2D Weight-Stationary Systolic Architecture for Matrix Multiplication

Abstract— **In this document, a 2D weight stationary systolic architecture for matrix multiplication is to be designed using system C.**

Keywords— **Systolic architecture, Processing element, Datapath, Controller, Clock cycle, BFM, Test bench, Shift-register,**

I. Introduction

In this computer assignment, a processing element should be designed, the operation of which is MAC (multiply and accumulate). Next, a 2D matrix multiplier should be implemented using these processing elements. After that, a BFM (bus functional model) of this multiplier should be developed in System C.

II. Processing Element

As explained in the homework PDF, the processing element consists of two 8-bit inputs named Di and Wi, a 24-bit input Si, and two control signals: StartPE and BusyPE. Wi and Di should be latched and multiplied together. The result of the multiplication and Si should then be accumulated and registered.

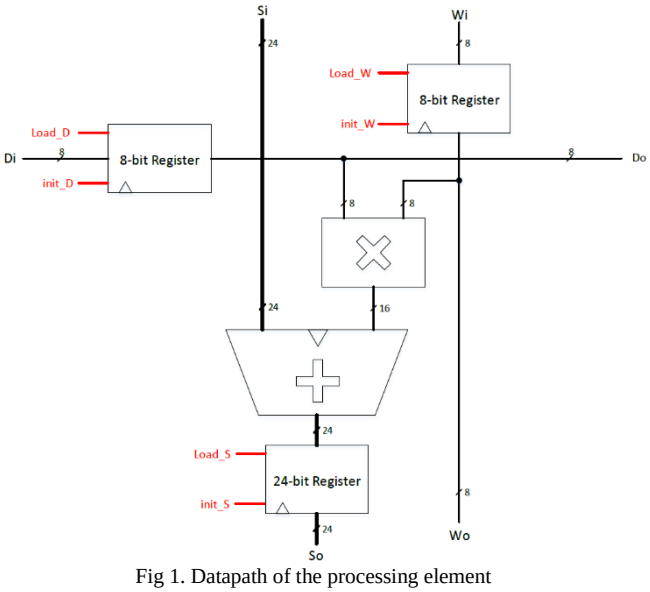

Fig 1. Datapath of the processing element

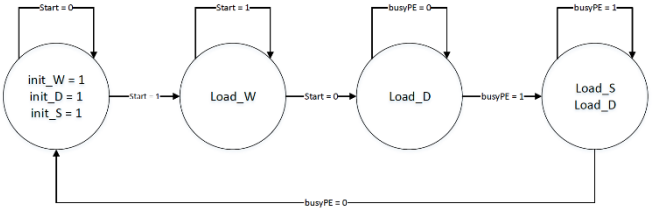

Fig 2. The processing element controller

As shown in Fig 1, there are three registers in the datapath to latch Di, Wi, and So. The controller, shown in Fig 2, waits for the start signal to become 1. After receiving a 1 on the start signal, the controller asserts the load signals for the W-register to latch Wi from its bus. Meanwhile, the controller waits for the start signal to become 0. Once the start signal becomes zero, the controller asserts the "load_D" signal to register Di's data from its bus. Whenever the "busyPE" signal is 0, the Di data is registered using an 8-bit register. When the busyPE becomes 1, the adder and multiplier in the datapath start to calculate the MAC operation, and the controller puts a 1 on the "load_S" and "load_D" signal to register the MAC output and receive new data in its Di input.

As explained in previous paragraph, if the start signal becomes 1 and remains 1 for a few consecutive clock cycles, the processing element register data from Wi bus in these clock cycles. This option can be used to load weight-values into the top processing elements and propagate value to beneath ones. This option is exactly implemented in horizontal way for Di and busyPE signal.

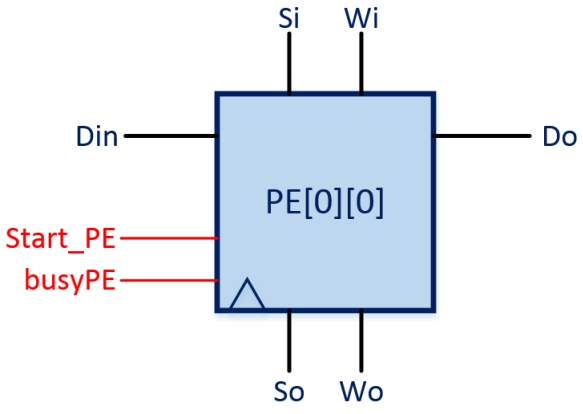

Fig 3. The processing element block

For verify this circuit, a test bench is to be developed is system C so different inputs in different clock cycle is passed to the Processing element circuit and the outputs can be verified.

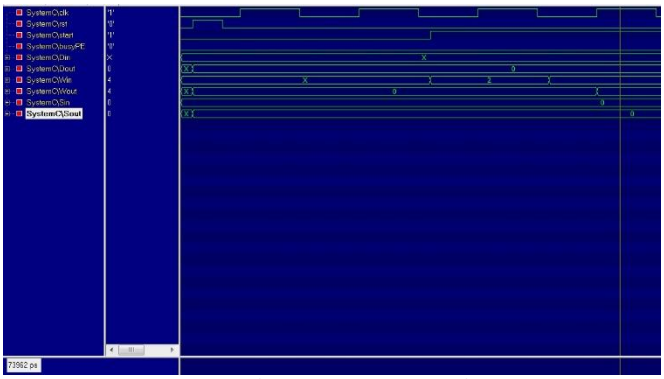

Fig 4. One of the PE-testbench waveform

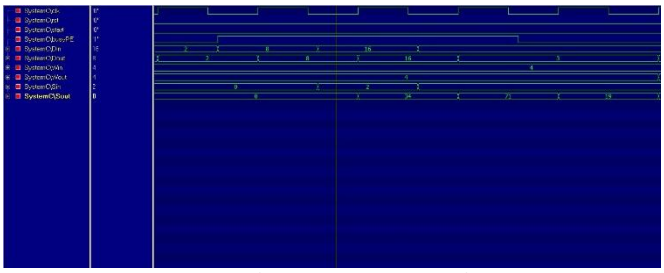

Fig 5. One of the PE-testbench waveform

III. 2D Weight Stationary Matrix Multiplier

A. RTL

Figure 6 shows a 3x3 matrix multiplier datapath. However, for the implementation to be dynamic in SystemC, the "template" syntax should be utilized. For instance, if a 4x4 matrix multiplier is instantiated in the testbench, the code should instantiate 16 processing elements.

Figures 6 and 7 show that the controller is waiting for the start signal to become 1. Once the start signal is received, the controller asserts all register load signals to initialize their values. After initializing the registers, the W-matrix values should be loaded into the processing elements. To achieve this, the controller puts 1 on all W-register shift-enable signals and on all processing element start signals. The controller should remain in this state for "mult_size - 2" clock cycles, which requires a counter to be implemented. Once the loading of W-matrix values is complete, the controller asserts the shift-enable signals for all R and D registers and puts 1 on all processing element busy signals. The controller should stay in this state for "3*mult_size - 1" clock cycles. In the next state, another counter is needed to count the number of iterations. When the circuit is instantiated in a testbench, it can be used not only once but also for consecutive times. Therefore, the state machine should check the number of iterations at the end of every iteration to determine whether the process is completed or not.

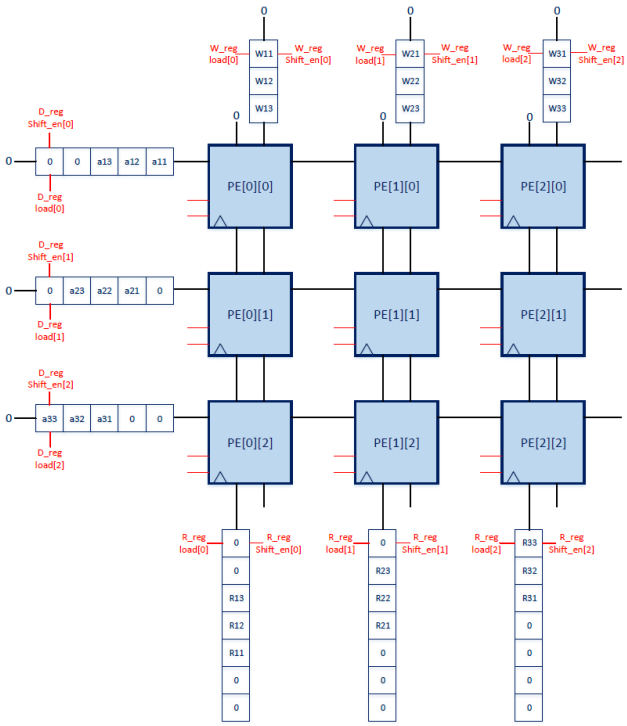

Fig 6. Image of the 3x3 matrix multiplier datapath

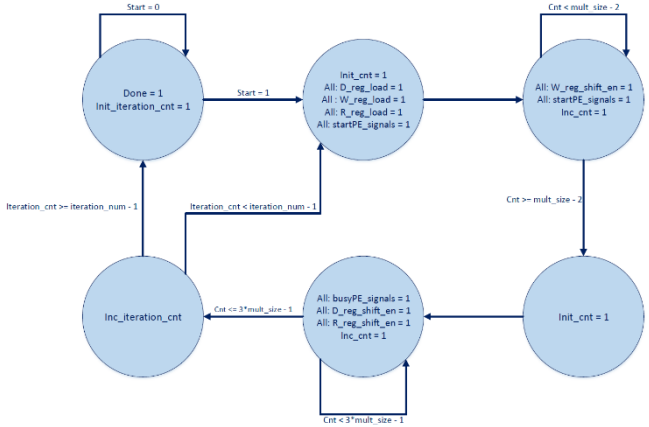

Fig 7. Image of the matrix multiplier Controller

B. BFM

The bus functional model (BFM) is a way of implementing a system that does not work exactly like the RTL design but can mimic it. For example, in the matrix multiplier, the RTL design requires several clock cycles to make the output ready. However, the BFM design multiplies two matrices in three nested loops in just one clock cycle but waits for a few clock cycles before asserting the outputs. While the function of both implementations is the same, at a closer look, the BFM is essentially imitating the operation of the real design.

IV. Output Justification

There are currently two implementations of the matrix multiplier, each of which requires the development of three different testbenches. The first testbench should provide the circuit with two matrices as inputs, where the size of each

matrix is the same as the multiplier size. In the second testbench, the size of the input matrices should be smaller than the multiplier size, while in the last testbench, the matrix sizes should be larger than the multiplier size.

A. *Test 1(same size)*

In first test two matrix 3x3 is passed to a 3x3 matrix multiplier for both RTL and BFM implementation. The matrices are:

$$D = \begin{bmatrix} 4 & 16 & 2 \\ 2 & 12 & 1 \\ 8 & 10 & 0 \end{bmatrix}, \quad W = \begin{bmatrix} 5 & 3 & 2 \\ 1 & 8 & 2 \\ 4 & 16 & 4 \end{bmatrix}$$

$$D \times W = \begin{bmatrix} 4 & 16 & 2 \\ 2 & 12 & 1 \\ 8 & 10 & 0 \end{bmatrix} \times \begin{bmatrix} 5 & 3 & 2 \\ 1 & 8 & 2 \\ 4 & 16 & 4 \end{bmatrix} = \begin{bmatrix} 44 & 172 & 48 \\ 26 & 118 & 32 \\ 50 & 104 & 36 \end{bmatrix}$$

Figures 8 and 9 demonstrate that the final outputs of both circuits are identical to the expected outputs. Furthermore, the done signal in each implementation becomes active after 310 ns, indicating that the BFM matrix multiplier can accurately mimic clock cycles. However, it's important to note that the BFM matrix multiplier output remains unchanged until the 'done' signal becomes active, while the RTL outputs change during the process before the done signal becomes active.

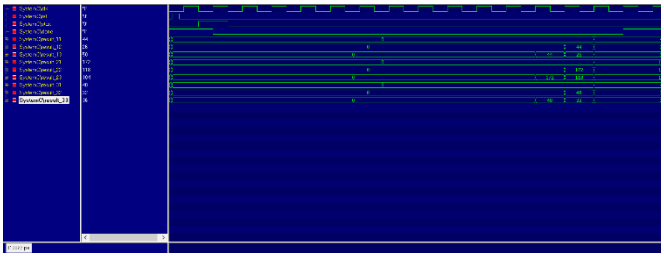

Fig 8. The outputs waveform of the RTL matrix multiplier

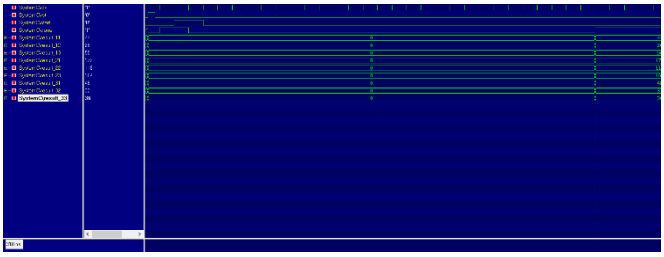

Fig 9. The outputs waveform of the BFM matrix multiplier

B. *Test 2(smaller size)*

In second test two 2x2 matrices is passed to 3x3 matrix multiplier for both RTL and BFM implementation. These two matrices are:

$$D = \begin{bmatrix} 8 & 33 \\ 6 & 21 \end{bmatrix}, W = \begin{bmatrix} 5 & 13 \\ 11 & 18 \end{bmatrix}$$

$$D \times W = \begin{bmatrix} 8 & 33 \\ 6 & 21 \end{bmatrix} \times \begin{bmatrix} 5 & 13 \\ 11 & 18 \end{bmatrix} = \begin{bmatrix} 403 & 698 \\ 261 & 456 \end{bmatrix}$$

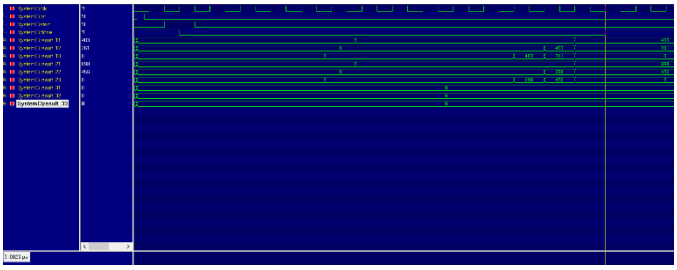

Fig 10. The outputs waveform of the RTL matrix multiplier

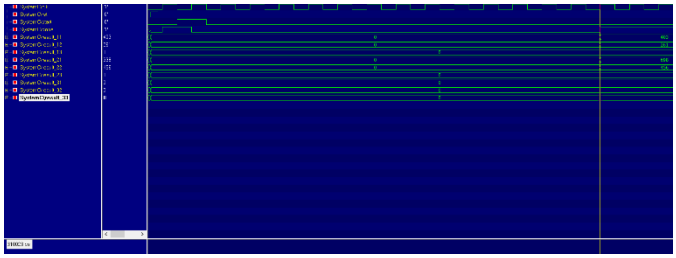

Fig 11. The outputs waveform of the BFM matrix multiplier

Figures 10 and 11 demonstrate that a matrix multiplier can be used to multiply two matrices with smaller sizes. However, it is necessary to insert 0s in every other element of the input matrices to ensure proper calculation.

C. *Test 3(larger size):*

In the last test, a multiplication between a 2x4 matrix and a 2x2 matrix it's needed to be done using a 2x2 matrix multiplier. To achieve this, we can divide the 2x4 matrix into two 2x2 matrices and pass them to the multiplier in two consecutive steps. First, the multiplier accepts the first matrix and multiplies it with the weight matrix, producing the first output. At this stage, the testbench saves this output and passes the second matrix to the multiplier. Once the output is ready, the results can be obtained by combining the two output matrices.". The matrices that are used in the testbench are:

$$D = \begin{bmatrix} 4 & 10 \\ 7 & 13 \\ 11 & 12 \\ 8 & 5 \end{bmatrix}, W = \begin{bmatrix} 5 & 3 \\ 6 & 15 \end{bmatrix}$$

$$D \times W = \begin{bmatrix} 4 & 10 \\ 7 & 13 \\ 11 & 12 \\ 8 & 5 \end{bmatrix} \times \begin{bmatrix} 5 & 3 \\ 6 & 15 \end{bmatrix} = \begin{bmatrix} 80 & 162 \\ 113 & 216 \\ 127 & 213 \\ 70 & 99 \end{bmatrix}$$

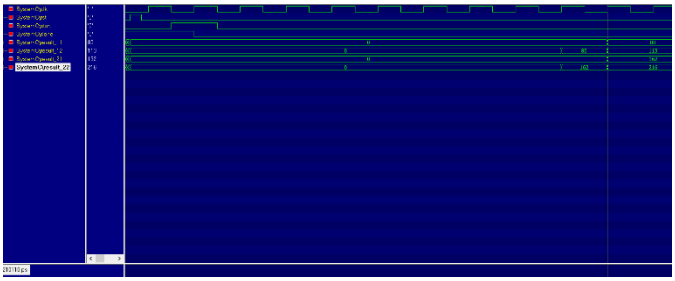

Fig 12. The first output of the RTL matrix multiplier

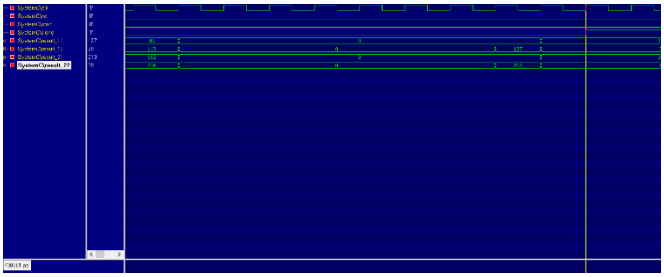

Fig 13. The second output of the RTL matrix multiplier

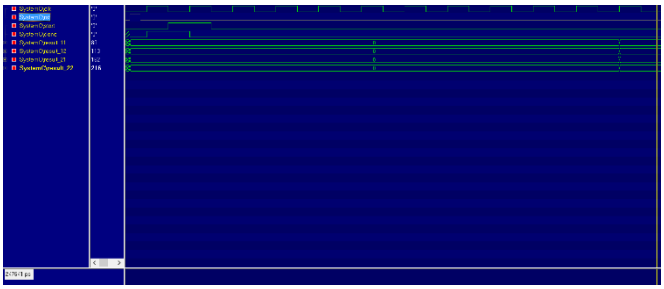

Fig 14. The first output of the BFM matrix multiplier

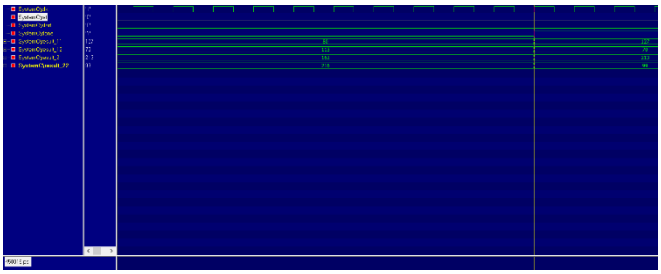

Fig 15. The second output of the BFM matrix multiplier

```
result BFM test1:
44 , 172 , 48
26 , 118 , 32
50 , 104 , 36
result BFM test2 :
403 , 698 , 0
261 , 456 , 0
0 , 0 , 0
result RTL test 2:
403 , 698 , 0
261 , 456 , 0
0 , 0 , 0
result RTL test 1:
44 , 172 , 48
26 , 118 , 32
50 , 104 , 36
result RTL test 3:
80 , 162
113 , 216
127 , 213
70 , 99
result BFM test3 :
80 , 162
113 , 216
127 , 213
70 , 99
```

Fig 16. Whole testbenches outputs

As shown in figure 12 to 15, one matrix multiplier can be used in consecutive step to calculate a larger matrices multiplication. figure 14 and 15 shows that the BFM can calculate the output precisely and it's clock accurate enough.

V. Conclusions

In this computer assignment, the systolic array matrix multiplier was designed and we learn how it can be used in different ways to calculate matrix multiplying. And the most important thing in this CA is how we can use BFM to create a component without know anything about its hardware implementation and just for using its function in the system-level design.