VLSI Implementation of an Edge-Oriented Image Scaling Processor

Pei-Yin Chen, *Member, IEEE*, Chih-Yuan Lien, and Chi-Pin Lu

***Abstract*—Image scaling is a very important technique and has been widely used in many image processing applications. In this paper, we present an edge-oriented area-pixel scaling processor. To achieve the goal of low cost, the area-pixel scaling technique is implemented with a low-complexity VLSI architecture in our design. A simple edge catching technique is adopted to preserve the image edge features effectively so as to achieve better image quality. Compared with the previous low-complexity techniques, our method performs better in terms of both quantitative evaluation and visual quality. The seven-stage VLSI architecture of our image scaling processor contains 10.4-K gate counts and yields a processing rate of about 200 MHz by using TSMC 0.18-μm technology.**

***Index Terms*—Image scaling, interpolation, pipeline architecture, VLSI.**

I. Introduction

IMAGE scaling is widely used in many fields [1]–[4], ranging from consumer electronics to medical imaging. It is indispensable when the resolution of an image generated by a source device is different from the screen resolution of a target display. For example, we have to enlarge images to fit HDTV or to scale them down to fit the mini-size portable LCD panel. The most simple and widely used scaling methods are the nearest neighbor [5] and bilinear [6] techniques. In recent years, many efficient scaling methods have been proposed in the literature [7]–[14].

According to the required computations and memory space, we can divide the existing scaling methods [5]–[14] into two classes: lower complexity [5]–[8] and higher complexity [9]–[14] scaling techniques. The complexity of the former is very low and comparable to conventional bilinear method. The latter yields visually pleasing images by utilizing more advanced scaling methods. In many practical real-time applications, the scaling process is included in end-user equipment, so a good lower complexity scaling technique, which is simple and suitable for low-cost VLSI implementation, is needed. In this paper, we consider the lower complexity scaling techniques [5]–[8] only.

Kim *et al.* presented a simple area-pixel scaling method in [7]. It uses an area-pixel model instead of the common point-pixel model and takes a maximum of four pixels of the original image to calculate one pixel of a scaled image. By using the area coverage of the source pixels from the applied mask in combination with the difference of luminosity among the source pixels, Andreadis *et al.* [8] proposed a modified area-pixel scaling algorithm and its circuit to obtain better edge preservation. Both [7] and [8] obtain better edge-preservation but require about two times more of computations than the bilinear method.

To achieve the goal of lower cost, we present an edge-oriented area-pixel scaling processor in this paper. The area-pixel scaling technique is approximated and implemented with the proper and low-cost VLSI circuit in our design. The proposed scaling processor can support floating-point magnification factor and preserve the edge features efficiently by taking into account the local characteristic existed in those available source pixels around the target pixel. Furthermore, it handles streaming data directly and requires only small amount of memory: one line buffer rather than a full frame buffer. The experimental results demonstrate that the proposed design performs better than other lower complexity image scaling methods [5]–[8] in terms of both quantitative evaluation and visual quality.

The seven-stage VLSI architecture for the proposed design was implemented and synthesized by using Verilog HDL and synopsys design compiler, respectively. In our simulation, the circuit can achieve 200 MHz with 10.4-K gate counts by using TSMC 0.18-μm technology. Since it can process one pixel per clock cycle, it is quick enough to process a video resolution of WQSXGA (3200×2048) at 30 f/s in real time.

This paper is organized as follows. In Section II, the area-pixel scaling technique is introduced briefly. Our method is presented in Section III. Section IV describes the proposed VLSI architecture in detail. Section V illustrates the simulation results and chip implementation. The conclusion is provided in Section VI.

II. Area-Pixel Scaling Technique

In this section, we first introduce the concepts of the area-pixel scaling technique. Then the hardware implementation issues of it are briefly reviewed.

A. An Overview of Area-Pixel Scaling Technique

Assume that the source image represents the original image to be scaled up/down and target image represents the scaled image. The area-pixel scaling technique performs scale-up/scale-down transformation by using the area pixel model instead of the common point model. Each pixel is treated as one small rectangle but not a point; its intensity is evenly distributed in the rectangle area. Fig. 1 shows an example of image scaleup process of the area-pixel model where a

Manuscript received January 11, 2008; revised May 26, 2008. First published March 10, 2009; current version published August 19, 2009. This work was supported in part by the National Science Council, Taiwan, under Grant NSC-96-2221-E-006-027- MY3. Also, this work made use of Shared Facilities supported by the Program of Top 100 Universities Advancement, Ministry of Education, Taiwan.

The authors are with the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan (e-mail: py-chen@csie.ncku.edu.tw; lian@csie.ncku.edu.tw; bigo@csie.ncku.edu.tw).

Digital Object Identifier 10.1109/TVLSI.2008.2003003

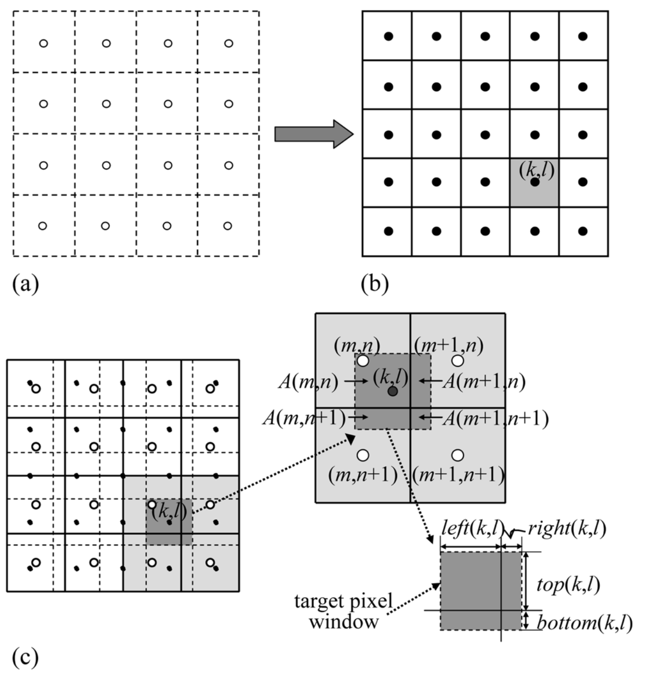

Fig. 1. Example of image enlargement using typical area-pixel model. (a) A source image of 4 × 4 pixels. (b) A target image of 5 × 5 pixels. (c) Relations of the target pixel and source pixels.

source image of 4 × 4 pixels is scaled up to the target image of 5 × 5 pixels. Obviously, the area of a target pixel is less than that of a source pixel. A window is put on the current target pixel to calculate its estimated luminance value. As shown in Fig. 1(c), the number of source pixels overlapped by the current target pixel window is one, two, or a maximum of four. Let the luminance values of four source pixels overlapped by the window of current target pixel at coordinate (k, l) be denoted as $F_S(m, n), F_S(m+1, n), F_S(m, n+1)$ and $F_S(m+1, n+1)$, respectively. The estimated value of current target pixel, denoted as $\hat{F}_T(k, l)$, can be calculated by weighted averaging the luminance values of four source pixels with area coverage ratio as

$$\hat{F}_T(k,l) = \sum_{i=0}^{1}\sum_{j=0}^{1}[F_S(m+i, n+j) \times W(m+i, n+j)] \quad (1)$$

where $W(m, n), W(m+1, n), W(m, n+1)$, and $W(m+1, n+1)$ represent the weight factors of neighboring source pixels for the current target pixel at (k, l). Assume that the regions of four source pixels overlapped by current target pixel window are denoted as $A(m, n), A(m+1, n), A(m, n+1)$, and $A(m+1, n+1)$, respectively, and the area of the target pixel window is denoted as $A_{\rm sum}$. The weight factors of four source pixels can be given as

$$\begin{aligned}&[W(m,n), W(m+1,n), W(m,n+1), W(m+1,n+1)]\\ &\quad = \left[\frac{A(m,n)}{A_{\rm sum}}, \frac{A(m+1,n)}{A_{\rm sum}}, \frac{A(m,n+1)}{A_{\rm sum}}, \frac{A(m+1,n+1)}{A_{\rm sum}}\right]\end{aligned} \quad (2)$$

where $A_{\rm sum} = A(m, n) + A(m+1, n) + A(m, n+1) + A(m+1, n+1)$.

Let the width and height of the overlapped region $A(m, n)$ be denoted as $\text{left}(k, l)$ and $\text{top}(k, l)$, and the width and height of $A(m+1, n+1)$ be denoted as $\text{right}(k, l)$ and $\text{bottom}(k, l)$, respectively, as shown in Fig. 1(c). Then, the areas of the overlapped regions can be calculated by

$$\begin{aligned}&[A(m,n), A(m+1,n), A(m,n+1), A(m+1,n+1)]\\ &\quad = [\text{left}(k,l) \times \text{top}(k,l), \text{right}(k,l) \times \text{top}(k,l),\\ &\qquad \text{left}(k,l) \times \text{bottom}(k,l),\\ &\qquad \text{right}(k,l) \times \text{bottom}(k,l)].\end{aligned} \quad (3)$$

Obviously, many floating-point operations are needed to determine the four parameters, $\text{left}(k, l), \text{top}(k, l), \text{right}(k, l)$ and $\text{bottom}(k, l)$, if the direct area-pixel implementation is adopted.

B. Hardware Implementation Issues for Area-Pixel Model

The main difficulty to implement the area-pixel scaling with hardware is that it requires a lot of extensive and complex computations. As shown in (1)–(3), total of 13 addition, eight multiplications, and one division floating-point operations are required to calculate one target pixel. In [7] and [8], the precision of those floating-point operations is set as 30 b. To obtain lower hardware cost, we adopt an approximate technique, mentioned later, to reduce implementation complexity and to improve scaling speed. Most variables are represented with the 4-, 6-, or 8-b unsigned integers in our low-cost scaling processor.

Furthermore, the typical area-pixel scaling method needs to calculate six coordinate values for each target pixel's estimation. To reduce the computational complexity, we employ an alternate approach suitable for VLSI implementation to determine those necessary coordinate values efficiently and quickly.

III. The Proposed Low-Complexity Algorithm

Observing (1)–(3), we know that the direct implementation of area-pixel scaling requires some extensive floating-point computations for the current target pixel at (k, l) to determine the four parameters, $\text{left}(k, l), \text{top}(k, l), \text{right}(k, l)$, and $\text{bottom}(k, l)$. In the proposed processor, we use an approximate technique suitable for low-cost VLSI implementation to achieve that goal properly. We modify (3) and implement the calculation of areas of the overlapped regions as

$$\begin{aligned}&[A'(m,n), A'(m+1,n), A'(m,n+1), A'(m+1,n+1)]\\ &\quad = [\text{left}'(k,l) \times \text{top}'(k,l), \text{right}'(k,l) \times \text{top}'(k,l),\\ &\qquad \text{left}'(k,l) \times \text{bottom}'(k,l), \text{right}'(k,l) \times \text{bottom}'(k,l)].\end{aligned} \quad (4)$$

Those $\text{left}'(k, l), \text{top}'(k, l), \text{right}'(k, l)$, and $\text{bottom}'(k, l)$ are all 6-b integers and given as

$$\begin{aligned}&[\text{left}'(k,l), \text{top}'(k,l), \text{right}'(k,l), \text{bottom}'(k,l)]\\ &\quad = \text{Appr}[\text{left}(k,l), \text{top}(k,l), \text{right}(k,l), \text{bottom}(k,l)]\end{aligned} \quad (5)$$

```
$win_w$ = initialwin($mf\_w$);   /* initial window values */
$win_h$ = initialwin($mf\_h$);
$A_{sum} = win_w \times win_h$;
for($k = 0; k < row; k = k + 1$)   /* target image size : $row$(height) × $col$(width) */
{
  for($l = 0; l < col; l = l + 1$)
  {
/* Approximate Module */
    Get $F_S(m,n), F_S(m,n+1), F_S(m+1,n)$ and $F_S(m+1,n+1)$;
     /* the luminance values of four source pixels overlapped by the window
        of current target pixel at coordinate $(k,l)$ */
    Calculate $win_{left}(k,l), src_{right}(m,n), win_{top}(k,l), src_{btm}(m,n)$;
     /* see equations(10),(11),(13) and (14) */
    $left'(k,l) = \min(src_{right}(m,n) - win_{left}(k,l), win_w)$;
     /* min : the minimum operator */
    $right'(k,l) = win_w - left'(k,l)$;
    $top'(k,l) = \min(src_{btm}(m,n) - win_{top}(k,l), win_h)$;
    $bottom'(k,l) = win_h - top'(k,l)$;
    $A'(m,n) = left'(k,l) \times top'(k,l)$;
    $A'(m+1,n) = right'(k,l) \times top'(k,l)$;
    $A'(m,n+1) = left'(k,l) \times bottom'(k,l)$;
    $A'(m+1,n+1) = right'(k,l) \times bottom'(k,l)$;

/* Edge-Catching Module */
    if ($top'(k,l) >= win_h/2$)
    {  $L_A = |F_S(m+1,n) - F_S(m-1,n)| - |F_S(m+2,n) - F_S(m,n)|$;
       if ($L_A >= 0$)   $A_C = A'(m,n)$;
       else   $A_C = A'(m+1,n)$;
       $A''(m,n) = A'(m,n) - L_A \times A_C/256$;
       /* the division operation is implemented by the shift operator */
       $A''(m+1,n) = A'(m+1,n) + L_A \times A_C/256$;
       $A''(m,n+1) = A'(m,n+1)$;   $A''(m+1,n+1) = A'(m+1,n+1)$;
    } else
    {  $L_A = |F_S(m+1,n+1) - F_S(m-1,n+1)| - |F_S(m+2,n+1) - F_S(m,n+1)|$;
       if ($L_A >= 0$)   $A_C = A'(m,n+1)$;
       else   $A_C = A'(m+1,n+1)$;
       $A''(m,n) = A'(m,n)$;   $A''(m+1,n) = A'(m+1,n)$;
       $A''(m,n+1) = A'(m,n+1) - L_A \times A_C/256$;
       $A''(m+1,n+1) = A'(m+1,n+1) + L_A \times A_C/256$;
    }
    $\hat{F}_T(k,l) = (F_S(m,n) \times A''(m,n) + F_S(m+1,n) \times A''(m+1,n) + F_S(m,n+1) \times A''(m,n+1)$
             $+ F_S(m+1,n+1) \times A''(m+1,n+1))/A_{sum}$;
  }
}
```

Fig. 2. Pseudocode of our scaling method.

where Appr represents the approximate operator adopted in our design and will be explained in detail later.

To obtain better visual quality, a simple low-cost edge-catching technique is employed to preserve the edge features effectively by taking into account the local characteristic existed in those available source pixels around the target pixel. The final areas of the overlapped regions are given as

$$\begin{aligned}&[A''(m,n), A''(m+1,n), A''(m,n+1), A''(m+1,n+1)]\\ &\quad = \Gamma([A'(m,n), A'(m+1,n), A'(m,n+1), A'(m+1,n+1)])\end{aligned} \tag{6}$$

where we adopt a tuning operator Γ to tune the areas of four overlapped regions according to the edge features obtained by our edge-catching technique. By applying (6) to (1) and (2), we can determine the estimated luminance value of the current target pixel. Fig. 2 shows the pseudocode of our scaling method. In the rest of this section, the approximate technique adopted in our design is introduced first. Then we describe the low-cost edge-catching technique in detail.

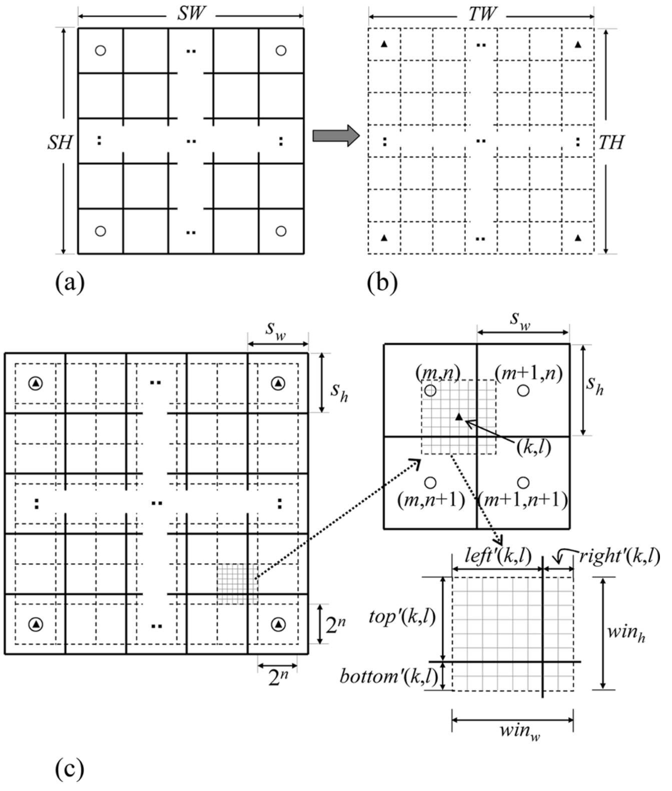

Fig. 3. Example of image enlargement for our method. (a) A source image of $\mathrm{SW} \times \mathrm{SH}$ pixels. (b) A target image of $\mathrm{TW} \times \mathrm{TH}$ pixels. (c) Relations of the target pixel and source pixels.

A. The Approximate Technique

Fig. 3 shows an example of our image scaleup process where a source image of $\mathrm{SW} \times \mathrm{SH}$ pixels is scaled up to the target image of $\mathrm{TW} \times \mathrm{TH}$ pixels and every pixel is treated as one rectangle. As shown in Fig. 3(c), we align those centers of four corner rectangles (pixels) in the source and target images. For simple hardware implementation, each rectangular target pixel is treated as $2^n \times 2^n$ grids with uniform size (n is 3 for Fig. 3). Assume that the width and the height of the target pixel window are denoted as win_w and win_h, then the area of the current target pixel window (A_{sum}) can be calculated as $\mathrm{win}_w \times \mathrm{win}_h$. In our design, the values of win_w and win_h are determined based on the current magnification factors, mf_w for x direction and mf_h for y direction where $mf\_w = \mathrm{TW}/\mathrm{SW}$ and $mf\_h = \mathrm{TH}/\mathrm{SH}$. In the case of image enlargement, $\mathrm{win}_w = 2^n$ when $100\% \leq mf\_w \leq 200\%$. When $200\% < mf\_w \leq 400\%$, win_w will be enlarged to 2^{n+1}, and so on. In the case of image reduction, win_w is reduced to 2^{n-1} when $50\% \leq mf\_w < 100\%$. When $25\% \leq mf\_w < 50\%$, win_w is 2^{n-2}, and so on. With the similar way, we can also determine the value of win_h by using mf_h. In the design, the division operation in (2) can be implemented simply with a shifter.

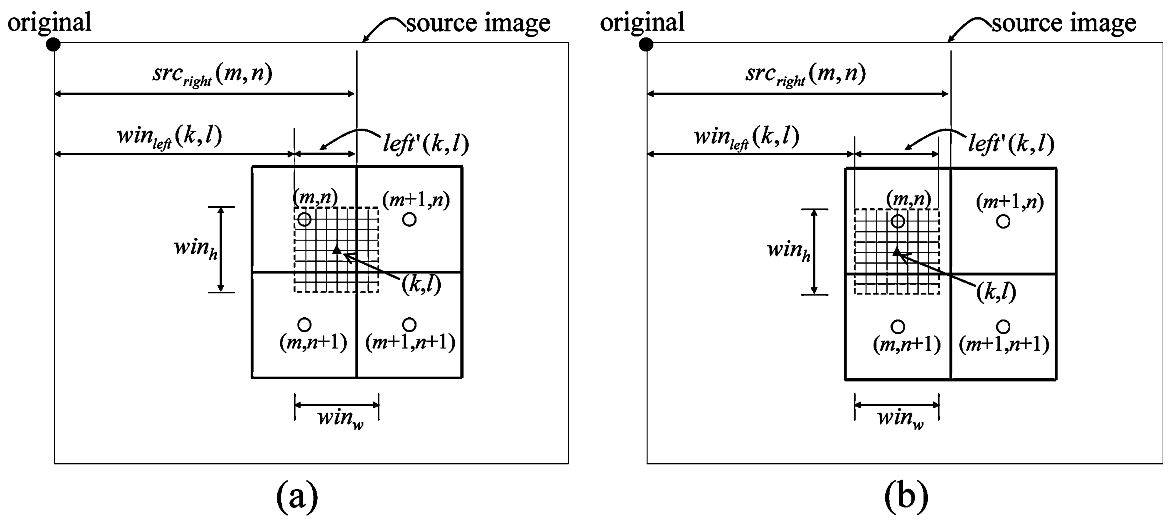

Fig. 4. Two possible cases for $\text{left}'(k,l)$. (a) $\text{left}'(k,l) = \text{src}_{\text{right}}(m,n) - \text{win}_{\text{left}}(k,l)$. (b) $\text{left}'(k,l) = \text{win}_w$.

As shown in Fig. 3(c), the relationships among $\text{left}'(k,l)$, $\text{top}'(k,l)$, $\text{right}'(k,l)$, and $\text{bottom}'(k,l)$ can be denoted as follows:

$$\text{right}'(k,l) = \text{win}_w - \text{left}'(k,l) \tag{7}$$

$$\text{bottom}'(k,l) = \text{win}_h - \text{top}'(k,l). \tag{8}$$

As soon as $\text{left}'(k,l)$ and $\text{top}'(k,l)$ are determined, $\text{right}'(k,l)$ and $\text{bottom}'(k,l)$ can be calculated easily. Thus, we focus on finding the values of $\text{left}'(k,l)$ and $\text{top}'(k,l)$ only. In our design, $\text{left}'(k,l)$ is calculated as

$$\text{left}'(k,l) = \min(\text{src}_{\text{right}}(m,n) - \text{win}_{\text{left}}(k,l), \text{win}_w) \tag{9}$$

where min represents the minimum operation, and $\text{win}_{\text{left}}(k,l)$ represents the horizontal displacement (in the unit of grid) from the left boundary of the source image to the left side of the current pixel window at coordinate (k,l), and can be calculated as

$$\text{win}_{\text{left}}(k,l) = \text{win}_{\text{left}}(k-1,l) + 2^n. \tag{10}$$

$\text{src}_{\text{right}}(m,n)$ represents the horizontal displacement (in the unit of grid) from the left boundary of the source image to the right side of the top-left source pixel overlapped by the current pixel window at (k,l), and can be calculated as

$$\text{src}_{\text{right}}(m,n) = \text{src}_{\text{right}}(m-1,n) + s_w + \tau_w \tag{11}$$

where s_w is the width of a source pixel relative to a $2^n \times 2^n$ target pixel and τ_w is the regulating value used to reduce the accumulated error caused by rounding s_w. Both s_w and τ_w are in the unit of grid. As shown in Fig. 4(a), if $\text{src}_{\text{right}}(m,n) - \text{win}_{\text{left}}(k,l)$ is smaller than win_w, the current target pixel's $\text{left}'(k,l)$ is equal to $\text{src}_{\text{right}}(m,n) - \text{win}_{\text{left}}(k,l)$. Otherwise, $\text{left}'(k,l)$ is equal to win_w, as shown in Fig. 4(b).

Similarly, $\text{top}'(k,l)$ is given as

$$\text{top}'(k,l) = \min(\text{src}_{\text{btm}}(m,n) - \text{win}_{\text{top}}(k,l), \text{win}_h) \tag{12}$$

where $\text{win}_{\text{top}}(k,l)$ represents the vertical displacement from the top boundary of the source image to the top side of the target pixel window at (k,l), and can be calculated as

$$\text{win}_{\text{top}}(k,l) = \text{win}_{\text{top}}(k,l-1) + 2^n. \tag{13}$$

$\text{src}_{\text{btm}}(m,n)$ represents the vertical displacement from the top boundary of the source image to the bottom side of the top-left source pixel overlapped by the target pixel window at coordinate (k,l), and can be calculated as

$$\text{src}_{\text{btm}}(m,n) = \text{src}_{\text{btm}}(m,n-1) + s_h + \tau_h \tag{14}$$

where s_h is the height of a source pixel in the unit of grid, and τ_h is the regulating value used to reduce the accumulating error caused by rounding s_h. Initially, $\text{win}_{\text{left}}(0,0) = (s_w - \text{win}_w)/2$, $\text{src}_{\text{right}}(0,0) = s_w$, $\text{win}_{\text{top}}(0,0) = (s_h - \text{win}_h)/2$, and $\text{src}_{\text{btm}}(0,0) = s_h$. All variables among (9)–(14) are integers. We use a few low-cost integer addition/subtraction operations rather than extensive floating-point multiplication/division computations to obtain the approximated values of $\text{left}'(k,l)$ and $\text{top}'(k,l)$. In the following paragraph, the steps to determine τ_w and τ_h are described.

Since we set each target pixel as $2^n \times 2^n$ grids, s_w and s_h can be denoted and calculated as follows:

$$s_w = [(\text{TW}-1)/(\text{SW}-1)] \times 2^n \tag{15}$$

$$s_h = [(\text{TH}-1)/(\text{SH}-1)] \times 2^n. \tag{16}$$

In the design, both s_w and s_h are rounded to integers. The rule is that a fractional part of less than 0.5 is dropped, while a fractional part of 0.5 or more leads to be rounded to the next bigger integer. The former will produce the rounding-down error and the latter will produce the rounding-up error. Each kind of errors is accumulated and might cause the values of $\text{left}'(k,l)$ or $\text{top}'(k,l)$ to be incorrect. To reduce accumulated rounding error, we adopt τ_w and τ_h to regulate $\text{left}'(k,l)$ and $\text{top}'(k,l)$, respectively. There are two working modes existed in our processor. At normal mode, the accumulation of rounding-up/down error of $\text{left}'(k,l)$ is less than one grid, so no regulation is needed and τ_w will be set to zero. As soon as the accumulation of

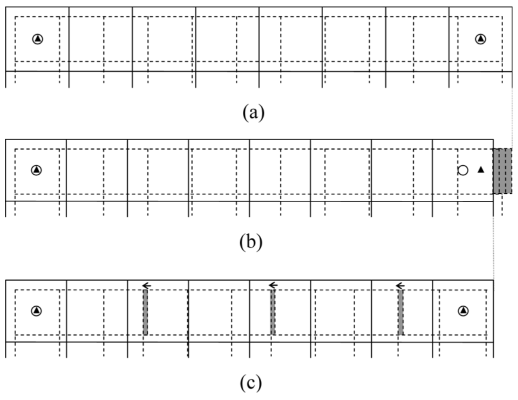

Fig. 5. Example of the overlapping method for rounding-down error. (a) An ideal scaleup example from 8×8 to 11×11. (b) The rounding-down error without regulation. (c) The proposed overlapping method.

rounding-up/down error of $\text{left}'(k,l)$ is greater than or equal to one grid, the processor will enter regulating mode and set the value of τ_w to $-1/1$. The same idea can be applied to $\text{top}'(k,l)$ and τ_h.

Let r_w represent the regulating times required for each row, thus it can be given as

$$r_w = \begin{cases} 2^n \times (\text{TW} - 1) - s_w \times (\text{SW} - 1), \\ \quad \text{if } s_w \text{ is rounded up to an integer} \\ s_w \times (\text{SW} - 1) - 2^n \times (\text{TW} - 1), \\ \quad \text{if } s_w \text{ is rounded down to an integer.} \end{cases} \tag{17}$$

In other words, there are r_w times that τ_w is set as -1 or $+1$ for each row. If s_w is rounded down, the total sum of grids at x direction in the target image is larger than that in the source image without regulation. Therefore, it is necessary to "compress" the target image by overlapping r_w grids. We choose r_w pixels in a row of the target image regularly, and shift left each pixel of them with one grid to finish aligning. Fig. 5 shows an example of the image scaleup process where a source image of 8×8 pixels is scaled up to the target image of 11×11 pixels. According to (15), s_w is $80/7 = 11.43$ since $\text{SW} = 8, \text{TW} = 11$, and $n = 3$. Then, s_w is rounded down to the integer 11 and r_w is 3, as shown in Fig. 5(b). Therefore, we overlap three grids via shifting left three target pixels with one grid in this row to do the job of aligning, shown in Fig. 5(c). The accumulation effect of rounding errors can be reduced. On the contrary, if s_w is rounded up, we need to "expand" the target image by inserting r_w grids. We choose r_w pixels in a row of the target image regularly, and shift right each pixel of them with one grid to do aligning. Fig. 6 shows another example of the image scaleup process where a source image of 8×8 pixels is scaled up to the target image of 13×13 pixels. According to (15), s_w is $96/7 = 13.71$ since $\text{SW} = 8, \text{TW} = 13$, and $n = 3$. In the example, s_w is rounded up to the integer 14 and r_w is 2, as shown in Fig. 6(b). Therefore, two grids are inserted via shifting right two target pixels with one grid in this row to do aligning, shown in Fig. 6(c). The same way is also applied to the vertical-direction process. Using (7)–(17), we can realize the approximate operator Appr in (5) with the low-complexity computations.

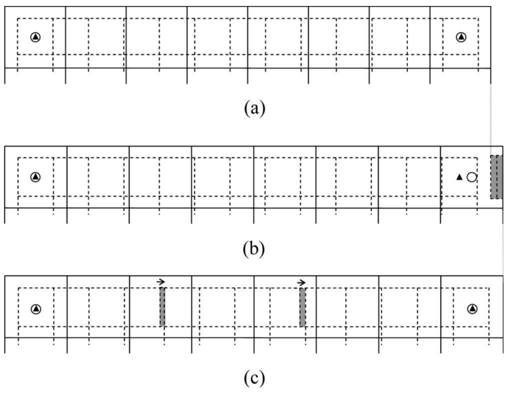

Fig. 6. Example of the inserting method for rounding-up error. (a) An ideal scaleup example from 8×8 to 13×13. (b) The rounding-up error without regulation. (c) The proposed inserting method.

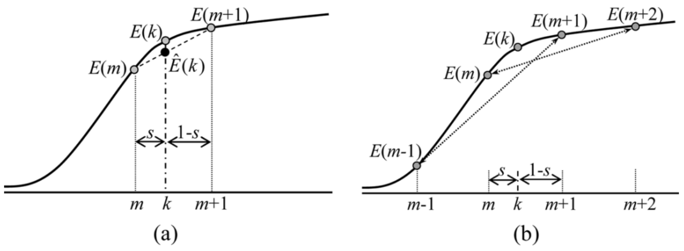

Fig. 7. Local characteristics of the data in the neighborhood of k. (a) An image-edge model. (b) The local characteristics.

B. The Low-Cost Edge-Catching Technique

In the design, we take the sigmoidal signal [15] as the image-edge model for image scaling. Fig. 7(a) shows an example of the 1-D sigmoidal signal. Assume that the pixel to be interpolated is located at coordinate k and its nearest available neighbors in the image are located at coordinate m for the left and $m+1$ for the right. Let $s = k - m$ and $E(k)$ represent the luminance value of the pixel at coordinate k. If the estimated value $\hat{E}(k)$ of the pixel to be interpolated is determined by using linear interpolation, it can be calculated as

$$\hat{E}(k) = (1-s) \times E(m) + s \times E(m+1). \tag{18}$$

As shown in Fig. 7(a), $\hat{E}(k)$ and $E(k)$ might not match greatly. To solve the problem, we modify the distance s to make $\hat{E}(k)$ approach $E(k)$ for a better estimation.

Assume that the coordinates of the four nearest available neighbors around the current pixel are located at $m-1, m, m+1$, and $m+2$, respectively, as shown in Fig. 7(b). In our design, we define an evaluating parameter L to estimate the local characteristic of the data in the neighborhood of k. It is given as

$$L = |E(m+1) - E(m-1)| - |E(m+2) - E(m)|. \tag{19}$$

If the image data are sufficiently smooth and the luminance changes at object edges can be approximated by sigmoidal functions, we can come to the following conclusion. $L = 0$ indicates symmetry, so s is unchanged. $L > 0$ indicates that the variation between $E(m+1)$ and $E(m-1)$ is quicker than that between

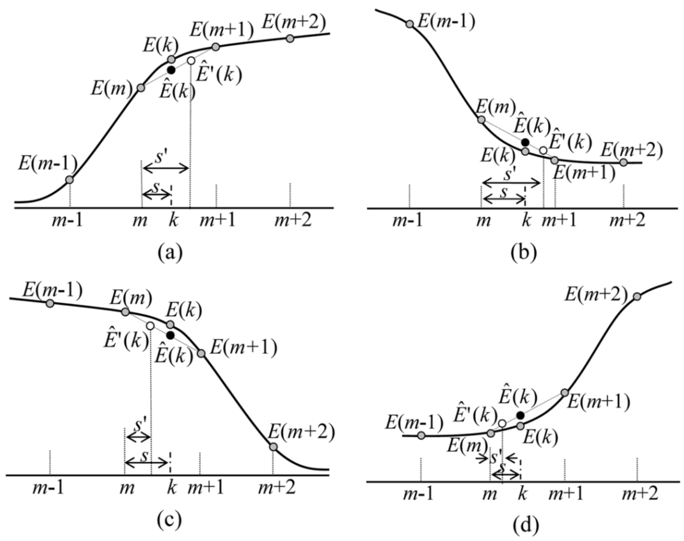

Fig. 8. Four possible cases for the sigmoidal image model. (a) Convex and $L > 0$. (b) Concave and $L > 0$. (c) Convex and $L < 0$. (d) Concave and $L < 0$.

$E(m+2)$ and $E(m)$. It means that the edge is more homogeneous on the right side, as shown in Fig. 8(a) and (b), so the pixel located at coordinate $m+1$ should affect the interpolated value more than the pixel located at coordinate m does. Hence, we can increase s in order to make the estimated value close to the expected value. On the contrary, $L < 0$ indicates the edge is more homogeneous on the left side, as shown in Fig. 8(c) and (d). Thus, we must decrease s to obtain a better estimation. Based on the above idea, we modify (18) and calculate the estimated value of the current pixel as

$$\hat{E}'(k) = (1 - s') \times E(m) + s' \times E(m+1) \tag{20}$$

where s' is calculated with a simple way and given as

$$s' = \begin{cases} s + L \times (1 - s)/2^8, & \text{if } L \geq 0 \\ s + L \times s/2^8, & \text{if } L < 0. \end{cases} \tag{21}$$

A small amount of operations is required to catch the local characteristic of the current pixel.

By using the concept of 1-D edge-catching technique shown in (19)–(21), we can tune the areas of four overlapped regions adaptively in the proposed 2-D scaling processor to obtain better image quality. Let L_A represent the evaluating parameter to estimate the local characteristic of the current pixel at coordinate (k, l). If $\mathrm{top}'(k, l)$ is greater than or equal to $\mathrm{win}_h/2$, it means that $A'(m, n)$ is bigger than or equal to $A'(m, n+1)$. Hence, the upper row (n) is more important than the lower row $(n+1)$ to catch edge features. Thus, L_A is given as

$$L_A = |E(m{+}1, n) – E(m–1, n)| – |E(m{+}2, n) – E(m, n)|. \tag{22}$$

$L_A = 0$ indicates symmetry, so $A'(m, n)$ is unchanged. $L_A > 0$ indicates that the variation between $E(m+1, n)$ and $E(m-1, n)$ is quicker than that between $E(m+2, n)$ and $E(m, n)$. It means that the edge is more homogeneous on the right-hand side, so we can increase $A'(m+1, n)$ in order to make the estimated value close to the expected one. On the contrary, $L_A < 0$ indicates the edge is more homogeneous on the left-hand side, thus we decrease $A'(m+1, n)$ to obtain a better estimate. Applying the above idea to (6), we can calculate the final areas of the overlapped regions as

$$\begin{aligned}&[A''(m,n), A''(m+1,n), A''(m,n+1), A''(m+1,n+1)] \\ &\quad = [A'(m,n) - L_A \times A_C/2^8, A'(m{+}1,n) + L_A \times A_C/2^8, \\ &\qquad A'(m,n{+}1), A'(m{+}1,n{+}1)]\end{aligned} \tag{23}$$

where $A_C = A'(m, n)$ if $L_A \geq 0$, and $A_C = A'(m+1, n)$ if $L_A < 0$.

On the contrary, if $\mathrm{top}'(k, l)$ is less than $\mathrm{win}_h/2$,, it means that $A'(m, n)$ is smaller than $A'(m, n+1)$. Hence, the lower row $(n+1)$ is more important than the upper row (n) to catch edge features. Thus, L_A is given as

$$\begin{aligned}L_A = &|E(m+1, n+1) - E(m-1, n+1)| \\ &- |E(m+2, n+1) - E(m, n+1)|.\end{aligned} \tag{24}$$

The final areas of the overlapped regions are given as

$$\begin{aligned}&[A''(m,n), A''(m{+}1,n), A''(m,n{+}1), A''(m{+}1,n{+}1)] \\ &\quad = [A'(m,n), A'(m{+}1,n), A'(m,n{+}1) - L_A \times A_C/2^8, \\ &\qquad A'(m+1,n+1) + L_A \times A_C/2^8]\end{aligned} \tag{25}$$

where $A_C = A'(m, n+1)$ if $L_A \geq 0$, and $A_C = A'(m+1, n+1)$ if $L_A < 0$.

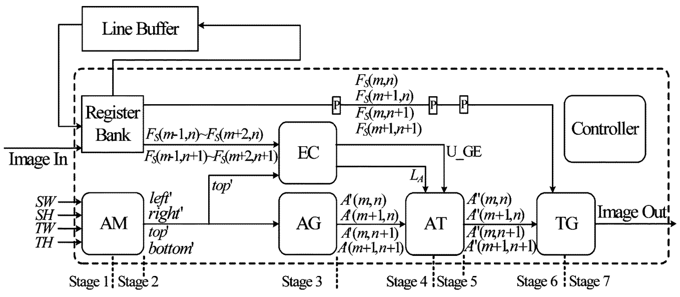

Fig. 9. Block diagram of VLSI architecture for our scaling method.

IV. VLSI Architecture

Our scaling method requires low computational complexity and only one line memory buffer, so it is suitable for low-cost VLSI implementation. Fig. 9 shows block diagram of the seven-stage VLSI architecture for our scaling method. The architecture consists of seven main blocks: approximate module (AM), register bank (RB), area generator (AG), edge catcher (EC), area tuner (AT), target generator (TG), and the controller. Each of them is described briefly in the following subsections.

A. Approximate Module

When a source image of SW × SH pixels is scaled up or down to the target image of TW × TH pixels, the AM performs (7)–(17) mentioned in Section III-A, and generates $\mathrm{left}'(k, l), \mathrm{top}'(k, l), \mathrm{right}'(k, l)$ and $\mathrm{bottom}'(k, l)$,

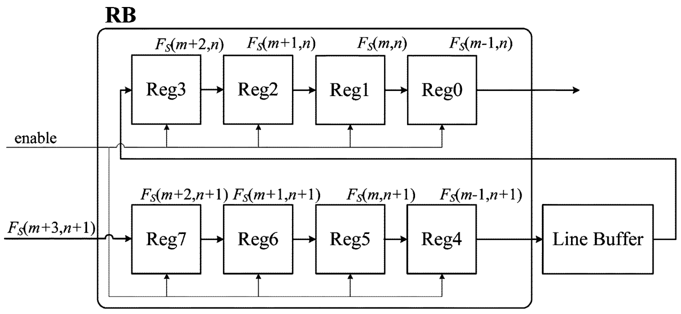

Fig. 10. Architecture of register bank.

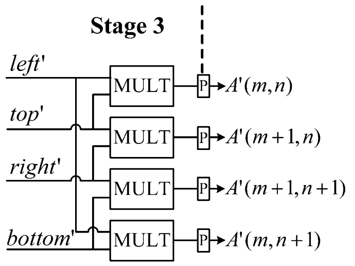

Fig. 11. Architecture of area generator.

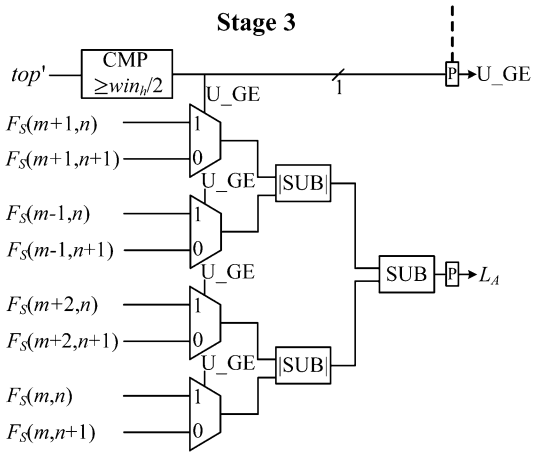

Fig. 12. Architecture of edge catcher.

respectively, for each target pixel from left to right and from top to bottom. In our VLSI implementation, n is set to 3, so each rectangular target pixel is treated as $2^3 \times 2^3$ uniform-sized grids. win_w and win_h are both 6-b integers and their values are restricted to power of 2, so $\mathrm{win}_w, \mathrm{win}_h \in \{1, 2, 4, 8, 16, 32\}$. Based on the approximate technique mentioned in the Section III-A, the minimum and the maximum of magnification factors (mf_w and mf_h) supported by the design are 0.125 and 8, respectively. Hence, the minimum and the maximum of magnification factor $(mf = mf\_w \times mf\_h)$ supported by the design are 1/64 and 64, respectively.

AM is composed of two-stage pipelined architecture. In the first stage, the coordinate (k, l) of the current target pixel and the coordinate (m, n) of the top-left source pixel overlapped by the current window are determined. In the second stage, AM first calculates $\mathrm{win}_{\mathrm{left}}(k,l), \mathrm{src}_{\mathrm{right}}(m,n), \mathrm{win}_{\mathrm{top}}(k,l)$ and $\mathrm{src}_{\mathrm{btm}}(m,n)$ according to (10)–(11) and (13)–(14), and then generates $\mathrm{left}'(k,l), \mathrm{right}'(k,l), \mathrm{top}'(k,l)$, and $\mathrm{bottom}'(k,l)$ according to (7)–(9) and (12).

B. Register Bank

In our design, the estimated value of the current target pixel $\hat{F}_T(k,l)$ is calculated by using the luminance values of 2×4 neighboring source pixels $F_S(m-1,n), F_S(m,n), F_S(m+1,n), F_S(m+2,n), F_S(m-1,n+1), F_S(m,n+1), F_S(m+1,n+1)$, and $F_S(m+2,n+1)$. The register bank, consisting of eight registers, is used to provide those source luminance values at exact time for the estimated process of current target pixel. Fig. 10 shows the internal connections of RB where every four registers are connected serially in a chain to provide four pixel values of a row in current pixel window, and the line buffer is used to store the pixel values of one row in the source image.

When the controller enables the shift operation in RB, two new values are read into RB (Reg3 and Reg7) and the rest 6-pixel values are shifted to their right registers one by one. The 8-pixel values stored in RB will be used by EC for edge catching and by TG for target pixel estimating.

C. Area Generator

For each target pixel, AG calculates the areas of the overlapped regions $A'(m,n), A'(m+1,n), A'(m,n+1)$, and $A'(m+1,n+1)$ according to (4). Fig. 11 shows the architecture of AG where P represents the pipeline register and MULT is the 4×4 integer multiplier.

D. Edge Catcher

EC implements the proposed low-cost edge-catching technique and outputs the evaluating parameter L_A, which represents the local edge characteristic of current pixel at coordinate (k,l). Fig. 12 shows the architecture of EC where SUB unit generates the difference of two inputs and |SUB| unit generates the two inputs' absolute value of difference. The comparator CMP outputs logic 1 if the input value is greater than or equal to $\mathrm{win}_h/2$. The binary compared result, denoted as U_GE, is used to decide whether the upper row (row n) in current pixel window is more important than the lower row (row $n+1$) in regards to catch edge features. According to (22) and (24), EC produces the final result L_A and sends it to the following AT.

E. Area Tuner

AT is used to modify the areas of the four overlapped regions based on the current local edge information (L_A and U_GE provided by EC). Fig. 13 shows the two-stage pipeline architecture of AT. If U_GE is equal to 1, the upper row (row n) in current pixel window is more important, so $A'(m,n)$ and $A'(m+1,n)$ are modified according to (23). On the contrary, if U_GE is equal to 0, the lower row (row $n+1$) is more important, so $A'(m,n+1)$ and $A'(m+1,n+1)$ are modified according to (25). Finally, the tuned areas $A''(m,n), A''(m+1,n), A''(m,n+1)$ and $A''(m+1,n+1)$ are sent to TG.

F. Target Generator

By weighted averaging the luminance values of four source pixels with tuned-area coverage ratio, TG implements (1) and (2) to determine the estimated value $\hat{F}_T(k,l)$. Fig. 14 shows the

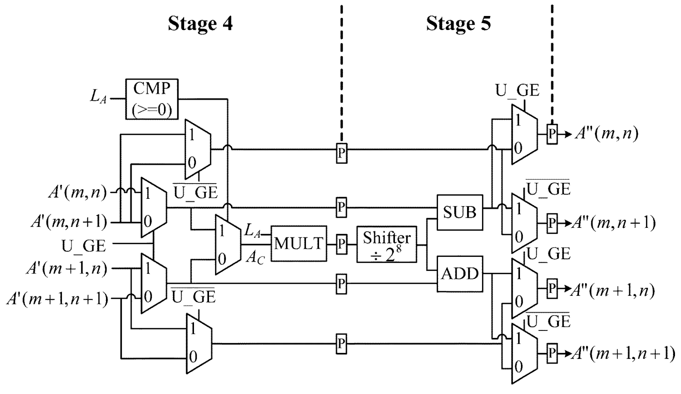

Fig. 13. Architecture of area tuner.

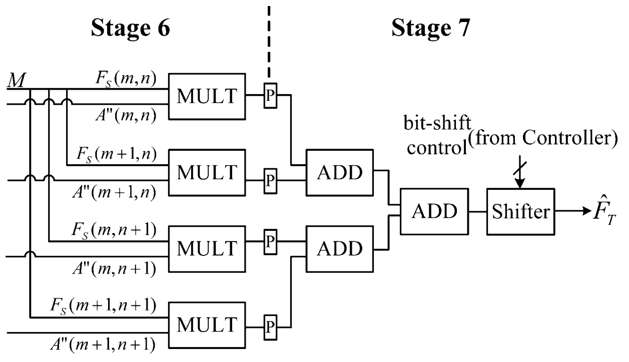

Fig. 14. Architecture of target generator.

two-stage pipeline architecture of TG. Four MULT units and three ADD units are used to perform (1). Since the value of A_{sum} is equal to the power of 2, the division operation in (2) can be implemented by the shifter easily.

G. Controller

The controller, realized with a finite-state machine, monitors the data flow and sends proper control signals to all other components. In the design, AM, AT, and TG require two clock cycles to complete their functions, respectively. Both AG and EC need one clock cycle to finish their tasks, and they work in parallel because no data dependency between them exists. For each target pixel, seven clock cycles are needed to output the estimated value $\hat{F}_T(k,l)$.

V. Simulation Results and Chip Implementation

To evaluate the performance of our image-scaling algorithm, we use 12 gray-scale test images of 512×512 8 b, shown in Fig. 15. For each single test image, we reduce/enlarge the original image by using the well-known bilinear method, and then employ various approaches to scale up/down the bilinear-scaled image back to the size of the original test image. Thus, we can compare the image quality of the reconstructed images for various scaling methods. Three well-known scaling methods, nearest neighbor (NN) [5], bilinear (BL) [6], and bicubic (BC) [9], two area-pixel scaling methods, Win (winscale in [7]) and M_Win (the modified winscale in [8]), and our method are used for comparison in terms of computational complexity, objective

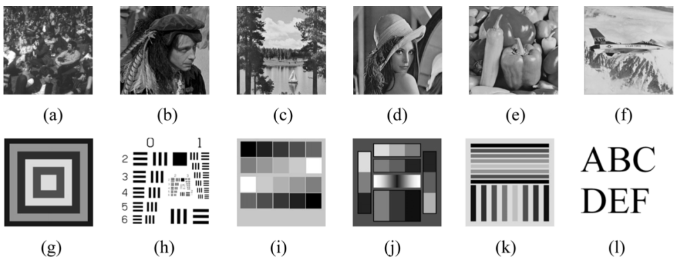

Fig. 15. Twelve reference images for testing. (a) Crowd. (b) Indian. (c) Lake. (d) Lena. (e) Peppers. (f) Plane. (g) Syn_1. (h) Syn_2. (i) Syn_3. (j) Syn_4. (k) Syn_5. (l) Syn_6.

TABLE I
Comparison of Execution Time for Different Methods (Unit: Second)

	NN	BL	BC	Win	M_Win	Our
Pentium 4	0.125	0.155	1.518	0.312	0.351	0.232
PXA270	36.46	85.42	1376.74	232.27	317.84	148.13

testing (quantitative evaluation), and subjective testing (visual quality), respectively.

To reduce hardware cost, we adopt the low-cost technique suitable for VLSI implementation to perform area-pixel scaling. To verify the computational complexity, all the six scaling methods are implemented in C language on the 2.8-GHz Pentium 4 processor with 512-MB memory and the 520-MHz INTEL XScale PXA270 with 64-MB memory, respectively. Table I shows the computing time (in the unit of second) of enlarging image "Lena" from the size of 400×400 to the size of 512×512 for the two processors. Obviously, our method requires less computing time than [7]–[9], and BC needs much longer time due to extensive computations.

To explore the performance of quantitative evaluation for image enlargement and reduction, first we scale the twelve 512×512 test images to the size of 400×400, 600×600, and 256×256, respectively, by using the bilinear method. Then, we scale up/down these images back to the size of 512×512 and show the results of peak signal-to-noise ratio (PSNR) in Tables II, III, and IV, respectively. Here, the output images of our scaling method are generated by the proposed VLSI circuit after post-layout transistor-level simulation. The output images of other scaling methods are all generated with software C programs. Simulation results show that our design achieves better quantitative quality than the previous low-complexity scaling methods [5]–[8]. However, the exact degree of improvement is dependent on the content of different images processed.

To explore the performance of visual quality, we enlarge image Syn_6 with different scaling methods. Fig. 16 shows the cropped regions of the original Syn_6 and reconstructed images. Obviously, NN produces noticeable blocks and BL blurs the scaled image, as shown in Fig. 16(b) and (c), respectively. As shown in Fig. 16(d), BC produces good image quality at the cost of high computational complexity. Both Win and M_Win are not good enough in regards to edge reservation, as shown in Fig. 16(e) and (f). The proposed design shows visually pleasing

TABLE II
COMPARISON OF PSNR FOR IMAGE ENLARGEMENT FROM THE SIZE OF 400 × 400 TO 512 × 512

	NN	BL	BC	Win	M_Win	Our
Crowd	32.04	34.86	36.80	34.86	34.81	35.53
Indian	31.36	33.21	34.58	33.29	33.28	33.88
Lake	30.61	32.49	33.87	32.51	32.46	33.07
Lena	32.54	34.48	35.95	34.58	34.54	35.20
Peppers	33.77	35.31	36.18	35.37	35.34	35.89
Plane	32.68	35.56	36.57	35.61	35.52	36.30
Syn_1	29.09	33.23	34.92	32.40	31.75	34.32
Syn_2	23.90	26.23	27.96	26.28	25.93	27.24
Syn_3	30.40	31.56	32.68	31.65	31.39	32.45
Syn_4	30.85	34.59	35.94	34.46	33.76	35.93
Syn_5	27.23	29.67	31.07	29.30	29.18	30.45
Syn_6	27.29	28.32	29.83	28.68	28.71	29.74
AVG.	30.15	32.46	33.86	32.41	32.22	33.33

TABLE III
COMPARISON OF PSNR FOR IMAGE REDUCTION FROM THE SIZE OF 600 × 600 TO 512 × 512

	NN	BL	BC	Win	M_Win	Our
Crowd	35.42	37.88	39.11	37.74	37.86	38.16
Indian	34.67	36.10	37.96	35.97	36.10	36.39
Lake	33.87	35.33	36.89	35.21	35.30	35.56
Lena	35.76	37.30	39.02	37.21	37.33	37.65
Peppers	36.92	37.88	39.53	37.76	37.85	38.11
Plane	36.22	38.62	39.95	38.51	38.64	38.94
Syn_1	38.56	38.06	39.81	38.01	38.08	39.08
Syn_2	27.22	29.42	30.88	29.28	30.03	30.21
Syn_3	35.28	35.62	37.13	35.51	35.83	36.58
Syn_4	34.46	37.57	39.01	37.47	37.28	38.50
Syn_5	33.41	34.59	36.04	34.42	35.25	35.29
Syn_6	29.18	30.84	32.13	30.73	31.47	31.72
AVG.	34.25	35.77	37.29	35.65	35.92	36.35

TABLE IV
COMPARISON OF PSNR FOR IMAGE ENLARGEMENT WITH THE INTEGER MAGNIFICATION FACTOR FROM THE SIZE OF 256 × 256 TO 512 × 512

	NN	BL	BC	Win	M_Win	Our
Crowd	28.64	31.29	32.49	28.64	28.64	31.30
Indian	27.85	29.99	31.01	27.85	27.85	30.14
Lake	27.07	29.58	30.54	27.07	27.07	29.62
Lena	29.26	31.31	32.03	29.26	29.26	31.47
Peppers	30.24	32.71	32.92	30.24	30.24	32.73
Plane	29.47	31.80	32.12	29.47	29.47	31.85
Syn_1	30.37	31.03	31.25	30.37	30.37	31.91
Syn_2	20.22	22.65	23.78	20.22	20.22	23.01
Syn_3	26.88	29.12	29.35	26.88	26.88	29.78
Syn_4	27.81	31.00	31.91	27.81	27.81	31.53
Syn_5	25.43	26.71	27.08	25.43	25.43	26.90
Syn_6	23.26	24.64	25.18	23.26	23.26	24.66
AVG.	27.21	29.32	29.97	27.21	27.21	29.57

picture for the reference image especially at its edges, as shown in Fig. 16(g).

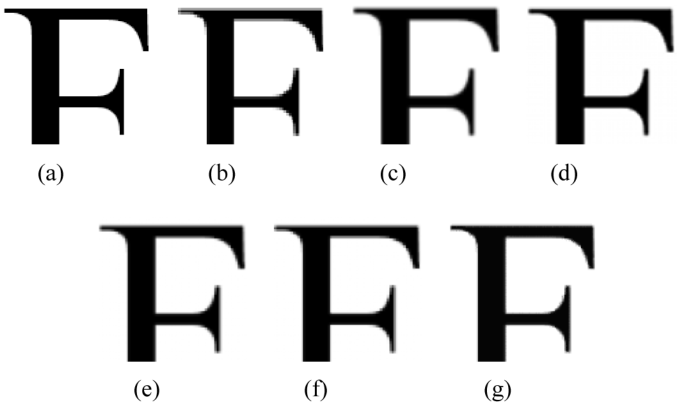

Fig. 16. Visual results of different scaling techniques for Syn_6. (a) Original. (b) NN. (c) BL. (d) BC. (e) Win. (f) M_Win. (g) Our.

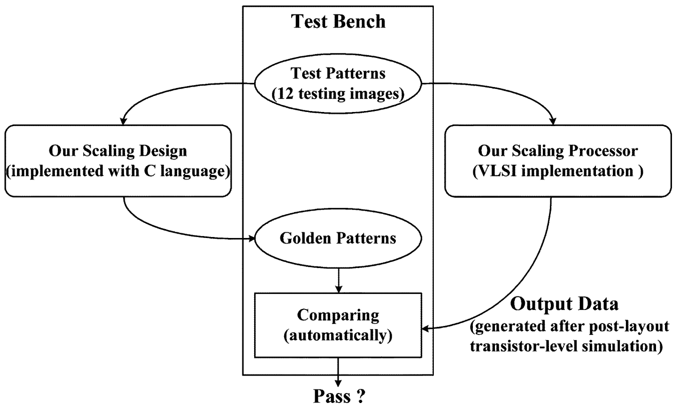
Test Bench
Test Patterns (12 testing images)
Our Scaling Design (implemented with C language)
Our Scaling Processor (VLSI implementation)
Golden Patterns
Comparing (automatically)
Output Data (generated after post-layout transistor-level simulation)
Pass ?

Fig. 17. Model used to verify our VLSI implementation.

The proposed VLSI architecture of the proposed design was implemented by using Verilog HDL. We used SYNOPSYS Design Vision to synthesize the design with TSMC's 0.18-μm cell library. The layout for the design was generated with SYNOPSYS Astro (for auto placement and routing), and verified by MENTOR GRAPHIC Calibre (for DRC and LVS checks), respectively. Nanosim was used for post-layout transistor-level simulation. Finally, SYNOPSYS PrimePower was employed to measure the total power consumption. Synthesis results show that the scaling processor contains 10.4-K gate counts and its core size is about 532 × 521 μm^2. It works with a clock period of 5 ns and can achieve a processing rate of 200 megapixels/second which is quick enough to process a video resolution of WQSXGA (3200 × 2048) at 30 f/s in real time. The power consumption is 16.47 mW with 1.8-V supply voltage.

Fig. 17 shows the model used to verify our VLSI implementation. The 12 testing images are used as the input test patterns. Using those test patterns, our scaling design implemented with the C program can generate the corresponding scaled images, denoted as the golden patterns. Finally, the golden patterns are automatically compared with the output data generated by our scaling circuit after post-layout transistor-level simulation.

Furthermore, the architecture was also implemented on the Altera Cyclone II EP2C8F256C6 FPGA platform for verification. The real images generated by the Altera implementation can be used for visual observation. The cropped region of

TABLE V
FEATURES OF FOUR SCALING IMPLEMENTATIONS

	Win [7]	M_Win [8]	HABI [14] (Bicubic)	Our		
FPGA device / ASIC Library	NA	NA	Xilinx Virtex-II Pro	Xilinx Virtex-II Pro XC2VP50	Altera CycloneII EP2C8F256C6	TSMC's 0.18µm process
Line Buffer	1	1	6	1	1	1
Area	29K gate counts	NA	890 CLBs	581 CLBs	1.06K logic elements	10.4K gate counts
Max Frequency	65 MHz	55 MHz	100 MHz	142 MHz	109 MHz	200 MHz
Computation Time	4.74 ms	5.60 ms	3.50 ms	2.17 ms	2.82 ms	1.54 ms

scaled image for our method in Fig. 16(g) is obtained with the Altera implementation of the proposed architecture. The operating clock frequency of Altera implementation is 109 MHz with 1.06-K logic elements and its power consumption is 173.75 mW.

Table V shows the comparing results of the three area-pixel scaling chips and the higher complexity bicubic scaling chip (HABI [14]). For easy comparison, the proposed processor is implemented with ASIC, Altera, and Xilinx, respectively. In the last row of Table V, the computation time of those implementations is obtained by scaling up the testing image from size 512×384 to size 640×480. Obviously, our chip requires less hardware cost and works faster than other chips.

VI. Conclusion

A low-cost image scaling processor is proposed in this paper. The experimental results demonstrate that our design achieves better performances in both objective and subjective image quality than other low-complexity scaling methods. Furthermore, an efficient VLSI architecture for the proposed method is presented. In our simulation, it operates with a clock period of 5 ns and achieves a processing rate of 200 megapixels/second. The architecture works with monochromatic images, but it can be extended for working with RGB color images easily.

References

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Reading, MA: Addison-Wesley, 1992.

[2] W. K. Pratt, *Digital Image Processing*. New York: Wiley-Interscience, 1991.

[3] T. M. Lehmann, C. Gonner, and K. Spitzer, "Survey: Interpolation methods in medical image processing," *IEEE Trans. Med. Imag.*, vol. 18, no. 11, pp. 1049–1075, Nov. 1999.

[4] C. Weerasnghe, M. Nilsson, S. Lichman, and I. Kharitonenko, "Digital zoom camera with image sharpening and suppression," *IEEE Trans. Consumer Electron.*, vol. 50, no. 3, pp. 777–786, Aug. 2004.

[5] S. Fifman, "Digital rectification of ERTS multispectral imagery," in *Proc. Significant Results Obtained from Earth Resources Technology Satellite-1*, 1973, vol. 1, pp. 1131–1142.

[6] J. A. Parker, R. V. Kenyon, and D. E. Troxel, "Comparison of interpolation methods for image resampling," *IEEE Trans. Med. Imag.*, vol. MI-2, no. 3, pp. 31–39, Sep. 1983.

[7] C. Kim, S. M. Seong, J. A. Lee, and L. S. Kim, "Winscale: An image scaling algorithm using an area pixel model," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 6, pp. 549–553, Jun. 2003.

[8] I. Andreadis and A. Amanatiadis, "Digital image scaling," in *Proc. IEEE Instrum. Meas. Technol. Conf.*, May 2005, vol. 3, pp. 2028–2032.

[9] H. S. Hou and H. C. Andrews, "Cubic splines for image interpolation and digital filtering," *IEEE Trans. Acoust. Speech Signal Process.*, vol. ASSP-26, no. 6, pp. 508–517, Dec. 1978.

[10] J. K. Han and S. U. Baek, "Parametric cubic convolution scalar for enlargement and reduction of image," *IEEE Trans. Consumer Electron.*, vol. 46, no. 2, pp. 247–256, May 2000.

[11] L. J. Wang, W. S. Hsieh, and T. K. Truong, "A fast computation of 2-D cubic-spline interpolation," *IEEE Signal Process. Lett.*, vol. 11, no. 9, pp. 768–771, Sep. 2004.

[12] H. A. Aly and E. Dubois, "Image up-sampling using total-variation regularization with a new observation model," *IEEE Trans. Image Process.*, vol. 14, no. 10, pp. 1647–1659, Oct. 2005.

[13] T. Feng, W. L. Xie, and L. X. Yang, "An architecture and implementation of image scaling conversion," in *Proc. IEEE Int. Conf. Appl. Specific Integr. Circuits*, 2001, pp. 409–410.

[14] M. A. Nuno-Maganda and M. O. Arias-Estrada, "Real-time FPGA-based architecture for bicubic interpolation: An application for digital image scaling," in *Proc. IEEE Int. Conf. Reconfigurable Computing FPGAs*, 2005, pp. 8–11.

[15] G. Ramponi, "Warped distance for space-variant linear image interpolation," *IEEE Trans. Image Process.*, vol. 8, no. 5, pp. 629–639, May 1999.

Pei-Yin Chen (M'08) received the B.S. degree in electrical engineering from National Cheng Kung University, Tainan, Taiwan, in 1986, the M.S. degree in electrical engineering from Pennsylvania State University, State College, in 1990, and the Ph.D. degree in electrical engineering from National Cheng Kung University, in 1999.

Currently, he is an Associate Professor in the Department of Computer Science and Information Engineering, National Cheng Kung University. His research interests include VLSI chip design, video compression, fuzzy logic control, and gray prediction.

Chih-Yuan Lien received the B.S. and M.S. degrees in computer science and information engineering from National Taiwan University, Tainan, Taiwan, in 1996 and 1998, respectively. Currently, he is working towards the Ph.D. degree in the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan.

His research interests include image processing, video compression, and VLSI chip design.

Chi-Pin Lu received the B.S. degree in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2005 and the M.S. degree in computer science and information engineering from National Cheng Kung University, Tainan, Taiwan, in 2007.

His research interests include VLSI chip design and video compression.