軟硬Lab6 Report

張彧豪, 0416005

*Abstract*—利用軟硬協同設計來加速findface這個程式的執行速度。

Keywords—臉部辨識、軟硬協同

# Introduction

Find face是一個利用”template matching”來做臉部辨識的C程式，辨識的方法是找出小圖face與大圖group中哪個區域的sum of absolute difference最小。在lab1已經知道這個程式要花最久時間的地方是比對部分，這次作業目的就是要利用硬體加上軟體的共同合作使這個find face程式的執行速度加快。

# Related Search

這次作業主要做的是臉部辨識，準確的來說為”template matching“，希望可以利用硬體來加速”template matching”的速度。

# Developing the Program

## 設計理念

這次作業的架構設計我是採用全硬體的架構，這個架構是我跟王柏堯(0416305)同學討論出來的，會這樣設計的原因在於在前五次的lab中，如果要讓find face程式有最大程度的加速，使用lab3加上lab4所學到的東西應該可以讓find face有最大程度的加速，lab3 先計算一個column後再右移可以減少資料傳輸的次數，修正lab2的缺點，但使用軟體進行資料傳輸還是較硬體慢，所以我們又將lab4所學到的DMA技術加進我們的設計中，期望這樣可以有最大程度的加速。

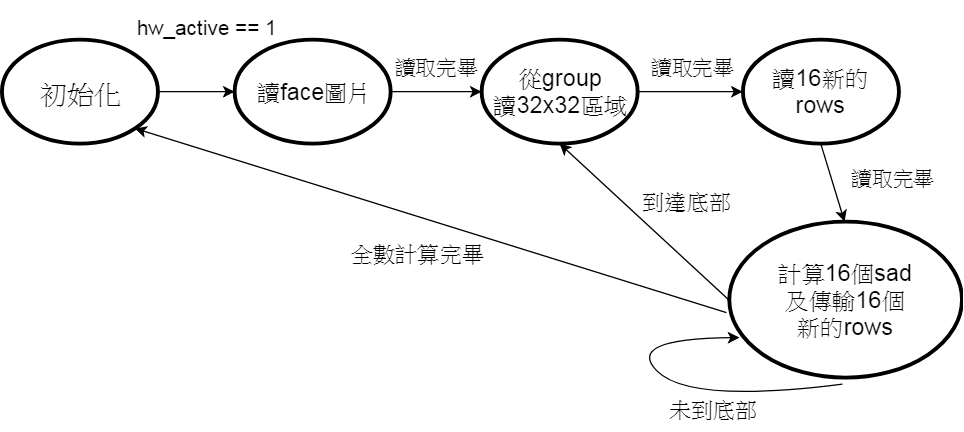


圖1. 此次lab狀態有限機

## 程式架構－硬體部分

在我們的設計中，硬體必須實作出像lab3那樣的流程控制，由於資源消耗的問題，我將Lab3中的reg\_bank換成shift registers設計，這樣可以讓硬體用shift的方式來準備好下一次要計算的group區域，這樣計算絕對值也不用還要看reg\_bank的值是多少後再進行計算，shift register會讓每次要進行計算的區域都在最上面。我們也有將lab3的加法樹改進，在lab3中我採用的是一次處理256個元素，每個加法元件相加兩個元素，需要八個cycles，搭配上pipeline，加完1024個元素需要11+1 = 12個cycles(最後一個cycles是將四個256個元素輸入的加法樹結果相加)；在這次的lab中，我採用的加法樹模型如圖2所示，採用的是更貪心的做法，每個加法元件相加4個元素，所以只要5個cycles就可以算出該區域的sad值，之後再用一個cycles比較，在計算單一個32x32區域的行為上比lab3少了將近一半的cycles數。

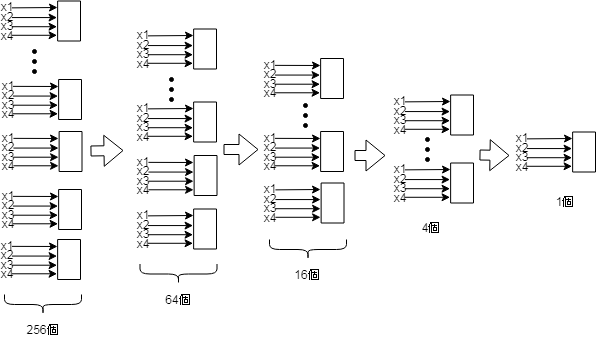


圖2. Lab6使用的加法樹圖示

由於還要將軟體寫值這個行為轉為硬體實作，所以要將lab4的設計也加進來，基本上使用硬體直接跟記憶體拿值的行為跟lab4完全一樣，我們採用固定的burst length，方便除錯跟撰寫程式碼。

由於lab3的計算模式為硬體算完一32x32區域後，等軟體將一新的row傳進才繼續開始算，我們認為這樣的方式可以與lab4的傳輸資料平行處理，因為lab3與lab4是獨立的計算行為，所以可以平行化處理。這邊的平行化並不是算一行的同時去抓取另外一行，而是先預備好多的16個rows，接著開始pipeline計算16個32x32的區域sad以及傳輸新的16個rows，等到傳輸及計算皆完成後就可以將再去將新的16行搬進來後就可以再開始計算sad值，可參考圖3。雖然說這樣蠻方便的，可以一次算出16個sad值，但這種平行化方法效率可能沒有算一行同時讀一行這種效率來的好，原因在於傳輸時間可能會比計算時間還長，那我們會採用的原因是這樣寫較好除錯且時序問題就不會是另一個很惱人的問題。

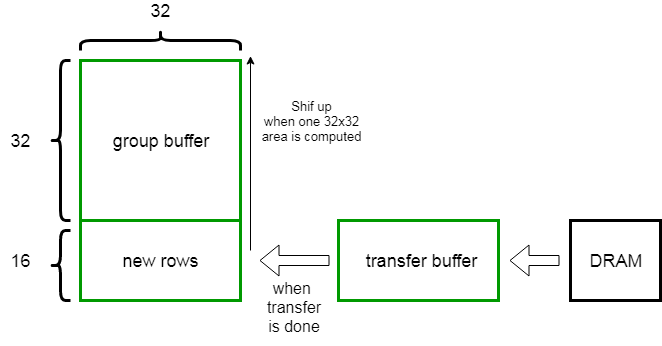


圖3.硬體平行架構

## 程式架構－軟體部分

因為我採用全硬體的方式來實作取值、計算並比較找出最小的sad值，軟體的事情幾乎都是硬體在做，軟體只要先把face跟group的起始位址藉由slave registers傳入硬體並傳入hardware active的訊號給硬體請它開始計算，最後軟體利用busy waiting的方式等硬體將結果經由slave registers送回軟體即可。我使用的是一次只計算一張face，因為我是採用register array實作，若同時計算兩張face會導致電路面積過大無法合成出來。

## 記憶體位址問題

在這次作業中，我遇到最麻煩的問題就是記憶體位址的問題，一開始我的burst length設為8，這樣可以一次把32-bytes長的row直接讀進來，但由於圖片在記憶體中是一byte一byte存的且master IP發出的位址卻必須間隔4 bytes，這樣導致使用硬體實作出lab3中往右移動一個byte的行為變得困難。

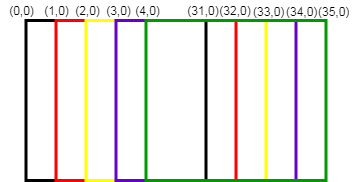
我的第一個解決方案是參考黃右萱同學提出的一次讀取36個bytes。

圖3. 36-bytes示意圖

如圖3所示，當一次讀進一個row且是36bytes時，下一次要再讀的起始位址就是原本的加4，且lab3中的往右移動一個byte後再開始計算的行為也可以被實作出來，且還是可以使用我前面提到的先預先傳16個新的rows後再開始計算並讀取下一輪的16個新的rows。如果這樣設計，運算的流程會變成算完一個32x32區域後，先往右再算，直到算到底後再移回原本的開頭往下再開始新的一輪，這樣的設計看似很好寫，但我在實作的過程中出了一點問題，我遇到的第一個問題是控制訊號不好寫，一直配不出適合的控制訊號，加上我使用register array實作這次lab，導致電路面積偏大，不好使用ILA debug，經過好幾天掙扎決定放棄這種方法。

我的第二種解決辦法是在軟體讀取照片後，將照片的像素值存進以unsigned int為型態的陣列中，這樣每個pixel就是存在一個4bytes的空間內，lab3的往右移動一個byte的行為就可以輕易的被實作出來，只是在lab6中位址是要移動4bytes，所以我將我的burst length設為32，這樣才能一次傳完一個原本32-bytes的row，當然這樣的實作方法讓導致傳輸時間是原本burst length設為8時候2~3倍時間，算是一種犧牲吧，拿時間來換取降低實作的複雜度。

## 流程

1. 首先先由軟體將第一張face、group的起始位址藉由slave registers傳給master IP。
2. 軟體使用slave register驅動master IP請其開始計算，軟體同時進入busy waiting等待最終結果由master IP經由slave registers傳回。
3. 硬體收到啟動訊號會先讀取face照片。
4. 接著讀取group中指定的32x32區域。
5. 再來讀取16個新的rows進來master IP。
6. 開始計算及比較16個sad及讀取新的16個rows。
7. 當讀取完畢後，重複第6步直到face的位置跑到group圖片的最底部，這時將x座標右移1後回到第三步開始。
8. 當已經算到最後一個32x32區域後，將hw\_active訊號拉下，並將算出的結果藉由slave registers傳回軟體。
9. 軟體將結果印出來，若還有face圖片需要比對，則回到第一步重新開始動作。

# Resule and Discussion

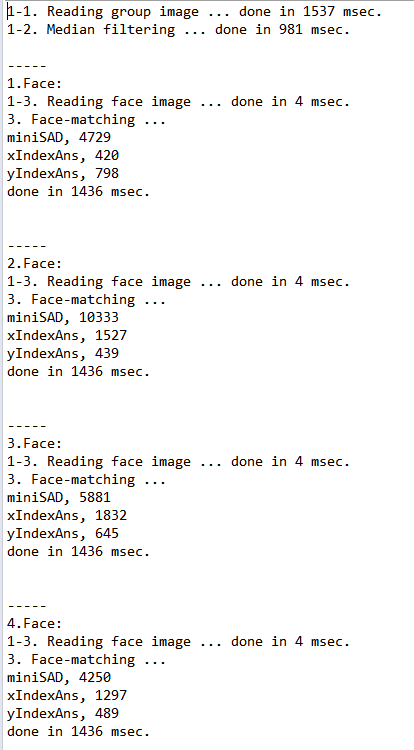


圖4.lab6執行結果

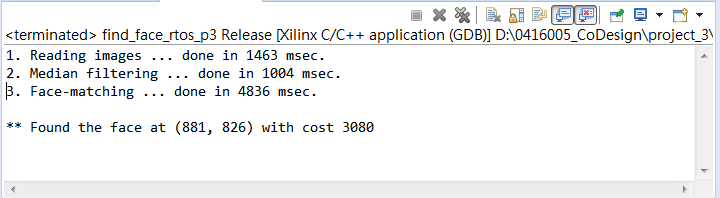


圖.5 lab3執行結果

由圖上可以看到，算出一張圖片的時間大約是1436msec，這個時間大約是lab3的三分之一左右(可由圖4及圖5看出)

# Conclusion

在這次的lab中，可以看到若是改成全硬體，速度真的可以加快不少，但同時也有硬體實作上的限制，例如：記憶體位址問題，但若克服天生的限制後其實硬體實作上是比較有效率的。那從這次lab的debug過程我也發現到，其實計算部分並不會佔到多少時間，反正時間大部分都花在資料傳輸上，記憶體的傳輸才是整個運算過程中的瓶頸。那這個lab若有block ram來實作的話還可以更快，因為電路面積所以可以同時實作好幾個運算單元，但因為時間及能力不足，沒有辦法實作出來。