Chapter 4

Automatic Real-time Focused Region Extraction System

Our objective in this chapter is to describe the proposed and implemented area efficient hardware architecture for real-time extraction of focused regions in a live color video stream and its FPGA based implementation. The implementation done by us uses minimum FPGA resources and can be integrated with other complex computer vision applications running on the same FPGA platform. To achieve this, we have made a three-pronged approach. First, we have coded a hardware implementation friendly scheme for extraction of focused regions in color videos using C/C++ and OpenCV libraries (for video reading and result displaying). In this scheme, edge width information, which is a distinguishing feature between the focused/less blurred edges and the non-focused/highly blurred edges, has been used to extract the focused regions. Second, an area optimized VLSI architecture for the coded focused region extraction scheme has been proposed and designed which greatly economizes on FPGA resource usages (area) as compared to existing architectures in the literature. Third and most important, the designed area efficient architecture has been integrated with the camera interface module and the DVI display controller to develop a working prototype system for real-time extraction of focused regions in a live video stream for video surveillance applications.

4.1. Extraction of Focused Regions

Extraction of focused regions in a video scene is an important pre-processing step in automated video surveillance systems and is used for several applications like motion detection in focused regions, selection of frames of interest to reduce further processing overheads, storing only frames with relevant information to reduce memory requirements, and image compression and transmission of only relevant information to reduce communication overheads. All these applications rely on the most salient or focused regions in the video scene. From, the optical property of the lens system it is a well established fact that the object of interest is usually well focused, while the other foreground and background objects appears blurry. Consider the image in Figure 4.1a. In this image there are two objects: a person’s face and a metallic device. The camera is focused on the metallic
device and the person’s face is non-focused/blurred. It is observed from this image that only
the focused metallic device is the object of interest in this complete image. Therefore, it is
necessary to extract only the focused metallic device for further processing as shown in
Figure 4.1b. The focused object in this frame contains the key information of the captured
image, and extracting this focused object gives that information of interest. Therefore, we,
in this chapter, aim to design a focused region extraction system capable of automatically
extracting the quantitative information in real-time from an incoming live video stream and
to avoid the redundant information in the scene which is not useful for further processing.

![Image of focused and non-focused objects](image)

Figure 4.1: (a) Focused and Non-focused Objects Present in the Scene; (b) Extracted Focused Regions.

### 4.2. Literature Review

Numerous approaches have been proposed in the literature for extraction of focused
regions in a video scene. All these schemes differ from each other in terms of their
approach, speed of operation, and computational complexity. Dedicated attempts for
extracting the focused regions (objects of interest) in a scene have been made by Li et al.
[69] and Katto and Ohta [70] using multiple cameras. Use of multiple cameras provides
additional information for better extraction of focused regions. However, its computational
complexity is more as multiple images are processed to extract the focused regions. A filter
based approach has been proposed by Kao et al. [71] for focused region extraction using a
single camera. This algorithm works well but is time consuming. Yang et al. [72] have
proposed a cellular Neural Network (CNN) based approach which is less time consuming as
compared to algorithm proposed by Kao et al. [71]. Edge information based schemes for
segmenting focused objects in visual images have been proposed by Swain and Chen [73]
and Tsai and Wang [74]. In the algorithm proposed by Swain and Chen [73], the focused
moving objects have been segmented using edge width information and motion detection
(using background subtraction) information. In the Tsai and Wang [74] approach, first the gradient image is computed for the complete image, and then for each edge pixel in the gradient image, the portion of the sharp/focused edge region in a small neighborhood window is evaluated using moment-preserving technique. To enhance the accuracy for extraction of focused regions, some researchers have proposed some novel techniques using 3D depth information by using multiple differently focused images [75] or multiple camera inputs [76]. Exploring the use of high frequency wavelet coefficients and statistical features, Wang et al. [77] have proposed an algorithm to separate a sharply focused object of interest from other foreground or background objects. The authors have claimed that the algorithm has achieved high accuracy at higher speeds. The algorithm classifies the large image blocks into interest object and background according to high frequency wavelet coefficients. The algorithm is robust, but if the object of interest is highly smooth or the initial classification is incorrect, this scheme is likely to fail. Won et al. [78] and Kim [79] have proposed automatic segmentation algorithms for low depth of field images. Won et al. [78] have used a local variance image field for the representation of pixel-wise spatial distribution of high frequency components in images and they have claimed that their algorithm yields more accurate segmentation results than the multi-resolution wavelet-based segmentation. Kim [79] has used morphological filters and region merging techniques to segment focused objects. The algorithm is less efficient in terms of speed when it is applied to image sequences because of the use of time consuming morphological operation. Zhang et al. [80] have demonstrated a novel unsupervised algorithm for extraction of salient regions of interest by integrating the wavelet modulus maxima edges and the color regions of the mean shift based segmentation. The authors have claimed to achieve good and acceptable segmentation results. A block based algorithm for focused area extraction by blind deconvolution has been presented by Kovacs et al. [81]. The algorithm works well for ordinary images with no explicit knowledge about image or exposure conditions. Since this algorithm is block based, therefore, there is a significant reduction in the computation time needed for focused map extraction. Another efficient and fast focused region extraction block-based scheme was presented by Kim et al. [82]. This was based on the calculation of color-based higher order statistics (HOS). Li and Ngan [83] have proposed a novel focused region segmentation algorithm based on matting model. In their approach, a saliency map of the input image was generated by using the re-blurring model and then morphological filtering was employed to smoothen and accentuate the salient regions. This method is capable of segmenting the focused regions effectively and accurately as compared to the algorithm.
proposed by Kim [79]. Li et al. [84] have proposed a novel segmentation algorithm based on learning model for extracting attention region. In order to segment focused objects, the authors have used a two-level segmentation method, which includes region and pixel level segmentations and have claimed to have achieved effective segmentation results. Kim and Jeong [85] have proposed a new segmentation algorithm to separate focused foreground region from non-focused background region by computing the blur parameter for each pixel. Then, these parameter values are clustered to identify blurry regions and sharp regions using belief propagation algorithm. Finally, by using a certain threshold value the algorithm separates foreground image from the background image. Li and Ngan [86] have proposed a supervised method for focused region extraction. The framework of the proposed method was divided into three parts, namely training, testing, and post-processing. The authors have claimed that their method outperforms other state of the art methods for focused object segmentation but is computationally intensive as many steps are performed. A new method for extracting sharp regions of interest using a combination of edges and regions was proposed by Neverova and Konik [87]. The algorithm first classifies each edge as either sharp or blurred based on gradient profile width estimation. Then a mean shift over segmentation is allowed to label each region as either "sharp" or "blurred" using the density of marked edge pixels. For edge detection the authors have used the Canny edge detector.

From the existing literature, it is observed that a majority of the algorithms proposed over the years for finding focused regions in visual images (as discussed above) can be placed in one of the two categories depending on which technique they are based upon, namely edge-based or region-based. Edge-based techniques perform faster as compared to region based extraction techniques, as the computational complexity of edge-based techniques is lower. The accuracy of edge-based techniques is relatively low as compared to complex region-based (or combination of region-based and pixel-based) techniques.

The main purpose of an FPGA-based automated video surveillance system designed and implemented in this thesis is to filter the frames of interest for further processing and storage based on motion detection in focused regions of a scene in the live video stream and not to exactly extract the focused regions. Therefore, the important aspect is real-time performance and not relatively increased accuracy at the cost of highly increased computational complexity which results in system performance degradation (by increasing the time required to process one frame). Due to this reason, we have chosen to go with an edge based
focused region extraction scheme. Another reason for choosing an edge-based method is the constraint on FPGA resource utilization (area constraint). The focused region extraction hardware architecture is only a small part of the much larger FPGA-based automated video surveillance system (involving many other complex functions like motion detection, history generation, object tracking, etc), and therefore, must not consume a disproportionately large part of the FPGA resources.

4.3. Algorithm Developed for Extraction of Focused Regions

As stated earlier in this chapter, from the property of the optical system, it is a well established fact that the objects at a particular distance from the lens are focused, whereas objects at other distances are blurred by varying degrees depending on their distances. In the current generation cameras, the objects of interest in a scene can be focused irrespective of its distance from the camera. The camera can be focused either at the objects which are near to the camera or at the objects which are away from the camera. If the camera is focused at objects which are near to the camera, then as the distance increases, the imaged object becomes progressively more non-focused. In case the camera is focused at objects which are far from the camera, then as the distance decrease, the imaged object becomes progressively more non-focused. The edges of focused objects are sharp and concentrated, whereas the edges of background or foreground non-focused objects are blurry and scattered [74]. The distinguishing feature between focused/less blurred and non-focused/highly blurred edges is edge width. Focused edges have steeper intensity gradient and smaller widths than non-focused edges [73]. Therefore, by measuring the edge width (amount of blur) for edge pixels in the observed image, the points on the boundary of the focused object can be detected. Based on this concept, the edge-based focused region extraction algorithms have been proposed in the existing literature.

The basic concept for categorizing focused and non-focused regions in an image using edge-width based approach is illustrated in Figure 4.2. Consider the image shown in Figure 4.2a. It shows the non-focused/blurred and focused/sharp regions. Consider the non-focused and focused regions marked by red color ellipses. The gradient values around the red marked pixel points inside both ellipses have been plotted. In each plot, the plotted gradient values are for 101 pixels of a row for red marked pixel inside the ellipse (50 on left side and 50 on right side). Figure 4.2b and Figure 4.2c show the gradient profiles/edge width profiles
for non-focused edge pixel and focused edge pixel respectively. The edge width at each particular edge point can be considered as a function of the degree of blur in a given image region. For the blurred/non-focused edge, the edge width is larger and for the sharper/focused edge, the edge width is smaller. In the image of Figure 4.2a, the camera is focused on metallic device. It shows that focused edge (in case of the metallic device) results in a thin and sharp edge and a smaller edge width (Figure 4.2c). The non-focused edge/blurred region (in case of the person’s face) yields a thick and scattered edge and a larger edge width (Figure 4.2b). Therefore, the width of the edge can be a descriptor for the focused object.

Figure 4.2: (a) Non-focused and Focused Regions in a Scene; (b) Gradient or Edge Width Profile of Non-focused Edge Pixel; (c) Gradient or Edge Width Profile of Focused Edge Pixel.

The success of edge width information based algorithms for extracting focused regions in a video scene depends upon the accuracy of edge detection method used. The Sobel operator based edge detection technique is widely used in existing literature due to its easier implementation. Sobel operator also has advantage over simple gradient operators because
of its property to counteract noise sensitivity. It is based on computing an approximation of
the gradient of the image intensity function [88]. It uses two 3x3 spatial masks ($H_x$ and $H_y$)
which are convolved with the original image to calculate the approximations of the gradient.
The Sobel operator uses two filters.

$$H_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$  \hspace{1cm} (4.1)

$$H_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$  \hspace{1cm} (4.2)

The $x$-coordinate is defined here as increasing in the down-direction, and the $y$-
coordinate is defined as increasing in the right-direction. Therefore, masks $H_x$ and $H_y$
compute the average gradient components $G_x$ and $G_y$ across the neighboring lines (in
vertical direction) or columns (horizontal direction), respectively. These gradient
components along two directions represent the edge strength along vertical ($x$) and
horizontal ($y$) direction. The local edge strength of pixel ($x$, $y$) is defined as the combined
gradient magnitude given by

$$ES(x,y) = \sqrt{G_x^2 + G_y^2}$$  \hspace{1cm} (4.3)

_The accuracy of Sobel operator for edge detection is relatively low because it uses
two masks which detect edges in horizontal and vertical directions only. We have
addressed this problem by using Sobel compass operator, which uses a larger set of
masks with narrowly spaced orientations [89]._

The masks $H_{0}$, $H_{45}$, $H_{90}$, $H_{135}$, $H_{180}$, $H_{225}$, $H_{270}$, and $H_{315}$ are given by

$$H_{0} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$  \hspace{1cm} (4.4)

$$H_{45} = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$  \hspace{1cm} (4.5)
Each mask computes the average gradient component \((G_0, G_{45}, G_{90}, G_{135}, G_{180}, G_{225}, G_{270},\) and \(G_{315}\)) along one of the eight possible directions of the compass and each computed gradient provides edge strength along that particular direction. For this reason, the use of Sobel compass edge detector for edge detection results in better accuracy.

The local edge strength \(ES\) at position \((x, y)\) is defined as the maximum of the eight masks output \((G_0, G_{45}, G_{90}, G_{135}, G_{180}, G_{225}, G_{270},\) and \(G_{315}\)) for pixel \((x, y)\) i.e.

\[
ES(x, y) = \max(G_0, G_{45}, G_{90}, G_{135}, G_{180}, G_{225}, G_{270}, G_{315})
\]  

(4.12)

It is observed that the masks \(H_0, H_{45}, H_{90},\) and \(H_{135}\) are identical to masks \(H_{180}, H_{225}, H_{270},\) and \(H_{315},\) respectively, except for the reversed sign i.e. \(G_{180} = -G_0, G_{225} = -G_{45}, G_{270} = -G_{90},\) and \(G_{315} = -G_{135}.\)
Since \( G_{180} = -G_0 \), \( G_{225} = -G_{45} \), \( G_{270} = -G_{90} \), and \( G_{315} = -G_{135} \), therefore, above equation can be rewritten as

\[
ES(x, y) = \max(|G_0|, |G_{45}|, |G_{90}|, |G_{135}|) 
\tag{4.13}
\]

Thus, to find the edges in all eight possible directions, the four masks \((H_0, H_{45}, H_{90}, \text{and} \ H_{135})\) must actually be applied to each pixel of the input image. In order to apply these four masks on pixel \((x, y)\), the eight neighboring pixels need to be accessed as shown in Figure 4.3.

| \(x-1, y-1\) | \(x-1, y\) | \(x-1, y+1\) |
| \(x, y-1\) | \(x, y\) | \(x, y+1\) |
| \(x+1, y-1\) | \(x+1, y\) | \(x+1, y+1\) |

Figure 4.3: Pixel Neighborhood \(N(x, y)\).

The edge strengths \(G_0\), \(G_{45}\), \(G_{90}\), and \(G_{135}\), induced by \(H_0\), \(H_{45}\), \(H_{90}\), and \(H_{135}\) masks, respectively, are calculated as a weighted sum of the pixel values in neighborhood \(N(x, y)\) and the weight coefficients are given by corresponding mask values (equations (4.4), (4.5), (4.6), and (4.7)).

\[
G_0 = \{ I(x+1, y-1) + 2I(x+1, y) + I(x+1, y+1) - I(x-1, y-1) - 2I(x-1, y) - I(x-1, y+1) \} \tag{4.14}
\]

\[
G_{45} = \{ I(x+1, y) + 2I(x+1, y+1) + I(x, y+1) - I(x-1, y) - 2I(x-1, y-1) - I(x-1, y) \} \tag{4.15}
\]

\[
G_{90} = \{ I(x-1, y+1) + 2I(x, y+1) + I(x+1, y+1) - I(x-1, y-1) - 2I(x-1, y) - I(x+1, y) \} \tag{4.16}
\]

\[
G_{135} = \{ I(x-1, y) + 2I(x-1, y+1) + I(x, y+1) - I(x+1, y) - 2I(x+1, y-1) - I(x+1, y) \} \tag{4.17}
\]

Where \(I(x+1, y-1)\) indicates the intensity value of the pixel at \((x+1, y-1)\) location.

Given an optimal threshold, e.g. \(T\), the \(E(x, y)\) function classifies the pixels into two opposite classes: edge pixels versus non-edge pixels, as

\[
E(x, y) = \begin{cases} 
1, & \text{edgepixel if} \quad ES(x, y) \geq T \\
0, & \text{non-edgepixel if} \quad ES(x, y) < T 
\end{cases} \tag{4.18}
\]
This process is applied to detect the edges for three color channels separately. The edge results of the three color channels are then integrated through fusion rule, given by equation (4.19). Pixel \( (x, y) \) of color image is classified as an edge pixel if it is so classified by at least one of its three color components, in which case \( CE(x, y) \) (color edge function) is set to 1. Otherwise, it is classified as a nonedge pixel and \( CE(x, y) \) (color edge function) is set to 0.

\[
CE(x, y) = \begin{cases} 
1, & \text{edgepixel} \quad \text{if } (RE(x, y) = 1) \lor (GE(x, y) = 1) \lor (BE(x, y) = 1) \\
0, & \text{nonedgepixel} \quad \text{Otherwise}
\end{cases}
\] (4.19)

Where \( RE(x, y) \), \( GE(x, y) \), and \( BE(x, y) \) are the edge function for red, green, and blue color channels respectively, obtained by applying the above mentioned edge detection scheme for each color channel independently.

After the edge detection using Sobel compass edge detector, the edge width is computed. For each edge detected point, the portion of the edge region \( P_{\text{EDGE}} \) (i.e. the region with high gradient magnitude) with respect to a small neighborhood window of 11x11 pixels in the edge detected image is computed using edge information given by equation (4.20). The size of 11x11 pixels for the window is chosen based on experiments performed for different sizes. It is found that this neighborhood size gives good results for focused vs. non-focused pixel classification.

\[
P_{\text{EDGE}} = \text{Total Number of Edge Pixels in Neighborhood Window} \] (4.20)

A focused edge will result in small \( P_{\text{EDGE}} \), whereas a non-focused edge will yield large \( P_{\text{EDGE}} \). For a given optimal threshold, e.g., \( TF \), the \( F(x, y) \) function classifies the pixels into two different classes: focused pixels versus non-focused pixels, as

\[
F(x, y) = \begin{cases} 
1, & \text{focusedpixel} \quad \text{if } P_{\text{EDGE}} < \text{Threshold} \\
0, & \text{non-focusedpixel} \quad \text{if } P_{\text{EDGE}} \geq \text{Threshold}
\end{cases}
\] (4.21)

The final binary image is constructed by using the output of function \( F(x, y) \). This output image contains focused edge regions (represented by white color) and non-focused background regions (represented by black color).

The above mentioned edge-width based scheme for extraction of focused regions was implemented by us using C/C++ programming language. For running the code, a Dell Precision T3400 workstation (with Windows XP operating system, quad-core Intel®...
Core™2 Duo Processor with 2.93 GHz Operating Frequency, and 4GB RAM) was used. The Open Computer Vision (OpenCV) libraries were used in the code for reading video streams (either stored or coming from camera) and displaying results of focused region extraction scheme. Achieved frame rate of this software-based implementation was found to be 6 frames per second (fps) for PAL (720x576) resolution color videos. Thus a fast workstation was incapable of providing real-time performance for the above mentioned edge based focused regions extraction scheme. To achieve real-time performance, we then chose to design a dedicated, area efficient, hardware architecture for the above mentioned scheme and implemented it on FPGA development platform. Details of the same are provided in the remaining sections of this chapter.

4.4. Proposed Architecture and Implementation of Focused Region Extraction System

The conceptual block diagram of an FPGA-based system, architected and implemented by us, for extracting focused edge regions in a live video stream coming from a camera is shown in Figure 4.4. This figure also illustrates the data flow within the system. The main components of a complete FPGA-based standalone focused region extraction system are: an analog Camera, a VDEC1 Video Decoder Board for analog to digital video conversion, a custom designed Interface PCB, a Xilinx ML510 (Virtex-5 FX130T) FPGA platform for implementing real-time focused region extraction architecture, and a display device (Display Monitor).

![Figure 4.4: Architecture and Dataflow Diagram of Focused Region Extraction System Implemented in this Thesis.](image-url)
Input video, captured by a **Sony Analog Camera**, is digitized by Digilent **VDEC1 Video Decoder Board**. The digital output signals from video decoder board are transferred to FPGA platform using high speed I/O ports (**HSIO PINS**) available on Xilinx **ML510 (Virtex-5 FX130T) FPGA Board** using custom designed **Interface PCB**. The components inside dashed blue line are available on Xilinx **ML510 (Virtex-5 FX130T) FPGA Board**. These include **FPGA Device** (shown by dashed red line), High Speed Input Output Interface (**HSIO PINS**), and **DVI Interface**. The **Camera Interface** module uses digital video signals available at the FPGA interface and extracts **RGB data** and generates **Video Timing Signals**. **DVI Display Controller** is used to display the processed data on **Display Monitor**. The design and development of **Camera Interface** module and **DVI Display Controller** have been presented earlier in chapter 2 of this thesis.

**RGB data** is the input for **Focused Region Extraction** architecture (shown by green dashed line) having three main modules: **Edge Detection**, **Edge Width Computation**, and **Focused Pixel Classification**. The edge is detected for incoming **RGB data** by **Edge Detection** module, and, based on the edge detection output the edge width is computed by **Edge Width Computation** module for each edge pixel. By comparing the computed edge width with a threshold value, the incoming pixel is classified by **Focused Pixel Classification** module as focused or non-focused pixel. **DVI Controller** module uses output of **Focused Pixel Classification** module and video timing signals (from **Camera Interface Module**) to display the processed pixels as output on the **Display Monitor**. The overall goal of the system is to extract focused edge regions in real-time in an incoming live video stream using edge width information.

**4.4.1. Proposed Area Efficient Edge Computation Architecture**

The goal of **Edge Detection** module is to perform edge detection on incoming color image data. This is achieved by performing edge detection three times, once each for red channel, green channel, and blue channel. For this, the incoming 24-bit RGB pixel data is separated into three color channel data R, G, and B (each of 8 bits), and then edge detection process is applied for three color components concurrently (in parallel). Edge results of the three color channels are fused to form final edge map of the color image. The final result is a 1-bit output – ‘1’ for edge pixel and ‘0’ for non-edge pixel. The dataflow diagram of this process is shown in Figure 4.5.
For edge detection on each color channel, we have used Sobel compass operator. The use of Sobel compass edge detector increases the accuracy of edge detection algorithm over Sobel edge detector by computing gradients in all eight compass directions. However, it also doubles the computational complexity as compared to Sobel edge detector, thereby almost doubling the requirement of FPGA resources. For this reason, we have proposed and implemented an area efficient hardware architecture for Sobel compass edge detector, which utilizes FPGA resources comparable to those as utilized by Sobel edge detector and gives the enhanced accuracy of Sobel compass edge detector.

The basic block level dataflow diagram for computation of Sobel compass edge detector is shown in Figure 4.6. It consists of four stages. In the first stage, the incoming pixel data from Camera Interface Module is stored in Buffer Memory. Four gradients along different directions (Gradient₀, Gradient₄₅, Gradient₉₀, and Gradient₁₃₅) are computed in the second stage. The Maximum Gradient is selected among these four and the final Edge Map is computed by comparing the maximum gradient value with a threshold in the third and fourth stages, respectively.
Sobel compass edge detector is a window based operator as 3x3 masks are used to compute gradient values along different directions over an input image requiring pixel neighborhood information for computing the edge map of a particular pixel. Therefore, input pixel data coming from the camera interface module cannot be processed directly. It is necessary to store at-least two rows of input image data in FPGA on-chip memory before the processing begins. To achieve this, we have used a smart buffer based memory architecture [90], [91] which utilizes two FIFOs and a set of nine registers in order to shift the image data into the computing window (Figure 4.7). The data is coming from the camera row by row and one pixel data in one clock cycle. When the buffers are filled, this architecture provides access to the entire pixel neighborhood every clock cycle. Length of the shift register depends upon the input image width. In this case, for PAL (720x576) size video, it is 717 (=720-3).

![Figure 4.7: Input Buffer Memory Architecture.](image)

The standard implementations of Sobel compass edge detector available in the existing literature [92], [93] use four processing elements in parallel for computing gradients along different directions as shown in Figure 4.8. For accurate comparisons with our results, we have coded the standard architecture (Figure 4.8) in VHDL, synthesized it using Xilinx ISE (Version 12.1) tool chain, and implemented it on Xilinx ML510 (Virtex-5 FX130T) FPGA board. The resulting maximum clock frequency for this architecture was 118.5 MHz.

The processing element architecture (PE) for gradient computation in four different directions (i.e. PE₀, PE₄5, PE₉₀, and PE₁₃₅) is shown in Figure 4.9. It performs addition, subtraction, and multiplication by 2 using shift operation.
In standard implementation, it is observed that all the four processing elements perform identical set of operations (addition, subtraction, and multiplication by 2) on inputs applied to them. The only difference among the four gradient computation units ($PE_0$, $PE_{45}$, $PE_{90}$, and $PE_{135}$) is in terms of the inputs applied to them at a particular time. Therefore, by switching the inputs applied to any one of the processing elements in an appropriate manner, that single processing element can be used to compute all the four gradients along different directions. This forms the basis of our proposal for area efficient architecture.

The effectiveness of this input switching concept for area reduction was demonstrated and published by us initially for Sobel operator based edge detector [94], [95], in which the proposed area efficient architecture used single processing element for computing both horizontal and vertical gradients and a reduction of approximately 45% was achieved in FPGA Slice utilization over existing implementations. The concept was further extended for Sobel compass edge detector, and a new resource efficient architecture was proposed for Sobel compass edge detector which utilized approximately 40% less FPGA Slices as compared to existing standard implementations. The details of proposed area efficient architecture for Sobel compass edge detector are explained in the following text in this section.
For real-time video surveillance applications, the required frame rate is 25 fps (frames per second) for PAL size color video. For safe side, we have considered a video frame rate that is twice of that required for video surveillance application i.e. a frame rate of 50 fps for PAL size color video. For the frame rate of 50 fps for PAL size color video, clock frequency at which image pixel data (in RGB) is available is 27 MHz. If the processing element is operated at 108 MHz (4 x 27 MHz), then, all the four gradients can be computed for available current image pixel data before the arrival of the next image pixel data using a single processing element. The operating frequency of 108 MHz is well within the limits of maximum operating frequency of the processing element architecture (118.5 MHz) obtained from synthesis results. The digital clock managers available on the FPGA are used to generate the clock frequency of 108 MHz from the 27 MHz pixel data clock.

Implementation of the proposed area efficient architecture for real-time computation of edges in an image using Sobel compass edge detector is shown in Figure 4.10. Gradient computations for all the directions are realized through a single processing element operating at 108 MHz. The processing element (PE) is used in an appropriate sequential order in different time slots for computing gradients along all the directions. This architecture greatly economizes on the FPGA resource usages (area) but needs internal storage elements to store results for future use and a set of multiplexers and de-multiplexers for switching of inputs and outputs in different time slots. This architecture also requires a controller which ensures proper functioning of the design. Control signals for Input selection multiplexer and output selection de-multiplexer are generated at 108 MHz so that inputs to processing element can be switched and output can be stored properly. Image pixel data moves through the computing window at 27 MHz. Gradient values for directions along 0, 45, 90, and 135 degrees are computed by the processing element in 4 consecutive clock cycles of 108 MHz clock frequency. The computed gradients for directions along 0, 45, and 90 degrees are stored in registers $R_1$, $R_2$, and $R_3$ respectively which work at 108 MHz frequency. Computed gradient for the direction along 135 degrees is directly used by the Maximum Gradient computation module for computing the maximum gradient value ($MGR$). The maximum gradient is compared with the user threshold and edge is computed by Edge Map computation module. The computed edge for current pixel is stored in output register $E$ which works at the clock rate of 27 MHz. Therefore, before the arrival of next pixel data (at 27 MHz), the final edge map of current pixel data is available.
Figure 4.11 illustrates the timing used for the operation of the area efficient Sobel compass edge detector architecture we have implemented. Assume that the input *Pixel Data* is arriving from the camera interface module (output of sliding window memory buffer) at *PCLK* clock frequency. The implemented architecture uses a single processing element which works at 4x*PCLK* clock frequency (*4PCLK*). *Input to PE* refers to the inputs to the processing element during that particular clock cycle. Input selection is done by using a multiplexer and control signals. *GR* represents the gradient computed by the processing element in a particular clock cycle of *4PCLK* clock (it is any of the four gradients along 0 degrees direction (*H₀*), 45 degrees direction (*H₄₅*), 90 degrees direction (*H₉₀*), and 135 degrees direction (*H₁₃₅*)). *R₁out*, *R₂out*, and *R₃out* are outputs of the registers used for intermediate storage of the gradients. These three registers operate at *4PCLK* clock frequency and are reset at the arrival of every new pixel data. *MGR* is the output of maximum gradient computation module which contains the maximum gradient value computed every fourth cycle of the *4PCLK* clock by using all the four gradients *H₀*, *H₄₅*, *H₉₀*, and *H₁₃₅*. *PAEout* is the final result of edge detection i.e. output of *E* register. The
proposed and implemented area efficient Sobel compass edge detector architecture computes the edge map for each incoming pixel at $PClk$ clock rate while utilizing much less FPGA resources.

The performance of the proposed and implemented area-efficient Sobel compass edge detector architecture is further improved by using pipelined processing element (Figure 4.12) at the cost of area occupied by pipelined registers. Resulted maximum clock frequency is 405.597 MHz. Improvement in clock frequency in pipelined architecture is due to the

![Timing Diagram for Computing Edge using Area Efficient Sobel Compass Edge Detector Architecture.](image1)

![Pipelined Processing Element Architecture.](image2)
segmentation of maximum combinational logic delay in the non-pipelined processing element.

4.4.2. Proposed Area Efficient Edge Width Computation Architecture

For computing the edge width, a window of 11x11 pixels has been considered. The 11x11 pixels size window is moved over the entire edge detected image for each edge detected pixel. In order to compute edge width for a pixel at \((x, y)\), the edge information of 11x11 window (neighborhood shown in Figure 4.13) must be available. Therefore, it is necessary to store the edge information of at-least 10 rows before the edge width computation processing can begin. To achieve this, we have used a smart buffer based memory architecture which utilizes 10 FIFOs and a set of registers (121 registers) in order to shift the edge detected image data into the computing window (Figure 4.14). The length of the FIFOs depends upon the width of the input image. In this case for PAL size video it is 720-11=709. The width of FIFOs and registers is 1-bit. This architecture is most suitable for online computation and places highest demand on the internal memory. Because modern FPGA devices contain large amounts of on-chip memory (Block RAMs), this architecture does not cause any problems.

![Figure 4.13: Pixel Neighborhood Required for Computing Edge Width.](image)
The 121 1-bit edge data are summed and the sum value indicates the edge width information for that particular pixel. The most obvious implementation for adding 121 1-bit values is shown in Figure 4.15. It requires seven stages of adders. The first stage requires sixty 2-bit adders, second stage requires thirty 3-bit adders, third stage requires fifteen 4-bit adders, fourth stage requires eight 5-bit adders, fifth stage requires four 6-bit adders, sixth stage requires two 7-bit adders, and final stage uses single 7-bit adder and produces a 7-bit number which corresponds to the edge-width of the current pixel (in centre of current processing window). The main disadvantage of this approach is that a large area is occupied by a large number of different adder units.
To overcome the problem of larger area associated with the above architecture, an alternate architecture has been proposed and designed by us for edge width computation as shown in Figure 4.16. This architecture performs fast computation with a lesser number of adder units and hence is more area efficient.

![Figure 4.16: Area Optimized Architecture for Edge Width Computation.](image)

In this proposed architecture, column-wise addition is performed in succession. The values $SC_1$ to $SC_{10}$ denote the sum of the edge outputs in the Column 1 to Column 10 of the 11x11 size window. When the edge detected outputs of the first column arrive they are added and stored in a 5-bit register as $SC_1$, and in the next clock cycle when the edge detected outputs of the second column arrive, the same operation is performed and the result is stored in another 5-bit register as $SC_2$, and so on till 10$^{th}$ column. Finally, when the 11$^{th}$ column pixels arrive they are added together and their sum is added with the resultant sum values $SC_1$ to $SC_{10}$ to get the final edge width of the 11x11 size window. This architecture is based on the fact that the sliding window moves by one pixel in horizontal direction, therefore, the sum values of previous ten columns ($SC_1$ to $SC_{10}$) can be reused to compute edge width of current pixel. This reusing of previously computed column sum values reduces the computational complexity and makes the architecture much more efficient in terms of FPGA resources and performance as compared to the architecture of standard
implementation. This architecture uses five 2-bit adders, three 3-bit adders, one 4-bit adder, six 5-bit adders, two 6-bit adders, and three 7-bit adders.

**4.4.3. Classification and Post Processing Architecture**

The classification and dilation unit (Figure 4.17) first classifies a pixel as focused pixel or non-focused pixel and then performs the dilation operation as post processing step to provide better connectivity between extracted regions. The classification is done by simply comparing the edge width value with a threshold. If the edge width value is less than a particular threshold value, the pixel is labeled as focused pixel, otherwise, it is labeled as non-focused pixel. This whole task is achieved by a comparator unit. The output is ‘1’ if the pixel is focused otherwise it is ‘0’. Finally, a post processing operation called dilation is performed on the output of the classification step. Dilation is performed by using a 5x5 structuring element. The dilation process is used to provide better connectivity between the focused pixels as it has the ability of filling in small holes and connecting disjoint objects. The dilation operation is performed by laying the structuring element over the focused pixels labeled image and sliding it across the image. The operation performed in the case of dilation is the logical OR operation on the data values lying under the structuring element.

![Figure 4.17: Architecture for Classification and Dilation.](image)

**4.5. Synthesis Results**

All the design modules of the proposed system for focused edge region extraction were coded in VHDL and simulated using ModelSim. A top level design module was created
which invoked the proposed and implemented pipelined area efficient edge detection architecture, area efficient edge width computation architecture, focused pixel classification and dilation architecture, camera interface module, and display controller module. A User Constraint File (UCF) was created to map the input/output ports of the design on actual pins of the FPGA. This top level design was synthesized using Xilinx ISE (Version 12.1) tool chain. The resulting configuration (.bit) file was stored in the Flash Memory to enable automatic configuration of the FPGA at power-on. A complete standalone prototype system working in real-time for automatic extraction of focused edge regions in a live video stream was developed and is shown in Figure 4.18. The components of the system are Xilinx ML510 (Virtex-5 FX130T) FPGA platform, Sony EVI D-70P camera, and display monitor. The implemented focused region extraction system uses only two interfaces (Camera Interface and DVI Controller) out of the four interfaces designed and presented earlier in chapter 2 of this thesis.

![Figure 4.18: Complete Focused Region Extraction System Setup.](image)

Table 4.1 shows the comparison of FPGA resources (post-place and route results) utilized by our proposed and implemented area efficient hardware architecture of Sobel compass edge detector and the standard architecture of Sobel compass edge detector. The post place-and-route results (Table 4.1) reveal that the FPGA resources utilized by area efficient architecture of Sobel compass edge detector proposed and designed by us in this
thesis are more than 40% less as compared to the standard architecture of Sobel compass edge detector existing in the literature [92], [93]. The reduced resources do not amount to one-fourth of the resources utilized by the standard architecture because of the resources utilized by the required additional input and output switching logic (multiplexers and de-multiplexers), registers, and controller for ensuring proper functioning of the area efficient architecture. The maximum operating frequency for the area efficient architecture is 118.5 MHz. Table 4.2 shows the comparison of FPGA resources (post-place and route results) utilized by pipelined area efficient hardware architecture of Sobel compass edge detector designed by us and the standard architecture of Sobel compass edge detector. Post place-and-route results (Table 4.2) reveal that FPGA resources utilized by the pipelined area efficient architecture of Sobel compass edge detector are more than 37% less as compared to the standard architecture of Sobel compass edge detector existing in the literature [92], [93]. Maximum operating frequency for the pipelined area efficient architecture is 405.597 MHz.

Table 4.1: Comparison of FPGA Resource Utilization for Sobel Compass Edge Detector: Standard Architecture versus Area Efficient Architecture Designed by us.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Standard Architecture [92], [93]</th>
<th>Implemented area Efficient Architecture</th>
<th>Percentage of Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Occupied Slices</td>
<td>67</td>
<td>40</td>
<td>40.3%</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>222</td>
<td>109</td>
<td>50.9%</td>
</tr>
<tr>
<td>Slice Registers</td>
<td>2</td>
<td>40</td>
<td>---</td>
</tr>
<tr>
<td>Route-thrus</td>
<td>16</td>
<td>2</td>
<td>87.5%</td>
</tr>
</tbody>
</table>

Table 4.2: Comparison of FPGA Resource Utilization for Sobel Compass Edge Detector: Standard Architecture versus Pipelined Area Efficient Architecture Designed by us.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Standard Architecture [92], [93]</th>
<th>Implemented Pipelined area Efficient Architecture</th>
<th>Percentage of Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Occupied Slices</td>
<td>67</td>
<td>42</td>
<td>37.3%</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>222</td>
<td>117</td>
<td>47.3%</td>
</tr>
<tr>
<td>Slice Registers</td>
<td>2</td>
<td>95</td>
<td>---</td>
</tr>
<tr>
<td>Route-thrus</td>
<td>16</td>
<td>2</td>
<td>87.5%</td>
</tr>
</tbody>
</table>
For an accurate comparison of results of standard architecture (Figure 4.19b) and the pipelined area efficient architecture (Figure 4.19c), we did the following experiment with online camera data. We used both the standard architecture and the pipelined area efficient architecture in parallel and applied same camera inputs (Figure 4.19a) to both the architectures simultaneously and stored the results (images) from both. A pixel by pixel subtraction operation was performed on the two stored images i.e. Edge_Standard (Figure 4.19b) and Edge_Pipelined-Area-Efficient (Figure 4.19c). For non-zero subtraction result (if two values are different), white color was displayed (R = 255, G = 255, B = 255) and for zero result (if two values are identical), black color was displayed (R = 0, G = 0, B = 0) on the monitor screen. The resulting output image/matrix after subtraction of Figure 4.19b and Figure 4.19c was black/zero. The output is shown in Figure 4.19d. The logic is given below:

If \((\text{Edge} \_\text{Standard} - \text{Edge} \_\text{Pipelined-Area-Efficient}) = 0)\)

then \{ (R = 0, G = 0, B = 0); \}

Else \{ (R = 255, G = 255, B = 255); \}

The results of this experiment establish that edge detection results of standard architecture are exactly matching with the edge detection results of pipelined area efficient architecture designed by us.
The Mean Square Error (MSE) was also computed for the test images and videos. The computed MSE for every frame of all the test bench videos was zero and this has confirmed that the implemented pipelined area efficient architecture for edge detection produces same edge detection results as the standard architecture (with no impact on the quality of the processed videos) but with 37% reduced FPGA slice requirements.

Table 4.3 shows the comparison of FPGA resources (post-place and route results) utilized by the proposed and implemented area efficient hardware architecture of edge width computation (Figure 4.16) and the standard architecture of edge width computation (Figure 4.15). The results (Table 4.3) reveal that FPGA resources utilized by the proposed and implemented area efficient architecture designed by us for edge width computation are approximately 70% (average of all four parameters) less as compared to the standard architecture. Table 4.4 presents the FPGA resources utilized (post place-and-route results) by each module of the area efficient architecture and their integration i.e. pipelined area efficient Sobel Compass Edge Detector architecture for color videos (SCED), area efficient Edge Width Computation architecture with buffer memory (EWC), Classification and Dilation (CD) architecture, and proposed and implemented complete architecture (SCED, EWC, CD). Table 4.5 presents the FPGA resources utilized (post place-and-route results) by
the complete implemented system for focused edge region extraction including Camera Interface and DVI display Controller. Maximum operating frequency for the complete system is 170.942 MHz, and, maximum operating frequency for the pipelined processing element is 405.597 MHz. The maximum possible frame rate for PAL (720x576) size color video is 244 frames per second.

Table 4.3: Comparison of FPGA Resource Utilization for Edge Width Computation Architectures: Standard Architecture versus Area Efficient Architecture Designed by us.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Standard Architecture</th>
<th>Implemented area Efficient Architecture</th>
<th>Percentage of Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Occupied Slices</td>
<td>73</td>
<td>17</td>
<td>82.2%</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>201</td>
<td>56</td>
<td>72.2%</td>
</tr>
<tr>
<td>Slice Registers</td>
<td>121</td>
<td>22</td>
<td>81.8%</td>
</tr>
<tr>
<td>Route-thrus</td>
<td>2</td>
<td>1</td>
<td>50.0%</td>
</tr>
</tbody>
</table>

Table 4.4: FPGA Resource Utilization by Individual Modules of Focused Edge Region Extraction System and their Integration.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Implemented Pipelined Sobel Compass Edge Detector Architecture for Color Videos (SCED)</th>
<th>Edge Width Computation with Buffer Memory (EWC)</th>
<th>Classification and Dilation (CD)</th>
<th>Proposed Complete Architecture (SCED+EWC+CD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Registers</td>
<td>429</td>
<td>72</td>
<td>29</td>
<td>531</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>520</td>
<td>97</td>
<td>35</td>
<td>656</td>
</tr>
<tr>
<td>Route-thrus</td>
<td>41</td>
<td>05</td>
<td>10</td>
<td>37</td>
</tr>
<tr>
<td>Occupied Slices</td>
<td>206</td>
<td>62</td>
<td>15</td>
<td>318</td>
</tr>
<tr>
<td>BRAMs 36K</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Memory (Kb)</td>
<td>108</td>
<td>180</td>
<td>72</td>
<td>360</td>
</tr>
<tr>
<td>DSP Slices</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DCMs</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 4.5: FPGA Resource Utilization by Complete Implemented System for Extraction of Focused Edge Regions.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Camera Interface (CI)</th>
<th>Display (DVI)</th>
<th>Complete Proposed Architecture (CPA)</th>
<th>Complete System (CPA+CI +DVI)</th>
<th>Total Available Resources</th>
<th>Percentage of Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Registers</td>
<td>391</td>
<td>27</td>
<td>531</td>
<td>1021</td>
<td>81920</td>
<td>1.25%</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>434</td>
<td>101</td>
<td>656</td>
<td>1216</td>
<td>81920</td>
<td>1.48%</td>
</tr>
<tr>
<td>Route-thrus</td>
<td>42</td>
<td>39</td>
<td>37</td>
<td>129</td>
<td>163840</td>
<td>0.08%</td>
</tr>
<tr>
<td>Occupied Slices</td>
<td>199</td>
<td>33</td>
<td>318</td>
<td>562</td>
<td>20840</td>
<td>2.70%</td>
</tr>
<tr>
<td>BRAMs 36K</td>
<td>3</td>
<td>0</td>
<td>10</td>
<td>13</td>
<td>298</td>
<td>4.36%</td>
</tr>
<tr>
<td>Memory (Kb)</td>
<td>108</td>
<td>0</td>
<td>360</td>
<td>468</td>
<td>10728</td>
<td>4.36%</td>
</tr>
<tr>
<td>DSP Slices</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>320</td>
<td>0.94%</td>
</tr>
<tr>
<td>DCMs</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>12</td>
<td>8.33%</td>
</tr>
<tr>
<td>IOs</td>
<td>16</td>
<td>22</td>
<td>36</td>
<td>36</td>
<td>840</td>
<td>4.28%</td>
</tr>
</tbody>
</table>

The complete FPGA-based implementation of the system working at 170.942 MHz clock frequency, with the processing element working at 405.597 MHz clock frequency processes the PAL (720x576) resolution color video 40 times faster than its C/C++ based software implementation on a Dell workstation with 2.93 GHz clock frequency. Synthesis results reveal that enough FPGA resources are left for implementing other complex computer vision algorithms required by a complete automated video surveillance system on the same FPGA platform, as the area efficient focused edge region extraction system proposed, designed, and implemented by us utilizes less than 5% Block RAMs and less than 2% FPGA slices available on the FPGA development platform. The implemented system yields good results and works in real-time.
The system architecture for extraction of focused edge regions, proposed, designed, and implemented by us is adaptable and scalable for different video sizes. The performance evaluation results for different video resolutions are given in Table 4.6. It is capable of processing high definition (HD) videos (1920x1080 resolution color videos) in real-time at the frame rate of 48 frames per second.

Table 4.6: Frame Rates of Proposed Pipelined Area Efficient Focused Region Extraction Architecture for Different Video Resolutions.

<table>
<thead>
<tr>
<th>Video Resolution</th>
<th>Frame Rate of Proposed Pipelined area Efficient Architecture (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD (1920 x 1080)</td>
<td>48</td>
</tr>
<tr>
<td>PAL (720 x 576)</td>
<td>244</td>
</tr>
<tr>
<td>NTSC (720 x 480)</td>
<td>293</td>
</tr>
<tr>
<td>VGA (640x480)</td>
<td>330</td>
</tr>
<tr>
<td>CIF (352 x 288)</td>
<td>1000</td>
</tr>
</tbody>
</table>

There exists no hardware implementation for extraction of focused regions in the existing literature and all the published algorithms in literature have been implemented in software using Personal Computers or Workstations and highest reported frame rate is 1 fps for QCIF size videos on Intel Pentium IV 2.78 GHz PC [79]. Thus it can be seen that the VLSI/hardware based real-time system for extraction of focused edge regions in a live video stream, designed and implemented by us in the thesis is far faster than the existing PC based systems.

4.6. Visual Results

The implementation has been tested for focused edge region extraction in live color video streams directly coming from the camera. The color video resolution is of standard PAL (720x576) size. Three different possible situations are considered for testing (Figure 4.20 to Figure 4.22). For all the three situations the top row shows the original frames taken/extracted from the live video streams. The output images produced by the proposed, designed, and implemented area efficient focused region extraction system for different
frames are shown just below the respective frames. Figure 4.20 shows the situation where complete scenes are in focus (all objects present in scene are focused). The output images show the extracted focused edge regions. The situations of completely out of focus scenes are shown in Figure 4.21. In this case the resultant images are blank/black as no focused edge regions exist in these frames. The examples of focused (metallic device) and non-focused (face) objects in the same scene have been shown in Figure 4.22. In this case, in the output images only focused edge regions (metallic device) extracted by the implemented system are shown. The implementation yields good results and works in real-time (at frame rate of 244 fps for PAL size color video). PAL resolution is projected to be the most commonly used video resolution for future video surveillance cameras. Therefore, the implemented system can easily and effectively detect focused regions in real-time for surveillance applications.

Figure 4.20: Focused Objects Present in the Scene and Extracted Focused Edge Regions.
Figure 4.21: Non-focused Objects Present in the Scene and Extracted Focused Pixels are Nil.

Figure 4.22: Non-focused Object (Face) and Focused Object (Metallic Device) Present in the Same Scene and Only Focused Edge Regions (Metallic Device) are Extracted.

4.7. Summary

In this chapter, we have presented the edge width information based scheme for extracting focused edge regions in a live video stream coming from the camera and have
proposed and designed a novel area efficient VLSI architecture to meet the real-time constraints of the video surveillance system. Sobel compass operator has been used for edge detection which provides better accuracy for edge detection as compared to Sobel operator. The proposed and implemented area efficient architecture of Sobel compass edge detector uses approximately 37% less FPGA resources as compared to Standard implementations of Sobel compass operator available in existing literature. The complete system including camera interface, area efficient focused region extraction architecture, and display interface has been implemented on Xilinx ML510 (Virtex-5 FX130T) FPGA Board. As demonstrated in video sequences, the proposed and implemented system robustly and automatically extracts the focused edge regions present in the scene in real-time (at 244 frames per second) for standard PAL (720x576) size color video. The implemented system can be used as standalone system for extracting focused edge regions in a live video stream in real-time. In chapter 6, this work will be integrated with motion detection architecture for designing a FPGA-based video surveillance system for selecting frames of interest based on motion in focused regions only.