Author: Elsie Rezinold Y

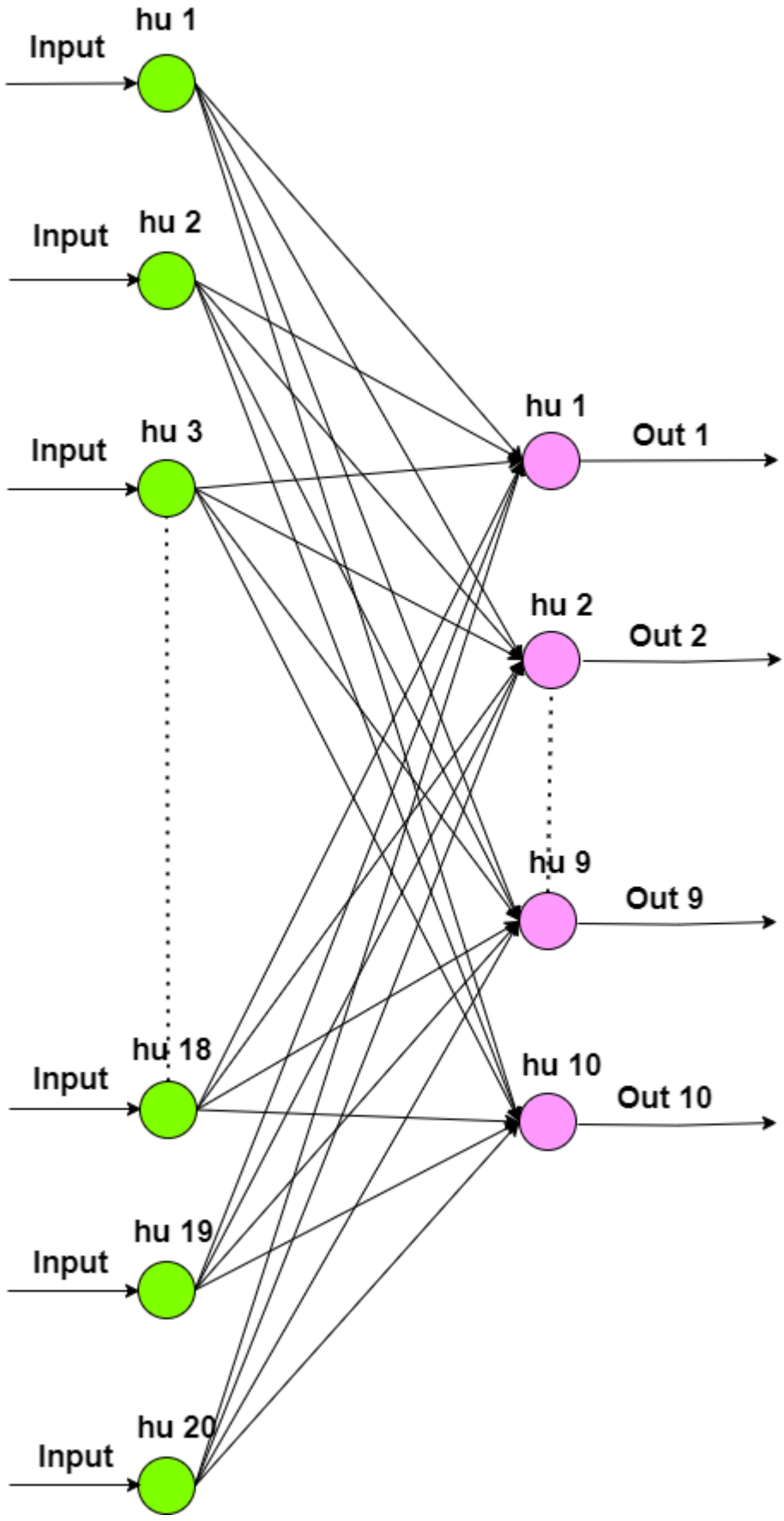
Design details

This project is divided in to two main parts.

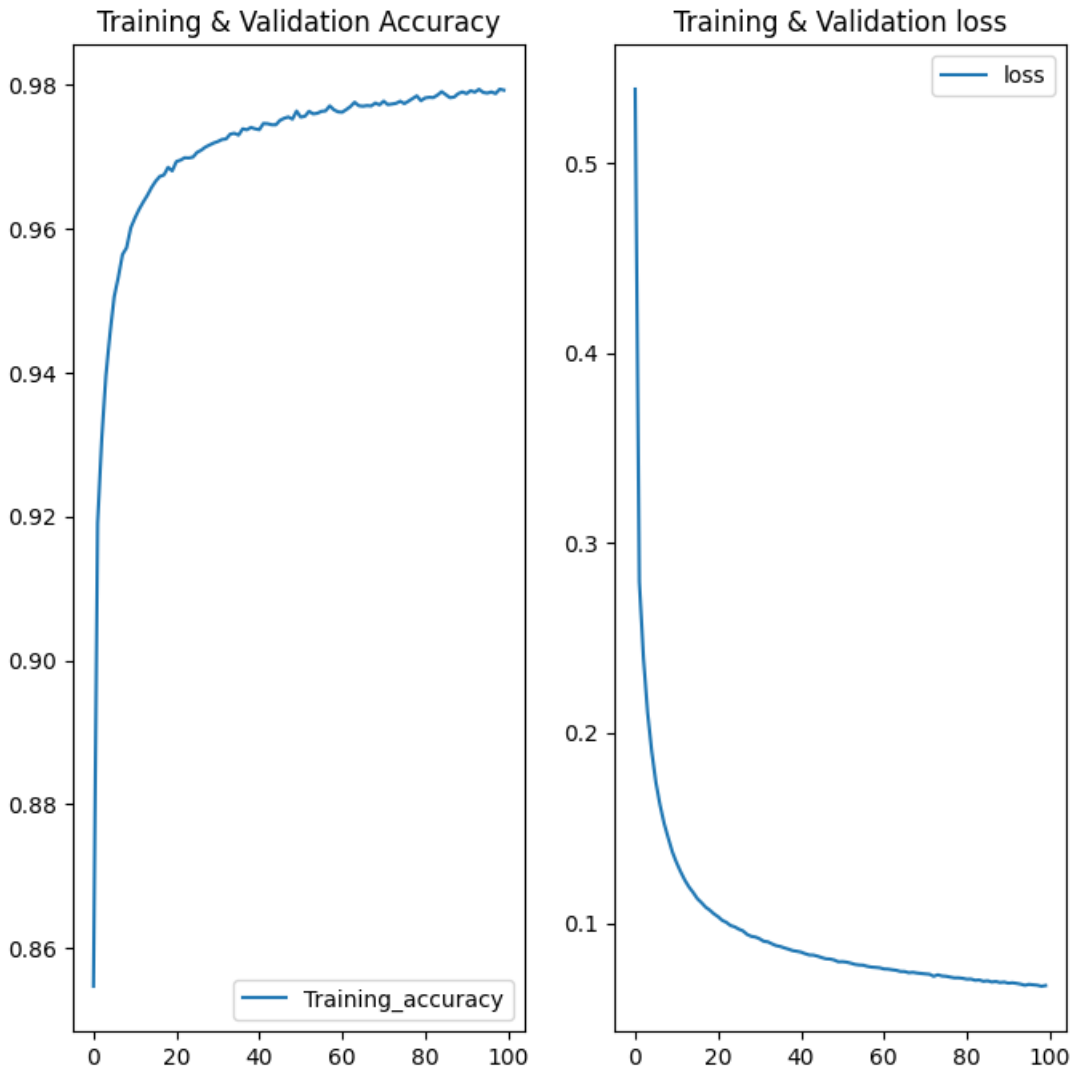
1. Train model
2. Build hardware

Train model:

I chose mnist data set freely available on tensor flow and built a model. I designed two dense layers, first dense layer with 20 hidden units and second with 10 as there are 10 output data classes. In the first layer the activation function is “RELU” for the second one it is “Sigmoid”. Optimizer is “adam”, loss function is “Sparse categorical crossentropy”. There are 60,000 images available for training and 10,000 for testing. The images are originally of size 28 x 28 but I have formatted it to 16 x 16. I have divided the images in to a batch size of 32. The model is trained for 100 epochs. The image data is divided by 255, which is the maximum pixel size. So all the image data that is 256 pixels are valued between 0 and 1.



The above figure depicts our model.



The above graph shows how the accuracy improved and loss came down during 100 epochs.

Model	Accuracy	Loss
Training Images	97.93%	0.0672
Test Images	96.38%	0.1517

My model has the above metrics.

The whole python note book is uploaded in “dl_model” folder. Tensorflow libraries have been used.

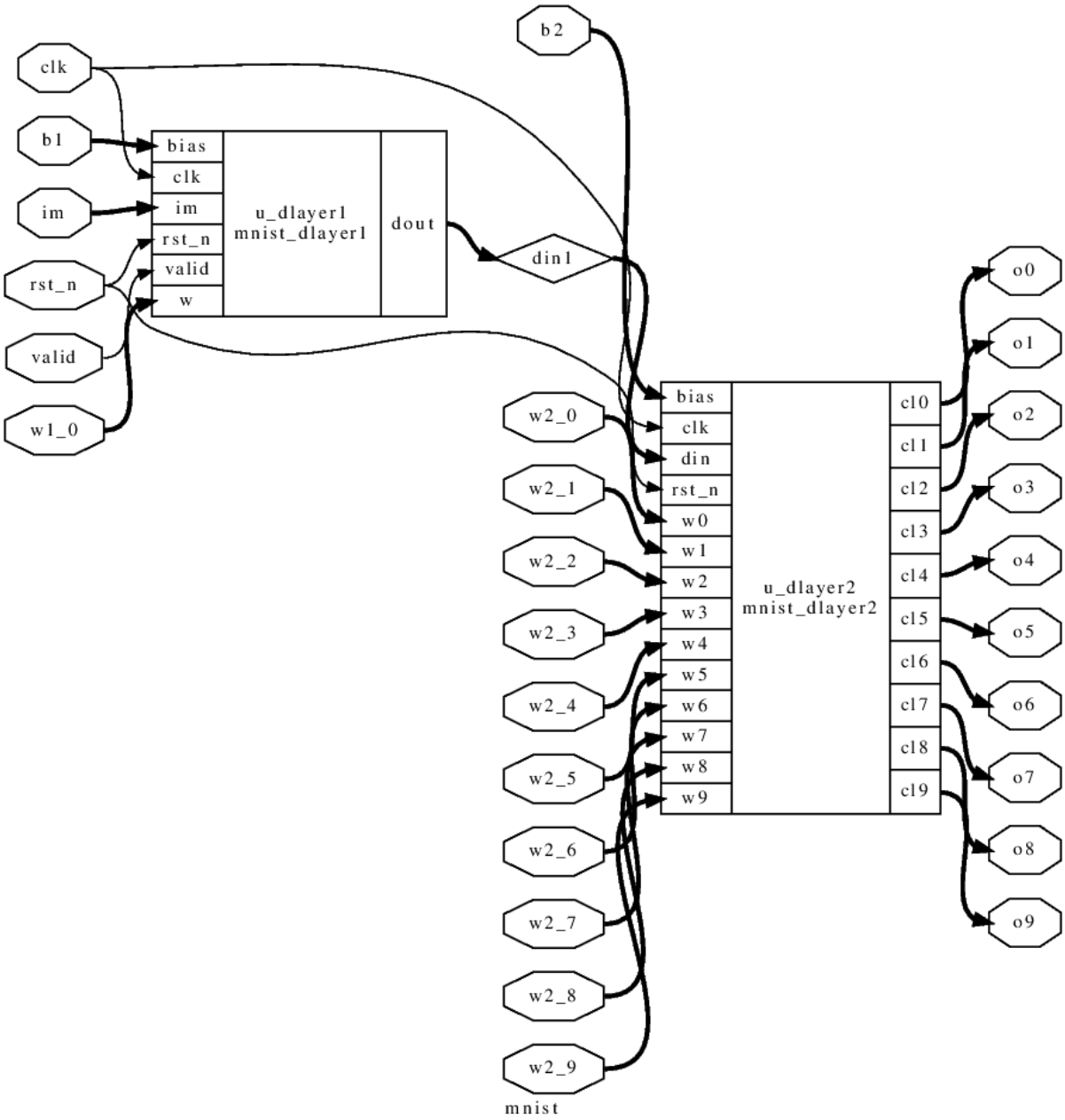
Build Hardware:

The whole hardware model mimicked the software model except for few hardware decisions.

The top module “mnist.v” has two dense layer modules. The whole incoming data is in float point format. This hardware expects inputs, weights and biases in ieee745 32-bit format. For layer1 we need 20 x 256 float point multipliers and 20 x 255 float point adders and 20 relu units. To keep the hardware minimum, we implemented only one hidden unit and reused that one hidden unit 20 times in 20 cycles to get the desired 20 outputs for layer1. Through pipelining the number came down to 256 float point multipliers and 255 float point adders and 1 relu unit.

Hardware Model	Input(bits)	Output(bits)
Mnist.v	256 x 32bits	10 x 32bits
Mnist_dlayer1.v	256 x 32bits	20 x 32bits
Mnist_dlayer2.v	20 x 32bits	10 x 32bits

For layer2 we need 10 x 20 float point multipliers and 10 x 19 float point adders and 10 sigmoid units. But, to keep the hardware simple we omitted the sigmoid unit. So, in layer 2 module there are only 10 x 20 float point multipliers and 10 x 19 float point adders.



The above figure gives an idea of the top-level module.

The module expects weights and biases as inputs along with the image data.

Going deeper in to micro architecture details, layer 1 takes 6 cycles to compute one hidden unit output. Since we have only one hidden unit implemented, this one hidden unit runs for 20 cycles to get all the 20 outputs. After all the outputs are generated, they are sent in to the layer2 at one time. In layer 2 we

have 10 hidden units are available in parallel. So, after the inputs arrive, we need only 6 cycles to get the final output.

Testing hardware:

First of all, the final weights, biases and image inputs needed are extracted in ieee745 32-bit format from the trained model, the process can be found in the same python notebook. Since, there is only one hidden unit available in layer1 the 20 weights and biases have to send every new clock sequentially as shown in testbench available in “v_tst” folder. But, layer 2 weights and biases can be sent together since all the units are available.

10,000 images data is sent and tested. It should be noted that since we are not performing sigmoid in hardware so only the pre-sigmoid output is tested i.e compared with the software output. The pre-sigmoid output for the 10,000 images is also extracted in the same python notebook, the technique can be found there.

The verification results show that out of 10,000 images. 9769 images hardware output is matching that of software’s. My best guess would be the other 231 images output image mismatch is due to float point arithmetic precision difference in hardware and software.

Find the report in the same test bench folder. The image, weights and biases files are not uploaded due to big size.

Netlist:

Netlist is generated using yosys tool using opensource liberty file. Please find it in the “synthesis” folder.