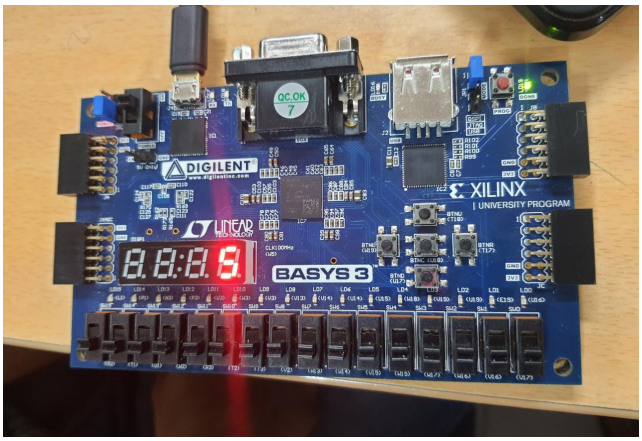
COL215P Hardware Assignment 2 Part 2

Multi-layer Perceptron (MLP)

Gurarmaan S. Panjeta & Aryan Dua
2020CS50426 & 2020CS50475

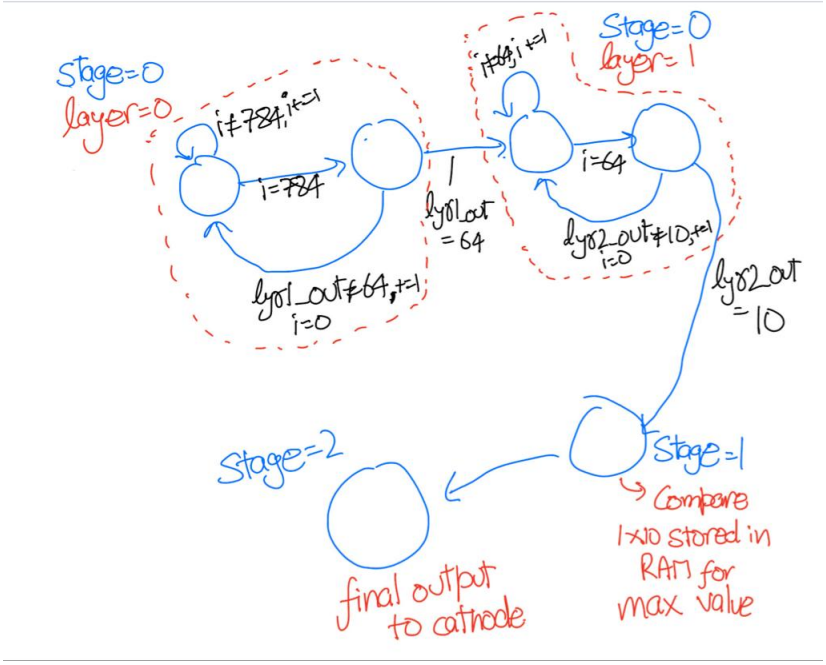


October 9, 2022

1 Sub-Components

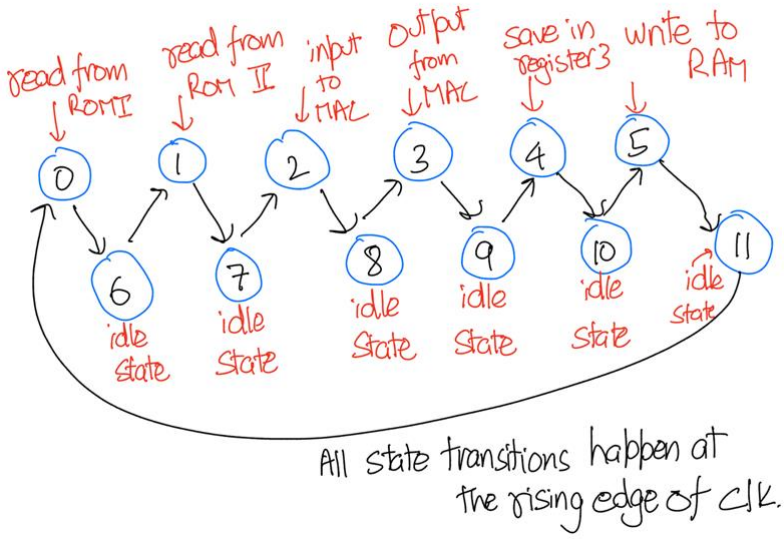
1.1 ”test.vhd” The top-level entity

This is the file that implements and brings together the MLP. It is a series of if and elsif conditions, which use control signals ”stage” - 0 while Multiplication, 1 while finding max out of 1x10, and 2 while output; ”layer”- 0 while computing layer 1 and 1 while computing layer 2; and some additional flags that convey transfer of operation to next layer, stage or part. These parameters demarcate regions for us to carry out their respective operations on.



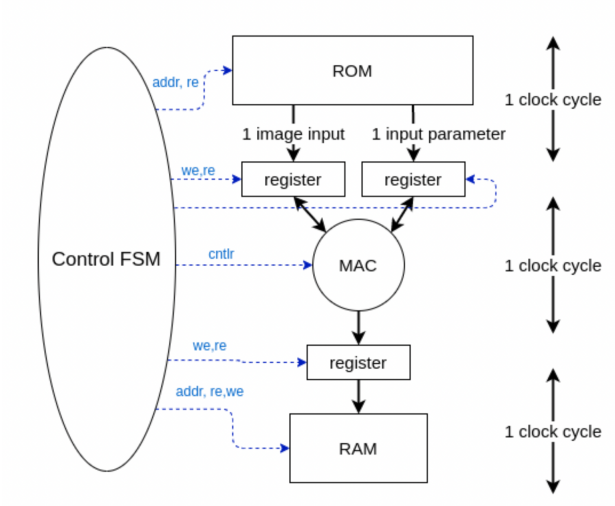
1.2 ”fsm.vhd” The control path

The circuit is based on an FSM that has 12 states, for memory read and write to registers, for MAC computation and for writing to RAM if necessary. After calculating the final 1x10 vector, we switch to stage = 1, where we compare and find the max value. Then, we output the corresponding character to the cathode output for display. This file gives the control signals to the datapath that tell it what actions to perform on the data. The states of the FSM change as follows :-



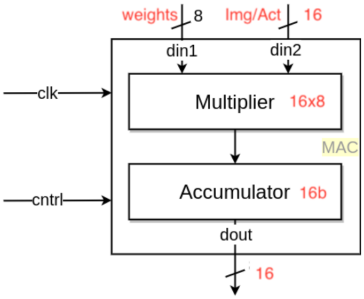
1.3 "data_path.vhd" The Data Path

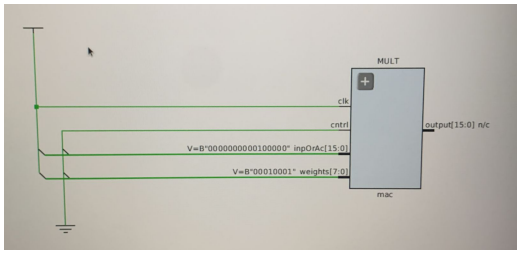
This module handles data manipulation and storage/retrieval. It gets it's state number and the control signals from the FSM and the test file, and accordingly performs operations.



1.4 "mac.vhd" Multiply-Accumulate Block (and Shifter)

The entity responsible for multiplication and accumulation of results, as the name suggests. The component receives a "cntrl" signal that tells the block whether the data received from the registers is the first product or not. If it is the first product, it stores the computed value in the accumulator, if not, it adds the value to the prexisting accumulated value and stores it to the accumulator again. The shifting (division by 32) is also performed by this component itself. The IO of the module is as follows:-

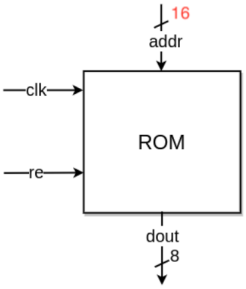


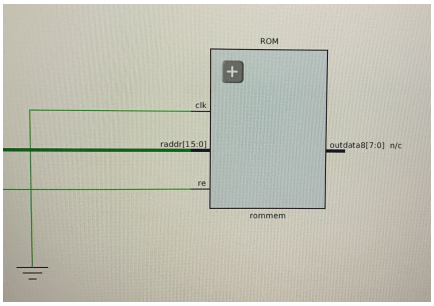


1.5 ”rom.vhd” ROM

An essential part of the assignment was to read the ”.mif” files and store it’s data in a module where it can be read out from. This module stores the data in an array of 51914 words. Of these, the first 784 (0-783) house the data of the image to be classified, (784-1023) have been left empty, and (1024-51913) house the weights and biases.

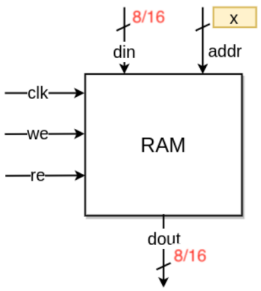
These words can be accessed if the ”read_enable”(re) signal is on, and a 16 bit address of the required word is fed into the component. The IO of the module is as follows:-

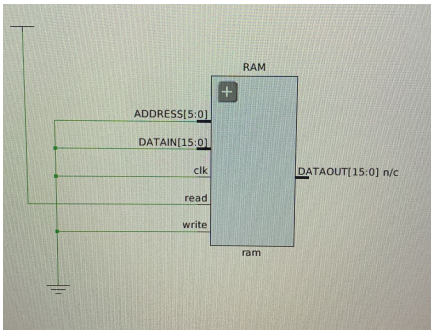




1.6 ”ram.vhd” RAM

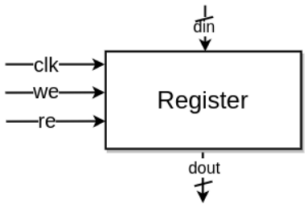
The Dynamic memory that stores intermediate results from each layer and provides the data for further computations. The final result is also stored here. The implementation houses a 785 word array, that has control signals like read enable and write enable that churn out the relevant outputs. The address is provided as a 10 bit input, and the relevant data is stored into/outputted from that address. The IO of the module is as follows:-

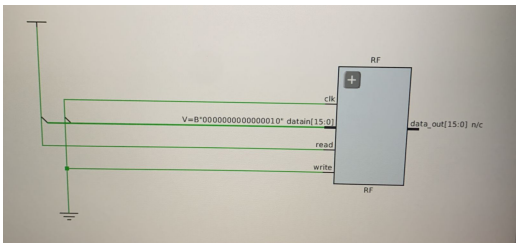




1.7 "reg.vhd" Register

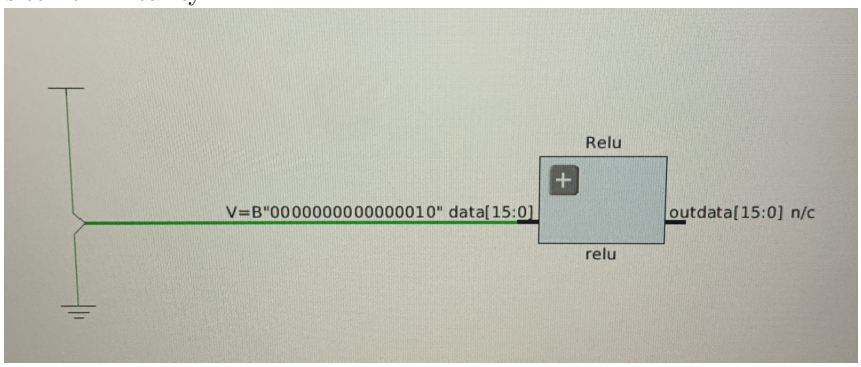
Registers are sequential elements that store temporary data across clock cycles in a multi-cycle design. With Control signals like read enable and write enable, they are efficient storage modules for intermediate results within a particular layer. These will be utilised to store a 1 word image input, input parameters (from ROM) and the outputs from MAC until the next cycle operates. The IO of the module is as follows:-





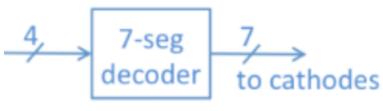
1.8 "relu.vhd" Comparator

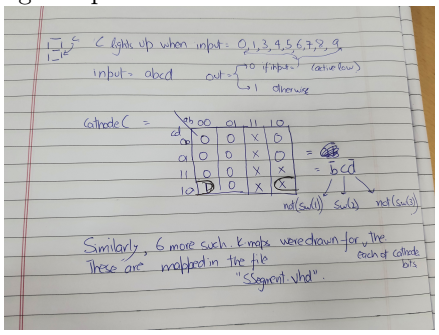
The ReLU functionality of the MLP dataflow has been materialised in this component. If the input is >0, we return it as is, but if it is <0, we return 0 instead. This introduced the requisite non-linearity.

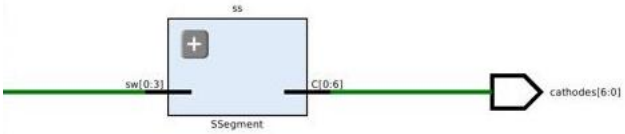


1.9 "SSegment.vhd" (Display)

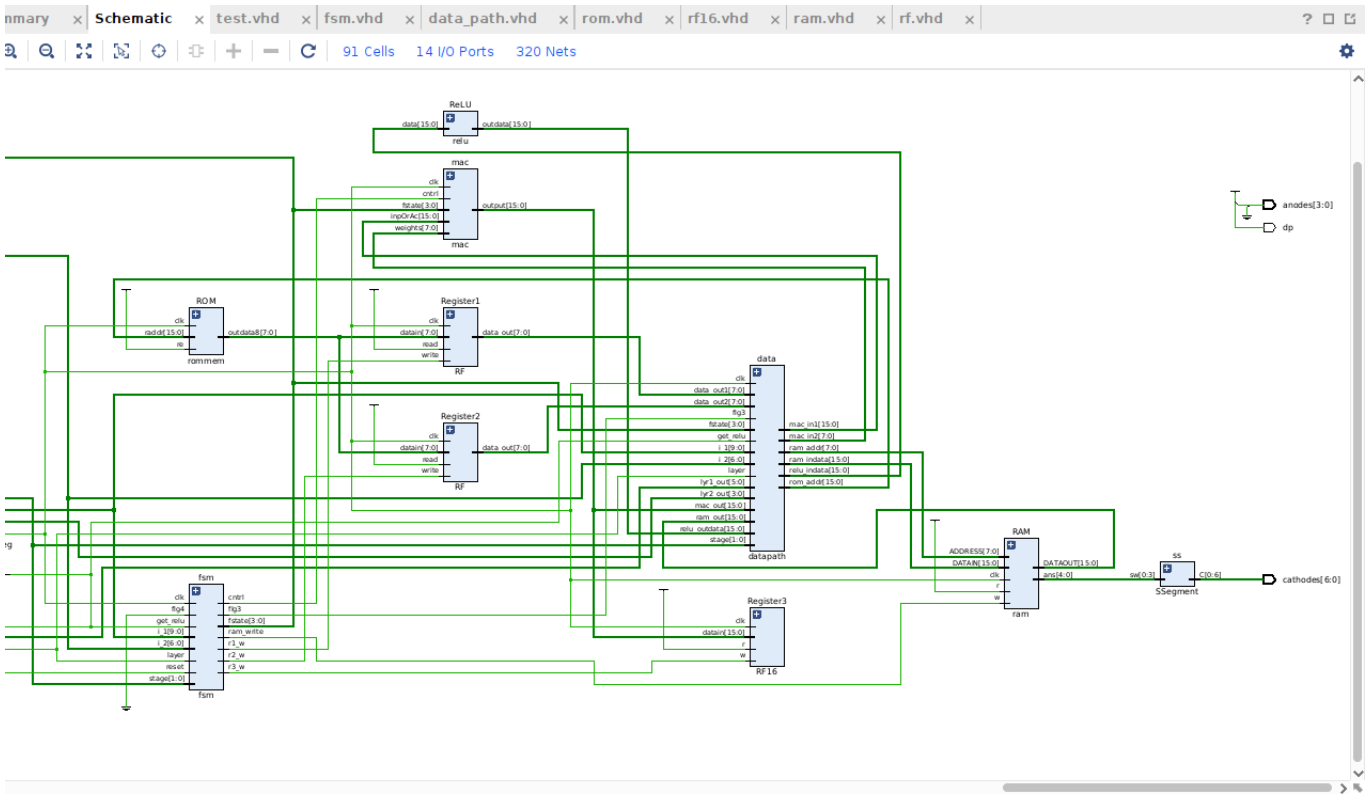
The "des.vhd" identifies the input as a number from 0 to 9. This number must now be sent to the 7 LEDs which should Light up to display the number in a human readable format. This is done by implementing Karnaugh maps for each of the 7 LED's.







2 Block Diagram

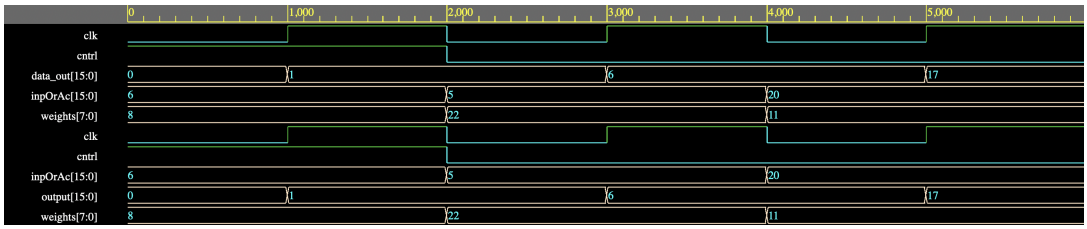


3 Testing of Components

3.1 "mac.vhd" Multiply-Accumulate Block (and Shifter)

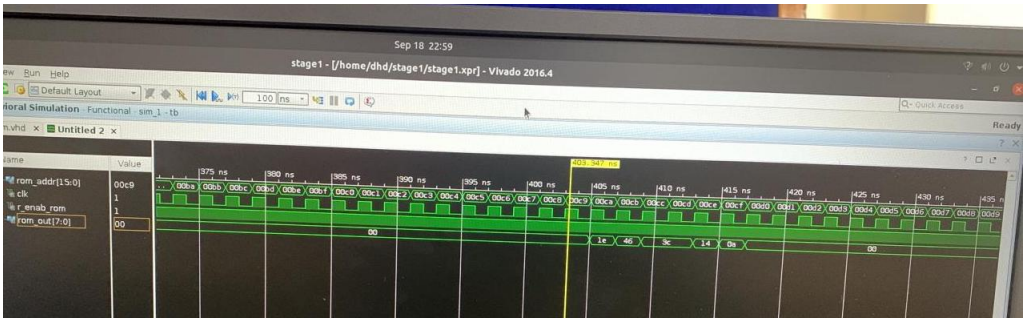
The testbench for this unit involved three components, testing the multiplication, testing the accumulation, and testing the shifting.

With the cntrl signal off, we test the multiplication and shifting for some set of value. Then, with cntrl on, we test the accumulation functionality and observe that the sum now reported is the previous val plus the computed multiplicand and divided by 32. This module, was tested and is reported to be running well. The waveform is attached below.



3.2 "rom.vhd" ROM

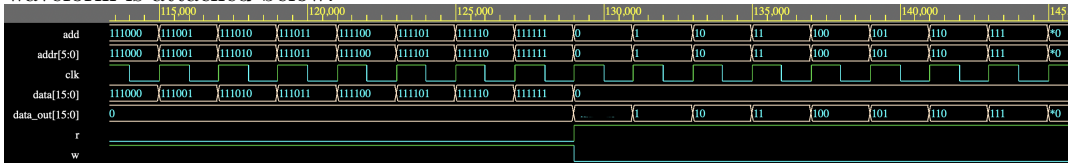
The testbench for this module tests whether the values (from the .mif files) stored at different indices of the ROM memory block are churned out when the relevant address is provided. The read_ enable signal is turned on, and a clocked output is tested and displayed as follows.



3.3 "ram.vhd" RAM

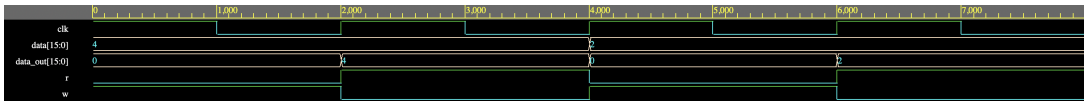
The Dynamic memory requires a little bit of testing, as it's functionality is diverse.

With the write_ enable signal, we write some values to the module. When the writing has been performed, we enable the read_ enable signal, and test whether these values match the ones we had written earlier. Since these match, the component is running smoothly. The waveform is attached below.



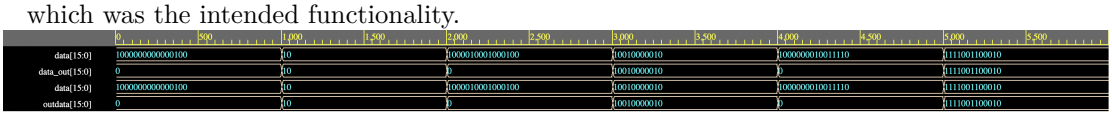
3.4 "reg.vhd" Register

The register is essentially a RAM with just a single index. The testing here involves the writing, reading and subsequent matching of the values. The read enable and write enable signals are similar to RAM. Since the matching is perfect, the component is operational. The following waveform supports the above arguments.



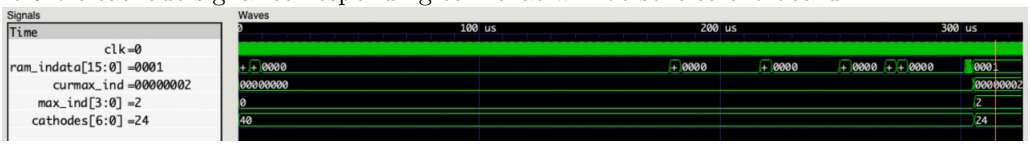
3.5 "relu.vhd" Comparator

The ReLU is Tested by supplying various inputs, and comparing it's outputs. For all negative inputs, the component returns 0 and for positive values it returns the value itself, which was the intended functionality.

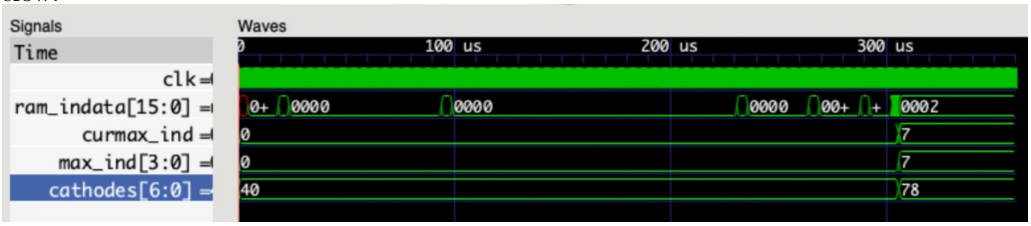


4 Testing the MLP on our system

When the input file "imgdata_digit2.mif" was given to ghdl installed on out laptops, the character 2 was recognised, as shown in the waveform of the simulation. This also shows the 7 bit cathode signal corresponding to 2 that will be sent to the board.

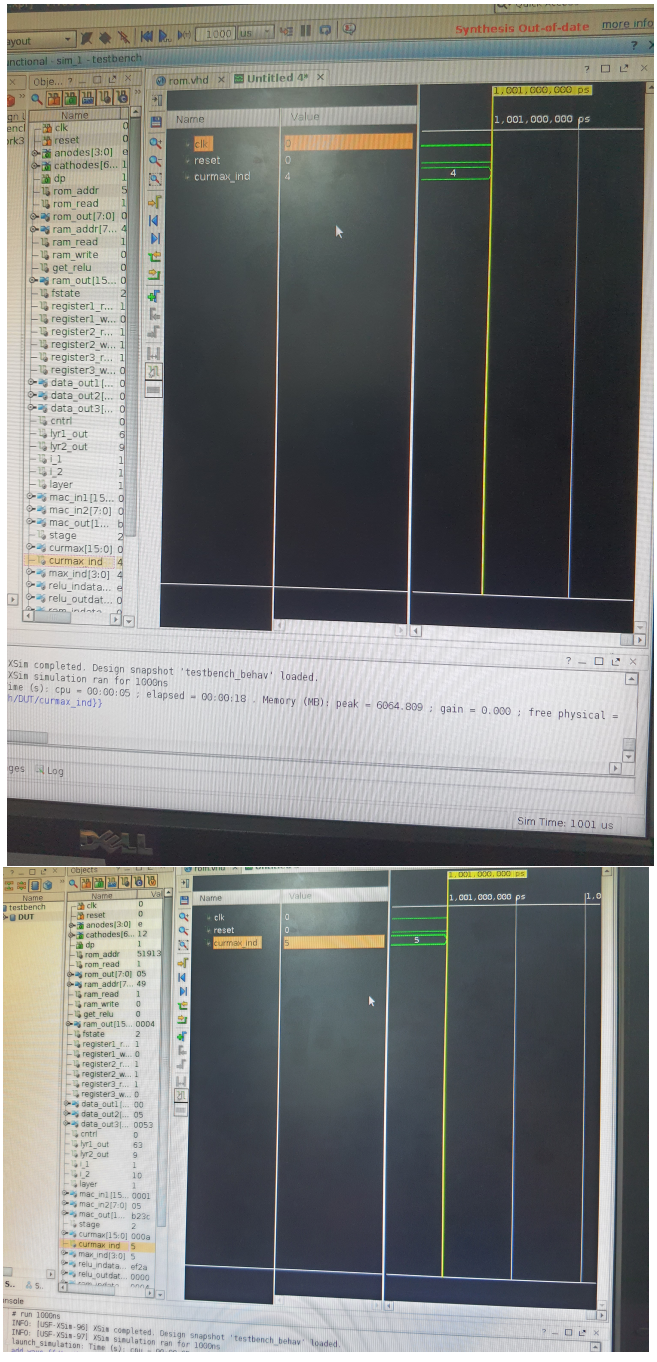


Similar test results for the file "imgdata_digit7.mif" are also attached, in that order, below.



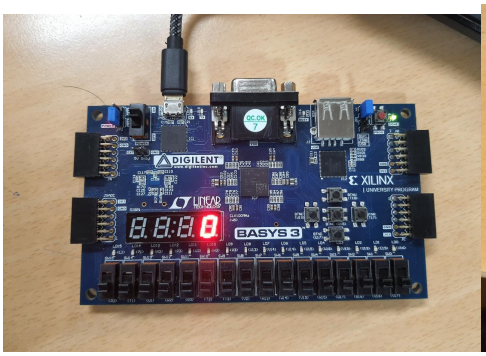
5 Simulation at the lab

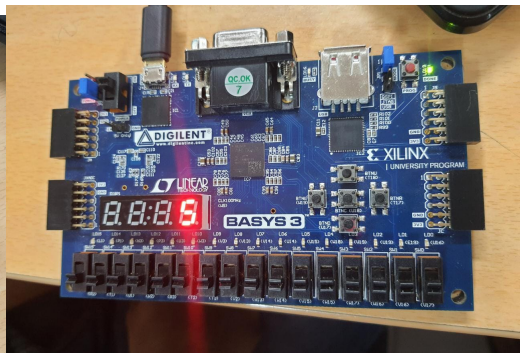
Upon running simulations on the Lab's computer, we found that our results were coherent with the one's obtained back home. For the files "imgdata_digit4.mif" and "imgdata_digit5.mif", the characters 4 and 5 are obtained in the waveform as follows :-



6 Synthesis and Implementation, Output

After successful simulation, we proceeded to run the files on the actual board. The output obtained when the file for 0 is run is as follows:


DIGILENT
XILINX
BASYS 3


DIGILENT
XILINX
UNIVERSITY PROGRAM
BASYS 3