Hardware Based Platform Independent Rendering for Multi-Projector Display

By

Kamran Babar
2010-NUST-MS-EE(S)-13

Supervised By

Dr. Rehan Hafiz
Assistant Professor

This thesis is submitted in partial fulfillment of the requirements for the degree of Masters of Science in Electrical Engineering (MS EE)

School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

August 2012

APPROVAL

It is certified that the contents of thesis document titled, "Hardware Based Platform Independent Rendering for Multi-Projector Display" submitted by Mr. Kamran Babar have been found satisfactory for the requirement of degree.

Advisor: Dr. Rehan Hafiz

Signature: _____

Date: _____

Committee Member1: Dr. M. Murtaza Khan

Signature: _____

Date: _____

Committee Member2: Dr. Khawar Khurshid

Signature: _____

Date: _____

Committee Member3: Dr.Osman Hasan

Signature: _____

Date: _____

To my parents and teachers

CERTIFICATE OF ORIGINALITY

I declare that the research work titled **"Hardware Based Platform Independent Rendering for Multi-Projector Display"** is my own work to the best of my knowledge. It contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at SEECS or any other education institute, except where due acknowledgment, is made in the thesis. Any contribution made to the research by others, with whom I have worked at SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic is acknowledged. I also verified the originality of contents through plagiarism software.

Author's Name: Kamran Babar

Signature: _____

ACKNOWLEGMENTS

I am grateful to Almighty Allah who gave me courage and strength to complete this thesis. I am really thankful to Dr. Rehan Hafiz for his kind attention and guidance during this thesis. I am also thankful to my worthy Committee members, Dr. M Murtaza Khan, Dr. Khawar Khurshid and Dr. Osman Hasan, for their support and becoming a part of this work. I am extremely gratified to Mr. Atif Ahmed for his valuable information regarding Geometric Correction and lending me his GPU Card for the testing of the hardware implementation. Finally, I am thankful to all who had helped me in any form during all this period.

Contents

1 Introduction 1
1.1 Motivation 2
1.2 Problem statement 2
1.3 Contributions 2
1.4 Thesis Organization 3
2 Literature Survey 4
2.1 Geometric Correction 4
1.1. Software Based Geometric Correction 4
2.2 Hardware Based Geometric Correction 6
3 Implementation 8
3.1 Algorithm 8
3.2 MATLAB Implementation 9
3.3 Hardware Implementation 11
3.3.1 Micro architecture 11
3.3.2 IAU (Image Acquisition Unit) 14
3.3.3 Parameter Calculation Block 15
3.3.4 Single Parameter Calculator 17
3.3.5 Interpolator 22
4 Results, Performance and Discussion 25
4.1 Resource Utilization 25
5 Conclusion 27
6 Future Work 28
7 References 29
Appendix 31
A. Matlab Codes 31
A.1 Matlab Code with Precalculated Parameters 31
A.2 Matlab Code with Param Calculation 35
A.3 Matlab Fixed Point Implementation 37
A.4 Param Calculation Unit Fixed Point Implementation 37
B. Verilog Codes 39
B.1 Single Parameter Calculator 39

B.2 Parameter Calculation Block.....44
B.3 Interpolator.....47

LIST OF FIGURES

Figure 1: Image before correction and after correction 4
Figure 2: Comparison of several correction methods 6
Figure 3: Original Image and Corrected Image 10
Figure 4: Calculated row index using Floating and Fixed Point arithmetic 10
Figure 5: FixedPt_Row(256) - FloatingPt_Row(256),FixedPt_Row(512) - FloatingPt_Row(512) 11
Figure 6: Micro Architecture 12
Figure 7: Data Capture and read timings 13
Figure 8: IAU Architecture 15
Figure 9: IAU read and write timings 15
Figure 10: Parameter Correction Block Architecture 17
Figure 11: PRU Datapath 20
Figure 12: PRU Datapath with Fixed Point 20
Figure 13: PRU Datapath2 21
Figure 14: PRU Pipelined Datapath 22
Figure 15: Interpolator Datapath 23
Figure 16: Interpolator Datapath2 23
Figure 17: Interpolator Pipelined Datapath 24

LIST OF TABLES

Table 1: System interfacing signals 14
Table 2: Parameter Calculation Block interfacing signals 16
Table 3: SPC interfacing signals 18
Table 4: Interpolator interfacing signals 22
Table 5: Resource Utilization for Single Channel 25
Table 6: Comparison between Single Channel and Three Channel (RGB) Resource Utilization 26
Table 7: Comparison between Single Projector and Dual Projector System 26

ABSTRACT

Multi-projector walls are being widely used for the construction of large panoramic views, due to their ability to deliver great visual experience in a cost-efficient manner. To generate seamless projection on such surfaces two major corrections are required: geometrical correction and the photometric correction. Geometric correction is performed to align the projected contents across multiple projectors and also to account for the distortions that appear due to the surface shape. Photo-metric corrections are performed to make the projection seamless across the projector boundaries. When the surface is planar, homography defines the relation to align the projected contents using 8 transformation parameters. However when the surface is curved further higher order warping is required and that complexity of such transformations is based on the complexity of surface. A geometric correction comprising homography and second order Bezier transformation can be employed to perform multi-projector projection on quadratic surfaces. Most of the existing correction systems are implemented in software, where correction program interrupts the video stream going to graphics chip, applies correction and finally sends out to the DVI/VGA port. For the software based solutions, GPUs are widely used for their capability to handle multiple threads at a time. Such software based solutions also share some resources with the system and hence affect the system performance. Some hardware based solutions were also developed, but they are mainly existed for low resolution images.

In this work, feasibility study is performed, and architecture is proposed for performing homography and Bezier transform in real time. This architecture can easily be implemented on FPGAs and can be fabricated into ASICs. The proposed system can be installed at any Video output channel; hence it removed the platform dependency. Second order Bezier transform is used to apply geometric correction to the video stream. After the successful simulations, the

design was tested on Xilinx Spartan 3A (DSP Video Starter Kit). The system was tested for resolutions up to 1024*768 with the frame rate of 75fps, but the current implementation can easily handle incoming video having resolutions up to 1600*1200. Also, the system was tested using DVI input and output ports, which can be easily changed for HDMI or other video standards.

Chapter1

1 Introduction

Applications like home theater, gaming, remote collaboration, scientific visualization, and human-computer interaction require great visual experience, so high resolution, large-scale and wide field-of-view are the requirements for such applications. Such solutions can be built by using multiple smaller display units; the result of such techniques is the inter-unit seams, which degrade the visual experience. So, Multi-projector walls are used for building large seamless displays, this solution also reduces the cost of the overall solution. Furthermore, to enhance the field-of-view along with immersive visual experience, curved projections surfaces are also employed.

Construction of such curved displays comes with some challenging tasks. The most significant of them is to perform the correction of image contents to reduce the image distortions caused by projecting the image on a curved surface, and to remove the inter-unit seams by aligning the image sub-sections projected by different projectors.

Besides the quality of geometric and photometric corrections applied, the feasibility of installation and calibration to be done by end-users is also kept in consideration. A full automated re-calibration is preferred after a specific time period. Furthermore, the runtime implementation of the geometric and photometric correction needed to be so efficient that it can process the videos at the frame rates offered by latest devices.

1.1 Motivation

Whenever projectors are installed in any permanent or semi permanent setup, geometric correction is required to visually correct the displayed contents. All systems which perform automatic geometric calibration use cameras to sense the projection environment. Image correction is the process in which camera parameters and camera-projector relationships are used.

A huge body of literature about geometric correction solutions exists to date [1][3][4][7][8][9][10][13][17], but little work has been done in the hardware domain. In existing techniques, mostly software based solutions are deployed for the correction of images, which make the system platform dependent, and also uses system resources like CPU, GPU and RAM. On the other side, already existing hardware based solutions are designed for low resolutions and lower frame rates [5][6][19]. Therefore a real-time and hardware based solution is designed for the geometric correction of incoming video from DVI channel. The proposed solution is designed by keeping targets of ease of use, platform independency, and less reliance on the system resources.

1.2 Problem statement

In the image correction, geometric correction is one of the computationally extensive steps. Most of the software based approaches used, make the system platform dependent and also affect the system performance by putting load on processor and requires memory.

The focus of this thesis is **Hardware Based Platform Independent Rendering for Multi-Projector Display.** The purpose of this research is to provide a real-time, standalone and platform independent solution for applying geometric correction of the DVI video stream. Due to easy prototyping, wide adoptability and variety of application, the target platform was FPGA. Later on this design can be easily moved to the standalone ASIC.

1.3 Contributions

Notable contributions of this thesis work are:

- The development of an efficient architecture for real-time multi-projector rendering suitable for curved surfaces.

- Architecture optimization by avoiding some multiplications involved in parameter calculation(PRU module)
- Row caching for better memory management.
- Pipelining for higher bandwidth

1.4 Thesis Organization

Thesis is organized in five chapters. Chapter 2 gives introduction about the geometric correction and existing hardware based techniques are discussed. Chapter 3 presents the architecture of the developed solution. Chapter 4 presents the results. Chapter 5 summarizes the work and chapter 6 proposed further possible enhancements of the system.

Chapter2

2 Literature Survey

2.1 Geometric Correction

Geometric correction is the process of matching the image projection according to the projection surface. The curved display need to have a physically consistent appearance across a large zone of viewing locations in 3D space. One way to achieve this is to imagine that the image to be displayed has been printed on a large, planar piece of paper that we conform, like wallpaper, to our display surface. Such a display would provide an intuitive, natural experience to one or more viewers, whether stationary or moving, without requiring adaptation of the projection parameters over time. Below diagram shows the results of application of geometric correction

Figure 1: Image before correction and after correction (taken from [1])

1.1. Software Based Geometric Correction

Most of the correction systems deployed are software based [1][3][4][7][8][9][10][13][17]. In such a scenario, a computer having graphics card performs several function i.e. splitting of the

video stream for multi projectors, calculation of the correction parameters and finally correction of images with the help of correction parameters.

A coordinate transform based approach is used for geometric correction in [1] for the construction of a panoramic projection from four projectors, each having a resolution of 1024*768. The system employs two AMD Opteron 2.19 GHz CPUs and two Nvidia Quadro FX3400 GPUs. The proposed system requires processor and memory resources, and can operate at the maximum rate of 24fps. Further projectors can only be added by employing more CPUs and GPUs. Hence, larger resolution images add to the system requirements significantly.

One of the major issues of image correction is the handling of data, which was outlined by [2]. For the video stream with resolution of 1600x1200 with 75fps, a continuous read and write access to RAM at the rate of 432MB/s each is required. This problem is more significant in software based techniques, as the system has to run several other applications in the meantime. Image data is needed to be rearranged in such a manner, that it can be accessed quickly. Software based solution gives only the higher level access to the resources, hence such solution can only manipulate the memory up to a specific extent.

Takahashiet.al [3] calculated the SIFT (Scale Invariant Feature Transform) feature points for the incoming video stream and then applies the geometric correction on the basis of the feature points. SIFT calculates the heterogeneous feature of an image, which are invariant to the size of the image and rotation of the image contents. The proposed scheme uses the SIFT feature points to get the relationship between the incoming image contents and the projected surface, hence it minimizes the requirement to recalibrate the system.

A geometrically aware and self-configuring system was proposed in [4].The objectives of the system are shape adaptive display, object adaptive display, Planar and curved display using a cluster of projectors. A novel calibration technique alongwith the correction techniques is used. ATI Radeon graphics card on Dell Inspiron was used to test the proposed system.

Several other texts suggest the software based image correction solutions. Such solutions were developed for a specific software and hardware environment, hence are not a good solution for general purpose consumer market. Furthermore, such solutions need their share from CPU and memory resources, which affects the system performance.

2.2 Hardware Based Geometric Correction

Due to the drawbacks and deficiencies with the software based approaches, as discussed in section 2.1, hardware based implementation were worked out [5][6][19]. Hence hardware based approaches offer platform independency and minimizes the reliance on any other software or hardware. For such systems, most commonly used platform is FPGA, which enable easy prototyping along with ability to introduce parallelism and pipelines. Furthermore, ASIC production is one step away from FPGA implementation.

Hardware based implementation requires all the calculation to be converted to fixed point. One of such implementation is suggested in [5] and is compared with an implementation on Pentium 4. The paper discussed, two hardware based approaches: A real time approach, in which the transformation polynomial to be evaluated was implemented inside the FPGA at the runtime, and the look up table based, in which the polynomials were evaluated offline and stored into the look up tables. Finally, a comparison between the FPGA based real time calculation, lookup table based and software based approaches, as shown in the table below (taken from [5]).

Implementation	Area (%)	SRAM (MB)	Clock speed	Time (ms) 600 × 600 pixels
Pentium 4 real time calculation	n/a	n/a	1.7 GHz	780
Pentium 4 lookup table	n/a	n/a	1.7 GHz	760
non-pipelined FPGA real time calculation	63	5	10 MHz	360
non-pipelined FPGA lookup table	7	7	22 MHz	130
pipelined FPGA real time calculation	64	5	25 MHz	15
pipelined FPGA lookup table	6	6	34 MHz	11

Figure 2: Comparison of several correction methods (taken from [5])

Lin Qiang[6] used coordinate transformation technique to apply the geometric correction of the barrel distorted image. First of all, to get the radius and angle information, the Cartesian coordinates were transformed into the polar coordinates. A correction polynomial is applied to calculate the corrected coordinates. Finally the corrected polar coordinates were transformed back to the Cartesian coordinated and bilinear interpolation was performed to achieve the results. CORDIC [20] (COordinate Rotation DIgital Computer) is the most commonly used approach for coordinate transformation in hardware domain. N iterations of the CORDIC algorithm were unrolled, which add up to make a large critical path. The critical path delay introduced due to the unrolling of the CORDIC stages, was minimized by adding pipeline stages. It is also concluded

that the multiplier used in interpolator add up to make a significant combinational path, so pipeline stages were also introduced in the interpolator to increase the clock frequency. The proposed system can run at the frequencies up to 64MHz, and is tested for 80x80 images.

Several hardware based approaches reflect some advantages on the software based solutions. Hardware implementation had shown platform independency, and does not affect the performance of the system by sharing the system resources. Furthermore, hardware implementations can be configured to work at the higher rates by adding further pipeline stages to the architecture

this research worked focused on the modern age video standards i.e. higher resolution like 1600x1200 and frame rates like 90fps/110fps. Architecture was developed which ensure the processing of data at the required rates to meet the system requirement. To achieve the objectives several techniques were used, such as pipelining of the datapath, resource sharing.

Chapter 3

3 Implementation

This chapter discusses the algorithm used for the geometric correction. First of all the algorithm was implemented in Matlab and an analysis of its results was done. Later on the algorithm was translated to the hardware. This chapter discusses the Matlab implementation and hardware architecture of the systems.

3.1 Algorithm

The basis for this research work was the work done in VISPro Lab SEECS [24]. A software based system was developed which applies geometric correction for projecting images on the curved surfaces using OpenGL. The technique used for geometric correction has two steps, first is to apply homography and then Bezier transformation is applied. Homography is the relationship between the vector space of the projector and the vector space of the viewer. It is a linear transformation and can be explained by matrix operations (Below equations are taken from [22]).

Given:

$$p_a = \begin{bmatrix} x_a \\ y_a \\ 1 \end{bmatrix}, p'_b = \begin{bmatrix} w'x_b \\ w'y_b \\ w' \end{bmatrix}, \mathbf{H}_{ab} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

Then: $p'_b = \mathbf{H}_{ab} p_a$

Where: $\mathbf{H}_{ba} = \mathbf{H}_{ab}^{-1}$.

Also:

$$p_b = p'_b / w' = \begin{bmatrix} x_b \\ y_b \\ 1 \end{bmatrix}$$

Here P_a is the original coordinates and H_{ab} is the homography matrix to convert the coordinates from P_a to P_b. When the below homography matrix is used, the image is shifted by 10 places in horizontal axis and 10 places in vertical direction

The homography matrix used in this case was

HomographyMatrix = [1 0 10]
[0 1 10]
[0 0 1];

Bezier surfaces are used for modeling of computer-aided design and computer graphics. A set of control points is used to define any Bezier surface. Bezier surfaces can be of any degree but bi-cubic are used in most of the applications. A given Bezier surface of order (n, m) is defined by a set of (n + 1)(m + 1) control points. A two-dimensional Bezier surface can be defined by the equation below, where the position of a point p as a function of the parametric coordinates u, v is given by(equations are taken from [21]):

$$\mathbf{p}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(u)\, B_j^m(v)\, \mathbf{k}_{i,j}$$

Where

$$B_i^n(u) = \binom{n}{i}\, u^i (1-u)^{n-i}$$

In this implementation bicubic Bezier with n=2 and m =2 is used, hence there were 9 control points used.

3.2 MATLAB Implementation

First of all a floating point Matlab implementation was performed. As, system was to be implemented on FPGA, so fixed point implementation on Matlab was performed. Several fixed point formats were applied and tested; finally a fixed point standard with minimum bits required but with results comparable to floating point implementation was selected. Below diagram shows the image obtained from fixed point implementation.

Figure 3: Original Image and Corrected Image

Matlab implementations are added in the Appendix. Due to the conversion to fixed point from floating point, some of the error (quantization error) is introduced. Hence the fixed point formats are chosen such that the difference in the results should not be significant, in the meantime keeping the number of bits minimum. Below figure shows the plots for the calculated row index for row = 256, using floating point and fixed point implementations.

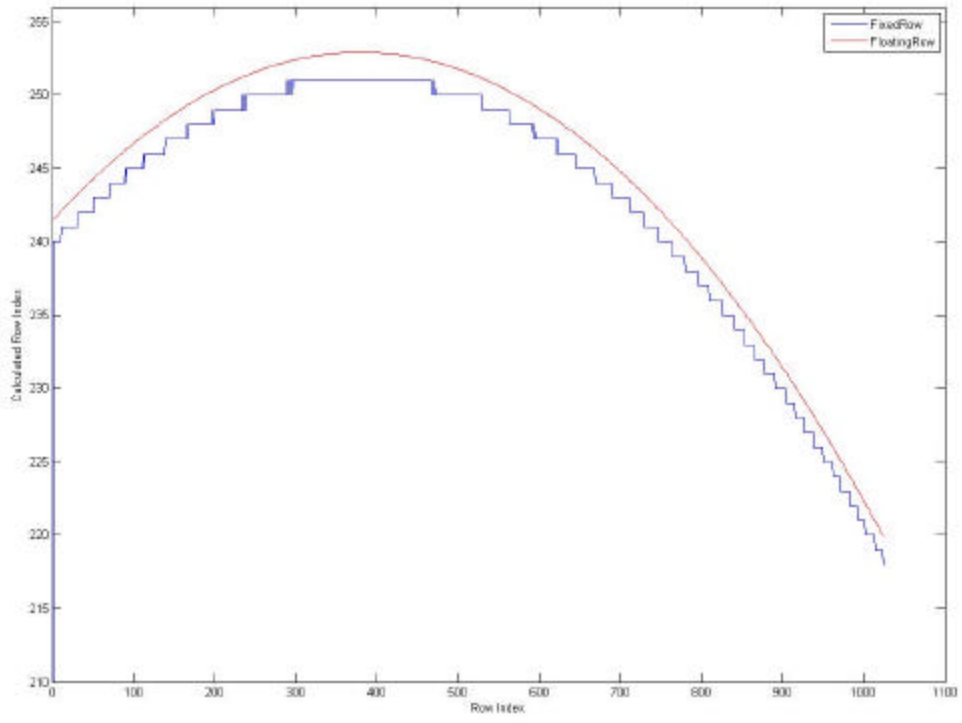

Figure 4: Calculated row index using Floating and Fixed Point arithmetic

Due to conversion some of the error is introduced, but the error is not much significant, as shown. Similarly, below figure shows the difference in the computed row index for row = 256 and row = 512.It can be noted that the variation in difference for a specific row lies between 0 and 1. As the target application is an image, so keeping in view that neighboring pixel values have the almost same values, the distortion shall be of limited noticeability. Below figure shows the difference between the row index calculated using fixed point and floating point implementation. Here it can be noticed that the difference between row indexes lies in the range of single point.

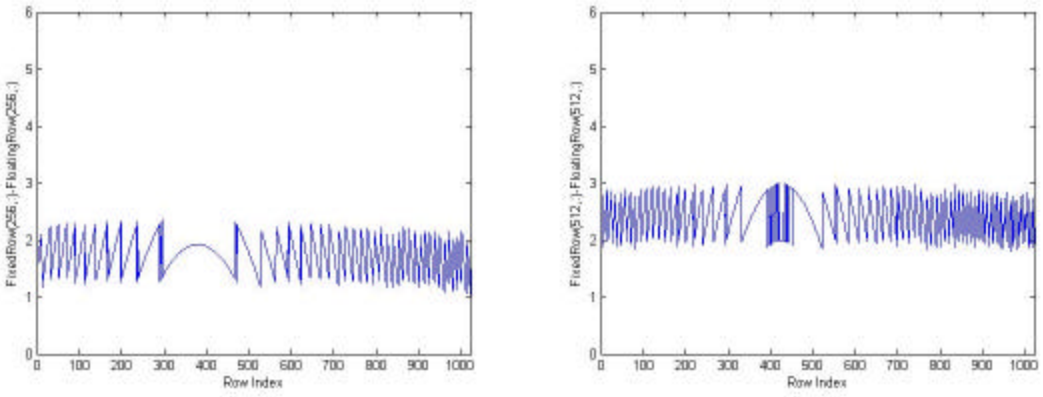

Figure 5: FixedPt_Row(256) - FloatingPt_Row(256),FixedPt_Row(512) - FloatingPt_Row(512)

3.3 Hardware Implementation

Below section explains the hardware architecture of the real time geometric correction system. This system gets input from the DVI Input decoder IC and processes the frames in the real time and finally sends the output to the DVI output IC (CH7301). This system runs on the DVI input clock of 165MHz. The system mainly uses external SRAM for storing frames.

3.3.1 Micro architecture

A modular approach is adopted for the system; hence each block has its own dedicated function. DVI Input module communicates with the DVI In port and caters for timings of the DVI In signals. Image Acquisition Unit (IAU) buffers the pixels data into its local fifos and when one complete row is stored, and then it moves the pixel data to the memory (SRAM). Memory Write Unit reads the pixel data from IAU FIFOs and stores into the memory (SRAM).Parameter Calculation Block (PRB) generates the corrected row and column index corresponding to the current row and column indexes by applying homography and Bezier transformation. The output

of PRU is fed to Memory Read Unit, as the address from where data is to be read. Memory Read Unit reads the data from memory (SRAM), when instructed by Memory Read Unit. Interpolator approximates the outgoing pixel value on the basis of the neighboring pixel values by applying bilinear interpolation in the real time. A FIFO of depth equal to row size (no_cols) is used for storing previously fetched pixel values to be used by interpolator. This fifo acts a cache for the data, hence reduces the read load on the memory. DVI Output Module sends the corrected pixel data to the DVI output port, so it generates the timings for the DVI output signals. An SRAM which is available on the Spartan 3E Kit is used for the buffering the video frames. The available SRAM had 1MB storage capacity, and it is used to buffer 2 frames at a time. Following diagram shows the interconnection between the modules of the system.

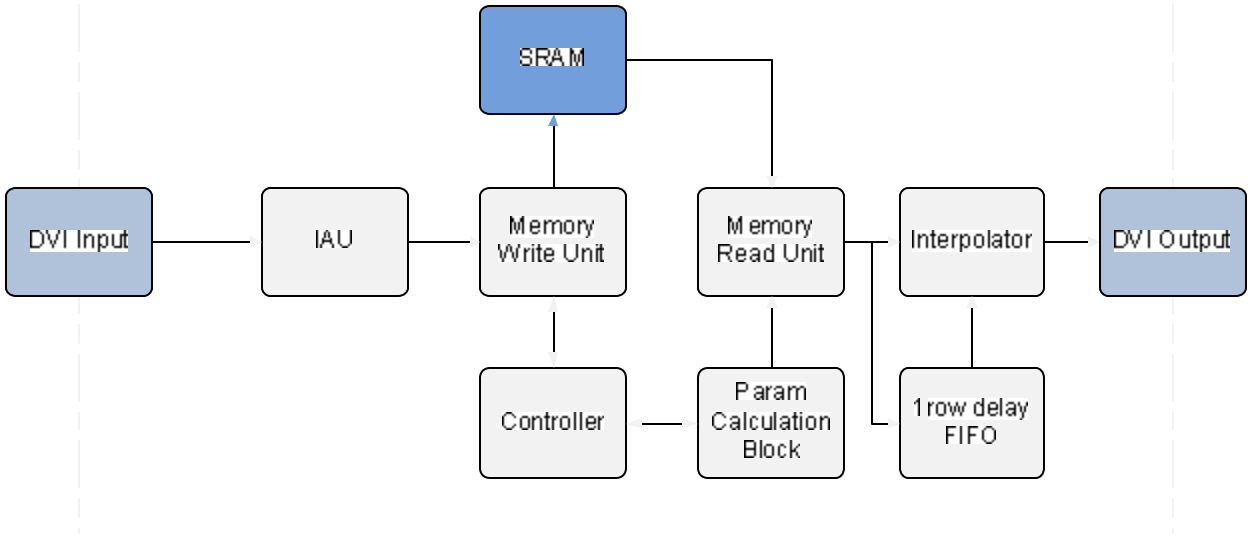

Figure 6: Micro Architecture

This system requires the buffering memory for two frames. The current incoming frame is being written into SRAM, and in the meantime the previously stored frame is read out from SRAM. The system processes the delayed frame and sends the output to the DVI output port, so a latency of one frame is introduced by this system. Below diagram shows the frame buffering and reading timings.

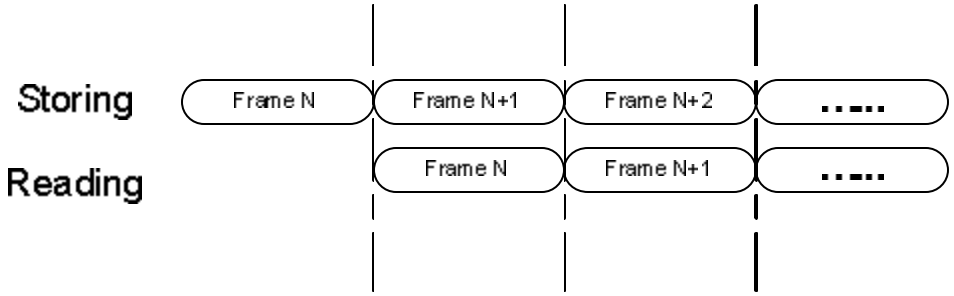

Figure 7: Data Capture and read timings

Below table gives the details of the interfacing signals, used to connect FPGA to DVI ports and other components.

Signal Name	Width	Direction	Description
reset_in_n	1	Input	Active Low reset for System
clock_out	1	Output	Output clock to SRAM
clock_enN	1	Output	Clock enable to SRAM. This signal is active Low
ADV_LDn	1	Output	Auto Increment or Load selection for SRAM. This pin selects the incrementation of address in Burst mode or start of new burst
WEn	1	Output	Active low write enable for SRAM. This pin selects whether write operation is to be performed of read operation
BWn	4	Output	Byte enable for write case. This signal masks the data bus going to SRAM
OEn	1	Output	Active Low Output enable. This signal is used to read data from SRAM in the read mode
CE1n	1	Output	Active low Chip enable. This signal should be low while SRAM operation is being performed
CE2	1	Output	Active high Chip enable. This signal should be high while SRAM operation is being performed
CE3n	1	Output	Active low Chip enable. This signal should be low while SRAM operation is being performed
address_out	18	Output	Address bus for SRAM
DQ	32	Inout	Data Bus for SRAM. This is bidirectional Bus. In case of write cycle, FPGA drives this bus and incase of read cycle, SRAM drives this bus
FPGA_CLK_33M	1	Input	33 MHz clock. This clock is required by Picoblaze, for initial configuration
IIC_SCLK	1	Output	IIC output clk
IIC_SDAT	1	Inout	IIC data signal
DVI_XCLK_P	1	Output	DVI output clock
DVI_XCLK_N	1	Output	DVI output inverted clock
DVI_DE	1	Output	DVI output Data enable. This signal is going to output

			decoder, it is used to insert idle cycles
DVI_H	1	Output	DVI output horizontal Sync. This signal is going to output decode(CH7301)
DVI_V	1	Output	DVI output vertical Sync. This signal is going to output decode(CH7301)
DVI_D	12	Output	DVI output data bus. This signal is going to output decode(CH7301). This signal has data on both of clock edges. Hence it transfers 24bits in 1 clock cycle
DVI_RESET_B	1	Output	DVI reset output
MEM_EN_B	1	Output	Memory enable. This signal disable the FMC card on the kit and enable SRAM
CTRL_SCL	1	Output	Control IIC clock
CTRL_SDA	1	Inout	Control IIC data signal
INCLK_P	1	Input	Input clock coming from DVI In channel. This is 165MHz clock
INCLK_N	1	Input	Input inverted clock coming from DVI In channel. This is 165MHz clock
DVIIN_DE	1	Input	DVI data enable input coming from DVI In channel
DVIIN_HS	1	Input	DVI horizontal sync input coming from DVI In channel
DVIIN_VS	1	Input	DVI vertical sync input coming from DVI In channel
DVIIN_R	10	Input	DVI input RED bus coming from DVI In channel
DVIIN_G	10	Input	DVI input GREEN bus coming from DVI In channel
DVIIN_B	10	Input	DVI input BLUE bus coming from DVI In channel

Table 1: System interfacing signals

3.3.2 IAU (Image Acquisition Unit)

IAU unit gets the input from the DVI input port and temporarily buffers the pixel data until it is written into the Memory. This module uses two FIFO of row size i.e. 1024 in the ping pong manner. Firstly incoming pixel data is stored into FIFO1, when FIFO1 gets full; the write logic switches to FIFO2 and starts storing data into FIFO2. When FIFO2 gets full, the write logic switches back to FIFO1, which would be empty by this time. While data is being stored into FIFO2, Memory write unit reads the data from FIFO1 and stores into the memory. Similarly, when data is being stored into FIFO1, FIFO2 will be read out. Below diagram depicts how two FIFOs are used.

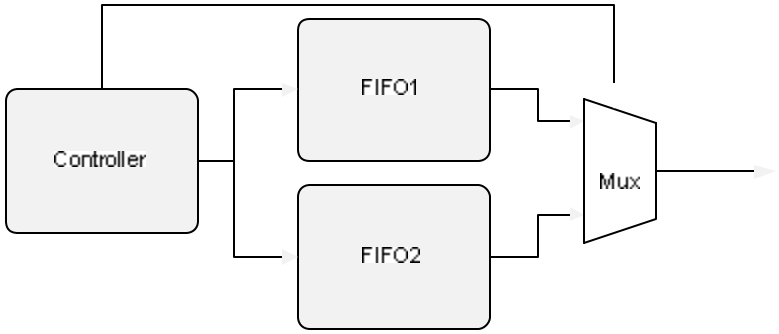

Figure 8: IAU Architecture

As the IAU module waits for the FIFO to get full before moving data to Memory, so a storing latency of Row Size is introduced. This means that any row is stored temporarily in IAU, after the completion of whole row data it is moved into the memory. This technique ensures that none of the pixel data is discarded while Memory is busy for other purposes. Below diagram shows the storage and reading of data from IAU.

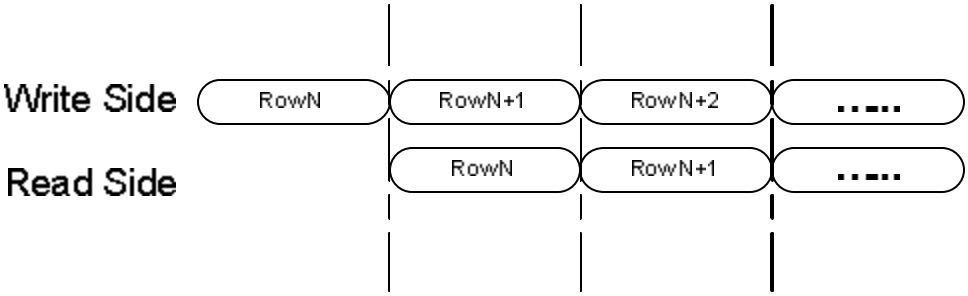

Figure 9: IAU read and write timings

3.3.3 Parameter Calculation Block

Parameter Calculation block is used to generate the corrected row and column index. This block consists of 4 Single Parameter Calculator units (SPCs) and a FIFO. This module uses a slower down version of the clock. Hence incoming clock is slow downed by a factor of 4. Following table describes the interfacing signals for this module.

Signal Name	Width	Direction	Description
clock	1	Input	Input clock signal
count_enable	1	Input	Enable signal. This signal is used to insert idle cycles during the operation, when DE signal is low on the DVI port. This signal halts the module at its current state.
reset_n	1	Input	Active low reset
input_addr	18	Input	Incoming Row and Column address. 2 LSbs of the column

			address are left and other bits along with row address are used as input to this module i.e. input_addr = {row_address,col_address[9:2]}
data_out	38	Output	Data out bus. This bus holds the corrected Row and Column Index. Each of the corrected Row and Column Index has 19bits, 11 for Integer part and 8 for fractional part. The format is data_out = {row_int,row_frac, col_int, col_frac};
output_enable	4	Output	Output enable signal. This signal is high when a valid data is available at the data_out bus

Table 2: Parameter Calculation Block interfacing signals

This module uses four instances of Single Parameter Calculator (SPC). SPC module requires approx 17ns to compute the corrected indexes, which is larger than the clock period of 6.06ns (165 MHz clock). So, the incoming clock is divided using a divide by 4 counter to meet the timing requirements of the SPC module. As, other modules are running on incoming clock and SPC is running on the Div4Clk, so to meet the throughput requirement, parallelism of FPGA is used and 4 parallel SPC modules are deployed. Finally to cater the clock domain crossing issues, a dual port asynchronous FIFO is used, which gets data from four SPC modules at Div4Clk and sends data out on original clock. Following diagram shows the micro architecture of the module.

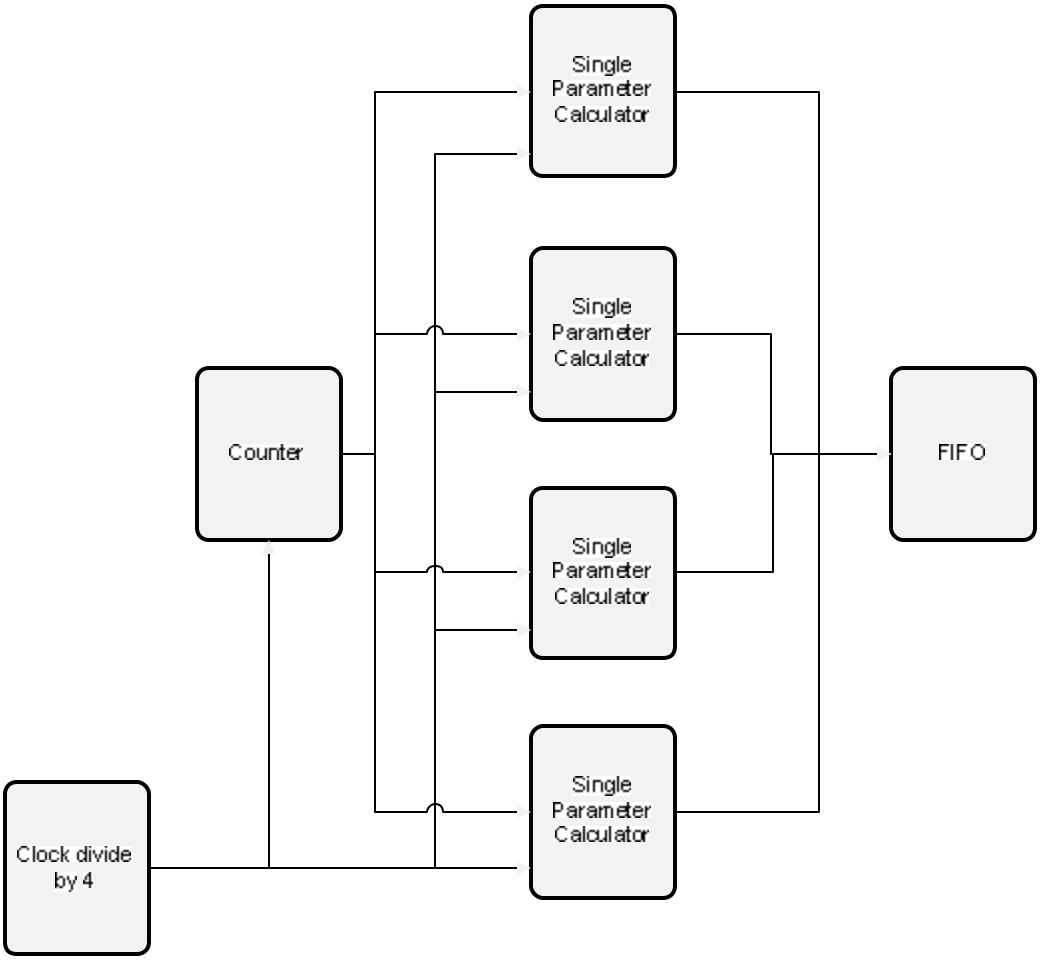

Figure 10: Parameter Correction Block Architecture

3.3.4 Single Parameter Calculator

SPC module gets the current row and column index as input, and applies the Bezier transform equations to calculate the row and column indexes. This module applies the Homographic correction and Bezier Transformation. The following table describes the interfacing signals for this module.

Signal Name	Width	Direction	Description
clock	1	Input	Input clock signal
reset_n	1	Input	Active low reset
row	10	Input	Incoming row index
col	10	Input	Incoming column index
row_out	11	Output	Output row index. This signal has the integer part of the corrected row
col_out	11	Output	Output column index. This signal has the integer part of the corrected column

row_frac	8	Output	This signal has the fractional part of the corrected row
col_frac	8	Output	This signal has the fractional part of the corrected column

Table 3: SPC interfacing signals

The equations for the calculation of the corrected row and column index using Bezier transform are given below. This module uses 18 correction coefficients, as shown as $P_{i,j}$ in the below equations.

- $Y(i,j) = cp_{1,1}*a_j^2 * c_i^2 + 2*cp_{2,1}*a_j^2 *c_i *d_i + cp_{3,1}*a_j^2 * d_i^2 + 2*cp_{4,1}*a_j*b_j*c_i^2 + 4*cp_{5,1}*a_j*b_j*c_i*d_i + 2*cp_{6,1}*a_j*b_j*d_i^2 + cp_{7,1}*b_j^2*c_i^2 + 2*cp_{8,1}*b_j^2*c_i*d_i + cp_{9,1}*b_j^2*d_i^2;$ (1)

- $X(i,j)= cp_{1,2}*a_j^2 * c_i^2 + 2*cp_{2,2}*a_j^2 *c_i *d_i + cp_{3,2}*a_j^2 * d_i^2 + 2*cp_{4,2}*a_j*b_j*c_i^2 + 4*cp_{5,2}*a_j*b_j*c_i*d_i + 2*cp_{6,2}*a_j*b_j*d_i^2 + cp_{7,2}*b_j^2*c_i^2 + 2*cp_{8,2}*b_j^2*c_i*d_i + cp_{9,2}*b_j^2*d_i^2;$ (2)

The $p_{i,j}$ are the control points for the Bezier transform, a and b corresponds to the current column index, where a = no_rows – b. Similarly c and d corresponds to the current row index, where c = no_cols – d. This system is currently using 4thequation. The hardware equivalent of the above equations uses multipliers, address and subtractors. During this thesis work, it was observed that, by using algrebriac equalities and properties of binary numbering system, we can convert some multiplication operations into concatenation and subtraction operations, which are shown below.

As $a' = 768 - b'$ (where 768 = no_rows)

By normalizing a the above equation by 768(no_rows)

$a = 1 - b$ (where $a = a'/768$ and $b = b'/768$)

$a^2 = (1-b)^2$

$a^2 = 1 - 2b + b^2$

$a^2 = (1 + b^2) - 2b$ (As $0 = b < 1$, so $1 + b^2 = \{1'b1, b^2\}$ and $2b = \{b, 1'b0\}$)

So $\boxed{a^2 = \{1'b1, b^2\} - \{b, 1'b0\}}$

Similarly $ab = (1 - b)\, b$

$$\boxed{ab = b - b^2}$$

It can be observed that ab and a^2 in the form of b and b^2, where b is the current column index. Similarly, cd and c^2canbe represented in the terms of d and d^2.

As $c' = 1024 - d'$ (where 1024 = no_cols)

By normalizing a the above equation by 1024(no_cols)

$c = 1 - d$ (where $c = c'/1024$ and $d = d'/1024$)

$c^2 = (1-d)^2$

$c^2 = 1 - 2d + d^2$

$c^2 = (1 + d^2) - 2d$ (As $0 = d < 1$, so $1 + d^2 = \{1'd1, d^2\}$ and $2d = \{d, 1'd0\}$)

So $\boxed{c^2 = \{1'd1, d^2\} - \{d, 1'd0\}}$

Similarly $cd = (1 - d)\, d$

$$\boxed{cd = d - d^2}$$

The above equations for Bezier transform along with the above derived equations, can be shown in the graphical format as below. Here the datapath for equation (1) is shown below, for the equation (2) only cp_i coefficients (control points) are replaced.

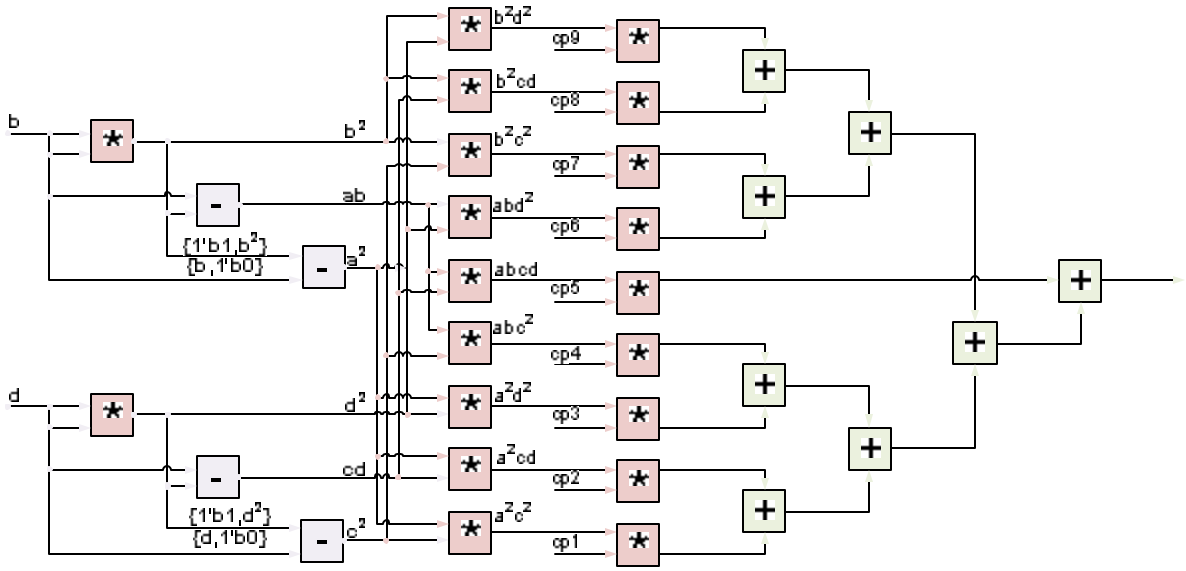

Figure 11: PRU Datapath

The above diagram shows the datapath of PRU module. As this module had to be implemented in FPGA, so all are operations were converted to fixed point format. The below diagram had different sections of datapath highlighted and the fixed point format used for that particular sections is shown at the bottom of the section.

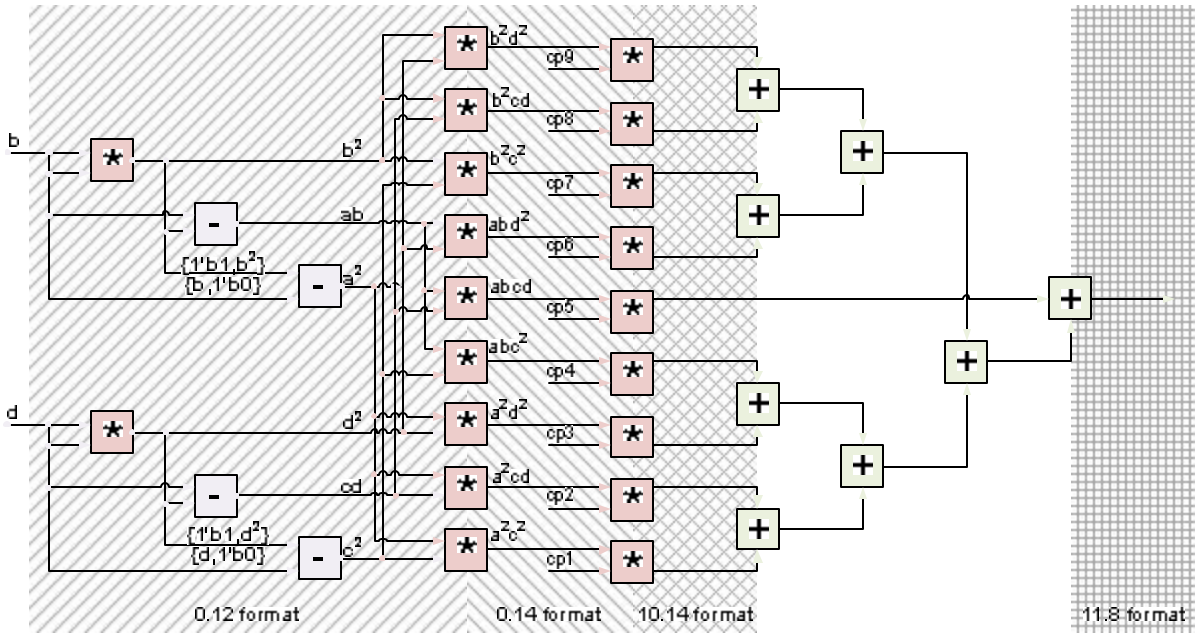

Figure 12: PRU Datapath with Fixed Point

The datapath for PRU module is made such that it can compute one resultant row and column index within one clock cycle. To meet the requirement of single cycle, all the blocks i.e. adders, subtracters and multiplier are made combinational. All the blocks add up and make a large combinational path, whose delay is approximately 50ns. To cut down the combinational delay pipeline stage are inserted and the architecture is made 4 stage pipelined architecture.

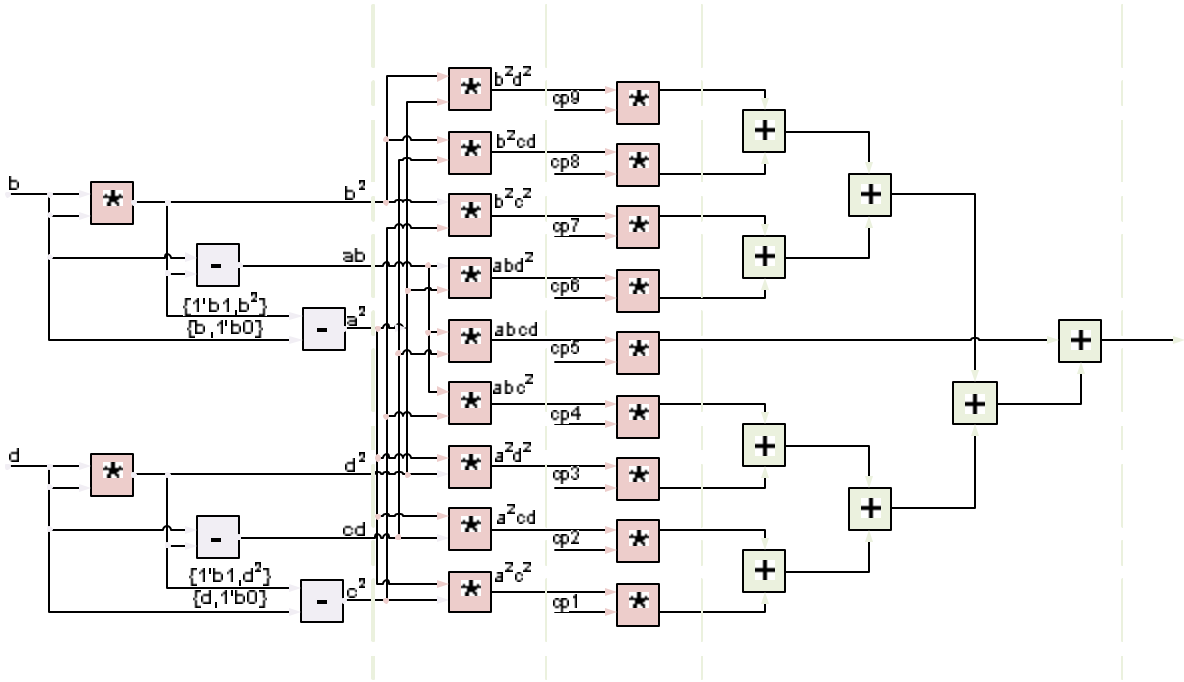

Figure 13: PRU Datapath2

At the each identified point, delay blocks (Registers) are added into the path to cut down the delay. Hence the critical path delay is reduced to approximately 17ns, which ensures that this module can easily operate at 41.25 MHz. 41.25MHz clock can be generated from 165MHz by slowing down it by a factor of 4, which is done in the Parameter Calculation Block module. Following diagram shows the datapath with delay block inserted.

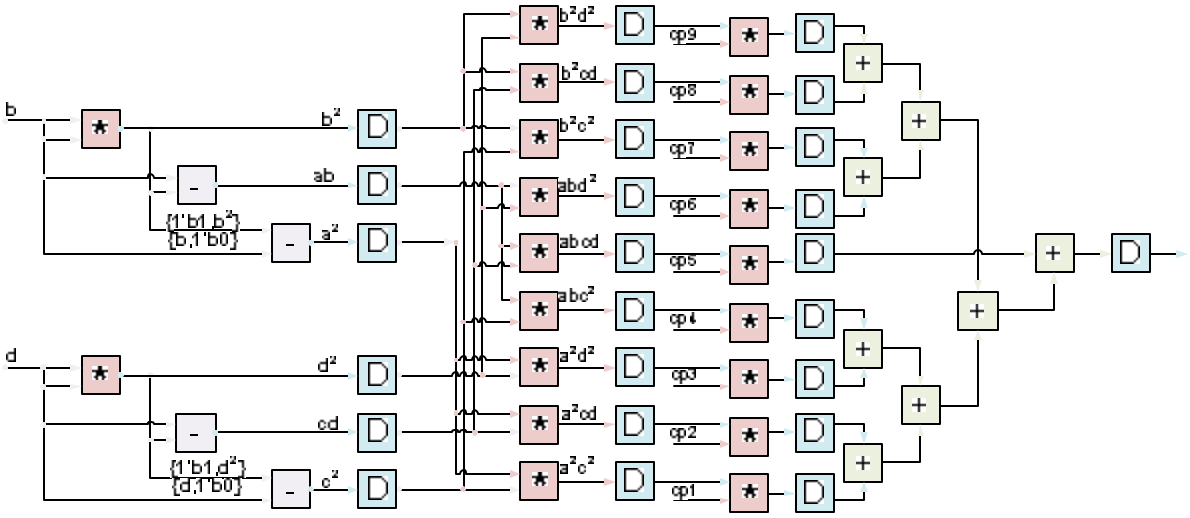

Figure 14: PRU Pipelined Datapath

3.3.5 Interpolator

Interpolator module is used to approximate the output pixel value on the basis of 4 original pixel values. It uses a FIFO to store the data of the previous row, so that it can be used while approximation of the value. Following table describes the interfacing signals of this module.

Signal Name	Width	Direction	Description
clock	1	Input	Input clock signal
reset_n	1	Input	Active low reset
image_data1	8	Input	Incoming image data1
image_data2	8	Input	Incoming image data2
image_data3	8	Input	Incoming image data3
image_data4	8	Input	Incoming image data4
param1	8	Input	Incoming coefficient for interpolation. This coefficient is the fractional part of the row index
Param2	8	Input	Incoming coefficient for interpolation. This coefficient is the fractional part of the column index
data_out	8	Output	Interpolated data output

Table 4: Interpolator interfacing signals

As literature survey suggests bilinear interpolation as most efficient for hardware with significant results both in forms of approximating the value and speed, so bilinear interpolation is used in this module. Bilinear interpolation uses pixels from two rows and two columns, to compute the resultantpixel values, as shown in the below equation.

$$P(x',y') = (1-x_f)(1-y_f)p(x_i,y_i) + (x_f)(1-y_f)p(x_{i+1},y_i) + (1-x_f)(y_f)p(x_i,y_{i+1}) + (x_f)(y_f)p(x_{i+1},y_{i+1}) \quad (3)$$

Equation(3) can be represented in the form of blocks as shown in below figure, here combinational multipliers, adder and subtractors were used to, whichadd up to make a significant combination delay Here param1(x_f) is the fractional part of the calculated column index a and param2(y_f) is the fractional part of the calculated row index and p11, p12, p21, p22 are the pixel values of the four neighbours from the original image.

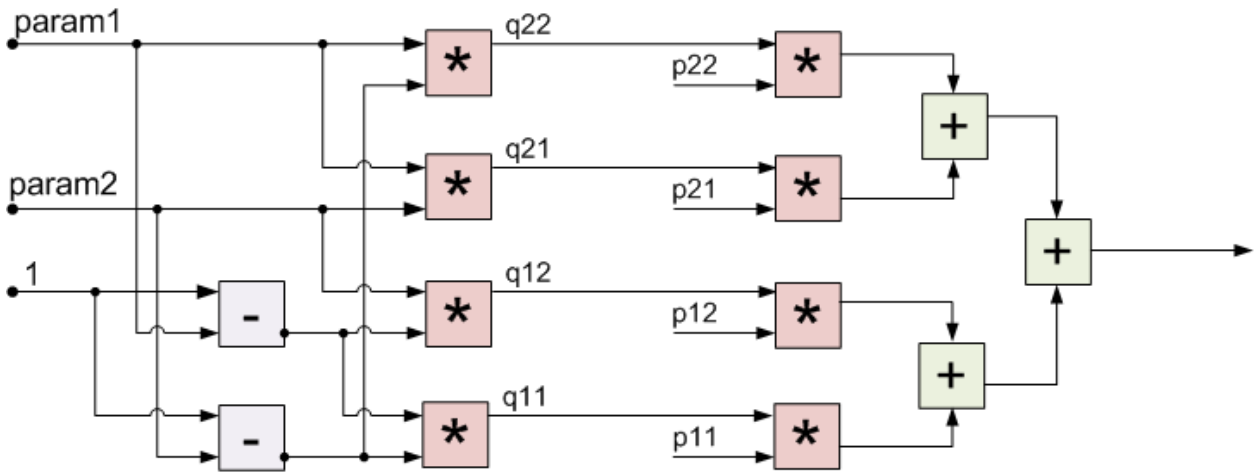

Figure 15: Interpolator Datapath

Again pipeline stages need to be inserted to minimize the critical path delay caused by this block. Hencethe datapath is cut by using 2 stage pipelined architecture. Two cut points were identified and 2 pipelines stages are inserted into the datapath.

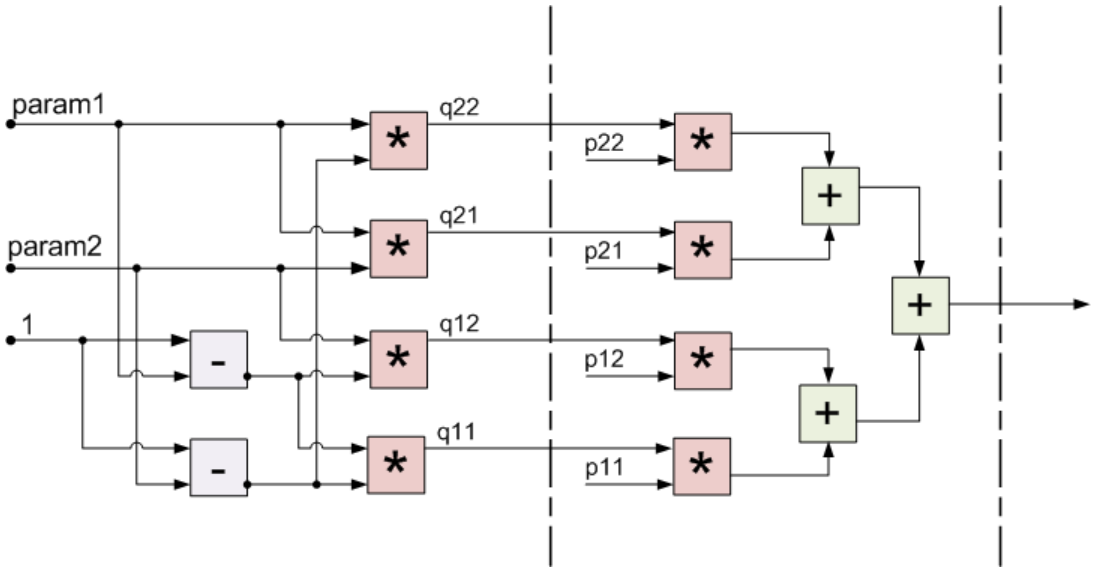

Figure 16: Interpolator Datapath2

Again delay blocks are inserted into the datapath at the identified points. Below diagram shows the datapath with delay blocks.

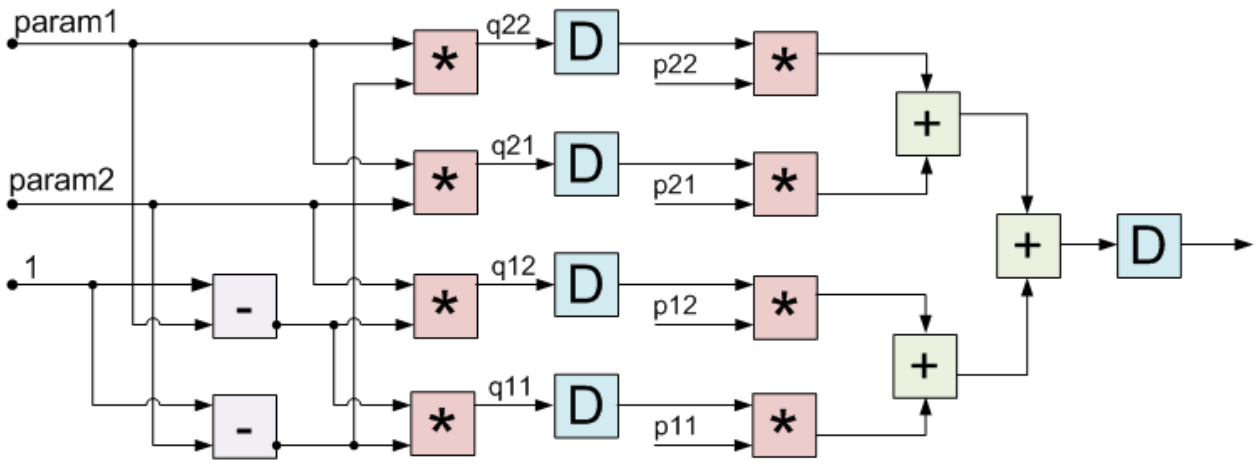

Figure 17: Interpolator Pipelined Datapath

Chapter 4

4 Results, Performance and Discussion

This chapter discusses the results and performance of the system. Current implementation of the system supports resolutions upto 1024*768. As the system uses the pixel clock of 165MHz, so the current architecture can easily work with the resolutions up to 1600*1200 with a maximum frame rate of 75fps, by just increasing the sizes of FIFOs and Memory. Higher resolutions and frame rates can be supported by increasing the input clock frequency and inserting more pipeline stages to the critical paths. For the real time testing Xilinx Spartan 3E FPGA was used. Due to the limitations in the Spartan 3A Kit, the architecture is tested only for one projector and one channel only(grey scale). Below table gives summary of the resource utilization for single projector and one channel.

4.1 Resource Utilization

Following is the resource utilization summary on a Xilinx Spartan3E device.

Selected Device: 3sd3400afg676-4

Number of Slices:	2564 out of 23872 10%
Number of Slice Flip Flops:	2111 out of 47744 4%
Number of 4 input LUTs:	3545 out of 47744 7%
Number used as logic:	3468
Number used as Shift registers:	9
Number used as RAMs:	68
Number of IOs:	133
Number of bonded IOBs:	103 out of 469 21%
Number of BRAMs:	8 out of 126 6%
Number of GCLKs:	3 out of 24 12%
Number of DCMs:	1 out of 8 12%
Number of DSP48s:	126 out of 126 100%

Table 5: Resource Utilization for Single Channel

Above table shows that approximately 10% of the slices are being used. Later on, architecture was synthesized for single projector and three channels (RGB). In this case, only some of the logic in interpolator was replicated and memory width was increased, as memory lies outside FPGA, hence not a significant difference in the resource utilization was noticed. Below table gives a comparison of the resource utilization incase of the single channel and three channels.

Resource Name	Available	One Channel	Three Channels	%age usage	%age diff
Number of Slices:	23872	2564	2916	12.2%	1.47%
Number of Slice Flip Flops:	47744	2111	2447	5.1%	0.7%
Number of 4 input LUTs:	47744	3545	3911	8.2%	0.77%
Number of BRAMs:	126	8	10	7.9%	1.6%
Number of DSP48s:	126	126	126	100%	0%

Table 6: Comparison between Single Channel and Three Channel (RGB) Resource Utilization

It can be noted from the above table that due to the resource sharing inside FPGA, little change was noticed while moving to three channels from single channel. Later on, the architecture was synthesized for two projectors with three channels each; in this case, some of the multipliers and registers in PRU module were shared by both channels and all of the other logic instantiated twice. Below table shows the comparison of the resource utilization incase of the single projector and dual projector system.

Resource Name	Available	Single projector	Dual Projector	%age usage	%age diff
Number of Slices:	23872	2916	8758	36%	24%
Number of Slice Flip Flops:	47744	2447	5049	10%	5%
Number of 4 input LUTs:	47744	3911	14316	29%	21%
Number of BRAMs:	126	10	20	15%	8%
Number of DSP48s:	126	126	126	100%	0%

Table 7: Comparison between Single Projector and Dual Projector System

From the above table, it is obvious that for making system to catering an additional projector, the resource which gets utilized most is the slices and additional projector cost 24% of the available slices. Hence, it can be summarized that we can cater 4 projectors inside a single Spartan 3AFPGA. Comparing with the software based approaches, which typically can only cater 2 projectors/GPU card (CPU system) unless an expensive GPU is used. Thus the proposed scheme allowing 4 projectors/Spartan3A FPGA, with each having resolution up-to 1600*1200 with 75fps provides a cost efficient and high resolution solution.

Chapter 5

5 Conclusion

Due to extensive computations, geometric correction is one of main tasks in the process of matching the projected images to the curved projected surfaces. Most of the image correction systems use GPUs for achieving the required results, which makes the correction system platform dependent and affect the system performance. In this thesis, a hardware based solution for geometric correction is developed. This system can be attached to any DVI connector, hence making it platform independent. Several optimizations were proposed like algorithmic optimization for avoiding multiplication, row caching for better memory management, pipelining for higher bandwidth, resource sharing.

The proposed architecture can apply geometric correction to 4 parallel video streams in a single FPGA in real time, which can operate at frame rates of 75fps with maximum resolution of 1600x1200 for each video stream. This architecture can be used to construct 4x1 array for panoramic view, or 2x2 array for high resolution output (3200x2400).

Chapter 6

6 Future Work

In this thesis, Spartan 3A XtremeDSP Video Starter Kit was used to testing the proposed architecture. Due to the limitations of the memory available on kit, the design was testedin real world environment with single channel. A customized hardware specific for this solution can be worked out, which will apply the correction on all three channels and can cater multiple video streams.

Higher order Bezier transformation can be achieved, by working around the proposed system. Current implementation applies first order Bezier transformation;

A system for real-time photometric correction can also be worked out as current implementation applies only geometric correction on the incoming video stream.

With the implementation of interpolator and video splitter module at the DVI In, high resolution (upto 3200x2400) display from an ordinary video source (1600X1200) can be constructed.

7 References

1. Michael Harville, Bruce Culbertson, Irwin Sobel, Dan Gelb, Andrew Fitzhugh, Donald Tanguay. ”Practical Methods for Geometric and Photometric Correction of Tiled Projector Displays on Curved Surfaces”, IEEE 2006.
2. H. K. RAMAPRIYAN, “Data Handling for the Geometric Correction of Large Images”, IEEE 1977
3. Toru Takahashi, Norihito Numa, Takafumi Aoki and Satoshi Kondo, “A Geometric Correction Method for Projected Images Using SIFT Feature Points”, ACM 2008.
4. Ramesh Raskar, Jeroen van Baar, Paul Beardsley, Thomas Willwacher, Srinivas Rao and Clifton Forlines, “iLamps: Geometrically Aware and Self-Configuring Projectors”, ACM 2003.
5. David Eadie, Fergal Shevlin, Andy Nisbet, “Correction of Geometric Image Distortion Using FPGAs”
6. Lin Qiang and Nigel M Allinson, “FPGA Implementation of Pipelined Architecture for Optical Imaging Distortion Correction”, IEEE 2006.
7. Michael Brown, AditiMajumder, and Ruigang Yang, “Camera-Based Calibration Techniques for Seamless Multiprojector Displays”, IEEE 2005.
8. Jay P. Kapur, Masters Project Report, “Geometric Correction of SEM Images”.
9. Yau-Chat Tsoi and Michael S. Brown, “Geometric and Shading Correction for Images of Printed MaterialsA Unified Approach Using Boundary”, IEEE 2004.
10. M. Brown,W. B. Seales. “A practical and flexible tiled display system.” In Pacific Conf. on Comp. Graphics and Apps, 2002.
11. N. Chang. “Efficient dense correspondences using temporally-encoded light patterns.” In PROCAMS’03.
12. A. Raij, G. Gill, A. Majumder, H. Towles, H. Fuchs. “PixelFlex2:a comprehensive, automatic, causally-aligned multiprojector display”. In PROCAMS’03.
13. Frank Crosby, “Geometric Correction through Complex Interpolation”, IEEE 2002.
14. Yong Li and Daiyin Zhu,” The Geometric-Distortion Correction Algorithm for Circular-Scanning SAR Imaging”, IEEE 2009.
15. Thomas M. Lehmann, Claudia Gonner, and Klaus Spitzer, “Survey: Interpolation Methods in Medical Image Processing”, IEEE 1999.
16. K. W. Simon, “Digital Image Reconstruction and Resampling for Geometric Manipulation”, IEEE 1975.
17. Frank Crosby, “Geometric Correction through Complex Interpolation”, IEEE 2002.
18. Denis Zorin and Alan H. Barr, “Correction of Geometric Perceptual Distortions in Pictures”, ACM 1995.
19. Hernandez, A. “Real-Time Image Distortion Correction using FPGA-based System”, IEEE 2006.

20. Jack E. Volder, The CORDIC Trigonometric Computing Technique, IRE Transactions on Electronic Computers, pp330-334, September 1959
21. http://en.wikipedia.org/wiki/B%C3%A9zier_surface
22. http://en.wikipedia.org/wiki/Homography
23. Atif Ahmed, Rehan Hafiz, "Bezier based geometrical correction for quadratic surface using uncalliberated camera", Internal report, VISPro Lab, SEECS.
24. Atif Ahmed, Rehan Hafiz, Muhammad Murtaza Khan, "Geometric Correction for Uneven Quadric Projection Surfaces based on Recursive Sub-Division of Bezier Patches using a Single Un-Calibrated Camera", Internal Project Report VISPro Lab SEECS, 2011

Appendix

Appendix

A. Matlab Codes

A.1 Matlab Code with Pre-calculated Parameters

```
close all;
clear all;
clc;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

image1 = imread(...
'C:/Documents and Settings/Kamran/Desktop/download2_1024_768.jpg');
path = 'C:/Documents and Settings/Kamran/Desktop/bez1024768.txt';
output_filename= ...
'C:/Documents and Settings/Kamran/Desktop/download2_1024_768_output.jpg';
fid = fopen(path,'r');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create filter matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
filter_size = 3;
filter_coff = create_filter(filter_size);

% filter_coff = [0.3536 0.4472 0.5 0.4472 0.3536;
%     0.4472 0.7071 1 0.7071 0.4472;
%     0.5 1 0 1 0.5;
%     0.4472 0.7071 1 0.7071 0.4472;
%     0.3536 0.4472 0.5 0.4472 0.3536;];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% image parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
no_rows = 768;
no_cols = 1024;
factor = 2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read transformation table
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
new_data = fscanf(fid,'%g,%g');
i = 1:1:size(new_data)/2;
new_data2(1,i) = new_data(i*2-1);
new_data2(2,i) = new_data(i*2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Process transformation matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j = 1:no_rows
```

```
for k = 0:no_cols - 1
      new_data3_1(j,k+1) = new_data2(1,k*768+j);
end
end

for j = 1:no_rows
for k = 0:no_cols - 1
      new_data3_2(j,k+1) = new_data2(2,k*768+j);
end
end

new_data3_1 = new_data3_1 * no_cols;
new_data3_2 = new_data3_2 * no_rows;

fclose(fid);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% transfrom the image using transformation matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
image_output = uint8(zeros(no_rows,no_cols,3));
image_output2 = uint8(zeros(factor * no_rows,factor * no_cols));

image_output(factor * no_rows,factor * no_cols,3) = uint8(0);

for j = 1:no_rows
for k = 1:no_cols
row = factor * no_rows - round(factor * (new_data3_2(j,k)));
col = factor * round(new_data3_1(j,k));
if((row < factor * no_rows + 1 && row > 0) &&...
                    (col < factor * no_cols + 1 && col > 0))
image_output(row, col,1) = image1(j, k,1);
image_output(row, col,2) = image1(j, k,2);
image_output(row, col,3) = image1(j, k,3);
          image_output2(row, col) = image_output2(row, col) + 1;
end
end
end
% image_output = image_output * 50;
image_output = uint8(image_output);
imwrite(image_output, output_filename,'JPEG');

%imshow(image_output);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Process channel 1 by interpolation filter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

display('Processing Channel 1 \n');
image_output2 = image_output(:,:,1);

[no_rows2, no_cols2] = size(image_output2);



image_output3 = [zeros(filter_size,no_cols2);image_output2;...
zeros(filter_size,no_cols2)];
image_output3 = [zeros(no_rows2+2*filter_size,filter_size) ...
```

```
image_output3 zeros(no_rows2+2*filter_size,filter_size)];
image_output4(:,:,1) = image_output3;

for j = filter_size  + 1 : no_rows2+filter_size
for k = filter_size + 1 : no_cols2+filter_size
if(image_output3(j,k)==0)
%for l = 1:3
temp = double(image_output3(j-filter_size:...
j+filter_size,k-filter_size:k+filter_size));
              sum1 = sum(sum(temp.*filter_coff));
              temp2 = ~(temp == 0);
              sum2 = sum(sum(temp2.*filter_coff));
              image_output4(j,k,1) = round(sum1/sum2);

%end
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Process channel 2 by interpolation filter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
display('Processing Channel 2 \n');
image_output2 = image_output(:,:,2);

[no_rows2, no_cols2] = size(image_output2);

image_output3 = [zeros(filter_size,no_cols2);image_output2;...
zeros(filter_size,no_cols2)];
image_output3 = [zeros(no_rows2+2*filter_size,filter_size) ...
image_output3 zeros(no_rows2+2*filter_size,filter_size)];
image_output4(:,:,2) = image_output3;

for j = filter_size + 1 : no_rows2+filter_size
for k = filter_size + 1 : no_cols2+filter_size
if(image_output3(j,k)==0)
%for l = 1:3
temp = double(image_output3(j-filter_size:...
j+filter_size,k-filter_size:k+filter_size));
              sum1 = sum(sum(temp.*filter_coff));
              temp2 = ~(temp  == 0);
              sum2 = sum(sum(temp2.*filter_coff));
              image_output4(j,k,2) = round(sum1/sum2);

%end
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Process channel 3 by interpolation filter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

display('Processing Channel 3 \n');
image_output2 = image_output(:,:,3);
```

```
[no_rows2, no_cols2] = size(image_output2);



image_output3 = [zeros(filter_size,no_cols2);image_output2;...
zeros(filter_size,no_cols2)];
image_output3 = [zeros(no_rows2+2*filter_size,filter_size) ...
image_output3 zeros(no_rows2+2*filter_size,filter_size)];
image_output4(:,:,3) = image_output3;

for j = filter_size + 1 : no_rows2+filter_size
for k = filter_size + 1 : no_cols2+filter_size
if(image_output3(j,k)==0)
temp = double(image_output3(j-filter_size:j+filter_size,...
k-filter_size:k+filter_size));
          sum1 = sum(sum(temp.*filter_coff));
          temp2 = ~(temp == 0);
          sum2 = sum(sum(temp2.*filter_coff));
          image_output4(j,k,3) = round(sum1/sum2);
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Downsample the output image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
image_output5 = image_output4(filter_size+1:no_rows2+filter_size,...
              filter_size+1:no_cols2+filter_size,:);
rows_filter = 1:2:no_rows2;
cols_filter = 1:2:no_cols2;

image_output6 = image_output5(rows_filter,cols_filter,:);
index = (image_output6(:,:,1)==0);
%image_output6 = image_output5(rows_filter,cols_filter,:);
image_output6(:,:,1) = image_output6(:,:,1) + uint8(255.*index);
image_output6(:,:,2) = image_output6(:,:,2) + uint8(255.*index);
image_output6(:,:,3) = image_output6(:,:,3) + uint8(255.*index);
imshow(image_output6);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
functionfilter_coeff = create_filter(filter_size)

row = 1;
col = 1;

fori = -1 * filter_size : filter_size
col = 1;
for j = -1 * filter_size : filter_size
if(i ==0 && j==0)
filter_coeff(row,col) = 0;
else
filter_coeff(row,col) = 1/sqrt(i^2+j^2);
end
col = col + 1;
end
row = row + 1;
```

```
end

end
```

A.2 Matlab Code with Param Calculation

```
HomographyMatrix =      [1 0 10
                         0 1 10
                         0 0  1];

ControlPoint = [1                1;
            351.854634052214     10.1406410187149;
            768              1;
            -2.78956151874882    502.286460227351;
            410.261027861973     484.765595179795;
            827.501390247189     445.481022535271;
            1                1024;
            303.253314330282     1107.08502900172;
            768              1024];

%***************************** Code ************************************%

siz=[(ControlPoint(9,1)-ControlPoint(1,1)+1),(ControlPoint(9,2)-ControlPoint(1,2)+1)];
stepx = siz(1)-1;
stepy = siz(2)-1;

%%  parametric variables declaration
a = 1:-1/(stepy):0;
c = 1:-1/(stepx):0;
b = 1-a;
d = 1-c;

%%  Ideal control points location
p1 = ControlPoint(1,:);
p2 = ControlPoint(2,:);
p3 = ControlPoint(3,:);
p4 = ControlPoint(4,:);
p5 = ControlPoint(5,:);
p6 = ControlPoint(6,:);
p7 = ControlPoint(7,:);
p8 = ControlPoint(8,:);
p9 = ControlPoint(9,:);
X = zeros(stepx+1,stepy+1);
Y = zeros(stepx+1,stepy+1);

%s=size(I);

%image1 = imread('C:/Users/Trentino_1/Desktop/data_related_to_thesis/ec-1024x768-desktoppictures2.jpg');
image1 = imread('C:/Users/Trentino_1/Desktop/data_related_to_thesis/test.jpg');
output_filename = 'C:/Users/Trentino_1/Desktop/data_related_to_thesis/ec-1024x768-desktoppictures_output.jpg';

output_data = ones(siz(1),siz(2))*255;
%%  Bezier surface and transformation
fori=1:(siz(1))
for j=1:(siz(2))
```

```
    Y(i,j)= p1(1)*(a(j)^2)*(c(i)^2) + p2(1)*2*(a(j)^2)*c(i)*d(i) + p3(1)*(a(j)^2)*(d(i)^2) ...
        + p4(1)*2*a(j)*b(j)*(c(i)^2) + p5(1)*4*a(j)*b(j)*c(i)*d(i) + p6(1)*2*a(j)*b(j)*(d(i)^2) ...
        + p7(1)*(b(j)^2)*(c(i)^2) + p8(1)*2*(b(j)^2)*c(i)*d(i) + p9(1)*(b(j)^2)*(d(i)^2);

   X(i,j)= p1(2)*(a(j)^2)*(c(i)^2) + p2(2)*2*(a(j)^2)*c(i)*d(i) + p3(2)*(a(j)^2)*(d(i)^2) ...
        + p4(2)*2*a(j)*b(j)*(c(i)^2) + p5(2)*4*a(j)*b(j)*c(i)*d(i) + p6(2)*2*a(j)*b(j)*(d(i)^2) ...
        + p7(2)*(b(j)^2)*(c(i)^2) + p8(2)*2*(b(j)^2)*c(i)*d(i) + p9(2)*(b(j)^2)*(d(i)^2);

      if((Y(i,j) < 767 && Y(i,j) >= 0) && (X(i,j) < 1024 && X(i,j) >= 0))

x_int = floor(X(i,j));
y_int = floor(Y(i,j));

x_frac = X(i,j) - x_int;
y_frac = Y(i,j) - y_int;

x_int = x_int + 1;
y_int = y_int + 1;

          q11 = (1 - y_frac) * (1 - x_frac);
          q12 = (1 - y_frac) * (x_frac);
          q21 = (y_frac) * (1 - x_frac);
          q22 = (y_frac) * (x_frac);

          p11 = image1(y_int, x_int);
          p12 = image1(y_int, x_int + 1);
          p21 = image1(y_int + 1, x_int);
          p22 = image1(y_int + 1, x_int + 1);

output_data(i,j) = p11 * q11 + p12 * q12 + p21 * q21 + p22 * q22;

end

end
end
output_data = uint8(output_data);
imshow(output_data);
```

A.3 Matlab Fixed Point Implementation

```
%image1 = imread('C:/Users/Trentino_1/Desktop/data_related_to_thesis/ec-1024x768-desktoppictures2.jpg');
 image1 = imread('C:/Users/Trentino_1/Desktop/data_related_to_thesis/test.jpg');
output_filename = 'C:/Users/Trentino_1/Desktop/data_related_to_thesis/ec-1024x768-desktoppictures_output.jpg';
fori=1:768
for j=1:1024
      [row_m(i,j),col_m(i,j)] = param_calculator(i,j);
end
end
output_data = ones(768,1024)*255;
fori=1:768
for j=1:1024
%[row1,col1] = param_calculator(i,j);
      row1 = row_m(i,j);
      col1 = col_m(i,j);
if((row1 < 767 && row1 >= 0) && (col1 < 1024 && col1 >= 0))

x_int = floor(col1);
y_int = floor(row1);

x_frac = col1 - x_int;
y_frac = row1 - y_int;

x_int = x_int + 1;

y_int = y_int + 1;

          q11 = (1 - y_frac) * (1 - x_frac);
          q12 = (1 - y_frac) * (x_frac);
          q21 = (y_frac) * (1 - x_frac);
          q22 = (y_frac) * (x_frac);

          p11 = image1(y_int, x_int);
          p12 = image1(y_int, x_int + 1);
          p21 = image1(y_int + 1, x_int);
          p22 = image1(y_int + 1, x_int + 1);

output_data(i,j) = p11 * q11 + p12 * q12 + p21 * q21 + p22 * q22;

end
end
end
output_data = uint8(output_data);
imshow(output_data);
```

A.4 Param Calculation Unit Fixed Point Implementation

```
function [row1,col1] = param_calculator(row,col)

ControlPoint = [1                1;
            351.854634052214    10.1406410187149;
            768              1;
```

```
            2.78956151874882    502.286460227351;
            410.261027861973    484.765595179795;
            827.501390247189    445.481022535271;
            1                   1024;
            303.253314330282    1107.08502900172;
            768                 1024];


cp = floor(ControlPoint.*1024);      %10.10 format

d = floor(row*1.3333);               %0.10 format
b = floor(col);                      %0.10 format

a = floor((1024 - col));             %0.10 format
c = floor((1024 - row*1.3333));      %0.10 format

b2 = floor(b*b/256);                 %0.12 format
d2 = floor(d*d/256);                  %0.12 format

a2 = 4096+b2-8*b;                    %0.10 format
c2 = 4096+d2-8*d;                    %0.10 format
% a2 = floor((a*a)/256);             %0.12 format
% c2 = floor((c*c)/256);             %0.12 format

ab = 4*b-b2;                         %0.10 format
cd = 4*d-d2;                         %0.10 format

% ab = floor(a*b/256);               %0.12 format
% cd = floor(c*d/256);               %0.12 format

temp1(1) = floor(a2*c2/1024);        %0.14 format
temp1(2) = floor(a2*cd/1024);
temp1(3) = floor(a2*d2/1024);
temp1(4) = floor(ab*c2/1024);
temp1(5) = floor(ab*cd/1024);
temp1(6) = floor(ab*d2/1024);
temp1(7) = floor(b2*c2/1024);
temp1(8) = floor(b2*cd/1024);
temp1(9) = floor(b2*d2/1024);

temp2(1) = floor(temp1(1)*cp(1,1)/1024);    %10.14 format
temp2(2) = floor(temp1(2)*cp(2,1)/1024);
temp2(3) = floor(temp1(3)*cp(3,1)/1024);
temp2(4) = floor(temp1(4)*cp(4,1)/1024);
temp2(5) = floor(temp1(5)*cp(5,1)/1024);
temp2(6) = floor(temp1(6)*cp(6,1)/1024);
temp2(7) = floor(temp1(7)*cp(7,1)/1024);
temp2(8) = floor(temp1(8)*cp(8,1)/1024);
temp2(9) = floor(temp1(9)*cp(9,1)/1024);

temp3(1) = floor(temp1(1)*cp(1,2)/1024);    %10.14 format
temp3(2) = floor(temp1(2)*cp(2,2)/1024);
temp3(3) = floor(temp1(3)*cp(3,2)/1024);
temp3(4) = floor(temp1(4)*cp(4,2)/1024);
temp3(5) = floor(temp1(5)*cp(5,2)/1024);
temp3(6) = floor(temp1(6)*cp(6,2)/1024);
```

```
temp3(7) = floor(temp1(7)*cp(7,2)/1024);
temp3(8) = floor(temp1(8)*cp(8,2)/1024);
temp3(9) = floor(temp1(9)*cp(9,2)/1024);

row_int = temp2(1) + 2*temp2(2) + temp2(3) - ...
        2*temp2(4) + 4*temp2(5) + 2*temp2(6) + ...   %12.14 format
temp2(7) + 2*temp2(8) + temp2(9);

col_int = temp3(1) + 2*temp3(2) + temp3(3) + ...
        2*temp3(4) + 4*temp3(5) + 2*temp3(6) + ...   %12.14 format
temp3(7) + 2*temp3(8) + temp3(9);

row1 = (row_int/16384);                              %12.14 format
col1 = (col_int/16384);
end
```

B. Verilog Codes

B.1 Single Parameter Calculator

```
module single_parameter_calculator(
        input     clock,
        inputreset_n,
        input [9:0] row,
        input [9:0] col,
        output [10:0] row_out,
        output [10:0] col_out,
        output   [7:0] row_frac,
        output   [7:0] col_frac
   );

        wire [9:0] b,d;
        reg [11:0] a2,b2,c2,d2,ab,cd;

        wire [19:0] temp;
        wire [19:0] temp1, temp2;

        wire [23:0] temp11;
        wire [23:0] temp12;
        wire [23:0] temp13;
        wire [23:0] temp14;
        wire [23:0] temp15;
        wire [23:0] temp16;
        wire [23:0] temp17;
        wire [23:0] temp18;
        wire [23:0] temp19;

        reg [13:0] temp11_reg;
        reg [13:0] temp12_reg;
```

```
reg [13:0] temp13_reg;
reg [13:0] temp14_reg;
reg [13:0] temp15_reg;
reg [13:0] temp16_reg;
reg [13:0] temp17_reg;
reg [13:0] temp18_reg;
reg [13:0] temp19_reg;

wire [34:0] temp21;
wire [34:0] temp22;
wire [34:0] temp23;
wire [34:0] temp24;
wire [34:0] temp25;
wire [34:0] temp26;
wire [34:0] temp27;
wire [34:0] temp28;
wire [34:0] temp29;

reg [23:0] temp21_reg;
reg [23:0] temp22_reg;
reg [23:0] temp23_reg;
reg [23:0] temp24_reg;
reg [23:0] temp25_reg;
reg [23:0] temp26_reg;
reg [23:0] temp27_reg;
reg [23:0] temp28_reg;
reg [23:0] temp29_reg;

wire [34:0] temp31;
wire [34:0] temp32;
wire [34:0] temp33;
wire [34:0] temp34;
wire [34:0] temp35;
wire [34:0] temp36;
wire [34:0] temp37;
wire [34:0] temp38;
wire [34:0] temp39;

reg [23:0] temp31_reg;
reg [23:0] temp32_reg;
reg [23:0] temp33_reg;
reg [23:0] temp34_reg;
reg [23:0] temp35_reg;
reg [23:0] temp36_reg;
reg [23:0] temp37_reg;
reg [23:0] temp38_reg;
reg [23:0] temp39_reg;

reg [18:0] row_reg, col_reg;

wire [27:0] row_int,col_int;
```

```
//          wire [8:0] row_frac,col_frac;

        parameter [20:0] cp11 = 21'h000400;
        parameter [20:0] cp12 = 21'h057F6B;
        parameter [20:0] cp13 = 21'h0C0000;
        parameter [20:0] cp14 = 21'h000B29;
        parameter [20:0] cp15 = 21'h06690B;
        parameter [20:0] cp16 = 21'h0CEE01;
        parameter [20:0] cp17 = 21'h000400;
        parameter [20:0] cp18 = 21'h04BD03;
        parameter [20:0] cp19 = 21'h0C0000;

        parameter [20:0] cp21 = 21'h000400;
        parameter [20:0] cp22 = 21'h002890;
        parameter [20:0] cp23 = 21'h000400;
        parameter [20:0] cp24 = 21'h07D925;
        parameter [20:0] cp25 = 21'h079310;
        parameter [20:0] cp26 = 21'h06F5ED;
        parameter [20:0] cp27 = 21'h100000;
        parameter [20:0] cp28 = 21'h114C57;
        parameter [20:0] cp29 = 21'h100000;

        assign temp = 10'h555 * (row+10);
        assign b = col+10;
        assign d = temp[17:8];
        assign temp1 = b * b;
        assign temp2 = d * d;

        always@(posedge clock or negedge reset_n)
        begin
                if(~reset_n) begin
                        b2              <= 10'b0;
                        a2              <= 10'b0;
                        d2              <= 10'b0;
                        c2              <= 10'b0;
                        ab              <= 10'b0;
                        cd              <= 10'b0;
                end
                else begin
                        b2              <= temp1[19:8];
                        a2              <= {1'b1,temp1[19:8]} - {b[9:0],3'b000};
                        d2              <= temp2[19:8];
                        c2              <= {1'b1,temp2[19:8]} - {d[9:0],3'b000};
                        ab              <= {b,2'b00} - temp1[19:8];
                        cd              <= {d,2'b00} - temp2[19:8];
                end
        end

        assign temp11 = a2 * c2;
        assign temp12 = a2 * cd;
        assign temp13 = a2 * d2;
```

```
	assign temp14 = ab * c2;
	assign temp15 = ab * cd;
	assign temp16 = ab * d2;
	assign temp17 = b2 * c2;
	assign temp18 = b2 * cd;
	assign temp19 = b2 * d2;

	always@(posedge clock or negedge reset_n)
	begin
		if(~reset_n) begin
			temp11_reg	<= 14'b0;
			temp12_reg	<= 14'b0;
			temp13_reg	<= 14'b0;
			temp14_reg	<= 14'b0;
			temp15_reg	<= 14'b0;
			temp16_reg	<= 14'b0;
			temp17_reg	<= 14'b0;
			temp18_reg	<= 14'b0;
			temp19_reg	<= 14'b0;
		end
		else begin
			temp11_reg	<= temp11[23:10];
			temp12_reg	<= temp12[23:10];
			temp13_reg	<= temp13[23:10];
			temp14_reg	<= temp14[23:10];
			temp15_reg	<= temp15[23:10];
			temp16_reg	<= temp16[23:10];
			temp17_reg	<= temp17[23:10];
			temp18_reg	<= temp18[23:10];
			temp19_reg	<= temp19[23:10];
		end
	end

	assign temp21 = cp11 * temp11_reg;
	assign temp22 = cp12 * temp12_reg;
	assign temp23 = cp13 * temp13_reg;
	assign temp24 = cp14 * temp14_reg;
	assign temp25 = cp15 * temp15_reg;
	assign temp26 = cp16 * temp16_reg;
	assign temp27 = cp17 * temp17_reg;
	assign temp28 = cp18 * temp18_reg;
	assign temp29 = cp19 * temp19_reg;

	always@(posedge clock or negedge reset_n)
	begin
		if(~reset_n) begin
			temp21_reg	<= 24'b0;
			temp22_reg	<= 24'b0;
			temp23_reg	<= 24'b0;
			temp24_reg	<= 24'b0;
			temp25_reg	<= 24'b0;
```

```
                        temp26_reg      <= 24'b0;
                        temp27_reg      <= 24'b0;
                        temp28_reg      <= 24'b0;
                        temp29_reg      <= 24'b0;
                end
                else begin
                        temp21_reg      <= temp21[34:11];
                        temp22_reg      <= temp22[34:11];
                        temp23_reg      <= temp23[34:11];
                        temp24_reg      <= temp24[34:11];
                        temp25_reg      <= temp25[34:11];
                        temp26_reg      <= temp26[34:11];
                        temp27_reg      <= temp27[34:11];
                        temp28_reg      <= temp28[34:11];
                        temp29_reg      <= temp29[34:11];
                end
        end

        assignrow_int = ((temp21_reg + {temp22_reg,1'b0}) + (temp23_reg - {temp24_reg,1'b0})) +
                                                                ((({temp26_reg,1'b0} + temp27_reg) +
(({temp28_reg,1'b0} + temp29_reg)) +
                                                                {temp25_reg,2'b00};

        assign temp31 = cp21 * temp11_reg;
        assign temp32 = cp22 * temp12_reg;
        assign temp33 = cp23 * temp13_reg;
        assign temp34 = cp24 * temp14_reg;
        assign temp35 = cp25 * temp15_reg;
        assign temp36 = cp26 * temp16_reg;
        assign temp37 = cp27 * temp17_reg;
        assign temp38 = cp28 * temp18_reg;
        assign temp39 = cp29 * temp19_reg;

        always@(posedge clock or negedge reset_n)
        begin
                if(~reset_n) begin
                        temp31_reg      <= 24'b0;
                        temp32_reg      <= 24'b0;
                        temp33_reg      <= 24'b0;
                        temp34_reg      <= 24'b0;
                        temp35_reg      <= 24'b0;
                        temp36_reg      <= 24'b0;
                        temp37_reg      <= 24'b0;
                        temp38_reg      <= 24'b0;
                        temp39_reg      <= 24'b0;
                end
                else begin
                        temp31_reg      <= temp31[34:11];
                        temp32_reg      <= temp32[34:11];
                        temp33_reg      <= temp33[34:11];
                        temp34_reg      <= temp34[34:11];
```

```
                        temp35_reg      <= temp35[34:11];
                        temp36_reg      <= temp36[34:11];
                        temp37_reg      <= temp37[34:11];
                        temp38_reg      <= temp38[34:11];
                        temp39_reg      <= temp39[34:11];
                end
        end

        assigncol_int = ((temp31_reg + {temp32_reg,1'b0}) + (temp33_reg + {temp34_reg,1'b0})) +
                                                                ((({temp36_reg,1'b0} + temp37_reg) +
({temp38_reg,1'b0} + temp39_reg)) +
                                                                {temp35_reg,2'b00};

        always@(posedge clock or negedge reset_n)
        begin
                if(~reset_n) begin
                        row_reg         <= 19'b0;
                        col_reg <= 19'b0;
                end
                else begin
                        row_reg         <= row_int[23:5];
                        col_reg <= col_int[23:5];
                end
        end

        assignrow_out = row_reg[18:8];
        assigncol_out = col_reg[18:8];

        assignrow_frac = row_reg[7:0];
        assigncol_frac = col_reg[7:0];

endmodule
```

B.2 Parameter Calculation Block

```
moduleparameter_calculation_block(
        input clock,
        inputcount_enable,
        inputreset_n,
        input    [17:0] input_addr,
        output [37:0] data_out,
        outputIRU_done,
        outputregoutput_enable
  );

        reg [17:0]      counter_out;
        reg [3:0]       count_enable_reg;
        reg                     clock_d2;
        reg                     clock_d4;
```

```
	wire [37:0] data1_out;
	wire [37:0] data2_out;
	wire [37:0] data3_out;
	wire [37:0] data4_out;

	always@(posedge clock or negedge reset_n)
	begin
		if(~reset_n)
			clock_d2	<= 1'b0;
		else
			clock_d2	<= ~clock_d2;
	end

	always@(posedge clock_d2 or negedge reset_n)
	begin
		if(~reset_n)
			clock_d4	<= 1'b0;
		else
			clock_d4	<= ~clock_d4;
	end

	single_parameter_calculator inst1 (
.clock		(clock_d4),
.reset_n	(reset_n),
.row			(counter_out[17:8]),
.col			({counter_out[7:0],2'b00}),
.row_out	(data1_out[37:27]),
.col_out	(data1_out[18:8]),
.row_frac	(data1_out[26:19]),
.col_frac	(data1_out[7:0])
);

	single_parameter_calculator inst2 (
.clock		(clock_d4),
.reset_n	(reset_n),
.row			(counter_out[17:8]),
.col			({counter_out[7:0],2'b01}),
.row_out	(data2_out[37:27]),
.col_out	(data2_out[18:8]),
.row_frac	(data2_out[26:19]),
.col_frac	(data2_out[7:0])
);

	single_parameter_calculator inst3 (
.clock		(clock_d4),
.reset_n	(reset_n),
.row			(counter_out[17:8]),
.col			({counter_out[7:0],2'b10}),
```

```
.row_out      (data3_out[37:27]),
.col_out      (data3_out[18:8]),
.row_frac     (data3_out[26:19]),
.col_frac     (data3_out[7:0])
);

      single_parameter_calculator inst4 (
.clock        (clock_d4),
.reset_n      (reset_n),
.row                (counter_out[17:8]),
.col                ({counter_out[7:0],2'b11}),
.row_out      (data4_out[37:27]),
.col_out      (data4_out[18:8]),
.row_frac     (data4_out[26:19]),
.col_frac     (data4_out[7:0])
);

      PCB_fifoyour_instance_name (
        .rst                (~reset_n),       // input rst
        .wr_clk             (clock_d4),       // input wr_clk
        .rd_clk             (clock),          // input rd_clk
        .din                ({data1_out,data2_out,data3_out,data4_out}), // input [151 : 0] din
        .wr_en              (fifo_wren),      // input wr_en
        .rd_en              (fifo_rden),      // input rd_en
        .dout               (data_out),       // output [37 : 0] dout
        .full               (full),           // output full
        .empty              (empty)           // output empty
      );
      always@(posedge clock or negedge reset_n)
      begin
            if(~reset_n)
                  output_enable  <= 1'b0;
            else
                  output_enable  <= ~empty;
      end

      assignfifo_rden = ~empty;

      always@(posedge clock_d4 or negedge reset_n)
      begin
            if(~reset_n)
                  count_enable_reg      <= 4'b0000;
            else
                  count_enable_reg      <= {count_enable_reg[2:0],count_enable};
      end
      assignfifo_wren = count_enable_reg[3];
      always@(*)
            counter_out     <= input_addr;
```

```
        assignIRU_done = ((counter_out == 18'h2FFFF) &count_enable)?1'b1:1'b0;
endmodule
```

B.3 Interpolator

```
module Interpolator(
        input                   clock,
        input                   reset_n,
        input   [3:0]   image_data1,
        input   [3:0]   image_data2,
        input   [3:0]   image_data3,
        input   [3:0]   image_data4,
        input   [7:0]   param1,
        input   [7:0]   param2,
        outputreg[7:0]  data_out
  );

        reg [15:0] mult1;
        reg [15:0] mult2;
        reg [15:0] mult3;
        reg [15:0] mult4;

        reg [3:0]       image_data1_reg;
        reg [3:0]       image_data2_reg;
        reg [3:0]       image_data3_reg;
        reg [3:0]       image_data4_reg;

        wire [13:0] data_out_temp;
        wire [7:0] factor1_2;
        wire [7:0] factor2_2;

        wire [11:0] mult21;
        wire [11:0] mult22;
        wire [11:0] mult23;
        wire [11:0] mult24;

        assign factor1_2 = 8'd255 - param1[7:0];
        assign factor2_2 = 8'd255 - param2[7:0];

        always@(posedge clock or negedge  reset_n)
        begin
                if(~reset_n) begin
                        mult1           <= 16'h0000;
                        mult1           <= 16'h0000;
                        mult1           <= 16'h0000;
                        mult1           <= 16'h0000;
```

```
                end
                else begin
                        mult1   <= factor1_2 * factor2_2;
                        mult2   <= param1[7:0] * factor2_2;
                        mult3   <= factor1_2 * param2[7:0];
                        mult4   <= param1[7:0] * param2[7:0];

                end
        end

        assign mult21 = image_data1_reg * mult1[15:8];
        assign mult22 = image_data2_reg * mult2[15:8];
        assign mult23 = image_data3_reg * mult3[15:8];
        assign mult24 = image_data4_reg * mult4[15:8];
        assigndata_out_temp = (mult21 + mult22) +       (mult23 + mult24);

        always@(posedge clock or negedge reset_n)
        begin
                if(~reset_n)begin
                        image_data1_reg         <= 4'b0;
                        image_data2_reg         <= 4'b0;
                        image_data3_reg         <= 4'b0;
                        image_data4_reg         <= 4'b0;
                end
                else begin
                        image_data1_reg         <= image_data1;
                        image_data2_reg         <= image_data2;
                        image_data3_reg         <= image_data3;
                        image_data4_reg         <= image_data4;
                end
        end

        always@(posedge clock or negedge reset_n)
        begin
                if(~reset_n)
                        data_out          <= 8'b0;
                else
                        data_out          <= data_out_temp[11:4];
        end

endmodule
```