HARDWARE IMPLEMENTATION OF FOG/HAZE REMOVAL SYSTEM

By

NC Saleha Shafique
NC Afsheen Ahmed
CSS Ammal Abbas

A thesis submitted to the Department of Electrical Engineering, Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfillment for the requirements of a B.E Degree in Telecommunication Engineering

JUNE 2013

ABSTRACT

HARDWARE IMPLEMENTATION OF FOG/HAZE REMOVAL SYSTEM

The problem that we face when doing long distance photography or making of video is that the light reflected from a surface is scattered in the atmosphere before it reaches the camera and this is due to the presence of aerosols such as dust, mist and fumes which deflect light from its original course of propagation and this effect increases tremendously in the presence of fog or haze. Because of this, the image or video captured appears blur, faint and it lacks visual vividness. So the main aim of this project is to remove the effects of noise that is caused due to the presence of fog/haze from a real time video. The algorithm for the removal of effects of fog/haze is implemented on FPGA. The system first captures the corrupted video with fog/haze with the help of a camera, and then passes this video from the implemented algorithm which removes the effects of fog/haze from the video and in the end displays a clear video without the effects of fog/haze on the VGA monitor.

DISSERTATION

DEDICATED TO MILITARY COLLEGE OF SIGNALS

ACKNOWLEGMENTS

We are grateful to Dr Adil Masood for providing us an opportunity to perform our BE final year project in his Image processing group. As supervisor, his support and guidance has always been a valuable asset for our project. His keen interest and discussions over the work always provided a ray of hope in difficult times. We are really thankful for his guidance in our project work. In the end we would like to thank Rector National University of Sciences & Technology for funding this project. We are also grateful to the whole staff of the Department of Electrical Engineering especially to Head of the department for their support in many forms and enthusiasm.

TABLE OF CONTENTS

TABLE OF FIGURES

ABBREVIATIONS

FPGA	Field Programmable Gate Array
ISE	Integrated Software Environment
USB	Universal Serial Bus
UCF	User Constraint File
PC	Personal Computer
VGA	Video Graphics Array
I/O	Input/Output
GP I/O	General Purpose Input/Output
CLAHE	Contrast Limited Adaptive Histogram Equalization

CHAPTER 1

INTRODUCTION

This thesis proposes the Fog/Haze removal system. The design that has been explained incorporates a complete system that takes a video from the camera and then applies the algorithm for removal of effects of fog/haze from the video and then after removing the effects of fog/haze, displays a clear video on VGA (monitor).

1.1 Background

Nowadays there is an increase trend of video making and photography. Some people use video making or photography for some kind of special purposes like security, making advertisements, military purposes etc., whereas the other people do video making or photography as their hobby.

Photography and video making is done in almost every season and probably everyday. People used to make videos and take pictures on sunny, cloudy, rainy, foggy days and nights and also in indoor and outdoor environments.

Although video making and photography is done in different kinds of environments but people always want to get the best results. They always want to see the best quality picture and this best quality picture includes enhanced colors, no blurriness, clearness, sharpness and also this picture should not contain any kind of noise.

Systems and programs are developed to edit the pictures and videos to provide the best quality pictures to viewers without any kind of noise but no system is yet developed that could remove the effects of fog or haze from the pictures or videos. Different papers have proposed different kinds of algorithms but these algorithms are not implemented physically. And also there is a need of removing the effects of fog/haze from the real time videos to avoid so many problems like accidents in foggy weather. So there is a need to implement physically the proposed algorithms for real time videos too.

1.2 Problem Statement

The problem we face when doing long distance photography or making of video is that videos coming from visible spectrum contain noise. This noise increases tremendously in the presence of fog/haze. In almost every practical scenario the light reflected from a surface is scattered in the atmosphere before it is received by the camera. This is because of the presence of aerosols such as dust, mist and fumes which deflect light from its original course of propagation. In long distance photography of foggy scenes this process has a substantial effect on the image in which contrasts are reduced and surface colors become faint. Such videos/images often lack visual vividness and appeal; moreover they offer a poor visibility of the scene contents.

1.3 Objective of the project

The project’s objective is to implement an algorithm on a hardware that would remove the effects of fog/haze from real time videos. We are dealing with real time videos in this project. So we really need to have very fast processing rate and to acquire this high processing rate, we have chosen FPGA (Field Programmable Gate Array) as our hardware. The main components of the system are shown in the figure1.1

The detailed design of the system has been shown in this figure. It consists of three parts. First part shows a camera which is used for acquisition of a corrupted video with fog/haze. Second part shows the hardware which is FPGA, in the first step within the hardware video with fog/haze effects is converted to frames, then these frames are passed through the algorithm which will remove these effects and then convert the frames back to the video and in the last part displays the clear video on VGA monitor.

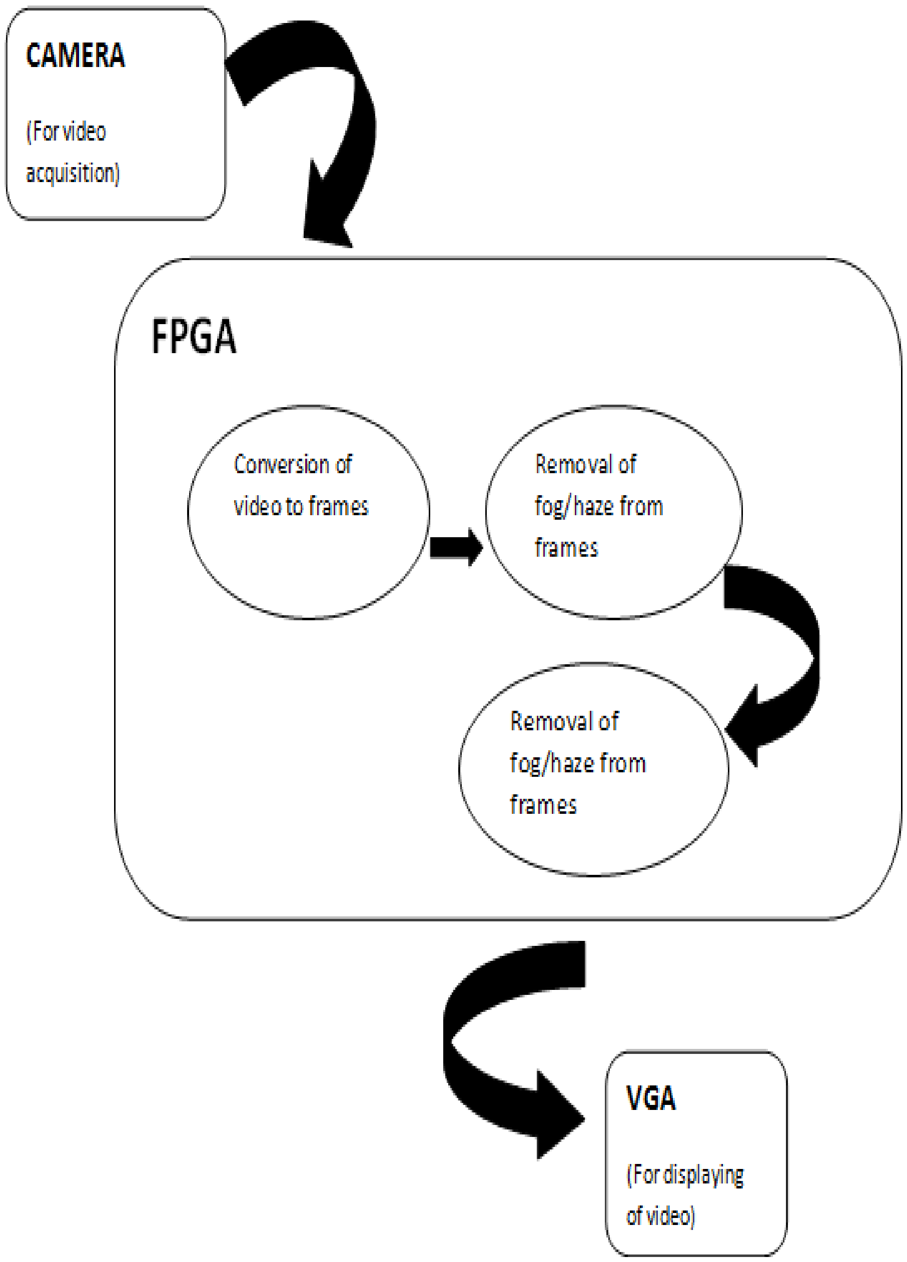

CAMERA
(For video acquisition)
FPGA
Conversion of video to frames
Removal of fog/haze from frames
Removal of fog/haze from frames
VGA
(For displaying of video)

Figure 1.1 System Overview

1.4 Project Contribution

The algorithm for the removal of effects of fog/haze from real time videos has been implemented in a compact and efficient design. The system is able to receive a video captured with the help of a camera as an input and then pass this video that is corrupted with fog/haze from the hardware that will remove the effects of noise because of fog/haze and in the end display the clear video as an output of this system on VGA (monitor).

1.5 Thesis Organization

Our thesis is organized into four chapters. The first chapter introduces the background, motivation, research objectives, and scope of work and contribution of this project. The second chapter describes the theory of techniques that are used to enhance the images and to remove the effects of noise from an image to produce the best results. The third chapter presents the research methodology, system design procedures and application tools and the last chapter is about future enhancements and conclusion.

CHAPTER 2

LITERATURE REVIEW

2.1 Background knowledge

In this chapter the basic information about the project techniques to fully understand the problem is defined.

2.1.1 Weather terminology

The presence of fog/ haze in the atmosphere can reduce visibility. This is dangerous for all kind of traffic, whether it is on road, sea or air. This can lead to serious accidents. Images or videos that are taken in bad or foggy weather have poor contrast and color. To remove the effects of bad weather, the physical processes need to be understood that cause these effects. As light propagates, its key characteristics (intensity, color, enhancement, brightness) are altered due to atmospheric particles.

2.1.2 Particles in space

Weather conditions are affected by the sizes and types of particles involved in effecting the atmospheric photo conditions and their concentration in atmosphere. These particles results in a variety of weather conditions which are briefly discussed below.

2.1.2.1 Haze

It is constituted of aerosols which is a dispersion of small particles suspended in gas form in the atmosphere. Haze has a number of sources which includes volcanic ashes, combustion products, sea salt and foliage exudation. These sources produce particles which respond quickly to changes in relative humidity and act as center of small water droplets whenever humidity is high. Haze particles are larger than air molecules but their size is smaller than fog droplets. Haze produces a grayish or bluish hue and tends to effect visibility.

2.1.2.2 Fog

Fog occurs at the time when the relative humidity of an air constituent reaches to saturation. At this time, some particles convert into water droplets by condensation. Hence fog and many other types of haze have similar way of formation. Moreover increase in humidity is a one of the important and sufficient reason to turn haze into fog.

This transition is slow and gradual and a state comes during this conversion which is referred to as mist. There are different types of fog (for example advection fog, radiation fog etc.) which differ from each other in their formation process. Fog is typically a few hundred feet thick while haze has an altitude of several kilometers. The practical difference between fog and haze is that fog greatly reduces the visibility.

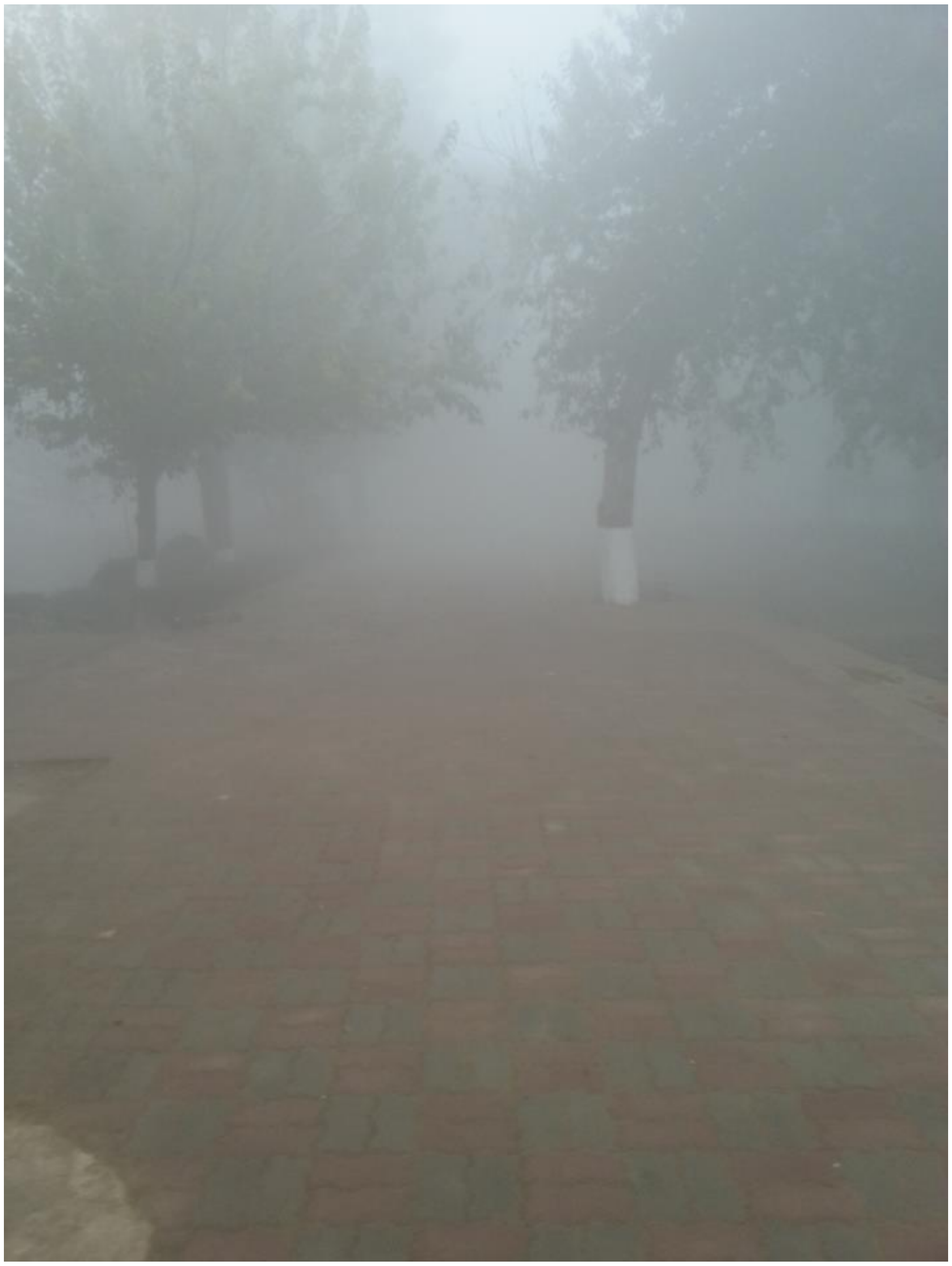


Figure 2.1 foggy weather

2.1.2.4 Cloud

The main difference between fog and clouds is that clouds exist at higher altitudes rather than reside at ground level. Most clouds compose of water droplets like water vapours, fog but some are composed of long ice crystals and ice-coated dust grains. Since we restrict ourselves at lower level vision for photography rather than high altitudes, so clouds are of less relevance at this level.

2.1.2.5 Rain and Snow

The process of converting cloud droplets to rain is complex one. Rain causes random temporal and spatial variations in images, and hence must be deal differently from the static or stable weather conditions like fog, haze, etc. Snow is also handled similarly because its flakes are rough in shape and have more complex optical properties and vary in shape. It is set aside for now.

This project is focused on stable and steady weather conditions such as haze, fog, mist and other aerosols (causing mist). Dynamic weather conditions such as rain, hail and snow as well as turbulence are not handled.

2.1.3 Sky Illumination

The sky is normally overcast under misty or foggy conditions. In such cases we assume that the irradiance is dominated by the radiance of the sky at each scene point, and it is not significant due to other scene points.

2.1.4 Possible solution

A possible solution for this reduced visibility is to have a camera to record the scene and an algorithm to remove the effects of fog or haze in real time videos show the fog free scene on a display screen (LCD, VGA monitor, etc). However, there are no such algorithms that use a real time video as the input , although there are multiple methods implemented that can remove fog from a series of images (specifically a single image).

2.2 Literature Review

Video enhancement problem can be formulated as given an input low quality video and the output is high quality video for specific applications. Our goal is to deal with this situation in which we think how can we make video more clearer or subjectively better. Clear vision has become an integral part of everyday life in the areas having heavy fog. In such areas main aim is to remove the visual appearance of video to make the view clear for travelling and other military uses.

Different techniques are implemented which provide a better transform representation for future automated video processing, such as analysis, segmentation, and recognition of the consecutive images taken in video form. Moreover, it helps analyses background information that is essential to understand the behavior of objects.

The existing techniques of video enhancement can be classified into two broad categories i.e. Self-enhancement and frame-based fusion enhancement. Traditional methods of video enhancement are to enhance the low quality video itself. For foggy videos, these enhancement techniques are applied to video coming from camera and atmospheric veil is extracted from the scene which results in removal of fog from videos. It doesn't embed any high quality background information since scene is changing continuously.

Such techniques are contrast enhancement method, HDR-based video enhancement, wavelet-based transform video enhancement and compressed-based video enhancement. These techniques are called self-enhancement of low-quality video.

Self-enhancement doesn't enough luminous of low quality video, because in the foggy video some areas are so luminant that all the information is already lost in those regions. No matter how much illumination enhancement we apply, it will not be able to bring back lost information.

Frame-based fusion enhancement refers to low quality video, which fuse illumination information in different time video. It is approached by extracting high quality background information to embed low quality video. Various researchers are extracting information to understand the solution of problems that are how can we combine information from different background images in a meaningful way and how can we pick high quality high quality background parts while retaining low quality information.

2.2.1 Contrast Enhancement

All these techniques are used for contrast enhancement of an image. Since video is a sequence of consecutive images, contrast enhancement is done on each image(known as frame). There are two types of contrast enhancement techniques i.e. Model-based and Non-model-based.

2.2.2 Non-model-based

These techniques do contrast enhancement without knowing the cause of image degradation. The main techniques included in non-model-based contrast enhancement are histogram equalization, contrast limited adaptive histogram equalization retinex theory and, unsharp masking based techniques.

2.2.3 Model-based

This technique performs contrast restoration by reversing the underlying cause. It includes multiple image approaches, depth based approaches and single image approaches.

Non model based techniques

2.2.3.1 Histogram equalization

Histogram equalization is one of the most commonly used methods for contrast enhancement. It alters the spatial histogram of an image to closely match a uniform distribution. The main objective of this technique is to achieve a uniform distributed histogram by using the cumulative density function of the input image.

The advantages of the HE include (i) it suffers from the problem of being poorly suited for retaining local detail due to its global treatment of the image. (ii) small-scale details that are often associated with the small bins of the histogram are eliminated.

The disadvantage is that it is not a suitable algorithm for some applications such as consumer electronic products, where it colors changes continuously and brightness preservation is necessary to avoid annoying artifacts. The equalization result is usually an undesired loss of visual data, of quality, and of intensity scale. Fig.2 shows the experimental result of histogram equalization.

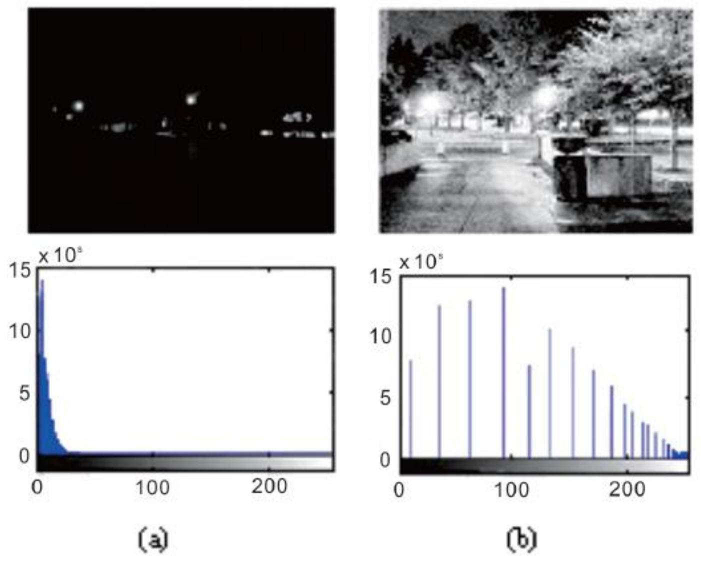


Figure 2.2 Histogram equalization. (a) Original frame of low quality and histogram image (b) the result of histogram equalization and histogram image.

The histogram is divided into three regions as dark, mid and bright. In order to keep original histogram features, the differential information is extracted from input histogram and then desired histogram is specified based on this information and some extra parameters are added such as gain value of the input image.

2.2.2.2 Contrast Limited Adaptive Histogram Equalization

CLAHE forms small regions of the image which are called *tiles*, and operates on them instead of operating on the entire image. The 'Distribution' parameter histogram is specified for complete image, each tile's contrast is enhanced, so that when histogram of the output region is taken it approximately matches the Distribution parameter histogram. The neighboring tiles are then combined using bilinear interpolation to eliminate artificially induced boundaries. In homogeneous areas where there is not so much color distinction, the contrast is limited to avoid amplifying any noise that might be present in the image during taking histogram equalization.

Contrast limited adaptive histogram equalization differs from ordinary adaptive histogram equalization in its contrast limiting. The contrast limiting procedure is applied for each neighborhood from which a transformation function is derived. It helps to prevent the over amplification of noise that occurs in adaptive histogram equalization.

2.2.2.3 Usharp masking

Unsharp masking is an image sharpening technique, normally available for digital image processing. The technique uses a blurred or unsharp positive image to create a mask of the original image. The created unsharp mask is then combined with the negative image thus creates an image that is less blurry than the original image. The resulting image, although clearer, may be a less accurate representation of the image’s subject. It is generally a linear or non-linear filter that amplifies the high frequency components of the coming images.

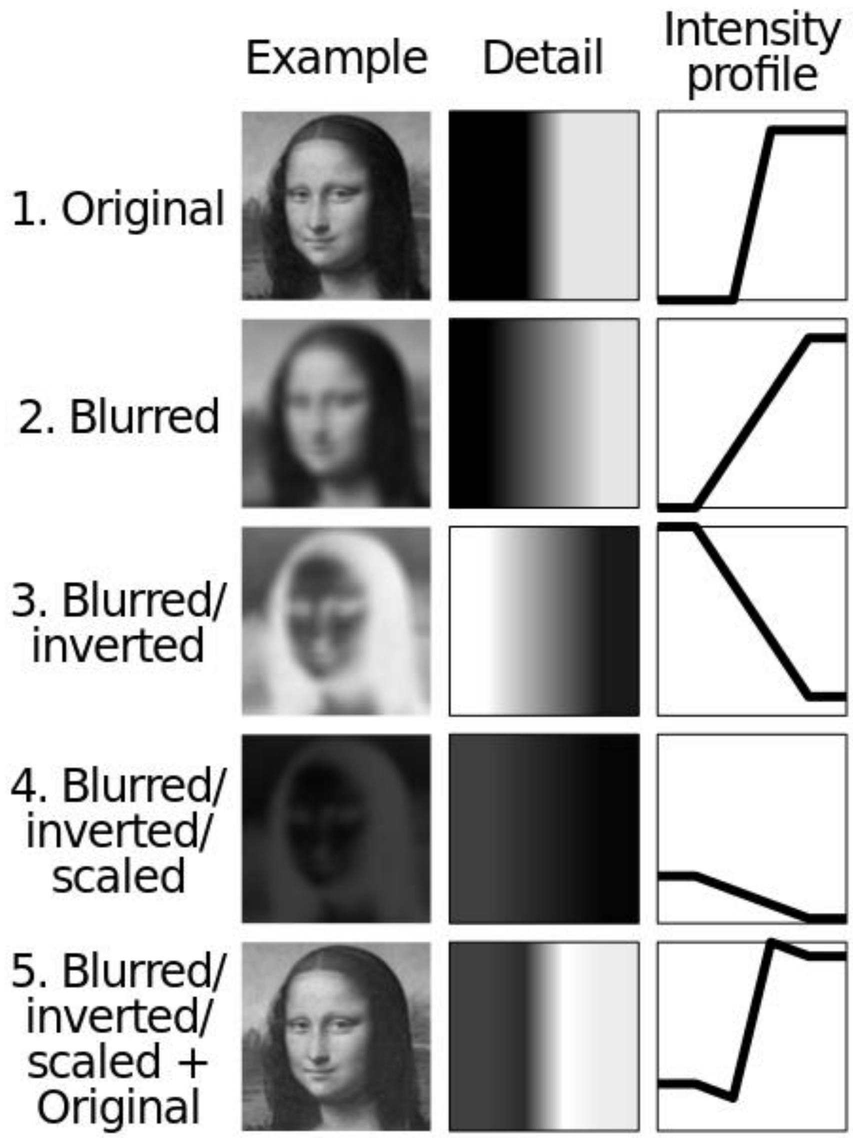


Figure 2.3 simplified principle of unsharp masking

2.2.2.4 Retinex theory

Retinex theory has been known for more than 30 years as a simple and effective model of human vision. The name retinex comes from the contraction of two words “retina” and “cortex” which indicates the intension to take into account the biological elements that influence our visual perception.

The basic concept is to separate the components of an image i.e. illumination and reflectance. The available luminance data in the image is the product between illumination and reflectance. The reflectance component can be obtained by calculating the ratio between luminance and an estimate of illumination. To estimate illumination information, a low-pass filter is used. It includes single Gaussian estimation, multi-scale Gaussian estimation and bilateral filter estimation. Using retinex theory has several advantages. (i)the both images (reflectance image and illumination image) can be obtained from a single image instead of a sequence of images, (ii)retinex method does not require different algorithms to study vast and no training images are needed, and (iii) there is no assumption about lighting sources and shadow.

An input color image is decomposed into intensity image *I(x, y)* and a color layer *C(x, y).* the color layer C=(r, g, b) is given by dividing the input pixel values by the intensity *I(x, y).* color space including different color planes like RGB, HSV, Y cb cr. This intensity image *I(x, y)* is turned into the luminance layer *L(x, y)* and the reflectance layer *R(x, y)* by using retinex theory.

Intensity image *I(x, y)* is represented by the product of the luminance *L(x,y)* and the reflectance image *R(x, y).*

$$\mathbf{I(x, y) = L(x, y)\, R(x, y)} \qquad \textbf{Equation 2.1}$$

Luminance *L(x, y)* is assumed to be low frequency component of an image *I(x, y)* and reflectance image *R(x, y)* is calculated as the ratio of the image *I(x, y)* and the luminance *L(x, y).*

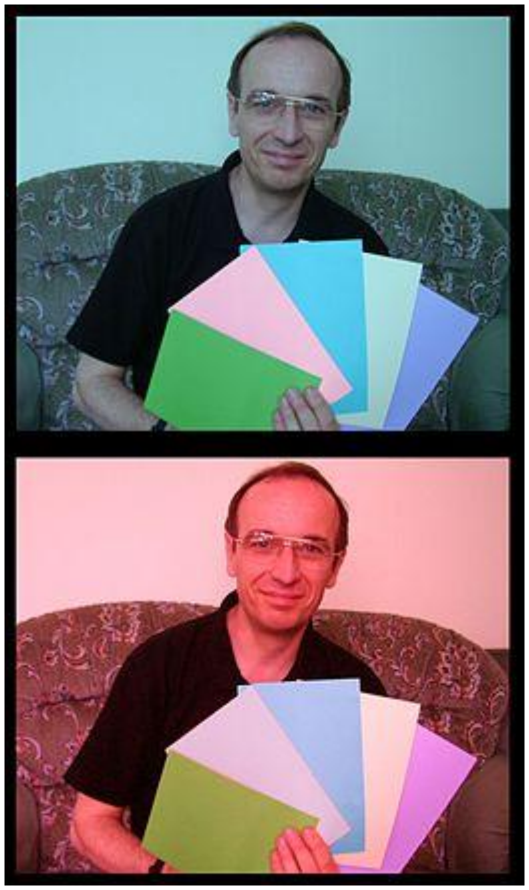


Figure 2.4 Assumption about color changes due to the color cast of surrounding photo

2.3 Single image fog removal

For visibility enhancement in bad weather for video we start by analyzing the two mentioned methods and the fog model. In contrast single image visibility enhancement is more difficult than the video coming in a sequence of consecutive images. This is due to the fact that limited information comes from an image and causes difficulty of estimating depth with this little information. Multiple methods exist that are capable of enhancing the visibility with the single image i.e. the work of Tarel [1].

The choice of methods to analyse and compare depends on how often they are cited and their appearance in comparison with other methods. Another method for choice are the differences between the work done for the two methods that are compared for better results. Tan's [2] work is more capable of improving the visibility under thick fog but can take several minutes to compute. Tarel [1] work has more difficulties while handling images with thicker fog but can restore an image within delay of seconds.

These methods are well suited for creating a good estimation of the atmospheric veil, but they focus only on a single image

2.3.1 Fog model

Fog is a cloud of humidity near the surface of Earth, substance of water droplets. As light travels through this substance a part of light is absorbed and this amount of light which is absorbed depends on the fog substance through which the light travels and the absorption coefficient. According to the Beer-Lambert law there is a logarithmic dependence between the transmission of light and the product of distance and absorption coefficient,

$$\mathbf{I}(x) = \mathbf{L}_{\infty}\rho(x)e^{-\beta d(x)} + \mathbf{L}_{\infty}(1 - e^{-\beta d(x)})$$ **Equation 2.2**

This fog model is the common model used when dealing with removing fog. The model is shown in Equation 2.2. The parameter I(x) is the intensity of image that contains fog. in Beer-Lambert law the both parameters i.e. direct attenuation and the air light are logarithmic dependent. The direct attenuation exists out of air light color, which is the color reflectance of the object in the image.

2.3.2 White Balance

The lightening conditions cause a color cast under which we take a photograph, which depends on the temperature of color of the used light source, i.e. camera capturing video. Low color temperature light sources cause a red/amber color cast. As the color temperature rises to a higher color temperature, like sun, the color cast is bluish. So to balance the white color we take a better source which comes in between the two and it can be removed by performing white balance also known as color balance. By removing color cast original colors of an image are restored. It can improve visibility also. But these improvements are not so significant because the fog factor is still present. It only helps to restore colors in a better way by using a better source. A significant part of white balance is the air light color. To compute the air light color we have to look at two ways.

2.3.3 Tarel et al.'s Implementation

This algorithm is faster and has better results while comparing with other algorithms on fog removal. The computational time is less than a second for a single image, the time required is linear in the amount of pixels**.**

$$\mathbf{I}(x) = R(x)(1 - \frac{V(x)}{L_\infty}) + V(x)$$

Equation 2.3

$$R(x) = \frac{\mathbf{I}(x) - V(x)}{1 - V(x)}$$

Equation 2.4

The fog model Tarel et al. uses is based on previous implementations to make it better. Atmospheric veil V(x) is introduced

$$\hat{V}(x) = L_{\infty}(1 \hat{-} e^{-\beta d(x)}),$$ **Equation 2.5**

The fog model is rewritten in equation (2.3). I(x) defines the intensity image which contains fog and R(x) defines the restored output image. White balance is done in pre-processing stage thus rewritten the equation to equation (2.4) which isolates output image R(x).

2.4 Video based fog removal

The main advantage while dealing with video instead of single images is the extra information of the surroundings that video can provide, but there also exists the first problem that how can we extract information from video that can help improve the results of single image methods? After exploring two different methods of Tan and Tarel to obtain information from video we apply the found information to improve the results of single image removal. We are using extra information, so our result should be better than simply applying a single image fog removal method to each consecutive frame.

Extracting video information is time consuming i.e. the required computational time of our method will be slower than a single image method.

we compare the air light of a single image enhancement method and our improved air light to ground truths manually created with a graphics editing program.

We choose air light for comparison instead of restored images as air light is dependent only on depth. The corrected air light is created by applying information from video and using the intensity information from video of a single image.

So for a single image, the method that we create to enhance visibility in bad weather for video is not fast and is therefore not useful in applications that require fast visibility enhancement, e.g., road sign detection in bad weather for cars. Our method is applicable for applications that process static material (images) from security cameras.

2.5 FPGA (Field Programmable Gate Array)

We have used FPGA as hardware for fast processing of algorithm because delay is not acceptable in our case, as during moving on roads or when moving to have clear path delay may cause accidents.

A **field-programmable gate array** (**FPGA**) is an integrated circuit designed to generally use a hardware description language (HDL) similar to that used for an application-specific integrated circuit (ASIC). FPGAs have large resources of logic gates and RAM blocks to implement complex digital designs using these blocks and logic gates. As FPGA designs gives very fast IOs and bidirectional data buses it becomes an important task to verify correct timing of valid data within setup time and hold time traditionally. FPGAs have been used for specific applications where the volume of production is small and fast processing is required. For such low-volume applications, the companies pay for hardware costs per unit for a programmable chip so it is more affordable than developing resources for creating an ASIC for such application. Today, affordable cost and fast performance dynamics of FPGA have broadened the range of its applications, hence being useful in many companies related projects like in aerospace and defense, audio, automotive and broadcast, electronics, medical, image processing etc.

2.5.1 SPARTAN 3E XC 3S 500e

We have used SPARTAN 3E XC 3S 500E kit. The complete package of this kit is

XC3S500E FGG320-4C.

Here XC specifies that the kit is of Xilinx, 3S gives information about the family. In this kit 3S specifies that the kit is of SPARTAN 3 family. 500E gives the number of K gates used in FPGA kit. FGG is the pin package of the device and 320 are the number of I/Os used in the device. The number can be increased by making use of GP I/Os. Last is – 4 specifies the speed grade of the device, specifies that the kit is 5% faster than the kit having speed -3. C is the commercial temperature grade.

CHAPTER 3

SYSTEM DESIGN

3.1 Project Workflow

This chapter discuss about the method and interfaces that have been used to make this project successful. The method includes the discussion of the project workflow, system design procedure, techniques and tools utilized in this work. This project involved the effort of software hardware integration system design process, which involves interface design and software development. Hence, it calls for software system design, in which the algorithm is designed in the software and embedded in the hardware used.

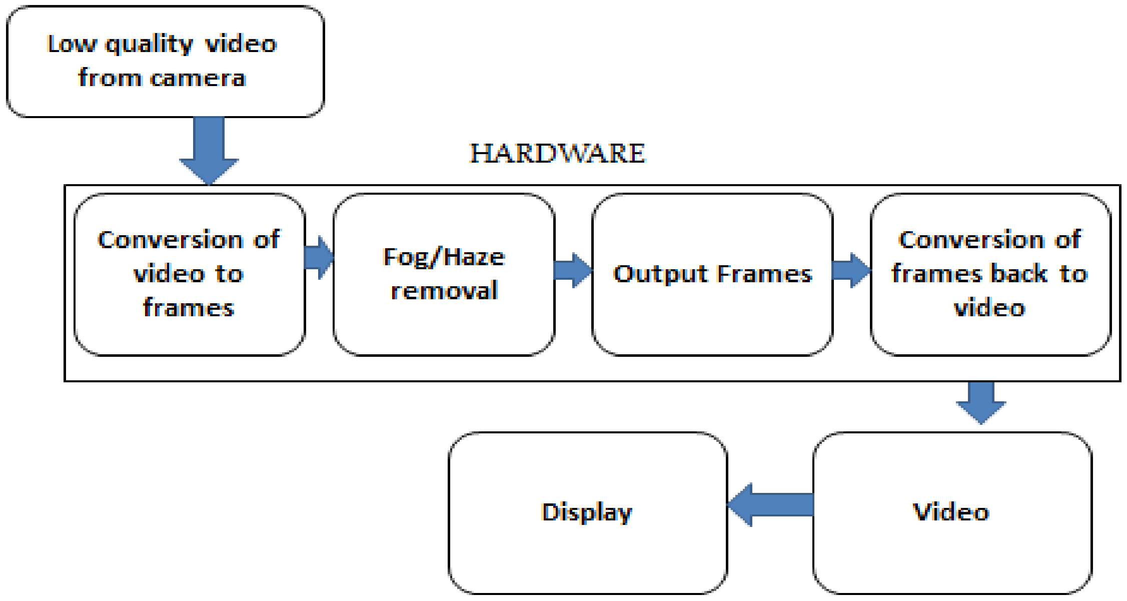

Low quality video from camera
HARDWARE
Conversion of video to frames
Fog/Haze removal
Output Frames
Conversion of frames back to video
Video
Display

Figure 3.1 Project Block Diagram

3.2 Program Planning

The works continues with the literature reviews on Fast visibility restoration from a single color or grey level and about visibility in bad weather from a single image. We deal with the single color image because it is important to understand the fundamental concept and operations carried out to handle a single plane of an image. To design the whole systematic project consecutive images are handled in three planes i.e. red, green and blue.

Several tools and techniques are understood and familiarized to deal with the consecutive stream of images in three color plane making a smooth video. These include familiarizing with techniques to preserve image edges, color restoration, atmospheric veil extraction, conversion from video to frames or consecutive images and from frames to video back , tone mapping, median filtering. The system is running on Microsoft windows as software is installed in the system. Matlab was understood thoroughly for video transmission to hardware, i.e. FPGA kit through serial interfacing (RS 232).

For designing different modules of algorithm initially, we write different modules using MODELSIM and this software was used to extract different modules.

The project workflows continue with designing and creating different modules, integrating these modules using Xilinx ISE thus developing the complete prototype. Finally, the VGA monitor is interfaced with the FPGA kit using VGA port for displaying clear video coming from webcam interfaced with Microsoft windows using Matlab.

The code is burnt on FPGA kit using Xilinx ISE software for creating bit file of code and ADEPT for interfacing FPGA kit with Microsoft system. The system used to get data from the interfaced webcam, video is captured in Matlab, then sent to the FPGA board, processing is done on the board, and then data is sent to VGA port. At the end, output is displayed, both on VGA monitor as well as Amber LED of the FPGA board turn on depicting successful simulation of algorithm on FPGA.

3.3 The Hardware

The hardware of the project is shown in figure 3.2. It is designed in such a way to ensure that it is cost effective and fulfills all the requirements of the project. For this reason webcam is used.

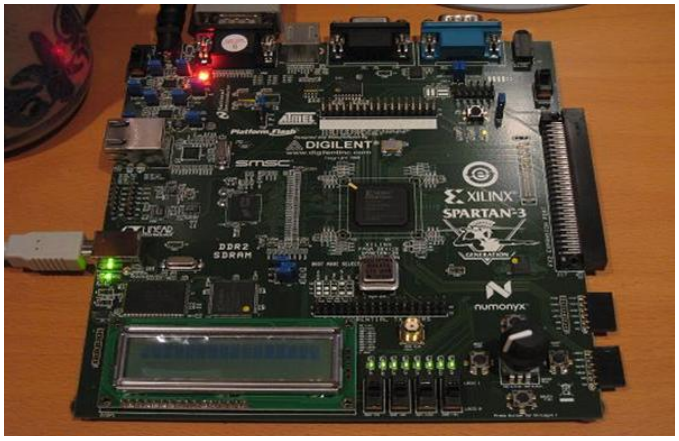


Figure 3.2 FPGA (hardware used)

3.3.1 The Setup

The requirement of the project was that image acquisition should be done in a confined environment. The reason for this was to make sure that the physical conditions like temperature, light ,humidity etc are least altered during the course of database compilation as these factors greatly affect the image acquisition. This results in different alterations to observed in the image preprocessing phase.

The setup is made of wood which includes FPGA kit enclosed in it and code burnt in the PROM of FPGA. A camera is attached to the system and they system is connected with the FPGA through serial port of FPGA. A monitor is interfaced with FPGA which is used to display the enhanced video after passing through the algorithm.

The whole setup is done in such a way that it is affected minimum by different weather conditions. For this reason FPGA is enclosed in the wooden box to avoid the effects of fog /haze on the kit and also wood is an insulator, so it avoids heating of the hardware.

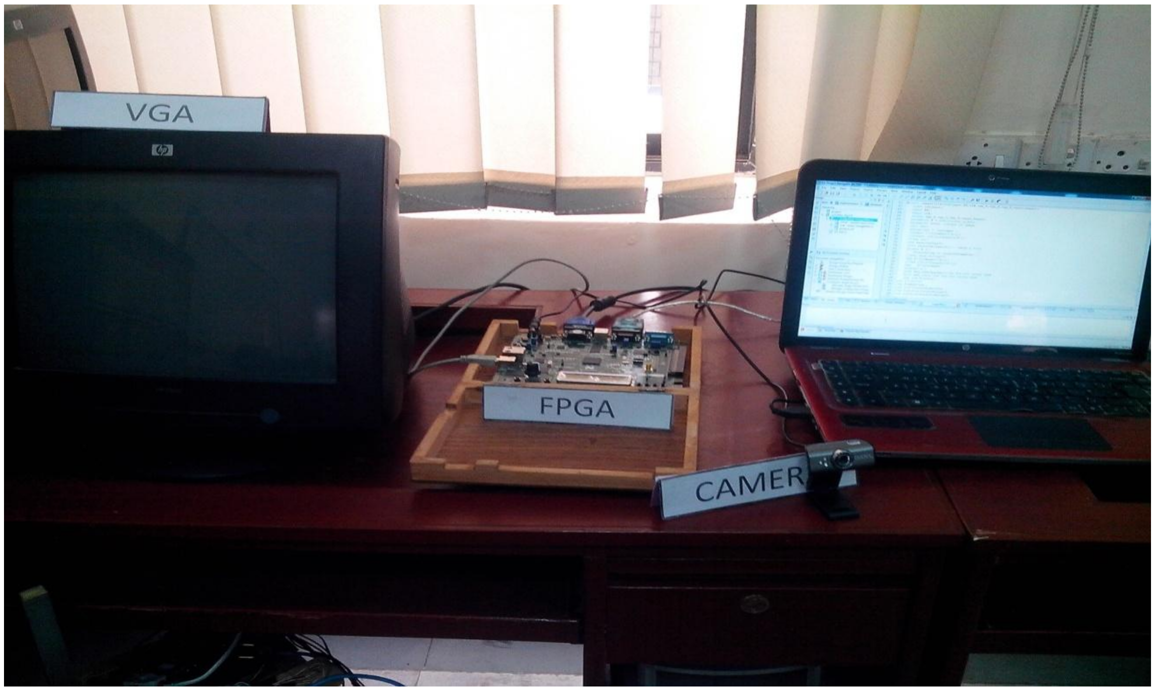


Figure 3.3 The complete setup

3.3.2 Connecting the camera

One of the main aims of this project was to make it cost effective. Since FPGA kit is of high cost and the camera which are available to come up to the requirement of the project were very costly, so a simple low cost camera, i.e. a webcam is used. The objective of using this camera is to take video in different available sizes depending on the availability of the display screen size and camera resolution. Since high resolution affects the frame rate of video, for real time videos one of the two is to compromised. So to get better and high frame rate resolution is limited.

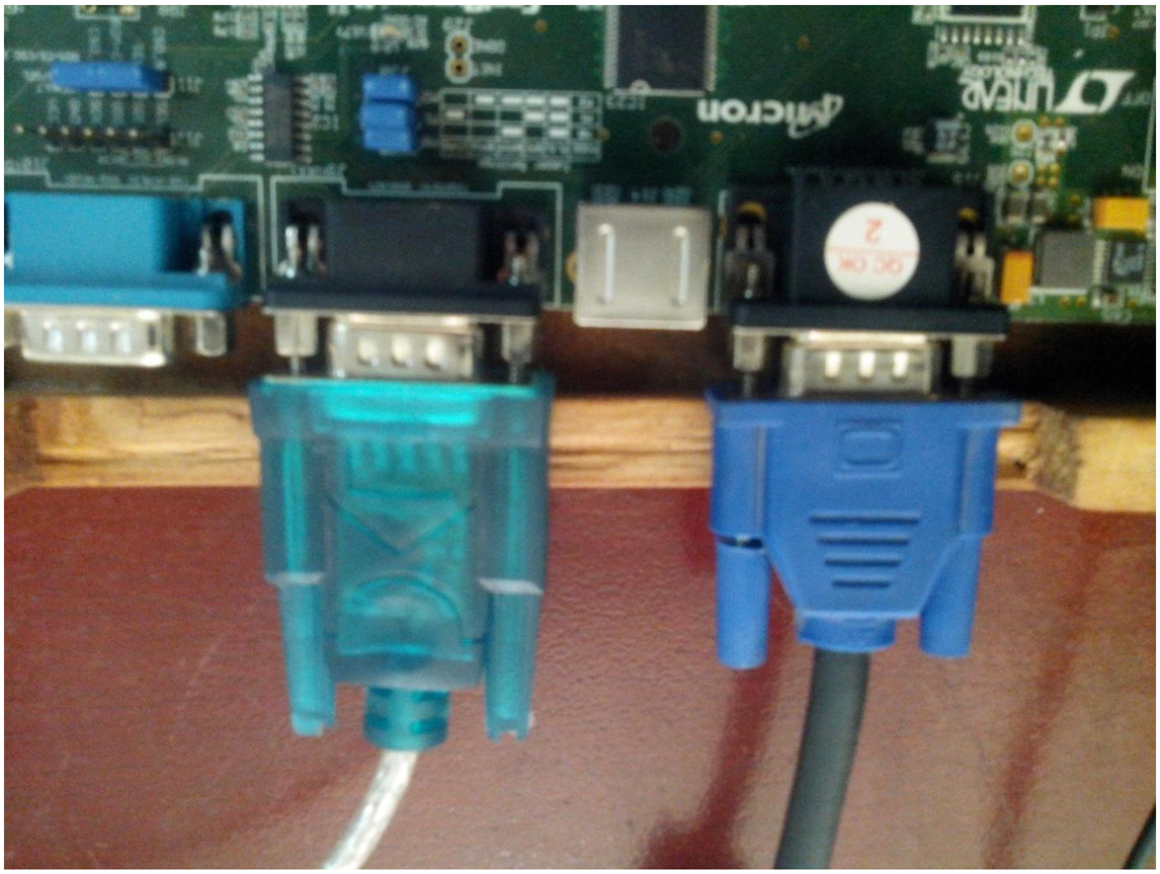


Figure 3.4 The serial and VGA port

3.3.3 Serial interface (RS-232)

A serial interface is a simple and easy way to connect an FPGA to a PC. In the project, a serial interface is created to connect an asynchronous serial link like RS-232 in an FPGA consisting of three parts (i) Baud generator, (ii) Transmitter, and (iii) Receiver.

3.3.3.1 How the RS-232 serial interface works

An RS-232 interface has many characteristics including usage of a connector "DB-9", having 9 pins, bidirectional, full-duplex communication (data can be send and receive at same time) and communicate at a maximum speed of 10KBytes/s approx.

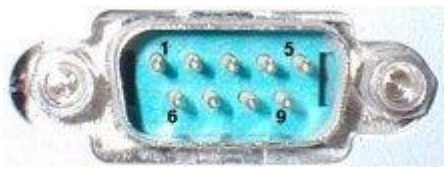


Figure 3.5 serial connector

3.3.3.2 DB-9 connector

It has 9 pins, but the 3 important ones are:

- Pin 2: RxD (for receiving data).
- Pin 3: TxD (for transmitting data).
- Pin 5: GND (ground pin).

We use three wires to connect these three pins for sending and receiving data.

3.3.3.3 Serial communication

Data is sent one bit at a selected chunk of time. A single wire is used for each direction. Computer needs at least several bits of data, so the data is "serialized" before being sent. Data is sent in chunks of 8 bits. The LSB (data bit 0) is sent first followed by the MSB (bit 7).

3.3.3.4 How fast can we send data?

Baud tells about speed which specifies how many bits-per-seconds can be sent. Common implementations of the RS-232 connector do not allow any speed to be used. Standard speed is settled for the project having common values of 1200 bauds, 9600 bauds, 38400 bauds or 115200 bauds (usually the fastest communication speed through serial connector).

At 115200 bauds, each bit lasts for (1/115200) = 8.7µs. When we transmit 8-bits data, it lasts for 8 x 8.7µs = 69µs. two bits are added because each byte requires an extra start and stop bit, so we actually need 10 x 8.7µs = 87µs, thus giving a speed of 11.5KBytes per second which can be considered as maximum speed at this baud rate.

At 115200 bauds, some PCs with large functionality chips require a "long" stop bit (1.5 or 2 bits long) which make the maximum speed drop to around 10.5KBytes per second. This factor adds some delay and speed of the video transmission is affected.

3.3.3.5 Baud generator

We want to use the serial link at maximum speed, i.e. 115200 bauds. slower speeds are easy to generate. FPGAs usually run at speed well above 115200Hz . So we use a high-speed clock and divide it down to generate a "tick" as close as possible to 115200 times a second.

3.3.3.6 RS-232 transmitter

It works likes (i) The transmitter takes an 8-bit data, and serializes it (starting when the "TxD_start" signal is asserted), (ii) The "busy" signal is asserted while a transmission occurs. The "TxD_start" signal is ignored during that time.

The RS-232 parameters used are fixed, i.e. 8-bits data, no-parity bits, 2 stop bits etc.

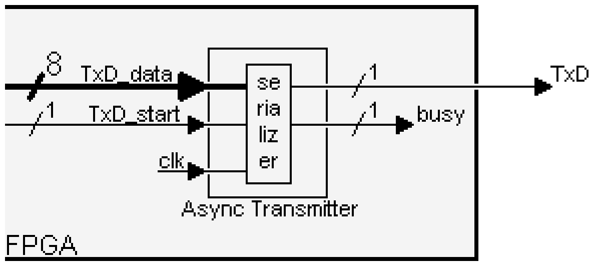


Figure 3.5 a sync transmitter

3.3.3.7 RS-232 receiver

Our implementation works like (i) the module assembles data from the RxD line as it comes. (ii) As a byte is being received, it appears on the "data" bus. Once a complete byte has been received, "data_ready" is asserted for one complete clock.

"data" is valid only when "data_ready" is asserted. The rest of the time, don't use it as new data may come that shuffles it.

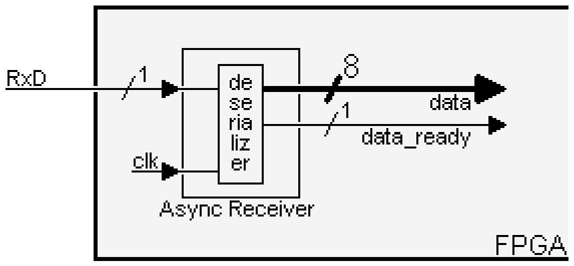


Figure 3.6 a sync receiver

3.3.3.8 How to use the RS-232 transmitter and receiver

This design allows controlling of FPGA pins from our PC (through our PC's serial port).

In more details, the design (i) Creates 8 outputs on the FPGA (port named "GPout"). GPout is updated by any character that the FPGA receives. (ii) Creates 8 inputs on the FPGA (port named "GPin"). GPin is transmitted every time the FPGA receives a character.

The GP(general purpose) outputs can be used to control anything remotely from the PC, might be LEDs or LCD on FPGA board.

3.3.4 Interfacing the monitor

The VGA monitor is interfaced with the VGA port of the FPGA kit. It displays the consecutive stream of images with a specified frame rate so that images form a video.

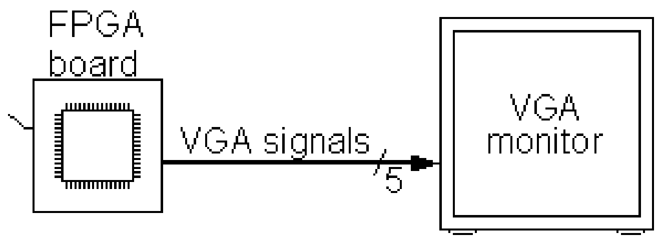


Figure 3.7 VGA signal transmission

3.3.5 VGA Interface

A VGA interface has 15 pins and requires 5 pins that transmit signals to display a picture:

- R, G and B (red, green and blue signals). Pin 1, 2 and 3.
- HS and VS (horizontal and vertical synchronization). Pin 13 and 14.

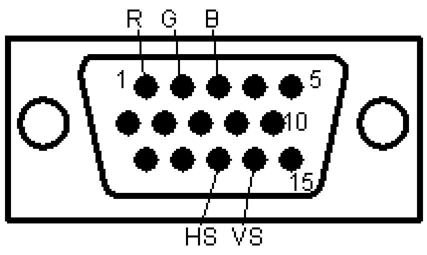


Figure 3.8 VGA connector

The transmit signals are of two types, i.e. analog and digital. The R, G and B are analog signals, while HS and VS are digital signals.

3.3.5.1 Creating a VGA video signal from FPGA pins

To drive the VGA connector, connections are made as following:

- Pins 13 and 14 of the VGA connector (HS and VS) are digital signals so can be driven directly from two FPGA pins (or through low values resistors, like 10Ω or 20Ω).
- Pins 1, 2 and 3 (R, G and B) are 75Ω analog signals with nominal values of 0.7V. With 3.3V FPGA outputs, we use three 270Ω series resistors. The resistors form voltage dividers by using 75Ω resistors in the monitor inputs so that 3.3V become 3.3*75/(270+75)=0.72V, pretty close to 0.7V. Driving these 3 pins with different combinations of 0's and 1's gives us up to 8 colors.
- Pins 5, 6, 7, 8 and 10 are Ground pins.

3.3.5.2 Frequency generator

A monitor always displays a picture line-by-line starting from top-to-bottom. Each line is drawn from left-to-right. That cannot be changed.

We start by sending short pulses on HS and VS at fixed intervals to specify the drawing of an image on a monitor. HS makes a new line to start drawing; while VS tells that the bottom has been reached (makes the monitor go back up to the top line).

For the standard VGA video signal (640x480 resolution), the frequencies of the pulses should be:

Vertical Freq (VS)	**Horizontal Freq (HS)**
60 Hz (=60 pulses per second)	31.5 kHz (=31500 pulses per second)

The sync generator is best rewritten to be used as a Verilog module where R, G and B signals are generated outside. Also the X and Y counters are more useful when they start counting from the drawing area.

3.4 The Techniques

The techniques which we used in our algorithm are described in detail

3.4.1 Median filter

Median Filter is a type of filter that is used for smoothening and reducing the noise from an image while preserving strong edges. It is a simple filter that uses the median to obtain its result. What the filter does is replacing the value of a pixel by the median of the values of surrounding pixels up to a defined range. In the case of noise this works because the median is unaffected by the outlying values from the noise for the time there is not too much noise. Same procedure goes for preserving edges, however thin lines on a plain background will not be preserved.

3.4.2 Bilateral filter

A Bilateral Filter is a filter based on Gaussian blur but preserves edges. The Gaussian blur uses the same Gaussian kernel for every pixel of the image. The kernel is used for averaging the pixels on weight. The Bilateral Filter changes the kernel to conserve edges. The results of applying bilateral filter to an image of air light. The edges are preserved by taking the change of intensity into account and penalizing large changes. The change in intensity is taken into account because high changes in intensity are observed as edges. By penalizing the pixels with high difference in intensity they affect the smoothing process less leading to maintaining strong edges within an image. Intuitively this can be seen as smoothing over pixels while excluding pixels whose intensity differs significantly.

A Median Filter only requires a window size as input whereas a Bilateral Filter also requires parameters for the change in intensity.

3.4.3 Sobel module

Edges describe boundaries of an image and are hence of fundamental importance in image processing. Edges in images contain strong intensity contrasts as intensity changes from one pixel to the next. Edge detection of an image preserves the important structural properties in an image, significantly reduces the amount of data at that point and filters out useless information. Edge detection can be performed by using different techniques. These methods may be grouped into two categories, Laplacian and gradient. The gradient method detects the edges by taking the first derivative of the image and look for the maximum and minimum in that image. The Laplacian method takes second derivative of the image and searches for zero crossings to find edges. An edge has the shape of an increasing ramp and calculates the derivative of the image can highlight its location.

Sobel filter calculates an approximation of the image intensity gradient function. It results in the correspondence of gradient vector at each point of the image. The operator calculates the direction of the largest possible increase from light to dark and the rate of change in that direction. The result therefore shows how "abruptly" or "smoothly" the image changes at that point, and therefore how an edge is represented, as well as how that edge is to be oriented. An edge in an image can be pointed in a variety of directions thus using four filters to detect diagonal, horizontal and vertical edges in the image which is blurred. The edge detection result returns the value of first derivative in the horizontal direction (Gx) and vertical direction (Gy). From these filters and first derivative, edge gradient and direction is determined .

The edge direction angle is rounded to one of four angles representing vertical, horizontal and the two diagonals. Mathematically, the gradient of a two-variable function (the image intensity function) is at each image point a 2D vector with the components given by the derivatives in the horizontal and vertical directions. At each pixel, the gradient vector points to the direction where there is the largest possible intensity increase, and the gradient vector length corresponds to the rate of change in that direction. This shows that the result of the Sobel operator at an image point i.e. in a region of constant image intensity is a zero vector and at a point where an edge occurs is a vector which points across the edge, from darker to brighter regions (intensity increasing from 0 to 1).

3.4.4 FPGA Implementation

Gaussian filter is used to reduce noise and minimize extra details of the image to perform smooth edge detection with the help of 2D-convolution which is important to modern image processing. A window of some specific size and shape is set for applying algorithm over image. A specific counter for infinite value is used to take a consecutive sequence of images (in the form of video).

Counter

```
always @ (posedge clk or negedge rst_n)
begin
if (rst_n==0)
counter1<=0;
else

  begin
counter1<=(counter1+1);
        enable=0;

   end
end
always @ (posedge clk or negedge rst_n)
begin
counter1<=0;
enable<=1;
end

always @ (posedge clk or negedge rst_n)
begin
if (counter2<2050)
        counter2<=counter2+enable;
else
begin
        counter2<=0;
        done<=1;
end
```

Median filter is applied on each coming pixel continuously giving a median value at each pixel. The median window is specified according to the coming stream of videos. A counter is used which sets the window of median filter accordingly.

Calling median filter for three planes

```
median uutR(clk, rst_n, donei, outR);
median uutG(clk, rst_n, donei, outG);
median uutB(clk, rst_n, donei, outB);
```

After applying median filter to each coming pixel stream, atmospheric veil is extracted and a grey level image is formed which is then passed to each color plane, i.e. Red, Green and Blue. A “div” core is written in Xilinx ISE for specifying division

```
div_core YourInstanceName
 (              // div 1
        .clk(clk),
        .dividend(outR*3), // Bus [15 : 0]
        .divisor(nbfo), // Bus [15 : 0]
        .quotient(s1)); // Bus [15 : 0]
        //.fractional(outR*3/nbfo)); // Bus [15 : 0]
```

the minimum value of image is taken which is then stored in each plane and color is restored accordingly

```
assign nbfo = outR+outG+outB ; // divide by 3
assign R1=(s1)**balance;  // div 1
assign G1=(s2)**balance;  // div 2
assign B1=(s3)**balance;  // div 3


assign nbfo1= (R1+G1+B1); // divide by 3
assign R2=(t1);     // div 4
assign G2=(t2);     // div 5
assign B2=(t3);     // div 6
assign R3=(u1);     // div 7
assign G3=(u2);     // div 8
assign B3=(u3);
```

taking minimum value

```
always@(*)
begin
if (G3<B3)

begin
if (G3<R3)
min = G3;
end

else if (R3<B3)
begin
if (R3<G3)
min = R3;
end

else
min=B3;
end

assign w= min;

median uutw(clk, rst_n, donei, wm);
```

The image is normalized when subtraction is done from minimum value

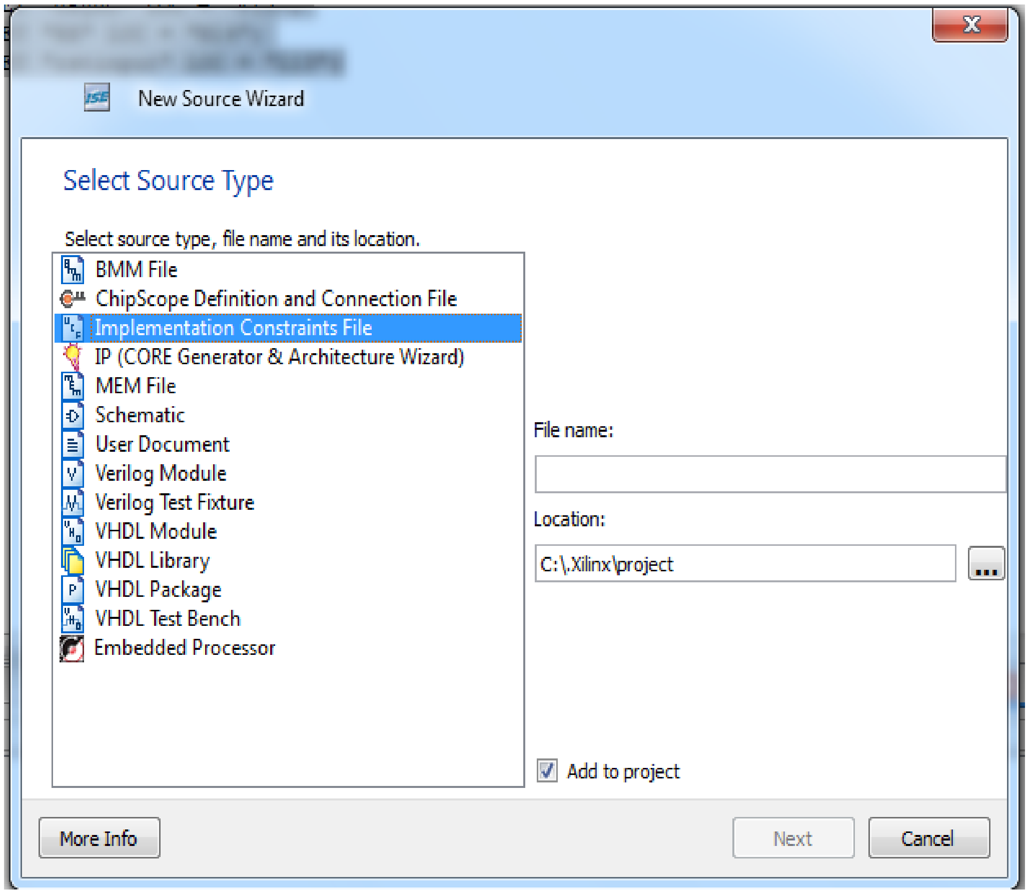
```
assign temp= w[index]-wm[index];
always @ (posedge clk or negedge rst_n)
begin
if (temp[7]==1)
sw[index]=(~temp+1);

else
sw[index]=temp;
end
```

After white balancing the image is restored having clear color of each pixel and hence fog is removed.

3.4.5 UCF Implementation

UCF file constraints are used during the implementation process f the top level module. We can enter the timing and placement constraints in the UCF file and the addresses of all the pins used for taking input and giving output of the FPGA kit. We have edit UCF file in the project navigator by following :

From tool bar, select project then new source a box will appear. From the box select " Implementation Constraints File" , add file name and UCF is appeared in the design panel.



New Source Wizard

Select Source Type

Select source type, file name and its location.

- BMM File
- ChipScope Definition and Connection File
- Implementation Constraints File
- IP (CORE Generator & Architecture Wizard)
- MEM File
- Schematic
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name:

Location: C:\.Xilinx\project

[✓] Add to project

Figure 3.9 adding UCF in design panel

Open the UCF from the design panel and write down the input, output, clock and reset pin numbers in the UCF text plane in the following format.

```
NET "vga_R" LOC = "H14";
NET "vga_G" LOC = "H15";
NET "vga_B" LOC = "G15";
NET "clk" LOC = "C9";
NET "vsync" LOC = "F14";
NET "hsync" LOC = "F15";
NET "RX" LOC = "M14";
NET "rstinput" LOC = "L13";
```

Hence by UCF the address of input where the data is coming and the output where it has to display is added and this approach gives an easy way to configure the output.

CHAPTER 4

FUTURE ENHANCEMENTS AND CONCLUSION

4.1 Conclusion

The algorithm for the removal of effects of fog/haze is implemented on FPGA (Field Programmable Gate Array). The algorithm is written in the language of FPGA that is Verilog. There are basically three parts of the project

- Software
- Hardware
- Interfacing

4.1.1 Software

The software part of the project includes the programming for

- Camera interfacing
- Algorithm
- VGA interfacing

The camera interfacing part is done by using the serial port of the FPGA. The video from the camera is received by the computer and then is sent to the USB port through a code in Matlab. The USB port of the computer is connected to the serial port of FPGA, so this is how the video is received by the FPGA. Due to all this procedure there is some delay in reception of the corrupted video.

Then comes the algorithm part, the corrupted video that is received by FPGA, now passes through the algorithm for the removal of effects of noise which was present in the video because of fog and haze. The algorithm is applied on the video to remove these effects and to give a clear video without the effects of fog/haze.

After removal of effects of fog/haze from the video, FPGA passes this clear video to its VGA port which will display the clear video without these effects.

4.1.2 Hardware

Hardware consists of

- USB camera
- FPGA kit
- VGA monitor

USB camera is used for capturing the corrupted video with fog/haze.

The algorithm is implemented on FPGA for capturing of video, removing of effects of fog/haze and then displaying of video.

VGA monitor is used to display the clear results obtained after the application of algorithm on the video.

4.1.3 Interfacing

Camera interfacing and VGA interfacing is done with FPGA kit by writing the code in Verilog in the software known as Xilinx ISE and in which the bit file is generated for the code, for this interfacing. And in the end the code of algorithm and VGA and Camera interfacing is burnt on FPGA kit using software known as ADEPT.

4.2 Future Enhancements

As the camera interfacing is done using serial port of FPGA, so there is some delay in the fog removal process, so one of the future enhancements is that camera can be directly interfaced with the FPGA kit so that there should be no delay in the fog removal process.

Secondly this system could be made portable so that it can be installed anywhere and anytime including cars and other automobiles, so to avoid a number of accidents that occur because of increased effects of fog and unclear vision.

Thirdly an amendment can be made in the algorithm so that it may be able to remove the effects caused by snow in hilly areas and in other areas too, to get the best results of video making and photography in almost every kind of environment.

BIBLIOGRAPHY

REFERENCES

[1] N Hautire and J –P Tarel, ‘’ ***Fast Visibility Restoration from a Single Colour or Gray Level Image***’’ ,pp. 2201 – 2208, 12th IEEE conference from Sept. 29 2009-Oct. 2 2009

[2] Robby T. Tan, “***Visibility in Bad Weather from Single Image***”, 2008

[3] YunboRao, Leiting Chen, “***A Survey of Video Enhancement Techniques*****”,** 2012

[4]Digital Image Processing Using MATLAB by Rafael C. Gonzalez, Richard E. Woods and Steven L. Eddins.(chapter 3,4)

[5]H.D Cheng , X.J. ShI, ‘’***A simple and effective histogram equalization approach to image enhancement’’,*** pp. 158-170, volume 14, March 2004.

[6]Jing Yu, Qingmin Liao, ‘’***Fast single image fog removal using edge-preserving smoothing’’*** *,*pp. 1245-1248 , IEEE conference 2011

[7] FPGA Based System Design By Wayne Wolf, Prentice Hall Modern Semiconductor Design Series

[8] Farbiz, F. Amirkabir Univ. of Technol., Tehran, *‘’ A new fuzzy logic filter for image enhancement’’*, pp. 110-119, vol. 30.

[9]Babu T Chacko, Siddharth Shelly, **‘’Real-Time Video Filtering and Overlay Character Generation on FPGA**’’, **pp. 184-189, International conference** on Recent Trends in Information, Telecommunication and Computing**, IEEE 2010.**

[10]www.mathworks.com

[11] www.edaboard.com

[12] www.xilinx.com

[13] Programming Streaming FPGA Applications Using block Diagrams in Simulinkby Brian C. Richards, Chen Chang, John Wawrzynek, Robert W. Brodersen.

APPENDIX A

Source code for median filter

```
module median(clk, rst_n, donei, out);

integer i, j;

input clk, rst_n;

reg [7:0] mem[0:625];

//wire a, b;

output reg donei;

output  out;

reg [0:7] temp [0:8];

//wire i, j;

//wire unsigned [0:7] sum;

//wire unsigned [0:15] ans;

reg donej;

always @ (posedge clk or negedge rst_n)

begin

   if (!rst_n)

      begin

      i<=0;

      donei <=0;

      end

      else if  ((i<625) && (donej))

                  begin

                  i=i+1;

                   donei<=0;

             end

                  else

        begin

                  i<=0;
                  donei=1;
                   end
end
always @ (posedge clk or negedge rst_n)
begin
if (!rst_n)
begin
donej<=1;
j<=0;
end
else if (j<i)
begin
j=j+1;
donej<=0;
end
else
begin
j<=0;
donej<=1;
end
mem[i] <= mem[j];
mem[j] <= mem[i];
end
assign out =mem[312] && { 8 { donei}};
endmodule
```

COMPLETE CODE FOR FPGA

```
module imagefilter(rstinput,RX,clk,vga_R,vga_G,vga_B,vsync,hsync);
      input rstinput;
      input RX;
      input clk;
      output vga_R,vga_G,vga_B,vsync,hsync;
      parameter N = 100;//total pixels
      parameter n=10; //width of image
      wire rst;
      assign rst = ~rstinput ;
      //input [7:0]image[0:N-1];
      //output [10:0]fimage[0:N-1];
      //reg rst;
      reg receiveflag=0;
      wire receiveflagwire;//---mayb a wire
      always @ (*)
            receiveflag <= receiveflagwire;
      //reg receiveflag=0;
      reg [7:0]image[0:N-1];
      //reg [10:0]fimage[0:N-1];
      reg   [7:0]fimage;
      integer u ;
      wire Rxd_endofpacket;//we are not using them
      wire RxD_idle;//we are not using them
      reg [14:0]i=n+1;
reg [7:0]j=0;
reg fimageflag ;
reg [14:0]countpixels=0;
reg [14:0]allreceiveflag=0;
reg allreceiveconfirmflag=0;
```

```
      //-----------------------------------------------------------
---------------

 serialreceive serial(RxD_bit,clk,receivedata, receiveflagwire);

//--------------initialise RAM to zero---------------------------
initial
begin
      //receiveflag=1;
      for (u=0; u<N; u=u+1)
      begin
                  image[u]<=0;
                  //fimage[u]<=0;
      end
end
//-----------------------------------------------------------------

/*
reg [2:0]states=0;
reg [2:0]nextstate=0;
always @ (posedge clk)
begin
      case(state)
      3'b000:     if (allreceiveflag == 99) state <= 3'b001 else state
<= 3'b000;
      3'b001:     if (fimageflag ==1) state <= 3'b010             else
state <= 3'b001;
      3'b010:     state <= 3'b000;
      default:    state <= 3'b000;
      endcase
end
```

```
always @ (state)

begin

      case (state)

      3'b000:     begin  fimageflag <=0; allreceiveconfirmflag <=0; end

      3'b001:     begin  fimageflag <=0; allreceiveconfirmflag <=1; end

      3'b010:

end*/

//--------------------------------------receiving image-----------------
--------

reg [14:0]ur;

always @ (posedge receiveflag or negedge rst)

begin

      if (~rst)

            ur<=0;

      else

            ur <= ur +1;

end

always @ (posedge receiveflag)

begin

      image[ur]<=receivedata;

end

//-----------------------------------------------------------------------
-------

always @ (posedge receiveflag or negedge rst)

begin

      if (~rst)

      begin

      allreceiveflag <=0;

      allreceiveconfirmflag <=0;

      end

      else
      begin

      if (allreceiveflag == N-1)

      begin

      allreceiveflag <=0;

      allreceiveconfirmflag <=1;

      end

      else

      begin

      allreceiveflag <= allreceiveflag + 1;

      allreceiveconfirmflag <=0;

      end

      end

end

//------------------RUN WHEN IMAGE IS RECEIVED-------------------------
--------------

always @ (posedge clk)

begin

      if (allreceiveconfirmflag & ~fimageflag & countpixels <(N-(n+n+n-
2+n-2)) )

      countpixels <= countpixels + 1;

      else

      countpixels <=0;

end

//-------------------CHECK FOR COMPLETE FILTER ACTION------------------
---------------------

always @ (posedge clk or negedge rst)

begin

      if (~rst)

            fimageflag <= 0;

      else

      begin
      if (countpixels == (N-(n+n+n-2+n-2)) | fimageflag
)////////////error here| allreceiveconfirmflag

            fimageflag <= 1;

      else

            fimageflag <= 0;

      end

end

//-------------------------BASIC COUNTER FOR HORIZONTAL PIXEL MOVEMENT-
---------------------------------

always @ (posedge clk)

      begin

            if (allreceiveconfirmflag)

                  j<=j+1;

            else

                  j<=0;

      end

//-----------------------------------------------------------------------


reg TXD_start=0;

wire TXD_busy;

reg [7:0]txddata;

//


// vga module

main uut (clk,receive,vga_R, vga_G,vga_B, vsync, hsync );

always @ (posedge clk or negedge rst)

begin

      if (~rst)

            i<=(n+1);

      else

      begin
      if (allreceiveconfirmflag & ~fimageflag & ~TXD_busy)
            begin
                  if( j == (n-3) )
                                          begin i <= i + 3 ; end
                  else
                                          begin i <= i + 1; end
                  end
            else
            i <=i;
      end
end
//------------------------------------------------------------------------
-----
always @(i)//posedge clk)
begin
            if (allreceiveconfirmflag & ~fimageflag)
                        fimage = (image[i-
n+1]+image[i+1]+image[i+n+1]+image[i-1]+image[i-1-n]+image[i-
1+n]+image[i+n]+image[i-n])>>3;
            else
                  fimage  = fimage ;
end


//-----------initialise serial transmission--------------



always @ (fimage)
begin
      txddata = fimage;
end
reg [1:0]counter_serial=0;
wire counterbit ;
```

```
always @ (txddata,counterbit)

begin

      if (TXD_busy == 0)

      TXD_start=1;

      else

      TXD_start=0;

end


assign counterbit = counter_serial[1];


always @ (posedge clk)

begin

      counter_serial <= counter_serial +1;

end


SERIAL


module serialreceive(RxD_bit,clk,receivedata, receiveflag);


input clk;

input RxD_bit;

output reg [7:0] receivedata=0;

output reg receiveflag=0;


// BAUD RATE FOR RECEIVIG DATA

      wire baudclk;

      parameter cck_freq = 16384;

      parameter baudrate      =     151;

      //baudrate = 460800 bps

      //clock freq = 50MHz
      //108 cycles of crystal freq is our one cycle of serial

      //parameter valuetobeadded = 5208;

      //reg [6:0]try = 0;


      reg [14:0]baudgenerator=0;

      //reg baudclk;

      reg baudstartbit=0;


      always @(posedge clk)

      begin

            if (baudstartbit == 1)

                  baudgenerator <= baudgenerator[13:0] + baudrate;

            else

                  baudgenerator <=0;

      end



      assign baudclk = baudgenerator[14];



// STATEMACHINE FOR RECEIVE DATA

reg [3:0] recstate=0;


always @(posedge clk)

begin

case(recstate)

  4'b0000: if(~RxD_bit) recstate <= 4'b1000; // start bit found?

  4'b1000: if(baudclk) recstate <= 4'b1001;  // bit 0

  4'b1001: if(baudclk) recstate <= 4'b1010; // bit 1

  4'b1010: if(baudclk) recstate <= 4'b1011; // bit 2
  4'b1011: if(baudclk) recstate <= 4'b1100; // bit 3

  4'b1100: if(baudclk) recstate <= 4'b1101; // bit 4

  4'b1101: if(baudclk) recstate <= 4'b1110; // bit 5

  4'b1110: if(baudclk) recstate <= 4'b1111; // bit 6

  4'b1111: if(baudclk) recstate <= 4'b0001; // bit 7

  4'b0001: if(baudclk) recstate <= 4'b0010;

  4'b0010: if(baudclk) recstate <= 4'b0000;// stop bit

  default: recstate <= 4'b0000;

endcase

end

//DATA RECEPTION

always @ (posedge baudclk)

begin

      case (recstate[3:0])

      4'b1000:    begin receivedata <= {receivedata[7:1],RxD_bit};
receiveflag <= 0; end

      4'b1001:    begin receivedata <=
{receivedata[7:2],RxD_bit,receivedata[0]};receiveflag <= 0; end

      4'b1010:    begin receivedata <=
{receivedata[7:3],RxD_bit,receivedata[1:0]};receiveflag <= 0; end

      4'b1011:    begin receivedata <=
{receivedata[7:4],RxD_bit,receivedata[2:0]};receiveflag <= 0; end

      4'b1100:    begin receivedata <=
{receivedata[7:5],RxD_bit,receivedata[3:0]};receiveflag <= 0; end

      4'b1101:    begin receivedata <=
{receivedata[7:6],RxD_bit,receivedata[4:0]};receiveflag <= 0; end

      4'b1110:    begin receivedata <=
{receivedata[7],RxD_bit,receivedata[5:0]};receiveflag <= 0; end

      4'b1111:    begin receivedata <=
{RxD_bit,receivedata[6:0]};receiveflag <= 0; end

      4'b0001: begin receivedata <= receivedata; receiveflag <= 1; end

      default: begin receivedata <= receivedata; receiveflag <= 0; end

      endcase

end
```

```
always @ (recstate[3:0])

begin

      if (recstate[3:0]== 4'b0000)

            baudstartbit <=0;

      else

            baudstartbit <=1;

end

//-----------------------------------------------------------------------

/*reg [4:0]counterforflag=0;

always @ (posedge clk)

begin

      if (receiveflag == 1)

            counterforflag <= counterforflag[3:0] + 1;

            else

            counterforflag <=0;

end

/*always @ (counterforflag)

begin

      if (counterforflag[4]==1)

            begin

                  receiveflag <=0;

            end

end*/

endmodule


DISPLAY

module main(clk,receive,vga_R, vga_G,vga_B, vsync, hsync


       );
       input [7:0] receive;

       input clk;

       reg [7:0]txddata;

       output  vsync, hsync, vga_R, vga_G, vga_B;

wire inDisplayArea;

wire [9:0] CounterX;

wire [8:0] CounterY;


assign vga_R =txddata;

assign vga_G =txddata;

assign vga_B =txddata;


hvsync_generator uut(clk, vsync, hsync);

endmodule


module hvsync_generator(clk, vsync, hsync, inDisplayArea, CounterX,
CounterY);

input clk;

output vsync, hsync;

output inDisplayArea;

output [9:0] CounterX;

output [8:0] CounterY;

//////////////////////////////////////////////////

reg [9:0] CounterX;

reg [8:0] CounterY;

wire CounterXmaxed = (CounterX==10'h2FF);

always @(posedge clk)

if(CounterXmaxed)

      CounterX <= 0;

else

      CounterX <= CounterX + 1;
```

```
always @(posedge clk)

if(CounterXmaxed) CounterY <= CounterY + 1;


reg   vga_HS, vga_VS;

always @(posedge clk)

begin

      vga_HS <= (CounterX[9:4]==6'h2D); // change this value to move
the display horizontally

      //horizontal screen positioning, reducing value shifts screen
horizontally right and vice versa

      vga_VS <= (CounterY==490); // change this value to move the
display vertically

      //vertical screen positioning, reducing this value shifts screen
vertically down and vice versa

end

reg inDisplayArea;

always @(posedge clk)

if(inDisplayArea==0)

      inDisplayArea <= (CounterXmaxed) && (CounterY<480);

else

      inDisplayArea <= !(CounterX==639);

assign hsync = ~vga_HS;

assign vsync = ~vga_VS;

endmodule


SERIAL TRANSMISSION

module ser(txddata,TxD_start,TXD_busy,clk,TXD);


      input TxD_start,clk;

      input [7:0]txddata;

      output reg TXD;

      output reg TXD_busy;
      wire baudclk;
      parameter cck_freq = 16384;
      parameter baudrate      =     151;
      //108 cycles of crystal freq is our one cycle of serial
      //parameter valuetobeadded = 5208;
      reg [6:0]try = 0;

      reg [14:0]baudgenerator=0;
      //reg baudclk;

      always @(posedge clk)
      begin
            baudgenerator <= baudgenerator[13:0] + baudrate;
      end
            assign baudclk = baudgenerator[14];



/*    wire outa;
      always @(posedge clk)
      begin

            if(try == 108)
            try <= 0;
            else
            try = try +1;
      end

      assign outa = (try == 108)?1:0;
*/
/*
```

```
      //end

      reg [14:0]a=0;

      always @(posedge baudclk)

      begin

            a<=a+1;

      end


      // 10 bits for the accumulator ([9:0]), and one extra bit for the
accumulator carry-out ([10])

reg [10:0] acc =0;   // 11 bits total!


always @(posedge clk)

  acc <= acc[9:0] + 59; // use only 10 bits from the previous result,
but save the full 11 bits


wire BaudTick = acc[10]; // so that the 11th bit is the carry-out

*/


//state machine for serial data transfer


reg [3:0] state=0;


always @(posedge clk)

case(state)

  4'b0000: if(TxD_start) state <= 4'b0100;

  4'b0100: if(baudclk) state <= 4'b1000; // start

  4'b1000: if(baudclk) state <= 4'b1001; // bit 0

  4'b1001: if(baudclk) state <= 4'b1010; // bit 1

  4'b1010: if(baudclk) state <= 4'b1011; // bit 2

  4'b1011: if(baudclk) state <= 4'b1100; // bit 3

  4'b1100: if(baudclk) state <= 4'b1101; // bit 4
  4'b1101: if(baudclk) state <= 4'b1110; // bit 5
  4'b1110: if(baudclk) state <= 4'b1111; // bit 6
  4'b1111: if(baudclk) state <= 4'b0001; // bit 7
  4'b0001: if(baudclk) state <= 4'b0000; // stop1
  //4'b0010: if(baudclk) state <= 4'b0000; // stop2
  default: if(baudclk) state <= 4'b0000;
endcase

reg muxbit=0;
reg [7:0]TxD_data;

always @ (*)
      begin
      TxD_data <= txddata;
      end

      always@(*)
      begin
      if(state == 0)
            TXD_busy <= 0 ;
      else
            TXD_busy <= 1;
      end
always @(state[2:0])
case(state[2:0])
  0: muxbit <= TxD_data[0];
  1: muxbit <= TxD_data[1];
  2: muxbit <= TxD_data[2];
  3: muxbit <= TxD_data[3];
  4: muxbit <= TxD_data[4];
  5: muxbit <= TxD_data[5];
  6: muxbit <= TxD_data[6];
  7: muxbit <= TxD_data[7];
endcase

// combine start, data, and stop bits together
always @ (*)
      TXD = (state<4) | (state[3] & muxbit);

Endmodule
```

APPENDIX B

Matlab code for Serial Transmission to FPGA

```
clc
clear all
close all
imaqhwinfo('winvideo')
imaqhwinfo('winvideo',2)
vidobj=videoinput('winvideo',2,'YUY2_160x120');
set(vidobj, 'ReturnedColorSpace', 'RGB');
start(vidobj)
preview(vidobj)


while(1)

frame = getsnapshot(vidobj);
% actual_image = imread('a.jpg');
% actual_image = rgb2gray(actual_image);
%
% actual_image = imnoise(actual_image,'salt & pepper',0.02);

%actual_image = 10.*randint(10,10,[1 10]);
actual_image = uint8(frame);
m = 15
n = 15


s2= serial('COM7');
s2.baudrate = 9600;
s2.databits = 8;
s2.stopbit = 1;
s2.InputBufferSize = 10000;
fopen(s2);

for j= 1:m
   for k = 1:n
       fwrite(s2,frame(j,k))
   end
end

end
```