HARDWARE REALIZATION OF VIDEO STITCHING ALGORITHM ON FPGAS

A Project Report

submitted by

SREE KALYAN PERURI

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2014

THESIS CERTIFICATE

This is to certify that the thesis titled **HARDWARE REALIZATION OF VIDEO STITCHING ALGORITHM ON FPGAS**, submitted by **Sree Kalyan Peruri**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr.K.Sridharan
Research Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: May 2014

ACKNOWLEDGEMENTS

I am thankful to **Dr.K.Sridharan** for his guidance throughout the project. His valuable insight helped me a lot for the progress of my project. I also thank my friends Hari Ananth, Tahiyah who helped me throughout the project by discussing various concept necessary for the project. I thank my college IIT Madras for providing me the infrastructure required for my project. I thank my parents for giving me the motivation and support all the way through. Finally, I thank my project mate M.S.Chandrasekhar for making the project go as smooth as possible.

ABSTRACT

KEYWORDS: Video Stitching; Hessian filters; SURF; Interest Points; Image Processing in Verilog

In this project, an efficient hardware architecture for Video Stitching is designed and implemented on a high speed parallel processing FPGA platform. An efficient algorithm has been designed to process two simultaneous video streams and display on a monitor using side by side configuration. An image feature detection algorithm called Speeded Up Robust Features is used which contains filtering an image using Hessian filters. Random sample consensus is used for feature matching to achieve video stitching. The architecture designed uses stream processing for hardware and parallel processing of FPGA to achieve real-time processing.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

SURF	Speeded Up Robust Features
IP	Interest Points
RANSAC	Random Sample Consensus
FPGA	Field Programmable Gate Arrays
SIFT	Scale Invariant Feature Transform

CHAPTER 1

INTRODUCTION

Technologies for aligning images and stitching them into seamless photo mosaics are among the oldest and most widely used in computer vision. The goal of video stitching is to create wide-angle and high-resolution panorama video from various video sources. Different from image stitching technologies, video stitching technologies require more strict real-time processing ability. Video stitching technologies have broad applications, e.g. largescale video surveillance, wide-view video conferences, video editing, video entertainments, video games, etc.
As the result of rapid development of the display industry, high resolution displays are coming into daily life of people and becoming highly affordable. There is a desired requirement to use wide-angle and high-resolution videos. Video stitching technologies may be a solution to create data for various high definition video requirements. Video stitching technologies are also a powerful tool to provide panorama videos in visual surveillance to monitor very large places, e.g. supermarkets, squares, long streets, large buildings, etc. Our target is an efficient technology, possibly real-time, for automated panoramic video construction to enable applications such as streaming videos from multiple capturing devices and allowing another party to watch the constructed panoramic video in real-time. This would offer the remote party a better viewing experience, in terms of wider field-of-view. In this report we discuss a new hardware implementation of Video Stitching algorithm based on high speed parallel processing FPGA.

1.1 Video stitching

The scope of video stitching algorithm is to join videos that are taken by two different cameras for a better field of vision. People are familiar with image stitching as it has a greater degree of freedom. This freedom arises from aligning the images by translation and rotation. Video stitching on the other hand has its restrictions.

There has to be an overlap between the two videos and without loss of generality the cameras are taken to be side by side. A video stitching algorithm has following stages.

1. Interest Point detection
2. Interest Point Matching
3. Overlap recognition and Aligning

Interest Point detection This is the first and most important part of video stitching. Interest points are used because they have well founded function, well defined position in image space, stable under local and global perturbations and immune to illumination variations. They are very useful in object tracking. The interest points have to be detected for all the video streams that need to be stitched.

Interest Point Matching Once the interest point locations have been found out, they have to be matched to find the common portion of the image in the video frames.

Overlap recognition and Aligning After finding the best match for the interest points in video frames, we have the information to see if the video frames can be stitched or not. When the video frames are good enough to stitch, the matching algorithm gives the amount of compensation and movement required to combine the common portions of the video frames. This is used to move the frames and combine the videos as required.

1.2 Literature Survey

The following section describes the study carried out in Video stitching in the literature. The first part of any video stitching algorithm is detecting the interest points. An Interest point is a point in an image which has the following features. It has a clear, preferably mathematically well-founded, definition, it has a well-defined position in image space, it is stable under local and global perturbations in the image domain as illumination/brightness variations, such that the interest points can be reliably computed with high degree of reproducibility. Old methods of corner point detection included Harris corner detection as described in C.Harris and M.Stephens (1988).

But for a video stitching algorithm, the accuracy of the corner points detected in C.Harris and M.Stephens (1988) is not sufficient. So, we look at some modern approaches such as D.Lowe (2004) which uses a Laplacian of Gaussian filter approach for detecting the Interest points. This process is called Scale Invariant feature Transform(SIFT) which is robust but when the algorithm is being implemented on a FPGA, the speed is more important than robustness. A new method is used in video stitching which is an optimized version of D.Lowe (2004) and is called Speeded Up Robust Features(SURF) explained in H.Bay and L.Gool (2006) . This is different from D.Lowe (2004) as it does not use LoG(Laplacian of Gaussian) filters, but uses an approximate version of it.

More detailed version of SURF can be found in Evans.C (2009) which explains how the algorithm can be approached from a theoretical point of view and translates how it can be implemented. Very few resources are available on SURF which are related to FPGA. WeiLong (2013) is one of the work done on FPGA but the algorithm described can only be applied to small size images as they are using only blockRAM and for higher image sizes more memory should be available.

The algorithm proposed here involves using the SURF algorithm as described in Evans.C (2009) and WeiLong (2013) but the difference is that the algorithm is optimized for bigger image sizes. An approach based on DDR memory available on the target FPGA board is used. This allows for more resource utilization compared to work done in the previous papers.

After the interest points have been found out as described in H.Bay and L.Gool (2006), the interest points have to be matched in order to align the frames correctly. An algorithm proposed in A.Fischler and C.Bolles (1981) gives detailed analysis of how sample matching can be done. It is called Random sample consensus. The proposed algorithm iscapable of interpreting/smoothing data containing a significant percentage of gross errors, and is thus ideally suited for applications in automated image analysis where interpretation is based on the data provided by error-prone feature detectors.

CHAPTER 2

ALGORITHM DESCRIPTION

This section describes the proposed algorithm that is used for video stitching. The various steps involved in video stitching are explained step by step. The flowchart for video stitching algorithm is shown in the Figure 2.1.

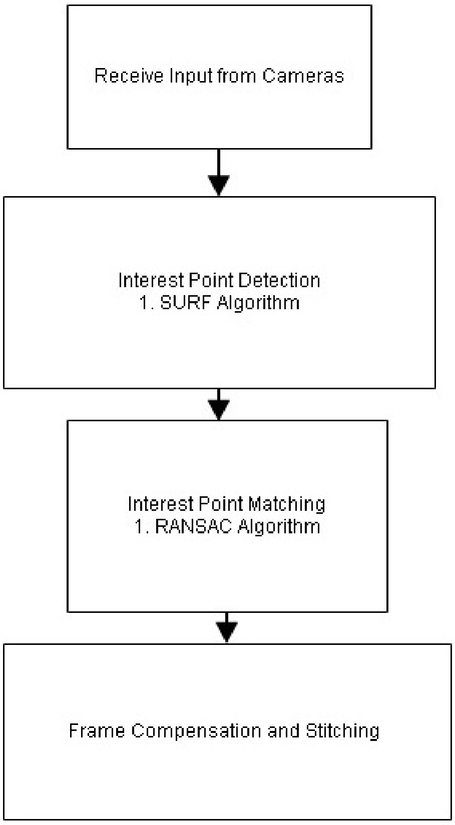


Figure 2.1: Algorithm flowchart

To find out how much the frames have to be compensated for video stitching, it is essential to find the distance between the common portions of both frames. In order to find this distance, the first step involved is finding the feature points of the frame which is followed by matching them with feature points of the other frame. The algorithms for finding and matching interest points are explained in this chapter.

2.1 SURF algorithm

For Interest Point detection we used the latest algorithm that is proposed in this field and it is called SURF. Proposed by H.Bay in 2006,SURF is the abbreviation of Speeded up Robust FeaturesH.Bay and L.Gool (2006) and it is a novel approach for a video stitching algorithm. This robust algorithm is described clearly in the sections below.

2.1.1 Integral Image

Integral Image is also called a summed area table which is a data structure and algorithm for quickly and efficiently generating the sum of values in a rectangular subset of a grid. Much of the performance increase in SURF can be attributed to the use of integral image as described in Viola and Jones (2001). The integral image is computed rapidly from an input image and is used to speed up the calculation of any upright rectangular area. Given an input image I and a point (x,y) the integral image $I_{\sum}$ is calculated by the sum of the values between the point and the origin. Formally this can be defined by the formula:

$$I_{\sum}(x,y) = \sum_{i=0}^{i\leq x} \sum_{j=0}^{j\leq y} I(x,y) \tag{2.1}$$

Using the integral image, the task of calculating the area of an upright rectangular region is reduced four operations. If we consider a rectangle bounded by vertices A, B, C and D as in 2.2, the sum of pixel intensities is calculated by:

$$\sum = A + D - (C + B) \tag{2.2}$$

Since computation time is invariant to change in size this approach is particularly useful when large areas are required. SURF makes good use of this property to perform fast convolutions of varying size box filters at near constant time. This is the standard method for SURF but in the algorithm that is described in this paper, this is not used as calculating the integral image needs a lot of memory. Although the calculation is faster when the integral image procedure is used, memory con-

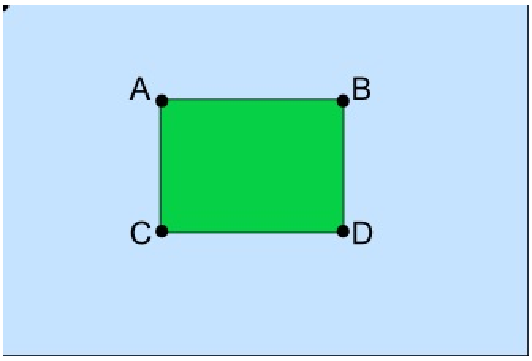


Figure 2.2: Area Computation using Integral Images

straints on the target board does not allow much freedom. The main reason for using the integral image is to speed up the calculations, but it can be done without using this procedure. More details about how it is done is explained in further sections.

2.1.2 Fast Hessian Detector

The SURF detector is based on the determinant of the Hessian matrix. In order to motivate the use of the Hessian, we consider a continuous function of two variables such that the value of the function at (x, y) is given by $f(x, y)$. The Hessian matrix H, is the matrix of partial derivatives of the function f.

$$H(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \tag{2.3}$$

The value of the discriminant is used to classify the maxima and minima of the function by the second order derivative test. Since the determinant is the product of eigenvalues of the Hessian we can classify the points based on the sign of the result. If the determinant is negative then the eigenvalues have different signs and hence the point is not a local extremum, if it is positive then either both eigenvalues are positive or both are negative and in either case the point is classified as an extremum.

Now we should translate this theory to work with images rather than a continuous function. First we replace the function values f(x,y) by the image pixel intensities $I(x, y)$. Next we require a method to calculate the second order partial derivatives of the image. In D.Lowe (2004) this is done using LoG which is a complicated procedure but in case of H.Bay and L.Gool (2006) the LoG is approximated by the filters shown in Figure 2.3.

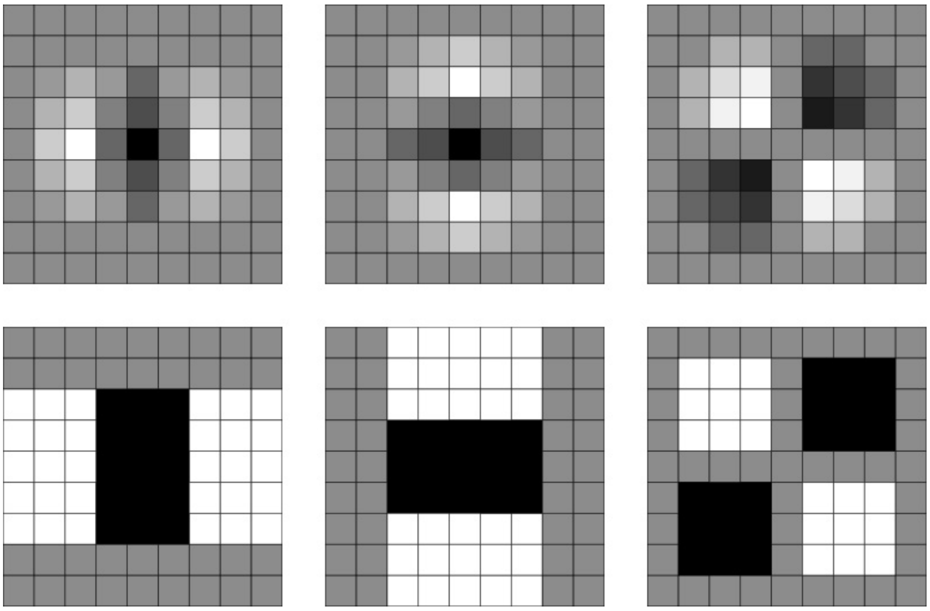


Figure 2.3: Figure showing the original LoG filters and approximated D_{xx}, D_{yy} and D_{xy} filters respectively

As shown in the figure the 9×9 approximated filters are used to calculate the determinant of Hessian Matrix. In Figure 2.3, the D_{xy} filter the black regions are weighted with a value of 1, the white regions with a value of -1 and the remaining areas not weighted at all. The D_{xx} and D_{yy} filters are weighted similarly but with the white regions have a value of -1 and the black with a value of 2. And the Hessian determinant is calculated using the formula shown below.

$$det(H_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2 \tag{2.4}$$

2.1.3 Scale Space Construction

After calculating the determinant of Hessian Matrix, in order to detect interest points using the determinant of Hessian it is first necessary to introduce the notion of a scale-space. A scale-space is a continuous function which can be used to find extrema across all possible scales as explained in A.Witkin (1983). As the processing time of the kernels used in SURF is size invariant, the scale-space can be created by applying kernels of increasing size to the original image. This allows for multiple layers of the scale-space pyramid to be processed simultaneously and negates the need to subsample the image hence providing performance increase.

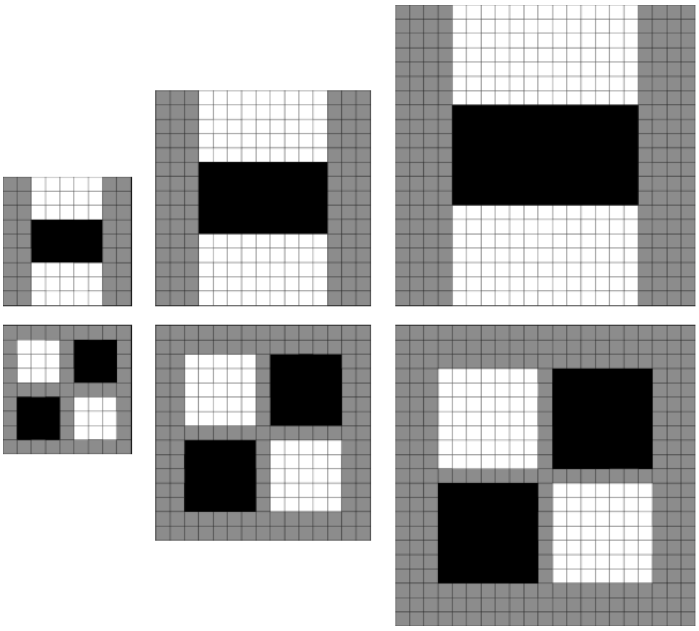


Figure 2.4: Filter Structure for Non Maximal supression

2.1.4 Interest Point Detection

After thresholding, a non-maximal suppression is performed to find a set of candidate points. To do this each pixel in the scale-space is compared to its 26 neighbours, comprised of the 8 points in the native scale and the 9 in each of the scales above and below. Figure 2.4 illustrates the non-maximal suppression step. At this stage we have a set of interest points with minimum strength determined by the threshold value and which are also local maxima/minima in the scale-space.

2.2 RANSAC Algorithm

Interest point detection is the major part of video stitching algorithm, but it is not complete without matching. The algorithm used for this is Random sample consensus, as described in A.Fischler and C.Bolles (1981). A hardware implementation of the algorithm can be seen in Lan-Rong Dung (2013).

The RANSAC algorithm can be applied to get the homography of each image pair. This is an iterative algorithm,to start four initial feature matches are selected in the random selection step of each iteration. In image matching there are multiple frames to be matched. In this case we have two frames, so the basic idea is to consider mappings between image pair include scaling + rotation and affine transformation, as most distortions present in natural-world images can be sufficiently accurately approximated by such mappings. Since perspective distortions are locally approximated well enough by affine functions, affine transformations are considered the distortion model for images matching.

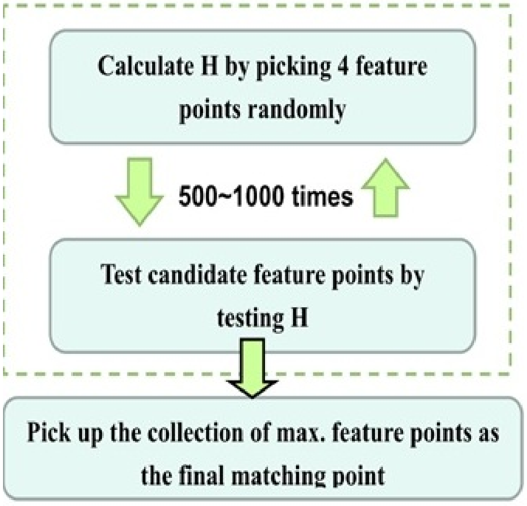

Calculate H by picking 4 feature points randomly
500~1000 times
Test candidate feature points by testing H
Pick up the collection of max. feature points as the final matching point

Figure 2.5: RANSAC Algorithm

As there is only 2D transformation involved the task is to find the transformation matrix. If (x, y) are the source image I coordinates, and (u, v) represent the destination image J coordinates, the affine transformation is represented by a homogeneous matrix H.

$$\begin{bmatrix} U \\ V \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, H = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

In the matrix shown above there are 9 parameters out of which 6 are unknowns. So, we take 3 interest points at random from each of the frames and find the corresponding H matrix. Considering 3 points $(x1, y1)$, $(x2, y2)$ and $(x3, y3)$ in the source image I, and their counter-parts $(u1, v1)$, $(u2, v2)$ and $(u3, v3)$ in the destination image J, we have to solve the following system of equations.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{2.6}$$

Instead of solving the big equation provided above, it can be divided into two sets of equations considering the computational complexity and saving hardware. The new reduced set of equations are provided below.

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{2.7}$$

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} d \\ e \\ f \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{2.8}$$

Now, the H matrix has been found out the task is to test the transformation matrix using other points. This process is repeated until we can find the H matrix with required accuracy. The final transformation matrix gives the matching points in both the frames. The Figure 2.5 shows the total block diagram of the algorithm.

2.3 Frame Stitching

After the image registration i.e., feature point matching is completed, we have the points that are similar in both the frames and we have the transformation matrix. For image stitching, the images have to be overlapped where the common portions exist. This can be interpreted in a different way considering the case of video stitching.

The freedom that exists for image stitching doesn't exist for video stitching. Image stitching has no real time constraints. So, any number of images can be stitched and the time required depends on how efficiently the algorithm has been implemented. In case of video stitching, the positioning of the cameras is very important. As the video stream is continuously coming in, there is only a limited time that a frame can be delayed as the buffer sizes cannot be infinitely large. Generally for any video stitching algorithm, the cameras are taken to be horizontally/vertically aligned enhancing the field of view of the output video. So, stitching in case of videos involves movement of a frame sideways to overlap the common portion in both the frames.

From the previous stage, the points that are matched in frames are available. The amount of frame displacement for proper stitching can be found out by calculating the euclidean distance between the matched coordinates. The output video frame can be obtained by moving one frame keeping the other frame fixed. An illustration of how this is achieved is described in the latter section.

CHAPTER 3

ARCHITECTURE FOR HARDWARE DESIGN

3.1 Architecture of Two Video Streams Display

The main part of video stitching algorithm is to combine two different video streams which have some common portion among them. To achieve this, it is important to display two videos on a display monitor. This allows us to see the overlap between two videos and decide whether the two videos can be stitched or not. The block diagram in Figure 3.1 shows how various blocks involved and this architecture can be used on any platform.

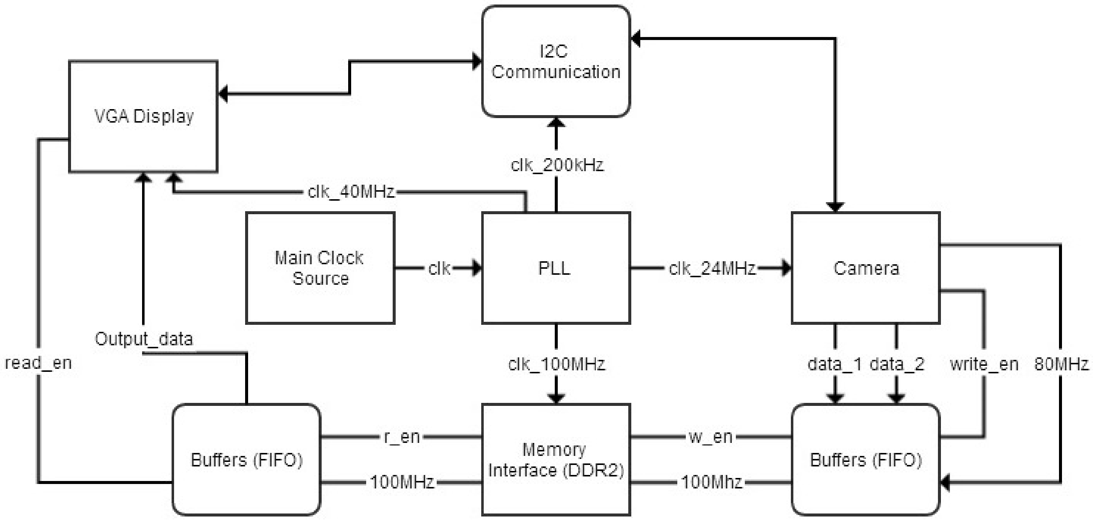


Figure 3.1: Basic Architecture for Implementation

The figure shows different blocks using different frequencies. The camera is configured to send pixel data at a rate of 80MHz. The camera data comes at 80MHz which does not match with the frequency of the data written to the VGA Display. This is not a problem because one pixel data is available only after two clock cycles which makes the frequency of camera effectively 40MHz which is equal to the frequency of the Display. For any platform, communication between various devices is essential and this is achieved by using I2C protocol. It is also necessary for a PLL to generate required clock frequencies for various modules involved. For the data to redirect from the camera to the display, there should be some buffers involved to avoid mismatch in reading and writing frequency. The memory device used in this case is a DDR2 external RAM which operated in burst mode and has a frequency of 100MHz. There are read enables as well as write enables to ensure the data is transferred correctly.

3.2 VLSI Architecture for Two video Display

This section illustrates how various modules are connected and how the data flows between various modules. The details of the hardware used to implement this architecture are described in the section below.

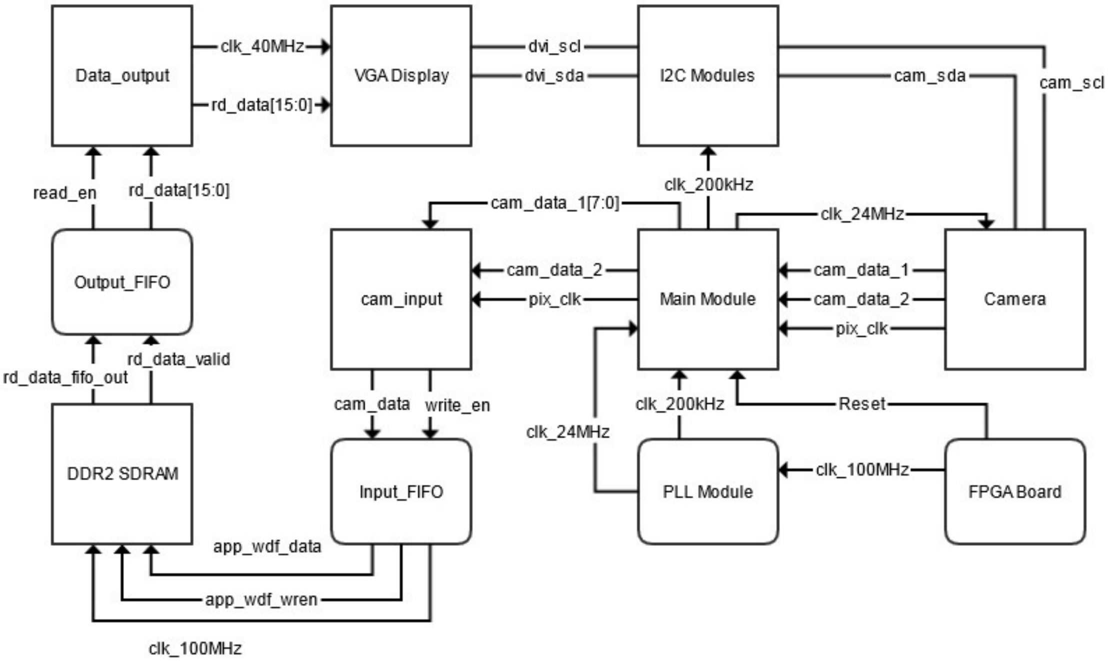


Figure 3.2: VLSI Architecture

All the important signals that connect between various modules are mentioned in the figure above. $pix - clk$ is the clock at which the cameras send the data. DDR2 uses a FIFO interface, which has some signals. $app - wdf - data$ is the port through which the actual data is sent. $app - wdf - wren$ is the write enable to the DDR memory. $rd - data - valid$ is a valid signal, which tells the data is valid to be read from the memory. $rd - data - fifo - out$ is the data output which is sent to the display module via a output FIFO. From the display module $rd - data$ is sent to VGA Display. The Input and Output FIFO also contain appropriate read and write enables which ensure proper flow of the data. $dvi - scl$ and $dvi - sda$ are the configuration details for the display. Similarly, $cam - scl$ and $cam - sda$ are the configuration details for the camera. The configuration of the camera and the display can be changed in their respective I2C modules as required.

3.3 Architecture for Hessian Filters

When a filter is applied to an image it means that the value of the pixel is impacted by the weights of the pixels surrounding it. How it is impacted depends on the type of the filter applied. So, for any image a pixel needs it's surrounding pixels for a filter to be applied i.e., it needs the pixels in the rows above it as well as the rows below it. It is not a problem when a filter is applied to an image because it is already stored in the memory and all the pixel values are present. It is a challenging task when we are dealing with real-time filtering. When a video stream is coming in continuously, one cannot buffer everything to calculate the filter value later, this will lead to buffer becoming full and the new data is lost. To deal with this we propose a process called row buffering.

Row buffering, as the name suggests we buffer rows of data for applying the filter. Buffering is not storing data, it means delay the data so that there is time to do calculations. The number of rows buffered depends on the size of the filter that we are going to apply. If we consider a 3×3 filter, the architecture designed can be seen in Figure 5.1.

I represents the video stream and the window has the filter function ready to apply when the third row starts. For a 3×3 filter we need first two rows to be buffered. Similarly, for a 9×9 filter we need 8 row buffers for implementation.

But the data flow management itself is not sufficient to apply a filter. It is also essential for the filter function to work to obtain the correct values for a filter. Detailed architecture of how the filter function is applied is show in the Figure 3.4. All the three filters are implemented using this architecture. The output of these filters are used to calculate the determinant. This is the step next to displaying two

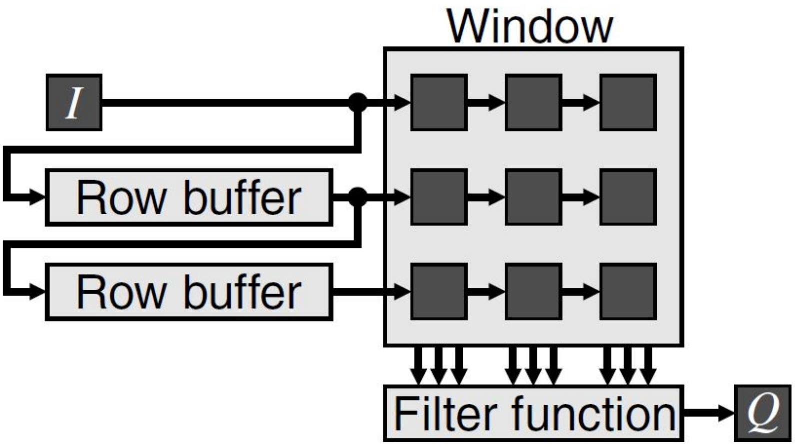


Figure 3.3: Architecture for Filter Implementation

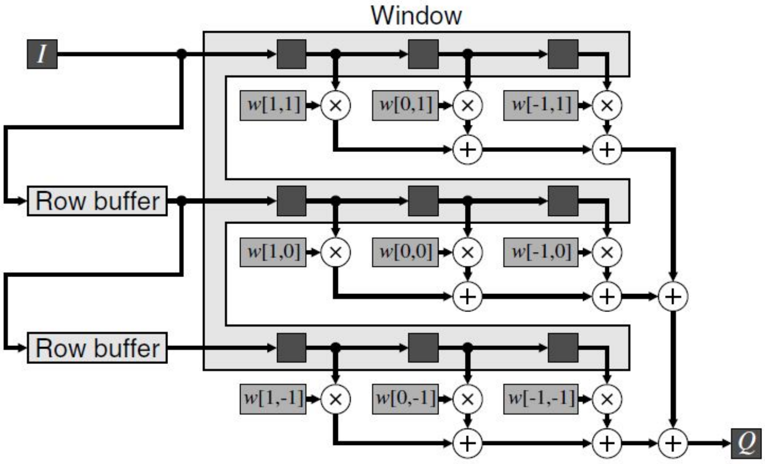


Figure 3.4: Implementation of Linear Filtering

video streams. So, this has to be applied to both video streams. After finding the filter values, determinant value has to be found out. The following section gives the architecture for finding the interest points based on the determinant values.

3.4 Architecture for Finding Interest Points

As the pixels come in, they are send into the hessian filter module to find the filter values accordingly and the determinant values are sent into the Interest point detection module. The point to be noted is that the hessian filter module will apply the filter as soon as the first pixel comes in, but for a filter to be applied on the pixel it needs the values that are not yet available from the camera. The correct filter values will be available from the fifth pixel in the fifth row as a 9×9 filter is being used. This is a border condition and considering the size of the image being used, the error in detecting the interest points towards the borders of the image can be ignored. This is very important when designing the hardware because it turns out that the resources used for eliminating the border conditions are more complex and undesirable.

At the end of Hessian filter module, the determinant values are send to the interest point module. In the interest point detection module, one maximum point is taken in every 9×9 block. For an image size of 800×600, this amounts to 5900 feature points, which is a huge number. To reduce the number of feature points detected, we choose the feature points that are much greater than the average of all the feature points detected.

$$I > k \times A \tag{3.1}$$

I is the determinant value of the feature point detected, A is the average of all the feature points and k is a constant value, which depends on the final number of interest points that we need to detect. The next step is to do scale space filtering for more accuracy, to do that hessian filters should be applied across larger scales, which compromises the space and time of the hardware. So, the feature points are detected on a single scale. The Figure 3.5 shows the block diagram of how Interest point module works.

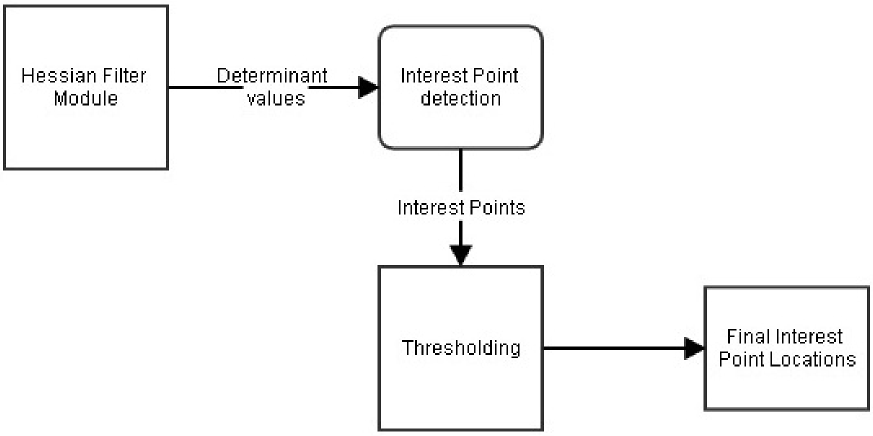


Figure 3.5: Interest Point detection Block Diagram

Detecting the interest points means, finding the location where the maximum value of the determinant exists. Considering software implementation detecting the maximum value in a block of values is easy. In hardware implementation it is hard because, values of the 9×9 block are not completely available. It can be seen that we have encountered a similar problem when calculating the filter values. So, a similar architecture has been designed which takes the maximum values continuously as the pixels come in.

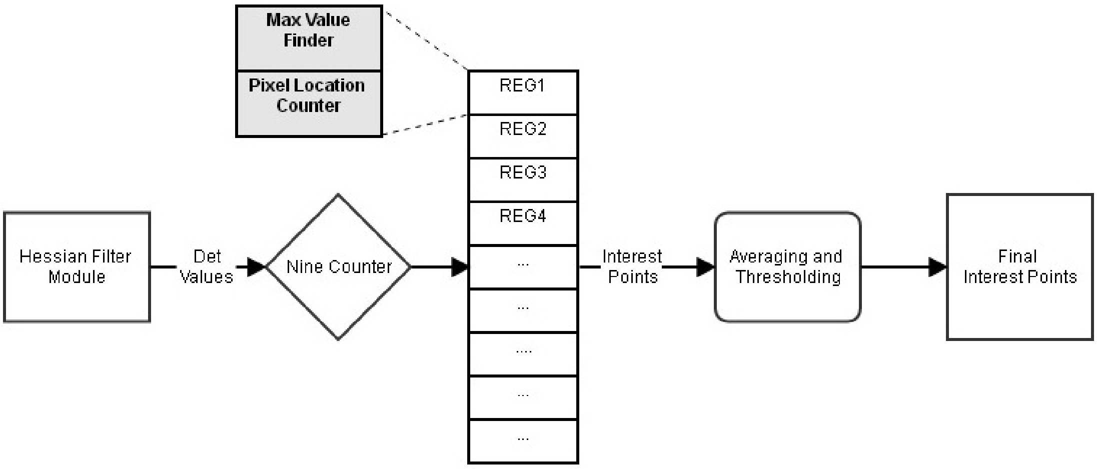


Figure 3.6: VLSI Architecture for IP detection

The Figure 3.6 shows the vlsi architecture for interest point detection. As the interest points are detected in a 9×9 block, there is a nine counter involved. The purpose of this counter is to store the maximum value among the 9 pixels in the registers. When the counter resets, the register count increases. The maximum value among the next nine determinant values is stored in the second register and so on. When the next row of pixels come in, the maximum value of the first nine determinant values in the second row is compared with the value stored in the first register and replaced accordingly. After the first 9 rows of pixels arrive, we have the first set of feature points. These are sent for averaging. After the feature point detection is complete, the average can be found out and depending on the number of feature points required we can change the threshold. A counter is kept to keep track of the pixel coordinates which will become the input to Interest point matching module.

CHAPTER 4

DETAILS OF THE UTILIZED HARDWARE

4.1 Proposed Approach

The project is targeted on a LX50T Virtex-5 FPGA Board. The camera input is taken from two CMOS cameras on a VmodCAM and it is connected to the FPGA via a VHDC connector. The algorithm designed is applied on the incoming video stream and is sent to the DVI transmitter to display on a VGA monitor. The block diagram of the project is shown in Figure 3.2. The communication between the camera and the FPGA is done by using the I^2C protocol. The same applies for the communication between the Display and the target board.

4.2 Overview of the Hardware

The following peripherals of Virtex-5 Genesys FPGA board are exploited for this application

1. 100MHz onboard clock
2. VHDC (Very High Density Connector) for camera interface
3. HDMI out (High Definition Multimedia Interface) connector for display
4. Chrontel 7301 DVI(Digital Visual Interface) transmitter
5. Programming port and power supply
6. LCD, switches and LED for testing purpose
7. DDR2 SDRAM (Dual Data Rate Synchronous Dynamic RAM)

4.3 External Systems

4.3.1 Camera details

1. Resolution : 800 x 600
2. Frame rate: 30 FPS
3. Pixel rate: 80MHz
4. Data out : 8-bit parallel, RGB 565 format
5. Interface : I2C

4.3.2 Display Details

1. VGA Display
2. Resolution: 800 x 600

4.4 FPGA Details

The following modules have been instantiated in the Virtex-5 FPGA for implementing the algorithm

1. Clock generator module
2. I/O Interface module
   (a) Camera input module
   (b) Camera I2C interface module
   (c) DVI output module
   (d) DVI I2C interface module
3. Input output FIFO modules (Data and Address FIFOs)
4. DDR Controller module (Memory Interface Generater(MIG)- Xilinx IPCore) for interfacing DDR2 SDRAM

4.5 Clock Generation Details

The Genesys board has an onboard clock of 100MHz. This is the master/main clock(MCLK). This clock is buffered and input to the clock generation module. The following clocks are generated from this module

1. 24MHz is sent to the camera
2. 80MHz for the DVI output module. 40MHz derived from 80MHz is sent to the DVI transmitter
3. 200KHz for the DVI camera I2C module. The 200KHz is internally generated from a 24MHz clock signal
4. 100MHz (for DDR2 data storage)

4.6 Input Interface

For the video stitching project we need two cameras that are available on the VmodCAM. The configuration of both the cameras are set to be the same and the details are below. The cameras chosen for implementation provides output with a resolution of 800X600 at a frame rate of 15fps. For implementation, the camera is set to a resolution of 800x600. The cameras communicates with FPGA through I2C (Inter IC communication). The cameraiic module generates the control signals for the I2C interface. The cameras initial settings and the control signal for interface is send to the cameras from the cameraiic module. A clock of 24MHz generated from the main system clock of 100MHz is inputted to the cameras. The cameras output three main signals.

1. A pixel clock of 80MHz, which is generated internally by the cameras from the 24MHz input clock.
2. Data out(8 bit): Each pixel is defined by 16 bit RGB565 format. Two consecutive data outs (of 8 bit each) gives one pixel information.
3. Enable signals: The enable line consists of line enable and frame enable signals. The data output from the cameras is valid only when both the line enable and frame enable signals are high.

4.7 DDR2 Interface

Considering the memory constraints imposed by hardware when using the blockRAM, the external DDR2 memory available on the board is also used. DDR2 has 256MB of memory and it can be used only for storage and it has no parallel processing and computational power unlike blockRAM. To initialize DDR2, IPCore from inside ISE has to be invoked. A controller module is present to properly transfer the data from the cameras to the display. For starters, as described in the section above the data is taken from the camera and display directly via the DDR memory. This is done to understand how the memory accessing procedures work in DDR memory.

4.8 Output Interface

For sending the video information DVI controller or transmitter is required. Genesys board is provided with a DVI transmitter (Chrontel 7301 IC) external to the FPGA. The DVI transmitter drives the HDMI output through I2C interface. Similar to the camera initialization, the display is also to be initialized for its internal settings. Hence, the display initialization is done in the $init - 7301iic$ module. The idea is to take the stitching video frames and then display it on the monitor. The video display is only 800×600 and both the cameras input 800×600 images. So, the video streams from each camera is cropped to 400×600. These cropped frames are stitched and then sent to the display module.

Along with the output data, 40MHz differential clock (derived from 80MHz signal), reset, line enable, frame enable and DVI enable signals are sent to the DVI transmitter. Since the colour pixel has a depth of 24 bit, the 24-bit is divided into two 12 bit signal and is sent out during both the positive and negative clock edges for delivering a 24 bit information in single clock pulse.

CHAPTER 5

IMPLEMENTATION DETAILS

As discussed in literature the first part of SURF i.e., the Integral Image calculation is skipped in this algorithm. The reason behind skipping the Integral image calculation is that it is a trade off to use bigger images. When we use an image of size 800×600, the block memory available on the FPGA is not sufficient to store the complete integral image. And two video streams are simultaneously coming in, this implies we need to store two integral images each of 480000 pixels. From WeiLong (2013), it can be seen that approximately 21-bits are sufficient to store the integral image of each pixel. This amounts to 19.22 Mbits, but the block memory available on the FPGA is only 1.7 Mbits. So, the external DDR2 memory that is present on the FPGA board is effectively used in storing the video stream and sending it out to display on the monitor.

5.1 Two Video Streams Side by Side Display

Displaying two video streams simultaneously using DDR memory is important. It is the main part of video stitching as without displaying the two video streams, there is no possiblility of applying the video stitching algorithm. The procedure to do this is as follows.

1. The main clock on the FPGA board is 100MHz. This clock is sent into the PLL (Phase Locked Loop) and three clocks are generated 24MHz, 200kHz and 40MHz.

2. The 200kHz clock is needed to configure the I^2C communication with the Camera as well as the VGA display with the FPGA board. As two cameras are being used two $cam - iic$ modules have to be instantiated.

3. The 24MHz clock is sent to the external camera which is attached to the FPGA board via a VHDC connector. The camera uses this clock to generate a 80MHz clock and this is the rate at which it sends the pixels in.

4. The two cameras available on the VmodCAM are activated and they start sending data at 80MHz and 8-bits for a single clock cycle. The camera is configured

such that each pixel data is 16-bits, this means pixel data from both cameras come in at 40MHz.
5. When a pixels come in, it is sent to the the $cam - input$ module and this is where they are converted to grayscale. Grayscale is used for processing purposes when we get to applying the filter but for now colour video stream is used.
6. From both cameras we extract $pix - cnt$, as the name suggests it gives the number of the pixel that is coming in.
7. Using this $pix - cnt$ we can choose to store either the first camera data or the second camera data depending upon region in which the stream is displayed on the monitor.
8. The DDR uses 128-bit write in a single clock cycle. To do this we have created a buffer which becomes full when 8 pixels come in (each pixel is of 16-bit length) and then writes into DDR.
9. So, we need a controller module for the DDR memory to properly control when the signals need to be enabled for reading and writing.
10. Now that both streams are in the DDR memory, we have to display them on the monitor. From DDR pixels are extracted and sent into the $d - output$ module. From the DDR memory 16-bit output is extracted.
11. Code is written in the $d - output$ module as to how the pixels are sent to the monitor for displaying. The results obtained from implementation are illustrated in the next section.

5.2 Implementation of Hessian filters

An new module is added to the modules that are present for implementing the Hessian filters. First, the filters will be applied to single frame and then are extended to second frame. The architecture for implementing the filters is explained in the sections above. As the filter used is a 9×9 filter, we need 8 row buffers. These row buffers are realized using $fifos$.

The data from the camera comes in at 80MHz, 8-bit per clock cycle. As we are using RGB-565 format, two clock cycles give one pixel value. The pixel value is converted to grayscale as it is coming in and sent into the first fifo buffer as shown in the figure below

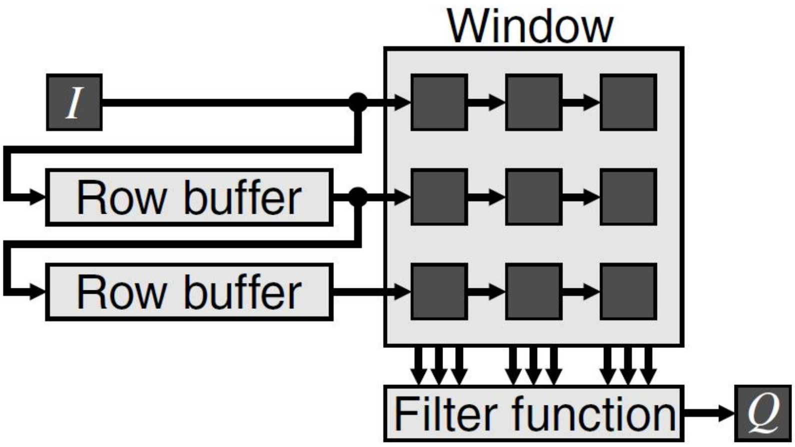


Figure 5.1: Architecture for Filter Implementation

The row width of the image is 800 pixels, so the depth of each fifo should be 800. This is called stream processing. The entire module operates on a 40MHz clock as it is the frequency at which the grayscale values are created. All the details that are given in the above section regarding the frequencies and hardware connections remain the same in case of applying hessian filters. As the pixels come in the filter values are calculated and the filter values are stored in a register which is redirected to DDR memory to be sent into the display module. As it is already specified the DDR memory operates at a frequency of 100MHz.
The architecture for hessian filter module is provided in the above section. The registers used for redirecting the filter values to DDR memory have a data width of at least 16-bit. DDR has a large amount of memory and the complete data can be stored in it. But, when the output from DDR module is sent to VGA display the data width is not sufficient for displaying the entire 16-bits. The VGA display uses 24-bit colour. As grayscale is being used, truncated 8-bits can only be used to display the filter values on the monitor. The snapshots of results obtained from hardware implementation are shown in the Figures 6.4, 6.5, 6.6.

5.3 Implementation of Interest Point Detection

The output from the filter values are sent to the IP detection module. The vlsi architecture for feature detection is provided in the section above. As we need to calculate the maximum in a 9×9 block, registers are provided depending on the width of the image. The width of the image is 800, so the number of registers that need to be used is approximately 89. The register count is increased by 1 for every nine pixels that come in. The value that is stored in the register is the maximum value of the nine pixels. When the 800 pixels arrive, the register counter again points to the first register. The values from the second row are compared with the value stored in the register and the value in the register is replaced accordingly. Only after 7209 clock cycles i.e., 9 rows and 9 columns, the first maximum value is obtained and is send to the averaging and thresholding part. For, thresholding the average of the determinant values of all the feature points have to be present. For this to happen, it should wait until the camera inputs the complete frame. To do this we have to store the data in a blockRAM. Instead, the average value can be found out for the first frame and this can be used for thresholding other frames. This is attributing to the fact that there is no rapid moving objects in the scene which allows us to save some memory. After thresholding, the interest points are detected and the comparision of the results obtained is shown in Figure 6.9.

CHAPTER 6

IMPLEMENTATION RESULTS

The test setup used for implementing the algorithm is shown in the figure below.

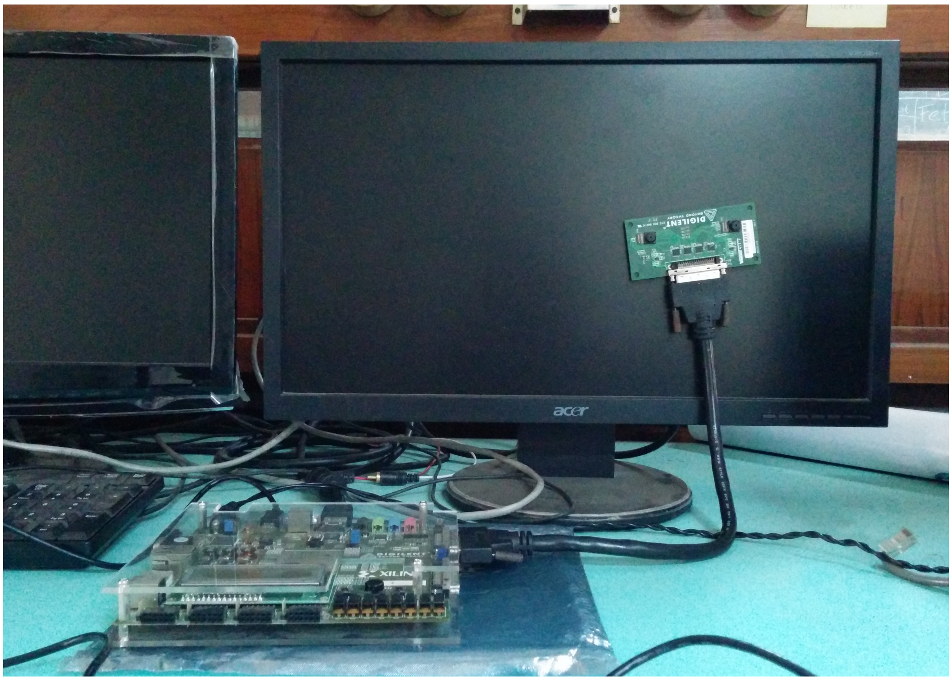


Figure 6.1: Test Setup for Implementation

6.1 DDR2 Two Video stream Display

6.1.1 Results

The most important part of a video stitching algorithm is to detect the similar parts in both images and to combine them appropriately. When there is no similarity between two video streams, there is no point is doing video stitching. First result Figure 6.2 shows a snapshot of the video stream which has no overlap between the left hand side and the right hand side. This is not suitable for video stitching. So, the camera data is reconfigured to obtain overlap making it suitable for stitching. Figure 6.3 shows the overlap between both the video streams.

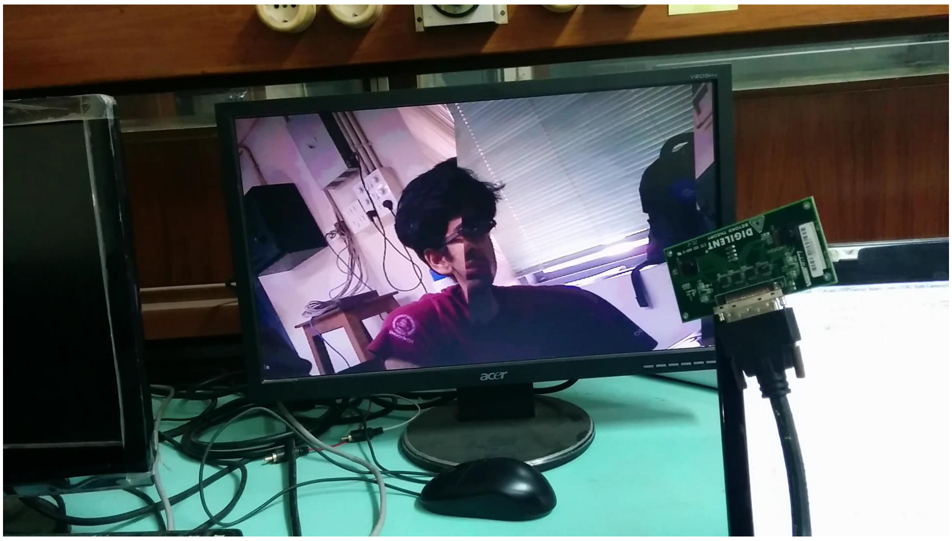


Figure 6.2: Two cameras display without Overlap

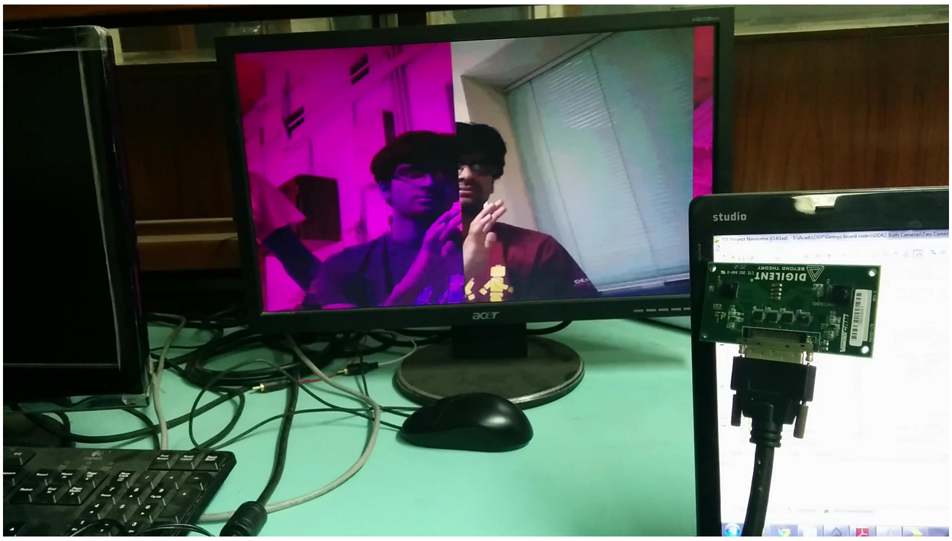


Figure 6.3: Two cameras display with Overlap

6.1.2 Synthesis Reports

The program is synthesized using Xilinx 13.2 ISE and results are obtained. For this part of the algorithm the utilization summary cannot be calculated independently as the modules don't have any meaning without support of other modules. So, the total device utilization is shown below. The device used for implementing this is Virtex-5 LX50T-ff1136-3. Three tables showing Logic utilization, logic distribution and Special feature utilization are given below.

Logic Distribution	Used	Available	Utilization Percentage
Number with Unused Flipflop	1418	4780	29
Number with an unused LUT	2094	4780	43
Number of fully used LUT-FF pairs	1268	4780	26
Number of unique control sets	423		

Table 6.2: Logic Distribution Summary

Special Feature Utilization	Used	Available	Utilization Percentage
Number of Block RAM/FIFO	7	60	11
Number of BUFG/BUFG CTRL	14	32	43
Number of PLL-ADV	2	6	33

Table 6.3: Special Feature Utilization Summary

Logic Utilization	Used	Available	Utilization Percentage
Number of Slice Registers	3362	28800	11
Number of Slice LUTs	2686	28800	9
Number used as Logic	2663	28800	9
Number used as Memory	23	7680	0

Table 6.1: Logic Utilization summary

6.2 Hessian Filters

6.2.1 Results

The filters are applied on the incoming video stream and is displayed on the monitor. Also a MATLAB version of filter implementation is used for checking the results obtained from hardware implementation. The figures below are a snapshot of filtered video streams that are being displayed on 800×600 VGA Display.

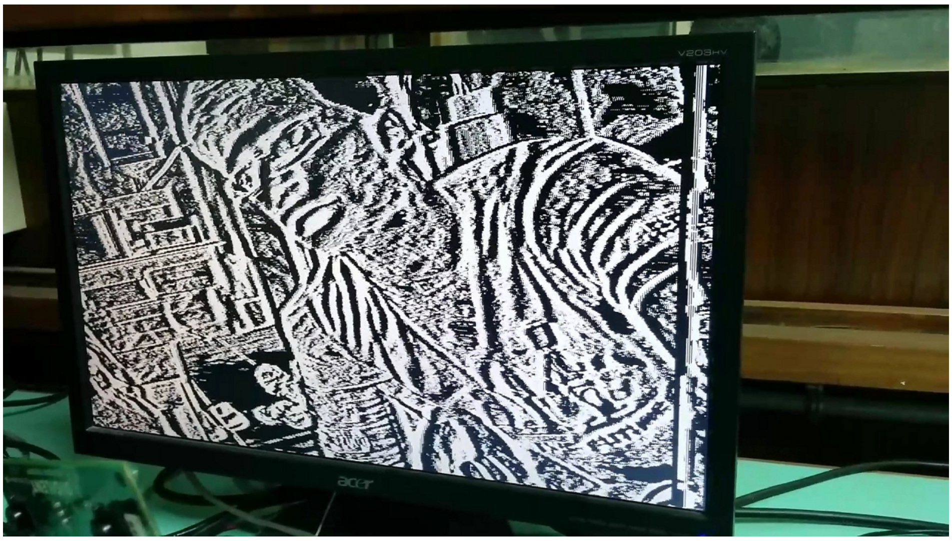


Figure 6.4: Dxx Filter

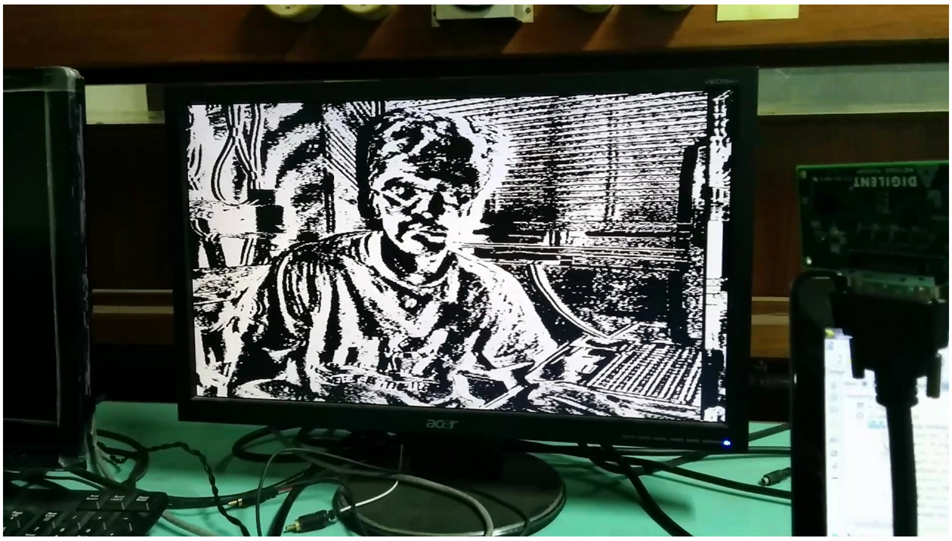


Figure 6.5: Dyy Filter

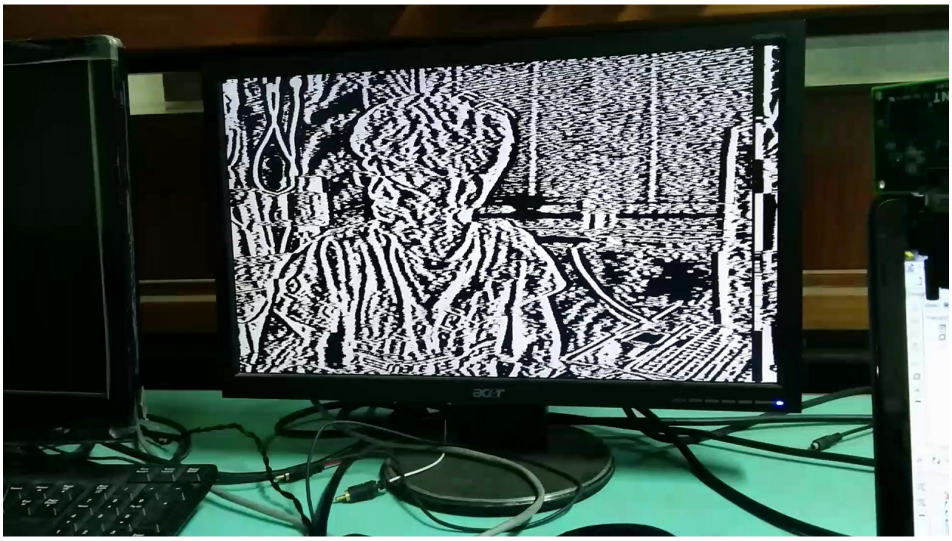


Figure 6.6: Dxy Filter

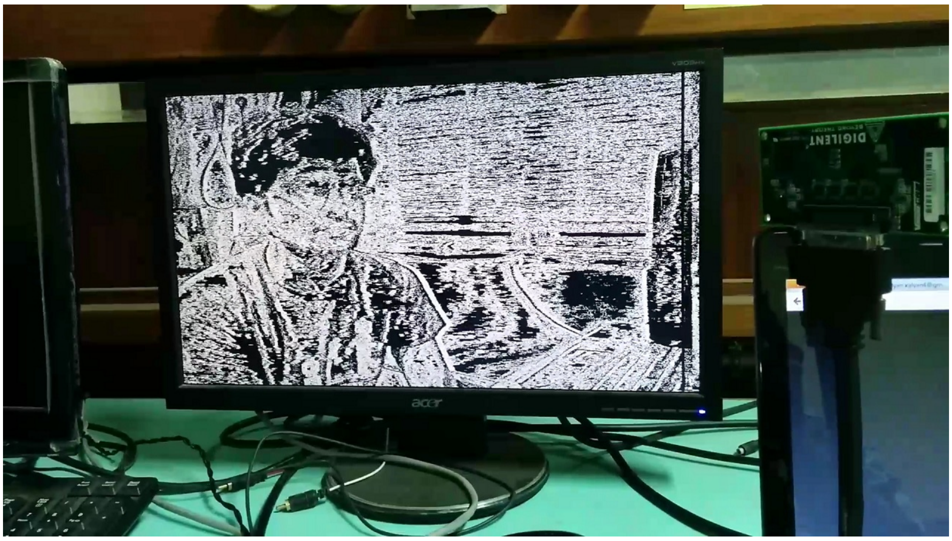
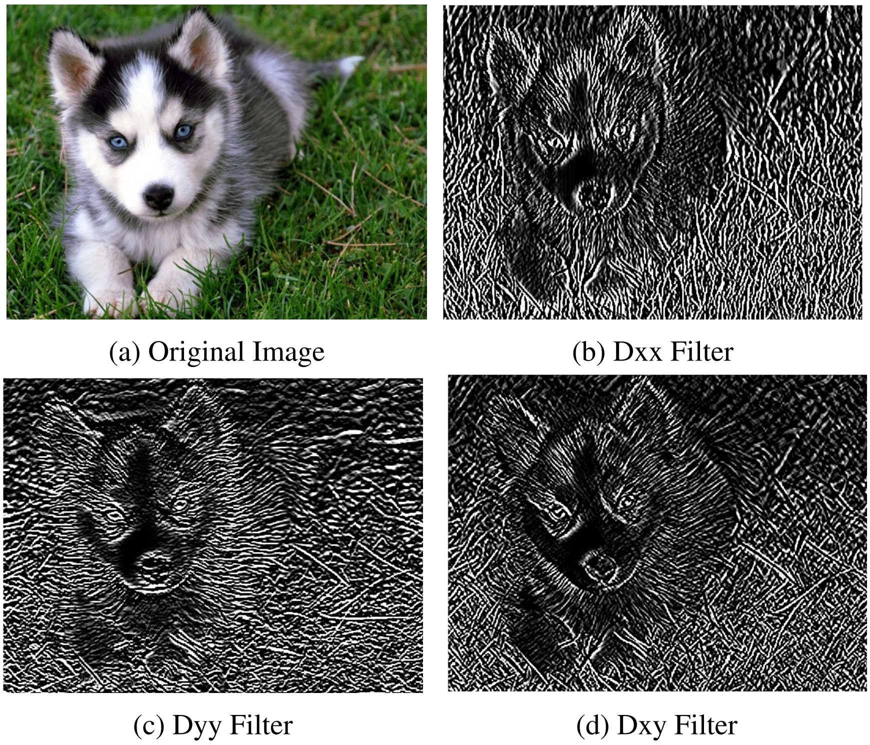


Figure 6.7: Determinant Values

The Figures give information about the filters that are applied to the video stream. The Figures 6.4, 6.5, 6.6 are not accurate when compared to the MATLAB version of the filters. This happens because the VGA display accepts 24-bit colour input. In case of gray scale it is only 8-bits, but the filter output values need at least 16-bits. So, the picture shows only the truncated filter values. This is same in case of Figure 6.7 as the determinant value needs 21-bits for storage.



(a) Original Image

(b) Dxx Filter

(c) Dyy Filter

(d) Dxy Filter

Figure 6.8: Hessian Filters in MATLAB

Now coming to the MATLAB implementation of filters, they are more accurate than the ones shown in the VGA display. For simulation purposes an image is used in MATLAB. The Figure 6.8 shows the implementation of filters in software.

6.2.2 Synthesis Reports

For implementation purposes, only data from single camera is used. The tables below show Logic utilization, logic distribution and special feature utilization.

Logic Utilization	Used	Available	Utilization Percentage
Number of Slice Registers	4541	28800	15
Number of Slice LUTs	5643	28800	19
Number used as Logic	5588	28800	19
Number used as Memory	55	7680	0

Table 6.4: Logic Utilization summary

Logic Distribution	Used	Available	Utilization Percentage
Number with Unused Flipflop	3946	8487	46
Number with an unused LUT	2844	8487	33
Number of fully used LUT-FF pairs	1697	8487	19
Number of unique control sets	524		

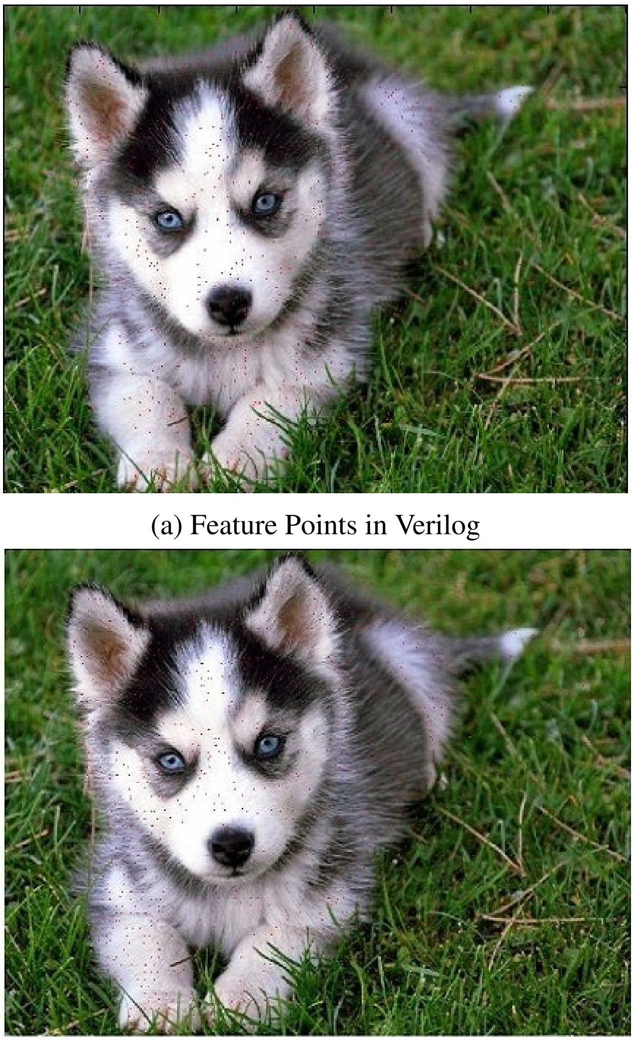
Table 6.5: Logic Distribution Summary

Special Feature Utilization	Used	Available	Utilization Percentage
Number of Block RAM/FIFO	11	60	18
Number of BUFG/BUFG CTRL	14	32	43
Number of PLL-ADV	2	6	33

Table 6.6: Special Feature Utilization Summary

6.3 Interest Point Detection

Interest point detection in software could not be done with a real time video that is being recorded in the cameras. So, to check how well the interest points in Verilog match with the MATLAB results, an image is used. The image is sent into the verilog module and the points found are plotted on the image. On the other hand same algorithm is implemented in MATLAB and the points are plotted on the image.



(a) Feature Points in Verilog

(b) Feature Points in MATLAB

Figure 6.9: Comparision of Hardware and Software Implementation

In Figure 6.9 the points in red are the detected interest points. As the picture suggests, the hardware and software implementation are a good match.

6.4 Video Stitching

6.4.1 Video Stitching with Static Compensation

This part of video stitching does not involve matching the frames in order to obtain the distance that the frame is required to move. So, a static movement is given to the frame and the results are obtained. The problem with this is it does not adjust with the depth.

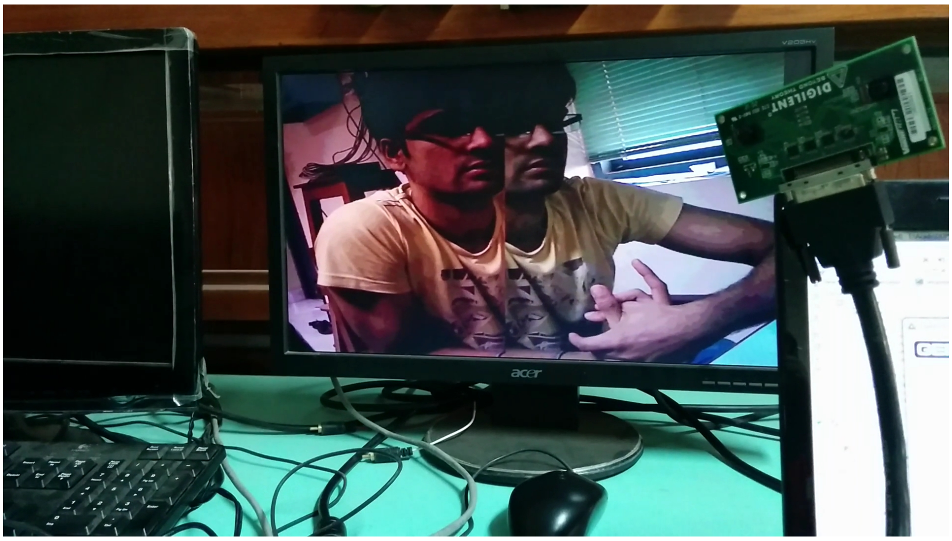


Figure 6.10: Frames without Stitching

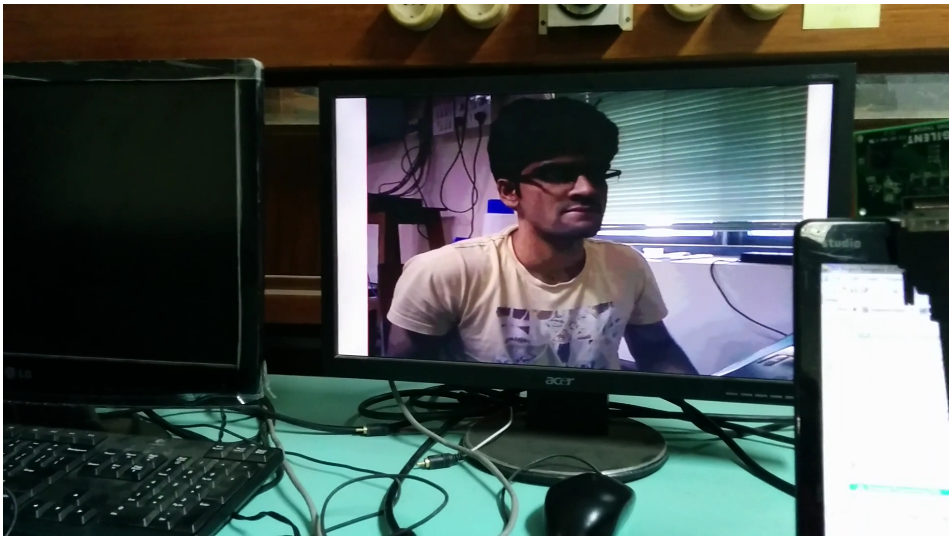


Figure 6.11: Frames with static compensation

Figure 6.10 shows the two frames side by side without compensation. The result of static compensation can be seen in the Figure 6.11. When the frame is moved, the common portions of the two videos overlap decreasing the width of the output image. The white patches on the side of the display are a result of this.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

An efficient hardware architecture for two video streams display and video stitching was designed. A verilog code was developed, synthesized in Xilinx ISE version 13.2 and tested on Gensys Virtex 5 FPGA board. The architecture was implemented and tested for an input resolution of 800×600. The following are the sequence of steps involved in implementing the video stitching algorithm.

1. Studied the requirement of the project and carried out literature survey and finalized the steps involved in video stitching algorithm based on hardware implementation requirement.
2. Estimated the amount of hardware required as the size of the image used is large and the algorithms are modified and designed according to the requirements.
3. An efficient architecture for single video stream display was designed and implemented using Xilinx 13.2 on a Virtex-5 FPGA. This is the hardware utilized for the complete project.
4. For video stitching, two video stream display is necessary. So, architecture for two video stream display is designed, implemented and the results are provided in the section above.
5. Hardware architecture for SURF algorithm was designed, implemented and the detected interest points. Hessian filters are implemented and the results are compared to software implementation.
6. Interest points are detected for an image using Verilog and the results are tested with MATLAB.
7. Extending the two video stream display, stitching with static compensation is achieved in real time.

CHAPTER 8

APPENDIX

8.1 Chipscope Pro for debugging the Hardware

This section describes the Xilinx ChipScope Analyzer. ChipScope is a set of tools made by Xilinx that allows us to easily probe the internal signals of design inside an FPGA. While the design is running on the FPGA, we can trigger when certain events take place and view any of the design's internal signals. Because the ChipScope analyzer logic is implemented in the FPGA, it has some important limitations. The sample memory of the analyzer is limited by the memory resources of the FPGA. In a design that uses much of the FPGA's memory, there may not be much memory left over for the ChipScope cores. Also, ChipScope cannot sample as quickly as an external logic analyzer. Generally, ChipScope sampling rate will be the same as the design's clock frequency. It is therefore not possible to detect glitches with ChipScope. In order to use the ChipScope internal logic analyzer in an existing design project, first the ChipScope core modules should be generated, which perform the trigger and waveform capturing functionality on the FPGA. Afterwards, these cores are instantiated in the Verilog code connected to the signals to be monitored. The complete design is then recompiled. Instead of loading the resulting .bit file onto the FPGA using iMAPCT, the ChipScope Analyzer application is used to configure the FPGA. ChipScope Analyzer also provides the interface for setting the trigger criteria for the ChipScope cores, and for displaying the waveforms recorded by those cores.

8.2 Procedure for invoking IPcore

1. In the design window Right click on the project top module.
2. Click Newsource.
3. Click on IP(coregenerator and ArchitectureWizard) in the newsourcewizard pop up window.

4. Give a file name, select addtoproject and click next
5. Search for the required IPcore and select from the drop down menu. The IPcore can be anything, blockRAMs, fifos, DDR memory. In this project we have used blockRAMs, fifos, DDR MIG(memory interface generator), PLL(Phase locked loop).
6. Click next and finish.
7. Wait for the IPcore window to pop up
8. Download the datasheet available in the left bottom of the IPcore window.
9. Read the datasheet for selecting other features of the core for configuring the IPcore as per the specific requirement of the user. This depends upon the IPcore that is invoked.
10. After selecting all the features click generate.
11. A .v file and .XCO file is generated in the folder and the IPcore is added to the design window which indicates that IPcore is added to the project.

REFERENCES

1. **A.Fischler, M.** and **R. C.Bolles** (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 381–395.

2. **A.Witkin** (1983). Scale-space filtering. *Int. Joint conf. Artif. Intelligence*, 2:1019.

3. **C.Harris** and **M.Stephens** (1988). A combined corner and edge detector. *Proc. Alvey Vision Conf*, 147–151.

4. **D.Lowe** (2004). Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110.

5. **Evans.C** (2009). Notes on the opensurf library. *Technical Report CSTR–09–001, University of Bristol.*

6. **H.Bay, T.** and **L.Gool** (2006). Surf-speeded up robust features. *ECCV*, 1:404–417.

7. **Lan-Rong Dung, Y.-Y. W., Chang-Min Huang** (2013). Implementation of ransac algorithm for feature-based image registration. *Journal of Computer and Communications*, 46–50.

8. **Viola, P.** and **M. Jones** (2001). Rapid object detection using a boosted cascade of simple features. *CVPR(1)*, 511–518.

9. **WeiLong, Z.** (2013). An efficient vlsi architecture of surf extraction for high resolution and high frame rate video. *SCIS*, 1–14.