人工智慧晶片設計與應用

AI-ON-CHIP FOR MACHINE LEARNING AND INFERENCE

Final Project

M16111048　湯詠涵

N26110794　蘇亦豪

N26102157　吳孟玲

目錄

1. 專案介紹

1.1 專案目標

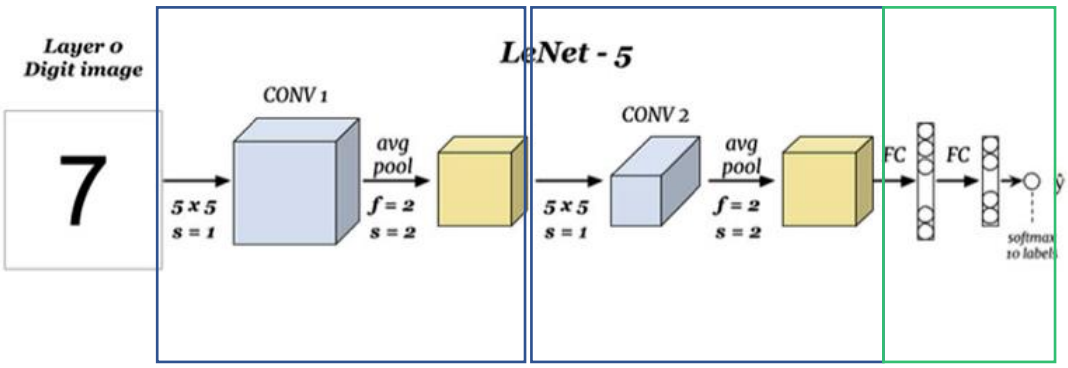
本次專案欲實踐 LeNet 加速器，設計方面分成軟體和硬體設計會在後方進行介紹。在軟體方面，我們希望可以去除 Fully-Connercted 的使用，以 Convolution 替代，減少設計 PE 負擔，並在保持良好準確的情況下，增加效率，且能夠實作於晶片領域上；在硬體方面，我們希望達成增加彈性、減少能量消耗並且讓 PE 使用率盡量提高。

1.2 專案實踐成果

2. 軟體設計

2.1 軟體基本架構

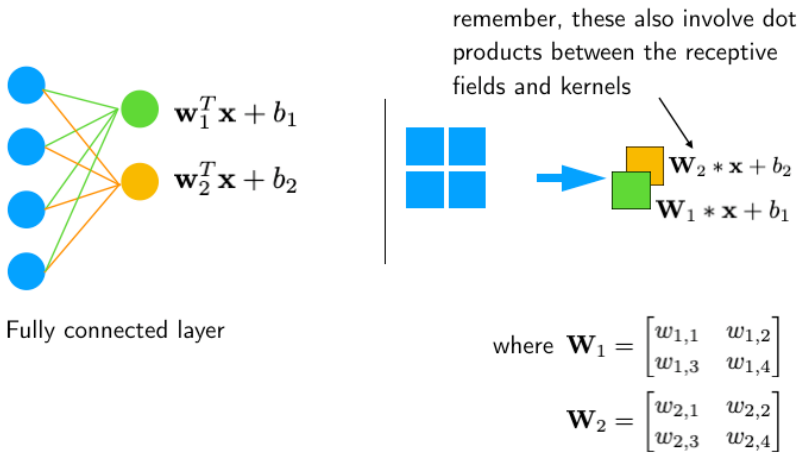
本次模型架構所選用的是 LeNet，其原始架構如下圖：



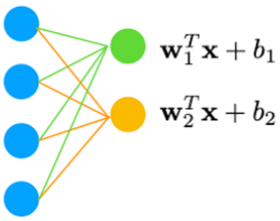
2.2 軟體架構修改

將 Convolution layer 替代 Fully connected layer，以期減少 PE 負擔。

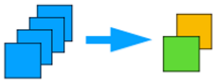
A. Kernels equal to the input size



B. Convolution with 1x1 kernels

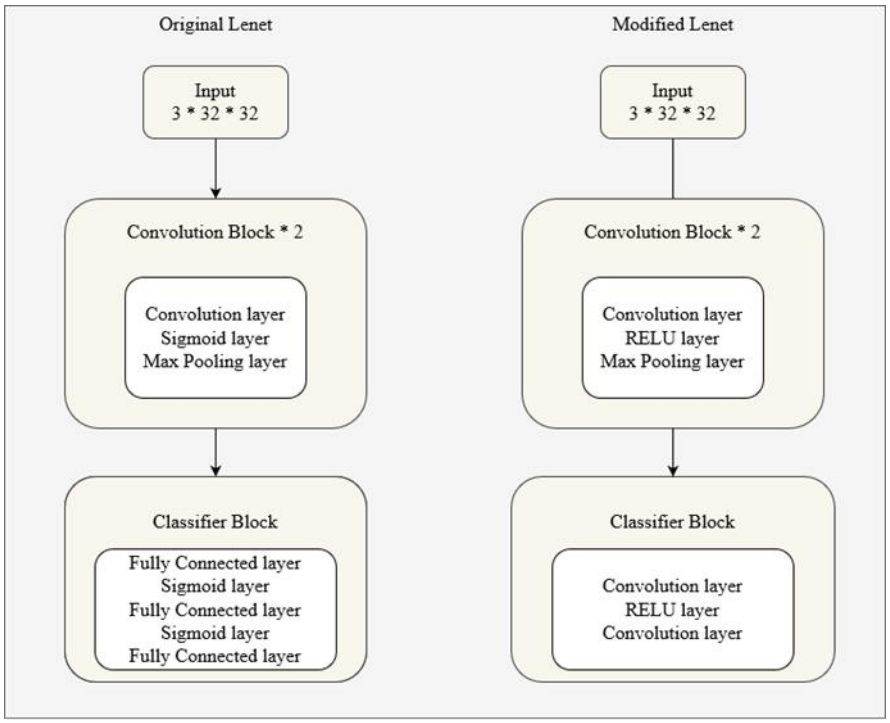


Fully connected layer



Or, we can concatenate the inputs into 1x1 images with 4 channels and then use 2 kernels (remember, each kernel then also has 4 channels)

2.3 更改後的架構



Layer	Ifmap shape (rowxcolxch)	Filter Shape (rowxcolxch) x set	Output Shape (rowxcolxch)	padding
0	32x32x1	(5x5x16) x 1	28x28x16	0
Max Pooling : Output Shape 14*14*16, kernel=2*2, stride = 2				0
1	28x28x16	(5x5x32) x 16	10x10x32	0
Max Pooling : Output Shape 5*5*32, kernel=2*2, stride = 2				0
2	10x10x32	(3x3x32) x 32	3x3x32	0
3	3x3x32	(3x3x10) x 32	1x1x10	0

2.4 實驗比較

A. 不同 Optimizer，可以發現 Adam 表現最佳。

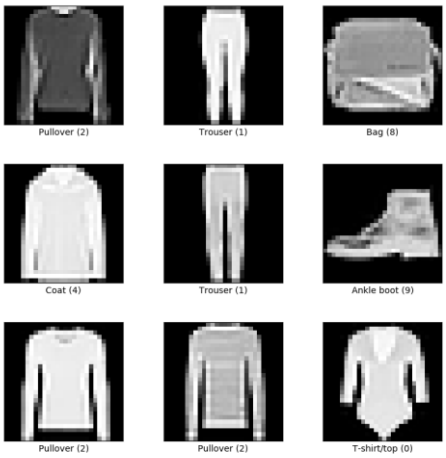
Optimizer / Structure	**Adam**	RMSprop	SGD
Origin	**86.53%**	86.01%	70.79%
Modified – 1	**90.15%**	89.20%	89.12%
Modified – 2	**88.30%**	87.89%	88.30%

B. 不同 epochs 數

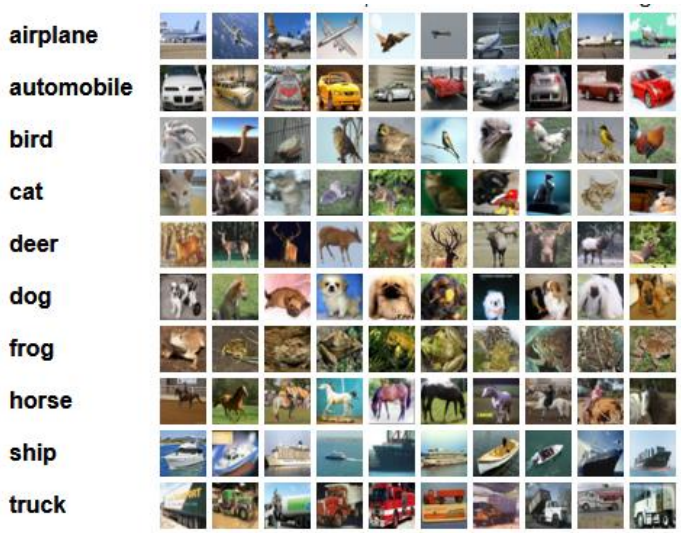
Structure / Epochs	Origin	Modified – 1	Modified – 2
20	85.24%	83.77%	87.66%
30	86.53%	**90.15%**	88.30%
40	87.83%	88.41%	88.39%

2.5 資料集

A. Mnist Fashion



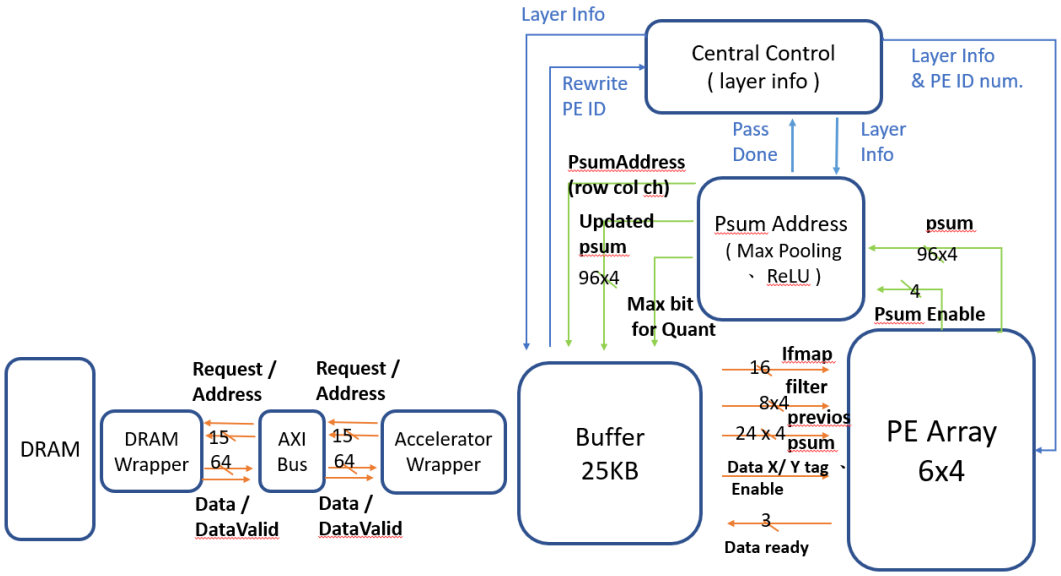
B. Cifar10


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

3. 硬體設計

3.1 硬體基本架構

Finish Layer 1 & Layer 2
DRAM access time : 130 ns
Clock cycle : 10ns
Hybrid Ofmap bandwidth : 8 - 4 - 4 - 4 - 4
Filter 8 bit
Psum 24bit



Buffer 會不斷的輸出資料給 PE Array，Layer0 時，由於 ifmap 和 filter 的 channel 為 1，ifmap 和 filter 都是一次傳輸 1 筆 8bit 的資料，Layer1 時，ifmap 被 Quant 回 4bit，ifmap 和 filter 都是一次傳輸 4 筆不同 channel 的資料。Buffer 偵測到即將接收下一筆資料的 PE 都是 ready 的，會把資料傳出去，enable 拉成 1，並且附上 X、Y tag，接著準備下一筆資料。

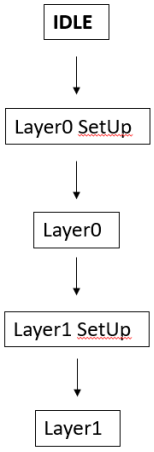
PE Array 中的 Multicast Cast Controller 會有自己的 ID，如果 tag=ID，ifmap、previous psum、filter 就會進入特定的 PE。

PE 將資料運算完後，把 Psum 傳回 Psum Address，在這邊 Psum Address Module 會接收到 PE00、01、02、03 所輸出的 96bit psum 和 psum enable 。在 Layer0 時，Psum Address 接收到的資料都會做 MaxPooling 和 ReLU，計算出此筆 psum 的 row column channel，存回 buffer 的 L1_I_MEM[row][column][channel]；在 Layer1，則是在最後一輪 Processing Pass(Pass3-0 到 3-7)，才會做 MaxPooling 和 ReLU，其他的 Processing Pass(Pass0-0 到 2-7)都會直接將接收到的 psum，存回 buffer 的 L1_pre_psum。另外，Psum Address 會找出此層 layer 最大的數值，並且將這個最大數值的 bit 數，傳給 Buffer。以及利用算出來的 row column channel，得知此輪的 Processing Pass 是否結束。

如果此輪的 procssing pass 已經結束 ，Central Control 會將 Processing Pass 更新，如果此層 layer 的 Processing Pass 已經都完成了，Central Control 就會發放下一層 layer 的資訊。
DRAM 在此雖然可以成功傳輸資料，但是由於時間的因素來不及完成，DRAM 把 image 和 filter 傳給 Buffer，所以一開始資料都是由 Buffer 直接讀進來的。
DRAM 會藉由 AXI 和加速器溝通，在 130ns 可以傳送一筆 64bit 的資料。
此加速器一次可以處理一張輸入照片，batch_size=1

3.2 Central Control

Central Control 會發放以下的資訊在 State SetUp，進入 SetUp 後的下一個 clk，就會進入 Layer。
Input Output Layer Shape
PE & Buffer related : Filter set, Channel set, Sliding window size, ifmap bandwidth, number of row of psum PE need to compute
Schedule : Total Filter set 、Total PassX PassY
Post Processing : Maxpooling(Layer 1& 2)、ReLU(Every layer)、Class(Last Layer)



3.3 Processing Pass for the First Layer

以下是產出第一層的流程，ifmap 輸入是 32*32*1 的黑白照片，Filter 是 5*5*1*16，做完 convolution、MaxPooling、ReLU 會得到 14*14*16 的 ofmap。
Channel 為 1，C_total=0，Filter 共有 16 組，每一組都是 5*5*1，M_total=16。

```
for(c1=0; c1< (C_total/4; c1++) begin
 previous psum = Psum [x][y][M_total]
 for(c0=0; c0<4; c0++) begin
  for(m1=m0; m1< (M_total /4 ); m1++) begin
  for(m0=0; m0<4; m0++) begin
   for(x=0; x <= 31; E++) begin
    for(y=0; y <= 31; F++) begin
     for(R=0; R<=4; R++) begin
      for(S=0; S<=4; S++) begin
        E= x-R; (x>= R)
        F =y-S; (y>= S)
        M = m1 * 4 + m0;
        C = c1 * 4 + c0;
        OA [E][F][M] = previous psum [x][y][M] + IA[x][y][C]*Filter[R][S][C][M] ;
```

Data of each Pass		Pass0-0	Pass0-1	Pass0-2	Pass0-3
Filters	Filter IDs	0-3	4-7	8-11	12-15
	Channel IDs	0	0	0	0
Ifmaps	ifmap IDs	0	0	0	0
	Channel IDs	0	0	0	0
Psums	ofmap IDs	0	0	0	0
	Channel IDs	0-3	4-7	8-11	12-15

3.4 Processing Pass for Second Layer

以下是產出第 2 層的流程，ifmap 輸入是 14*14*16，Filter 是 5*5*16*32 完 convolution、MaxPooling、ReLU 會得到 5*5*32ofmap。

Channel 為 16，C_total=14，Filter 共有 32 組，每一組都是 5*5*16，M_total=32。

Set Previous Psum to 24’d0 for First PassX

PassX-Y 會加上 Pass(X-1) - Y 所產出的 previous psum

Do maxpooling and ReLU while Last PassX

```
for(c1=0; c1< (C_total/4; c1++) begin
 previous psum = Psum [x][y][M_total]
 for(c0=0; c0<4; c0++) begin
  for(m1=m0; m1< (M_total /4 ); m1++) begin
  for(m0=0; m0<4; m0++) begin
   for(x=0; x <= 13; E++) begin
    for(y=0; y <= 13; F++) begin
     for(R=0; R<=4; R++) begin
      for(S=0; S<=4; S++) begin
        E= x-R; (x>= R)
        F =y-S; (y>= S)
        M = m1 * 4 + m0;
        C = c1 * 4 + c0;
        OA [E][F][M] = previous psum [x][y][M] + IA[x][y][C]*Filter[R][S][C][M] ;
```

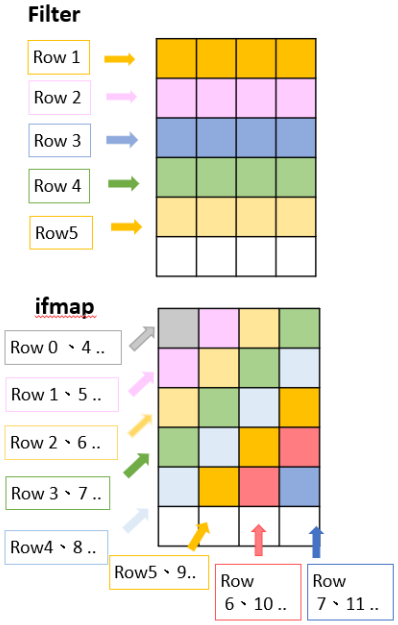
Data of each Pass		Pass0-0 Pass1-0 Pass2-0 Pass3-0	Pass0-1 Pass1-1 Pass2-1 Pass3-1	Pass0-2 Pass1-2 Pass2-2 Pass3-2	Pass0-3 Pass1-3 Pass2-3 Pass3-3	Pass0-4 Pass1-4 Pass2-4 Pass3-4	Pass0-5 Pass1-5 Pass2-5 Pass3-5	Pass0-6 Pass1-6 Pass2-6 Pass3-6	Pass0-7 Pass1-7 Pass2-7 Pass3-7
Filters	Filter IDs	0-3	4-7	8-11	12-15	16-19	20-23	24-27	28-31
	Channel IDs	0-3 4-7 8-11 12-15	0-3 4-7 8-11 12-15	0-3 4-7 8-11 12-15	0-3 4-7 8-11 12-15	0-3 4-7 8-11 12-15	0-3 4-7 8-11 12-15	0-3 4-7 8-11 12-15	0-3 4-7 8-11 12-15
Ifmap	Channel IDs	0-3 4-7 8-11 12-15	0-3 4-7 8-11 12-15	0-3 4-7 8-11 12-15	0-3 4-7 8-11 12-15	0-3 4-7 8-11 12-15	0-3 4-7 8-11 12-15	0-3 4-7 8-11 12-15	0-3 4-7 8-11 12-15
Psum	Channel IDs	0-3	4-7	8-11	12-15	16-19	20-23	24-27	28-31

3.5 6x4 PE array

以下將 PE 做編號。PE 的相對位置可以對應到下面的左圖。

PE00	**PE01**	**PE02**	**PE03**
PE10	**PE11**	**PE12**	**PE13**
PE20	**PE21**	**PE22**	**PE23**
PE30	**PE31**	**PE32**	**PE33**
PE40	**PE41**	**PE42**	**PE43**
PE50	**PE51**	**PE52**	**PE53**

以下的左圖相同顏色的 PE 會接收到相同 row 的 filter 或 ifmap。
Filter shape 為 5*5。因此不會用到 PE50、51、52、53。



在這邊 PE Array 設計成 6x4，高度是 6，是因為要實作的 filter shape 是 5*5 及 3*3，當 filter shape 為 5*5，PE00、01、02、03 可以接收到 filter row1 的資料，PE10、11、12、13 可以接收到 filter row2 的資料，…以此類推，PE50、51、52、53 則是 unused PE，如左圖所示；當 filter shape 為 3*3，PE00、01、02、03、PE30、31、32、33 可以接收到 filter row1 的資料，PE10、11、12、13、PE40、41、42、43 可以接收到 filter row2 的資料，PE20、21、22、23、PE50、51、52、53 可以接收到 filter row3 的資料。

寬度設計為 4，除了 4 是偶數，讓 Psum Address Module 在 MaxPooling 的處理上相對容易，也是因為 layer0 的輸入有 32 個 row，且 layer2、3 的 Ifmap row 分別只有 5、3 個，避免掉過多的 unused PE。

由於 PE 寬度為 4，因此會重複發放 ifmap 的 row，如左圖所示 PE00 會接收到 ifmap 的 row0、4、8、12 以此類推。

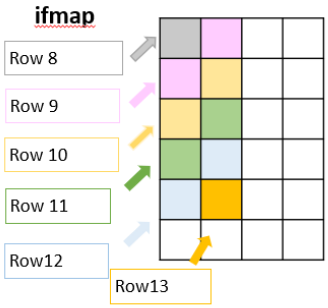
3.6 Ifmap Delivery

LAYER	Ifmap	Stage0	Stage1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	active PE rate
0	Row	0-7	4-11	8-15	12-19	16-23	20-27	24-31	$\frac{5}{6}$
1	Row	0-7	4-11	8-13					$\frac{20+20+10}{24+24+24}$
2	Row	0-4							$\frac{3}{4}$
3	Row	0-3							$\frac{1}{4}$

以上的表格是每一層 PE 接收到的 ifmap row。

在 layer 0 stage0，PE00 會接收到 row0，PE01、PE10 會接收到 row1，PE20、11、02 會接收到 row2…. PE43 會接收到 row7，在 stage1，PE00 會接收到 row4，PE01、PE10 會接收到 row5，PE20、11、02 會接收到 row6…. PE43 會接收到 row11。

在 layer 1 stage2，ifmap row 接收情形如左圖所示，ifmap ID 則如右圖所示，只會用到 10 個 PE。Layer 1 stage0、1 都會用到 20 個 PE。Buffer 在發送完 stage1 的 ifmap 會通知 Central Control 去 rewrite PE 的 ID。



Unused PE :
Layer1 Stage 2

Y ID	X ID			
0	0	1	7	7
0	1	2	7	7
0	2	3	7	7
0	3	4	7	7
0	4	5	7	7
0	7	7	7	7

Layer 0、layer2、layer3 的 ifmap ID 如下圖所示。標黃底的是不會用到的 PE。Layer 2 、3 分為兩組 3*4 的 PE(藍色及綠色)，接收不同 Set 的 filter，接收相同的 ifmap。

Layer0

Y ID	X ID			
0	0	1	2	3
0	1	2	3	4
0	2	3	4	5
0	3	4	5	6
0	4	5	6	7
0	8	8	8	8

Layer2

Y ID	X ID			
0	0	1	2	7
0	1	2	3	7
0	2	3	4	7
0	0	1	2	7
0	1	2	3	7
0	2	3	4	7

Layer3

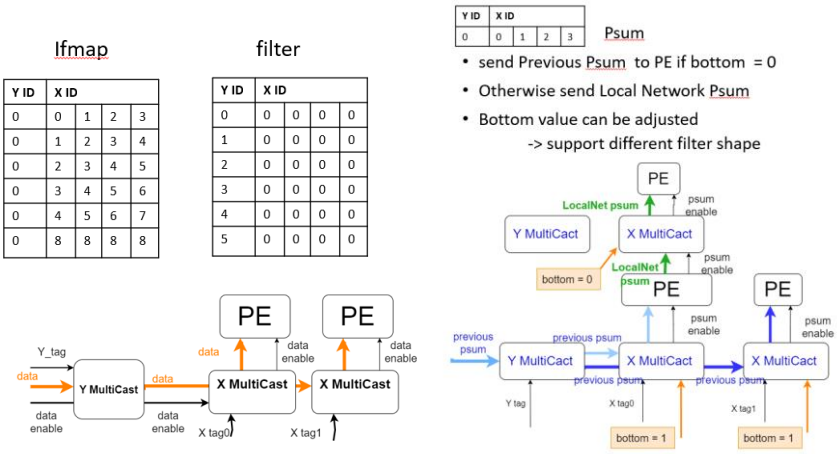
Y ID	X ID			
0	0	7	7	7
0	1	7	7	7
0	2	7	7	7
0	0	7	7	7
0	1	7	7	7
0	2	7	7	7

PE00	**PE01**	**PE02**	**PE03**
PE10	**PE11**	**PE12**	**PE13**
PE20	**PE21**	**PE22**	**PE23**
PE30	**PE31**	**PE32**	**PE33**
PE40	**PE41**	**PE42**	**PE43**
PE50	**PE51**	**PE52**	**PE53**

3.7 MultiCast Controller

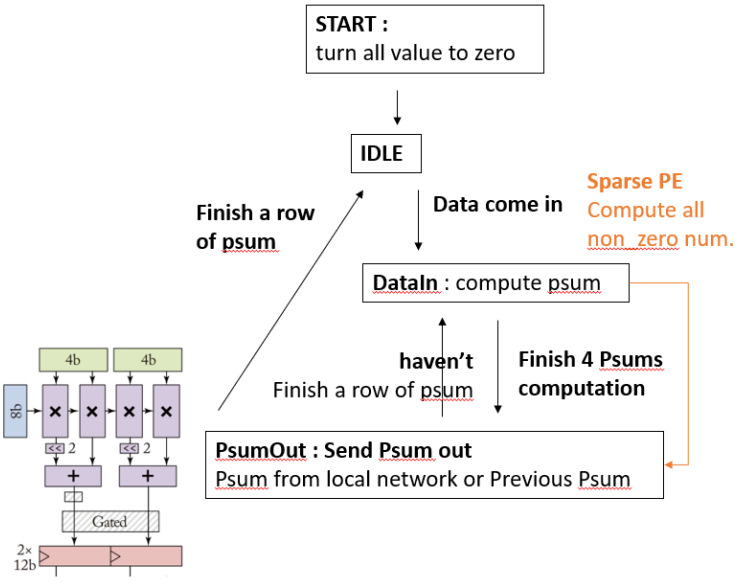
以下是 ifmap 和 filter 在第一層的 ifmap ID 和 filterID，左圖的 data 為 ifmap 和 filter 的 MultiCast Controller 設計，右圖是 psum 的傳送方式。資料從 Buffer 發放到指定 PE 的條件如下:

- **How PE get data: Ready=1 、Enable=1 、(Tag = ID)**
- **Ready : PE is able to receive data**
- **Enable : data is ready**
- **ID : PE address for receiving data**
- **Tag : Buffer sends data with tag**



此外 layer0、1，PE40、41、42、43 所對應的 Psum MultiCast Controller，會將 bottom 設為 1，只能接收 previous psum，其他的 Psum MultiCast Controller bottom 都是 0 接收其他 PE 來的 psum。

3.8 Processing Engine



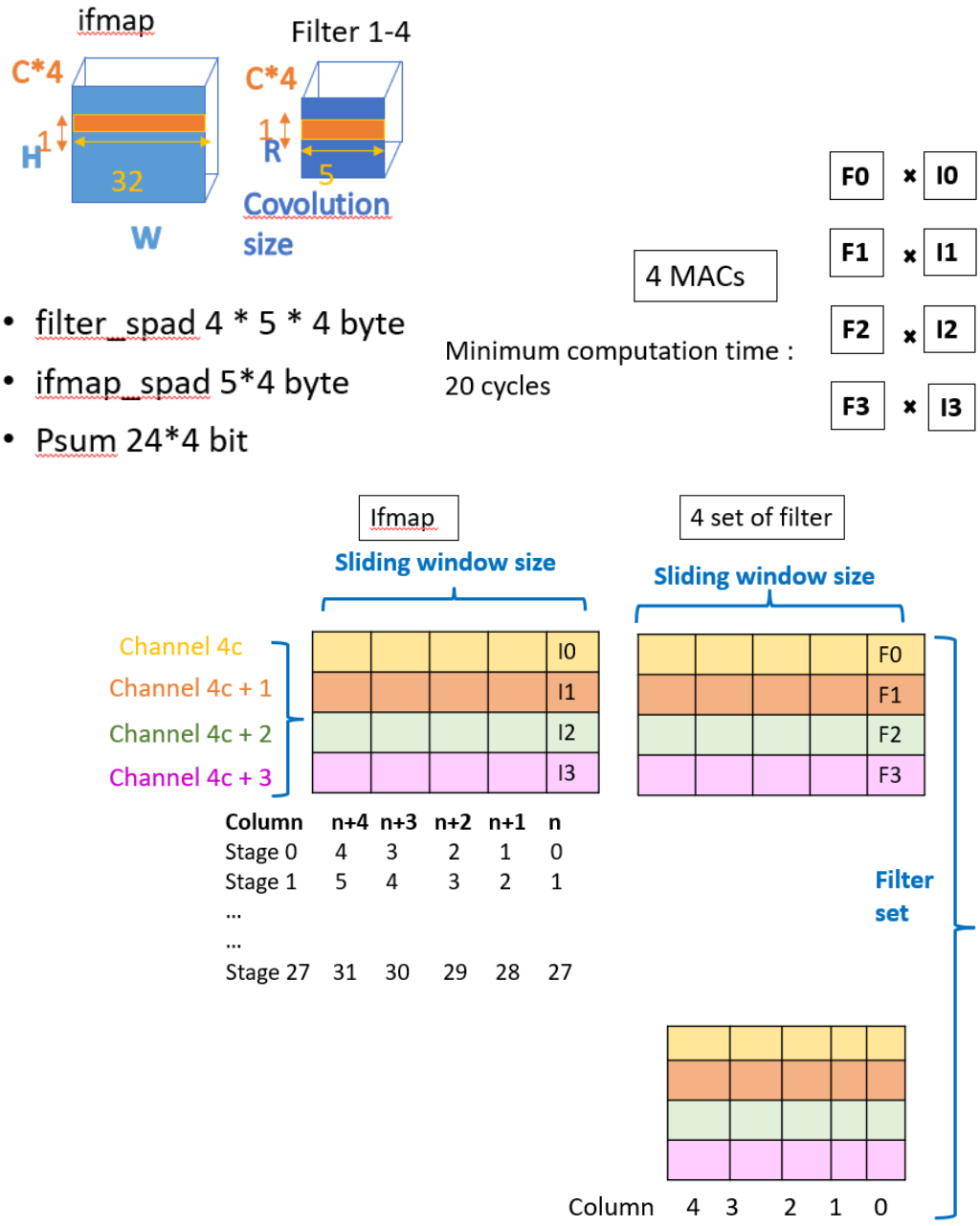
Able to support different Output Shape (range 0-31)、
Filter Set (range 0-3) 、
Channel Set (range 0-3)、
Sliding Window Size (range 0-4)、
Ifmap Bandwidth (8 or 4 bit)

- Able to compute Psum while receiving new data
- Record number of output psum and finished number of row of psum
- MACs perform 8bit x 8bit

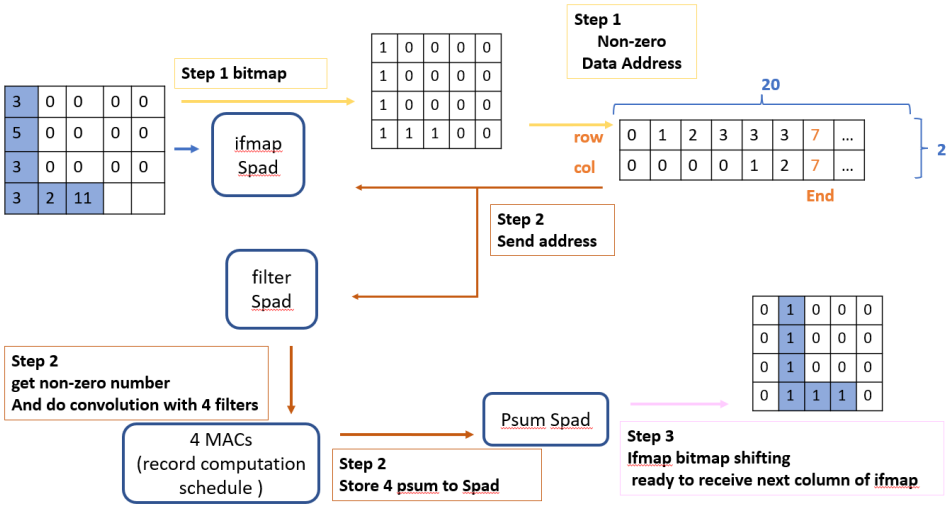
• MACs Do Gated while ifmap[7:4] = 4'b0000，此設計是因為在 layer1 之後的 ifmap 都是 4bit

Sparse PE 只需要將非零的數值做運算即可回到 State DataIn。

3.9 PE



3.10 Sparse PE



3.11 Psum Buffer Address

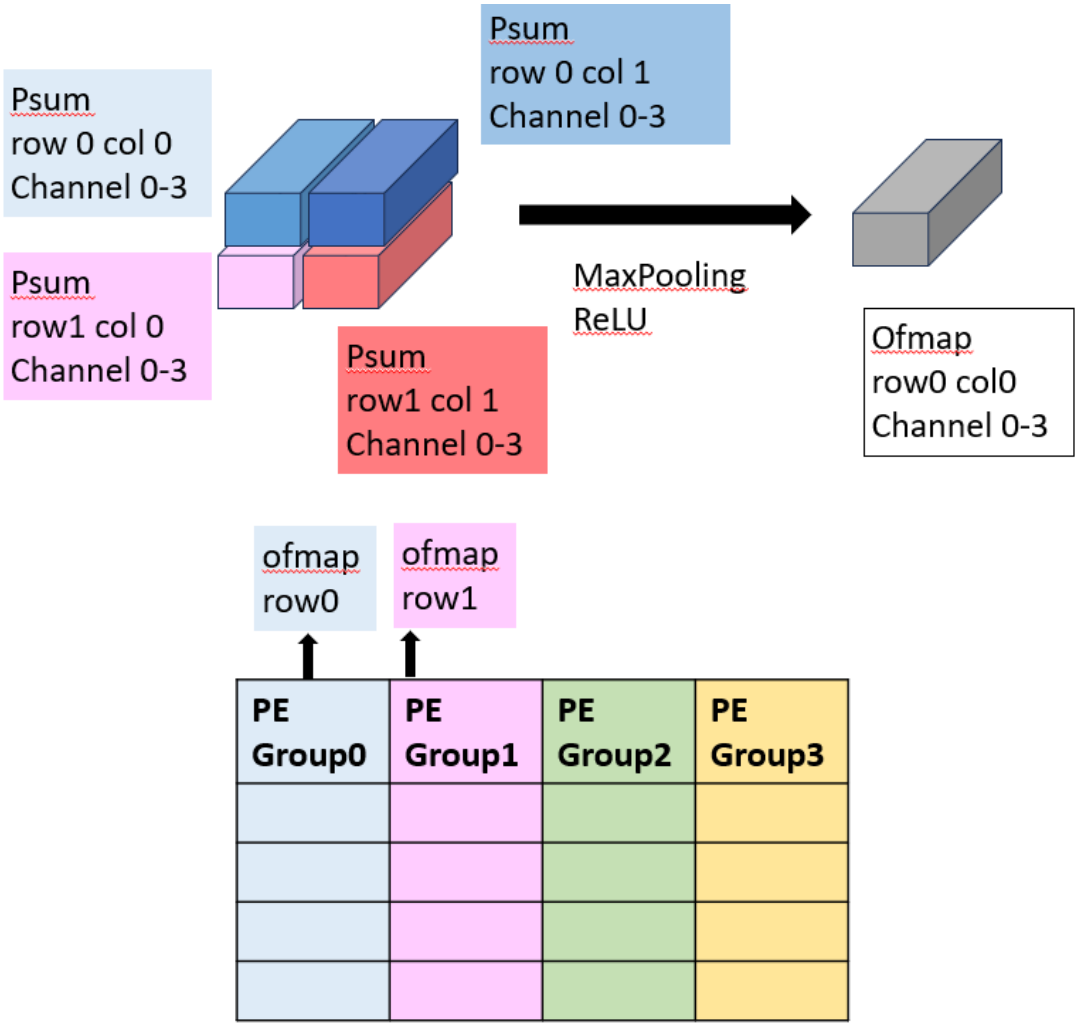
Signal

PassX = PassX total -> Do maxpooling & ReLU

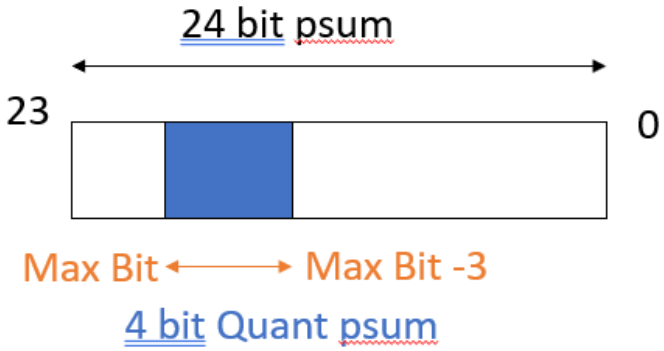
PassY -> compute output channel address

Output Layer Shape -> Finish a Processsing Pass

PE Set	Group	Ofmap	Ofmap Buffer address
PE Set 0	Group 0	Row 0, 4 .... (E-3)	Column : 0 - F **Row : 0, 4 .... (E-3)** Buffer Channel : 0-3、4-7、8-11.....(C-3)-C
	Group 1	Row 1, 5 .... (E-2)	Column : 0 - F **Row : 1, 5 .... (E-2)** Buffer Channel : 0-3、4-7、8-11.....(C-3)-C
	Group 2	Row 2, 6 .... (E-1)	Column : 0 - F **Row : 2, 6 .... (E-1)** Buffer Channel : 0-3、4-7、8-11.....(C-3)-C
	Group 3	Row 3, 7 ....E	Column : 0 - F **Row : 3, 7 ....E** Buffer Channel : 0-3、4-7、8-11.....(C-3)-C


Psum
row 0 col 0
Channel 0-3
Psum
row 0 col 1
Channel 0-3
Psum
row1 col 0
Channel 0-3
Psum
row1 col 1
Channel 0-3
MaxPooling
ReLU
Ofmap
row0 col0
Channel 0-3
ofmap
row0
ofmap
row1
PE
Group0
PE
Group1
PE
Group2
PE
Group3

3.12 Buffer

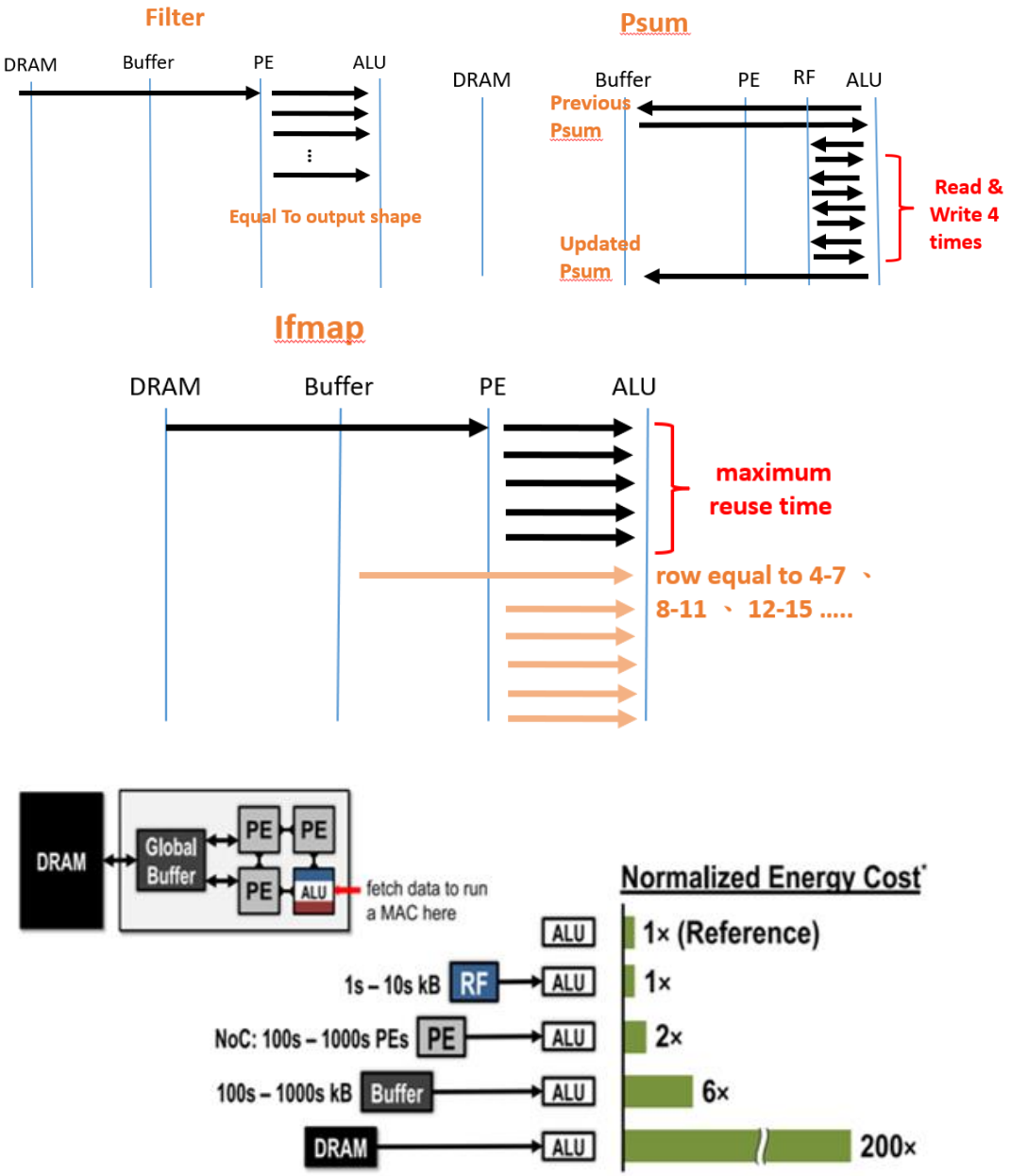


Psum Address module
Find maximum number
Send max bit of a layer to Buffer

e.g. Maximum bit is 15
Choose [15:12] as next layer input

- Total Space : 57.848 K Byte
- 8bit All Filters: 25.296 KB
  (5x5x1)x16
  (5x5x16)x32
  (3x3x32)x32
  (3x3x32)x10
- (8bit) Image: (32x32)x1 --- 1 KB
- (24bit)Previous Psum : 10x10x32 --- 9.6KB
- (24bit)Ofmap : 14x14x32 --- 18.816KB
- (4 bit)Quant Ofmap : 14x14x32 --- 3.136KB

3.13 Estimated Data Movement power



3.14 Analysis

A. PE

- Total cycle : 69949
- Average Active cycle count

(41114+33593+41096+33592+41162+41198+33714+33758+42751+42720+34911+34941+42750+42752+34975+34973+42871+42753+34945+34942) / 20 = 765511/20 = 38275.55

- Utilization rate : (38276/69949) * (5/6) = 0.46
- Average Effectual operation

(33712+33405+26640+28434+33327+35450+28470+29111+35120+36462+28776+2

8436+36735+34838+28615+31077+34723+33157+30470+30968) = 637926 /20 = 31896.3

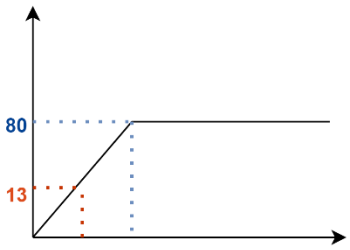
- Average InEffectual operation

(105808+106115+87280+85486+106193+104070+85450+84809+104400+103058+85144+85484+102785+104682+85305+82843+104797+106363+82952+83450)/20 = 1896474/20 = 94823.7

- Effectual rate : 31896.3/ (31896.3+ 94823.7) = 0.252
- Roofline model

  Computation bound: (5*4)*4 = 80 (mul/per cycle)

  Implement: (5*4)*(20/30) = 13.33 (mul/per cycle)



B. Sparse PE

- Sparse PE : time for waiting data to arrive is long
- Total cycle : 58589
- Average Active cycle count

(19519+16109+19765+16522+21285+21926+17866+17948+21918+22266+17832+17935+22342+22008+18037+18748+22504+21361+18793+18598)/20 = 393282/20= 19664.1

- Utilization rate : (19664.1 /58589) * (5/6) = 0.280
- Average Effectual operation

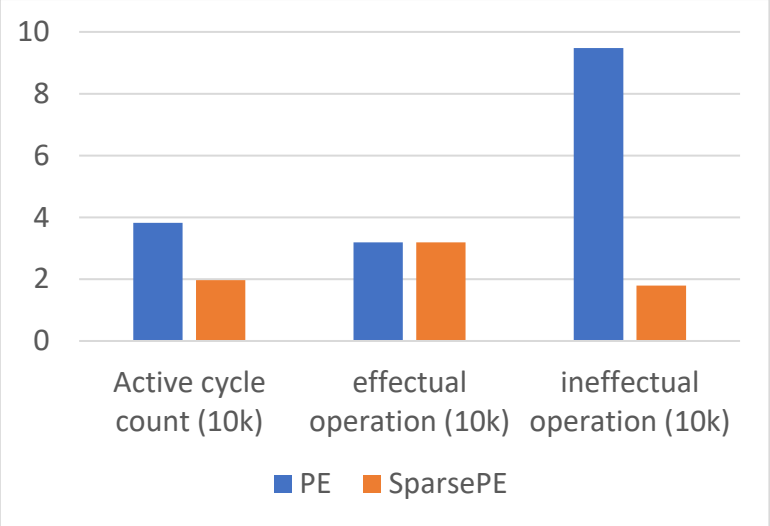
(33712+33405+26640+28434+33327+35450+28470+29111+35120+36462+28776+28436+36462+34838+28615+31077+34723+33157+30470+30968)/20 = 637653/20 = 31882.65

- Average InEffectual operation

(18640+19331+16112+15982+19409+19718+15946+15561+20048+20242+15896+16620+19969+19306+16441+16923+19421+19579+17032+16890)/20 = 359066/20 = 17953.3

- Effectual rate : 31882.65/ (31882.65+ 17953.3) = 0.640

C. Comparison



3.15 Processing Pass for Third & Fourth

```
for(c1=0; c1< (C_total/4; c1++) begin
 previous psum = Psum [x][y][M_total]
 for(c0=0; c0<4; c0++) begin
  for(m1=m0; m1< (M_total /7 ); m1++) begin
Spatial for(m0=0; m0<7; m0++) begin
   Spatial for(h1=0; h1 < H/4; E++) begin
   Spatial for(w1=0; w1 < W/4; F++) begin
    for(R=0; R<=2; R++) begin
     for(S=0; S<=2; S++) begin
       x = h1*4 + 7;
       y = w1*4 + 7;
       E= x-R; (x>= R)
       F =y-S; (y>= S)
       M = m1 * 7 + m0;
       C = c1 * 4 + c0;
       OA [E][F][M] = previous psum [x][y][M] + IA[x][y][C]*Filter[R][S][C][M] ;
```

Data of each Pass		Pass0-0	Pass0-1	Pass0-2	Pass0-3
Filters	Filter IDs	0-7	8-15	16-23	24-31
	Channel IDs	0-3	0-3	0-3	0-3
Ifmaps	ifmap IDs	1	1	1	1
	Channel IDs	0-3	0-3	0-3	0-3
Psums	ofmap IDs	1	1	1	1
	Channel IDs	0-3	4-7	8-11	12-15

3.16 DRAM Design

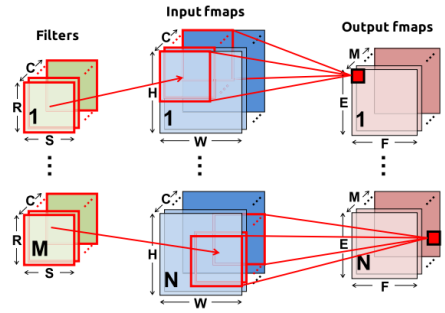


Fig. 1. Computation of a CNN layer.

以下是 PE00 在 Layer0 的 Processing Pass0-0，PE00 從 Buffer 接收到資料，到運算出 4 筆 24bit 的 Psum，送到 Psum Address Module，並做完 ReLU 和 MaxPooling 之後，將 Psum 傳回 Buffer。而後，做 Quant 成為 output feature map 的過程。

Filter:

FRSCM 四個數字分別代表上圖的 RSCM，即 Filter 的 Row Column Channel Set 。

如 F4001 ，代表的是 (R)Row = 4，(S)Column = 0，(C)Channel = 0 ，(M)Set = 1。

Filter Set0

F0000	F0100	F0200	F0300	F0400
F1000	F1100	F1200	F1300	F1400
F2000	F2100	F2200	F2300	F2400
F3000	F3100	F3200	F3300	F3400
F4000	F4100	F4200	F4300	F4400

Filter Set1

F0001	F0101	F0201	F0301	F0401
F1001	F1101	F1201	F1301	F1401
F2001	F2101	F2201	F2301	F2401
F3001	F3101	F3201	F3301	F3401
F4001	F4101	F4201	F4301	F4401

Filter Set2

F0002	F0102	F0202	F0302	F0402

Filter Set3

F1002	F1102	F1202	F1302	F1402
F2002	F2102	F2202	F2302	F2402
F3002	F3102	F3202	F3302	F3402
F4002	F4102	F4202	F4302	F4402

F0003	F0103	F0203	F0303	F0403
F1003	F1103	F1203	F1303	F1403
F2003	F2103	F2203	F2303	F2403
F3003	F3103	F3203	F3303	F3403
F4003	F4103	F4203	F4303	F4403

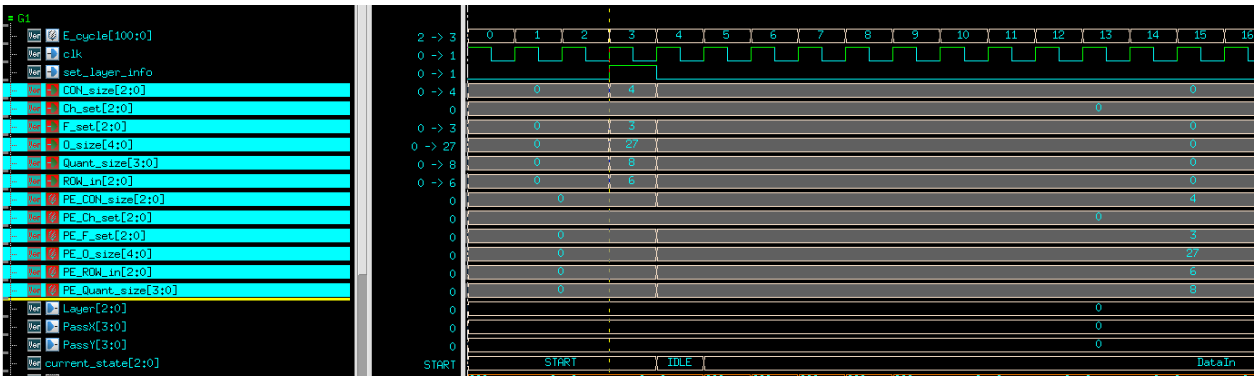
Input Fmap：

FHWC 三個數值分別代表上圖的 HWC，即 image 的 Row、Column、Channel。

如 I31 31 0 ，代表的是 (H)Row = 31，(W)Column = 31，(C)Channel = 0。

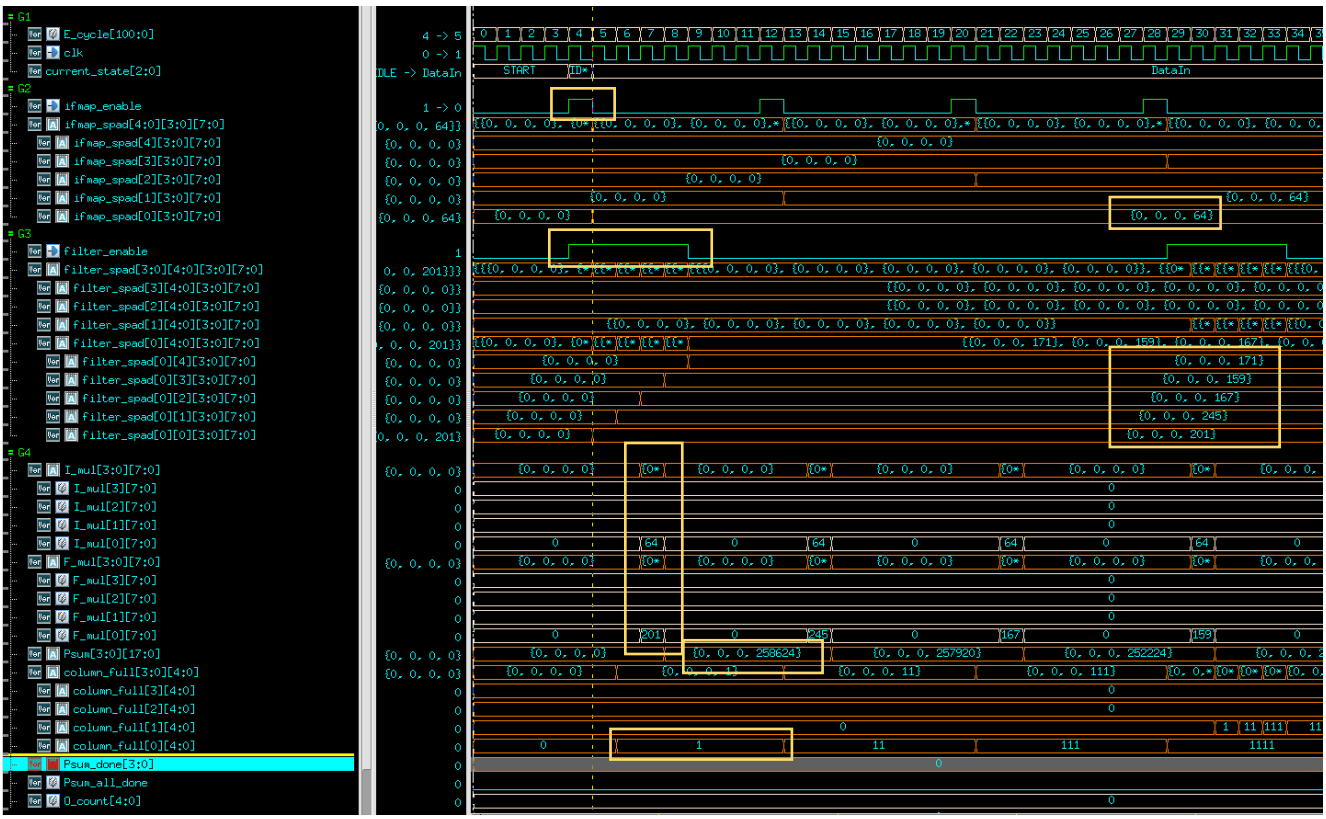
I000	I010	I020	I030	I040	I050	I060	I070	…	I0 31 0
I100	I110	I120	I130	I140	I150	I160	I170	…	I1 31 0
I200	I210	I220	I230	I240	I250	I260	I270	…	I2 31 0
I300	I310	I320	I330	I340	I350	I360	I370	…	I3 31 0
I400	I410	I420	I430	I440	I450	I460	I470	…	I4 31 0
I500	I510	I520	I530	I540	I550	I560	I570	…	I5 31 0
I600	I610	I620	I630	I640	I650	I660	I670	…	I6 31 0
I700	I710	I720	I730	I740	I750	I760	I770	…	I7 31 0
…	…	…	…	…	…	…	…	…	…
I31 00	I31 10	I31 20	I31 30	I31 40	I31 50	I31 60	I3170	…	I31 31 0

在 Cycle 2，Central Control 將 Layer0 PE 的運算資料傳送給每個 PE，CON_size 代表的是 Convolution Sliding Window 的大小，filter shape 為 5*5，CON_size=4；Ch_set 代表的是要處理的 channel 數目，輸入的是黑白照片，Ch_set=0；F_set 代表的是有多少個 set 的 filter 輸入，共 4 個，F_set=3，O_size 代表 output feature map 一個 row 有多少個 psum，下一層 layer shape 是 28*28*16，O_size=27；Row_in 代表這個 PE 會有多少個 row 的輸入，因為 PE00 會輸入 image row0、4、8、12、16、20、24，共 7 個，Row_in 為 7；Quant_size 是這層輸入的 ifmap 是 8bit 的，Quant_size 為 8。



Buffer 傳送資料給 PE00，PE00 接收資料情形如下圖所示

cycle 4~8



在 cycle4 時，Ifmap Row0, Column0, Channel0 (I000)進來 PE。Ifmap_enable = 1，資料存入 ifmap_spad[0][0]。

在 cycle4~8 時，Filter Row0, Column0-4, Channel0, Set0 (F0000 、F0100、F0200、F0300、F0400)依序進來 PE。filter_enable = 1，資料存入 filter_spad[0][4:0][0]。

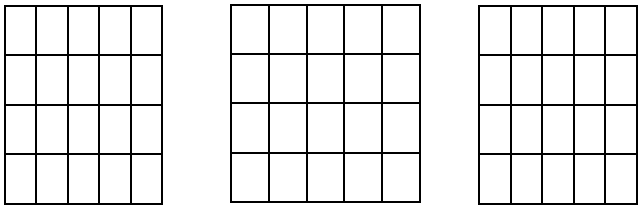
Column_Full 會去記錄現在是否有足夠的資料做 psum 運算，因已經有足夠的資料，將(I000,8'd0,8'd0,8'd0)放入 MAC 的 I_mul[3:0]、(F0000,8'd0,8'd0,8'd0)放入 MAC 的 F_mul[3:0]，，得 I000* F0000，再將運算結果存入 Psum[0]。

Ifmap Spad [4:0][3:0]

I000				

Filter Spad [3:0][4:0][3:0]

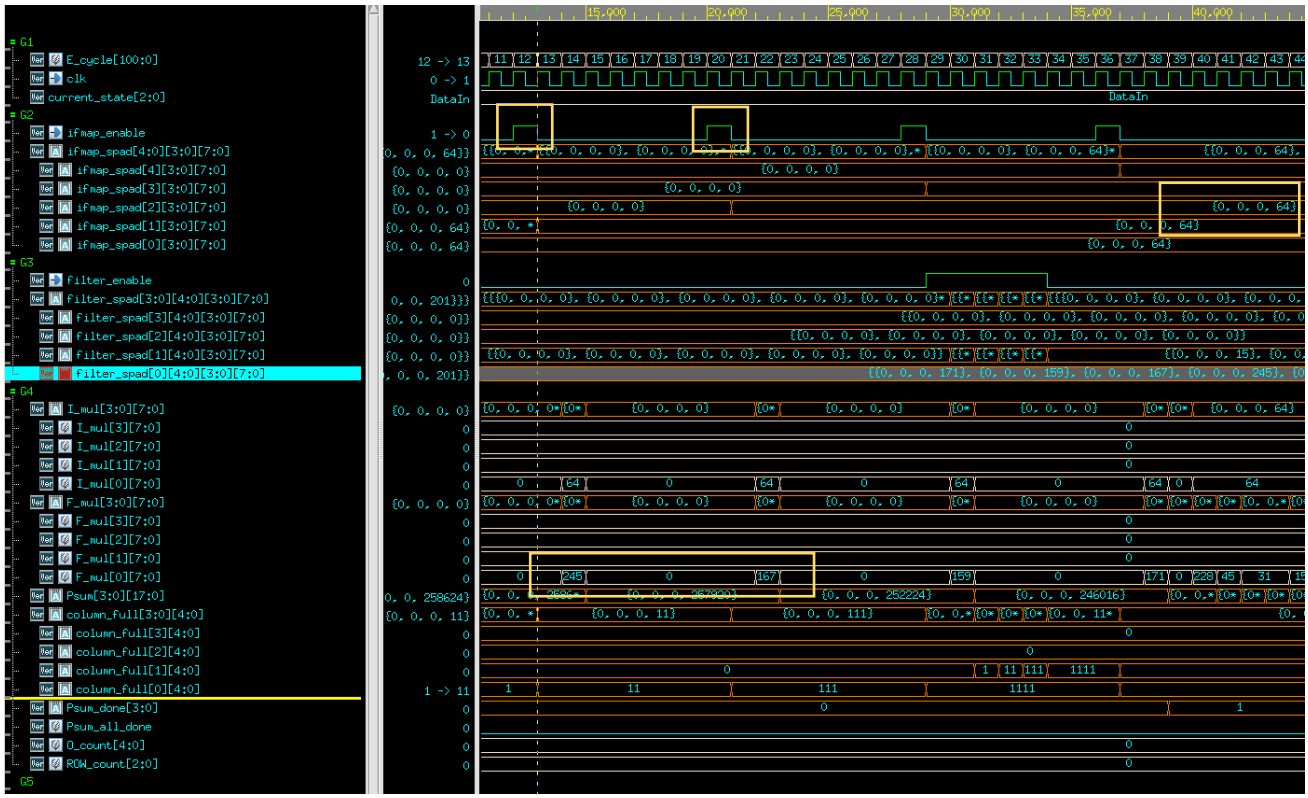
F0000	F0100		F0200	F0300	F0400



Psum[3:0]

I000* F0000			

cycle 12、20



在 cycle12、20 時，Ifmap Row0, Column1-2, Channel0，即 I010、I020 分別進來 PE。

將(I010,8'd0,8'd0,8'd0)、(F0100,8'd0,8'd0,8'd0)及(I020,8'd0,8'd0,8'd0)、(F0200,8'd0,8'd0,8'd0)，兩組資料放入 MAC，更新 Psum[0]。

Ifmap Spad [4:0][3:0]

I000	I010	I020		

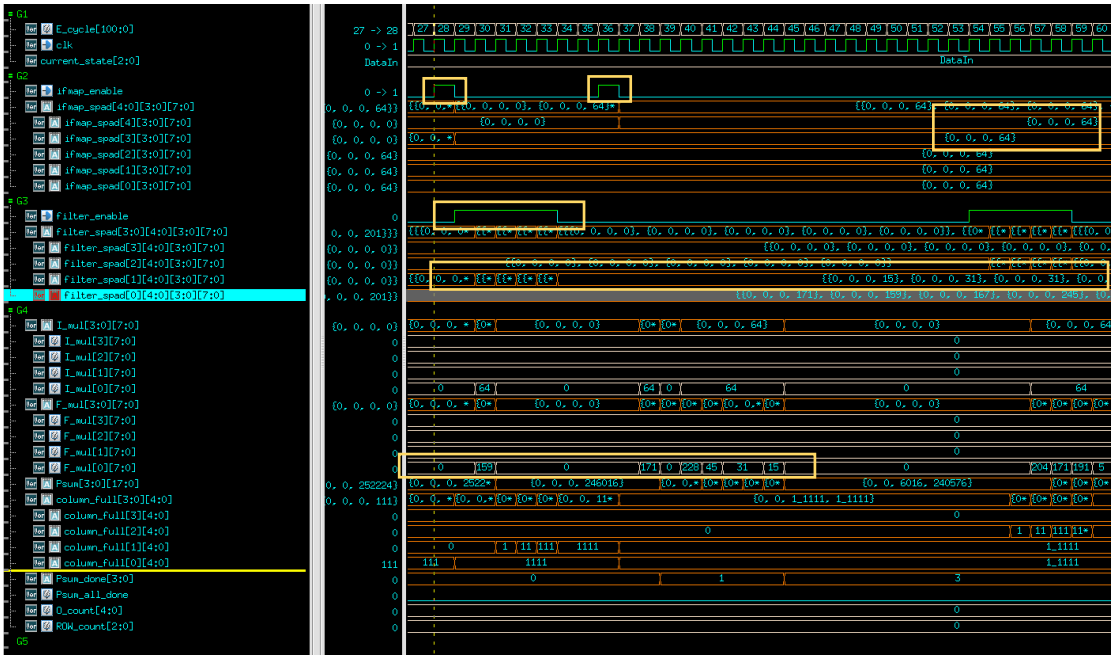
Filter Spad

F0000	F0100	F0200	F0300	F0400

Psum[3:0]

I000* F0000 + I010* F0100 + I020* F0200			

cycle 28~36



在 cycle28-36，Ifmap Row0, Column3-4, Channel0(H=0、W=3-4、C=0)，Filter Row0, Column0-4, Channel0, Set1 (R=0、S=0-4、C=0、M=1)的資料進來 PE。

將 ifmap 和 filter set1 做 convolution 的運算，依序將 5 組資料放入 MAC，更新 Psum[1]。

Ifmap Spad [4:0][3:0]

I000	I010	I020	I030	I040

Filter Spad

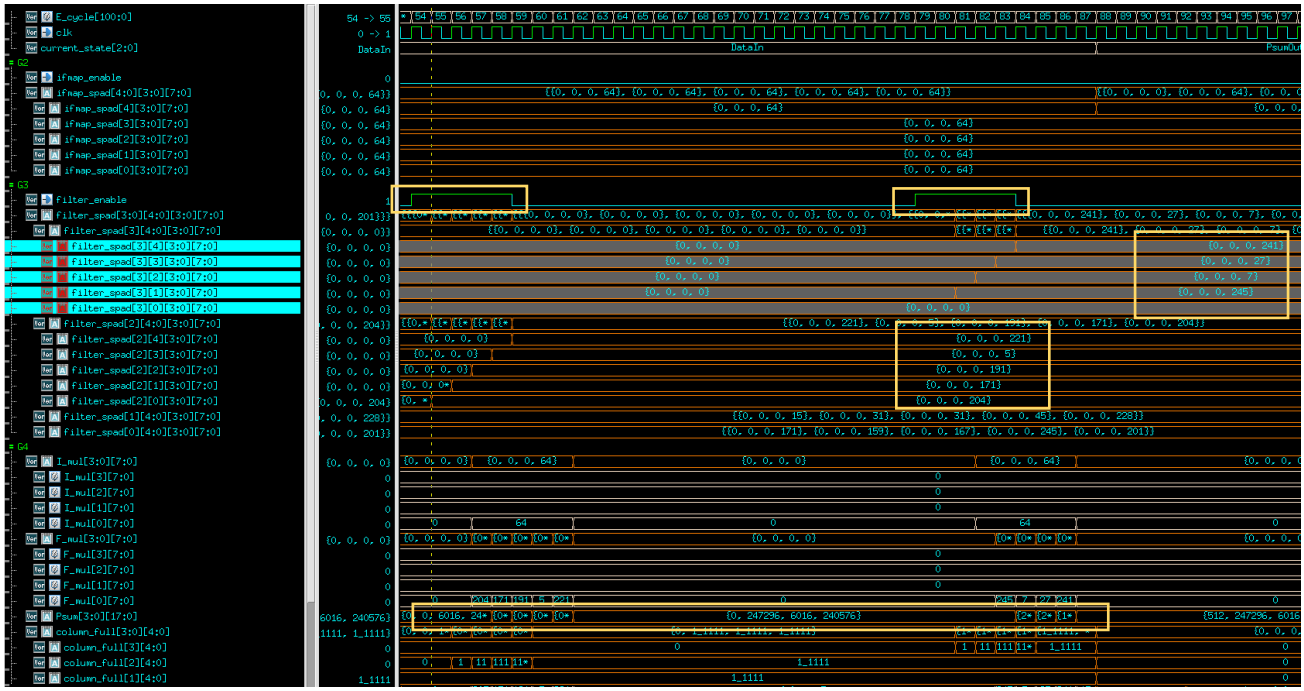
F0000	F0100	F0200	F0300	F0400

F0001	F0101	F0201	F0301	F0401

Psum[3:0]

I000* F0000 + I010* F0100 + I020* F0200+ I030* F0300 + I040* F0400	I000* F0001 + I010* F0101 + I021* F0200+ I031* F0301 + I040* F0401		

Cycle 54-83



在 cycle54-58，Filter Row0, Column0-4, Channel0, Set2 (R=0、S=0-4、C=0、M=2)
的資料進來 PE。
在 cycle79-83，Filter Row0, Column0-4, Channel0, Set3 (R=0、S=0-4、C=0、M=3)
的資料進來 PE。
將 ifmap 和 filter set2、3 做 convolution 的運算，依序將 10 組資料放入 MAC，
更新 Psum[2]、Psum[3]。

Ifmap Spad [4:0][3:0]

I000	I010	I020	I030	I040

Filter Spad

F0000	F0100	F0200	F0300	F0400

F0001	F0101	F0201	F0301	F0401

F0002	F0102	F0202	F0302	F0402

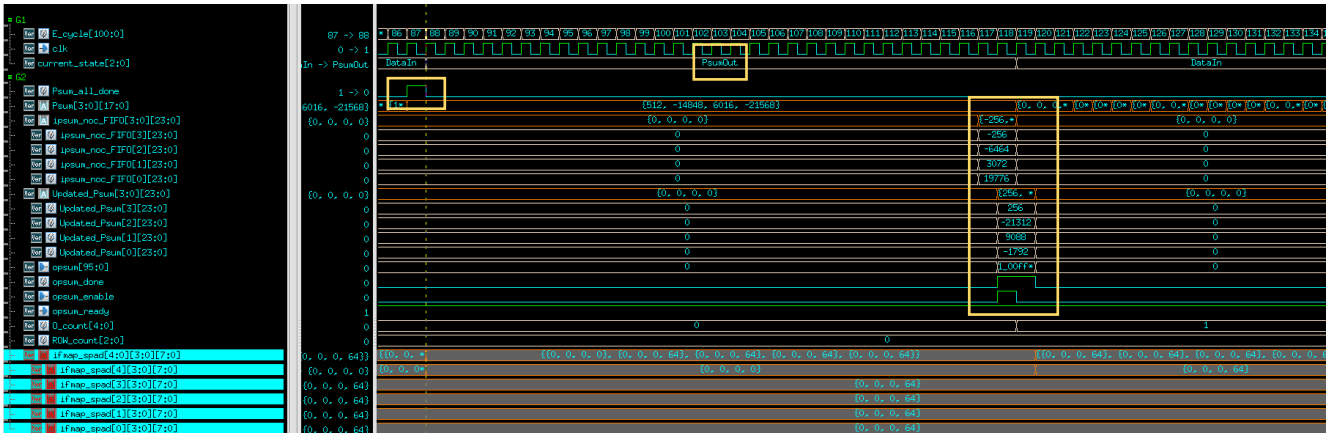
F0003	F0103	F0203	F0303	F0403

Psum[3:0]

Psum[0]	Psum[1]	I000* F0002 + I010* F0102 + I022* F0202+ I030* F0302 + I040* F0402	I000* F0003 + I010* F0103 + I021* F0203+ I031* F0303 + I040* F0403

將所有資料都運算完成之後，在 cycle88，state 進入 PsumOut，此時等待 PE40 將 LocalNetwork Psum 傳輸給 PE30，更新 4 個 Psum，PE30 再將 Psum 給 PE20，PE20 再將 Psum 給 PE10。PE00 在 cycle118，接收到 PE10 傳來的 24bit 的 4 筆 Psum(ipsum_noc_FIFO[3:0])，將 ipsum_noc_FIFO[0]+Psum[0]=Updated_Psum[0]、ipsum_noc_FIFO[1]+Psum[1] =Updated_Psum[1]、ipsum_noc_FIFO[2]+Psum[2]= =Updated_Psum[2]、ipsum_noc_FIFO[3]+Psum[3] =Updated_Psum[3]，opsum 將這 96bit 的 psum，傳送到 Psum Address Module。並且將 O_count 加一，代表完成一組 psum。

此時也會將 ifmap spad 往左 shift 一個 column，在 cycle119 進入 State DataIn 會接收 I050。Filter 持續留在 Spad 中，



Ifmap Spad [4:0][3:0]

I010	I020	I030	I040	

Filter Spad

F0000	F0100	F0200	F0300	F0400

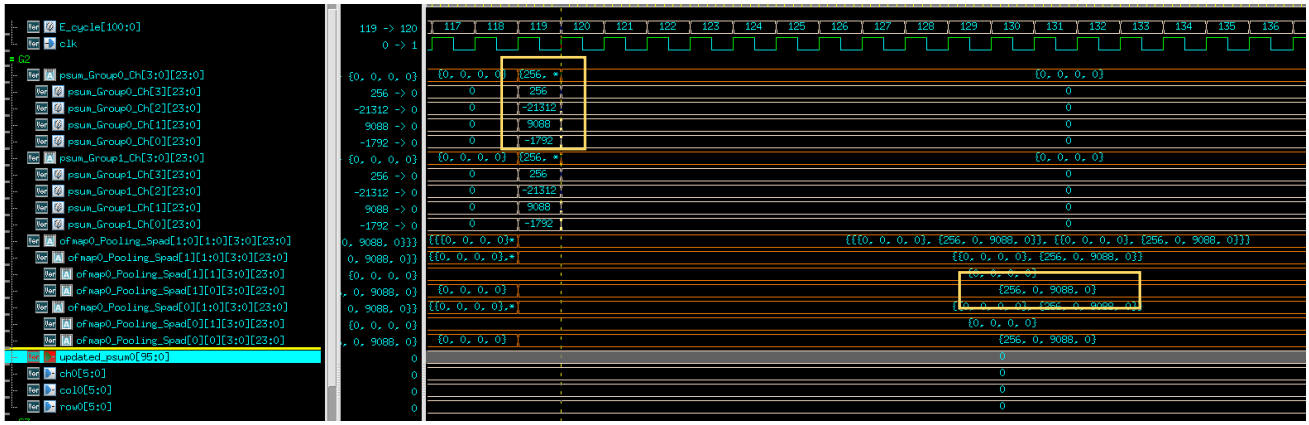
F0001	F0101	F0201	F0301	F0401

F0002	F0102	F0202	F0302	F0402

F0003	F0103	F0203	F0303	F0403

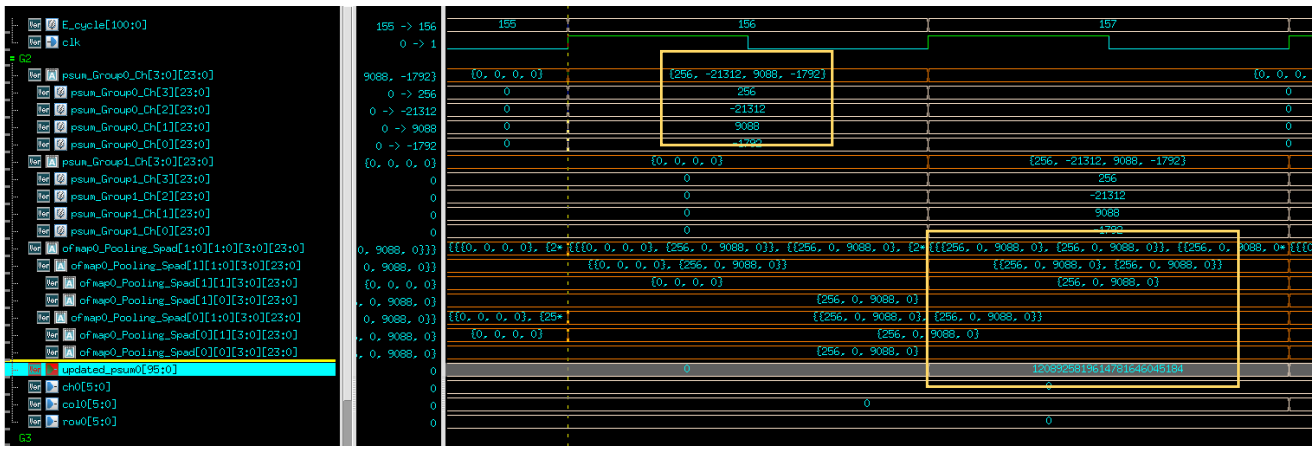
Psum Address Module

在 cycle119，接收到 Psum Group0 的 4 筆 Psum，會將其做 ReLU 後存入 ofmap0_Pooling_Spad[0][0][3:0]，和 Psum Group1 的 4 筆 Psum，做 ReLU 存入 ofmap0_Pooling_Spad[1][0][3:0]。



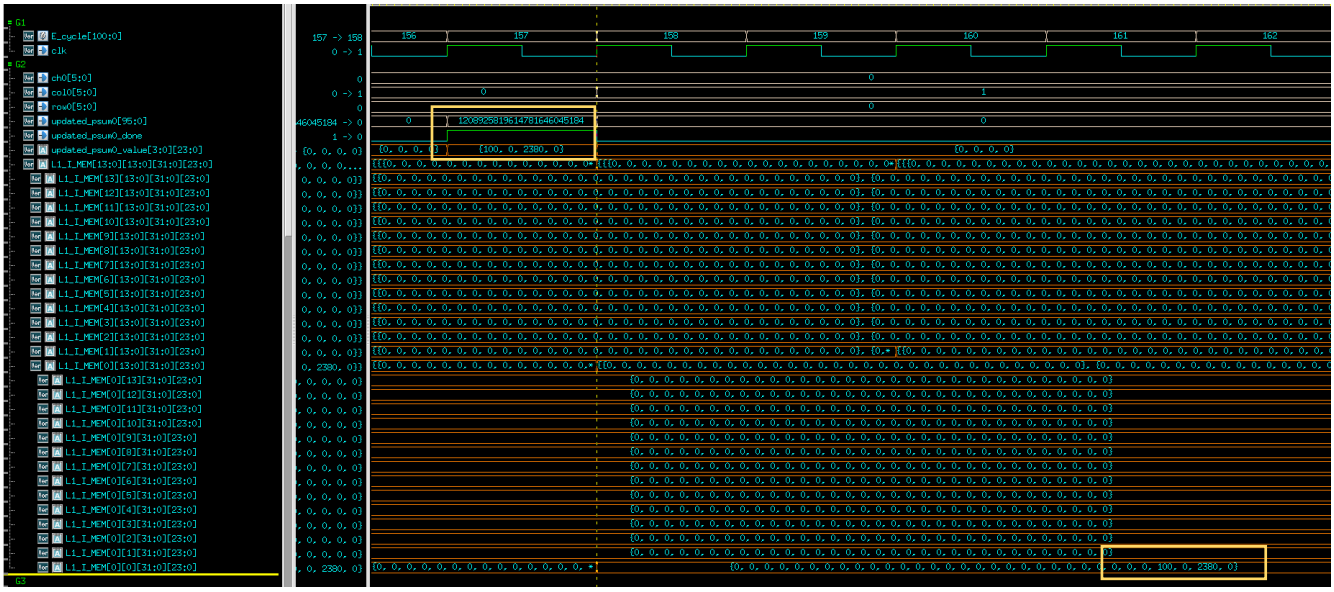
在 cycle156，接收到 Psum Group0 的第 2 組 4 筆 Psum，會將其做 ReLU 後存入 ofmap0_Pooling_Spad[0][1][3:0]，和 Psum Group1 的第 2 組 4 筆 Psum，做 ReLU 存入 ofmap0_Pooling_Spad[1][1][3:0]。

在此會比較 ofmap0_Pooling_Spad[0][0][0]、ofmap0_Pooling_Spad[0][1][0]、ofmap0_Pooling_Spad[1][0][0]、ofmap0_Pooling_Spad[1][1][0]，選出最大的數值，作為 updated_psum0[23:0]。



輸出給 buffer 的 data	MaxPooling 做比較的數值
updated_psum0[23:0]	ofmap0_Pooling_Spad[0][0][0]、 ofmap0_Pooling_Spad[0][1][0]、 ofmap0_Pooling_Spad[1][0][0]、 ofmap0_Pooling_Spad[1][1][0]
updated_psum0[47:24]	ofmap0_Pooling_Spad[0][0][1]、 ofmap0_Pooling_Spad[0][1][1]、 ofmap0_Pooling_Spad[1][0][1]、 ofmap0_Pooling_Spad[1][1][1]
updated_psum0[71:48]	ofmap0_Pooling_Spad[0][0][2]、 ofmap0_Pooling_Spad[0][1][2]、 ofmap0_Pooling_Spad[1][0][2]、 ofmap0_Pooling_Spad[1][1][2]
updated_psum0[95:72]	ofmap0_Pooling_Spad[0][0][3]、 ofmap0_Pooling_Spad[0][1][3]、 ofmap0_Pooling_Spad[1][0][3]、 ofmap0_Pooling_Spad[1][1][3]

Buffer 接收到 96bit 的 updated_psum0 之後，會藉由 Psum Address Module 換算出的 row=0、column=0、channel=0 存入 L1_I_MEM[0][0][3:0]。成為下一層 layer 的 input。



以下是 PE00 即將完成第 28 組 psum，O_count 為 27。
完成後會將 ifmap_spad 清空，準備接收 ifmap row4 的資料。
ifmap row4 和 Filter Set 0-3 做 convolution。

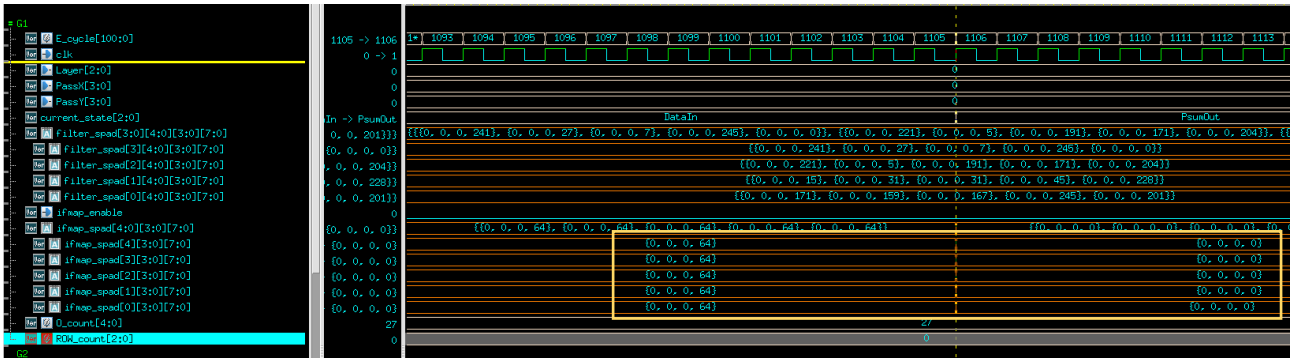
Ifmap Spad [4:0][3:0]

I0 27 0	I0 28 0	I0 29 0	I0 30 0	I0 31 0

→

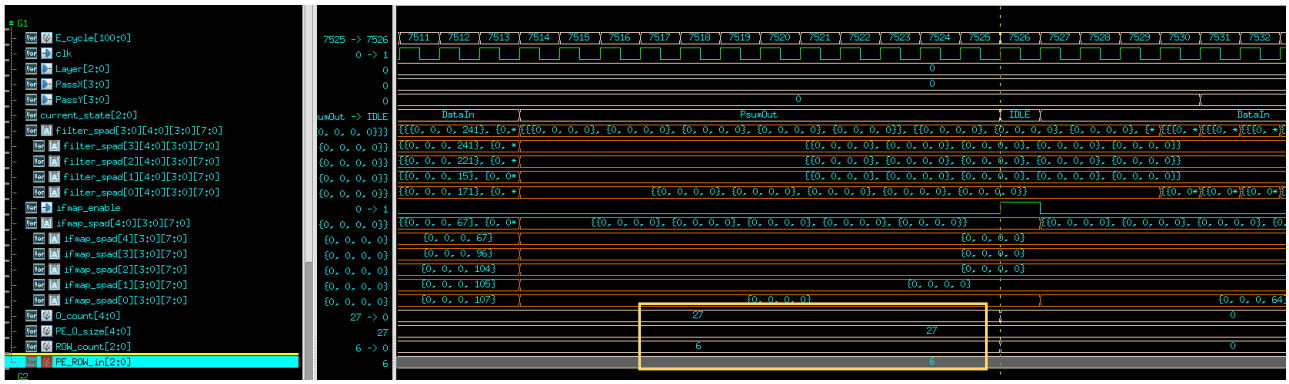
I400	I410	I410	I420	I430

PE00 將 ifmap_spad 清空後，
會接著接收右邊表格的資料。



以下是即將完成 Processing Pass0-0，module 的運作。

PE00 接收完 image row0、4、8、12、16、20、24，Row_count 為 6。Filter_Spad 在 Processing Pass0-1，要接收下一組的 Filter Set 4-7，因此將 filter_spad 清空。



Ifmap Spad [4:0][3:0]

I24 27 0	I24 28 0	I24 29 0	I24 30 0	I24 31 0

I000	I010	I010	I020	I030

PE00 將 ifmap_spad 清空後，
會接著接收右邊表格的資料。

Filter Spad

F0000	F0100	F0200	F0300	F0400

F0001	F0101	F0201	F0301	F0401

F0002	F0102	F0202	F0302	F0402

F0003	F0103	F0203	F0303	F0403

PE 將 filter_spad 清空後，會接著接收下面表格的資料。

F0004	F0104	F0204	F0304	F0404

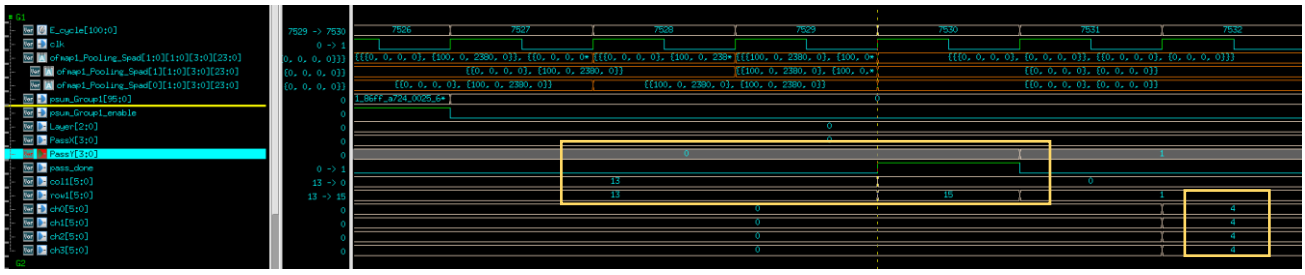
F0005	F0105	F0205	F0305	F0405

F0006	F0106	F0206	F0306	F0406

F0007	F0107	F0207	F0307	F0407

PE Group 3、4 已經輸出 processing pass0-0 最後兩筆 96bit 的 psum，並且將 PE Group 3、4 輸出的 4 筆資料(位於 ofmap1_Pooling_Spad)，做 Maxpooling 和 ReLU 之後，會產生一筆 96bit 的 psum。Psum Address Module 輸出的 row1、column1、channel1 為 13、13、0，得知此筆 psum 會存入 Buffer L1_I_MEM 的 [13][13][3:0]，即 processing pass0-0 的在 Psum Address Module 輸出的最後一筆 psum。

資料輸出後，Psum Address Module 會將 pass_done 拉起為 1，傳回 Central Control，Central Control 得知已經完成一個 processing pass，將 Pass Y 加 1。Psum Address Module 在 processing pass 0-1 輸出給 psum 的 channel 為 4-7，channel0,1,2,3 變為 4 (ch0,1,2,3 = PassY << 2)。



4. 結論

4.1 專案回顧

本次專案實踐了 LeNet 加速器，雖未能完成整體全部架構，但以進度來說已完成約 7 成，僅剩餘最後兩層結構未能在時限內完成，其餘硬體架構皆以完成相關功能設計，但仍存在許多有待完善的部分，將會持續努力以完成本次專案。

5. 參考資料

[1]. OpenCV official documentation
https://opencv.org/
[2]. Python official documentation
https://docs.python.org/3/