

FPGA IMPLEMENTATION OF LICENSE PLATE DETECTION AND RECOGNITION

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

BY  
SERAP SARIKAVAK

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

SEPTEMBER 2013



Approval of the thesis:

**FPGA IMPLEMENTATION OF LICENSE PLATE DETECTION  
AND RECOGNITION**

submitted by **SERAP SARIKAVAK** 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. Gönül Turhan Sayan  
Head of Department, **Electrical and Electronics Engineering** \_\_\_\_\_

Assoc. Prof. Dr. Mehmet Mete Bulut  
Supervisor, **Electrical and Electronics Engineering Dept., METU** \_\_\_\_\_

Prof. Dr. Gözde Bozdağı Akar  
Co-Supervisor, **Electrical and Electronics Engineering Dept., METU** \_\_\_\_\_

**Examining Committee Members:**

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 \_\_\_\_\_

Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı  
Electrical and Electronics Engineering Dept., METU \_\_\_\_\_

Hasan Irmak, M.Sc.  
REHİS, ASELSAN \_\_\_\_\_

**Date : 05.09.2013**

**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 : Serap SARIKAVAK

Signature :

## **ABSTRACT**

# **FPGA IMPLEMENTATION OF LICENSE PLATE DETECTION AND RECOGNITION**

Sarıkavak, Serap

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

September 2013, 143 pages

In this thesis, license plate detection and recognition system based on Gabor approach is proposed and recognition part of the system is implemented on the FPGA platform. The purpose of this project is to develop a system that extracts license plate region from image, taken from proposed distance, and recognizes the characters on this region. For this project, techniques on the literature are investigated and some of them are implemented.

In the localization of the plate region, color space conversion, image enhancement, binarization, connected component labeling, Gabor filters and morphological operations are used. To segment the characters on the plate region, notch detection algorithm which uses the vertical projection of the plate region and morphological operations are used. Finally, characters are recognized by the feature-based matching algorithm.

Character segmentation and character recognition parts of the proposed algorithm are implemented in the Altera Cyclone IV E FPGA which has an embedded soft core processor. To implement the algorithm, FPGA logic collaborates with the software application of the processor.

**Keywords:** License Plate Recognition, Plate Region Detection, Character Segmentation, Character Recognition, Gabor Filters, FPGA, Embedded Processor

## ÖZ

# FPGA DONANIMI ÜZERİNDE ARAÇ PLAKASI ALGILAMA VE TANIMA

Sarıkavak, Serap

Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü  
Tez Yöneticisi: Doç. Dr. Mehmet Mete Bulut  
Ortak Tez Yöneticisi: Prof. Dr. Gözde Bozdağı Akar

Eylül 2013, 143 sayfa

Bu tezde, Gabor yaklaşımı temelli araç plakası algılama ve tanıma sistemi önerilmiş ve tanıma algoritması FPGA üzerinde gerçekleştirilmiştir. Bu çalışmanın amacı belirli bir mesafeden çekilmiş araç görüntüsünden plaka bölgesini algılayan ve bu bölgedeki karakterleri tanıyan bir sistem geliştirmektir. Bu çalışmada, literatürde kullanılan metotlar incelenmiş ve bu metotların bazıları önerilen sistemde kullanılmıştır.

Araç plakasının yerini saptarken, renk uzayı çevrimi, görüntü geliştirme, ikili resim elde etme, bağıntılı bileşen etiketleme, Gabor filtreleri ve morfolojik işlemler kullanılmıştır. Plaka bölgesindeki karakterleri ayırmak için, plaka bölgesinin dikey izdüşümünü kullanan girinti algılama algoritması ve morfolojik işlemler kullanılmıştır. Son olarak, karakterler özellik tabanlı karşılaştırma algoritması kullanılarak tanınmıştır.

Önerilen algoritmanın karakter ayırma ve karakter tanıma kısımları, içinde sabit olmayan gömülü işlemci olan Altera Cyclone IV E FPGA’inde gerçekleştirilmiştir. Algoritmayı gerçekleştirmek için FPGA mantık kapıları işlemcinin uygulama yazılımı ile işbirliği yapmaktadır.

Anahtar Kelimeler: Araç Plakası Tanıma, Plaka Bölgesi Algılama, Karakter Ayırıştırma, Karakter Tanıma, Gabor Filtreleri, FPGA, Gömülü İşlemci

*To My Family...*

## **ACKNOWLEDGEMENTS**

First of all, I would like to express my deepest gratitude to my supervisors Assoc. Prof. Dr. Mehmet Mete Bulut and Prof. Dr. Gözde Bozdağı Akar for their valuable guidance, advice, encouragements and insight throughout the research.

I would like to thank ASELSAN Inc. for the support given throughout this study.

I would like to thank my sister, Hülya, for her endless support and encouragement which helped me thinking positively in any circumstances.

I would like to express my special appreciation to my family for their continuous support and patience over the years. This thesis is dedicated to them.



# TABLE OF CONTENTS

<b>ABSTRACT</b> .....	<b>v</b>
<b>ÖZ</b> .....	<b>vi</b>
<b>ACKNOWLEDGEMENTS</b> .....	<b>viii</b>
<b>TABLE OF CONTENTS</b> .....	<b>ix</b>
<b>LIST OF TABLES</b> .....	<b>xiii</b>
<b>LIST OF FIGURES</b> .....	<b>xiv</b>
<b>CHAPTERS</b>	
<b>1. INTRODUCTION</b> .....	<b>1</b>
1.1 GENERAL.....	1
1.2 SCOPE OF THE THESIS .....	2
1.3 OUTLINE OF THE THESIS.....	3
<b>2. LITERATURE OF LICENSE PLATE DETECTION AND RECOGNITION</b> ....	<b>5</b>
2.1 EXTRACTION OF PLATE REGION.....	5
2.2 SEGMENTATION OF CHARACTERS .....	8
2.3 RECOGNITION OF PLATE CHARACTERS .....	11
<b>3. PROPOSED SYSTEM</b> .....	<b>15</b>
3.1 PLATE REGION DETECTION.....	17
3.1.1 <i>IMAGE ENHANCEMENT</i> .....	17
3.1.1.1 GAUSSIAN FILTER.....	17
3.1.1.2 CONTRAST ENHANCEMENT.....	18
3.1.2 <i>GABOR WAVELET TRANSFORM</i> .....	19
3.1.3 <i>OTSU THRESHOLDING and MORPHOLOGICAL OPERATIONS</i> .....	22
3.1.4 <i>IDENTIFICATION OF PLATE REGION</i> .....	25
3.1.4.1 FILTERING WITH PLATE LENGTH and WIDTH PARAMETERS ..	26
3.1.4.2 IDENTIFICATION OF EXACT PLATE REGION.....	31
3.1.5 <i>HORIZONTAL and VERTICAL TRIMMING</i> .....	33
3.2 CHARACTER SEGMENTATION .....	36
3.2.1 <i>NOTCH DETECTION and SEGMENTATION OF EACH CHARACTER</i> ....	36

3.2.2 MORPHOLOGICAL OPERATION.....	38
3.3 CHARACTER RECOGNITION .....	38
3.3.1 CODEWORD GENERATION.....	38
3.3.2 CODEWORD MATCHING and RECOGNITION.....	40
3.3.2.1 IDENTIFICATION OF LETTER AND NUMBER.....	42
3.3.2.2 IDENTIFICATION OF 0, 4 AND 7.....	42
3.3.2.3 IDENTIFICATION OF U, V AND Y .....	43
3.3.2.4 IDENTIFICATION OF 3, 5 AND 8.....	44
3.3.2.5 IDENTIFICATION OF 3, 8 AND 9.....	46
3.3.2.6 IDENTIFICATION OF 6 AND B .....	47
3.3.2.7 IDENTIFICATION OF 2 AND Z .....	48
<b>4. RECOGNITION OF LICENSE PLATE USING FPGA HARDWARE.....</b>	<b>51</b>
4.1 HARDWARE ARCHITECTURE .....	51
4.1.1 COMPONENTS USED IN THE SYSTEM .....	52
4.1.1.1 CYCLONE IV E FPGA.....	52
4.1.1.2 EPCS16 SERIAL FLASH AS CONFIGURATION DEVICE .....	53
4.1.1.3 SDRAM .....	53
4.1.1.4 BUTTONS and LEDS .....	53
4.1.1.5 USER I/Os.....	53
4.1.2 BLOCK DIAGRAM OF THE HARDWARE ARCHITECTURE.....	54
4.2 FPGA ARCHITECTURE .....	54
4.2.1 FPGA BLOCKS.....	55
4.2.1.1 PLL.....	56
4.2.1.2 RESET GENERATOR.....	57
4.2.1.3 LED INDICATOR .....	58
4.2.1.4 DEBOUNCE FILTER .....	59
4.2.1.5 QSYS SYSTEM LOGIC .....	60
4.2.1.5.1 CLOCK AND RESET SOURCE .....	62
4.2.1.5.2 NIOS II PROCESSOR.....	63
4.2.1.5.3 INTERVAL TIMER .....	64
4.2.1.5.4 SYSTEM ID .....	64
4.2.1.5.5 PIO.....	64

4.2.1.5.6 EPCS CONTROLLER .....	64
4.2.1.5.7 SDRAM CONTROLLER.....	65
4.2.1.5.8 TIGHTLY COUPLED INSTRUCTION MEMORY .....	65
4.2.1.5.9 ON-CHIP RAM .....	65
4.2.1.5.10 JTAG UART .....	65
4.2.1.5.11 UART INTERFACE .....	66
4.2.1.5.12 IMAGE MANAGE BLOCK .....	66
4.2.1.5.13 CODEWORD GENERATION BLOCK.....	73
<b>4.2.2 SOFTWARE APPLICATION FOR EMBEDDED PROCESSOR .....</b>	<b>78</b>
4.2.2.1 UP and DOWN SIDE TRIMMING.....	82
4.2.2.2 CHARACTER SEGMENTATION .....	84
4.2.2.3 CHARACTER RECOGNITION.....	87
4.2.2.3.1 CODEWORD GENERATION.....	87
4.2.2.3.2 CODEWORD MATCHING AND RECOGNITION.....	91
4.2.2.3.2.1 IDENTIFICATION ALGORITHMS .....	93
4.2.2.3.2.1.1 IDENTIFICATION OF LETTER AND NUMBER .....	93
4.2.2.3.2.1.2 IDENTIFICATION OF 0, 4 AND 7.....	94
4.2.2.3.2.1.3 IDENTIFICATION OF U, V AND Y .....	95
4.2.2.3.2.1.4 IDENTIFICATION OF 3, 5 AND 8.....	95
4.2.2.3.2.1.5 IDENTIFICATION OF 3, 8 AND 9.....	96
<b>5. EXPERIMENTAL RESULTS.....</b>	<b>97</b>
5.1 TEST RESULTS FOR THE MATLAB IMPLEMENTATION .....	97
5.2 TEST RESULTS FOR THE FPGA IMPLEMENTATION .....	111
5.3 COMPARISON WITH THE STATE OF THE ART .....	119
5.4 RESOURCE UTILIZATION .....	121
5.5 EXECUTION TIMES .....	123
<b>6. CONCLUSIONS AND FUTURE WORK.....</b>	<b>125</b>
6.1 CONCLUSIONS.....	125
6.2 FUTURE WORK.....	127
<b>REFERENCES .....</b>	<b>129</b>
<b>APPENDICES</b>	
<b>A. NIOS II PROCESSOR.....</b>	<b>133</b>

<b>B. RS232 EXTENDER CIRCUIT .....</b>	<b>135</b>
<b>C. GUI INTERFACE OF THE LPDRS.....</b>	<b>137</b>
<b>D. DATA TABLES .....</b>	<b>139</b>

## LIST OF TABLES

### TABLES

Table 3-1 Results for the Identification of Exact Plate Region Step .....	32
Table 3-2 Decision Table for Identification of ‘0’, ‘4’ and ‘7’ .....	43
Table 3-3 Decision Table for Identification of ‘U’, ‘V’ and ‘Y’ .....	44
Table 3-4 Decision Table for Identification of ‘3’, ‘5’ and ‘8’ .....	46
Table 3-5 Decision Table for Identification of ‘3’, ‘8’ and ‘9’ .....	47
Table 3-6 Decision Table for Identification of ‘6’ and ‘B’ .....	48
Table 3-7 Decision Table for Identification of ‘2’ and ‘Z’ .....	49
Table 4-1 Register Map for the Image Manage Block.....	70
Table 4-2 Register Map for the Codeword Generation Block.....	75
Table 5-1 Test results for the MATLAB Implementation .....	97
Table 5-2 Reasons of the License Plate Extraction Failures .....	105
Table 5-3 Character Recognition Rate for the MATLAB Implementation .....	108
Table 5-4 Test Results for the FPGA Implementation .....	111
Table 5-5 Character Recognition Rate for the FPGA Implementation.....	115
Table 5-6 Characters Represented by Same Codeword Pair.....	116
Table 5-7 Performance of License Plate Recognition .....	118
Table 5-8 Performance Comparison of the Results .....	121
Table 5-9 Resource Utilization of the License Plate Recognition System.....	121
Table 5-10 Execution Times of the LPR System .....	123
Table D-1 Lookup Table for the Characters .....	139
Table D-2 Plate Database.....	142

## LIST OF FIGURES

### FIGURES

Figure 3-1 Flow Diagram of the Proposed System .....	16
Figure 3-2 Gaussian Function .....	17
Figure 3-3 Gaussian Kernel .....	18
Figure 3-4 Images Before and After the Contrast Enhancement .....	19
Figure 3-5 Gabor Wavelet Transform of the Input Image .....	21
Figure 3-6 Most Acceptable Gabor Response of the Input Image .....	22
Figure 3-7 Binarized Image with Otsu Thresholding After Gabor Wavelet Transform .....	23
Figure 3-8 Structuring Element Used For Morphological Closing .....	24
Figure 3-9 Binary Image After Morphological Closing .....	25
Figure 3-10 Flow Diagram of the Identification of Plate Region .....	26
Figure 3-11 Flow Diagram of the Filtering Algorithm.....	28
Figure 3-12 Extracted Plate Regions .....	31
Figure 3-13 Flow Diagram of the Identification of Exact Plate Region.....	32
Figure 3-14 Horizontal Profiling Results.....	34
Figure 3-15 Before and After Horizontal Trimming .....	34
Figure 3-16 Vertical Profiling Results.....	35
Figure 3-17 Before and After Trimming Operation .....	35
Figure 3-18 Vertical Projection of the Plate Region without DC value .....	36
Figure 3-19 Notch Extraction with Vertical Projection.....	37
Figure 3-20 Results for the Segmented Characters .....	38
Figure 3-21 Cross-point and Codeword Calculations .....	40
Figure 3-22 Flow Diagram of the Recognition .....	41
Figure 3-23 Position of row index values for cross-point calculations of “047” .....	43
Figure 3-24 Position of row index values for cross-point calculations of “UVY” .....	44
Figure 3-25 Position of row index values for cross-point calculations of “358” .....	45
Figure 3-26 Position of row index values for cross-point calculations of “389” .....	46
Figure 3-27 Position of row index value for cross-point calculation of “6B” .....	48
Figure 3-28 Position of row index value for cross-point calculation of “2Z” .....	49
Figure 4-1 Top View of the Development Board.....	52
Figure 4-2 Hardware Architecture of the System.....	54
Figure 4-3 FPGA Blocks for the Recognition.....	56
Figure 4-4 PLL Interface .....	56
Figure 4-5 Power Distribution of the Development Board.....	57
Figure 4-6 Reset Generator Interface .....	58
Figure 4-7 Block Diagram for the Reset Generator .....	58
Figure 4-8 Led Indicator Interface .....	58
Figure 4-9 Block Diagram of the Led Indicator .....	59

Figure 4-10 Acceptance of Pulse on the User Buttons .....	59
Figure 4-11 Integrated Components of the Qsys System Logic .....	61
Figure 4-12 Qsys System Logic Interface.....	62
Figure 4-13 Nios II Processor Interface .....	63
Figure 4-14 Block Diagram of the UART Core .....	66
Figure 4-15 Avalon MM Interface Connection of the Image Manage Block .....	67
Figure 4-16 Image Manage Block Interface.....	67
Figure 4-17 Read and Write Transfers of the Avalon MM Interface .....	68
Figure 4-18 Block Diagram of the Image Manage Block .....	69
Figure 4-19 Avalon Process Interface.....	69
Figure 4-20 Avalon Interface State Transitions .....	70
Figure 4-21 FIFO Process Interface .....	71
Figure 4-22 Summation Process Interface .....	72
Figure 4-23 Avalon MM Interface Connection of the Codeword Generation Block .....	73
Figure 4-24 Avalon MM Interface of the Codeword Generation Block.....	74
Figure 4-25 Block Diagram of the Codeword Generation Block.....	75
Figure 4-26 Counter Process Interface .....	77
Figure 4-27 State Machine of the Counter Process .....	77
Figure 4-28 Block Diagram of the Software Application .....	79
Figure 4-29 Flow Diagram of the Up and Down Side Trimming .....	83
Figure 4-30 Flow Diagram of the Character Segmentation .....	86
Figure 4-31 Flow Diagram of the Codeword Generation .....	90
Figure 4-32 Flow Diagram of the Codeword Matching and Recognition .....	92
Figure 5-1 Sample plates and their Gabor responses for which the plates cannot be extracted from the image.....	108
Figure A-1 Nios II Processor Core Block Diagram.....	134
Figure C-1 GUI of LPDRS .....	137





# CHAPTER 1

## INTRODUCTION

### 1.1 GENERAL

In recent years, the use of vehicles has been increasing because of population growth and human needs. Due to growing number of vehicles, traffic problems such as theft, speeding, red light violation etc. increase and control of vehicles is becoming a big problem. As a result of traffic control problem, vehicle tracking, recognition and management has become major topics for the modern traffic control systems. Automatic vehicle identification (AVI) systems are used to provide effective control. AVI system is the essential step for the intelligent traffic systems. There are many applications which make use the AVI systems such as traffic control [21, 27], car park automation [28], highway electronic toll collection, border and custom checkpoints, red light violation enforcement etc.

Technique of radio frequency identification (RFID) [29], infrared, microwave and image recognition are the most common methods which are included in the current vehicle identification systems. RFID, infrared and microwave techniques depend on installation of transponders on the vehicles. For the system using transponders, the rate of the accurate detection and recognition is low when the vehicle is in high speed driving.

AVI systems which use laser or radio frequency to identify the vehicle, utilize a small windshield-mounted tag. A reader is mounted in each lane and identifies the vehicle by reading the ID-tag while the vehicle approaching the barrier [29].

Vehicle license plate recognition has improved for the vehicle monitoring and management in recent years. License plate recognition is a form of automatic vehicle identification. The advantage of the license plate recognition system is that there is no need for the any special owner or driver compliance such as the use of tag because every vehicle on the traffic already has a license plate. However, RFID is more advantageous than license plate recognition in terms of weather conditions. Physical condition of the license plate such as dirt, damages or changed surface affect the license plate recognition whereas RFID is not affected from physical disturbances.

License plate detection and recognition system has mainly four stages; the car image capturing stage, plate region detection stage, the character extraction stage and character recognition stage. In the system, the image of the incoming car is captured by infrared sensors, which can easily detect the passing of the car. Then, in plate region detection and character extraction stages, the locations of a vehicle license plate and the characters are

identified using edge detection [30, 31], feature projection [16, 24], neural networks [32, 33], fuzzy logic [34], genetic algorithms [35] or morphology based technologies. After the characters are extracted, by template matching methods, character recognition stage matches the features of the extracted character with the template pattern in the database and the most similar pattern is determined as a recognized character [21].

Recognition rate, reliability and processing speed are the three important parameters for the AVI systems. In outdoor applications, robustness and real-time operations are the important issues in which many studies emphasized on. For the reliable license plate recognition system, efficient algorithms are required to provide robustness and real-time operations.

## **1.2 SCOPE OF THE THESIS**

In this thesis, license plate detection and recognition for 640x480x24 color images are examined and recognition of the license plate part is implemented in the FPGA.

First, algorithms for the license plate detection and recognition system have been searched and analyzed. Then, Gabor based license plate detection is chosen as the main approach to extract the plate region because of the success of the Gabor filters in many applications, e.g., face detection and recognition, iris recognition and fingerprint matching etc. After the localization of the license plate, algorithm developed for the recognition is implemented on Altera DB\_START\_4CE10-Cyclone IV E Development Board.

In the developed algorithm, there are three main stages, namely, plate region detection, character segmentation and character recognition. In the first stage, position of the candidate plate regions are identified and exact plate region is determined. Extraction of the plate region is based on Gabor Wavelet Transform and morphological operations. With Gabor transform, pixel transition regions are found and with morphological operation, transition regions are merged. By observing the density and the size of the candidate regions, exact plate region is determined. In the second stage, characters in the plate region are segmented. In the final stage, combined codeword for each segmented character is calculated and candidate codeword is compared with the template codewords. If there is a match, candidate character is recognized as the template character under constraint.

Before the FPGA implementation, algorithm is developed and verified in MATLAB. After satisfactory performance is achieved, character segmentation and character recognition parts of the algorithm are implemented in the FPGA. In the FPGA implementation, plate regions extracted with the MATLAB are utilized to test the license plate recognition system. To send the image of plate region and size information to the FPGA, Graphical User Interface (GUI) is designed. The detailed description of the designed GUI is given in Appendix C.

For the test of the license plate detection and recognition system, photographs obtained from the car parks of METU and ASELSAN are utilized. Image dataset is same with the

some part of the dataset used in [26]. Images are in 640x480x24 color images format. The distance of the camera to the car is in between 4-7 meters.

### **1.3 OUTLINE OF THE THESIS**

This thesis is composed of six main chapters.

In Chapter 1, the main scope of the thesis is explained and a brief introduction to the subject is given.

In Chapter 2, literature review for plate region extraction, segmentation of characters and character recognition is given.

In Chapter 3, the developed algorithm for the license plate detection and recognition is explained in detail.

In Chapter 4, FPGA implementation of the character segmentation and character recognition parts of the algorithm is explained from both hardware and software point of view. Block diagrams for the hardware modules and pseudo codes for the implemented software are provided.

In Chapter 5, test results for the MATLAB and FPGA implementations are provided. Proposed system is compared with the previous studies and performance comparison is given. Moreover, resource usage for the FPGA implementation is provided.

Chapter 6 gives a particular summary of the whole idea presented in the thesis and offers ideas for the future work.



## CHAPTER 2

### LITERATURE OF LICENSE PLATE DETECTION AND RECOGNITION

License plate detection and recognition systems consist of three main parts. These systems start with extraction of plate region, then characters in the plate region are segmented according to some constraints and finally plate characters are recognized by processing the segmented characters. In the literature, various techniques are proposed for these three steps. In the following sections, methods used in the previous works are given in each of the steps.

#### 2.1 EXTRACTION OF PLATE REGION

Babu and Krishnan [12] apply a morphology based method for license plate extraction from vehicle images. Proposed algorithm is based on combining morphological operation, sensitive to specific shapes on the edge images of the vehicles. Algorithm gives good responses for license plate images with complicated background. In the algorithm, Sobel mask is used to detect vertical edges in the input image because plate region consists of many vertical edges as a result of borders, letters and numbers. After edges on the image are found, the resultant image is converted into a binary image and candidate plate region is extracted by finding number of rows containing highest number of ones. To fill the holes on the image, morphological dilation is applied horizontally and vertically. After the dilation operation, morphological erosion is applied to exclude the extra regions which do not belong to the plate.

Iwanowski [1] also uses mathematical morphology. A set of filters is used which removes unnecessary image elements but preserves the position of the plate and shape of the characters. As a first step, top-hat contrast enhancement is applied to improve the contrast of the image. The formula for the top-hat contrast enhancement is given in (2.1).

$$g = 3f - \gamma^{(n)}(f) - \varphi^{(n)}(f) \quad (2.1)$$

where  $f$  is the image,  $\gamma$  represents the opening operator and  $\varphi$  represents the closing operator. After the contrast enhancement, to remove most of the unnecessary areas on the image, a combination of two top-hats (white and black) by reconstruction is applied by using the formulas given in (2.6). Formulas for the background cleaning are given in (2.2), (2.3), (2.4), (2.5), and (2.6).

$$WTH^{(n)}(f) = f - \gamma^{(n)}(f), \quad BTH^{(n)}(f) = \varphi^{(n)}(f) - f \quad (2.2)$$

where *WTH* stands for white top-hat and *BTH* for the black top-hat.

The geodesic dilation and erosion of size 1 are defined as given in (2.3), respectively.

$$\delta_g^{(1)}(f) = \delta^{(1)}(f) \wedge g, \quad \varepsilon_g^{(1)}(f) = \varepsilon^{(1)}(f) \vee g \quad (2.3)$$

where  $f$  represents an input image,  $g$  is a mask image,  $\wedge$  and  $\vee$  stands for the minimum and maximum values among pixels of the same coordinates on both images computed for all pixels.  $\varepsilon$  and  $\delta$  represent erosion and dilation, respectively.

The geodesic erosion and dilation of size  $n$  are defined as given in (2.4), respectively.

$$\varepsilon_g^{(n)}(f) = \varepsilon_g^{(1)}(\underbrace{\varepsilon_g^{(1)} \dots \varepsilon_g^{(1)}(f) \dots}_{n\text{-times}}), \quad \delta_g^{(n)}(f) = \delta_g^{(1)}(\underbrace{\delta_g^{(1)} \dots \delta_g^{(1)}(f) \dots}_{n\text{-times}}) \quad (2.4)$$

The reconstruction by dilation and erosion are defined as given in (2.5), respectively.

$$R_g(f) = \delta_g^{(i)}(f), \quad R_g^*(f) = \varepsilon_g^{(i)}(f) \quad (2.5)$$

where  $i$  is defined as the lowest number such that  $\delta_g^{(i)}(f) = \delta_g^{(i+1)}(f)$  and  $\varepsilon_g^{(i)}(f) = \varepsilon_g^{(i+1)}(f)$  for dilation and erosion, respectively.

The filters of opening by reconstruction and closing by reconstruction are defined as given in (2.6), respectively.

$$\gamma_R^{(n)}(f) = R_f(\varepsilon^{(n)}(f)), \quad \phi_R^{(n)}(f) = R_f^*(\delta^{(n)}(f)) \quad (2.6)$$

After background cleaning, in order to detect the proper area, the directional filtering is applied and then image is binarized. After these processes, the detected area contains all the characters.

Ozbay and Ercelebi [13] use edge detection and smearing algorithms to extract the plate region. To find the plate region, firstly, smearing algorithm is applied to the binarized image. Smearing is a method for the extraction of the areas which contains text area. With smearing algorithms, image is processed along vertically and horizontally. If the number of bright pixels is less than a desired threshold or greater than any other desired threshold, bright pixels are converted to black pixels. To specify the plate position, a morphological dilation is implemented followed by the smearing algorithm.

Hung and Hsieh [14] utilize the color characteristics of the barking lights in the input image. Firstly, locations of the barking lights are detected. Then, plate detection region is found by using the probability distribution of the license plate between the two lights. Wavelet transform and projection method are used as a proposed technique of the vehicle license plate detection. To obtain the relationship between the frequency and the location, multi-resolution of the signal is performed by the wavelet transform. One level discrete

wavelet transformation is used to extract the high frequency content from the image. Since the number of characters is known, the number of vertical boundary points in the horizontal projection is also known which is twice of the number of characters in the plate region. Horizontal projection of the frequency band is obtained by adding the energy values at the same horizontal edge of the frequency band. Horizontal rows with higher horizontal projection values represent the upper and lower edges of the license plate. After horizontal projection, vertical projection is used to detect the left and right side edges of the plate region. If the projection value is smaller than the predefined threshold that column may not be the part of the license plate and excludes the checking region. By this approach, right and left boundary of the license plate is obtained.

Duman et al. [15] focuses on feature extraction approaches to extract license plate regions. Hierarchical edge detection method based on discrete wavelet transform is used to restrict the feature extraction regions. The aim of the hierarchical edge detection is to eliminate the regions which cannot be plate region and minimize the number of regions which require detailed processing. In order to achieve this aim, discrete wavelet transform is used. Since plate region contains many edges due to boundaries, numbers and letters, high frequency components of the wavelet transform is detected as candidate plate regions and other regions are discarded. After the hierarchical edge detection, to differentiate which region is the exact plate region and which is not, three different feature extraction methods are used. These methods are observing histogram of the candidate plate region, horizontal profile of candidate region chosen in the edge map and segmentation of background and foreground objects.

Commeli et al. [16] extract the location of the plate region by observing the unusual characteristics presented by the image function. They look for the geometrical characteristics such that rectangular region which contains black character over white background. Therefore, the algorithm picks within the image, the area presenting the maximum local contrast which corresponds the rectangle containing the license plate. A gradient analysis is applied on the whole image which leads to estimate of a rectangle.

Wang and Lee [17] use the magnitude of vertical gradients to detect the candidate plate regions. After that, these regions are evaluated based on three geometrical features: the ratio of width and height, the size and the orientation. Because the color of characters is not same with the background of the plate region, the gradients of the original image can be used to detect the candidate license plate region. The gradients are derived by multiplying with a mask value for each pixel and its neighboring pixels. Sobel operators use two masks to find vertical and horizontal gradients [18, 19]. License plate region is hardly separated by using the horizontal gradients. However, detection of license plate from vertical gradients is easy because magnitude of the vertical gradients is strong in the characters of the license plate while weak on vertical lines. To detect the most possible license plate regions from candidate plate regions, geometrical properties of the license plate such as area, orientation and density, which is the ratio between the black regions and the area of the bounding rectangle, are used as a measure of the possibility value.

Kahraman et al. [20] utilize Gabor transform for the detection of the plate region. Gabor filters are the one of the major tools for the texture analysis. In the algorithm, texture over the image is analyzed in an unlimited number of directions and scales. Gabor feature of the image is obtained with the formula given in (2.7).

$$r(x, y) = \iint_{\Omega} I(\xi, \eta) g(x - \xi, y - \eta) d\xi d\eta \quad (2.7)$$

In the (2.7), input image  $I(x, y)$ ,  $x, y \in \Omega$ , where  $\Omega$  is the set of image points, is convolved with 2D Gabor function  $g(x, y)$ ,  $x, y \in \Omega$  to obtain a Gabor feature image  $r(x, y)$ .

Following family of Gabor functions are used as given in (2.8).

$$g_{\lambda, \theta}(x, y) = e^{-\frac{(x'^2 + y'^2)}{\sigma^2}} \cos(2\pi \frac{x'}{\lambda}) \quad (2.8)$$

$$x' = x \cos \theta + y \sin \theta$$

$$y' = -x \sin \theta + y \cos \theta$$

where  $\sigma$  and  $\lambda$  represent standard deviation and the wavelength, respectively. The angle parameter  $\theta$  specifies the orientation of the normal to the parallel and negative lobes of the Gabor filters. High values in the Gabor response  $r(x, y)$  indicate probable plate regions. To extract the exact region, thresholding algorithm is applied to convert the image to binary image and morphological dilation operator is utilized to merge the nearby regions. Eight-connected blob coloring is used to specify the exact plate region.

## 2.2 SEGMENTATION OF CHARACTERS

Character segmentation is the requisite part of the license plate recognition system because each character is recognized by using the segmented characters. Unless characters are segmented successfully, license plate cannot be recognized satisfactorily.

After the plate region is localized accurately, segmentation of characters part is applied. There are several approaches developed for the character segmentation.

Kahraman et al. [20] apply nonlinear vector quantization to segment the license plate characters to its exact boundary. In the vector quantization methods, pixel values are assigned to one of the finite number of vectors and by using binary split tree approach, these vectors are determined in such a way that the quantization error is minimized. Quantization error equation is given in (2.9).

$$E = \sum_{n \in Q} \sum_{s \in C_n} \|I_s - q_n\|^2 \quad (2.9)$$

where  $\{q_n\}$  is the set of quantization vectors and  $C_n$  is the set of pixels assigned to the vector  $q_n$ . In the vector quantization approach, initially all pixels are assigned to same



class and then, class is divided into two sub classes according to the second order statistics within the class. Quantization vector  $q_n$  is assumed to be equal the class mean.

$$R_n = \sum_{s \in C_n} I_s I_s^T \quad m_n = \sum_{s \in C_n} I_s \quad K_n = |C_n| \quad q_n = \frac{m_n}{K_n} \quad (2.10)$$

Class covariance matrix is given in (2.11)

$$\check{R}_n = R_n - \frac{1}{K_n} m_n m_n^T \quad (2.11)$$

Due to the division of the classes into two sub classes, new class vectors,  $(q_{2n}, q_{2n+1})$ , are computed by calculating the unit vector  $e_n$  which minimizes the expression given in (2.12)

$$\sum_{s \in C_n} ((I_s - q_n)^T e)^2 = e^T \check{R}_n e \quad (2.12)$$

This represents the largest eigenvalue of  $\check{R}_n$ . After unit vectors are obtained, the pixel values are assigned to  $C_{2n}$  or  $C_{2n+1}$  classes as given in (2.13) and (2.14).

$$C_{2n} = \{s \in C_n: \quad e_n^T I_s \leq e_n^T q_n\} \quad (2.13)$$

$$C_{2n+1} = \{s \in C_n: \quad e_n^T I_s > e_n^T q_n\} \quad (2.14)$$

Splitting classes stops either when the maximum vector number is reached or when the class variance is less than a predefined threshold. After the plate region is quantized, connected component analysis is implemented to obtain the character segments.

Wu and Chiu [21] propose a thresholding approach, modified adaptive thresholding, MAT, which combines the BAT, basic adaptive thresholding, and c-means algorithm to segment the foreground. The proposed system uses only the Y-component in the YUV color space representation to extract the characters. After thresholding, modified connected component labeling method is used to explore the region adjacent to each pixel. In the proposed modified connected component labeling method, attributes are obtained which are the coordinates, area, the aspect ratio and density of a component. After the attributes of connected components have been determined, candidates of the character components are determined by using thresholding operators which examine candidates according to their width, height and area. Horizontal position and widths of the characters are adjusted and ratio and densities of the components are modified. Objects with excessively large or small aspect ratios and densities are eliminated by applying ratio and density thresholding. As a final step, connected components which are within the positional tolerance and the size tolerance determined as character components.

In the system proposed by Wang and Lee [17], binarized license plate images which are binarized with the Otsu method are utilized. The de-noise process is applied to eliminate small regions and the boundary. For the binarized license plate image, the orientation is obtained. Then, connected component based method is used to segment the image into single characters. For the proposed system, the digits on the license plate are fixed on car

license plates and the characters lie in horizontal orientation. By using these properties, characters of the connected component are confirmed. Due to the possible various rotation angles of the characters in the license plate, the major axis of each single character is measured and the inverse rotation transformation is implemented to normalize the character.

Ozbay and Ercelebi [13] use smearing method to segment the characters. In the proposed system, firstly, image is filtered for enhancing the image and removing the noise and unwanted pixel values. Then, morphological dilation is applied to separate the characters from each other if the characters are close to each other. After dilation, horizontal and vertical smearings are applied to find the character regions. As a final step, plate characters are cut by finding starting and end points of characters in horizontal direction.

Iwanowski [22] implements binarization and labeling to extract single characters on the image. In the first step, image is binarized with fixed threshold values as given in (2.15).

$$[T_{[a,b]}(f)](i,j) = \begin{cases} 1 & \text{if } 0 \leq f(i,j) \leq 90 \\ 0 & \text{otherwise} \end{cases} \quad (2.15)$$

To remove unnecessary noise and other small areas of value '1' on the binarized image, morphological opening is applied. After that, labeling of binary image is performed by using the labeling algorithm. Algorithm results in the image with labeled characters, so that the first character from the left gets label '1', next one label '2' etc. For the extraction of single characters, (2.16) is used.

$$[T_{[l]}(f)](i,j) = \begin{cases} 1 & \text{if } f(i,j) = l \\ 0 & \text{otherwise} \end{cases} \quad (2.16)$$

Pseudo code for the labeling algorithm as given:

```

for i := 0 to x : for j := 0 to y:
    if p(i,j)=0 then l(i,j)=0 else l(i,j)=-1
label := 0
for i := 0 to x : for j := 0 to y: if l(i,j)=-1 then:
    label := label+1
    stack_push([i,j])
    while not(stack_empty):
        [i,j] := stack_pop
        l(i,j) := label
        for m:=-1 to 1: for n:=-1 to 1:
            if l(i+m,j+n)=-1 then:
                stack_push([i+m,j+n])

```

where  $p$  and  $l$  represents the binary image of the license plate and the labeled output image, respectively.  $l$  contains the particles of image  $p$  labeled beginning from the value 1. The final value of variable  $label$  represents the number of labeled characters.

Babu and Krishnan [12] use edge detection and region growing for the segmentation of the characters. Firstly, plate region is enhanced with the contrast stretching method. The idea behind the contrast stretching is to increase the dynamic range of the gray levels in the image. After that, as an edge detection method, Laplacian transformation is used to detect the edges of the characters. To find the location of the candidate regions of the characters, region growing is applied followed by the edge detection. The basic approach for region growing is to start with the seed point and from this region grows by appending to each seed those neighboring pixels that have properties similar to that seed. Since in the process of region growing more than one character may be combined together in a region or a character may be split into some regions, the information about the ratio of the character width and height is used and accurate positions of character segmentation regions are obtained. According to these positions, characters from original license plate are extracted accurately.

### 2.3 RECOGNITION OF PLATE CHARACTERS

After each character is segmented, recognition procedure starts. For the recognition of the license plate character, there are many approaches. The most common approaches in the literature are template based and neural network based algorithms.

Fahmy [23] uses the BAM (bidirectional associative memories) neural network for the recognition. According to Fahmy, using neural networks in automatic character reading has the potential of providing higher levels of accuracy than the conventional methods. In the proposed method, each character included in the plate region is scaled to fit a 16x16 pixel image prior to introducing it to the input layer of a neural network and then, scaled version of the characters is used for character recognition. BAM system is trained with actual character patterns extracted from license plate. These character patterns are stored in different files, each is used to store one of the letters from A to Z or one of the digits from 0 to 9 with its desired output. The output of the BAM is a 6x6 pixel matrix and each character is assigned to a unique output pattern in this matrix.

Lee et al. [24] use template matching and post-processing techniques for the character recognition. To match the characters of the license plate with the template characters, size of the extracted characters is normalized to 40x40. Then, Jaccard value is used as a matching measure which is given in (2.17).

$$J = \frac{M_{11}}{(M_{11}+M_{10}+M_{01})} \quad (2.17)$$

After matching, to confirm the recognition result, post-processing using character dictionary is applied.

Babu and Krishnan [12] use cross correlation combined with neural network. Cross correlation is used to calculate the similarities between two images. To examine the similarity between the image and the template, assumptions are made and constant terms in

the cross correlation equation are eliminated. Equation used as a similarity measure is given in (2.18).

$$c(u, v) = \sum_x \sum_y f(x, y)t(x - u, y - v) \quad (2.18)$$

where  $f(x, y)$  and  $t(x, y)$  represent the segmented character and the template, respectively. Each character is presented to a bank of neural network, each looking for a specific character or digit on which it has been previously trained. The final decision for each character is chosen by a winner takes all comparison of the network outputs. However, if the value of the winner's output is low, this is recognized as a failure and recognition result is aborted.

Ozbay and Ercelebi [13] use normalized cross-correlation with template matching. In the proposed system, segmented characters are normalized to 36x18. Method measures the correlation coefficient between a number of known images with the same sized unknown images. Highest correlation coefficient between the images produces the best match. The normalized cross-correlation between the image pair is given in (2.19).

$$R(m, n) = \frac{\sum_j \sum_k F_1(j, k)F_2(j-m+(M+1)/2, k-n+(N+1)/2)}{[\sum_j \sum_k |F_1(j, k)|^2]^{1/2} [\sum_j \sum_k |F_2(j-m+(M+1)/2, k-n+(N+1)/2)|^2]^{1/2}} \quad (2.19)$$

for  $m=1, 2, \dots, M$  and  $n=1, 2, \dots, N$  where  $M$  and  $N$  are odd integers.  $F_1(j, k)$  and  $F_2(j, k)$  for  $1 \leq j \leq J$  and  $1 \leq k \leq K$  represent two discrete images denoting the image to be searched and the template, respectively.

Wang and Lee [17] propose a system which extracts the features of the segmented character and recognizes the character by using discriminant function. To tolerate variations among samples of the same character and differentiate variations in different characters, significant features are extracted. To increase robustness of the recognition method, segmented characters are rotated to a normalized coordinate system. Crossing-count features and peripheral background are extracted. For the crossing-count calculation, segmented character is divided into 64 sub-images. Each sub-image of the quarter is then segmented non-uniformly into 4 strips in both horizontal and vertical directions. To speed up the extraction, four scan lines are selected to extract features in each strip and crossing counts are calculated by calculating the number of strokes intersecting each scan line. After crossing count feature are calculated, peripheral background area feature is extracted. Peripheral background area feature is the length of line segments which are from the boundary of the image to the character contour. To extract the feature, extracted character is segmented into 16 strips in both horizontal and vertical directions. In each strip, two values of length from both of the image boundary to the character contour divided by the length of the strip are measured. For the character recognition, discriminant and combined discriminant functions are used which are given in (2.20) and (2.21), respectively.

$$d(x, \mu_j) = \sum_{i=1}^{16} \frac{(x_i - \mu_{ji})^2}{\sigma_{ji}^2} + c \log \sigma_{ji} \quad (2.20)$$

$$Cb = w_1d(CCFs, \mu_j) + w_2d(PBAFs, \mu_j) \quad (2.21)$$

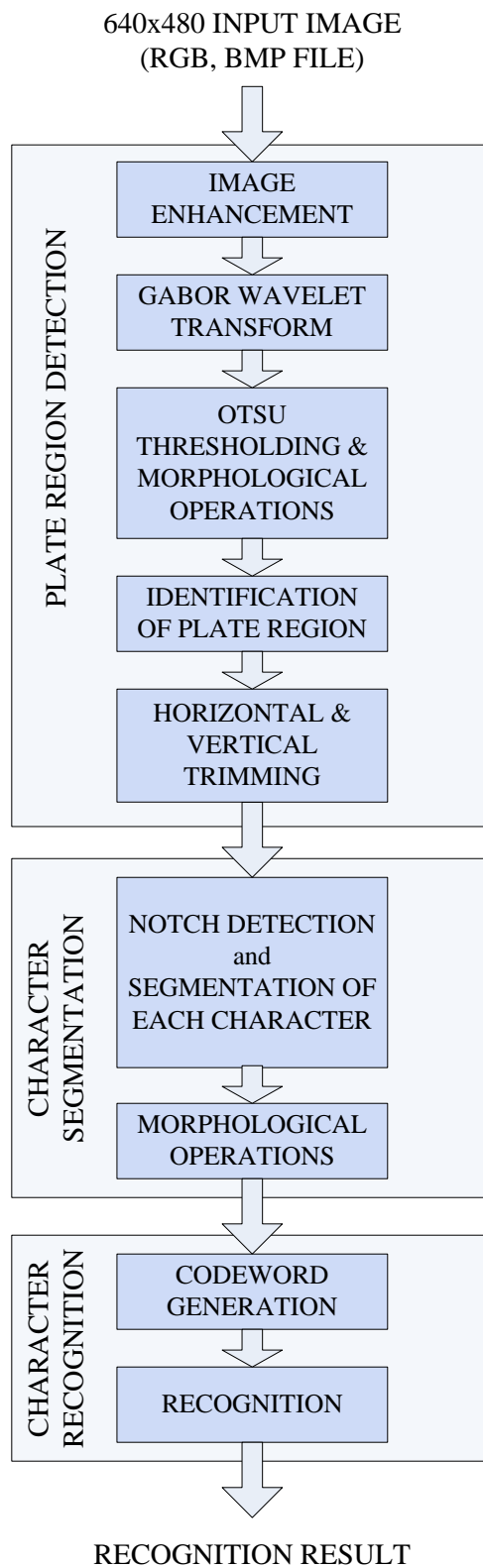
where feature vector  $x$  is derived from the segmented character and  $\mu_j$  is the mean vector of the reference character  $j$ .  $\sigma_j$  is the standard deviation accompanying each feature of character  $j$ .  $c$  is the constant obtained from the training data.  $w_1$  and  $w_2$  are the coefficients which are determined according to experimental results. CCFs stands for crossing count features and PBAFs stands for peripheral background area features. The combined discriminant function between the segmented character and each reference character is evaluated and smallest value among them is chosen as the recognized character.



## **CHAPTER 3**

### **PROPOSED SYSTEM**

Proposed license plate detection and recognition system (LPDRS) consists of three main subsystems. These are plate region detection, character segmentation and character recognition. The input image is given to the LPDRS. In the plate region detection step, exact location of the license plate is found. First, the RGB image is represented in HSV color space and value component is used to decrease the computational complexity. For the detection of candidate plate regions, Gabor wavelet transform and morphological closing are utilized. In order to obtain reasonable results, image filtering and contrast enhancement are implemented before the Gabor wavelet transform. After candidate plate regions are obtained, extent of each candidate plate region is analyzed to find the exact plate region. If the trimming is necessary in the extracted plate region, up-down and right-left hand side trimmings are achieved to eliminate unwanted pixels. In the character segmentation step, each character in the plate region is extracted and morphological opening is used to eliminate unwanted pixels which may be missed while trimming. Finally, in character recognition step, for each character in the license plate, codeword is generated and license plate characters are identified by matching algorithms which compare the generated codeword with codewords in the database. Flow diagram of the proposed system is given in Figure 3-1. In this chapter, each process in the flow diagram will be explained in detail.



**Figure 3-1 Flow Diagram of the Proposed System**



### 3.1 PLATE REGION DETECTION

The aim of the plate region detection step is to find positions of candidate plate regions in the input image with their sizes and coordinates and determine the exact plate region. In the input image, pixels are divided into two groups whether they belong to plate region or not. Only pixels of the plate region are transferred to later stages.

#### 3.1.1 IMAGE ENHANCEMENT

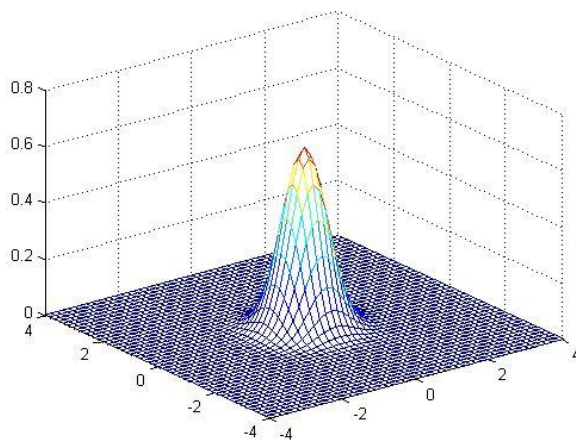
For the plate region detection, color information is not a necessity. The only need to detect the plate region is to know positions of bright and dark pixels. To decrease the computational complexity and to exclude the color information, RGB representation of the input image is converted into HSV color space. In HSV representation, value component represents the quality by which a light color is distinguished from a dark one. Since it is important to find out light and dark pixels' transitions in the input image, value component is used for further processes. Value component of the image is filtered with Gaussian filter to remove the high frequency components and smoother image is obtained. After filtering, contrast enhancement is applied with morphological operations.

##### 3.1.1.1 GAUSSIAN FILTER

In order to reduce high frequency components and to remove noise, image is filtered with Gaussian filter and smoother image is obtained.

An isotropic 2D Gaussian function  $g(x, y)$  is given in (3.1) and the plot of the function is shown in Figure 3-2.

$$g(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \quad (3.1)$$



**Figure 3-2 Gaussian Function**

Gaussian function is used as a 2D distribution as a point spread function. Since the image is stored as a collection of discrete pixels, discrete approximation of Gaussian function is required. Figure 3-3 shows a suitable integer-valued convolution kernel that approximates a Gaussian with a  $\sigma$  of 0.5.

0	0	0.0002	0	0
0	0.0113	0.0837	0.0113	0
0.0002	0.0837	0.6187	0.0837	0.0002
0	0.0113	0.0837	0.0113	0
0	0	0.0002	0	0

**Figure 3-3 Gaussian Kernel**

Gaussian filtering blurs the image. The degree of smoothing is determined by the standard deviation of the Gaussian. The Gaussian outputs a weighted average of each pixel's neighborhood, with the average weighted more towards the value of the central pixels. Because of this, a Gaussian provides gentler smoothing and preserves edges better than similarly sized mean filter.

### 3.1.1.2 CONTRAST ENHANCEMENT

Gabor wavelet transform gives maximum response at the black to white pixel transitions. After image is filtered, to maximize the Gabor response, contrast enhancement over the image is applied. Contrast enhancement is achieved by morphological operators.

Morphological operators are used not only to remove image objects or noise of certain kind but also to detect objects of particular characteristics [1]. In order to achieve this, top-hat operators have been used which are given in (3.2) and (3.3).

$$WTH^n(f) = f - \gamma^{(n)}(f) \quad (3.2)$$

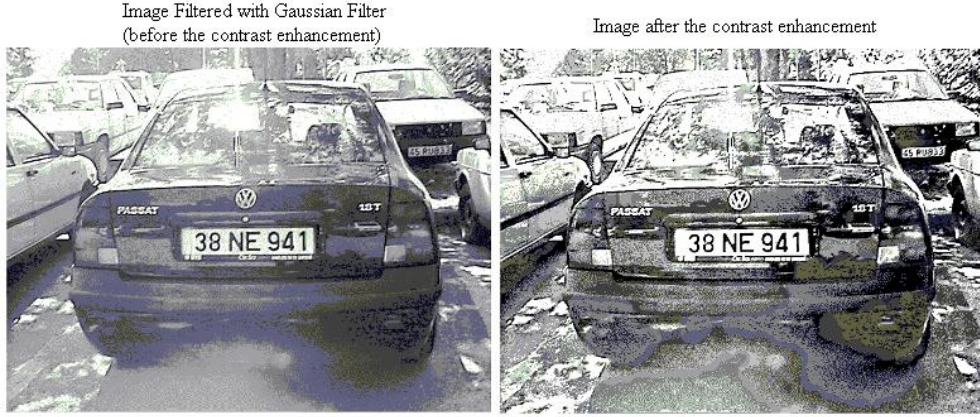
$$BTH^{(n)}(f) = \varphi^{(n)}(f) - f \quad (3.3)$$

where  $f$  is the image,  $\gamma$  represents the opening operator,  $\varphi$  represents the closing operator,  $WTH$  stands for white top-hat and  $BTH$  for black top-hat. The descriptions 'white' and 'black' indicates types of objects that are detected by a particular operator, lighter or darker than the background.

Top-hat operators can be applied to contrast enhancement. By combining the original image with images with detected objects, the contrast improves. This combination is achieved by adding the original image to the result of white top-hat and by subtracting the result of black top-hat. The equation for contrast enhancement is given in (3.4).

$$g = f + WTH^{(n)}(f) - BTH^{(n)}(f) = 3f - \gamma^{(n)}(f) - \varphi^{(n)}(f) \quad (3.4)$$

Image before and after the contrast enhancement can be seen in Figure 3-4.



**Figure 3-4 Images Before and After the Contrast Enhancement**

### 3.1.2 GABOR WAVELET TRANSFORM

There is a similarity between Gabor filters and the receptive field of simple cells in the visual cortex [2, 3]. Therefore, the Gabor feature based methods are among the top performers for feature extraction. 2D Gabor filters proposed by Daugman are local spatial band pass filters that achieve the theoretical limit for conjoint resolution of information in the 2D spatial and 2D Fourier domains [4]. An ensemble of simple cells is best modeled as a family of 2D Gabor wavelets sampling the frequency domain in a log-polar manner. The composition of image into these states is called the wavelet transform of the image [5].

An input image  $I(\vec{x})$ , where  $\vec{x}$  is the image points, is convolved with Gabor-based wavelets which is a family of kernels  $\psi_{\vec{k}}$ . The parameter  $\vec{k}$  determines the wavelength and orientation of the kernel  $\psi_{\vec{k}}$ . Convolution of the input image with Gabor-based wavelets is given in (3.5).

$$(WI)(\vec{k}, \vec{x}_0) := \int \psi_{\vec{k}}(\vec{x}_0 - \vec{x}) I(\vec{x}) d^2x = (\psi_{\vec{k}} * I)(\vec{x}_0) \quad (3.5)$$

The kernels  $\psi_{\vec{k}}$  in image coordinates take the form of a plane wave restricted by a Gaussian envelope function:

$$\psi_{\vec{k}}(\vec{x}) = \frac{\vec{k}^2}{\sigma^2} \exp\left(-\frac{\vec{k}^2 \vec{x}^2}{2\sigma^2}\right) \left[ \exp(i\vec{k}\vec{x}) - \exp\left(-\frac{\sigma^2}{2}\right) \right] \quad (3.6)$$

Center frequency of the filter characterized by the wave vector:

$$\vec{k}_{v\mu} = k_v e^{i\theta_\mu} = \begin{pmatrix} k_x \\ k_y \end{pmatrix} = \begin{pmatrix} k_v \cos \theta_\mu \\ k_v \sin \theta_\mu \end{pmatrix} \quad \text{with } k_v = \frac{k_{max}}{fv}, \quad \theta_\mu = \frac{\pi\mu}{8} \quad (3.7)$$

where  $f$  represents the spacing factor between kernels in the frequency domain. In the LPDRS, Gabor wavelet kernels are investigated with the values of  $k_{max} = \frac{\pi}{2}$  and  $f = \sqrt{2}$ . In (3.7),  $k_v$  represents scale and  $\theta_\mu$  represents the orientation.

In the Gabor kernels (3.6), the first term in the square brackets determines the oscillatory part of the kernel. The second term compensates for the DC value of the kernel. DC value of the kernel avoids unwanted dependence of the filter response on the absolute intensity of the image. For sufficiently high values of  $\sigma$ , the effect of the DC term becomes negligible. The complex valued  $\psi_{\vec{k}}$  is a combination of cosine (even part) and sine (odd part) [5].

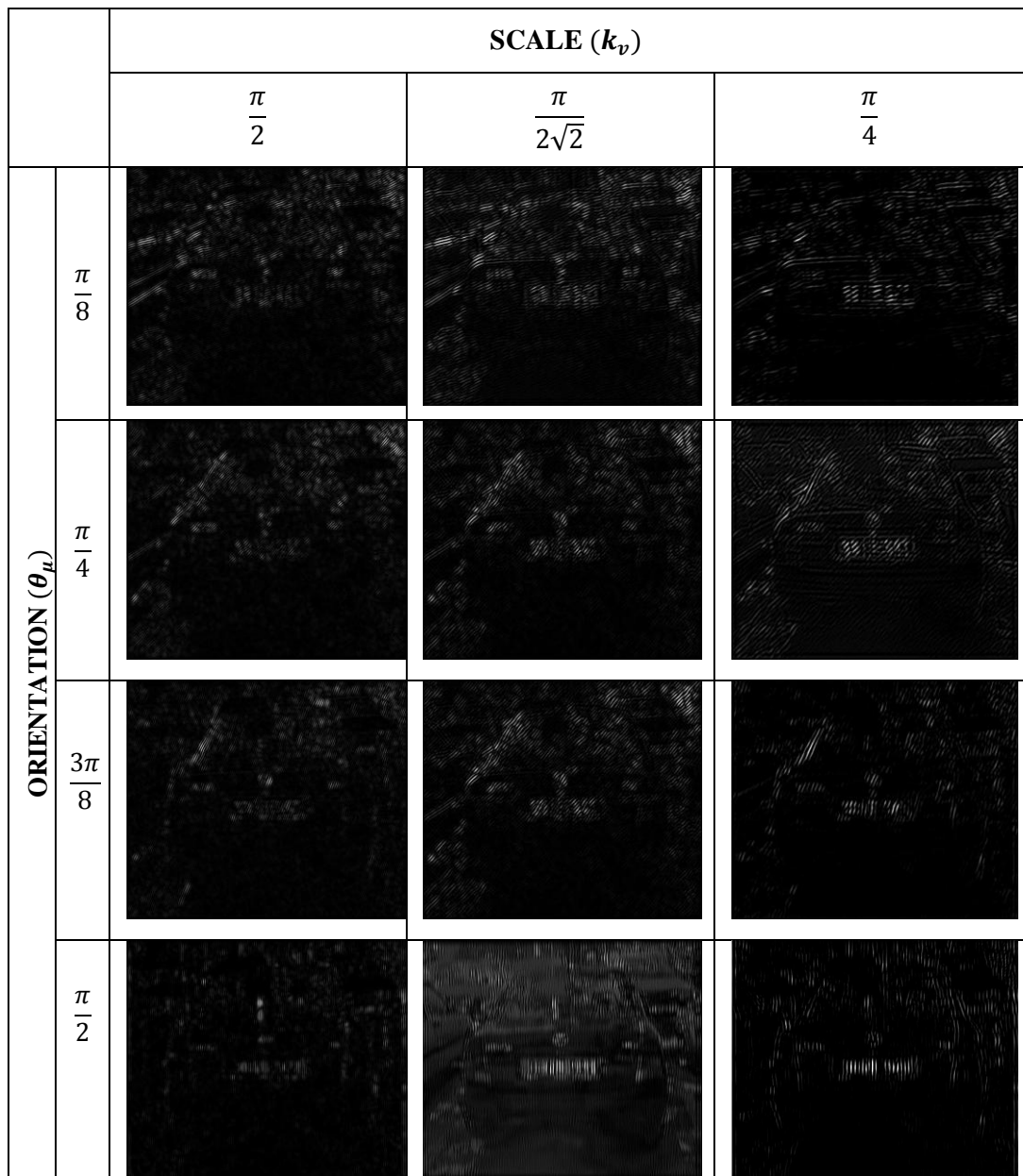
Response of Gabor wavelets  $\psi_{\vec{k}}$  in Fourier space is given in (3.8).

$$(F\psi_{\vec{k}})(\vec{k}_0) = \exp\left(-\frac{\sigma^2(\vec{k}_0-\vec{k})^2}{2\vec{k}^2}\right) - \exp\left(-\frac{\sigma^2(\vec{k}_0+\vec{k})^2}{2\vec{k}^2}\right) \quad (3.8)$$

where  $F$  represents the Fourier transform. Gabor filters is mainly a product of an elliptical Gaussian and a complex plane wave. In the Gabor wavelet transforms, first exponential provides a band pass filter which is a Gaussian centered at the characteristic frequency  $\vec{k}$ . The second exponential removes the DC component of the  $\psi_{\vec{k}}$ . The Gabor wavelets are defined by wave vector  $\vec{k}$ , which controls the width of the Gaussian window and wavelength and orientation of the oscillatory part. The parameter  $\sigma$  controls the ratio of window width to wavelength, i.e., the number of oscillations under the envelope function.

In the LPDRS, for extracting the license plate feature of image, the system uses Gabor filters. Gabor features extract local pieces of information which are then combined to detect the region of interest which is a candidate plate region for this case.

Gabor transform gives only a rough estimate of the boundary of the plate region. In the license plate detection and recognition system, size of the Gabor wavelet kernels is chosen as 16x16. The size of the kernel has a great importance for the computation time. Since resolution of the input image is 640x480, larger kernel size increases the computation time and lower kernel size results from the loss of the plate region features on the image. For the Gabor kernels standard deviation is chosen as  $\sigma = 2\pi$ . In the LPDRS, Gabor wavelets are generated with 3 logarithmically spaced frequency levels and 4 orientations indexed by  $\nu \in \{0, \dots, 2\}$  and  $\mu \in \{1, \dots, 4\}$ . Real parts of the Gabor wavelet transforms of the input image for 3 frequency levels and 4 orientations can be seen in Figure 3-5.



**Figure 3-5 Gabor Wavelet Transform of the Input Image**

As it can be seen in the Figure 3-5, plate region can be detected mostly for  $k_v = \frac{\pi}{4}$  and  $\theta_\mu = \frac{\pi}{2}$ . For other frequency levels or orientations not only plate region is detected but also other parts of the image can emerge. The most acceptable Gabor response for this scale and orientation is given in Figure 3-6.



**Figure 3-6 Most Acceptable Gabor Response of the Input Image**

Input images used in the license plate detection and recognition system are taken in front of the vehicle without giving any orientation to the camera. Therefore, it is meaningful to obtain most acceptable response for  $\theta_{\mu} = \frac{\pi}{2}$ . Gabor wavelet filters seek for the black to white pixel transitions and the most reasonable region is the plate region since it consists of black characters with white background. High values in the Gabor transform of the image indicate probable plate regions.

### 3.1.3 OTSU THRESHOLDING and MORPHOLOGICAL OPERATIONS

In order to detect the plate region, thresholding algorithm is applied and binary image is obtained from grayscale image. Then, morphological operator is utilized to merge nearby regions. In the license plate detection and recognition system, Otsu's method is used as a thresholding algorithm. Otsu's method is a histogram shape-based thresholding. In the algorithm, pixels of the image are separated into two classes by calculating the optimum threshold, therefore, their within-class variance becomes minimum and between-class variance becomes maximum [6].

Weighted within-class variance is given in (3.9).

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t) \quad (3.9)$$

where the class probabilities are estimated as:

$$q_1(t) = \sum_{i=1}^t P(i) \quad \text{and} \quad q_2(t) = \sum_{i=t+1}^L P(i) \quad (3.10)$$

pixels with levels  $[1, \dots, t]$  belongs the one class and pixels with levels  $[t+1, \dots, L]$  belongs to other class.

Class means are given in (3.11).

$$\mu_1(t) = \sum_{i=1}^t \frac{iP(i)}{q_1(t)} \quad \text{and} \quad \mu_2(t) = \sum_{i=t+1}^I \frac{iP(i)}{q_2(t)} \quad (3.11)$$

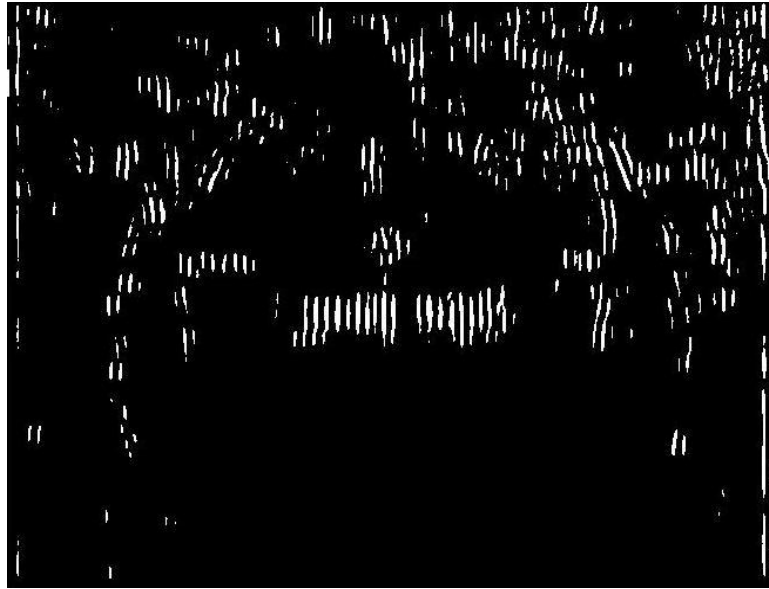
Individual class variances are given in (3.12) and (3.13).

$$\sigma_1^2(t) = \sum_{i=1}^t [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \quad (3.12)$$

$$\sigma_2^2(t) = \sum_{i=t+1}^I [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)} \quad (3.13)$$

With these equations, weighted within-class variance is run through the full range of  $t$  values  $[1,256]$  and the threshold value from these values is chosen which minimizes the  $\sigma_w^2(t)$ .

Grayscale image is binarized with Otsu's method after the Gabor-wavelet transform and it can be seen in Figure 3-7.



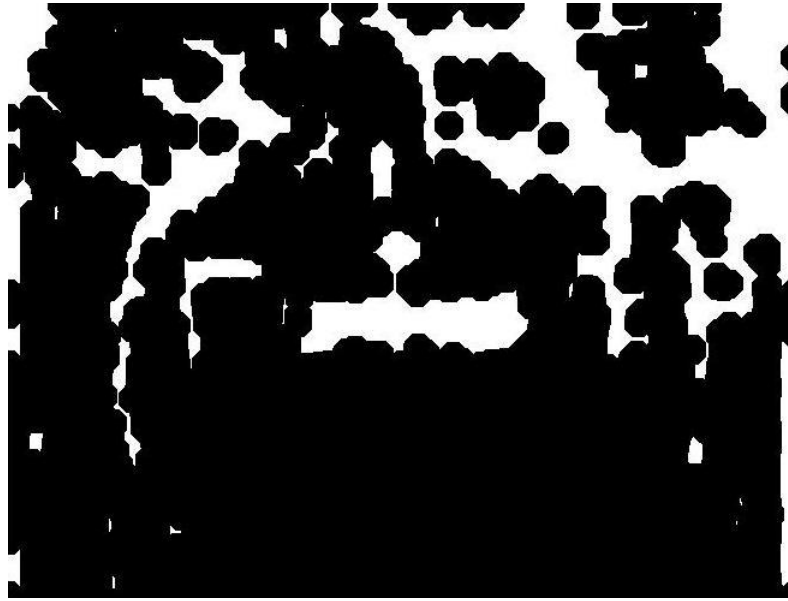
**Figure 3-7 Binarized Image with Otsu Thresholding After Gabor Wavelet Transform**

After probable plate regions are indicated as high values in the binary image, morphological closing is applied to the binary image to merge nearby regions. Closing operation enlarges the boundaries of the bright regions in the image and shrinks background color holes in such regions. It is used to combine the bright strips that belong to plate region. For closing operation, first binary image is dilated and then the dilated image eroded with same structuring element. Disk-shaped structuring element with radius 12 is used. Structuring element used in closing operation and the binary image after the closing operation is shown in Figure 3-8 and Figure 3-9, respectively.

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

**Figure 3-8 Structuring Element Used For Morphological Closing**

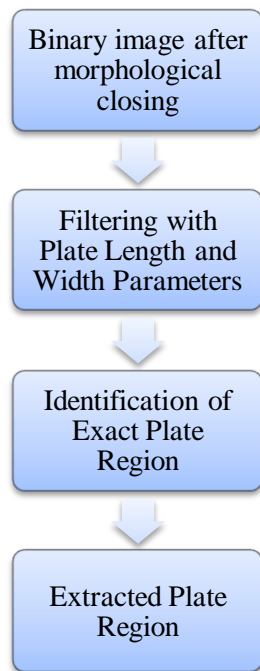




**Figure 3-9 Binary Image After Morphological Closing**

#### **3.1.4 IDENTIFICATION OF PLATE REGION**

After bright pixels are merged with morphological closing operation, candidate plate regions can be extracted which comply with the predefined plate length and width constraints. License plates can be single line or double line. In the license plate detection and recognition system, to extract candidate plate regions, binary image is filtered according to predefined plate length and width parameters. For this approach, two sets of threshold values are defined, one set for the single line plate case and the other set for the double line plate case. First, algorithm searches for the single line plate and if it finds a candidate region, algorithm for the double line plate case is not processed. If the candidate plate region is not found for the single line plate case, algorithm for the double line plate case is processed. The reason to place a priority to process algorithms is to decrease the computation time and not to process the nearly same algorithms with different thresholds two times. After the algorithms are processed, connected component labeling is implemented and image is filtered with more restricted thresholds. By this approach, additional candidate regions are extracted. For some plates, after the morphological closing, plate region becomes thinner and it cannot be found by filtering the image with plate width and length parameters. Therefore, filtering is achieved with more restricted threshold values. After the candidate plate regions are extracted, exact plate region is detected by analyzing the density of the bright pixels. Since plate region consists of black characters with white background, it is expected to have white pixel density larger than 0.5. Flow diagram of the identification of the plate region step can be seen in Figure 3-10.



**Figure 3-10 Flow Diagram of the Identification of Plate Region**

In the following sections, each substep for the identification of plate region is explained in detail.

#### **3.1.4.1 FILTERING WITH PLATE LENGTH and WIDTH PARAMETERS**

Filtering with plate length and width parameters can be divided into three substeps. Firstly, image is filtered according to single line plate case. Then, if the candidate plate region is not found, image is filtered according to double line plate case parameters. Otherwise, this substep is skipped. Finally, image is filtered with restricted parameters in order to find thinner plate regions. After candidate plate regions of each step are extracted, they are merged and the image becomes ready for the connected component analysis.

For the single line plate case, the algorithm seeks a continuous bright region with 10 pixels width and 140 pixels length. It extracts regions which have at least 10 pixels width and 140 pixels length. After regions are extracted, to analyze the content of each region connected component labeling with four neighboring is utilized and for each region, region properties such as region size, extent of the region, which specifies the ratio of pixels in the region to pixels in the total bounding box, are extracted. In the LPDRS, region whose extent is larger than 0.9 is taken into account and other regions are removed. Since filtering with parameters step seeks for at least 10 pixels width and 140 pixels length, extracted region is not always rectangular. There may be fluctuations at the edges. To eliminate the region with sharp fluctuations, LPDRS controls the extent of the region. If the extent of the region is larger than 0.9 which means it consists of 90% bright pixels, this region can be candidate and it is enlarged 15 pixels from top and bottom to reduce the risk of the cropping license

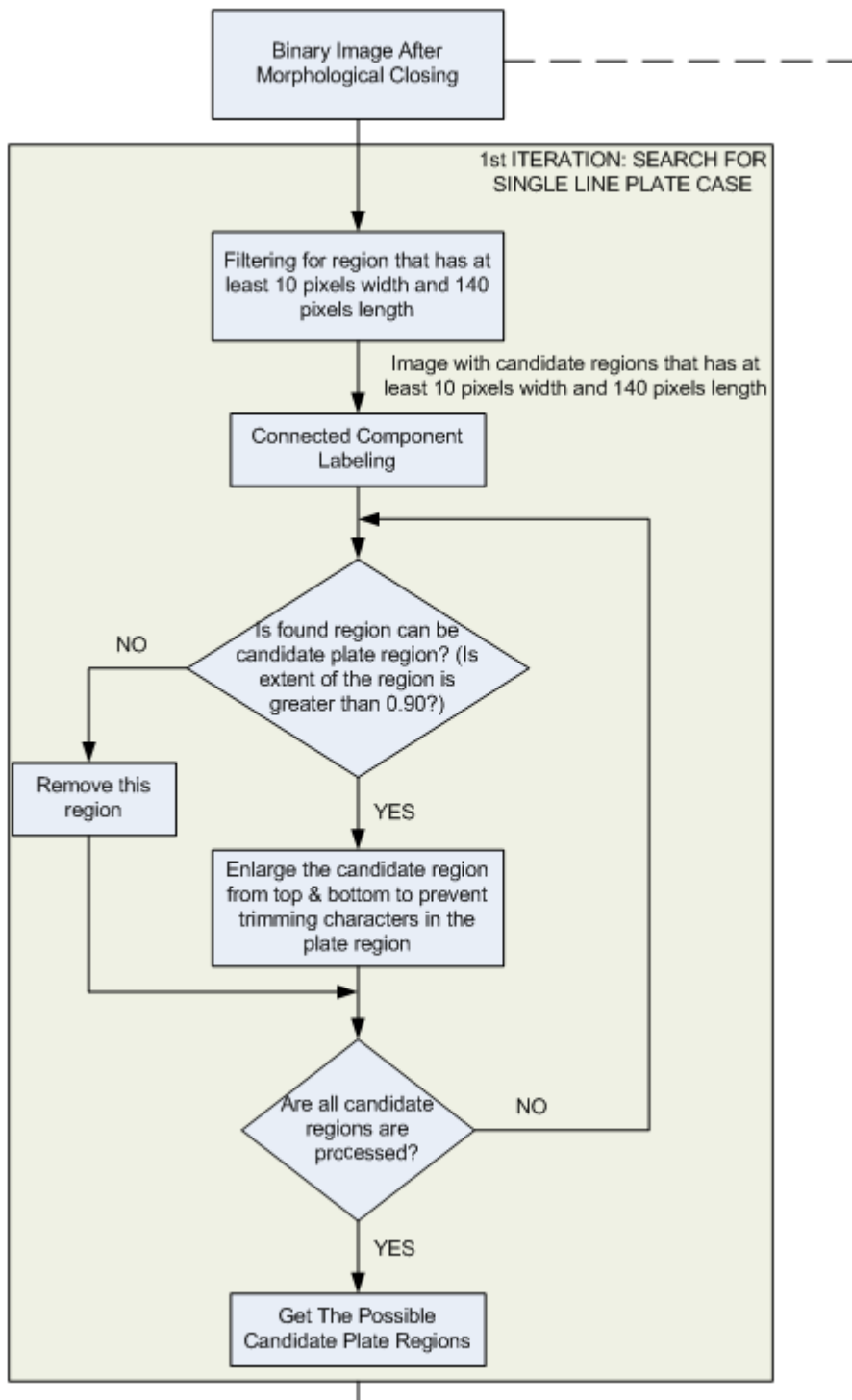
plate characters. All possible regions are analyzed and candidate plate regions are extracted in same manner. If there are candidate plate regions, the algorithm continues with filtering with restricted parameters step, otherwise it continues with double line plate case.

For the double line plate case, the algorithm seeks a continuous bright region with 20 pixels width and 90 pixels length. It extracts regions which have at least 20 pixels width and 90 pixels length. Same steps are followed as in the single line plate case. Connected component labeling with four neighboring is utilized and for found regions, region properties are specified. Regions whose extents are larger than 0.9, are enlarged by 25 pixels from top and bottom of the region to reduce the risk of cropping license plate characters. The algorithm investigates all possible regions and candidate plate regions are identified.

For filtering with restricted parameters case, connected component labeling is utilized and for regions whose width is in between 10 pixels to 60 pixels and length is in between 120 pixels to 215 pixels are extracted. If there is such a region, it is enlarged 5 pixels from top and bottom to reduce the risk of cropping license plate characters. Otherwise, the region is removed. In some plate regions, plate region becomes thinner due to some side effects such as illumination and plate region cannot be extracted with previous steps. The reason to process this step is to find region which contains semi thinner parts.

Recognition of the plate region is achieved for single line plate case. If the candidate plate regions are found for double line case, the LPDRS system stops and cannot recognize the license plate. If there are candidate plate region, LPDRS continues with identification of exact plate region step, otherwise LPDRS stops processing.

Flow diagram of the filtering algorithm can be seen in Figure 3-11.



**Figure 3-11 Flow Diagram of the Filtering Algorithm**

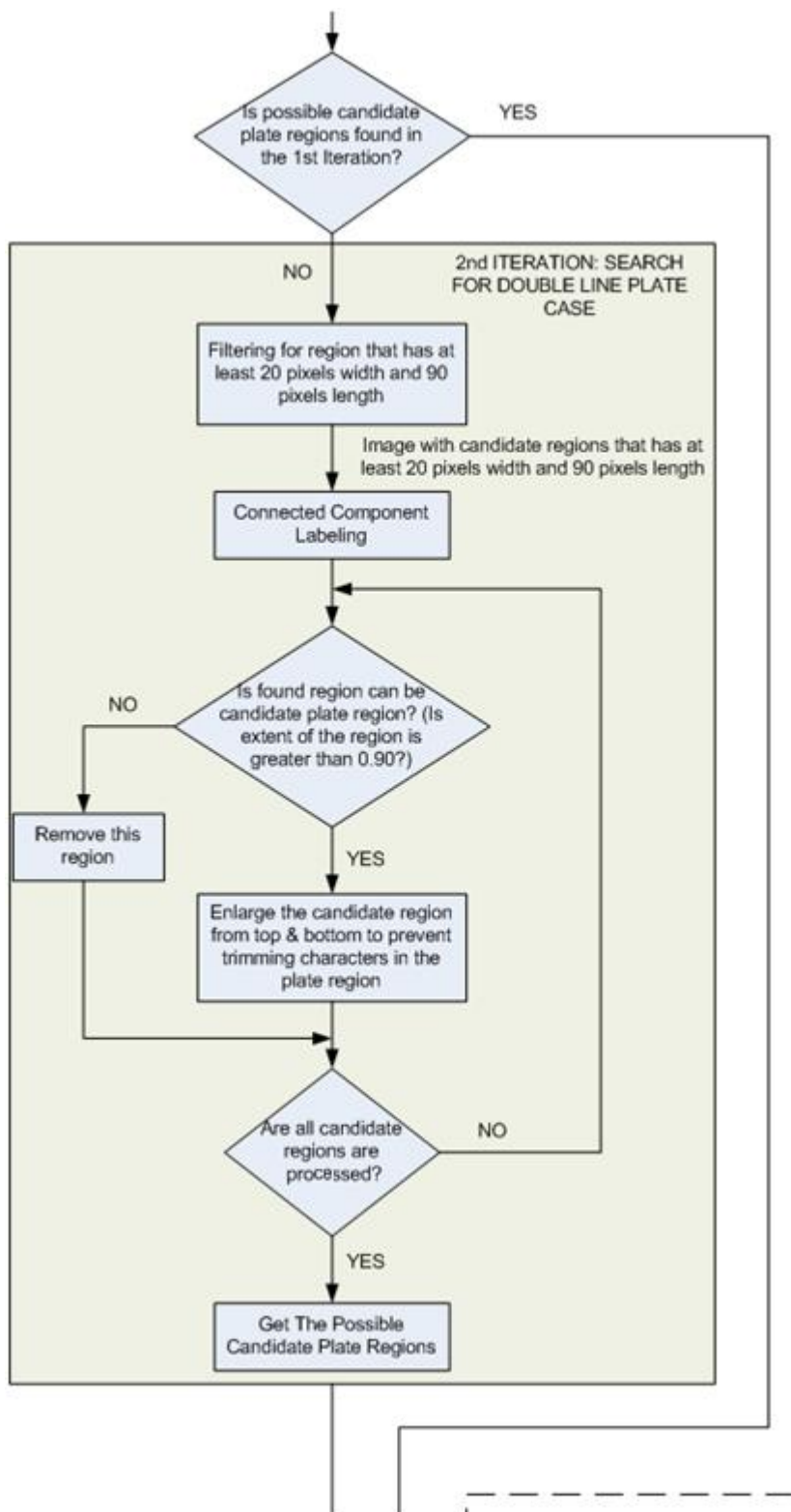
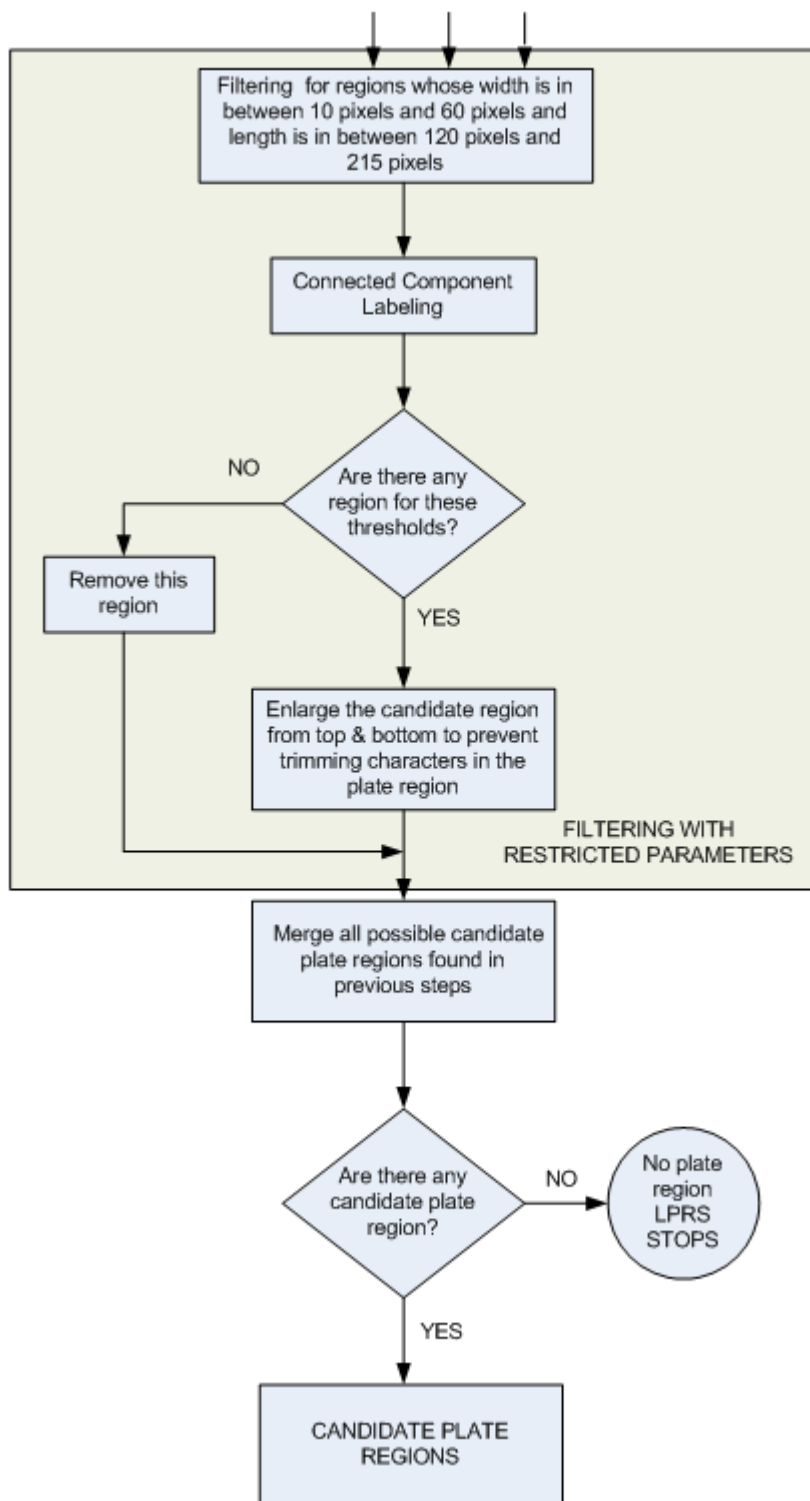
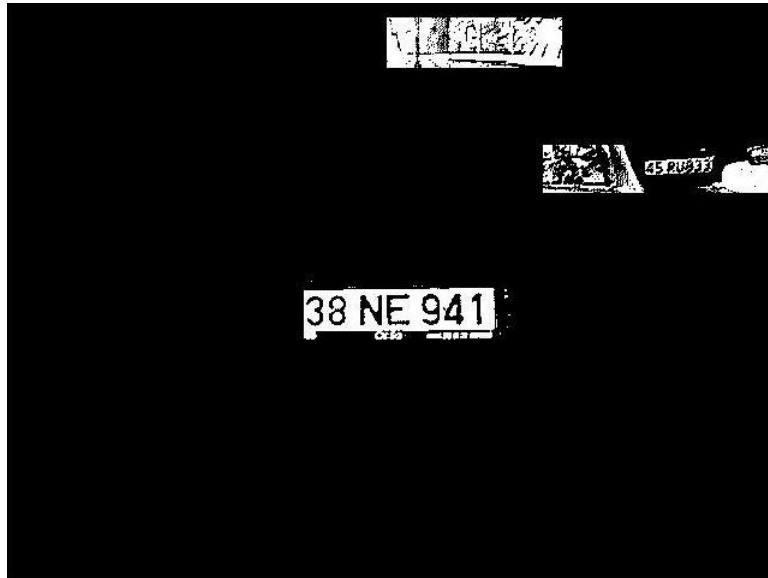


Figure 3-11 Flow Diagram of the Filtering Algorithm (Continued)



**Figure 3-11 Flow Diagram of the Filtering Algorithm (Continued)**

For the image, extracted candidate plate regions can be seen in Figure 3-12.



**Figure 3-12 Extracted Plate Regions**

#### **3.1.4.2 IDENTIFICATION OF EXACT PLATE REGION**

After candidate plate regions are extracted, connected component labeling with four neighboring is applied to the image and candidate region properties are specified. For each candidate region, white pixel density is calculated as in (3.14).

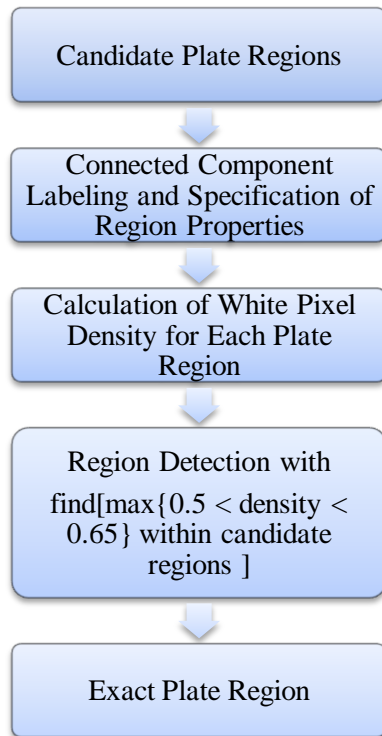
$$d_{white} = \frac{\sum_{x=1}^L \sum_{y=1}^W I(x,y)}{L*W} \quad (3.14)$$

where  $L$  and  $W$  represent the length and width of the candidate region, respectively.  $I$  represents the image of the candidate plate region, and  $x$  and  $y$  represent pixel positions in terms of  $x$  and  $y$  axes, respectively. In the numerator, summation gives how many bright pixels exist in the candidate region and denominator gives the total number of pixels.

White pixel density calculation is done for each candidate plate region. Exact plate region is detected by looking the white pixel density of each plate region. License plates used in the license plate detection and recognition system consist of black characters with white background. Therefore for actual license plates, it is expected that white pixel density should be larger than 0.5. To find actual threshold value for white pixel density, plate regions are analyzed and it is observed that white pixel density threshold should be at most 0.65.

To extract the plate region, maximum of the white pixel densities which is smaller than 0.65 is found and by this approach exact plate region is extracted. Regions which have




greater white pixel density than 0.65 are discarded. Flow diagram of the identification of exact plate region step is given in Figure 3-13.



**Figure 3-13 Flow Diagram of the Identification of Exact Plate Region**

Extracted plate region with calculated white pixel densities can be seen in Table 3-1.

**Table 3-1 Results for the Identification of Exact Plate Region Step**

Region ID	Candidate Region	White Pixel Density
1		0.5420 (Extracted Region)
2		0.7971
3		0.3830



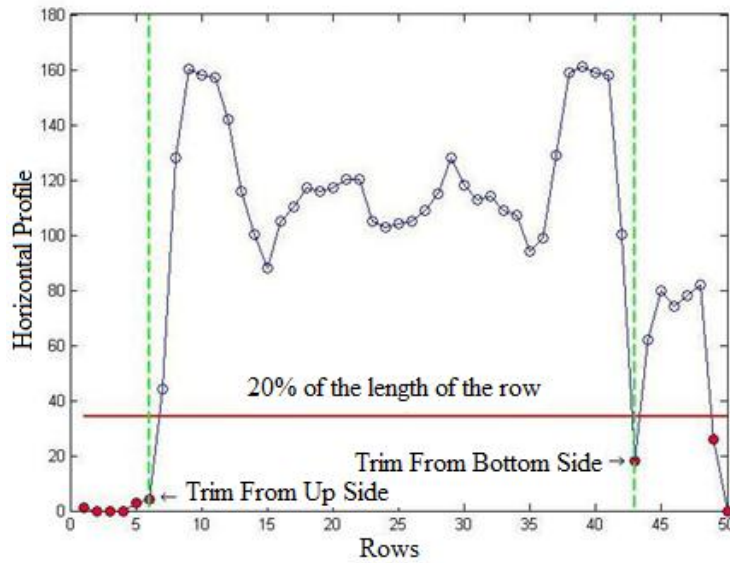
### **3.1.5 HORIZONTAL and VERTICAL TRIMMING**

After the exact plate region is extracted, vertical and horizontal trimming are performed if there are black pixels at the edges of the extracted plate region. Trimming is required because black pixels at the edges can be a misleading factor for the character segmentation step. As a result of black pixels at the edges, plate region may not be segmented properly in the character segmentation step.



Horizontal trimming, trimming from top and bottom of the image, is performed by using horizontal profiling. In horizontal profiling, for each row in the plate region, the number of bright pixels is found and behavior of this profile is compared with the ideal horizontal profile which is expected. Result of the comparison determines whether the trimming will be performed or not and if the trimming will be performed, the decision of which side should be trimmed is given.

In ideal case, black pixels at the edges are not expected and there should be bright regions before characters start. Therefore, in ideal horizontal profile, there should be high values at top and bottom sides, which represent the number of bright pixels. In the middle rows, horizontal profile values should be less than up and down side edge values and there should be fluctuations due to black pixels' of the characters.

In horizontal trimming, for each row, the number of bright pixels for that row is found. 20% tolerance is given for each row because there may be black pixels right and/or left-hand sides, which are not trimmed yet. Index values of rows, whose horizontal profile values are smaller than 20% of the length of the row, are found and horizontal trimming is performed whether from top or bottom or both. Decision algorithm is applied to determine which side should be trimmed. If the difference between index values is greater than 10 pixels, it means both up and down trim should be performed. Otherwise, if the difference between index values is smaller than 10 pixels and there is an index value, whose value is smaller than 15 that indicates row at that region, it means up side trimming will be performed. If the difference between the index values is smaller than 10 pixels, the up side trimming is not performed and the index value is greater than 25 pixel, it means down side trimming will be performed. Horizontal profile values and index values for trimming can be seen in Figure 3-14 and Figure 3-15.



**Figure 3-14 Horizontal Profiling Results**

Before Horizontal Trimming	After Horizontal Trimming
	

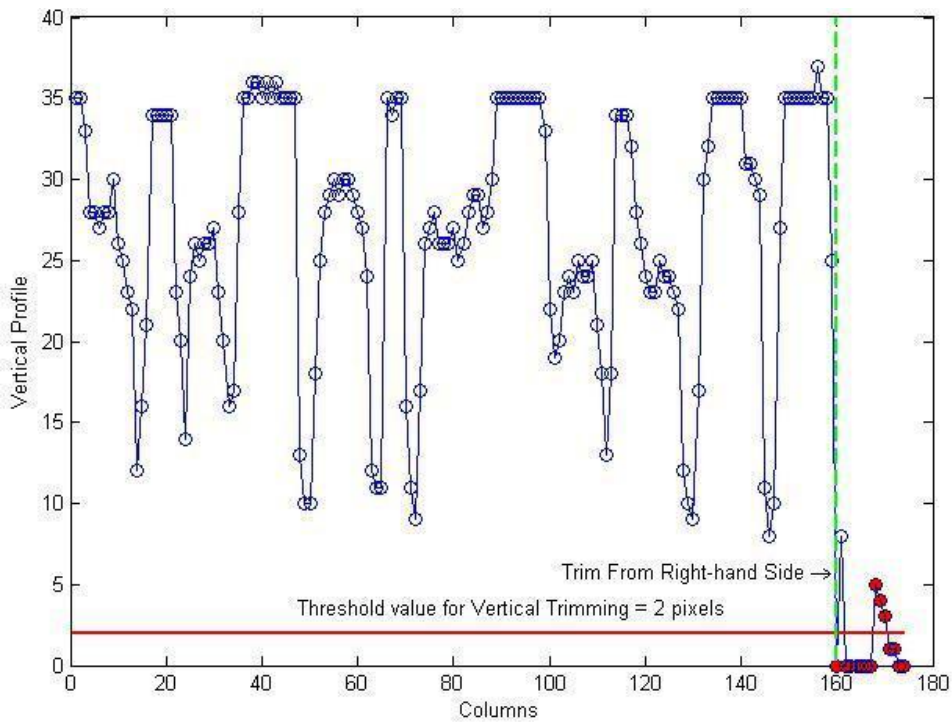
**Figure 3-15 Before and After Horizontal Trimming**

After the horizontal trimming, vertical trimming, trimming from right and left-hand side, is performed if required. In vertical trimming same approach is used as in the horizontal profiling. Vertical trimming is made use of the vertical profiling. In vertical profiling, for each column in the plate region, the number of bright pixels is found and behavior of this profile is compared with the ideal vertical profile that is expected. Result of the comparison determines whether the trimming will be performed or not and if the trimming will be performed, the decision of which side should be trimmed is given.


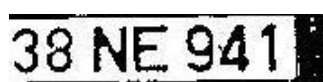

In ideal case, bright pixel regions are expected at the right and left-hand sides. If there are black pixels, these pixels are thought as characters and they are also segmented in a wrong manner. In middle columns, fluctuations occur due to black pixels of characters.

In vertical trimming, for each column, the number of bright pixels for that column is found. 2 pixels tolerance is given for each column because there may be still black pixels after the horizontal trimming at the top and bottom. The index values of columns, whose vertical profile values are smaller than 2 pixels, are found and vertical trimming is performed whether from right or left-hand side or both. Decision algorithm is applied to determine which side should be trimmed.

If the difference between index values is greater than 15 pixels, it means both right and left-hand side trim should be performed. Otherwise, if the difference between index values is smaller than 15 pixels and there is an index value, whose value is smaller than 25, which represents the column number at that region, it means left-hand side trimming will be performed. If the difference between index values is smaller than 15 pixels, the left-hand side trimming will not be performed and the index value is greater than the number of columns minus 25 pixels, it means right-hand side trimming will be performed. Vertical profile values and index values for trimming can be seen in Figure 3-16 and Figure 3-17, respectively.



**Figure 3-16 Vertical Profiling Results**

Before Horizontal Trimming	After Horizontal Trimming and Before Vertical Trimming	After Vertical Trimming ( Plate Used for Segmentation)
		

**Figure 3-17 Before and After Trimming Operation**

After trimming operation is performed, plate region can be used in the character segmentation step.

### 3.2 CHARACTER SEGMENTATION

In the license plate detection and recognition system, license plate is recognized character by character. Therefore, segmentation of each character is required. Before the recognition of each character, character on the plate region is segmented and prepared for the recognition step. Then, it is recognized by the recognition algorithm. Later, another character is segmented, prepared and recognized. Segmentation and recognition is applied character by character.

Character segmentation step consists of two main substeps. In the first step, characters on the plate region are segmented. In the second step, morphological operations are applied to the binary image in order to remove unwanted pixels.

#### 3.2.1 NOTCH DETECTION and SEGMENTATION OF EACH CHARACTER

In order to segment the plate region character by character, vertical projection of the plate region is used. Due to some discontinuities at the top and bottom of the image, there may be still some black pixels. To eliminate the adverse effect of the top and bottom rows which may have black pixels that may be missed while trimming, middle row is taken as a reference and the last 90% of the upper row from top to middle and the first 90% of the lower rows from middle to bottom is used for vertical projection. For the vertical projection, the number of black pixels is counted for each column. DC value of the vertical projection is not important. For the notch detection, sharp peaks are taken into account. Therefore, DC level of the projection can be removed. In order to remove the DC level, mean of the vertical projection is calculated and then subtracted from itself. Vertical projection of the plate region without DC level can be seen in Figure 3-18.

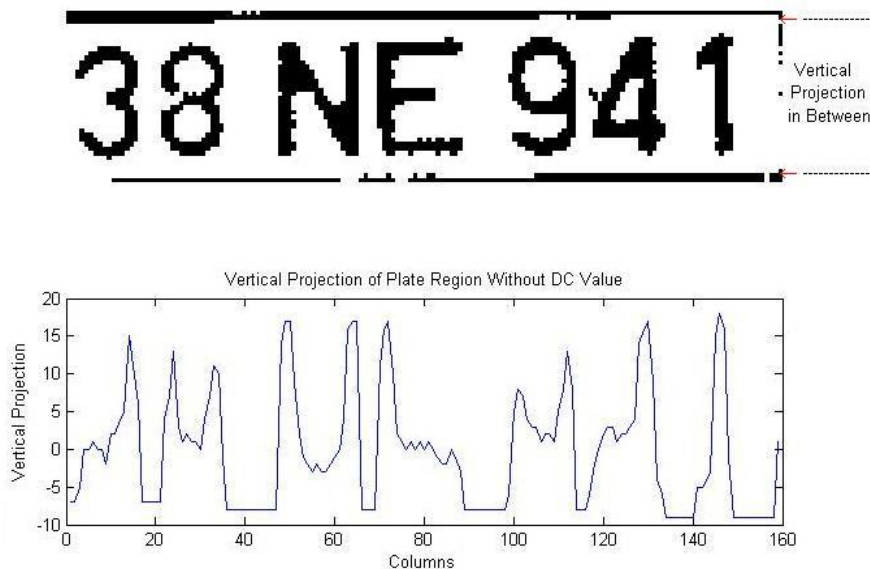


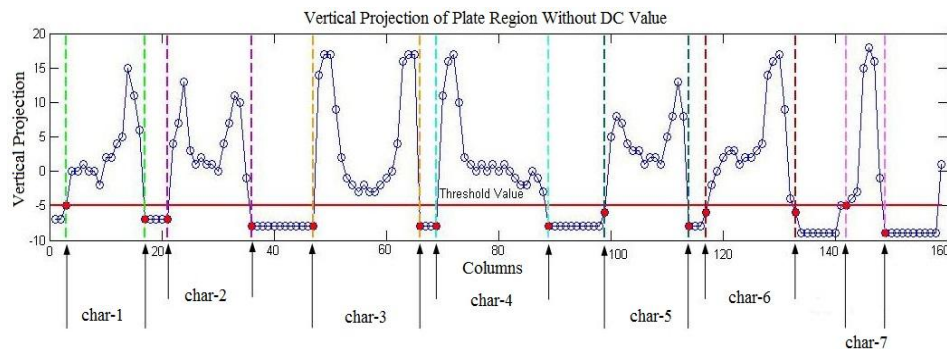
Figure 3-18 Vertical Projection of the Plate Region without DC value

As it can be seen in the Figure 3-18, silhouettes of each character can be seen in the vertical projection diagram. These notches are used to segment each character.

For the notch detection, values of the vertical projection which are smaller or equal to the half of the minimum ripple value are used. Index values which are smaller or equal to the half of the minimum ripple value are found which can be seen below the red line in Figure 3-19. In Figure 3-19, threshold value represents the half of the minimum ripple value. After index values are specified, they are analyzed that whether they construct a notch or not. To determine notches, differences between the successive index values are investigated. If the difference between the successive indexes equal or greater than 6, it means that there is a notch, which indicates a character on the plate region. Other notches in the plate region are found similarly. The reason to examine the difference between the successive indexes is because; there are always bright pixel regions between characters. These threshold value specified as 6 indicates width of the minimum required bright pixel area between the characters.

To specify the location of the notch, two index values are required. One index value represents the notch start index and one for the notch end index. After notch start and end index values are specified, these character regions are segmented before the recognition part.

Vertical projection of plate region without DC value, and notch extractions with start and end index values can be seen in Figure 3-19.



**Figure 3-19 Notch Extraction with Vertical Projection**

After the start and end index values of notches are found, these characters represented by notches can be extracted and the number of characters, which is the half of the number of notches, in the plate region, is found.

### 3.2.2 MORPHOLOGICAL OPERATION

After the character segmentation, there may be some black pixels missed while trimming at the edges. Due to effect of these pixels, character may not be recognized or may be recognized in a wrong manner. Therefore, these pixels should be removed. To eliminate these pixels, connected component labeling with four neighboring is applied and region properties are extracted. The region with larger size gives the region which has black pixels for only character. To obtain smooth contour of the character, morphological opening is applied. First, binary image of each character is eroded then it is dilated with 2x2 structuring element. Effect of connected component labeling on segmented characters and results after the morphological opening can be seen in Figure 3-20.

Segmented Characters Before Connected Component Labeling Analysis							
Segmented Characters After Connected Component Labeling Analysis							
Segmented Characters After Morphological Opening							

Figure 3-20 Results for the Segmented Characters

### 3.3 CHARACTER RECOGNITION

Character recognition steps can be divided into two substeps. First, codeword of each character is generated to represent the character as a code. Then, generated codeword is matched with all codewords kept in the database. If there is a match, character is recognized. If there is more than one match, character is identified with identification algorithms and if there is no match, character cannot be recognized.

#### 3.3.1 CODEWORD GENERATION

To decrease the complexity of the matching, codewords are used instead of matching the segmented character images with the ideal expected character images [26]. Each codeword represents a character. Codewords are not unique. Two different characters may be

represented with same codeword; therefore, identification algorithms are required to identify characters.

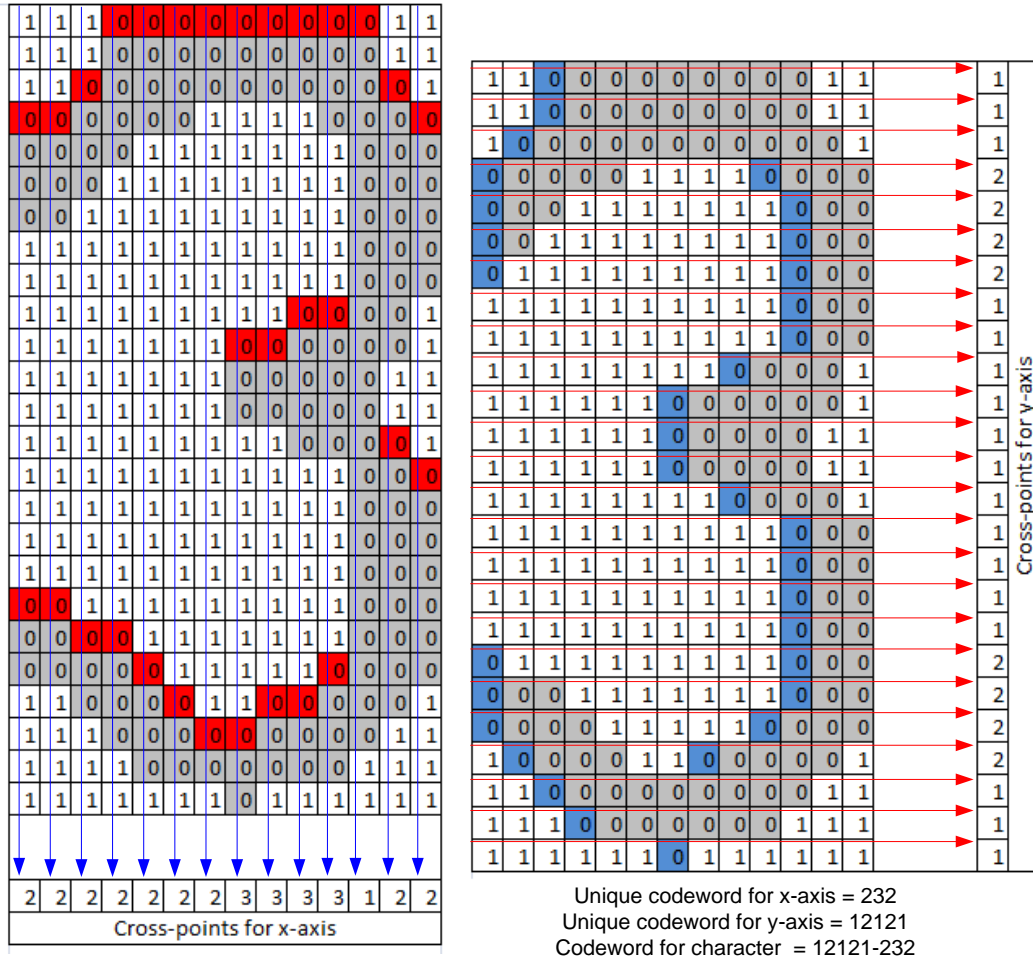
To generate a codeword of the character, for x-axis and y-axis, two codewords are generated and then they are concatenated.

In x-axis codeword generation, for each column in the character image, how many times black pixels are crossed is counted and this value constructs cross point for single column. For all columns, cross points are calculated and concatenated. Values in the concatenated cross-point array are made unique. They are made unique because successive values cannot be same. The reason to make unique the cross-point array is to eliminate repeated values. Cross-point values should be repeated at least two times successively, to put this value to unique array. This unique cross-point array constructs the codeword for x-axis.

In y-axis codeword generation, for each row in the character image, how many times black pixels are crossed is counted and this value constructs cross point for single row. For all rows, cross points are calculated and concatenated. Values in the concatenated cross-point array are made unique as in the same manner with x-axis codeword generation. Obtained unique cross-point array constructs the codeword for y-axis.

After both x-axis codeword and y-axis codeword are generated, x-axis codeword is concatenated with y-axis codeword and codeword of the character is constructed.

Cross-point calculations and generated codeword values can be seen in Figure 3-21.



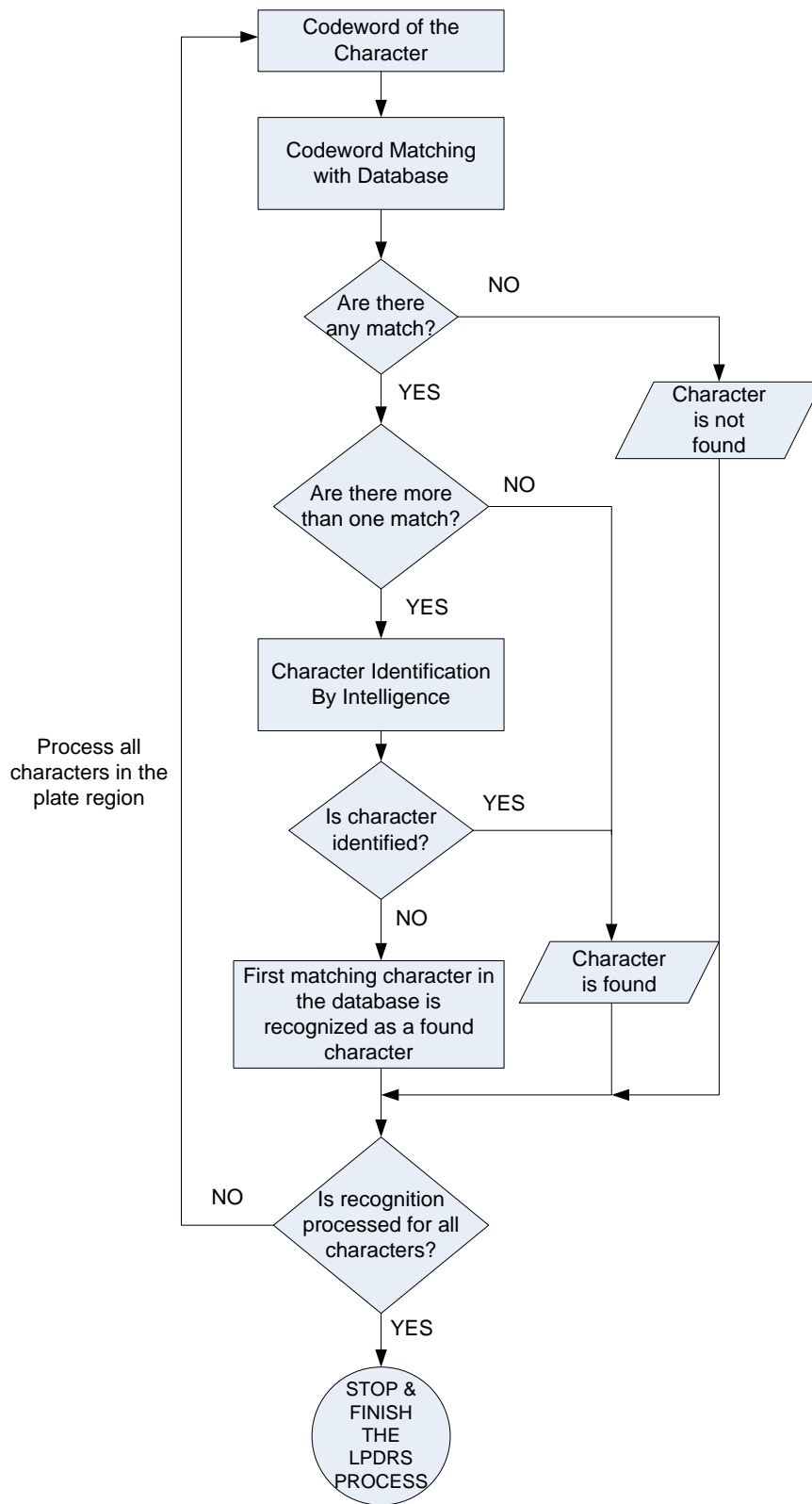
**Figure 3-21 Cross-point and Codeword Calculations**

### 3.3.2 CODEWORD MATCHING and RECOGNITION

Recognition is simply a comparison of features of template and the candidate. The best matching template gives the recognition result. In this thesis, codeword generated in the previous step is compared with the ideal codewords kept in the database. Different characters may be represented with same codeword. Therefore, to distinguish these characters, identification algorithms are implemented for special cases such as to distinguish '2' from 'Z'. If the algorithm is not able to distinguish the character, first matching character in the database is recognized as a found character. If there is no match, character cannot be recognized.

Flow diagram of the recognition steps can be seen in Figure 3-22.





**Figure 3-22 Flow Diagram of the Recognition**

In the recognition step of the LPDRS, if the template is matched with the candidate, ASCII code of that character which is specified in the database is assigned as the ASCII code of the candidate. After matching of codewords, identification algorithms are processed by using the ASCII codes of the recognized character and image of the segmented character.

### 3.3.2.1 IDENTIFICATION OF LETTER AND NUMBER

In Turkish license plates, first two and last two characters should be numbers. To decrease the probability of false recognition, algorithm for the discrimination of letter and number is implemented. ASCII codes of the first two and last two characters should be smaller than 58, because ASCII codes of numbers are represented in between 48 and 57 in decimal. If ASCII code of these characters is equal or greater than 58, which means the character is not a number, this ASCII code is eliminated from found recognition result.

Moreover, in Turkish license plates, third character must be a letter. Therefore, ASCII code of the third character should be equal or greater than 65 which represents 'A'. The corresponding ASCII codes of the letters in the Turkish alphabet are greater than 64. For the third character, if the confusion occurs, character is identified according to whether it's ASCII code is greater than 64 or not. In the license plates, fourth and fifth characters may be number or letter. However, it is also known that if the fifth character is a letter, the fourth character should also be a letter. In the identification of letter and number algorithm, if the ASCII code of the fifth character is greater than 64, ASCII code of the fourth character should be also greater than 64 otherwise this ASCII code is eliminated from found recognition result.

### 3.3.2.2 IDENTIFICATION OF 0, 4 AND 7

'0', '4' and '7' numbers share the same codeword which is 121-121. These numbers can also be represented with different codewords found in the database. If the codeword of the segmented character is found as 121-121, the segmented character can be '0', '4', '7', 'D' or 'P'. If the segmented character is one of the first two or last two characters in the plate region, possibility of being 'D', 'O' or 'P' is eliminated by letter and number identification algorithm. After possibility of being 'D', 'O' or 'P' is eliminated, these numbers are distinguished by "047" identification algorithm. ASCII codes of these numbers are 48, 52 and 55, respectively.

In order to identify these numbers, for each character segment some samples are taken from previously specified rows and cross-points are calculated for these rows. By investigating the value of cross-points for specified rows, numbers are distinguished.

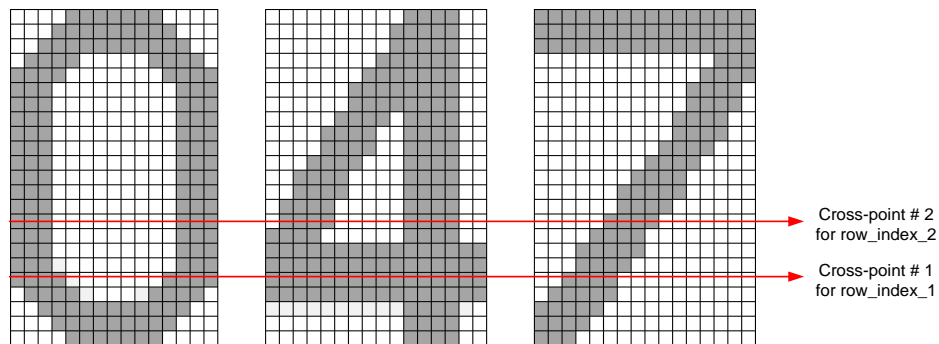
To distinguish '0', '4' and '7', first, row index values from which samples are taken are calculated. Row index values are calculated as in (3.15) and (3.16).

$$row_{index1} = length_{character} - floor\left(\frac{length_{character}}{5}\right) \quad (3.15)$$

$$row_{index2} = length_{character} - floor\left(2 * \frac{length_{character}}{5}\right) \quad (3.16)$$

where  $row_{index1}$  and  $row_{index2}$  represent row indexes from which samples are taken and  $length_{character}$  represents the length of the binary image of the segmented character. Length of the character is divided by five, because when the ideal '0', '4' and '7' numbers are analyzed, it is observed that these numbers can be distinguished by investigating the cross-points for these calculated rows.

For ideal images of '0', '4' and '7' numbers with 23 pixel image length, the position of row index values for which cross-points are calculated can be seen in Figure 3-23.



**Figure 3-23 Position of row index values for cross-point calculations of “047”**

By analyzing the cross-point values, decision of “047” identification is given according to Table 3-2.

**Table 3-2 Decision Table for Identification of ‘0’, ‘4’ and ‘7’**

<b>Cross-point #1</b>	2	1	1	2
<b>Cross-point #2</b>	2	2	1	1
<b>Recognition Result</b>	Recognized as ‘0’	Recognized as ‘4’	Recognized as ‘7’	Character cannot be recognized

### 3.3.2.3 IDENTIFICATION OF U, V AND Y

‘U’, ‘V’ and ‘Y’ letters share same codewords which are 121-21 and 1-21. ASCII codes of these letters are 85, 86 and 89, respectively.

In order to identify these letters, for each character segment, samples are taken from previously specified rows and cross-points are calculated for these rows. By investigating the value of cross-points for specified rows, letters are distinguished.

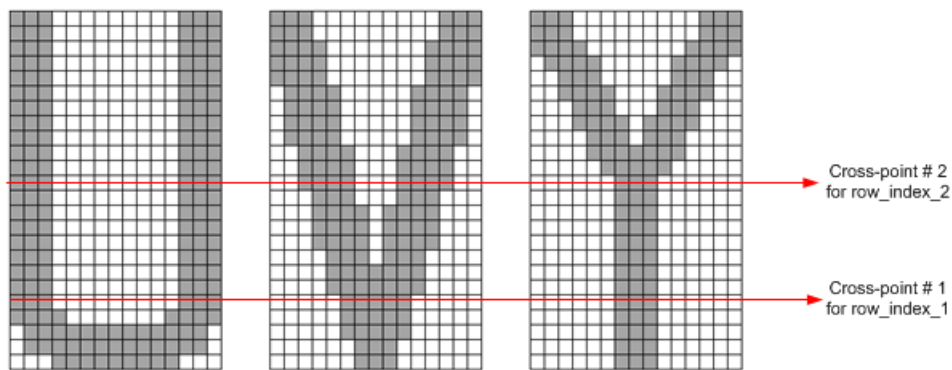
To distinguish ‘U’, ‘V’ and ‘Y’, first, row index values from which samples are taken are calculated. Row index values are calculated as in (3.17) and (3.18).

$$row_{index1} = length_{character} - floor\left(\frac{length_{character}}{5}\right) \quad (3.17)$$

$$row_{index2} = floor\left(\frac{length_{character}}{2}\right) \quad (3.18)$$

where  $row_{index1}$  and  $row_{index2}$  represent row indexes from which samples are taken and  $length_{character}$  represents the length of the binary image of the segmented character. Divisor of the length of the binary image is found by analyzing the ideal ‘U’, ‘V’ and ‘Y’ letters.

For ideal images of ‘U’, ‘V’ and ‘Y’ numbers with 24 pixel image length, the position of row index values for which cross-points are calculated can be seen in Figure 3-24.



**Figure 3-24 Position of row index values for cross-point calculations of “UVY”**

By analyzing the cross-point values, decision of “UVY” letter identification is given according to Table 3-3.

**Table 3-3 Decision Table for Identification of ‘U’, ‘V’ and ‘Y’**

<b>Cross-point #1</b>	2	1	1	2
<b>Cross-point #2</b>	2	2	1	1
<b>Recognition Result</b>	Recognized as ‘U’	Recognized as ‘V’	Recognized as ‘Y’	Character cannot be recognized

### 3.3.2.4 IDENTIFICATION OF 3, 5 AND 8

‘3’, ‘5’ and ‘8’ have the same codeword which is 232-12121. If the codeword of the segmented character is found as 232-12121, the segmented character can be ‘3’, ‘5’, ‘8’ or ‘S’. If the segmented character is one of the first two or last two characters in the plate region, possibility of being ‘S’ is eliminated by letter and number identification algorithm. After letter elimination is applied, these numbers are distinguished by “358” identification algorithm. ASCII codes of these numbers are 51, 53 and 56, respectively.

In order to identify these numbers, for each character segment two samples are taken for two row index values and cross-points are calculated for these rows. By investigating the value of cross-points for specified rows, numbers are distinguished.

To distinguish ‘3’, ‘5’ and ‘8’, first, row index values from which samples are taken are calculated. Row index values are calculated as in (3.19) and (3.20).

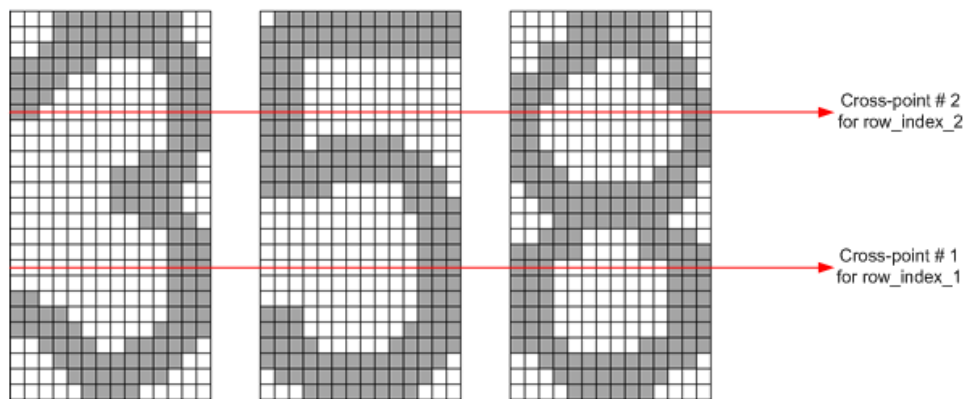
$$row_{index1} = length_{character} - floor\left(\frac{length_{character}}{3}\right) \quad (3.19)$$

$$row_{index2} = floor\left(\frac{length_{character}}{3}\right) - 1 \quad (3.20)$$

where  $row_{index1}$  and  $row_{index2}$  represent row indexes from which samples are taken and  $length_{character}$  represents the length of the binary image of the segmented character. Divisor of the length of the binary image is found by analyzing the ideal ‘3’, ‘5’ and ‘8’ numbers.

By the analysis of the cross-points for these calculated rows, the numbers can be distinguished.

For ideal images of ‘3’, ‘5’ and ‘8’ numbers with 25 pixel image length, the position of row index values for which cross-points are calculated can be seen in Figure 3-25.



**Figure 3-25 Position of row index values for cross-point calculations of “358”**

By analyzing the cross-point values, decision of “358” number identification is given according to Table 3-4.

**Table 3-4 Decision Table for Identification of ‘3’, ‘5’ and ‘8’**

<b>Cross-point #1</b>	2	1	1	2
<b>Cross-point #2</b>	2	2	1	1
<b>Recognition Result</b>	Recognized as ‘8’	Recognized as ‘3’	Recognized as ‘5’	Character cannot be recognized

**3.3.2.5 IDENTIFICATION OF 3, 8 AND 9**

‘3’, ‘8’ and ‘9’ numbers have the same codeword which is 231-12121. If the codeword of the segmented character is found as 231-12121, the segmented character can be ‘3’, ‘8’ or ‘9’. ASCII codes of these numbers are 51, 56 and 57, respectively.

In order to identify these numbers, for each character segment samples are taken from previously specified rows and cross-points are calculated for these rows. By investigating the value of cross-points for specified rows, numbers are distinguished.

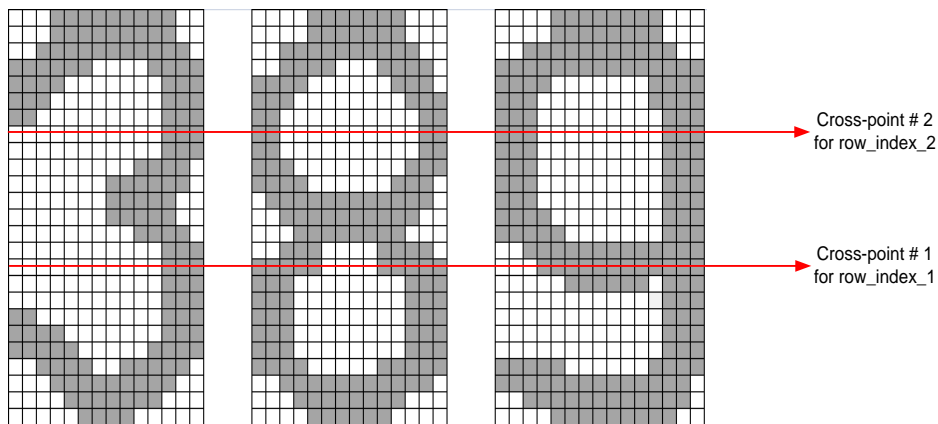
To distinguish ‘3’, ‘8’ and ‘9’, first, row index values from which samples are taken are calculated. Row index values are calculated as in (3.21) and (3.22).

$$row_{index1} = 2 * floor\left(\frac{length_{character}}{3}\right) \tag{3.21}$$

$$row_{index2} = floor\left(\frac{length_{character}}{3}\right) \tag{3.22}$$

where  $row_{index1}$  and  $row_{index2}$  represent row indexes from which samples are taken and  $length_{character}$  represents the length of the binary image of the segmented character. Divisor of the length of the binary image is found by analyzing the ideal ‘3’, ‘8’ and ‘9’ letters.

For ideal images of ‘3’, ‘8’ and ‘9’ numbers with 25 pixel image length, the position of row index values for which cross-points are calculated can be seen in Figure 3-26.



**Figure 3-26 Position of row index values for cross-point calculations of “389”**

By analyzing the cross-point values, decision of “389” number identification is given according to Table 3-5.

**Table 3-5 Decision Table for Identification of ‘3’, ‘8’ and ‘9’**

<b>Cross-point #1</b>	1	2	1	2
<b>Cross-point #2</b>	1	2	2	1
<b>Recognition Result</b>	Recognized as ‘3’	Recognized as ‘8’	Recognized as ‘9’	Character cannot be recognized

### 3.3.2.6 IDENTIFICATION OF 6 AND B

‘6’ and ‘B’ characters have the same codeword which is 132-12121. If 132-12121 codeword is obtained in one of the first two, last two characters or in the third character in the plate region, these characters can be distinguished by letter and number identification algorithm. Otherwise, “6B” identification algorithm is used to distinguish these characters. ASCII codes of these characters are 54 and 66, respectively.

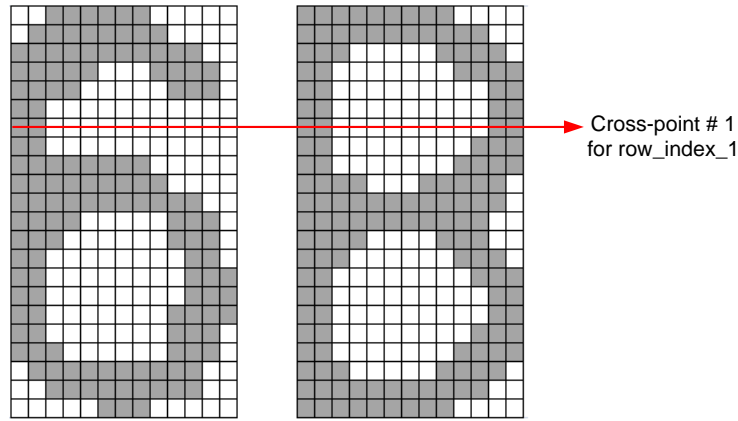
In order to identify these characters, for each character segment samples are taken from previously specified row and cross-point is calculated for that row. By investigating the value of cross-point for the specified row, characters are distinguished.

To distinguish ‘6’ and ‘B’, first, row index value from which sample is taken is calculated. Row index value is calculated as in (3.23).

$$row_{index1} = floor\left(\frac{length_{character}}{3}\right) \quad (3.23)$$

where  $row_{index1}$  represents the row index from which sample is taken and  $length_{character}$  represents the length of the binary image of the segmented character. Divisor of the length of the binary image is found by analyzing the ideal ‘6’ and ‘B’ characters.

For ideal images of ‘6’ and ‘B’ characters with 22 pixel image length, the position of row index value for which cross-point is calculated can be seen in Figure 3-27.



**Figure 3-27 Position of row index value for cross-point calculation of “6B”**

By analyzing the cross-point value, decision of “6B” character identification is given according to Table 3-6.

**Table 3-6 Decision Table for Identification of ‘6’ and ‘B’**

<b>Cross-point #1</b>	1	2
<b>Recognition Result</b>	Recognized as ‘6’	Recognized as ‘B’

### 3.3.2.7 IDENTIFICATION OF 2 AND Z

‘2’ and ‘Z’ characters share the same codewords which are 232-121, 232-1 and 32-1. If these codewords are obtained in one of the first two, last two characters or in the third character in the plate region, these characters can be distinguished by letter and number identification algorithm. Otherwise, “2Z” identification algorithm is used to distinguish these characters. ASCII codes of these characters are 50 and 90, respectively.

In order to identify these characters, for each character segment samples are taken from previously specified row and cross-point is calculated for that rows. By investigating the value of cross-point for the specified row, characters are distinguished.

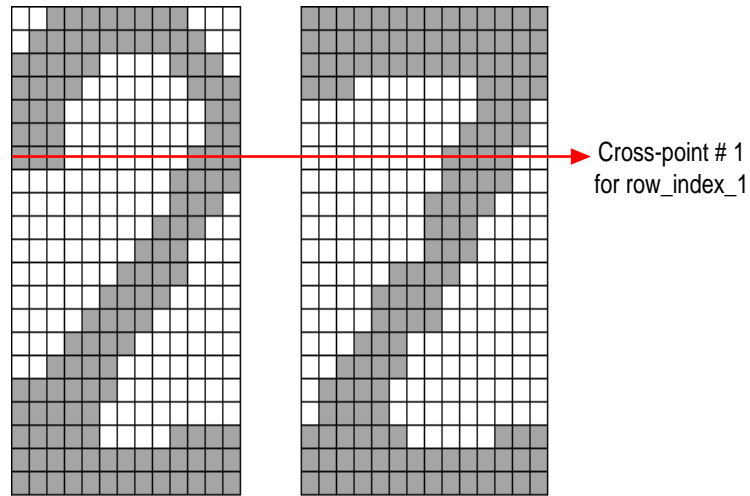
To distinguish ‘2’ and ‘Z’, first, row index value from which sample is taken is calculated. Row index value is calculated as in (3.24).

$$row_{index1} = floor\left(\frac{length_{character}}{3}\right) \quad (3.24)$$

where  $row_{index1}$  represents the row index from which sample is taken and  $length_{character}$  represents the length of the binary image of the segmented character. Divisor of the length of the binary image is found by analyzing the ideal ‘2’ and ‘Z’ characters.



For ideal images of ‘2’ and ‘Z’ characters with 21 pixel image length, the position of row index value for which cross-point is calculated can be seen in Figure 3-28.



**Figure 3-28 Position of row index value for cross-point calculation of “2Z”**

By analyzing the cross-point value, decision of “2Z” character identification is given according to Table 3-7.

**Table 3-7 Decision Table for Identification of ‘2’ and ‘Z’**

<b>Cross-point #1</b>	1	2
<b>Recognition Result</b>	Recognized as ‘Z’	Recognized as ‘2’



## **CHAPTER 4**

### **RECOGNITION OF LICENSE PLATE USING FPGA HARDWARE**

License plate detection and recognition algorithm, explained in Chapter 3, is built and verified in MATLAB. After verification, character segmentation and character recognition parts of the LPDRS are implemented in Altera DB\_START\_4CE10-Cyclone IV E Development Board.

FPGA hardware implementation of the recognition algorithm consists of logical blocks and soft processor embedded in the FPGA. Logical blocks and soft processor are synthesized and programming file is generated. FPGA hardware and soft processor application file are programmed into the serial flash with a small batch file and Quartus Programmer.

For the design and simulation of the system, Altera Quartus II, Modelsim Simulation Software and Nios II Software Build Tools for Eclipse are used. Altera Signaltap II Logic Analyzer is used to debug the system in real time.

#### **4.1 HARDWARE ARCHITECTURE**

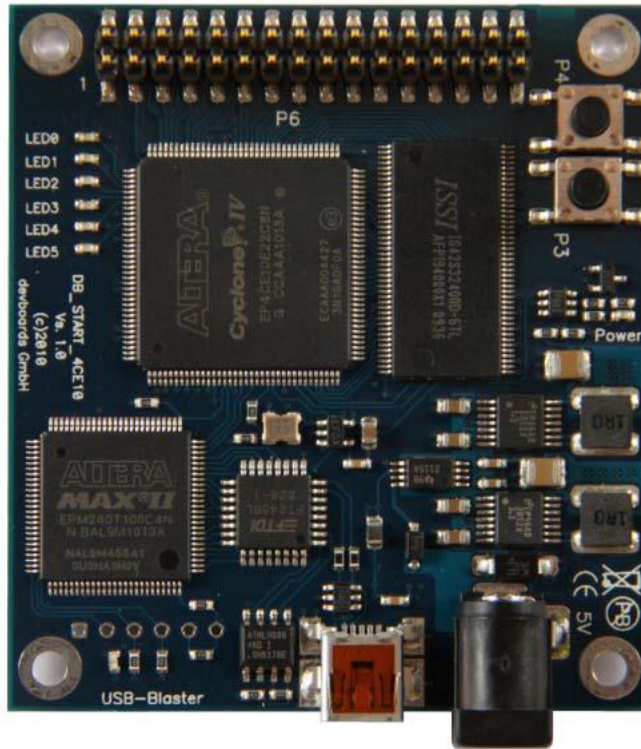
Recognition part of the license plate detection and recognition system is implemented on the FPGA platform by using the Altera DB\_START\_4CE10 Cyclone IV E Development Board. Development board used in this thesis is small sized, low cost and low power board and it provides easy usage and peripheral adaptation for the user. The main reason to use this board is to obtain low cost implementation.

The following features are integrated in the development board [7]:

- EP4CE10E22C8N Cyclone IV E FPGA
- EPCS16 serial flash as a configuration device
- 16 Mbyte SDRAM
- 19 I/O pins
- 5x input pins
- 6x user LEDs
- 2x users buttons
- JTAG interface
- Crystal oscillator
- Embedded USB Blaster for debugging and programming
- EPM240T100C5N Max II CPLD for embedded USB Blaster

- Power supply with integrated USB
- Small size, dimension : 70 x 60 mm

Top view of the development board can be seen in Figure 4-1.



**Figure 4-1 Top View of the Development Board**

#### **4.1.1 COMPONENTS USED IN THE SYSTEM**

For the implementation of the character segmentation and character recognition parts of the LPDRS, FPGA, Serial Flash, SDRAM, Buttons, and User I/Os are used. The functions of these components are explained in this section.

##### **4.1.1.1 CYCLONE IV E FPGA**

In this thesis, Altera Cyclone IV E FPGA, EP4CE10E22C8N, is used. Cyclone IV E devices operate at maximum 472.5 MHz input clock frequency. Soft processor can be applied in most of the Altera FPGAs. In this thesis, soft core processor is implemented in the FPGA and image processing algorithms are processed. The image of the license plate is received by serial port which is constructed with the user I/Os and image processing algorithms are implemented to recognize the license plate. Some parts of the image processing algorithms are processed by FPGA logic blocks and some parts are processed by the soft core processor.

Nios II processor is used as a soft core processor which is the most widely used soft processor in the FPGA industry. The Nios II processor delivers unprecedented flexibility for cost-sensitive, real-time, safety-critical, ASIC optimized and application processing needs. Detailed information about Nios II processor is given in the Appendix A.

#### **4.1.1.2 EPCS16 SERIAL FLASH AS CONFIGURATION DEVICE**

FPGA devices in Altera Company have RAM based configuration memories which are volatile and therefore, when the power goes down, the configuration memory is cleared. With RAM based devices that support active serial configuration, configuration data must be reloaded each time the device powers up, the system reconfigures, or when new configuration data is required. Serial configuration devices are flash memory devices with a serial interface that can store configuration data for FPGA which support active serial configuration and reload the data to the device upon power up or reconfiguration.

In Altera DB\_START\_4CE10-Cyclone IV E development board, Altera serial configuration device, EPCS16, is used as a configuration memory. At power up, the configuration file is loaded from serial flash to the FPGA and the program starts to run. In software embedded systems using soft processor, software program is also stored in the serial flash device and software application is loaded at power up.

EPCS16 is 16 Mbit serial flash memory device which uses the active serial configuration scheme to configure the FPGA. Since it has four pin interface, it is easy to use.

#### **4.1.1.3 SDRAM**

Altera DB\_START\_4CE10-Cyclone IV E development board has 128Mbit Synchronous DRAM. The IS42S32400A device is a 4Mx32 bit SDRAM which is used for the application memory. After the software application is loaded from the flash device to the SDRAM, application will start to run on the SDRAM.

#### **4.1.1.4 BUTTONS and LEDS**

There are two buttons on the development board. One of these buttons is used as the system reset. This button resets the overall system and the other button is reserved for the user implementation.

LED interface is used to indicate the reset case at reset time. After the reset, control of the led interface is given to user.

#### **4.1.1.5 USER I/Os**

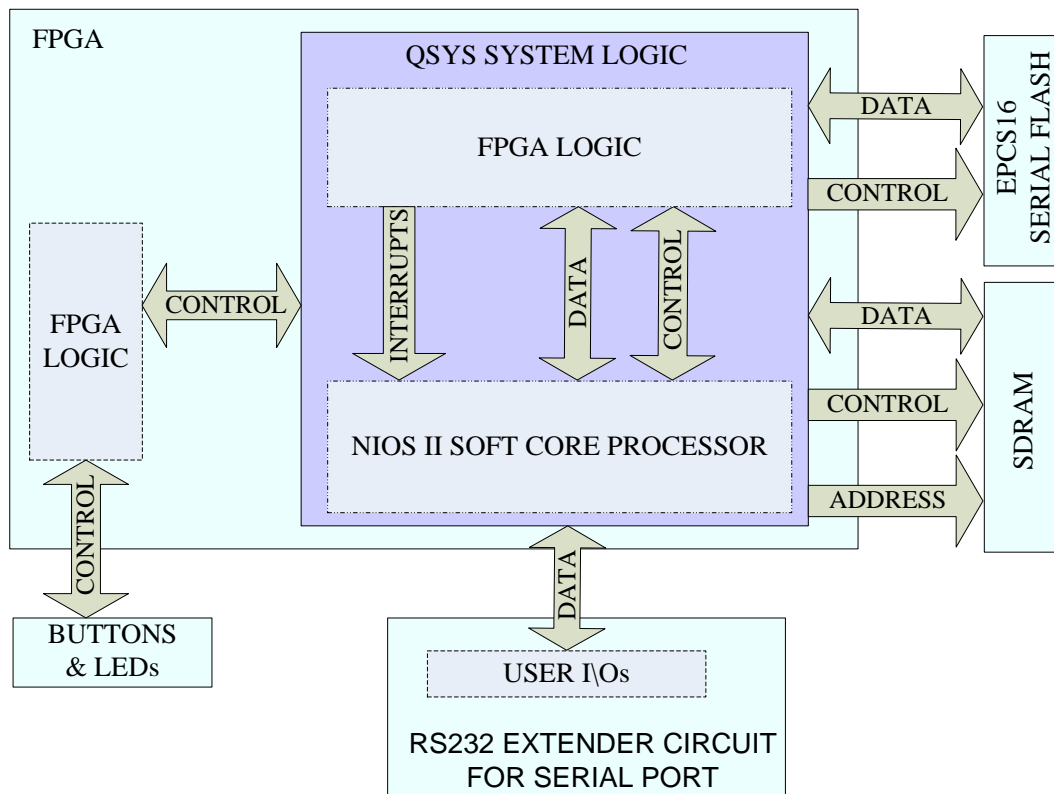
In the development board, there is a pin header with 32 signals directly connected to the FPGA. In this thesis, this pin header is used to construct a serial port. Pins used for communication require transceiver to adjust the voltage levels of the communication line. At transmitter side, this transceiver converts from the UART's logic levels to RS-232

compatible signal levels, and at receiver side converts RS-232 compatible signals to UART's logic levels.

To obtain RS-232 compatible signals, RS-232 extender circuit is designed and connected to the development board with the user I/Os. Schematic of the RS-232 extender circuit is given at the Appendix B.

#### 4.1.2 BLOCK DIAGRAM OF THE HARDWARE ARCHITECTURE

Block diagram of the hardware architecture can be seen in Figure 4-2.



**Figure 4-2 Hardware Architecture of the System**

In EPCS16 non-volatile serial flash device, both programming data for FPGA and software application for the Nios II processor are stored. At power up, FPGA is configured from flash. After configuration, software application is read from the serial flash and copied to the SDRAM. After the copy operation is finished, program counter is set to the starting address of the SDRAM and application starts to execute.

#### 4.2 FPGA ARCHITECTURE

In this thesis, character segmentation and character recognition parts of the license plate detection and recognition system are implemented in the FPGA. FPGA design is achieved

with the Altera's System Integration Tool, Qsys, to obtain single, compact system block which maintains all FPGA logic blocks and Nios II embedded soft core processor on itself.

Combining embedded soft core processor and FPGA logic on a single chip provides a high performance embedded application and flexibility in both hardware and software design. Considering drawbacks and benefits of FPGA and embedded processor, work division is achieved. FPGA designs are generally more time consuming and have limitations in changing and debugging. However, complex operations can be achieved easily by FPGA and execution time decreases to a large extent when compared with the processor. On the other hand, embedded processor based designs are easy to debug and verify. These designs can be changed easily without limitations. However, as a drawback, complex operations may take larger execution times.

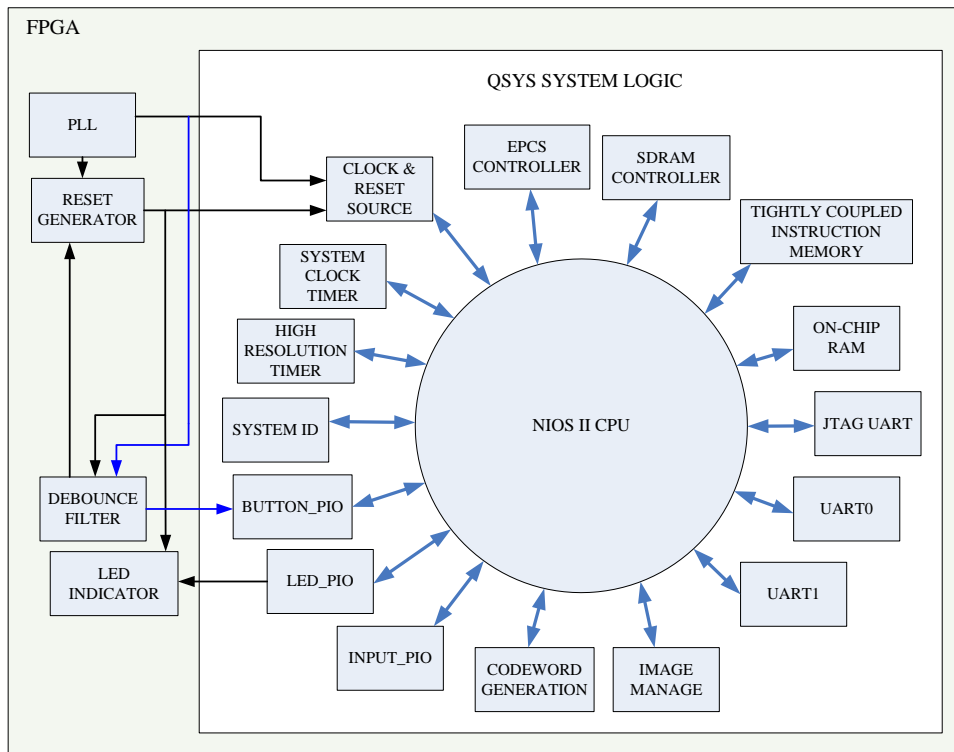
Each step for the character segmentation and character recognition parts of the LPDRS is achieved by either FPGA or Nios II embedded processor by considering the benefits and drawbacks of the FPGA based and processor based designs. In order to use the FPGA implementation, a GUI is designed in C#. GUI interface is used to load the image and to show recognition result with execution time of the recognition process. Detailed description of the designed GUI is given in Appendix C.

FPGA implementation of the license plate recognition can be divided into two parts. First, FPGA logical blocks are generated by using the Qsys system integration tool and then image processing algorithms are implemented with software application which uses the previously generated hardware blocks. Each step is explained in detailed in the following sections. For the FPGA side blocks, generation of Qsys system block is explained. For the software application of the embedded processor, the pseudo code of the algorithm is given.

#### **4.2.1 FPGA BLOCKS**

For the license plate recognition, blocks used for the recognition are generated in a single compact form with the Qsys system integration tool. There is also some control blocks outside the Qsys system block such as reset generator, PLL, debounce filter and led direction indicator etc. These blocks and their functions are explained in this section.

FPGA blocks of the recognition are given in Figure 4-3.

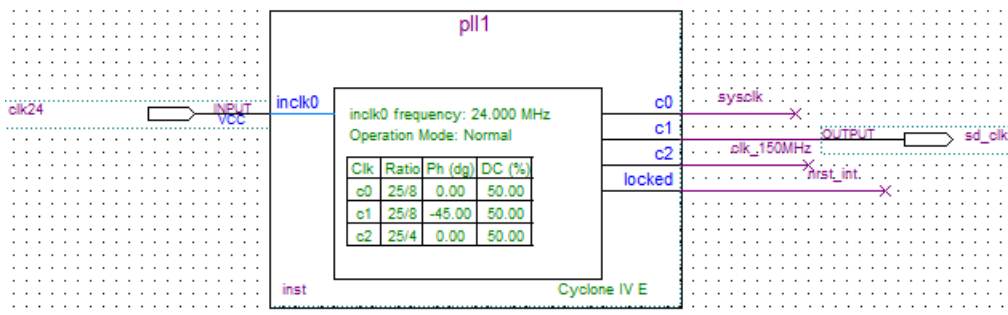


**Figure 4-3 FPGA Blocks for the Recognition**

#### 4.2.1.1 PLL

Altera DB\_START\_4CE10 Cyclone IV E development board includes a free running 24 MHz crystal Quartz oscillator with  $\pm 50$ ppm which drives the FPGA directly. This frequency is not used directly for SDRAM or embedded processor. Therefore, PLL is used to generate system clock, external clock for the SDRAM and debug clock for real time debugging.

To generate stable frequencies in the FPGA design, Altera MegaWizard interface is used to specify PLL circuitry. PLL interface is given in Figure 4-4.



**Figure 4-4 PLL Interface**

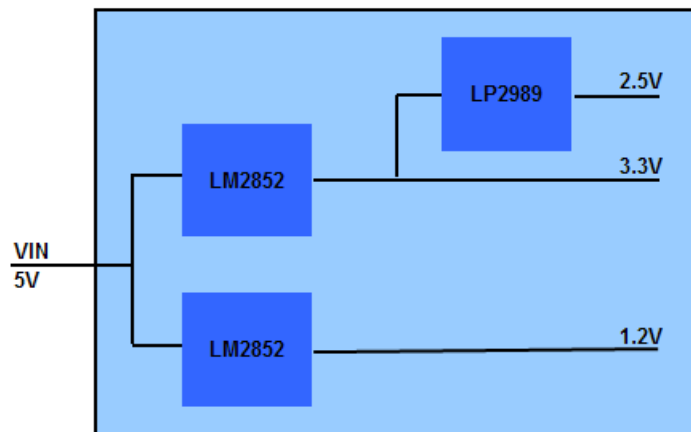


Three clocks are generated by PLL. ‘sysclk’ is the general system clock used in the Qsys and embedded processor which is 75 MHz. ‘sd\_clk’, 75 MHz clock, is the external clock for the SDRAM and ‘clk\_150MHz’, 150 MHz clock, is used for the debug which requires faster clock frequency to sample signals in the FPGA. ‘locked’ output acts as an indicator when the PLL has reached phase-locked. It stays high as long as the PLL is locked, and stays low when the PLL is out-of-lock.

#### 4.2.1.2 RESET GENERATOR

Reset generator block generates the reset of the FPGA block by observing the power good signal of the board, ‘locked’ signal of the PLL and the system reset button.

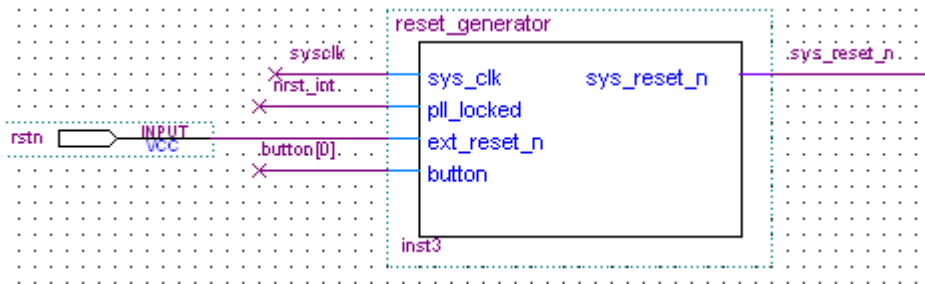
Main supply voltage of the Altera DB\_START\_4CE10 Cyclone IV E development board is 5V, which is supplied with the USB connection to the PC or with the 5V regulated center positive input power supply. Required powers are generated on the development board. Power distribution can be seen in Figure 4-5.



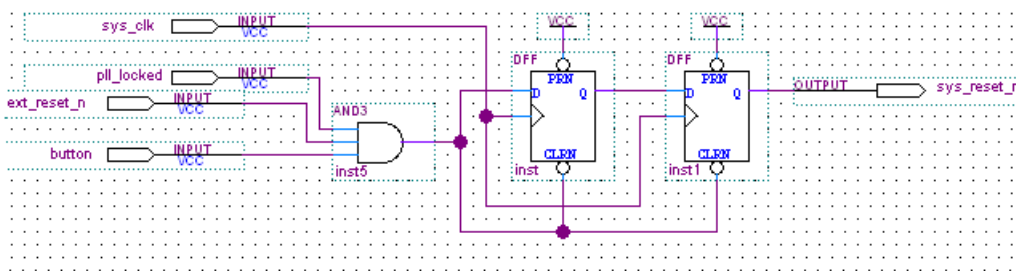
**Figure 4-5 Power Distribution of the Development Board**

1.2V is the core voltage, 2.5V is the analog supply for the FPGA and 3.3V is the I/O voltage. Power good signal is generated by monitoring the 1.2V and 3.3V and acts as a power up reset for the reset generator block.

System reset button enables the user; reset the system at any time. ‘locked’ signal indicates the stability of the system clock. Therefore, it is also related with the reset because system should stay at reset while the PLL is out-of-lock. Reset generator interface and block diagram for the reset generator are given in Figure 4-6 and Figure 4-7, respectively.



**Figure 4-6 Reset Generator Interface**

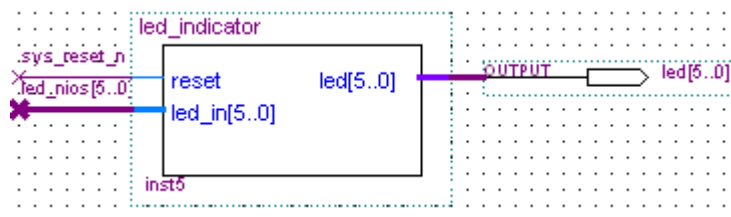


**Figure 4-7 Block Diagram for the Reset Generator**

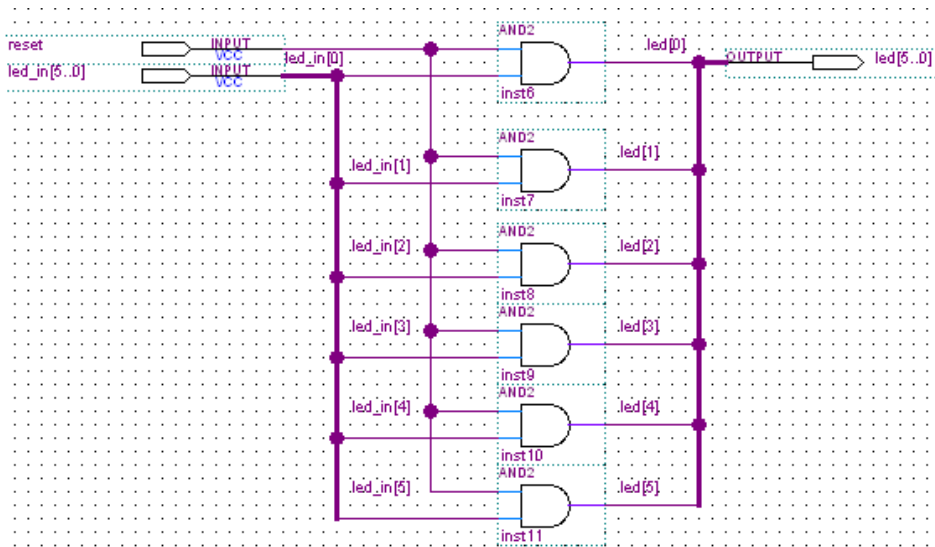
Since the input of the reset generator block is asynchronous, metastability may emerge. To tolerate metastability, two successive synchronizing flip-flops are added to the reset generator block. Output of the reset generator block is the system reset of the Qsys system logic.

#### 4.2.1.3 LED INDICATOR

LED indicator controls the LED interface. At reset time, all LEDs are lighted; otherwise it is controlled by Qsys system block. LED indicator interface and block diagram of the LED indicator are given Figure 4-8 and Figure 4-9, respectively.



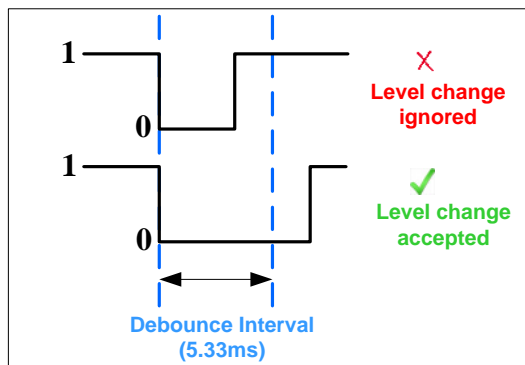
**Figure 4-8 Led Indicator Interface**



**Figure 4-9 Block Diagram of the Led Indicator**

#### 4.2.1.4 DEBOUNCE FILTER

Contact bounce at the user buttons may cause numerous events. To avoid it, debounce filter is implemented for the user buttons. New value is only accepted if it is stable for a predefined period of time. Debounce filter is generated with the HDL code. According to the code, a pulse with duration less than specified debounce interval is rejected. A pulse with duration of one or more debounce intervals is accepted and a respective event is generated. Acceptance of pulse on the user buttons for the HDL code of the debounce filter block is given in Figure 4-10 .



**Figure 4-10 Acceptance of Pulse on the User Buttons**

#### 4.2.1.5 QSYS SYSTEM LOGIC

Qsys is a system integration tool which is included as a part of the Quartus II software. Qsys is used to integrate customized HDL components, IP cores and user defined custom components. It automatically creates high performance interconnect logic by eliminating error-prone and time-consuming task of writing HDL to specify the system level connections.

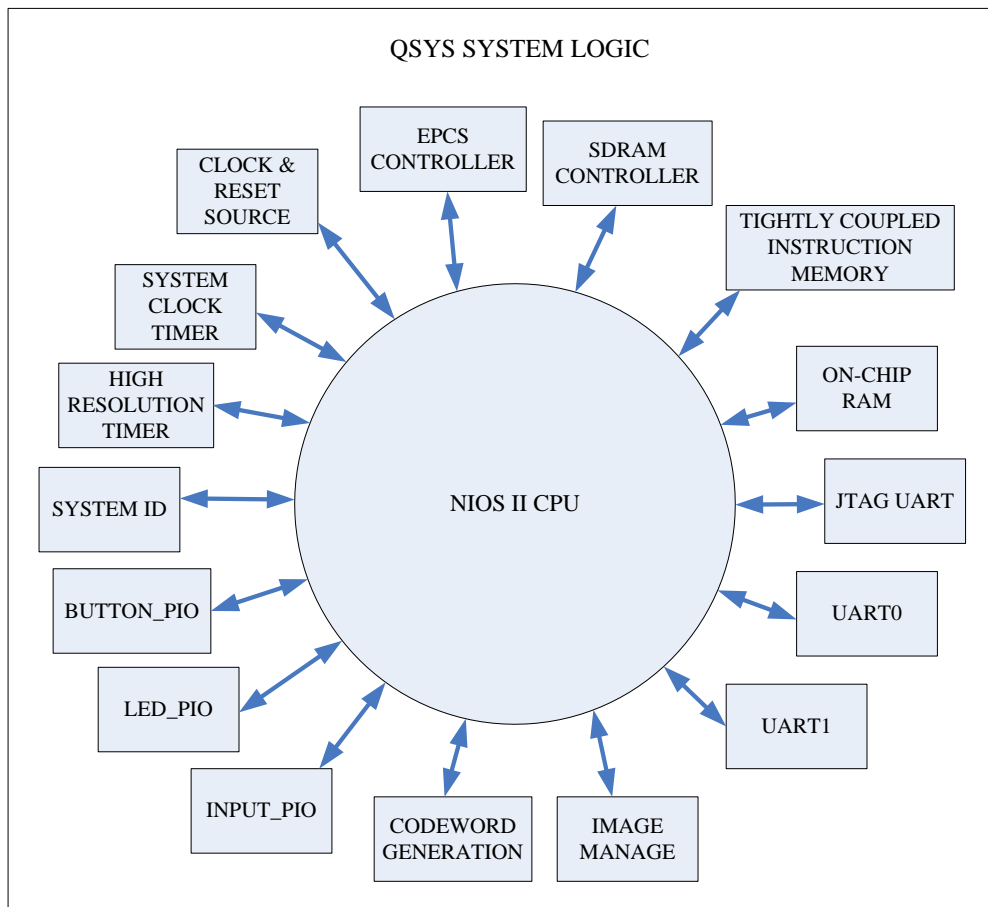
Qsys provides the following advantages for hardware system design [11]:

- Automates the process of customizing and integrating components
- Supports modular system design
- Supports visualization of large systems
- Supports optimization of interconnect fabric and pipelining within the system
- Fully integrated with the Quartus II software

Qsys supports hierarchical system design. With hierarchical system design, it offers the following advantages:

- Enables team-based, modular design by dividing large designs into subsystems
- Enables design reuse by allowing any Qsys system as a component
- Enables scalability by allowing instantiations of multiple instances of a Qsys system.

In this thesis, Qsys is used for integration and modular system design is obtained. Components, integrated into the Qsys system logic, use Avalon interface which simplifies system design. Block diagram for the integrated components is given in Figure 4-11. In the following sections each component is explained in detail.



**Figure 4-11 Integrated Components of the Qsys System Logic**

Components which have interfaces with the external peripherals are exported as ports in the Qsys system logic. Port interface can be seen in Figure 4-12.

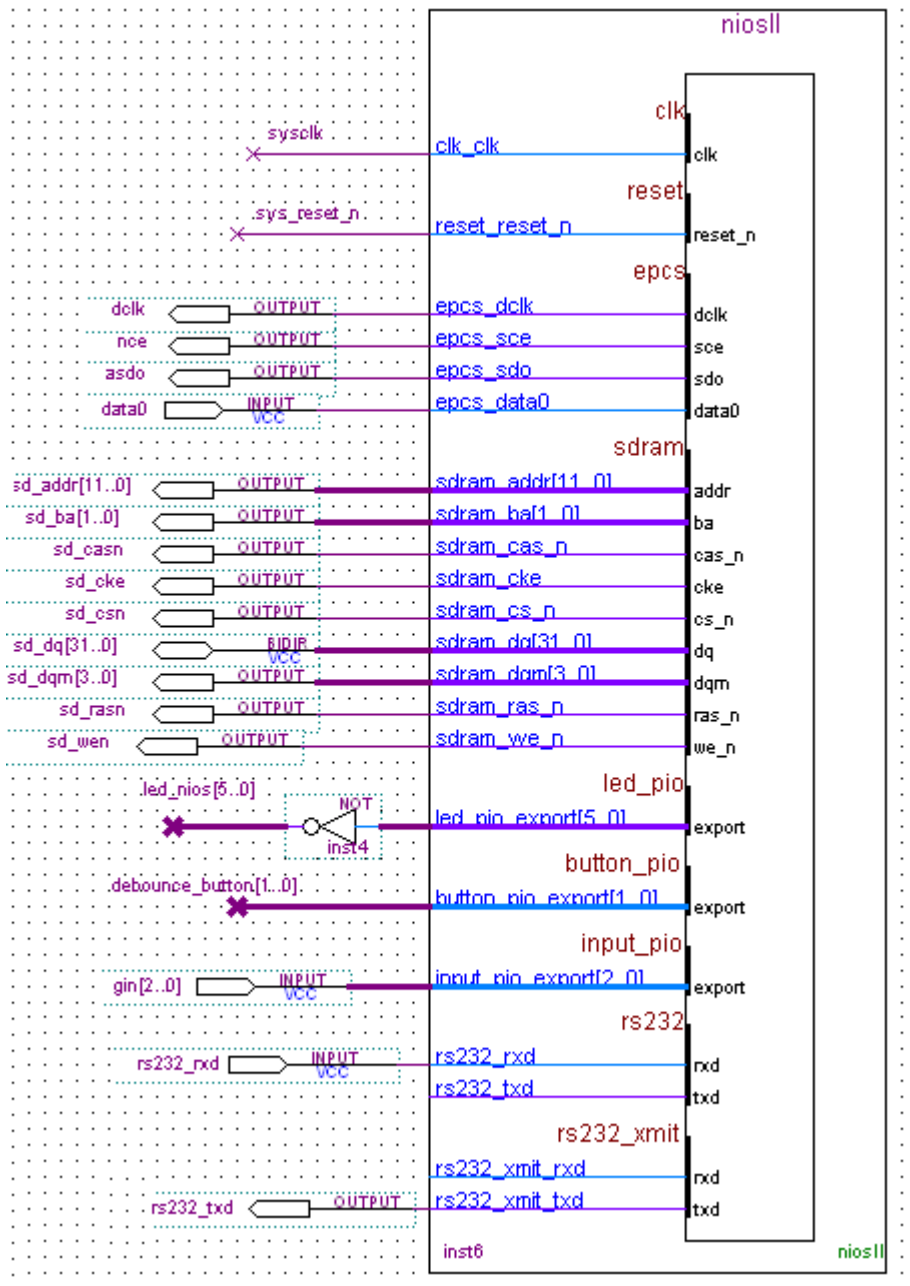


Figure 4-12 Qsys System Logic Interface

#### 4.2.1.5.1 CLOCK AND RESET SOURCE

Clock and reset source component which is a component in the Altera library, sets the frequency and drives clock and reset signals out of component. All the integrated components which use system clock and system reset are connected to the output of the clock and reset source block.

#### 4.2.1.5.2 NIOS II PROCESSOR

Nios II processor is a general purpose RISC processor core. Nios II processor system consists of a Nios II processor core, a set of on-chip peripherals, on-chip memory, and interfaces to off-chip memory, all implemented on a single Altera device. Nios II processor system uses a consistent instruction set and programming model. Nios II software is developed on the Nios II Embedded Design Suite (EDS) which contains all the software development tools. Detailed information about Nios II processor is given at the Appendix-A.

In this thesis, standard type Nios II core is used. Interface of the processor is 32 bit. It provides instruction cache, branch prediction, hardware multiplication and hardware division. EPCS16 flash device is set as the reset vector memory to load the software application from flash at power up or at reconfiguration time and SDRAM is set as the exception vector memory. It has 4 Kbyte instruction cache. Burst transfers are disabled in the core. It also provides debug level. With JTAG target connection, software download, use of software breakpoints, four hardware breakpoints, four data triggers, instruction trace, on-chip trace, data trace and off-chip trace become available. Interface of the Nios II processor is given in Figure 4-13.

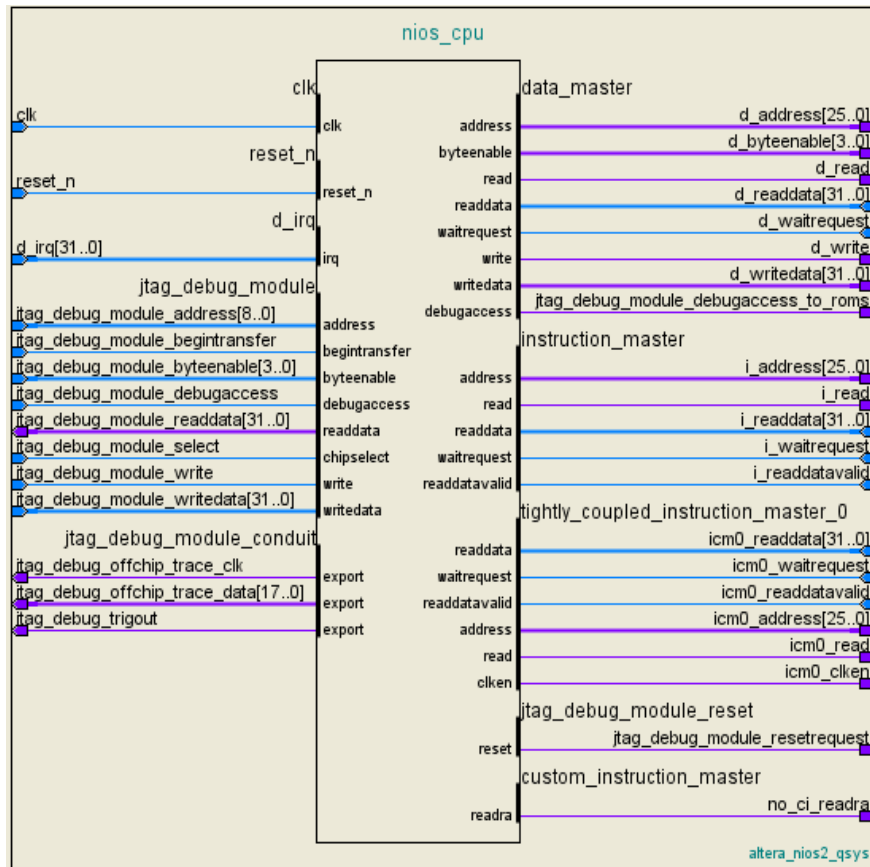


Figure 4-13 Nios II Processor Interface

#### **4.2.1.5.3 INTERVAL TIMER**

There are two timers in the Qsys system logic, which are system clock timer and high resolution timer. System clock timer has 1 ms timeout period. If time timeout exceeds IRQ is generated and sent to the processor. This timer is used by the software application. High resolution timer has 1 us timeout period. If time timeout exceeds, IRQ is generated. It is used for algorithms that require sensitivity in time. High resolution timer is used to measure the execution times of the software application codes. These timers have no peripheral interface. Therefore, there is no pin exported outside the Qsys system logic block.

#### **4.2.1.5.4 SYSTEM ID**

Nios II processor system use the System ID core to verify that an executable program was compiled targeting the actual hardware image configured in the FPGA. If the expected ID in the executable does not match the system ID core in the FPGA, it is possible that the software will not execute correctly.

#### **4.2.1.5.5 PIO**

The parallel input/output (PIO) core with Avalon interface provides a memory-mapped interface between Avalon memory mapped slave port and general purpose I/O ports. The I/O ports connect either to on-chip user logic, or to I/O pins that connect to devices external to the FPGA. Nios II processor controls the PIO ports by reading and writing the register-mapped Avalon memory mapped interface. Under the control of the processor, PIO core captures data on its inputs and drives data to its outputs.

There are three different Parallel I/O components in the Qsys system logic. These are Button PIO, LED PIO and Input PIO.

Button PIO is integrated to observe whether the user buttons are pressed or not. Since there are two user buttons, the width of the Button PIO is two and it is configured as input. If any of the buttons is pressed, IRQ is generated to the processor.

LED PIO is integrated to control the LED interface. Since there are six leds on the development board, the width of the PIO is six and it is configured as output. By accessing registers of the LED PIO, LEDs are lighted or turned out.

Input PIO is integrated in order to use the general input pins provided in the I/O connector of the development board. Width of the PIO is three and it is configured as input.

#### **4.2.1.5.6 EPCS CONTROLLER**

The EPCS serial flash controller core with Avalon interface allows Nios II system to access EPCS16 serial configuration device. Altera provides drivers that is integrated into the Nios II hardware abstraction layer (HAL) system library, allowing read and write



access to the EPCS16 device using the HAL application program interface (API) for the flash devices. Using EPCS serial flash controller core, program code can be stored in the EPCS device. It provides a boot-loader feature that allows Nios II system to store the main program code in the EPCS device. Also, non-volatile program data and device configuration data can be stored in the flash device.

#### **4.2.1.5.7 SDRAM CONTROLLER**

SDRAM controller core with the Avalon interface provides an Avalon memory mapped interface to external SDRAM device. With SDRAM controller, FPGA is connected easily to the SDRAM chips. The interface of the external SDRAM chip presents the signals defined by the PC100 standard. These signals are connected externally to the SDRAM chip through the I/O pins on the FPGA device. The timing and sequencing of signals depends on the configuration of the core. The core preset is configured as single Micron MT48LC4M32B2-7 chip because settings of this chip are similar to ISSI-chip used on the Altera DB\_START\_4CE10-Cyclone IV E Development Board.

#### **4.2.1.5.8 TIGHTLY COUPLED INSTRUCTION MEMORY**

Tightly coupled memory is integrated into the Qsys system logic in order to increase the system performance if required. System performance is increased by placing some code into the tightly-coupled memory. Altera FPGAs include on-chip memory blocks that can be used as RAM or ROM in Qsys. On-chip memory has fast access time, compared to off-chip memory. Tightly coupled instruction memory is a 16 Kbyte on-chip RAM with 32-bit data width.

#### **4.2.1.5.9 ON-CHIP RAM**

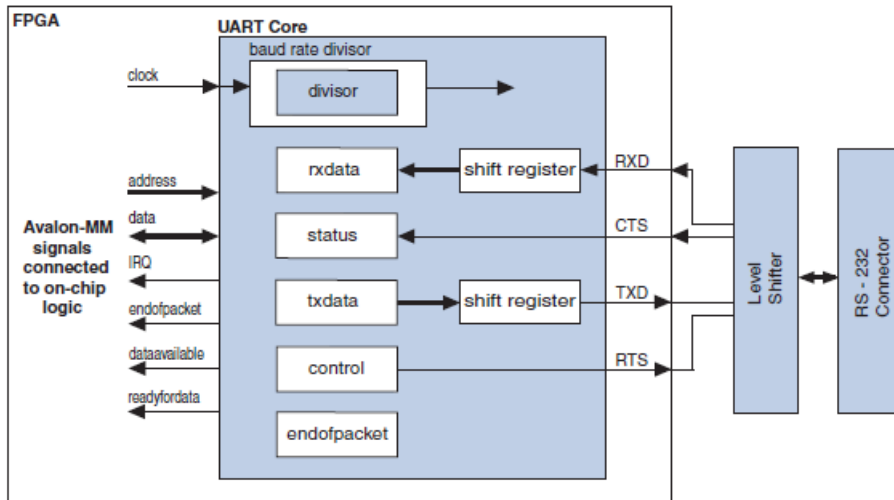
Software developed for the processor may require on-chip RAM to keep arrays or variables used in the image processing algorithm. Since on-chip memory has fast access time and it has Avalon interface with the processor, 8 Kbyte RAM with 32-bit data width is integrated into the Qsys system block. It is used by the image processing algorithms developed for the Nios II processor.

#### **4.2.1.5.10 JTAG UART**

JTAG UART is integrated into the system to easily debug the software without need of the RS-232 serial connection. JTAG UART core provides an Avalon interface that hides the complexities of the JTAG interface from embedded software programmers. Nios II processor communicates with the core by reading and writing control and data registers. The JTAG UART core uses the JTAG circuitry built into Altera FPGAs, and provides access via the JTAG pins on the FPGA. The host PC can connect to the FPGA via Altera JTAG download cable. For the Nios II processor, device drivers are provided in the hardware abstraction layer (HAL) system library, allowing software to access the core using the ANSI C Standard Library *stdio.h* routines.

#### 4.2.1.5.11 UART INTERFACE

The UART core is used in order to communicate serial character streams between FPGA and external device. The core implements the RS-232 protocol timing and provides Avalon memory mapped slave interface that allows Nios II processor to communicate with the core simply by reading and writing control and data registers. Block diagram of the UART core is given in Figure 4-14.

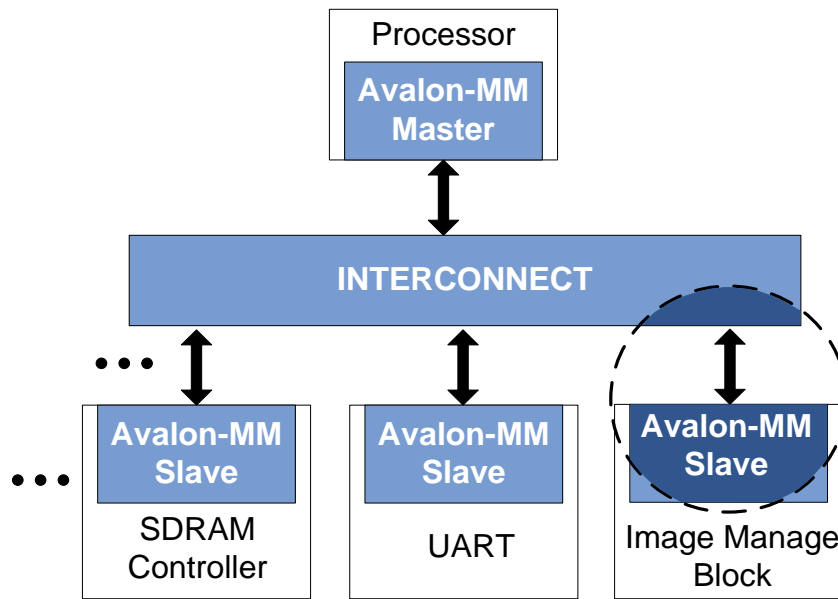


**Figure 4-14 Block Diagram of the UART Core**

There are two UART cores in the system; one for to receive the license plate image and one for to send the result of the executed algorithm to the GUI. There is only one RS-232 connector; therefore, receive data of one UART and transmit data of the other UART are connected to the RS-232 connector. With this approach, there is no intervention while receiving or sending data. Baudrates of the UART cores are 115200 bps. Communication is achieved with eight data bits, one stop bit and no parity bit. There is no flow control in the communication line.

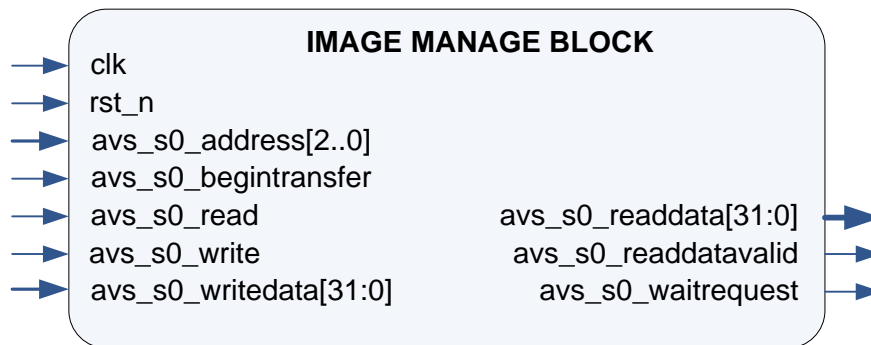
#### 4.2.1.5.12 IMAGE MANAGE BLOCK

Image Manage Block is developed in order to be used in the software application developed for the Nios II processor. Image Manage Block has Avalon memory mapped interface (Avalon MM) to provide easy integration into the Qsys system logic. HDL code of the Avalon interface is developed according to Avalon Interface Specification [8]. Avalon MM is an address based read/write interface which has typical master-slave connections. Figure 4-15 shows the connection of the Image Manage Block, highlighting the Avalon MM slave interface connection to the interconnect fabric.



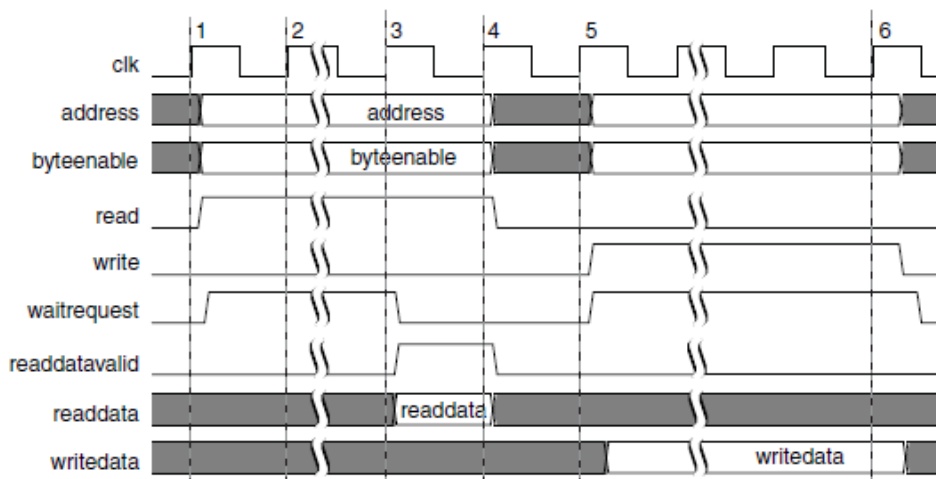
**Figure 4-15 Avalon MM Interface Connection of the Image Manage Block**

Avalon MM interface of the Image Manage Block can be seen in Figure 4-16.



**Figure 4-16 Image Manage Block Interface**

Avalon memory mapped interface is used for read and write transfers of processor and Image Manage Block in the memory-mapped system. It is a synchronous interface. Detailed explanation of the Avalon MM fundamental signals, shown in Figure 4-16, can be obtained from Avalon Interface Specification [8]. Typical Avalon MM read and write transfers are shown in Figure 4-17.

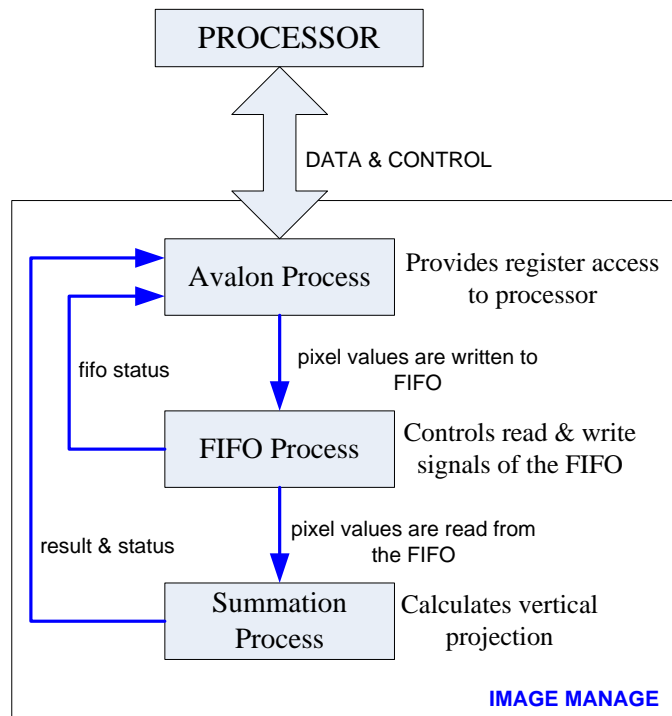


**Figure 4-17 Read and Write Transfers of the Avalon MM Interface**

In the Avalon MM interface, *avs\_s0\_begintransfer* signal is asserted for the first cycle of each transfer which indicates the start of the transfer. Image Manage Block can stall the interconnect fabric for as many cycles as required by asserting the *avs\_s0\_waitrequest* signal. When the Image Manage Block asserts *avs\_s0\_waitrequest*, the transfer is delayed and the address and control signals are held constant. Transfer is completed on the rising edge of the first *clk* after the Image Manage Block deasserts *avs\_s0\_waitrequest* signal. Detailed explanations about the Avalon MM read and write transfers can be obtained from Avalon Interface Specification [8].

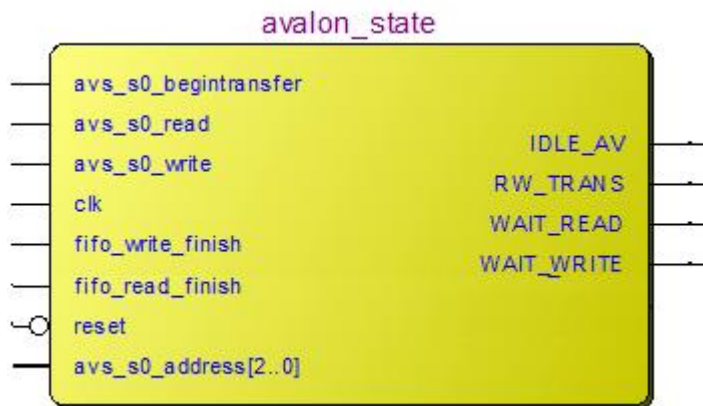
The main function of the Image Manage Block is to calculate vertical projection of the license plate. For each column of the image, all pixel values belong to that column are summed and put in a register to provide access to processor. In order to calculate vertical projection, HDL code of the Image Manage Block is developed. HDL code of the Image Manage Block contains three processes which can be seen in Figure 4-18. Avalon process is used to provide access to the Avalon registers, FIFO process controls read and write operations of the show ahead FIFO and summation process achieves the summation. All processes work with states. State transitions are achieved with enable signals.

Block diagram of the Image Manage Block is given in Figure 4-18.

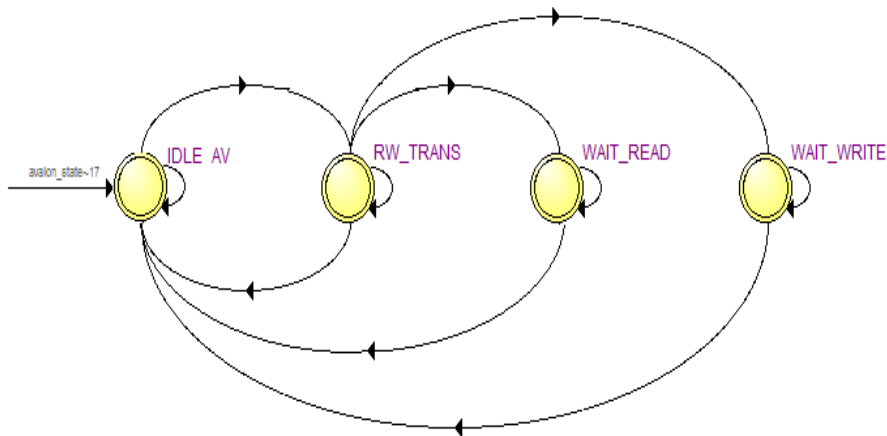


**Figure 4-18 Block Diagram of the Image Manage Block**

Input signals and states of the Avalon process can be seen in Figure 4-19 and Figure 4-20, respectively.



**Figure 4-19 Avalon Process Interface**



**Figure 4-20 Avalon Interface State Transitions**

When there is no read or write transfer, Avalon process waits in the *IDLE\_AV* state. If read or write transfer is started with the *avs\_s0\_begintransfer* signal, it goes to *RW\_TRANS* state. In the *RW\_TRANS* state, read and write transfers of internal Image Manage Block register are achieved by observing *avs\_s0\_read*, *avs\_s0\_write* and *avs\_s0\_address[2..0]* signals. If the processor wants to write data to FIFO, *RW\_TRANS* state goes to *WAIT\_WRITE* state and *avs\_s0\_waitrequest* is asserted to wait the completion of the FIFO write. *avs\_s0\_waitrequest* signal is asserted because Avalon interface is 32-bit while FIFO interface is 8-bit. To write 32-bit data, Avalon process must wait until the completion of the write operation. To declare the completion of the write operation, *fifo\_write\_finish* signal is sent from FIFO Process to Avalon process and *WAIT\_WRITE* state goes back to *IDLE\_AV* state. If the processor wants to read data from FIFO, *RW\_TRANS* state goes to *WAIT\_READ* state and *avs\_s0\_waitrequest* signal is asserted to wait the completion of the FIFO read. *avs\_s0\_waitrequest* signal is asserted because Avalon interface is 32-bit while FIFO interface is 8-bit. To read 32-bit data, Avalon process must wait for the completion of the 32-bit data read. To declare the completion of the read operation, *fifo\_read\_finish* signal is sent from FIFO Process to Avalon process and *WAIT\_READ* state goes back to *IDLE\_AV* state. Register map for the Image Manage Block can be seen in Table 4-1.

**Table 4-1 Register Map for the Image Manage Block**

BASE = IMAGE_MANAGE_BASE		
avs_s0_address[2..0]	Type*	Register
0x0	R/W	ADDR_FIFO: Data written to this register is written to FIFO and data read from this register is read from the FIFO
0x1	RO	FIFO_STATUS : Register which indicates full, empty and used status of the FIFO

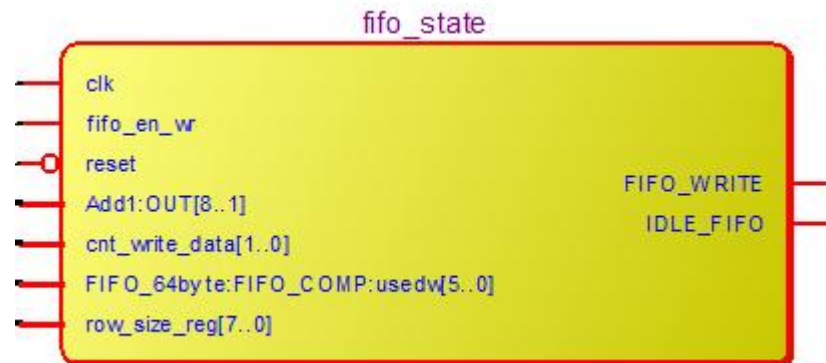
**Table 4-1 Register Map for the Image Manage Block (Continued)**

BASE = IMAGE_MANAGE_BASE		
avs_s0_addr[2..0]	Type*	Register
0x2	R/W	ROW_SIZE: Register for the row size of the image
0x3	R/W	COLUMN_SIZE: Register for the column size of the image
0x4	R/W	START_ADDR: Processor sends 32 bit data to write the data FIFO. But for some images the number of rows is not divided by 4; therefore, some of the bits in the data bus cannot be used for the sum operation. With start address register block knows with which pixel sum operation starts and it gives the accurate result.
0x5	RO	SUM_ADDR :Register gives the result of the vertical projection for single column
0x6	RO	STATUS_SUM : Register gives the status of the sum operation whether sum operation is completed or not. If the sum operation is completed, SUM_ADDR register can be read by the processor.
0x7	R/W	ADDR CHECK : Register to check whether data is able to be written to or read from the block correctly.

\*R/W : Read and Write operations are achieved.

\*RO : Read only: Only read operation is achieved.

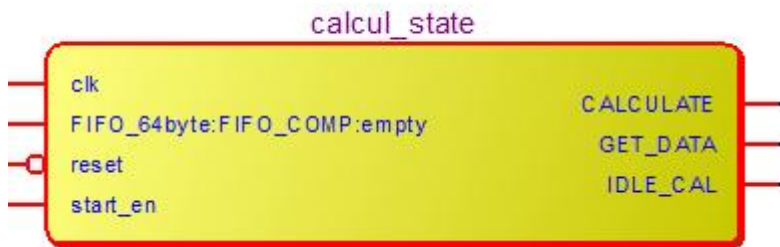
Input signals and states of the FIFO process can be seen in the Figure 4-21.



**Figure 4-21 FIFO Process Interface**

FIFO process is generally used to write the image data to the FIFO. Read operation from FIFO is not achieved from the processor. Therefore, only write operation is explained. If there is no write request, FIFO process waits in the *IDLE\_FIFO* state. When the processor wants to write data to the FIFO, *fifo\_en\_wr* signal asserted in the Avalon process and *IDLE\_FIFO* state goes to *FIFO\_WRITE* state in the FIFO process. Before writing data to the FIFO, processor sets the *ROW\_SIZE* and *START\_ADDR* registers of the Avalon process given in Table 4-1. *ROW\_SIZE* register indicates how many pixels exist for single column. Processor sends 32 bit data to FIFO in each transfer. Each pixel in the image is represented with 1 byte. If the row size of the image is not divided into 4, some bytes in the 32-bit data bus of the processor should not be included into the vertical projection calculation. Therefore, to know start address of the valid byte is important and start address is written to the *START\_ADDR* register of the Avalon process. After the number of data which equals to *ROW\_SIZE* register is written to FIFO, summation process of the Image Manage Block is enabled with *start\_en* signal and *fifo\_write\_finish* signal is asserted. Then, *FIFO\_WRITE* state goes back to *IDLE\_FIFO* state. With the assertion of the *fifo\_write\_finish* signal, *WAIT\_WRITE* state of the Avalon process also goes back to *IDLE\_AV* state.

Input signals and states of the summation process can be seen in Figure 4-22.



**Figure 4-22 Summation Process Interface**

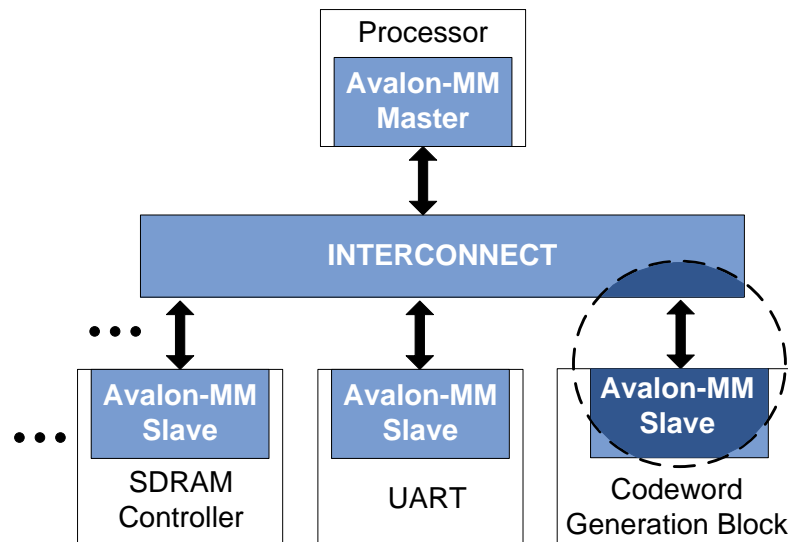
If summation process is not enabled in the FIFO process, process waits in the *IDLE\_CAL* state until activation signal, *start\_en*, is asserted by the FIFO process. After *start\_en* is asserted, *IDLE\_CAL* state goes to *GET\_DATA* state and process reads the pixel values from FIFO one by one. After each data is read from the FIFO, process goes to *CALCULATE* state and sums the read data by going back and forth the *GET\_DATA* and *CALCULATE* states. After all data is read from the FIFO and all pixel values are summed, summation process goes to *IDLE\_CAL* state. Empty signal is asserted by the FIFO when there is no data in the FIFO. After the summation is completed, *STATUS\_SUM* register of the Avalon process is updated which indicates the calculation of the vertical projection for a single column is completed and then the result becomes ready in the *SUM\_ADDR* register for the processor access. For each column in the images, the processes are launched by the processor.



Image Manage Block functions can be summarized as: Row size of the license plate is written to the row size register of the block. Row size value indicates how many pixel values for each column will be summed. Then, pixel values are sent to the block. Image Manage Block contains 64 byte show ahead FIFO. Pixels values received from the processor are written to the FIFO. After all pixels of that column are received, a flag emerges which enables the summation process. With enabling the summation process, all pixel values are read from the FIFO, the sum of pixel values is calculated and written to the result register to provide access to processor. After completion of summation, status of the vertical projection operation is updated in the status register which indicates whether the sum calculation is completed or not. By observing the status register, processor can obtain the vertical projection result for each column one by one.

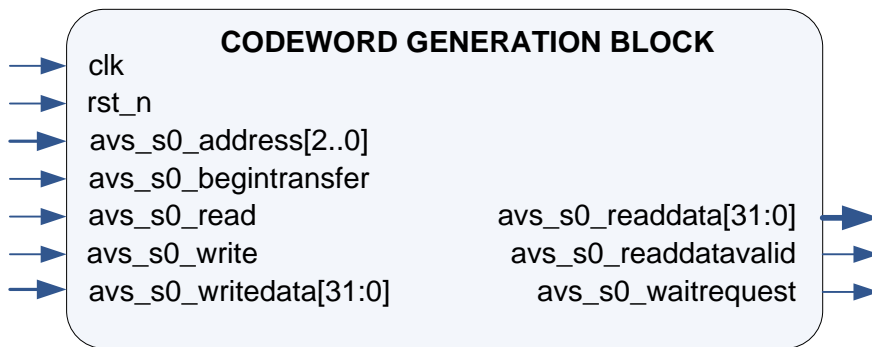
#### 4.2.1.5.13 CODEWORD GENERATION BLOCK

Codeword Generation Block is used to generate codeword for each character after the character segmentation part. Generated codeword is used for the recognition part. To calculate the codeword for each character, HDL code is developed and integrated into the Qsys system logic. Codeword Generation Block has an Avalon memory mapped slave interface. Connection of the Codeword Generation Block with the processor can be seen in Figure 4-23.



**Figure 4-23 Avalon MM Interface Connection of the Codeword Generation Block**

Avalon MM interface of the Codeword Generation Block can be seen in Figure 4-24.



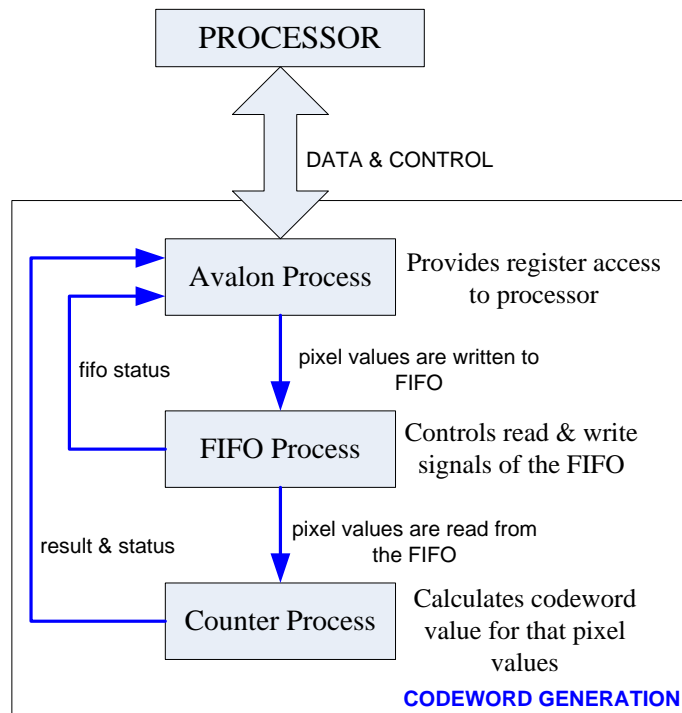
**Figure 4-24 Avalon MM Interface of the Codeword Generation Block**

Read and write transfers of the Avalon MM interface is same with the Image Manage Block as explained in the Section 4.2.1.5.12.

The main function of the Codeword Generation Block is to calculate both x-axis and y-axis codewords. Column wise or row wise calculation is achieved according to the block settings and data sent from the processor. For each row (or column), block counts how many '1' to '0' transition occurs. The value of the counter gives the codeword of that row (or column). After codeword of each row (or column) is generated, it is made unique by the processor.

For the generation of codeword, HDL code of the Codeword Generation Block is developed. HDL code of the Codeword Generation Block consists of three processes which can be seen in Figure 4-25. Avalon process is used to provide access to the Avalon registers, FIFO process controls read and write operations of the show ahead FIFO and counter process counts the '1' to '0' transitions. All processes work with states. State transitions are achieved with enable signals.

Block diagram of the Codeword Generation Block is given in Figure 4-25.



**Figure 4-25 Block Diagram of the Codeword Generation Block**

Avalon and FIFO processes have the same interface and states with the Image Manage Block. Operations of the processes are same with the Avalon and FIFO processes of the Image Manage Block as explained in Section 4.2.1.5.12. However, in Codeword Generation Block, internal register are different from the registers of the Image Manage Block. Register map for the Codeword Generation Block is shown in the Table 4-2.

**Table 4-2 Register Map for the Codeword Generation Block**

BASE = CODEWORD_GENERATION_BLOCK		
avs_s0_address[2..0]	Type	Register
0x0	R/W	ADDR_FIFO : Data written to this register is written to FIFO and data read from this register is read from the FIFO
0x1	RO	FIFO_STATUS : Register which indicates full, empty and used status of the FIFO
0x2	RO	CODEWORD : Register gives generated codeword for each column or row either for x-axis or y-axis according to the configuration

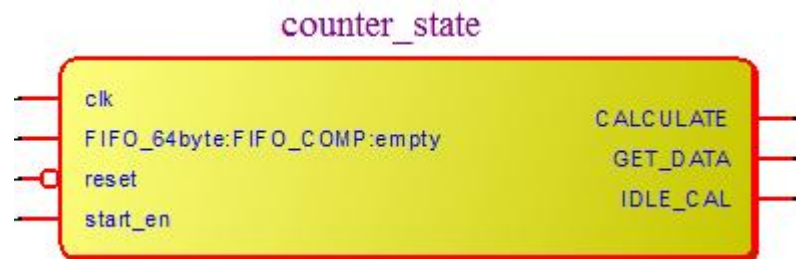
**Table 4-2 Register Map for the Codeword Generation Block (Continued)**

BASE = CODEWORD_GENERATION_BLOCK		
avs_s0_address[2..0]	Type	Register
0x3	RO	CODEWORD_STATUS : Register gives the status of the codeword generation operation whether codeword generation is completed or not. If the codeword generation is completed, CODEWORD register can be read by the processor.
0x4	O/Y	START_ADDR : Processor sends 32 bit data to write the data FIFO. But for some images the number of rows is not divided by 4; therefore, some of the bits in the data bus cannot be used for the codeword generation. With start address register, block knows with which pixel codeword generation starts and it gives the accurate result.
0x5	O/Y	PIXEL_SIZE : Register for the column or row size of the single character
0x6	NA	RESERVED: Reserved register is not used
0x7	O/Y	ADDR_CHECK : Register to check whether data is able to be written to or read from the block correctly.

FIFO process is generally used to write the image data to the FIFO. Read operation from FIFO is not achieved from the processor. Therefore, only write operation is explained. If there is no write request, FIFO process waits in the *IDLE\_FIFO* state. When the processor wants to write data to the FIFO, *fifo\_en\_wr* signal asserted in the Avalon process and *IDLE\_FIFO* state goes to *FIFO\_WRITE* state in the FIFO process. Before writing data to the FIFO, processor sets the *PIXEL\_SIZE* and *START\_ADDR* registers of the Avalon process given in Table 4-2. If Codeword Generation Block is used for the x-axis codeword generation, row size of the segmented character is written to the *PIXEL\_SIZE* register. If Codeword Generation Block is used for the y-axis codeword generation, column size of the segmented character is written to the *PIXEL\_SIZE* register. Processor sends the data 32 bit. Each pixel in the image is represented with 1 byte. If the value written to *PIXEL\_SIZE* register is not divided into 4, some bytes in the 32-bit data bus of the processor should not be included into the codeword generation. Therefore, to know start address of the valid byte is important and start address is written to the *START\_ADDR* register of the Avalon process. After the number of data which equals to *PIXEL\_SIZE* register is written to FIFO, counter process of the Codeword Generation Block is enabled with *start\_en* signal, and

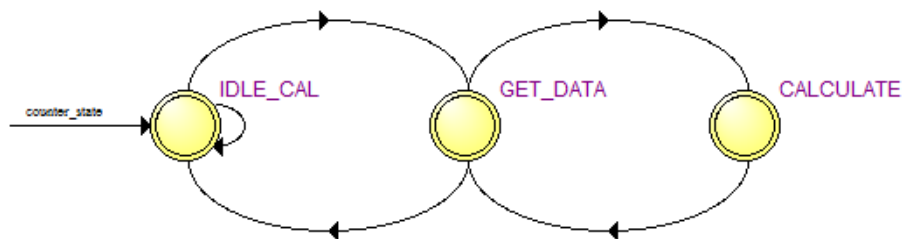
then *fifo\_write\_finish* signal is asserted. With the completion of the FIFO write operation, *FIFO\_WRITE* state goes back to *IDLE\_FIFO* state. With the assertion of the *fifo\_write\_finish* signal, *WAIT\_WRITE* state of the Avalon process also goes back to *IDLE\_AV* state.

Counter process of the Codeword Generation block has the same interface with the summation process of the Image Manage Block but the operation of the counter process is completely different. Input signals and states of the counter process can be seen in Figure 4-26.



**Figure 4-26 Counter Process Interface**

State machine for the counter process can be seen in Figure 4-27 .



**Figure 4-27 State Machine of the Counter Process**

Counter process counts the ‘1’ to ‘0’ transitions and generates codeword for each row or column. This process is enabled by the FIFO process. After all row (column) data is written to the FIFO, FIFO process enables the counter process for the generation of the codeword. FIFO process behaves in the same manner as explained in Section 4.2.1.5.12. If counter process is not enabled in the FIFO process, process waits in the *IDLE\_CAL* state until activation signal, *start\_en*, is asserted by the FIFO process. After *start\_en* signal is asserted, *IDLE\_CAL* state goes to *GET\_DATA* state and process reads the pixel values from FIFO one by one. After each data is read from the FIFO, process goes to *CALCULATE* state and counts the ‘1’ to ‘0’ transitions by comparing the previous data read from the FIFO with the newly arrived read FIFO data. By going back and forth between the *GET\_DATA* and *CALCULATE* states, all pixel values are examined. After all data is read from the FIFO and all pixel values are examined, counter process goes to *IDLE\_CAL* state. Empty signal is asserted by the FIFO when the FIFO is empty. After the

codeword generation is completed, CODEWORD\_STATUS register of the Avalon process is updated which indicates the codeword generation for a single row or column is completed and the result becomes ready in the CODEWORD register for the processor access. For each column and row in the segmented character, these processes are launched by the processor. Codeword Generation Block does not know whether it generates x-axis codeword or y-axis codeword. This information is known by the processor.

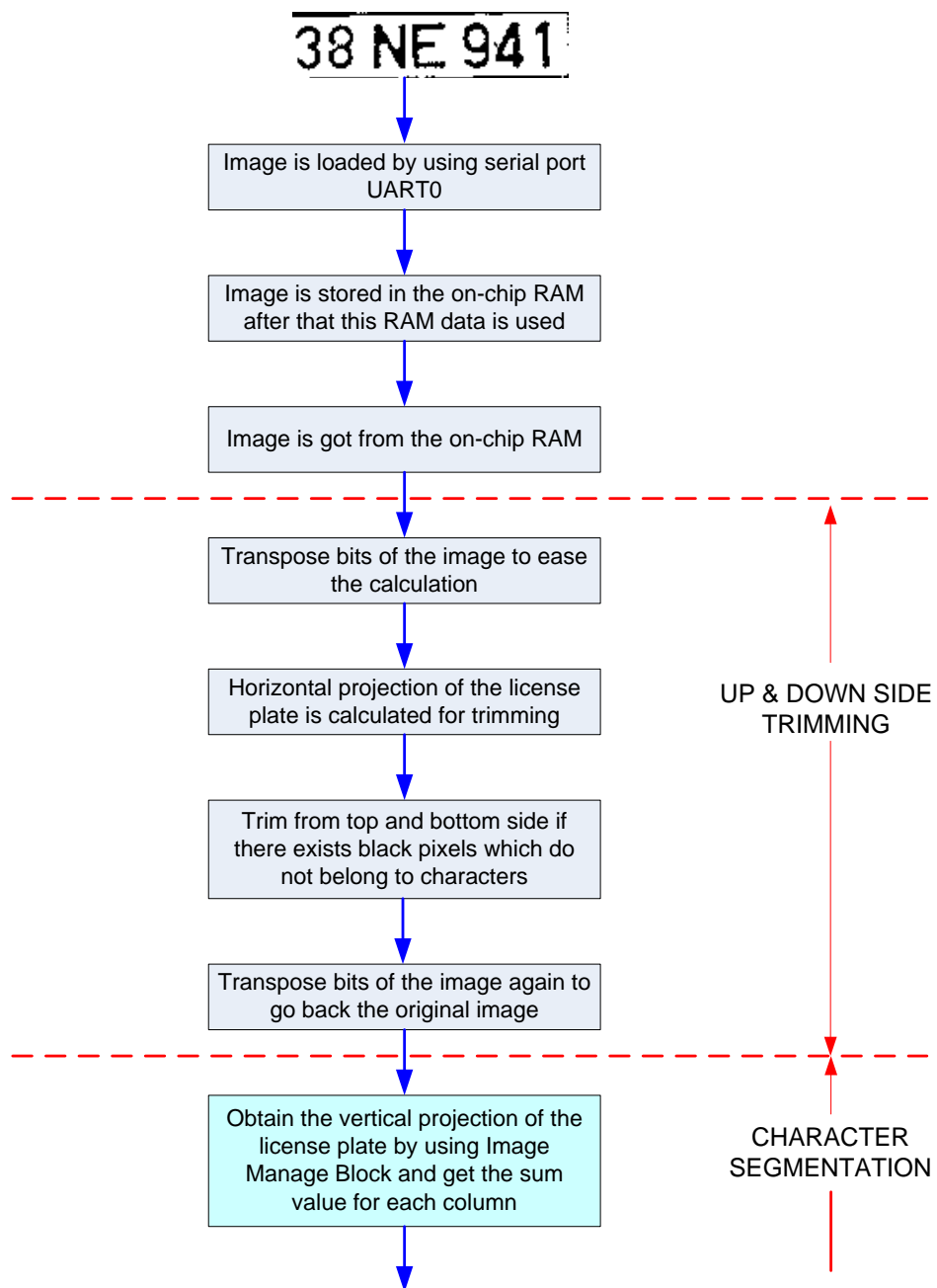
Operation of the Codeword Generation Block can be summarized as: Row (or column) size of the license plate is written to the size register of the block. Row (or column) size value indicates the number of pixels for which '1' to '0' transitions will be counted. Then, pixel values are sent to the block. Codeword Generation Block contains 64 byte show ahead FIFO. Pixels values received from the processor are written to FIFO. After all pixels of that row (or column) are received, a flag emerges which enables the counter process. With enabling the counter process, pixel values are read from the FIFO one by one and the number of '1' to '0' transitions for that row (or column) is calculated and written to the result register to provide access to processor. After all pixel values are counted, status condition is updated in the status register which indicates whether the codeword generation is completed or not. By observing the status register, processor can obtain the codeword value for that row (or column).

#### **4.2.2 SOFTWARE APPLICATION FOR EMBEDDED PROCESSOR**

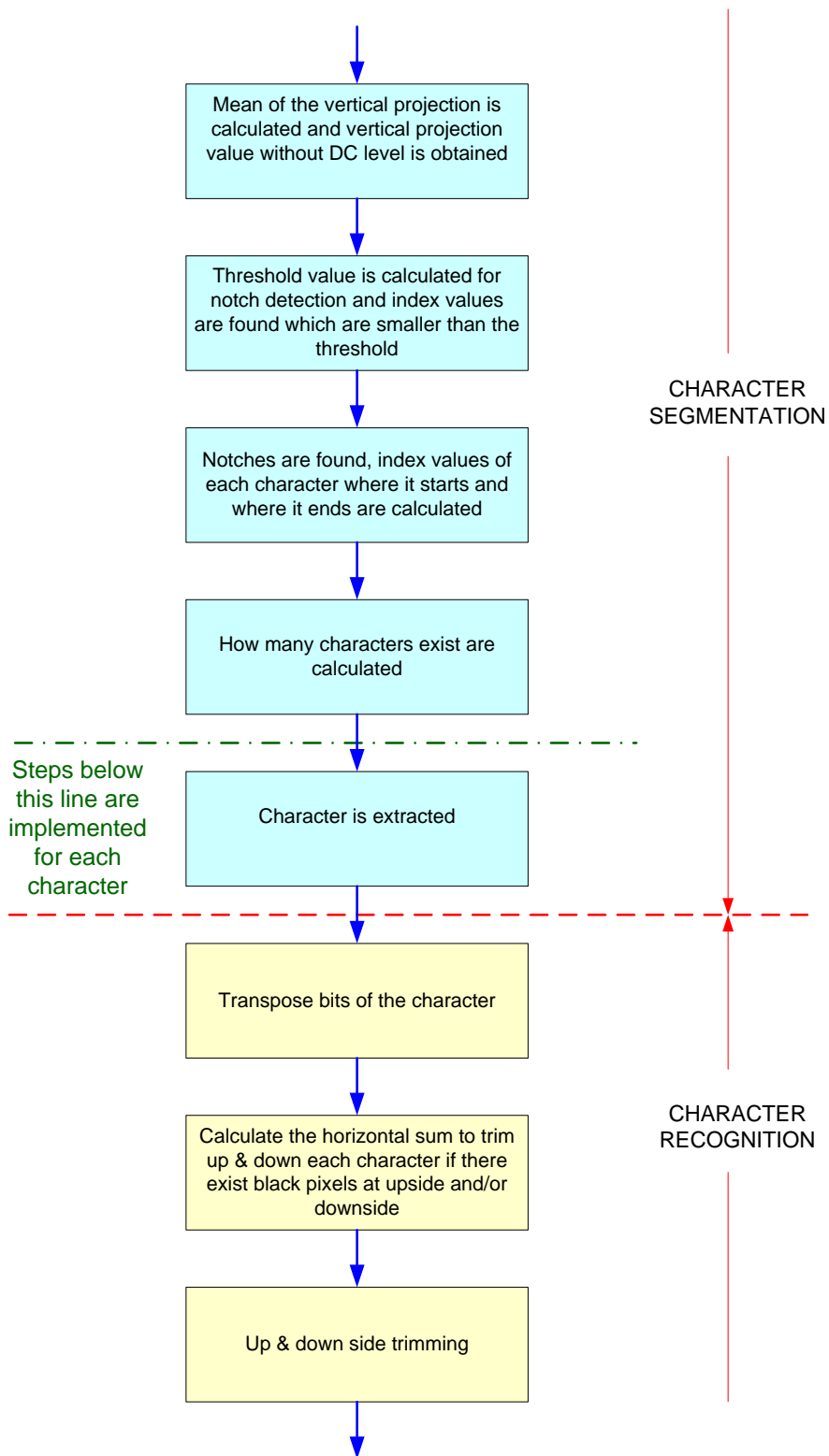
After FPGA blocks are designed, software is developed for the Nios II processor. Software application implements image processing algorithms by using the FPGA blocks and controls the flow of the algorithm.

At power up, the board receives a power-on reset signal, which resets the FPGA. After reset, FPGA is configured successfully from the EPCS16 configuration flash. Then, Nios II processor and other peripherals receive a hardware reset and enter the component's reset state. Next, Nios II processor jumps to its preconfigured reset address which is specified in the Qsys system logic and processor begins running instructions found at this address. The reset vector contains a packaged boot loader. The program is stored in flash memory but runs from SDRAM. Therefore, boot loader copies the application image from flash to SDRAM for program execution. After the boot loader executes, the processor jumps to the beginning of the program's initialization block and program waits the image from the serial port.

The block diagram for the software application is given in Figure 4-28.

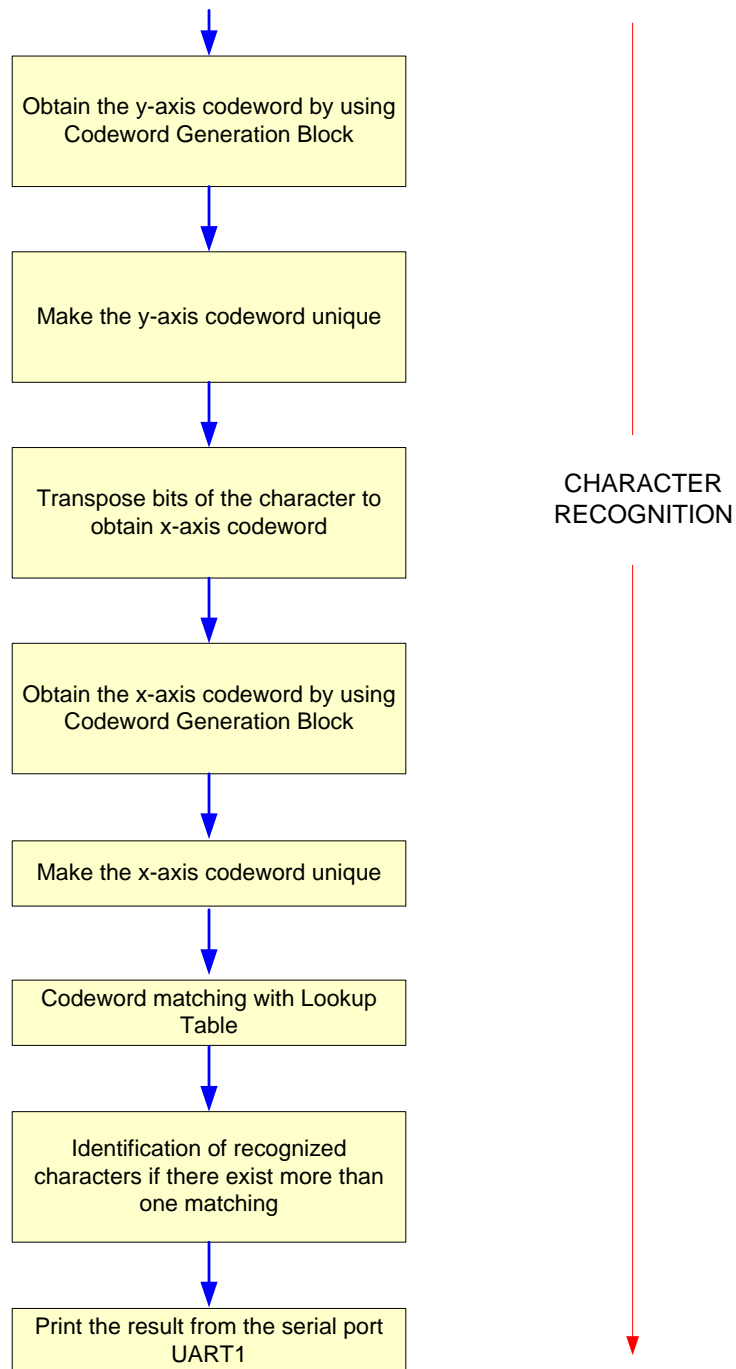


**Figure 4-28 Block Diagram of the Software Application**



**Figure 4-28 Block Diagram of the Software Application (Continued)**





**Figure 4-28 Block Diagram of the Software Application (Continued)**

All the steps in the algorithm proceed sequentially. The pseudo code of the algorithm which achieves character segmentation and character recognition parts is given for better explanation in the following sections.

#### 4.2.2.1 UP and DOWN SIDE TRIMMING

In up and down side trimming part, if there are black pixels at the top and bottom of the license plate which do not belong to character pixels, these pixels are discarded.

After the license plate image data is read from the on-chip RAM, it is kept in the array. To calculate the horizontal projection, image should be transposed. License plate is kept in the array such a manner: first index value of the array is the first row and first column of the image pixels, second index value of the array consists of second row and first column of the image pixels, etc. However, to calculate the horizontal projection, in the first index of the array should consist of first row and first column of the image pixels and second index of the array should consist of first row and second column of the image pixels. Therefore, by transposing the image, successive pixel values of a single row are kept successive index values of the array. The pseudo code of the transpose step is given below.

##### Transpose function:

```
for x from 1 to row size
    for y from 1 to column size
        calculate the index value of the original image array as  $(x * \text{column size} + y)$ 
        calculate the index value of the transposed image array as  $(y * \text{row size} + x)$ 
        set the value of the transposed image for calculated index as the value of the
        original image array for calculated index
```

After the image is transposed, horizontal projection of each row is calculated.

```
for x from 1 to row size
    for y from 1 to column size
        calculate the index value of the transposed image for the horizontal summation as
         $(x * \text{column size} + y)$ 
        set the horizontal array for that row index by summing the array value itself
        with the transposed array value for the calculated index
```

Index values which are smaller than the predefined threshold are found. By observing the difference between the successive index values, index values for up and down side trimming are found. After that, trimming is performed. Threshold value to find smaller index values is fixed and is set as 10 pixels to give tolerance to bright pixels at the top and bottom side of the image and to avoid trimming the characters from the middle of the plate region. The pseudo code of the algorithms is given in the following sections.

**FindSmallerIndex function:**

*for x from 1 to row size*

*if horizontal projection array for that row index is smaller than threshold*

*assign this value to the index value array*

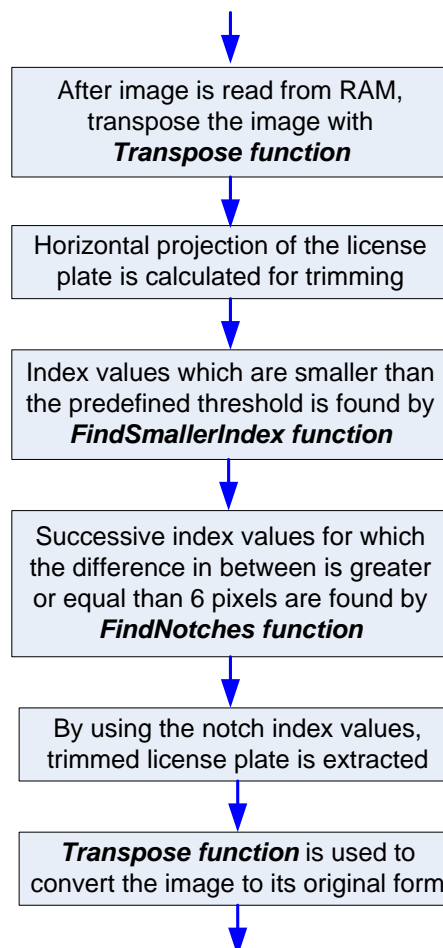
**FindNotches function:**

*for k from 1 to size of the index value array*

*if the difference between the successive values of the index value array is greater and equal than 6 pixels*

*assign these values as the notch indexes*

Notch index values give the row index values where the license plate should be trimmed. By using these index values, image data kept in the RAM is trimmed and then transposed by using the transpose function to convert the image to its original form. Flow diagram of the up and down side trimming is given in Figure 4-29.



**Figure 4-29 Flow Diagram of the Up and Down Side Trimming**

#### 4.2.2.2 CHARACTER SEGMENTATION

In the character segmentation part, each character is extracted one by one by observing notches in the plate region. After index values of each character, which indicate the position of the character, are found, algorithm proceeds with the character recognition part.

After up and down side trimming part, vertical projection of the license plate region is obtained by using the Image Manage Block integrated into the Qsys and for each column, sum value is obtained. Sum value consists of fluctuations. These fluctuations rise when passing through character pixels and fall when passing through bright regions between characters. The pseudo code for the vertical projection calculation is given below.

```
for c from 1 to column size of the plate region  
  write the column pixels to the Image Manage Block  
  if the status register of the Image Manage Block assigned as calculation is completed  
    read the result register of the Image Manage Block  
    assign the result value to the vertical sum array variable of the processor  
  else  
    wait until the calculation is completed, poll the status register continuously
```

After the vertical projection of the license plate is obtained, mean of the vertical sum is calculated. The pseudo code of the mean calculation is given below.

##### **mean array function:**

```
sum the all index values of the vertical sum array  
divide the total sum to the size of the vertical sum array
```

After the mean of the vertical sum array is obtained, vertical projection without DC level is obtained by subtracting the mean value from the original vertical sum array.

To find the positions of the characters, minimum value of the vertical sum without DC level array is found. Minimum value is always negative because of the DC level subtraction. Half of the minimum value gives threshold, which is used to find the index values of the vertical sum without DC level array smaller than this threshold. The pseudo code of the function which is used to find the minimum value is given below.

##### **FindMinimum function:**

```
assign the value at the first index of the array as minimum value  
for x from 1 to size of the array-1  
  get the value of the array at x+1 th index  
  if value of array at x+1 th index is smaller than the minimum value  
    assign the minimum value as value of array at x+1 th index
```

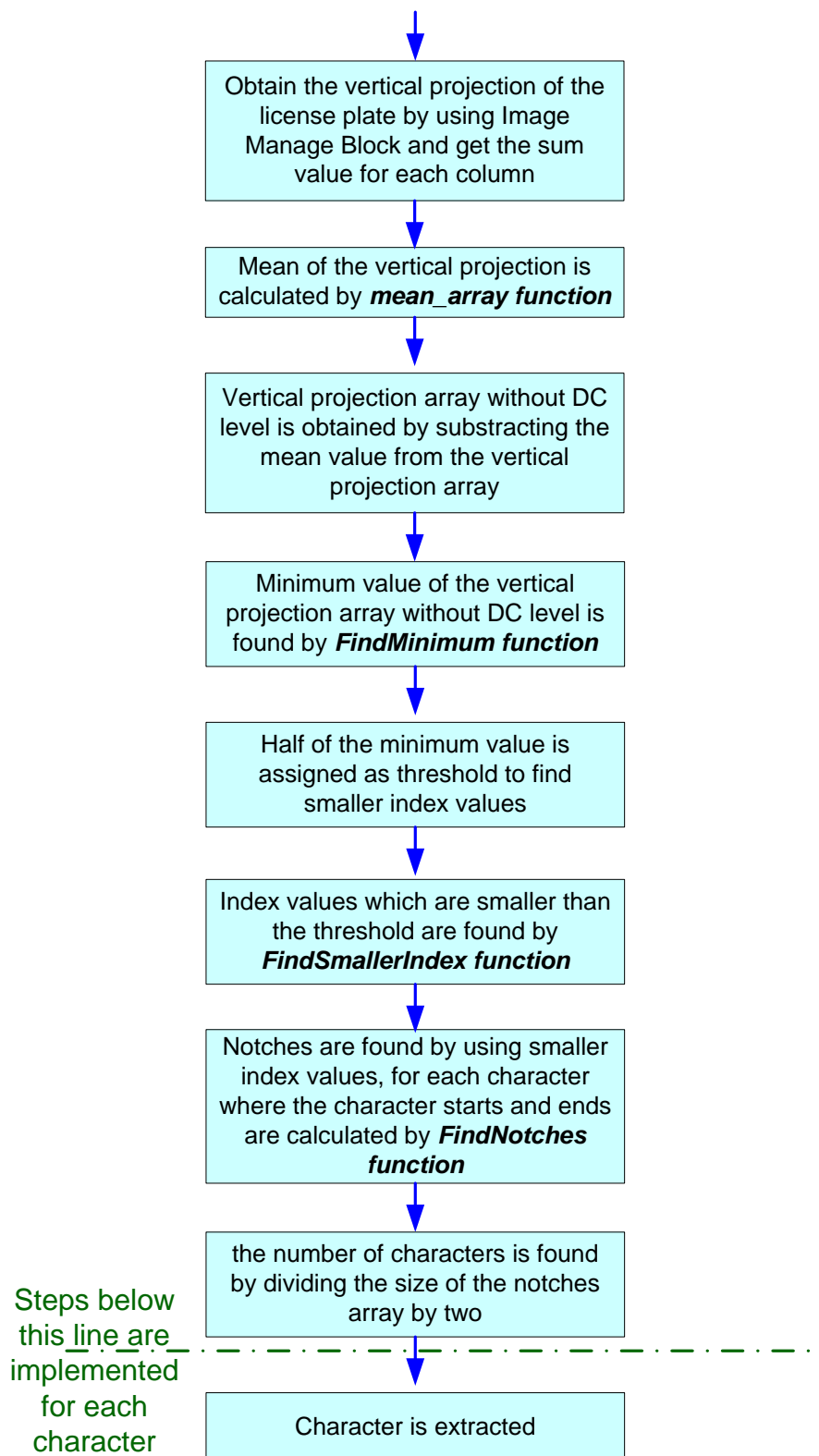
After the minimum value is calculated, half of the minimum value is obtained and assigned as threshold. This threshold value is used to find index values which are smaller than the threshold. After smaller index values are obtained, notches are found by implementing *FindSmallerIndex* and *FindNotches* functions whose pseudo codes are given in Section 4.2.2.1. The pseudo code of the algorithm is given below.

*threshold is found by dividing minimum value by 2*

*call for FindSmallerIndex function: smaller index values are found with FindSmallerIndex function*

*call for FindNotches function: notches and size of notches array are found with FindNotches function*

Each character is represented by two index values where character pixels start and end. Therefore, the number of characters can be found by dividing the size of the notches array by two. Since notches array keeps positions of characters, characters are extracted one by one using array values. After character is segmented, it is sent to the character recognition part. Flow diagram for the character segmentation part is given in Figure 4-30.



**Figure 4-30 Flow Diagram of the Character Segmentation**

In MATLAB implementation of the algorithm, after the character is extracted, morphological opening and connected component labeling with four neighboring are applied. However, in FPGA implementation, morphological opening and connected component labeling are not implemented. Connected component labeling takes large execution time for the processor and without connected component labeling, morphological opening misleads the character recognition part.

#### **4.2.2.3 CHARACTER RECOGNITION**

In character recognition part, codeword generation is achieved for each character and generated codeword is matched with codewords stored in the lookup table. If there is a match, character is recognized. If there is more than one match, character is identified with identification algorithms. If there is no match, character cannot be recognized.

After the character is segmented, it is recognized and it continues with recognition of other characters. Detailed explanation of the codeword generation and recognition steps of the algorithm are given in the following sections.

##### **4.2.2.3.1 CODEWORD GENERATION**

After the character is segmented, if there are black pixels at the top and bottom side of the character, up and down side trimming for each character is implemented because connected component labeling and morphological opening is not implemented for the large execution time. If the trimming operation doesn't take place, faulty codewords can be generated and this may result in recognition errors. After the trimming operation, x-axis and y-axis codewords of each segmented character are generated as explained in Section 3.3.1. For trimming same algorithm steps are followed as in Section 4.2.2.1. The pseudo code for the character trimming is given below.

*call for Transpose function to ease the trimming*

*calculation of horizontal projection as in Section 4.2.2.1*

*call for FindSmallerIndex function to find smaller index values at top and bottom which consists of all black pixels; threshold value is set as 1*

*call for FindNotches function to find trim positions*

*trim the character from up and down with found trim positions by using these values as index values of the array*

After character is trimmed, codewords are generated by using Codeword Generation Block which is integrated into the Qsys. For each row of the character image, codeword is generated and then this character string is made unique. Calculation result gives the y-axis codeword. The pseudo code for the algorithm is given below.

*for x from 1 to row size of the character*

*write row pixels to the Codeword Generation Block*

*if the status register of the Codeword Generation Block is assigned as calculation is completed*

*read the result register of the Codeword Generation Block*

*assign the result value to the y-axis codeword array variable of the processor*

*else*

*wait until the calculation is completed, poll the status register continuously*

After the y-axis codeword is calculated, codeword array is made unique by observing the successive values in the codeword array. The value is placed in the unique codeword array, if it exists more than one times successively. For example, if the initial codeword is “12111”, ‘2’ is discarded and unique codeword becomes “1”. However, if the initial codeword is “12211”, unique codeword becomes “21”. The pseudo code to make the codeword unique is given below.

**Unique function:**

*for x from 1 to length of the codeword -3*

*a window is generated with the codeword values such that:*

*value1 is equals to xth index value of the codeword*

*value2 is equals to x+1th index value of the codeword*

*value3 is equals to x+2th index value of the codeword*

*value4 is equals to x+3th index value of the codeword*

*if x is the 1 which means unique function starts now*

*if value1 equals to value2*

*assign value1 as the unique code for that row*

*else*

***if value1 is not equal both of the value2 and value 3***

*if there is no code in the unique array*

*assign value3 as the unique code for that row*

*else*

*if the previous unique code is equal not equal to value2*

*if value2 equals to value4*

*assign value2 as the unique code*

*else*

*assign value3 as the unique code*

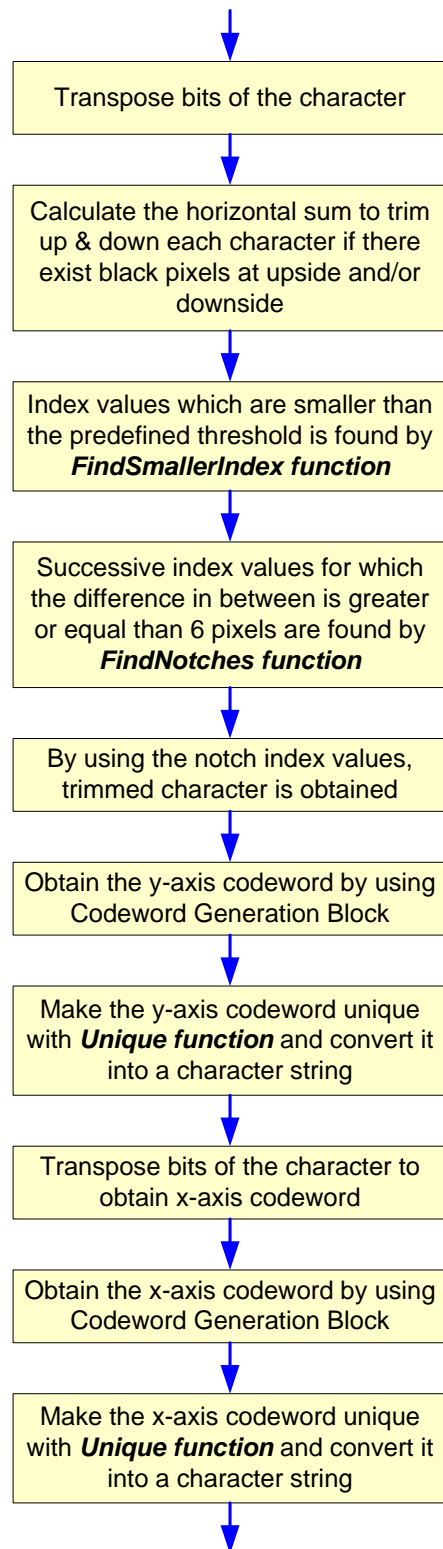
***else if value1 is not equal to value2 but equals to value3***

*if there is no code in the unique array*



*assign value1 as the unique code*  
***else if value1 equals to both of the value2 and value3***  
*if there is no code in the unique array*  
*assign value1 as the unique code*  
***else if value1 is equal to value2 but not equal to value3***  
*if there is no code in the unique array*  
*assign value2 as the unique code*

After the unique codeword for the y-axis is generated, it is converted into character string to compare the codeword with the values in the lookup table. After y-axis codeword is generated, bits of the character are transposed with the *Transpose function* and x-axis codeword is generated by using same approach. x-axis codeword is generated by using the column size of the character instead of using row size. The algorithm steps are same as given above. After unique codeword of the x-axis is generated, two codewords are not concatenated as in the MATLAB case because these codewords are matched sequentially in the character matching step of the software algorithm. Therefore, there is no need for concatenation. Flow diagram of the codeword generation step is given in Figure 4-31.



**Figure 4-31 Flow Diagram of the Codeword Generation**

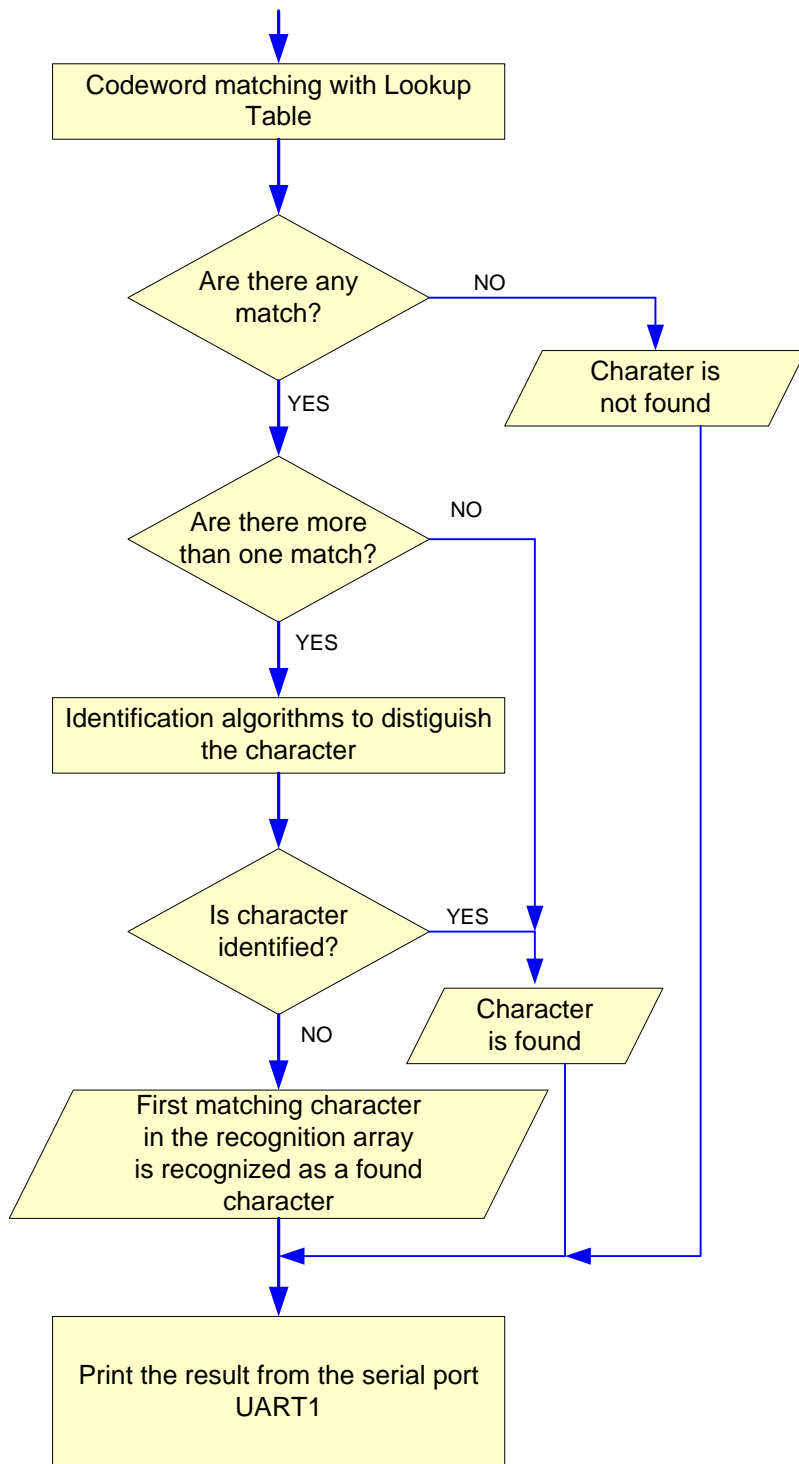
#### 4.2.2.3.2 CODEWORD MATCHING AND RECOGNITION

In codeword matching and recognition part, generated codewords are compared with the values in the lookup table. If there exists an entry which has the same x-axis and y-axis codewords with generated codewords, this entry is assigned as the recognized character. Different characters may be represented by the same codewords. For this case, more than one character is recognized. To distinguish these characters, identification algorithms are implemented for special cases. If the algorithm is not able to distinguish the character, first matching character in the database is recognized as a found character. If there is no match, character cannot be recognized. There are 88 entries in the lookup table. Characters in the lookup table with corresponding codeword values are given in Appendix-D.

The pseudo code for the matching algorithm is given below.

```
for x from 1 to size of the lookup table  
    get the lookup table entry for x  
    assign y-axis codeword to codeword1 variable  
    assign x-axis codeword to codeword2 variable  
    if y-axis codeword of the generated codeword equals to codeword1  
        if x-axis codeword of the generated codeword equals to codeword2  
            assign this lookup table entry to the recognized characters array
```

After generated codewords are matched with the entries of the lookup table, recognized characters are kept in the recognition array. If there is more than one entry in the recognition array, identification algorithms are implemented. If the identification algorithm is not able to identify the character, first matching character in the recognition array is recognized as the found character. After the implementation of the identification algorithm, result of the recognition process is sent to GUI interface by using serial port. The explanations and pseudo codes for the identification algorithms are given in the following sections. The flow diagram of the codeword matching and recognition part is given in Figure 4-32.



**Figure 4-32 Flow Diagram of the Codeword Matching and Recognition**

#### 4.2.2.3.2.1 IDENTIFICATION ALGORITHMS

Codewords are not specific to characters. Different characters can be represented with same codewords. In such cases, to identify the exact character in the plate region, identification algorithms are implemented. The algorithm determines which identification algorithm should be implemented by looking at the ASCII values of the recognition array. The pseudo code of the decision making algorithm is given below.

- *implement identification of letter and number algorithm*
- *if there are three recognized characters in the recognition array*
  - *If ASCII codes of these characters are 51, 53 and 56*
    - *implement identification of 3, 5 and 8 algorithm*
  - *else if ASCII codes of these characters are 51, 56 and 57*
    - *implement identification of 3, 8 and 9 algorithm*
  - *else if ASCII codes of these characters are 85, 86 and 89*
    - *implement identification of U, V and Y algorithm*
  - *else if ASCII codes of these characters are 48, 52 and 55*
    - *implement identification of 0, 4 and 7 algorithm*

In the assigned identification algorithms, *Recognition\_Codeword function* is used. This function calculates the codeword of a single row by using the Codeword Generation Block of the Qsys. Row index value, for which codeword is calculated, is specified inside the identification algorithm. The pseudo code of the *Recognition\_Codeword function* is given below.

##### **Recognition\_Codeword function:**

```
write all pixels belongs to sampled row to the Codeword Generation Block  
if the status register of the Codeword Generation Block is assigned as calculation is  
completed  
    read the result register of the Codeword Generation Block  
    return from the function with result value  
else  
    wait until the calculation is completed, poll the status register continuously
```

#### 4.2.2.3.2.1.1 IDENTIFICATION OF LETTER AND NUMBER

In Turkish license plates, first two and last two characters should be numbers and third character should be a letter. ASCII codes of numbers are represented in between 48 and 57 in decimal and ASCII codes of the letters are equal to or greater than 65. Therefore, for the recognized first two and last two characters, corresponding ASCII codes should be in between 48 and 57 and for the third character ASCII code for the recognized character should be equal to or greater than 65. If the corresponding ASCII codes are different from

the defined interval, recognition result is eliminated. The pseudo code of the algorithm is given below.

```
if character is one of the first two or last two characters  
    for x from 1 to size of the recognition array for this character  
        get the ASCII code at the xth index of the recognition array  
        if ASCII code is smaller than 48 or greater than 57  
            assign 0 to xth index of the recognition array  
else if character is the third character  
    for y from 1 to size of the recognition array for this character  
        get the ASCII code at the yth index of the recognition array  
        if ASCII code is smaller than 65  
            assign 0 to yth index of the recognition array
```

In Turkish license plates, if the fifth character is a letter, fourth character should also be a letter. The pseudo code of the algorithm is given below.

```
if ASCII code of the fifth character is equal to or greater than 65  
    get the ASCII codes in the 4th row of the recognition array  
    for x from 1 to size of the recognition array of the 4th row for this character  
        get the ASCII code at the xth index of the recognition array of the 4th row  
        if ASCII code is smaller than 65  
            assign 0 to xth index of the recognition array of the 4th row
```

#### **4.2.2.3.2.1.2 IDENTIFICATION OF 0, 4 AND 7**

'0', '4' and '7' numbers are identified as explained in the Section 3.3.2.2. The pseudo code of the identification algorithm implemented in the software is given below.

```
find the row_index1 where the first sample is taken as in Equation (3.15)  
find the row_index2 where the second sample is taken as in Equation (3.16)  
take the first sample by using row_index1  
take the second sample by using row_index2  
call for Recognition_Codeword function for the first sample and obtain the codeword1  
call for Recognition_Codeword function for the second sample and obtain the codeword2  
if codeword1 equals to 2 and codeword2 equals to 2  
    recognized character is '0'
```

*else if codeword1 equals to 1 and codeword2 equals to 2*  
     *recognized character is '4'*  
*if codeword1 equals to 1 and codeword2 equals to 1*  
     *recognized character is '7'*  
*else*  
     *character cannot be recognized*

#### **4.2.2.3.2.1.3 IDENTIFICATION OF U, V AND Y**

'U', 'V' and 'Y' letters are identified as explained in the Section 3.3.2.3. The pseudo code of the identification algorithm implemented in the software is given below.

*find the row\_index1 where the first sample is taken as in Equation (3.17)*  
*find the row\_index2 where the second sample is taken as in Equation (3.18)*  
*take the first sample by using row\_index1*  
*take the second sample by using row\_index2*  
*call for Recognition\_Codeword function for the first sample and obtain the codeword1*  
*call for Recognition\_Codeword function for the second sample and obtain the codeword2*  
*if codeword1 equals to 2 and codeword2 equals to 2*  
     *recognized character is 'U'*  
*else if codeword1 equals to 1 and codeword2 equals to 2*  
     *recognized character is 'V'*  
*if codeword1 equals to 1 and codeword2 equals to 1*  
     *recognized character is 'Y'*  
*else*  
     *character cannot be recognized*

#### **4.2.2.3.2.1.4 IDENTIFICATION OF 3, 5 AND 8**

'3', '5' and '8' numbers are identified as explained in the Section 3.3.2.4. The pseudo code of the identification algorithm implemented in the software is given below.

*find the row\_index1 where the first sample is taken as in Equation (3.19)*  
*find the row\_index2 where the second sample is taken as in Equation (3.20)*  
*take the first sample by using row\_index1*  
*take the second sample by using row\_index2*  
*call for Recognition\_Codeword function for the first sample and obtain the codeword1*  
*call for Recognition\_Codeword function for the second sample and obtain the codeword2*  
*if codeword1 equals to 2 and codeword2 equals to 2*  
     *recognized character is '8'*

*else if codeword1 equals to 1 and codeword2 equals to 2*  
*recognized character is '3'*  
*if codeword1 equals to 1 and codeword2 equals to 1*  
*recognized character is '5'*  
*else*  
*character cannot be recognized*

#### **4.2.2.3.2.1.5 IDENTIFICATION OF 3, 8 AND 9**

'3', '8' and '9' numbers are identified as explained in the Section 3.3.2.5. The pseudo code of the identification algorithm implemented in the software is given below.

*find the row\_index1 where the first sample is taken as in Equation (3.21)*  
*find the row\_index2 where the second sample is taken as in Equation (3.22)*  
*take the first sample by using row\_index1*  
*take the second sample by using row\_index2*  
*call for Recognition\_Codeword function for the first sample and obtain the codeword1*  
*call for Recognition\_Codeword function for the second sample and obtain the codeword2*  
*if codeword1 equals to 1 and codeword2 equals to 1*  
*recognized character is '3'*  
*else if codeword1 equals to 1 and codeword2 equals to 2*  
*recognized character is '9'*  
*if codeword1 equals to 2 and codeword2 equals to 2*  
*recognized character is '8'*  
*else*  
*character cannot be recognized*

Algorithms for “Identification of 6 and B” and “Identification of 2 and Z” are not implemented in the processor side because there are slight differences between these characters. Without applying morphological operations and connected component labeling, most likely recognition result will be erroneous.



## CHAPTER 5

### EXPERIMENTAL RESULTS

License plate detection and recognition algorithm is first developed and verified in MATLAB. After the development is finished and the verification is completed, only recognition part of the algorithm is implemented in the FPGA for single line plate cases. Some processing steps, such as morphological opening and connected component labeling, are not implemented in the FPGA because they take more time than expected and execution time increases. Test results of the MATLAB and FPGA implementations are explained in this section.

In MATLAB implementation, after detection of the plate position, plate region is stored to use the image in FPGA. In FPGA tests, these localized plate regions are used for the implementation of the recognition algorithm. Size of the plate region is not fixed. Size information is sent from the GUI designed for the system. After the image is sent to the FPGA, FPGA starts processing and displays the recognition results on the terminal which is connected to the serial port.

In the tests, 175 different images for different vehicles are worked on. Images are taken with different light conditions and 4-7 meters away from the vehicle. The plate database is listed in Table D-2.

#### 5.1 TEST RESULTS FOR THE MATLAB IMPLEMENTATION

While testing the algorithm in MATLAB, first, position of the plate region is detected by the Gabor transform and morphological closing is applied. Then, each character is segmented and recognized one by one. Test results for the MATLAB implementations are given in Table 5-1.

**Table 5-1 Test results for the MATLAB Implementation**

<b>Image Number</b>	<b>Original License Plate</b>	<b>Recognized License Plate</b>	<b>Explanation</b>	<b>Plate Region Extraction Success*</b>
0	38 NE 941	38NE941		1
1	06 ANZ 72		Error in locating the license plate	0

**Table 5-1 Test results for the MATLAB Implementation (Continued)**

<b>Image Number</b>	<b>Original License Plate</b>	<b>Recognized License Plate</b>	<b>Explanation</b>	<b>Plate Region Extraction Success*</b>
2	35 EL 786		License plate located successfully. It is a double line plate, therefore; recognition algorithm is not implemented	1
3	06 LS 717		Error in locating the license plate	0
4	06 EEA 80	06E-A8-	2 error (E)( 0)	1
5	06 AAH 83	06AA-83	1 miss (H)	1
6	06 ZTM 61	06ZIJ61	2 miss (T )(M)	1
7	06 ACF 96		Error in locating the license plate	0
8	06 NV 015	06NV015		1
9	34 UEN 01		Error in locating the license plate	0
10	06 AKV 30		Error in locating the license plate	0
11	06 VCU 16	06VC16	1 miss (U)	1
12	06 FG 382	06FG962	2 error (3-9)(8-6)	1
13	06 V 6304	-6V630-	2 error (0)( 4)	1
14	06 GJ 988	06GL868	1 miss (J) + 2 error (9-8)(8-6)	1
15	06 GD 229	40BG0229	1 extra (4) + 2 error (6-B)(D-0)	1
16	06 BF 926	06-F926	1 error (B)	1
17	11 AK 680		Error in locating the license plate	0
18	06 ZVP 28	06ZV728	1 error (P-7)	1
19	41 DA 848		Error in locating the license plate	0
20	06 VTE 73	06VE79	1 miss(T) + 1 error (3-9)	1
21	06 AJC 01	02AIC01	1 miss (J) + 1 error (6-2)	1
22	06 BS 913	06-389	1 miss (1) + 4 error (B)(S-3)(9-8)(3-9)	1
23	06 E 9787	06E8667	3 error (9-8)(7-6)(8-6)	1

**Table 5-1 Test results for the MATLAB Implementation (Continued)**

<b>Image Number</b>	<b>Original License Plate</b>	<b>Recognized License Plate</b>	<b>Explanation</b>	<b>Plate Region Extraction Success*</b>
24	06 UE 242	06UEZ42	1 error (2-Z)	1
25	45 EE 616		License plate located successfully. It is a double line plate, therefore; recognition algorithm is not implemented	1
26	06YAM 63	---P-631	1 extra (1) + 5 error (0)(6)(Y)(A-P)(M)	1
27	06 YUY 61	06YY61	1 miss (U)	1
28	06 GH 039	06G099	1 miss (H) + 1 error (3-9)	1
29	06 LN 567		Error in locating the license plate	0
30	06 FFA 51	06FF-5	1 error (A) + 1 miss (1)	1
31	06 YNP 76		Error in locating the license plate	0
32	06 RED 88	06RE068	2 error (D-0)(8-6)	1
33	06 Z 6339	-1DBZ6-99	2 extra (-)(1) + 4 error (0-D) (6-B)(3)(3-9)	1
34	06 MVB27	06MUB27	1 error (V-U)	1
35	06 YDU 85	06Y085	1 miss (U) + 1 error (D-0)	1
36	35 KFM 75	5-KFM-2	4 error (3-5) (5) (7) (5-2)	1
37	06 KNT 90	06KIJ9--	3 miss (N)(T)(0)	1
38	06 RNM90	06RN-90	1 miss (M)	1
39	06 YRU 41	06YRLL41	1 miss (U)	1
40	06 ZFS 48	06ZF308	2 error (S-3)(4-0)	1
41	32 AN 568	2-NZ6-	1 miss (3) + 3 error (A) (5-Z) (8)	1
42	06 JV 031		Error in locating the license plate	0
43	06 ZTT 60	06Z60	2 miss (T)(T)	1
44	06 GM 575	06GM5-5	1 error (7)	1
45	06 ZVS 41	06ZV341	1 error (S-3)	1
46	06 LM 649	0--49	3 miss (6) (L)(M) + 1 error (6)	1

**Table 5-1 Test results for the MATLAB Implementation (Continued)**

<b>Image Number</b>	<b>Original License Plate</b>	<b>Recognized License Plate</b>	<b>Explanation</b>	<b>Plate Region Extraction Success*</b>
47	06 LMS 97		Error in locating the license plate	0
48	06 JM 694	6-69	3 miss (0)(J)(4) + 1 error (M)	1
49	06 LPC 54		Error in locating the license plate	0
50	06 VB 035	06Y60-2	4 error (V-Y)(B-6) (5-2) (3)	1
51	06 VTK 75	06VK-5	1 miss (T) + 1 error (7)	1
52	06 ZVJ 64	--ZV-4	2 miss(J)(6) + 2 error (0)(6)	1
53	06 TNA 59	46NA59	1 miss (T) + 1 error (0-4)	1
54	06 EM 653	06EM653		1
55	06 DJ 294	06UIZ94	3 error (D-U)(J-I) (2-Z)	1
56	06 FKS 81	06FK381	1 error (S-3)	1
57	06 MLD 18	06ML013	2 error (8-3)(D-0)	1
58	06 KJ 258		Error in locating the license plate	0
59	06 YZC 78	-BUZC-6	1 miss (0) + 4 error (6-B)(Y-U)(7)(8-6)	1
60	06 H 0184	06IF-18	3 miss (H)(0)(4)	1
61	06 GC 115		Error in locating the license plate	0
62	06 HB 496		Error in locating the license plate	0
63	06 FVY 64	06FUY64	1 error (V-U)	1
64	06 SC 197		Error in locating the license plate	0
65	06 SM 805	06SH605	2 error (M-H) (8-6)	1
66	06 KTY 39	06KIY39-	1 extra (-) + 1 error (T-I)	1
67	06 AD 406	06A0406	1 error (D-0)	1
68	06 YUF 55	06YF55	1 miss (U)	1
69	06 YBT 50	06Y650	1 miss (T) + 1 error (B-6)	1
70	06 YML 50		Error in locating the license plate	0
71	06 ZSS 91	10BZ5391	1 extra (1) + 3 error (6-B)(S-5)(S-3)	1

**Table 5-1 Test results for the MATLAB Implementation (Continued)**

<b>Image Number</b>	<b>Original License Plate</b>	<b>Recognized License Plate</b>	<b>Explanation</b>	<b>Plate Region Extraction Success*</b>
72	06 TRZ 81		Error in locating the license plate	0
73	06 VVL 74	06IIV74	2 miss (V)(L)	1
74	06 KJS 97	06KJ597	1 error (S-5)	1
75	06 ZFE 94		Error in locating the license plate	0
76	06 BM 277	06BMZ77	1 error (2-Z)	1
77	06 GV 261	06GUZ61	2 error (V-U)(2-Z)	1
78	06 ZVR 08	-BZVR08	1 miss (0) + 1 error (6-B)	1
79	16 NB 505	6-BZ02	1 miss (1) + 3 error (N) (5-Z)(5-2)	1
80	06 KYY 24	-0BKYYZ4 -	2 extra (-)(-) + 2 error (6-B)(2-Z)	1
81	06 LCE 24	06ICE24	1 miss (L)	1
82	06 TAV 30	06AV30	1 miss(T)	1
83	06 BB 162		Error in locating the license plate	0
84	06 PB 353		Error in locating the license plate	0
85	06 VB 035		Error in locating the license plate	0
86	06 YNZ 47	46YHZ47	2 error (0-4) (N-H)	1
87	06 FM 787	-0BFM787	1 extra (-) + 1 error (6-B)	1
88	06 UB 952		Error in locating the license plate	0
89	06 ZBK 09		Error in locating the license plate	0
90	06 SG 911		Error in locating the license plate	0
91	06 RNH 19		Error in locating the license plate	0
92	06 ZMJ 55		Error in locating the license plate	0
93	06 YML 39	06YML39		1
94	06 EAE 87	46EAE87	1 error (0-4)	1
95	06 EB 195	06-BL95	2 error (E)(1-L)	1
96	06 UB 008		Error in locating the license plate	0
97	06 HLD 71	06I-L071	1 miss(H) + 1 error (D-0)	1

**Table 5-1 Test results for the MATLAB Implementation (Continued)**

<b>Image Number</b>	<b>Original License Plate</b>	<b>Recognized License Plate</b>	<b>Explanation</b>	<b>Plate Region Extraction Success*</b>
98	06 ZKM 55	06ZKM55		1
99	06 VTF 54	46Y-F56	1 miss (T) + 3 error (0-4)(V-Y)(4-6)	1
100	06 JH 617		Error in locating the license plate	0
101	06 KUH 65	06KILL-65	2 miss (U)(H)	1
102	06 NB 150		Error in locating the license plate	0
103	06 FJ 557	06FIL557	1 miss (J)	1
104	06 YTE 47	06Y--77	1 miss (T) + 2 error (E)(4-7)	1
105	06 DJ 294	06DIZ94	1 miss (J) + 1 error (2-Z)	1
106	06 SC 197		Error in locating the license plate	0
107	06 GV 613		Error in locating the license plate	0
108	06 VV 941		Error in locating the license plate	0
109	06 AB 664	-0BA6B64	1 extra (-) + 3 error (6-B)(B-6)(6-B)	1
110	06 MML 54	06MML54		1
111	06 UM 938		Error in locating the license plate	0
112	06 GJ 015	06GIL015	1 miss (J)	1
113	06 AB 789	06A6789	1 error (B-6)	1
114	06 EHR 26	-0BEL-R26	1 extra (-) + 1 error (6-B) + 1 miss (H)	1
115	06 PG 742		Error in locating the license plate	0
116	06 VPC 41		Error in locating the license plate	0
117	06 MUZ 64		Error in locating the license plate	0
118	06 FPL 06	06FDL06	1 error (P-D)	1
119	06 GN 623		Error in locating the license plate	0
120	06 RVR 44	06RUR44	1 error (V-U)	1
121	06 YMH 60	06YHTT64	1 miss (H) + 2 error (M-H)(0-4)	1
122	06 TNA 59	06-NA59	1 miss (T)	1

**Table 5-1 Test results for the MATLAB Implementation (Continued)**

<b>Image Number</b>	<b>Original License Plate</b>	<b>Recognized License Plate</b>	<b>Explanation</b>	<b>Plate Region Extraction Success*</b>
123	06 MZK 93		Error in locating the license plate	0
124	06 JB 489		Error in locating the license plate	0
125	06 LB 509	-0BL6Z49	1 extra (-) + 4 error (6-B)(B-6)(5-Z)(0-4)	1
126	06 TKL 06	06IKL06	1 miss (T)	1
127	06 KPM 96		Error in locating the license plate	0
128	06 YD 468		Error in locating the license plate	0
129	06 BN 824	-6BH824	2 error (0)(N-H)	1
130	06 KKY 99	06KKY99		1
131	06 JK 870		Error in locating the license plate	0
132	06 YB 868		Error in locating the license plate	0
133	06 GA 049	-6-4-2	2 miss (0)(6) + 5 error (G-6)(A)(0-4)(4)(9-2)	1
134	06 VUF 61		Error in locating the license plate	0
135	06 GV 354		Error in locating the license plate	0
136	06 YZC 78	06YZC78		1
137	06 MDB 52	06H065	3 error (M-H)(D-0)(B-6) + 1 miss (2)	1
138	06 ZHT 89	06ZT-T89	1 miss (H)	1
139	06 BB 162	06BBL62	1 error (1-L)	1
140	06 ZZE 84	06ZZE87	1 error (4-7)	1
141	06 LEE 85	06E-65	1 miss (L) + 2 error (E)(8-6)	1
142	06 DM 529	06DM529		1
143	06 KHZ 04	06KIZ04	1 miss (H)	1
144	06 GV 492		Error in locating the license plate	0
145	06 YMH 26	06YMLL26	1 miss (H)	1
146	06 KVF 51		Error in locating the license plate	0
147	06 VVR 58	-0BUVR58	1 extra (-) + 2 error (6-B)(V-U)	1

**Table 5-1 Test results for the MATLAB Implementation (Continued)**

<b>Image Number</b>	<b>Original License Plate</b>	<b>Recognized License Plate</b>	<b>Explanation</b>	<b>Plate Region Extraction Success*</b>
148	06 KFY 42	06KFY72	1 error (4-7)	1
149	06 GA 965	06GA965		1
150	06 GD 009	06-0-49	4 error (G)(D-0)(0)(0-4)	1
151	06 GH 264	06GIL264	1 miss (H)	1
152	06 VKC 70	46YKC70	2 error (0-4)(V-Y)	1
153	06 KG 629		Error in locating the license plate	0
154	06 GR 895	46B-L692	5 error (0-4)(G-B) ( R)(8-6)(5-2) + 1 extra (L)	1
155	06 FM 969	06F-969	1 error (M)	1
156	06 SS 798		Error in locating the license plate	0
157	06 TKD 26		Error in locating the license plate	0
158	34 VD 4751	94Y04-1	4 error (3-9)(V-Y) (D-0)(7) + 1 miss (5)	1
159	06 RZJ 36	46RZJ36	1 error (0-4)	1
160	06 RHS 42	06RH502	2 error (S-5)(4-0)	1
161	06 BM 277		Error in locating the license plate	0
162	06 BB 162		Error in locating the license plate	0
163	06 FU 644	46FU644	1 error (0-4)	1
164	06 BL 040	06-L040	1 error (B)	1
165	06 TSZ 03	06SZ03	1 miss (T)	1
166	06 EM 653		Error in locating the license plate	0
167	06 EKN 80	06EKN80		1
168	06 MKC 25		Error in locating the license plate	0
169	06 MHP 10	06M70	2 miss (H)(1) + 1 error (P-7)	1
170	06 MBH 36	-0BM6LL96	1 extra (-) + 3 error (6-B)(B-6)(3-9) + 1 miss (H)	1
171	06 MZK 93		Error in locating the license plate	0
172	06 ZFS 22	06ZF322	1 error (S-3)	1



**Table 5-1 Test results for the MATLAB Implementation (Continued)**

<b>Image Number</b>	<b>Original License Plate</b>	<b>Recognized License Plate</b>	<b>Explanation</b>	<b>Plate Region Extraction Success*</b>
173	06 JB 489		Error in locating the license plate	0
174	06 ZVJ 64	10BZV64	1 extra (I)+ 1 error (6-B) + 1 miss (J)	1

Note: error: Error in Character Recognition

miss: Error in Character Segmentation

extra: Extra Character is Detected

\* Plate Region Extraction Success 1: Plate region is localized successfully

0: Error in the localization of the plate region

For 57 images out of 175 test images, plate regions are not extracted successfully. For 25 of them, no candidate plate region is extracted and for 32 of them, plate regions are localized erroneously.  $((175-57)/175) \times 100 = 67.43\%$  is the plate region extraction ratio of the whole test image database for the MATLAB implementation. For the extraction of the plate region, Gabor transform and morphological closing are used. Gabor transform gives rough estimate about the boundary of the plate region and searches for the black to white pixel transitions. Transition regions give local pieces of information about the plate position. Information obtained by the Gabor filter is combined and it constructs the region of interest. For some plates, plate region is dirty, characters are too small or distances between the characters are too large. In such cases, Gabor filter gives erroneous estimate about the plate region. For these reasons, plate region cannot be extracted successfully. Reasons for the plate extraction failures are given in Table 5-2.

**Table 5-2 Reasons of the License Plate Extraction Failures**

<b>Image Number</b>	<b>Original License Plate</b>	<b>Explanation</b>	<b>Reason for Failure</b>
1	06 ANZ 72	Error in locating the license plate	Size of the plate region after Gabor transform is not applicable for filtering the image with predefined plate width and length parameters.
3	06 LS 717		
127	06 KPM 96		
161	06 BM 277		
162	06 BB 162		
166	06 EM 653		







**Table 5-2 Reasons of the License Plate Extraction Failures (Continued)**

<b>Image Number</b>	<b>Original License Plate</b>	<b>Explanation</b>	<b>Reason for Failure</b>
9	34 UEN 01	Error in locating the license plate	Characters are too small and dirty. Therefore, Gabor transform cannot estimate place position.
10	06 AKV 30		
19	41 DA 848	Error in locating the license plate	Plate region is noisy and it is trimmed erroneously. Therefore, whole plate region cannot be extracted.
29	06 LN 567		
91	06 RNH 19		
171	06 MZK 93		
31	06 YNP 76	Error in locating the license plate	Characters in the plate region are separated with large distances. Therefore, Gabor transform gives erroneous estimate and plate region consists of disjoint parts
42	06 JV 031		
47	06 LMS 97		
17	11 AK 680		
7	06 ACF 96		
58	06 KJ 258		
61	06 GC 115		
62	06 HB 496		
64	06 SC 197		
75	06 ZFE 94		
84	06 PB 353		
85	06 VB 035		
88	06 UB 952		
90	06 SG 911		
96	06 UB 008		
100	06 JH 617		
102	06 NB 150		
106	06 SC 197		
107	06 GV 613		
111	06 UM 938		
115	06 PG 742		
116	06 VPC 41		
117	06 MUZ 64		
119	06 GN 623		
123	06 MZK 93		
124	06 JB 489		
128	06 YD 468		

**Table 5-2 Reasons of the License Plate Extraction Failures (Continued)**

<b>Image Number</b>	<b>Original License Plate</b>	<b>Explanation</b>	<b>Reason for Failure</b>
131	06 JK 870	Error in locating the license plate	Characters in the plate region are separated with large distances. Therefore, Gabor transform gives erroneous estimate and plate region consists of disjoint parts
132	06 YB 868		
135	06 GV 354		
144	06 GV 492		
153	06 KG 629		
156	06 SS 798		
168	06 MKC 25		
173	06 JB 489		
49	06 LPC 54	Error in locating the license plate	Due to illumination, plate region is too small for filtering the image with predefined plate width and length parameters.
70	06 YML 50	Error in locating the license plate	Length of the some regions in the plate region is too small for filtering the image with predefined plate width and length parameters. Therefore, whole plate region cannot be extracted or wrong plate region is extracted.
72	06 TRZ 81		
89	06 ZBK 09		
108	06 VV 941		
134	06 VUF 61		
146	06 KVF 51		
157	06 TKD 26		
83	06 BB 162	Error in locating the license plate	Wrong candidate plate region is assigned because bright pixel density of wrong region is larger than exact region.
92	06 ZMJ 55		

To detect the position of the plate region in the image, Gabor transform of the image is binarized with Otsu thresholding and morphological closing is applied to combine the image regions which are the responses of the Gabor transform. For some plates, Gabor transform does not give uniform results. Due to large distances between characters (Figure 5-1 (c)), dirty plate regions with small characters (Figure 5-1 (a)) and dark plate regions (Figure 5-1 (b)), Gabor response is divided into several parts and plate region cannot be extracted successfully. Gabor responses of several images for which the place of the license plate cannot be found are shown in Figure 5-1.

	Original Image	Gabor Response of the Plate Region
(a)		
(b)		
(c)		

**Figure 5-1 Sample plates and their Gabor responses for which the plates cannot be extracted from the image**

For single line plates which are localized correctly, character segmentation and character recognition algorithms are implemented. According to Table 5-1, these steps are not implemented for 59 images ((for 57 images plate region is not localized correctly) + (2 images are the double line plates)). For remaining  $175-59 = 116$  images, algorithm continues with the recognition part. Character recognition and segmentation rates for the 116 images whose plate regions are localized correctly are given in Table 5-3.

**Table 5-3 Character Recognition Rate for the MATLAB Implementation**

Character	Number of Occurrences	Number of Extracted Characters	Number of Recognized Characters	Extraction Rate (%)	Recognition Rate (%)
0	142	136	117	95.77	86.03
1	22	18	16	81.82	88.89
2	26	25	19	96.15	76.00
3	21	20	10	95.24	50.00
4	37	35	27	94.59	77.14
5	36	35	27	97.22	77.14
6	144	141	123	97.92	87.23
7	23	23	17	100.00	73.91
8	30	30	20	100.00	66.67

**Table 5-3 Character Recognition Rate for the MATLAB Implementation (Continued)**

Character	Number of Occurrences	Number of Extracted Characters	Number of Recognized Characters	Extraction Rate (%)	Recognition Rate (%)
9	33	33	29	100.00	87.88
A	16	16	12	100.00	75.00
B	17	17	7	100.00	41.18
C	6	6	6	100.00	100.00
D	12	12	2	100.00	16.67
E	18	18	14	100.00	77.78
F	17	17	17	100.00	100.00
G	12	12	9	100.00	75.00
H	14	1	1	7.14	100.00
J	11	3	2	27.27	66.67
K	16	16	16	100.00	100.00
L	12	8	8	66.67	100.00
M	24	21	15	87.50	71.43
N	11	10	7	90.91	70.00
P	3	3	0	100.00	0.00
R	11	11	10	100.00	90.91
S	11	11	2	100.00	18.18
T	16	2	1	12.50	50.00
U	8	2	2	25.00	100.00
V	23	22	13	95.65	59.09
Y	20	20	18	100.00	90.00
Z	21	21	21	100.00	100.00
<b>Total</b>	<b>813</b>	<b>745</b>	<b>588</b>	<b>91.64</b>	<b>78.93</b>

For the MATLAB implementation, it is concluded from Table 5-1 and Table 5-3 that,

(57 plates whose characters are not extracted due to location error x 7 characters / plate) + (2 plates whose characters are not extracted because they are double line plates x 7 characters / plate) + (68 characters are missed while segmenting characters) + (157 characters are decided wrongly or cannot be decided) = totally 638 characters are not identified correctly. Characters are missed either position of the character is detected erroneously and only some part of the character is extracted or position of the character cannot be detected by the notch detection method.

For the recognition, 115 plates x 7 characters / plate + 1 plate x 8 characters / plate = 813 characters are worked with. 68 characters are not segmented accurately. Therefore, character segmentation percentage becomes  $((813-68) / 813) \times 100 = 91.64\%$ . For the

recognition of the segmented characters, 745 characters are used and 588 characters are identified. 157 characters are identified wrongly or cannot be identified. Therefore, recognition percentage over the extracted characters becomes  $(588/745) \times 100 = 78.93\%$ . Recognition rate of the characters over the whole localized plates is  $((588 \text{ identified characters} / (115 \text{ plates} \times 7 \text{ characters / plate} + 1 \text{ plate} \times 8 \text{ characters / plate} = 813 \text{ characters})) \times 100 = 72.32\%$ . For the whole image database (174 images  $\times$  7 characters / plate + 1 plate  $\times$  8 characters / plate = 1226 characters), the overall recognition rate decreases to  $(588 \text{ identified characters} / 1226 \text{ characters}) \times 100 = 47.96\%$  because of the plate region localization errors.

Segmentation rates of the 'H' and 'U' letters are considerable low as given in Table 5-3. For the character segmentation, notch detection method is used. Due to black pixels at the two edges of the 'H' and 'U' letters, two peaks are constructed and there is a minimum value in between in the vertical projection. Therefore, 'H' and 'U' characters are divided into two and cannot be extracted correctly. If these characters are extracted correctly, they can be recognized accurately.

Segmentation and recognition rates for the 'J' and 'T' characters are also low because tail of the 'J' character and horizontal black pixels at the top of the 'T' character are usually trimmed when segmenting the character due to low vertical projection values at the edges of the character image. As a result of the trimming, these characters either are not segmented successfully or if segmented, they are confused with 'I' or 'L' characters and cannot be identified correctly.

Although 'S' character is extracted successfully, recognition rate is low. Since 'S' has the same ideal codeword pair, "12121-232", with '3', '5' and '8' numbers, confliction occurs. Therefore, it is usually confused with one of these numbers. If the 'S' character is the third character in the license plate or if it is in between the two identified letters in the plate region, character is recognized successfully with the identification of letter and number algorithm.

(1, I, L, T), (2-Z), (0,4, 7, D, O, P), (0, D), (6,8), (B,8), (B-6), (3-8-9), (3,5,8,S), (U,V,Y) are the most confused pairs for the identification step. Each set of characters has the same codeword pair and they are usually identified incorrectly if identification algorithms are not implemented.

In Table 5-1, there are unexpected confused pairs. These pairs are (A,P), (5,2), (5,Z), (6,8), (M,H), (7,6), (U,D), (J,T) and (N,T). Codeword pairs for these characters are generated erroneously due to noise in the pixels of the character and generated codeword corresponds to another character in the lookup table.

## 5.2 TEST RESULTS FOR THE FPGA IMPLEMENTATION

In FPGA implementation, only character segmentation and character recognition parts are implemented. Plate regions obtained by the MATLAB implementation is used for testing. In MATLAB implementation, after the characters are segmented, connected component labeling and morphological opening are implemented to eliminate the adverse effect of black pixels which do not belong to the characters. However, since these steps increase the execution time, connected component labeling and morphological opening are not implemented in the FPGA.

In MATLAB implementation, for 116 images, character segmentation and character recognition algorithms are tested. In FPGA, 107 images out of these 116 images are used to test the FPGA implementation of the algorithm. 9 images cannot be used because sizes of the plate regions are larger than 8 Kbyte and these plate regions cannot be stored in the on-chip RAM. Test results for the FPGA implementations are given in Table 5-4.

**Table 5-4 Test Results for the FPGA Implementation**

Image Number	Original License Plate	FPGA Recognition	Explanation
0	38 NE 941	38NE941	
4	06 EEA 80	06EEA8-	1 error (0)
5	06 AAH 83	06AA1133	1 miss(H) + 1 error (8-3)
6	06 ZTM 61	06Z1--6	2 miss (M)(1) + 1 error (T-1)
8	06 NV 015	06NV015	
11	06 VCU 16	-6VC116	1 miss(U) + 1 error (0)
12	06 FG 382	06FG382	
13	06 V 6304	06V630-	1 error (4)
14	06 GJ 988	06G1863	4 error (J-1)(9-8)(8-6)(8-3)
15	06 GD 229	06B0229	2 error (G-B)(D-0)
16	06 BF 926	06-F92-	2 error (B)(6)
18	06 ZVP 28	06ZV028	1 error (P-0)
20	06 VTE 73	06VLE73	1 error (T-L)
21	06 AJC 01	0-ALC-1	3 error (6)(J-L)(0)
22	06 BS 913	06--819	4 error (B)(S)(9-8)(3-9)
23	06 E 9787	06E9667	2 error (7-6)(8-6)
24	06 UE 242	06UE242	
26	06 YAM 63	-----38-	7 error (0)(6)(Y)(A)(M)(6-3)(3-8) + 1 extra(-)
27	06 YUY 61	06YLY81	1 miss (U) + 1 error (6-8)
28	06 GH 039	06G1099	1 miss (H) + 1 error (3-9)

**Table 5-4 Test Results for the FPGA Implementation (Continued)**

<b>Image Number</b>	<b>Original License Plate</b>	<b>FPGA Recognition</b>	<b>Explanation</b>
30	06 FFA 51	06FFA51	
32	06 RED 88	06RE08-	2 error (D-0)(8)
33	06 Z 6339	--D-26-99	2 extra(-)(-) + 5 error (0-D) (6)(Z-2)(3) (3-9)
34	06 MVB 27	06MV827	1 error (B-8)
35	06 YDU 85	06Y4185	1 miss(U) + 1 error(D-4)
36	35 KFM 75	9--M-2	6 error (3-9)(5)(K)(F)(7)(5-2)
37	06 KNT 90	06-N190	2 error (K)(T-1)
38	06 RNM 90	06R--90	2 error (N)(M)
39	06 YRU 41		Image size is larger than 8 Kbyte. Therefore, it is not tested.
40	06 ZFS 48	06ZF308	2 error (S-3)(4-0)
41	32 AN 568	52AN268	2 error (3-5)(5-2)
43	06 ZTT 60	06Z1160	2 error (T-1)(T-1)
44	06 GM 575	06GM3-5	2 error (5-3)(7)
45	06 ZVS 41	06ZV341	1 error (S-3)
46	06 LM 649	---649	1 miss (L)+3 error (0)(6)(M)
48	06 JM 694	06JM694	
50	06 VB 035	06Y6092	4 error (V-Y)(B-6)(3-9)(5-2)
51	06 VTK 75	06VLK75	1 error (T-L)
52	06 ZVJ 64	--ZV-4	1 miss (J) + 3 error (0)(6)(6)
53	06 TNA 59	26NA-9	1 miss (T) + 2 error (2-0)(5)
54	06 EM 653	06EM65-	1 error (3)
55	06 DJ 294	06UJ294	1 error (D-U)
56	06 FKS 81	06FK384	2 error (S-3)(1-4)
57	06 MLD 18	06M1018	2 error (L-1)(D-0)
59	06 YZC 78	06YZC78	
60	06 H 0184	06D1-4	1 miss (H) + 2 error (D-0)(8)
63	06 FVY 64	06FVY60	1 error (4-0)
65	06 SM 805	06SM605	1 error (8-6)
66	06 KTY 39	06KLY393	1 error (T-1) + 1 extra(3)
67	06 AD 406	06AU006	2 error (D-U)(4-0)
68	06 YUF 55	06YUF55	
69	06 YBT 50	06Y6150	2 error (B-6)(T-1)
71	06 ZSS 91	10B23391	1 extra(1)+4 error (6-B)(Z-2) (S-3)(S-3)



**Table 5-4 Test Results for the FPGA Implementation (Continued)**

<b>Image Number</b>	<b>Original License Plate</b>	<b>FPGA Recognition</b>	<b>Explanation</b>
73	06 VVL 74	06A0-J74	1 miss (V) + 2 error (V-A)(L-J)
74	06 KJS 97	06KJ397	1 error (S-3)
76	06 BM 277	06BM277	
77	06 GV 261	06GY261	1 error (V-Y)
78	06 ZVR 08	06ZVR08	
79	16 NB 505	-6N6202	4 error (1)(B-6)(5-2)(5-2)
80	06 KYY 24	-0BKYY24-	1 error (6-B) + 2 extra(-)(-)
81	06 LCE 24		Image size is larger than 8 Kbyte. Therefore, it is not tested
82	06 TAV 30	06IAV30	1 error (T-I)
86	06 YNZ 47	46YN247	2 error (0-4)(Z-2)
87	06 FM 787		Image size is larger than 8 Kbyte. Therefore, it is not tested
93	06 YML 39	06YM139	1 error (L-1)
94	06 EAE 87	46EAE87	1 error (0-4)
95	06 EB 195	06E8193	2 error (B-8)(5-3)
97	06 HLD 71	06I11071	1 miss (H) + 2 error (L-1)(D-0)
98	06 ZKM 55		Image size is larger than 8 Kbyte. Therefore, it is not tested
99	06 VTF 54	06VLF54	1 error (T-L)
101	06 KUH 65	06K111165	2 miss (U)(H)
103	06 FJ 557	06F11357	1 miss (J) + 1 error (5-3)
104	06 YTE 47	-0BY1E07	1 extra (-) + 3 error (6-B)(T-1)(4-0)
105	06 DJ 294	06DJ294	
109	06 AB 664		Image size is larger than 8 Kbyte. Therefore, it is not tested
110	06 MML 54	06MM154	1 error (L-1)
112	06 GJ 015	06G11015	1 miss (J)
113	06 AB 789	06A8068	4 error (B-8)(7-0)(8-6)(9-8)
114	06 EHR 26		Image size is larger than 8 Kbyte. Therefore, it is not tested
118	06 FPL 06	06F0106	2 error (P-0)(L-1)

**Table 5-4 Test Results for the FPGA Implementation (Continued)**

<b>Image Number</b>	<b>Original License Plate</b>	<b>FPGA Recognition</b>	<b>Explanation</b>
120	06 RVR 44	46RVR44	1 error (0-4)
121	06 YMH 60	06 YM1160	1 miss (H)
122	06 TNA 59	10BLNA59	1 extra (1) + 2 error (6-B)(T-L)
125	06 LB 509	06I6249	4 error (L-I)(B-6)(5-2)(0-4)
126	06 TKL 06	06IK106	2 error (T-I)(L-1)
129	06 BN 824	-6BN824	1 error (0)
130	06 KKY 99	06KKY99	
133	06 GA 049	-----	1 extra (-) + 2 miss (0)(6)+ 5 error (G)(A)(0)(4)(9)
136	06 YZC 78	06YZC78	
137	06 MDB 52	06M065	2 error (D-0)(B-6) + 1 miss (2)
138	06 ZHT 89	06Z11189	1 miss (H) + 1 error (T-1)
139	06 BB 162		Image size is larger than 8 Kbyte. Therefore, it is not tested
140	06 ZZE 84	06Z2--	2 miss (8)(4) + 2 error (Z-2) (E)
141	06 LEE 85	06EE65	1 miss (L) + 1 error (8-6)
142	06 DM 529	06DM32-	2 error (5-3)(9)
143	06 KHZ 04		Image size is larger than 8 Kbyte. Therefore, it is not tested
145	06 YMH 26	06YM112	2 miss (H)(6)
147	06 VVR 58	-0BVVR58	1 extra (-) + 1 error (6-B)
148	06 KFY 42	06KFY72	1 error (4-7)
149	06 GA 965	06GA965	
150	06 GD 009	06B004-	4 error (G-B)(D-0)(0-4)(9)
151	06 GH 264	06B11264	1 error (G-B) + 1 miss (H)
152	06 VKC 70	06VKC70	
154	06 GR 895	06B-F895	2 error (G-B)(R) + 1 extra (F)
155	06 FM 969	06FM969	
158	34 VD 4751	94Y00-51	5 error (3-9)(V-Y)(D-0)(4-0)(7)
159	06 RZJ 36	46RZJ96	2 error (0-4)(3-9)
160	06 RHS 42	06RH342	1 error (S-3)
163	06 FU 644	06FU644	

**Table 5-4 Test Results for the FPGA Implementation (Continued)**

<b>Image Number</b>	<b>Original License Plate</b>	<b>FPGA Recognition</b>	<b>Explanation</b>
164	06 BL 040	06-1-040	1 error (B) + 1 miss (L)
165	06 TSZ 03	06I3203	3 error (T-I)(S-3)(Z-2)
167	06 EKN 80	06EKN80	
169	06 MHP 10	06M1101-	1 miss (H) + 2 error (P-0)(0)
170	06 MBH 36		Image size is larger than 8 Kbyte. Therefore, it is not tested
172	06 ZFS 22	06ZF322	1 error (S-3)
174	06 ZVJ 64	-0BZVJ64	1 extra (-) + 1 error (6-B)

Character recognition and segmentation rates for the 107 images which are tested in FPGA are given in Table 5-5.

**Table 5-5 Character Recognition Rate for the FPGA Implementation**

<b>Character</b>	<b>Number of Occurrences</b>	<b>Number of Extracted Characters</b>	<b>Number of Recognized Characters</b>	<b>Extraction Rate (%)</b>	<b>Recognition Rate (%)</b>
0	132	131	113	99.24	86.26
1	20	19	17	95.00	89.47
2	23	22	22	95.65	100.00
3	20	20	8	100.00	40.00
4	33	32	24	96.97	75.00
5	34	34	22	100.00	64.71
6	130	128	113	98.46	88.28
7	21	21	16	100.00	76.19
8	29	28	20	96.55	71.43
9	33	33	27	100.00	81.82
A	15	15	13	100.00	86.67
B	13	13	2	100.00	15.38
C	5	5	5	100.00	100.00
D	12	12	2	100.00	16.67
E	16	16	15	100.00	93.75
F	16	16	15	100.00	93.75
G	12	12	7	100.00	58.33
H	11	1	1	9.09	100.00
J	11	8	6	72.73	75.00

**Table 5-5 Character Recognition Rate for the FPGA Implementation (Continued)**

Character	Number of Occurrences	Number of Extracted Characters	Number of Recognized Characters	Extraction Rate (%)	Recognition Rate (%)
K	14	14	12	100.00	85.71
L	11	8	0	72.73	0.00
M	21	20	17	95.24	85.00
N	11	11	10	100.00	90.91
P	3	3	0	100.00	0.00
R	9	9	8	100.00	88.89
S	11	11	1	100.00	9.09
T	16	15	0	93.75	0.00
U	7	3	3	42.86	100.00
V	23	22	18	95.65	81.82
Y	19	19	18	100.00	94.74
Z	19	19	14	100.00	73.68
<b>Total</b>	<b>750</b>	<b>720</b>	<b>549</b>	<b>96.00</b>	<b>76.25</b>

For the FPGA implementation, it is concluded from Table 5-4 and Table 5-5 that,

(30 characters are missed during character segmentation) + (171 characters are decided wrongly or cannot be decided) = totally 201 characters are not identified correctly. For the recognition, 116 plates are worked with. 9 plates cannot be used because of being larger than 8 Kbyte in size. Therefore, for the FPGA tests, (115 - 9) plates x 7 characters / plate + 1 plate x 8 characters / plate = 750 characters are used. For the recognition of the segmented characters, 720 characters are used and 549 characters are recognized. 171 characters are identified wrongly or cannot be identified. Therefore, recognition percentage over the extracted characters becomes  $(549/720) \times 100 = 76.25\%$ . Recognition rate of the whole images used for the FPGA implementation is  $((549 \text{ identified characters}) / (106 \text{ plates} \times 7 \text{ characters / plate} + 1 \text{ plate} \times 8 \text{ characters / plate} = 750 \text{ characters}) \times 100 = 73.2\%$ .

Since codewords are not unique and one codeword can represent more than one character, characters may not be recognized accurately and recognition rate decreases. Expected confused characters which have the same codeword pairs are given in Table 5-6.

**Table 5-6 Characters Represented by Same Codeword Pair**

y-axis codeword	x-axis codeword	Confused Characters
10000	10000	1, I, L, T
10000	23200	2, Z

**Table 5-6 Characters Represented by Same Codeword Pair (Continued)**

<b>y-axis codeword</b>	<b>x-axis codeword</b>	<b>Confused Characters</b>
10000	32000	2, Z
12100	23200	
12100	12100	0, 4, 7, D, O, P
12100	12310	0, D
12100	12320	
12100	21000	
12120	12300	
12121	13100	6, 8
12121	13120	8, B
12121	13200	6, B
12121	23100	3,8,9
12121	23200	3, 5, 8,S
21000	10000	U, V, Y
21000	12100	

‘0’, ‘4’, ‘7’, ‘D’, ‘O’ and ‘P’ characters share the same codeword which is “121000-121000”. Therefore, these characters may conflict. As given in Table 5-5, recognition rate of ‘D’ and ‘P’ is low, as a result of the confliction.

Segmentation of ‘H’ and ‘U’ letters are also considerably low in FPGA implementation due to the notch detection method as explained in Section 5.1.

‘B’ letter is confused with ‘6’ and ‘8’ and this confusion makes the recognition rate of ‘B’ low. To distinguish ‘B’ from ‘6’ and ‘B’ from ‘8’, letter and number identification algorithm is applied over the whole plate characters. Identification of ‘6’ and ‘B’ algorithm can be applied as a future work. In Turkish type license plates, first two and last two characters must be numbers and third character must be a letter. Therefore, letter and number identification algorithm is implemented for these characters. However, if there is an extra character at the beginning, the restriction of being a letter in the third character may result in error. Fourth and fifth characters may be a number or a letter. If the fifth character is a letter, fourth character should also be a letter. Therefore, if there is an ‘8’ and ‘B’ or ‘6’ and ‘B’ confliction at third character, it can be identified. If these characters are in the place of fourth character and if the fifth character is a letter, they can be identified otherwise, it is hard to distinguish.

‘2’ and ‘Z’ characters are confused. However, this confusion is more likely to be solved because one of the characters is a letter while the other one is a number. By applying identification of letter and number algorithm they can be distinguished for the first two, last

two and for the third character in the license plate. If the fifth character is a letter, confliction is resolved for the fourth character. Moreover, identification of ‘2’ and ‘Z’ algorithm may be implemented in the FPGA as a future work.

‘3’, ‘5’, ‘8’ and ‘S’ characters are confused. If the character whose recognition is in progress is identified as a number with the identification of letter and number algorithm, identification of ‘3’, ‘5’, and ‘8’ algorithm is implemented. If ‘S’ character is the third character or if it is the fourth character when the fifth character is a letter, it can be distinguished from ‘3’, ‘5’ and ‘8’. Otherwise, ‘S’ character is always recognized as one of the ‘3’, ‘5’ and ‘8’ characters.

‘1’, ‘I’, ‘L’ and ‘T’ characters have the same codeword pair which is “10000-10000”. Since they share the same codeword pair and codeword pairs are prone to errors, recognition rates for these characters are low.

Recognition rate for the ‘J’ characters is not very high because tail of the ‘J’ character is trimmed when segmenting the character. As a result of the trimming, codeword pair for the ‘J’ is generated erroneously which is different than the value given in the lookup table. Due to errors in the codeword pair, recognition rate of the ‘J’ letter is not very high.

According to Table 5-4, there are unexpected confused pairs. These pairs are (G,6), (8,6), (7,6), (5,2), (2,0), (D,U), (1,4), (V,A) and (V,0). Codeword pairs for these characters are generated erroneously due to noise in the pixels of the character and generated codeword corresponds to another character in the lookup table. Unexpected confused pairs are also results from the lack of connected component labeling and morphological opening steps which are implemented in MATLAB but not implemented in the FPGA.

Performance of the FPGA implementation for 107 test images is given in Table 5-7.

**Table 5-7 Performance of License Plate Recognition**

<b>Recognition Result</b>	<b>Number of Plates</b>	<b>Performance Rate (%)</b>
All Characters Recognized Accurately	18	16.82
Recognized with only 1 character error	27	25.23
Recognized with only 1 missed character	2	1.87
Recognized with only 2 character error	22	20.56
Recognized with only 2 missed character	2	1.87
Recognized with 1 missed character and 1 character error	10	9.35
Recognized with 2 missed character and 1 character error	1	0.93

**Table 5-7 Performance of License Plate Recognition (Continued)**

<b>Recognition Result</b>	<b>Number of Plates</b>	<b>Performance Rate (%)</b>
Recognized with 1 missed character and 2 character error	6	5.61
Recognized with 2 missed character and 2 character error	1	0.93
Recognized with 3 character error	3	2.80
Recognized with 3 character error and 1 missed character	2	1.87
Recognized with more than 3 character error	13	12.15
<b>Total number of Plates</b>	<b>107</b>	<b>100</b>

As it can be seen from the Table 5-4 and Table 5-7, for only 16.82 % of the test images, all characters are recognized correctly. This low ratio results from the character segmentation and character recognition errors. In order to segment the character, mean value of the plate region is calculated and notch detection algorithm is implemented by using the mean value. For some plates, there are unwanted pixels over the image which affects the mean value adversely. Due to mean value variations, characters are segmented erroneously which reduces the performance of the system. Moreover, due to unwanted pixels which do not belong to characters, codeword of the characters are generated erroneously and erroneous codewords also reduce recognition performance.

In order to compare the test results of the MATLAB and FPGA implementations, connected component labeling and morphological opening steps of the character segmentation part are removed and algorithm used in the MATLAB is made same with the FPGA algorithm. Revised MATLAB algorithm is tested with 107 images used in the FPGA tests. According to the test, exactly same results are obtained for all images.

### **5.3 COMPARISON WITH THE STATE OF THE ART**

The result of the proposed system is compared with the previous studies. For the detection of the plate region, Üçüncü [26] implements edge detection operators, morphological operations and run-length smoothing algorithm. For the same image dataset, better plate region localization results are obtained by the Üçüncü. For the 99 % of the images, license plate is located correctly whereas plate region detection rate for the proposed algorithm is 67.43%. The reason for obtaining lower result is that Gabor response of the image suffers from the large distances between the letter-to-number and number-to-letter which yields disconnected parts for the plate region. Moreover, morphology implemented Gabor response can make connections with the outer pixels which do not belong the plate region. To extract the candidate region, Gabor response is filtered with predefined plate length and

width parameters. Since, there may be connections with outer pixels, plate region cannot be extracted successfully.

In the Üçüncü's method, for the character segmentation X-Y-tree decomposition algorithm is used. After the characters are segmented, feature-based character recognition algorithm is implemented for the recognition. In the Üçüncü's method with the same database, character segmentation and character recognition rates are 97.49% and 95.5%, respectively. With the X-Y-tree decomposition algorithm, better results are obtained than the proposed method because it is an iterative approach with analyses both horizontal and vertical projection and uses the adaptive thresholding. In the proposed method, notch detection algorithms which make use of the only vertical projection by considering the FPGA implementation. Implementation of iterative algorithms increases the execution time and reduces the performance of the system.

In the proposed system, FPGA implementation of the character recognition gives 73.2% success rate. This result is lower than the result obtained by the Üçüncü. For the character recognition, nearly same feature-based character recognition algorithm is implemented. However, in Üçüncü's method, some preprocessing techniques are used to obtain better results such as adaptive thresholding, histogram equalization, finding the upper and lower limits of the character which are not implemented in the FPGA. Therefore, better success rate is obtained for the Üçüncü's method.

In the method proposed by the Hung and Hsieh [14], license plate detection region is set using the probability distribution of the license plate between the two barking lights in the captured image. This method eliminates environmental interference during the license plate detection and improves the rate of accuracy of the license plate detection and recognition. For the test of the Hung and Hsieh's method, 257 vehicle images are used. Images are taken by the video camera installed under the windshield of the vehicle. Images are captured in various environmental conditions such as clear day, rainy day, daytime and evening. The size of the captured images is 320x240x24-bit color picture. For the license plate localization and character recognition better results are obtained by the Hung and Hsieh's method but character segmentation method proposed in this thesis has nearly same success rate.

Kahraman et al. [20] also uses Gabor approach for the license plate detection. In the system, nonlinear vector quantization is used to eliminate false alarms and segment the license plate characters to its exact boundary. In the system, 12 Gabor filters with three scales and four directions are used whereas a single Gabor filter is used in the proposed system. Moreover, in Kahraman's method eight-connected blob coloring is used whereas in the proposed algorithm, connected component labeling with four neighboring is used. Since twelve Gabor filters in the Kahraman's method give better response, better success rate is obtained for the license plate detection. For the character segmentation, since vector quantization method minimizes the quantization error, better segmentation rate is obtained. For the test of the system, 300 images were used. 167 images were taken day and the



remaining were taken night conditions with various sizes, styles and forms of the license plate. The resolution of the images ranged from 512x384 to 768x576 [20].

Performance comparison of the previous studies with the proposed system is given in Table 5-8.

**Table 5-8 Performance Comparison of the Results**

<b>Steps</b>	<b>Recognition Rate (for the Hung and Hsieh's method)</b>	<b>Recognition Rate (for the Kahraman et al.'s method)</b>	<b>Recognition Rate (for the proposed method's MATLAB implementation)</b>
License Plate Localization	95.33%	98%	67.43%
Character Segmentation	91.05%	94.2%	91.64%
Character Recognition	88.71%	NA	78.93%

## 5.4 RESOURCE UTILIZATION

In FPGA designs, logic and memory utilizations are the important parameters about which the designer should be careful. A major consideration when programming FPGA is the amount of the circuitry code used on the chip. It is sometimes useful to know how much space a specific function will use when translated to look up tables and flip flops.

In Table 5-9, resource utilization of the License Plate Recognition System is given. In the system implemented in the FPGA only 68% of the total logic elements and 60% of the FPGA memory bits are used.

**Table 5-9 Resource Utilization of the License Plate Recognition System**

<b>RESOURCE</b>	<b>USAGE</b>
<b>Total logic elements</b>	7,036 / 10,320 ( 68 % )
Combinational with no register	2851
Register only	947
Combinational with a register	3238
<b>Logic element usage by number of LUT inputs</b>	
4 input functions	3424
3 input functions	1749
<=2 input functions	916
Register only	947

**Table 5-9 Resource Utilization of the License Plate Recognition System (Continued)**

<b>RESOURCE</b>	<b>USAGE</b>
<b>Logic elements by mode</b>	
normal mode	5526
arithmetic mode	563
<b>Total registers*</b>	4303/10732(40%)
Dedicated logic registers	4185/10320(41%)
I/O registers	118/412(29%)
<b>Total LABs: partially or completely used</b>	529/645 (82%)
User inserted logic elements	0
Virtual pins	0
<b>I/O pins</b>	75/92 (82%)
Clock pins	4/3 (133%)
Dedicated Input Pins	3/9(33%)
<b>Global signals</b>	10
<b>M9Ks</b>	40/46 (87%)
<b>Total memory bits</b>	256,384 / 423,936 ( 60 % )
<b>Total block memory implementation bits</b>	368,640 / 423,936 ( 87 % )
<b>Embedded Multiplier 9-bit elements</b>	4 / 46 ( 9 % )
<b>PLLs</b>	1 / 2 ( 50 % )
<b>Global clocks</b>	10 / 10 ( 100 % )
<b>JTAGs</b>	1 / 1 ( 100 % )
<b>CRC blocks</b>	0 / 1 ( 0 % )
<b>ASMI blocks</b>	0 / 1 ( 0 % )
<b>Impedance control blocks</b>	0 / 4 ( 0 % )
<b>Average interconnect usage (total/H/V)</b>	33% / 33% / 33%
<b>Peak interconnect usage (total/H/V)</b>	46% / 45% / 49%
<b>Maximum fan-out node</b>	pll1:inst altpll:altpll_compo nent pll1_altpll:auto_genera ted wire_pll1_clk[0]~clkctrl
<b>Maximum fan-out</b>	4039
<b>Highest non-global fan-out signal</b>	niosII:inst6 niosII_nios_cpu :nios_cpu W_stall~1
<b>Highest non-global fan-out</b>	606
<b>Total fan-out</b>	39727
<b>Average fan-out</b>	3.48

\* Register count does not include registers inside RAM blocks or DSP blocks.

## 5.5 EXECUTION TIMES

FPGA implementation of the License Plate Recognition System operates at 75 MHz clock frequency. FPGA logic blocks are embedded with the Altera system integration tool and processor controls the operations of the hardware. It is expected that execution time of the hardware side is always same. However, since processor controls the logic blocks, duration of the operations cannot be same. Execution times for processor side depend on the size and content of the input image. Number of pixels used for the character segmentation and character recognition affects the computation time. Increasing the size of the input image increases the execution times. Moreover, black pixels at the top and bottom of the plate region require trimming. Trimming operation over the all plate region increases the computation time. For some confused characters, identification algorithms are implemented which also increase the duration of the operation. Execution times are measured in worst case where trimming take place and the size of the plate region is large. Measured and calculated execution times of the system for a single plate are presented in Table 5-10.

**Table 5-10 Execution Times of the LPR System**

<b>Operation</b>	<b>Execution Time</b>
Reading from and Writing to RAM	56.476 ms
Up and Down Side Trimming	103.170 ms
Vertical Projection without DC value	12.030 ms
Notch detection	1.133 ms
Character Segmentation	10.260 ms
Up and Down Side Trimming for the Character	37.013 ms
Codeword Generation	29.362 ms
Codeword Matching	10.217 ms
Identification of Character	11.396 ms
<b>Total Execution Time</b>	<b>271.239</b>

From the Table 5-10, it is seen that “Up and Down Side Trimming” part has the longest execution time. In this part, black pixels which do not belong to characters are trimmed over the all plate region. When the number of pixels in the plate region is large, since all pixels values are used for trimming, computation time increases. Moreover, if there are no confused characters in the plate region, identification of characters part is not processed and it decreases the time of the computation. In the Table 5-10, worst case scenario is given over the used dataset. Maximum latency of the system becomes almost 272 ms. In 272 ms, a car can go 5.3 m with average 70 km/h speed. This distance is highly convenient for the implementation of the license plate recognition system because images used in the dataset are taken from 4-7 meters away from the vehicle.



## CHAPTER 6

### CONCLUSIONS AND FUTURE WORK

#### 6.1 CONCLUSIONS

In this thesis, license plate detection and recognition is studied and verified in MATLAB and then, recognition part is implemented in the FPGA. Implemented algorithm is tested with 175 images which are taken with different lighting conditions. The algorithm developed for the FPGA is a combination of previously proposed algorithms. In order test the FPGA implementation, 116 plate regions are used which are obtained from the MATLAB implementation.

In the first step of the proposed algorithm, location of the plate region in the input image is found by utilizing Gabor transform and morphological operations. For the detection of the plate region, image is converted into HSV color representation and value component of the image is used to decrease the complexity and to increase the performance by eliminating illumination effects over the image. Contrast enhancement is applied to obtain better results from Gabor transform. Gabor response of the image gives the black-to-white pixel transition which exist considerably in plate region. Gabor response of the images is merged by using morphological operation and uniform candidate plate regions are extracted. By observing the density and the size of the candidate region, exact plate region is determined.

In the character segmentation step of the proposed algorithm, vertical projection of the extracted plate region is calculated. Each character in the plate region represents a notch in the vertical projection. By using the distances between notches, characters are separated.

As a final step, each segmented character is recognized by using template matching. For each character, codeword is calculated which represents the character with a sign. By comparing calculated codeword with the template codeword stored in the database, character is recognized.

For same cases, more than one character is recognized for the candidate character and this situation results in confusion. To solve this problem, identification algorithms are implemented for the confused characters.

The algorithm of the character segmentation and character recognition parts are implemented on Altera Cyclone IV E FPGA. For the implementation of the algorithm, network-on-chip structure is constructed with the Altera system integration tool, Qsys, and compact system block is obtained. In the Qsys system, Nios II soft processor which is embedded into the FPGA is utilized. The workload of the algorithms is divided between

FPGA logic and processor according to the complexity of the algorithm. FPGA and Nios II soft processor is running with 75 MHz clock. Operating clock frequency can be increased but it doesn't affect the execution times. The main item which determines the execution time performance is the read and write cycles of the external memory. Since, used SDRAM has its own clock for its transmission bus; increase in the operating FPGA frequency will not affect the performance of the implemented algorithm.

License plate detection rate is 67.43% for the MATLAB implementation of the image database. Wrong detections occur if the plate region is dirty or there is a screw on the region. If characters are too small or separated in large distances or if the characters make connection with the upper or lower side of the plate holder, it is not possible to locate the license plate with this algorithm. If the separation between the number-to-letter and letter-to-number is high, morphology implemented Gabor response consists of disconnected parts and these parts are not considered as a plate region. These reasons result in low extraction ratio.

Character segmentation rate is higher than 90% for the image dataset. Noise over the image, false pixel values between characters and characters whose vertical projection consist of low values such as L, T etc. decrease the segmentation rate. Since these reasons affect the vertical projection and character segmentation is achieved according to the vertical projection value of the plate region, wrong segmentations may occur.

Character recognition rate is around 76%. Recognition ratio is low because the character size is so small that the noise factors become effective on the recognition step. If there is an error due to noise or false pixel values, codewords which represent characters are generated erroneously and this affects the matching result. Also, identification algorithms for confused characters do not cover the all confused characters and wrong recognition may occur. Usage of the different fonts with different sizes, thickness in the plate region, small picture stickers exist on the plate, plates that are not holded straight and shadow on the plate region are the other factors reducing the recognition ratio. Moreover, in template matching stage, codeword of the candidate character is compared with the templates one by one and if the total index values are same; this template is considered as a recognized character. This situation also decreases the recognition rate. If the template matching of the character is accepted with some similarity measure, recognition rate of the candidate character increases.

For the FPGA implementation, only 68% of the FPGA logic resources are used. Since the development board used for the FPGA implementation is a low cost and low power board, it decreases the cost of the system.

## **6.2 FUTURE WORK**

Some extra processes can be applied as a future work to improve the recognition and extraction ratio. One of the major problem of the plate region detection is the disconnected parts of the morphology implemented Gabor response due to large distances between number-to-letter and letter-to number. As a future work, after finding the Gabor response, these connected regions can be analyzed whether they contains characters or not. If there is a character on the region, enlargement of the disconnected parts can be applied. If the disconnected parts are separated in the range of predefined distance, these disconnected parts can be merged.

Plate regions used in this thesis is parallel to the ground and Gabor filters used in the proposed algorithm is orientation dependent. To provide orientation invariant license plate detection and recognition system, skew correction algorithm may be applied which is not the subject of this thesis.

For the character segmentation step, if not all of the characters are segmented successfully, the place of the missed characters can be guessed and characters in this region are tried to be found for the recognition step as a future work.

For the confused characters, the number of identification algorithms can be increased.

In order the implement the plate region detection step in the FPGA, larger FPGA can be used as a future work. Since Gabor filtering is achieved over the 640x480 resolution images, it requires larger FPGA. Moreover, if the used development board contains peripherals such as seven segment display or LCD display, the proposed system becomes standalone system which shows the results on the screen. With standalone system, license plate detection and recognition system can be used independently to improve the traffic control, park automatic and law enforcements etc.





## REFERENCES

1. Iwanowski, Marcin. "Automatic car number plate detection using morphological image processing." *Przeład Elektrotechniczny* 81 (2005): 58-61.
2. Daugman, John G. "Two-dimensional spectral analysis of cortical receptive field profiles." *Vision research* 20, no. 10 (1980): 847-856.
3. Daugman, John G. "Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters." *Optical Society of America, Journal, A: Optics and Image Science* 2, no. 7 (1985): 1160-1169.
4. Lee, Tai Sing. "Image representation using 2D Gabor wavelets." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 18, no. 10 (1996): 959-971.
5. Lades, Martin, Jan C. Vorbruggen, Joachim Buhmann, Jörg Lange, Christoph von der Malsburg, Rolf P. Wurtz, and Wolfgang Konen. "Distortion invariant object recognition in the dynamic link architecture." *Computers, IEEE Transactions on* 42, no. 3 (1993): 300-311.
6. Otsu, Nobuyuki. "A threshold selection method from gray-level histograms." *Automatica* 11, no. 285-296 (1975): 23-27.
7. Datasheet of Altera DB\_START\_4CE10-Cyclone IV E Development Board  
[http://www.ebv.com/download/?file=fileadmin/HOME/1\\_Products/EBV/DB\\_Start\\_4CE10\\_Dev.Board/DB\\_START\\_4CE10\\_v1-0.pdf&no\\_cache=1](http://www.ebv.com/download/?file=fileadmin/HOME/1_Products/EBV/DB_Start_4CE10_Dev.Board/DB_START_4CE10_v1-0.pdf&no_cache=1) last accessed date: April 23, 2013
8. Avalon Interface Specification.  
[http://www.altera.com/literature/manual/mnl\\_avalon\\_spec.pdf](http://www.altera.com/literature/manual/mnl_avalon_spec.pdf) last accessed date: July 13, 2013
9. Nios II Processor Reference Handbook  
[www.altera.com/literature/hb/nios2/n2cpu\\_nii5v1.pdf](http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf) last accessed date: May 14, 2013.
10. Datasheet of LTC2801IDE <http://cds.linear.com/docs/en/datasheet/2801234fe.pdf> last accessed data : June 13,2013
11. Creating a System with Qsys [http://www.altera.com/literature/hb/qts/qsys\\_intro.pdf](http://www.altera.com/literature/hb/qts/qsys_intro.pdf) last accessed date : June 14, 2013.
12. Babu, C. Nelson Kennedy, and Krishnan Nallaperumal. "A license plate localization using morphology and recognition." In *India Conference, 2008. INDICON 2008. Annual IEEE*, vol. 1, pp. 34-39. IEEE, 2008.

13. Ozbay, Serkan, and Ergun Ercelebi. "Automatic vehicle identification by plate recognition." *World Academy of Science, Engineering and Technology* 9, no. 41 (2005): 222-225.
14. Hung, Kuo-Ming, and Ching-Tang Hsieh. "A Real-Time Mobile Vehicle License Plate Detection and Recognition." *Tamkang Journal of Science and Engineering* 13, no. 4 (2010): 433-442.
15. Duman, S., R. Oktem, and A. Enis Cetin. "Alternative feature extraction approaches for car plate recognition." In *Signal Processing and Communications Applications Conference, 2005. Proceedings of the IEEE 13th*, pp. 324-327. IEEE, 2005.
16. Comelli, Paolo, Paolo Ferragina, Mario Notturmo Granieri, and Flavio Stabile. "Optical recognition of motor vehicle license plates." *Vehicular Technology, IEEE Transactions on* 44, no. 4 (1995): 790-799.
17. Wang, Shen-Zheng, and Hsi-Jian Lee. "Detection and recognition of license plate characters with different appearances." In *Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE*, vol. 2, pp. 979-984. IEEE, 2003.
18. Kim, K. M., B. J. Lee, K. Lyou, and G. T. Park. "The automatic recognition of the plate of vehicle using the correlation coefficient and Hough transform." *Journal of Control, Automation and System Engineering* 3, no. 5 (1997): 511-519.
19. Gonzalez, R., and R. Woods. "Digital Image Processing 2nd Edition Prentice Hall." Upper Saddle River, NJ (2002).
20. Kahraman, Fatih, Binnur Kurt, and Muhittin Gökmen. "License plate character segmentation based on the gabor transform and vector quantization." In *Computer and Information Sciences-ISCIS 2003*, pp. 381-388. Springer Berlin Heidelberg, 2003.
21. Wu, B-F., S-P. Lin, and C-C. Chiu. "Extracting characters from real vehicle licence plates out-of-doors." *IET Computer Vision* 1, no. 1 (2007): 2-10.
22. Iwanowski, Marcin. "An application of mathematical morphology to filtering and feature extraction for pattern recognition." *Przeegl d Elektrotechniczny* 80, no. 4 (2004).
23. Fahmy, Maged MM. "Automatic number-plate recognition: neural network approach." In *Vehicle Navigation and Information Systems Conference, 1994. Proceedings.*, 1994, pp. 99-101. IEEE, 1994.
24. Lee, Eun Ryung, Pyeoung Kee Kim, and Hang Joon Kim. "Automatic recognition of a car license plate using color image processing." In *Image Processing, 1994. Proceedings. ICIP-94.*, IEEE International Conference, vol. 2, pp. 301-305. IEEE, 1994.

25. Kahraman, Fatih, B. Evrim Demiröz, Binnur Kurt, and Muhittin Gökmen. "Bakış Açısından Bağımsız Gürbüz Plaka Tanıma Sistemi." *Endüstri ve Otomasyon*, sayı 106, sayfa 28-32, Ocak 2006.
26. Üçüncü, Barış. "Computer Based Identification of Car License Plate", Msc Thesis, METU, (2000).
27. Castello, Paolo, Christopher Coelho, Enrico Del Ninno, Ennio Ottaviani, and Michele Zanini. "Traffic monitoring in motorways by real-time number plate recognition." In *Image Analysis and Processing, 1999. Proceedings. International Conference on*, pp. 1128-1131. IEEE, 1999.
28. Sirithinaphong, Thanongsak, and Kosin Chamnongthai. "The recognition of car license plate for automatic parking system." In *Signal Processing and Its Applications, 1999. ISSPA'99. Proceedings of the Fifth International Symposium on*, vol. 1, pp. 455-457. IEEE, 1999.
29. Automatic Vehicle Identification  
[http://www.federalapd.com/AutomaticVehicleIdentification\\_7053.aspx](http://www.federalapd.com/AutomaticVehicleIdentification_7053.aspx) last accessed date : August 2, 2013.
30. Barroso, J., E. L. Dagless, A. Rafael, and J. Bulas-Cruz. "Number plate reading using computer vision." In *Industrial Electronics, 1997. ISIE'97., Proceedings of the IEEE International Symposium on*, pp. 761-766. IEEE, 1997.
31. He, Ming G., Alan L. Harvey, and Paul Danelutti. "Car number plate detection with edge image improvement." In *Signal Processing and Its Applications, 1996. ISSPA 96., Fourth International Symposium on*, vol. 2, pp. 597-600. IEEE, 1996.
32. Kim, Kl Kim, K. I. Kim, J. B. Kim, and H. J. Kim. "Learning-based approach for license plate recognition." In *Neural Networks for Signal Processing X, 2000. Proceedings of the 2000 IEEE Signal Processing Society Workshop*, vol. 2, pp. 614-623. IEEE, 2000.
33. Wei, Wu, Yuzhi Li, Mingjun Wang, and Zhongxiang Huang. "Research on number-plate recognition based on neural networks." In *Neural Networks for Signal Processing XI, 2001. Proceedings of the 2001 IEEE Signal Processing Society Workshop*, pp. 529-538. IEEE, 2001.
34. Zimic, Nikolaj, Jelena Ficzkó, Miha Mraz, and Jernej Virant. "The fuzzy logic approach to the car number plate locating problem." In *Intelligent Information Systems, 1997. IIS'97. Proceedings*, pp. 227-230. IEEE, 1997.
35. Kim, Sang Kyoon, Dae Wook Kim, and Hang Joon Kim. "A recognition of vehicle license plate using a genetic algorithm based segmentation." In *Image Processing, 1996. Proceedings., International Conference on*, vol. 1, pp. 661-664. IEEE, 1996.



## APPENDIX A

### NIOS II PROCESSOR

The Nios II processor is a general-purpose RISC processor core with the following features [9]:

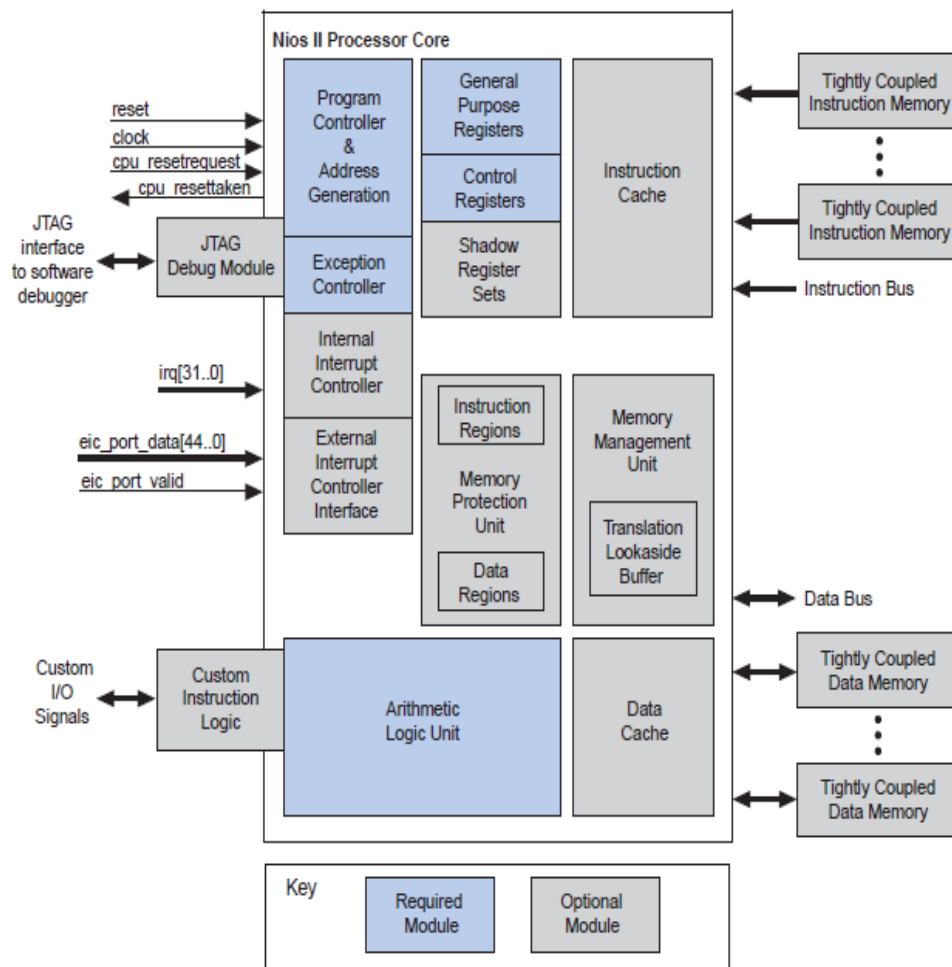
- Full 32-bit instruction set, data path, and address space
- 32 general-purpose registers
- Optional shadow register sets
- 32 interrupt sources
- External interrupt controller interface for more interrupt sources
- Single-instruction  $32 \times 32$  multiply and divide producing a 32-bit result
- Dedicated instructions for computing 64-bit and 128-bit products of multiplication
- Floating-point instructions for single-precision floating-point operations
- Single-instruction barrel shifter
- Access to a variety of on-chip peripherals, and interfaces to off-chip memories and peripherals
- Hardware-assisted debug module enabling processor start, stop, step, and trace under control of the Nios II software development tools
- Optional memory management unit (MMU) to support operating systems that require MMUs
- Optional memory protection unit (MPU)
- Software development environment based on the GNU C/C++ tool chain and the Nios II Software Build Tools (SBT) for Eclipse
- Integration with Altera's SignalTap® II Embedded Logic Analyzer, enabling real-time analysis of instructions and data along with other signals in the FPGA design
- Instruction set architecture (ISA) compatible across all Nios II processor systems
- Performance up to 250 DMIPS

The Nios II architecture defines the following functional units:

- Register file
- Arithmetic logic unit (ALU)
- Interface to custom instruction logic
- Exception controller
- Internal or external interrupt controller
- Instruction bus

- Data bus
- Memory management unit (MMU)
- Memory protection unit (MPU)
- Instruction and data cache memories
- Tightly-coupled memory interfaces for instructions and data
- JTAG debug module

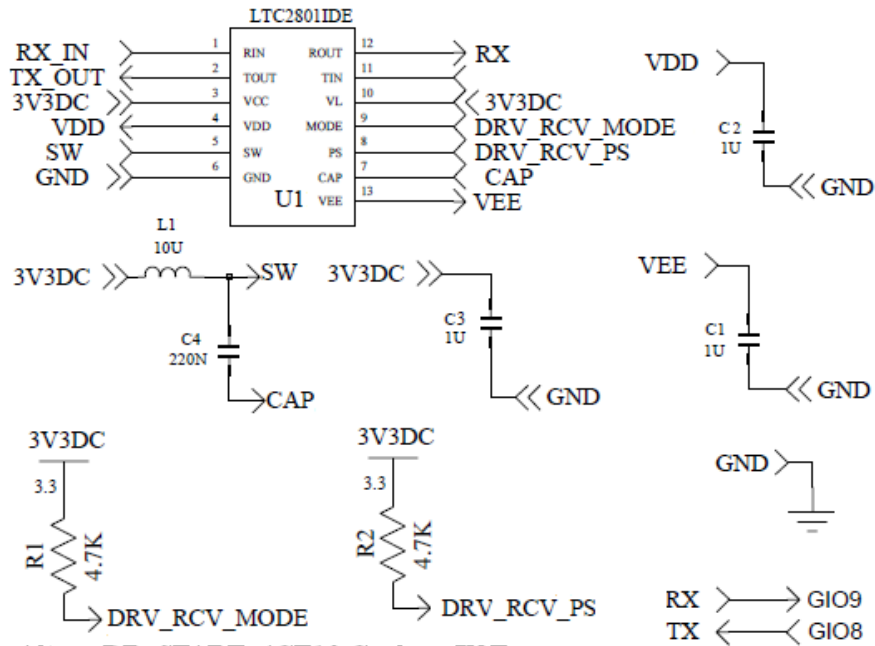
Block diagram of the Bios core is given in Figure A-1.



**Figure A-1 Nios II Processor Core Block Diagram**

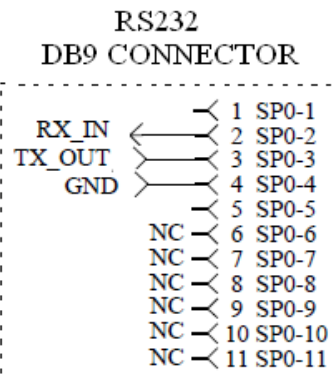
## APPENDIX B

### RS232 EXTENDER CIRCUIT



Altera DB\_START\_4CE10 Cyclone IV E  
Development Board  
User I/O Connector

Function	FPGA	Pin	Pin	FPGA	Function
GND		1	2		GND
GIO0	110	3	4	106	GIO1
GIO2	105	5	6	104	GIO3
GIO4	103	7	8	101	GIO5
GIO6	100	9	10	99	GIO7
GIN0	91	11	12	90	GIN1
GIN2	89	13	14	88	GIN3
GIN4	25	15	16		
GND		17	18		GND
GIO8	87	19	20	98	GIO9
GIO10	83	21	22	11	GIO11
GIO12	136	23	24	7	GIO13
GIO14	31	25	26	121	GIO15
GIO16	46	27	28	120	GIO17
GIO18	65	29	30		GND
GND		31	31		GND



\* Detailed information about LTC2801IDE device can be found in [10].





## APPENDIX C

### GUI INTERFACE OF THE LPDRS

GUI interface is developed to send the image via serial port. Interface is developed on Microsoft Visual C# 2010 Express. Recognition result of the LPDRS is displayed on the GUI, after the recognition process is completed.

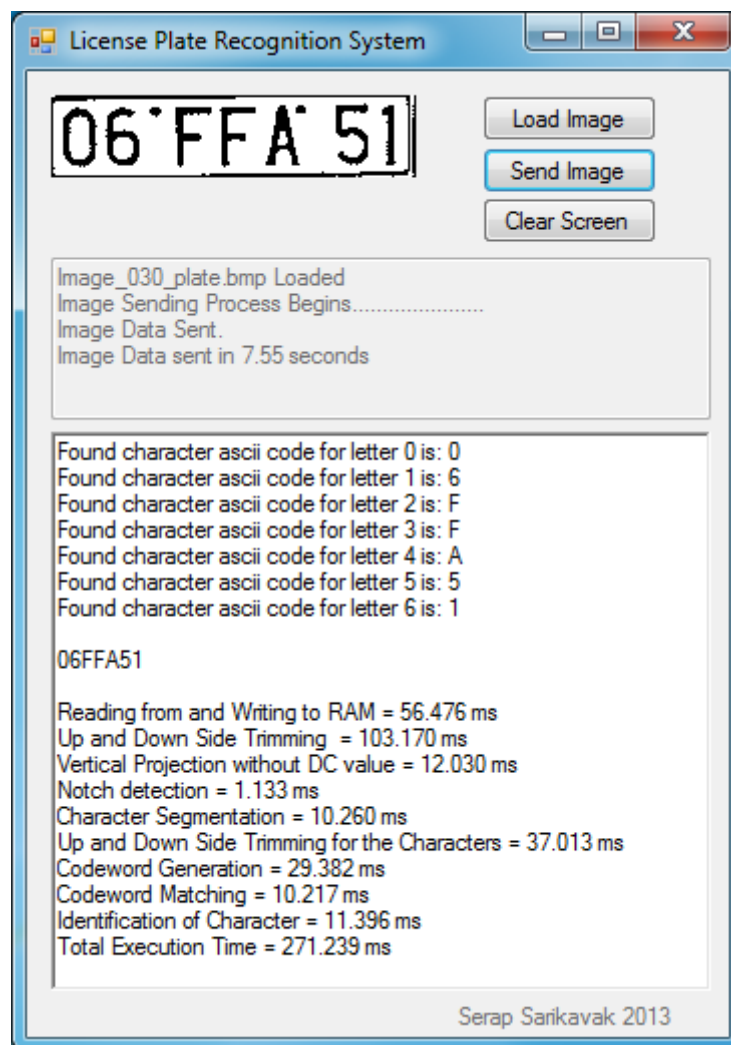


Figure C-1 GUI of LPDRS



## APPENDIX D

### DATA TABLES

This appendix contains data tables for the proposed license plate recognition algorithm.

Table D-1 shows the codewords and ASCII codes that are used for the character recognition process.

Table D-2 shows the plates that are used as test images.

**Table D-1 Lookup Table for the Characters**

<b>Character</b>	<b>y-axis codeword</b>	<b>x-axis codeword</b>	<b>ASCII code in decimal</b>
'0'	12100	12100	48
'0'	12100	21000	48
'0'	12100	12320	48
'0'	12120	12300	48
'0'	12100	12310	48
'1'	10000	10000	49
'2'	12100	23200	50
'2'	10000	23200	50
'2'	10000	32000	50
'2'	12100	20000	50
'2'	12121	23210	50
'2'	21210	21000	50
'3'	12121	23120	51
'3'	12100	23000	51
'3'	12121	23420	51
'3'	12121	23100	51
'3'	12121	23200	51
'4'	12100	12000	52
'4'	12100	12100	52
'4'	12321	12100	52

**Table D-1 Lookup Table for the Characters (Continued)**

<b>Character</b>	<b>y-axis codeword</b>	<b>x-axis codeword</b>	<b>ASCII code in decimal</b>
'5'	10000	12321	53
'5'	12121	23200	53
'6'	12121	13200	54
'6'	12100	13100	54
'6'	12121	13100	54
'6'	12100	13000	54
'6'	12121	24300	54
'6'	12100	13210	54
'6'	12121	13210	54
'6'	12100	13200	54
'7'	12100	12100	55
'8'	12100	13120	56
'8'	12121	21312	56
'8'	12121	21320	56
'8'	12121	21310	56
'8'	12121	13000	56
'8'	12121	24200	56
'8'	12121	13100	56
'8'	12121	23200	56
'8'	12121	13120	56
'8'	12121	23100	56
'9'	12121	23100	57
'9'	12100	23100	57
'A'	12120	12100	65
'A'	21212	12121	65
'B'	12121	13120	66
'B'	12121	13200	66
'C'	12121	12000	67
'C'	12121	20000	67
'D'	12100	12100	68
'D'	12100	21000	68
'D'	12100	12320	68
'D'	12120	12300	68
'D'	12100	12310	68
'E'	10000	13200	69

**Table D-1 Lookup Table for the Characters (Continued)**

<b>Character</b>	<b>y-axis codeword</b>	<b>x-axis codeword</b>	<b>ASCII code in decimal</b>
'E'	10000	13000	69
'E'	12000	13212	69
'F'	10000	12100	70
'F'	10000	12000	70
'G'	12121	12320	71
'H'	21200	10000	72
'I'	10000	10000	73
'J'	12100	10000	74
'K'	21200	12000	75
'L'	10000	10000	76
'M'	23400	10000	77
'M'	21320	10000	77
'N'	23200	10000	78
'N'	20000	10000	78
'O'	12100	12100	79
'P'	12121	12100	80
'P'	12100	12100	80
'R'	12120	12321	82
'R'	12120	12000	82
'S'	12121	23200	83
'S'	12121	23000	83
'S'	12121	30000	83
'T'	10000	10000	84
'U'	21000	10000	85
'U'	21000	12100	85
'V'	21000	10000	86
'V'	21000	12100	86
'Y'	21000	10000	89
'Y'	21000	12100	89
'Z'	12100	23200	90
'Z'	10000	23200	90
'Z'	10000	32000	90
'Z'	10000	20000	90

**Table D-2 Plate Database**

<b>Image Number</b>	<b>License Plate</b>	<b>Image Number</b>	<b>License Plate</b>	<b>Image Number</b>	<b>License Plate</b>
Image_000	38 NE 941	Image_036	35 KFM 75	Image_072	06 TRZ 81
Image_001	06 ANZ 72	Image_037	06 KNT 90	Image_073	06 VVL 74
Image_002	35 EL 786	Image_038	06 RNM 90	Image_074	06 KJS 97
Image_003	06 LS 717	Image_039	06 YRU 41	Image_075	06 ZFE 94
Image_004	06 EEA 80	Image_040	06 ZFS 48	Image_076	06 BM 277
Image_005	06 AAH 83	Image_041	32 AN 568	Image_077	06 GV 261
Image_006	06 ZTM 61	Image_042	06 JV 031	Image_078	06 ZVR 08
Image_007	06 ACF 96	Image_043	06 ZTT 60	Image_079	16 NB 505
Image_008	06 NV 015	Image_044	06 GM 575	Image_080	06 KYY 24
Image_009	34 UEN 01	Image_045	06 ZVS 41	Image_081	06 LCE 24
Image_010	06 AKV 30	Image_046	06 LM 649	Image_082	06 TAV 30
Image_011	06 VCU 16	Image_047	06 LMS 97	Image_083	06 BB 162
Image_012	06 FG 382	Image_048	06 JM 694	Image_084	06 PB 353
Image_013	06 V 6304	Image_049	06 LPC 54	Image_085	06 VB 035
Image_014	06 GJ 988	Image_050	06 VB 035	Image_086	06 YNZ 47
Image_015	06 GD 229	Image_051	06 VTK 75	Image_087	06 FM 787
Image_016	06 BF 926	Image_052	06 ZVJ 64	Image_088	06 UB 952
Image_017	11 AK 680	Image_053	06 TNA 59	Image_089	06 ZBK 09
Image_018	06 ZVP 28	Image_054	06 EM 653	Image_090	06 SG 911
Image_019	41 DA 848	Image_055	06 DJ 294	Image_091	06 RNH 19
Image_020	06 VTE 73	Image_056	06 FKS 81	Image_092	06 ZMJ 55
Image_021	06 AJC 01	Image_057	06 MLD 18	Image_093	06 YML 39
Image_022	06 BS 913	Image_058	06 KJ 258	Image_094	06 EAE 87
Image_023	06 E 9787	Image_059	06 YZC 78	Image_095	06 EB 195
Image_024	06 UE 242	Image_060	06 H 0184	Image_096	06 UB 008
Image_025	45 EE 616	Image_061	06 GC 115	Image_097	06 HLD 71
Image_026	06 YAM 63	Image_062	06 HB 496	Image_098	06 ZKM 55
Image_027	06 YUY 61	Image_063	06 FVY 64	Image_099	06 VTF 54
Image_028	06 GH 039	Image_064	06 SC 197	Image_100	06 JH 617
Image_029	06 LN 567	Image_065	06 SM 805	Image_101	06 KUH 65
Image_030	06 FFA 51	Image_066	06 KTY 39	Image_102	06 NB 150
Image_031	06 YNP 76	Image_067	06 AD 406	Image_103	06 FJ 557
Image_032	06 RED 88	Image_068	06 YUF 55	Image_104	06 YTE 47
Image_033	06 Z 6339	Image_069	06 YBT 50	Image_105	06 DJ 294
Image_034	06 MVB 27	Image_070	06 YML 50	Image_106	06 SC 197
Image_035	06 YDU 85	Image_071	06 ZSS 91	Image_107	06 GV 613

**Table D-2 Plate Database (Continued)**

<b>Image Number</b>	<b>License Plate</b>	<b>Image Number</b>	<b>License Plate</b>	<b>Image Number</b>	<b>License Plate</b>
Image_108	06 VV 941	Image_131	06 JK 870	Image_154	06 GR 895
Image_109	06 AB 664	Image_132	06 YB 868	Image_155	06 FM 969
Image_110	06 MML 54	Image_133	06 GA 049	Image_156	06 SS 798
Image_111	06 UM 938	Image_134	06 VUF 61	Image_157	06 TKD 26
Image_112	06 GJ 015	Image_135	06 GV 354	Image_158	34 VD 4751
Image_113	06 AB 789	Image_136	06 YZC 78	Image_159	06 RZJ 36
Image_114	06 EHR 26	Image_137	06 MDB 52	Image_160	06 RHS 42
Image_115	06 PG 742	Image_138	06 ZHT 89	Image_161	06 BM 277
Image_116	06 VPC 41	Image_139	06 BB 162	Image_162	06 BB 162
Image_117	06 MUZ 64	Image_140	06 ZZE 84	Image_163	06 FU 644
Image_118	06 FPL 06	Image_141	06 LEE 85	Image_164	06 BL 040
Image_119	06 GN 623	Image_142	06 DM 529	Image_165	06 TSZ 03
Image_120	06 RVR 44	Image_143	06 KHZ 04	Image_166	06 EM 653
Image_121	06 YMH 60	Image_144	06 GV 492	Image_167	06 EKN 80
Image_122	06 TNA 59	Image_145	06 YMH 26	Image_168	06 MKC 25
Image_123	06 MZK 93	Image_146	06 KVF 51	Image_169	06 MHP 10
Image_124	06 JB 489	Image_147	06 VVR 58	Image_170	06 MBH 36
Image_125	06 LB 509	Image_148	06 KFY 42	Image_171	06 MZK 93
Image_126	06 TKL 06	Image_149	06 GA 965	Image_172	06 ZFS 22
Image_127	06 KPM 96	Image_150	06 GD 009	Image_173	06 JB 489
Image_128	06 YD 468	Image_151	06 GH 264	Image_174	06 ZVJ 64
Image_129	06 BN 824	Image_152	06 VKC 70		
Image_130	06 KKY 99	Image_153	06 KG 629		