REPORT FOR COL215 HARDWARE ASSIGNMENT-3

Objective:

To design a core in VHDL which multiplies two input matrices (A and B) of size 128*128 containing 8-bit unsigned integers to give an output matrix (C) using a MAC, RAM, ROM, registers and an FSM to stitch together all the parts.

Procedure:

The created matrix multiplier is based on sequential logic and has two main paths for the flow of variables. The first is the data path consisting of sub-components where major arithmetic operations and storage occurs. The second is the control path which decides the proper order in which the above components are called in order to facilitate the proper multiplication of the matrices.

1.DATA PATH:

1.1.**ROM**:

We create two ROM elements using the IP catalog in Vivado. These ROM elements are used to store the input matrices A and B in a 1-D fashion. We provide the address (index in this 1-D array) of the needed element as the input to the ROM in order to get the corresponding stored element of the matrix.

1.2.**Register**:

A register is a storage element (for 8 bits in our case) used in sequential logic. It consists of a read(RE) and write(WE) enable used to prevent mixing of operations. We implement a register in VHDL by creating an entity and writing the above logic. We then use 2 such registers each of which corresponds to an input matrix.

1.3.**RAM**:

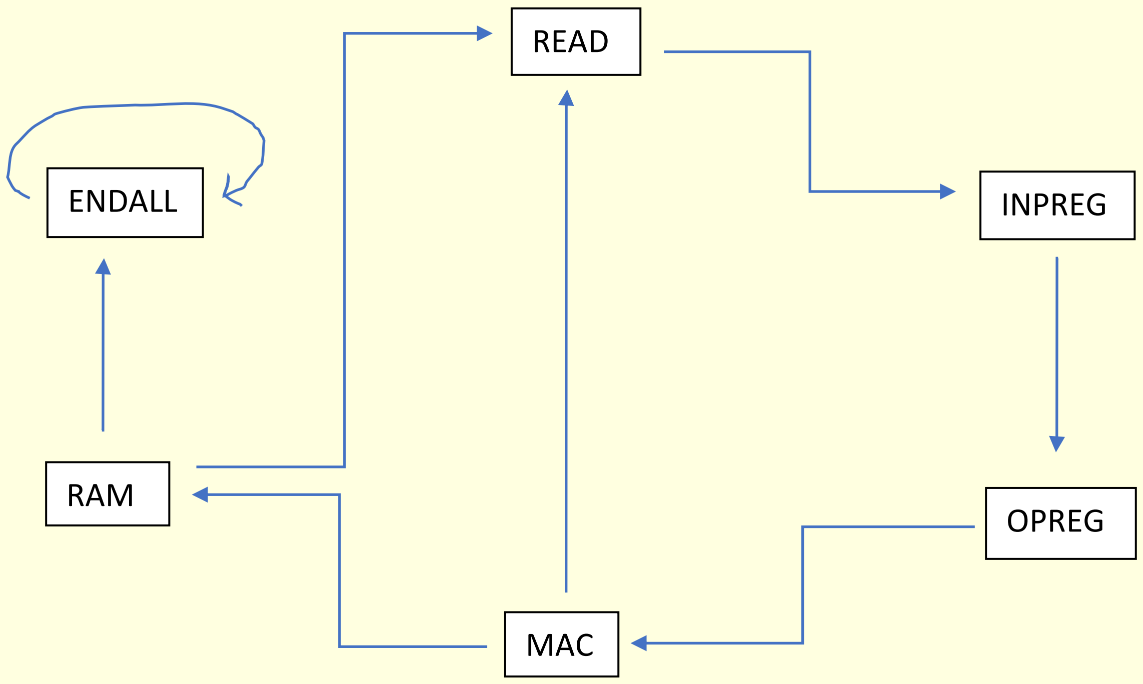
The RAM is yet another storage element created using the IP catalog of Vivado. It is similar to the ROM except that it also allows the write operation. We use the RAM to store the final output matrix by writing into it and then read it to display the required index on the board.

1.4.**MAC**:

The MAC (or Multiply-Accumulate block) is the main unit for the arithmetic operations namely consecutive addition followed by multiplication of the row-column pairs. We convert the 8-bit hexadecimal numbers to integers and for the multiplication part.

2.CONTROL PATH:

The control path consists of a finite state machine (FSM) with the below state diagram. There are 6 states in the FSM each corresponding to some part of the entire logic.



1.**READ:**

We read the current 8-bit numbers from the ROM and send them to the registers.

2.**INPREG:**

We receive the 8-bit numbers sent by the ROM in this state.

3.**OPREG:**

We take the 8-bit numbers stored in the registers and send them to the MAC.

4.**MAC:**

We multiply the two 8-bit numbers and add them to the current result in this state.

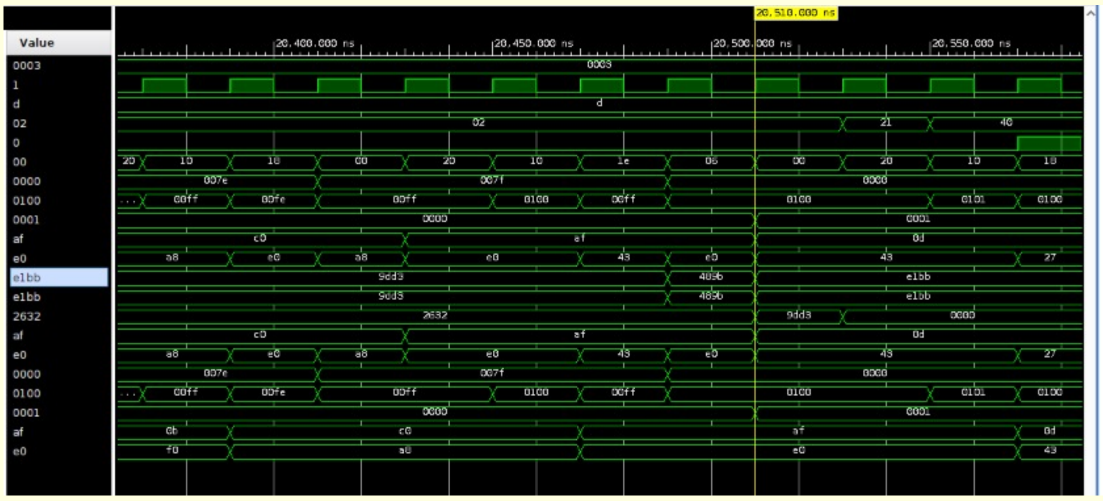
5.**RAM:**

The final value at the index of output matrix which has been computed now is sent to store in the RAM in this state.

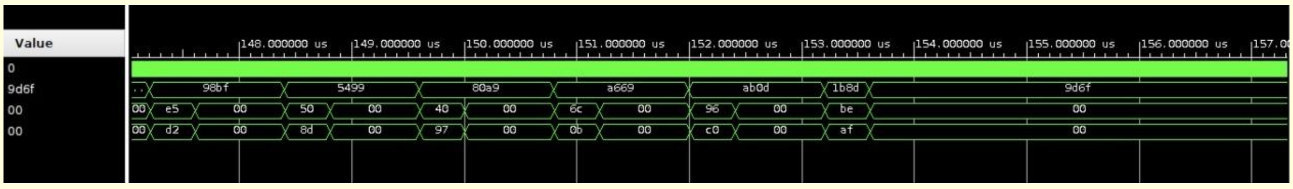
6.**ENDALL:**

This state marks the end of the computation of the output matrix. We cannot exit this state once reached. We have a self-loop here which helps us check the various indices of the output matrix by changing switches.

Simulation Snapshots:

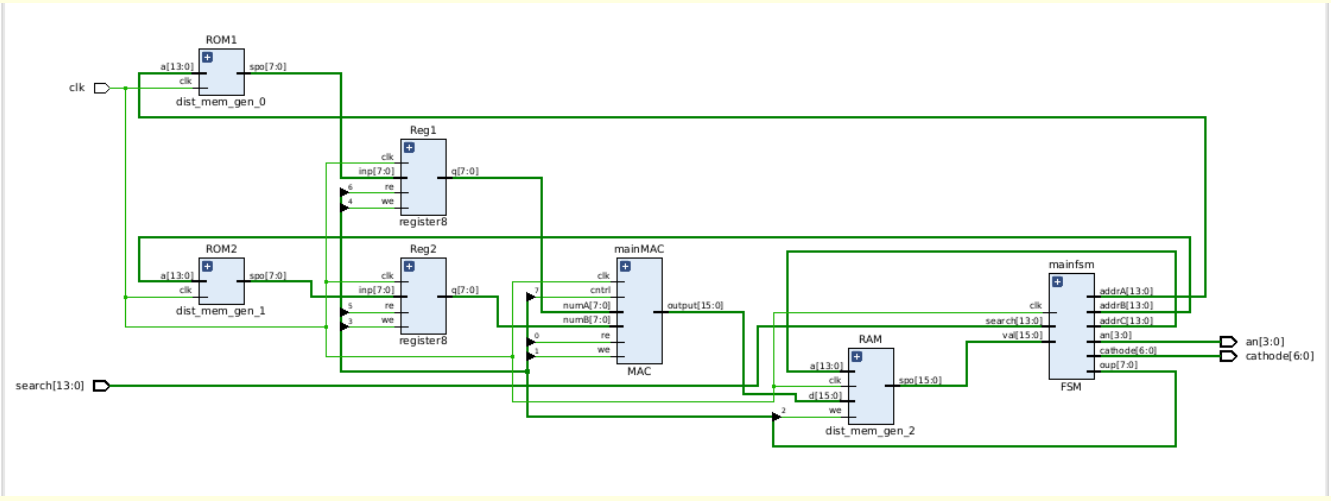


The above diagram shows the working of the FSM with all the variables.



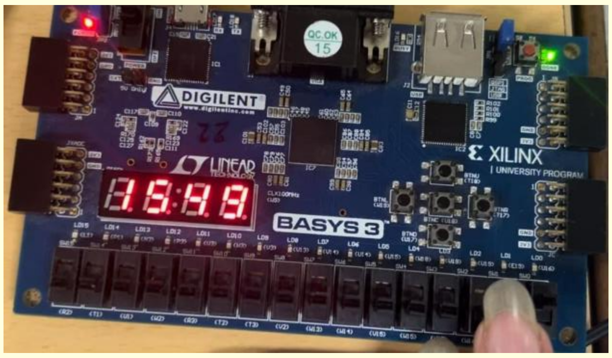
The above diagram shows a few intermediate results followed by the single output.

Block Diagram:



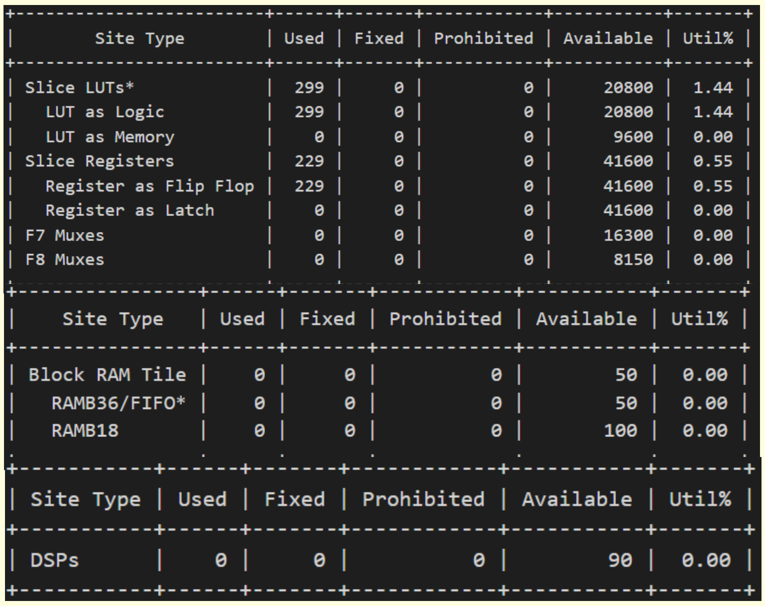
The above block diagram shows the modularity of the code as we have divided it into the various elements.

Lab Work:



The output when multiplying the matrix given in test COE file in column major order with itself at the 1st row and 2nd column.

Synthesis Report:



```
+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |  299 |     0 |          0 |     20800 |  1.44 |
|   LUT as Logic          |  299 |     0 |          0 |     20800 |  1.44 |
|   LUT as Memory         |    0 |     0 |          0 |      9600 |  0.00 |
| Slice Registers         |  229 |     0 |          0 |     41600 |  0.55 |
|   Register as Flip Flop |  229 |     0 |          0 |     41600 |  0.55 |
|   Register as Latch     |    0 |     0 |          0 |     41600 |  0.00 |
| F7 Muxes                |    0 |     0 |          0 |     16300 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |      8150 |  0.00 |
+-------------------+------+-------+------------+-----------+-------+
|     Site Type     | Used | Fixed | Prohibited | Available | Util% |
+-------------------+------+-------+------------+-----------+-------+
| Block RAM Tile    |    0 |     0 |          0 |        50 |  0.00 |
|   RAMB36/FIFO*    |    0 |     0 |          0 |        50 |  0.00 |
|   RAMB18          |    0 |     0 |          0 |       100 |  0.00 |
+-----------+------+-------+------------+-----------+-------+
| Site Type | Used | Fixed | Prohibited | Available | Util% |
+-----------+------+-------+------------+-----------+-------+
| DSPs      |    0 |     0 |          0 |        90 |  0.00 |
+-----------+------+-------+------------+-----------+-------+
```

We can clearly see the number of used LUTs, BRAMs (Block RAM), DSPs and Flip-Flops. This is the synthesis utilization report. The number of LUTs further decreased to 297 post-implementation.