FPGA IMPLEMENTATION OF REAL TIME DIGITAL VIDEO STABILIZATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

İSMAİL ÖZSARAÇ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

FEBRUARY 2011

Approval of the thesis:

FPGA IMPLEMENTATION OF REAL TIME DIGITAL VIDEO STABILIZATION

submitted by **İSMAİL ÖZSARAÇ** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. İsmet Erkmen _____
Head of Department, **Electrical and Electronics Engineering**

Assist. Prof. Dr. İlkay ULUSOY _____
Supervisor, **Electrical and Electronics Eng. Dept., METU**

Examining Committee Members:

Assoc. Prof. Dr. Tolga ÇİLOĞLU _____
Electrical and Electronics Engineering Dept., METU

Assist. Prof. Dr. İlkay ULUSOY _____
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Cüneyt BAZLAMAÇCI _____
Electrical and Electronics Engineering Dept., METU

Assist. Prof. Dr. Şenan Ece GÜRAN SCHMIDT _____
Electrical and Electronics Engineering Dept., METU

Dr. Erkan YAVUZ _____
Image Processing Department, ASELSAN, MGEO

Date: **February 10, 2011**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : İsmail ÖZSARAÇ

Signature :

ABSTRACT

FPGA IMPLEMENTATION OF REAL TIME DIGITAL VIDEO STABILIZATION

Özsaraç, İsmail

M.Sc., Department of Electrical and Electronics Engineering

Supervisor : Assist. Prof. Dr. İlkay Ulusoy

February 2011, 95 pages

Video stabilization methods are classified as mechanical and digital. Mechanical methods are based on motion sensors. Digital methods are computer programs and classified into two as time domain and frequency domain based on the signal processing methods used for the motion analysis. Although, mechanical methods have good real time stabilization performance, they are not suitable for small platforms such as mobile robots. On the other hand, digital video stabilization methods are easy to implement on various hardware, however, they require high computational load and long processing time.

Two different digital video stabilization methods, one frequency and one time domain algorithms, are implemented on FPGA to realize their real time performances. Also, the methods are implemented and tested in MATLAB. FPGA results are compared with MATLAB's to see the accuracy performance.

The input video format is PAL of which frame period is 40ms. The FPGA implementation is capable of producing new stabilization data at every PAL frame which allows the implementation to be classified as real time. Also, the simulation and hardware tests show that FPGA implementation can reach the MATLAB accuracy performance.

Keywords: Real Time Digital Video Stabilization, Phase Correlation, Full Search, FPGA

ÖZ

GERÇEK ZAMANLI SAYISAL VİDEO SABİTLEME'NİN FPGA UYGULAMASI

Özsaraç, İsmail
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Yrd. Doç. Dr. İlkay Ulusoy

Şubat 2011, 95 sayfa

Video sabitleme yöntemleri mekanik ve sayısal olarak sınıflandırılır.Mekanik yöntemler hareket algılayıcılar üzerine kuruludur. Sayısal yöntemler bilgisayar programlarıdır ve hareket analizi için kullandıkları sinyal işleme yöntemine göre zaman bölgesi ve frekans bölgesi olarak iki sınıfa ayrılırlar. Mekanik yöntemler iyi gerçek zaman sabitleme performansına sahip olmalarına rağmen, hareketli robotlar gibi küçük platformlar için uygun değillerdir. Diğer taraftan, sayısal video sabitleme yöntemlerini değişik donanımlarda uygulanmak kolaydır, ancak bunlar yüksek hesaplama yüküne ve uzun işlem zamanına ihtiyaç duyarlar.

İki farklı sayısal video sabitleme yöntemi, biri frekans diğeri zaman bölgesi algoritmaları, gerçek zaman performanslarını anlamak için FPGA üzerinde uygulanır. Ayrıca, yöntemler MATLAB'ta uygulanır ve test edilir. FPGA sonuçları MATLAB'ınkiler ile doğruluk performansının görülmesi için karşılaştırılır.

Giriş video yapısı çerçeve süresi 40 ms olan PAL'dır. FPGA uygulaması her PAL çerçevesinde yeni bir sabitleme verisi üretme yeteneğine sahiptir, bu da uygulamann gerçek zamanlı olarak nitelendirilmesine olanak sağlamaktadır. Ayrıca, benzetim ve donanım test sonuçları FPGA uygulamasının MATLAB doğruluk performansına ulaşabildiğini göstermektedir.

Anahtar Kelimeler: Gerçek Zamanlı Sayısal Video Sabitleme, Faz İlintisi, Tam Arama, FPGA

To My Wife,

To My Family…

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere thanks to my supervisor Assist. Prof. Dr. İlkay ULUSOY, for her support, friendly attitude and encouragement at each stage of this thesis study.

I would like to thank to ASELSAN, MGEO Electronics Design and Image Processing Departments for the support given throughout this study.

I would like to forward my appreciation to all colleagues for their continuous encouragement.

I would like to present my thanks to Örsan AYTEKİN for sharing his knowledge.

I would like to thanks to my parents, Hüseyin ÖZSARAÇ and Gülbeyaz ÖZSARAÇ and my sister Fatma ÇAKIROĞLU and my brother M. Sefa ÖZSARAÇ for their support and unlimited love.

Lastly, special thanks to my wife, Havva ÖZSARAÇ for all her support, guidance, sharing her knowledge and help in implementation and for showing great patience during my thesis.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Video processing is widely used in many areas such as health, city planning, auto industry, space and military where accurate image frames are required. For instance, in a surgical operation where cameras are used, the operator needs real time video which is stable to understand the correct location of the problem. In a military system where object tracking is used, consecutive frames should be stable in the spatial domain, so that tracking algorithm can work properly.

The first and necessary step of all video processing algorithms is to remove the undesired global movements which can be in translational and rotational formats [1]. The characteristics of these undesired motions are related with the platform where the video source is located. In the avionic platforms all the movements described above can be occurred, however in ground applications where the motions are in spatial directions, the translational movements are most commonly encountered [2]. These undesired motions can be removed by different video stabilization methods.

There are mainly two approaches to stabilize the video frames, which are mechanical and digital methods [3]. Mechanical methods use sensors (gyros and accelerometer) to detect the motion and realize the stabilization by changing the location of the camera against the direction of the movement. Although this approach is very successful in stabilization and used in many platforms, its complex and huge structure is not suitable for small robots in laboratory or handy cams.

On the other hand, digital video stabilization can be used at every platform where hardware and software can be implemented.

Digital video stabilization methods have three implementation steps; motion estimation, motion correction and frame correction. Motion estimation is the most important and time consuming part. The motion estimation algorithms are mainly grouped into two according to the information that they use; frequency and time (spatial) domain [4]. Frequency domain algorithms are less sensitive to local motions but their computational load is high, so frequency domain approaches are not preferable for real time applications. There are several spatial domain approaches which are generally classified as block based and feature based [3] which have different computation load and calculation accuracies.

In this study, two different digital motion estimation approaches which are frequency and spatial domain will be implemented in FPGA to see their real time performance. The results of the methods will be compared according to computation time, accuracy, and logic usage and power consumption.

Real time implementation requires a strict pipelining of the video processing applications. Every task in the application list should be finished in a certain time [5]. Since video stabilization is the first step of the whole video path, it should also be completed according to real time constraints. In frequency domain implementation, the sub-block method is used to reduce computation time and hardware requirements. Because whole frame approach needs high speed external memories and FPGAs which are not available on all platforms. In spatial domain, Full Search (FS) algorithm is used which provides the most accurate results among the other spatial approaches with its high computational load [2]. The computational load of the FS algorithm can be handled with parallel structure of the FPGA.

This thesis study provides valuable comparison results about the performance of the two different digital motion estimation approaches. These results are obtained by

implementing the algorithms in a real time video stabilization system which is located on a mobile robot.

This thesis includes five chapters. First Chapter covers the introduction, problem definition and the main goal of the study. The Second Chapter summarizes the theoretical background of the proposed video stabilization algorithms. In the Third Chapter, the elements of the real time video processing environment are mentioned. Also, in this chapter the FPGA implementation steps of the algorithms are given. The implementation and comparison results are provided in Chapter Four. Finally, in Chapter Five the conclusion and future work of this study are given.

CHAPTER 2

DIGITAL VIDEO STABILIZATION METHODS

Digital video stabilization methods have three main parts. These methods start with motion estimation, then the estimated motions are evaluated according to some constraints and finally the shift on the frame is removed [7]. The stabilization flow is shown in Figure 1.

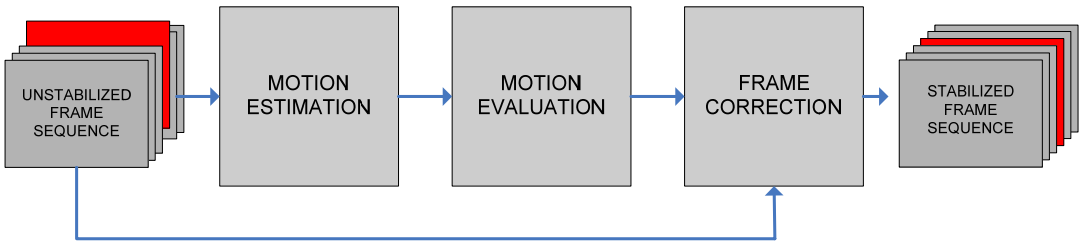

Figure 1: General Digital Video Stabilization Flow

Motion estimation is the most critical and time consuming part of the flow [8]. The results of this part determine the whole accuracy of the stabilization process. There are different approaches for motion estimation but these approaches can be generally grouped as time (spatial) and frequency domain. Motion evaluation is the second part of the flow. The results of the motion estimation process are evaluated and the incorrect estimations are removed. Frame correction is the last step to stabilize the frame sequence. This process uses the evaluated motion information and apply inverse shift to the corrupted video frame.

Motion evaluation and frame correction are mainly identical steps for all digital video stabilization methods but motion estimation may be different in different methods. In this chapter, two main motion estimation approaches; time and frequency domain will be explained. For frequency domain, sub-block phase correlation and for time domain, area based full searched algorithms will be detailed.

2.1 MOTION ESTIMATION

2.1.1 Frequency Domain

The translational difference between the reference image and current image in time domain results in phase difference in frequency domain. The basic implementation of this method which provides the most accurate result uses the whole image frame for the phase calculation [25]. However, using whole frame requires high computational load which prevents real time implementation [2]. For that reason, sub-blocks from the original frame are used in phase calculation.

Texture is the total intensity difference of the neighbor pixels. Since, sub-blocks are small parts of the whole frame, they contain less texture and low texture may result in faulty estimations. Therefore, selection of the sub-blocks is very critical for the accuracy. There are two main criteria in the selection of sub-blocks which are size and location.

To increase the texture, the size of the sub-block may be increased, but increase in the size will result in higher computational load. Therefore, the optimum size should be determined which should contain enough texture and be computable in real time. Figure 2 is an example for the texture density according to the sub-block

size. The small sub-block contains less texture, and the larger sub-block contains more texture.

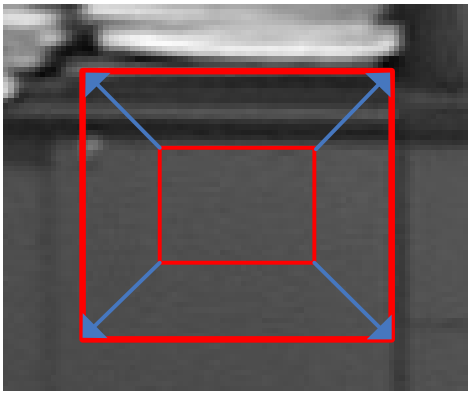

Figure 2: Sub-Block Size & Texture

The second criterion is the location of the sub-block. There are two approaches for the location; constant and dynamic. The constant sub-blocks are located on pre-determined location on the image frame. As shown in Figure 3, these sub-blocks may have different locations and orientations.

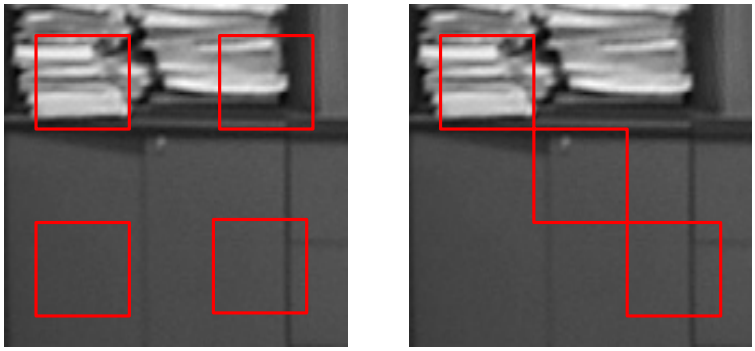

Figure 3: Constant Sub-Block Locations

The constant sub-block location method may also suffer from less texture. To solve this problem, dynamic location method can be used. In this method, the location of the sub-blocks can be changed according to the texture analysis. In Figure 4,

sub-block A contains less texture; however sub-block B contains more, so the sub-block location is moved to location B.

Figure 4: Dynamic Sub-Block Location

Texture analysis (TA) is done by comparing the neighbour pixels in the sub-block in both x and y direction.

$$TA = \sum_{y=1}^{N} \sum_{x=2}^{N} |P(y,x) - P(y,x-1)| + \sum_{y=2}^{N} \sum_{x=1}^{N} |P(y,x) - P(y-1,x)| \tag{2.1}$$

The location and size of the sub-blocks are arranged according to the texture analysis. After that the phase correlation is started with the sub-blocks that have more texture on the reference and current images.

Let R_{sb} and C_{sb} represent the sub-blocks in reference and current images respectively. Assume that there is a translational shift between the sub-blocks:

$$R_{sb}(x,y) = C_{sb}(x + \Delta x, y + \Delta y). \tag{2.2}$$

F_R and F_C are the two-dimensional Discrete Fourier Transforms (DFT) of the sub-blocks R_{sb} and C_{sb}. Firstly column DFT which is in x direction is calculated, then the row DFT in y direction is calculated as follows:

$$F_R(u,v) = \iint (R_{sb}(x,y)e^{-j2\pi ux}dx)e^{-j2\pi vy}dy\,, \tag{2.3}$$

$$F_C(u,v) = \iint (C_{sb}(x,y)e^{-j2\pi ux}dx)e^{-j2\pi vy}dy\,. \tag{2.4}$$

If we put C_{sb} $(x + \Delta x, y + \Delta y)$ instead of R_{sb} in equation (2.3);

$$\begin{aligned}
F_R(u,v) &= \iint (C_{sb}(x+\Delta x, y+\Delta y)e^{-j2\pi ux}dx)e^{-j2\pi vy}dy \\
F_R(u,v) &= \iint (C_{sb}(x',y')e^{-j2\pi u(x'-\Delta x)}dx')e^{-j2\pi v(y'-\Delta y)}dy' \\
F_R(u,v) &= \iint (C_{sb}(x',y')e^{-j2\pi ux'}e^{j2\pi u\Delta x}dx')e^{-j2\pi vy'}e^{j2\pi v\Delta y}dy' \\
F_R(u,v) &= e^{j2\pi(u\Delta x+v\Delta y)}\iint (C_{sb}(x',y')e^{-j2\pi ux'}dx')e^{-j2\pi vy'}dy' \\
F_R(u,v) &= e^{j2\pi(u\Delta x+v\Delta y)}F_C(u,v)
\end{aligned} \tag{2.5}$$

Thus, the translational shift in time domain, results in a phase difference in frequency domain. This phase difference can be obtained by the normalized cross power spectrum:

$$e^{j2\pi(u\Delta x+v\Delta y)} = \frac{F_R(u,v)F_C(u,v)^*}{\left|F_R(u,v)F_C(u,v)^*\right|}. \tag{2.6}$$

The two dimensional Inverse Discrete Fourier Transform (IDFT) of the normalized cross power spectrum gives the phase correlation surface *P(x, y)*. Phase correlation surface has a peak at location (Δx, Δy) due to the delta function in equation (2.7) [2].

$$\begin{aligned}
P(x,y) &= F^{-1}(e^{j2\pi(\Delta x+\Delta y)})\,, \\
P(x,y) &= \delta(x+\Delta x, y+\Delta y)\,.
\end{aligned} \tag{2.7}$$

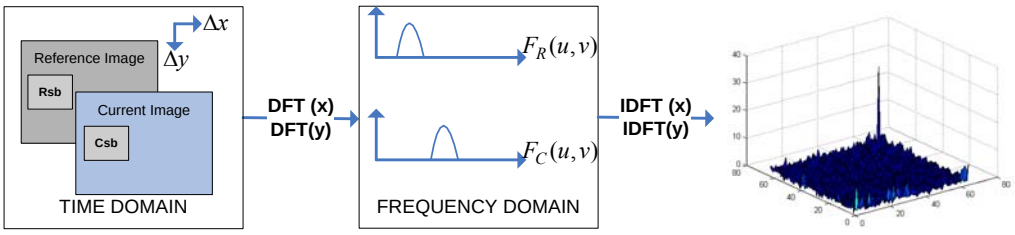

Figure 5: The Flow of Phase Correlation

Figure 5 shows the whole flow of the phase correlation process. At the end of this process, amplitudes of the phase correlation surface points are compared and the maximum value (peak) is found. The location of the peak (Δx, Δy) is used to estimate the translational shift between the sub-blocks.

The location of the peak gives information about the direction of the shift as well. If we separate the sub-blocks into four regions, every region has its own shift direction. Figure 6 shows the possible shift directions and the regions on the sub-block. For example, if the peak is located in region (1), the shift is in the negative direction for both *x* and *y*. The polarities of the shifts are arranged according to the frame structures. y direction represents the rows and x direction shows the columns (pixels). The row number is increasing in +y direction and the pixel number is increasing in +x direction.

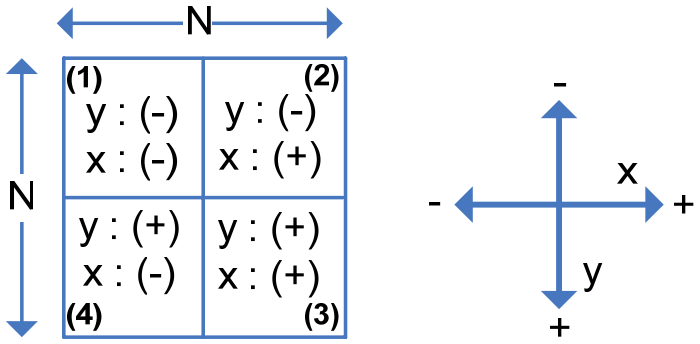

Figure 6: The Shift Directions and Regions of Sub-Blocks

Like the direction, the shift values are also calculated according to the location of the peak. N is the dimension of the sub-block. The calculation of shift value differs if the peak location is greater or less than (N/2). Table 1 shows the calculation steps of the *x* and *y* shift values.

Table 1 : Calculation of Frequency Domain Shift Values

Algorithm steps	Description
1	Calculate the location of the peak value. (Δx, Δy)
2	If Δx is greater than or equal to (N/2). (Δx >= N/2)
X_SHIFT = N - Δx + 1	
3	If Δx is less than (N/2). (Δx < N/2)
X_SHIFT = 1 - Δx	
4	If Δy is greater than or equal to (N/2). (Δy >= N/2)
Y_SHIFT = N - Δy + 1	
5	If Δy is less than (N/2). (Δy < N/2)
Y_SHIFT = 1 - Δy	

2.1.2 Time (Spatial) Domain

There are different methods to calculate motion estimation in time domain. These methods which are feature based, region based and area based use the similarities in the reference and current images, respectively. Among these methods, area based approach which is also called as Full Search (FS) provides the most accurate motion estimation results [6].

Since reference and current images are translational shifted copies of each other, they contain similar sub-blocks. FS tries to find the similar sub-blocks and assign the translational shifts according to the sub-block locations.

FS works on search areas to find the similar sub-blocks. The current image is divided into sub-search areas (SA). A sub-block is selected from the reference image which is located at the centre of the search area. This sub-block is called as reference sub-block. Then, every sub-block on the current image search area which are called as current sub-block, are compared with the reference sub-block.

According to the comparison results, the shift between the reference and current sub-blocks are obtained.

Figure 7 shows the search area on the current image, reference sub-block and possible current sub-blocks around.

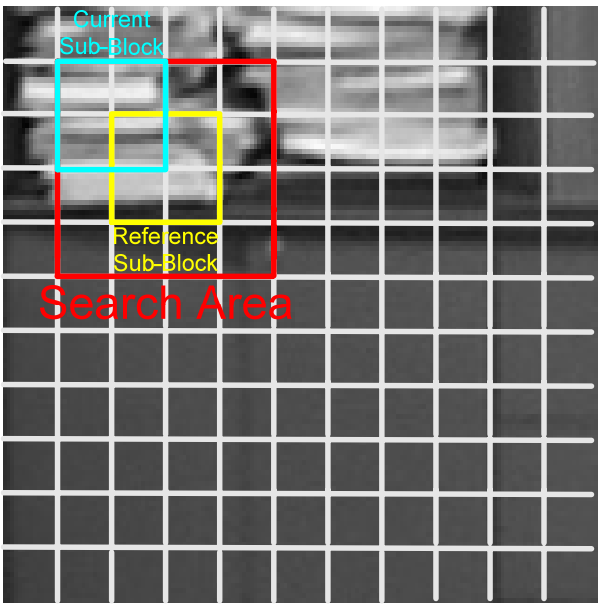

Figure 7: Full Search Algorithm on the Image

For the comparison between reference and current sub-blocks, correlation is used. The correlation is obtained from the pixel intensity values by Mean Absolute Difference (MAD). In equation (2.7), N is the dimension of the sub-block. $R_{sb}(x, y)$ and $C_{sb}(x, y)$ are the pixel intensity values on the reference and current sub-blocks respectively.

$$MAD = \frac{\sum_{x=1}^{N}\sum_{y=1}^{N}\left|RSB(x,y) - CSB(x,y)\right|}{N^2} \tag{2.8}$$

The reference sub-block is compared with every possible current sub-block by using MAD. Then the minimum MAD value is selected as the match between the

sub-blocks in the reference and current images. In spite of Frequency Domain approach, the whole image is divided into search areas and every neighbor search areas are checked. As a result, there occurs several match values between the reference and current images. This match values contain the shift information. As shown in Figure 8, search areas are neighbor of each other. This approach provides accurate matching of the sub-blocks.

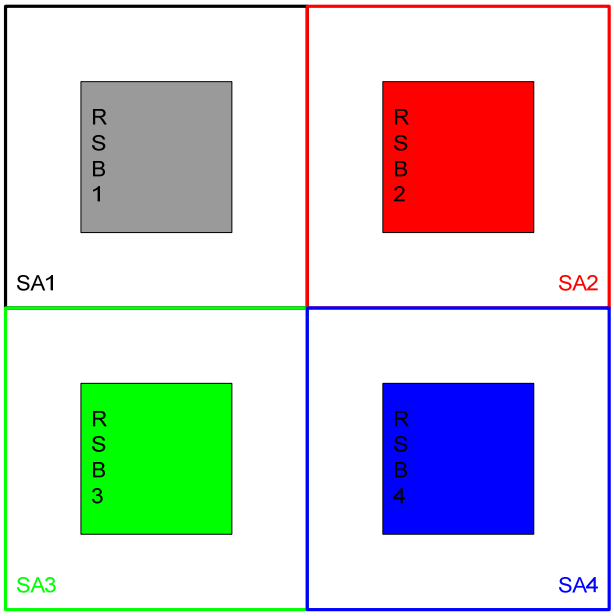

Figure 8: Search Area & Sub-Blocks

After finding the matched sub-blocks from reference and current images, the shift values (motion estimations) are calculated according to the location differences. Table 2 explains the algorithm for the calculation of shift values in time domain. With the calculated shift values, like frequency domain, the texture analysis is done on the reference sub-block. This analysis result is sent to the evaluation part with the motion estimations and MAD.

Table 2 : Calculation of Time Domain Shift Values

Algorithm steps	Description
1	Let search area be MxM and block size be NxN Let $R_{sb}(x, y)$ be the center of the reference sub-block Let $C_{sb}(x, y)$ be the center of the current sub-block Let dx be the shift value in *x* direction and dy be the shift value in *y* direction. dx is equal to (R_{sb} (x) - C_{sb} (x)). dy is equal to (R_{sb} (y) - C_{sb} (y)). If dx is positive, then x-shift is negative, vice versa. If dy is positive, then y-shift is negative, vice versa.
2	Take a search area from current image
3	Take a sub-block from reference image which is at the center of the selected search area
4	Take a sub-block from current image which starts at the first point of search area.
5	Set MAD to high value.
6	Calculate the MAD between the reference and current sub-block.
7	If MAD is less than the previous one, set MAD value to calculated one. Update dx and dy.
8	If the current sub-block reaches the end of the search area, pass to the neighbor search area and move to step 9. Else pass to the neighbor current sub-block and move to step 6.
9	If the search area reaches the end of the image, go to step 10. Else assign dx and dy values as the shift values between the reference and current image for this search area.
10	End of the algorithm

2.2 MOTION EVALUATION

The calculated translational shift values (motion estimations) between reference and current images are evaluated according to their accuracy before using them in the frame correction.

This evaluation is different for frequency and time domain as it is in motion estimation. The number of frequency domain motion estimation results is less than time domain results because frequency domain calculations are done on a predetermined number of sub-blocks while time domain calculations are done for whole image. Firstly, frequency domain evaluation will be detailed, and then time domain approach will be explained

For frequency domain motion evaluation, there are two main criteria which are texture analysis (TA) and the amplitude of the peak. TA and amplitude of the peak determines how the estimated motion from the related block will effect the frame correction.

If the texture of the selected sub-block is not high enough, the phase calculation in frequency domain will not be correct and there occurs several peaks on phase correlation surface with similar amplitudes as shown in Figure 9. Since the motion estimations with low texture can not be used for the frame correction, they should be filtered by a texture threshold (T_T). This threshold value can be different according to the application area. For instance, the threshold value can be lower if the stabilization is performed on naval or airborne platform where texture is low. On the other hand, if the stabilization is performed on a mobile robot in the laboratory where texture is high, T_T should be higher.

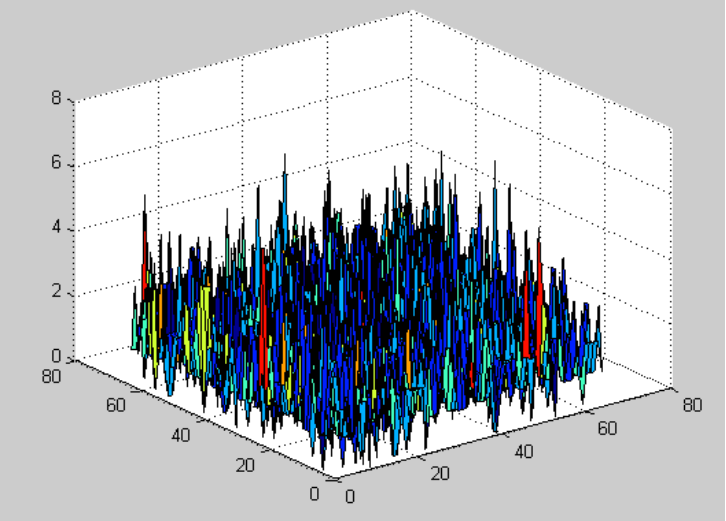

Figure 9: Peak Results for Low Texture

After the frequency domain motion estimations are filtered by T_T, the amplitudes of the peaks on the phase correlation surface are evaluated. Theoretically, there should be only one peak on the surface and the other values should be zero. However, in real time applications there can be blurring between the reference and current images. Also, the pixel intensity values between the successive image frames can be different because of the camera performance and the environmental changes. Therefore, there occur several peaks with different amplitudes on the correlation surface. The comparison of the highest peak with the other peaks gives valuable information about the accuracy of the motion estimation. Also, there can be several higher peaks with similar amplitudes, so the number of peaks which are higher than a predetermined peak threshold (T_P) is also a good data for the accuracy.

There are several approaches to determine the final shift values for frequency domain by using the TA and peak amplitudes. These approaches require different computation load according to their complexity and they provide different accuracies which will be detailed in Chapter-4. Table 3 gives the approaches to calculate the final shift value.

Table 3 : Frequency Domain Motion Evaluation Approaches

Abbreviation	N_S : Number of Sub-Blocks used in motion estimation N_{SF}: Number of Sub-Blocks that can pass the filtering. T_T : Texture Threshold A_P : Peak Amplitude A_s^{PC} : Phase Correlation Surface Average T_P : Peak Threshold P_H : Peak with highest amplitude HPAR: The ratio of the P_H to the A_s^{PC}. XT: Sum of x-shift values. YT: Sum of y-shift values. XF : Final x-shift value YF : Final y-shift value
Approach	**Description**
WOTT (Without T_T)	Do not apply filtering by T_T. Calculate XT and YT. $XF = XT / N_S$; $YF = YT / N_S$; (If XF or YF are not integer, they are rounded)
WTT (With T_T)	Apply filterin by T_T. Calculate XT and YT. $XF = XT / N_{SF}$; $YF = YT / N_{SF}$;
HP (Highest Peak)	Apply filtering by T_T. Find the sub-block with P_H. Assign XF this sub-block's x-shift value. Assign YF this sub-block's y-shift value.
WPT (With T_P)	Apply filtering by T_T. Apply filtering by T_P. Calculate XT and YT. $XF = XT / N_{SF}$; $YF = YT / N_{SF}$;
HPAR	Apply filtering by T_T. Apply filtering by T_P. Calculate HPAR for the filtered sub-blocks. Find the sub-block with highest HPAR. Assign XF this sub-block's x-shift value. Assign YF this sub-block's y-shift value.

In time domain motion estimation, the number of calculated sub-block is related with the image and sub-block size. Let, the image size is R x C and search area size is M x M, then approximately (R / M)*(C / M) sub-blocks are used for the motion estimation. Despite of frequency domain approach, time domain motion estimation results many *x*-shift and *y*-shift values.

These shift values are called as local motion vectors which contain both local and global motions. Local motions are the movements of the objects and global motion is the translational shift. Therefore, the local shift values should be filtered to reach the global values.

Like frequency domain, time domain also uses the texture analysis. Firstly, the motion estimations are filtered by T_T and then they are used in the evaluation. MAD data shows how the reference and current sub-block match each other. If the MAD is less, this means that there are not blurring, intensity change and the sub-blocks are free of local motions. So, a MAD threshold (T_{MAD}) is used to evaluate the motion estimation values coming from the sub-blocks. Table 4 explains the time domain motion evaluation steps.

Table 4 : Time Domain Motion Evaluation Steps

Abbreviation	N_S : Number of Sub-Blocks used in motion estimation N_{SF}: Number of Sub-Blocks that can pass the filtering. T_T : Texture Threshold T_{MAD} : Mean Absolute Difference Threshold XT: Sum of x-shift values. YT: Sum of y-shift values. XF : Final x-shift value YF : Final y-shift value
Evaluation Steps	**Description**
1	Apply filterin by T_T.
2	Apply filterin by T_{MAD}.
3	Calculate XT and YT. $XF = XT / N_{SF}$; $YF = YT / N_{SF}$;

2.3 FRAME CORRECTION

Frame correction is realized by reverse shift operation on the current image. The final shift values which are coming from the motion evaluation step are used in the correction. Since borders of the image can disappear during the shake, frame correction can not generate a complete image. Therefore, it is better to use the images with lower resolution, otherwise some dark areas occurs at the borders.

For instance, if the image has a resolution of 576 (vertical) x 720 (horizontal) which is PAL format, the resolution can be decreased to 512x512. Then, the correction can be applied on at maximum of 32 pixels in the vertical and approximately 100 pixels in the horizontal without losing any pixel. The resolution decrease may be different between the usage areas. Some applications may require all image resolution and accept the dark areas at the borders or some of them may use different lower resolutions. Figure 10 shows the whole image, area of interest and the possible shift values.

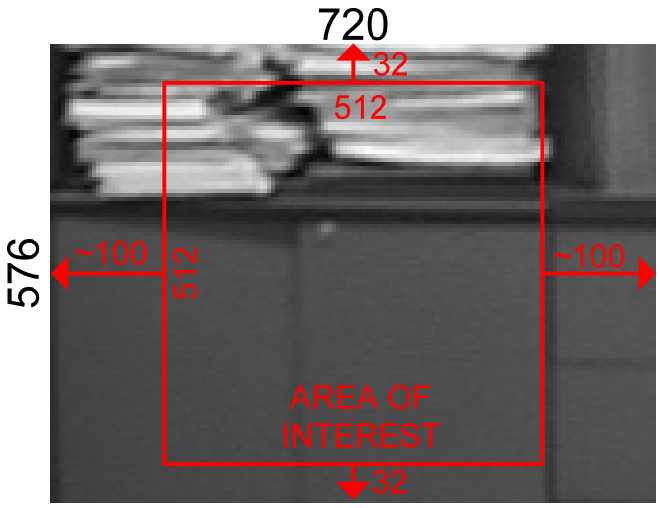

Figure 10: Area of Interest for the Correction

The correction is done by controlling the read sequence of the image from the memory. If there is a shift in $+y$ direction which is equal to Q, the first row is read from the Qth row. This process results in an inverse shift in $-y$ direction. Or if there is $-y$ shift between the reference and current image, the frame read starts at $-Qth$ row. For an image frame, minus row is not possible, so we accept the start row of the area of interest as the first row. For the minus rows, we read from the upper side of the area of interest.

Figure 11 shows the reference, current image and corrected images. In this figure, there is a positive shift in y direction and the correction is realized by changing the first row location.

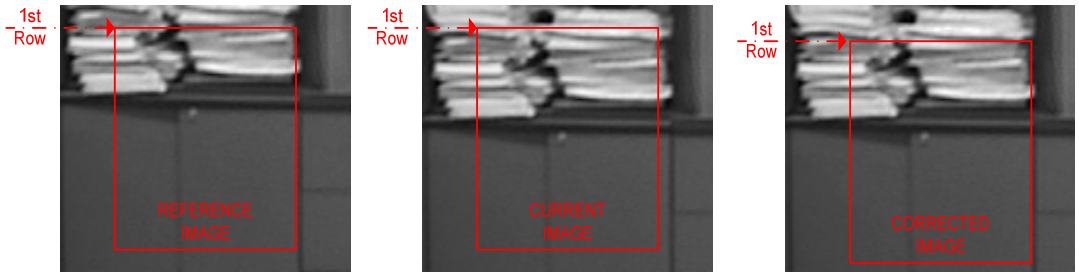

Figure 11: Reference & Current and Corrected Images

CHAPTER 3

IMPLEMENTATION OF REAL TIME VIDEO STABILIZATION

In this thesis study, Phase Correlation (PC) and Full Search (FS) digital video stabilization algorithms are implemented in Field Programmable Gate Arrays (FPGA), since FPGAs are suitable processing units for the applications where computation load is high [9].

The basic advantage of the FPGA structure is its flexibility. Namely, different tasks which need to be run in parallel can be implemented in FPGA. Also, the tasks in the FPGA can be changed by loading a new configuration. This is the basic difference between the ASICs and FPGAs. ASICs can also be used for parallel tasks but they can not be reconfigured [13].

However, only FPGA is not enough for real time video processing application, external units are required for data management [10]. To clarify these units, firstly, the elements of the real time video processing environment will be explained. Then, the FPGA implementation of PC and FS algorithms will be detailed.

3.1 REAL TIME VIDEO PROCESSING ENVIRONMENT

Real time video processing requires a well defined timing between units. Every task should be completed in a certain time to match with the real time constraints [12].

Also, some tasks should be constructed in a pipeline order to maintain video flow. Pipelining structure will be explained in 3.1.3 section.

The processing starts with the video inputs. Incoming video frames are stored in external memories. Then, the stored data is read by processing units and the results are displayed via video outputs.

3.1.1 Video Inputs

There are mainly two types of video inputs; analog and digital. Analog videos are widely used in many areas for years. The basic analog video standards are PAL (Phase Alternation Line-Europe) and NTSC (National Television Standards Committee-USA) which have been used since the foundation of the color TV [14].

Digital video standards are newer when compared to analog standards. T mheir usage is increased by the design of high speed interfaces. The well known digital standards are DVI (Digital Video Interface) and Cameralink [15, 16].

The type of video input is related with the capability of the hardware. In this thesis study, the daughter card which is located on the main board may receive both analog (PAL-NTSC) and digital (DVI) video inputs. There are integrated circuits (ICs) to capture the video signals. Since the cameras produce PAL analog video, our processing environment is constructed to receive analog video, but future applications may also use DVI digital video.

PAL analog video which has a resolution of 576 (row) x 720 (pixel) @ 25Hz, is captured by video decoder IC (Texas Instruments-TVP5154). This IC decodes the analog video and produces digital signals which are sent to FPGA. There are internal registers in the video decoder to control its functionality. These registers are set according to the application, namely the video input may be converted to NTSC.

The registers are controlled by I2C (Inter Integrated Circuit) interface which is implemented in FPGA [17]. Figure 12 shows the video flow from the cable to FPGA.

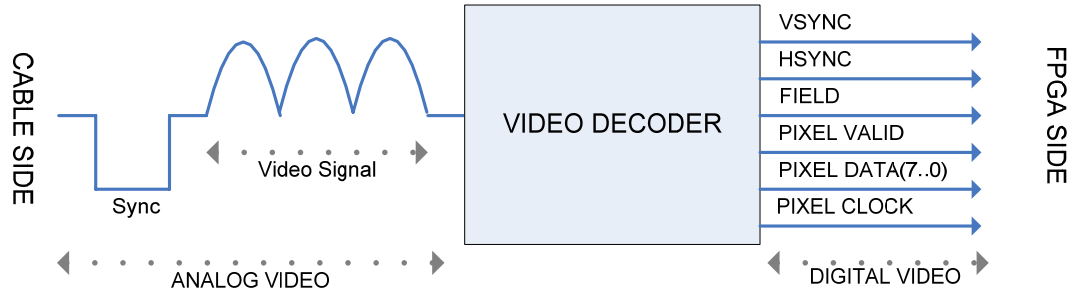

Figure 12: Video Decoder Interface

Digital video signals are used to obtain active video pixels which carry the intensity values of the captured scene. VSYNC signal indicates the active and blank time of the video frame. It is high when the video frame is active and it is low in the frame vertical blank. FIELD signal gives information about the frame field. PAL analogue video is transmitted in interlace format [14]. Firstly odd lines then even lines are transmitted. This interlace sequence is also related with FIELD signal, namely, when FIELD is high, it means that odd field is active and when it is low even field is active. HSYNC is used for the pixel lines and PIXEL VALID signal covers the active PIXEL DATA which is 8-bit.

PIXEL DATA is sent to the FPGA in YCbCr format. In this format Y carries the luminance information and Cb & Cr hold the chrominance data [18]. In this implementation only Y data will be used, for this reason Y data is separated from the other data by using the PIXEL VALID and PIXEL CLOCK signals. PIXEL CLOCK is 27MHz for the digitized PAL video. Figure 13 shows the digital video frame signals and their orientation according to each other.

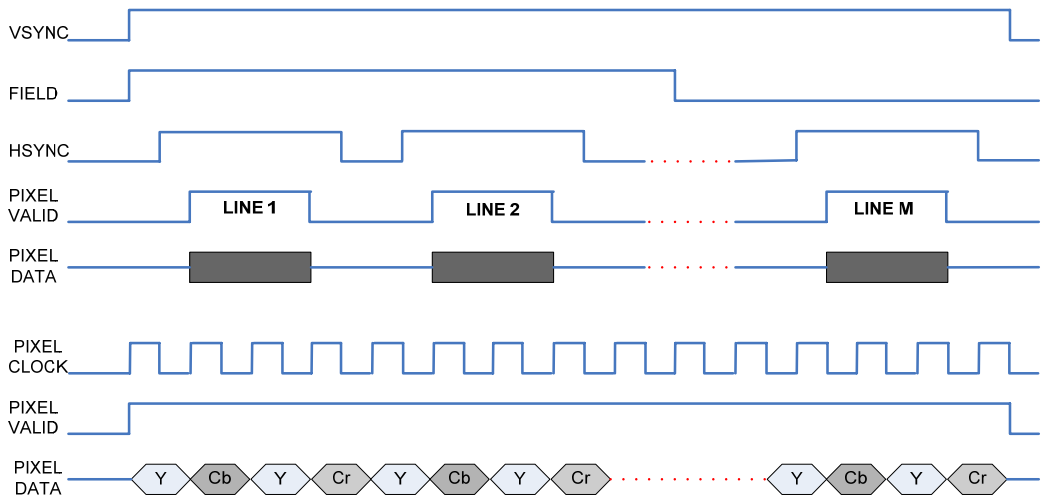

Figure 13: Video Frame Signals

3.1.2 Memory Interfaces

In digital video stabilization, external memories are used to store data which are the pixel intensity values before or after processing. There are mainly two types of memories which are used in real time video processing applications. These memories are SRAMs (Static Random Access Memory) and SDRAMs (Synchronous Dynamic Random Access Memory).

SRAMs are formed in address-data format. There are address locations that store the data which is shown in Figure 14. They provide quick reach to the address locations, namely write/read operation only takes a few clock. For that reason, they are suitable for the video processing applications where pixel values from different line-column locations are required [19]. However, SRAMs suffer from memory capacity and speed. Most of the SRAM bus interface is not fast as SDRAMs and there is not large memory space as in SDRAMs.

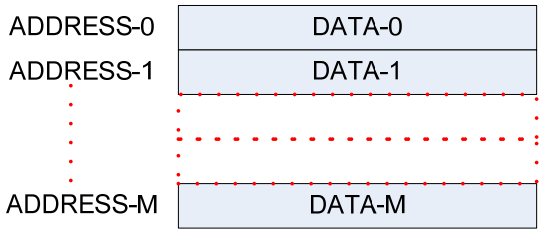

Figure 14: SRAM Internal Structure

SDRAMs are in bank-row-column-data structure. The banks are the main storage units and every bank is formed by rows which contain columns. The data is stored in these columns. The internal structure of SDRAM is shown in Figure 15. Since, SDRAM structure is more complex when compared with SRAMs; their write/read operations require more clock. First of all, the bank is selected, then the row is opened and the data can be written in or read from the column. If the write/read operation continues on the same row, the next operation only takes one clock. So, SDRAMs are suitable for applications where the same line is written or read. Also, SDRAMs have high data bandwidth since they can work in high clock frequencies. In video stabilization, the basic idea is to compare the lines from reference and current image. Therefore, SDRAMs are more suitable for the stabilization and used in other stabilization systems [20].

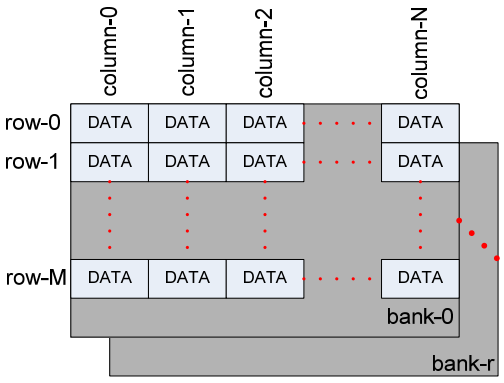

Figure 15: SDRAM Internal Structure

There are several internal blocks in FPGA implementation that require write/read interface with SDRAM. These blocks may try to reach memory at the same time. Therefore, a memory management block is implemented to control the data flow. This block is called SDRAM Controller and shown in Figure 16.

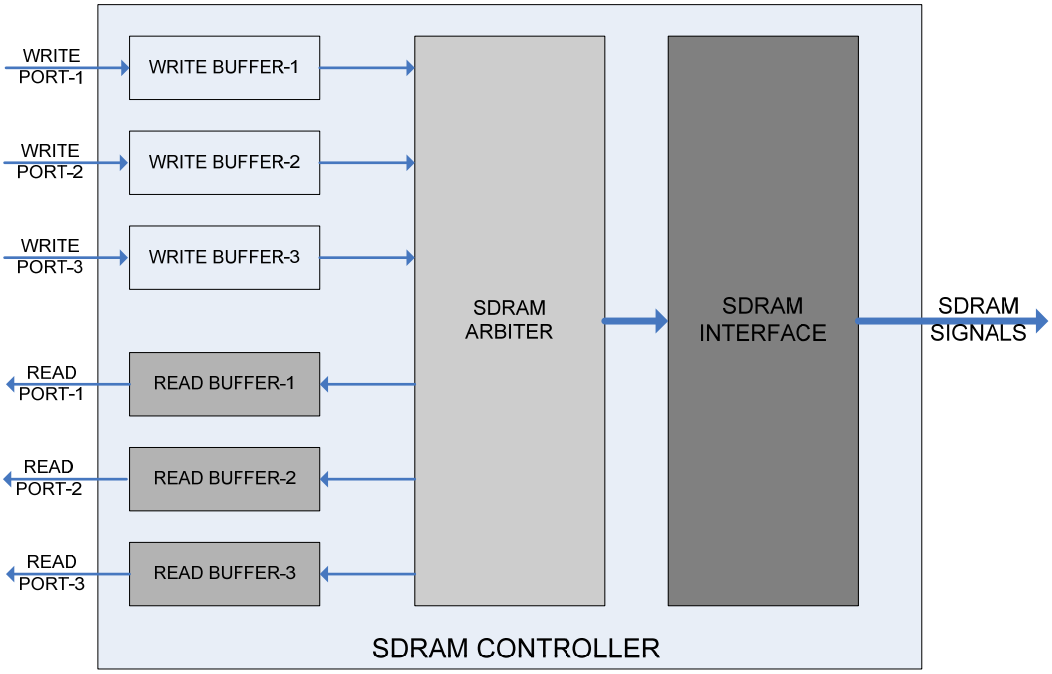

Figure 16: SDRAM Controller

SDRAM Controller consists of write/read buffers, SDRAM Arbiter and SDRAM Interface. Write buffers are used as a pre-cache to prevent data lost. Read buffers provide continues data flow for the read blocks since arbiter can load the buffers before they are needed. The total number of write/read buffers is six and the total number can be changed by slight code modification on the arbiter. The write/read sequence and signals of these buffers will be detailed at APPENDIX A.

SDRAM arbiter controls the write/read operation to the SDRAM memory. It checks the buffers for a write or read request one by one. The priority of the SDRAM is for

write buffers because the incoming data should be stored as quick as possible to prevent buffer overwrite and data lose.

SDRAM Interface generates the necessary signals for SDRAM memory; it writes or read according to control signals that come from the arbiter. This SDRAM Interface is an IP (Intellectual Property) provided by Altera (FPGA Vendor) [21].

3.1.3 Processing Units

The processing unit selection is done according to the application requirements. For the applications where complex mathematical operations are required DSP (Digital Signal Processor) can be used [22]. For the algorithms that require iterative calculations a CPU (Central Computing Unit) is a better selection. The graphic processor can be adapted for specific video processing applications [23]. Among all these different processing units, high computation load is a problem, especially when parallelism is required. FPGAs are dedicated hardware units to solve such problems [11].

The main sub-unit of an FPGA is LE (Logic Element). LEs can be configured according to the application. Since, LEs are individual, there can be different combinations which can work in parallel. This capability makes the FPGAs proper units that can handle high computational load [9].

Figure 17 shows how FPGA parallel structure works. There are three tasks in the application, task-1's and task-2's outputs are sent to the task-3. Each task is implemented by different LE combination, so these tasks can be completed at the same time. Also this is a good example to explain pipeline structure. Each task receive its input and generate the outputs, outputs of some tasks become the inputs of another task. Therefore, there exist a pipeline and in every clock cycle, each task may continue its process.

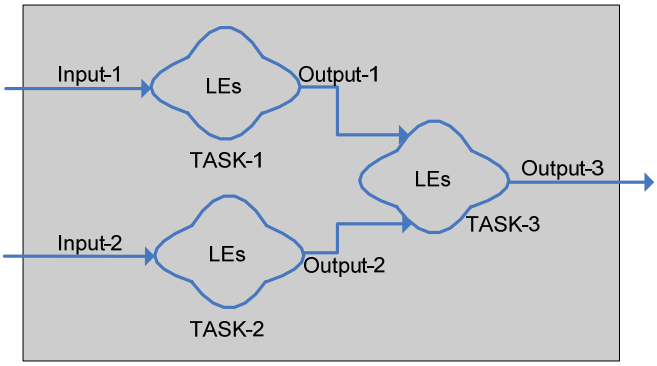

Figure 17: FPGA Parallel Structure

In this thesis study, Cyclone III EP3C120 FPGA is used as the processing unit. This FPGA contains 119,088 LEs, 4 PLL (Phase Locked Loop), 3,981,312 internal memory bits and 576 9x9 dedicated multipliers [19]. PLLs generate different clocks from the input clock. In an FPGA implementation many different clock domains are required by different blocks. Internal memory bits are used by the blocks which need to store small amount of data and can not access to external memories like SDRAMs. Dedicated multipliers are used for the mathematical operations. In Figure 18, the internal structure of FPGA is given.

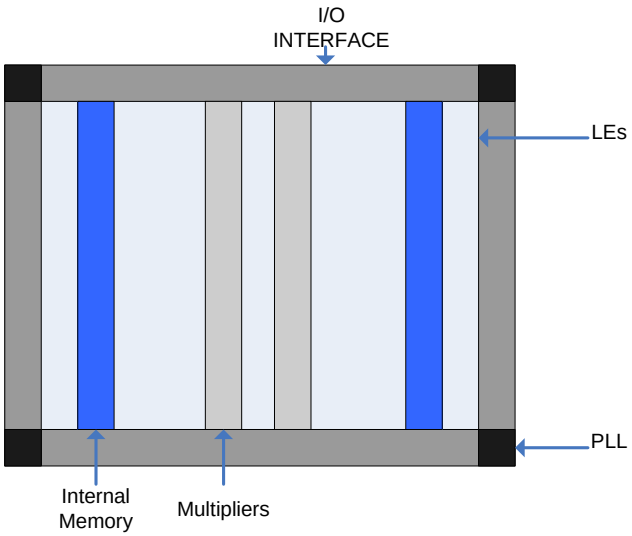

Figure 18: Internal FPGA Structure

3.1.4 Video Outputs

Video outputs are used to display the results of the processing blocks. In video stabilization, the output is the stabilized video sequence, in stereo matching it is the disparity map and in object tracking, it is the sign on the object. Like video inputs, there are two video output types; analog and digital. All video input standards are also valid for video outputs.

Analog video outputs can be displayed on CRT (Cathode Ray Tubes) monitors, digital outputs are displayed on digital monitors like LCD (Liquid Crystal Display). In this thesis study, the output of the stabilization algorithm will be displayed on LCD via DVI standard. The DVI signals are generated by FPGA and the physical transmission is handled by DVI transmitter IC (Texas Instruments-TFP410).

Figure 19 shows the video output interface. DVI transmitter receive the necessary control signals and pixel values from the FPGA and generates the physical transmission signal according to the standard. There are 4 output channels in the transmission, these channels are differential. CH-0, CH-1 and CH-2 carry the pixel values and synchronization signals, CLOCK channel transmits the clock .

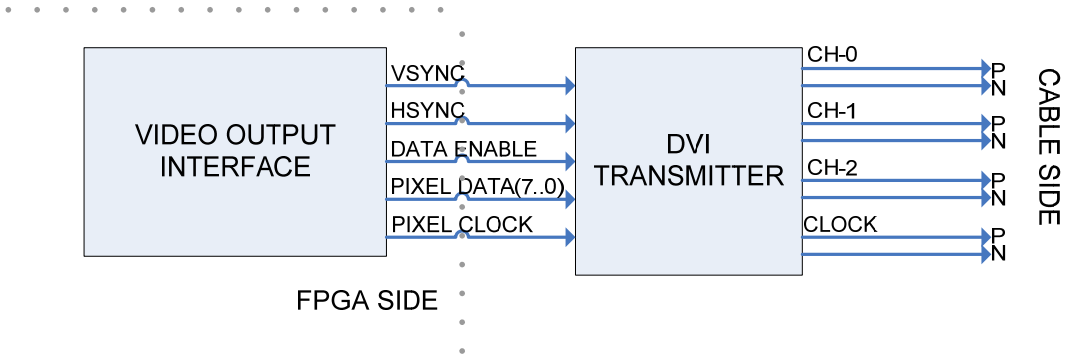

Figure 19: Video Output Interface

For the real time video processing environment, Altera Cyclone III Development Board and Bitec HSMC (High Speed Mezzanine Card) daughter cards are used [19]. Daughter cards provide video input and output interfaces; they are plugged to the Cyclone III main board. Figure 20 shows the hardware structure and the components on the cards; also in Figure 21 the internal structure of the FPGA is given.

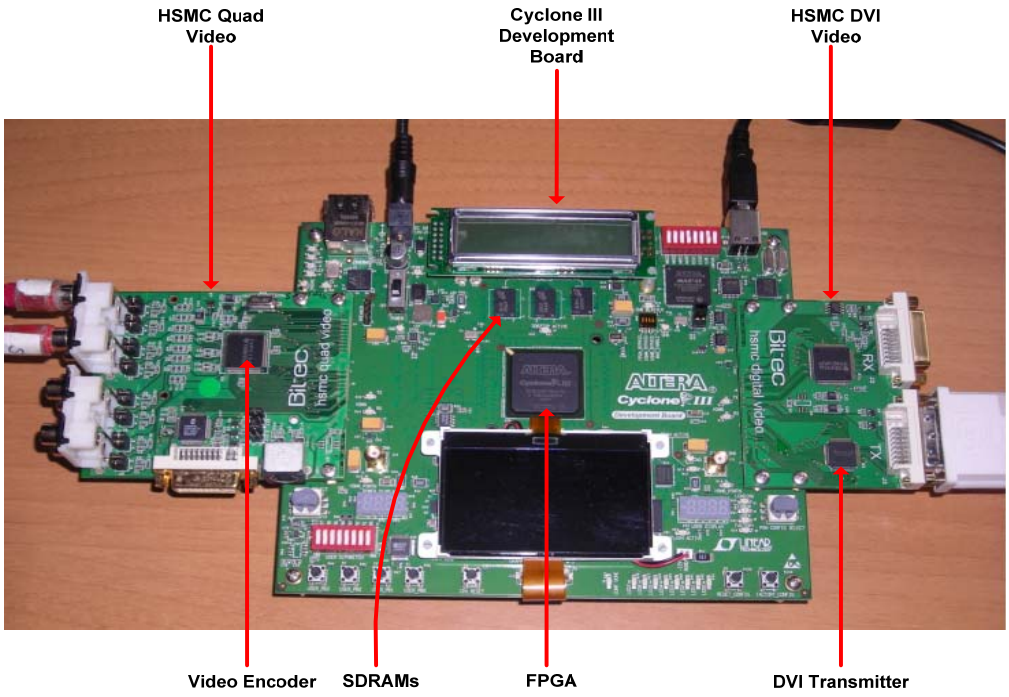

Figure 20: The Hardware Structure

The digitized analog video signals are read by the video input interface. Then, this video is written to the specific location of SDRAM memory by video write interface. Reference and current images are written to the different part of the memory and video read blocks are informed about the locations. After video write block finishes the operation, video read blocks get the reference and current image from the memory. Video stabilization block processes the data and calculates the

shift values between the images. Finally, video output block read the current image according to the shift values and display it on the screen.

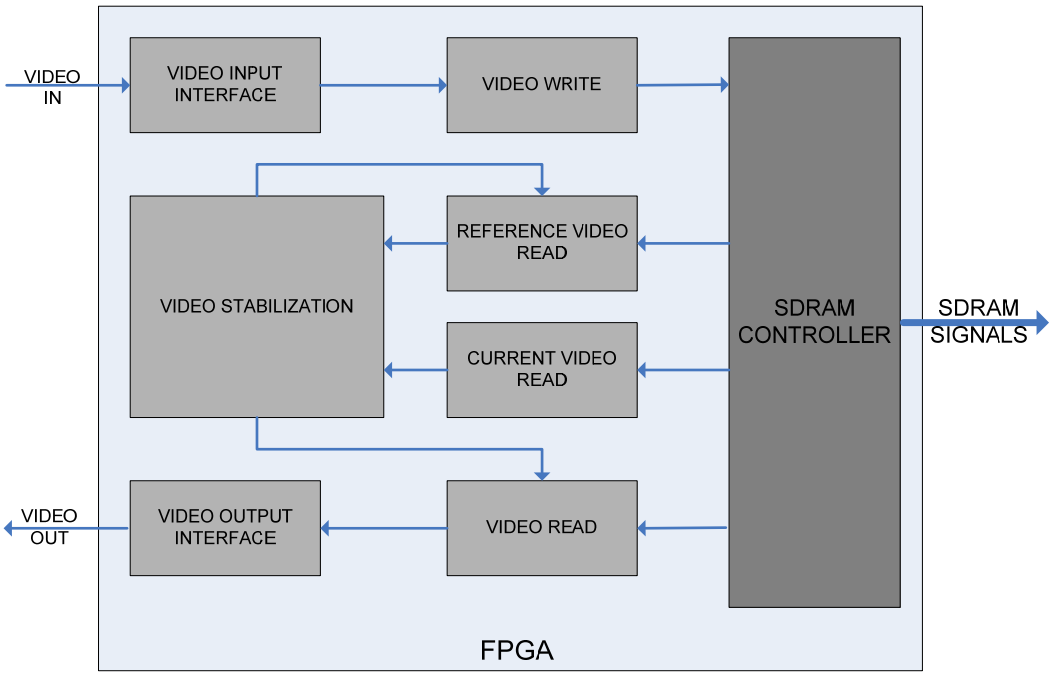

Figure 21: FPGA Internal Structure

There are three input image buffers (IIB) and three output image buffers (OIB) in SDRAM. IIBs are used to store reference and current images. The incoming video frames are written to these buffers consecutively. OIBs are used for display purposes, the current image is displayed from these buffers according to the calculated shift values. Figure 22 shows the buffers and video connections.

Three buffers method prevents overwrite problem, while two buffers are used for processing, the other buffer is used to store the incoming video frame. In output path, the output video may have a different frame rate than input video, so using three buffers provide opportunity to change frame rate without data lost.

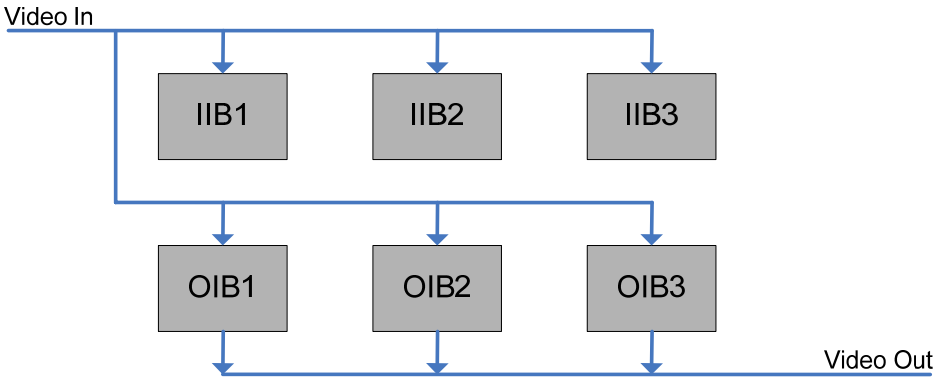

Figure 22: Image Buffers in SDRAM

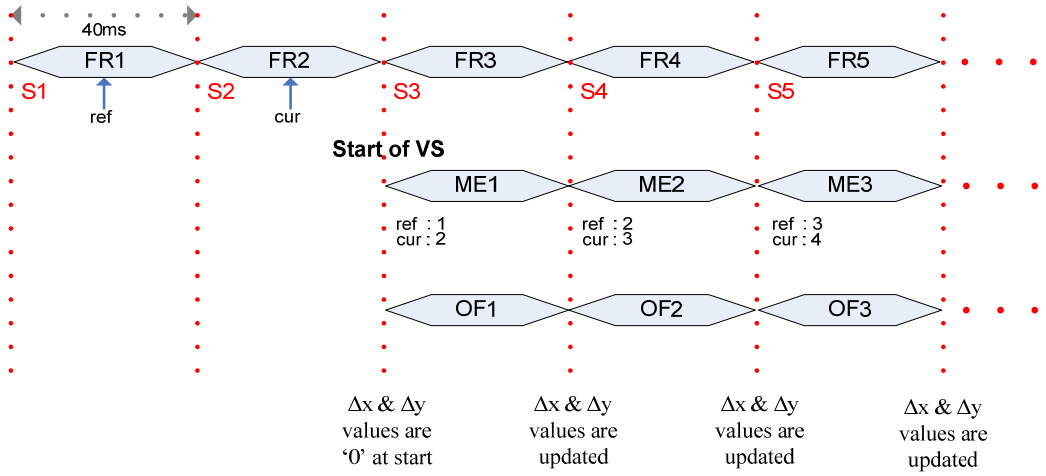

Figure 23: Video Flow in Stabilization

Figure 23 shows the video flow in digital video stabilization and Table 5 explains the steps.

Table 5 : Video Flow and Stabilization Steps

Abbreviation	FR : Frame ref : Reference Image. cur : Current Image. VS : Video Stabilization ME : Motion Estimation OF : Output Frame. IIB : Input Image Buffer OIB : Output Image Buffer
Steps	**Description**
S1	Write FR-1 to the IIB-1 and OIB-1 FR-1 is the first frame and it is the initial reference image.
S2	Write FR-2 to the IIB-2 and OIB-2 FR-2 is the initial current image. Set Δx & Δy (shift values) to zero.
S3	Write FR-3 to the IIB-3 and OIB-3 Read OIB-1 (FR-1) according to Δx & Δy for video output display. Read IIB-1 according to Δx & Δy as reference image (FR-1). Read IIB-2 as current image (FR-2). Run ME algorithm and update Δx & Δy .
S4	Write FR-4 to the IIB-1 and OIB-1 Read OIB-2 (FR-2) according to Δx & Δy for video output display. Read IIB-2 according to Δx & Δy as reference image (FR-2). Read IIB-3 as current image (FR-3). Run ME algorithm and update Δx & Δy .
S5	Write FR-5 to the IIB-2 and OIB-2 Read OIB-3 (FR-3) according to Δx & Δy for video output display. Read IIB-3 as reference image (FR-3). Read IIB-1 as current image (FR-4). Run ME algorithm and update Δx & Δy .

3.2 PHASE CORRELATION FPGA IMPLEMENTATION

This section explains the necessary FPGA blocks and signals for the phase correlation method. FPGA blocks are coded with VHDL (Very High Speed Integrated Circuit Hardware Description Language). The Video Stabilization block in Figure 21 is the main block for frequency and time domain. For frequency domain, this block is named as Video Stabilization Frequency (VSF).

VSF consists of *VSF_main_controller*, *texture_analysis*, *phase_correlation, VSF_motion_evaluation* and *reference&current* internal memories. The connections between these blocks are shown in Figure 24.

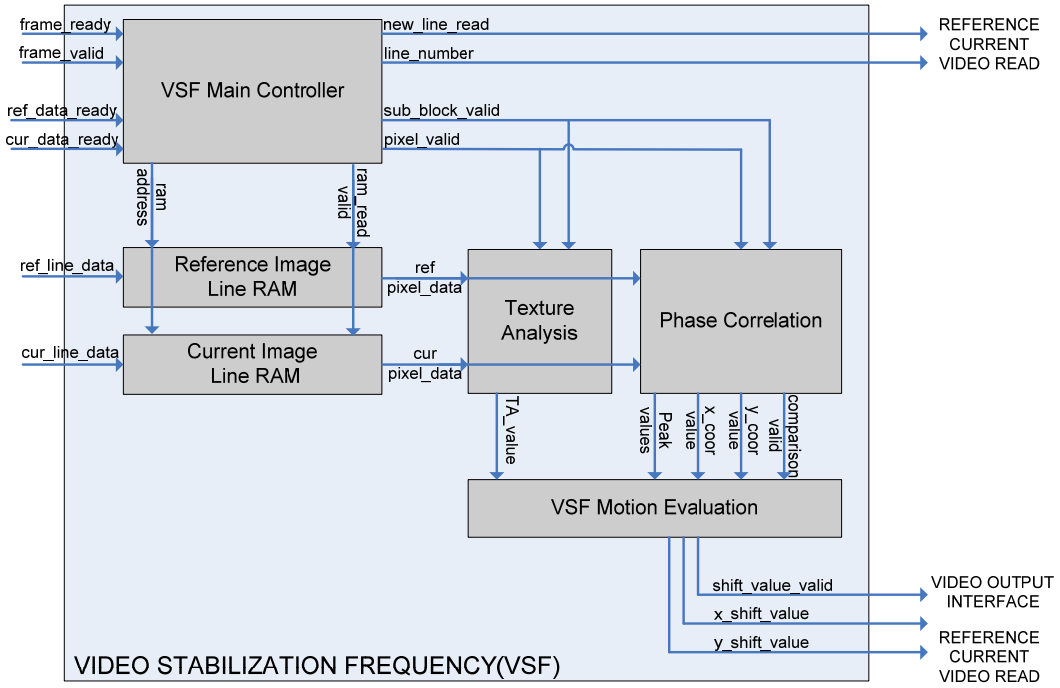

Figure 24: VSF Internal Structure

VSF_main_controller is responsible for the pixel data management. This block controls the stabilization flow by checking and generating several signals. Frame ready signal asserted by video write interface (VWI) when it finishes reference and current frames write operation to the SDRAM. Frame valid signal which is controlled by video input interface, indicates the start of a new frame and it is the general control signal for all blocks to restart the operations.

The main controller organizes the whole reference and current image line read operations from the SDRAM via video read interfaces (VRI). According to the sub-block location which is described in section 2.1.1, the controller sends the line numbers to the VRI with new line read signal. After VRI finishes its read operation

from SDRAM, it writes the pixel data to the *reference & current* internal ram and asserts a data ready signal for the main controller. Main controller starts read operation from internal rams and assert pixel valid signal to inform *texture_analysis* and *phase_correlation* blocks that the coming pixel values will be inserted into the calculations. Sub-Block valid signal is used to separate different sub-blocks. It is asserted when a new sub-block is going to start and deasserted when the sub-block is finished.

Texture analysis (TA) of the sub-block is done by *texture_analysis* block. Sub-block valid signal starts a new texture analysis and the analysis result is sent to the evaluation block by TA values signal. Figure 25 shows the internal structure of the texture analysis block. Pixel comparison block calculates the pixel differences as described by equation (2.1). Line FIFOs (First In First Out) hold the previous reference and current line pixel values which are used in the vertical difference calculation. When pixel valid signal is asserted, comparison block calculates the pixel difference between the neighbor pixels among horizontal and vertical direction. It adds the difference values to the previous one and find the total result. At the end of the sub-block, the total difference (TA value) is sent to the evaluation block.

Phase Correlation (PC) block consists of *PC_main_controller*, *1-D_fft*, *ref&cur_ram*, *line_FIFO*, *normalization* and *magnitude* block. PC block realizes the mathematical operation described in section 2.1.1. In Figure 26, the sub-blocks of PC are shown.

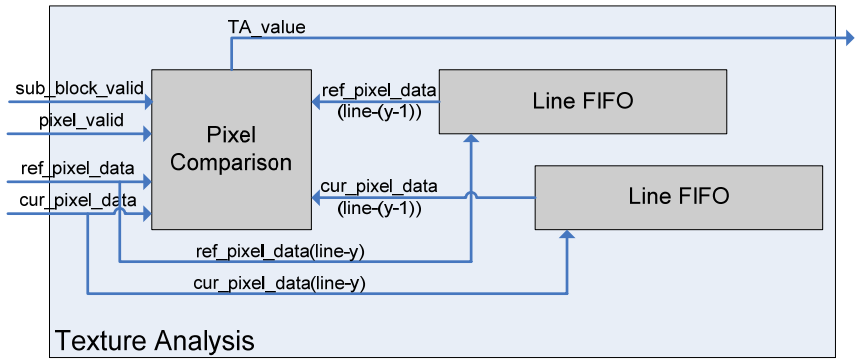

Figure 25: Texture Analysis Block

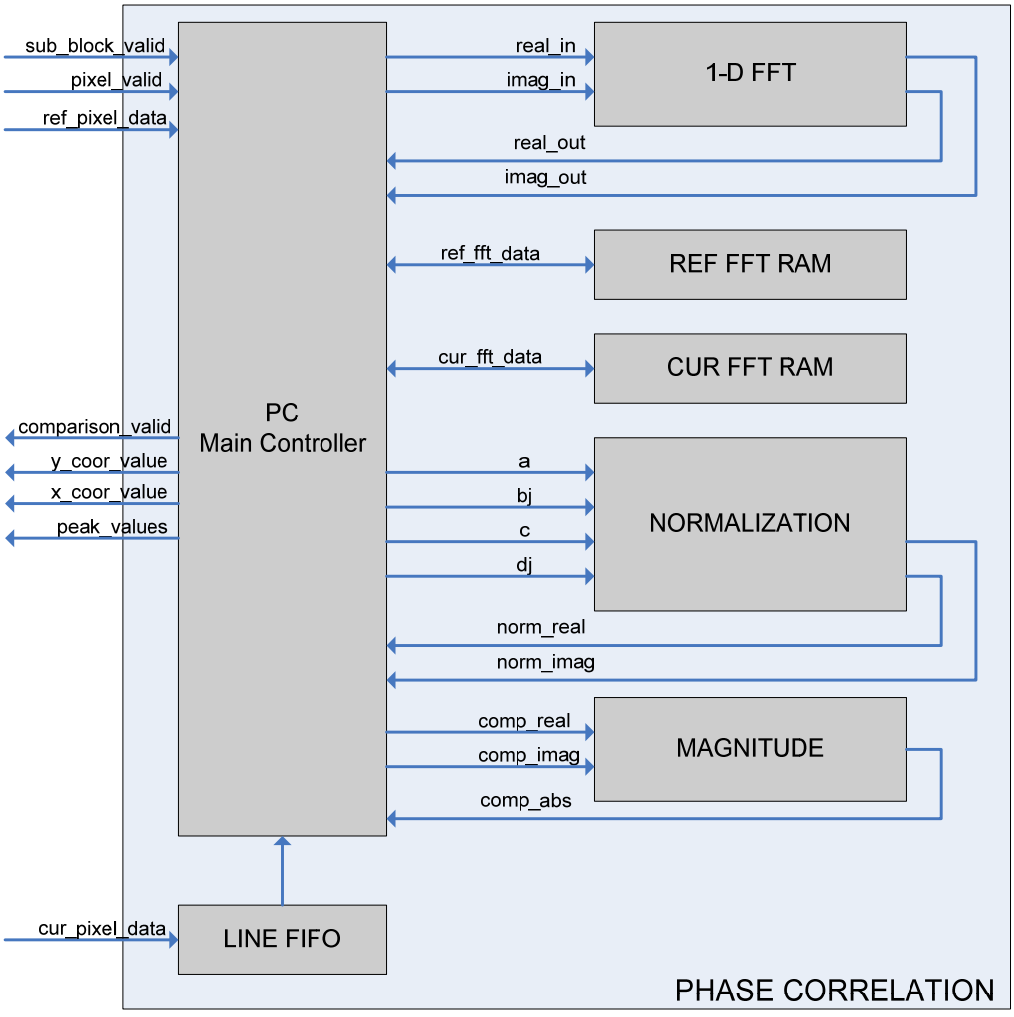

Figure 26: Phase Correlation Block

PC main controller starts the operation with the assertion of sub-block valid signal. It generates the necessary signals for the *1-D_fft* block which performs one dimensional Discrete Fourier Transform. Reference and current line pixel values arrive the PC at the same time with pixel valid signal, but only one line can be processed in *1-D_fft* block at a time, so current line pixel values are stored in line FIFO and processed after reference line is finished. The reason for using only one fft block is to reduce the total logic usage in FPGA. PC main controller arranges the data flow and perform all operations which are required DFT and IDFT with one fft block. *1-D_fft* block is an IP which is provided by Altera [24]. It can perform both DFT and IDFT. The properties of this IP block will be detailed at APPENDIX A.

The numbers in frequency domain has real and imaginary parts, PC main controller assign reference and current pixel values to real_in signal and set imag_in to zero. Then *1-D_fft* block gives the fft result with real_out and imag_out signals. The results of the reference and current line DFT are stored in *ref_fft_ram* and *cur_fft_ram* respectively. After, sub-block finishes, the PC main controller starts the transpose DFT with the stored data in the RAMs. The transpose process is done by controlling the read sequence. The data is written to the ram in row-column order which is shown in Figure 27, but it is read in column-row order. This read sequence is equal to transpose process and there is no need to use another memory for transpose.

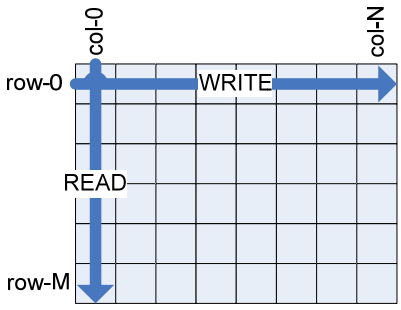

Figure 27: FFT_RAM Write/Read Sequence

After PC main controller finishes the transpose DFT of both reference and current sub-block, it starts the normalization. Let $F_R(u,v)$ and $F_C(u,v)$ be the 2-D DFT of reference and current sub-block respectively. Then,

$$F_R(u,v) = a(u,v) + b(u,v)\,j \tag{3.1}$$

$$F_C(u,v) = c(u,v) + d(u,v)\,j \tag{3.2}$$

$$F_R(u,v)xF_C(u,v)^* = (a+bj)*(c-dj)$$

$$F_R(u,v)xF_C(u,v)^* = (ac+bd)+(bc-ad)\,j \tag{3.3}$$

$$\left|F_R(u,v)*F_C(u,v)\right| = \sqrt{(ac+bd)^2+(bc-ad)^2} \tag{3.4}$$

The normalized value of $F_R(u,v)$ and $F_C(u,v)$ is obtained from the division of (3.3) by (3.4);

$$Norm(F_R,F_C) = \frac{(ac+bd)+(bc-ad)\,j}{\sqrt{(ac+bd)^2+(bc-ad)^2}} \tag{3.5}$$

The normalization block performs the operations which are defined from (3.3) to (3.5). The internal structure of the normalization block is shown in Figure 28.

The normalization outputs; norm_real and norm_imag values are directly sent to the inverse Discrete Fourier Transform (IDFT). The results of the inverse fft operation are written to the ref*_fft_ram*. The same ram is used to store different operation results and reusability decreases the internal memory usage which provides effective usage of FPGA resources. After all IDFT values are stored in the ram, transpose IDFT starts. The transpose operation is again realized by ram read sequence as explained in Figure 27.

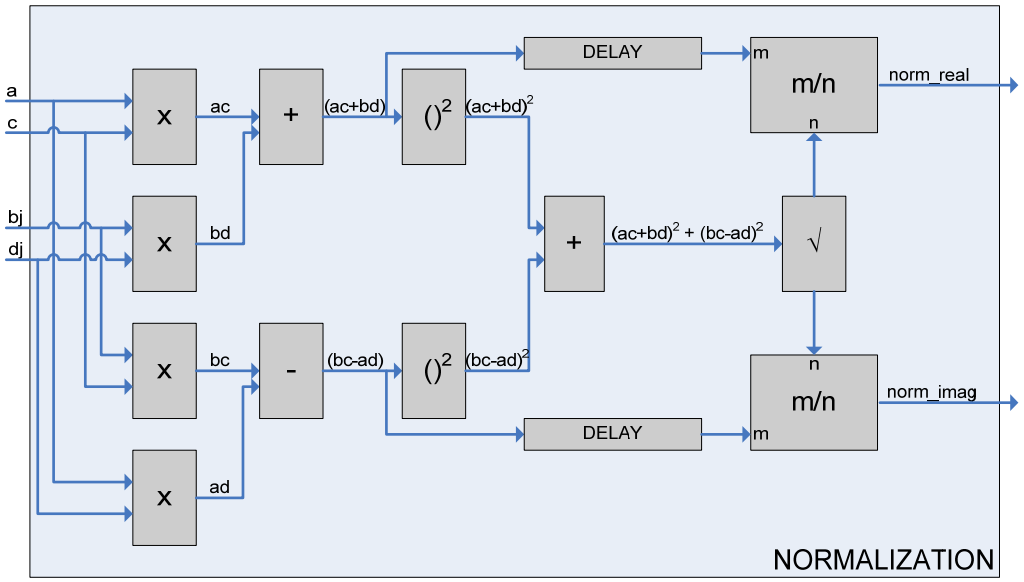

Figure 28: Normalization Block

The results of the transpose IDFT may have also real and imaginary parts. To construct the peak surface, the absolute magnitude of these values should be found. This operation is performed by magnitude block which is shown in Figure 29. Comp_real and comp_imag signals carry the real and imaginary parts respectively. Comp_abs gives the absolute magnitude value. Let *P(x,y)* be the two dimensional IDFT of the normalization values, then;

$$P(x, y) = e(x, y) + f(x, y)j \tag{3.6}$$

$$|P(x, y)| = \sqrt{e^2 + f^2} \tag{3.7}$$

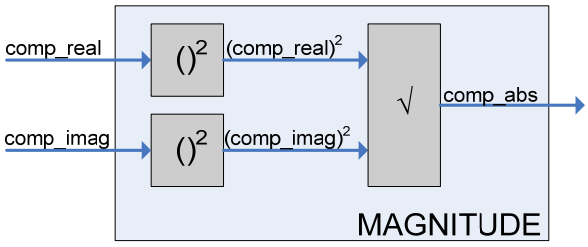

Figure 29: Magnitude Block

The magnitude block's outputs form the peak surface. The peak values contain information about the shift value. Since the implementation is in real time, there can be several peaks on the surface due to mismatch between the consecutive frames which may results from environmental changes such as lightning. To find the correct peak value and also the shift value, the peak values with their x & y coordinate are sent to the VSF Motion Evaluation block by comparison valid signal.

Motion evaluation block receive the peak values and texture analysis values for the calculated sub-blocks. There are different evaluation methods which are explained in section 2.2. In this FPGA implementation HPAR (The ratio of Highest Peak to the surface average) method will be used. The selection reason of this method will be explained in results chapter.

The results of the evaluation block are the x_shift and y_shift values. These values are sent to the video read and video output interfaces. The video read interface arrange the number of the requested line by video stabilization block according to the y_shift value. Also, VRI changes the write start address of the reference internal ram which is shown in Figure 24, according to x_shift value. These coordinate changes perform the frame correction operation in real time. The details of frame correction are explained in section 2.3 and Figure 30 shows the operation. Since, the video stabilization block continues to read at same locations, the coordinate changes result in inverse shift of the frame.

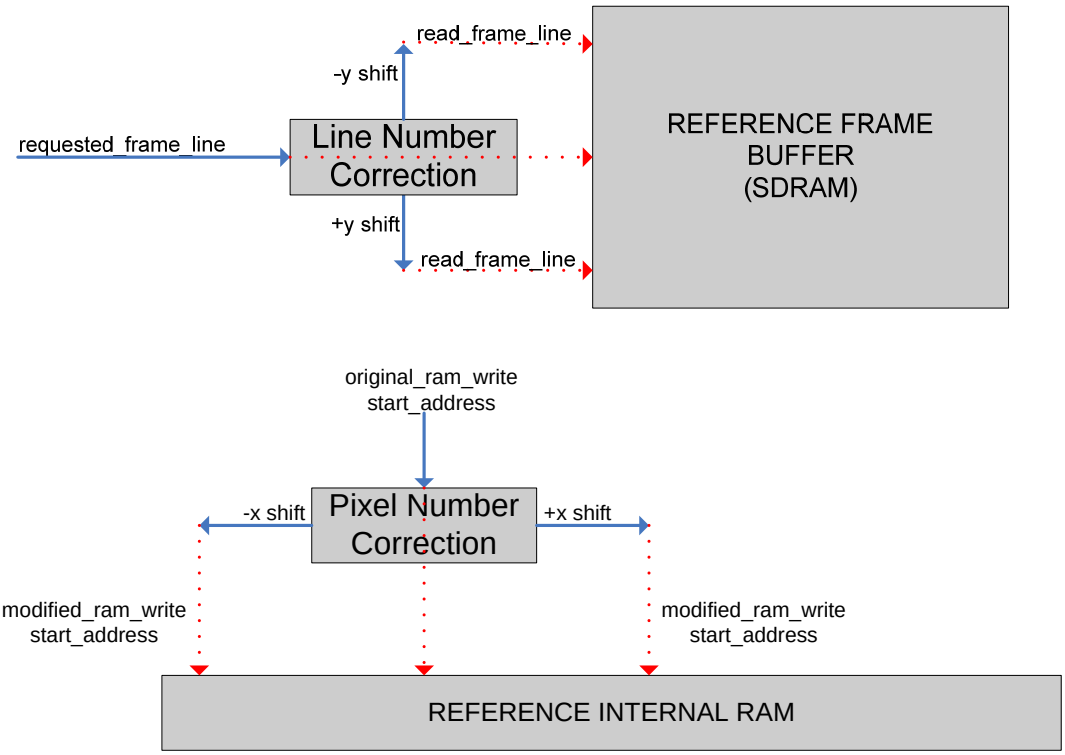

Figure 30: Frame Correction Operation

3.3 FULL SEARCH FPGA IMPLEMENTATION

The FPGA implementation of full search algorithm will be explained in this chapter. Like phase correlation, this method is also coded by VHDL. The main video stabilization block in Figure 21 is called as Video Stabilization Time (VST).

The internal structure of VST is shown in Figure 31. *VST_main_controller, reference_block_RAM, search_area_RAM, full_search* and *VST_motion_evaluation* are the sub-blocks. *VST_main_controller* controls the pixel data flow for reference block and search area. As explained in section 2.1.2, the reference block is located at the centre of the search area, so the line numbers for the reference block and search area are different. The main controller arranges the line numbers which are

sent to video read interface in order to locate the reference block at the centre of the search area.

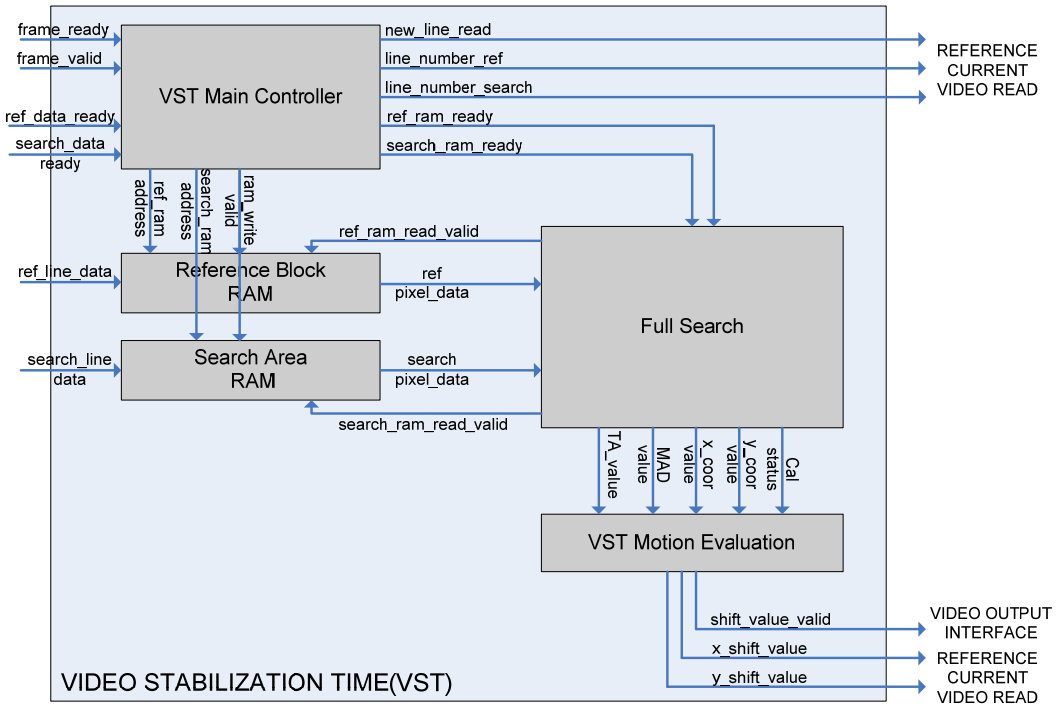

Figure 31: VST Internal Structure

The main controller has two separate processes which are responsible from reading the necessary reference and search lines from the memory. The read lines are written to the internal RAMs which are *reference_block_RAM* and *search_area_RAM*. After the processes fill the RAMs with pixel data, main controller assert the ref_data_ready and search_data_ready signals to inform the *full_search* block to start mean of absolute difference (MAD) calculations.

The internal structure of the internal RAMs is shown in Figure 32. In this representation, *reference_block_RAM* is located on *search_area_RAM* to show their orientation. The current image lines which include the search area are written to the *search_area_RAM* starting from line-1 to line-M. In this figure the dimensions of the search area is MxM. Meanwhile, the reference block lines which are read from

the reference image are written to the *reference_block_RAM*. Reference block is NxN.

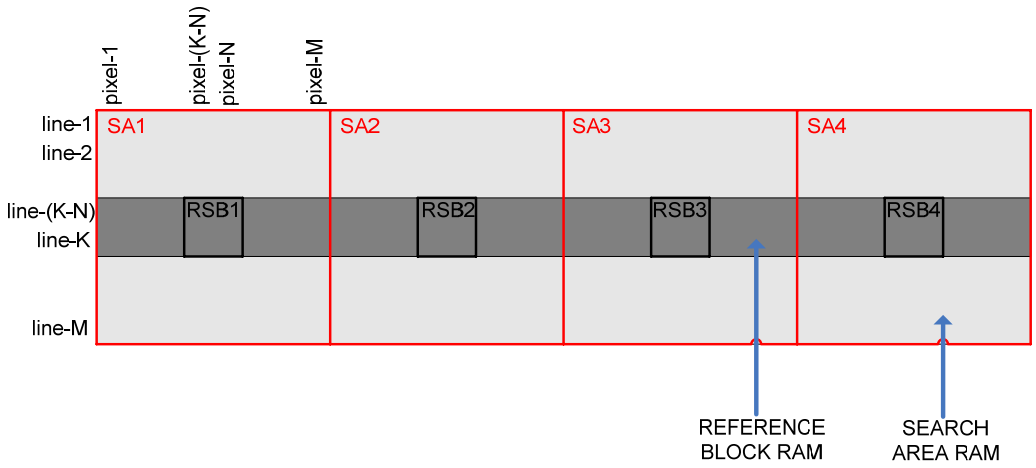

Figure 32: Reference Block & Search Area RAMs

The internal rams are reusable, after all pixel data is processed, the rams are filled with the new reference and search lines. The *VST_main_controller* is responsible to arrange this data flow. It updates the line numbers that are used for the SDRAM read operation, so when the new data is required, the video read interface switch to the new pixel data region. This process continues until all the reference and search regions are finished. The main controller waits for the end of the frame; then, it starts reading from the first region again. Figure 33 shows the data flow from SDRAM to internal RAMs.

The *full_search* block is responsible for the calculations of MAD, TA and shift values. It starts the operations by the assertion of the ref_data_ready and search_data_ready signals from the *VST_main_controller*. During the calculations cal_status signal remains at high state to indicate that the calculations continue, when all calculations are finished this signal goes to low state with the calculated values. The block consists of *FS_main_controller*, *texture_analysis*, *current_FIFO*

and *comparison* sub-blocks. The internal structure and the connections between the sub-blocks are shown in Figure 34.

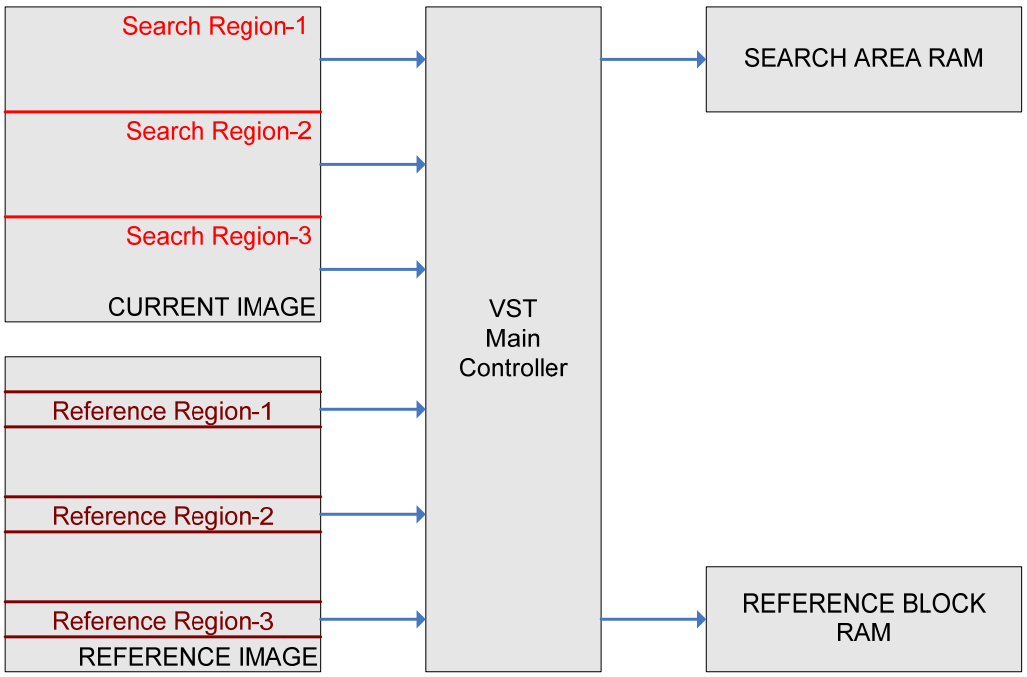

Figure 33: Data Flow from SDRAM to Internal RAMs

The *FS_main_controller* determine the internal ram addresses for the reference block and search area. As shown in Figure 32, the reference block data is located at different addresses of the internal ram, so the main control updates the ram addresses according to the reference block. After ready signals are asserted, main controller starts read operation. Firstly the reference pixel values are read and sent to the texture analysis and comparison block. Then, search area pixel values are read and written to the *current_FIFO*. This FIFO is used for the pipelining and it will be detailed while the comparison block is explained.

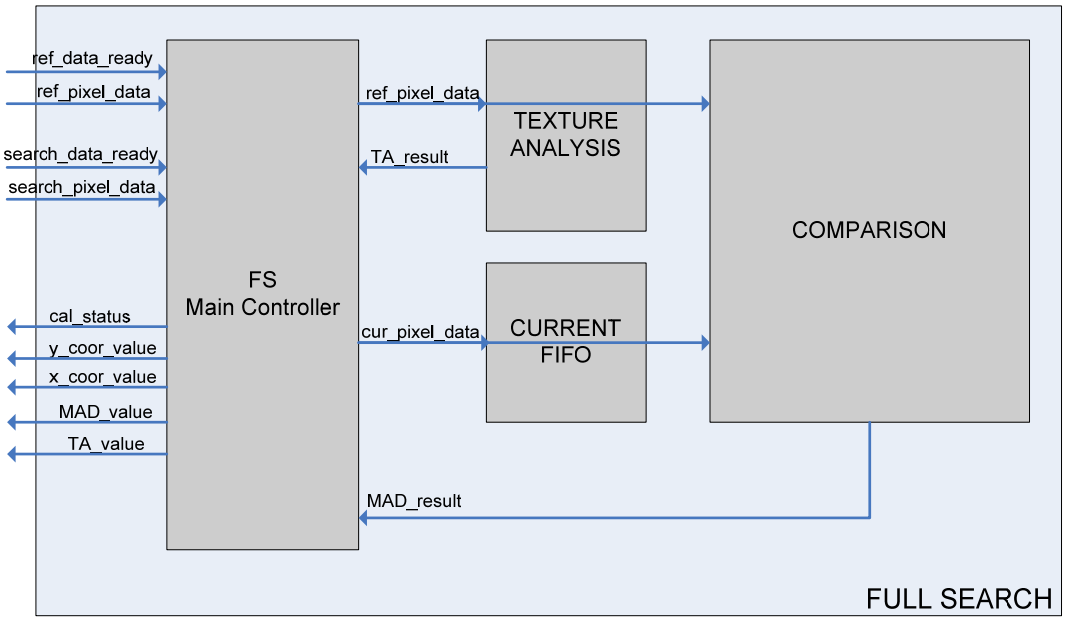

Figure 34: Full Search Block

Texture analysis is generally same with phase correlation with one difference. In full search only reference block texture analysis is calculated to be used in motion evaluation. The texture_analysis block is between the main controller and comparison block, so there is no need to create separate signals for the analysis. The texture analysis is realized during the data flow which enables the system to work synchronize.

Comparison block realize the mean absolute difference calculation between the reference and current blocks. This block contains N *row_compare* sub-blocks where N is the dimension of the reference block. Figure 35 shows the internal structure of the block. The absolute row differences are added and the MAD result is calculated.

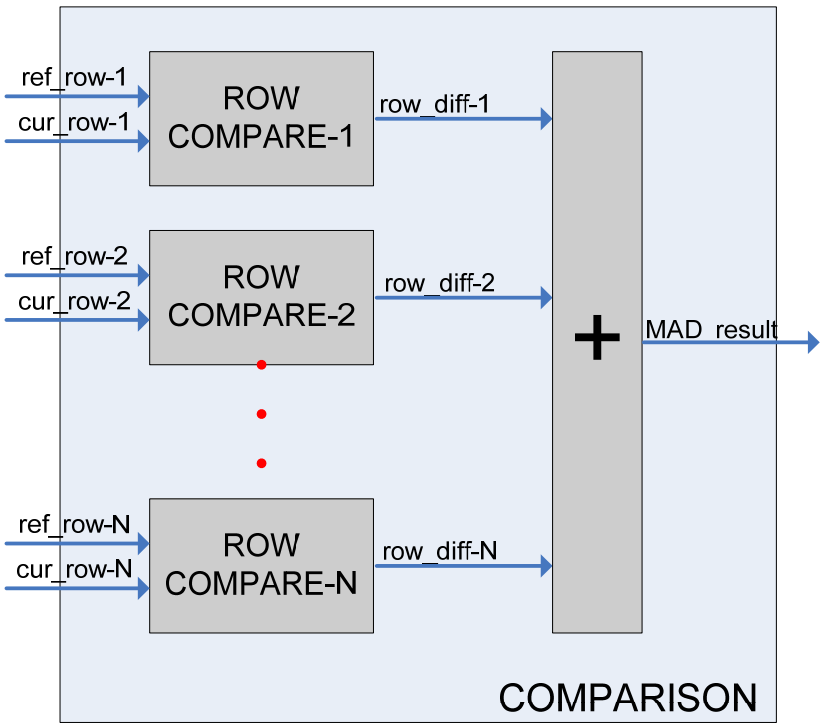

Figure 35: Comparison Block

Figure 36 shows the reference and current row pixel values; RSB (Reference Sub-Block), CSB (Current Sub-Block). Reference row values are loaded to *row_compare* bocks in line order. After all reference pixel values are loaded, the row_compare-1 has the first reference row, row_compare-2 has the second reference row and the other blocks hold the related row values.

The current sub-block row values are initially read from the *search_area_RAM* by *FS_main_controller* in row order. Main controller reads the search ram 1st line - 1st pixel, then it reads 2nd line -1st pixel and it continues until it finishes all 1st pixel read operation in the CSB-1(Figure 36). Then, the first pixels are combined and written to the *current_FIFO*. Main controller starts second pixel read operation; it reads all 2nd pixels in the CSB rows and writes them to the *current_FIFO*. The read operation is finished by reaching the last pixel location on the search area.

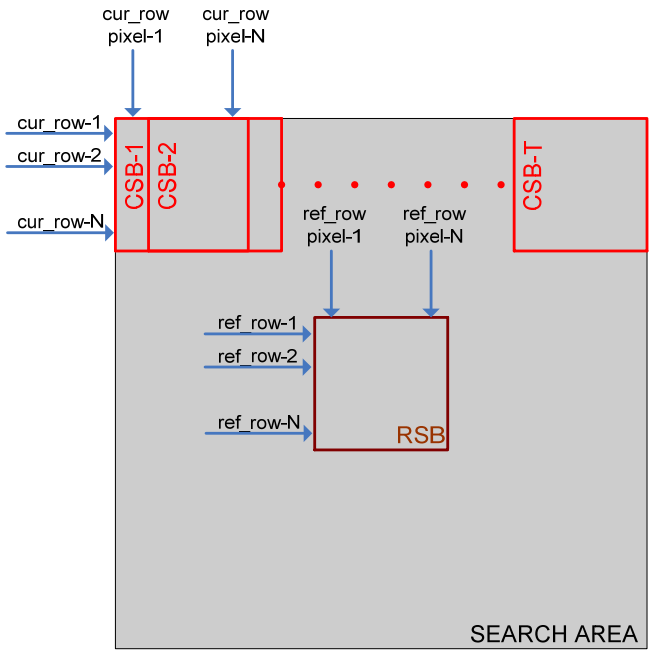

Figure 36: Reference & Current Block Pixel Value

Figure 37 shows the flow of the read operation. When the FIFO is filled, FIFO read operation starts and necessary current row pixel values are sent to the comparison block. After N read operation from the FIFO, all necessary pixel values are ready for the current and reference sub-block comparison.

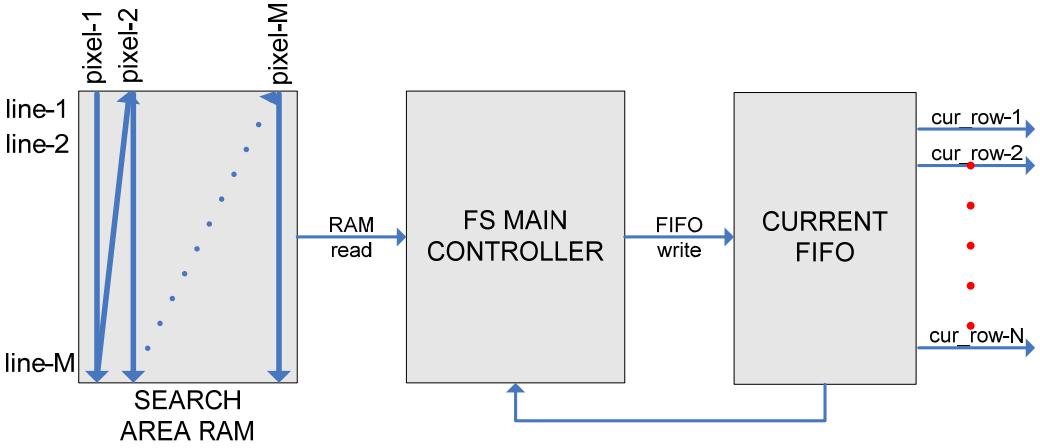

Figure 37: Search RAM Read Operation

Each row in the reference and current sub-block is compared with absolute difference operation. *Row_compare* block contains N absolute difference calculators for the pixel values. In Figure 38, absolute difference sub-blocks are shown. The connection between the absolute difference blocks behave like a shift register, so after CSB-1, CSB-2 is processed in a one pixel clock. This provides a pipeline structure which enables to finish all comparison between the search area and reference block in (M-N) x (M-N) clock.

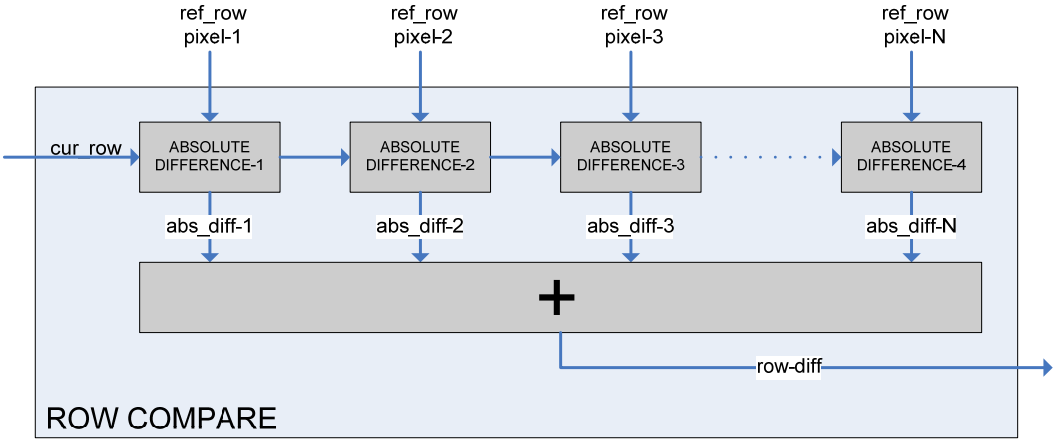

Figure 38: Row Compare Block Internal Structure

During the comparison between the reference and the current sub-blocks, *FS_main_controller* calculates the minimum MAD value and store the x_shift, y_shift and TA values. At the end of the comparison between the reference block and search area, the main controller deassert cal_status signal and sent the minimum MAD, TA, x_shift and y_shift values to the *VST_motion_evaluation* block.

Time domain motion evaluation will mainly used TA values for the evaluation because high TA results in correct MAD calculations. In this implementation, the five highest TA value will be selected. Among these selected values, three lowest MAD values are selected as the final results. The x_shift and y_shift values are the average of these final results.

Frame correction operation is similar with phase correlation. The coordinates of the required lines and pixels are updated according to the calculated evaluated shift values as shown in Figure 30.

CHAPTER 4

IMPLEMENTATION RESULTS AND COMPARISONS

The frequency and time domain digital video stabilization methods are evaluated according to accuracy, computation time, logic usage and power consumption. The accuracy test of the methods firstly realized in MATLAB and the results of this test determine the FPGA implementation. The accuracy comparison between MATLAB and FPGA results are obtained by VHDL simulation. Finally, FPGA comparison results of the selected methods are evaluated on real-time working hardware with PAL video.

This chapter is composed of three sub-sections. Firstly, the MATLAB results of phase correlation and full search algorithms will be explained. Then, the VHDL simulation results will be detailed. Finally, FPGA implementation results will be explained.

4.1 MATLAB IMPLEMENTATION

There are different parameters in stabilization methods such as block size, block location, texture analysis (TA), MAD (Mean of Absolute Difference), peak surface analysis. To see the effect of these parameters on the accuracy, the video stabilization methods are firstly implemented with MATLAB programming tool. In this implementation, a new function library is created to make the calculations similar with FPGA. In MATLAB, the numbers are in double-floating format;

however FPGA works with fix point numbers. Therefore, the floating structure of MATLAB is converted to fix point to make the calculations similar with FPGA.

The initial algorithm tests are done by synthetically shifted images to see that algorithms work properly. These tests provide 100 % accuracy for both methods so they will not be detailed since they do not provide a comparison data.

The comparison tests are conducted with real video sequence which is captured by a 640 x 480 camera, this resolution is similar with PAL format. Since, the shifts between the frames are real; the results are more similar with real-time working system. The accuracy of the method is evaluated according to pSNR (Peak Signal to Noise Ratio-dB) between the reference and corrected current image. In the following equations, *RI* is the reference image, *CCI* is the corrected current image and MxN is the image resolution.

$$Error = \sum_{y=1}^{M} \sum_{x=1}^{N} RI(x, y) - CCI(x, y) \tag{4.1}$$

$$MSE = \frac{Error}{M * N} \text{ (Mean Square Error)} \tag{4.2}$$

$$pSNR = 10 * \log_{10}(\frac{255^2}{MSE}) \tag{4.3}$$

The phase correlation MATLAB test results which are obtained by different parameter values will be explained first. Then, the results of full search method will be given. At last, the best results of these two methods which are obtained by optimum parameter values will be compared.

4.1.1 Phase Correlation Results

The first parameter that affects the phase correlation motion estimation accuracy is the block size. Figure 39 shows the different block sizes on the reference and current image and Table 6 gives the phase correlation results with these block sizes. In this table, TA is the texture analysis result, Peak Value is the magnitude of the highest peak on the surface and HPAR is the ratio of peak value to peak surface average.

Table 6 : Phase Correlation Results & Block Size

Block Size	8x8	16x16	32x32	64x64	128x128	256x256
X shift	1	1	1	0	0	2
Y shift	1	1	1	6	6	7
TA	216	503	1479	5022	51287	344247
Peak Value	0	0	0	15	22	27
HPAR	NaN	NaN	NaN	8.6	22.7	35.3
pSNR	19.2	19.2	19.2	20.8	20.8	22.5
Computation Time (s)	0.11	0.13	0.18	0.3	0.9	4.6

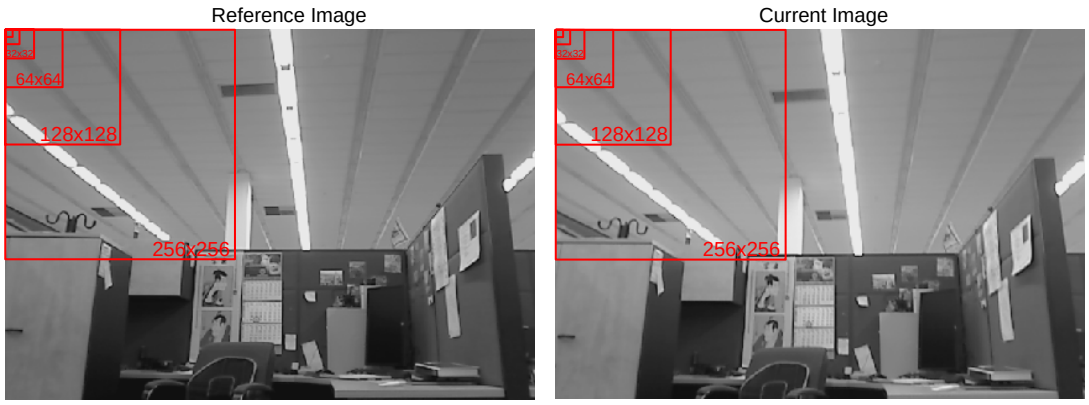

Figure 39: Block Size & Texture Variation

The texture in the selected block directly affects the quality of estimation. The increase in the block size also increases the texture. Therefore, the rise in texture results in better estimation and also higher pSNR. As shown in Figure 40, the low texture produces a wavy surface that contains many peaks. Some higher peaks have similar magnitude values and they cause faulty estimation.

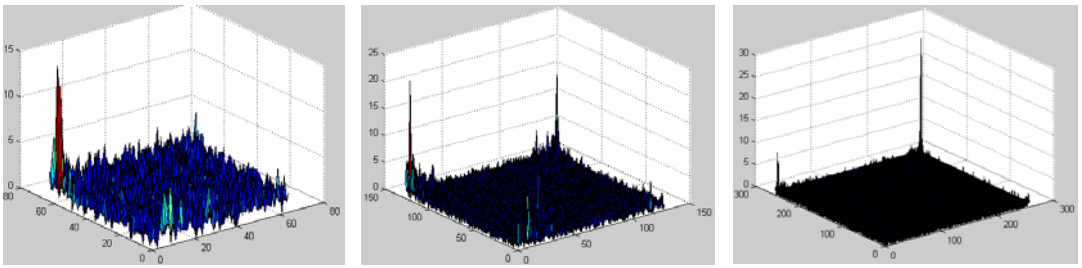

Figure 40: Peak Surfaces at Low Texture

The increase in the block size may be a solution for the texture problem but the increase in the block size requires longer computation time. This is a very critical issue for real time applications. Therefore, block size increase is a not a suitable option for this implementation.

The second parameter that affects the phase correlation motion estimation accuracy is the block location and the number of used blocks. Figure 41 shows totally 9 blocks which are located at different parts of the image. In this test the block size is 64x64. This size is selected, because its time-pSNR performance is better than others and this size can contain enough texture for the following tests.

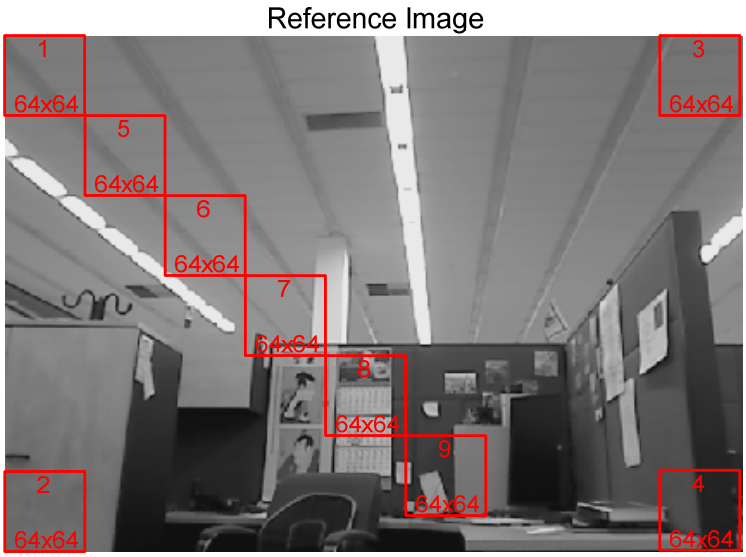

Figure 41: Different Block Locations

Table 7 : Phase Correlation Results & Block Location

Block Location	1	2	3	4	5	6	7	8	9
X shift	0	23	0	4	0	0	2	2	3
Y shift	6	0	6	4	6	0	6	6	5
TA	5022	4468	4199	22519	5113	7842	53049	61410	48670
Peak Value	15	8	21	14	14	21	28	20	29
HPAR	8.6	4.5	12.2	8.1	8.1	12.4	16.9	11.7	17.4
pSNR	20.8	15.4	20.8	20.4	20.8	18.6	23	23	22.3
Time (s)	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3

Phase correlation method is applied for all 9 sub-blocks and Table 7 gives the results. The PC results show that changing the block location can be an alternative to increase the texture. Also, using many sub-blocks provide opportunity to evaluate different calculation results. However, high block number needs longer computation time which can be also a problem for real time.

Another approach is to find the sub-block that contains the highest texture. Firstly, all possible sub-blocks in the image are analyzed and texture surface is obtained as shown in Figure 42. Then, the phase correlation starts at the highest texture location on the texture surface. Table 8 shows the calculation results for that location.

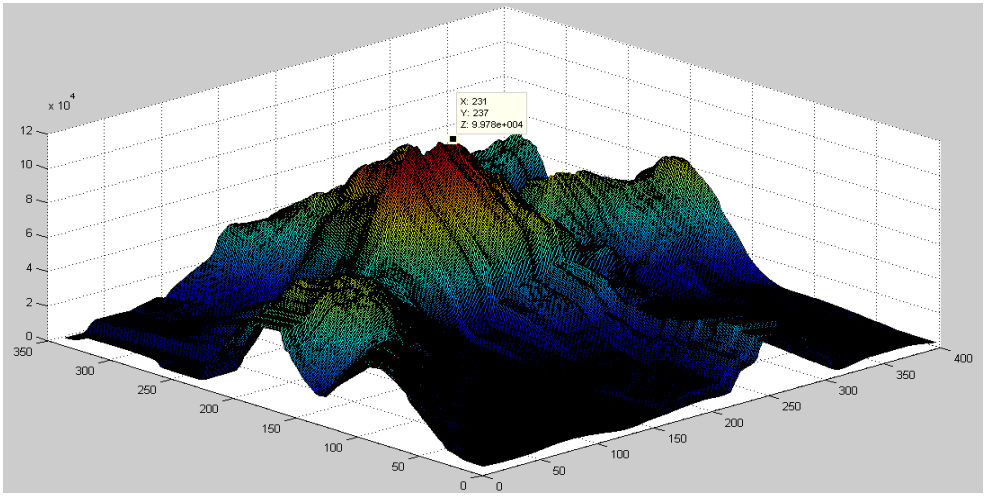

Figure 42: Texture Surface

Table 8 : Video Stabilization Results with Highest TA Block

	X shift	Y shift	TA	Peak Value	HPAR	pSNR	Time (s)
Highest TA Block (64x64)	**2**	**6**	**99775**	**27**	**16.1**	**23.08**	**85.7 85.4 : TA, 0.3 : VS**
Highest TA Block (128x128)	**3**	**6**	**252819**	**24**	**24.2**	**22.7**	**78.8 77.8 : TA, 1 : VS**

The texture surface analysis test results show that only texture is not an enough parameter for the estimation quality, because the pSNR result is similar with some blocks in Table 7, even though these blocks contain less texture. Also, the construction of the texture surface requires quite long time which is not feasible for real-time.

Also, other parameters such as peak surface analysis of the phase correlation can be used for the estimation. There can be pixel intensity differences between reference and current sub-block due to blurring and environmental changes; as a result the peak surface may contain many higher peaks even in high texture. Figure 43 shows the same pixels on the reference and current image which have different intensity values. Peak characteristics of the surface give information about the quality of the estimation. In Table 7, the higher peak and HPAR values give the higher pSNR results. Therefore, peak surface analysis is an alternative opportunity for the estimation.

Figure 43: Pixel Base Comparison from the Images

For accurate analysis, there are two opportunities. The first one is implementing a pre-analysis step like texture surface and using less sub-block. Other one is using several sub-blocks and obtaining many data from those blocks. The first opportunity is not suitable for real-time but the parallel structure of the FPGA can handle many sub-blocks in real time constraints. Therefore, the diagonal sub-blocks which are 1, 5, 6, 7, 8 and 9 in Figure 41 are used for the calculations. As a result, there are 6 blocks that generate texture, peak surface information. The phase correlation results of these sub-blocks are evaluated according to the methods described in section 2.2, Table 3.

In WOTT (without texture threshold), all the x & y shift results are used. The average of the x & y shift results are used as the evaluated results. In WTT (with texture threshold), firstly a texture threshold is determined. This threshold is the average of all texture results from the sub-blocks. Then, the x & y shift values of the sub-blocks whose texture is greater than the threshold are used for the final results calculation. In HP (highest peak), the sub-block which has the highest peak value is found and its x & y shift values are used. In WPT (with peak threshold), firstly, the sub-blocks that can pass texture threshold are found. Then, the average of the highest peak values of these sub-blocks is selected as the peak threshold. The final results are obtained from the average of the sub-blocks whose highest peak values are greater than peak threshold. HPAR is the average of the highest peak to the surface average. The sub-block with the highest HPAR value gives its x & y shift values as the final results. Table 9 shows the evaluation results for the described methods on reference and current frame in Figure 39.

The methods that are related with the peak surface analysis provide higher pSNR values. In this test, WTT produce the best pSNR but the test is conducted with only one frame pair and one frame comparison result can not be generalized. Therefore the tests are conducted with a frame sequence that contains 100 frames.

Table 9 : PC Motion Evaluation Results

	X shift	Y shift	T_T	Peak Value	T_P	HPAR	pSNR	Time (s)
WOTT	**1**	**5**	**NA**	**NA**	**NA**	**NA**	**21.6**	**1.6**
WTT	**2**	**6**	**30184**	**NA**	**NA**	**NA**	**23**	**1.6**
HP	**3**	**5**	**30184**	**29**	**NA**	**NA**	**22.3**	**1.6**
WPT	**3**	**6**	**30184**	**NA**	**25.6**	**NA**	**22.7**	**1.6**
HPAR	**3**	**5**	**30184**	**NA**	**25.6**	**17.4**	**22.3**	**1.7**

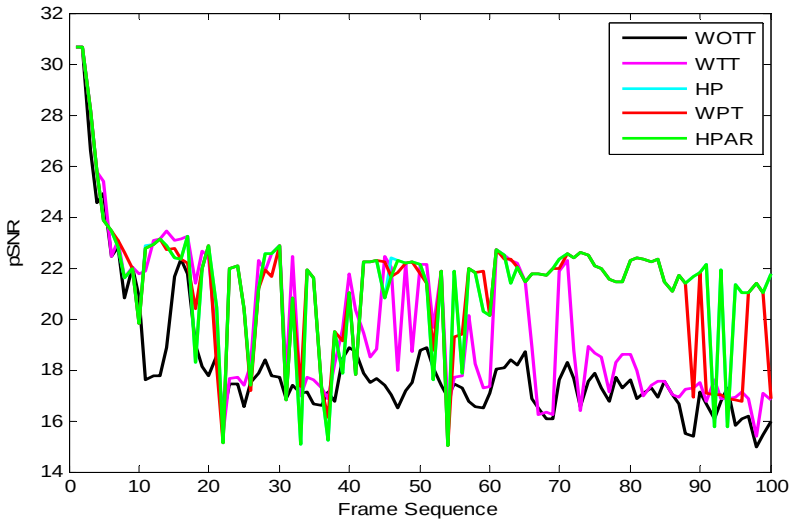

Figure 44: PC Motion Evaluation Results on Frame Sequence (Constant Reference Frame)

Table 10: Average pSNR and Computation Time Results for PC Motion Evaluation Methods on Frame Sequence (Constant Reference Frame)

	WOTT	**WTT**	**HP**	**WPT**	**HPAR**
pSNR Average	**18.1**	**19.8**	**21.4**	**21.2**	**21.4**
Total Time (s)	**153.6**	**149.3**	**150.1**	**149.7**	**150.5**

Figure 44 shows the pSNR evaluation results for the frame sequence according to different methods and Table 10 gives the average pSNR and computation time. In this evaluation, reference frame remain same and other frames are corrected according to it. The average pSNR values show that peak analysis provides better stabilization performance. There are some rapid drops on the figure, this drops are due to the blurring effect on the image, even the motion estimation is done correctly the pSNR calculations give lower values since there are various intensity differences between reference and corrected current image. Other studies show that

usual pSNR values are around 20-25 dB [26], but for a good stabilization the pSNR should reach 35dB [27].

For the mobile cameras, the reference image should be updated with the corrected current image. Figure 45 shows the pSNR results for the updated reference image approach. Table 11 shows that pSNR values are higher than the constant reference image model. This is logical because neighbour frames have similar intensity variations.

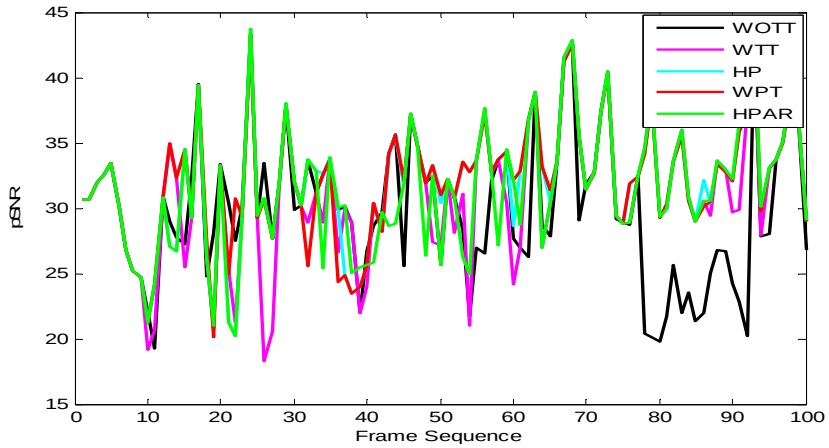

Figure 45: PC Motion Evaluation Results on Frame Sequence (Updated Reference Frame)

Table 11: Average pSNR Results for PC Motion Evaluation Methods on Frame Sequence (Updated Reference Frame)

	WOTT	**WTT**	**HP**	**WPT**	**HPAR**
pSNR Average	**29.8**	**31.1**	**31.9**	**32.1**	**31.4**

The frame sequence test shows that WPT has the highest pSNR performance. The peak surface analysis provides valuable information for the estimation quality.

Therefore in FPGA implementation peak surface analysis is used to reach acceptable stabilization performance.

4.1.2 Full Search Results

This sub-section explains the full search MATLAB test results. The first parameter for the full search algorithm is the reference block size. The size of the block is important for the texture analysis which in turn gives idea for the quality of the motion estimation. Figure 46 shows the reference block and the search area. For different reference block sizes, search area is 8 pixels bigger in all direction in the first experiment.

Figure 46: Reference Sub-Block and Search Area

Table 12 : Full Search Results & Block Size

Block Size	8x8	16x16	32x32	64x64	128x128	256x256
X shift	-8	-8	-8	-8	-8	2
Y shift	8	8	5	-8	-1	7
TA	131	382	1399	4970	51014	364964
MAD	4.6	9.2	12.7	13.1	14.3	15.7
pSNR	17.1	17.1	17.4	16.2	16.9	22.5
Computation Time (s)	0.05	0.06	0.09	0.15	0.3	0.9

Table 12 gives the Block-Size & pSNR test results; the increase in block size increases the texture and pSNR value. But, even the block size is around 128x128 the pSNR is less than 20dB. The reason for the low pSNR is the location of the reference block. Since, there are low texture around, increasing the block size do not provide higher texture.

To solve texture problem, full search algorithm is implemented with several reference blocks and search areas. In the original full search algorithm every neighbor reference block is checked, but for FPGA implementation, it is not feasible with the available resources. So, the method's main idea is kept same and only the neighbor search areas are checked. Figure 47 shows the search areas which are 35 in total, on the image.

Figure 47: Search Areas on the Image

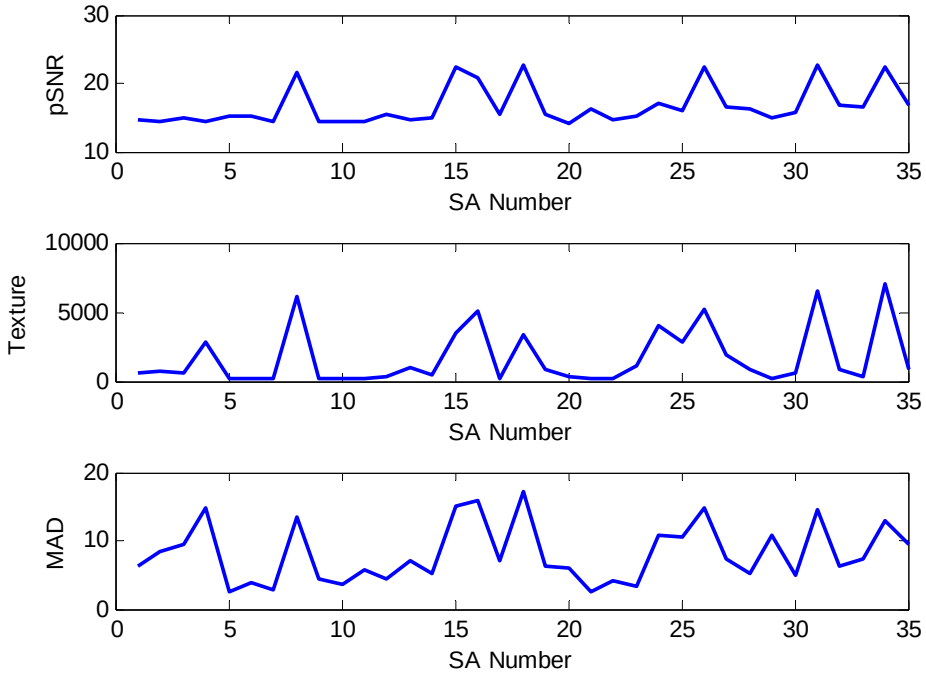

Figure 48: pSNR & Texture & MAD of Search Areas

Figure 48 shows the pSNR, texture and MAD (Mean of Absolute Difference) values for the numbered search areas. The pSNR values are higher at the locations where texture and MAD are both high. The higher pSNR values for higher texture are predicted results, but MAD values should be lower for the correct match. However, if we examine the MAD results, the average is small and the maximum value is around 20. So, it can be said that the higher texture results in higher MAD results.

The full search results with constant reference frame sequence are shown in Figure 49 and Table 13 gives the average pSNR and computation time. MAX_TA assigns the x & y shift values of the block which has the highest texture to the final values. MIN_MAD finds the block with minimum MAD value and gives its shift values as the final values. MAX_TA_AVG gets the average of the 5 highest texture blocks shift values. MAX_TA_MIN_MAD finds the 5 highest texture blocks and assign the shift values from the minimum MAD block among the highest ones.

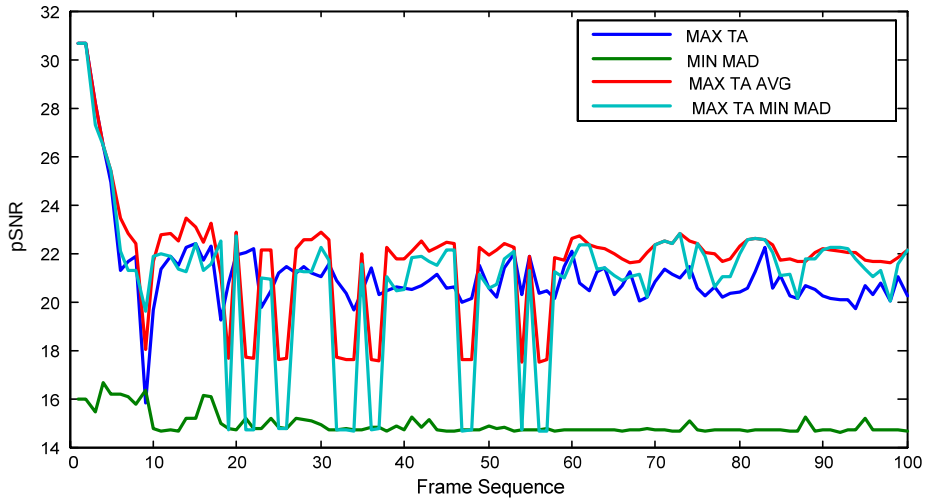

Figure 49: FS Motion Evaluation Results on Frame Sequence (Constant Reference Frame)

Table 13: Average pSNR and Computation Time Results for FS Motion Evaluation Methods on Frame Sequence (Constant Reference Frame)

	MAX TA	MIN MAD	MAX TA AVG	MAX TA MIN MAD
pSNR Average	21.5	14.9	21.7	20.8
Total Time (s)	1089.7	1133.3	1103.7	1092.7

The average pSNR results show that, MAX_TA_AVG gives the best stabilization results. This is logical when we compare the texture-estimation relation from Figure 48. In full search algorithm, high texture prevents mismatches and finds the correct estimation results. The down peaks on the graph are due to intensity changes between the frames which are occurred due to blurring and environmental changes.

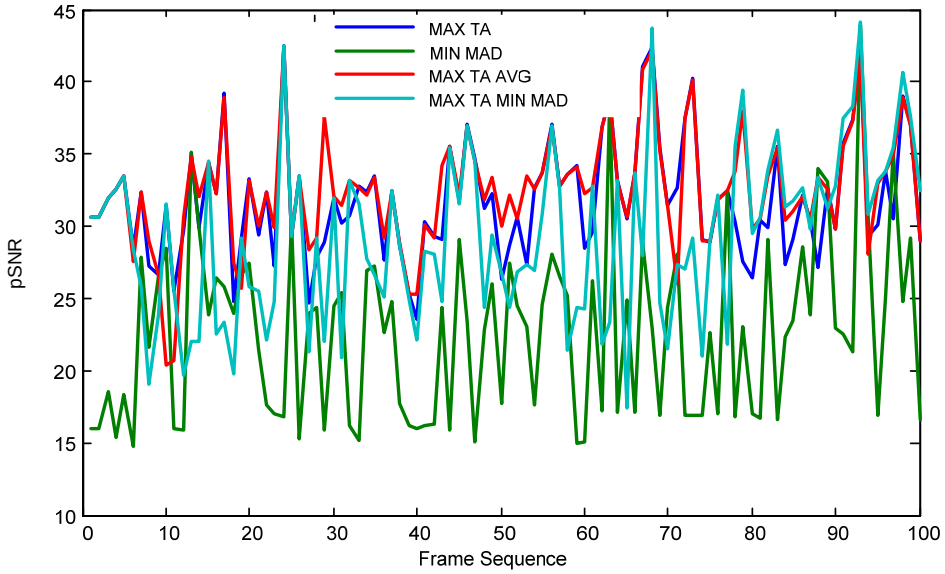

Figure 50: FS Motion Evaluation Results on Frame Sequence (Updated Reference Frame)

Table 14: Average pSNR Results for FS Motion Evaluation Methods on Frame Sequence (Updated Reference Frame)

	MAX TA	MIN MAD	MAX TA AVG	MAX TA MIN MAD
pSNR Average	31.7	22.6	32.4	29.2

Figure 50 shows the pSNR results with updated reference block and the average pSNR values are given in Table 14. As in phase correlation, also in full search, updated reference block approach gives higher pSNR when compared with constant model. The MAX_TA_AVG method again provides the highest pSNR; this test shows that MAX_TA_AVG method is the best approach for full search method. In FPGA implementation, with texture analysis also the MAD values will be used to find the sub-blocks which are free of local motion. Firstly, the texture analysis is used to determine the correct sub-blocks then MAD analysis will be used for local-global motion analysis. The comparison between PC and FS will also contain the local motion analysis which is obtained from the frame sequences that contains local motion.

4.1.3 Comparison Between Phase Correlation and Full Search

The previous sections give the stabilization results of phase correlation and full search algorithms on 100 frame sequence. According to the pSNR results, the MAX_TA_AVG full search approach reaches 32.4 dB pSNR which is the highest value among all approaches including phase correlation. In phase correlation, WPT approach gives 32.2 dB pSNR, it is the highest value in phase correlation. When the computation time results are examined in Figure 44 and Figure 49, the WPT

approach completes the calculations in 149.7 seconds; on the other hand MAX_TA_AVG approach uses 1103.7 seconds for the calculations. The phase correlation time performance is nearly ten times better than full search, so the accuracy-time ratio which is very critical in real time implementations; is higher in phase correlation.

The previous tests are conducted with local motion free frame sequences, therefore in the following comparison tests there will be local motions as shown in Figure 51. The local motions may result in faulty estimations and this problem decrease the pSNR performance.

Figure 51: Frame Sequence with Local Motion

The frame sequence is composed of 200 frames which in turn provides more comparison data between phase correlation and full search. Three approaches from phase correlation and also from full search are used. These approaches are the most successful ones according to pSNR results. For phase correlation, HP, WPT and HPAR and for full search, MAX_TA, MAX_TA_AVG and MAX_TA_MIN_MAD approaches are used.

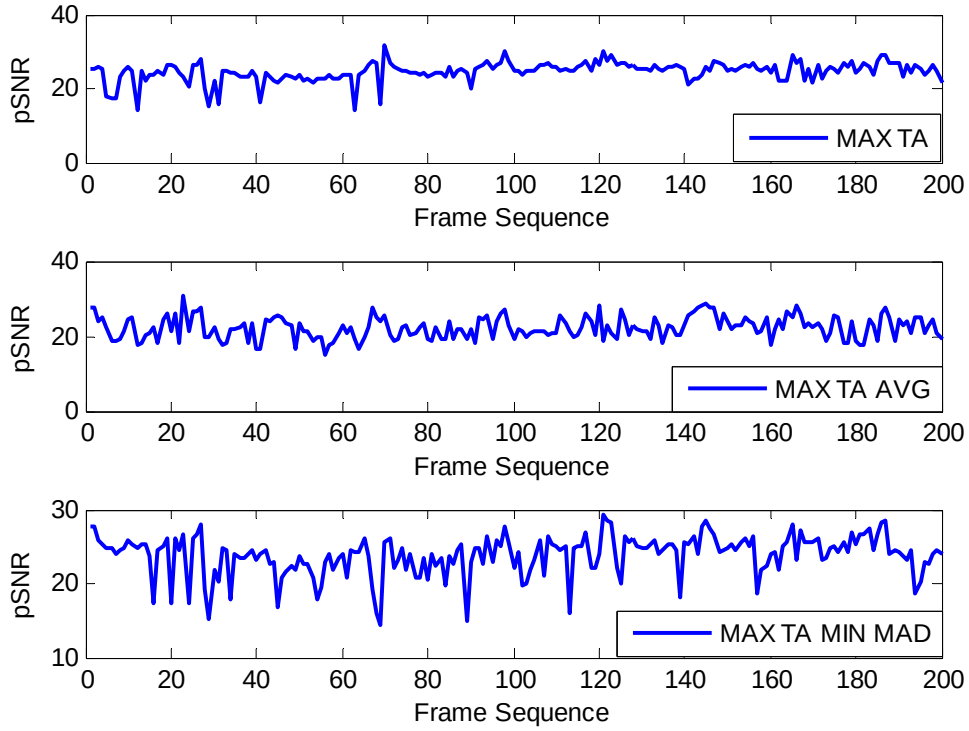

Figure 52: Full Search Approaches Results with Local Motion

Table 15: Average pSNR and Computation Time Results for Full Search Approaches with Local Motion

	MAX TA	MAX TA AVG	MAX TA MIN MAD
pSNR Average	24.6	22.2	23.8
Time(s)	2183	2172	2298

Figure 52 shows the full search approaches' pSNR values with 200 frame sequences that contain local motion. Table 15 gives the average pSNR and computation time for the evaluation methods. The MAX_TA approach gives the highest pSNR average but when we examine the result plots, MAX_TA_MIN_MAD reaches high pSNR values. The rapid drops on the curve decrease its average pSNR. The blurring

effects prevent MAX_TA_MIN_MAD approach to give high pSNR average. Even though this method's average pSNR is not the maximum, it is suitable for stabilization on local motions. The rapid drops can be filtered in FPGA implementation by checking the previous results.

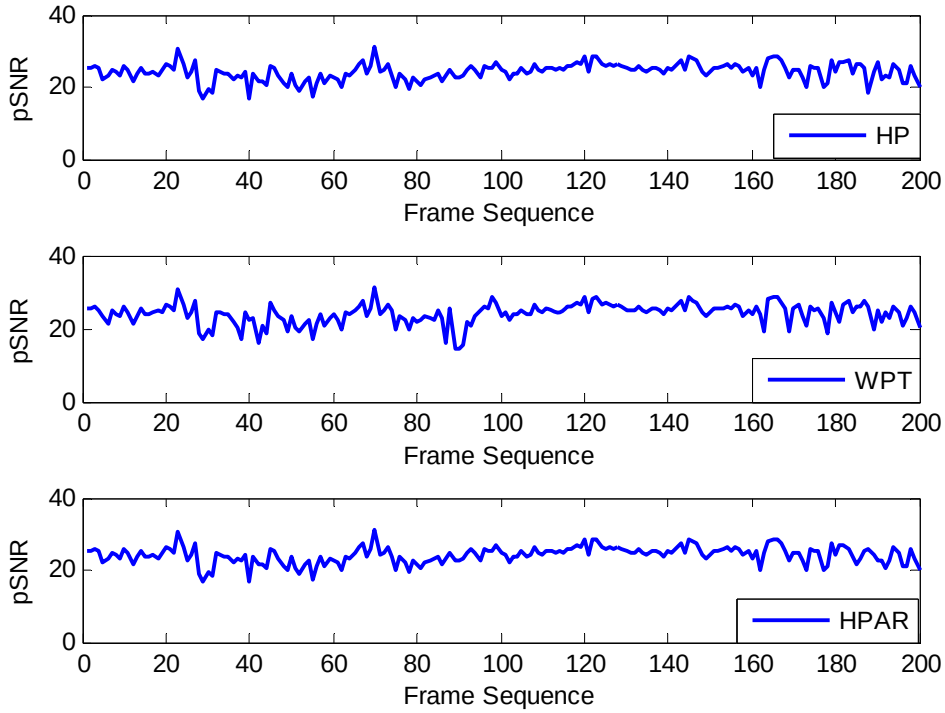

Figure 53: Phase Correlation Approaches Results with Local Motion

Table 16: Average pSNR and Computation Time Results for Phase Correlation Approaches with Local Motion

	HP	WPT	HPAR
pSNR Average	24.2	24.09	24.1
Time(s)	299.8	300.2	299.5

The phase correlation approaches give similar average pSNR values. HP approach provides the highest result. From the graphs in Figure 53, there are less rapid drops when compared with full search approaches and the average pSNR results of phase correlation approaches which is shown in Table 16, are similar and higher than full search.

As a result, phase correlation produces more consistent estimation results when compared with full search according to local motion test. Even though phase correlation pSNR results also contain rapid drops, the magnitudes of these drops are less than full search results. Therefore, it is possible to say, phase correlation can handle blurring and environmental changes more effectively. The MATLAB results show that phase correlation is better than full search method according to accuracy-time ratio, but the implementation structure of phase correlation is more complex than full search since it requires DFT/IDFT operations. So, the simulation and FPGA implementation results will show the final comparison results between the two methods.

4.2 FPGA SIMULATION

FPGA blocks are designed by VHDL code. The written code can be simulated in computer by Modelsim simulation tool. This tool shows the all signal flow as it is in FPGA. So, the results of this simulation give information about the performance of the FPGA implementation since the same code is downloaded to the FPGA.

The accuracy test of the written FPGA code is evaluated with simulation tool, because the real time working system may receive arbitrary frames and it is impossible to detect correct shift values. Since, the simulation uses the same VHDL code with FPGA, the simulation results can be used for accuracy comparison.

The simulation tool can receive frames as a text file, so the frames that are evaluated in MATLAB are converted to text files. The results of the VHDL simulations are observed on the signals which are described in section 3.2 and 3.3.

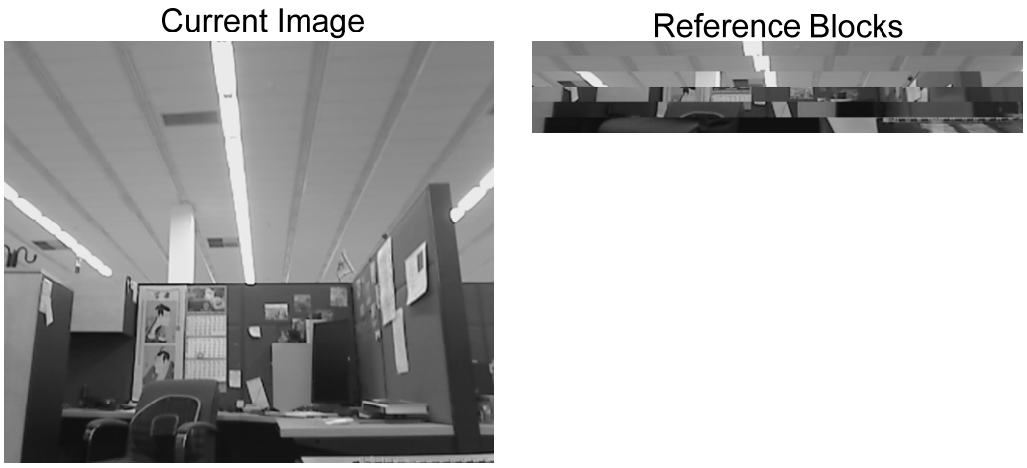

Figure 54: The Current Image and Reference Sub-Blocks

The full search simulation results will be explained firstly, and then phase correlation results will be detailed. Figure 54 shows the current image and the reference sub-blocks that are obtained from the reference image. This figure explains the methodology which is described in section 3.3 with Figure 32 and Figure 33.

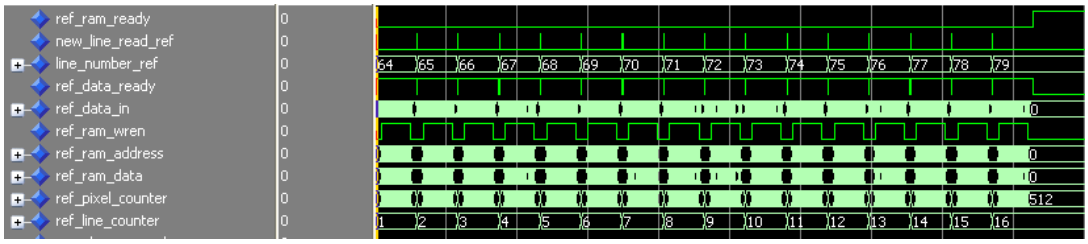

Figure 55: Full Search Reference Data Read

Full search FPGA implementation starts with the reference sub-blocks read operation from the SDRAM memory. 16 lines are read and sent to the comparison

blocks and also ref_ram_ready signal is asserted to inform the blocks that the read operation is completed. Figure 55 shows the signals and their states for the read operation. After reference sub-blocks, the search areas are read from the memory. Since, there are 80 lines for the search area, this operation takes longer time than the reference block read. In Figure 56, the necessary signals for the search area read operation are shown. At the end of the read operation search_ram_ready signal is asserted.

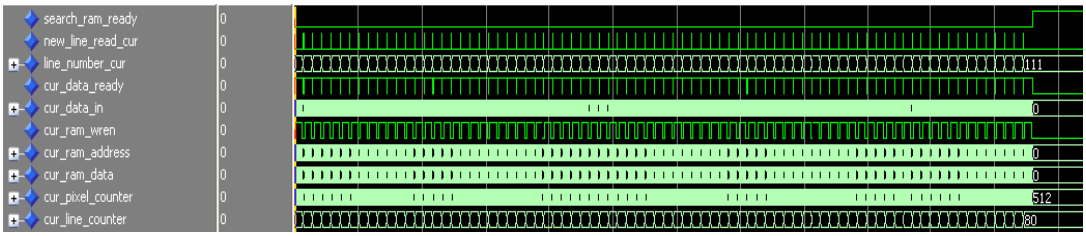

Figure 56: Full Search Current Image Search Area Read

With the completion of all necessary pixel read from the memory, the comparison operation starts. At every clock, a current sub-block and reference sub-block is compared and MAD value is calculated. During MAD value calculation also TA value is obtained from the reference sub-block. At the end of the calculations, the x_shift and y_shift values are sent to the evaluation block by cal_status signal. The shift values are 8 bit. The 8th bit is the sign; "1" means negative shift and "0" means positive shift. The MAD calculations and results are shown in Figure 57.

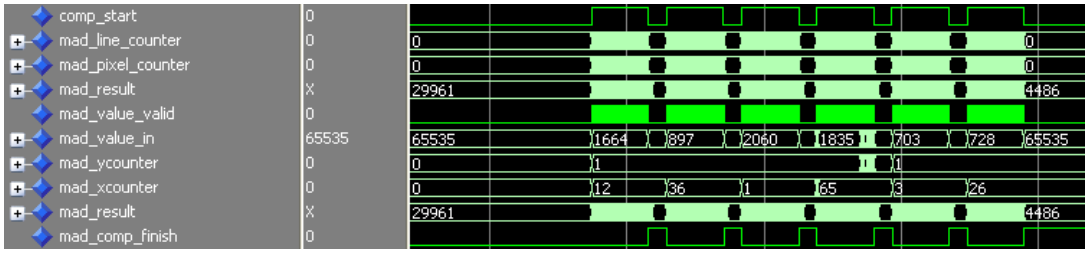

Figure 57: MAD Calculations and Results

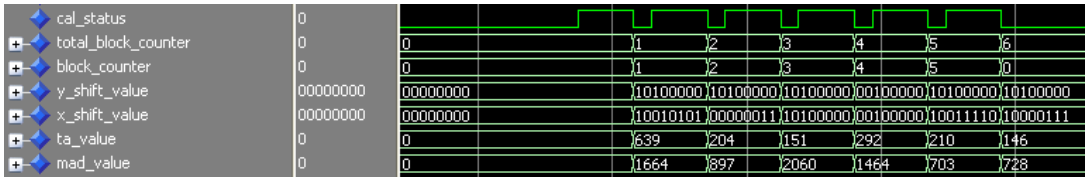

Figure 57: Continuation

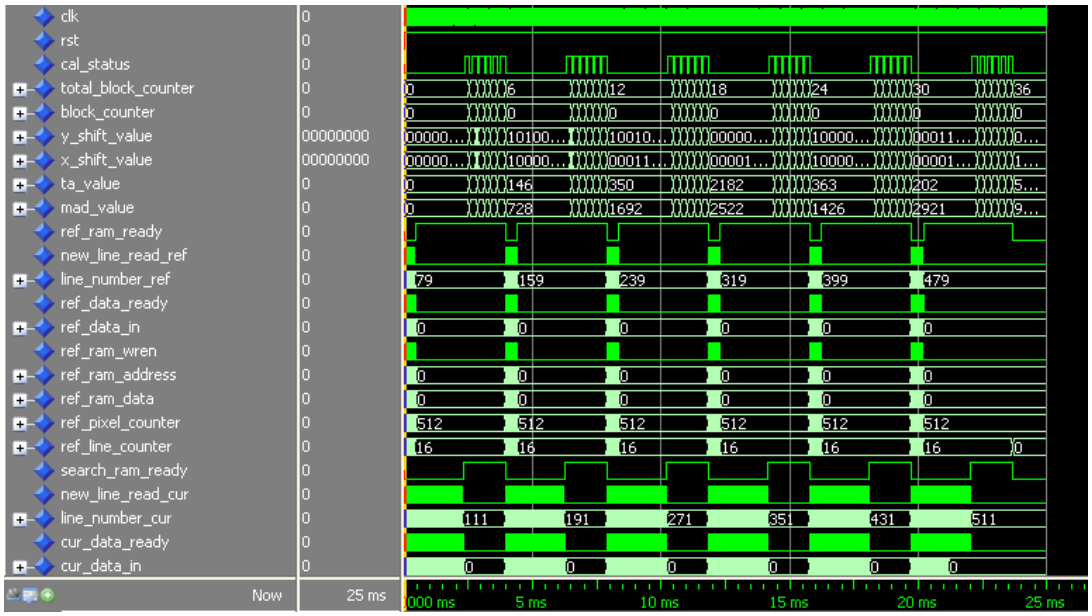

Figure 58: Full Search Calculation on a Frame

Figure 58 shows the all data flow for the full search method. All calculations finish in 25 milliseconds where it is approximately 10 seconds in MATLAB implementation. The results of the FPGA simulations are compared with MATLAB results. The obtained results are shown in Figure 59. According to the result plots, the FPGA implementation can reach the accuracy of MATLAB. The same accuracy is obtained in a shorter time, so the accuracy-time ratio of full search FPGA implementation is better than MATLAB.

After full search FPGA simulation results, phase correlation results will be explained. In this implementation, high number of sub-blocks approach is used because texture surface analysis requires longer computation time. Figure 60 shows the sub-blocks that are located on diagonal of the frame.

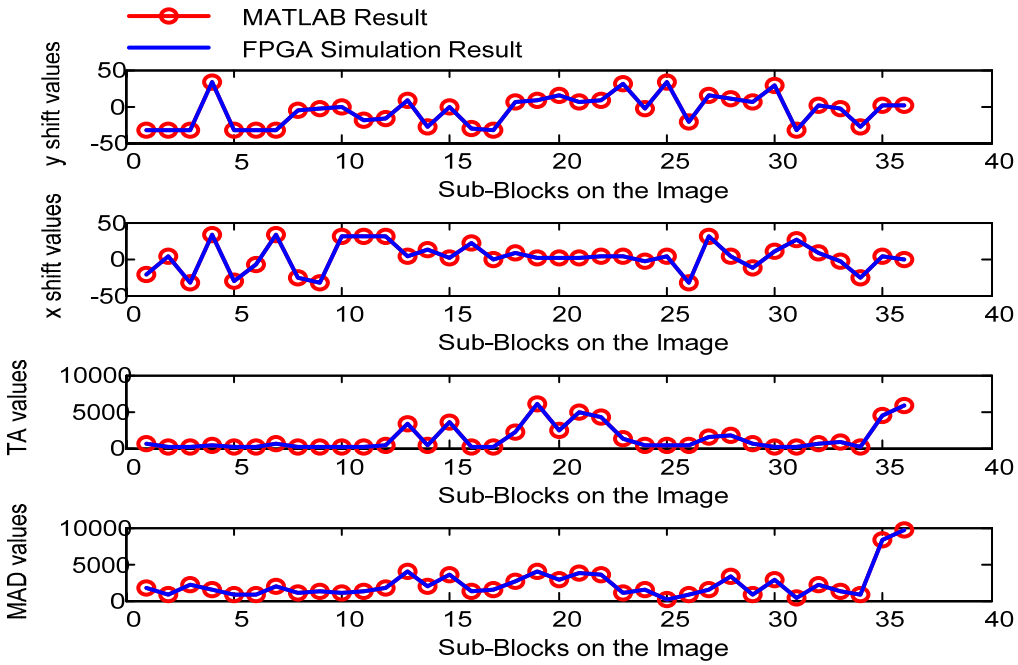

Figure 59: MATLAB & FPGA Full Search Comparison

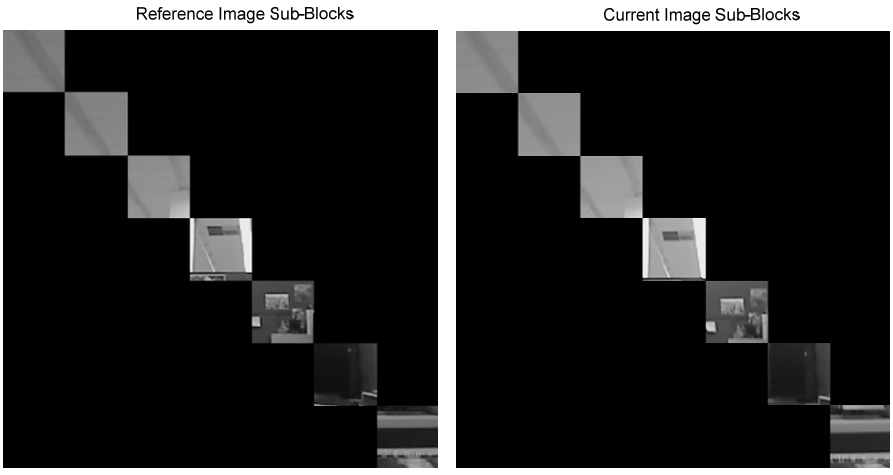

Figure 60: Phase Correlation Sub-Blocks on Reference and Current Images

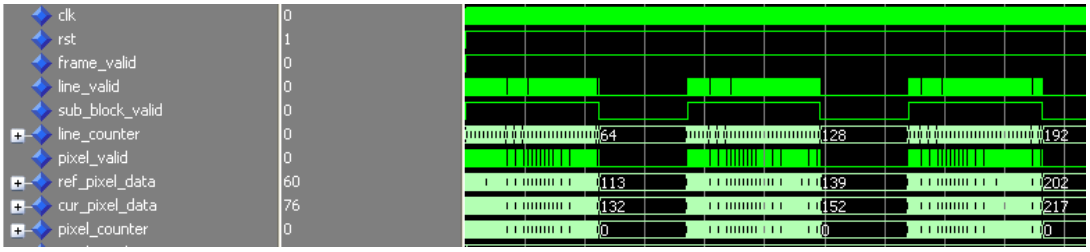

Figure 61: Phase Correlation Sub-Blocks Read Operation

Figure 61 shows the reference and current sub-blocks read operation from the SDRAM memory. The reference and current sub-block lines are sent to the phase correlation block for the DFT/IDFT operation.

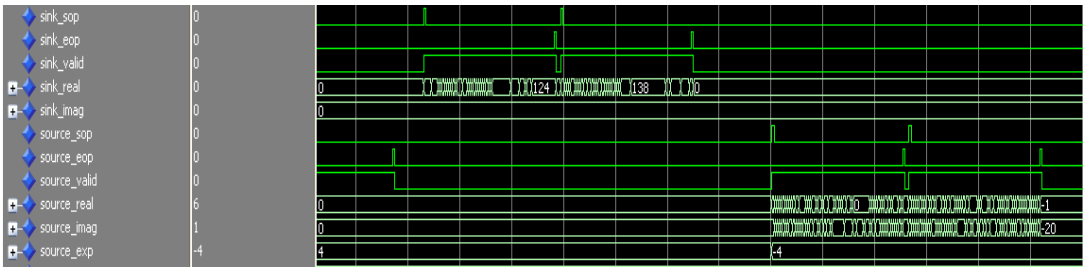

Figure 62: Phase Correlation DFT Operation

The reference and current sub-block lines are enter into the 1_D_fft block consecutively. The results of the fft operation are obtained after several clock cycles. The pc_main_controller block handles the wait time between the fft input and output. The fft operation is shown in Figure 62.

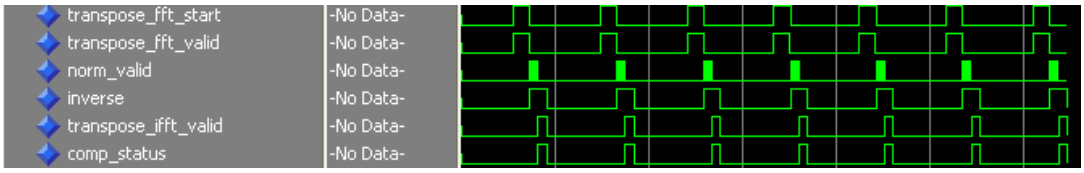

Figure 63: Phase Correlation Data Flow

In Figure 63, the all data flow of phase correlation implementation is shown. After DFT operation, transpose DFT operation starts. The calculated transpose DFT results from reference and current sub-blocks are normalized in normalization block. Then, inverse DFT (IDFT) operation and transpose IDFT operations are completed. During transpose IDFT operation, the calculated peak surface values are evaluated with comp_status signal.

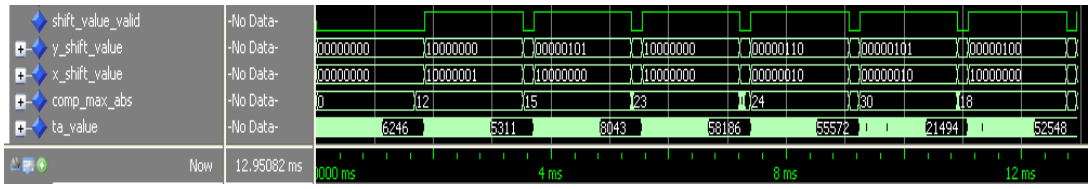

Figure 64: Phase Correlation FPGA Simulation Results

At the end of the operations, the shift values, highest peak value (comp_max_abs) and TA value are obtained and sent to the evaluation block by shift_value_valid signal. The relation between the signals is given in Figure 64. The formats of the shift values are same with full search. The phase correlation calculation for all sub-blocks takes 12.9 milliseconds while it is 2.1 seconds in MATLAB implementation.

The results of the phase correlation FPGA and MATLAB implementation are given in Figure 65. The FPGA simulation results are same with MATLAB's. Therefore it is possible to conclude that, in phase correlation method FPGA can reach the accuracy of MATLAB in a shorter time.

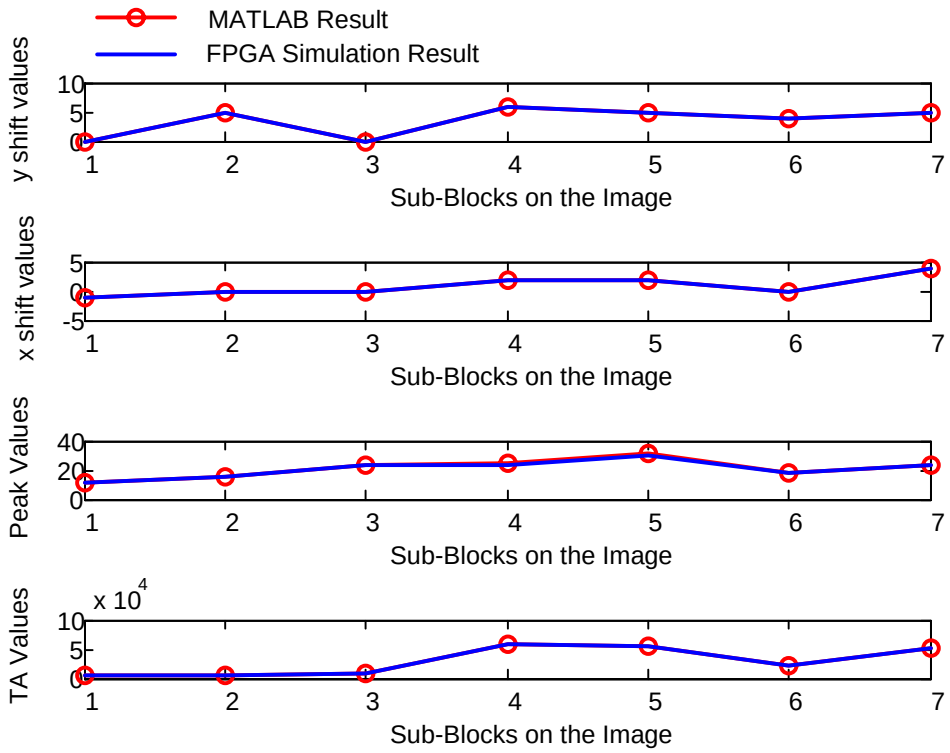

Figure 65: MATLAB & FPGA Phase Correlation Comparison

The simulation results show that the MATLAB performance comparison between the full search and phase correlation methods can also be used for FPGA implementation. In FPGA implementation section, two methods will be compared in terms of computation time on hardware, logic usage and power consumption. Also, the results of the real time FPGA implementation on the display while the camera is shaking will be explained.

4.3 FPGA IMPLEMENTATION

The FPGA blocks are designed with Altera Quartus II programming tool. The VHDL code is written by text editor, then the written VHDL is converted to schematic symbol for the connection with other blocks. Figure 66 shows the Quartus design environment. The blocks are connected to other blocks by internal signals. Design files section show the hierarchy in the implementation. In Figure 66, VST_FPGA_TOP_V1 is the top block.

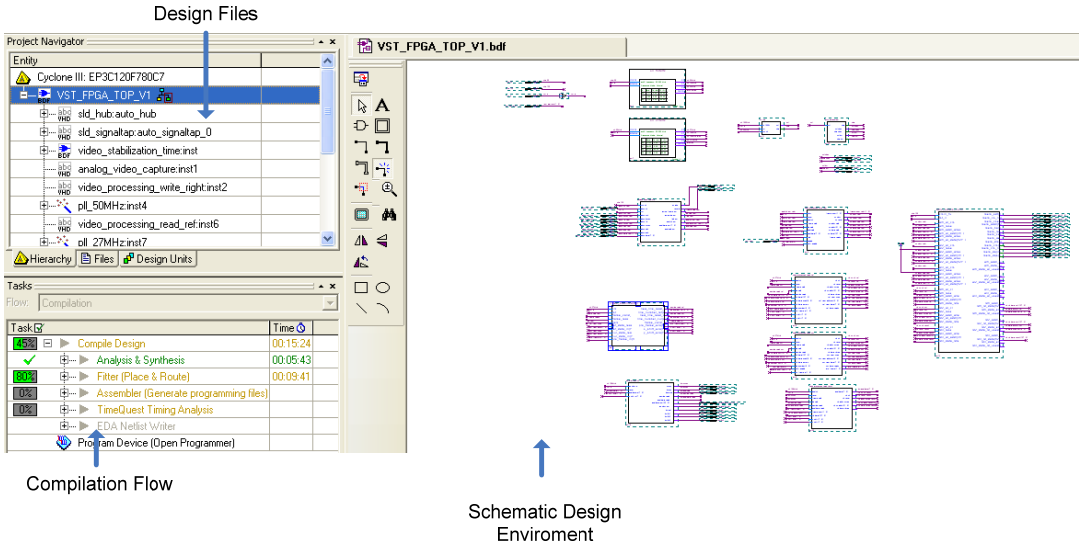

Figure 66: Altera Quartus II Design Environment

After all necessary blocks are coded and connecting to each other, the design is analysed for the coding errors. Then the necessary pin assignment is completed and full compilation flow is started.

During the compilation, the programming tool generates the routing in the FPGA. It connects the input/output pins to the related blocks. Also, the timing performance of the generated routing is tested according to the constraints that the user enters at start. At the end, this flow generates a bit file which is downloaded to the FPGA.

The full search and phase correlation methods are implemented in FPGA with the external blocks such as video and memory interfaces. The compilation report gives the logic, internal memory and embedded multiplier usage. The power consumption is measured from the main board. The computation time is measured by the time counter located in the FPGA code. Table 17 shows the compilation summary. In this table, the approximate time values are given, because the read operations from the SDRAM memory determines the computation time, and read operation time can be changed according to the memory usage by the other FPGA blocks. According to the tests on the FPGA, these time results are the highest observed values. These computation time results show that FPGA implementations of the digital video stabilization methods can finish the calculations less than one frame duration which is 40 milliseconds.

Table 17: FPGA Implementation Summary

	Computation Time (ms)	Power Consumption(W)	Logic Usage	Internal Memory Usage(Kbit)	Embedded Multiplier Usage(9x9)
Full Search	**~35**	**0.36**	**24,648**	**1788**	**0**
Phase Correlation	**~22**	**0.23**	**27,814**	**1888**	**44**

In sections 4.1 and 4.2, the results for different block sizes and search areas are examined. In FPGA implementation, these results are evaluated and the final block sizes are determined. Table 18 and Table 19 give the implementation summary for full search and phase correlation methods, respectively.

Table 18: Full Search FPGA Summary

	Reference Block Size	Search Area Size	Allowed x Shift (pixel)	Allowed y Shift (pixel)
Full Search	**16**	**80**	**+/- 32**	**+/- 32**

Table 19: Phase Correlation FPGA Summary

	Reference Block Size	Current Block Size	Allowed x Shift (pixel)	Allowed y Shift (pixel)
Phase Correlation	**64**	**64**	**+/- 32**	**+/- 32**

The methods are firstly tested by synthetic shifts which are generated by the push buttons on the main board. Different shift values are applied to current image and the estimated shift values are obtained by the "Signal Tap" tool of Altera. This tool monitors the signals while the codes are running in the FPGA. Figure 67 shows the synthetic shift values and the estimated ones on the signal tap screen.

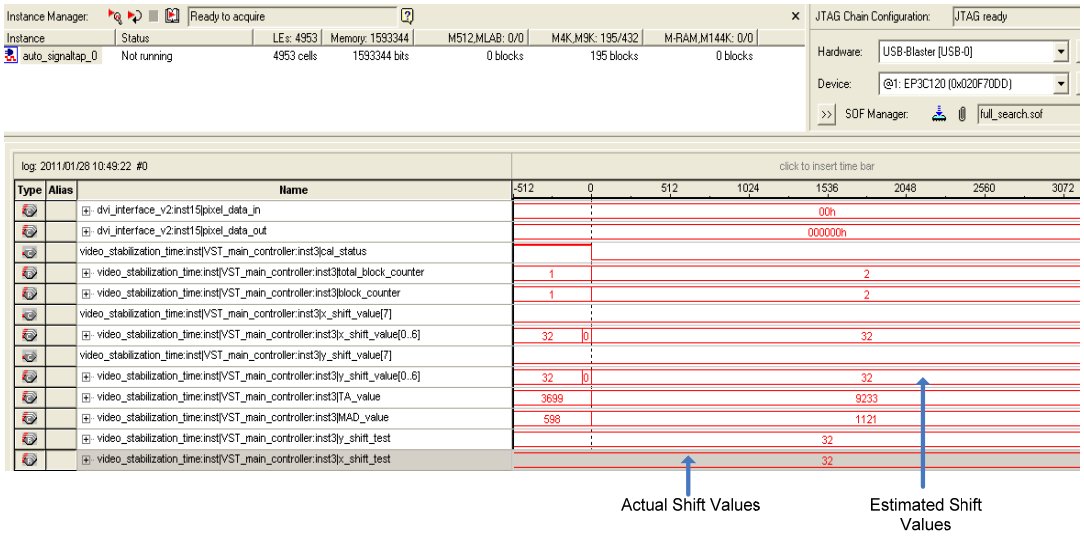

Figure 67: Altera Signal Tap Screen

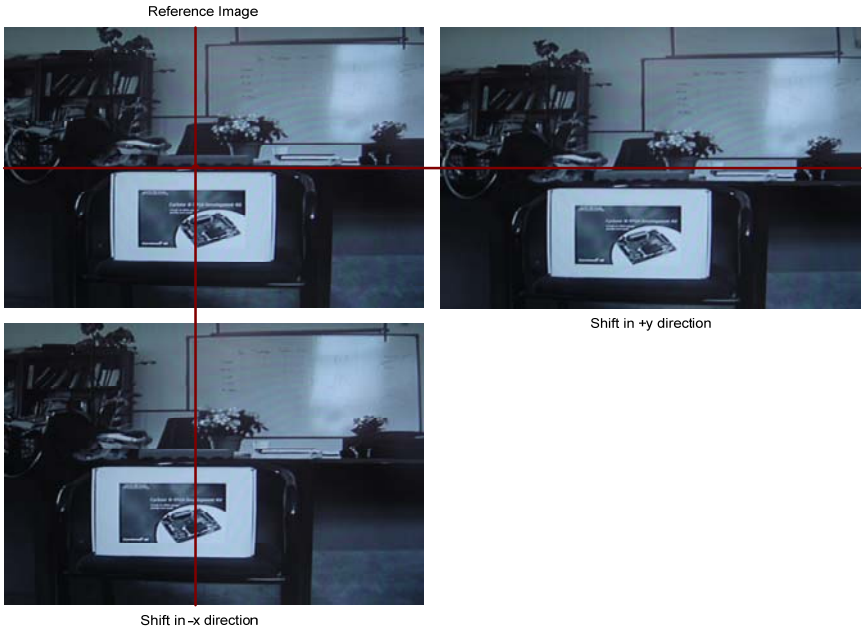

Figure 68: The Results of the Synthetic Shifts on Reference Image

Figure 68 shows the video frames which are captured from DVI monitor by a camera. These frames are the reference frame and the current frames after the synthetic shifts. The FPGA calculates the shift values by comparing the reference and current images.

In full search algorithm, the image is divided in search areas. The shift, TA and MAD values from these areas are evaluated in order to find the final motion estimation. Figure 69 shows the search areas on the image and Table 20 gives the estimation results for the different search areas.

Figure 69: The Search Areas on the Image

The results of the full search method shows that, the TA analysis is also important in real time working system as it is in MATLAB. The search areas with high texture find the correct shift values. In this example search area-2 contains extremely high texture when compared with others and its estimation results are correct, on the other hand, the other blocks from the first region (1,3,4,5,6) can not reach the correct shift values. The same block which has the highest texture is used for other shift tests and the results show that FPGA can calculate the exact shift values that are applied. For, high shift values such as -32 in y direction, the highest texture region passed to the search area-8, and the results of this area give the correct shift values.

Table 20: Full Search FPGA Results

Actual Shift Values		Search Area	Estimated Values			
y shift	x shift	Number	y shift	x shift	TA	MAD
0	0	1	-3	5	594	262
0	0	2	0	0	10009	2008
0	0	3	-2	0	627	280
0	0	4	0	11	913	429
0	0	5	0	-1	763	383
0	0	6	0	-4	804	375
5	5	1	4	-1	418	223
5	5	2	5	5	8993	1446
10	5	2	10	5	10650	1658
10	10	2	10	10	8700	1347
20	10	2	20	10	9722	1586
20	20	2	20	20	6318	1140
30	30	2	30	30	9156	1334
32	32	2	32	32	9233	1121
-5	0	2	-5	0	9368	1875
-5	10	2	-5	10	3531	565
-10	-15	2	-10	-15	3839	767
-32	-31	8	-32	-31	1337	530
-32	32	8	-32	32	5715	1241

After synthetic shifts, full search method is tested by the real shifts. The camera is located on a movable platform and shifts in x and y direction are applied. The results of the search areas are evaluated as explain in MATLAB implementation part. MAX_TA, MAX_TA_AVG and MAX_TA_MIN_MAD values are tested in FPGA. MAX_TA reaches the best stabilization performance among others but this

approach suffers from the local motions. When there exists a local motion on the image, this motion increases the texture at that region and this texture increase results in faulty estimations.

This problem is solved by using a texture analysis in the evaluation part. Many experiments are conducted and the effects of the local motions on the texture values are observed. The local motion cause a rapid texture increase which is impossible by a desired camera movement. Therefore, a comparison between two successive highest textures is used to prevent local motion effects. However, the FPGA tests show that the basic evaluation methods which are explained in MATLAB implementation part are not sufficient for the best evaluation. FPGA can compare the reference and current sub-blocks correctly but the results of these comparisons should be evaluated in more complex algorithms like Kalman which are more suitable for processors.

The phase correlation method is also implemented in FPGA with the external blocks such as video interfaces and memories. The FPGA tests are started with synthetic shifts. The results of these tests show that FPGA estimations are not correct when compared with the applied shifts. The source of the problem is searched by using a test pattern as a video source. The same pattern is also used in MATLAB and simulation to see the difference. After the tests, it is observed that, the FFT IP is not working as in MATLAB and simulation. So, the calculations for the phase correlation results in faulty estimations.

Figure 70 shows the FFT input which is sent to the IP in real time working FPGA and simulation, the same data sequence is applied for both FPGA implementation and FPGA simulation. The results of the FFT block are shown in Figure 71. The FFT outputs of FPGA simulation are the correct ones and these outputs results in correct shift values, however the FPGA implementation outputs are wrong and these wrong outputs prevent to calculate correct shift values. The phase correlation

method produces very good stabilization results in MATLAB and FPGA simulation, but in FPGA implementation, due to FFT problem, this method can not reach the desired results.

This problem shows that, in some cases, FPGA implementation on real hardware may not be the same with the simulations and previous analyses on MATLAB. There can be differences while the written code is compiled for the FPGA.

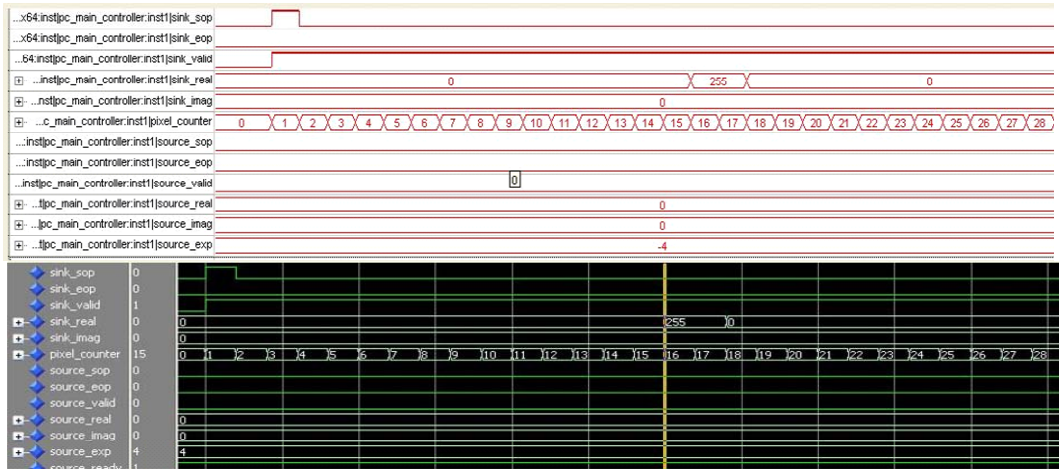

Figure 70: FPGA Implementation & Simulation FFT Inputs

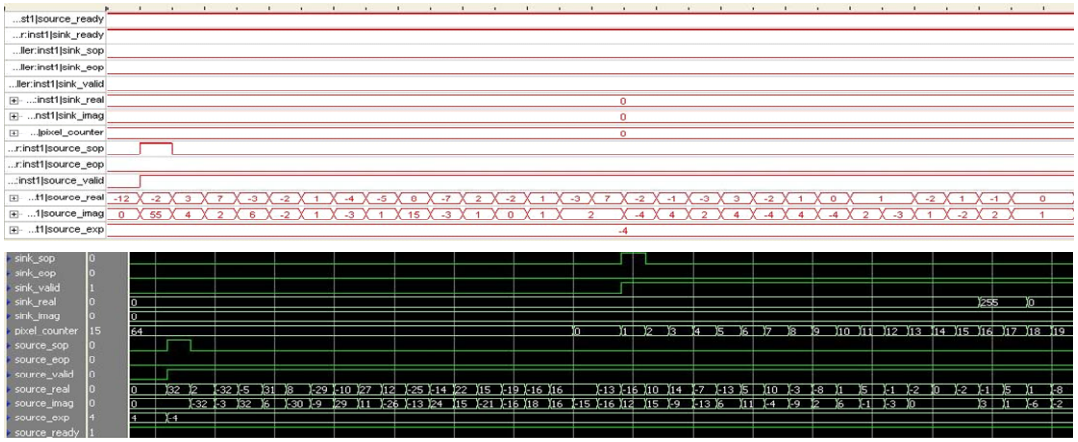

Figure 71: FPGA Implementation & Simulation FFT Outputs

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSIONS

Video processing applications are used in many areas such as consumer electronics and military areas. Every video processing application has its own initial requirement from the input video such as video format, resolution and frame rate, but the common requirement for all applications is the stabilization. Because, the translational and rotational changes in frame sequence cause faulty results in processing algorithms. Therefore video stabilization is the first and necessary step for other video processing applications.

Most of the systems use mechanical stabilization because these types of stabilizers prevent the undesired motion at the detector of the camera, so there will be no image loss. However, mechanical approach is not feasible for all platforms due to its huge and complex structure. Therefore, digital stabilization methods can be used as an alternative. Digital methods calculate the undesired motion by comparing two neighbor frames in the sequence. This comparison process requires long computation time and digital methods are not preferable in real time applications.

The recent advances in hardware elements, such as FPGAs, can handle the high computational load. The parallel structure of FPGA is configured in order to process many data at the same time. However, only FPGAs are not enough for all

video stabilization flow, some other elements such as video decoders and SDRAMs are required to construct a full system.

In this thesis, real time video stabilization is the main problem. Due to the platform which is mobile robot, mechanical methods can not be used. The platform also requires a real time flow. Since mechanical methods are not a solution and real time constraints are strict, digital methods are examined.

Firstly, the hardware that the digital video stabilization will be implemented is selected. Due to its high computation capacity, FPGA is chosen as a processing unit. The necessary video input & output, memory interfaces are constructed in FPGA. Then, the possible stabilization algorithms in literature are searched. Two main method; frequency and time domain approaches are selected as solution candidates. These methods are implemented in MATLAB to see their performance before FPGA coding. Since, the final implementation will be in FPGA, the MATLAB codes are written similar to FPGA. This approach provides more realistic comparison between the methods. After that, the FPGA (VHDL) codes are written for both methods. The VHLD codes are tested by simulation and then they are compiled for the FPGA on the main board. The debug tool (signal tap) is used to monitor real time working FPGA results. Finally, the stabilized video which is captured from a PAL video is displayed on DVI monitor.

The FPGA implementation results show that phase correlation method uses more logic elements than full search which is a disadvantage for small FPGAs. Although full search uses less logic elements, its power consumption is higher due to many comparisons in a shorter time, the higher power consumption may be a problem for the platforms which receive the input power from a battery.

The FPGA implementations of full search and phase correlation are tested by synthetic and real shifts. Full search synthetic shift tests show that FPGA can calculate the correct shift values at the search areas where texture is high. In real

shifts, this method suffers from blurring effect and environmental changes in the video frame sequence. Also, the local motions may result in faulty estimations since the texture analyses fails at the locations where local motions are occurred.

Phase correlation method gives encouraging results in MATLAB and FPGA simulation, but the FPGA implementation on real hardware shows that, this method can not work properly while FFT results are not correct. The problem in FFT IP block prevents this method to be implemented in FPGA successfully. But, the other parts of the FPGA implementation works correctly, so this method can be also used for video stabilization after the FFT problem is solved by the IP vendor.

To conclude, the implementation and comparison results show that, FPGAs are capable to run digital video stabilization methods. FPGA can calculate new stabilization data at every 40 ms which takes several seconds in MATLAB. The comparison between simulation and MATLAB results shows that FPGA can also reach the desired accuracy in motion estimations. The only difference arises at the motion evaluation part. The basic evaluation approaches gives good results in synthetic shifts but in real shifts with local motions, these approaches fail in some cases.

5.2 FUTURE WORK

FPGAs can handle high computational load but they are not suitable for recursive algorithms. In the motion evaluation part, different filters such as Kalman can improve the accuracy, but such algorithms are suitable for processors. Therefore, a processor can be can be used with FPGA to implement more complex evaluation algorithms.

REFERENCES

1. Jesse S. J., Zhigang Z. & Guangyou X., “Digital Video Sequence Stabilization Based on 2.5D Motion Estimation and Inertial Motion Filtering.”, Real Time Imaging 7 , 357-365, 2001.

2. Ertürk S., “Digital Image Stabilization with Sub-Image Phase Correlation Based Global Motion Estimation.”, IEEE Transactions on Consumer Electronics, vol. 49, no. 4, 2003.

3. Bayrak S., “Video Stabilization: Digital and Mechanical Approaches”, M.S. Thesis, METU, 2008.

4. Kumar S., Biswas M. & Nguyan T. Q., “Global Motion Estimation in Frequency and Spatial Domain”, ICASSP, 2004.

5. Özsaraç H., “FPGA Implementation of Graph Cut Method For Real Time Stereo Matching ”, M.S. Thesis, METU, 2010.

6. Ertürk S., “Real-Time Digital Image Stabilization Using Kalman Filters”, Real-Time Imaging 8, 317-328, 2002.

7. Tico M. & Vehvilailen M., “Robust Method of Video Stabilization”, 15th European Signal Processing Conference, 2007.

8. Yang J., Schonfeld D. & Mohamed M., “Robust Video Stabilization Based on Particle Filter Tracking of Projected Camera Motion“, IEEE Transactions on Circuits and Systems for Video Technology, Vol 19, No. 7, July 2009.

9. Vermeulen E. “Real Time Video Stabilization For Moving Platforms“, 21st Bristol UAV Systems Conference, April 2007.

10. Klupsch S., Ernst M., Huss S. A., Rumpf M. & Strzodka R., “Real Time Image Processing based on Reconfigurable Hardware Acceleration”, Proceedings of IEEE Workshop Heterogeneous reconfigurable Systems on Chip, 2002.

11. Neoh S. H. & Hazanchuk A., “Adaptive Edge Detection for Real-Time Video Processing using FPGAs ”, CF-EDG031505-1.0, 2005.

12. Dias A. F., Lavarenne C., Akil M. & Sorel Y., “Optimized Implementation of Real-Time Image Processing Algorithms on Field Programmable Gate Arrays“, Fourth International Conference on Signal Processing, ICSP'98, October 1998.

13. El-Ashmawi A., “Migrating FPGAs to Structured ASICs in Avionics to Reduce SEU Susceptibility”, OpenSystems Publishing-2007.

14. www.digitalcreationlabs.com , “Digital Video Overview”, last accessed date: 25.12.2010.

15. Specifications of the Camera Link Interface Standard for Digital Cameras and Frame Grabbers, October 2000

16. Digital Visual Interface (DVI), Revision 1.0, April 1999

17. The I2C Bus Specification Version 2.1, January 2000

18. Jack K., “YCbCr to RGB Considerations”, Application Note, March 1997.

19. Altera Corporation, “Cyclone III Development Board Reference Manual”, March 2008.

20. DynaPel Systems, “DynaPel SteadyEye Digital Real-Time Video Stabilizer Brochure”, February 2005.

21. Altera Corporation, ”DDR and DDR2 SDRAM High Performance Controllers and ALTMEMPHY IP User Guide”, February 2010.

22. Altera Corporation White Paper, “FPGA vs DSP Design Reliability and Maintenance”, ver. 1.1,May 2007.

23. Datar A. & Padhye A. “Graphics Processing Unit Architecture (GPU Arch)”, presentation, April 2005.

24. Altera Corporation,”FFT MegaCore Function User Guide”, November 2009.

25. Engelsberg A. & Schmidt G., “A comparative review of digital image stabilizing algorithms for mobile video communications”, IEEE Transactions Consum. Electron., vol. 45, pp. 591-597,1999.

26. Morimoto C. & Chellappa R., “Evaluation of Image Stabilization Algorithms”, ICASSP, vol.5, pp. 2789 – 2792, 1998.

27. www.videoclarity.com,“General Video Quality Defined”, last accessed date: 03.01.2010.

APPENDIX A

INTERNAL FPGA BLOCKS AND SIGNAL EXPLANATIONS

DDR2 SDRAM CONTROLLER

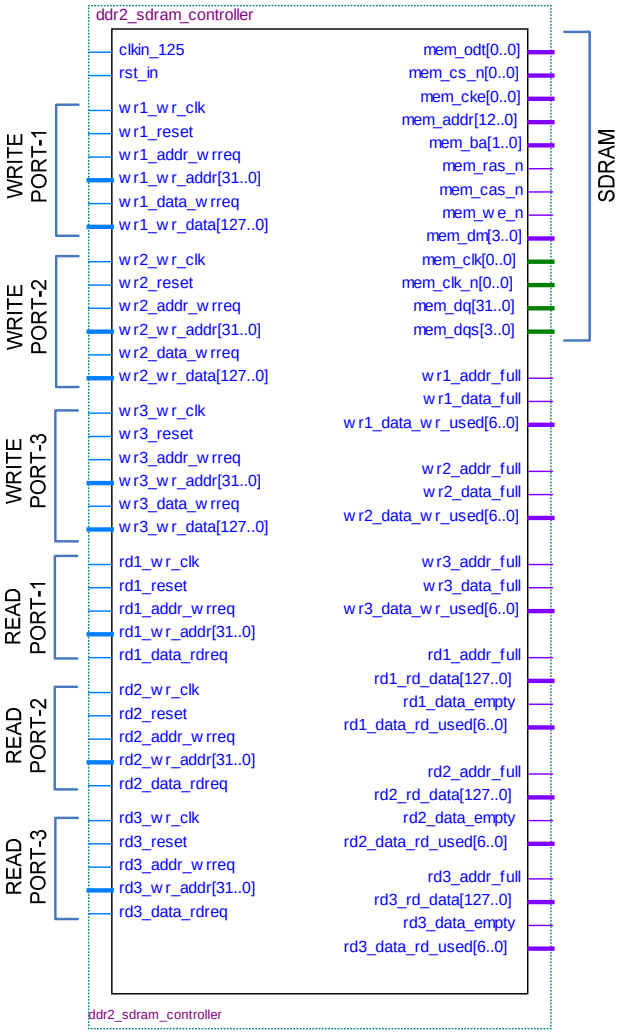

Figure 72: SDRAM Controller Block Diagram

Table 21 : DDR2 SDRAM Controller Signals

Signal	Description
clkin_125MHz	Controller main clock. It is 125MHz in this implementation and the maximum clock frquency is 150MHz.
rst_in	Controller reset. Reset all blocks inside the controller. "0": reset, "1": normal operation
wr*wr_clk	Write port clock. Port write clocks may be different, the arbiter handles the clock domain switch operation.
wr*_reset	Write port reset. Reset the selected port buffers. "1":reset, "0":normal operation
wr*_addr_wrreq	Address write request. This signal indicates that a new data packet has been written to port buffer and it will be transferred to the SDRAM.
wr*_wr_addr	Addreess. This bus is 32 bit and lower 23 bit shows the write start address in SDRAM. Addr (31..23) = Total numbers of written data. Addr (22..21) = Bank address. Addr (20..8) = Row address. Addr (8..0) = Column address.
wr*_data_wrreq	Data write request. The data is written to the port buffers.
wr*_wr_data	Data. 128 bit
wr*_addr_full	Address write request buffer full indicater. When this signal is '1', it means that the request buffer is full and a new request should wait until the arbiter empty it.
wr*_data_full	Data buffer full indicater. When it is "1", the written data will be lost.
wr*_data_wr_used	Shows the total number of data in the buffer.
rd*_wr_clk	Read port clock. Port read clocks may be different, the arbiter handles the clock domain switch operation.
rd*_reset	Read port reset. Reset the selected port buffers. "1":reset, "0":normal operation
rd*_addr_wrreq	Address read request. This signal indicates that a new data packet will be read from the SDRAM. After that signal goes high, arbiter read the data and write to the read buffers.

Table 21**:** Continuation.

rd*_wr_addr	Addreess. This bus is 32 bit and lower 23 bit shows the read start address in SDRAM. Addr (31..23) = Total numbers of data to be read. Addr (22..21) = Bank address. Addr (20..8) = Row address. Addr (8..0) = Column address.
rd*_data_rdreq	Data read request from the buffers.
rd*_rd_data	Data. 128 bit
rd*_addr_full	Address read request buffer full indicater.
rd*_data_empty	Read data buffer empty signal. When this signal is "1", there is no data in the buffer.
rd*_data_rd_used	Shows the total number of data in the buffer

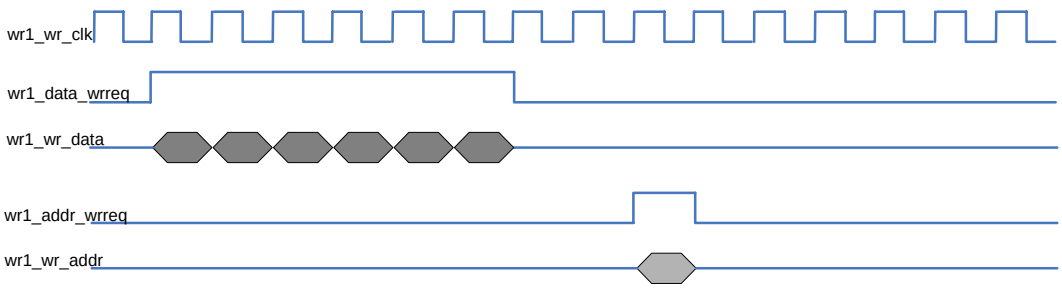

Figure 73: Write Operation to Write_Port1

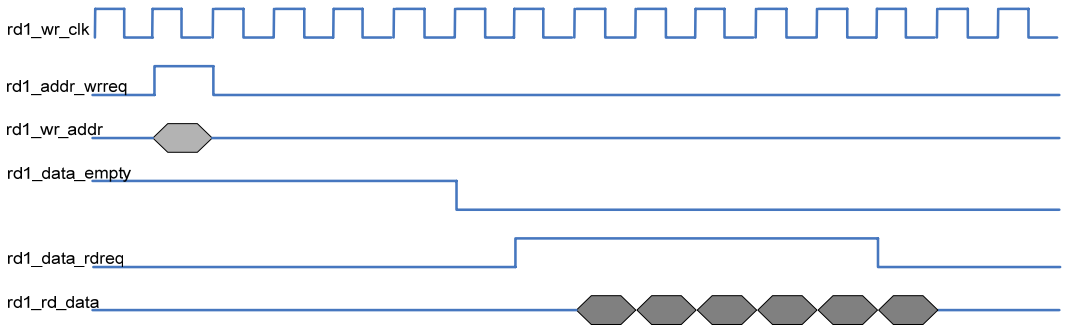

Figure 74: Read Operation from Read_Port1

1-D FFT IP Block

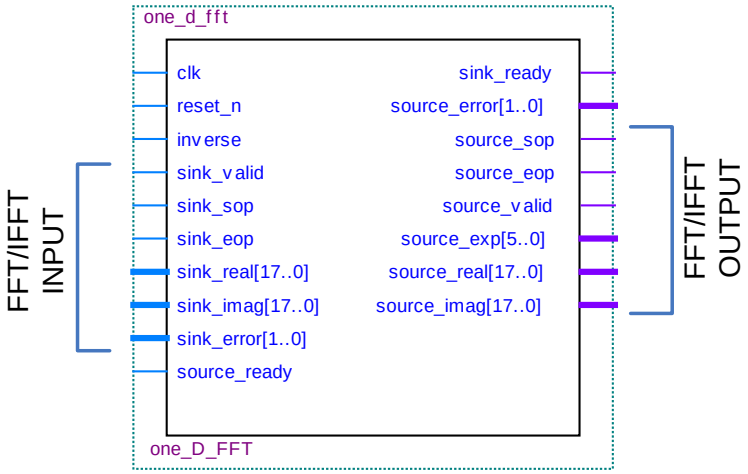

Figure 75: FFT IP Block Diagram

Table 22 : FFT IP Signals

Signal	Description
clk	FFT block clock input. 27MHz in this application
reset_n	FFT block reset. "0": reset, "1": normal operation
inverse	DFT / IDFT selection "0": DFT, "1": IDFT
sink_valid	FFT block input valid. The data given with this signal is processed in FFT block.
sink_sop	Input data start signal.
sink_eop	Input data end signal.
sink_real	The real part of the input data. 18 bit.
sink_imag	The imaginary part of the input data. 18 bit.
sink_error	The error signal for the input data [24].
source_ready	FFT block can give the calculation results.
sink_ready	FFT block can receive new data packet.
source_error	The error signal for the FFT result [24].
source_sop	FFT output start signal.
source_eop	FFT output end signal.
source_valid	FFT output data is valid.
source_exp	FFT output exponantial. This signal is used to scale real and imaginary parts [24].
sink_real	FFT output real part. 18 bit.
sink_imag	FFT output imaginary part. 18 bit.

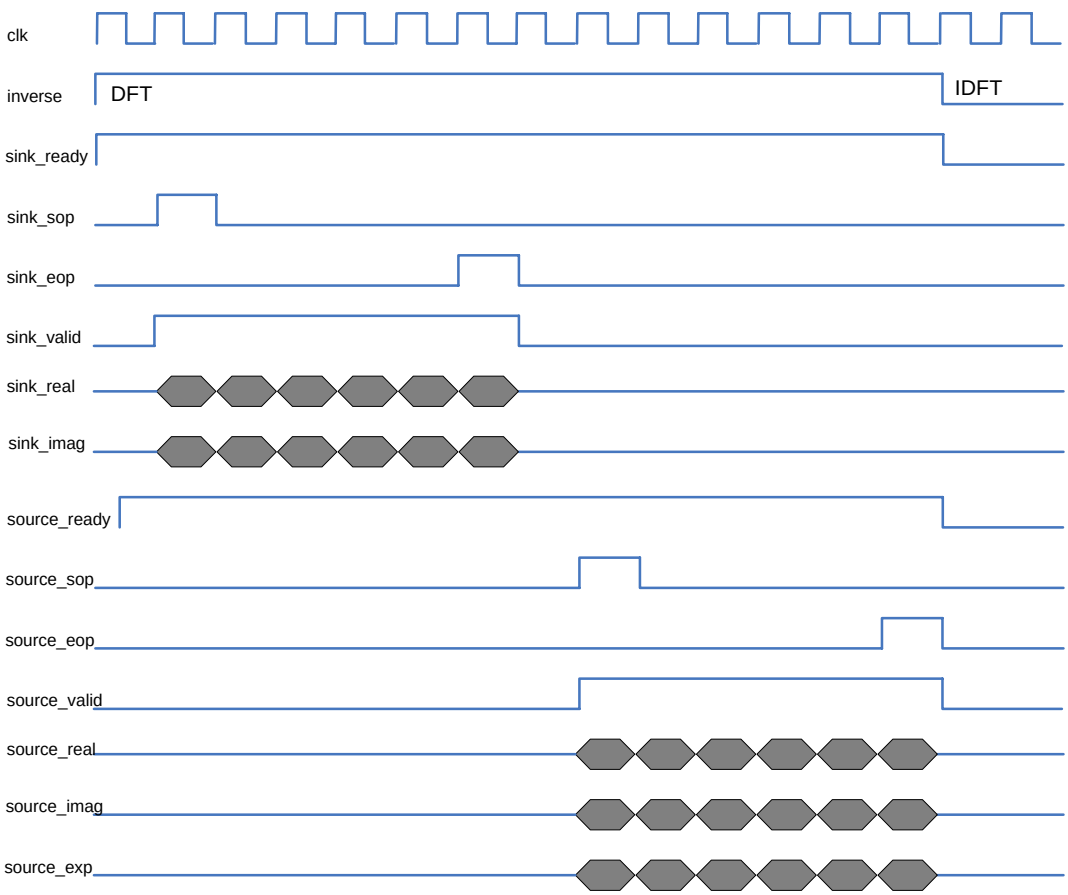

Figure 76: FFT IP Input/Output Flow