recall = global\_true\_positives / (global\_true\_positives + global\_false\_negatives)

True positives are when a reported object matches the type and location (IoU > 0.5) of an object from the golden data.

The minimum accuracy should be **TBD**, otherwise a penalty is applied.

#### THROUGHPUT

The design should achieve at least 5 FPS (**subject to change**), otherwise a penatly is applied.

#### SCORING FUNCTION

The score for a team is calculated as follows **(Subject to change)**:

team score = F1-score^2 \* fps

High level implementation (MATLAB/SystemC)

Profile the code in Vivado HLS

* Implement the desired tasks in Vivado HLS compatible C/C++ code
* Profile the Code (Feature in Vivado HLS)
* Help us understand the data dependencies
  + which part of code is parallelizable
  + which part is computationally heavy or high latency (move to PL side)
  + more data dependency (implement in software-PS)
  + Performances vs Resource utilization trade off

Hello World

<https://www.youtube.com/watch?v=bs3bzgAZqqw>

Contest : <https://dac-sdc.github.io/2023/>

#### FPGA Platform

You should use the Ubuntu 22.04 PYNQ PYNQ image, available at <http://www.pynq.io/board.html>.

The base design framework, with additional setup instructions, is provided here: <https://github.com/dac-sdc/fpga_starter_2023>.

## Training Dataset

Link to download training dataset: <https://drive.google.com/file/d/1ceQ5y_rCReSZ26HzzCf2muDNbovjyl5k/view?usp=share_link>

The dataset contains:

* 12000 training images in the *JPEGImages* directory (a few extra images with *\_1* suffix are included such as 00009\_1.jpg, which are simply mirror images of the base image and can be ignored).
* Labelled object types and locations in the *label* directory. For example, *JPEGImages/00001.jpg* will have an associated *label/00001.json* file describe object types and locations in the image.
* Object types:

| **Type** | **Name** | **Example** | **Identifying Data** |
| --- | --- | --- | --- |
| 1 | Motor Vehicle |  | Bounding Box |
| 2 | Non-motorized Vehicle |  | Bounding Box |
| 3 | Pedestrian |  | Bounding Box |
| 4 | Red Traffic Light |  | Bounding Box |
| 5 | Yellow Traffic Light |  | Bounding Box |
| 6 | Green Traffic Light |  | Bounding Box |
| 7 | Off Traffic Light |  | Bounding Box |

## Submission

You can optionally submit your design for the preliminary submissions (see [schedule](https://dac-sdc.github.io/2023/schedule/)). This allows you to check that your solutions is working on our evaluation platform.

### Requirements:

1. Submit the following files:
   * Your Jupyter notebook (\*.ipynb)
   * Hardware files (\*.bit and \*.hwh)
   * Any other files we need to run your design (ie a weights file)
   * DO NOT submit the *dac\_sdc.py* file.
2. Your notebook should be structured as explained in the example notebook provided (dac\_sdc.ipynb).
   * Leave the sys.path.append(os.path.abspath("../common")) statement in the notebook so that the official dac\_sdc.py file can be located.
3. Your notebook must run without error using the *Run All Cells* command in Jupyter.
4. Do not hardcode any file paths.
5. If you are using the v1 board, be sure to fix the power rail names before submission.
6. Place all of your files in a single zip archive and submit it.

### Final Submission

For the final submission, follow the instructions above. In addition:

1. Submit all source files for your design, in a zip archive.
2. Your design must be available, open-source, and in working condition in order to be considered for an award. You are permitted to use publicly available closed source tools and IP (Xilinx’s DPU); however, all of your work (any modifications and configurations to commercial, closed-source tools), must be accessible.

#### Submission Links

* Prelim #1: <https://forms.gle/PbgUfGFmnVgwnkT66>
* Prelim #2: <https://forms.gle/3xZauRg5GpiGGgz96>

## Design

### Previous Contest Winning Designs

*Note:* These are designs for the FPGA contest, and were for a different image detection problem/dataset than this year.

* 2022: <https://github.com/jgoeders/dac_sdc_2022_designs>
* 2021: <https://github.com/jgoeders/dac_sdc_2021_designs>
* 2020: <https://github.com/jgoeders/dac_sdc_2020_designs>
* 2019: <https://github.com/xyzxinyizhang/2019-DAC-System-Design-Contest>
* 2018: <https://github.com/xyzxinyizhang/2018-DAC-System-Design-Contest>

# PYNQ VENV Activate Updates

echo "export PYNQ\_JUPYTER\_NOTEBOOKS=${PYNQ\_JUPYTER\_NOTEBOOKS}" >> /etc/profile.d/pynq\_venv.sh

echo "export BOARD=$BOARD" >> /etc/profile.d/pynq\_venv.sh

echo "export XILINX\_XRT=/usr" >> /etc/profile.d/pynq\_venv.sh

source /etc/profile.d/pynq\_venv.sh

systemctl start jupyter.service

FINN : <https://github.com/Xilinx/finn>

### HW

Make a custom model using Xilinx model zoo or training, pruning, quantizing, and compiling a new model, please refer to the [Vitis AI documentation](https://docs.xilinx.com/r/en-US/ug1414-vitis-ai) [[8]](https://xilinx.github.io/kria-apps-docs/kv260/2022.1/build/html/docs/smartcamera/docs/customize_ai_models.html) , it is supposed to run on a dpu.

Build Vivado Design : can be used for Modifying the Vivado design and creating a new XSA [[9]](https://xilinx.github.io/kria-apps-docs/kv260/2022.1/build/html/docs/build_vivado_design.html)

PYNQ

Some error about sensor96b : <https://github.com/Avnet/Ultra96-PYNQ/tree/image_v3.0>

* + Step 1 : Create an FPGA design for a class of related applications (Vivado)
  + Export the bitstream and a C API for programming the design
  + Wrap the C API to create a python library
  + Import the bitstream and library in your python scripts and program

DPU PYNQ Setup : <https://github.com/Xilinx/DPU-PYNQ/blob/master/boards/README.md>

### SW

1. Build VItis Platform [[10]](https://xilinx.github.io/kria-apps-docs/kv260/2022.1/build/html/docs/build_vitis_platform.html)
2. Integrate Accelerator Overlay [[11]](https://xilinx.github.io/kria-apps-docs/kv260/2022.1/build/html/docs/build_accel.html)
3. Generating Custom Firmware Binaries [[12]](https://xilinx.github.io/kria-apps-docs/kv260/2022.1/build/html/docs/generating_custom_firmware.html)
4. Use Pynq overlay to communicate with DMA using python [[13]](https://github.com/Xilinx/PYNQ_Workshop/blob/master/Session_4/6_dma_tutorial.ipynb) [[14]](https://pynq.readthedocs.io/en/latest/pynq_libraries/dma.html)

VItis AI Library APIs[[15]](https://www.xilinx.com/developer/articles/introduction-to-vitis-ai-library-apis.html)

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

### Environment Setup and Installation [[16]](https://www.xilinx.com/developer/articles/part1-environment-setup-and-Installation.html)

#### Setup Target Machine

##### Setup PYNQ on your Kria KV260 board

* Setup the Ubuntu 22.04 image on your Kria KV260 board <https://www.xilinx.com/products/som/kria/kv260-vision-starter-kit/kv260-getting-started/getting-started.html>
  + *Note:* You may need to update the Boot FW in order to boot the Ubuntu 22.04 image. There are instructions here <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/1641152513/Kria+K26+SOM#Boot-FW-update-with-xmutil>. This requires first booting the Ubuntu 20.04 image, installing the Xilinx software tools with in the Ubuntu 20.04 environment, and then using the xlnx-config --xmutil <cmd> tool to install the new firmware. Follow the instructions on that page carefully.
* Follow the instructions at <https://github.com/Xilinx/Kria-PYNQ> to install the PYNQ system (includes Jupyter notebooks).

#### Setup Host Machine

Install Pytorch Docker , Vitis AI GPU docker : <https://xilinx.github.io/Vitis-AI/docs/install/install.html#build-docker-from-scripts>

[Host]$ cd Vitis-AI/board\_setup/mpsoc

[Host]$ ./host\_cross\_compiler\_setup.sh

Install Vivado Vitis

sudo apt-get install -y libstdc++6 libgtk2.0-0 dpkg-dev libtinfo5 libncurses5

1. Install CUDA . Pentalinux Xilinx
2. PYNQ :

Step 1: Setup the tools

Make sure you 'source' the settings64.sh (Vivado) and settings.sh (PetaLinux) scripts to add them to your path

Step 2: One time PYNQ tools setup

* Clone PYNQ from<https://github.com/Xilinx/PYNQ> and checkout branch: image\_v3.0
* cd into the clone and proper branch, then execute "./sdbuild/scripts/setup\_host.sh"
* Install any requested additional Debian apt packages that setup\_host.sh asks for
* Once setup\_host.sh is successful, reboot and re-login
* You may remove the just cloned PYNQ git repo, it is no longer needed

### MODEL

#### Model Inspector

1. <https://github.com/Xilinx/Vitis-AI/blob/v3.0/examples/vai_quantizer/pytorch/inspector_tutorial.ipynb>

#### Prepare database for Training Segmentation Models

1. <https://github.com/Xilinx/Vitis-AI/blob/v3.0/examples/vai_quantizer/pytorch/inspector_tutorial.ipynb>

#### Accelerating ML Preprocessing in Vitis : Vitis-Vision [[18]](https://www.xilinx.com/developer/articles/accelerating-ml-preprocessing-with-vitis-vision.html)

1. Caffe [17]
2. brand new model that is not included in the Vitis AI Library, we suggest that you handle the pre-processing and post-processing yourself, using the **VART APIs** for programming : <https://www.xilinx.com/developer/articles/introduction-to-vitis-ai-library-apis.html>
3. Whole Application Acceleration
4. <https://github.com/Xilinx/Vitis-AI/blob/v3.0/examples/waa/README.md>
5. <https://medium.com/mlearning-ai/letterbox-in-object-detection-77ee14e5ac46>

DPU effectively sees values in the range [-1.0, +1.0).

#### Training the Pytorch Model [[20]](https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/#2-select-a-model) [View](https://netron.app/)

1. <https://towardsdatascience.com/training-yolo-for-object-detection-in-pytorch-with-your-custom-dataset-the-simple-way-1aa6f56cf7d9>
2. utils/dataloaders.py
3. class LoadImages:
4. utils/general.py
5. def non\_max\_suppression(
6. models/common.py
7. class DetectMultiBackend(nn.Module):
8. <https://github.com/ultralytics/yolov5/issues/36>

#### Evaluating the Floating Point Models on the Host PC

1. <https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/Inspect-Float-Model-Before-Quantization>

#### Quantizing and Compiling the Segmentation networks for DPU implementation [[19](https://github.com/Xilinx/Vitis-AI-Tutorials/blob/1.4/Design_Tutorials/09-mnist_pyt/README.md)]

1. channel pruning
2. <https://github.com/Xilinx/Vitis-AI/tree/v3.0/examples/vai_optimizer>
3. Quantizing the Model
4. For example/resnet18\_quant.py, command line to do hardware-aware calibration:
5. python resnet18\_quant.py --quant\_mode calib --target DPUCAHX8L\_ISA0\_SP
6. command line to test hardware-aware quantized model accuracy:
7. python resnet18\_quant.py --quant\_mode test --target DPUCAHX8L\_ISA0\_SP
8. command line to deploy quantized model:
9. python resnet18\_quant.py --quant\_mode test --target DPUCAHX8L\_ISA0\_SP --subset\_len 1 --batch\_size 1 --deploy
10. PyTorch examples are available as follows:
11. PT Post-Training Quantization (Inculded Hardware-Aware Quantization Strategy)
12. <https://github.com/Xilinx/Vitis-AI/blob/v3.0/src/vai_quantizer/vai_q_pytorch/example/resnet18_quant.py>
13. PT Quantization Aware Training
14. <https://github.com/Xilinx/Vitis-AI/blob/v3.0/src/vai_quantizer/vai_q_pytorch/example/resnet18_qat.py>
15. <https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/vai_q_pytorch-QAT>
16. It is recommended to use small learning rates when performing QAT.
17. <https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/Configuration-of-Quantization-Strategy>

#### Vitis AI Compiler

1. vai\_c\_xir

#### (Optional)AI Optimiser

1. <https://docs.xilinx.com/access/sources/dita/map?ft:locale=en-US&url=ug1333-ai-optimizer&Doc_Version=3.0%20English>

#### Running the Models

1. Launch a file transfer program - I like to use MobaXterm, though others such as pscp or WinSCP can also be used
2. Vitis AI Runtime (VART) is a set of API functions that support the integration of the DPU into software applications.
3. Integrating the DPU¶
4. Vitis Integration
5. reference design and IP repository: <https://github.com/Xilinx/Vitis-AI/tree/v3.0/dpu>
6. Vivado Integration
7. reference design and IP repository: <https://github.com/Xilinx/Vitis-AI/tree/v3.0/dpu>
8. tutorial: <https://github.com/Xilinx/Vitis-AI-Tutorials/blob/2.0/Tutorials/Vitis-AI-Vivado-TRD/>
9. VART : <https://github.com/Xilinx/Vitis-AI/tree/v3.0/src/vai_runtime/quick_start_for_embedded.md>

To familiarize yourself with the unified APIs, use the VART examples. These examples are only to understand the APIs and do not provide high performance. These APIs are compatible between the edge and cloud, though cloud boards may have different software optimizations such as batching and on the edge would require multi-threading to achieve higher performance. If you desire higher performance, see the Vitis AI Library samples and demo software.

If you want to do optimizations to achieve high performance, here are some suggestions:

* Rearrange the thread pipeline structure so that every DPU thread has its own "DPU" runner object.
* Optimize display thread so that when DPU FPS is higher than display rate, skipping some frames. 200 FPS is too high for video display.
* Pre-decoding. The video file might be H.264 encoded. The decoder is slower than the DPU and consumes a lot of CPU resources. The video file has to be first decoded and transformed into raw format.
* The batch mode on Versal boards needs special consideration as it may cause video frame jittering. ZCU102 has no batch mode support.
* OpenCV cv::imshow is slow, so you need to use libdrm.so. This is only for local display, not through X server.

<https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/vart.Runner-Example>

1. <https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/Vitis-AI-Examples>
2. DPU PYNQ : : <https://github.com/Xilinx/DPU-PYNQ/blob/master/boards/README.md>

#### Post-processing the Hardware Inference Output

1. Check the compiled quantised model in Netron and make custom operators if required
2. <https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/Custom-OP-Workflow>
3. <https://docs.xilinx.com/r/en-US/ug1354-xilinx-ai-sdk/Example-Code>
4. Vitis AI Profiler
5. <https://github.com/Xilinx/Vitis-AI/tree/v3.0/examples/vai_profiler>

<https://github.com/Xilinx/Vitis-AI-Tutorials/blob/1.4/Design_Tutorials/16-profiler_introduction/README.md>

Reference models : Model zoo : <https://xilinx.github.io/Vitis-AI/docs/reference/ModelZoo_VAI3.0_Github_web.htm>

<https://xilinx.github.io/Vitis-AI/docs/workflow-model-zoo.html>

Parameters :

<https://docs.xilinx.com/r/en-US/ug1354-xilinx-ai-sdk/yolo_v3_param>

Documentation : <https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/Development-Flow-Overview>

## Usage

The get started, users have to run the following command on the Ultra96 board:

cd /home/root/jupyter\_notebooks

git clone https://github.com/dac-sdc/fpga\_starter\_2023.git

Remember the user name and password are both xilinx.

After the above step is completed successfully, you will see a folder fpga\_starter\_2023 under your jupyter notebook dashboard. Open the sample\_team/dac\_sdc.ipynb notebook for directions on where to begin.

<https://github.com/Xilinx/Kria-PYNQ>

<https://pynq.readthedocs.io/en/latest/getting_started/pynq_z1_setup.html>

### Folder Structure

1. sample\_team: This folder contains files for a sample team. This includes a .bit and .tcl file that defines the hardware, and a .ipynb jupyter notebook, and a hw folder that is used to create a Vivado project. You should create a new folder for your team, where you will keep all of your files.
2. images: All the test images are stored in this folder. Replace the example images in this directory with the full training set.
3. result: The results contain the output xml produced when execution is complete, and contains the runtime, energy usage, and predicted location of each object in each image.

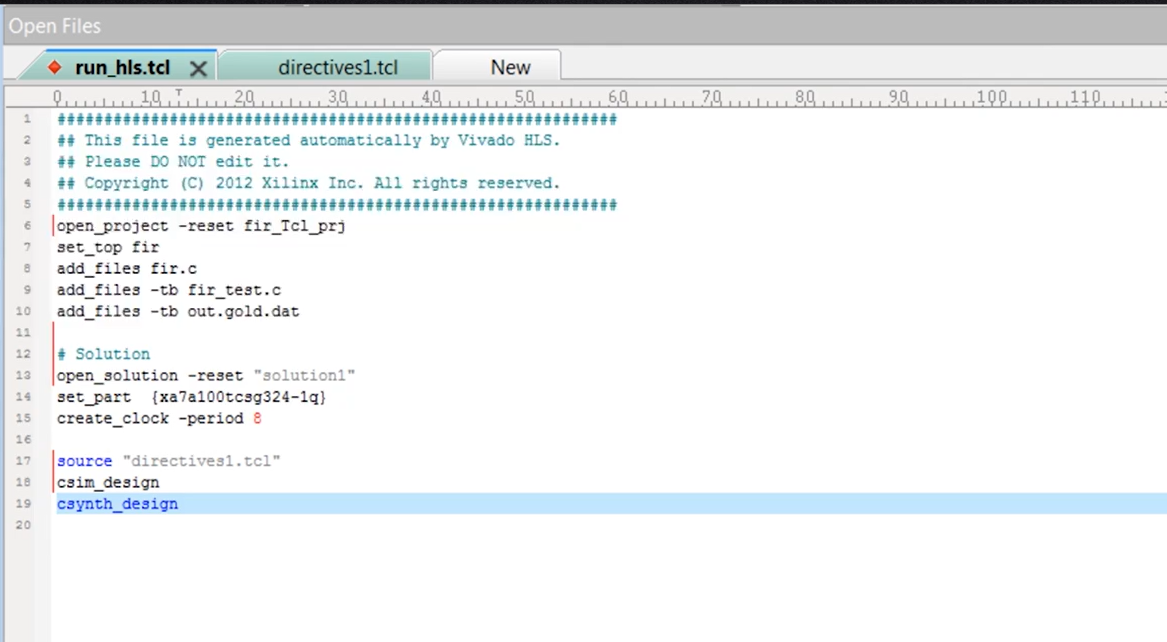
### Submission

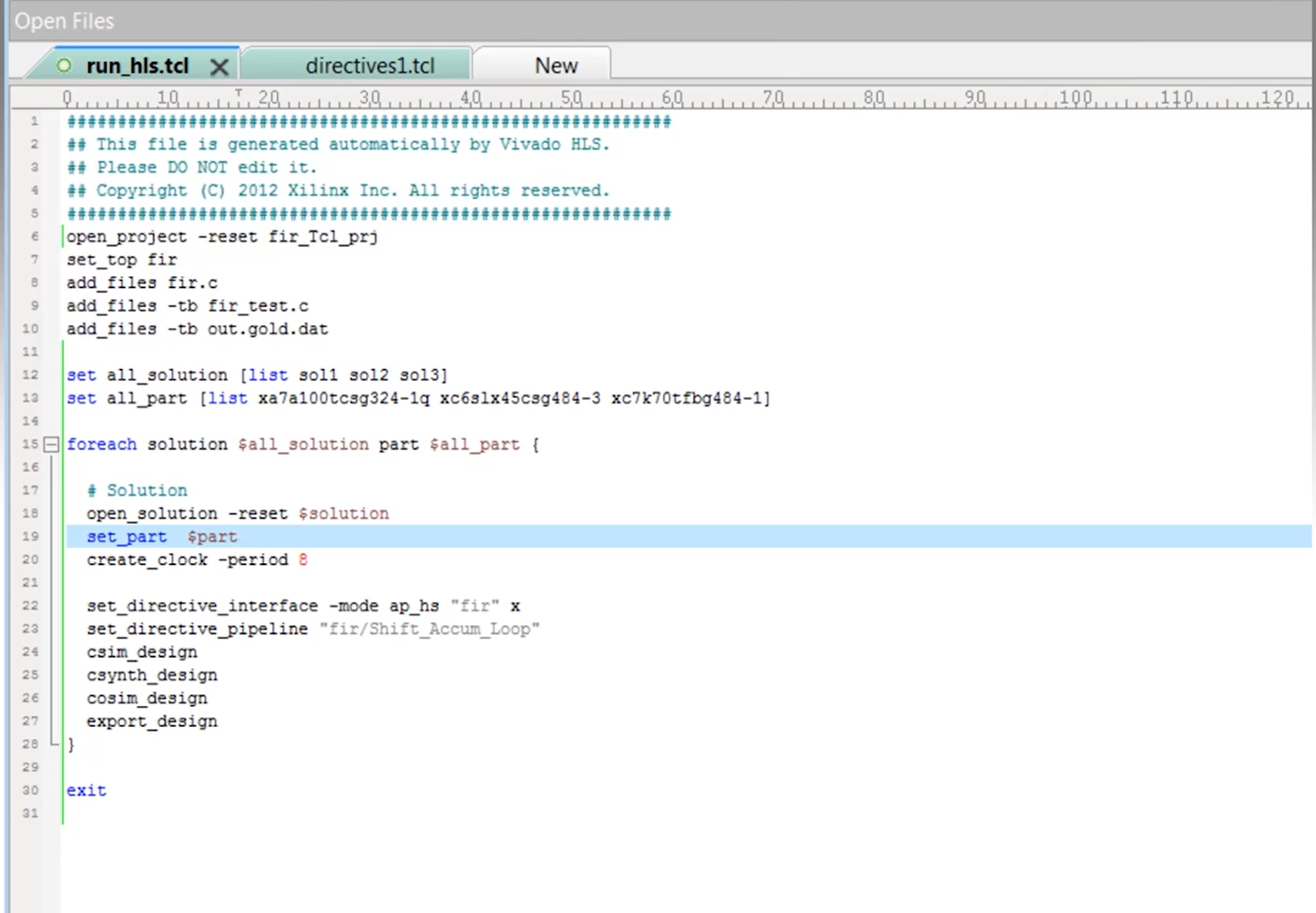
# Refereneces:

1. Generating project TCL file and regenerating project from TCL file in Vivado: <https://www.youtube.com/watch?v=BXhLFecEPc8>

FPGA NN Playlists:

1. Vivado HLS Course Training(2d conv also): <https://www.youtube.com/playlist?list=PLo7bVbJhQ6qzK6ELKCm8H_WEzzcr5YXHC>
2. Machine Learning on FPGAs(only neuron): <https://www.youtube.com/playlist?list=PLGzeDuLmmxDpEsCAjf_sYrMC6p-Y0Ummk>
3. Reconfigurable Embedded Systems with Xilinx Zynq APSoC : <https://www.youtube.com/playlist?list=PLXHMvqUANAFOviU0J8HSp0E91lLJInzX1> (detailed)
   1. Line Buffer
   * Send 4 lines of image in line buffer as a 3x3 kernel will do convolution on 3 line buffers and we will simultaneously write data on the 4th line buffer
   * Each line buffer gets 8 bits input and gives 3 bytes output
   * Design of line buffer : <https://youtu.be/n35zS__YEFQ>
   * Control logic takes 8 bit input for each pixel and outputs the 72 bits 3x3 block
   * Control logic : <https://youtu.be/v8pHH-q-0sE>
   1. Convolution (<https://youtu.be/6El_NQrpgCY> )
      1. Added pipelining , stages are multiplication , addition(combinational) and division
   2. A valid signal starting from input is passed through the pipeline and output valid becomes high when input data after processing reaches output
4. Fixed Point vs Floating Point
5. Converting TCL to vivado hls project : <https://www.xilinx.com/video/hardware/using-the-vivado-hls-tcl-interface.html>





| vivado\_hls -f run\_hls.tcl | Make project using tcl file |
| --- | --- |
| vivado\_hls -p fir\_tcl\_prj | Open project in vivado |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. **TUTORIAL :** <https://github.com/dac-sdc/fpga_starter_2023>
2. <https://xilinx.github.io/kria-apps-docs/kv260/2022.1/build/html/docs/smartcamera/docs/customize_ai_models.html>
3. <https://xilinx.github.io/kria-apps-docs/kv260/2022.1/build/html/docs/build_vivado_design.html>
4. <https://xilinx.github.io/kria-apps-docs/kv260/2022.1/build/html/docs/build_vitis_platform.html>
5. <https://xilinx.github.io/kria-apps-docs/kv260/2022.1/build/html/docs/build_accel.html>
6. <https://xilinx.github.io/kria-apps-docs/kv260/2022.1/build/html/docs/generating_custom_firmware.html>
7. <https://github.com/Xilinx/PYNQ_Workshop/blob/master/Session_4/6_dma_tutorial.ipynb>
8. <https://pynq.readthedocs.io/en/latest/pynq_libraries/dma.html>
9. <https://www.xilinx.com/developer/articles/introduction-to-vitis-ai-library-apis.html>
10. <https://www.xilinx.com/developer/articles/part1-environment-setup-and-Installation.html>
11. <https://www.xilinx.com/developer/articles/part3-training-models.html>
12. <https://www.xilinx.com/developer/articles/accelerating-ml-preprocessing-with-vitis-vision.html>
13. <https://github.com/Xilinx/Vitis-AI-Tutorials/blob/1.4/Design_Tutorials/09-mnist_pyt/README.md>
14. <https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/#2-select-a-model>

# APPENDIX

## ERRORS

The resulting .bit file is located in /prj/<project name>/out/redpitaya.bit This file must be copied to /opt/redpitaya/fpga on the Red Pitaya itself.

If the script returns the following error:

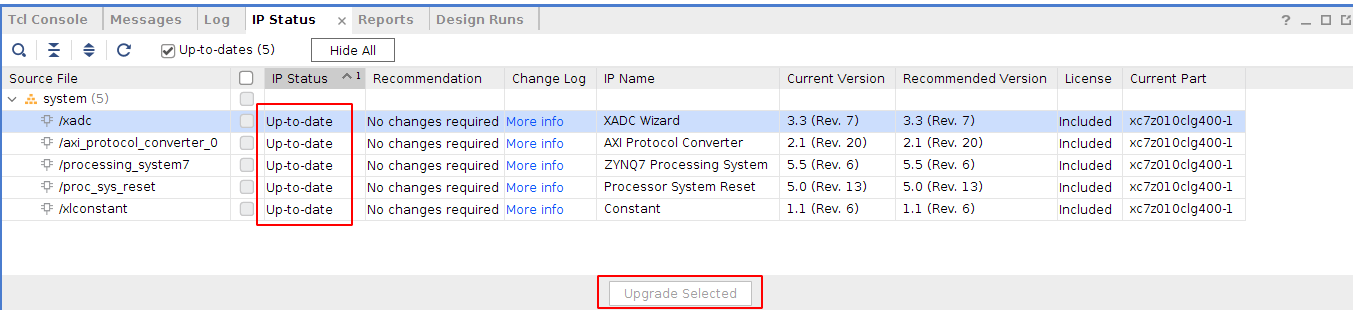
### BD\_TCL-109" "ERROR" "This script was generated using Vivado 2020.1 ....

First, find the line containing

set scripts\_vivado\_version 2020.1

and change 2020.1 to your version. This is a quick and dirty way to get the build working in other versions of Vivado. However, this could be problematic if some of the IPs used are different in your version.

To update the script properly, open the project GUI(see below), go to menu Reports-> Report IP Status. A new tab opens below the code window. If all IPs are not up-to-date, they need to be updated. Before doing this, the TCL script must still be manually modified to your Vivado version, or the block design will not be created when Vivado starts.



When IPs are up-to-date, go to the tab Tcl console and run command:

write\_bd\_tcl systemZ10.tcl

Of course, the script may also be named systemZ20.tcl systemZ20\_14.tcl, depending on your board.

This generates a new tcl script that replaces the old script in fpga/prj/<project name>/ip

**Upgrade Version**

**.sh**

#!/bin/bash

set -ex

# Argument 1: old vivado version sourceme

# Argument 2: new vivado version sourceme

# Run this in the directory of the board you'd like to upgrade

project=$(basename $(pwd))

source $1

make project

cp src/tcl/${project}\_bd.tcl src/tcl/${project}\_bd.tcl.bak

source $2

vivado -mode batch -source ../common/scripts/upgrade\_version.tcl -tclargs \*/\*.xpr src/tcl/${project}\_bd.tcl

**.tcl**

open\_project [lindex $argv 0]

#update\_compile\_order -fileset sources\_1

export\_ip\_user\_files -of\_objects [get\_ips -all] -no\_script -reset -quiet

upgrade\_ip [get\_ips -all] -log ip\_upgrade.log

validate\_bd\_design

write\_bd\_tcl -force [lindex $argv 1]