Indian Institute of Technology Delhi

COL 215 - Digital Logic and System Design

Assignment 3

Hardware for Matrix Multiplication

Garv Nagori, Aryan Sharma
2021CS10549, 2021CS10553

Contents

1 Introduction

The objective of this assignment was to design hardware in VHDL to multiply two 128 × 128 matrices. The inputs were given in a *coeffecient* (.coe) file and were stored in two ROMs in the hardware whereas the output was stored in a RAM.

2 Strategy

We are going to use registers to store intermediate memory in between multiplications. A Multiplier-Accumulator circuit (MAC) is used to multiply two numbers and add in the register. Once all 128 multiplications have been performed we output the data from the register and write it to the RAM and continue the cycle for the next element of the matrix. We will use a Finite State Machine to control the MAC unit and perform data transfer between ROM, the intermediate registers and RAM.
We have used the following components, **Main, MAC, Register8, Register8_1, FSM, Display, Clock, MUX, Decoder**. The last three were the same as Assignment 1 and are used solely to display all the 4 numbers simultaneously and display to control the three of them. The **main components** here are **MAC and FSM**. MAC for the data path and FSM for the control path. MAC takes input two numbers, read and write enable, control and outputs the number after multiplying and adding 128 times. FSM has six states which it changes to according to the previous state and controls the flow of data to and from MAC and ROM/RAM. The block diagram of our implementation is shown below.

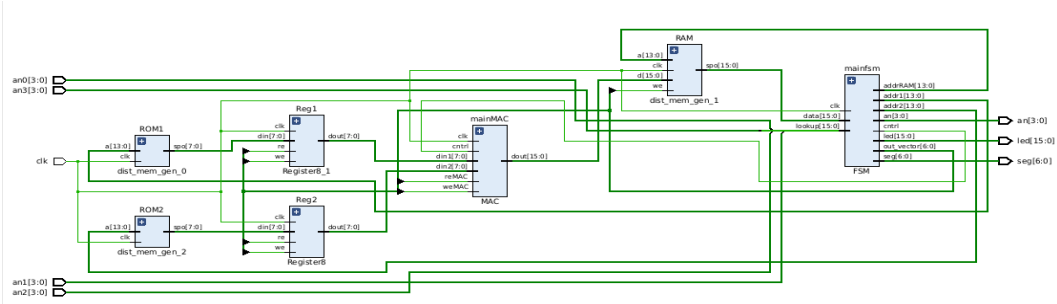


Figure 1: Block Diagram

2.1 Registers

This component acts as a temporary memory device to store the data from the ROM. There are two registers named Register8 and Register8_1 which are the memory components. There are two signals re and we for read enable and write enable respectively. Read enable is given priority over write enable. The register is only activated whenever there is a rising edge on the clock. The temporary variable store acts as the memory variable. When we is on the store is overwritten and the new value is assigned. Whereas when re is on the the output dout is assigned to store. The value stored is of 8 bits since the data from the ROM is of 8 bit length.

2.2 MAC

There are six signals as input - reMAC, weMAC, din1, din2, clk and cntrl. din1 and din2 are the two numbers which we have to multiply and add in the accumulator. reMAC and weMAC are read enable and write enable respectively. When weMAC is on we multiply the two numbers and add it to the built-in register and when reMAC is on we assign the output dout to the value stored in the accumulator. Now the cntrl signal is to identify if 128 multiplications have occured and we have to restart the accumulating process. ac is the internal memory to store and add the next set of numbers.

2.3 FSM

The Finite State Machine is the brain of the hardware. There are six states in the FSM through which it can go - STROM, STReg_write, STReg_read, STMAC, STRAM, STEND. data is sent to display on the FPGA board. lookup is the switches on the board and led is the corresponding leds of the switches. seg and an are for the seven segment display. addr1 and addr2 are the addresses of the ROM which we have to access to sed to the MAC. addrRAM is the address to send to RAM to write the data on the RAM or to read and display on the FPGA Board. cntrl is the signal being sent to MAC with the same name. out_vector is a vector correspong to reReg1, reReg2, weReg1, weReg2, weRAm, weMAC and reMAC respectively. We call the display the data in the RAM. We maintain 4 variables row1, row2, col1 and col2 to traverse the two input matrices (row2 = col1 here but seperate variables to make it easier to understand).

The states of the FSM in detail are: (a) **STROM** At this state, we calculate the addresses of the input elements using ro1, row2, col1 and col2 and send it to the ROM so that we can start reading from it. out_vector is "0000000" since we do not need to write or read from any register, MAC or RAM. After this we call state STReg_write.

(b) **STReg_write** At this state, we load the values from the ROM into the registers. We keep sending the addresses to the ROM so that it doesn't stop sending data before it is loaded into the register. out_vector is "0011000" since we want to write over the existing values in the two registers. This state is seperate from STROM since if there was only one state for both then there might have been an error if the register is written over before the ROM can start sending the data. The next state in the FSM is STReg_read.

(c) **STReg_read** Here, we read the values loaded in the registers to the MAC. out_vector is therefore "1100000" and the next state is STMAC.

(d) **STMAC** In this state, we read the values into the MAC and we do not disable reReg1 and reReg2 so that the the values are loaded before being stopped. Here, we also check if we are starting from a new iteration (or row2=col1=0). If so, then we set cntrl as 1 and out_vector as "1100111" to start loading the value into the RAM. Then the next state is STRAM. If not then we loop back to state STROM and out_vector is "1100010".

(e) **STRAM** Here we just write the data from the MAC into the RAM. If row1 = 128 then that means we have reached the end of the matrix multiplication and thus send the FSM to state STEND otherwise loop back to STROM. out_vector for STROM is "0000101" and for STEND is "0000100".

(f) **STEND** The matrix multiplication is over at this step and now we only need to read from the RAM according to the address given by the switches and display it on the FPGA Board.

In the sequential block we also increment row1 and col2 if we have reached the end of the line (row2 = col1 = 127) and reset row2 and col1 to 0.

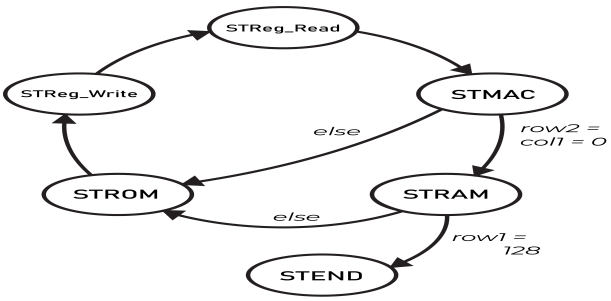


Figure 2: FSM used in the Hardware

2.4 Display

We have used programs from Assignment 1 in our implementation, which are used to display the the digits on the seven-segment display. As mentioned before these are the components Decoder, MUX, Clock. We call this component in the FSM since we read from the RAM if the state reached is STEND.

3 Main Component

We use all the above components together in the **Main** component. We call all the components here and map all the data of the components to temporary signals. The clock we use is of 100 MHz generated by the FPGA Board. This completes our data path and control path. As a good representative measure we have also mapped the fourteen leds corresponding to the address of the RAM we have to find out.

4 Simulations and Test Runs

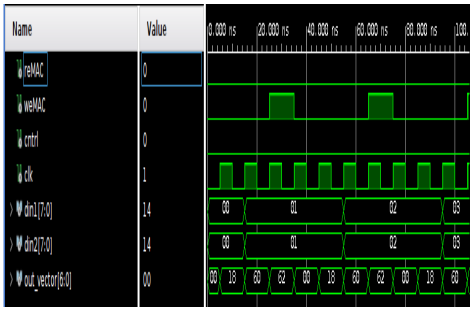
4.1 'FSMsim'

Simulation of the 'FSM' and 'MAC' together. '00', '18', '60' and '62' denote the different out_vector and thus different states.

```
begin
    a1 : FSM port map (clk => clk, cntrl => cntrl, addr1 => addr1, addr2 => addr2,
            addrRAM => addrRAM, out_vector => out_vector, data=>data, lookup=>lookup);
    UUT : MAC port map(reMAC => reMAC, weMAC => weMAC,din1 => din1,din2 => din2,
            clk => clk,cntrl => cntrl,dout => dout);
    reMAC <= out_vector(0);
    weMAC <= out_vector(1);
    re1 <= out_vector(6);
    re2 <= out_vector(5);
    clk <= not clk after 5 ns;
    process(re1,re2)
    begin
        if(re1 = '1') then
            din1 <= std_logic_vector(to_unsigned(to_integer(unsigned(din1) + 1),8));
            din2 <= std_logic_vector(to_unsigned(to_integer(unsigned(din2) + 1),8));
        end if;
    end process;

end Behavioral;
```

(a) FSMsim.vhd

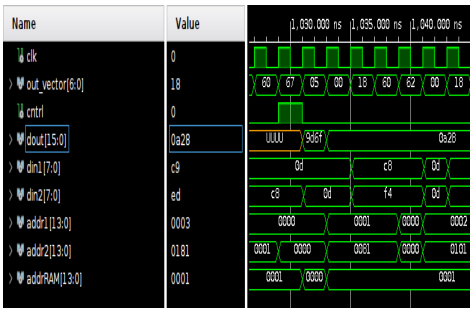


(b) Output Waveforms

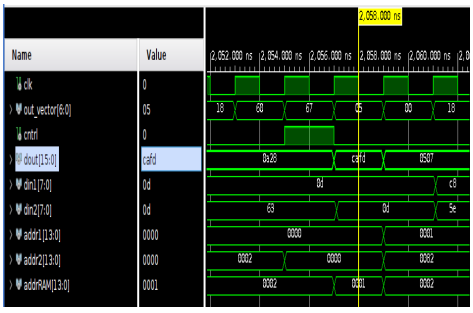
Figure 3: Simulation of the FSM.vhd file to check it

4.2 'Main'

Simulation of the main function to check the first two values in the RAM.



(a) At index 00 - 9d6F



(b) At index 1 - CAdF

Figure 4: Simulation of the Main.vhd file to check it

5 Conclusion

In this Assignment we designed hardware to multiply two matrices. Coding in a particular language gives us a time complexity of $O(n^3)$ which takes a few seconds to run for n=128 whereas the hardware can do the same in a few microseconds.