EFFICIENT FPGA IMPLEMENTATION OF IMAGE ENHANCEMENT USING VIDEO STREAMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

HAZAN GÜNAY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JANUARY 2010

Approval of the thesis

EFFICIENT FPGA IMPLEMENTATION OF IMAGE ENHANCEMENT USING VIDEO STREAMS

submitted by **Hazan Günay** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İsmet Erkmen
Head of Department, **Electrical and Electronics Engineering Dept., METU** _____

Prof. Dr. Murat AŞKAR
Supervisor, **Electrical and Electronics Engineering Dept., METU** _____

Examining Committee Members:

Prof. Dr. Hasan GÜRAN _____
Electrical and Electronics Engineering Dept., METU

Prof. Dr. Murat AŞKAR _____
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Aydın ALATAN _____
Electrical and Electronics Engineering Dept., METU

Dr. Selim EMİNOĞLU _____
Natural and Applied Sciences, METU

Serkan DİNMEZ, MSc. _____
MGEO, ASELSAN

Date: 13.01.2010

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Hazan Günay

Signature :

ABSTRACT

EFFICIENT FPGA IMPLEMENTATION OF IMAGE ENHANCEMENT USING VIDEO STREAMS

Günay, Hazan
M.Sc., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Murat Aşkar

January 2010, 73 pages

This thesis is composed of three main parts; displaying an analog composite video input by via converting to digital VGA format, license plate localization on a video image and image enhancement on FPGA.

Analog composite video input, either PAL or NTSC is decoded on a video decoder board; then on FPGA, video data is converted from 4:2:2 YCbCr format to RGB. To display RGB data on the screen, line doubling de-interlacing algorithm is used since it is efficient considering computational complexity and timing.

When taking timing efficiency into account, image enhancement is applied only to beneficial part of the image. In this thesis work, beneficial part of the image is considered as numbered plates. Before image enhancement process, the location of the plate on the image must be found.

In order to find the location of plate, a successful method, edge finding is used. It is based on the idea that the plate is found on the rows, where the brightness variation is largest. Because of its fast execution, band-pass filtering with finite response (FIR) is used for highlighting the high contrast areas.

Image enhancement with rank order filter method is chosen to remove the noise on the image. Median filter, a rank order filter, is designed and simulated. To improve image quality while reducing the process time, the filter is applied only to the part of the image where the plate is.

Design and simulation is done using hardware design language VHDL. Implementations of the chosen approaches are done on MATLAB and Xilinx Virtex-2 Pro FPGA. Improvement of the implementation considering speed and area is evaluated.

Keywords: Image Enhancement on FPGA, Video Stream, Plate Localization, Edge Finding by VHDL, Median Filter by VHDL.

ÖZ

VİDEO AKIŞINI KULLANARAK FPGA ÜZERİNDE VERİMLİ GÖRÜNTÜ İYİLEŞTİRME UYGULAMASI

Günay, Hazan
Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Murat Aşkar

Ocak 2010, 73 sayfa

Bu tez üç ana parçadan oluşmaktadır; FPGA üzerinde, analog kompozit video görüntüsünün sayısal VGA formatına dönüştürülerek ekrana basılması, görüntü üzerinde plaka yeri saptama ve görüntü iyileştirme.

Analog kompozit video girişi, PAL ya da NTSC formatı, video çözücü kart üzerinden FPGA donanımına aktarılarak 4:2:2 YCbCr formatından RGB formatına dönüştürülür. RGB sayısal video datasını ekranda göstermek için satır tekrarı yoluyla binişimsizleştirme işlemi uygulanır. Satır tekrarı ile binişimsizleştirme hesap karmaşıklığı ve zaman göz önüne alındığında verimli bir uygulama olduğu için seçilmiştir.

Zaman verimliliği göz önüne alındığında, görüntü iyileştirme sadece görüntünün gerekli (kullanılacak) kısmına yapılır. Bu tezde, görüntü üzerinde kullanılacak kısım

araç plakasıdır. Görüntü iyileştirme işleminden önce plakanın yerinin tespit edilmesi gereklidir.

Plaka yer tespiti için başarılı bir metod olan kenar bulma yöntemi kullanılmışır. Bu yöntem, plakanın görüntü üzerinde en büyük parlaklık değişiminin olduğu satırlarda olması esasına dayanır. Sonlu yanıtlı bant geçiren süzgeç yüksek kontrast seviyeli alanları vurgulamak için hızlı uygulamasından dolayı kullanılır.

Sıralama düzenleyici süzgeç ile görüntü iyileştirme yöntemi, görüntü üzerindeki gürültüyü kaldırmak için kullanılmıştır. Sıralama düzenleyici bir süzgeç olan ortanca süzgeç tasarlanıp, simüle edilmiştir. Süzgeç, görüntü kalitesini artırırken, işle zamanını azaltmak için sadece plaka olan bölgeye uygulanmıştır.

Tasarım ve simülasyon donanım tasarım dili VHDL ile yapılmışltır. Seçilen yaklaşımların gerçeklenmesi MATLAB ve Xilinx Virtex-2 Pro FPGA donanımı üzerinde yapılmıştır. Uygulamanın hız ve alan açısından gelişimi değerlendirilmiştir.

Anahtar Kelimeler: FPGA Donanımı Üzerinde Görüntü İyileştirme, Video Akışı, Plaka Yer Tespiti, Kenar Bulma VHDL Uygulaması, Bant Geçiren Süzgeç VHDL Uygulaması, Ortanca Süzgeç VHDL Uygulaması.

ACKNOWLEDGMENTS

I wish to express my deepest thanksgiving to my supervisor, Prof. Dr. Murat AŞKAR for his boundless help, excellent supervision and leading guidance from beginning to end of thesis work.

I also express my sincere gratitude to my colleagues from ASELSAN A.Ş and ASELSAN A.Ş itself, for their initiative ideas and guidance that helped to construct this work.

I would also like to thank TUBITAK for their support during my MSc. study.

None of this would have been possible without the constant support and endless patience of my family. Their insight throughout my career has been invaluable.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

LIST OF TABLES

TABLES

LIST OF ABBREVIATONS

EAV : End of Active Video

FPGA : Field Programmable Gate Array

NTSC : National Television Systems Committee

PAL : Phase Alternating Line

RAM : Read Access Memory

RGB : Red Green Blue

SAV : Start of Active Video

VHDL : VHSIC Hardware Design Language

VHSIC : Very High Speed Integrated Circuits

YCbCr : Luma, blue-difference and red-difference chroma components

CHAPTER 1

INTRODUCTION

Recently, Field Programmable Gate Array (FPGA) technology has become a viable target for the implementation of algorithms of video image processing applications. The unique architecture of the FPGA has allowed the technology to be used in many applications enclosing all aspects of video image processing [8].

Field Programmable Gate Arrays (FPGAs) represent reconfigurable computing technology which is in some ways ideally suited for video processing. Reconfigurable computers are processors which can be programmed with a design, and then reprogrammed (or reconfigured) with virtually limitless designs as the designer's needs change. FPGAs generally consist of a system of logic blocks (usually lookup tables and flip-flops) and some amount of Random Access Memory (RAM), all wired together using an array of interconnects. All of the logic in an FPGA can be 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.

Today, FPGAs can be developed to implement parallel design methodology, which is not possible in dedicated DSP designs. ASIC design methods can be used for FPGA design, allowing the designer to implement designs at gate level. However, usually engineers use a hardware language, which allows for a design methodology similar to software design. This software view of hardware design allows for a lower overall support cost and design abstraction.

There is a need for intelligent traffic management systems in order to cope with the constantly increasing traffic on today's roads. Video based traffic surveillance is an important part of such systems [5]. Information about current situations can be

automatically extracted by image processing algorithms. Beside vehicle detection and tracking, identification via license plate recognition is important for a variety of applications. These include, e.g. automatic congestion charge systems, access control, tracing of stolen cars, or identification of dangerous drivers.

Automatic License Plate Recognition systems are very popular and studied all over the world. Two main parts of these systems are;

- Finding license plates in images (Plate Localization)
- Reading text from license plates.

The problems about the images with license plates are;

- Poor image resolution (the plate is too far away – low quality camera)
- Motion Blur
- Poor lighting and low contrast due to overexposure, reflection or shadows
- Dirt on the plate.

In order to obtain clear and readable images, image enhancement techniques, most of them are based on filters to remove noise and unwanted effects of the light, are used. To improve image quality with an efficient way, realization on FPGA is a good choice.

1.1. Scope of Thesis

In this thesis, image processing algorithms are implemented on FPGA hardware. The aim is enhancing the plate area on the image considering speed and area on FPGA. Without giving much effort on enhancing the whole image, only the plate area of the image is enhanced.

The focus of this thesis lies on the first part of the problem, finding license plates in video streams. To find the location of the plate, the observation that number plates usually appear as high contrast areas in the image (black-and-white or black-and-yellow), is used. Also, another assumption is that the letters and numbers are placed

in the same row (i.e. at identical vertical levels), resulting in frequent changes in the horizontal intensity.

This provides the reason for detecting the horizontal changes of the intensity, since the rows that contain the number plate are expected to exhibit many sharp variations. Hence, the edge finding method is exploited to find the location of the plate.

In order to improve time and area efficiency, instead of reading the whole image or reading the plate characters from the image, plate area on the image is enhanced and made more readable and clearer.

To enhance a part of the image, noise removal and smoothing median filter is used due to easy development.

As a general looking, this thesis is composed of (1) decoding analog video data to digital, (2) finding location of the plate on the video, (3) enhancement of the plate part of the video, and (4) as a final step, displaying the processed image on the monitor.

Design and simulation is achieved by VHDL using the Xilinx ISE development environment [17]. For FPGA implementation, Xilinx Virtex- 2 XUPV2 Pro [18] development board is used. To get the analog video, VDEC1 video decoder board [23] is used.

1.2. Thesis Outline

In Chapter 2, a brief history of the used processes is given. Main parts of the system and the methods used in the literature are discussed. De-interlacing algorithms, from primitive to complex methods, are given. Existing color space conversion equations and constants are described followed by automatic number plate localization methods and their efficiencies. Finally, existing image enhancement approaches are outlined. The studies which use different techniques are cited.

In Chapter 3, general description of the hardware of the system, FPGA development boards and video decoder board, are described. Used hardware design languages,

namely Verilog and VHDL, are discussed. Finally, used analog video interfaces (i.e. PAL and NTSC) are described.

In Chapter 4, design of the whole system is described from the video input to output. Design is considered as a 3 part process; video receiver, number plate localization and image enhancement. For the video receiver part, configuration of the decoder, video conversions, de-interlacing, buffering and video synchronization timing are explained. Then, used plate localization method and its implementation are described. Finally, image enhancement filter design is outlined.

In Chapter 5, implemented algorithms and results by using video streams obtained from different sources are given. Results obtained from these algorithms are discussed and evaluated.

In Chapter 6, a conclusion is drawn and possible future studies are discussed.

CHAPTER 2

STATE OF THE ART

This thesis is composed of the following important algorithms;

- De- Interlacing
- Color Space Conversion
- Automatic Numbered Plate Localization
- Image Enhancement

The following subsections explain the most common existing algorithms in the literature.

2.1. Existing De-Interlacing Algorithms

De-interlacing is a technique that combines the sequences of the even and the odd fields in a video frame. Figure 1and Figure 2 show the scanning techniques of the progressive and interlaced displays respectively [1].

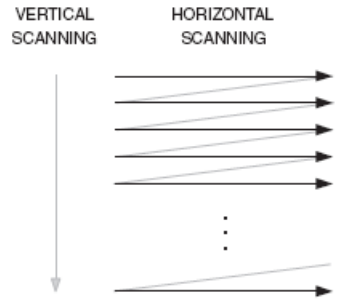


Figure 1. Progressive Displays (Scanning the line of an image consecutively, one after another)

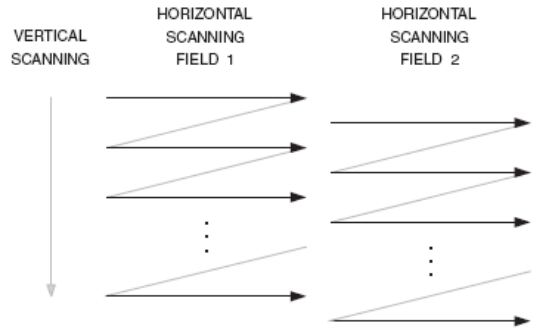


Figure 2. Interlaced Displays (Scanning the odd lines, then the even lines)

Since de-interlaced videos are combining separate fields, the resolution is increased; the quality of the video with motion would be greater. Nevertheless, unless an appropriate method is not used for the displays that are suitable for progressive videos, LCD or plasma display, de-interlacing artifacts occur. The following paragraphs give the information regarding the most common methods of de-interlacing to obtain a video without artifacts.

2.1.1. Weave De-Interlacing- Field Combination

Weave method includes the combining the consecutive fields as seen in Figure 3. Frame rate is half of the field rate. This method is suitable for frames without motion. However, if used for videos with motion, it results in artifacts called “mouse teeth” [1].

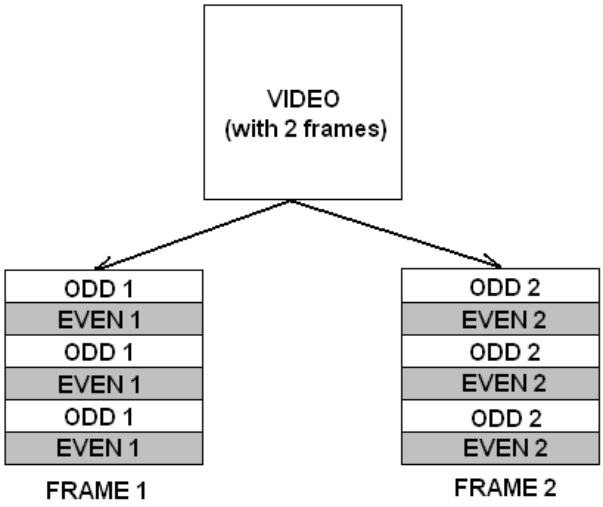


Figure 3. Weave De-Interlacing

2.1.2. Bob De-Interlacing (Line Doubling) - Field Extension

Bob method includes doubling the frame's only even or odd fields, as seen Figure 4. Frame rate is equal to the field rate but the spatial resolution is half of the original frame. By this method, mouse teeth artifact occurrence is prevented. However, for stationary videos, there is reduction in the quality of the video since vertical resolution is halved.

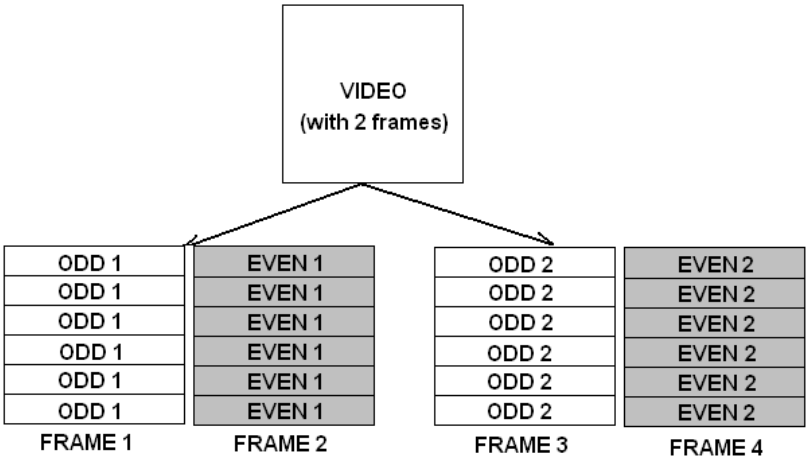


Figure 4. Bob (Line Doubling) De-Interlacing

2.1.3. Scan Line Interpolation De-Interlacing - Field Extension

Scan line interpolation method includes averaging the up and down line for the empty even or odd field as seen in Figure 5. Frame rate is equal to the field rate. By this method mouse teeth artifact occurrence is prevented, too. However, for stationary videos, there is still reduction in the quality of the video.

Despite the fact that this method is better than the Bob method theoretically, human eye can not detect the difference between them. Since the implementation of the scan line interpolation is more difficult than the Bob method on FPGA, line doubling is implemented in this work.

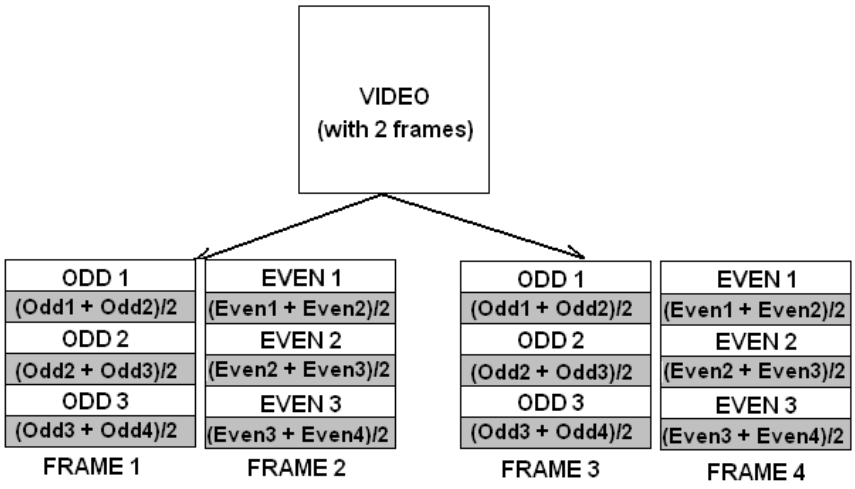


Figure 5. Scan line Interpolation De-Interlacing

2.1.4. Motion Adaptive De-Interlacing

Motion Adaptive De-Interlacing method exploits the Weave method for stationary sections of the frame and the Line Doubling method for moving sections of the frame. In the literature, there are several methods to find the motion in a frame; the most common method encompasses first defining the moving / stationary areas by looking at the consequent fields and then applying the necessary algorithms to the related areas of the frame [2] [3]. This method eliminates the drawbacks of the two

methods mentioned previously, but due to its implementation complexity, it is not used in this work.

2.2. Existing Color Space Conversion Algorithms – YCrCb to RGB

The YCbCr color space was developed as part of ITU-R BT.601 during the development of a world-wide digital component video standard [1]. YCbCr is a scaled and offset version of the YUV color space. Y is defined to have a nominal 8-bit range of 16–235; Cb and Cr are defined to have a nominal range of 16–240 [1]. There are several YCbCr sampling formats, such as 4:4:4, 4:2:2, 4:1:1.Y represents the overall brightness or luminance; Cr and Cb (chromaticity) represent the color information of a pixel.

The red, green, and blue (RGB) color space is widely used in computer graphics since color displays use red, green and blue to create the desired color. Red, green, and blue are three primary additive colors (individual components are added together to form a desired color) and are represented by a three-dimensional, Cartesian coordinate system. All three components of RGB are defined to have a nominal 8-bit range of 0–255 each.

The following subsections give the used conversion functions found in the literature.

2.2.1. Conversion Equation - 1

Microsoft's YUV color space to RGB color space conversion calculation is given by (2.1) [4].

$$\begin{aligned} \text{Red} &= 298 * (Y-16) + 498* (Cr-128) + 128 \\ \text{Green} &= 298 * (Y-16) – 100* (Cb - 128) - 208* (Cr-128) \\ \text{Blue} &= 298 * (Y-16) – 516* (Cb - 128) + 128 \end{aligned} \tag{2.1}$$

2.2.2. Conversion Equation - 2

A color in the YCbCr space is converted to the RGB color space by considering gamma-correction for 8 bit YCbCr values using the following equations (2.2) [6].

$$\begin{aligned} \text{Red} &= 1.164 * (Y - 64) + 1.596 * (Cr - 512) \\ \text{Green} &= 1.164 * (Y - 64) - 0.813 * (Cr - 512) - 0.392 * (Cb - 512) \\ \text{Blue} &= 1.164 * (Y - 64) + 2.017 * (Cb - 128) \end{aligned} \qquad (2.2)$$

2.3. Existing Automatic Numbered Plate Localization Algorithms

Plate localization is defined as finding the location of a plate in a given frame. Localization is the first step of Automatic Numbered Plate Recognition (ANPR) algorithms. In this subsection, three algorithms are presented; each of which uses different approaches and takes advantage of different features of plates.

2.3.1. Number Plate Localization on the Basis of Edge Finding

Number Plate Localization on the Basis of Edge Finding [9] is one of the most successful methods in the literature. It is based on the observation that the license plate is an area in the image with high contrasts, usually composed of black and white or black and yellow. The characters on the plate are organized in one row, or a few rows. The largest brightness variation of the rows is taken as the limit of the search. The edge finding algorithm is applied to the whole image as a first step for the purpose of highlighting the high contrast areas that are characteristic of number plates. Enyedi et al. [9] consider using a band-pass filter with finite response (FIR) as the best method for highlighting the high contrast areas, due to its fast execution. This kind of filters produces a sum of the inputs multiplied by relevant coefficients. The original and the filtered image can be seen in Figure 6. As a second step, vertical

position of the plate is found. It is established based on the amplitude of the rows' sums; by searching where the amplitude reaches its highest value.

The upper and lower ends of the plate are located in the following way: rows with values half of the maximal row is searched in both up and down directions until the first local minima is found. The upper and lower edge of the plate should be located near these minimums. To avoid getting stuck in local minimums caused by the image noise, the lowpass filter is applied to the vector of rows' sums. Figure 7 indicates the result of the procedure. The horizontal location of the plate is determined similarly, except the values are summed up for columns. In this case, a larger area might be selected as the possible location of the plate, because of the gaps between the letters, which are uniform. They are merged at the post processing phase, where the knowledge about the size and aspect ratio of the plate is used. The intersection on the Figure 7 shows the probable locations of the plate on the image. By searching the correct aspect ratio in this part, exact location is obtained.

Figure 6. The original image and image processed by FIR edge filter [9]

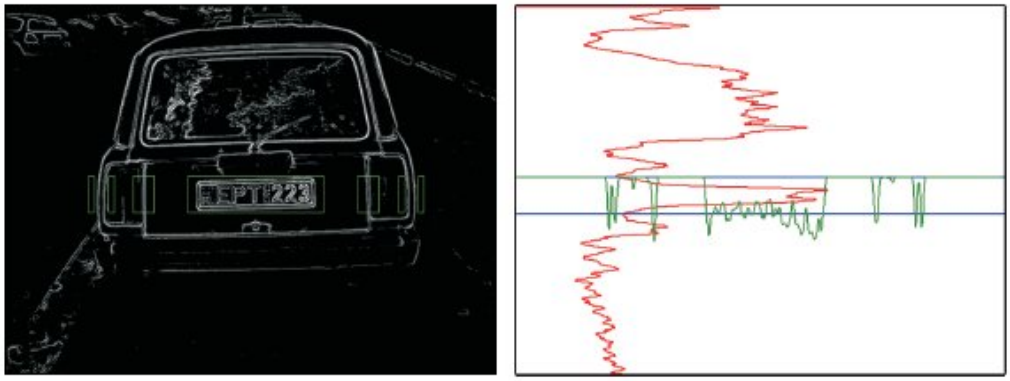


Figure 7. The localized license plate and the sums in rows with the local minimums marked and the sums in columns [9]

2.3.2. Number Plate Localization on the Basis of Adaptive Thresholding

The second method of locating the license plate is presented in [7]. It is based on adaptive thresholding. For the method to work correctly under a variety of lighting conditions, the authors decided to find the appropriate threshold iteratively. The process of the method can be seen in Figure 8. The main idea behind this approach is to find a threshold, which creates uniform black or white areas when applied to the input image. The only area to remain non-uniform is the license plate. The input data is a single 320 x 240 image. At first, the input image is preprocessed with a median 3x3 filter to reduce the noise. The first step of the thresholding algorithm is to binarize the image A to produce image B. The initial threshold can be set to $T = G_{max} - (G_{max} - G_{min}) / 3$. Where G_{min} and G_{max} are the minimal and maximal values of the image, respectively. The second step reduces the background disturbance and produces image C. This is done by recalculating the value of each pixel in the following way:

$$C(i; j) = | B(i; j) - B(i; j - 1)| \quad i = 0, ..., 319; j = 1, ..., 239$$

$$B(i; 0) \; j = 0: \tag{2.2}$$

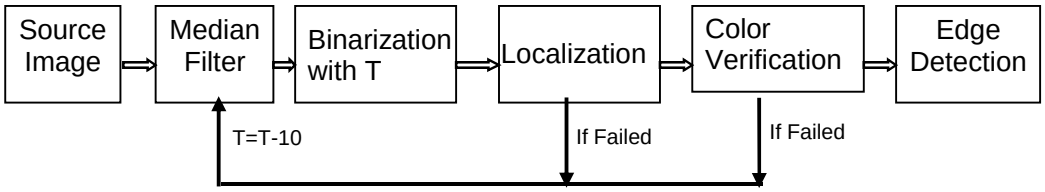


Figure 8. Adaptive Thresholding Algorithm [7]

After this operation, most of the background is usually set to 0. The remaining characters consist of thin insular vertical lines, while the background is irregular. The median filter $(1; 1; 1; 1; 1)_T$ is applied to the image C and image D is produced. The next and the third step is vertical localization of the plate. Image D is projected vertically (to a single column). Then this column is browsed from the bottom to the top until the value greater than a constant threshold is found. This is probably the bottom edge of the plate and it is labeled P_b. The browsing continues until a value less than t is found. This may be the localization of the plate's top and is labeled P_h. The height $H = P_b - P_h$ of the plate candidate must be between 10 and 30 pixels. If H is so, the next step can be performed. Otherwise the browsing of the column is carried on until reaching the top of the image, which indicates failure (another threshold value T must be tested). Horizontal localization is the fourth step. The image D is limited only to lines between P_h and P_l. Vertical projection is calculated. The horizontal position of the plate is established similarly as the vertical one. The only difference is that some short intervals of lower values are allowed showing the gaps between the characters. The left and the right edge are marked as P_l and P_r, respectively. If the candidate region has width $W = P_r - P_l$ between 40 and 90 pixels the color verification can be done, and a final decision is made.

2.3.3. Number Plate Localization on the Basis of Fuzzy Set Theory

The third approach concerning the plate localization problem is described in [10]. The main idea of this algorithm is splitting the 768 x 576 image into 75 x 25 tiles, for

each of which the fitness value is calculated. The plate is localized within the tile that has the highest fitness value. The fitness value calculation uses the fuzzy set theory.

The fuzzy set theory enables the transformation of a description in natural language into a mathematical formula. This description is the entry point of the algorithm. The authors defined a license plate as an object having the following features: a bright area with dark regions which is located in the middle or lower middle part of the image with a bright border and with dimensions: 530mm x 120mm. This description was, then transformed into the fuzzy system. Every feature was described with a membership value to the relevant fuzzy set. For example, the membership function of the pixel to the bright or dark class is shown at Figure 9. This membership value is used for computing the bright and dark pixel sequences' length. The length has also its membership function. Such functions are also created for the horizontal and vertical position of the tile. The overall fitness of the tile is calculated as the product of the membership values for the horizontal and, vertical positions, the average dark sequence length and the average bright sequence length. The tile with the highest overall fitness value contains the license plate. The exact borders are established according to the feature bright border.

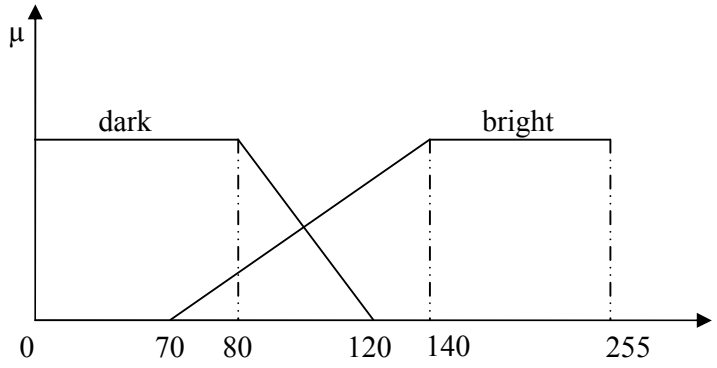


Figure 9. Membership functions of a pixel to the bright and dark fuzzy sets [10]Error! Reference source not found.

There are several other techniques using Mean Shift procedure [11] apart from these three presented.

2.4. Existing Image Enhancement Algorithms

To improve the quality of an image, to remove noise, and to enhance the details of the image, several image enhancement techniques are used. The following subsections give information about commonly used methods in the literature.

2.4.1. Image Enhancement with Rank Order Filter Method

The rank order filter, a nonlinear filter, is a popular algorithm in image processing systems. Noise removal and smoothing are two of the benefits of rank order filters. The median filter, which is a rank order filter, is especially useful in noise removal [12].

This filter works by analyzing a neighborhood of pixels around an origin pixel, for every valid pixel in an image. Often, a 3x3 area, or window of pixels is used to calculate its output. For every pixel in an image, the window of neighboring pixels is found. Then the pixel values are sorted in ascending, or rank, order. Next, the pixel in the output image corresponding to the origin pixel in the input image is replaced with the value specified by the filter order. Figure 10 shows an example of this algorithm for a median filter (order 5), a filter that is quite useful in salt -and-pepper noise filtering [13]. Since the rank order filter uses no arithmetic, a mathematical description is difficult to represent efficiently.

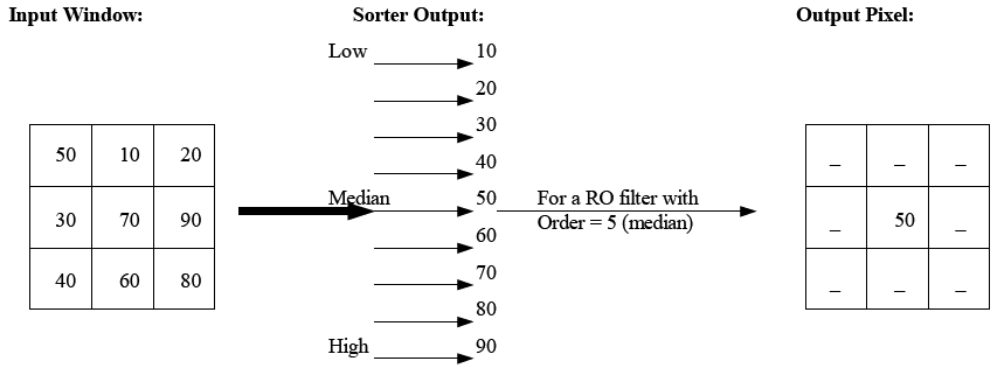


Figure 10. Rank Order Filter Operation [16]

2.4.2. Image Enhancement with Morphological Operators [14]

The term morphological image processing refers to a class of algorithms that utilizes the geometric structure of an image. Morphology can be used on binary and grayscale images, and is useful in many areas of image processing, such as skeletonization, edge detection, restoration, and texture analysis. A morphological operator uses a structuring element to process an image. We usually think of a structuring element as a window passing over an image, which is similar to the pixel window used in the rank order filter. Similarly, the structuring element can be of any size, but 3x3 and 5x5 sizes are common. When the structuring element passes over an element in the image, either the structuring element fits or does not fit. At the places where the structuring element fits, we achieve a resultant image that represents the structure of the image [14].

There are two fundamental operations in morphology: erosion and dilation [14]. It is common to think of erosion as shrinking (eroding) an object in an image. Dilation does the opposite; it grows the image object. Both of these concepts depend on the structuring element and how it fits within the object. For example, if a binary image is eroded, the resultant image is one where there is a foreground pixel for every origin pixel where its surrounding structuring element sized fit within the object. The output of a dilation operation is a foreground pixel for every point in the structuring element at a point where the origin fits within an image object [14].

Grayscale morphology is more powerful and more difficult to understand. The concepts are the same, but instead of the structuring element fitting inside a two -dimensional object; it is thought to either fit or not fit within a three -dimensional object. Grayscale morphology also allows the use of grayscale structuring elements. Binary structuring elements are termed flat structuring elements in grayscale morphology. The combination of grayscale images and grayscale structuring elements can be quite powerful [14]. One of the strongest features of morphological image processing extends from the fact that the basic operators, performed in different orders, can yield many different, useful results. For example, if the output of an erosion operation is dilated, the resulting operation is called an opening. The

dual of opening, called closing, is a dilation followed by erosion. These two secondary morphological operations can be useful in image restoration, and their iterative use can yield further interesting results, such as skeletonization and granulometries of an input image. Grayscale erosion and dilation can be achieved by using a rank order filter as well. Erosion corresponds to a rank order filter of minimum order, and dilation corresponds to a rank order filter of maximum order. The reason for this is that the result of a minimum order rank order filter is the minimum value in the pixel neighborhood, which is exactly what an erosion operation is doing. This also holds true for a maximum order rank order filter and a dilation operation. However, the rank order filter only works as a morphological operation with a flat structuring element. This is because the rank order filter window works as a sort of structuring element consisting of all ones. Still, this is a powerful feature, since grayscale morphology using flat structuring elements accounts for the most common usage of morphology.

2.4.3. Image Enhancement with Convolution

Convolution is another commonly used algorithm in DSP systems [15]. It belongs to a class of algorithms called spatial filters. Spatial filters use a wide variety of *masks*, also known as *kernels*, to calculate different results, depending on the function desired. For example, certain masks yield smoothing, while others yield low pass filtering or edge detection.

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. This value is the output for the origin pixel of the output image for that position.

The input pixel window is always the same size as the convolution mask. The output pixel is rounded to the nearest integer. When carried over an entire input image, this algorithm will result in an output image with reduced salt-and-pepper noise. An important aspect of the convolution algorithm is that it supports a virtually infinite

variety of masks, each with its own feature. This flexibility allows for many powerful uses.

CHAPTER 3

SYSTEM DESIGN

3.1. General View of the System

Field Programmable Gate Array (FPGA) technology has become an effective target for the implementation of algorithms suited to video image processing applications since parallel processing has been made possible. As image sizes and bit depths grow larger, software has become less useful in the video processing. In this thesis work FPGA board, video decoder board, a camera and a monitor are used as seen in Figure 11. Video decoder board gets the analog composite video and converts the video signal to digital (4:2:2 YCbCr format), then transfers the signal with the 27 MHz pixel clock to the FPGA Board. The video processing algorithms are applied to the image and then forwarded to the monitor. The FPGA board is configured by Xilinx ISE 9.2i software [17] by a PC. The following subsections give information about the sub-systems.

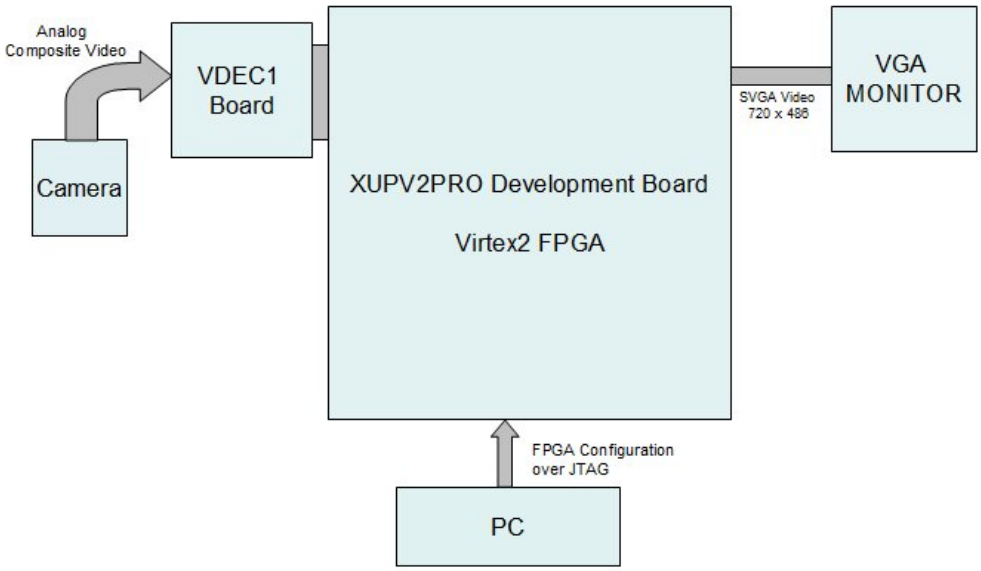


Figure 11. General View of the System

3.1.1. FPGA Development Board

Xilinx ML-402 [18] was the first choice for the implementation with Virtex-4 FPGA on it. This board is used for learning hardware design language and example implementations such as UART communication, generating images for VGA monitor, SRAM read/write, etc. Unfortunately, ML-402 board does neither support a video decoder nor a high speed connector for the on-board decoder board, so Xilinx XUPV2 [18] with Virtex-2 Pro FPGA was chosen for the final implementation. The following subsections describe the properties of both of the boards.

3.1.1.1. ML-402 Virtex-4 FPGA Evaluation Board

The ML-402 evaluation platform provides a Virtex-4 family - XC4VSX35-FF668-10 FPGA [20]. Main features of the platform are the following [18]:

- 64-MB DDR SDRAM, 32-bit interface running up to 266-MHz data rate
- One differential clock input pair and differential clock output pair with SMA connectors
- One 100-MHz clock oscillator and one extra open 3.3V clock oscillator socket

- General purpose DIP switches, LEDs, and push buttons
- Expansion header with 32 single-ended I/O, 16 LVDS capable differential pairs, 14 spare I/Os shared with buttons and LEDs, power, JTAG chain expansion capability, and IIC bus expansion
- Stereo AC97 audio codec with line-in, line-out, 50-mW headphone, and microphone-in (mono) jacks
- RS-232 serial port
- 16-character x 2-line LCD display
- One 4-Kb IIC EEPROM
- VGA output - 140 MHz / 24-bit video DAC
- PS/2 mouse and keyboard connectors
- System ACE™ CompactFlash configuration controller with CompactFlash connector
- ZBT synchronous SRAM - 9 Mb on 32-bit data bus with four parity bits
- Intel StrataFlash (or compatible) linear flash chips (8 MB)
- 10/100/1000 tri-speed Ethernet PHY transceiver
- USB interface chip (Cypress CY7C67300) with host and peripheral ports
- Xilinx XC95144XL CPLD to allow linear flash chips to be used for FPGA configuration
- Xilinx XCF32P Platform Flash configuration storage device
- JTAG configuration port for use with Parallel Cable III or Parallel Cable IV cable
- Onboard power supplies for all necessary voltages
- 5V @ 3A AC adapter
- Power indicator LED

The block diagram of the board can be seen in Figure 12.

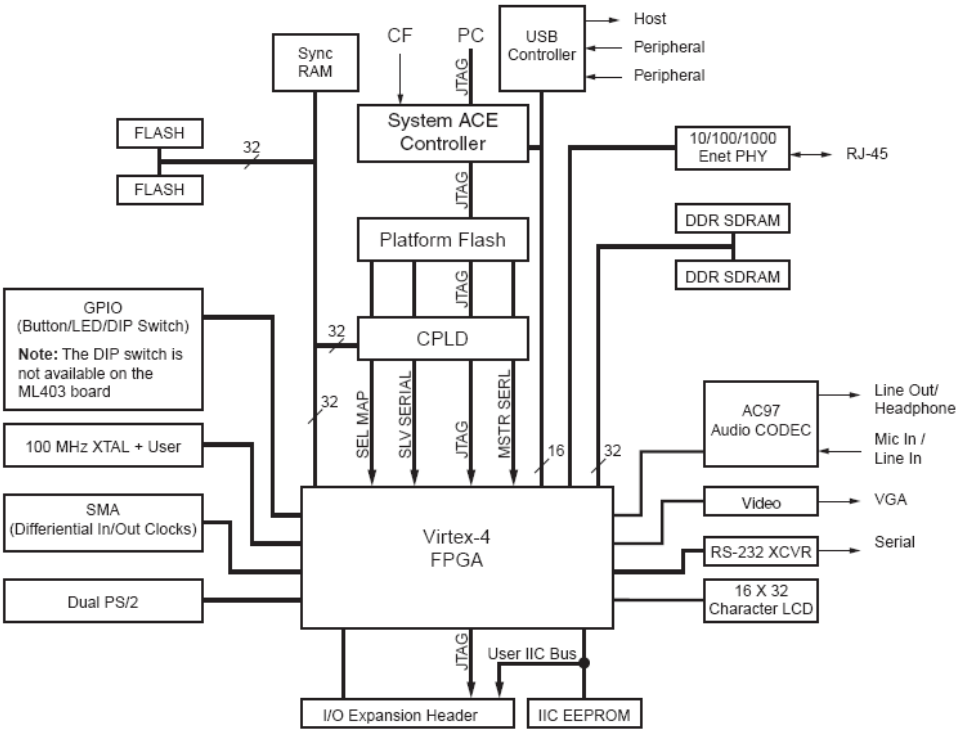


Figure 12. Block Diagram of ML-402 [18]

3.1.1.2. XUPV2 Pro Virtex-2 FPGA Evaluation Board

The XUP Pro Virtex-2 Evaluation Board is an advanced hardware platform that consists of a high performance Virtex-II Pro Platform FPGA [23] surrounded by peripheral components that can be used to create a complex system [18]. Main features of the platform are the following [18]:

- Virtex®-II Pro FPGA with PowerPC® 405 cores
- Maximum 2 GB of Double Data Rate (DDR) SDRAM
- CompactFlash connector
- Embedded Platform Cable USB configuration port
- Programmable Configuration PROM
- On-board 10/100 Ethernet PHY device
- RS-232 DB9 serial port
- Two PS-2 serial ports

- Four LEDs connected to Virtex-II Pro I/O pins
- Four switches connected to Virtex-II Pro I/O pins
- Five push buttons connected to Virtex-II Pro I/O pins
- Six expansion connectors joined to 80 Virtex-II Pro I/O pins with over-voltage protection
- High-speed expansion connector joined to 40 Virtex-II Pro I/O pins
- AC-97 audio CODEC with audio amplifier and speaker/headphone output
- Microphone and line level audio input
- On-board XSGA output, up to 1200 x 1600 at 70 Hz refresh
- Three Serial ATA ports, two Host ports and one Target port
- Off-board expansion MGT link, with user-supplied clock
- 100 MHz system clock, 75 MHz SATA clock
- Provision for user-supplied clock
- On-board power supplies
- Power-on reset circuitry
- PowerPC 405 reset circuitry

The block diagram of the board can be seen in Figure 13.

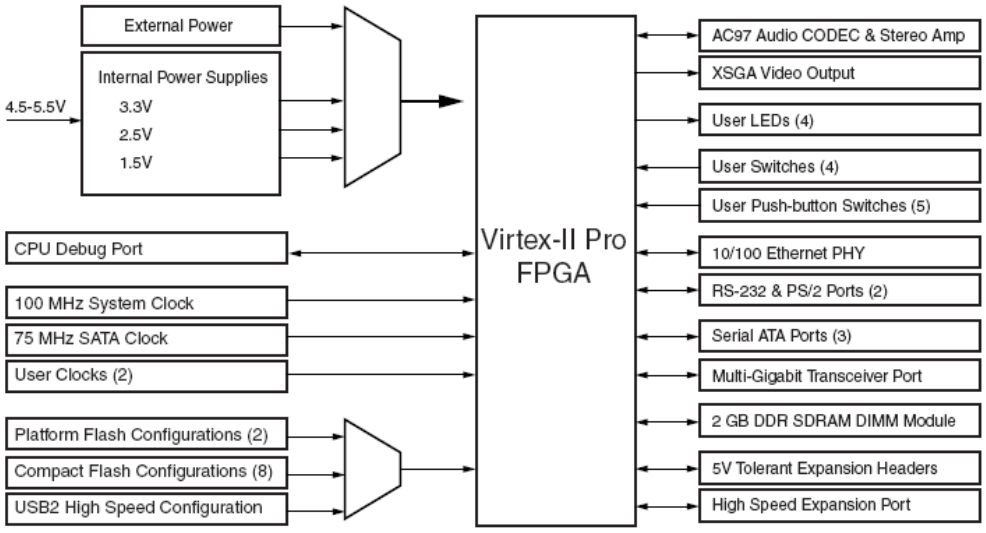


Figure 13. XUP V2 Pro Development Plaform [18]

The photo of the board can be seen in Figure 14.

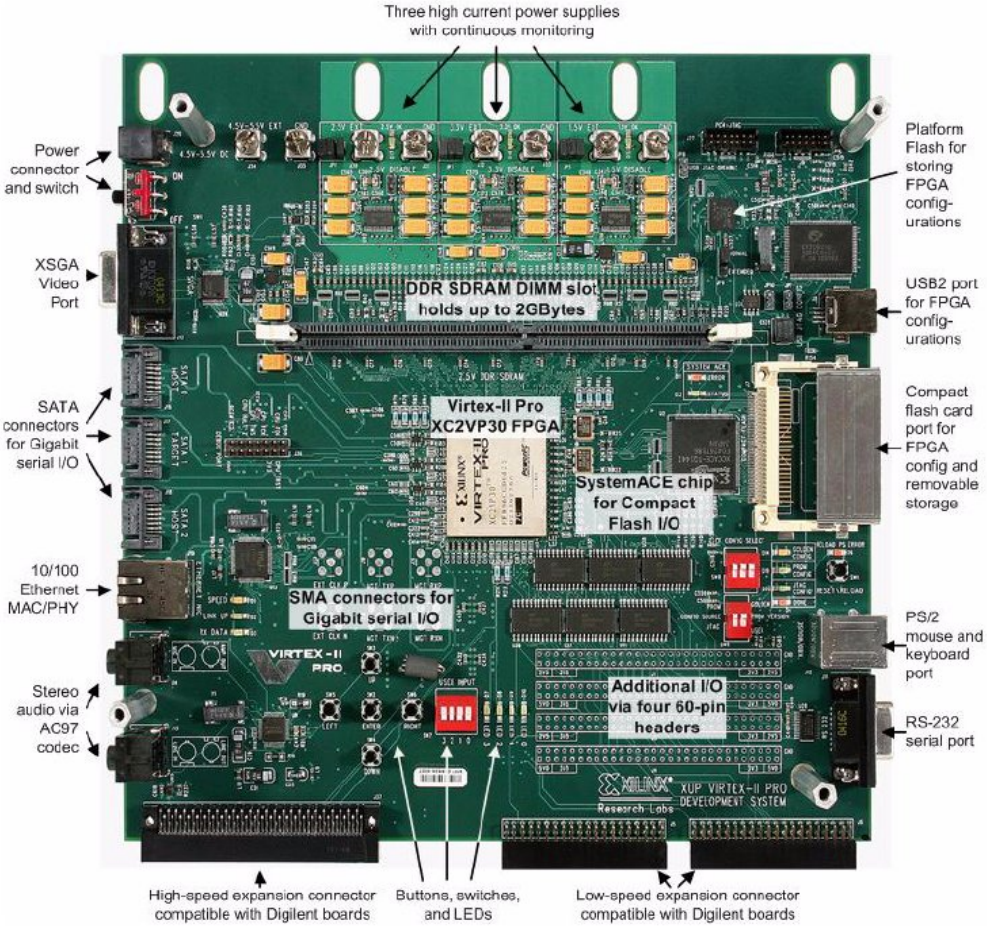


Figure 14. XUPV2 Pro Board [18]

The two main features used in this thesis of this board are high speed expansion connector and the video encoder. The expansion connector is compatible with the VDEC1 board. The video encoder provides operation with desired pixel clock at 27 MHz.

3.1.2. VDEC1 Board [23]

To digitize the analog video input, Digilent's [23] video decoder VDEC1 board is used. Video decoder board is built around ADV7183B, the video decoder chip of Analog Devices. It can digitize NTSC, PAL and SECAM video signals into YCbCr

4:2:2 component video data-compatible with 8-bit ITU-R BT.656 [23]. The board is connected to the XILINX XUPV2 Evaluation board via a Hirose FX2 connector. Figure 15 shows the general look of the VDEC1 board. Different video inputs, composite, RGB and S-video are supported through the provided video input connectors.

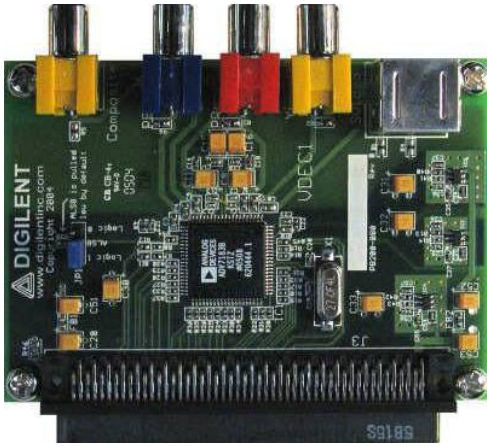


Figure 15. VDEC1 Board [23]

Figure 16 shows the block diagram of VDEC1 board.

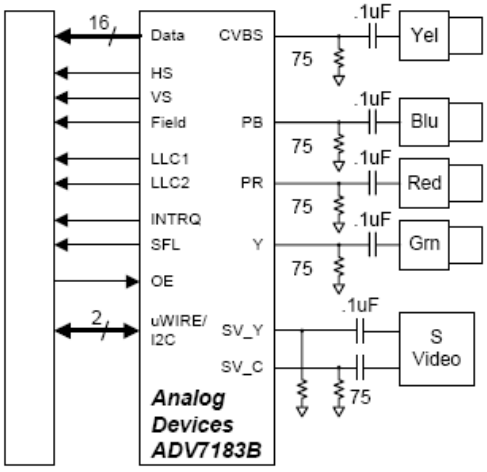


Figure 16. Block Diagram of VDEC1 Board [23]

3.1.2.1. ADV 7183 Video Decoder [23]

The ADV7183B integrated video decoder automatically detects and converts standard analog video signal standards (NTSC, PAL, and SECAM) into 4:2:2 component video data-compatible with 8-bit ITU-R BT.656 [25]. Functional block diagram of the ADV7183 is seen in Figure 17.

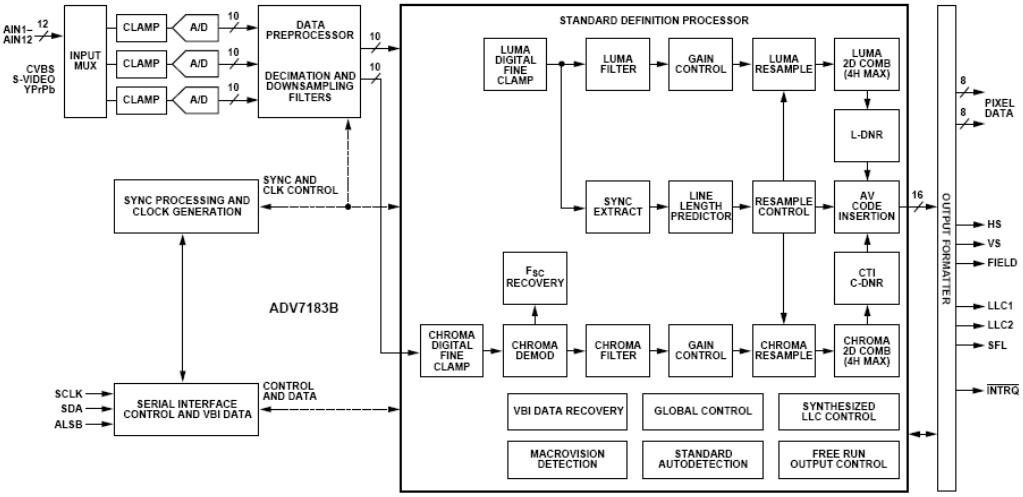


Figure 17. Functional Block Diagram of ADV 7183 [23]

The advanced and highly flexible digital output interface enables performance video decoding and conversion in line locked clock-based systems. The analog input channels accept standard composite, S-Video, YPrPb video signals. The fixed 54 MHz clocking of the ADCs and data path for all modes allows very precise, accurate sampling and digital filtering. The ADV7183B modes are set up over a 2-wire, serial, bidirectional port (I^2C-compatible).

3.2. Design Language

In order to create an FPGA design, a designer has several options for algorithm implementation. While gate-level design can result in optimized designs, the learning curve is considered difficult for most engineers, and the knowledge is not portable

across FPGA architectures. The following subsections discuss the two common high-level hardware design languages (HDLs) in which FPGA algorithms are designed.

3.2.1. VHSIC Hardware Design Language (VHDL)

In recent years, VHSIC (Very High Speed Integrated Circuit) Hardware Design Language (VHDL) has become a sort of industry standard for high-level hardware design. Since it is an open IEEE standard [26], it is supported by a large variety of design tools and is quite interchangeable (when used generically) between different vendors' tools. It also supports inclusion of technology-specific modules for most efficient synthesis to FPGAs.

The first version of VHDL, IEEE 1076-87, appeared in 1987 and has since undergone an update in 1993, appropriately titled IEEE 1076-93. It is a high-level language similar to the computer programming language Ada, which is intended to support the design, verification, synthesis and testing of hardware designs.

In this thesis work, VHDL is chosen as the design language since the adoption is easier than Verilog.

3.2.2. Verilog Hardware Design Language

Originally intended as a simulation language, Verilog HDL represents a formerly proprietary hardware design language. Currently Verilog can be used for synthesis of hardware designs and is supported in a wide variety of software tools. It is similar to the other HDLs, but its adoption rate is decreasing in favor of the more open standard of VHDL. Still, many designers favor Verilog over VHDL for hardware design, and some design departments use only Verilog. Therefore, as a hardware designer, it is important to at least be aware of Verilog.

3.3. Video Interfaces

Video Decoder Board supports NTSC, PAL and SECAM analog video formats. Both NTSC and PAL input video characteristics are defined by the design. The following subsections give information about the analog PAL and NTSC video formats.

3.3.1. PAL Composite Video Interface

The Phase Alternate Line (PAL) is one of the worldwide used analog television encoding systems. In addition to the luminance and synchronization signals, chrominance signal is also carried by PAL signals. It supports 4.43 MHz color carrier frequency. The vertical synchronization is 50 Hz that means about 50 frames per second and the vertical synchronization is 16.625 kHz that means 16.625 lines per second. The resolution of the PAL video signal is 864 x 625.

3.3.2. NTSC Composite Video Interface

The National Television System Committee (NTSC) is the analog television system. In addition to the luminance and synchronization signals, chrominance signal is also carried by NTSC signals. It supports 3.579545 MHz color carrier frequency. The vertical synchronization is 59.94 Hz that means about 60 frames per second and the vertical synchronization is 15.734 kHz that means 15.734 lines per second. The resolution of the NTSC video signal is 858 x 525.

This chapter has given brief information about the development platform, hardware design languages and analog video standards.

CHAPTER 4

IMAGE ENHANCEMENT USING VIDEO STREAM

Considering the modularity and the testability of the system, different blocks are designed and tested. Figure 18 indicates all the system blocks and the following subsections describe the functionality of each block.

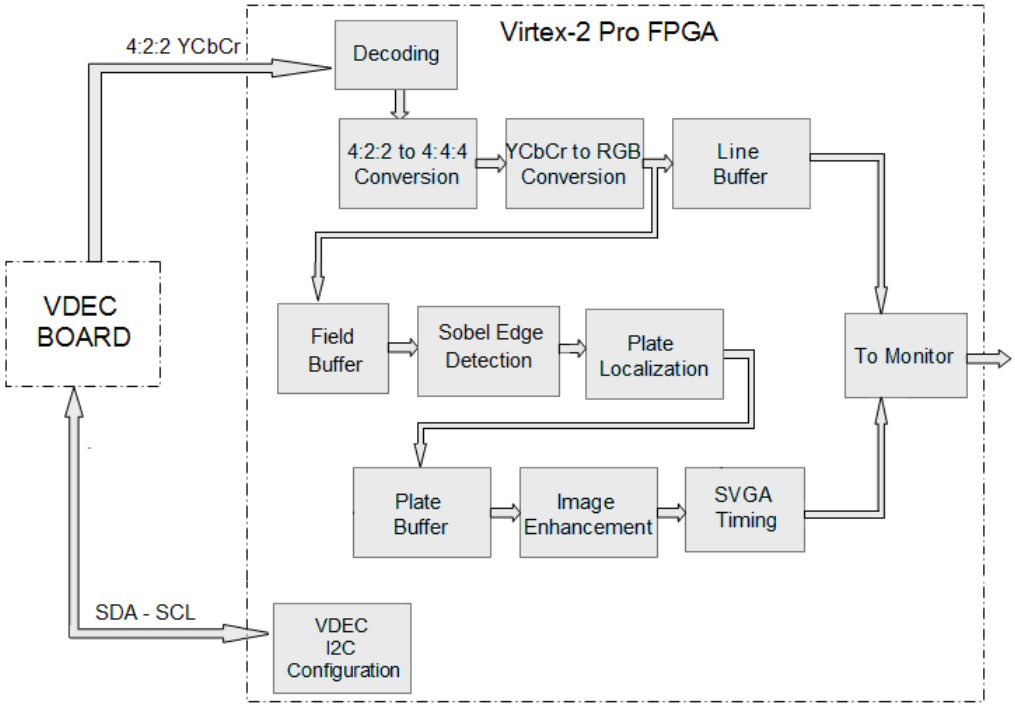


Figure 18. Main Blocks of the System

4.1. Video Receiver

In order to process video signals, analog to digital conversion must be done. The video decoder board performs this conversion. Since the decoder has the capability to accept different formats of video signals, the configuration must be done first to have

the desired output. After configuring the decoder, the data from the decoder must be separated to video data and synchronization data. Section 4.1.1 describes the VDEC I^2C configuration block and 4.1.2 describes the decoding, 4:2:2 to 4:4:4 and YCbCr to RGB conversion blocks. Section 4.1.3 describes the de-interlacing functionality and finally Section 4.1.4 describes the timing adjustment block.

4.1.1. Video Decoder Configuration

The ADV7183B supports Inter-Integrated Circuit Serial Bus Interface (I^2C). The configuration is accomplished by the I^2C interface. The following subsections provide information about the I^2C bus and the registers of the ADV7183B.

4.1.1.1. I2C Interface

Two inputs, serial data (SDA) and serial clock (SCLK), carry information between the ADV7183B and the FPGA, I^2C master controller. Each slave device has a unique address in I^2C communication. The ADV7183B's I^2C port allows the user to set up and configure the decoder and to read back captured data. The ADV7183B has two possible slave addresses for both read and write operations, depending on the logic level on the ALSB pin. These four unique addresses are shown in following Table 1. The ADV7183B's ALSB pin controls Bit 1 of the slave address. The LSB (Bit 0) sets either a read or write operation. Logic 1 corresponds to a read operation; Logic 0 corresponds to a write operation. In this work, ALSB bit is set to 1, so the slave address for write is 0x40.

Table 1. I^2C Addresses for ADV7183B

ALSB	R/W	Slave Address
0	0	0x40
0	1	0x41
1	0	0x42
1	1	0x43

To control the device on the bus, a specific protocol must be followed. First, the master, FPGA initiates a data transfer by establishing a start condition, which is defined by a high-to-low transition on SDA while SCLK remains high. A low-to-high transition on the SDA line while SCL is high defines a stop condition. Start and stop conditions (seen in Figure 19) are always generated by the master. The bus is considered to be busy after the Start condition. The bus is considered to be free again a certain time after the stop condition.

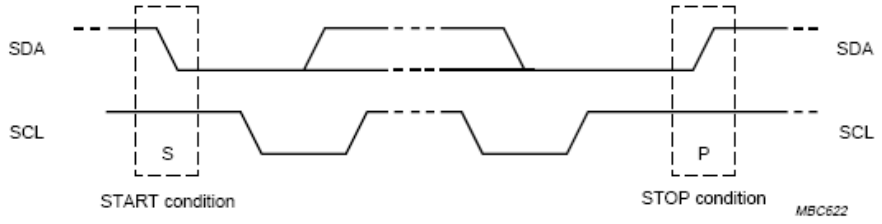


Figure 19. I2C Start and Stop Conditions [23]

Start bit indicates that an address/data stream will follow. All peripherals respond to the start condition and shift the next eight bits (7-bit address + R/W bit). The bits are transferred from MSB down to LSB. The peripheral that recognizes the transmitted address responds by pulling the data line low during the ninth clock pulse; this is known as an acknowledge bit. All other devices withdraw from the bus at this point and maintain an idle condition. The idle condition is where the device monitors the SDA and SCLK lines and waits for the start condition. The ADV7183B acts as a standard slave device on the bus. The data on the SDA pin is eight bits long, supporting the 7-bit addresses and the R/W bit. The ADV7183B has 249 sub addresses to enable access to the internal registers. It therefore interprets the first byte as the device address and the second byte as the starting sub address. A data transfer is always terminated by a stop condition. Stop and start conditions can be detected at any stage during the data transfer. If these conditions are asserted out of sequence with normal read and write operations, they cause an immediate jump to the idle condition. During a given SCLK high period, the user should only issue one

start condition, one stop condition, or a single stop condition followed by a single start condition.

Figure 20 summarizes the bus communication; first a start condition occurs, then address of the ADV7183B (first 7 bit is fixed, 8. bit indicates whether the sequence is write or read) is implemented on the SDA line, an acknowledge bit occurs, then sub address of the register is implemented following by an acknowledge, then the data of the register with an acknowledge occurs and finally stop condition stops the communication on the bus.

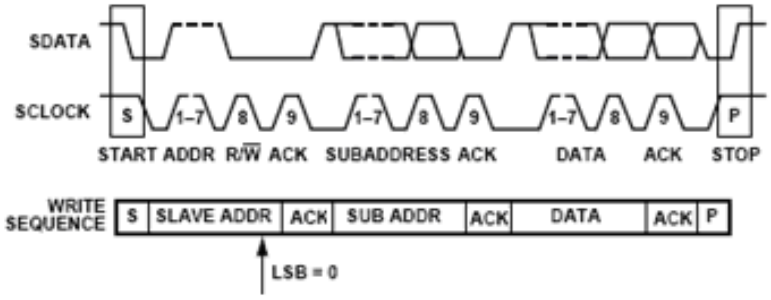


Figure 20. Write Sequence of ADV7183B [23]

4.1.1.2. Video Decoder Register Information

In order to get the required video input from the decoder, registers of ADV7183B must be set to the correct values. To get 8 bit 4:2:2, ITU-R BT.656 compatible video data with composite video input and 27 MHz clock, the following register values shall be implemented. Table 2 shows the used registers, values and the descriptions.

Table 2. ADV7183B Set Register Addresses and Values [23]

Register Address	Register Value	Description
0x00	0x04	CVBS Video Input on analog channel 5.
0x15	0x00	Slow down digital clamps
0x17	0x41	Recommended setting of filter adjustment
0x27	0x58	Pixel Delay Control recommended setting
0x3A	0x16	Power Down ADC1 and ADC2 recommended setting
0x50	0x04	Maximum edge noise control recommended setting
0x0E	0x80	Recommended setting for ADI control
0x50	0x20	DNR Noise Threshold Recommended setting
0x52	0x18	Recommended setting
0x58	0xED	Recommended setting
0x77	0xC5	Recommended setting
0x7C	0x93	Recommended setting
0x7D	0x00	Recommended setting
0xD0	0x48	Recommended setting
0xD5	0xA0	Recommended setting
0xD7	0xEA	Recommended setting
0xE4	0x3E	Recommended setting
0xEA	0x0F	Recommended setting
0x0E	0x00	Recommended setting

4.1.2. Digital Video

4.1.2.1. ITU-R BT 656 4:2:2 YCrCb Video Format

This subsection defines the interface for transmitting 4:2:2 YCbCr digital video between decoder and FPGA. Active video resolutions are either 720 x 486 (525/60 video systems- NTSC) or 720 x 576 (625/50 video systems - PAL).

The BT.656 parallel interface [25] uses 8 bits of multiplexed YCbCr data and a 27 MHz clock. Conventional video timing signals (HSYNC, VSYNC, and BLANK) also being transmitted by video decoder however BT.656 uses unique timing codes embedded within the video stream. This reduces the number of wires and IC pins required for a BT.656 video interface.

The 4:2:2 YCbCr data is multiplexed into an 8-bit stream: $Cb_0Y_0Cr_0Y_1Cb_2Y_2Cr_2$, etc. Figure 21 and Figure 22 illustrate the format for 525/60 and 625/50 video systems, respectively.

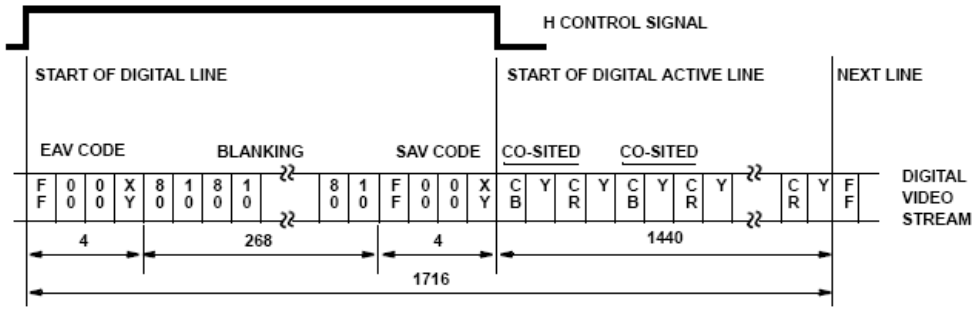


Figure 21. BT.656 8-bit Parallel Interface Data Format for NTSC (525/60) Video Stream [25]

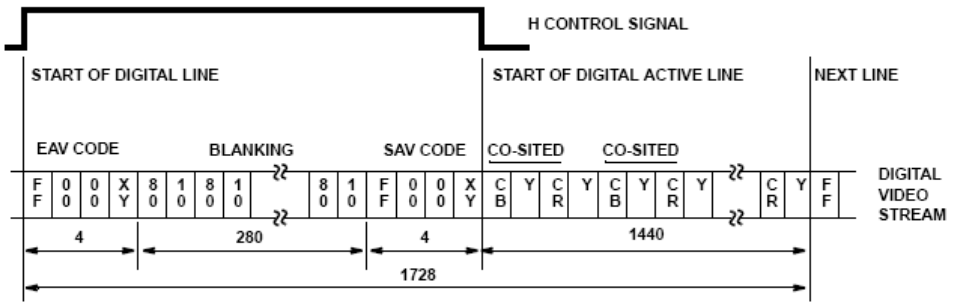


Figure 22. BT.656 8-bit Parallel Interface Data Format for PAL (625/50) Video Stream [25]

SAV (start of active video) and EAV (end of active video) codes are embedded within the YCbCr video stream. They eliminate the need for the HSYNC, VSYNC, and BLANK timing signals normally used in a video system. The XY status word sequence is shown in 0.

Table 3. BT.656 XY Status Word

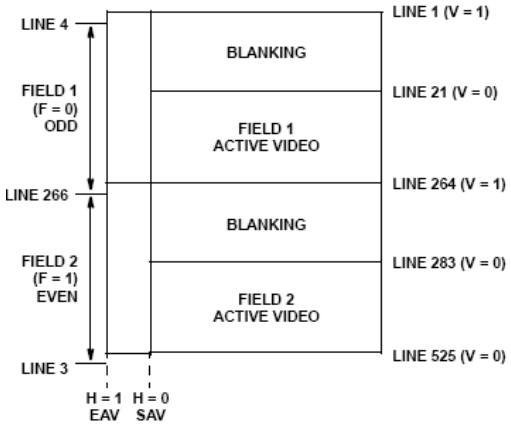
	D7	D6	D5	D4	D3	D2	D1	D0
Status Word	1	F	V	H	P3	P2	P1	P0

The XY status word, which also indicates whether it is an SAV or EAV sequence, is defined as:

- F = 0 for odd fields; F = 1 for even fields
- V = 1 during vertical blanking
- H = 0 at SAV, H = 1 at EAV
- P3-P0 = protection bits
    - P3 = V xor H
    - P2 = F xor H
    - P1 = F xor V
    - P0 = F xor V xor H

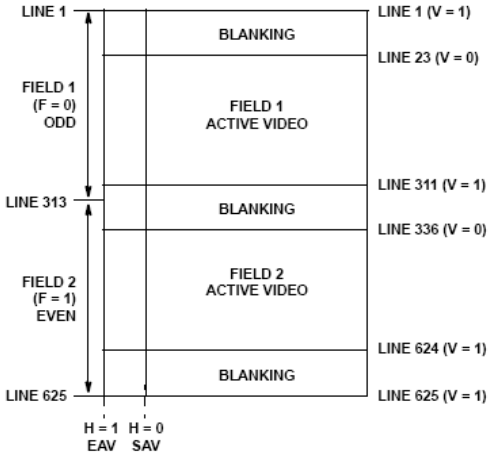
These protection bits enable single-bit errors to be detected and corrected.

Figure 23 and Figure 24 indicates the blanking periods, EAV and SAV intervals both for NTSC and PAL systems.



LINE NUMBER	F	V	H (EAV)	H (SAV)
1-3	1	1	1	0
4-20	0	1	1	0
21-263	0	0	1	0
264-265	0	1	1	0
266-282	1	1	1	0
283-525	1	0	1	0

Figure 23. BT.656 Vertical Blanking Intervals and EAV, SAV Sequences for NTSC [25]



LINE NUMBER	F	V	H (EAV)	H (SAV)
1-22	0	1	1	0
23-310	0	0	1	0
311-312	0	1	1	0
313-335	1	1	1	0
336-623	1	0	1	0
624-625	1	1	1	0

Figure 24. BT.656 Vertical Blanking Intervals and EAV, SAV Sequences for PAL [25]

In this thesis, instead of synchronization signals, status word is used. First the data sequence, FF, 00, 00 is waited then XY coded is decoded as F, H and V. These signals are processed to determine the active video and blanking intervals. Since the data sequence is waited, 4 clock duration of delay occurs. In order to compensate this delay; video timing signals are also delayed by a pipeline delay block. The main

functionalities of the decoding block are obtaining the digital video data from the video decoder; extracting the video timing information from the input signal; decoding the input signal into 4:2:2 YCrCb format; providing the timing information.

4.1.2.2. 4:2:2 YCrCb to 4:4:4 YCrCb Conversion

Since the human eye is less sensitive to color than luminance [28], bandwidth can be optimized by storing more luminance detail than color detail. At normal viewing distances, there is no perceptible loss incurred by sampling the color detail at a lower rate. In video systems, this is achieved through the use of color difference components. The signal is divided into a luma (Y) component and two color difference components (chroma – Cb and Cr). In YCbCr 4:2:2, the two chroma components are sampled at half the sample rate of luma, so horizontal chroma resolution is cut in half. This reduces the bandwidth of a video signal by one-third with little or no visual difference.

The 4:2:2 video data to 4:4:4 data conversion is required to get the desired video data. The process of 4:2:2 to 4:4:4 conversion consists simply of creating the missing Cr and Cb components. This can be accomplished by duplicating the Cr and Cb information. Figure 25 indicates the sampling forms of both 4:2:2 and 4:4:4.

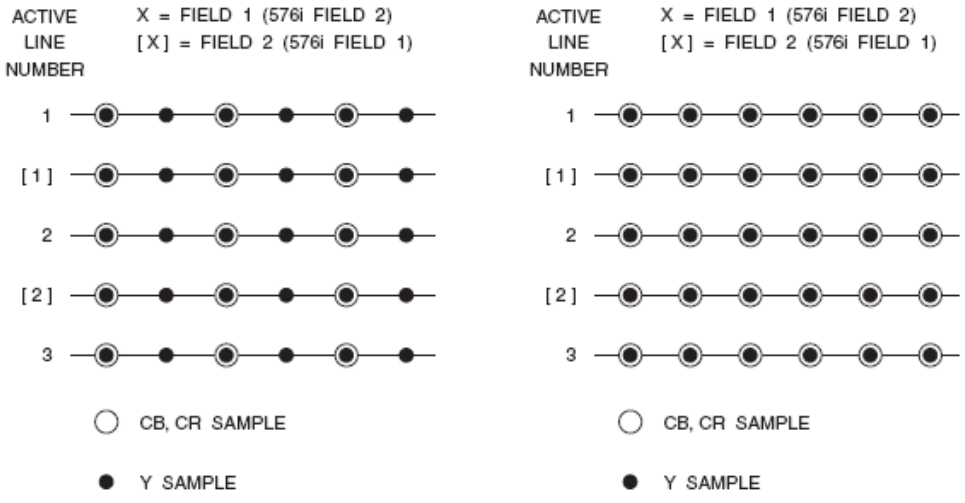


Figure 25. 4:2:2 YCbCr Sampling (left hand side), 4:4:4 YCbCr Sampling (right hand side) [1]

Each line of video sampled at 27 MHz, generates 16-bit 4:2:2 YCbCr data, resulting in 720 active samples of Y per line, and 360 active samples each of Cb and Cr per line. This data is converted to 13.5 MHz, generating 720 active samples of 24-bit 4:4:4 YCbCr data.

The Y data and the CbCr data are demultiplexed, and the 27 MHz sample clock rate is increased by two times to 13.5 MHz as seen Figure 26.

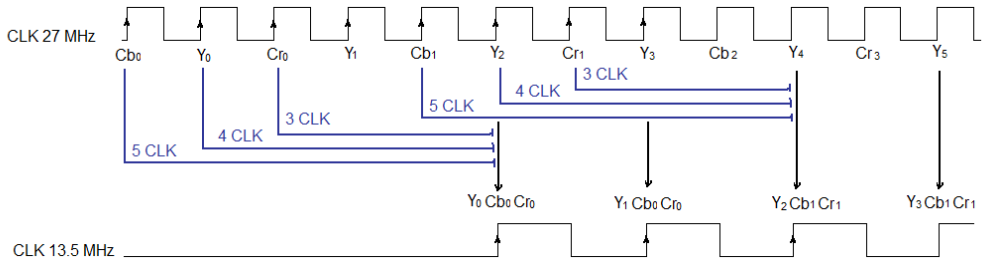


Figure 26. 4:2:2 YCbCr Data Generation

4.1.2.3. Color Space Conversion – YCrCb to RGB Conversion

This block converts the signal from the YCrCb format into RGB format as seen in equation 4.1 (color space conversion). Also, to ensure that the calculated values of RGBs are in range of 0 to 255, a comparison process is done in this block.

$$\begin{aligned} R &= 1.164*(Y-64)+1.596*(Cr-512) \\ R &= 1.164*(Y-64)-0.813*(Cr-512)-0.392*(Cb-512) \\ R &= 1.164*(Y-64)-2.017*(Cb-512) \end{aligned} \qquad (4.1)$$

4.1.3. De-Interlacing

In this thesis, line duplication is chosen as the de-interlacing technique since with optimized effort, quality of the video is so good that the human eye can not detect the difference from more complex techniques. For line duplication, 2 block rams are

used for buffering 1 line. The following subsection describes the usage of block rams and the control.

4.1.3.1. Line Buffer Control

Two Block RAMs (line buffer) are used as line buffers. By default, first is used in write mode and the second is used in read mode. For the write mode, a 13.5 MHz clock is used and for read mode a 27 MHz is used. It switches the functions of reading and writing at the end of the Read or Write Mode, i.e. after the first line is written on the first block ram, second block ram is switched to the write mode for the second line and first block ram is switched to read mode. One line write period is about 53.3 μs and read period is 26.7 μs. By line duplication, 1 line is written 2 times, that is to say reading 2 lines takes the same time as writing one line. In this block, only one line delay occurs while displaying the image.

4.1.3.2. Field Buffer Control

For the plate localization processing, at least 1 video field - 720 pixels x 242 lines (for NTSC video input) - should be buffered. Since plate localization algorithm is performed on the luma (Y) component of the analog video input, for one pixel four bit Y data is stored. For decreasing the used memory area four most significant bits of the luma component is used for one pixel value. The minimum memory area is 720 x 242 x 4 bit = 696690 bits. The internal block ram area of Virtex-2 pro FPGA is about 2 Mbits. About 700 kilobit block ram area is feasible for the process time and efficiency. As a default state, field buffer is at write mode as line buffer. When find the plate statement is true, it waits until the total field is stored; then is changes its state from write to read. Again for de-interlacing, it reads one line twice to achieve line-doubling method. It generates a gray scale image which is used for the localization algorithm.

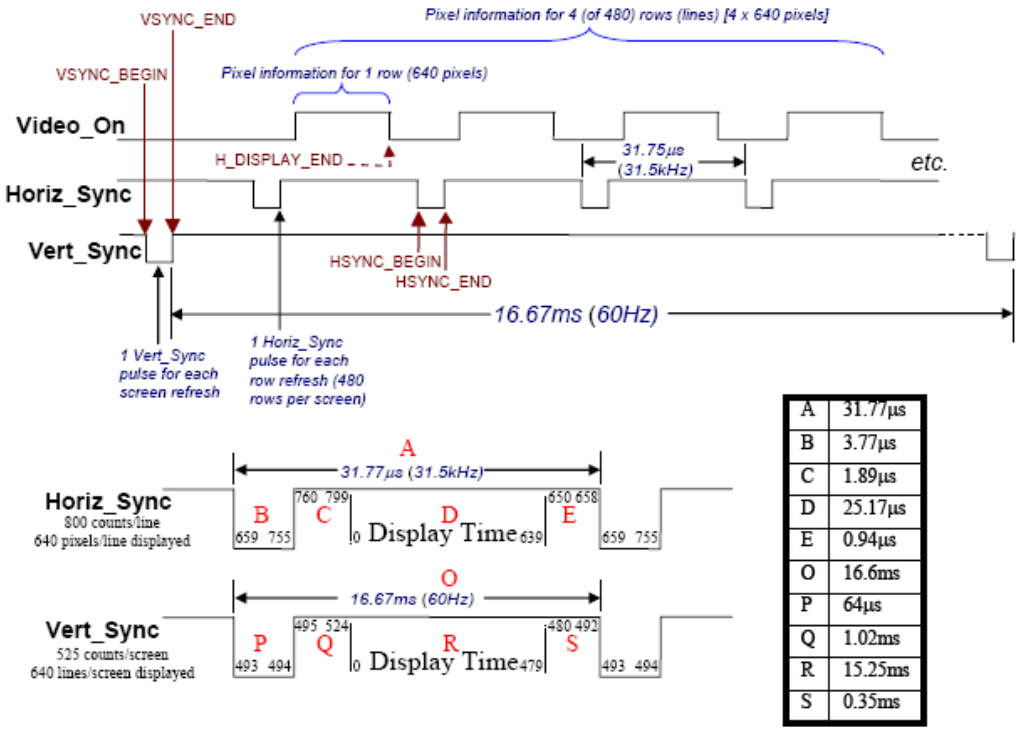
4.1.4. Video Timing Adjustment

This block is used to generate an appropriate HSYNC, VSYNC, and blanking pixel clock from the timing data which is obtained from the video decoding block. The

following subsections define the necessary timing information for VGA and SVGA respectively.

4.1.4.1. VGA Timing Generation

VGA active video has 640 x 480 resolution with 25 MHz pixel clock and 60 Hz refresh frequency. However, the total pixel resolution including blanking and synchronization pixels is 800 x 525. Figure 27 shows the timing information.

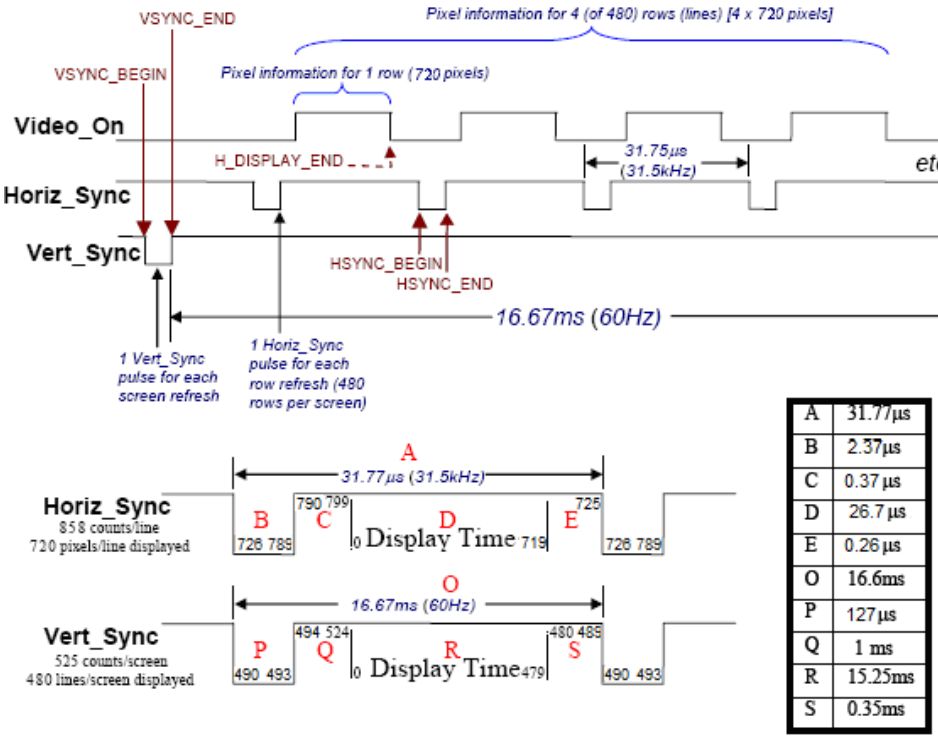


A	31.77µs
B	3.77µs
C	1.89µs
D	25.17µs
E	0.94µs
O	16.6ms
P	64µs
Q	1.02ms
R	15.25ms
S	0.35ms

Figure 27. VGA Timing Diagram[27]

4.1.4.2. SVGA Timing Generation

SVGA active video has 720 x 480 resolution with 27 MHz pixel clock and 60 Hz refresh frequency. However, the total pixel resolution including blanking and synchronization pixels is 858 x 525. Figure 27 indicates the timing information.



A	31.77µs
B	2.37µs
C	0.37 µs
D	26.7 µs
E	0.26 µs
O	16.6ms
P	127 µs
Q	1 ms
R	15.25ms
S	0.35ms

Figure 28. SVGA Timing

4.2. Number Plate Localization

There is a huge variety of license plate types. Only white/yellow and single row plates are addressed in this work. Also, since the edge detection method is used, all plates are assumed to have a light background (white or yellow) with dark characters (black, blue or brown). Moreover, to decrease the process time, it is assumed that the camera is located in a fixed position that the plate on the image is not close the borders. That is to say, while the calculations are handled instead of beginning from first row, it begins from 40^{th} pixel and stop at 680^{th} pixel instead of 720^{th} pixel.

A simple, effective and fast edge finding algorithm must be applied in the first step, which sufficiently highlights the characters of the number plate in contrast to the intensity of the other areas in the image. Sobel Edge Detection is selected for the first step, and then a plate localization algorithm is applied to the binary edge image. The following sub-sections describe the details of the algorithms.

4.2.1. Sobel Filter

The Sobel filter performs a 2-D spatial gradient measurement on an image. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image. The Sobel filter uses a pair of 3x3 convolution windows, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows). The window is slid over the image, manipulating a square of pixels at a time.

If we define A as the gray scale source image, and G_x and G_y are two images which at each point contain the horizontal and vertical derivative approximations, the computations are shown in equations 4.2 and 4.3 for horizontal and vertical gradient window, respectively.

$$G_x = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \tag{4.2}$$

$$Gy = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \tag{4.3}$$

Then the magnitude of the gradient is calculated using the equation 4.4.

$$|G| = \sqrt{(G_x)^2 + (G_y)^2} \tag{4.4}$$

An approximate magnitude can be calculated using equation 4.5.

$$|G| = |G_x| + |G_y| \tag{4.5}$$

If A is denoted as the pixel values as in equation 4.6; the gradient value is shown in equation 4.7.

$$A = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \end{bmatrix} \tag{4.6}$$

$$\begin{aligned} p_5 = |G| = &\left|(p_1 - p_7) + 2x(p_2 - p_8) + (p_3 - p_9)\right| + \\ &\left|(p_1 - p_3) + 2x(p_4 - p_6) + (p_7 - p_9)\right| \end{aligned} \tag{4.7}$$

4.2.2. Edge Detection

Edges characterize boundaries and are therefore used in plate localization. Edges in images are areas with strong intensity contrasts – a jump in intensity from one pixel to the next. Edge detecting an image significantly reduces the amount of data and filters out useless information, while preserving the important structural properties in an image. While there are many ways to perform edge detection; gradient (Sobel) method is chosen for this thesis. The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image. An edge has the one-dimensional shape of a ramp and calculating the derivative of the image can highlight its location.

A pixel location is declared an edge location if the value of the gradient exceeds some threshold. Edges have higher pixel intensity values than those surrounding it. So once a threshold is set, comparison of the gradient value to the threshold value is done and the edge is detected whenever the threshold is exceeded.

From the experience of the tested image in A. Alshennawy et. al [29], it is found that the best achieved result for edge detection threshold is 80; i.e. the range zero to 80 is indicated as black and from 80 to 255 is indicated as white.

To obtain binary image from the grayscale image; first image is convolved with sobel filter, then a threshold is applied to the pixel values; if the pixel value is greater than 80, pixel is denoted as 1, else denoted as 0.

4.2.3. Plate Localization

Light background with dark changes provides the motive for detecting the horizontal changes of the intensity, since the rows that contain the number plate are expected to exhibit many sharp variations. Accordingly, the algorithm first determines the extent of intensity variation for each row, while in the second step it selects the adjacent rows which exhibit the biggest changes. Number plates are highly likely to be in these rows. The horizontal position of the number plate must also be determined, which is accomplished by using the previously determined values that characterize the changes. The variations are highest at the letters (black letters on white background); therefore this is where the rate of change within a row is expected to be the highest [9].

The row position of the number plate must be found in the second step by using a picture obtained by edge detection. The algorithm searches the intensity changes on binary image for each row. Having summed up the results of filtering for each row (sum of filtered rows) the row position of the number plate is determined on the basis of the statistical properties of the individual rows. To provide a fast algorithm, simply the row featuring the highest amplitude is selected. For example, for Turkish standard numbered plate; there are seven or eight characters on the plate and one character has at least two intensity variations; as a total one numbered plate has at least 14-16 intensity change. It shows that the number plate is most likely to be located on the rows having biggest intensity changes.

Following this, the upper and lower boundaries of the number plate are approached by searching the maximal intensity value of the rows. To this end, the following procedure is applied: first the rows featuring one half of the maximal value are searched in the image, and then adjacent rows which comply with the search criteria are extracted from the image. The position of the numbered plate is aligned between these rows.

The horizontal position of the number plate is found in the third step of the algorithm by applying a procedure considering the specific property of numbered plates. The numbered plates have constant proportion considering height and width. Usually, the width is about 51 cm and the height is about 11 cm. Investigation of the boundary ratios relies on the fact that the ratio of the horizontal and vertical sizes of a number plate greater than 3 and smaller than 5. This step of the algorithm begins with the searching the edge change on the found row area of the image both from left side and the right side. If edge change occurs, then the algorithm takes the width / height ratio and compares the value with 3 and 5. If the ratio is between these, then the plate horizontal location is found. If the found ratio does not fulfill this criterion, the search process must be continued in another place. In the case of area evaluation, those regions are eliminated that are too small to process or are too big, even if they fulfill the boundary ratio requirement.

4.3. Image Enhancement

In image processing, several algorithms belong to a category called windowing operators. Windowing operators use a window, or neighborhood of pixels, to process their output. For example, windowing operator may perform an operation like finding the average of all pixels in the neighborhood of a pixel. The pixel around which the window is found is called the *origin*.

This part of the work is based on the exploitation of image processing algorithms which employ pixel windows to process their output. Although a pixel window may be of any size and shape, a square 3x3 size was chosen for this application because it is large enough to work properly and small enough to implement efficiently on hardware.

The rank order filter is a particularly common algorithm in image processing systems. It is a nonlinear filter, so while it is easy to develop, understanding its properties is more difficult. It offers several useful effects, such as smoothing and noise removal. The median filter, which is a rank order filter, is especially useful in noise removal [12].

The rank order filter works by analyzing a neighborhood of pixels around an origin pixel, for every valid pixel in an image. Often, a 3x3 area, or window, of pixels is used to compute the output. For every pixel in an image, the window of neighboring pixels is found. Then the pixel values are sorted in ascending, or rank, order. Next, the pixel in the output image corresponding to the origin pixel in the input image is replaced with the value specified by the filter order.

Since the rank order filter uses no arithmetic, an efficient mathematical representation is difficult. Figure 29 indicates the rank order filter process.

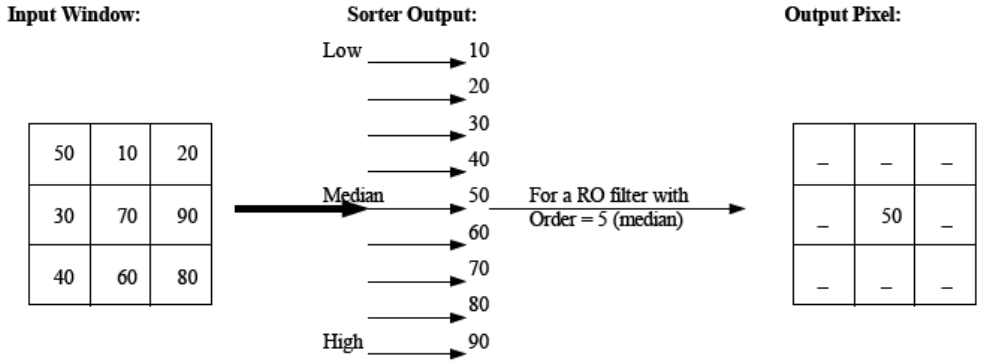


Figure 29. Rank Order Filter Process [22]

A rank order filter using a 3x3 window has 9 possible orders and a rank order filter, using a 5x5 window has 25 possible orders. No matter what the window size used in a particular rank order filter, using the middle value in the sorted list will always result in a median filter.

In order to implement a moving window system in VHDL, a design was devised that takes advantage of certain features of FPGAs. FPGAs generally handle flip -flops quite easily, but instantiation of memory on chip is more difficult. It is determined that the output of the architecture should be vectors for pixels in the window, along with a data valid signal, which is used to inform an algorithm using the window generation unit as to when the data is ready for processing. Since it was deemed necessary to achieve maximum performance in a relatively small space, FIFO units

specific to the target FPGA were used. A 3x3 window size was chosen because it was small enough to be easily fit onto the target FPGAs, and is considered large enough to be effective for most commonly used image sizes. With larger window sizes, more FIFOs and flip-flops must be used, which increases the FPGA resources used significantly.

Since its operation is fairly simple, it is an ideal choice. As discussed above, the rank order filter must first sort the pixel values in a window in ascending (or rank) order. The most efficient method accomplishing this is with a system of hardware compare/sort units, which allow for sorting a window of nine pixels into an ordered list for use in the rank order filter. This system results in a sorted list after a latency of 14 clock cycles. Since the design is pipelined, after the initial latency the system produces a valid sorted list on every clock cycle.

After the sorted list is generated with the VHDL entity, the algorithm describing the rank order filter functionality can operate on the list to produce its output. As is discussed above, the rank order filter outputs a pixel value in the origin location as specified by the rank of the filter. 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 placed 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 plate regions in the images are composed of two-dimensional data, two counters were needed: one to count rows and one to count columns in the image.

CHAPTER 5

IMPLEMANTATION AND RESULTS

The algorithms described in this thesis work is first implemented in MATLAB, after having real and efficient results, they are transferred to FPGA implementation and realized on Xilinx Virtex-2 Pro FPGA.

5.1. Plate Localization on MATLAB

To obtain efficient results and make sure the algorithm provides the expected results; numbered plate localization part of the work is first implemented in MATLAB.

MATLAB provides some ready to use functions for reading the image, edge detection, gray scale conversion, and maximum and minimum value search. The following paragraphs explain the plate localization algorithm written in MATLAB, and indicate the results.

First, the image is read from the directory with *imread* command; MATLAB stores the pixel values by 480 x 640 x 3 uint8 matrix. For the MATLAB implementation the resolution of the used images are 480 x 640. Since the algorithm depends on the intensity change in the image, binary image is needed. Binary image is generated by converting image to the gray scale first and then applying edge detection function. As a second step, with *rgb2gray* function; a new matrix 480 x 640 x 1 uint8 size is generated by taking the average of Red, Green, Blue pixel values. Then, for sobel edge detection *edge (image, sobel)* command is used. An intensity_change matrix is formed with size 480 x 1 by using the edge image. This matrix indicates the number of intensity change in a row; that is to say, the value of matrix for each row is increased when the side by side three pixel values are 0, 1, 0 or 1, 0, 1. For the next step, maximum value and the maximum index of the intensity_change matrix is found by *max* command. The maximum value is divided by two and taking

maximum index as the middle, 81 row of matrix is compared with the half of maximum value. The adjacent rows greater than half value are the candidates of plate located rows. The height of the plate is calculated as approximately five times the difference of maximum row and minimum row. For the column location of numbered plate, this proportion is checked. While minimum and maximum column location greater than the proportion, the algorithm searches on the rows 1, 0 ,1 sequence both from right and left side. The MATLAB code for the number plate localization algorithm can be found in Appendix-A. The following figures are taken from MATLAB report. Figure 30 shows the original image and the gray scale image.

Figure 30. Original Image (left hand side) and Gray Scale Image (right hand side)

Figure 31 shows the sobel edge detected binary image and the intensity change of the rows. The sobel edge detection threshold is used as MATLAB default.

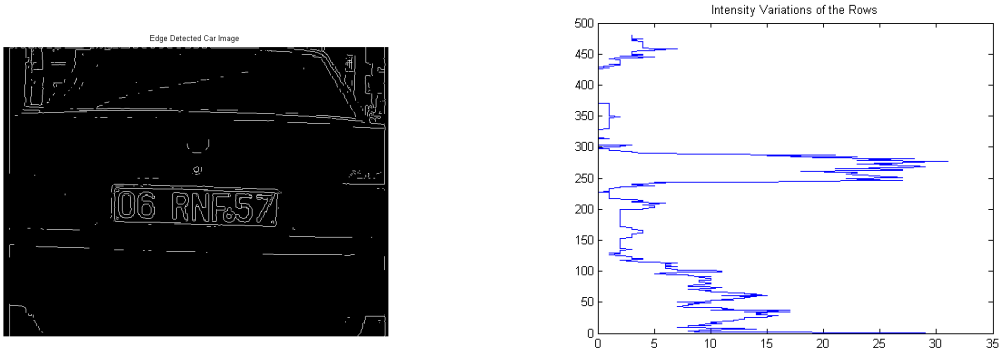


Figure 31. Sobel Edge Detected Image (left hand side) and Intensity Variation Table (right hand side)

Figure 32 shows the result of calculation of the rows which have greater value than half of maximum value of intensity variation and the plate location on the image. The row and column location of the numbered plate is found.

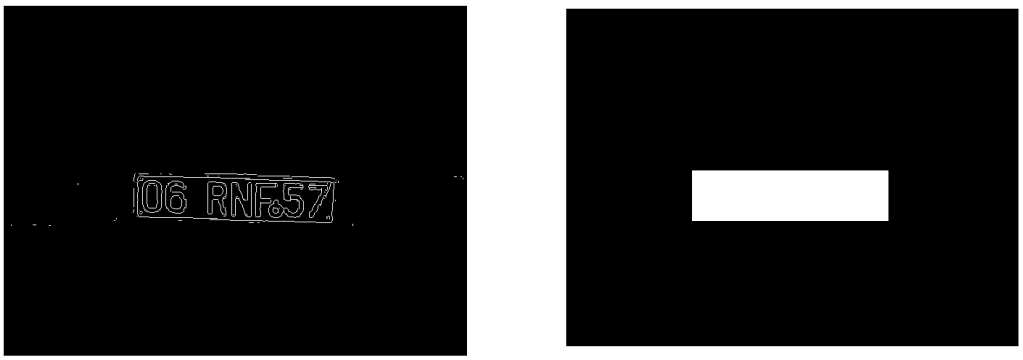


Figure 32. Plate Located Rows of Binary Image (left hand side) and Plate Location on the Image (right hand side)

Finally, the result of the whole algorithm is shown in Figure 33.

Figure 33. Result in MATLAB

Figure 34, Figure 35, Figure 36, Figure 37 shows the same process for different source image.

Figure 34. Original Image (left hand side) and Gray Scale Image (right hand side)

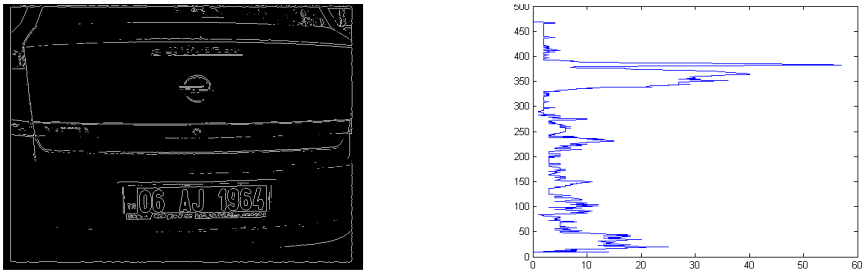


Figure 35. Sobel Edge Image (left hand side) and Intensity Variation Table (right hand side)

Figure 36. Plate Located Rows (left hand side) and Plate Location (right hand side)

Figure 37. Example of Implementation in MATLAB

5.2. FPGA Implementation

The design and simulation platform of this thesis is Xilinx ISE 9.2i Design Suit [17]. The design language is VHDL. The project properties can be seen in Figure 38.

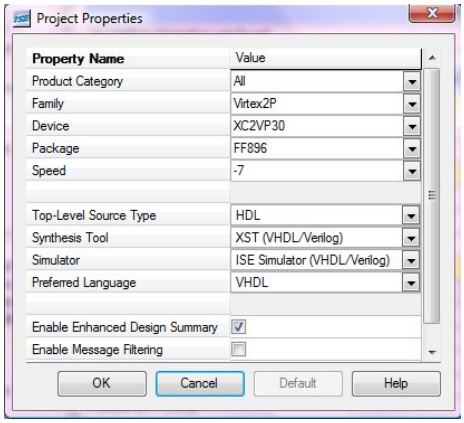


Figure 38. Project Properties

After the completion of writing VHDL codes, design is synthesized. Syntax check is done in this phase. If synthesizing is done successfully, design implementation is performed, including translating, mapping, place and routing. As a final step, a programming file is generated. In order to install generated programming file, a variety of options are possible; PROM, FPGA itself or flash. Since the programming time is less than the other options, FPGA is chosen. To download the bit file to the FPGA, Xilinx IMPACT is used, which is embedded in ISE 9.2i.

5.2.1. Project Modules

This subsection describes the written VHDL modules and their functionality.

5.2.1.1. ADV7183 Programming Module

This module provides I^2C programming, considering ADV7183 video decoder as slave and FPGA as master. The entity is defined as in Figure 39:

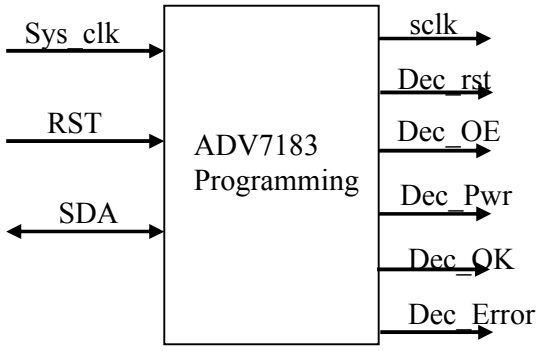


Figure 39. ADV 7183 Programming Entity

Syt_CLK stands for 100 MHz system clock, rst is active high reset input signal, sclk and sda is for I^2C communication; sda is bidirectional since video decoder sends acknowledge signal after receiving 8 bit data. Dec_rst, decod_oe, dec_pwr are the reset, output enable and power down signals for the video decoder. Dec_ok indicates the video decoder is programmed as desired, this signal triggers the video capture block; i.e. video capture block waits the dec_ok signal to begin processing coming data. Dec_error indicates the problem in programming decoder, if an error occurs; a LED on the board is activated.

5.2.1.2. Video Capture Module

This module provides analog video processing functions. The entity is defined as in 0:

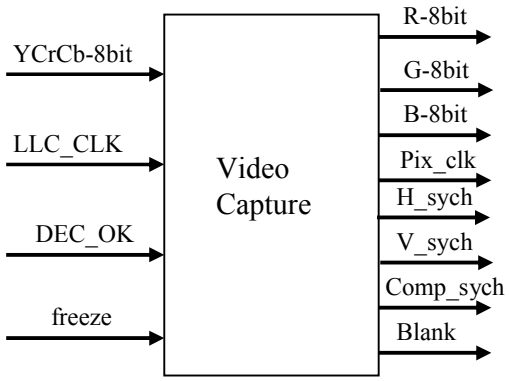


Figure 40. Video Capture Entity

Video capture is composed of different modules explained in the following subsections.

5.2.1.2.1. Decoding Module

This module provides ITU.BT 656 YCrCb and synchronization signals decoding. The entity is defined as in Figure 41:

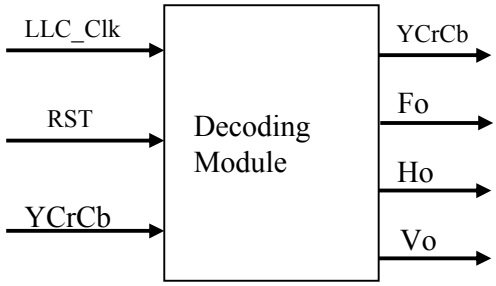


Figure 41. Decoding Entity

Field (Fo), Horizontal Synchronization (Ho) and Vertical Synchronization signals are obtained from the SAV and EAV data in YCrCb_in input.

5.2.1.2.2. 4:2:2 YCrCb to 4:4:4 YCrCb Conversion Module

This module provides data conversion from 4:2:2 to 4:4:4 format by repeating chrominance information for two pixels. The entity is defined as in 0:

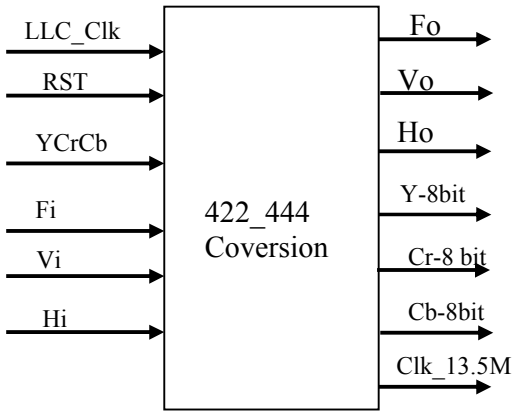


Figure 42. 4:2:2 to 4:4:4 Conversion Entity

To convert data, first four data is waited and 24 bit output data is generated by repeating the chrominance values. For the input data rate; clock is 27 Mhz, however repeating data decreases the rate half of the input. This module provides 13.5 Mhz clock and Y Cr Cb data at this rate.

5.2.1.2.3. YCrCb to RGB Color Space Conversion Module

This module provides color space conversion from YCrCb to RGB format. The entity is defined as in Figure 43 :

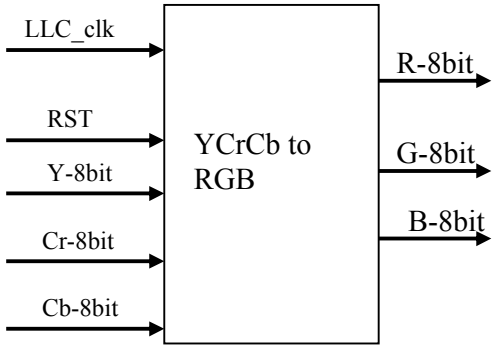


Figure 43. YCrCb to RGB Conversion Entity

Clock input for this module is 13.5 Mhz from the previous module.

While implementing this module, some problems occur; such as defining decimal numbers or negative number multiplication. The results of these problems are black shadows on the bright sides of the image. This is solved by using the VHDL code given in Appendix-B.

5.2.1.2.4. Line Buffer Module

This design has two line buffers. While one is at writing mode, the other is at reading mode. Write clock is 13.5 Mhz, and reading clock is 27 Mhz. De-interlacing is performed by reading one line twice. Only odd fields are stored in the line buffers. The entity is defined as in 0:

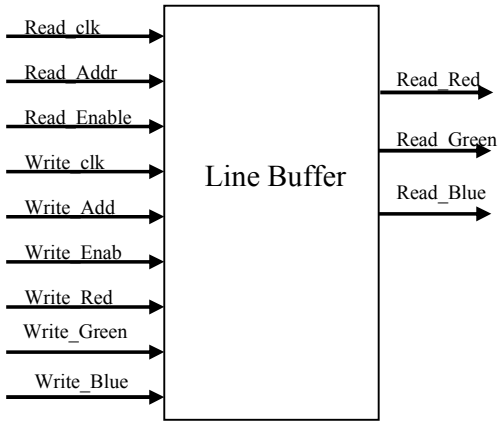


Figure 44. Line Buffer Entity

For one line buffer, three (for Red, Green and Blue data) RAMB16_S9_S9 ready to use block ram component of XILINX is used.

5.2.1.2.5. Field Buffer Module

Two single port 720 x 123 x 4 bit RAM is used from the IP generator of Xilinx for the field buffer. The entity is defined as in Figure 45:

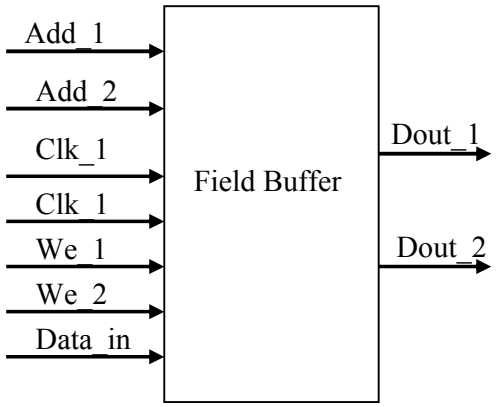


Figure 45. Field Buffer Entity

For the field buffers, as in line buffers, write clock is 13.5 Mhz and read clock is 27 Mhz.

5.2.1.2.6. Negative Edge Detection Module

This module provides the reset signal for the SVGA timing adjustment block. It generates the output when the field bit from decoder is going high to low. This means, the input video is odd field. Only odd fields are used for display purpose. The entity is defined as in Figure 46:

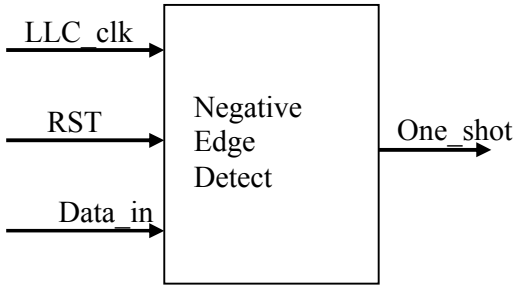


Figure 46. Negative Edge Detection Entity

5.2.1.2.7. SVGA Timing Generation Module

This module provides the reset signal for the SVGA timing adjustment block. It generates the outputs necessary for the video encoder. Also, it counts the pixels on a row, which gives the read address for both the line buffer and field buffer. The entity is defined as in 0:

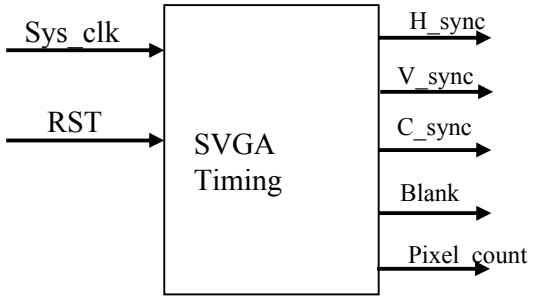


Figure 47. SVGA Timing Generator Entity

There are two counters defined in this block; one is for counting the pixel number, one is for counting the line number. Although active video area is 720 x 486; 858 x 525 counters are used to count the blanking areas.

5.2.1.2.8. Delay Module

This module provides four clock delay for timing synchronization signals. This delay is necessary since the 4:2:2 to 4:4:4 block provides Y Cr Cb data with four clock delay. The entity is defined as in Figure 48 :

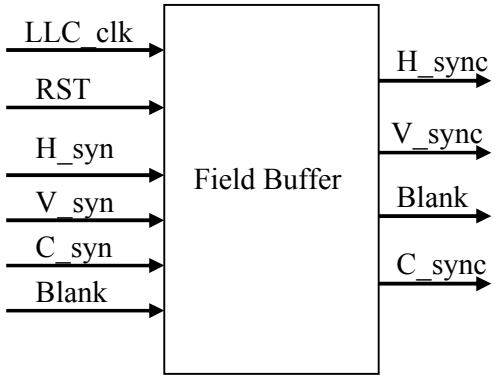


Figure 48. Delay Module Entity

5.2.1.3. Sobel Edge Detection Module

This module starts processing after 3 line input stored in the field buffer. It uses 4 MSB of Y data coming from 4:4:4 block and address_in is the same address as field buffer. It provides the address (pixel and row information) and the binary data of one pixel. The entity is defined as in Figure 49:

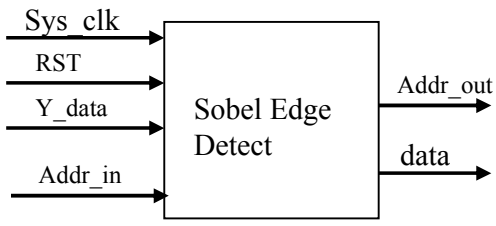


Figure 49. Sobel Edge Detection Entity

This module first applies the 3 x 3 sobel filter to the whole image both horizontal and vertical direction; then calculates the gradient. This process smoothes the image then with threshold value 80, binarization process is completed. Binarization means comparison of each pixel with 80; if greater than 80, the pixel value is set 1; else 0.

5.2.1.4. Number Plate Localization Module

This module provides the same algorithm explained in Section 5.1. The entity is defined as in Figure 50 :

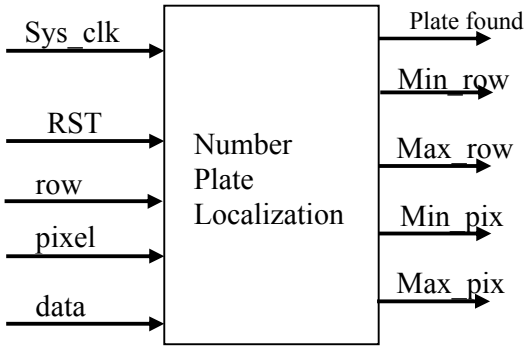


Figure 50. Number Plate Localization Entity

5.2.1.5. Image Enhancement Module

Image enhancement is performed during sobel filtering operation and defining the threshold. Moreover, to increase readability rank order filter is also applied to the image too. Image enhancement process is performed while number plate localization algorithm is running, no extra time delay occurs for this process. The entity is defined as in Figure 51:

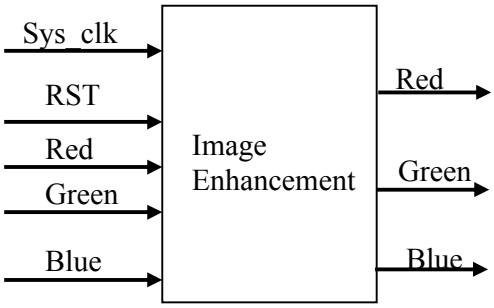


Figure 51. Image Enhancement Entity

5.2.2. Project Design Summary

The following Figure 52 indicates the design summary of the design in XILINX ISE Design Suit. Design occupies %69 of the total FPGA area.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	396	27,392	1%	
Number of 4 input LUTs	1,341	27,392	4%	
Logic Distribution				
Number of occupied Slices	1,007	13,696	7%	
Number of Slices containing only related logic	1,007	1,007	100%	
Number of Slices containing unrelated logic	0	1,007	0%	
Total Number of 4 input LUTs	1,752	27,392	6%	
Number used as logic	1,341			
Number used as a route-thru	384			
Number used as Shift registers	27			
Number of bonded IOBs	52	556	9%	
IOB Flip Flops	9			
IOB Dual-Data Rate Flops	1			
Number of PPC405s	0	2	0%	
Number of Block RAMs	25	136	18%	
Number of MULT18X18s	4	136	2%	
Number of GCLKs	4	16	25%	
Number of GTs	0	8	0%	
Number of GT10s	0	0	0%	
Total equivalent gate count for design	1,671,765			
Additional JTAG gate count for IOBs	2,496			

Figure 52. Design Summary

5.2.3. Results

The following figures show the results obtained from FPGA development board.

Figure 53 shows the original image obtained by a camera.

Figure 53. Original Source Image

Figure 54 shows the image on the screen after edge detection algorithm is applied in FPGA.

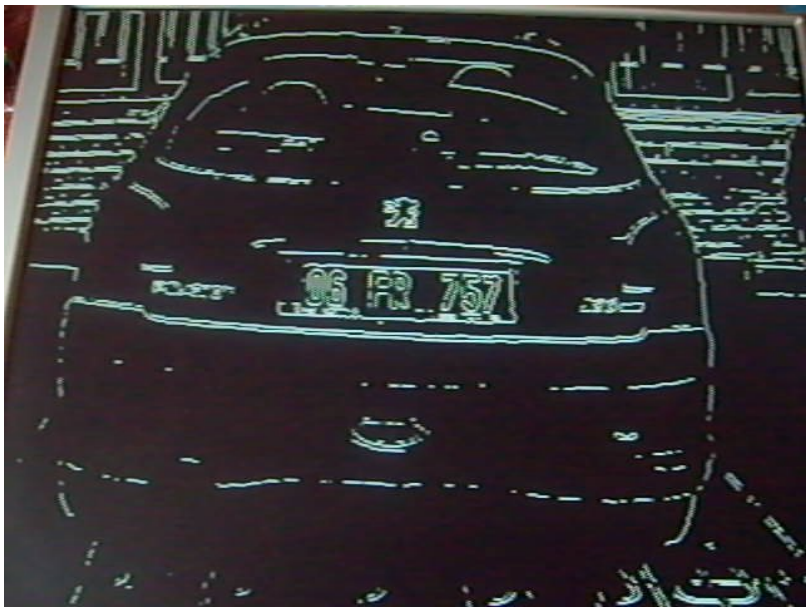


Figure 54. Edge Detected Image

Figure 55 shows the FPGA plate localization algorithm result on binary image.

Figure 55. Results Obtained FPGA Output

Figure 56 shows another source image obtained by camera.

Figure 56. Original Source Image

Figure 57 shows the edge detected image.

Figure 57. Edge Detected Image

Figure 58 shows the plate localization algorithm result designed in FPGA.

Figure 58. Results Obtained FPGA Output

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1. Conclusions

In this thesis, video processing on FPGA was studied. Three main steps of the work, i.e. design, simulation and implementation, are accomplished. For design and simulation, VHDL design language is used due to its closer structure to software. The main distinctive feature of the plate, i.e. high contrast variation, is used to detect the location of the plate on the image. Then, noise removal and smoothing methods are used to enhance the visual quality of the plate region on the image. The applied algorithms are tested with static images and video streams both in MATLAB and FPGA. In order to provide more accurate and realistic results, images taken in different time intervals and different weather conditions are tested.

The design presented here is quite capable, and it tries to take advantage of the parallelism possible with FPGA devices. A great deal of knowledge was gained from the completion of this project. While FPGAs are excellent for some uses, such as large number of image processing applications, difficulties in using more complex mathematics, speak volumes towards the argument of using dedicated DSP chips for some applications. Indeed, it is expected that a designer who desires the best combination of speed and flexibility should look toward a system consisting of both FPGAs and DSPs. Such a system can take advantage of the positive aspects of each architecture, and can allow the designer to create an algorithm on a system that is best suited for it. But it should also be noted that this work's algorithms were excellent choices for FPGA implementation. This is because they do not include complex mathematics.

For image processing algorithms, digital filters are used. First image processing part is plate localization. For plate localization, horizontal changes of the intensity, since the rows that contain the number plate are expected to exhibit many sharp variations,

is calculated and summed. The sums that have maximum values are considered as plate areas. To find the high contrast areas, edges on the images, a band pass filter with finite response is used. The reason for using finite response filter instead of infinite is the fast execution time.

After finding the plates on the video stream, they are buffered. As an image enhancement method, rank order median filter is used on all three of the plate regions. Then, the processed regions are evaluated considering lowest noise and finally displayed on the screen within the whole image.

6.2. Future Work

In order to improve the performance of whole process several improvements can be implemented.

First beside FPGA, a DSP can be used to improve the process time. DSP and FPGA can work together. While the processor is performing the complex computations, FPGA runs the image processing algorithms. Also, memory limitation of the FPGA can be avoided by adding on board DDR SDRAMs. That way, both odd field buffer and even field buffer can be implemented and calculation time for the plate localization decreased the half of the time in this work.

The previous paragraphs referred to the quality of the system. The further development assumes also adding new functionality, such as enhancement of the whole image or extracting plate characters from the image.

REFERENCES

[1] Keith Jack, “Video Demystified”, Elsevier Press, Maryland Heights, 2004

[2] S. Keller, F. Lauze, M. Nielsen, “A Total Variation Motion Adaptive Deinterlacing Scheme”, Lecture Notes in Computer Science, 2005

[3] A. Skarabot, G. Ramponi, L. Buriola, "FPGA Processor for a Videowall Image Processor", SPIE Intern. Symp.

[4] “Converting Between YUV and RGB” [On-Line] Available: http://msdn2.microsoft.com/en-us/library/ms893078.aspx, last accessed date: 16/11/2009

[5] C. Arth, F. Limberger, H. Bischof, “Real Time Plate Recognition on an Embedded DSP Platform”, in Proc. IEEE Conf. CVPR, Jun., 2007, pp. 1–8

[6] L. Pillahi, “Color Space Converter- XILINX Application Note 283”, v1.3.1, 24-03-2005

[7] G. Cao, J. Chen, and J. Jiang, “An Adaptive Approach to Vehicle License Plate Localization”, 2003

[8] C. Chou, S. Mohanakrishnan, J. Evans, “FPGA Implementation of Digital Filters”, Proc. ICSPAT, 1993

[9] B. Enyedi, L. Konyha, and K. Fazekas, “Real Time Number Plate Localization Algorithms”, Journal of ELECTRICAL ENGINEERING, pp. 247-258, 2006

[10] N. Zimic, J. Ficzko, M. Mraz, and J. Virant, “The Fuzzy Logic Approach to the Car Number Plate Locating Problem”, iis, 1997

[11] W. Jia, H. Zhang, and X. He, “Mean Shift for Accurate Number Plate Detection”, Third Inter-National Conference on Information Technology and Applications (ICITA'05), 2005

[12] J. Russ, “The Image Processing Handbook”, CRC Press, Boca Raton, FL, 1992

[13] Z. Hussain, “Digital Image Processing – Practical Applications of Parallel Processing Techniques”, Ellis Horwood, West Sussex, UK, 1991

[14] E. Dougherty, “An Introduction to Morphological Image Processing”, SPIE, Bellingham, WA, 1992

[15] W. Pratt, “Digital Image Processing”, Wiley, New York, NY, 1978

[16] E. Nelson, “Implementation of Image Processing Algoritms on FPGA Hardware ”, Vanderbilt Univesity, 2000

[17] Xilinx, “XILINX ISE 9.2i Design Suit and Software Manuals”, 07.10.2007

[18] Xilinx, “ML-402 Evaluation Platform User Guide” , v2.5, 24.05.2006

[19] Xilinx, “Xilinx University Program Virtex-II Pro Development System Hardware Reference Manual – UG069”, v1.1, 09.04.2008

[20] Xilinx, “Virtex-4 FPGA User Guide – UG070”, v2.6, 01.12.2008

[21] Xilinx, “Virtex-2 Pro Platform FPGA User Guide – UG012”, v4.2, 5.11.2007

[22] A. Nelson, “Implementation of Image Processing Algorithms on FPGA Hardware”, 2000

[23] Digilent, “VDEC1 Reference Manual” , doc:502.046, rev. 4.12.2005

[24] Analog Devices, “Multi format SDTV Video Decoder - ADV 7183 Datasheet”, Norwood, MA, 2005

[25] Intersil, “BT.656 Video Interface for ICs”, Application Note AN9728.2, July 2002

[26] IEEE Std 1076, “IEEE Standard VHDL Language Reference Manual”, January 2000

[27] “VGA Information – EEL 4712” [On-Line] Available: http://www.mil.ufl.edu/4712/docs/vga_figures.pdf, last accessed date: 23/08/2009

[28] K. Yeong-Taeg, “Luminance preserving color conversion for 24-bit RGB displays”, IEEE 13th International Symposium, 25-28 May 2009 pp. 271 – 275

[29] A. Alshennawy, “Edge Detection in Digital Images Using Fuzzy Logic Technique”, World Academy of Science, Engineering and Technology 51, 2009

APPENDIX-A

```
%   I : original image
%   A : grayscale image
%   BW : edge image
%   intensity_change : total edge variation of the rows

I=imread('carimage.bmp');
A=rgb2gray(I);
BW = edge(A,'sobel');

[row,col]=size(BW);
intensity_change=zeros(row,1);

for i=1:row
    for j=1:col-2
        if ((BW(i,j)==1)&& (BW(i,j+1)==0)&& (BW(i,j+2)==1)) ||
((BW(i,j)==0)&& (BW(i,j+1)==1)&& (BW(i,j+2)==0))
            intensity_change(i,1)=intensity_change(i,1)+1;
        end;
    end;
end;

[max_val,max_ind] = max(intensity_change);
max_search= (max_val /2);

row_can=zeros(81,1);
for i=(max_ind-40):(max_ind+40)
    intensity_change(i);
    if intensity_change(i)>max_search
        row_can (i-max_ind+41)=i;

    else
        row_can (i-max_ind+41)=0;
    end
end

count=0;
for i=1:81
    if row_can(i)==0
        count=count+1;
    end
end

plate_row=zeros((81-count),1);
say=1;

for i=1:81
    if row_can(i)~= 0
        plate_row(say)=row_can(i);
        say=say+1;
```

```
    end
end

max_plate_row=max(plate_row);
min_plate_row=min(plate_row);

PLATE_IM=zeros(480,640);
for i=1:480
    if (i>= min_plate_row)&& (i<=max_plate_row)

        PLATE_IM(i,:)=BW(i,:);
    else
        PLATE_IM(i,:)=0;
    end
end

height=max_plate_row-min_plate_row;
width_max=5*height;
half = uint16(max_plate_row-min_plate_row-1)/2;
half_row= (min_plate_row)+(half);


col_can_start=0;
i=50;
col_can_stop=0;
j=600;

while (col_can_start~=(i-1) || i<638)
   if (PLATE_IM (half_row,i)==0 && PLATE_IM (half_row,i+1)==1 &&
PLATE_IM (half_row,i+2)==0)
       col_can_start=i;
       break
   else
       i=i+1;
   end
end

while (col_can_stop~=(j+1) || col_can_stop>col_can_start)
    if (PLATE_IM (half_row,j)==0 && PLATE_IM (half_row,j-1)==1 &&
PLATE_IM (half_row,j-2)==0)
       col_can_stop=j;
       break
    else
       j=j-1;
    end
end

while (col_can_stop-col_can_start)>width_max
        PLATE_IM (half_row,col_can_start)=1;
        PLATE_IM (half_row,col_can_stop)=1;

        i=(col_can_start+1);
        while (col_can_start~=i || i<638)
            if (PLATE_IM (half_row,i)==0 && PLATE_IM
(half_row,i+1)==1 && PLATE_IM (half_row,i+2)==0)
                col_can_start=i;
                break
```

```
            else
                i=i+1;
            end
        end
        j=(col_can_stop-1);
        while (col_can_stop~=j || col_can_stop>col_can_start)
            if (PLATE_IM (half_row,j)==0 && PLATE_IM (half_row,j-
1)==1 && PLATE_IM (half_row,j-2)==0)
                col_can_stop=j;
                break
            else
                j=j-1;
            end
        end
        if (col_can_stop-col_can_start)<=width_max
            break
        end
    end

    PLATE_IM2=zeros(480,640);
    for i=1:480
        for j=1:640
        if (i>= min_plate_row)&& (i<=max_plate_row)
            if (j>= col_can_start)&& (j<=col_can_stop)
                PLATE_IM2(i,j)=1;
            end
        else
            PLATE_IM2(i,j)=0;
        end
        end
    end

    PLATE_IM3=uint8(PLATE_IM2);
APLATE = A.*PLATE_IM3;
```

APPENDIX-B

```
process (clk,rst)

variable X_int_v: std_logic_vector(18 downto 0) := (others =>'0');
variable A_int_v: std_logic_vector(18 downto 0) := (others =>'0');
variable B1_int_v: std_logic_vector(18 downto 0) := (others =>'0');
variable B2_int_v: std_logic_vector(18 downto 0) := (others =>'0');
variable C_int_v: std_logic_vector(18 downto 0) := (others =>'0');

begin
      if rising_edge(clk) then
            if (rst = '1') then
                  A_int <= (others =>'0');
                  B1_int <= (others =>'0');
                  B2_int <= (others =>'0');
                  C_int <= (others =>'0');
                  X_int <= (others =>'0');

            else
                  X_int_v  := unsigned(const1) * signed(Y_reg - 16);
                  X_int    <= signed(X_int_v(18 downto 8)) +
                  signed(Y_reg- 16);


                  A_int_v  := unsigned(const2) * signed(Cr_reg-
                  128);
                  A_int    <= signed(A_int_v(18 downto 8)) +
                  signed(Cr_reg- 128);

                  B1_int_v := unsigned(const3) * signed(Cr_reg-
                  128);
                  B1_int   <= (B1_int_v(18 downto 8)); --93184

                  B2_int_v := unsigned(const4) * signed(Cb_reg-
                  128);
                  B2_int   <= B2_int_v(18 downto 8); --44800

                  C_int_v  := unsigned(const5) * signed(Cb_reg-
                  128);
                  C_int    <= signed(C_int_v(18 downto 8)) +
                  signed(Cb_reg- 128) + signed(Cb_reg- 128);
            end if;
      end if;
end process;
```

```
process (clk,rst)

variable R_int_v: std_logic_vector(10 downto 0) := (others =>'0');
variable G_int_v: std_logic_vector(10 downto 0) := (others =>'0');
variable B_int_v: std_logic_vector(10 downto 0) := (others =>'0');

begin
      if rising_edge(clk) then
            if (rst = '1') then
                  R_int <= (others =>'0');
                  G_int <= (others =>'0');
                  B_int <= (others =>'0');
            else
                  R_int_v := signed(X_int) + signed(A_int);

                  if R_int_v (10) = '1' then
                        R_int <= (others =>'0');
                  elsif R_int_v >= 256 then
                        R_int <= "11111111";
                  else
                        R_int <= R_int_v(7 downto 0);
                  end if;

                  G_int_v := signed(X_int) - signed(B1_int) -
                  signed(B2_int);

                  if G_int_v(10) = '1' then
                        G_int <= (others =>'0');
                  elsif G_int_v >= 256 then
                        G_int <= "11111111";
                  else
                        G_int <= G_int_v(7 downto 0);
                  end if;

                  B_int_v := signed(X_int) + signed(C_int);

                  if B_int_v(10) = '1' then
                        B_int <= (others =>'0');
                  elsif B_int_v >= 256 then
                        B_int <= "11111111";
                  else
                        B_int <= B_int_v(7 downto 0);
                  end if;
            end if;
      end if;
end process;
```