Specification and Simulation of Digital Systems – VHDL Assignment Matrix Multiplier

Overview

The goal of this assignment is to implement a device that is capable of doing dot matrix multiplication between two 32x32 matrices and store the result in a third matrix of the same dimension.

Data are in fixed point format Q16.16, this means 16 bits for the integer part and 16 bit for the fractional part. Words of the matrices are made of 32 bits.

The module allows to:

- Write the input matrices by direct addressing (row address and column address).
- Read the output matrix again by direct addressing as writing operation.
- Trigger the computation
- Reset matrices values via a reset synchronous signal.
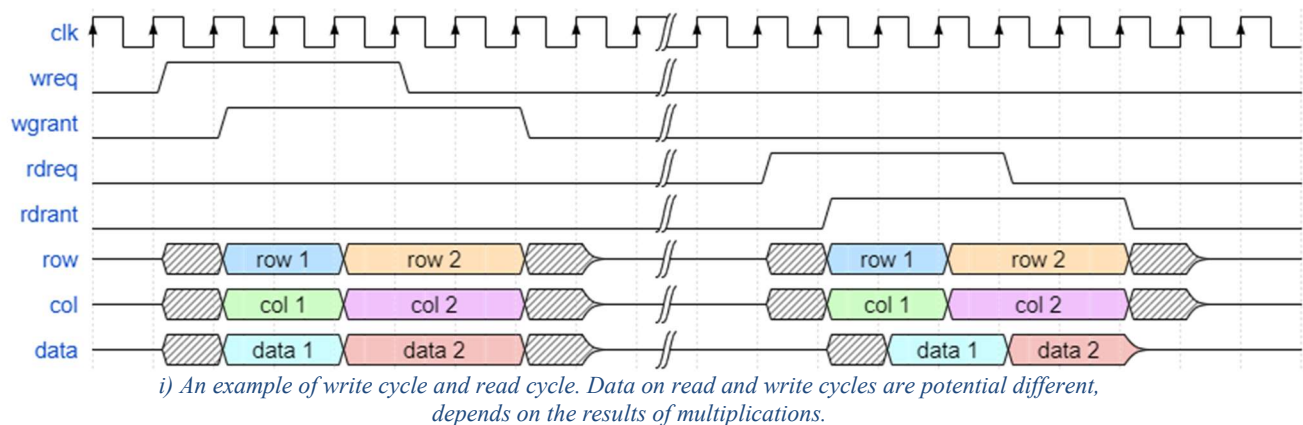

Device overview and I/O
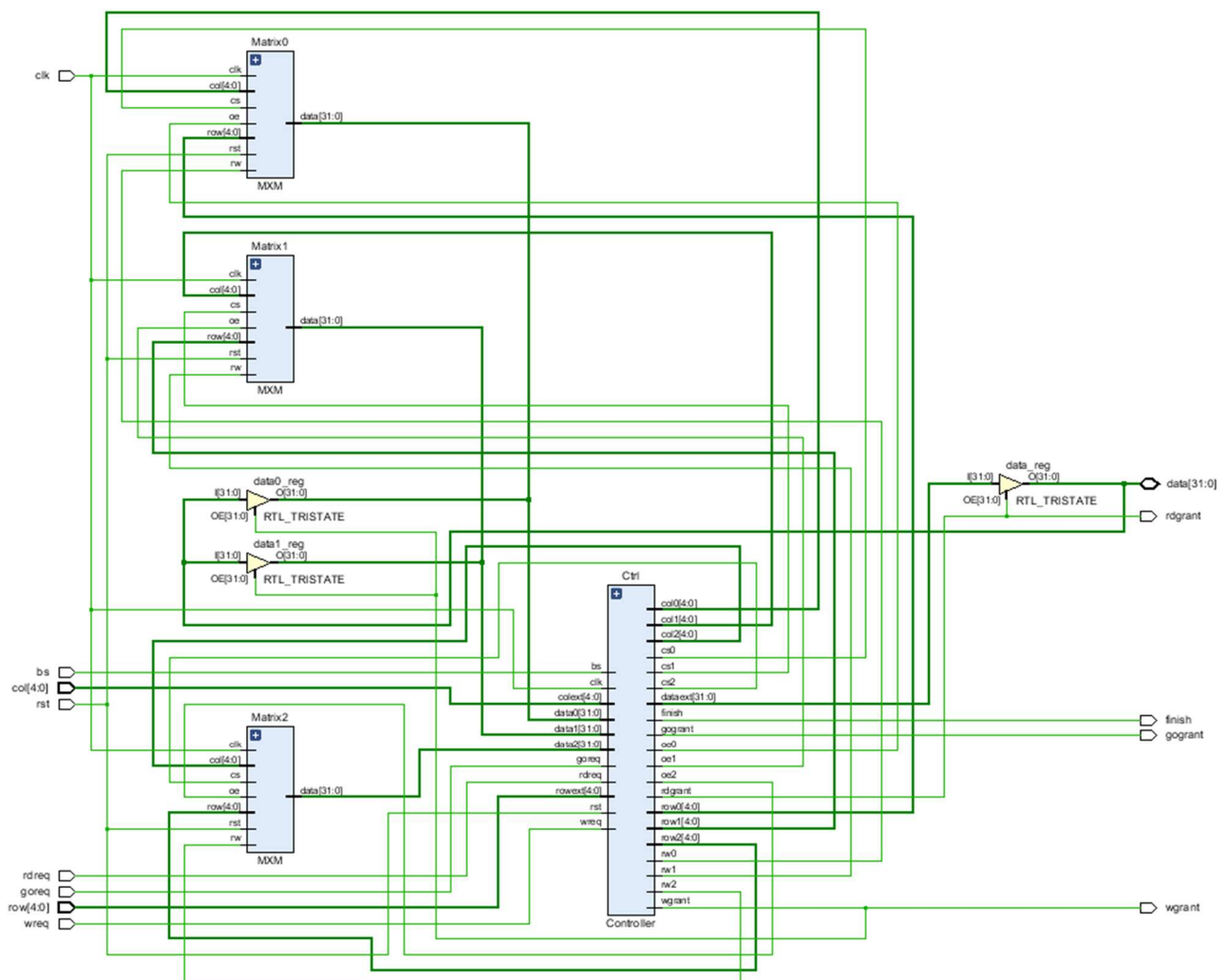
The internal structure of the device is pretty complicate but it is useful for highlighting some important features of the device.

The Input/Output list of the device is:

- Clk/Rst: for Clock input and Synchronous Reset

- Row/Column address: used to addressing rows and columns of matrices

- BS: means "Bank Select", it's useful during write operations to indicate what matrix the user wants to write data into. Bs=0 means matrix0 while Bs=1 means matrix1.

- Data: is the data Input/Output of the device. Here the user can feed the data to write or read from it.It's in high-z while computation and through the usage of some buffer, it's possible to connect it in a busy bus (driven by some other logics) without interfering with computation.

- Wreq/Wgrant: With Wreq high, the user can ask the device to enable the writing operation in the selected Bank. When Wgrant is enabled, the user can start to provide Row and Colum addresses together with data to write in that position. These three inputs must be provided at each two clock cycles and must stay stable for all this time for a correct latching of data.

- Rdreq/Rdgrant: With Rdreq high, the user can ask the device to enable the reading operation from the result matrix. When Rdgrant is high, user have to feed in row and column he wants to read then after 1 clock cycle the data will be available on "Data" output. This means that, as for the write operation, Row and Col shall stay stable for 2 clock cycles.

- Goreq/Gogrant: With Goreq high, the user can ask the device to start the computation. When it will start, Gogrant will be high until the end when the "Finish" signal will be set high for one clock cycle.

i) An example of write cycle and read cycle. Data on read and write cycles are potential different, depends on the results of multiplications.

Implementation of matrices

Matrices are simply implemented as a set of RAM blocks. In particular, we have 32 RAM blocks (for 32 columns) and each RAM block is made of 32 words made of 32 bit each as said.

The column address of the matrix, internally will redirect the Matrix's "Output Enable" and "Chip Select" input signals to the right RAM block through a multiplexer driven by the Column address.

The algorithm implemented by the Controller

The (simple) algorithm implemented by the controller in order to resolve the computation is made of a few steps. There are a lot of index to manage (two for matrix) so the final Finite State Machine (a Moore machine) implemented is quiet complex.
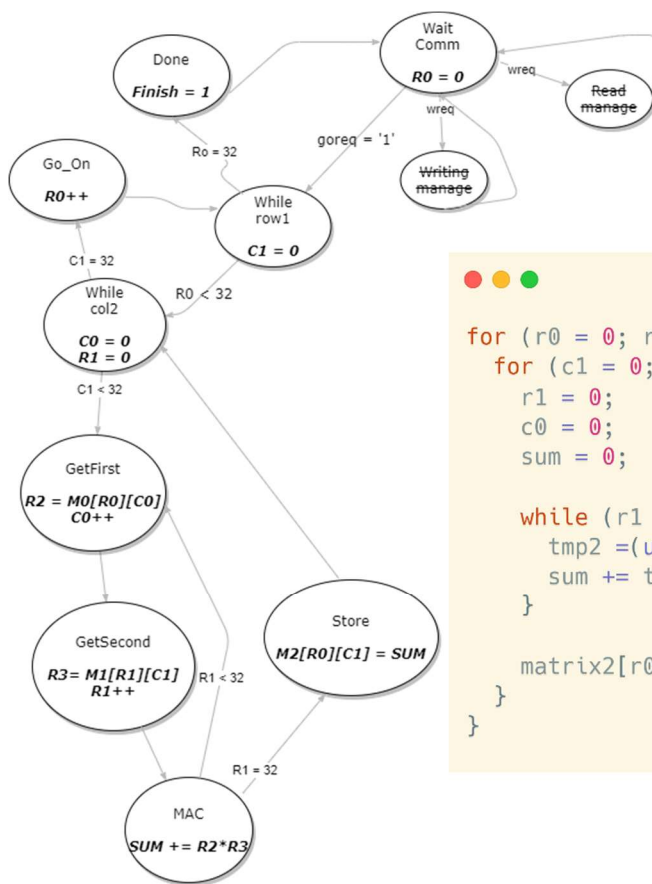

```c
for (r0 = 0; r0 < NROWS; r0++) {
  for (c1 = 0; c1 < NCOLS; c1++) {
    r1 = 0;
    c0 = 0;
    sum = 0;

    while (r1 < 32) {
      tmp2 =(uint64_t)matrix0[r0][c0++]*matrix1[r1++][c1];
      sum += tmp2;
    }

    matrix2[r0][c1] = (uint32_t)(sum >> FIXED_POINT);
  }
}
```

It's true that we want to make computations in fixed point, but the strength of this format is that it's easy to work with integer values. In fact, all the multiplications and sums are done treating all items as integers.

I want to underline how intermediate computations (multiply and sum) are done in 64 bits instead of 32 bits as original data. This is done in order to not lose information during processing. At the end, the final result will truncate the result to 32 bits.

Verification of the correctness

In order to test and simulate the design, a test bench is used. This test bench makes use of three different data file (mat0.mem, mat 1.mem, mat2.mem) that contains the values of the first two matrices and the expected result of the third matrix.

The test bench starts to load into the device under test the content of mem0 and mem1, then give order to starts the computation. As soon as the device is ready, the test bench will start to read from the device and will compare each result with the content inside mat2.mem. Some assertions will warn if there is something wrong with the results of the computation.

A C program is available to generate random values for the three files, that accepts as command line argument the path to store the generated files.

An overview inside the controller

The FSM seen before is implemented as an FSM with data path. The data path is shown below, and underline the most important aspects and control signals used by the FSM in order to control the correct path for data, from the input to the output.