高效率資料流圖像信號處理器電路設計

Hardware Design for Image Signal Processor with Efficient Dataflow

作者：李承霖、丁昱升、陳欽安(照學號排序)

指導教授/實驗負責教授：闕志達

摘要(Abstract)：一般相機或手機內部的影像訊號處理器(ISP)可以將拜爾原始圖檔(Bayer raw image)轉換為標準的全彩圖檔(sRGB)，傳統的步驟包含去馬賽克、降躁、白平衡以及伽瑪校正。本研究在沒有 SRAM 的設計前題下，提出一個高效率的資料流，將此 4 個步驟以化約為兩個階段進行，以減少圖片的總輸入次數。另外，也透過增加設計的暫存器數量，進一步降低晶片與記憶體的溝通次數。接著我們也有針對晶片暫存器的數量以及所降低運算週期數的權衡進行討論。並在最後透過實作出繞線佈局完成的晶片進行真實圖像的處理模擬。

一、簡介

大部分數位相機、錄影器、掃描器使用的數位感光元件大多數都是用一種拜爾濾色鏡的濾色陣列來製作彩色影像，而相機內部的影像訊號處理器即可以將影像感測器的原始訊號接續轉換為標準的全彩(sRGB)，其中主要過程即包含去馬賽克、降躁、白平衡、伽瑪校正等運算。

1. 傳統影像處理流程

a 拜爾濾色鏡

拜爾濾色鏡中有 50%的綠色，25%的藍色，以及 25%的紅色濾色鏡，排列方式如下圖所示。因此，拜爾濾色相機拍攝出的原始圖檔，每個像素只會紀錄紅綠藍三色中的其中一種 (視該像素對應到那一個顏色的拜爾濾色鏡)；為了要得到該像素的紅綠藍數值，進而得到全色彩影像，需要經過去馬賽克插值演算法來將拜爾圖像轉為全彩圖像。

b 去馬賽克(Demosaic)

由於要將拜耳圖像中每個 pixel 遺失的兩個色值找出，我們需要進行插值運算。根據 [1], 我們考慮正確性以及硬體實作效率之後，決定以線性內插法來處理拜爾圖像。如下圖所示，處理拜爾圖像時會有 Fig.1 中四種狀況要考慮。

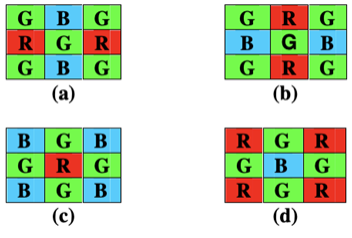

Figure 1 拜爾模式

(1) 對紅、藍像素進行內插

在 Fig.1(a), (b)案例中，我們要求出正中間像素的紅色數值及藍色數值，在線性內插演算法中，就是取相鄰同色像素的平均值。也就是說，在案例(a)中，紅色數值為左右兩個紅色像素數值相加除二，藍色數值為上下兩個藍色像素數值相加除二。

而在 Fig.1(c)案例中，由於本身就是紅色像素，因此只需要求取藍色像素數值，即為四周的藍色像素的平均值。同理，在 Fig.1 案例(d)中，本身就為藍色像素，因此只需求紅色像素數值，及為四周紅色像素的平均值。

(2) 對綠像素進行內插

在 Fig.1(a), (b)案例中，由於自身就是綠色像素，所以不需在求取綠色像素數值；在 Fig.1(c), (d)案例中，綠色像素數值就須由上下左右相鄰的綠色像素數值加總平均取得。

c 去噪(Denoise)

為了降低相鄰兩像素的差異(noise reduction)，我們採用中值濾波器，可以有效消除區域極端值對圖像的影響。如 Fig.2，中值濾波器實作方法就是將周圍（包含自身）的 9 個像素值平均得到去噪後的色值。

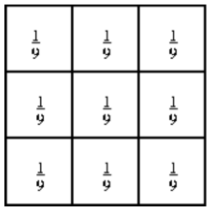

Figure 2 中值濾波器

d 白平衡(White balance)

白平衡就是要確保無論拍攝照片時的光線照明類型如何，都要能精確的

再現白光。在此硬體實作中，我們採取灰度世界法白平衡演算法，此演算法假設紅綠藍三個通道(channel)的平均值都會趨於同一灰度值 K。

e 伽瑪校正(Gamma Correction)

用於將整張圖片的色彩調亮或調暗，我們會將每一個像素的色值調整為 $255\times(\text{原色值}/255)^{1/2.2}$。

2. 設計環境與限制

由於晶片製程為 UMC 18nm，同時不具有巨集記憶體(marco memory)，因此我們無法將整張圖片存在晶片上來進行運算；因此，為了減少遍歷像素的次數，我們利用資料重複利用(data reuse)以及流水線設計(pipelining)的方式（於以下部分介紹），將整個傳統圖像處理演算法的遍歷像素次數降為 2 次。此外，通過輸入圖像大小，我們的硬體也能夠支援不同尺寸的圖像。

二、設計方法與電路架構

1. 填充預先處理(Padding pre-processing)

在去馬賽克以及降躁的兩個階段中，對於輸出的圖片中邊界一圈的像素與其他像素的演算法會稍微不同，因為 3x3 的窗口內會有沒涵蓋到圖片範圍的部分。為求硬體設計的快速及便捷，我們以填補的方式來解決此問題。在原始的拜爾圖檔中，我們在外圍填補兩像素單位的拜爾模式(bayer pattern)，與鄰近的拜爾模式相同，使得邊緣像素值的運算能夠與中間像素值運算的邏輯一模一樣。因此拜爾圖檔圖片大小將會從 n(長)×m(寬) 變為(n+2)×(m+2)

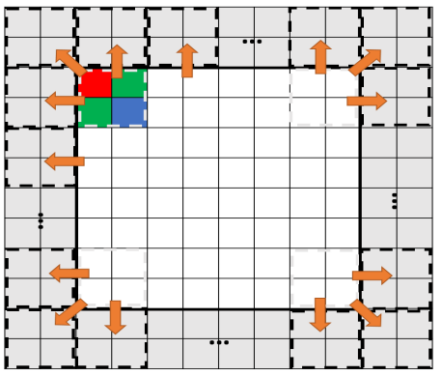

Figure 3 填充預先處理

2. 資料重複利用(Data reuse)

a. 瓶頸(Bottleneck)

我們先假定照片的拜爾圖檔的大小為 n×m ，則對於標準全彩圖片則總共有 n×m×3 (channel)的像素值。若我們將傳統圖像處理中 4 個階段所

做的輸入輸出分開考慮，理論上在去馬賽克階段輸入一次拜爾圖檔大小的像素數量，在去噪、白平衡及伽瑪校正分別需要輸入各一次 n×m×3 個像素值（對於一張大小為 1024×1024 的圖片來說就輸入一次需要約 1050 萬個輸入週期）。這樣的輸入週期量，會讓圖像處理的大部分時間都會在等輸入的像素，因此重複利用像素資料是增加處理效率的關鍵。我們先從 4 個階段中所使用的演算法進行分析。可以發現對於去馬賽克和去噪的運算來說，每一個輸出像素的都是需要相鄰像素的資訊；對於白平衡以及伽瑪校正來說，在計算完統計參數之後，即可直接透過輸入像素求出相應位置的輸出像素。而因為去馬賽克與去噪都有需要鄰近像素點資料計算的特性，因此計算出統計參數的運算可以與這兩個運算合併。計算出統計參數之後，需要再輸入一次圖片以完成白平衡與伽瑪校正的計算，因此整個圖像處理需要輸入兩次圖片。

另外，求出統計數據後，白平衡以及伽瑪校正的輸出像素僅需知道對應的輸入像素的值便能完成計算，資料流的優化空間有限，因此本研究將著重優化第一次圖片輸入執行的資料流。第一階段，我們以去馬賽克的輸出能夠為去噪作為輸入使用為目標。

如 Fig.4 所示，若是希望在去噪輸出能夠得到 1x1 的像素，則在去噪需要有 3x3 的輸入像素，推得去馬賽克需要有 5x5 的輸入像素。因此若是希望去噪輸出能夠得到 n x m 的像素，則在去噪需要有 (n+1) x (m+1) 的輸入像素，推得去馬賽克需要有(n+2) x (m+2)的輸入像素。第二階段我們在各自的階段中提升像素使用率。若計算順序是由上至下，則可以與上下的像素共用像素資料，左右亦然。我們的設計將圖片以 4 列為一個輸出單位，以 Fig.5 中的順序輸出：

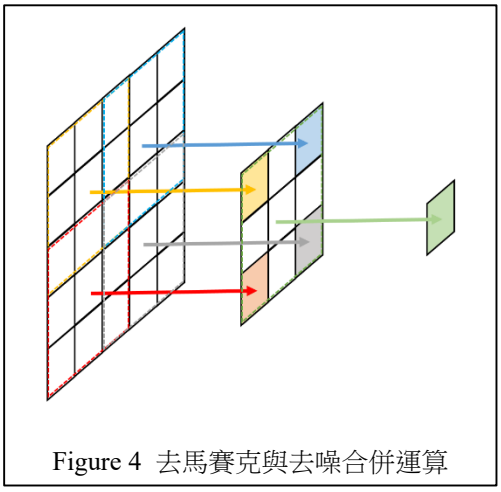

Figure 4 去馬賽克與去噪合併運算

如此一來可以藉由上下共用，左右共用來提升像素使用率。 在下段將會以演算法細節來說明各自需要使用的儲存空間。

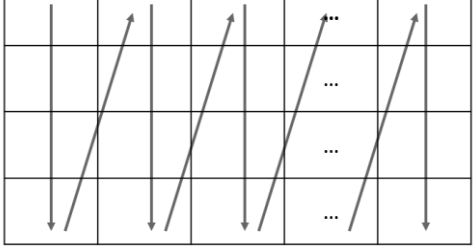

Figure 5 像素輸入順序

b. 暫存(Buffer)

在上述當中，去馬賽克的內插算法以及去噪的中值算法都會讓圖片在兩個維度上各少一個像素單位。因此在圖片以 4 列為一個輸出單位的情況下，去噪的階段需要以 6 列作為一個輸入單位，而去馬賽克的階段需要以 8 列作為一個輸入單位。在去馬賽克這個階段當中，因為計算拜爾模式的演算法是以 3x3 的窗口進行運算，且輸入順序如上圖所示（在輸入到第三欄的時候，第一欄的像素資訊已經被窗口運用完畢），因此至少需要儲存 2 欄像素值，為求運算方便以 3 欄的方式儲存。而同樣在去噪的階段因為中值也因為 3x3 的窗口，需要儲存 4 列中的 2 欄的像素。

c. 流水線優化(Pipeline optimization)

(1) 去馬賽克(Demosaic)

如前所述，在我們設計中，24 個像素的數值會被暫存進一個 8x3 的二維暫存器當中，接著會根據所在的位置對九宮格內的某些像素進行加總，並由控制模組選擇要輸出哪一個像素的數值。根據去馬賽克的演算法，有些像素的輸出是 4 個像素的加總值、有些是 2 個像素的加總，有些則是輸出九宮格中央像素的數值。舉例來說，對照 Fig.7，有些輸出需要加總 1 號、3 號、7 號以

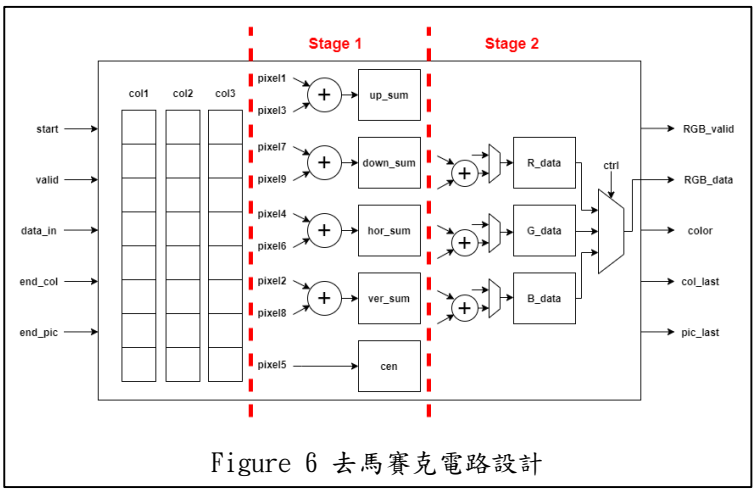

Figure 6 去馬賽克電路設計

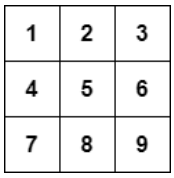

Figure 7 九宮格示意圖

及 9 號像素的數值，有些需要加總 4 號以及 6 號像素的數值，有些需要加總 2 號以及 8 號像素的數值，有些則是直接輸出 5 號像素的數值。而為了避免直接加總 4 個像素數值會產生過長的關鍵路徑(Critical path)，加總部分的電路採用流水線設計，將加總的運算分成兩個階段完成。一個階段只會加總 2 個像素的數值，並儲存到暫存器當中，下一個階段再從這些暫存器或是前面儲存原始像素的暫存器選取資料進行加總。

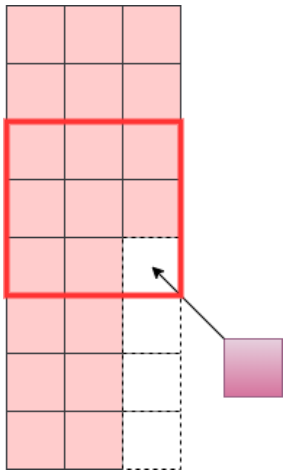

Figure 9 去馬賽克輸出條件

Fig.8 為控制輸入資料要存入哪一個暫存器的有限狀態機(Finite state machine)。每次重新開始時，會先依序讀取第一欄與第二欄的 16 個像素，再讀取第三欄的前 2 個像素。接著每當一個像素輸入之後，如 Fig.9 所示，以其左上角像素為中心的九宮格的像素數值即都被暫存起來了，故此時該像素位置的紅、綠、藍數值都可以被計算而輸出。如此重複 6 次，即可將第二欄的去馬賽克結果進行輸出。col3 暫存器被填滿之後，接下來暫存的窗口會往右移動一個像素，意即 col1 暫存器會被 col2 暫存器中的數值取代，而 col2 暫存器會被 col3 暫存器中的數值取代，此時 col3 暫存器便可以開始存放第四欄中的前 2 個像素。以此推類，隨著暫存的窗口持續右移，即可將整張圖片的 8 列都處理完畢。接著再回到第一欄，並往下移動 4 個像素重複上述步驟，就可以逐步將整張圖片處理完畢。Fig.10 為控制輸出值的有限狀態機，如上所述，每個加法運算會分成兩個階段完成，所以在階段二結束時，紅、綠、藍的數值即完成計算。然而為了配合後面的去噪模組輸入，3 個顏色的數值會分 3 個週期依序輸出。在 STAGE 2 的狀態中即可輸出紅色數值，接著依序輸出綠色數值以及藍色數

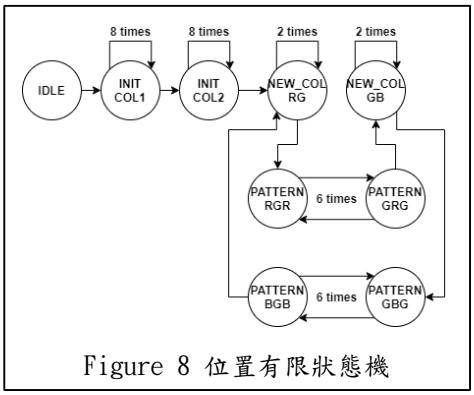

Figure 8 位置有限狀態機

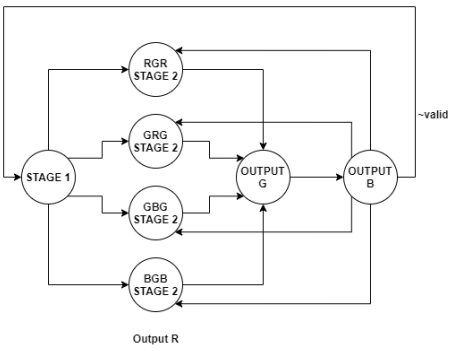

Figure 10 輸出有限狀態機

值，再回到 STAGE 2 的狀態輸出下一個像素的紅色數值。以此推類即可完成整張圖片的去馬賽克輸出。

(2) 去噪(Denoise)

在整個除噪過程中，採用中值濾波器進行運算，因此需要涵蓋的運算單元包含加法運算以及算出平均值的常數除法器。除此以外在每個輸出像素中，中值運算所需要的 9 個像素點一部分來自此模組中的矩陣。此模組的關鍵路徑則是通過矩陣取值後進行加法及常數除法器的運算。為了能夠將每個輸入進來的像素使用效率提高，如前述所提到我們將暫存 6×2×3 (列×行×通道) 個像素值實際輸出以每 4 列全彩圖檔的顏色為一個單位，在這 4 列中從上到下，由左而右依序輸出。在實作上進行兩部分的優化。

第一部分是採用流水線設計分成三個階段：Stage 1 是將從矩陣取出所需要的值，Stage 2 則是加法運算，Stage 3 是常數除法運算。

第二部分則是在加法運算上減少關鍵路徑。若以最直觀的設計，在 Stage 2 必須要有 9 個數字相加，如此至少需要有四層樹狀的加法器才能達成，不管是在加法法器數量或是關鍵路徑上都是負擔。

以下圖為例，我們假設下圖為紅綠藍中其中一個通道的計算。我們可以發現在計算 Fig.11 灰色位置的輸出像素時所需要的是深色身熱紅藍黃三塊分別是 3 個像素的加總的總和，然而在輸出像素是由上至下，前面兩塊黑色的輸出像素已經在計算過程中將深藍紅色的 3 個像素加總計算過，可以將其存下來。

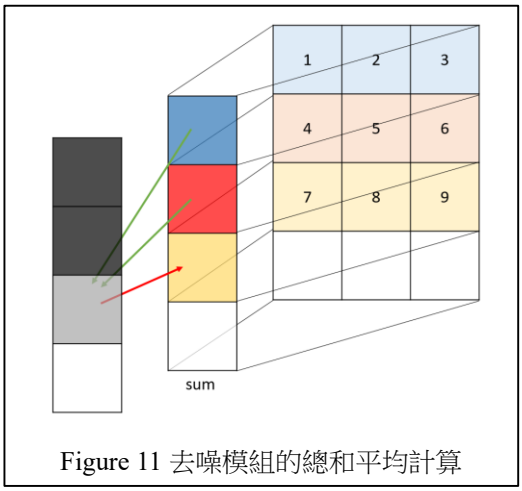

Figure 11 去噪模組的總和平均計算

因此，在求取灰色方格(就是 9 格方格的平均值)時，由於上兩列的總合已經被計算完畢(Fig.11 的深藍，深紅方格)，在 9 號方格，僅

需對 7,8,9 方格及深藍深紅進行加總僅需計算黃色的 3 個像素(紅色箭頭)以及 Fig.11 中紅藍黃格子的總和，因此將其儲存起來後可以發現只需（2 個 10-bit 的數字以及 3 個 8-bit 的數字），減少了 1/4 的關鍵路徑。總和需要額外的 2(Fig.11 中深藍、深紅)×3（顏色通道）個 10-bit 暫存器的空間存放計算結果。

在計算順序上：先輸入同個像素的三個顏色、在將同一欄的 4 列中的像素由上而下輸入、由左而右將每一欄的像素輸出，因此在狀態（state）上分成三個：INIT12, INIT2, OUT。

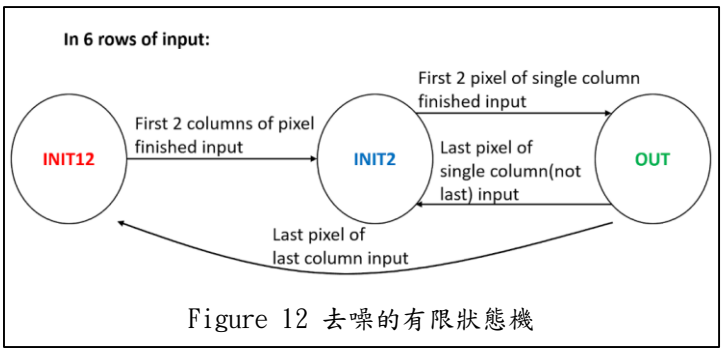

Figure 12 去噪的有限狀態機

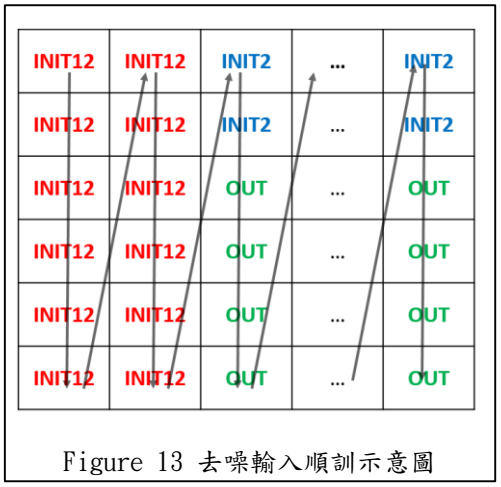

Figure 13 去噪輸入順訓示意圖

以每六列的輸入像素為一循環，狀態的轉換主要是要以輸入像素的位置而定。每個狀態都會將輸入像素數值存入暫存器中，而只有在 OUT 狀態會將計算結果輸出。

(3) 統計值計算(Statistic Calculation)

(i) 平均

透過三個累加器，計算紅綠藍三顏色通道的總和值，待遍歷圖片中所有像素點後，通過先比較後相減（compare and subtraction）的方式實現除法器，以取得三個通道的平均值。

(ii) 增益值

因為增益值計算需要 16-bit 的除法運算，為了減少關鍵路徑，我們採取 16 個週期先比較後相減的方式實現 16-bit 的除法器，以下列四個式子計算出 RGB 三個通道的增益係數。

$$K = (r\_mean + g\_mean + b\_mean) / 3 \quad (1)$$

$$K\_R = K / r\_mean \quad (2)$$

$$K\_G = K / g\_mean \quad (3)$$

$$K\_G = K / g\_mean \quad (4)$$

第一次輸入的資料處理到這個統計數據計算完成後結束。

(4) 白平衡(WB)

此為第二次輸入資料的第一個階段。經過平均模組 增益值模組 得到紅綠藍三個通道的增益值，此為第一次輸入資料將增益值及每個像素的色值輸入，乘上增益值輸出，得到白平衡後的色值。

3. 模式選擇

為了使得此設計可以選擇任意階段的組合進行圖片處理，我們的設計以下圖的方式將所有模組包裝起來，並且加入一個輸入訊號 mode，用以選擇要讓輸入經過哪些階段的圖片處理。舉例來說，如果只想讓圖片進行去噪的處理，去噪模組前的多工器將會選擇 data_in 作為輸入，而 data_out 的多工器則會選擇去噪模組的輸出作為輸出。

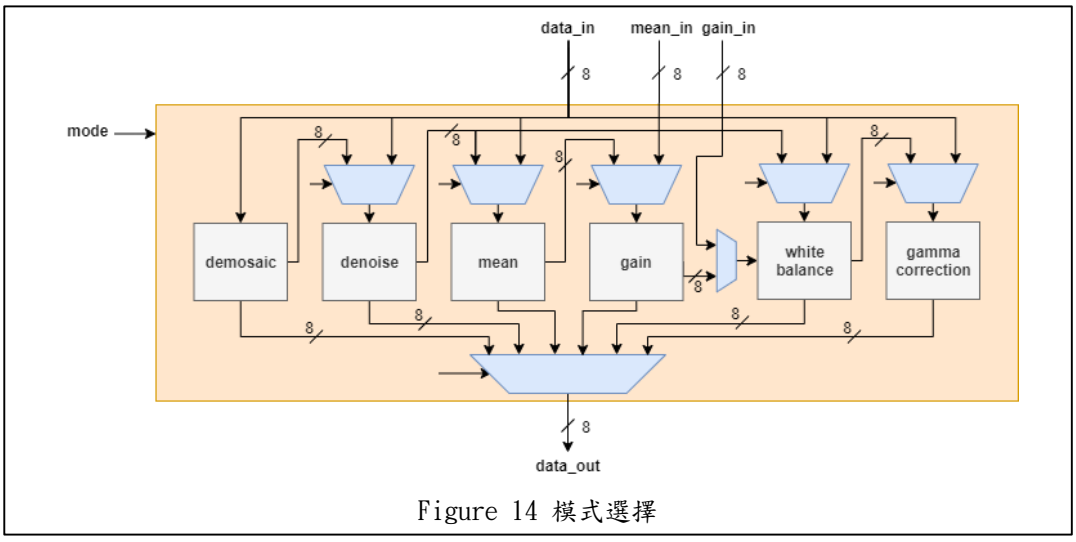

Figure 14 模式選擇

4. 彈性化的圖片尺寸

為了可以處理不同大小的圖片，在統計數據的階段需要根據不同的大小去進行不同除數的除法運算。因此在統計數據的加總階段時，會利用一個計數器

去計算圖片的總像素數，再將像素總和除以該數值以求得平均。

5. 伽瑪校正

由於色值的值域範圍為[0,255]，我們以查表的方式來實現伽瑪校正中的非線性方程式：輸出色值 = 輸入色值 $^{1/(2.2)}$ 。

6. 軟體驗證及資料生成

我們透過 python 進行了位元真(bit true)的軟體模擬，同時也在每一個圖像處理的階段，生成了硬體測試所需的輸入資料以及標準資料。

三、結果與討論(Result and Discussion)

1. 去馬賽克及去噪暫存大小對於執行週期數量的影響

在資料重複利用的項目中，傳統影像處理在去馬賽克以及去噪兩個階段時，將圖片分成每 4 列為一個處理單位以提升資料重複利用的效率。接下來我們將以去馬賽克以及去噪的輸出列數如何影響資料重複利用率與處理週期探討。

首先定義資料重複利用率的效率為：輸出像素數量(RGB)/輸入像素數量(Bayer Raw)。

我們先以一個 n 列的處理單位探討，會發現因填充像素的關係，在去馬賽克的輸入必須輸入 n+4 列處理。所以當 n 的數量非常大時，效率將約接近 100%。

因此在面積固定的情況下理論上將 n 提高到最大值便可以將執行時間壓到最低。

而因為在這兩個階段中，拜爾圖檔的圖片大小與全彩圖檔的圖片大小為 1：3，受限於輸入輸出的頻道僅有 1：1，輸入的像素必須遵守 3 個週期內最多送一次的限制。所以對於在經由前處理後像素數量為(n+2) × (m+2)的拜爾圖檔圖片，至少需要經過

3 × (n+2) × (m+2) × (n+4) / n 的執行週期才可以完成這兩個階段。

2. 硬體結果圖

a. 火車樣圖

Figure 15 原始拜爾圖檔(以黑白顯示)

Figure 16 去馬賽克圖檔

Figure 17 去噪圖檔

Figure 18 白平衡圖檔

Figure 19 伽瑪校正圖檔

Figure 20 原始圖檔（經由電腦處理）

b. 室外樣圖

Figure 21 原始拜爾圖檔(以黑白顯示)

Figure 22 去馬賽克圖檔

Figure 23 去噪圖檔

Figure 24 白平衡圖檔

Figure 25 伽瑪校正圖檔

Figure 26 原始圖檔（經由電腦處理）

3. 自動化繞線結果(APR Result)

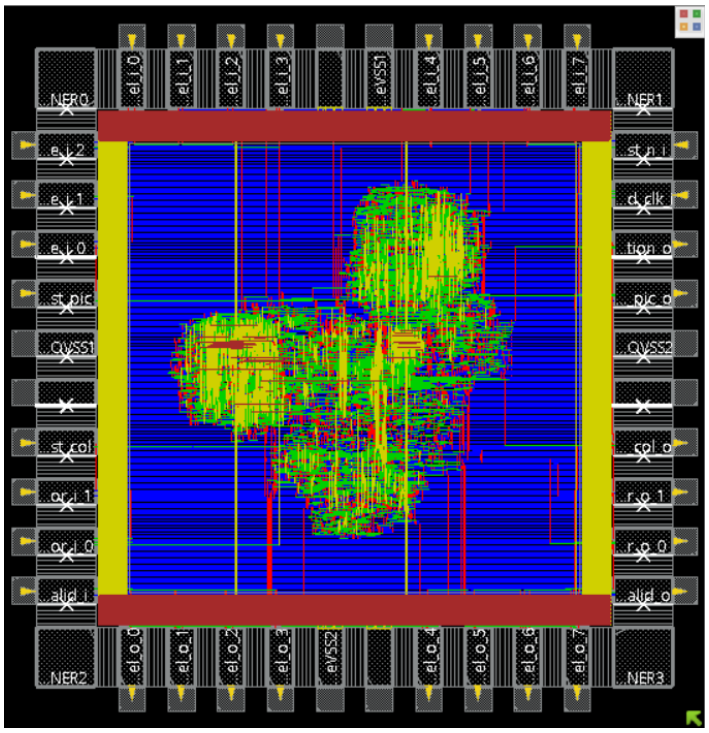

Figure 27. innovus 軟體繞線結果

如 Fig.27 所示

4. 晶片規格(Chip specification)

Description	
Process	UMC 0.18um Mixed-Mode and RFCMOS 1.8V/3.3V
Power Supply	3.3V

Frequency	4.17MHz
Core size (μm2)	1059.58x1059.84
Chip size (μm2)	1499.78x1500.24
Power	27.878mW
PADs	34

四、結論(Conclusion) 在本研究中，我們將傳統影像處理的流程合併運算，將原本的 4 個步驟分成兩個階段進行圖片的輸入，以大幅減少晶片向記憶體索取像素資料的次數。另外，在不使用 SRAM 的限制下，我們提出了一個高效率的資料流，透過增加像素的重複使用率，更進一步減少晶片與記憶體的互動頻率，進而加速整個 ISP 的運算流程。除此之外，我們的設計也可以針對不同大小的圖片進行處理，並且可以自由選擇任意的階段的影像處理，藉此讓此設計具有非常高的彈性。

另一方面，根據前文的討論，增加晶片的暫存器數量即可以減少與記憶體的溝通次數進而降低整體運算時間。本研究中，也透過將直觀的 3x3 處理單位提升至 8x3 大幅提升所花的週期數。然而，當暫存器數量越來越多，可以減少的週期數將會越來越少，意即此優化效果會有邊際效應。所以未來的研究可以針對晶片面積以及運算速度的平衡進行討論，以在一定的面積限制下，設計出運算效率更佳的 ISP 晶片。

五、參考文獻(Reference)

[1] SAKAMOTO, Tadashi; NAKANISHI, Chikako; HASE, Tomohiro. Software pixel interpolation for digital still cameras suitable for a 32-bit MCU. IEEE Transactions on Consumer Electronics, 1998, 44.4: 1342-1352.