VLSI Architecture Design of Guided Filter for 30fps Full-HD Video

Chieh-Chi Kao, Jui-Hsin Lai, *Member, IEEE,* and Shao-Yi Chien, *Member, IEEE*

***Abstract*—Filtering is widely used in image and video processing for various applications. Recently, the guided filter had been proposed and became one of the popular filtering methods. In this paper, to achieve the computation demand of guided filtering in Full-HD video, a double integral image architecture for guided filter ASIC design is proposed. In addition, a reformation of guided filter formula is proposed which can prevent the error resulted from truncation in fractional part and modify the regularization parameter ε on user's demand. The hardware architecture of guided image filter is then proposed and can be embedded in mobile devices to achieve real-time HD applications. To the best of our knowledge, this work is also the first ASIC design for guided image filter. With TSMC 90nm cell library, the design can operate at 100MHz and support for Full-HD (1920x1080) 30 fps with 92.9K gate counts and 3.2KB on-chip memory. Moreover, for the hardware efficiency, our architecture is also the best comparing to other previous works with bilateral filter.**

***Index Terms*—Guided filter, integral image, double integral image architecture**

I. Introduction

Filtering is an image processing technique widely adopted in computer vision, computer graphics, computational photography and etc. More specifically, filtering can be applied in many applications like noise reduction, texture editing, detail smoothing/enhancement, colorization, relighting tone mapping, haze/rain removal and joint upsampling. The most popular technique is the edge-preserving bilateral filter [1]. Liu et al. [2] applied bilateral filter to image noise reduction, Durand et al. [3] used bilateral filter on high dynamic range (HDR) images. Based on bilateral filter, joint bilateral filter is developed by Petschnigg et al. [4] in flash/no-flash denoising. In [5], Kopf et al. used joint bilateral filter for upsampling problems. For bilateral filter, real-time implementation [6] usually adopts histogram-based approximation due to its computation efficiency and memory concern. Guided filter has the non-approximation characteristic and offers an ideal option for real-time filter applications on HD videos.

Although bilateral filter has a good edge-preserving characteristic, it has been noticed that it may have artifacts in detail decomposition [7] and HDR compression [3]. Artifacts are resulted from those pixels which around the edge may have unstable Gaussian weighted sum. To overcome this problem,

C.-C. Kao, J.-H. Lai, and S.-Y. Chien are with Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan (R.O.C.). e-mail: sychien@cc.ee.ntu.edu.tw.

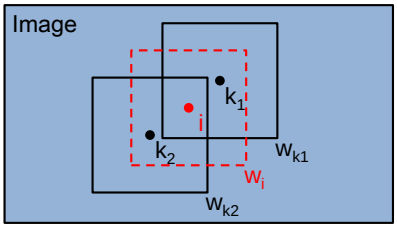


Fig. 1: The illustration of pixel i and related windows (w_i and w_k.)

He et al. [8] proposed guided filter, which can filter output by considering the content of the guiding image. Compared with bilateral filter, the guided filter can perform better at the pixels near edges. Moreover, the guided filter is a non-approximate linear-time algorithm, which is a very important strength for real-time applications.

Recently, many applications adopted guided filter as the filtering method. He et al. [9] used guided filter at the post-processing step to further improve the alpha mask. Ding et al. [10] used guided filter to filter out the image saliency under the guidance of the original image. In [11], because of the degradation in quality caused by fast approximation bilateral filter, guided filter was used for fast cost volume filtering. In [12], in order to suppress color noise while preserving color structures, guided image filtering was used to smooth the result of transferred colors. Zhang et al. [13] used guided filter to refine the crude transmission map. It has high computational load since it has to filter the transmission map at every frame. In [14], guided filter was applied to propagate the value from edge locations into the unknown region by using the blurry image as the guided image. Hosni et al. [15] applied guided filter to 3D spatial temporal space for computing temporally coherent disparity maps. For some applications mentioned above, a high throughput guided filter is needed. However, it is not applicable for CPU computation. Although GPU provides an alternative solution to high throughput guided filter, it has higher cost and power demand which is not suitable for mobile devices like digital camera or mobile phone. Therefore, a VLSI architecture design of guided filter is proposed in this paper.

The remainder of this paper is structured as follows. A brief review and implementation challenges of guided filter are described in Section II. Architecture design of guided filter is described in Section III. The proposed double integral image architecture is described in Section IV. Next, in Section V implementation results will be shown and discussed. Finally, conclusion is given in Section VI.

(a) Guidance I (b) Guided Filter (c) Crop

(d) Filter Input p (e) Joint Bilateral Filter (f) Crop

Fig. 2: Comparison of guided filter and joint bilateral filter by flash/no-flash denoising. In the result of joint bilateral filter, gradient reversal artifacts are noticeable near some edges, as shown in (f).

II. Guided Filter and Challenges for Implementation

Here we briefly review the main idea and equations of guided filter. Given a guidance image I and an input image p, a filtered output image q can be produced by guided filter. There is a key assumption of guided filter: The output image q is a local linear transformation from the guidance image I. That is, an output pixel q_i is a linear transform of guidance image I in a window w_k centered at pixel k:

$$q_i = a_k I_i + b_k, \forall i \in w_k \tag{1}$$

where a_k and b_k are linear constant coefficients in window w_k. The relation of pixel i and related windows (w_i and w_k) is shown in Fig. 1. Since $\nabla q = a\nabla I$, the output image q has edge only if the guidance image I has edge. In order to find the appropriate coefficients, He et al. [8] defined a cost function in window w_k as follows:

$$E(a_k, b_k) = \sum_{i \in w_k} ((a_k I_i + b_k - p_i)^2 + \varepsilon {a_k}^2) \tag{2}$$

The parameter ε here prevents a_k from becoming too large. The solution of a_k and b_k for above cost function are as follows:

$$a_k = \frac{\frac{1}{|w|}\sum_{i \in w_k} I_i p_i - \mu_k \bar{p_k}}{{\sigma_k}^2 + \varepsilon} \tag{3}$$

$$b_k = \bar{p_k} - a_k \mu_k. \tag{4}$$

$|w|$ is the number of pixel in window w_k, μ_k and ${\sigma_k}^2$ are the mean and variance of I in window w_k, and $\bar{p_k}$ is the mean of input image p in window w_k.

After the coefficients are well defined, the next step is to calculate all the local windows in the whole image. However, for any pixel i, it may have different output pixel value q_i calculated by different local windows. An average process is adopted for pixel i, which is defined as follows:

$$q_i = \frac{1}{|w|}\sum_{k:i \in w_k} (a_k I_i + b_k) = \bar{a_i} I_i + \bar{b_i} \tag{5}$$

where $\bar{a_i} = \frac{1}{|w|}\sum_{k \in w_i} a_k$ and $\bar{b_i} = \frac{1}{|w|}\sum_{k \in w_i} b_k$.

Although the modification in (5) makes $\nabla q \neq a\nabla I$, the average process can make the gradient of q smaller than guidance I near strong edges. Therefore, most of the abrupt changes in I are still conserved in this model, which means $\nabla q \approx \bar{a}\nabla I$.

The computations of sum of windows and averages in (3), (4) and (5) can be efficiently accelerated by integral image, which greatly reduces the computational complexity. Based on the data in [8], for a gray-scale optimized guided filter, processing a 1-Megapixel image on PC needs 80ms. However, it is not fast enough for 30 fps Full-HD (1920x1080) videos. Therefore, a hardware architecture is proposed to accelerate the computation of guided filter.

Compared with joint bilateral filter, guided filter has the edge-preserving smoothing property but does not suffer from the gradient reversal artifacts near edges, as the example of flash/no-flash denoising shown in Fig. 2. The result produced by guided filter in Fig. 2 uses the same parameters and fixed point calculation as the proposed system. The parameters for joint bilateral filter are $|S| = 31$ (window size), $\sigma_s = 31$, $\sigma_r = 0.02$. Note that the implementation of joint bilateral filter in Fig. 2 is not accelerated approximation like histogram-based joint bilateral filter [6] but accurate direct calculation. Furthermore, compared with N-bins histogram-based joint bilateral filter [6] from the view of hardware design, the proposed guided filter architecture has lower cost. The reason why N-bins histogram-based joint bilateral filter has higher cost is that it needs N sets of histogram calculation engine and memory, which greatly increases the implementation cost in ASIC design. On the contrary, although the proposed double integral image architecture has two stages for integral image calculation and larger number of bits in calculation, the proposed architecture does not need N sets of calculation engine in each stage. Therefore, the proposed architecture can achieve lower implementation cost.

The following are the design challenges of the proposed design. For a Full-HD frame, even adopt the integral image approach, the memory demand for guided filter is $(4+2) \times 1920 \times 1080 \times 4 = 49,766,400 \simeq 49.77$MBytes. (Here counts for integral images of I, I^2, p, Ip, a and b at single precision.) It is impractical to use such amount of on-chip memory in an ASIC design; therefore, memory reduction is an important issue in our design. Saving all the integral image data in off-chip memory, the system needs to ask for data while needed, which brings huge bandwidth. As a consequence, a double integral image architecture for guided filter is proposed to solve the problem of trade-off between on-chip memory size and data bandwidth.

The second challenge is trade-off between design complexity and the number of total gate count. Using IEEE 754 single precision data format makes the system easier to design;

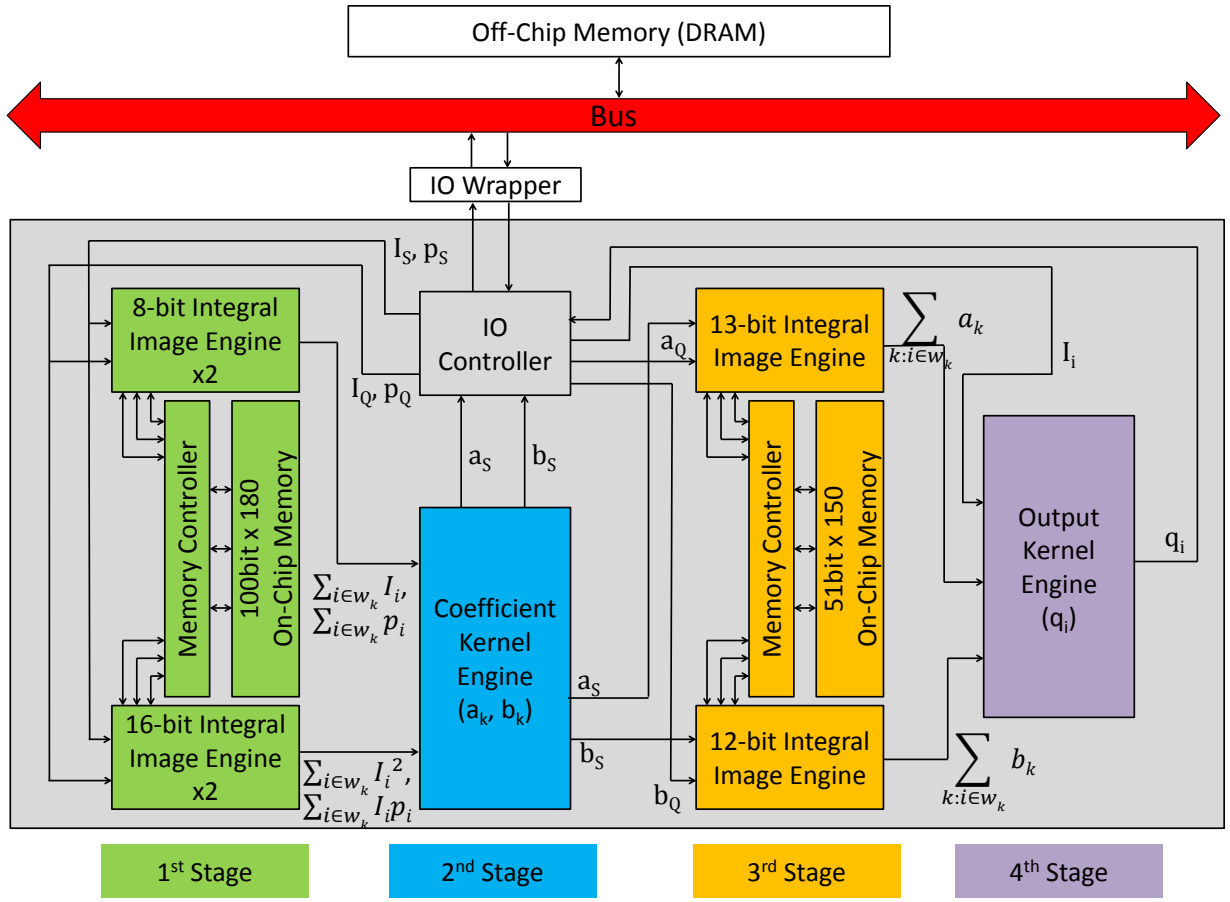


Fig. 3: The proposed architecture of guided filter with double integral image.

however, it takes more bits to store data, which means the size of on-chip memory increases sharply. Since there are few average operations (use division) in the whole algorithm, the fixed point arithmetic should take care of the trade-off between number of bits in fraction part and the accuracy of the final answer. A reformation of guided filter formula for hardware design is proposed, which can avoid the error resulted from the truncation of fractional part in fixed point calculations.

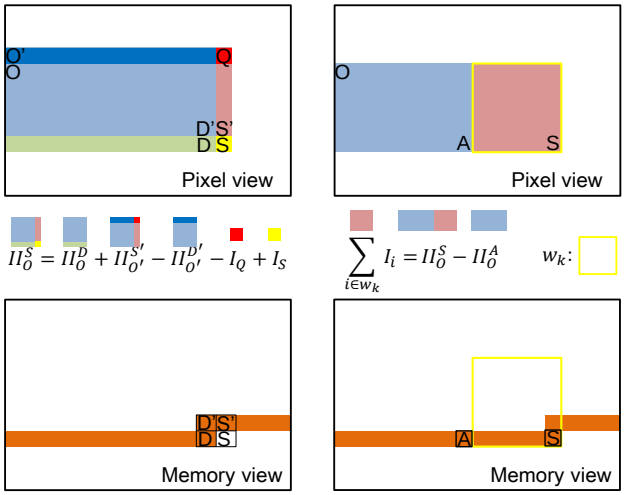
The last one is boundary handling issue. Tseng et al. [6] proposed a novel architecture for integral histogram, but the lower boundary (last few rows) for each frame is neglected. We extend the architecture proposed by Tseng to solve the boundary problem.

III. Design of Hardware Architecture

Fig. 3 shows the whole proposed architecture design of guided filter. The design consists of six Integral Image Engines, one Coefficient Kernel Engine, and one Output Kernel Engine. The Integral Image Engine is extended from the histogram calculation engine proposed in [6]. However, the novelties of this work are not at the Integral Image Engine, but the design of Coefficient (a_k, b_k) and Output (q_i) Kernel Engine, and the double integral image architecture for guided image filter. For the hardware implementation, the operating frequency is 100MHz, and the input video format is Full-HD (1920x1080), 30fps. Since the implementation in the paper is a prototype of the proposed guided filter architecture, an IO wrapper is needed between the ASIC design and bus, as shown in Fig. 3. The specification of the IO wrapper depends on the parameters of guided filter and the type of bus which are chosen by the user. The proposed scalable architecture can be also applied to different specifications of guided filter by modifying the hardware parameters. If a specific application of different parameter is needed, the reader can use the proposed architecture for guided filter to implement customized IP depending on their requirements.

The whole design adopts the stripe-based method proposed in [6], which can decompose a frame into several vertical stripes. Therefore, in the integration and extraction process of each stripe, as shown in Fig. 4, the required memory reduces from the width of frame to the width of stripe plus extended region.

For each cycle, IO controller gives data (I_S, I_Q, p_S and p_Q) of pixel Q and S (as shown in Fig. 4) to the 1st Stage(8-bit Integral Image Engine x 2, 16-bit Integral Image Engine x 2). Stage 1 calculates the sums of window (e.g. $\sum_{i\in w_k} I_i$) of I, p, $I \times p$ and I^2, and then passes them to the 2nd Stage (Coefficient Kernel Engine). After the coefficients are calculated, a_S and b_S will be sent to the 3rd Stage (12-bit Integral Image Engine and 13-bit Integral Image Engine). In the meantime, Coefficient



(a) Integration Process (b) Extraction Process

Fig. 4: Concept of the integral image engine. It consists of integration and extraction part.

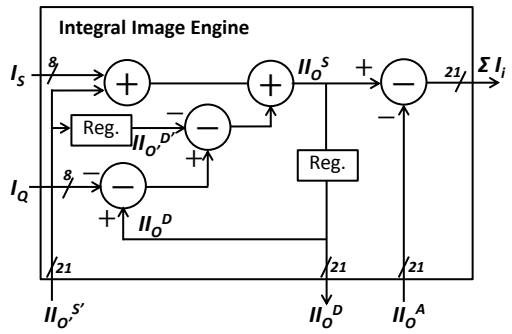


Fig. 5: The hardware design of 8-bit Integral Image Engine. It calculates the integral image of guidance image (I).

Kernel Engine gives a_S and b_S to IO controller and writes out to the buffer area at off-chip memory. In this way, when the 3rd Stage needs a_Q and b_Q for integral image integration, the buffer at off-chip memory can provide them. Finally, the 4th Stage (Output Kernel Engine) receives the sums of window of a and b, and I_i from IO controller, the final output q_i can be calculated.

Detailed descriptions of each hardware design are shown in the following part. In the figures of each hardware design, the adders and multipliers are implemented directly in Verilog without pipeline or parallel processing. However, for the dividers, we use the pipelined dividers in Synopsys DesignWare Library, and the number of stage in each divider is also annotated in the figures (Fig. 7, 8, 12) of hardware design.

A. Integral Image Engine

Concept of the integral image engine is shown as shown in Fig. 4. This engine uses pixel values (e.g. I_S, I_Q) as input and calculates the sum of window of it ($\sum_{i \in w_k} I_i$). As the integral histogram architecture proposed by Tseng et al. [6], the usage of on-chip memory can be effectively reduced by

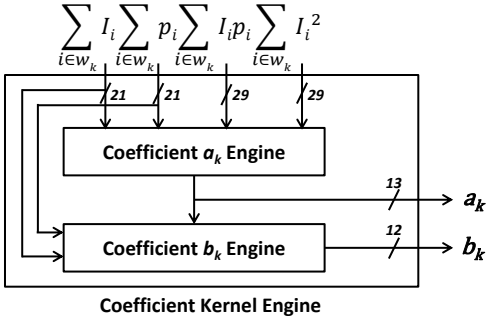


Fig. 6: The hardware design of Coefficient Kernel Engine.

runtime updating method (RUM), stripe-based method (SBM) and sliding origin method (SOM). We modify the architecture from the proposed method in [6] to calculate integral image. By the stripe-based method, the amount of processing data at each time is reduced to a stripe from a whole frame. Fig. 4a shows the concept of integration process. With RUM and SOM, the amount of on-chip memory for integral image is reduced to only a row (the orange row in Fig. 4) from a whole stripe. In the integration process, it takes three integral images (II) and two pixel data (I). On the other hand, the extraction process takes two integral images (II) as input data. As shown in Fig. 5, the extraction process (calculation of $\sum_{i \in w_k} I_i$) and the integration process (calculation of $II_O{}^S$) can be implemented together. By lifetime analysis and buffer registers, all the needed input data can be reduced to two integral images ($II_{O'}{}^{S'}$ and $II_O{}^A$) and two pixel data (I_S, I_Q) in each cycle. Fig. 5 shows the 8-bit Integral Image Engine (IIE) for I in the first stage. The other 5 IIEs in Fig. 3 share the same architecture with Fig. 5 but with different number of bits in the datapaths. For each IIE, there are two IIs to be read from the on-chip memory and one II to be written to on-chip memory in every operating cycle. Therefore, 2 memory controllers are used to handle the data passing between IIEs and on-chip memory. Because the word number demanded of on-chip memory is low, register files are used rather than SRAM. Furthermore, since two IIs are read from the memory at each time, two memory banks are used for the on-chip memory. For example, in the on-chip memory of the first stage (100bit×180), 2 two-port register files (each of 90 words, 100 bit) are used.

B. Coefficient Kernel Engine

In this section, the reformation of guided filter formula for hardware design is proposed. Using the reformed formula, the architecture of a_k and b_k engine avoids the error propagation in decimal computation brought by truncation in each division operation. The proposed architecture of Coefficient Kernel Engine is shown in Fig. 6.

1) Reformation of a_k Formula: In this section, a reformation of guided filter formula is proposed to avoid the error resulted from truncation of fractional part during the fixed-point calculations. In (3) and (4), there are three division operations which lead to the emergence of fractional part in the calculation. Since we have to use fixed-point arithmetic

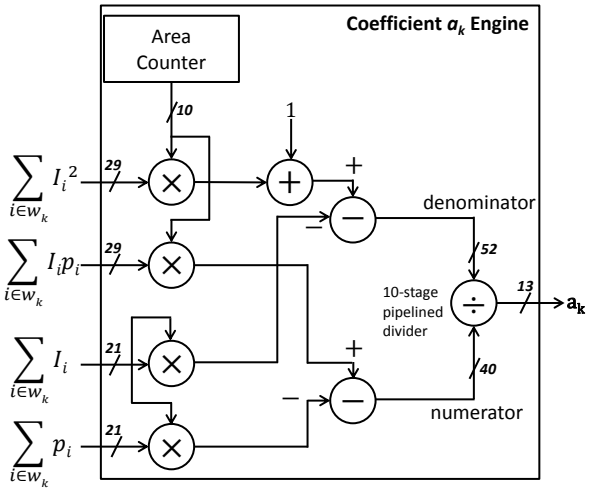


Fig. 7: The hardware design of a_k Kernel.

to reduce the gate counts in the ASIC design, the number of bits in fractional part in every calculation greatly affects the accuracy of the final output. Therefore, (3) and (4) are reformed for hardware architecture, which makes the intermediate calculation operates in integer domain without any precision loss during truncation process. The reformation of a_k is as follows:

$$\begin{aligned} a_k &= \frac{\frac{1}{|w|}\sum_{i\in w_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k{}^2 + \varepsilon} \\ &= \frac{\frac{1}{|w|}\sum_{i\in w_k} I_i p_i - \mu_k \bar{p}_k}{(\frac{1}{|w|}\sum_{i\in w_k} I_i{}^2 - \mu_k{}^2) + \varepsilon} \\ &= \frac{|w|\sum_{i\in w_k} I_i p_i - |w|^2 \mu_k \bar{p}_k}{|w|\sum_{i\in w_k} I_i{}^2 - |w|^2 \mu_k{}^2 + |w|^2 \varepsilon} \\ &= \frac{|w|\sum_{i\in w_k} I_i p_i - \sum_{i\in w_k} I_i \sum_{i\in w_k} p_i}{|w|\sum_{i\in w_k} I_i{}^2 - (\sum_{i\in w_k} I_i)^2 + |w|^2 \varepsilon} \end{aligned} \tag{6}$$

where $\sum_{i\in w_k} I_i p_i$, $\sum_{i\in w_k} I_i{}^2$, $\sum_{i\in w_k} I_i$ and $\sum_{i\in w_k} p_i$ are sums of $I \times p$, $I \times I$, I, p in window w_k. In this reformation, all the multiplications, additions and subtractions are computed in integer domain, which means no truncation loss. The only fraction part resulted from the final division, and the choice of number of bits in fraction part will be discussed in Section III-B3. Compared with the brute force calculation of original formula (3), the times of division is reduced from 4 to 1. Three divisions in average operations are replaced by two multiplications, which effectively reduces the circuit complexity and calculation error.

As a_k shown above, b_k can be also calculated in integer domain for truncation error prevention. The reformed formula is as follows:

$$\begin{aligned} b_k &= \bar{p}_k - a_k \mu_k \\ &= \frac{\sum_{i\in w_k} p_i - a_k \sum_{i\in w_k} I_i}{|w|} \end{aligned} \tag{7}$$

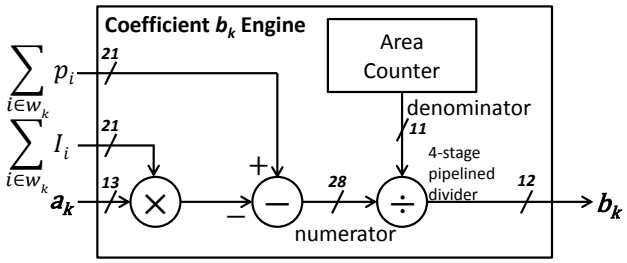


Fig. 8: The hardware design of b_k Kernel.

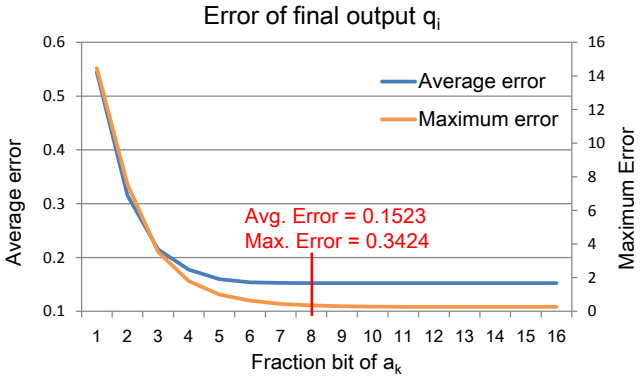


Fig. 9: The effect in average output error and maximum output error by number of bits in fraction part of a_k, given 1-bit for sign bit and 4-bit for integer part. The red line indicates the final choice in our design, 8-bit of fraction part in a.

where the $\sum_{i\in w_k} p_i$ and $\sum_{i\in w_k} I_i$ are sums of p and I in window w_k. Compared with the brute force calculation of original formula (4), the times of division is reduced from 2 to 1.

2) Architecture of a_k and b_k Kernel: The proposed architecture for coefficient a_k is shown in Fig. 7. In the reformed formula (6), there are two multiplications needing the area ($|w|$) of window (w_k) as input. For handling the boundary conditions, an area counter is needed for different windows, since the window area changes at stripe boundary.

In the calculation of denominator, 1 is added as the regularization parameter ε (i.e. set $|w|^2\varepsilon = 1$). Since the formula is reformed to integer domain for hardware design, the 1 added here is actually equal to a very small ε in the original formula (3). (That is $1/|w|^2$, which equals $1/961^2 \simeq 10^{-6}$ at most windows in our design.) The added number can be modified to add any value depends on the user's demand (at precision $1/|w|^2$). The proposed a_k kernel is a flexible design which can be adapted different applications.

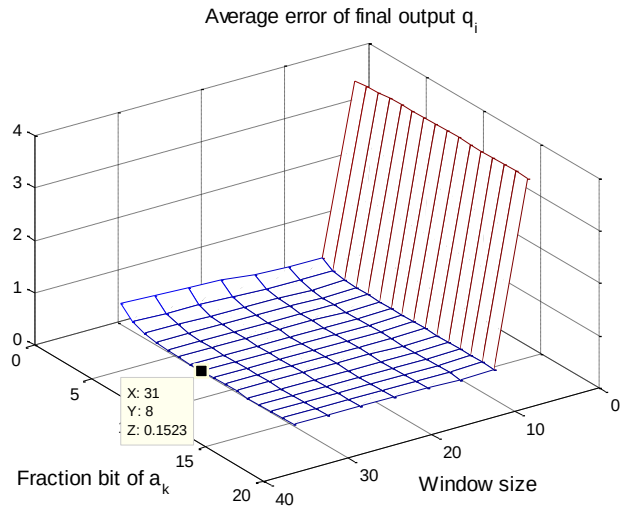
The proposed architecture for coefficient b_k is shown in Fig. 8. Because in the reformed formula of b_k still needs $|w|$ as denominator, there is an area counter in the b_k Kernel. Moreover, the number of bits for the area is extended to 11 due to 1-bit sign extension.

3) Word Length Analysis: In this part, we analyze the effect in precision resulted from the number of bits in a_k and b_k. First, note that in (5), b_k is directly used after an average process. After the average process, the number of bits of

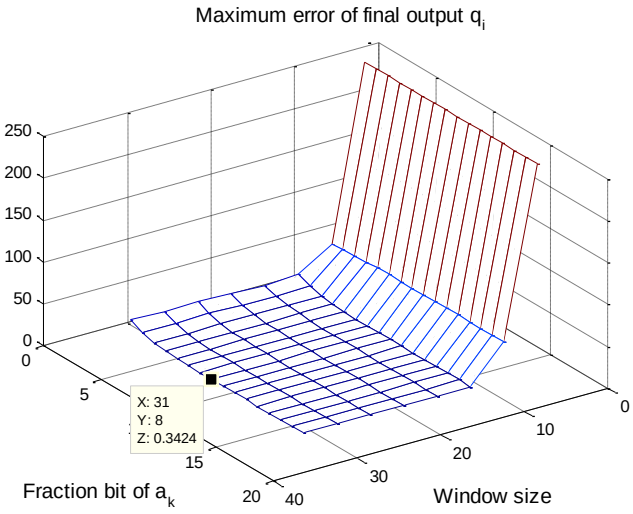
fraction part in b_k has little impact on the final output answer. We have tried to change the number of bits of fraction part in b_k from 1 to 10 bits, but the maximum error and average error of output q_i remain the same. As a result, we fix the number of bits in fraction part for b_k to 1 in our implementation. On the contrary, a_k is multiplied by I_i after an average process in (5), and also multiplied by the summation of a window in I in (7), which makes the bit number of fractional part greatly influence the final precision. The plot of average and maximum output error versus the number of bits in fraction part in a_k are shown in Fig. 9. The final choice of number of bits in fraction part for a_k is 8-bit, since the precision cannot be further greatly improved by increasing number of bits in fraction part. As shown in Fig. 9, the chosen point has average error at 0.1523 and maximum error at 0.3424.

A further analysis of the effect on the output error (q_i) with respect to window size and the number of bits in fraction part of a_k is shown in Fig. 10. When window size is small, both average and maximum error trend down with window size. Moreover, there are sharp edges both in average and maximum error at window size between 3 to 7. The error is resulted from the fixed number of bits in integer part of a_k. Because there are only 4-bit in the integer part, the limitation of maximum and minimum value of a_k are 16 and -16. In Fig. 11a, the true maximum and minimum value of a_k are far beyond the limits at window size equals 3. In the first row of (6), because smaller window size results in smaller window area ($|w|$); the smaller the window size, the larger the a_k. As shown in Fig. 11a, the needed number of bits in integer part of a_k decreases with window size.

Another fact may be noticed from Fig. 10a is the average error of output q_i increases with window size. Because the increase of average error is inconspicuous in Fig. 10a, a closer view is shown in Fig. 11b. In (7), coefficient a_k is multiplied by a summation of a window in I. The larger the window size, the larger the summation of I. Therefore, for larger value of the summation, truncation error in a_k brings about larger error in b_k and final output q_i. All in all, there are two aspects which can result in error at the output: the overflow and truncation problem in a_k. For smaller window size, the overflow problem dominates the output error. In Fig. 11a, as a_k does not overflow for window size larger than 11, the error is dominated by the truncation error. As shown in Fig. 11b, the average error increases with window size due to the truncation error in a_k. For different window sizes, the selection of number of bits in integer and fraction part in a_k plays an important role in the proposed architecture. The user should choose the suitable number of bits in a_k depending on the specification of desired design.



(a) Average Output Error



(b) Maximum Output Error

Fig. 10: The effect in average and maximum output error of window size and number of bits in fraction part of a_k, given 1-bit for sign bit and 4-bit for integer part. The black point with annotation in (a) and (b) indicates the final choice in our design.

C. Output Kernel Engine

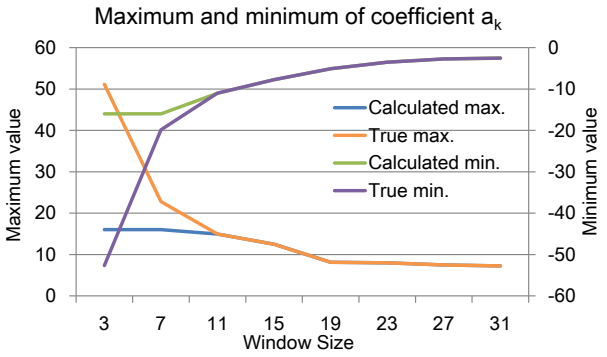
In this section, architecture for output q_i is proposed. For the hardware design, the formula in (5) is reformed to integer domain as in Section III-B. The formula is reformed as follows:

$$\begin{aligned} q_i &= \bar{a}_i I_i + \bar{b}_i \\ &= \frac{I_i \sum_{k:i\in w_k} a_k + \sum_{k:i\in w_k} b_k}{|w|} \end{aligned} \tag{8}$$

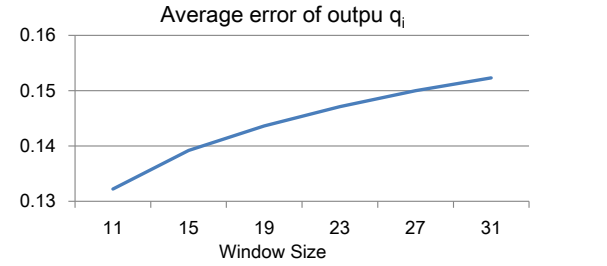
where $\sum_{k:i\in w_k} a_k$ and $\sum_{k:i\in w_k} b_k$ are sums of window of coefficients a_k and b_k. The design is shown in Fig. 12. The architecture is similar with b_k Kernel, however, we did not fold these two architectures together because each of these two Kernels already has high utilization rate. More details about utilization rates will be discussed in Section IV-C.

IV. Double Integral Image Architecture

As mentioned in Section II, although guided filter can be implemented by integral image method, it is impossible



(a) Maximum and Minimum Curve of a_k



(b) Average Output Error

Fig. 11: (a) The true and calculated maximum and minimum value of a_k with respect to window size. Because the number of bits in integer part of a_k is set to 4-bit, the maximum and minimum value of calculated a_k are saturated at 16 and -16 for window size smaller than 11. (b) Average output error with respect to window size. (Number of bits in integer and fraction part of a_k are fixed at 4 and 8.)

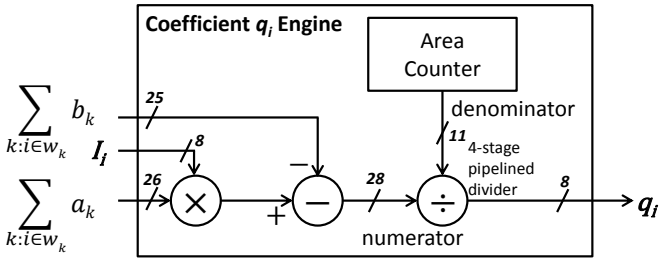


Fig. 12: The hardware design of q_i Kernel.

to store the intermediate coefficients a_k and b_k in on-chip memory due to its large memory demand. Therefore, a double integral image architecture is proposed, which is based on the integral histogram architecture proposed by Tseng et al. [6]. The data flow of double integral image architecture can be separated into two parts. In the following, the whole dataflow will be introduced.

A. Dataflow of Double Integral Image Architecture

The first part is illustrated in Fig. 13. The first and second stages here are referred to the same stages in Fig. 3. The input data is only a stripe rather than a full frame due to the stripe-based method. Given stripe width w_s, window size $|S|$ ($|w| = |S|^2$), frame height H_f, the size of each input stripe is $(w_s + 2(|S|-1)) \times H_f$. Choosing the number of pixel in a window ($|w|$) to be a power of 2 can remove the need of dividers in the following calculations. However, since it cannot produce a symmetric ($(2k+1)-by-(2k+1), k \in N$) window, this option is not taken into consideration. Output sums of window at the first stage represent the summations of input image (both guidance image and original image) in a specific window. In Fig. 13, the blue area in the first stage means it outputs the sums of window of corresponding window whose center is in the blue area at the input stripe. Take the sum of window of I as an example, the output at location (x_1, y_1) in the first stage is defined as:

$$SW_I(x_1, y_1) = \sum_{i \in w_{(x_0,y_0)}} I_i, \tag{9}$$

where $x_0 = x_1 - (\frac{|S|-1}{2})$, $y_0 = y_1 - (\frac{|S|-1}{2})$, $x_1 \geq |S|-1$, $y_1 \geq (\frac{|S|-1}{2})$, and $w_{(x_0,y_0)}$ is the window centered at (x_0, y_0) with window size $|S|$ in the input stripe. The relationship between input image and the first stage can be explained by an easy example. As shown in Fig. 15, the sum of window centered at pixel 11 (red cross), can be derived at pixel 22 (red point) in the next stage. In the same way, as shown in Fig. 13, the red and orange crosses at the first stage are the summations of the red and orange squares in the input stripe. In our architecture, each stage outputs the data of one point in a cycle, which means the whole system is well pipelined. Outputs at the first stage can be used to calculate the coefficients at the second stage, and the correspondence between the first and second stages is $x_1 = x_2$, $y_1 = y_2$.

In order to reduce the on-chip memory for the storage of a and b, at the second stage, the outputs a and b are written to a small buffer at the off-chip memory. The required off-chip memory size is $|S| \times (w_s + (|S|-1)) \times (w_a + w_b)$, where w_a is the bit number of a and w_b is the bit number of b. Moreover, the oldest data in the buffer will be sent to the third stage for the integral image calculation and then overwritten by the new a and b produced by the second stage.

The second part of the dataflow is illustrated in Fig. 14. In the second stage, the available data is the blue area. However, we are going to calculate the sums of window of a and b, which means that the valid positions of window center shrink to the green area from blue area. In the third stage, the sums of window of a and b are calculated, which follows the equation below (take a for example):

$$SW_a(x_3, y_3) = \sum_{k \in w_{(x_2,y_2)}} a_k, \tag{10}$$

where $x_2 = x_3 - (\frac{|S|-1}{2})$, $y_2 = y_3 - (\frac{|S|-1}{2})$, $x_3 \geq 2(|S|-1)$, $y_3 \geq |S|-1$, and $w_{(x_2,y_2)}$ is the window centered at (x_2, y_2) with window size $|S|$ in the second stage. As shown in Fig. 14, the brown cross at the third stage is the summation of the brown square in the input second stage. Given the sums of window of a and b, the output q can be calculated at the fourth stage. Now we are going to map the location of output q at the fourth stage to the location of input stripe. It can be

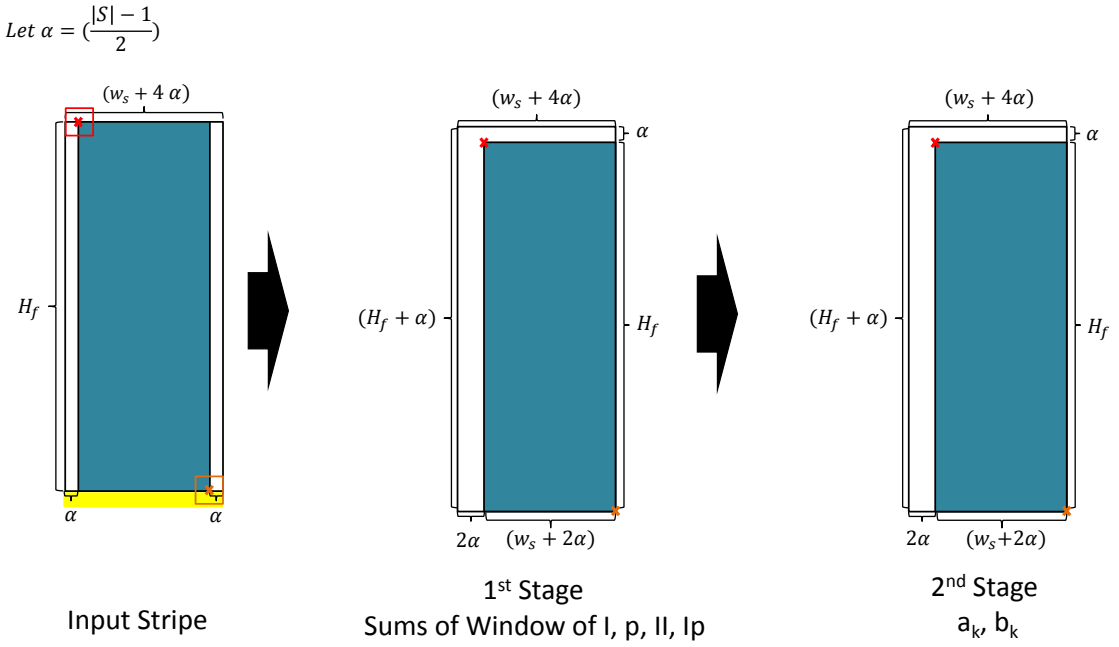


Fig. 13: The first part of dataflow of the proposed double integral image architecture.

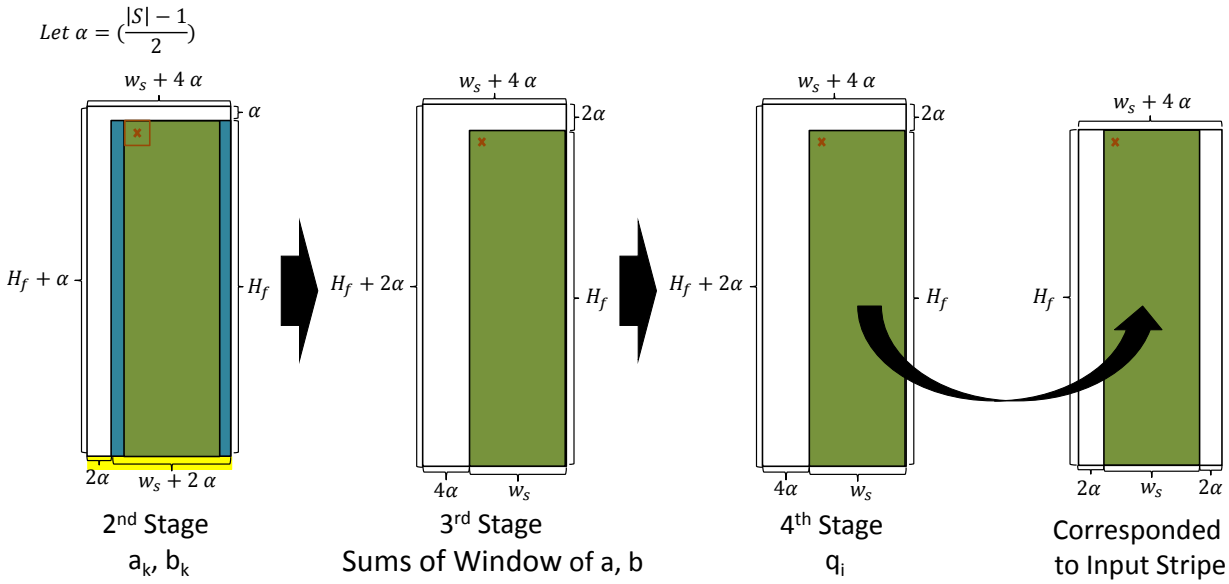


Fig. 14: The second part of dataflow of the proposed double integral image architecture. The right most part shows the relationship of output q_i and corresponded pixel location in the input stripe.

derived as follows:

$$\begin{aligned}(x_0, y_0) =& (x_1 - (\frac{|S|-1}{2}), y_1 - (\frac{|S|-1}{2})) \\ =& (x_2 - (\frac{|S|-1}{2}), y_2 - (\frac{|S|-1}{2})) \\ =& (x_3 - (\frac{|S|-1}{2}) - (\frac{|S|-1}{2}), y_3 - (\frac{|S|-1}{2}) - (\frac{|S|-1}{2})) \\ =& (x_4 - (|S|-1), y_4 - (|S|-1)),\end{aligned} \tag{11}$$

where $w_s + 2(|S|-1) > x_4 \geq 2(|S|-1)$, $H_f + |S| - 1 > y_4 \geq |S| - 1$ or $w_s + |S| - 1 > x_0 \geq |S| - 1$, $H_f > y_0 \geq 0$. This correspondence is also illustrated in the right most part of Fig. 14.

Table I shows the comparison of used resource between the proposed double integral image architecture and the other implementation methods. The direct method only calculates the coefficient and store all the integral images at off-chip memory, which results in high bandwidth and off-chip memory demand. The single integral image architecture uses the

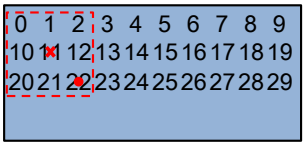


Fig. 15: The illustration of sum of window process.

TABLE I: RESOURCE USED BY DIFFERENT ARCHITECTURE OF GUIDED FILTER PER FRAME

	Direct Method	Single Integral Image	Proposed Double Integral Image
Off-chip Memory	51.58MB	6.48MB	14.53KB
Bandwidth	2.13Gb	327.11Mb	262.31Mb

method similar to [6], and stores all intermediate coefficients (a, b) at the off-chip memory. Compared with single integral image architecture, the proposed double integral image architecture can update intermediate coefficients at the off-chip memory and uses the coefficients directly once they are calculated, which reduces 99.776% of off-chip memory and 19.8% of bandwidth. Compared with the direct method, the proposed architecture reduces 99.972% of off-chip memory and 87.683% of bandwidth.

B. Boundary Handling

In hardware design, boundary handling is always an annoying but critical issue. Here an extended null stripe method (ENS) is proposed. At every integral image step, an extended null stripe with height $\frac{|S|-1}{2}$ is added at the end of the stripe. Every data point in the null stripe is set to 0. If there is no extended null stripe added at input stripe and Stage 2, the integral images at the next stages (Stage 1 and Stage 3) would be lack of the answer of last $\frac{|S|-1}{2}$ rows. Take Stage 1 for example, the function of extended null stripe can be explained by the relation of (x_0, y_0) and (x_1, y_1). The integral image centered at (x_0, y_0) generates at (x_1, y_1) in Stage 1. Because $y_1 = y_0 + \frac{|S|-1}{2}$, if there is no extended null stripe added at input stripe, the calculation stops at the last row of input stripe ($y_0 = H_f$), which means the integral images in Stage 1 are only produced to row $H_f - \frac{|S|-1}{2}$. An example of numerical expression is as follows: Given the height of the stripe (H_f) and the size of window ($|S|$) are 1080 and 31, if there is no extended null stripe added at input stripe, the integral images we can get in Stage 1 would become those centered only from row 0 to row 1064 (1079-(31-1)/2 = 1064) rather than 0 to 1079. The yellow stripes at the input stripe in Fig. 13 and the second stage in Fig. 14 are all extended null stripes.

C. Hardware Performance and Parameter Setting

In the section, hardware performance and parameter setting of the proposed architecture are discussed. The hardware utilization rate in each stage is shown in Table II. The utilization

TABLE II: HARDWARE UTILIZATION RATE IN EACH STAGE

Stage 1	Stage 2	Stage 3	Stage 4	Average
97.56%	80.19%	81.30%	64.15%	80.80%

TABLE III: SPECIFICATION OF THE PROPOSED GUIDED FILTER CHIP

Technology	TSMC 90nm CMOS Mixed Mode Signal MS General Purpose LowK Cu 1P9M
Frame Size	1920×1080
Frame Rate	30
Filter Window Size	31×31
Stripe Width	120
Operating Frequency	100MHz
Chip Size	1.46×1.43 mm^2
Core Size	0.92×0.89 mm^2
Power Consumption	22.522 mW
Gate Counts	92,895
On-Chip Memory (Byte)	3,206

rate can be regarded as the ratio of area of color region to the whole area at each stage in Fig. 13 and Fig. 14. The average utilization rate is 80.80%. Although the architecture of b_k Kernel and the Output Kernel are quite similar, their utilization rates are 80.19% and 64.15%, which means there is no room for folding architecture. However, in case of working with different clock frequencies on the different stages, folding techniques could be applied by adding buffer registers.

The stripe width w_s is an important parameter because it affects computation cycle, on-chip memory and off-chip bandwidth. Increasing w_s can reduce computation cycle and off-chip bandwidth; however, the on-chip memory increases, which means the hardware cost increases. For parameter setting, the detailed discussion of computation cycle, on-chip memory and off-chip bandwidth of the proposed double integral image architecture are shown as follows. As shown in Fig. 13 and Fig. 14, the needed computation cycle for each stripe is:

$$(w_s + 2(|S|-1)) \times (H_f + |S| - 1). \tag{12}$$

For a whole frame, the total number of computation cycle becomes:

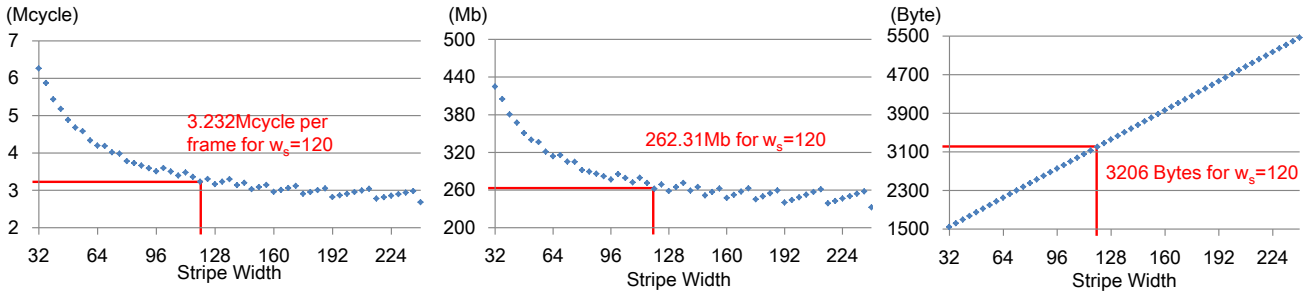
$$\lceil W_f / w_s \rceil \times (w_s + 2(|S|-1)) \times (H_f + |S| - 1), \tag{13}$$

where W_f and H_f are the width and height of frame. Note that, there are 2 bubble cycles for resetting the registers in the integral image engines in each row during the computation. The plot of computation cycle per frame versus stripe width is shown in Fig. 16a.

Second, the needed bandwidth is as follows:

$$\begin{aligned}\{&[(w_s + 2(|S|-1)) \times H_f \times 32] + [w_s \times H_f \times 16] + \\ &[(w_s + |S| - 1) \times H_f \times 2 \times (w_a + w_b)]\} \times \lceil W_f / w_s \rceil,\end{aligned} \tag{14}$$

where $w_a = 16$ and $w_b = 9$ are the numbers of bits in coefficients a and b, $[(w_s + 2(|S|-1)) \times H_f \times 32]$ is the bandwidth of input pixel data (I_S, I_Q, p_S, p_Q) at the first stage, $[w_s \times H_f \times 16]$ is the bandwidth of the fourth stage including input data I_i and output answer q_i, $[(w_s + |S| - 1) \times H_f \times 2 \times (w_a + w_b)]$ is the



(a) Computation Cycle per Frame (b) Off-chip Bandwidth (c) On-chip Memory

Fig. 16: Hardware performance with different values of stripe width.

bandwidth of output a_S, b_S at the second stage and input a_Q, b_Q at the third stage. The plot of off-chip bandwidth versus stripe width is shown as Fig. 16b.

Last, the required on-chip memory is as follows:

$$[(w_s+2(|S|-1))]\times[2\times(8+w_{m1})+2\times(16+w_{m1})] \\ +[(w_s+|S|-1)]\times[(w_a+w_{m3})+(w_b+w_{m3})], \quad (15)$$

where w_{m1} is equal to $\lceil log_2[|S|(w_s+2(|S|-1))]\rceil$ and w_{m3} is equal to $\lceil log_2[|S|(w_s+|S|-1)]\rceil$. In our design, both w_{m1} and w_{m3} are equal to 13. As shown in Fig. 16c, the size of on-chip memory increases linearly with the stripe size w_s.

With the constraint of operation frequency (100MHz), take the above three aspects into consideration, w_s is set to 120 in our implementation and the needed computation cycle for one stripe becomes 202,020. For the whole frame, it takes 3,232,320 cycles for computation, as shown in Fig. 16a. For 30 frames of Full-HD video, it takes 96,969,600 cycles for computation. The processing time for 30 frames equals the total number of computation cycle (96,969,600 cycles) multiplied by unit computation cycle time (10ns), which means that it takes 0.969696 sec for calculation and meets our specification of operation frequency.

V. Implementation Result

In this section, our implementation result is presented and compared with other previous works. Our specification is operating at 100MHz frequency and 30fps Full-HD (1920x1080). The proposed guided filter architecture is implemented with TSMC 90 nm 1P9M technology. The implementation result and chip layout are shown in Table III and Fig. 17. The hardware design spends 92.9K equivalent NAND gates and 3.2KB on-chip memory to achieve the throughput of 30fps Full-HD 1080p at clock rate 100MHz. The power consumption is the post-layout simulation result by Synopsys IC Compiler.

There are few implementations for guided filter, and the comparison is in Table IV. Compared with previous GPU and CPU implementations, the proposed architecture offers an efficient solution of relatively low hardware cost with high throughput. Although GPU has higher throughput, it needs high power supply. The proposed architecture can achieve higher throughput by adding more processing elements since each stripe can be processed in parallel. For the same throughput, hardware cost and power consumption of the proposed ASIC design are much lower than GPU. Moreover, the proposed design can be embedded into devices like digital camera which has no GPU on it. Another convenient platform for real-time filter implementation is DSP platform. Although we did not implement the guided filter algorithm on DSP platform, we found previous works to justify the value of guided filter ASIC design. Gangwal et al. [17] implemented joint-bilateral filters on TriMedia TM3270. According to [17], it is the first reporting of a real-time implementation of joint-bilateral filters on an embedded DSP. Operating at 350MHz, it can achieve maximum throughput at 20.7 MPixel/sec (720x576@50Hz). However, according to [18], the estimated power consumption for the design in [17] at 350MHz is 281.44mW (350(MHz) x 0.935(mW/MHz) x 86%(cycle budget)) and the chip area is 8.08 mm^2. Moreover, the realization of the TM3270 is in a low power CMOS process technology, with a 90 nm feature size, so it is fair to compare the power consumption and area with the example design of the proposed architecture, which is also implemented with 90 nm technology. Compared with the work in [17], the proposed design can achieve 2.99X throughput with reducing 92.0% of power consumption and 74.2% of chip area. Although it may not be totally fair to compare the implementations of different filter on different platforms, the above mentioned comparison still provides a supporting reason for the necessity of ASIC filter design in hand-held devices where the power consumption is the major concern.

The proposed architecture for guided filter is also compared with previous ASIC implementations for other edge preserving filters, as shown in Table V. Although the presented comparison in Table V consists of different kinds of algorithm, all the goals of these filters are the same, which is filtering in image processing. Compared with of other state-of-the-art ASIC filter designs, the proposed guided filter architecture greatly reduces the usage not only in equivalent gate counts but also in on-chip memory. For the same specification, compared with the joint bilateral filter design in [6], the proposed architecture uses 35.1% of gate counts and 13.9% of on-chip memory to achieve the better performance since guided filter can avoid the gradient reversal artifacts that bilateral filter may have in detail enhancement. As shown in Fig. 18, the designed architecture is validated by comparing the filtered result with the result generated by software in double precision.

TABLE IV: COMPARISON OF DIFFERENT GUIDED FILTER IMPLEMENTATIONS

	He [8]		Rhemann [11]	Proposed
Implementation	CPU Core2Duo 2.0GHz		GPU GeForce GTX480	ASIC
Image Size (Pixel)	1.0M		1.0M	2.07M
Image Type	Gray-scale	RGB	RGB	Gray-scale
Throughput (Pixel/sec)	12.5M	3.33M	200M	62M

TABLE V: COMPARISON WITH OTHER ASIC FILTER IMPLEMENTATIONS

	Han [16]	Tseng [6]	Proposed
Filter Type	Bilateral Filter	Joint Bilateral Filter	Guided Filter
Technology	TSMC 0.18um	UMC 90nm	TSMC 90nm
Frame Size	320x240	1920x1080	1920x1080
Frame Rate	144	30	30
Filter Window Size	11x11	31x31	31x31
Operating Frequency	60 MHz	100 MHz	100 MHz
Throughput (Pixel/s)	11M	62M	62 M
Gate Counts	355K	276.2K	92.9K
On-Chip Memory (B)	7.8K	23K	3.2K

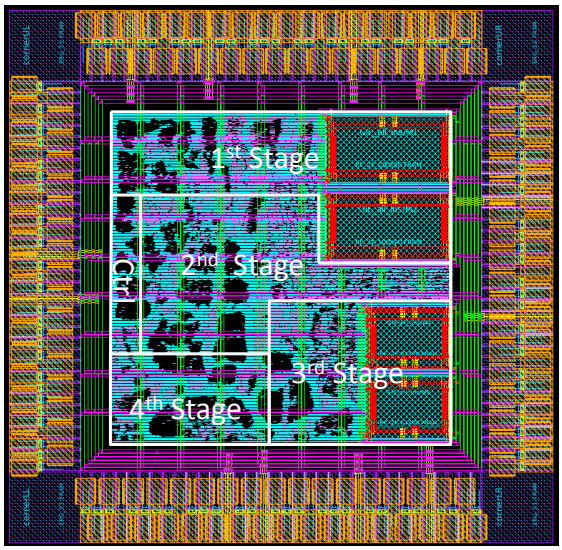
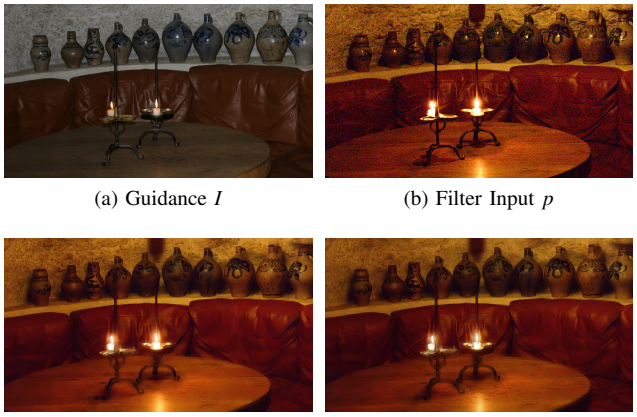


Fig. 17: Chip layout.



(a) Guidance I (b) Filter Input p

(c) Filtered Result of Software in Double Precision (d) Filtered Result of the Proposed Architecture

Fig. 18: Comparison of the filtered result of the proposed architecture and the filtered result of software in double precision by flash/no-flash denoising.

VI. Conclusion

This paper has proposed an VLSI architecture design for guided filter. To the best of our knowledge, this work is also the first work to implement guided filter in hardware. With the proposed architecture, an example design has throughput of 30fps Full-HD (1920x1080) with 92.9K NAND gates and 3.2KB on-chip memory is implemented. Compared with other previous filter ASIC designs, the proposed guided filter design using TSMC 90nm technology reduces not only gate counts but also on-chip memory, and still has high throughput. Moreover, recent literature showed that guided filter performs well in many applications, which means the guided filter may perform better than other filters. In other words, using guided filter can save hardware cost without the loss in quality. In this paper, the coefficients are only calculated for gray-scale guidance image (I). By modifying the coefficient kernel in the second stage, the proposed architecture can be able to handle color image. Apply the proposed architecture to RGB guided filter is the primary future work of this paper.

References

[1] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proc. IEEE 6th Int. Conf. Computer Vision (ICCV)*, Bombay, India, 1998, pp. 839–846.

[2] C. Liu, W. Freeman, R. Szeliski, and S. B. Kang, "Noise estimation from a single image," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, vol. 1, New York, NY, 2006, pp. 901–908.

[3] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 257–266, Jul. 2002.

[4] G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe, and K. Toyama, "Digital photography with flash and no-flash image pairs," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 664–672, Aug. 2004.

[5] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, "Joint bilateral upsampling," *ACM Trans. Graph.*, vol. 26, no. 3, article 96, Jul. 2007.

[6] Y.-C. Tseng, P.-H. Hsu, and T.-S. Chang, "A 124 Mpixels/s VLSI design for histogram-based joint bilateral filtering," *IEEE Trans. Image Process.*, vol. 20, no. 11, pp. 3231–3241, Nov. 2011.

[7] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski, "Edge-preserving decompositions for multi-scale tone and detail manipulation," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 67:1–67:10, Aug. 2008.

[8] K. He, J. Sun, and X. Tang, "Guided image filtering," in *Proc. 11th European Conf. Computer vision (ECCV): Part I*, Crete, Greece, 2010, pp. 1–14.

[9] K. He, C. Rhemann, C. Rother, X. Tang, and J. Sun, "A global sampling method for alpha matting," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, Colorado Springs, CO, 2011, pp. 2049–2056.

[10] Y. Ding, J. Xiao, and J. Yu, "Importance filtering for image retargeting," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, Colorado Springs, CO, 2011, pp. 89–96.

[11] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, Colorado Springs, CO, 2011, pp. 3017–3024.

[12] A. Y.-S. Chia, S. Zhuo, R. K. Gupta, Y.-W. Tai, S.-Y. Cho, P. Tan, and S. Lin, "Semantic colorization with internet images," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 156:1–156:8, Dec. 2011.

[13] J. Zhang, L. Li, Y. Zhang, G. Yang, X. Cao, and J. Sun, "Video dehazing with spatial and temporal coherence," *Vis. Comput.*, vol. 27, pp. 749–757, Jun. 2011.

[14] Y. Cao, S. Fang, and F. Wang, "Single image multi-focusing based on local blur estimation," in *Proc. Sixth Int. Conf. Image and Graphics (ICIG)*, Hefei, China, 2011, pp. 168–175.

[15] A. Hosni, C. Rhemann, M. Bleyer, and M. Gelautz, "Temporally consistent disparity and optical flow via efficient spatio-temporal filtering," in *Advances in Image and Video Technology, LNCS*, vol. 7087, Y.-S. Ho, Ed. Berlin: Springer, 2012, pp. 165–177.

[16] S.-K. Han, "An architecture for high-throughput and improved-quality stereo vision processor," *Master's Thesis, Dept. Electrical Engineering, Univ. of Maryland, College Park, MD*, 2010.

[17] O. Gangwal, E. Coezijn, and R.-P. Berretty, "Real-time implementation of depth map post-processing for 3D-TV on a programmable DSP (TriMedia)," in *Dig. Tech. Papers Int. Conf. Consumer Electronics (ICCE)*, Las Vegas, NV, 2009, pp. 1–2.

[18] J.-W. van de Waerdt, S. Vassiliadis, S. Das, S. Mirolo, C. Yen, B. Zhong, C. Basto, J.-P. van Itegem, D. Amirtharaj, K. Kalra, P. Rodriguez, and H. van Antwerpen, "The TM3270 media-processor," in *Proc. 38th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO-38)*, Barcelona, Spain, 2005, pp. 331–342.

Chieh-Chi Kao received the B.S. degree in electrical engineering and the M.S. degree in the Graduate Institute of Electronics Engineering from National Taiwan University (NTU), Taipei, Taiwan, in 2010 and 2012, respectively. His research interests are in image & video processing, computational photography, computer vision, and related VLSI architecture.

Jui-Hsin Lai received the B.S. degree in electronics engineering from the National Chiao-Tung University (NCTU), Hsinchu, Taiwan, in 2005. In 2011, he received the Ph.D. degree from the Graduate Institute of Electronics Engineering at National Taiwan Uni- versity (NTU), Taipei, Taiwan. During 2007 to 2011, he was a project leader in Yotta-Labs, an IC design house in Taipei, for designing the algorithm and hardware architecture of vision-based object tracking, face detection, and recognition. Since 2011, he was been a post-doctoral fellow in the Graduate Institute of Networking and Multimedia, NTU. His research interests include the applications of interactive multimedia, computer vision, sports video, video/image processing, and VLSI architecture design of multimedia processing.

Shao-Yi Chien (S'99–M'04) received the B.S. and Ph.D. degrees from the Department of Electrical Engineering, National Taiwan University (NTU), Taipei, Taiwan, in 1999 and 2003, respectively. During 2003 to 2004, he was a research staff in Quanta Research Institute, Tao Yuan County, Taiwan. In 2004, he joined the Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, as an Assistant Professor. Since 2012, he has been a Professor. His research interests include video segmentation algorithm, intelligent video coding technology, perceptual coding technology, image processing for digital still cameras and display devices, computer graphics, and the associated VLSI and processor architectures. He has published more than 200 papers in these areas.

Dr. Chien serves as an Associate Editor for IEEE Transactions on Circuits and Systems for Video Technology, IEEE Transactions on Circuits and Systems I: Regular Papers, and Springer Circuits, Systems and Signal Processing (CSSP). He also served as a Guest Editor for Springer Journal of Signal Processing Systems in 2008. He also serves on the technical program committees of several conferences, such as ISCAS, ICME, SiPS, A-SSCC, and VLSI-DAT.