COL215P Assignment 2 Stage 2

Aniruddha Deb
2020CS10869

Sachit Sachdeva
2020CS10840

September 2022

Design

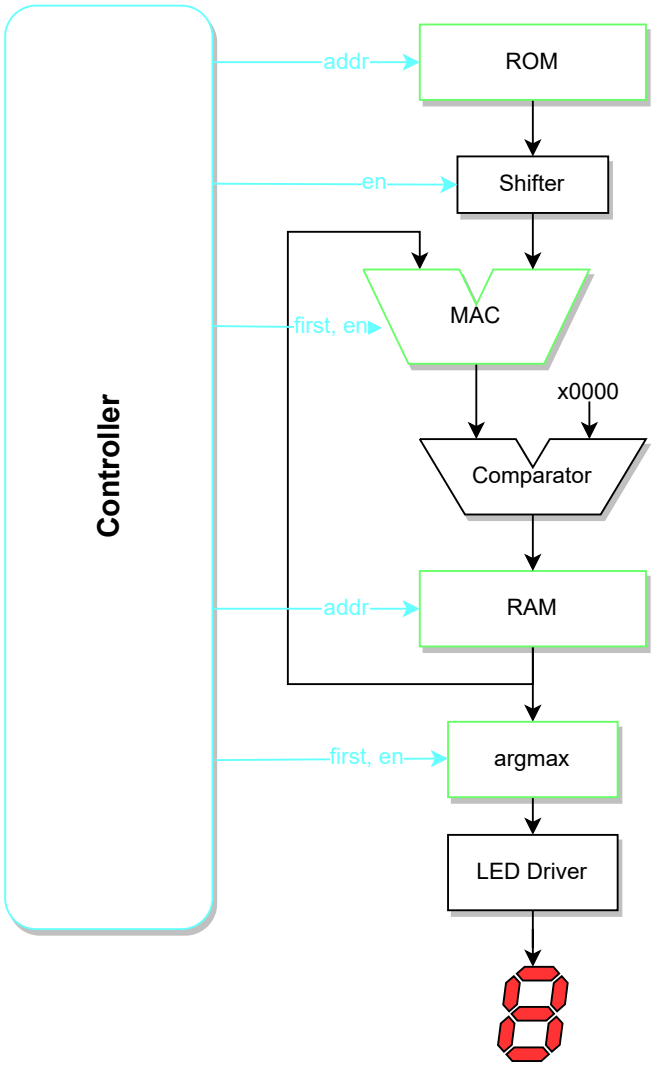


Figure 1: The overall design of the model

The following entities of first stage were used in our design - ROM, RAM, MAC, Shifter, Register, Comparator, Argmax, LED Driver.

Initially in about 784 clock cycles we read the 1 x 784 input image from the ROM into our RAM. (NOTE - Input image is not directly read from ROM while computation, but rather stored in RAM). The section in RAM storing the input image can be considered as a local memory from where we would be reading our first level input.

The RAM is divided into 3 spaces. One for storing the input image read from ROM(which can be considered as a local memory) and the rest 2 for storing the result of intermediate and final computation.

Once the input image is read into RAM, image input from local memory/RAM and weights from ROM are read parallelly and fed into the MAC, which multiplies and accumulates the result. The reading of next input for MAC and result accumulation are all happening in a pipelined manner. After multiplication of corresponding row and column, bias is added, the result is shifted by 5 bits, and later passed through RELU (comparator). These intermediates results are later stored in a particular space of RAM.

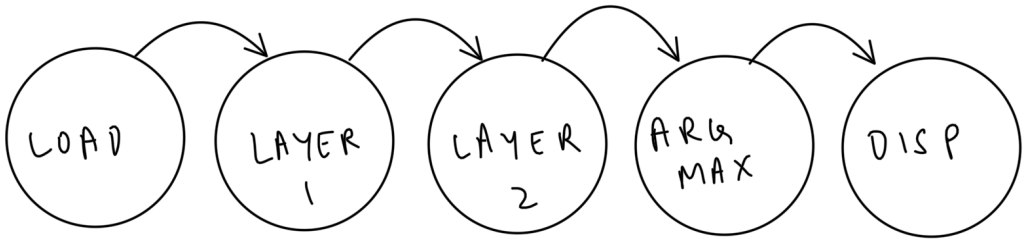
Once the first layer of computation is done, the second layer of computation proceeds in a similar manner reading input from the intermediate space of RAM and storing the output in the result space of RAM (while reading corresponding weights and biases from ROM).

After the final results (corresponding to a vector of size 10) are stored in the RAM, this data is fed to the Argmax module. The argmax reads each of these 10 values from the max and reports the index of the maximum among these values (this index corresponds to the output of classification). This index is fed to the LED-Driver which displays the corresponding digits

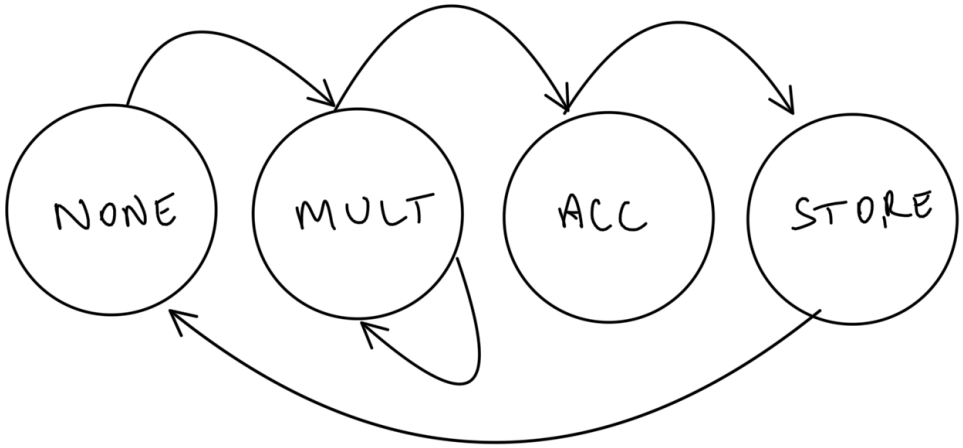
The efficiency of our design lies in the fact that both inputs to the MAC are read parallelly in one clock cycle, and reading of MAC inputs along with accumulation of partial results in the accumulator happen in a pipelined manner. Because the hardware threshold in the current scenario is the number of clock cycles it takes to read from the ROM, in any design which reads weights and biases from the ROM in a pipelined manner (either to first to a separate local memory or directly), the pipeline threshold would still be the number of clock cycles for ROM reads, thereby making both choices equally efficient.

Controller State Machine

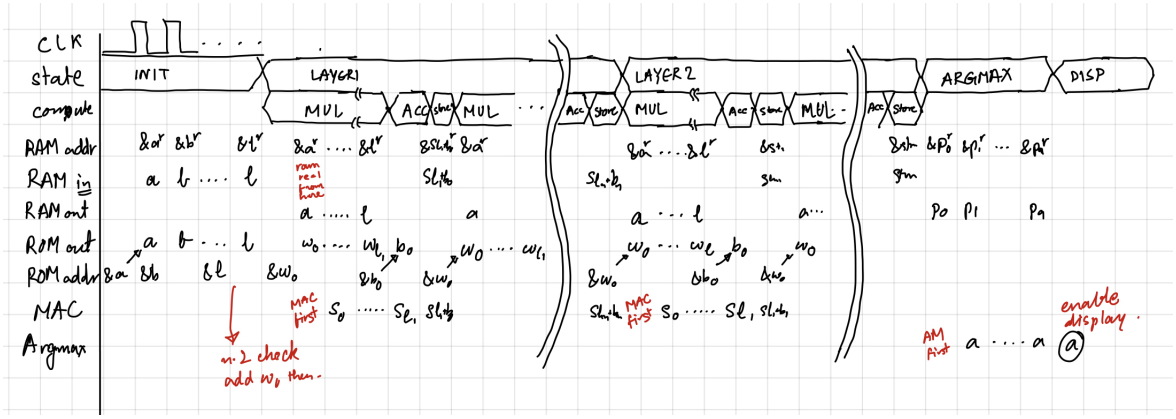
Our design consists of 2 hierarchical FSM's. The top level FSM provides overall sequentiality to the process of Input read from RAM, Layer-1,2 computation, Argmax computation and Display. The states are LOAD, LAYER1, LAYER2, ARGMAX, DISP.



Corresponding to state LAYER1 and 2 we had a lower level FSM, responsible for sequentially performing row-column multiplication, bias addition and storage of the final/intermediate result in RAM. The states were NONE, MULT, ACC, STORE. State will be NONE when the state of higher level FSM is neither LAYER1 nor LAYER2. In the MULT state multiplication of weights and corresponding input/intermediate result takes place. In the ACC state the bias is read and added. In the STORE state the intermediate/final result is stored back to RAM



Timing Diagram



As a result of being fully pipelined, the design needs to ensure that the indices used as input to RAM and ROM are available one clock cycle before the data corresponding to those indices is actually used. To implement this accurately, a timing diagram was used to simplify the state transitions (shown above)

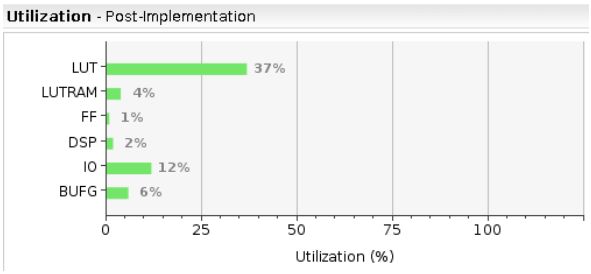
The timing diagram shows the various states the system is present in, as well as the clock cycles which the system spends in those states. It also shows at what instance of time, which datapoints are required to the various modules for them to provide the correct outputs.

Broadly, the ROM address is always one less than the RAM address, as ROM reads are synchronous whereas RAM reads are asynchronous. The value stored in MAC is two cycles ahead of the ROM address and one cycle ahead of the RAM address, and this is taken care of when changing states to ACC and MUL, as shown in the timing diagram.

State transitions between larger states also need to be handled by the FSM, namely transitioning to the ROM address needed by the next state, one clock cycle before the next state actually starts. This is implemented correctly in our code.

Implementation results

The Synthesis and Implementation were done on Vivado 2016.4, on the Narmada machine. Synthesis took approximately 8-10 minutes and implementation took 4-5 minutes.



Utilization graph

Utilization - Post-Implementation

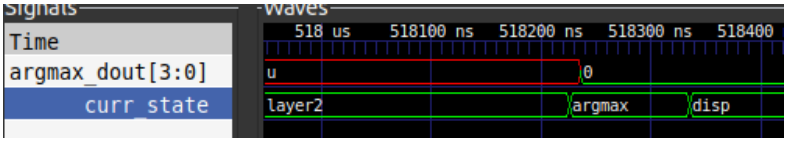
Resource	Utilization	Available	Utilization %
LUT	7601	20800	36.54
LUTRAM	352	9600	3.67
FF	248	41600	0.60
DSP	2	90	2.22
IO	13	106	12.26
BUFG	2	32	6.25

Utilization Table

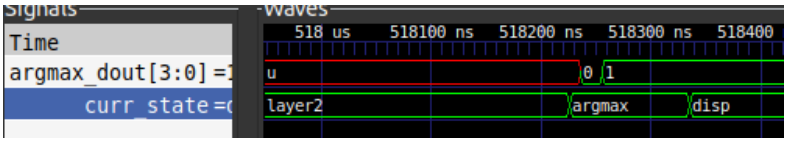
Simulation Waveforms/Output

The following are the outputs of simulation on their respective mif files using the controller testbench:

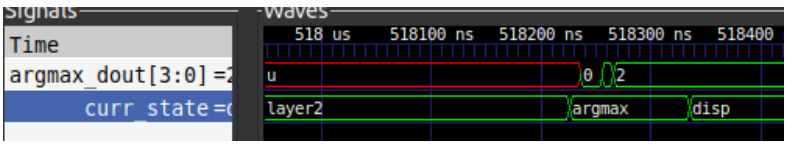
imgdata_digit0.mif



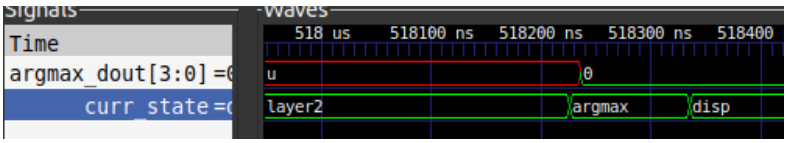
imgdata_digit1.mif



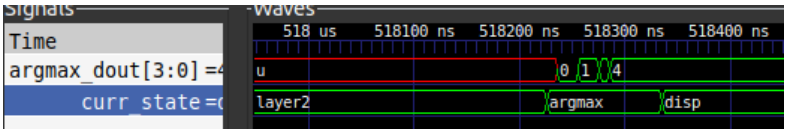
imgdata_digit2.mif



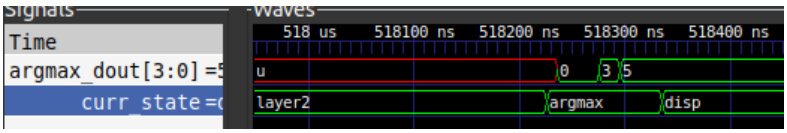
imgdata_digit3.mif



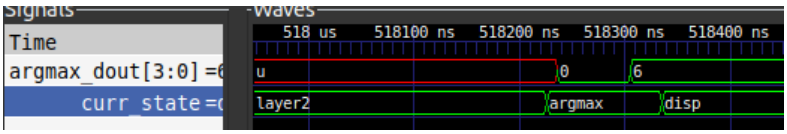
imgdata_digit4.mif



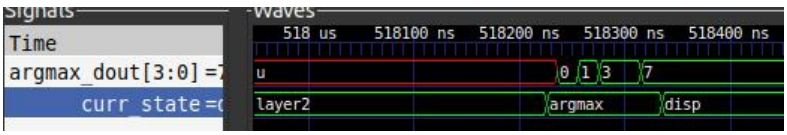
imgdata_digit5.mif



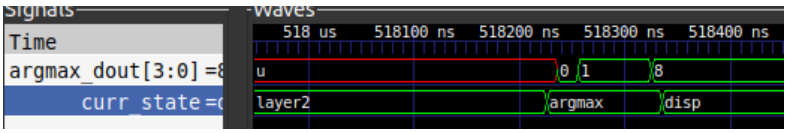
imgdata_digit6.mif



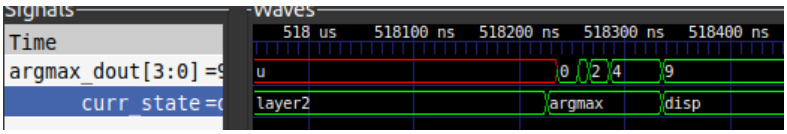
7.mif*



8.mif*



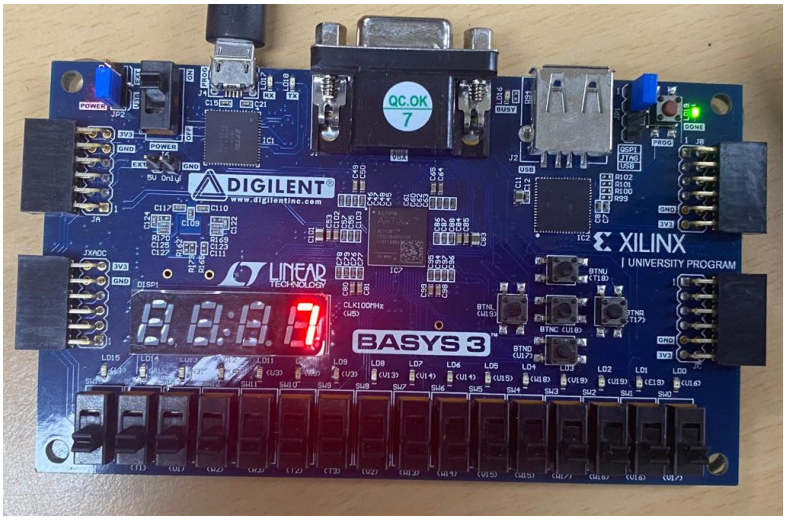
9.mif*



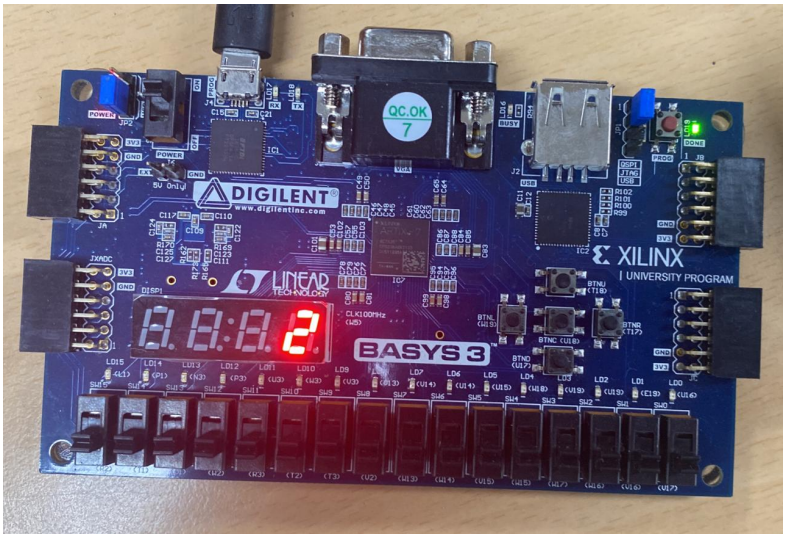
* - mif files made by us from the MNIST dataset

Testing on basys board

Using 7.mif:



Using 2.mif:



Using 4.mif: