James Beaumont-Field
135 View Terrace
Bicton WA, 6157
November 3rd 2004

Professor Syed Islam
Head of School
Department of Electrical and Computer Engineering
Curtin University of Technology
Bentley WA 6102

Dear Professor Islam,

I hereby present my thesis entitled “Image Acquisition and Processing on the Low Vision Image Enhancer” as part of the requirement to complete the degree of Bachelor of Engineering (Electronics and Communications Engineering).

Yours Sincerely,

……...

James Beaumont-Field

Curtin University of Technology

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

TITLE:

Image Acquisition and Processing on the Low Vision Image Enhancer

AUTHOR:

FAMILY NAME:
Beaumont-Field

GIVEN NAME:
James

DATE 5 November 2004	SUPERVISOR Mr. Iain Murray
DEGREE Electronics and Communications Engineering	OPTION

ABSTRACT

This thesis outlines the design and testing of various image acquisition and processing techniques and their implementation on a portable image enhancer for low vision users. Included is a description of the most common causes of low vision, their characteristic symptoms, and the reasoning behind the proposed countermeasures. The result is a basic design for a portable image enhancer, intended to aid students with low vision by providing several specifically designed user controllable filters.

INDEXING TERMS

Low Vision, Image Processing, Threshold, Image Sensor, Image Enhancement

	GOOD	AVERAGE	POOR
TECHNICAL WORK			
REPORT PRESENTATION			

EXAMINER

CO-EXAMINER

Curtin
UNIVERSITY OF TECHNOLOGY
School of Electrical
and Computer Engineering

Image Acquisition and Processing on the Low Vision Image Enhancer

By

James Beaumont-Field
12040580

Thesis submitted as part of the requirements for the completion of the Bachelor of Engineering (Electronics and Communications Engineering) degree.

November 2004

Abstract

This thesis outlines the design and testing of various image acquisition and processing techniques and their implementation on a portable image enhancer for low vision users. Included is a description of the most common causes of low vision, their characteristic symptoms, and the reasoning behind the proposed countermeasures. The result is a basic design for a portable image enhancer, intended to aid students with low vision by providing several specifically designed user controllable filters.

Acknowledgements

I would like to express my appreciation to my project supervisor Iain Murray. His knowledge and support throughout this project has made this work possible.

I would also like to thank Andrew Pasquale for his patience and helpful suggestions throughout the life of the project. His knowledge has proven invaluable in progressing through the work.

Nomenclature

BIOS – Basic Input Output System

CCD – Charge Coupled Device

CMOS – Complementary Metal Oxide Semiconductor

CODEC – Coder/Decoder

CPU – Central Processing Unit

DSP – Digital Signal Processor

EDMA – Enhanced Direct Memory Access

EMIF – External Memory Interface

FFT – Fast Fourier Transform

FIR – Finite Impulse Response

FPGA – Field Programmable Gate Array

I^2C – Inter-Integrated Circuit

LoVIE – Low Vision Image Enhancer

McBSP – Multi-Channel Buffered Serial Port

SDRAM – Synchronous Dynamic Random Access Memory

Table of Contents

Table of Figures

Table of Equations

1. Introduction

1.1 The Problem

Visual aids are used extensively in all areas of education. When trying to express ideas, verbal communication is usually not sufficient. Traditional teaching methods involve the use of blackboards, whiteboards, overhead projectors and other types of visual demonstration to supplement verbal explanations. Books and computers are also important learning tools for they facilitate easy storage and retrieval of large amounts of information in a form that can be easily distributed. Unfortunately, many people are visually impaired to such an extent that in a normal teaching environment the usefulness of the apparatus mentioned above is greatly decreased. The result can be a far less effective learning experience.

Any vision impairment severe enough to impede the performance of common everyday tasks, while still allowing some useful visual discrimination, is referred to as low vision. Currently there are over 300 000 blind or vision impaired people in Australia, with 24 900 affected people in Western Australia alone. While vision impairment is more likely to affect older adults, a significant number (47 %) of those affected are younger people. It is the latter, who are more likely to be receiving some type of formal education.

There are many different types of visual impairment, many developing from common conditions such as diabetes. Impairments such as glaucoma and cataracts can result in blurred, distorted vision as well as a reduced field of vision. Some disorders, such as

macular degeneration, can cause central blind spots, reduction in contrast sensitivity and even time varying changes in the individual's colour sensitivity. The result can be a person who has difficulty with low-contrast images, who in addition may be more sensitive to certain ranges of colour. To make an individual's interaction with the teaching environment more productive, a method of making existing visual aids clearer for a particular user needs to be established. The device defined in this report is anticipated to be a portable, low cost solution to the problems outlined above.

Besides the device's intended purpose, there are many other uses that could see it fill several niches in the consumer market. As the image sensor used in the device is capable of picking up infrared light as well as visible light, it would prove useful in certain low light situations. If packaged with an infrared spotlight, the device would be an excellent tool for reading text books in areas of low ambient light (for example lecture halls and libraries) without disturbing others. Pointing the afore-mentioned infrared spotlight at the material to be viewed would have a similar effect to using a regular spotlight when viewed through the device, while remaining invisible to all other onlookers. In the home, everyday activities such as cooking could be made simpler by using the device to magnify recipes such that they could be seen from anywhere in the kitchen.

1.2 The Solution

Although similar devices do already exist on the market, they have various shortcomings. Portable devices with similar functionality can be priced at over $2000, putting them far out of reach for most students. Others priced more reasonably are only capable of

reading up-close material (for example textbooks) and/or do not include any device to present the enhanced image; rather they require a television or computer monitor to display their output, effectively rendering the device non-portable in most situations.
For a more detailed description on current enhancer devices, see Arulliah, E. [19].

It is intended that the device outlined in this document be priced far more reasonably; this is possible mainly due to the fact that it is being developed as a student project. Most costs related to development such as development man-hours, development equipment and space concerns usually have to be recovered as part of the retail price. Since development was done at no charge, using equipment and space already supplied, the device does not carry these overheads and thus will benefit from a greatly reduced price.

Most of the electronic visual aids already available on the market operate on a relatively simple level. In order to make an image clearer they employ simple contrasting and thresholding techniques resulting in a high contrast reproduction of the original image. More advanced devices may offer the option to change foreground and background colours as well as feature line markers to help the user stay focused on the desired information. The device defined in this document offers all of the features outlined above as well as employing more advanced image processing techniques. For example, the device will try to intelligently determine the difference between the foreground (information) and the background (whiteboard, shadows, dirt etc.) in order to display a more efficient representation of the desired information. Also, instead of just generally

improving an image, attempts have been made to directly counter common low vision symptoms using specifically designed filters.

1.3 Thesis Outline

The following gives a summarised overview of this document, describing all chapters and their contents.

Section 2. Some Theoretical Approaches to Low Vision Enhancement for Students:

This section looks at the causes and symptoms of the most common vision disorders. A broad review of the theoretical issues relevant to the problem is given, as well as a justification of the approach taken in the design of filters on the LoVIE. A brief outline of the other hardware components in the system is also provided.

Section 3. Image-Processing Techniques and Test Results:

This section examines the various image-processing techniques implemented in the LoVIE design and also contains an in-depth explanation of the development process employed for each filter.

Section 4. Hardware Based Development Platform:

This chapter describes the requirements and selection of the digital signal processor used in the design. It includes a relevant description of the features available on the selected DSP that make it suitable for development of the LoVIE.

Section 5. Camera Selection and Evaluation

This chapter gives a description of the CMOS image sensor used in the design. It addresses such issues as why this image sensor is suitable and how it was implemented.

Section 6. User Interface:

An overall description of the user interface design is given, including accessible features, justification and hardware implementation.

Section 7. Conclusion:

This chapter summarises the progress made on the LoVIE design. It also lists several recommendations for future development.

2. Some Theoretical Approaches to Low Vision Enhancement for Students

It is reasonable to assume that in order for any image enhancement to be effective for a low vision user, the enhancement should in effect, counter the detrimental effects of the user's condition. It is upon this premise that the enhancement algorithms used on the LoVIE were developed. Obviously, a single device could not possibly be tailored to assuage every symptom associated with low vision. Instead, the approach taken in the design of the LoVIE is to address the most common problems faced by low vision students. By giving the user precise control over the different enhancements available, the output from the device can be better tailored to suit individual needs.

The four most common causes of low vision after birth are macular degeneration, cataracts, glaucoma, and diabetic retinopathy [1]. Macular degeneration affects the centre of the retina, or macula. It degenerates as a result of a breakdown of the retinal pigment epithelium, the insulating layer between the retina and the layer of blood vessels behind it. The result is that the macula is effectively starved and dries out, causing blurred vision. The breakdown of the retinal pigment epithelium can also let harmful components of blood into the retina, leading to fluid leakage that can cause a loss of central vision.

A Cataract is an area of the eye's lens that has become cloudy or opaque. It can be inherited at birth but are most commonly found in the elderly. Environmental factors such as smoking or toxic substances as well as metabolic conditions such as diabetes can

greatly accelerate this condition. Cataracts affect a patient's night vision and perception of light, and can also cause sensitivity to glare.

Glaucoma is the name given to damage of the optic nerve, due to an increase in fluid pressure inside the eye [2]. This increase in fluid pressure is due to improper drainage and causes damage by restricting the blood supply to the optic nerves. This starvation causes the nerves to die off; leading to blind spots first in the sufferer's peripheral vision and, without treatment, total blindness. People with the greatest risk of developing glaucoma include the elderly or those with diabetes.

Finally, diabetic retinopathy is caused by damage to the blood vessels of the retina [3]. Due to this damage, these blood vessels can leak and cause blurred vision, or in more severe cases haemorrhage and cause vision loss and scaring. People with type I or II diabetes are at risk of developing this condition; the likelihood and severity of diabetic retinopathy increases with the duration the patient has suffered diabetes.

All the symptoms outlined above can be categorised into 3 broad problem areas:

- Blurred vision – the patient has difficulty with fine details in images.
- Localised vision degradation – certain areas of a patient's vision are lost or distorted.
- Altered colour/contrast perception – patients have difficulty perceiving slight changes in contrast and may experience discomfort when viewing certain colours.

By implementing several independent processes that combat each problem area separately, it is intended that the LoVIE be able to provide suitable enhancement for most low vision users.

In addition to the actual image enhancement, there are several other factors in the design of the low vision enhancer that need to be addressed. Firstly, the system requires some way of capturing the input images for processing. As the device is intended for low vision users, the selected camera should require little or no setup time from the user.

Secondly, the LoVIE must offer the user a method of controlling the featured enhancements such that they may be tailored to suit each particular user's needs. Above all, any user interface implemented must be clear and simple to use. Offering too complicated an interface would prove counter productive as the user would be deterred from attempting to use the device to its full potential.

Finally, without a clear method of representing the output, any enhancement provided would be rendered useless. As such, a display capable of giving large, clear output must be implemented without reducing the devices portability significantly. As this report focuses mainly on the image capture and processing aspects of the LoVIE, little detail is given concerning the display subsystem. For an in depth description of the display interface, refer to Lowe, J. [20].

3. Image-Processing Techniques and Test Results

3.1 Algorithm Design Process

3.1.1 Introduction

While designing and testing an image enhancement algorithm, constraints such as processing time, memory usage and compatibility are of little or no concern. As such, instead of testing designs directly on the C6711, a high-level design environment that allows algorithms to be quickly and easily implemented and tested was needed. The MATLAB software package, including the MATLAB Image Processing Toolbox, allows images to be easily loaded into memory and manipulated via a high-level language similar to C.

Once each algorithm was working as desired in the MATLAB environment (See Appendix B), the code was ported to C and compiled and tested on the C6711 via Code Composer Studio (See section 4.2.4). While it is possible to embed code from the MATLAB environment directly onto the C6711, it was felt that the level of control over memory allocation and optimisation required, would only be attainable by coding the algorithms manually. The following is a short description of the more useful tools provided and implemented in the MATLAB environment.

3.1.2 Working with Images

All test images were be loaded saved to and from the MATLAB design environment using the *imread* and *imwrite* commands as follows

```
>> image=imread('test.bmp');
…process image...
>> imwrite(image,’test result.bmp’);
```

As only 8-bit greyscale images need be considered for the LoVIE design, once loaded, images were converted to the correct format using the *rgb2gray* command as follows

```
>>image=rgb2gray(image);
```

At this point, the image can be manipulated in the same way as a standard matrix. Additionally, many of the processing functions developed require the image matrix represented in *double* format, instead of the default 8-bit unsigned integer (*uint8*). The *double* format represents each pixel as a decimal number between 0 and 1 while the *uint8* format represents them as an integer number between 0 and 255. Images were converted between the two formats using

```
>>image=im2double(image);
>>image-im2uint8(image);
```

3.1.3 Transferring Images between MATLAB and the C6711

Often, it was desired that a test image be loaded into memory on the C6711 to test the ported enhancement algorithms against a known result. At other times it was necessary

to use a captured image from the M3188A in order to test the designed algorithms in MATLAB. The easiest way to achieve this was by using the "Load Data" and "Save Data" functions from within Code Composer Studio. These functions allow the C6711's memory space to be read from or written to using a raw hexadecimal data file. Thus two MATLAB functions, *ti_data_read* and *ti_data_write,* were created that enabled MATLAB to read and write image data in this format. These functions are available in Appendix B and were used in the following manner

```
>>image=ti_data_read('captured.dat');
…process image…
>>ti_data_write(image, 'processed.dat');
```

3.1.4 Output Format for Display Subsystem

In order for the display subsystem to function correctly, the output image must be constantly available, properly formatted and stored in a pre-defined memory space. The output image is stored depending on whether the output required is colour or greyscale. The colour output format is as yet only used whenever the threshold filter (See Section 3.4.1) is applied and configured to display the foreground and background in colours other than black and white. To indicate the format of each particular frame, a register within the display subsystem is set via the EMIF (See Section 4.2.3.1). Figure 3.1 shows the pixel formats for colour and greyscale pixels.

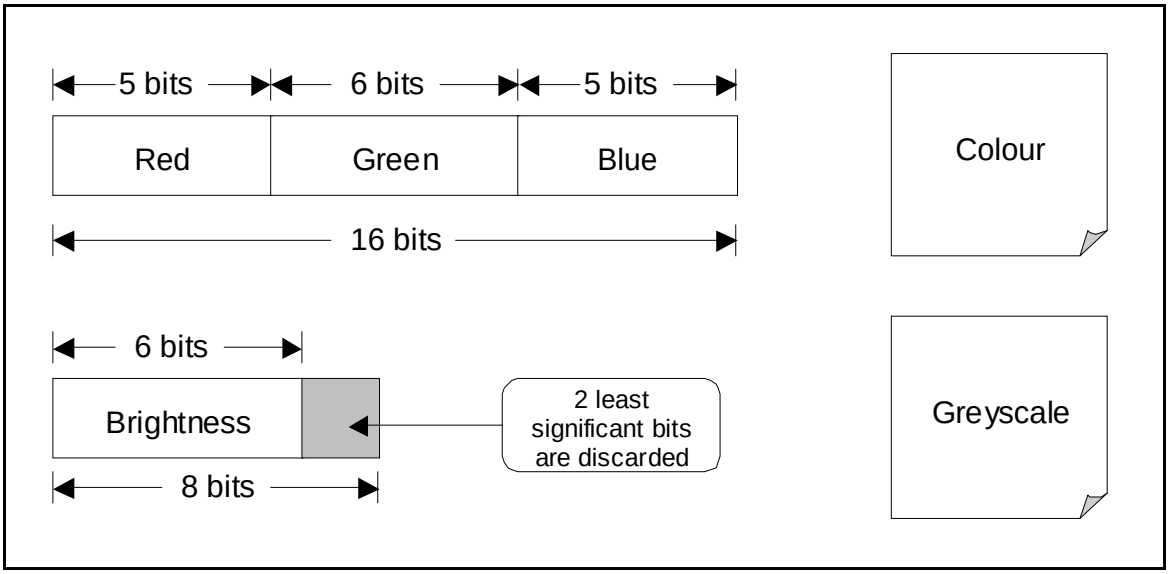


Figure 3.1 Output Pixel Formats

Greyscale output is stored identically throughout the system, using 8-bits per pixel. Thus the resultant image need only be transferred to the appropriate location before being output to the display subsystem. Currently, as the display uses only 6–bits per greyscale pixel, the 2 least significant bits are discarded before being passed to the display. In the future it is recommended that a dithering algorithm be implemented to optimise image quality (See Section 7.2).

Colour output is handled in a slightly different manner. Once the thresholded image is ready for output it is processed in the following manner. Replicas of the desired foreground and background colour pixels are stored in memory. For each pixel in the image, a colour foreground or background pixel is transferred to the equivalent location in the output buffer. While the greyscale output could be processed in the same manner,

this would require a 16-bit transfer per pixel where only 8 bits are needed. Thus the differing methods are utilised to minimise the bandwidth requirement of the display subsystem.

3.2 Techniques to Combat Blurred Vision

3.2.1 Effects of Blurred Vision

One of the most common symptoms in low vision individuals, as described earlier, is blurred vision. Those with blurred vision experience a general loss in sharpness of an image making it appear "out of focus". For the purposes of investigation and testing, a blurred effect was simulated using a Gaussian blur convolution matrix (See Appendix B).

3.2.2 Edge Enhancement

In order to attempt to reverse the effect of blurred vision, it is first necessary to establish an equivalent model for a blurriness filter. Upon examining the example image given in Figure 3.2, sharp edges appear to have been smeared, making fine details difficult to distinguish. In other words, the high frequency components of the image appear to have been reduced or removed altogether. Thus, it has been suggested that general blurriness is the equivalent of applying a low-pass filter to an image.

Figure 3.2 Simulation of Blurred Vision

Peli and Fine [4] proposed that the best way to go about countering this low-pass filtering effect would be to pre-amplify all high frequency components in the image before submitting it for viewing. If done appropriately, the image perceived by the end user should be close to the original image. Figure 3.3 gives a block diagram of the inverse filter suggested.

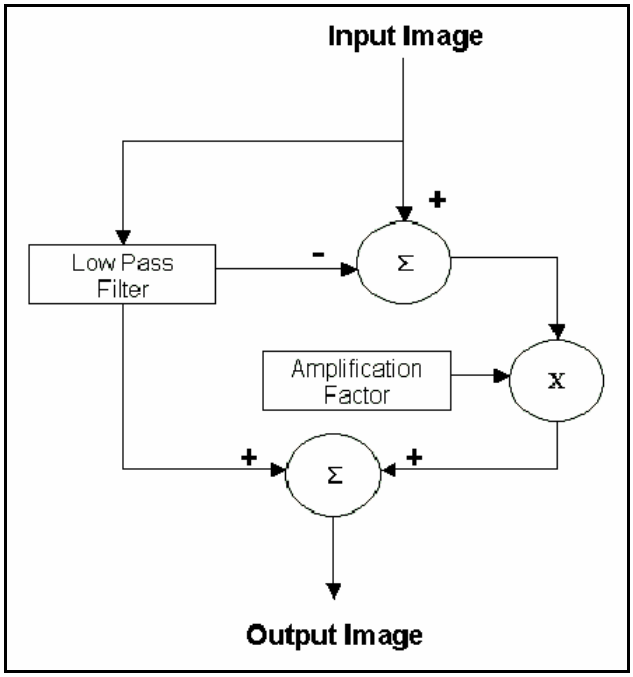


Figure 3.3 Block Diagram for High Frequency Amplification

To implement this filter on the C6711 in the frequency domain, a 2-D Fourier transform must be performed on each frame twice. In order to compute the Fourier transform of an image using the computationally efficient radix-2 FFT, the image first needs to be zero padded so that both dimensions are radix-2 numbers, resulting in an image with dimensions of 1024x512. Next, the FFT of each line in the image has to be computed, followed by the FFT of each column; thus, a total of 512 1024-point and 1024 512-point FFT's need to be performed to transform one image. Consequently, it is not practical to attempt to implement filters in the frequency domain in real time. As such, an alternate method of amplifying an image's high frequency components in the spatial domain will be established.

The equivalent of low-pass filtering in the frequency domain can be achieved in the spatial domain using a 2 dimensional FIR filter. As all other operations involved in the filter are linear (specifically addition and multiplication), they can also be carried out in the spatial domain without any further transformations.

Figure 3.4 and Figure 3.5 below show the effect the Edge Enhancement filter has on a test image taken from the LoVIE. The normalised low-pass filter cut-off frequency is 0.2 and the foreground is amplified by a factor of 10 (all parameters are user definable). It should be noted that as the edges of the text represent a relatively high change in frequency, they have been emphasised, as have the edges of the whiteboard.

Figure 3.4 Test Image Before Edge Enhancement

Figure 3.5 Test Image After Edge Enhancement

A second feature of the Edge Enhancement filter is the ability to reduce the significance of low frequency components. In other words, any soft transitions such as reflections or shadows across the whiteboard can be reduced or even eliminated altogether. Figure 3.6 shows the output of the Edge Enhancement filter with the background amplification at 0.2 and the normalised cut-off frequency at 0.08.

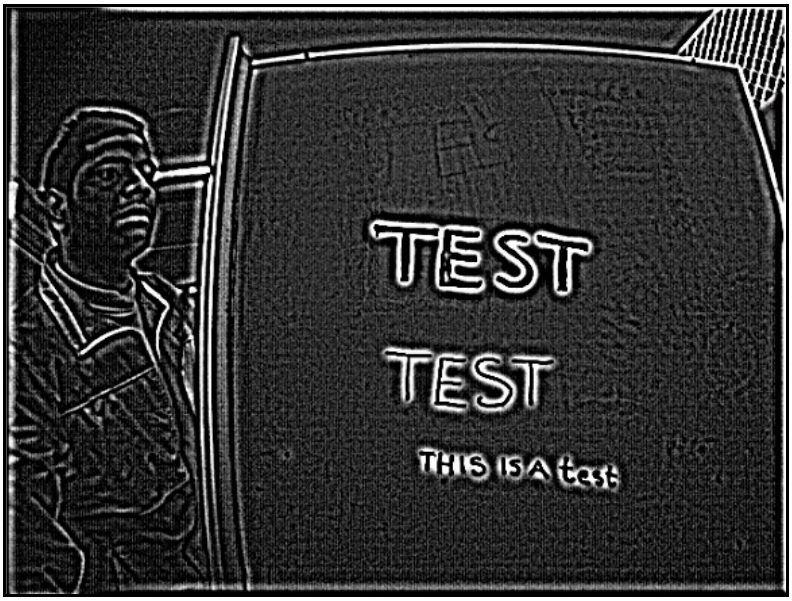


Figure 3.6 Test Image with Low Frequency Components Reduced

It should be noted that with proper adjustment, it is possible to obtain a result similar to an edge detection filter, except that the effective size of the text has been increased. E. M. Fine and E. Peli [4] found that while purely increasing text size did have a small effect on reading rates, the enhanced text produced better results in this respect. Unfortunately, the same study did not produce conclusive results on the benefits of the Edge Enhance filter on reading rates for low vision individuals in general. However, as the LoVIE is designed as an aid to improve clarity, not reading rates, it was felt the inclusion of this filter was justified.

3.2.3 Edge Outline

After examining many simulated examples of blurred vision, it was proposed that in real terms, blurred vision has the effect of making boundaries indistinct. The effect of this is

to make relatively intricate symbols such as text and diagrams extremely difficult to discern. A logical remedy for this would be to enhance the definition of boundaries such that when blurred, they are more easily recognised.

To achieve this on the LoVIE involved a two-step process. Firstly, the image is passed through a Sobel edge detector to isolate the boundary components of the image. The Sobel edge detector was chosen because it has a fairly low sensitivity to noise while being relatively fast, computationally, compared with similar quality edge detectors [5].

The second step of the process involved combining the boundary data obtained from the Sobel edge detector with the original image in some way as to highlight areas judged by the Sobel operator to be significant. As this step will have quite a significant effect on the filter's output, a fair amount of time was spent experimenting with different methods of combining the two frames. The final method decided upon, involved brightening all boundary regions in the image relative to the detected boundary strength. The result of this was a bright line superimposed on the edges of well-defined areas such as text with less contrasted areas obtaining a subtly lightened outline.

Figure 3.7 Test Image with Edge Outline Filter Applied

While the Edge Outline filter is based upon a logical premise, it remains to be seen whether it is effective. The current implementation of the filter (See Appendix A) takes around 5 seconds to complete one frame, and as such requires some optimisation before it could be applied in real time. Another issue is that the use of the Edge Outline filter could not be justified through simulation. Figure 3.8 shows the resultant image of the Edge Outline filter as seen through simulated blurred vision compared with that of the original unprocessed image. The Edge Outline filter does not appear to have improved the clarity of the blurred image; as such it has been left out of the current implementation of the LoVIE. It is suggested, however, that the use of this filter should not be completely ruled out until testing can be undertaken on actual low vision individuals.

(a)

(b)

Figure 3.8 Original Test Image (a) and Test Image with Edge Outline (b) with a Low Pass Filter Applied

3.2.4 Focus Lines

Individuals with low vision often have difficulty following a single line of text if it is surrounded by other text or visual information. This can be a result of blurred vision,

lack of fine eye-movement control or a combination of both. In order to compensate for this difficulty, the LoVIE provides a filter that enables the user to highlight a region of the display to help keep track of areas of interest. As shown in Figure 3.9, the user definable region is bordered on each side by a thick black line. Areas outside the 'focus area' appear darkened, and thus less attractive to the eye.

Figure 3.9 Test Image with Focus Filter Applied

3.3 Techniques to Combat Localised Vision Degradation

3.3.1 Digital Zoom

Localised vision degradation is a symptom that can make reading extremely difficult. Sufferers experience partial or total loss of vision in certain areas, making it difficult to recognise text, faces, or anything requiring the perception of fine details. In this case,

more so than with other symptoms, the best possible outcome of any enhancement applied, is only to reduce the significance of the impairment in an educational environment.

By using digital zoom techniques, the overall size of the image region of interest can be increased, thus reducing the portion of the image distorted relative to the information contained therein. Figure 3.10 below shows normal and zoomed images, as they would appear to a student experiencing localised vision loss. It should be noted that in the normal image the dark region would cover the entire middle word, while in the zoomed image the affected region distorts only fragments of individual letters.

(a)

(b)

Figure 3.10 Test Image with Simulated Localised Vision Loss Before (a) and After (b) Digital Zoom

The digital zoom currently used on the LoVIE involves a simple bilinear interpolation method. While not the most effective interpolation method available, bilinear interpolation is one of the most efficient true interpolation methods available and thus was used in the current digital zoom implementation to facilitate use until other interpolation methods can be implemented and optimised (See Section 7.2.7).

3.4 Techniques to Combat Altered Colour/Contrast Perception

3.4.1 Thresholding

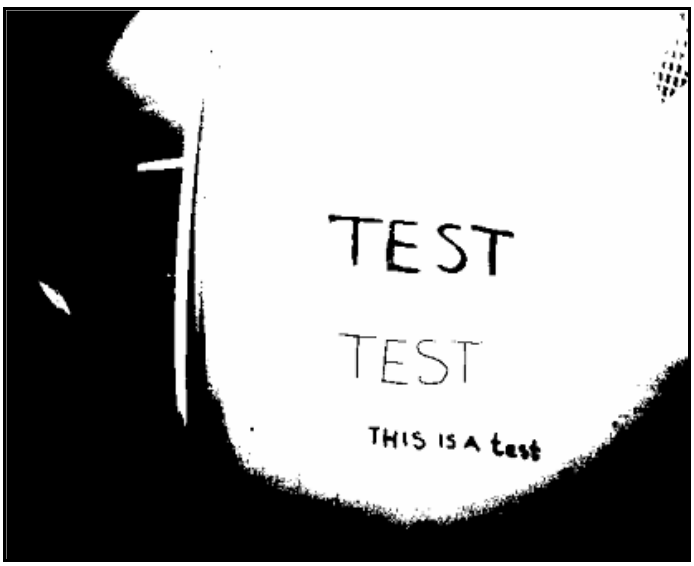
3.4.1.1 General Threshold Process

Many low vision disorders can cause a reduction in contrast perception. Individuals with this symptom can experience difficulty discerning subtle changes in contrast, making faint or distorted images difficult to interpret. In an educational environment, this situation could be caused by things such as the use of a faint marker on a whiteboard, or shadows cast across a blackboard. Fortunately, this is a symptom that can be countered effectively.

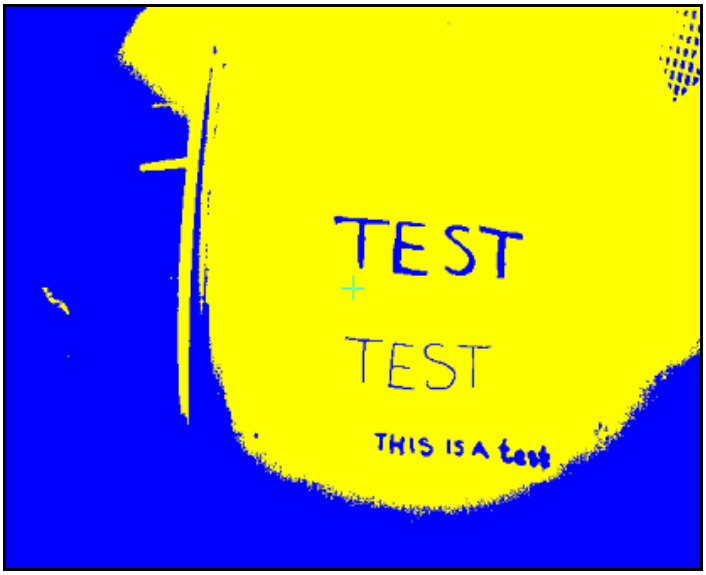
The basic idea behind thresholding involves choosing a mid point or threshold value for an image. All pixels darker than this value are considered to belong to the foreground component of the image, while all pixels brighter than this are considered background (in other words, dark text on a white background). By performing this operation on all pixels in an image, it can be compressed into binary form, increasing all contrasts to the maximum possible. The key to properly thresholding an image is choosing an appropriate midpoint brightness value between foreground and background. Several different methods of intelligently obtaining this value, based on the input image were investigated. The two most successful and thus fully implemented methods are outlined below. While these methods are generally close to the optimal value, they can sometimes

be wrong. As such, the user is given the option to manually alter the automatically selected threshold value (See Section 6).

Once the image has been divided into foreground and background components, the LoVIE is able to process each component independently. The main benefit of this, at this stage of development, is that the user is given the option to choose a combination of foreground and background colours that they find the most comfortable to view. For example, those with retinitis pigmentosa experience excessive glare when looking at purely white light [6], and therefore usually prefer to view a colour foreground and background combination. Figure 3.11 shows the output of the threshold filter with black-and-white and colour outputs.



(a)



(b)

Figure 3.11 Test Image with Threshold Filter Configured for Greyscale (a) and Colour (b) Outputs

3.4.1.2 White Background Assumption Technique

One method by which the LoVIE selects a threshold value was derived based upon the assumption that the input image will consist mainly of a white background (the whiteboard) with black foreground (text, diagrams). Figure 3.12 shows a typical histogram for such an image.

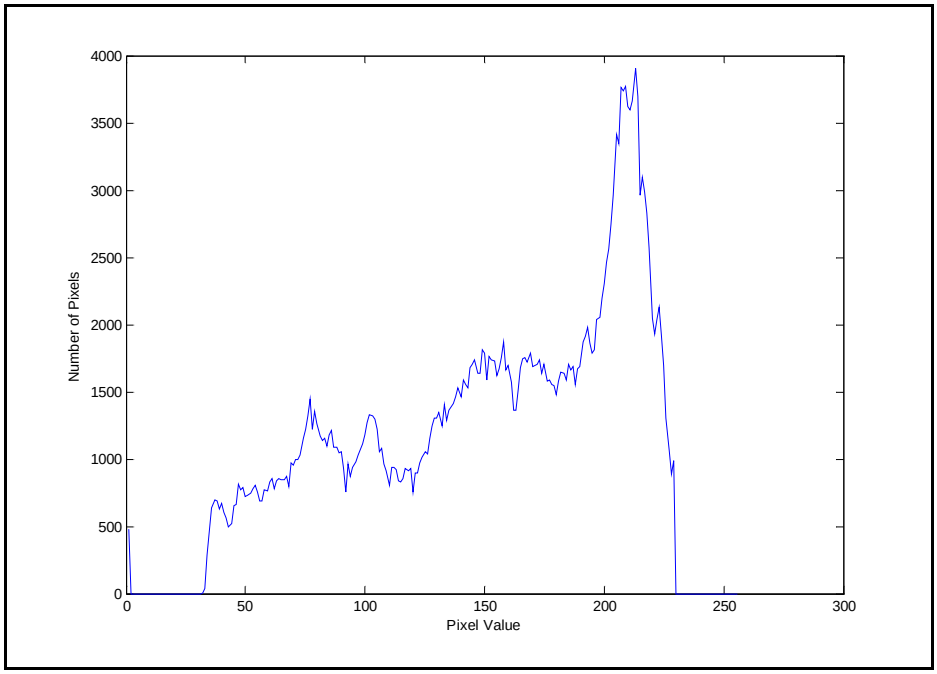


Figure 3.12 Typical Histogram of a Mainly White Background Image

The large peak to the right of the plot is a result of the white coloured background making up most of the image. If it is assumed that the mean of the brightness values associated with the background is the same as the median, then the middle of the peak can be found simply by finding the maximum value in the histogram. By making another assumption that the background is the brightest component of the input image, then the half width of the peak can be found by calculating the distance of the maximum from the highest brightness value (234 in this case). Once the width and location of the peak are known, a threshold value can be established at a point slightly lower than the lowest value of the peak. In this way, the background has been set to incorporate the whiteboard while the foreground is considered to be everything else. While testing, it was found that this method worked particularly well if the above assumptions were true. However this

method fails once the white background no longer occupies a significant portion of the input image.

3.4.1.3 Fuzzy Logic Technique

The second technique for establishing a valid threshold point implemented on the LoVIE involved a two-step process. Firstly a logarithm contrast operator, given by Equation 3.1 [7] (where R is the maximum brightness in the input image), adjusts the image so that the contrast of lighter coloured areas (shadows or reflections on the whiteboard) is reduced, while the contrast of darker areas, such as text, is increased.

$$Q(i, j) = c \log(1 + |P(i, j)|)$$

$$c = \frac{255}{\log(1 + |R|)}$$

Equation 3.1 Log Contrast Equation

Figure 3.13 gives a graphical representation of the transfer function of the logarithm operator. It should be noted that the rate of change of the operator is much higher for lower brightness values.

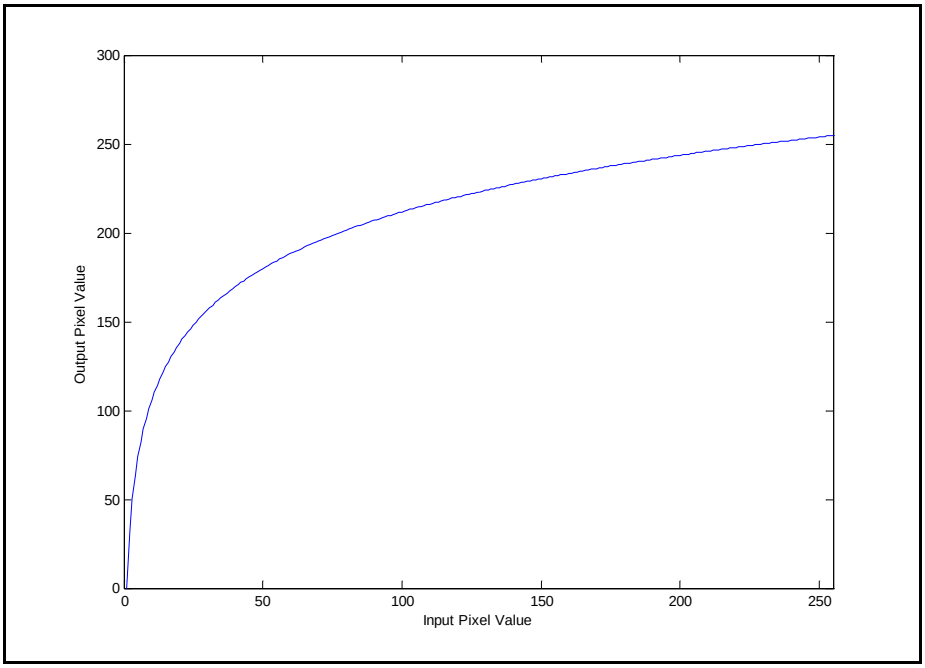


Figure 3.13 Transfer Function of Log Contrast Operator

Once the contrasting operation is complete, the second stage of the Fuzzy Logic Technique is carried out. This involves summing the darkness coefficient of each pixel to obtain the number of "dark" pixels in an image. In order to obtain this coefficient, each pixel is classified according to a "darkness" fuzzy set. A typical transfer function of this fuzzy set is given in Figure 3.14.

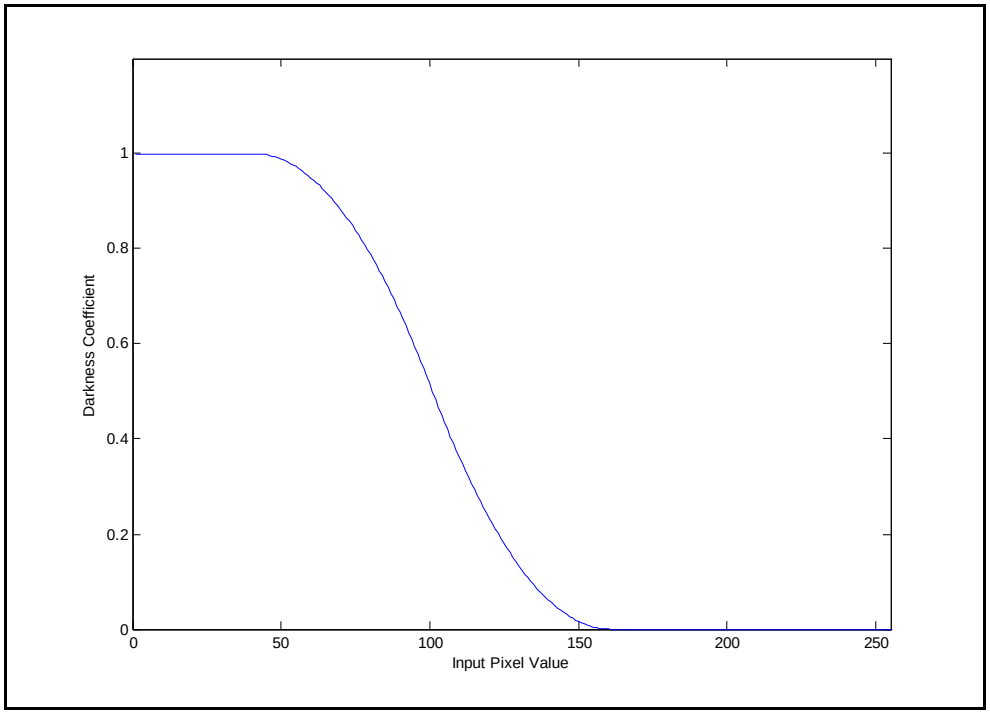


Figure 3.14 Typical Transfer Function of "Darkness" Fuzzy Set

As darkness is a purely relative measure, the cut-offs of the transfer function are modified according to the histogram of the log-contrasted image. From the fuzzy transfer function a value between 1, meaning purely dark, and 0, meaning purely light, for each pixel is obtained. The sum of these values, or "darkness coefficients", is taken to be the true number of dark, or foreground, pixels in an image. Once this sum is obtained, the threshold point is calculated by finding the location in the images histogram with the number of pixels darker than that point being equal to the sum of the darkness coefficients.

3.5 Basic Image Controls

As well as including some more advanced features, it was felt that the LoVIE should still include the facility to adjust basic image parameters such as brightness and contrast. While the C6711 itself could be used to implement these controls with some basic processing, this was seen as an unnecessary increase in processing load as the hardware used in the LoVIE has these facilities already built in.

Firstly, the display used in the current design features controls that can be used to adjust the output contrast, colour and brightness much like that of a typical television or monitor. Secondly, the M3188A camera module features automatic gain control and white balancing algorithms that attempt to output the best possible image for any lighting conditions. The facility to adjust these parameters on the M3188A manually could be implemented quite easily with the addition of an I^2C interface [8].

4. Hardware Based Development Platform

4.1 Hardware Platform Requirements

Today, there are many hardware based development platforms that facilitate the development and implementation of embedded systems. Many platforms are designed for specific purposes including audio processing, security, and imaging. As such, a wide range of platforms with diverse feature sets are available. Due to this, choosing an appropriate platform for a particular application is an extremely important stage of the development process.

Before deciding on a development platform for the LoVIE, a set of requirements were established to ensure an appropriate choice would be made. The requirements decided upon, and the reasoning behind each, are outlined below.

First and foremost, as the LoVIE was to provide several image enhancements, simultaneously if necessary, processing power was a concern. At this stage of development, particulars were not known, but it was expected that an image resolution of 640x480 pixels with 256 levels of grey would be sufficient. The majority of image enhancements fall into 2 categories. Point to point operations, such as thresholding and contrast adjustment, involve transforming each pixel individually based on upon a certain set of rules. A 640x480 image features 307200 pixels, each of which may need to be transformed several times before being output to the display. As an example, to apply one enhancement to a frame at a rate of 15 frames per second (the desired rate at this

stage of development) would involve at least 307200 x 15 = 4608000 complex calculations per second, a fairly heavy processing load for a development platform. Convolution based enhancements, including FIR filtering and digital zooming, transform each pixel according to a weighted sum of itself and its surrounding pixels. Thus, instead of one calculation per pixel, the processor must perform several (depending on the convolution mask size). The platform decided upon must be capable of these speeds to be of any practical use in development.

The second major issue in deciding on a development platform is available memory space. In order to perform several image enhancements on one frame, as well as the original frame, each intermediate frame needs to be stored for the next enhancement to process. Because of this it was estimated that as many as 6 frames may need to be kept in memory at any one time. A single 640x480 256-level greyscale image requires approximately 300 kilobytes; therefore for the initial estimate of 6 frames, at least 1.8 megabytes of memory space would be required. This is not an unusual or particularly large requirement by today's standards, but it remains an important requirement none the less.

The development platform decided upon will need to be capable of receiving image data from an image sensor and outputting each enhanced frame to a display in a suitable manner. As such, the development platform decided upon must feature appropriate interfaces, capable of transmitting and receiving data at a link bandwidth of at least 37 Mbps (See Equation 4.1).

$$640 \times 480 \times 8 \times 15 = 36864000 \approx 37 Mbps$$

Equation 4.1 Approximate Bandwidth Requirement at 15 fps

In addition to this, it was intended that the user be provided with a means of controlling the featured enhancements via an appropriate user interface (See Section 6). The chosen platform must be able to accommodate this and ideally, more, to allow further functionality to be added as necessary.

Finally, to facilitate implementation and optimisation of the designed enhancements, the chosen platform should feature a flexible and stable design environment, preferably with a highly developed set of debugging tools. As this is the first major design undertaken by the students involved, many mistakes were anticipated. Without appropriate tools, the development process could become unnecessarily difficult and lengthy. Also, an optimised library set including relevant mathematical operations such as 2D FIR filtering and matrix operations would greatly speed up implementation and simplify the optimisation process.

4.2 The TMS320C6711 Digital Signal Processor

4.2.1 Overview

The Texas Instruments TMS320C6711 is a low cost (approximately $550 Australian dollars at time of printing), high performance 32-bit floating point digital signal processor designed especially for complex multi-channel and multifunction applications. It was chosen as the hardware development platform for the LoVIE as it meets or exceeds all the requirements outlined in Section 4.1. The following sections describe the relevant components of the C6711 and their capabilities.

4.2.2 Processor and Memory

The TMS320C6711-BGFN150 is the specific C6711 model used in development of the LoVIE. Designed specifically for high performance DSP applications, it features a 150 MHz CPU capable of up to 900 million floating point operations per second. The CPU features 32 general-purpose registers of 32-bit word length as well as 8 independent functional units for fixed and floating-point operations. As well as 512 kilobits of on-chip program/data memory, the C6711 features 16 megabytes of external SDRAM addressable via a glueless external memory interface.

The specifications above were seen to meet those defined in Section 4.1; thus the C6711 was deemed an appropriate choice in this respect.

4.2.3 Interfaces

4.2.3.1 The External Memory Interface (EMIF)

The C6711 features a 32-bt external memory interface that provides a glueless interface to both synchronous and asynchronous memory spaces and is capable of addressing up to 256 megabytes of external memory [9]. While, the C6711 comes packaged with 16 megabytes of external SDRAM available via the EMIF, for the LoVIE it is also currently used as an interface to the display subsystem [20].

4.2.3.2 The Multi-channel Buffered Serial Port (McBSP)

The McBSP featured on the C6711 features 2 serial port channels each capable of acting either as a standard serial port or a set of independent general purpose binary input output pins.

The serial port interface provides [10]

- Full-duplex communication
- Double-buffered data registers, which allow a continuous data stream
- Independent framing and clocking for reception and transmission
- Direct interface to industry-standard CODECs, analogue interface chips (AICs), and other serially connected A/D and D/A devices
- External shift clock generation or an internal programmable frequency shift clock
- 8-bit data transfers with LSB or MSB first
- Programmable polarity for both frame synchronization and data clocks

- Highly programmable internal clock and frame generation

For the LoVIE, both channels of the McBSP have been reconfigured to act as general-purpose input/outputs. The design of the McBSP is such that only 6 independent pins are available for input per channel. Figure 4.1 below shows the register configuration of the McBSP channels.

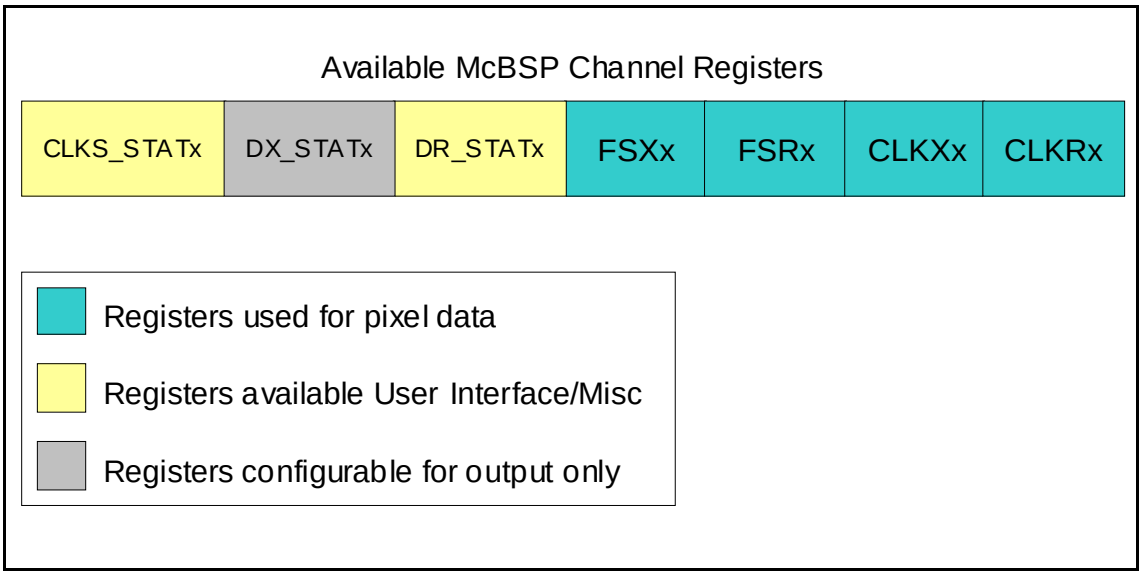


Figure 4.1 Register Configuration of the McBSP Channels

With only 6 pins available per channel, the current interface method (See Section 5.3.4) utilises 4 registers per channel to receive the 8-bit data bus from the M3188A camera module (See Section 5.2). Currently, the remaining free pins on the McBSP are used in the implementation of the user interface, outlined in Section 6, and to request each frame from the M3188A.

4.2.3.3 Enhanced Direct Memory Access (EDMA)

The C6711's EDMA is a highly efficient data transfer engine theoretically capable of transferring 8, 16, or 32 bit blocks at speeds of up to 1200 megabits per second at a CPU rate of 150 MHz. This data rate is much higher than the required 37 megabits per second required for image reception. It features 16 independently programmable channels that can be initiated by and synchronised with, among other things, the CPU, external interrupts and timing events.

In the LoVIE design, the EDMA is used not only to transfer image data within the external SDRAM, but also to transfer received image data from the 2 McBSP channels onto the SDRAM and to transfer the output image to the display subsystem via the EMIF. The advantage of this is that once set up, the EDMA does not require the CPU to carry out the transfer. In this manner, the LoVIE is able to process one frame while simultaneously receiving the next.

4.2.3.4 External Hardware Interrupts

To enable control by external devices or users, the C6711 features 4 3.3V external pins available as interrupt sources that can be used to trigger software events or provide synchronisation information.

In the current LoVIE design, external interrupts 4 and 5 are wired to the pixel clock output of the M3188A to act as synchronisation events for the EDMA transfers from the McBSP channels to the SDRAM. External interrupts 6 and 7 are used as parameter adjustment buttons in the current implementation of the user interface.

4.2.4 Design Environment

All source code design, debugging and testing on the C6711 was done within the Texas Instruments Code Composer Studio design environment (Version 2.00.00). Code Composer features a fully integrated C/C++ compiler, assembler, linker and visual linker.

To facilitate debugging, Code Composer Studio supports simple, conditional and hardware breakpoints as well as a symbol browser and an advanced watch window, all of which were used extensively during development.

To provide simple control over the various peripherals available on the C6711, Code Composer Studio also includes a DSP/BIOS kernel that features interrupt handling, a chip support library and several DSP libraries. The graphical user interface of the DSP/BIOS proved extremely useful in configuring the EDMA transfers and McBSP channels as they provide a simple and clear view of the relevant registers.

While at times quite unstable, the Code Composer Studio package has many features that made it invaluable while implementing the various LoVIE designs. The graphical

interface proved especially useful when configuring the various peripherals, and thus was considered a suitable software environment for the implementation of the LoVIE.

5. Camera Selection and Evaluation

5.1 Camera Requirements

As the processing carried out by the LoVIE depends heavily on image quality, the choice of an appropriate image sensor was a critical aspect of the design process. Before the selection process could begin, a number of primary and secondary requirements for the image sensor itself were established.

First and foremost, the selected image sensor needed to give output of an acceptable quality. It was decided that colour video output would be unnecessary for the intended purpose, as it would only increase the visual complexity of the image, as well as the time required to process it. With finding a balance between image quality and required processing time in mind, it was decided that an output resolution of 640x480 pixels with a depth of 256 grey levels per pixel would be more than sufficient. As the LoVIE will be expected to output a fairly smooth video stream to maintain clarity, the output frame rate should be at least 15 fps.

The second requirement was that the image sensor output the image data in an appropriate format via an appropriate interface. The appropriate inputs available on the TMS320C6711 device are limited to 2 McBSP channels, an external memory interface, and four hardware interrupts (See Section 4.2.3). With this in mind, and to offer the maximum flexibility, it was desired that the chosen image sensor offer output via a data bus of at least 8 bits width for parallel transfers of each pixel. Any control data lines

were to be tied to the external hardware interrupts. The TMS320C6711 features 3.3V I/O, buffered up to 5 volts, so it was preferable that the chosen image sensor be compatible with this to negate the need for additional circuitry.

As the LoVIE is eventually intended to be a low cost portable device, power consumption was a major concern in selecting an appropriate image sensor. While CCD image sensor chips are less sensitive to noise than equivalent CMOS chips [11], they can consume as much as 100 times more power than an equivalent CMOS device and are also fairly expensive. Because of this, it was decided that a CMOS image sensor would be the more appropriate choice. If noise sensitivity does become a problem further along in development, it is expected that an equivalent CCD device could be substituted into the existing interface framework with few problems.

Secondary requirements were established as a means of choosing between several devices that matched the needs outlined above. Although not entirely necessary, a means of controlling the camera via the TMS320C6711 (preferably using the I^2C bus protocol) would be convenient. If available, this would provide a means of adjusting the camera's frame rate, contrast, and brightness among other things, without additional processing.

Preferably, the CMOS image sensor chip would come packaged with all components necessary to drive it. While the necessary circuitry could be implemented if required, this would be unnecessarily time consuming and hence would decrease development time available for more important aspects of the LoVIE.

An auto focus feature would increase the versatility of the camera, but would most likely increase the power consumption, size and weight of the camera package. If deemed necessary later in development, a compatible lens incorporating auto focus could be obtained.

5.2 The M3188A CMOS Camera Module

The M3188A camera module [12] incorporates the OmniVision OV7120 1/3" CMOS image sensor [8] along with all discrete components necessary to drive it. It features an output resolution of 640x480 pixels with 256 grey levels (8 bits) per pixel available at up to 30 fps. The image data is available via an 8-bit bus line synchronised with an internally generated pixel clock. Data can be output in various formats, including interlaced and progressive scan modes. All I/O on the module operates at 5V; thus the M3188A meets or exceeds all primary requirements outlined in Section 5.1.

Additionally, the M3188A provides various functions including frame rate, contrast and brightness adjustment as well as windowing and noise suppression, configurable via an I^2C interface. Many functions available via the I^2C interface can also be enabled automatically on power-up by attaching pull-up or pull-down resistors to specified pins. This may be useful further on in the development process to reduce the time required to configure the camera at start up, or possibly negate the need for an I^2C interface altogether.

Unfortunately, the M3188A comes packaged with a fish-eye lens (f = 3.7mm), probably because the module is intended for security applications. While this was not entirely suitable for viewing written material at distances greater than about 3 metres, it was sufficient for development and testing purposes. Further on in development, a compatible lens with a focal length of about 12mm [13] will need to be obtained.

5.3 Camera Interface Methods

5.3.1 Timing Overview

In order for the interface methods described below to be meaningful to the reader, a brief description of the timing and purpose of the output signals from the M3188A will be given. The OV7120 is configured to operate in progressive scan mode for all timing and interface descriptions given.

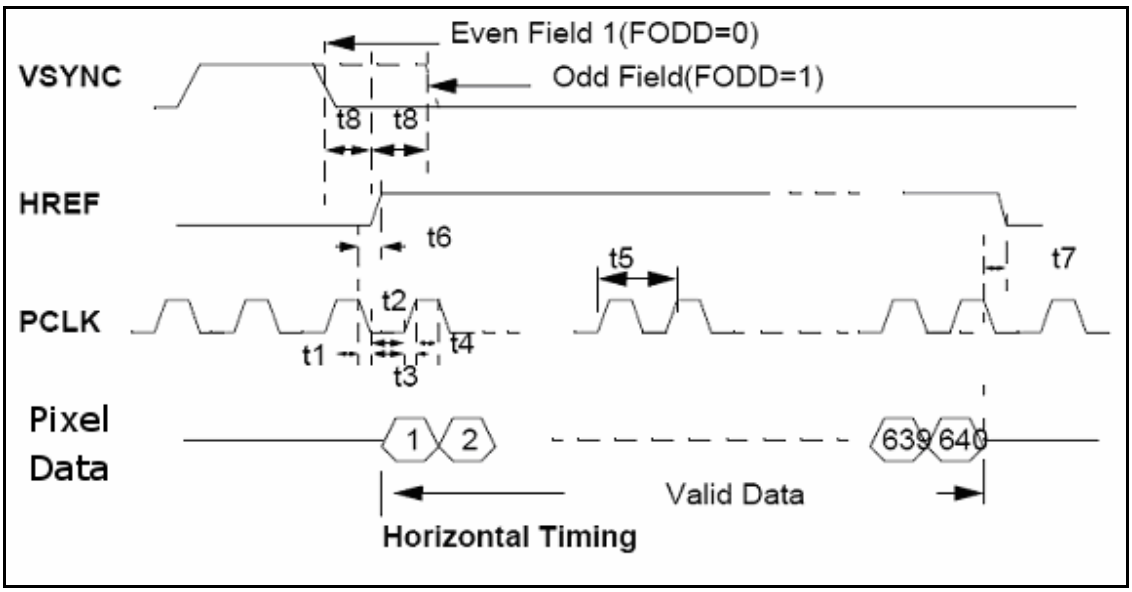


Figure 5.1 Timing Diagram for the M3188A

As shown in Figure 5.1, the M3188A requires three control lines and one 8-bit data bus be utilised for correct operation. The VSYNC line goes high momentarily to indicate the start of each frame while the HREF line is high for an entire line (640 pixels) of valid data. Pixel data can to be read off the data bus on each rising edge of the pixel clock.

5.3.2 Interrupt Controlled Data Transfer via McBSP

This method was used on the first attempt to interface the M3188A with the C6711 DSP platform; unfortunately it is also the method that produced the poorest results. As the image data was to be received from the M3188A via an 8-bit bus line, it was concluded that the best way to receive the data on the C6711 was via the McBSP configured to act as a set of independent general-purpose binary input/output pins (See Section 4.2.3.2). On receiving each pixel, the C6711's EDMA was to be used to transfer the data to an appropriate location in memory.

As this was the first attempt at using the C6711's EDMA in a practical sense, nothing about the limitations of the engine were known apart from those provided in the various technical documents available [14],[15]. For this reason, it was decided that processing of the various signals from the camera would be done with the EDMA directly. Specifically, this involved linking the HREF and VSYNC lines to external interrupts 6 and 7 respectively, with the pixel clock attached to external interrupts 4 and 5 and pixel data received via the McBSP as shown in Figure 5.2 below.

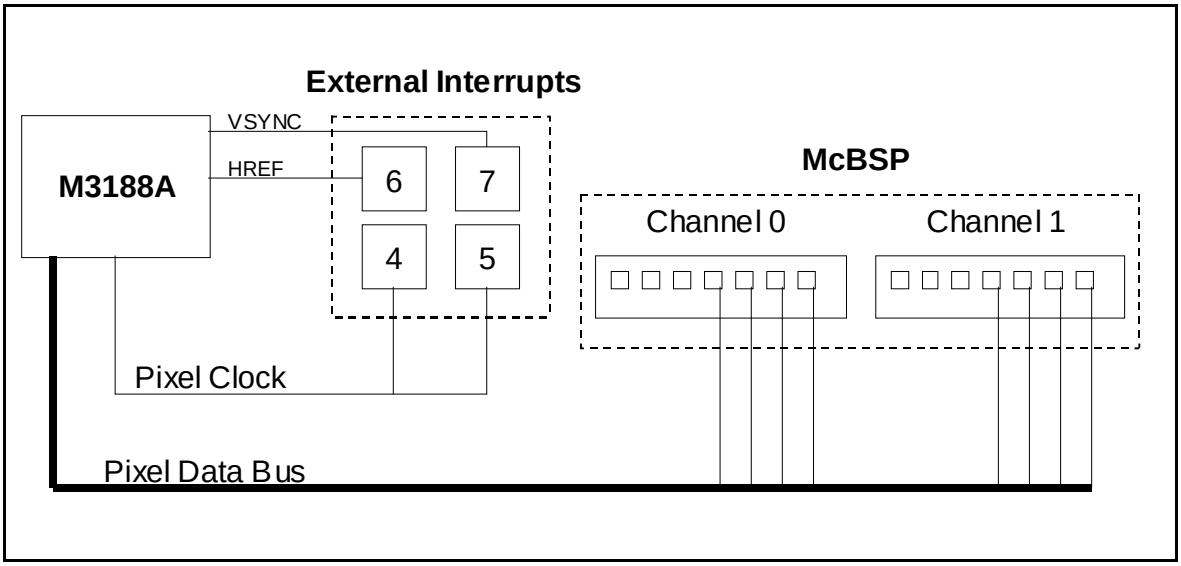


Figure 5.2 Wiring Diagram for Interrupt Controlled Data Transfer

Essentially, once the processor was ready for a new frame, the VSYNC interrupt would be enabled. On receiving this interrupt, the EDMA was configured to transfer one line of pixel data (640 pixels) starting at the first address in the frame buffer, and the HREF interrupt was enabled. On receiving an HREF interrupt, the EDMA transfer was started, synchronised with the pixel clock interrupt. Once the line of data was transferred, the destination address of the EDMA transfer was increased by 640 bytes and the transfer counter reset, ready for a new line of data. Once 480 lines of data were received in this manner, the HREF and VSYNC interrupts were disabled until a new frame was required.

As each McBSP channel features only 6 pins capable of receiving binary data independently, it was necessary to use 2 McBSP channels to receive all 8 bits of data. As

such 2 concurrently running EDMA transfers were needed to transfer this data, which caused several problems.

Firstly, while the EDMA engine on the C6711 was more than capable of maintaining the bandwidth required for transferring 15 frames per second (approximately 37 Mbps), 2 simultaneous transfers from the McBSP registers involved transferring 8 redundant bits for every 8 data bits. Tests carried out running 2 concurrent EDMA transfers with various configurations revealed the maximum obtainable effective bandwidth to be approximately 31 Mbps, while in reality the actual achieved bandwidth was closer to 62 Mbps. When combining this limitation with the delays involved with constantly reconfiguring and restarting the EDMA transfers, the highest frame rate obtainable with this interface method was 3 frames per second. The second significant problem involved the resultant image itself. While the image as a whole was quite clear, it seems the transfer introduced speckled areas (Figure 5.3) that tended to adversely affect the implemented filters. Also it seems that in the transfer process, several levels of grey were lost, distorting areas of low contrast gradient. The reason these spurious pixels were being introduced into the image could not be ascertained so it was assumed that random bits were being lost due to the strain placed on the EDMA engine.

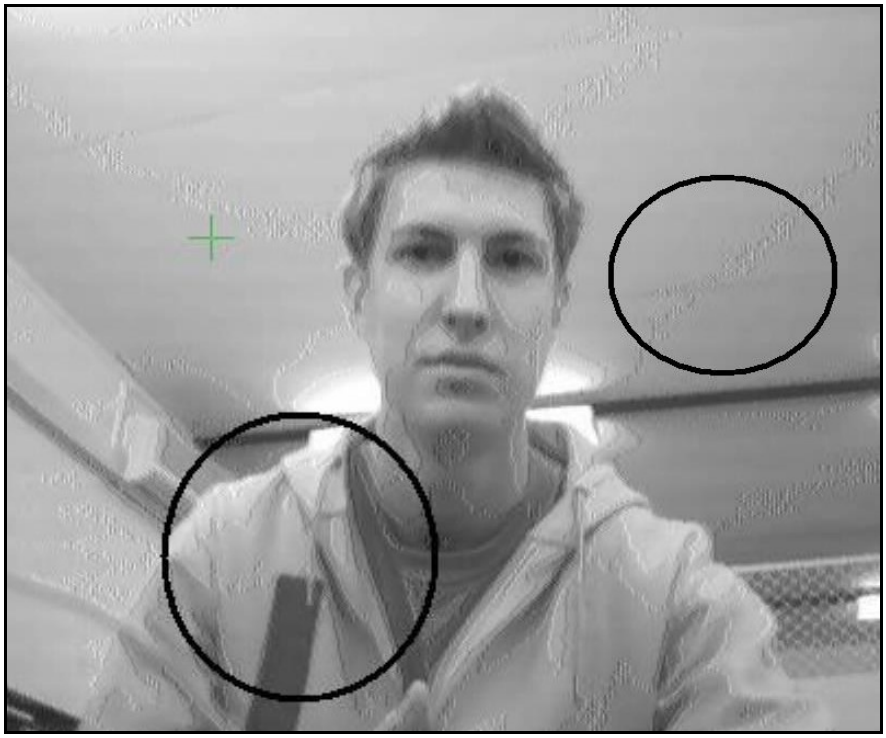


Figure 5.3 Example Image Obtained via Interrupt Controlled Interface with Problem Examples Circled

Due to the reasons outlined above, this interface method was deemed to be unsatisfactory for its intended purpose. Therefore, further work was done to develop the alternative methods outlined below.

5.3.3 Interrupt Triggered Gated Pixel Clock Signal via McBSP

At this stage of development, the main reason for the low frame rate achieved was thought to be the constant initialising and reconfiguring of the EDMA necessary in the interface method outline in Section 5.3.2. As such, 2 methods of eliminating the need for these reconfigurations were devised.

The first method discussed involved simply dumping all pixel data onto the SDRAM. The states of VSYNC and HREF were embedded in this data simply by attaching these lines to spare McBSP pins. Once an entire frame was received, all necessary processing was to be carried out by the CPU. However, it was quickly discovered that this method was redundant, as the extra processing time required extracting the frame would negate any resulting gain in transfer speed.

The second method attempted, involved using a standard AND gate to combine the signals from the pixel clock and the HREF line. The resulting output would give a positive edge only when valid pixel data was available. The only restriction on the AND gate used was that its propagation delay be small enough, that it did not cause the data bus and pixel clocks output to be misaligned. At 3 fps, the OV7120 pixel clock runs with a period of roughly 727 nanoseconds. As the OV7120 guarantees valid pixel data on the positive edge of the pixel clock only, it was established that the propagation delay of the AND gate should not be more than one tenth of the pixel clock period. The AND gate used in the implementation of the interface method was the 74HC08 Quad 2-input AND gate. The 74HC08 has a propagation delay around 7ns [16], far below the limit established and was also readily available in the technical storeroom; thus, it was deemed suitable.

With this method implemented, the EDMA need only be configured and started once, at the start of each frame. It would receive only pixel data necessary for the frame and trigger an interrupt once the transfer was complete.

The output images obtained with the AND gate implementation contained none of the spurious pixels found in the results of the previous method, thus in this respect the AND gate implementation was successful. However, for reasons unknown the frame was now misaligned, giving an output image similar to that from a television with a badly adjusted vertical hold (See Figure 5.4).

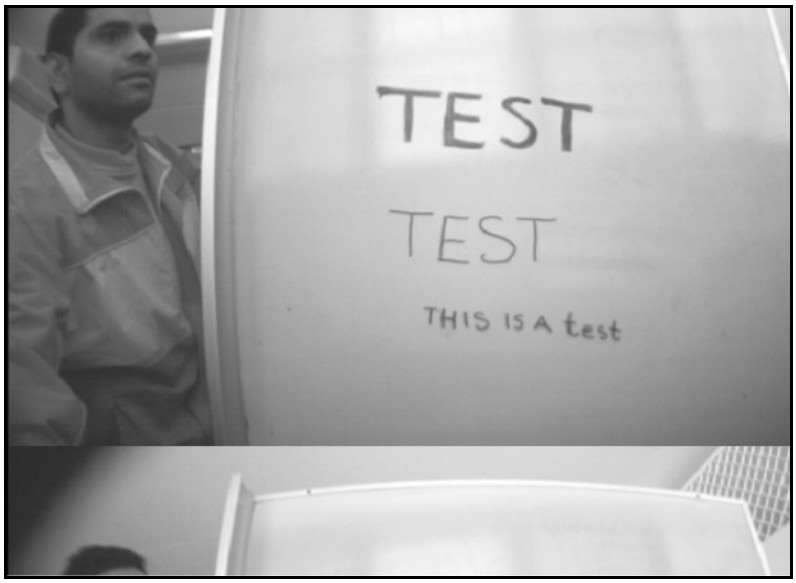


Figure 5.4 Example Resultant Frame from Interrupt Triggered Gated Pixel Clock Interface

As mentioned in Section 5.3.2, the EDMA transfer is triggered on receiving the VSYNC interrupt. The measured delay for an interrupt routine on the C6711 was around 880ns, more than fast enough to set up and start the EDMA transfer before the M3188A begins to transmit data. Consequently, the problem was not hardware related on the C6711's part. Several tests were run, but the source of this particular problem could not be found. For this reason the Interrupt Triggered Gated Pixel Clock interface was not suitable for

use in the final implementation. Methods of guaranteeing proper frame synchronisation were investigated, resulting in the interface design discussed in Section 5.3.4.

5.3.4 SRAM Mode with Gated Pixel Clock

In order to address the frame synchronisation problem discussed in Section 5.3.3, a method needed to be devised that would ensure the EDMA transfer began on the first pixel of the frame every time. The OV7120 datasheet [8] states that by setting a particular I^2C register, the image sensor can be put into *SRAM* mode. Once in *SRAM* mode, the OV7120 goes into a wait state with the VSYNC line high and the data bus tri-stated. While in this state, a single frame can be output from the OV7120 by requesting it via the I^2C interface or an external pin on the chip itself. As the McBSP is currently tied up receiving pixel data, an I^2C controller on the C6711 has as yet not been implemented; as such the frame is requested by sending a pulse to the appropriate pin on the OV7120.

Since this method means the data is available as soon as requested, and not as a part of a continuous stream, there are no synchronisation issues. Also, as the signal is still run through the AND gate, as described in Section 5.3.3, the resulting output consists solely of one frame of valid data. Operating the OV7120 in this mode simplifies the reception of data on the C6711 greatly. Once a frame is required, the EDMA is configured to transfer 307200 pixels from the McBSP to an appropriate location in memory. Once the set up is complete, a pulse is sent via one of the McBSP output pins to the appropriate pin on the OV7120. Once all necessary data is received, the EDMA triggers a software interrupt that updates all necessary flags and begins processing the frame.

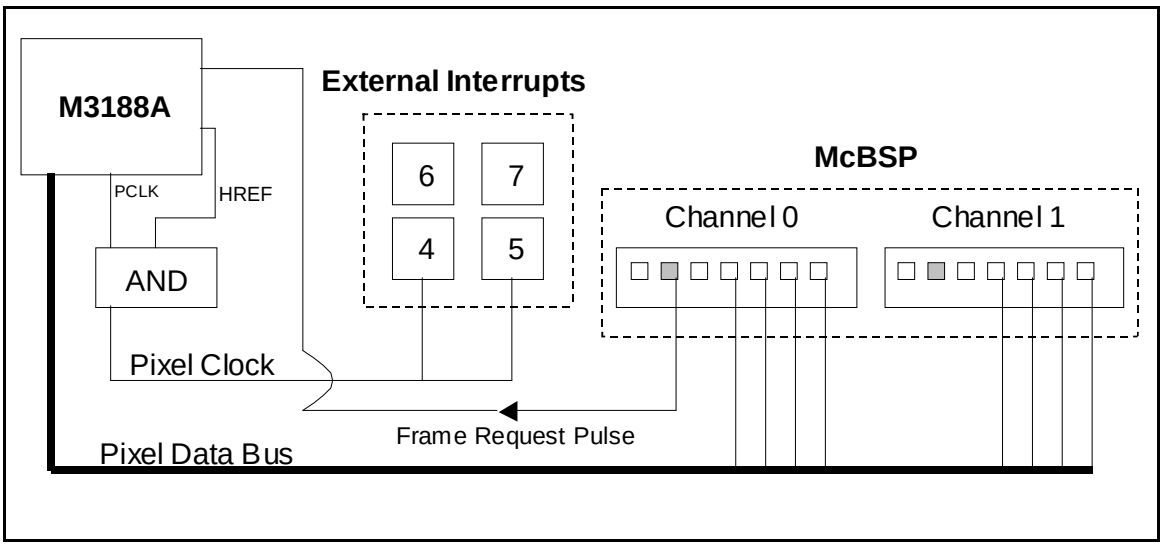


Figure 5.5 Wiring Diagram of SRAM Mode with Gated Pixel Clock Interface

As this interface method resolves all timing and processing issues inherent in the previous interface methods, Figure 5.6 shows optimal image quality can be attained with relatively low processing requirements on the C6711. Unfortunately, as this method still uses the McBSP to receive the data, the same speed limitations outlined in the previous methods still apply.

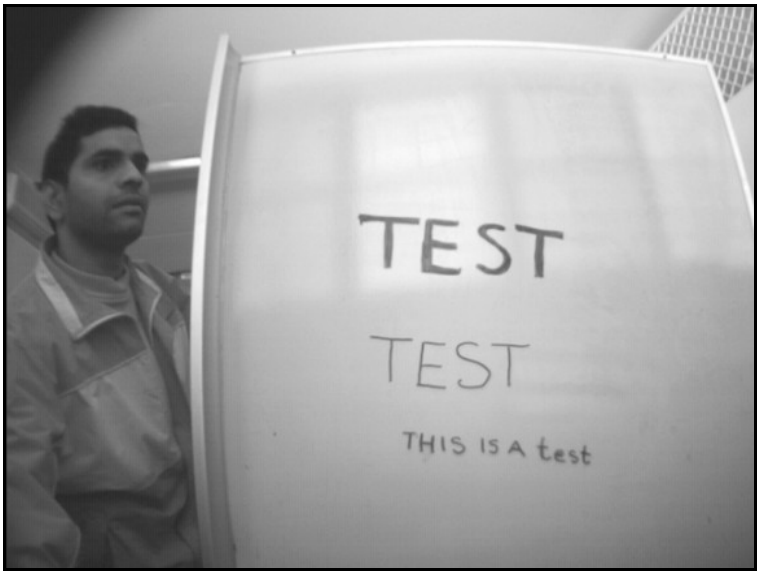


Figure 5.6 Resultant Image from SRAM Mode with Gated Pixel Clock Interface

6. User Interface

6.1 User Interface Design

As the LoVIE is intended for people with poor vision, it must feature a simple, and above all, clear method for accessing all user controllable parameters.

Currently, there are 4 main enhancements available; Focus, Edge Enhance, Zoom and Threshold. Aside from enabling and disabling the enhancements, each has at least 1 user definable parameter. Focus requires the user to be able to adjust the location and width of the focus area. Thresholding allows not only the threshold level to be adjusted, but also the foreground and background colours of the output image. Zoom can be adjusted to give different magnification factors, while the Edge Enhance filter allows the user the ability to adjust the cut off frequency, as well as the foreground and background amplifications. A visual representation of the effects and their parameters is given in Figure 6.1.

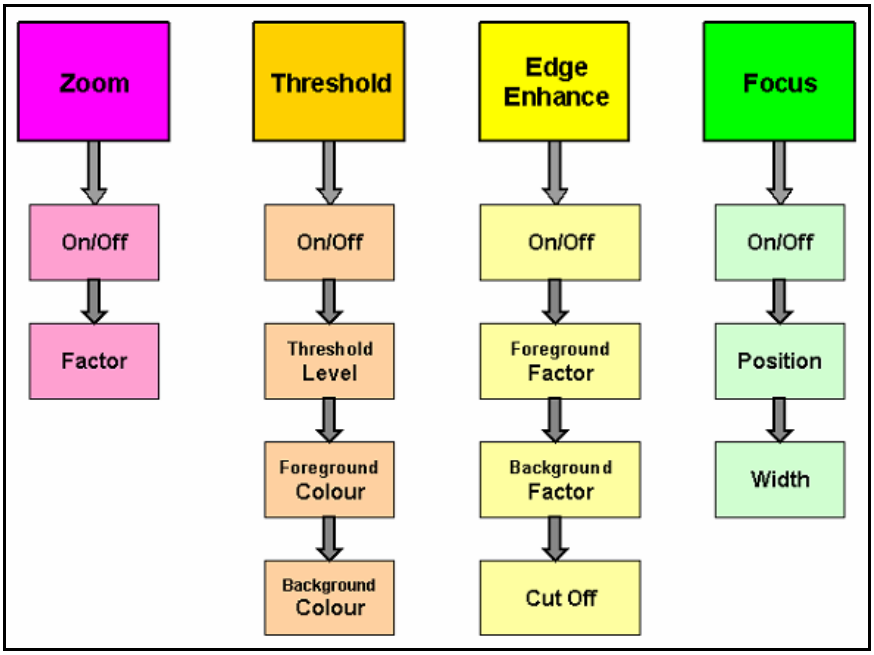


Figure 6.1 Flow Diagram of User Interface Design

To create a simple interface, it was decided that the controls should be based around 4 function buttons and 2 adjustment buttons. Each individual function button can be used to cycle through its assigned enhancement parameters while the adjustment buttons can be used to alter the selected parameter. With one separate button for each function, they could be quite large and clearly labelled and/or colour coded. Also as the same two buttons are used to adjust all parameters, with practice the LoVIE could be operated on tactile memory alone.

6.2 User Interface Implementation

With the current hardware implementation of the LoVIE, there are 3 general-purpose input/output pins available (via the McBSP) and 2 hardware interrupts. When triggered,

the hardware interrupts can interrupt software flow, therefore their current states are available at any time. On the other hand, the 3 McBSP pins must be continually polled in order to obtain their state. Due to this, it was decided that the 3 McBSP pins be used as the 4 main function buttons, while the two interrupts be used as the adjustment buttons.

As 4 function buttons are required with only 3 McBSP pins, a combination of the 3 pin states is used to trigger each function. While only 2 pins would be sufficient, the 3 McBSP pins can support up to $2^3 = 8$ functions, allowing room for future developments. The corresponding registers for the three available pins are DR0, CLKS0 and DR1, a truth table and wiring diagram of the current configuration is given below.

DR0	CLKS0	DR1	FUNCTION
0	0	1	Threshold
0	1	0	Edge
1	0	0	Zoom
1	0	1	Focus

Figure 6.2 Truth Table for User Interface Register Configuration

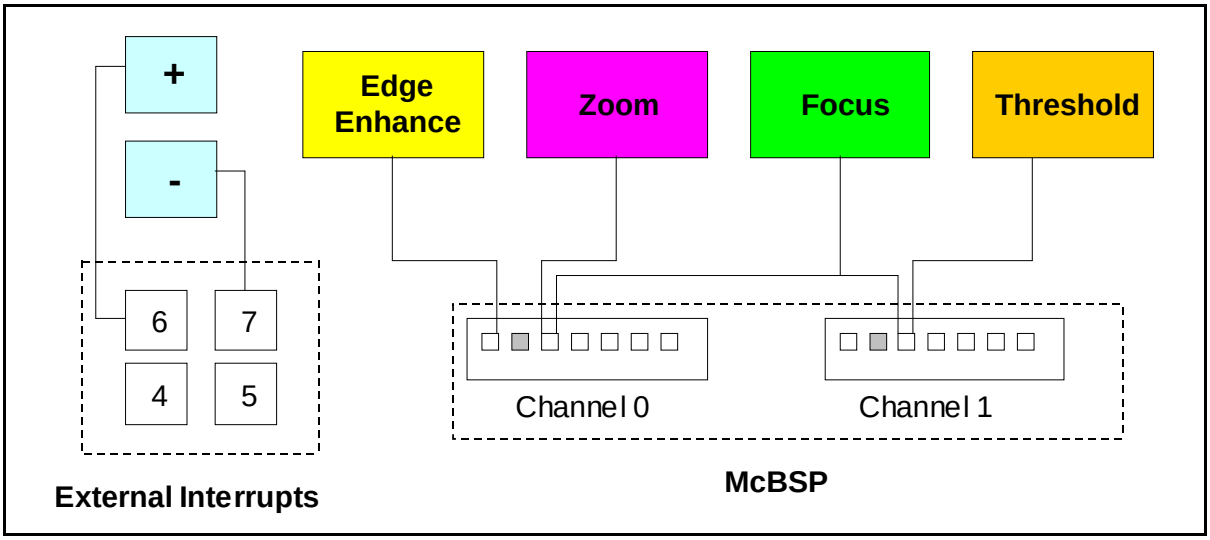


Figure 6.3 Wiring Diagram for User Interface

The 3 pins are polled once at the start of every frame processing cycle. While the two 2 adjustment buttons are updated instantly on triggering, the modified parameters will not take effect until the modified filter is next applied.

7. Conclusion

7.1 Progress Summary

The LoVIE preliminary design involved in-depth research into several areas of image enhancement techniques, including fuzzy processing, image segmentation and a number of domain-transform methods. Because of this research, it was possible to create several custom filters, from the ground up, designed specifically for use with the LoVIE.

Currently, there are 4 filters in use on the system – Focus, Edge Enhance, Zoom and Threshold. A fifth, Edge Outline, has been implemented but is as yet untested. While some of these filters, namely Edge Enhance and Zoom, are not yet running at speeds that could be classified as “real-time”, this is definitely an achievable goal. Theoretically the hardware platform used in development is more than capable of the calculations required; however due to time constraints, no time has yet been put into improving and optimising the code involved.

Concerning the image sensor hardware, an adequate interface between the M3188A and the C6711 was implemented. While the current interface in use limits the transfer of images to 3 frames per second, it is felt that a far more efficient system design is achievable. Some research was carried out on the subject, but again due to time constraints, the matter was not developed further.

7.2 Future Developments

7.2.1 Overview

While a basic design for the LoVIE has been defined, there are several aspects of the system that need to be improved or developed further before the device could be considered ready for commercial use. The following sections outline several such areas where further work is necessary or possible.

7.2.2 Optimisation

As mentioned previously, the existing hardware configuration allows up to 3 frame transfers per second. Unfortunately the current implementations of the designed filters are not capable of running at these speeds. Ultimately, a realistic goal for the processing system should be around 15 frames per second. However, it is expected that the final frame rate be variable, depending on the processing load. As yet no optimisation work has been done at all, thus a fairly basic initial optimisation process should result in a significant speed improvement. Further, as many filters involve 2D convolution, it is suggested that fast 2D convolution methods such as the *nesting*, *splitting* and *polynomial* transforms would be a productive area of investigation.

7.2.3 Camera Interface via C6711 EMIF

As a possible solution to the frame rate limitation inherent in the current interface methods, it is proposed that by using an FPGA as an external controller, the M3188A could be interfaced with the C6711 via its external memory interface (See Section 4.2.3.1). The FPGA would buffer incoming data from the M3188A and retransmit it to

the C6711 via a 32-bit data bus on request. Doing this would eliminate the need for 2 simultaneous EDMA transfers, or any pre processing of the frame upon reception. It is expected that by doing this, rates of up to 15 fps would be achievable.

Though not yet implemented, a similar set up has been successfully used in the display subsystem interface [20]. Aside from improving on the current achievable frame rate, this rather more elegant solution would have the added bonus of freeing up the McBSP, enabling it to be used as an I^2C controller.

7.2.4 On Board I^2C Controller

The current camera interface method requires the use of both McBSP channels to function correctly. As such, all I^2C register sets on the M3188A were controlled during development via an external controller (namely the Z8 Encore Evaluation Board). Freeing up the McBSP channels by making use of the C6711's EMIF for the interface would mean an on board I^2C controller could be implemented on the C6711 relatively easily.

7.2.5 Appropriate Filter Testing and Further Design

While all filters implemented on the LoVIE were designed based on the findings of others or on the advice of people experienced in working with low vision students, they are yet to be properly tested. Tests carried out on low vision individuals within an appropriate environment would provide valuable feedback on the real world usefulness of

the current filters and possibly provide ideas for improvements and/or further enhancements.

7.2.6 Implementation of Dithering Algorithm on Greyscale Output

The display used in the LoVIE design is not capable of displaying any more than 64 levels of grey. Currently, to format the output appropriately, the 2 least significant bits of each output pixel are discarded before transferring it to the display subsystem. While the results achieved using this method are satisfactory, a noticeable improvement could be gained by first dithering the image to 64 grey levels before discarding the unused bits.

It may be suggested that since the display is capable of using only 6 bits per pixel, then attempting to transfer 8 bits from the image sensor is unnecessary. However, some of the filters implemented, particularly zoom and threshold, are quite sensitive to changes in contrast. Thus it was felt that better results could be achieved by maintaining optimal image quality while applying these filters.

7.2.7 Implementation of Edge Preserving Interpolation Methods

It is suggested that the final zoom implementation make use of the many available edge preserving interpolation techniques [17],[18]. Edge preserving interpolation acts as normal interpolation techniques but attempts to preserve the sharpness of edges and thus overall clarity of a zoomed image. The sharper the edges obtained from the output of the

digital zoom, the more effective the performance of the Edge Enhance and Threshold filters.

7.2.8 Further User Interface Development

At this point in time the user interface provides no feedback as to the parameter selected or its adjusted value. It is suggested that such feedback be implemented, possibly in the following manner. Firstly, a large on screen display should be implemented to give a clear visual representation of the current operation. Additionally, the LoVIE could make use of audible cues to indicate the current operation being performed by the user.

7.2.9 Advanced Image Processing Techniques

While the filters currently implemented are on par with, and in some respects exceed, the capabilities of equivalent devices on the market, there is room for improvement. The ultimate goal of the LoVIE is to be able to intelligently recognise the information desired, even going so far as to recognise what is text and what isn't. Were the device capable of this, all interfering image components could be removed completely. While this is not yet proven to be feasible, research into a number of areas including image segmentation and fuzzy image processing could prove quite beneficial in this respect.

8. References

[1] Health Insite, Australian Government, "Low Vision Conditions," [online] October 2004, http://www.healthinsite.gov.au/topics/Low_Vision_Conditions (Accessed: 2 November 2004).

[2] The Association for the Blind of W.A. (Inc.), "Understanding Blindness and Eye Conditions," [online] September 2004, http://www.abwa.asn.au/understandingblindness.html (Accessed: 2 November 2004).

[3] AllRefer.com, "AllRefer Health - Diabetic Retinopathy," [online] 2003, http://health.allrefer.com/health/diabetic-retinopathy-info.html (Accessed: 2 November 2004).

[4] E.M. Fine and E. Peli, "Enhancement of Text for the Visually Impaired," *Journal of the Optical Society of America*, vol. 12(7), pp. 1439-1447, July 1995.

[5] Bob Fisher, Simon Perkins, Ashley Walker and Erik Wolfart, University of Edinburgh, "Feature Detectors – Sobel Edge Detector," [online] 1994, http://www.cee.hw.ac.uk/hipr/html/sobel.html (Accessed: 2 November 2004).

[6] R.L. Windsor, L.K. Windsor, The Low Vision Centers of Indiana, "Understanding the Visual Problems of Retinitis Pigmentosa,"

http://www.eyeassociates.com/images/understanding_the_visual_problem1.htm (Accessed: 2 November 2004).

[7] Bob Fisher, Simon Perkins, Ashley Walker and Erik Wolfart, University of Edinburgh, "Image Arithmetic – Logarithm Operator," [online] 1994, http://www.cee.hw.ac.uk/hipr/html/pixlog.html (Accessed: 2 November 2004).

[8] OmniVision Technologies Technical Staff, OV7620/OV7120 Product Specifications – Rev. 1.2, OmniVision Technologies, Inc., July 10 2001.

[9] Texas Instruments Technical Staff, TMS320C6711 Datasheet – SPRS088L, Texas Instruments Incorporated, May 2004.

[10] Texas Instruments Technical Staff, TMS320C6000 DSP Peripherals Overview Reference Guide – SPRU190G, Texas Instruments Incorporated, September 2004.

[11] HowStuffWorks, "What is the difference between CCD and CMOS image sensors in a digital camera?," [online] 2004, http://electronics.howstuffworks.com/question362.htm (Accessed: 2 November 2004).

[12] Amazon Electronics, M3188A Datasheet, Amazon Electronics Inc., 2004.

[13] CCTVConsult.com, "Focal Length and Angle of View," [online] 2004, http://www.cctvconsult.com/pages/angle.htm (Accessed: 2 November 2004).

[14] J. Bowen and J. Ward, TMS320C621x/TMS320C671x EDMA Architecture, Texas Instruments Incorporated, March 2004.

[15] Texas Instruments Technical Staff, TMS320C6000 DSP Enhanced Direct Memory Access (EDMA) Controller Reference Guide – SPRU234, Texas Instruments Incorporated, July 2003.

[16] Philips Semiconductors Technical Staff, 74HC08; 74HCT08 Quad 2-input AND gate Datasheet, Philips Semiconductors, July 25 2003.

[17] Thiadmer Riemersma, ITB CompuPhase, "Quick Image Scaling by 2," [online] August 2004, http://www.compuphase.com/graphic/scale2.htm (Accessed: 2 November 2004).

[18] S.E. El-Khamy (et al), "A new edge preserving pixel-by-pixel (PBP) cubic image interpolation approach," in *National Radio Science Conference, 2004. NRSC 2004. Proceedings of the Twenty-First*, 2004, pp. 337-345.

[19] Arulliah, E, “Image Acquisition and Processing on the Low Vision Image Enhancer,” Undergraduate thesis, Curtin University of Technology, 2004

[20] Lowe, J, “Display Subsystem Development for the Low Vision Image Enhancer,” Undergraduate thesis, Curtin University of Technology, 2004

Appendix A – C Source Code

```
/* SRAM mode with gated pixel clock interface source code*/
/* requires main_cfg.c to configure hardware             */

#define CHIP_6711

#define _TI_ENHANCED_MATH_H 1
#include <math.h>
#include <std.h>
#include <stdio.h>
#include <log.h>
#include <hwi.h>
#include <csl.h>
#include <csl_irq.h>
#include <csl_mcbsp.h>
#include <csl_edma.h>
#include "main_cfg.h"

//first define memory locations
#define FRAME_ONE_A 0x800A9000 //Memory location of frame
#define FRAME_ONE_B 0x800F5000 //buffers

#define FRAME_TWO_A 0x80141000 //Have left 0x1000 between
#define FRAME_TWO_B 0x8018D000 //each data segment

#define PROCESSED_FRAME_START 0x801D9000 //start of actual frame
#define FINAL_FRAME     0x80225000  // start of processed frame

//list of globals needed
int EDMA_finished,
new_frame,
count,
EDMA_number,
CPU_number,
CPU_finished;


int swap_times;  //if we only want it to swap a certain number of times

unsigned char *processed_start =(unsigned char *)PROCESSED_FRAME_START,
              *PCR0            = (unsigned char *)0x018C0024;
unsigned int  *PCR1            = (unsigned int *)0x01900024;

unsigned char *EDMA_frame_a, //pointer to data segment A for EDMA
              *EDMA_frame_b,       //pointer to data segment B for EDMA


int enable;

/*function declarations*/
void initialise(void); //setup variables/interrupts. run once at start
void EDMA_frame_finished(void);//called when EDMA transfer completed
void CPU_process(void);//post processing of frame


void main(void)
{
      initialise();
      for (count=0;count<0x4d000;count++)
      {
```

```
            *((unsigned char *)FRAME_ONE_A+count)=0x00;
            *((unsigned char *)FRAME_ONE_B+count)=0x00;
            *((unsigned char *)FRAME_TWO_A+count)=0x00;
            *((unsigned char *)FRAME_TWO_B+count)=0x00;
            *((unsigned char *)PROCESSED_FRAME_START+count) = 0x00;
      }
      count=0;

      while(EDMA_finished<15) //transferring 15 frames
      {
            if (new_frame==0)
            {
                  EDMA_enableChannel(hEdmaExtint4); //first set up
                  EDMA_enableChannel(hEdmaExtint5); //EDMA transfer
                  new_frame=1;
                  *PCR1 = 0x3010; //then send a pulse to the M3188A
                  *PCR1 = (*PCR1 | 0x3030);
            }
            CPU_process(); //once the frame is received, post-process it
      }
      EDMA_close(hEdmaExtint4); //once all frames are received
      EDMA_close(hEdmaExtint5); //tidy up and finish
      MCBSP_close(hMcbsp1);
      MCBSP_close(hMcbsp0);

      printf("finished and all over \n");
}

void initialise(void)
{

      EDMA_finished=0;
      new_frame=0;
      EDMA_number = 1;
      CPU_number = 0;
      CPU_finished = 1;

      IRQ_resetAll();    //clear all interrupts
      EDMA_intClear(1);
      EDMA_intEnable(1); //enable TCINT 1

      IRQ_enable(IRQ_EVT_EDMAINT);
      IRQ_globalEnable();
}



void EDMA_frame_finished(void)
{
      //this function is called by the edma when it is finished a frame

      EDMA_disableChannel(hEdmaExtint4);
      EDMA_disableChannel(hEdmaExtint5);
      printf("frame finished %d\n",EDMA_finished);
      EDMA_intClear(1);//clear the cipr register so this
      //function will trigger again

      if(CPU_finished) //once the frame finishes, switch buffer
      {
            if(EDMA_number == 1)
            {
                  EDMA_number = 2;
                  CPU_number = 1;
                  edmaCfg0.dst = FRAME_TWO_A;
```

```
                  edmaCfg1.dst = FRAME_TWO_B;
            }
            else
            {
                  EDMA_number = 1;
                  CPU_number = 2;
                  edmaCfg0.dst = FRAME_ONE_A;
                  edmaCfg1.dst = FRAME_ONE_B;
            }
            CPU_finished = 0;
      }
      EDMA_config(hEdmaExtint4,&edmaCfg0); //reset the EDMA channels
      EDMA_config(hEdmaExtint5,&edmaCfg1);      //for each frame

      EDMA_finished++;
      new_frame = 0;
}

void CPU_process(void) //perform post processing of frame
{
      long int read_counter;
      int i;
      unsigned char *CPU_frame_a,
            *CPU_frame_b;
      if(CPU_number != 0)
      {
            if(CPU_number==1)
            {
                  CPU_frame_a = (unsigned char *)FRAME_ONE_A;
                  CPU_frame_b = (unsigned char *)FRAME_ONE_B;
            }
            else
            {
                  CPU_frame_a = (unsigned char *)FRAME_TWO_A;
                  CPU_frame_b = (unsigned char *)FRAME_TWO_B;
            }
            //this for loop reads both half pixels at once
            //combines them into a single pixel and then
            //writes the output to processed_start

            for(read_counter=0;read_counter<307200;read_counter++)
            {
                  //entire process done in one line to save time
                  *(processed_start+read_counter)                        =
((*(CPU_frame_b+read_counter) & 0x0F)<<4)|(*(CPU_frame_a+read_counter) &
0x0F);
            }


            CPU_finished = 1;

      }
```

```
/*Hardware setup file generated for SRAM interface */
/*generated automatically by Code Composer.*/
/*This file is included solely to illustrate the*/
/*hardware    configuration    used    by    this    interface.    */
/*                                                  */
/*   Do *not* directly modify this file.  It was    */
/*   generated by the Configuration Tool; any  */
/*   changes risk being overwritten.                */

/* INPUT main.cdb */
```

```
/*  Include Header File  */
#include "main_cfg.h"

/*  Config Structures */
EDMA_Config edmaCfg0 = {
    0x50310000,         /*  Option  */
    0x018C0024,         /*  Source Address - Numeric   */
    0x01DF0280,         /*  Transfer Counter  */
    0x800A9000,         /*  Destination Address - Numeric   */
    0x00000000,         /*  Transfer Index  */
    0x02800000          /*  Element Count Reload and Link Address  */
};

EDMA_Config edmaCfg1 = {
    0x50200000,         /*  Option  */
    0x01900024,         /*  Source Address - Numeric   */
    0x01DF0280,         /*  Transfer Counter  */
    0x800F5000,         /*  Destination Address - Numeric   */
    0x00000000,         /*  Transfer Index  */
    0x02800000          /*  Element Count Reload and Link Address  */
};

MCBSP_Config mcbspCfg0 = {
    0x00000000,         /*  Serial Port Control Reg. (SPCR)   */
    0x000000A0,         /*  Receiver Control Reg. (RCR)   */
    0x000000A0,         /*  Transmitter Control Reg. (XCR)   */
    0x203F1F0F,         /*  Sample-Rate Generator Reg. (SRGR)   */
    0x00000000,         /*  Multichannel Control Reg. (MCR)   */
    0x00000000,         /*  Receiver Channel Enable(RCER)   */
    0x00000000,         /*  Transmitter Channel Enable(XCER)   */
    0x00003000          /*  Pin Control Reg. (PCR)   */
};

/*  Handles  */
EDMA_Handle hEdmaExtint4;
EDMA_Handle hEdmaExtint5;
MCBSP_Handle hMcbsp0;
MCBSP_Handle hMcbsp1;

/*
 *  ======== CSL_cfgInit() ========
 */
void CSL_cfgInit()
{
    hEdmaExtint4 = EDMA_open(EDMA_CHA_EXTINT4, EDMA_OPEN_RESET);
    hEdmaExtint5 = EDMA_open(EDMA_CHA_EXTINT5, EDMA_OPEN_RESET);
    hMcbsp0 = MCBSP_open(MCBSP_DEV0, MCBSP_OPEN_RESET);
    hMcbsp1 = MCBSP_open(MCBSP_DEV1, MCBSP_OPEN_RESET);
    EDMA_config(hEdmaExtint4, &edmaCfg0);
    EDMA_config(hEdmaExtint5, &edmaCfg1);
    MCBSP_config(hMcbsp0, &mcbspCfg0);
    MCBSP_config(hMcbsp1, &mcbspCfg0);
}
```

```
/*Source code used to implement protoptype user interface*/
/*Currently does not execute filters, only reads in and alters*/
/*parameter values*/

#define CHIP_6711

#include <std.h>
```

```
#include <stdio.h>
#include <log.h>
#include <hwi.h>
#include <csl.h>
#include <csl_irq.h>
#include <csl_mcbsp.h>
#include <csl_edma.h>
#include "uicfg.h"

#define PCR0_ADDR 0x018C0024
#define PCR1_ADDR 0x01900024

//void log_contrast(); //apply log contrast function
//void fuzzy_threshold(); //calculate threhold value
//void zoom(int mode);//zoom image, mode depends on whether thresholding
is enabled
//void edge_enhance();  //enhance edges
//void focus(); //use focus lines
void initialise(); //set camera/interrupts/edma/default params
//void power_down(); //stop everything, sw powerdown camera
void check_UI();
void adjust_threshold();
void adjust_edge();
void adjust_zoom();
void adjust_focus();

typedef struct
{
      unsigned int enable;
      signed int offset;
      unsigned char foreground;
      unsigned char background;
} thresh;

typedef struct
{
      unsigned int enable;
      float strength;
} edge;

typedef struct
{
      unsigned int factor;
} zoom;

typedef struct
{
      unsigned int enable;
      int position;
      int width;
} focus;


int       UI_prev,UI_comb;
int           current_parameter=0;

unsigned char *pcr0 = (unsigned char *)PCR0_ADDR;

unsigned char *pcr1 = (unsigned char *)PCR1_ADDR;
thresh threshold;
edge  edge_enhance;
zoom  zoom_in;
focus focus_lines;
```

```
void main()
{
	//step 1 : set everything up
	initialise();

	//step 2 : begin main loop
	while (1)
	{
		//check user inputs
		check_UI();
		printf("The current UI number is : %d\n",current_parameter);
		//grab frame
		//process frame
		//output frame
	}

	//step 3 : shutdown
	//power_down();

}

void initialise()
{
	threshold.enable=0;
	threshold.offset=0;
	threshold.foreground=0x00;
	threshold.background=0xFF;

	edge_enhance.enable=0;
	edge_enhance.strength=1;

	zoom_in.factor=1;

	focus_lines.enable=0;
	focus_lines.position=240;
	focus_lines.width=20;
	UI_prev=0;
}

void check_UI()
{


	UI_comb=(*pcr0&0x10)>>4|(*pcr0&0x40)>>5|(*pcr1&0x40)>>4;
	if (UI_comb==UI_prev)
	{
		UI_comb=0;
	}
	else
	{
		UI_prev=UI_comb;
	}

	switch(UI_comb)
	{
	case 4	:	if (current_parameter==1)
				current_parameter=2;
		else if (current_parameter==2)
			current_parameter=3;
		else
			current_parameter=1;
		break;

	case 2	:	if (current_parameter==4)
```

```
                              current_parameter=5;
            else
                  current_parameter=4;
            break;

      case 1      :     current_parameter=6;
            break;

      case 5      :     if (current_parameter==7)
                              current_parameter=8;
            else if (current_parameter==8)
                  current_parameter=9;
            else
                  current_parameter=7;
            break;
      default     :     printf("default\n");
      }
}

void positive_interrupt()
{
      switch(current_parameter)
      {
      case 1:
            threshold.enable = 1;
            break;
      case 2:
            threshold.offset++;
            break;
      case 3:
            threshold.foreground=(threshold.foreground+1) % 3;
            break;
      case 4:
            edge_enhance.enable = 1;
            break;
      case 5:
            edge_enhance.strength+=0.2;
            break;
      case 6:
            zoom_in.factor=(zoom_in.factor + 1 )%3;
            break;
      case 7:
            focus_lines.enable = 1;
            break;
      case 8:
            focus_lines.position+=2;
            if((focus_lines.position + focus_lines.width/2)>480)
                  focus_lines.position = 480-focus_lines.width/2;
            break;
      case 9:
            focus_lines.width +=2;
            if(focus_lines.position>240)
            {     if((focus_lines.position + focus_lines.width/2)>480)
            focus_lines.width-=2;
            }
            else
            {
                  if((focus_lines.position - focus_lines.width/2)<0)
                        focus_lines.width-=2;
            }
            break;
            //default;
      }
}
```

```
void negative_interrupt()
{
      switch(current_parameter)
      {
      case 1:
            threshold.enable = 0;
            break;
      case 2:
            threshold.offset--;
            break;
      case 3:
            threshold.background=(threshold.background+1) % 3;
            break;
      case 4:
            edge_enhance.enable = 0;
            break;
      case 5:
            edge_enhance.strength-=0.2;
            break;
      case 6:
            if(zoom_in.factor != 1)
                  zoom_in.factor=(zoom_in.factor - 1 )%3;
            break;
      case 7:
            focus_lines.enable = 0;
            break;
      case 8:
            focus_lines.position-=2;
            if((focus_lines.position - focus_lines.width/2)<0)
                  focus_lines.position = focus_lines.width/2;
            break;
      case 9:
            focus_lines.width -=2;
            if(focus_lines.width<5)
                  focus_lines.width = 5;
            break;
            //default;
      }
}
```

```
/*Hardware setup file generated for user interface */
/*generated automatically by Code Composer.*/
/*This file is included solely to illustrate the*/
/*hardware   configuration   used   by   the   user   interface.   */
/*   Do *not* directly modify this file.  It was    */
/*   generated by the Configuration Tool; any  */
/*   changes risk being overwritten.               */

/* INPUT ui.cdb */

/*  Include Header File  */
#include "uicfg.h"

/*  Config Structures */
MCBSP_Config mcbspCfg0 = {
    0x00000000,        /*  Serial Port Control Reg. (SPCR)   */
            0x000000A0,        /*  Receiver Control Reg. (RCR)   */
            0x000000A0,        /*  Transmitter Control Reg. (XCR)   */
            0x203F1F0F,          /*   Sample-Rate Generator Reg. (SRGR)
*/
            0x00000000,        /*  Multichannel Control Reg. (MCR)   */
```

```
            0x00000000,         /*  Receiver Channel Enable(RCER)   */
            0x00000000,         /*  Transmitter Channel Enable(XCER)   */
            0x00003000          /*  Pin Control Reg. (PCR)   */
};

/*  Handles  */
MCBSP_Handle hMcbsp0;
MCBSP_Handle hMcbsp1;

/*
*  ======== CSL_cfgInit() ========
*/
void CSL_cfgInit()
{
    hMcbsp0 = MCBSP_open(MCBSP_DEV0, MCBSP_OPEN_RESET);
    hMcbsp1 = MCBSP_open(MCBSP_DEV1, MCBSP_OPEN_RESET);
    MCBSP_config(hMcbsp0, &mcbspCfg0);
    MCBSP_config(hMcbsp1, &mcbspCfg0);
}
```

```
//This code implements a simple digital zoom using a variation of
//bilinear interpolation.   The zoom factor and offset can both be
//specified, as required by the LoVIE.

#include <stdio.h>

#define _TI_ENHANCED_MATH_H 1
#include <math.h>
#define input_location  0x80000000
#define output_location 0x8004B018



void main()
{
      int const rows=480;
      int const cols=640;
      float const factor=2.0;
      int y_offset=300,x_offset=270;
      float
average[3][3]={{1.0/15,2.0/15,1.0/15},{2.0/15,3.0/15,2.0/15},{1.0/15,2.0
/15,1.0/15}};

      int row_zoomed,col_zoomed,i_in,j_in;
      int equiv_0_0,
            equiv_0_1,
            equiv_0_2,
            equiv_1_0,
            equiv_1_2,
            equiv_2_0,
            equiv_2_1,
            equiv_2_2;
      int i,j,out_pointer_offset,equiv_input_offset;
      unsigned char * input = (unsigned char *) input_location;
    unsigned char * output = (unsigned char *) output_location;


      row_zoomed=rows/factor;
      col_zoomed=cols/factor;


      for (i=0;i<640*480;i++){
```

```
            *(output+i)=0x00;
      }

      if (x_offset>(480-(480/factor)))
      {
            x_offset=480-(480/factor)-1;
      }

      if (y_offset>(640-(640/factor)))
      {
            y_offset=640-(640/factor)-1;
      }


      for (i=0;i<row_zoomed;i++)
      {

            for (j=0;j<col_zoomed;j++)
            {
                  /*spread the pixels out by the factor specified*/

      *(output+((i*(int)factor*cols)+(j*(int)factor)))=*(input+(((i+x_of
fset)*cols))+(j+y_offset));
            }
    }


    //now to interpolate
    for (i=0;i<480;i++)
    {
            for (j=0;j<640;j++)
            {
            //first we need to know what pixel we are talking about
                  out_pointer_offset=i*640+j;

                  if (*(output+out_pointer_offset)==0x00)
                  {

            //calculate the closest equivalent pixel in input frame
                        i_in=round((i/factor))+x_offset;
                        j_in=round((j/factor))+y_offset;
                        equiv_input_offset=(i_in)*640+(j_in);
                        equiv_0_0=equiv_input_offset-641;
                        equiv_0_1=equiv_input_offset-640;
                        equiv_0_2=equiv_input_offset-639;
                        equiv_1_0=equiv_input_offset-1;
                        //note that 1_1 is equiv_input_offset
                        equiv_1_2=equiv_input_offset+1;
                        equiv_2_0=equiv_input_offset+639;
                        equiv_2_1=equiv_input_offset+640;
                        equiv_2_2=equiv_input_offset+641;

      *(output+out_pointer_offset)=average[0][0] * *(input+equiv_0_0)+
                              average[0][1] * *(input+equiv_0_1)+
                              average[0][2] * *(input+equiv_0_2)+
                              average[1][0] * *(input+equiv_1_0)+
                        average[1][1] * *(input+equiv_input_offset)+
                              average[1][2] * *(input+equiv_1_2)+
                              average[2][0] * *(input+equiv_2_0)+
                              average[2][1] * *(input+equiv_2_1)+
                              average[2][2] * *(input+equiv_2_2);

                  }
            }
```

```
      }

      printf("completed\n");
}
```

```
//Threshold using white background assumption technique.
//Calculates the threshold value based on the method described in
//section 3.4.1.1

#include <stdio.h>
#define _TI_ENHANCED_MATH_H 1
#include <math.h>

#define ROWS 480
#define COLS 640
#define IMAGE_IN 0x800A9000
#define IMAGE_OUT  0x80240000

#define TEMP 0x800F5000
#define HISTO 0x80140002


int histo[256]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
histo_max,
histo_max_location;

//unsigned char *out = (unsigned char *)IMAGE_OUT;
//unsigned char *inp = (unsigned char *)IMAGE_IN;
float s_factor;


void logcontrast(void);
void histogram(void);
int assumption_threshold(void);
int last_sig_histo();
int sig_factor(void);
void threshold(int);
unsigned char red[]={0xFF,0x00,0x00,0xFF,0x00,0xFF},
blue[]={0x00,0xFF,0x00,0x00,0x00,0xFF},
green[]={0x00,0x00,0xFF,0xFF,0x00,0xFF};

unsigned int foreground;
unsigned int background;


void main()
```

```
{

      int th_value;
      s_factor = 0;
      histogram();
      s_factor = sig_factor();
      foreground = 1; //these parameters selected the foreground
      background = 3; //and background colours
      th_value = assumption_threshold();
      printf("\nThresh value = %d", th_value);
      threshold(th_value);
      printf("\ncompleted\n");

}

/*********************************************************************
* this function finds the distribution of the pixel values between
* 0 and 255
*********************************************************************/
void histogram()
{
      unsigned char *tem = (unsigned char *)TEMP;

      int i;
      unsigned char ch;


      for(i=0;i<307200;i++)
      {
            ch = *(tem+i);
            histo[ch]++ ;

      }
}


int assumption_threshold()
{

      unsigned int  thresh_value;
      int i;

      histo_max=0;
      for (i=0;i<256;i++)
      {
            if (histo[i]>histo_max)
            {
                  histo_max=histo[i];
                  histo_max_location=i;
            }

            if (histo_max_location<250)
            {
                  thresh_value=histo_max_location-(255-
histo_max_location);
            }
            else
            {
                  thresh_value=histo_max_location-(255-
histo_max_location+5);
            }


      }
```

```
      return thresh_value;

}

void   threshold(int   thresh_value)   /*this   function   generates   the
thresholded image*/
{
    int i;
      unsigned char *tem = (unsigned char *)IMAGE_IN;
      unsigned char *output = (unsigned char *)IMAGE_OUT;


      for(i=0;i<302700;i++)
      {
            if( *(tem+i)<thresh_value)
            {
                  *(output+(i)*3) = red[foreground];
                  *(output+(i)*3+1) = green[foreground];
                  *(output+(i)*3+2) = blue[foreground];
            }
            else
            {
                  *(output+(i*3))  = red[background];
                  *(output+i*3+1) = green[background];
                  *(output+i*3+2) = blue[background];
            }
      }
}

int last_sig_histo()
{
      int flag;
      int i;
      unsigned int p_end;

      /* the following finds the last significant pixel of the histo
      array    */

      flag = 0;
      i=255;
      do
      {
            if((histo[i]>s_factor) & (histo[i-1]>s_factor))
            {
                  flag  = 1;
                  p_end = i;
            }
            i--;
      }
      while(i>=0 && flag==0);

      return p_end;
}

int sig_factor()
{
      float factor;
      unsigned int max;
      int i;

      /* the following loop finds the maximum value of the distribution
      array histo: see histogram for the values in histo*/
      max = 0;
      for(i=0;i<256;i++)
```

```
	{
		if(histo[i]>max)
			max = histo[i];
	}
	/* a pixel value is considered significant if it occurs more than
	0.05 times the occurance of the most occuring pixel    */
	factor = round(0.05*max);//rounding the value

	return factor;
}
```

```
//This function transfers frames from the M3188A using the SRAM mode
//with gated pixel clock interface and thresholds them using the fuzzy
//logic technique. Appropriate hardware configuration is listed
//previously

#define CHIP_6711

#define _TI_ENHANCED_MATH_H 1
#include <math.h>
#include <std.h>
#include <stdio.h>
#include <log.h>
#include <hwi.h>
#include <csl.h>
#include <csl_irq.h>
#include <csl_mcbsp.h>
#include <csl_edma.h>
#include "pingpongcfg.h"

//first define memory locations
#define FRAME_ONE_A 0x800A9000 //These are probably alright.
#define FRAME_ONE_B 0x800F5000 //These are probably alright.

#define FRAME_TWO_A 0x80141000 //Have left 0x1000 between
#define FRAME_TWO_B 0x8018D000 //each data segment

#define PROCESSED_FRAME_START 0x801D9000 //start of actual frame
#define FINAL_FRAME     0x80225000  // start of processed frame

//list of globals needed
int EDMA_finished,
new_frame,
count,
EDMA_number,
CPU_number,
CPU_finished;

unsigned int histo[256];
float s_factor;
int swap_times;  //needed if we only want it to swap a certain number of
times

unsigned char *processed_start =(unsigned char *)PROCESSED_FRAME_START,
              *PCR0            = (unsigned char *)0x018C0024;
unsigned int  *PCR1            = (unsigned int *)0x01900024;

unsigned char *EDMA_frame_a, //pointer to data segment A for EDMA
              *EDMA_frame_b; //pointer to data segment B for EDMA

int enable;

/*function declarations*/
```

```
void initialise(void); //setup variables/interrupts. run once at start
void EDMA_frame_finished(void);
void CPU_process(void);//this will change, need to separate out
void zmf(int x[],float y[],int start, int end);
void threshold(int);

void main(void)
{
      initialise();
      for (count=0;count<0x4d000;count++)
      {
            *((unsigned char *)FRAME_ONE_A+count)=0x00;
            *((unsigned char *)FRAME_ONE_B+count)=0x00;
            *((unsigned char *)FRAME_TWO_A+count)=0x00;
            *((unsigned char *)FRAME_TWO_B+count)=0x00;
            *((unsigned char *)PROCESSED_FRAME_START+count) = 0x00;

      }
      count=0;

      while(EDMA_finished<3)
      {
            if (new_frame==0)
            {
                  EDMA_enableChannel(hEdmaExtint4);
                  EDMA_enableChannel(hEdmaExtint5);
                  new_frame=1;
                  *PCR1 = 0x3010;
                  *PCR1 = (*PCR1 | 0x3030);
            }
            CPU_process();
      }
      EDMA_close(hEdmaExtint4);
      EDMA_close(hEdmaExtint5);
      MCBSP_close(hMcbsp1);
      MCBSP_close(hMcbsp0);

      printf("finished and all over \n");
}

void initialise(void)
{

      EDMA_finished=0;
      new_frame=0;
      EDMA_number = 1;
      CPU_number = 0;
      CPU_finished = 1;

      IRQ_resetAll();    //clear all interrupts
      EDMA_intClear(1);
      EDMA_intEnable(1); //enable TCINT 1


      IRQ_enable(IRQ_EVT_EDMAINT);
      IRQ_globalEnable();
}



void EDMA_frame_finished(void)
{
      //this function is called by the edma when it is finished a frame
```

```
	EDMA_disableChannel(hEdmaExtint4);
	EDMA_disableChannel(hEdmaExtint5);
	printf("frame finished %d\n",EDMA_finished);
	EDMA_intClear(1);    //clear the cipr register so this function
will
                                    //trigger again
	if(CPU_finished)
	{
		if(EDMA_number == 1)
		{
			EDMA_number = 2;
			CPU_number = 1;
			edmaCfg0.dst = FRAME_TWO_A;
			edmaCfg1.dst = FRAME_TWO_B;
		}
		else
		{
			EDMA_number = 1;
			CPU_number = 2;
			edmaCfg0.dst = FRAME_ONE_A;
			edmaCfg1.dst = FRAME_ONE_B;
		}
		CPU_finished = 0;
	}
	EDMA_config(hEdmaExtint4,&edmaCfg0);
	EDMA_config(hEdmaExtint5,&edmaCfg1);


	EDMA_finished++;
	new_frame = 0;
}

void CPU_process(void)
{
	long int read_counter;
	int i;
	unsigned char *CPU_frame_a,
		*CPU_frame_b;
	if(CPU_number != 0)
	{
		if(CPU_number==1)
		{
			CPU_frame_a = (unsigned char *)FRAME_ONE_A;
			CPU_frame_b = (unsigned char *)FRAME_ONE_B;
		}
		else
		{
			CPU_frame_a = (unsigned char *)FRAME_TWO_A;
			CPU_frame_b = (unsigned char *)FRAME_TWO_B;
		}
		//this for loop reads both half pixels at once
		//combines them into a single pixel and then
		//writes the output to processed_start
		for(i=0;i<256;i++)
			histo[i] = 0;
		for(read_counter=0;read_counter<307200;read_counter++)
		{
			//entire process done in one line to save time
			*(processed_start+read_counter)                    =
((*(CPU_frame_b+read_counter) & 0x0F)<<4)|(*(CPU_frame_a+read_counter) &
0x0F);
			// histogram
		}
		for(read_counter=0;read_counter<307200;read_counter++)
```

```
            {
                  if(*(processed_start+read_counter)+1>256)
                        printf("Greater than 256\n");
                  if(*(processed_start+read_counter)+1<0)
                        printf("less than 0\n");

                  histo[*(processed_start+read_counter)+1]            =
histo[*(processed_start+read_counter)+1]+1;

            }
            s_factor = 0;
            s_factor = sig_factor();
            threshold(fuzzy_threshold());
      }
      CPU_finished = 1;

}

int fuzzy_threshold()
{
      float y[256];
      int x[256];
      unsigned int p_start, p_end, p_range;
      unsigned int  thresh_value;
      int i;
      unsigned int sum;
      unsigned int histo0;
      unsigned char *tem = (unsigned char *)PROCESSED_FRAME_START;

      p_start = first_sig_histo(s_factor);
      p_end   = last_sig_histo(s_factor);

      p_range = p_end - p_start;

      for(i=0;i<256;i++)
      {
            x[i]=i;
            y[i] = 0;
      }


      zmf(x,y,round(p_start+0.1*p_range), round(p_start+0.4*p_range));

      sum = 0;

      for(i=0;i<307200;i++)
            sum = sum + (y[(*(tem+i)+1)]);


      thresh_value = p_start;

      histo0 = histo[0];

      while(histo0<sum)
      {
            thresh_value = thresh_value + 1;
            histo0 = histo0 + histo[thresh_value];
      }

      return thresh_value;
}

void threshold(int thresh_value)
{
```

```
    int i;
      unsigned char *tem = (unsigned char *)PROCESSED_FRAME_START;
      unsigned char *out = (unsigned char *)FINAL_FRAME;

      for(i=0;i<307200;i++)
         {
               if( *(tem+i)<thresh_value)
                     *(out+i) = 0;
               else
                     *(out+i) = 255;

         }
}
int last_sig_histo()
{
      int flag;
      int i;
      unsigned int p_end;

// the following finds the last significant pixel of the histo array


      flag = 0;
      i=255;
      do
      {
            if((histo[i]>s_factor) )
            {
                  flag  = 1;
                  p_end = i;
            }
            i--;
      }
      while(i>=0 && flag==0);

      return p_end;
}

int first_sig_histo()
{
      int flag;
      int i;
      unsigned int p_start;

//the following finds the last significant pixel of the histo array

      flag = 0;
      i=0;
      do
      {
            if((histo[i]>s_factor))
            {
                  flag  = 1;
                  p_start = i;
            }
            i++;
      }
      while(i<256 && flag==0);

      return p_start;
}


int sig_factor()
```

```
{
      float factor;
      unsigned int max,max2;
      int i;

/* the following loop finds the maximum value of the distribution array
   histo: see histogram for the values in histo */
      max = 0;
      for(i=0;i<256;i++)
            if(histo[i]>max)
                  max = histo[i];
            max2 = 0;
            for(i=0;i<256;i++)
                  if(histo[i]>max2 & histo[i]!=max)
                        max2 = histo[i];

/* a pixel value is significant if it occurs more than 0.05 times the
   occurance of the most occuring pixel                */

                  factor = (0.1*max2+0.5);//rounding the value

                  return factor;
}

void zmf(int x[],float y[],int start, int end)
//this function generates the fuzzy transfer function
{
      int i;
      int range;
      float mid;

      range = end - start;
      mid = (end+start)/2;

      for(i=0;i<256;i++)
      {
            if(x[i]<start)
                  y[i] = 1;
            else if(start<=x[i] && x[i]<=mid)
            {
                  y[i] =  (((float)x[i]-start)/range);/*^2;*/
                  y[i] = 1-2*y[i] * y[i];
            }
            else if(mid<x[i] && x[i]<=end)
            {
                  y[i] = (((float)end-x[i])/range); /*^2;*/
                  y[i] = 2*y[i] * y[i];
            }
            else
                  y[i]=0;

            //printf("\ny %d = %f",i,y[i]);

      }

}
```

```
//This code adds focus lines to a picture.

#include <stdio.h>
#define _TI_ENHANCED_MATH_H 1
#include <math.h>
#define input_location  0x80000000
```

```
#define output_location 0x8004B018


void main()
{
      int const rows=480;
      int const cols=640;
      int i,line1,line2,centre,width;
      unsigned char * input = (unsigned char *) input_location;
    unsigned char * output = (unsigned char *) output_location;

      centre=302;
      width=60;
      line1=centre-round(0.5*width);
      line2=centre+round(0.5*width);

      for (i=0;i<rows*cols;i++)
      {
            if ((i<(line1-1)*640)|(i>=(line2+1)*640))
            //if outside guidelines
            {
                  if (*(input+i)<0x40)
                        *(output+i)=0x00;
                  //cant get any darker than black...
                  else
                        *(output+i)=*(input+i)-0x40;//make image darker

            }
            else if ((i>(line1+2)*640)&(i<(line2-2)*640))
            //if inside guidelines
            {
                  *(output+i)=*(input+i);
            }
            else  //otherwise, there should be a line there
            {
                  *(output+i)=0x00; //a black line
            }
      }

      printf("completed\n");

}
```

```
//This function performs applies the Edge Outline
//filter to a frame.

#include <stdio.h>
#define _TI_ENHANCED_MATH_H 1
#include <math.h>

#define input_location  0x80000000
#define output_location 0x8004B018
#define edge_location   0x80096030


void main()
{
      double edge_strength=1; //Strength of edge outline

      //sobel operators
      double gx[3][3]={{1,  0 , -1},{2,  0,  -2},{1,  0,  -1}};
      double gy[3][3]={{-1, -2 , -1},{0,  0,  0},{1,  2,  1}};
```

```
      unsigned char * line_ref;
      long int i,j,row_pos;
      double horiz_edge,vert_edge;
      float pixel_value,max_edge=0.0;
      unsigned char * input = (unsigned char *) input_location;
      unsigned char * output = (unsigned char *) output_location;
      unsigned char * edge = (unsigned char *) edge_location;



for (i=0;i<479;i++)
{
      //read position line_ref over the rows we need
      row_pos=i*640;
      line_ref = (unsigned char *)(input+(row_pos));
      for (j=1;j<639;j++)
      {
            //now perform edge detection
            horiz_edge=edge_strength*gx[0][0]*(int)*(line_ref+(j-1))+
            edge_strength*gx[0][1]*(int)*(line_ref+j)+
            edge_strength*gx[0][2]*(int)*(line_ref+(j+1))+
            edge_strength*gx[1][0]*(int)*(line_ref+640+(j-1))+
            edge_strength*gx[1][1]*(int)*(line_ref+640+j)+
            edge_strength*gx[1][2]*(int)*(line_ref+640+(j+1))+
            edge_strength*gx[2][0]*(int)*(line_ref+1280+(j-1))+
            edge_strength*gx[2][1]*(int)*(line_ref+1280+j)+
            edge_strength*gx[2][2]*(int)*(line_ref+1280+(j+1));

            vert_edge=edge_strength*gy[0][0]*(int)*(line_ref+(j-1))+
            edge_strength*gy[0][1]*(int)*(line_ref+j)+
            edge_strength*gy[0][2]*(int)*(line_ref+(j+1))+
            edge_strength*gy[1][0]*(int)*(line_ref+640+(j-1))+
            edge_strength*gy[1][1]*(int)*(line_ref+640+j)+
            edge_strength*gy[1][2]*(int)*(line_ref+640+(j+1))+
            edge_strength*gy[2][0]*(int)*(line_ref+1280+(j-1))+
            edge_strength*gy[2][1]*(int)*(line_ref+1280+j)+
            edge_strength*gy[2][2]*(int)*(line_ref+1280+(j+1));


*(edge+(i*640)+j)=(int)sqrt((vert_edge*vert_edge+horiz_edge*horiz_edge))
;
            if (*(edge+(i*640)+j)>max_edge)
                  max_edge=*(edge+(i*640)+j);
        }
    }

      //now combine the edge detection output with the original frame
      for (i=0;i<640*480;i++){
      pixel_value=round((*(input+i) *(*(edge+i)/max_edge)))+*(input+i);
            if (pixel_value>255)
                  *(output+i)=0xFF;
            else
                  *(output+i)=pixel_value;
      }

      printf("completed\n");

}
```

```
//This function performs applies the Edge Enhance
//filter to a frame.
```

```
//Note: For the C port of this code, the variable cutoff frequency
//has not yet been implemented.
//The cutoff frequency is set to 0.2

#include <stdio.h>
#define _TI_ENHANCED_MATH_H 1
#include <math.h>

#define input_location  0x80000000
#define output_location 0x8004B018
#define low_location   0x80096030
#define high_location  0x800E1094


void main()
{
      double f_amp=100; //Foreground amplification
             b_amp=0.2; //Background amplification

   double h={{0.0163,0.0325,0.0163},{0.0325,0.8047,0.0325},
                        {0.0163,0.0325,0.0163}};
      unsigned char * line_ref;
      long int i,j,row_pos;
      float pixel_value,max_pixel=0.0;
      unsigned char * input = (unsigned char *) input_location;
   unsigned char * output = (unsigned char *) output_location;
      unsigned char * low = (unsigned char *) low_location;
   unsigned char * high = (usigned char *) high_location;

max_pixel=0;


for (i=0;i<479;i++)
{
      //read position line_ref over the rows we need
      row_pos=i*640;
      line_ref = (unsigned char *)(input+(row_pos));
      for (j=1;j<639;j++)
      {
            //now apply low pass filter
            low=h[0][0]*(int)*(line_ref+(j-1))+
                h[0][1]*(int)*(line_ref+j)+
                h[0][2]*(int)*(line_ref+(j+1))+
                h[1][0]*(int)*(line_ref+640+(j-1))+
                h[1][1]*(int)*(line_ref+640+j)+
                h[1][2]*(int)*(line_ref+640+(j+1))+
                h[2][0]*(int)*(line_ref+1280+(j-1))+
                h[2][1]*(int)*(line_ref+1280+j)+
                h[2][2]*(int)*(line_ref+1280+(j+1));
    }

        }
    }

      //now obtain high frequency components
      for (i=0;i<640*480;i++)
   {
            *(high+i)=round(*(input+i)- *(low+i));
            if (*(high+i)>255)
                  *(output+i)=0xFF;
   }

   //combine image components
      for (i=0;i<640*480;i++)
```

```
   {
          *(output+i)=round((f_amp * *(high+i)) +(b_amp * *(low+i)));
      if (*(output+i)>max_pixel)
          max_pixel=*(output+i);
   }

   //and amplify
   for (i=0;i<640*480;i++)
   {
          *(output+i)=round((*(output+i)/max_pixel)*255);
   }

      printf("completed\n");
}
```

Appendix B – MATLAB Source Code

```
function output=ti_data_read(input)

%This function reads in a memory dump from
%Code Composer studio and attempts to convert it
%to an image file for use in the MATLAB environment.

output=zeros(480,640);
fid = fopen(input,'rt');
tline= fgetl(fid);  %ignore header line

for row=1:1:480
  for column=1:4:640
    tline= fgetl(fid);
    output(row,column+3)=hex2dec(tline(3:4));
    output(row,column+2)=hex2dec(tline(5:6));
    output(row,column+1)=hex2dec(tline(7:8));
    output(row,column)=hex2dec(tline(9:10));
  end;
end;
output=uint8(output);
```

```
function output=ti_data_write(input,file)

%This function wites a MATLAB image matrix
%to a file that can be directly read to memory by
%Code Composer

input=double(input);
fid = fopen(file,'wt');
fprintf(fid,'%d %d %d %d %d\n',1651,1,0,1,0);

for j=1:1:480
    for i=1:4:640

fprintf(fid,'0x%02x%02x%02x%02x\n',input(j,i+3),input(j,i+2),input(j,i+1
),input(j,i));
    end
end

status = fclose(fid)
output=1;
```

```
function output=edge_enhance(input,cutoff,f_amp,b_amp)

%This function applies the edge enhance filter using FIR filtering.
%---------------------------------------------------------------------
%Output - Output image
%Input - Input image
%cutoff - Normalised cutoff frequency
%f_amp - Foreground amplification
%b_amp - Background amplification

input=double(input);
b = fir1(2,cutoff); %Generate 1D FIR filter with N=3
h = ftrans2(b);     %and convert it to 2D
```

```
low=filter2(h,input); %Filter to obtain low freq components


high=input-low;  %Sepearate high freq components
high_amp=high*f_amp;  %Amplify
low_amp=low*b_amp;    %Amplify
output=high_amp+low_amp;    %Sum and output
output=uint8(output);
```

```
function [output]=gauss_smth(input)

%This function applies the gaussian smoothing filter use
%to simulate blurred vision through this report
%-------------------------------------------------------

input=double(input);

[cols,rows]=size(input);
output=zeros([cols,rows]);

%This is the gaussian convolution mask
gauss=(1/159)*[2 4 5 4 2;4 9 12 9 4;5 12 15 12 5;4 9 12 9 4;2 4 5 4 2];

output=uint8(conv2(input,gauss));
```

```
function [output]=logcontrast(input)

%This function applies the log contrast operator used by the
%fuzzy threshold method

[rows,cols]=size(input);
output=zeros([rows,cols]);

%this line is only needed in matlab
input=double(input);

histo=histogram(input);

%a pixel value is considered significant if it occurs
%more than 0.05 times the occuranceof the most occuring pixel
sig_factor=round(0.05*max(histo));

flag=0;
p_end=0;
for i=length(histo):-1:1 %for each pixel value
    if (histo(i)>sig_factor)&&(flag==0) %check if its significant
        flag=1;                         %if yes, define it as the end
        p_end=i
    end;
end;

c=255/(log(1+p_end));
for i=1:1:rows
    for j=1:1:cols
        output(i,j)=c*log(1+abs(input(i,j))); %apply log contrast

    end;
end;

%this line is only needed in matlab
output=uint8(output);
```

```
function [histo]=histogram(input)

%This function generates the histogram of an image
%-------------------------------------------------

%needed for matlab
input=double(input);
[rows,cols]=size(input);
histo=zeros(1,256);
for i=1:1:rows
    for j=1:1:cols
        histo(input(i,j)+1)=histo(input(i,j)+1)+1;
    end;
end;
```

```
function [output,l_list]=fuzzy_threshold(input)

%This function finds the fuzzy threshold value of
%an image and applies the threshold function
%at this value.
%-----------------------------------------------

input=double(input);
histo=histogram(input);
[rows,cols]=size(input);
output=zeros([rows,cols]);

sig_factor=round(0.05*max(histo));
%a pixel value is significant if it occurs more than 0.05 times the
%occurance of the most occuring pixel

flag=0;
p_start=0;
for i=1:1:length(histo) %for each pixel value
    if (histo(i)>sig_factor)&&(flag==0) %check if its significant
        flag=1;                         %if yes, define it as the start
        p_start=i;
    end;
end;

flag=0;
p_end=0;
for i=length(histo):-1:1 %for each pixel value
    if (histo(i)>sig_factor)&&(flag==0) %check if its significant
        flag=1;                         %if yes, define it as the end
        p_end=i;
    end;
end;

p_range=p_end-p_start;
x=0:1:255;
%generate fuzzy seet transfer function
y=zmf(x,[round(p_start+0.1*p_range),round(p_start+0.4*p_range)]);
sum=0;



for i=1:1:rows
    for j=1:1:cols
        %calculate the number of dark pixels
        sum=sum+y(input(i,j)+1);
```

```
    end
end


thresh_value=p_start;
i=histo(1);
while (i<sum)
    thresh_value=thresh_value+1;
    i=i+histo(thresh_value);
end;
%generate thresholded output
output=uint8(threshold(input,thresh_value));
```

```
function output=threshold(input,level)

%This function applies a simple threshold to an image
%on being given a specific threshold value
%-----------------------------------------------------

input=double(input);
[rows,cols]=size(input);
output=zeros([rows,cols]);


for i=1:1:rows
    for j=1:1:cols

        if (input(i,j)<level))
                output(i,j)=0;

            else
                output(i,j)=255;
        end
    end;
end;
output=uint8(output);
```

```
function [post]=zoom(input,factor)

%This zoom function uses gaussian interpolation to achieve
%zoom factors of up to 4.
%-----------------------------------------------------------
input=double(input);
[rows,cols]=size(input);
pre=zeros([rows,cols]);
post=zeros([rows,cols]);

%Establish portion of image to be zoomed
row_zoomed=(rows-(rows/factor))/2;
col_zoomed=(cols-(cols/factor))/2;

%Gaussian filter
filter=(1/16)*[1 2 3 4 3 2 1;2 4 6 8 6 4 2;3 6 9 12 9 6 3;
          4 8 12 16 12 8 4;3 6 9 12 9 6 3;2 4 6 8 6 4 2;1 2 3 4 3 2 1];

for i=row_zoomed:1:rows-row_zoomed-1
    for j=col_zoomed:1:cols-col_zoomed-1
        pre(((i-row_zoomed)*factor)+1,((j-
col_zoomed)*factor)+1)=input(i,j);
    end;
end;
```

```
    post=conv2(pre,filter);
    post=post(1:rows,1:cols);
post=uint8(post);
```

```
function [output] = sobel_conv(input)

%This function implements a Sobel edge detector.
%It is used by the edge outline filter
%-----------------------------------------------

input=double(input);
gx=-0.25.*[-1 0 1;-2 0 2;-1 0 1];
gy=-0.25.*[1 2 1;0 0 0;-1 -2 -1];

output_rowgrad=conv2(input,gx);
 output_colgrad=conv2(input,gy);

 [cols,rows]=size(output_rowgrad);
 output=zeros([cols,rows]);
 for j=1:1:rows
     for i=1:1:cols

output(i,j)=sqrt((output_rowgrad(i,j)^2)+(output_colgrad(i,j)^2));
    end
end
output=output(1:480,1:640);
output(1:7,1:640)=0;
output(473:480,1:640)=0;   %get rid of white edges introduced by conv2
function
output(1:480,1:7)=0;
output(1:480,633:640)=0;
output=uint8(output);
```

```
function output=edge_outline(input,strength)

%This function implements the edge outline filter
%-----------------------------------------------

% This line is neceesary to process images in
% the uint8 8 (ie 0-255) format
input=double(input);
output=zeros(size(input));

edge=double(sobel_conv(input));
max_edge=max(max(edge));
edge=strength*(edge./max_edge);

for row=1:1:480
  for column=1:1:640

output(row,column)=input(row,column)+(input(row,column)*edge(row,column)
);
  end;
end;

output=uint8(output);
```

```
function [output]=focus(input,pos,width)

%This function applies the focus filter
%Where the focus area's centre and width can be specified
```

```
[rows,cols]=size(input);
output=zeros([rows,cols]);
input=double(input);
ymin=pos-0.5*width;
ymax=pos+0.5*width;

for i=1:1:rows
    if (i<ymin)||(i>ymax)
        output(i,1:1:cols)=(input(i,1:1:cols)-60);
    else if (i==ymin)||(i==ymax)
        output(i,1:1:cols)=0;
    else
        output(i,1:1:cols)=input(i,1:1:cols);
    end;
end;
output=uint8(output);
end;
```