Hardware Implementation of full-HD 30fps Guided Image Filter

Shih-Wei Hsieh (r07943017), Yu-Chen Chou (r07943018)

***Abstract*—Edge-preserving filters, such as median filter, bilateral filter, anisotropic filter, and guided image filter, have been widely used in various modern computer vision tasks. Among them, guided image filter is the most efficient and powerful one. In this final project, we implemented a real-time, full-HD guided image filtering hardware based on double integral image engine. We explored different architecture by applying data-interleaving on different parts of our design, and examined the power-area trade-off between them. Synthesized with TSMC 40nm technology, our design can run at 100 MHz with 3.18 KB on-chip memory while producing full-HD (1920x1080) images at 30 frames/sec.**

Keywords- guided image filter, hardware accelerator

I. INTRODUCTION

邊緣保留濾波器(edge-preserving filter)是一個在電腦視覺與影像處理中很常使用的技巧，這類濾波器的功用在於可以在影像的平滑區域扮演像是高斯濾波的效果來消除雜訊，但不會像高斯濾波器一樣將影像的邊緣模糊化，而能保持影像原有的邊緣，同時去除雜訊(如 Figure 1)。

Fig. 1. 原圖/Guided image filtered result/ Gaussian filtered result

這類的濾波器有相當多不同的種類，包含 anisotropic filter, bilateral filter, guided image filter 等等。其中最近期，也是目前最常被使用的一種就是 guided image filer (GIF)，比起其他種類，像是 bilateral filter 的速度 $O(r^2)$，它的優點在於速度與 window size 無關，也就是說速度上是 O(1)。

除了去除雜訊外，GIF 還有很多的其他應用，像是影像除霧(iamge dehazing)、高動態範圍(HDR)、影像去背(image matting)、立體匹配(stereo matching)等等。以最近常見的自駕車研究為例，立體匹配可以用來求車前的視差(disparity)，進而轉換為深度圖(depth)，由於這類應用明顯需要處理時間是即時的，所以有一個 real-time GIF hardware 會是在這樣的應用當中不可或缺。

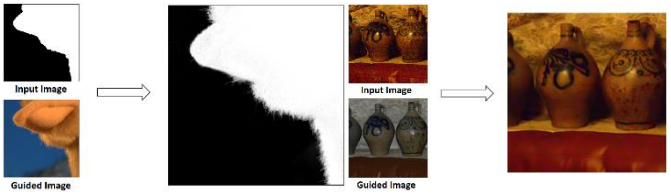

Fig. 2. GIF 的兩個應用。左: image matting; 右: flash/no-flash denoising

GIF 的演算法有兩個 input，其一為 input image (p_i)以及 guidance image (I_i)，輸出的結果在 gradient 上需要I_i相近，但整體的顏色、結構等等則要跟p_i相近。為了達到這個目的，GIF 的第一步是將影像切成小 patch，找出一個 guidance image 的線性模型得到輸出的 patch:

$$\text{Output } q_i = a_k I_i + b_k, \forall i \in \omega_k \quad \text{Local window}$$

Guidance image

GIF 的目的就是找出上式中的a_k和b_k，來讓 output q_i與 input image 的p_i相近，可以觀察到上式中$\nabla q_i \propto \nabla I_i$，也就是說輸出圖會與 guidance 有一樣的邊緣，GIF 找出a_k和b_k的作法是去最小化下面這個 energy function:

$$E(a_k, b_k) = \sum_{i \in \omega_k} \underbrace{\left((a_k I_i + b_k - p_i)^2\right.}_{\text{Data term}} + \underbrace{\left.\epsilon a_k^2\right)}_{\text{Regularization term}}$$

上式中前項希望q_i跟p_i的差異愈小愈好，後項則是係數的 regularization，其中ϵ是使用者的自訂參數。這個能量函數本身是一個 linear ridge regression，答案可以由對能量函數取微分得到，分別如下:

$$a_k = \frac{\overbrace{\frac{1}{|\omega|}\sum_{i \in \omega_k} I_i p_i - \mu_k \bar{p}_k}^{\text{Cov(I, p)}}}{\underbrace{\sigma_k^2}_{\text{var(I)}} + \epsilon} \qquad b_k = \bar{p}_k - a_k \mu_k$$

如此對每個 patch 都這樣找出對應的a_k, b_k，並且得到對應的q_i，由於每個 pixel 都會出現在多個不同的 patch 當中，在每個 patch 裡同一個 pixel 可能有不同的數值，因此最終的 pixel value 會再將不同 patch 中同個 pixel 的數值平均起來得到:

$$q_i = \frac{1}{|\omega|} \sum_{k | i \in \omega_k} (a_k I_i + b_k)$$

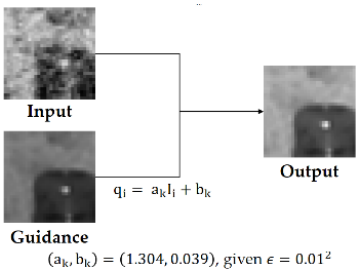

Fig. 3. Single patch output result

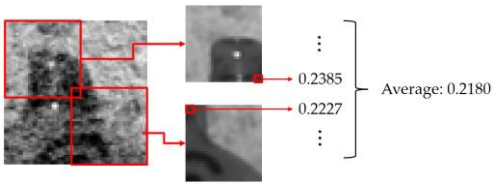

Fig. 4. Final pixel value is averaged from all patches containing the pixel

GIF 的演算法可以分為大致兩個部分，一個是估算a_k/b_k，另一個是由a_k/b_k算出q_i。下圖是數學式、軟體 pseudo code 以及硬體架構的對應關係。計算a_k/b_k，在軟體上會由中間上面的紅色框框計算得來，硬體上會是我們架構的前兩個 stage。由a_k/b_k算出q_i，在軟體上則由中間下面的紅色框框計算得來，硬體上則是我們架構的後兩個 stage。

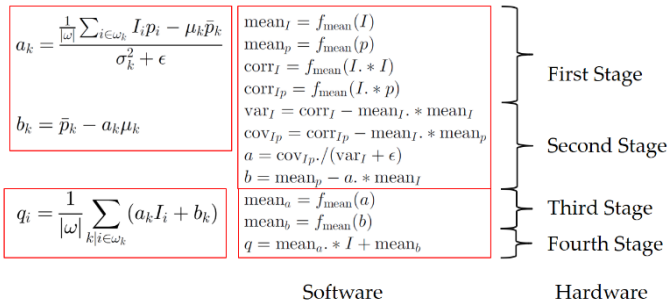

Fig. 5. GIF pseudo code and corresponding hardware stage

在硬體實作上我們參考[1]這篇 Paper，主要架構都跟他們類似，除了將它實做出來並作分析之外，我們也額外做一些 architectural transform，並比較轉換前後的設計在 power/ area 上的差異。在以下的報告中，Section 2 會介紹我們硬體架構和實作上的細節，以及合成後的結果。Section 3 介紹做架構轉換的方式以及結果比較。Section 4 則總結這篇報告。

II. Design Architecture

我們的硬體架構圖如下圖(Figure 6)所示，一共分成四個階段，第一和第三個 stage 是做 integral image，第二個 stage 計算出a_k和b_k，第四個 stage 則是計算出q_i。以下的 Subsection 會依據四個 stage 依序做介紹。

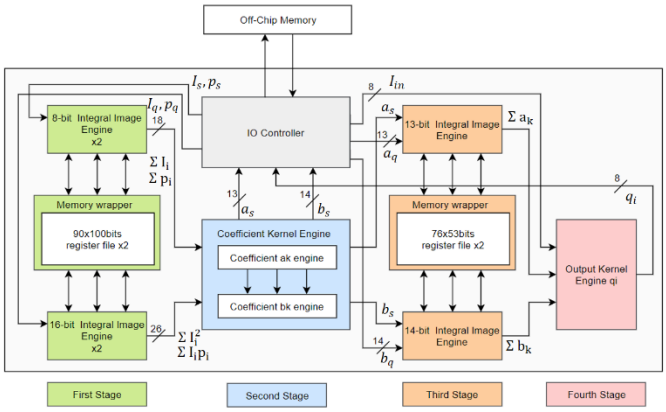

Fig. 6. Overall design architecture

A. *First Stage: Integral Image Engine*

Integral image 的計算是 GIF 當中最重要的一個部分，也是為何 GIF 可以跑在 constant time 的原因。這一步的目的就是要為影像中每一個像素周圍開出 window，算這些 window 內所有像素數值的總和，因為最終輸出的結果是每個像素周圍 window 的總和，也因此叫做 integral image。

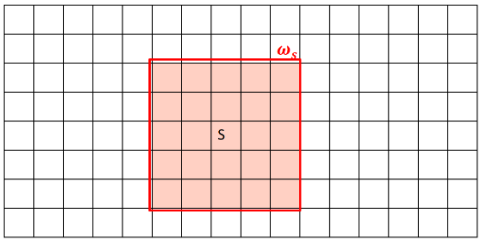

Fig. 7. Expected output for pixel s = $\sum_{i\in\omega_s} p_i$ (assume window size =5)

為了達到每個像素在 constant time 內算出結果，naïve 的一一針對每個像素周圍的 window 做加總是不可行的，這邊我們使用的方法如下。定義一個參數稱為II^s，內容是 pixel s 往左到底且往上 window size 個像素中，所有像素值的總和。如下圖(Figure 8)中，II^S就是紫色區域的總和。

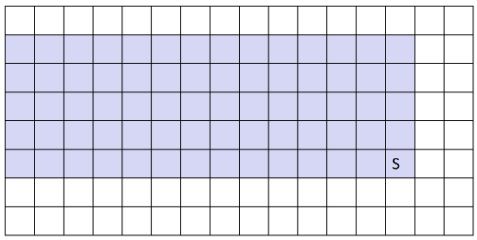

Fig. 8. II^S = sum of all pixels in colored area

有了以上的假設，我們可以推斷出II^S彼此之間的關係，如下圖 (Figure 9)所示:

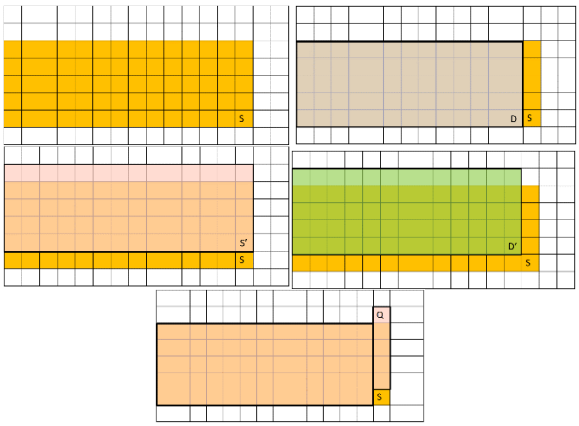

Fig. 9. IIE relations (Integration)

由上圖可以推斷，$II^{D} + II^{S'} - II^{D'}$的結果是得到上圖中最後一張圖的黃色區域，比起我們想要的II^{S}還差 pixel s 和 pixel q 的值 (I_s, I_Q)，因此最後我們得到下列關係:

$$II^{S} = II^{D} + II^{S'} - II^{D'} + I_S - I_Q$$

也就是說，只要擁有一個 pixel 左上($II^{D'}$)、上($II^{D'}$)、和左邊($II^{D'}$)的 II 值，再加上I_Q和I_S，就可以算出新的II^{S}，因此無論 window size 的大小，我們都可以在固定 operation 內算出新的II^{S}，而計算順序則需要從左上計算到右下。

任意一個 window 內的總和，則可由兩個 II 值相減得到，如下圖所示，紅色框框的 window 內的像素值總和，即是:

$$\sum_{i \in \omega_k} I_i = II^{S} - II^{A}$$

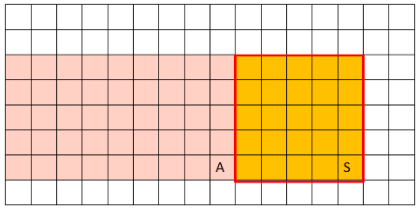

Fig. 10. IIE relations (Extraction)

統整以上的討論，一個像素需要計算出它的 local window 加總，共需要四個 II 值($II^{D'}, II^{S'}, II^{D}, II^{A}$)以及兩個像素值$(I_s, I_Q)$。在我們的實做中，四個 II 值都存在 local memory 當中，而I_s, I_Q則跟外部的 memory 要資料。

Local memory 的配置如下圖 (Figure 11)所示，我們僅需存下紅色標示的部分即可，而雖然我們沒存$II^{D'}$，但因為處理是由左向右，所以$II^{D'}$就是前一個 cycle 的$II^{S'}$，只需要在電路內加上一個 register 即可。同理II^{D}也是由II^{S}經過 shift register 得到。需要讀取的剩下$II^{S'}$和II^{A}，而計算出來的II^{S}則要存回 memory。

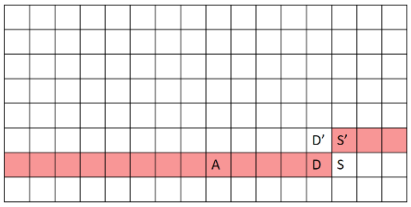

Fig. 11. Memory view for IIE

Stage 1 總共要算四個 IIE($f_{mean}(I)$, $f_{mean}(p)$,$f_{mean}(I^2)$, $f_{mean}(I * p)$)，算出來的 II 分別是 21/21/29/29 bits，影像寬度切 stripe 之後是 180 pixels，所以總共需要的 memory 大小是 180 words x (21+21+29+29) bits。由於我們每個 cycle 需要兩個 read 和一個 write，所以我們最後選擇用兩個 two-port 的 register file，大小分別是 90 words x 100 bits。(Register file 用 memory compiler 產生)

將前面所述寫成電路則如下圖所示，這個部分在之後還會在使用到。

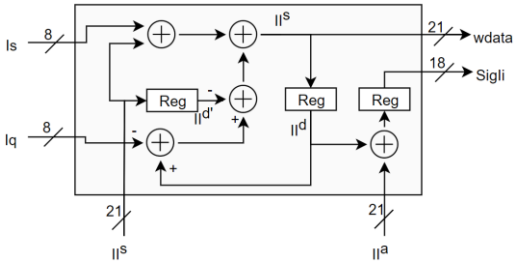

Fig. 12. IIE hardware

B. *Second Stage: Coefficient Kernel Engine*

結束 stage 1 的部分之後，stage 2 要由 stage 1 的結果計算出係數a_k，b_k:

$$a_k = \frac{\frac{1}{|\omega|}\sum_{i \in \omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon} = \frac{|w|\sum_{i \in w_k} I_i p_i - \sum_{i \in w_k} I_i \sum_{i \in w_k} p_i}{|w|\sum_{i \in w_k} {I_i}^2 - (\sum_{i \in w_k} I_i)^2 + |w|^2 \epsilon}$$

$$b_k = \bar{p}_k - a_k \mu_k$$

其中a_k中的ΣI_i, Σp_i, ΣI_i^2, $\Sigma I_i p_i$以及b_k中的p_k, μ_k都是在第一個 stage 算出來的結果。由於b_k計算需要a_k的結果，所以我們需要先算a_k，再算b_k:

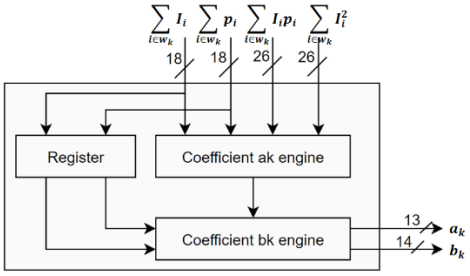

Fig. 13. Coefficient Kernel Engine

接下來會分開討論a_k和b_k的電路，下圖是 coeffcient a_k engine 的部分:

$$a_k = \frac{\frac{1}{|\omega|}\sum_{i\in\omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon} = \frac{|w|\sum_{i\in w_k} I_i p_i - \sum_{i\in w_k} I_i \sum_{i\in w_k} p_i}{|w|\sum_{i\in w_k} I_i^2 - (\sum_{i\in w_k} I_i)^2 + |w|^2\epsilon}$$

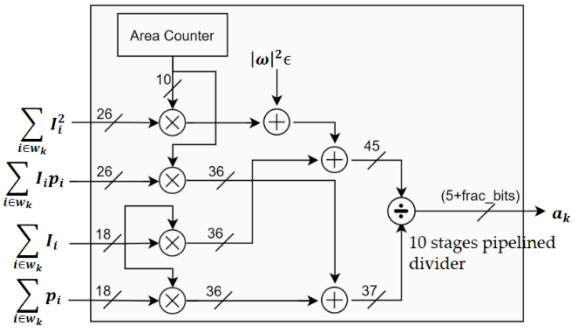

Fig. 14. Coefficient a_k engine

上圖中的 area counter 對應到式子中的|ω|，也就是 window size 的大小，由於處理的過程中需要處理邊界條件，在邊界的部分 window size 會比較小，所以需要用一個 counter 來數，而非直接用一個固定的常數。電路中除法器的部分，我們使用 design ware 提供的 pipelined 除法器，並設定使用 10 個 stage 來做合成 (47 bits/ 37 bits)。

在計算a_k之前，所有的運算都是沒有喪失任何精準度的，從a_k開始，我們有去對小數點的 word length 做取捨，以避免 word length 愈變愈長，在這個 section 的最後我們會對 word length 取捨與 performance 的關係做分析。

$$b_k = \bar{p}_k - a_k \mu_k$$

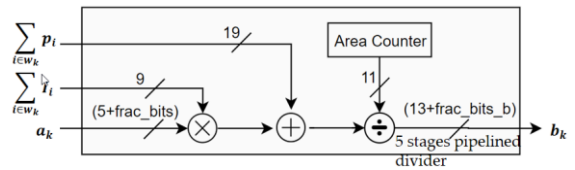

Fig. 15. Coefficient b_k engine

上圖是b_k的電路，除法器的部分由於這邊只要處理 31 bits/ 11 bits，我們選擇只用 5 個 pipeline stage 來合成。同a_k的部分，這邊我們也有分析b_k的小數位數對 performance 的關係。

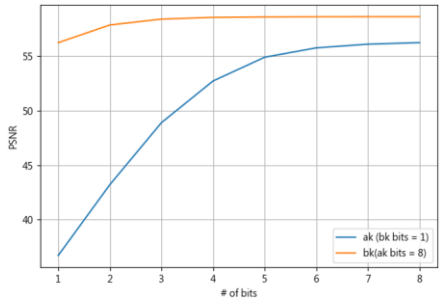

Fig. 16. Word length to performance (PSNR) analysis of a_k and b_k

上圖是a_k以及b_k的小數位數(bits)對比最終輸出的 PSNR 對比圖，PSNR 的部分是由 Verilog 輸出與 matlab 計算的結果計算出來的。可以觀察到a_k小數位數對 PSNR 的影響比較劇烈，而b_k則幾乎沒有什麼影響。一個原因可能是因為b_k的計算過程中需要使用到a_k的結果，所以若a_k變得不準，b_k也會同時受到影響。

分析過程中我們都會固定除了# of bits 以外的其他參數，但實際上的 PSNR 也會受到 window size 的大小影響，而這個維度我們就沒有再作分析(我們的應用固定 window size = 31 x 31)。基於不同的 application 所需要的準度不同，有了這樣的分析，我們就可以針對特定的應用選擇最佳的 word length。在這邊我們最終選擇的(a_k, b_k)小數位數分別是 (8, 1) bits。

C. *Third Stage: Integral Image Engine for a_k and b_k*

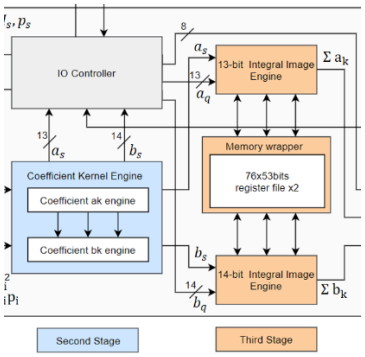

Fig. 17. Integral Image Engine for a_k and b_k

第三個 stage 跟第一個 stage 很像，也是由 IIE 所組成，不同的地方在於 bit 數的不同，IIE 也只有 2 個，因此 on-chip memory 的大小也會有所不同。這個 stage 的 II 算出來分別是 26 和 27 bits，stripe width 則是 150 pixels (與第一個 stage 的 180 pixels 不同的原因，是因為每過一次 integral image，長寬就會因邊界條件各減少 window size-1)，同樣由於需要兩個 read+ 一個 write，所以需要兩個 2-port register file，理想的 register file 大小是 75 words x 53 bits，但由於 memory compiler 限制 word size 必須至少是 2 的倍數，因此最後選擇用 76 words x 53 bits。

D. *Fourth Stage: Output Kernel Engine*

這部分除了 bit 數稍有不同之外，架構基本上跟第二個 stage 的b_k engine 一模一樣，因此不再贅述。

E. *Achieving speed of 30fps*

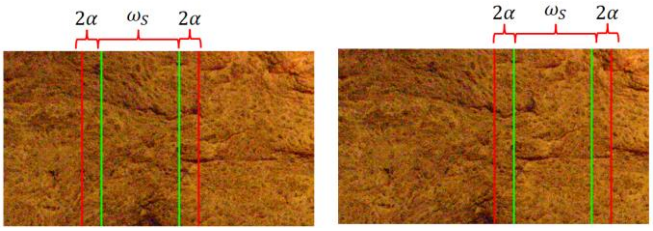

Fig. 18. Striped based processing

我們在處理一張 1920*1080 的影像時，會將整張圖切成一張一張寬度ω_s的 stripe ，且因為邊界條件的處理，左右兩邊需要多處理各2α的寬度(window size: $(2\alpha+1, 2\alpha+1)$)，因為2α是固定的，如果ω_s愈寬，padding 造成的計算時間的 overhead 就愈小，我們可由此算出一個 total computation cycle 的理論值:

$$Total\ computation\ cycle: \lceil W_f/\omega_s \rceil * (\omega_s + 4\alpha) * (H_f + 3\alpha)$$
$$, where\ (H_f, W_f) = (1080, 1920)$$

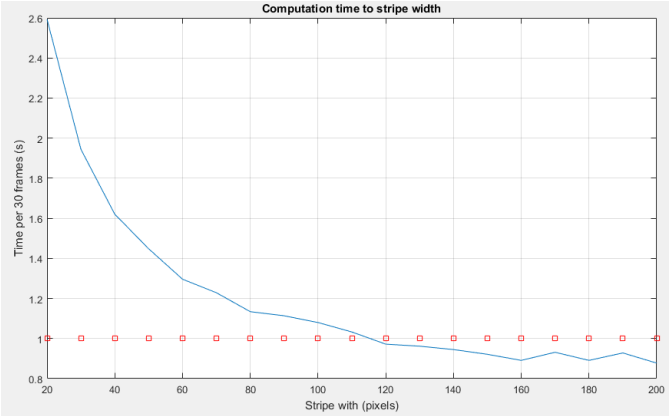

Fig. 19. Time per 30 frames (seconds) to Stripe width (pixels)

上圖是處理 30 個 frame 所花的時間(秒數)對應 stripe 寬度的關係圖，stripe 寬度如果愈寬，處理時間就愈短，但是代價就是 on-chip memory (1^{st} stage/ 3^{rd} stage)會線性增加。我們的目標是希望每秒 30 個 frame，所以最後選擇剛剛好能達到這個標準的 stripe 寬度，也就是 120 個 pixel。模擬出來的結果，每個 stripe 所花的時間是 2.025ms，換算成 30 個 frame 就是花 0.972 秒，有確實符合期望的 30fps 速度。

```
Q Pattern     129599 is passed !. Expected candidate =  16, Response candidate =  16 !!
--------------------------- Simulation Stops !!---------------------------
================================================================================


        ****************************
        **                        **       /|__/|
        **  Congratulations !!    **      / 0,0  |
        **                        **     /_ ^ ^ _\  |
        **  Simulation Complete!! **   /^ ^ ^ ^ \ |W|
        **                        **  |^ ^ ^ ^ ^|W|
        ************** ************   \m___m__|_|


================================================================================
Simulation complete via $finish(1) at time 2025480 NS + 0
./tb_IIE.v:189              $finish;
```

Fig. 20. Verilog simulation result

合成的結果我們會統一在最後的 section 做討論，下一個 section 接著說明我們做架構轉換的方式。

III. ARCHITECTURAL TRANSFORM

由於整個硬體的架構中有不少重複的地方，像是第一個 stage 的 4 個 IIE 和第三個 stage 的 2 個 IIE，這些地方可以靠著一些 architectural transform 來降低 area。這個 section 主要介紹我們如何用 data-interleaving 的方式，減少需要使用的 IIE 和除法器個數。

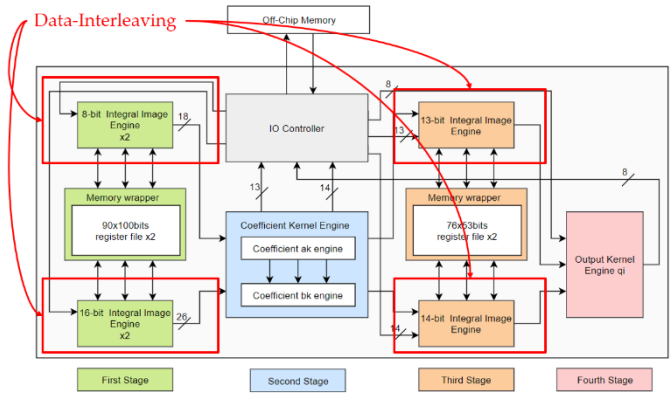

Fig. 21. Perform data-interleaving on IIE

A. *Data-interleaving on IIE*

架構轉換前的 IIE 處理方式如下圖所示，資料輸入時會同時輸入兩組的像素值(假設為 (I_S, I_Q) 及 (p_S, p_Q))，輸出則是局部 window 的積分影像 (ΣI_i及Σp_i)。這部分對應上課的投影片，兩個 IIE 分別就是兩個 PE，輸入的資料也是平行輸入、平行輸出。

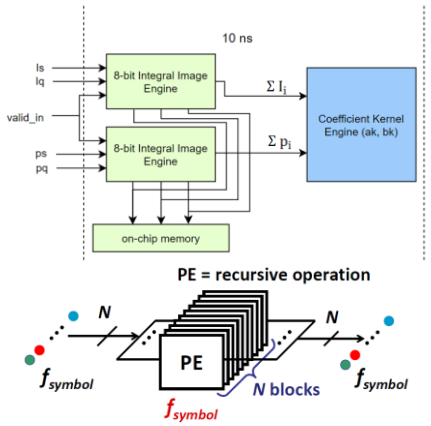

Fig. 22. Original IIE structure

Data-interleaving 方式如下圖，使用兩倍的 interleaving 需要跑在 2 倍的頻率(原本 100MHz，interleaved 部分則是 200MHz)。每個 cycle 交替處理來自 I 和 p 的資料，輸出的ΣI_i需要用先暫存器儲存，並和下一個 cycle 的Σp_i一起傳進下一個 stage。

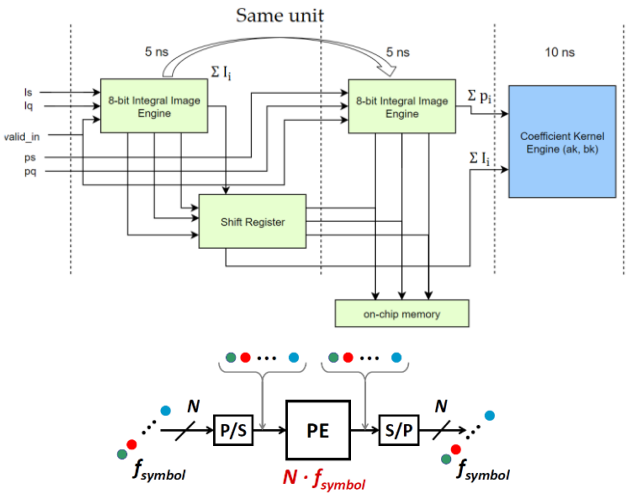

Fig. 23. Interleaved IIE structure

IIE 內部的電路存在一些 feedback-loop (如下圖左邊的兩個 register)，這些部分的 pipelining stage 在 interleaving 時需要變成原本的兩倍，來自不同 source 的資料才能做到正確的對應:

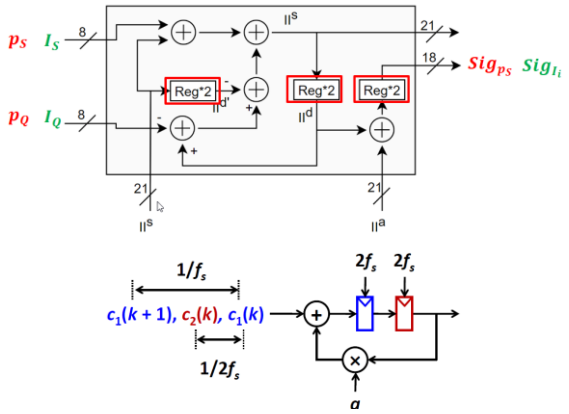

Fig. 24. Interleaved IIE internal structure

IIE 的架構轉換會應用在第一個 stage 和第三個 stage 的 6 個 IIE 上，結果可以節省到僅使用 3 個 IIE。除此之外，我們先前提到第二個 stage 的b_k engine 和第四個 stage 彼此之間的架構也非常類似，因此這也是我們可以做架構轉換的一個部分。

B. Data-interleaving on pipelined divider

這部分簡單敘述我們除法器的 data-interleaving 方式，由於第二個 stage 和第四個 stage 的除法器都是處理 31 bits/ 11 bits，且原本的 pipelined stage 也都是 5 個 stage，我們可以做除法器的 sharing 來節省空間。

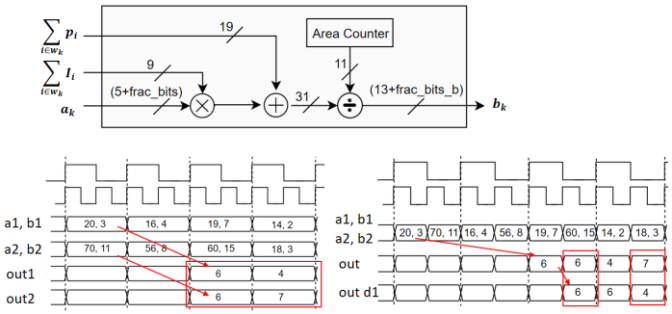

Fig. 25. Interleaved divider architecture

Figure 25 的左下波型圖是原本的設計，兩個除法器分別吃兩組資料，並在同樣的 cycle delay 後輸出結果(圖中兩個 cycle 僅為示意用)，而右下的波型圖則是轉換後的波型，兩組資料交替輸入單一一個除法器，經過轉換前兩倍的 pipeline stage 後輸出結果，其中一個輸出經過一個 clock delay 後，和另一個一起在 10ns 的正緣輸出。

.

IV. SYNTHESIS RESULTS AND COMPARISON

我們總共有合成三個設計，包含原始的設計(Section 2)、IIE 做 data-interleaving 的設計、以及 IIE 和除法器都做 data-interleaving 的設計。原始設計跑在 100MHz，使用 TSMC 40nm technology，standard cell 使用 high V_t、regular V_t、low V_t三種。合成的結果如下表所示:

	A	*B*	*C*
	Original design (100 MHz)	Transformed design (IIE)	Transformed design (IIE+div)
Dynamic Power	5.314 mW	7.503 mW	7.735 mW
Leakage Power	0.9766 mW	1.088 mW	1.033 mW
Total Power	6.290 mW	8.591 mW	8.768 mW
Memory Area	43724.41	43724.41	43724.41
Comb+ Noncomb Area	37733.85	35583.79	33760.77
Total Area (μm^2)	81458.26	79308.20	77485.18
Hvt/Rvt/Lvt ratio	80.00/ 9.00/ 11.00%	64.79/13.97/21.24 %	64.43/15.24/20.34 %

Table. 1. Synthesis result of all designs

上表中，power 的部分是用 primetime 對模擬出的 vcd 作分析得來的。可以觀察到以 power 來說，原始的設計只有消耗 6.3mW，但兩個 transform 過的設計則分別消耗 8.6mW 與 8.77mW，經過 data-interleaving 的設計明顯 power 增加許多，而這也是預期的結果，由於做了 interleaving 的設計需要跑在 5ns，timing margin 上比起原始設計要小很多，可能 tool 就需要用 area 或 power 大但更快的 cell 來合成。

Area 的部分，register file 占的面積三個都一樣，而可以發現做愈多 hardware sharing 的設計，也的確面積愈小，這也是我們預期的結果。

最後一欄是不同 threshold voltage 的 cell 佔的比例，原始設計的 high V_t 比例明顯比其他設計要高的許多，也代表他有比較多的 timing margin，才能用這麼多速度慢而 leakage power 小的 high V_t cell。

這三個設計跑的 simulation 時間都一樣，因此我們在 performance 上是比較他們的 power-area product，會發現 Original design (A)優於 Transformed design (IIE+div) (C) 優於 Transformed design (IIE) (B)。因此若以 PAP 作為評斷標準的話，還是原始的設計為最佳。

以下是我們設計的# of chip pins/on-chip memory size/off-chip memory size:

# of chip pins	on-chip memory size	off-chip memory size
175	3.18 KB	15.326 KB

最後，我們比較比較 Matlab 計算出來的圖和用 Verilog 產生的圖 (Figure 26)，可以發現兩者非常接近，PSNR 計算出來的結果是 49.5341 dB。

Fig. 26. Filtered output by Matlab (top image) and Verilog (bottom image)

APPENDIX: DESIGN WARE PIPELINED DIVIDER ANALYSIS

這個附錄紀錄我們針對硬體設計過程中，所使用到的除法器去進行的分析。我們先以 matlab 生成 1000 組 golden data，去觀察在不同頻率、pipeline stages 下合成所得到的面積、功耗。這裡所要測試的除法器固定被除數為 31 bits，除數為 11 bits，合成同樣是使用 TSMC 40 nm 製程，得出來的結果整理如下表。

從表中可以觀察到時脈週期為 2.0 ns 的面積及功耗明顯比 5.0 ns 都還要大，可以推測說 compiler 為了能滿足時間上的需求而使用較大面積或者是功耗較大的 cell。從這幾組數據功耗和面積的乘積來看，時脈週期 5.0 ns、pipeline stage 為 4 的時候表現最好，其次是時脈週期為 5.0 ns、pipeline stage 為 8 的時候。雖然時脈週期為 2.0 ns 時，除法器的面積和功耗都較大，但他們能有較少的 latency 和運算速度。

Pipeline stages	2	4	8	2	4	8
Cycle time	2.0ns			5.0ns		
Dynamic Power	2.344 mW	2.137 mW	1.083 mW	0.779 mW	0.500 mW	0.499 mW
Leakage Power	0.454 mW	0.208 mW	0.119 mW	0.092 mW	0.057 mW	0.023 mW
Total Power	2.798 mW	2.345 mW	1.202 mW	0.871 mW	0.557 mW	0.522 mW
Total Area (μm^2)	10878.46	7796.25	6466.52	4830.61	4186.50	5739.63
Power*Area (mW*μm^2)	30437.93	18282.21	7772.76	4207.46	2331.88	2996.09
Hvt/Rvt/Lvt ratio	3.91/ 15.86/ 80.24%	16.40/ 25.79/ 57.81%	33.91/ 23.24/ 42.85%	19.67/ 28.85/ 51.47%	37.25/ 33.50/ 29.25%	81.70/ 14.30/ 4.01%

Table. 2 Analysis of different pipelined stages of design ware divider

REFERENCES

[1] He, Kaiming, Jian Sun, and Xiaoou Tang. "Guided image filtering." *IEEE transactions on pattern analysis and machine intelligence* 35.6 (2012): 1397-1409.

[2] Kao, Chieh-Chi, Jui-Hsin Lai, and Shao-Yi Chien. "VLSI architecture design of guided filter for 30 frames/s full-HD video." *IEEE Transactions on Circuits and Systems for Video Technology* 24.3 (2013): 513-524.