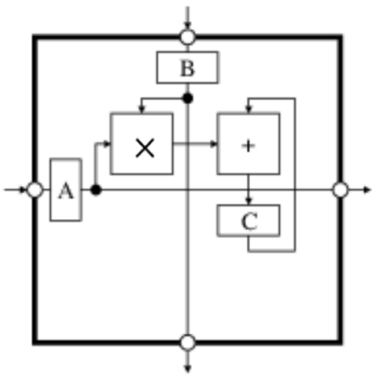
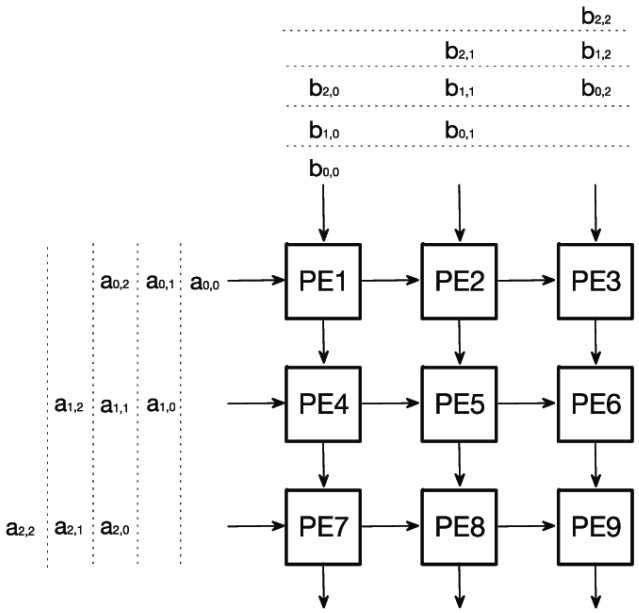
LAB#B Systolic array

- Introduction

Systolic array 是一硬體上實現矩陣乘法的一種方式，由 M*N 個 PE 組成，每一個 PE 都是由一個乘法器和加法器組成，架構和資料傳輸方向如下圖，其中 A、B 代表 A、B 矩陣的其中一個 element，C 代表當前的 partial sum。



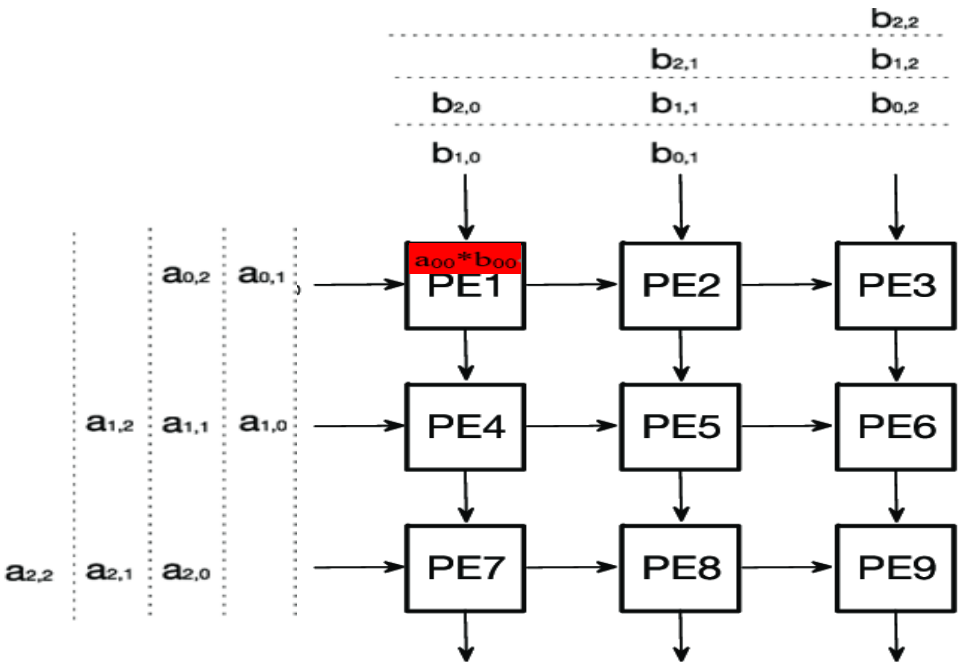
將 PE 組成 M*N 的矩陣排列其架構如下(圖為 3x3 矩陣)。



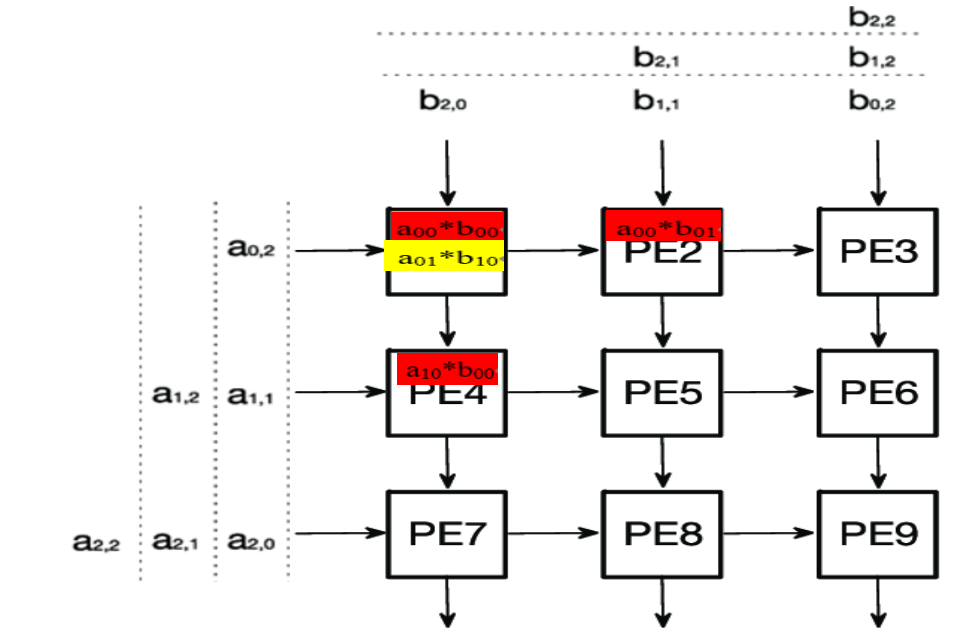
其運算流程如下。

假設兩輸入矩陣皆為 3x3 矩陣，兩矩陣相乘 A*B，個別的 element 用小寫 a、b 和下標代表對應的矩陣位置

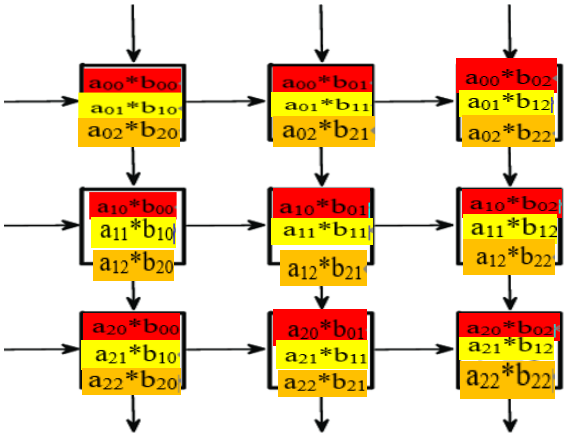
Step1:先將 a_{00} 和 b_{00} 移入 PE1 後相乘並暫存起來，如下圖片所示。



Step2:將 a_{00} 往右移給 PE2、b_{00} 往下移給 PE4，同時 a_{01} 和 b_{10} 傳入 PE1、a_{10} 傳入 PE4 並與 b_{00} 相乘、b_{01} 傳入 PE2 並與 a_{00} 相乘，結果如下。



Step3:接著按上述規則將 A 的每一個 element a 往右傳給下一個 PE、B 的每一個 element b 往下傳給下一個 PE，完成以下過程。



將每一個 PE 相乘的結果相加，得到結果就是 A*B 矩陣相乘的結果。

- HLS 實現方式

  code:

```
systolic1:
    for (int k = 0; k < a_col; k++) {
#pragma HLS LOOP_TRIPCOUNT min = c_size max = c_size
    systolic2:
        for (int i = 0; i < MAX_SIZE; i++) {
#pragma HLS UNROLL
        systolic3:
            for (int j = 0; j < MAX_SIZE; j++) {
#pragma HLS UNROLL
                // Get previous sum
                int last = (k == 0) ? 0 : localC[i][j];

                // Update current sum
                // Handle boundary conditions
                int a_val = (i < a_row && k < a_col) ? localA[i][k] : 0;
                int b_val = (k < b_row && j < b_col) ? localB[k][j] : 0;
                int result = last + a_val * b_val;

                // Write back results
                localC[i][j] = result;
            }
        }
    }
```

在 systolic1 k=0 時，會載入在 introduction 圖片中的紅色部分的運算，但這裡與在 introduction 的步驟不同，不是依序右移和下移載入，而是為了硬體使用效率最大化，內層的 systolic2 和 3 pragma 有使用 unroll，同一時間同時載入紅色區塊，暫存在 localC 裡面，也就是每一個 PE 裡面。

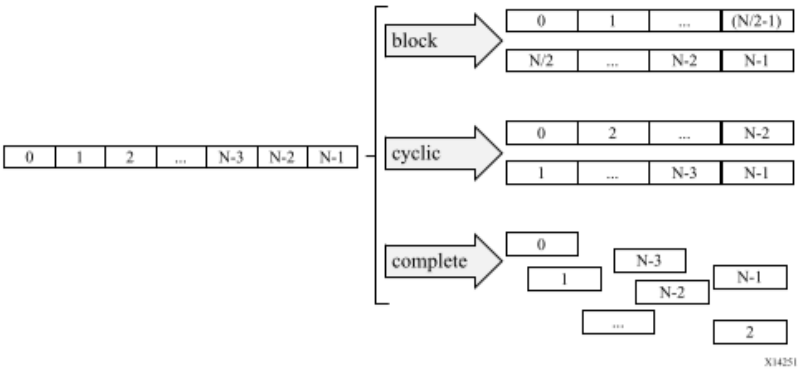
在 systolic1 k=1 時，會載入在 introduction 圖片中的黃色部分的運算，也是同時全部一次載入，並將紅色和黃色的部分相加，暫存在 localC

在 systolic1 k=2 時，會載入在 introduction 圖片中的橘色部分的運算，也是同時全部一次載入，並將紅色和黃色和橘色的部分相加，但注意只會相加一次，因為原先存在 localC 就是紅色和黃色的部分，並存回 localC。
若為 3x3 矩陣，到這步就完成全部運算。

- Array partition

array_partition 是將 array 區分成幾個區塊，若 factor=4 就是區分成 4 個，並同時輸出入，且會根據不同的 type 有不同的區分方式如下。

Figure 57: **Array Partitioning**



1. block 是將 array 區分成 N/factor 個，並同時輸出 factor 筆。
2. cyclic 是先輸出 array 的 index 為 0~factor-1 的位置數值，下一個 clock 再輸出 factor~2 factor-1，以此類推。
3. complete 則是將 array 所有數值一次輸出。

在這次 lab 中的 array partition 還有使用到 dim 參數。dim 是指在多為矩陣中，要根據哪一個維度進行區分，像是如果宣告一陣列 A[5][4]，若設定 factor = 2、dim=2 的話，則 A 會被分成[5][2]。
詳細如下圖所示。

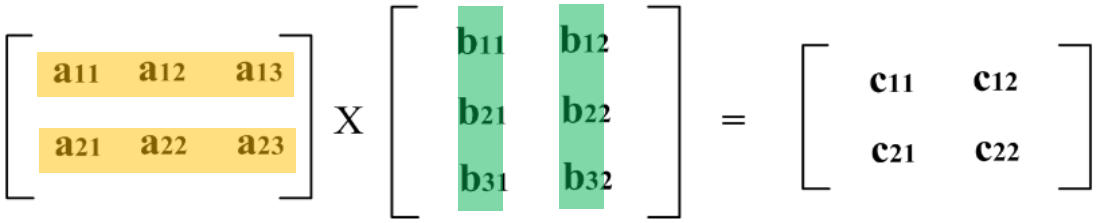
cyclic	factor = 2		dim = 1
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23

cyclic	factor = 2		dim = 2
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23

因此在本次 lab 中有設定 A、B 兩個陣列根據不同的維度分區。
程式碼如下。

```
int localA[MAX_SIZE][MAX_SIZE];
#pragma HLS ARRAY_PARTITION variable = localA dim = 1 complete
int localB[MAX_SIZE][MAX_SIZE];
#pragma HLS ARRAY_PARTITION variable = localB dim = 2 complete
```

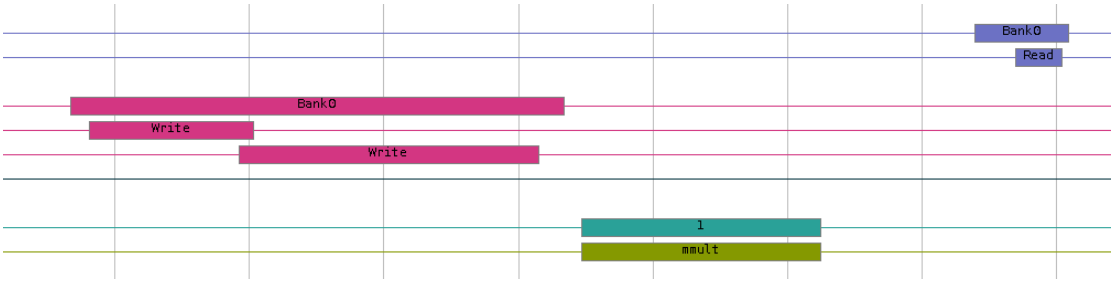
按上面程式碼設定 A、B 的區分如下圖所示



dim=1　　dim=2

- Timeline

模擬的 time line 如下。



可以看到 write 有兩個，必須先把 A、B 矩陣先寫至 data buffer 裡面，其中 mmult 和 write 有相依性，必須等到 write 結束後 mmult 才會開始運行。
最後輸出僅有一個 C 且花費時間較短，因為是將 C 的 array partition 設定為 dim=0 type=complete。

- Report comparison

我將 for-loop 的 systolic2 和 systolic3 改成 roll 和 unroll，兩者的 runtime 進行比較。

Hw-emulation	Roll	Unroll
runtime	0.04ms	0.021ms

模擬結果滿令我意外，按參數設定，會是 16x16 的矩陣乘 16x16 的矩陣運算，若沒 unroll 也就是得執行至少 256 個 clock cycle，花費時間應為 unroll 的 256 倍，但結果卻只有兩倍，我想應該是編譯器進行優化。

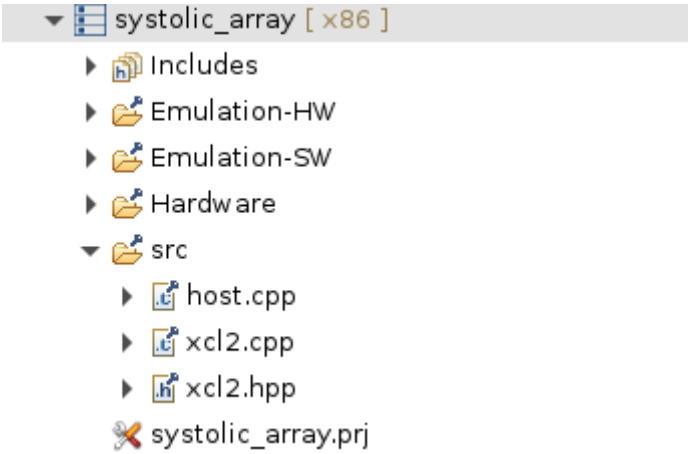
- 專案

  這個題目無法直接從 github 上的資訊建立 systolic array，所以我採用與 lab3 一樣的專案建立方式，使用 vitis 幫我產生出 xclbin 檔。

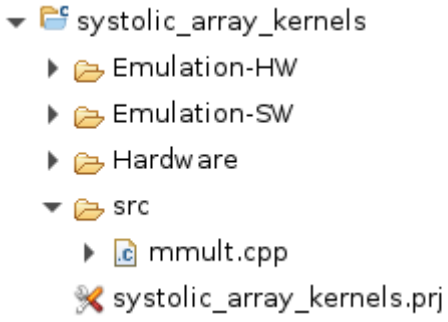
  步驟如下:

  step1:建立專案時選擇 u50 板子

  step2:將我附上的 github 連結中的 4 個 cpp 檔載下，並將 host.cpp、xcl.cpp、xcl.h 放置在 host 中的 src 資料夾。



  step3:將 mmult.cpp 放置在 kernel 中的 src 資料夾。



  step4:將 mmult 加速 hw function。

  step5:開始執行 sw-emulation、hw-emulation、hardware。

  step6:新增 System Project Debug 組態時，編輯 Program Arguments，只需要./binary_container_1.xclbin 即可。

- github

  https://github.com/ecoyukino/LAB_B_SystolicArray