Computer Aided Design (CAD) -Assignment Description

University of Tehran
Electrical and Computer Engineering Department
Fall 98

CA2: Hardware implementation of a basic "Neuron" in Artificial Neural Networks (ANNs)

In this assignment, you have to implement a pre-designed model of a basic artificial neuron which is used in ANNs by using VHDL language.

- **INTRODUCTION**

An Artificial Neural Network (ANN) is a computational model based on the structure and functions of biological neural networks. Information that flows through the network affects the structure of the ANN because a neural network changes - or learns, in a sense - based on that input and output. ANNs are considered nonlinear statistical data modeling tools where the complex relationships between inputs and outputs are modeled or patterns are found. An ANN is based on a collection of connected Basic units called "Artificial Neurons".

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 several neurons in the next layer, to which it sends data. Figure 1 shows an example of a 3-layer neural network with 3 inputs, 5 neurons in the hidden layer and 2 neurons at the output 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.

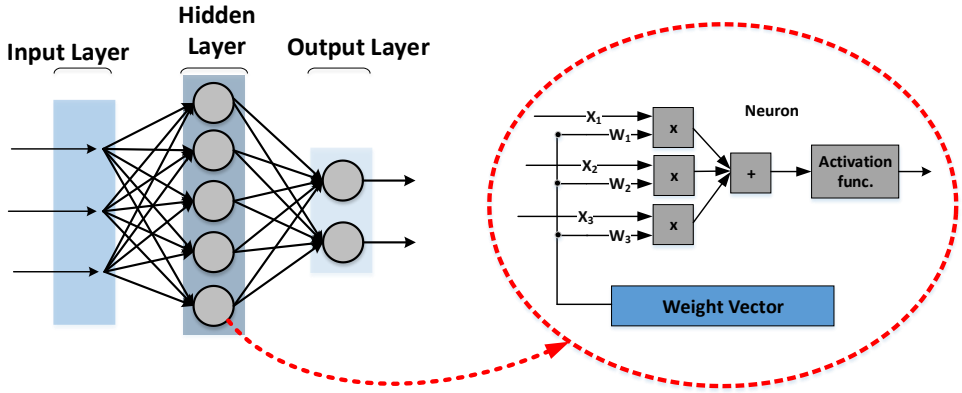

Fig1. A three-layer feed-forward MLP

- **PROJECT DESCRIPTION**

In this assignment, you should implement a single neuron, depicted in Figure 2. Each neuron in MLP computes the dot product of the inputs and weight vectors and passes the result to an activation function to produce the final output. The weight vector of each neuron is determined during an offline or online training phase. Figure 2 illustrates the required arithmetic operations of a basic neuron, but realistic implementations usually use a **MAC** (multiply-and-accumulate) unit for each neuron to carry out the multiplication/addition operations in serial. Figure 3 shows the implementation of a neuron by applying a MAC module.

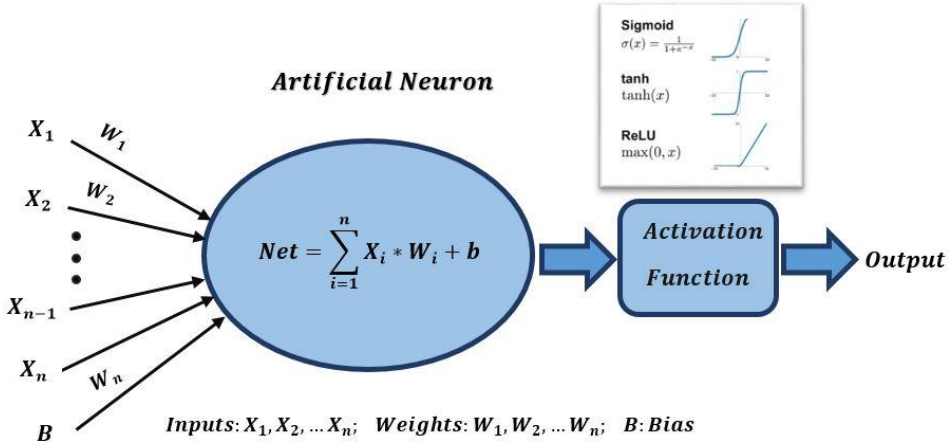

Fig2. A basic artificial neuron model

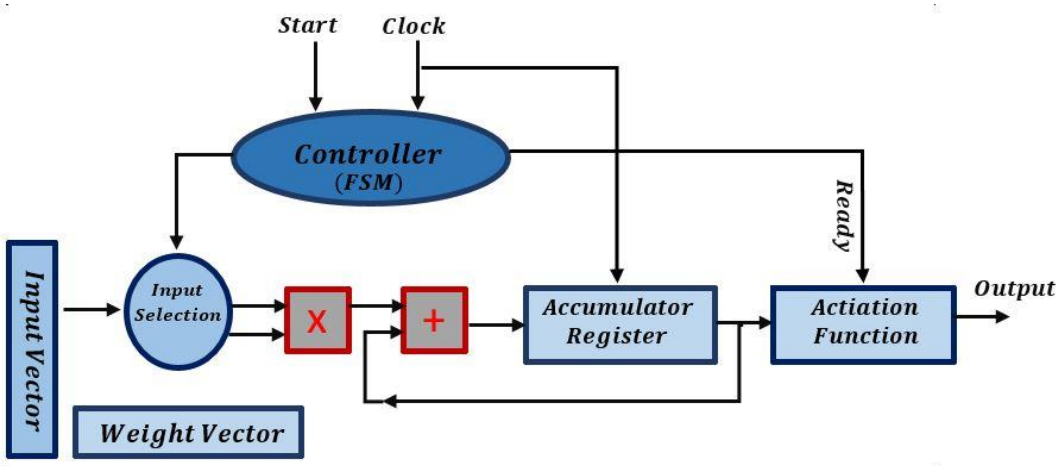

Fig3. A structural model of an artificial neuron

As illustrated in Figure 3, initiates computation by receiving a start signal. The controller then reads each input data and its corresponding weight and passes them to the MAC unit in several iterations. Once the MAC calculation is completed, a ready signal is passed to the activation function unit to generate the neuron output.

Activation function for this step of the project is a simple threshold-based function (ReLu): if the MAC output is greater than 0.5, the MAC result is directly passed to the output, otherwise the neuron output is zero

$$\textbf{Activation Function}: \quad f(Output) = \begin{cases} Output & Output > 0.5 \\ 0 & Output \leq 0.5 \end{cases}$$

Note that *

- The weight register, controller, adder, and multiplier must be constructed separately as different entities and then be integrated into a structural design.
- For the Multiplier unit in MAC Module you should use
    - An accurate 8*8 multiplier using LUTs
    - An approximate 8*8 multiplier design that is described in the next part ***(Figure 4*)***.
- The number of neuron inputs in Figure 3 must be parameterized. Set it to a sample integer value when you are testing the design.
- Assume one MAC operation (one multiply and one add) can be completed in a single clock cycle.
- Assume that adder and multiplier operate in range between -1 to +1 with the same length.
- Fill weight and input vectors with random numbers that range between -1 to +1 (WL=1+8; 1-bit sign,
- 8-bit Fixed point binary value)).
- Design the controller by an FSM

FPGA Optimized Approximate Multiplier (8*8). In CA1 you implemented a 4*4 approximate multiplier, now you should extend your design to create an 8*8 approximate multiplier. Figure 4 shows how four 4*4 multipliers can be merged to make an 8*8 multiplier.

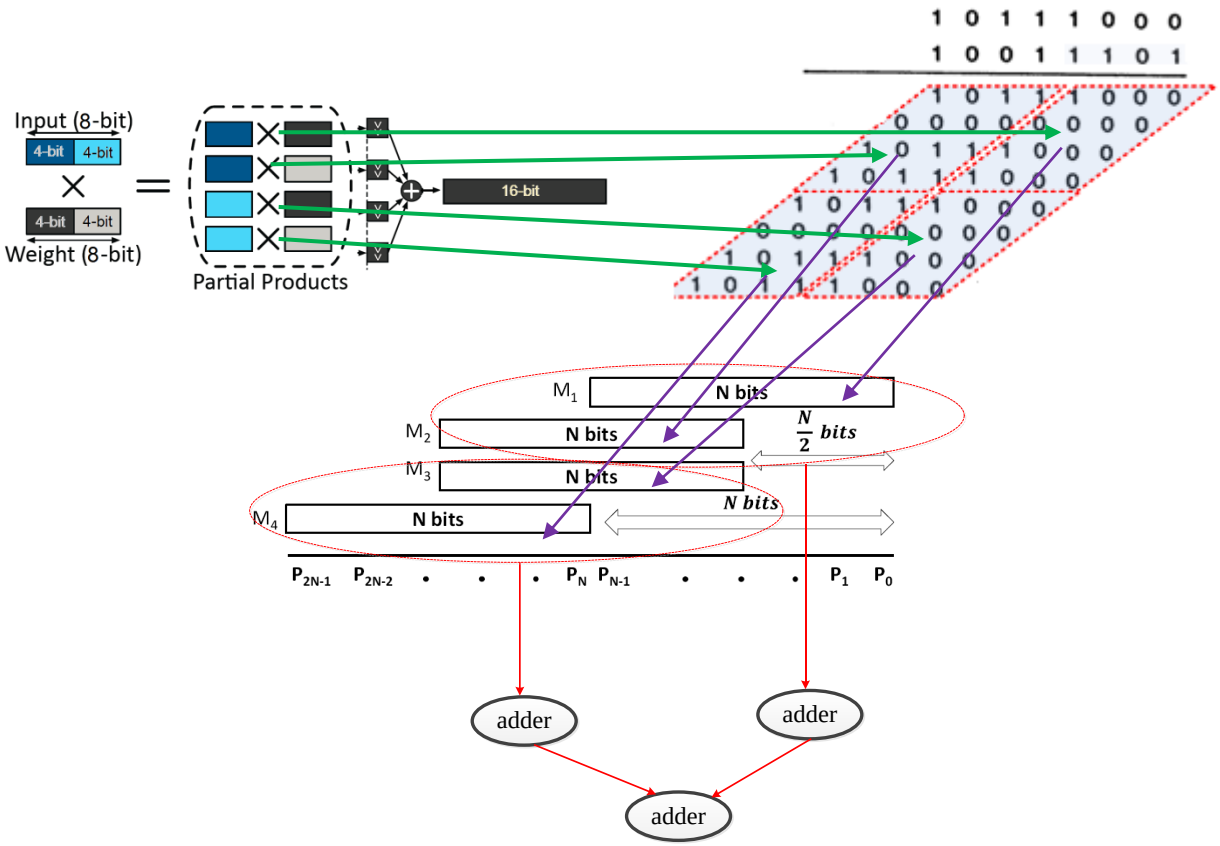

Figure 4. Making 8*8 multiplier with four 4*4 multipliers

This way, you should use 4*4 multipliers to multiply high 4-bit and low 4-bit parts of each 8-bit operand separately and them shift and them up to make the 8*8 result.

Multiplying signed numbers. To support both positive and negative numbers by the multiplier, you should add an extra sign bit to the leftmost of each 8-bit operands to have 9-bit sign-magnitude representation of them. "0" indicates a positive integer, and "1" indicates a negative integer. (For example in sign-magnitude method (1 0001 1000) is a representation of (-24). The sign "1" means negative and the magnitude is 24 in 8-bit binary value). You should handle the sign of the result value in the 8*8 multiplier by using logical gates.

Deliverables:

- *The **complete VHDL code** of the design and the Vivado synthesis and implementation results (Choose your target FPGA in the synthesis tool as **Artix7-series** family.)*
- *A **testbench** that instantiates the design as a component and feeds it with a clock and monitors its output*
- *A **comprehensive report** of the steps involved in the project, detailed block diagram of your design and the results (*If you do not send the report, you will lose a large percentage of the score)*
    - *The report also must include the FSM of the controller unit*

Good luck :)