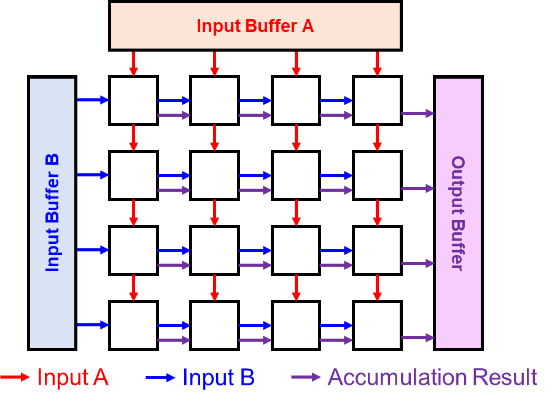
**ELEC 522 Assignment 2**

Student: Sixu Li (S01435077)

# I. Design Motivation & Architecture

The task of this assignment is to design a systolic array that can perform matrix – matrix multiplication (MMM) & matrix – vector multiplication (MVM). Since matrix – matrix multiplication is a series of matrix – vector multiplication. So we can actually use the same control & data flow for MMM & MVM. When executing MVM, zero-padding will be adopted to fit the MMM flow.

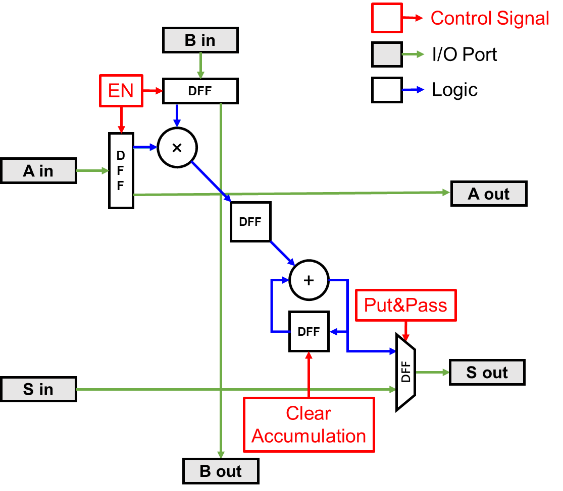
Following the above analysis, a vanilla systolic array can handle the required task.



The above figure showed the overall architecture of the vanilla systolic array. The input buffer will feed the data to the array from two direction and the output data will be collected in one other direction. The PE array follows a mesh connection, with each PE only communicate with its neighboring PEs. Using the above architecture, each node in the PE array can receive 2 complete vectors to perform vector – vector inner product. So that using the entire 2D PE array, MMM can be calculated. Using one row / column, MVM can be calculated.

# II. Processing Element Design

To perform Vector – Vector inner product, each PE need to multiply the vectors in element-wise mode. Then accumulation need to be performed to get the result. Because the PE array need to pass the result to the output buffer in a systolic style. An output register is also needed.



The above figure showed the detailed architecture of each PE, the input will first pass two gated DFF to be multiplied, then pass a DFF for accumulation. The adder performs the function of accumulation, the inputs of the adder are the multiplication result and the adder output. The output of the adder is connected to a DFF with reset so that the accumulation result can be cleared between multiple jobs.

# III. Design Implementation in Model Composer

The above design is implemented in Model Composer.

图示

描述已自动生成图示

描述已自动生成

The above two figures showed the test setup and one detailed PE.

图示

描述已自动生成

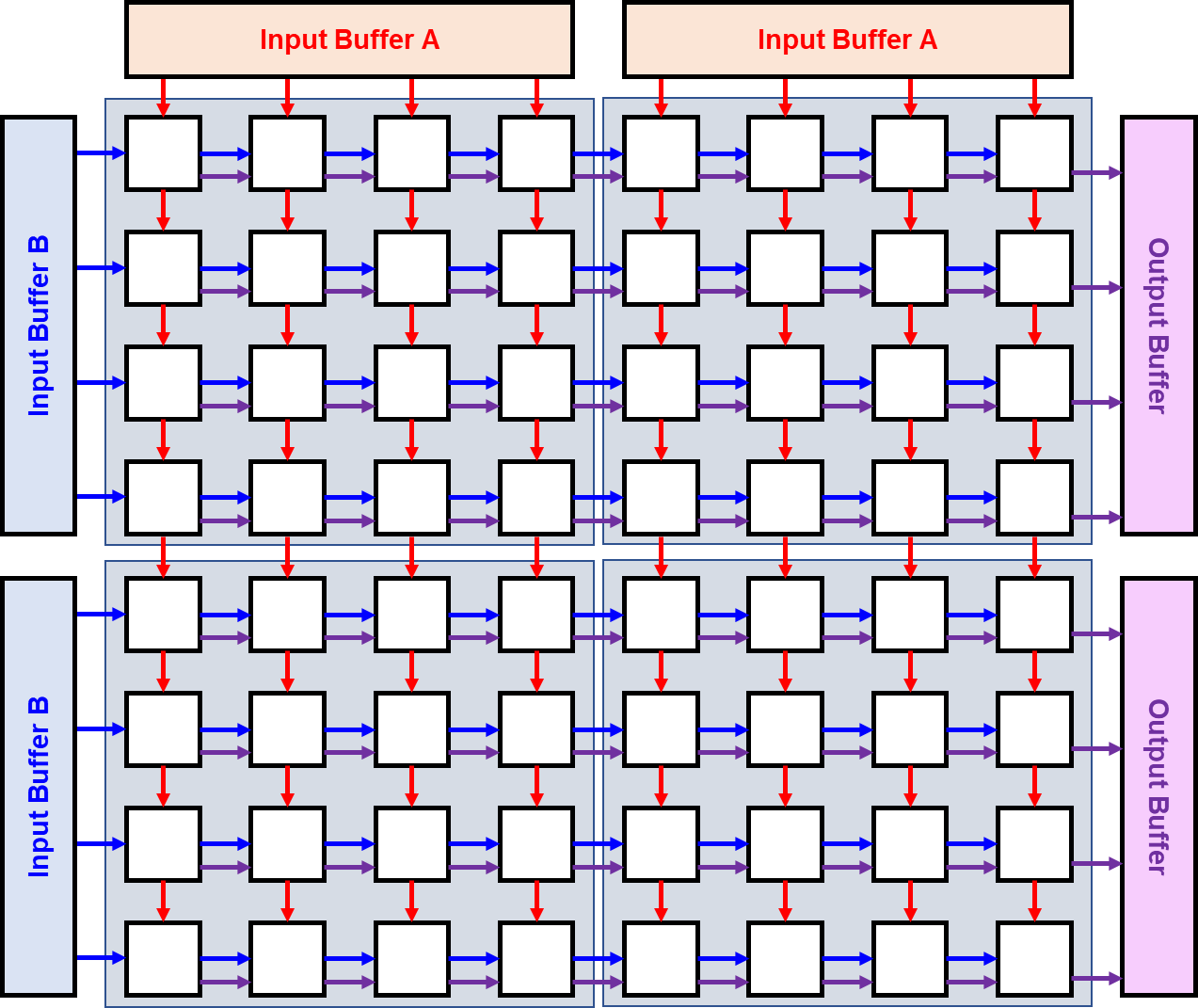
The above figure showed the interconnect between the PEs. The four control signal is connected to every PE while the data signal only exists among the neighboring PEs.

# IV. Evaluation

## 4.1 Delay & Scalability Analysis

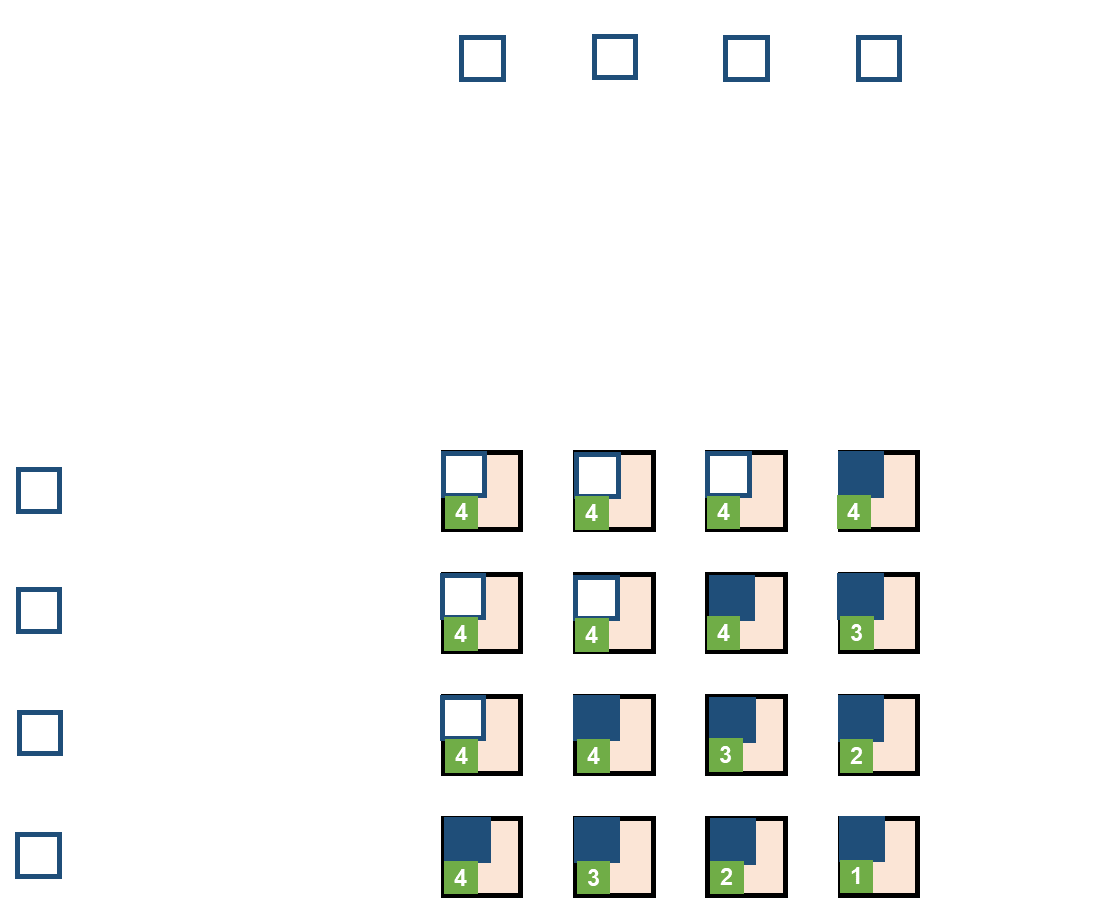
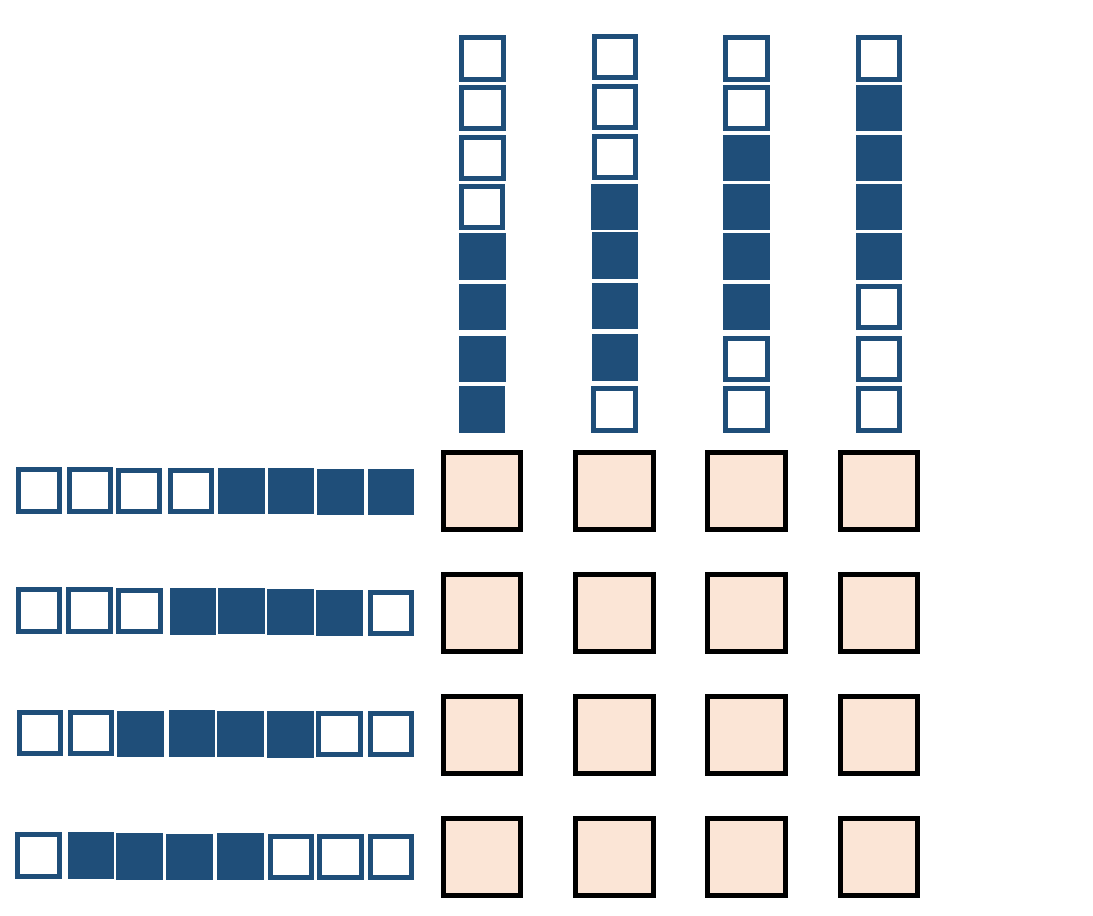
### 4.1.1 Scalability Analysis

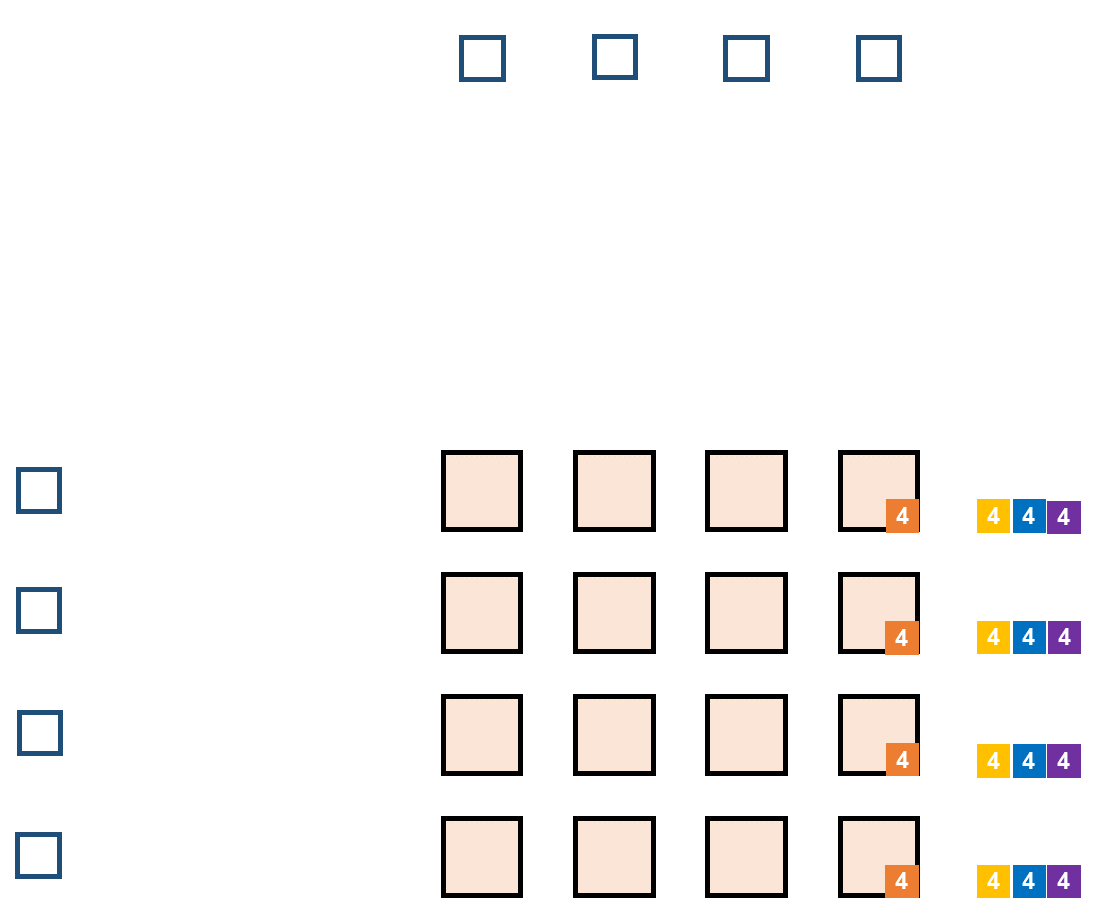
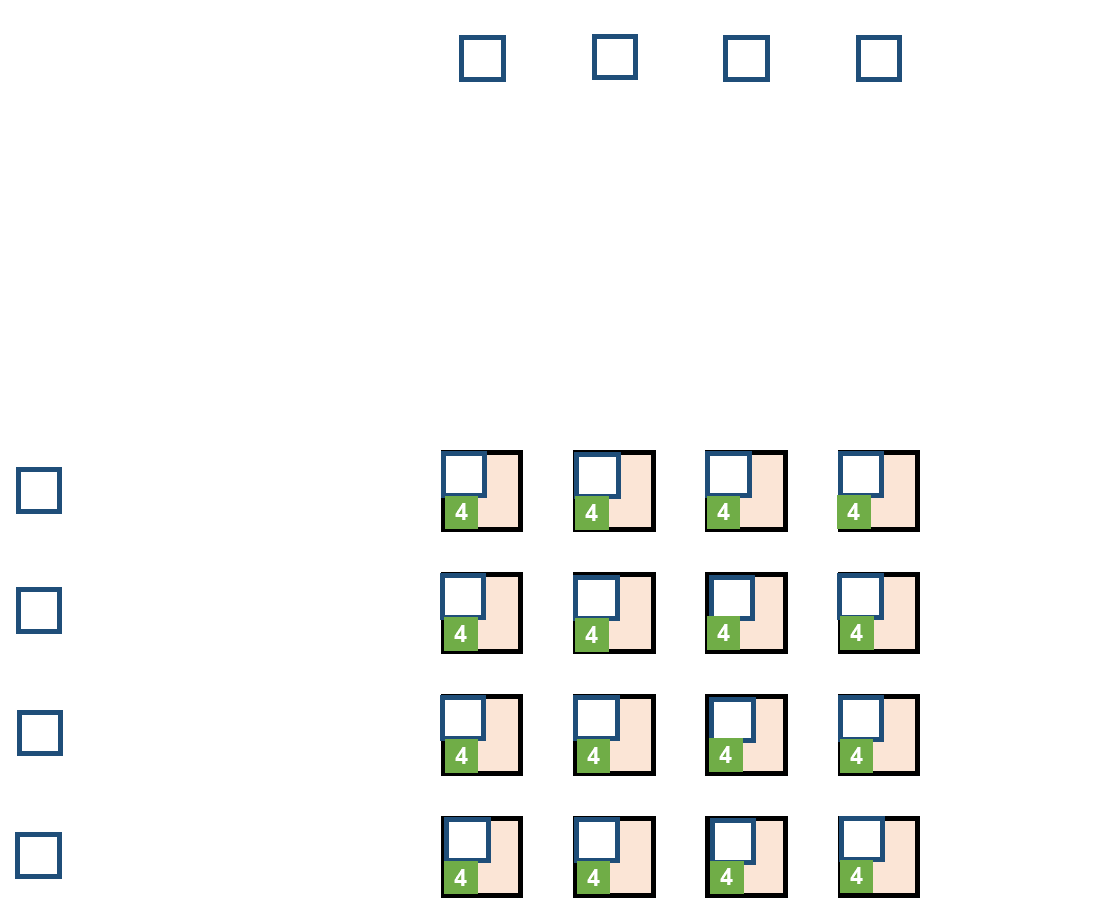
The proposed systolic array can be directly connected to form a large systolic array. The below figure showed an example of the directly connected PE cluster with 4 PE Array.



### 4.1.2 Delay Analysis

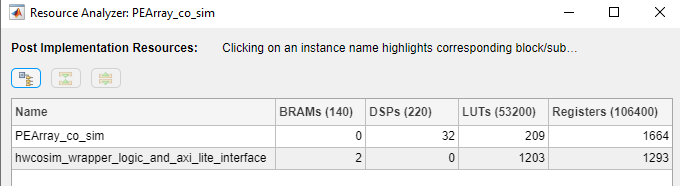
There are serval stages for the PE array to perform the Matrix-Matrix Multiplication. 1. Prepare stage, the data are in the input buffer. 2. Filling array, because of the systolic dataflow, N + 3 clock cycles are needed before all the PEs received the first input pair. 3. Wait propagation, because the PE has a size of 4x4, the last input need 3 clock cycles to arrive the most remote PE. 4. Wait computation, for better timing, there are some registers inside the PE, serval clocks are need for the PE to finish its computation. 5. Put & Pass accumulation, after all the computation is completed, the sum result will be put on a register and passed to the output buffer. The flowing 4 figures give an illustration for the stages 1,2,3,5.

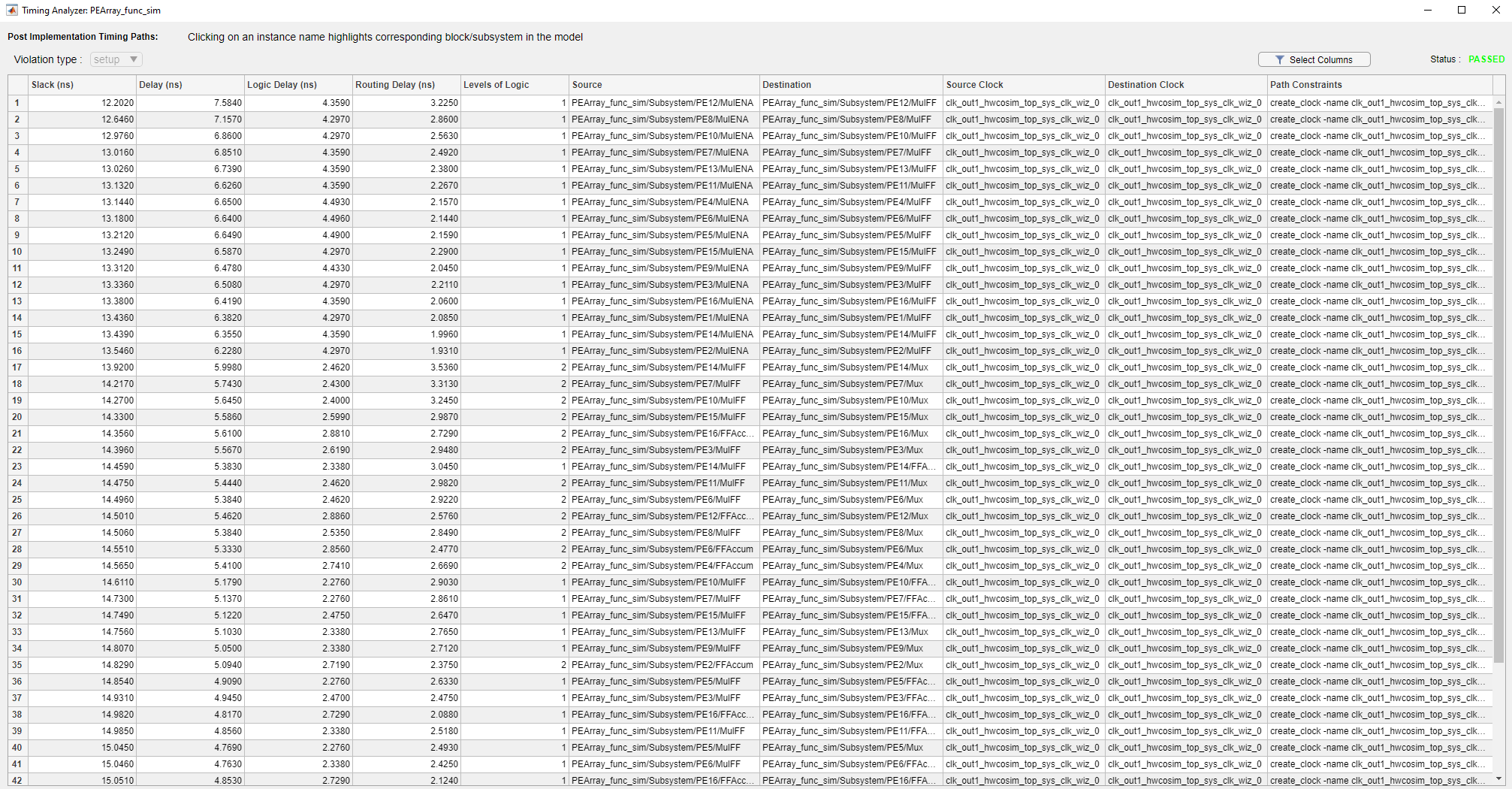




## 4.2 Performance & Area

The following two figures showed the reported area consumption and the timing report.





## 4.3 Functional & HW-CO simulation Results

To evaluate the correctness of the proposed systolic array. A test platform is build with matlab script. The detailed code is attached. The script will first generate random numbers, then create the input variables in the workspace. After all the inputs are generated, Simulink will be called to perform the simulation. The result will be outputted to the workspace. The script will read the result and calculate the relative error between the systolic array output and the matlab result.

### 4.3.1 Functional simulation

文本, 聊天或短信

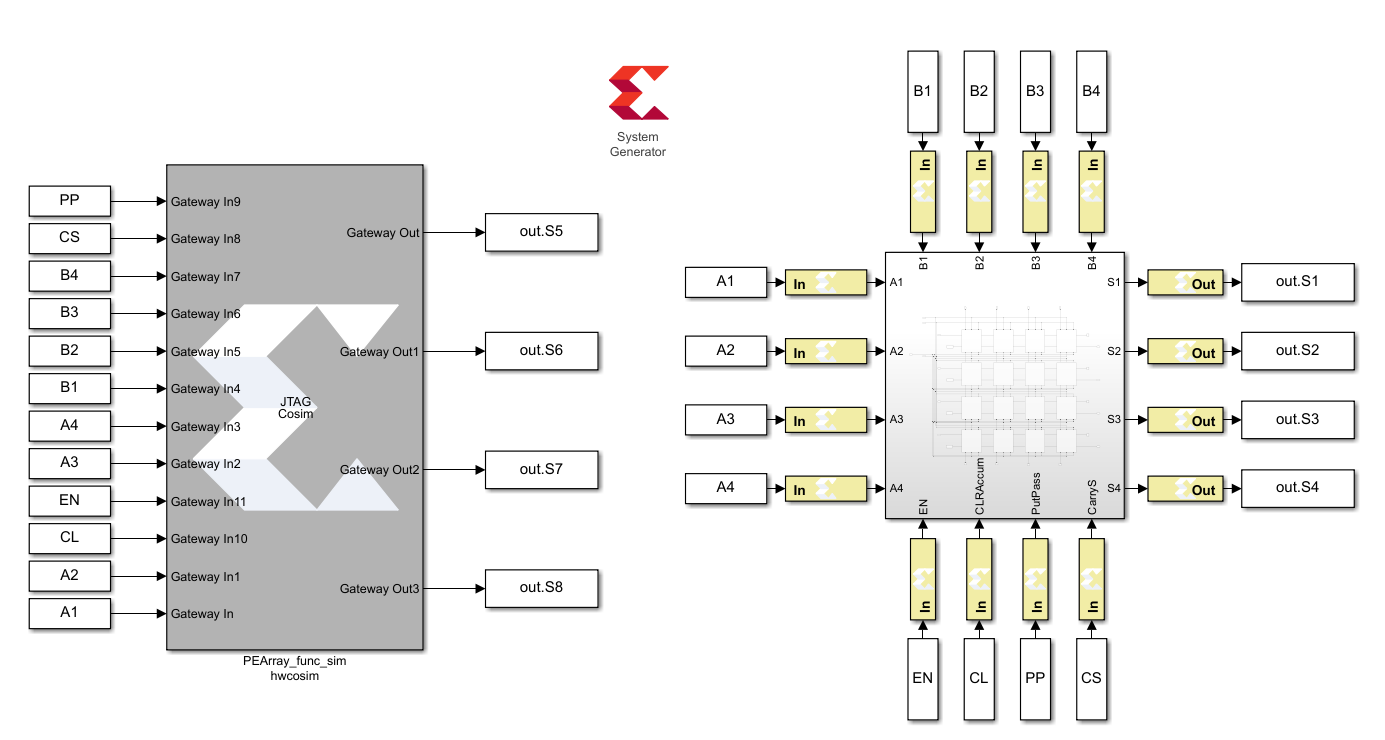
描述已自动生成图形用户界面

低可信度描述已自动生成

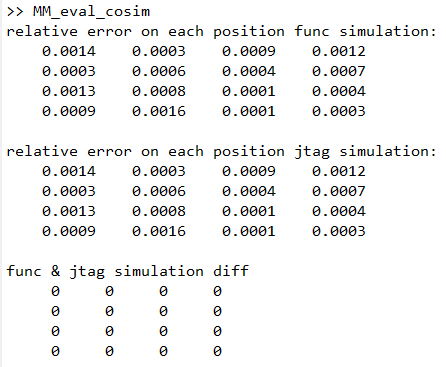
The above two figures showed the relative error on each element. All under 1%.

### 4.3.2 HW Co-simulation

After Generating using Model Composer, the following schematic is designed to evaluate correctness of the design.



The evaluation of jtag co-simulation is the same as functional simulation. The following screenshot shows the correctness of jtag simulation compared to the functional simulation.



# Evaluation Code – Matrix-Matrix Multiplication

MA = rand(4,4) \*2;

MB = rand(4,4) \*2;

simstep = 14;

AI1 = [MA(1,:),0,0,0,0,0,0,0,0,0,0,0];

AI2 = [0,MA(2,:),0,0,0,0,0,0,0,0,0,0];

AI3 = [0,0,MA(3,:),0,0,0,0,0,0,0,0,0];

AI4 = [0,0,0,MA(4,:),0,0,0,0,0,0,0,0];

BI1 = [MB(:,1)',0,0,0,0,0,0,0,0,0,0,0];

BI2 = [0,MB(:,2)',0,0,0,0,0,0,0,0,0,0];

BI3 = [0,0,MB(:,3)',0,0,0,0,0,0,0,0,0];

BI4 = [0,0,0,MB(:,4)',0,0,0,0,0,0,0,0];

ENI = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];

CLI = [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0];

PPI = [1,1,1,1,1,1,1,1,1,1,1,0,1,1,1];

CSI = [0,0,0,0,0,0,0,0,0,0,0,1,1,1,1];

A1.time = 0:1:simstep;

A2.time = 0:1:simstep;

A3.time = 0:1:simstep;

A4.time = 0:1:simstep;

B1.time = 0:1:simstep;

B2.time = 0:1:simstep;

B3.time = 0:1:simstep;

B4.time = 0:1:simstep;

EN.time = 0:1:simstep;

CL.time = 0:1:simstep;

PP.time = 0:1:simstep;

CS.time = 0:1:simstep;

A1.signals.dimensions = 1;

A2.signals.dimensions = 1;

A3.signals.dimensions = 1;

A4.signals.dimensions = 1;

B1.signals.dimensions = 1;

B2.signals.dimensions = 1;

B3.signals.dimensions = 1;

B4.signals.dimensions = 1;

EN.signals.dimensions = 1;

CL.signals.dimensions = 1;

PP.signals.dimensions = 1;

CS.signals.dimensions = 1;

A1.signals.values = AI1';

A2.signals.values = AI2';

A3.signals.values = AI3';

A4.signals.values = AI4';

B1.signals.values = BI1';

B2.signals.values = BI2';

B3.signals.values = BI3';

B4.signals.values = BI4';

EN.signals.values = ENI';

CL.signals.values = CLI';

PP.signals.values = PPI';

CS.signals.values = CSI';

out = sim('PEArray\_func\_sim',15);

oS1 = out.S1.Data';

oS2 = out.S2.Data';

oS3 = out.S3.Data';

oS4 = out.S4.Data';

O = [flip(oS1(1,13:end));

flip(oS2(1,13:end));

flip(oS3(1,13:end));

flip(oS4(1,13:end))];

G = MA \* MB;

E = G - O;

relativeErr = abs(E) ./ abs(G);

disp("relative error on each position:");

disp(relativeErr);

# Evaluation Code – Matrix-Matrix Multiplication (JTAG)

MA = rand(4,4) \*2;

MB = rand(4,4) \*2;

simstep = 14;

AI1 = [MA(1,:),0,0,0,0,0,0,0,0,0,0,0];

AI2 = [0,MA(2,:),0,0,0,0,0,0,0,0,0,0];

AI3 = [0,0,MA(3,:),0,0,0,0,0,0,0,0,0];

AI4 = [0,0,0,MA(4,:),0,0,0,0,0,0,0,0];

BI1 = [MB(:,1)',0,0,0,0,0,0,0,0,0,0,0];

BI2 = [0,MB(:,2)',0,0,0,0,0,0,0,0,0,0];

BI3 = [0,0,MB(:,3)',0,0,0,0,0,0,0,0,0];

BI4 = [0,0,0,MB(:,4)',0,0,0,0,0,0,0,0];

ENI = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];

CLI = [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0];

PPI = [1,1,1,1,1,1,1,1,1,1,1,0,1,1,1];

CSI = [0,0,0,0,0,0,0,0,0,0,0,1,1,1,1];

A1.time = 0:1:simstep;

A2.time = 0:1:simstep;

A3.time = 0:1:simstep;

A4.time = 0:1:simstep;

B1.time = 0:1:simstep;

B2.time = 0:1:simstep;

B3.time = 0:1:simstep;

B4.time = 0:1:simstep;

EN.time = 0:1:simstep;

CL.time = 0:1:simstep;

PP.time = 0:1:simstep;

CS.time = 0:1:simstep;

A1.signals.dimensions = 1;

A2.signals.dimensions = 1;

A3.signals.dimensions = 1;

A4.signals.dimensions = 1;

B1.signals.dimensions = 1;

B2.signals.dimensions = 1;

B3.signals.dimensions = 1;

B4.signals.dimensions = 1;

EN.signals.dimensions = 1;

CL.signals.dimensions = 1;

PP.signals.dimensions = 1;

CS.signals.dimensions = 1;

A1.signals.values = AI1';

A2.signals.values = AI2';

A3.signals.values = AI3';

A4.signals.values = AI4';

B1.signals.values = BI1';

B2.signals.values = BI2';

B3.signals.values = BI3';

B4.signals.values = BI4';

EN.signals.values = ENI';

CL.signals.values = CLI';

PP.signals.values = PPI';

CS.signals.values = CSI';

out = sim('PEArray\_co\_sim',15);

oS1 = out.S1.Data';

oS2 = out.S2.Data';

oS3 = out.S3.Data';

oS4 = out.S4.Data';

oS5 = out.S5.Data';

oS6 = out.S6.Data';

oS7 = out.S7.Data';

oS8 = out.S8.Data';

O = [flip(oS1(1,13:end));

flip(oS2(1,13:end));

flip(oS3(1,13:end));

flip(oS4(1,13:end))];

H = [flip(oS5(1,13:end));

flip(oS6(1,13:end));

flip(oS7(1,13:end));

flip(oS8(1,13:end))];

G = MA \* MB;

Es = G - O;

Eh = G - H;

relativeErrS = abs(Es) ./ abs(G);

relativeErrH = abs(Eh) ./ abs(G);

disp("relative error on each position func simulation:");

disp(relativeErrS);

disp("relative error on each position jtag simulation:");

disp(relativeErrH);

disp("func & jtag simulation diff");

disp(O - H)