Logic Design

EECS101001

Lab 4: Matrix Operations – Report

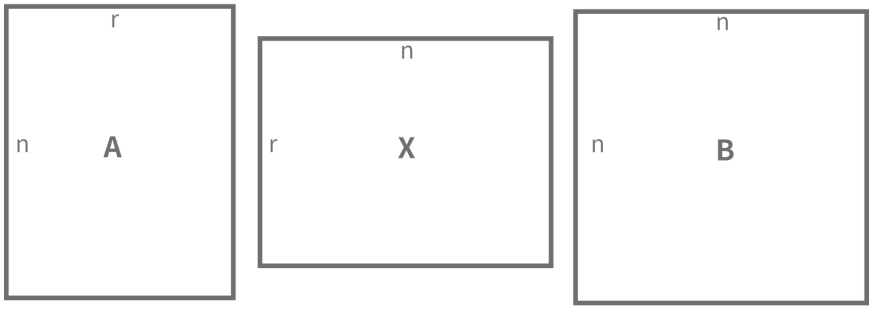
104021215 熊 磊

Matrix Operator

1. Foreword

This Lab is trying to implement a matrix operator. Finally we will get **Y = AX+B**; size of Y is *n x n*, size of A is *n x r*, size of X is *r x n*, and size of B is *n x n*.

The input includes `clk`, `reset` and `in_data`, while the output includes index of the matrix we want to read, function choice, out_data and finish.

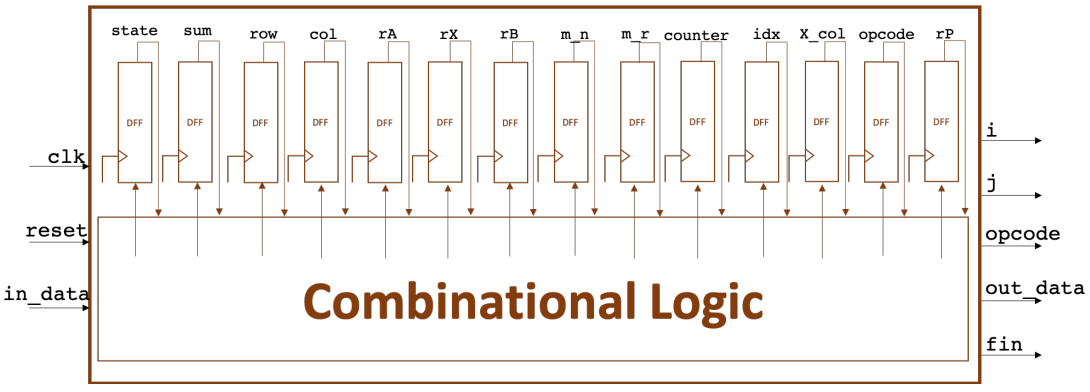


Input

name	width	function
clk	1	時間訊號
reset	1	reset=0 進行 reset
in_data	10	讀取的資料

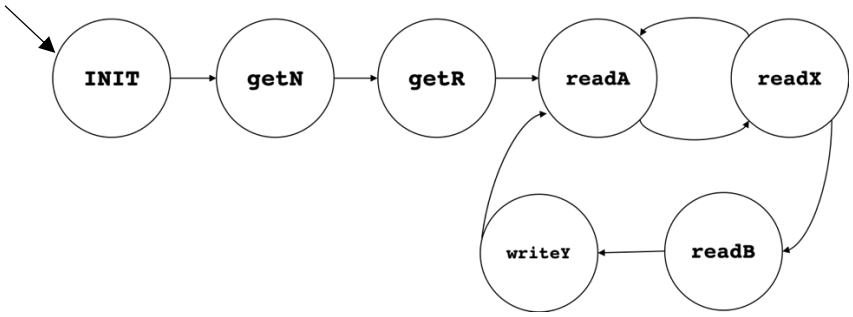
Output

name	width	function
i	10	row index 0<=i<=9
j	10	column index 0<=i<=9
opcode	3	控制讀寫
out_data	20	輸出的資料
fin	1	結束時讓 fin=1



2. Implementation Details and Designs

First, we draw a State Transition Graph as Follow:



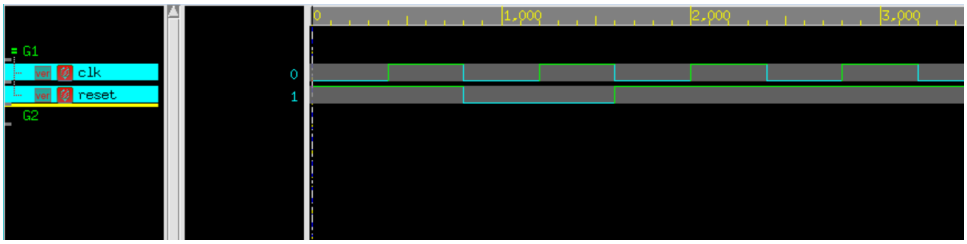
We need at least 7 States to detect the pattern, hence we must define states with at least 3 bits.

```
1  `define INIT    3'b111
2  `define GET_N   3'b000
3  `define GET_R   3'b001
4  `define READ_A  3'b010
5  `define READ_X  3'b011
6  `define READ_B  3'b100
7  `define WRITE_Y 3'b101
```

Note that we have to design an independent DFF outside the MO module. And this time we need many DFFs to store lots of data, it will be introduced in detail later.

In the beginning, first **negedge** will get `reset = 0`. And we set `next_state` to **INIT** when `reset == 0`.

```
108    assign next_state = (reset == 1'b0)? `INIT : next_state1;
```



Now we determine when to change states, and to which state.

```
95     always@(*)begin // next_state
96         case(state)
97             `INIT: next_state1 = `GET_N;
98             `GET_N: next_state1 = `GET_R;
99             `GET_R: next_state1 = `READ_A;
100            `READ_A: next_state1 = `READ_X;
101            `READ_X: next_state1 = (counter == 2*m_r-10'b1)? `READ_B : `READ_A;
102            `READ_B: next_state1 = `WRITE_Y;
103            `WRITE_Y: next_state1 = `READ_A;
104            default: next_state1 = `INIT;
105        endcase
106    end
```

Note that `counter` is used to count how many times that matrix A and matrix X have been read. When it read `2*r` times, which means it had completed once inner product computation, ready to add the value of matrix B, and was going to change state to `readB`.

In each DFFs, it stores lots of data, and, also, control the main operation. The function of each wire is as follows:

wire	function
state	Transmit the current state, initial will be **INIT**.
sum	Transmit the sum of inner produce of A and X, when state is **READ_A** and **READ_B**. And it will reset to 0 in state **WRITE_Y** and **INIT**.
row	Transmit the row index of A. When it finishes r times inner product for each column in X, row index will increase by 1.
col	Transmit the column index of A. When it finishes once element multiplication of inner product, col index will increase by 1, and it will reset to 0 when it finishes once inner product of each colume in state **WRITE_Y**.
rA	Transmit the value of A[row][col].
rX	Transmit the value of X[col][X_col].
rB	Transmit the value of B[row][X_col].
rP	Transmit the value of correct rA that is going to be multiply by rX.
m_n	Transmit the row size of matrix A.
m_r	Transmit the column size of matrix A.
counter	Transmit the times that read in matrix A and matrix X. And when it has read 2r times, finished once inner produce of matrix multiplication, it will reset to 0.
idx	Transmit the turns that whether matrix A or matrix X take. 0 is for matrix A, and 1 is for matrix X.
X_col	Transmit the column index of X.

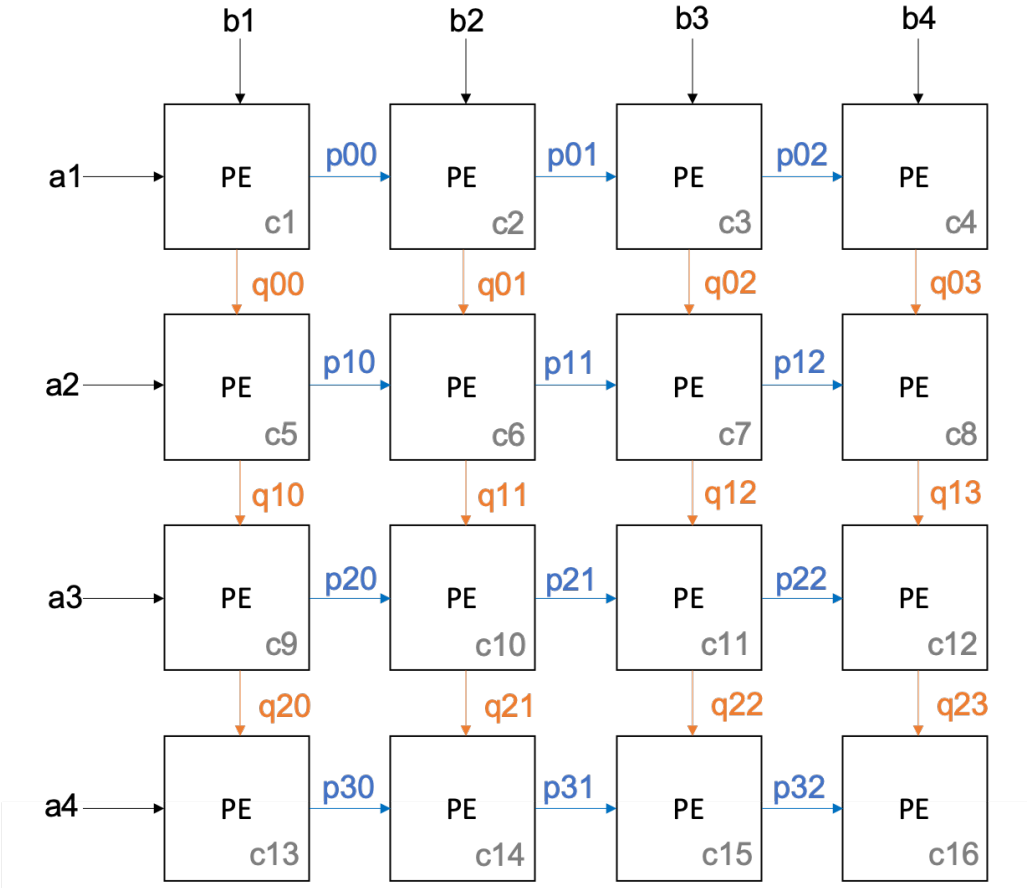
Hence, we can do the matrix operation!

Systolic Array

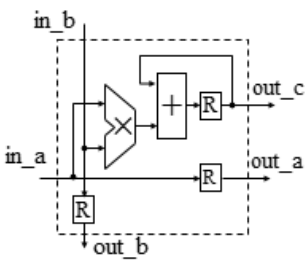
1. Forewords

This lab is going to design a 4x4 multiplication using systolic array. The idea is very simple, systolic array has a network of tightly coupled data processing units (DPUs), say nodes. Each node independently computes a partial result as a function of the data received from its upstream neighbors, stores the result within itself and passes it downstream.

For this lab, the input will come from a1~a4, b1~b4. And output will toward c1~c16. We draw a simple diagram below, and we are ready to design.

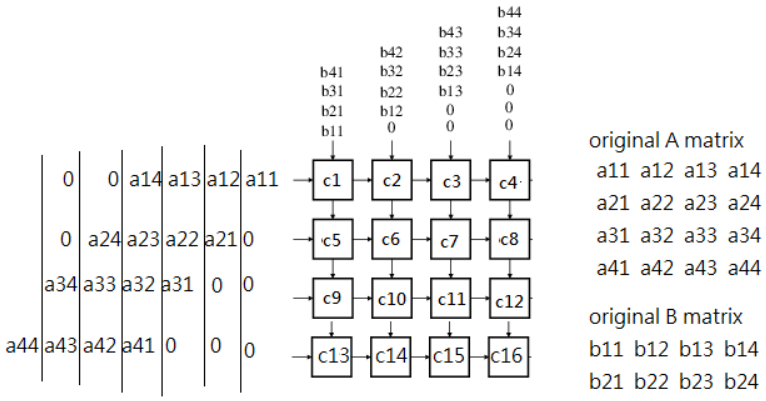


Where the design of Processing Element (PE) we use here is



Hence, the PE can be described like this:

```
module PE(clk,reset,in_a,in_b,out_a,out_b,out_c);
    parameter size=10;
    input wire reset,clk;
    input wire [size-1:0] in_a,in_b;
    output reg [2*size-1:0] out_c;
    output reg [size-1:0] out_a,out_b;

    always @(posedge clk)begin
        if(reset) begin
            out_a=0;
            out_b=0;
            out_c=0;
        end
        else begin
            out_c=out_c+in_a*in_b;
            out_a=in_a;
            out_b=in_b;
        end
    end
endmodule
```



original A matrix

a11 a12 a13 a14
a21 a22 a23 a24
a31 a32 a33 a34
a41 a42 a43 a44

original B matrix

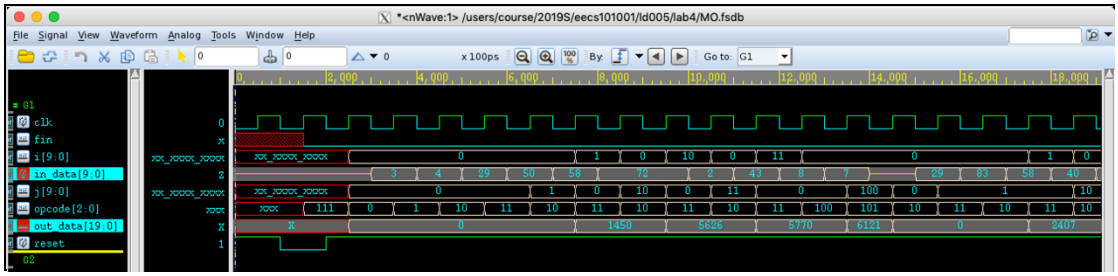
b11 b12 b13 b14
b21 b22 b23 b24
b31 b32 b33 b34
b41 b42 b43 b44

In SA module, we can instantiate 16 PEs and wait for 11 cycles, it will finish $N^3(= 4^3)$ multiplications.

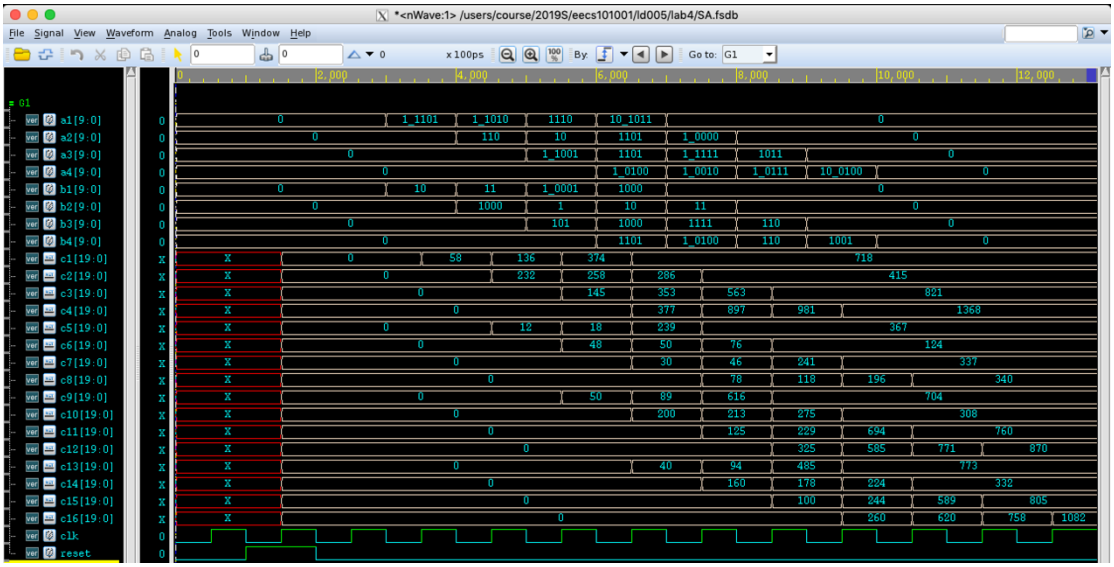
```
module SA(clk,reset,a1,a2,a3,a4,b1,b2,b3,b4,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,c14,c15,c16);

    parameter size=10;
    input clk,reset;
    input  [size-1:0] a1,a2,a3,a4,b1,b2,b3,b4;
    output  [2*size-1:0] c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,c14,c15,c16;

    wire [size-1:0] p00,p01,p02,p10,p11,p12,p20,p21,p22,p30,p31,p32,q00,q01,q02,q03,q10,q11,q12,q13,q20,q21,q22,q23;

    PE pe1 (.clk(clk), .reset(reset), .in_a(a1),    .in_b(b1),  .out_a(p00),.out_b(q00),.out_c(c1) );
    PE pe2 (.clk(clk), .reset(reset), .in_a(p00),   .in_b(b2),  .out_a(p01),.out_b(q01),.out_c(c2) );
    PE pe3 (.clk(clk), .reset(reset), .in_a(p01),   .in_b(b3),  .out_a(p02),.out_b(q02),.out_c(c3) );
    PE pe4 (.clk(clk), .reset(reset), .in_a(p02),   .in_b(b4),  .out_a(),   .out_b(q03),.out_c(c4) );

    PE pe5 (.clk(clk), .reset(reset), .in_a(a2),    .in_b(q00), .out_a(p10),.out_b(q10),.out_c(c5) );
    PE pe6 (.clk(clk), .reset(reset), .in_a(p10),   .in_b(q01), .out_a(p11),.out_b(q11),.out_c(c6) );
    PE pe7 (.clk(clk), .reset(reset), .in_a(p11),   .in_b(q02), .out_a(p12),.out_b(q12),.out_c(c7) );
    PE pe8 (.clk(clk), .reset(reset), .in_a(p12),   .in_b(q03), .out_a(),   .out_b(q13),.out_c(c8) );

    PE pe9  (.clk(clk), .reset(reset), .in_a(a3),   .in_b(q10), .out_a(p20),.out_b(q20),.out_c(c9) );
    PE pe10 (.clk(clk), .reset(reset), .in_a(p20),  .in_b(q11), .out_a(p21),.out_b(q21),.out_c(c10));
    PE pe11 (.clk(clk), .reset(reset), .in_a(p21),  .in_b(q12), .out_a(p22),.out_b(q22),.out_c(c11));
    PE pe12 (.clk(clk), .reset(reset), .in_a(p22),  .in_b(q13), .out_a(),   .out_b(q23),.out_c(c12));

    PE pe13 (.clk(clk), .reset(reset), .in_a(a4),   .in_b(q20), .out_a(p30),.out_b(),   .out_c(c13));
    PE pe14 (.clk(clk), .reset(reset), .in_a(p30),  .in_b(q21), .out_a(p31),.out_b(),   .out_c(c14));
    PE pe15 (.clk(clk), .reset(reset), .in_a(p31),  .in_b(q22), .out_a(p32),.out_b(),   .out_c(c15));
    PE pe16 (.clk(clk), .reset(reset), .in_a(p32),  .in_b(q23), .out_a(),   .out_b(),   .out_c(c16));
endmodule
```

3. nWave Result

Matrix Operator



Systolic Array



4. Report Area

Matrix Operator

```
****************************************
Report : area
Design : MO
Version: K-2015.06-SP1
Date   : Tue May 21 03:43:51 2019
****************************************

Library(s) Used:

    slow (File: /theda21_2/CBDK_IC_Contest/cur/SynopsysDC/db/slow.db)

Number of ports:                          546
Number of nets:                          1480
Number of cells:                          885
Number of combinational cells:            727
Number of sequential cells:               138
Number of macros/black boxes:               0
Number of buf/inv:                        132
Number of references:                      47

Combinational area:               8692.385453
Buf/Inv area:                      512.614792
Noncombinational area:            3488.157061
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                 12180.542514
Total area:                 undefined
```

Systolic Array

```
****************************************
Report : area
Design : SA
Version: K-2015.06-SP1
Date   : Tue May 21 04:37:08 2019
****************************************

Library(s) Used:

    slow (File: /theda21_2/CBDK_IC_Contest/cur/SynopsysDC/db/slow.db)

Number of ports:                         3026
Number of nets:                          8018
Number of cells:                         4208
Number of combinational cells:           3504
Number of sequential cells:               656
Number of macros/black boxes:               0
Number of buf/inv:                        208
Number of references:                      16

Combinational area:              62953.172359
Buf/Inv area:                      706.118389
Noncombinational area:           18467.712402
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                 81420.884762
Total area:                 undefined
```

5. Encountered Problems and Solving
   - In this lab, we have to be very careful in handling row index and col index.
   - Also, determine when to change state from **READ_X** to **READ_B** is important, here I use counter for help.
   - In systolic array, we can do more thing in same limited time, helping us increasing throughput. However, it needs more costs in space and money.

6. Questions
   - This time the lab is very interesting. Most problem have been figured out on my own!

7. Impression and Experience
   Finite State Machine is very powerful and also amazing. Everything if we can present in FSMs, then we can describe clearly! Also, thank to TAs, the end of the semester is coming, you did a great job! ☺