Real-time realisation of noise-immune gradient-based edge detector

P.-Y. Hsiao, C.-H. Chen, H. Wen and S.-J. Chen

Abstract: A computational field-programmable gate array (FPGA) realisation for edge detection that is particularly immune to noise by a digital approximated Gaussian smoothing filter is described. The proposed systolic array architecture was examined for convolution operation in order to put simplicity and regularity to the design. Moreover, most of the presented processing structures are highly pipelined, so that the goal of real-time computing is substantially achieved with the processing frame rate reaching up to 280 frames per second. For an efficient hardware mapping, the absolute difference mask algorithm was adopted because of its regularity and independent operations, as well as its important property of performing one-pixel-edge localisation. A scalable first in, first out (FIFO) design was also proposed to make the edge detector applicable to five different image sizes. The FPGA realisation on the presented versatile development platform shows that the proposed design improves both the speed and the hardware usage. This is attributed to the utilisation of the proposed parallel and pipelined structure so that a fast operating speed of 73.6 MHz, which is about 265 times faster than the digital signal processing environment, is obtained in the present investigation.

1 Introduction

Edge detection plays a key role in the widespread application of image processing such as in computer vision [1–4], pattern analysis [5–7] and so on. Basically, edges can be identified as the locations of abrupt discontinuity in the grey level of an image [1, 5, 8]. Although edge detection is mostly employed in the pre-processing step of image processing, the resulting quality of edges may seriously affect the performance of the remaining steps. As described in the literature on digital image processing, edge detection can be done in both spatial and frequency domains [8–11], wherein the spatial domain filtering shows more versatility and feasibility.

As the computation speed of application-specific integrated circuits (ICs) and general-purpose processors has grown exponentially in recent years, more and more studies have been devoted to optimising and speeding-up those computationally-complex applications. Gradient image acquisition is the usual method for edge detection. Indeed, it performs quite well in spite of the fact that the operations are much simpler than those of advanced algorithms like the Canny edge detector [10]. The most widely used edge detectors in low-level edge detection are the 3×3 masks such as those of Sobel and Prewitt wherein they extract the first-order derivatives of an image. However, a review of the literature reveals that all of these methods start from a mathematical manipulation before a software-oriented algorithm is developed. Afterwards, the hardware realisation is eventually tackled. On the other hand, the adopted low-computational complexity of the absolute difference mask (ADM) algorithm [9] is especially suitable for hardware realisation.

With the advantages of rapid prototyping and ease of verification, using the field-programmable gate array (FPGA) is a good choice for realising image processing hardware design. Several FPGA and/or very large scale integration (VLSI) usages of image processing realisations have been presented in recent years [3, 4, 6, 7, 11–18]. In the present paper, we employ a systolic array structure to optimise the design that has proven to be highly regular and efficient for mask operation [12].

2 Noise reduction strategies

2.1 Spatial filtering

For most applications, image processing tasks are applied to real world images. As the source images are usually retrieved from the complementary metal-oxide-semiconductor (CMOS) or charge-coupled device (CCD) image acquisition devices, such as 1D scanners or 2D digital still cameras, noise signal is easily introduced into the image during the acquisition stage and an additional step is required to eliminate it. These noise signals can be visually recognised as dots in the images that appear in a distinctive greyscale value from the quantised number point-of-view.

The technique of noise reduction usually involves the averaging of all the values within the local area using various sizes of masks. As expected, a larger-sized averaging mask has a better noise reduction capability because more pixels contribute their values to the target pixel. After a local averaging, an image would look blurred or smoothed. This is the reason why the noise reduction process is usually called the smoothing process. The most commonly used averaging mask is shown in Fig. 1*a*. However, images processed by this mask become quite blurred and a lot of detailed information contained in the

IEE Proceedings online no. 20050199

doi:10.1049/ip-cdt:20050199

Paper first received 25th November 2005 and in revised form 28th February 2006

P.-Y. Hsiao and H. Wen are with the Department of Electronic Engineering, Chang Gung University, 259 Wen-Haw 1st Road, Kwei-Shan Tao-Yuan 333, Taiwan, Republic of China

C.-H. Chen and S.-J. Chen are with the Department of Electrical Engineering, National Taiwan University, 1 Roosevelt Rd. Sec. 4, Taipei 106, Taiwan, Republic of China

E-mail: pyhsiao@mail.cgu.edu.tw

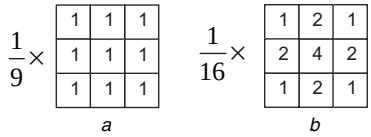

Fig. 1 *Averaging masks for smoothing*
a Common averaging mask
b Weighted averaging mask

original image can be lost. A more practical averaging mask is the weighted average mask that gives more emphasis on the centre pixel. The weights are larger in the central area and decrease in the outer positions. An example of the weighted average mask is illustrated in Fig. 1*b*.

2.2 Gaussian mask

In the present study, we chose the weighted averaging mask for noise reduction, particularly the 2D Gaussian function for coefficient selection. An example of the distribution of the 2D Gaussian function is shown in Fig. 2 with a standard deviation $\sigma = 0.9$.

The Gaussian function is a good model for weighted averaging mask. With different standard deviations σ, we obtain various weighted masks with different decay rates and smoothing capabilities. In the present design, we used a 5×5 smoothing mask for better performance and feasibility. The coefficients of the mask are generated by substituting x and y with -2, -1, 0, 1, 2 in the 2D Gaussian equation. Fig. 3 shows the results of the Gaussian mask at $\sigma = 0.9$ and 1.2.

2.3 Gaussian 2^n approximation

Our goal is to design a hardware smoothing unit for noise reduction with simple and nearly optimal smoothing filters. The aforementioned masks shown in Fig. 3 have coefficients that are difficult to apply on hardware because they are all floating-point numbers. However, we can still approximate those values using one or more power-of-two terms that can be effectively applied in digital circuits.

To generate the desired 2^n-approximated mask, we first define λ as the number of terms used to approximate the original coefficients and we let λ be either 2 or 3. Thus, two power-of-two terms will be used if $\lambda = 2$ or three terms at the most if otherwise. The list of the decimal

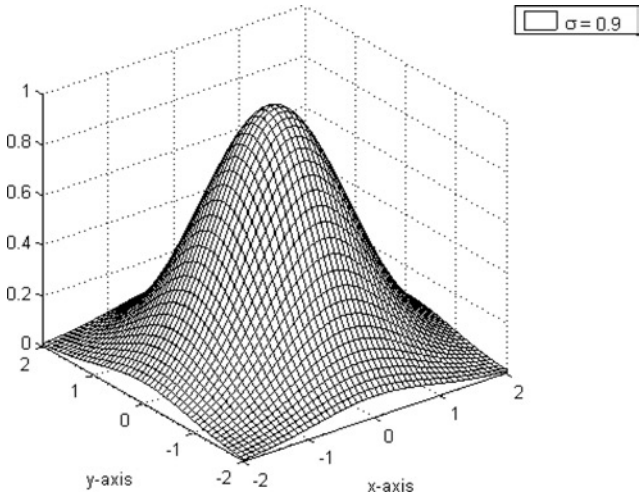

Fig. 2 *Distribution of 2D Gaussian function*

0.001418	0.009035	0.016750	0.009035	0.001418
0.009035	0.057569	0.106727	0.057569	0.009035
0.016750	0.106727	0.197859	0.106727	0.016750
0.009035	0.057569	0.106727	0.057569	0.009035
0.001418	0.009035	0.016750	0.009035	0.001418

a

0.007332	0.020779	0.029406	0.020779	0.007332
0.020779	0.058888	0.083334	0.058888	0.020779
0.029406	0.083334	0.117928	0.083334	0.029406
0.020779	0.058888	0.083334	0.058888	0.020779
0.007332	0.020779	0.029406	0.020779	0.007332

b

Fig. 3 *Two examples of 5×5 Gaussian masks*
a $\sigma = 0.9$
b $\sigma = 1.2$

values of power-of-two terms with the power range from 0 to -7 is as follows:

$$2^0 = 1.000000 \qquad 2^{-1} = 0.500000$$
$$2^{-2} = 0.250000 \qquad 2^{-3} = 0.125000$$
$$2^{-4} = 0.062500 \qquad 2^{-5} = 0.031250$$
$$2^{-6} = 0.015625 \qquad 2^{-7} = 0.007813$$

Then, we develop a heuristic algorithm to generate the 2^n-approximated Gaussian masks. First we approximate each of the 5×5 elements by successively passing each of the power-of-two terms listed earlier. If the remaining value of the element is larger or equal to a term, we subtract that value from that term. This operation is repeated for this element until either the number of terms, λ, for the element is reached, or the remaining value is less than 2^{-7}. This method is applied to all the 5×5 elements. Note that, if the original coefficient is smaller than 2^{-7}, 0 will be used to approximate the value.

The normalisation term for each mask is estimated by first summing up all of the twenty-five 2^n-approximated values. Then the 2^n approximation for this normalisation term is determined using the same method for the 5×5 elements, except that the range of the power-of-two term is extended to 2^{-15} instead of 2^{-7}. Two examples of the 2^n-approximated Gaussian masks for $\sigma = 0.9$, $\lambda = 3$ and for $\sigma = 1.2$, $\lambda = 2$ are illustrated in Fig. 4.

To describe the errors between the original Gaussian mask and the 2^n-approximated Gaussian mask in a formal way, we define each of the normalised elements in the original Gaussian mask as

$$G_\sigma(i,j)|_{i,j=1\text{–}5,\sigma=0.1\text{–}5}$$

We can also define each of the approximated normalised elements in the 2^n-approximated Gaussian mask as

$$A_{\sigma\lambda}(i,j)|_{i,j=1-5,\sigma=0.1-5,\lambda=2\text{ or }3}$$

Consequently, the absolute accumulated error for the 2^n-approximated Gaussian mask is defined as

$$E_{\sigma\lambda} = \sum_{i=1}^{5}\sum_{j=1}^{5} |G_\sigma(i,j) - A_{\sigma\lambda}(i,j)|$$

The decision for the acceptable accumulated error affects the algorithm we adopted to approximate the Gaussian mask. Fig. 5 shows the distribution of the accumulated errors for $\lambda = 2$ and $\lambda = 3$ 2^n-approximation with respect to the standard deviation from 0.1 to 5.0. On the basis

$(2^{-3}+2^{-8}+2^{-9})\times$

2^{-4}	2^{-3}	2^{-4}	2^{-4}	0
2^{-4}	$2^{-2}+2^{-3}+2^{-4}$	$2^{-1}+2^{-2}+2^{-4}$	$2^{-2}+2^{-3}+2^{-4}$	2^{-4}
2^{-3}	$2^{-1}+2^{-2}+2^{-4}$	$2^{0}+2^{-1}+2^{-4}$	$2^{-1}+2^{-2}+2^{-4}$	2^{-3}
2^{-4}	$2^{-2}+2^{-3}+2^{-4}$	$2^{-1}+2^{-2}+2^{-4}$	$2^{-2}+2^{-3}+2^{-4}$	2^{-4}
0	2^{-4}	2^{-3}	2^{-4}	0

a

$(2^{-4}+2^{-7})\times$

0	2^{-2}	$2^{-2}+2^{-3}$	2^{-2}	0
2^{-2}	$2^{-1}+2^{-2}$	$2^{0}+2^{-2}$	$2^{-1}+2^{-2}$	2^{-2}
$2^{-2}+2^{-3}$	$2^{0}+2^{-2}$	$2^{0}+2^{-1}$	$2^{0}+2^{-2}$	$2^{-2}+2^{-3}$
2^{-2}	$2^{-1}+2^{-2}$	$2^{0}+2^{-2}$	$2^{-1}+2^{-2}$	2^{-2}
0	2^{-2}	$2^{-2}+2^{-3}$	2^{-2}	0

b

Fig. 4 *Two examples of 2^n-approximated 5×5 Gaussian masks*
a $\sigma = 0.9$, $\lambda = 3$
b $\sigma = 1.2$, $\lambda = 2$

of empirical experiences, we chose the acceptable accumulated error for σ to be within [0, 0.15].

3 Gradient-based ADM algorithm

The edge detection algorithm, proposed by Alzahrani and Chen [9], presents a regular computational structure that is suitable for a VLSI implementation and integration, and is divided into the three stages shown in Fig. 6. The fact that the algorithm can be designed in a highly pipelined fashion helps to produce one single-pixel localised edge per clock cycle. The first stage is dedicated to noise removal so that the features can be correctly extracted. The second stage involves the determination of the edge strength and direction of each pixel. Finally, the edge detection and localisation are performed, and the single-pixel edge map is produced for further processing in the last stage. In the present paper, we propose an efficient pipelined structure to accomplish the aforementioned three-staged algorithm, which is different from the tree-based architecture that appeared in the previous implementation [9].

In Fig. 6, the Gaussian smoothing filter is employed to perform the image smoothing of the first stage. Here we

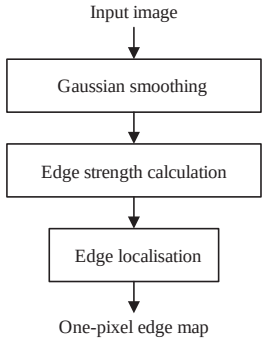

Fig. 6 *Processing stages*

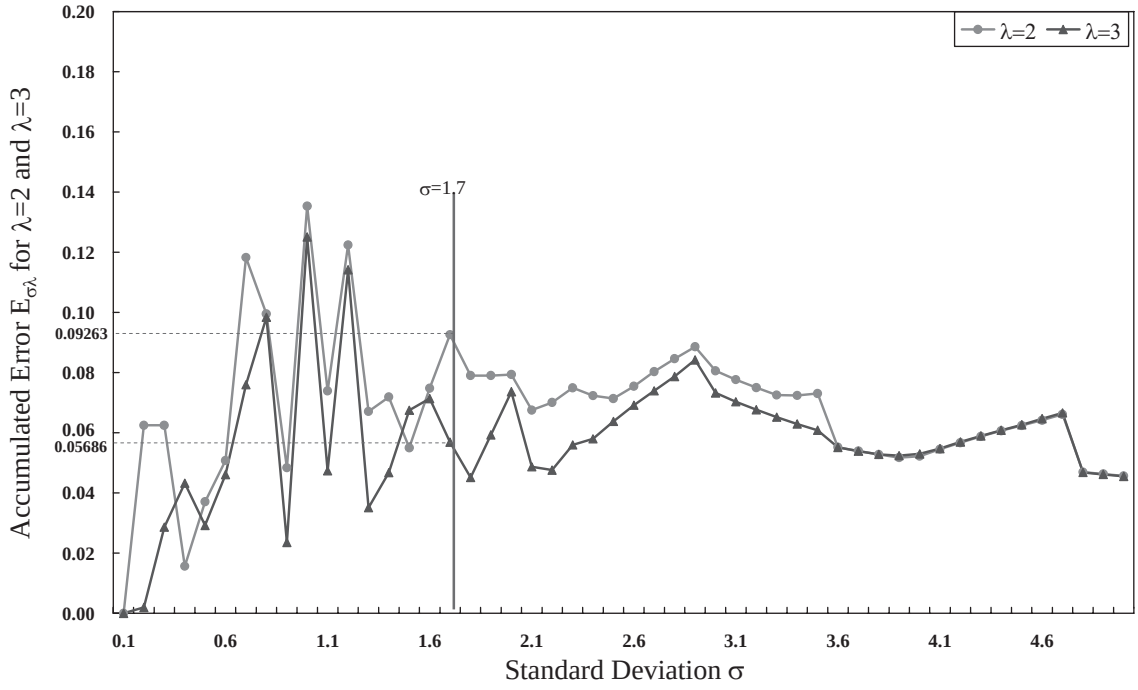

Fig. 5 *Accumulated error for $\lambda = 2, \lambda = 3$ with respect to standard deviation σ*

adopted our 2^n-approximated Gaussian mask instead of the semi-Gaussian mask proposed by Alzahrani and Chen [9]. The semi-Gaussian mask is a 5×5 digitally approximated smoothing filter that is simpler than our proposed 2^n-approximated Gaussian mask in Section 2, but has a much higher accumulated error. It actually averages all the grey values of the pixels in the 5×5 area with decaying weights. The coefficients of this mask are selected to be in the form of $1, 3/4, 1/2, 1/4$ in order that the operations can be performed in the hardware by merely adding and bit-shifting operations. Fig. 7 illustrates the semi-Gaussian smoothing mask proposed by Alzahrani and Chen [9]. Note that in Fig. 7, all elements are divided by two and the normalisation coefficient is multiplied by two for an easy comparison between the semi-Gaussian mask and our 2^n-approximated Gaussian masks.

In the work of Alzahrani and Chen [9], we did not find any information on which σ to use in order to generate the semi-Gaussian mask. Thus, we conducted a simple experiment to determine it. The semi-Gaussian mask was passed to all the real Gaussian masks with σ in the range of [0.1, 5]. The absolute-accumulated-error curve obtained between the semi-Gaussian mask and each of the σ is presented in Fig. 8. From the results, we could predict that the original mask [9] was generated at $\sigma = 1.7$, as it gave the smallest error.

From Figs. 5 and 8, we can see that under the same standard deviation of $\sigma = 1.7$, the proposed 2^n-approximated Gaussian mask has a better precision over the semi-Gaussian mask, regardless of the terms employed. For a more precise Gaussian-approximated smoothing mask, we adopted our $\sigma = 1.7$, $\lambda = 2$ 2^n-approximated mask in the present paper as shown in Fig. 9.

The second stage shown in Fig. 6 calculates the edge strength and direction of each pixel. After finding the four absolute differences, the ADM algorithm chooses the maximum value as the edge strength, and the direction corresponding to the minimum value as the edge direction. Note that for each direction, the difference is calculated using four pixels instead of two in order to obtain better precision.

Once the edge strength is determined, a local maximum is calculated, that is one pixel is compared with two neighbouring pixels in the edge direction. The maximum

$2^{-3} \times$

2^{-3}	2^{-2}	2^{-2}	2^{-2}	2^{-3}
2^{-2}	$2^{-2}+2^{-3}$	2^{-1}	$2^{-2}+2^{-3}$	2^{-2}
2^{-2}	2^{-1}	1	2^{-1}	2^{-2}
2^{-2}	$2^{-2}+2^{-3}$	2^{-1}	$2^{-2}+2^{-3}$	2^{-2}
2^{-3}	2^{-2}	2^{-2}	2^{-2}	2^{-3}

Fig. 7 *Semi-Gaussian smoothing mask proposed by Alzahrani and Chen [3]*

$2^{-4} \times$

2^{-2}	$2^{-2}+2^{-3}$	2^{-1}	$2^{-2}+2^{-3}$	2^{-2}
$2^{-2}+2^{-3}$	$2^{-1}+2^{-2}$	$2^{-1}+2^{-2}$	$2^{-1}+2^{-2}$	$2^{-2}+2^{-3}$
2^{-1}	$2^{-1}+2^{-2}$	$2^{0}+2^{-3}$	$2^{-1}+2^{-2}$	2^{-1}
$2^{-2}+2^{-3}$	$2^{-1}+2^{-2}$	$2^{-1}+2^{-2}$	$2^{-1}+2^{-2}$	$2^{-2}+2^{-3}$
2^{-2}	$2^{-2}+2^{-3}$	2^{-1}	$2^{-2}+2^{-3}$	2^{-2}

Fig. 9 *Proposed 2^n-approximated Gaussian smoothing mask for $\lambda = 2$, $\sigma = 1.7$*

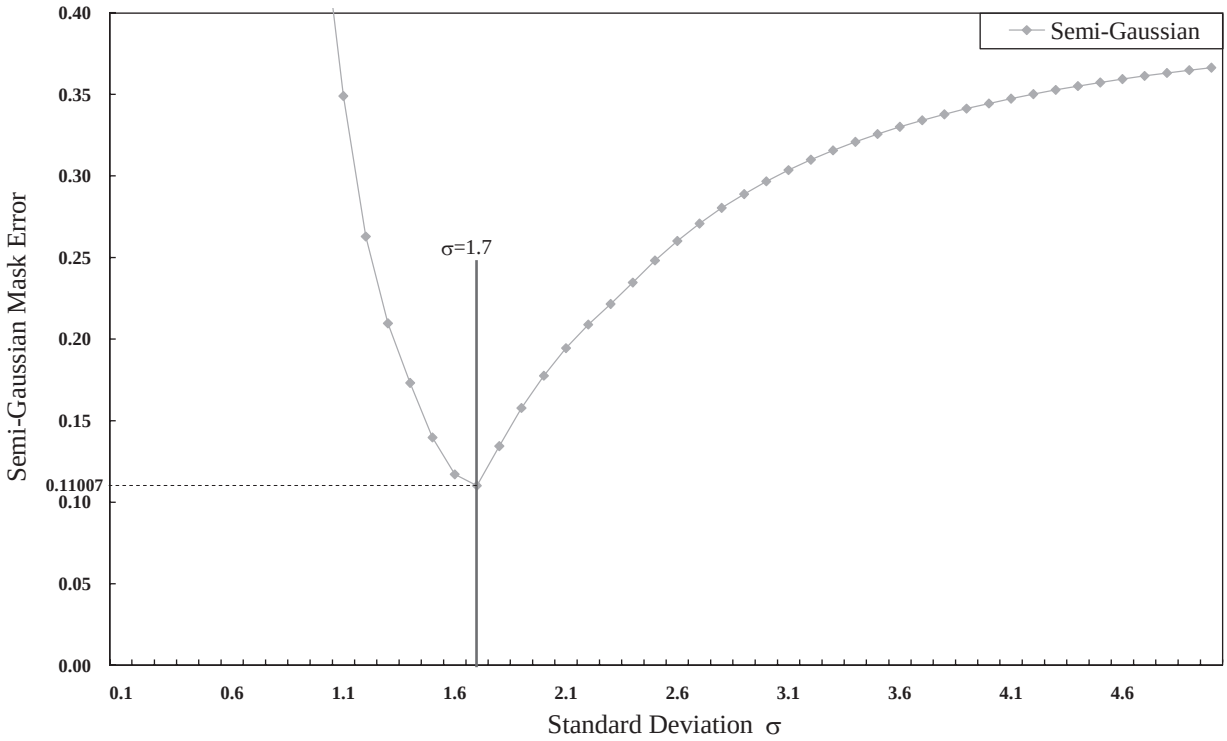

Fig. 8 *Semi-Gaussian mask error with respect to standard deviation σ*

Note that the semi-Gaussian error for $\sigma < 1.0$ is way over 0.4 and thus omitted here

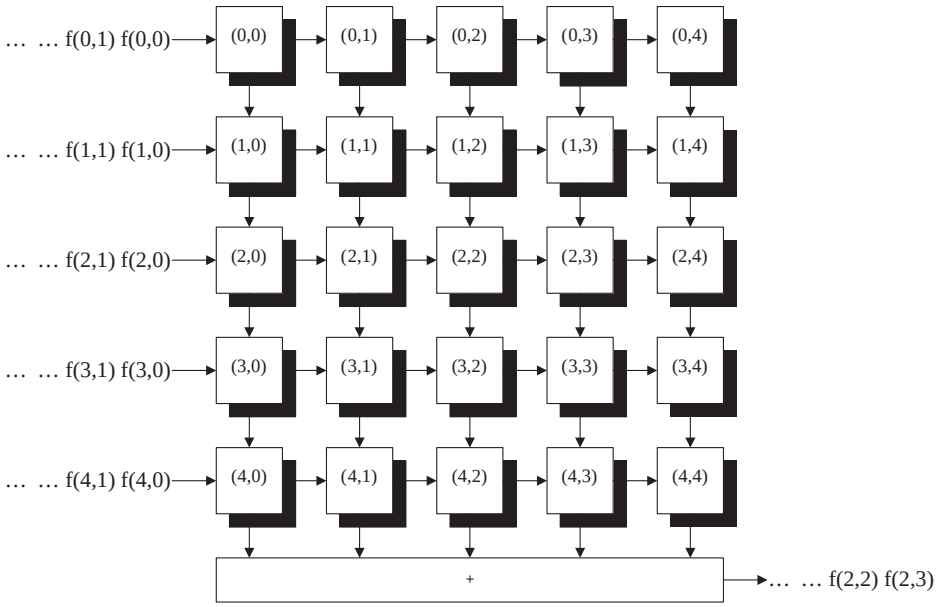

Fig. 10 *Proposed 5×5 systolic array for the smoothing unit*

greylevel pixel in the 3 × 3 region goes through the final thresholding step. Thus, those pixels that are greater than the threshold are identified as edge points.

4 Pipelined architecture and timing analysis

In addition to two of our schemes for a scalable first in first out (FIFO) design, the proposed architecture is divided into three function units that correspond to the three stages described in Section 3. As shown later on, the major modifications and improvements over the original design are Scheme 2 scalable FIFO design, the more precise Gaussian smoothing unit and the edge localisation unit. The scalable FIFO design enables the possibility of using the same hardware resource to process images with five different sizes. With the proposed optimised architecture, not only is the hardware usage reduced, but also the path delay is considerably improved.

4.1 Systolic Gaussian smoothing unit

As the convolution is the sum of the products of each pixel and the corresponding mask coefficient, the arithmetic operation can be executed independently. Thus, the parallel architecture of a 5 × 5 systolic array, as shown in Fig. 10, is presented. Each processing element (PE) in this array should be responsible for the pixel-weighting multiplication and the addition with the preceding stage PE. As multiplication requires a large hardware resource and longer execution time, the proposed architecture rearranges all the coefficients so that it can be implemented simply by bit-shifting. Fig. 10 illustrates the data flow in the systolic array and Fig. 11 unveils the inner structures of the PEs. Note that we divided the PEs into four categories according to their positions in the array. Their structures are slightly different to enable hardware resources to be saved.

For example, Fig. 11*c* is the general form of the PE that is used in the lower left 4 × 4 area shown in Fig. 10. As the horizontal dashed line is elaborated, the PE operates in a pipelined manner in order to meet the timing demand for the systolic array to ensure the correctness of the function. Fig. 11*a* shows the structure of the upper four PEs as depicted in Fig. 10. This structure is a simplified version of the general PE wherein the data path of the preceding row input is removed. The structure shown in Fig. 11*b* is also derived from the general PE and is mounted on the upper-right corner of the systolic array. The data path of the input from the preceding row and that of the output in the next column are all eliminated. Finally, Fig. 11*d* shows the structure of the four right PEs with the data path of output in the next column removed. Another important component is the delay unit for the control of incoming data. Five pixel data obtained from the FIFO should be delayed by one more clock cycle than the previous row.

4.2 Edge strength unit and edge location unit

Edge strength and direction are determined by comparing four absolute differences. To keep this unit functioning correctly, the most intuitive way of designing it is to form a

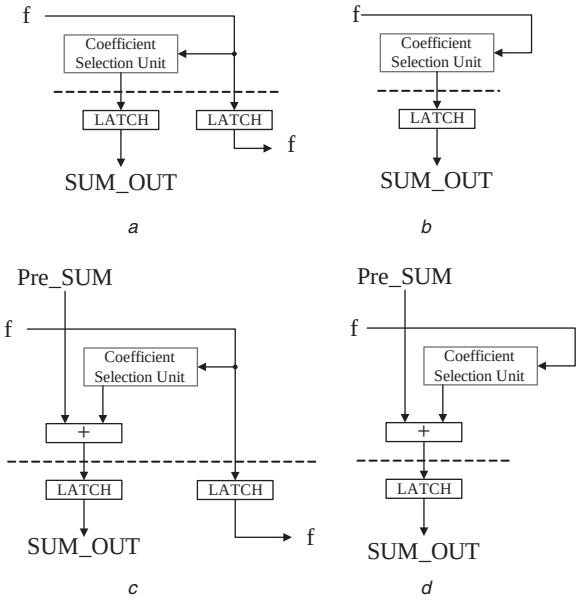

Fig. 11 *PE structures in four different positions*

a Upper
b Upper-right
c General
d Right

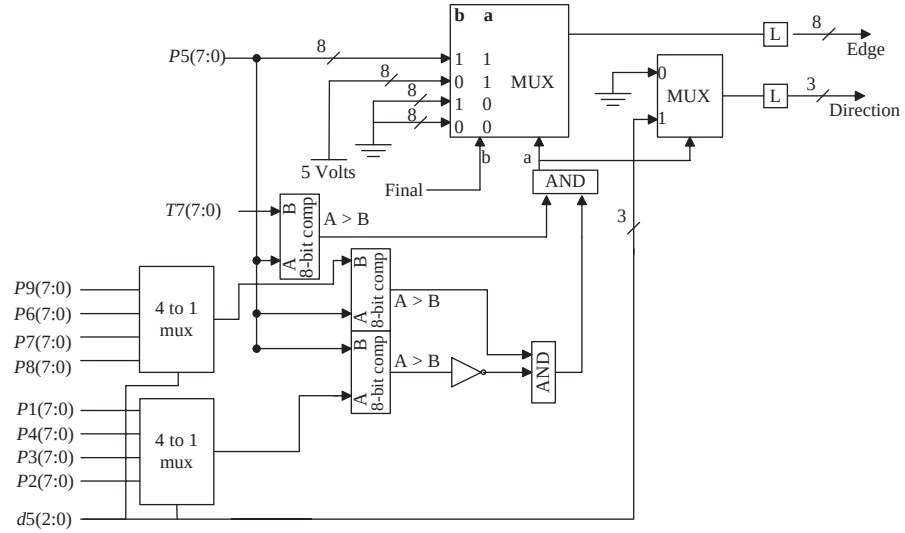

Fig. 12 *Optimised structure of the edge localisation unit*

parallel-pipeline so that 16 inputs, 8 additions, 4 absolute difference calculations and comparisons can be completed in 4 clock cycles. Here we basically keep the original design of the unit intact.

The edge location unit performs the one-pixel edge localisation by comparing the edge strength values of the centre pixel with that of the other two edge pixels in the derived edge direction. In the original design, there were nine

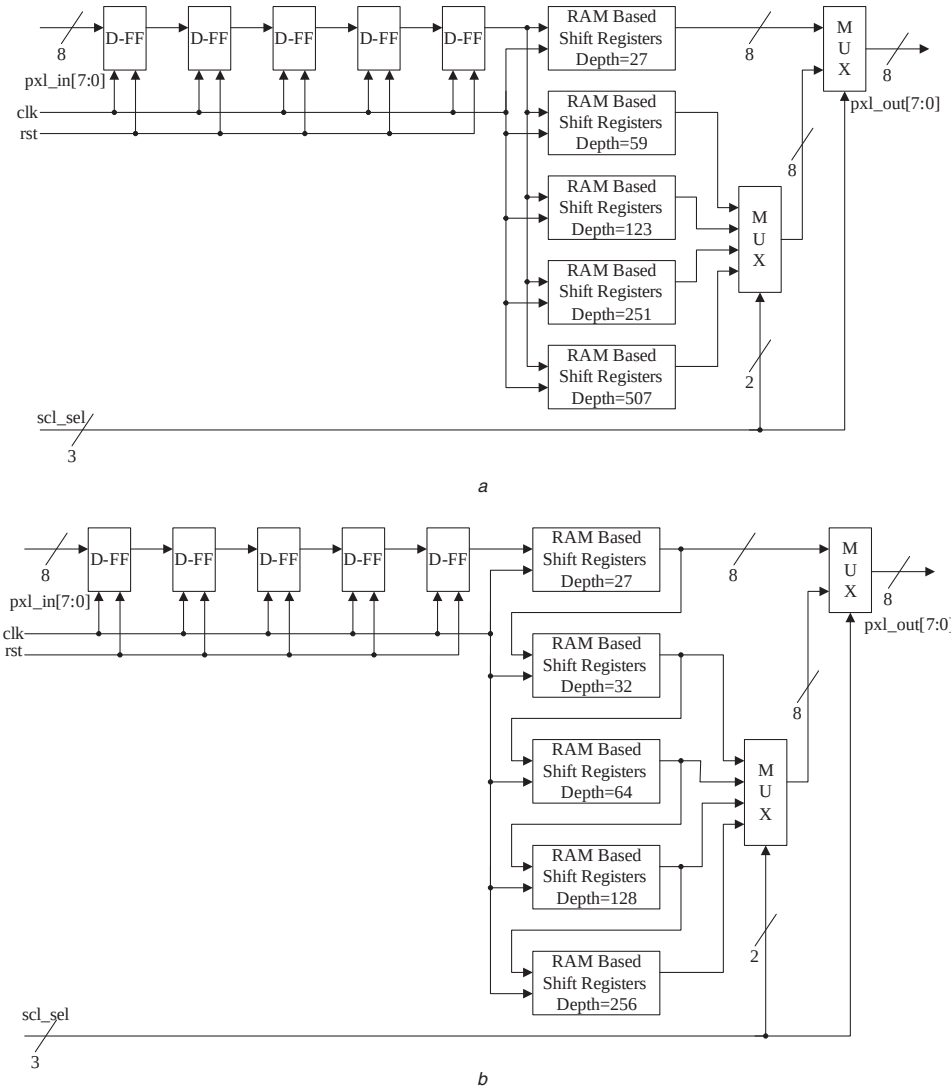

Fig. 13 *Scalable FIFO*

a Scheme 1
b Scheme 2

Table 1: Hardware usage of scalable FIFO schemes

	Fixed size (512 × 512)	Scalable Scheme 1	Scheme 2
Hardware usage (gate counts)	132 160	258 968	137 368

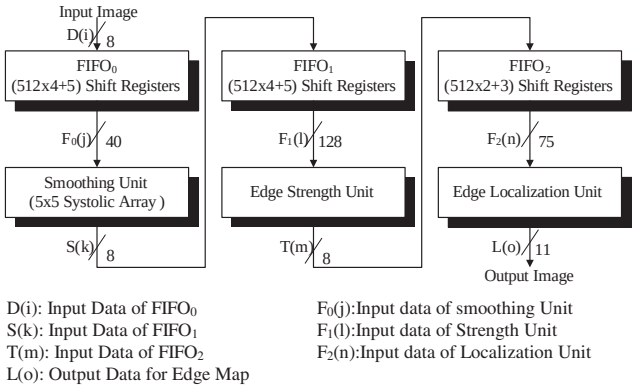

Fig. 14 *Pipelined computational structure*

comparators used for edge localisation, but only three comparators are used in the present work. The reason behind this is that we first multiplex four input pairs, which are controlled by the edge direction, ahead of the comparison rather than doing it afterwards. It is noted that about 66% of the hardware resources are saved in this unit compared with the design of Alzahrani and Chen. The optimised structure is shown in Fig. 12, wherein the 'Final' control signal decides the edge map output as a binary image while 'Final' = 0, or as the original grey values of the edge pixels while 'Final' = 1.

4.3 Scalable FIFO design

As the size of the FIFO depends on the size of the input image, it is a good idea to design a FIFO that is applicable to various image sizes. With a run-time reconfiguration capability, the hardware is capable of handling images of different sizes without service downtime. Take the FPGA-based hardware design as an example. In order to adjust the design so that the hardware can perform image processing tasks of different image sizes, a code modification and a full recompilation to generate the FPGA ROM file are required. The whole process is time-consuming and the application-specific integrated circuit (ASIC) digital tape-out flow is even more costly. For the more recent and advanced FPGA technology, to reconfigure hardware still requires some microcode reconfiguration that can take fewer service downtimes, but not very many FPGAs support this feature.

The common sizes of input images are usually powers of two. Thus, we developed a scalable FIFO that can be used for five different sizes, namely, 32, 64, 128, 256 and 512 bits. However, the way we developed this flexible structure determines the hardware usage of the whole design and thus, we had to find out the best structure with minimal hardware usage. Below, we propose two schemes to implement a scalable FIFO and we prove that Scheme 2 is the most efficient way to complete the task.

Fig. 13*a* is the straightforward Scheme 1 to complete a scalable FIFO design where five shift register components in different sizes are multiplexed at the output. Although it is a simple and straightforward design, the huge costs of the FPGA resources make it impractical.

The other idea of designing the scalable FIFO is shown in Fig. 13*b*. Obviously, Scheme 2 is a lot better than the original Scheme 1. Owing to the consecutive property of the shift register, the row shift register can be cut into a few segments. We may bypass the registers in the different lengths and output the data. Thus, scalable FIFO in five different sizes can be obtained efficiently. Table 1 gives the detailed information about the hardware usage of the two scalable FIFO schemes. It is noted that Scheme 2 reveals a scalable FIFO in five different sizes with only 3.8% increment in the hardware usage with respect to a fixed size of 512-bit wide FIFO, whereas Scheme 1 practically doubles the usage.

4.4 Timing analysis

All of the aforementioned proposed function units work in a pipelined manner such that, with a certain amount of latency, the edge pixel is efficiently generated at every clock cycle. The whole picture of the pipelined design is shown in Fig. 14.

For a successful pipeline scheme, we have to be concerned not only with the inside of each function unit, but also with the external parallel execution schedule for the overall architecture [6, 8, 9, 11]. As shown in Table 2, the latency of the whole architecture is $L_0 + L_S + L_1 + L_T + L_2 + L_L = 2583$ cycles. It means that we are able to obtain the first pixel of the output image (edge map) after 2583 cycles. On the other hand, this structure is suitable for images of all sizes, which is within a 512-pixel width while needing only to adjust the width of the FIFO to as large as the input image.

Table 2: Illustration of the latency calculation

	t_0	t_1	...	$t_0 + L_0$	$t_0 + L_0 + 1$	.....	$t_0 + L_0 + L_S + L_1 + L_t + L_2$	$t_0 + L_0 + L_S + L_1 + L_t + L_2 + L_L$
$FIFO_0$	D(0)	D(1)	...	D(1027)	D(1028)	.....	D(2582)	D(2583)
Smoothing unit	.	.	...	$F_0(0)$	$F_0(1)$	.....	$F_0(1555)$	$F_0(1556)$
$FIFO_1$	.	.	...	.	.	.....	S(1545)	S(1546)
Edge strength unit	.	.	...	.	.	.....	$F_1(518)$	$F_1(519)$
$FIFO_2$	.	.	...	.	.	.....	T(514)	T(515)
Localisation unit	.	.	...	.	.	.....	$F_2(0)$	$F_2(1)$

Latency (cycle)
$FIFO_0$: $L_0 = 512 \times 2 + 3 = 1027$ Smoothing unit: $L_S = 10$
$FIFO_1$: $L_1 = 512 \times 2 + 3 = 1027$ Edge strength unit: $L_T = 4$
$FIFO_2$: $L_2 = 512 \times 1 + 2 = 514$ Edge localisation unit: $L_L = 1$

5 Experiments

The present work was primarily implemented in a schematic manner for the top structure and accompanied by the Verilog HDL for all the detailed function units. We used the Xilinx Spartan II 200 FPGA(xc2s200,-5c) as the carrier for our design. The development CAD environment of the entire processing was the Xilinx Foundation. This tool helps the designer to go through the entire design flow, from the design entry, to the synthesis, to the placement and routing and finally to the programming itself. At the same time, both the function and timing simulation are also supported. As described in the implementation reports, this design can be operated at an extremely high frequency of 73.6 MHz. The three function units use a total of 11 298 gates, while the whole design occupies 90% of the device. The performance and hardware usage detail of the top design and of each of the function units are shown in Table 3, whereas the performance comparison between the proposed design and that of Alzahrani and Chen [9] is shown in Table 4.

Moreover, our design is verified by both C programming and FPGA prototyping on the developed versatile development platform with an embedded ARM7 controller [6, 16]. Figs. 15*a* and *b* show a real-world image and the edge detection result. The processing time of a 512 × 512 image on a TI TMS320C6416T DSP is 0.95 s. On the other hand, the FPGA implementation of the proposed architecture produces one edge pixel for each clock cycle. Thus, at 73.6 MHz working frequency, a 512 × 512 image needs only 3.56 ms for the whole process, which is about 265 times faster than the DSP execution. When working at maximum frequency, our design reaches a high frame rate of 280 frames/s for 512 × 512 images. Fig. 16 gives the top view of the presented image processing development platform [16] for hardware prototyping and verification wherein the edge map can be shown on a 120 × 160 LCD in real time.

6 Conclusion

For real-time image processing applications, a hardware realisation is obviously preferred for a more superior processing capability instead of using a general purpose processor. From the point of view of the hardware, such advantages like parallelism and modularity make those algorithms of low-computational complexity and independent operations as good choices; for example, the ADM that was adopted here. In the present paper, we have shown an efficient architecture that is suitable for low-cost VLSI prototyping. A systolic array of Gaussian smoothing filter with the proposed 2^n-approximated parameter was also investigated for its highly regularised structure. The goal of real-time processing was achieved; that is, an output of one-edge pixel every clock cycle under a maximum working frequency of 73.6 MHz and the resulting processing speed is 120 times faster than the software simulation. Moreover, we also proposed a scalable FIFO design in two distinct schemes that can be used for five different sizes. The present work achieved a compact and flexible primitive core for edge detection at a high frame rate of 280 frames/s with low complexity and hence, suitable for various high mobility-demanding machine vision applications [2, 7].

In Section 2, we introduced a heuristic algorithm to generate the proposed 2^n-approximated Gaussian masks. For our future work, owing to the Gaussian masks being quite useful for various real-time imaging applications, we are interested in realising an accurate, adaptive and parameterised architecture for a general purpose Gaussian operator.

Table 3: Performance illustration

Proposed architecture	Smoothing unit	Strength unit	Localisation unit	Total
Gate count	7624	3042	632	11 298
Maximum frequency, MHz	–	–	–	73.6
Latency	10	4	1	15

Table 4: Performance comparison

	Proposed architecture	ADM [9]
Gate count	354 202	351 796
Max. frequency, MHz	73.6	64.1
Latency	2583	2578
Acc. Gaussian error	0.09263	0.11007
Pipeline structure	Systolic array	Tree based
Hardware complexity	High regularity and simplicity	Normal
Image processing size	Five different sizes	Fixed size

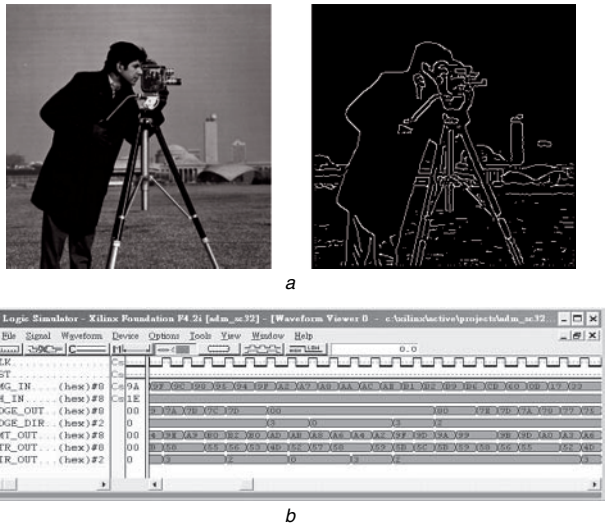

Fig. 15 *Processing result*
a Edge map
b Extracted waveform

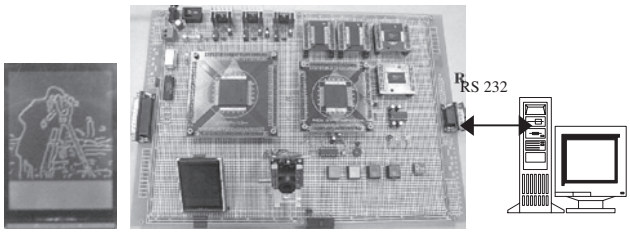

Fig. 16 *Proposed versatile development platform*

7 Acknowledgments

This work was supported in part by National Chip Implementation Center under grant D35-93C-76b and National Science Council, Taiwan, Republic of China

under grant NSC93-2215-E-182-005 and NSC94-2215-E-182-010. Special thanks to Assistant Professor Wen-Chung Kao and his students Sheng-Hong Wang, Lien-Yang Chen, of National Taiwan Normal University, for their technical support on the DSP tool and platform.

8 References

1 Forsyth, D.A., and Ponce, J.: 'Computer vision: a modern approach' (Prentice-Hall, New Jersey, 2002)

2 Lorca, F.G., Kessal, L., and Demigny, D.: 'Efficient ASIC and FPGA implementations of IIR filters for real time edge detection'. Int. Conf. on Image Processing, 26–29 October 1997, pp. 406–409

3 Hsiao, P.Y., Hua, C.H., and Lin, C.C.: 'A novel FPGA architectural implementation of pipelined thinning algorithm'. IEEE Int. Symp. on Circuits and Systems, Vancouver, Canada, 23–26 May 2004, pp. 593–596

4 Huang, S.S., Chen, C.J., Hsiao, P.Y., and Fu, L.C.: 'On-board vision system for lane recognition and front-vehicle detection to enhance driver's awareness'. IEEE Int. Conf. on Robotics and Automation, New Orleans, LA, USA, 26 April–1 May 2004, pp. 2456–2461

5 Gonzalez, R.C., and Woods, R.E.: 'Digital image processing' (Prentice-Hall, New Jersey, 2002, 2nd edn.)

6 McBader, S., and Lee, P.: 'An FPGA implementation of a flexible, parallel image processing architecture suitable for embedded vision systems'. Proc. 2nd Int. Symp. on Parallel and Distributed Processing, 22–26 April 2003, pp. 228–232

7 Dung, L.R., and Lin, M.C.: 'A maskable memory architecture for rank-order filtering', *IEEE Trans. Consum. Electron.*, 2004, **50**, (2), pp. 558–564

8 Basu, M., and Woods, R.E.: 'Gaussian-based edge-detection methods – a survey', *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, 2002, **32**, (3), pp. 252–260

9 Alzahrani, F.M., and Chen, T.: 'A real-time edge detector: algorithm and VLSI architecture', *Real-Time Imaging*, 1997, **3**, (5), pp. 363–378

10 Canny, J.: 'A computational approach to edge detection', *IEEE Trans. Pattern Anal. Mach. Intell.*, 1986, **8**, (6), pp. 679–698

11 Demigny, D.: 'On optimal linear filtering for edge detection', *IEEE Trans. Image Process.*, 2002, **11**, (7), pp. 728–737

12 Boo, M., Antelo, E., and Bruguera, J.D.: 'VLSI implementation of an edge detector based on Sobel operator'. 20th EUROMICRO Conf., 5–8 September 1994, pp. 506–512

13 Hajjar, A., and Chen, T.: 'A VLSI architecture for real-time edge linking', *IEEE Trans. Pattern Anal. Mach. Intell.*, 1999, **21**, (1), pp. 89–94

14 Sivaswamy, J., Salcic, Z., and Ling, K.L.: 'A real-time implementation of nonlinear unsharp masking with FPLDs', *Real-Time Imaging*, 2001, **7**, (2), pp. 195–202

15 Hsiao, P.Y., Hsu, Y.C., Lee, W.T., Tsai, C.C., and Lee, C.H.: 'An embedded analog spatial filter design of the current-mode CMOS image sensor', *IEEE Trans. Consumer Electron.*, 2004, **50**, (3), pp. 945–951

16 Hsiao, P.Y., Wen, H., and Chen, Y.P.: 'Real-time implementation of noise-immune gradient-based edge detection'. IEEE 7th Int. Symp. on Signals, Circuits and Systems, Romania, 14–15 July 2005, pp. 633–666

17 Kanopoulos, N., Vasanthavada, N., and Baker, R.L.: 'Design of an image edge detection filter using the Sobel operator', *IEEE J. Solid-State Circuits*, 1988, **23**, (2), pp. 358–367

18 Lee, C.Y., Cathoor, F.V.M., and De Man, H.J.: 'An efficient ASIC architecture for real-time edge detection', *IEEE Trans. Circuits Syst.*, 1989, **36**, (10), pp. 1350–1359