Master Thesis

FPGA Implementation of License Plate Detection and Recognition

Farid Rosli

Matriculation Number: 337961

Technische Universität Berlin
School IV · Electrical Engineering and Computer Science
Department of Computer Engineering and Microelectronics
Embedded Systems Architectures (AES)
Einsteinufer 17 · D-10587 Berlin

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Engineering

according to the study and examination regulations for the master's degree programme in Computer Engineering of 25.01.2012 at the Technische Universität Berlin.

Department of Computer Engineering and Microelectronics
Embedded Systems Architectures (AES)
Technische Universität Berlin
Berlin

Author Farid Rosli

Thesis period 2014-05-05 to 2014-11-08

Referees Prof. Dr. Ben Juurlink, Embedded Systems Architecture (AES)
Prof. Dr.-Ing. Carsten Gremzow, Computer Technology (RT)

Supervisor Dr.-Ing. A. Elhossini, Embedded Systems Architecture (AES)

Declaration

According to §13(10) of the Examination Regulations

I hereby confirm to have written the following thesis on my own, not having used any other sources or resources than those listed.

Berlin, March 23, 2015

Farid Rosli

Abstract

Rapid development of the Field Programmable Gate Array (FPGA) offers an alternative way to provide acceleration for computationally intensive tasks such as digital signal and image processing. Its ability to perform parallel processing shows the potential in implementing a high speed vision system. Out of numerous applications of computer vision, this thesis focuses on the hardware implementation of one that is commercially known as Automatic Number Plate Recognition (ANPR). It is a modern CCTV-based surveillance method that has the ability to locate and recognize vehicle registration number. Morphological operations and Optical Character Recognition (OCR) algorithms have been implemented on a Xilinx Zynq-7000 All-Programmable SoC to realize the functions of an ANPR system. Test results have shown that the designed and implemented processing pipeline that consumed 63 % of the logic resources is capable of delivering the results with relatively low error rate. Most importantly, the computation time satisfies the real-time requirement for many ANPR applications.

Zusammenfassung

Rasante Entwicklungen der Field Programmable Gate Array (FPGA) bietet eine Alternative, rechenintensive Aufgaben wie digitale Signal- und Bildverarbeitung zu beschleunigen. Die Fhigkeit zur parallele Verarbeitung zeigt ein Potential für die Implementierung eines Hochgeschwindigkeitsvideoverarbeitungssystems. Aus den zahlreichen Anwendungen der Computer Vision konzentriert sich diese Arbeit auf der Hardware-Implementierung eines so genannten Automatic Number Plate Recognition (ANPR). Es ist ein modernes videobasiertes Überwachungssystem, das die Fähigkeit zur Lokalisierung und Erkennung des Fahrzeugkennzeichens besitzt. Morphologische Operationen und Algorithmen zu Optical Character Recognition (OCR) wurden auf dem Xilinx-Zynq-7000 All-Programmable SoC implementiert, um die Funktionen eines ANPR-Systems zu realisieren. Die Testergebnisse haben gezeigt, dass die entworfene und implementierte Verarbeitungspipeline, die 63 % der Logikressourcen verbraucht in der Lage ist, die Ergebnisse mit niedrigen Fehlerrate zu liefern. Am wichtigsten ist dass die Rechenzeit die Echtzeitanforderungen für viele Anwendungen der ANPR erfüllt.

Contents

List of Tables

List of Figures

1 Introduction

In recent years, the significant evolution of computer vision can be seen as it is making its way into an increasing number of application domains. The research and development in the field of computer or machine vision has defined methods for processing and analyzing image from real world to provide human capabilities of understanding images to machines and robots.

There is a strong and growing demand for computer vision systems in the automotive domain. Intelligent cars that are available in the market nowadays are equipped with various camera-based driver assistance systems such as lane detection, night view assist, pedestrian detection and traffic sign recognition [14]. These applications are required to work reliably in a large range of lighting and climactic conditions and to process a very high frame rate video signal in real-time.

Besides those above-mentioned applications, the Automatic Number Plate Recognition (ANPR) is also one of the computer vision applications that is widely used in the automotive domain. However, it is better known as a surveillance technology rather than a driver assistance system. Vehicles are usually identified by their registration number which are easily readable by humans. But for machines, a plate number is a grey image defined as a mathematical function that represents light intensity at a certain point within an image [20].

1.1 Configurations for various ANPR applications

ANPR system is a set of hardware and software components that receives a graphical signal such as static picture or sequence of images and recognizes license plate characters from it. The hardware part of the system mainly consists of camera and image processor. With minimal configuration like this, the system has to perform continuous sampling and processing of video signal. This method require high power consumption and system resources. A hardware trigger that works in conjunction with a sensor that detects presence of a vehicle can be installed in the hardware part of the system so that it will be activated and deactivated automatically [6]. This is a very efficient solution for applications such as car parks and access control which are not deployed in areas with congested traffic.

It is often necessary for the ANPR systems to use a capturing device with a very high shutter speed to avoid an undesired motion blur effect in the snapshot that is caused by the fast movement of the vehicle. If an application processes a video instead of a still

image, using a high frame rate video camera is recommended. This is an important requirement if an ANPR system is intended to be used for monitoring vehicle speed. Unlike normal camera that works in a visible light spectrum, an infrared camera can improve the object detection process as it works better in the darkness or low light condition. The number plate that is made of reflexive material are much more highlighted than the rest of the image and hence make the detection and recognition easier [6].

The image processing unit of the ANPR system recognizes a snapshot captured by the camera and returns a text representation of the detected license plate. ANPR cameras can send captured images to a central processing unit for further processing or they can also have its own dedicated image processors. For the implementation of real-time ANPR systems, the latter approach is normally used, which is also the focus in this thesis. The algorithms can initially be developed and tested on powerful desktop computers before they are ported to a low cost and compact embedded computing platform.

In some applications, camera positioning is done in different ways. For instance, the cameras for vehicle speed and traffic monitoring are installed overhead about a few metres. In Figure 1.1, it is shown that camera height and detection range must be adjusted such that the vertical angle remains below a certain value, which is typically 30°[11][2]. Different types of camera offers different coverage width. A wide lens camera is an excellent choice to perform vehicle detection on multiple lane, whereas a telescopic lens has a very large detection range but small coverage width and thus suitable to monitor one lane.

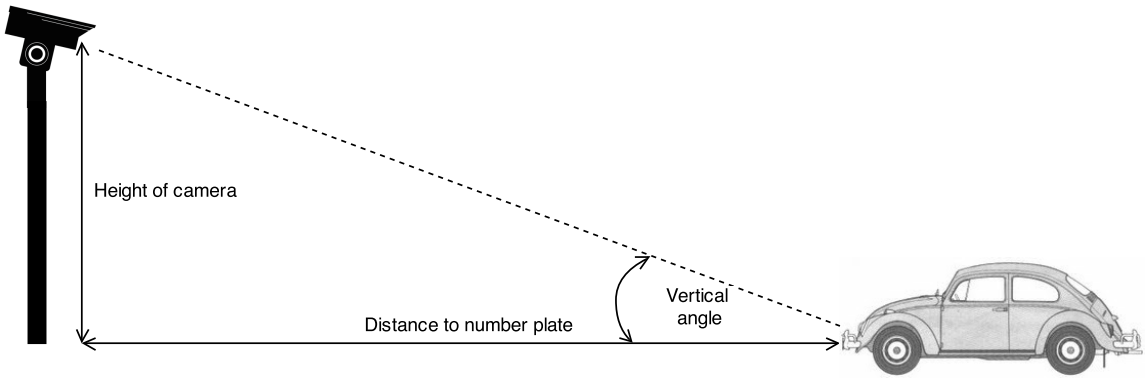


Figure 1.1: Vertical angle and camera height.

In parking and access control, the camera is positioned away from the line of travel as shown in Figure 1.2. Another configuration parameter that needs to be considered in this case is the distance of the camera from the line of travel. If the horizontal angle is too large, the image of the license plate will be skewed which will affect the detection and recognition results.

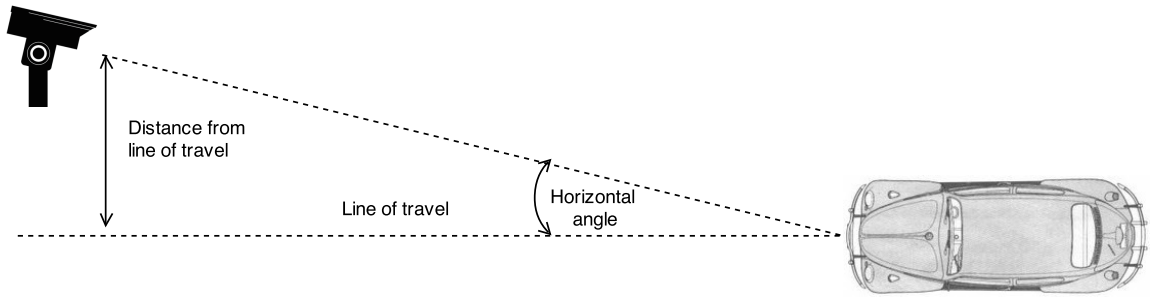


Figure 1.2: Horizontal angle and distance from line of travel.

1.2 Project details

The main objective of this thesis is to create a framework for a custom computer vision application that runs on a reconfigurable hardware and operates in real-time. An application in the automotive domain is chosen, which is the ANPR to be implemented on FPGA. The selected application must be constructed with standard computer vision and image processing algorithms, which later can simply be reused by other vision applications.

The first step is to investigate some processing methods and algorithms such as image filtering and implement them as digital logic circuits. These circuit elements will be combined to realize a custom processing block. Custom processing blocks are arranged as a processing pipeline to perform the license plate detection and recognition. All of the components must be developed to becomse highly configurable so that the whole processing pipeline can be ported easily from one FPGA device to another. Simulation of the processing pipeline is performed to verify that the processing pipeline works as intended. When the simulation shows expected results, the pipeline is synthesized to fit in the target device which is the Xilinx Zynq-7000 All Programmable SoC. It has a dual core ARM Cortex-A9 processor that operates up to 667 MHz and is targeted for video processing applications.

In this thesis, the scenario as shown in Figure 1.3 is implemented. The camera positioning for this case follows Figure 1.1, which is suitable for wide range of applications. The Z-7020 is the FPGA, in which the processing pipeline that consists of two stages is implemented. In real case, the detected license plate will be directed to central database where it will be viewed by the authorities.

1.3 Outline

This thesis is divided into 7 chapters and organized as follows:

Chapter 2 gives overview of other related research works that proposes their own methods

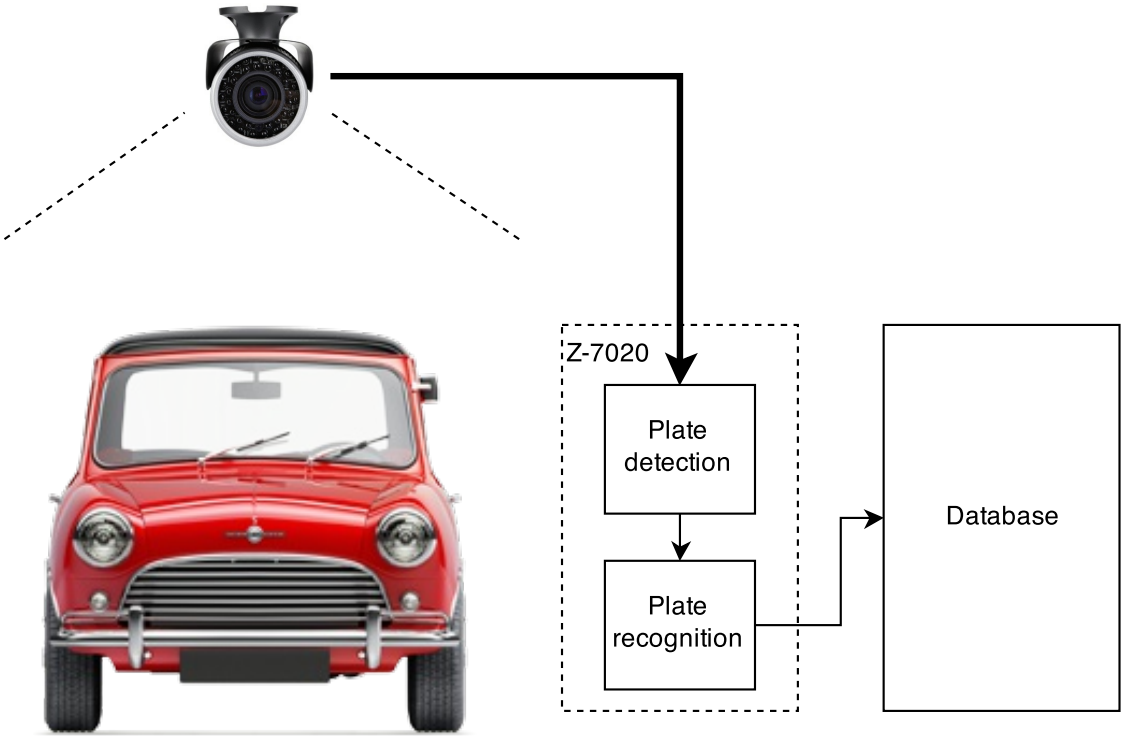


Figure 1.3: System setup for license plate detection and recognition system with FPGA.

for license plate detection and recognition. Some of the proposed methods are studied and improvised to develop the processing pipeline in this thesis.

Chapter 3 provides the fundamentals of image processing and pattern recognition algorithm that are used for the realization of ANPR system.

Chapter 4 describes the proposed methods for license plate detection and recognition. It is designed and tested on a desktop using MATLAB. Software flow diagram for each stage is also included.

Chapter 5 reveals the architecture of each processing block. It is divided into three sections. The first section describes the top module of the processing pipeline. Hardware architecture for the license plate detection and character recognition will be explained in the next section. The last section contains the hardware implementation for standard image processing algorithms such as morphological operation, connected component labelling, bounding box and also several OCR algorithms. All of these components that are developed in this thesis are meant to be generic and reusable for other computer vision applications.

Chapter 6 presents the testing procedure and analysis of the implemented license plate

detection and recognition system. Performance between the MATLAB code that is used to verify the proposed method and the hardware processing pipeline is compared.

Chapter 7 summarizes the whole work that is done so far and also includes a few suggestions for further improvements on the applied methods for license plate detection and recognition.

2 State of the Art

The advances in semiconductor and computer technology have contributed in the evolution of a wide variety of powerful hardware architectures. Field Programmable Gate Array (FPGA), Graphic Processing Unit (GPU) and Multicore Central Processing Unit (CPU) are three different architectures that have dominated the field of computer vision due to their parallel nature [18]. Parallel architectures are often desired when dealing with massive amount of visual inputs. Assigning complex computation tasks to hardware and exploiting the parallelism and pipelining in computer vision algorithms will yield significant speed up in run time.

Factors such as programmability, performance, programming cost and sources of overhead in the design flows must be all taken into considerations. Generally, FPGA provides the best expectation of performance, flexibility and low overhead. GPU tends to be easier to program and require less hardware resources [23]. One problem with programming the GPU is the instability of the frameworks and most of them are vendor specific [15].

In comparison with GPU and multicore CPU, FPGA based system offers better computational power consumption. Its parallel processing capability, low cost and reconfigurable hardware are the key for a vision system to obtain the best performance. Due to the complexity of vision systems, more work has been dedicated to implement most part of the vision algorithms on hardware using FPGA. Modern FPGAs are equipped with several embedded resources such as embedded processor cores. This increase the capabilities of these devices to implement complex systems.

2.1 Related Work

Today, video surveillance system are installed widely to detect vehicle on the road and identify them by using image processing algorithms to recognise their number plates. Generally, this is done in two steps. The first step is known as license plate detection or localisation. When a vehicle is detected by a camera, its license plate will be located within the image. This is followed by a so called license plate character recognition that determines the plate number.

Different methods can be applied to detect the license plate within an image. Most common approaches are the combination of edge detection and binary morphology as explained in [22]. The detection rate with this method is highly affected by the quality of the image. Furthermore, they are based on the assumption that car plates have strong

edges that will survive strong filtering. Unfortunately, this is not very effective for urban environment. Colour filtering methods can also be used, but it is not suitable since different countries use different background colour for their license plates. License plate detection and segmentation algorithm with histogram projection as proposed in [19] and [8] is found to be challenging especially when the image contains a lot of details in the background. Defining the threshold value require a few steps of mathematical calculation and the process of finding the peak may introduce some delay in the processing. A simpler approach that produces satisfactory result is by using greyscale morphology that is already used in [16] and [21].

Optical character recognition (OCR), that is normally used to translate scanned images of handwritten text into machine encoded text, is applicable for the recognition of license plate characters. The work in [27] divides a handwritten character into several rectangular zones to obtain a 13-element feature vector. Each element represents the number of foreground pixel in each defined zone. In [17] the similar method is reused to divide a 32 x 32 pixel character image into 16 zones. The authors have proposed eight directional distribution features which are calculated for each zone. Three different classification methods, which are artificial neural networks (ANN), support vector machine (SVM) and k-nearest-neighbour (KNN) are used to recognise the characters.

Some of the methods and algorithms described in the research works above can be combined and implemented on embedded hardware platform. License plate detection and recognition on an embedded DSP platform is covered in [10]. The total processing time, beginning from image acquisition until character classification requires 41.35 ms. The implementation of plate localisation on Xilinx Virtex-4 as described in [26] is capable of processing one image in 3.8 ms and it consumes less than 30% of the on-chip resources. It shows that modern FPGAs provide sufficient resources for a complete license plate detection and recognition system. A complete ANPR system is implemented on FPGA in [28]. The system that utilizes 80 % of the Xilinx Virtex-4 LX40 resources is capable of processing a standard definition image (640 x 480) in less than 10 ms.

Algorithms for license plate detection is also developed in [25] using Handel-C and then translated into Verilog HDL with Celoxca's DK4 to implement on Virtex II Pro. Performance comparison between the software and the hardware implementation is stated at the end of the paper. The resource utilization for their hardware implementation is about 2 to 3 times more than the work in [26] and yet requires longer processing time for an input image with a smaller resolution. It indicates that the Verilog HDL code is not well optimized by the translation tool. However, it is proven in their achieved results that the same algorithms execute 4 times faster with the hardware than the software.

Some companies have their own solution for the ANPR system and offer that as a commercial product. For example, Imagsa Technologies develops a smart camera called Atalaya that has a resolution of 2048 x 1024 pixel to perform detection and recognition on two lanes [1]. Parallel processing algorithms are implemented on FPGA to provide capabil-

ties of real-time scene analysis and high-speed image processing to their system. With integrated high speed megapixel CMOS sensor, the system is capable of processing 250 frames per second.

2.2 Summary

There are a large number of completed and also still ongoing researches related to the license plate detection and recognition. Most of them focused on defining new techniques and methods by applying different vision algorithms. Implementation on embedded platform were mostly done with DSP because only in recent years FPGA has shown potential in the field of computer vision. Therefore, this thesis adopted some of the defined methods and algorithms to design and implement parallel processing on a reconfigurable hardware. However, computationally expensive image processing and pattern recognition algorithms such as AdaBoost, watershed segmentation, Gabor filter, neural networks and SVM classifier that will potentially cause processing delay and inefficient resources allocation are not considered.

3 Fundamentals

The overview of the image processing algorithms that are implemented on the processing hardware are provided in this chapter. It will cover the basics of morphological filtering that is applied in the pre-processing stage. Connected component labelling and bounding box algorithm are also very often used in computer vision and image processing. These two important algorithms will be mentioned briefly in this chapter. Additionally, an algorithm for pattern recognition that is known as k-means clustering will also be introduced.

3.1 Morphological operation

The identification of objects within a digital image can be a very challenging task. Morphological algorithm is usually applied to binary images, in which a pixel's value is restricted to either 0 or 1. However, it also works with grey scale images. An essential part of the morphological operations is the structuring element that is used to probe the input image (Figure 3.1). A structuring element is a matrix containing 0's and 1's that can have arbitrary shape and size. The centre pixel of the structuring element identifies the target pixel which is to be processed. In a morphological operation, the value of each pixel in the output image is based on comparison of the corresponding pixel in the input image with its neighbours.

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Figure 3.1: Example of a square structuring element 5 x 5.

The value of the output pixel can be determined by applying a certain rule to the corresponding pixel and its neighbouring pixels in the input image. These rules define the morphological operation as dilation or erosion. For dilation, the value of the output pixel is the maximum value of all the pixels in the input pixel's neighbourhood. It makes an object become larger and tends to remove holes that exist within the object. For erosion,

the value of the output pixel is the minimum value of all the input pixel's neighbourhood. Erosion tends to remove pixels on object boundaries and reduce the size of an object. The number of pixels added or removed from the objects depends on the size and shape of the structuring element used to process the image [24].

(a) Grey scale image

(b) Dilation

(c) Erosion

(d) Closing

(e) Opening

Figure 3.2: Example of grey scale morphological filtering with structuring element 5 x 5

Dilation and erosion are often used in combination to implement image processing operations. For example, the morphological closing of an image is defined as dilation followed by erosion. Both operation is done by using the same structuring element. The reversed operation is called morphological opening, where erosion is done before the dilation by

using the same structuring element. Opening removes lighter blobs from the image that are smaller than the structuring element, whereas closing removes darker blobs [7]. Figure 3.2 illustrate the results of different morphological operation that is applied on an original grey scale image.

Several techniques may be used to simplify and reduce the computation complexity of the morphological filter, and thus reduce the logic requirement for the implementation with FPGA. The structuring element in Figure 3.1 is separable, which means it is decomposable into a cascade of two one-dimensional filters. Firstly, a 1 x W filter is applied on the input image and then the resulting image is filtered with a W x 1 filter. This will reduce the number of both multiplications and associated additions from $W2$ to $2W$, where in this case W is equal to 5.

3.2 Object detection

Object detection is the process of finding a specific region in a digital image. A simple object detection process can be done on a binary image that is shown in Figure 3.3. Two distinct regions that appear as connected white pixels may represent objects in the original RGB image that must be localised by the vision system. These regions may also be called regions of interest (ROI).

Figure 3.3: A pre-processed image that contains two regions.

The problem that appears to the system is it does not know how many regions could appear in an image. Furthermore, if the image contains two or more regions, the system must able to distinguish all of them. An existing solution for this problem is to label those connected white pixels with a unique label. The algorithm that does exactly this is called connected component labelling. In fact, it is one of the most important steps in many machine vision applications and also one of the most fundamental operations.

Two types of connectivity can be used to label the target pixel. These types are shown in Figure 3.4. The first type is known as 4-connectivity. During the labelling process, only the above and the left neighbouring pixels are checked if they are not background pixel. The second type is called 8-connectivity which considers four neighbouring pixels that are

(a) (b)

Figure 3.4: Types of connectivity for connected component labelling. (a) 4-connectivity (b) 8-connectivity

highlighted in yellow in the figure. This algorithm is executed by iterating the image pixel by pixel and assign a unique label to each foreground pixel. Normally, positive integer or natural numbers are used as label and the first detected object in the image will have the least number from other objects. A set of connected foreground pixels that forms a region in a binary image will have the same label. For example, the binary image in Figure 3.3 will produce two different labels (e.g. 1 and 2) because there are two regions.

The classic connected components labelling algorithm requires two passes through an image. In the first pass, when an object pixel is encountered, the adjacent pixels that have already been labelled are examined to determined the label for the current pixel. If they are background pixels, a new label will be assigned. Otherwise, the pixel will inherit the label from its neighbour. For a **U** shaped object, each of the branches will have a different label and when they join at the bottom, these labels must be merged. One of the two labels will continue to be used and all instances of the other labels need to be replaced with the label that was retained. Since many merges may occur in the processing of an image, relabelling is deferred so that all of the merged labels may be changed at once. A merger table records such merges, and is used to relabel all of the pixels within the image consistently in the second pass [7].

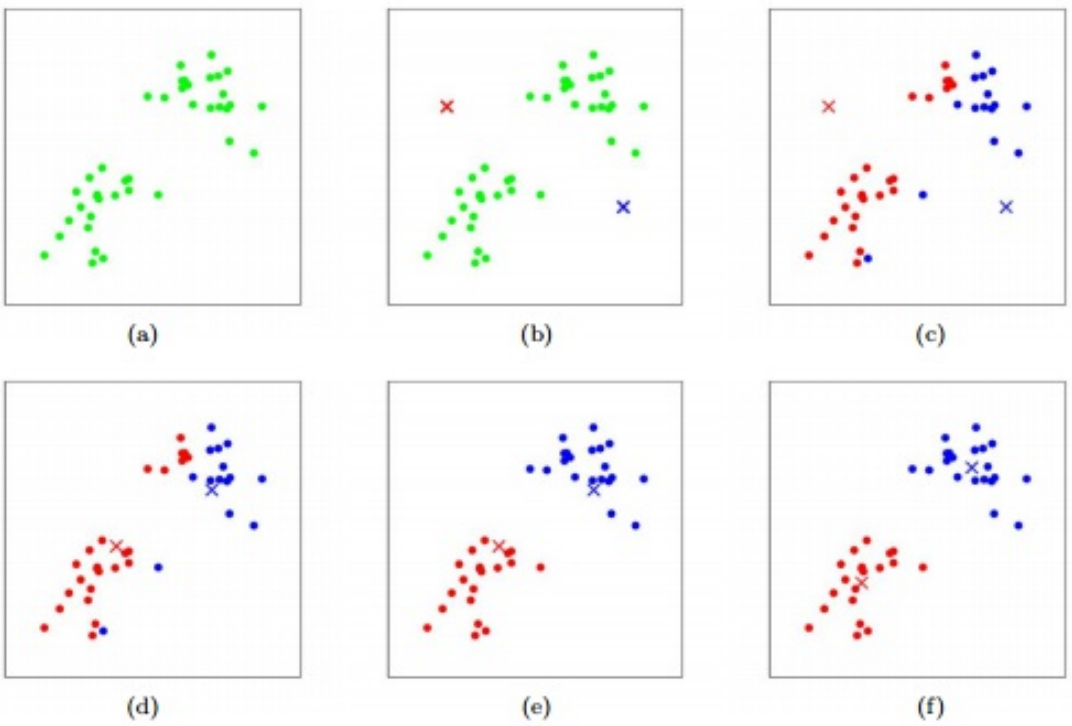
Besides labelling, the coordinate of an object pixel can also be recorded. Minimum and maximum of x and y-coordinate that belong to a label will determine the rectangular boundary that encloses the detected region. Mathematically, the rectangular boundary or also known as bounding box can be defined by the Equation 3.1. With this information, the specific region within an image can be segmented or cropped from the original image to be further processed.

$$\begin{aligned} x_{min,i} &= min\{x_p|(x_p,y_p \in C_i)\} \\ x_{max,i} &= max\{x_p|(x_p,y_p \in C_i)\} \\ y_{min,i} &= min\{y_p|(x_p,y_p \in C_i)\} \\ y_{max,i} &= max\{y_p|(x_p,y_p \in C_i)\} \end{aligned} \tag{3.1}$$

Where C_i is a set of pixels associated with object i.

3.3 Pattern Classification

Nowadays several unsupervised classification algorithms are used in pattern recognition. k-means clustering is one of the simplest unsupervised learning algorithm. The main idea is to partition the data points into smaller number of clusters. In general, there are n data points that have to be grouped into k clusters that can be selected randomly from any data point. The remaining data points will be assigned to their closest centroid. In order to accomplish this, the distances between a data point and the selected centroids are calculated. Euclidean distance is normally used in this algorithm. When all available data points are assigned to a cluster, a mean vector is calculated using all the data points within the cluster. The mean vector becomes the new centroid and hence some of the data points might be assigned to new centroids. The iteration is done until the assignment of every data points does not change. Figure 3.5 demonstates the process of finding the centroids.



[Source: http://stanford.edu/~cpiech/cs221/handouts/kmeans.html]

Figure 3.5: The data points are shown as dots and the centroids are shown as crossees. (a) Plot of the data points. (b) Random initialization of centroids. (c) to (f) Illustration of iterations to determine the final position of the centroids.

This algorithm aims at minimizing an objective function known as the sum of squared distances (errors) between each data point and its cluster centroid. The function is defined in the following Equation 3.2.

$$SSE = \sum_{i=1}^{M} \sum_{j=1}^{N} (\|x_i - v_j\|)^2 \tag{3.2}$$

Where

M	is number of clusters.
N	is the number of data points in a cluster.
$\|x_i - v_j\|$	is the Euclidean distance between cluster centroid and data point.

k-means-clustering algorithm has a few advantages as well as drawbacks. It is fast, robust and relatively efficient. However, the data sets must be well separated from each other to achieve the best result. Otherwise, random selection of centroid during the training phase would not yield satisfactory results. In the other words, the data points might not be categorised correctly [3].

4 Proposed Algorithm

Before the implementation on the hardware begins, the complete algorithm that detects and recognises a vehicle license plate in an image must be carefully designed and tested. The goal is to develop an algorithm that is tolerant to noise signals and able to detect a vehicle license plate with a specific shape, orientation and size. MATLAB, an interactive development environment from MathWorks, is chosen for this purpose due to its simplicity. The algorithm works on an input image that has a resolution of 640 x 480 pixels (Figure 4.1). The image provides either a front or rear view of a car where its license plate is normally attached.

Source: http://www.bmwblog.com/2011/07/05/retrospective-driving-impressions-of-a-bmw-2002/

Figure 4.1: Example of an input image with a specified resolution.

The software flow diagram is presented in Figure 4.2. The subroutines for the detection and recognition part will be explained throughout the chapter. At the beginning, the database that contains training data for character recognition is loaded. Section will provide a brief explanation of the offline training procedure to construct the database. Then, the parameters such as threshold value for image binarization and the dimensions of the license plate as well as the characters must be initialized.

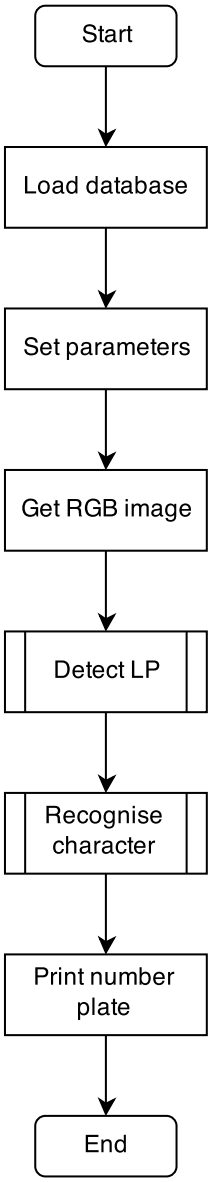


Figure 4.2: The software flow diagram of a license plate detection and recognition system will represent the processing pipeline in the hardware implementation.

4.1 License plate detection

The software flow diagram for the license plate detection algorithm is shown in Figure 4.3. Firstly, an input image as shown in Figure 4.1 will be converted to grey scale image to determine the region of interest. The grey scale image will go through several stages of greyscale morphology with different sizes of rectangular structuring elements.

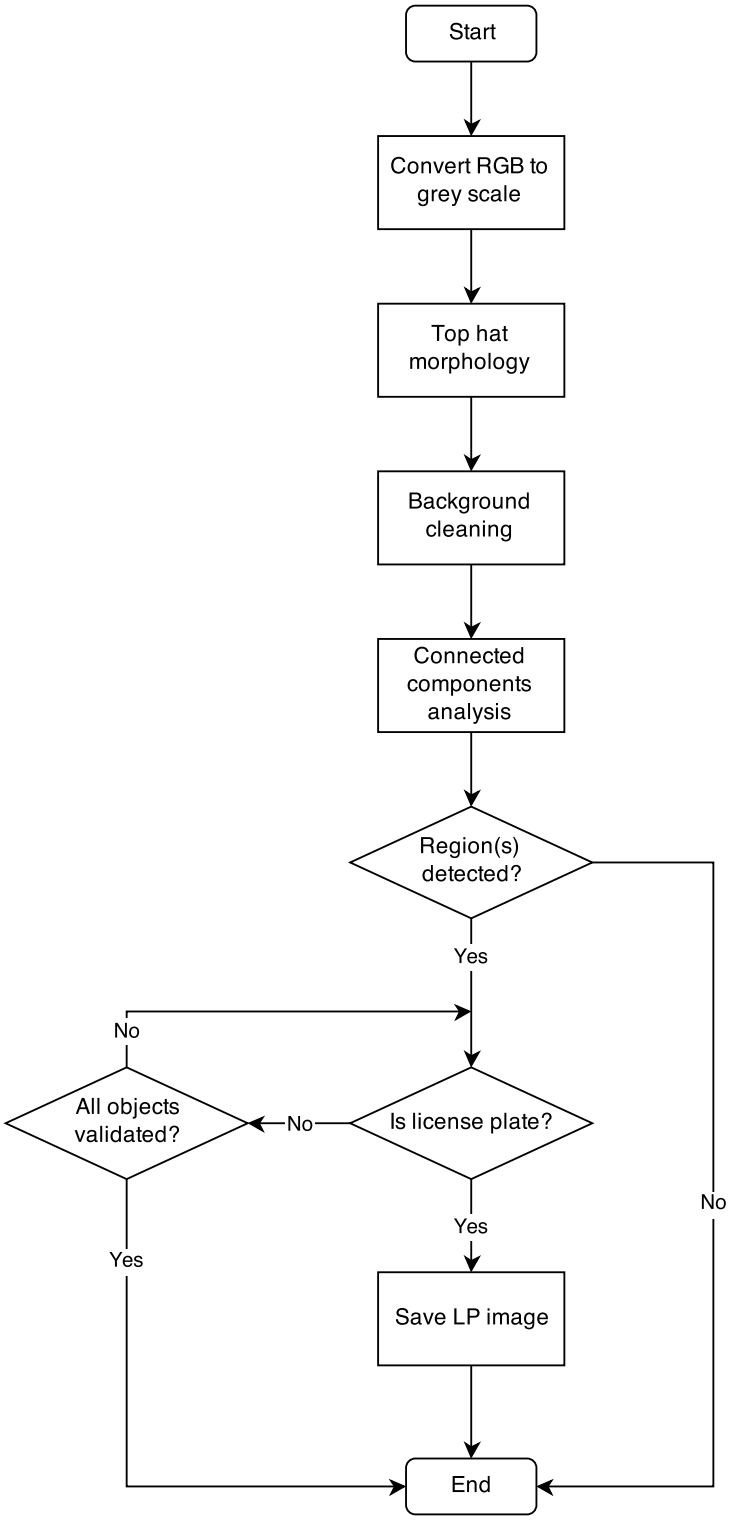
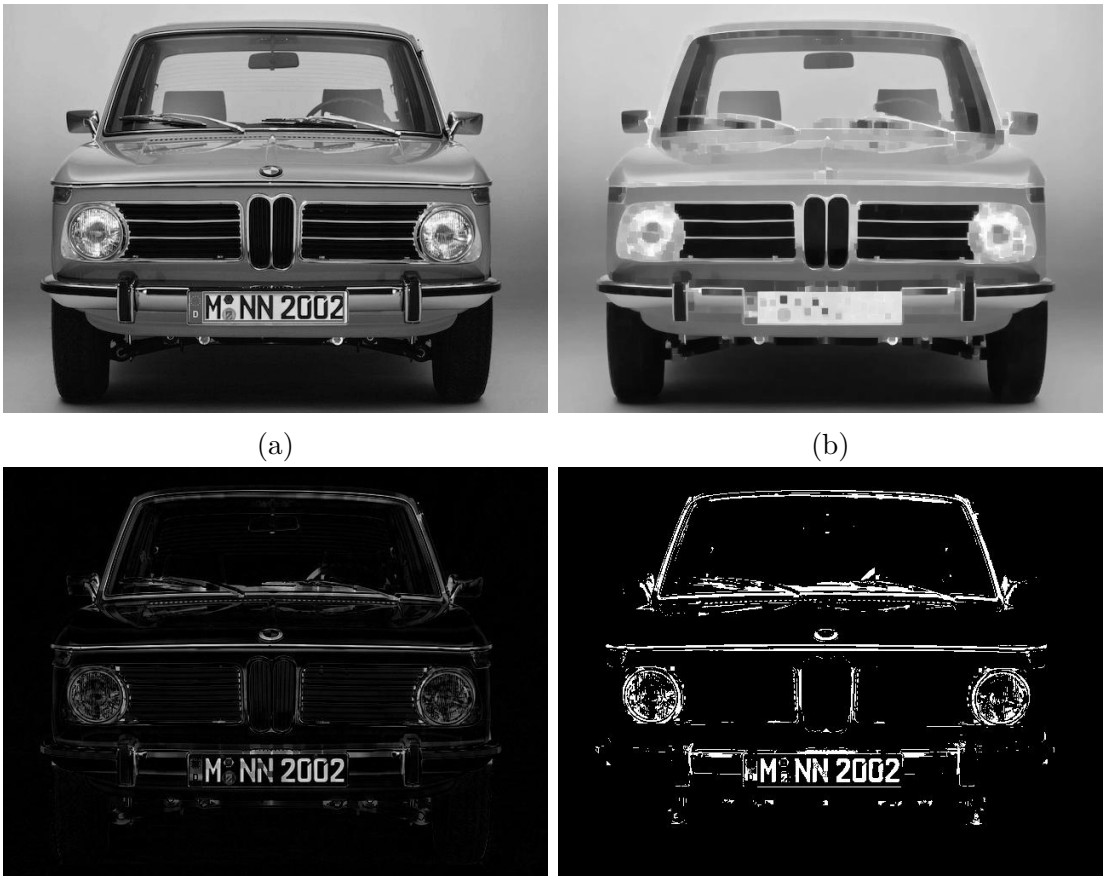


Figure 4.3: Software flow diagram of a license plate detection stage.

4.1.1 Top hat

The approach with mathematical morphology is chosen to locate the license plate of the vehicle in the image. Firstly, the closing operation with a structuring element of 7 x 7 pixel is applied to erase the characters of the license plate in the image (Figure 4.4b). When image subtraction is performed between the resulting image and the initial grey scale image, an image as shown in Figure 4.4c is obtained. This operation is known as black top hat morphology. It returns an image containing object or element that is smaller than the structuring element and darker than its surrounding. Since european license plates mostly has white background and the characters are black in colour, they will remain as foreground objects in the output image if the size of the structuring element is chosen correctly. A binary image (Figure 4.4d) is obtained from the top hat image by using thresholding.



(a) (b)

(c) (d)

Figure 4.4: Black Top Hat Transform. (a) is produced by subtracting the grey scale image in (b) and the result of morphological closing in (c). A binary image in (d) is produced by thresholding the image.

4.1.2 Background cleaning

Apart from the license plate characters, there are still some unwanted elements that appear in the resulting image. Therefore, greyscale morphology is applied again to the top hat image to find the region that contains the vehicle license plate.

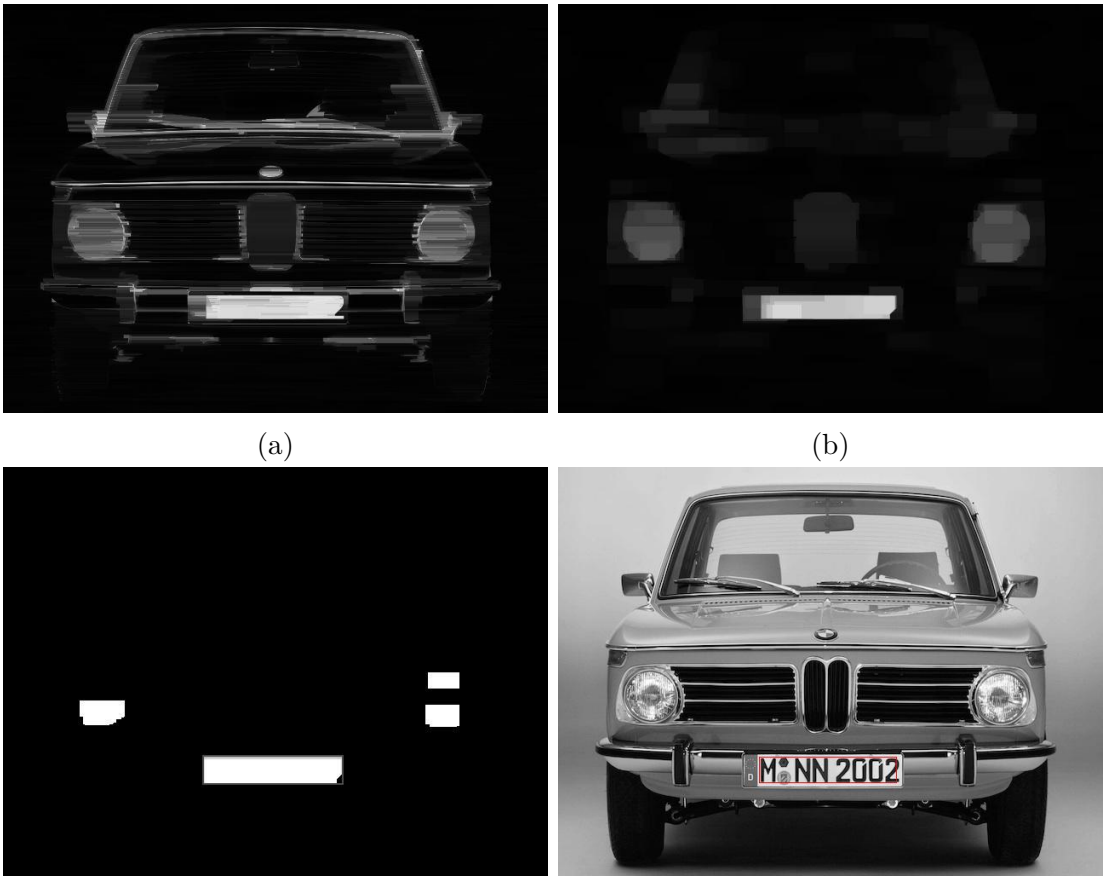


Figure 4.5: Finding the region that contains the car license plate. (a) Rough detection is done with morphological closing by using horizontal structuring element. (b) The unwanted details are removed with morphological opening. (c) Binary image is produced by thresholding. (d) The bounding box is drawn on a grey scale image as a red rectangle.

The license plate location can be detected roughly with a closing operation (Figure 4.5a). The closing operation uses a structuring element with a size of 1 x 45 pixel. The length is chosen such that it has at least twice the length of a license plate character. After that, the unwanted elements that does not belong to the license plate area is removed by using morphological opening with a rectangular structuring element 15 x 25 pixel (Figure 4.5b). The resulting image contains several noticeable light areas, and the area of the license plate appears to be lighter than other areas. Thresholding is applied to produce a binary image that maintains the license plate area and suppresses the darker regions (Fig-

ure 4.5c). Finally, and optionally, binary morphology dilation with a structuring element of 5 x 5 pixel can be applied to the binary image in order to enlarge the license plate region.

The presence of noise may result other regions to appear in the image. At this point, the implemented algorithm will apply the connected component labelling algorithm to be able to distinguish these regions. They will be segmented by applying bounding box algorithm to find the rectangular boundary that encloses each region (Section 3.2). A license plate will appear at a certain size in the image. Its minimum width should be 20 % of the image width. The height of the license plate should be 5 % of the image height. This criteria will be applied in license plate segmentation by determining if the width and length of each bounding box satisfy this condition.

4.2 Character segmentation

Once the region of the license plate is known, it is cropped from the binary image based on the bounding box information. The process of finding characters on a license plate is actually the same as finding the license plate in the input image. This is illustrated by the software flow diagram in Figure 4.6. Connected component analysis is used again to label each character and a bounding box that encloses each character is defined so that it can be segmented from the image and sent to the recognition unit or classifier. Similarly, there are unwanted elements or noise within the image of the license plate. These elements will also be detected as objects or regions by the connected component labelling. The same criteria is applied as in the license plate detection to ensure that the detection regions are license plate characters. The bounding box that encloses a detected object must have a minimum width and height. The result of character segmentation is shown in Figure 4.7. Valid objects, which are the license plate characters are bounded with red rectangle.

4.3 Feature extraction

The process of extracting features from a license plate character is shown in Figure 4.8. In optical character recognition, a method called zoning is one of the approaches for feature extraction. An image can be divided into several sections. A feature such as pixel density can be determined for each section or zone. An image of a license plate character is partitioned into eight zones as shown in Figure 4.9. Pixel density, which in this case in simply the number of foreground pixel in each zone is calculated. Thus, an image can have a feature vector of at least eight elements.

Character recognition process can be made more reliable by combining several methods for feature extraction so that a larger feature vector can be obtained. For instance, types of edges for each extracted character can be determined to produce a feature vector of a character. An edge can be defined as a 2 x 2 white to black transition [20]. According to this definition, there can be 14 different edge types which are illustrated in Figure 4.10.

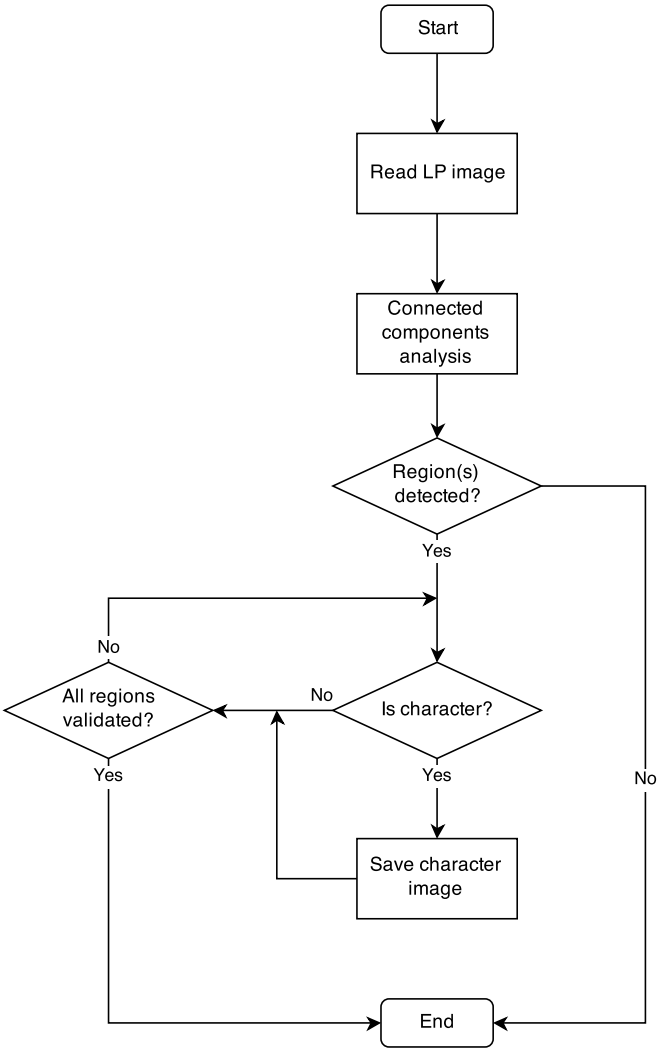


Figure 4.6: Software flow diagram of a license character segmentation.

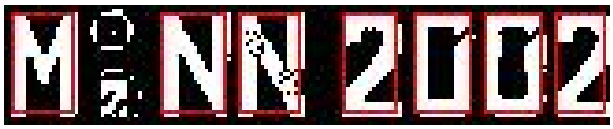


Figure 4.7: The bounding box for each character are drawn on the binary image of the license plate.

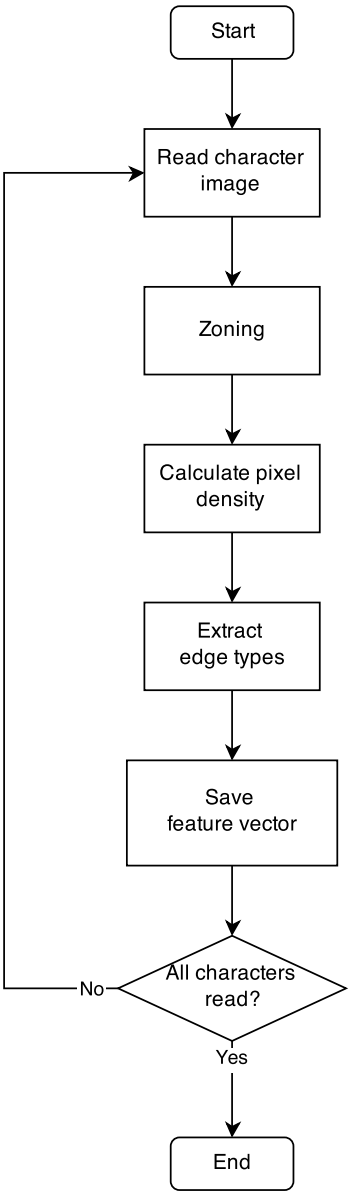


Figure 4.8: Software flow diagram for character feature extraction.

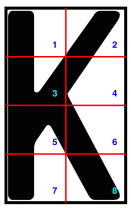


Figure 4.9: A letter "K" is divided into eight zones.

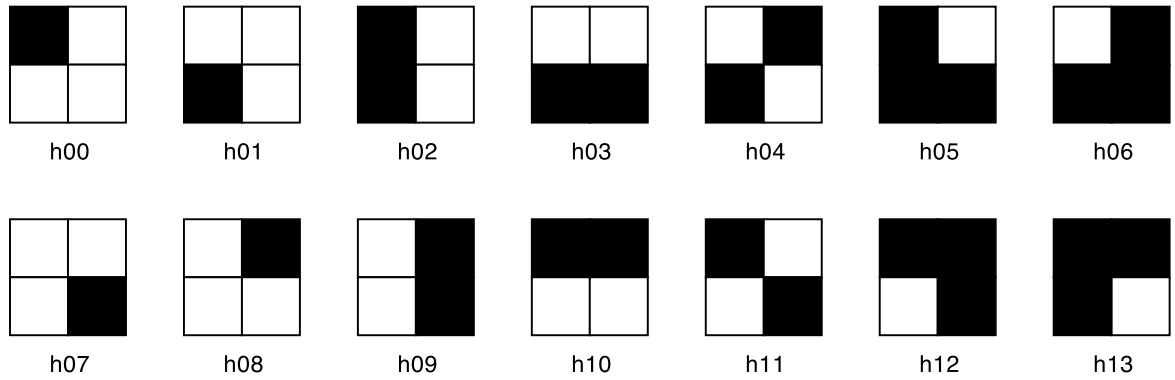


Figure 4.10: Edge types that a character image can have.

In order to reduce the vector descriptor, the similar types of edges can be combined as suggested in the following list:

1. h00 + h01 (vertical edges)
2. h02 + h03 (horizontal edges)
3. h04 + h06 + h09 ("/"-type diagonal edges)
4. h05 + h07 + h08 ("\"-type diagonal edges)
5. h10 (bottom right corner)
6. h11 (bottom left corner)
7. h12 (top right corner)
8. h13 (top left corner)

Types of character edges are determined for every zone of a character image. As the result, another feature vector of 64 elements is obtained for a character. This vector is combined with the feature vector for pixel density and this results in the total length of 72 feature elements.

4.4 Character classification

In this thesis, the vision system is developed to be only capable of offline training. The detected characters are classified with a modified k-means clustering algorithm. For each alphabetical character, at least 5 sample images are taken and a feature vector are extracted from each image. A mean vector is determined by using the feature vectors derived for a character. Each mean vector is stored into a database which is used for the classification task. There are 36 types or classes of characters that the designed vision system

must be able to be recognize. The characters and their class numbers are shown in Table 4.1. Special letters and characters are not considered for the recognition task.

1	A	19	S
2	B	20	T
3	C	21	U
4	D	22	V
5	E	23	W
6	F	24	X
7	G	25	Y
8	H	26	Z
9	I	27	0
10	J	28	1
11	K	29	2
12	L	30	3
13	M	31	4
14	N	32	5
15	O	33	6
16	P	34	7
17	Q	35	8
18	R	36	9

Table 4.1: Classes of characters that should be detected by the vision system.

The recognition is done by finding the minimum sum of absolute error. Absolute error is used instead of square error to reduce the complexity for the hardware implementation. The Equation 4.1 is used to determine the sum of error. The extracted feature vector will be subtracted with each mean vector in the database. The result is known as an error vector, where each element of the vector can be summed up to give a sum or error. Finally, the class to which a character belongs to is determined by finding the index of the minimum sum of error. Software flow diagram in Figure 4.11 describes the steps in character classification.

$$\epsilon = \sum_{i=1}^{N} \parallel x_i - \mu_i \parallel \tag{4.1}$$

Where

ϵ is the sum of absolute error
x_i is the i-th element of the feature vector.
μ_n is the i-th element of the mean vector.
N is the total number of vector elements

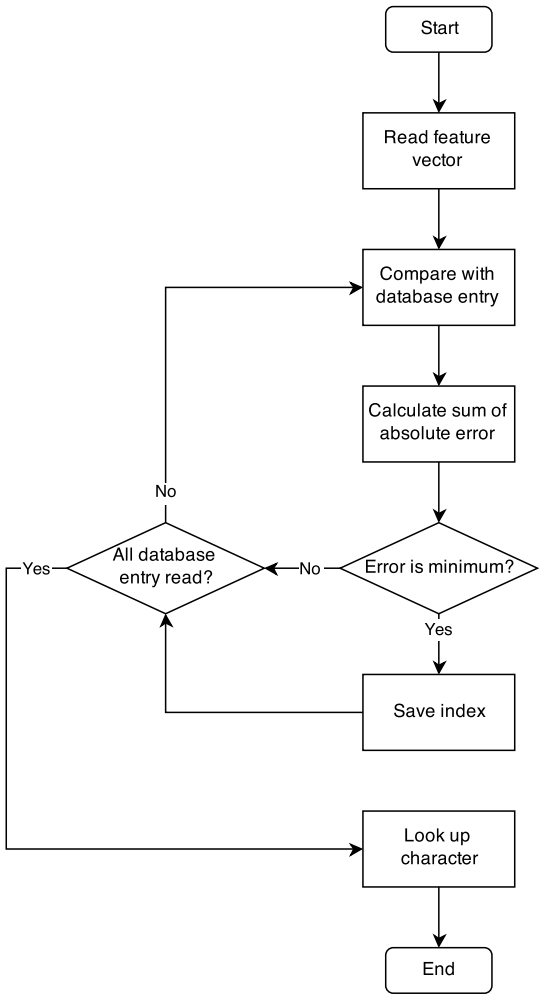


Figure 4.11: Software flow diagram for character recognition.

4.5 Summary

The methods for license plate detection and recognition have been defined step by step in this chapter. Every parts have been combined together and several tests were performed with 30 test images. It shows that the designed method achieved success rate of 90 % for the license plate detection and 83.3 % for the recognition. The same algorithm can used be for the hardware implementation. Some parts can even be improved by exploiting parallelism to accelerate the processing.

5 Implementation

The first step of the implementation is to design the hardware components for image processing algorithms. This includes grey converter, morphological filtering, thresholding, top hat, connected components labelling and bounding box. Additionally, the hardware implementation of the algorithms for optical character recognition such as zoning, pixel density and edge matching will also be explained. The components will be configured to perform the desired task, which is license plate detection and recognition. Simulation of each hardware component as well as the whole processing pipeline will be executed to examine the correctness of the constructed architecture. Xilinx ISE is the development tool that is used to implement each architecture.

5.1 Complete processing pipeline

The complete processing pipeline for the vision system is shown in Figure 5.1. It consists of two main parts, which are the detection unit and the recognition unit. The detection unit is composed of the pre-processing and also the license plate segmentation units. It receives a stream of input image in RGB format and produces a binary image of a license plate. The recognition unit receives the binary image and perform character segmentation in its first processing stage. Two license plate character images are segmented simultaneously and therefore parallel execution of character classification is possible. Instead of a character, a class number is given at the output port of the processing pipeline by the classification unit.

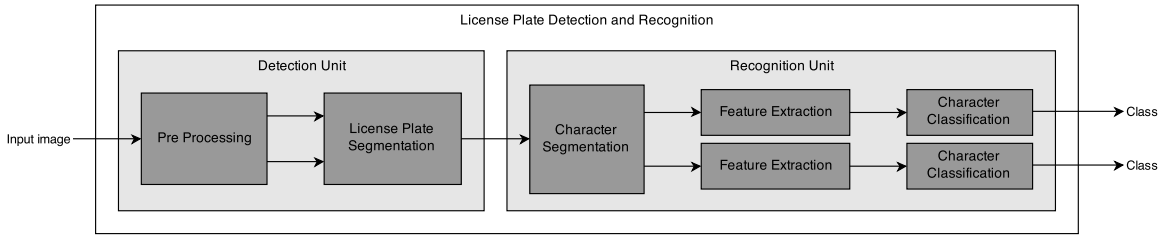


Figure 5.1: Processing pipeline that performs the algorithm for license plate detection and recognition.

5.2 Custom hardware for object detection and recognition

The subcomponents of the two main units in the processing pipeline are explained in the following subsections. These components are constructed by image processing blocks that will be presented in the Section 5.3.

5.2.1 Image pre-processing

Figure 5.2 shows the hardware configuration for image pre processing. The grey converter prepares a grey scale image to the black top hat unit by converting the colour space of the RGB image. The background cleaning and thresholding of the resulting image of black top hat morphology are executed in parallel. Therefore, this processing unit will produce two different output images similar to that are shown in Figure 4.4d and Figure 4.5c in the previous chapter.

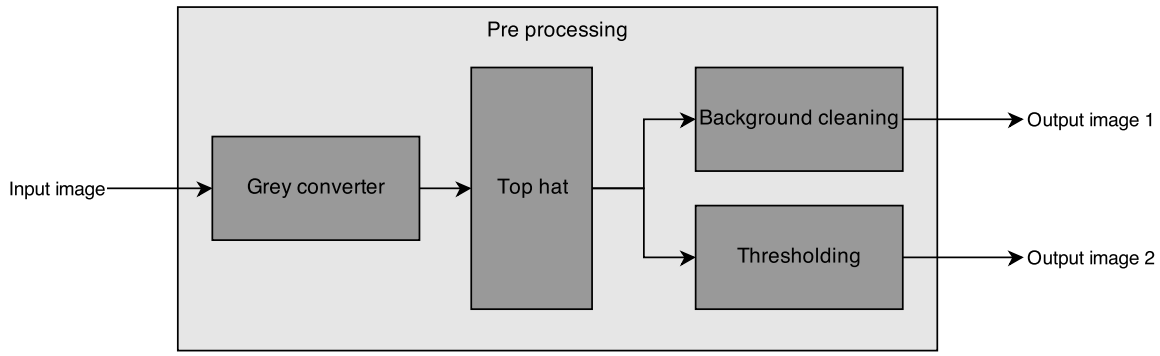


Figure 5.2: Image pre processing unit receives an RGB image and produces two different binary images.

The background cleaning consists of two morphological operations, which are opening and closing. The morphological closing is done with a horizontal filter, which has a structuring element of 1 x 45, whereas the opening is done with a normal rectangular structuring element of 15 x 25. Additionally, it also contains a global thresholding unit and a binary morphology, which is used to dilate the resulting binary image from the previous processing stages. The processing pipeline for the background cleaning is shown in Figure 5.3.

5.2.2 License plate and character segmentation

The license plate segmentation module receives two different binary images which are produced by the pre-processing module. An image as shown in Figure 4.4d must be stored temporarily in a frame buffer. Another binary image as shown in Figure 4.5c will

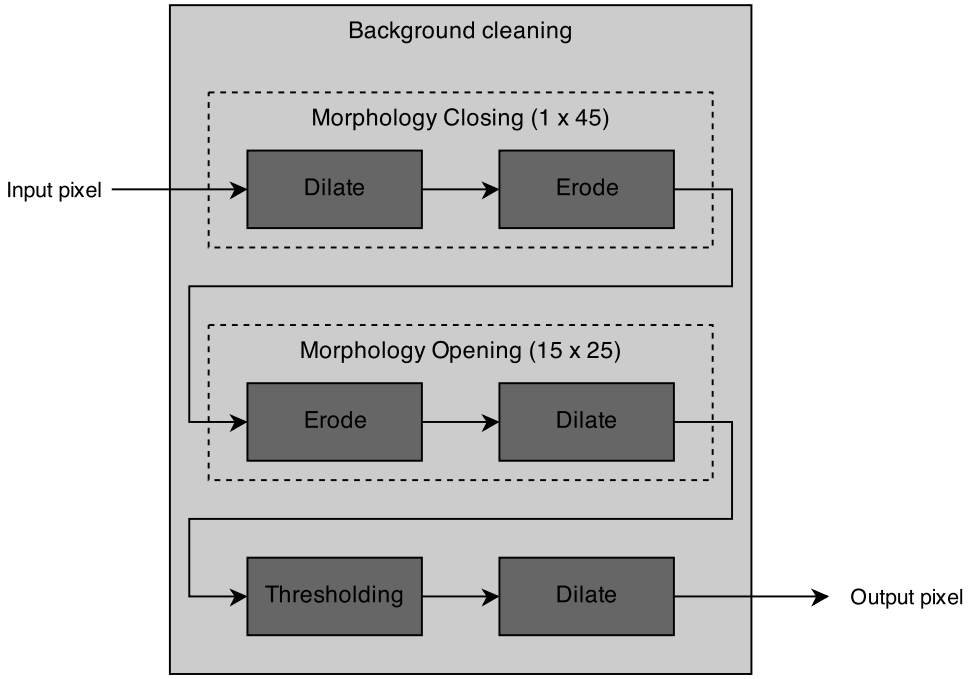
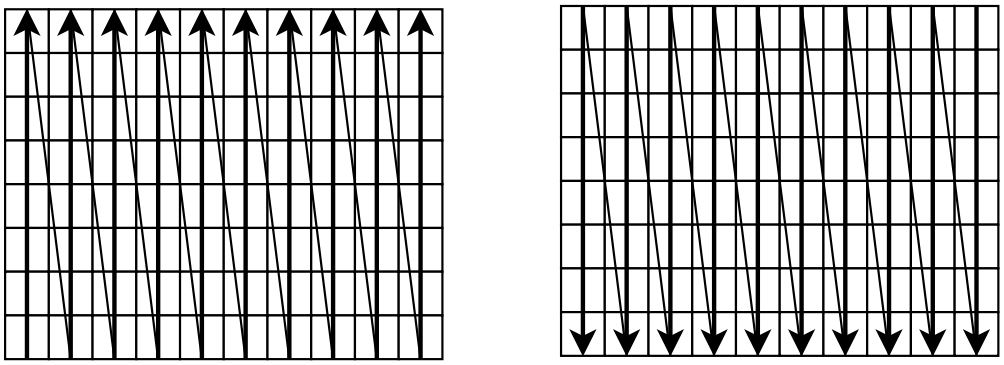


Figure 5.3: Processing pipeline for the background cleaning.

be passed to the connected components analysis to detect any region that represents the license plate. The top left and bottom right coordinate of each object is stored in the bounding box. When the whole image is labelled, the coordinates of each object will be accessed by another module. Based on the coordinate information, this module can determine the width and height of the detected region and thus will validate if a region corresponds to a license plate. The coordinate will be translated to a memory address, and it will be used to read the binary image in the frame buffer.

After the license plate is read from the frame buffer, it is forwarded to the character segmentation module. The license plate is rotated clockwise by 90 degrees to ensure that the first character will always have the least label. In other words, the character labelling must be left aligned. This is important for the license plate recognition so that the characters will not be out of order. Rotating is simply done by accessing the memory in a defined pattern (Figure 5.4a) during read operation. Figure 5.16 shows the basic implementation for the license plate and character segmentation. Unlike license plate segmentation, the character segmentation contains two instances of bounding box and blob read components so that it is capable to read two character images at the same time to speed up the recognition process. Since the image of the license plate is rotated, the character reading operation is done as shown in Figure 5.4b so that the orientation of the character image is corrected.



(a) (b)

Figure 5.4: (a) License plate image is read from bottom to top. (b) Character image is read from top to bottom

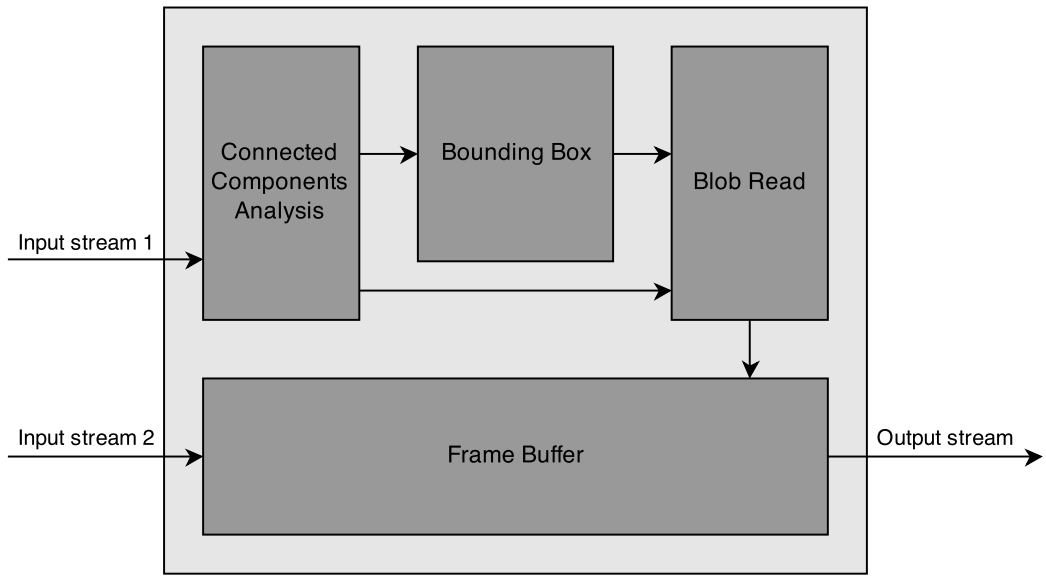


Figure 5.5: Basic implementation for license plate and character segmentation.

5.2.3 Feature extraction

As explained in section 5.3.7, a character image is divided into eight zones. Therefore, feature extraction is done for each zone instead of the whole image. The hardware architecture for feature extraction of a character image is implemented according to Figure 5.6. The zone selector acts like a demultiplexer that receives a zone number from the zoning module and enables a corresponding zone. Even though each zone will receive the pixel stream of a character image, it will not be processed as long as the zone is not enabled. Each zone will produce a feature vector that contains 9 elements. Therefore, the architecture will produces a total number of 72 feature elements for a character.

5.2.4 Classification of license plate character

The classification algorithm requires a database that stores a mean vector for each license plate character. The component shown in Figure 5.7 stores a mean vector of one out of 36 characters. Therefore, the classification unit must include 36 units of the component. An absolute error between a feature vector and each mean vector must be determined. This is done by doing element-wise subtraction of the feature vectors. Absolute error is chosen instead of square error to avoid the usage of multiplier that may increase the computational complexity and hence reducing the computation speed. In this way, more logic resources can also be reserved. When absolute error of each feature element is obtained, the sum of absolute error is calculated by simply adding up all the elements in the error vector.

Pairwise classification algorithm is used in this case, where two errors are compared at the same time. The pairwise classification unit is implemented as shown in the following Figure 5.8. Since there are 36 classes that exist in the database, 35 classification units are instantiated and configured to form a classification tree.

5.3 Architectures for image processing algorithms

The final section of this chapter will introduce the developed architectures for image processing algorithms. In total, there are nine algorithms that have been realized in this thesis and reused to compose custom blocks for the processing pipeline.

5.3.1 Grey converter

The image data that the camera delivers to the system has an RGB format. Before it can be further processed at the next stages in the processing pipeline, the input image must be converted to a greyscale image. For each input pixel, the red, green and blue components are extracted from the 24-bit pixel data in order to calculate the grey value for the corresponding output pixel. A so called luminosity method is used here which calculates the weighted average of red, green and blue pixel [12]. The calculation is done based on Equation 5.1 which is approximated from the original equation in order to

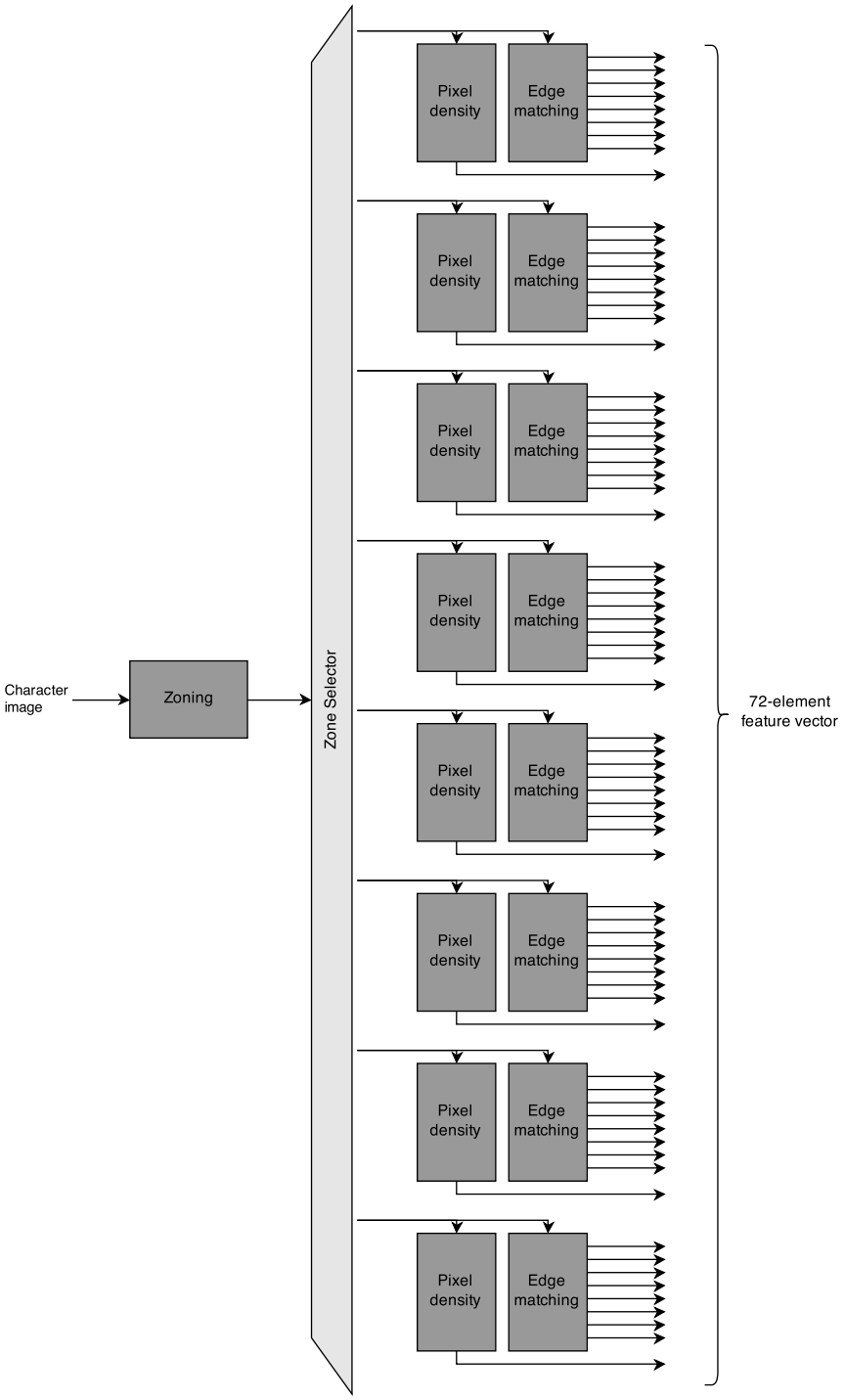


Figure 5.6: Feature extraction module.

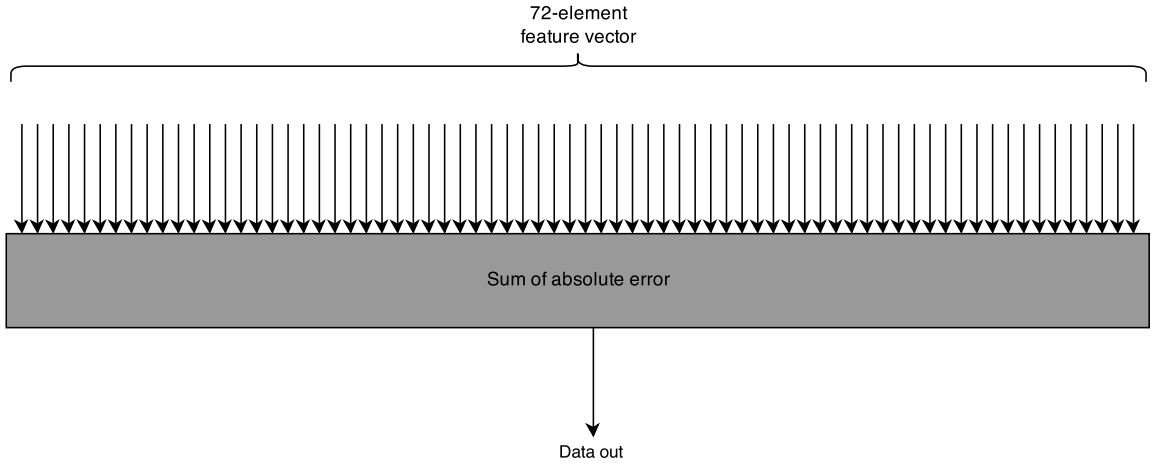


Figure 5.7: Hardware architecture that compares an input feature vector with a mean vector by subtracting each vector element and sums up all elements to produce sum of error.

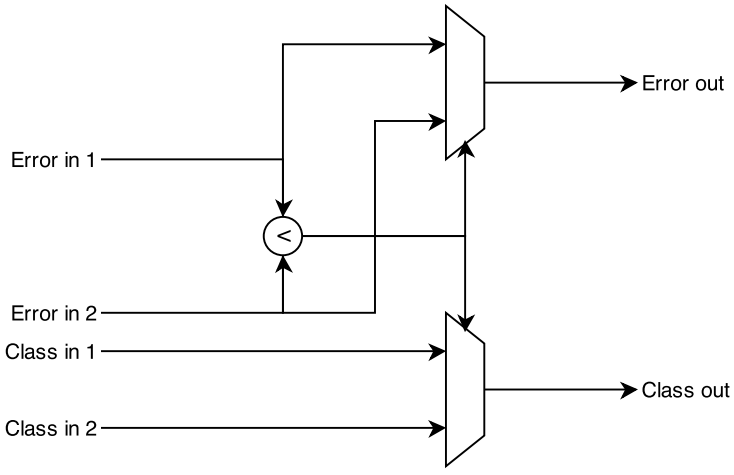


Figure 5.8: Pairwise classification unit that receives sum of errors and the corresponding class numbers.

achieve faster computation time. Instead of using multiplier, shift right logical by certain amount of bits is applied to the pixel values. The hardware implementation for the grey converter is illustrated in Figure 5.9.

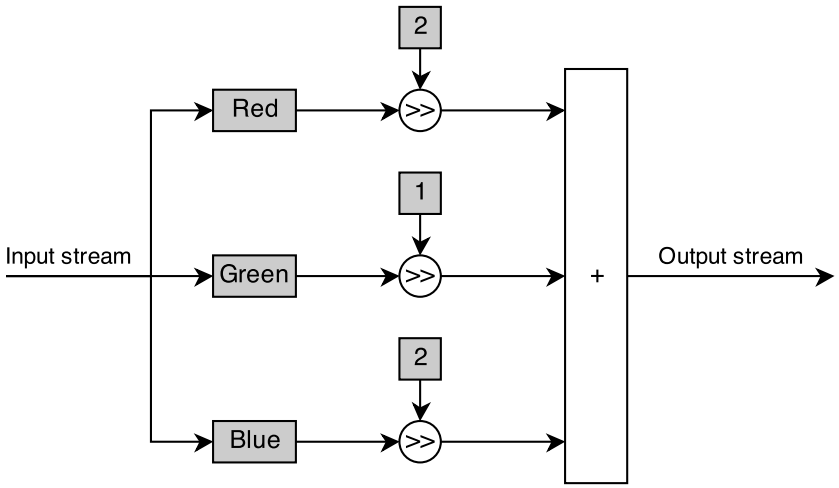


Figure 5.9: Conversion from 24 bit RGB pixel to 8 bit grey pixel.

$$\begin{aligned} p_{grey} &= 0.21 \cdot p_{red} + 0.72 \cdot p_{green} + 0.07 \cdot p_{blue} \\ &\approx 0.25 \cdot p_{red} + 0.5 \cdot p_{green} + 0.25 \cdot p_{blue} \end{aligned} \tag{5.1}$$

5.3.2 Morphological filter

The architecture for grey scale morphology is based on the design in [7]. For rectangular structuring element, efficient separable implementations are applicable as shown in Figure 5.10. The implementation of this architecture allows users to select between both morphological operation by setting a bit at the generic port. This will simply configure the comparison unit to output either minimum or maximum value. Additionally, the length of the row buffer, which is implemented using block RAM can also be adjusted via the generic port. To implement opening and closing, the proposed architecture must be duplicated and ordered accordingly.

5.3.3 Global thresholding

Thresholding in its basic form compares each pixel in the image with a threshold level and assigns the output to true or false (white or black) as shown in Figure 5.11. It can be implemented by just using a comparator. Thresholding is a point operation because each pixel is treated identically. Since the output pixel has only two variations, thresholding

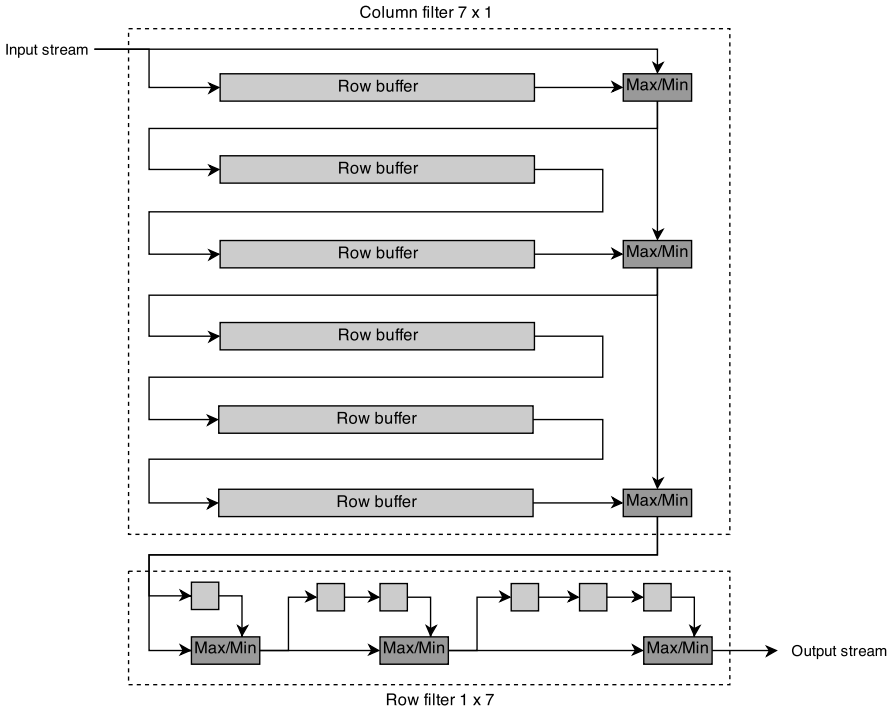


Figure 5.10: Hardware architecture of a morphological filter with rectangular structuring element 7 x 7.

is a way of converting a grey scale image to a binary image. The threshold value can be selected experimentally. It can simply be set via generic port as an integer value.

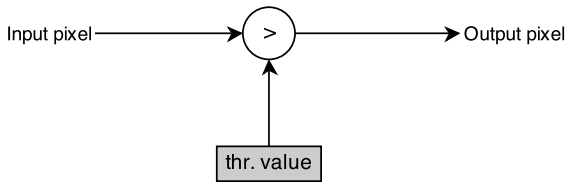


Figure 5.11: Implementation of global thresholding algorithm.

5.3.4 Top hat

A top hat transform is performed by subtracting the greyscale image and morphological image. Since it a morphological closing is used instead of opening, the algorithm implemented here is a black top hat. The architecture of a separable morphological filter with structuring element 7 x 7 as explained in section 5.3.2 is instantiated twice here. One is used as dilation and the other as erosion. The result of the grey converter is delayed by buffering them in a block RAM while morphological closing is running at the same time. The block RAM that is used for frame delay has a size of 4096 Bytes. The pixel

subtraction is executed as soon as the result of morphological processing is available. The architecture for the FPGA implementation of the top hat transform is shown in Figure 5.12.

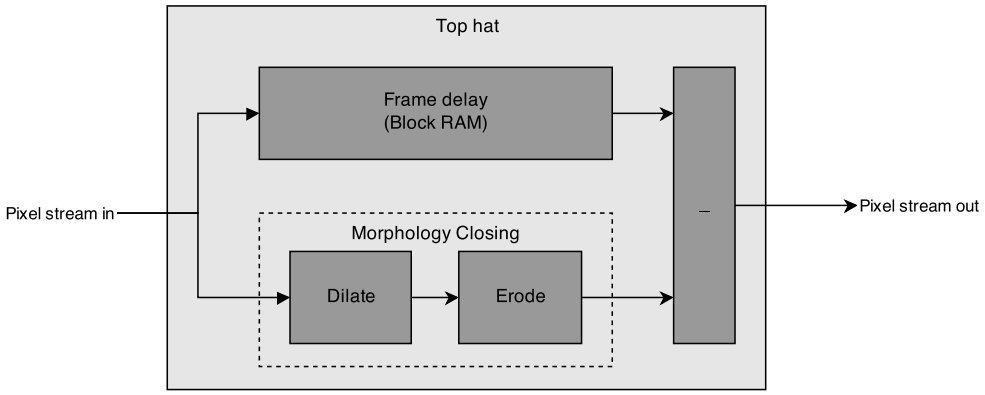


Figure 5.12: Implementation of top hat morphological filter.

5.3.5 Connected component labelling

A modification has been introduced that enables a single pass algorithm that eliminates the need of frame buffering and significantly reducing the latency, making it ideally suited for processing streamed images on an FPGA. A single pass algorithm extracts the features of interest for each component while determining the connectivity. This removes the need of producing a labelled image and avoids the second relabelling pass. However it requires merging and relabelling on the fly to ensure that consistent results are obtained.

The single pass connected components labelling is shown in Figure 5.13. The neighbourhood context provides the labels of the adjacent pixels to the current pixel. The neighbourhood pixel labels are stored in registers A, B and C, and these are shifted along each clock cycle as the window is scanned across the image. Unlike the architectures proposed in [13] and [9], the neighbourhood pixel to the left of the currently processed pixel is stored internally in the labelling unit. A row buffer caches the labels from the previous row. These must be looked up in the merger table to correct the label. The row buffer must be read in advanced to initialize the register B and C in the neighbourhood context before a new row is streamed [7].

Label selection assigns the label for the current pixel based on the labels of its neighbour. It follows the classic two pass algorithm. The background pixels are always labelled 0. If all the neighbouring pixels are background, a new label is assigned to the input pixel. If only a single label appears among the labelled neighbours, that label is selected for the input pixel. Otherwise, if there are two distinct labels, those regions are merged and the smaller of the two labels is retained and selected for the input pixel [7].

The merger table is used as a look-up table on the output of the row buffer. This ensures that the correct label is used for any labels that have been stored in the row buffer which have subsequently been merged. When a new label is created a new entry is added to the merger table pointing to itself. This avoids the need for initialising the merger table prior to processing. To keep the merger table up to date, whenever a merger occurs, the label that is replaced should subsequently point to the merged label [7].

The merger control block updates the merger table when two objects are merged. The larger label is modified to point to the smaller label used to represent the region. When a series of mergers occur with the smaller label on the right, it is possible for a chain of mergers to result. Such mergers are pushed onto a chain stack. These chains are then resolved during the horizontal blanking period by visiting them in reverse order, updating the merger table so that each of the old labels points to the smallest label [7].

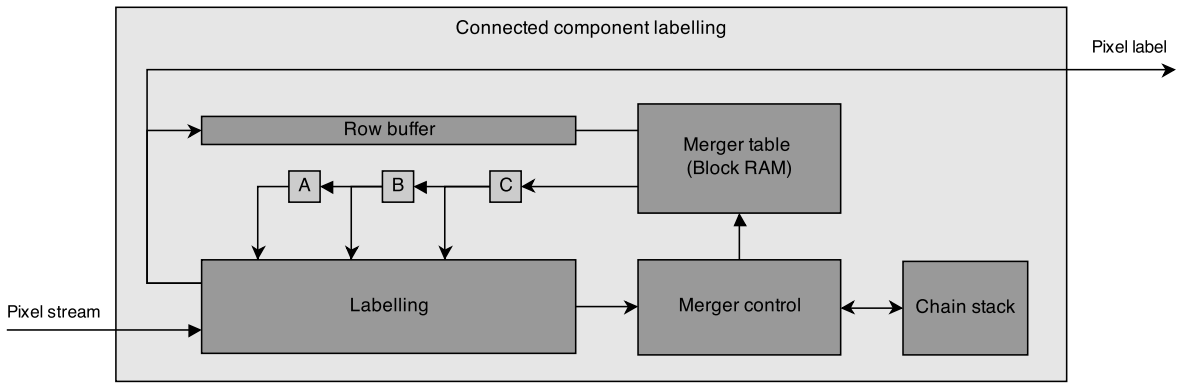


Figure 5.13: Hardware architecture for connected component labelling.

5.3.6 Bounding box

Figure shows the implementation of the bounding box processor for a binary image that contains multiple label. In the frame blanking period, init is set to one. It provides initialisation when the first pixel of an object is detected by forcing the coordinate of that pixel to be loaded into xmin, xmax, ymin and ymax. Since the image is scanned in raster order, the first pixel detected will provide the ymin value. The y value of the following object pixel is clocked into the ymax register because the last pixel detected in the frame will have the largest y. After the first pixel is detected, the current x coordinate is compared with xmin and xmax. At the end of the frame, the four registers indicate the extent of the object pixels within the image [7].

The block RAMs for xmin and xmax need to be dual port because the values need to be

read for comparison. Since there is typically only a small number of labels, using fabric RAM gives a resource efficient implementation [7].

Compared to other segmentation algorithms such as watershed and Hough transform, a simple bounding box provides low processing and computation cost which makes it more preferred for hardware implementation. If the noise can be successfully filtered, it provides an efficient method of extracting some basic object parameters [7].

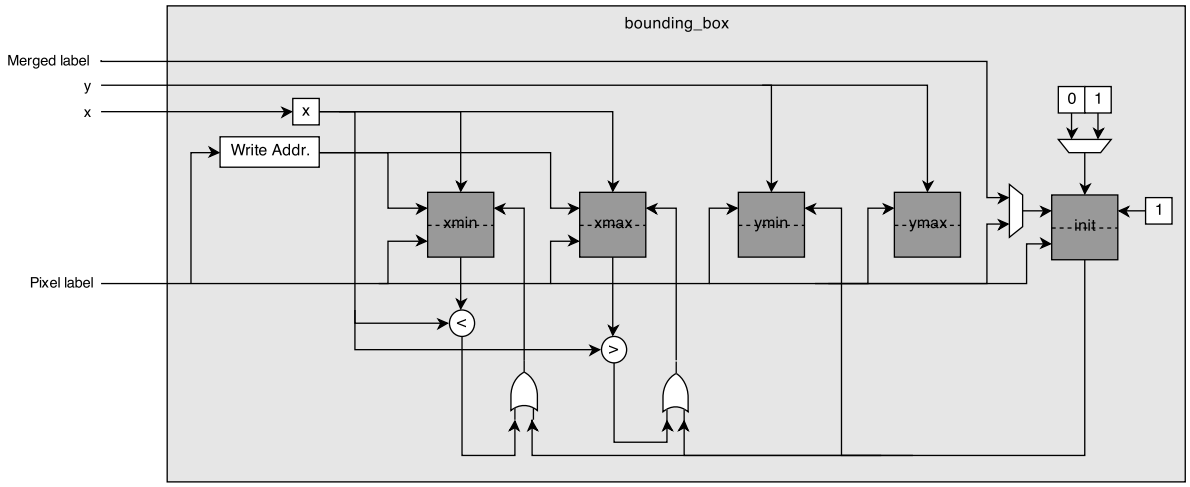


Figure 5.14: Implementation of bounding box algorithm.

5.3.7 Zoning

The first step to perform feature extraction for each license plate character is to divide a character image into several sections or zones. In this project, each segmented character is divided into eight zones according to Figure 4.9. Zoning of a character image is possible when its width and length are known. These parameters are calculated during the segmentation of a character using bounding box. The borders that define the zones can be calculated by dividing the width and height by 2 and 4 respectively. In total, there are four borders that will be calculated by this unit.

5.3.8 Pixel density

By assuming that all images have the same dimension, the pixel density can simply be determined by summing all the foreground pixel. This can simply be implemented with a counter on a hardware. The density value is stored in a register and will be kept until a new image has arrived. Then the register will be reset to store a density value for a new image.

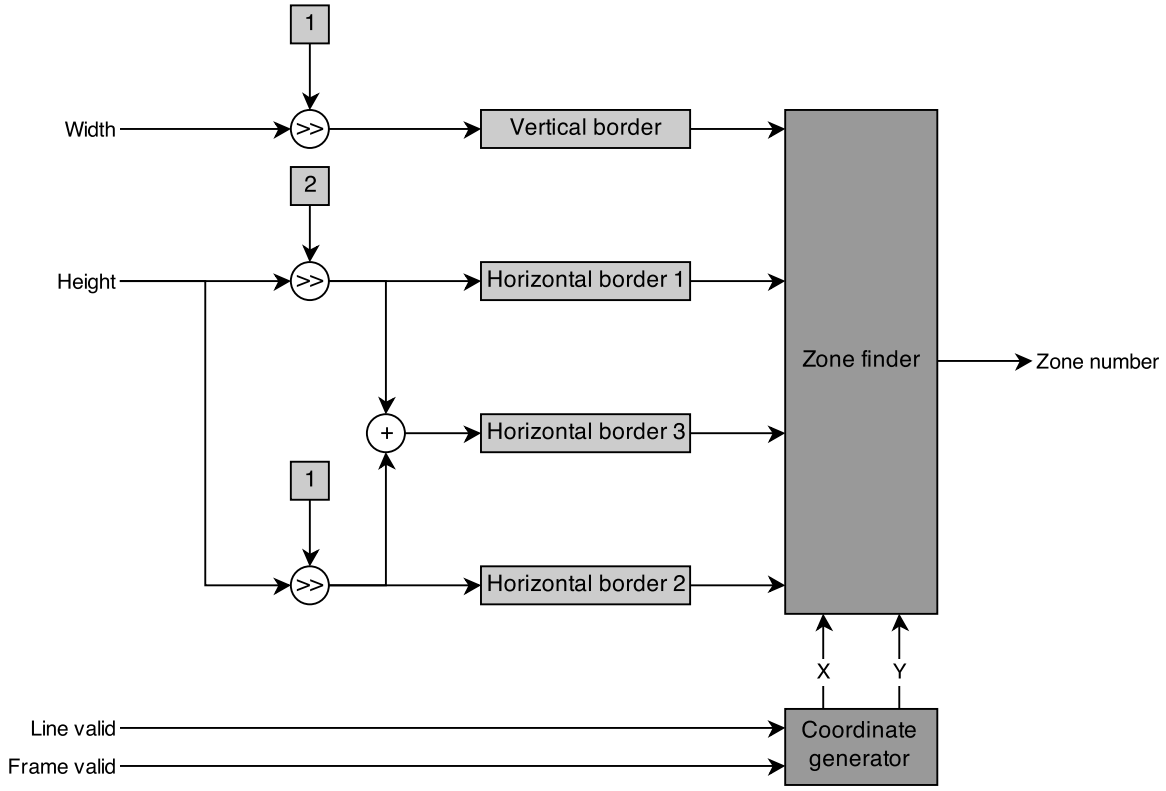


Figure 5.15: Hardware architecture to assign a group of pixel of a character image to a certain zone.

5.3.9 Edge matching

As explained in Section 4.3, there are 14 types of edges that an image can have. Each type is implemented as shown Figure 5.16. The coefficient of each field is initialized in registers c1 to c4 via the generic ports. The row buffer is used to store the previous row of the image. The result of the comparison will switch the multiplexer that selects the operand for the adder. If an edge match occurs, the register that stores the number of detected edges will increment. It will be reset when a new character image is received.

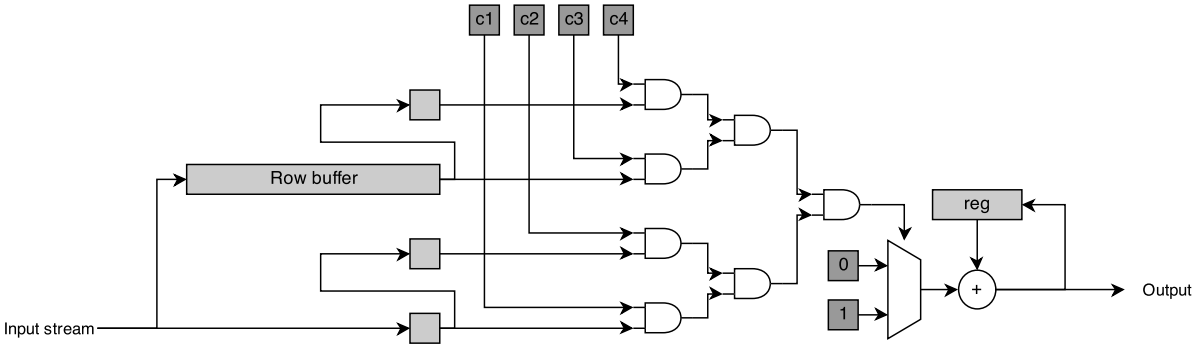


Figure 5.16: Implementation of edge matching.

5.4 Summary

This chapter has presented all architectures that are used in order to perform license plate detection and recognition. It began by describing the top level block diagram and the custom blocks to perform some functions. The architectures that are described in the final section are standard and defined algorithms for image processing and computer vision. These architectures should also be able to be reused for other applications.

6 Analysis

The implemented processing pipeline is tested with several different images which are stored as bitmap files. These image are collected from several different sources. A test bench that is created exclusively for the processing pipeline will read an image and send it to the processing pipeline to be processed. Continuous sending of the same image simulates a video processing. The sending rate is equivalent to a video frame rate and it can be adjusted in the test bench. Figure 6.1 illustrates the configuration to perform the test. At one end, pixel data of an input RGB image is passed to the input port. The pipeline will process them and produce the output image of certain processing stage. These images will be written as a bitmap file by the test bench in a different location for viewing purpose. The license plate characters are given out at the other end of the pipeline.

As explained in Section 5.2.4, the database for the character recognition is stored in the registers in the processing pipeline instead of a block RAM. Before the test can begin, the database must be updated. A set of images is used to train the classifier in the processing pipeline. The output from the feature extraction unit is recorded, which will be used to calculate the mean vector of each character. When the database is updated, another different set of test images will be used to measure the success rate of detection and recognition. At least 30 images are used for the training phase and the test.

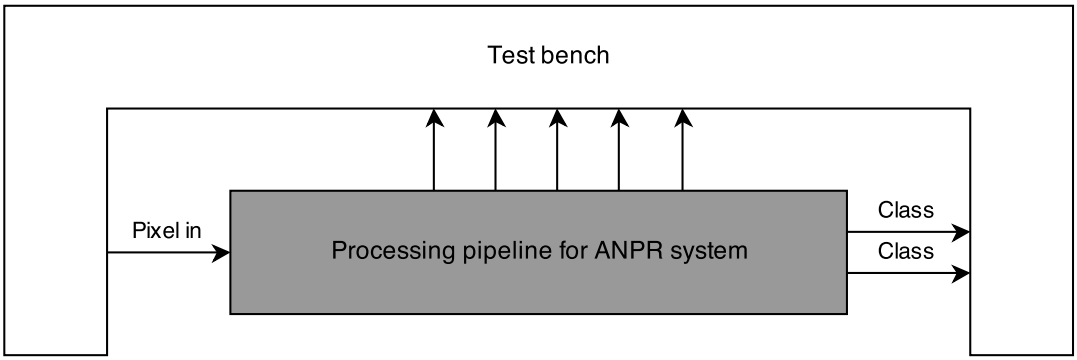


Figure 6.1: Testing the processing pipeline.

6.1 Detection and recognition result

Figure 6.2 shows one of the input images that is fed by the test bench to the processing pipeline. If the FPGA is set to operate at maximum clock frequency to process an image with a VGA resolution, license plate detection can be accomplished in less than 10 ms. This should satisfy real-time processing requirement of any ANPR application, especially for the detection of fast moving cars on a highway. The processing pipeline manages to achieve a plate detection successful rate of 90 % and recognition rate of 83 %. Complete test results are indicated in Table 6.1. The pre-processing stages (morphology operations plus the connected component analysis) consumes 97% of the total time required by the system to process a single frame.

Figure 6.3, 6.4, 6.5, 6.6 and 6.7 provide sample output images of the processing pipeline for the detection process, which is passed to the character recognition unit for further processing.

Figure 6.2: Test input image.

Processing stage	Execution time (ms)	Success rate (%)
Pre-processing	6.012	N/A
License plate detection	0.12	90
Character segmentation	0.0203	N/A
Character recognition	0.0204	83.3

Table 6.1: Performance and accuracy results

Figure 6.3: Output of the grey converter.

Figure 6.4: Output of the top hat morphology.

Figure 6.5: First output image of the pre-processing stage

Figure 6.6: Second output image of the pre-processing stage.

Figure 6.7: Result of the license plate detection.

6.2 System performance

A detail logic utilization is listed in Table 6.2. The processing pipeline has a maximum operating frequency of 57.823 MHz and minimum clock period of 17.294 ns. Multipliers are only used for translating pixel coordinate in cartesian format into memory address to read the license plate image as well as the character image from block RAMs. Since one license plate image and two character images are read at a time, only three multipliers are required in the whole processing pipeline. Block RAM is mostly used as row and frame buffers for morphological filtering, license plate and character image segmentation. The result also shows that the developed processing pipeline fits into the device and there are still available resources that can be utilized for future development.

Device utilization summary			(xc7z020-1-clg484)	
Slice Logic Utilization				
Number of Slice Registers:	8456	out of	106400	7%
Number of Slice LUTs:	33975	out of	53200	63%
Number used as Logic:	33887	out of	53200	63%
Number used as Memory:	88	out of	17400	0%
Number used as RAM:	88			
Specific Feature Utilization				
Number of Block RAM/FIFO:	127	out of	140	90%
Number using Block RAM only:	127			
Number of BUFG/BUFGCTRL/BUFHCEs	4	out of	104	3%
Number of DSP48E1s:	3	out of	220	1%
Primitive and Black Box Usage				
		RAMS		191
			RAM128X1D	6
			RAM32M	4
			RAM32X1D	24
			RAMB18E1	60
			RAMB36E1	97

Table 6.2: Device utilization summary of the processing pipeline.

The chart in Figure 6.8 summarizes the resource allocation to every stage of the processing pipeline. The classifier takes almost half of the on-chip resources due to the large adder trees used to calculate the sum of absolute errors. Other components that mainly uses block RAM for their task such as plate detection and character segmentation requires only 2 %.

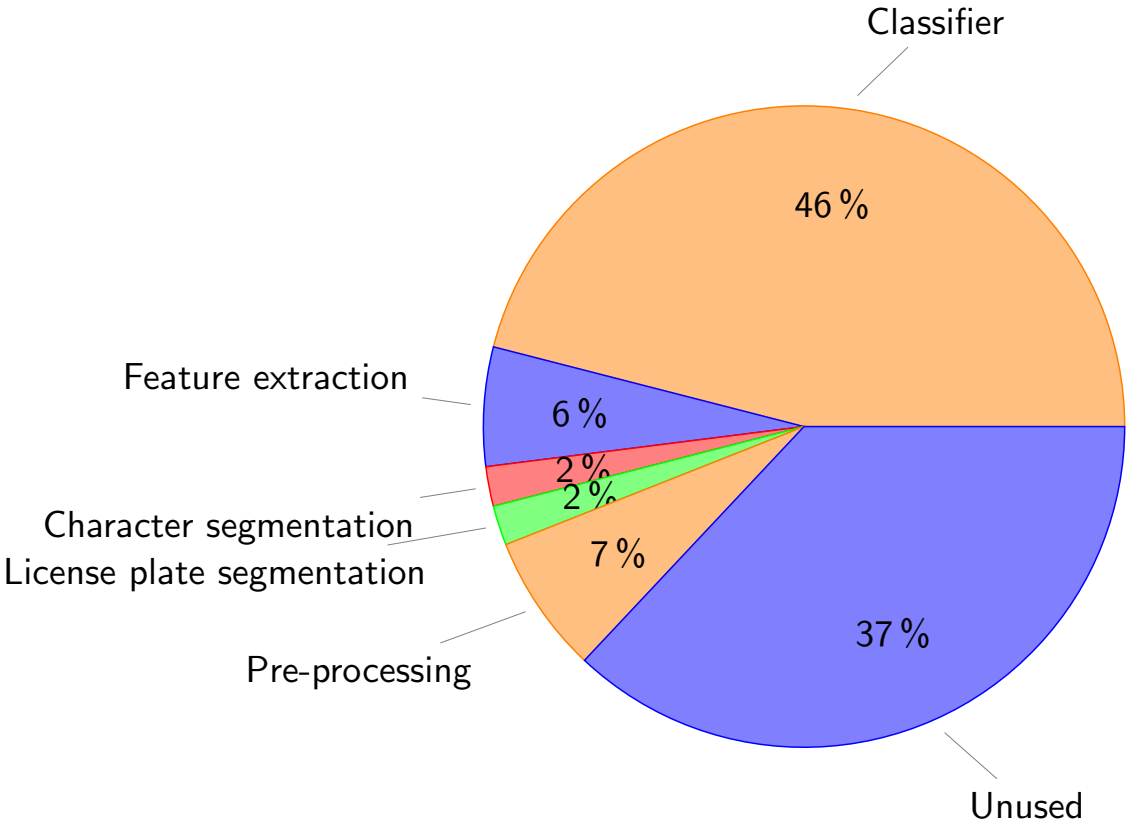


Figure 6.8: Resource allocation for each processing stage.

7 Conclusion

A lot of related researches have been discussed throughout this work to find a solution for a flexible plate detection and recognition system that can be deployed in various applications with minimal reconfiguration. Studies have shown that the performance and success rate depend on the installed imaging equipment. Better results can be achieved by using sophisticated capturing device with additional sensors that enhance the image and video quality.

In this thesis, the processing pipeline for the license plate detection and recognition system has been designed and implemented on an FPGA. It consists of two main parts which are assigned for license plate localization and license plate character recognition respectively. The plate detection part process the input image by converting it to grey scale image. Top hat morphology converts the image into a binary image that contains candidate regions of the license plate. The license plate segmentation unit applies a criteria that will validate each region to check if it covers the license plate area.

Most of the on-chip resources are allocated to the license plate recognition part to make it capable of processing multiple characters in parallel. At the moment, the character segmentation unit can read two characters of the license plate at the same time. Feature extraction and classification of both character can also be executed in parallel. The pipeline has been made highly reconfigurable so that when it is ported to other FPGA device that offers more logic resource, it can be configured to process more characters in parallel.

According on the test results, the morphological approach is proven to be very effective for the license plate detection task. The classification with k-means clustering also proves to be reliable compared to support vector machine and neural networks. In fact, less hardware resources makes it possible to duplicate the classifier unit to perform parallel recognition. Parallel execution of recognition reduces the computation time.

As there are still more logic resources available on the chip, the processing pipeline can be customized to improve the flexibility of the vision system. To make the system work with multiple resolution, image resizing algorithm can be implemented and integrated to the processing pipeline. Hough transform which is implemented in [5] will add the ability of the processing pipeline to detect and recognize a rotated license plate, thus the success rate of detection and recognition will be improved.

8 Future Work

The developed processing pipeline is already sythesized and ready to be used to build a prototype of an ANPR system. ZedBoard [4] is a development kit that will serve this purpose. It is equipped with Xilinx Zynq-7000 (Z-7020), which is the selected device for the synthesis of the developed processing pipeline.

In order to make the prototype works like the real ANPR system, a camera has to be attached to the ZedBoard as a capturing device. Since there is no input port available for video signal on the ZedBoard, one alternative is to connect the board and a personal laptop with an ethernet cable, provided that the computer has a built-in webcam which will act as a capturing device for the ANPR prototype.

The Z-7020 integrates a dual-core ARM Cortex-A9 based processing system (PS) and Xilinx programmable logic (PL) in a single device. The two domains are interconnected via AXI bus. The developed processing pipeline will be integrated in the PL and connected to the AXI bus over which the pixel data is sent and received. Furthermore, the processing system must be configured to receive packets of data from the ethernet interface. Then, the packet that contains the pixel data can either be buffered in the internal block RAM or the external DDR memory that the ZedBoard is equipped with until the whole image is received.

To view the results of the detection unit, the image of the license plate can be sent back to the computer over the ethernet cable, whereas the recognition results can simply be sent via serial communication interface so that it can be viewed with a command terminal or similar program.

ANPR system that is built on FPGA will create higher demand since the overall performance has been proven to be better than a software based solution. Research and development in this field and related area of studies will continue to move forward to improve the existing ANPR system.

Bibliography

[1] Atalaya. http://www.imagsa.com/web/eng/atalaya.php.

[2] Automatic number plate recognition (anpr) calculator for mobotix cameras. https://www.anpronline.net/calculator.html.

[3] Data clustering algorithms. https://sites.google.com/site/dataclusteringalgorithms/k-means-clustering-algorithm.

[4] Zedboard.org. "http://zedboard.org/product/zedboard".

[5] Medhat Moussa. Ahmed Elhossini. A reconfigurable architecture for real-time vision systems on fpga. In *22nd International Conference on Microelectronics (ICM 2010)*, 2010.

[6] Alexandr A. Motyko. Alexandr A. Kryachko, Boris S. Timofeev. The algorithm for cars license plate segmentation. In Yevgeni Koucheryavy Sergey Balandin, Sergey Andreev, editor, *Internet of Things, Smart Spaces, and Next Generation Networks and Systems.* Springer, 2014.

[7] Donald G. Bailey. *Design for Embedded Image Processing on FPGAs.* John Wiley Sons (Asia) Pte Ltd, 2011.

[8] Kuo-Ming Hung Hsieh-Chang Huang. Ching-Tang Hsieh, Liang-Chun Chang. A real-time mobile vehicle license plate detection and recognition for vehicle monitoring and management. In *Pervasive Computing (JCPC), 2009 Joint Conferences on*, pages 197–202, Dec 2009.

[9] Donald G. Baily Christopher T. Johnston. Fpga implementation of a single pass connected components algorithm. In *4th IEEE International Symposium on Electronic Design, Test Applications*, 2008.

[10] Horst Bischof Clemens Arth, Florian Limberger. Real-time license plate recognition on an embedded dsp-platform.

[11] Mike Constant. An introduction to anpr. http://www.cctv-information.co.uk/i/An_Introduction_to_ANPR.

[12] John D. Cook. Three algorithms for converting color to grayscale. http://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/.

[13] C.T. Johnston D.G. Bailey. Single pass connected components analysis. 2007.

[14] Michael Grimm. Camera-based driver assistance systems. *Advanced Optical Technologies*, 2013.

[15] Cristian Grozea, Zorana Bankovic, and Pavel Laskov. Fpga vs. multi-core cpus vs. gpus: Hands-on experience with a sorting application. In Rainer Keller, David Kramer, and Jan-Philipp Weiss, editors, *Facing the Multicore-Challenge*, volume 6310 of *Lecture Notes in Computer Science*, pages 105–117. Springer Berlin Heidelberg, 2010.

[16] Marcin Iwanowski. *Automatic car number plate detection using morphological image processing.* PhD thesis, Warsaw University of Technology, Institute of Control and Industrial Electronics EC Joint Research Centre, Institute of Environment and Sustainability.

[17] Rajneesh Rani. Kartar Singh Siddharth, Renu Dhir. Comparative recognition of handwritten gurmukhi numerals using different feature sets and classifiers. *International Conference*

on Recent Advances and Future Trends in Information Technology (iRAFIT2012) Proceedings published in International Journal of Computer Applications® (IJCA), 2012.

[18] Anders Kjær-Nielsen. *Real-time Vision using FPGAs, GPUs and Multi-core CPUs.* PhD thesis, University of Southern Denmark, 2010.

[19] O. Sharifi Tehrania-P. Moallem M. Ashourian, N. Daneshmandpoura. Real time implementation of a license plate location recognition system based on adaptive morphology. *International Journal of Engineering*, 2013.

[20] Ondrej Martinsky. Algorithmic and mathematical principles of automatic number plate recognition systems. Master's thesis, Brno University of Technology, 2007.

[21] Francesca Odone. *Experiments on a License Plate Recognition System.* PhD thesis, DISI, Università degli Studi di Genova.

[22] Vladimir Shapiro, Georgi Gluhchev, and Dimo Dimov. Towards a multinational car license plate recognition system. *Machine Vision and Applications*, 17(3):173–183, 2006.

[23] Jeremy W. Sheaffer-Kevin Skadron Shuai Che, Jie Li and John Lach. Accelerating compute-intensive applications with gpus and fpgas.

[24] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing.* California Technical Publishing.

[25] M. Arai T. Kanamori, H. Amano and Y. Ajioka. A high speed license plate recognition system on an fpga. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 554–557, Aug 2007.

[26] X. Zhai, F. Bensaali, and S. Ramalingam. Real-time license plate localisation on fpga. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 14–19, June 2011.

[27] Chongliang Zhong, Yalin Ding, and Jinbao Fu. Handwritten character recognition based on 13-point feature of skeleton and self-organizing competition network. In *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on*, volume 2, pages 414–417, May 2010.

[28] Faycal Bensaali Reza Sotudeh Zoe Jeffrey, XiaojunZhai and Aladdin Ariyaeeinia. Automatic number plate recognition system on an arm-dsp and fpga heterogeneous soc platforms.