

# 專題報告

Group\_K

組長：林揚智 B073012001

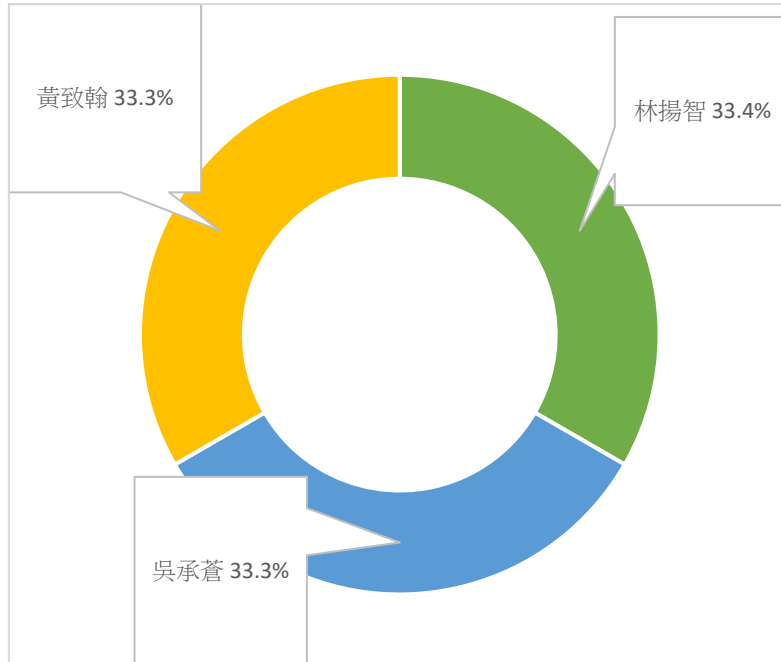
組員：吳承蒼 B083012001

組員：黃致翰 B083012002

# 目錄

<a href="#">工作分配與貢獻</a>	3
一、 <a href="#">摘要</a>	3
二、 <a href="#">專題介紹</a>	4
三、 <a href="#">演算法</a>	5
四、 <a href="#">硬體架構</a>	6
4-1 <a href="#">整體架構</a>	6
4-2 <a href="#">FSM</a>	7
4-3 <a href="#">LAC</a>	8
4-4 <a href="#">GAC</a>	10
4-5 <a href="#">GDM</a>	12
4-6 <a href="#">DS</a>	13
五、 <a href="#">波形模擬</a>	14
5-1 <a href="#">A Level 電路</a>	14
5-2 <a href="#">腳位限縮及新增 TMR 與 Scan Chain 電路</a>	16
六、 <a href="#">可靠度</a>	19
七、 <a href="#">Fault Coverage 評估</a>	20
八、 <a href="#">總結</a>	21
九、 <a href="#">參考文獻</a>	21
十、 <a href="#">心得</a>	21

## 工作分配與貢獻



組員簽名：

林揚智 吳承蒼 黃致翰

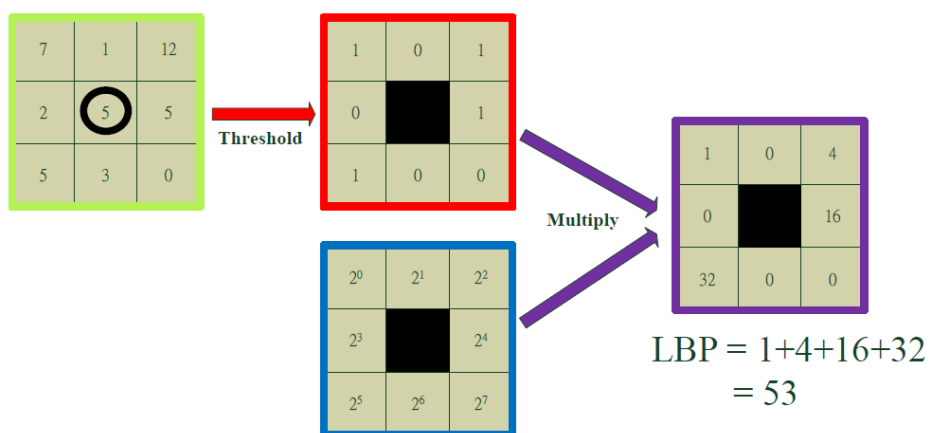
## 一、摘要

此次專題需將 LBP (Local Binary Patterns) 以硬體電路呈現，達成指定的 A Level 等級。完成後將腳位限縮至實際下線時的要求，並使用 innovus 將製作出的電路布局檔實際送到 TSRI 進行下線。最後再將其電路新增可靠度及可測試性功能，以加強電路的實用性。

## 二、專題介紹

LBP 電路運作原理是將一張圖片的每一個 pixel 與周圍的八個 pixel 互相比較大小後，再依序以其各自的權重值相加成為此中心 pixel 的 lbp 值(lbp\_data)，如下圖所示，計算公式如下：

$$LBP(x,y) = \sum_{p=0}^{P-1} s(g_p - g_c) 2^p \quad , \text{ 而 } s(z) = \begin{cases} 1, & z \geq 0 \\ 0, & \text{otherwise} \end{cases}$$



將以上計算方法套用至整張圖片便是此電路的目標。



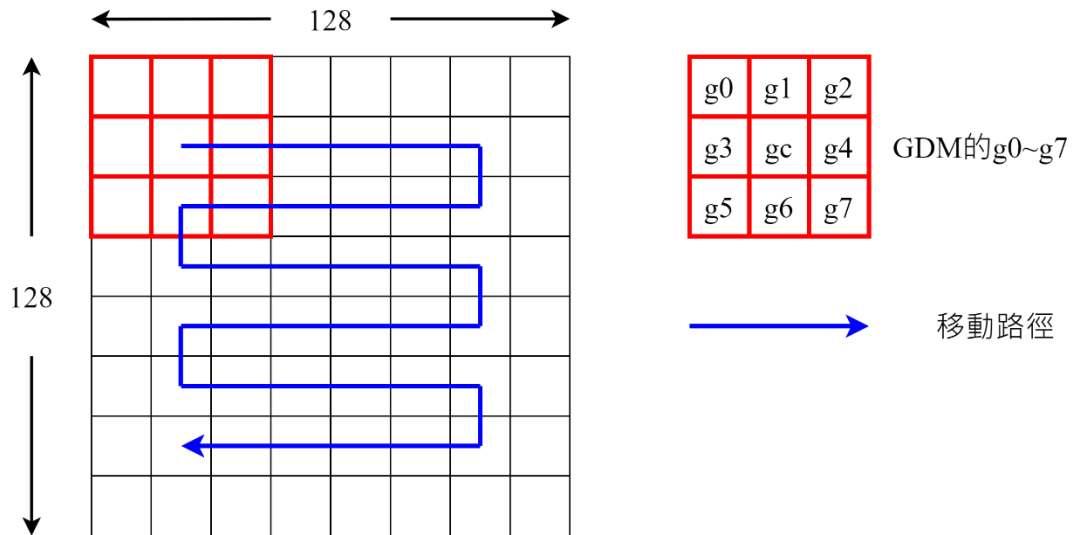
輸入灰階影像



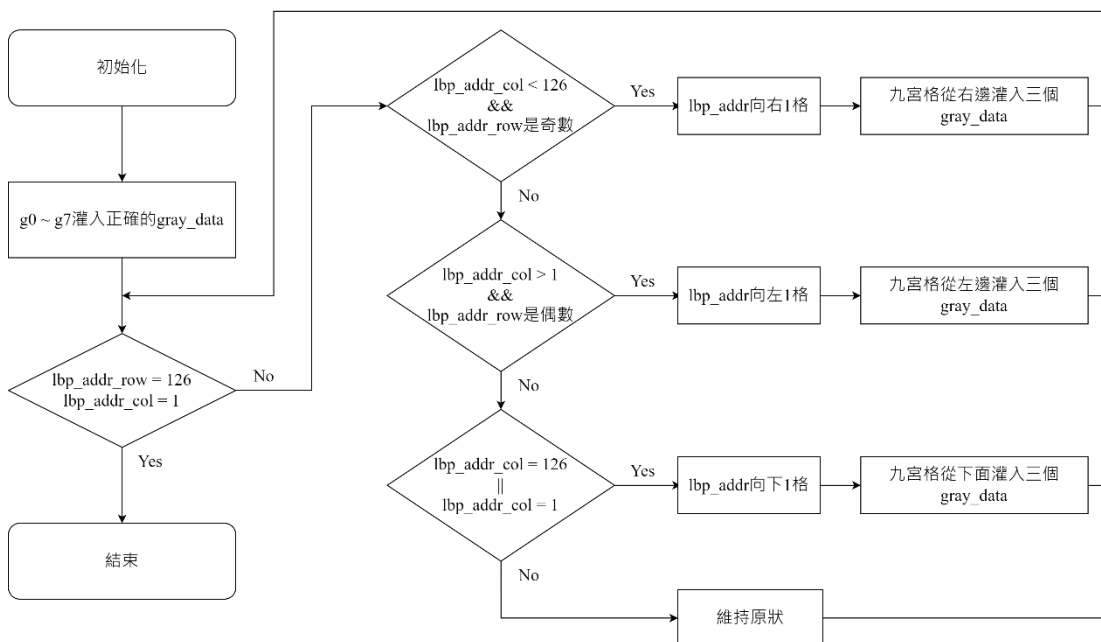
輸出灰階特徵影像

### 三、演算法

在進入演算法前需要先了解九宮格如何移動，設計掃描路線如下：

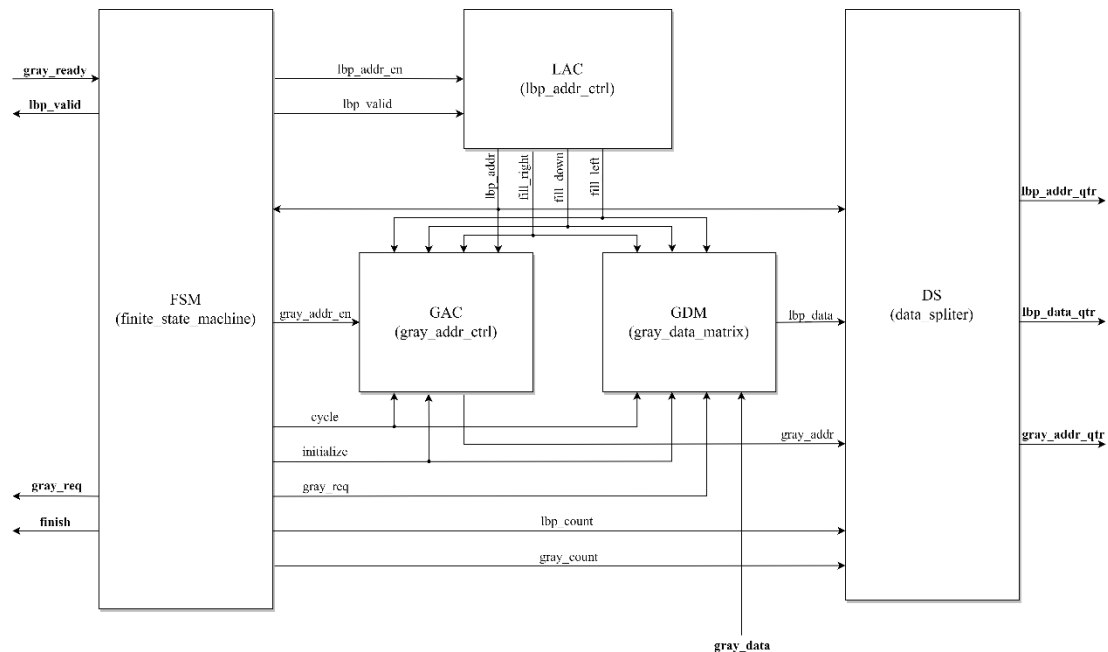


14 位元的記憶體位址前 7 位元可表示 row、後 7 位元表示 column，因此正好如上所說的 128\*128 方格。上圖路徑的優點是九宮格只會向右、向左、或向下移動 1 格，而也就是 row 只會+1，而 column 只會+1 或-1。假如是一行行掃描時，雖然概念上直覺，但需要多一個將 column 變回 1 的機制，使電路可能需要再多出一個狀態，因此才採用此方法。演算法如下：



## 四、硬體架構

### 4-1 整體架構：



#### FSM (finite state machine):

此電路的狀態機，目的為控制 GAC、LAC、GDM 在正確的時間更新正確的值，也是控制 DS 依序輸出 `gray_addr_qtr`、`lbp_addr_qtr`、`lbp_data_qtr` 的正確部分。

#### LAC (lbp address control):

控制 `lbp_addr` 的輸出。`lbp_addr` 即為 GDM 中心 pixel 的位址，移動的方式為由左向右後下降一格再由右向左，以上方法持續到 `lbp_addr` 到達圖片左下角的 pixel 位址為止。

#### GAC (gray address control):

控制 `gray_addr` 的輸出。GDM 需要有 `gray_addr` 向 Host 端(虛擬記憶體)提取正確的 `gray_data`，而 `gray_data` 的取決方式會影響 `lbp_data` 的正確性。才因此此模組需計算出用哪些 `gray_addr` 才能把正確的 `gray_data` 輸入進 GDM，以算出正確的結果。

#### GDM (gray data matrix):

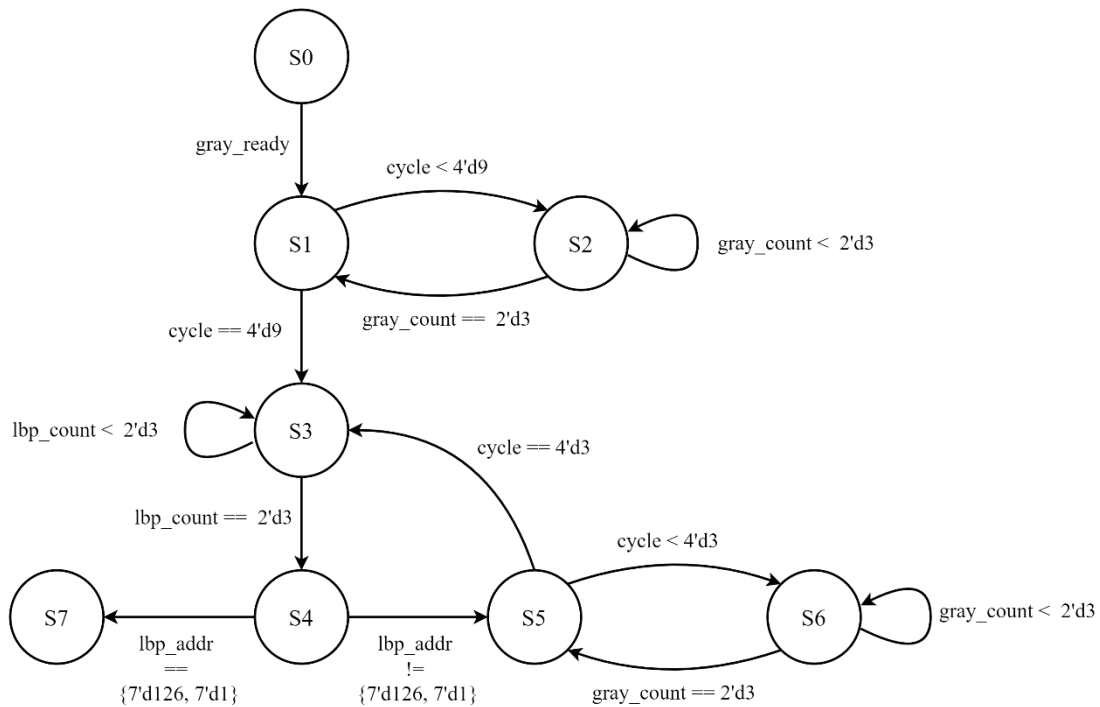
在一開始有提及 `lbp_data` 的計算方法，是將中心 `gray_data` 的值與其鄰近九宮格的其他 8 個 `gray_data` 進行比較，在依各自的權重值相加後得到 `lbp_data`。因此這裡採用了 9 個 DFF 來達成此目的，此九宮格會依據當下一個 `lbp_addr` 的位址及走向來決定此 9 個 DFF 之間要如何去賦予下一個值，讓這 9 個 DFF 看起來就像一個九宮格在向特定方向移動一樣。

※ DS (Data splitter, 此模組在 A Level 電路中未出現，目的為限縮腳位):

此為簡單的一個資料分段區，gray\_addr 與 lbp\_addr、lbp\_data 會分別以 4 對 1 的多工器藉由 gray\_count、lbp\_count 的值依序輸出正確的部分 (gray\_addr\_qtr、lbp\_addr\_qtr、lbp\_data\_qtr)，達到腳位限縮的目的。

※ 原本 A Level 電路有 50 個腳位，限縮後有 24 個腳位

#### 4-2 FSM (finite state machine) :



以上為此電路之狀態圖，各狀態所表示的意義為：

**S0:** 初始狀態

**S1:** 九宮格裝入初始 g0 ~ g7 的 gray\_data 值

**S2:** 將 gray\_addr 分成四等分送給 host 端

**S3:** 將 lbp\_addr 與 lbp\_data 分成四等分送給 host 端

**S4:** 更新 lbp\_addr 位址

**S5:** 抓取相對於 lbp\_addr 的正確 gray\_addr

**S6:** 將 gray\_addr 分成四等分送給 host 端

**S7:** 結束

當 gray\_ready 輸入時，表示 host 端已經準備好接收 lbp 端的資料，因此進入 S1。接下來 lbp 端要控制 gray\_addr 來得到對應的 gray\_data 值，但是腳位限縮的問題，必須將其分成四個部分送出，這也是 S1 與 S2 所做的事。cycle 表示丟出的 gray\_addr 數量，而 gray\_count 則表示丟出 gray\_addr 中四等分的其中一部份，正因為分成四等分，需要讓 gray\_count 從 0 數到 3 才進入到下一個狀

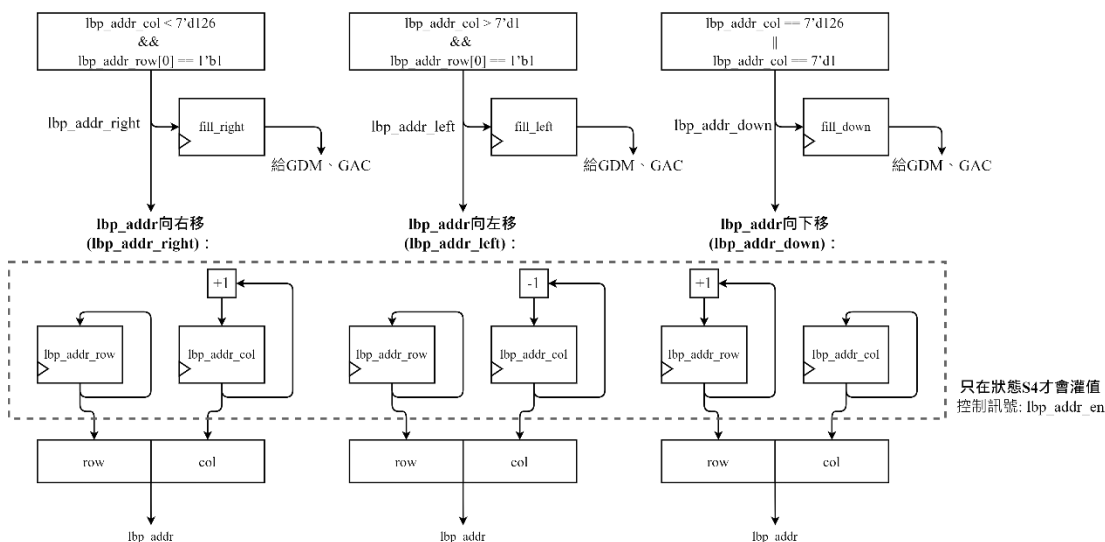
態，另外當 cycle = 9 時，表示已經丟出 9 個 gray\_addr，也表示已經將 g0 ~ g7 填滿了，因此會進入到 S3。

由於 g0 ~ g7 已經填滿，表示 lbp\_data 在那個時刻已經計算出來了，因此需要將 lbp\_addr 與 lbp\_data 傳送給 host 端。但同理，由於腳位限縮的問題，會將 lbp\_addr 與 lbp\_data 分成四等分傳送出去，因此這裡也使用到了 lbp\_count 來區分不同的部分，在當 lbp\_count = 3 時，表示已經將四個部分傳送出去，此時就需要更新 lbp\_addr 而進入到 S4。

S4 會將舊的 lbp\_addr 更新成 LAC 模組所計算好之新的 lbp\_addr，有了新的 lbp\_addr，再加上知道目前九宮格要移動的方向，便可以知道 gray\_addr 所需要的值。更新完 lbp\_addr 後會進入到 S5，此處的 S5、S6 和 S1、S2 扮演的角色相同，是要將 gray\_addr 分成四等分後傳送給 host 端，S5 負責計算新的 gray\_addr，而 S6 負責將四等分送出。當 cycle = 3 時表示已經將新的三個 gray\_data 從正確的方位灌入九宮格，此時又會產生新的 lbp\_data，因此又要回到 S3 來將新的 lbp\_addr 與 lbp\_data 分段送出，如此一直循環下去。

一旦當 lbp\_addr = {7'd126, 7'd1} 時，此位址已經是掃描路徑的最後一格，因此當 lbp\_addr 與 lbp\_data 在 S3 將資料送出後，進入 S4 會滿足 lbp\_addr == {7'd126, 7'd1} 的條件而到 S7，表示電路已經完成工作。

#### 4-3 LAC (lbp address control) :



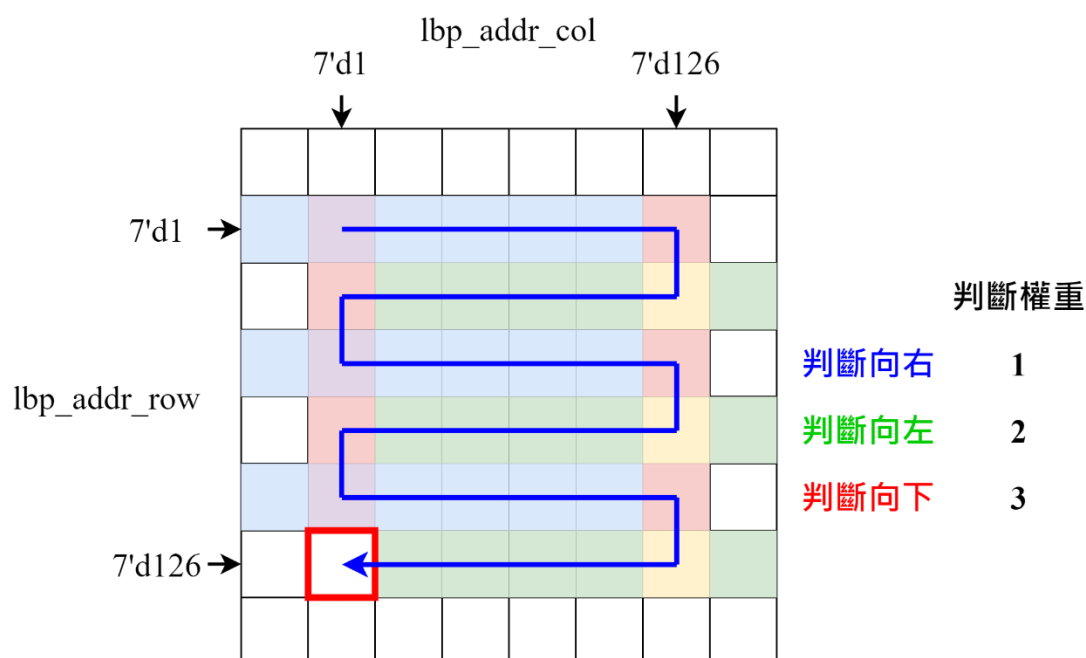
上一個部分有提及此模組的目的是為了計算新的 lbp\_addr。而 lbp\_addr 代表的正好就是九宮格的中心，也就是當下 g0 ~ g7 的 gc 當下所表示的位址。在三、演算法的部分中，有提及九宮格掃描的路徑是反 S 字形掃描，而那條路徑正是 lbp\_addr 一路上所需要產生的值。



此模組會將前 7 位元與後 7 位元分成 lbp\_addr\_row 和 lbp\_addr\_col，因為使用此分法會比起直接使用 14 位元方式更容易觀察行與列之間的關係。此模組的重點便在於轉折處的判斷，也就是判斷何時向下、何時向左、何時向右轉彎。而此處正是使用三、**演算法**所提及的演算法來進行判斷，主要三個判斷依據為：

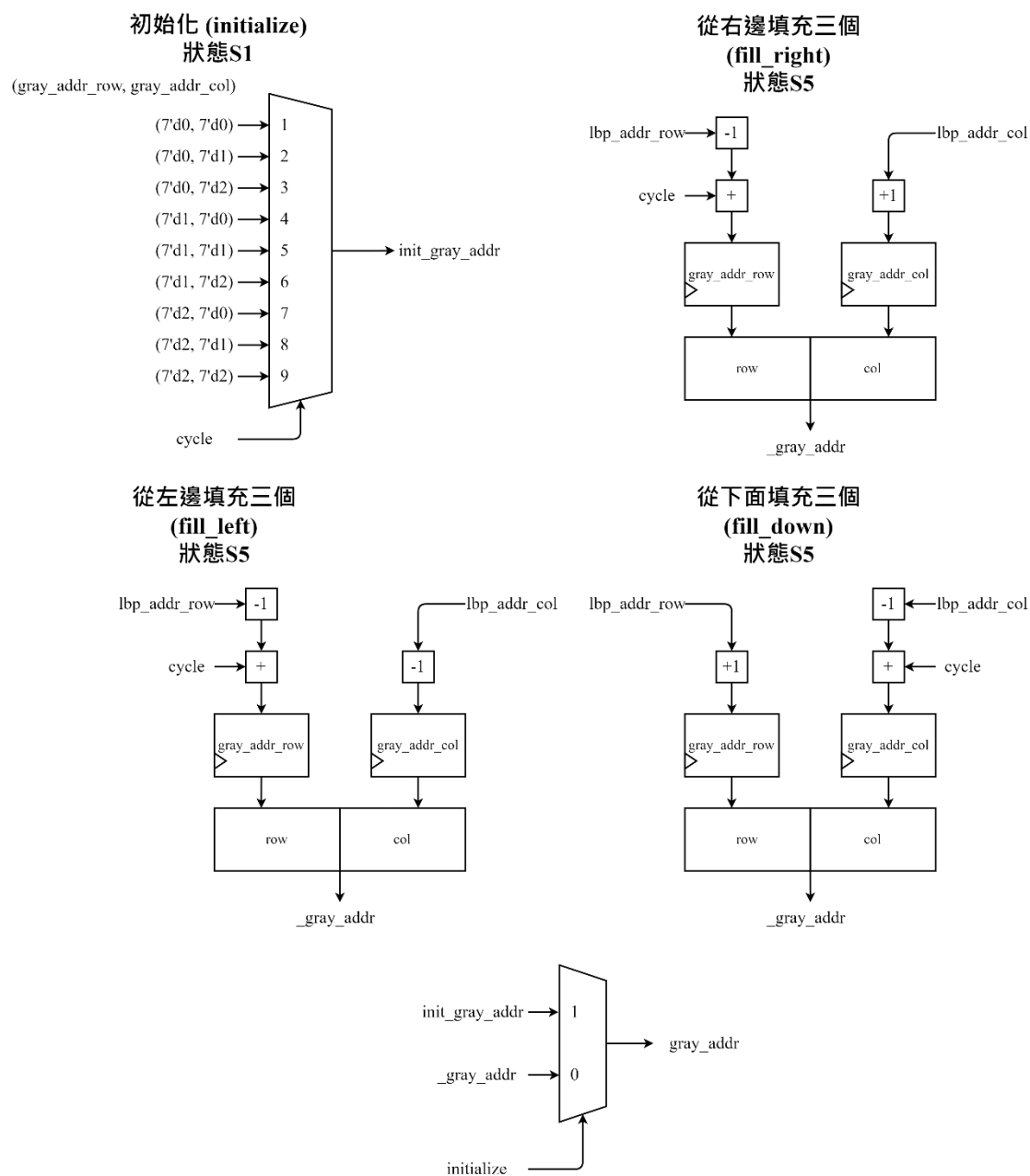
1. `lbp_addr_col < 7'd126 && lbp_addr_row[0] == 1'b1`
2. `lbp_addr_col > 7'd1 && lbp_addr_row[0] == 1'b0`
3. `lbp_addr_col == 7'd126 || lbp_addr_col == 7'd1`

以下以圖片解說：



另外關於 fill\_right、fill\_left、fill\_down 的部分會在下一個部分仔細探討，此三個訊號是在決定 GAC 的 gray\_addr 和 GDM 的 g0~g7 方位判斷的重要資訊，fill\_right、fill\_left、fill\_down 需要比起 lbp\_addr\_right、lbp\_addr\_left、lbp\_addr\_down(前者三者與 gray\_addr 的方位有關，後三者與 lbp\_addr 走向有關)延後一個 clk 以避免產生錯誤的 gray\_addr 以及從錯誤的方位灌入 g0~g7，因此使用 DFF 來錯開一個 clk 來避免此問題。

#### 4-4 GAC (gray address control) :

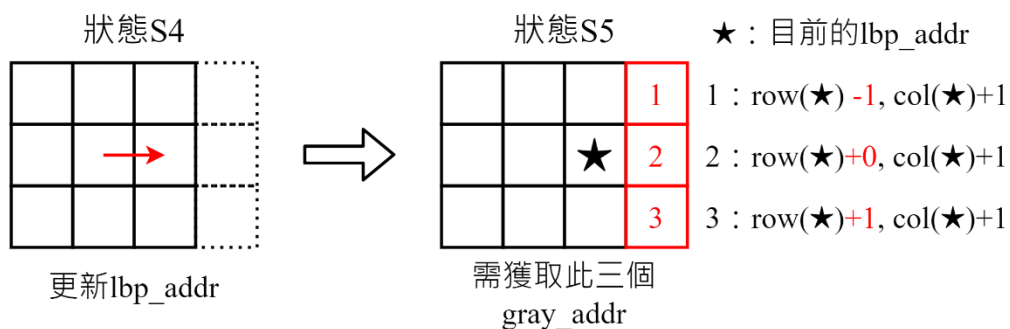


gray\_addr 的變化主要分成兩種形式：1. 初始 g0 ~ g7 的 gray\_addr， 2. 九宮格移動時相對於 lbp\_addr 的 gray\_addr。

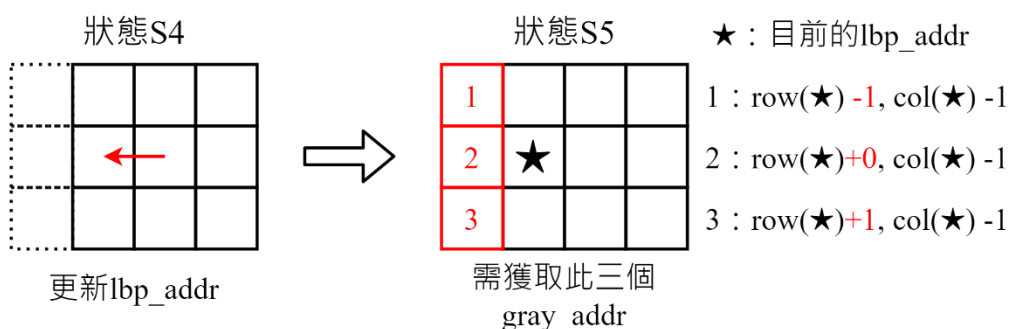
第一種只會發生在狀態 S1，也就是說在初始左上角的九宮格，此處設計直接以一個 MUX 並且使用 cycle 選擇對應下正確的 gray\_addr，並且此時狀態機是會送出 initialize 訊號的，表示最後 gray\_data 的值就是 init\_gray\_data (也就是上圖中最下面的多工器輸出)。

第二種形式需要 lbp\_addr 來做為相對位置的依據，當下會以接收到 fill\_right、fill\_left、還是 fill\_down 來決定 gray\_addr 相對於 lbp\_addr 該如何決定。以下用圖說明：

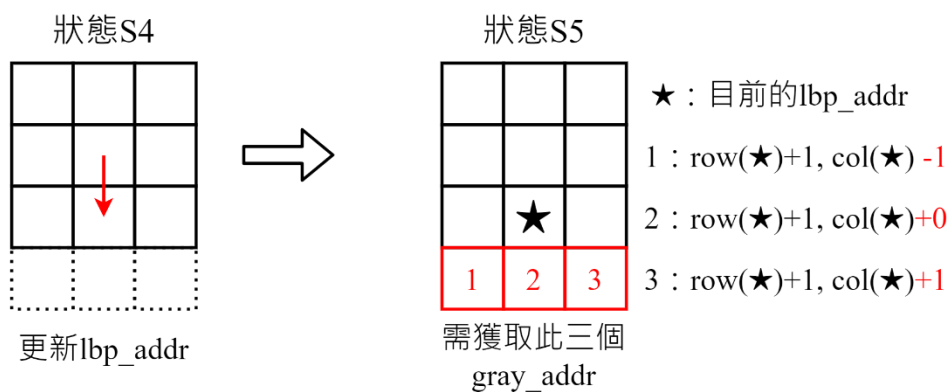
### fill\_right 情況



### fill\_left 情況

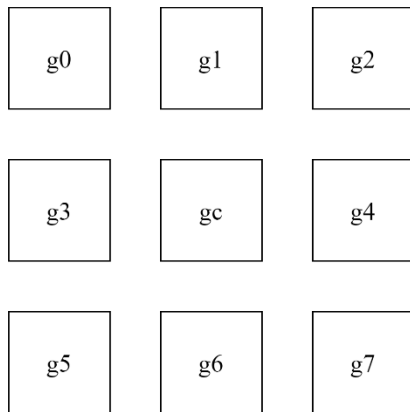


### fill\_down 情況

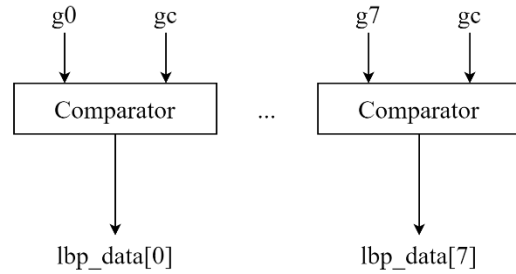


觀察可得 fill\_right 與 fill\_left 的 row 可以直接以  $\text{lbp\_addr\_row} - 1'd1 + \text{cycle}$  表示，而 fill\_down 可以直接以  $\text{lbp\_addr\_col} - 1'd1 + \text{cycle}$  表示，這也正是此模組使用的方法。

#### 4-5 GDM (gray data matrix) :

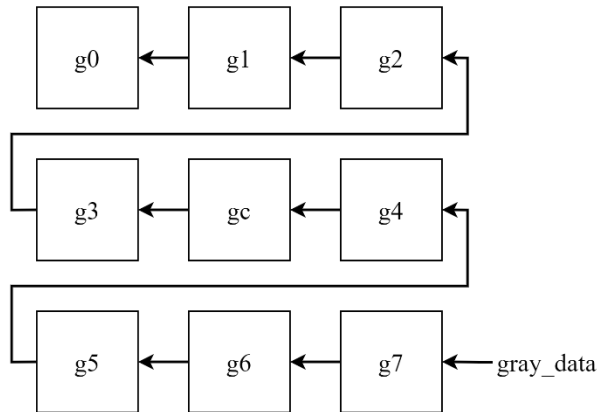


g0 ~ g7裝入對應的gray\_data



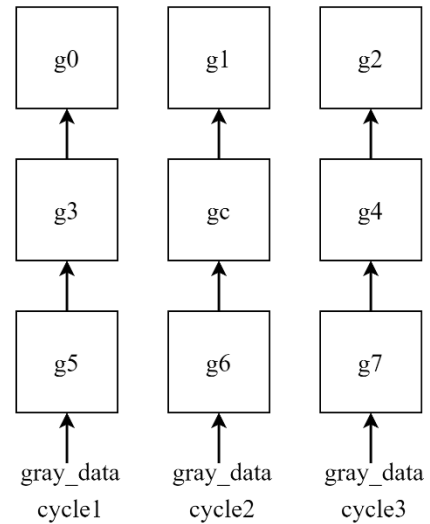
載入初始的g0 ~ g7 :

en = gray\_req & initialize



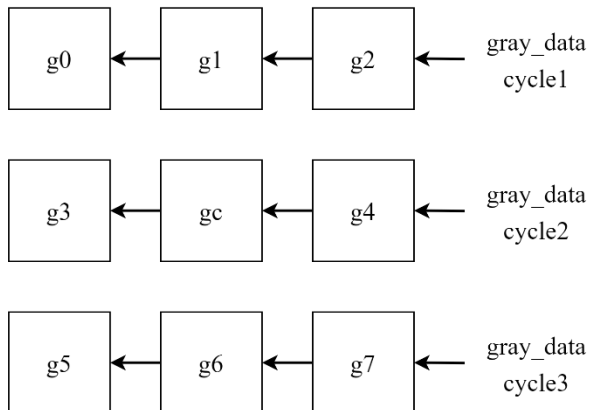
九宮格向下移 :

en = gray\_req & fill\_down



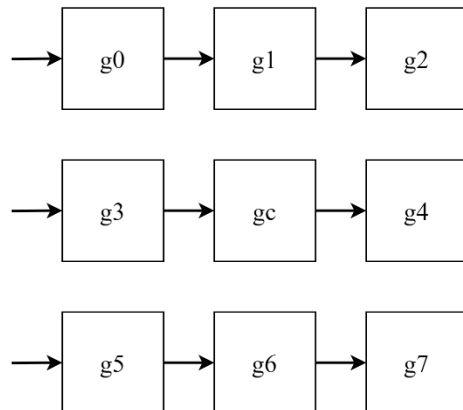
九宮格向右移 :

en = gray\_req & fill\_right



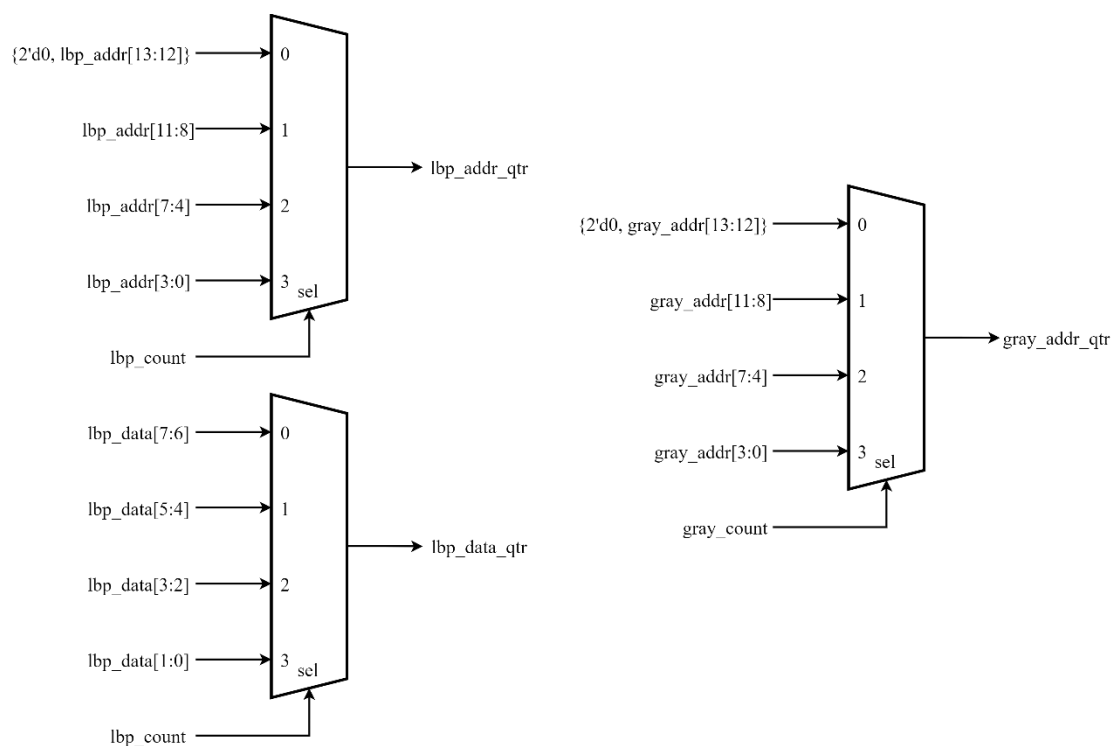
九宮格向左移 :

en = gray\_req & fill\_left



GDM 與 GAC 其實是連貫的，也就是從 GAC 送出 gray\_addr 後得到的 gray\_data 會直接送入 GDM 裡，以供計算 lbp\_data。但這些 gray\_data 灌入的方式與先前提及移動九宮格的方式有些不同，真正達到像九宮格移動的方式，其實是一列列或一行行的灌入 gray\_data 值，並利用 Shift Register 的特性將先前的 gray\_data 值向後方推，讓先前已經花時間獲得的 gray\_data 資料可以留給下一次計算 lbp\_data 時使用。而一行行或一系列列的放式灌入的順序，正是先前在 GAC 說明圖所使用的順序，因此過了 3 個 cycle 後，lbp\_data 就可以計算完成，隨後再將 lbp\_valid 變成 1 就可以讓 host 端接收到正確的 lbp\_data 和 lbp\_addr 值。

#### 4-6 DS (Data splitter) :

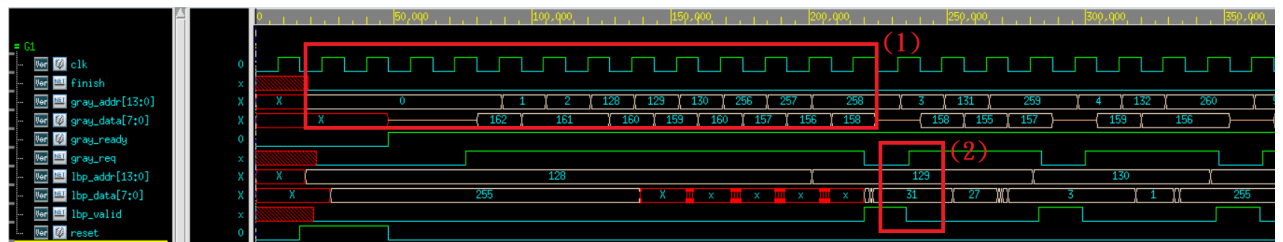


此模組的目的，便是將原本 gray\_addr、lbp\_addr 從 14 個腳位限縮成 4 個腳位，以及將 lbp\_data 從 8 個腳位限縮成 2 個腳位。使用的方法便是使用 4to1 的 MUX，並且以 gray\_cout 以及 lbp\_count 選擇送出的部分。

## 五、波形模擬

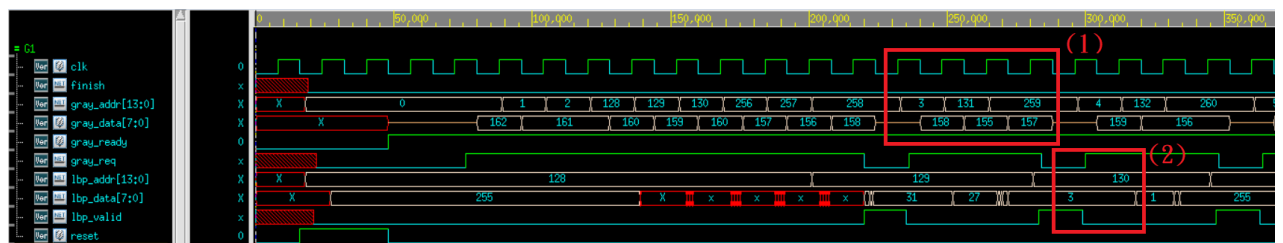
針對波形模擬，主要分為兩個電路來說明：

### 5-1 A Level 電路：



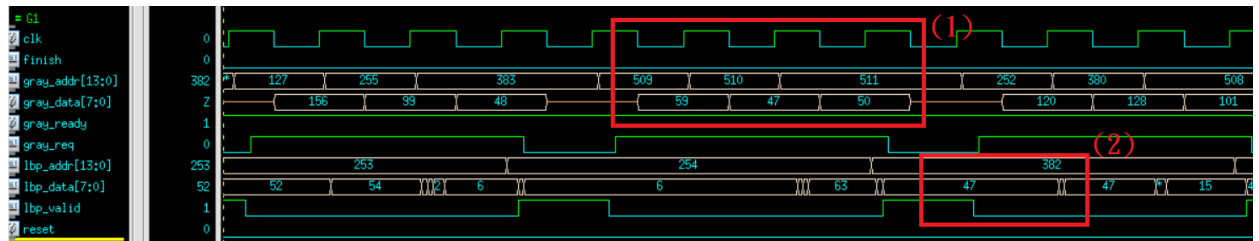
#### [說明]

電路起初運作會先對最左上角之 9 個 gray\_addr 依序索取其對應的 gray\_data，即上圖編號 1 之紅框處。接著，進行 LBP 運算後，將 lbp\_valid 拉起為 1，讓運算值存入九宮格中心點位址(lbp\_addr)所對應的資料(lbp\_data)，上圖編號 2 之紅框處表示於 lbp\_addr = 129 存入 lbp\_data = 31。



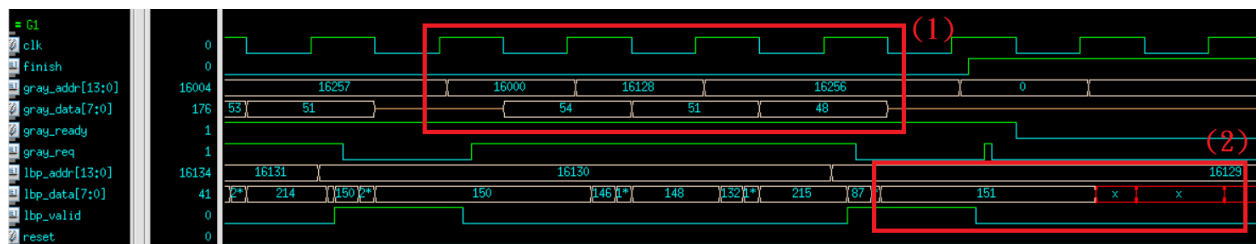
#### [說明]

之後計算每一個中心點的 lbp\_data，依照移動方向來更新前一個九宮格中的三個 gray\_addr，重新存取對應的 gray\_data 後進行 LBP 運算。同樣地，進行 LBP 運算後，將 lbp\_valid 拉起為 1，讓運算值存入九宮格中心點位址(lbp\_addr)所對應的資料(lbp\_data)。上圖編號 1 之紅框處為原九宮格**向右位移**後，更新最右排 gray\_addr = 3、131、259，重新進行 LBP 運算後，上圖編號 2 之紅框處表示於 lbp\_addr = 130 存入 lbp\_data = 3。



#### [說明]

當整列 LBP 運算完畢並存入該列所有中心點之 lbp\_data 後，即須將九宮格**向下位移**。上圖編號 1 之紅框處表示向下位移需要更新九宮格最下排，也就是 gray\_addr = 509、510、511。同樣透過拉起 gray\_req 為 1 來索取該位址對應之 gray\_data，再進行 LBP 運算，上圖編號 2 之紅框處表示於 lbp\_addr = 382 存入 lbp\_data = 47。



#### [說明]

依照上述所說明的流程重複進行至最後一個 LBP 運算，將倒數第二個九宮格**向左位移**，更新九宮格中的最左排 gray\_addr = 16000、16128、16256，即上圖編號 1 之紅框處。接著，同樣進行 LBP 運算，上圖編號 2 之紅框處表示於 lbp\_addr = 16129 存入 lbp\_data = 151。最後，拉起 finish 訊號為 1 以通知 Host 進行資料比對。

#### [小結]

Combinational area:	4963.197571
Buf/Inv area:	258.004796
Noncombinational area:	3416.866085
Macro/Black Box area:	0.000000
Net Interconnect area:	undefined (No wire load specified)
Total cell area:	8380.063656
Total area:	undefined

```
140.117.166.165 - PuTTY

Output pixel: 0 ~      12000 are correct!
Output pixel: 0 ~      13000 are correct!
Output pixel: 0 ~      14000 are correct!
Output pixel: 0 ~      15000 are correct!
Output pixel: 0 ~      16000 are correct!
Output pixel: 0 ~      16383 are correct!

-----

Congratulations! All data have been generated successfully!

-----PASS-----

Simulation complete via $finish(1) at time 1016272 NS + 0
./testfixture.v:130      #(`CYCLE/2); $finish;
ncsim> exit
soc05 [~/modules/test]
-UISG111a01- $
```

Timing = 1,016,272、Area = 8380.063656

→Score = 8,516,424,052 < 12,000,000,000。(滿足要求)

## 5-2 腳位限縮及新增 TMR 與 Scan Chain 電路：

### [說明]

由於下線時必須考慮到有限的腳位數量，因此，我們針對 gray\_addr[13:0]、lbp\_addr[13:0]和 lbp\_data[7:0]限縮腳位，再透過 4 個 cycle 來完整傳輸。我們將 14-bit gray\_addr 和 lbp\_addr 縮成 4-bit；8-bit lbp\_data 縮成 2-bit。因此，測試檔 testfixture 也必須相對應修改成：當 gray\_addr 經過 4 個 cycle 傳輸後，拉起 gray\_req 訊號為 1，當下的 gray\_addr 才是有效位址，lbp\_addr 和 lbp\_data 亦是如此，同樣經過 4 個 cycle 傳輸後，拉起 lbp\_valid 為 1，當下的 lbp\_addr 才是有效位址，再將運算值正確存入該位址。測試檔 testfixture 新增的部分如下圖：

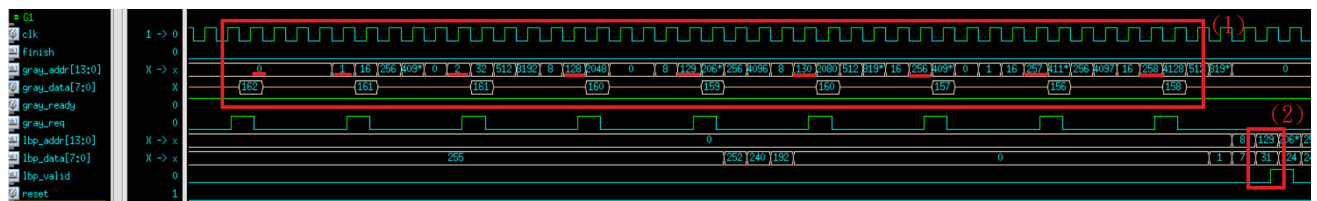
```
173 // New feature-----
174 module data_combiner(gray_addr_qtr, lbp_data_qtr, lbp_addr_qtr, gray_addr, lbp_data, lbp_addr, clk);
175 input      clk;
176 input  [3:0] gray_addr_qtr, lbp_addr_qtr;
177 input  [1:0] lbp_data_qtr;
178 output [13:0] gray_addr, lbp_addr;
179 output [7:0]  lbp_data;
180
181 reg  [3:0] gray_addr_qtr3, gray_addr_qtr2, gray_addr_qtr1, gray_addr_qtr0;
182 reg  [3:0] lbp_addr_qtr3, lbp_addr_qtr2, lbp_addr_qtr1, lbp_addr_qtr0;
183 reg  [1:0] lbp_data_qtr3, lbp_data_qtr2, lbp_data_qtr1, lbp_data_qtr0;
184
185 assign gray_addr = {gray_addr_qtr3[1:0], gray_addr_qtr2, gray_addr_qtr1, gray_addr_qtr0};
186 assign lbp_addr  = {lbp_addr_qtr3[1:0], lbp_addr_qtr2, lbp_addr_qtr1, lbp_addr_qtr0};
187 assign lbp_data  = {lbp_data_qtr3[1:0], lbp_data_qtr2, lbp_data_qtr1, lbp_data_qtr0};
188
189 always @(negedge clk) begin
190     gray_addr_qtr3 <= gray_addr_qtr2;    lbp_addr_qtr3 <= lbp_addr_qtr2;    lbp_data_qtr3 <= lbp_data_qtr2;
191     gray_addr_qtr2 <= gray_addr_qtr1;    lbp_addr_qtr2 <= lbp_addr_qtr1;    lbp_data_qtr2 <= lbp_data_qtr1;
192     gray_addr_qtr1 <= gray_addr_qtr0;    lbp_addr_qtr1 <= lbp_addr_qtr0;    lbp_data_qtr1 <= lbp_data_qtr0;
193     gray_addr_qtr0 <= gray_addr_qtr;    lbp_addr_qtr0 <= lbp_addr_qtr;    lbp_data_qtr0 <= lbp_data_qtr;
194 end
```



### [DFT 說明]

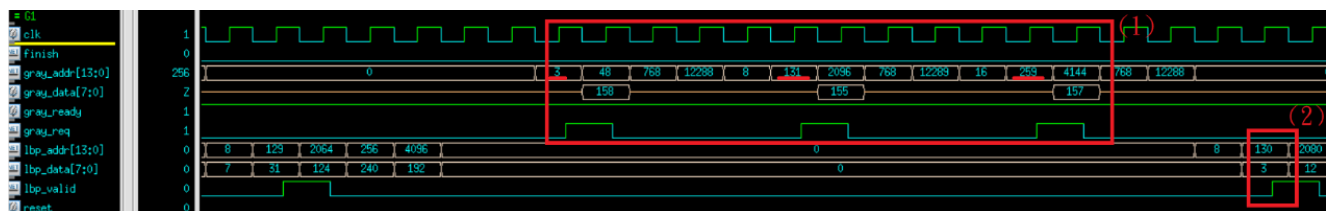
在考慮 DFT 時，會將 DFF 置換成 Sdff，同時也會多出 scan enable(SE)、scan input(SI)和 scan output(SO)三個腳位。因此，對測試檔 testfixture 執行 ncverilog 指令時，欲使電路操作在正常模式，必須要在測試檔 testfixture 中另外定義這三個變數。另外因為 Sdff 是使用於測試的情況，與現在 testfixture 的驗證不同，因此將 SI 設為 0，SE 也設為 0(Normal Mode)，而 SO 可以閒置沒有關係。

```
41 // DFT ports
42 reg test_se = 0;
43 reg SCAN_IN_1 = 0;
44 wire SCAN_OUT_1;
```



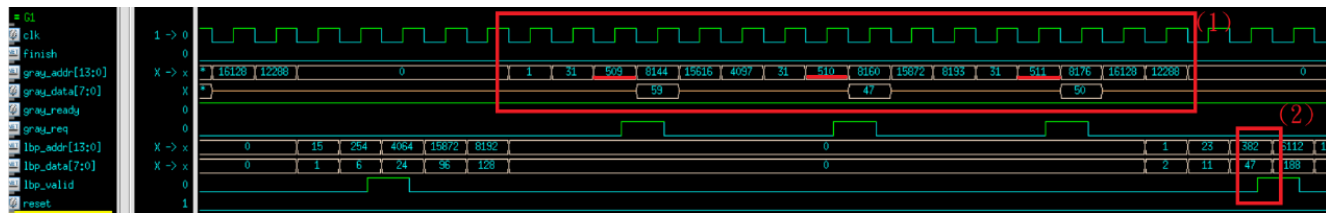
### [說明]

由於 gray\_addr 採用串列輸入的緣故，當依序存取 gray\_addr 所對應的 gray\_data 過程中，必須多等 4 個 cycles，再拉起 gray\_req 為 1，才能確保索取正確 gray\_addr 對應的 gray\_data。上圖編號 1 之紅框處為起初運作時，對最左上角之九個 gray\_addr 依序存取其對應的 gray\_data。明顯地，與第一個電路相比所需之 cycle 數增加不少。接著，進行 LBP 運算後，一樣必須多等 4 個 cycles，lbp\_valid 才拉起為 1，讓運算值存入九宮格中心點位址(lbp\_addr)與其對應的資料(lbp\_data)，上圖編號 2 之紅框處表示於 lbp\_addr = 129 處存入 lbp\_data = 31。



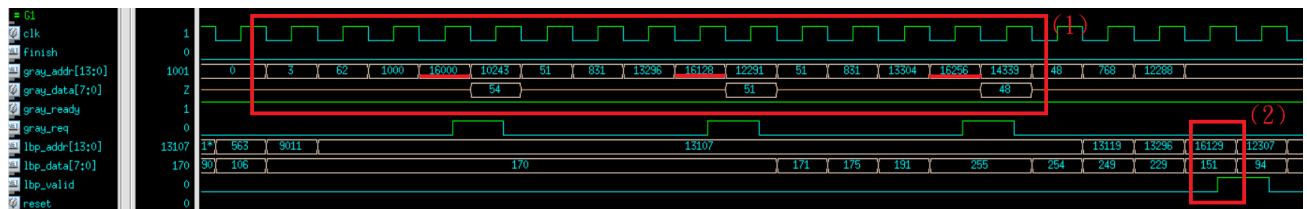
### [說明]

之後計算每一個中心點的 lbp\_data，皆是依照移動方向來更新前一個九宮格中的三個 gray\_addr，同樣因為串列輸入的緣故，必須等待完整的 gray\_addr 輸入後，再拉起 gray\_req 為 1，索取正確位址所對應的 gray\_data 後進行 LBP 運算。最後，將 lbp\_valid 拉起為 1，讓運算值存入更新後的九宮格中心點位址 (lbp\_addr)所對應的資料(lbp\_data)。上圖編號 1 之紅框處為原九宮格**向右位移**後，更新最右排 gray\_addr = 3、131、259，重新進行 LBP 運算後，上圖編號 2 之紅框處表示於 lbp\_addr = 130 存入 lbp\_data = 3。



### [說明]

當整列 LBP 運算完畢並存入該列所有中心點之 lbp\_data 後，即須將九宮格**向下位移**。上圖編號 1 之紅框處表示向下位移需要更新九宮格最下排，也就是 gray\_addr = 509、510、511。同樣因為串列輸入的緣故，必須等待完整的 gray\_addr 輸入後，再拉起 gray\_req 為 1，以索取該正確位址對應之 gray\_data。再進行 LBP 運算，上圖編號 2 之紅框處表示於 lbp\_addr = 382 存入 lbp\_data = 47。



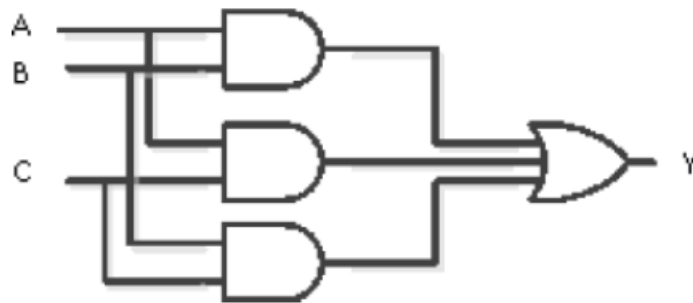
### [說明]

依照上述所說明的流程重複進行至最後一個 LBP 運算，將倒數第二個九宮格**向左位移**，更新九宮格中的最左排 gray\_addr = 16000、16128、16256，一樣因為串列輸入的緣故，必須等待完整的 gray\_addr 輸入後，再拉起 gray\_req 為 1，以索取該正確位址對應之 gray\_data，即上圖編號 1 之紅框處。接著，同樣進行 LBP 運算，上圖編號 2 之紅框處表示於 lbp\_addr = 16129 存入 lbp\_data = 151。最後，拉起 finish 訊號為 1 以通知 Host 端進行資料比對。

## 六、可靠度

我們認為，對於我們所設計的 LBP 電路而言，經常會需要針對記憶體的位置，來索取或存取該位址所對應的資料。特別像是當我們的電路因為腳位的限制而將 14-bit 的 gray\_addr 改用 4-bit 來進行串列傳輸，因此，提供正確記憶體位址之重要性更是我們提高可靠度的一大方向。

於是，我們將 TMR(Triple Module Redundancy)的設計概念應用在子電路 GAC(gray address control)和 LAC(lbp address control)當中的 gray\_addr[13:0]和 lbp\_addr[13:0]。目的希望透過額外複製兩份電路，再將各個 bit 經過投票機電路來決定最終的輸出值，使其中一個電路發生錯誤時仍能正常運作。



但是新增 TMR 也意味著電路成本的上升，一旦新增了電路便無法避免面積的上升，甚至也有可能造成最高運行速度的下降。如下圖所示，原本 A Level 電路面積為  $8380\mu\text{m}^2$ ，但是加上了 TMR 後面積上升至  $12792\mu\text{m}^2$  (再加上 SDF 會變成  $14292\mu\text{m}^2$ )，約增加了  $12792 - 8380 / 5 = 883$  個 NAND 邏輯閘大小。另外因為最長路徑的增加使 testfixture 也只能將 CYCLE 調成 20ns，此為新增了 TMR 可靠度需付出的代價。

Combinational area:	7286.938192
Buf/Inv area:	424.349993
Noncombinational area:	5504.668018
Macro/Black Box area:	0.000000
Net Interconnect area:	undefined (No wire load specified)
Total cell area:	12791.606210
Total area:	undefined

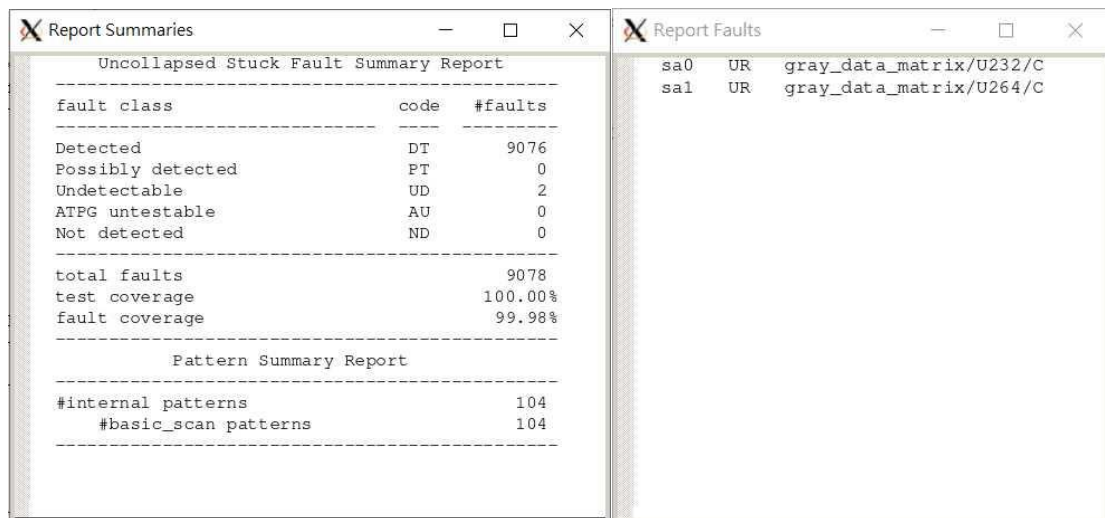
## 七、Fault Coverage 評估

在進行合成前需將原本的 tcl script 加上：

```
set_dft_signal -view existing_dft -type reset -port [find port "reset"] -active_state 1
```

此指令可以去除原本對於非同步 reset 的限制。

完成加入 scan chain 後，即可使用 tetra max 來進行 ATPG。由於是 Full scan，可以使用 combinational ATPG 的方式對我們的電路進行 fault coverage、test coverage 等等的評估，結果如下：



fault class	code	#faults
Detected	DT	9076
Possibly detected	PT	0
Undetectable	UD	2
ATPG untestable	AU	0
Not detected	ND	0
total faults		9078
test coverage		100.00%
fault coverage		99.98%

#internal patterns	104
#basic_scan patterns	104

sa0	UR	gray_data_matrix/U232/C
sa1	UR	gray_data_matrix/U264/C

由圖可見，電路一共產生了 9078 個 single stuck-at faults，而上圖顯示 test coverage 為 100%，表示我們除了 undetectable fault 之外，其他 fault 均可被測試到。而我們的 fault coverage 為 99.98%，表示我們有"極"少數的 undetectable fault 存在於我們的電路當中，我們認為站在可測試性的立場來思考，我們重視的是 100% 的 test coverage，也就是觀察是否所有的 detectable fault 都能被測得；反觀 undetectable fault 並不會造成電路發生錯誤。然而，對於存在 undetectable fault 即代表電路當中存在 redundancy hardware，也值得我們深入探討是否有特殊的設計考量，而必須要採用這樣的方式。

## 八、總結

關於本次 LBP 的電路製作，可看見每當要新增一項功能時，都必須將電路的成本納入考慮。以新增可靠度功能為例，原本的電路面積只需  $8380\mu\text{m}^2$ ，但是加上了 TMR 電路後，面積的需求就上升至  $12792\mu\text{m}^2$ ，並且不僅僅是面積的上升，電路的最長路徑也會因此造成影響。再以新增可測試性功能為例，新增可測試性的電路加上了 Scan Chain 後，面積又從  $12792\mu\text{m}^2$  上升  $14292\mu\text{m}^2$ 。但新增功能後換來的好處就是可靠度上升，電路不會因為一個區塊的不正常運作就導致整體電路的錯誤計算，並且一旦有錯誤時，是可以藉由可測試性的功能知曉。因此可以知道電路的設計永遠離開不了利害之間的取捨，而厲害的電路便是可以找到眾多變因的平衡點，並且將其優點得來的效益彌補本身無法避免的缺點，這也是此電路所具有的價值。

## 九、參考文獻

本次專題並未參考任何文獻。

## 十、心得

林揚智：

首先，想要謝謝兩位隊員每一次的討論與合作，很高興在上學期實用數位系統設計的課程中能夠和兩位一起努力，感謝這緣分能夠延續到之後大學專題的製作，也對於最後晶片實作感到相當興奮。同時，也很感謝帶領我們的專題學長 堡崧，謝謝你在我們遇到問題時都能及時提供建議與解決辦法，也時時地提醒我們進度和肯定我們的一切。

有了實用數位設計這門課期末專題的經驗後，我們在 LBP 電路的設計上，遇到的難題減少了許多，經過幾次的概念發想和討論後，直接進行電路實現。幸運地，電路合成後也順利達到 IC 競賽 A-level 的標準，這對我們來說，無疑是一大鼓勵和認可。在做專題的過程中，如同遊戲一關一關克服一樣，我們緊接著要面對的是下線要求，除了腳位上的限制之外，更要配合 TSRI 的時程以及通過晶片的各項驗證。此次 cell-based 的下線流程是我們第一次。過去曾聽過一句話：「Divide and Conquer」，我認為相當適合運用在我們面對各種下線要求。首先，針對多位元的輸入改用串列輸入的方式進行縮腳位；反覆調整 clk 來通過 innovus 的佈局設計；輸入各項指令來完整通過所有 DRC。當時間不斷推進至下線期限時，我們也逐步完成各項要求。然而，在上傳至雲端 DRC 驗證

時，卻有其他的錯誤發生。這個難關我們找尋了各種可能導致的原因，也詢問了學長這個問題。最後始終無法完全消除那些跑過雲端 DRC 後出現的錯誤，即便如此，我們仍然盡量去滿足要求，至於其他無法控制的，就交由命運吧，想不到做專題也能學到跟哲學有關的想法。值得一提的是，我們也不出所料遭到退件，後來也發現是我們繳交的 TRF 版本過舊，重新於補件時間上傳後便順利通過下線申請。

在晶片下線出去後，我們持續針對可靠度和 DFT 的設計進行設計，一開始，總是會認為我們才剛學不久，「我們有辦法完成嗎？」諸如此類的質疑不斷出現。這邊也要再次感謝學長 堡歲的意見提供，指引我們可以努力的方向，同時也別想太多，做就對了。我認為專題製作的實戰經驗對於一門課程的掌握度有相當大的幫助，不僅檢視自己在學習上是否有誤解或是盲點，更是訓練將書上所傳授的概念實際運用在真實電路當中的能力。

最後，我認為跑完整個 cell-based 的晶片設計流程，更確定自己未來想往晶片設計的方向努力，不單單只是因為薪水較多，更多的是實現自我價值與創造力展現。前陣子，剛經歷了研究所推甄，也很幸運在最後加入了研究 SerDes 的實驗室，即便自己未來朝向的方向較偏 Full-custom(全客戶式)的類比電路設計，我始終認為這此的專題經驗給了我不少收穫。同時，也很期待任何之後晶片下線回來會遇到需要克服的種種，我堅定地相信，我們是有辦法可以一一解決的！

### 吳承蒼：

這次的專題可以說是將上學期從實用數位設計這門課學到的知識再進一步去運用，也幸好有實用數位這門課程，在軟體的操作上不會有太大的問題。這次的題目我們選擇了 LBP，也就是局部二值模式，當初會選擇這個題目也可以說是覺得有趣，想嘗試原本可能以軟體實現的程式改成以硬體架構的形式設計出來。設計 LBP 的過程中意外地挺順利的，大約花了 2 至 3 天就將電路達到了題目需求的 A Level 等級，也許是因為我們在實用數位的期末專題花了非常多時間去捉摸細微的地方，讓我們在設計上可以預期應該要出現的結果。

接下來就是關於下線的部分，我認為這是這次專題裡面最困難的，也許是陌生的關係，花了不少時間在這上面。首先是腳位限縮的問題，由於教育晶片只有 40 個腳位但是又有一些腳位是需要分配給電源的，因此減少下來只剩下 24 個腳位，而原本的 LBP 電路卻是 50 個腳位，整整是兩倍再多一點。幸好在助教的幫助下找到的一個簡單的解決方法，也就是使用多工器將某些輸出分成四等分，經過這個方法之後我們就剛剛好把 50 腳位的電路成功限縮成正好 24 個腳位。當然限縮腳位後測試檔 testfixture 也要跟著修改，而我們使用的方法就是利用 Shift Register 的方式將分段的訊號拼起來，也挺幸運的修改的過後跑一次就成功了。電路驗證完成後，就是合成與 innovus 的部分，這部分我認為是最折磨人的地方，關於合成的部分，多了許多沒看過的指令但是基本上跟循著

講義不會出太大的錯誤。但是到了 innovus 階段就像是驗收成果的地方，因為這裡可以知道合成出來的電路經過 layout 後 WNS 是否還在允許範圍，一旦不行的話就只能重新從合成再來一遍，並且最令人頭痛的就是 innovus 運作的速度，比起其他軟體慢了許多，因此這方面我們也花的非常多時間。

關於下線的部分結束後，就是新增可靠度及可測試性的部分，這部分沒有花太多的時間，因為可靠度的部分我們有預習相關的內容，而我們選擇的方案就是 TMR，硬體上的實現也非常簡單。最後的部分就是可測試性，正好這學期有學習相關內容，因此加上 Scan Chain 時也算相當順利，雖然有遇到非同步 reset 的問題，但經詢問助教後只需加一行指令就可以避免了。這次的專題讓我體驗到了一個晶片製作需經過哪些流程，以及如何提高電路的可靠度及可測試性，可以說是收穫相當豐富。

### 黃致翰：

在上學期修完實用數位系統設計之後，也對設計一個數位電路有基本的了解了，於是就想說接著來做老師的專題。在一開始設計電路的時候，我們反而比實用數位系統的期末專題還更快就做出來了，可能是因為實用數位的幫助很大吧，我們也迅速的完成了 A-level 的要求。然而，這只是一開始所完成的一個小小目標而已，之後還要讓我們的這顆電路可以實際下到晶片上面。

關於下晶片這塊，我們只有以前在課堂上有聽過老師提起，實際上要下晶片的要求比我們想像中嚴格。最一開始，我們迷迷糊糊的認為只要按照步驟做就可以完成了，卻沒想到因為晶片腳位數量限制的關係，我們必須把我們的電路腳位縮到只能有 40 隻，其中還包含了電源腳位，我們在縮腳位的這個區塊卡了相當長的一段時間，因為濃縮出來的電路必須要比較少的輸入輸出，功能上面又得完全一樣，好在後來，我們犧牲了些許的操作頻率成功換取了腳位濃縮成功。

下一個挑戰就是在做晶片 layout 時，我們花了相當長的時間在操作跟熟悉 innovus 這套軟體，在做的過程中我們可以說是 try and error，不斷的遇到了許多錯誤，像是在跑錯誤列表示發現有些錯誤並不是可容許的假錯，我們回頭思考發現可能是電路的哪邊出了問題，因此又修改了一下電路，而修改電路就意味著整個下線流程要重新跑一遍，我們也在這之中投入了最多的時間跟精力，幸好最後在助教及組員的協助之下，成功的把下線東西送到了 TSRI，而後面也非常幸運的通過了 TSRI 的流程，成功的被排到了下線名單之中。

而最後，因為老師的專題要求，我們還得加入高可靠度的設計才算是完成我們專題，而加入高可靠度會非常地吃電路面積，我們只是加了一個小小的可靠度設計就讓電路面積極大幅度的增加，想必也是老師說之後在業界的挑戰，因為以後在業界，所有的電路一定會有高可靠等等的設計，絕非只有設計電路那麼的簡單，透過這次專題我也算是對業界有非常初步的認識了。

最後，我要非常的感謝我的組員，有了他們，時常救了迷迷糊糊的我，讓

我可以搞懂現在在幹嘛，如果沒有他們，要一個人完成這些東西，我覺得有非常大的困難，勢必得花更多的時間精力，有了他們簡直都是事半功倍。而最後，更是感謝帶我們的助教學長，都不辭辛勞地回答我們遇到的各種疑難雜症，即使有時候只是我們自己犯的一些小小粗心錯誤，助教也會耐心地回答我們，我們有問題助教也都是回我們問題回的非常的快速，真的是非常感謝助教的幫忙與指導。