COL215 - Hardware Assignment 3 Report

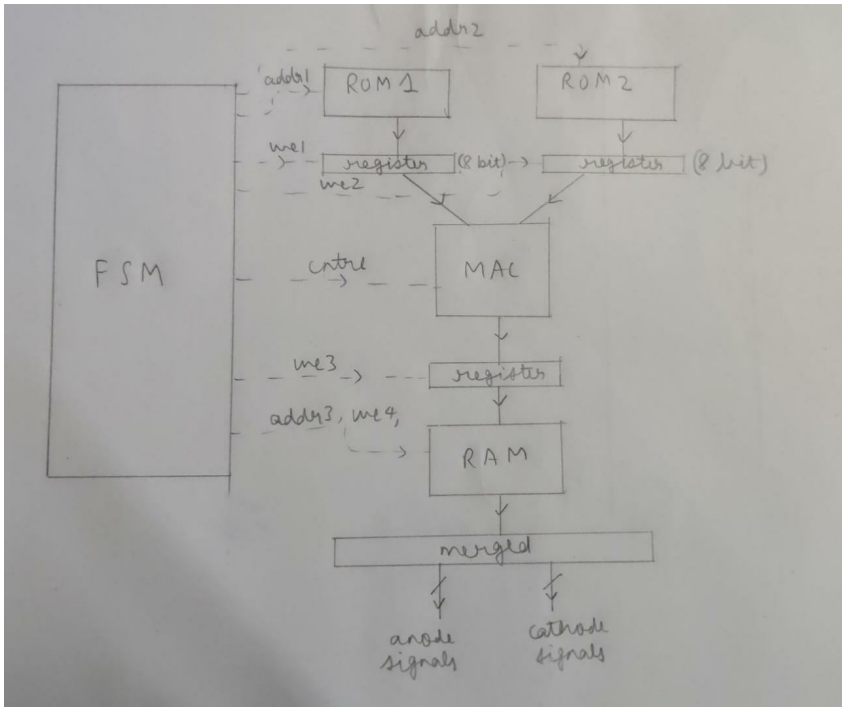
Ashish Arora (2021CS10069) & Aaveg Jain (2021CS10073)

Brief description of the problem: Given two input matrices of size 128x128 and an 8-bit unsigned integer, design a core in VHDL which performs matrix multiplication given input matrices

Modules/Components used in our projects

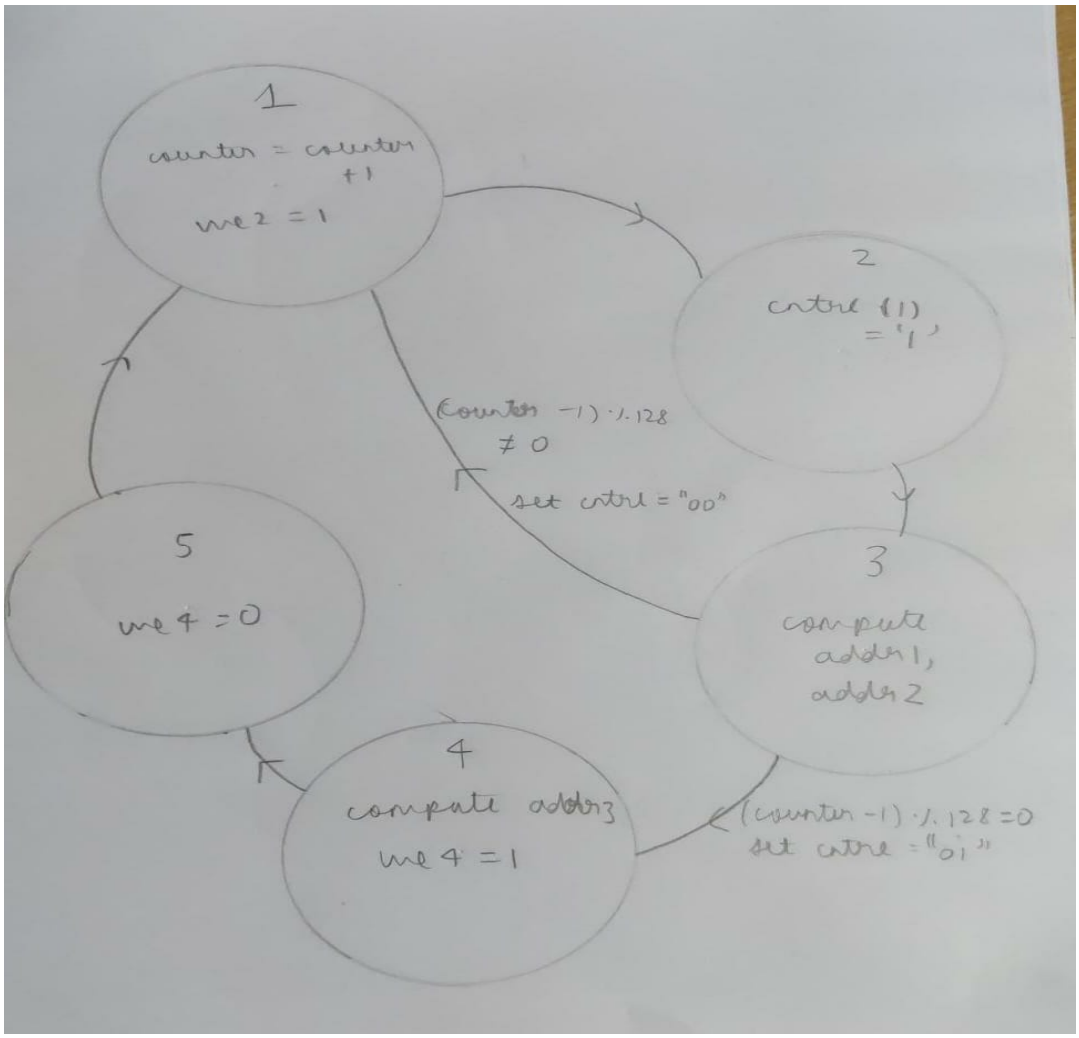
1) **(fsm.vhd)** A Finite State Machine (FSM) which controls the overall flow of the program. This file contains all other modules as components
2) **(eight_bit_register.vhd)** Register for storing the value obtained from ROM
3) **(sixteen_bit_register.vhd)** Register for storing the value to be written in the RAM
4) **(multiplier.vhd)** Multiplier-Accumulate Unit (MAC) for multiplying two 8 bit numbers and storing the cumulative sum until a row and column of the two input matrices are fully multiplied
5) **(dist_mem_gen_0 & dist_mem_gen_1)** Memory blocks for ROM to store the two input matrices respectively
6) **(dist_mem_gen_2)** Memory block for RAM
7) **(merged.vhd)** Used to display the output on 7-segment display

Heres the overview of the design using a block diagram



We now explain how they work together to produce the required result
Finite State Machine (FSM): This is the backbone of our code and controls the flow of accessing values from the ROM, processing them, and later writing them to the RAM

Here is a diagram showing how it works



The corresponding states in the code are:
1 -> 00001 (reading from rom to register)
2 -> 00010 (values from register to mac)
3 -> 00100 (mac operations take place, addresses to next memory block computed)
4 -> 01000 (value transferred to register, address to ram computed)
5 -> 10000 (values stored in ram)

We use counter to track our statechange and to compute addresses to memory
Define base = 128 and n = 128*128*128 + 1
We update counter by +1 each time we reach the state 1

For computing the address of the memory to access values of the matrix for next computation cycle, we use the following formula

addr1 = (counter mod base)*base + counter/(base*base)
addr2 = ((counter/base) mod base)*base + counter mod base

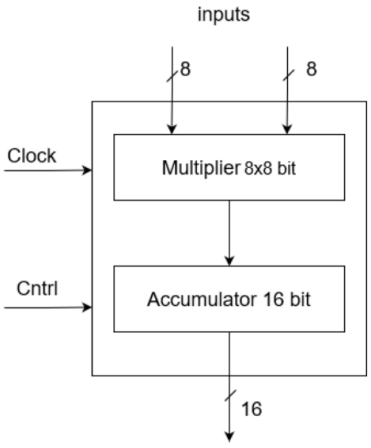
For computing the address of RAM to store value obtained we use the formula
addr3 = (((counter-1)/(base*base)) mod base) + ((((counter-1)/base))mod base)*base

As in the figure, When we are at state 3 we have two path options. One is to go to state 1 and the other is to go to state 4. When we are multiplying a row and column of the matrix, we need to go through 128 cycles of state1 -> state2 -> state3, to multiply each entry and then accumulate the sum. One's we are done we have the corresponding entry for the output matrix and so we go to state 4 from there.
Hence we check if (counter -1) mod base = 0, If it is we go to state 4 else we go back to state 1 to complete our current computation.

We do this till counter <=n

Register
Register has a clock, data and we input. In each clock cycle it checks whether we = 1 and if it then it updates its memory signal to store the input data and sets the output to memory

Multiplier-Accumulate Unit (MAC)
The MAC unit has the following structure



It calculates the product of the two eight bit input signals passed.
Then if cntrl(0) is 0 it resets the accumulated sum and starts afresh, setting the current sum to the newly calculate product
Otherwise if cntrl(0) is not 0, then it adds the newly calculate product to the accumulated sum
For MAC Control we check the value of cntrl(1), it is activate when cntrl(1) is 1

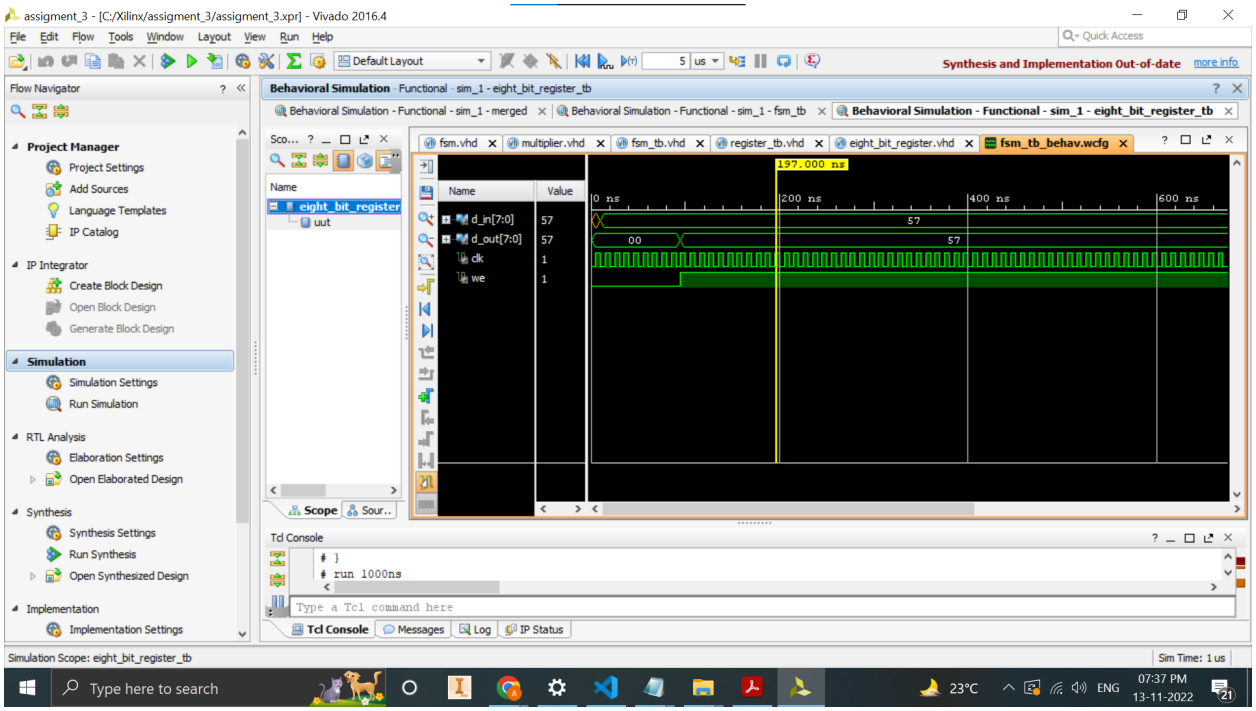
cntrl(0) is basically set to 0 whenever we begin the multiplication of two rows of a matrix, we then change it to 1 throughout the computation so that the sum gets accumulated then once the rows are multiplied its later set back to 0

The **ROM memory blocks** are generated using the .coe file given

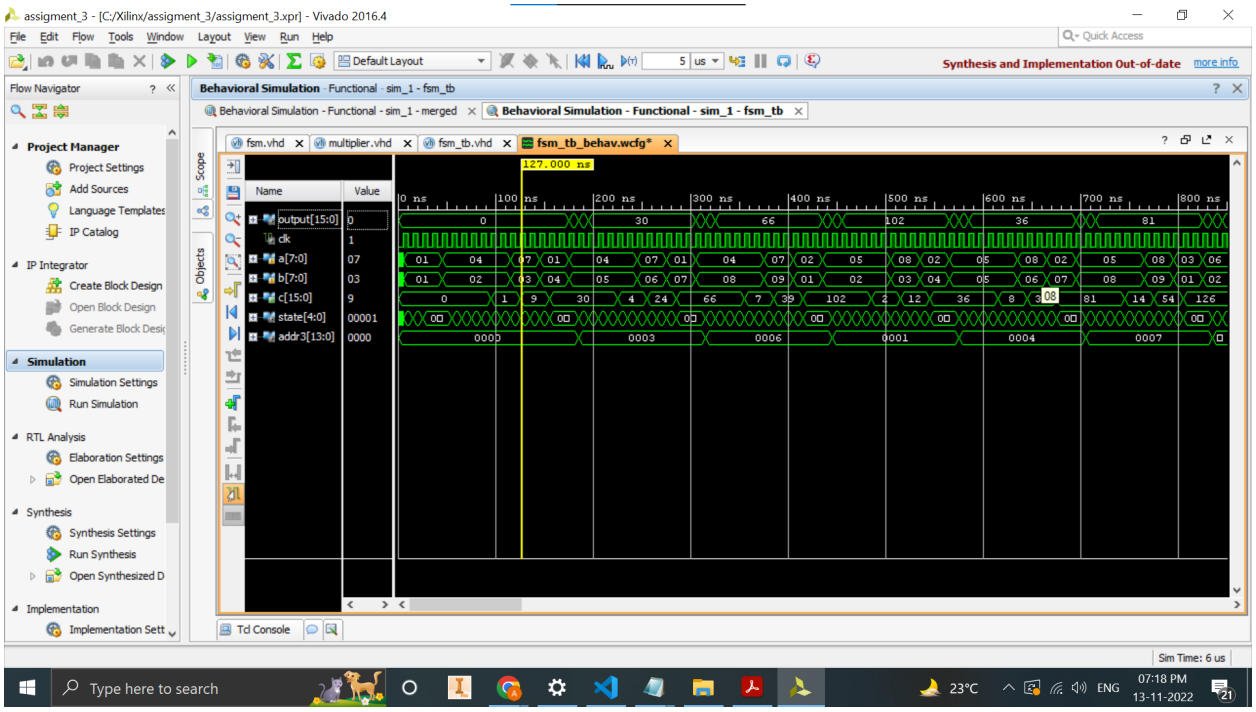
We then use the code made in assignment 1 for displaying output on 7-segment display

Simulated Waveforms
The following matrix was used as input for the simulation
1 2 3
4 5 6
7 8 9

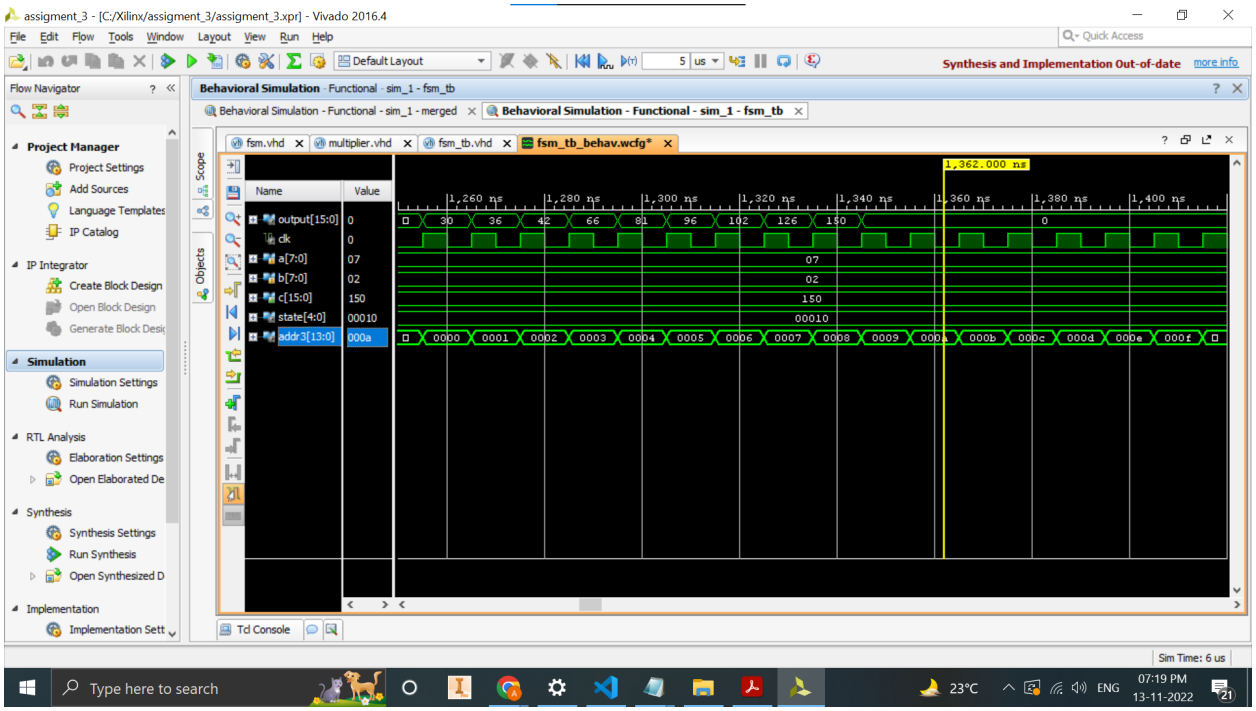
eight_bit_register



demonstrates simulation of MAC (adding and accumulating) ; also shows RAM being written with values



after fsm has completed running, final RAM values displayed by changing address(addr3)



showing the state changes