Aaron Etheridge

Steve Furgurson

Graduates

4/25/19

ECGR 5146

Intro to VHDL

Systolic Array Final Project

**C block Timing**

(**aNij**\***bNji**)T  N = Clock cycle it appears

First RAM read T = Clock cycle the multiplication occurs

Second RAM read n - Cij  n = Anti-diagonal

Second RAM read with delay of one (stored in a register for one clock cycle)

**=>WB(X): Clock cycle for first write back. X = Delay in FIFO**

**=>WB(X): Clock cycle for second write back. X = Delay in FIFO**

1 - **C11** = (**a111**\***b111**)1 + (**a412**\***b521**)5+ (**a813\*b831**)8 + (**a1114**\***b1241**)12 + (**a1515**\***b1551**)15 **=>16(0)**

(**A1811**\***B1911**)19 + (**A2212**\***B2221**)22 + (**A2513**\***B2631**)26 + (**A2914**\***B2941**)29 + (**A3215**\***B3351**)33

**=>34(0)**

2 - **C21** = (**a221**\*b211)2 + (**a522**\*b621)6 + (**a923**\*b931)9 + (**a1224**\*b1341)13 + (**a1625**\*b1651)16 **=>17(0)**

(**A1921**\*B2011)20 + (**A2322**\*B2321)23 + (**A2623**\*B2731)27 + (**A3024**\*B3041)30 + (**A3325**\*B3451)34

**=>35(0)**

2 - **C12** = (a211\***b212**)3 + (a612\***b622**)6 + (a913\***b932**)9 + (a1314\***b1342**)13 + (a1615\***b1652**)16 **=>18(1)**

(A2011\***B2012**)20 + (A2312\***B2322**)23 + (A2713\***B2732**)27 + (A3014\***B3042**)30 + (A3415\***B3452**)34

**=>36(1)**

3 - **C31** = (**a331**\*b311)3 + (**a632**\*b721)7 + (**a1033**\*b1031)10 + (**a1334**\*b1441)14 + (**a1735**\*b1751)17 **=>19(1)**

(**A2031**\*B2111)21 + (**A2432**\*B2421)24 + (**A2733**\*B2831)28 + (**A3134**\*B3141)31 + (**A3435**\*B3551)35

**=>37(1)**

3 - **C22** = (a321\*b312)3 + (a622\*b722)7 + (a1023\*b1032)10 + (a1424\*b1442)14 + (a1725\*b1752)17 **=>20(2)**

(A2121\*B2112)21 + (A2422\*B2422)24 + (A2823\*B2832)28 + (A3124\*B3142)31 + (A3525\*B3552)35

**=>38(2)**

3 - **C13** = (a311\***b313**)3 + (a712\***b723**)7 + (a1013\***b1033**)10 +( a1414\***b1443**)14 + (a15\***b1753**)17 **=>21(3)**

(A2111\***B2113**)21 + (A2412\***B2423**)24 + (A2813\***B2833**)28 + (A3114\***B3143**)31 + (A3515\***B3553**)35

**=>39(3)**

4 - **C41** = (**a441**\*b411)4 + (**a742**\*b21)8 + (**a1143**\*b1131)11 + (**a1444**\*b1541)15 + (**a1845**\*b1851)18 **=>22(3)**

* (**A2141**\*B2211)22 + (**A2542**\*B2521)25 + (**A2843**\*B2931)29 + (**A3244**\*B3241)32 + (**A3545**\*B3651)36

**=>40(3)**

4 - **C32** = (a431\*b412)4 + (a32\*b22)8 + (a1133\*b1132)11 + (a1534\*b1542)15 + (a1835\*b1852)18 **=>23(4)**

* (A2231\*B2212)22 + (A2532\*B2522)25 + (A2933\*B2932)29 + (A3234\*B3242)32 + (A3635\*B3652)36

**=>41(4)**

4 - **C23** = (a421\*b413)4 + (a22\*b23)8 +( a1123\*b1133)11 + (a1524\*b1543)15 + (a1825\*b1853)18 **=>24(5)**

* (A2221\*B2213)22 + (A2522\*B2523)25 + (A2923\*B2933)29 + (A3224\*B3243)32 + (A3625\*B3653)36

**=>42(5)**

5 - **C42** = (a541\*b512)5 + (a942\*b922)9 + (a1243\*b1232)12 + (a1644\*b1642)16 + (a1945\*b1952)19

**=>25(5)**

* (A2341\*B2312)23 + (A2642\*B522)26 + (A3043\*B3032)30 + (A3344\*B342)33 + (A3745\*B3752)37

**=>43(5)**

5 - **C33** = (a531\*b513)5 + (a932\*b923)9 + (a1233\*b1233)12 + (a1634\*b1643)16 + (a1935\*b1953)19

**=>26(6)**

* (A2331\*B2313)23 + (A2632\*B2623)26 + (A3033\*B3033)30 + (A3334\*B3343)33 + (A3735\*B3753)37

**=>44(6)**

6 - **C43** = (a641\*b613)6 + (a1042\*b1023)10 + (a1343\*b1333)13 +(a1744\*b1743)17 + (a2045\*b2053)20

**=>27(6)**

* (A2441\*B2413)24 + (A2742\*B2723)27 + (A3143\*B3133)31 + (A3444\*B3443)34 + (A3845\*B3853)38

**=>45(6)**

This reads in one ‘a’ element and one ‘b’ element per clock cycle. Since there are more rows than columns, every 4th cloth cycle there will be an ‘a’ element passed into a register to introduce a stall for one clock cycle. There will then be a delay in the next three clock cycles with four delays in total (4,5,6,7). For simplicity, only the ‘a’ elements will be stored in registers for delays since there are more ‘a’ rows than ‘b’ columns.

Clock cycles with stalls:

4,5,6,7

11,12,13,14

18,19,20,21

25,26,27,28

32,33,34,35



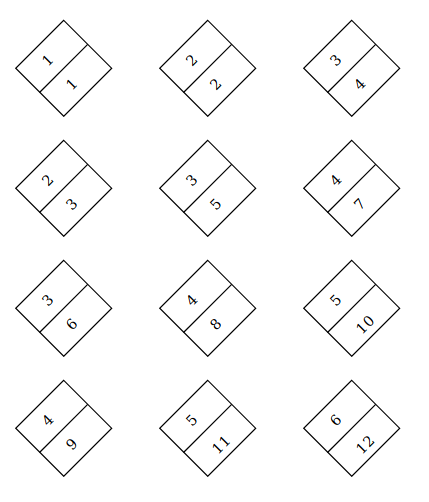
****

Figure 1: Cycle Completed vs Cycle Written to RAM

The top number is not actually the cycle the block is computed, just a representation of the order they are computed. By subtracting the bottom number from the top number, results in the delays that need to be applied to the output of each C block. For example C11 doesn’t need a delay because it is computed and can be sent back in the cycle immediately after. But for C43 the partial sum will be calculated in the 6th cycle. But will not be able to be written back till the 12 cycle because other blocks preceding it hasn't been written back to RAM.

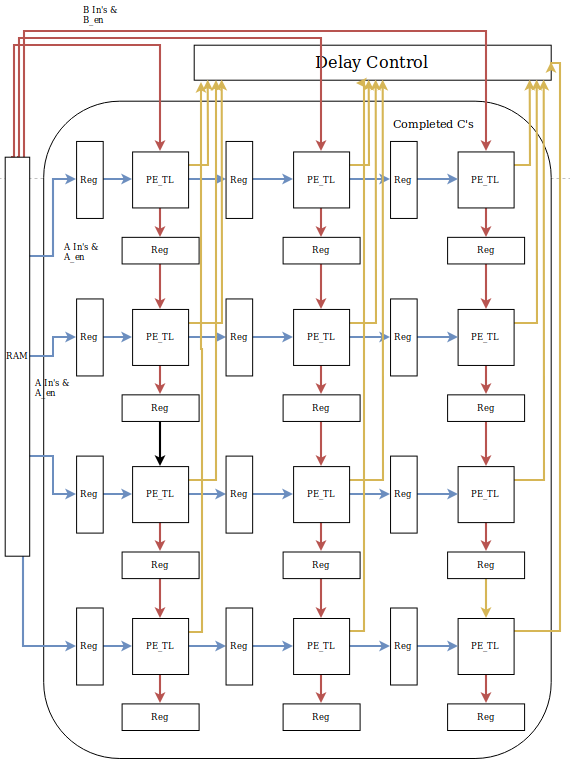


Figure 2: MAC2Reg Array data Flow

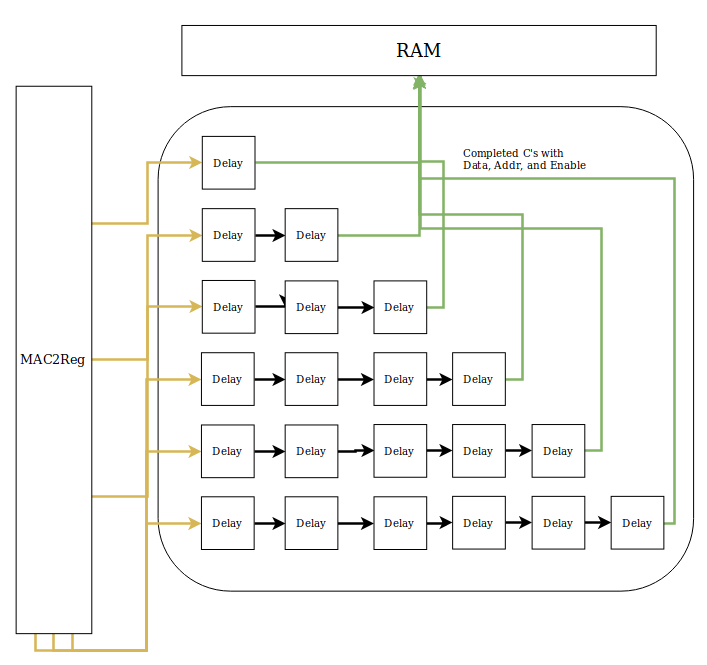


Figure 3: Delay Control

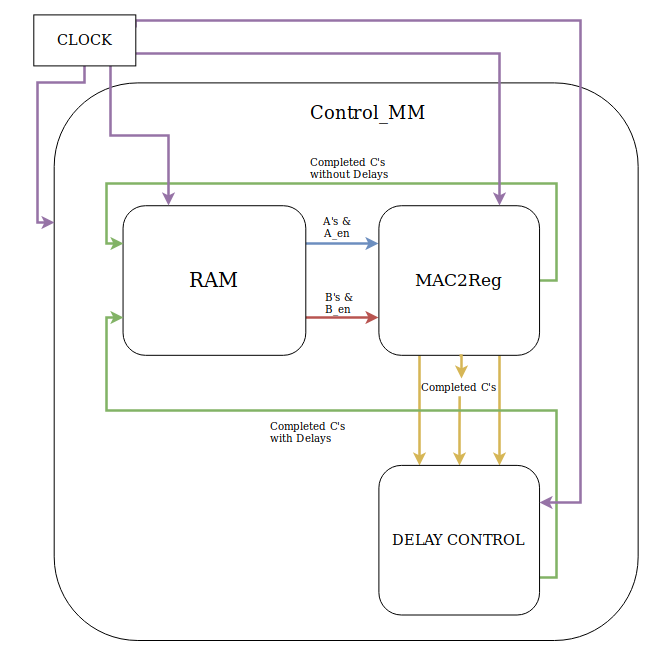


Figure 3: Overview

**PE\_TL Blocks in MAC2Reg**

The PE\_TL block is the core compute unit that does the multiplication and summation. It takes in two input data signals and two input enable signals. When both the a and b enable signals are high, the data for a and b are both ready for calculation. Once the calculation is complete a counter increments by one. Once the counter reaches 5, the partial sum is completed and sent out of the MAC2Reg component with the C\_data and a C\_en. The number 5 is specific to this matrix layout because each partial sum will consist of 5 sums. MAC2Reg then port maps each individual C block to the delayControl block. After the 5th compute the partial sum is reset to 0 at the beginning of the next clock cycle.

**Reg Blocks in MAC2Reg**

The reg (register) blocks simply delay the data and enable for a clock cycle. This is critical in our implementation of the systolic array. When data moves to a C block it moves inside the C block as well as the register after the C block so that it is ready for computation by the proceeding C block.

**RAM**

Once all the clock cycles were figured out for each C block, as they were listed above; the ram could be developed. The A matrix was transposed so that it could be easily incremented through with a simple counter. By transposing the A matrix, the incrementing array order matched order that data needed to be sent to the c blocks. The B matrix didn’t have to be transposed because it was already stored in the correct order for how the MAC2Reg processed the data.

A process that was triggered by the clk counted the number of clocks that had occured. This count was modulused with 4. The resulting modulus broke the feeding process into 4 parts. If the remainder was 0, the data would be sent to the A0 row, by sending an [A\_en =”1000”] signal. The B0 data would also be sent using an [B\_en =”100”] signal. If the remainder was 1 the “0100” and “010” signals would be sent to A\_en and B\_en respectively. If the remainder of the modulus was 3 then only the A\_en=”0001” was sent and B would get nothing. Every time an enable was sent to either a or b an iterator would add one moving the iterators through both the A and B matrices. Both of the A and B matrices were held in the same array or RAM block so an offset of 40 was the starting point from which B would be iterated through.

**DelayController**

Once the partial sums are completed inside the MAC2Reg, they are sent to the delay controller component. Each C block has its own input, depending on what C block the data is coming from, the data is paired with an address that is used to write back the data to memory. Inside the delay controller, every input from each individual C block is mapped to a specific amount of delays. These delays were calculated using figure 1. Once the C’s flow through their appropriate delays, they are sent back with an address to RAM.

Control\_MM

Control\_MM is the top file that contains the MAC2Reg, the Delay Controller, and the RAM with everything port mapped. The only thing that feeds into this is the clock which runs everything.

Timing Diagram

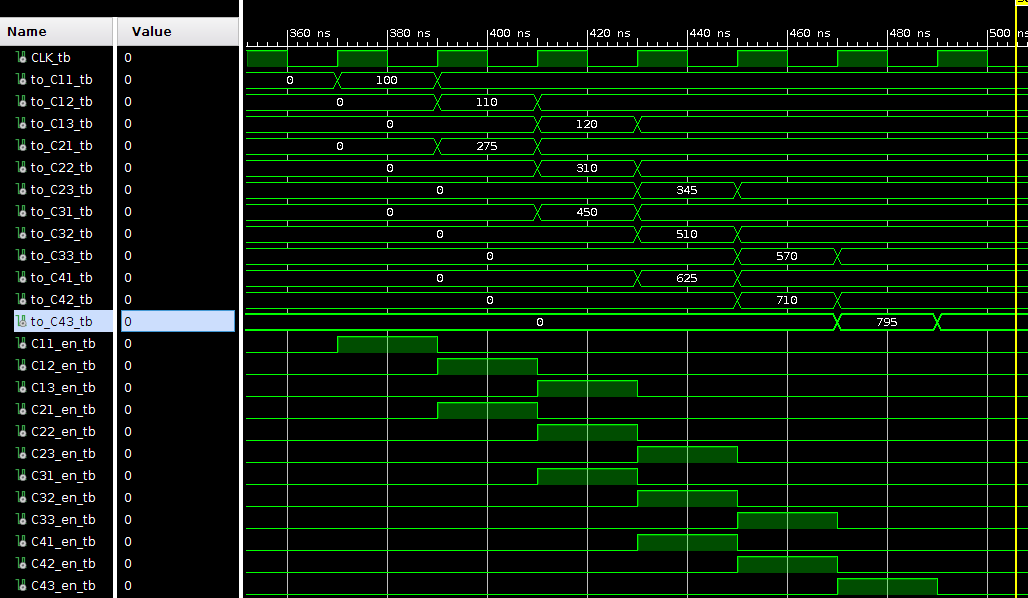


Figure 5: Top half of Timing Diagram

to\_C11\_tb through to\_C43\_tb are the completed partial sums as well as demonstrating which order they complete. In certain cycles, 2 or 3 complete in the same clock cycle. The C\_XX\_en\_tb’s tie to when the partial sums are complete. In the cycles between the partial sums are set to zero to make sure values are not being carried from one output to another.

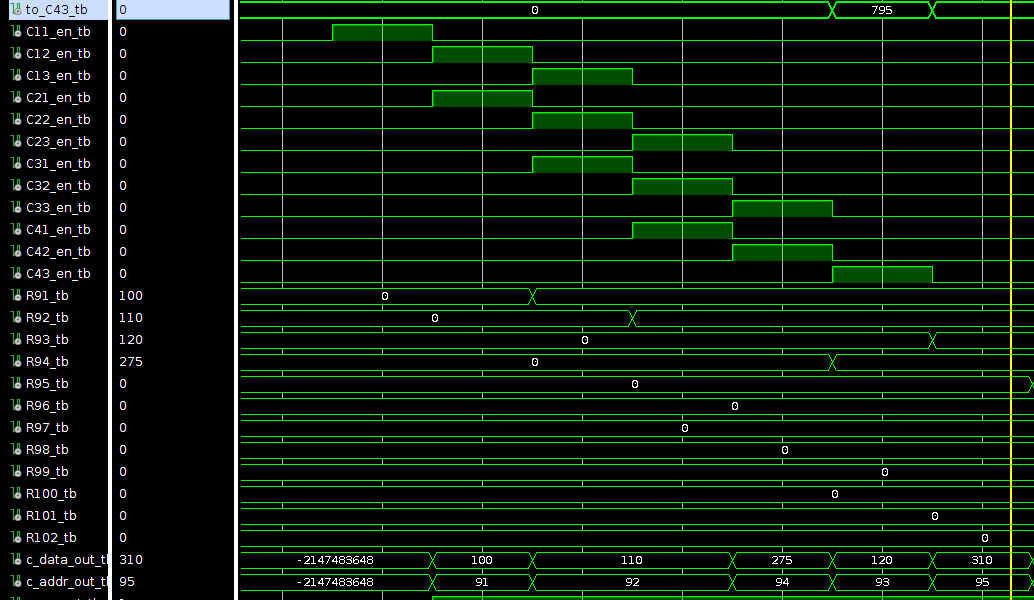


Figure 6: Bottom Half of Timing Diagram

As before, the CXX\_en\_tb’s represent when the partial sum blocks are completed. C\_data\_out\_tb is the data that is being written out in a particular clock cycle. The number below is the address that the data is being written to in the ram array. The delayControl unit controls when and where the resulting partial sums are sent to. For all 12 C’s are assigned their own write back address in the RAM array.

The C’s are all addressed between 91 and 102. In our implementation of the systolic array, the C13 block is represented by the array address 93 because the array is ordered from top left to bottom right. However, the C21 (addr(94)) completes its calculation on the second clock cycle, while the C13 isn’t calculated till that is why the array addresses are out of order. A similar pattern of array addresses persists throughout the program because of diagonal lines of computation.

**Conclusion**

In this project we designed and implemented a systolic array for computing matrix multiplication. We explored and noted the advantages (simple design, high concurrency operations) and the disadvantages (specialized and not generally applicable because computation needs to fit the PE organization). Simulating the two read accesses and one write access into the RAM block also enlightened us to the bottleneck between memory access and execution. In the end we felt this project not only progressed our knowledge of VHDL but also gave us a better understanding of designing digital circuits.