

Name: 王領崧

Student ID: 107062107

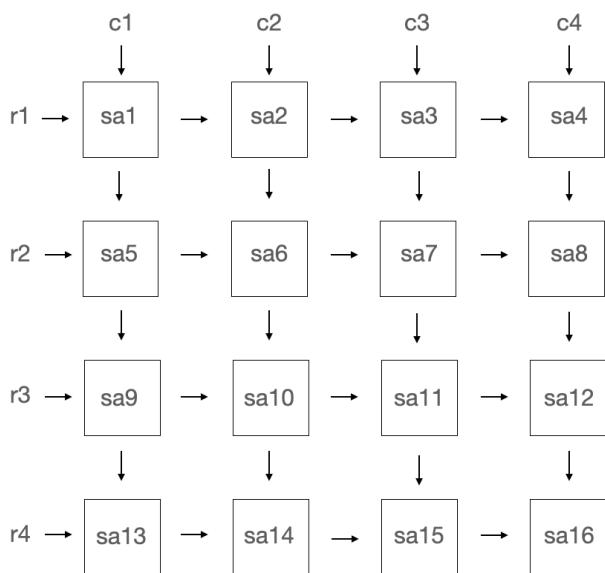
## 1. Design Concept

此次的 design 以 finite state machine 為主軸，利用不同的 state 去拿取對應的 input, weight，將它們都存放在各自的 buffer 中，並且 buffer 在每個 cycle，都會輸出值作為 systolic array 的 8 個 input (buffer 就像是 queue)。Systolic array 運作的模式為 output stationary，當計算完一整組的 output 時，會有另外一個 opw\_state，來處理把結果存放到 output\_sram 中 (計算仍然進行)，這樣重複的循環直到所有計算完成。底下會針對每個部分有更詳細的介紹。

### Systolic array

4 \* 4 的 systolic array 我是一起寫成一個 module，雖然很繁瑣，但其實每一個 array 都是做同樣的動作，如下面的附圖，就是由左側和上方傳進來的數值相乘後，與原本存在 array 的數值做相加，然後再將它們往下/右邊傳。另外會有一個 control signal -> first round，用來對 array 裡面的 output 進行 reset 的動作。

右下方為其中一個 systolic array 的 code

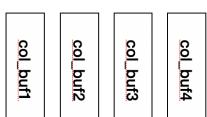


```

// sa9
next_sa9_r = r3;
next_sa9_c = sa5_c;
if (first_round == 6'd3) begin
  next_sa9 = r3 * sa5_c;
end
else begin
  next_sa9 = sa9 + r3 * sa5_c;
end
  
```

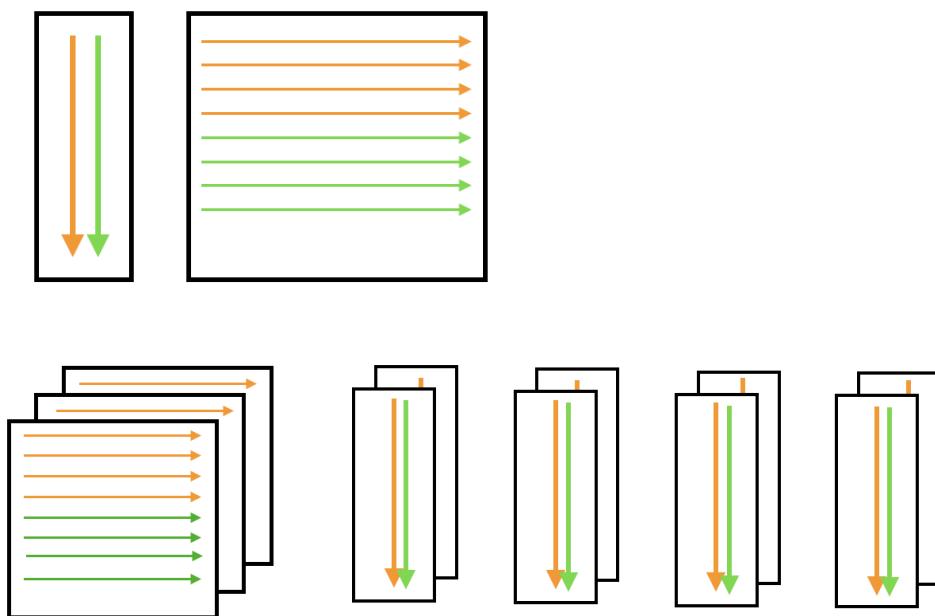
### row/col buffer

每一個 buffer 都是 32 個 bit，與讀進來的 data 大小相同，row buffer 是用來存放 input data，column buffer 是用來存放 weight data，他們就如同 queue 一樣，每個 cycle 就將自己前 8 個 bit ([7:0]) 傳進去 systolic array 的 r1-r4, c1-c4，然後做一個 shift 的動作 (後面就補 0)，當 buffer 沒有數值的時候，下一個 cycle 就會從 input\_rdata / weight\_rdata 得到新的數值，如此周而復始地進行下去。(buffer & systolic array 的關係圖)



**data 讀取的順序**

由於我的 systolic array 是採用 output stationary, 會一次把某個 output data 計算完成後, 才將它放到 output\_sram 中, 因此不會有從 sram 拿進拿出, 進行修改的動作, 省去 implement 的困難和複雜。所以我的讀法會從 input 前 4 個 row 和 weight 前 4 個 col (fc) / 前 4 組 kernel (conv) 開始, 同一個 buffer 存放的是計算同一個 output 的 data, 計算完後, weight 回到初始狀態, 然後 row 再往下 4 個, 或者如果 input 已經到最下面了, 則就是換下 4 個 kernel, 然後 input 重頭開始。另外因為在 fully connected 中, weight 只有一排, 然而為了維持 systolic 運作的一致, col buffer 所存放的值都會是一樣的, 亦即同一個 weight address access 四次, 並且只看 1, 6, 11, 16 的值。下方有一個簡易的示意圖, 橘色的線為當今運算所 access data 的順序, 綠色的線為下一回合的計算所 access data 的順序。(下一為 fc, 下二為 conv)

**opw\_state (output write state)**

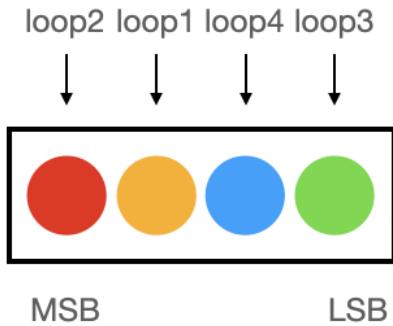
這個 state 是用來管控制寫 output data 的狀態, 當沒有要進行寫的動作, 會停留在 idle, 當偵測到完成一組 output 的計算時, opw\_state 就會進入 clk1 - clk7 的 state, 依序 (clk1: 1 / clk2: 2, 5 / clk3: 3, 6, 9 / ..... ) 在正確的 cycle 將正確的 systolic array output 讀取並存放起來, 當全部的 output 都讀取完成後, 就會進入傳送 data 到 output sram 的動作。因為一次只能傳一個值, 所以這邊會進入一個 send1 - send16 總共 16 個 state 來傳完所有的結果 (如果是 fc 就只要 4 個 state), 然後再回到 idle 待命。因此在此次的作業中, 計算一個 output 的結果都超過 23 個 cycle, 因此傳送 output data 和計算的行為可以同時進行, 不會去干擾到。

**finite state machine & 主架構**

state 大致分成三個 part, 第一個 part 是 initialize, 第二個 part 是 loop, 第三個 part 是 finish。由於 sram 讀取的方式是一次拿到 4 個值, 我們的 systolic array 又剛好是  $4 \times 4$  的形式, 因此可以做成一個循環, 利用 row(col) 與 row(col) 之間差一個 cycle, 不斷地由 row(col) buffer 1-4 這樣讀取數值進來。

initialize 的部分除了將變數 reset 外, 還會先進行頭 4 次的 access, 讓 8 個 buffer 裡面都有數值, 接著就會進入 loop 的階段。

loop 的部分就是 loop1, loop2, loop3, loop4 4 個 state 不斷循環, 每個 loop 就是將 input / weight address 設為相對應 buffer 要拿去的值 (1-> row\_buf1 & col\_buf1), 然後在下一個 state 將值放入 buffer 中 (loop2 -> 將 r\_data 放入 buf1 中)。底下有個 row\_buf1 與 loop state 對應的示意圖。



當計算完所有的 output 後，就會進入 finish 的部分，因為計算完結果後，仍要等待把它們從 systolic array 中拿出來，並依序放進 sram 中，因此會先在 wait\_to\_finish 中等待，等到所有的數值都傳至 output sram 後，整個計算才算真正的結束，這時候就會進入到最後的 finish state，並且把 Ready, Wait 的數值拉起來。

## 2. Simulation and Discussion

### simulation result (conv)

```
[Success] golden[ 1590]= 1570 | data[ 1590]= 1570
[Success] golden[ 1591]= -522 | data[ 1591]= -522
[Success] golden[ 1592]= -11636 | data[ 1592]= -11636
[Success] golden[ 1593]= -1039 | data[ 1593]= -1039
[Success] golden[ 1594]= -2674 | data[ 1594]= -2674
[Success] golden[ 1595]= -12592 | data[ 1595]= -12592
[Success] golden[ 1596]= -12643 | data[ 1596]= -12643
[Congratulation!] all value are correct!
[Time usage] 673130 ns
Simulation complete via $finish(1) at time 677220 NS + 0
./testbench.v:217      $finish;
ncsim> exit
```

### synthesis result (conv)

```
[Success] golden[ 1592]= 1570 | data[ 1592]= 1570
[Success] golden[ 1593]= 4031 | data[ 1593]= 4031
[Success] golden[ 1594]= -522 | data[ 1594]= -522
[Success] golden[ 1595]= -11636 | data[ 1595]= -11636
[Success] golden[ 1596]= -1039 | data[ 1596]= -1039
[Success] golden[ 1597]= -2674 | data[ 1597]= -2674
[Success] golden[ 1598]= -12592 | data[ 1598]= -12592
[Success] golden[ 1599]= -12643 | data[ 1599]= -12643
[Congratulation!] all value are correct!
[Time usage] 673130 ns
Simulation complete via $finish(1) at time 677220220 PS + 0
./testbench.v:217      $finish;
ncsim> exit
```

**simulation result (fc)**

```
[Success] golden[    75]=      -2980 | data[    75]=      -2980
[Success] golden[    76]=      -2750 | data[    76]=      -2750
[Success] golden[    77]=     -1344 | data[    77]=     -1344
[Success] golden[    78]=      7660 | data[    78]=      7660
[Success] golden[    79]=      6788 | data[    79]=      6788
[Success] golden[    80]=     -2016 | data[    80]=     -2016
[Success] golden[    81]=     11665 | data[    81]=     11665
[Success] golden[    82]=      7611 | data[    82]=      7611
[Success] golden[    83]=      -479 | data[    83]=      -479
```

[Congratulation!] all value are correct!

[Time usage] 101450 ns

Simulation complete via \$finish(1) at time 105540 NS + 0

./testbench.v:217 \$finish;

ncsim> exit

**synthesis result (fc)**

```
[Success] golden[    77]=     -1344 | data[    77]=     -1344
[Success] golden[    78]=      7660 | data[    78]=      7660
[Success] golden[    79]=      6788 | data[    79]=      6788
[Success] golden[    80]=     -2016 | data[    80]=     -2016
[Success] golden[    81]=     11665 | data[    81]=     11665
[Success] golden[    82]=      7611 | data[    82]=      7611
[Success] golden[    83]=      -479 | data[    83]=      -479
```

[Congratulation!] all value are correct!

[Time usage] 101450 ns

Simulation complete via \$finish(1) at time 105540310 PS + 0

./testbench.v:217 \$finish;

ncsim> exit

**Discussion**

## 1. Critical path

data arrival time	-----	-----	-----	-----	3.70
clock clk (rise edge)			40.00	40.00	
clock network delay (ideal)			0.00	40.00	
systolic_array/sai_reg[31]/CK (DFFRHQX1)			0.00	40.00	r
library setup time			-0.07	39.93	
data required time				39.93	
-----					
data required time				39.93	
data arrival time				-3.70	
-----					
slack (MET)				36.23	

圖中顯示的 required time 是 39.93, 也就是 critical path 所需要的傳輸時間, 因為它小於一個 clock cycle 40ns, 表示我的 design 是符合此次作業要求的頻率, 不會因為 delay 而產生數值亂掉的問題發生。

## 2. Area

Number of ports:	5179
Number of nets:	17988
Number of cells:	11069
Number of combinational cells:	9404
Number of sequential cells:	1592
Number of macros/black boxes:	0
Number of buf/inv:	1381
Number of references:	93
Combinational area:	23909.562533
Buf/Inv area:	1186.398026
Noncombinational area:	9801.036219
Macro/Black Box area:	0.000000
Net Interconnect area:	undefined (No wire load)
Total cell area:	33710.598752
Total area:	undefined

\*\*\*\*\* End Of Report \*\*\*\*\*

依照老師上課講義給的公式，換算出來我的 design 大概會使用 6623 個 gate，其中 Combinational area 佔了 4697，non-Combinational 佔了 1925，算是蠻合理的，因為 code 主體都是乘加器和邏輯判斷，只有很小塊是用來做 sequential block。另外有跟其他同學稍微比較一下，大致上用的差不多。

### 3. 單個 MAC vs 16 個 MAC

FC 計算量 (單個 MAC):  $84 * 120 \rightarrow 10080$  cycle  $\rightarrow 403200$ ns

FC 計算量 (16 個 MAC): 101450ns

理論上來說應該要是 4 倍，因為 input 只有一排，同時間只能產出 4 個 output，可是實際上來看，因為實作上有一些 initialize 與最後傳送 output 值所要等待的 cycle，所以結果會比 4 倍在小一點。

Conv 計算量 (單個 MAC):  $25 * 100 * 6 * 16 \rightarrow 240000$  cycle  $\rightarrow 9600000$ ns

Conv 計算量 (16 個 MAC): 673130ns

理論上來說應該要是 16 倍，因為同時間可以產出 16 個 output，可是實際上來看，與上述原因相同，所以結果會比 16 倍在小一點。

### 4. Unfolded problem

如果讀進來的 data 沒有經過 unfolded 的話，我們可以自己寫一個 unfold 的 module 來處理，與 python code 的寫法類似，先 create 好最後的 result 的陣列大小，然後利用 for loop 展開，去賦予每個 index 對應的值，這樣也能達到 unfolded data 的效果。

### 3. Summary

lab3 將 machine learning 上常見的 convolution 和 fully connected 的運算，利用  $4 * 4$  systolic array 實作出來。原本都是使用很 high-level language 去做運算，很難利用硬體來加速，但是我們這次使用 verilog 來 implement，也針對自己設計出來電路的 area & timing 做探討，看看是否會超過所設計的 40ns cycle，同時觀察是否能達成理想上 16 倍的加速效果。以最後分析的結果來看，能很接近理想中的倍數，但幾乎不可能可以到達 16 倍，因為我們的 design 不可能每一個 cycle 都拿來做運算，一定會存在一些 cycle 是要拿來做 initialization, data movement 的處理，所以效果會稍微差一點，不過我認為如果計算的 matrix 大小再更大，加速的效果幾乎就可以視為 16 倍了。

遇到的困難 & 心得：此次參考老師的做法，先把 design 想好，再將它們用 code 實作出來，因此花費了相當多的時間在設計 design 上面，確保是可以 work 的，才開始著手寫 code。然而很遺憾地，一開始在 simulation 上還是出現了 error，debug 後才發現自己在邊界判斷上面寫錯了變數，將 input 進來的 width 搞錯，處理完這個之後，simulation 和 synthesis 就都通過了，整理來說算是蠻順利的，雖然途中遇到一些小困難，但都沒有花上太多時間處理。很感謝這次有跟許多同學討論交

流想法，也有不斷地把想法畫出來，才能在初始的 design 就是正確的。不過還是最感謝助教和老師延長一個禮拜的 deadline！