CAR LICENSE PLATE DETECTION

Naikur Bharatkumar Gohil
B.E., Dharmsinh Desai University, India, 2006

PROJECT

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

ELECTRICAL AND ELECTRONIC ENGINEERING

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

FALL
2010

CAR LICENSE PLATE DETECTION

A Project

by

Naikur Bharatkumar Gohil

Approved by:

____, Committee Chair
Jing Pang, Ph. D.

____, Second Reader
Preetham Kumar, Ph. D.

____
Date

Student: Naikur Bharatkumar Gohil

I certify that this student has met the requirements for the format contained in the University format manual, and that this project is suitable for shelving in the Library and credits is to be awarded for the Project.

__________________________, Graduate Coordinator ______________________
Preetham Kumar, Ph. D. Date

Department of Electrical and Electronic Engineering

Abstract

of

CAR LICENSE PLATE DETECTION

by

Naikur Bharatkumar Gohil

In last couple of decades, the number of vehicles has increased drastically. With this increase, it is becoming difficult to keep track of each vehicle for purpose of law enforcement and traffic management. License Plate Recognition is used increasingly nowadays for automatic toll collection, maintaining traffic activities and law enforcement. Many techniques have been proposed for plate detection, each having its own advantages and disadvantages. The basic step in License Plate Detection is localization of number plate. The approach mentioned in this project is a histogram based approach. This approach has an advantage of being simple and thus faster. Initially, license plate localization is implemented using MATLAB and verified for its functionality. Once the functionality is verified, the algorithm is implemented on EVM320DM6437 Texas Instrument (TI) Digital Video Development Starter Kit (DVDSK). By implementing this algorithm on a kit, we eliminate need of computers and thus think of portable implementation of such application.

In this approach, a digital camera is used to capture video and feed it to the kit. The kit processes each frame individually and provides the co-ordinates of location with maximum probability of having a number plate. Later, this information is used for recognizing actual number of the license plate. The plate detection algorithm is implemented in C, compiled and debugged using Code Composer Studio (CCS), an IDE provided by TI. Successful demonstrations were given by downloading the algorithm onto TI's EVM320DM6437 DVDSK.

______________________, Committee Chair
Jing Pang, Ph. D.

____
Date

ACKNOWLEDGMENTS

First, I would like to thank my project advisor Dr. Jing Pang for providing me an opportunity to work on this project under her guidance. She has provided valuable guidance, time and resources during different phases of this project. She had always provided motivation and shown utter dedication towards her work. By providing an opportunity to work on this project, she helped me to gain profound understanding of image and video processing fundamentals. She also inspired me to work on EVM320DM6437 DSP Hardware to implement this project and making the implementation successful.

I would also like to thank Dr. Preetham Kumar for giving a second look to my project and providing valuable suggestion to improve this project. His support during completion phase of my project had been very instrumental.

Lastly, I would like to thank my friends for providing me constant inspiration and support during various aspects of this project.

Naikur Bharatkumar Gohil

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Introduction

With increasing number of vehicles on roads, it is getting difficult to manually enforce laws and traffic rules for smooth traffic flow. Toll-booths are constructed on freeways and parking structures, where the car has to stop to pay the toll or parking fees. Also, Traffic Management systems are installed on freeways to check for vehicles moving at speeds not permitted by law. All these processes have a scope of improvement. In the center of all these systems lies a vehicle. In order to automate these processes and make them more effective, a system is required to easily identify a vehicle. The important question here is how to identify a particular vehicle? The obvious answer to this question is by using the vehicle’s number plate.

Vehicles in each country have a unique license number, which is written on its license plate. This number distinguishes one vehicle from the other, which is useful especially when both are of same make and model. An automated system can be implemented to identify the license plate of a vehicle and extract the characters from the region containing a license plate. The license plate number can be used to retrieve more information about the vehicle and its owner, which can be used for further processing. Such an automated system should be small in size, portable and be able to process data at sufficient rate.

Various license plate detection algorithms have been developed in past few years. Each of these algorithms has their own advantages and disadvantages. Arth *et al.* [1] described the method in which license plate is detected using confidence related predictions. As multiple detections are available for single license plate, post –processing methods are applied to merge all detected regions. In addition, trackers are used to limit the search region to certain areas in an image. Kwasnicka *at el.* [2] suggests a different approach of detection using binarization and elimination of unnecessary regions from an image. In this approach, initial image processing and binarization of an image is carried out based on the contrast between characters and background in license plate. After binarizing the image, it is divided into different black and white regions. These regions are passed through elimination stage to get the final region having most probability of containing a number plate.

1.2 Purpose of this Project

The main purpose of this project is to detect a license plate from an image provided by a camera. An efficient algorithm is developed to detect a license plate in various luminance conditions. This algorithm extracts the license plate data from an image and provides it as an input to the stage of Car License Plate Recognition. This algorithm is downloaded onto Texas Instrument's TMS320DM6437 digital video development platform. The image of a vehicle is given as an input from the camera. Extracted image of the number plate can be seen on television for verification purpose.

This author wants to gain an insight on implementing image-processing algorithm on actual hardware. The scope of this project is to detect the license plate from the given image and observe the output on television. This project can work as a base for future improvements in the field of image processing, especially in license plate extraction and plate number recognition.

1.3 Significance of this Project

Through this project, an algorithm for license plate detection from an image is implemented in MATLAB. On the hardware side, this algorithm is downloaded onto TI's Digital Video Development Platform that contains Texas Instrument's DaVinci Digital Signal Processors. Code Composer Studio is used to develop and implement the algorithm on actual hardware using C programming language. The EVM320DM6437 Development Kit is also studied for the purpose of available resources and their appropriate usage.

1.4 Organization of Report

Chapter two of the report explains the fundamentals of image processing, information about pixel in an image, RGB and YCbCr format and NTSC and PAL standard. It also explains the way a captured video frame is stored in EVM320DM6437 kit.

Chapter three provides information about the overall architecture of Texas Instrument's Digital Video Development platform. It also provides brief information about various components present on the hardware board.

Chapter four explains the algorithm developed for license plate detection in MATLAB. It also discusses about the necessity and output of each step of the algorithm in detail.

Chapter five contains information about the implementation of the algorithm for license plate detection on actual hardware. It discusses about various steps of algorithm in detail. Suitable images are also provided to show output at different stages in applied algorithm.

The information about the results obtained after implementing this project and its performance analysis is concluded in Chapter six of this report.

Chapter 2

FUNDAMENTALS OF IMAGE PROCESSING

An image is used to convey useful information in a visible format. An image is nothing but an arrangement of tiny elements in a two-dimensional plane. These tiny elements are called Pixels. A large number of pixels combine together to form an image, whether small or large.

Each pixel represents certain information about the image, like color, light intensity and luminance. A large number of such pixels combine together to form an image. Pixel is the basic element used to describe an image. Mostly, each pixel in an image is represented in either RGB (Red Green Blue) format or YCbCr format. In case of an RGB image, all the three components, namely R, G and B combine together to convey information about the color and brightness of a single pixel. Each component consumes certain memory space during image processing.

In case of a YCbCr image, each pixel in an image is represented as a combination of Y and Cb/Cr values. Here, Y stands for luminance, which describes light intensity, and Cb/Cr stands for chroma component, which describes color information for an image. Over the time, it has been found that YCbCr components of an image convey sufficient amount of information compared to its counter parts RGB, with less amount of memory space. This is a major advantage nowadays, as most of the applications require sufficient information at very high speed and less storage.

2.1 RGB Format

In case of an RGB image, each pixel is represented by three different components R, G and B. Each of these components requires at least 8 bits for their storage. In general, a single pixel may require upto 8 * 3 bits for its storage. An example of a representation of single pixel in RGB format is shown below.

R	G	B	R	G	B

Fig. 2.1 Representation of pixels in RGB format.

The value of R, G and B each ranges from 0-255. A value of (0, 0, 0) represents a black pixel, (255, 0, 0) represents a red pixel and (0, 255, 0) represents a green pixel. So, 8 bits are required to store value for a single component.

2.2 YCbCr Format

In contrast to RGB format, the YCbCr format is available with various kind of interleaving. For example, a 4:2:2 YCbCr format suggests that a single pixel is represented by two components, Y and C. Cb and Cr components are interleaved among the pixels. So if one pixel is represented by a combination of Y and Cb, the adjacent pixel will be represented by a combination of Y and Cr. Even if the Cb and Cr components are interleaved, its effect is not visible to human eye.

Y	Cb	Y	Cr	Y	Cb

Fig. 2.2 Representation of pixels in YCbCr format.

Values for Y, Cb and Cr vary from 0-255. Thus, to store a single pixel, the amount of storage required is 8 * 2 bits, which is less compared to that required by RGB format. For this project, Texas Instrument's EVM320DM6437 kit is to be used for license plate detection. The kit contains internal buffers to store the incoming frames of video. The format for the type of storage is shown below.

Cb	Y	Cr	Y
Cb	Y	Cr	Y
Cb	Y	Cr	Y

Fig. 2.3 A part of frame buffer storage for input video frames.

From the above image, it is seen that the storage of frame starts with a C component and then a Y component. Therefore, at the 0th location, one can find the C component while at the 1st and alternate locations of Frame Buffer, one can find the Y component.

2.3 NTSC and PAL Standards

NTSC and PAL are the two most commonly used standards used for broadcasting. NTSC stands for National Television System Committee. This standard is being used in most parts of Northern America and countries like South Korea, Japan, and etcetera. Videos broadcasted using NTSC standard contains a sequence of images with resolution of 720 * 480 pixels. The video is displayed at the frame rate of 30 frames per second.

PAL stands for Phase Alternate Line. PAL Standard is used mainly in countries like India, China, United Kingdom, and etcetera. This standard supports the video resolution of 720 * 576 pixels at the frame rate of 25 frames per second.

NTSC system is to be used for this project while working on Texas Instrument's EVM320DM6437 kit. This kit supports both the standards. A jumper setting on kit is available to switch between these video standards.

Chapter 3

TMS320DM6437 AND CODE COMPOSER STUDIO OVERVIEW

The TMS320DM6437 Digital Video Development Platform (DVDP) includes high performance software programmable Digital Media Processors (DMP), which reduces time-to-market for development of new multimedia applications. DMP, being programmable, provides an important feature of flexibility that helps in easy development and debugging of multimedia applications. The kit is designed to support various Video-over-IP applications such as set-top boxes, surveillance cameras, digital video recorders and encoding and decoding flexibility for various industry-standards available in market.

There are many advantages of using DMP over FPGA or ASIC. ASIC has few disadvantages amongst which its higher cost and longer time-to-market are crucial in mass production and marketing. FPGAs too are considerably harder to program in certain applications. DMP provides a good balance in terms of cost, flexibility, easily programmability and the time-to-market compared to rest two. The new DaVinci series processor included in this DVDP supports new set of extended instructions that helps to increase its overall performance.

3.1 Main Components of TMS320DM6437 DVDP

This platform contains both stand-alone and PCI-based evaluation and development of any application that uses TI’s DaVinci processors. It contains various hardware components that support some interesting features. Some of the key features of this platform are:

1) TI's DM6437 processor with operating frequency of 600 MHz
2) Ethernet / PCI Bus Interface
3) 128 Mbytes of DDR2 SDRAM
4) 64MB of NAND Flash, 2MB of SRAM, 16 MB of non-volatile Flash memory
5) TVP5146M2 Video Decoder, supports S-video and composite video
6) Configurable BIOS load option
7) JTAG Emulation Interface

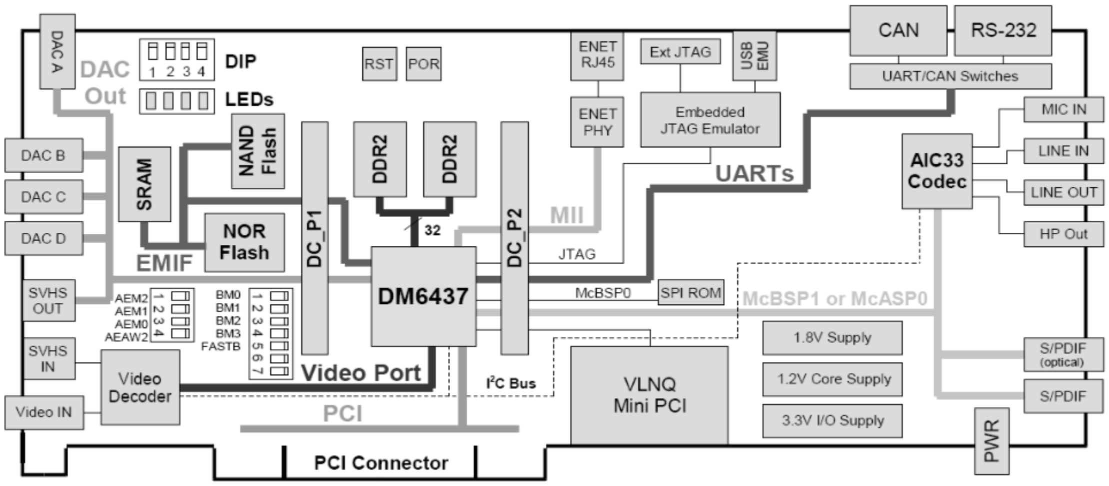


Fig. 3.1 TMS320DM6437 hardware block diagram [3].

3.2 DM6437 Processor

DM6437 is a high performance processor from TI's DaVinci Family that supports clock rates of 400-, 500-, 600-, 660- and 700 MHz. The DSP core contains eight functional units, two general-purpose register files and 2 data paths. Eight functional units can execute eight instructions simultaneously. Each functional unit is assigned a dedicated task of multiplication, arithmetic, logical and branch operation, load data from memory into registers and store data from registers into memory. The two general-purpose register files, namely A and B, contains 32 32-bit registers, providing a total of 64 registers. These registers support data types that include packed 8-bit data, packed 16-bit data, 32-bit data, 40-bit data and 64-bit data values. In case of values exceeding 32-bits, a pair of registers is used to represent 40-bit and 64-bit data values.

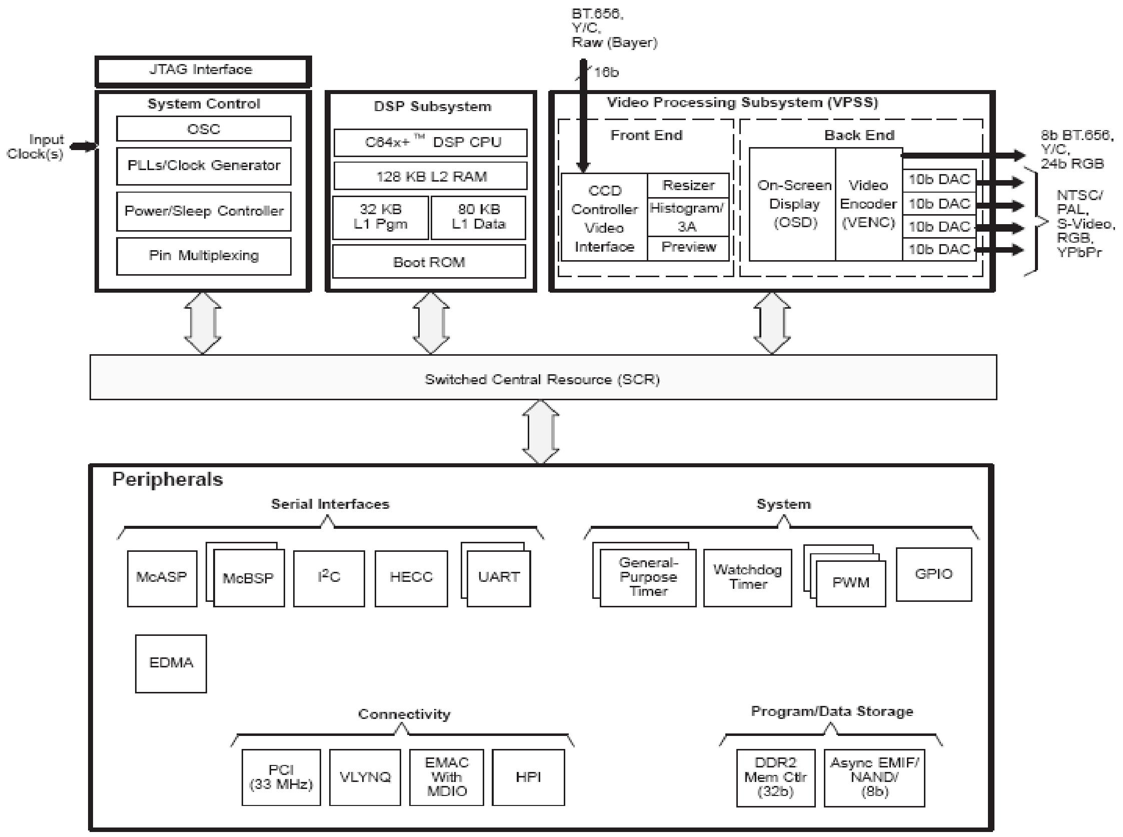


Fig. 3.2 TMS320DM6437 block diagram [4].

3.3 CPU Core

The CPU core consists of Level 1 Program (L1P) cache, Level 1 Data (L1D) cache and Level 2 (L2) Unified cache. L1P cache has a size of 32 Kbytes and can be configured either as a memory-mapped or direct-mapped cache. L1D cache has a size of 80 Kbytes, out of which 48 Kbytes is configured as memory-mapped and 32 Kbytes can be configured either as a memory-mapped or 2-way set associative cache.

L2 cache can be upto 256 Kbytes in size and shared as program as well as data. L2 memory can be configured as a standalone SRAM or as a combination of Cache an SRAM. The size of L2 cache can be varied by making changes in the configuration of system. These changes can be performed in the gel file (evmdm6437.gel) by changing the parameter CACHE_L2CFG. If the value of this parameter is set to 0, it indicates that no L2 cache is configured and whole memory is used as SRAM. By changing the value of CACHE_L2CFG to 1, 2, 3 or 7, one can get L2 cache size of 32KB, 64KB, 128KB or 256KB respectively [5].

3.4 Ethernet and PCI Interface

The Ethernet interface on DM6437 provides an interface between the board and the external network. This interface supports both 10 Mbps and 100 Mbps network connections. The Peripheral Component Interconnect (PCI) provides an interface to connect DM6437 with other PCI-compliant devices. This connection provides an easy way for movement of data from one device to another.

3.5 Video Processing Sub-system (VPSS)

The Video Processing Sub-system consists of a Video Processing Front End (VPFE) and a Video Processing Back End (VPBE). A VPFE provides an interface to connect external input devices such as image censors, video decoders, etc. A VPFE consist of a Closed Circuit Device Controller (CCDC), Preview Engine, Resizer, Hardware 3A (H3A) Static Generator and Histogram blocks. A CCDC provides an interface to connect image sensors and video decoders. Preview Engine converts a Bayer Pattern input into YUV 4:2:2 format. The Resizer block resizes the input image to a desired resolution. H3A block provides support for Auto Focus (AF), Auto Exposure (AE) and Auto White Balance (AWB).

A VPBE provides an interface to connect external devices for displaying output such as LCD monitors, TV displays, etc. It consists of an On-screen Display and a Video Encoder. A Video Encoder consists of a Digital LCD interface and an Analog interface. Analog interface provides analog video output. Digital LCD interface provides RGB or YCbCr output.

3.6 On-board Memory

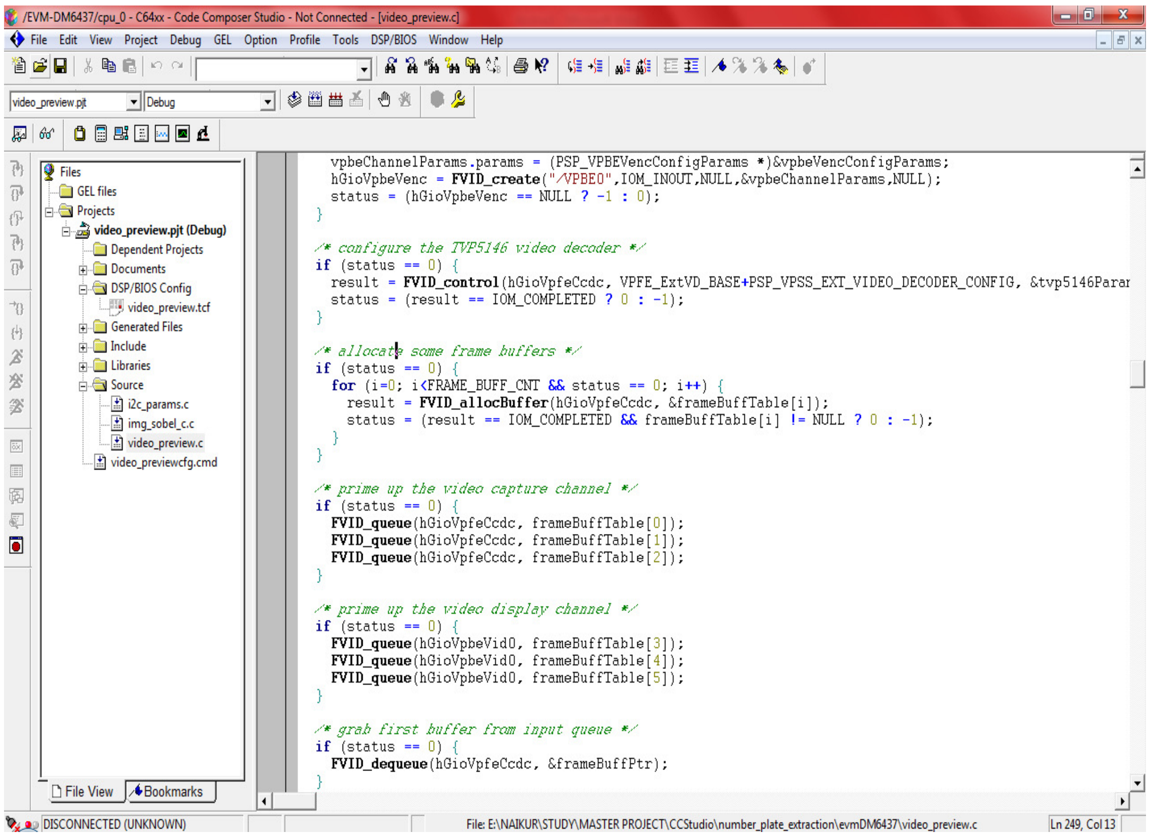
The TMS320DM6437 consist of a 128 Mbytes of DDR2 SDRAM. The SDRAM is used for storage of program, video or data. Also, the board contains16 Mbytes NOR Flash, 2 Mbytes SRAM and 64 Mbytes NAND Flash. NAND and NOR Flash are used mainly as a boot-loaders, while SRAM is mainly used for debugging application code.

3.7 Code Composer Studio

Code Composer Studio (CCS) is an Integrated Development Environment (IDE) used to develop applications targeted to Texas Instrument’s Digital Signal Processors. CCS provides a single IDE to develop an application by offering following features:

- Programming DSP using C/C++
- Ready-to-use built-in functions for video and image processing
- Run-time debugging on the hardware
- Debugging an application using software breakpoints

Some of the steps involved in developing a successful application include creation of a project environment, development of code using C or C++, linking appropriate library functions, compiling the code, converting it into an assembly language code and downloading it onto the DM6437 platform using JTAG interface. The image of the IDE for CCS is shown below:


/EVM-DM6437/cpu_0 - C64xx - Code Composer Studio - Not Connected - [video_preview.c]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
video_preview.pjt
Debug
Files
GEL files
Projects
video_preview.pjt (Debug)
Dependent Projects
Documents
DSP/BIOS Config
video_preview.tcf
Generated Files
Include
Libraries
Source
i2c_params.c
img_sobel_c.c
video_preview.c
video_previewcfg.cmd
File View
Bookmarks
DISCONNECTED (UNKNOWN)
File: E:\NAIKUR\STUDY\MASTER PROJECT\CCStudio\number_plate_extraction\evmDM6437\video_preview.c
Ln 249, Col 13

```
    vpbeChannelParams.params = (PSP_VPBEVencConfigParams *)&vpbeVencConfigParams;
    hGioVpbeVenc = FVID_create("/VPBE0",IOM_INOUT,NULL,&vpbeChannelParams,NULL);
    status = (hGioVpbeVenc == NULL ? -1 : 0);
  }

  /* configure the TVP5146 video decoder */
  if (status == 0) {
    result = FVID_control(hGioVpfeCcdc, VPFE_ExtVD_BASE+PSP_VPSS_EXT_VIDEO_DECODER_CONFIG, &tvp5146Parar
    status = (result == IOM_COMPLETED ? 0 : -1);
  }

  /* allocate some frame buffers */
  if (status == 0) {
    for (i=0; i<FRAME_BUFF_CNT && status == 0; i++) {
      result = FVID_allocBuffer(hGioVpfeCcdc, &frameBuffTable[i]);
      status = (result == IOM_COMPLETED && frameBuffTable[i] != NULL ? 0 : -1);
    }
  }

  /* prime up the video capture channel */
  if (status == 0) {
    FVID_queue(hGioVpfeCcdc, frameBuffTable[0]);
    FVID_queue(hGioVpfeCcdc, frameBuffTable[1]);
    FVID_queue(hGioVpfeCcdc, frameBuffTable[2]);
  }

  /* prime up the video display channel */
  if (status == 0) {
    FVID_queue(hGioVpbeVid0, frameBuffTable[3]);
    FVID_queue(hGioVpbeVid0, frameBuffTable[4]);
    FVID_queue(hGioVpbeVid0, frameBuffTable[5]);
  }

  /* grab first buffer from input queue */
  if (status == 0) {
    FVID_dequeue(hGioVpfeCcdc, &frameBuffPtr);
  }
```

Fig. 3.3 Integrated development environment for CCS.

Combining all the features such as advanced DSP core, interfaces and on-board memory, along with the advantages of CCS, TMS320DM6437 was considered an obvious choice for this project. The DVDP stood out as an excellent platform for this project. The debugging capabilities of CCS were extensively used throughout the development of License Plate Detection algorithm.

Chapter 4

MATLAB IMPLEMENTATION

4.1 Introduction

This chapter describes the implementation of License Plate Detection algorithm using MATLAB. MATLAB is a very powerful software tool used to implement the tasks that require extensive computation. It provides easy and quicker implementation of algorithms compared to C and C++. The key feature in MATLAB is that it contains a rich library functions for image processing and data analysis. This makes MATLAB an ideal tool for faster implementation and verification of any algorithm before actually implementing it on a real hardware. Sometimes, debugging of errors on actual hardware turns out to be a very painful task. MATLAB provides an easy approach for debugging and correction of errors in any algorithm. Other than this, MATLAB contains many features including workspace, plot, imread, imhist, imshow, etc. for data analysis and image processing, which makes it a better choice over other software languages like C and C++.

Considering the above advantages, the writer of this project initially implemented an algorithm for License Plate Detection using MATLAB. The algorithm initially used various inbuilt functions and implemented few user defined routines related to image processing. Once the algorithm was developed, it was verified with multiple input images containing car number plates. The input images contained number plates that were aligned horizontally as well as at some angle from horizontal axis. Once the algorithm was completely verified, the in-built functions of MATLAB were replaced by user-defined functions. A flow-chart showing the basic implementation of algorithm is shown below:

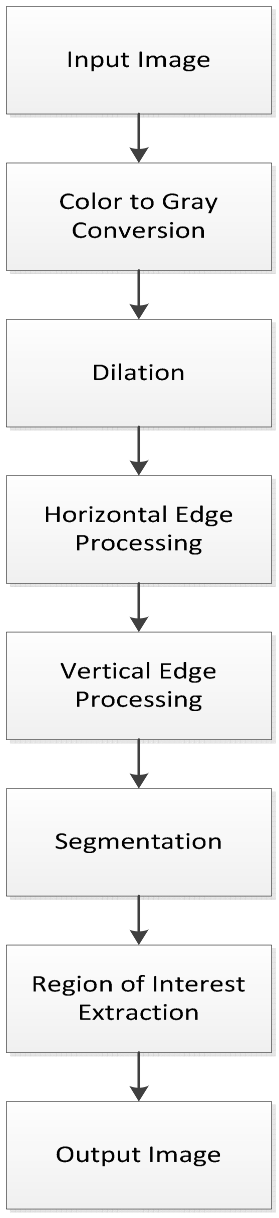


Fig. 4.1 Flowchart showing license plate detection algorithm in MATLAB.

The steps of implementing License Plate Detection algorithm in MATLAB are described below.

4.2 Convert a Colored Image into Gray Image

The algorithm described here is independent of the type of colors in image and relies mainly on the gray level of an image for processing and extracting the required information. Color components like Red, Green and Blue value are not used throughout this algorithm. So, if the input image is a colored image represented by 3-dimensional array in MATLAB, it is converted to a 2-dimensional gray image before further processing. The sample of original input image and a gray image is shown below:

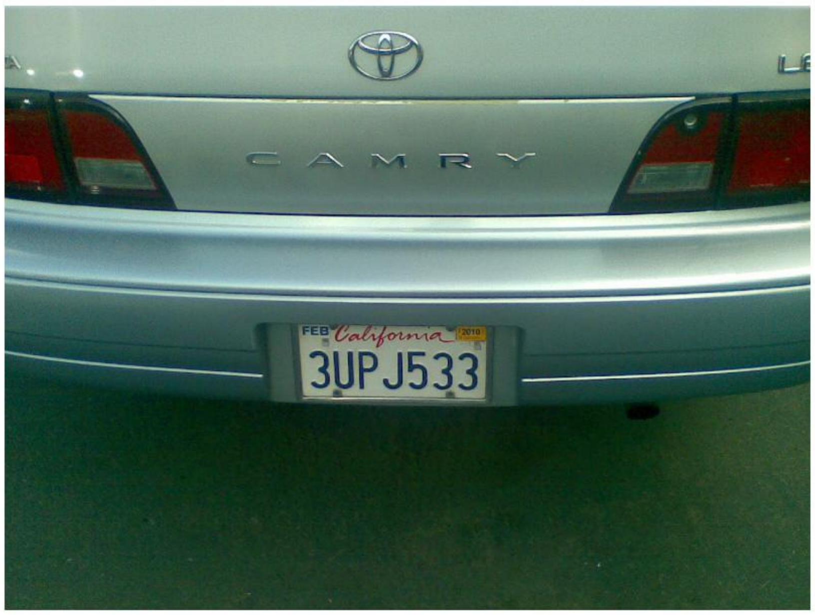


Fig 4.2 Original color image.

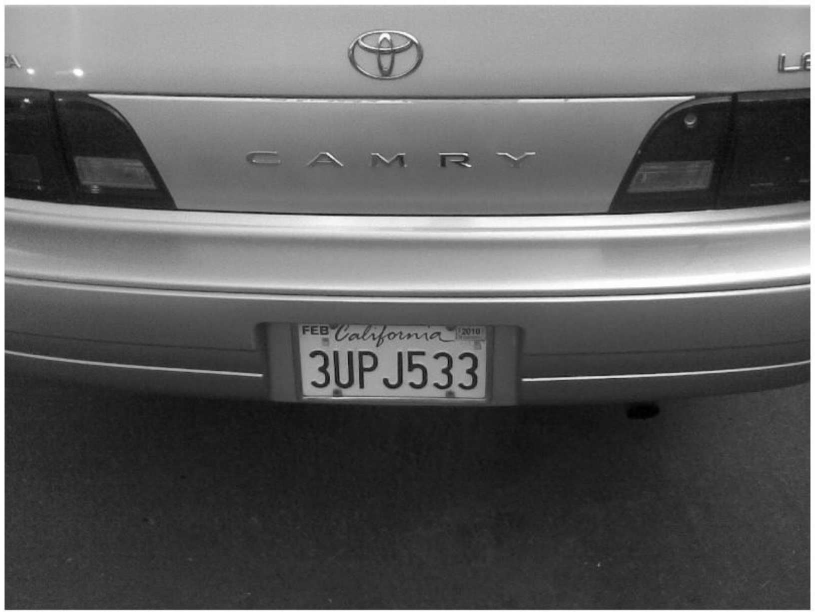


Fig. 4.3 Gray image.

4.3 Dilate an Image

Dilation is a process of improvising given image by filling holes in an image, sharpen the edges of objects in an image, and join the broken lines and increase the brightness of an image. Using dilation, the noise with-in an image can also be removed. By making the edges sharper, the difference of gray value between neighboring pixels at the edge of an object can be increased. This enhances the edge detection.

In Number Plate Detection, the image of a car plate may not always contain the same brightness and shades. Therefore, the given image has to be converted from RGB to gray form. However, during this conversion, certain important parameters like difference in

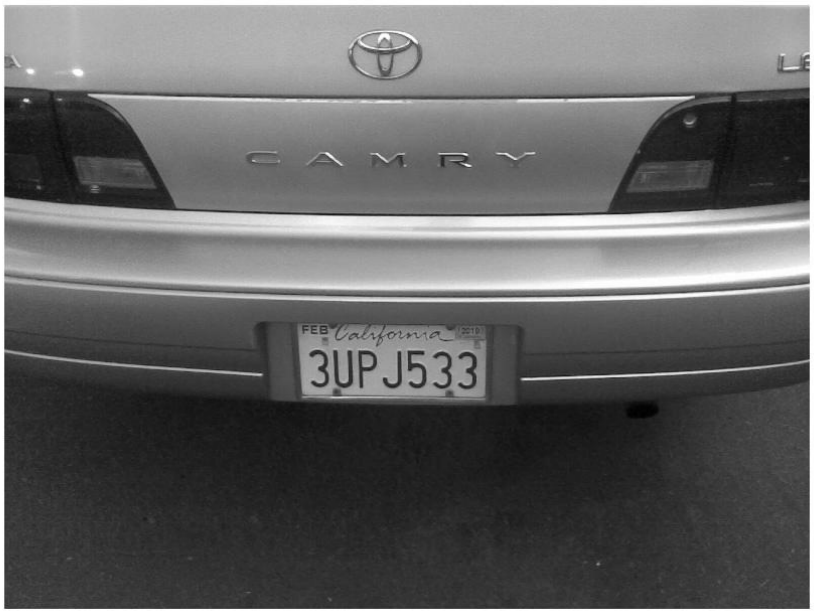


Fig. 4.4 Dilated image.

color, lighter edges of object, etc. may get lost. The process of dilation will help to nullify such losses.

4.4 Horizontal and Vertical Edge Processing of an Image

Histogram is a graph representing the values of a variable quantity over a given range. In this Number Plate Detection algorithm, the writer has used horizontal and vertical histogram, which represents the column-wise and row-wise histogram respectively. These histograms represent the sum of differences of gray values between neighboring pixels of an image, column-wise and row-wise.

In the above step, first the horizontal histogram is calculated. To find a horizontal histogram, the algorithm traverses through each column of an image. In each column, the algorithm starts with the second pixel from the top. The difference between second and first pixel is calculated. If the difference exceeds certain threshold, it is added to total sum of differences. Then, algorithm will move downwards to calculate the difference between the third and second pixels. So on, it moves until the end of a column and calculate the total sum of differences between neighboring pixels. At the end, an array containing the column-wise sum is created. The same process is carried out to find the vertical histogram. In this case, rows are processed instead of columns.

4.5 Passing Histograms through a Low Pass Digital Filter

Referring to the figures shown below**,** one can see that the histogram values changes drastically between consecutive columns and rows. Therefore, to prevent loss of important information in upcoming steps, it is advisable to smooth out such drastic changes in values of histogram. For the same, the histogram is passed through a low-pass digital filter. While performing this step, each histogram value is averaged out considering the values on it right-hand side and left-hand side. This step is performed on both the horizontal histogram as well as the vertical histogram. Below are the figures showing the histogram before passing through a low-pass digital filter and after passing through a low-pass digital filter.

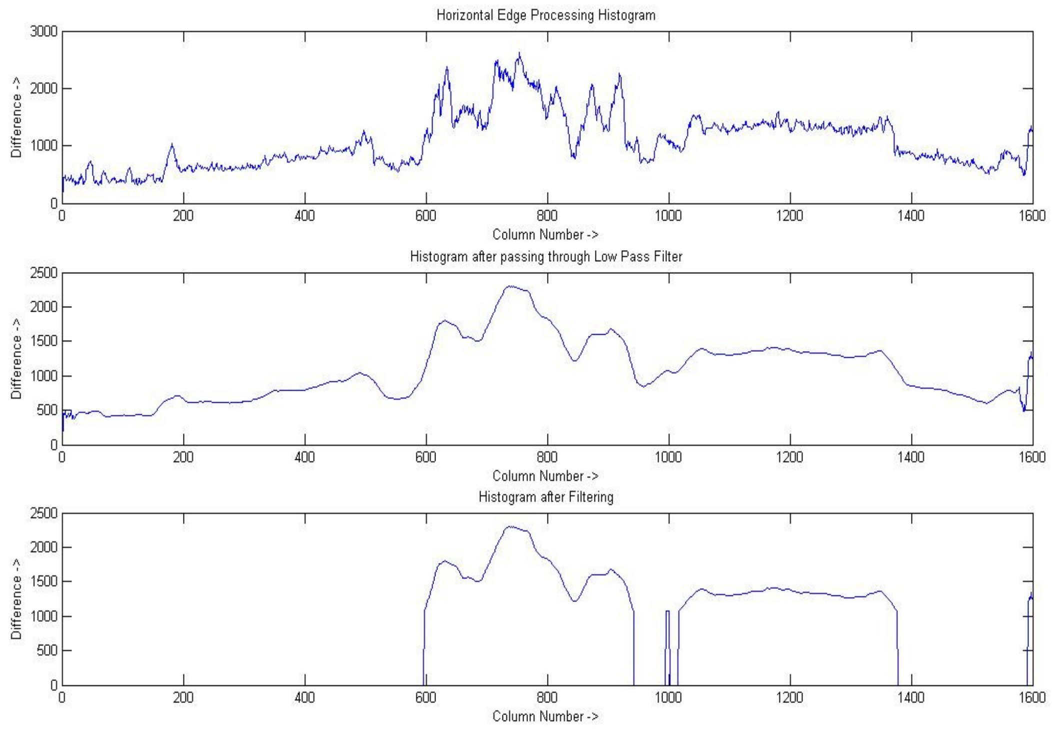


Fig. 4.5 Horizontal edge processing histogram.

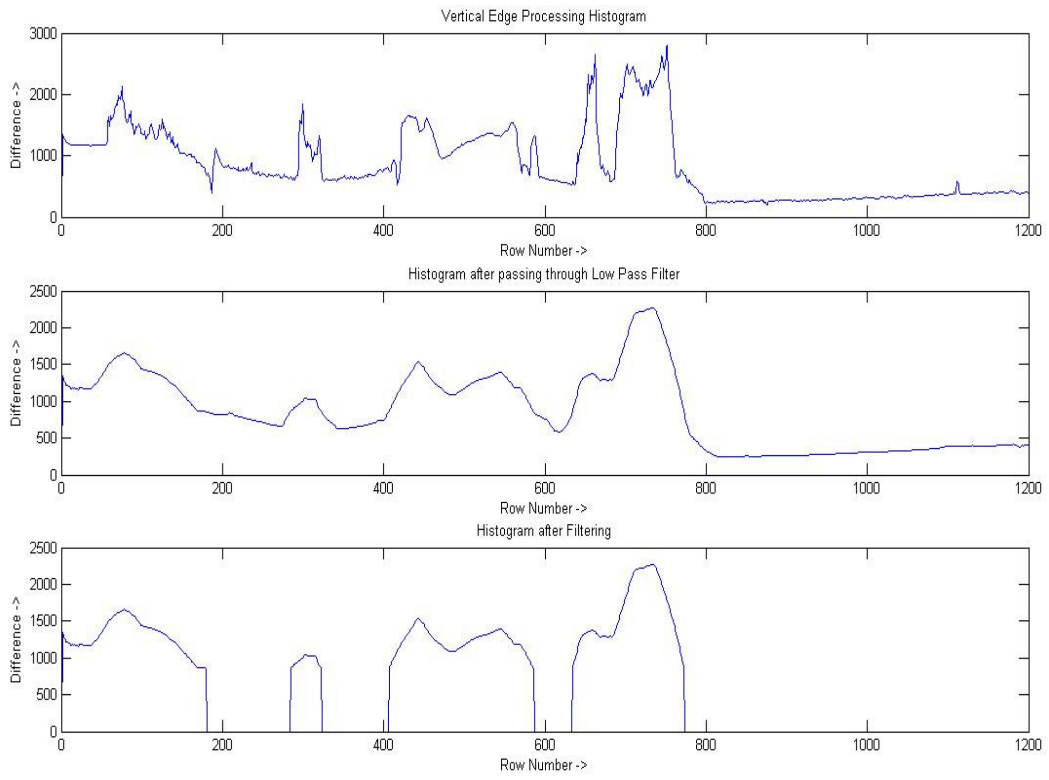


Fig. 4.6 Vertical edge processing histogram.

4.6 Filtering out Unwanted Regions in an Image

Once the histograms are passed through a low-pass digital filter, a filter is applied to remove unwanted areas from an image. In this case, the unwanted areas are the rows and columns with low histogram values. A low histogram value indicates that the part of image contains very little variations among neighboring pixels. Since a region with a license plate contains a plain background with alphanumeric characters in it, the difference in the neighboring pixels, especially at the edges of characters and number plate, will be very high. This results in a high histogram value for such part of an image.

Therefore, a region with probable license plate has a high horizontal and vertical histogram values. Areas with less value are thus not required anymore. Such areas are removed from an image by applying a dynamic threshold.

In this algorithm, the dynamic threshold is equal to the average value of a histogram. Both horizontal and vertical histograms are passed through a filter with this dynamic threshold. The output of this process is histogram showing regions having high probability of containing a number plate. The filtered histograms are shown below:

4.7 Segmentation

The next step is to find all the regions in an image that has high probability of containing a license plate. Co-ordinates of all such probable regions are stored in an array. The output image displaying the probable license plate regions is shown below.

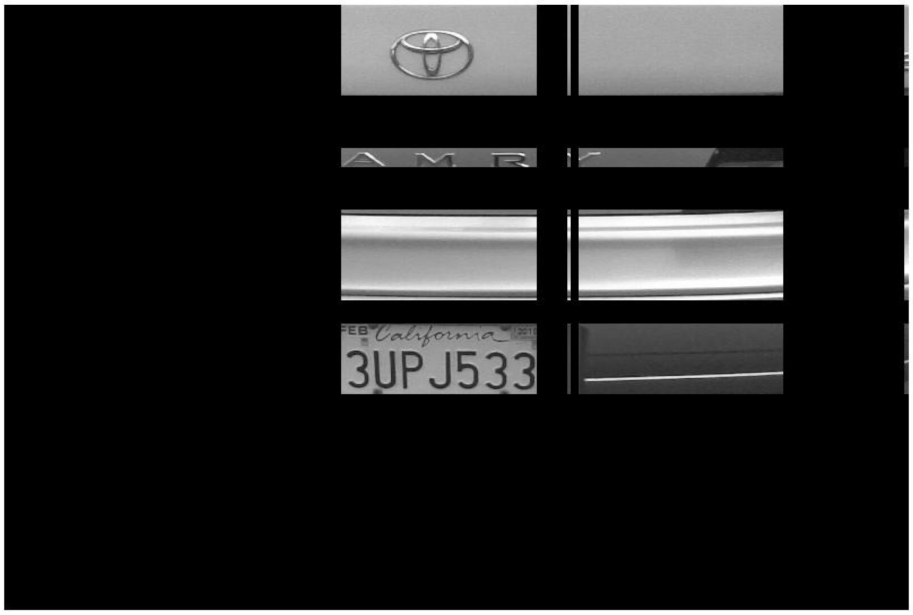


Fig. 4.7 Output of segmentation.

4.8 Region of Interest Extraction

The output of segmentation process is all the regions that have maximum probability of containing a license plate. Out of these regions, the one with the maximum histogram value is considered as the most probable candidate for number plate. All the regions are processed row-wise and column-wise to find a common region having maximum horizontal and vertical histogram value. This is the region having highest probability of containing a license plate. The image detected license plate is shown below:

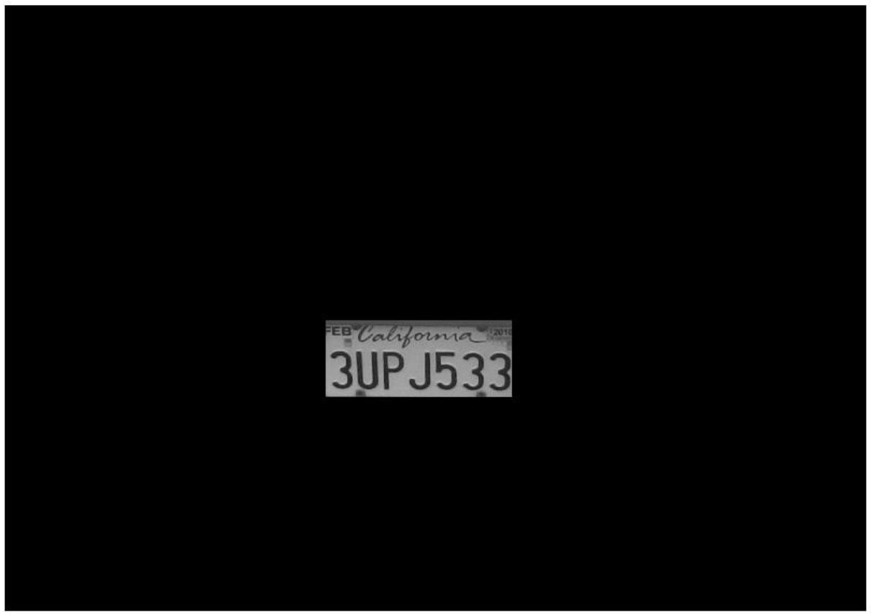


Fig. 4.8 Detected license plate.

This algorithm was verified using several input images having resolution varying from 680 * 480 to 1600 * 1200. The images contained vehicles of different colors and varying intensity of light. With all such images, the algorithm correctly recognized the number plate. This algorithm was also tried on images having number plate aligned at certain angle (approximately 8-10 degree) to horizontal axis. Even with such images, the number plates were detected successfully. After successfully implementing and verifying the algorithm in MATLAB, it was coded in C for implementation on actual hardware. The details about the same are covered in next chapter.

Chapter 5

IMPLEMENTATION OF LICENSE PLATE DETECTION ON EVM320DM6437

In this chapter, the author describes the steps to implement the License Plate Detection algorithm on an EVM320DM6437 hardware kit containing DaVinci DSP Processor. The basic features of the kit are explained in detail in chapter three.

The detection algorithm implemented in this project consists of steps like capturing a frame, extracting Y component, removing of noise, dilation, processing of image and region of interest extraction. A given image consists of a luminance (Y) and a chrominance (C) component. The luminance component mainly represents the gray scale of a pixel. This information is essential in further processing using this algorithm. The chrominance component is not essential, as the algorithm is independent of such information. The Y component is extracted from the frame and stored in a two-dimensional array.

After this, the Y components are processed for noise removal. In this project, a linear filter is applied for noise removal. Each pixel in an image is set to the average value of its neighboring eight pixels. Thus, any spontaneous noise component in the captured frame is removed. This helps in reducing any false processing of the frame. Once the noise is removed, the frame is dilated. In the process of dilation, each pixel is set to a value equal to the maximum value of the neighboring pixel. Dilation mainly helps in sharpening the edges in an image and connects the broken lines in an image. Thus, the quality of captured image is improved.

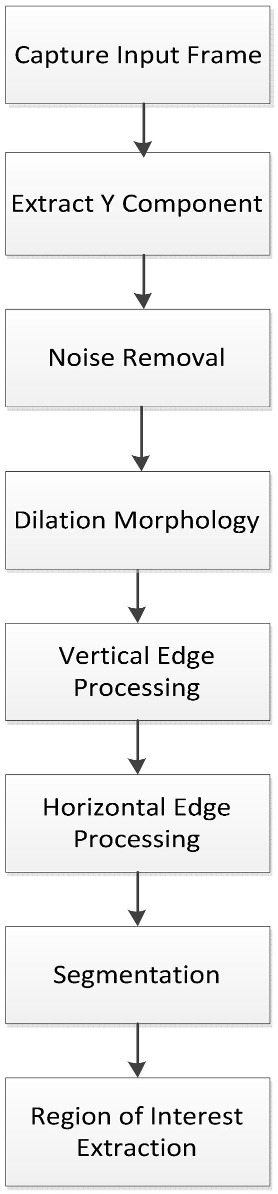


Fig. 5.1 Flowchart showing license plate detection algorithm on EVM320DM6437.

After improving the quality of an image, the next step is to process the image for finding the edges. The image is processed row-wise and column wise to check for edges.

Based on the difference of values between neighboring pixels, histograms are created. These histograms represent the sum of differences between pixel values, both row-wise and column-wise. The histograms are then smoothened by passing it through a low-pass filter. After smoothening, the histogram is further passed through a band pass filter to filter out the unnecessary values in histograms. The threshold used for the filter is a dynamic threshold, which is equal to the average of all the histogram values, for rows and for columns. Applying dynamic threshold has an advantage that it makes algorithm independent of the light conditions and background color for an image.

Next step in the process is Segmentation of an image. In this step, all the unnecessary regions from the image are removed. A list of regions having highest probability of containing a license plate is made. All these regions are then processed to find a single segment having highest probability of containing a number plate. Thus, this is the overall algorithm implemented for license plate detection. The detailed explanation of implementation on EVM320DM6437 is explained below.

5.1 Capture Input Frame

The first step in this process is to capture a video frame input through a camera. The camera provides a composite video output, which serves as an input to EVM320DM6437 kit. The video format chosen in this project is NTSC. Therefore, each frame consists of 720 * 480 pixels. The frame is stored in on-board SRAM by the Video Processing Front End (VPFE). VPFE returns a frame buffer pointer, which can be used to access the stored frame. In this project, the frames are processed one after the other. So, only one frame is stored in memory at a given time.

5.2 Extract Y Component

This algorithm uses the Y component of an image for plate detection. For that, the Y component has to be extracted from an image to make processing easier. Now, inside the kit, the frame is stored in format shown below:

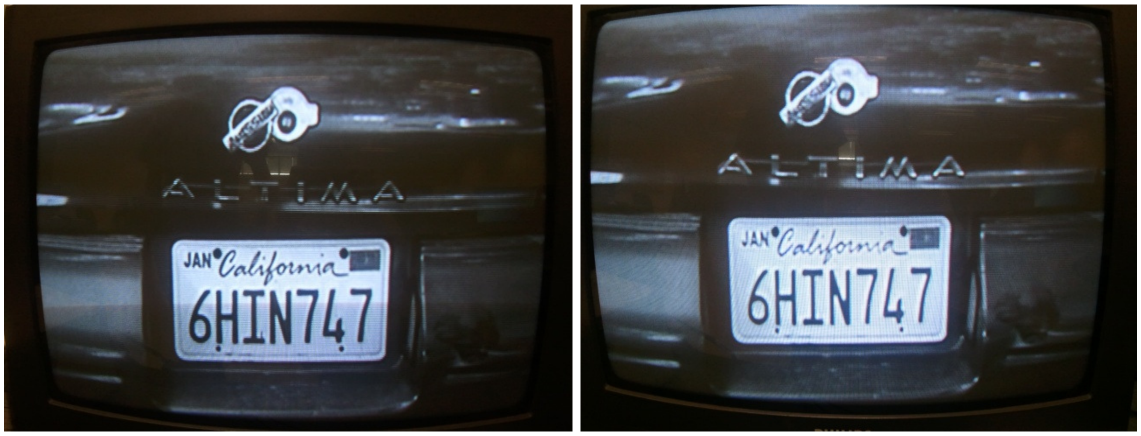
Cb	0
Y	1
Cr	2
Y	3
Cb	4
Y	5
:	:
Cr	:
Y	:

Fig. 5.2 Format for frame storage in EVM320DM6437.

Therefore, the Y component has to be extracted from every alternate location and stored in a new location, which will be used for further processing.

5.3 Noise Removal

In this step, the input image is passed through a linear filter for noise removal. An image is passed through a 9 * 9 mask. Therefore, value of each pixel is set equal to the average value of its 8 neighboring pixels. This type of filter for noise removal is called linear smoothening filter. Such a filter is very effective in terms of performance and speed. Thus, this filter was selected for implementation on actual hardware.



(a) (b)

Fig. 5.3 (a) Image after noise removal (b) Image after dilation.

5.4 Dilation

In order to join broken edges and remove noisy pixels, dilation is implemented on the image after noise removal. In this step, each pixel is compared with its neighboring pixels and its value is set equal to the maximum value out of both the neighboring pixels. This process makes edges of an image sharper. In turn, it helps in better detection of an image. If an input image is blurred, this step will help to improve such blurred image and make it easy for detection.

5.5 Vertical Edge Processing

After removing noise and passing an image through dilation, the next step is to process an image for license plate detection. This is done in two steps, processing image column-wise and row-wise. In case of vertical edge processing, the image is processed column-wise. Each of the 720 columns is traversed one after another. A histogram is prepared based on this processing. This histogram is stored in an array of 720 elements. The histogram generated, will have some very drastic changes due to presence of noise and disturbance in an image. To remove such unnecessary disturbance, the histogram is passed through a low-pass filter to smoothen out the changes in histogram values. To implement this step, each histogram value is set to a value equal to the average of previous fifteen histogram values and next fifteen histogram values. After performing this step, the histogram is passed through a band-pass filter. In this case, a dynamic threshold is applied and all the values less than this threshold are set to 0. This will remove the unnecessary columns from an image.

5.6 Horizontal Edge Processing

This step is similar to the previous step except some of the changes. This step is performed on an image one row after another. A total of 480 histogram values will be obtained and stored in an array. The array will be passed through a low-pass smoothening filter and then through the band-pass filter. The result will be an array of histogram with necessary rows only.

5.7 Segmentation

Once the processing of an image is completed, the next step is segmentation. On this hardware, the segmentation is performed by observing the values of filtered histograms. A set of row and column numbers having highest probability of containing a number plate are prepared. These set of values are passed to the next step known as region of interest extraction.

5.8 Region of Interest Extraction

For this project, the region of interest in any given image is the license plate. This region is found by applying a concept that for a given image, the region containing a number plate will have maximum number of edges compared to any other part in an image. Applying this concept to all the extracted segments, the co-ordinates of the required region are extracted.

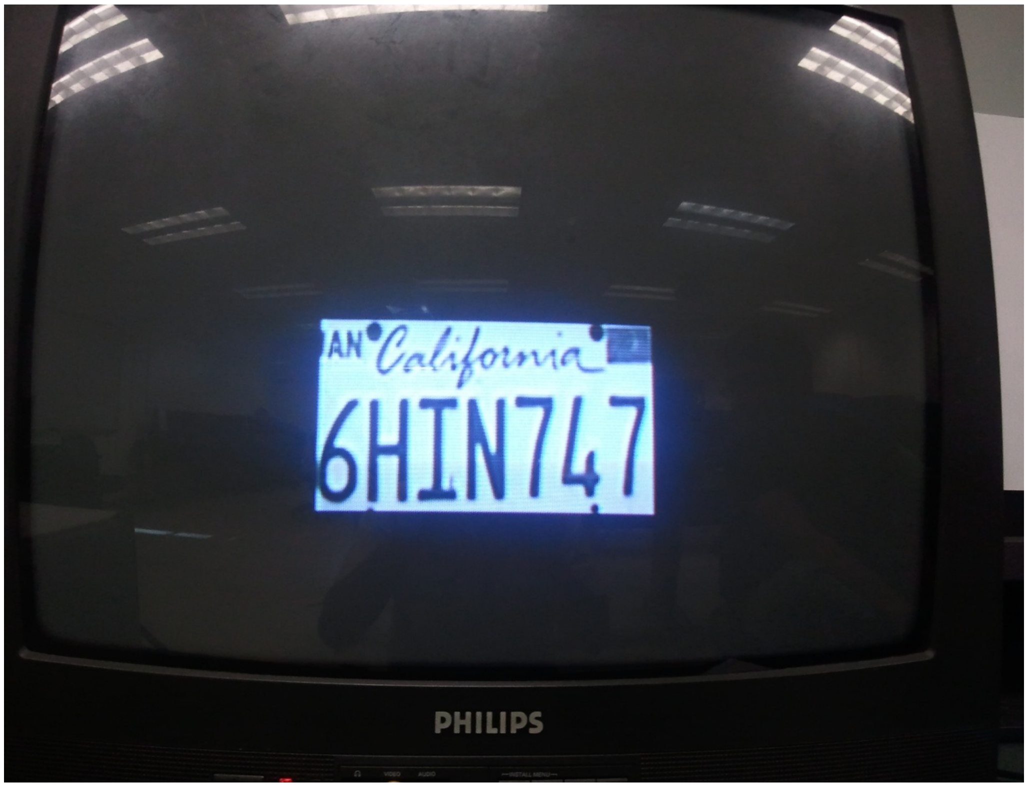


Fig. 5.4 License plate extracted from original image.

These co-ordinate values are then used to extract the license plate and pass it to the next step of License Plate Detection. For more information about recognizing the license plate using EVE320DM6437, the readers are suggested to read the project on Car License Plate Recognition [6].

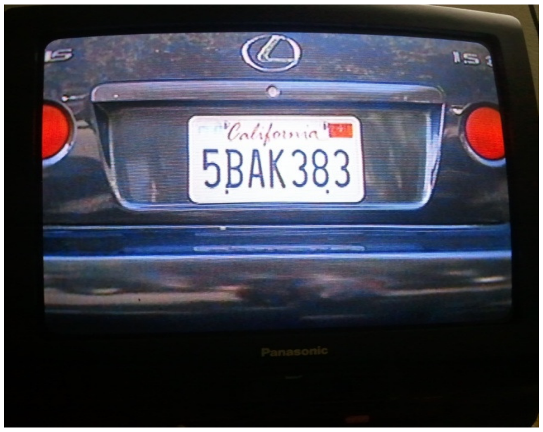
Chapter 6

RESULTS AND PERFORMANCE ANALYSIS

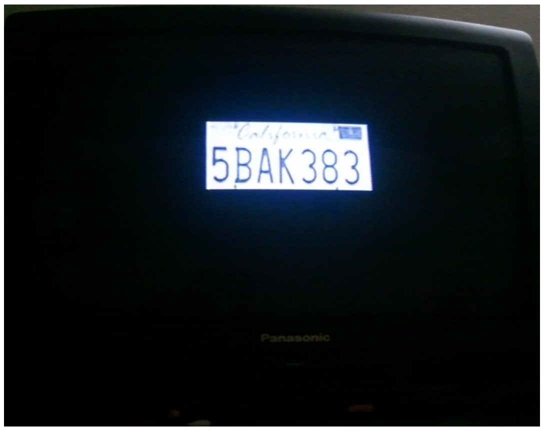
This chapter contains the output results of Car License Plate Detection algorithm implementation on EVM320DM6437, advantages and disadvantages of the applied algorithm and future scope of improvements. The License Plate Detection algorithm is successfully implemented on EVM320DM6437 hardware using Code Composer Studio and C language. Performance analysis is done by implementing various optimization techniques described later.

6.1 Results

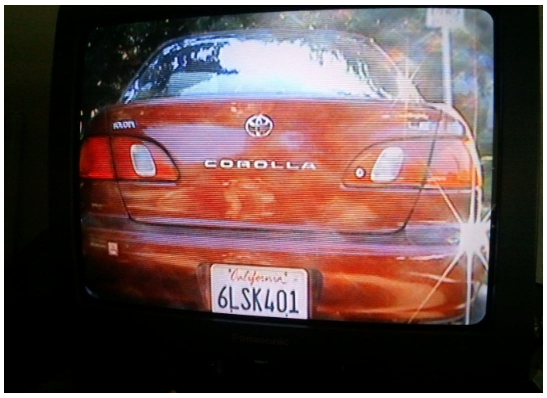
The algorithm was tested using different license plates having various background conditions, light condition and image quality. Some of the output results are shown below:

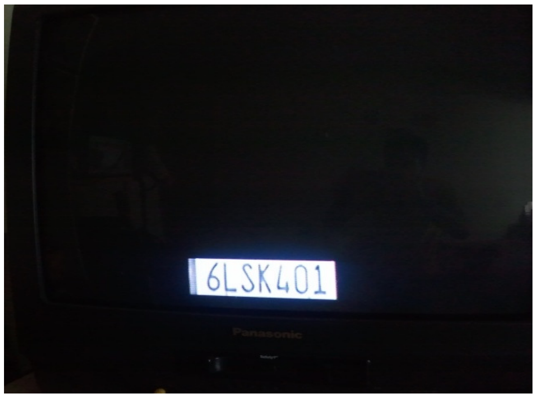


(a)



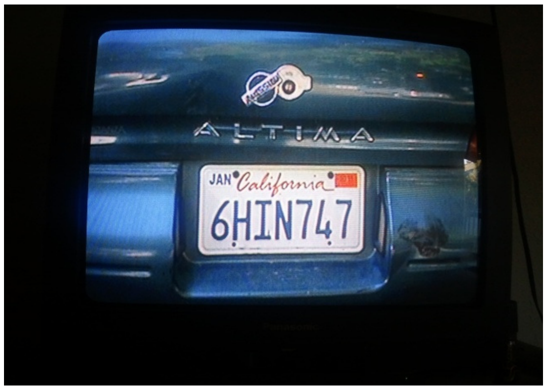
(b)

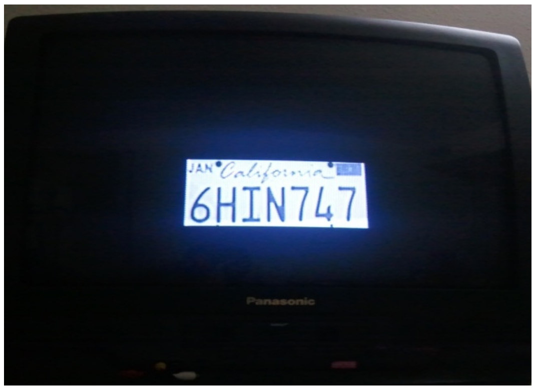




(c)

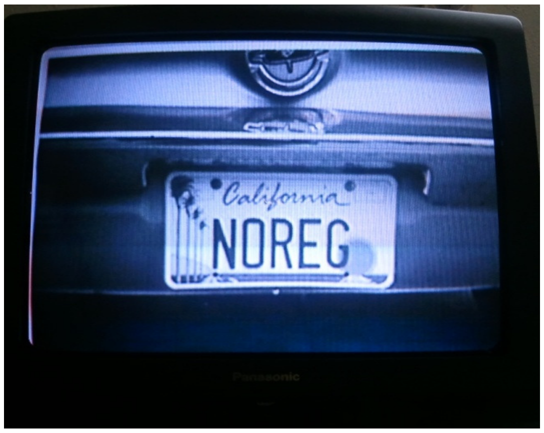
(d)

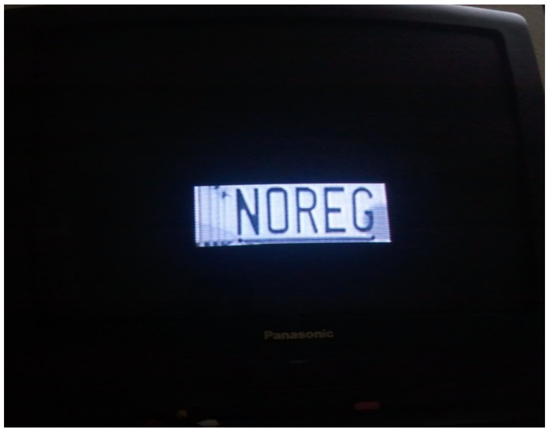




(e)

(f)





(g)

(h)

Fig 6.1 (a)(c)(e)(g) Images of actual cars (b)(d)(f)(h) Images of detected license plates.

The table below shows the type and number of plates and success ratio for detection of plates.

License Plate Conditions	Success using MATLAB Implementation (%)	Success using Hardware Implementation (%)
Sunlight	100	94
Cloudy weather	100	92
Shade	100	94
Different Backgrounds	100	95

Table 6.1 Performance of License Plate Detection under Various Conditions

The Matlab implementation helped to verify the algorithm very efficiently. Due to certain limitations of hardware like input image from camera and processing overhead on kit, the success ratio for the algorithm on hardware is slightly less compared to the implementation in Matlab.

Function	No. of CPU Cycles (Without Optimization)	Time in msec. (Clock Frequency is 600 MHz)
Noise Removal	18651800	31.08
Dilation Morphology	40882061	68.14
Vertical Edge Proecessing	46889824	78.15
Horizontal Edge Processing	25067790	41.77
Region of Interest Extraction	95500	0.16

Table 6.2 Performance without Optimization

A summary of performance analysis for various functions without any optimization is shown above. For a processor working on the clock frequency of 600 MHz, the above table shows that the time taken by hardware to detect a number plate is nearly 219.3 milliseconds. For this algorithm, the input is a fixed sized image of resolution 720 * 480. The image can contain license plate of varying size. So, the total time taken to execute this algorithm depends greatly upon the size of the license plate. If the license plate is bigger, the total execution time increases, while if license plate is smaller, the total execution time decreases comparatively. This algorithm can work with varying size of license plate.

6.2 Optimization

The initial C code contained a number of un-optimized conditional loops like for and if-else. The compiler optimization techniques were also tuned for best performance. Moreover, L2 cache was enabled and disabled to check for performance improvement. In order to improve the total execution time, various optimization techniques were used. These optimization techniques include:

a) Code Optimization

b) Compiler Optimization

c) Cache Optimization

The C code developed for this algorithm was optimized for better performance using various techniques. The number of branch statements generated by the assembly code was targeted. By combining for loops, which perform operations on same data set, the total number of branches was reduced. Conditional statements like if and else were combined together for better performance.

Code Composer Studio provides different choices for generating optimized assembly code. These optimization criteria provide trade-off between speed of execution and code size. In this project, speed of execution was considered as an important optimization criteria compared to code size. The table provided below shows the improved performance after code and compiler optimization.

Function	No. of CPU Cycles (With Compiler Optimization)	Time in msec (Clock Frequency is 600 MHz)
Noise Removal	10169661	16.98
Dilation Morphology	8965617	14.97
Vertical Edge Processing	35768225	59.73
Horizontal Edge Processing	12275745	20.50
Region of Interest Extraction	65759	0.11

Table 6.3 Performance with Optimization

From the above table, it can be seen that the total time taken to detect a number plate is nearly 112.3 milliseconds. So, code and compiler optimization provided upto 48.79% improvement in performance on hardware. Apart from these optimization techniques, cache size was also increased to check for improved performance. The EVM320DM6437 kit provides a feature to change the size of L2 cache from 0 KB to 256 KB. This change can be made by changing the parameter CACHE_L2CFG parameter in the gel file named evmdm6437.gel. Effect of changing cache size greatly depends upon how algorithm accesses data. The performance before and after adding L2 cache of 256 KB is shown in a table below.

Function	No. of CPU Cycles (Without L2 Cache)	No. of CPU Cycles (With L2 Cache)
Noise Removal	18651800	18652256
Dilation Morphology	40882061	40881228
Vertical Edge Processing	46889824	46879591
Horizontal Edge Processing	25067790	25060517
Region of Interest Extraction	95500	95276

Table 6.4 Performance without L2 Cache and with L2 Cache

Performance improvement of a code without and with L2 cache depends upon how the data is accessed by the algorithm. Depending upon temporal and spatial locality, performance with L2 cache can be improved.

This chapter explained about the results achieved using the developed algorithm for License Plate Detection. The detection algorithm is implemented on EVM320DM6437 and is optimized for better performance. The analysis above shows the performance improvement achieved by optimization process. This project can be used further as a base to develop systems, which can be interfaced with a database containing information about car owners and license plate. This way, a real-time system can be developed for automating the process of toll collection, traffic management, law enforcement, vehicle parking, etc.

APPENDIX

Simulation code and hardware implementation code

1. MATLAB Simulation Code:

```
clc;    % Clear command window.
clear all;  % Delete all variables.
close all;  % Close all figure windows except those created by imtool.
imtool close all;   % Close all figure windows created by imtool.
workspace;  % Make sure the workspace panel is showing.

% Read Image
I = imread ('car2_image.jpg');

figure(1);
imshow(I);

% Extract Y component (Convert an Image to Gray)
Igray = rgb2gray(I);

[rows cols] = size(Igray);

%% Dilate and Erode Image in order to remove noise
Idilate = Igray;
for i = 1:rows
   for j = 2:cols-1
      temp = max(Igray(i,j-1), Igray(i,j));
      Idilate(i,j) = max(temp, Igray(i,j+1));
   end
end

I = Idilate;
figure(2);
imshow(Igray);
figure(3);
title('Dilated Image')
imshow(Idilate);

figure(4);
imshow(I);

difference = 0;
```

```
sum = 0;
total_sum = 0;
difference = uint32(difference);

%% PROCESS EDGES IN HORIZONTAL DIRECTION
disp('Processing Edges Horizontally...');
max_horz = 0;
maximum = 0;
for i = 2:cols
    sum = 0;
    for j = 2:rows
        if(I(j, i) > I(j-1, i))
            difference = uint32(I(j, i) - I(j-1, i));
        else
            difference = uint32(I(j-1, i) - I(j, i));
        end

        if(difference > 20)
            sum = sum + difference;
        end
    end
    horz1(i) = sum;

    % Find Peak Value
    if(sum > maximum)
        max_horz = i;
        maximum = sum;
    end

    total_sum = total_sum + sum;
end
average = total_sum / cols;

figure(5);
% Plot the Histogram for analysis
subplot(3,1,1);
plot (horz1);
title('Horizontal Edge Processing Histogram');
xlabel('Column Number ->');
ylabel('Difference ->');

%% Smoothen the Horizontal Histogram by applying Low Pass Filter
disp('Passing Horizontal Histogram through Low Pass Filter...');
```

```
sum = 0;
horz = horz1;
for i = 21:(cols-21)
    sum = 0;
    for j = (i-20):(i+20)
        sum = sum + horz1(j);
    end
    horz(i) = sum / 41;
end

subplot(3,1,2);
plot (horz);
title('Histogram after passing through Low Pass Filter');
xlabel('Column Number ->');
ylabel('Difference ->');

%% Filter out Horizontal Histogram Values by applying Dynamic Threshold
disp('Filter out Horizontal Histogram...');
for i = 1:cols
    if(horz(i) < average)
        horz(i) = 0;
        for j = 1:rows
            I(j, i) = 0;
        end
    end
end

subplot(3,1,3);
plot (horz);
title('Histogram after Filtering');
xlabel('Column Number ->');
ylabel('Difference ->');

%% PROCESS EDGES IN VERTICAL DIRECTION
difference = 0;
total_sum = 0;
difference = uint32(difference);

disp('Processing Edges Vertically...');
maximum = 0;
max_vert = 0;
for i = 2:rows
    sum = 0;
    for j = 2:cols          %cols
        if(I(i, j) > I(i, j-1))
            difference = uint32(I(i, j) - I(i, j-1));
        end
        if(I(i, j) <= I(i, j-1))
           difference = uint32(I(i, j-1) - I(i, j));
        end

        if(difference > 20)
            sum = sum + difference;
        end
    end
    vert1(i) = sum;

    %% Find Peak in Vertical Histogram
    if(sum > maximum)
        max_vert = i;
        maximum = sum;
    end
    total_sum = total_sum + sum;
end
average = total_sum / rows;

figure(6)
subplot(3,1,1);
plot (vert1);
title('Vertical Edge Processing Histogram');
xlabel('Row Number ->');
ylabel('Difference ->');

%% Smoothen the Vertical Histogram by applying Low Pass Filter
disp('Passing Vertical Histogram through Low Pass Filter...');
sum = 0;
vert = vert1;

for i = 21:(rows-21)
    sum = 0;
    for j = (i-20):(i+20)
        sum = sum + vert1(j);
    end
    vert(i) = sum / 41;
end
```

```
subplot(3,1,2);
plot (vert);
title('Histogram after passing through Low Pass Filter');
xlabel('Row Number ->');
ylabel('Difference ->');

%% Filter out Vertical Histogram Values by applying Dynamic Threshold
disp('Filter out Vertical Histogram...');
for i = 1:rows
    if(vert(i) < average)
        vert(i) = 0;
        for j = 1:cols
            I(i, j) = 0;
        end
    end
end

subplot(3,1,3);
plot (vert);
title('Histogram after Filtering');
xlabel('Row Number ->');
ylabel('Difference ->');

figure(7), imshow(I);

%% Find Probable candidates for Number Plate
j = 1;
for i = 2:cols-2
    if(horz(i) ~= 0 && horz(i-1) == 0 && horz(i+1) == 0)
        column(j) = i;
        column(j+1) = i;
        j = j + 2;
    elseif((horz(i) ~= 0 && horz(i-1) == 0) || (horz(i) ~= 0 && horz(i+1) == 0))
        column(j) = i;
        j = j+1;
    end
end

j = 1;
for i = 2:rows-2
    if(vert(i) ~= 0 && vert(i-1) == 0 && vert(i+1) == 0)
        row(j) = i;
        row(j+1) = i;
        j = j + 2;
    elseif((vert(i) ~= 0 && vert(i-1) == 0) || (vert(i) ~= 0 && vert(i+1) == 0))
        row(j) = i;
        j = j+1;
    end
end

[temp column_size] = size (column);
if(mod(column_size, 2))
    column(column_size+1) = cols;
end

[temp row_size] = size (row);
if(mod(row_size, 2))
    row(row_size+1) = rows;
end

%% Region of Interest Extraction
%Check each probable candidate
for i = 1:2:row_size
     for j = 1:2:column_size

            % If it is not the most probable region remove it from image
            if(~((max_horz >= column(j) && max_horz <= column(j+1)) && (max_vert >=
row(i) && max_vert <= row(i+1))))

                %This loop is only for displaying proper output to User
                for m = row(i):row(i+1)
                    for n = column(j):column(j+1)
                        I(m, n) = 0;
                    end
                end
            end
    end
end

figure(8), imshow(I);
```

2. Hardware implementation code in C:

```
/*
 * ======== video_preview.c ========
 */

/* runtime include files */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <c6x.h>

/* BIOS include files */
#include <std.h>
#include <gio.h>
#include <tsk.h>
#include <trc.h>

/* PSP include files */
#include <psp_i2c.h>
#include <psp_vpfe.h>
#include <psp_vpbe.h>
#include <fvid.h>
#include <psp_tvp5146_extVidDecoder.h>

/* CSL include files */
#include <soc.h>
#include <cslr_sysctl.h>

/* BSL include files */
#include <evmdm6437.h>
#include <evmdm6437_dip.h>

/* Video Params Defaults */
#include <vid_params_default.h>

#include <bcache.h>

// IMAGE PROCESSING HEADER FILES
#include "img_sobel.h"
#include "img_yc_demux_le16.h"
#include "img_erode_bin.h"
```

```
#include "img_dilate_bin.h"

/* This example supports either PAL or NTSC depending on position of JP1 */
#define STANDARD_PAL  0
#define STANDARD_NTSC 1

#define FRAME_BUFF_CNT 6

static int read_JP1(void);

static CSL_SysctlRegsOvly sysModuleRegs = (CSL_SysctlRegsOvly
)CSL_SYS_0_REGS;

//*****************************************************
// USER DEFINED FUNCTIONS
//*****************************************************

void extract_y (void * currentFrame);
void write_y (void * currentFrame);
void noise_removal();
void dilation_morphology();
void vertical_edge_processing();
void horizontal_edge_processing();
void region_of_interest_extraction();

//*****************************************************
// VARIABLE ARRAYS
//*****************************************************

unsigned char I[480][720];
unsigned char I_temp[480][720];

/*
 * ======== main ========
 */
void main() {

   printf("Video Preview Application\n");
   fflush(stdout);

   /* Initialize BSL library to read jumper switches: */
   EVMDM6437_DIP_init();
```

```
    /* VPSS PinMuxing */
    /* CI10SEL   - No CI[1:0]                   */
    /* CI32SEL   - No CI[3:2]                   */
    /* CI54SEL   - No CI[5:4]                   */
    /* CI76SEL   - No CI[7:6]                   */
    /* CFLDSEL   - No C_FIELD                   */
    /* CWENSEL   - No C_WEN                     */
    /* HDVSEL    - CCDC HD and VD enabled       */
    /* CCDCSEL   - CCDC PCLK, YI[7:0] enabled   */
    /* AEAW      - EMIFA full address mode      */
    /* VPBECKEN  - VPBECLK enabled              */
    /* RGBSEL    - No digital outputs           */
    /* CS3SEL    - LCD_OE/EM_CS3 disabled       */
    /* CS4SEL    - CS4/VSYNC enabled            */
    /* CS5SEL    - CS5/HSYNC enabled            */
    /* VENCSEL   - VCLK,YOUT[7:0],COUT[7:0] enabled */
    /* AEM       - 8bEMIF + 8bCCDC + 8 to 16bVENC   */
    sysModuleRegs -> PINMUX0    &= (0x005482A3u);
    sysModuleRegs -> PINMUX0    |= (0x005482A3u);

    /* PCIEN    =   0: PINMUX1 - Bit 0 */
    sysModuleRegs -> PINMUX1 &= (0xFFFFFFFEu);
    sysModuleRegs -> VPSSCLKCTL = (0x18u);

    return;
}

/*
 * ======== video_preview ========
 */
void video_preview(void) {

  FVID_Frame *frameBuffTable[FRAME_BUFF_CNT];
  FVID_Frame *frameBuffPtr;
  GIO_Handle hGioVpfeCcdc;
  GIO_Handle hGioVpbeVid0;
  GIO_Handle hGioVpbeVenc;
  int status = 0;
  int result;
  int i;
  int standard;
  int width;
  int height;
  int flag = 1;
  int start, stop;

  /* Set video display/capture driver params to defaults */
  PSP_VPFE_TVP5146_ConfigParams tvp5146Params =
      VID_PARAMS_TVP5146_DEFAULT;
  PSP_VPFECcdcConfigParams      vpfeCcdcConfigParams =
      VID_PARAMS_CCDC_DEFAULT_D1;
  PSP_VPBEOsdConfigParams vpbeOsdConfigParams =
      VID_PARAMS_OSD_DEFAULT_D1;
  PSP_VPBEVencConfigParams vpbeVencConfigParams;

  standard = read_JP1();

  /* Update display/capture params based on video standard (PAL/NTSC) */
  if (standard == STANDARD_PAL)  {
      width  = 720;
      height = 576;
      vpbeVencConfigParams.displayStandard =
PSP_VPBE_DISPLAY_PAL_INTERLACED_COMPOSITE;
  }
  else {
      width  = 720;
      height = 480;
      vpbeVencConfigParams.displayStandard =
PSP_VPBE_DISPLAY_NTSC_INTERLACED_COMPOSITE;
  }
  vpfeCcdcConfigParams.height = vpbeOsdConfigParams.height = height;
  vpfeCcdcConfigParams.width = vpbeOsdConfigParams.width = width;
  vpfeCcdcConfigParams.pitch = vpbeOsdConfigParams.pitch = width * 2;

  /* init the frame buffer table */
  for (i=0; i<FRAME_BUFF_CNT; i++) {
    frameBuffTable[i] = NULL;
  }

  /* create video input channel */
  if (status == 0) {
    PSP_VPFEChannelParams vpfeChannelParams;
    vpfeChannelParams.id     = PSP_VPFE_CCDC;
    vpfeChannelParams.params =
(PSP_VPFECcdcConfigParams*)&vpfeCcdcConfigParams;
```

```
    hGioVpfeCcdc =
FVID_create("/VPFE0",IOM_INOUT,NULL,&vpfeChannelParams,NULL);
    status = (hGioVpfeCcdc == NULL ? -1 : 0);
  }

  /* create video output channel, plane 0 */
  if (status == 0) {
    PSP_VPBEChannelParams vpbeChannelParams;
    vpbeChannelParams.id     = PSP_VPBE_VIDEO_0;
    vpbeChannelParams.params =
(PSP_VPBEOsdConfigParams*)&vpbeOsdConfigParams;
    hGioVpbeVid0 =
FVID_create("/VPBE0",IOM_INOUT,NULL,&vpbeChannelParams,NULL);
    status = (hGioVpbeVid0 == NULL ? -1 : 0);
  }

  /* create video output channel, venc */
  if (status == 0) {
    PSP_VPBEChannelParams vpbeChannelParams;
    vpbeChannelParams.id     = PSP_VPBE_VENC;
    vpbeChannelParams.params = (PSP_VPBEVencConfigParams
*)&vpbeVencConfigParams;
    hGioVpbeVenc =
FVID_create("/VPBE0",IOM_INOUT,NULL,&vpbeChannelParams,NULL);
    status = (hGioVpbeVenc == NULL ? -1 : 0);
  }

  /* configure the TVP5146 video decoder */
  if (status == 0) {
    result = FVID_control(hGioVpfeCcdc,
VPFE_ExtVD_BASE+PSP_VPSS_EXT_VIDEO_DECODER_CONFIG,
&tvp5146Params);
    status = (result == IOM_COMPLETED ? 0 : -1);
  }

  /* allocate some frame buffers */
  if (status == 0) {
    for (i=0; i<FRAME_BUFF_CNT && status == 0; i++) {
      result = FVID_allocBuffer(hGioVpfeCcdc, &frameBuffTable[i]);
      status = (result == IOM_COMPLETED && frameBuffTable[i] != NULL ? 0 : -1);
    }
  }
```

```
/* prime up the video capture channel */
if (status == 0) {
  FVID_queue(hGioVpfeCcdc, frameBuffTable[0]);
  FVID_queue(hGioVpfeCcdc, frameBuffTable[1]);
  FVID_queue(hGioVpfeCcdc, frameBuffTable[2]);
}

/* prime up the video display channel */
if (status == 0) {
  FVID_queue(hGioVpbeVid0, frameBuffTable[3]);
  FVID_queue(hGioVpbeVid0, frameBuffTable[4]);
  FVID_queue(hGioVpbeVid0, frameBuffTable[5]);
}

/* grab first buffer from input queue */
if (status == 0) {
  FVID_dequeue(hGioVpfeCcdc, &frameBuffPtr);
}

/* loop forever performing video capture and display */
while ( flag && status == 0 ) {

  /* grab a fresh video input frame */
  FVID_exchange(hGioVpfeCcdc, &frameBuffPtr);

      // Extract Y values from the frame
      extract_y ((frameBuffPtr->frame.frameBufferPtr));
      noise_removal();
      dilation_morphology();
      vertical_edge_processing();
      horizontal_edge_processing();
      region_of_interest_extraction();
      // Put Y values back into Frame
      start = TSCL;
      write_y ((frameBuffPtr->frame.frameBufferPtr));
      stop = TSCL;

      BCACHE_wbInv((void*)(frameBuffPtr->frame.frameBufferPtr), 480*720*2, 1);

      /* display the video frame */
  FVID_exchange(hGioVpbeVid0, &frameBuffPtr);

      printf("\nTOTAL TIME = %d :-)\n", (stop - start));
```

```
        flag = 0;
  }
}

int column_hist[720], row_hist[480];
int max_column_hist, max_row_hist;
int max_column_number, max_row_number;
int column_total_sum, row_total_sum;
int column_hist_average, row_hist_average;
int row1, row2, col1, col2;

void extract_y (void * currentFrame)
{
        int r, c;
        int offset;
        offset = 1;

        for(r = 0; r < 480; r++)
        {
                for(c = 0; c < 720; c++)
                {
                        I_temp[r][c] = * (((unsigned char * )currentFrame) + offset);
                        offset = offset + 2;
                }
        }
}

void write_y (void * currentFrame)
{
        int r, c;
        int offset;
        offset = 1;

        for(r = 0; r < 480; r++)
        {
                for(c = 0; c < 720; c++)
                {
                        if(r > row1 && r < row2 && c > col1 && c < col2)
                                * (((unsigned char * )currentFrame)+ offset) = I[r][c];
                        else
                                * (((unsigned char * )currentFrame)+ offset) = 0;
                        offset++;
                        * (((unsigned char * )currentFrame)+ offset) = 0x80;
                        offset++;
                }
        }
}

//*********************************************
// NOISE REMOVAL USING LINEAR SMOOTHENING FILTER
//*********************************************
void noise_removal()
{
   int r, c;

        for (r = 1; r < (480 - 1); r++)
        {
                for (c = 1; c < (720 - 1); c++)
                {
                        I[r][c] = (I_temp[r-1][c-1] + I_temp[r-1][c] + I_temp[r-1][c+1] +
I_temp[r][c-1] + I_temp[r][c+1] + I_temp[r+1][c-1] + I_temp[r+1][c] +
I_temp[r+1][c+1])/8;
                }
        }
}

//*******************
// DILATION MORPHOLOGY
//*******************
void dilation_morphology()
{
        int r, c;
        unsigned char temp, temp_prev;

        for (r = 1; r < (480-1); r++)
        {
                temp = 0;
                for (c = 1; c < (720 - 1); c++)
                {
                        // Find maximum of the previous, current and next pixel, for dilation
                        temp_prev = temp;
                        temp = (temp > I[r][c+1]) ? ((I[r][c-1] > I[r][c]) ? I[r][c-1] : I[r][c]) :
I[r][c+1];
                        I[r][c-1] = temp_prev;
                }
```

```
	}
}

//******************************
// PROCESS VERTICAL EDGES IN IMAGE
//******************************
void vertical_edge_processing()
{
	int r, c, i, difference;
	int sum = 0;
	max_column_hist = 0;
	column_total_sum = 0;

	for (c = 0; c < 720; c++)
	{
		column_hist[c] = 0;
		sum = 0;
		for (r = 1; r < (480-1); r++)
		{
			if(I[r][c] > I[r-1][c])
			{difference = I[r][c] - I[r-1][c];}
			else
			{difference = I[r-1][c] - I[r][c];}

			if(difference > 30)
			{sum = sum + difference;}
		}

		column_hist[c] = sum;
		column_total_sum = column_total_sum + sum;

		if (c > 10 && c < 710 && max_column_hist < sum)
		{
			max_column_hist = column_hist[c];
			max_column_number = c;
		}
	}

	//*******************************************
	// PASSING VERTICAL EDGE THROUGH LOW-PASS FILTER
	//*******************************************
	for (c = 15; c <= (720-15); c++)
	{
```

```
                sum = 0;
                for (i = (c-15); i <= (c+15); i++)
                        {
                                sum = sum + column_hist[i];
                        }
                column_hist[c] = sum / 31;
        }

        //*********************************************
        // FILTERING OUT COLUMNS USING DYNAMIC THRESHOLD
        //*********************************************
        column_hist_average = column_total_sum / 720;

        for (c = 0; c < 720; c++)
        {
                if(column_hist[c] < column_hist_average)
                {
                        column_hist[c] = 0;
                }
        }
}

void horizontal_edge_processing()
{
        int r, c, i;
        int sum, difference;
        max_row_hist = 0;
        row_total_sum = 0;

        for(r = 0; r < 480; r++)
        {
                row_hist[r] = 0;
        }

        for (r = 0; r < 480; r++)
        {
                sum = 0;
                for (c = 1; c < (720-1); c++)
                {
                        if(I[r][c] > I[r][c-1])
                        {difference = I[r][c] - I[r][c-1];}
                        else
                        {difference = I[r][c-1] - I[r][c];}
                        if(difference > 30)
                        {sum = sum + difference;}
                }
        row_hist[r] = sum;
                row_total_sum = row_total_sum + sum;

                if(r >= 10 && r <= 470 && max_row_hist < sum)
                {
                        max_row_hist = sum;
                        max_row_number = r;
                }
        }

        //**************************************************
        // PASSING HORIZONTAL EDGE THROUGH LOW-PASS FILTER
        //**************************************************
        sum = 0;
        for (r = 10; r <= (480-10); r++)
        {
                sum = 0;
                for (i = (r-10); i <= (r+10); i++)
                        {
                                sum = sum + row_hist[i];
                        }
                row_hist[r] = sum / 21;
        }

        //*********************************************
        // FILTERING OUT ROWS USING DYNAMIC THRESHOLD
        //*********************************************
        row_hist_average = row_total_sum / 480;

        for (r = 0; r < 480; r++)
        {
                if(row_hist[r] < row_hist_average)
                {
                        row_hist[r] = 0;
                }
        }
}

int column[] = {0,0,0,0,0,0,0,0,0,0};
int row[] = {0,0,0,0,0,0,0,0,0,0};
void region_of_interest_extraction()
{
        int i, j;

        //*************
        // SEGMENTATION
        //*************
        j = 0;
        for (i = 2; i <= (720 - 2); i++)
        {
           if(column_hist[i] != 0 && column_hist[i-1] == 0 && column_hist[i+1] == 0)
                  {
                            column[j] = i;
                            column[j+1] = i;
                            j = j+2;
                  }
                  else if ((column_hist[i] != 0 && column_hist[i-1] == 0) || (column_hist[i] !=
0 && column_hist[i+1] == 0))
                  {
                            column[j] = i;
                            j = j+1;
                  }
        }

        j = 0;
        for (i = 2; i <= (480 - 2); i++)
        {         if(row_hist[i] != 0 && row_hist[i-1] == 0 && row_hist[i+1] == 0)
                  {
                            row[j] = i;
                            j = j + 1;
                            row[j] = i;
                            j = j+1;

                  }
                  else if ((row_hist[i] != 0 && row_hist[i-1] == 0) || (row_hist[i] != 0 &&
row_hist[i+1] == 0))
                  {
                            row[j] = i;
                            j = j+1;
           }
        }
        for(i = 0; i < 10; i = i+2)
        {
                if(max_row_number >= row[i] && max_row_number <= row[i+1])
                {
                        row1 = row[i];
                        row2 = row[i+1];
                }
                if(max_column_number >= column[i] && max_column_number <=
column[i+1])
                {
                        col1 = column[i];
                        col2 = column[i+1];
                }
        }
}

/*
 * ======== read_JP1 ========
 * Read the PAL/NTSC jumper.
 *
 * Retry, as I2C sometimes fails:
 */
static int read_JP1(void)
{
   int jp1 = -1;

   while (jp1 == -1) {
    jp1 = EVMDM6437_DIP_get(JP1_JUMPER);
    TSK_sleep(1);
   }
   return(jp1);
}
```

BIBLIOGRAPHY

[1] Clemens Arth, Florian Limberger and Horst Bischof, "Real-Time License Plate Recognition on an Embedded DSP-Platform", Proceedings of IEEE conference on Computer Vision and Pattern Recognition, pp 1-8, June 2007.

[2] Halina Kwasnicka and Bartosz Wawrzyniak, "License plate localization and recognition in camera pictures", AI-METH 2002, November 13-15, 2002.

[3] Texas Instruments Inc., "TMS320DM6437 DVDP Getting Started Guide", Texas, July 2007.

[4] Texas Instruments Inc., "TMS320DM6437 Digital Media Processor", Texas, Page 5, November 2006 – Revised June 2008.

[5] Texas Instruments Inc., "TMS320C64x+ DSP Cache User's Guide", Literature Number: SPRU862A, Table 1-6, Page 23, October 2006.

[6] Pramod Kapadia, "Car License Plate Recognition Using Template Matching Algorithm", Master Project Report, California State University, Sacramento, Fall 2010.