Computer Aided Design (CAD) -Assignment Description
University of Tehran
Electrical and Computer Engineering Department
Fall 00

Hardware implementation of a Multi-Layer Perceptron (MLP) Neural Network

In this project, you have to design and implement a simple MLP artificial neural network in Verilog. The neural network can approximate a complex function or classify the MNIST dataset.

- **INTRODUCTION**

In this project, the goal is to implement a simple MLP (Multi-Layer Perceptron) model of neural networks. To test the design, you can use Matlab or any other tool (like TensorFlow) that generates neural networks from an input training data. Alternatively, you can use a neural network for image classification based on the MNIST dataset that we provide you.

Multi-Layer Perceptron (MLP) - The most common form of neural networks is the Feed-Forward Multi-Layer Perceptron (MLP). A feed-forward neural network is an ANN wherein connections between the neurons do not form a cycle. An n-layer MLP consists of one input layer, n-2 intermediate (hidden layers), and one output layer. Each layer consists of a set of basic processing elements or neurons. An individual neuron is connected to several neurons in the previous layer, from which it receives data, and also it is connected to several neurons in the next layer, to which it sends data. Except for the input nodes, each neuron uses a nonlinear activation function.

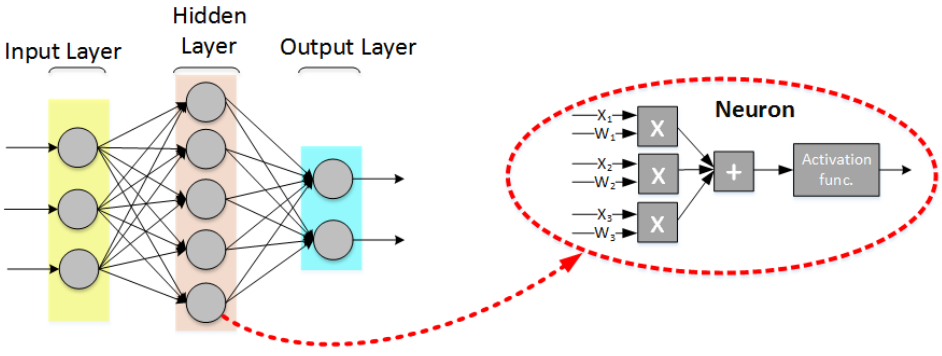

Figure 1. An MLP neural network with one hidden layer

Figure 1 illustrated the structure of such a network with one hidden layer. Consequently, all neurons in any given layer i receive the same set of inputs from layer i-1. Associated with each input, each neuron keeps a weight that specifies the impact of the input in the final output.

The output of each neuron in hidden layers is the weighted sum of the neuron inputs, and it is presented as input to the next layer after passing through a nonlinear function. The data is finally propagated to the

output layer after passing through one or more hidden layers. So, in hidden and output layers, each neuron has the computation formula as Relation.1.

$$y_j = f(\sum_{i=1}^{n} W_{ij} \times x_i + b_j)$$

x_i and y_j are the input and output values of the neuron. W_{ij} and b_j shows the value of neuron weights and biases, respectively. Function **f (.)** represents the activation function. The most common activation function is ReLU: it converts negative numbers to zero, but keeps the positive numbers unchanged.

- **PROJECT DESCRIPTION**

In this project, you have to design and implement a simple MLP neural network in hardware.
The neural network hardware is implemented as a set of processing units (PUs) and a controller. Each PUs is a multiply-and-accumulate unit that has eight inputs. It has eight multipliers, an adder tree, and an activation function. Figure 2 shows the architecture.

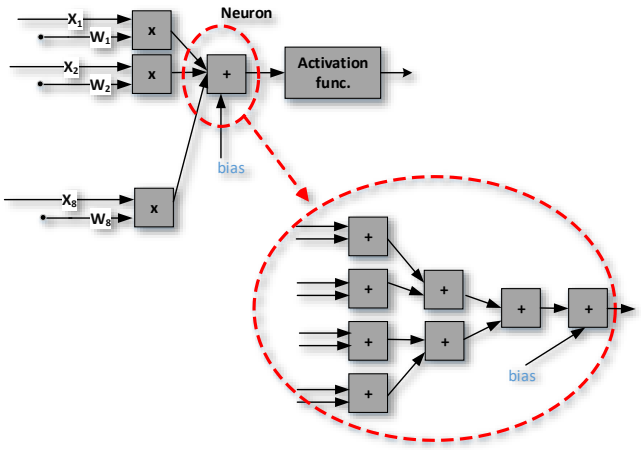

Figure 2. PU structure

The design works by a clock. Assume that the computation of the PU requires four steps: reading data from memory, multiplier, the adder tree + activation, and writing the results to the registers. Each step takes one clock cycle.

- The hardware is composed of a controller and eight PUs. The controller allocates a single PU to each neuron by sending the appropriate weights and inputs to PUs. In this scheme, if a neuron has more than eight inputs, the computations of this neuron must be executed **serially** on the hardware (running the computations of 8 inputs in parallel each time). Then the partial results are summed to generate the final output.
- Since there are eight PUs, at most eight neurons can be run in parallel.
- The controller is responsible for arranging the data assignment to the hardware neurons in this design.
- The computations of the neural network layers (hidden and output) should be executed serially.

- In this project, you should implement the design based on access to limited hardware resources. Consider the case that only **8 hardware neurons** are provided as resources to perform all network computations (Figure 3).
- For **MAC** unit, you can simply use the high level multiplication(a*b) in the Verilog code in this project (**Note that** to multiply two 8-bit sign-magnitude numbers, you should use the 7 * 7 unsigned multiplication and handle the sign-bit separately.)
- As mentioned before, to make better use of resources, all network computations should be performed using the limited number of Neurons (multiply-accumulate units (MACs)). Therefore, it is required to design a system **controller** and specify a **memory access pattern** (read and write).
  Use separate **single port memory buffers** for input data and network parameters (weights and biases). Also to save the middle data (the output values of a layer that are inputs of the next layer) you can use **registers. (See Figure 3)**

- **Optional: you will receive 10% bonus if you design the PU as a 4-satge pipeline.**

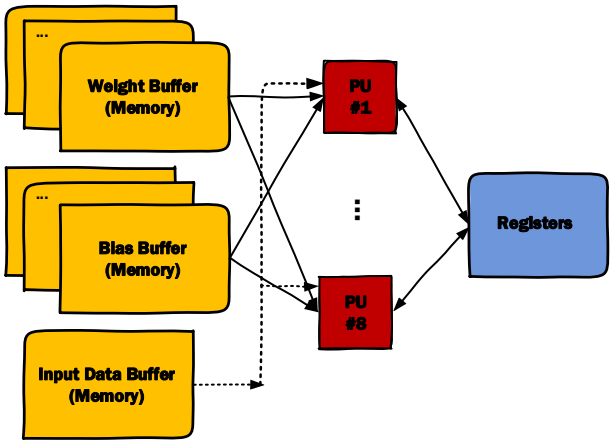

Figure 3. Resources for hardware implementation of neural network in this project

Note that

- MLPs can have one or more hidden layers. Most MLPs generated by Matlab has one hidden layer. The MNIST neural network we provide for you has also one hidden layer. If you train your own network by Matlab, make sure that the hidden layer has more than 8 neurons.
- Weights and biases are 8-bit with sign-magnitude representation.
- In the MAC structure each 8-bit weight must be multiplied by the corresponding 8-bit input, and each 8-bit bias value must be multiplied by +127(8-bit sign-magnitude) and then added to the sum of the weight-input products.
- You should Consider 15 bits for multiplier output, 21 bits for MAC accumulator and 8 bits for the output of ReLU Function.
- The output of the hidden layers must shift 9 bits to the right before entering the activation function (Due to the scaling of the output layer inputs).
- For passing the accumulator output through the activation function if the result value exceeds from 8 bits, the saturation operation should be performed. If a register has a **saturation** mode of operation, then an **overflow** and **underflow** condition is set to the maximum positive or negative

value that are allowed (for example +127 and -127 in 8-bit sign-magnitude representation). It is also necessary to handle the sign bit correctly.

Deliverables:

- The **complete Verilog code** of the design
- A **testbench** that instantiates the design as a component and feeds it with a clock, and monitors its output
- You should assess the following and mention them in the report
    - The functionality and **accuracy** of the network based on the test data
- *A **comprehensive report** of the project's steps, a detailed block diagram of your design (Controller and Datapath structures), and the results.

Good luck :)