Report HW Assignment 2

Atul Jeph 2020CS10329 Aryan Gaurav 2020CS10327

October 2022

1 Progress Report

We would first like to mention that we weren't able to complete the whole assignment. In terms of progress, we were able to complete nearly 80 percent of the assignment ,i.e., our code runs smoothly till the second layer (the hidden layer) but, due to some issues, we weren't able to get the third layer (the output layer) correctly and hence, we didn't display the output class number on the seven-segment display.

2 Approach

We first created all the required modules, i.e., MAC, RAM, ROM, Register, Comparator, Shifter and a testbench stitching all the modules together.

2.1 MAC

The MAC module is as stated in the assignment; it takes an 8-bit weight input, a 16-bit image input/activation, consists of a 16*8 multiplier module and an accumulator. The accumulator is for summing all the pair wise products of image vector and weight matrix's column elements. The accumulator sets the *sum signal* equal to the value of first pair wise product; this is done by passing the *control signal* value as 1 while passing the first pair from the *testbench* to the *RAM* to the *MAC* module; for the rest of the product accumulation, the *control signal* is kept 0. The final output from the *MAC* module is a 16-bit value.

2.2 Shifter and Comparator

The shifter and comparator module come just after the MAC module. The comparator module checks if the value returned from the MAC module is non-negative or not; if yes, it leaves it as it is, else it makes it 0 (implements the ReLU function). The shifter then divides the processed value by 32, i.e., shifts the binary value of the output to the right by 5 bits. It is after these two modules only that the value finally reaches the *RAM*.

2.3 ROM

The ROM module is for loading the image vector (1*784), the weight matrices (784*64 and 64*10) and biases (64 and 10) from the given MIF file. But since the number of inputs is too large, we load the input in parts into the RAM (the *local memory*) and then, pass it from there to the *compute* section.

2.4 RAM

The *RAM* module is divided into two parts: the *local* memory and the *intermediate* memory. We have allocated a total space of 2000 words to the *local* memory and 64 words to the *intermediate* memory, 16-bits each. In the *local* memory, the first 784 words are for loading the image vector and starting from 1000, the next 784 words are for loading weight column values (1 column is processed at a time). For the biases needed in the *compute* section, we call them directly from the *ROM* module. The *intermediate* memory is for storing the second layer values.

We planned on overwriting the space in the *local* memory with the values attained in the *intermediate* memory, for calculation of the third layer and then, overwriting the *intermediate* memory with the values of the third layer.

2.5 Testbench

Here, all the modules are stitched together as stated in the above description. For example, the values are loaded in *RAM* from the *ROM* module via the testbench; the image and weight values are sent from *RAM* to the *compute* section via the testbench; the output from the *MAC* module is stored in *RAM* via the testbench etc.

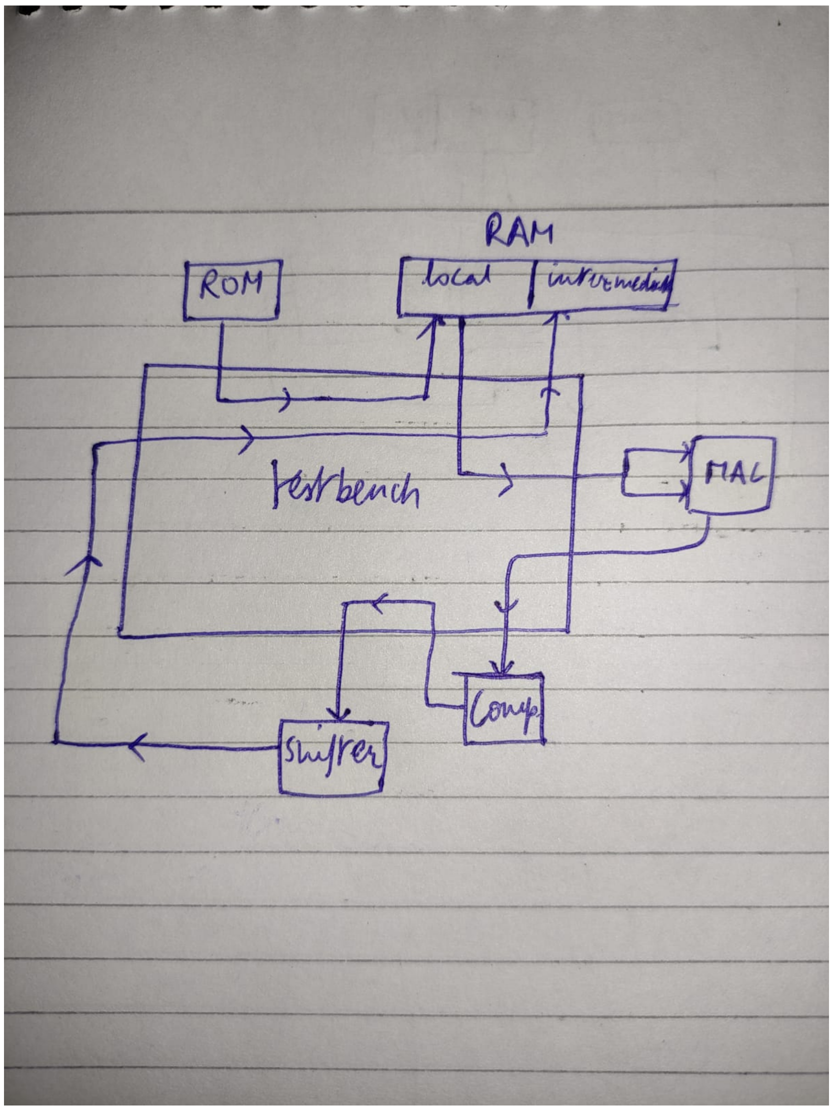

Figure 1: Block Diagram