TRAFFIC SIGN DETECTION USING FPGA

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

BY

İBRAHİM ÖZKAN

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

APRIL 2010

Approval of the thesis:

TRAFFIC SIGN DETECTION USING FPGA

submitted by **İBRAHİM ÖZKAN** 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** _____

Assoc.Prof. Dr. Mehmet Mete Bulut
Supervisor, **Electrical and Electronics Engineering** _____

Prof. Dr. Gözde Bozdağı Akar
Co-Supervisor, **Electrical and Electronics Engineering** _____

Examining Committee Members:

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

Assoc. Prof. Dr. Mehmet Mete Bulut
Electrical and Electronics Engineering Dept., METU _____

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Dept., METU _____

Assist. Prof. Dr. Çağatay Candan
Electrical and Electronics Engineering Dept., METU _____

Hakan Caner, M.Sc.
ASELSAN Inc. _____

Date: _____

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 : İBRAHİM ÖZKAN

Signature :

ABSTRACT

TRAFFIC SIGN DETECTION USING FPGA

Özkan, İbrahim

M. S., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Mehmet Mete Bulut
Co-Supervisor: Prof. Dr. Gözde Bozdağı Akar

April 2010, 91 pages

In this thesis, real time detection of traffic signs using FPGA hardware is presented. Traffic signs have distinctive color and shape properties. Therefore, color and shape based algorithms are chosen to implemented on FPGA. FPGA supports sufficient logic to implement complete systems and sub-systems.

Color information of images/frames is used to minimize the search domain of detection process. Using FPGA, real time conversion of YUV space to RGB space is performed. Furthermore, color thresholding algorithm is used to localize the sign in the image/video depending on the color.

Edges are the most important image/frame attributes that provide valuable information about the shape of the objects. Sobel edge detection algorithm is implemented on FPGA. After color segmentation, FPGA implementation of Sobel algorithm is used to find the edges of candidate traffic signs in real time. Later, radial symmetry based shape detection algorithm is used to determine circular traffic signs.

Each FPGA implemented algorithm is tested by using video sequences and static images. In addition, combined implementation of color based and shape based algorithms are tested. Joint application of color and shape based algorithms are used in order to reduce search domain and the processing time of detection process.

Designing architecture on FPGA makes traffic sign detection system portable as a final product and relatively more efficient than the computer based detection systems. The resulting hardware is suitable where cost and compactness constraints are important.

Keywords: Real Time Traffic Sign Detection, FPGA, Color Segmentation, Shape Based Detection

ÖZ

FPGA KULLANARAK TRAFİK İŞARETİ TESPİTİ

Özkan, İbrahim

Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Mehmet Mete Bulut

Ortak Tez Yöneticisi: Prof. Dr. Gözde Bozdağı Akar

Nisan 2010, 91 sayfa

Bu tezde, FPGA donanımı kullanılarak trafik işaretlerinin gerçek zamanlı tespiti gerçekleştirilmiştir. Trafik işaretleri kendine özgü renk ve şekil bilgisine sahiptir. Bu yüzden, renk ve şekil bilgisine dayalı algoritmalar, FPGA üzerinde gerçeklemek için seçilmiştir. FPGA'ler bütün trafik işaret tespit sistemini gerçeklemek için yeterli donanım mantığına sahiptir.

Resim veya video üzerindeki renk bilgisi, trafik işareti tespit sürecinin tarama alanını küçültmek için kullanılır. Bu amaçla, ilk önce FPGA kullanılarak YUV renk uzayından RGB renk uzayına çevrim gerçekleştirilmiştir. Daha sonra, renk eşikleme algoritması kullanılarak video üzerindeki trafik işaretlerinin olası yerleri bulunmuştur.

Kenarlar, cisimlerin şekilleri hakkında önemli bilgiler veren en önemli görüntü özelliklerinden biridir. Sobel kenar belirleme algoritması FPGA üzerinde gerçeklenmiştir. Video üzerinde renk bölütleme işleminden sonra, FPGA üzerinde

çalışan Sobel algoritması olası trafik işaretlerinin kenarlarını bulmak için kullanılır.. Daha sonra, radial simetri tabanlı şekil belirleme algoritması sayesinde dairesel trafik işaretleri tespit edilir.

FPGA'de gerçeklenen her algoritma video ve resimler üzerinde ayrı ayrı test edilmiştir. Ayrıca, renk ve şekil özelliklerinin beraber kullanılmasına dayalı algoritma da test edilmiştir. Renk ve şekil bilgisini beraber kullanılması olası trafik işaretlerinin bulunduğu alanı sınırlanmış, dolayısıyla işlem zamanını düşürmüştür

Mimarinin FPGA kullanılarak tasarlanması, trafik işaret tespit sistemini taşınabilir yapmıştır. Ayrıca diğer bilgisayar tabanlı işaret tespit sistemlerine göre daha etkili bir sistem olmuştur. Sonuçlanan donanım fiyatın ve kendi başına çalışabilirliğin önemli olduğu yerler için uygundur.

Anahtar Kelimeler: Gerçek Zamanlı Trafik İşareti Tespiti, FPGA, Renk Bölümleme, Şekil Temelli Tespit

To My Family

ACKNOWLEDGEMENTS

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

I also express my sincere gratitude to my co-supervisor, Prof. Dr. Gözde BOZDAĞI AKAR, for her initiative ideas and guidance that helped to construct this work.

I would like to thank ASELSAN Inc. for facilities provided for the completion of this thesis.

I would like to express my thanks especially to Hakan CANER and to my all friends for their support and fellowship.

I would also like to thank TÜBİTAK for its support on scientific and technological researches.

I would like to express my special appreciation to my family for their continuous support and encouragements.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

With the improvements in technology, making the machines fully autonomous is the most popular topic of researchers in order to minimize the human factor. Human perception abilities depend on the individual's physical and mental conditions. Therefore, these abilities can be affected by many factors such as tiredness [30]. There are outstanding applications about minimizing the human based faults such as fully autonomous vehicle systems and driving support systems.

Most of the traffic accidents occur because of driver's fault. Today, many automobile manufacturers spend million dollars to provide the safety of drivers and pedestrians. Driving Supporting Systems and autonomous vehicle system are common topics in order to decrease the accidents.

Driving Support System or autonomous vehicle system rely on vision-based recognition of surrounding area in order to make driving decisions. Studies on these areas have been focused on three main tasks: 1) road detection; 2) obstacle detection; and 3) sign recognition. Road detection and Obstacle detection have been studied for many years, with many good results, but traffic sign recognition is still an open search area [4]. Driving supporting systems can recognize traffic signs accurately; therefore, false recognizing of signs can be prevented. For autonomous vehicle systems, sign recognition can be used to guide the vehicle according to the information coming from signs.

Traffic signs give much useful information about the environment when driving. Missed signs can cause dangerous situations or even accidents [32]. Therefore automated recognition of traffic is a serious topic for driver supporting systems and autonomous vehicles.

Traffic signs have some discriminative features such as color and shape properties, which can be used for detection and recognizing of traffic signs. However, when using these features there are some challenging occasions, listed as follows;

- Lightning conditions are changeable during day. Especially in bad lighting conditions, it is harder to gather color and contour information of traffic signs [33].
- Other objects can occlude traffic signs. Because of occlusion, detection of traffic sign can be even impossible.
- Weather conditions such as rain, snow or fog, effects the detection of sign candidates.
- Surface metal of traffic signs may physically be damaged or changed.

Either driving support systems or autonomous vehicles have to be fast and robust to detect signs in real time and recognize them precisely [25]. A late detection would completely be useless [33]. In addition to these, sign detection system has to be produced at low cost. In real time, the system has to run complicated algorithms with good performance. Achieving this level of processing, powerful and expensive systems are needed. Due to its parallel architecture and small size, the FPGA platform is suitable for implementing various vision based safety systems on automotive vehicles at low cost [34].

1.1 Scope of Thesis

In this thesis, detection of circular traffic signs in static images and in video sequences is studied and implemented on FPGA.

First algorithms for traffic sign detection have been searched and most suitable ones have been selected for real time applications. After that, selected algorithms have been implemented on FPGA hardware.

The developed algorithm, for the detection of traffic signs, based on color segmentation, edge detection and shape detection. Shape detection is applied after the joint application of color segmentation and edge detection algorithms in order to explore circular signs within the frame.

Standard definition PAL video is digitized at 720x576 resolution which results in a 13,5MHz pixel rate. To detect or eliminate different features within the image, the filtering operation receives input data at a rate of over 10 mega samples per second. Coupled with new high resolution standards and multi channel environments, processing requirements can be even higher. Achieving this level of processing power using programmable DSP requires multiple processors [26]. In addition to these, the image processing algorithms has been limited to software implementation which is slower due to the limited processor speed [27]. An alternative solution is FPGA integrated circuits. Reconfigurable systems based on FPGA integrated circuits are used for fast prototyping implementations of the complete real time designs. [28]

The developed algorithms compared with the state of the art techniques found in the literature in terms of detection performance. Once the performance is guaranteed, it is implemented on FPGA using ISE software.

1.2 Thesis Outline

In Chapter 2, algorithms for detection of traffic signs are given. Basically, these algorithms use color and shape information of traffic signs. In addition to these, FPGA based applications used in this project are cited.

In Chapter 3, selected algorithms for detecting traffic signs are explained. In addition, hardware implementations of these algorithms are expressed with details.

In Chapter 4, implemented algorithms in FPGA are tested with static images and video streams obtained from different sources. Experimental results for each algorithm are given in this section.

Chapter 5, includes the conclusion and the future work.

CHAPTER 2

BACKGROUND ON TRAFFIC SIGN DETECTION

2.1 Introduction

The traffic sign detection and recognition systems have an increasing interest in the last times, due to the importance of safety for drivers, passengers and pedestrians[1]. Signs give important information however; a driver may not notice a particular sign due to distractions or lack of concentration. In this case it may be helpful to make them aware of the information that they have missed [2].

Traffic signs have discriminating features like color and shape. Therefore, these properties can be used to detect traffic signs. Features of traffic signs can be summed up as shown in Table 2.1.

Table 2.1 Features of traffic signs

Sign Type	**Possible (Border) Colors**	**Sign Shape**
Restricting & Warning	Red, Blue, Black	Triangle, Rectangle, Octagon, Circle
Information	Blue, Red	Rectangle
Highway Information	Green	Rectangle

A vision based supporting systems for vehicle, serves as a driver –aid systems in controlling the car. This kind of system has three main tasks. First one is the road detection and following, which has been studied for a long time. Second one is the detection of obstacles on the road till control systems avoid them. The last one is detection and interpretation of traffic signs to provide feedback for safe driving

Many algorithms for detection of traffic signs have been developed up to now. The summary of these studies will be given in this chapter.

2.2 Detection of Traffic Signs

In this step, the aim is to locate a road sign candidate on the frame coming from a video capturing device. Outputs of detection step are used as inputs for recognition of traffic signs.

For the detection of traffic signs, many algorithms have been developed. Most of these algorithms base on segmentation via color features of the signs [3], [4], [5], [7], [10], [11].

In addition to color features, there are many algorithms using the shape information of the traffic signs. Furthermore, there are some approaches which use a two step strategy. First, color properties in any of the color domains are used for segmentation. Then, shape features are used to make final decision about traffic signs detection.

2.2.1 Detection Using Color Feature

As shown in Table 2.1, traffic signs have discriminating colors which are red, blue, and green. Therefore, color segmentation algorithms are very popular in traffic sign detection.

The methods used in detection with color features, includes similar steps. First of all; a suitable color space such as RGB, YUV, HSB, etc. is selected and then

segmentation algorithms are used for desired color. A binary image is formed as an output of the segmentation algorithm. Succeeding that, image processing algorithms are used for that binary image to provide group of pixels with meaningful localization of traffic signs [13]. These image processing algorithms can be some filtering application for noise reduction or some morphological operation for removing irrelevant information. This process can be summarized in Figure 2.1.

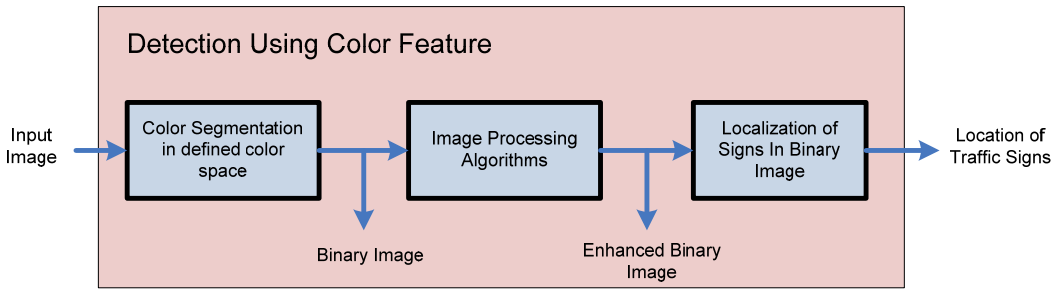

Figure 2.1 Detection of Traffic Signs using color feature

Damavandi and Mohammadi [14] suggest that using YCrCb color space gives the best practical result because of its segregated luminance component. Besides, extraction of red chrominance is less time consuming. After that, a median filter and a thinning algorithm are applied to remove irrelevant information and reveal the skeleton for the other stages of algorithm.

Escalera et al. [15], [16] propose that, HSI color space can be used for segmentation due to its robustness in brightness variation. For detection process, only hue and saturation components are considered; intensity component is not taken into account.

For hue and saturation components, look up tables are generated. Formulas given in (2.1) and (2.2) are used to generate these look up tables. Constructed look up tables can be seen in Figure 2.2.

$$H(i)=\begin{cases}255\dfrac{i_{\min}-i}{i_{\min}} & 0\le i\le i_{\min}\\ 0 & i_{\min}\le i\le i_{\max}\\ 255\dfrac{i-i_{\max}}{i_{\max}} & i_{\max}\le i\le 255\end{cases} \tag{2.1}$$

$$S(i)=\begin{cases}i & 0\le i\le i_{\min}\\ 255 & i_{\min}\le i\le 255\end{cases} \tag{2.2}$$

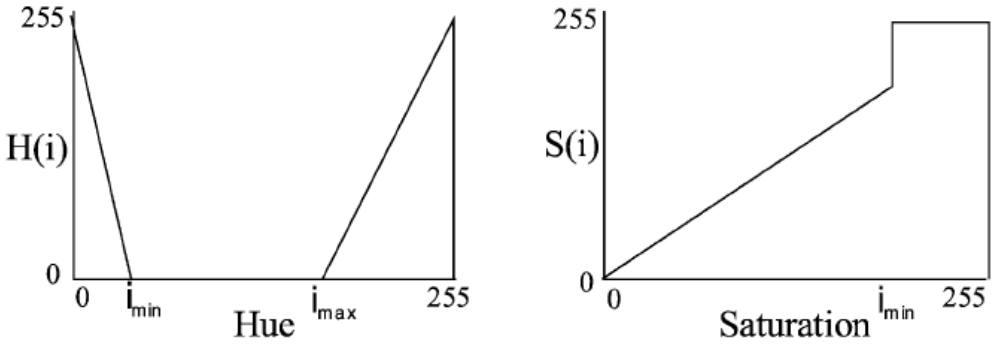

Figure 2.2 Hue, Saturation look up tables.

i values represents the original values for hue and saturation, and $H(i)$, $S(i)$ are the mapped values.

Table 2.2 Color enhancement values (normalized to 255)

	Hue Min.	**Hue Max.**	**Saturation Min.**
Red Signs	11-224	0(255)	23
Blue Signs	128-143	137	84

Ghica, Yu and Yuan [39] proposed to use vector based threshold in RGB color space. Red, Green and Blue are defined as basis vectors and pixel color *c* can be represented by:

$$c = c1 * Red + c2 * Green + c3 * Blue \tag{2.3}$$

where $0 \le c1, c2, c3 \le 1$

For a given reference color (expressed by using same basis functions) $r = (r1, r2, r3)$ the distance d between pixel color c and reference color r is as follow:

$$d = \sqrt{(r1 - c1)^2 + (r2 - c2)^2 + (r3 - c3)^2} \tag{2.4}$$

The threshold function F is given by:

$$F(c) = \begin{cases} (0,0,0) & if \quad \|r - c\| \le t \\ c & if \quad \|r - c\| > t \end{cases} \tag{2.5}$$

where t is the suitable threshold.

This detection module based on color segmentation. They use a morphological filter to remove non-sense pixels of function F.

Escalera et al.[4] proposed that hue component calculation requires evaluation of some non-linear and trigonometric equations which are expensive to evaluate for every single pixel. Therefore, they developed the RGB color threshold algorithm based on the idea of Kamada and Yoshida [17]. For red colored traffic signs, they used ratio of blue and green components to red component for defining the threshold criteria as shown below.

$$g(x,y) = k_1 \left\{ \begin{array}{l} R_a \le f_r(x,y) \le R_b \\ G_a^{'} \le \dfrac{f_g(x,y)}{f_r(x,y)} \le G_b^{'} \\ B_a^{'} \le \dfrac{f_b(x,y)}{f_r(x,y)} \le B_b^{'} \end{array} \right\} \qquad (2.6)$$

$g(x,y) = k_2$ in any other case.

where

- g(x,y) is the decision output of color threshold.
- $f_r(x,y)$, $f_g(x,y)$, $f_b(x,y)$ are the red, green, blue components of each pixel
- $R_a, R_b, G_a^{'}, G_b^{'}, B_a^{'}, B_b^{'}$ are the threshold values.

Kantawong [18] states that RGB color space is very sensitive to lightning and converting HSI color space has heavy computational cost. Hence, Kantawong proposes a modified RGB version which uses color thresholding as a contrast improvement technique.

The equations used for this technique are expressed below:

$$p(x,y) = 255 - \frac{\sum(f_r, f_g, f_b)}{3} \qquad (2.7)$$

where f_r, f_g, f_b are red green and blue values of the selected pixel.

$$p(x,y)_{avg} = \frac{\left(10 * p(x,y) + \sum_{1}^{8} g(x,y)\right)}{9} \qquad (2.8)$$

where $g(x,y)$ is the neighboring pixels of the selected pixel.

$$p(x,y)=\begin{cases} black & if \quad p(x,y)_{avg} \geq 3 * P(x,y)_{avg} \\ white & if \quad p(x,y)_{avg} < 3 * P(x,y)_{avg} \end{cases} \tag{2.9}$$

where $P(x,y)_{avg}$ is the average color value of all pixels in the image.

Janssen et al [19] states a color conversion algorithm from RGB space to remove the sensitivity of RGB color space to lighting. They used normalized color space "Irg".

$$I = \frac{f_r + f_g + f_b}{3} \qquad r = \frac{f_r}{3I} \qquad g = \frac{f_g}{3I} \tag{2.10}$$

where I corresponds to intensity, r corresponds to normalized red, g corresponds to normalized green and R,G,B correspond to red, green, blue signal of camera.

Fleyeh [30] proposed a color segmentation algorithm based on converting the RGB images into HSV color spaces. The HSV color space is used because; Hue component is invariant to the variations in light conditions. A fuzzy system is used specify the range of each pixel's color. Hue and Saturation membership function are used to define each sign color as shown in Figure 2.3 and Figure 2.4. Seven Fuzzy rules are defined to segment a certain color;

1. If (*Hue* is **Red1**) and (*Saturation* is **Red**) then (*Result* is **Red**)
2. If (*Hue* is **Red2**) and (*Saturation* is **Red**) then (*Result* is **Red**)
3. If (*Hue* is **Yellow**) and (*Saturation* is **Yellow** then (*Result* is **Yellow**)
4. If (*Hue* is **Green**) and (*Saturation* is **Green**) then (*Result* is **Green**)
5. If (*Hue* is **Blue**) and (*Saturation* is **Blue**) then (*Result* is **Blue**)
6. If (*Hue* is **Noise1**) then (*Result* is **Black**)
7. If (*Hue* is **Noise2**) then (*Result* is **Black**)

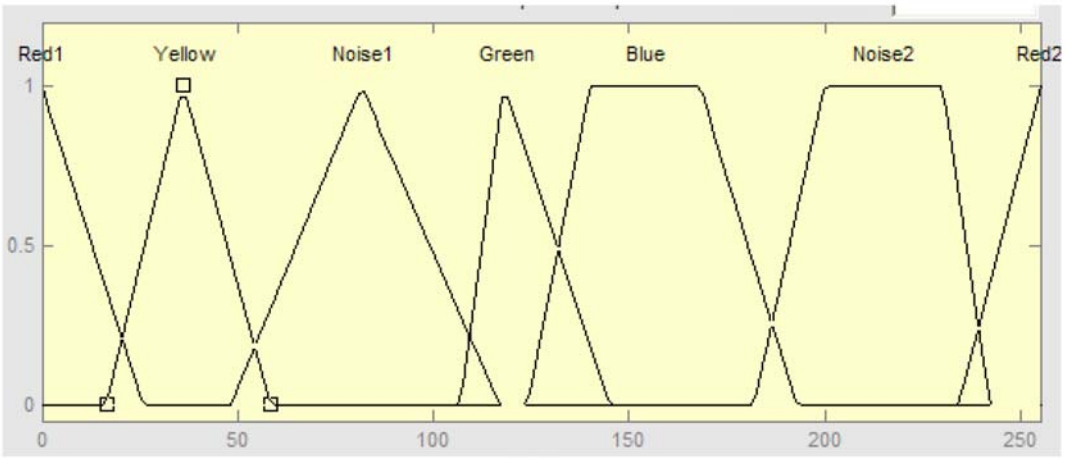

Figure 2.3 Hue Membership Function

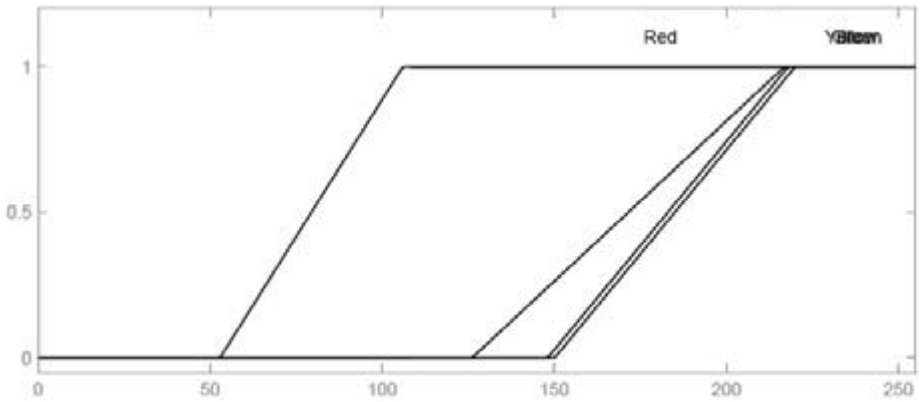

Figure 2.4 Saturation Membership Function

"Result" output variable is shown in Figure 2.5 which consists of five member functions, one for each color. They represent a certain range of gray levels in the output image which correspond the colors used in road signs [30]. Relationship between Hue, Saturation and Results membership functions is shown in Figure 2.6.

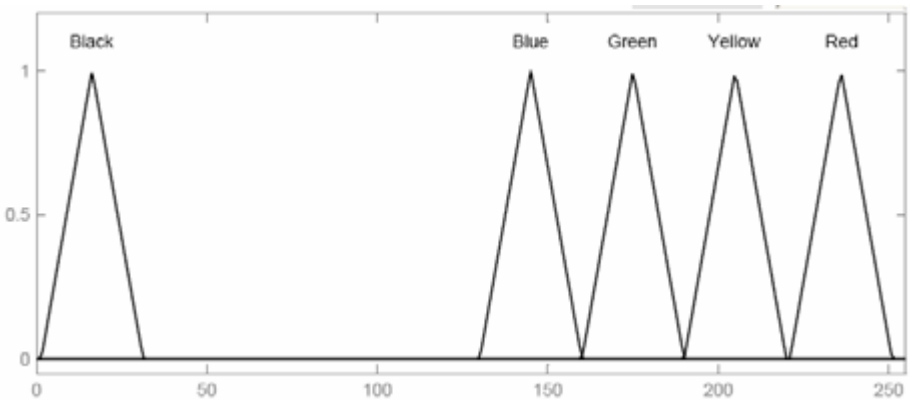

Figure 2.5 The Output Function

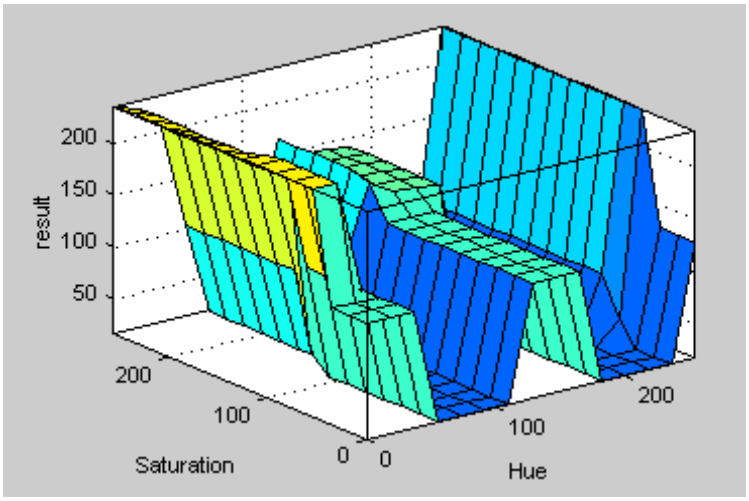

Figure 2.6 The Fuzzy System Surface

In [25] RGB color segmentation is used to detect candidate regions with two restriction rules. First rule gives good results in good lighting conditions. Normalized color information is used in second rule and gives good results in dark images. These rules are explained as follows;

1. Pixel belongs to red sign if: (For good lighting conditions)

$$(f_r(i,j) > 50) \text{ and } (f_r(i,j) - f_g(i,j) > 15) \text{ and } (f_r(i,j) - f_b(i,j) > 15) \qquad (2.11)$$

where $f_r(i,j), f_g(i,j), f_b(i,j)$ are red green and blue values of the selected pixel with coordinates (i,j).

2. This rule gives good results in bad lighting conditions.

$$
\begin{aligned}
k &= \frac{255}{\max(f_r(i,j), f_g(i,j), f_b(i,j))} \\
\tilde{f}_r(i,j) &= f_r(i,j) \cdot k \\
\tilde{f}_g(i,j) &= f_g(i,j) \cdot k \\
\tilde{f}_b(i,j) &= f_b(i,j) \cdot k
\end{aligned}
\tag{2.12}
$$

Pixel belongs to red sign if;

$$
(\tilde{f}_r(i,j) - \tilde{f}_g(i,j) > 10) \quad \textit{and} \quad (\tilde{f}_r(i,j) - \tilde{f}_b(i,j) > 10) \tag{2.13}
$$

For green and blue information signs;

$$
M_{i,j} = \max(f_g(i,j), f_b(i,j)) \tag{2.14}
$$

Pixel belongs to information sign if,

$$
M_{i,j} - f_r(i,j) > 0.4 \cdot f_r(i,j) \tag{2.15}
$$

2.2.2 Detection Using Shape Feature

In addition to color features, traffic signs have also discriminating shapes. Therefore, this characteristic of traffic signs can be used for sign detection. Shape detection algorithms are more robust to changes in illumination conditions, however flawed shaped signs or occluded with other objects make this process quite difficult. In Figure 2.7, main steps for sign detection using shape information are summarized.

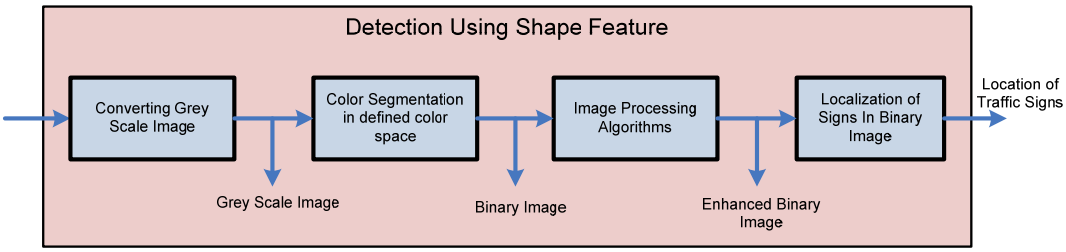

Figure 2.7 Detection of Traffic Signs using shape feature

Escalera et al. [4] state, a kind of corner detector based on optimal corner detector can be used for shape detection. The detector is applied on binary image coming from color segmentation algorithm Special mask are generated for different shapes of traffic signs. Corners can be detected from the convolution of the image with those masks. After convolution process, the regions exceeding some threshold value are accepted as corners. Finally, using the geometrical relationship between labeled corners, geometrical shapes are detected.

Garcia-Garrido, Sotelo and Martin-Gorostiza [1] use Canny method for edge detection. With the aim of making the detection more reliable, they have chosen to adapt, the two canny thresholds in a dynamic way, depending on the histogram distribution of the image. Therefore, histogram has been divided into eight regions and a pair of threshold levels has been assigned to each one of the regions. With this approach, it is possible to use the same algorithm either under good visibility conditions, in the day, or under less favorable conditions, at night, or in the rain.

Hough Transform is used by them in order to detect triangular signs. A straight line in the xy plane with a distance to the origin ρ and the angle of the normal line to this straight line passing through the origin, with the abscissa axis θ can be expressed as:

$$x \cdot \cos(\theta) + y \cdot \sin(\theta) = \rho \tag{2.16}$$

where the parameter space $p = (\rho, \theta)$ should be quantized.

They propose, using Circular Hough Transform, circular signs and stop signs can be detected. A circumference in the xy-plane with center (X, Ψ) and radius ρ can be expressed as;

$$(x - \mathrm{X})^2 + (y - \Psi)^2 - \rho^2 = 0 \tag{2.17}$$

where the parameter space, $p = (\mathrm{X}, \Psi, \rho)$ should be quantized.

The Hough Transform for lines and circular objects has heavy computational costs. To decrease the computation time, Hough transform is applied to the points of closed contours. The closed contours are determined using "chain code".

Wu et al. [20] state an algorithm very similar to Loy and Barnes's method. Algorithm based on voting of every edge for a potential center of traffic signs. They claim that their algorithm reduces the execution time 2-5 times theoretically when compared to similar algorithms.

Two different cameras for detection purpose are used by J. Miura et al. [21]. One camera –the wide camera– is used to detect traffic sign candidates by using color, intensity and shape information. After that, for each candidate, second camera –the telephoto camera– is used to capture the candidate in a larger size. Therefore, edge detection can be applied to candidates coming from the first camera. In can be said, color and shape detection are applied consecutively.

Fang et al [13] apply color and shape detection algorithm simultaneously to generate a geometry model of signs. This approach prevents formation of regions falsely rejected by color segmentation.

In [30], ellipticity, triangularity, rectangularity, and octagonality measures are defined to decide the shape of the sign. In the definition of these measures another variable I_1 is used.

$$I_1 = (\mu_{20}\mu_{02} - \mu_{11}^2) / \mu_{00}^4 \tag{2.18}$$

where μ_{20} , μ_{02} , μ_{11} are the second order central moments, and μ_{00} is the area of the object. Using I_1 ellipticity measure can be found as follows:

$$E = \begin{cases} 16\pi^2 I_1 & if \quad I_1 \le 1/(16\pi^2) \\ 1/(16\pi^2 I_1) & otherwise \end{cases} \tag{2.19}$$

E gets the values between [0, 1], and for perfect ellipse it gets 1.

Triangularity measure T is given by:

$$T = \begin{cases} 108 I_1 & if \quad I_1 \le 1/108 \\ 1/(108 I_1) & otherwise \end{cases} \tag{2.20}$$

T gets the values between [0, 1], and for perfect triangle it gets 1.

Rectangularity is measured by calculating the area of the region under consideration to the area of its minimum bounding rectangle (MBR) [31]. R1 is the rectangularity calculated for horizontally aligned objects, R2 is the rectangularity of objects oriented in any other angle and calculated as the area of its minimum bounding rectangle,

Octagonality measure O is given by:

$$O = \begin{cases} 15.932\pi^2 I_1 & if \quad I_1 \le 1/(15.932\pi^2) \\ 1/(15.932\pi^2 I_1) & otherwise \end{cases} \tag{2.21}$$

where O gets the value between [0, 1], and for perfect octagonal it gets 1.
To perform the shape classification of traffic signs, five rules can be used as follows;

1. If **(R1 is Low)** and **(R2 is Low)** and **(T is One)** and **(E is Low)** and **(O is High)** then
2. If **(R1 is One)** or **(R2 is One)** then **(Shape is Rectangle)**
3. If **(R1 is Low)** and **(R2 is Low)** and **(T is High)** and **(E is Low)** and **(O is One)** then **(Shape is Octagon)**
4. If **(R1 is Low)** and **(R2 is Low)** and **(T is High**) and **(E is One)** and **(O is Low)** then **(Shape is Circle)**
5. If **(R1 is not One)** and **(R2 is not One)** and **(T is not One)** and **(E is not One)** and **(O is not One)** then **(Shape is Undefined)**

The membership functions of the variables at the above are shown in Figure 2.8-12.

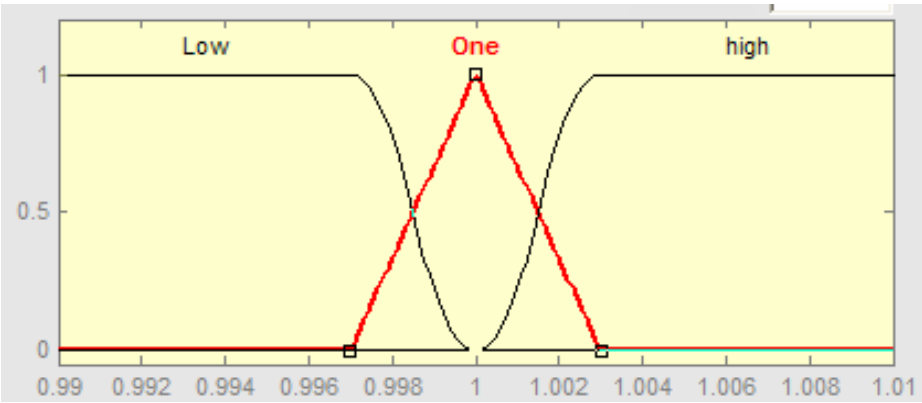

Figure 2.8 The E Membership Functions.

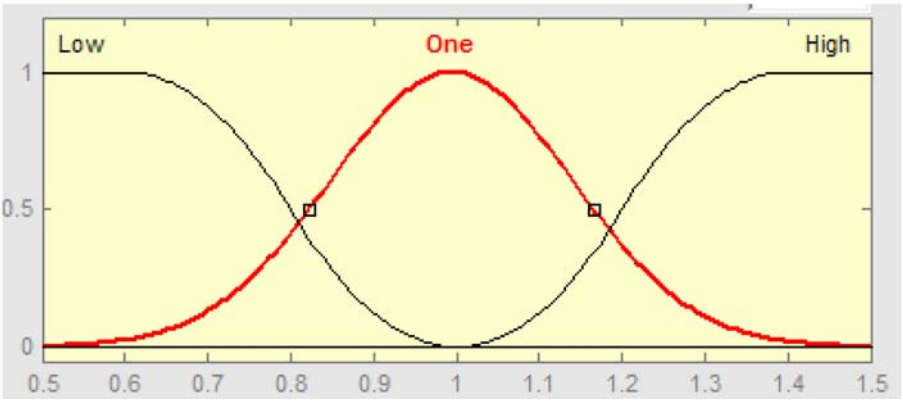

Figure 2.9 The T Membership Functions.

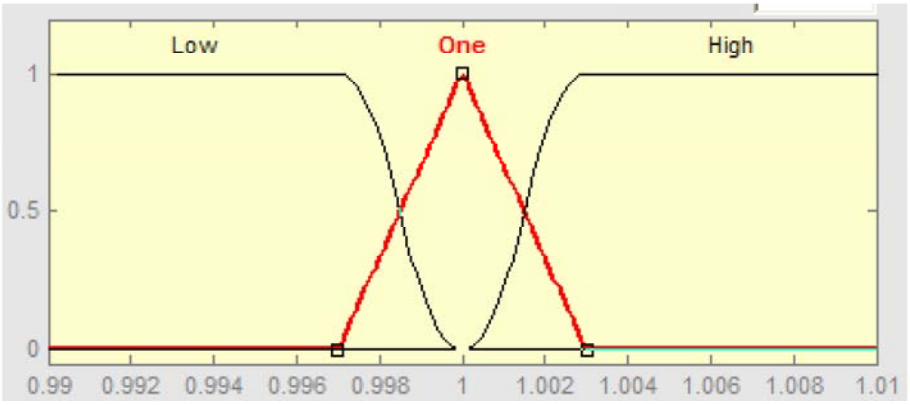

Figure 2.10 The O Membership Functions

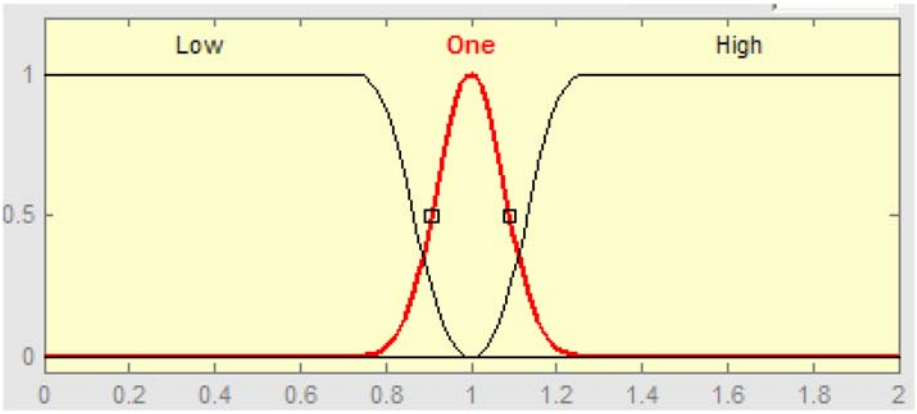

Figure 2.11 The R1 Membership Functions.

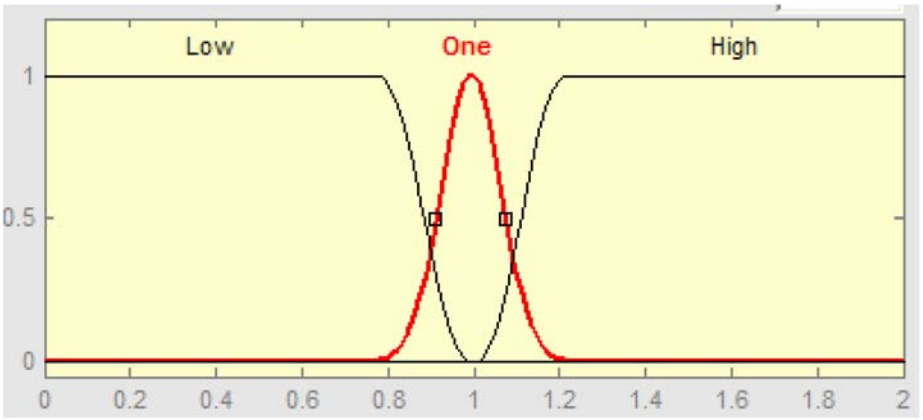

Figure 2.12 The R2 Membership Functions.

In [6], stop sign detection system using FPGA is expressed. Because of the memory requirements of the system, several lines of pixel are buffed instead of whole image. Then, angle of gradient is calculated for every pixel and quantized into one of the different angular bins. To calculate the pixel's gradient equation (2.22) can be used.

$$g = \tan^{-1}(\frac{g_y}{g_x}) \tag{2.22}$$

where g is the pixel's gradient angle; g_x, g_y are pixels gradients in the horizontal and vertical directions respectively.

Equation (2.22) is not suitable for FPGA implementation therefore, another method is used. Method bases on quantizing of the pixel's angular bin from its corresponding gradients. The angle spacing of the 4-bin is shown in Figure 2.13.

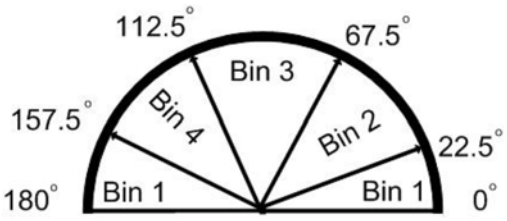

Figure 2.13 Angular spacing of 4-bin.

The next step is to calculate an entry to the Integral Map (IMap) at the corresponding pixel location [6]. Dual port RAMs in FPGA are used to store IMaps. After that, detection module based on moving window is used. At every pixel location in the image, selected HoG features within the detection window are extracted from the IMaps. Each HoG feature value is compared to its appropriate threshold. The results are then combined to determine the output [6].

In [8], FPGA implementation of canny edge detection algorithm is explained for real-time image and video processing applications. Basically, 3 modules are

generated in FPGA, which are convolution module, non-maximum suppression and hysteresis thresholding. Convolution modules are used to convolve the image/video data with Gaussian or derivative of Gaussian. During this process, line buffers and adder tree are used. The non-maximal suppression module identifies pixels that are local maxima in the direction of the gradient using the magnitude and orientation of the pixels. The hysteresis thresholding module uses two threshold values to detect edges. Embedded RAM blocks in FPGA are used to store data during thresholding operation. According to [8], FPGAs are good alternatives, which can be used to off-load computationally-intensive and repetitive functions as co-processors.

In [9], FPGA implementation of Sobel edge detection algorithm has proposed. In proposed architecture, pixel values coming from decoder are processed with blocks which compute the convolution with Sobel masks. During these processes horizontal and vertical gradient values are calculated. Magnitude of gradient is calculated using an adder. In [9], magnitude form is taken as sum of magnitudes of vertical and horizontal gradients. Square root form of magnitude is not suitable for FPGA implementation; therefore it is not used during implementation. After magnitude calculation, a comparator is used to detect edge values. This architecture is capable of operating at a clock frequency of 134.756 MHz. The edges of 512 × 512 pixel image can be found out in 1.95 ms.

CHAPTER 3

DETECTION OF TRAFFIC SIGNS USING FPGA HARDWARE

Road signs are two dimensional with discriminating colors and shapes. Therefore, most of the solutions rely heavily on these features of a road sign [22]. Offline detection of traffic signs is not very difficult problem in principle, but in real time applications time constraints make it a quite difficult problem.

Xilinx ML507 FPGA board and Video Interface Input/Output card are used for real time traffic sign detection. Information about these hardware is given at the below.

Algorithms chosen for FPGA hardware are built up in MATLAB. Details of these algorithms are explained in this section.

Algorithms are implemented in FPGA with Xilinx ISE software using VHDL. VHDL codes are synthesized and loaded to FPGA with Xilinx XST.

3.1 HARDWARE ARCHITECTURE

3.1.1 XILINX ML507 FPGA BOARD

Xilinx ML507 FPGA demo board includes Virtex-5 field programmable gate array (FPGA). Detailed information about FPGA and Xilinx ML507 FPGA Board is given at the appendix.

For processing of video signal, Video Interface Input/Output Card is used. ML507 single ended I/O provide interface with Video Interface I/O Card. Block diagram of the system can be shown in Figure 3.1.

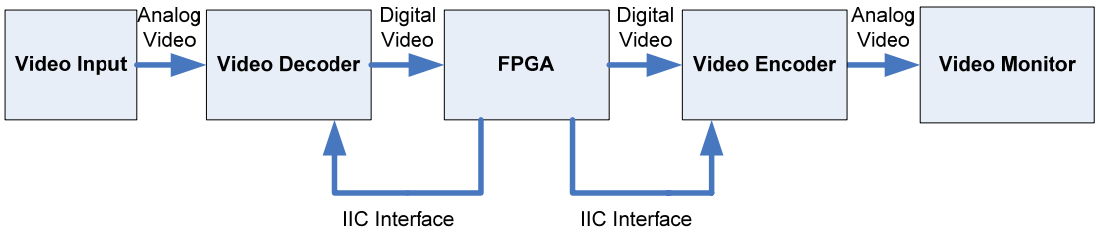

Figure 3.1 Block Diagram of the system

3.1.2 VIDEO INTERFACE INPUT/OUTPUT CARD

Video Interface Input/Output Card includes decoder and encoder integrated circuits for video signals. Decoders are used for digitizing the analog video signals to YCbCr 4:2:2 sampling format [23]. The decoder in this card can be used for composite or S-video signals in NTSC or PAL formats.

Decoder processes the incoming PAL/NTSC video signal in YCrCb 4:2:2 format and then, sent 8 bits wide digital signal to FPGA according to BT656 video standard.

BT. 656 standard defined the parallel and serial interfaces for transmitting 4:2:2 YCbCr digital video between equipment in studio and pro-video applications. Active video resolutions are either 720x486 (525/60 video systems) or 720x576 (625/50) video systems [35].

The BT.656 parallel interface uses 8 or 10 bits of multiplexed YCbCr data and a 27MHz clock. Instead of the conventional video timing signals (HSYNC, VSYNC, and BLANK), 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.

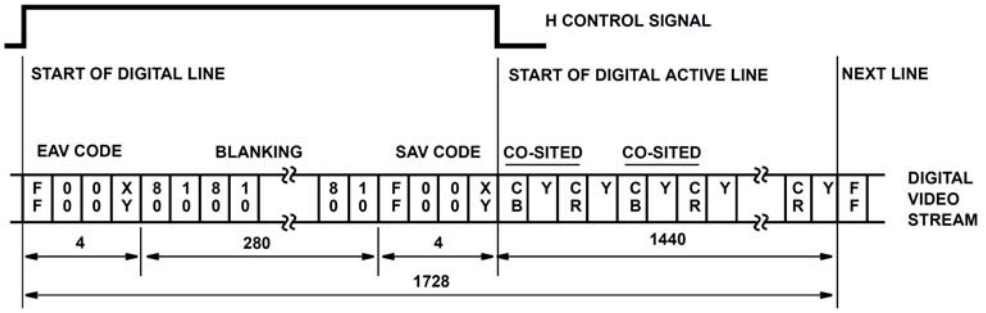

Figure 3.2 BT.656 8-bit Parallel Interface Data Format for 625/50 Video Systems

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 EAV and SAV sequences are shown in Table 3.1

Table 3.1 BT656 EAV and SAV Sequence

BIT7	BIT6 (F)	BIT5 (V)	BIT4(H)	BIT3	BIT2	BIT1	BIT0
1	F=0 First field F=1 Second field	V=1 BLANK V=1 AKTIF VIDEO	H=0 SAV H=1 EAV	Reserved bits			

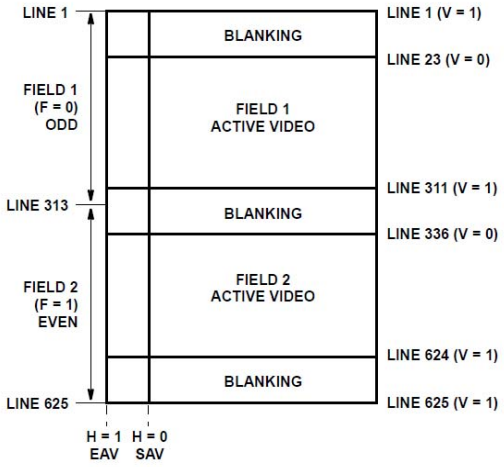

Figure 3.3 Typical BT.656 Vertical Blanking Intervals For 625/50 Video Systems

Decoder in Video Interface I/O Card needs 24,576 MHz clock signal. Therefore, an 24,576 MHz oscillator is used in Video Interface I/O Card. I2C Interface is used for communication between the FPGA and Video Interface I/O Card. The parameters used for video decoder are written to RAM in the decoder via I2C interface.

Encoder in the Video Interface I/O Card converts the digital video signal in YCbCr 4:2:2 sampling format to analog video signal in the PAL/NTSC format [24]. The parameters that video encoder is used are written to RAM in the decoder via I2C interface. Encoder is used in this thesis for monitoring the output of the algorithms. Therefore, it has no contribution to performance of algorithms running in the FPGA.

The schematic of the Video Interface I/O card can be seen at the appendix B.

3.1.3 INTER-INTEGRATED CIRCUIT (I2C) BUS

I2C (Inter Integrated Circuits) is a data bus which was developed in the early 1980's. Using the I2C bus integrated circuits can be easily connected with each other. Many integrated circuits especially video integrated circuits have this data bus. The basic information about I2C bus serial communication standard is presented below.

I2C bus connections can be seen in Figure 3.4. A unique address is used for each I2C device. Each device can be a transmitter or a receiver. During data transfer, any I2C bus device can be configured as a master or a slave device. The clock signal SCL is generated by the master device. Also, the master device addresses the other devices which are called slave devices. The I2C bus interface module designed for the video encoder and decoder supports only slave mode of operation. They only receive the messages which are sent for their addresses.

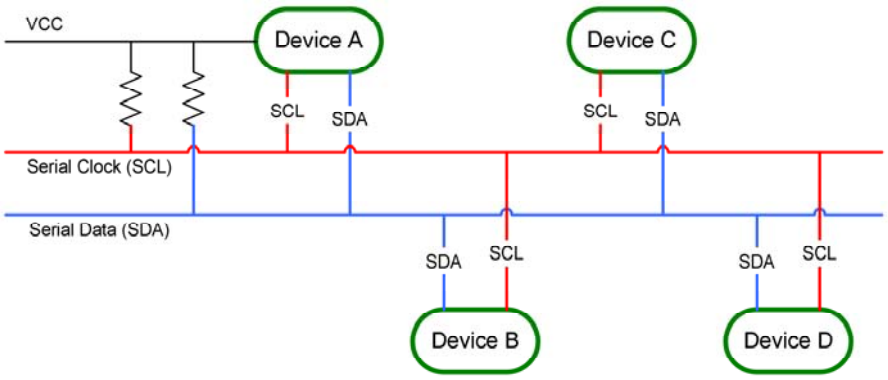

Figure 3.4 I2C Bus Connections

The serial data pin (SDA) and the serial clock pin (SCL) are used for I2C bus communication. These pins are bidirectional. Pull-up resistors are used with supply voltage for SDA and SCL pins as shown in Figure 3.4.

I2C communication is started by the master module by pulling SDA pin to low while SCL pin is high. With the beginning of communication, all the slave devices start to wait for slave address data. Slave address data consists of a 7 bits slave address and an R/W bit. The master device sends the slave address data. An acknowledgment signal is sent by the slave device which has the equal slave address data. The acknowledgment signal corresponds to pull SDA to low. The data transfer direction is determined by the R/W bit. If R/W bit is equal to 0; the master device transmits data to a slave device. If R/W bit is equal to 1; the master device receives data from a slave device. The transmitters send the most significant bit of the data first. The receiver send the acknowledge bit after each byte on the ninth SCL clock.

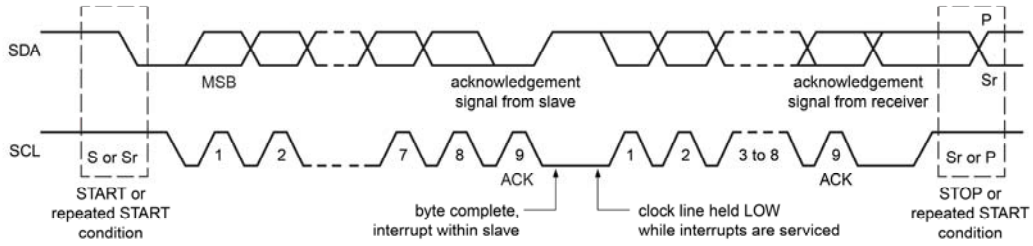

Figure 3.5 I2C Bus Serial Communication Timing

The timing diagram of I2C bus communication is shown in Figure 3.5. For transmission, one clock pulse is generated by the master device for each data bit. The master device generates START and STOP conditions. While SCL is high, a START condition reveals itself as a high to low transition and a STOP condition reveals itself as a low to high transition on the SDA line. After START signal, the bus enters the busy condition. The bus exits from the busy condition with the STOP signal.

Decoders and encoders used in Video Interface I/O Card have I2C bus. Using I2C, configuration parameters can be loaded to encoder/decoder's RAM.

I2C data bus doesn't work properly at high frequencies. 400 KHz and lower frequencies are appropriate for I2C data bus. Therefore, using the 24,576 MHz clock, a clock signal for I2C which is 384 KHz and a data signal which is 192 KHz are generated in FPGA.

Video decoder has two input ports which can be configured for PAL/NTSC video formats. To configure the inputs for PAL/NTSC video format, parameters are loaded to the decoder's RAM via I2C bus at startup. In addition to these, the coefficients used for converting analog video to digital video signal, are loaded to RAM of the decoder.

Video encoder has 3 video output ports. One of these ports can be selected.

To configure the outputs for PAL/NTSC video format, at startup parameters are loaded to the decoder's RAM via I2C bus at startup. In addition to these, the coefficients used for converting analog video to digital video signal, are loaded to RAM of the decoder.

3.2 FPGA IMPLEMENTED MODULES

3.2.1 VIDEO MATRIX & OUTPUT INTERFACE

To analyze the results of algorithms used in traffic sign detection, video signal is generated. Video encoder converts this video signal to analog video signal and forwards to monitor.

3.2.2 VIDEO ANALYZER

Using video analyzer function, EAV and SAV sequences are detected from the video signal. Synchronization signals defined in BT656 standard are obtained from these sequences. These synchronization signals are, vertical and horizontal synchronization signals, PAL video active signal, odd and even field active signals.

3.2.3 TRAFFIC SIGN DETECTION MODULES

Proposed algorithms for real time traffic sign detection includes;

- Color Segmentation
- Edge Detection
- Shape Recognition

These three steps are implemented in Xilinx ML507 FPGA board. All these three steps are working for real time video signal. These steps are explained in details at the below.

3.2.3.1 COLOR SEGMENTATION

3.2.3.1.1 Theoretical Background

Red, blue and green are dominant colors which are used on traffic signs. Certain thresholds and the inter-relationship between colors can be used to set these colors to localize the traffic sign.

In order to localize traffic signs, one of the color spaces can be used. Most popular ones are RGB and HSV color spaces.

In color segmentation, HSV color domain is not chosen in [25] due to its own computational complexity. RGB color threshold algorithm suggested by Escalera [4] is used. For the red traffic signs formula (3.1) can be used.

$$g(x,y) = k_1 \left\{ \begin{array}{l} R_a \le f_r(x,y) \le R_b \\ G_a^{'} \le \dfrac{f_g(x,y)}{f_r(x,y)} \le G_b^{'} \\ B_a^{'} \le \dfrac{f_b(x,y)}{f_r(x,y)} \le B_b^{'} \end{array} \right\} \qquad (3,1)$$

$$g(x,y) = k_2 \text{ in any other case.}$$

where $R_a, R_b, G_a^{'}, G_b^{'}, B_a^{'}, B_b^{'}$ are the threshold values; g(x,y) is the decision output of color threshold and, $f_r(x,y)$, $f_g(x,y)$, $f_b(x,y)$ are the red, green, blue components of each pixel.

The YUV color space is used by the PAL, NTSC, and SECAM color video/TV standards. The YCbCr color space was developed as part of the ITU-R BT.601 during the development of a world-wide digital component video standard. YCbCr is a scaled and offset version of the YUV color space. Y is defined to have a nominal range of 16-235; Cb and Cr are defined to have a nominal range of 16-240. There are several YCbCr sampling formats, such as 4:4:4, 4:2:2, and 4:2:0.

The conversion between RGB and YCrCb formats is given by:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1{,}164 & 1{,}596 & 0 \\ 1{,}164 & -0.813 & -0{,}391 \\ 1{,}164 & 0 & 2{,}018 \end{bmatrix} \cdot \begin{bmatrix} Y-16 \\ CR-128 \\ CB-128 \end{bmatrix} \qquad (3,2)$$

For every two Y values there is only one Cb and Cr value, this means that care must be taken when converting from YCrCb to RGB. In the digital video stream shown in Figure 3.2, every pixel has Y, Cb values or Y, Cr values. In order to calculate the RGB value of the pixel, YCrCb values of two pixels are used.

3.2.3.1.2 Hardware Implementation on FPGA

The block diagram of color segmentation for red component is presented in Figure 3.6.

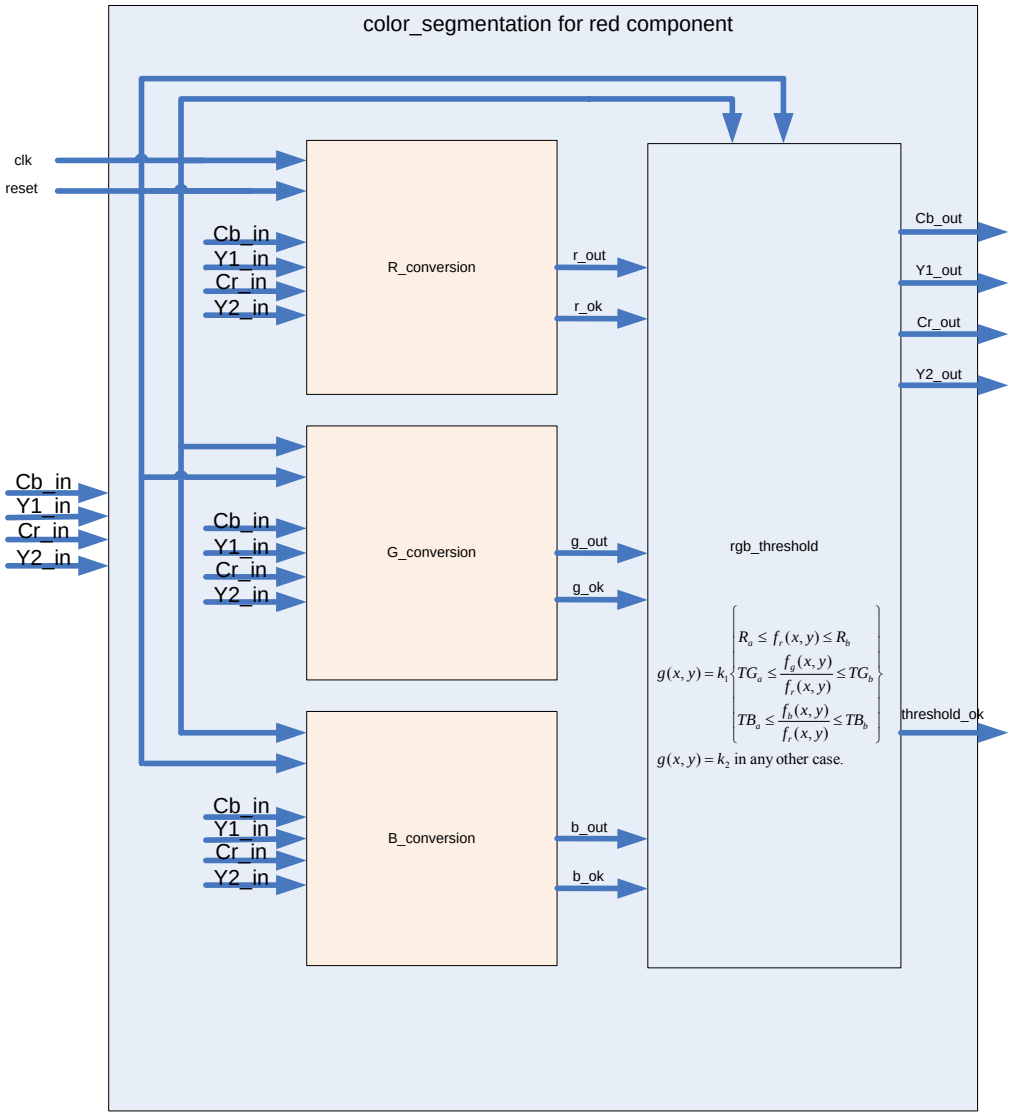

Figure 3.6 Block diagram of RGB conversion and color segmentation module

In order to make color segmentation in real time, Cb, Y, Cr components of video signal have to be controllable. The state transitions of the color segmentation are shown in Figure 3.7.

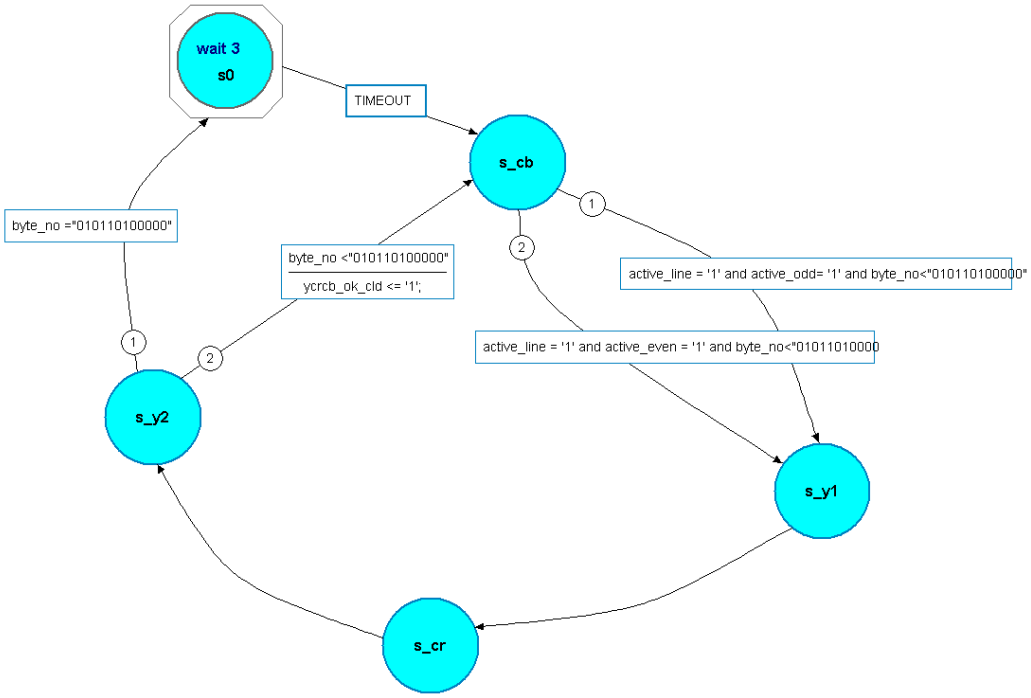

Figure 3.7 State transitions of RGB conversion module as output by HDL designer program

"Video_examiner" block in FPGA, analyzes the digital video signal coming from decoder, and extracts the synchronization signals. These signals are "active_line", "active_odd" and "active_even". The "active_line" indicates that active video signal is coming from decoder. "active_odd" and "active_even" indicate the odd and even fields of active video respectively as shown in Figure 3.3.

In Figure 3.7, arrows shows the direction of state flow. Statements on the arrows indicates the conditions and the numbers on the arrows indicates the number of conditions. The initial state is "s_cb". When the synchronization signals ("active_line", "active_odd", "active_even") come from "video_examiner" component, state changes to "s_y1". Using these states, Cb, Y, Cr components of

digital video signal can be controlled. When the state is “s_y2”, byte number coming from decoder is incremented one. If byte number reaches 720, that means end of the line has come. In that situation, 3 clock cycle wait statement occurs in order to analyze EAV code. All the arrows seen in Figure 3.7 indicate the direction of state flow.

During color segmentation, losing data affects the performance of traffic signs detection. Clock output frequency of the video decoder is 27 MHz. Therefore, clock frequency of the “rgb_conversion” module is chosen as 200 MHz in order not to lose any data. The whole process takes 4 clock cycles in 200 MHz.

The threshold values used in equation (3,1) are chosen after experimental tests. Red threshold vales for formula (3.1) are taken as:

$R_a = 75$, $R_b = 255$, $G_a^{'} = 0$, $G_b^{'} = 0{,}45$, $B_a^{'} = 0$, $G_a^{'} = 0{,}45$.

For darker images threshold values are taken as:

$R_a = 55$, $R_b = 255$, $G_a^{'} = 0$, $G_b^{'} = 0{,}65$, $B_a^{'} = 0$, $G_a^{'} = 0{,}65$.

Similarly other threshold values are generated for green and blue components.

3.2.3.2 EDGE DETECTION

3.2.3.2.1 Theoretical Background

The shape of the signs plays an important role in detection process. The edge information of the signs can be used in order to recognize the shape of the signs. In the edge detection process, boundaries of objects in the image distinguished by quick changes of intensity are searched. Most edge detection operators are based on first derivative based operation. First derivative or first difference based operators are the simplest ones that are why one of them is selected for implementation on hardware device.

Sobel Edge Detection

Significant local changes in an image can be detected using edge detection algorithms. First difference based edge detection operators such as Sobel, use the gradient information in the image. The gradient can be expressed as a measure of change in a function and, it is defined as the vector:

$$G[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \partial f / \partial x \\ \partial f / \partial y \end{bmatrix} \tag{3.3}$$

$G[f(x,y)]$ vector points in the direction of the maximum rate of increase of the function *f(x,y),* and the magnitude of the gradient, given by (3.4) equals the maximum rate of increase of *f(x,y)*per unit distance in the direction *G* [37].

$$G[f(x,y)] = \sqrt{G_x^2 + G_y^2} \tag{3.4}$$

The Sobel operator 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 edge detector uses a pair of 3x3 convolution masks, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows). A convolution mask is usually much smaller than the actual image. As a result, the mask is slid over the image, manipulating a square of pixels at a time. The actual Sobel masks are shown below:

$$Grad_X = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad , \quad Grad_Y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3.5}$$

The weights used in the horizontal and vertical neighborhood are used for noise smoothing by giving more importance to neighborhood pixels. The reason for choosing 3 × 3 neighborhood is to make the operator less sensitive to noise.

In this thesis Sobel operator is implemented in FPGA because of its low computational complexity and good edge detection capability in noisy conditions.

3.2.3.2.2 Hardware Implementation on FPGA

The FPGA implementation of the Sobel edge-detector algorithm is partitioned into three different stages as shown in Figure 3.8.

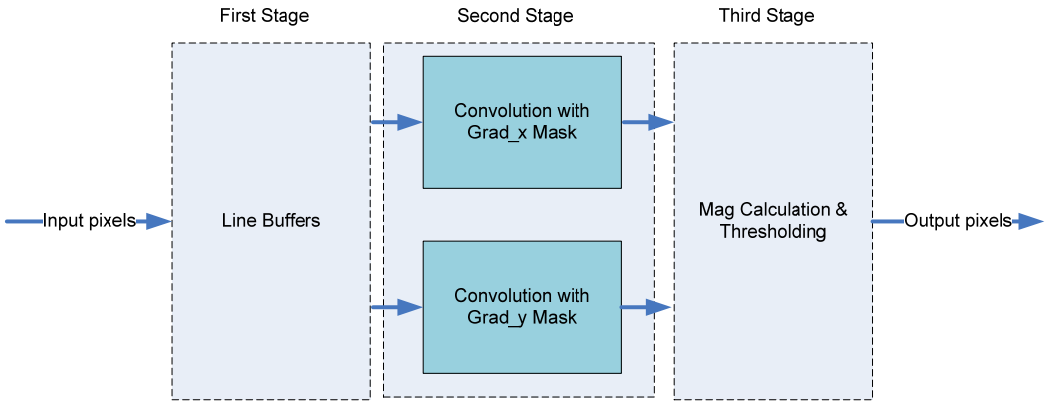

Figure 3.8 Block diagram of hardware implementation of Sobel algorithm

3.2.3.2.2.1 Line Buffering

For the hardware implementation of 2D filter for video signal, the present pixel, the previous pixel and the following one are used. Therefore; two coming lines recorded the block RAM of the FPGA. Because of the PAL video signal, input video size is 288x720 pixels. 288 represents the line numbers and 720 represents the pixels numbers at each line. Consecutive two coming lines are recorded to dual port RAM of the FPGA. When the next line comes, previous lines are read from the RAM and 3 pixel's information emitted for 2D filtering process. First output of line buffers is presented at Figure 3.9.

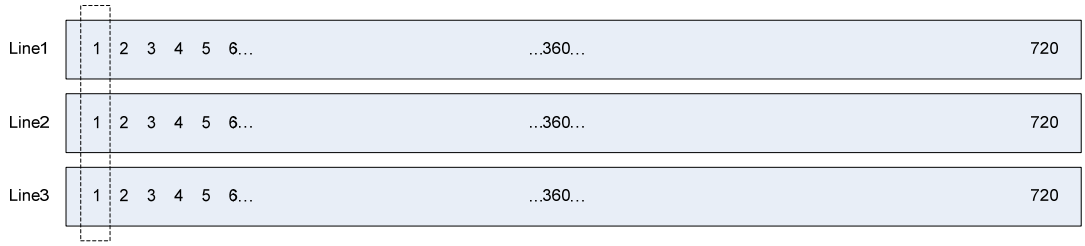

Figure 3.9 First output of line buffers

When the second pixel of Line3 comes, pixel's information of Line1, Line2 and Line3 is emitted. Second output of line buffers is presented at Figure 3.10.

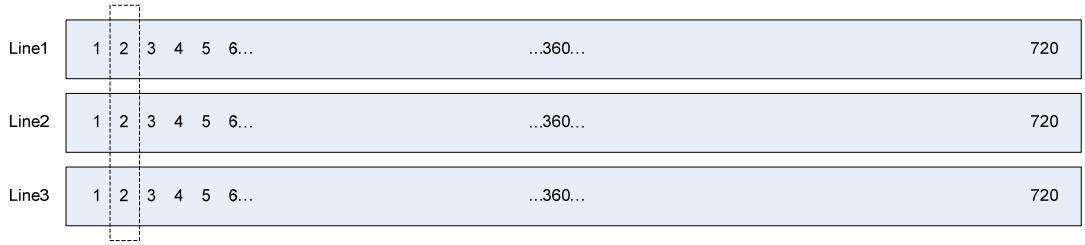

Figure 3.10 Second output of line buffers

For every coming pixel of Line3, three pixel's information is emitted. When a new line comes, information of previous two lines is used. This case can be seen at Figure 3.11.

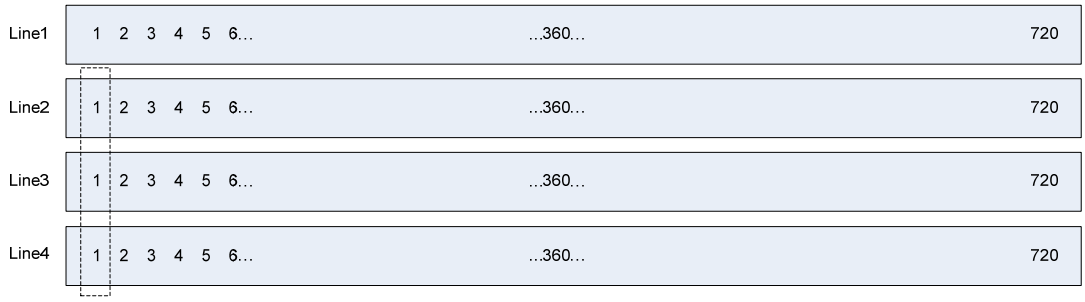

Figure 3.11 Line buffering for a new line

For each line, all the columns are scanned. After the buffering of Line288, process waits for other field (odd/even field) to start over buffering. Line buffering module in FPGA, starts to emit pixel's information after Line3; therefore it causes a three line delay in real time processing.

3.2.3.2.2.2 Convolution with 3x3 Gradient Matrix

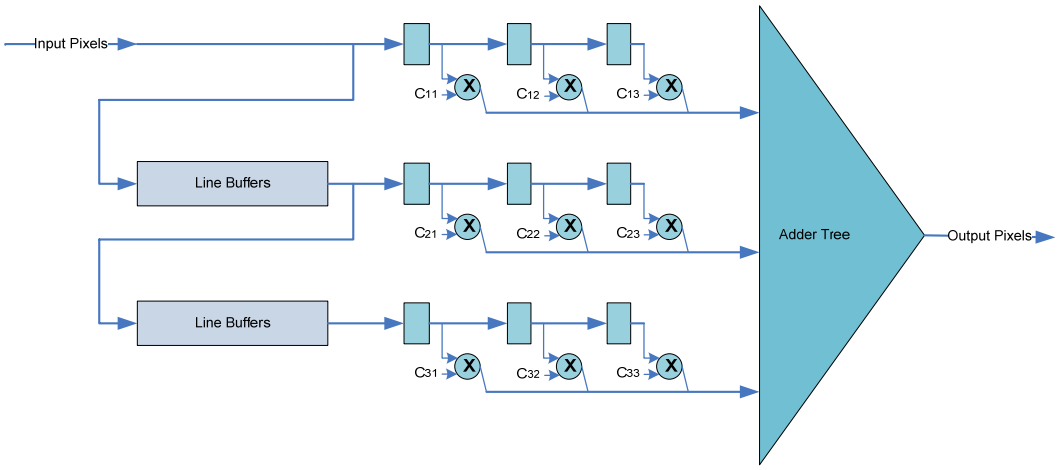

Figure 3.12 Hardware Implementation of 2D filter

Pixels coming from line buffers are buffered again to produce 3x3 window. In Figure 3.13, R letter indicates the row number and C letter indicates the column number of pixel.

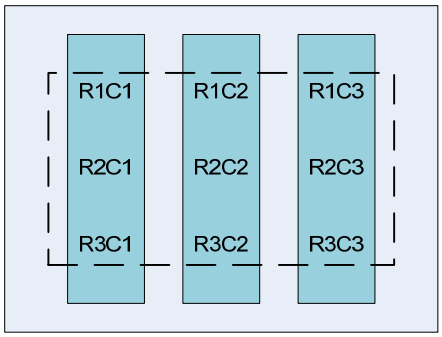

Figure 3.13 First generated 3x3 window

After coming of the new three pixels from the line buffer, a new 3x3 window is generated. The new window can be seen at Figure 3.14.

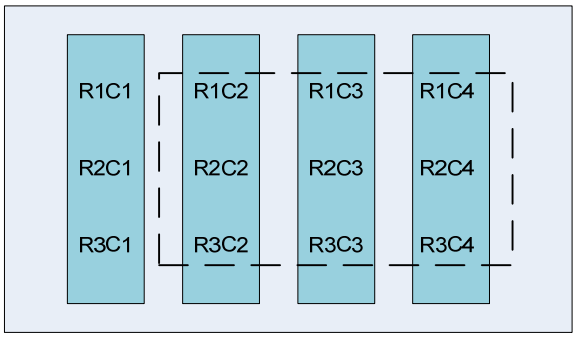

Figure 3.14 Second generated 3x3 window

For a new coming line, these processes are repeated. Generated windows after coming of a new line are presented in Figure 3.15.

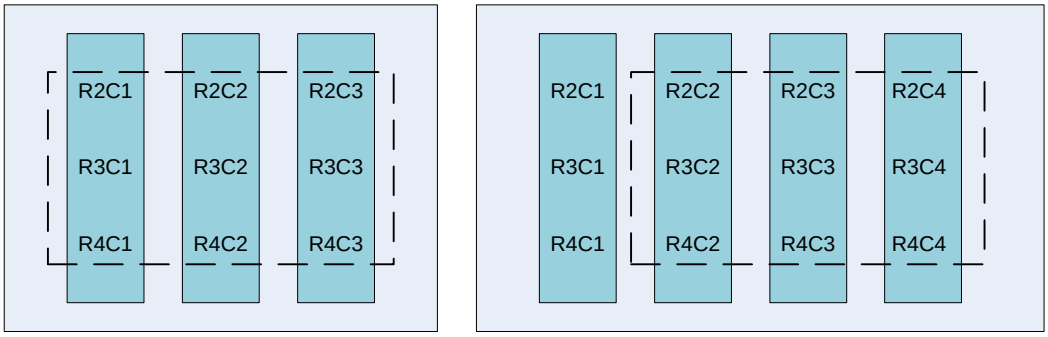

Figure 3.15 Generated windows after coming of a new line

Generated windows are multiplied with the Sobel gradient mask's coefficients. After that, results of the multiplication are summed as in Figure 3.12. Outcome of this process (convolution) is the result for the middle pixel value in the 3x3 window.

Convolution process with "Grad_x" mask and "Grad_y" mask are taking place concurrently. Then, absolute values of the results are found using a comparator and adder. Absolute values are used for magnitude calculation of gradient.

3.2.3.2.2.3 Magnitude Calculation & Thresholding

Let u(x, y) be a two dimensional edge segment. Orthogonal edge gradient can be formed by running difference of pixels in horizontal and vertical direction. It is defined in magnitude form in square root form as

$$G(x,y) = \left[[G_1(x,y)]^2 + [G_2(x,y)]^2\right]^{1/2} \tag{3.6}$$

Unfortunately, the square-root operation in this formula is inherently slow and complicated to calculate, either in software or in hardware. Therefore, for applications which do not require a full-precision magnitude value, the use of magnitude estimation can save calculations. This well-known and widely used algorithm is a true gem of DSP because it provides considerable savings in calculation, at the cost of only a minimal loss of accuracy.

$$G(x,y) = \alpha \cdot \max(|G_1|,|G_2|) + \beta.\min(|G_1|,|G_2|) \tag{3.7}$$

"α" and "β" are two constants whose values can be chosen to trade among RMS error, peak error, and implementation complexity. For our applications they are taken as; 1 and 1/2 [41].

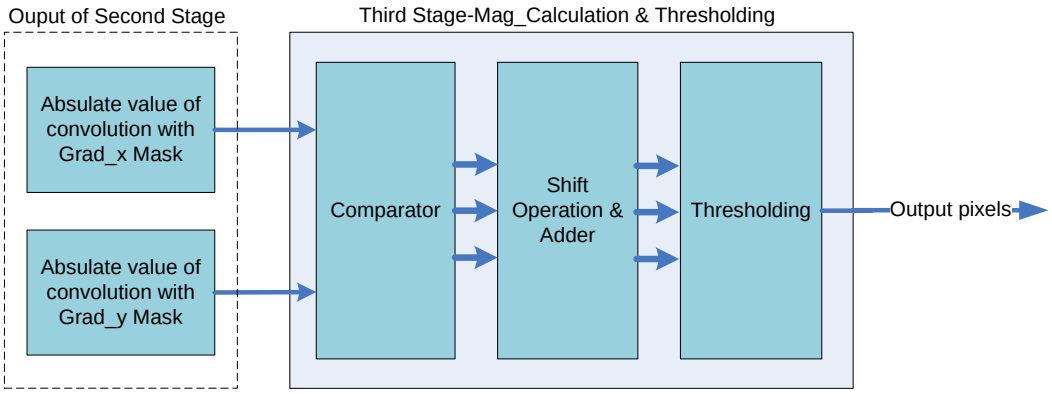

Figure 3.16 Block Diagram of Magnitude Calculation & Thresholding

In FPGA, absolute values of convolution processes are compared with each other to determine which one is the smallest one. Then, using the equation 3.7, magnitude value of the gradient is calculated. Shift operation is used to divide the minimum of |G1| or |G2|. After the calculation of gradient magnitude, a thresholding algorithm works. In principal, thresholding includes a comparator to detect if the magnitude value is greater than a predetermined threshold value. Optimum threshold values can be assigned after experimental tests.

3.2.3.3 SHAPE RECOGNITION

3.2.3.3.1 Detection of Circular Signs

For the detection of circular signs, fast radial symmetry detector is used. Many shape detectors are non-robust because they require closed shapes. Robust techniques such as Hough circle detection are slow to compute over large images. The fast radial
symmetry detector can be run as a detector at frame rate [2].

3.2.3.3.1.1 Theoretical Background

The transformation here works on the gradient information of images. Hence, edge detection is accompanied to gathered frames. Then, algorithm searches the gradient information for a circle with an initial estimate of its radius. Algorithm is carried on a set of radii (around initial estimate) to locate sign candidates.

In the algorithm, remaining non zero gradient elements votes for a potential circle center a distance *r* (where *r* is the radius of the candidate circle) away along the line of the gradient (either having same direction or having opposite direction). For locating the center of the circle, an orientation projection image is calculated for each image frame by examining the gradient image.

For a given pixel, p, the gradient g, is calculated using the Sobel edge operator. If this pixel lay on the arc of a circle, then its centre would be in the direction of the

gradient [2]. Locations of positively and negatively affected pixels can be seen from Figure 3.17.

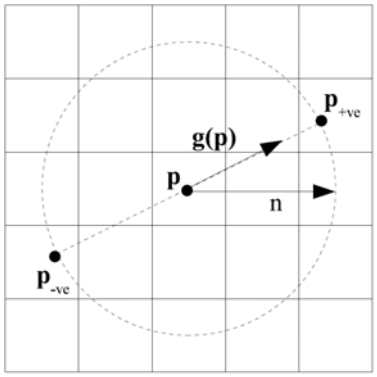

Figure 3.17 Locations of affected pixels

The coordinates of a positively- affected pixel are defined as:

$$p_{+ve}(p) = p + round(\frac{g(p)}{\|g(p)\|}n) \tag{3.8}$$

At each radius $n \in N$, an orientation projection image O_n is formed. It is initially zero and updated by

$$O_n(p_{+ve}) = O_n(p_{+ve}) + 1 \tag{3.9}$$

The vote image is defined as:

$$\hat{F}_n(p) = \text{sgn}(\tilde{O}_n(p))\left(\frac{\left|\tilde{O}_n(p)\right|}{k_n}\right)^{\alpha} \tag{3.10}$$

where,

$$\text{sgn}(\tilde{O}_n(p)) = \frac{\tilde{O}_n(p)}{\left|\tilde{O}_n(p)\right|} \tag{3.11}$$

and

$$\tilde{O}_n(p) = \begin{cases} O_n(p) & \text{if} \quad O_n(p) < k_n \\ k_n & \text{otherwise} \end{cases} \tag{3.12}$$

therefore;

$$\hat{F}_n(p) = \frac{\tilde{O}_n(p)}{k_n^{\alpha}} \cdot \left| \tilde{O}_n(p) \right|^{\alpha-1} \tag{3.13}$$

Radial symmetry image is obtained:

$$S_n = \hat{F}_n * A_n \tag{3.14}$$

where,

A_n is the two dimensional Gaussian.

In order to calculate full transform image for more than one radius value that averages all the symmetry contributions over all radii considered [2]:

$$S = \frac{1}{|N|} \Sigma_n S_n \tag{3.15}$$

This transform process can be adapted for circular sign detection. Suitable parameters of this transform are chosen for our case.

Determining the Parameters

In order to implement real time radial symmetry for circular traffic signs, some parameters have to be determined. These parameters are;

- $n \in N$: Set of radii

- α :The radial strictness parameter
- k_n : The normalizing factor
- A_n : The Gaussian kernel

Set of radii

When driving a car, there is an optimum distance to detect the traffic sign before it becomes too large. Therefore, this optimum value can be used for our detection process as a radius value. For this algorithm, the set of [16,18] values can be used as the radii set. However, only the radius value 16 is used in FPGA implementation of the algorithm because of the time restrictions of making multiplication on hardware.

The radial strictness parameter α

The radial strictness parameter α determines how strictly radial the radial symmetry must be for the transform to return a high interest value [29]. That means, choosing lower α value emphasizes non-radially symmetric features over the image. However, a higher α value causes elimination of some non-radially symmetric points such as lines.

For the FPGA implementation of algorithm, α value is determined as 1 because of minimizing the computation time of $\hat{F}_n$.

The normalizing factor k_n

When the radius becomes larger, the number of gradient elements affecting the transformation increases. For the case of using of different radii values, O_n have to be normalized to make every orientation projection image into a similar scale. Normalization of O_n is achieved by dividing with k_n.

For the FPGA implementation of algorithm, k_n value is determined experimentally as 16, because of minimizing the computation time of F_n

The Gaussian Kernel A_n

The purpose of the Gaussian kernel A_n is to spread the influence of the positively and negatively-affected pixels as a function of the radius n [29]. 2-D Gaussian is used because of its radially symmetric structure; therefore, convolution process has a consistent effect over all gradient orientations. The standard deviation with the range n determines the arc of influence which applies to all affected pixels. In addition, amplifying the magnitude is necessary to prevent the effect of gradient elements becoming negligible at large radii as a result of being spread too thinly across the image [29]

For the FPGA implementation of algorithm, Gaussian kernel is chosen as 2D Gaussian of size $n \times n$ with,

$$\sigma = 0.25n \quad \text{and} \quad n = 3 \tag{3.16}$$

3.2.3.3.1.2 Hardware Implementation on FPGA

Implemented algorithm in FPGA can be summarized in Figure 3.18. First stage was implemented in Sobel Edge detection. In this part, implementation of the second stage in Figure 3.18 will be explained.

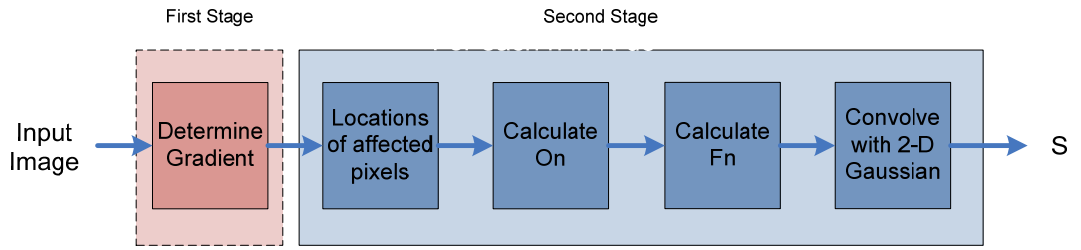

Figure 3.18 Block diagram of the algorithm

In the second stage; first, locations of positive affected pixels are determined. In order to accomplish this purpose, row and column number of the pixel and

magnitude of the gradients have to be known. These variables are calculated in the "Sobel Edge Detection" part, shown in Figure 3.19 in Stage-1.

Clock multiplier is used to produce 350 MHz clock from 200 MHz system clock. 350 MHz clock signal is chosen, because, it is the maximum clock frequency of divider modules.

Using equation (3.7) gradient magnitude can be calculated. Gradient values coming from the Sobel edge detection are divided with the magnitude of the gradient value. Therefore; two divider modules are used to calculate to find positive affected pixels locations. Locations are expressed with row and column numbers.

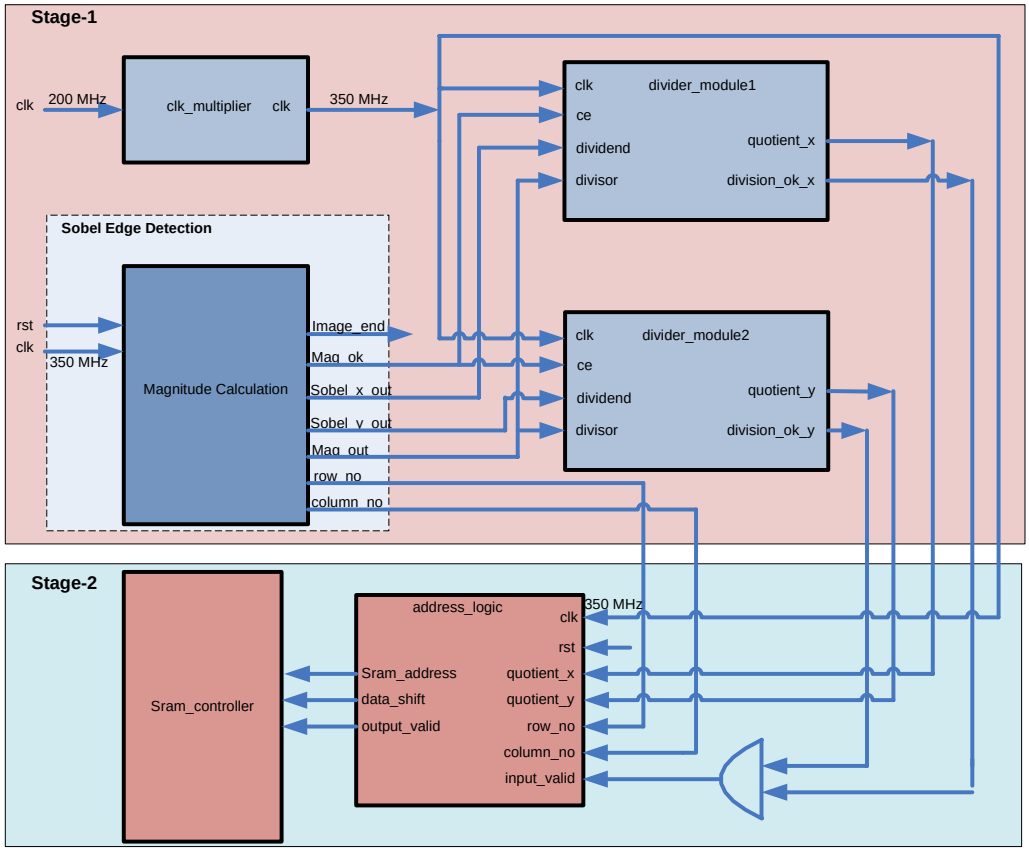

Figure 3.19 Hardware modules to find the locations of positive affected pixels

In Equation (3.8), the radius value (n) is taken as 16. Therefore, multiplication of gradient values with *n* corresponds to a shifting operation. “Sobel_x_out” and “Sobel_y_out” are 16 bits output ports of “Magnitude_Calculation” module, which include the results of multiplication of gradient values with radius. In 350 MHz, 2 clock cycles are used for magnitude calculation. Simulation for magnitude calculation block is shown in Figure 3.20. Row and column numbers are calculated in this block. Using the gradient information coming from sobel edge block, magnitude of the gradient is calculated using the Equation (3.7). In Figure 3.20, “mag_out[15:0] ” shows the output value corresponds to “sobel_x_in[11:0]” and “sobel_y_in[11:0]”. In addition, “sobel_x_out[15:0] ” and “sobel_y_out[15:0]” are the radius times “sobel_x_in[11:0]” and “sobel_y_in[11:0]” in which the radius is 16. Divider modules use the “sobel_x_out[15:0] ” and “sobel_y_out[15:0]” as inputs to calculate Equation (3.8).

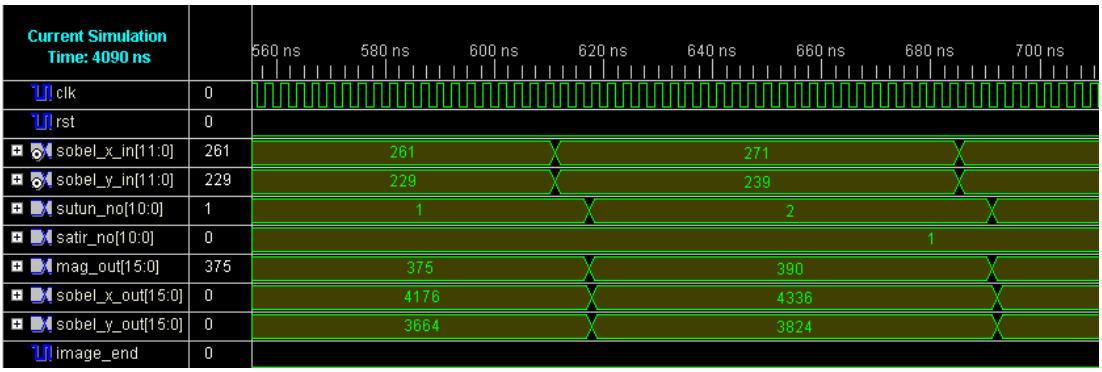

Figure 3.20 Simulation for Magnitude Calculation block

Divider modules work with 350 MHz and spend 18 clock cycles to generate results. 13,5 MHz is the clock frequency of the luminance value of the video signal. Therefore; neither divider module nor magnitude calculation module cause to loss of data.

Divider modules calculate the right side of the Equation (3.8) (marked area shown in (3.17)). The locations of positive affected pixels are calculated in “address_logic” component.

$$p_{+ve}(p) = p + round(\frac{g(p)}{\|g(p)\|} n) \tag{3.17}$$

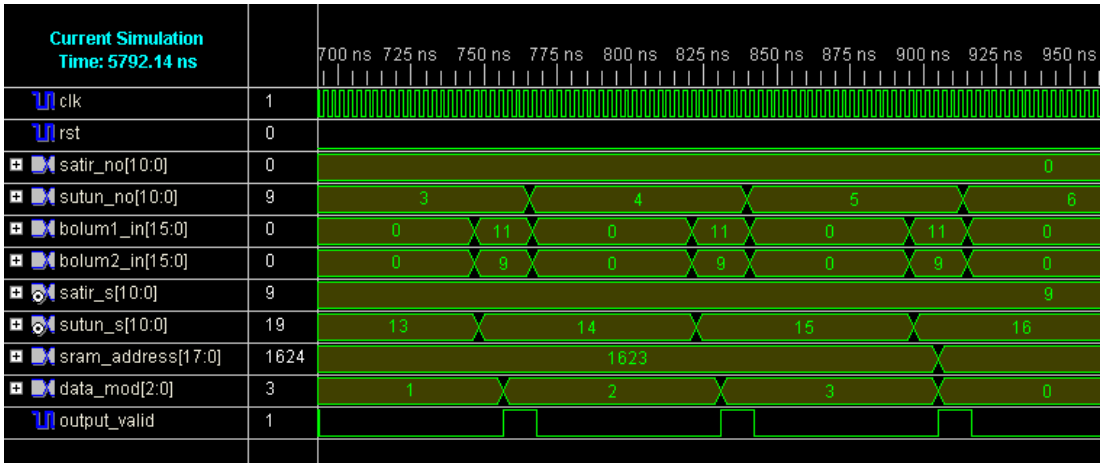

Figure 3.21 Simulation for Address logic block

In "address_logic" component, "satir_s[10:0]" and "sutun_s[10:0]" are the internal signals which indicate the locations of positive affected pixels.

satir_s <= satir_no + bolum2_in (3.18)

sutun_s <= sutun_no + bolum1_in (3.19)

Equations (3.16) and (3.17) are used to calculate the locations of positive affected pixels. In these equations, "bolum1_in" indicates the output value of "divider module1" and "bolum2_in" indicates the output value of "divider module2".

After calculating the coordinates of positive affected pixels, orientation projection image is calculated. The orientation projection images are initially zero. For each pair of affected pixels p_{+ve} the corresponding point in the orientation projection image O_n is incremented by one [29].

O_n has a size 288x720x8 bits therefore, internal memory of FPGA is not sufficient to save a place for O_n voting. In order to find O_n, an external memory (SRAM) is used. SRAM has 32 bits data bus so, "address_logic" module, shown in Figure 3.19, is used to determine the SRAM address with data shifting information. *On (0,0), On(0,1), On(0,2) and On(0,3)* are recorded the same SRAM address. But, they are in the different intervals of the 32 bits data as shown in Figure 3.22.

SRAM				Address
On(0,3)	On(0,2)	On(0,1)	On(0,0)	0000
On(0,7)	On(0,6)	On(0,5)	On(0,4)	0001
...	...	...	...	...
On(0,719)	...	...	...	00B3
...	...	...	On(1,0)	...
				...
				...

Figure 3.22 Addresses of SRAM

Suitable SRAM address is generated in "address_logic" block. Furthermore, data shifting information is also calculated in "address_logic" block. As shown in Figure 3.21, "sram_address[17:0]" indicates address of SRAM and "data_mod[2:0]" indicates the shifting information.

To calculate the address of SRAM and data shifting information, Equation (3.20) and (3.21) are used.

$$\text{data\_mod} = \text{sutun\_s } mod\ 4 \tag{3.20}$$

$$\text{sram\_address} = \text{satir\_s x (0xB4)} + [\text{sutun\_s- (data\_mod)}] / 4 \tag{3.21}$$

SRAM controller module works with 200 MHz clock. Writing from or reading to SRAM spends 7 clock cycles. During the voting of O_n, reading and writing

processes take place consecutively. Therefore, 14 clock cycle is used for voting application of O_n. 13,5 MHz is the clock frequency of the luminance value of the video signal. For this reason, during the voting of O_n, there is no loss of data. However, when the row and columns numbers reach the limit values (288 for row; 720 for column), “image_end” signal rises up from the “ mag_calculation” module. With the rising edge of that signal, calculation process of F_n starts, shown in Figure 3.23 (marked area in the figure).

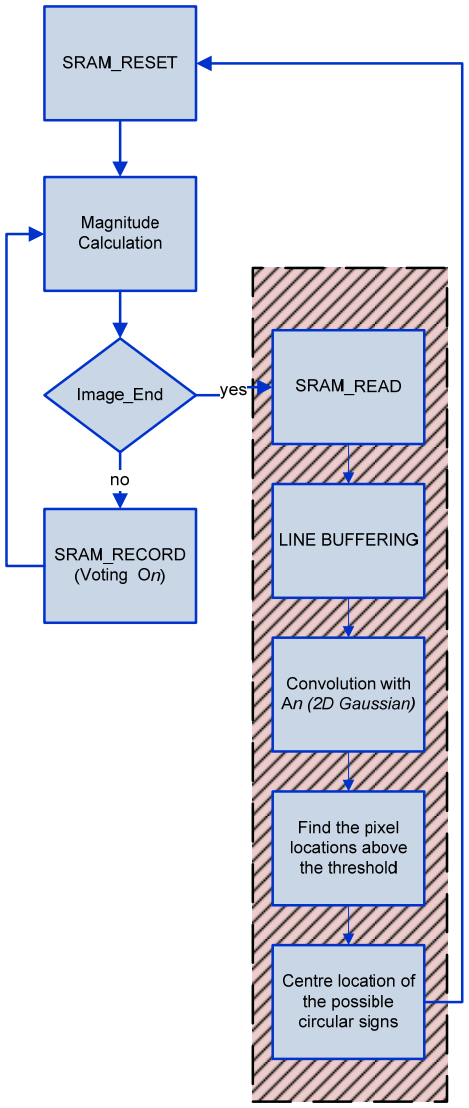

Figure 3.23 Flow chart of the circular sign detection algorithm

In Equation (3.12), k_n is selected as 16. To find the vote image $\hat{F}_n(p)$ using Equation (3.13), $\tilde{O}_n(p)$ is divided by k_n which is 16. Normally, the result is a floating point number. In FPGA, working with floating point number is not easy. Therefore, for Equation (3.13) dividing operation with k_n is ignored. In this way, $\tilde{O}_n(p)$ and $\hat{F}_n(p)$ can be calculated as fixed point numbers.

After finding $\hat{F}_n(p)$, Radial symmetry image S_n is calculated using Equation (3.14). A_n is a two dimensional Gaussian kernel which is used to spread the influence of the positively and negatively-affected pixels as a function of the n. For FPGA implementation, n is taken as 3 and standard deviation is taken as σ = 0.85. Therefore normalized Gaussian kernel A_n is:

$$A_n = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.22}$$

To find Radial symmetry image S_n, a convolution operation is needed. Using FPGA, convolution operation of $\hat{F}_n(p)$ with A_n is a similar process like described in section 3.2.3.2.2.2. First of all, line buffers are used as described in 3.2.3.2.2.1. Then, a 3x3 window is generated from the pixels coming from line buffers. Generated windows are multiplied with the Gaussian 3x3 mask's coefficients. As you notice, coefficients of A_n are arranged as to be multiply of 2. Therefore, multiplication operation with these coefficients can be performed with simple shifting operations.

Results of multiplication of 3x3 window and Gaussian mask's coefficients are compared with a threshold value as shown in Figure 3.24.

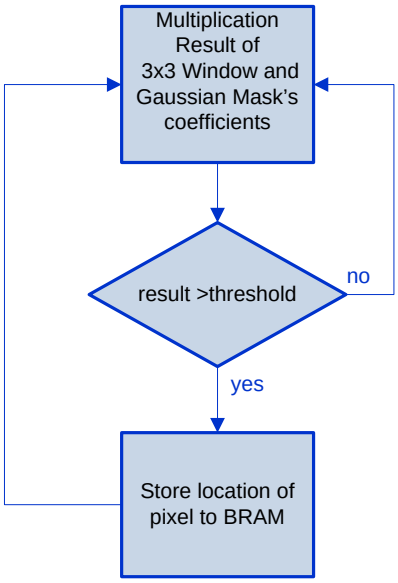

Figure 3.24 Flow chart of the comparison of convolution result with threshold

Results stored in Block RAM indicate the possible center locations of circular traffic signs in the video frame.

Threshold values are chosen after MATLAB simulations and experimental results. For our applications threshold value is taken as 50.

CHAPTER 4

EXPERIMENTAL RESULTS

4.1 Introduction

In order to validate the algorithms implemented on FPGA, as stated in Chapter 3, they are first developed in MATLAB. For the test of the algorithms, static images [36] are initially used. After porting to FPGA real time video sequences are also used.

For real time sequences, a laptop with S-Video output port is used while playing the stored video. In additions to video frames, static images are used in tests of algorithms.

First step of the proposed algorithm is color segmentation. Second step of proposed algorithm is edge detection. Edge detection algorithms are applied after color segmentation algorithms. Results of joint application of algorithms are given for different images and video sequences in section 4.4.1.

Third step of proposed algorithm is circle shape recognition based algorithms. Results of joint applications of all algorithms are given for different images and video sequences in section 4.4.2.

Algorithms applied for detection of traffic signs are developed on two different platforms which are:

- MATLAB
- Xilinx ISE 10.1

4.2 Test Results

Three groups of images with the following properties are used in testing of the MATLAB and FPGA implementations of algorithms.

Well Illuminated: This set includes well illuminated 47 images which contains 19 triangular, 13 circular, 15 rectangular traffic signs (25 red, 22 blue signs). Images are captured from video scenes. Resolutions of the images are 720x576. An example for well-illuminated images is given in Figure 4.1.

Figure 4.1 Well-illuminated traffic sign

Poorly Illuminated: Images in this set are either not well illuminated or illumination causes detection process to be more difficult (i.e. sunlight directly coming just behind the traffic sign).This set includes 42 images which contain 17 triangular, 15 circular, 10 rectangular traffic signs (27 red, 15 blue signs). Images are captured from video scenes. Resolutions of the images are 720x576. An example for poorly-illuminated images is given in Figure 4.2.

Figure 4.2 Poorly-illuminated traffic sign

Shadowed: This set includes 30 images which are in shade. This group of images consists of 14 circular, 3 rectangular, 13 triangular traffic signs (21 red, 9 blue). Resolutions of the images are 720x576. An example for shadowed images is given in Figure 4.3.

Figure 4.3 Shadowed traffic sign

4.2.1 MATLAB Test Results of Algorithms on Separate Images

In literature, there are many algorithms to detect traffic signs. Basically, these algorithms use color and shape features of traffic signs. For real time application of traffic sign detection, joint application of shape based algorithms and color segmentation is preferred to decrease processing time of detection. Steps defined in Figure 4.4 are implemented by using MATLAB.

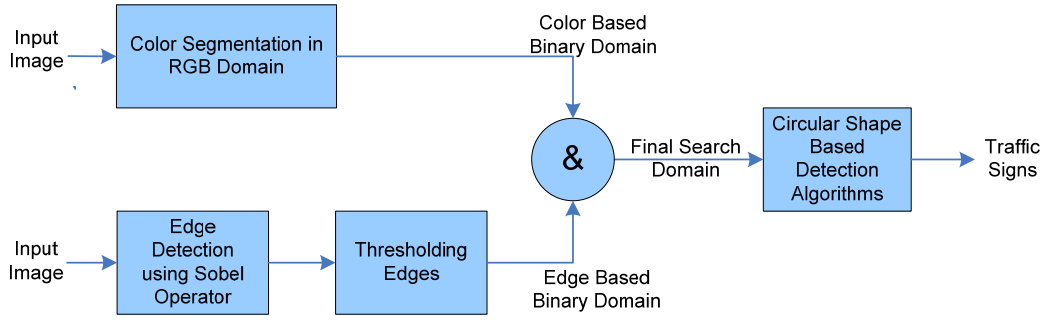

Figure 4.4 Joint application of color and shape based algorithms

For MATLAB simulations, parameters for circular shape detection algorithm are different from the parameter values defined in 3.2.3.3.1.1. The parameter values are taken as:

- $n \in N$: Set of radii is taken as [16,18]
- α:The radial strictness parameter is taken as 1,
- k_n : The normalizing factor is taken as 18,
- A_n : The Gaussian Kernel; $\sigma = 0.25n$ and $n = 3$

MATLAB test results are given in following sections. Finding the location of circular shape's centers at wrong places is taken as a false positive case.

4.2.1.1 Test Results on Well-Illuminated Images

For this set, threshold values for color segmentation are taken as:

$R_a = 75$, $R_b = 255$, $G_a^{'} = 0,45$ $G_b^{'} = 0,45$, $B_a^{'} = 0,45$, $G_a^{'} = 0,45$.

Threshold value for circular shape detection algorithm is selected as 40. Detection result is given in Table 4.1.

Table 4.1 Output results for circular sign detection system for well illuminated images (t=40)

Shape	***Signs to be Detected***	***Detected Traffic Signs***	***% of Detection***	***False Positives***
Circular	13	13	% 100	3

Threshold value for circular sign detection algorithm determines the detection rate and number of false positives. Small threshold values cause more false positives. Objects which have less radially symmetric features can be detected. The threshold value is selected as 50. Detection result is given in Table 4.2.

Table 4.2 Output results for circular sign detection system for well illuminated images (t=50)

Shape	***Signs to be Detected***	***Detected Traffic Signs***	***% of Detection***	***False Positives***
Circular	13	12	% 92,30	1

4.2.1.2 Test Results on Poorly Illuminated Images

For poorly illuminated images, color segmentation module has an important role for the performance of whole system. Threshold values for color segmentation are taken as:

$R_a = 75$, $R_b = 255$, $G_a^{'} = 0,45$ $G_b^{'} = 0,45$, $B_a^{'} = 0,45$, $G_a^{'} = 0,45$.

Threshold value for circular detection algorithm is selected as 40. Detection result is given in Table 4.3.

Table 4.3 Output results for circular sign detection system for poor illuminated images

Shape	***Signs to be Detected***	***Detected Traffic Signs***	***% of Detection***	***False Positives***
Circular	15	8	%53,3	0

For poor illumination, thresholds for color segmentation can be optimized. Threshold values for color segmentation can be taken as:

$R_a = 55$, $R_b = 255$, $G_a^{'} = 0,65$ $G_b^{'} = 0,65$, $B_a^{'} = 0,65$ $G_a^{'} = 0,65$.

Table 4.4 shows the output results of the system after color segmentation threshold change.

Table 4.4 Output results for circular sign detection system for poor illuminated images (after threshold changing for color segmentation)

Shape	***Signs to be Detected***	***Detected Traffic Signs***	***% of Detection***	***False Positives***
Circular	15	11	%73,3	2

Change of color segmentation threshold increases the rate of detection but, it also causes false positive detection.

4.2.1.3 Test Results on Shadowed Images

Circular sign detection algorithm gives good results for partial shadowed images. Threshold values for color segmentation are taken as:

$R_a = 75$, $R_b = 255$, $G_a^{'} = 0,45$ $G_b^{'} = 0,45$, $B_a^{'} = 0,45$ $G_a^{'} = 0,45$.

Threshold value for circular detection algorithm is selected as 40. Table 4.5 shows the output results of the system.

Table 4.5 Output results for circular sign detection system for shadowed images

Shape	***Signs to be Detected***	***Detected Traffic Signs***	***% of Detection***	***False Positives***
Circular	14	7	%50	1

For shadowed images, thresholds for color segmentation can be optimized. Threshold vales for color segmentation can be taken as:

$R_a = 55$, $R_b = 255$, $G_a^{'} = 0$, $G_b^{'} = 0,65$, $B_a^{'} = 0$, $G_a^{'} = 0,65$.

Table 4.6 shows the output results of the system after color segmentation threshold change.

Table 4.6 Output results for circular sign detection system for shadowed images (after the change of color segmentation module threshold)

Shape	***Signs to be Detected***	***Detected Traffic Signs***	***% of Detection***	***False Positives***
Circular	14	9	%64,28	3

Similarly, changing the threshold of circular shape detection module can increase the detection rate. Partial shadowed images reveal less symmetric parts. Therefore, decreasing the threshold of shape detection algorithm increases the detection rate of system. In a contrary manner, false positives tend to increase. Threshold vales for color segmentation can be taken as:

$R_a = 55$, $R_b = 255$, $G_a^{'} = 0$, $G_b^{'} = 0{,}65$, $B_a^{'} = 0{,}65$ $G_a^{'} = 0{,}65$, and

Threshold for circular shape detection algorithm has changed from 40 to 35.

Table 4.7 shows the output results of the system after shape detection threshold change.

Table 4.7 Output results for circular sign detection system for shadowed images (after the change of shape detection threshold and color segmentation threshold)

Shape	***Signs to be Detected***	***Detected Traffic Signs***	***% of Detection***	***False Positives***
Circular	14	10	%71,42	4

MATLAB results are compared with the results found in the [38]. Once the performance is guaranteed, it is implemented on FPGA using ISE software.

4.2.2 Test Results of FPGA Implementation

4.2.2.1 Test Results of Joint Application of Color Segmentation and Edge Detection Implementation on Separate Images

Color Segmentation and edge detection algorithms are intermediate steps for detection of traffic signs. The last step, shape recognition, will determine the detection rate of hardware implementation. However, the outputs of color segmentation and edge detection modules directly affect the performance of traffic sign detection. Therefore, using following two case, performance of color segmentation and edge detection modules can be discussed.

Case-1 (Pass): If the output of module (hardware implementation of algorithm) consists of traffic signs completely (with no corruption), algorithm is assumed as successfully. In the output image, there can be other objects with the traffic signs. Figure 4.5 shows the image captured from a video scene. Figure 4.6 shows the output of color segmentation module. As seen in Figure 4.6, output of the color segmentation module shows the traffic sign successfully. Output of joint application of color segmentation and edge detection module is shown at Figure 4.7.

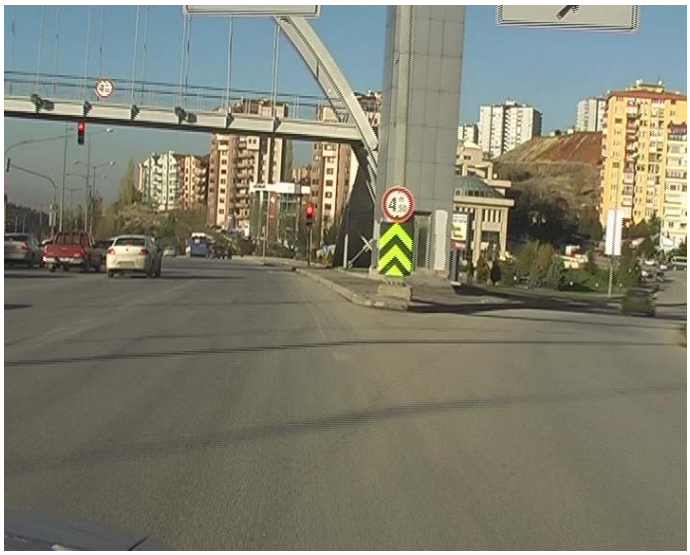

Figure 4.5 Original image having one circular sign

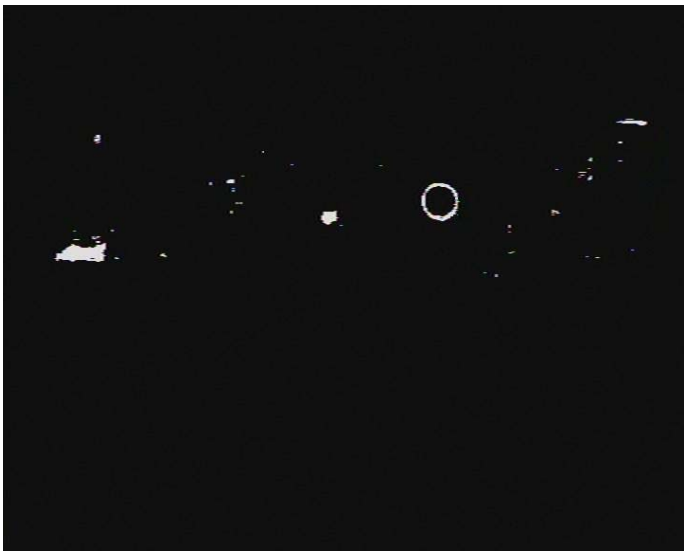

Figure 4.6 Output of color segmentation module (for red component)

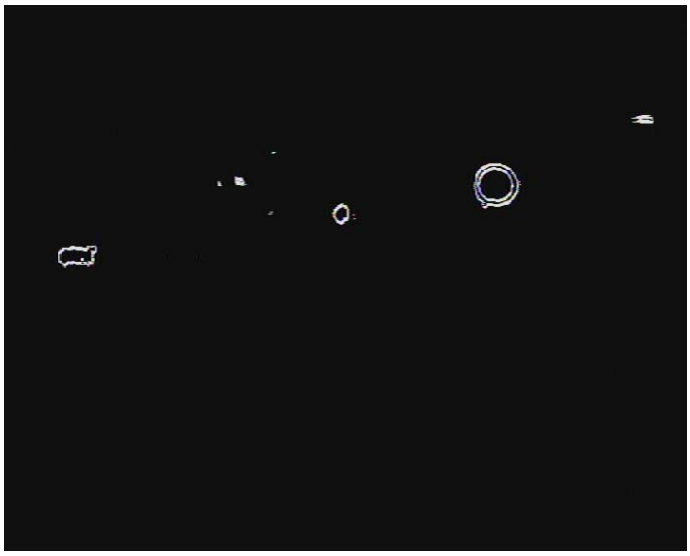

Figure 4.7 Output of joint application of color segmentation & edge detection module

Case-2(Fail): If the output of module doesn't include traffic sign completely, algorithm is assumed as unsuccessful. Figure 4.8 shows the image captured from a video scene. Figure 4.9 shows the output of color segmentation module. As seen in Figure 4.9, right side of the triangle doesn't come out from the algorithm because of the shadow effect of trees. This case assumed as a failure. Output of joint application of color segmentation and edge detection module is shown at Figure 4.10.

Figure 4.8 Original image having one circular sign

Figure 4.9 Output of color segmentation module (for red component)

Figure 4.10 Output of joint application of color segmentation & edge detection module

The shape recognition module uses the output images shown in Figure 4.7 and Figure 4.10. Therefore, performance of edge detection depends on the color segmentation and edge detection modules.

Well illuminated, poorly illuminated, and shadowed images are studied for two cases expressed above.

4.2.2.1.1 Test Results on Well-Illuminated Images

For well illuminated images, color segmentation and edge detection module works perfectly. The real performance of sign detection is determined by shape detection module. Table 4.8 shows the output results for well-illuminated images according to the defined cases in section 4.4.1.

Table 4.8 Output results for joint application of color segmentation & edge detection module

Color	**Signs to be Detected**	**Number of Passed**	**Number of Failed**	***% of Passed***
Red	25	23	2	92
Blue	22	19	3	86,36

Five failed case occurs because of the occlusion of other objects.

4.2.2.1.2 Test Results on Poorly-Illuminated Images

For poorly illuminated images, especially when the sun shines just behind the sign, the output image of color segmentation module can corrupt. Therefore, the performance of color segmentation module has a big role in performance of traffic sign detections. Table 4.9 shows the output results for poorly-illuminated images according to the defined cases in section 4.4.1.

Table 4.9 Output results for joint application of color segmentation & edge detection module

Color	**Signs to be Detected**	**Number of Passed**	**Number of Failed**	***% of Passed***
Red	27	15	12	55,55
Blue	15	8	7	53,33

4.2.2.1.3 Test Results on Shadowed Images

Color segmentation module fails because of the dark shadows appearing over whole or most part of signs. Actually, partial shadows don't affect the shape of the sign coming from the joint application of color segmentation and edge detection module. Sign recognition algorithms still give good results for partial shadowed images. But, these kinds of situations are admitted as a failure in Table 4.10 according to the defined cases in section 4.4.1.

Table 4.10 Output results for joint application of color segmentation & edge detection module

Color	**Signs to be Detected**	**Number of Passed**	**Number of Failed**	***% of Passed***
Red	21	12	9	57,14
Blue	9	4	5	44,4

4.2.2.2 Test Results of Circular Sign Detection Implementation on Separate Images

Circular shape detection algorithm uses the gradient information of pixels as inputs. That means in FPGA implementation of circular sign detection algorithm uses the output of joint application of color segmentation, edge detection algorithms as shown in Figure 4.4. The output of whole system is the location information of possible circular traffic sign. In the output video circular sign is marked with square and whole other areas are whitened. The output of system is shown in Figure 4.11.

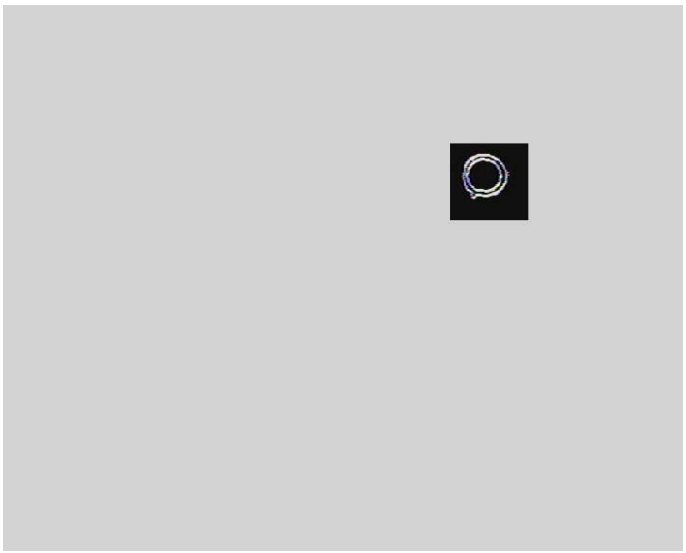

Figure 4.11 Output of the system

Three group of pictures described in 4.2 are used to determine the performance of whole system. For the FPGA implementation of circular sign detection algorithm based on radial symmetry, some approximations are taken into considerations. Especially, the parameter values defined in 3.2.3.3.1 are approximated for FPGA implementations. Therefore, there are differences between the results of MATLAB simulations and FPGA implementations.

4.2.2.2.1 Test Results on Well-Illuminated Images

From the MATLAB results, optimum thresholds for color segmentation module and shape detection module are selected.

Threshold vales for color segmentation are:

$R_a = 75$, $R_b = 255$, $G_a^{'} = 0$, $G_b^{'} = 0{,}45$, $B_a^{'} = 0$, $G_a^{'} = 0{,}45$.

Threshold value for circular sign detection module is 50.

Table 4.11 shows the results for well-illuminated images.

Table 4.11 Output results for circular sign detection system for well illuminated images

Shape	***Signs to be Detected***	***Detected Traffic Signs***	***% of Detection***	***False Positives***
Circular	13	11	% 84,61	2

The result of FPGA implementation is similar to the result of MATLAB. Because of the approximations for FPGA implementations, some of the signs can't be detected. In addition, using only one radius value and approximation of Gaussian kernel as expressed in Equation (3.22), cause to increase of false positives.

4.2.2.2.2 Test Results on Poorly Illuminated Images

From the MATLAB results, optimum thresholds for color segmentation module and shape detection module are selected.

Threshold vales for color segmentation are:

$R_a = 55$, $R_b = 255$, $G_a^{'} = 0$, $G_b^{'} = 0{,}65$, $B_a^{'} = 0$, $G_a^{'} = 0{,}65$.

Threshold value for circular sign detection module is 40.

Table 4.12 shows the output results of the system.

Table 4.12 Output results for circular sign detection system for poorly illuminated images

Shape	***Signs to be Detected***	***Detected Traffic Signs***	***% of Detection***	***False Positives***
Circular	15	10	%66,6	1

The detection rate is close to the MATLAB results.

4.2.2.2.3 Test Results on Shadowed Images

From the MATLAB results, optimum thresholds for color segmentation module and shape detection module are selected.

Threshold vales for color segmentation are:

$R_a = 55$, $R_b = 255$, $G_a^{'} = 0$, $G_b^{'} = 0{,}65$, $B_a^{'} = 0$, $G_a^{'} = 0{,}65$.

Threshold value for circular sign detection module is 35.

Table 4.13 shows the output results of the system.

Table 4.13 Output results for circular sign detection system for shadowed images

Shape	***Signs to be Detected***	***Detected Traffic Signs***	***% of Detection***	***False Positives***
Circular	14	9	%64,28	2

Circular sign recognition algorithm gives good results for partial shadowed images. Similarly, the detection rate of FPGA implementation is lower than the MATLAB results.

4.2.3 Timing and Resource Usage

For color segmentation algorithm, two Y values are used to detect the RGB values of a pixel. Video decoder clock frequency is 27 MHz and system clock frequency is 200 MHz. Information of two pixels ($Cb_1Y_1\ Cr_2Y_2$) is used to find the one pixel's RGB value. 4 clock cycles in 200 MHz are needed for color conversion from YUV space to RGB space. In addition, 2 clock cycles are used for color thresholding. Therefore, 6 clock cycles are used for color segmentation and there is no loss of

data. Two pixel delay occurs because of the color space conversion. Delay is expressed as (1/13,5MHz)74 ns x 2 clock cycles = 148 ns.

Logic resource utilization of designed hardware on FPGA is given in Figure 4.12.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	906	44,800	2%
Number used as Flip Flops	906		
Number of Slice LUTs	1,774	44,800	3%
Number used as logic	1,585	44,800	3%
Slice Logic Distribution			
Number of occupied Slices	668	11,200	5%
Number of LUT Flip Flop pairs used	1,989		
Specific Feature Utilization			
Number of BlockRAM/FIFO	52	148	35%
Number using BlockRAM only	52		
Total primitives used			
Number of 36k BlockRAM used	51		
Number of 18k BlockRAM used	1		
Total Memory used (KB)	1,854	5,328	34%
Number of BUFG/BUFGCTRLs	6	32	18%
Number used as BUFGs	6		

Figure 4.12 Logic Resource Utilization after Color Segmentation

Edge detection module uses the outputs of color segmentation module as inputs. Coming data is buffered for convolution operation with Sobel masks. Therefore,

two video line-delays occur. Delay is expressed as; (1/13,5MHz) x720 pixels x 2 lines = 106 μs .

Logic resource utilization of designed hardware on FPGA is given in Figure 4.13.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	1,086	44,800	2%
Number used as Flip Flops	1,086		
Number of Slice LUTs	1,977	44,800	4%
Number used as logic	1,803	44,800	4%
Slice Logic Distribution			
Number of occupied Slices	761	11,200	6%
Number of LUT Flip Flop pairs used	2,273		
Specific Feature Utilization			
Number of BlockRAM/FIFO	32	148	21%
Number using BlockRAM only	32		
Total primitives used			
Number of 36k BlockRAM used	31		
Number of 18k BlockRAM used	2		
Total Memory used (KB)	1,152	5,328	21%
Number of BUFG/BUFGCTRLs	9	32	28%
Number used as BUFGs	9		

Figure 4.13 Logic Resource Utilization after Edge Detection

Circular shape recognition module uses the outputs of edge detection module as inputs. Voting process of O_n as shown in Figure 3.22, runs simultaneously with the

video signal. After the voting process of O_n, the radial symmetry image S_n process starts to run with the rising edge of signal "image_end". This process doesn't work simultaneously with the video signal (marked area in Figure 3.22) However, working in 200 MHz provides to complete the line buffering and convolution with Gaussian mask processes before the next video field comes. The calculations are presented below;

For line buffering operations;

- 50 MHz clock is used
- 288x720 pixels are buffered

Therefore;

(1/50MHz) x 288 x 720 = 4,14 ms is needed for whole process.

Convolutions with Gaussian mask (3x3) works with 200MHz clock. The important point is that clock of convolution operation has to be at least three times bigger than the clock of line buffering. Because, producing 3x3 window from line buffer's coming data, multiplication with Gaussian mask coefficients and adding the results take at least 3 clock cycles.

For PAL (625-line/50 Hz video systems) standard there are 50 fields for a second, which means there are 20 ms between two consecutive fields. Therefore, after 4.14 ms, the sign detection system catches up the next coming field. Consequently, the system works with a delay of 20 ms.

Logic resource utilization of designed system hardware in FPGA is given in Figure 4.14.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	3,117	44,800	6%
Number used as Flip Flops	3,117		
Number of Slice LUTs	3,326	44,800	7%
Number used as logic	3,077	44,800	6%
Slice Logic Distribution			
Number of occupied Slices	1,486	11,200	13%
Number of LUT Flip Flop pairs used	4,539		
Specific Feature Utilization			
Number of BlockRAM/FIFO	45	148	30%
Number using BlockRAM only	45		
Total primitives used			
Number of 36k BlockRAM used	44		
Number of 18k BlockRAM used	2		
Total Memory used (KB)	1,620	5,328	30%
Number of BUFG/BUFGCTRLs	13	32	40%
Number used as BUFGs	13		

Figure 4.14 Logic Resource Utilization of Circular Sign Detection System

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In this thesis, detection of traffic signs using FPGA was studied. For detection, two discriminating feature; namely color and shape of the signs are used. The joint implementation of shape and color based algorithms is utilized to improve the performance of detection process. Algorithms implemented on FPGA are tested with video streams and images.

For color based algorithms HSV space is not preferred, because of the computational complexity. RGB color domain is used for FPGA implementation. From the experimental results given in chapter 4, it is understood that RGB color space is convenient for color segmentation for well illuminated images. In addition, RGB color space gives satisfactory results for poorly illuminated and shadowed images.

For edge detection, Sobel edge detection algorithm is implemented, because of its low computational complexity and good edge detection capability in noisy conditions.

For circular shape detection radial symmetry detector is used because of its robustness against rotation and high detection rates. But these algorithms need

recursive computations which are hard to implement on FPGA. Therefore, some approximations are taken into consideration during the FPGA implementation of these algorithms. Because of these approximations, performance rate decrease according to the MATLAB simulations.

The output result of this joint implementation has a critical significance in whole system. For well illuminated images, detection rate is higher than %90. For poorly and shadowed images, detection rate is about %70 with optimized parameter values. There are false positive values because of the approximations taken into consideration in the implementation of FPGA. First of all, during the implementation of circular sign detection algorithm, only one radius value is used instead of set of radii. Also, the radial strictness parameter(α) is chosen as 1 in order to simplify the calculations. Taking the radial strictness parameter α as a lower value, causes to increase of non-radially symmetric features. These are summarized as the main reasons of false positives values.

The implemented algorithms also yield low delay in computation on FPGA. Color segmentation and edge detection algorithm on FPGA causes 107 μs delay. Circular shape detection algorithm on FPGA uses one field (odd or even) of a video frame and causes 4.14 ms delay. Since the frame rate is 25 frame/sec (Δt= 40msec), all the computation is completed within a frame time. There are 20 ms between two consecutive fields. After 4.14 ms, the circular shape detection algorithm catches up with the next coming field. Therefore, the algorithm causes one field long delay.

Color Segmentation, edge detection and circular sign detection algorithms uses low resources of FPGA. However, implementation of triangular and rectangular sign detection algorithms increases the resource usage of FPGA. Still, max resource usage of FPGA can be limited with %15. This gives the opportunity to make the implementation on a cheaper FPGA device. Therefore, traffic sign detection system

can be used as a standalone system or used with another system to improve vehicle's safety.

5.2 Future Work

In order to detect traffic signs except circular traffic signs, other shape detection algorithms can be implemented on FPGA.

In addition, the performance of circular traffic sign detection system can be improved with some additional modules. The parameters can be defined using image statistics. Gathering information about the brightness of the signs gives the opportunity to adjust optimum parameters to detect circular signs.

During the hardware implementation of the algorithms, processor is not used. In order to increase the detection rate, circular shape detection algorithm can be implemented on processor. However, this operation causes more delay.

As another future work, detection process can be combined with the recognition process to identify the traffic signs.

REFERENCES

[1] M. A. Garcia-Garrido, M. A. Sotelo, E. Mart´ın-Gorostiza, “Fast Traffic Sign Detection and Recognition Under Changing Lighting Conditions” Proceedings of the IEEE ITSC 2006. 2006 IEEE Intelligent Transportation Systems Conference Toronto, Canada, September 17-20, 2006.

[2] N. Barnes, A. Zelinsky, “Real-time radial symmetry for speed sign detection” 2004 IEEE Intelligent Vehicle Symposium University of Parma, Italy. June 14-17 2004

[3] G. Piccioli, E. De Michelli, P. Parodi, M. Campani, “A Robust Method for Road Sign Detection and Recognition,” Image and Vision Computing, vol.14, pp.209-223, 1996.

[4] A. De la Escalera, L. E. Moreno, M. A. Salichs, J. M. Armigol, “Road Traffic Sign Detection and Classification” Industrial Electronics, IEEE Transactions on, Vol. 44, No. 6., pp. 848-859, 1997.

[5] N. Kehtarnavaz, A. Ahmad, “Traffic Sign Recognition in Noisy Outdoor Scenes,” Proceedings of the Intelligent Vehicles '95 Symposium, pp. 460-465, Sept. 1995.

[6] T. P. Cao, G. Deng, “Real-Time Vision-based Stop Sign Detection System on FPGA”, Digital Image Computing: Techniques and Applications, pp:465-471, 2008

[7] L. Priese, R. Lakmann, V. Rehrmann, “Ideogram Identification in a Realtime Traffic Sign Recognition System,” In Proc. Intelligent Vehicles Symp.pp. 310-314, 1995.

[8] H.S. Neoh, A. Hazanchuk. “Adaptive Edge Detection for Real-Time Video Processing using FPGAs”, Global Signal Processing-Citeseer, 2004

[9] T. A. Abbasi, M. U. Abbasi, “A Proposed FPGA Based Architecture for Sobel Edge Detection Operator”, J. of Active and Passive Electronic Devices, Vol. 2, pp. 271–277, Old City Publishing, Inc., 2007

[10] S. EsTable, J. Schick, F. Stein, R. Janssen, R. Ott, W. Ritter, Y. J. Zheng, “A Real Time Traffic Sign Recognition System," in proc. Intelligent Vehicles'94, 1994.

[11] L. Estevez, N. Kehtarnavaz, “A Real-Time Histographic Approach to Road Sign Recognition,” Proceedings of the IEEE Southwest Symposium on Image Analysis and Interpretation, pp. 95-100, 1996.

[12] “Field Programmable Gate Arrays (FPGA)”, [Online] Available: http://www.globalspec.com/reference/3203/Field-Programmable-Gate-Arrays-FPGA, Last accessed date: 08/01/2010

[13] C.Y. Fang, S. W. Chen, C. S. Fuh, “Road-Sign Detection and Tracking,” Vehicular Technology, IEEE Transactions on Volume 52, Issue 5, pp. 1329-1341, Sept. 2003.

[14] Y.B. Damavandi, K. Mohammadi, “Speed Limit Traffic Sign Detection & Recognition” Proceedings of the 2004 IEEE. Conference on Cybernetics an Intelligent Systems Singapore, 1-3 December, 2004.

[15] A. de la Escalera, J. Armingol, and M. Mata, “Traffic Sign Recognition and Analysis for Intelligent Vehicles” lmage and Vision Comput., Vol. 21, pp. 247-258, 2003.

[16] A. de la Escalera, J. M. Armingol, M. A. Salichs, “Traffic Sign Detection for Driver Support Systems” International Conference on Field and Service Robotics, 2001.

[17] H. Kamada and M. Yoshida, “A Visual Control System Using Image Processing and Fuzzy Theory,” in Vision Based Vehicle Guidance, I. Masaki, Ed. Berlin, Germany: Springer-Verlag, 1992, pp. 111–128.

[18] S. Kantawong, “Road Traffic Signs Detection and Classification for Blind Man Navigation System” International Conference on Control, Automation and Systems 2007 Oct. 17-20, 2007 in COEX, Seoul, Korea

[19] R. Janssen, W. Ritter, F. Stein, and S. Ott. “Hybrid Approach for Traffic Sign Recognition,” In Proc. of Intelligent Vehicles Conference, pp 390-395,1993.

[20] G. Wu, W. Liu, X. Xie, Q. Wei, “A Shape Detection Method Based On The Radial Symmetry Nature and Direction-Discriminated Voting” 1-4244-1437-7/07 IEEE. ICIP 2007

[21] J. Miura, T. Kanda, and Y. Shirai, “An active vision system for real-time traffic sign recognition” In Proc. IEEE Conf. on Intelligent Transportation Systems (ITS), pages 52–57, Dearborn, MI, 2000.

[22] Y.Y. Nguwi, A.Z. Kouzani, “A Study on Automatic Recognition of Road Signs” 1-4244-0023-6/06 IEEE CIS 2006

[23] Philips Semiconductors 9-bit video input processor SAF7113H R21/02/pp81 Document order number: 9397 750 12902 Date of release: 22 Mar 2004

[24] Analog Devices Chip Scale PAL/NTSC Video Encoder with Advanced Power Management ADV7174/ADV7179 datasheet C02980–0–2/04(A),2004

[25] V. Andrey, K. H. Jo, “Automatic Detection and Recognition of Traffic Signs using Geometric Structure Analysis” SICE-ICASE International Joint Conference 2006 Oct. 18-21, 2006 in Bexco, Busan Korea

[26] H. S. Neoh, A. Hazanchuk, “Adaptive Edge Detection for Real-Time Video Processing using FPGAs”, ALTERA,2005”

[27] T. A. Abbasi, M. U. Abbasi, “A Proposed FPGA Based Architecture for Sobel Edge Detection Operator”, Active and Passive Electronic Devices, Vol. 2, pp. 271–277, 2007

[28] A. Trost, B. Zajc, “Design of Real-Time Edge Detection Circuits on Multi-FPGA Prototyping System” ELECO 99 International Conference on Electrical and Electronics Engineering, 1999

[29] G. Loy, A. Zelinsky, “Fast Radial Symmetry for Detecting Points of Interest” IEEE Transactions on Pattern Analysis and Machine Intelligence” Vol. 25, No. 8, August 2003

[30] H. Fleyeh, “Traffic Sign Recognition By Fuzzy Sets” Inteliigent Vehicles Syposium, 2008 IEEE, pp.422-427, 4-6 June 2008

[31] P. L. Rosin, “Measuring Shape: Ellipticity, Rectangularity, and Triangularity”, Cardiff University, Machine Vision and Applications Journal, pp.172-184, July 2003

[32] A. Broggi, P. Cerri, P. Medici, P.P. Porta, “Real Time Road Signs Recognition”, Proceedings of the 2007 IEEE Intelligent Vehicles Syposium Istanbul, Turkey, June 13-15, 2007

[33]“Road Sign Recognition Survey” [On-line] Available: http://euler.fd.cvut.cz/research/rs2/files/skoda-rs-survey.html, last accessed date: 02/01/2010

[34] T. P. Cao, G. Deng, “Real Time Vision-based Stop Sign Detection System on FPGA” 978-0-7695-3456-5/08, 2008 IEEE Computer Society

[35] “BT.656_Video_Interface_for_ICs”, Application Note AN9728.2, Intersil Americas Inc., 2002.

[36]Alcalá de Henares (Madrid, Spain) Universidad de Alcalá http://roadanalysis.uah.es, last accessed date: 03/11/2010

[37] R. Jain, R. Kasturi, B.G. Schunck, “Machine Vision”, McGraw-Hill, Inc., 1995

[38] Emre Ulay, “Color and Shape Based Traffic Sign Detection”, MS Thesis, METU, Nov. 2008.

[39] D. Ghica, S.W. Lu, X. Yuan “Recognition of Traffic Signs by Artificial Neural Network”, 1995 IEEE International Conf. on Neural Networks, Perth, Australia, vol.3, pp.1444-1449, Nov. 1995.

[40] M. Bülent Havur, “Traffic Sign Recognition for Unmanned Vehicle Control”, MS Thesis, METU, Nov. 2006.

[41] M. Frerking, “Digital Signal Processing In Communications Systems”, Springer, 1994

APPENDIX A

STRUCTURE OF FPGA AND DESIGN FLOW

In this appendix general information about FPGAs and its structures is explained.

A.1 General Informations

Field Programmable Gate Arrays (FPGA) are a new computing architecture that is under development since the early '90s. FPGAs contain as many as tens of thousands of logic cells and an even greater number of flip-flops. FPGAs are used in applications ranging from data processing and storage, to instrumentation, telecommunications, and digital signal processing. Other terms for FPGA include logic cell array (LCAs) and programmable application-specific integrated chip (pASIC).

Field-programmable gate arrays are available with different numbers of system gates, shift registers, logic cells, and look up tables. Logic blocks or logic cells (LCs) do not include I/O blocks, but generally contain a look up table to generate any function of inputs, a clocked latch (flip-flop) to provide registered outputs, and control logic circuits for configuration purposes. Logic cells are also known as logic array blocks (LABs), logic elements (LEs) and configurable logic blocks (CLBs). Look up tables (LUTs) or truth tables are used to implement a single logic function by storing the correct output logic state in a memory location that corresponds to each particular combination of input variables.

FPGA processors change the view on algorithmic problem solving and have the advantage of being extremely powerful for many applications. Widely used computer architectures have a fixed central processing unit (CPU) operating on data stored in a memory. Programs determine the sequence of single instructions executed by the CPU. This is a disadvantage for algorithms that can be executed in parallel. In contrast, FPGA computers have no given processor structure but offer large amounts of logic gates, registers, RAM, and routing resources. These can be used for performing logical and arithmetical operations, for variable storage, and to transfer data between different parts of the system. Programs do not determine the sequence of execution but the logical structure of the reconfigurable machine. Therefore, algorithms are not only executable in parallel but are executed using a minimum amount of hardware. Typically, thousands of operations can be performed in parallel on an FPGA computer during every clock cycle. [12]

A.2 Structure

A 2.1 Configurable Logic Block (CLBs)

The CLB is the basic logic unit in an FPGA. Exact numbers and features vary from device to device, but every CLB consists of a configurable switch matrix with 4 or 6 inputs, some selection circuitry (MUX, etc), and flip-flops as seen in Figure A-1. The switch matrix is highly flexible and can be configured to handle combinatorial logic, shift registers, or RAM.

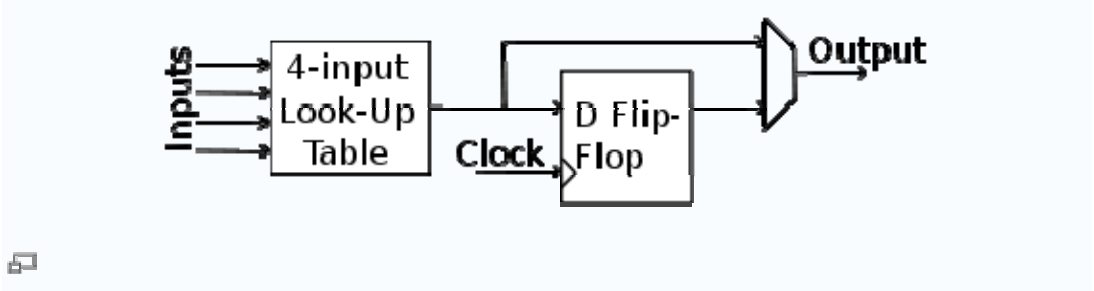

Figure A.1 Typical logic block

A 2.2 Memory

Embedded Block RAM memory is available in most FPGAs, which allows for on-chip memory in your design. These allow for on-chip memory for your design. Xilinx FPGAs provide up to 10 Mbits of on-chip memory in 36 kbit blocks that can support true dual-port operation.

A 2.3 Complete Clock Management

Digital clock management is provided by most FPGAs in the industry (all Xilinx FPGAs have this feature). The most advanced FPGAs from Xilinx offer both digital clock management and phase-looped locking that provide precision clock synthesis combined with jitter reduction and filtering.

A 2.4 IOB Details

Today´s FPGAs provide support for dozens of I/O standards thus providing the ideal interface bridge in your system. I/O in FPGAs is grouped in banks as shown in Figure A.2 with each bank independently able to support different I/O standards. Today´s leading FPGAs provide over a dozen I/O banks, thus allowing flexibility in I/O support.

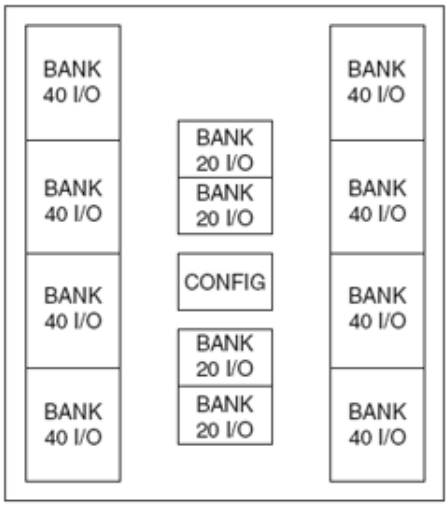

Figure A.2 FPGA I/O Banks

A.3 FPGA Design Flow

A simplified version of design flow is given in the Figure A.3.

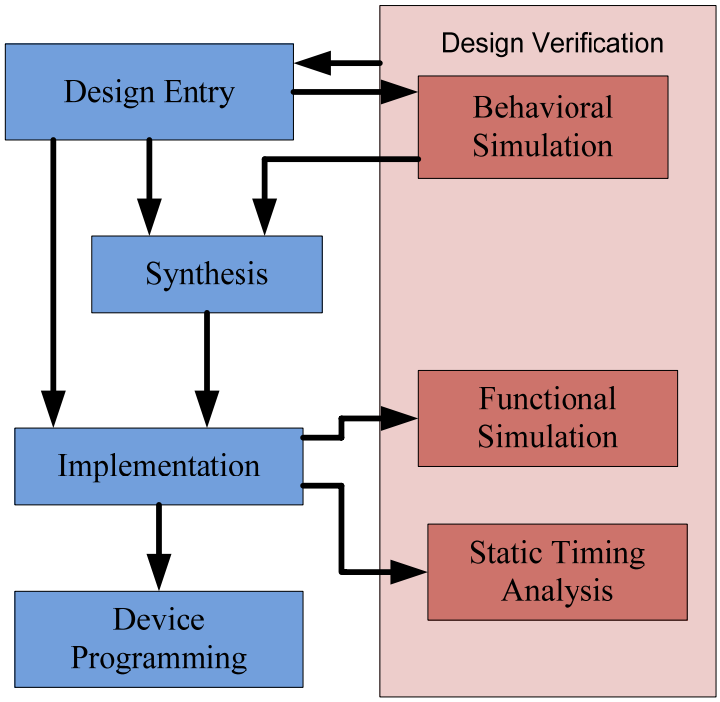

Figure A.3 Design Flow

A 3.1 Design Entry

There are different techniques for design entry. Schematic based, Hardware Description Language and combination of both. Selection of a method depends on the design and designer. If the designer wants to deal more with Hardware, then Schematic entry is the better choice. When the design is complex or the designer thinks the design in an algorithmic way then HDL is the better choice. Language based entry is faster but lag in performance and density. HDLs represent a level of abstraction that can isolate the designers from the details of the hardware

implementation. Schematic based entry gives designers much more visibility into the hardware. It is the better choice for those who are hardware oriented.

A 3.2 Synthesis

Synthesis process translates VHDL or Verilog code into a device netlist formate. i.e a complete circuit with logical elements(gates, flip flops, etc…) for the design. If the design contains more than one sub designs, ex. to implement a processor, we need a CPU as one design element and RAM as another and so on, then the synthesis process generates netlist for each design element. Synthesis process will check code syntax and analyze the hierarchy of the design which ensures that the design is optimized for the design architecture, the designer has selected. The resulting netlist(s) is saved to an NGC(Native Generic Circuit) file (for Xilinx® Synthesis Technology (XST)).

A 3.3 Implementation

This process consists a sequence of three steps
1.Translate
2.Map
3. Place and Route

Translate process combines all the input netlists and constraints to a logic design file. This information is saved as a NGD (Native Generic Database) file. This can be done using NGD Build program. Here, defining constraints is nothing but, assigning the ports in the design to the physical elements (ex. pins, switches, buttons etc) of the targeted device and specifying time requirements of the design. This information is stored in a file named UCF (User Constraints File).

Tools used to create or modify the UCF are PACE, Constraint Editor etc

Map process divides the whole circuit with logical elements into sub blocks such that they can be fit into the FPGA logic blocks. That means map process fits the

logic defined by the NGD file into the targeted FPGA elements (Combinational Logic Blocks (CLB), Input Output Blocks (IOB)) and generates an NCD (Native Circuit Description) file which physically represents the design mapped to the components of FPGA. MAP program is used for this purpose.

Place and Route PAR program is used for this process. The place and route process places the sub blocks from the map process into logic blocks according to the constraints and connects the logic blocks. For example; if a sub block is placed in a logic block which is very near to IO pin, then it may save the time but it may affect some other constraint. So tradeoff between all the constraints is taken account by the place and route process. The PAR tool takes the mapped NCD file as input and produces a completely routed NCD file as output. Output NCD file consists the routing information.

A 3.4 Device Programming

Now the design must be loaded on the FPGA. But the design must be converted to a format so that the FPGA can accept it. BITGEN program deals with the conversion. The routed NCD file is then given to the BITGEN program to generate a bit stream (a .BIT file) which can be used to configure the target FPGA device. This can be done using a cable. Selection of cable depends on the design.

APPENDIX B

HARDWARE DESIGN CONSIDERATIONS

In this appendix, hardware design considerations will be introduced.

B.1 ML507 Development Board

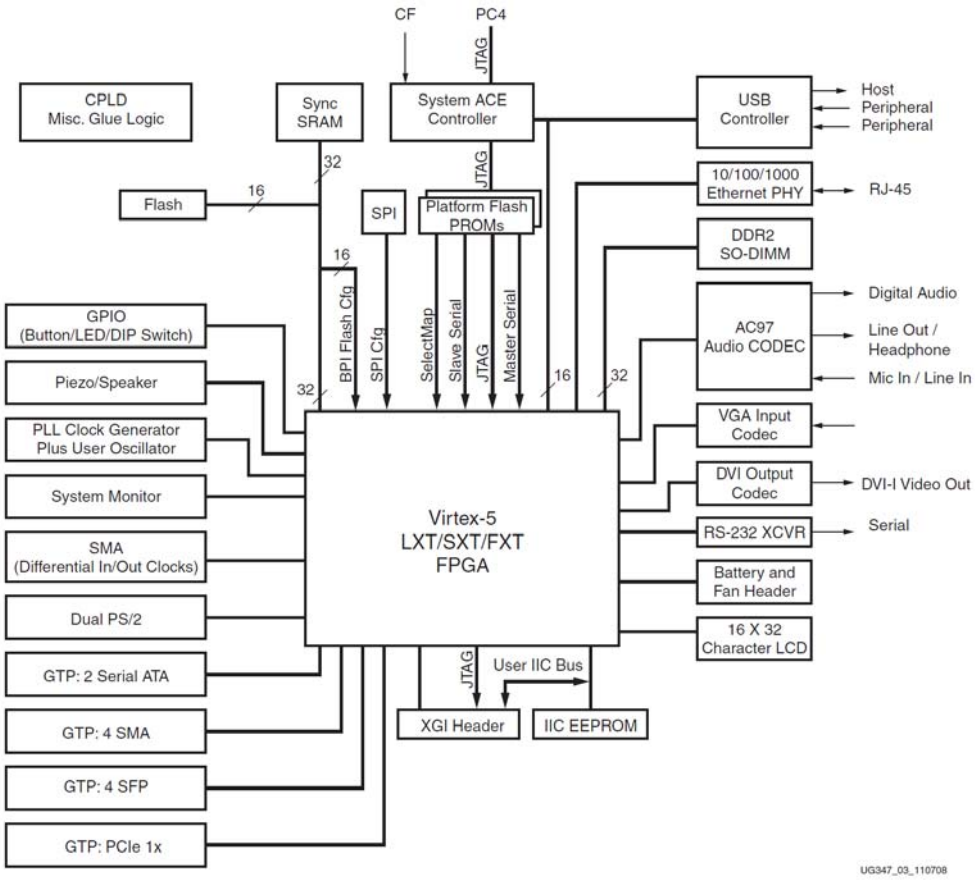

Figure B.1 Virtex-5 FPGA ML50x Evaluation Platform Block Diagram

General properties of ML507 FPGA board are given at the below;

- Xilinx Virtex-5 FPGA ML507 (XC5VFX70T-1FFG1136)
- Two Xilinx XCF32P Platform Flash PROMs (32 Mb each) for storing large device configurations
- Xilinx System ACE™ CompactFlash configuration controller with Type I CompactFlash connector
- 64-bit wide, 256-MB DDR2 small outline DIMM (SODIMM), compatible with EDK supported IP and software drivers
- Clocking
- Programmable system clock generator chip
- One open 3.3V clock oscillator socket
- External clocking via SMAs (two differential pairs)
- General purpose DIP switches (8), LEDs (8), pushbuttons, and rotary encoder
- 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 I2C bus expansion
- Stereo AC97 audio codec with line-in, line-out, 50-mW headphone, microphone-in jacks, SPDIF digital audio jacks, and piezo audio transducer
- RS-232 serial port, DB9 and header for second serial port
- One 8-Kb I2C EEPROM and other I2C capable devices
- PS/2 mouse and keyboard connectors
- ZBT synchronous SRAM, 9 Mb on 32-bit data bus with four parity bits
- JTAG configuration port for use with Parallel Cable III, Parallel Cable IV, or Platform
- USB download cable
- 5V @ 6A AC adapter

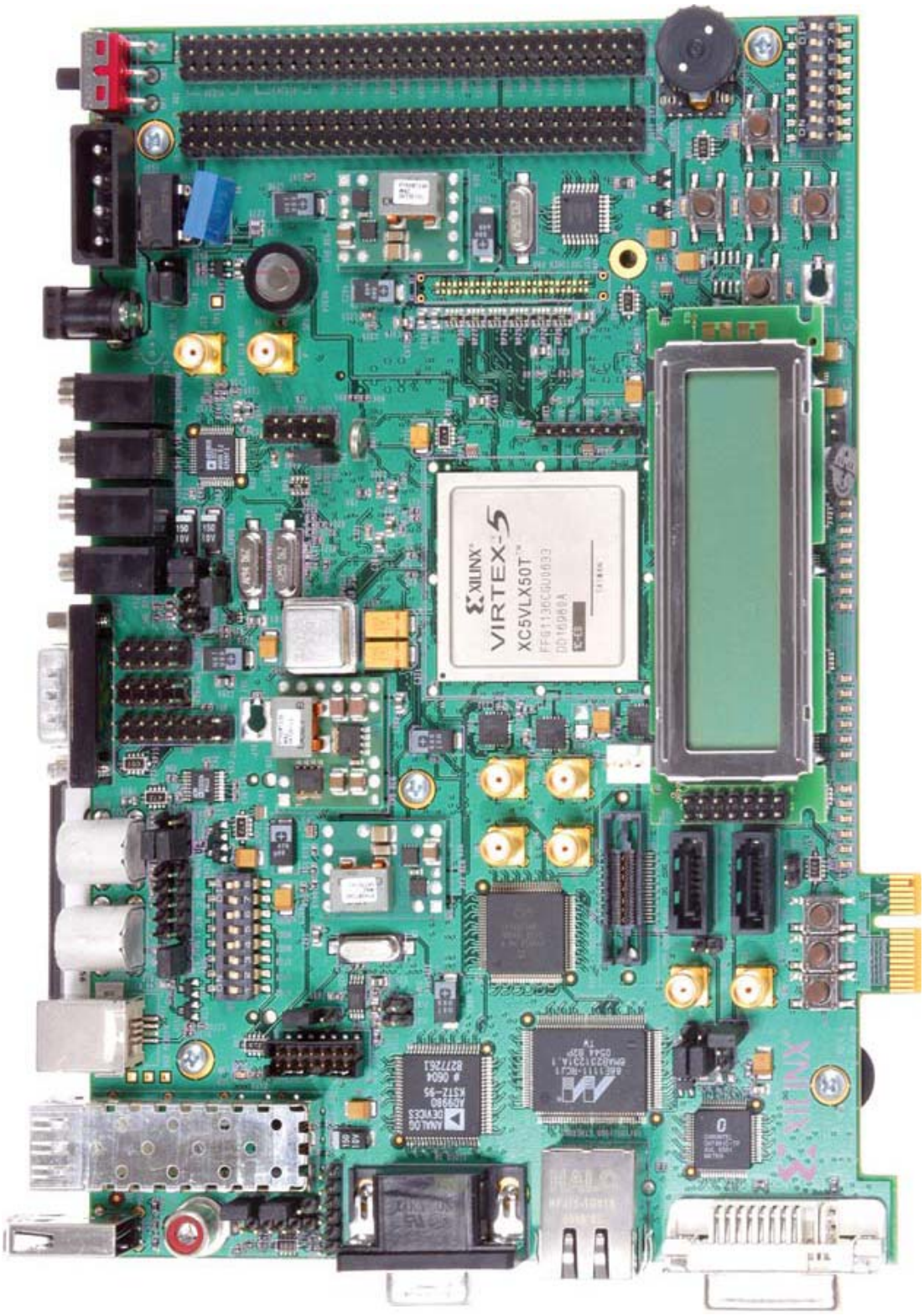

Figure B.2 ML507 development board

B.2 Video I/O Interface Card

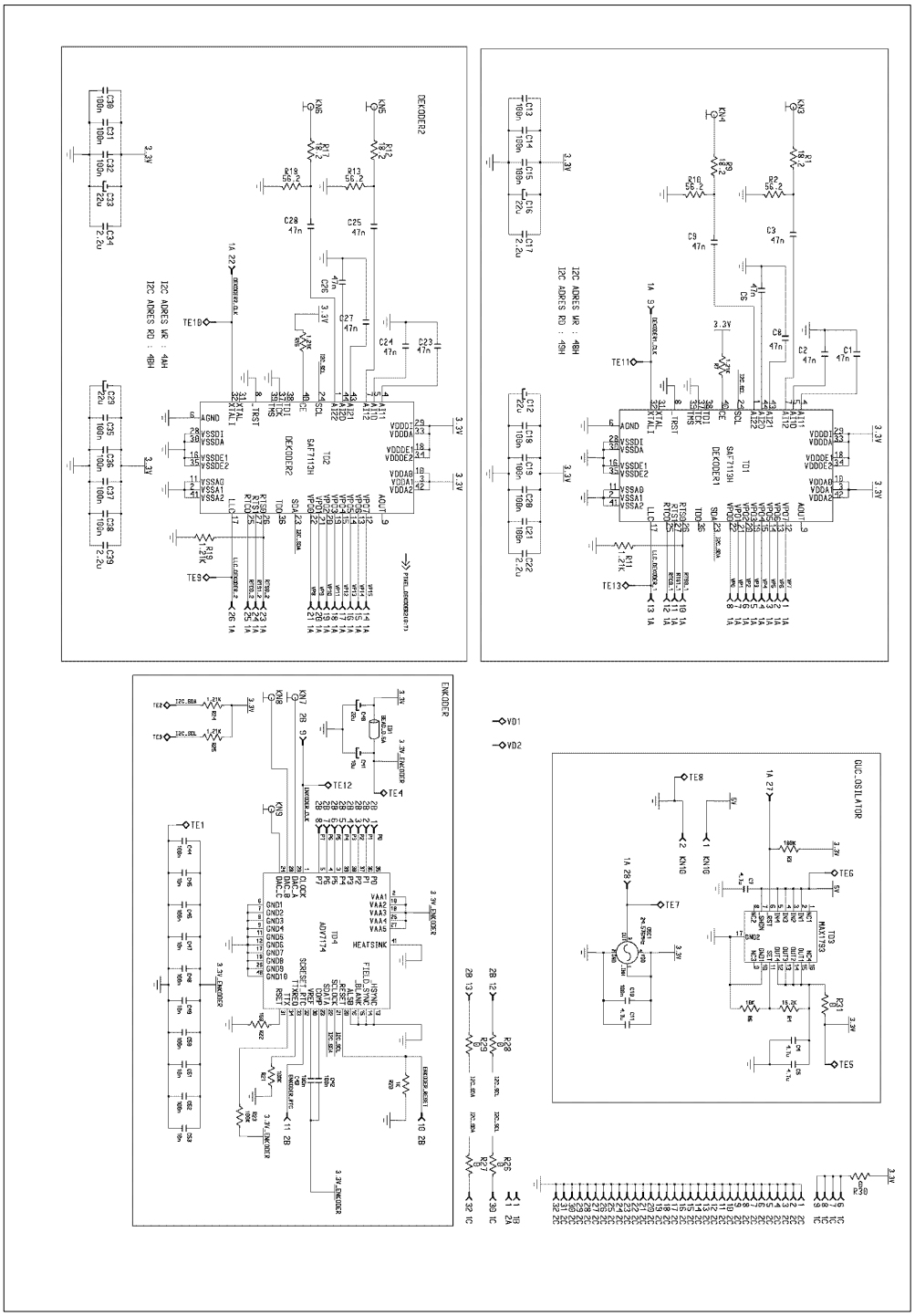

Figure B.3 Video Interface I/O Card

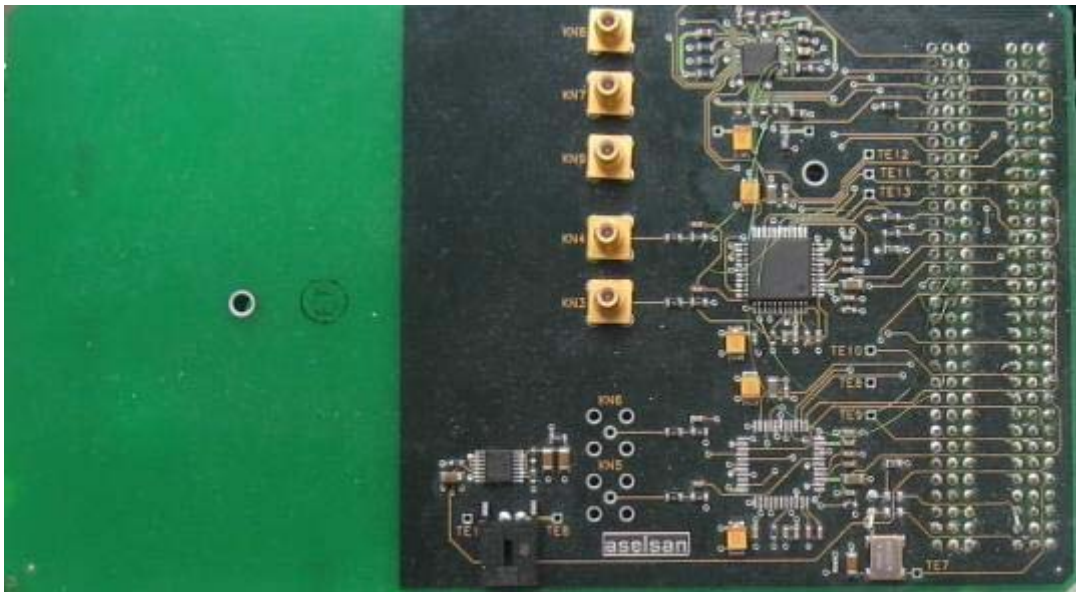

Figure B.4. Video Interface I/O Card Front Side

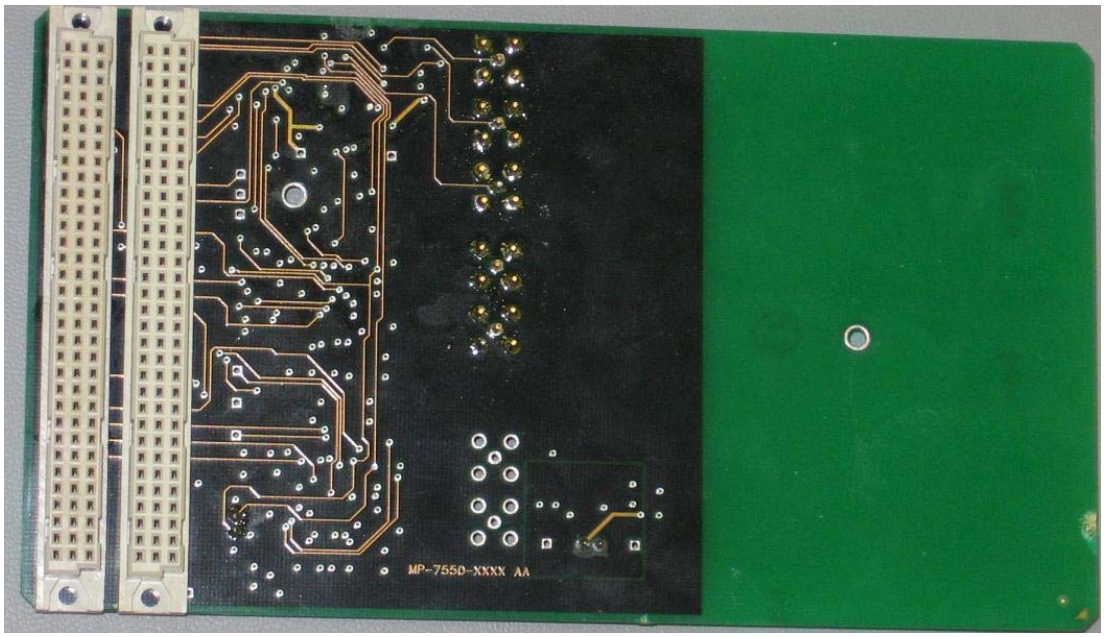

Figure B.5. Video Interface I/O Card Back Side

B.3 Programming Device

During the FPGA design, Xilinx ISE is used as development environment. In addition, XST is used for synthesis, ChipScope for on-chip debugging and IMPACT for programming the FPGA. VHDL is selected as coding language.

Xilinx Program Cable USB is used to program memory devices on board via the FPGA JTAG PORT. Xilinx IMPACT tool controls this device. Platform Cable USB is shown in Figure B.6.

Figure B.6. Xilinx Platform Cable USB