COL215P
Assignment 2 - Part 1
Machine Vision Through Neural Network Hardware

Shreya Arora(2019CS10401)
Ishita Chaudhary (2019CS10360)

The implementation for developing a 3-layer Multi-layer Perceptron (MLP) for the vision task consists of the following six subcomponents in the datapath:

1. Multiplier Accumulator Component (MAC): To perform vector-matrix multiplication
2. Read Write Memory (RWM or Random Access Memory RAM): To store outputs of all layers
3. Read Only Memory (ROM): To store parameters and the input image
4. Registers: To store intermediate results across clock cycles
5. Shifter: To perform division by a factor of 32
6. Comparator: To implement ReLU (Rectified Linear Unit)

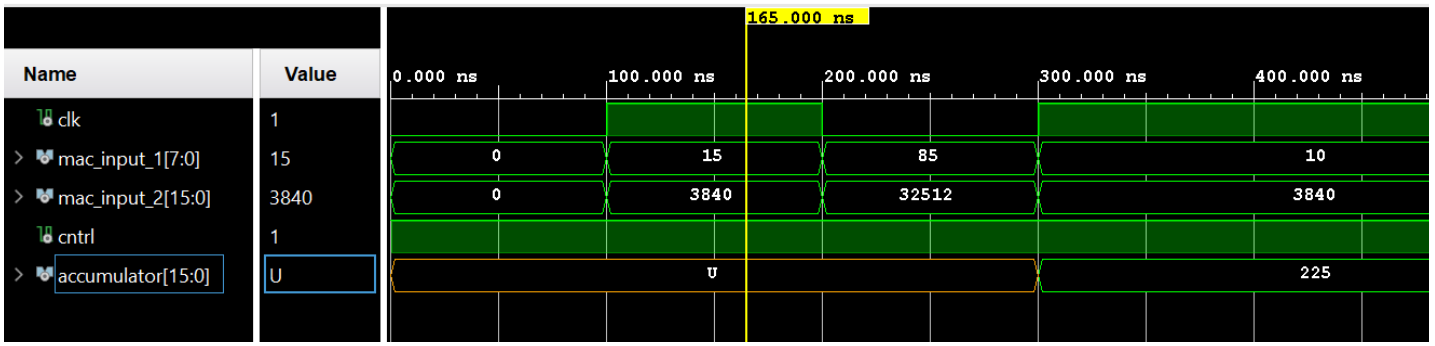
Component 1: Multiplier Accumulator Component (MAC)

The Multiply Accumulate Block (MAC) consists of a 16x8 multiplier, a control register, and a 16-bit adder to accumulate all the products. The 16x8 multiplier is used for multiplying the 8-bit weight with the 16-bit input/activation.

For the 16x8 multiplier, we have 2 inputs - *'mac_input_1'* (8-bit binary vector) and *'mac_input_2*' (16-bit binary vector). We multiply these inputs to get the *'product_vec',* a 24-bit *vector*.

The *'cntrl'* input signal signifies if it is the first product of the compute or not. If it has the value 1 and it is the first product, then the *'accumulator'* is assigned the value of the product_vec, truncated to 16 bits, by taking the leftmost 16 bits of product_vec. Else if the *'cntrl'* is 0, product_vec (truncated to 16 bits in a similar manner) gets added to the existing value of the accumulator and gets stored as the updated value of the accumulator.

Simulated Waveforms



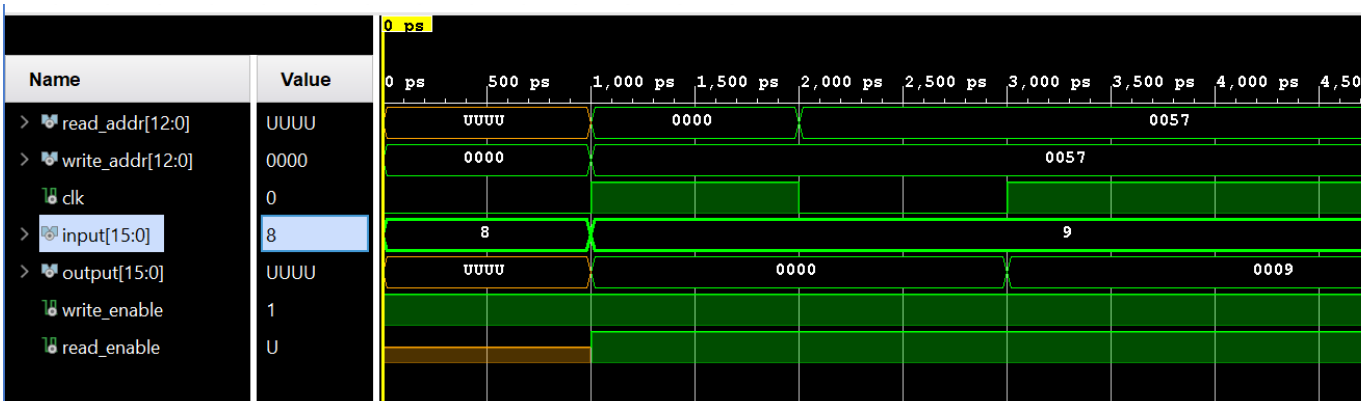
Component 2: Read Write Memory or Random Access Memory (RWM or RAM)

The Random Access Memory (RAM) creates a local memory in the form of an array of size 1000 consisting of 16-bit intermediate values. We do the following operations on the rising edge of the clock.

If *'read_enable'* has the value '1', then we output the 16-bit value stored at the address *'read_addr'* in the local memory array.

If the *'write_enable'* has the value '1', then we update the value at address *'write_addr'* with *'input'* in the local memory array.

Simulated Waveforms



Component 3: Read Only Memory (ROM)

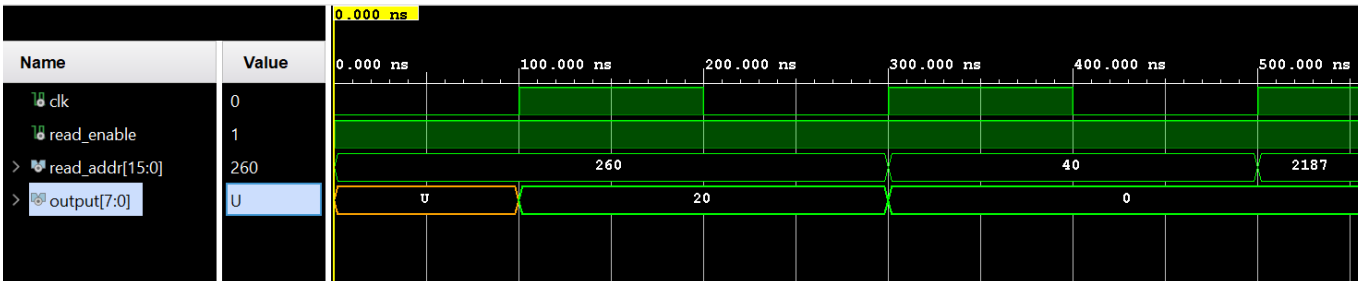
The Read Only Memory (ROM) is used for primarily 2 purposes:

1. To store the **Input Image** of size 28x28 with each pixel of 8 bits or 1B. Thus, it stores 784 8-bit wide words corresponding to the input image, and these words start getting stored from the address 0 (0000_{16}).
2. To store the **Weights and Biases**. The network consists of a total of 50816 weights and 74 biases of 8 bits each. Thus, in total, the ROM stores 50890 8-bit wide words, which start getting stored from the address 1024 (0400_{16}).

In our implementation of the ROM, we load the image, the weights, and the biases via their respective MIF files into Vivado. We load the image (from the image MIF file), weights (from the weights-bias MIF file), and biases (from the weights-bias MIF file) by reading them from the MIF file in flattened form into a vector *'rom_block.'* The image is read from the addresses 0-1023. The weights are read from the addresses 1024-51840. The weights are read from the addresses 51840-51913.

Once we have loaded the image, weights, and biases into the *'rom_block,'* if *'read_enable'* has the value '1', then we output the 8-bit word value stored at the address integer value of the *'read_addr'* in the *'rom_block'* memory array.

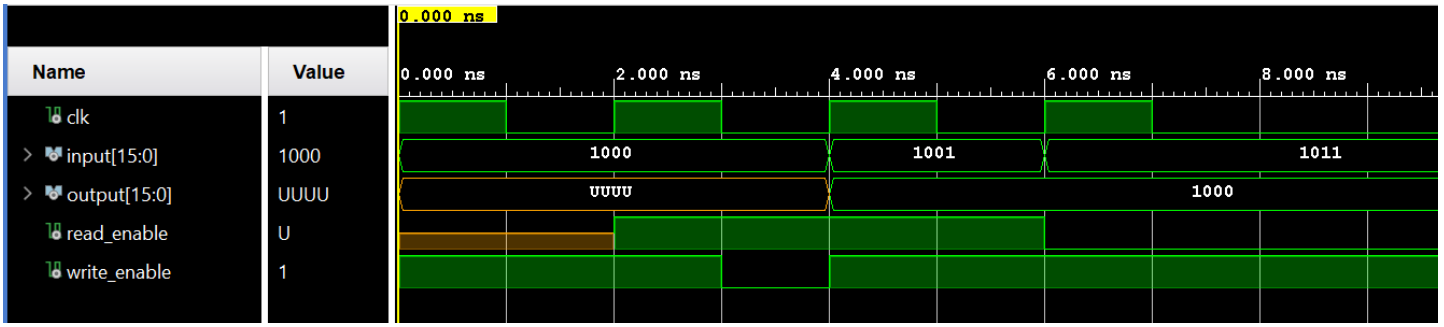
Simulated Waveforms



Component 4: Registers

The register is used to store intermediate data across clock cycles in a multi-cycle design like this. On the rising edge of the clock, if *'read_enable'* has the value '1', we read the output by storing the 16-bit value stored in the register. If the *'write_enable'* has the value '1', then we write the value of the register with *'input.'*

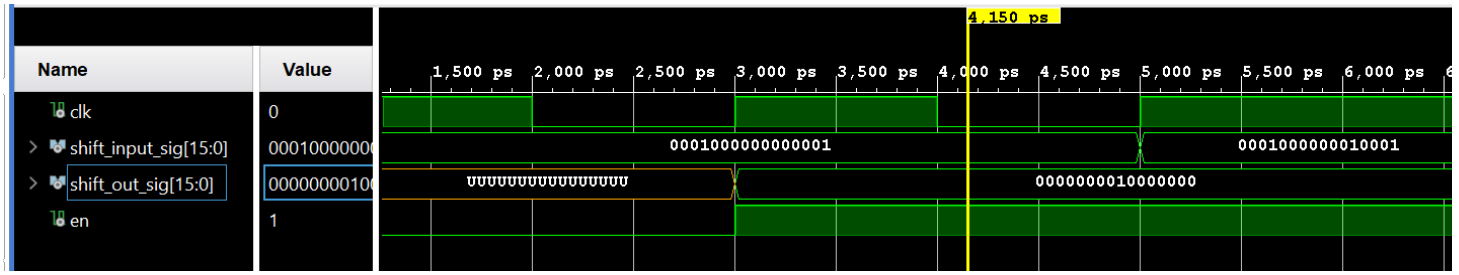
Simulated Waveforms



Component 5: Shifter

The purpose of the Shifter is to perform division by a constant factor of 32 or 2^5. While the parameters were scaled up by a factor of 32 for enhanced precision, after performing multiplication and accumulation, we will have to scale these down as well. For this, we will divide the value by a factor of 32. The division operation is synonymous with the right shift operation. To divide an input by 2, we can perform a right shift by 1. In order to divide by 32 or 2^5, we will perform a right shift by 5. In the output, *'shift_out_sig'*, we take '00000' in the beginning and concatenate the leftmost 11 bits of the input, *'shift_input_sig'* to it and consequently, drop the rightmost 5 bits of the input.

Simulated Waveform



Component 6: Comparator

The purpose of the Comparator is to implement ReLU or the Rectified Linear Unit activation function. This activation function converts all negative values to 0, and all positive values remain as is. Thus, the implementation of the comparator checks is the input value '*comp_input_sig*' being given to the comparator is negative or positive. This is done by checking the leftmost bit of the binary input given to the comparator. If this bit has a value of 1, this signifies that the input is negative, else the input is positive.

In case the input is negative, the output of the comparator, '*comp_out_sig*' gives the value x"0000", else it has the same value as the input.

Simulated Waveforms

Test Case 1:
comp_input_sig: x1001 (decimal: 4097)
comp_out_sig: x1001 (decimal: 4097)

Test Case 2:
comp_input_sig: 1000000000000000 (decimal: -32768)
comp_out_sig: x0000 (decimal:0)

Test Case 3:
comp_input_sig: x1011 (decimal: 4113)
comp_out_sig: x1011 (decimal: 4113)