

COL215P - Hardware Assignment 2

Hardware Implementation of MLP in VHDL

Submission Deadline: 30th September 2022

1 Introduction

This assignment is an exercise to implement a 3-layer Multi-layer Perceptron (MLP) in VHDL and test your implementation by running inference on 10-class fashion MNIST dataset on Basys3 board. This assignment involves the design and integration of memories, registers and multiplier-accumulator components (MAC) in your design.

2 Problem Description

Implement a 3-layer neural network inference in VHDL and test it on Basys3 board. Figure 1 shows an overview of the structure of the neural network with 3 layers: Input (28x28 or 1x784 when flattened), Hidden Layer 1 (1x128) and Output Layer (1x10). The network has been pre-trained on Fashion MNIST dataset. You will be provided with the pre-trained network parameters (Weights: 784x128 and 128x10, Biases: 128 and 10) in the form of a MIF file that can be loaded in Vivado. The description of how to use MIF file is given in Section 2.6.

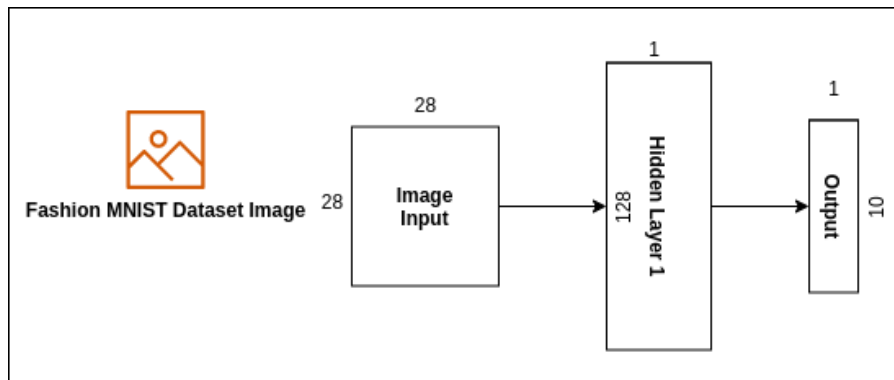


Figure 1: 3-layer MLP pre-trained on Fashion-MNIST dataset with 8-bit integers.

2.1 Operations performed in an MLP

The network shown in Figure 1 consists of all fully-connected layers in which all neurons between two layers are connected to each other. The operations in fully-connected layers are done using simple matrix-matrix multiplication or vector-matrix multiplication (which we will be doing in this assignment) as shown in Figure 2. Since these vectors and matrices could be very large, loading them from memory is extremely slow. It might sometimes be worth to load only a part of these matrices into a local memory instead of loading the complete matrix before starting the computation, as shown in Figure 2.

The overall design would consist of the following:

1. a Multiplier-Accumulator block (MAC): to perform vector-matrix multiplication
2. a Read-Write Memory (RWM, more popularly known as RAM or Random-Access Memory) - You will be using this memory to store outputs of all the layers
3. a Read-Only Memory (ROM) - You will be using this memory to store the parameters and the input image
4. Registers - You will be using these to store temporary results across clock cycles
5. an overall circuit that stitches all modules to perform neural network inference on the given image.

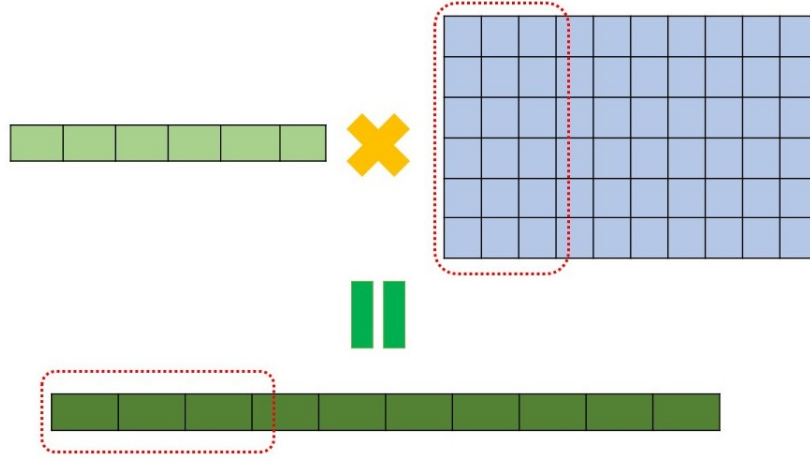


Figure 2: Vector-Matrix multiplication in a Fully-connected layer of MLP

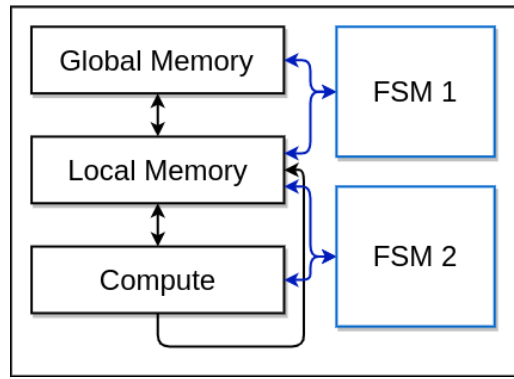


Figure 3: Overview of the design implementation

Figure 3 shows a brief overview of the hardware implementation with control path and datapath. The datapath includes reading the inputs and parameters from a global memory (designed as RAM and ROM) and passing it over to the compute unit for computation. When the computation needs large number of inputs at the same time, sometimes another level of memory hierarchy is introduced in the design to store a part of the input. This concept was introduced to you in Figure 2. While the compute unit is working on the inputs already read in the local memory, the system can keep reading more inputs from the global memory in parallel. This helps to improve hardware utilization and improve performance with more parallelism.

The figure also shows a control path with 2 FSM's. One FSM controls the reading of inputs from global to local memory and another FSM controls the communication between local memory and compute. Figure 4 shows an overview of the control and data path taking an example of one set of inputs from one layer of the network.

2.2 Multiply-Accumulate block or MAC

The first component to be designed in this assignment is a MAC unit that has a 8x8 multiplier, an 8-bit register and an adder to keep accumulating the products. Figure 5 shows the high-level overview of a multiplier-accumulate unit with the required input and output ports.

cntrl signal can be used to tell the unit whether it is the first product of the compute or not. If it is the first product, assign sum to the first product. After that the product needs to be accumulated with the previous sum. The other signals are self-explanatory.

2.3 Memories

The design requires memories to store the inputs, temporary activations and the outputs of the networks. We will be storing the image, weights and biases in a read-only memory or ROM and we will store the outputs and temporary data into a random-access memory or RAM.

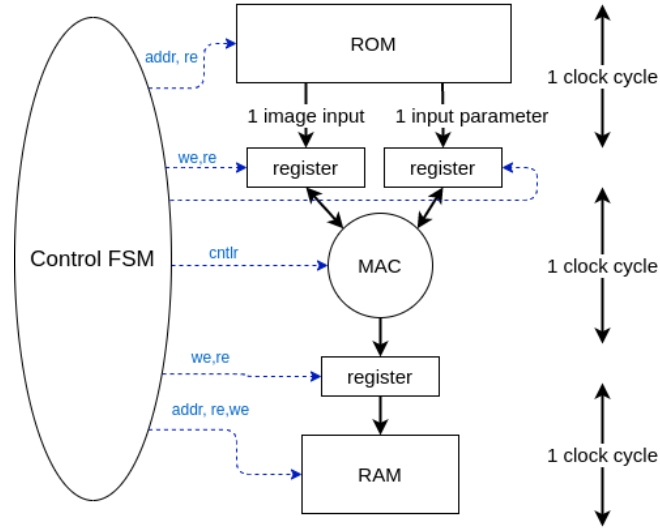


Figure 4: Overview of the control and data path for one set of inputs from one layer to generate one partial output

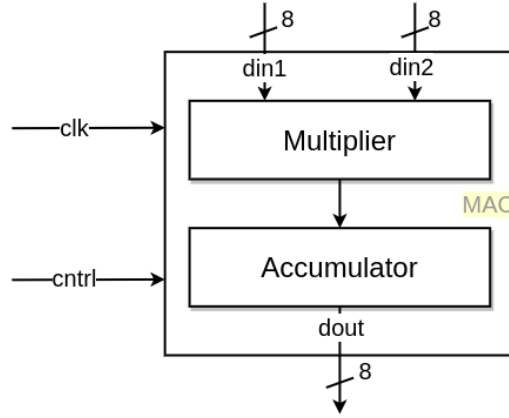


Figure 5: Multiplier-Accumulate Unit

2.3.1 Read-Only Memory or ROM

This memory will be used to store the following:

- Input Image: 28x28 image with each pixel of 8-bits or 1B. For storing the input we need 784 words, 8-bit wide. These will be stored at address 0 (0000_{16}) onwards.
- Weights and Biases: The network has total of 101632 weights and 138 biases of 8-bits each. So for storing these, we need 101770 words, 8-bit wide. These will be stored at address starting from 1024 (0400_{16}).

2.3.2 Random-Access Memory or Read-Write Memory or RAM

This memory will be used to store the temporary activations generated by each layer and the final output of the network. It will also be used to create the local memory talked about earlier. **We leave it upto you to decide the size of the local memory needed.**

2.3.3 Registers

Registers are sequential elements that store temporary data across clock cycles in a multi-cycle design. The signal description is given in Figure 6. You can use these registers as per your design requirements.

2.4 Control Path

MLP execution is highly sequential and layer-wise. The control path is basically an FSM in your design which is supposed to do all the sequencing of the data required starting from generating addresses of the data in the

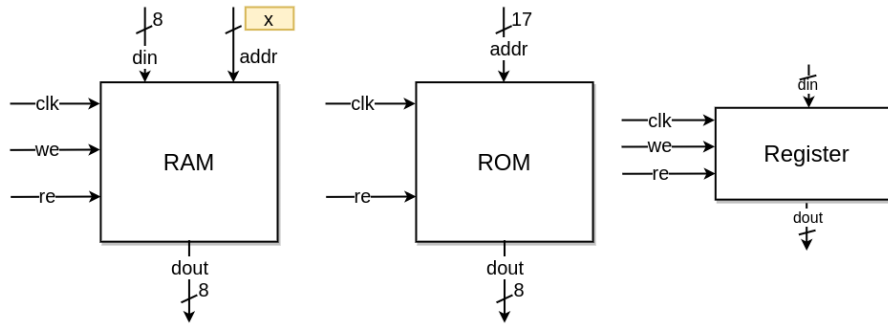


Figure 6: RAM, ROM and Registers block diagram

order in which you want to read the data, generating the cycle-wise signals to be sent to the datapath to control the modules in the design.

2.5 Output

Fashion MNIST dataset has 10-classes. You need to run inference of the given image on your design and display the output class number on one seven-segment display.

2.6 Using a MIF file in Vivado

Image, weight and bias parameter will be loaded into the vivado via MIF file. To read the MIF file, you need to use `std.textio.all` in the code. Here is the sample function to read the MIF file and store the contents in the array. You need to add this function in your ROM VHDL module. This function needs to be called for initializing ROM.

The image is read from the MIF file and stored in flattened form into a vector `rom_block`. You need to use this vector to initialize ROM module. The data in the MIF file is stored in ADDR:DATA form, where ADDR size is 10-bit and DATA size is 8-bit. Image pixel (x,y) will generate the address as xy in binary. For eg, if pixel is (2,2) then ADDR will be 0001000010.

```
entity ROM_MEM is
    generic (
        ADDR_WIDTH      : integer := 10;
        DATA_WIDTH      : integer := 8;
        IMAGE_SIZE       : integer := 784;
        IMAGE_FILE_NAME  : string := "imgdata.mif"
    );
end ROM_MEM;

architecture Behavioral of ROM_MEM is
    TYPE mem_type IS ARRAY(0 TO IMAGE_SIZE) OF std_logic_vector((DATA_WIDTH-1) DOWNT0 0);

    impure function init_mem(mif_file_name : in string) return mem_type is
        file mif_file : text open read_mode is mif_file_name;
        variable mif_line : line;
        variable temp_bv : bit_vector(DATA_WIDTH-1 downto 0);
        variable temp_mem : mem_type;
    begin
        for i in mem_type'range loop
            readline(mif_file, mif_line);
            read(mif_line, temp_bv);
            temp_mem(i) := to_stdlogicvector(temp_bv);
        end loop;
        return temp_mem;
    end function;
    // Signal declarations
    ...
    signal rom_block: mem_type := init_mem(IMAGE_FILE_NAME);
```

```
...  
  
begin  
    //Your ROM code  
end Behavioral;
```

3 Submission and Demo Instructions

Since this assignment will span over 4 weeks, you will be evaluated every two weeks in the lab depending on your progress. We have broken down the design into blocks and you need to spend time accordingly to complete this assignment.

1. **Week 1 and Week 2: Design and test all the sub-components of the datapath in the assignment. For this week, we are giving you a sample MIF file to get your memory block working. The actual image and weights would be uploaded later. You can test your ROM using this file. You can test your RAM using a testbench.**
2. **Week 3 and Week 4: Design the control path and integrate it with the data path. You have to test the design and show the class-output on seven-segment display.**
3. Demo should be given in the assigned lab slot.
4. You are required to submit zip file containing the following on Gradescope:
 - VHDL files for all the designed modules.
 - Simulated waveforms with test-cases of each sub-component and complete design
 - A short report (1-2 pages) explaining your approach. Include block diagram depicting the modules.
5. Post your doubts in Assignment 2 thread on Piazza.
6. It is advisable that you should be ready with your design before the lab session, and during the session, perform validation of the design by downloading it into the FPGA board

4 Resources references

- IEEE document: <https://ieeexplore.ieee.org/document/8938196>
- Basys 3 board reference manual: https://digilent.com/reference/_media/basys3:basys3_rm.pdf
- Online VHDL simulator: <https://www.edaplayground.com/x/A4>