Lab-FINN

Fast, Scalable Quantized Neural Network Inference on FPGAs, FINN

Lecturer: Hua-Yang Weng

2022 Fall

FINN LAB Setup

- You can also find these settings in run-docker.sh

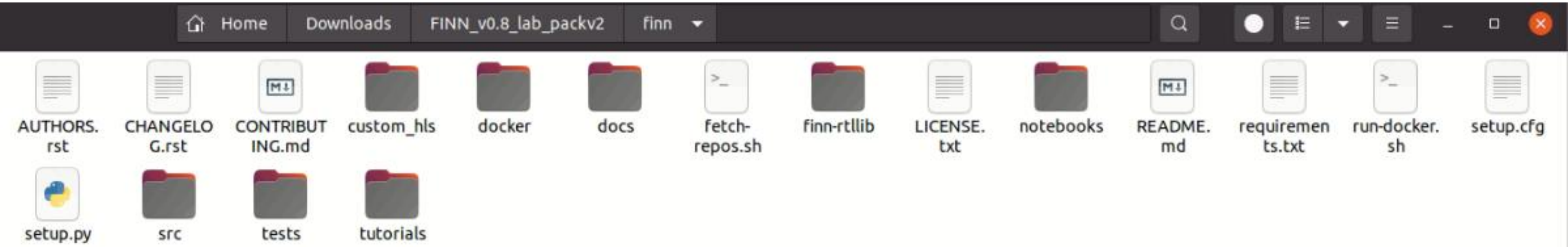
※ **You need to open a new terminal to activate them!**

- Only needs to be set for the first time.
- Open bashrc file and edit environment vars
  - vi ~/.bashrc
- Create build directory in your home
  - mkdir ~/build
- Add the following codes at the end of ~/.bashrc file
  - export FINN_HOST_BUILD_DIR=/path/to/your/model/build
  - export FINN_XILINX_PATH =/opt/Xilinx
  - export FINN_XILINX_VERSION =2022.1
  - export PYNQ_BOARD=Pynq-Z2

```
export FINN_XILINX_PATH=/opt/Xilinx
export FINN_XILINX_VERSION=2022.1
export PYNQ_BOARD=Pynq-Z2
export FINN_HOST_BUILD_DIR=/mnt/HLSNAS/huayang/FINN_v0.8_lab_packv2/finn/build
```

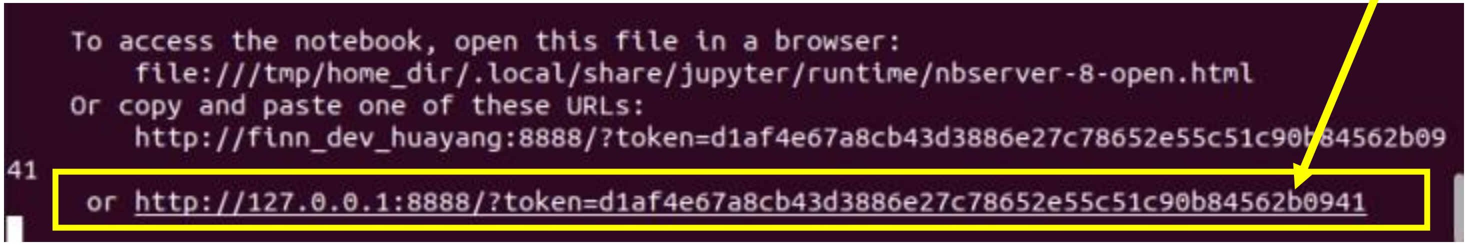
FINN Directory

- First, unzip the file FINN_v0.8_lab_packv2.zip
- finn/
  - run-docker.sh The script we are going to use.
  - src/ Containing partial FINN source code
  - notebooks/ Containing the jupyter tutorial notebooks



- Run docker to use FINN
  - $ ./run-docker.sh notebook

Ctrl + left click to open the notebook



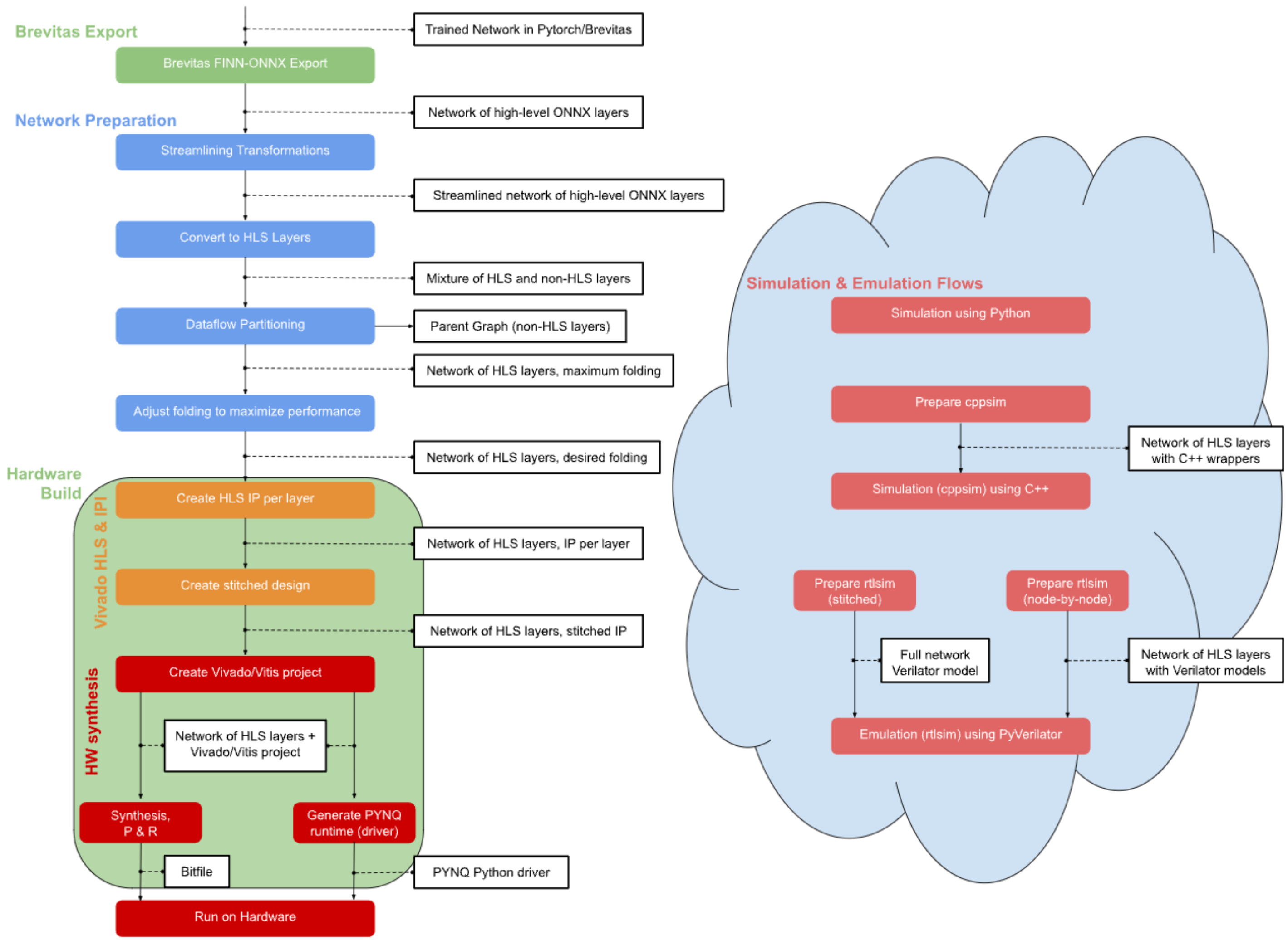
Outline

- Part1: End-to-End FINN Flow for a Simple Convolutional Net
  - finn/notebooks/end2end_example/bnn-pynq/cnv_end2end_example.ipynb
- Part2 : VGG on CIFAR-100
- Part3: Performance Improvement
  - finn/notebooks/end2end_example/bnn-pynq/tfc_end2end_example.ipynb
- Questions
- Report & Submission

Outline

- Part1: End-to-End FINN Flow for a Simple Convolutional Net
  - finn/notebooks/end2end_example/bnn-pynq/cnv_end2end_example.ipynb
- Part2 : VGG on CIFAR-100
- Part3: Performance improvement
- Report & Submission

Quick Recap of the End-to-End Flow

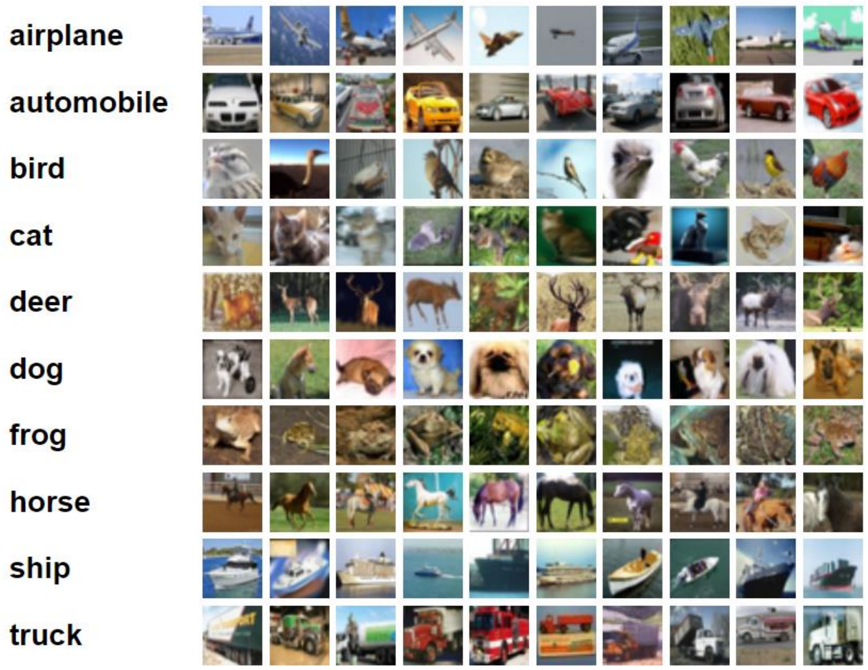


Quick Introduction to the CNV-w1a1 Network

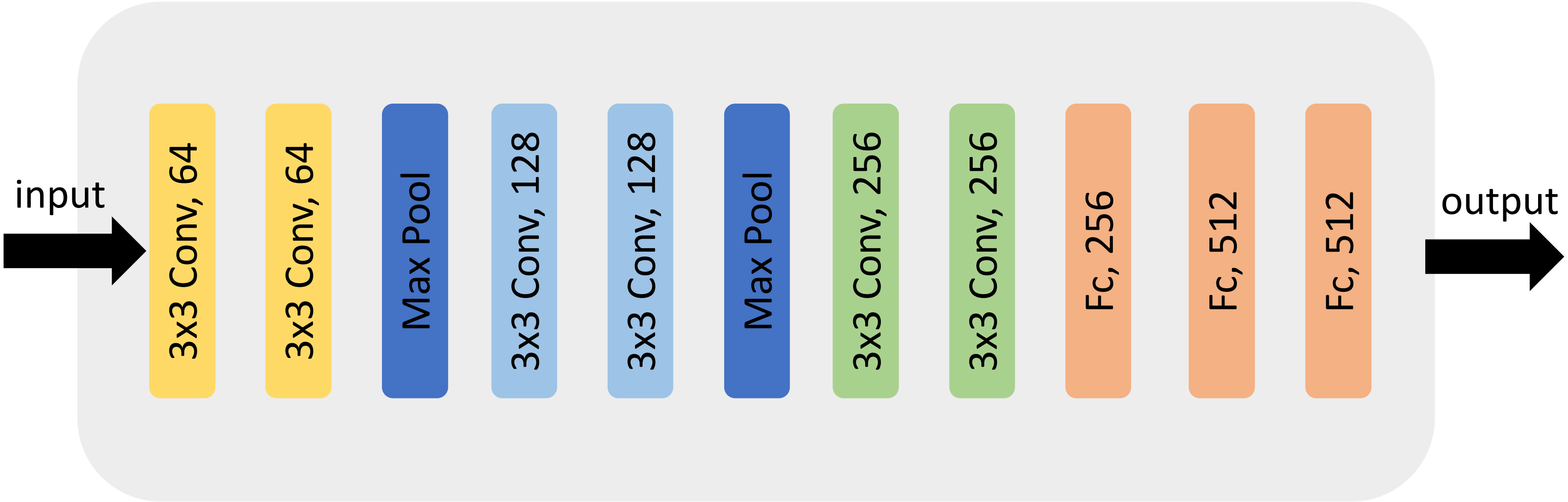
- Input size: 32x32 8bits RGB Data
- Weights and Activations are binary values.
- Output is a number that represents the result predicted by our model.

CIFAR-10 dataset

- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class.
- There are 50000 training images and 10000 test images.



Network Architecture

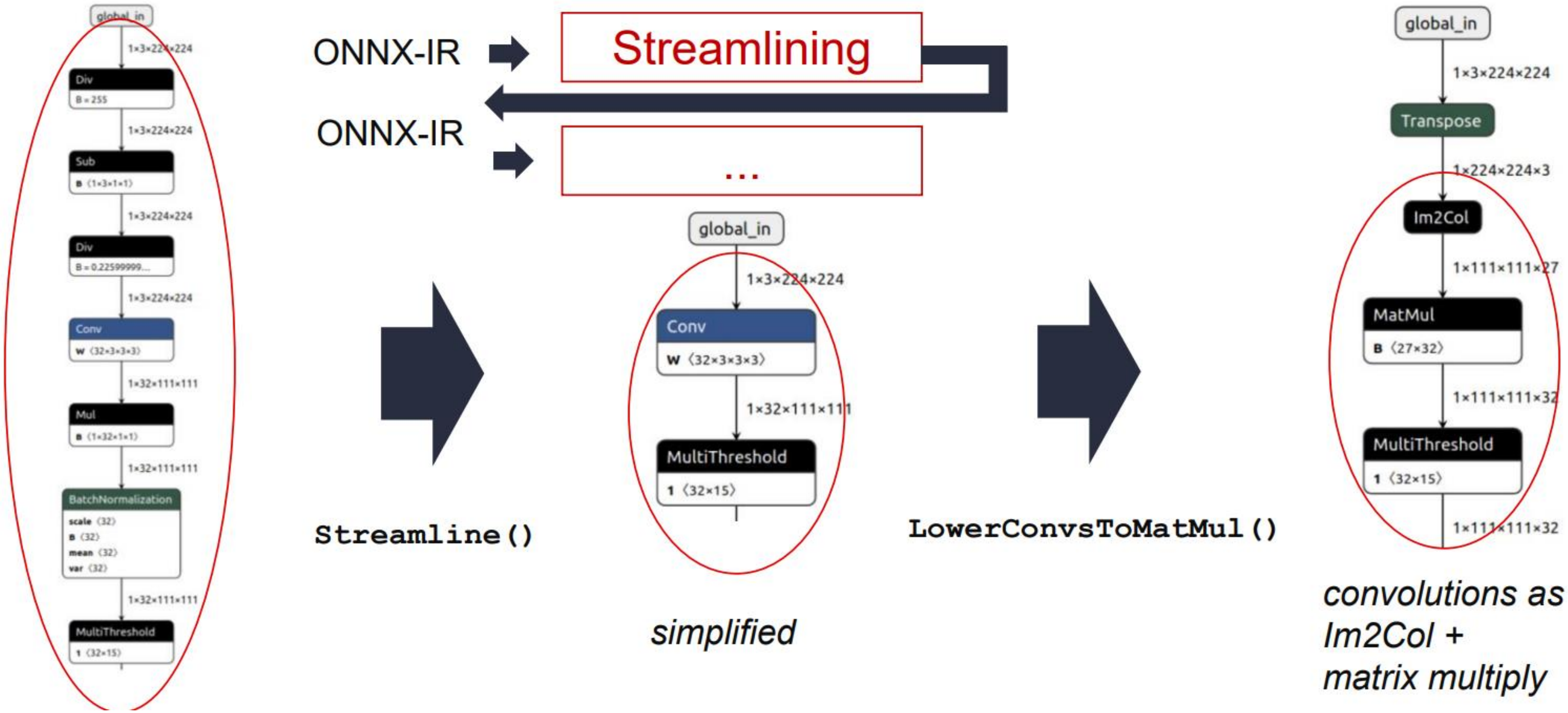


Brevitas Export, FINN Import and Tidy-Up

- We will start by exporting the pretrained CNV-w1a1 network to ONNX.
- Run the "tidy-up" transformations to have a first look at the topology.

```python
import onnx
from finn.util.test import get_test_model_trained
import brevitas.onnx as bo
from finn.core.modelwrapper import ModelWrapper
from finn.transformation.infer_shapes import InferShapes
from finn.transformation.fold_constants import FoldConstants
from finn.transformation.general import GiveReadableTensorNames, GiveUniqueNodeNames, RemoveStaticGraphInputs

cnv = get_test_model_trained("CNV", 1, 1)
bo.export_finn_onnx(cnv, (1, 3, 32, 32), build_dir + "/end2end_cnv_w1a1_export.onnx")
model = ModelWrapper(build_dir + "/end2end_cnv_w1a1_export.onnx")
model = model.transform(InferShapes())
model = model.transform(FoldConstants())
model = model.transform(GiveUniqueNodeNames())
model = model.transform(GiveReadableTensorNames())
model = model.transform(RemoveStaticGraphInputs())
model.save(build_dir + "/end2end_cnv_w1a1_tidy.onnx")
```

Transformation



Transformation

- Pre-processing
  - Divides the input uint8 data by 255 so the inputs to the CNV-w1a1 network are bounded between [0, 1]
- Post-processing
  - Takes the output of the network and returns the index (0-9) of the image category with the highest probability (top-1).

Dataflow Partitioning

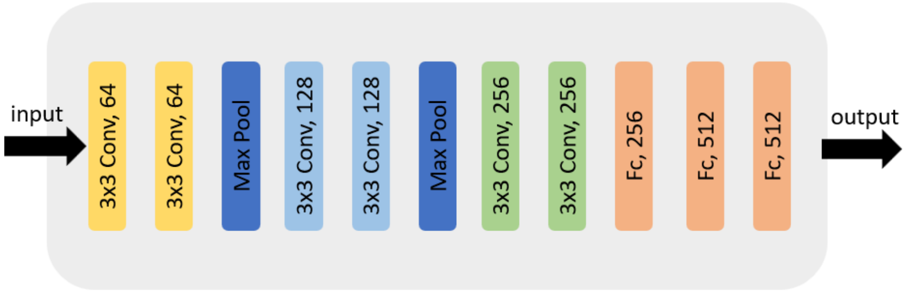
- We'll first convert the layers that we can put into the FPGA into their HLS equivalents and separate them out into a dataflow partition.

```
model = ModelWrapper(build_dir + "/end2end_cnv_w1a1_streamlined.onnx")
model = model.transform(to_hls.InferBinaryStreamingFCLayer(mem_mode))
model = model.transform(to_hls.InferQuantizedStreamingFCLayer(mem_mode))
# TopK to LabelSelect
model = model.transform(to_hls.InferLabelSelectLayer())
# input quantization (if any) to standalone thresholding
model = model.transform(to_hls.InferThresholdingLayer())
model = model.transform(to_hls.InferConvInpGen())
model = model.transform(to_hls.InferStreamingMaxPool())
# get rid of Reshape(-1, 1) operation between hlslib nodes
model = model.transform(RemoveCNVtoFCFlatten())
# get rid of Tranpose -> Tranpose identity seq
model = model.transform(absorb.AbsorbConsecutiveTransposes())
# infer tensor data layouts
model = model.transform(InferDataLayouts())
parent_model = model.transform(CreateDataflowPartition())
parent_model.save(build_dir + "/end2end_cnv_w1a1_dataflow_parent.onnx")
sdp_node = parent_model.get_nodes_by_op_type("StreamingDataflowPartition")[0]
sdp_node = getCustomOp(sdp_node)
dataflow_model_filename = sdp_node.get_nodeattr("model")
# save the dataflow partition with a different name for easier access
dataflow_model = ModelWrapper(dataflow_model_filename)
dataflow_model.save(build_dir + "/end2end_cnv_w1a1_dataflow_model.onnx")
```

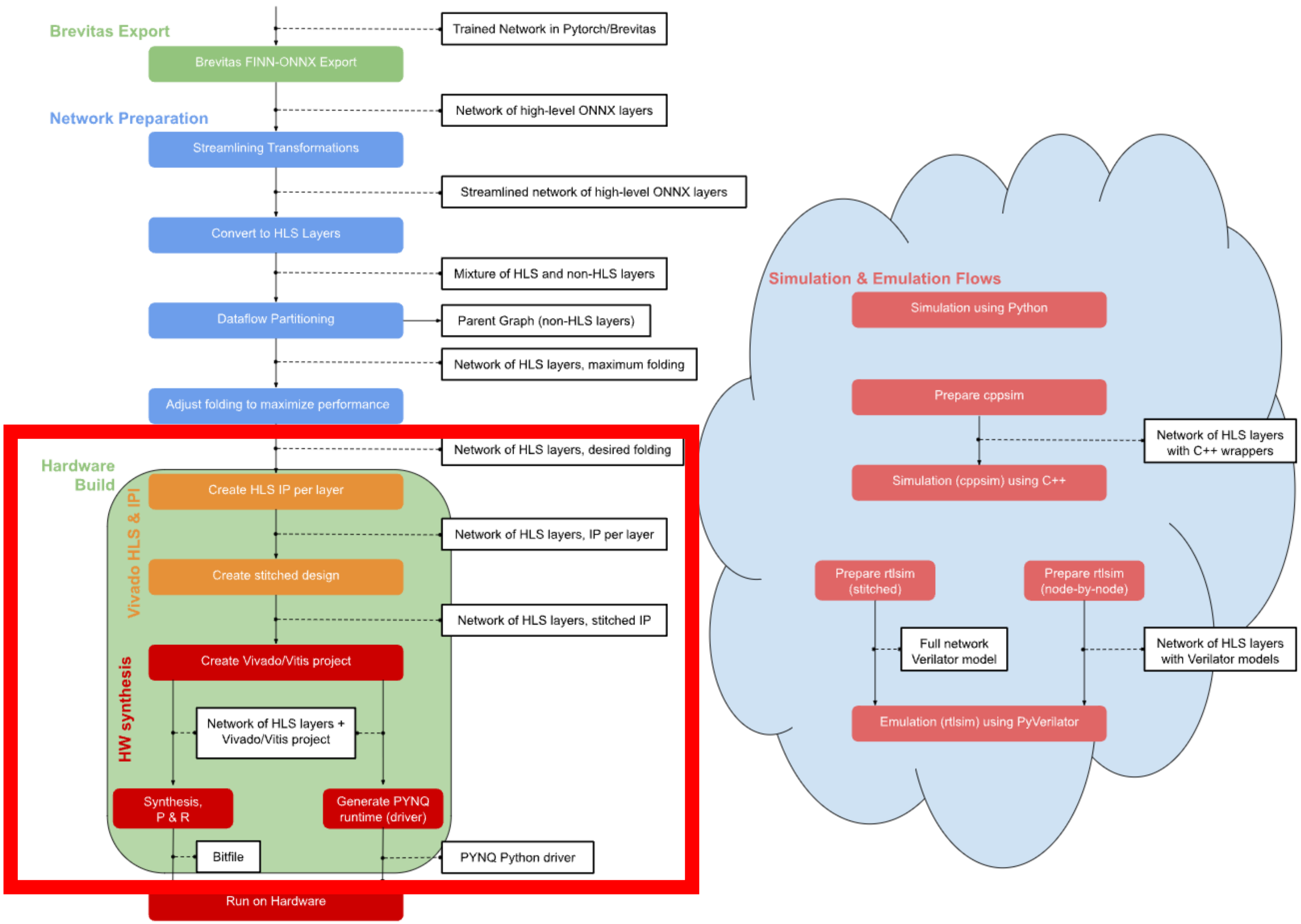
Adjust folding factor

- We have to set the folding factors for certain layers to adjust the performance of our accelerator.

```python
model = ModelWrapper(build_dir + "/end2end_cnv_w1a1_dataflow_model.onnx")
fc_layers = model.get_nodes_by_op_type("StreamingFCLayer_Batch")
# each tuple is (PE, SIMD, in_fifo_depth) for a layer
folding = [
    (16, 3, 128),
    (32, 32, 128),
    (16, 32, 128),
    (16, 32, 128),
    (4, 32, 81),
    (1, 32, 2),
    (1, 4, 2),
    (1, 8, 128),
    (5, 1, 3),
]
for fcl, (pe, simd, ififodepth) in zip(fc_layers, folding):
    fcl_inst = getCustomOp(fcl)
    fcl_inst.set_nodeattr("PE", pe)
    fcl_inst.set_nodeattr("SIMD", simd)
    fcl_inst.set_nodeattr("inFIFODepth", ififodepth)

# use same SIMD values for the sliding window operators
swg_layers = model.get_nodes_by_op_type("ConvolutionInputGenerator")
for i in range(len(swg_layers)):
    swg_inst = getCustomOp(swg_layers[i])
    simd = folding[i][1]
    swg_inst.set_nodeattr("SIMD", simd)

model = model.transform(GiveUniqueNodeNames())
model.save(build_dir + "/end2end_cnv_w1a1_folded.onnx")
```



Hardware Build



Hardware Generation

- Specify the target board and clock period
- ZynqBuild is the hardware generation function
- This step may take about 120 minutes depending on your host computer.

```
test_pynq_board = "Pynq-Z2"
target_clk_ns = 10

from finn.transformation.fpgadataflow.make_zynq_proj import ZynqBuild
model = ModelWrapper(build_dir+"/end2end_cnv_w1a1_folded.onnx")
model = model.transform(ZynqBuild(platform = test_pynq_board, period_ns = target_clk_ns))
model.save(build_dir + "/end2end_cnv_w1a1_synth.onnx")
```

Deployment and Remote Execution

- Here, Please rent the remote PYNQ-Z2 service first
  - (ASK your TAs for the service IP)

```
##################################################
#                                                #
#       Welcome to BoLedu's OnlineFPGA service   #
#                                                #
#       Select from OnlineFPGA service menu:     #
#       System maintenance: 6:00am to 7:00am     #
#                                                #
#       email: onlinefpga@gmail.com              #
#                                                #
##################################################
```

```
[1] register an BoLedu account
[2] already have an account
[0] exit OnlineFPGA service


please enter your option:
>>
```

```python
import os

# set up the following values according to your own environment
# FINN will use ssh to deploy and run the generated accelerator
# ip = "192.168.2.99"
# username = os.getenv("PYNQ_USERNAME", "xilinx")
# password = os.getenv("PYNQ_PASSWORD", "xilinx")
# port = os.getenv("PYNQ_PORT", 22)
# target_dir = os.getenv("PYNQ_TARGET_DIR", "/home/xilinx/finn_cnv_end2end_example")
# # set up ssh options to only allow publickey authentication
# options = "-o PreferredAuthentications=publickey -o PasswordAuthentication=no"

# # test access to PYNQ board
# ! ssh {options} {username}@{ip} -p {port} cat /var/run/motd.dynamic

ip = "xxx.xxx.xxx.xxx"    # your current build (not pynq) machine ip
username = 'huayang'    # your current build (not pynq) machine account
password = 'x'  # dummy, don't modify
port = 1100   # your current build (not pynq) machine ip port
target_dir = os.getenv("PYNQ_TARGET_DIR", "/home/xilinx/finn_cnv_end2end_example")
```

Enter your ip info of the
"Host Build machine" (not pynq)

The following sections will be using both ***Host machine*** & ***remote pynq*** Jupyter notebooks

DeployToPYNQ

- SCP command to transfer the bitstreams to remote PYNQ
  - The current version of our lab requires user to manually type this command into the rented PYNQ board.

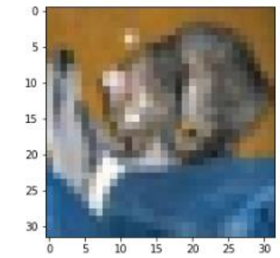
```
from finn.transformation.fpgadataflow.make_deployment import DeployToPYNQ

model = ModelWrapper(build_dir + "/end2end_cnv_w1a1_synth.onnx")
model = model.transform(DeployToPYNQ(ip, port, username, password, target_dir))
model.save(build_dir + "/end2end_cnv_w1a1_pynq_deploy.onnx")
```

```
Please manually copy and paste the following command in the remote pynq jupter notebook.
=========================================
mkdir -p /home/xilinx/jupyter_notebooks/FINN/finn_dev_huayang
scp -P[redacted] -r huayang@[redacted]:/mnt/HLSNAS/huayang/FINN_v0.8_lab_packv2/finn/build/pynq_deployment_idlhln4h /
home/xilinx/jupyter_notebooks/FINN/finn_dev_huayang
=========================================
```

jupyter Logout

```
root@pynq:/home/xilinx/jupyter_notebooks# mkdir -p /home/xilinx/jupyter_notebooks/FINN/finn_dev_huayang
root@pynq:/home/xilinx/jupyter_notebooks# ls
 base       course-lab_1    FINN              logictools          pynq_peripherals
 common     course-lab_2    getting_started   pynq_composable    'Welcome to Pynq.ipynb'
root@pynq:/home/xilinx/jupyter_notebooks# scp -P[redacted] -r huayang@[redacted]:/mnt/HLSNAS/huayang/FINN_v0.8_lab_packv2/fi
nn/build/pynq_deployment_idlhln4h /home/xilinx/jupyter_notebooks/FINN/finn_dev_huayang
huayang@[redacted] password:
resizer.bit                                          100% 3951KB  10.2MB/s   00:00
resizer.hwh                                          100%  271KB   5.0MB/s   00:00
driver_base.py                                       100%   20KB   3.4MB/s   00:00
datatype.py                                          100%   10KB   2.1MB/s   00:00
__init__.py                                          100%     0    0.0KB/s   00:00
basic.py                                             100%   13KB   2.6MB/s   00:00
__init__.py                                          100%     0    0.0KB/s   00:00
data_packing.py                                      100%   18KB   2.5MB/s   00:00
__init__.py                                          100%     0    0.0KB/s   00:00
driver.py                                            100% 5032     1.0MB/s   00:00
validate.py                                          100% 4113   959.3KB/s   00:00
```

Execute with a Sample Image



- SCP Again: Host input .npy -> remote PYNQ
- Execute remote PYNQ using driver.py
- SCP Again: remote PYNQ output.npy -> Host

```
import numpy as np
from finn.core.onnx_exec import execute_onnx

model = ModelWrapper(build_dir + "/end2end_cnv_w1a1_pynq_deploy.onnx")
iname = model.graph.input[0].name
oname = model.graph.output[0].name
ishape = model.get_tensor_shape(iname)
input_dict = {iname: x.astype(np.float32).reshape(ishape)}
ret = execute_onnx(model, input_dict, True)
```

```
Please manually copy and paste the following command in the remote pynq jupter notebook.
============== copy input to PYNQ board ================
scp -P[redacted] -r huayang@[redacted]/mnt/HLSNAS/huayang/FINN_v0.8_lab_packv2/finn/build/pynq_deployment_idlhln4h/i
nput.npy /home/xilinx/jupyter_notebooks/FINN/finn_dev_huayang/pynq_deployment_idlhln4h

============== use platform attribute for correct remote execution ================
cd /home/xilinx/jupyter_notebooks/FINN/finn_dev_huayang/pynq_deployment_idlhln4h; python3 driver.py --exec_mode=exec
ute --batchsize=1 --bitfile=resizer.bit --inputfile=input.npy --outputfile=output.npy --platform=zynq-iodma

============== copy generated output to local ==========================
scp -P[redacted] /home/xilinx/jupyter_notebooks/FINN/finn_dev_huayang/pynq_deployment_idlhln4h/output.npy huayang@[redacted]
:/mnt/HLSNAS/huayang/FINN_v0.8_lab_packv2/finn/build/pynq_deployment_idlhln4h

============== Please open a new cell in the original jupyter notebook and paste the below ================
outp = np.load('{}/output.npy'.format(model.get_metadata_prop('pynq_deploy_dir')))
ret[model.graph.output[0].name] = outp
```

Download the dataset

- Download the CIFAR10 dataset @ remote pynq

```
In [27]: # ! ssh {options} -t {username}@{ip} -p {port} 'echo {password} | sudo -S pip3 install git+https://github.com/fbcotte
         print("Please paste the below command into pynq terminal")
         print("=========================================================")
         print("su xilinx")
         print("sudo python3 -m pip install git+https://github.com/fbcotter/dataset_loading.git@0.0.4#egg=dataset_loading")

         Please paste the below command into pynq terminal
         =========================================================
         su xilinx
         sudo python3 -m pip install git+https://github.com/fbcotter/dataset_loading.git@0.0.4#egg=dataset_loading
```

- If your pynq cannot download the mnist dataset, then post the issue to the LAB-FINN discussion space

Xilinx FINN

Validating the Accuracy on a PYNQ Board

- Validating the Accuracy @ remote pynq using validate.py

In [28]:
```
# ! ssh {options} -t {username}@{ip} -p {port} 'cd {target_dir_pynq}; echo {password} | sudo -S python3.6 validate.py
print("Please paste the below command into pynq terminal")
print("============================================================")
print("su ")
print("python3 validate.py --dataset cifar10 --batchsize 1000")
```

```
Please paste the below command into pynq terminal
============================================================
su
python3 validate.py --dataset cifar10 --batchsize 1000
```

```
[sudo] password for xilinx: Tar File found in dest_dir. Not Downloading again
Extracting Python CIFAR10 data.
Files extracted
batch 1 / 10 : total OK 851 NOK 149
batch 2 / 10 : total OK 1683 NOK 317
batch 3 / 10 : total OK 2522 NOK 478
batch 4 / 10 : total OK 3370 NOK 630
batch 5 / 10 : total OK 4207 NOK 793
batch 6 / 10 : total OK 5044 NOK 956
batch 7 / 10 : total OK 5887 NOK 1113
batch 8 / 10 : total OK 6728 NOK 1272
batch 9 / 10 : total OK 7570 NOK 1430
batch 10 / 10 : total OK 8419 NOK 1581
Final accuracy: 84.190000
```

Screen Dump this result

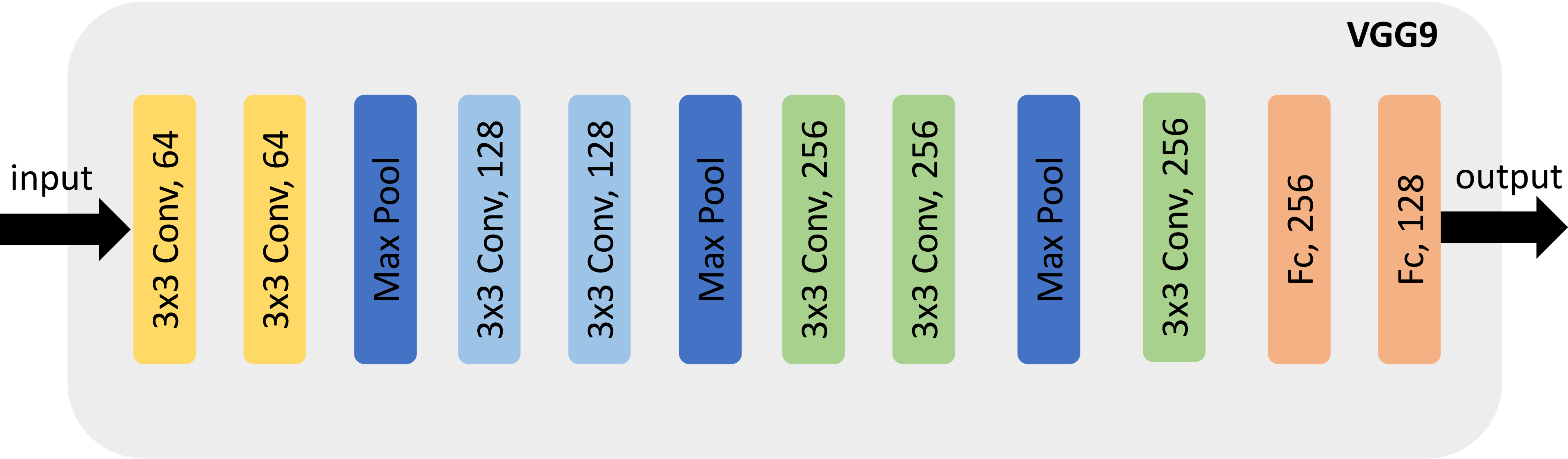
Part1 Report

- Screen Dump
  - Each ONNX Graph
  - Model Accuracy
- Describe your observations and understanding of each transformation

Outline

- Part1: End-to-End FINN Flow for a Simple Convolutional Net
- Part2 : VGG on CIFAR-100
- Part3: Performance improvement
- Report & Submission

VGG9


VGG9
input
3x3 Conv, 64
3x3 Conv, 64
Max Pool
3x3 Conv, 128
3x3 Conv, 128
Max Pool
3x3 Conv, 256
3x3 Conv, 256
Max Pool
3x3 Conv, 256
Fc, 256
Fc, 128
output

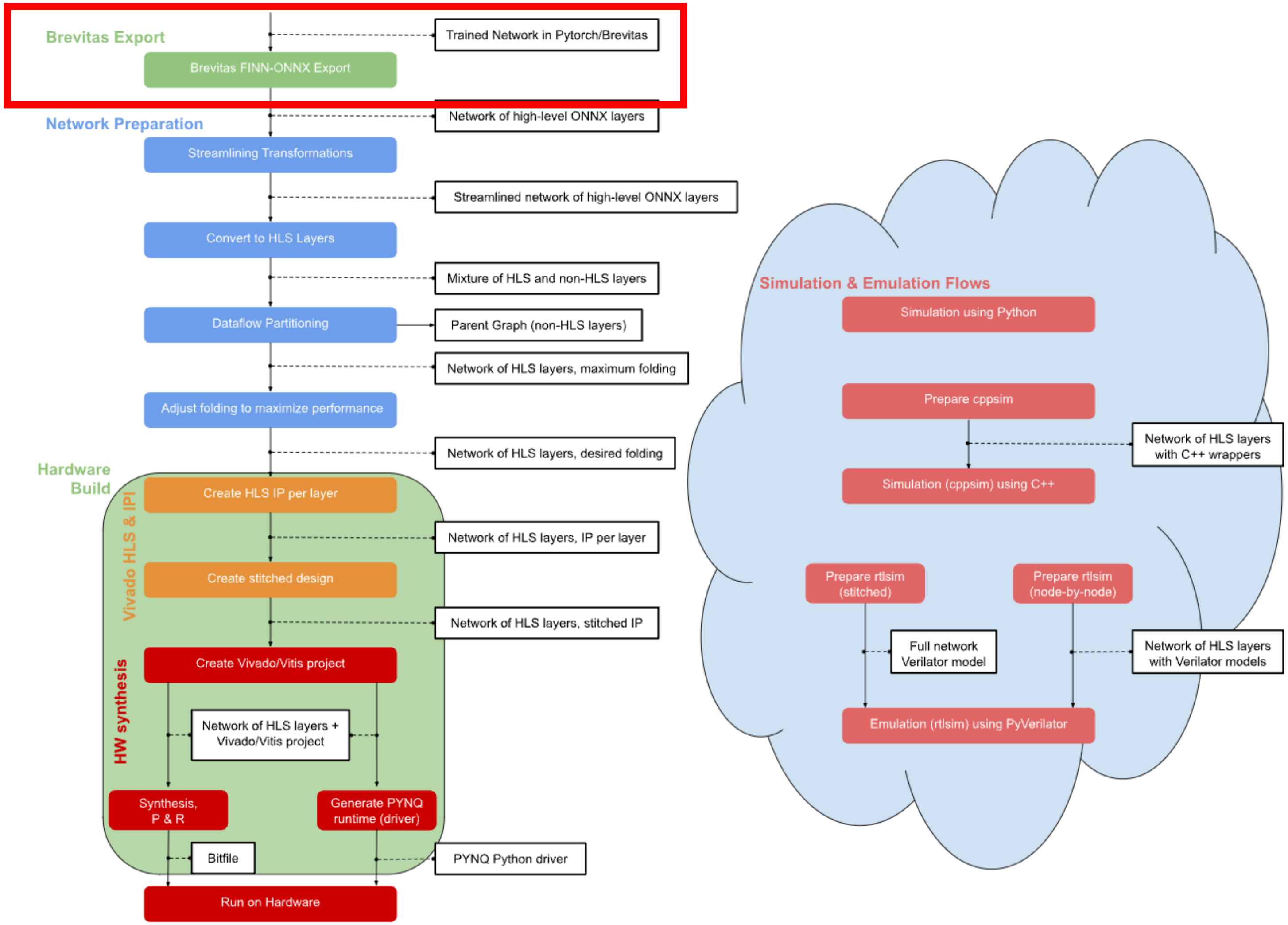
CIFAR100

- It has 100 classes containing 600 images each.
- There are 500 training images and 100 testing images per class.

Classes
beaver, dolphin, otter, seal, whale
aquarium fish, flatfish, ray, shark, trout
orchids, poppies, roses, sunflowers, tulips
bottles, bowls, cans, cups, plates
apples, mushrooms, oranges, pears, sweet peppers
clock, computer keyboard, lamp, telephone, television
bed, chair, couch, table, wardrobe
bee, beetle, butterfly, caterpillar, cockroach
bear, leopard, lion, tiger, wolf
bridge, castle, house, road, skyscraper
cloud, forest, mountain, plain, sea
camel, cattle, chimpanzee, elephant, kangaroo
fox, porcupine, possum, raccoon, skunk
crab, lobster, snail, spider, worm
baby, boy, girl, man, woman
crocodile, dinosaur, lizard, snake, turtle
hamster, mouse, rabbit, shrew, squirrel
maple, oak, palm, pine, willow
bicycle, bus, motorcycle, pickup truck, train
lawn-mower, rocket, streetcar, tank, tractor

https://www.cs.toronto.edu/~kriz/cifar.html

Model Prepare



Define Network Architecture

- We need to define our network architecture in CNV.py
  - Set the correct padding numbers for each conv layer

Change the following simple Network example to VGG9

```
CNV_OUT_CH_POOL = [(64, False,1), (64, True,1), (128,False,1), (128, True,1), (256,False,1), (256,True,0),(256,False,0)]
INTERMEDIATE_FC_FEATURES = [(256*1, 128)]
LAST_FC_IN_FEATURES = 128
LAST_FC_PER_OUT_CH_SCALING = False
POOL_SIZE = 2
KERNEL_SIZE = 3
```

Zero-padding

- You will find out the network define is very similar to PyTorch when using Brevitas.

CNV.py

Specify Dataset

- Change Dataset from CIFAR-10 to CIFAR-100

```
from torchvision import transforms
from torchvision.datasets import MNIST, CIFAR100
```

```
    if dataset == 'CIFAR10':
        train_transforms_list = [transforms.RandomCrop(32, padding=4),
                                 transforms.RandomHorizontalFlip(),
                                 transforms.ToTensor()]
        transform_train = transforms.Compose(train_transforms_list)
        builder = CIFAR10

    elif dataset == 'MNIST':
        transform_train = transform_to_tensor
        builder = MirrorMNIST
    else:
        raise Exception("Dataset not supported: {}".format(args.dataset))
```

trainer.py

Configuration File

- Create a configuration file vgg_1w1a.ini for VGG9
- Bit width of weigh and activation is 1. (Binary Network)

cfg > vgg_1w1a.ini

```
[MODEL]
ARCH: CNV
PRETRAINED_URL: ""
EVAL_LOG: ""
DATASET: CIFAR100
IN_CHANNELS: 3
NUM_CLASSES: 100

[QUANT]
WEIGHT_BIT_WIDTH: 1
ACT_BIT_WIDTH: 1
IN_BIT_WIDTH: 8


```

Command

- Training
  - **python3 bnn_pynq_train.py --network VGG_1W1A**
- Testing
  - **python3 bnn_pynq_train.py --evaluate --network VGG_1W1A --resume ./ experiments /VGG_1W1A_xxxxxx/checkpoints/best.tar**
  - **Note that for this model, it's normal to train with only around 50% top1 testing accuracy. (You can try other bit widths and share @ GITHUB DISCUSSIONS)**

FINN Compile

- Go to the same jupyter notebook as Part 1
  - Change the .onnx file to your VGG best.onnx output

1. Brevitas Export, FINN Import and Tidy-Up

Similar to what we did in the TFC-w1a1 end-to-end notebook, we will start by exporting the pretrained CNV-w1a1 network to ONNX, importing that into FINN and running the "tidy-up" transformations to have a first look at the topology.

In [2]:
```
import onnx
from finn.util.test import get_test_model_trained
import brevitas.onnx as bo
from qonnx.core.modelwrapper import ModelWrapper
from qonnx.transformation.infer_shapes import InferShapes
from qonnx.transformation.fold_constants import FoldConstants
from qonnx.transformation.general import GiveReadableTensorNames, GiveUniqueNodeNames, RemoveStaticGraphInputs

# cnv = get_test_model_trained("CNV", 1, 1)
# bo.export_finn_onnx(cnv, (1, 3, 32, 32), build_dir + "/end2end_cnv_w1a1_export.onnx")
# model = ModelWrapper(build_dir + "/end2end_cnv_w1a1_export.onnx")
model = ModelWrapper(build_dir + "/best.onnx")
model = model.transform(InferShapes())
model = model.transform(FoldConstants())
model = model.transform(GiveUniqueNodeNames())
model = model.transform(GiveReadableTensorNames())
model = model.transform(RemoveStaticGraphInputs())
model.save(build_dir + "/end2end_cnv_w1a1_tidy.onnx")
```

FINN Compile

- Change the folding factors respectively

This is the previous part 1 model with 6 conv + 3FC s
Now, VGG9 we have 7 Conv + 2FCs, and the PE & SIMD should be set s.t. the II of each layer is the same.

(However, there are some rules for setting them)

```
model = ModelWrapper(build_dir + "/end2end_cnv_w1a1_dataflow_model.onnx")
fc_layers = model.get_nodes_by_op_type("MatrixVectorActivation")
# each tuple is (PE, SIMD, in_fifo_depth) for a layer
folding = [
    (16, 3, 128),
    (32, 32, 128),
    (16, 32, 128),
    (16, 32, 128),
    (4, 32, 81),
    (1, 32, 2),
    (1, 4, 2),
    (1, 8, 128),
    (5, 1, 3),
]
```

FINN Deploy

XILINX FINN

- The same as Part1 (However, dataset change to cifar100)
  - Modify the validate.py and add cifar100 in remote PYNQ.

```
    elif dataset == "cifar10":
        from dataset_loading import cifar

        trainx, trainy, testx, testy, valx, valy = cifar.load_cifar_data(
            dataset_root, download=True, one_hot=False
        )
    elif dataset == "cifar100":
        from dataset_loading import cifar

        trainx, trainy, testx, testy, valx, valy = cifar.load_cifar_data(
            dataset_root, download=True, one_hot=False, cifar10=False
        )
    else:
        raise Exception("Unrecognized dataset")
```

- Change the dataset option from cifar10 -> cifar100

```
In [36]: # ! ssh {options} -t {username}@{ip} -p {port} 'cd {target_dir_pynq}; echo {password} | sudo -S python3.6 validate.py
         print("Please paste the below command into pynq terminal")
         print("========================================================")
         print("su ")
         print("python3 validate.py --dataset cifar100 --batchsize 1000")

         Please paste the below command into pynq terminal
         ========================================================
         su
         python3 validate.py --dataset cifar100 --batchsize 1000
```

```
Final accuracy: 41.940000
```

Currently, this model drops 10% acc.

BOL edu

Part2 Report

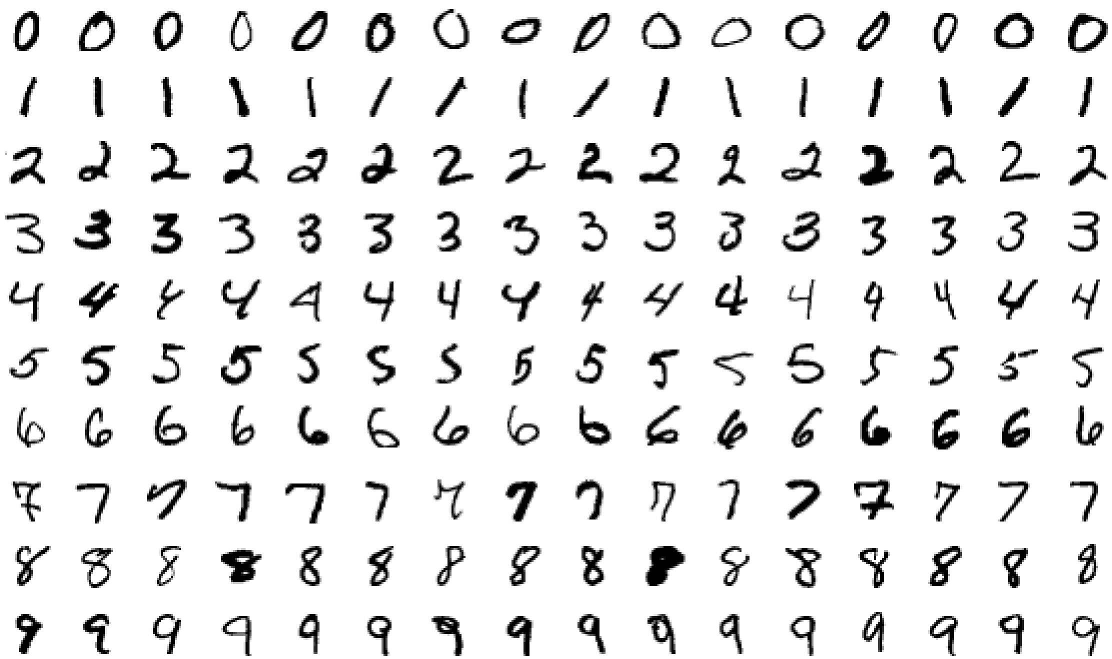
- File
  - vgg_w1a1.onnx
- Screen Dump
  - Accuracy of this model on both Server and FPGA

Outline

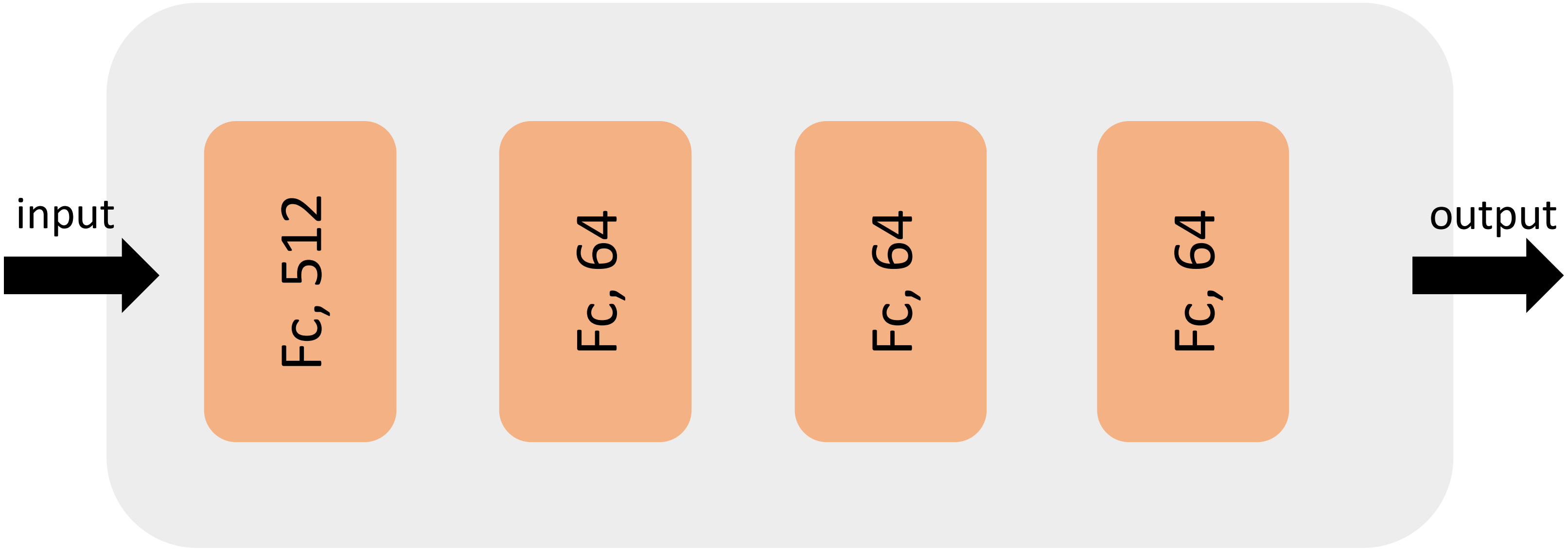
- Part1: End-to-End FINN Flow for a Simple Convolutional Net
- Part2 : VGG on CIFAR-100
- Part3: Performance improvement
  - finn/notebooks/end2end_example/bnn-pynq/tfc_end2end_example.ipynb
- Report & Submission

Handwritten Digit Classification

- Fully-connected network trained on the MNIST data set



Network Architecture



Folding: Adjusting the Parallelism

- We can set folding factors for each layer, controlled by the **PE** (parallelization over outputs) and **SIMD** (parallelization over inputs) parameters.

```
fc_layers = model.get_nodes_by_op_type("StreamingFCLayer_Batch")
# (PE, SIMD, in_fifo_depth, out_fifo_depth, ramstyle) for each layer
config = [
    (16, 49, 16, 64, "block"),
    (8, 8, 64, 64, "auto"),
    (8, 8, 64, 64, "auto"),
    (10, 8, 64, 10, "distributed"),
]
for fcl, (pe, simd, ififo, ofifo, ramstyle) in zip(fc_layers, config):
    fcl_inst = getCustomOp(fcl)
    fcl_inst.set_nodeattr("PE", pe)
    fcl_inst.set_nodeattr("SIMD", simd)
    fcl_inst.set_nodeattr("inFIFODepth", ififo)
    fcl_inst.set_nodeattr("outFIFODepth", ofifo)
    fcl_inst.set_nodeattr("ram_style", ramstyle)

# set parallelism for input quantizer to be same as first layer's SIMD
inp_qnt_node = model.get_nodes_by_op_type("Thresholding_Batch")[0]
inp_qnt = getCustomOp(inp_qnt_node)
inp_qnt.set_nodeattr("PE", 49)
```

Throughput Test on PYNQ Board

- FINN provides the **throughput_test_remote** function for the throughput test.
  - What it really does is use the PYNQ library, including **overlays**, **mmio**, … etc, to execute and compute the inference time.

```python
from finn.core.throughput_test import throughput_test_remote

model = ModelWrapper(build_dir + "/tfc_w1_a1_pynq_deploy.onnx")
res = throughput_test_remote(model, 10000)
print("Network metrics:")
for key in res:
    print(str(key) + ": " + str(res[key]))
```

```
Network metrics:
runtime[ms]: 10.43391227722168
throughput[images/s]: 958413.2714850444
DRAM_in_bandwidth[Mb/s]: 751.3960048442748
DRAM_out_bandwidth[Mb/s]: 0.9584132714850445
fclk[mhz]: 100.0
N: 10000
```

Adjusting the Parallelism

- Try to set Layer2 PE and SIMD number to 1 (Exp1)
- Try your best to achieve the best performance by adjusting the PE and SIMD. (Exp2)
  - Try 3 sets (Including results from Exp1) at least
  - List them in a table

Part3 Report

- Screen Dump
  - Model Accuracy (At least 90%)
  - All of your performance results
- List all your results in a table
- Write down your observations and explain possible reasons for this result (Exp1)
- Explain your methodology for adjusting the folding factors and analyze the results. (Exp2)

Outline

- Part1: End-to-End FINN Flow for a Simple Convolutional Net
- Part2 : VGG on CIFAR-100
- Part3: Performance Improvement
- Questions
- Report & Submission

Question1

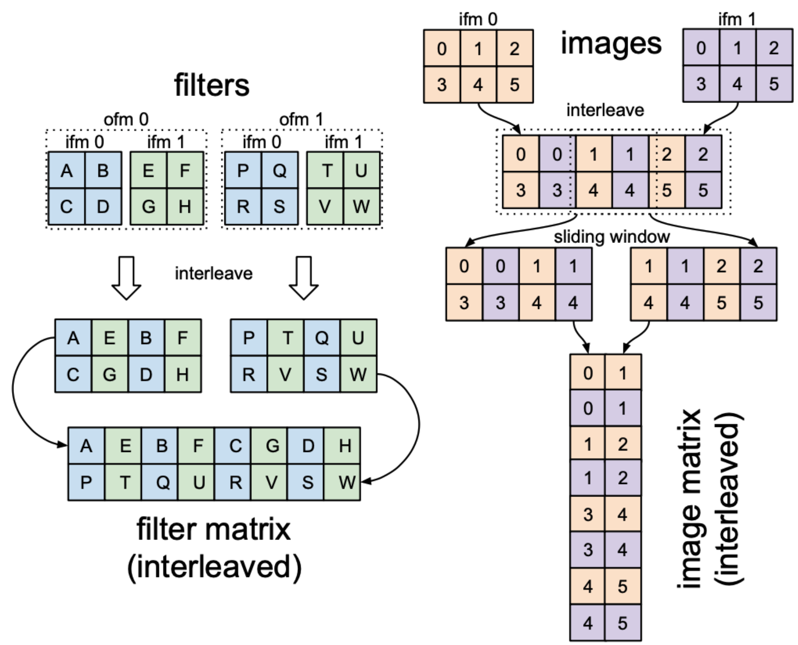
- Please draw the circuit diagrams for the following two PE and SIMD configuration.
  - PE: 4 SIMD: 1
  - PE: 8 SIMD: 8

Question2

- Following the previous question, suppose we need to calculate a 8 x 8 matrix multiplication; please calculate the latency for the two configurations.
- You have to draw some pictures to explain how the calculations are allocated to the hardware.

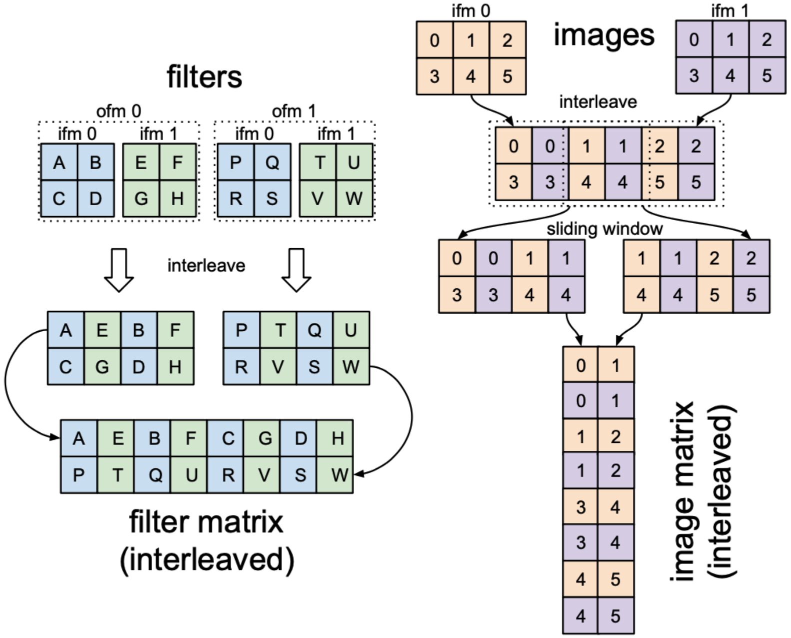
Question3

- For a $H \times W \times C_i$ feature map, and given C_0 stride 1 filters of size $K \times K$, what is the shape of the filter matrix and image matrix? (Assuming the input images are padded for simplicity, i.e. the output resolution is still H x W.)



Question4

- In NN Hardware, the image feature map in the figure is of size ***2 x 3***. However the real application the image might be of size ***224 x 224***. In this case, can we deal with the whole image? Are there any practical solutions in the perspective of hardware design?



Grading

- Part1: 20%
- Part2: 30%
- Part3: 30%
- Questions : 20%

Submission (1/2)

- **Hierarchy:**
  - **GroupID_lab_FINN/**
    - **Part1/**
      - **All of the Screen Dumps**
    - **Part2/**
      - **All of the Screen Dumps**
    - **Part3/**
      - **All of the Screen Dumps**
    - **Question.pdf**
    - **Report.pdf**

The file name of the screen dump should be comprehensible.

Submission (2/2)

- Compress all above files in a single zip file named GroupID_lab_FINN.zip
- Submit: Please ask your TAs where to submit.