<n26112437>_<劉兆軒> AIAS 2023 Lab 9 HW Submission

This is copied version.
Contributer: **Haouo**

1. 請不要用這份template 交作業, 建立一個新的codimd 檔案, 然後copy & paste 這個 template 到你創建的檔案做修改。
2. 請修改你的學號與姓名在上面的 title, 以避免TA 修改作業時把檔案跟人弄錯了
3. 在Playlab 作業中心繳交作業時, 請用你創建的檔案鏈結繳交, 其他相關的資料與鏈結請依照Template 規定的格式記載於codimd 上。

記得在文件標題上修改你的 <學號> <姓名>

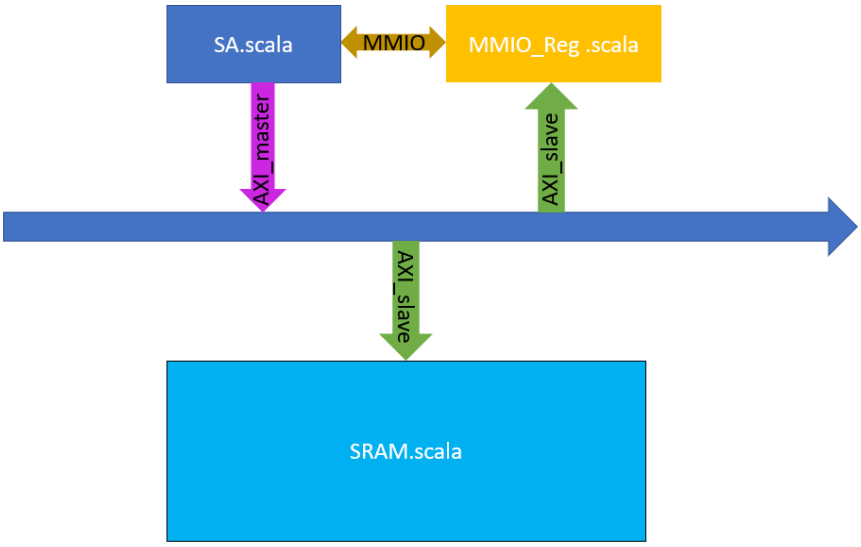
Gitlab code link

Please paste the link to your private Gitlab repository for this homework submission here.

- https://playlab.computing.ncku.edu.tw:4001/kevin1217/lab09
  (https://playlab.computing.ncku.edu.tw:4001/kevin1217/lab09)

Homework 9-1 - Make accelerator move data by itself

- 說明與要求：
  - 根據作業要求，同學的block diagram應如下圖所示:



將 systolic array 內的 local memory 移除，讓 SA 是透過 AXI bus 直接向 SRAM 拿資料，因此會需要幫 SA 加上一個 master port。此外，在 HW9-1 中，同學可以暫時不將 CPU 掛載上 bus，先將矩陣乘法會用到的 data preload 進 SRAM，並且事先給予 MMIO resigters 運算過程中會用到初始值的方式，來模擬 CPU setup SA 的過程，如此一來可以簡化整個測試流程和測試的 scale。

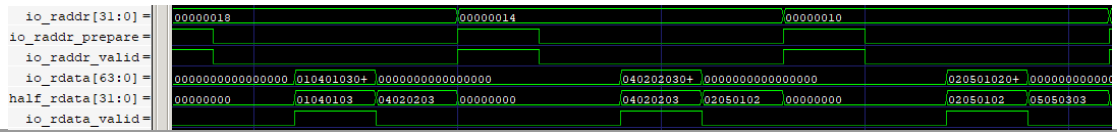
- 在 memory space 不變的情況，請同學先將矩陣乘法所需的資料 preload 進 SRAM，演示一次**SA從SRAM讀值並將運算後的結果寫回SRAM的過程**，透過 emulator 轉化出硬體所需的檔案，需要做的有兩件事情：

  1. **SRAM 預設初始值**，可以和 Lab9-1-2 的矩陣乘法內容相同，而 SA 透過 AXI bus 對其進行讀寫
  2. **MMIO_Regfile也會有預設值**，可以在一開始 `ENABLE` 設為 HIGH，並且指定好 matrices A、B、C 的 addresses，讓 SA 一開始就能夠進行資料讀取，就不用透過 CPU 來做 setup。

- 需要克服的點：

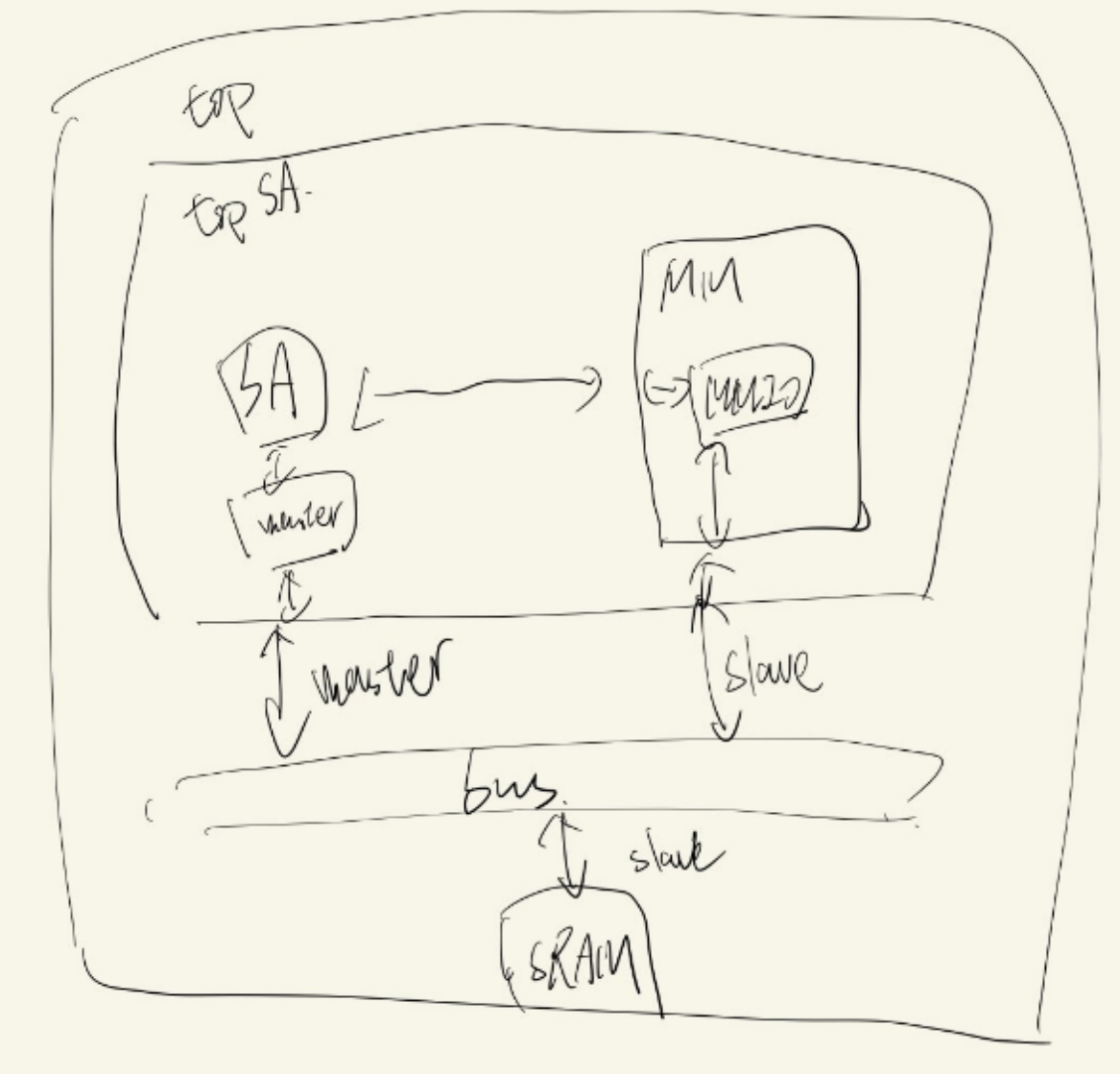
  - 當資料不再是每一個clock送入一筆，SA該如何因應，請同學寫下你是如何解決？要注意的是，這裡的SA讀寫資料所花的時間應假設為**未知**，而非fix-length!! (hint: **FSM Design**)
  - 針對 SA FSM Design，芸甄 同學有寫一份更詳細的文件，同學可以參考 - SA master設計的注意要點 (https://playlab.computing.ncku.edu.tw:3001/535US-VgTjCHWjYYSUNX8w?view)
  - **Ans:**

    當收到rdata_valid後隔一個cycle，我才會對cnt計數(留一個cycle才能存到第二個half_data)，以及讓raddr_prepare為true.B(使得raddr_valid為true)，否則raddr保持原樣，這樣子raddr和rdata才不會亂掉。

- 結果呈現：
  - 利用在 Lab9-1-3 裡所提供的方法(printf)，可以根據 STATUS 信號，印出你SRAM裡存放輸入矩陣以及輸出矩陣的資訊，截圖貼上來。
  - 測試方式說明:
    // 說明你的測試方式與內容，資料的存放位址等等。

額外撰寫一個top用來當作最外層，主要是參考topVectorCPU做修改，還有新增SA_master當作和bus溝通的橋樑，其中memory初始值放在h8000的位址，a_base_addr放在sram的0x00、b_base_addr放在0x10、c_base_addr放在0x20，下圖是主要架構。



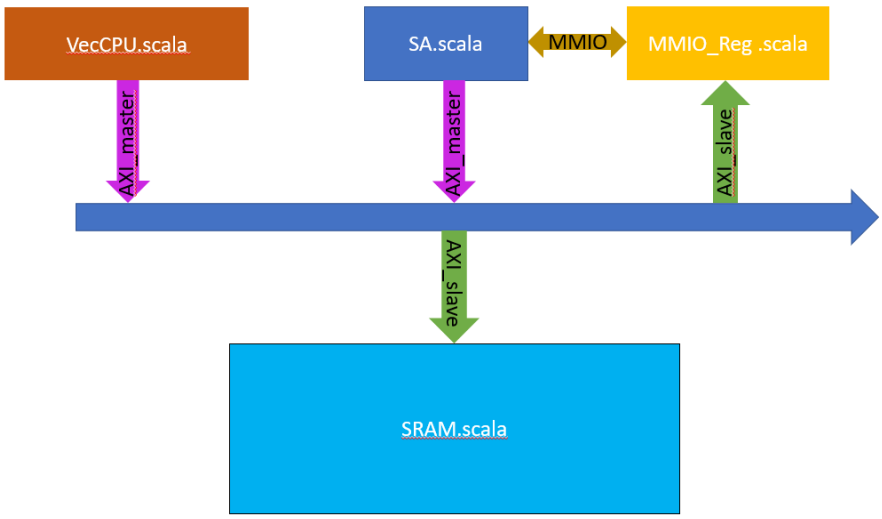
測試方式主要是寫一個對應top最外層的toptest，設計dm_data和dm_addr去讀memory的data，當矩陣做完運算後，去poke dm_addr給他我要看的memory位址，再透過peek去讀dm_data的內容，來驗證說最終答案的正確性。

  - 結果(貼圖):
    // 請把印出結果貼在這裡~

```
outChannelName: 00018941.out
cmdChannelName: 00018941.cmd
STARTING test_run_dir/aias_lab9.ontop.topTest1284677750/Vtop
[info] [0.004] SEED 1687171241189
[info] [0.018] A_mat :
[info] [0.019] MEM[07] ~ MEM[00] : 0x0202010304040303
[info] [0.020] MEM[15] ~ MEM[08] : 0x0303010304030303
[info] [0.020] B_mat :
[info] [0.021] MEM[23] ~ MEM[16] : 0x0205010205050303
[info] [0.032] MEM[31] ~ MEM[24] : 0x0104010304020203
[info] [0.032] C_mat :
[info] [0.032] MEM[39] ~ MEM[32] : 0x1b20101729361827
[info] [0.033] MEM[47] ~ MEM[40] : 0x2026131d25341624
Enabling waves..
Exit Code: 0
```

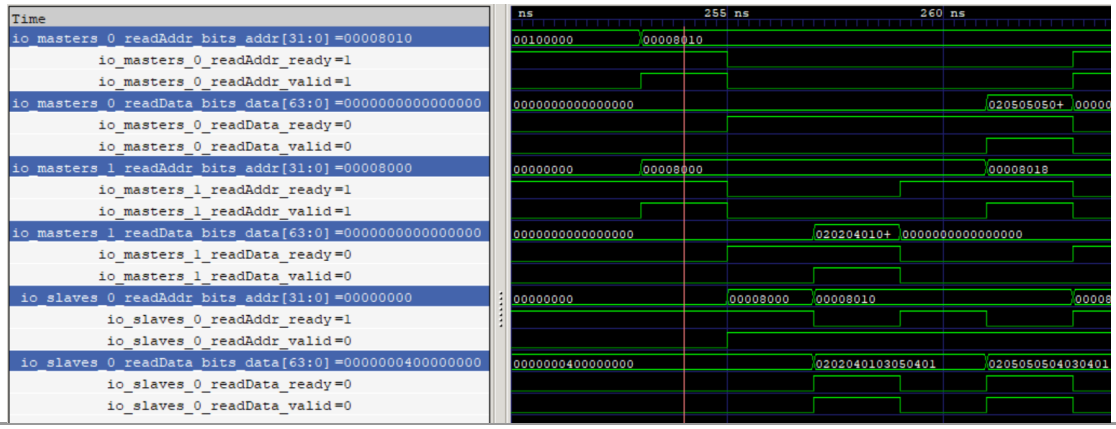
Homework 9-2 - Make the AXI bus support one more master (Accelerator)

- 說明與要求：
  - 根據作業要求，同學最後的block diagram應如下圖所示:



  - 在2 masters以上的情況下，Arbiter會去決定誰在當下發出的request"算數"，方式有兩種，一種是Round-Robbin(輪流)，另一種則是Priority(index越小，優先權越大)，在Lab中所採用的機制為前者(RRArbiter)，提供給同學參考的測試情境應如下：
    - CPU開啟ENABLE信號後，SA開始透過AXI讀取SRAM的值，假設CPU並非向lab所示範那樣hang住等待SA所發出的 `status` 信號，假設它接下來也需要對SRAM執行讀寫(舉例:執行SIMD運算)
    - 同學必須截下在2位master同時發出request的情況下中，Abiter會如何選擇，並擷取說明被選擇的那方的波形，以及沒有被選擇到的又該如何處理?
      // 截圖並說明

由波形圖可以看出當master0(CPU)和master1(SA)同時發出請求時，slave0(dm)透過排班優先處理了master1的request，沒被選中的master0會等待master1完成再接著執行



- Bonus：
  - 現有的AXI只能處理一筆request(將1 master與1 slave)，但若是2 master所發出的requests並不衝突(找不同的slaves)，同學是否能將AXI擴增至能平行處理多筆requests呢?(同時有多組master/slave配對成功。)

Homework 9-3 Put Everything together to run a larger NxN (where N is 16) matrix multiplication

- 說明與要求：
  - Option 1 (當輸入矩陣小於硬體時)
    - 無止境的放大生成硬體以符合輸入資料的要求，這種做法的缺點當資料遠小於硬體時會耗能，耗面積，但同學也應考慮到當實際硬體與運算資料的"形狀不符"時，該如何利用MMIO所提供的資訊去改變Controller的設定
      - 舉例：4x4 SA如何完成[2x3]x[3x2]的矩陣乘法呢?
  - Option 2 (當輸入矩陣大於硬體時)
    - 對資料做mapping(組合、拆分)以符合硬體規格(4x4)，此種作法較為實際，因硬體為有限且早已確定，應該將資料調整成符合硬體要求的樣子做運算。
  - 測資要求與結果呈現：
    - 你是採用Option 1 還是 Option 2，說明Controller是如何設計?(Option 1)，或者資料該如何調整成硬體所能接受的樣子(資料如何存放、如何mapping)(Option 2)。
    - 同學實作出的硬體要能夠運算以下測資(自行準備並在下方說明)：[2x3]x[3x2]、[4x4]x[4x4]、[16x16]x[16x16]三筆矩陣乘法。

1.[2x3]x[3x2]

- A矩陣：

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- B矩陣:

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 4 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- 存放以及mapping：
  將[2x3]和[3x2]矩陣的data調整成[4x4]矩陣的格式，將多的位置補0。

- 結果：
  最後存回去memory的格式應符合一般2x2矩陣的連續擺放

```
[info] [0.287]
[info] [0.288] Value in the Vector RegFile
[info] [0.289] vector_reg[00] : 0x0000000000000000 vector_reg[01] : 0x000000000a070502 vector_reg[02] : 0x0000000000000000 vector_reg[03] : 0x0000000000000000
[info] [0.289] vector_reg[04] : 0x0000000000000000 vector_reg[05] : 0x0000000000000000 vector_reg[06] : 0x0000000000000000 vector_reg[07] : 0x0000000000000000
[info] [0.289] vector_reg[08] : 0x0000000000000000 vector_reg[09] : 0x0000000000000000 vector_reg[10] : 0x0000000000000000 vector_reg[11] : 0x0000000000000000
[info] [0.290] vector_reg[12] : 0x0000000000000000 vector_reg[13] : 0x0000000000000000 vector_reg[14] : 0x0000000000000000 vector_reg[15] : 0x0000000000000000
[info] [0.290] vector_reg[16] : 0x0000000000000000 vector_reg[17] : 0x0000000000000000 vector_reg[18] : 0x0000000000000000 vector_reg[19] : 0x0000000000000000
[info] [0.290] vector_reg[20] : 0x0000000000000000 vector_reg[21] : 0x0000000000000000 vector_reg[22] : 0x0000000000000000 vector_reg[23] : 0x0000000000000000
[info] [0.290] vector_reg[24] : 0x0000000000000000 vector_reg[25] : 0x0000000000000000 vector_reg[26] : 0x0000000000000000 vector_reg[27] : 0x0000000000000000
[info] [0.290] vector_reg[28] : 0x0000000000000000 vector_reg[29] : 0x0000000000000000 vector_reg[30] : 0x0000000000000000 vector_reg[31] : 0x0000000000000000
Enabling waves..
```

2.[4x4]x[4x4]

- A矩陣：

$$\begin{bmatrix} 1 & 4 & 5 & 3 \\ 1 & 4 & 2 & 2 \\ 5 & 5 & 3 & 3 \\ 4 & 3 & 1 & 5 \end{bmatrix}$$

- B矩陣:

$$\begin{bmatrix} 1 & 4 & 3 & 4 \\ 5 & 5 & 5 & 2 \\ 4 & 4 & 5 & 4 \\ 3 & 4 & 3 & 5 \end{bmatrix}$$

- 存放以及mapping：
  data正常放，A矩陣起始位址為0x8000、B矩陣起始位址為8010、C矩陣起始位址為0x8020。

- 結果：

2.[16x16]x[16x16]

- A矩陣：

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 7 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 7 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 7 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 7 \\
2 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 8 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 8 & 0 & 0 \\
0 & 0 & 2 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 8 & 0 \\
0 & 0 & 0 & 2 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 8 \\
4 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 4 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 4 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 4 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 4 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 4 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 4
\end{pmatrix}$$

=

- B矩陣:

$$\begin{pmatrix}
2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 4 & 0 & 0 \\
0 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 4 & 0 \\
0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 4 \\
4 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\
0 & 4 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 & 0 \\
0 & 0 & 4 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 & 0 \\
0 & 0 & 0 & 4 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 3 \\
6 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\
0 & 6 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 2 & 0 & 0 \\
0 & 0 & 6 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 2 & 0 \\
0 & 0 & 0 & 6 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 2 \\
8 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 8 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 8 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 8 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 1
\end{pmatrix}$$

- 存放以及mapping：
  這裡是將[16x16]的矩陣透過Pertitioned Matrix將矩陣切成16個[4x4]的小矩陣，並將A和B矩陣對應的小矩陣相乘做累加，A矩陣起始位址為0x8000、B矩陣起始位址為8100、C矩陣起始位址為0x8200，其中要注意我memory_mapped.S檔案內的B矩陣切成Pertitioned Matrix放每個小矩陣data的順序要依照col majer去擺放。

- 結果：

$$\begin{pmatrix}
100 & 0 & 0 & 0 & 84 & 0 & 0 & 0 & 50 & 0 & 0 & 0 & 30 & 0 & 0 & 0 \\
0 & 100 & 0 & 0 & 0 & 84 & 0 & 0 & 0 & 50 & 0 & 0 & 0 & 30 & 0 & 0 \\
0 & 0 & 100 & 0 & 0 & 0 & 84 & 0 & 0 & 0 & 50 & 0 & 0 & 0 & 30 & 0 \\
0 & 0 & 0 & 100 & 0 & 0 & 0 & 84 & 0 & 0 & 0 & 50 & 0 & 0 & 0 & 30 \\
120 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 60 & 0 & 0 & 0 & 40 & 0 & 0 & 0 \\
0 & 120 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 60 & 0 & 0 & 0 & 40 & 0 & 0 \\
0 & 0 & 120 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 60 & 0 & 0 & 0 & 40 & 0 \\
0 & 0 & 0 & 120 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 60 & 0 & 0 & 0 & 40 \\
40 & 0 & 0 & 0 & 30 & 0 & 0 & 0 & 20 & 0 & 0 & 0 & 30 & 0 & 0 & 0 \\
0 & 40 & 0 & 0 & 0 & 30 & 0 & 0 & 0 & 20 & 0 & 0 & 0 & 30 & 0 & 0 \\
0 & 0 & 40 & 0 & 0 & 0 & 30 & 0 & 0 & 0 & 20 & 0 & 0 & 0 & 30 & 0 \\
0 & 0 & 0 & 40 & 0 & 0 & 0 & 30 & 0 & 0 & 0 & 20 & 0 & 0 & 0 & 30 \\
60 & 0 & 0 & 0 & 50 & 0 & 0 & 0 & 30 & 0 & 0 & 0 & 20 & 0 & 0 & 0 \\
0 & 60 & 0 & 0 & 0 & 50 & 0 & 0 & 0 & 30 & 0 & 0 & 0 & 20 & 0 & 0 \\
0 & 0 & 60 & 0 & 0 & 0 & 50 & 0 & 0 & 0 & 30 & 0 & 0 & 0 & 20 & 0 \\
0 & 0 & 0 & 60 & 0 & 0 & 0 & 50 & 0 & 0 & 0 & 30 & 0 & 0 & 0 & 20
\end{pmatrix}$$

  我把矩陣運算結果存到vec暫存器內，轉成10進制後會和正確答案一樣。

```
[info] [3.035]
[info] [3.036] Value in the Vector RegFile
[info] [3.037] vector_reg[00] : 0x0000640000000064 vector_reg[01] : 0x0000640000000064 vector_reg[02] : 0x0000540000000054 vector_reg[03] : 0x0000540000000054
[info] [3.037] vector_reg[04] : 0x0000320000000032 vector_reg[05] : 0x0000320000000032 vector_reg[06] : 0x00001e000000001e vector_reg[07] : 0x00001e000000001e
[info] [3.037] vector_reg[08] : 0x0000780000000078 vector_reg[09] : 0x0000780000000078 vector_reg[10] : 0x0000640000000064 vector_reg[11] : 0x0000640000000064
[info] [3.038] vector_reg[12] : 0x00003c000000003c vector_reg[13] : 0x00003c000000003c vector_reg[14] : 0x0000280000000028 vector_reg[15] : 0x0000280000000028
[info] [3.038] vector_reg[16] : 0x0000280000000028 vector_reg[17] : 0x0000280000000028 vector_reg[18] : 0x00001e000000001e vector_reg[19] : 0x00001e000000001e
[info] [3.039] vector_reg[20] : 0x0000140000000014 vector_reg[21] : 0x0000140000000014 vector_reg[22] : 0x00001e000000001e vector_reg[23] : 0x00001e000000001e
[info] [3.039] vector_reg[24] : 0x00003c000000003c vector_reg[25] : 0x00003c000000003c vector_reg[26] : 0x0000320000000032 vector_reg[27] : 0x0000320000000032
[info] [3.039] vector_reg[28] : 0x00001e000000001e vector_reg[29] : 0x00001e000000001e vector_reg[30] : 0x0000140000000014 vector_reg[31] : 0x0000140000000014
```

Homework 9-4 Prepare software for the Conv2D operation an compare with the result in Lab 8.

- 將在Lab 8裡用SIMD所完成的的Conv2D運算改由Systolic Array來實現，進而比較兩者的表現。所以需要做的事情有以下兩件

  1. 如何將 Conv2D 轉換成 Matrix Multiplication ?

> 我有寫一個ipynb透過torch.nn.Unfold函式將原本的input_data轉換成可以做Matrix Multiplication的格式，其中檔案附在memory-mapped-software資料夾內(AIAS_im2col.ipynb)。

  2. 如何利用 4x4 的 systolic array 處理超過 4x4 的矩陣乘法 ?

> 非4的倍數則補0到4的倍數，之後再用hw9-3提到的Pertitioned Matrix技巧將矩陣切割再做運算。

- Answer the question below

  1. 你覺得是SIMD-type accelerator 效能比較好, 還是 systolic-array-based 的accelerator 比較好 ?

> 我覺得這裡Systolic Array的加速器比SIMD型加速器更好，他可以達到的平行化效果更佳，但是Systolic Array要先做較麻煩的資料前處理，透過im2col和因應SA大小做Pertitioned Matrix才能進行conv的運算。
> 由下圖比較兩者花費的cycle數，會發現Systolic Array少了SIMD三倍的cycles數，所以Systolic Array用來計算conv2效果更好。
> Systolic Array: 花個1818cycles

```
[info] [0.887]
[info] [0.887] Value in the Vector RegFile
[info] [0.888] vector_reg[00] : 0x000078665442301e vector_reg[01] : 0x000078665442301e vector_reg[02] : 0x0000000000000000 vector_reg[03] : 0x0000000000001411
[info] [0.889] vector_reg[04] : 0x0000000000001411 vector_reg[05] : 0x0000000000007866 vector_reg[06] : 0x0000000000007866 vector_reg[07] : 0x0000000000007866
[info] [0.889] vector_reg[08] : 0x0000000000007866 vector_reg[09] : 0x0000000000000000 vector_reg[10] : 0x0000000000000000 vector_reg[11] : 0x0000000000000000
[info] [0.890] vector_reg[12] : 0x0000000000000000 vector_reg[13] : 0x0000000000000000 vector_reg[14] : 0x0000000000000000 vector_reg[15] : 0x0000000000000000
[info] [0.890] vector_reg[16] : 0x0000000000000000 vector_reg[17] : 0x0000000000000000 vector_reg[18] : 0x0000000000000000 vector_reg[19] : 0x0000000000000000
[info] [0.890] vector_reg[20] : 0x0000000000000000 vector_reg[21] : 0x0000000000000000 vector_reg[22] : 0x0000000000000000 vector_reg[23] : 0x0000000000000000
[info] [0.890] vector_reg[24] : 0x0000000000000000 vector_reg[25] : 0x0000000000000000 vector_reg[26] : 0x0000000000000000 vector_reg[27] : 0x0000000000000000
[info] [0.891] vector_reg[28] : 0x0000000000000000 vector_reg[29] : 0x0000000000000000 vector_reg[30] : 0x0000000000000000 vector_reg[31] : 0x0000000000000000
Enabling waves..
Exit Code: 0
[info] [0.921] RAN 1818 CYCLES PASSED
```

> SIMD: 花5838個cycles

```
[info] [3.684] Value in the Vector RegFile
[info] [3.685] vector_reg[00] : 0000000000000000 vector_reg[01] : 0302010007060504 vector_reg[02] : 210e2b786d5a4734 vector_reg[03] : 0000000000000000
[info] [3.685] vector_reg[04] : 0000000000000000 vector_reg[05] : 0000000000000000 vector_reg[06] : 0000000000000000 vector_reg[07] : 0000000000000000
[info] [3.685] vector_reg[08] : 0000000000000000 vector_reg[09] : 0000000000000000 vector_reg[10] : 0000000000000000 vector_reg[11] : 0000000000000000
[info] [3.686] vector_reg[12] : 0000000000000000 vector_reg[13] : 0000000000000000 vector_reg[14] : 0000000000000000 vector_reg[15] : 0000000000000000
[info] [3.686] vector_reg[16] : 0000000000000000 vector_reg[17] : 0000000000000000 vector_reg[18] : 0000000000000000 vector_reg[19] : 0000000000000000
[info] [3.686] vector_reg[20] : 0c2a78665442301e vector_reg[21] : 0c2a78665442301e vector_reg[22] : 0c2a78665442301e vector_reg[23] : 0c2a78665442301e
[info] [3.686] vector_reg[24] : 0c2a78665442301e vector_reg[25] : 0c2a78665442301e vector_reg[26] : 0000000000000000 vector_reg[27] : 0000000000000000
[info] [3.686] vector_reg[28] : 0000000000000000 vector_reg[29] : 0000000000000000 vector_reg[30] : 0000000000000000 vector_reg[31] : 0000000000000000
test top Success: 0 tests passed in 5843 cycles in 3.761473 seconds 1553.38 Hz
[info] [3.688] RAN 5838 CYCLES PASSED
```

  2. 如果你想要讓acclerator performance 更好, 你覺得你的design 上能做什麼改善?

像是這裡是做conv的運算，所以會有大量重複的計算過程，我可以省去一些重複的計算步驟來優化acclerator的 performance，如將資料經過im2col後，矩陣大小會非常大(這裡的input為18x36的size)，但會發現其中矩陣有大量的data重複，所以就可以省略掉一些空間(變成18x6的size)，整體而言的計算效能可以大幅提升。

結果:

將透過systolic-array算出來的值Load到vec暫存器，將v0,v2分別代表上下部分。

```
[info] [0.887]
[info] [0.887] Value in the Vector RegFile
[info] [0.888] vector_reg[00] : 0x0000078665442301e vector_reg[01] : 0x0000078665442301e vector_reg[02] : 0x0000000000000000 vector_reg[03] : 0x0000000000001411
[info] [0.889] vector_reg[04] : 0x0000000000001411 vector_reg[05] : 0x0000000000007866 vector_reg[06] : 0x0000000000007866 vector_reg[07] : 0x0000000000007866
[info] [0.889] vector_reg[08] : 0x0000000000007866 vector_reg[09] : 0x0000000000000000 vector_reg[10] : 0x0000000000000000 vector_reg[11] : 0x0000000000000000
[info] [0.890] vector_reg[12] : 0x0000000000000000 vector_reg[13] : 0x0000000000000000 vector_reg[14] : 0x0000000000000000 vector_reg[15] : 0x0000000000000000
[info] [0.890] vector_reg[16] : 0x0000000000000000 vector_reg[17] : 0x0000000000000000 vector_reg[18] : 0x0000000000000000 vector_reg[19] : 0x0000000000000000
[info] [0.890] vector_reg[20] : 0x0000000000000000 vector_reg[21] : 0x0000000000000000 vector_reg[22] : 0x0000000000000000 vector_reg[23] : 0x0000000000000000
[info] [0.890] vector_reg[24] : 0x0000000000000000 vector_reg[25] : 0x0000000000000000 vector_reg[26] : 0x0000000000000000 vector_reg[27] : 0x0000000000000000
[info] [0.891] vector_reg[28] : 0x0000000000000000 vector_reg[29] : 0x0000000000000000 vector_reg[30] : 0x0000000000000000 vector_reg[31] : 0x0000000000000000
Enabling waves..
```

比對lab8的值是對的！