**110-1 數電實驗 Final Project Report**

**Hand-Motion Controlled Doodle Jump**

**Team07** B08901061陳永縉 B08901062張家翔 B08901102李允恩

**層級架構**

1. Motion Detection

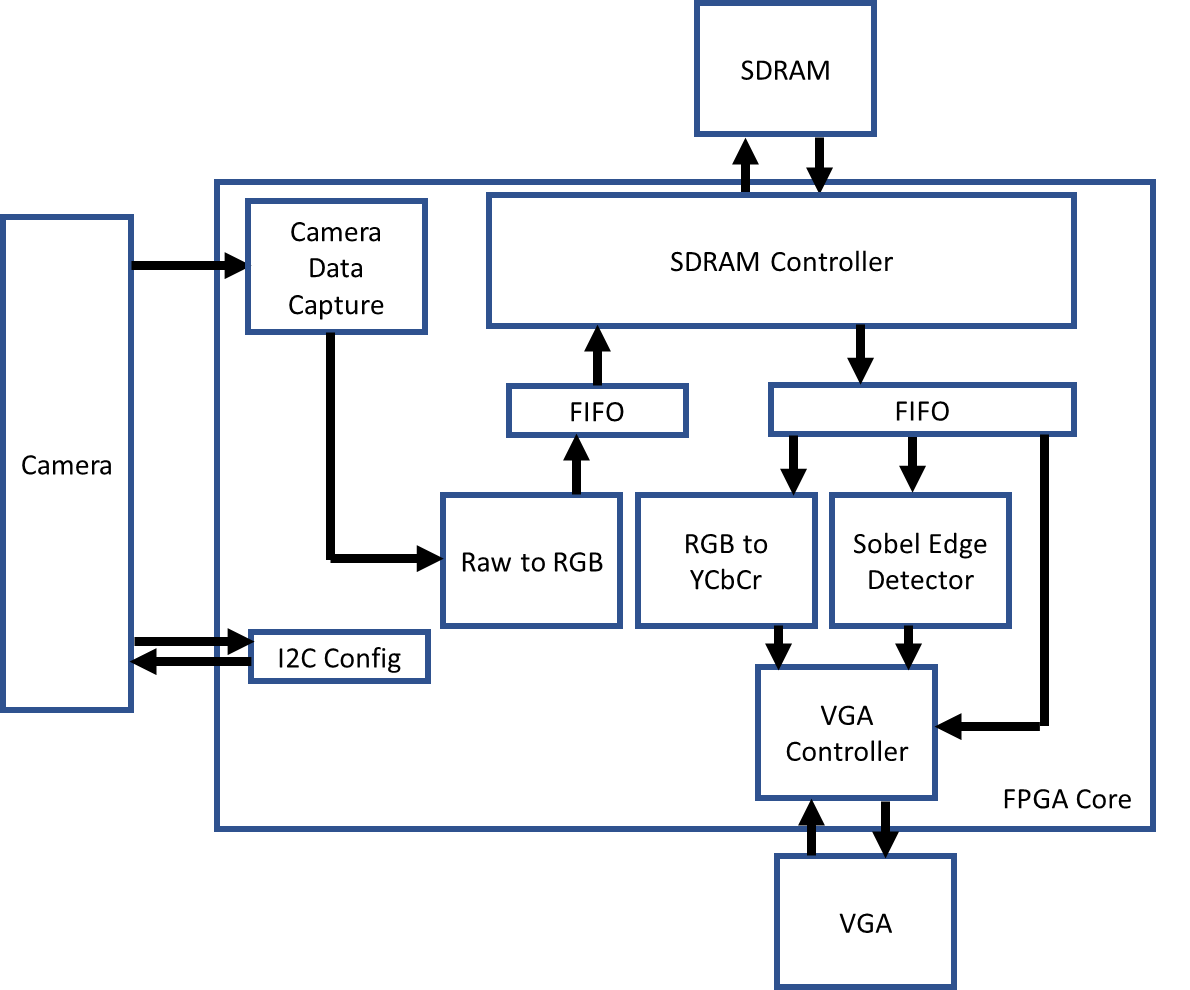
* DE2\_115\_CAMERA
  + Reset\_Delay(輸出reset信號)
  + CCD\_Capture(鏡頭畫面擷取)
  + RAW2RGB(把D5M的資料轉成RGB格式)
  + SEG7\_LUT\_8(七段顯示器)
    - SEG7\_LUT
  + Sdram\_Control(控制SDRAM資料讀寫)
    - control\_interface
    - command
    - sdr\_data\_path
    - Sdram\_WR\_FIFO
    - Sdram\_RD\_FIFO
  + I2C\_CCD\_Config(D5M模組的I2C)
    - I2C\_Controller
  + RGB2YCbCr(將RGB格式轉成YCbCr格式)
    - MAC\_3
  + VGA\_Controller(處理VGA顯示)
    - CalcHand(計算手部位置與手勢)
    - Sobel(邊緣偵測)
      * Sobel\_buffer\_3(儲存先前row資料的Line Buffer)
        + Sobel\_Line\_Buffer
      * Sobel\_MAC\_3
      * PA\_3
    - Purify(邊緣偵測純化降噪)
      * Sobel\_buffer\_3(儲存先前row資料的Line Buffer)
  + Voice\_Detector
    - I2CInitializer(WM8731模組的I2C)
    - AudRecorder(音訊輸入)
    - Volume(音量判斷)

1. Doodle Jump Game

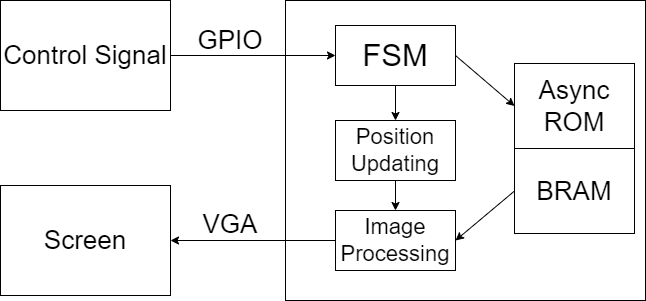
* DE2\_115
  + src
    - Top (統整所有子模組，寫FSM)
    - Display (處理VGA輸出訊號)
    - sprite (處理遊戲主角顯示)
    - plate (處理遊戲中板子的顯示)
    - monster (處理遊戲中怪獸的顯示)
    - item (處理標題、文字等顯示)
  + memory\_control
    - rom\_async (控制FPGA LUT-ROM)
    - rom\_sync (控制BRAM)
  + pictures (儲存遊戲所需圖片資料)
    - doodle, plate, monster, title, etc.

**Block Diagram**

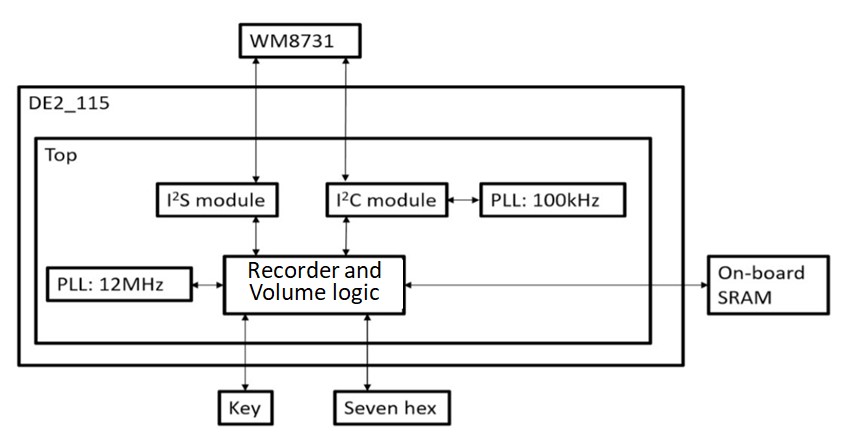
1. DE2\_115\_Camera



1. Game

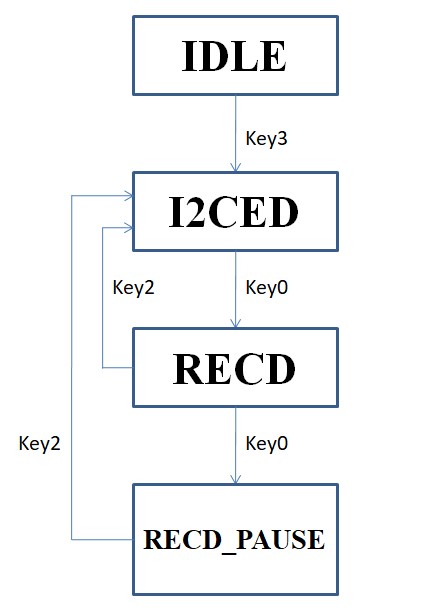


1. Volume

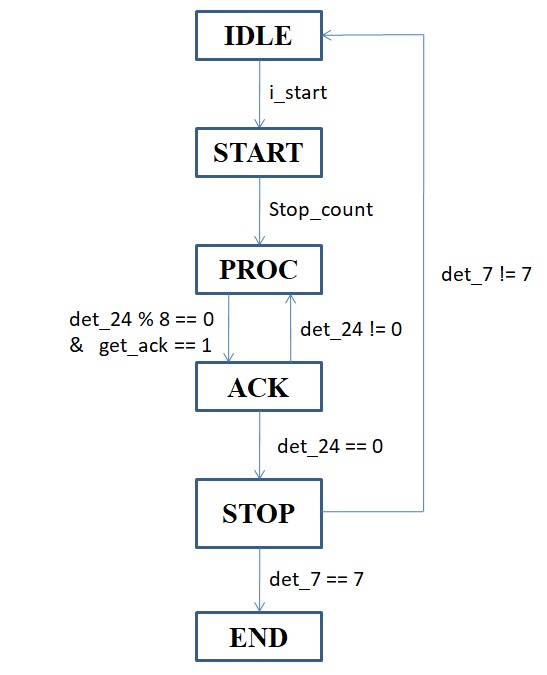


**FSM**

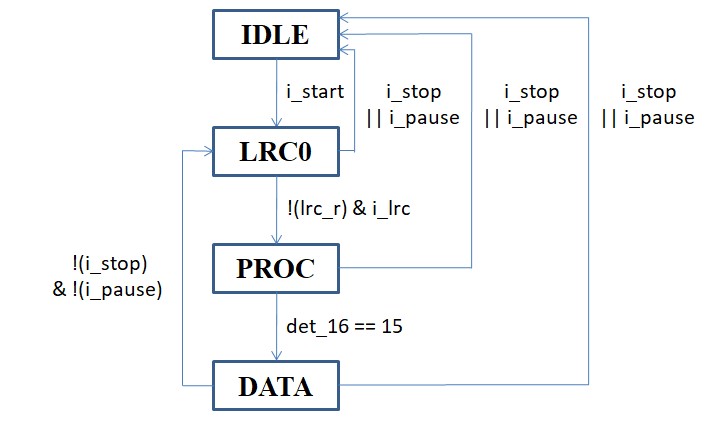
1. Voice\_Detector



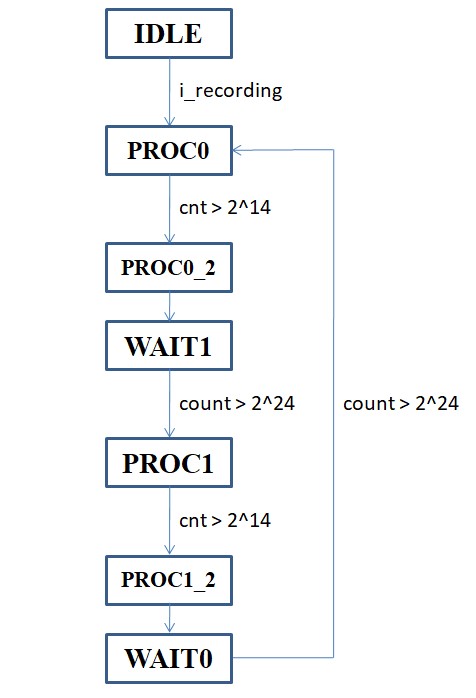
2. I2CInitializer



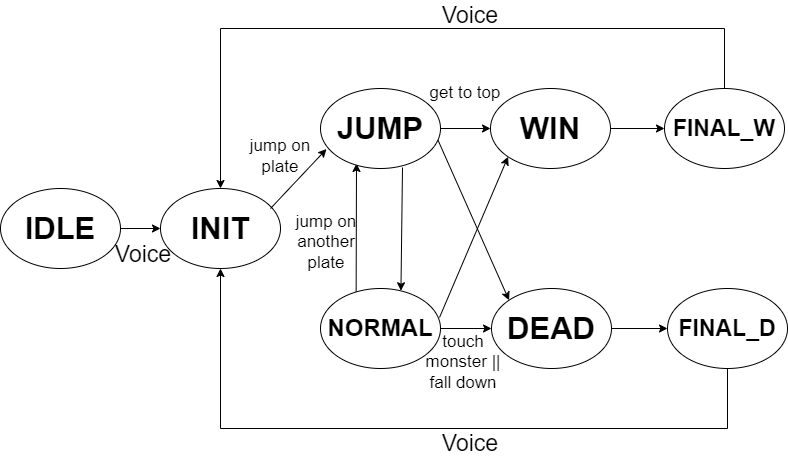
3. AudRecorder



4. Volume

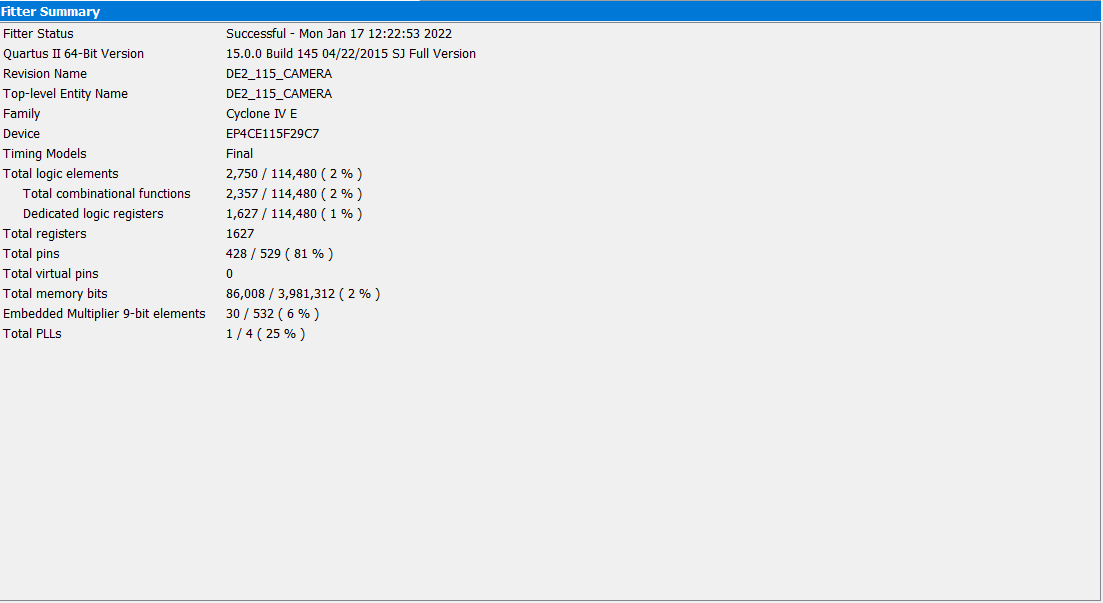


5. Game

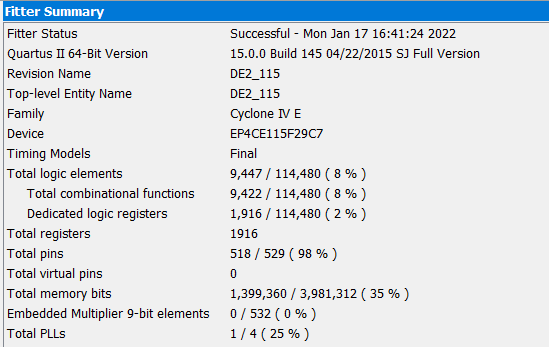


**Fitter Summary**

1. Motion Detection

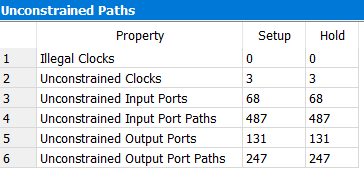


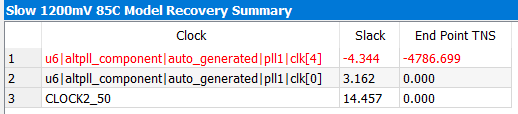
1. Game



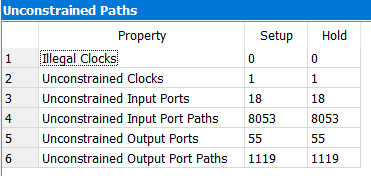
Timing Analyzer

1. Motion Detection





1. Game



**使用所需器材**

DE2-115 FPGA 板\*2、TRDB-D5M鏡頭、音源線、麥克風、螢幕\*2、VGA線\*2、電源線。

**使用方式與詳細步驟**

Motion Detection FPGA可以藉由調整SW[15:12]顯示不同的實時影像供玩家觀看

| SW[15] | SW[14] | SW[13] | SW[12] | else |
| --- | --- | --- | --- | --- |
| 實時鏡頭影像 | 皮膚色偵測影像 | 邊緣偵測影像 | 皮膚色邊緣偵測影像 | 皮膚色邊緣偵測純化影像 |

玩家可以藉由調整SW[8:1]來設定邊緣偵測的閾值，SW[8:1]即代表閾值的二進位表示。

**Game - Doodle Jump**

遊戲控制：開機後，按下key1進行重置。當玩家透過麥克風輸入足夠大的聲音時，遊戲開始，玩家可以在相機前左右移動手掌，控制遊戲中的人物左右移動，也可以利用開掌和握拳來發射子彈；而以上的這些控制也可以使用DE2-115上的按鈕和開關來達成。詳細的控制選項如下表：

| 玩家操控方法 | 對應的DE2-115板上操控 | 遊戲狀態 | 附註 |
| --- | --- | --- | --- |
| 手掌向左移動 | 按下key3 | 遊戲主角向左移動 | 手部操控時，向左、向右依照移動幅度各有三種速度；以DE2-115操控時則是固定速度 |
| 手掌向右移動 | 按下key2 | 遊戲主角向右移動 |
| 握拳 | 按下key0 | 遊戲主角發射子彈 |  |
| 開掌 | 無 | 遊戲主角不發射子彈 |  |
| 對麥克風說話 | 打開SW[0] | 遊戲開始/重新開始 | 只在遊戲尚未開始時有效 |

遊戲方法：此遊戲改編自手機遊戲doodle jump。遊戲開始後，遊戲中的主角會不斷往上跳(不需操控就會一直跳)，玩家透過向左或向右的手勢操控遊戲主角跳上飄在空中的板子，若成功跳上最上面的板子，則遊戲成功；若在過程中沒有著地於板子或碰到飄在空中的怪獸，則遊戲結束。可以透過握拳的方式讓遊戲中的主角向上發射子彈，若怪獸碰到子彈就會消失。

關卡選擇：每一次進入遊戲，關卡都是隨機的，也就是板子的位置和怪獸的位置都是隨機的。若想要固定關卡重複挑戰，可以調整DE2-115上SW[8:1]的開關，只要固定開關，例如打開1、4、5的開關，每次重新開始時關卡會固定(除了所有開關都關閉時，關卡會隨機)。

**實作設計技術細節與巧思**

1. 手部位置辨識

我們利用顏色辨識偵測手部在畫面中的位置。為了更好的從畫面中篩選出皮膚色，我們將RGB的影像資料形式轉換成YCbCr的色彩空間，以此免除亮度所造成的影響。根據測試結果，我們將91<=Cb<112, 133<=Cr<180的pixel定義為皮膚色。

YCbCr與RGB的轉換公式如下:

Y = (0.257\*R) + (0.504\*G) + (0.098\*B) + 16

Cr = (0.439 \* R) - (0.368 \* G) - (0.071 \* B) + 128

Cb = - (0.148 \* R) - (0.291 \* G) + (0.439 \* B) + 128

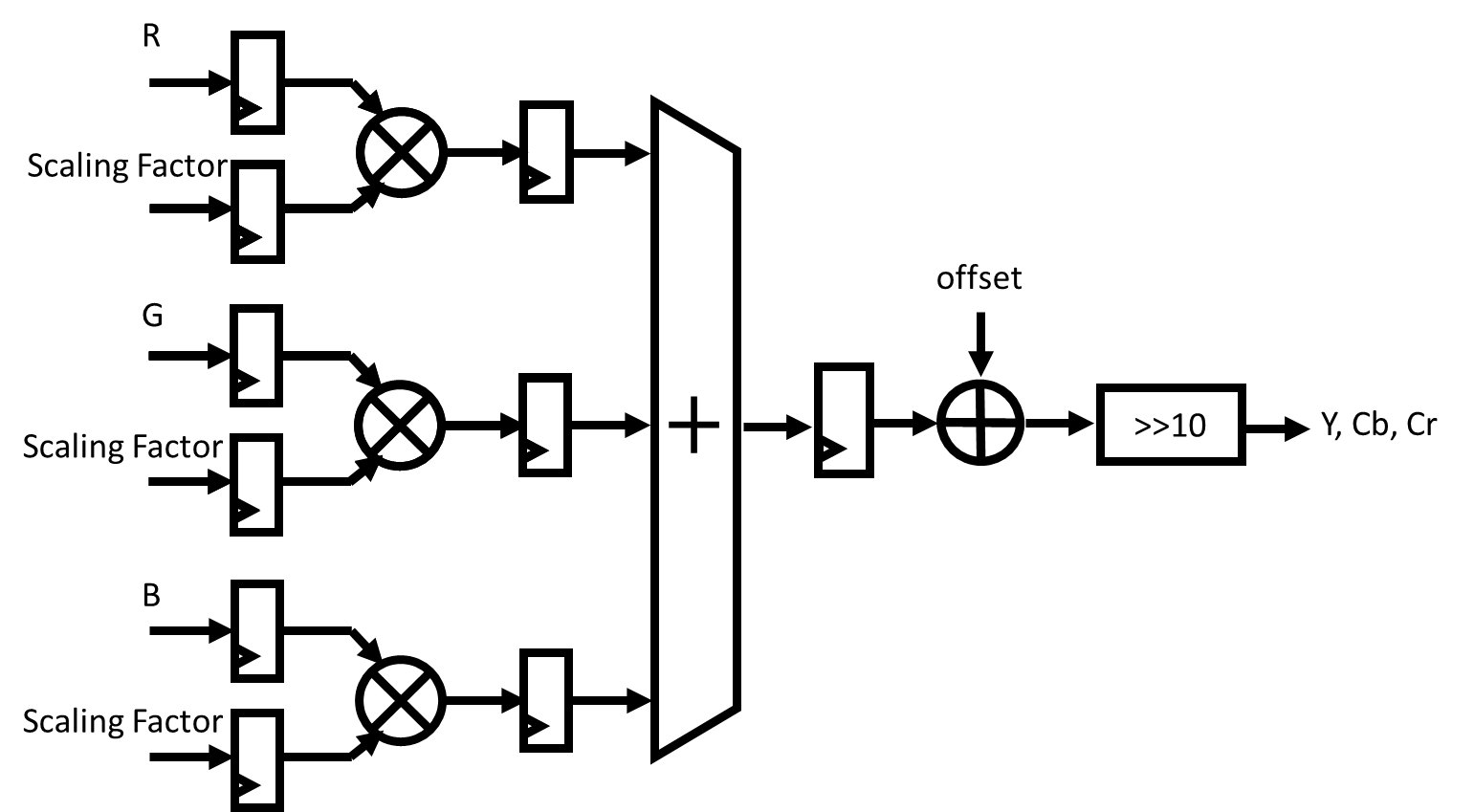
為了方便於硬體計算，我們將運算式放大1024倍，得到

Y = [(263 \* R) + (516 \* G) + (100 \* B) + 16384] >> 10

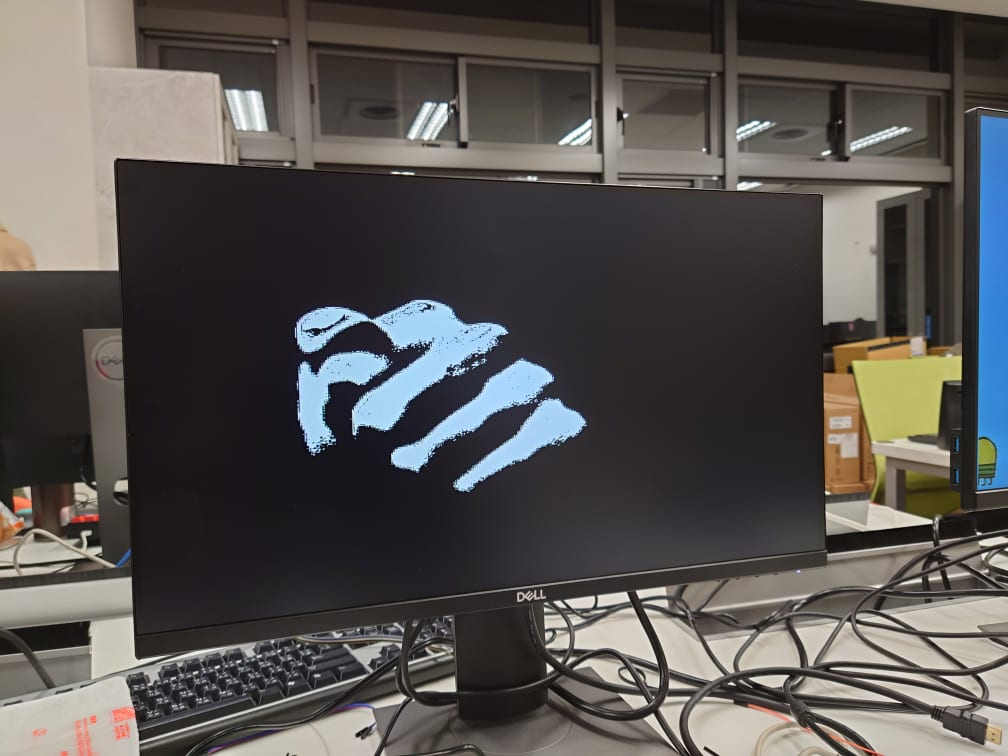
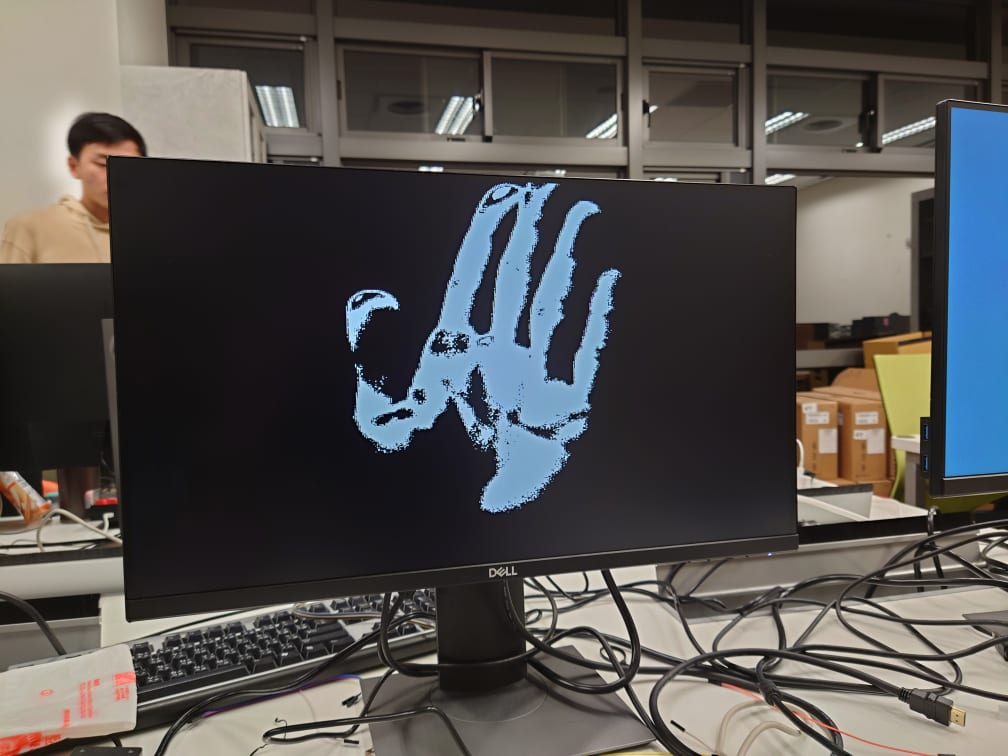
Cr = [(450 \* R) - (377 \* G) - (731 \* B) + 131072] >> 10

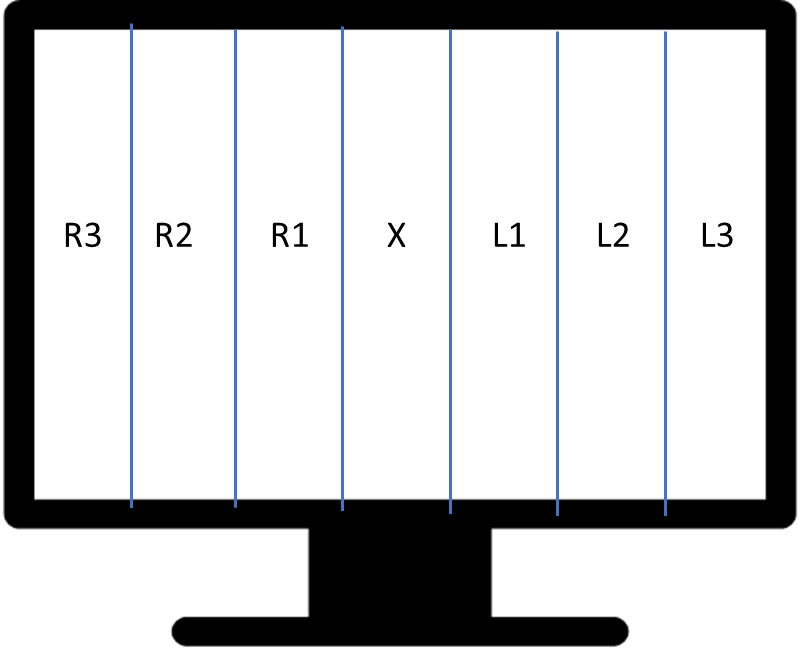
Cb = [- (152 \* R) - (299 \* G) + (450 \* B) + 131072] >> 10

並以此公式設計硬體電路如下:



以下為經過皮膚色顏色辨識後的圖像



為了辨識手部的左右位置，我們將畫面等比例切成七等分，並分別計算每個區段的皮膚色pixel數量，選出擁有最多pixel的區段，如果該區段的pixel數量大於設定的閾值，就將對應的移動訊號傳送給遊戲控制的FPGA板。控制信號的對應如下:

X:維持原地

R1:向右(慢)

R2:向右(中)

R3:向右(快)

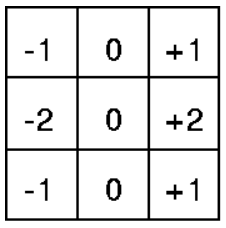
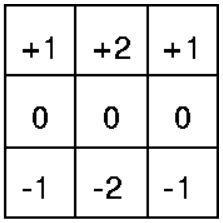
L1:向左(慢)

L2:向左(中)

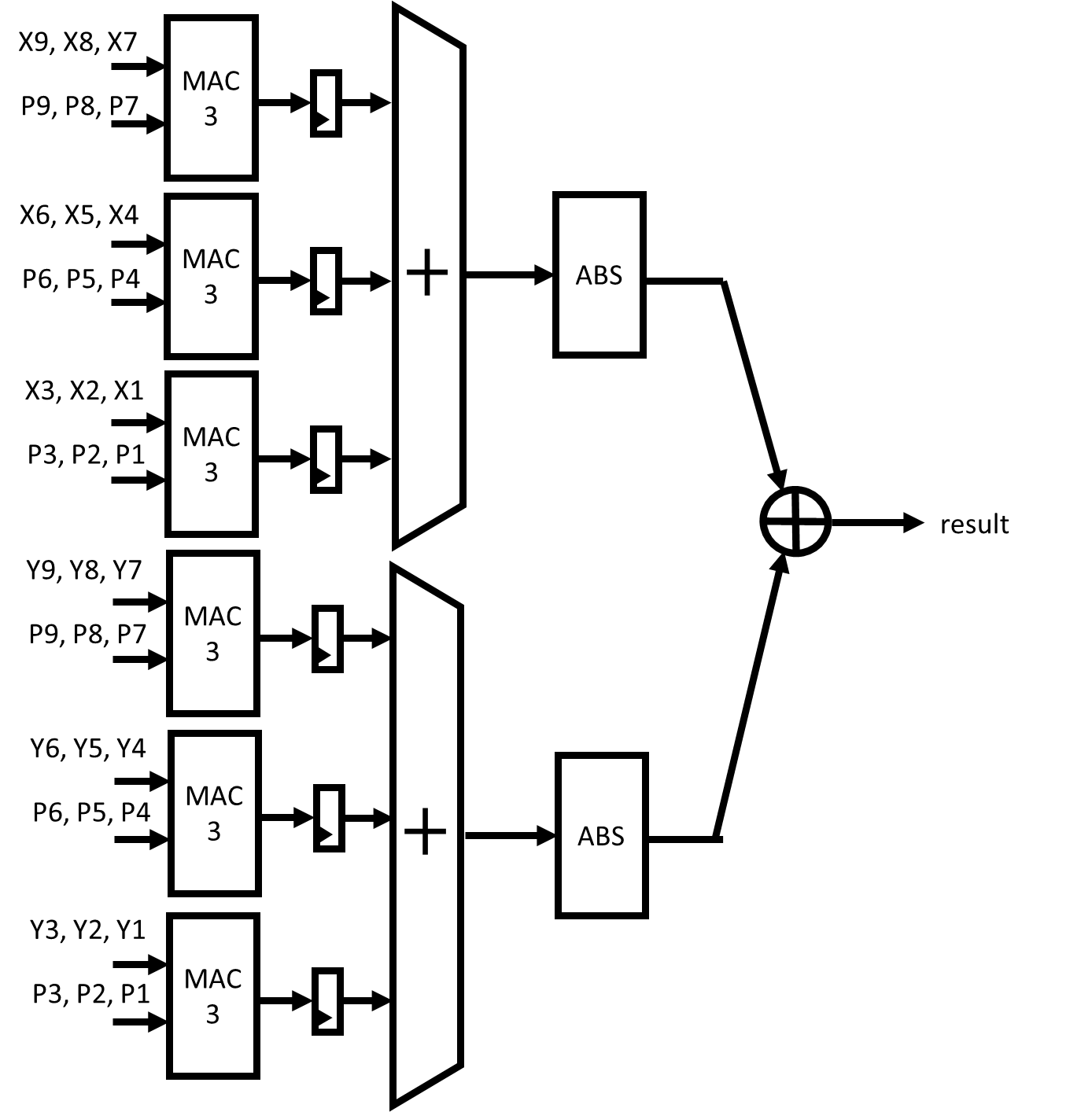
L3:向左(快)



1. 邊緣偵測

邊緣偵測能夠偵測高梯度的pixel，以此來近似玩家手部的周長，因此我們利用此方法以有效的識別兩種不同的手勢。我們先將圖像和x方向與y方向的Sobel filter做卷積，並將兩者計算後的數值做絕對值相加。

為了在目前時刻擁有前兩個row的資料，我們利用了長度為800的line buffer，並利用VGA的讀取信號作為Clock Enable信號來記錄資料。硬體電路的設計方法如下:



在做完Sobel Edge的運算後，我們還有做三個後處理的步驟

* 二值化

將pixel根據與閾值的大小關係設為黑或白兩色，玩家可以利用SW來自行設定閾值大小

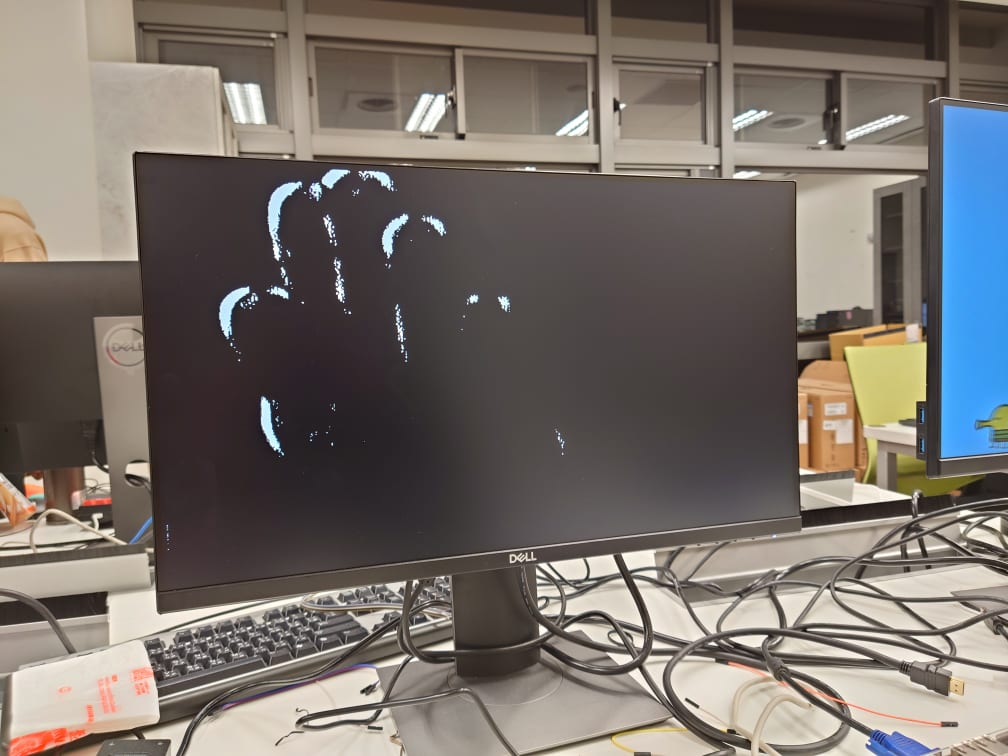
* 皮膚色篩選

為了使背景的邊緣不被偵測出來，將非皮膚色的邊緣全部去除，此處理能大幅提升背景的容忍度。

* 純化降噪

對於每一個pixel，如果周遭的pixels都非邊緣的話，則將此pixel亦設為非邊緣點，此處理降低了圖片上的噪點，使手部邊緣能倍計算得更精確。

以下為邊緣偵測的結果(邊緣偵測閾值:80)



我們計算整個畫面中邊緣點的pixel數量，若數量大於子彈發射閾值，便發送子彈發射的信號給遊戲控制的FPGA板

1. 音量偵測

在遊戲開始前，我們利用麥克風進行收音，若接受到的聲音分貝大於一定的值(設定為正常說話的音量)，則將遊戲開始訊號傳遞至主遊戲程式。詳細原理如下：在I2C初始化成功後，就開始透過I2S接受音訊資料，在WM8731的16-bit Audio Input中，[14:10]這五位數是紀錄音量大小的資訊，從11111代表的+12dB遞減至00000代表的-34.5dB，依此我們可以藉由理論和嘗試找到適當的閾值，使得對準麥克風說話時能順利觸發控制訊號。

此外，由於難以控制人聲會持續的cycle數，在每一次接受到「大分貝聲音」時，會將遊戲開始訊號pull high兩個cycle(I2S和主遊戲clk差異)，然後pull down至0並持續約莫1秒的時間，然後才能繼續針對外界聲音分貝做出反應，此設計即是為了避免一次聲音觸發多次開始訊號的問題。

1. 遊戲圖像存取

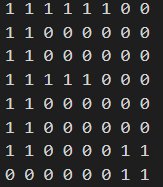
遊戲的圖像處理部分，因為此遊戲中的圖案重複率高、畫質要求不大，因此我們參考網路上相關專案的方式進行設計(<https://projectf.io/posts/hardware-sprites/>)。

* 圖片儲存

每張圖被分為兩個檔案儲存，第一個檔案為顏色和編號的對應，如下圖：

此檔案代表編號0的顏色為CCC、編號1的顏色為AAA，以此類推，而顏色的表示法為12 bits RGB，顏色順序為R、G、B，因此874的意思為紅色8、綠色7、藍色4。

第二個檔案為圖像中的pixel與顏色編號的對應，如下圖：



此檔案代表第1~6個pixel的顏色是編號1、第7~8個pixel的顏色是編號0，以此類推。對應第一個檔案的顏色資料即可知道每個pixel的顏色為何

* 圖片讀取

程式中我們使用兩種記憶體，分別是ROM和BRAM。第一個檔案的顏色對應中，因為顏色種類少(16個以內)，因此使用與clock非同步的ROM (LUT)存取；第二個檔案中的每個數值代表一個pixel的資料，這些資料是有順序性的，在VGA顯示中，每一個cycle都會處理一個pixel的資料並輸出，也就是每個cycle會往後拿一個資料。因此我們使用FPGA中的block ram (BRAM)來存取資料，藉此減少resource的消耗和latency的增加。

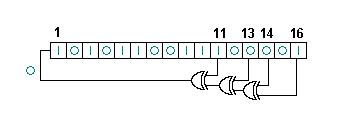
* 圖片顯示

在每個cycle中，程式會檢查這個cycle要顯示的pixel的位置與各圖片的位置是否重疊，並選擇並依照圖片顯示的先後次序，決定這個pixel要顯示哪一個圖片(若每張圖片的位置與這個pixel都沒有重疊，則顯示背景)。決定好要顯示的圖片之後，利用ROM及BRAM讀取這個pixel的顏色編號與對應的顏色，並轉換成VGA可以顯示的24 bits RGB格式，進行VGA輸出。

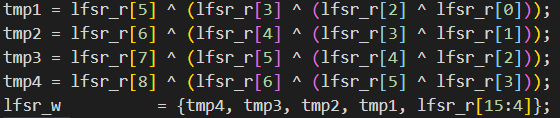
本遊戲使用的VGA輸出格式為640\*480，加上預設的synchronization signal，水平共有800個cycle，垂直共有525個cycle。因此稱800\*525個cycle為一個frame，每一個frame開始時都要依照目前遊戲的狀態，計算每張圖片的位置以及新的遊戲狀態，得到這一個frame的正確圖片座標。

1. 隨機關卡設計

我們希望玩家每次進入遊戲時，遊戲內的板子、怪物的位置都可以有所變化，以增加遊戲的多元性和好玩程度。為了產生隨機的板子及怪物位置，我們先預先設定好1000個板子的座標及20個怪物的座標，再以LFSR(Linear-Feedback Shift Register)產生亂數數列，螢幕上的板子及怪獸的出現位置就會依照此序列來產生。



LFSR的實作方法為給定一個16’b的數值作為LFSR的初始值，每次亂數數值需要更新時，根據上圖表示的運算方法算出新的數值加到數列最左邊，再將最右邊的數取出，連續產生四次即得到新的亂數。詳細的運算方法如以下的運算式。



這個LFSR亂數產生器可以產生2^16-1個循環元，且因為我們一次取出4個值，4與2^16-1互質，所以這個亂數產生器要產生2^16-1個數值之後才會開始出現重複的pattern，在平常的遊戲體驗下應該算是夠用的亂數產生器了。

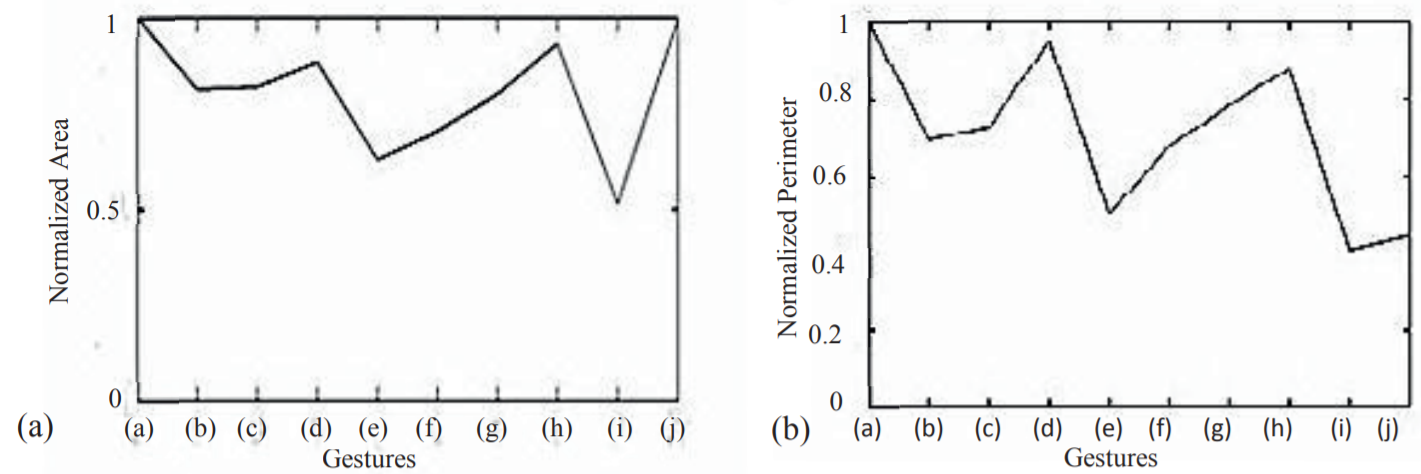
1. 遊戲可玩性調整

為了使遊戲可玩性增高，我們考慮遊戲主角移動速度、跳躍高度和水平移動速度等因素，設計每個板子之間的距離，並確保沒有「不可能破的關」，每兩個板子之間的距離皆為有辦法達到的距離。

另外，我們也盡量將遊戲動畫設計得跟doodle jump手機遊戲一樣，因此要計算每個frame中遊戲主角的移動速度與位置，以及畫面的調整(例如跳上板子後，背景畫面先往下移動直到板子碰到畫面底端，遊戲主角再向上移動)，力求最好的畫面呈現。

**遇到的問題或挑戰與解決方法**

1. 為了解決玩家距離差異對手勢偵測的影響，我們原先計畫在以邊緣偵測與皮膚色辨識得到手部的周長與面積後，以周長^2/面積做為參考數值區分手勢。此計算方式理論上能在不同距離下辨識出玩家的手勢。但手部的面積與周長在多數的手勢下都呈正相關(如下圖所示)，因此會使此數值的差異不明顯，經過測試後，此數值在不同手勢下的差異不夠大，當玩家手部快速動態移動時，便無法正確預估出玩家的手勢。



1. 邊緣偵測不只會偵測到手部的邊緣，也會偵測到背景不均勻的部分。為了解決這個問題，我們將所有非皮膚色的pixel邊緣值都清除，並再經過純化來降噪，使得圖像中的邊緣偵測結果的噪點更少。
2. 本專題使用兩個DE2-115板，其中一個處理聲音與影像訊號，另一個處理遊戲，板子之間使用杜邦線連接GPIO pin腳。剛開始測試時，常常會有接收端無法收到輸出端訊號的情況，而且是偶發性的，接收成功的機率大概為50%。仔細研究後，我們發現兩個板子的clock訊號不同步且頻率不同(12MHz與25MHz)，因此若輸出端的輸出信號只維持1個cycle，則接收端有可能會忽略此訊息。解決方法是輸出端的輸出信號維持更久(2~3個cycle)，最後成功解決這個問題。

**本學期實驗的心得與建議**

本學期的實驗包含三次的Lab Assignment和期末的final Project，難度從淺到深，而最終的Final Project更是各組之間的軍備競賽，大家竭盡所學，勞心勞力的希望將最完美的結果呈現給教授、助教和各位同學看，在如此艱辛的過程中，相信不論是在寫Verilog和Debug方面，抑或是分析問題並尋求理想解法上都能受益良多，也相當感謝楊教授在Demo時對於各組的提點還有最後給予所有同學的修課和學習建議。

還有，在此想要特別感謝兩位助教在這堂課程中的協助，不論是在學期初的教學課程，或是在我們做實驗中遇到任何問題時都不吝及時給予協助，包含需要器材，對Lab有疑問等等。

最後是建議的部分，個人希望在Lab Assignment Demo完後，能有一堂課的時間由助教講評這次的實驗，針對同學們的整體表現給予簡單的提點，也可以將某些組別做的特別好的地方拿出來討論，抑或是請他們上台簡單介紹，我相信這對於同學們的學習效果會有莫大的助益。