International Journal of Theoretical and Applied Computer Sciences
Volume 1 Number 1 (2006) pp. 9–34
(c) GBS Publishers and Distributors (India)
http://www.gbspublisher.com/ijtacs.htm

Implementation and Evaluation of Image Processing Algorithms on Reconfigurable Architecture using C-based Hardware Descriptive Languages

Daggu Venkateshwar Rao*, Shruti Patil, Naveen Anne Babu and V Muthukumar
Department of Electrical and Computer Engineering
University of Nevada Las Vegas.
Las Vegas, NV 89154
**E-mail: venkim@egr.unlv.edu*

Abstract

With the advent of mobile embedded multimedia devices that are required to perform a range of multimedia tasks, especially image processing tasks, the need to design efficient and high performance image processing systems in a short time-to-market schedule needs to be addressed. Image Processing algorithms implemented in hardware have emerged as the most viable solution for improving the performance of image processing systems. The introduction of reconfigurable devices and system level hardware programming languages has further accelerated the design of image processing in hardware. Most of the system level hardware programming languages introduced and commonly used in the industry are highly hardware specific and requires intermediate to advance hardware knowledge to design and implement the system. In order to overcome this bottleneck various C-based hardware descriptive languages have been proposed over the past decade [25]. These languages have greatly simplified the task of designing and verifying hardware implementation of the system. However, the synthesis process of the system to hardware was not completely addressed and was conducted using manual methods resulting in duplication of the implementation process. Handel-C is a new C-based language proposed that provides direct implementation of hardware from the C-based language description of the system. Handel-C language and the IDE tool introduced by Celoxica Ltd. provides both simulation and synthesis capabilities. This work evaluates the performance and efficiency of Handel-C language on image processing algorithms and is compared at simulation level with another popular C-based system level language called SystemC and at synthesis level with the industry standard Hardware Descriptive language, Verilog. The image processing algorithms considered include, image filtering, image smoothing and edge detection. Comparison parameters at simulation level include, man-hours for implementation, compile time and lines of code. Comparison parameters at synthesis level include logic resources required, maximum frequency of operation and execution time. Our evaluations show that the Handel-C implementation perform better at system and synthesis level compared to other languages considered in this work. The work also proposes a novel hardware architecture to implement Canny's edge detection algorithm. The proposed architecture is capable of producing one edge-pixel every clock cycle. A comparison of our architecture for Canny's edge detection with other architecture is also discussed.

1. 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 general 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 parallelism and pipelining in algorithms yield significant reduction in execution times [2].

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 (DSPs) 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, during design fabrication the presence of a single error renders the chip useless. DSPs are a class of hardware devices that fall somewhere between an ASIC and a PC in terms of the performance and the design complexity. DSPs are specialized microprocessors, typically programmed in C, or with assembly code for improved performance. It is well suited to extremely complex math intensive tasks such as image processing. Knowledge of hardware design is still required, but the learning curve is much lower than other design choices [3]. 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. Therefore, FPGAs are an ideal choice for implementation of real time image processing algorithms [4].

FPGAs have traditionally been configured by hardware engineers using a Hardware Design Language (HDL). The two principal languages used are Verilog HDL (Verilog) and Very High Speed Integrated Circuits (VHSIC) HDL (VHDL) which allows designers to design at various levels of abstraction. 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.

C-based hardware descriptive languages have been proposed and developed since the late 1980s. Some of the C-based hardware descriptive languages include Cones [33], HardwareC [30], Transmogrifier C [27], SystemC [28], OCAPI [31], C2Verilog [32], Cyber [34], SpecC [26], Nach C [29], CASH [24]. A new C like hardware description language called Handel-C introduced by Celoxica [2, 5], allows the designer to focus more on the specification of an algorithm rather than adopting a structural approach to coding.

Given the importance of digital image processing and the significance of their implementations on hardware to achieve better performance, this work addresses implementation of image processing algorithms like median filter, morphological, convolution and smoothing operation and edge detection on FPGA using Handel-C language. Also novel architectures for the above mentioned image processing algorithms have been proposed. The Canny’s edge detection algorithm implementation is compared at simulation and synthesis levels with other hardware descriptive languages. The Handel-C implementation is compared with systemC language at simulation level and with Verilog at synthesis level. The Canny’s edge detection implementation is also compared with standard C implementations and with design implemented using SA-C language [35].

2. Prior Work

The last few years have seen an unprecedented effort by researchers in the field of image processing in hardware. Prior research can be categorized based on the type of hardware and the image processing algorithm implemented. The type of hardware considered for image processing acceleration inlcude Application Specific Integrated Chips (ASIC), Digital Signal Processors (DSP) and Reconfigurable Logic Devices (FPGA). The image processing algorithms considered for hardware implementation include: convolution, image filtering and edge detection (Sobel’s, Prewitt’s and Canny’s edge detection). Some researchers have also considered hardware implementations specific to FPGA vendors like Xilinx, Amtel and Altera.

Convolution and Image Filtering

G.S. Richard [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 note [7] discusses a 3x3 convolution filter with run-time reconfigurable vector multiplier in Atmel FPGA. Lorca, Kessal and Demigny [8] proposed a new organization of filter at 2D and 1D levels, which reduces the memory size and the computation cost by a factor of two for both software and hardware implementations. A. Nelson [11] implemented the Rank Order Filter, Erosion, Dilation, Opening, Closing and Convolution algorithms using VHDL and MATLAB on Altera FLEX 10K FPGA and Xilinx Virtex FPGA. S. Hirai, M. Zakouji and T. Tsuboi [12] implemented three image processing algorithms for computation of the image gravity center, detection of object orientation using a radial projection and computation of Hough transform. They developed an FPGA-based vision system that used the Xilinx Virtex2E mounted on the FPGA board. The algorithms were coded using C/C++ and compiled into the HDL language CycleC using SystemCompiler. CycleC can be converted into VHDL and Verilog.

Edge Detection

Fahad Alzahrani and Tom Chen [9] present a high performance, pipelined edge detection VLSI architecture for real time image processing applications. It is capable of producing one edge-pixel every clock cycle at a clock rate of 10 MHz and the architecture can process 30 frames per second. Peter Baran et.al. [13] implemented the edge detection convolution algorithm. They wrote the code in ImpulseC, compiled using Impulse Co-developer and synthesized for Altera Nios processor using Altera Quartus tools. Various IP core vendors offer VHDL codes for image processing algorithms like edge detection using Sobel’s and Prewitt’s edge detection algorithms, Laplacian filer, low-pass filter, convolution etc. [14]. These codes can be synthesized into FPGAs using appropriate synthesis software. Altera has implemented Prewitt edge detection Using SOPC Builder & DSP Builder Tool Flow on the Altera Nios II processor [15]. H. Neoh and A. Hazanchuk present implementation of the Canny edge detection algorithm and Prewitt filter for the Altera FPGAs using DSP Builder, a development tool that interfaces between the Altera Quartus II design software and MATLAB/Simulink tools [16]. Draper et. al. have implemented many image processing algorithms on FPGAs [17]. In one of their implementations, they measured the performance of SA-C routines for the first three steps of canny edge detection algorithm using the SA-C compiler and executing them on an Annapolis Microsystems WildStar using a single Xilinx XV-2000E FPGA.

P. Mc Curry, F. Morgan and L. Kilmartin have compared the performance of FPGA and distributed RAM architecture with current programmable DSP-based implementations by implementing among others, using the RC 1000-PP Virtex FPGA based development platform and Handel-C HDL. Laplacian edge detection, morphological algorithms and several image processing algorithms are implemented [18]. W. Luk and T. Wu and I. Page have devised an approach to partition algorithmic procedures into ASIC and general processors. They tested their approach for the canny edge detection algorithm on an FPGA [19]. Trost et. al. designed a reconfigurable system using FPGA modules based on Xilinx XC4008E, Spartan XCS40 and Virtex XCV100 FPGA devices for implementation of real-time image processing circuits. They used the VHDL language to test the system for image processing algorithms like image filtering (low pass, high pass, non-linear) edge detection (Sobel, Canny, Non-linear Laplace) and image rotation [20,21].

This work presents a novel architecture model of four image processing algorithm (Median Filtering, Morphological Operations, Convolution and Edge detection) on reconfigurable architecture using Handle-C. For performance comparison of our implementation the Canny's edge detection algorithm was also implemented using SystemC and Verilog hardware descriptive languages. For the Handel-C implementation the algorithms were 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. Only simulation level design was implemented using SystemC and was compiled using OSCI SystemC compiler. The verilog implementation was simulated using ALDEC's Active HDL simulator and synthesized using Xilinx's Xilinx Synthesis Tool (XST). The implementation environment for verilog was similar to the Handel-C implementation.

3. 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 or lower value that is different from the neighborhood point. Linear 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. 3x3 window) over an image. At each window position the sampled values of signal or image are sorted, and the median value of the samples replaces the sample in the center of the window as shown in Figure 1.

Let W be a window with an odd number of points. Then the median filter is given by

$$y_s = median\{x_{r+s} : r \in W\} \qquad (2.1)$$

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.

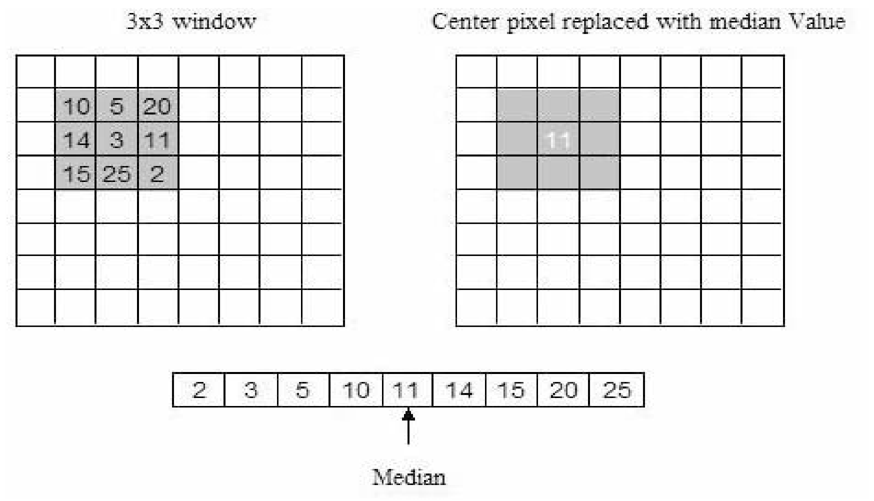


Figure 1: Median Filter

Morphological Operation

The term morphological image processing refers to a class of algorithms that transforms 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 Figure 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 3x3 and 5x5 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 (Figure 2a) or does not fit inside the object (Figure 2b).

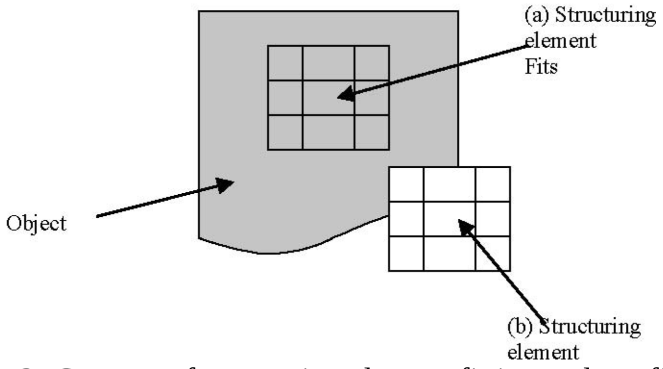


Figure 2: Concept of structuring element fitting and not fitting

The most basic building blocks for many morphological operators are erosion and dilation [15]. 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.

Given a *(2M+1) x (2M+1)* structuring element B and an *NxN* image array A, the dilation of A by B may be defined as:

$$C(n_1, n_2) = Max[(A(n_1 - i, n_2 - j) + B(i, j))] \tag{2.2}$$

for all i, j such that $-M \leq i, j \leq M$, with $0 \leq n_1, n_2 \leq N\text{-}1$.

Given a *(2M+1) x (2M+1)* structuring element B and an *NxN* image array A, the erosion of A by B may be defined as:

$$C(n_1, n_2) = Min[(A(n_1 - i, n_2 - j) + B(i, j))] \tag{2.3}$$

for all i, j such that $-M \leq i, j \leq M$, with $0 \leq n_1, n_2 \leq N\text{-}1$.

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, 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 to scan a window of some finite size 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:

$$y(m,n) = \sum_{i=0}^{Height\,of\,image} \sum_{j=0}^{width\,of\,image} h(i,j)\; x(m-i, n-j), \tag{2.4}$$

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. Figure 3 shows a 3x3 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.

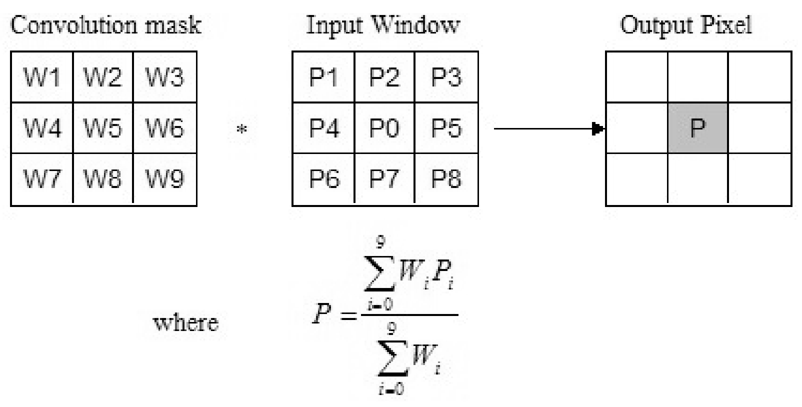


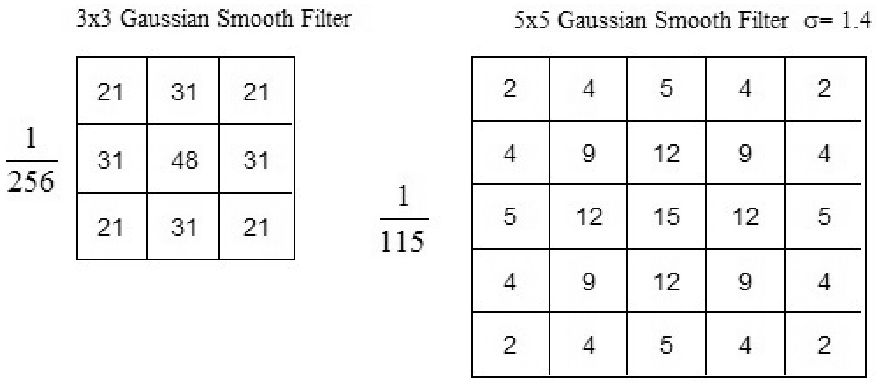
Figure 3: Convolution

The Gaussian smoothing operator is a 2-D convolution operator that is used to `blur' images and remove detail and noise. In this sense it is similar to the mean filter, but it uses a different kernel that represents the shape of a Gaussian hump.
In 2-D, an isotropic (i.e. circularly symmetric) Gaussian has the form:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{2.4}$$

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 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 or masked at this point. Figure 4, shows a suitable integer valued convolution mask that approximates a Gaussian with a σ = 1.4.



3x3 Gaussian Smooth Filter

$$\frac{1}{256}\begin{bmatrix} 21 & 31 & 21 \\ 31 & 48 & 31 \\ 21 & 31 & 21 \end{bmatrix}$$

5x5 Gaussian Smooth Filter σ= 1.4

$$\frac{1}{115}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Figure 4: Convolution Masks

Edge Detection

Edges are places in the image with strong intensity contrast. Edges often occur at image locations representing object boundaries; edge detection is extensively used in image segmentation when we want to divide the image into areas corresponding to different objects. Representing an image by its edges has the further advantage that the amount of data is reduced significantly while retaining most of the image information. Edges can be detected by applying a high pass frequency filter in the Fourier domain or by convolving the image with an appropriate kernel in the spatial domain. In practice, edge detection is performed in the spatial domain, because it is computationally less expensive and often yields better results. Since edges correspond to strong illumination gradients, the derivatives of the image are used for calculating the edges.

The Canny edge detection algorithm is considered a “standard method” and is used by many researchers, because it provides very sharp and thin edges. The Canny operator works in a multi-stage process. Canny edge detection uses linear filtering with a Gaussian kernel to smooth noise and then computes the edge strength and direction for each pixel in the smoothed image. This is done by differentiating the image in two orthogonal directions and computing the gradient magnitude as the root sum of squares of the derivatives. The gradient direction is computed using the arctangent of the ratio of the derivatives. Candidate edge pixels are identified as the pixels that survive a thinning process called non-maximal suppression. In this process, the edge strength of each candidate edge pixel is set to zero if its edge strength is not larger than the edge strength of the two adjacent pixels in the gradient direction. Thresholding is then done on the thinned edge magnitude image using hysteresis. In hysteresis, two edge strength thresholds are used. All candidate edge pixel values below the lower threshold are labeled as non-edges, and the pixels values above the high threshold are considered as definite edges. All pixels above low threshold that can be connected to any pixel above the high threshold through a chain are labeled as edge pixels.

The schematic of the canny edge detection is shown in Figure 5.

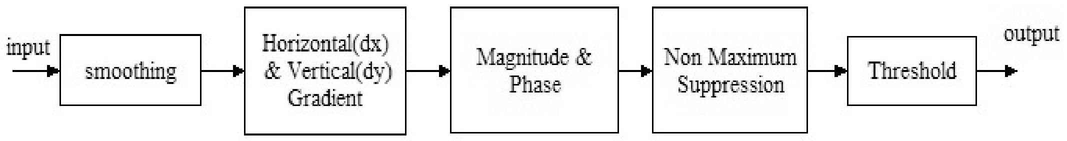


Figure 5: Schematic of canny edge detection

Smoothing

In the first stage the 5x5 Gaussian convolution mask of standard deviation (σ = 1.4) described in Section 2.3 is used for smoothing. The effect of Gaussian convolution is to blur an image. The degree of smoothing is determined by the standard deviation of the Gaussian.

Gradient Calculation

After smoothing the image and eliminating the noise, the next step is to find the edge strength by taking the gradient of the image. Most edge detection methods work on the assumption that an edge occurs where there is a discontinuity in the intensity function or a very steep intensity gradient in the image as shown in Figure 6. Most edge-detecting operators can be thought of as gradient-calculators. Because the gradient is a continuous-function concept and images are discrete functions, we have to approximate it. Since derivatives are linear and shift-invariant, gradient calculation is most often done using convolution. Numerous kernels have been proposed for finding edges, some of the kernels are: Roberts Kernel, Kirsch Compass Kernel, Prewitt Kernel, Sobel Kernel, and many others.

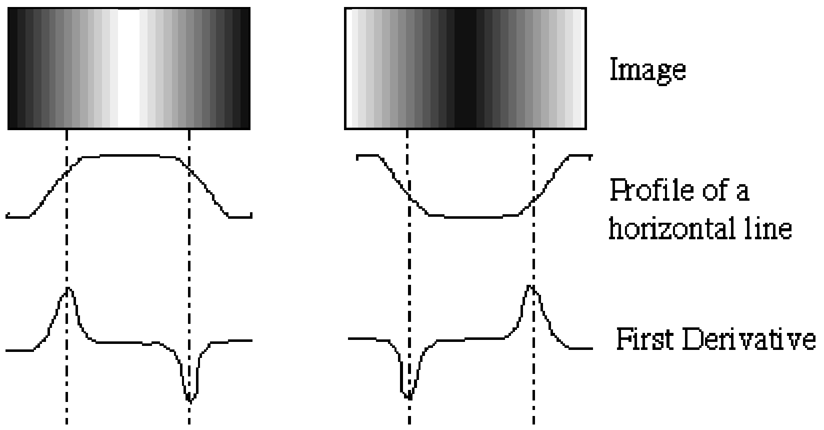
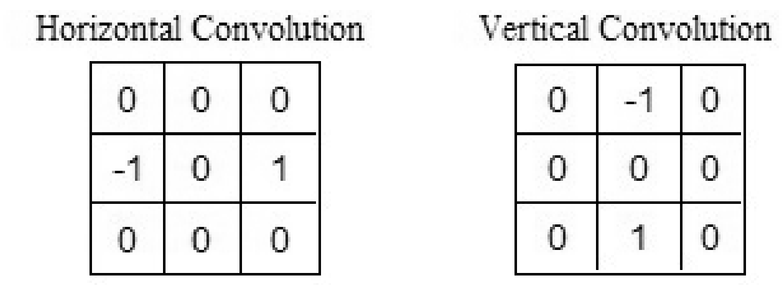


Figure 6: Gradient of image

The Prewitt kernels are based on the simple idea of the central difference between rows for horizontal gradient and difference between columns for vertical gradient.

$$\frac{\partial I}{\partial x} \approx \frac{I(x+1,y)-I(x-1,y)}{2} \quad ,and \quad \frac{\partial I}{\partial y} \approx \frac{I(x,y+1)-I(x,y-1)}{2} \tag{2.5}$$

The following convolution masks are derived from equations.



Horizontal Convolution

0	0	0
-1	0	1
0	0	0

Vertical Convolution

0	-1	0
0	0	0
0	1	0

Figure 7: Gradient of image

These convolutions are used for calculating the horizontal and vertical gradients.

Magnitude and Phase

Convolution of the image with horizontal and vertical gradients produces horizontal gradient (dx) and vertical gradient (dy) respectively. The absolute gradient magnitude (|G|) is calculated by the mean square root of the horizontal (dx) and vertical (dy) gradients. That is, $|G| = \sqrt{dx^2 + dy^2}$. To reduce the computational cost of magnitude, it is often approximated with absolute sum of the horizontal and vertical gradients ($|G| = |dx| + |dy|$).

The direction of the gradient (θ) is calculated by arctangent of the vertical gradient to the horizontal gradient:

$$\theta = \arctan(dy / dx) \tag{2.6}$$

Since arctangent is a very complex function and also requires floating point numbers, it is very difficult to implement such functions on FPGA. Instead, the value and sign of the components of the gradient is analyzed to calculate the direction of the gradient. If the current pixel is $P_{x,y}$ and the values of the derivatives at that pixel are dx and dy, the direction of the gradient at P can be approximated to one of the sectors shown in the Figure 8.

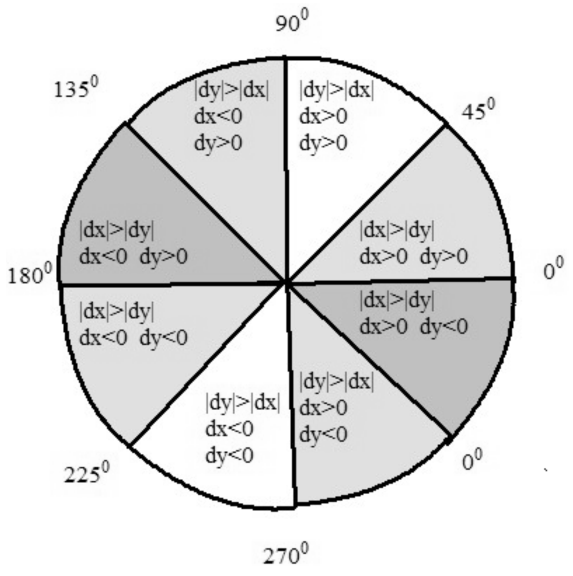


Figure 8: Gradient Orientation

Non-Maximum Suppression

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. For example, if the approximate direction of the gradient is between 0^0 and 45^0, the magnitude of the gradient at $P_{x,y}$ is compared with the magnitude of the gradient at adjacent points as shown in Figure 9.

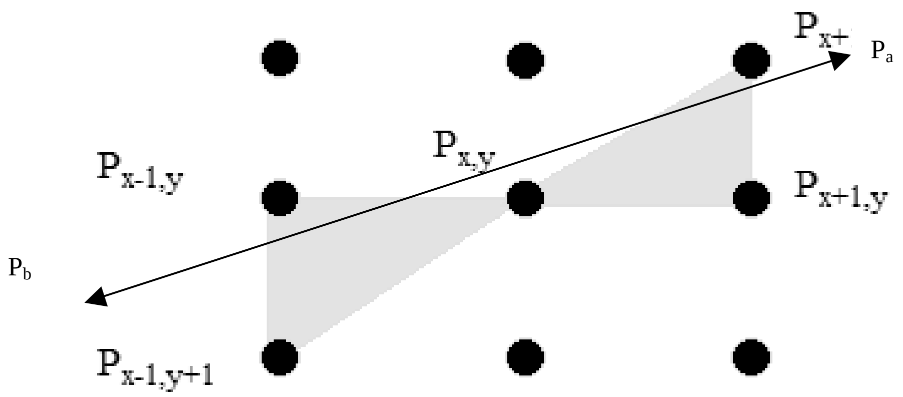


Figure 9: Pixel Interpolation

where $P_{x,y} = |dx_{x,y}| + |dy_{x,y}|$. The values of the gradient at the point P$_a$ and P$_b$ are defined as follows:

$$P_a = \frac{P_{x+1,y-1} + P_{x+1,y}}{2}, where\ P_{x+1,y-1} = |dx_{x+1,y-1}| + |dy_{x+1,y-1}|\ and\ P_{x+1,y} = |dx_{x+1,y}| + |dy_{x+1,y}| \qquad (2.7)$$

$$P_b = \frac{P_{x-1,y+1} + P_{x,y+1}}{2}, where\ P_{x-1,y+1} = |dx_{x-1,y+1}| + |dy_{x-1,y+1}|\ and\ P_{x,y+1} = |dx_{x,y+1}| + |dy_{x,y+1}| \qquad (2.8)$$

The center pixel $P_{x,y}$ is considered as an edge , if $p_{x,y} > p_a$ and $p_{x,y} > p_b$. If *neither* condition is not satisfied then the center pixel is eliminated.

Threshold

The output image of non-maximum suppression stage may consist of broken edge contours, single edge points which contribute to noise. This can be eliminated by thresholding with *hysteresis*. Two thresholds are considered for hysteresis, one high threshold other low threshold. If any edge response is above a high threshold, those pixels constitute definite edge output of the detector for a particular scale. Individual weak responses usually correspond to noise, but if these points are connected to any of the pixels with high threshold, they are more likely to be actual edges in the image. Such connected pixels are treated as edge pixels if their response is above a low threshold.

To get thin edges two thresholds (high threshold (T$_H$) and low threshold (T$_L$)) are used. If the gradient of the edge pixel is above the T$_H$, it is considered as an edge pixel. If the gradient of the edge pixel is below T$_L$ then it is unconditionally set to zero. If the gradient 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$_L$.

4. 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.

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 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 DK2 development environment. DK2 does not provide synthesis, and the suite must be used in conjunction with a synthesis tool to complete the design flow from idea to hardware. The Handel-C implementation is compared with SystemC language at simulation level and with Verilog synthesis level.

SystemC is a system design language that provides a consistent modeling approach and unified foundation for software and hardware development. It fills the gap between traditional hardware-description languages (HDLs) and software-development methods based on C/C++. SystemC comprises C++ class libraries and a simulation kernel used for creating behavioral- and register-transfer-level designs. The simulation of SystemC is conducted using the open source SystemC compiler.

Verilog is a hardware description language (HDL) used to model electronic systems. The language (sometimes called Verilog HDL) supports the design, testing, and implementation of analog, digital, and mixed-signal circuits at various levels of abstraction. The language differs from a conventional programming language in that the execution of statements is not strictly linear. A subset of statements in the language is synthesizable. If the modules in a design contain only synthesizable statements, software can be used to transform or synthesize the design into a netlist that describes the basic components and connections to be implemented in hardware. The netlist may then be transformed into a form describing the standard cells of an integrated circuit (e.g. an ASIC) or a bitstream for a programmable logic device (e.g. a FPGA).

Handel-C implementation 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.

RC1000 PCI board manufactured by Celoxica is used for our hardware implementation. The RC1000 is a PCI bus plug-in card for PC’s. It has one large Xilinx Virtex E FPGA, four 2MB banks of memory for data processing operations, a programmable clock and 50 auxiliary I/Os. All four memory banks are accessible by both the FPGA and any device on the PCI bus. A FPGA has two of its pins connected to clocks. One pin is connected to either a programmable clock or an external clock. The programmable clocks are programmed by the host PC, and have a frequency range of 400kHz to 100MHz. The RC1000 FPGA can be programmed from the host PC over the PCI bus.

5. Hardware Implementation

The algorithms implemented in this work use the moving window operator. The moving window operator usually processes 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 3x3 moving window is used for the median filtering, morphological and edge detection algorithms and a 5x5 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 (Figure 10). 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 less associated logic for reading and writing.

For a 3x3 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 x M). To access all the values of the window for every clock cycle the two FIFO buffers must be full. Figure 10, shows the architecture of the 3x3 moving window. For every clock cycle, a pixel is read from the RAM and placed into the bottom left corner location of the window.

Median Filter

A median filter is implemented by sliding a window of odd size on an image. A 3x3 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 3x3 window operation discussed in Sub-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 [16].

Morphological Operations

The basic morphological operators are erosion and dilation. The erosion and dilation of a grayscale image are called grayscale erosion or dilation. This grayscale erosion is performed by minimum filter operation and the dilation is performed by maximum filter operation. In a 3x3 minimum filter operation, the center pixel is replaced by a minimum value of the pixels in the window. In the maximum filter operation, the center pixel is replaced by a maximum value of the pixels in the window. The implementations of minimum and maximum filter operations are similar to the implementation of median filter.

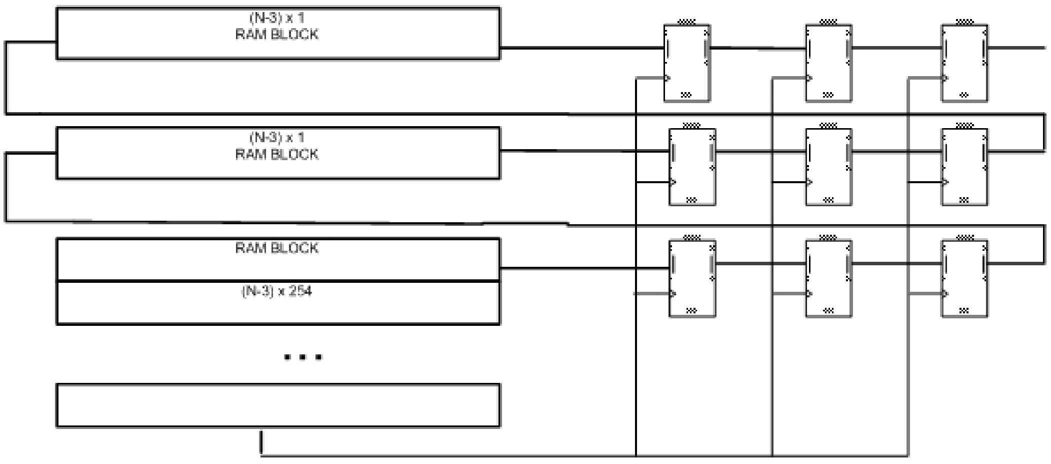


Figure 10: Hardware implementation of a 3 x 3 moving window operator

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. Therefore, to perform a 3x3 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 work consists of four stages:

- Image smoothing;
- Vertical and Horizontal Gradient Calculation;
- Directional Non Maximum Suppression;
- Threshold.

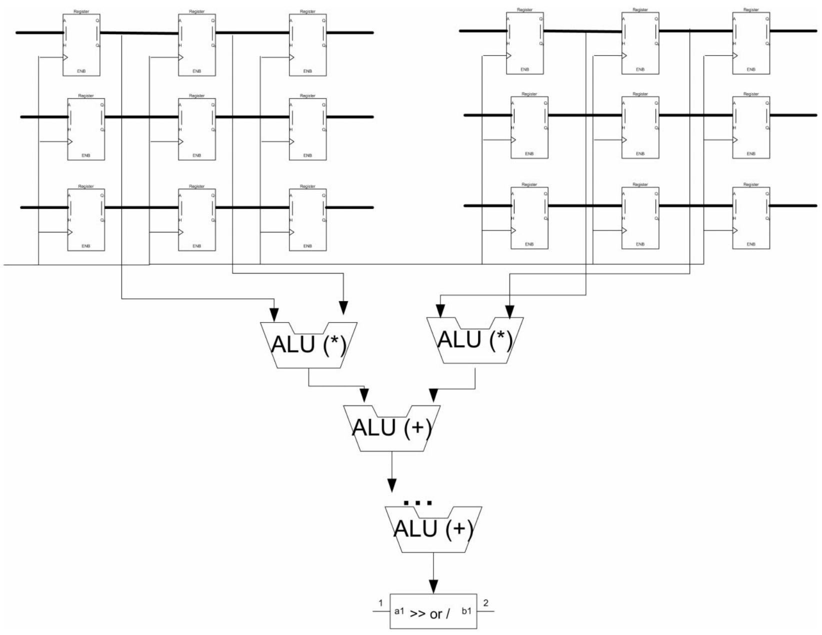


Figure 11: Architecture for Convolution operation

Normally, the edge detection algorithm is implemented by first applying the Gaussian smoothing algorithm to the entire image. 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 Figure 9 the magnitude and phase calculation stage and non-maximum suppression stage are combined to directional non-maximum stage. A pipelined architecture shown in Figure13 is designed.

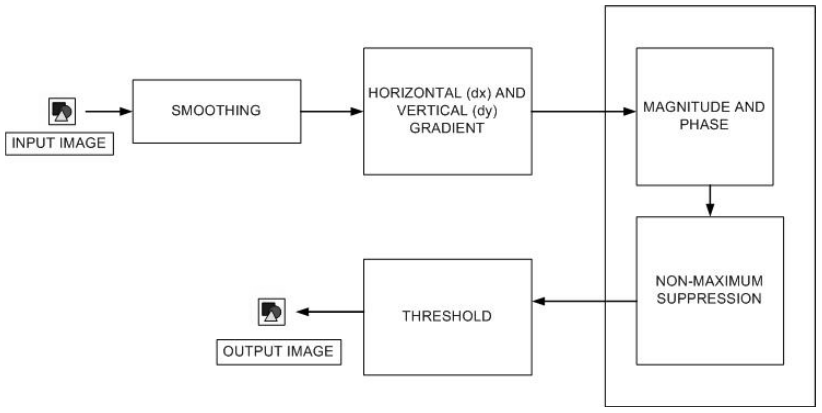


Figure 12: Block diagram of Canny edge detection Algorithm

Image Smoothing
Smoothing of the image is achieved by 5x5 Gaussian convolutions. A 5x5 moving window operator is used, four FIFO buffers are employed to access all the pixels in the 5x5 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 3x3 convolution kernels shown in Section 2.3. 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 3x3 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 3x3 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 thresholds, 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 3x3 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.

Figure 13: Architecture of Canny edge detection algorithm

5. Results and Conclusions

The image processing algorithms discussed above were modeled in Handel-C using the Celoxica's DK2 environment. The design was implemented on RC1000-PP Xilinx Vertex-E FPGA based hardware. The execution time and maximum frequency of operation of the hardware for the image processing algorithms on a 256x256 size grayscale "Lena" image is shown in Table 1. The table also compares the execution time and frequency of operation with that of the software implementation in C language. The speed of our FPGA solution for the image processing algorithms is approximately 15 times faster than the software implementation. The division operation in the Gaussian Convolution has been implemented using the division hardware and using shift registers and their performances tabulated. The multiplication operation in the Gaussian smoothing has been the multiplication hardware and Constant Coefficient Multiplier (KCM).

Table 1: Timing Result edge detection algorithm on 256 x 256 gray scale image

		Xilinx Vertex-E FPGA		Pentium III @ 1300 MHz	
		Freq [MHz]	Time [ms]	Freq [MHz]	Time [ms]
Median Filter, Morphological Operation		25.9	2.56	-	51
Gaussian Convolution	Direct division by 115	25.9	2.62	-	31
	Division using right shift(>> 7)	42	1.57	-	31
Gaussian Smoothing	Direct Multiplication	42.03	1.58	-	16
	LUT based Multiplication	50.99	1.31	-	16
Edge Detection		16	4.2	-	47

Table 2: Implementation Cost on FPGA

Design	LUTs	Flip-Flops	Block RAMs	CLB Slices	I/O Blocks
Median, Erosion & Dilation	406	298	2 (1%)	652 (3%)	145 (35 %)
5x5 Convolution div by 115	1040	926	4 (2%)	994 (5%)	145 (35%)
5x5 Convolution div by shift (>> 7)	597	802	4 (2%)	1035 (5%)	145 (35%)
3x3 Convolution direct multiplication	735	442	2 (1%)	479 (2%)	145 (35%)
3x3 Convolution LUT based Multiplication	384	428	2 (1%)	479 (2%)	145 (35%)
Edge Detection	945	807	4 (2%)	1820 (10%)	145 (35%)

Table 2 lists the detailed implementation cost on hardware and Figure 15 shows the output of hardware implemented images using Handel-C.

Table 3, compares the architecture proposed with the standard C implementations and with design implemented using SAC language [35]. The table shows that the execution time obtained by the proposed method is less than other implementations. However, the size of the image and the type of the image used in implementing Canny edge detection using SAC language is unknown.

A comparison of hardware area achieved for Canny's edge detection using Handel-C and Verilog is also performed and tabulated in Table 4. The HandelC implementation used the Block SelectRAM (BRAMs) are dedicated blocks of memory that can store large amounts of data. Each memory block is four CLBs high and is organized into memory columns stretching the entire height of the chip. There is one such memory column between every twelve CLB columns. The block SelectRAM also includes dedicated routing to provide an efficient interface with both CLBs and other block SelectRAM. The Verilog implementation does not utilize this

Block SelectRAM. Another important point is that during Verilog implementation, the logic and memory resource usage was maximized to achieve better timing performance.

Table 3: Experimental results with time and speed Comparisons

	Handel-C	PC @ 1300MHz	SA-C	PC @ 800MHz	PC with MMX @ 800MHz
FPGA CLK	16 MHz	-	32.2 MHz	-	-
Execution Time	4.2 ms	47 ms	6 ms	850 ms	135.8 ms

Table 4: Experimental results with area comparisons

FPGA Resources	VERILOG	HANDEL-C
Speed (VirtexE)	Not available	16 MHz
Slices	58%	10%
Slice Registers	55%	3%
LUTs	57%	5%
IOB	9%	35%
GCLKs	100%	50%
Block RAMs	-	6%
Equivalent Gate Count	306,236	127, 176

Table 5, summarizes the design efforts for implementing the Canny’s edge detection algorithm for Verilog, SystemC and Handel-C and their simulated results on the “lena” image are shown in Figure 14 (a), (b) and (c) respectively. The characteristics considered for design efforts are, 1) Design Time which include the time spent for coding, simulation and debugging the application, 2) Compilation Time, the time required for complete simulation of a 256 x 256 image, 3) Program Size, is the number of lines of code for the respective implementations.

Table 5: Comparing design efforts for Canny Edge Detection using Verilog, SystemC and Handel-C

	VERILOG	SystemC	HANDEL-C
Design Time	56 hrs	112 hrs	56 hrs
Compilation Time	54 Sec	5 mins	50 Sec
Program Size (lines)	1318	3239	1233

In conclusion, Handel-C performance for canny edge detection at simulation and synthesis level is better than SystemC and Verilog implementations. However, Handel-C language was introduces as language for Software Engineers to quickly prototype software concepts in hardware using behavioral model. Our current efforts to efficiently implement image processing algorithms in hardware using high level languages, ascertains that hardware knowledge and implementation at Register Transfer Level (RTL) are necessary. However, the class of image processing algorithms considered in our work is limited to basic algorithms and further work on complex image processing algorithms need to be performed.

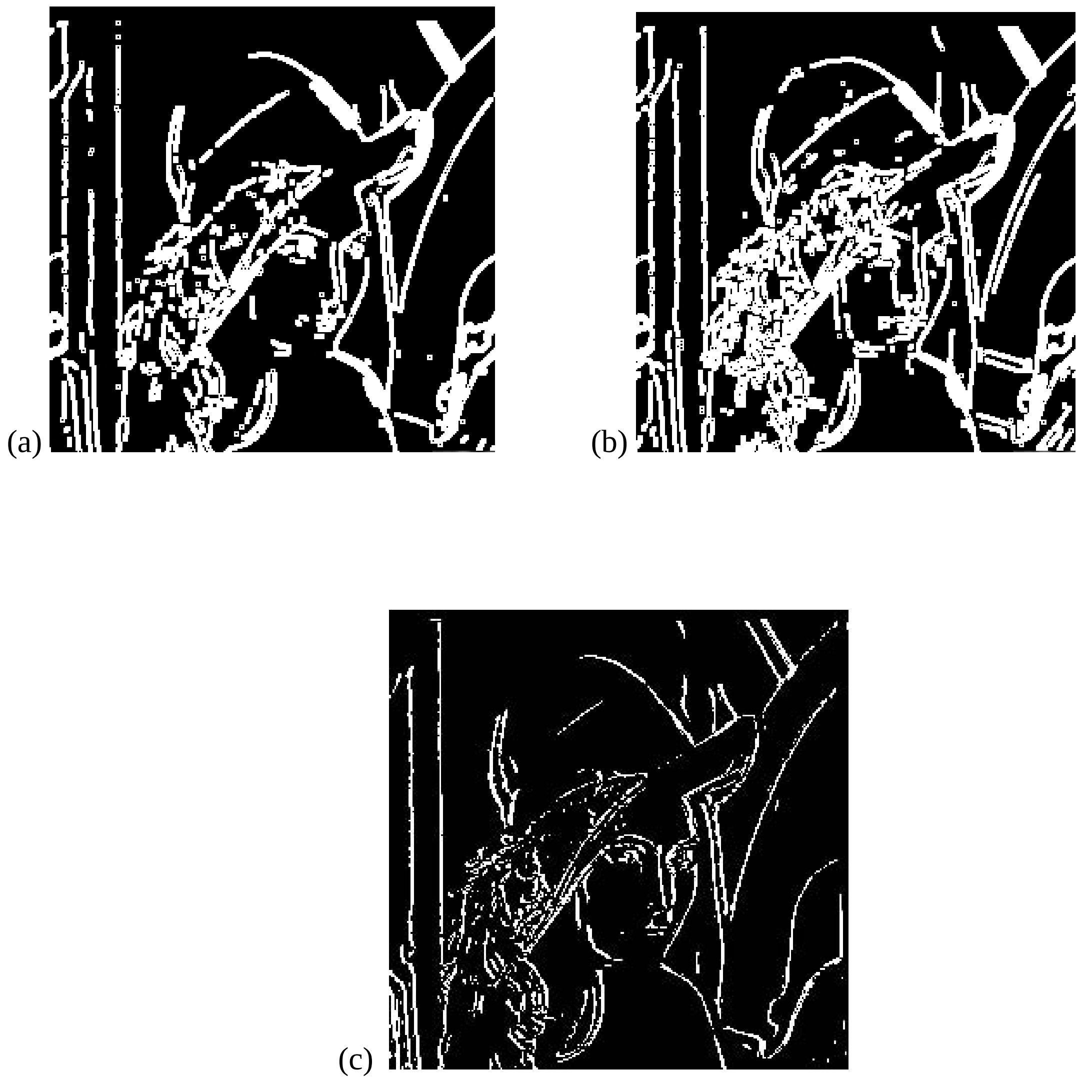
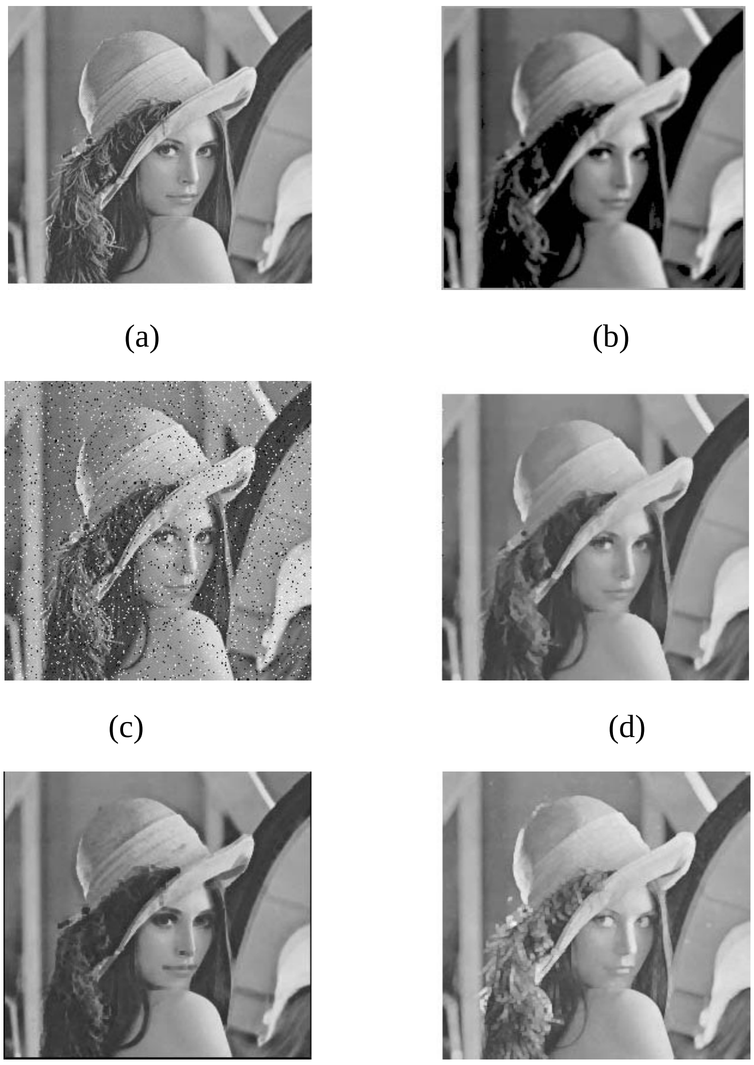


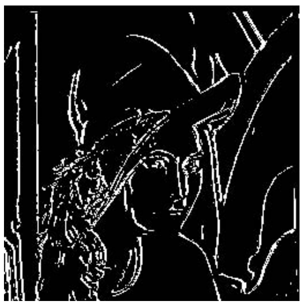
Figure 14: Comparison of Canny's edge detection algorithm simulated and synthesized in a) Verilog, b) SystemC and c) Handel-C



(a) (b)

(c) (d)

(e) (f)



(g)

Figure 15: (a) Original Image, (b) Gaussian convolution, (c) Salt and pepper noise, (d) Median filter, (e) Erosion, (f) Dilation, (g) Edge detection

References

[1] John C. Ross. Image Processing Hand book, CRC Press. 1994.

[2] Peter Mc Curry, Fearghal Morgan, Liam Kilmartin. Xilinx FPGA implementation of a pixel processor for object detection applications. In the Proc. Irish Signals and Systems Conference, Volume 3, Page(s):346 – 349, Oct. 2001.

[3] M. Moore. A DSP-based real time image processing system. In the Proceedings of the 6th International conference on signal processing applications and technology, Boston MA, August 1995.

[4] Stephen D.Brown, R.J. Francis, J.Rose, Z.G.Vranesic. Filed Programmable Gate Arrays, 1992.

[5] Celoxica, http://www.celoxica.com. Handel-C Language Reference Manual, 2003. RM-1003-4.0.

[6] Richard Shoup. Parameterized Convolution Filtering in a Field Programmable Gate Array Interval. Technical Report, Palo Alto, California .1993.

[7] ATMEL Application Note. 3x3 Convolver with Run-Time Reconfigurable Vector Multiplier in Atmel AT6000 FPGAs. AT6000 FPGAs Application Note 1997.

[8] F.G.Lorca, L Kessal and D.Demigny. Efficient ASIC and FPGA implementation of IIR filters for Real time edge detection. In the International Conference on image processing (ICIP-97) Volume 2. Oct 1997.

[9] Fahad Alzahrani and Tom Chen. Real-time high performance Edge detector for computer vision applications. In the Proceedings of ASP-DAC ,1997, pp 671-672.

[10] V.Gemignani, M. Demi, M Paterni, M Giannoni and A Benassi. DSP implementation of real time edge detectors. In the Proceedings of speech and image processing pp 1721-1725,2001.

[11] Nelson. Implementation of Image Processing Algorithms on FPGA Hardware. Masters Thesis, Graduate School of Vanderbilt University, 2000.

[12] Shinichi Hirai, Masakazu Zakouji, Tatsuhiko Tsuboi, Implementing Image Processing Algorithms on FPGA-based Realtime Vision System, Proc. 11th Synthesis and System Integration of Mixed Information Technologies (SASIMI 2003), pp.378-385, Hiroshima, April, 2003.

[13] *Peter Baran, Ralph Bodenner and Joe Hanson.* **Reduce Build Costs by Offloading DSP Functions to an FPGA.** FPGA and Structured ASIC Journal.

[14] J. Maddocks & R. Williams. VHDL Image Processing Source Modules REFERENCE MANUAL. Nov 2003.

[15] Altera, Edge Detection Using SOPC Builder & DSP Builder Tool Flow, ver 1.0, Application Note 377, May 2005.

[16] Hong Shan Neoh and Asher Hazanchuk, Adaptive Edge Detection for Real-Time Video Processing using FPGAs, GSPx 2004 Conference, 2004.

[17] B. Draper, W. Bohm, J. Hammes, W. Najjar, R. Beveridge, C. Ross, M. Chawathe, M. Desai, J. Bins. Compiling SA-C Programs to FPGAs: Performance Results. Int. Conf. on Vision Systems, Vancouver, July 7-8, 2001. p. 220-235.

[18] Peter Mc Curry, Fearghal Morgan, Liam Kilmartin, Xilinx FPGA Implementation of an Image Classifier for object detection applications. IEEE Signal Processing Society, International conference on image processing, Thessaloniki, Greece, 2001 pp. 346-349

[19] W. Luk and T. Wu and I. Page. Hardware-Software Codesign of Multidimensional Programs. In the Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, April 1994, pp.82-90, Napa, CA

[20] Trost, A. Zemva, B. Zajc, Programmable System for Image Processing, Field-Programmable Logic and Applications, Elsevier, pp. 490-494, 1998.

[21] Trost, A. Zemva, B. Zajc, FPGA Based Rapid Prototyping System for Image Processing Circuits. In Proc. of Int. Workshop on Intelligent Communications and Multimedia Terminals, COST, pp. 55-59, Ljubljana, 1998.

[22] J.Canny. A computational approach to the edge detection. IEEE Trans Pattern and Machine Intelligent. Vol PAMI-8 1986 pp 679-698.

[23] Venkateshwar Rao Daggu. Design and Implementation of an Efficient Reconfigurable Architecture for Image Processing Algorithms using Handel-C. Masters Thesis, UNLV, 2003.

[24] M. Budiu and S. C. Goldstein. Compiling application-specific hardware. In Proc. FPL, LNCS 2438, pp. 853–863, Montpellier, France, 2002.

[25] S. A. Edwards. The challenges of hardware synthesis from C-like languages. In Proc. IWLS, Temecula, California, June 2004.

[26] D. D. Gajski, J. Zhu, R. D¨omer, A. Gerstlauer, and S. Zhao. SpecC: Specification Language and Methodology. Kluwer, 2000.

[27] D. Galloway. The Transmogrifier C hardware description language and compiler for FPGAs. In Proc. FCCM, pp. 136–144, Napa, CA, 1995.

[28] T. Grotker, S. Liao, G. Martin, and S. Swan. System Design with SystemC. Kluwer, 2002.

[29] T. Kambe et al. A C-based synthesis system, Bach, and its application. In Proc. ASP-DAC, pp. 151–155, Yokohama, Japan, 2001.

[30] D. C. Ku and G. De Micheli. HardwareC: A language for hardware design. T.R. CSTL-TR-90-419, Stanford University, CA, Aug. 1990.

[31] P. Schaumont et al. A programming environment for the design of complex high speed ASICs. In Proc. DAC, pp. 315–320, 1998.

[32] D. Soderman and Y. Panchul. Implementing C algorithms in reconfigurable hardware using C2Verilog. In Proc. FCCM, pp. 339–342, 1998.

[33] E. Stroud, R. R. Munoz, and D. A. Pierce. Behavioral model synthesis with cones. Design & Test of Computers, 5(3):22–30, July 1988.

[34] K.Wakabayashi. C-based synthesis experiences with a behavior synthesizer, “Cyber”. In Proc. DATE, pp. 390–393, 1999.

[35] J. Hammes, R. Rinker, W. Najjar, B. Draper. A High-level, Algorithmic Programming Language and Compiler for Reconfigurable Systems. The 2nd International Workshop on the Engineering of Reconfigurable Hardware/Software Objects (ENREGLE), part of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, NV, June 26-29, 2000.

[36] Draper, R. Beveridge, W. Böhm, C. Ross and M. Chawathe. "Implementing Image Applications on FPGAs," International Conference on Pattern Recognition, Quebec City, Aug. 11-15, 2002.