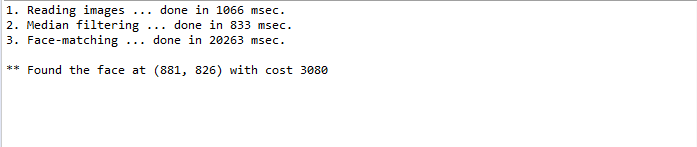
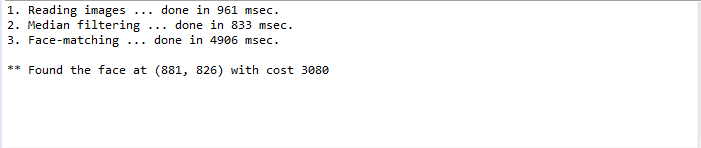
Shlab lab6 report

何元斌, 0416313

# Introduction

在lab3中，我們已經把計算sad的細節放入硬體中實做，在純軟體實做時 中，得出的速度和時間如下圖: 

而lab3把大部分的計算交給硬體後，得出的時間如下:

可以看到，所花的時間大幅縮減，代表硬體平行運算的特性著實在處理大量但簡單的計算時，有著顯著的加速效果，

本次lab6，討論除了藉由平行運算來加速外，還能如何利用硬體來加速。

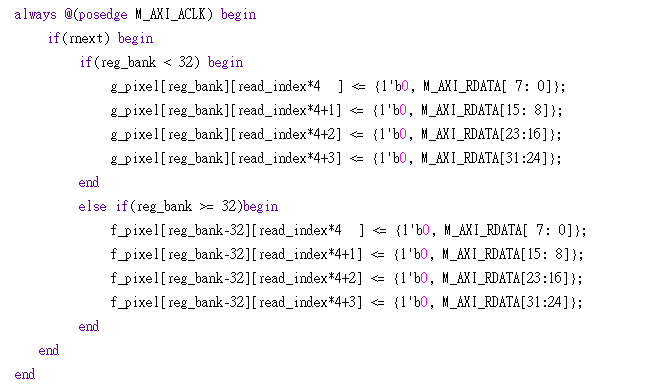
# Hypothesis

A : 假設HOT SPOT主要來自哪裡 :

一開始，在猜想是哪一部分占據最大的時間時，我認為是由於需藉由軟體讀取大量的資料，導致時間拉長，所以利用lab4所學，我把讀取資料的部分交給硬體處理，並把實作的細節，也就是原本在lab3中的adder tree等等，放入master來時做，而slave只負責當master和軟體溝通的橋梁。

B:implement:

把lab3的內容放入lab4後，遇到的第一個麻煩是，該如何拿到資料，在lab3中，我們直接藉由軟體讀取pixel後，傳輸給slave進行運算，但，由於我們想直接藉由硬體來讀取資料，因此，我讓軟體把每一行的初始地址傳給硬體，而硬體在接收到這些地址後，直接burst 8個word，也就是32個pixel，如此一來，硬體可直接從burst後的資料拿值並進行運算(如右上圖，在讀完一次burst後，直接從M\_AXI\_RDATA拿取資料，而非像LAB3一樣從軟體透過SLAVE REG把資料傳給硬體)，pagination anywhere in the paper. Do not number text heads-the template will do that for you.

而無須透過軟體讀取資料。

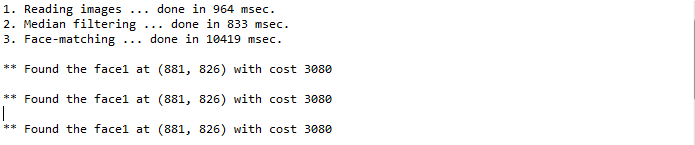
C:結果:

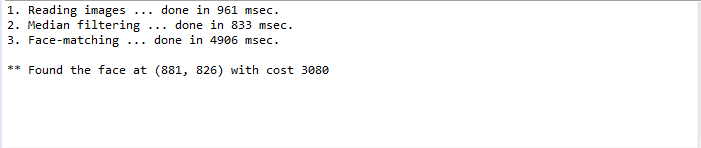
是實證明，我的猜想是錯誤的，因為實做完後，速度反而降低(如下圖):D:\0416313_lab6\1.png

因此，並非式讀取資料太慢，而是由於軟體本身速度就非常慢，而硬體必須等待軟體結束一個階段後，才能進行下一階段，因此，正確的加速法應是儘量把細節交給硬體做，減少硬體等待軟體的時間，但，由於時間不夠讓我研究如何繼續把讀取資料的功能放入硬體(也就是把大部分功能放入硬體，軟體只負責傳入每張圖片的開頭地址)，因此，我換了一種較簡單的加速方法。

# 最終採用的加速方法

放棄了把單一的FACE MATCH加速的方法後，我著手於同時MATCH多張FACE，首先，把4張FACE都存入硬體中(由於REG不夠用，所以使用了BRAM)，然後，每讀取GROUP的一塊區域(32x32)後，便和4張face比較，並分別記錄各自的min\_sad，然後計算下一區域，如此一來，便無需重複讀取整張group四次(如果只是單純的把1張face match做4次)，而只需讀取group一次，減少了軟硬體之間的溝通，讓硬體不用一直等待軟體，其結果如下:





第一章圖是4個face match，第2張圖是lab3的單一face match，可以看到，時間比原本單純的把4個lab3串起來(4\*5=20)快了1倍，但計算量並無減少，可見軟體的速度著實比硬體慢了許多。

# 困難:

實作細節上，有些特別的困難需要克服，例如:

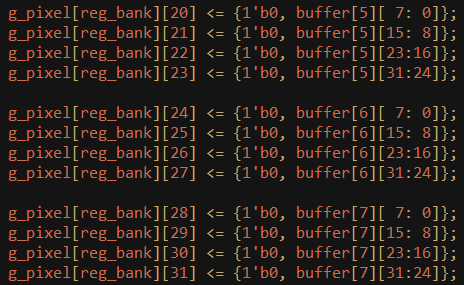
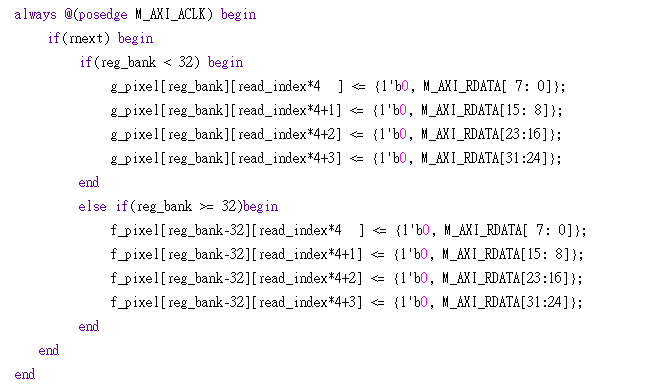
1.多個FSM:

由於控制burst需要一個FSM，而控制計算ADDER TREE也需要一個FSM，因此會碰到一個檔案內有多個FSM的情形，這種情況下，兩者需互相配合，例如，由於控制BURST的FSM也控制了跳出的時機(HW\_DONE)，因此，控制BURST須等待控制計算的FSM完成後，才能讓HW\_DONE升起，由於資料的傳輸和計算是由不同的FSM控制，所以須格外小心。

2.DEBUG:

由於直接從硬體讀取資料，拿取資料的過程也可能會有錯誤，再者，控制訊號也變得非常繁多，因此，偵錯也會變得困難。

3.面積控制:

有時候，一些改動也會造成面積大幅增加，進而導致generate bitstream的時間達福上升，增加debug難度，例如下圖兩種讀取資料的對比: 

上圖使用的面積為2萬多，而下圖使用的面積為5萬多，導致generate bitstream的時間從20分鐘上升到1小時28分鐘，而差別只有間接透過buffer拿值，和直接從M\_AXI\_RDATA拿值，這些不可預期的面積增加，往往導致debug時間大量上升。

# 儲存資料

由於需執行多個face match，為了避免重複讀取同一圖片，最好的辦法是把圖片存起來，但由於硬體資源不夠，須藉由block ram存圖片，但由於bram的地址控制過於複雜(只能有一個addr)，所以我並未實做。

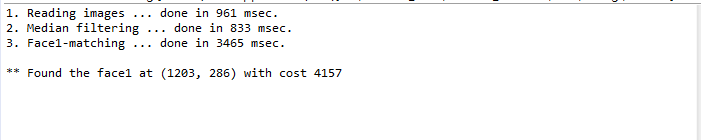
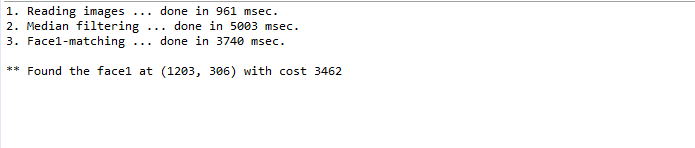
# 軟體加速:

由於把演算法之類的部分都交給硬體了，相對的，軟體的功能就剩非常少了，因此，軟體能加速的方法只剩

編譯器加速，且對於那些已經把功能都搬到硬體的步驟來

講，效果可說是微乎其微(例如對第三步驟的face match來

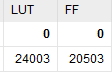
講，加速效果非常不明顯，如下兩圖的對比)(上圖是未優化，下圖是有優化)，只有對那些還由軟體處理的步驟有顯著的效果(例如第2個步驟的median filtering):



# 面積:

實作後發現面積變大很多，如下兩圖的對比(上圖是lab6，下圖是lab3):

D:\0416313_lab6\7.png



我認為面積之所以會變大如此之多，應該是因為控制訊號太多，或者是其他原因，有待繼續研究。

# 結論:

硬體程式的設計比起軟體程式的設計難上許多，不僅是硬體程式是平行化，硬體程式還要考慮資源使用量、clock的控制，再者，硬體的debug也難上許多、時間也拉長許多，因此，這份作業著實花了不少心血，而我能力不足，放棄了一開始的全硬體寫法，只能在倒數第2天改成較簡單的，使用bram的寫法，這也讓我有點氣餒，希望日後若碰到相同的情況，能有足夠的能力達成目標。

硬體由於其平行處理的關係，對於那些簡單但大量的工作非常有效率，但，其控制也相較於軟體的控制來的複雜許多，因此，才需要由軟體來控制硬體，但，軟體控制的部分越多，越是拖慢了硬體的腳步，再甚者，硬體由於需要實體資源，更考驗了程式人員對於編寫程式的能力，如何使用更少的LUT來達成更快的速度，更小的面積是個重點，在速度和程式人員的能力中間取得平衡將是個關鍵，最後，希望在這軟體漸漸成為主流的年代，也莫忘記硬體加速的潛力。