Department of Computer Science & Engineering

Assignment 2: NEURAL NETWORKS

Abhinav Barnawal, Eshan Jain
Entry No.: *2020CS50415, 2020CS50424*
Class: *2201-COL215B*
Session: *2022-23*
Email: *cs5200415@iitd.ac.in, cs5200424@iitd.ac.in*

Course: *Digital Logic & System Design* – Instructor: *Prof. Preeti Ranjan Panda*
Submission date: *October 7, 2022*

Our Approach

This project is aimed to implement a 3-layer neural network inference in VHDL and test it on the BASYS3 FPGA board. The modular description is as follows:

1. **RAM (Random Access Memory):** It takes in the clock, an 8-bit address from where the data needs to be accessed, a 16-bit data output which is read from the given address, write enable, 16-bit data to be written into memory at the given address. This is used to store the temporary outputs produced by each layer and the final output.

2. **ROM (Read Only Memory):** It takes the address to be read from and outputs the corresponding data. It is used to store the image data, and the weights and biases data read from the mif file.

3. **Register:** It takes in a clock, a 4-bit address, a 16-bit input, a write enable, and returns a 16-bit output. Its job is to store the temporary data generated during computation across various cycles.

4. **MAC (Multiplier Accumulator Block):** It takes in two operands of 15 bits and 8 bits, multiplies them and adds them to a 16-bit accumulator, and generates a 16-bit output. It performs vector-matrix multiplication to produce the output for each layer.

5. **Shifter:** It takes in a 16-bit number as input and returns a 16-bit output down-shifted by 5 bits. Its job is to perform integer division on the 16-bit result of MAC by 32, using a down shifter.

6. **Comparator:** It takes in a 16-bit input and returns the 16-bit ReLU output. The ReLu output of an integer is the positive part, that is, it returns the number itself if it is positive or else 0. It is used to implement the ReLU operation of the neural network.

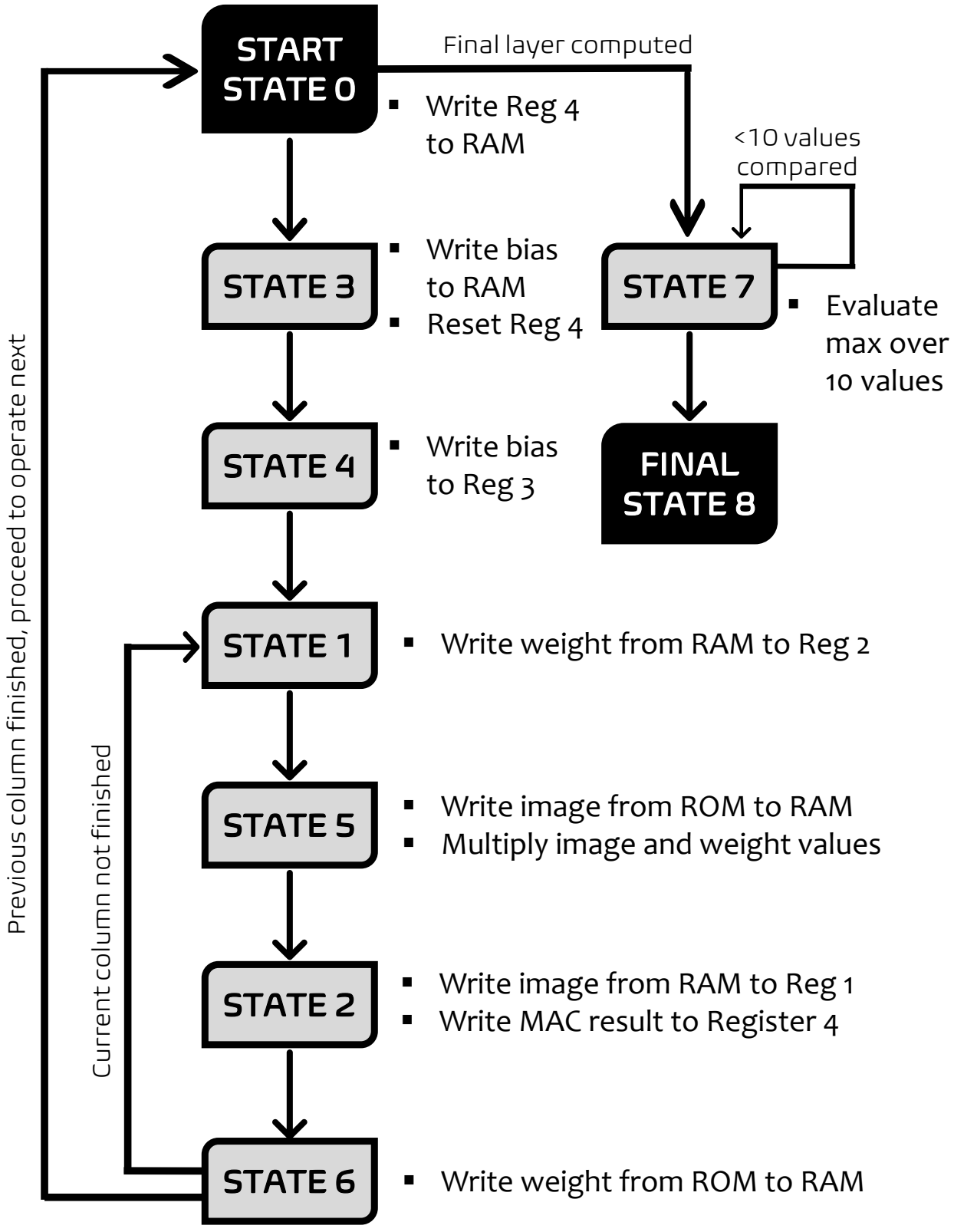
7. **FSM:** The FSM has 17 states in total. We have separated the states of the FSM as per the requirement of reading different values from the ROM (image, weight, bias) at different stages of the control. We also have made separate states for handling the different levels (the hidden layer and the output layer). Also, for each state, we have a wait state before it, which waits for a clock cycle to ensure that the read/write operation is complete and there are no conflicts. In verbose: in the state 0, we update the read address of the current bias value to be added into the ROM read address, similarly in states 1 and 2, we update the image and the weight values to be multiplied respectively into the ROM read address. The rest of the states are used to handle the level transitions and cycle delays appropriately.

8. **Control Block:** The control block is used to set the addresses and the data to be written and read, based on the state of the FSM.

9. **Decoder:** It is used to decode a number into the cathode signals to be displayed on the FPGA board.

10. **Design Block:** It is the top-level entity that instantiates and connects all the components.

FINITE STATE MACHINE



Simulation Description

We have used `ghdl and gtkwave` software to simulate the design and display the waveform. Please follow the given guidelines to simulate and test the code seamlessly.

- Uncomment the lines 12 and 137 of `design.vhd` before simulating the design.
- If `ghdl` is used for simulation on **windows**, then just run the command `make` and `make wave` to see the waveform.
- To provide new input and weight files, update the respective paths at lines 39 and 40 of `design.vhd`.
- To clean up unwanted files from the directory run `make clean`.

Only the major signals have been displayed here, with only the values during the end of the simulation due to the large number of cycles and huge waveform. Further, please note that all other signals are self-explanatory. For more information, please view `wave.gtkw` using `gtkwave` by running the command `gtkwave wave.gtkw`. The brief description is as follows:

`clk[1]`: Clock input of the register (`10 ns`).
`result[3:0]`: Result of the execution, correctly updated only after state 8.
`state`: State of the FSM.
`level`: Level of computation of the Neural network.

Simulation Results

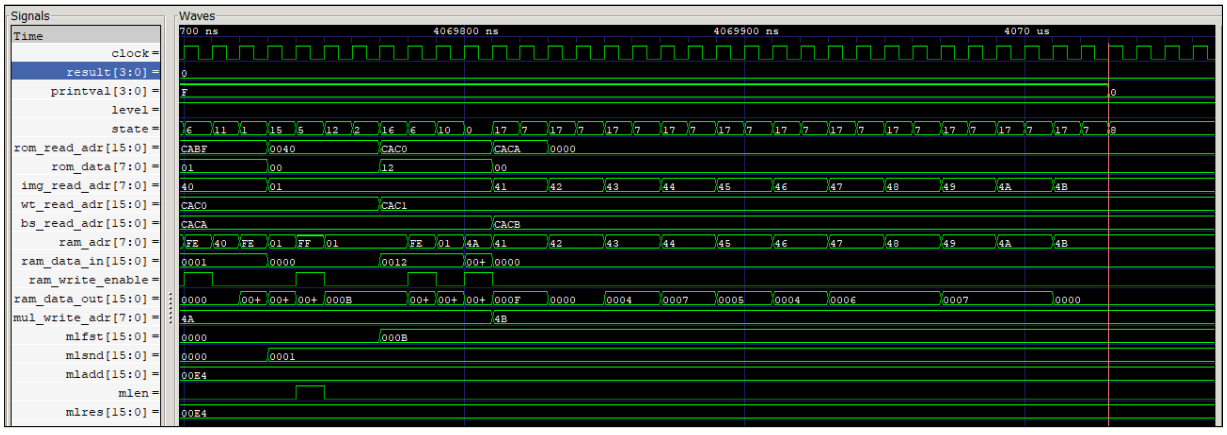


Figure 1: Simulation result with predicted class 0

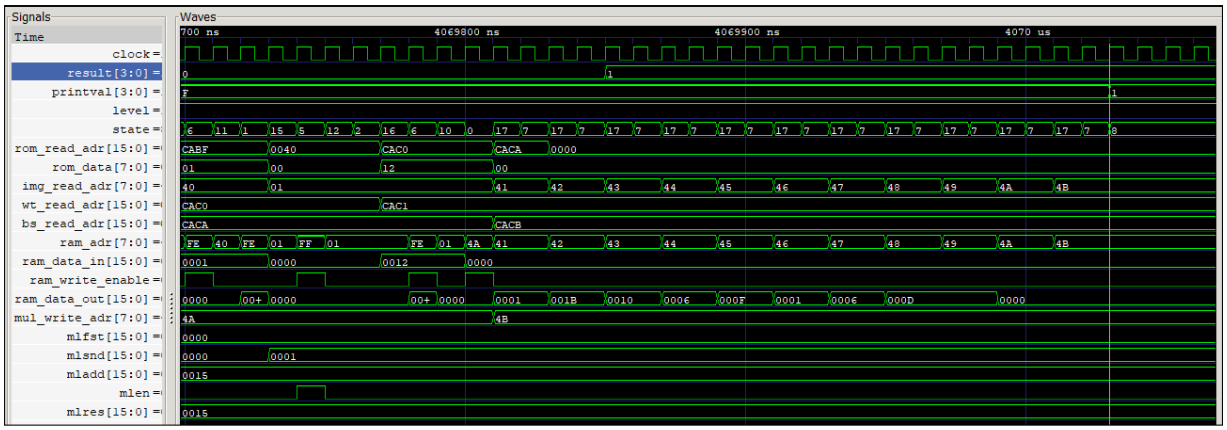


Figure 2: Simulation result with correct predicted class 1

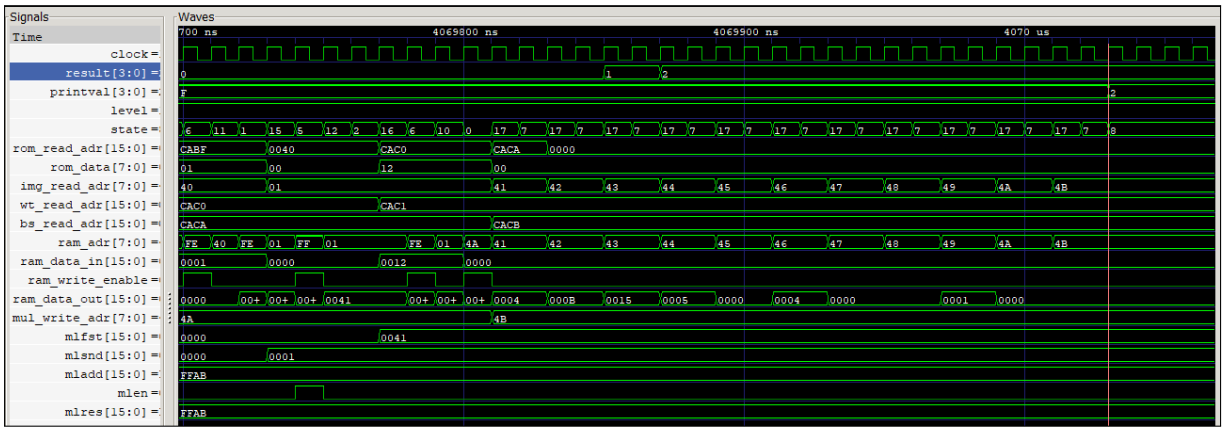


Figure 3: Simulation result with correct predicted class 2

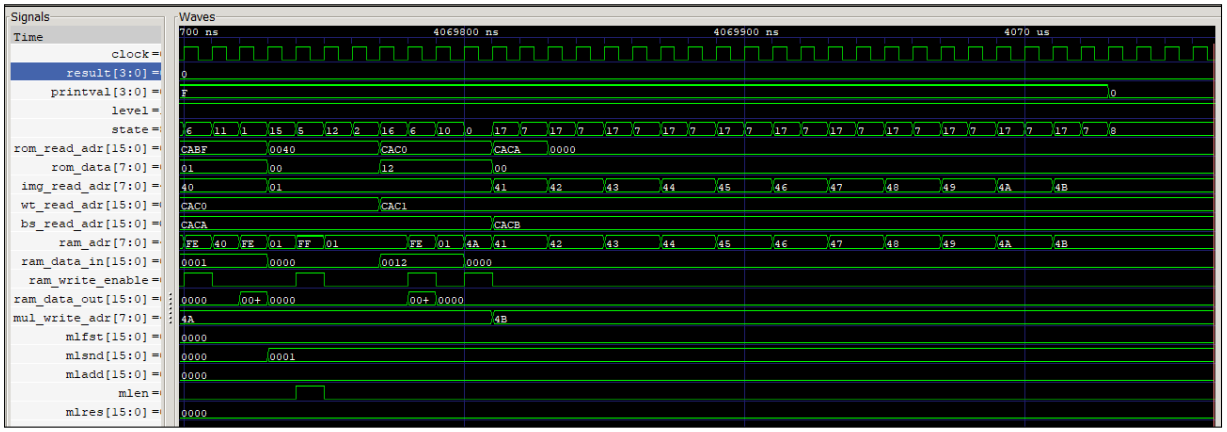


Figure 4: Simulation result with incorrect predicted class 0 for class 3

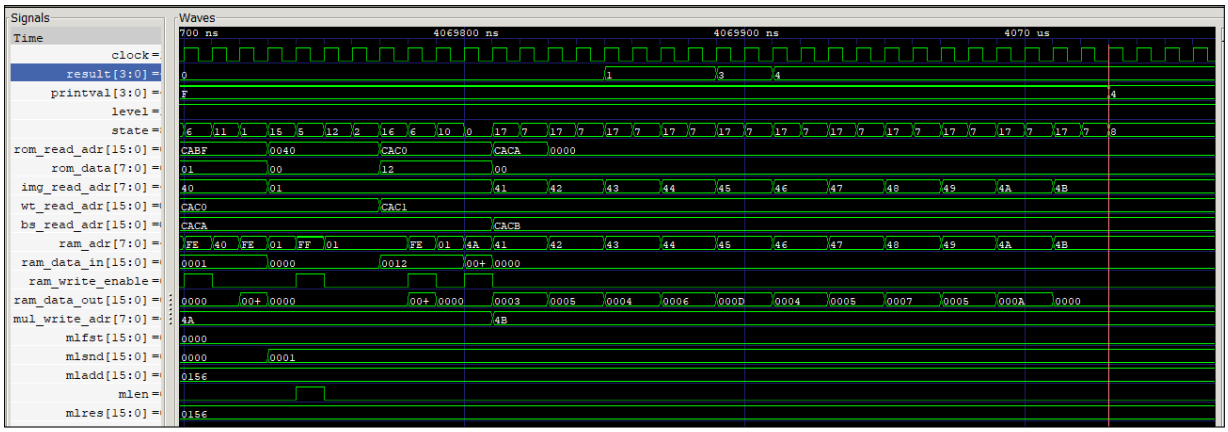


Figure 5: Simulation result with correct predicted class 4

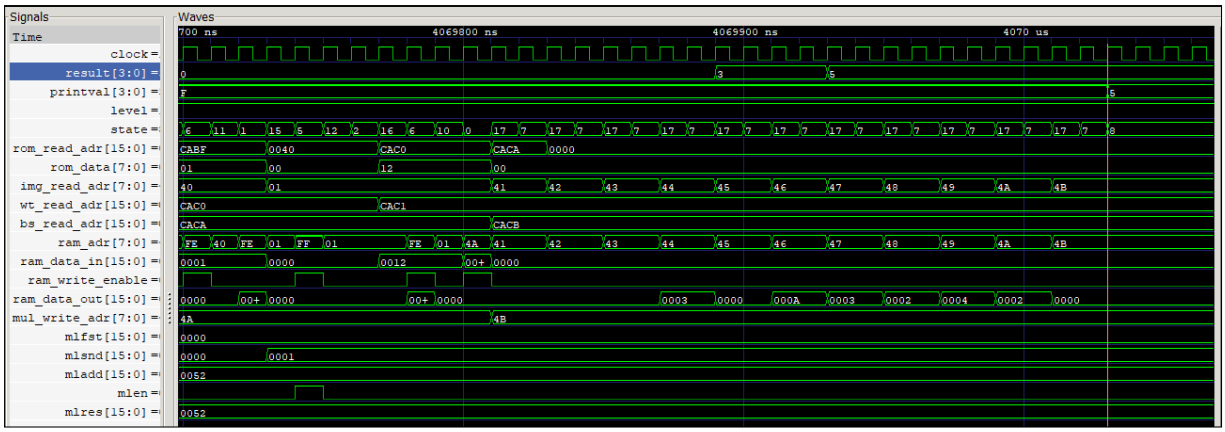


Figure 6: Simulation result with correct predicted class 5

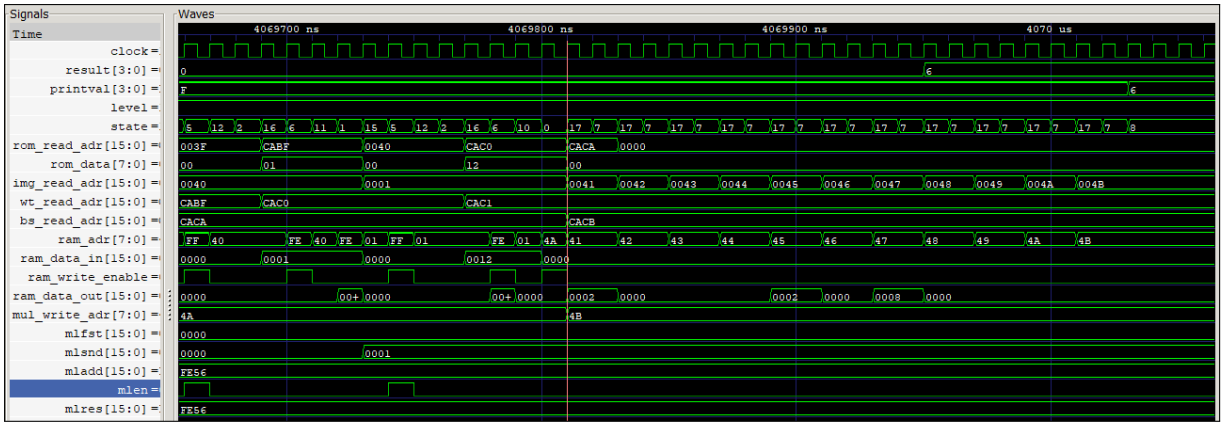


Figure 7: Simulation result with correct predicted class 6

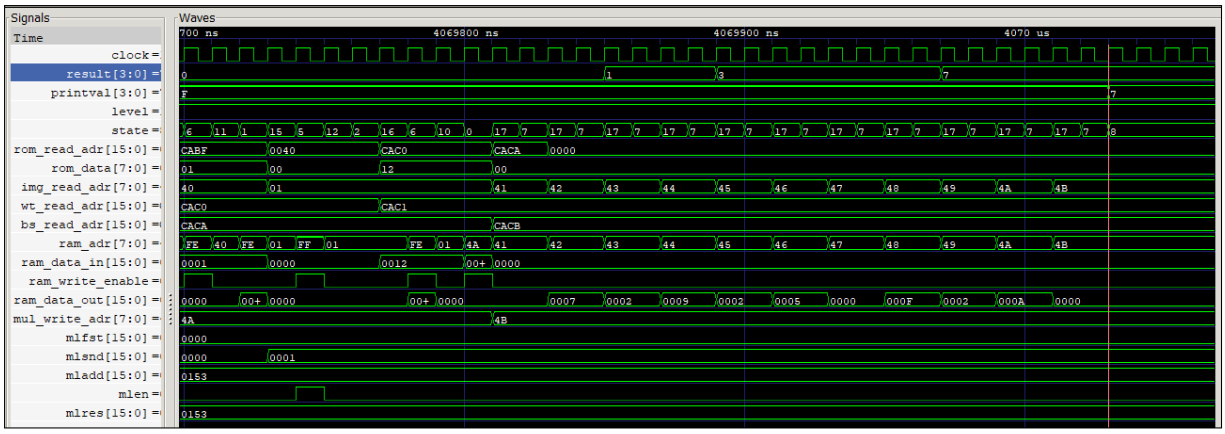


Figure 8: Simulation result with correct predicted class 7

Configured FPGA Board

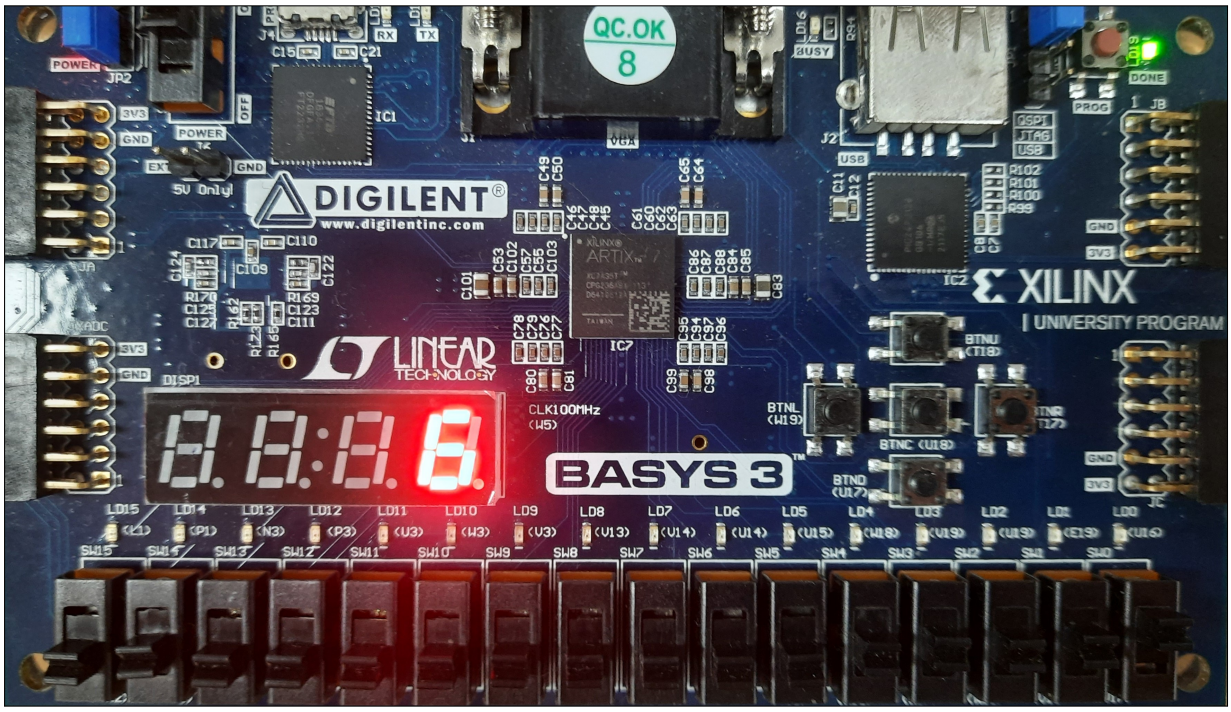


Figure 9: Displaying final output of the design for class 6 on the FPGA board

Block Diagram

We describe the architecture of our implementation in the following figure: