International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-2, Issue-1, January 2014

Review on Image Processing: FPGA Implementation perspective

Mohassin Ahmad, Abdul Gaffar Mir and Najeeb-ud-din Hakim

Abstract— **Digital image processing (DIP) is an ever growing area with a variety of applications including medicine, video surveillance, and many more. In order to improve the performance of DIP systems image processing algorithms are implemented in hardware instead of software. The idea here is mainly to obtain a system faster than software image processing. Image processing tasks such as filtering, stereo correspondence and feature detection are inherently highly parallelizable. Thus FPGAs (Field Programmable Gate Arrays) can be a useful approach in the area of Digital Signal Processing. FPGAs provide advantage of the parallelism, low cost, and low power consumption. They are semiconductor devices that contain a number of logic blocks, which can be programmed to perform anything from basic digital gate level techniques, to complex image processing algorithms. This paper provides an overview of the various works that demonstrate the benefits of using FPGAs to implement image processing algorithms like median filter, morphological, convolution, smoothing operation and edge detection, etc. Gray-level images are very common in image processing. These types of images use eight bits to code each pixel value, which results in 256 different possible shades of grey, ranging from 0 (black value) to 255 (white value). Latest generations FPGAs compute more than 160 billion multiplication and accumulation (MAC) operations per second.**

Index Terms— **FPGA, Digital Image Processing (DIP), algorithms**

I. INTRODUCTION

Digital image processing is the processing and display of images. Image processing is used for the modification of the image. There are three main categories of image processing: Image enhancement, image restoration, and image classification. Image enhancement provides more effective display of data for visual interpretation. It helps a user to view the image and recognize different segments of an image. An example of this is to edit the shades in an image. This technique is very useful for assisting with distinction of different objects in an image. Rectification and restoration of an image is another important aspect of image processing. It deals largely with image correction, which may be necessary due to the image being affected by geometric distortion or noise. It can also remove blurring whereby a poor quality image may be upgraded to one with better quality and distinguishable features. Image classification is where images are classified based on colors or shapes present in the image. This can be useful in order for a computer to differentiate between different types of images.

There are many useful applications of image processing. It is used as remote sensing for robot guidance, and target recognition. It is also used for industrial inspection, and in medical technology such as X-Ray enhancement. A very useful application of digital image processing is to view the various color intensities present in an image and split the image into segments based on the results. The biggest performance bottleneck is the time involved in processing the images captured by the camera. 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. This leads to explore possible hardware based alternatives. Recently, 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 DIP in hardware. FPGAs are often used as implementation platforms for real-time image processing applications. A Field Programmable Gate Array (FPGA) is a programmable (or reconfigurable) device [1] in which the final logic structure can be directly configured by the end user. An FPGA consists of an array of uncommitted elements that can be programmed or interconnected (or configured) according to a user's specification in a virtually limitless number of ways. Being reprogrammable and easily upgradable, an FPGA offers a compromise between the flexibility of general-purpose processors and the hardware based speed of ASICs. They allow rapid prototyping of a system and offer an inexpensive option to validate system requirements [2]. Placing the functionality of image processing applications onto hardware allows faster processing as it is no longer necessary to split the individual instructions into fetch, decode and apply cycle needed in the typical processing unit of a computer.

In this paper a survey implementation of image processing applications on FPGAs with an emphasis on the salient features of FPGAs has been presented. The rest of the paper is organized as follows. Section II highlights the advantages & limitations of FPGAs. Section III details algorithm mapping and window based operators. Section IV describes the various filtering algorithms like convolution filtering, median filtering, etc. Section V describes histogram based algorithms. In section VI, motion based algorithm have been described. Finally, section VII summarizes prior research in the FPGA implementation of image processing algorithms.

Manuscript received January 17, 2014.

Mohassin Ahmad, Research Scholar, Department of ECE, National Institute of Technology, Srinagar, J & K, India (e-mail: mohassin623@gamil.com).

Abdul Gaffar Mir,Associate Professor, Department of ECE, National Institute of Technology, Srinagar, J & K, India.

Najeeb-ud-din Hakim, Professor, Department of ECE, National Institute of Technology, Srinagar, J & K, India.

II. EVALUATION OF FPGAS AS PLATFORM FOR DEVELOPING DIP APPLICATIONS

A. Advantages of FPGAs

Many advantages of FPGAs make them a preferred choice of implementation in DIP realm. Based on the survey, many significant features have been found which are as follows:-

a) A characteristic of many image-processing methods is the multiple iterative processing of data sets such as four stages of canny edge detector, which require performing multiple passes over the image. These steps, which have to be performed sequentially on a general-purpose computer, can be fused in one pass in FPGA, as their structure is able to exploit spatial and temporal parallelism. FPGA can perform multiple image windows in parallel and multiple operations within one window also in parallel.

b) By employing several optimizations techniques such as Loop Fusion, Loop Unrolling etc efficient usage of FPGA resources and speed-up in implementations is possible by avoiding many redundant operations.

c) FPGAs are capable of parallel I/O, which allows them to perform read (from memory), process and write (to memory) simultaneously. Many operations such as convolutions, finding square root etc can be executed much faster by using pipelining and parallelism.

d) All of the logic in an FPGA can be rewired, or reconfigured, with a different design as often as the designer likes. This type of architecture allows a large variety of logic designs dependent on the processor's resources), which can be interchanged for a new design as soon as the device can be reprogrammed.

e) FPGAs provide the flexibility to reprogram and upgrade to new standards. Easy Upgradeability ensures that FPGAs solutions evolve quickly with no risk of obsolescence.

f) The reusability and efficiency of hardware implemented on FPGA, is especially useful in developing Image Processing IP (intellectual property) as it allows an efficient system in terms of cost & performance. Possibility of quick integration of the IP blocks without a need of modification or repetition of verification cycle [3] simplifies debugging and thus greatly reduces the time-to-market.

g) Because of its LUT based architecture, some convolution masks (such as constant coefficient multiplier or KCMs) can be implemented very efficiently [4].

h) High computational density in FPGA together with a low development costs allows even the lowest volume consumer market to bear the development costs of FPGAs. In fact, compared to ASICs, FPGAs are especially useful in a lower volume type of application. With low-cost FPGAs high definition solutions can now be implemented for less than US$1.00 per 1,000 logic elements (LEs)

A lot of research has been recently done on utilizing FPGAs as development platform for DIP algorithms. In this paper the related work in the area has been presented.

B. Limitations of FPGAs

On the other hand, there are also the limitations of FPGAs for image processing applications.

a) There are many overheads in FPGA design. This include data transfer times which is the time required to upload (or download) the data, from (or to) reconfigurable processor to(or from) host; time for reconfiguration.

b) FPGAs are excellent choice only for those algorithms which don't use floating -point mathematics or complex mathematics. Division, direct multiplication etc are very complex and expensive on FPGA. Hence, the designers have to reformulate their algorithms & avoid complex mathematics (e.g. implementing a divide by 8 using the bit shifting method of division instead of a divide by 9).

c) Current FPGAs cannot be reconfigured quickly as the process of modifying or combining FPGA circuits is also laborious.

d) The size of memory that can be implemented using standard logic cells on an FPGA is limited, as implementing memory is an inefficient use of FPGA resources.

e) Routines where complex tasks cannot be broken down into simpler tasks must perform a more serial method of processing, which is not entirely efficient with FPGAs.

f) Hardware offers much greater speed than a software implementation, but it comes with a price of increased development time inherent in creating a hardware design. Most software designers are familiar with C, but in order to develop a hardware system, one must either learn a hardware design language such as VHDL or Verilog, or use a software-to-hardware conversion scheme, such as Streams-C, which converts C code to VHDL, or MATCH, which converts MATLAB code to VHDL

III. ALGORITHM MAPPING

Algorithms for image processing are normally classified into one of three levels: low, intermediate or high. Low-level algorithms operate on individual pixels or neighborhoods [5]. Intermediate-level algorithms either convert pixel data into a different representation, such as a histogram, coordinate or chain code, or operate on one of these higher representations. High-level algorithms aim to extract meaning from the image using information from the other levels. This could be in the form of rejecting a component or identifying where an object is within an image. When moving from low to the high-level representations there is a corresponding decrease in exploitable parallelism due to the change from pixel data to more descriptive representations. However there is also a reduction in the amount of data that must be processed, allowing more time to do the processing. Due to their structure, FPGAs are most appropriate for use with computationally intensive tasks which form the vast majority of low and intermediate-level operations. The large data sets and regular repetitive nature of the operations can be exploited. For this reason it has been traditional in many systems for the FPGA to handle the low-level operations and then pass the processed data to a microprocessor which then executes the high-level operations. With increasing FPGA size, it is now possible to implement processor cores on the reconfigurable fabric, which means the FPGA can form the core of the system.

Low-level image processing operators can be classified as local operators (point operators, window Operator) s and global operators, with respect to the way the output pixels are computed from the input pixels [6].The local operators depend on data from a relatively small neighborhood that is local in spatial and temporal dimensions. Examples include thresholding, convolution, and motion estimation. Global operators depend on data from the entire image. Examples include transforms like the fast Fourier transform and principle component analysis as well as statistical histogram techniques.

A. Window-based Image Operators

Window-based operators need only partial or local information about the image, that is they are is restricted to a small neighbor of image data centered on a reference pixel A window-based image operator is performed when a window with an area of $w \times w$ pixels is extracted from the input image and it is transformed according to a window mask or kernel, and a mathematical function produces an output result [7]. The window mask is the same size as the image

window and their values are constant through the entire image processing & the function is applied independently at all pixel locations. The values used in the window mask depend on the specific type of features to be detected or recognized. Usually a single output data is produced by each window operation and it is stored in the corresponding central position of the window as shown in Fig 1.

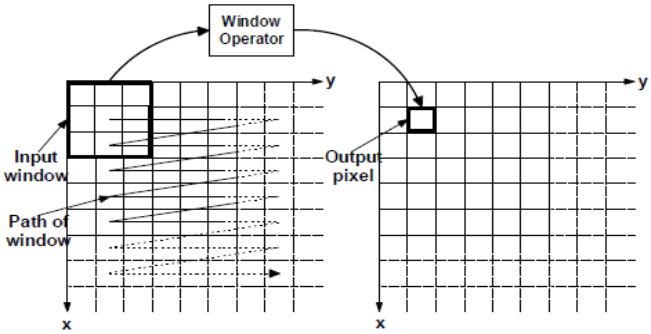

Fig 1: Schematic representation of a window based operation

A moving window is used for implementing various image processing algorithms like median filtering, morphological, edge detection algorithms and in Gaussian smoothing operation. The window moves in raster-scan order. Window based operators are also inherently parallelizable and hardware implementations can greatly accelerate their implementation. [8].

For processing purposes, the straightforward approach is to store the entire input image into a frame buffer, accessing the neighborhood pixels and applying the function as needed to produce the output image. Under this approach, every pixel in the image is read several times. Memory bandwidth constraints make obtaining all these pixels each clock cycle impossible. Input data from the previous N-1 rows can be cached using a shift register (or circular memory buffer) when the window (say size of N×N) is scanned along subsequent lines. Instead of sliding the window across the image, the above implementation now feeds the image through the window. Introducing the row buffer data structures adds additional complications but reduces the redundant transfers. With the use of both caching and pipelining there needs to be a mechanism for adding to the row buffer and for flushing the pipeline. The size of the row buffer is given as W-N, where M is the width of the image and W the size of the window (NxN).

Another complication occurs when the window extends outside the image boundary. There are several options for dealing with this; the simplest is to assume one row wraps into the next. A better option is to replicate the edge pixels of the closest border [9]. Such image padding can be considered as a special case of pipeline priming. When a new frame is received the first line is pre-loaded into the row buffer the required number of times for the given window size. Before processing a new row the first pixels are also pre-loaded the required number of times, as is the last pixel of the line and the last line. Fig 2 shows the implementation of the 3×3 Window Operations. 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. To access all the values of the window for every clock cycle the two row buffers must be full. For every clock cycle, a pixel is read from the RAM and placed into the bottom right corner location of the window.

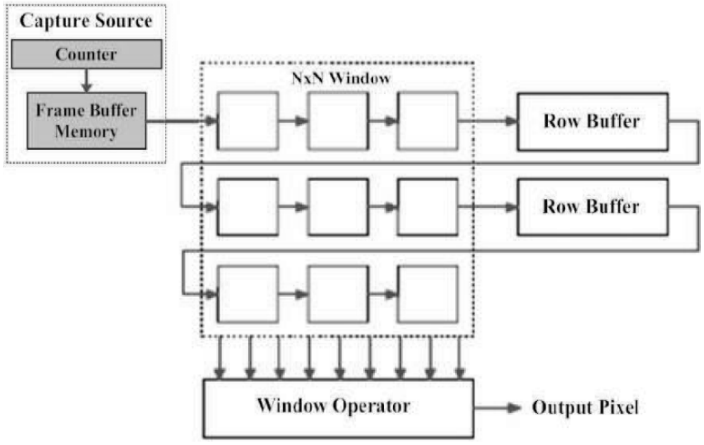

Fig 2: Window Operation using Buffers

The architecture in Fig 2 places the lowest demand on external memory bandwidth, but the highest demand on internal memory bandwidth. Each pixel in external memory is accessed only once but for an image width, W, and kernel width, N, the FPGA must store ((N-1)*W + N) pixels on-chip, because many modern FPGA devices have a large amount of on-chip memory. To get the optimal design, three constraints are considered. The first is the area constraint; the number of slices in the FPGA limits how many copies of processing elements we can program on the chip. Second, memory bandwidth defines the maximum data transfer speed between the FPGAs and the external memory banks. Third, the size of the on chip memory that is used as buffers reduces redundant data transfer.

IV. IMAGE PROCESSING ALGORITHMS

There has been a lot of research in the field of image processing in hardware by utilizing FPGAs. The image processing algorithms considered for hardware implementation include: convolution, image filtering and edge detection (Sobel's, Prewitt's and Canny's edge detection). These algorithms are based on window-based processing.

A. Convolution Filtering

Many image processing operations such as scaling and rotation require re-sampling or convolution filtering at each pixel in the image. Convolution is a standard operation often used for filtering (blurring, sharpening, compression, noise cleaning) or re-sampling (compositing, scaling, rotation, warping, texture mapping), etc.The convolution algorithm can be calculated in the following manner. For each input pixel window, the values in that window are multiplied by the convolution mask. Next, those results are added together and divided by the number of pixels in the window. The output function at each pixel is the weighted sum of neighboring pixels given by Equation (1)

$$F(i, j) = \sum_{k,l \in K} W_{k,l} * \text{Image}\ (i - k, j - l) \qquad (1)$$

Where *W* represents the convolution filter kernel.

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. By selecting the appropriate weights; convolution can implement low-pass, high-pass and band-pass frequency domain filters used extensively in image enhancement and feature extraction.

Low-pass filters use positive weights and are used for image smoothing. High pass filters use a kernel with a positive center weight and negative outer weights and are used to enhance high frequency components in an image such as edges and fine detail. If the filter size is m x m, then each output pixel depends on m^2 adjacent pixels, and thus m^2 multiplies and adds are required at each site. Therefore, high performance can be achieved by exploiting parallelism.

There are various ways to implement a convolution filter. The convolution algorithm uses adders, multipliers, and dividers to calculate its output. On FPGAs, use of mathematics tends to slow down performance. Many designers favor techniques that reduce the algorithm's dependency on complex mathematics. Another obstacle in this algorithm's design was implementing the capability to handle negative numbers. In a proper convolution algorithm, the mask can (and often does) consist of negative numbers. Effectively, the VHDL had to be written to handle these numbers by using *signed* data types. Signed data simply means that a negative number is interpreted into the 2's complement of its non -negative dual. This means that all vectors within the design must use an extra bit as compared to unsigned numbers. The extra bit always carries the sign of the number – 0 for a positive number, 1 for a negative number. Addition and multiplication were instantiated using simple + and * signs in the VHDL code. Since a proper convolution involves a division by the number of pixels in the window, some thought had to be put into this part of the algorithm's hardware implementation. Hardware dividers on FPGAs are quite large and slow. So, instead of division the bit shifting method is used. Fig 3 shows a graphic representation of the mathematics of the hardware convolution. Note that a valid output for the convolution algorithm occurs six clock cycles after the first window is valid. Since this design is pipelined and will run in the megahertz range, this kind of startup latency has very little effect on overall design speed [10], [11].

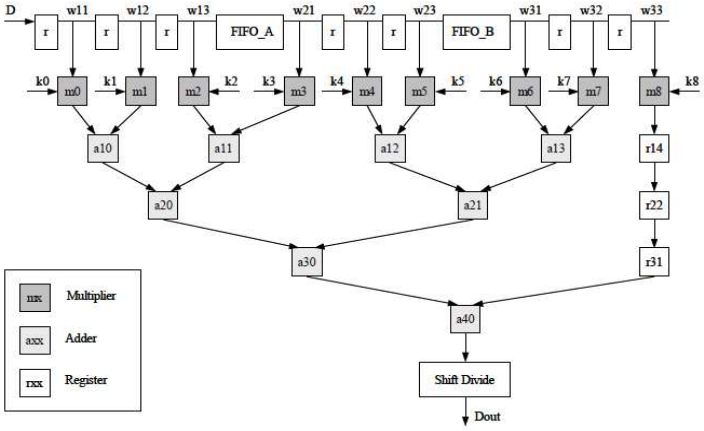

Fig 3: Hardware Design of Convolution

The main problem in implementing and computing convolution is speed, area and power which affect the image processing system. Implementing the algorithm in parallel hardware will speed up the process but the implementation itself is very complex and requires a huge silicon area Optimization of the convolution algorithm can be easily achieved if one has limited kernel specifications. For example, if all coefficients in the kernel are powers of two, the VHDL synthesizer is able to result in a design that uses fewer resources. This is due, of course, to the way numbers are represented in digital systems, where a number that is a power of two is represented with only one bit. Further optimization is possible by reducing the bit widths of the kernel constants. This is result in a smaller coefficient data range, but this compromise may be acceptable in certain cases.

B. Median Filtering

Median filtering is a powerful instrument used in image processing. The median filter is a non linear filter which is commonly used to remove impulsive noise from images, while preserving edges and other details. Two common types of impulsive noise are salt and pepper noise, and random-valued noise. Impulsive noise replaces the intensities associated to a certain percentage of pixels by the maximum or minimum possible intensity (salt and pepper noise) or by any value between the maximum and minimum intensity (random-valued noise). A median filter is effective in removing this type of noise without affecting the distinguishing characteristics of the signal. A median filter can outperform linear low-pass filters for which also smooth out edges and other details which are present in the original image. A standard median operation is implemented by sliding a window of odd size (e.g. 3x3window) over an image [12]. 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 Fig 4.

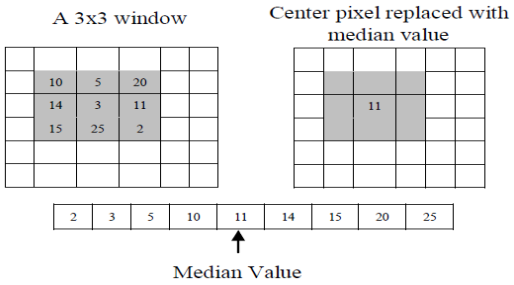

Fig.4: Median Filter.

Median filtering is usually based on data sorting algorithms, including bubble sort, quick sort, and insertion sort. Several techniques based on these algorithms have been proposed in the literature for implementing median filters on hardware in these sorting schemes, the incoming pixels pass through a network of comparators and swapping units. The comparators compare two to three incoming pixels at once, while the swapping unit sorts them accordingly. It can be mentioned at this point that to find median of a sequence of size (2N+1) using bubble sort requires N(2N+1) sorting units and (2N+1) registers. As the window size increases, the number of compare-and-swap unit's increases significantly. In order to do this properly, a counter must be used to tell the output data-valid signal when to change to its 'on' state. Since it is desired that the output image be the same size as the input image, and use of the window generator effectively reduces the amount of valid output data, borders with zero value pixels must be place around the image. In order to do this properly, the counters are used to tell the algorithm when the borders start. A VHDL counter was written to count pixel movement as the data streams into the entity. Since images are two-dimensional data, two counters were needed: one to count rows and one to count columns in the image. Optimization techniques may lead to some reduction in the number of these units. The parallel strategy leads to a significant reduction compared to the wave sorter approach. In this strategy, it is necessary to consider total number of required steps to sort an array that is the steps used to read data from memory and the steps required to store the sorted data back to memory [13]. With this kind of method, data can be stored in the array by sending a datum to the first register and later, when the second datum is sent to the first register, the value on the first array is shifted to the second register. The necessary number of steps for sorting is equal to the number of characters in the biggest group of identical characters divided by 2. The parallel sorting strategy is shown in Fig 5. Each node is a two element sort, with the

lower input exiting the node on the left, the higher input leaving on the right.

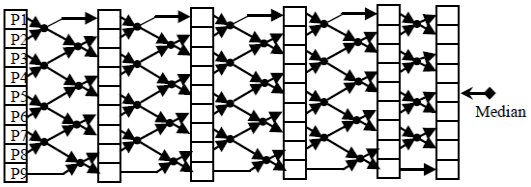

Fig 5: Structure of the parallel sorting strategy

Table I shows the results for the implementation of median filter window size 3×3. It is observed that as the image size increases, the processing time increases along with the percentage utilization of the resources of the FPGA. This increase is linear with the size of the image after the image size exceeds 64×64 pixels. However the percentage increase in the CLBs utilized grows nonlinearly

Table I: Synthesis result for median filter with window size 3×3 using Xilinx device

Image size	Processing time (m sec)	Utilization of slices (in percentage)
4×4	0.00246	4(2000 out of 4656)
16×16	0.11766	4(221 out of 4656)
32×32	0.54006	5(237 out of 4656)
64×64	2.306	5(250 out of 4656)
128×128	9.525	5(259 out of 4656)

C. Morphological algorithm

The morphological algorithm 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 reconstruction, edge detection, restoration and texture analysis. Morphological operators are defined as combinations of basic numerical operations taking place over an image by using a structuring element. The structuring element is a window that scans over an image and modifies it according to some specified rule. The most basic morphological operations are dilation and erosion. Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the structuring element used to process the image. In the morphological dilation and erosion operations, the state of any given pixel in the output image is determined by applying a rule to the corresponding pixel and its neighbors in the input image.

Rules for Dilation and Erosion:

Erosion- The value of the output pixel is the minimum value of all the pixels in the input pixel's neighborhood. In a binary image, if any of the pixels is set to 0, the output pixel is set to 0 based on the logical AND relationship [14]. Erosion can be used to eliminate unwanted white noise pixels from an otherwise black area. It allows a white pixel will remain white in the output image is if all of its neighbors are white.

Dilation- The value of the output pixel is the maximum value of all the pixels in the input pixel's neighborhood. It uses the NAND rather than the AND logical operation. Being the opposite of erosion, dilation will allow a black pixel to remain black only if all of its neighbors are black. This operator is useful for removing isolated black pixels from an image. Binary erosion and dilation masks are:

1	1	1		0	0	0
1	1	1		0	0	0
1	1	1		0	0	0

Erosion Dialation

The grayscale erosion is performed by minimum filter, whereas the dilation is performed by maximum filter. In a 3 x 3 minimum filter, the centre pixel is replaced by a minimum value of the pixels in the window. In a maximum filter, the centre pixel is replaced by a maximum value of the pixels in the window. The implementation of minimum & maximum filters is similar to the median filters implementation. Morphological Dilation of a grayscale image is illustrated in the Fig 6:

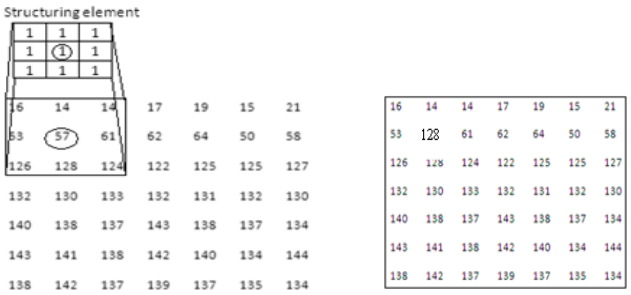

Fig 6: Input image Output image

V. HISTOGRAM BASED ALGORITHMS

Image enhancement involves techniques to sharpen the image features such as edges, boundaries or contrast to make a graphic display more useful for display and analysis. There are various spatial domain image enhancement techniques like Median Filter, Contrast Stretching, Negative image transformation, Power law transformation and Histogram Equalization. Median filter has been already discussed, is a non-linear, low-pass filtering method, which are used mainly to remove salt-and pepper noise from an image. Contrast stretching attempts to improve an image by stretching the range of intensity values it contains to make full use of possible values. Contrast stretching is restricted to a linear mapping of input to output values. The negative transformation results in reversing of the grey level intensities of the image thereby producing a negative like image. Power law transformation is also called as gamma correction, which is given by the expression $s = cr^{\gamma}$. For various values of γ different levels of enhancements can be obtained. For achieving high performance image enhancement methods are implemented on FPGAs.

A. Histogram Equalization

Histogram equalization is one of the commonly used image enhancement tecniques. Histogram equalization is considered to be most popular because of its simplicity and better performance on all types of images. Histogram equalization is a transformation that stretches the contrast by redistributing the gray level values uniformly [15]. Digital images are represented as two dimensional pixel arrays. Each pixel indicates the brightness or color of the image at a given point. Suppose we have an image which is predominantly dark. Then its histogram would be concentrated towards the lower end of the grey scale and all the image detail is compressed into the dark end of the histogram. If we could 'stretch out' the grey levels at the dark end to produce a more uniformly distributed histogram then the image would become much clearer. Histogram equalization creates an image with equally distributed brightness levels over the whole brightness scale (0-255). It maximizes the overall contrast; a nearly uniform (i.e. flat)

distribution is produced. The histogram equalization algorithm is implemented on FPGAs which provide finer flexibility and powerful computing capability. Fig 7 shows the Block Diagram of Histogram Equalization algorithm.

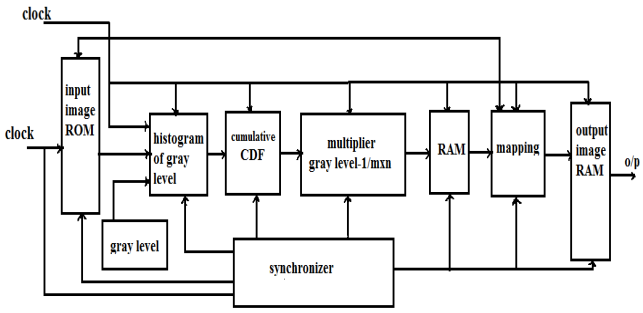

Fig 7: Block Diagram of Histogram equalization algorithms

The ROM is initialized by image. Histogram of gray level block will counts the occurrence of each pixel value (gray value block) in the image in 1-D array. Cumulative block will count each value in the array with previous one, multiplier block will multiply CDF(cumulative distribution function) array with constant value, Then mapping block will mapping each pixel in unmodified image to corresponding value in new matrix. The minimum time period in this system is 5ns for a test image of 100×100. It has been found that the computation speed improved further by considering optimization in FPGA implementation systems.

B. Thresholding technique

The histogram–based method [16] can be used to determine the threshold value for image binarization. For extracting useful information from an image this algorithm divides the pixels into two groups --background and foreground objects then the optimum threshold is obtained. All pixels above a determined (threshold) grey level are assigned to the object, and all pixels below that level are assumed to be outside the object (background).The object pixel is given a value of "1" while a background pixel is given a value of "0" which gives the binary output from thresholding. The selection of the threshold level is very important, as it will affect any measurements of parameters concerning the object, such as poor contrast, inconsistency between sizes of object and background, non-uniformity in the background, and correlated noise. The threshold level is normally taken as the lowest point in the trough between the two peaks or the mid-point between the two peaks may be chosen. For hardware implementation of the thresholding method speed and complexity have to be considered and these conditions are met by using FPGAs. The hardware architecture is based on a weight-based clustering threshold algorithm in which the gray level pixels of an image are divided into two clusters, foreground and background. The hardware includes three major modules; memory controller unit, weight-updating unit, and thresholding unit. The modules and their interconnections are illustrated in Fig 8.

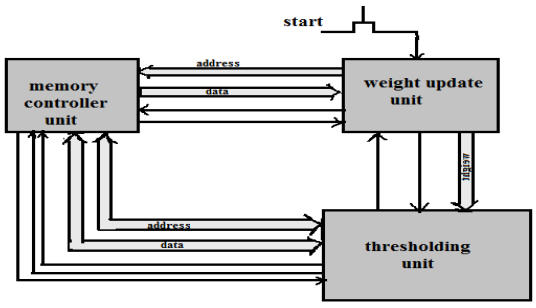

Fig 8: Thresholding block diagram

The "memory controller unit" generates timing for control signals to send/receive image pixels data to/from the memory. The "weight-updating unit" is in fact an arithmetic processor to calculate the weights and threshold values. Each input pixel is read from memory, compared with the weights and the closer weight is updated. The update is done based on the difference between the input pixel and the weight, scaled by a learning rate factor. Once a complete frame of the image is processed, the center of background and foreground clusters is computed. "Thresholding unit" determines the threshold value by averaging the weights. Then every single pixel of the same image is fetched from memory via memory controller unit. Each read pixel is compared to the threshold value, and the result is written back to the memory.

In this algorithm we need real numbers for some parts of the numerical computation. These non-integers, such as floating point allow a wide range of values to be represented. But floating-point arithmetic units consume significantly greater hardware resources than the integer arithmetic and this make it more suitable for million gates FPGA like Xilinx Virtex series. Of course so many enhancement and optimization has been proposed for the real numbers. Since the resource of the current FPGA device is limited and because the focus of this algorithm is not on the high precision of the numbers, all numbers are represented in integer and an approximation is applied for the arithmetic.

C. Edge Detection algorithm

An edge in an image is a contour across which the brightness of the image changes abruptly. However, image data is discrete, so edges in an image often are defined as the local maxima of the gradient [17].The edges of image are considered to be most important image attributes that provide valuable information for human image perception. The edge detection refers to algorithms which aim at identifying points in a digital image at which the image brightness changes sharply. The basic edge-detection operator is a matrix area gradient operation that determines the level of variance between different pixels. Examples of gradient-based edge detectors are Roberts, Prewitt, and Sobel operators. All the gradient-based algorithms have kernel operators that calculate the strength of the slope in directions which are orthogonal to each other, commonly vertical and horizontal. Then the different components of the slopes are combined to give the total value of the edge strength.

These algorithms consist in a 2-D first derivate operator applied to the grey-scale image to highlight regions of the image with high first spatial derivates. The edges are translated into ridges in the gradient magnitude of the image. The algorithm tracks along the top of these ridges and sets to zero all pixels that are not actually on the ridge top and give a thin line in the output. The edge detection of an image is the convolution products of the image pixels with different masks which result in the calculation of the horizontal and the vertical gradient. The two gradients are calculated using differences between adjacent pixels.

Prewitt Edge Detection

One way to find edges is to use the Prewitt kernels. The Prewitt kernels are based on the idea of the central difference and give equal weightage to all pixels when averaging. The vertical and horizontal kernel for the prewitt algorithm is given in Fig 9.

1	0	-1
1	0	-1
1	0	-1

Gx

1	1	1
0	0	0
-1	-1	-1

Gy

Fig 9: Prewitt Edge Detector – Horizontal & Vertical kernel.

These convolutions are applied to the grey-scale image to get the horizontal (Gx) and the vertical gradients (Gy). These kernels can then be combined together to find the absolute magnitude of the gradient at each point. These kernels are, however, sensitive to noise. The gradient magnitude is given by:

$$|G| = \sqrt{Gx^2 + Gy^2} \; ; \quad |G| = |Gx| + |Gy| \qquad (2)$$

Gradient direction is given as,

$$\Theta = \tan^{-1}(Gy/Gx) \qquad (3)$$

Sobel Edge Detector

The Sobel algorithm provides a differencing as well as noise smoothing operation in the single kernel. Thus, noise sensitivity of first gradient based operations can be avoided by the use of this algorithm. The Sobel operator only considers the two orientations which are 0 and 90 degrees. The vertical and horizontal kernels for the Sobel algorithm are given in Fig 10.

1	0	-1
2	0	-2
1	0	-1

Gx

1	2	1
0	0	0
-1	-2	-1

Gy

Fig 10: Sobel Edge Detector – Horizontal & Vertical kernel.

With Sobel operator the lines corresponding to edges becomes thickens compared with the Roberts Cross and Prewitt operator output due to the increased smoothing of the Sobel operator [18]. The FPGA allows implementation of these algorithms with parallel architecture. The hardware architecture is shown in the Fig 11.

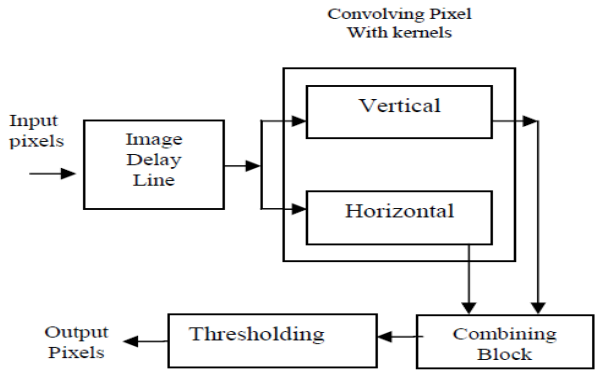

Fig 11: Hardware Architecture of edge detection algorithm

The architecture consists of four major blocks. First the input pixels are passed through the Image delay line. Image delay line shifts the incoming image pixels through line buffers to create a delay line. Buffer depth depends on the number of pixels in each line. The number of buffer lines depends on the size of the convolution kernel. The pixels are forwarded to the vertical and horizontal kernel. The kernel performs convolution operation with the pixels and the result of which is forwarded to the combining block .This block combines the results of horizontal and vertical convolution. The final output of this block is the sum of the absolute values of the results of horizontal and vertical convolution. The output of the combining block is the output may consist of some spurious noise. By controlling the threshold value in the thresholding block the effect of noise can be reduced. Finally the output pixels are taken from the Thresholding block.

VI. MOTION BASED IMAGE ALGORITHMS

A. Sum of Absolute Differences (SAD) Algoritm

The Sum of Absolute Differences (SAD) algorithm is a simple area base image matching technique for determining a correlation between two images [19]. It is used in stereo imaging, where two cameras are used to image a scene from two different locations so that a physical point appears in different locations in each camera image. This algorithm compares a window in one image, with every possible window in the other image. The relative pixel offset between a window area and its best match (greatest correlation) gives a value of stereo disparity. This is repeated for every window of the initial image, with a greater disparity indicating that the object is closer to the cameras. The hardware required to capture stereo images can be implemented inexpensively and with greater computational performance by using FPGAs. This algorithm can be formulated as a window-based operator, though some aspects must be considered:

a) The coefficients of the window mask are variable and new windows are extracted from the first image to constitute the reference block. Once the processing in the search area has been completed, the window mask must be replaced with a new one, and the processing goes on the same way until all data is processed.

b) The different windows to be correlated are extracted in a column-based order from the search area to exploit data overlapping and sharing. The pixels are broadcasted to all the processors to work concurrently.

When the SAD value is processed, data is available in a row format therefore when blocks are processed vertically; previous read data in

the search area are overlapped for two block search SAD algorithm is shown in Fig 12.

The pixel values of the left and right stereo image are subtracted, and the absolute value of these differences is taken. This absolute value is then summed along each three pixel column and row. When these operations are completed, each PE contains the 3x3 SAD values for that disparity. The disparity is then increased (shift one of the images across by one pixel), and the operations repeated. The disparity that gave the lowest SAD will become that pixel's value in the final disparity map. The generated disparity map then gives an indication of the relative distance of each image pixel from the cameras.

1	4	6
-1	8	9
2	-3	-2

a)

1	4	6
1	8	9
2	3	2

b)

↓	↓	↓
4	15	17
↑	↑	↑

c)

-	-	-
→	36	←
-	-	-

d)

Fig 12: SAD Algorithm a) Left and Right Image Differences b) Absolute Value of Left and Right Image Differences c) Column SAD d) Sum of Absolute Differences for a Single Disparity Level

B. Background Subtraction Algorithm

Background subtraction is the most used algorithm for detecting motion in images. The main target of motion detection process is to segment the foreground pixels that belong to the moving objects [20] .To achieve this there are several approaches for (a) the background subtraction, (b) the temporal difference of two successive frames and (c) the optical flow. The background subtraction approach detects the moving regions by subtracting the current frame (pixel by pixel) from a reference image called background. On the other hand, the second approach has the same principle of the background subtraction method, but in this case the reference image is the previous frame instead of the background method. The third method is based on the fact that the object motion information is contained in the brightness changes of the image.

The three described methods have good performance for motion detection problem, however, optical flow is a very complex algorithm (it is necessary to store more than one image), requiring high memory resources. On the other hand, the background subtraction and the temporal difference are low cost algorithms. However, temporal difference has problems for detecting the object's shape. Therefore, a background subtraction based motion detection system has low computational cost, high performance.

In background subtraction method firstly, each image of sequence is subtracted from background. Then the resulting image from the subtraction is segmented in order to produce a binary image that represents the moving regions on the image. Mathematically, the background subtraction algorithm can be defined by the following equation (4)

$$d(x,y,t) = \begin{cases} 1 \text{ if } \left| f(x,y,t) - B(x,y) \right| > T_d \\ 0 \text{ otherwise} \end{cases} \qquad (4)$$

Where Td is a predetermined threshold, f(x, y, t) is an image taken at time t and B(x, y) is the reference image (or background). In the dynamic image analysis, all pixels in the motion image d(x, y, t) with value "1" are considered as moving objects in the scene.

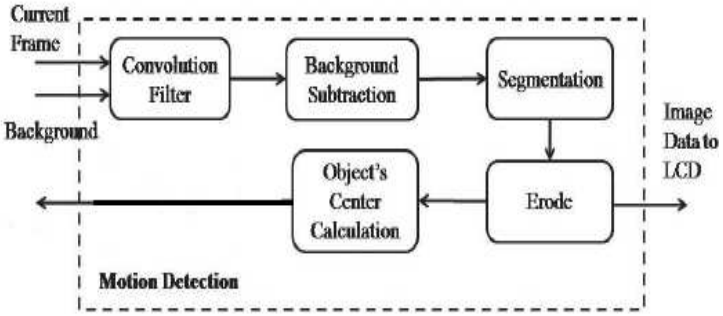

Fig 13: Motion detection architectutre by background subtraction

The background subtraction algorithm is implemented after the filtering stage. The filter is applied to both current frame and background images and consists of a low-pass filter. After this, the filtered current frame is subtracted from the filtered background; the absolute value of resulting pixels is calculated. Then the output image is segmented in order to generate a binary image, where pixels tagged with "1" belong to the moving object and pixels tagged with "0" belong to background. After that the erosion operation is performed in order to eliminate noise generated by the segmentation operation and the resulting image of this stage is displayed on an LCD. This approach is shown in Fig 13.

VII. CONCLUSION

A number of image processing algorithms have been discussed such as generic convolution, filtering, gray-level image morphology, matrix multiplication, Gaussian filtering, etc. The image processing algorithms are inherently parallel and are often implemented with long sequences of basic operations. The high performances of image processing algorithms have been achieved by implementing them on FPGAs. The performance comparison with other existing conventional architectures confirms the promising advantages of the FPGA-based approaches, which combine the flexible hardware design with high parallelism. It is a convenient platform to develop and accelerate image processing applications under real-time constraints. The platform has proven to be capable of handling a large amount of data with low area utilization, to benefit from parallelism as well as to attain a higher data transfer using a reduced bus bandwidth. The execution time and maximum frequency of operation of the hardware for the image processing algorithms on a 256x256 size grayscale image is shown in Table 2 [21]. The table also compares the execution time and frequency of operation with that of the software implementation in C language.

Table 2: Timing Result edge detection algorithm on 256 x 256 gray scale images

Image Processing Algorithm	Xilinix Vertex E FPGA		Pentium III @ 1300 MHz	
	Freq [MHz]	Time [ms]	Freq [MHz]	Time [ms]
Median Filter, Morphological Operation	25.9	2.56	-	51
Convolution Operation	25.9	2.62	-	31
Smoothing method	42.03	1.58	-	16
Edge Detection	16	4.2	-	47

The speed of our FPGA solution for the image processing algorithms is approximately 15 times faster than the software implementation. While FPGAs are excellent for some uses, such as a large number of image processing applications, but there are difficulties in using more complex mathematics for dedicated applications. So a lot of research needs to be done in this field to provide the optimal designs. Furthermore, the most demanding operation of image processing chain is the image filtering, which makes a strong use of the convolution operation. The main problem in implementing convolution is speed, area and power. Several FPGA design approaches have reduced power, hardware resources and area significantly. Still the higher level techniques are needed and future work will aim at further improving the design and trade-off between various parameters.

REFERENCES

[1] S. D. Brown, R. J. Francis, J. Rose, Z.G.Vranesic, "Field-Programmable Gate Arrays," *Kluwer Academic Publishers, Boston*, 1992.

[2] Sparsh Mittal, Saket Gupta and S. Dasgupta, "FPGA: An Efficient and Promising Platform for Real-Time Image Processing Applications," *Proceedings of the National Conference on Research and Development in Hardware & Systems (CSI-RDHS)*, 2008.

[3] Rahul V. Shah, "Image Processing Applications on New Generation FPGAs," *eInfochips Ltd.*, March 7, 2006.

[4] K. T. Gribbon, D. G. Bailey, A. Bainbridge-Smith, "Development Issues in Using FPGAs for Image Processing," *Proceedings of Image and*

Vision Computing New Zealand, pp. 217–222, Hamilton, New Zealand, December 2007.

[5] Griselda Saldaña-González and Miguel Arias-Estrada, "FPGA Based Acceleration for Image Processing Applications," ISBN :978-953-307-026-1, December 2009.

[6] C. T. Johnston, K. T. Gribbon, D. G. Bailey , "Implementing Image Processing Algorithms on FPGAs," *Proceedings of the Image and Vision Computing New Zealand Conference 2003*, Massey University, Palmerston North, New Zealand, pp. 408-413, Nov. 2003.

[7] Cesar Torres-Huitzil, Miguel Arias-Estrada, "FPGA-Based Configurable Systolic Architecture For Window-Based Image Processing," *EURASIP Journal on Applied Signal Processing*, Vol. 7, ISSN:1024–1034, 2005

[8] Haiqian Yu, Miriam Leeser, "Automatic Sliding Window Operation Optimization for FPGA- Based Computing Board," *14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2006.

[9] Donald G Bailey, "Image Border Management for FPGA Based Filters,"*Sixth IEEE International Symposium on Electronic Design, Test and Application*, 2011.

[10] Nelson, "A Further Study of Image Processing Techniques on FPGA Hardware," *Independent Study Pape*r, May 2000.

[11] Stefano Di Carlo, Giulio Gambardellay, Marco Indaco, Daniele Rolfoy, Gabriele Tiotto, Paolo Prinetto, "An Area-Efficient 2-D Convolution Implementation on FPGA for Space Applications," *Proceedings of the IEEE 6th International Design and Test Workshop (IDT)*, Vol. 7,Issue 3, 2011.

[12] Madhuri Gundam, Dimitrios Charalampidis, "Median Filter on FPGAs," *44th IEEE Southeastern Symposium on System Theory* , University of North Florida, Jacksonville, FL March 11-13, 2012.

[13] Elmoncef Benrhouma, Meddeb Souad, Abdulqadir Alaqeeli, Hamid Amiri, "Study and Design of Median Filter," *SIDOP 2nd Workshop on Signal and Document Processing*, 2012.

[14] R.Arunmozhi, G.Mohan, "Implementation of Digital Image Morphological Algorithm on FPGA using Hardware Description Languages," *International Journal of Computer Applications*, Vol. 57, No.5, pp 0975 – 8887, Nov. 2012.

[15] Stephanie Parker, J. Kemi Ladeji-Osias, "Implementing Histogram Equalization Algorithm in Reconfigurable Hardware," July 30, 2009.

[16] N Otsu, "A Threshold Selection Method from Gray-Level Histogram," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 9, N0. 1, 1979, pp.62-66.

[17] Sudeep K C and Dr. Jharna Majumdar, "A Novel Architecture for Real Time Implementation of Edge Detectors on FPGA," *International Journal of Computer Science* Issues, Vol. 8, Issue 1, January 2011

[18] G. Anusha, Dr.T. Jaya Chandra Prasad, Dr..D. Satya Narayana, "Implementation of SOBEL Edge Detection on FPGA," *International Journal of Computer Trends and Technology*, Vol. 3, Issue 3, 2012

[19] P. Greisen, S. Heinzle, M. Gross, and A. P. Burg, "An FPGA-based Processing Pipeline for High-Definition Stereo Video," *EURASIP Journal on Image and Video Processing*, Vol. 2011, p. 18, Nov. 2011

[20] C. Sanchez-Ferreira, J. Y. Mori, C. H. Llanos, "Background Subtraction Algorithm for Moving Object Detection in FPGA," *Proceedings of the 2012 IEEE Southern Conference on Programmable Logic (SPL)*,2012

[21] Daggu Venkateshwar Rao, Shruti Patil, Naveen Anne Babu and V Muthukumar, "Implementation and Evaluation of Image Processing Algorithms on Reconfigurable Architecture using C-based Hardware Descriptive Languages," *International Journal of Theoretical and Applied Computer Sciences*, Vol. 1, No. 1, 2006.