# 209AS Lab2

Due date: May 22th midnight

Ver 1.2

**Update log:**

Ver 0.1: Initial version

Ver 0.5: Added Specifications of the Verilog module for part 1, as well as submission format and instructions.

Ver 1.0: Added Part 2. Added positive input assumptions to part 1. Added due date.

Ver 1.1: Updated due date. Fixed the full name of OPS/watt and OPS/mm2 to include the per second. Updated the assumptions of part 1 to avoid confusion. Updated the provided synthesis script for part 1 on the server to fix the library path.

Ver 1.2: Added more detail to the requirement of part 2 (Sign-magnitude representation and requirements on picking the bits from the 8-bit version). Also added detailed report and code requirement for part 2.

**A note regarding submission:**

If you work in groups, only one member needs to submit the codes, and he/she should submit both parts using his/her name and UID. Also, please stick with the member who submitted Lab1 to make grading easier for me, thanks.

# Part 1

This is part 1 of lab 2. In this part, you will write Verilog code to design and synthesize a simple dot product unit, similar to the ones you've seen in Lab 1. You will explore the impact of dot product unit size and accumulation bitwidth on the power efficiency and area efficiency of the design.

You will use Cadence Genus to synthesize your Verilog code to obtain the latency, area, and power of your dot product unit. For those who have taken 201A/D or 216A, this should be pretty straightforward. For those who have no prior experience

dealing with Genus, a tutorial slide has been uploaded to Piazza, where you can learn some Genus basics. The script to synthesize your design will be provided, so what you really need to do is just write the Verilog code, run the synthesis script and check the results.

**Definition of a dot product unit**: A circuit that can perform the computation of a full dot product between two input vectors in one cycle. You can imagine it consists of N multipliers followed by an adder tree to sum up all multiplication results.

**Accumulation bitwidth:** For fixed-point arithmetic, the bitwidth of the multiplication results between two M-bit inputs needs be larger than M to capture the full result. Similarly, the dot product output (the sum/accumulation of N such multiplication results) requires an even larger bitwidth.

However, for a fixed-point neural network setup, usually the output activation bitwidth is same as the input activation bitwidth, which means the bitwidth of the dot product output is less than what required without loss in precision. In this case, truncation is usually performed at the output.

**How to synthesize your design using Cadence Genus**

Please refer to the uploaded genus tutorial slides ***GenusTutorial209as.pdf*** **(especially page 6-8)** to check how you can synthesize your Verilog code, as well as getting the synthesis reports.

You need to copy the materials provided in **/w/class.1/ee/ee209w/ee209wt2/material/lab2/part1** to your working directory, which contains the sample synthesis script. You need to modify a few places in the synthesis script to synthesize your design, please refer to the tutorial slides for details.

## Your task:

**Assumptions/requirements for this part:**

- Both inputs are positive. You don't need to deal with negative numbers.
- When calculating OPS (operations per second) for a dot product operation, the number of operations of a dot product of size N should be 2N. For example, a dot product of size 4 = 4 MAC = 8 OPS.
- You can use the default clock period in the provided synthesis script for OPS calculation if the slack is positive.

- The output need to be registered, while the inputs do not need to be registered.

**(a):** Design a single dot product unit that can perform the dot product operation between two 8-element vectors (8 multiplications). The bitwidth for both input vectors are 8-bit.

You need to write Verilog code for this design, and synthesize it using Genus to obtain timing, power, and area results. **You then need to report the OPS/W (number of operations per second per watt) and OPS/mm2 (number of operations per second per 1 mm2 area) of your design.**

You need to perform the above analysis for 3 cases detailed below:

(i): Dot product output bitwidth is whatever required to get accurate result.

(ii): Dot product output bitwidth is 8 bit, keep all MSBs.

(iii): Dot product output bitwidth is 8 bit, keep all LSBs. In this case, you need to saturate the output if it overflows. For example, if any bit except the last 8 bits is non-zero, then the output will be 8b'11111111.

**(b):** Repeat (a) for a dot product unit with size 128 (128-element vectors), keep everything else the same.

In your report, show the required results for all 3 output bitwidth configuration and 2 dot product size. When comparing the results for different output bitwidth configuration, what do you observe? Please explain your observation.

## Verilog module format:

A skeleton code of the dot product unit module is provided in the material directory (material/lab2/part1/dotproduct_sample.v). The specification of input and output ports are defined in this skeleton module, which will be used in the testbench to evaluate the correctness of your implementation.

The test bench will not be provided, but you can implement your own testbench to verify the correctness of your code.

**Module name format:** The name of the Verilog DPU module should be *DotProductUnit_N_full*, *DotProductUnit_N_msb*, *DotProductUnit_N_lsb* for the three cases, and change N to 8 or 128 for the two DPU sizes. This name convention is also included in the provided skeleton module.

## Submission format:

**For report:** The report should be **6-8 pages slides** (maximum 8 pages) for **part 1 and part 2 combined** (but in PDF format), summarizing your approach and results.

**For code**, you need to submit **6 separate Verilog files**, one for each case. The names should be:

dpu_N_full.v , dpu_N_msb.v , dpu_N_lsb.v

Replace N with 8 or 128 for two DPU sizes.

Put all 6 files into a single tarball, and name the tarball as UID_Lastname_Firstname_Lab2p1_pinXXXX.tar.gz, make sure the permission is configured correctly.

And submit the tarball to '/w/class.1/ee/ee209w/ee209wt2/submission/project2/'.

**Please follow the submission instructions from Lab 1 for full instructions and example.**

# Part 2

In this part, you will implement a custom quantization-aware training (QAT) flow using PyTorch.

For the provided neural network, you need to quantize the weight of each convolution layer in a special way, detailed below using QAT.

## Detailed descriptions

You need and only need to quantize the weights from floating point precision into 8-bit fixed point precision for convolution layers. You don't need to quantize the activations and weights for non-conv layers.

You will perform quantization-aware-training (QAT) in this part, which means you need to retrain the network and perform the quantization during retraining to let the network 'learn' the quantization behavior.

**The goal of your quantization framework is to quantize the weights to 8 bits in a way that the 8-bit weights network accuracy should still be relatively decent if the same weights are truncated to 4-bit. (The same weights can achieve decent accuracy for both 8-bit and 4-bit). You can determine how the weight bits are truncated for the 4-bit cases, but it must be truncated from the 8-bit weights.**

The above goal of your quantization framework means that you cannot just train an 8-bit weight network on its own and then just truncate from it to obtain the 4-bit results. You need to train for this goal, **which means you need to set your goal properly during retraining so that the network can be optimized for this particular goal.**

You need to implement the quantization as fake quantization, which means the underlying data type is still floating-point, only the values are quantized (modified) to correct fixed-point values. The quantization should be implemented in the forward pass of each convolution layer. You are **not** required to store the final retrained weights in fixed-point format.

The definition of the target network you will be using for this part (given as the original training script) and the pretrained weights for the full-precision network are provided (lab2part2.py and lab2p2weights.pth) under "*material/lab2/part2*", so that you don't need to train the original network. You can load the pretrained weights and retrain the network with your quantization function implemented.

**Straight-through estimator (STE):** One problem with quantization during training is that the gradient of a quantization function is not defined. For deep learning frameworks such as PytTorch that supports automatic backpropagation, the gradient cannot flow through the quantization function properly, making the network cannot properly learn the quantization behavior. For quantization-aware training, STE is typically used to work around this problem. STE sets the gradient of quantization function to 1, which essentially bypasses the quantization function during backward pass and making the quantization function trainable. To reduce the complexity of this part, a standard implementation of STE is provided under the material folder (STE_example.py), please refer to this example to implement your own STE-based quantization function.

## Requirements and notes

- **Use sign-magnitude representation for fixed-point (unless you already used other representation, in that case, clearly mention it in your report). When quantizing from 8-bit to 4-bit, you always keep the sign bit and then choose 3 consecutive bits from the rest 7 bits. The three bits doesn't have to be LSB or MSB, it can be whatever 3 consecutive bits. However, you need to keep the selection rule the same for either the entire layer or entire network.**
- You can retrain the network for up to 30 epochs (to avoid excessive training time).
- You can adjust the optimizer, learning rate, and scheduler as you like.
- This is an open-ended project, there is no single correct answer. There are probably multiple ways to achieve this goal.

## What to include in your report

When you make accuracy comparisons, you compare to the inference accuracy of the provided pre-trained weights **without** further training. **You also need to compare your accuracy to the accuracy of QAT for pure 8-bit quantization and pure 4-bit quantization**.

In summary, there are 5 results that need to be reported and compared.

1. Inference accuracy of pretrained FP network
2. Accuracy of QAT for pure 8-bit quantization
3. Accuracy of QAT for pure 4-bit quantization
4. Accuracy of your method for 8-bit
5. Accuracy of your method for 4-bit (need to share the bits with 8-bit version)

You need to also describe your approach to this problem (e.g., how the training flow is designed to optimize for 8-bit and 4-bit at the same time) and how the 4-bit weights are selected from the 8-bit bits.

## What your submitted code should do

Your submitted code should take your submitted weights (in FP format), perform the inference with your quantization flow, and reports (prints out) the 8-bit and 4-bit accuracy. Note that your submitted code should only do inference, not training. The accuracy output should be same as what you reported in the report.

In the end of your code/script, you need to have two lines of print statement like this:

- Accuracy of 8-bit version is xx%
- Accuracy of 4-bit version is xx%

## Submission instructions

**For report:** The report should be **6-8 pages slides** (maximum 8 pages) for **part 1 and part 2 combined** (but in PDF format), summarizing your approach and results.

**For code**, you need to submit both your QAT script as well as the trained weights.

The QAT script should be named lab2p2_qat.py, and the weights should be named as lab2p2_qatweights.pth.

Put both files into a single tarball named as
UID_Lastname_Firstname_Lab2p2_pinXXXX.tar.gz

, and submit the tarball to '/w/class.1/ee/ee209w/ee209wt2/submission/project2/'.

**Please follow the submission instructions from Lab 1 for full instructions and examples.**