Automatic Number Plate Recognition on FPGA

Xiaojun Zhai

A thesis submitted in partial fulfilment

of the requirements of the University of Hertfordshire

for the degree of Doctor of Philosophy

The programme of research was carried out in the Science and

Technology Research Institute (STRI), University of Hertfordshire,

United Kingdom.

January 2013

Abstract

Intelligent Transportation Systems (ITSs) play an important role in modern traffic management, which can be divided into intelligent infrastructure systems and intelligent vehicle systems. Automatic Number Plate Recognition systems (ANPRs) are one of infrastructure systems that allow users to track, identify and monitor moving vehicles by automatically extracting their number plates. ANPR is a well proven technology that is widely used throughout the world by both public and commercial organisations. There are a wide variety of commercial uses for the technology that include automatic congestion charge systems, access control and tracing of stolen cars. The fundamental requirements of an ANPR system are image capture using an ANPR camera and processing of the captured image. The image processing part, which is a computationally intensive task, includes three stages: Number Plate Localisation (NPL), Character Segmentation (CS) and Optical Character Recognition (OCR). The common hardware choice for its implementation is often high performance workstations. However, the cost, compactness and power issues that come with these solutions motivate the search for other platforms. Recent improvements in low-power high-performance Field Programmable Gate Arrays (FPGAs) and Digital Signal Processors (DSPs) for image processing have motivated researchers to consider them as a low cost solution for accelerating such computationally intensive tasks. Current ANPR systems generally use a separate camera and a stand-alone computer for processing. By optimising the ANPR algorithms to take specific advantages of technical features and innovations available within new FPGAs, such as low power consumption, development time, and vast on-chip resources, it will be possible to replace the high performance roadside computers with small in-camera dedicated platforms. In spite of this, costs associated with the computational resources required for complex algorithms together with limited memory have hindered the development of embedded vision platforms.

The work described in this thesis is concerned with the development of a range of image processing algorithms for NPL, CS and OCR and corresponding FPGA architectures. MATLAB implementations have been used as a proof of concept for the proposed algorithms prior to the hardware implementation. The proposed architectures are speed/area efficient architectures, which have been implemented and verified using the Mentor Graphics RC240 FPGA development board equipped with a 4M Gates Xilinx Virtex-4 LX40. The proposed NPL architecture can localise a number plate in 4.7 *ms* whilst achieving a 97.8% localisation rate and consuming only 33% of the available area of the Virtex-4 FPGA. The proposed CS architecture can segment the characters within a NP image in 0.2-1.4 *ms* with 97.7% successful segmentation rate and consumes only 11% of the Virtex-4 FPGA on-chip resources. The proposed OCR architecture can recognise a character in 0.7 *ms* with 97.3% successful recognition rate and consumes only 23% of the Virtex-4 FPGA available area. In addition to the three main stages, two pre-processing stages which consist of image binarisation, rotation and resizing are also proposed to link these stages together. These stages consume 9% of the available FPGA on-chip resources. The overall results achieved show that the entire ANPR system can be implemented on a single FPGA that can be placed within an ANPR camera housing to create a stand-alone unit. As the benefits of this are drastically improve energy efficiency and removing the need for the installation and cabling costs associated with bulky PCs situated in expensive, cooled, waterproof roadside cabinets.

Certificate of Originality

I hereby certify that the work presented in this report is my original research and has not been presented for a higher degree at any other university or institute.

Signed: [illegible] Dated: 16/06/2013

(Xiaojun Zhai)

Acknowledgments

My first thanks go to my principle supervisor, Dr. Faycal Bensaali for providing his professionalism, caring and friendly support throughout my research.

I also very appreciate the guidance and help provided by my second supervisor Dr. Soodamani Ramalingam.

I would like also to thank my family for their financial support during the research, without them I cannot undertake this research.

I would also like to thank my wife for fully supporting my research, without her support I cannot reach this achievement.

Special thanks Prof. Yichuang Sun and Dr. Lily Meng for their valuable discussions and suggestions.

This is also a great opportunity to thank Prof. Reza Sotudeh for his financial support and allowing me to attend the conference and training.

Many thanks to all academic and technical staff of the School of Engineering and Technology, especially Dr. George Pissanidis, Mr Johann Siau, Mrs Scarlett Xiao, Mr Tony Cook, Mr John Wilmot and Mr Colin Manning.

Last but not least, I am also thankful to my colleagues including Dr. Wansu Lim and Mr Ahmed Amate for establishing a friendly working environment.

Author’s Publication

Journal Papers

Published

- X. Zhai, F. Bensaali and S. Ramalingam, “Improved Number Plate Localisation Algorithm and its Efficient FPGA Implementation”, IET Circuits, Devices & Systems, vol.7, issue 2, pp. 93-103, Jan, 2013.
- X. Zhai, F. Bensaali and R. Sotudeh, “FPGA-based Number Plate Binarisation and Adjustment for ANPR Systems”, SPIE Journal of Electronic Imaging, vol. 22, issue 1, pp. 1-11, Jan, 2013.
- X. Zhai, F. Bensaali, “Improved Number Plate Character Segmentation Algorithm and Its Efficient FPGA Implementation”, Journal of Real-time Image Processing, Springer, DOI: 10.1007/s11554-012-0258-5, pp. 1-13, June, 2012.

Accepted

- X. Zhai, F. Bensaali and R. Sotudeh, “Real-Time Optical Character Recognition on FPGA for ANPR”, accepted for publication in IET Circuits, Devices & Systems, May, 2013.

Conference Papers

- X. Zhai, F. Bensaali and R. Sotudeh, “OCR-Based Neural Network for ANPR”, in Proceedings of IEEE International Conference on Image Systems and Techniques (IST), pp. 393-397, UK, July, 2012.
- X. Zhai, F. Bensaali and S. Ramalingam, “Real-Time License Plate Localisation on FPGA”, in the proceedings of IEEE Computer Vision and Pattern Recognition Workshops, pp. 14-19, USA, June, 2011.

- X. Zhai, F. Bensaali and S. Ramalingam, “License Plate Localisation based on Morphological Operations”, in the proceedings of 11th International Conference on Control, Automation, Robotics and Vision, pp. 1128-1132, Singapore, December, 2010.

Table of Contents

List of Figures

List of Tables

Abbreviations

ANN - Artificial Neural Network

ANPR - Automatic Number Plate Recognition

API - Application Programming Interface

ASIC - Special Purpose Application Specific Integrated Circuit

BP - Back Propagation

CAST - Centre for Applied Science and Technology

CCA - Connected Component Analysis

CLB - Configurable Logic Block

CPU - Central Processing Unit

CS - Character Segmentation

DBM - Deep Boltzmann Machine

DBN - Deep Belief Net

DFF – D Flip Flop

DK - DK Design Suite

DSP - Digital Signal Processor

EDIF - Electronic Data Interchange Format

FPGA - Field Programmable Gate Array

G-DCD - Global Direction Contributivity Density

GPP - General Purpose Processor

GPU - Graphic Processing Unit

GUI - Graphical User Interface

HD - High Definition

HDL - Hardware Description Language

HLS - High Level Synthesis

HLS - Hue Saturation Intensity

HMR - Horizontal Memory Reader

IDE - Integrated Development Environment

ITS - Intelligent Transportation System

L-DCD - Local Direction Contributivity Density

LUT - Lookup Table

MAC - Multiply-Accumulate

MPGA - Mask Programmable Gate Arrays

MRF - Markov Random Field

NADC - National ANPR Data Centre

NN - Neural Network

NP - Number Plate

NPL - Number Plate Localisation

OCR - Optical Character Recognition

PAL - Platform Abstraction Layer

PAL - Programmable Arrays Logic

PALSim - PAL Virtual Simulation Platform

PAR	Place-and-Route
PNN	Probabilistic Neural Network
RMSE	Root-Mean-Square Error
RSE	Root Square Error
RTL	Register Transfer Level
SCG	Scaled Conjugate Gradient
SD	Standard Definition
SE	Structuring Elements
SOM	Self-Organising Map
SVM	Support Vector Machine
VCP	Vertical Critical Point
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VMR	Vertical Memory Reader
XNF	Xilinx Netlist Format
XPA	XPower Analyser
XST	Xilinx Synthesis Technology

Chapter 1: Introduction

This chapter provides an introduction to the rapidly emerging Automatic Number Plate Recognition (ANPR) technology. It begins with a brief overview of current ANPR systems, applications and high performance solutions for their implementation in sections 1.1 and 1.2. Following the brief overview, a summary of the motivations and objectives of this research is given in Section 1.3 and then conclude with the organisation of the thesis in Section 1.4.

1.1 ANPR systems

Intelligent Transportation Systems (ITSs) have had a wide impact on people’s life as their scope is to improve transportation safety and mobility using multiple technology-based systems [1], which includes communication, information and satellite technologies in traffic congestion, safety enhancement and improving quality of environment [2]. ANPR is used as an important technology for intelligent infrastructure systems like electronic payment systems, access control, tracing of stolen cars, or identification of dangerous drivers [3-5].

ANPR systems have been successfully operated in UK for several decades. First generation ANPR systems were invented in 1976 at the Home Office Scientific Development Branch in England (now known as the Home Office Centre for Applied Science and Technology, CAST) and they have successfully detected simple crimes: Tracking and finding stolen vehicles and prosecuting uninsured or un-taxed road users [6]. One successful example is the UK’s “Ring of Steel” around the city of London. The area covered includes London congestion charge zone, where motorists are required to pay a congestion charge. There are currently 1,500 ANPR cameras that monitor anywhere in the zone and around 98% of vehicles moving within the zone are caught on cameras. The video streams are transmitted to the National ANPR Data Centre (NADC) where ANPR software processes the registration plate of the vehicle [7]. Currently, the vehicle information is gathered from fixed cameras strategic sites (e.g. main roads, motorways and petrol stations), mobile units (e.g. police van) and CCTV in towns and cities, there are 35 million number plate reads per day and this number is increasing every year [8]. Figure 1-1 illustrates the applications of ANPR.

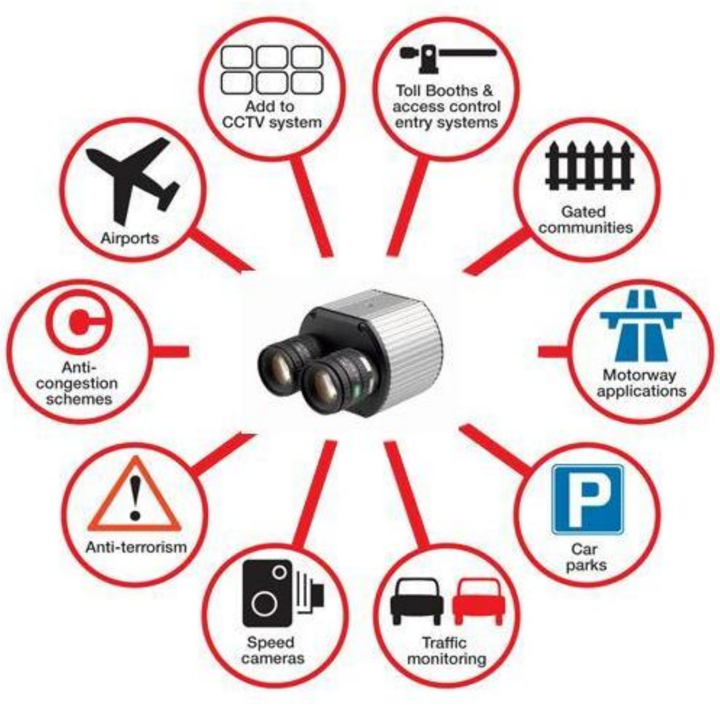


Figure 1-1: ANPR applications [9]

Typically, an ANPR system consists of three stages: Number Plate Localisation (NPL), Character Segmentation (CS), and Optical Character Recognition (OCR). The NPL stage is where the Number Plate (NP) being detected. The CS stage is an important pre-processing step before applying OCR, where each character from the detected NP is segmented before recognition. In the last stage, characters are segmented from the NP so that only useful information are retained for recognition where the image format will be converted into characters by pre-defined transformation models [5].

Currently, the common hardware implementation choice for ANPR implementation is often high performance workstations and expensive computers [10]. The reason of that is the nature of image processing applications as it involves performing complex tasks repeatedly on a large set of image data under real-time requirements. Current ANPR systems generally use a separate camera and a stand-alone PC for processing. However, the cost, compactness and power issues that come with these solutions motivate the search for more cost and size efficient platforms. Hardware accelerators are efficient technics that provide extensions to the computational capabilities of a specific system to improve the performance and reduce the power consumption.

1.2 Hardware Solutions for ANPR Systems

As mentioned in section 1.1, most methods in current ANPR systems utilise general purpose Central Processing Units (CPUs) to perform complex and computationally intensive image processing algorithms. The CPU must read each instruction from memory, decode it and then execute it. Additionally, any operation needs to be implemented from basic arithmetic and logical operations in CPU, which slow down the execution speed for each individual operation.

Therefore, in order to achieve real-time performance, specialist hardware platforms can be one of valuable solution for accelerating computationally intensive image processing algorithms. Currently, the most commonly used hardware for solving such problem are Digital Signal Processors (DSPs), Graphic Processing Units (GPUs), Special Purpose Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs). From these hardware platforms, most of researchers have chosen DSP and/or FPGA as their platform for implementing ANPR systems [11] [12] [13]. However, most of the hardware based systems focus only on one or two stages of ANPR system due to limited hardware resources or complexity of the chosen algorithms. In the following sections, a general overview of these hardware platforms is given.

1.2.1 Digital Signal Processors

DSPs are specialised microprocessors that have optimised architectures suitable for the implementation of operations used in digital signal processing. DSPs are widely used in digital processing systems such as wireless communications, audio and video processing [11]. The DSP features are outlined below [14]:

1) High performance: DSPs are capable of performing one or more multiply-accumulate (MAC) operations in one instruction cycle, which can be used in matrix multiplication operation found in many digital filters and other image processing algorithms (e.g. colour space conversion). DSPs are known for their irregular instruction sets, which allow several operations to be encoded in a single instruction. In general, DSP instruction sets allow data moves to be performed in parallel with an arithmetic operation.

2) Low latency: DSPs often have special memory architectures that allow them to fetch multiple data and/or instructions in a single instruction cycle. They also have specialised execution control, which allows tight loops to be repeated without spending any instruction cycles for updating and testing the loop counter or for jumping back to the top of the loop.

As a result, DSPs have been widely used in intensive tasks, such as image processing, audio processing [15, 16] . They provide the computing power necessary to process large amounts of data in real-time [11, 17]. However, the power consumption and space constraint issues limited their usage in portable devices [18].

1.2.2 Graphic Processing Unit

GPU is a specialised electronic processor to accelerate 2D or 3D image processing. Over the past ten years, the performance and capabilities of GPUs have increased considerably; they are widely used in embedded systems, personal computers, mobile phones and game stations. Currently, GPUs are not only powerful graphics engines but also highly parallel processors substantially that have advantages over general purpose CPU, which offer a great deal of promise for future computing systems. Below are some GPU features [19]:

1) GPUs can deliver high computing performance to process billions of pixels per second, which makes them a suitable hardware platform candidate for complex real-time applications.
2) GPUs can offer parallelism for many types of embarrassingly parallel task including ray tracing and weather modelling, where data parallelism is exhibited for high throughput type computations.
3) GPUs can offer very deep pipelining to increase the throughput rate. The pipeline is also feed-forward, removing the penalty of control hazards, further allowing optimal throughput of primitives through the pipeline.

Although GPUs can offer high computational performance for real-time image processing, they need high memory bandwidth and huge computational graphic hardware resources to speed up the processing, which can result in very high power consumption. Therefore, GPUs are unsuitable for embedded vision applications with restricted power constraints [20].

1.2.3 Special Purpose Application Specific Integrated Circuits

ASICs are designed specifically to efficiently perform a given computation task because they can be optimised for one or more design metrics, such as power consumption and area. However, after fabrication the circuit cannot be altered unless the chip is modified, and this is an expensive process when considering the difficulties in replacing ASICs in a large deployed system.

The main disadvantages of this hardware approach can be summarised as follows:

1) Development times for ASICs are normally longer than other hardware solutions.

2) Costs of ASICs are reduced only when the volume required justify their fabrication costs.

3) Flexibility for ASIC is worse than other hardware solutions. Once this special purpose hardware is built, it is not possible to change the hardware to meet other needs. The only solution is to use a new hardware to replace the existing one to meet its requirements.

Re-designing and re-fabricating any part of the ASIC increase the cost of the product. Structured ASIC can cut the expenses by more than 90% and speedup time-to-market for derivative chips. The reason of that is only a small number of chip layers must be custom-produced in Structured ASIC (SASIC), which is bridging the gap between FPGA and Standard-cell ASIC designs. For example, users normally need to design power, clock, and test structures, the SASIC provides those predefined architectures and therefore can save time and expense for designer compared to Standard-cell ASIC. SASIC technology is especially suitable for platform ASIC designs that have integrated most of the IP blocks and leave some space for custom changes [21] [22].

Alternative hardware approach to avoid many of ASIC's disadvantages is the use of reconfigurable hardware in the form of FPGAs, which can increase the flexibility of ANPR system before fabrication the circuit. This technology is introduced in the next section.

1.2.4 Field Programmable Gate Arrays

FPGA is an integrated circuit that can be configured by a customer after manufacturing. The first commercially viable FPGA was designed by Xilinx in 1985, which had programmable gates and programmable interconnects between gates that served as a hybrid device between Programmable Arrays Logic (PALs) and Mask Programmable Gate Arrays (MPGAs). Although it only had 64 Configurable Logic Blocks (CLBs) with two 3-input Lookup Tables (LUTs), this was a beginnings of a new technology and market [23]. However, nowadays with great flexibility, capacity and performance, FPGAs opened up completely new avenues in high-performance computation, forming the basis of reconfigurable computing [24].

FPGAs can be used to implement any logical function that an ASIC could perform, with the added advantage of ability to update the functionality after shipping with a minor cost. FPGAs can also be the final products without fabricating an ASIC, the cost of FPGAs are lower than the ASIC when the volume of product is not high. During the last decade, as new material technologies are introduced for fabricating the FPGA chips, more and more resources can be integrated into a single FPGA chip [25]. For example, a latest Xilinx Virtex-7 FPGA contains 1,955,000 logic cells, 3,600 DSP slices and many other resources on the chip [26]. The architectural innovations makes new 28 nm FPGAs are well suited for high performance and low-power consumption applications [27].

Recently, FPGAs fused features of embedded microcontrollers with FPGA fabric to make FPGAs easier for embedded designers. This allows developers to apply a combination of serial and parallel processing to address the challenges in cutting-edge research on topics ranging from programming technology, cryptography to real-time systems. For example, Xilinx Zynq-7000 series FPGAs enable extensive system level differentiation, integration, and flexibility through hardware, software, and I/O programmability [28]. There is no doubt that the use of FPGAs has increased the performance of a wide range of computationally intensive applications [29].

Since Virtex-4 FPGA has been used for the hardware implementations of different ANPR stages, the following sub-sections focus mainly on the architecture of Virtex-4 FPGA.

FPGA Structure

The most common FPGA structure consists of an array of CLBs, switching matrix and connection matrix. Generally, the CLBs can be interconnected to each other through some sort of configurable switching and connection matrix. Figure 1-2 provides a simplified diagram of a generic FPGA architecture [30].

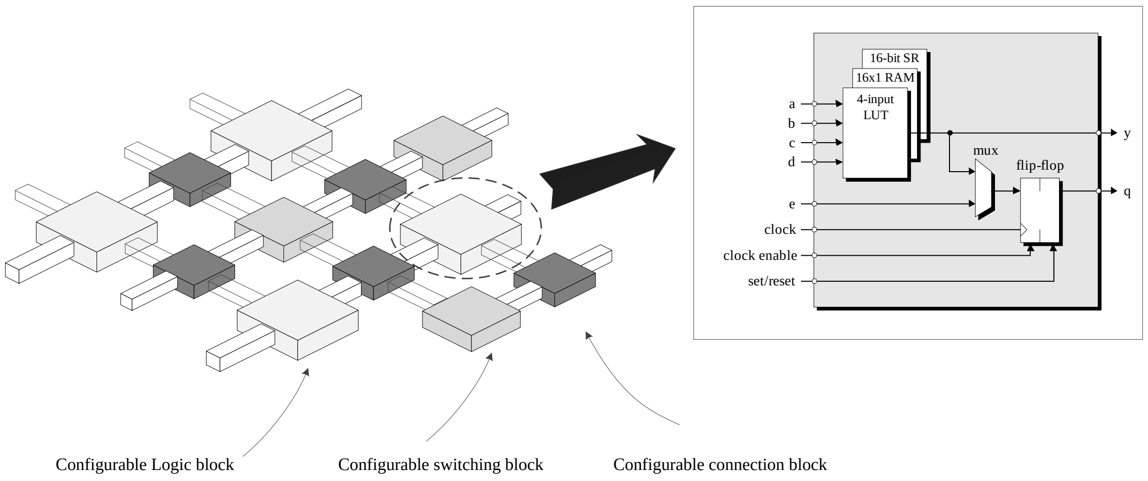


Figure 1-2: A simplified diagram of a generic FPGA architecture

CLBs

The CLBs are the main logic resource for implementing sequential as well as combinatorial circuits. A typical CLB consists of a few logical cells, called slices. For example, Virtex-4 CLB contains four interconnected slices, each paired slices are organised as a column. Each pair in a column has an independent carry chain. However, only the slices in left column have a common shift chain (i.e. SLICE (2) and SLICE (4)) Figure 1-3 shows the arrangement of slices within the CLB for a typical Virtex-4 FPGA.

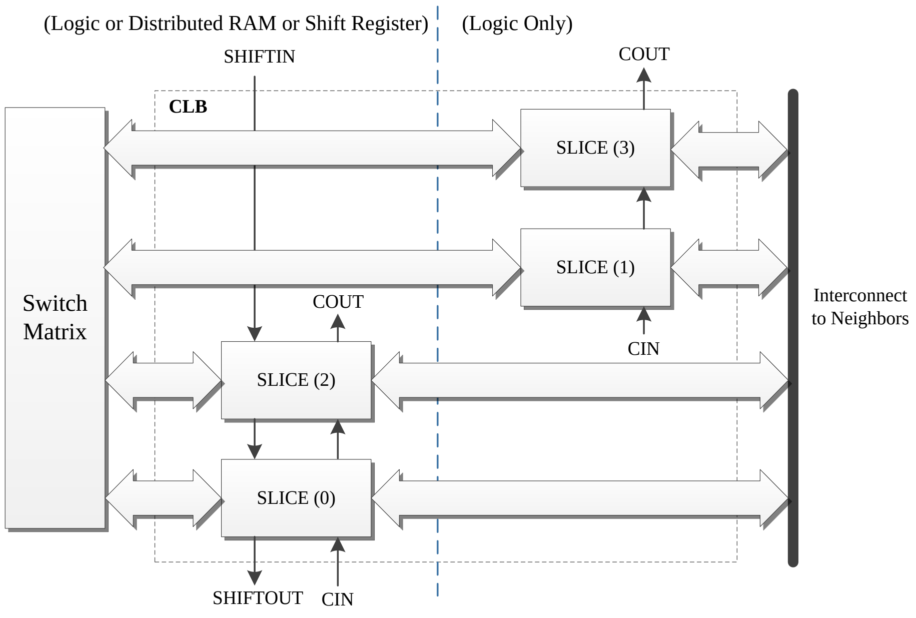


Figure 1-3: Arrangement of slices within the CLB for a typical Virtex-4 FPGA [31]

A simplified diagram of a Xilinx Virtex-4 FPGA slice is presented in Figure 1-4. It consists of two logic-function generators (or LUTs), two D-Flip-Flops (DFFs), multiplexers, carry and arithmetic logic [31]. Both slice pairs provide logic, arithmetic and ROM functions, but left slice pair supports two additional functions: storing data using 16-bit distributed RAM and shifting data with 16-bit shift registers. The LUTs allow any generic four-input logic function to be implemented. The DFF can be used for pipelining, registers, state holding functions for finite state machines, or any other situation where clocking is required. The carry logic is used to speed up carry-based computations in the cell, such as addition and subtraction. The multiplexers are used to combine outputs of the LUTs and so to implement 5-input combinational circuit.

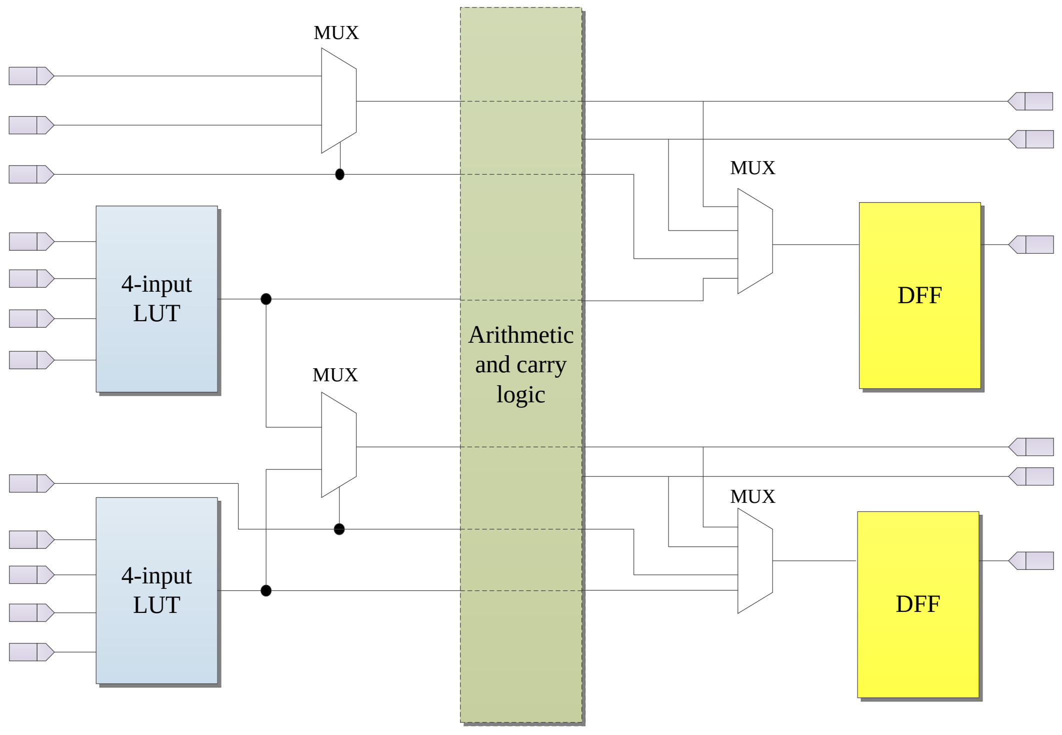


Figure 1-4: A simplified diagram of a Xilinx Virtex-4 FPGA slice [31]

Routing

The routing architecture is designed to handle versatile connection configurations, where horizontal and vertical routing channels are used to connect the CLBs in rows and columns. The routing resources available on the architecture are:

- ***Connection Blocks***

The connection blocks connect the channel wires with the input and output of the CLBs, where signals flow from the CLBs into the connection block, and then along longer wires within the routing channels [32].

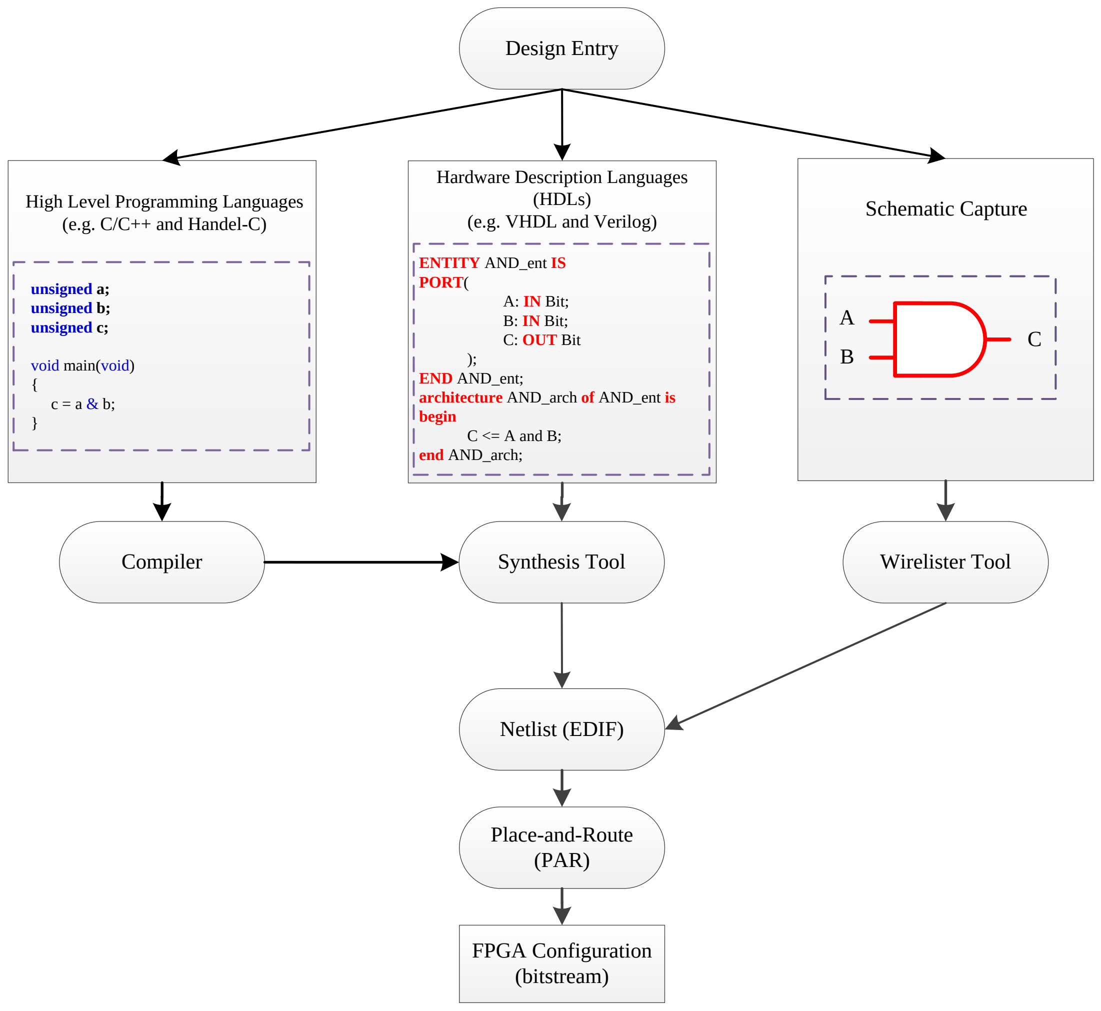
- ***Switch Boxes***

The switch boxes allow wires to switch between vertical and horizontal wires to change their routing direction. When a wire enters a switch box, there are three programmable switches that allow it to connect to three other wires in adjacent channel segments. In the routing architecture of an FPGA, the connection blocks and switch boxes surrounding a single CLB typically have thousands of programming points, which allow to support fairly arbitrary interconnection patterns [32].

In addition to the basic features, modern FPGAs also provide many advanced features such as digital signal processing slices (e.g. multipliers and accumulator), dual-port RAM blocks and embedded processor cores (e.g. PowerPC). More details about the used features of Virtex-4 FPGA can be found in APPENDIX A.

FPGA Design Flow and Synthesis

A typical design flow for FPGA design consists of a number of tools: Hardware Description Languages (HDLs), schematic capture tools, simulation tools, netlist converters and Place-and-Route (PAR) tools. FPGA design cycle is given in Figure 1-5.



```
unsigned a;
unsigned b;
unsigned c;

void main(void)
{
    c = a & b;
}
```

```
ENTITY AND_ent IS
PORT(
            A: IN Bit;
            B: IN Bit;
            C: OUT Bit
       );
END AND_ent;
architecture AND_arch of AND_ent is
begin
       C <= A and B;
end AND_arch;
```

Figure 1-5: FPGA design cycle

The FPGA design flow starts with design entry and ends with bitstream generation in a top-down manner. There are several vendors in the FPGA market, i.e. Xilinx, Altera and Lattice. Each vendor provides its own front end design tools such as Xilinx ISE from Xilinx [33] and Quartus II from Altera [34] .

A. Design Entry

FPGA designs can be entered either schematically using a schematic editor or textually using programming languages:

- **Schematic Design Entry:** Schematic tools provide a graphic interface for design entry. Designers can design and connect individual logic components and combine them to create functional blocks. Once a design has been specified using schematic capture, it can be converted into a netlist by a schematic-to-netlist converter tool. Because the common logic components are usually defined and stored in a library that is supplied by the FPGA vendor, schematic designs produced for a specific FPGA architecture are not easily portable to other FPGA.

- **HDLs:** There are two most commonly used HDLs in industry, which are Very High Speed Integrated Circuit Hardware Description Language (VHDL) [35] and Verilog [36]. These languages are standard and device independent where hardware architectures described using any of them can be synthesised into a circuit suitable for any FPGA.

- **Handel-C Language** In early 90s, Handel-C was introduced and used in the programming of FPGAs as a high level programming language at Oxford University Computing Laboratory [25], and then it became a product in Celoxica from September 2000. However, Handel-C was first acquired by Agility in 2008, after 2009 it was purchased and maintained by Mentor Graphics [37]. Handel-C is essentially an extended subset of C, specifically designed for controlling hardware instantiation with an emphasis on parallelism [38]. Unlike many other design languages that rely on going via several intermediate stages, Handel-C can be either directly targeted on hardware or compiled to a number of design languages in a hardware compilation system known as the Mentor Graphic Development Kit (DK) [37], and then synthesised to the corresponding hardware using the common synthesis tools (e.g. Xilinx ISE). The main advantages of Handel-C over the other programming languages are the rapid prototyping and the software liked simulator. The works presented in [39] [40] have shown that Handel-C shortens design time by a factor of 3-4 times with approximately the same operating speed compared to traditional HDLs. Unlike many traditional HDL simulators that only provide simulation waveform of the outputs, Handel-C simulator can display contents and the status of all variables in a program or design for every clock cycle, which make the debugging process much easier compare to traditional HDL simulators. More details about the used features of Handel-C can be found in APPENDIX B.

Synthesis and Netlist Representation

There are two major synthesis areas used in digital design: Register Transfer Level (RTL) and High Level Synthesis (HLS). Typically the synthesiser converts HDL (VHDL/Verilog) code into a RTL, e.g. Xilinx ISE [33] and Altera Quartus II [34]. Recently Xilinx introduced a new generation of synthesis tools named Vivado HLS, which can automatically transform high level programming languages (e.g. C/C++) to a RTL specification and then synthesised into Xilinx FPGA [27]. After running synthesis the designs are mapped onto a specific structure suitable for the target architecture and become netlist files that are accepted as input to the next implementation stage. A netlist is a standard textual representation of a design, which contains a more structured description of the functionality of the design and specific component information of an FPGA vendor. The netlist format can be in the standard Electronic Data Interchange Format (EDIF) format [41] or in another vendor specific format (e.g. Xilinx netlist Format (XNF) from Xilinx).

Place-and-Route Tools

PAR tools can map the design of integrated circuits and routing information onto the FPGA architecture. The process of PAR consists of two steps, placement and routing. At the placement step, all the electronic components, circuitry and logic elements are placed in a generally limited amount of space. After the placement step, the routing step decides the design of all the wires needed to connect the placed components under the requirements of the rules and limitations of the manufacturing process. In addition, the PAR can find timing constraints associated with the design and invoke timing-driven routing automatically. Once the PAR operation has been successfully performed, a bitstream file can be generated and download into FPGA for the configuration [33].

Compared to other hardware solutions, FPGA can provide the most flexibility and competitive performance for a real-time image processing design.

1.3 Research Motivations and Objectives

The fundamental requirements of an ANPR system are image capture using an ANPR camera, and processing of the captured image. The image processing part, which is a computationally intensive task, includes three image processing stages (i.e. NPL, CS and OCR). The common hardware choice for its implementation is often high performance and expensive computers. However, the cost, compactness and power issues that come with these solutions motivate the search for other low cost platforms.

As mentioned in Section 1.2, DSP, GPU, ASIC and FPGA are commonly used hardware solutions for accelerating computationally intensive image processing tasks. After comparing their merits and drawbacks, FPGAs is chosen in this research work as the hardware platform based on the following advantages: First of all, FPGA allows truly parallel computations to be placed in a circuit. Although many modern General Purpose Processors (GPPs), DSPs and GPUs can emulate parallelism by switching tasks very rapidly or rely on specialised hardware architectures, having operations truly performed in parallel results in a much faster processing speed even in a relatively lower operating clock frequency. Secondly, FPGA can create actual hardware to test instead of simply relying on simulators. The reconfigurable ability of FPGA allows a design to be completely tested and debugged before an ASIC is created, saving on production costs. However, algorithms to be implemented on FPGAs need to be carefully chosen or developed to fully exploit the parallelism and on-chip resources offered by those devices. Therefore, FPGA is an extremely powerful tool for accelerating image processing algorithms, and also balance the gap between software and hardware design to allow maximum performance and flexibility can be delivered during the research development. By optimising the ANPR algorithms to take specific advantage of technical features and innovations available within new FPGAs, such as low power consumption, development time, and vast on-chip resources, it will be possible to replace the 3GHz roadside computers with small in-camera dedicated all-in-one platforms.

As the main aim of this research project is to implement the entire ANPR system on one single FPGA, the initial sub-objectives then can be summarised as follows:

- To develop improved real-time NPL algorithm that is suitable for FPGA implementation;
- To develop novel efficient architectures for the proposed NPL algorithms and implement the architectures on FPGA;
- To investigate and develop improved real-time CS algorithm and its novel efficient architecture implementation on FPGA;
- To investigate and develop improved real-time OCR algorithm and its novel efficient architecture implementation on FPGA; and
- To investigate and develop NP binarisation and adjustment algorithms and their novel efficient architecture implementations on FPGA.

1.4 Organisation of the Thesis

The structure of the remainder of this thesis is as follows. Chapter 2 takes a closer look at the most recent software and hardware based ANPR systems. Subsequently, the proposed NPL, CS, OCR and pre-processing algorithms and novel efficient architecture implementations on FPGA are introduced in Chapter 3, 4, 5 and 6, respectively. Chapter 7 describes the proposed entire ANPR System on FPGA and an approach to extend it to High Definition (HD). Concluding remarks and opportunities for future work are presented in Chapter 8.

Chapter 2:Related Work

2.1 Introduction

In previous chapter, ANPR systems and their common solutions have been briefly discussed. An ANPR system normally consists of three main stages: NPL, CS and OCR, and each stage of the ANPR system is mainly based on different image processing and pattern recognition algorithms. The common implementations of the ANPR systems can be software or hardware. Software based solutions currently have been researched intensively and robust algorithms have already been proposed for each stage of an ANPR system [5, 42-45], however, there are only few state-of-the-art hardware implementations. This chapter takes a closer look at the most recent software and/or hardware solutions for NPL, CS and OCR. A synopsis of the shortcomings of existing work and concluding remarks are also provided.

2.2 Number Plate Localisation

The performance of the NPL stage, in terms of speed and localisation rate, is crucial to the entire system, because it directly influences the accuracy and efficiency of the subsequent steps [45]. Generally, NPL algorithms reported in previous research are mainly classified into three classes: edge detection, colour processing and texture-based algorithms.

2.2.1 Edge Detection based Algorithms

Techniques based on edge detection statistics featured very good results in previous research works [46-50], the reason is that the algorithms utilise the change of brightness in the NP region is more remarkable and more frequent than elsewhere. In order to obtain the change of brightness, one of the algorithms is computing the gradient magnitude and the local variance of an image. Figure 2-1 shows an example of NPL algorithm using edge detection technique.

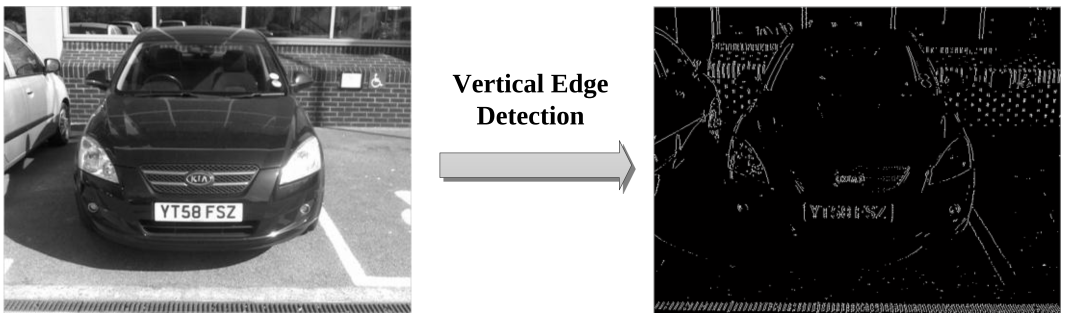


Figure 2-1: An example of NPL algorithm using edge detection technique

As shown in Figure 2-1, after a vertical edge detection operator is applied on a greyscale car image, the most of vertical edge information is appeared in the NP region, which means the NP can be easily localised using the density of edge information. However, a disadvantage of this method is that it cannot deal with the complex images, since the edge detector is too sensitive to unwanted edges, which may also show a high edge magnitude or variance (e.g. the radiator region in the front view of the vehicle). In spite of this, after combining with morphological operations that eliminate some unwanted edge information, the NPL rate is relatively high and fast, compared to other methods [51]. In [47], a combination of edge detection and morphology based algorithm is proposed for controlling highway charging system is proposed, the flow chart of the proposed algorithm is shown in Figure 2-2.

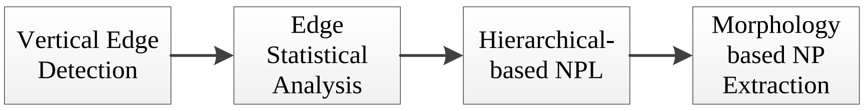


Figure 2-2: The flow chart of a combination of NPL algorithm based on edge detection and morphology

The results achieved in [47] show that the average accuracy of NPL is 99.6% (9786 from 9825 images). In order to achieve this impressive result, a fixed distance and angle of a camera is required to boost the NPL rate to a high level of accuracy, which means candidate regions are expected in a specific position and priority is given to them.

Edge detection is usually followed by Connected Component Analysis (CCA) which is a well-known algorithm in binary image processing that detects connected regions in binary digital images and label their pixels into components based on pixel connectivity (e.g. 4-connectivity or 8-connectivity) [52]. Once all groups of pixels have been labelled, many useful geometrical measurements and features in each binary group can be extracted, for example, area, aspect ratio, width and height. Those measurements and features are frequently integrated in NPL algorithms for the localisation of NP region [53] and [54]. Figure 2-3 illustrates an example of CCA algorithm.

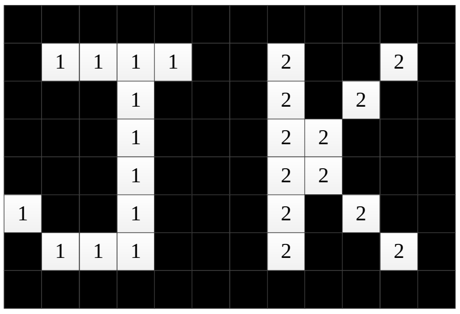


Figure 2-3: An example of CCA algorithm

In Figure 2-3, there are two characters labelled in two different groups based on the pixel connectivity, and according to the predefined measurements and features, the characters can be easily localised and extracted from the original image.

2.2.2 Colour Processing based Algorithms

Many colour processing based algorithms are proposed in the previous NPL works. The principle of these algorithms is utilising the expected NP appearance in a specific country, for example, NP background and character colour. The basic idea of the NP region extraction is based on a unique colour combination of a NP background and foreground character in the car image. For example, as the Chinese NPs have specific formats, the work presented in [55], suggested that all pixels in the input image should be classified using the Hue Saturation Intensity (HSI) colour model into 13 categories based on variance of illumination in the RGB domain. They are dark blue, blue, light blue, dark yellow, yellow, light yellow, dark black, black, grey black, grey white, white, light white and others. In addition to the colour categories, the width to height ratio values are also used for classification of the NP regions. At the end of the process, the NP region is extracted vertically, and then horizontally from the car image. Figure 2-4 shows the vertical and horizontal extractions of a NP region.

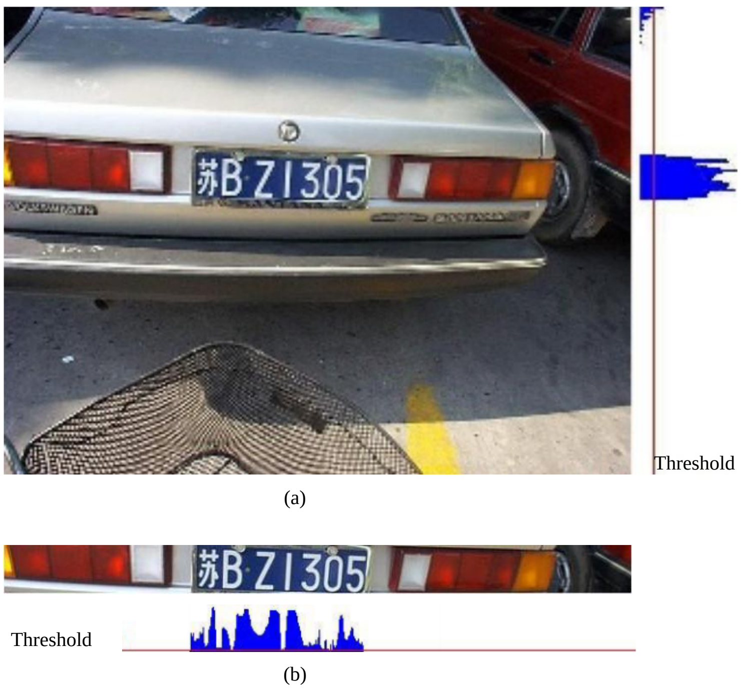


Figure 2-4: NP region extraction. (a) vertical extraction (b) horizontal extraction [55]

Fuzzy logic has also been introduced in [56-61] to classify the colours in the NPL stage. The NP is described and given some membership function for the fuzzy sets, for example, ‘bright’, ‘dark’, ‘bright and dark sequence’, ‘texture’, and ‘yellowness’, and then the fuzzy logic can be used for classification of the proposed fuzzy sets. The following intuitive rules of an NP region have been defined in [58] based on human perception:

- Bright rectangle area that includes some dark areas;
- The border of the NP is bright;
- Approximately localised in the middle or lower middle part of the image;
- Size of the NP is about 530×120 mm.

A fuzzy set with trapezoidal membership functions on the interval [0, 255], used to present the concept of illumination condition, where ‘0’ and ‘255’ represent black and white colours respectively. The input image (768×576 pixels) is partitioned into many sub-images (75×25 pixels) and the fitness to the four rules is calculated for each sub-image. However, according to the experimental results, this method is computationally intensive task and the last two rules restrict the algorithm to identify NPs in a specific distance.

In [56], an edge detector is designed to find three kinds of edges from an image, which include black-white, red-white and green-white. Firstly, an edge image **E** is initialised with only white, black, red and green colours, and then the RGB colour model is transformed into the HSI model. The basic idea is to generate a fuzzy map from **H**, **S** and **I** maps and edge image **E**. Finally, those fuzzy maps are combined together into a single map **M**. This method showed an NPL rate of 97.9% using a database with 1088 colour images.

2.2.3 Texture-based Algorithms

Texture-based algorithms mainly use image transformation to analyse the texture information. The most common image transformation techniques include Gabor filters, Hough transform and wavelet transform. These techniques directly analysing texture information without limitation of the NP direction and size. In the work presented in [62], the Gabor filter is used to extract the features of image. The filter responses that result from the convolution with Gabor filter are directly used as NP detector. There are three different scales and four directions used in a 12-Gabor filter. High values in the image $r(x, y)$ indicate probable plate regions. Finally the NP regions are extracted by applying 8-connectivity CCA algorithm. A high NPL successful rate of 98% using 300 images has been achieved. However, this method is computationally expensive and slow for large images.

Another texture-based algorithm is proposed in [63], where the edges in the input image are first detected, and then a contour algorithm is used to detect closed boundaries of objects, the contour lines are transformed to Hough coordinate to find two interacted parallel lines that are considered as a plate-candidate. Since the numbers of pixels in the contour lines are much less than the pixels in the original image, the calculation of Hough transform is more efficient and the speed of the algorithm has improved significantly without accuracy loss. The NPL rate achieved in [63] is 98.8% when using only close shots of the vehicle. Improved methods to speed up the transformations are described in [64], [65] and [66].

In [67], a Haar scaling function for wavelet transform is proposed. The grey-level image is firstly binarised by a predefined threshold to highlight the feature of NP region, and then applying wavelet transform with different parameters to generate four corresponding sub-images: low-pass-filtered, characteristics contained in vertical direction, characteristics contained in horizontal direction, and cater-corner characteristics, namely LL, LH, HL and HH, respectively. After wavelet transform, five steps to localise the NP region are proposed:

1. Find the reference line by horizontal variation in LH sub-image;
2. Decide the size of the mask;
3. Find the candidate regions below the reference line;
4. Candidate region verification;
5. Searching the complete NP region.

The average accuracy of detection achieved in this paper was 92.4%, however, the fixed size of mask and reference line for finding the candidate make the method is unreliable, because it required the distance between the vehicle and acquisition camera to be in a fixed range.

2.2.4 Discussion

The NPL algorithms presented in the previous sections normally require pre-defined working environment to extract the NP from an input image, for example, camera distance, background environment, vehicle position and lighting condition. In real-world applications, NPL algorithms should be able to cope with variable camera-to-car distances and environments. On the other hand, according to the reported processing speed of NPL in the literature, the NPL stage is the most computational intensive stage in the entire ANPR system. For a real-time ANPR system, efficient and robust NPL algorithms are required to accelerate the speed of the entire system. Table 2-1 summarises the performance of the existing NPL algorithms.

Table 2-1: Performance of Existing NPL Algorithms

NPL Algorithm	Country	Image Type	NPL Successful Rate (%)	Speed (*ms*)
Improved Bernsen algorithm and CCA [45]	Japan	Greyscale	97.16	158
Edge detection and morphology [47]	China	Greyscale	99.6	100
Fuzzy logic [59]	Taiwan	Colour	97.9	N/A
Fuzzy logic [60]	China	Colour	95.1	400
Support vector machine [61]	Korea	Colour	92.7	1280
Wavelet transform [44]	Taiwan	Greyscale	97.3	180
Sliding concentric windows [68]	Greece	Greyscale	96.5	N/A

2.3 Character Segmentation

CS is an important stage in ANPR systems as correctly and accurately segmented characters are more likely to be successfully recognised [5]. In recent years, many CS techniques have been developed for text in printed documents [69] [70], however, due to the real-life use of ANPR systems, the obtained NP images are noisy (e.g. uneven illumination, inclined NP and connected characters) [1]. In order to overcome these issues, a wide variety of modified or improved character segmentation techniques have been developed. There are mainly three CS algorithm categories:

- Projections and binary algorithms;
- Contours tracking algorithms; and
- Classifiers based algorithms.

The three types of CS algorithms are discussed with more details in the following sections.

2.3.1 Projections and Binary Image Processing Algorithms

The most common used CS algorithm is the one based on vertical and horizontal projections of the pixels [50, 59, 71-73]. The idea is to sum up pixels that belong to one column or row of a binary NP image and obtain two row and column vectors (or projections), then analyse them based on their projection histogram to identify the local minimum critical points where the characters need to be segmented. The proposed process is illustrated in Figure 2-5 below.

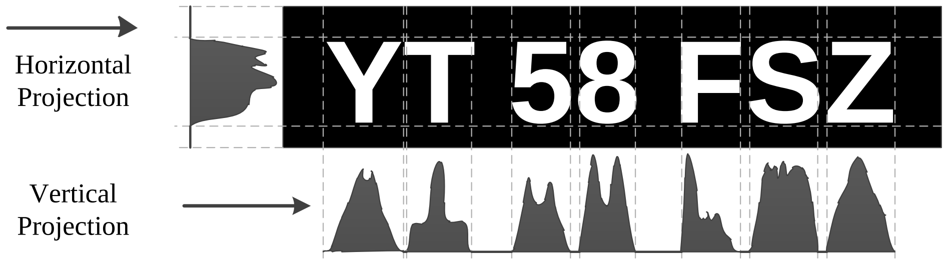


Figure 2-5: Horizontal and Vertical projection

The main advantage of this method is its low complexity and straightforward implementation, but it does not perform well when the NP has connected characters due to noise, and the entire horizontal pixels projection cannot provide exact horizontal position of a character when NP is inclined or noisy, which will cause difficulty when identifying the local critical points of each character.

Another used method applied to binary images for CS is CCA which is based on some geometric conditions where height, width, and area of characters need to be measured [74-79]. In this method the correct position of each character can be extracted even if the

NP is inclined, but it requires each character to be fully connected and neighboured characters must not be connected. Figure 2-6 illustrates a failure case of CS when only using the CCA algorithm.

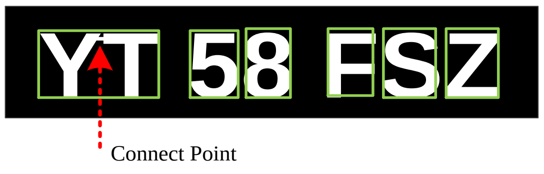


Figure 2-6: A failed example of CS when using the CCA algorithm only

Therefore, in order to overcome this issue, this method is usually combined with mathematical morphology. Adaptive approaches for degraded NP images have been developed in [80] and [81] which include morphological operations. These methods apply the thickening and pruning algorithm to binary images to remove noise and search critical segmentation points in the projection histogram. For the aforementioned task, prior knowledge of the maximum quantity of segments for character or number was employed to decide whether the merging is necessary. The morphological operators are used for the merging and separating overlapping or connected characters [80].

2.3.2 Contours Tracking Algorithms

The second type of CS algorithms is contour tracking. The works proposed in [82] and [83] fall into this category, where the boundary information of characters is used. The algorithm extracts contour line for each character into eight and four directions by using 3×3 and 2×2 masks respectively, and then divides NP region into higher part and lower part using density indicating histogram for y-axis direction. Figure 2-7 shows a 2×2 mask and the required progressing for extracting contour line.

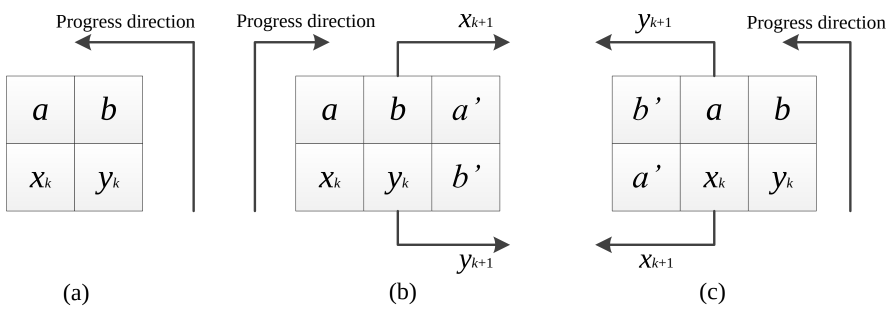


Figure 2-7: 2×2 mask and the contour line extraction process. (a) 2×2 mask. (b) when a and b are boundary pixels. (c) when a and b are background pixels.

The 2×2 mask algorithm chooses a boundary pixel in the corresponding region as a starting point, and then determines the next progressing direction of mask by considering two pixels *a* and *b*. The tracking start direction is anticlockwise, if *a* and *b* are boundary and background pixel respectively, the tracking direction remains anticlockwise. If *a* and *b* are either boundary pixels, or background and boundary pixels respectively, the tracking process starts from the right side neighbouring pixel and continues clockwise as shown in Figure 2-7 (b). If *a* and *b* are both background pixels, then the tracking process starts from the left side neighbouring pixel and continues in the opposite of the previous direction as shown in Figure 2-7 (c).

In [84] and [85], a shape-driven active contour model is established, which uses a variation fast matching algorithm for NP character segmentation, where a coarse extracting of boundaries and class labels of each character are proposed using a shape driven fast marching technique with a gradient and curvature dependent speed function:

$$F = \frac{F_k}{1+\alpha\left|\nabla G_\sigma * I\right|} \tag{2.1}$$

Where F_k denotes a curvature related term in order to keep the propagating curve as smooth as possible [85].

Firstly, the algorithm initialises the front at the borders of the image, and then performs fast matching interactions with speed function using Equation 2.5. Figure 2-8 shows the coarse segmentation results:

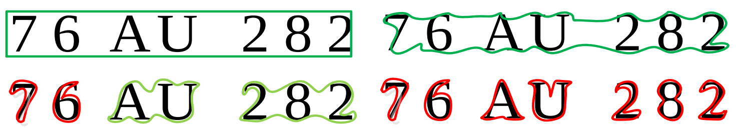


Figure 2-8: The coarse segmentation results [85]

After locating the coarse character boundary, a fine character boundary process is used to locate fine character boundaries and classify them with evolving active contours in the fast marching scheme, which depends on gradient, curvature and shape similarity information. The method is also capable of segmenting broken characters and using the final merged segmentation results for recognition.

2.3.3 Classifiers Based Algorithms

The third category for character segmentation is based on classifier networks. The method proposed in [86] and [87] models the extraction of characters as a Markov Random Field (MRF), where prior knowledge of NP is used to maximise a posteriori probability. Subsequently, a genetic algorithm with a local greedy mutation operator is employed to optimise the objective function and convergence. The method was developed for CS in NP video sequences.

In [88], a method for segmentation of a line of characters in a noisy low resolution image of a car NP is introduced, where the hidden Markov chains are used to model a stochastic relation between a input image and the corresponding CS. A training set of examples with a ground truth segmentation provided by a user that is used for the learning of the statistical model, which allows the classifier to mimic the user's segmentation and exploits the entire prior knowledge specific for the application at hand. For the prior knowledge, they assume that the characters can be segmented into sectors with the same but unknown width. The proposed method is able to segment characters correctly even in images of a very poor quality. The error rate 3.3% was achieved on the testing set with 1000 examples captured by a real ANPR system. However, for example, the number and width of characters are normally unknown such as the number of characters on a UK NP can be in the range 3 to 7, and inclined NPs can affect the width of characters.

2.3.4 Discussion

The low computing complexity of pixel projection makes it the most common used method for character segmentation. However, this method relies on the shape of characters. Although CCA has the ability to overcome this problem, but all characters on the NP must be isolated and each character must be fully connected. Contour tracking and classifier networks can perform better in more complex environments, but their higher computational complexity limits the flexibility for its hardware implementation. Table 2-2 summarises the performance of the existing CS algorithms.

Table 2-2: The performance of existing CS algorithms

CS Algorithm	Country	CS Successful Rate (%)	Speed (*ms*)
Pixel Projection [45]	Japan	98.34	35
Pixel Projection [50]	Australia	98.82	200
CCA [56]	China	95.6	2000
Bicubic interpolation and fixed position parameters	Greece	89.1	N/A
CCA [79]	Korea	97.2	150
Contours Tracking [82]	Korea	97.7%	N/A
Hidden Markov Chains [88]	Czech Republic	96.7%	N/A

CS stage is a very important stage in the entire ANPR system as the OCR stage fully relies on isolated characters and incorrectly segmented characters are not likely to be successfully recognized. In fact, accurate segmentation of degraded NP images is still a problem in ANPR systems, most of the recognition errors in the ANPR systems are due to segmentation errors.

2.4 Optical Character Recognition

OCR has become an important and widely used technology, which translates scanned images of printed text into machine encoded text. This technology is also used for the recognition of segmented characters in the last stage of an ANPR system. The OCR system for ANPR is relatively less complex compare to other common OCR systems (e.g. hand writing and text scanning) as characters on NP have uniform fonts [89] and [90]. However, in order to handle the noisy and unknown outdoor environment effects, the ANPR system needs a stable OCR algorithm. Most used algorithms are based on statistical classifiers, Artificial Neural Networks (ANN), and common pattern matching techniques [5].

2.4.1 Statistical Classifiers

The statistical classifiers can be divided into two sub-classes: single stage classifier and multistage classifier. Support Vector Machine (SVM) is one of the widely used classifiers for both sub-classes. The work done in [91] uses four SVM-based character recognisers indexed by SVM_{uc}, SVM_{un}, SVM_{lc} and SVM_{ln}, which are used to recognise the characters. Each SVM recogniser is used to recognise different characters located in different positions on Korean NPs where the characters are listed in two lines (e.g. upper characters, upper numerals, lower character and lower numerals). In case of characters, one-per-class decomposition method is used to classify between multi-classes characters. In case of numbers, they use 10 SVMs to recognise 0-9 respectively, and the maximum value of the outputs is selected.

In the work presented in [45], a group of Chinese characters is processed as a character string, the entire character string was normalised and taken as the object of study to reduce the difficulty of character segmentation and post-processing. Feature extraction approaches of Global Direction Contributivity Density (G-DCD) and Local Direction Contributivity Density (L-DCD) were proposed where the eight stroke directions L_i ($i = 1,2,...,8;$

0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°) are used to indicate eight distances between the pixel on a stroke and eight directional edges of the stroke. Figure 2-9 shows the direction feature of stroke.

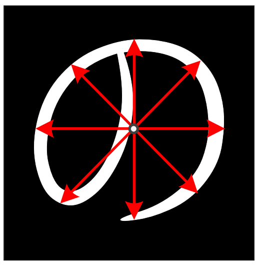


Figure 2-9: The direction feature of stroke

- G-DCD: This is a 1-D feature vector that reflects the complexity, direction, and connected relationship of character strokes. Basically, scanning the image from left to right in the direction of $T = 0, 1, 2, 3$, denotes a 0°, 90°, 45°, and 135°, respectively, all cross points of the scanning line and stroke contour are obtained, then the direction features of all the cross points are calculated and added.
- L-DCD: This feature reflects the local structure of the character. Instead of extracting the features from the whole character image, L-DCD is computed from all sub-images, and then is divided into the original image.
- Contour feature: This feature is represented by the *x*-axis distance between the boundary of character image and the first character pixel when horizontally (0°) scanning the character image from left to right. Similarly, contour features also include other directions, i.e., 45°, 90° and 135°.

In this work, SVM was used as a classifier to recognise characters based on the above features. This was tested using Japanese NPs that including numbers, Kana (Japanese script), and the strings of characters that represent the area. The recognition rates for numbers, Kana and strings of characters are 99.5%, 98.6% and 97.8% respectively.

Many researchers have integrated multistage classifiers to improve the recognition rate of OCR. In [92], a two-stage hybrid OCR system is presented. It firstly uses four statistical sub-classifiers to independently recognise the input character and then the results are combined using the Bayes method [93]. Secondly, if the recognised character from the first stage belong to the set of ambiguous characters (e.g. I/1, B/8 and O/D), a structural stage is used for a further decision. The coarse-to-fine strategy is used in [94] to efficiently organise character from a large number of possible candidates, where the characters are sorted by their shape and ambiguous cases are grouped together, then only specific features are considered in the similar character group.

2.4.2 Artificial Neural Networks

ANNs are intelligent computing architectures widely used for pattern recognition, and the most commonly used and simplest Neural Network (NN) architecture is multilayer feed-forward NN, which can classify inputs into a set of target categories. Typically, the works done in [95] and [48, 96] use binary pixels values and average intensity value of character image to feed the inputs of NN, which can achieve good performance even under complex environments in ANPR systems. A typical multilayer feed-forward NN is shown below in Figure 2-10.

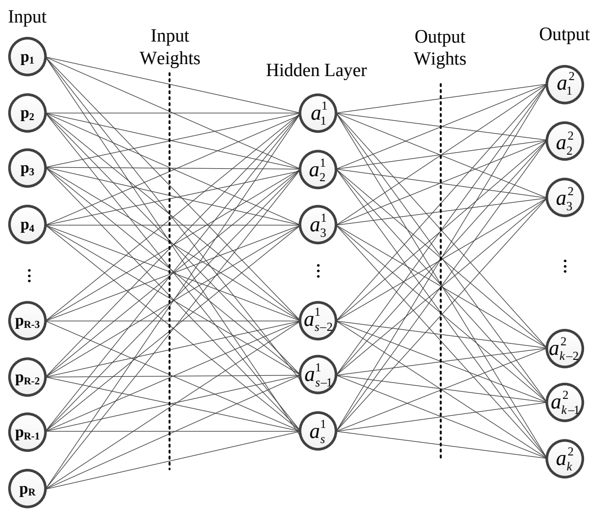


Figure 2-10: A multilayer feed-forward NN

In [95], a standard Backpropagation network is used which has an architecture with 3 layers and 129, 20 and 36 neurons on the input, hidden and output layers respectively. In [96], they proposed a multiplayer feed-forward network that consists of 209 inputs, 104 hidden neurons and 33 outputs. The designed NN can recognise one character at a time. The training algorithm is Backpropagation. The achieved character recognition rate was 95%.

In order to further improve the character recognition rate, employing features extraction of the character images is needed. In [97] and [98], extra procedures during the training stage or after obtaining the results of the NN to handle difficult characters that belong to the set of ambiguous characters are used (see Figure 2-11).

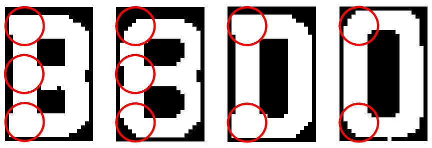


Figure 2-11: The set of ambiguous characters

In [98], the ambiguous characters are used more often for the NN training. After the additional training, the recognition rate was reported to be 98.2%. However, in [56], once misclassified characters are found, an additional minor comparison between the unknown character and the classified character is applied, where only distinguishing parts of ambiguous characters are compared. However, the features extraction normally needs complex computation or multiple stages to extract features. There are also other types of NNs used for classification, such as Probabilistic Neural Network (PNN) [68, 99] , Deep Boltzmann Machines (DBM) [100] and Deep Belief Nets (DBN) [101], which normally give more accurate result but require more memory space and learning time.

2.4.3 Pattern Matching

Common pattern matching is a technique for finding a target image whether it matches a template image or not, which can be one solution for recognising single font and fixed size characters such as NP characters. Normally, the target image is used to compare an image in the pre-defined template data set one by one, where Root Square Error (RSE) or cross-correlation algorithms are used to determine the best matching result.

In [102], a cross-correlation operator is applied between a sub-area of the normalised greyscale image and each prototype. Let g , $\bar{g}$, f and $\bar{f}$ be template image, average grey level of the template image, acquired image and average grey level of the acquired image respectively. The normalized cross-correlation operator defined in the discrete case as follows:

$$C_{fg} = \frac{\sum\sum(f-\bar{f})(g-\bar{g})}{\sqrt{\sum\sum(f-\bar{f})^2(g-\bar{g})^2}} \tag{2.2}$$

The recognition decisions are based on the normalized cross-correlation values C_{fg} . Due to the difference between the character thicknesses for each province, different template sizes are designed for recognising those characters. Once the province has been recognised, the system computed the cross-correlation value for each of the 31 character templates. At each step, the cross-correlation values between the searched template and the corresponding acquired image are stored in a matrix with 31×163 elements. Each matrix row contains the cross-correlation values of each examined template and each matrix column contains the list of cross-correlation value of all the templates in that position over the image. The recognised character will be decided using the greatest mean cross-correlation value that is obtained from each matrix column. The algorithm has been tested with a database that contains 1823 character images, and the achieved recognition rate was 97.97%. However, because the positions of the characters within the NP image are unknown, extractions of each character require a fixed distance and size. In addition, a high computational cost is also required by the cross-correlation measure.

The work in [103] uses root-mean-square error (RMSE) for calculating similarity of a prototype and a given binary image. A method is first proposed to estimate the character size. The estimated character size is resampled to 28 pixels, and then a match competition is performed to find the best match image in the template. If a candidate can be found, the height will be used as the height of character, otherwise, the estimated height will be increased by one until a best match is found. To evaluate the new width, a linear interpolation method is used to resize the character based on the new height. The RMSE approach is used at the end to measure the similarity of a template and the given image. The RMSE e_{rms} can be computed using the following equation:

$$e_{rms} = \left[\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x,y) - f(x,y)]^2 \right]^{1/2} \tag{2.3}$$

where $f(x,y)$ and $\hat{f}(x,y)$ represent the binary values of the input and template images ($M \times N$) respectively. Again, in order to reduce the computational cost required by RMSE measure, the proposed algorithm searches the best match template character by character in the NP based on the previous estimation of the positions of the characters. Unlike the cross-correlation value, the minimum value of the RMSE indicates which the best match template is.

However, these pattern matching methods are not suitable to recognise the slanted and noisy character, which is the type of characters that need to be processed in ANPR systems [5].

2.4.4 Discussion

Incorrectly segmented characters from the CS stage, where characters are not in the expected position or parts of them are missed, may affect the OCR recognition. The NNs and statistical classifiers, which give better results compare to common pattern matching techniques, can overcome this problem due to their strong memorability and self-adapting ability. However, in order to achieve good performance, large amount of samples are needed to train the NNs. Although the OCR technologies are already mature and continuously enhanced over time, they still need improvement in set of ambiguous character (1/I, 0/O, 0/D, 2/Z, 8/B and 5/S). Table 2-3 summarises the performance of the existing OCR algorithms for ANPR systems.

Table 2-3: The performance of existing OCR algorithms for ANPR systems

OCR Algorithm	Country	Character Recognition Rate (%)	Speed (*ms*)
ANN [43]	China	97.1	N/A
SVM [45]	Japan	97.03	18
ANN [48]	Australia	92.03	N/A
Self-organised Map (SOM) [59]	Chinese	95.6	N/A
Probabilistic Neural Network [68]	Greece	89.1	128

2.5 Hardware based ANPR System

Recent improvements in the computing power of FPGAs and DSPs have motivated researchers to consider them as an alternative solution to implement ANPR systems. These devices can be used as a low-cost System-on-chip solution that allows the FPGA or DSP-based processing unit to be placed within an ANPR camera housing to create ‘intelligent cameras’– namely cameras that record and process images for sending back to a server. The main advantage of these solutions is not only the significant improvement in the processing speed, but also the significant reduction of power consumption and cost of the processing unit, which outperform all existing software-based solutions. The most recent DSP and FPGA-based ANPR systems are discussed in this section.

2.5.1 DSP-based ANPR System

As mentioned in Chapter 1, a DSP is a specialised microprocessor that has an optimised architecture for the operations of digital signal processing purpose. The major advantage of DSP-based ANPR systems is their real-time capabilities in city scenarios, which allows the system to catch all the objects that moves through the scene irrespective of the object speed.

In [11], the authors proposed an embedded DSP platform based ANPR system and it can process a video stream in real-time. The proposed system consists of NPL, CS and OCR modules. Approximate NPs are detected first using Viola Jones detector [104], which is widely used in face and object detection, within the NPL module, and then a post-processing is performed followed by a tracker process. The tracker process is initialised when each new plate region is detected, which can optimise the detection process and create a relation between subsequent frames. Subsequently, the detected plates are handed over to the character recognition module. Each character is extracted by the segmentation process and recognised through classification process. Finally, history information is acquired by the tracker that will be used to improve the classification result in the post-processing unit. The flow chart of the overall system is shown below.

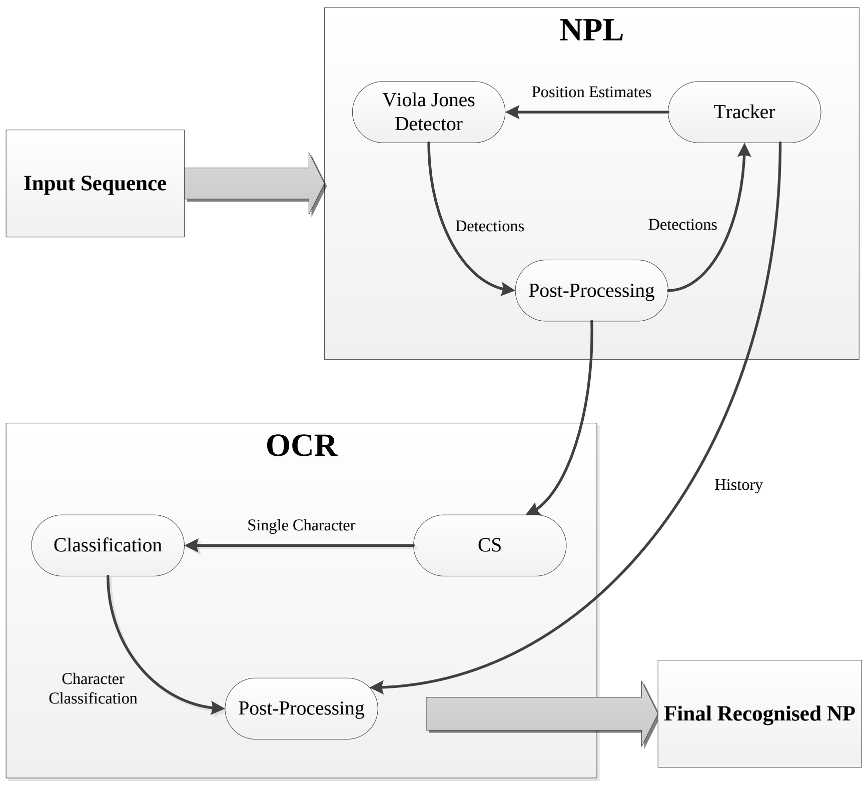


Figure 2-12: The overall system flow chart [11]

A frame resolution of 352×288 was used in the proposed system, the detection and recognition process operates on each individual frame, and then transmits the results and a compressed image to a client. The used classifiers for the Viola and Jones detector and the SVM are trained off-line on a general purposed computer, and then were ported to the DSP platform. Two sets of test data are used for the proposed system, one set contains 260 images of number plates is used for training, another set of test data is the extracted video frames that is used for testing system performance [11].

A single Texas Instruments^{TM} C64 fixed point DSP with 1MB of cache RAM and 16MB SDRAM was used for the experimental test. On the average, the SVM takes 7.3 *ms* for localisation of a full plate, and the achieved average frame rate is 19 fps (52.11 *ms*) [11].

2.5.2 Hybrid DSP /FPGA-based NPL System

A hybrid DSP/FPGA-based NPL system was presented in [12], where the FPGA is only used for buffering video frames/scanlines between the DSP and a video input processor placed inside the camera while the DSP is used as the main processing unit. The main features of the embedded platform are:

- One TMS320C6414 DSP with 600 MHz and 1MB cache;
- 128MB SDRAM for video compression, processing and storage of temporary data;
- 4MB Flash Memory for firmware storage;
- One video input processor;
- One FPGA for buffering video frames/scanlines between the video input processor and the DSP.

The Viola and Jones detector is applied for the NPL algorithm, which needs 140 *ms* to process a single 352×288 frame in a 15-minute demonstration video. Overall detection rate for 100 number plates is 96% [12].

2.5.3 FPGA-based ANPR System

In the work presented in [105], a video processing methodology for a FPGA-based ANPR system was proposed. During the design, Gabor filter, threshold and CCA algorithms were used for the NPL, the Self-organising map (SOM) was used to identify the characters. The system had been tested with a large database, and is suitable for applications where cost, compactness and efficiency are limited. The overall ANPR system block diagram is shown below in Figure 2-13.

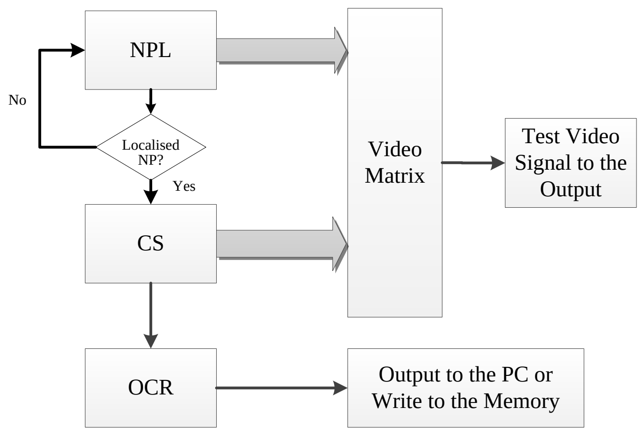

NPL
No
Localised NP?
Yes
CS
OCR
Video Matrix
Test Video Signal to the Output
Output to the PC or Write to the Memory

Figure 2-13: The overall ANPR system block diagram in [105]

In the NPL stage, the Gabor filter was first applied to the image to remove the unnecessary data, and then the image was converted into binary image using an optimum threshold. Subsequently, a dilation morphological operation was used on the result binary image to connect the NP region. Finally, a CCA algorithm was applied to extract the binarised NP within the previous binary image. At the end of the NPL process, if there is no localised NP, the same process would repeat.

In the CS stage, the characters and digits were segmented within the NP area obtained from the NPL stage. The pixel projection method was used to locate the character positions on the NP. At the end of the CS stage, the algorithm includes a checking process to validate the number of characters and based on the checking results, an adjustment of the critical point finding threshold was carried out to improve the segmentation result.

In the OCR stage, SOM was proposed to recognise the segmented characters. A SOM normally has two layers: input and computation layers, the weights of the SOM are calculated during the learning phase. The hamming distance between each neuron and the input image is calculated and makes a decision on the output character.

A Xilinx ML40x board equipped with a Virtex-4 FPGA is used and implemented for the experimental test. The system was tested with a database of 1436 car images obtained from highway and parking lots at various times of the day, it approximately needs 500 *ms* complete a single recognition and achieved 73% overall recognition rate [105]. The FPGA logic utilisation of the system is given in Table 2-4.

Table 2-4: FPGA Logic Utilisation

FPGA Logic Utilisation	Used
Flip-Flops used	43551
Number of 4-input LUT’s	50310

In this work, an FPGA based real-time and low-cost ANPR system was successfully designed.

2.5.4 FPGA-based NPL System

In the work presented in [13], a high speed FPGA off-loading engine for NPL system is proposed. The main goal of this work is to detect the NP itself and measure the size, which can then be used to measure distance between the cars. The flow chart of the proposed NPL algorithm is shown below in Figure 2-14.

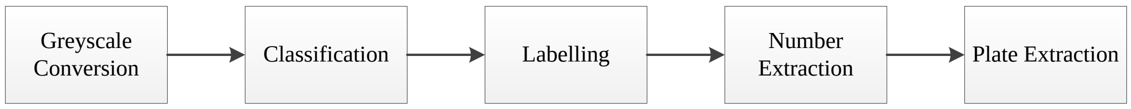


Figure 2-14: The flow chart for the NPL algorithm in [13]

In the beginning of the process, 24-bit RGB pixels are converted into 8-bit greyscale pixels, and then a classification is performed to classify each pixel into black, white and the others. For example, if the greyscale value of a target pixel is greater than its neighbour pixels, the target pixel will be classified into ‘white’, otherwise, it will be classified into ‘black’. In the labelling, each labelled region consists of the neighbouring pixels with the same colour. There are four conditions used to locate NP candidate regions within the resulted image, which are:

- The height is limited in the range from 14 to 160 pixels;
- Height to width ratio is from 1.25 to 20;
- The area of the number occupies 20% to 80% of the rectangle region;
- The noise within the rectangle area is less than 5% noise.

In order to localise the NP region, three extra conditions are used:

- The error rate of the height of numbers are less or equal to 20%;
- All the numbers are in the same line;
- The noise ratio is less than 5%.

The used 256×256 image data is horizontally divided into 16 segments without any accuracy loss. This process is shown below in Figure 2-15.

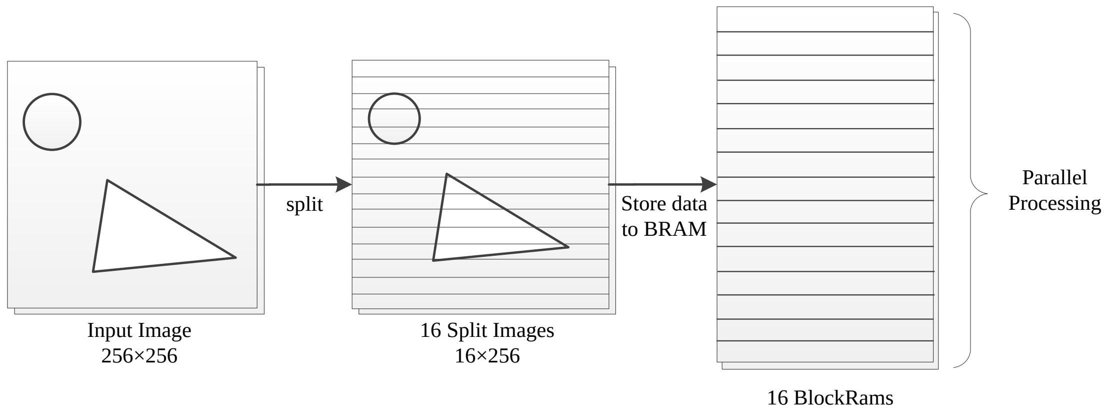


Figure 2-15: Division of the image [13]

The proposed system was implemented on a prototype FPGA development board called ReCSiP-2 equipped with a Virtex-2 (XC2VP70-5FF1517) FPGA [106]. The overall block diagram of the NPL system is illustrated in Figure 2-16.

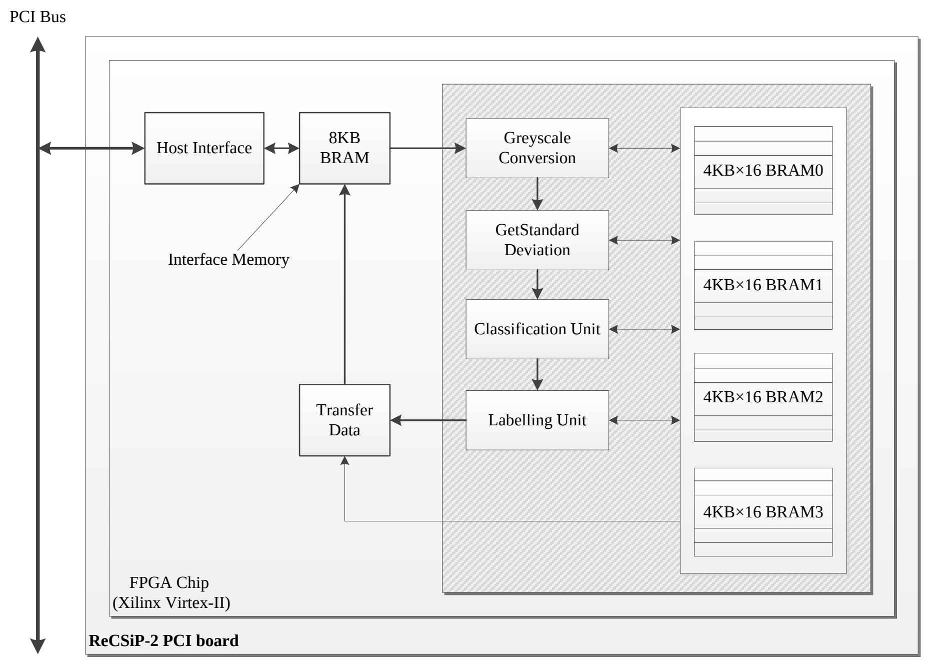


Figure 2-16: The overall block diagram of the NPL system [13]

The system was designed using Handel-C, and then translated into Verilog with Mentor Graphic DK4. The maximum frequency is 72.062 MHz. The required on-chip resources is summarised in Table 2-5.

Table 2-5: FPGA Logic Utilisation [13]

FPGA Logic Utilisation	Amount	Ratio (%)
Slices	22206/33088	67
BRAMs	88/328	26
Embedded Multipliers	28/328	8

In order to prove the concept and evaluate the performance of the FPGA based system, the same system was also implemented in software. The reported detection rate is 87.7% in day time and 84.1% in night time. A comparison between the software and hardware implementation is given in Table 2-6.

Table 2-6: Performance comparison of the software and hardware implementation [13]

Stage of the NPL System	Software (*ms*)	Hardware (*ms*)	Speed up
Greyscale Conversion	8.03	0.76	10.57
Classification	12.92	3.09	4.18
Labelling	17.57	5.40	3.25
Total	38.52	9.25	4.16

The proposed work has provided a high speed parallel solution for localisation of the NP on FPGA. However, the proposed system is only designed for the first stage of an ANPR system, and the proposed algorithm extracts the NP only using the illumination information, which may limit its performance for extracting the NP from images taken in complex environments with various objects and lighting environment.

2.5.5 Discussion

The high performance and low-cost System-on-chip solutions allow the entire ANPR system to be implemented on a single chip that can be placed within an ANPR camera housing to create a stand-alone unit thus drastically improving energy efficiency whilst removing the need for the installation and cabling costs associated with bulky PCs situated in expensive, cooled, waterproof roadside cabinets.

Generally, the software-based ANPR systems have a higher detection rate compared to hardware based systems, however the processing time of the former is higher than the latter. The design and implementation of hardware-based ANPR systems is limited by the hardware architecture and the available resources. Efficient methods and techniques to implement ANPR algorithms should be considered to map them on the chosen hardware. The performance of existing FPGA and DSP-based ANPR systems in terms of speed and recognition rate is listed in Table 2-7.

Table 2-7: Existing FPGA and DSP-based ANPR systems

ANPR System	Character Set	System Part	Image Type	Hardware Platform	Successful Rate (%)	Speed (*ms*)
[11]	Australia	Whole ANPR system	Greyscale	TI C64 DSP	85	52.11
[12]	Australia	NPL	Colour	TI C6414 DSP and Altera FPGA	96	140
[105]	Turkey	Whole ANPR system	Greyscale	FPGA Virtex-4	73	500
[13]	Japan	NPL	Greyscale	FPGA Virtex-2 and PC	87	9.25
[107]	US	Whole ANPR system	Greyscale	FPGA Virtex-2 and PC	N/A	N/A
[28]	UK	NPL	Greyscale	C64plus DSP	96.1	4.86

2.6 Limitation of Existing Work and Research Opportunities

As it can be seen from the preceding sections, most of existing ANPR systems are based on software implementation, only few solutions are based on hardware. There still remains plenty of scope for further research in exploiting reconfigurable computing for ANPR systems to improve the system performance and efficiency. The major limitations of the existing work can be summarised as follows:

- The existing ANPR algorithms required high performance platform to achieve the real-time constraint, the cost of the power consumptions were not discussed;
- A relatively small volume of recent ANPR systems were implemented in hardware;
- The existing ANPR algorithms were not specially designed for FPGA implementation, where parallelism could be exploited;
- The existing hardware-based ANPR systems were only highlighting the speed, however, most of them did not achieve a satisfactory recognition rate, which limited the practical usage of the solutions.

Based on the limitations of existing work, the main contributions of the work presented in this thesis can be summarised as follows:

- A low complexity and robust NPL algorithm suitable for a single FPGA implementation has been developed in the thesis, where a novel NP feature extraction and enhancing method based on two morphological open operations and an image subtraction operation are proposed. In addition, a novel efficient FPGA architecture and its area/speed efficient implementation have also been proposed in the thesis;
- A low complexity and robust CS algorithm based on pixel projection and morphological operations has been presented in the thesis, where an NP height optimisation step and two optional morphological operations are introduced to reduce and remove noise impact to achieve a more precise horizontal and vertical segmentation result. In addition, a novel real-time architecture and its area/speed efficient implementation have also been proposed in the thesis.
- A low complexity and robust OCR algorithm based on feed-forward neural network has been presented in the thesis, where the use of noise added training process could result on the neural network with better performance than the normally trained neural network. In addition, a novel real-time architecture and its area/speed efficient implementation have also been proposed in the thesis; and
- A local thresholding NP binarisation and low complexity NP adjustment algorithms have been introduced to solve the problem of uneven illumination and automatically adjust NPs respectively in the thesis, which could improve the NPL result prior to CS stage. In addition, two area/speed efficient architectures based on the proposed NP binarisation and adjustment algorithms have also been presented in the thesis.

2.7 Conclusion

This chapter has summarised the state-of-art algorithms, architectures and systems for

ANPR application implemented on software and hardware-based platforms using different design methodologies and implementation approaches. In addition, limitations of existing work were stated. It is the aim of the research work presented in this thesis to address the limitations presented in the previous sections with efficient means of providing high performance ANPR system through the use of FPGA.

Chapter 3:Number Plate Localisation Algorithm and its Efficient FPGA Implementation

3.1 Introduction

As mentioned in the previous chapter, the NPL stage is crucial to the entire system, because it directly influences the accuracy and efficiency of the subsequent steps (i.e. CS and OCR). The NPL stage is also the most computational intensive stage in the entire ANPR system. For a real-time ANPR system, efficient and robust NPL algorithms are required to accelerate the speed of the entire system.

This Chapter is concerned with the NPL stage, where a speed and area-efficient architecture based on a low complexity NPL algorithm suitable for FPGA implementation is presented. The proposed algorithm is mainly based on morphological open and close operations, which replaces the traditional edge detection operator to reduce the computation complexity whilst maintaining a satisfactory detection rate. A MATLAB implementation of the proposed algorithm is used as a proof of concept prior to the hardware implementation, and the proposed architecture implemented and verified using the Mentor Graphics RC240 FPGA development board equipped with a 4M Gates Xilinx Virtex-4 LX40. For comparison purposes two different databases, including a public one, were used. The first one contains1000 images with UK NPs while the second one , taken from an online database, contains 307 images with Greek NPs with a resolution of 640×480 [108]. For the UK database, some images were collected by the author with the rest provided by CitySync Ltd. [109] who are one of the leading UK providers of ANPR solutions. The images were grouped into six different sets based on different criteria such as distance and illumination conditions, the proposed algorithm and FPGA implementation have been tested using both databases, however, due to private data protection is required by CitySync, only authorised image samples are exhibited in the thesis.

The remainder of this Chapter is organised as follows: Section 3.2 describes the morphological based algorithm. The proposed NPL architecture is then described in Section 3.3. The MATLAB and analysis of the experimental results are given in Section 3.4. Section 3.5 is concerned with FPGA implementation and discussion of the experimental results. Section 3.6 discusses the conclusions of this chapter.

3.2 Number Plate Localisation Algorithm

A NP image is normally recorded as a pattern with high variations of contrast. This feature is used to locate the plate and has been found to be relatively robust to changes in lighting conditions and view orientation. Most of the previous works that were based on morphological operations have used edge detection to extract the edge information around the NP region followed by morphological operations as a fusion tool to connect the pixels together in that region [5]. After that a Connected Component Analysis (CCA) labelling algorithm is used for the NP region selection. However, the edge detectors are based on matrix multiplication and the entire image needs to be scanned, which increases the computing cost of the algorithm. Therefore, in this section, a morphological *open* operation and image subtraction are used to replace the edge detection operator, which reduces the computation complexity whilst maintaining a satisfactory detection rate.

The proposed algorithm is mainly based on two *open* and one *close* morphological operations, the first *open* morphological operation is used to extract the features of the NP, the second *open* operation is used to remove noise, and the *close* operation is then used to fuse the pixels in the NP region together.

The proposed algorithm consists of two major stages:

1. Morphological operations for extracting plate features;
2. Selection of candidate regions.

Figure 3-1 shows a block diagram of the proposed NPL system.

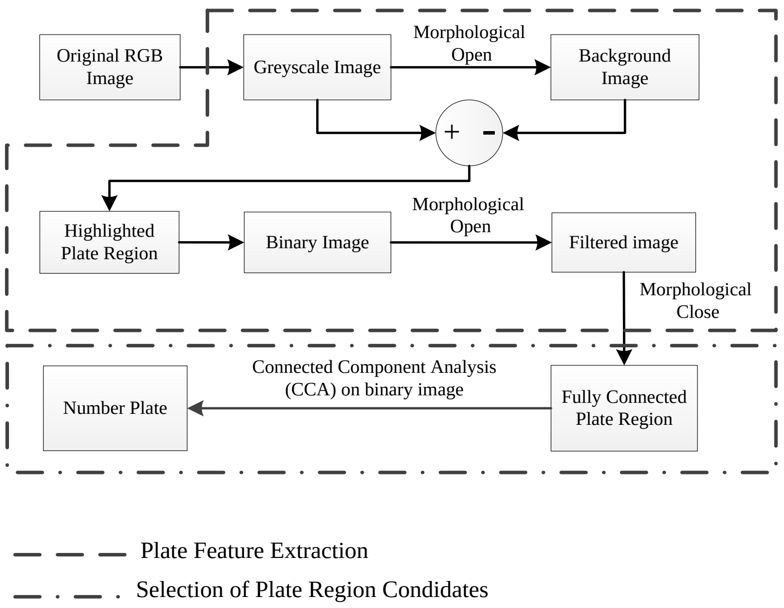


Figure 3-1: Block diagram of NPL system

3.2.1 Plate Feature Extraction

The proposed algorithm mainly utilises three morphological operations to minimise the pixels of the non-plate region and to enhance those of the plate region. The original RGB image is first converted into a greyscale image, which will be used as an input to the following block where the first morphological *open* operation is used.

The morphological *open* operation is an *erosion* followed by a *dilation* and the opposite operation (i.e. *close* operation) is a *dilation* followed by an *erosion*. The shape of the morphological operations is based on a suitable structuring shape employed as a probe called the Structuring Element (SE) [110]. *Open* I_O and *close* I_C operations can be performed as shown in Equation 3.1 and 3.2 respectively where *I* denotes a greyscale input image, $\oplus$ denotes a *dilation* operation and ! denotes an *erosion* operation:

$$I_o = (I\,!\;SE)\oplus SE \tag{3.1}$$

$$I_c = (I\oplus SE)\,!\;SE \tag{3.2}$$

When applying the morphological *open* operation on a greyscale image, pixels are ‘averaged’ in the area of SE. When applying it on a binary image, pixels are erased if the SE area is not fully filled by pixels with value ‘1’.

UK NPs normally consists of black characters on a white background. This feature causes the pixel values to be highly variant in the NP region. On the contrary, the margin area of the NP region normally consists of a constant colour, in particular the windscreen glass and engine hood. Therefore, if applying a morphological *open* operation with enough large SE on greyscale car image, characters can then be removed from the NP region while the remaining features of the rest of image are kept. By performing a subtraction between the original greyscale image and the resulting image after the *open* operation image, the output is a highlighted plate region image. This process can be summarised in the following pseudocode.

Proposed algorithm: NP feature extraction

```
1. Input image: colour car image
2. Output image: Highlighted NP region image
3. for all pixels in the input image do
4. grayscale pixels = RGB2Gray(original colour pixels);
5. Shifting the pixels into SE;
6. background pixels = open(SE);
7. highlighted NP pixels =   grayscale pixels - background pixels
8. end
```

Compare to the existing algorithms, the proposed algorithm uses a morphological *open* operation and image subtraction to replace the edge detection operator. As morphological open operation has less computational intensity than the edge detection operator, which could significantly improve processing speed. In addition to the above, with a specially designed SE, the proposed algorithm could extract more accurate NP features rather than only focus on common edge features, which should improve the ability of proposed algorithm for tolerating the noise.

The size of SE is decided based on the gap between two neighbouring characters on the NP. Due to the variant distances between the car and the camera, the size range of NPs ($H \times W$)

in the used databases is between 18×160 (pixels) and 60×300 (pixels). Let $d_{h\max}$ (pixels) and $d_{v\min}$ (pixels) denote the maximum distance between the two neighbouring characters on the horizontal and the minimum distance between the character and boundary of the NP respectively. They both depend on the size of the NP and the shape of neighbouring characters (See Figure 3-2).

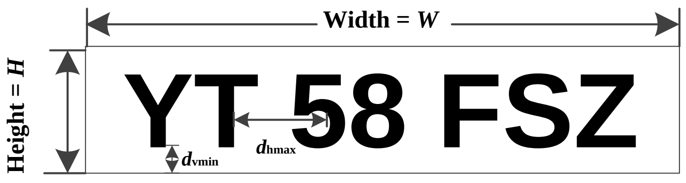


Figure 3-2: An NP example

On the other hand, the size of SE ($S_1 \times S_2$) is determined by $d_{h\max}$ and $d_{v\min}$, where $1 \le S_1 \le d_{v\min}$, $d_{h\max} \le S_2 \le d_{h\max} + \Delta$, Δ as a variable that is calculated based on experiment results.

For the UK database, images were randomly taken from different real-world environments with variant NP sizes (note that details of this database are given in Section 3.5). Based on the above description on how the SE size is selected and tests performed using the UK database and then validated with the on-line public Greek database, the size of the used SE for the *open* operation was set to 3×30. Figure 3-3 shows the used ‘rectangle’ shaped SE.

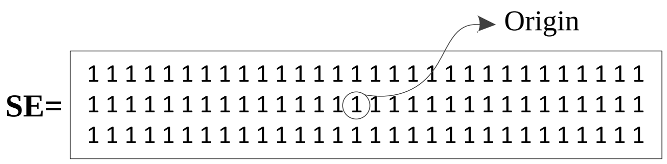


Figure 3-3: A ‘rectangle’ shaped SE with size $= 3 \times 30$

This 3×30 SE has an origin pixel point, which is the centre of the whole SE. The origin point is mainly used for marking the SE’s location when the morphological operation is performed.

A morphological *open* operation with this 3×30 SE is performed on an original greyscale image, which will generate a background image (Non-NP region). The background image is subtracted from the original greyscale image and the result of this operation is a highlighted NP region. Figure 3-4 illustrates this process.

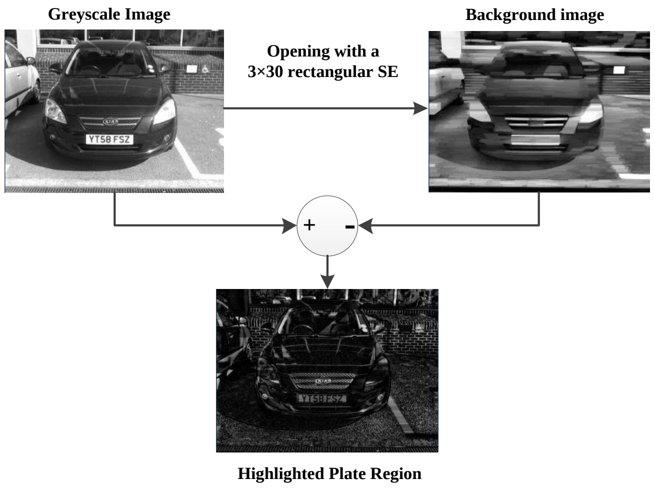


Figure 3-4: The process for highlighting the plate region.

In order to further eliminate Non-NP regions, the highlighted plate region image is binarised. Let $g_{\min}$ and $g_{\max}$ denote the minimum and maximum pixel value of the highlighted NP region in the database respectively, the best threshold T_b should adaptively change from $g_{\min}$ to $g_{\max}$ when different images are applied, however, this process requires extra memory to store the entire image, before analysing the best T_b for each input image. For FPGA implementation, this slows down the processing speed and increases hardware usage. Therefore, the proposed algorithm uses a fixed threshold T_f to replace T_b. If $T_f \leq g_{\min}$, all the highlighted NP regions should be kept after image binarisation. For the used databases $g_{\min} \geq 60$, therefore the value of the fixed threshold T_f is 60. Although the fixed threshold can benefit hardware implementation, a lower threshold value will increase the noise level (i.e. area of Non-NP regions). To overcome this problem, an extra morphological *open* operation is used to remove the noise. Figure 3-5(b) shows the result after noise removal.

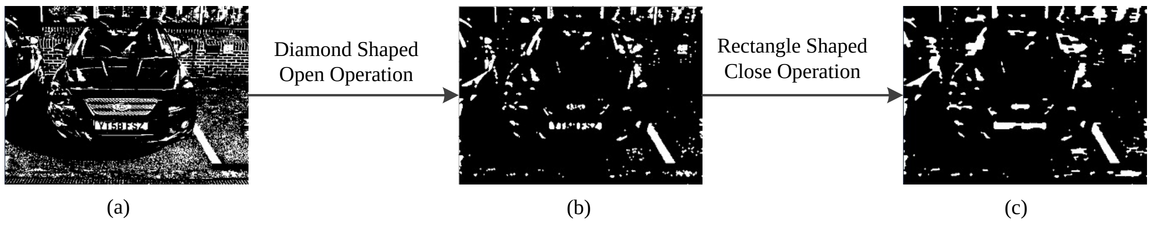


Figure 3-5: The process of image binarisation and enhancement. (a) Binarised Image, (b) Image after Open Operation, (c) Image after Close Operation

For the process shown in Figure 3-5 ‘diamond’ and ‘rectangle’ shaped SEs are used for the last morphological *open* and close operations. The two SEs are shown in Figure 3-6.

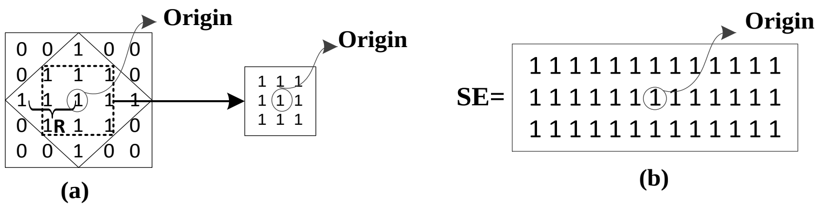


Figure 3-6: The ‘diamond’ shaped and ‘rectangle’ shaped SEs. (a) ‘diamond’ shaped SE with R=2 and 3×3 ‘rectangle’ shaped SE, (b) A 3×13 ‘rectangle’ shaped SE

Figure 3-6 (a) shows a ‘diamond’ shaped SE, where the matrix has a radius R=2 and all 1’s are inside the ‘diamond’. When this special structure is used during the *open* operation on a binary image only diamond-shaped regions filled by 1s are kept. This operation is very useful in erasing net-shaped and narrow lines surrounding the plate area. This SE can efficiently erase most of the unwanted information, as can be seen in Figure 3-5 (b). However, in order to reduce hardware usage, the ‘diamond’ shaped SE has been replaced by a 3×3 ‘rectangle’ shaped SE for hardware implementation. As can be seen from Figure 3-6 (a) the difference between the ‘diamond’ shaped SE and the ‘rectangle’ one is that the first has an extra four corners. Although the *open* operation can effectively remove noise, some pixels in the NP region can also be eliminated. Therefore, the system needs an extra operation to fully fill the plate region to connect the pixels. A morphological *close* operation is used for this purpose. Figure 3-6 (b) shows a ‘rectangle’ shaped SE for the *close* operation, where the matrix has 3×13 ‘rectangle’ shaped 1s. Any non 1 pixels in this rectangle region are changed to 1, which means all the parts in this region are fused together. As can be seen from Figure 3-5 (c), the plate region can clearly be identified as it is a group of connected pixels which can be easily extracted using some known geometrical conditions (e.g. Width / Height ratio).

3.2.2 Selection of Candidates Plate Region

The output image from the previous stage consists of a set of groups of connected pixels. A labelling algorithm CCA is used to mark these pixels. In the proposed work, the CCA uses a '4-connectivity' method, and labels them using different numbers. Once all the groups of pixels have been determined, each pixel is labelled based on the group it belongs to. Therefore, a set of potential candidates can be selected from the image using the known geometrical conditions, which mainly consist of the width, height and ratio of the plate region. Let P denote the extracted plate region with the size $H \times W$, the first criterion is the ratio R between the height and width of P (i.e. $R = W/H$). The second criterion is the range of H and W. The third criterion is the area of P. Ranges for H, W and R were selected to be relatively large enough to cover most of the possible sizes of the plate region in the databases. Basically, there are two selection conditions (Condition 1 and Condition 2) used for this purpose. For both conditions, the width, height, area and ratio of the NP are considered. Condition 1 is stricter than Condition 2 where some of the candidates may not meet Condition 1 but can meet Condition 2. The maximum and minimum coordinates of the rectangular plate regions that pass one of the conditions are returned. Normally, the strictest condition (i.e. Condition 1) is perfectly suited for selecting candidates from good condition images (e.g. daytime and clear images); while Condition 2 can be used for selecting candidates from bad quality images (e.g. far view, blur and complex background images). Figure 3-7 shows a flow diagram that illustrates the selection process.

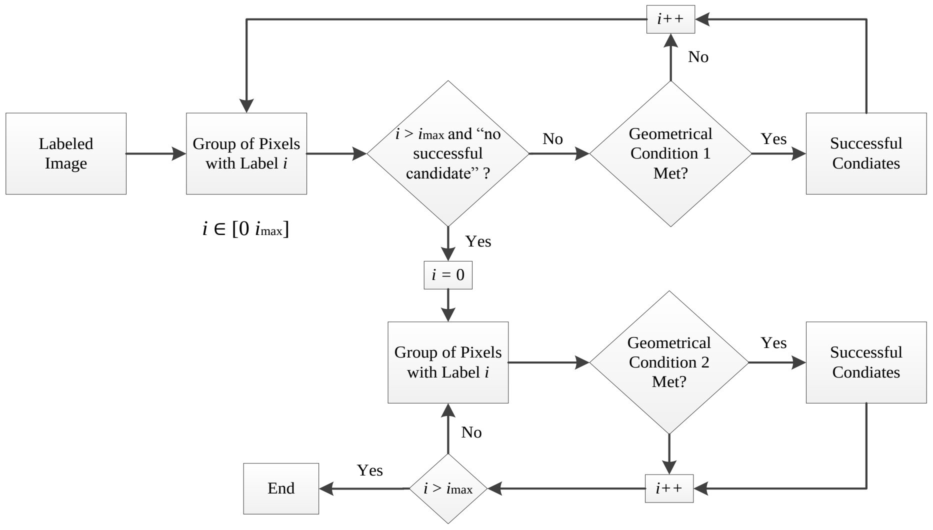

Labeled Image
Group of Pixels with Label i
$i \in [0\ i_{max}]$
$i > i_{max}$ and "no successful candidate" ?
No
Geometrical Condition 1 Met?
Yes
Successful Condiates
i++
No
Yes
$i = 0$
Group of Pixels with Label i
Geometrical Condition 2 Met?
Yes
Successful Condiates
No
End
Yes
$i > i_{max}$
i++

Figure 3-7: Flowchart of selection process

The final NP will be extracted from original greyscale image. Figure 3-8 shows the selected NP.

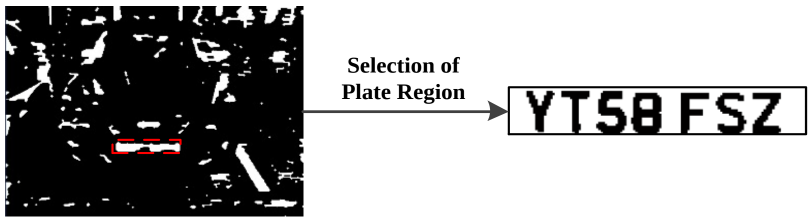


Figure 3-8: Selection of Number plate

3.3 Proposed Number Plate Localisation Architecture

Morphological operations based architecture consists mainly of an image reader, three morphological operations and CCA. Therefore, this architecture can be designed using the following modules:

- Memory Reader Module;
- Converter Module;
- Morphological Operations Module; and
- CCA Module.

The structure of the proposed architecture is shown in Figure 3-9.

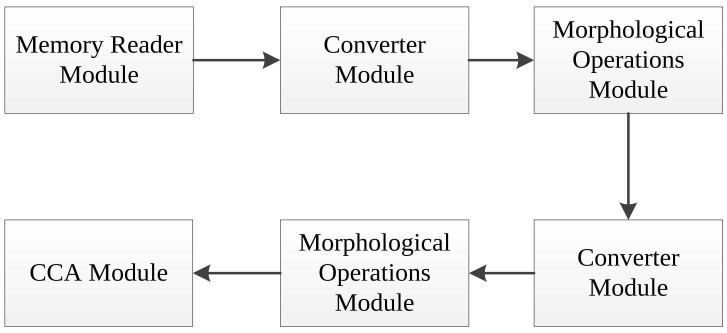

Memory Reader Module
Converter Module
Morphological Operations Module
Converter Module
Morphological Operations Module
CCA Module

Figure 3-9: Morphological operations based system

The first morphological operations module consist of two *open* operations, the second morphological operation module only consist of a *close* operation. The modules shown in Figure 3-9 are discussed in the following subsections.

3.3.1 Memory Reader and Converter Module

The first module in the proposed architecture is the memory reader and converter. The memory reader part of the module is used to read the RGB values for each pixel from the original RGB image which has a size of 640×480 and to assign a position coordinate. Figure 3-10 shows a block diagram of the memory reader.

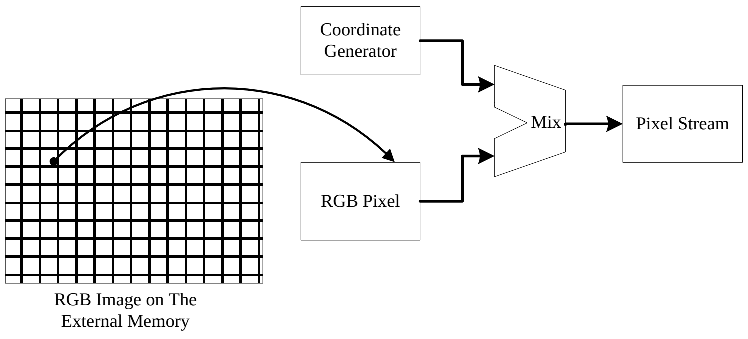


Figure 3-10: Block diagram of memory reader

The converter part of the module is used for the standard RGB (24 bits) to greyscale conversion (8 bits) using Equation 3.3 [111]:

$$Y = \frac{R \times 77 + G \times 155 + B \times 29}{256} \tag{3.3}$$

This module is also used for the greyscale to binary conversion using a fixed threshold T_f out of 255 (i.e. $T_f = 60$), which means all values less than T_f will be treated as '0' and values larger or equal to T_f will be treated as '1'.

3.3.2 Morphological Operations Module

The morphological operations module consists of the morphological *open* and the morphological *close* sub-modules. According to the Equation (3.1) and (3.2), the morphological *open* operation and the morphological *close* operation can be divided into two sub-filters respectively, i.e. the morphological *dilation* and the morphological *erosion* sub-filters, where the order in each case decides whether the morphological operation is *open* or *close*. The greyscale *dilation* calculates the maximum pixel value in a specific SE. On the contrary, the greyscale *erosion* calculates the minimum value in a specific SE.

The proposed algorithm uses 3×30 rectangle shaped SE, however, for efficient hardware implementation where pipelining can be exploited, this rectangular shaped SE has been decomposed into two small rectangle SEs with the sizes 1×30 and 3×1. Figure 3-11 shows the block diagram of the proposed pipelined *dilation* filter.

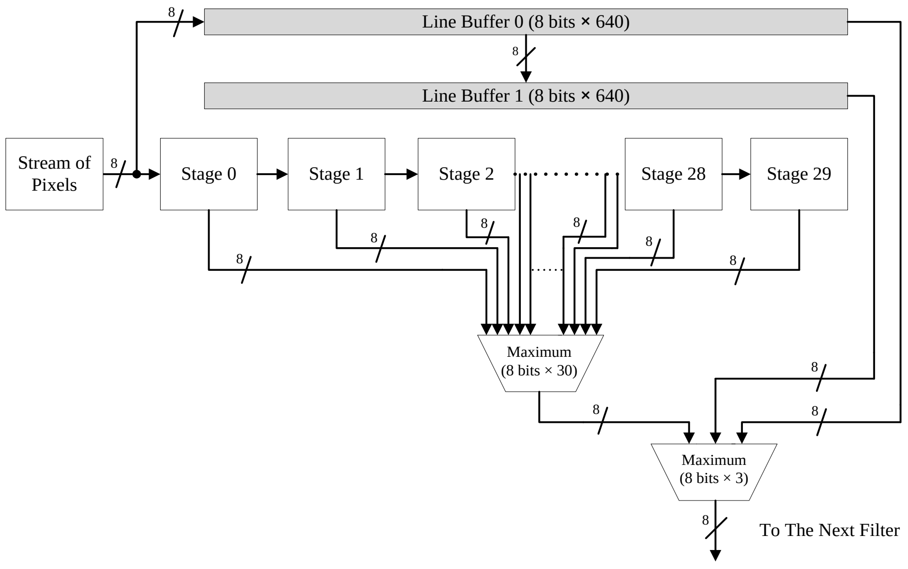


Figure 3-11: The block diagram of a pipelined *dilation* filter

The process starts when the value of current input pixel is simultaneously passed into the internal buffers “Stage 0” and “Line Buffer 0” then after every clock cycle it is passed to the next stage until it reaches “Stage 29” and then the maximum pixel value of the current 30 pixels in the 30 stages is calculated. In the meantime, the values of the pixels from two consecutive lines of the greyscale image (i.e. 640 pixels per line) are stored in the two line buffers in order to calculate the maximum value from three consecutive pixels from the same column. The first origin of SE (1×30) is the fifteenth pixel of the first line, so the first coordinate of output should be kept consistent with the coordinate of the fifteenth pixel instead of the coordinate of the current input pixel.

The structure of the *erosion* filter is similar to the *dilate* filter. The only difference is that the minimum value of the pixels is calculated instead of the maximum one.

In the proposed architecture, there are three different SEs used for the three morphological operations (i.e. ‘rectangle’ shaped SEs: 3×3, 3×13, 3×30) which can be easily implemented using the block diagrams shown in Figure 3-11 and Figure 3-12 by simply changing the number of stages (i.e. if the size of SE is 3×3, it requires three stages). The ‘diamond’ shaped SE has been replaced by the ‘rectangle’ shaped SE (3×3) in order to use the same block diagrams shown in Figure 3-11 and 3-12 which reduces the hardware complexity.

3.3.3 CCA Module

The CCA module is used to mark and select a candidate plate region from the entire binary image. Generally, the pixels of the input pixel stream are divided into several groups or blobs by the CCA module. The grouping is based on the pixels’ connectivity. Figure 3-13 demonstrates this procedure.

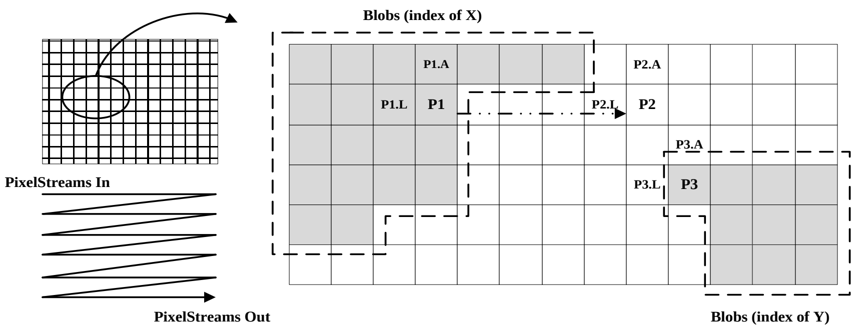


Figure 3-12: The block diagram of CCA

The grouping is performed as follows. The binary stream is scanned from left to right starting from the top line. For instance, a comparison between the current pixel “P1” from Figure 3-13, its upper pixel “P1A” and left pixel “P1L”, which have already been grouped, is performed. All pixels with value ‘0’ will be assigned to one group with an index ‘0’. If the value of “P1” is ‘1’ and the indexes of its neighbours are the same and not ‘0’ then “P1” will be assigned the same index as its neighbours. If the indexes of the two neighbours are different and not ‘0’, then the indexes of this pixel and its upper neighbour “P1A” will be the same as its left neighbour (i.e. “P1L”). If the indexes of the two neighbours are different and one of them is ‘0’, then the index of this pixel will be the non-zero index of its neighbour. If the pixel value is ‘1’ but the indexes of its neighbours are both ‘0’, the index of a new group will be assigned to this pixel. Finally, the coordinates of each rectangular shaped group are recorded for the selection of candidates.

Once the whole image is scanned, the selection of a candidate region is performed using the selection process shown in Figure 3-7 which is mainly based on the geometrical relationship of the NP region.

3.4 MATLAB Implementation and Results

The proposed algorithm was first tested in a MATLAB environment using a database of 1000 images containing UK NPs and verified using an on-line public database of 307 images containing Greek NPs. The resolution of all used images is 640×480. The UK number plate database consists of six different sample sets and the on-line Greek database consists of three different sample sets, which are taken from natural scenes obtained in various illumination conditions and different distances between the camera and vehicles. The three sample sets from the Greek database are similar to the first three sample sets from the UK database. Therefore, for the purpose of performance testing of the proposed algorithm, all samples sets from both databases were grouped into six sets. The first three sample sets are:

- Sample Set 1: day time colour: this set contains 631 images from the UK NP database and 136 from the Greek one. The NP regions in this sample set are clear and normal size, which were taken from the front view of the cars at day time with various illumination environments;
- Sample Set 2: day time close view: this set contains 70 images from the UK NP database and 122 from the Greek one. The size of the NP regions in this sample set is large and the images contain less complex background environment information, which were taken from the close view of the cars at day time with various illumination environments; and
- Sample Set 3: day time with shadows: this set contains 68 images from the UK NP database and 49 from the Greek one. The NP regions and the backgrounds contain shadows, which were taken at day time with various illumination environments.

The remaining three sample sets are:

- Sample Set 4: day time moving vehicles: this set only contains 140 moving vehicle images from the UK NP database, which were taken from the motor way at day time with various illumination environments;
- Sample Set 5: day time far view: this set contains 75 images from the UK NP database. The size of the NP regions in this sample set is small and the images contain more complex background environment information, which were taken at day time but with longer camera-subject distances;

- Sample Set 6: night time infrared: this set contains 17 images from the UK NP database, which were taken from an infrared camera at night time.

Table 3-1 shows images from each sample set and the size range of the NP, where the lowest and highest height/width of NP are 18/160 and 60/300 respectively in the databases. Therefore, the expected H, W and R values should fall in the following regions: $18 < H < 30$, $60 < W < 300$ and $2 < R < 9$.

Table 3-1: The Samples of Used Database

	Sample set 1 (Day time colour)	Sample set 2 (Day time close view)	Sample set 3 (Day time with shadows)	Sample set 4 (Day time moving vehicles)	Sample set 5 (Day time far view)	Sample set 6 (Night time infrared)
NP sizes	30×160 up to 40×220	42×230 up to 60×300	26×120 up to 42×230	30×160 up to 38×200	18×160 up to 30×160	30×160 up to 38×200
UK						
Greek				N/A	N/A	N/A

Table 3-2 shows the MATLAB implementation results in terms of NPL rate using all sample sets.

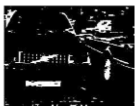
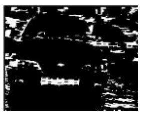
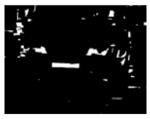
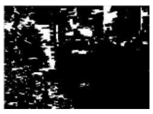
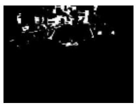
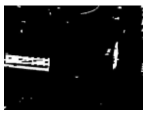
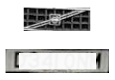
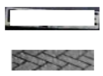
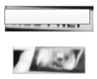
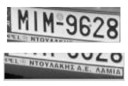
Table 3-2: Successful NPL Rate by Sample Sets (MATLAB Implementation Results)

Database	Sample set 1	Sample set 2	Sample set 3	Sample set 4	Sample set 5	Sample set 6	Overall
UK database	619/631 (98.1%)	69/70 (98.6%)	66/68 (97.1%)	135/139 (97.1%)	73/75 (97.3%)	17/17 (100%)	979/1000 (97.9%)
Greek database	133/136 (97.8%)	120/122 (98.3%)	48/49 (97.9%)	N/A	N/A	N/A	301/307 (98.0%)

The proposed algorithm has an overall 97.9% NPL rate when tested using the UK images and 98.0% when using the Greek images. The NPL rate is high for sample sets 1, 2 and 6 compared to sample sets 3, 4 and 5, which is due to the fact that the scenes in the latter sample sets contain more complex background environments. Generally, the proposed algorithm shows a similar NPL result and a relatively stable performance for both databases.

Although the two geometrical conditions have effectively improved the NPL rate, some images still cannot be handled successfully. Generally, there are two main failed data image sets (see Table 3-3): (1) Images with more than one successful candidate including the NP itself. (2) Images with no successful candidate. The main reasons for the first set are environment background and NP selection conditions. Since various illumination conditions and the range of NP size is very large (18×160 to 60×300), two selection conditions cannot fully cover all NPs. In some cases, the false candidates in some images are very similar in size to the true candidates in other images in the database which cannot be excluded. In order to overcome this problem, a validation process should be added before character segmentation for cases where there is more than one successful candidate. For the second set, there are no successful candidates due to the length of the distance between the camera and the car which results in very small NP images and an increase in the background noise~~s~~. In this situation the NP feature cannot be extracted properly by the proposed morphological operations.

Table 3-3: Failed Images in Both Databases (MATLAB Implementation)

	(1)			(2)		
Original Image						
Image before CCA						
Detected NP				No successful candidate	No successful candidate	

3.5 FPGA Implementation and Results

The proposed architecture for NPL has been simulated in PAL Virtual Platform (PALSim) [112]. After simulation, the architecture has been successfully implemented and verified using the Mentor Graphics RC240 FPGA development board [113]. Handel-C and PixelStreams, which is a library that can be used for rapid development of video image streaming applications, have been used for the hardware description of the proposed architecture [114]. The details of the experimental tools can be found in APPENDIX B.

The original RGB image is first stored in an external memory on the RC240 board. The external memory data width is 32 bits, which means every pixel value (24 bits) can be saved on a single memory location. In Figure 3-10 each RGB pixel is combined with its corresponding position coordinate and synchronisation information and then sent to the filter blocks previously outlined in Figure 3-1 running in parallel. Every clock cycle, one processed data pixel is passed from one block to the next. Example codes of the NPL implementation can be found in APPENDIX C.

Both UK and Greek databases have been used for testing and validating the FPGA implementation. The results show a similar performance compared to the software implementation in terms of NPL rate where the entire overall rate is 97.8%. As floating-point arithmetic is used in MATLAB implementation, it has slightly better performance compare to FPGA based fixed-point arithmetic implementation. Table 3-4 shows the FPGA implementation results when using all sample sets.

Table 3-4: Successful NPL Rate by Sample Sets (FPGA Implementation Results)

Database	Sample set 1	Sample set 2	Sample set 3	Sample set 4	Sample set 5	Sample set 6	Overall
UK database	618/631 (97.9%)	69/70 (98.6%)	66/68 (97.1%)	134/139 (96.4%)	73/75 (97.3%)	17/17 (100%)	977/1000 (97.7%)
Greek database	133/136 (97.8%)	120/122 (98.3%)	48/49 (97.9%)	N/A	N/A	N/A	301/307 (98.0%)

3.5.1 Hardware Usage, Running Frequency and Power Consumption

Due to the low complexity of the proposed algorithm, the proposed architecture requires only 33% of the on-chip FPGA resources. Table 3-5 summarises the required on-chip resources.

Table 3-5: Usage of FPGA on-chip Resources

On-chip Resources	Used	Available	Utilisation
Occupied Slices	6,195	18,432	33%
LUTs	8,871	36,864	24%
Flip-Flops	4,088	36,864	11%
BRAMs	18	96	18%

33% of the on-chip FPGA slices are used to implement the proposed NPL architecture. In these slices, 24% LUTs are used to implement logic operations and RAMs in the design. 11% flip-flops are mainly used as register to buffer the data for enabling the high throughput pipeline manner in the design. 18% BRAMs are mainly used to store the image pixels for the morphological operations. The total 33% on-chip resource usage leaves 67% to be used for implementing the next stages of an ANPR system (i.e. CS and OCR).

The maximum running frequency is 86 MHz and the number of clock cycles needed for one image to be processed is 401247. The execution time for processing one frame can be roughly calculated using the following equation:

$$T = \frac{c}{f} \tag{3.4}$$

where *T* is the execution time in *ms*; *c* is the number of clock cycles needed for one image; and *f* is the maximum running frequency in Hz.

Based on Equation (3-4), the proposed architecture can process one image and produce a result in 4.7 *ms*. This means that the proposed architecture satisfies the minimum requirement for real-time processing. The result achieved in terms of maximum running frequency and area used for implementing this important part of an ANPR system shows that there is enough room to implement the whole ANPR system on one FPGA.

The power consumption of the designed circuit has also been analysed using Xilinx XPower Analyser [115], and the results obtained are shown in Table 3-6.

Table 3-6: Estimation of Power Consumption

Resource Type	Value of Power (mW)
Clocks	202
Logic	8
Signals	4
BRAMs	10
IOs	163
Clock Managers	157
Leakage	348
Total Power	892

The total power consumption of FPGAs consists of quiescent and dynamic components. The quiescent power is consumed due to transistor leakage. The dynamic power is consumed by fluctuating power as the design runs, i.e. clock power, logic power, signal power, BRAMs power and IOs power, which are directly affected by the chip clock frequency and the usage of chip area [115]. The total dynamic power consumption of the proposed architecture is 339 mW out of the total power consumption 892 mW.

3.5.2 Comparison with Existing Work

A comparison of the experimental computational speed and NPL rate with existing PC, DSP and FPGA based implementations of NPL is shown in Table 3-7.

Table 3-7: Performance Comparison

NPL Systems	Platform	Processor Clock Speed (MHz)	Image Resolution (pixels)	NPL Time (*ms*)	NPL Rate (%)
Proposed System on FPGA	FPGA Virtex-4	86	640×480	4.7	97.8%
Proposed System on PC	PC	2300	640×480	143	97.9%
[47]	PC	1700	768×534	100	99.6%
[12]	DSP C6414 and FPGA	600	352×288	141.62	96%
[13]	FPGA Virtex-2	72.062	256×256	9.25	87%

The proposed system outperforms existing ones as it shows a higher NPL rate and faster NPL speed with higher resolution compared to the databases used in systems [12] and [13] on the table. Although the testing databases used for the three methods are different, the proposed system has been tested and verified using a large local database and an on-line public database and shows stable results. However, it should also be noted that the databases used in [12] and [13] are not available as they are not public databases, but the used databases contain similar cases like the ones presented in those works.

However based on the results published, when compared to system [47], the proposed work has a faster NPL speed but slightly lower NPL rate. This is due to the fact that fixed measures of distance and angle, based on prior knowledge, have been used for the algorithm used in [47], which is based on edge detection and morphological operations. This prior knowledge boosts the results to a high level of accuracy which is not the case for the proposed algorithm which uses images taken from different distances and angles more reflective of real life recordings.

The proposed method in [12] uses a DSP for NPL implementation and a FPGA for buffering video frames between a video input processor and the DSP. Although the DSP frequency is 600 MHz, the processing time for one image is higher than the one for the proposed system. This is due to the fact that the proposed architecture is fully parallelised and requires less clock cycles which significantly increases the NPL speed.

By comparing the results of the PC and FPGA-based implementations of the proposed algorithm, it can clearly be seen that the latter outperforms the former with a 30-time speed-up with close accuracy; therefore, the proposed FPGA-based system can be used as a viable solution to replace software based solutions where cost, size and energy consumption will be reduced.

3.6 Conclusion

Recently, FPGAs have become a viable solution for performing computationally intensive tasks. Owing to the importance and the use of ANPR systems in law enforcement, an efficient NPL algorithm has been proposed in this Chapter for FPGA implementation. The algorithm is based on morphological operations and is multiplier/divider-free and requires only 33% of the available on-chip resources of a Virtex-4 FPGA. Parallel building blocks have been used for the FPGA implementation and the whole system runs with a maximum frequency of 86 MHz and is capable of processing one 640×480 image in 4.7 *ms* with a localisation rate of 97.8%.

Chapter 4: Number Plate Character Segmentation Algorithm and its Efficient FPGA Implementation

4.1 Introduction

In the previous chapter, a morphological operation based NPL algorithm and its efficient architecture implementation have been introduced. The next main stage of an ANPR system is CS stage, where the characters within the NP are correctly and accurately segmented. In order to achieve a good result in the CS stage, the NP normally be properly rotated and binarised at the end of the NPL stage before it is used for CS. Improved algorithms and new FPGA architectures for NP binaristion and rotation are presented in Chapter 6. In this Chapter, localised NPs are considered binarised and adjusted. Traditional pixel projection based CS algorithms have difficulty handling characters which are not at the same horizontal level, especially when the NPs are taken from different camera views. Furthermore, noise also significantly affects the results of CS. This Chapter presents an improved CS algorithm which uses pixel projection and morphological operations to improve the processing time and remove noise impact to achieve a more precise horizontal and vertical segmentation result. A MATLAB implementation of the proposed algorithm is used as a proof of concept prior to the hardware implementation. An area/speed efficient architecture based on the proposed algorithm is also presented, where parallelism offered by FPGAs and pipelining technique has also been exploited to achieve high running frequency and throughput rate. The use of multipliers has been avoided in some building blocks from the proposed architecture which significantly reduces on-chip resources usage and power consumption. The proposed architecture is implemented and verified using the Mentor Graphics RC240 FPGA development board (see APPENDIX A). A database of 1000 UK binary NP images with varying resolutions is used for testing the performance of the proposed architecture.

The remainder of this Chapter is organised as follows, Section 4.2 describes the proposed algorithm. The proposed CS architecture is described in Section 4.3. The MATLAB implementation and analysis of the experimental results are presented in Section 4.4. Section 4.5 is concerned with FPGA implementation and discussion of the experimental results. Section 4.6 concludes the Chapter.

4.2 Proposed Character Segmentation Algorithm

The proposed CS algorithm is mainly based on pixel projection and morphological operations. Compare to existing works based on pixel projection method [50, 59, 71-73], two optional morphological operations have been introduced in the proposed improved algorithm to minimise the impact of noise and the entire horizontal pixel projection step has been replaced by an NP height optimisation step. These modifications improve the robustness of the vertical projection and also accelerate processing speed.

The proposed method has three stages:

1. Pre-projection stage;
2. Vertical projection;
3. Horizontal projection.

Figure 4-1 shows the block diagram of the proposed CS system.

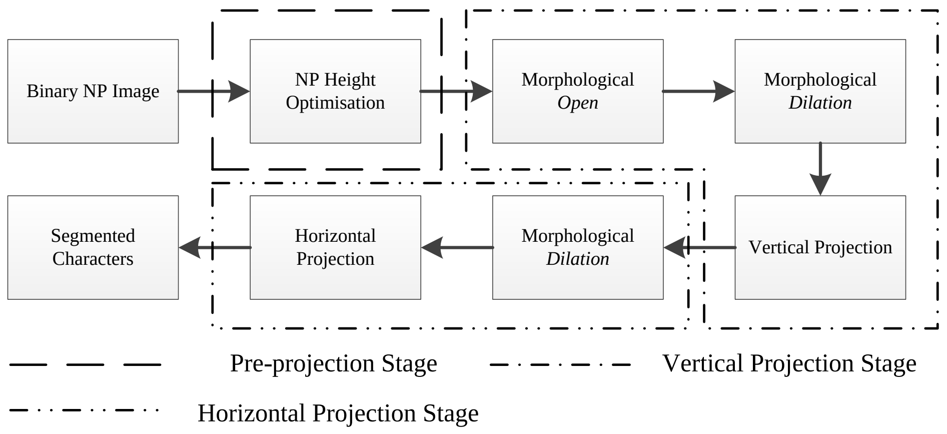


Figure 4-1: Block diagram of the proposed character segmentation system

The input binary NP images are the outputs of the NPL stage. All images from the NPL stage are binarised and inclined images must be roughly rotated before they are fed to the character segmentation stage.

4.2.1 Optimising NP Height

The traditional pixel projection algorithm normally performs an entire horizontal pixel projection followed by an entire vertical pixel projection where the resulting image from horizontal projection stage is used. According to the practical experimental results, although the horizontal pixel projection is a necessary step, the horizontal position of characters cannot be extracted when the NP is inclined or contains unnecessary information (e.g. national label and NP margin). For this reason and also to accelerate the processing speed, the proposed method replaces the entire horizontal pixel projection with the NP height optimisation step, which:

- Removes unnecessary parts of the NP; and
- Accelerates the processing speed by reducing the size of the image.

Firstly, this step analyses height and width parameters of the NP and decides whether NP height optimisation is required or not. If an optimisation step is required, two morphological operations are applied, otherwise only one morphological operation is applied. Figure 4-2 demonstrates the flow chart of the pre-projection stage.

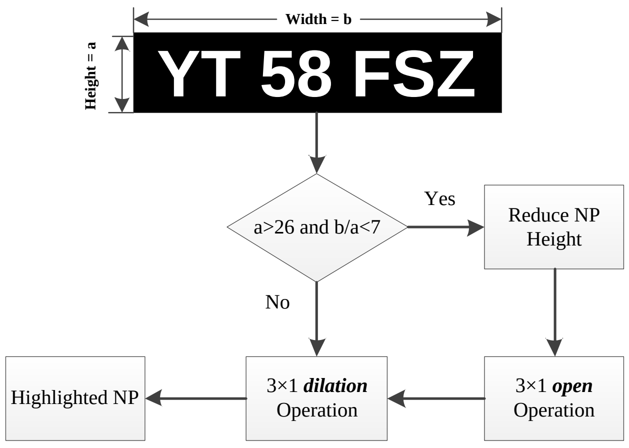


Figure 4-2: Flowchart of the pre-projection stage

In Figure 4-2, the known height and width of the NP are denoted as *a* and *b* respectively and can be used in branch conditions to determine whether the NP height must be reduced and the morphological *open* operation applied before applying the *dilation* operation. By analysing the images in the used database, the value of a varies from 18 to 60, the majority of NP heights are greater than 26 pixels, and normally have sufficient pixels in the character region. Therefore, the branch conditions are set as “ $a > 26$ and $b/a < 7$ ”. Further processing is performed on NPs that meet the set condition where two operations applied (i.e. NP height reduction and morphological open operation). For the first operation, the current NP height will be cropped by $0.15 \times a$ from the top and bottom of the NP which leaves a new NP height of $0.7 \times a$. This cropping process result is shown in Figure 4-3.

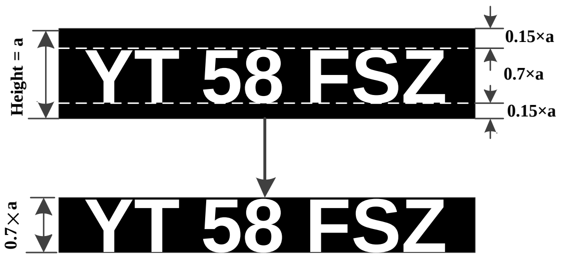


Figure 4-3: The NP height reduction process

The cropping factor 0.15 is the result of analysis of UK NP standard [116] and the NPs in our database. The cropped NP images can be categorised, as shown in Figure 4-4, as:

- Category 1: cropped NP images with full characters;
- Category 2: cropped NP images with full characters and additional noise; and
- Category 3: cropped NP images with cut characters.

The information lost in the third category does not affect the result of the vertical projection.

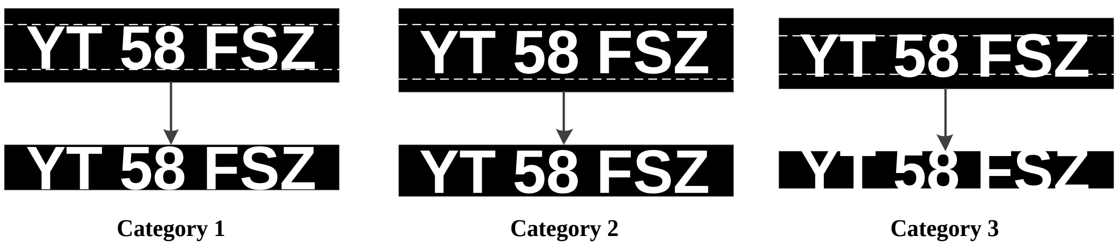


Figure 4-4: The three cropping categories

4.2.2 Vertical and Horizontal Projections

Once an NP image has been cropped, the morphological operations are performed. The shape of the morphological operation is based on a suitable structuring shape SE [110]. A vertical line-shape SE, shown in Figure 4-5 (a), with a size of 3×1 is used to perform the *open* operation in the proposed algorithm.

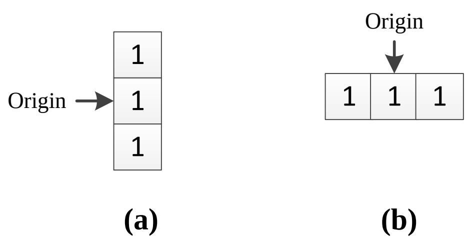


Figure 4-5: The 3×1 and 1×3 SEs. (a) 3×1, (b) 1×3

There are two basic morphological operations: *erosion* and *dilation*. An *erosion* operation (⊖) calculates the minimum pixel value in the SE, and assigns it to the *origin*; by contrast, a *dilation* operation (⊕) calculates the maximum pixel value in the SE. Let *I* denotes an image, morphological *erosion* and *dilation* operations transform *I* to a new image using an SE *T* with *s* elements, which are defined by:

$$I\,!\;T = \min_{s\in T} I_s \tag{4.1}$$

$$I \oplus T = \max_{s\in T} I_s \tag{4.2}$$

The morphological *open* operation is an erosion followed by a dilation. The *open* operation is used to remove unwanted margins and connections between characters before applying vertical projection. A 3×1 morphological *dilation* operation using the same SE is then used to enhance the vertical shape of each character.

If $p_{x,y}$ denotes a value of pixel at coordinates (*x*, *y*) in a NP image ($a\times b$), the vertical projection value at y^{th} column of NP image can be calculated by:

$$v_y = \sum_{x=0}^{a-1} p_{x,y} \tag{4.3}$$

where $x = 0,1,2,...,a-1$ and $y = 0,1,2,...,b-1$.

Figure 4-6 (a) shows vertical projection histogram before cropping the original NP image. The minimal value in the histogram is approximately ‘5’, and this value is not consistent for all NPs in the database, therefore it is not easy to find a constant threshold at which characters and non-characters can be separated. In Figure 4-6 (b), although the minimal value of the histogram map is ‘0’, gaps between neighbour characters are not clearly seen. Figure 4-6 (c) shows the pixel vertical histogram after applying morphological operations.

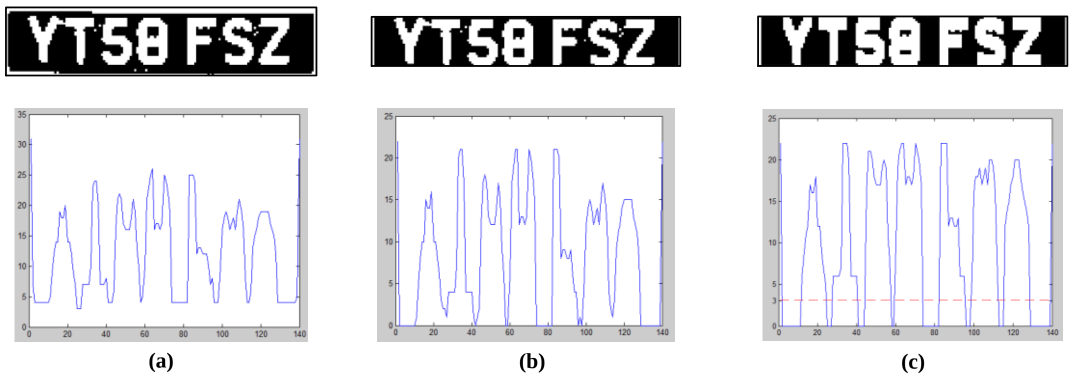


Figure 4-6: NP images and their vertical projection histograms. (a) Original NP, (b) After performing horizontal cropping, (c) After performing morphological operations

In order to find the critical points between two characters, the proposed algorithm uses an approach that consists of the following two steps:

1) Find a group of points *A* from vertical projection array $V = \{v_0, v_1, v_2, ..., v_{b-1}\}$.
2) Find a group of critical points *B* from group *A*.

The points in group *A* should meet the conditions $v_y \le t_1$ and $|(v_{y+1} - v_y)| \ge t_2$ where thresholds t_1 and t_2 have been found by experiment.

Let Cd, Cv, Ca denote the difference between two neighbour points in group *A*, the vertical pixel projection value at position $A_i + 2$ and the average vertical pixel projection value from A_i to A_{i+1} respectively, and can be calculated by:

$$Cd_i = A_{i+1} - A_i \tag{4.4}$$

$$Cv_i = v_{A_i+2} \tag{4.5}$$

$$Ca_i = \frac{\sum_{y=A_i}^{A_{i+1}} v_y}{Cd_i} \tag{4.6}$$

where i is an integer $\leq b-1$.

Let w_{min} and w_{max} denote the minimum and maximum expected character widths respectively. v_{min} denotes the minimum expected vertical histogram value at position $A_i + 2$. $Vavg_{min}$ and $Vavg_{max}$ denote the minimum and maximum expected average vertical histogram values for each character respectively. The points in group B should meet the conditions $w_{min} \leq Cd_i \leq w_{max}$, $Cv_i \geq v_{min}$ and $Vavg_{min} \leq Ca_i \leq Vavg_{max}$.

Once all the critical points of characters have been found, all characters in the original NP are segmented using these critical points. Figure 4-7 shows an example of vertical cropping and a character horizontal projection.

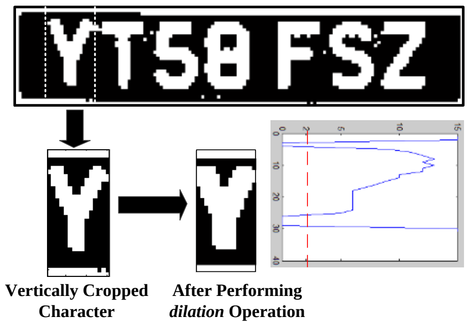

YT58 FSZ
Vertically Cropped Character
After Performing *dilation* Operation
0
5
10
15
10
20
30
40

Figure 4-7: Character horizontal projection

In Figure 4-7, the character ‘Y’ has been cropped from the original NP image, followed by a horizontal projection operation. In order to enhance the horizontal projection histogram, the horizontal line-shape SE, shown in Figure 4-5 (b), with a size of 1×3 is used to perform a *dilation* operation. The same critical point localisation approach used for vertical projection is also used for localising horizontal critical points. Once these critical points are localised, the character 'Y' is fully segmented. The rest of characters on the NP are segmented using same method. Figure 4-8 shows the fully segmented NP.

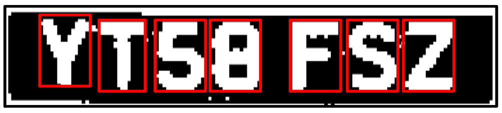


Figure 4-8: A fully segmented NP

Overall process of the proposed CS algorithm can then be summarised in the following pseudocode.

Proposed algorithm: CS algorithm

```
1.  Input images: localised NP image ($a\times b$), where $a$ is height of NP, b is width of NP
2.  Output images: Segmented character image
3.  if (a>26 and b/a<7) then
4.          reducing height of NP $a$ to $0.7a$
5.          morphological open input image using 3×1 SE
6.          morphological dilation after the open
7.  else
8.            morphological dilation for the input image
9.  end
10.         obtain highlighted NP image
11. for all pixels in the highlighted NP image do
12.          generating vertical projection histogram
13.         finding the vertical critical points between two characters
14.         generating horizontal projection histogram for each vertical cropped character image
15.         finding the horizontal critical points for each character
16. end
```

Compare to the existing algorithms, two optional morphological operations have been introduced in the proposed CS algorithm to eliminate noise impact and the entire horizontal pixel projection step has been replaced by an NP height optimisation step. These modifications improve the robustness of the CS algorithm and also accelerate processing speed.

4.3 Proposed Character Segmentation Architecture

The proposed CS architecture consists of vertical and horizontal projection modules and each module consists of the three main blocks listed below:

1. ***Pre-Projection Block***: vertical and horizontal pre-projection blocks are used in vertical and horizontal projection modules respectively.
2. ***Morphological operator***: *open* and *dilation* morphological operations are used in the vertical projection module while only *dilation* is used in the horizontal projection module.
3. ***Critical point localiser***: Almost the same block is used for both vertical and horizontal projection modules. The only difference is the set conditions.

The structure of the proposed system is shown below in Figure 4-9.

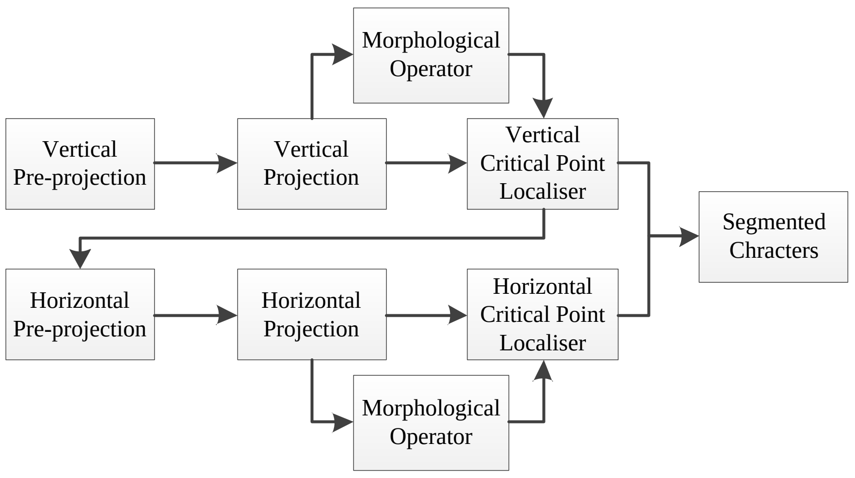


Figure 4-9: Vertical and horizontal projection based system

4.3.1 Vertical Projection Module

The first building block in the vertical projection module is the vertical pre-projection block which consists of the Vertical Memory Reader (VMR) and NP height Optimiser. This operation is followed by a morphological operation and a horizontal critical point localisation. The overall block diagram of the vertical projection module is shown in Figure 4-10.

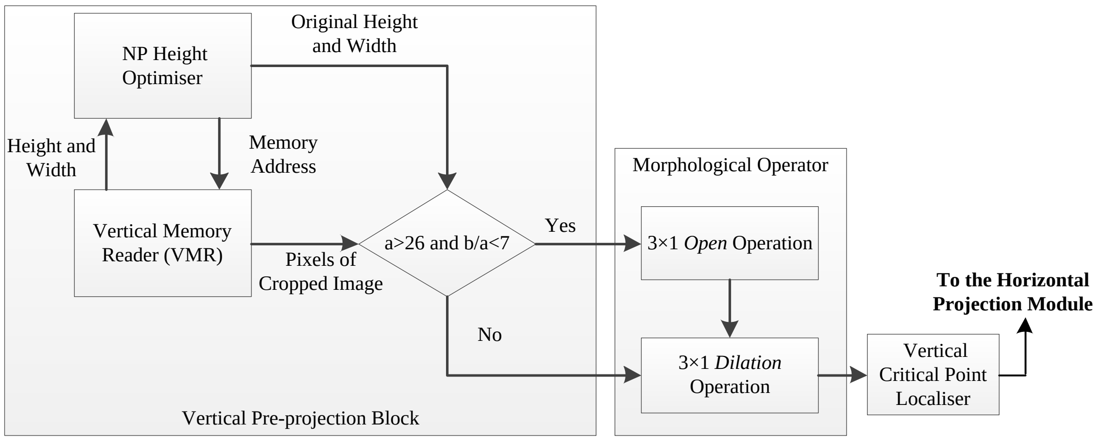


Figure 4-10: The overall vertical projection block diagram

Vertical Pre-projection Block

$a \times b$ pixels of the binary NP image are scanned row by row, from top to bottom and from left to right and stored in memory. In Figure 4-10, the processing starts with the memory reader module, where the known NP height a and width b are passed to the NP height optimiser to calculate the memory start and end addresses where the first and last pixels of the horizontally cropped image are stored in memory. Figure 4-11 illustrates a horizontally cropped NP with the start and end pixels.

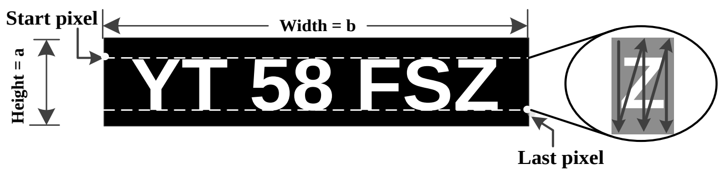


Figure 4-11: Cropped NP image with first and last pixels

Although the theoretical fixed horizontal cropping factor 0.15 can be used to successfully crop the majority of the NPs, some of them cannot be properly cropped. This is due to the fact that NPs with heights below 40 are more likely to have insufficient pixels on characters’ regions than the NPs with greater heights (e.g. where margins of some NPs have already been cropped during the NPL stage or already have small margins in the NP). Therefore, NP height can be used to determine which cropping value should be used. In the proposed algorithm, four range-specific cropping values in four different NP height ranges are used: $26 \le a < 30$, $30 \le a < 40$, $40 \le a < 50$ and $50 \le a \le 60$.

As the main aim of this research project is to implement the entire ANPR system on one single FPGA, this solution avoids the need of multiplications to calculate individual cropping factors which will significantly reduce the hardware resources usage. Therefore, integrated on-chip multipliers are saved for other ANPR stages (i.e. NPL [117] and OCR). Table 4-1 shows the four ranges and corresponding start and end address calculation.

Table 4-1: Calculation of memory start and end address within each NP height range

	$26 \le a < 30$	$30 \le a < 40$	$40 \le a < 50$	$50 \le a \le 60$
Start Address	$3 \times b$	$4 \times b$	$6 \times b$	$8 \times b$
End Address	$a \times b - 2 \times 3 \times b$	$a \times b - 2 \times 4 \times b$	$a \times b - 2 \times 6 \times b$	$a \times b - 2 \times 8 \times b$

In the next step, the VMR reads the stored pixels from memory starting from the calculated start address and scanning the stored image column by column, from top to bottom and from left to right. Before the pixels of the cropped image read by VMR are sent to the morphological operator, the NP image is tested against the geometrical condition $a > 26 \,\&\, b/a < 7$. If the condition is met, all pixels read by the VMR for that NP will pass through *open* and *dilation* morphological operators in the morphological operator block, otherwise, they will only pass through the *dilation* operator.

Morphological Operator

Open and *dilation* morphological operations are performed within this block. As the input data are binary, the *max* and *min* can be simplified to a logical *OR* and a logical *AND* operation as shown in Equations 4.7 and 4.8 respectively.

$$I \,!\, T = \mathop{\wedge}_{s \in T} I_s \tag{4.7}$$

$$I \oplus T = \bigvee_{s \in T} I_s \tag{4.8}$$

Figure 4-12 shows in block level diagram how the two operations are applied to the pixels coming from the vertical pre-projection block.

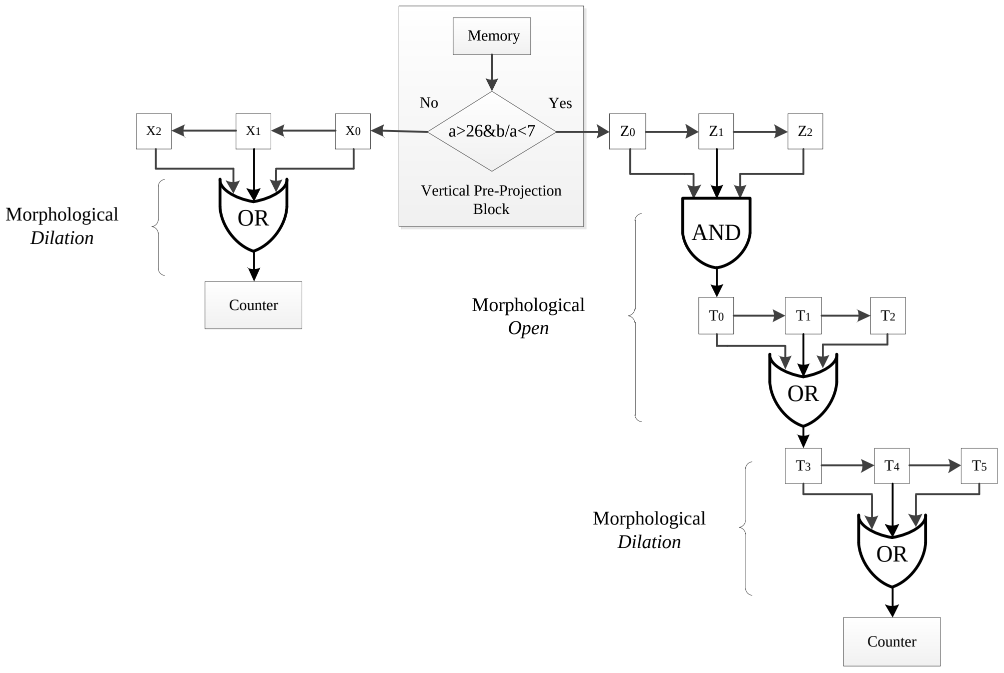


Figure 4-12: Block level diagram of the morphological operation process

In Figure 4-12, "Z_0, Z_1, Z_2", "X_0, X_1, X_2" and "$T_0, T_1, ..., T_5$" are one-bit buffers which are used for buffering pixels read from memory. Data stored in these buffers are propagated from one buffer to the next every one clock cycle. If the set condition is met, the value of the current input pixel is passed into "Z_0" then after two clock cycles a logical *AND* operation is performed on the data stored in "Z_0, Z_1, Z_2" and the result is stored in buffer "T_0". The next stage is similar but instead of using a logical *AND* operation, a logical *OR* operation is performed on the data stored in "T_0, T_1, T_2"which ends the *open* operation. The result is stored in "T_3". The *open* operation is then followed by a *dilation* operation where a logical *OR* operation is calculated using the data stored in "T_3, T_4, T_5". If the final result value is '1', the counter which was initialised to zero will be incremented by one. When one column of pixels from the NP is processed, the result of the counter will be stored and the counter re-initialised to zero.

If the set condition is not met, only morphological *dilation* will be performed, which is exactly same operation used after the *open* operation when the condition is met.

Vertical Critical Point Localiser

A Vertical Critical point localiser is used to localise critical points from the vertical projection data stream generated from the previous block (i.e. morphological operator). Figure 4-13 shows in block level diagram the process of localising critical points of vertical projection.

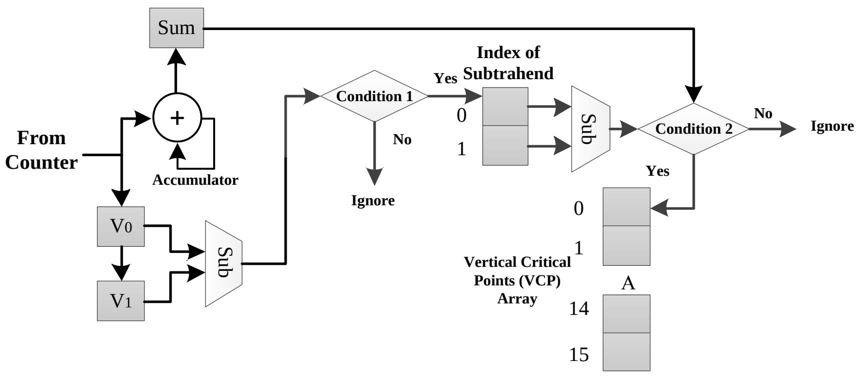


Figure 4-13: Block level diagram of the process of localising the critical points of the vertical projection

Data are continuously stored in the buffers V_0 and V_1 from the morphological operation module. Once two values are available in those two buffers, the first value is subtracted from the second one and the absolute value of the result is passed through the condition 'condition 1' set to generate group *A*. Once there is a value in *A*, an accumulator starts to accumulate the vertical projection value until the second value of *A* is found, then the current result stored in register *Sum* is used in 'condition 2'. A new accumulation process starts when the previous accumulated value is used in 'condition 2'. The index of the subtrahend is stored temporarily in an array of two elements. Once two values are available in the two-element array, the same subtraction operation is applied and the result passes through the condition ‘condition 2’ set to generate group *B*. The successful are stored in a vertical critical points array of 16 elements which are used by the horizontal projection module.

4.3.2 Horizontal Projection Module

The first building block in the horizontal projection module is the horizontal pre-projection block. Unlike the vertical pre-projection block, the horizontal pre-projection block consists only of a Horizontal Memory Reader (HMR) which uses the 16-element array output from the vertical projection module to read characters’ pixels from memory. This operation is followed by a morphological operation and a horizontal critical point localisation. The overall block diagram of the horizontal projection module is shown in Figure 4-14.

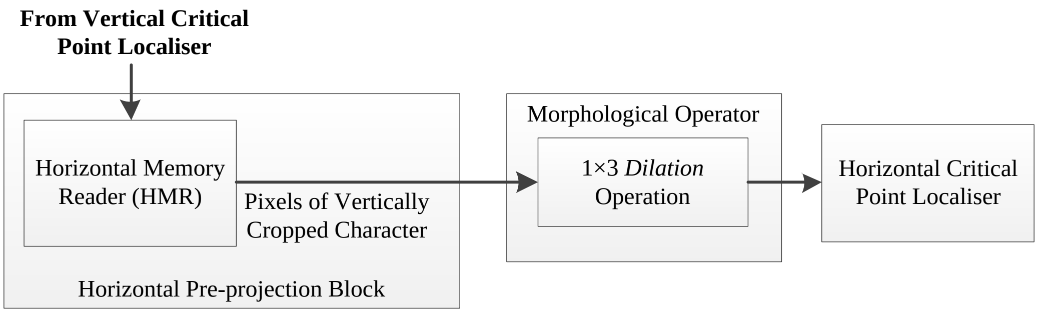

From Vertical Critical Point Localiser
Horizontal Memory Reader (HMR)
Pixels of Vertically Cropped Character
Horizontal Pre-projection Block
Morphological Operator
1×3 Dilation Operation
Horizontal Critical Point Localiser

Figure 4-14: Overall block diagram of the horizontal projection module

Horizontal Pre-projection Block

In the horizontal pre-projection block, the HMR uses the column numbers stored in the Vertical Critical Points (VCP) array to read the characters’ pixels from memory. Two elements are used for each character and the pixels are scanned row by row, from top to bottom and from left to right. For example, pixels of the first row in the first character are stored in memory locations VCP_0 to VCP_1 which are the first and second elements of the VCP array. The second row starts from $VCP_0 + b$, where *b* is the width of NP. Figure 4-15 illustrates the process with a vertically cropped NP. The read pixels will be passed through *a dilation* morphological operator in the morphological operator block.

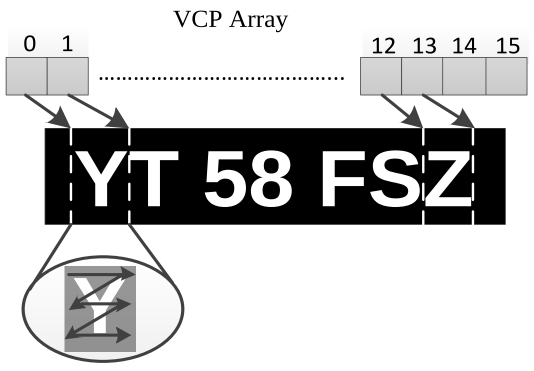


Figure 4-15: The vertically cropped NP process

The vertical critical points array has 16 elements in total, however a UK NP has a maximum of seven characters, therefore, no more than 14 elements will be used. The two extra elements may be used in some cases as temporary storage in case an error segmented region occurs.

Morphological Operator and Horizontal Critical Point Localiser

The morphological Operator block consists only of a 1×3 *dilation* operator which is similar to the 3×1 *dilation* operator used in the vertical projection module (see Figure 4-12). The only difference being the direction in which the data is read (i.e. VMR vs HMR).

The horizontal critical point localiser is also similar to the vertical critical point localiser, the only difference being the value of threshold set.

Once the all vertical and horizontal critical points have been found, they can be used to extract all of the pixels associated with a particular character.

4.4 MATLAB Implementation and Results

The proposed CS algorithm was tested in a MATLAB environment using a database of 1000 binary UK NPs with varying resolutions. The MATLAB implementation was used as a proof of concept prior to the hardware implementation. The database predominantly consists of three different sample sets: normal, inclined and noisy NPs, which are taken from our previously implemented NPL system [117].

1) Sample set 1: contains normal NPs where characters are clear and obvious.

2) Sample set 2: contains inclined NPs where characters are on different horizontal levels (the horizontal inclination angles $|\alpha|<4°$).

3) Sample set 3: contains noisy NPs where noise information or connected characters are included in NPs (e.g. screws, national labels, and unnecessary boundaries).

During the experimental testing, the successful cases for each sample that was counted manually in multiple times. Table 4-2 shows a sample image from each sample set and the successful segmentation rate for each set where all characters in a NP are correctly isolated from each other.

Table 4-2: Successful character segmentation rates by sample set

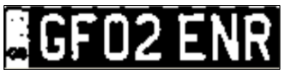
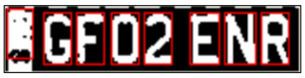
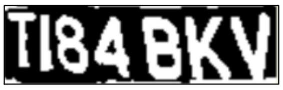
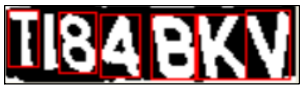
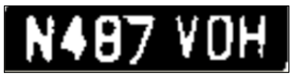
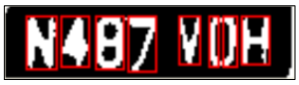
	Sample Set 1	Sample Set 2	Sample Set 3	Overall
UK NPs				
Successful Character Segmentation Rate	205/212 (96.7%)	337/351 (96.0%)	420/437 (96.1%)	962/1000 (96.2%)

The proposed algorithm has an overall 96.2% successful character segmentation rate when tested using UK NP images. Sample set 1 has relatively higher character segmentation rate than sample sets 2 and 3, which is due to the fact that the scenes in the latter sample sets contain more complex background environments or inclined NPs. Since morphological operations are used in the proposed algorithm to remove the noise, the impact of noise has been significantly reduced. In addition, as the proposed algorithm only analyse the width/height of the character, the gap between the adjacent characters does not affect the segmentation rate. Using MATLAB and a Dual Core 2.4GHz, 3G RAM PC, the average processing time was found to be 22.3 *ms* per image.

Segmentation failures fell into one of three categories: (1) the number of segmented characters is more than the actual number of characters on NP. This is caused by the non-character parts on the NP being very similar in shape to a character. (2) One or more characters are missed. This is caused by inclined characters or a degraded original image. (3) The character is split into two separated characters. This is due to insufficient pixels in the character.

As the image size range in the database is relatively large (i.e. 18×190 - 60×300), the proposed algorithm uses five different conditions to localise vertical critical points. Although five conditions are used, there are still a few NPs that were not segmented properly. Table 4-3 shows some examples. The NP from category 1 contains a non-character component that was segmented as a character. Character ‘1’ in the NP from category 2 was missed. Character ‘0’ in the NP from category 3 was also missed because of the small number of pixels in the characters which was caused by the poor original image quality.

Table 4-3: Samples of failed images.

Category	Original NP	Segmented NP
(1)		
(2)		
(3)		

4.5 FPGA Implementation and Results

The proposed architecture for CS has been simulated using the PAL Virtual Platform (PALSim) [112]. After simulation, the architecture has been successfully implemented and verified using the Mentor Graphics RC240 FPGA development board. Handel-C has been used for hardware description of the proposed architecture. For details of the experimental tools can be found in APPENDIX B.

The two main building blocks of the proposed architecture are the vertical and horizontal projection blocks shown in Figures 4-10 and 4-14 respectively. Pipelining has been used in their implementation and an NP image can be processed in $(x \times C_1 + C_2)$ clock cycles where:

- x is the number of characters in the NP
- C_1 is the number of clock cycles to complete one vertical projection and depends on the size of the NP image
- C_2 is the number of clock cycles to complete one horizontal projection depends on the size of the NP image

Sample codes for the CS implementation are discussed in APPENDIX C.

4.5.1 Proposed environment for character segmentation on FPGA

Figure 4-16 shows a general view of the entire CS system. It consists of a host application (GUI), a UK NP database and the RC240 FPGA development board. The host application was implemented using Visual Studio 2008 and gives the user the ability to select an NP image from the database, display it and send it to the FPGA for processing. Once processed, the output from the FPGA is displayed on the same GUI. More details for implementing FPGA host application are discussed in APPANDIX A.

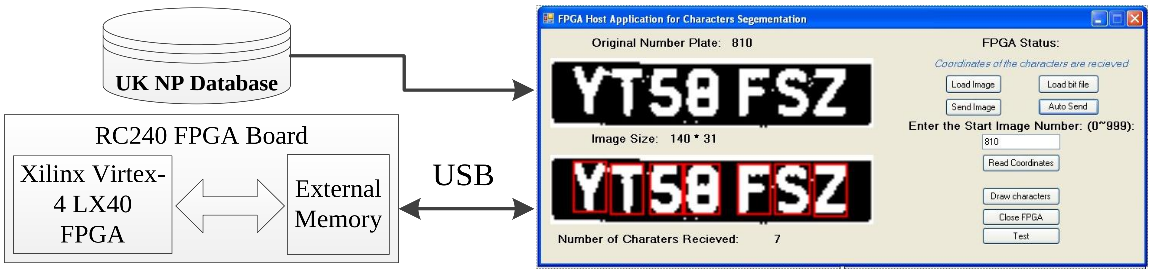

UK NP Database
RC240 FPGA Board
Xilinx Virtex-4 LX40 FPGA
External Memory
USB
FPGA Host Application for Characters Segementation
Original Number Plate: 810
YT58 FSZ
Image Size: 140 * 31
YT58 FSZ
Number of Charaters Recieved: 7
FPGA Status:
Coordinates of the characters are recieved
Load Image
Load bit file
Send Image
Auto Send
Enter the Start Image Number: (0~999):
810
Read Coordinates
Draw characters
Close FPGA
Test

Figure 4-16: Host application for character segmentation

The UK binary NP database used for the MATLAB implementation, has also been used for testing and validating the FPGA implementation. Table 4-4 shows the FPGA implementation results when using the three sample sets.

Table 4-4: Successful character segmentation rate by sample set for FPGA implementation results

	Sample Set 1	Sample Set 2	Sample Set 3	Overall
Successful CS Rate	208/212 (98.1%)	343/351 (97.7%)	426/437 (97.5%)	977/1000 (97.7%)

The CS rate is higher than that achieved in the software implementation where the overall rate is 97.7%. The improvement is due to the use of the four height range-specific cropping values shown in Table I in place of the single horizontal cropping factor 0.15, to find the most suitable memory reading address in the VMR block.

4.5.2 Hardware Usage, Running Frequency and Power Consumption

Due to the low complexity of the proposed algorithm, the proposed architecture requires only 11% of the on-chip FPGA resources. Table 4-5 summarises the required on-chip resources.

Table 4-5: Usage of FPGA on-chip Resources

On-chip Recourses	Used	Available	Utilisation
Occupied Slices	2,100	18,432	11%
LUTs	2,964	36,864	8%
Flip-Flops	1,449	36,864	3%
BRAMs	2	96	2%
DSP48s	1	64	1%

11% of the on-chip FPGA slices are used to implement the proposed CS architecture. In these slices, 8% LUTs, 3% flip-flops and 2% BRAMs are used to implement logic operations, registers and RAMs respectively. A DSP48s slice is used to perform the arithmetic calculation in the Critical Point Localiser block. According to the previous chapter, NPL implementation requires 33% of the on-chip resources, therefore, the total hardware usage of NPL and CS is 44%, which leaves 56% of the FPGA area to be used for the remaining part of an ANPR system (i.e. OCR).

The maximum running frequency is 74.5 MHz and the number of clock cycles needed for one image to be processed is between 13000-103000 (including the clock cycles for image reading, vertical projection and horizontal projection), which depends on the resolution of the input NP and number of characters in it. The execution time for processing one frame can be calculated using the Equation 3.4.

Based on Equation 3.4, the proposed architecture can process one image (18×99 - 60×300) and produce a result in $0.2 \sim 1.4ms$. The difference in the execution time is due to the size of the images which affect the number of clock cycles. The smaller the size of the image, the lower the number of clock cycles is required. The execution times achieved mean that the proposed architecture satisfies the requirement for real-time processing.

The power consumption of the designed circuit has also been analysed using Xilinx XPower Analyser [115], and the results obtained are shown in Table 4-6.

Table 4-6: Estimation of Power Consumption

Resource Type	Value of Power (mW)
Clocks	120
Logic	2
Signals	2
BRAMs	2
IOs	40
Clock Managers	211
Leakage	344
Total Power	721

In Table 4-6, the clocks power, logic power, signal power, BRAMs power and IOs power belong to the dynamic power, they are directly affected by the user design resource usage. The quiescent power is consumed due to transistor leakage, which is depending on the chosen hardware. The total dynamic power consumption for the proposed CS implementation is 269 mW out of the total power consumption 721 mW.

The proposed CS algorithm reduces processing time by using an optimising NP height module to reduce any horizontal noise effect and so avoid the need to use the entire horizontal projection, and so the vertical projection can then be applied without affecting the segmentation accuracy. The morphological operations are used to enhance pixel projections, which can significantly improve the segmentation rate when NPs contain noise.

Table 4-7 lists the results of character segmentation for recent ANPR systems that use the pixel projection approach and they are either software or hardware based systems.

Table 4-7: Performance Comparison

CS Technique	CS Rate (%)	Platform/Processor	Speed (Sec)
[59]	95.6%	Pentium 1.6 GHz PC	2
[50]	98.8%	Pentium 2.8GHz	0.2
[11]	N/A	Texas Instruments C64	0.0018
[105]	87.16%	Virtex-4 FPGA	N/A
Proposed system on PC	96.2%	Dual Core 2.4GHz	0.023
Proposed system on FPGA	97.7%	Virtex-4 FPGA	0.0002~0.0014

Generally, the main advantage of hardware based systems is the fast processing speed, which is of particular interest in real-time environments. By comparing the results of the PC and FPGA-based implementations, it can be clearly seen that the latter outperforms the former by a factor of 8. It also outperforms the existing solutions in terms of speed and/or accuracy. Although different databases are used in the works [50], [59] and [105], similar image cases are contained in our database.

4.6 Conclusion

In this Chapter, an improved CS algorithm has been proposed for FPGA implementation, which is based on a combination of histogram projection and morphological operations. Furthermore, an efficient architecture based on the proposed algorithm has been successfully implemented and tested using the Mentor Graphics RC240 FPGA development board. It requires only 11% of the available on-chip resources of a Virtex-4 FPGA, runs with a maximum frequency of 74.5 MHz and is capable of processing one image in $0.2 \sim 1.4ms$ with a successful segmentation rate of 97.7% when using a database of 1000 NP images.

Chapter 5: Number Plate Character Recognition Algorithm and its Efficient FPGA Implementation

5.1 Introduction

In the previous chapter, a low complexity pixel projection and morphological operations based CS algorithm and its efficient architecture implementation have been discussed. The next main stage of an ANPR system is OCR stage, where the segmented characters are recognised and converted into encoded texts. As concluded from the Chapter 2, incorrectly segmented characters from the CS stage, where characters are not in the expected position or parts of them are missed, may affect the OCR operation. NNs and statistical classifiers, which give better results compare to common pattern matching technique, can overcome this problem due to their strong memorability and self-adapting ability. However, in order to achieve good performance, large amount of samples are needed to train the NNs. In addition to the advantages of using NNs mentioned above, the parallelism and modularity of NN can be perfectly mapped onto FPGA using parallelism and pipeline techniques. The reconfigurable ability of FPGAs also provides a rapid way to adapt the weights and topologies of NNs [118].

In this Chapter, a low complexity and robust OCR algorithm based on feed-forward NN is presented where two non-overlapping real NP character image data sets are used for training and testing the proposed NN. An area/speed efficient architecture based on the proposed algorithm is also presented, which has been successfully implemented on a Virtex-4 FPGA. Because the proposed architecture is an off-line NN, there is no need to train NN on FPGA. Large amounts of trained weights are stored in external RAMs, which can be easily updated without changing the FPGA configuration. The proposed architecture for implementing the two layers feed-forward NN on FPGA for real-time OCR application is designed to process a large number of neurons in a pipelined manner to achieve high running frequency and throughput rate. The use of multipliers has been avoided in the first layer from the proposed architecture, which significantly reduces on-chip resources usage and power consumption.

A MATLAB implementation of the proposed algorithm was used as a proof of concept prior to the hardware implementation. An efficient architecture based on the proposed algorithm is also presented. It has been implemented and verified using the Mentor Graphics RC240 FPGA development board. The used UK character images were segmented from NP images that were collected by author and provided by CitySync Ltd. [109] who are one of the leading UK providers of ANPR solutions. The images are from outdoor real-world environments, which cover a wide range of conditions in terms of various weather, lighting and contrast. The results achieved indicate that the FPGA can provide 12-time speedup over the MATLAB implementation with the same recognition rate.

The rest of this Chapter is organised as follows: Section 5.2 describes the proposed OCR algorithm. The MATLAB implementation and analysis of the experimental results are presented in Section 5.3. Section 5.4 is concerned with the description of the proposed OCR architecture. Its FPGA implementation and discussion of the experimental results are presented in Section 5.5. Section 5.6 concludes the Chapter.

5.2 Proposed OCR Algorithm

The proposed OCR algorithm uses a multi-layer feed-forward NN to translate scanned character images into machine encoded text. Typically, an N-layer NN consists of a set of input vectors, N-1 hidden layers, one output layer and a set of output vectors. Each layer consists of a set of neurons and corresponding transfer function (e.g. sigmoid, linear) [119]. Figure 5-1 shows a two-layer feed-forward network with one hidden layer and one output layer.

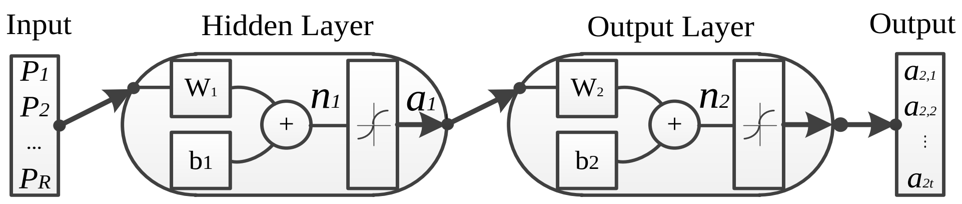


Figure 5-1: The architecture of two-layer feed-forward network

A hidden layer consists of *S* neurons and each neuron has *R* weights, which can be presented in a $S{\times}R$ matrix called Input Weight matrix **I** as shown in equation 1. The input vector **p** has *R* elements $[p_1, p_2, \ldots, p_R]^T$, which are multiplied by **I** and the resulting matrix is summed with a bias vector $\mathbf{b_1}$ to form vector n_1 as shown in equation 2. The output of the hidden layer $\mathbf{a_1}$ is the result of applying the transfer function on $\mathbf{n_1}$ (see Equation 5.3).

$$\mathbf{I} = \begin{bmatrix} w_{1,1}\ w_{1,2}\ \ldots\ w_{1,R} \\ w_{2,1}\ w_{2,2}\ \ldots\ w_{2,R} \\ \ldots\ldots \\ w_{S,1}\ w_{S,2}\ \ldots\ w_{S,R} \end{bmatrix} \tag{5.1}$$

$$\mathbf{n_1} = \mathbf{I} \cdot \mathbf{p} + \mathbf{b_1} \tag{5.2}$$

$$\mathbf{a_1} = f_1(\mathbf{n_1}) \tag{5.3}$$

The same operations applied in the hidden layer are used in the output layer, which consists of *K* neurons, where $\mathbf{a_1}$ is used as the input vector (see Equations 5.4, 5.5 and 5.6).

$$\mathbf{L} = \begin{bmatrix} w_{1,1}\ w_{1,2}\ \ldots\ w_{1,S} \\ w_{2,1}\ w_{2,2}\ \ldots\ w_{2,S} \\ \ldots\ldots \\ w_{K,1}\ w_{K,2}\ \ldots\ w_{K,S} \end{bmatrix} \tag{5.4}$$

$$\mathbf{n_2} = \mathbf{L} \cdot \mathbf{a_1} + \mathbf{b_2} \tag{5.5}$$

$$\mathbf{a_2} = f_2(\mathbf{n_2}) \tag{5.6}$$

In UK NP system, there are 25 letters and 9 numbers. ‘I’ is a non-used character and ‘O’ and ‘0’ are considered the same. In the new regulations introduced in September 2001, the format of a UK NP consists of two letters, two numbers, a space and three further letters [116]. This can be used to identify the letter ‘O’ from the number ‘0’. In an ANPR system, the segmented characters are presented to the OCR recogniser with different sizes. Therefore, all the images of the NP characters must be resized to the same size before using them in NN training or testing. Because the width of character ‘1’ is 1/3 of the size of other characters, the identification of this character is very easy during the character segmentation stage prior to sending it to the NN for recognition. Figure 5-2 shows the character set used in UK NPs:

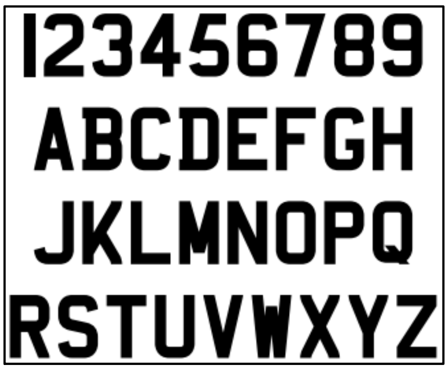


Figure 5-2: The UK NP character set [116]

The most commonly used images in OCR are binary images. They are also used also in other stages of ANPR system. Binary images require less computational intensity compare to other types of images which significantly decreases the computation for real-time applications such as ANPR. In the proposed work, a 2-D binary image matrix $w \times l$ is transformed into 1-D vector **p** with R elements $[p_1, p_2,\ldots, p_R]^T$ to be used to form the inputs of NN, where pixels of binary image are read row by row to form the **p**. Due to only having 33 possible characters, the output of NN has been decided as $\mathbf{a_2}=[a_{2,1}, a_{2,2},\ldots,a_{2,33}]^T$. In order to find the most suitable NN architecture for OCR task, the different number of neurons S and size of input vector **p** are used to create different NNs.

In order to choose the best suitable training algorithm for the proposed neural network, several training algorithms are used to train the neural network. For example, Scaled Conjugate Gradient (SCG) algorithm [120], Backpropagation (BP) algorithm and Levenberg-Marquardt algorithms [121]. Because the training speed of SCG is much faster than traditional back propagation algorithm (BP), and also it gives better results than with other training methods, the SCG is chosen as the training method of the NN. Before the start of training, the NN was initialised using Nguyen-Widrow initialisation algorithm [122], where the weights and biases in each layer are initialised and distributed approximately evenly over the input space. In order to use the proposed algorithm with NPs prior to 2001 or recognise different font characters, an extra character data set needs to be used to train the NN (e.g. Germany characters from German ANPR system).

5.3 MATLAB Implementation and Result Discussion

MATLAB has been used as a proof of concept of the proposed algorithm and it has also been used to generate the weights of the neural network. 6436 binary images with varying resolutions from the previous CS stage were used [123].

First of all, the binary images of the characters are resized to the same size. To select the right size, several sizes of input images have been used for NN training. The larger the size of the image the higher is the recognition rate but large sizes significantly increase the complexity of the structure of the NN as the number of weights will increase. The size corresponding to the best suitable result is used for the final NN. Based on experimental results, the size of the input binary image was decided to be 34×22.

The entire database has been divided into two non-overlapping groups: group 1 and group 2. The first group has 45% of characters, which is used only for neural network training. The second group has 55% of characters, which is used only for neural network testing. Figure 5-3 shows some examples from those two groups after resizing.

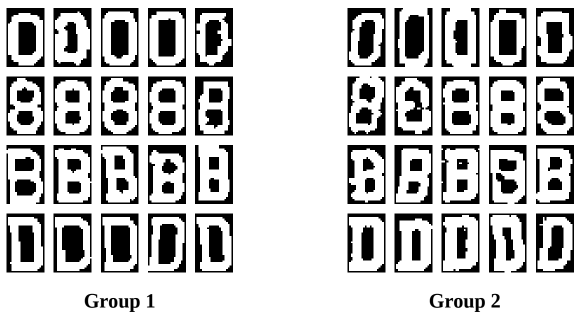


Figure 5-3: Sample characters from both training and testing groups

There are two separate sets of training data that were used to train two different neural networks respectively. The first set is the original 45% training data and the second one is the same set but with added random values between 0 and 1 from the standard normal distribution on the images. The SCG training algorithm is based on supervised learning algorithm. This means that the weights of neural network are updated based on each calculation of a pair consisting of an input image and a desired output from training data set. Therefore, adding noise to the training data set will not affect the training process. In the SCG training process, the 45% training data are divided into three sub sets, where 80%, 15% and 5% data are used for training, validation and testing respectively, the maximum iterations, performance goal, minimum performance gradient are set to 1000, 0 and 1×10^{-6} respectively and the maximum validation failure σ and λ are set to 6, 5×10^{-5} and 5×10^{-7} respectively. The achieved training, validation and testing successful rates are 99.85%, 99.87% and 99.94%. In order to obtain more accurate performance of the trained neural network, another non-overlapping testing data set is used for the testing.

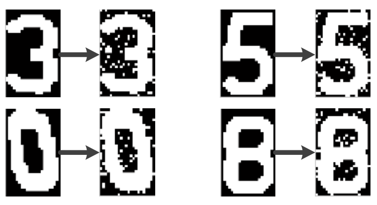


Figure 5-4: Examples of training data with random noise

The rest of the 55% of character images are used for testing trained NNs with different sizes (i.e. 30, 50, 70 and 100 neurons) and training sets (i.e. with and without noise added).

Figure 5-5 shows the character recognition rates when different training data sets are used. The NN is trained using the set with noise which will result in a better performance in terms of character recognition rate. In addition, several NNs with different sizes were trained and tested independently and the results show that the more neurons there are the better is the character recognition rate and higher is the number of weights which significantly increases the scale of the NN architecture and the on-chip resources usage on FPGA. The NN with 50 neurons was used for final NN implementation and it needs 221 iterations for the training.

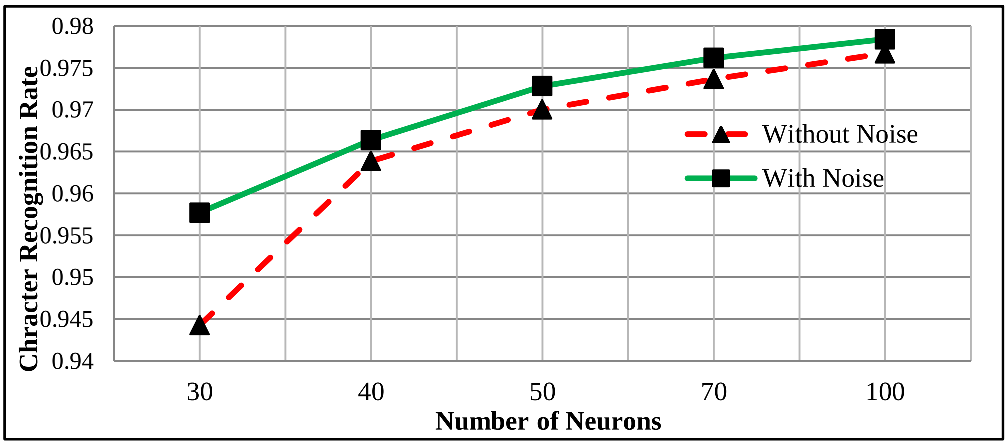


Figure 5-5: Character recognition rate with different numbers of neurons

Overall character recognition rate of the 55% testing data is around 97.3%, and the 55% testing data have been divided into different data sets where each set contains images of the same character in order to analyse the performance of NN on different the characters. Figure 5-6 shows the recognition rate of each character using the proposed OCR algorithm.

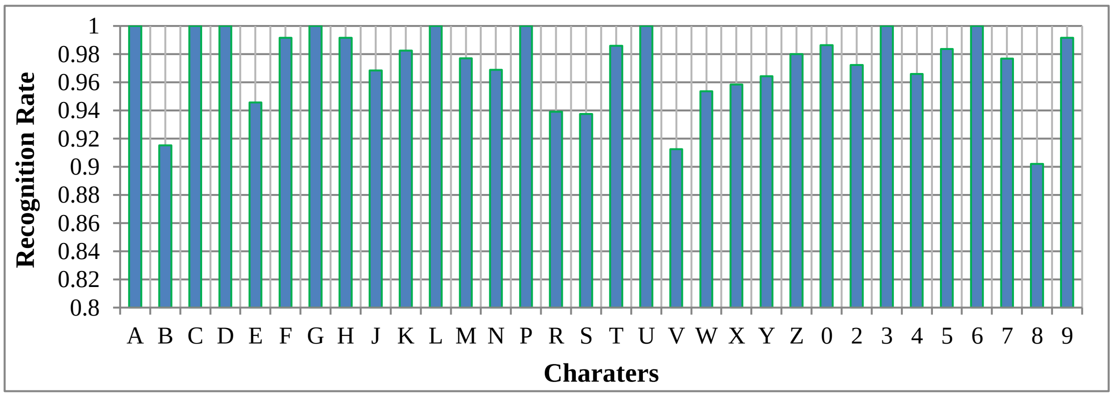


Figure 5-6: The recognition rate of each character

As shown in Figure 5-6, the proposed algorithm handles well simple character, e.g. ‘A’, ‘C’ and ‘6’, however, for the ambiguous characters, e.g. ‘B’, ‘V’ and ‘8’, the proposed OCR algorithm has lower recognition rate for two main reasons:

- Character’s similarity with other characters (e.g. ‘B’ and ‘8’); and
- Image quality (see Figure 5-7).

In order to recognise these characters, some particular features need to be extracted from them and not only pixels’ information (e.g. distinguishing parts of ambiguous characters and contour feature). On the other hand, image quality is also a key factor that affects the result. The bad quality of images is a result of badly segmented images from the previous stage (i.e. segmentation stage) or inclined characters. In some situations, screws of the NP, dust and boundary of NP can all affect the recognition process. Figure 5-7 shows some misread character images from the used database:

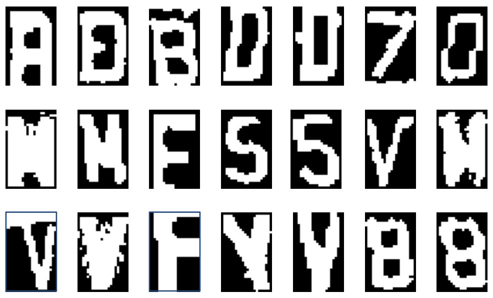


Figure 5-7: Examples of failed characters

Since there is no public UK NP character database, the alternative well known MNIST handwritten digits database [86] is used to compare the results achieved for the proposed NN and other approaches. However, the MNIST database only contains handwritten digits with varying fonts are, which is not the case for the NP character. The comparison results are shown in Table 5-1.

Table 5-1: Comparison of the proposed NN algorithm with other approaches using MNIST database

Approach	Algorithm	Error Rate (%)
The Proposed Approach	2-layer NN with 50 Neurons	5.33
[100]	DBM	0.95
[101]	DBN	1.17
[124]	2-layer NN with 800 Neurons	1.6
[125]	SVM	1.4

As shown in Table 5-1, the proposed NN has a higher error rate when compared to other approaches. However, it has a significantly lower computational complexity and requires less number of neurons to perform this complex task compared to other approaches. As the fonts of UK NP characters are unified, recognising the NP characters is relatively easier compared to recognising handwritten digits, thus the proposed NN has better result for recognising NP character than handwritten digits. On the other hand, as the main aim of this research project is to implement the entire ANPR system on one single FPGA as a low cost solution and high performance stand-alone unit, the resources are saved for other stages of ANPR (i.e. NPL and CS).

5.4 Proposed OCR Architecture

The proposed OCR architecture mainly consists of four modules: pre-processing module, hidden layer module, output layer module and index finder module. Its block diagram is shown in Figure 5-8.

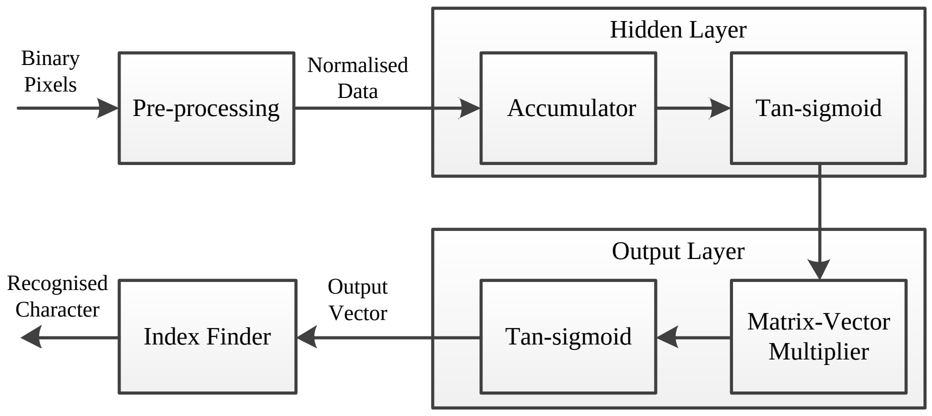


Figure 5-8: Block diagram of proposed architecture

5.4.1 Pre-processing Module

In this module, the main task is data normalisation. The output from the pre-processing module is fed to the hidden layer modules which includes a Tan-sigmoid function. Due to the range of Tan-sigmoid function values which is between [-1, 1], the input data to the hidden layer need to be normalised by the pre-processing module using Equation 5.7:

$$y = \frac{(y_{\max} - y_{\min}) \times (x - x_{\min})}{(x_{\max} - x_{\min})} + y_{\min} \tag{5.7}$$

where y_{max} and y_{min} are the expected maximum and minimum outputs, which are equal to '1' and '-1', respectively. x_{max} and x_{min} are the expected maximum and minimum inputs, which are '2' and '0', respectively. x and y are the actual input and output of the pre-processing module. Equation 7 can be simplified to Equation 5.8:

$$y = x - 1 \tag{5.8}$$

Due to use of binary images for testing, the value of x is either '0' or '1', the possible resulted values of y can be either '-1' or '0'.

5.4.2 Hidden Layer module

The hidden layer module consists of two sub-blocks:

- Accumulator block
- Tan-sigmoid block

Accumulator and Tan-sigmoid blocks are used to perform Equations 5.2 and 5.3 respectively.

The accumulator

The first block in the hidden layer is the accumulator, which is used mainly to perform the operations in Equation 5.2. Since the input data to this block can be either '-1'or '0' (i.e. the elements of vector **p**), the matrix vector multiplication $\mathbf{I} \cdot \mathbf{p}$, where $w_{s,i}$ and $p_{1,i}$ are the i^{th} weight in connection and value of the input respectively. The s^{th} element of the vector $n_1$can be calculated using Equation 5.9:

$$w_{s,i}\,p_{1,i} = \begin{cases} -w_{s,i} & (p_{1,r} = -1) \\ 0 & (p_{1,r} = 0) \end{cases} \tag{5.9}$$

The multiplications in Equation 5.9 can be replaced with a two to one multiplexer and an accumulator, which will reduce the hardware usage. The weights and biases can be attained from MATLAB once the NN has been trained. Those weights in hidden layer will be converted to their opposite numbers and stored into external RAM. In the accumulator, the weights $-w_{s,i}$ are read from external RAM and accumulated if $p_{1,\mathrm{r}}$ is equal to '-1'.

Figure 5-9 shows the internal structure of the proposed accumulator where three external 32-bit word access memories are used and each memory location contains two 16-bit weights. Each clock cycle, six weights are read from the three banks at the same time, passed through a set of multiplexers, to perform Equation 5.9, and then summed together. The result will be stored into an array $n_1$with S elements, which is used as an input to the next block.

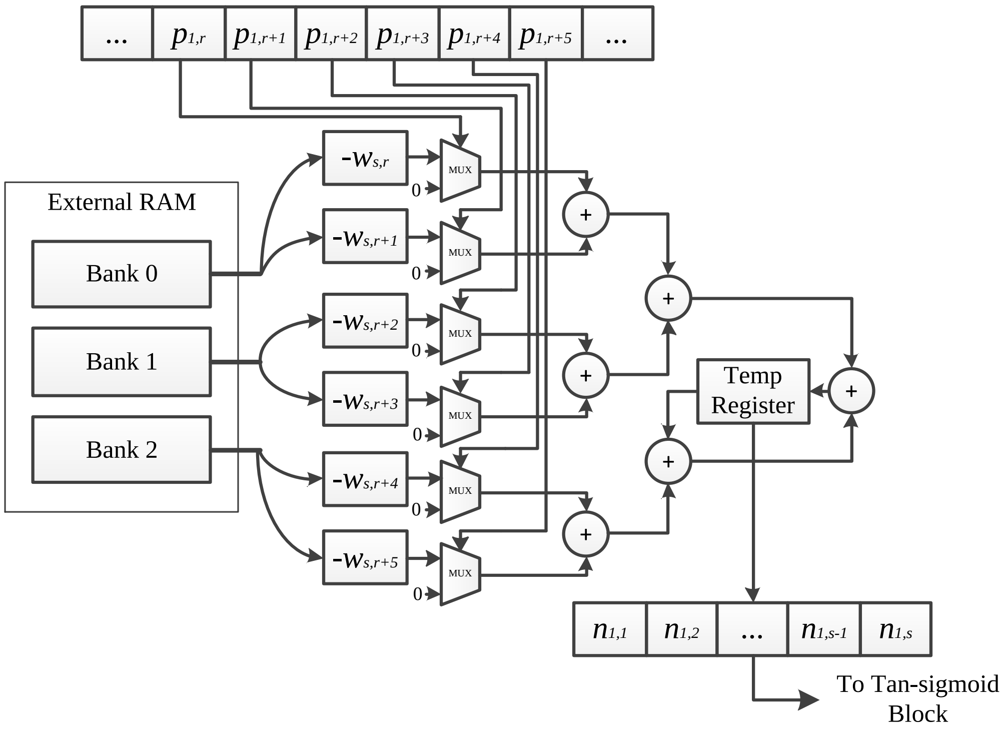


Figure 5-9: The block diagram of the accumulator

Tan-sigmoid block

The Tan-sigmoid block is used to perform the Tan-sigmoid function shown in Equation 5.10 (Figure 5-10 (a) shows its graph). When $x <$ -5 or $x >$ +5, the values of $\tanh(x)$ is closed to -1 and +1 respectively. When $-5 \le x \le +5$, the values of $\tanh(x)$ have been pre-calculated for using samples of x in this range $\{-5,-4.99,...,4.99,5\}$ where a step size 0.01 is used. Therefore, Equation 5.10 has been simplified to Equation 5.11 and the graph of the simplified Tan-sigmoid is shown in Figure 5-10 (b).

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.10}$$

$$\tanh(x) = \begin{cases} -1 & (x < -5) \\ \dfrac{e^x - e^{-x}}{e^x + e^{-x}} & (-5 \le x \le +5,\ stepsize: 0.01) \\ +1 & (x > +5) \end{cases} \tag{5.11}$$

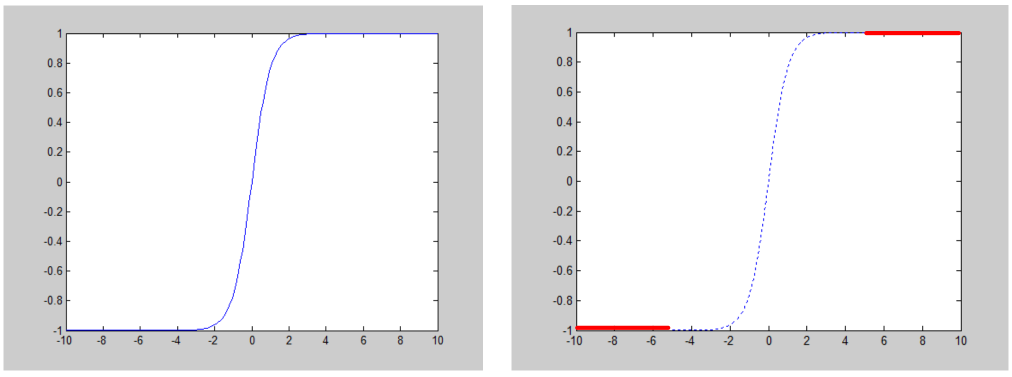


Figure 5-10: The graphs of the Tan-sigmoid. (a) Original Tan-sigmoid function, (b) Simplified Tan-sigmoid function

The results from Equation 5.11 are pre-calculated and stored in a ROM. Since the used step size is 0.01 for the range -5 to $+5$, there are 1001 results to be stored in the ROM. In order to access the correct pre-calculated results from the ROM, the following formula needs to be used to calculate the address:

$$address = (x+5) \times 100 \tag{5.12}$$

The entire ROM-based Tan-sigmoid block is shown in Figure 5-11. a_1 is a register used to store the current result from Tan-sigmoid block, which represents one element from the vector a_1 in Equation 5.3.

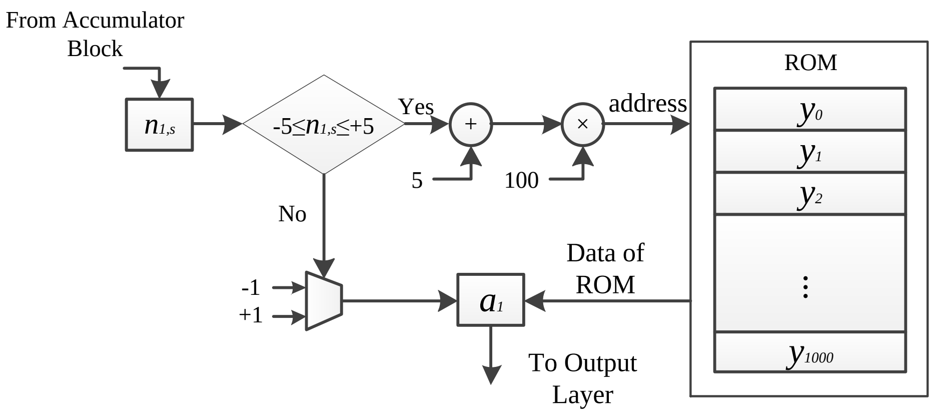


Figure 5-11: ROM-based Tan-sigmoid

5.4.3 Output Layer Module

The output layer module consists of two sub-blocks:

- Matrix-vector Multiplier
- Tan-sigmoid block

Multiplier and accumulator block is used to perform Equation 5.5, and Tan-sigmoid block is the same block used in the hidden layer.

Matrix-vector Multiplier

The matrix-vector multiplier is the first block in the output layer and its main task is to perform the matrix-vector multiplication in Equation 5.5. The overall block diagram of the matrix-vector multiplier is shown in Figure 5-12.

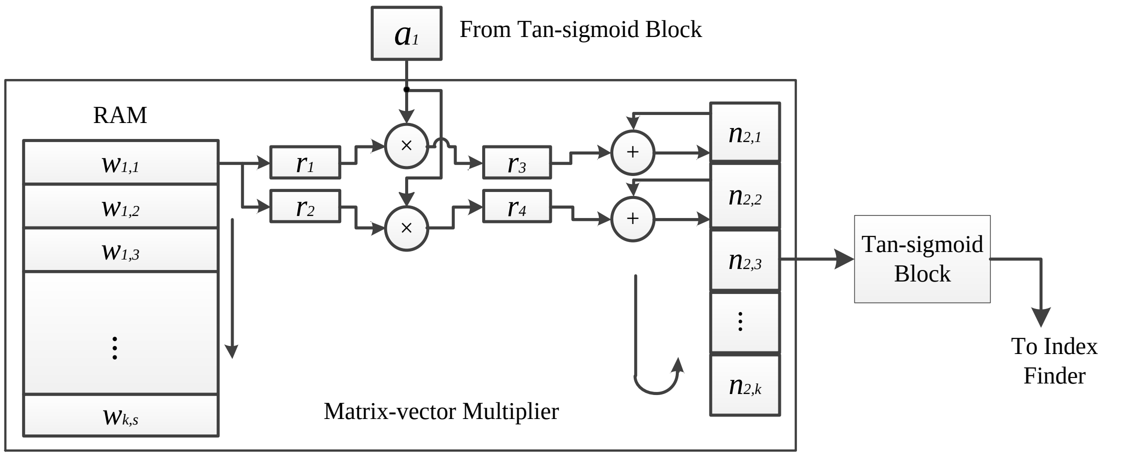


Figure 5-12: Block diagram of matrix-vector multiplier

All $K \times S$ elements of the matrix $\mathbf{L}$ are stored in one RAM column by column. Each time one element from the vector $\mathbf{a}_1$ is presented to this block from the Tan-sigmoid function block, is multiplied with all elements in the corresponding column from the matrix $\mathbf{L}$ and the results are accumulated with the previously calculated partial products. The hidden layer and the output layer work in parallel and data from the hidden layer are produced with a slower rate, which keeps the matrix-vector multiplier idle once the multiplication and accumulation operations are completed for one column until the next element from the vector $\mathbf{a}_1$ is presented.

In Figure 5-12, r_1 and r_2 are two registers that are used to store elements of matrix $\mathbf{L}$ read from the RAM. r_3 and r_4 are two registers used to store results of multiplications. $[n_{2,1}, n_{2,1}, ..., n_{2,k}]$ is an array with k elements, which is used to store the result of accumulation. Each clock cycle, $n_{2,m} = n_{2,m} + r_3$ and $n_{2,m+1} = n_{2,m+1} + r_4$ is performed and then index m is incremented by two until it meets $m = k$. Once this condition is met, m will be initialised to 0. These operations are repeated S times until all the weights from matrix $\mathbf{L}$ are read.

5.4.4 Index Finder Module

The index finder module is used to find the index of the maximum element of the output vector from the output layer and match it with the corresponding character. Figure 5-13 shows the block diagram of the index finder.

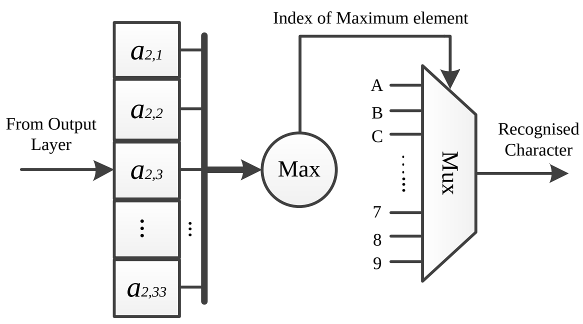


Figure 5-13: Block diagram of index finder

An array a_2 has 33 elements that come from the output layer of NN, where the index of the maximum element will be passed to select a corresponding character from a set of pre-defined character.

5.5 FPGA Implementation and Results

The proposed architecture for OCR has been simulated using the PAL Virtual Platform (PALSim) in Mentor Graphics DK Design Suite 5.3 [112]. After simulation, the architecture has been successfully implemented and verified using the Mentor Graphics RC240 FPGA development board equipped [113]. Handel-C language has been used for the hardware description of the proposed architecture [37]. For details of the experimental tools can be found in APPENDIX B.

The two main building blocks of the proposed architecture are the hidden layer and output layer shown in Figure 5-8. Pipelining has been used in their implementation and a character image can be processed in $(x \times c_1 + c_2)$ clock cycles where:

- x is the number of iterations used to execute the operations inside the hidden layer. This number depends on the number of neurons used which is 50 for the proposed algorithm.
- c_1 is the number of clock cycles to complete one iteration in the hidden layer.
- c_2 is the number of clock cycles required to complete one iteration in output layer.

Sample codes for the OCR implementation are discussed in APPENDIX C.

5.5.1 Data Representation

The general idea behind this FPGA based NN implementation is to use the weights of the NN that have been trained in MATLAB to set up an off-line NN. The weights have been represented using floating-point arithmetic in MATLAB, which provides large dynamic range and very high precision. However, floating-point arithmetic is not suitable for hardware implementation as it requires significant large amounts of on-chip resources and slows the designs down [126]. Fixed-point arithmetic is an efficient way to provide cheap fast non-integer support for small range real numbers [127]. Since the weights of the proposed NN belong to the range [-1, 1], fixed-point arithmetic has been used to represent them with the format shown in Figure 5-14.

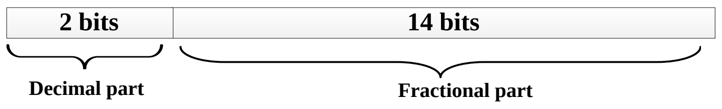
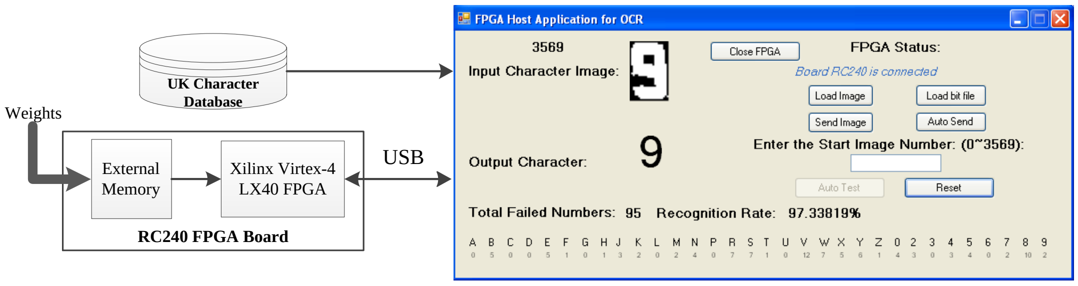


Figure 5-14: 16-bit fixed-point number representation

Since the decimal part of the weights can be '-1', '+1' or '0', two bits can be used to represent them. The number of bit in the fractional part has been set to 14, which is sufficient to give a good accuracy for this ANPR application.

5.5.2 Proposed Environment for FPGA Implementation

Figure 5-15 shows a general view of the proposed FPGA based OCR system. It consists of a Graphical User Interface (GUI), a UK characters database and the RC240 FPGA development board. The host application was implemented using Visual Studio 2008 and it gives the user the ability to select and display a character image from the UK characters database and send it to the FPGA for processing. Once processed, the recognised character from the FPGA will be displayed on the same GUI. This GUI also displays the numbers of failed cases for each character.



3569

9

Board RC240 is connected

Total Failed Numbers: 95 Recognition Rate: 97.33819%

A	B	C	D	E	F	G	H	J	K	L	M	N	P	R	S	T	U	V	W	X	Y	Z	0	2	3	4	5	6	7	8	9
0	5	0	0	5	1	0	1	3	2	0	2	4	0	7	7	1	0	12	7	5	6	1	4	3	0	3	4	0	2	10	2

Figure 5-15: The proposed FPGA based OCR system

5.5.3 FPGA Implementation Results

The same UK character database used for MATLAB implementation has been used for testing and validating the FPGA implementation. Only 95 characters were missed, which is very close to MATLAB results. Figure 16 shows a comparison between the recognition rates obtained using MATLAB and FPGA. As can be seen from Figure 16, the only difference is in only one missed case for character 'K'.

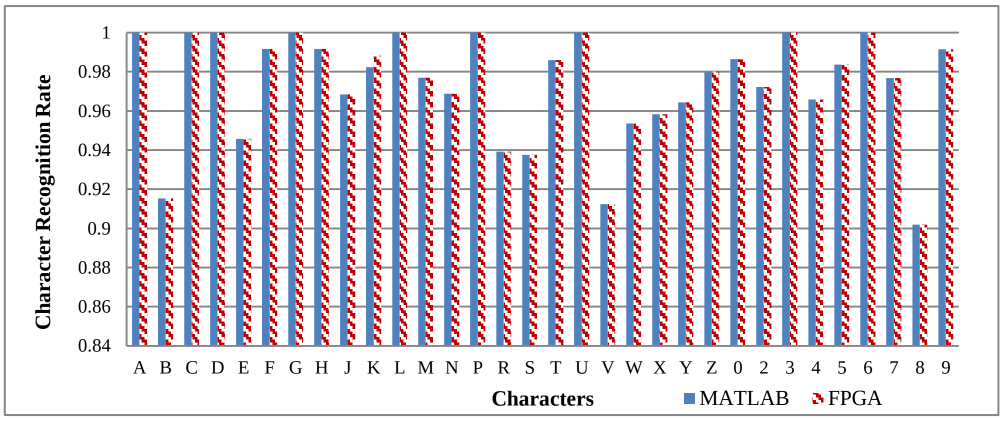


Figure 5-16: Comparison of MATLAB and FPGA implementation

The FPGA resources utilisation results of the proposed architecture using fixed-point number representation for implementing a feed-forward NN are shown in Table 5-2.

Table 5-2: Usage of on-chip Resources

On-chip Resources	Used	Available	Utilisation
Occupied Slices	4,342	18,432	23%
LUTs	7,967	36,864	21%
Flip-Flops	2,711	36,864	7%
BRAMs	5	96	5%
DSP48s	8	64	12%

Due to the fact that most of the weights are stored in the external memory, only few on-chip resources were used (i.e. block RAMs and distributed RAMs). Overall, 23% of the on-chip FPGA slices are used to implement the proposed OCR architecture. In these slices, 21% LUTs are used to implement logic operations and distributed RAMs. 7% flip-flops are mainly used for buffering the data to enable the high throughput pipeline manner in the design. 5% BRAMs are mainly used to store the weights in the output layer of the NN and lookup table of Tan-sigmoid block. The DSP48s slices are mainly used to perform the arithmetic calculations in the NN output layer. According to the previous chapters, NPL and CS implementations require 44% of the on-chip resources, therefore, the total hardware usage for the NPL, CS and OCR is 67%, which leaves 33% of the FPGA resources to be used for linking the three proposed architectures together (e.g. adapting binarisation and image resizing).

The maximum running frequency for the proposed architecture is 65.7 MHz and the number of clock cycles needed for one character image to be processed is 44148. The FPGA board used is equipped with three off-chip memory banks. All of them were used to read and process the weights in parallel. The number of clock cycles can be reduced if more memory banks are used.

The execution time for processing one image can be calculated by Equation 3.4, the proposed architecture can process one character image (34×22) and produce a result in 0.7 *ms*. Using MATLAB and Dual Core 2.4GHz, 3G RAM PC, the average processing time is 8.4 *ms* per image, which is 12 times slower than the FPGA implementation. The 12-time speed-up is due to the exploitation of the parallelism offered by FPGA when designing the proposed architecture.

The power consumption of the designed circuit has also been analysed using Xilinx XPower Analyser [115], and results obtained are shown in Table 5-3.

Table 5-3: Estimation of Power Consumption

Resource Type	Value of Power (mW)
Clocks	271
Logic	12
Signals	26
BRAMs	5
IOs	90
Clock Managers	157
Leakage	348
Total Power	909

In Table 5-3, the total dynamic power consumption for the proposed OCR implementation is 487 mW, which includes clocks power, logic power, signal power, BRAMs power and IOs power. The rest of 422mW is consumed by transistor leakage, which is depending on the chosen hardware. The total power consumption is 909 mW.

A comparison of the results achieved for the proposed FPGA based OCR system and the other hardware and software based approaches in terms of recognition rate and execution time is also shown in Table 5-4.

Table 5-4: Performance Comparison of FPGA based OCR System

OCR Technique	Character Recognition Rate (%)	Platform	Speed (*ms*)
Proposed System on FPGA	97.3	Vertex-4 FPGA	0.7
Proposed System on PC	97.3	PC 2.4 GHz	8.4
SVM [45]	97.03	PC1.8 GHz	18
SOM [105]	90.93	Vertex-4 FPGA	N/A
SVM [11]	94	DSP C6416	2.88

For all existing works presented in Table 5-4, including the proposed one, the character images are the output images from the CS stage and all of them were taken under different conditions (e.g. lighting, dirty plates) from real-world environment. The achieved results show that the proposed work outperforms the existing work in terms of recognition rate and speed, and also by comparing the results of the proposed PC and FPGA-based implementations, it can be clearly seen that the latter outperforms the former by a factor of 12, which means it presents an advantage over software-based solutions in terms of cost, size and energy consumption.

5.6 Conclusion

In this Chapter, a feed-forward ANN based OCR algorithm that meets the requirements of a real-time ANPR system has been proposed. A parallel and pipelined architecture based on the proposed algorithm has also been presented and it has been successfully implemented and tested using the Mentor Graphics RC240 FPGA development board. The proposed architecture requires only 23% of the available on-chip resources of a Virtex-4 FPGA, runs with a maximum frequency of 65.7 MHz and is capable of processing one character image in 0.7 *ms* with a successful recognition rate of 97.3% when using a database of 3700 character images.

Chapter 6:FPGA-based Number Plate Binarisation and Adjustment for ANPR Systems

6.1 Introduction

The previous chapters are mainly focus on one aspect of ANPR system, such as NPL [128-130], CS [1, 131] or OCR [132]. However, for a robust ANPR system, an exhaustive and meticulous discussion of the pre-processing stages is required. Two important pre-processing stages in ANPR systems are NP binarisation and rotation. NP image binarisation converts an 8-bit grey level NP image to a black and white image and the simplest way to perform this is to choose a fixed threshold value and classify all pixels in the image. However, brightness distribution in a NP image may cause some parts of character to be missed and noise impact to be increased after performing image binarisation due to the problem of uneven illumination. In such cases, there are two main approaches to deal with this problem which are global and local threshold based binary algorithms. One global threshold binary method is Otsu method [133], where the target and background in a given image are separated by maximizing the variances of the histogram. However, this method does not consider the correlation between the pixels in an image such as the one in NP images [134]. In this type of image, the correlation between pixels becomes more important than the grayscale values and using the global threshold with this type of images it is difficult to separate the NP characters from the background. The local binary method is often used to solve this problem as it considers the correlation between the pixels in a NP image. Adaptive local binary method is one of the local binary methods. In this method, an image is first divided into sub-blocks and then each sub-block is processed with a filter[45].

NP adjustment is also a very important pre-processing step in an ANPR system. Slanted NPs are very likely to cause failure of character segmentation in the CS stage. The main challenges in this step are how to correctly and efficiently calculate the rotation angles and adjust the NP accordingly. The most common used approaches to analyse the shape of the NP to calculate the rotation angles are pixel projection, Hough transformation or CCA [45] [71]. The main disadvantage of these methods is their computationally intensive nature to calculate the rotation angle, which could slow down the entire ANPR system.

This Chapter presents a NP binarisation algorithm which uses local binarisation method to solve the problem of uneven illumination and low complexity NP adjustment algorithms to automatically adjust NPs horizontally and vertically, which could improve the NPL result prior to CS stage. Two area/speed efficient architectures based on the proposed NP binarisation and adjustment algorithms are also presented and have been implemented and verified using a Mentor Graphics RC240 FPGA development board.

The remainder of this Chapter is organised as follows, Section 6.2 describes the proposed NP binarisation and rotation algorithms. The proposed binarisation and rotation architectures are then described in Section 6.3. Section 6.4 is concerned with FPGA implementation and discussion of the experimental results. Section 6.5 concludes the Chapter.

6.2 NP Pre-processing

In ANPR systems there are several pre-processing stages such as NP binarisation, rotation and character resizing. In this Chapter improved methods that take advantage of the NP image characteristics are presented for NP binarisation and rotation. Due to the fact of that NP images are taken from different lighting environments in real-world conditions, fixed or Global threshold binarisation methods are very likely to fail to separate the characters and background after NP binarisation. On the other hand, due to the angle of NP orientation, the NP image may have a slant and distortion which are very likely to cause failure of character segmentation in the CS stage. These two problems can significantly affect the recognition rate of the entire ANPR system. There is a need for efficient NP pre-processing algorithms and architectures to address these problems. Figure 6-1 illustrates the main building blocks of an ANPR system.

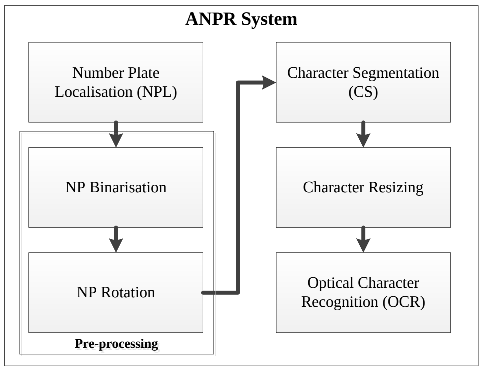

ANPR System
Number Plate Localisation (NPL)
NP Binarisation
NP Rotation
Pre-processing
Character Segmentation (CS)
Character Resizing
Optical Character Recognition (OCR)

Figure 6-1: The building blocks of an ANPR system

The inputs of the first pre-processing stage (i.e. NP binarisation) are the localised grayscale NP images, where they are binarised and then rotated in the next pre-processing stage (i.e. NP rotation). The processed images must meet the input requirements of the CS stage where the characters need to be clearly displayed in NP image and the shape and positions of the characters need to be adjusted properly.

6.2.1 NP binarisation

In ANPR systems the NP images are taken under different lighting conditions which give varying brightness distribution. If the well-known global threshold method is applied for NP image binarisation, the resulting images are not going to meet the input requirements mentioned above and are likely to fail the segmentation stage. Therefore, a local threshold method is proposed to solve this problem, which divides the entire NP image into many $m \times n$ blocks. Different thresholds are then calculated for each block and thus the entire NP image is binarised according to local illumination information.

In the proposed NP binarisation algorithm, a square $w \times w$ window is used to scan NP images column by column from left to right where each pixel from the NP image is the centre of the window. A mean filter is used to calculate the mean value for each window, and the mean value is used to calculate the local threshold.

Suppose that $f(x, y)$ denotes the grey value for pixel (*x*, *y*), which is always the centre point of a square window B with size $w \times w$. The window mean value $f_{mean}(x,y)$ is computed by Equation 6.1:

$$f_{mean}(x,y) = \frac{\sum_{(x,y)\in B} f(x,y)}{w^2} \qquad (6.1)$$

The local threshold $T(x,y)$ is then obtained by:

$$T(x,y) = f_{mean}(x,y) - \Delta t \qquad (6.2)$$

Where Δt is an threshold offset which is used to adjust the threshold value.

The binary image is obtained by:

$$b(x,y) = \begin{cases} 0, & \text{if } f(x,y) \le T(x,y) \\ 1, & \text{else} \end{cases} \qquad (6.3)$$

In this algorithm, w and Δt have significant impacts on the processing results, they are both identified by experimental tests. In the proposed system, these tests have shown that the constant value '6' for Δt has given the best binarisation results. w is determined by two other factors:

1) The size of characters in the NP image, for example, the stroke width of each character is normally around 8 pixels.

2) As the main aim of this research is to implement the entire ANPR system on one single FPGA [123] [117] [135], power of two numbers are used for w to avoid the need of multiplications as they consume a lot of on-chip FPGA resources and replace them with shifters.

Figure 6-2 shows a comparison between the use of global and local binarisation methods with different window sizes.

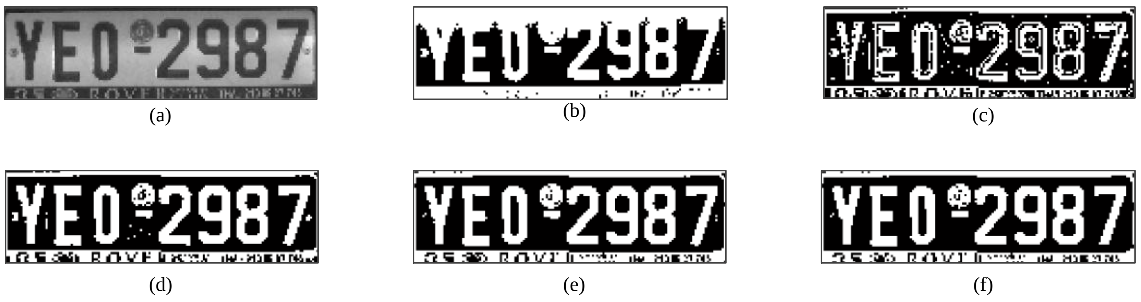


Figure 6-2: Results of using global and local binarisation methods with different window sizes. (a) Greyscale NP. (b) Global binarisation method. (c) Local binarisation method, $w = 4$. (d) Local binarisation method, $w = 8$. (e) Local binarisation method, $w = 16$. (f) Local binarisation method, $w = 32$.

As it can be seen from Figure 6-2, global binarisation method has failed to separate the NP image background and characters in the image, however, using local binarisation method better results can be achieved compared to the global method. The higher the value of *w* the better is the binarisation result, but high *w* value means more computations and hardware usage. For the proposed system and based on the obtained results from the experimental tests *w* has been chosen to be equal to 8.

6.2.2 NP Adjustment

In real-world scenarios, NP images can be slanted and distorted due to many factors such as the car and ANPR camera positions. Thus, horizontal and vertical adjustments are required after NP binarisation. In this section an algorithm for calculating the horizontal and vertical rotation angles is presented. Once the angles are found, a 2-D rotation method can be applied to adjust the NP image horizontally [136] followed by applying a cropping method to crop the Non-NP pixels from the rotated NP image. After cropping, the resulted image is vertically adjusted.

Horizontal Adjustment

The proposed algorithm calculates rotation angle by utilising the output image from the NPL stage. Existing algorithms to calculate the rotation angles require some characters analysis to obtain the angles. This affects significantly the computation time which is a very important factor in such real-time application. The proposed algorithm uses the output image from the NPL stage without the need to analyse the characters in the NP region of the image. Figure 6-3 shows and example of an input image to the NPL stage and the processed image.

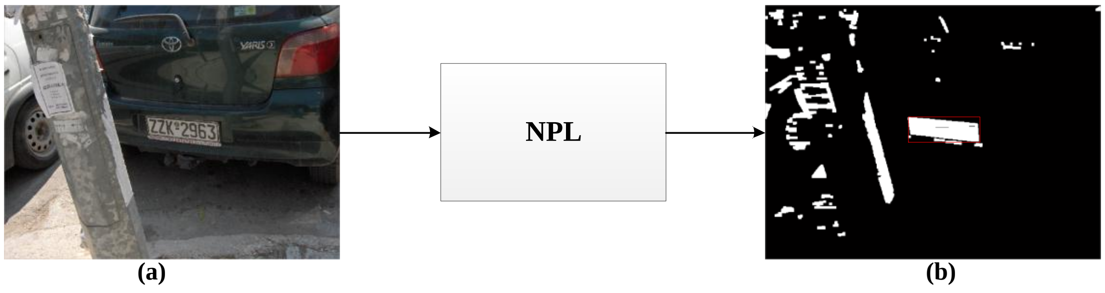


Figure 6-3: Input and output images of the NPL stage. (a) Input image. (b) Output image

In Figure 6-3, the original colour image is processed in the NPL stage which produces a binary output image. Connected Component Analysis (CCA) is used to localise the NP region in the output image and once the NP is localised, the proposed rotation algorithm will be used to calculate the rotation angle. An example of a localised NP region that needs to be binarised and adjusted is shown in Figure 6-4.

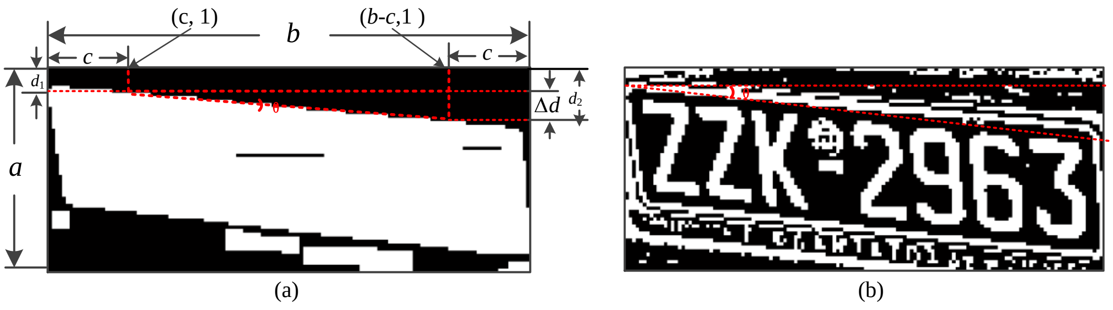


Figure 6-4: (a) Localised NP region. (b) Binarised NP image

Let $a \times b$ be the size of the localised NP region and θ be the horizontal rotation angle. As illustrated in Figure 4, θ can be calculated by the proposed algorithm uses an approach that consists of the following two steps:

1) Search vertically the first NP pixel with value ‘1’ in the localised NP region from top to bottom at positions $(1,1)$ and $(1,b-c)$. Then obtain two vertical distance values d_1 and d_2. c is an offset constant used to ensure that the first NP pixel is found

in the correct NP top boundary. According to experimental tests, when *c* is close to *b*/4 the value of θ is more precise.

2) Calculate the difference $\Delta d = d_2 - d_1$.

According to trigonometric relations, θ is calculated using the following equation:

$$\theta = \tan^{-1} \frac{\Delta d}{b - 2 \times c} \tag{6.4}$$

After obtaining θ from the localised NP region, a 2-D rotation method will be applied to rotate the binarised NP image. In this Chapter, the nearest neighbour interpolation method has been chosen to perform the horizontal rotation which is based on the following equations:

$$x_2 = \cos\theta \times (x_1 - b/2) - \sin\theta \times (y_1 - a/2) + b/2 \tag{6.5}$$

$$y_2 = \sin\theta \times (x_1 - b/2) + \cos\theta \times (y_1 - a/2) + a/2 \tag{6.6}$$

Where *a* and *b* are height and width of the binarised NP image respectively, (x_1, y_1) and (x_2, y_2) are the old and new coordinates of a given pixel on the NP image respectively.

The rotation operation produces output locations (x_2, y_2) which may not be within the boundaries of the original NP image and they will be ignored. It is worth noting that the size $a \times b$ will be kept and as a result some pixels in the boundaries of the NP will be filled with value ‘0’. Due to the fact of the rotation algorithm may produce coordinates that are not integers, the proposed method uses the nearest integer coordinate values. The binarised NP region after the rotation is shown in Figure 6-5.

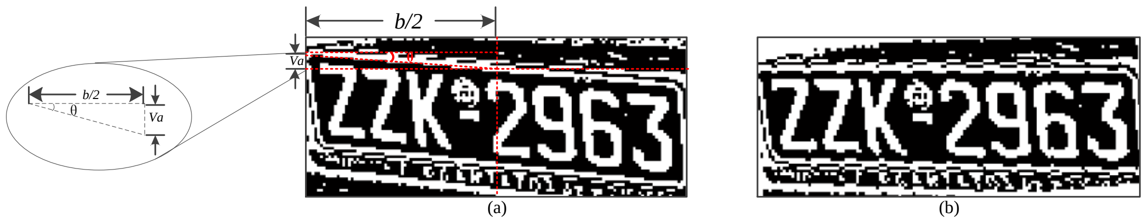


Figure 6-5: (a) Binarised NP image. (b) Rotated NP image.

As it can be seen from Figure 6-5 (b), there are many non-NP pixels in the boundaries that are generated after rotation; therefore, a cropping process is needed to reduce the height of NP image. A simple method has been proposed to perform this operation. In this method, the rotated NP image will be cropped by V_a from the top and bottom of the NP which leaves a new NP height of $a-2V_a$. The cropped NP image is shown in Figure 6-6.

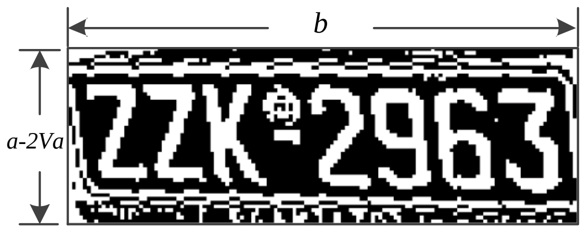


Figure 6-6: The cropped NP image

From Figure 6-5 (a), the following to trigonometric relation can be obtained:

$$tan\theta = \frac{V_a}{b/2} \tag{6.7}$$

Thus, the cropping parameter V_a can be calculated using the following equation:

$$V_a = tan\theta \times b/2 \tag{6.8}$$

Vertical Adjustment

After horizontal rotation, NP images may still need vertical rotation to adjust the slant as shown in Figure 6-7 (a). Since there is a vertical slant, it is difficult to separate the character with CCA or other projection techniques. For such cases, a vertical adjustment method, which based on horizontal shifting of pixels, is proposed and presented in this section to solve the problem.

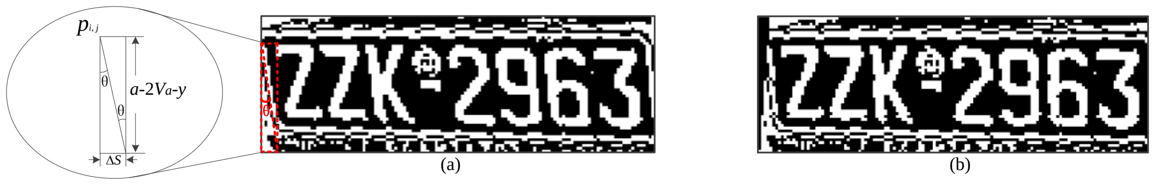


Figure 6-7: (a) NP image before vertical correction. (b) NP image after vertical correction

In Figure 6-7 (a), if an NP image was rotated horizontally with an angle θ the resulted NP image may have a vertical slant. To adjust this, the NP image must be shifted with a value Δs that depends on the horizontal rotation angle θ and a variable y which is the vertical coordinate of the pixel to be shifted. From Figure 6-7 (a):

$$tan\theta = \frac{\Delta s}{a - 2V_a - y} \tag{6.9}$$

For a pixel $P_{i,j}$, y is equal j. Thus, the shifting value Δs for each pixel can be calculated using the following equation:

$$\Delta s = (a - 2V_a - j) \times \tan\theta \tag{6.10}$$

The shifted NP image after vertical adjustment is shown in Figure 6.7 (b).

6.3 Proposed Pre-Processing Architectures

The proposed pre-processing architectures consist of the two main modules listed below:

1) ***Binarisation***: this module is used to convert the NP greyscale image to a binary image and is based on the local threshold binarisation method presented in Section 6.2.1.

2) ***Adjustment*:** this module is used to calculate the horizontal rotation angle, perform 2-D horizontal image rotation, calculate the vertical shifting value and perform the vertical shifting operation. The module is based on the methods presented in section 6.2.2.

The block diagram of the proposed pre-processing modules is shown below in Figure 6-8.

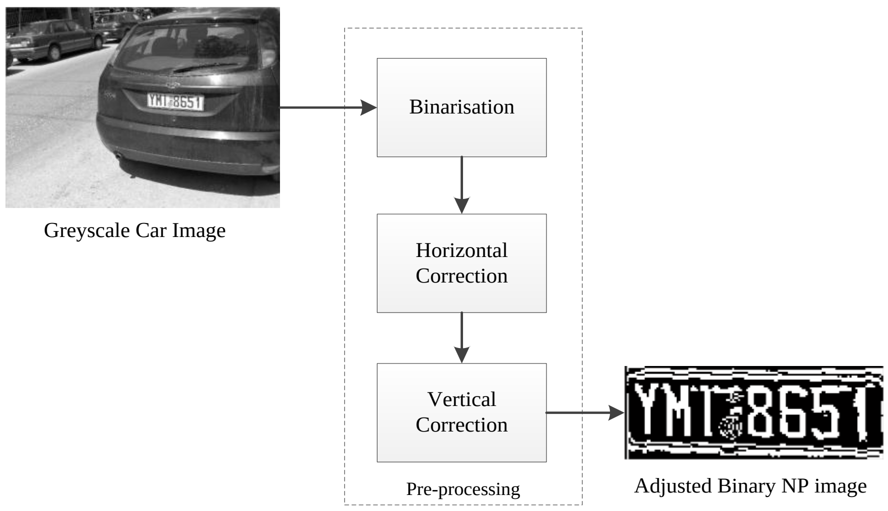


Figure 6-8: Block diagram of the pre-processing modules

6.3.1 Binarisation Module

The Binarisation Module is the first module in the pre-processing stage which consists in three blocks which are the memory reader, mean and local threshold filters. The overall block diagram of the binarisation module is shown in Figure 6-9.

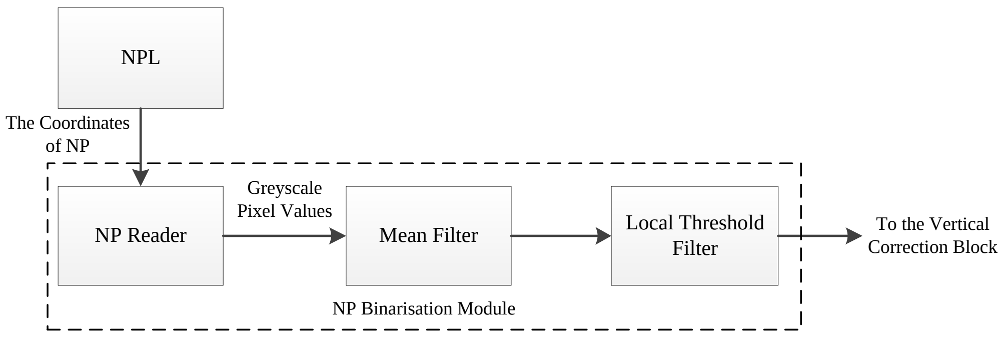


Figure 6-9: The overall block diagram of the binarisation module

The input images to the NPL stage are 640×480 colour car images from two UK and Greek databases [135]. During the NPL processing the input colour image is converted to greyscale and stored in one external memory. The greyscale image is used in further processing to generate the final outputs which consist of a binary car image and the NP region coordinates (i.e. top left and bottom right corners). The binary car image is stored in another external memory to be able to access it in parallel with the greyscale image[135].

In Figure 6-9, the NP reader first obtains the coordinates of NP region from NPL stage, and then uses them to calculate memory address of NP region. The greyscale pixel values of NP region are read from the first external memory where the greyscale image is stored and feed them to the mean filter block.

Let (x, y) denote the coordinates of a pixel in the NP rectangular region, (x_0, y_0) and (x_1, y_1) denote the left top and right bottom corner coordinates of the NP rectangular region respectively. The memory address of any pixel in the NP rectangular region can be calculated by:

$$address_{(x,y)} = 640 \times y + x, \quad x_0 \le x \le x_1 \;\&\; y_0 \le y \le y_1 \tag{6.11}$$

The pixels in the NP region are read column by column from left to right using the addresses calculated using the Equation 6.11.

Mean Filter

The mean filter is the block that performs the main process of binarisation, which consists of a window shifter and an averaging filter.

Window Shifter

The window shifter is a 8×8 matrix used to buffer pixels from the NP image. This window shifter scans the NP image column by column from left to right. Figure 6-10 shows the architecture of the window shifter.

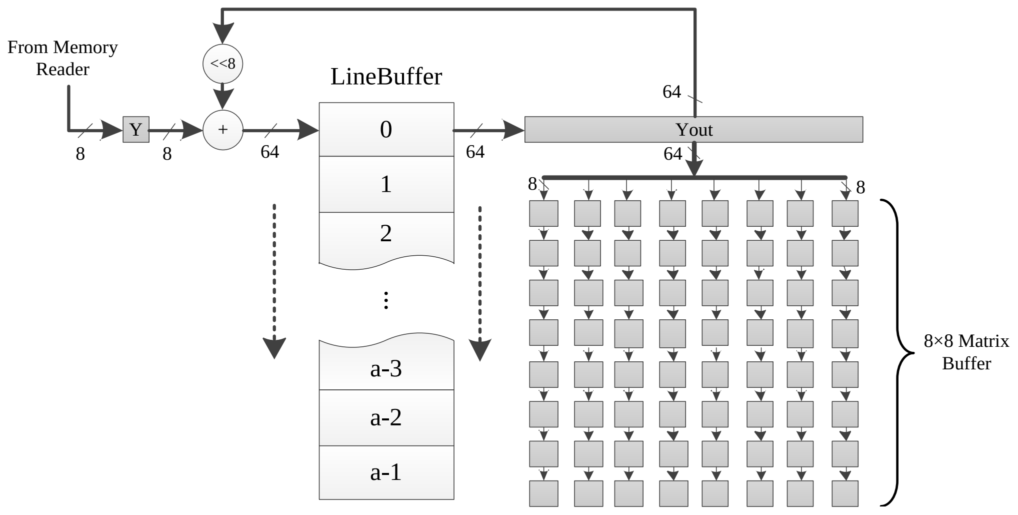


Figure 6-10: The window shifter

In Figure 6-10, ‘Y’ is an 8-bit register used to temporarily store a pixel value from the NP Reader block. ‘LineBuffer’ is an $a \times 64$ dual-port RAM, where a is the height of the NP image, used to store NP pixels from eight columns and each memory location will contain eight pixels from the same row and to do this an eight-bit shifter and an adder are used. Thus, the content of each memory location in the ‘LineBuffer’ Y' is calculated using the following equation:

$$Y' = Y + Yout << 8 \tag{6.12}$$

Following this process, the first eight pixels from the first row of the NP image will be stored in the first memory location of ‘LineBuffer’ after $a \times 8$ clock cycles. After that the next eight-pixel rows will be stored in the corresponding location every clock cycle. ‘Yout’ is a 64-bit temporary register used to store the content of one location from ‘LineBuffer’ every clock cycle. Once all eight pixels from a row are saved in ‘Yout’ it will be transferred to the first row of the 8×8 matrix buffer which will be propagated to the next row every clock cycle. This process is repeated until the 8×8 matrix is full and transferred to the next module (i.e. averaging filter).

All steps described in this section need to be repeated for all pixels in the NP image.

Averaging Filter

The averaging filter consists of 21 adders and one 8-bit right shifter and is used to calculate the mean value of the 8×8 matrix buffer by simply adding all 8-bit values and divide the result by 64. The division is performed using the 6-bit right shifter. Figure 6-11 shows the architecture of the averaging filter.

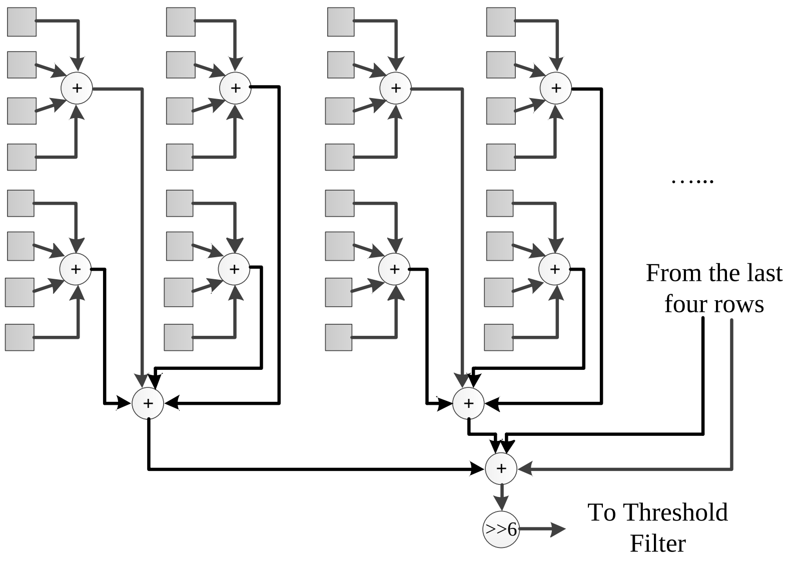


Figure 6-11: Architecture of the Average Filter

To avoid long delay paths in the hardware implementation and as it can be seen from Figure 6-11, to obtain the final sum every four elements of are added together. The average value can then be calculated by right shifting the sum by eight bits.

Local Threshold Filter

The window shifter will be applied to the whole NP greyscale image pixel by pixel and each greyscale pixel will be the centre of the window. Local threshold filter module calculates a local threshold to be used to produce the binary value of the corresponding greyscale value of a pixel to produce a binary NP image. According to Equation 6.2, the threshold can be calculated by applying the subtraction of average value and a constant Δt. This threshold is used as a condition to decide whether the current average value from the average filter should be set to ‘1’ or ‘0’. The binary pixels will be stored in a 256×1 dual-port RAM, the adjustment module will read the pixels from this RAM simultaneously.

Figure 6.12 shows the architecture of local threshold filter block.

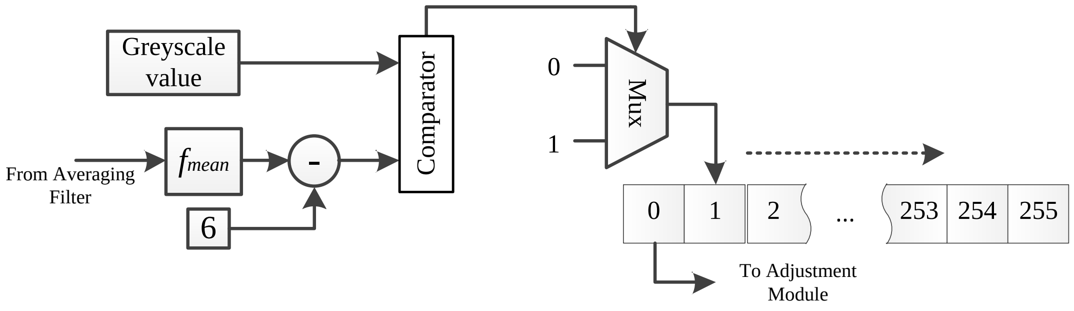


Figure 6-12: Architecture of the local threshold filter

In Figure 6-12, the address of the 256×1 dual-port RAM is incremented by one every clock cycle, when it reaches the last location it will be initialised to '0'.

6.3.2 Adjustment Module

The adjustment module consists of three blocks which are rotation angle calculator, coordinates correction block and pixels reader block. The overall block diagram of the rotation module is shown in Figure 6-13.

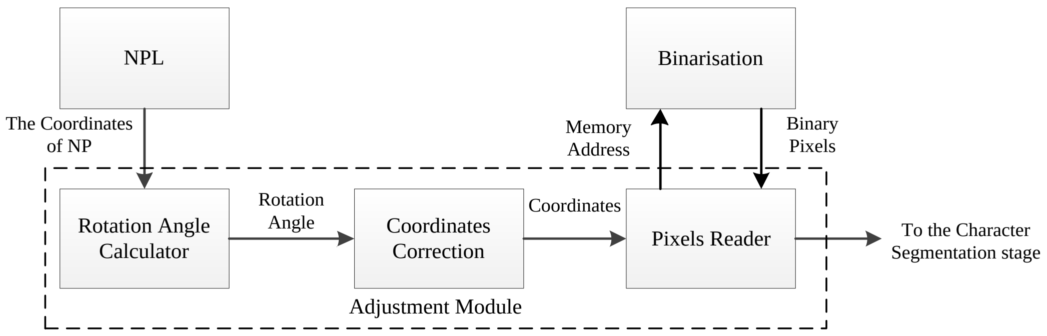


Figure 6-13: The overall block diagram of the rotation module

Rotation Angle Calculator

Since the output image from NPL stage stored in the second external memory, the rotation angle calculator uses the coordinates of NP and calculates the addresses to read the binary pixels from the memory. According to Figure 6-4 (a), the memory addresses for pixels in columns $x_0 + c$ and $x_0 + b - c$ can be calculated as follows:

$$MA_1(r_1) = 640 \times (y_0 + r_1 - 1) + x_0 + c \tag{6.13}$$

$$MA_2(r_2) = 640 \times (y_0 + r_2 - 1) + x_0 + b - c \tag{6.14}$$

Where x_0 *and* y_0 are the coordinates of the pixel in the left corner of NP, r_1 and r_2 are the NP row number, c is the offset constant and b is the width of the NP.

According to the proposed algorithm presented in section 6.2.2, the memory addresses MA_1 and MA_2 are calculated separately, where r_1 and r_2 are incremented by one until the first NP pixel with value ‘1’ is found, then their difference is calculated.

The rotation angle θ can be calculated using Equations 6.4. In order to reduce the hardware usage and improve the performance $\alpha = 1/\tan\theta$ is calculated instead of the rotation angle θ as all needed calculations in the coordinates correction block are based on α.

$$\alpha = \frac{b - 2 \times c}{\Delta d} \tag{6.15}$$

Coordinates Correction

The coordinates correction block performs horizontal and vertical adjustments. For horizontal adjustment, the main task is based on Equations 6.5 and 6.6. However, in order to avoid the calculation of trigonometric functions and reduce hardware usage, simplified equations are used in this block.

The slant angles of NP images used from UK and Greek databases are always less than 10°. Therefore, the corresponding trigonometric functions *sin* and *cos* can be replaced *tan* and value ‘1’ respectively as $\sin\theta \approx \tan\theta$ and $\cos\theta \approx 1$ when $|\theta| < 10^\circ$. Thus, Equations 6.5 and 6.6 can be simplified as follows:

$$x_2 = x_1 - \frac{(y_1 - a/2)}{\alpha} \tag{6.16}$$

$$y_2 = y_1 + \frac{(x_1 - b/2)}{\alpha} \tag{6.17}$$

For vertical adjustment, the main task is based on Equation 6.10. x_2 needs to be shifted horizontally by Δs in order to perform the vertical adjustment, thus, Equation 6.16 can be written as:

$$x_2 = x_1 - \frac{(y_1 - a/2)}{\alpha} - \Delta s \tag{6.18}$$

Figure 6-14 illustrates the architecture of horizontal and vertical adjustments.

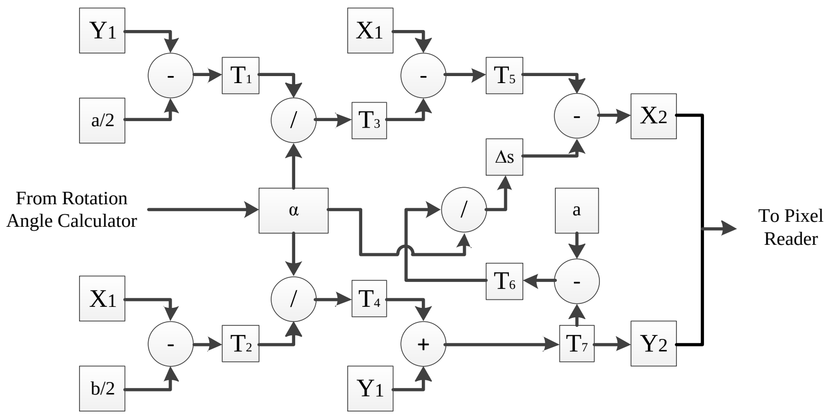


Figure 6-14: The proposed architecture for horizontal and vertical adjustments

In Figure 6-14, '$T_1, T_2, ..., T_7$' are buffers that are used to temporarily store intermediate results, which can efficiently reduce path delay and improve the throughput rate of the architecture. 'X_1, Y_1' and 'X_2, Y_2' are the registers used to store the original and new coordinates respectively. Each operation in Equations 6.17 and 6.18 requires one clock cycle and data stored in the buffers are propagated from one buffer to the next every clock cycle. The final results are passed to the pixel reader block.

Pixel Reader

The pixel reader block reads binary pixels from the dual-port RAM in the local threshold filter block from the binarisation module. In this block two functions are performed:

1) Checking the new coordinates from coordinates correction block. If new coordinates exceed the boundary of the NP, they will be discarded.

2) Calculating the reading address and read the binary pixels from the dual-port RAM in the binarisation module, then feed them to two dual-port RAMs with sizes 256×1 and 2048×1 that will be used in CS stage.

The reading address of the dual-port RAM is calculated by:

$$MA_{read} = (x_2 \times a + y_2) / 256 \tag{6.19}$$

Where x_2 and y_2 are the new coordinates, *a* is the height of the NP.

Figure 6-15 shows the proposed architecture of the pixel reader.

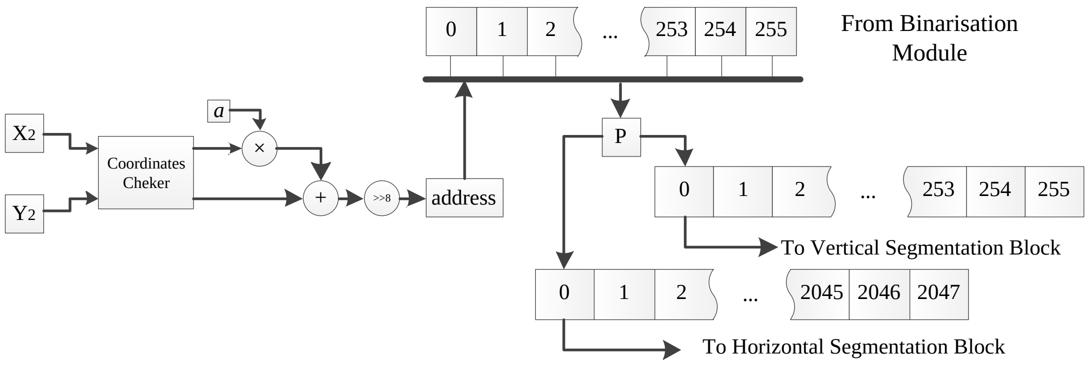


Figure 6-15: Architecture of the pixel reader

In Figure 6-15, if the new coordinates stored in ‘X_2’ and ‘Y_2’ are valid coordinates, after passing the coordinate checker block, the corresponding binary pixel is read from the dual-port RAM in the binarisation module and stored the in the temporary buffer ‘P’. The stored value in ‘P’ will be simultaneously saved in the two dual-port RAMs to be used in the CS stage.

6.4 FPGA Implementation and Results

The proposed architectures for NP binarisation and adjustment have been simulated using the PAL Virtual Platform (PALSim) [112]. After simulation, the architectures have been successfully implemented and verified using the Mentor Graphics RC240 FPGA development board [113]. Handel-C has been used for hardware description of the proposed architecture, which is a high-level language that is at the heart of a hardware compilation system known as the Mentor Graphic DK. Handel-C has additional constructs to support parallelism and pipelining [37] and [38]. For details of the experimental tools can be found in APPENDIX B.

The binarisation and adjustment modules run in parallel and pipelining has also been used in their implementation to achieve high throughput rate and an NP image can be processed by both modules in $(b+6)\times C$ clock cycles where:

- b is the width of the NP
- 6 is a constant delay that allows enough pixels to be stored in the dual-port RAM from the binarisation module
- C is the number of clock cycles required to complete binarisation for one column from the NP image

Sample codes for the binarisation and adjustment implementation are discussed in APPENDIX C.

6.4.1 Proposed Environment for NP Binarisation and Rotation on FPGA

Figure 6-16 illustrates the proposed environment for NP binarisation and adjustment implementation. It contains a host application (GUI), NP database and the RC240 FPGA development board. The host application was developed using Visual Studio 2010 and gives the user the ability to select a car image from the database, display and send it to the

FPGA for processing. Once processed, the localised, binarised and adjusted NPs are processed on the FPGA and send back to the host to be displayed in the same GUI.

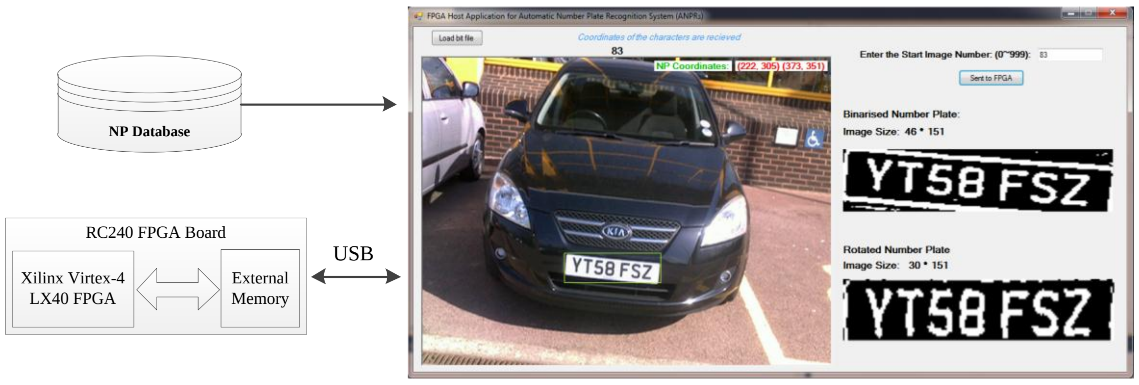

NP Database
RC240 FPGA Board
Xilinx Virtex-4 LX40 FPGA
External Memory
USB
FPGA Host Application for Automatic Number Plate Recognition System (ANPRs)
Load bit file
Coordinates of the characters are recieved
83
NP Coordinates: (222, 305) (373, 351)
Enter the Start Image Number: (0~999): 83
Sent to FPGA
Binarised Number Plate:
Image Size: 46 * 151
Rotated Number Plate
Image Size: 30 * 151
YT58 FSZ

Figure 6-16: Host application for NP binarisation and adjustment

6.4.2 Hardware Usage, Running Frequency and Power Consumption

Due to the low complexity of the proposed algorithms, the binarisation and adjustment architectures require only 9% of the on-chip FPGA resources. Table 6-1 summarises the required on-chip resources.

Table 6-1: Usage of FPGA on-chip Resources

On-chip resources	Used	Available	Utilisation
Occupied Slices	1,763	18,432	9%
LUTs	2,649	36,864	7%
Flip-Flops	1,574	36,864	4%
BRAMs	3	96	3%

9% of the on-chip FPGA slices are used to implement the proposed OCR architecture. In these slices, 8% LUTs, 3% flip-flops and 2% BRAMs are used to implement logic operations, registers and RAMs respectively. According to the previous chapters, NPL, CS and OCR implementations require 67% of the on-chip resources, therefore, the total hardware usage for the NPL, CS, OCR, binarisation and adjustment is 76%, which leaves 24% of the FPGA resources to be used for the remaining part of an ANPR system (i.e.

character resizing).

The maximum running frequency is 95.8 MHz and the number of clock cycles needed for one image to be processed is between 6297-16519, which depends on the resolution of the localised NP. The execution time for processing one frame can be calculated using Equation 3.4. The proposed architecture can process one image (18×99 - 60×300) and produce a result in $0.07-0.17\ ms$. The difference in the execution time is due to the size of the images which affect the number of clock cycles. The smaller the size of the image, the lower the number of clock cycles is required. The execution times achieved mean that the proposed architecture satisfies the minimum requirement for real-time processing. The results achieved in terms of maximum running frequency and area used for implementing this part of the ANPR system show that there is enough room to implement the whole ANPR system on a single FPGA chip.

The power consumption of the designed circuit has also been analysed using Xilinx XPower Analyser [115], and the results obtained are shown in Table 6-2.

Table 6-2: Estimation of Power Consumption

Resource Type	Value of Power (mW)
Clocks	126
Logic	12
Signals	12
BRAMs	4
IOs	33
Clock Managers	211
Leakage	344
Total Power	743

In Table 6-2, the total power consumption is 743 mW. The clocks power, logic power, signal power, BRAMs power and IOs power are formed as the dynamic power. The total dynamic and quiescent power consumption for the proposed binarisation and adjustment implementation is 290 mW and 453 mW respectively.

6.4.3 Experimental Results

MATLAB implementations of the proposed algorithms were used as a proof of concept prior to the hardware implementation where floating-point arithmetic and functions from the image processing toolbox were used. However, the FPGA implementation uses simplified integer based arithmetic. NP images from the Greek and UK databases have been used for testing the MATLAB and FPGA implementations.

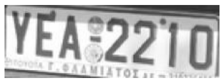
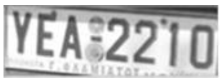
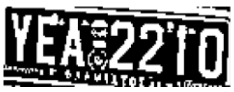
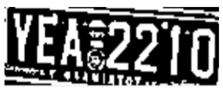
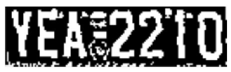
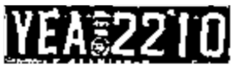
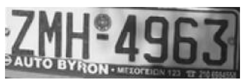
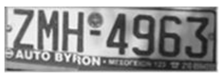
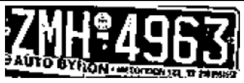
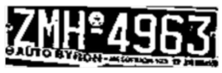
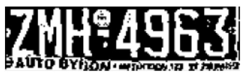
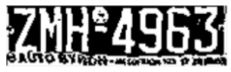
In order to compare the similarity of output images from MATLAB and FPGA implementations, the noise on the NP images is first removed using Gaussian filter and then 2-D correlation coefficient of the processed NP images are used to estimate the similarity of the two results [137]. As it can be seen from Table 6-3 the similarity of MATLAB and FPGA is around 67.2%.

Table 6-3: Similarity result for MATLAB/FPGA

	MATLAB	FPGA	Similarity
NP Example 1			72.1%
NP Example 2			63.8%
NP Example 3			64.8%
NP Example 4			68.0%

In order to compare the software and FPGA-based implementations in term of the computation speed, the proposed algorithm has also been implemented using a PC equipped with an Intel Core i7 2.8GHz and 8G RAM. Table 6-4 shows the results of the MATLAB and FPGA implementations in terms of computation time.

Table 6-4: MATLAB/FPGA result comparison

		MATLAB Implementation		FPGA	
		NP Image	Consumption Time	NP Image	Consumption Time
NP Example 1	Original greyscale NP		N/A		N/A
	Binarised NP		32 *ms*		0.11 *ms*
	Adjusted NP				
NP Example 2	Original greyscale NP		N/A		N/A
	Binarised NP		48 *ms*		0.12 *ms*
	Adjusted NP				

The images have been successfully binarised and adjusted using the proposed algorithms, where the characters are clearly isolated from each other and the vertical and horizontal positions of the NPs are properly adjusted. The FPGA processes a NP image 290 times faster than MATLAB implementation due to the low complexity of the proposed algorithms, the arithmetic techniques used, parallelism and pipelining exploited in the hardware implementation of the proposed architectures.

6.5 Conclusion

In this Chapter, two optimised low complexity NP binarisation and adjustment algorithms have been proposed to successfully link NPL and CS stages. Efficient area/speed architectures based on the proposed algorithms have also been presented and successfully implemented and tested using the Mentor Graphics RC240 FPGA development board, which together require only 9% of the available on-chip resources of a Virtex-4 FPGA, run with a maximum frequency of 95.8 MHz and are capable of processing one image in $0.07 - 0.17ms$.

Chapter 7: Standard Definition ANPR System on FPGA and an Approach to Extend it to HD

7.1 Introduction

Each ANPR stage is discussed separately in the previous chapter, as the main aim of this research project is to implement the entire ANPR system on a single FPGA chip that can be placed within an ANPR camera housing to create a stand-alone unit that can drastically improve energy efficiency and remove the installation and cabling costs of bulky PCs situated in expensive, cooled, waterproof roadside cabinets.

A range of image processing algorithms for each stage of the ANPR system and corresponding new FPGA architectures have been proposed in [117, 123, 132, 135, 138, 139]. This Chapter describes the linking process of previously designed architectures from each stage of the ANPR system to be implemented on a single stand-alone FPGA-based processing unit. By optimising the ANPR algorithms to take specific advantage of technical features and innovations available within FPGAs, such as parallelism computing feature, low power consumption, development time, and vast on-chip resources, it will be possible to replace the powerful roadside computers with small in-camera dedicated platforms.

In addition to the proposed ANPR system, this Chapter also introduces a preliminary research for how the proposed Standard Definition (SD) NPL algorithm can be applied to HD NPL system.

The rest of this Chapter is organized as follows. The description of the FPGA based ANPR system is given in section 7.2. Section 7.3 is concerned with the implementation of the FPGA based ANPR system. Section 7.4 introduces a preliminary research for HD NPL. Finally a conclusion is given in section 7.5.

7.2 Proposed FPGA based ANPR System

A typical ANPR system consists of three main stages: NPL, CS and OCR. In addition to these main stages, there are few pre-processing stages needed to link the three main stages. Figure 7-1 demonstrates the main building blocks of an ANPR system.

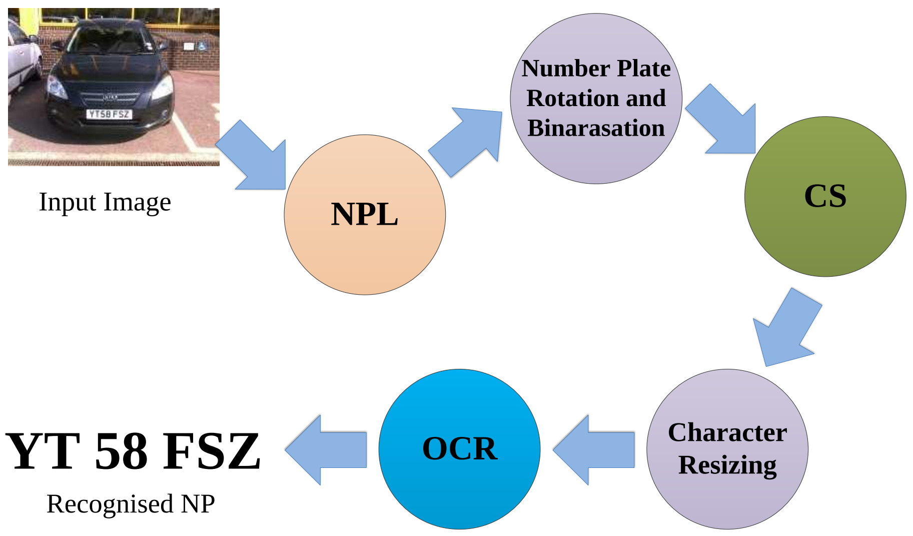


Figure 7-1: Main building blocks of an ANPR system

The next sections describe what has been used to link each block to the next one.

7.2.1 Number Plate Localisation Module

The main aim of the NPL module is to correctly localise the NP within the original input car image, where two sets of coordinates for the NP's across corners are detected. To link the NPL module to the CS module there is a need to binaries and adjust the output NP images from the NPL module. Binarisation and adjustments modules proposed in Chapter 6 are used to achieve this. Figure 7-2 demonstrates the linking process of the NPL module, binarisation and rotation blocks.

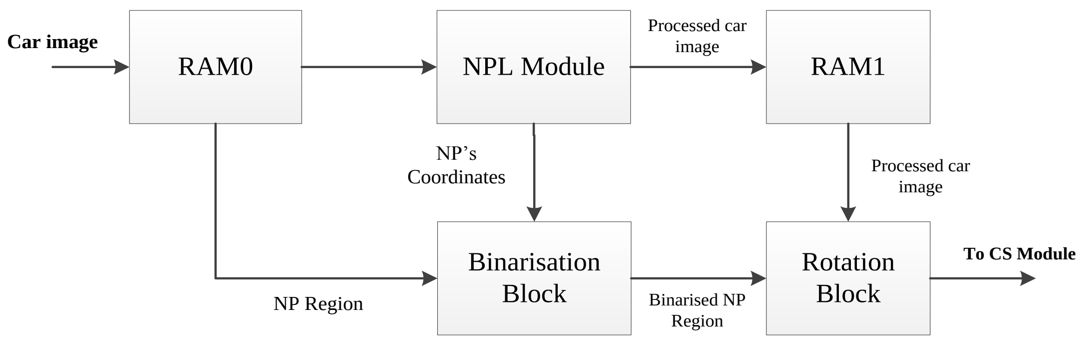


Figure 7-2: Process of the NPL module, binarisation and rotation blocks

The NPL module reads and processes first the original car image. Once the coordinates of top left and bottom right corners of the NP are found, the binarisation block starts to read the NP region from the first RAM (i.e. RAM0). Simultaneously, the rotation block starts to read the processed car image from the second RAM (i.e. RAM1), and then it will calculate the rotation angle. Finally, the rotation block will send the pixels of the rotated NP to the CS module.

7.2.2 Character Segmentation Module

The main aim of the CS module is to correctly segment the characters within the NP region, where each segmented character is passed to the next module. However, since the OCR module needs all the segmented characters to have the same size, each segmented character is resized in the character resizing block before it is passed to the OCR module.

Figure 7-3 shows the linking process of the CS and character resizing modules.

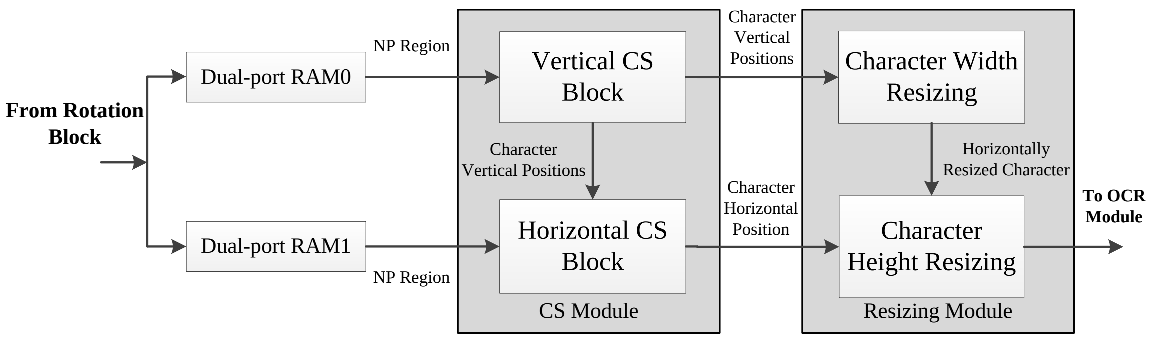


Figure 7-3: Process of the CS and character resizing modules

The pixels within the adjusted NP region are simultaneously stored in two dual-port RAMs (i.e. dual-port RAM0 and RAM1). The vertical CS Module reads the NP region from the first dual-port RAM to calculate the vertical positions of each character. Once the vertical positions of the character are obtained, they will be passed to the character width resizing and the horizontal CS blocks which will work in parallel to resize the width of the character and calculate the horizontal positions of the character respectively. The character height resizing block will use the horizontal positions of the character, which is the output of the horizontal CS block, to resize the height of the character. Finally, the fully resized characters are sent to the OCR module serially.

7.2.3 Optical Character Recognition Module

The main aim of the OCR module is to correctly recognise the segmented characters. Because the OCR module is slower than the CS and resizing modules, there is a need for a buffer to be placed between these modules and the OCR module, which can temperately store the resized characters. Another dual-port RAM is used for this purpose. Figure 7-4 shows the linking process of the OCR module to the previous modules.

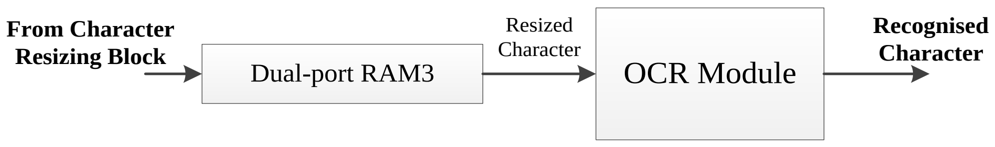


Figure 7-4: Process of the OCR module

The OCR module reads the pixels of each resized character from the dual-port RAM3, and processes the pixels at same time. Each recognised character from a segmented NP is stored in an array.

7.3 FPGA Implementation and Results

The whole ANPR system has been simulated using the PAL Virtual Platform (PALSim) [112]. After simulation, the system has been successfully implemented and verified using the Mentor Graphics RC240 FPGA development board [113]. For details of the experimental tools can be found in APPENDIX B.

The ANPR modules run in parallel and pipelining has also been used in their implementation to achieve high throughput rate and a car image can be processed by the modules in $C_1 + C_2 \times n$ clock cycles where:

- n is the number of character within a NP image
- C_1 is the number of clock cycles required to localise the NP within a car image
- C_2 is the number of clock cycles required to recognise each character within a NP image

Sample codes for the entire ANPR implementation are introduced in APPENDIX C.

Due to the external storage limitation on the RC240 development board, a new image cannot be loaded to the external RAM while the OCR stage is being processed; therefore, the proposed ANPR system requires a total of $(C_1 + (C_2 \times n)) \times x$ clock cycles to process x car images when the car images are continually sent to the system. However, if the FPGA board is equipped with extra memory banks, the NPL and OCR modules can access the external RAMs simultaneously, which will significantly reduce the total clock cycles to $C_1 + (C_2 \times n) \times x$ for processing x car images.

7.3.1 Proposed Environment for ANPR on FPGA

The proposed ANPR system has two main parts: RC240 FPGA development board and a GUI running in a host application. The RC240 FPGA development board performs the calculation of the ANPR system, and sends the results to the host. The host sends the original car images to the FPGA board, and displays the processed images and characters on the GUI. Once a car image is sent to FPGA board it will be processed by the FPGA, and then the localised, adjusted, segmented and recognised NPs are sent back to the host to be displayed in the same GUI. Figure 7-5 shows the GUI host.

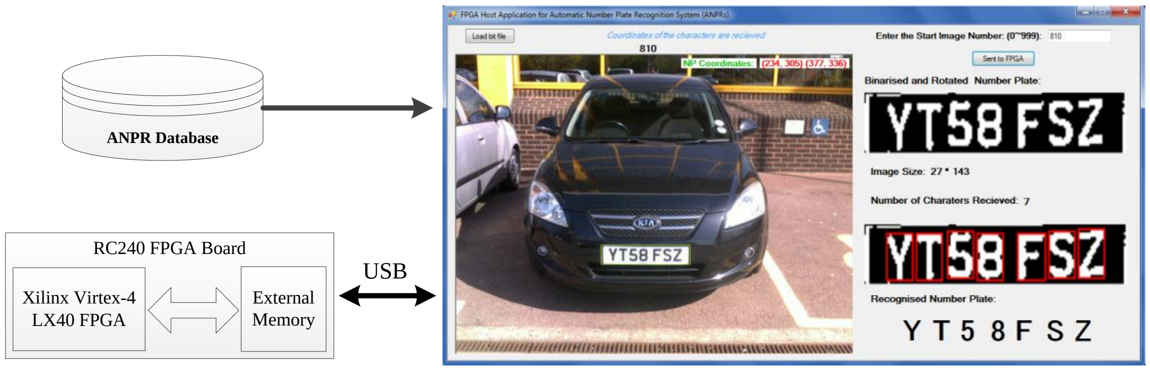

ANPR Database
RC240 FPGA Board
Xilinx Virtex-4 LX40 FPGA
External Memory
USB
FPGA Host Application for Automatic Number Plate Recognition System (ANPRs)
Load bit file
Coordinates of the characters are recieved
810
NP Coordinates:
(234, 305) (377, 336)
Enter the Start Image Number: (0~999):
810
Sent to FPGA
Binarised and Rotated Number Plate:
Image Size: 27 * 143
Number of Charaters Recieved: 7
Recognised Number Plate:
Y T 5 8 F S Z

Figure 7-5: The GUI host

7.3.2 Hardware Usage, Running Frequency and Power Consumption

The entire ANPR system consists of NPL, CS, OCR and pre-processing modules, those architectures require 80% of the on-chip resources of the Virtex-4 LX40 FPGA. Table 7-1 summarises the required on-chip resources.

Table 7-1: Usage of FPGA on-chip Resources

On-chip resources	Used	Available	Utilisation
Occupied Slices	14,775	18,432	80%
LUTs	22,556	36,864	61%
Flip-Flops	8,547	36,864	23%
Block Rams	30	96	31%
DSP48s	12	64	18%

As shown in Table 7-1, the total on-chip usage is 80% for the entire ANPR system, leaving 20% of the FPGA area to be used for other purposes, for example, communication and display units, which allows the FPGA to act as a stand-alone unit.

The maximum running frequency is 57.6 MHz and the number of clock cycles needed for one image to be processed is between 506686-683278, which depends on numbers of the characters within the NP image. The execution time for processing one frame can be calculated using Equation 3.4, the proposed architecture can process one image and produce a result in 11 *ms*. This means that the proposed architecture satisfies the minimum requirement for real-time processing.

The power consumption of the designed circuit has also been analysed using Xilinx XPower Analyser [115], and the results obtained are shown in Table 7-2.

Table 7-2: Estimation of Power Consumption

Resource Type	Value of Power (mW)
Clocks	282
Logic	13
Signals	13
BRAMs	12
IOs	30
Clock Managers	211
Leakage	348
Total Power	910

Table 7-2 shows that the total power consumption of the proposed architectures is 910 mW, which is comprised of 459 mW dynamic power and 451 mw quiescent power. The total power consumption is very low compared to computer based ANPR systems.

As far as the overall performance calculation is concerned, this can be calculated by

$$A = (L \times S \times R)\% \tag{7.1}$$

where A is the overall system accuracy, and L, S, and R are the percentage of successful NPL, CS and OCR rates, respectively. According to the previous works in Chapter 3-5, the NPL, CS and OCR rates are 97.8%, 97.7% and 97.3%, respectively. Therefore, the overall system accuracy is around 93.0%.

7.3.3 Comparison with Existing Work

A comparison of the experimental computational speed and successful rate with existing PC, DSP and FPGA based implementations of ANPR system is shown in Table 7-3.

Table 7-3: Performance Comparison

ANPR System	Character Set	Hardware Platform	Successful Rate (%)	Speed (*ms*)
[11]	Australia	TI C64 DSP	85	52.11
[105]	Turkey	FPGA Virtex-4	73	500
[45]	Japan	PC Intel Core 1.8 GHz	93.54	284
[43]	Chinese	PC 3 GHz	93.9	293
Proposed FPGA based ANPR System	UK	FPGA Virtex-4	93.0	11

As shown in Table 7-3 and by comparing the results of the PC and FPGA-based implementations, the main advantage of hardware based ANPR systems is the fast processing speed which is of particular interest in real-time environments. The proposed FPGA based ANPR system outperforms the fastest software and hardware based ANPR systems by a factor of 26 and 4.7 respectively, it also outperforms the existing hardware solutions in terms of accuracy. Although the recognition rate of the proposed system is close to that of some PC-based systems, it presents an advantage over software-based solutions in terms of cost, size and energy consumption.

7.4 A preliminary research for HD NPL

Recently, HD cameras become an important trend in ANPR because higher image quality and resolution can provide better performance for CS and OCR after NPL. However, most known approaches for SD NPL are not suitable for real-time HD image processing as the real-time requirement will not be met due to the computationally intensive cost of localising NP. In this section, an approach is proposed to localise the NP image from a HD image using the SD NPL algorithm without significantly increasing the computational cost.

7.4.1 Proposed Approach:

In general, the NP region in a NP image occupies only a small percentage of the total image surface and NPL algorithms search all pixels from an input NP image to localise the NP region. With a HD image, the search time for the NP region can be increased by up to 9-fold compared to a SD image. The proposed approach significantly enhances the HD NPL processing speed without losing the quality of the HD NP for the rest of the ANPR system. As the main area of interest in an input ANPR image is only the NP region, the proposed approach firstly localise the position of the NP from a SD image resized from the input HD image, and then calculate the coordinates of NP region in the HD image using the obtained position coordinates of the NP region in the SD image. The block diagram of the proposed approach is shown in Figure 8-1.

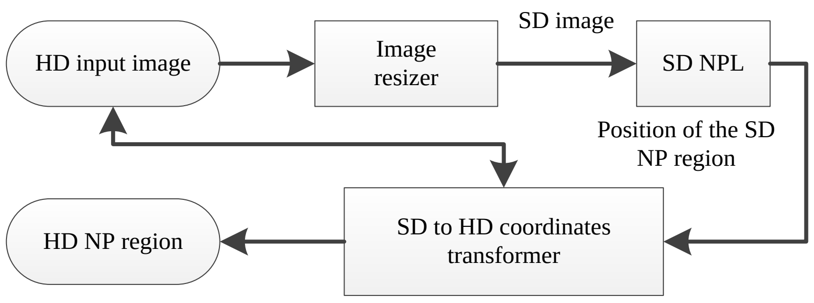

HD input image
Image resizer
SD image
SD NPL
Position of the SD NP region
HD NP region
SD to HD coordinates transformer

Figure 7-6: HD NPL block diagram

As shown in Figure 7-6, a HD input image is first resized to a SD image using a resizing algorithm. In this Chapter, the nearest neighbour interpolation algorithm has been used. Let assume that the sizes of the HD input image and the resized HD image are $A\times B$ and $a\times b$ respectively, the width ratio α_w and height ratio α_h are equal to A/a and B/b respectively. The pixels within the resized HD image are sampled from the original HD input image using the following equations:

$$X_r(n_w) = 1 + n_w \times \alpha_w \tag{7.2}$$

$$Y_r(n_h) = 1 + n_h \times \alpha_h \tag{7.3}$$

where $X_r(n_w)$ and $Y_r(n_h)$ are resized coordinates at horizontal n_w and vertical n_h positions respectively, n_w and n_h are integers that belong to the ranges $\{0,1,2,...,b-1\}$ and $\{0,1,2,...,a-1\}$ respectively.

The total pixels in the resized HD image are reduced by a factor of $\alpha_w \times \alpha_h$, and because of the high quality and resolution of HD images, the NP region in the input HD image is larger than the NP region in the resized image (i.e. SD image). The resizing process will only affect the size of the NP region and not its features, which means the proposed SD NPL algorithm [135] can be used to localise the NP region in the resized HD image. Figure 7-7 shows an example of the input HD image and the resized one.

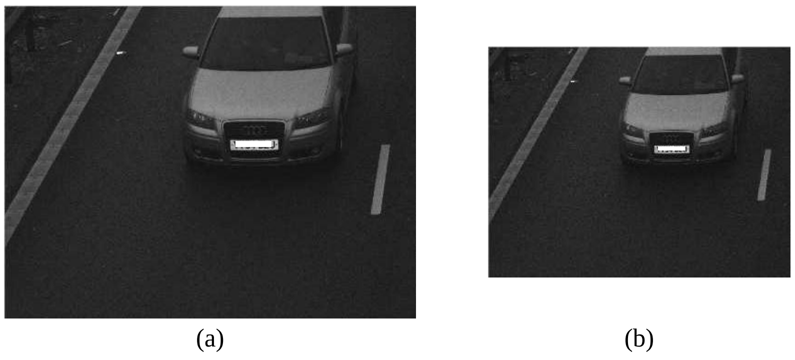


Figure 7-7: An example of the HD and resized HD images. (a) The HD input image (1392×1040). (b) The resized HD image (640×480)

The used SD NPL algorithm is mainly based on morphological *open* and *close* operations where two morphological *open* operations are used to enhance the NP features and a morphological *close* operation is used to highlight the NP region [135].

Once the SD NP region is localised, a SD to HD coordinates transformer is used to calculate the coordinates of the HD NP region based on the coordinates of the SD NP region. Let (x_0, y_0) and (x_1, y_1) are the NP's left-up and right-down corners within the SD image respectively, (X_0, Y_0) and (X_1, Y_1) are the expected NP's left-up and right-down corners within the HD image respectively. The following equations are used to calculate the coordinates of the HD NP region:

$$X_0 = x_0 \times \alpha_w \tag{7.4}$$

$$Y_0 = y_0 \times \alpha_h \tag{7.5}$$

$$X_1 = x_1 \times \alpha_w \tag{7.6}$$

$$Y_1 = y_1 \times \alpha_h \tag{7.7}$$

The HD NP region is retrieved from the HD input image based on the calculated positions (X_0, Y_0) and (X_1, Y_1).

Experimental Results and Analysis: In order to verify the proposed approach, the proposed SD NPL algorithm, applied to resized and original HD images, have been implemented in MATLAB. HD images with 1392×1040 resolution taken from the motorway using HD cameras have been used for testing. The PC used to conduct the experiments is an Intel Core i7 2.8GHz with 8G RAM.

In the proposed implementation, the HD input image is first resized to 640×480 SD image, and then the proposed SD NPL algorithm is used to localise the NP region. Once the NP region is detected, equations (7.4-7.7) are used to calculate the corresponding coordinates for the HD NP region from the original HD image. Table 7-4 shows the comparison in terms of processing speed when using the proposed approach and the SD NPL approach [135] with SD images from normal SD cameras and original HD images.

Table 7-4: MATLAB implementation results

Implementation	Input image (pixels)	Scanned pixels	Processing speed (*ms*)
SD NPL [135] (SD images)	307,200	307,200	143
Proposed approach (HD images)	1,447,680	307,200	168
SD NPL (HD images)	1,447,680	1,447,680	507

According to Table 7-4, using the proposed approach, only 21% of the original HD image is scanned which significantly increase the processing speed by a factor of 3 compared to scanning the entire HD input image. Whilst compared to a normal SD NPL implementation without resizing, the processing speed for the proposed approach is close to SD NPL processing speed. This means the resizing and SD to HD coordinate transformations processes increase the processing speed by just a factor of 1.17.

A speed and area-efficient architecture for the image resizer has been implemented using the Mentor Graphics RC240 FPGA development board. Due to the low complexity of the resizing algorithm, the proposed architecture requires only 3% of the on-chip FPGA resources.

Using the proposed approach, the HD NPL which consists of the SD NPL [135], the image resizer and the HD to SD coordinates transformer, requires 36% of the FPGA on-chip resources, which leaves 64% to be used for implementing the next stages of an ANPR system (i.e. CS and OCR). The maximum running frequency for the image resizing module is 86.9 MHz, and the number of clock cycles needed to resize one image is 307206, thus the execution time for resizing one image is 3.5 *ms*. The proposed SD NPL module process one image in 4.7 *ms* [1]. As the image resizer and the SD NPL modules can run in parallel, an HD image can be fully processed in 4.7 *ms* by the HD NPL module. This means that the proposed architecture satisfies the minimum requirement for real-time processing, and it is faster than software based implementation by a factor of 36.

Existing works have shown that meeting real-time processing constraints using software based solutions to perform HD NPL in a uniprocessor system is a complex task. In [140], the maximum achieved number of frames per second (fps) is 15, which is far from real-time processing requirements (i.e. 25 fps). In [141], an operator context scanning (OCS) algorithm to accelerate the searching of the NP region within the HD image was proposed, and results have shown that the system is capable of processing 22 fps. When compared with the proposed hardware-based NPL implementation, the proposed work can achieve more than 200 fps, and the proposed FPGA-based system can be used as a viable solution to replace software-based solutions where cost, size and energy consumption will be reduced.

7.5 Conclusion

In this Chapter, all three stages of an ANPR system (i.e. NPL, CS and OCR) have been successfully linked together, implemented and tested using the Mentor Graphics RC240 FPGA development board, which requires only 80% of the available on-chip slices of a Virtex-4 FPGA, runs with a maximum frequency of 57.6 MHz and is capable of processing one image in 11 *ms* with a successful recognition rate of 93%.

The achieved results show that the entire ANPR system can be implemented on a single FPGA chip, which can be placed within an ANPR camera housing to create a stand-alone unit which will drastically improve energy efficiency and remove the installation and cabling costs of bulky PCs situated in expensive, cooled, waterproof roadside cabinets.

In addition to the above, this Chapter also presents a solution that utilises a SD NPL algorithm to localise a NP from a HD image under real-time constraint, which significantly increases the processing speed for HD NPL.

Chapter 8: Conclusions and Future Work

8.1 Introduction

ANPRs are rapidly becoming used for a vast number of applications to track, identify and monitor moving vehicles by automatically extracting their NPs. The fundamental requirements of an ANPR system are image capture using an ANPR camera and processing of the captured image. The image processing part, which is a computationally intensive task, includes three stages: NPL, CS, and OCR. The ANPR algorithms should operate fast enough to fulfil the requirements of real-time operation, which means they should not miss a single vehicle that moves through the camera [135]. Consequently, the common hardware choice for ANPR implementation is often high performance workstations. However, the cost, compactness and power issues that come with these traditional solutions motivate the search for other platforms. Developments in digital circuit technology, especially rapid development of FPGAs, offer alternative way to provide a low cost acceleration for such computationally intensive tasks.

The main goal of the work reported in this thesis is to design and implement efficient and novel architectures for ANPR system using different design methodologies for accelerating digital image processing algorithms.

A range of image processing algorithms and architectures for each ANPR stage have been developed and optimised, which can take specific advantage of technical features and innovations available within new FPGAs, such as low power consumption, development time, and vast on-chip resources, it will be possible to replace the powerful roadside computers with small in-camera dedicated platforms.

The proposed ANPR architectures have been implemented and verified using the Mentor Graphics RC240 FPGA development board equipped with a 4M Gates Xilinx Virtex-4 LX40.

In the rest of this Chapter results obtained throughout this research are summarised and evaluated. Some possible routes to be investigated for a future extension of this work are also provided.

8.2 Evaluation of Results and Contributions

The preceding Chapters described different design methodologies used for efficient design and implementation of image processing algorithms for ANPR System on FPGAs. This section is concerned with the evaluation of the work presented in these Chapters.

8.2.1 Measurement of Success

In this project the performance measurement and comparison of the proposed algorithms and architectures were presented.

The comparison was based on the computation time, on-chip area required and system accuracy. Computation time and on-chip area required that depend on the various design optimisation strategies and parameters, for example, resolution of input car image and data format. In the case of the implementation of these architectures, the measurement was given on the number of slices, LUTs, Flip-flops, BRAMs, DSP48s, clock cycles and the maximum running frequency of the design. In the case of the system accuracy, A MATLAB implementation of the proposed algorithms were used as a proof of concept prior to the hardware implementation, a comparison of the proposed software and hardware implementations was given for each stage of the proposed system, where different criteria such as type and colour plates, illumination conditions, various angles of vision, and indoor or outdoor images were considered in the tested databases.

8.2.2 Results Achieved

A set of goals were specified in Chapter 2, which would determine the success of the work presented in this thesis. Taking into account the initial objectives, the following points can be made about the achievements of the project:

- A low complexity and stable NPL algorithm suitable for a single FPGA implementation has been developed, where a novel NP feature extraction and enhancing method based on two morphological operations and an image subtraction operation was proposed. Results obtained have shown stable performances in terms of the successful recognition rate and computation time in comparison with existing software systems [135] and [138];
- A novel efficient architecture based on the proposed NPL algorithm has been designed and successfully implemented on a FPGA. The performance in terms of the area used, the maximum running frequency and successful recognition rate of the proposed architectures has been assessed and has shown that the proposed system has a higher frequency and recognition rate, less processing time when compared with existing hardware based systems [135] and [117];
- An improved low complexity and stable CS algorithm based on pixel projection and morphological operations has been developed. The improvement in the proposed CS algorithm has significantly reduced the processing time of segmentation and obtain more precise horizontal and vertical segmentation result when compared with existing software systems [123];
- A novel real-time architecture based on the proposed enhanced CS algorithm and its area/speed efficient implementation have been proposed. Parallelism offered by FPGAs and pipelining technique have also been exploited to achieve high running frequency and throughput rate. The use of multipliers has been avoided in some building blocks from the proposed architecture which significantly reduces on-chip resources usage. Results obtained show that the proposed system outperforms existing state-of-the-art hardware based implementations in terms of segmentation rate and processing speed [123];
- A low complexity and stable OCR algorithm based on feed-forward NN has been

developed. An area/speed efficient architecture based on the proposed algorithm has also been designed and successfully implemented on a FPGA. The proposed architecture for implementing the two layer feed-forward NN on FPGA can process a large number of neurons in a pipelined manner to achieve high running frequency and throughput rate. The use of multipliers has been avoided in the first layer from the proposed architecture, which significantly reduces on-chip resources usage. The implementation has achieved higher character recognition rate and processing speed compared to existing hardware based OCR implementations for ANPR systems [132];

- A low computational complexity NP binarisation and adjustment methods have been developed to solve an important practical issue for real-time ANPR system and the corresponding area/speed efficient architectures based on the proposed algorithms have been successfully implemented on a FPGA, where a simplified integer based arithmetic of trigonometric transformation has been used to reduce hardware usages without accuracy loss; and
- All proposed ANPR stages have been successfully linked together and implemented on a single FPGA, where parallelism and pipelining technique have been exploited to achieve high running frequency and throughput rate. The achieved results have shown that the proposed system has the fastest processing speed when compared with existing ANPR systems.

8.2.3 Limitations

The objectives stated in Chapter 2 have been met and fully achieved. However, few of restrictions and limitations have been found during the development of this research project:

- In the case of the system accuracy, some of the failed cases are caused by incomplete NP and missed segmented characters, the NPL and CS stages of the

ANPR system can be improved further by introducing a NP checking module to help finding the correct NP or the character positions; and

- The proposed architectures have been implemented on the RC240 development board equipped with Virtex-4 FPGA. It is worth mentioning that the proposed designs can also be implemented on the latest FPGA platforms equipped with Virtex-6 or Virtex-7, which allows developing more advanced features and functions of ANPR system. For instance, the processing speed can be further improved using more external memory banks and on-chip resources. As far as the extra on-board resources are concerned, FPGA board equipped with video collection and communication units that will allow exploring new functions of ANPR system.

8.3 Future Work

The work undertaken in this research project has concentrated on development of novel/improved ANPR algorithms and their efficient architectures for the proposed ANPR algorithms and their implementations on FPGA. A set of objectives for the future included:

- *Further optimisation of the proposed ANPR algorithms:*

  There is always scope for more improvement in algorithms. In this case further improvement and optimisation can be done on each stage of ANPR. For example, in the NPL stage, the system can detect more than one NP at a time by using extra processing steps. In the CS stage, a NP checking stage can be developed to obtain more precise NP position. In OCR stage, different methodologies can be used for recognition, and see their impacts;

- *Further evaluation of the presented FPGA based ANPR system:*

  The proposed FPGA based ANPR system has been evaluated in each stage of

ANPR processing, however, it is necessary to use a large database to evaluate the entire FPGA based ANPR system, where different categories of the database can be used to evaluate the system performance under different environments (e.g. night and day time, distances and angles);

- *Developing efficient architectures specifically for the above optimised and HD ANPR systems:*

  The main challenge of developing efficient architectures for optimised and high definition ANPR systems is the hardware utilisation and throughput. For example, the optimised algorithm may need extra flow control step to find the best candidate of the detected NPs. In the case of HD ANPR system, because huge pixel data need to be processed, hardware utilisation and processing time can be significantly increased;

- *Further validation on a custom FPGA platform:*

  Evaluating the accuracy of the presented ANPR on a custom development board equipped with the latest FPGA (e.g. Virtex-7) that is also very important. In this case, the ANPR system can be tested on motorways or car park entrances; and

- *Investigating high definition ANPR system:*

  HD images provide a better image resolution with clear objects in the picture [140]. Not only a wide area can be covered by a HD camera (e.g. two to four lanes can be covered by a HD camera), but also an improvement of successful CS and OCR rates could be achieved in CS and OCR stages when using a HD image. Therefore, HD images allow achieving higher NP recognition success rate compared to standard definition images. However, in order to localise the HD NP image, the whole HD image is needed to be processed, which is computational intensive task.

Reference

[1] X. Jia, X. Wang, W. Li, and H. Wang, “A Novel Algorithm for Character Segmentation of Degraded License Plate Based on Prior Knowledge,” in *IEEE International Conference on Automation and Logistics*, 2007, pp. 249-253.

[2] A. A. Shah and L. J. Dal, “Intelligent Transportation Systems in Transitional and Developing Countries,” *IEEE Aerospace and Electronic Systems Magazine,* vol. 22, pp. 27-33, 2007.

[3] A. O. Yerdut, Y. B. Eldeniz, and H. G. Ilk, “Automatic license plate recognition based on a projection method,” in *IEEE 19th Conference on Signal Processing and Communications Applications (SIU)*, 2011, pp. 182-185.

[4] J. Zhang and J. Xu, “Research of overall program on highway toll collection system,” in *International Conference on Information Science and Technology*, 2011, pp. 1218-1221.

[5] C. N. E. Anagnostopoulos, I. E. Anagnostopoulos, I. D. Psoroulas, V. Loumos and E. Kayafas “License plate recognition from still images and video sequences: A survey," *IEEE Transaction Intelligent Transportation System,* vol. 9, pp. 377-391, 2008.

[6] *Product Brief: Intelligent Policing (IP) ANPR*. Available: http://www.ipl.com/papers/IP%20ANPR%20product%20brief.pdf (Acessed on Oct, 2012)

[7] R. Gurney, M. Rhead, S. Ramalingam, and N. Cohen, “Working towards an International ANPR Standard: an initial investigation into the UK Standard,” in *46th IEEE International Carnahan Conference on Security Technology*, USA, 2012, pp. 331-337.

[8] S. Connor, *Surveillance UK: why this revolution is only the start*. Available: http://www.independent.co.uk/news/science/surveillance-uk-why-this-revolution-is-only-the-start-520396.html (Acessed on Oct, 2012)

[9] Networkvideosystems, *Arecont ANPR IP cameras*. Available: http://networkvideosystems.co.uk/Arecont-ANPR-ip-cameras (Acessed on Oct, 2012)

[10] D. Shan, M. Ibrahim, M. Shehata, and W. Badawy, “Automatic License Plate Recognition (ALPR): A State-of-the-Art Review,” *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 23, pp. 311-325, 2013.

[11] C. Arth, F. Limberger, and H. Bischof, “Real-Time License Plate Recognition on an Embedded DSP-Platform,” presented at the IEEE Conference on Computer Vision and Pattern Recognition, 2007.

[12] C. Arth, C. Leistner and H.Bischof, “TRIcam: an embedded platform for remote traffic surveillance,” in *Proceedings of IEEE Computer Vision and Pattern Recognition Conference*, 2006, pp. 125-134.

[13] T. Kanamori, H. Amano, M. Arai, D. Konno, T. Nanba, and Y. Ajioka,

"Implementation and Evaluation of a High Speed License Plate Recognition System on an FPGA," in *7th International Conference on Computer and Information Technology*, 2007, pp. 567-572.

[14] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP Processor Fundamentals: Architectures and Features*, 1 ed.: IEEE Press Series on Signal Processing, 1997.

[15] J. Batlle, J. Martı, P. Ridao, and J. Amat, "A New FPGA/DSP-Based Parallel Architecture for Real-Time Image Processing," *Elsevier Real-time Imaging*, vol. 8, pp. 345-356, 2002.

[16] T. B. Welch, C. H. G. Wright, and M. G. Morrow, *Real-Time Digital Signal Processing from MATLAB to C With the TMS320C6x DSPs*: Taylor & Francis Group, LLC, 2011.

[17] F. Bensaali, "Accelerating Matrix Product on Reconfigurable Hardware for Image Processing Applications," PhD, School of Computer Science, The Queen's University of Belfast, 2005.

[18] A. Hayim, M. Knieser, and M. Rizkalla, "DSPs/FPGAs Comparative Study for Power Consumption, Noise Cancellation, and Real Time High Speed Applications," *Journal of Software Engineering and Applications*, vol. 3, pp. 391-403, 2010.

[19] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU Computing," *Proceedings of the IEEE*, vol. 96, pp. 879-899, 2008.

[20] S. Ehsan, A. F. Clark, and K. D. McDonald-Maier, "Hardware based Scale- and Rotation-Invariant Feature Extraction: Aretrospective Analysis and Future Directions," in *Second International Conference on Computer and Electrical Engineering*, 2009, pp. 620-624.

[21] K.-C. Wu and Y.-W. Tsai, "Structured ASIC, evolution or revolution?," in *international symposium on Physical design*, 2004, pp. 103-106

[22] I. Kuon, R. Tessier, and J. Rose, "FPGA Architecture: Survey and Challenges," *Foundations and Trends® in Electronic Design Automation*, vol. 2, pp. 135-253, 2008.

[23] A. Ret, "Xilinx," *Fortune*, 1990, pp.81

[24] B. Zahiri, "Structured ASICs: Opportunities and Challenges," in *International Conference on Computer Design*, 2003, pp. 404-409.

[25] P. H. W. Leong, "Recent Trends in FPGA Architectures and Applications," presented at the 4th IEEE International Symposium on Electronic Design, Test & Applications, 2008.

[26] Xilinx, Inc., *7 Series FPGAs Overview*. Available: http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf (Acessed on May, 2013)

[27] Xilinx, Inc. *Xilinx Next Generation 28 nm FPGA Technology Overview*. Available: http://www.xilinx.com/support/documentation/white_papers/wp312_Next_Gen_28_nm_Overview.pdf (Acessed on May, 2013)

[28] Xilinx, Inc., *Zynq-7000 All Programmable SoC*. Available: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/index.htm

(Acessed on May, 2013)

[29] J. imenez, I. Urriza, L. A. Barragan, D. Navarro, J. I. Artigas, and O. Lucia, “Hardware-in-the-loop simulation of FPGA embedded processor based controls for power electronics,” 2011.

[30] I. Kuon, R. Tessier, and J. Rose, “FPGA Architecture: Survey and Challenges,” *Electronic Design Automation*, vol. 2, pp. 135-253, 2007.

[31] Xilinx, Inc. *Virtex-4 User Guide*. Available: www.xilinx.com (Acessed on Jan, 2012)

[32] K. Comption and S. Hauck, “Reconfigurable Computing: A Survey of Systems and Software,” *ACM Computing Surveys*, vol. 34, pp. 171-210, 2002.

[33] Xilinx, Inc. *Xilinx ISE*. Available: www.xilinx.com (Acessed on Jan, 2012)

[34] Altera Corporation. *Quartus II*. Available: www.altera.com (Acessed on Jan, 2012)

[35] IEEE, “IEEE Standard VHDL Language Reference Manual,” ed: The Institute of Electrical and Electronics Engineers, Inc, 2008.

[36] IEEE, “IEEE Standard Verilog Hardware Description Language,” ed: The Institute of Electrical and Electronics Engineers, Inc., 2001.

[37] Mentor Graphics Corporation. *Handel-C User Manual*. Available: http://www.mentor.com/ (Acessed on Jan, 2012)

[38] F. Bensaali, A. Amira, and A. Bouridane, “Accelerating matrix product on reconfigurable hardware for image processing applications,” *IET Circuits, Devices & Systems*, vol. 152, pp. 236-246, 2005.

[39] S. M. Loo, B. E. Wells, N. Freije, and J. Kulick, “Handel-C for Rapid Prototyping of VLSI Coprocessors for Real Time Systems,” in *34th Shoutheastern Symposium on System Theory*, 2002, pp. 6-10.

[40] P. Voles, L. Holasek, and M. Vasilko, “ANSI C and Handel-C Based Rapid Prototyping Framework for Real-Time Image Processing Algorithms,” in *International Conferene on Engineering of Reconfigurable Systems and Algorithms*, 2002.

[41] J. D. Crawford, “EDIF: A Mechanism for the Exchange of Design Information,” *IEEE Design & Test of Computers*, vol. 2, pp. 63-69, 1985.

[42] J.-M. Guo and Y.-F. Liu, “License Plate Localization and Character Segmentation With Feedback Self-Learning and Hybrid Binarization Techniques,” *IEEE Transactions on Vehicular Technology*, vol. 57, pp. 1417-1424, 2008.

[43] Y.-P. Huang, C.-H. Chen, Y.-T. Chang, and F. E. Sandnes, “An intelligent strategy for checking the annual inspection status of motorcycles based on license plate recognition,” *Expert Systems with Applications*, vol. 36, pp. 9260-9267, 2009.

[44] Y.-R. Wang, W.-H. Lin, and S.-J. Horng, “A sliding window technique for efficient license plate localization based on discrete wavelet transform,” *Expert Systems with Applications*, vol. 38, pp. 3142-3146, 2011.

[45] Y. Wen, Y. Lu, J. Yan, Z. Zhou, von Deneen K.M. and P. Shi, “An Algorithm for License Plate Recognition Applied to Intelligent Transportation System,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, pp. 830-845, 2011.

[46] M. Vargas, S. L.Toral, F. Barrero, and F. Cortés, "A License Plate Extraction Algorithm Based on Edge Statistics and Region Growing," *Lecture Notes in Computer Science, Springer,* vol. 5716, pp. 317-326, 2009.

[47] H. Bai and C. Liu, "A hybrid license plate extraction method based on edge statistics and morphology," in *17th International Conference on Pattern Recognition*, 2004, pp. 831-834.

[48] J. Jiao, Q. Ye, and Q. Huang, "A configurablemethod formulti-style license plate recognition," *Pattern Recognition Letters,* vol. 42, pp. 358-369, 2009.

[49] N. Thome, A. Vacavant, L. Robinault, and S. Miguet, "A cognitive and video-based approach for multinational License Plate Recognition," *Machine Vision and Applications,* vol. 22, pp. 389-407, 2011.

[50] H. Xiangjian, Z. Lihong, W. Qiang, J. Wenjing, B. Samali, and M. Palaniswami, "Segmentation of characters on car license plates," in *IEEE 10th Workshop on Multimedia Signal Processing*, 2008, pp. 399-402.

[51] C. Anagnostopoulos, T. Alexandropoulos, V. Loumos, and E. Kayafas, "Intelligent traffic management through MPEG-7 vehicle flow surveillance," in *IEEE International Symposium on Modern Computing,* 2006, pp. 202-207.

[52] H. Samet and M. Tamminen, "Efficient Component Labeling of Images of Arbitrary Dimension Represented by Linear Bintrees," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 10, pp. 579-586, 1988.

[53] P. Wu, H.-H. Chen, R.-J. Wu, and D.-F. Shen, "License plate extraction in low resolution video," in *18th International Conference on Pattern Recognition*, 2006, pp. 824-827.

[54] P. Tarabek, "Fast license plate detection based on edge density and integral edge image," in *IEEE 10th International Symposium on Applied Machine Intelligence and Informatics (SAMI),* 2012, pp. 37-40.

[55] X. Shi, W. Zhao, and Y. Shen, "Automatic License Plate Recognition System Based on Color Image Processing," in *Computational Science and Its Applications*. vol. 3483, O. Gervasi, M. Gavrilova, V. Kumar, A. Laganá, H. Lee, Y. Mun, *et al.*, Eds., ed: Springer Berlin / Heidelberg, 2005, pp. 307-314.

[56] S.L. Chang, L.S. Chen, Y.C. Chung and S.W. Chen, "Automatic license plate recognition," *IEEE Transactions on Intelligent Transportation Systems,* vol. 5, pp. 42-53, 2004.

[57] J. A. G. Nijhuis, M. H. Ter Brugge, K. A. Helmholt, J. P. W. Pluim, L. Spaanenburg, R. S. Venema, *et al.*, "Car license plate recognition with neural networks and fuzzy logic," in *IEEE International Conference on Neural Networks*, 1995, pp. 2232-2236.

[58] N. Zimic, J. Ficzko, M. Mraz, and J. Virant, "The fuzzy logic approach to the car number plate locating problem," in *Intelligent Information Systems*, 1997, pp. 227-230.

[59] S. Chang, Chen, L., Chung, Y. and Chen, S., "Automatic license plate recognition," *IEEE Transaction on Intelligent Transpotation Systerms,* vol. 5, pp. 42-53, 2004.

[60] F. Wang, L. Man, B. Wang, Y. Xiao, W. Pan, and X. Lu, "Fuzzy-based algorithm

for color recognition of license plates," *Journal of Pattern Recognition Letters*, vol. 29, pp. 1007-1020, 2008.

[61] K. Kim, K. Jung, and J. Kim, "Color Texture-Based Object Detection: An Application to License Plate Localization," in *Pattern Recognition with Support Vector Machines*. vol. 2388, S.-W. Lee and A. Verri, Eds., ed: Springer Berlin / Heidelberg, 2002, pp. 321-335.

[62] F. Kahraman, B. Kurt, and M. Gökmen, "License Plate Character Segmentation Based on the Gabor Transform and Vector Quantization," in *Computer and Information Sciences* vol. 2869, A. Yazici and C. Sener, Eds., ed: Springer Berlin / Heidelberg, 2003, pp. 381-388.

[63] T. D. Duan, T. L. H. Du, T. V. Phuoc, and N. V. Hoang, "Building an Automatic Vehicle License-Plate Recognition System," in *International Conference in Computer Science*, 2005, pp. 59-63.

[64] C. R. Jung and R. Schramm, "Rectangle detection based on a windowed Hough transform," in *Computer Graphics and Image Processing, 2004. Proceedings. 17th Brazilian Symposium on*, 2004, pp. 113-120.

[65] Y. Cheng, J. Lu, and T. Yahagi, "Car license plate recognition based on the combination of principal components analysis and radial basis function networks," in *Signal Processing, 2004. Proceedings. ICSP '04. 2004 7th International Conference on*, 2004, pp. 1455-1458 vol.2.

[66] M. Rouhani, "A Fuzzy Feature Extractor Neural Network and its Application in License Plate Recognition," in *Computational Intelligence, Theory and Applications*, B. Reusch, Ed., ed: Springer Berlin Heidelberg, 2006, pp. 223-228.

[67] H. Ching-Tang, J. Yu-Shan, and H. Kuo-Ming, "Multiple license plate detection for complex background," in *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, 2005, pp. 389-392 vol.2.

[68] C. N. E. Anagnostopoulos, I. E. Anagnostopoulos, V. Loumos, and E. Kayafas, "A License Plate-Recognition Algorithm for Intelligent Transportation System Applications," *IEEE Transactions on Intelligent Transportation Systems, ,* vol. 7, pp. 377-392, 2006.

[69] T. Chew Lim, H. Weihua, Y. Zhaohui, and X. Yi, "Imaged document text retrieval without OCR," *IEEE Transactions on Pattern Analysis and Machine Intelligence* vol. 24, pp. 838-844, 2002.

[70] J. Min-Chul, S. Yong-Chul, and S. N. Srihari, "Machine printed character segmentation method using side profiles," in *IEEE International Conference on Systems, Man, and Cybernetics* 1999, pp. 863-867.

[71] Y. Zhang and C. Zhang, "A new algorithm for character segmentation of license plate," in *IEEE Intelligent Vehicles Symposium*, 2003, pp. 106 - 109.

[72] W. Tsang-Hong, N. Feng-Chou, L. Keh-Tsong, and C. Yon-Ping, "Robust license plate recognition based on dynamic projection warping," in *IEEE International Conference on Networking, Sensing and Control*, 2004, pp. 784-788.

[73] H. Al-Yousefi and S. S. Udpa, "Recognition of Arabic characters," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 14, pp. 853-857,

1992.

[74] M. F., G. M., and A. J., “New methods for automatic reading of VLP's (Vehicle License Plates),” in *International conference on signal processing, pattern recognition and applications*, 2002.

[75] L. Gang, Z. Ruili, and L. Ling, “Research on Vehicle License Plate Location Based on Neural Networks,” in *First International Conference on Innovative Computing, Information and Control*, 2006, pp. 174-177.

[76] S. Zhang, M. Zhang, and X. Ye, “Car plate character extraction under complicated environment,” in *IEEE International Conference on Systems, Man and Cybernetics*, 2004, pp. 4722-4726.

[77] H. Mahini, S. Kasaei, and F. Dorri, “An Efficient Features - Based License Plate Localization Method,” in *18th International Conference on Pattern Recognition*, 2006, pp. 841-844.

[78] V. Shapiro and G. Gluhchev, “Multinational license plate recognition system: segmentation and classification,” in *17th International Conference on Pattern Recognition*, 2004, pp. 352-355.

[79] Y. Youngwoo, B. Kyu-Dae, Y. Hosub, and K. Jaehong, “Blob extraction based character segmentation method for automatic license plate recognition system,” in *IEEE International Conference on Systems, Man, and Cybernetics*, 2011, pp. 2192-2196.

[80] S. Nomura, K. Yamanaka, O. Katai, H. Kawakami, and T. Shiose, “A novel adaptive morphological approach for degraded character image segmentation,” *Pattern Recognition*, vol. 38, pp. 1961-1975, 11// 2005.

[81] N. S., Y. K., and K. O., “A new method for degraded color image binarization based on adaptive lightning on grayscale versions,” *IEICE Transaction Information System*, vol. E87-D, pp. 1012–1020 2004.

[82] K.-B. Kim, S.-W. Jang, and C.-K. Kim, “Recognition of Car License Plate by Using Dynamical Thresholding Method and Enhanced Neural Networks,” in *Computer Analysis of Images and Patterns*. vol. 2756, N. Petkov and M. Westenberg, Eds., ed: Springer Berlin / Heidelberg, 2003, pp. 309-319.

[83] W. K. PRATT, *Digital Image Processing: PIKS Inside*, Third ed.: John Wiley and Sons, Inc., New York, 2001.

[84] A. Capar and M. Gokmen, “Concurrent Segmentation and Recognition with Shape-Driven Fast Marching Methods,” in *18th International Conference on Pattern Recognition*, 2006, pp. 155-158.

[85] P. Stec and M. Domanski, “Efficient unassisted video segmentation using enhanced fast marching,” in *International Conference on Image Processing*, 2003, pp. 427-430.

[86] Y. Cui and Q. Huang, “Extracting characters of license plates from video sequences,” *Machine Vision and Applications*, vol. 10, pp. 308-320, 1998.

[87] M. I. Schlesinger and V. Hlavác, *Ten Lectures on Statistical and Structural Pattern Recognition* vol. 24: Springer, 2002.

[88] V. Franc and V. Hlaváč, “License Plate Character Segmentation Using Hidden

Markov Chains " in *Pattern Recognition*. vol. 3663, W. Kropatsch, R. Sablatnig, and A. Hanbury, Eds., ed: Springer Berlin / Heidelberg, 2005, pp. 385-392.

[89] S. Mori, H. Nishida and Yamada H., *Optical Character Recognition*: John Wiley & Sons, Inc. NY, USA, 1999.

[90] N. Mani, and B. Srinivasan, "Application of articial neural network model for optical character recognition," presented at the IEEE International Conference on Systems, Man, and Cybernetics, 1997.

[91] K. K. Kim, K. I. Kim, J. B. Kim, and H. J. Kim, "Learning-based approach for license plate recognition," in *IEEE Signal Processing Society Workshop, Neural networks for Signal Processing*, 2000, pp. 614-623.

[92] X. Pan, X. Ye and S. Zhang "A hybrid method for robust car plate character recognition " presented at the IEEE International Conference on Systems, Man and Cybernetics, 2004.

[93] L. Xu, A. Krzyzak, and C. Y. Suen, "Methods of combining multiple classifiers and their application to handwriting recognition," *IEEE Transaction on Systerm, Man, Cybernetics,* vol. 22, pp. 418-435, 1992.

[94] Y. Amit, D. Geman, and X. Fan, "A coarse-to-fine strategy for multiclass shape detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 26, pp. 1606-1621, 2004.

[95] S. Draghici, "A Neural Network Based Artificial Vision System for Licence Plate Recognition," *International Journal of Neural Systems,* vol. 08, pp. 113-126, 1997.

[96] C. Oz, and F. Ercal, "A Practical License Plate Recognition System for Real-Time Environments," *Computational Intelligence and Bioinspired Systems,* vol. 3512/2005, pp. 497-538, 2005.

[97] M. Rasooli, S. Ghofrani and E. Fatemizadeh, "Farsi License Plate Detection based on Element Analysis and Characters Recognition," *International Journal of Signal Processing, Image Processing and Pattern Recognition,* vol. 4, pp. 65-80, 2011.

[98] M. Raus, and L. Kreft, "Reading car license plates by the use of artificial neural networks," in *the 38th Midwest Symposium on Circuits and Systems,* 1995, pp. 538-541.

[99] Y. Hu, F. Zhu, and X. Zhang, "A Novel Approach for License Plate Recognition Using Subspace Projection and Probabilistic Neural Network," *Lecture Notes in Computer Science*, vol. 3497, pp. 821-827, 2005.

[100] R. Salakhutdinov and H. Larochelle, "Efficient Learning of Deep Boltzmann Machines," in *the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)* 2010.

[101] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation,* vol. 18, pp. 1527-1554, 2006.

[102] P. Comelli, P. Ferragina,M. N. Granieri, and F. Stabile, "Optical recognition of motor vehicle license plates," *IEEE Transaction on Vehicular Technology* vol. 44, pp. 790-799, 1995.

[103] Y. Huang, S. Lai, and W. Chuang, "A Template-Based Model for License Plate

Recognition," in *IEEE International Conference on Networking, Sensing & Control*, Taipei, 2004, pp. 737-742.

[104] P. Viola and M. Jones, "Robust Real-Time Face Detection," *International Journal of Computer Vision*, vol. 57, pp. 137-154, 2004/05/01 2004.

[105] H. Caner, H. S. Gecim, and A. Z. Alkar, "Efficient Embedded Neural-Network-Based License Plate Recognition System," *IEEE Transactions on Vehicular Technology*, vol. 57, pp. 2675-2683, 2008.

[106] Y. Osana, T. Fukushima, and H. Amano, "Implementation of ReCSiP: A ReConfigurable Cell Simulation Platform," in *Field Programmable Logic and Application*. vol. 2778, P. Y. K. Cheung and G. Constantinides, Eds., ed: Springer Berlin / Heidelberg, 2003, pp. 766-775.

[107] N. Bellas, S. M. Chai, M. Dwyer, and D. Linzmeier, "FPGA implementation of a license plate recognition SoC using automatically generated streaming accelerators," in *20th International Parallel and Distributed Processing Symposium*, 2006, pp. 8-16.

[108] MediaLab-NTUA. *MediaLab LPR Database*. Available: http://www.medialab.ntua.gr/research/LPRdatabase.html (Acessed on Sep. 2011)

[109] *CitySync Limited*. Available: http://www.citysync.co.uk (Acessed on Sep. 2011)

[110] F. Y. Shih and Y.-T. Wu, "Decomposition of arbitrary gray-scale morphological structuring elements," *Pattern Recognition*, vol. 38, pp. 2323-2332, 12// 2005.

[111] M. Grundland and N. A. Dodgson, "Decolorize: Fast, contrast enhancing, color to grayscale conversion," *Pattern Recognition*, vol. 40, pp. 2891-2896, 11// 2007.

[112] Mentor Graphics Corporation. *PAL User Manual*. Available: http://www.mentor.com/ (Acessed on Jun, 2011)

[113] Mentor Graphics Corporation. *RC240 Datasheet*. Available: http://www.mentor.com/ (Acessed on Jun, 2011)

[114] Mentor Graphics Corporation. *PixelStreams User Manual*. Available: http://www.mentor.com/ (Acessed on Jun, 2011)

[115] Xilinx, Inc., *Xpower Tutorial: FPGA Design*. Available: http://www.xilinx.com/ (Acessed on Jun, 2011)

[116] Driver, and Vehicle Licensing Agency. *Display of Registration Marks for Motor Vehicles*. Available: www.direct.gov.uk/motoring (Acessed on Jun, 2011)

[117] X. Zhai, F. Bensaali, and S. Ramalingam, "Real-Time License Plate Localisation on FPGA," in *17th IEEE Workshop on Embedded Computer Vision and Pattern Recognition*, 2011, pp. 14-19.

[118] A. R. Omondi and J. C. Rajapakse, *FPGA Implementations of Neural Networks* vol. XII, 2006.

[119] H. Demuth, M. Beale and M. Hagan, *Neural Network Toolbox 6 User's Guide*: The MathWorks, Inc. , 2008.

[120] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, pp. 525-533, 1993.

[121] E. Alpaydın, *Introduction to Machine Learning*, Second ed.: Massachusetts Institute of Technology, 2010.

[122] D. Nguyen and B. Widrow, “Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights,” in *International Joint Conference on Neural Networks*, 1990, pp. 21-26.

[123] X. Zhai and F. Bensaali, “Improved Number Plate Character Segmentation Algorithm and its Efficient FPGA Implementation,” *Journal of Real-Time Image Processing*, 2012.

[124] P. Y. Simard, D. Steinkraus, and J. C. Platt, “Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis,” in *the Seventh International Conference on Document Analysis and Recognition (ICDAR 2003)*, 2003, pp. 958-962.

[125] D. Decoste and B. chölkopf, “Training Invariant Support Vector Machines,” *Machine Learning*, vol. 46, pp. 161-190, 2002.

[126] F. Bensaali, A. Amira, and R. Sotudeh, “Floating-Point Matrix Product on FPGA,” in *IEEE/ACS International Conference on Computer Systems and Applications*, 2007, pp. 466-473.

[127] F. Bensaali and A. Amira, “An FPGA Based Parallel Matrix Multiplier for 3D Affine Transformations,” *IET Vision, Image and Signal Processing Special Issue on Rapid Prototyping of Signal Processing Algorithms*, vol. 153, pp. 739-746, 2006.

[128] D. Zheng, Y. Zhao, and J. Wang, “An efficient method of license plate location,” *Pattern Recognition Letters*, vol. 26, pp. 2431-2438, 2005.

[129] B. R. Lee, K. Park, H. Kang, H. Kim, and C. Kim, “Adaptive Local Binarization Method for Recognition of Vehicle License Plates,” in *Combinatorial Image Analysis*. vol. 3322, J. Žunic, Ed., ed: Springer Berlin / Heidelberg, 2004, pp. 646-655.

[130] W. Jia, H. Zhang, and X. He, “Region-based license plate detection,” *Journal of Network and Computer Applications*, vol. 30, pp. 1324-1333, 2006.

[131] M.-S. Pan, J.-B. Yan, and Z.-H. Xiao, “Vehicle License Plate Character Segmentation,” *International Journal of Automation and Computing*, vol. 05, pp. 425-432, 2008.

[132] X. Zhai, F. Bensaali, and R. Sotudeh, “OCR-Based Neural Network for ANPR,” in *IEEE International Conference on Imaging Systems and Techniques*, Manchester, UK, 2012, pp. 393-397.

[133] N. Otsu, “A Tlreshold Selection Method from Gray-Level Histograms,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, pp. 62-66, 1979.

[134] F. Yang, Z. Ma, and M. Xie, “A Novel Binarization Approach for License Plate,” in *2006 1ST IEEE Industrial Electronics and Applications*, 2006, pp. 1-4.

[135] X. Zhai, F. Bensaali, and S. Ramalingam, “Improved Number Plate Localisation Algorithm and its Efficient FPGA Implementation,” *IET Circuits, Devices & Systems, vol. 7, issue 2*, 2013.

[136] H. Goldstein, *Classical Mechanics*, 2nd ed.: Addison-Wesley, 1980.

[137] J. Cohen, P. Cohen, S. G. West, and L. S. Aiken, *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*: Psychology Press,

2002.

[138] X. Zhai, F. Bensaali and S. Ramalingam, “License plate localisation based on morphological operations,” in *11th Int. Conf. Control Automation Robotics & Vision*, 2010, pp. 1128-1132.

[139] X. Zhai, F. Bensaali, and R. Sotudeh, “FPGA-based Number Plate Binarisation and Adjustment for ANPR Systems,” *Journal of Electronic Imaging,* pp. 1-11, 2013.

[140] S. Hao, L. Chao, W. Qi, and X. Zhang, “Real-Time Anti-Interference Location of Vehicle License Plates Using High-Definition Video,” *IEEE Intelligent Transportation Systems Magazine,* vol. 1, pp. 17-23, 2009.

[141] I. Giannoukos, C.-N. Anagnostopoulos, V. Loumos, and E. Kayafas, “Operator context scanning to support high segmentation rates for real time license plate recognition,” *Pattern Recognition,* vol. 43, pp. 3866-3878, 2010.

[142] Xilinx, inc., *Virtex-4 Family Overview*. Available: www.xilinx.com (Acessed on Jun, 2011)

[143] Mentor Graphics Corporation. *RC Host Library and FTU3 User Manual*. Available: www.mentor.com (Acessed on Jun, 2011)

[144] Mentor Graphics Corporation. *DK User Manual*. Available: www.mentor.com (Acessed on Jun, 2011)

[145] Mentor Graphics Corporation. *Fixed-point Library*. Available: www.mentor.com (Acessed on Jun, 2011)

[146] Mentor Graphics Corporation. *Pipelined Floating-point Library*. Available: www.mentor.com (Acessed on Jun, 2011)

[147] Xilinx, Inc., *Xilinx Timing Analyser User's Guide*. Available: www.xilinx.com (Acessed on Jun, 2011)

Appendix A: RC240 Prototyping Platform

In this research project, the RC240 hardware platform was used to prototype the proposed designs. The RC240 board is equipped with Xilinx Virtex-4 LX40 FF1148-10 FPGA and is packaged with a set of support libraries including the Platform Abstraction Layer (PAL) and PixelStreams image and video processing library [113]. It mainly has three banks of 1MB×36-bit pipelined SRAM directly connected to the FPGA for data processing operations, a USB device ports for data communication with host PC, and two user programmable clocks. All three memory banks are accessible by the FPGA and host PC. The high-speed USB 2.0 interface allows high data rate communication between host PC and FPGA applications on the board. Two programmable clocks are programmed by the host PC, and have frequency range of 2 MHz to 300 MHz. Figure A-1 shows the RC240 overview.

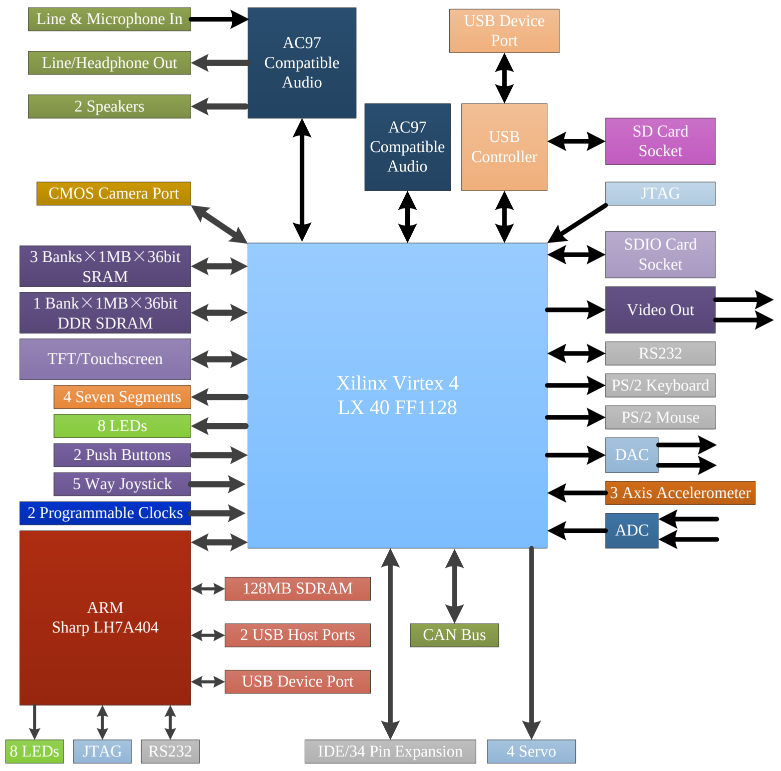


Figure A-1: The RC 240 overview [103]

A.1 Virtex-4 FPGA

The Xilinx Virtex-4 family greatly enhances programmable logic design capabilities, and make it a powerful alternative to ASIC technology [142]. Virtex-4 FPGA family consists of three platform sets LX, FX, and SX, which offering multiple feature choices and combinations to address all complex applications. In the FPGA, there are dedicated DSP slices, high-speed clock management circuitry, and source-synchronous interface blocks. Virtex-4 devices are produced on a 90 *nm* copper process using 300 *mm* wafer technology. A summary of the Virtex-4 family main features are listed as follows:

- XtremeDSP Slice:
    - 18×18, two's complement, signed Multiplier;
    - Optional pipeline stages; and
    - Built-in accumulator (48-bit) and Adder/Subtractor.
- Smart RAM Memory Hierarchy
    - Distributed RAM;
    - Dual-port 18-Kbit RAM blocks; and
    - High-speed memory interface supports DDR and DDR2 SDRAM, QDR-II, and RLDRAM-II.
- Flexible Logic Resource:
- Secure Chip AES Bitstream Encryption
- 90 *nm* Copper CMOS Process
- 1.2V core Voltage
- Flip-Chip Packaging including Pb-Free Package Choices

The main available resources of the used Virtex-4 FPGA in this research project are listed in Table A-1.

Table A-1: Virtex-4 XC4VLX40 on-chip resources

Device	Configurable Logic Blocks (CLBs)				XtremeDSP Slices	Block RAM	
	Array Size	Logic Cells	Slices	Max Distributed RAM (Kb)		18Kb Blocks	Max Block RAM (Kb)
XC4VLX40	128×36	41,472	18,432	288	64	96	1,728

A.2 Host-FPGA Communication

The RC240 board is supported with a macro library (the RC host library) that simplifies the process of initialising and communicating to the hardware [143]. The library provides the following functionalities:

- Initialisation and configuration a board;
- USB data transfer between PC and the RC240 board;
- Set on board clock rate;
- Access the external memory on the RC240 board; and
- Error checking and debugging are included in a C or C++ program that runs on the host PC and performs data transfer.

In this research project, the USB data transfer function is used to communicate between the Host and RC240 board. It allows transfer one byte data a time between the host and FPGA. A USB data control module in the FPGA is used to control the data flow, and access the external memory. The overview of the Host-FPGA communication system is shown in Figure A-2.

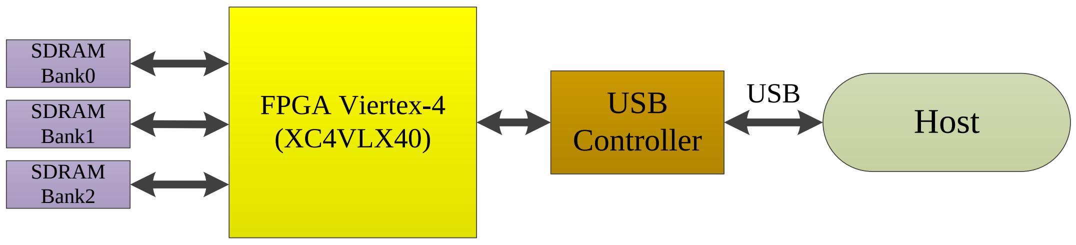


Figure A-2: The overview of the Host-FPGA communication system

A.2.1 Example Functions from RC Host Library

The RC Host Library allows you to communicate with a Mentor Graphic RC board from a host computer via the USB interface [143]. In this section, some of important functions from the RC Host Library are introduced.

Opening and Closing Boards [143]

```
typedef RCBoard;
RCStatus RCBoardOpen (int BoardNum, RCBoard *BoardPtr);
RCStatus RCBoardClose (RCBoard Board);
```

Description:

The functions **RCBoardOpen()** and **RCBoardClose()** are used to open and close the used **RCBoard** respectively, where **BoardNum** and **BoardPtr** indicate which board is attached and its pointer to variable of type **RCBoard** respectively.

Communicating over USB **[143]**

```
RCStatus RCUSBWrite (RCBoard Board, int Bytes, const char *Buffer, int
*BytesWritten);
RCStatus RCUSBRead (RCBoard Board, int Bytes, char *Buffer, int *BytesRead);
```

Description:

The functions **RCUSBWrite()** and **RCUSBRead()** are used to write a byte of data to an application running in the **RCBoard** and read a byte of data from the **RCBoard** to the host PC respectively, where ***BytesWritten** and ***BytesRead** indicate the number of bytes are needed for the writing and reading respectively. In order to use the two functions, the following two functions also need to be used in the application running in the FPGA:

```
macro proc PalDataPortRun (HandleCT, ClockRate);
macro proc PalDataPortRead (Handle, DataPtr);
macro proc PalDataPortWrite (Handle, Data);
```

PalDataPortRun() is used to initialise the USB port, **PalDataPortRead()** is used to read a byte of data from the USB port to the FPGA, and store it in the register **DataPtr,**

PalDataPortWrite() is used to write a byte of data from the FPGA to the USB port.

A.2.2 Accessing the External RAMs

In this research work, many parts of algorithms need to access the external RAMs, for example, car images and weights of NN are stored in the external RAMs. The RC240 board supports PL2 RAMs which can be read from or written to in exactly one clock cycle, but the address supplied two clock cycles earlier. The following API functions are used to access the RAMs.

```
macro proc PalPL2Run (HandleCT, ClockRate);
macro proc PalPL2RAMSetReadAddress (Handle, Address);
macro proc PalPL2RAMSetWriteAddress (Handle, Address);
macro proc PalPL1RAMRead (Handle, DataPtr);
macro proc PalPL1RAMWrite (Handle, Data);
```

Description:

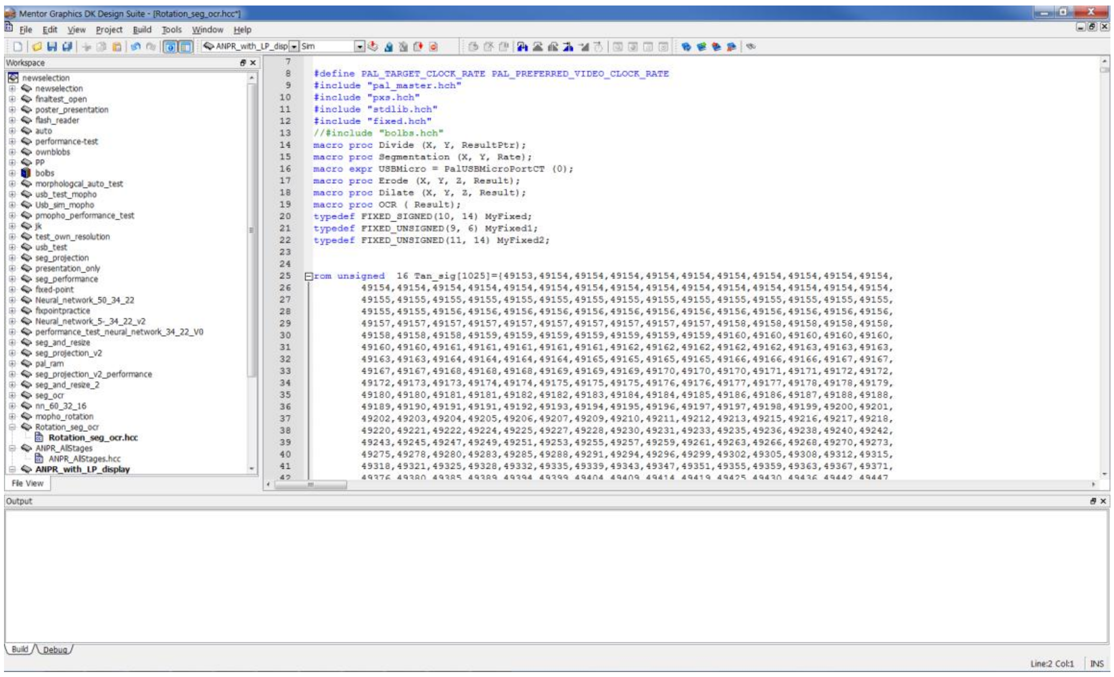
PalPL2Run() must be used in parallel with the rest of functions in the program, which indicates which memory bank and clock frequency will be used. **PalPL2RAMSetReadAddress()** and **PalPL2RAMSetWriteAddress()** are used to set the reading and writing addresses respectively, the set addresses for reading and writing that will occur two clock cycles later after use the functions. **PalPL2RAMRead()** and **PalPL2RAMWrite()** read or write a single item of data from or to the address in the RAM set two clock cycles earlier respectively.

Appendix B: Tools and Software Packages

In this research project, DK design suite [144] and Xilinx ISE [33] were used to program and evaluate the proposed designs on FPGAs. Details about these two tools are given in the following sections.

B.1 DK Design Suite

DK design suite provides a development environment to design and compile hardware circuits using Handel-C. The environment includes an Integrated Development Environment (IDE) along with code and macro libraries. Circuit development takes place within the IDE, and you can configure builds for debug, simulation or hardware. Figure B-1 shows the design environment of DK.



```

#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#include "pal_master.hch"
#include "pxs.hch"
#include "stdlib.hch"
#include "fixed.hch"
//#include "bolbs.hch"
macro proc Divide (X, Y, ResultPtr);
macro proc Segmentation (X, Y, Rate);
macro expr USBMicro = PalUSBMicroPortCT (0);
macro proc Erode (X, Y, Z, Result);
macro proc Dilate (X, Y, Z, Result);
macro proc OCR ( Result);
typedef FIXED_SIGNED(10, 14) MyFixed;
typedef FIXED_UNSIGNED(9, 6) MyFixed1;
typedef FIXED_UNSIGNED(11, 14) MyFixed2;


rom unsigned  16 Tan_sig[1025]={49153,49154,49154,49154,49154,49154,49154,49154,49154,49154,49154,
          49154,49154,49154,49154,49154,49154,49154,49154,49154,49154,49154,49154,49154,49154,49154,
          49155,49155,49155,49155,49155,49155,49155,49155,49155,49155,49155,49155,49155,49155,49155,
          49155,49155,49156,49156,49156,49156,49156,49156,49156,49156,49156,49156,49156,49156,49156,
          49157,49157,49157,49157,49157,49157,49157,49157,49157,49157,49158,49158,49158,49158,49158,
          49158,49158,49158,49159,49159,49159,49159,49159,49159,49159,49160,49160,49160,49160,49160,
          49160,49160,49161,49161,49161,49161,49161,49162,49162,49162,49162,49162,49163,49163,49163,
          49163,49163,49164,49164,49164,49164,49165,49165,49165,49165,49166,49166,49166,49167,49167,
          49167,49167,49168,49168,49168,49169,49169,49169,49170,49170,49170,49171,49171,49172,49172,
          49172,49173,49173,49174,49174,49175,49175,49175,49176,49176,49177,49177,49178,49178,49179,
          49180,49180,49181,49181,49182,49182,49183,49184,49184,49185,49186,49186,49187,49188,49188,
          49189,49190,49191,49191,49192,49193,49194,49195,49196,49197,49197,49198,49199,49200,49201,
          49202,49203,49204,49205,49206,49207,49209,49210,49211,49212,49213,49215,49216,49217,49218,
          49220,49221,49222,49224,49225,49227,49228,49230,49231,49233,49235,49236,49238,49240,49242,
          49243,49245,49247,49249,49251,49253,49255,49257,49259,49261,49263,49266,49268,49270,49273,
          49275,49278,49280,49283,49285,49288,49291,49294,49296,49299,49302,49305,49308,49312,49315,
          49318,49321,49325,49328,49332,49335,49339,49343,49347,49351,49355,49359,49363,49367,49371,
          49376,49380,49385,49389,49394,49399,49404,49409,49414,49419,49425,49430,49436,49442,49447
```

Figure B-1: The DK design synthesis tool

DK produces a Netlist file, which is used during the Place and route stage (PAR) to generate the bitstream file. This process is shown in Figure B-2.

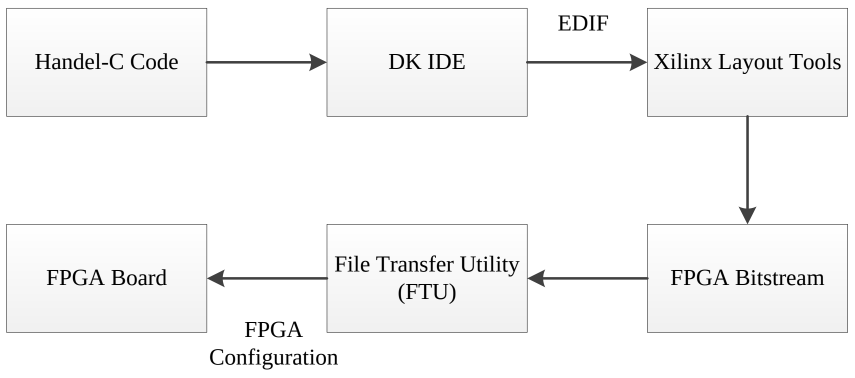


Figure B-2: DK design flow

B.1.1 Handel-C

Handel-C is a high level programming language which targets low-level hardware, most commonly used in the programming of FPGAs. It is a rich subset of C, with non-standard extensions to control hardware configuration with an emphasis on parallelism. Unlike other C to hardware tools which rely on going via several intermediate stages, Handel-C allows hardware to be directly targeted from software.

Parallel Hardware Generation

Handel-C has additional constructs to support the parallelisation of code using the *par* statement, which means any instructions inside the *par* statement that will be execute in parallel at exactly the same instant in time by two separated pieces of hardware. In addition to this feature, Handel-C also provides construction to support the sequential coding using *seq*, which means any instructions inside the *seq* statement that will be executed sequentially. In Figure B-3 shows two examples when using *par* and *seq* respectively [134].

```
Sequential Block
seq
{
        a=1;
        b=2;
        c=a+b;
}
// c=3 and 3 Clock Cycles
```

```
Parallel Block
par
{
        a=1;
        b=2;
        c=a+b;
}
// c=0 and 1 Clock Cycles
```

Figure B-3: The *seq* and *par* constructs

Variables

One basic variable type is integer in Handel-C, which can be signed or unsigned with any width and mapped to hardware registers [37].

DK also provides two platform-independent libraries for other types manipulation: Fixed-point and pipelined floating-point [145] [146]. The fixed-point library allows defining different widths of the fractional and integer parts of the number and provides macros to perform arithmetic operations. The pipelined floating-point library allows the floating-point operations to be performed in a pipelined manner on floating-point numbers.

In this research project, the fixed-point library was used for all fixed-point arithmetic operations.

Memory

Handel-C provides two keywords *ram* and *rom* to implement RAMs and ROMS respectively. There are mainly two different types of RAM used in this research project:

- **Distributed RAM**: it is implemented in look-up tables in the logic blocks of the FPGA.
- **Block RAM**: it is available on certain chips and has high-capacity but limited numbers can be used.

Additionally, both types of RAM can be defined as multiple-ported RAMs (MPRAMs) using the *mpram* keyword.

B.1.2 Platform Abstraction Layer

The Platform Abstraction Layer (PAL) is a component of the DK design suite, which is an Application Programming Interface (API) for peripherals. The API offers a standard interface to hardware, enabling portable Handel-C applications that can run on different FPGA/PLD RC boards without modification [112]. Figure A-4 shows the examples of PAL supported peripherals.

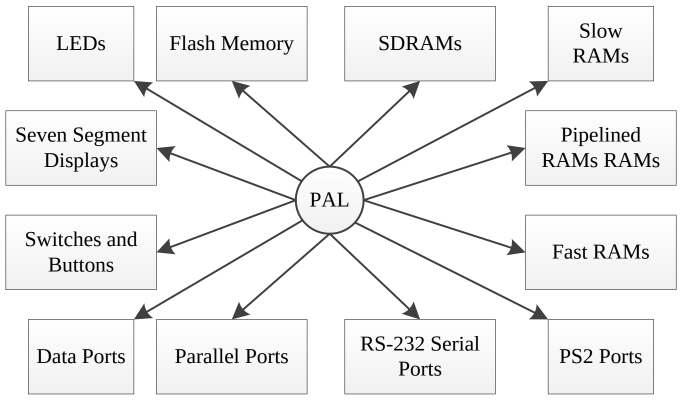
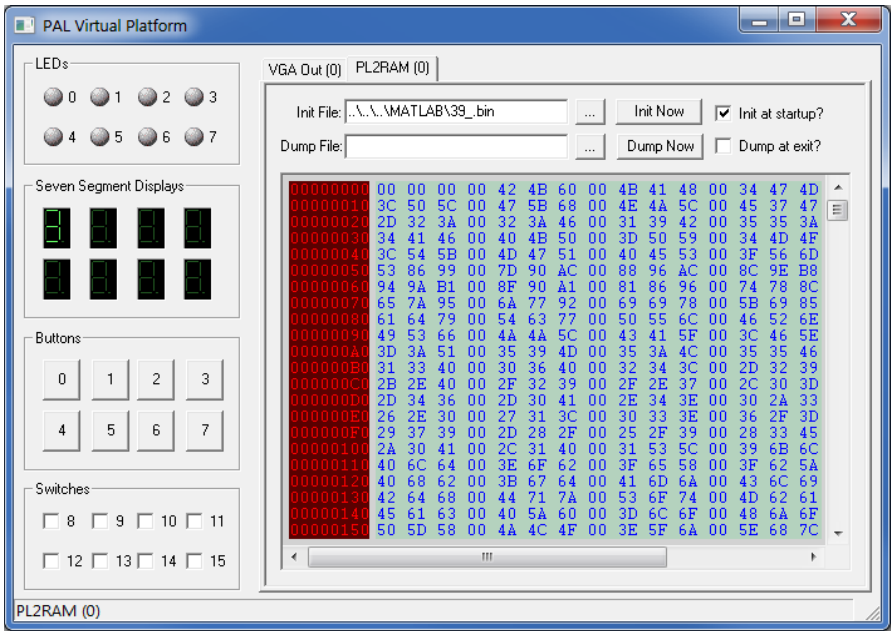


Figure B-4: The examples of PAL supported peripherals

PAL Virtual Platform

PAL Virtual Platform (PALSim) allows simulating PAL designs, where a visual representation of the behaviour of devices is provided for simulation. For example, VGA screen, RAM and LEDs can be observed from the PALSim GUI. Figure B-5 shows a PALSim example.



Init File: ..\..\..\MATLAB\39_.bin

Dump File:

[✓] Init at startup?

[ ] Dump at exit?

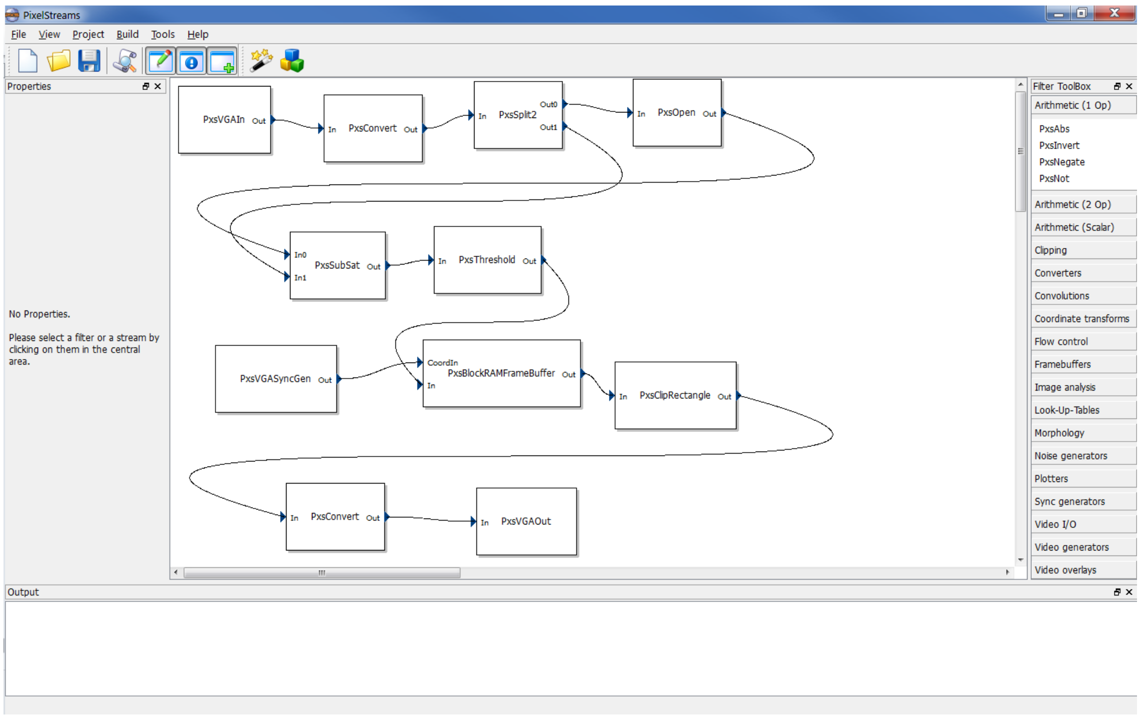
```
00000000 00 00 00 00 42 4B 60 00 4B 41 48 00 34 47 4D
00000010 3C 50 5C 00 47 5B 68 00 4E 4A 5C 00 45 37 47
00000020 2D 32 3A 00 32 3A 46 00 31 39 42 00 35 35 3A
00000030 34 41 46 00 40 4B 50 00 3D 50 59 00 34 4D 4F
00000040 3C 54 5B 00 4D 47 51 00 40 45 53 00 3F 56 6D
00000050 53 86 99 00 7D 90 AC 00 88 96 AC 00 8C 9E B8
00000060 94 9A B1 00 8F 90 A1 00 81 86 96 00 74 78 8C
00000070 65 7A 95 00 6A 77 92 00 69 69 78 00 5B 69 85
00000080 61 64 79 00 54 63 77 00 50 55 6C 00 46 52 6E
00000090 49 53 66 00 4A 4A 5C 00 43 41 5F 00 3C 46 5E
000000A0 3D 3A 51 00 35 39 4D 00 35 3A 4C 00 35 35 46
000000B0 31 33 40 00 30 36 40 00 32 34 3C 00 2D 32 39
000000C0 2B 2E 40 00 2F 32 39 00 2F 2E 37 00 2C 30 3D
000000D0 2D 34 36 00 2D 30 41 00 2E 34 3E 00 30 2A 33
000000E0 26 2E 30 00 27 31 3C 00 30 33 3E 00 36 2F 3D
000000F0 29 37 39 00 2D 28 2F 00 25 2F 39 00 28 33 45
00000100 2A 30 41 00 2C 31 40 00 31 53 5C 00 39 6B 6C
00000110 40 6C 64 00 3E 6F 62 00 3F 65 58 00 3F 62 5A
00000120 40 68 62 00 3B 67 64 00 41 6D 6A 00 43 6C 69
00000130 42 64 68 00 44 71 7A 00 53 6F 74 00 4D 62 61
00000140 45 61 63 00 40 5A 60 00 3D 6C 6F 00 48 6A 6F
00000150 50 5D 58 00 4A 4C 4F 00 3E 5F 6A 00 5E 68 7C
```

Figure B-5: An example of PALSim

PixelStreams

PixelStreams is a library of parameterisable IP for creating video processing systems, where each IP block is assembled into filter networks, connected by streams [114]. In DK design suite, filter networks can be assembled programmatically in Handel-C or graphically using the PixelStreams GUI. The PixelStreams architecture effectively eliminates these issues by providing reusable flow control components to create pipelined hardware.

PixelStreams is designed primarily for dealing with high-speed video/still image input processing and analysis, which has the high data rate and highly parallel nature of the generated hardware. Figure B-6 shows an example using PixelStreams GUI.



No Properties.

Please select a filter or a stream by clicking on them in the central area.

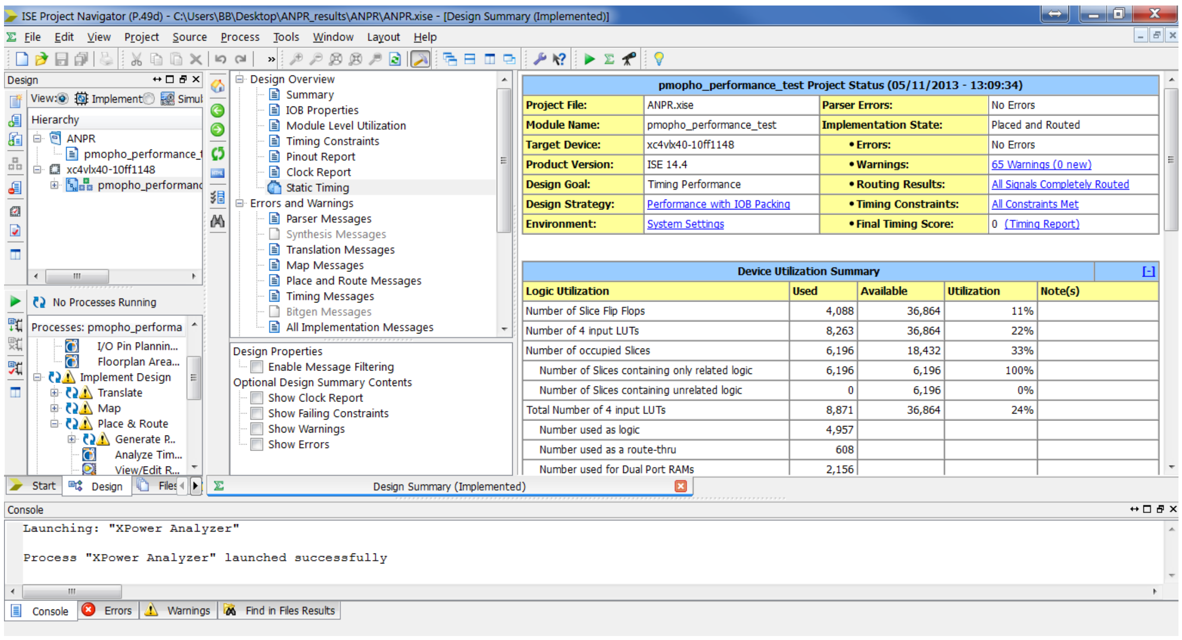
Figure B-6: An example of PixelStreams GUI

B.2 Xilinx ISE

Xilinx ISE is a tool that can synthesise and analyse HDL designs. It allows the developer to:

- synthesise the designs;
- perform timing analysis;
- simulate a design;
- power consumption analysis; and
- configure the target device [33].

Figure B-7 shows the Xilinx ISE 14 project navigator.



ISE Project Navigator (P.49d) - C:\Users\BB\Desktop\ANPR_results\ANPR\ANPR.xise - [Design Summary (Implemented)]

pmopho_performance_test Project Status (05/11/2013 - 13:09:34)			
Project File:	ANPR.xise	**Parser Errors:**	No Errors
Module Name:	pmopho_performance_test	**Implementation State:**	Placed and Routed
Target Device:	xc4vlx40-10ff1148	• **Errors:**	No Errors
Product Version:	ISE 14.4	• **Warnings:**	65 Warnings (0 new)
Design Goal:	Timing Performance	• **Routing Results:**	All Signals Completely Routed
Design Strategy:	Performance with IOB Packing	• **Timing Constraints:**	All Constraints Met
Environment:	System Settings	• **Final Timing Score:**	0 (Timing Report)

Device Utilization Summary

Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	4,088	36,864	11%	
Number of 4 input LUTs	8,263	36,864	22%	
Number of occupied Slices	6,196	18,432	33%	
Number of Slices containing only related logic	6,196	6,196	100%	
Number of Slices containing unrelated logic	0	6,196	0%	
Total Number of 4 input LUTs	8,871	36,864	24%	
Number used as logic	4,957			
Number used as a route-thru	608			
Number used for Dual Port RAMs	2,156			

```
Launching: "XPower Analyzer"

Process "XPower Analyzer" launched successfully
```

Figure B-7: the Xilinx ISE 14 project navigator

B.2.1 Xilinx Timing Analyser

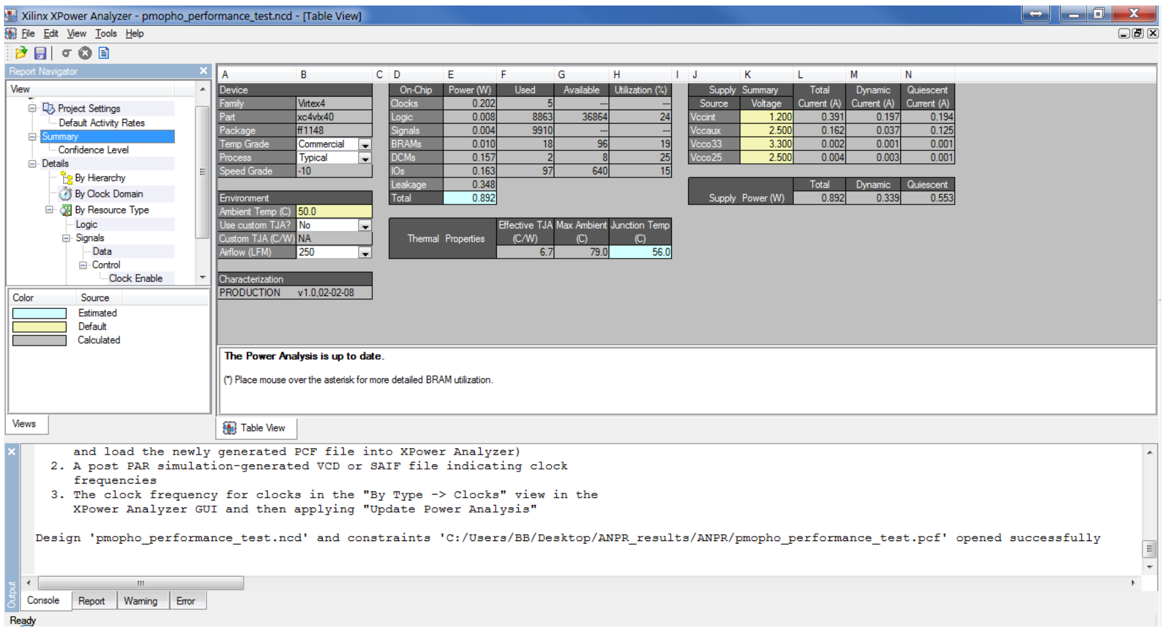
Xilinx Timing Analyser is used to perform static timing analysis of an FPGA, where a report about the delay along a given path or paths and the slack based upon the specified timing requirements are generated [147]. One of a timing analysis report is called timing summary that providing constraint coverage statistics. In the research project, the minimum period and maximum running frequency of the design are the main focus.

B.2.2 Xilinx XPower Analyser

XPower Analyser (XPA) can perform an analysis on real design data after design implementation is finished. XPA calculates power based on quiescent and dynamic power consumption in CMOS circuits:

- **Quiescent power**: it results primarily from transistor leakage current in the device. Leakage current is either from source-to-drain or through the gate oxide, and exists even when the transistor is logically "OFF".
- **Dynamic power**: it is associated with design activity and switching events in the core or I/O of the device and it is determined by nod capacitance, supply voltage, and switching frequency.

The interface of Xilinx XPA is shown in Figure B-8.



Color	Source
🟦	Estimated
🟨	Default
▨	Calculated

Device	
Family	Virtex4
Part	xc4vlx40
Package	ff1148
Temp Grade	Commercial
Process	Typical
Speed Grade	-10

Environment	
Ambient Temp (C)	50.0
Use custom TJA?	No
Custom TJA (C/W)	NA
Airflow (LFM)	250

Characterization	
PRODUCTION	v1.0,02-02-08

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.202	5	---	---
Logic	0.008	8863	36864	24
Signals	0.004	9910	---	---
BRAMs	0.010	18	96	19
DCMs	0.157	2	8	25
IOs	0.163	97	640	15
Leakage	0.348			
Total	0.892			

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	6.7	79.0	56.0

Supply Summary Source	Voltage	Total Current (A)	Dynamic Current (A)	Quiescent Current (A)
Vccint	1.200	0.391	0.197	0.194
Vccaux	2.500	0.162	0.037	0.125
Vcco33	3.300	0.002	0.001	0.001
Vcco25	2.500	0.004	0.003	0.001

	Total	Dynamic	Quiescent
Supply Power (W)	0.892	0.339	0.553

The Power Analysis is up to date.

(*) Place mouse over the asterisk for more detailed BRAM utilization.

```
    and load the newly generated PCF file into XPower Analyzer)
  2. A post PAR simulation-generated VCD or SAIF file indicating clock
     frequencies
  3. The clock frequency for clocks in the "By Type -> Clocks" view in the
     XPower Analyzer GUI and then applying "Update Power Analysis"

Design 'pmopho_performance_test.ncd' and constraints 'C:/Users/BB/Desktop/ANPR_results/ANPR/pmopho_performance_test.pcf' opened successfully
```

Figure B-8: Xilinx XPA user interface

Appendix C: Sample Codes and FPGA Chip Layouts for ANPR Implementation

In this research work, all the FPGA implementations are written in Handel-C, and every main stages or modules are presented as independent functions, which can be reused or called by the main program. In this appendix, sample codes and FPGA chip layouts for the proposed NPL, CS, OCR and pre-processing implementations are introduced in the following each section respectively.

C.1 Number Plate Localisation Implementation

C.1.1 Sample Codes for Number Plate Localisation Implementation

The NPL implementation is mainly based on the PixelStreams filters and a NP selection module, where parallel hardware implementations are generated. By using streams and pipelining the throughput remains real-time image processing performance. The main code script of the NPL implementation is exhibited below.

```
/*
 * This code is part of UH PhD research work
 * Number plate localisation
 * Copyright (c) X.Zhai 5.2010
 */

#define PAL_TARGET_CLOCK_RATE PAL_PREFERRED_VIDEO_CLOCK_RATE
#include "pal_master.hch"
#include "pxs.hch"
#include "bolbs.hch"// User libs

void main (void)
{
    /*
     * Variable
     */
```

```
macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;
macro expr Mode     = SyncGen2GetOptimalModeCT   (ClockRate);
macro expr Width     = 640;
macro expr Height    = 480;
macro expr PL2RAM    = PalPL2RAMCT (0);

static unsigned 1 t = 0;
static  signed 16 X0, Y0, X1, Y1;
static unsigned Address ;
unsigned  8 rate;
static  signed 16 wid,len;
static signed 16 X0_, Y0_, X1_, Y1_;
static unsigned 32 area;

/*
 * Streams
 */
PXS_PV_S (Stream0, PXS_EMPTY);
PXS_PV_S (Stream1, PXS_EMPTY);
PXS_PV_S (Stream2, PXS_RGB_U8);
PXS_PV_S (Stream10, PXS_MONO_U8);
PXS_PV_S (Stream15, PXS_MONO_U8);
PXS_PV_S (Stream16, PXS_MONO_U8);
PXS_PV_S (Stream17, PXS_MONO_U8);
PXS_PV_S (Stream18, PXS_MONO_U8);
PXS_PV_S (Stream19, PXS_MONO_U8);
PXS_PV_S (Stream22, PXS_MONO_U8);
PXS_PV_S (Stream23, PXS_MONO_U8);
PXS_PV_S (Stream29, PXS_MONO_U1);
PXS_PV_S (Stream30, PXS_MONO_U1);
PXS_PV_S (Stream31, PXS_MONO_U1);
PXS_PV_S (Label,    PXS_MONO_S16);
PXS_PV_S (Stream48, PXS_MONO_U8);
PXS_PV_S (Stream49, PXS_MONO_U8);
PxsBlobList Blobs;

/*
 * Filters
 */
par
{

 PxsVGASyncGen (&Stream0, Mode); //VGA Sync
```

```
    PxsValve (&Stream0, &Stream1, 1);  //Stream switching
    PxsPalPL2RAMReader (&Stream1, &Stream2, 640, PalPL2RAMCT(0), ClockRate);  //
Mems reader
    PxsConvert (&Stream2, &Stream10);  // RGB to Grayscale
    PxsSplit2 (&Stream10, &Stream15, &Stream16); //Stream Spliter
    PxsDualLineBuffer(&Stream15,&Stream48,&Stream49,Width); // DualLine Buffer
    PxsOpen3_30 ( &Stream16, &Stream17,Width);//Morphological open 3*30
    PxsSynchronise (&Stream49, &Stream17, &Stream22, &Stream23, 60);//Pixel sync
    PxsSubSat (&Stream22, &Stream23, &Stream18);//Image substraction
    PxsClipRectangle (&Stream18, &Stream19, 0, 0, 639, 479);//Image Clip
    PxsThreshold (&Stream19, &Stream29, 60, 255);//Binarisation
    PxsOpen3_3 ( &Stream29, &Stream30,Width);//Morphological open 3*3
    PxsClose3_13 (&Stream30, &Stream31, Width);//Morphological close 3*13
    PxsLabelBlobs1(&Stream31, &Label, Width, &Blobs, ClockRate);//CCA

      /*
       * NP Selection
       */
      while(1)
           {

          unsigned  i;

           PxsAwaitVSync (&Stream31);

           do
               {
                  delay;
               }while(PxsBlobListNumBlobs (&Blobs)==0);
               PxsBlobListLock (&Blobs);

               for(i=1;i<=PxsBlobListNumBlobs (&Blobs);i++)
              {

                 PxsBlobListGetBoundingBox (&Blobs, i,  &X0, &Y0, &X1, &Y1);
                 par
                 {
                 wid = Y1-Y0;
                 len = X1-X0;
                 }
                    Divide(len, wid, &rate);
                  if(rate>=3 & rate <=6 & len >=60 & wid >=20 & len <=240& wid <=50)
//rate>4 && rate<6
                  {

                     PalSevenSegWriteDigit (PalSevenSegCT (0), rate[3:0], 0);
                     PxsBlobListGetArea(&Blobs, i, &area);

                     if(area>1500 & area<15000 &Y0>30&Y0<450&Y1>50&Y1<460 )
                     {
                        par
                        {
                            X0_ = X0 ;
                            Y0_ = Y0 ;
                            X1_ = X1 +3;
                            Y1_ = Y1 ;

                        }
                        t = 1;
                        }
                    else
                    {
                        delay;
                    }
                  }


                 else
                 {
                      delay;
                 }
              }
               if (t==0)
                 {
                      for(i=1;i<=PxsBlobListNumBlobs (&Blobs);i++)
                     {

                         PxsBlobListGetBoundingBox (&Blobs, i,  &X0, &Y0, &X1, &Y1);
                         par
                         {
                         wid = Y1-Y0;
                         len = X1-X0;
                         }
                            Divide(len, wid, &rate);
```

```
                                    if((rate>=2 && rate <=4 && len >=90 && wid >=28 && len  <=
                                    200 && wid <=55)
                                     |(rate<=9&rate>=6&len>=150&wid<=30&wid>=12&len<=300)
                                     |(rate>=3&rate<=6&len>=70&wid<=30&wid>=15&len<=200)  )
                                      {

                                          PxsBlobListGetArea(&Blobs, i, &area);
                                          if(area>900 & area<15000 )
                                           {
                                             par
                                                 {
                                                     X0_ = X0 ;
                                                     Y0_ = Y0 ;
                                                     X1_ = X1+3 ;
                                                     Y1_ = Y1 ;
                                                 }
                                                 t=1;
                                           }

                                      }
                                  }

                        }
            PxsBlobListUnlock (&Blobs);
        }
    }
}
```

C.1.2 FPGA Chip Layout for Number Plate Localisation Implementation

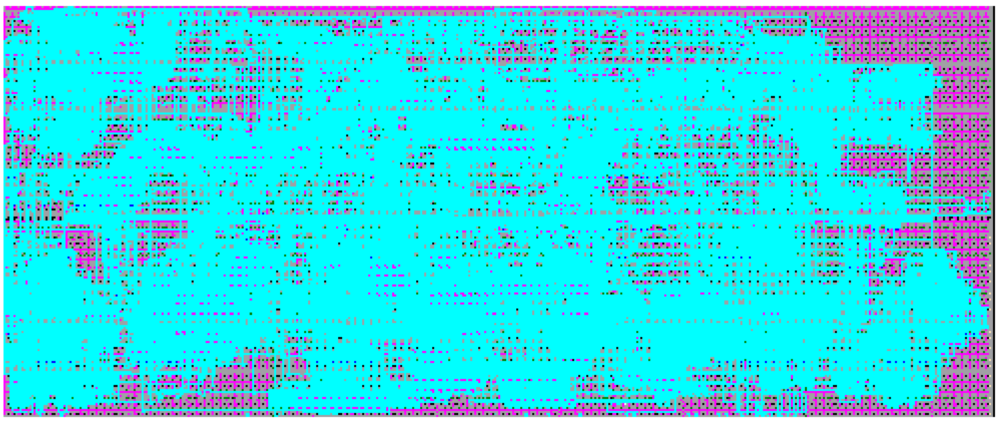


Figure C-1: FPGA Chip layout for the proposed NPL implementation

C.2 Character Segmentation Implementation

C.2.1 Sample Codes for Character Segmentation Implementation

The CS implementation mainly consists of two modules: vertical and horizontal projection modules. The two modules are implemented in pipeline manner, the horizontal projection module starts to work when the first vertical position set are localised from the vertical projection module. The partial code script for CS implementation is shown below.

```
/*
 * Vertically Reading memory and performing morphological operations
 */
        par{
              do
              {
                  par
                  {
                  seq{
                  par{
                    do
                    {
                         if(readable)
                          {
                              delay;
                          }

                          PalPL2RAMSetReadAddress (PL2RAM, Address);

                      delay;
                      par
                          {
                              PalPL2RAMRead (PL2RAM, &ReadData);
                              Address = Address + N;
                          }
                        D1++;
                      if(ReadData[0])
                          {

                             z[0]=1;
                              if(D1==1||D1==M1)
```

```
                {
                    u++;
                }

            }
        else
            {
                z[0]=0;
            }

            Temp1=1;


        }  while (D1!=M1);

    do{
      if(Temp1||D1==M1){
      if(mark==0)
      {
          par{
                   if (D2>=2&&D2<=0@(M1))

                   {
                       Erode (z[0], z[1], z[2], z[3]);
                   }
                   else
                   {
                       delay;
                   }
                   if (D2>=4&&D2<=0@(M1+2))
                   {
                       Dilate(z[3], z[4], z[5], z[6]);

                   }
                   else
                   {
                       delay;
                   }
                   if (D2>=6&&D2<=0@(M1+4))
                   {
                       Dilate(z[6], z[7], z[8], z[9]);
                   }
                   else
                   {
                       delay;
                   }
                z[8]=z[7];
                z[7]=z[6];
                z[5]=z[4];
                z[4]=z[3];
                z[2]=z[1];
                z[1]=z[0];
                if(z[9])
                   {
                       u++;
                   }
                   else
                   {
                       delay;
                   }
                       D2++;

                       Temp1=0;
            }
        }
        else
        {
            if(D2<=0@(M1)){
            par
                {
            if (D2>=2&&D2<=0@(M1))
                {
                   Dilate(z[0], z[1], z[2], Temp2);

                }
                else
                {
                   delay;
                }

                   if (Temp2)
                       {
                           u++;
                       }
                   else
                   {
```

```
                        delay;
                    }
                    D2++;
                    z[2]=z[1];
                    z[1]=z[0];
                    Temp1=0;
                }
            }
            else
            {
                D2++;
            }
        }
    }
    else
    {
        delay;
    }
    }while(D2!=M1+4);
}

       par
    {
        tt1=1;
        u_temp=u;
        u=0;
        z[0]=0;
        z[1]=0;
        z[2]=0;
        z[4]=0;
        z[5]=0;
        z[6]=0;
        z[7]=0;
        z[8]=0;
        z[9]=0;
        D2=0;
        D1=0;
        Temp2=0;
        Address = Address - A1 + 1;
    }


}
```

```
/*
 * Horizontally Reading memory and performing morphological close operation
 */
par{

        seq{
                par{
                  do
                  {



                     readable=1;
                     PalPL2RAMSetReadAddress (PL2RAM, Address1);
                     readable=0;
                      par
                          {
                              PalPL2RAMRead (PL2RAM, &ReadData1);
                              Address1++;
                          }
                          D3++;
                      if(ReadData1[0])
                          {
                              z_[0]=1;
                              if(D3==1||D3==w1)
                             {
                                 v++;
                             }
                          }
                      else
                          {
                              z_[0]=0;
                          }
                          Temp5=1;
                     }while(D3!=w1);
                  do
                   {
                       if(Temp5){
                      par
                          {
                           if  (D4>=2)
                          {
                              Dilate (z_[0], z_[1], z_[2], Temp4);
                          }
                          else
                          {
                              delay;
                          }

                              if(Temp4)
                              {
                                  v++;
                              }
                              else
                              {
                                delay;
                              }
                              Temp5=0;
                              D4++;
                              z_[2]=z_[1];
                              z_[1]=z_[0];
                          }
                      }
                      else
                      {
                          delay;
                      }
                  }  while (D4 !=w1);
              }
              par
                  {
                      v_temp=v;
                      z_[0]=0;
                      z_[1]=0;
                      z_[2]=0;
                      z_[3]=0;
                      D3=0;
                      D4=0;
                      Address1 = Address1 + N-w2;
                      v=0;
                      tt4=1;
                  }
          }
```

C.2.2 FPGA Chip Layout for Character Segmentation Implementation

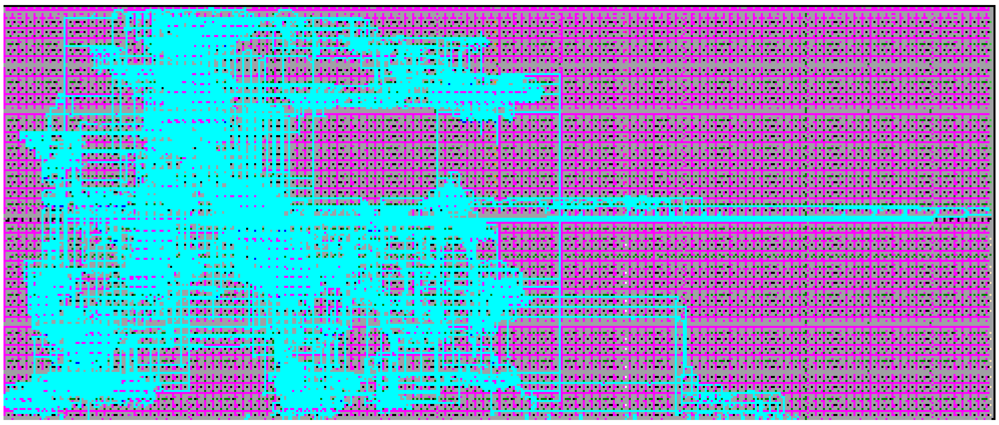


Figure C-2: FPGA Chip layout for the proposed CS implementation

C.3 Optical Character Recognition Implementation

C.3.1 Sample Codes for Optical Character Recognition Implementation

The OCR implementation consists of two NN layers: hidden and output layers. Pipelining is used in their implementations to achieve high throughput. Fixed-point arithmetic is used to represent the proposed NN weights and perform their calculations. The partial code scripts are shown below.

```
/*
 * Fix-point declaration
 */
typedef FIXED_SIGNED(10, 14) MyFixed;
typedef FIXED_UNSIGNED(11, 14) MyFixed2;
/*
 * Hidden layer
 */
        par
            {
                PalPL2RAMSetReadAddress (PL2RAM0, Address);
                PalPL2RAMSetReadAddress (PL2RAM1, Address);
                PalPL2RAMSetReadAddress (PL2RAM2, Address);
            }
        delay;
```

```
par
    {
        PalPL2RAMRead (PL2RAM0, &T1);
        PalPL2RAMRead (PL2RAM1, &T2);
        PalPL2RAMRead (PL2RAM2, &T3);
        Address++;
    }
    img=img1[i];
par
    {
      if(img[1]==0)
      {

        if(T1[15]==1)
          {
            fix2.FixedIntBits=(-1@(signed)(T1[15:14]));
          }
          else
          {
            fix2.FixedIntBits=(signed)(0@T1[15:14]);
          }
        if(T1[13]==1)
        {
            fix2.FixedFracBits=(-1@(signed)T1[12:0]);
        }

        else
          {
            fix2.FixedFracBits=(signed)(T1[13:0]);
          }
      }
      else
      {
          par
          {
            fix2.FixedIntBits=0;
            fix2.FixedFracBits=0;
          }
      }
      if(img[0]==0)
      {
        if(T1[31]==1)
          {
            fix1.FixedIntBits=(-1@(signed)(T1[31:30]));
          }
          else
          {
              fix1.FixedIntBits=(signed)(0@T1[31:30]);
          }
        if(T1[29]==1)
        {
            fix1.FixedFracBits=(-1@(signed)(T1[28:16]));
        }
          else
          {

            fix1.FixedFracBits=(signed)(T1[29:16]);
          }
     }
     else
     {
          par
          {
            fix1.FixedIntBits=0;
            fix1.FixedFracBits=0;
          }
     }
     if(img[3]==0)
     {
        if(T2[15]==1)
          {
            fix4.FixedIntBits=(-1@(signed)(T2[15:14]));

          }
          else
          {
              fix4.FixedIntBits=(signed)(0@T2[15:14]);
          }
          if(T2[13]==1)
          {
             fix4.FixedFracBits=(-1@(signed)T2[12:0]);
          }
        else
          {

            fix4.FixedFracBits=(signed)(T2[13:0]);
          }

      }
      else
      {
          par
          {
            fix4.FixedIntBits=0;
            fix4.FixedFracBits=0;
          }
      }
      if(img[2]==0)
      {
          if(T2[31]==1)
          {
            fix3.FixedIntBits=(-1@(signed)(T2[31:30]));

          }
          else
          {
              fix3.FixedIntBits=(signed)(0@T2[31:30]);
          }
          if(T2[29]==1)
          {
              fix3.FixedFracBits=(-1@(signed)(T2[28:16]));
          }
          else
          {

              fix3.FixedFracBits=(signed)(T2[29:16]);
          }
      }
      else
      {
          par
          {
            fix3.FixedIntBits=0;
            fix3.FixedFracBits=0;
          }
      }
      if(img[5]==0)
      {
         if(T3[15]==1)
```

```
            {
              fix6.FixedIntBits=(-1@(signed)(T3[15:14]));
            }
            else
            {
                fix6.FixedIntBits=(signed)(0@T3[15:14]);
            }
            if(T3[13]==1)
            {
                fix6.FixedFracBits=(-1@(signed)(T3[12:0]));
            }
          else
            {

                fix6.FixedFracBits=(signed)(T3[13:0]);
            }
        }
        else
        {
            par
            {
              fix6.FixedIntBits=0;
              fix6.FixedFracBits=0;
            }
        }
        if(img[4]==0)
        {
          if(T3[31]==1)
            {
              fix5.FixedIntBits=(-1@(signed)(T3[31:30]));
            }
            else
            {
                fix5.FixedIntBits=(signed)(0@T3[31:30]);
            }
            if(T3[29]==1)
            {
                fix5.FixedFracBits=(-1@(signed)T3[28:16]);
            }
            else
            {
                fix5.FixedFracBits=(signed)(T3[29:16]);
            }
```

```
              }
              else
              {
                  par
                  {
                    fix5.FixedIntBits=0;
                    fix5.FixedFracBits=0;
                  }
              }
            }
         par
            {
                fix_t1=FixedAdd(fix1, fix2);
                fix_t2=FixedAdd(fix3, fix4);
                fix_t3=FixedAdd(fix5, fix6);
            }
            fix_f1.FixedIntBits=fix_N[m].FixedIntBits;
            fix_f1.FixedFracBits=fix_N[m].FixedFracBits;
         par
            {
                fix_t4=FixedAdd(fix_t1, fix_t2);
                fix_t5=FixedAdd(fix_t3, fix_f1);
            }
            fix_f1=FixedAdd(fix_t4, fix_t5);
            fix_N[m].FixedIntBits=fix_f1.FixedIntBits;
            fix_N[m].FixedFracBits=fix_f1.FixedFracBits;
         if(Address%125==0)
         {

             m++;
             fix_N[m].FixedIntBits=0;
             fix_N[m].FixedFracBits=0;
             i=0;
             m_temp=1;
         }
         else
         {
            i++;
         }
/*
 * Output layer
```

```
*/

     par
       {
         seq
           {
            do{
               do
               {
               if(lw[lwi][15]==1)
                   {
                   lw_t[lw_ti].FixedIntBits=-1@(signed)(lw[lwi][15:14]);
                   }
               else
                   {
                   lw_t[lw_ti].FixedIntBits=(signed)(0@lw[lwi][15:14]);
                   }
               if(lw[lwi][13]==1)
                   {
                   lw_t[lw_ti].FixedFracBits=-1@(signed)(lw[lwi][12:0]);
                   }
               else
                   {
                   lw_t[lw_ti].FixedFracBits=(signed)(lw[lwi][13:0]);
                   }
                   lw_ti++;
                   lwi++;
               }while(lw_ti!=0);

               par
               {
                   fix_lw_mu1 = FixedMultSigned(lw_t[0],fix_f);
                   fix_lw_mu2 = FixedMultSigned(lw_t[1],fix_f);
               }
               fix_N2[N2i]=FixedAdd(fix_N2[N2i],fix_lw_mu1);
               N2i++;
               fix_N2[N2i]=FixedAdd(fix_N2[N2i],fix_lw_mu2);
               N2i++;
            }while(N2i!=0);

            if(lwi==1600)
             {
             do{
                do{
                   if(lb[ibi][15]==1)
                   {
                   lw_t[lw_ti].FixedIntBits=-1@(signed)(lb[ibi][15:14]);
                   }
                 else
                   {
                   lw_t[lw_ti].FixedIntBits=(signed)(0@lb[ibi][15:14]);
                   }
                 if(lb[ibi][13]==1)
                   {
                   lw_t[lw_ti].FixedFracBits=-1@(signed)(lb[ibi][12:0]);
                   }
                 else
                   {
                   lw_t[lw_ti].FixedFracBits=(signed)(lb[ibi][13:0]);
                   }
                   lw_ti++;
                   ibi++;
                }while(lw_ti!=0);
                par{
                fix_N2[N2i]=FixedAdd(fix_N2[N2i],lw_t[0]);
                fix_N2[N2i+1]=FixedAdd(fix_N2[N2i+1],lw_t[1]);
                }
                N2i=N2i+2;
                m_temp2=1;
               }while(N2i!=0);
            }
            else
            {
                delay;
            }
          }
        seq
          {
             fix_tt1=fix_N[m_1];
            if(FixedGT(fix_tt1,fix_5)||FixedLT(fix_tt1,fix__5))
              {
                 if(FixedLT(fix_tt1,fix__5))
                 {
                     par
                     {
                         fix_f.FixedIntBits=-1;
```

```
                    fix_f.FixedFracBits=0;
                }
            }
            else
            {
                par
                {
                    fix_f.FixedIntBits=1;
                    fix_f.FixedFracBits=0;
                }
            }
        }
    else
        {
            fix7=FixedAdd(fix_N[m_1], fix_5);
            sig_temp=FixedCastSigned(FIXED_ISUNSIGNED, 11, 14, fix7);
            fix8=FixedMultUnsigned(sig_temp, fix_100);
            i1=(unsigned)FixedToInt(fix8);
            a=Tan_sig[i1];
            par
            {
                if(a[15]==1)
                {
                fix_f.FixedIntBits=-1@(signed)(a[15:14]);
                }
                else
                {
                fix_f.FixedIntBits=(signed)(0@a[15:14]);
                }
                if(a[13]==1)
                {
                fix_f.FixedFracBits=-1@(signed)(a[12:0]);
                }
                else
                {
                fix_f.FixedFracBits=(signed)(a[13:0]);
                }
            }
        }
    }
}
```

C.3.2 FPGA Chip Layout for Optical Character Recognition Implementation

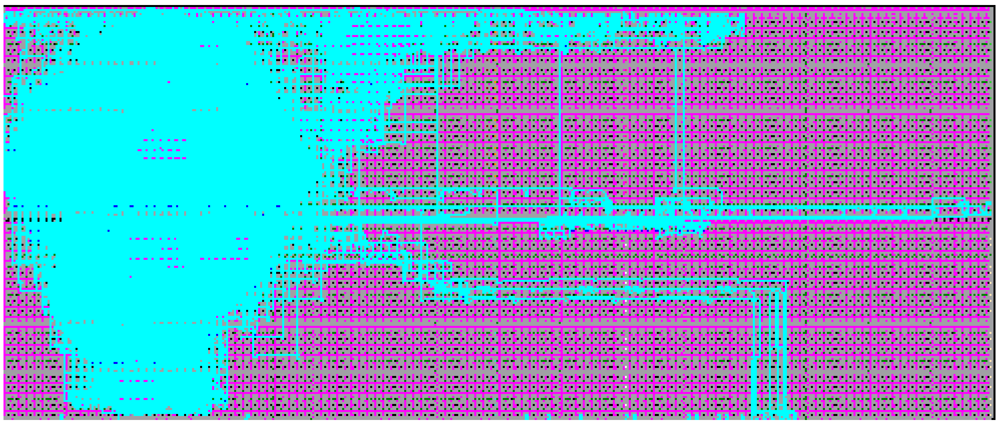


Figure C-3: FPGA Chip layout for the proposed OCR implementation

C.4 Pre-processing Implementation

C.4.1 Sample Codes for Pre-processing Implementation

The pre-processing implementation consists of binarisation and adjustment modules, they are running in pipeline manner to achieve high throughput. The adjustment module starts to work after the first processed pixel is generated from the binarisation module. The partial code scripts of pre-processing implementation are shown below.

```
/*
 * Binarisation
 */

par
{
    LineBuffer.W[HY[5:0]-3] = ((WX!=0)?(Yout<<8)+adju(Y,56):(Y@Y@Y@Y@Y@Y@Y));

    if(WX>=3)
    {
        if(HY!=3)
        {
    par
    {
```

```
par (j = 0; j < 7; j++)
{
    par (i = 0; i < 8; i++)
    {
        Matrix_8x8[j][i] = Matrix_8x8[j + 1][i];
    }
}

Matrix_8x8[7][0] = Yout[55:48];
Matrix_8x8[7][1] = Yout[47:40];
Matrix_8x8[7][2] = Yout[39:32];
Matrix_8x8[7][3] = Yout[31:24];
Matrix_8x8[7][4] = Yout[23:16];
Matrix_8x8[7][5] = Yout[15:8];
Matrix_8x8[7][6] = Yout[7:0];
Matrix_8x8[7][7] = Y;

if(HY>=7)
{
    par
    {

    par
    {
        S[0] = adju(Matrix_8x8[0][0], 16) + adju(Matrix_8x8[0][1], 16)
        + adju(Matrix_8x8[0][2], 16) + adju(Matrix_8x8[0][3], 16);
        S[1] = adju(Matrix_8x8[0][4], 16) + adju(Matrix_8x8[0][5], 16)
        + adju(Matrix_8x8[0][6], 16) + adju(Matrix_8x8[0][7], 16);
        S[2] = adju(Matrix_8x8[1][0], 16) + adju(Matrix_8x8[1][1], 16)
        + adju(Matrix_8x8[1][2], 16) + adju(Matrix_8x8[1][3], 16);
        S[3] = adju(Matrix_8x8[1][4], 16) + adju(Matrix_8x8[1][5], 16)
        + adju(Matrix_8x8[1][6], 16) + adju(Matrix_8x8[1][7], 16);
        S[4] = adju(Matrix_8x8[2][0], 16) + adju(Matrix_8x8[2][1], 16)
        + adju(Matrix_8x8[2][2], 16) + adju(Matrix_8x8[2][3], 16);
        S[5] = adju(Matrix_8x8[2][4], 16) + adju(Matrix_8x8[2][5], 16)
        + adju(Matrix_8x8[2][6], 16) + adju(Matrix_8x8[2][7], 16);
        S[6] = adju(Matrix_8x8[3][0], 16) + adju(Matrix_8x8[3][1], 16)
        + adju(Matrix_8x8[3][2], 16) + adju(Matrix_8x8[3][3], 16);
        S[7] = adju(Matrix_8x8[3][4], 16) + adju(Matrix_8x8[3][5], 16)
        + adju(Matrix_8x8[3][6], 16) + adju(Matrix_8x8[3][7], 16);
        S[8] = adju(Matrix_8x8[4][0], 16) + adju(Matrix_8x8[4][1], 16)
        + adju(Matrix_8x8[4][2], 16) + adju(Matrix_8x8[4][3], 16);
        S[9] = adju(Matrix_8x8[4][4], 16) + adju(Matrix_8x8[4][5], 16)
```

```
+ adju(Matrix_8x8[4][6], 16) + adju(Matrix_8x8[4][7], 16);
S[10] = adju(Matrix_8x8[5][0], 16) + adju(Matrix_8x8[5][1], 16)
+ adju(Matrix_8x8[5][2], 16) + adju(Matrix_8x8[5][3], 16);
S[11] = adju(Matrix_8x8[5][4], 16) + adju(Matrix_8x8[5][5], 16)
+ adju(Matrix_8x8[5][6], 16)+ adju(Matrix_8x8[5][7], 16);
S[12] = adju(Matrix_8x8[6][0], 16) + adju(Matrix_8x8[6][1], 16)
+ adju(Matrix_8x8[6][2], 16) + adju(Matrix_8x8[6][3], 16);
S[13] = adju(Matrix_8x8[6][4], 16) + adju(Matrix_8x8[6][5], 16)
+ adju(Matrix_8x8[6][6], 16) + adju(Matrix_8x8[6][7], 16);
S[14] = adju(Matrix_8x8[7][0], 16) + adju(Matrix_8x8[7][1], 16)
+ adju(Matrix_8x8[7][2], 16) + adju(Matrix_8x8[7][3], 16);
S[15] = adju(Matrix_8x8[7][4], 16) + adju(Matrix_8x8[7][5], 16)
+ adju(Matrix_8x8[7][6], 16) + adju(Matrix_8x8[7][7], 16);
if(HY>=8)
{
    par
    {
        Q[0] = S[0] + S[1] + S[2] + S[3];
        Q[1] = S[4] + S[5] + S[6] + S[7];
        Q[2] = S[8] + S[9] + S[10] + S[11];
        Q[3] = S[12] + S[13] + S[14] + S[15];
        if(HY>=9)
        {
            par
            {
        Y_average = (signed)adju((Q[0] + Q[1] + Q[2] + Q[3])>>6, 9) ;
    Y_filted = Y_average -(signed)adju(Matrix_8x8[2][4],9) - 12;//

                if(Y_filted>=-5)
                {
                    par
                    {
      PixelsBuffer.W[PixelsBuffer_index_W] = 1;
                    PixelsBuffer_index_W++;
                    }
                }
                else
                {
                    par
                    {

                        PixelsBuffer.W[PixelsBuffer_index_W] = 0;
                        PixelsBuffer_index_W++;
```

```
                                    }
                                }
                            }
                        }
                        else
                        {
                            delay;
                        }
                    }
                }
                else
                {
                    delay;
                }
            }
        }
        }
        else
        {
            delay;
        }
}

/*
 * Adjustment
 */

par
{

temp1 = adjs(adju(HY1,10), 11);
temp2 = Hei_half - (adjs(adju(HY1,10),11));
temp5 = adjs(adju(Hei,8),11) - adjs(adju(HY1,10), 11);
HY1++;
if(HY1!=0)
{
    par
    {
        temp6 = temp5/adjs(Tan_a_trible,11);
        HYnew = adjs(((temp1) + (adjs(WX1,11) - Len_half)/adjs(Tan_a,11)),7);
        WXnew = adjs(adjs(adju(WX1,10), 11) + temp2/adjs(Tan_a,11),10);

        HYnew_= HYnew;
```

```
         WXnew_align = WXnew - adjs(temp6,10);

        // stage 1
         HYnew_1= HYnew_;
         WXnew_= WXnew_align;
        // stage 2
         HYnew_2 = HYnew_1;
         WXnew_1 = WXnew_;
        // stage 3
         HYnew_3 = HYnew_2;
         WXnew_2 = WXnew_1;
        // stage 4
         HYnew_4 = HYnew_3;
         WXnew_3 = WXnew_2;

         temp3 = adju(HYnew_1,18) + temp4;
         temp4 = adju(WXnew_align,18) * adju(Hei,18);

        // PixelsBuffer_index_R = ((temp3)%256)[7:0];
        PixelsBuffer_index_R = ((temp3))[15:0];
         if(HYnew_4>=0 && WXnew_3>=0 && HY1>=(6+H_dif[6:0]))
         {
             par
             {

                 if(adju(HYnew_4,7)<=Hei && adju(WXnew_3,9)<=Len )
                 {
                     par
                     {
                         Outputpixel = PixelsBuffer.R[PixelsBuffer_index_R];
                         sign2 = 1;
                         if(sign2)
                         {

                           par
                            {
                                ImageBuffer_V.W[index1] = Outputpixel;
                                ImageBuffer_H.W[index2] = Outputpixel;
                                index1++;
                                index2++;
                            }
                         }
                         else
```

```
                        {
                            delay;
                        }
                    }
                }
                else
                {
                    par
                    {
                        ImageBuffer_V.W[index1] = 0;
                        ImageBuffer_H.W[index2] = 0;
                        index1++;
                        index2++;
                    }
                }
            }
        }
        else
        {
            if(HY1>=(6+H_dif[6:0]))
            {
                par
                {
                    ImageBuffer_V.W[index1] = 0;
                    ImageBuffer_H.W[index2] = 0;
                    index1++;
                    index2++;
                }
            }
            else
            {
                delay;
            }
        }

    }
}
    else
    {
        delay;
    }
}
```

C.4.2 FPGA Chip Layout for Pre-processing Implementation

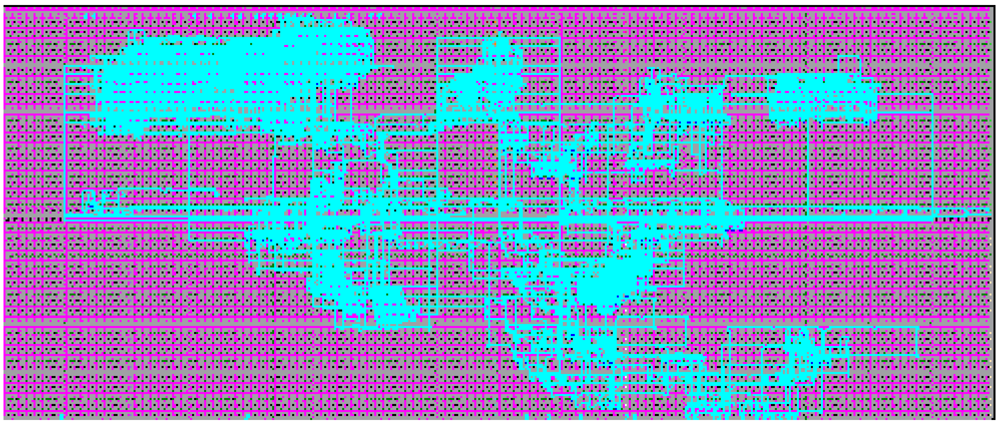


Figure C-4: FPGA Chip layout for the proposed pre-processing implementation

C.5 Entire ANPR Implementation

C.5.1 Sample Codes for Entire ANPR Implementation

The previous NPL, CS, OCR and pre-processing module are linked together to form the entire ANPR FPGA implementation, where the modules are running in parallel and pipelining manner. The entire ANPR system consists of two parts, RC240 FPGA development board and a GUI running in a host application. The pseudocode scripts are shown below.

```
/*
 * Host application pseudocode
 */
do
{
    load car image(n) from hardisk;
    for (i=1;i<=image_size;i++)
    {
        usb_write(pixel.R(i));
        usb_write(pixel.G(i));
        usb_write(pixel.B(i));
    }
    usb_read(NP.X0, NP.Y0, NP.X1, NP.Y1);
    plot_rectangluar(NP.X0, NP.Y0, NP.X1, NP.Y1);
    usb_read(adjusted_NP_width);
    usb_read(adjusted_NP_height);
    for (t=1; t<=adjusted_NP_width*adjusted_NP_hight;t++)
    {
      usb_read(adjusted_NP_pixel(t));
    }
    plot(adjusted_NP);
    usb_read(number_characted_segmented);
    for (t=1; t<=number_characted_segmented;t++)
    {
      usb_read(segmented_characte_.X0);
      usb_read(segmented_characte_.Y0);
      usb_read(segmented_characte_.X1);
      usb_read(segmented_characte_.Y1);
      plot_rectangluar(segmented_characte_.X0,
      segmented_characte_.Y0, segmented_characte_.X1,
      segmented_characte_.Y1);
    }
    for (t=1; t<=number_characted_segmented;t++)
    {
      usb_read(recognised_character);
      display(recognised_character);
    }
    n++;
}while(1);
/*
 * FPGA Implementation pseudocode
 */
void main()
{
    par
    {
        do
        {
            par
            {
                set_memory_address (PL2RAM0, Address);
                seq
                    {
                        usb_Read (USBMicro, &(R));
                        PalDataPortRead (USBMicro, &(G));
                        PalDataPortRead (USBMicro, &(B));
                    }
            }
            par
            {
                memory_write (PL2RAM0, 0 @ R @ G @ B);
                Address++;
            }
        } while (Address != image_size);
        NPL_module();
        while(1)
        {
            if (NP_candidate == 1)
            {
                NP = NP_selection();
                if(NP == 1)
                {
                    usb_write(NP_coordinates);
                }
            }
        }
        while(1)
        {
            if (NP == 1)
            {
                binarised_NP = NP_binarisation();
            }
        }
        while(1)
        {
            if (binarised_NP == 1)
            {
                adjusted_NP = NP_adjustment();
                usb_write(adjusted_NP_size);
                usb_write(adjusted_NP);
            }
        }
        while(1)
        {
            if (adjusted_NP == 1)
            {
                segmented_character = CS_module();
                usb_write(number_segemented_character);
                usb_write(segmented_character);
            }
        }
        while(1)
        {
            if (segmented_character == 1)
            {
                recognised_character = OCR_module();
                usb_write(recognised_character);
            }
        }

    }
}
```

C.5.2 FPGA Chip Layout for Entire ANPR Implementation

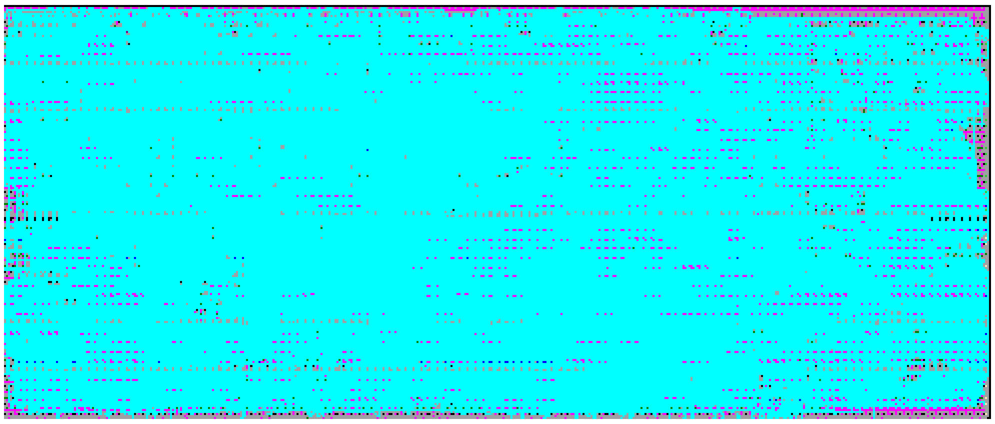


Figure C-5: FPGA Chip layout for the entire ANPR implementation