Integrated Circuit Design
Final project
Image Convolutional Circuit Design

1. 問題描述

本題請完成一圖像卷積電路設計(Image Convolutional Circuit Design，以下稱 CONV 電路)，CONV 電路輸入為一灰階圖像，電路需完成 3 層(Layer)的運算流程，其順序為 Convolutional(Layer 0)→Max-pooling(Layer 1)→Flatten(Layer 2)等共 3 層的運算處理流程，如下圖 1.所示。

首先在 Layer 0 中，輸入灰階圖像(尺寸為 64(寬)x64(高) pixels)需先經 zero-padding，接著進行採用 2 個濾鏡(或稱"核(Kernel)")的圖像卷積(Convolutional)運算，其 Kernel 尺寸為 3x3，接著再經過 ReLU 運算後方為 Layer 0 的結果，故其結果為 2 張尺寸為 64(寬)x64(高) pixels 的圖。

Layer 1 要進行最大池化(max-pooling)運算，須採用 2x2 的 max-pooling 視窗及步幅(stride)為 2 的規格進行。故結果將呈現 2 張尺寸為 32(寬) x 32(高) pixels 的圖。

最後 Layer 2 則是進行平坦化(Flatten)處理，將 Layer 1 輸出的 2 張 32(寬) x 32(高) pixels 的圖依規格排序成一個 2048 (32x32x2)個訊號值的序列並輸出即可完成。

上述所有 Layer，均可分層分次將處理結果寫回 testfixture 內建的記憶體內 (如圖 2.之 L0_MEM0、L0_MEM1、L1_MEM0、L1_MEM1 及 L2_MEM)中，每一 Layer 的輸出結果都有各自對應的記憶體存放空間用於放置輸出結果。

請參考附錄中所列的要求，附上評分所需要的檔案。

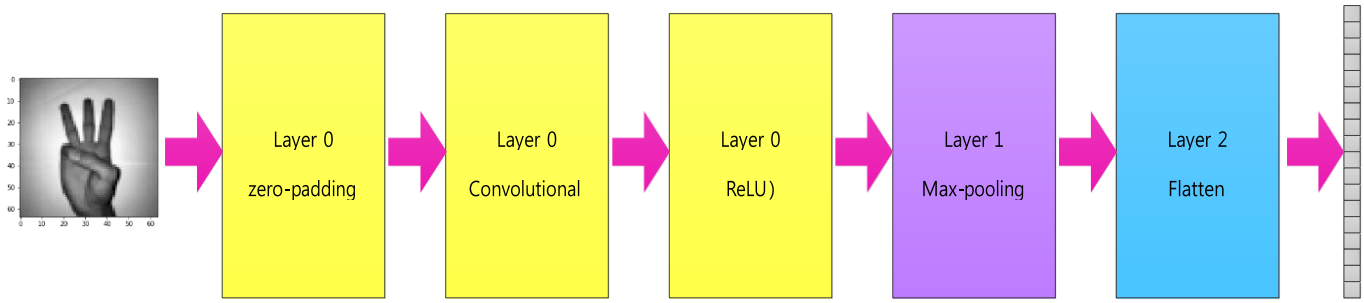

圖 1. 圖像卷積電路系統架構

2. 設計規格

2.1 系統方塊圖

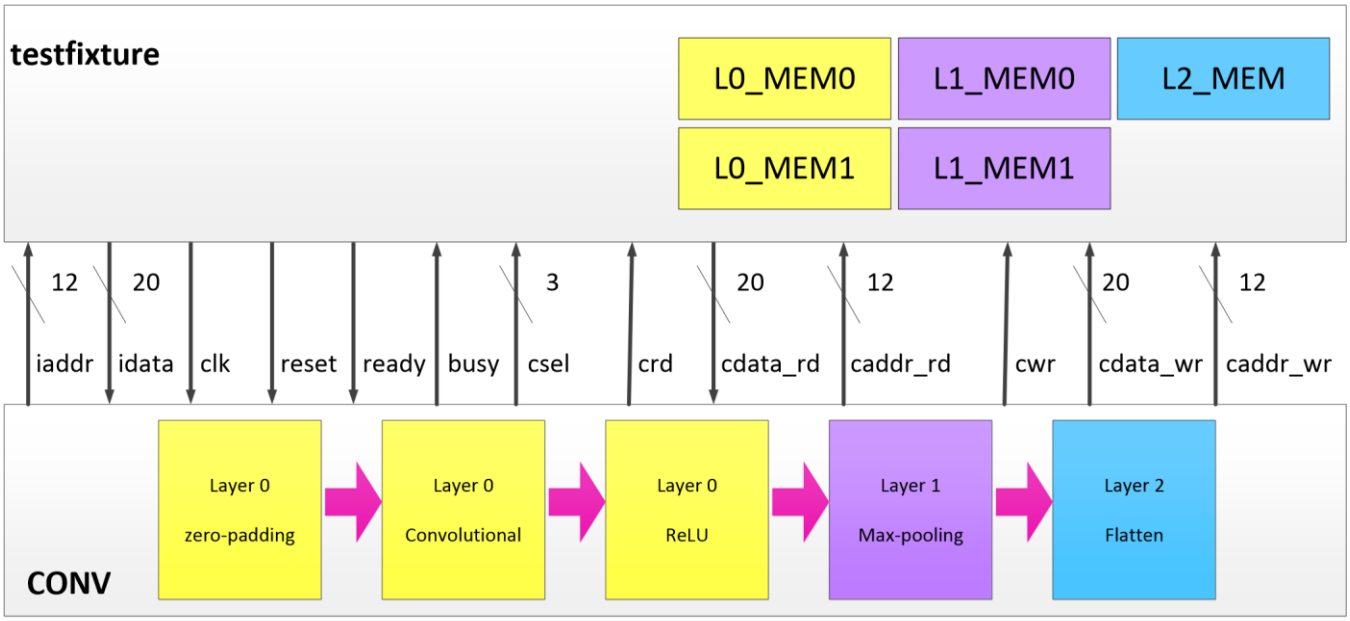

圖 2. 系統方塊圖

2.2 輸出入訊號和記憶體描述表一、輸入/輸出信號

Signal Name	I/O	Width	Simple Description
clk	I	1	系統時脈訊號。本系統為同步於時脈**正緣**之同步設計。
reset	I	1	**高**位準"非"同步(active low asynchronous)之系統重置信號。
ready	I	1	灰階圖像準備完成指示訊號。當訊號為 High 時，表示灰階圖像準備完成，此時 CONV 才可以開始向 testfixture 發送輸入灰階圖像資料索取位址。
busy	O	1	系統忙碌指示訊號。當 CONV 接收到 ready 訊號為 High，且 CONV 準備開始動作時，需將此訊號設為 High，表示準備開始進行輸入灰階圖像資料索取；待所有運算處理完成且輸出結果寫回 testfixture 後，需再將訊號設為 Low 表示動作結束。
iaddr	O	12	輸入灰階圖像位址訊號。指示欲索取哪個灰階圖像像素(pixel)資料的位址。
idata	I	20	輸入灰階圖像像素資料訊號，由 4 bits 整數(MSB)加上 16 bits 小數(LSB)組成，為有號數。testfixture 將 iaddr 所指示的位址之像素資料用此訊號送給 CONV。

crd	O	1	CONV 運算輸出記憶體**讀取致能**訊號。當時脈正緣觸發時，若此訊號為 High，表示要進行讀取動作。testfixture 會將 caddr_rd 位址指示之資料讀取到 cdata_rd 上。
cdata_rd	I	20	CONV 運算結果記憶體**讀取**訊號，由 4 bits 整數(MSB)加上 16 bits 小數(LSB)組成，為有號數。Testfixture 將記憶體資料傳送至 CONV 電路。
caddr_rd	O	12	CONV 運算結果記憶體讀取位址。CONV 電路各層的運算結果利用此訊號指示將要讀取 testfixture 中所內建各層輸出結果之記憶體的哪個位址。
cwr	O	1	CONV 運算輸出記憶體寫入致能訊號。當時脈正緣觸發時，若此訊號為 High，表示要進行寫入動作。testfixture 會將 cdata_wr 內容寫到 caddr_wr 所指示之位址。
cdata_wr	O	20	CONV 運算結果記憶體寫出訊號，由 4 bits 整數(MSB)加上 16 bits 小數(LSB)組成，為有號數。CONV 電路各層的運算結果利用此訊號輸出至 testfixture。
caddr_wr	O	12	CONV 運算結果記憶體寫入位址。CONV 電路各層的運算結果利用此訊號指示將要寫入到 testfixture 中所內建各層輸出結果之記憶體的哪個位址。
csel	O	3	CONV 運算處理結果寫入/讀取記憶體選擇訊號。此訊號指示目前寫入/讀取資料為 CONV 電路中哪一層的運算結果。 3'b000:表示沒有選擇記憶體。 3'b001: 寫入/讀取 Layer 0，Kernel 0 執行 Convolutional 的結果。 3'b010: 寫入/讀取 Layer 0，Kernel 1 執行 Convolutional 的結果。 3'b011: 寫入/讀取 Layer 1，將 Kernel 0 執行 Convolutional 後再進行 Max-pooling 運算的結果。 3'b100: 寫入/讀取 Layer 1，將 Kernel 1 執行 Convolutional 後再進行 Max-pooling 運算的結果。 3'b101:表示寫入/讀取 Layer 2，Flatten 層的運算結果。

2.3 系統功能、時序及記憶體對應方式描述

系統方塊圖如圖 2.所示。當 reset 啟動結束後，testfixture 會將 ready 訊號設為 High 表示灰階圖像資料及各層 Kernel 資料都已經準備完畢，接著 CONV 電路須將 busy 訊號設為 High 表示開始動作(如圖 3. t1 時間點)，而 testfixture 偵測到 busy 訊號為 High 後就會將 ready 訊號設為 Low 表示等待 CONV 電路處理題目所要求的動作(如圖 3. t2 時間點)。待各層所要求的動作都實現完成或已經將欲實現的 Layer 動作處理完成後，CONV 電路就可將 busy 訊號再次設為 Low 表示所有動作已經完成(如圖 3. t3 時間點)，此時 testfixture 就會準備下一張灰階圖像並將 ready 訊號設定為 High；並且 testfixture 一偵測到 busy 訊號再次被設定為 Low 就會立刻進行資料驗證比對，因此針對每一張輸入灰階圖像的運算處理，busy 訊號只允許在 CONV 電路開始動作時被設定為 High 一次，CONV 運算處理結束時設定為 Low 一次。在 busy 為 High 的過程中，CONV 電路可重複不限次數對

L0_MEM0、L0_MEM1、L1_MEM0、L1_MEM1 及 L2_MEM 進行讀取及寫入動作。

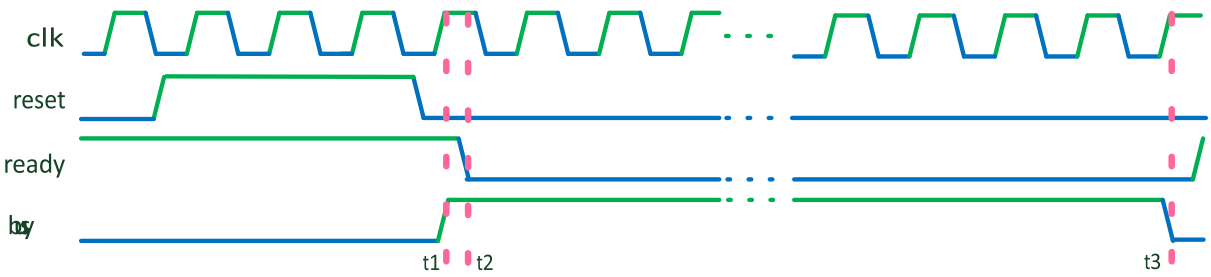

圖 3. 系統啟動及結束時序

助教評分時將視情況決定是否增加灰階圖像樣本數進行測試。

關於 CONV 電路各層之動作說明如下，整體流程如圖 4.所示:

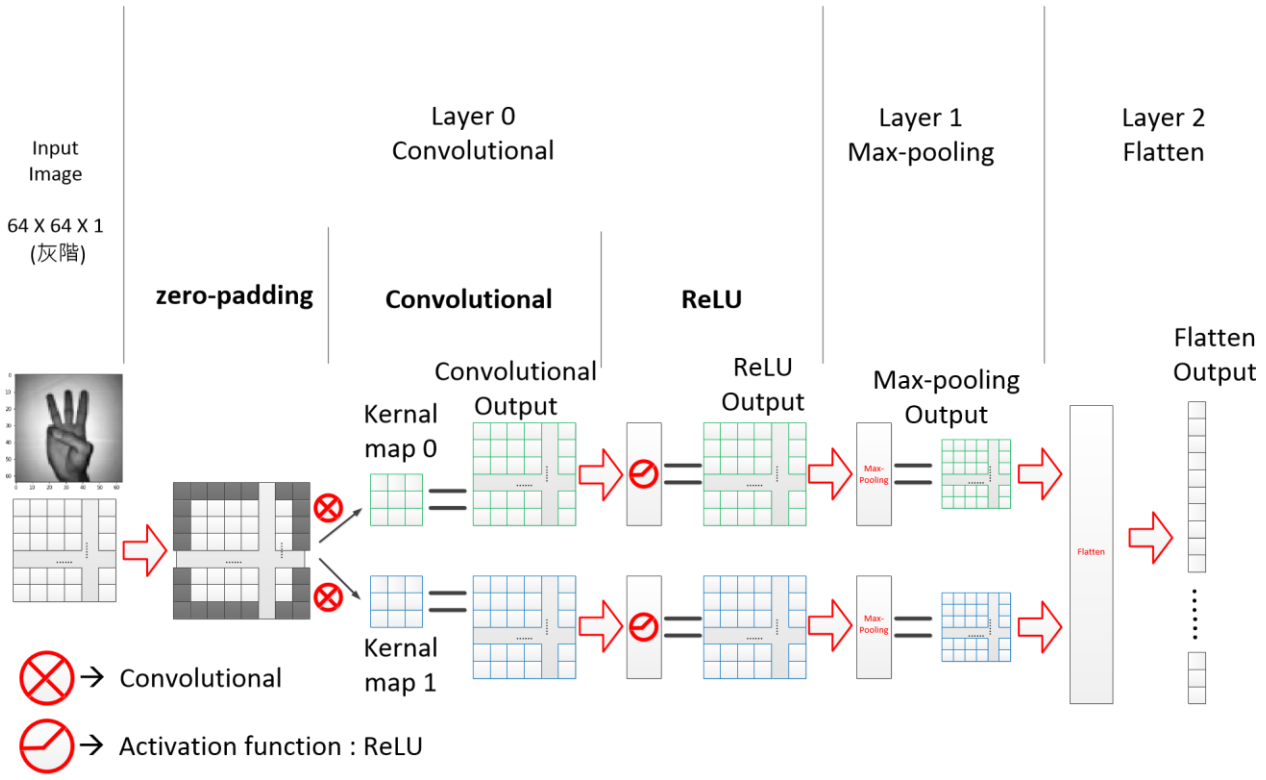

圖 4. CONV 電路系統流程圖

1. Input image 為本題輸入灰階圖像，其尺寸為 64 pixels(寬)x 64 pixels(高)，存放於 testfixture 的記憶體中，灰階圖像各 pixels 與其記憶體的對應方式如下圖 6.說明。動作時序上，CONV 電路需利用 iaddr 發送欲索取圖像資料的位址到 testfixture(如圖 5. t1 時間點)，testfixture 在每個時脈負緣後會將 iaddr 所指示位址之 pixel 資料利用 idata 送入 CONV 電路(如圖 5. t2 時間點)。

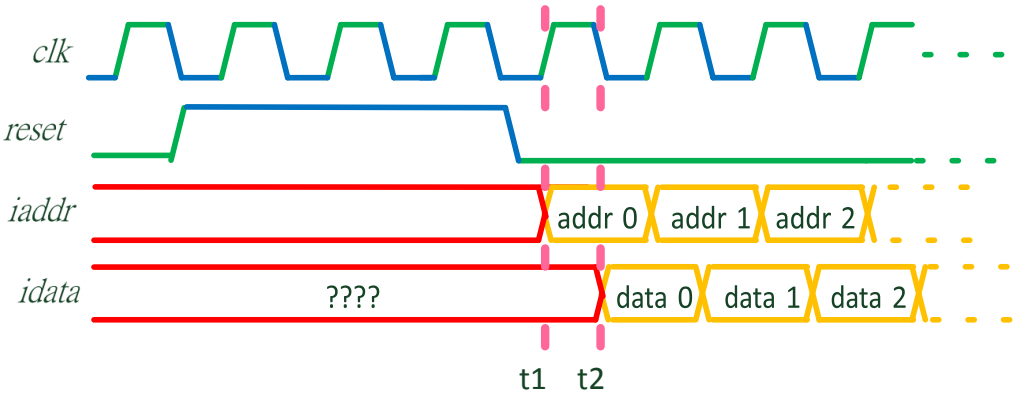

圖 5.灰階圖像資料記憶體時序

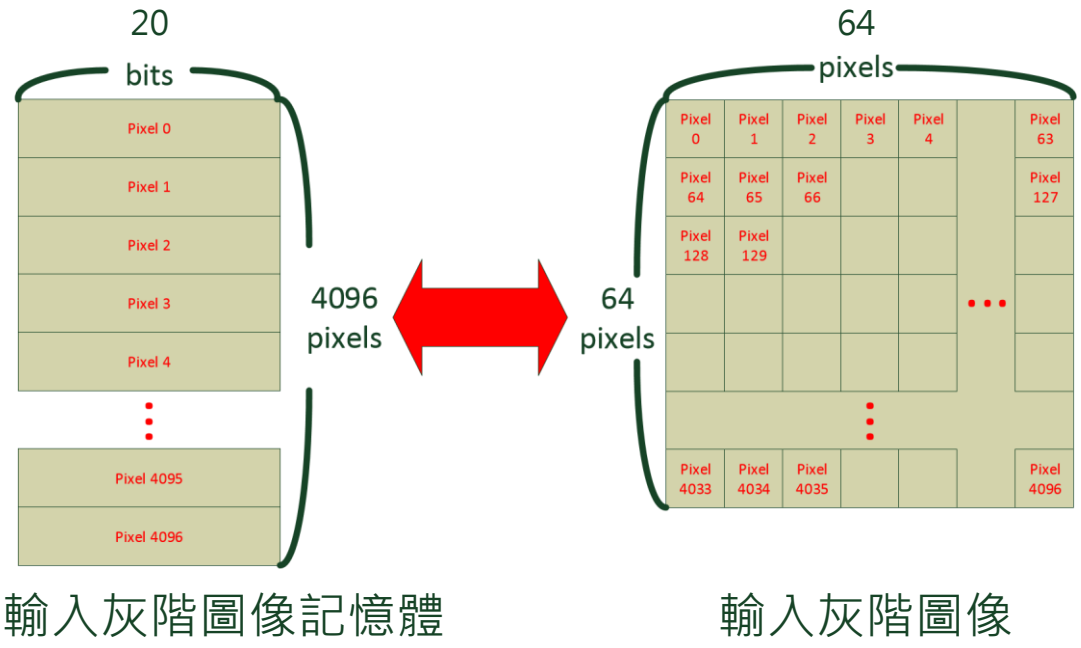

圖 6.輸入灰階圖像及其記憶體對應方式

2. Layer 0 層所要進行的動作是將 input image 進行 Zero-padding 處理後，再分別與 Kernel 0 及 Kernel 1 作 Convolutional 運算處理後得到 2 張尺寸皆為 64 pixels(寬)x 64 pixels(高)的圖，再將這兩張圖的每個 pixels 分別進行 ReLU 運算後得到 Layer 0 的結果。

A. Convolutional

本作業規定之 Convolutional 處理為針對待處理圖像以固定移動 1 pixel 的間隔，將 3x3 大小的 Kernel 疊到待處理圖上左上角，如圖 7.紅色框為例，圖中每一個 pixel 都

跟對應到 Kernel 上面的每個 pixel 相乘，最後全部相加，得到 1x2 + 2x0 + 3x1 + 0x0 + 1x1 + 2x2 + 3x1 + 0x0 + 1x3 =16， 接著 Kernel 往右移動一格，如圖 7.綠色框為例，並做相同的運算得到結果為 18，依此類推依序由左到右，由上到下移動(如圖 7.淺藍色框)計算到所有 pixels 乘積後可得 Convolutional 結果(如圖 7.中間所示)。最後每個 pixel 還要再各自加上 bias 值後才是 Convolutional 運算的最後輸出(如圖 7.最右邊區塊)。

本題給定之 Kernel 0 及 Kernel 1 分別如圖 8.及圖 9.所示，Kernel 0 所對應的 bias 值為(0.07446326, 10 進制)(01310, 16 進制, 4bits 整數+16bits 小數)；Kernel 1 所對應的 bias 值為(-0.55241907, 10 進制)(F7295, 16 進制, 4bits 整數+16bits 小數)。

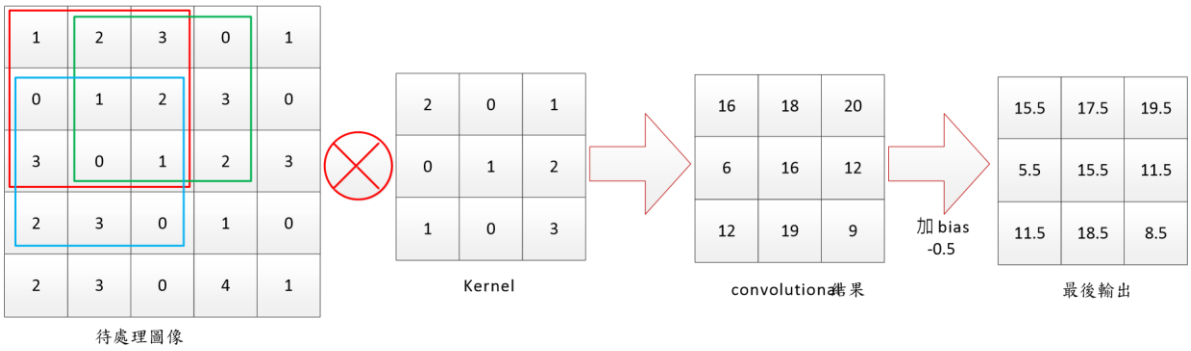

圖 7. Convolutional 運算範例

0.658674	0.573572	0.42681
0.0625617	-0.439696	-0.569037
-0.34829	-0.217964	-0.327755

Kernel 0 (10 進制)

-0.143247	0.162391	-0.212595
0.31637	0.184082	0.125697
0.23376	-0.17419	0.368789

Kernel 1 (10 進制)

圖 8. 本題給定 Kernel 0 和 Kernel 1 (10 進制)

0A89E	092D5	06D43
01004	F8F71	F6E54
FA6D7	FC834	FAC19

Kernel 0 (16 進制)

FDB55	02992	FC994
050FD	02F20	0202D
03BD7	FD369	05E68

Kernel 1 (16 進制)

圖 9. 本題給定 Kernel 0 和 Kernel 1 (16 進制) (4bits 整數+16bits 小數)

B. Zero-padding

如圖 7.範例所示，Convolutional 處理之結果會讓圖片尺寸縮小；但若在 Convolutional 處理前先進行填補(padding)就可讓圖片尺寸維持相同，本題規定須將待處理圖像周圍填補 1 pixel 的 0。如圖 10.所示為先做 zero-padding 補 1 pixel 後進行 Convolutional 處理的例子。

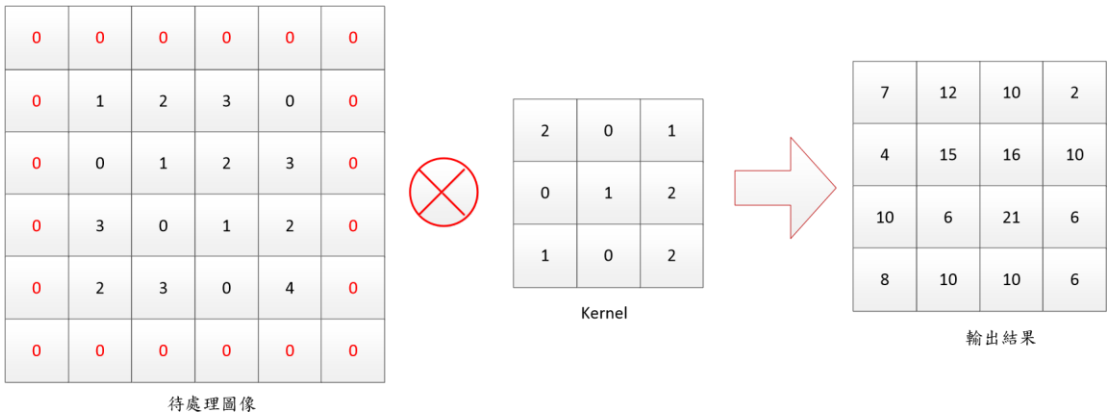

圖 10. Zero-padding 運算範例

C. ReLU

ReLU(Rectified Linear Unit)可以下算式表示，如果輸入 pixel 的值 x 小於或等於 0 則此 pixel 的值須調整成 y 為 0；但如果輸入 pixel 的值 x 大於 0 則此 pixel 的值仍為 x。 Convolutional 處理後的每一個 pixel 的值都要經過 ReLU 運算後方為 Layer 0 的最終結果。

$$y = \begin{cases} x\ (x > 0) \\ 0\ (x \leq 0) \end{cases}$$

D. CONV 電路須將 Layer 0 的結果共 2 張圖，分別用 caddr_wr 指定寫回 testfixture 記憶體的位址，運算結果資料用 cdata_wr 送到 testfixture，每張圖有各自獨立記憶體因此須設定 csel 為 3'b001(經 Kernel 0)及 3'b010(經 Kernel 1)來啟動各自的記憶體。其輸出圖與記憶體的對應方式如圖 11.所示。其中 csel=3'b001 所讀寫的記憶體為 L0_MEM0；cesl=3'b010 所讀寫的記憶體為 L0_MEM1。以上 L0_MEM0 及 L0_MEM1 之資料寬度 20bits，且為 4bits 整數+16bits 小數，Layer 0 實際計算結果可能大於 20 bits，同學須取整數 4bits 及小數點後 16 bits 之後做進位(第 17 個 bit 做四捨五入)作為輸出。

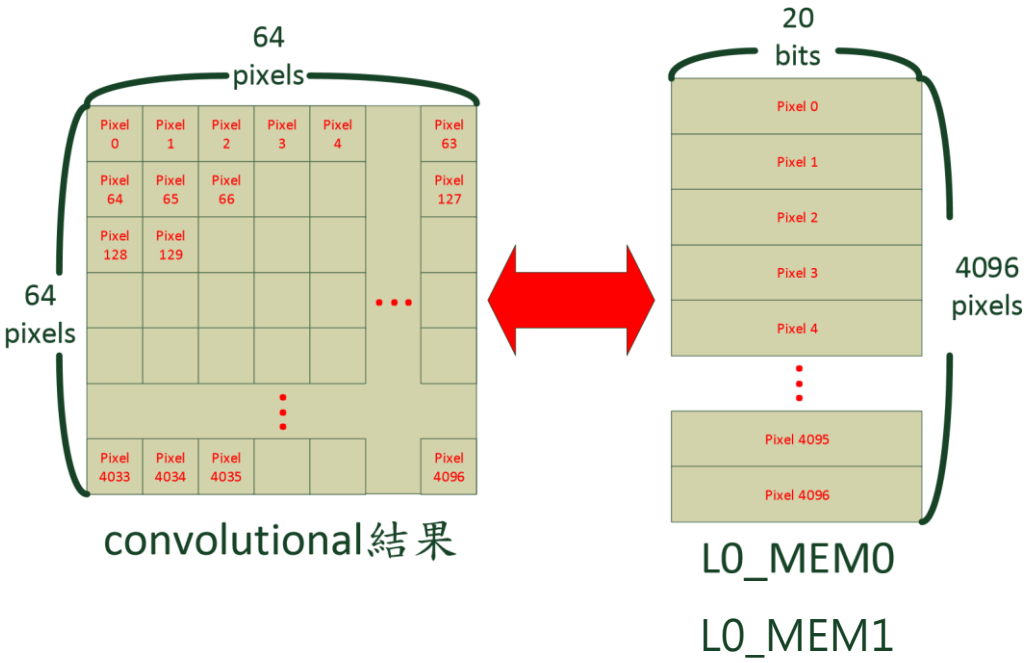

圖 11. L0_MEM0/L0_MEM1 與 convolutional 結果對應方式

3. Layer 1 層要進行 max-pooling 運算，最大池化層(max-pooling)是縮小水平及垂直空間的運算，本題規定以如下圖所示將 2 pixels(寬)X 2 pixels(高)的範圍整合成一元素，縮小空間大小。因此經 max-pooling 後的結果圖之尺寸只會有輸入圖尺寸的 1/2。

將 Layer 0 輸出的 2 張 64 pixels(寬)x 64 pixels(高)的圖，各自經 max-pooling 運算後輸出 2 張 32 pixels(寬)x 32 pixels(高)的結果圖，max-pooling 的過程如圖 12.所示。本題規定 max-pooling 視窗大小為 2x2，將輸入 Layer 1 的目標圖像依序由左到右，由上到下，步幅 (stride)為 2，依序取 2x2 大小的 pixels，取最大值為其輸出，因此圖 12.左上角的 2x2 pixels(黃色區域)為 max(5, 7, 0, 1)，因此結果為最大值 7；接著往右移動 2 格視窗(淺藍色區域)計算 max(7,3,2,3)得最大值為 7，其餘依此類推。

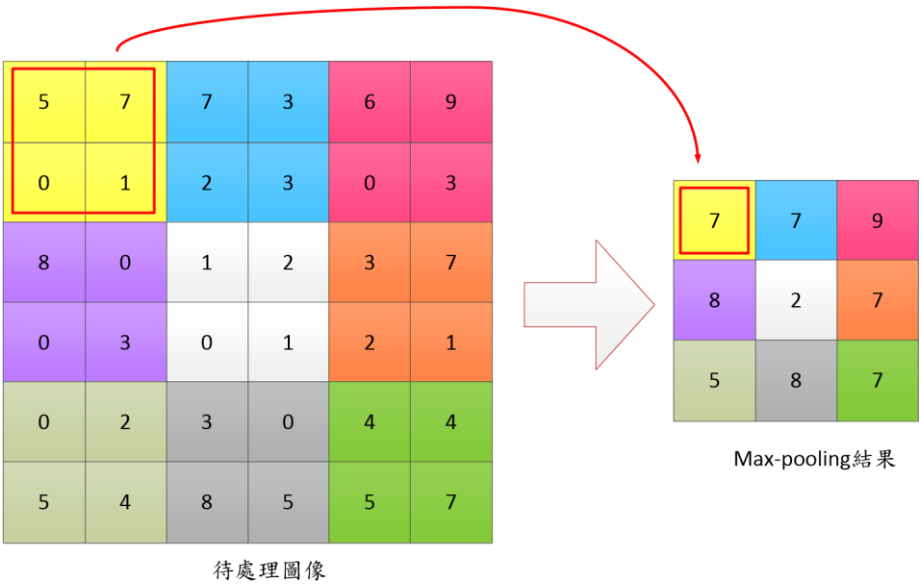

圖 12. max-pooling 運算說明範例

max-pooling 後的 2 張結果圖結果分別用 caddr 指定寫回 testfixture 記憶體的位址，運算結果資料用 cdata 送到 testfixture，每張圖有各自獨立記憶體因此須設定 csel 為 3'b011(經 Kernel 0)及 3'b100(經 Kernel 1)來啟動各自的記憶體。其輸出圖與記憶體的對應方式如圖 13.所示。其中 csel=3'b011 所讀寫的記憶體為 L1_MEM0；cesl=3'b100 所讀寫的記憶體為 L1_MEM1。

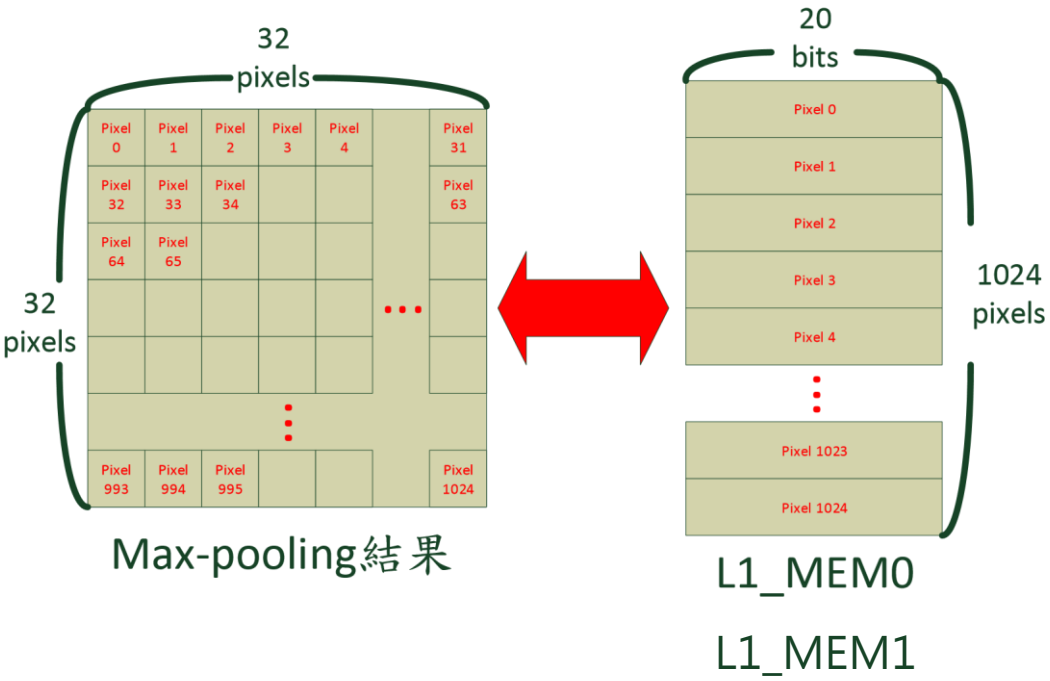

圖 13. L1_MEM0/L1_MEM1 與 max-pooling 結果對應方式

4. Layer 2 則是平坦化(flatten)運算。將 Layer 1 的輸出結果共 2 張 32x32 pixels 結果圖依序”Kernel 0→ Kernel 1”及”pixels 累加”排列成一長度為 2048 筆的 pixels 的資料向量(如下圖 14.所示)，再利用 csel 設定為 3’b101，用 caddr 指定 L2_MEM 的位址，由位址 0 開始依序將 2048 筆資料用 cdata 送出寫到 testfixture。

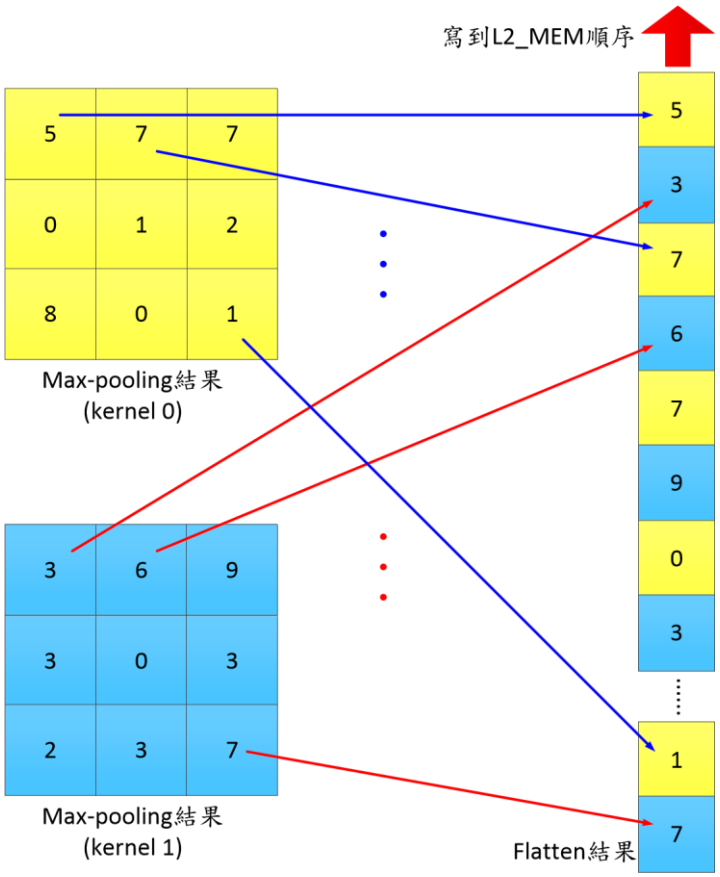

圖 14. Flatten 方式範例

特別注意 : 以上所述之 L0_MEM0、L0_MEM1、L1_MEM0、L1_MEM1 及 L2_MEM 之記憶體寬度皆為 20 bits(由 4 bits 整數(MSB)加上 16 bits 小數(LSB)組成)。

2.4 L0_MEM0、L0_MEM1、L1_MEM0、L1_MEM1 及 L2_MEM 之動作時序

各層輸出資料記憶體 L0_MEM0、L0_MEM1、L1_MEM0、L1_MEM1 及 L2_MEM 皆為 RAM model 且控制方式及時序皆相同，都可進行寫入及讀取動作。採用不同的 csel 設定值啟動各層輸出相對應的記憶體，使用 cwr 作為寫入致能訊號，crd 作為讀取致能訊號。讀取時，使用 caddr_rd 為記憶體位址，cdata_rd 作為讀取資料訊號。動作時序如下圖 15. 說明，當時脈負緣觸發時若 crd 為 High，則會在觸發後立刻將 caddr_rd 所指示位址的資料讀取到 cdata_rd 上(如圖 15. t1 時間點)。寫入時，使用 caddr_wr 為記憶體位址，cdata_wr 作為寫入資料訊號。

動作時序如下圖 16. 說明，當時時脈正緣觸發時若 cwr 為 High，則會將這時 cdata_wr 的資料寫入到 caddr_wr 所指示位址上(如圖 16. t1 時間點)。

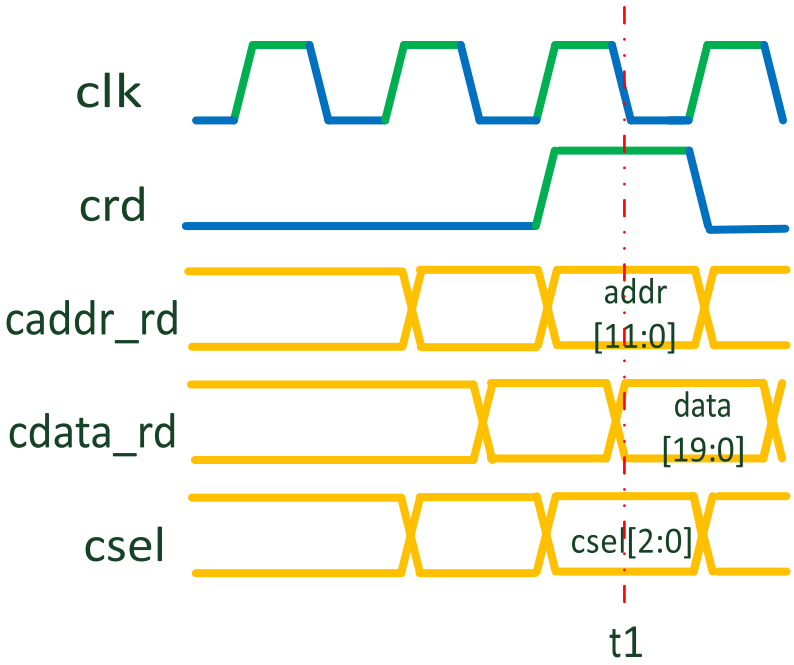

圖 15. 輸出資料記憶體 L0_MEM0、L0_MEM1、L1_MEM0、L1_MEM1 及 L2_MEM 讀取動作時序圖

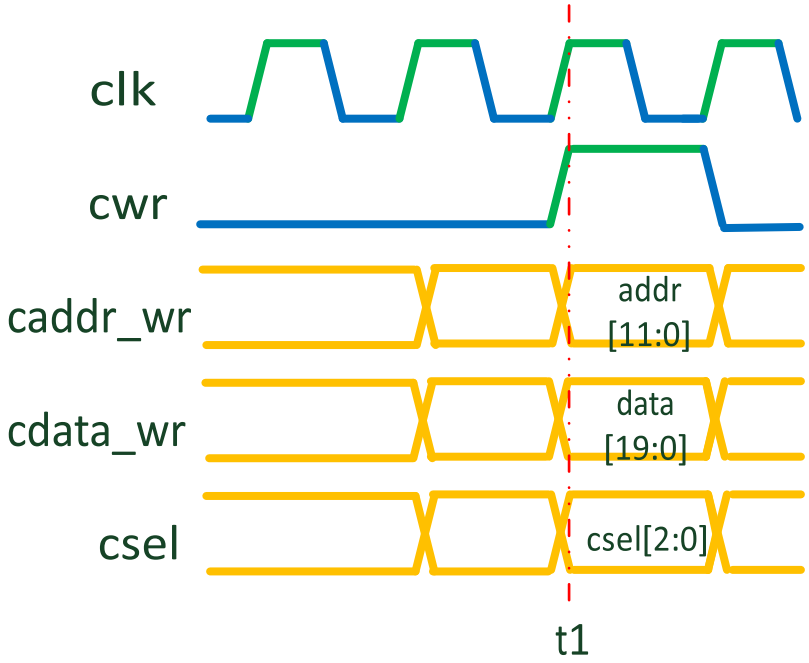

圖 16. 輸出資料記憶體 L0_MEM0、L0_MEM1、L1_MEM0、L1_MEM1 及 L2_MEM 寫入動作時序圖

附錄 A 設計檔案說明

1. 下表二.為提供同學的設計檔案

表二、設計檔

檔名	說明
testfixture.v	測試樣本檔。此測試樣本檔定義了時脈週期與測試樣本之輸入及預期輸出信號。
CONV.v	所使用的設計檔，已包含系統輸/出入埠之宣告
./dat_grad/cnn_sti.dat	測試樣本檔案
./dat_grad /cnn_layer0_exp0.dat	Layer 0 比對樣本檔案(Kernel 0)
./dat_grad /cnn_layer0_exp1.dat	Layer 0 比對樣本檔案(Kernel 1)
./dat_grad /cnn_layer1_exp0.dat	Layer 1 比對樣本檔案(Kernel 0)
./dat_grad /cnn_layer1_exp1.dat	Layer 1 比對樣本檔案(Kernel 1)
./dat_grad /cnn_layer2_exp.dat	Layer 2 比對樣本檔案
CONV.sdc	Design Compiler 電路合成規範檔。 本規範檔除了 **cycle** 可修改外，其餘 **constraints** 皆不可修改。
tsmc13_neg.v	Gate-level simulation 所需要之 cell library file

2. 請使用 CONV.v，進行本題電路之設計。其 Verilog 模組名稱、輸出/入埠宣告如下所示：

```
module  CONV(
        input           clk,
        input           reset,
        output          busy,
        input           ready,

        output  [11:0] iaddr,
        input   [19:0] idata,

        output          cwr,
        output  [11:0] caddr_wr,
        output  [19:0] cdata_wr,

        output          crd,
        output  [11:0] caddr_rd,
        input   [19:0] cdata_rd,

        output  [2:0]  csel
        );
```

3. 本題所提供的 Testbench 檔案，有多增加幾行特別用途的敘述如下：

```
`define SDFFILE    "./syn/CONV_syn.sdf"          // Modify your sdf file name

`ifdef SDF
    initial $sdf_annotate(`SDFFILE, CONV);
`endif
```

註:

1. SDF 檔案，請自行修改 SDF 實際檔案名稱及路徑後再模擬。
2. 在 Testbench 中，本題提供 `ifdef SDF 的描述，目的是讓 Testbench 可做為 RTL 模擬、合成後模擬及 layout 後模擬皆可使用。當同學在合成後及 layout 後模擬時，請務必多加上一個參數 +define+SDF ，並自行修改 SDF 實際檔案名稱，方可順利模擬。
3. 本題目共提供一組測試樣本，同學可依下面範例來進行模擬:

助教於評分時，將可能額外再使用其他組測試樣本進行評分工作。

RTL Simulation 時使用指令如下：

- 使用 ncverilog 模擬指令範例如下： *ncverilog testfixture.v CONV.v*

Gate-Level Simulation 時使用指令如下：

- 使用 ncverilog 模擬指令範例如下：
  ncverilog testfixture.v CONV_syn.v –v tsmc13_neg.v +define+SDF
  define 中加上 SDF 可讓測試程式引入 gate level netlist 的 sdf 檔案資訊。

Transistor-Level Simulation 時使用指令如下：

- 使用 ncverilog 模擬指令範例如下：
  ncverilog testfixture.v CONV_apr.v –v tsmc13_neg.v +define+SDF +ncmaxdelay
  define 中加上 SDF 可讓測試程式引入 transistor level netlist 的 sdf 檔案資訊。

附錄 B 評分用檔案

評分所需檔案可分為以下部份：(1) RTL design，若設計採模組化而有多個設計檔，請務必將使用到的各 module 檔寫進 CONV.v，以免進行評分時，無法進行編譯；(2) gate-level design，即由合成軟體所產生的 gate-level netlist，以及對應的 SDF 檔； (3) transistor-level design，即由 APR 軟體所產生的 transistor-level netlist 及 GDS 檔，以及對應的 SDF 檔；(4) report files，同學必須依照自己的設計內容，填寫 report.000 並另行撰寫 report.doc