COL215P- Assignment 2 Part 1

Name : Ujjwal Mehta | Entry No. 2020CS10401

Name : Somaditya Singh | Entry No. 2020CS10389

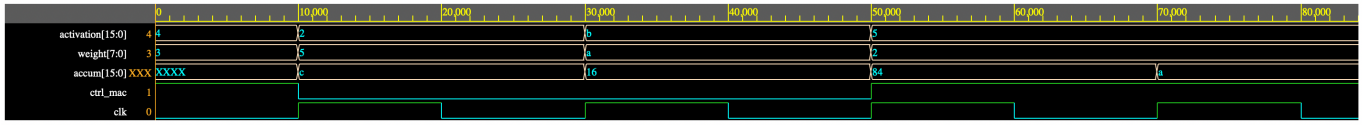
In this assignment part we have created all the modules needed to implement a neural network architecture. The following sections contains the information about all the modules of the data path along with their simulation images and explanation.

Multiply Accumulate Block(MAC)

The code implementation for this module is inside the mac.vhd file and the function of this component is to take 1 16 bit signed activation and 1 8 bit signed weight and multiply and truncate their product using LSB and then add the result in the accumulator or assign it to the accumulator depending on the control signals on every rising edge of clock cycle. The entity ports are shown below :

```
Port (

 weight : in signed(7 downto 0);

 activation : in signed(15 downto 0);

 clk : in std_logic;

 ctrl : in std_logic;

 accum : inout signed(15 downto 0));
```

The testing simulation screenshot is shown below :

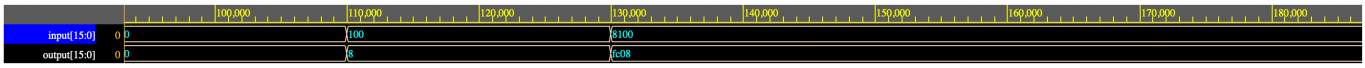


Shifter

The code implementation for this module is inside the shifter.vhd file and this component is used for doing a signed right shift by 5 bits on the 16 input bit given to it. This will later be used to shift the layer outputs. The entity ports are shown below :

```
Port (
    input : in signed(15 downto 0);
    output : out signed(15 downto 0));
```

The testing simulation screenshot is shown below :

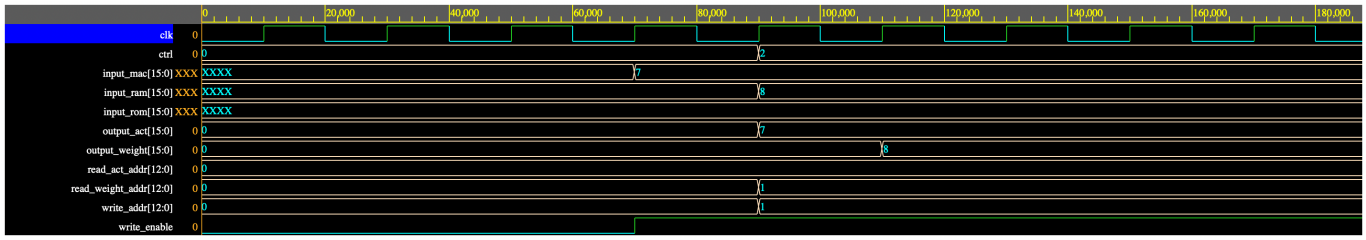


RAM (Random Access Memory)

The code implementation for this module is inside the ram.vhd file and this component is used for storing the layer outputs (global memory) as well as storing the 3 columns of weight matrix (treating like a local memory where another instance of RAM will be used to handle storing the ongoing computations) as well as accessing and modifying the activation data from local and global memory. In order to implement this we give the write_enable and control signal which on every rising edge will check if we need to write in the ram or not and if the control signal is 0 then this means we are writing from mac input, if the control signal is 1 then we writing from global memory and if the control signal is 2 then we write from ram which contains the activation value. The memory of single instance of ram is 3136 elements of `signed(15 downto 0)` where 3136 is 784*4 (since we need to load 3 weight columns as well the input image data column to initially). Now outside this clock we are always reading the weights and activation from the corresponding addresses. The entity ports are shown below :

```
Port (
clk :in std_logic;
ctrl : in std_logic;
write_enable : in std_logic;
write_addr : in unsigned(12 downto 0);
read_act_addr : in unsigned(12 downto 0);
read_weight_addr : in unsigned(12 downto 0);
input_mac : in signed(15 downto 0);
input_ram : in signed(15 downto 0);
output_act : out signed(15 downto 0);
output_weight : out signed(15 downto 0));
```

The testing simulation screenshot is shown below :

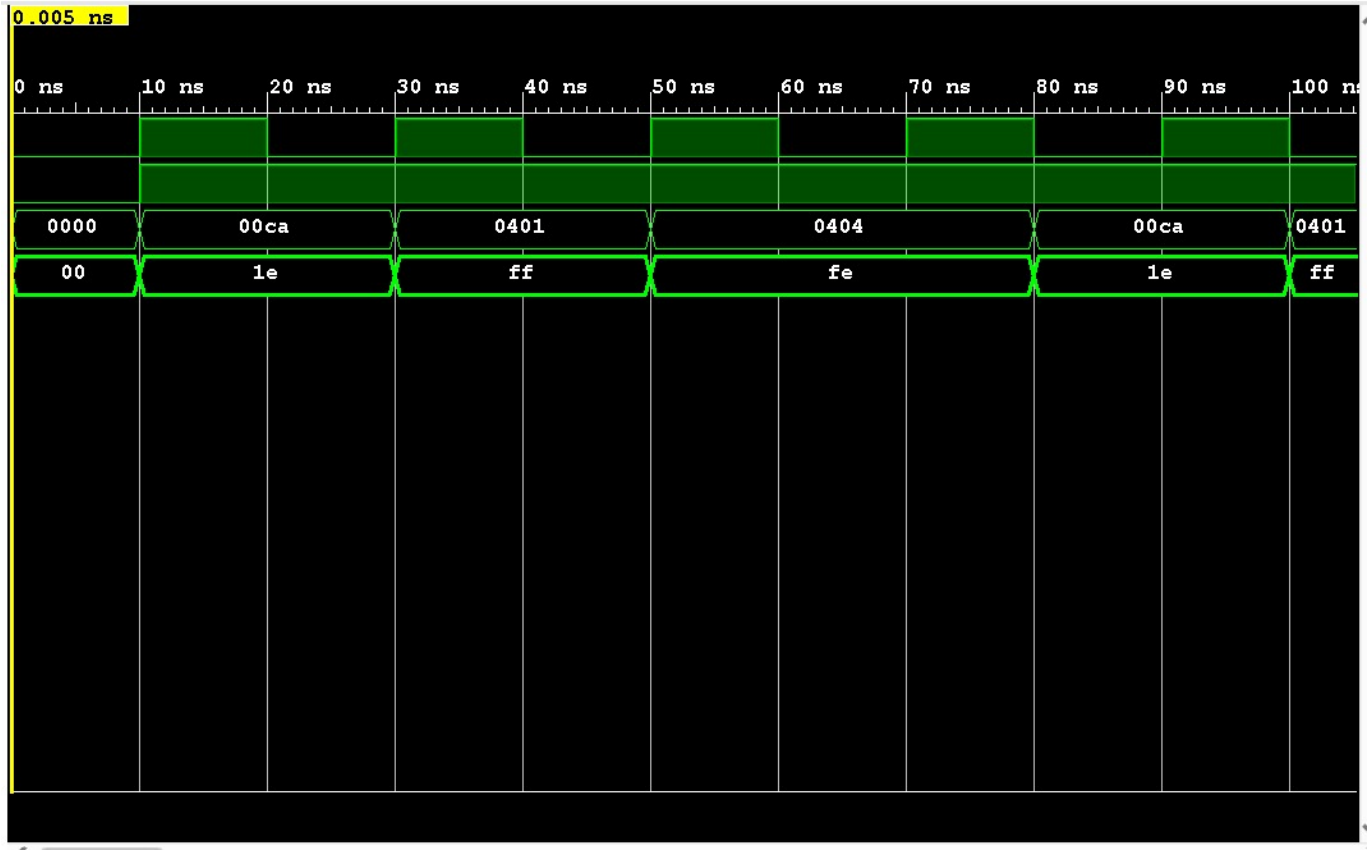


ROM (Read Only Memory)

The code implementation for this module is inside rom.vhd file and this component is used for reading the imagedata.mif and weights_bias.mif and finally store all the read result inside an array of size 51914 containing `std_logic_vector(7 downto 0)` where the data corresponds in the 784 elements of array corresponds to the image data and the data from 1024 index to 51913 index corresponds to the weight values. In order to use this component efficiently we give an address input and read the data at the corresponding address at rom. The file reading is done using the code given in the assignment itself. The entity port are shown below :

```
Port (
   clk : in std_logic;
   read_enable : in std_logic;
   addr : in unsigned(15 downto 0);
   read_out : out signed(7 downto 0));
```

The testing simulation screenshot is shown below :



Comparator

The code implementation for this module is inside the comparator.vhd file and this component is used for comparing 2 `signed(15 downto 0)` and outputing the one which is maximum. The entity port are shown below :

```
Port (
   input1 : in signed(15 downto 0);
   input2 : in signed(15 downto 0);
   output : out signed(15 downto 0));
```

The testing simulation screenshot is shown below :