Journal of Engineering and Applied Sciences 1 (2) : 103-111, 2006

Image Processing Algorithms on Reconfigurable Architecture Using Handel-C

Muthukumar Venkatesan and Daggu Venkateshwar Rao
Department of Electrical and Computer Engineering,
University of Nevada Las Vegas, Las Vegas, NV 89154, USA

Abstract: Computer manipulation of images is generally defined as Digital Image Processing (DIP). DIP is employed in variety of applications, including video surveillance, target recognition and image enhancement. Some of the algorithms used in image processing include convolution, edge detection and contrast enhancement. These are usually implemented in software but may also be implemented in special purpose hardware to reduce speed. In this work the canny edge detection architecture has been developed using reconfigurable architecture and hardware modeled using a C-like hardware language called Handle-C. The proposed architecture is capable of producing one edge-pixel every clock cycle. The hardware modeled was implemented using the DK2 IDE tool on the RC1000 Xilinx Vertex FPGA based board. The algorithm was tested on standard image processing benchmarks and significances of the result are discussed.

Key words: Image processing, architecture, handel-c

INTRODUCTION

Digital image processing is an ever expanding and dynamic area with applications reaching out into our everyday life such as medicine, space exploration, surveillance, authentication, automated industry inspection and many more areas. Applications such as these involve different processes like image enhancement and object detection[1]. Implementing such applications on a generable purpose computer can be easier, but not very time efficient due to additional constraints on memory and other peripheral devices. Application specific hardware implementation offers much greater speed than a software implementation. With advances in the VLSI (Very Large Scale Integrated) technology hardware implementation has become an attractive alternative. Implementing complex computation tasks on hardware and by exploiting the parallelism and pipelining in algorithms yield significant reduction in execution times[2,3].

There are two types of technologies available for hardware design. Full custom hardware design also called as Application Specific Integrated Circuits (ASIC) and semi custom hardware device, which are programmable devices like Digital Signal Processors (DSP's) and Field Programmable Gate Arrays (FPGA's). Full custom ASIC design offers highest performance, but the complexity and the cost associated with the design is very high. The ASIC design cannot be changed and the design time is also very high. ASIC designs are used in high volume commercial applications. In addition, if an error exist in the hardware design, once the design is fabricated, the product goes useless. DSP's are a class of hardware devices that fall somewhere between an ASIC and a PC in terms of the performance and the design complexity. DSP's are specialized microprocessor, typically programmed in C, perhaps with assembly code for performance. It is well suited to extremely complex math intensive tasks such as image processing. Hardware design knowledge is still required, but the learning curve is much lower than some other design choices[4]. Field Programmable Gate Arrays are reconfigurable devices. Hardware design techniques such as parallelism and pipelining techniques can be developed on a FPGA, which is not possible in dedicated DSP designs. Implementing image processing algorithms on reconfigurable hardware minimizes the time-to-market cost, enables rapid prototyping of complex algorithms and simplifies debugging and verification. Therfore, FPGAs are an ideal choice for implementation of real time image processing algorithms.

FPGAs have traditionally been configured by hardware engineers using a Hardware Design Language (HDL). The two principal languages being used are Verilog and VHDL. Verilog and VHDL are specialized design techniques that are not immediately accessible to software engineers, who have often been trained using imperative programming languages. Consequently, over the last few years there have been several attempts at translating algorithmic oriented programming languages directly into hardware descriptions. A new Clike hardware description language called Handel-C introduced by Celoxica[5], allows the designer to focus more on the

specification of an algorithm rather than adopting a structural approach to coding. The objective of this study is to implement image processing algorithms like median filter, morphing, convolution and canny edge detection on FPGA using Handel-C and compare against the performance of software implementation on a general purpose computer.

The last few years have seen an unprecedented effort by researchers in the field of image processing using reconfigurable devices. Richard G.S[6] discusses the idea of parameterized program generation of convolution filters in an FPGA. A 2-D filter is assembled from a set of multipliers and adders, which are in turn generated from a canonical serial-parallel multiplier stage. Atmel application notes[7] discuss 3x3 convolver with run-time reconfigurable vector multiplier in Atmel FPGA. Lorca, Kessal and Demigny[8] proposed a new organization of filter at 2 and 1 D levels, which reduces the memory size and the computation cost by a factor of two for both software and hardware implementations. Fahad Alzahrani and Tom Chen[9] present a high performance edge detection VLSI architecture for real time image processing applications, the architecture is fully pipelined. It is capable of producing one edge-pixel every clock cycle at a clock rate of 10 MHz, the architecture can process 30 frames per second. Gemignani *et al.*[10] presents the real time implementation of two mathematical operators which are commonly used to detect edges: (i) gradient of Gaussian, (ii) 'b' operator. The algorithms are implemented on digital signal processor.

This study presents a simple modeling of four image processing algorithm listed above on reconfigurable architecture using Handle-C. The algorithm was developed using the DK2 development environment and was implemented using the Xilinx Vertex FPGA based PCI board. The reason for selecting the Handle-C language pardigram is its C based syntax and the DK2 IDE's co-design framework that allows seamless integration of software and hardware image processing modules. The front-end Graphical User Interface (GUI) was developed using Visual C++ environment.

IMAGE PROCESSING ALGORITHMS

This section discusses the theory of most commonly used image processing algorithms like, 1) Filtering, 2) Morphological Operations, 3) Convolution and 4) Edge detection.

Median filter: A median filter is a non-linear digital filter which is able to preserve sharp signal changes and is very effective in removing impulse noise (or salt and pepper noise)[1]. An impulse noise has a gray level with higher low that is different from the neighborhood point. Linear

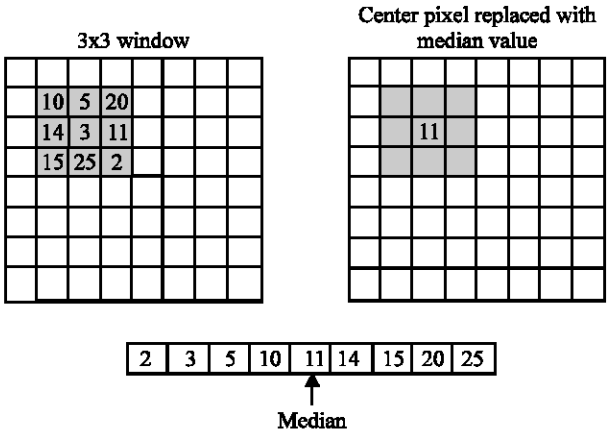

Fig. 1: Median filter

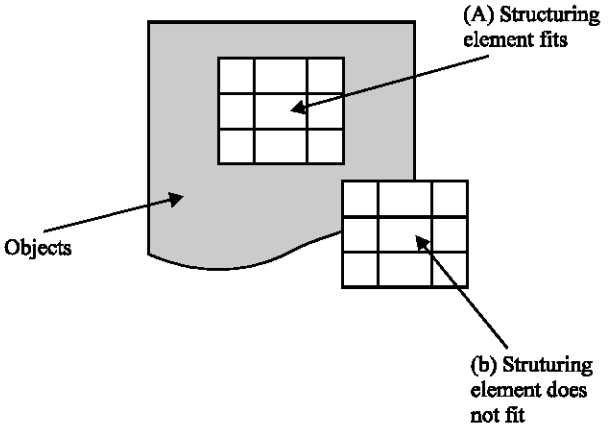

Fig. 2: Concept of structuring element fitting and not fitting

filters have no ability to remove this type of noise without affecting the distinguishing characteristics of the signal. Median filters have remarkable advantages over linear filters for this particular type of noise. Therefore median filter is very widely used in digital signal and image/video processing applications.

A standard median operation is implemented by sliding a window of odd size (e.g., 3×3 window) over an image. At each window position the sampled values of signal or image are sorted and the median value of the samples is taken as the output that replaces the sample in the center of the window as shown in Fig. 1.

The main problem of the median filter is its high computational cost (for sorting N pixels, the time complexity is O(N log N), even with the most efficient sorting algorithms). When the median filter is carried out in real time, the software implementation in general-purpose processors does not usually give good results. The execution times are reduced by implementing median filters on FPGAs.

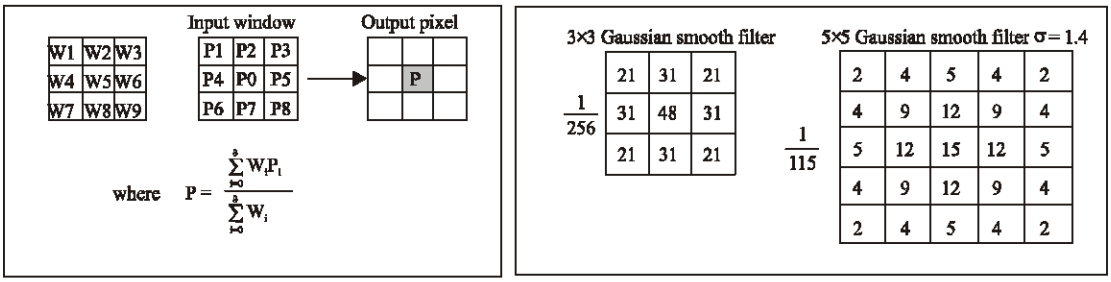

Fig. 3: (a) Convolution Operation and (b) Gaussian Function for smoothing

Morphological operation: The term morphological image processing refers to a class of algorithms that is interested in the geometric structure of an image. Morphology can be used on binary and gray scale images and is useful in many areas of image processing, such as skeletonization, edge detection, restoration and texture analysis.

A morphological operator uses a structuring element to process an image as shown in Fig. 2. The structuring element is a window scanning over an image, which is similar to the pixel window used in the median filter. The structuring element can be of any size, but 3×3 and 5×5 sizes are common. When the structuring element scans over an element in the image, there may be instances where the structuring element completely fits inside the object (Fig. 2a) or does not fit inside the object (Fig. 2b).

The most basic building blocks for many morphological operators are erosion and dilation[11]. Erosion as the name suggests is shrinking or eroding an object in an image. Dilation on the other hand grows the image object. Both of these objects depend on the structuring element and how it fits within the object.

Convolution operation: Convolution is a simple mathematical operation which is fundamental to many common image processing operators. Convolution is a way of multiplying together two arrays of numbers of different sizes to produce a third array of numbers. In image processing the convolution is used to implement operators whose output pixel values are simple linear combination of certain input pixels values of the image. Convolution belongs to a class of algorithms called spatial filters. Spatial filters use a wide variety of masks (kernels), to calculate different results, depending on the desired function

2D-Convolution, is most important to modern image processing. The basic idea is that a window of some finite size is scanned over an image. The output pixel value is the weighted sum of the input pixels within the window where the weights are the values of the filter assigned to every pixel of the window. The window with its weights is called the *convolution mask*. Mathematically, convolution on image can be represented by the following equation:

$$y(m,n)=\sum_{i=0}^{\text{Height of image}}\sum_{j=0}^{\text{width of image}} h(i,j)x(m-i,n-j),$$

where x is the input image, h is the filter and y is the image

An important aspect of convolution algorithm is that it supports a virtually infinite variety of masks, each with its own feature. This flexibility allows many powerful applications. For example, the derivative operators which are mostly used in edge detection use 3x3 window masks. They operate only one pixel and its directly adjacent neighbors in one clock cycle. Fig. 3a shows a 3×3 convolution mask operated on an image. The center pixel is replaced with the output of the algorithm. Similarly larger size convolution masks can be operated on an image.

In 2-D, a circularly symmetric Gaussian has the form

$$G(x,y)=\frac{1}{2\pi\sigma^2}\frac{-(x^2+y^2)}{2\sigma^2}$$

where σ is the standard deviation of the distribution.

The idea of Gaussian convolution is to use this 2-D distribution as a point spread function and this is achieved by convolution. Since the image is stored as a collection of discrete pixels. A discrete approximation to the Gaussian function is required to perform the convolution. In theory, the Gaussian distribution is non-zero everywhere, which would require an infinitely large convolution kernel, but in practice it is effectively zero more than about three standard deviations from the mean and so convolution kernel is truncated as shown in Fig. 3b.

Edge detection: The edge detection algorithm explained in this section is similar to canny edge detection. First the

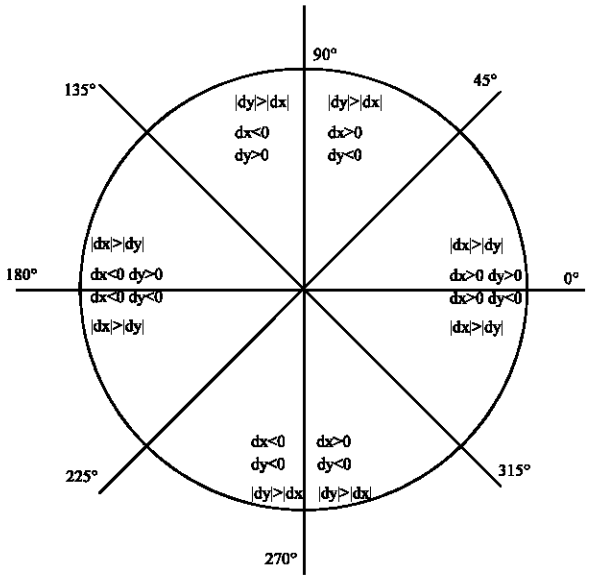

Fig. 4: Orientation

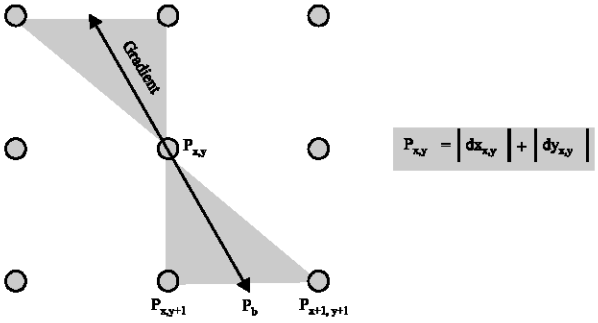

Fig. 5: Direction of the gradient

image is smoothed by Gaussian Convolution. A simple 2-D first derivative operator is applied to the smoothed image to highlight regions of the image with high first spatial derivatives. Edges translate into ridges in the gradient magnitude image. The algorithm then tracks along the top of these ridges and sets to zero all pixels that are not actually on the ridge top so as to give a thin line in the output, a process known as non-maximal suppression. The tracking process exhibits hysterisis controlled by two thresholds: T1 and T2, with T1>T2. Tracking can only begin at a point on a ridge higher than T1. Tracking then continues in both directions from that point until the height of the ridge falls below T2. This hysterisis helps to remove the edge fragments.

In the first stage the 5x5 Gaussian convolution mask of standard deviation (ó = 1.4) used for smoothing. The horizontal and vertical gradients are calculated using the differences between adjacent pixels, one way to find edges is to explicitly use a {-1 , +1 } operator. The Prewitt masks are based on the idea of the central difference:

$$\frac{\partial 1}{\partial x}=\frac{1(x+1,y)-(x-1,y)}{2}, \quad \text{and}$$
$$\frac{\partial 1}{\partial y}=\frac{1(x,y+1)-1(x,y-1)}{2}$$

corresponds to the following convolution kernel:

These convolutions are applied to the results obtained from the smoothing stage to get the horizontal (dx) and vertical (dy) gradients.

In non maximum suppression stage an edge point is defined to be a point whose strength is a local maximum in the direction of the gradient. This is a difficult constraint to satisfy and is used to thin the ridges found by thresholding. This step works with the magnitude and orientation of the gradient at the pixel under consideration and creates one pixel-width edges. The values of each component of the gradient determined from the previous stage are employed to determine the magnitude and direction.

Classically, to calculate the direction of the gradient the arctangent is employed. The arctangent is a very complex operation, which increases the logic depth and delay. The value and sign of the components of the gradient are used to analyze the direction. Consider the pixel , x y p and the derivative at the pixel are dx and dy, the gradient at p is approximated as shown in Fig. 4.

Once the direction of the gradient is known, the values of neighborhood pixels at the point under analysis are interpolated. The pixel that has no local maximum gradient magnitude is eliminated. Comparison is made among the actual pixel and its neighbors along the direction of the gradient.

For example, if the direction of the gradient to be between 90 and 135°, then compare the magnitude of the gradient at $P_{x, y}$ with the magnitude of the gradient at the points adjacent to $P_{x, y}$ in the direction of the gradient as shown in Fig. 5.

The value of the gradient at the points p_a and p_b are defined as follow:

$$Pa\frac{P_{x-1,y-1}+P_{x,y-1}}{2} \quad Pb=\frac{P_{x,y+1}+P_{x+1,y+1}}{2}$$

The center pixel $P_{x,y}$ is considered to be an edge, if $p_{x, y}>p_a$ and $p_{x, y}>p_b$. If both conditions are not satisfied the center pixel is eliminated. The output image of this stage consists of some individual pixels and is usually thresholded to decide which edges are significant. Two thresholds T_H (High Threshold) and T_L (Low Threshold)

are applied, where $T_H > T_L$. If the gradient magnitude is greater than T_H that pixel is considered as a definite edge. If the gradient magnitude is less than T_L than that pixel is unconditionally set to zero. If the gradient magnitude is between these two, then it is set to zero unless there is a path from this pixel to a pixel with a gradient above T_H; the path must be entirely through pixels with gradients of at least T_L. A 3×3 moving window operator is used to evaluate this threshold. The center pixel is said to be connected if at least one neighboring pixel value is greater than T_L and the resultant is an image with sharp edges.

IMPLEMENTATION RESOURCES

FPGAs have traditionally been configured by hardware engineers using a Hardware Design Language (HDL). Consequently, over the last few years there have been several attempts at translating algorithmic oriented programming languages directly into hardware descriptions. A new C like hardware description language called Handel-C introduced by Celoxica, allows the designer to focus more on the specification of an algorithm rather than adopting a structural approach to coding. For these reasons the Handel-C is used for implementation of image processing algorithms on FPGA.

Handle-C: Handel-C is essentially an extended subset of the standard ANSI-C language, specifically designed for use in a hardware environment. Unlike other C to FPGA tools which rely on going via several intermediate stages, Handel-C allows hardware to be directly targeted from software, allowing a more efficient implementation to be created. The language is designed around a simple timing model that makes it very accessible to system architects and software engineers.

The Handel-C compiler comes packaged with the Celoxica DK1 development environment. DK1 does not provide synthesis and the suite must be used in conjunction with one of any number of synthesis tools available to complete the design flow from idea to hardware.

Targets supported by handel-C: Handel-C supports two targets. The first is a simulator target that allows development and testing of code without the need to use any hardware. This is supported by a debugger and other tools. The second target is the synthesis of a netlist for input to place and route tools. Place and route is the process of translating a netlist into a hardware layout. This allows the design to be translated into configuration data for particular chips. When compiling the design for a hardware target, Handel-C emits the design in Electronic Design Interchange Format (EDIF) format. A cycle count is available from the simulator and an estimate of gate count is generated by the Handel-C compiler. To get definitive timing information and actual hardware usage, the place and route tools need to be invoked.

HARDWARE IMPLEMENTATION

The algorithms implemented in this work use the moving window operator. The moving window operator usually process one pixel of the image at a time, changing its value by some function of a local region of pixels (covered by the window). The operator moves over the image to process all the pixels in the image. A 3×3 moving window is used for the median filtering, morphological and edge detection algorithms and a 5×5 moving window used in Gaussian smoothing operation.

For the pipelined implementation of image processing algorithms all the pixels in the moving window operator must be accessed at the same time for every clock. A 2D-matrix of First In First Out (FIFO) buffers are used to create the effect of moving an entire window of pixels through the memory for every clock cycle (Fig. 6). A FIFO consists of a block of memory and a controller that manages the traffic of data to and from the FIFO. The FIFO's are implemented using circular buffers constructed from multi-port block RAM with an index keeping track of the front item in the buffer. The availability of multi-port block RAM in the Xilinx Vertex-E FPGA helps in achieving the read and write operations of the RAM in the same clock cycle. This allows a throughput of one pixel per clock cycle. The same effect can be achieved using double-width RAMs implemented in lookup tables on the FPGA. However, the use of block RAMs is more efficient and has lessassociated logic for reading and writing.

For a 3×3 moving window two FIFO buffers are used. The size of the FIFO buffer is given as W-M, where W is the width of the image and M the size of the window (M×M). To access all the values of the window for every clock cycle the two FIFO buffers must be full. Figure 6 shows the architecture of the 3×3 moving window. For every clock cycle, a pixel is read from the RAM and placed into the bottom left corner location of the window.

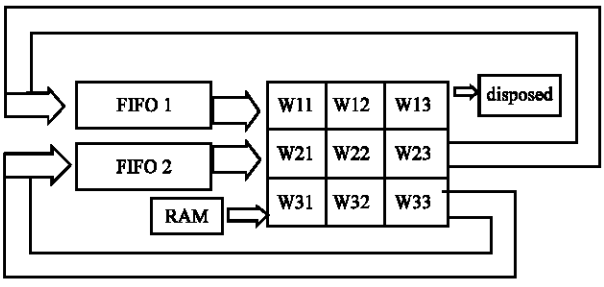

Fig. 6: Architecture of 3x3 moving window

Median filter: A median filter is implemented by sliding a window of odd size on a image. A 3×3 window size is chosen for implementation for median filter, because it is small enough to fit onto the target FPGA's and is considered large enough to be effective for most commonly used image sizes. The median filter uses the 3×3 window operation discussed in Section 2.1. The median filtering operation sorts the pixel values in a window in ascending order and picks up the middle value, the center pixel in the window is replaced by the middle value. The most efficient method of accomplishing sorting is with a system of hardware compare/sort units, which allows sorting a window of pixels into an ascending order.

Morphological operations: The basic morphological operators are erosion and dilation. The erosion and dilation of a grayscale image are called grayscale erosion or dilation. These grayscale erosion is performed by minimum filter, whereas the dilation is performed by maximum filter. In a 3×3 minimum filter, the center pixel is replaced by a minimum value of the pixels in the window In a maximum filter, the center pixel is replaced by a maximum value of the pixels in the window. The implementation of minimum and maximum filters is similar to the median filters implementation.

Convolution operation: Convolution is a very complex operation that requires huge computational power. To calculate a pixel for a given mask of size m x n, m * n multiplications , m *n-1 additions and one division are required. Therfore, to perform a 3×3 multiplication on a 256×256 gray scale image, 589824 multiplications , 393216 additions and one division are required.

Multiplication and division operators produce the deepest logic. A single cycle divide, or multiplication produces a large amount of hardware and long delays through deep logic. In order to improve the performance of the convolution operation, it is necessary to reduce the multiplication and division operators. Multiplication and division can be done using bit shifting, but this is only possible with the powers of 2's. Multiplier-less multiplication can be employed to do multiplication of non power of 2's digits, where multiplication is done with only shifts and additions from the binary representation of the multiplicand.

Edge detection: Hardware implementation of edge detection algorithm is discussed in this section. Edge detector operation implemented in this study consists of four stages:

- Image smoothing;
- Vertical and Horizontal Gradient Calculation;

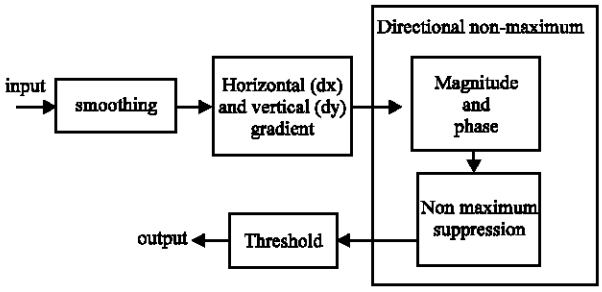

Fig. 7: Design Flow of Edge detection

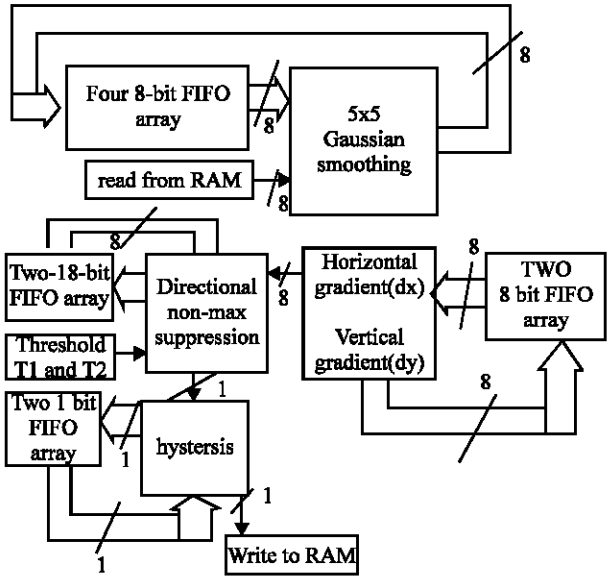

Fig. 8: Pipelined Edge Detection

- Directional Non Maximum Suppression;
- Threshold.

Normally, the edge detection algorithm is implemented by applying the Gaussian smoothing on the entire image. Furthermore, the smoothened image is used as an input to calculate the gradient at every pixel and these gradient values are used to calculate the phase and the magnitude at the pixel, which is followed by non-maximum suppression and output, is thus thresholded.

Using the above mentioned design, each stage is accomplished separately one after another. It can be perceived as a trade off between an efficient output and resources involved because, while implementing such design on the hardware, a lot of resources and clock cycles are consumed. As shown in Fig. 7 the magnitude and phase calculation stage and non-maximum suppression stage are combined to directional non-maximum stage. A pipelined architecture shown in Fig. 8 is designed.

Image smoothing: Smoothing of the image is achieved by 5×5 Gaussian convolutions. A 5×5 moving window operator is used, four FIFO buffers are employed to access all the pixels in the 5×5 window at the same time. Since the design is pipelined, the Gaussian smoothing starts once the 2 FIFO buffers are full. That is, the output is produced after a latency of twice width of image plus three cycles. The output of this stage is given as input to the next stage.

Vertical and horizontal gradient calculation: This stage calculates the vertical and horizontal gradients using 3×3 convolution kernels shown in Section 2.4. An 8-bit pixel in row order of the image produced during every clock cycle in the image smoothing stage is used as the input in this stage. Since 3×3 convolution kernels are used to calculate the gradients, neighboring eight pixels are required to calculate the gradient of the center pixel and the output pixel produced in previous stage is a pixel in row order. In order to access eight neighboring pixels in a single clock cycle, two FIFO buffers are employed to store the output pixels of the previous stage.

The gradient calculation introduces negative numbers. In Handel-C, negative numbers can be handled easily by using signed data types. Signed data means that a negative number is interpreted as the 2's complement of number. In the design, an extra bit is used for signed numbers as compared to unsigned 8 bit numbers (i.e., 9 bits are used to represent a gradient output instead of 8). Two gradient values are calculated for each pixel, one for vertical and other for horizontal. The 9 bits of vertical gradient and the 9 bits of the horizontal gradient are concatenated to produce 18 bits. Since the whole design is pipelined, an 18 bit number is generated during every clock cycle, which forms the input to the next stage.

Directional non maximum suppression: The output of the previous stage is used as input in this stage. In order to access all the pixels in the 3×3 window at the same time two eighteen bit FIFO buffers of width of the image minus three array size are employed. Once the direction of the gradient is known, the values of the pixels found in the neighborhood of the pixel under analysis are interpolated. The pixel that has no local maximum gradient magnitude is eliminated. The comparison is made between the actual pixel and its neighbors, along the direction of the gradient.

Threshold: The output obtained from the non maximum suppression stage contains single edge pixels which contribute to noise. This can be eliminated by thresholding. Two thresholds (high threshold (T_H) and low threshold (T_L) are employed. If the gradient of the edge pixel is above T_H, it is considered as a strong edge pixel. If the gradient of the edge pixel is below T_L, it is unconditionally set to zero. If the gradient is between these two threshold, then it is considered as a weak edge pixel. It is set to zero unless there is a path from this pixel to a pixel with a gradient above T_H; the path must be entirely through pixels with gradients with at least T_L threshold.

To get the connected path between the weak edge pixel and the strong edge pixel, a 3×3 window operator is used. If the center pixel is a strong edge pixel and any of the neighbors is a weak edge pixel, then weak edge pixel is considered as a strong edge pixel. The resultant image is an image with optimal edges.

RESULTS

The image processing algorithms discussed above were modeled in Handel-C using the DK2 environment. The design was implemented on RC1000-PP Xilinx Vertex-E FPGA based hardware. The timing result of the image processing algorithms on a 256x256 size gray scale Lena image is shown in Table 1.

The hardware implementation of the algorithms is compared with implementation on a PentiumIII 1.3 GHz machine using Visual C++ Version 6.0 without any optimization. The speed of our FPGA solution for the image processing algorithms is 20 times faster than the

Table 1: Timing result edge detection algorithm on 256×256 gray scale image

		Xilinx vertex-E		FPGA pentium III	
		Freq [MHz]	Time [ms]	Freq [MHz]	Time [ms]
Median filter, morphological operation		25.9	2.56	1300	51
Gaussian convolution	Direct division by 115	25.9	2.62	1300	31
	Division using right shift(>> 7)	42	1.57	1300	31
Gaussian smoothing	Direct multiplication	42.03	1.58	1300	16
	LUT based multiplication	50.99	1.31	1300	16
Edge detection		16	4.2	1300	47

Table 5.2 Implementation Cost on FPGA

Design	Lookup tables (LUTs)	Flip-flops	Block RAMs	CLB slices	Input/output Blocks
Median, Erosion and Dilation	406	298	2 (1%)	652 (3%)	145 (35 %)
5×5 Convolution div by 115	1040	926	4 (2%)	994 (5%)	145 (35%)
5×5 Convolution div by shift (>>7)	597	802	4 (2%)	1035 (5%)	145 (35%)
3×3 Convolution direct multiplication	735	442	2 (1%)	479 (2%)	145 (35%)
3×3 Convolution LUT based Multiplication	384	428	2 (1%)	479 (2%)	145 (35%)
Edge Detection	945	807	4 (%)	1820 (10%)	145 (35%)

Fig. 9: (a) Original Image, (b) Gaussian convolution, (c) Salt and pepper noise, (d) Median filter, (e) Erosion, (f) Dilation, (g) Edge detection

software implementation. Table 2 lists the implementation cost on hardware and Fig. 9 shows the output of hardware implemented images.

REFERENCES

1. John, C. Ross, 1994. Image Processing Hand book, CRC Press.
2. Stephen, D., R.J. Brown, J. Francis and Z.G. Rose, 1992. Vranesic Filed Programmable Gate Arrays.
3. Moore, M., 1995. A DSP-based real time image processing sytem". In the Proceedings of the 6th International conference on signal processing applications and technology, Boston MA, August.
4. Rolf, F.P. Molz, M. Engel, G. Fernando L.T. Moraes and R. Michel, 2000. Design of a Classification System for Rectangular Shapes Using a Co-Design Environment". In the 13th Symposium on Integrated Circuits and Systems Design , pp: 281-286.
5. Richard, G. Shoup, 1993. Parameterized Convolution Filtering in a Field Programmable Gate Array Interval. Technical Report, Palo Alto, California.
6. Convolver with Run-Time Reconfigurable Vector Multiplier in Atmel AT6000 FPGAs. AT6000 FPGAs Application Note 1997.
7. Gemignani, V., M. Demi, M. Paterni , M. Giannoni and A. Benassi, 2001. DSP implementation of real time edge detectors". In the Proceedings of speech and image processing, pp: 1721-1725.
8. Canny, J., 1986. A computational approach to the edge detection. IEEE Trans Pattern and Machine Intelligent. Vol PAMI-8 pp: 679-698.
9. Lorca, F.G., L. Kessal and D. Demigny, 1997. Efficent ASIC and FPGA implementation of IIR filters for Real time edge detection. In the International Conference on image processing (ICIP-97) Volume 2.
10. Fahad Alzahrani, 1997. Tom Chen Real-time high performance Edge detector for computer vision applications. In the Proceedings of ASP-DAC, pp: 671-672.
11. Celoxica Ltd. www.celoxica.com.
12. Peter, M.C., Curry, M. Fearghal and K. Liam, 2000. Xilinx FPGA implementation of a pixel processor for object detection applications. Proc Irish Signals and Systems Conference.
13. Xilinx VertexTM-E Field Prgrammable Gate Arrays (V2.4) July 17, 2002 Production Product specification.
14. Venkateshwar Rao Daggu, 2003. Design and Implementation of an Efficient Reconfigurable Architecture for Image Processing Algorithms using Handel-C. Masters Thesis, UNLV.
15. Man, Ng, 2002. High level design for high speed fpga devices.master's Thesis Dept of computing. Imperial College.