**Technion**

*Electrical Engineering Department*

High Speed Digital System Lab

Image Manipulation Core for FPGA- Total

**Version**: 1.02

**Last update**: 10.11.2012

**Written By**: Ran Mizrahi , Uri Tsipin

**Guided by:** Moshe Porian

Table of Contents

[1 Introduction 5](#_Toc340326177)

[1.1 Abstract 5](#_Toc340326178)

[1.2 Applications 5](#_Toc340326179)

[1.2.1 Pilot Helmets 5](#_Toc340326180)

[1.2.2 Intelligence Surveillance Devices 5](#_Toc340326181)

[1.2.3 Surgical Applications 6](#_Toc340326182)

[1.2.4 Printer 6](#_Toc340326183)

[2 Goals 7](#_Toc340326184)

[3 Project requirements 8](#_Toc340326185)

[4 The algorithm 9](#_Toc340326186)

[4.1 Crop Input Image 9](#_Toc340326187)

[4.2 Image Rotation 9](#_Toc340326188)

[4.2.1 Calculation of source pixel address 10](#_Toc340326189)

[4.2.2 Bilinear Interpolation 11](#_Toc340326190)

[4.3 Zoom 12](#_Toc340326191)

[4.3.1 Mathematical background 12](#_Toc340326192)

[4.3.2 Implementation 12](#_Toc340326193)

[4.4 User defined output resolution 13](#_Toc340326194)

[5 General description 14](#_Toc340326195)

[5.1 Project's Top Blocks Scheme 14](#_Toc340326196)

[5.2 Block Descriptions 15](#_Toc340326197)

[5.2.1 Rx Path 15](#_Toc340326198)

[5.2.2 Tx Path 15](#_Toc340326199)

[5.2.3 Memory Management 16](#_Toc340326200)

[5.2.4 Display Controller 16](#_Toc340326201)

[5.2.5 Wishbone Interconnector 17](#_Toc340326202)

[5.2.6 Image Manipulation 17](#_Toc340326203)

[5.3 Data flow 17](#_Toc340326204)

[6 Implementation 19](#_Toc340326205)

[6.1 Image Manipulation – Top Architecture 19](#_Toc340326206)

[6.1.1 Parameters Registers 19](#_Toc340326207)

[6.2 Address calculator 20](#_Toc340326208)

[6.2.1 General Description 20](#_Toc340326209)

[6.2.2 Illustration 20](#_Toc340326210)

[6.2.3 The algorithm 21](#_Toc340326211)

[6.2.4 µArchitecture 22](#_Toc340326212)

[6.2.5 Inputs 22](#_Toc340326213)

[6.2.6 Outputs 23](#_Toc340326214)

[6.2.7 Simulations 24](#_Toc340326215)

[6.2.8 Testing 25](#_Toc340326216)

[6.2.9 Matlab Test Results 27](#_Toc340326217)

[6.3 Bilinear Interpolator 30](#_Toc340326218)

[6.3.1 General description 30](#_Toc340326219)

[6.3.2 The algorithm 30](#_Toc340326220)

[6.3.3 µArchitecture 31](#_Toc340326221)

[6.3.4 Inputs 31](#_Toc340326222)

[6.3.5 Outputs 32](#_Toc340326223)

[6.3.6 Simulation 32](#_Toc340326224)

[7 Working methods 33](#_Toc340326225)

[7.1 Top down design 33](#_Toc340326226)

[7.2 Pipeline 33](#_Toc340326227)

[7.3 Test bench 34](#_Toc340326228)

[7.4 Component documenting 34](#_Toc340326229)

[7.5 Result comparison with Matlab 34](#_Toc340326230)

[7.6 Working with SVN 34](#_Toc340326231)

[8 Summery 35](#_Toc340326232)

[8.1 Problems during the process 35](#_Toc340326233)

[8.1.1 Working with fractures 35](#_Toc340326234)

[8.1.2 Trigonometric calculations 35](#_Toc340326235)

[8.1.3 Timing issues 35](#_Toc340326236)

[8.2 Architectural changes 35](#_Toc340326237)

[8.3 Coding conclusions 36](#_Toc340326238)

[8.4 Project Conclusions 36](#_Toc340326239)

[9 Abbreviations 37](#_Toc340326240)

Table Of Figures

[Figure 1 5](#_Toc340326302)

[Figure 2 5](#_Toc340326303)

[Figure 3 6](#_Toc340326304)

[Figure 4 6](#_Toc340326305)

[Figure 5 – Example Of Image Cropping 9](#_Toc340326306)

[Figure 6 - Example of 30 degree rotation 9](#_Toc340326307)

[Figure 7 - A counterclockwise rotation of a vector through angle θ 10](#_Toc340326308)

[Figure 8- bilinear illustration 11](#_Toc340326309)

[Figure 9- sine function 12](#_Toc340326310)

[Figure 10- scaling function (1) 12](#_Toc340326311)

[Figure 11- scaling function (2) 12](#_Toc340326312)

[Figure 12- project main diagram 14](#_Toc340326313)

[Figure 13- RX path block 15](#_Toc340326314)

[Figure 14- TX path block 15](#_Toc340326315)

[Figure 15- memory management block 16](#_Toc340326316)

[Figure 16- Display Controller block 16](#_Toc340326317)

[Figure 17- Image Manipulation block 17](#_Toc340326318)

[Figure 18- Image Manipulation- Top Arch 19](#_Toc340326319)

[Figure 19- Addr\_Calc illustration 20](#_Toc340326320)

[Figure 20- Output size vector 21](#_Toc340326321)

[Figure 21- Micro Arch. – Addr\_Calc 22](#_Toc340326322)

[Figure 22- Addr\_Calc simulation (1): first valid data 24](#_Toc340326323)

[Figure 23- Addr\_Calc simulation (2)- midway output 24](#_Toc340326324)

[Figure 24- Addr\_Calc simulation (3)- final procedure 25](#_Toc340326325)

[Figure 25- Addr\_Calc Testing Process 26](#_Toc340326326)

[Figure 26- View the results of Addr\_calc with Matlab 26](#_Toc340326327)

[Figure 27 - Simulation result 1 27](#_Toc340326328)

[Figure 28 - Simulation result 2 27](#_Toc340326329)

[Figure 29 - Simulation result 3 28](#_Toc340326330)

[Figure 30 - Simulation result 4 28](#_Toc340326331)

[Figure 31 - Simulation result 5 29](#_Toc340326332)

[Figure 32- Bilinear Interpolator illustration 30](#_Toc340326333)

[Figure 33- Bilinear algorithm scheme 30](#_Toc340326334)

[Figure 34- Micro Arch. Bilinear Interpolator 31](#_Toc340326335)

[Figure 35- Bilinear simulation 32](#_Toc340326336)

[Figure 36- Bilinear results- compare with Matlab 33](#_Toc340326337)

[Figure 37- Synthesis timing report (1) 35](#_Toc340326338)

[Figure 38- Synthesis timing report (2) 35](#_Toc340326339)

[Figure 39- Arch modification 35](#_Toc340326340)

# Introduction

## Abstract

Many modern digital devices excute different image manipulations. These manipulations requires image rotation and zooming.

Image Processing algorithms are "heavy consumers" of resources and therefor we would to boost the procces using hardware acceleration.

## Applications

### Pilot Helmets

Modern Day Pilot Helmets contains a digital display module. Due to the helmets geometry, being elliptic and not straight, there's a need to "deform" the displayed image.  
Part of the "deformation" algorithm requires to rotate the image and zooming in/out of the image.  


Figure 1

### Intelligence Surveillance Devices

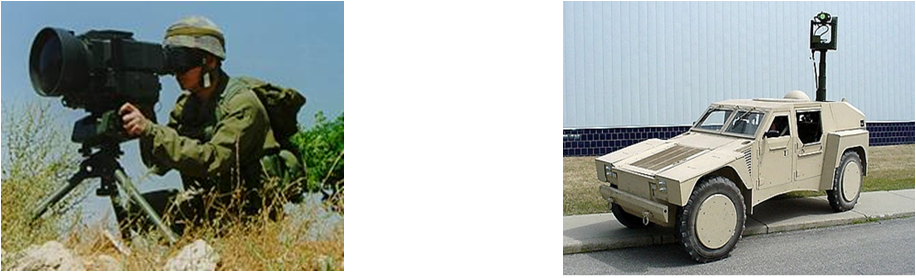
Security forces use different surveillance devices, and sometimes due to terrain conditions there is a need to rotate the image in order to make make the surveillance more comfortable.  


Figure 2

### Surgical Applications

Tiny cameras and Optical fibers are commonly used in various medical procedures. In order to improve the image and ease the procedure image rotation and zooming is necessary.

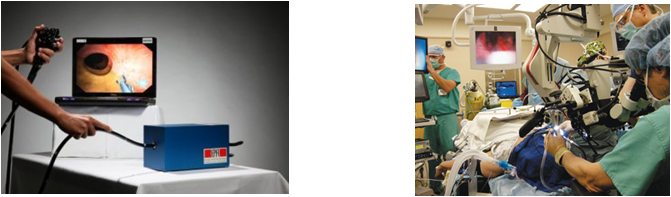


Figure 3

### Printer

New age printers have a built-in LCD display, using our feature the user will have the ability to edit (crop, zoom, rotate) the image before it is sent out to print simply and fast  


Figure 4

# Goals

Implement a FPGA core using VHDL. The core will execute the following objectives

* Full panoramic rotation: 0 to 360 degrees.
* Support of Zoom function.
* Support of crop image by user defined coordinates.

Building a GUI (Graphical User Interface) using Matlab. The GUI will transfer data packets (including the image, required zoom/rotation parameters/crop coordinates) to the FPGA.  
Finally, the GUI will display the output image for Debug purposes.

# Project requirements

* Input Image resolution must be 600x800 and monochromatic.
* Output image resolution will be 480x640.
* Zoom factor must be an integer, power of 2 (i.e. 4,16,32…)
* The rotation angle must be an integer.
* System clock- 100 MHz.
* Vesa (display) clock- 40 MHz.

# The algorithm

In the digital processing world images are represented by many different methods, in this project we will use the simplest way, a bitmap grey scale image.

The image is represented by a 2-dimensinal matrix, where every cell holds the grey scale of the image. Images are 8bit which corresponds with 256 grey levels.

The Algorithm executes three operations: crop, zoom and rotate.

## Crop Input Image

The user inputs (x\_start, y\_start) coordinates which describe the top left corner of the required cropped image, where (1,1) leaves the original picture intact.

The algorithm defines a new image which copies the original image matrix starting from  
(x\_start, y\_start) coordinates until the end of the matrix size.  
In case of colored picture, the matrix is three dimensional, the third dimension will be copied according to the same coordinates.

Figure 5 – Example Of Image Cropping

## Image Rotation

The rotation algorithm includes three main stages

* define a black picture with the required input dimensions
* Scanning the output image, pixel by pixel, and calculating the source coordinate of the pixel in original image.

Figure 6 - Example of 30 degree rotation

* Evaluating the grey/color level of the pixel using bi-linear interpolation.

### Calculation of source pixel address

Assume the pixel coordinate in the orginal image matrix is   
For a rotation over an angle ,  in the original image is mapped onto the point in the output image.

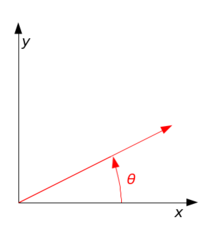


Figure 7 - A counterclockwise rotation of a vector through angle θ

The relation between the points is:



During the algorithm we scan the output image and we evaluate the grey level of the original non-rotated image and therefore we need to use the inverse transform:



Since a matrix cell address is a positive integer we have a problem calculating the source address using this method because cosine and sine functions give real values. Hence we will round up and down the values and evaluate the grey level using bilinear interpolation.

### Bilinear Interpolation

Once we round up the address values a problem rises, we cannot restore the original pixel address in the source image, and therefore we evaluate the grey/color level using bilinear interpolation.  
Bilinear Interpolation performs a weighted average between the four sourrounding neighbors of the required "real" address according to the following method:

* R1, R2 are the weighted averages between the top pair and the bottom pair , accordingly.
* Output is the weighted average between R1 and R2

Figure 8- bilinear illustration

The following formula describes the bilinear interpolation (weighted average):



## Zoom

### Mathematical background

The zoom function is achieved by using the mathematical operation of scaling the axis. For example- in one dimension: 

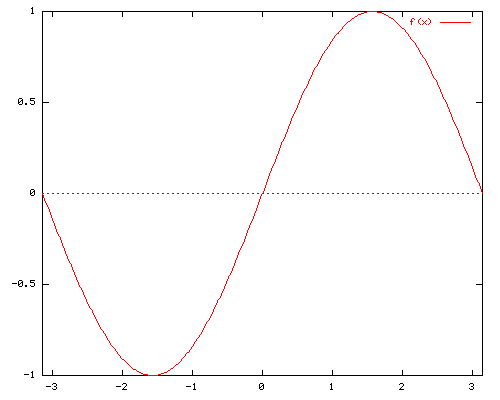


Figure 9- sine function

By scaling the X axis with the factor 'a', we receive the follow result:



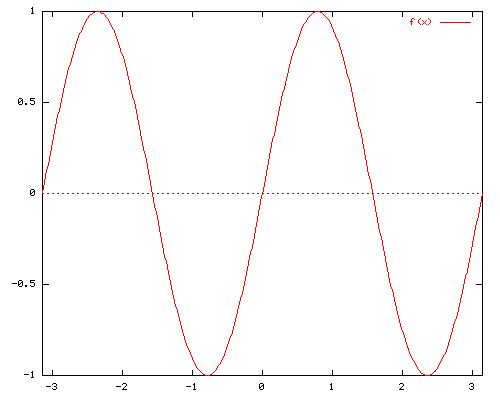


Figure 10- scaling function (1)



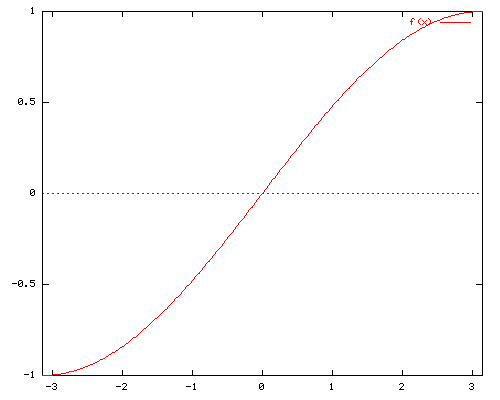


Figure 11- scaling function (2)

We induct the same principle in two dimensions- (x,y) axis.

### Implementation

In order to zoom in during the scan of the source image, the algorithm multiplies the number of appearances of a source pixel in the output image, according to the zoom factor.

In order to zoom out, in the output image we "skip" every specific amount of pixels in the source image- according the zoom factor. The following formula describes both actions:



## User defined output resolution

The algorithm assumes that the output resolution is smaller or equal to the source resolution. In case the output resolution is smaller, the algorithm shrinks the source image in order to fit it in the new frame. The implementation is identical to the zoom function, only with different factors. The following formula describes the action:



and the same for Y axis.

# General description

## Project's Top Blocks Scheme

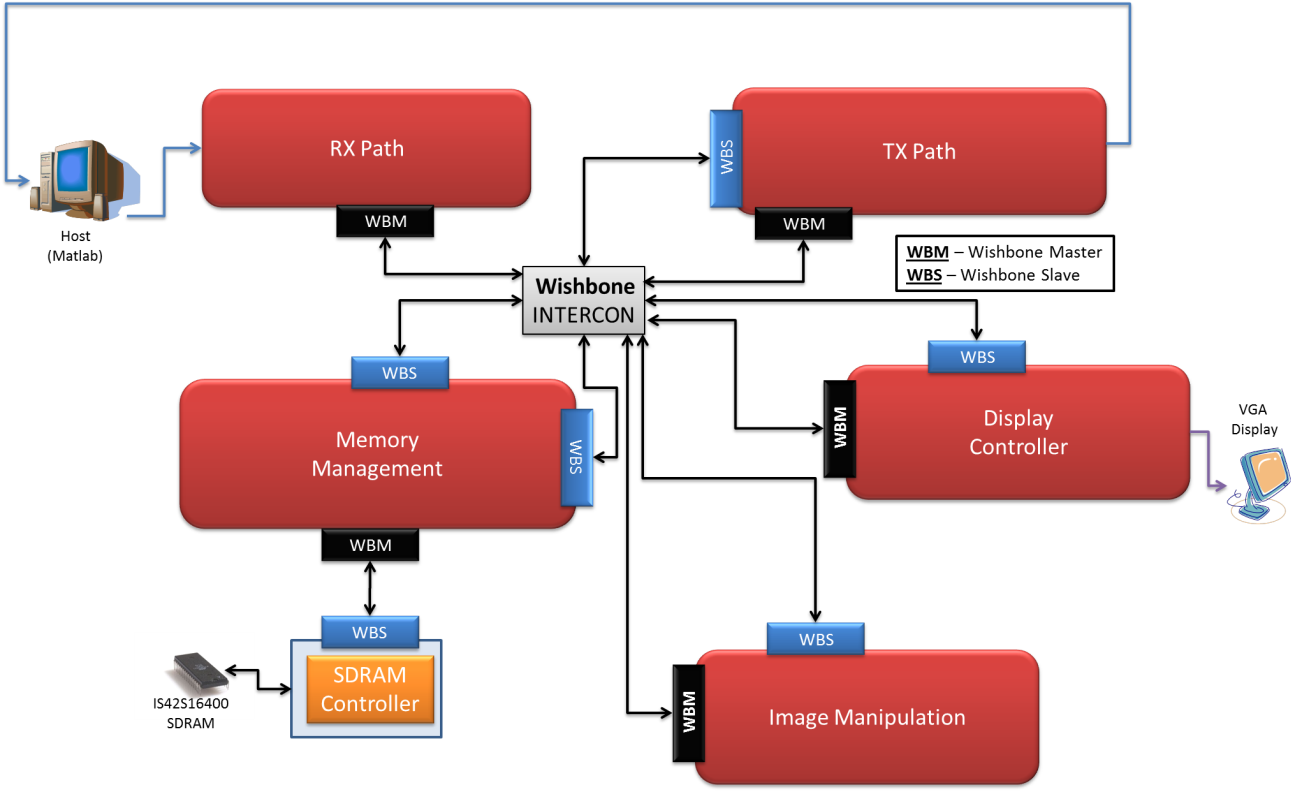


Figure - project main diagram

The following blocks were reused from a previous project:

Rx Path, Tx Path, Memory Management, Wishbone Intercon, Display Controller.  
The blocks were adapted to fit our project's needs.

The previous project goal was to display a compressed image (using Run-Length coding), where our project is using a bitmap image so the following changes were made:

* Removing the Run-Length extractor
* Adapting of the Host – creating a GUI that will support our needs
* Creating a new block – Image Manipulation

## Block Descriptions

### Rx Path

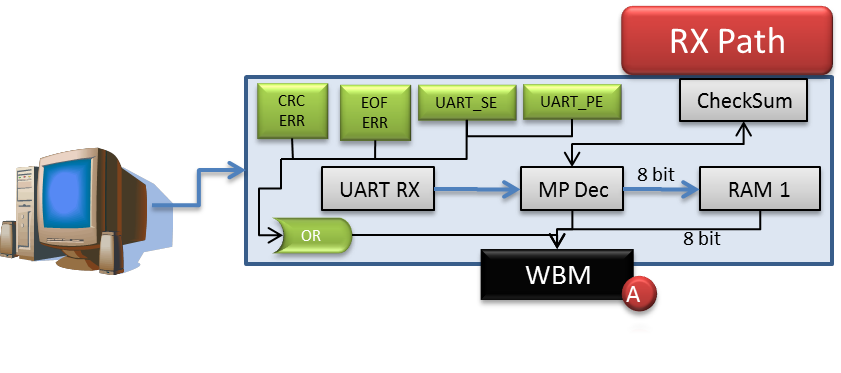


Figure - RX path block

This component is used for asynchronous serial data channel.

The receiver converts serial start bit, data, parity and stop bit to parallel data.

The data word length can be 5-8 bits, according to generic parameter. Parity bit can be odd or even or if decided can be inhibited, according to generic parameters.

All inputs and outputs are synchronized with the positive edge of the clock.

Any system clock and any baud rate are supported, according to generic parameter.

### Tx Path

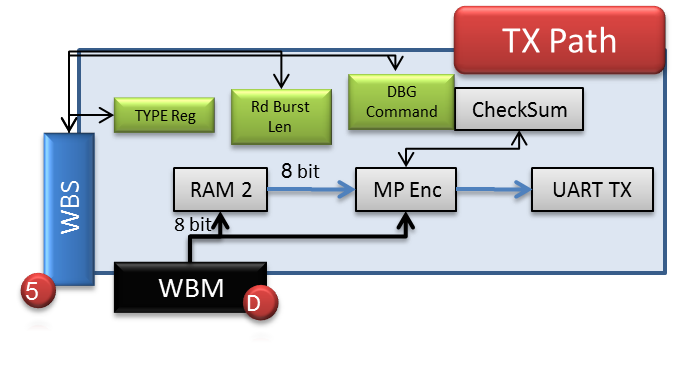


Figure - TX path block

The transmitter converts parallel data into serial form and automatically adds start bit, parity and stop bit.

The main use of the component is for debug.

### Memory Management

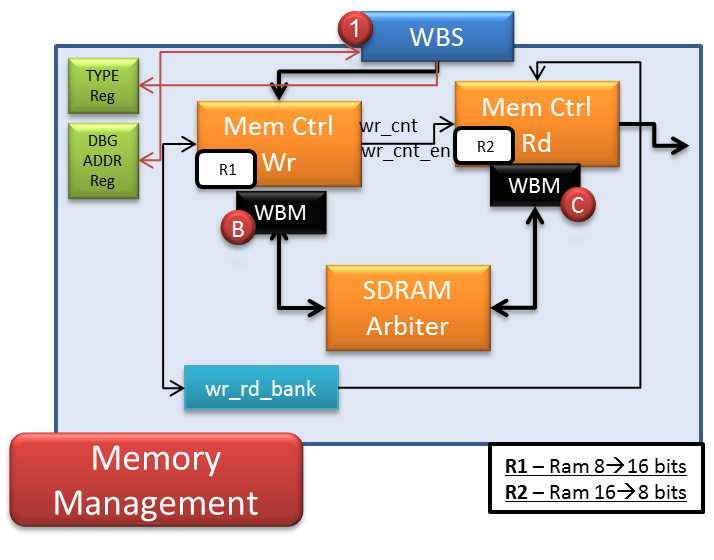


Figure - memory management block

The Memory Management block is in-charge of writing and reading data from the SDRAM.

### Display Controller

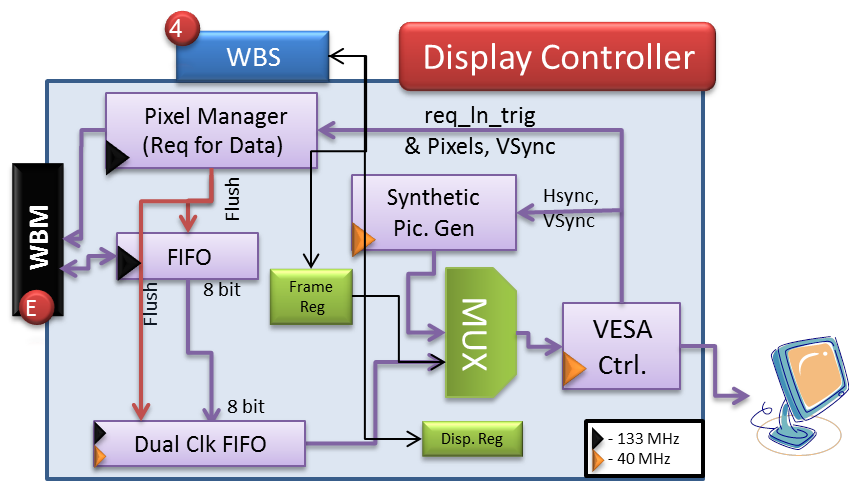


Figure - Display Controller block

The Display Controller block transmits the data from the SDRAM / Synthetic Picture Generator, through the DAC on the DE2, to the VGA.

The FIFO's depth is 256X2X3 = 1536 (X8 bits), for 3 SDRAM read burst.

The Dual Clock FIFO's depth is 640X6 = 3,840 (X8 bits), for 6 lines.

### Wishbone Interconnector

The system has 3 wishbone interconnectors

* **Intercon Z** - Two Wishbone Masters are connected to this interconnection:

1. WBM (A) from RX Path, which responsible to write data (SDRAM / Registers) to Wishbone Slaves.
2. WBM (D) from TX Path, which responsible to read data (SDRAM / Registers) from Wishbone Slaves.

* **Intercon Y -** This interconnection receives the TYPE register, which is transmitted from the WBM (A). In case of debug mode, WBM (D) will be connected to the path. In case of normal mode, WBM (E) will be connected to the path.
* **Intercon X -** According to WBM (D) command (TGC\_I), routes the path toward reading from SDRAM in the Memory Block or from the registers, through INTERCON (Z)

### Image Manipulation

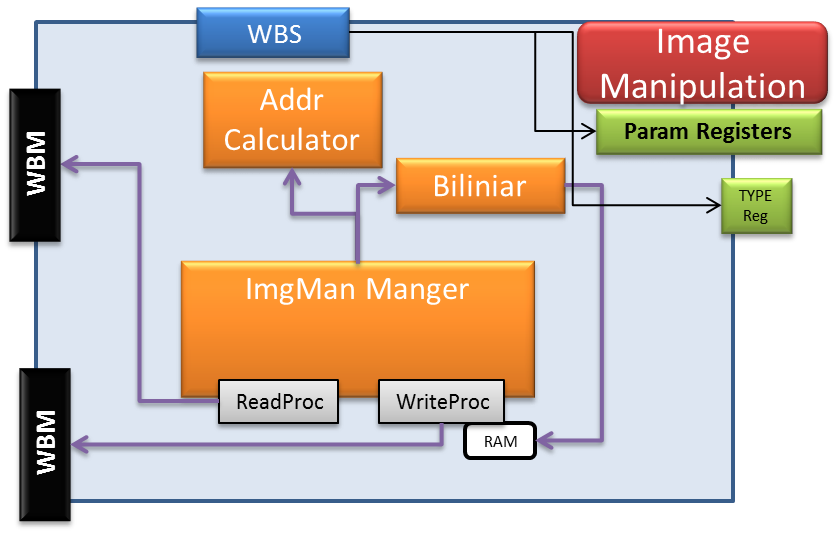


Figure - Image Manipulation block

Image Manipulation block is in-charge of rotating, zooming and cropping the image. The blocks receives the data through the Wishbone Intercon, performs the required manipulation and writes it back to the SDRAM via the Wishbone Intercon.

The main components of the block are

* ImgMan Manager – controller for the image manipulation block
* Address Calculator – Calculates "matrix address" of 4 pixels that are required for the bilinear-interpolation and convert the "matrix address" into a "1D" SDRAM address.
* Bilinear Interpolator – Calculates a mean average between 4 pixels
* Parameters Register – holds the user parameters: angle, ROI indexes (Xstart, Ystart), Zoom.

## Data flow

**Host** transmits wrapped message of data packets, composed of bitmap image and user parameters to the **RX Path**. Message is decoded, transmitted to the **Memory Management** block, and stored into **Storage Device** (parameters into registers, image to the SDRAM). Data is read from the memory, sent to image manipulation block, and rewritten to the SDRAM. Then the manipulated image is transmitted to the **Display** through the **Display Controller**.

# Implementation

This section contains general information about the coded components in this project, the image manipulation block.

As mentioned before, this block is part of a system which is responsible for writing data to an external memory device and displaying images on a screen via vga protocol.

## Image Manipulation – Top Architecture

### General Description

The image manipulation block is in-charge doing image manipulation calculations, hold user parameters (for manipulation) and send read/write requests to memory block.

### µArchitecture

The block holds instances of addr\_calc, bilinear, img\_man\_manager and of all the registers required (detailed ahead).

The block has two WBM interfaces:

1. rd\_WBM: connected to memory management block via interconnector Y.
2. wr\_WBM: connected Z to memory management block via interconnector Z.

The block has one WBS interface for writing user parameters (zoom, sin\_teta, etc.) in local registers.

### Inputs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **description** | **type** | **Size(bit)** | **Received from** |
| system\_clk | System clock | Std\_logic | 1 |  |
| system\_rst | System reset | Std\_logic | 1 |  |
| image\_tx\_en | Image manipulation complete - Enable display | Std\_logic | 1 |  |
| manipulation\_trig | trigger block | Std\_logic | 1 |  |
|  |  |  |  |  |
| wbs\_adr\_i |  |  |  |  |
| wbs\_tga\_i |  |  |  |  |
| wbs\_dat\_i |  |  |  |  |
| wbs\_cyc\_i |  | Std\_logic | 1 |  |
| wbs\_stb\_i |  | Std\_logic | 1 |  |
| wbs\_we\_i |  | Std\_logic | 1 |  |
| wbs\_tgc\_i |  | Std\_logic | 1 |  |
|  |  |  |  |  |
| wr\_wbm\_dat\_i |  |  |  |  |
| wr\_wbm\_stall\_i |  | Std\_logic | 1 |  |
| wr\_wbm\_ack\_i |  | Std\_logic | 1 |  |
| wr\_wbm\_err\_i |  | Std\_logic | 1 |  |
|  |  |  |  |  |
| rd\_wbm\_dat\_i |  |  |  |  |
| rd\_wbm\_stall\_i |  | Std\_logic | 1 |  |
| rd\_wbm\_ack\_i |  | Std\_logic | 1 |  |
| rd\_wbm\_err\_i |  | Std\_logic | 1 |  |

### Outputs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **description** | **type** | **Size(bit)** | **Received from** |
| wbs\_dat\_o |  |  |  |  |
| wbs\_stall\_o |  | Std\_logic | 1 |  |
| wbs\_ack\_o |  | Std\_logic | 1 |  |
| wbs\_err\_o |  | Std\_logic | 1 |  |
|  |  |  |  |  |
| wr\_wbm\_adr\_o |  |  |  |  |
| wr\_wbm\_tga\_o |  |  |  |  |
| wr\_wbm\_dat\_o |  |  |  |  |
| wr\_wbm\_cyc\_o |  | Std\_logic | 1 |  |
| wr\_wbm\_stb\_o |  | Std\_logic | 1 |  |
| wr\_wbm\_we\_o |  | Std\_logic | 1 |  |
| wr\_wbm\_tgc\_o |  | Std\_logic | 1 |  |
|  |  |  |  |  |
| rd\_wbm\_adr\_o |  |  |  |  |
| rd\_wbm\_tga\_o |  |  |  |  |
| rd\_wbm\_cyc\_o |  | Std\_logic | 1 |  |
| rd\_wbm\_tgc\_o |  | Std\_logic | 1 |  |
| rd\_wbm\_stb\_o |  | Std\_logic | 1 |  |

### Simulation

### Parameters Registers

Parameter Registers are located within img\_man\_top.vhd. The registers hold user parameters (such as zoom, crop coordinates etc.) that are required for coordinate calculations in order to execute the image manipulation.

The table below describes all the properties of the registers

|  |  |  |  |
| --- | --- | --- | --- |
| Register's name | Address (Hex) | Size (bytes) | Purpose |
| Type\_reg | 10 | 1 | Type of message |
| x\_start\_reg | 11 | 2 | X crop coordinate |
| y\_start\_reg | 13 | 2 | Y crop coordinate |
| zoom\_reg | 15 | 2 | Zoom ratio |
| cos\_reg | 17 | 2 | Cosine of rotation angle, multiplied by 0x100 |
| Sin\_reg | 19 | 2 | Sine of rotation angle, multiplied by 0x100 |

**Write to Registers**

***Note***: *Type\_reg should not be written by the software. It is done automatically by the design at each UART message transmission (message = SOF, TYPE … EOF). Type\_reg* *value will be set to the TYPE of the UART message*.

In order to write to register, the following UART message should be sent:

* SOF (=0x64)
* TYPE, where the MSB is '1'. (i.e: 0x80 = write to registers)
* ADDRESS, which is the register's address. (i.e: 2 for dbg\_reg)
* LENGTH, which is the burst length minus 1. (i.e: 2 for dbg\_reg, which is 3 bytes wide)
* PAYLOAD, which is the register's value to be written. (i.e: for dbg\_reg, PAYLOAD might be 0x[01 FA 00] )
* CRC, which is the checksum for TYPE🡪PAYLOAD (inclusive)
* EOF (=0xC8)

This will generate Wishbone Write transaction to the relevant register.

**Register's structure**

The registers are wrapped by *wbs\_reg* component, which translates Wishbone transaction to write / read from a specific register, according to the table above.

## Address calculator

### General Description

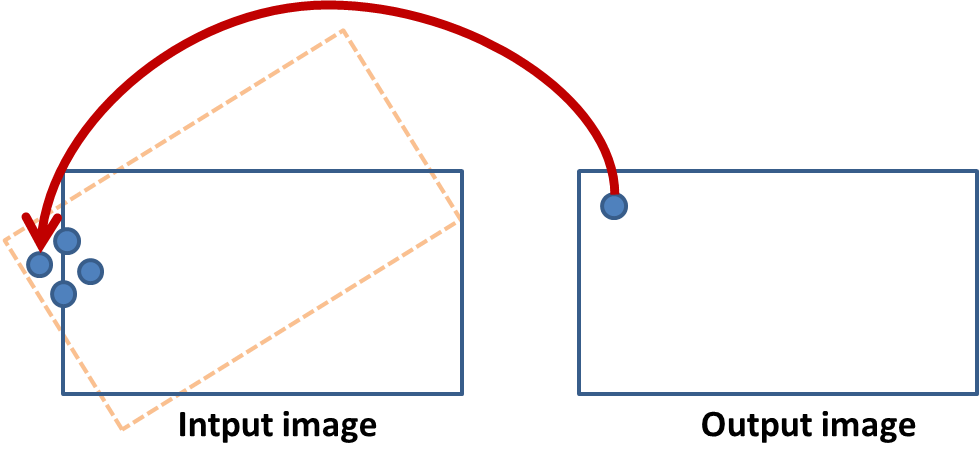
The unit calculates the source pixels address in matrix form a given current position (address, matrix form) in the output image.

The output is addresses of 4 pixels that are required for the bilinear interpolation.

The input is address of 1 pixel of current position in output image.

**Note: X represents row indexes, Y represents column indexes.**

### Illustration

****

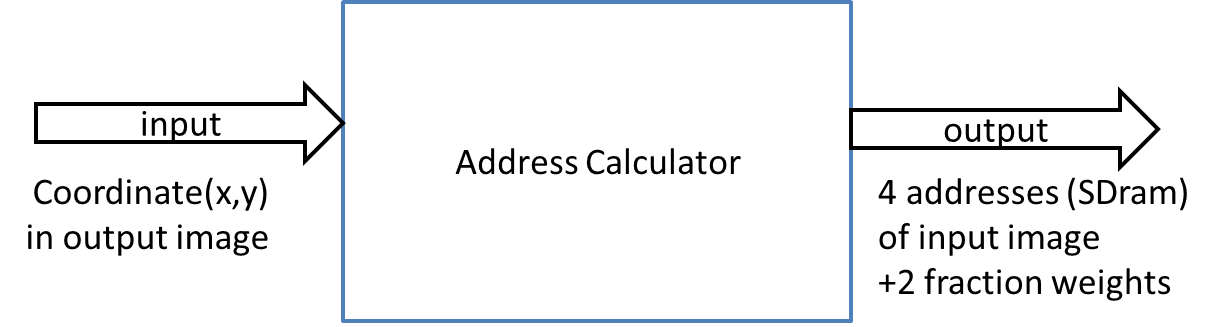


Figure - Addr\_Calc illustration

### The algorithm

The algorithm calculates the desired coordinates from the origin image, from given [x,y] coordinates. The mathematical equation the algorithm uses is:



For horizontal calculation the algorithm uses the same equation after replacing the relevant parameters.

In order to reduce calculation time and improve throughput, the algorithm disassembles the equation into 5 parts (Pipeline).

Every multiply operation that should be made, 1 bit must be added to prevent overflow effects (in order to work with std\_logic\_vectors or signed). In addition, the new size must be the size of the two arguments together. Finally, 1 bit of signed type is added (mid calculations might include negative results). Thus, the sizes of output ports were determined.

The following image may explain the final output form:

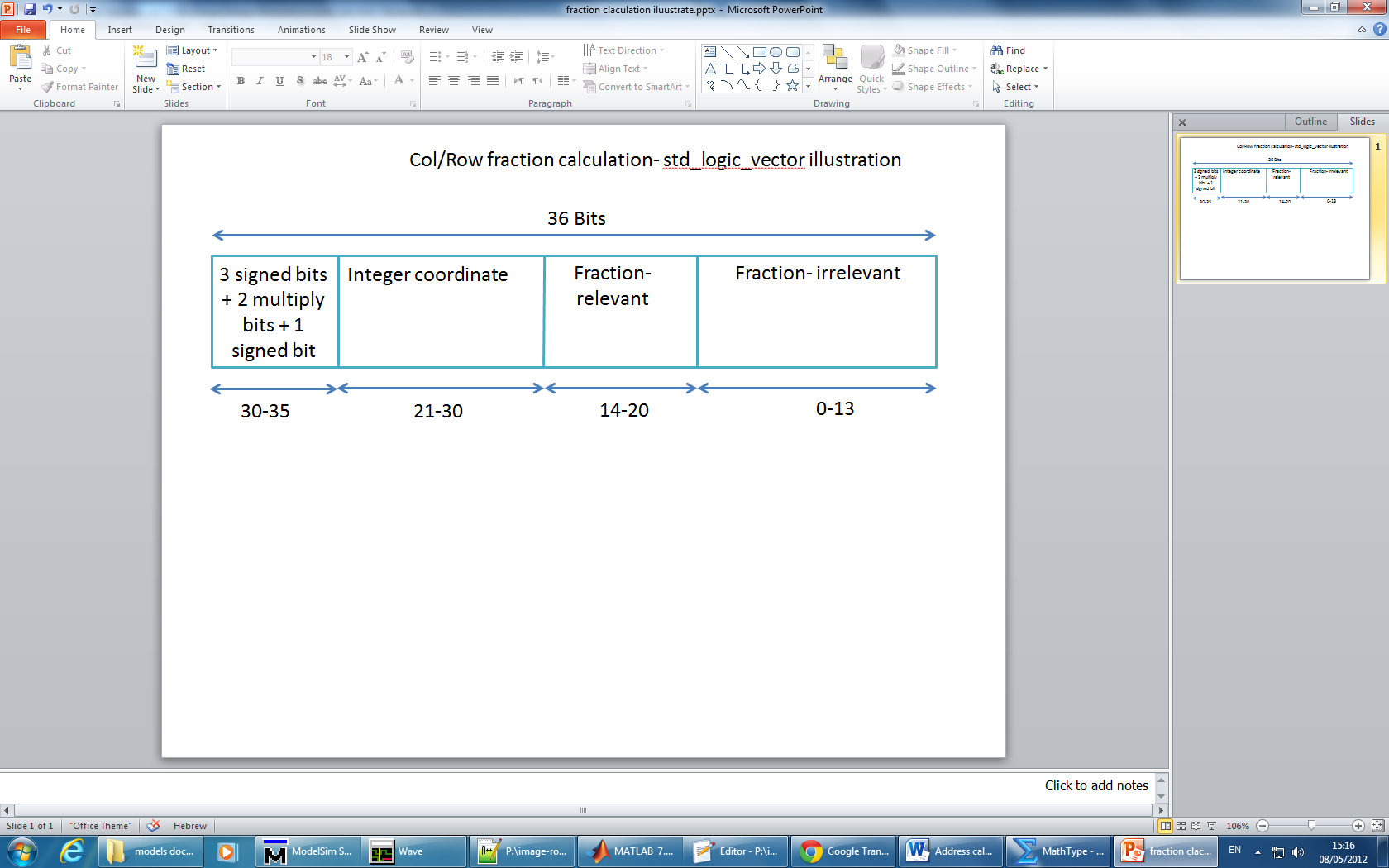


Figure - Output size vector

This std\_logic\_vector represent the desired output index in matrix form.

In order to convert this index to SDRAM form, the algorithm uses another simple equation and creates the output index port (23 bits).

### µArchitecture

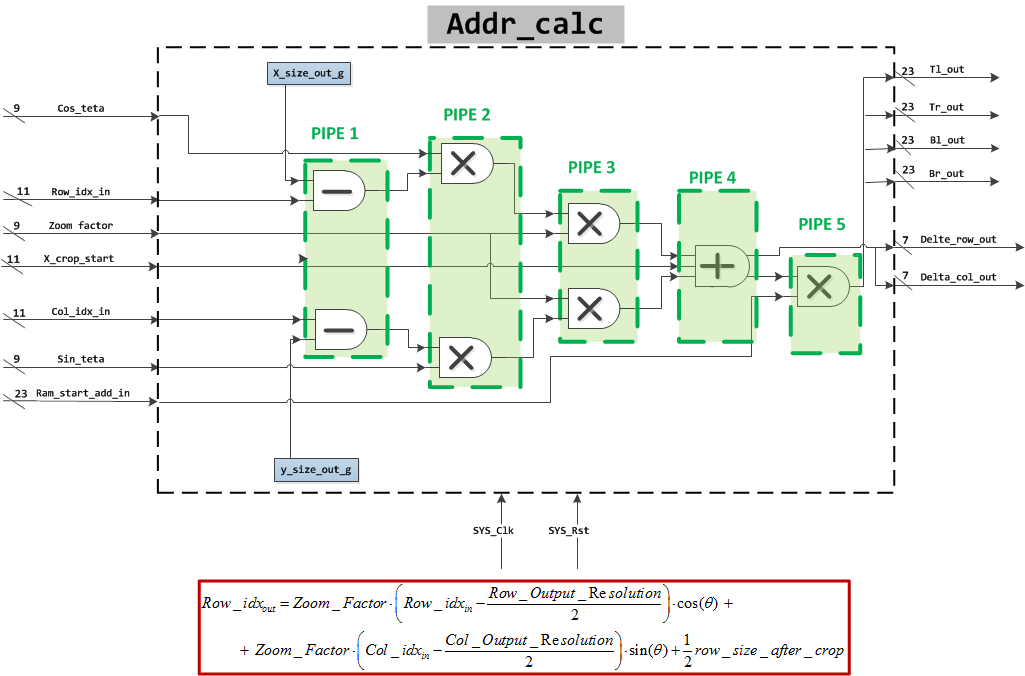


Figure - Micro Arch. – Addr\_Calc

### Inputs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **description** | **type** | **Size(bit)** | **Received from** |
| System\_clk | System clock | Std\_logic | 1 | Global system |
| System\_rst | System reset | Std\_logic | 1 | Global system |
| enable | Enable unit | Std\_logic | 1 | Img\_manager |
| Trigger unit | enable signal for addr\_calc | Std\_logic | 1 | Img\_manager |
| Zoom\_factor | Holds the zoom factor | signed | 9 | Param\_reg(Img\_man\_top) |
| Sin\_teta | Holds sin(teta) | signed | 9 | Param\_reg(Img\_man\_top) |
| Cos\_teta | Holds cos(teta) | signed | 9 | Param\_reg(Img\_man\_top) |
| Row\_idx\_in | Holds the current row index of the output image | signed | 11 | Param\_reg(Img\_man\_top) |
| col\_idx\_in | Holds the current column index of the output image | signed | 11 | Param\_reg(Img\_man\_top) |
| X\_crop\_start | Holds the row index of the top left pixel for crop | signed | 11 | Param\_reg(Img\_man\_top) |
| Y\_crop\_start | Holds the column index of the top left pixel for crop | signed | 11 | Param\_reg(Img\_man\_top) |
| X\_size\_in | Holds the number of rows in the input image | generic | 10 | Img\_man\_top (Mds\_top\_block) |
| Y\_size\_in | Holds the number of columns in the input image | generic | 10 | Img\_man\_top (Mds\_top\_block) |
| X\_size\_out | Holds the number of rows in the output image | generic | 10 | Img\_man\_top (Mds\_top\_block) |
| Y\_size\_out | Holds the number of columns in the output image | generic | 10 | Img\_man\_top (Mds\_top\_block) |

### Outputs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **description** | **type** | **size** | **Destination** |
| TL\_out | Holds the top left row index in input image.  Index in SDRAM mode | Std\_logic\_vector | 23 | Img\_man\_manager |
| TR\_out | Holds the top right row index in input image. Index in SDRAM mode | Std\_logic\_vector | 23 | Img\_man\_manager |
| BL\_out | Holds the bottom left row index in input image. Index in SDRAM mode | Std\_logic\_vector | 23 | Img\_man\_manager |
| BR\_out | Holds the bottom right row index in input image. Index in SDRAM mode | Std\_logic\_vector | 23 | Img\_man\_manager |
| delta\_row\_out | Holds the delta of row, for bilinear-interpolation | Std\_logic\_vector | 7 | Bilinear interpolation |
| delta\_col\_out | Holds the delta of col, for bilinear-interpolation | Std\_logic\_vector | 7 | Bilinear interpolation |
| out\_of\_range | Signal indicating when pixel is out of range,  '0' in range '1' out of range | Std\_logic | 1 | Img\_man\_manager |
| Data\_valid\_out | Indicates ready data in outputs | Std\_logic | 1 | Img\_man\_manager |
| unit\_finish | Indicating when block is finish working on current pixel | Std\_logic | 1 | Img\_man\_manager |

Outputs coordinates are 23 bits because 22 bits is the SDRAM size + 1 bit for multiply calculations need.

### Simulations

**first valid output**

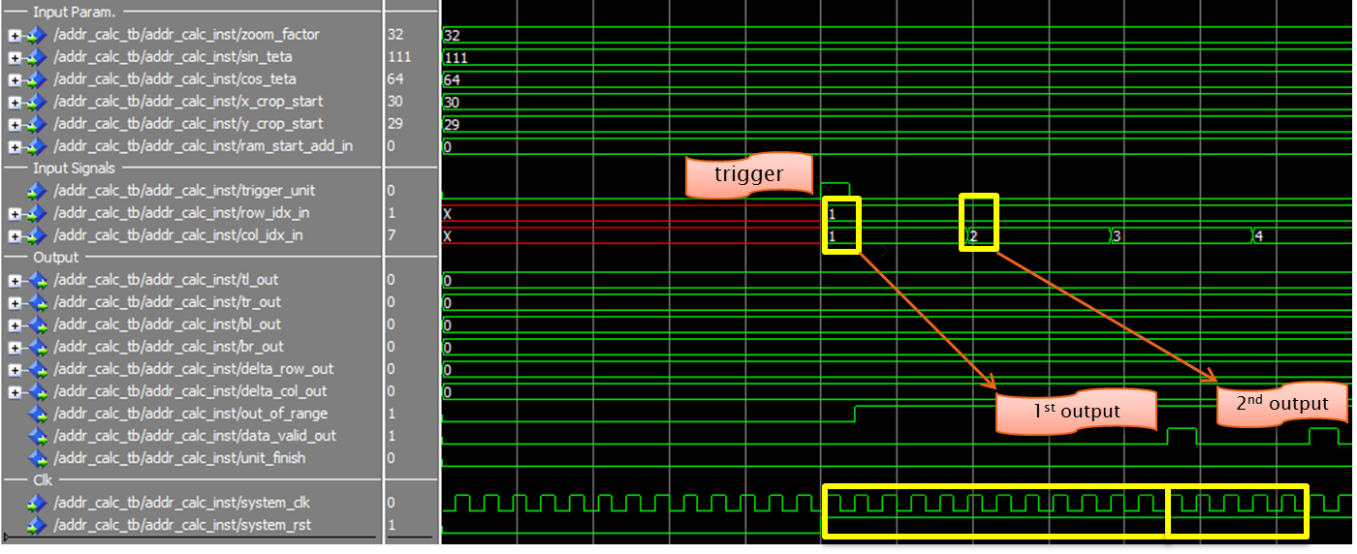


Figure - Addr\_Calc simulation (1): first valid data

The unit has a 12 cycle latency (number of cycles before the 1st valid result). As shown above, 5 cycles is the throughput of the system (i.e. every 5 cycles a new result is valid).

These values are controlled via generic ports:

* pipe\_depth\_g – number of latency cycles.
* valid\_setup\_g – number of throughput cycles.

It should be noted that these values were determined after trial and error (simulations) and they should not be changed.

**Midway valid output**

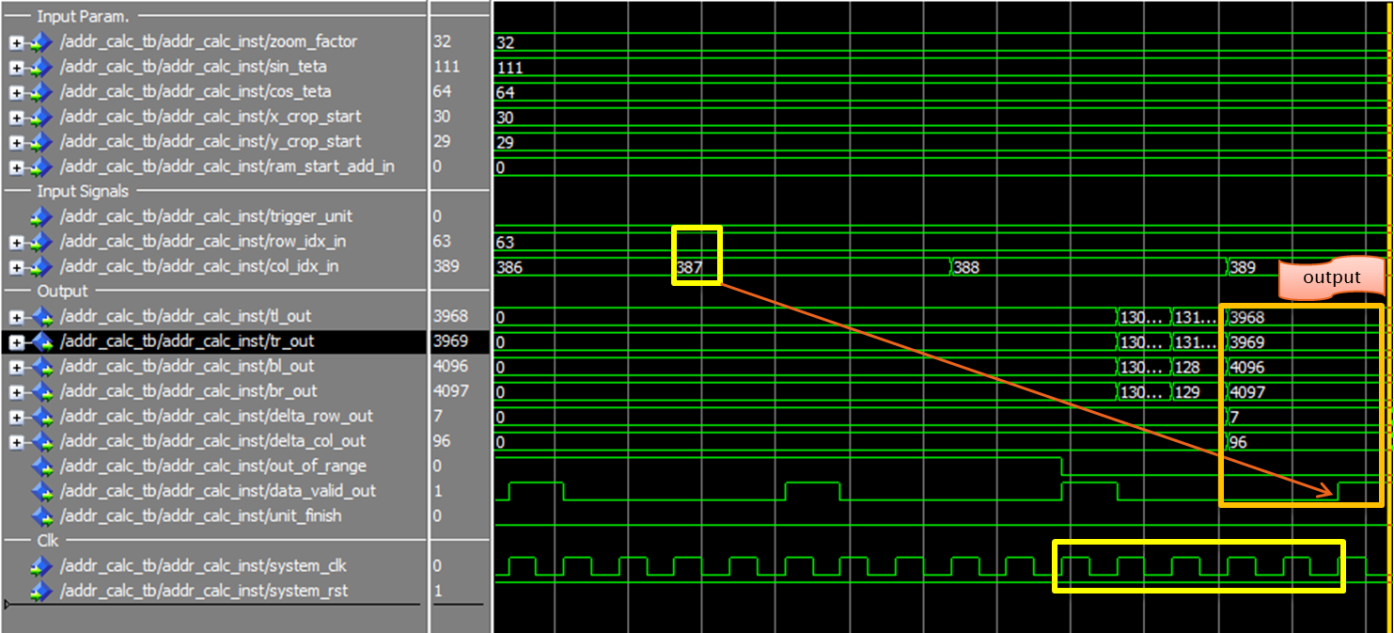
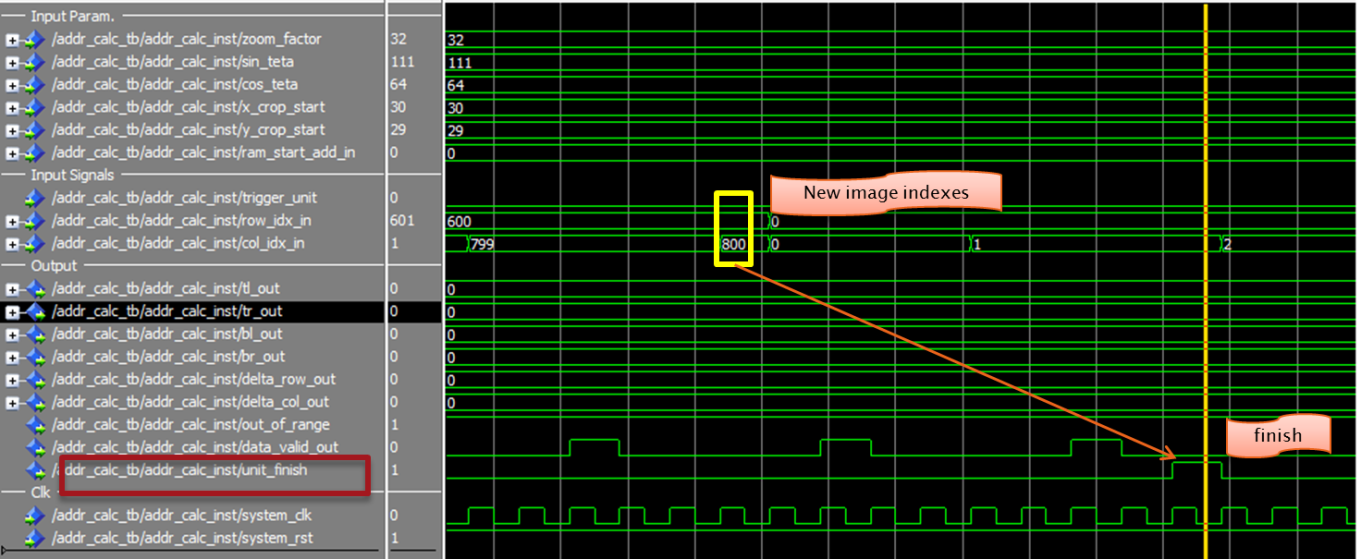


Figure - Addr\_Calc simulation (2)- midway output

5 cycles between every adjacent result is maintained.

**End of calculation process output**

****

finish

Figure - Addr\_Calc simulation (3)- final procedure

A process called *unit\_finish\_proc* is responsible for counting the number of pixels calculated so far. When last pixel is reached a flag named *unit\_finish* is raised.

After the completion a new image may begin its processing.

### Testing

In order to validate the accuracy of the coordinate calculations, a test bench was created. The test bench is written in VHDL, it connects to the *addr\_calc* block and injects test inputs: row and column indexes and user parameters.  
The row and column are generated by a nested loop that emulates the (matrix) coordinates of a 600x800 image.

The output of the *addr\_calc* is written into a text file ("*test\_modelsim.txt*") by the following format:

**col row tl tr bl br d\_row d\_col oor**

310 189 0 0 0 0 0 0 1

311 189 0 0 0 0 0 0 1

312 189 0 0 0 0 0 0 1

313 189 0 0 0 0 0 0 1

314 189 0 0 0 0 120 116 0

315 189 3932 3933 4060 4061 25 56 0

316 189 3932 3933 4060 4061 53 72 0

317 189 3932 3933 4060 4061 80 88 0

318 189 3932 3933 4060 4061 108 104 0

319 189 4061 4062 4189 4190 8 120 0

The next diagram describes the testing process:



Figure - Addr\_Calc Testing Process

The following stage was to inject the text file into a Matlab script ("*test\_addr\_calc\_output.m*") which simulates the full system. The system is simulated by implementing these main features:

* The image resides within the SDRAM ,modeled by 1d array
* Bilinear interpolation is done by Matlab instead of hardware
* The result image is displayed on the screen by Matlab

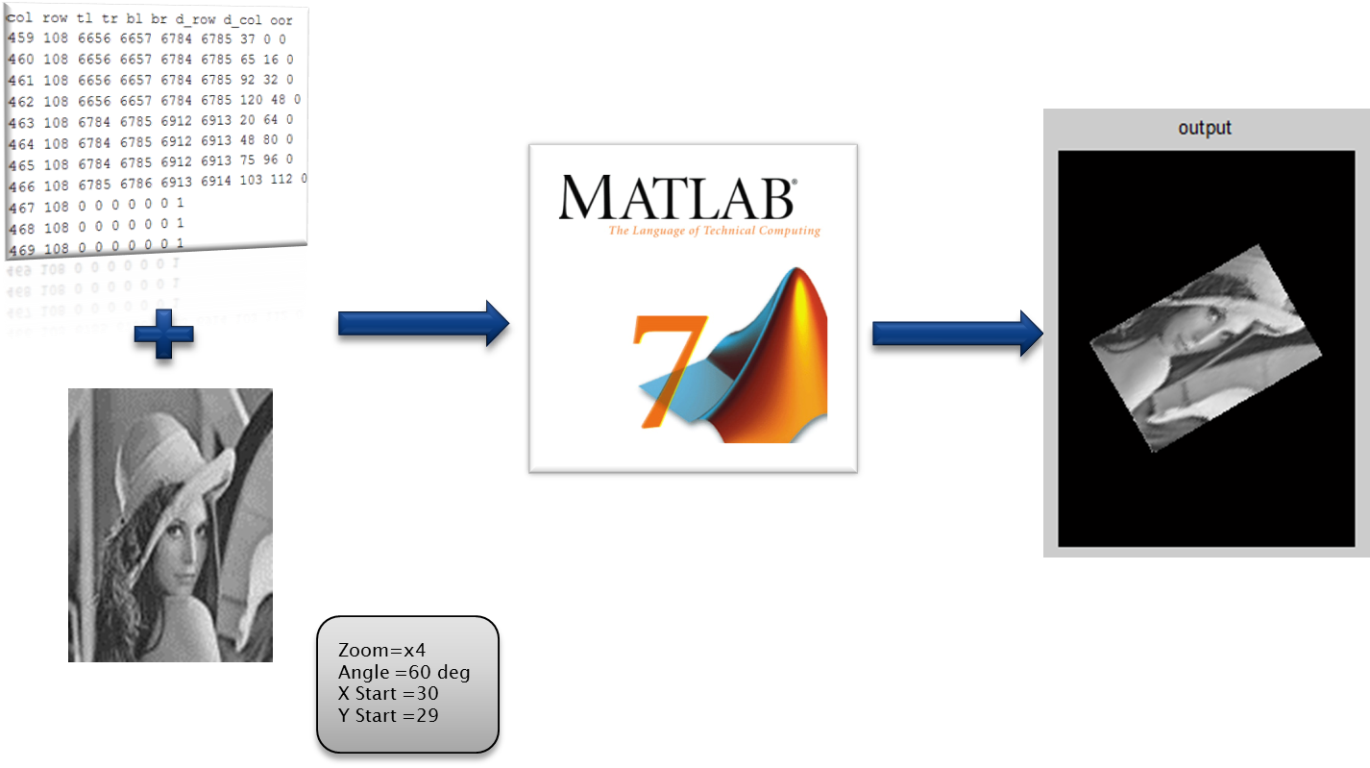


Figure - View the results of Addr\_calc with Matlab

### Matlab Test Results

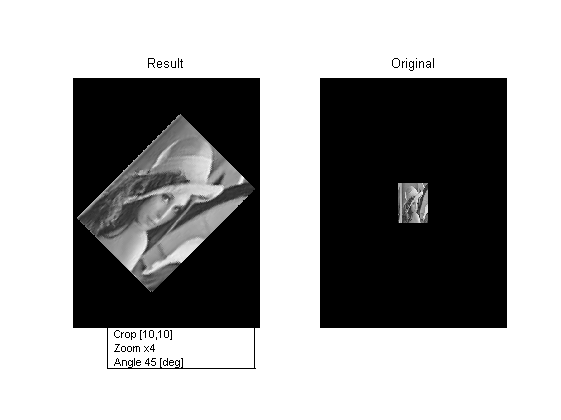


Figure - Simulation result 1

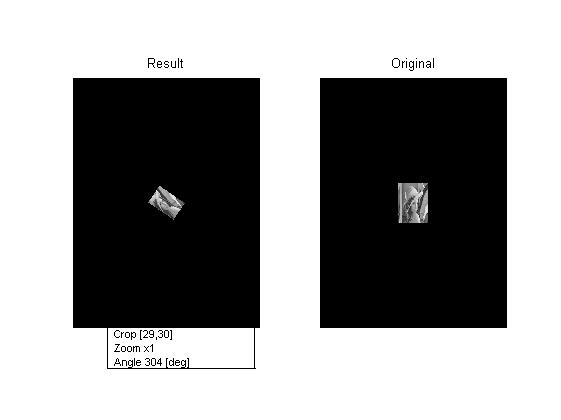


Figure - Simulation result 2

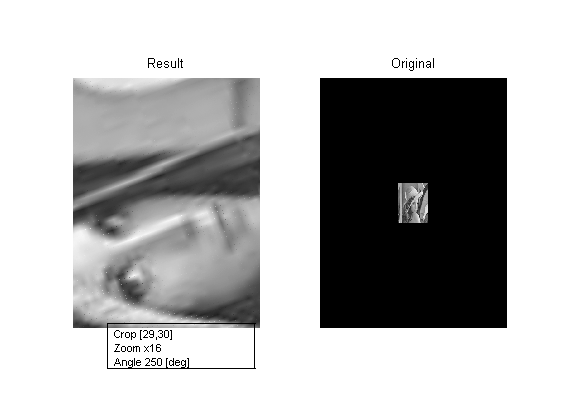


Figure - Simulation result 3

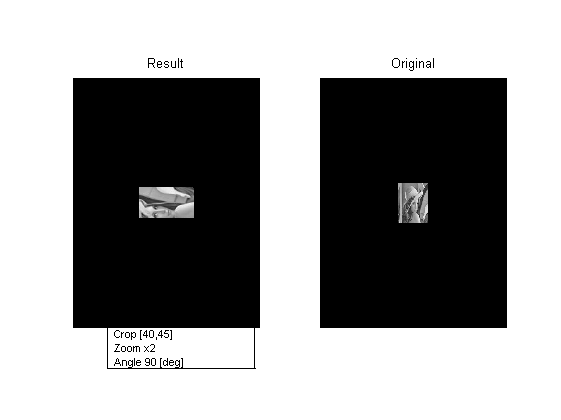


Figure - Simulation result 4

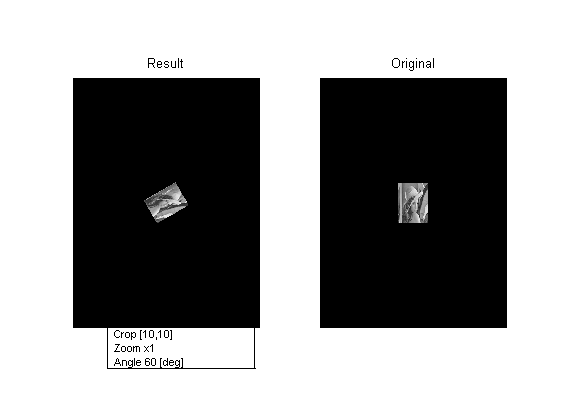


Figure - Simulation result 5

## Bilinear Interpolator

### General description

The unit calculates the mean average of 4 given gray-scale values, achieved by 2 stage calculation.

Figure - Bilinear Interpolator illustration

### The algorithm

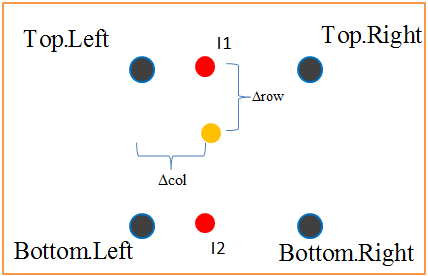


Figure - Bilinear algorithm scheme



The result is greyscale pixel, represented by 8 bits (256 grey levels).

### µArchitecture



Figure - Micro Arch. Bilinear Interpolator

### Inputs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **description** | **type** | **size** | **Recived from** |
| Pix\_1\_in | Value of 1st requested pixel | Std\_logic\_vector | 16/**8**? | Local\_register/SDRAM? |
| Pix\_2\_in | Value of 2nd requested pixel | Std\_logic\_vector | 16/**8**? | Local\_register/SDRAM? |
| Pix\_3\_in | Value of 3rd requested pixel | Std\_logic\_vector | 16/**8**? | Local\_register/SDRAM? |
| Pix\_4\_in | Value of 4th requested pixel | Std\_logic\_vector | 16/**8**? | Local\_register/SDRAM? |
| delta\_row | Holds the delta of row, for bilinear-interpolation | Std\_logic\_vector | 7 (trig\_frac\_size\_g-1 downto 0) | Addr\_calc |
| delta\_col | Holds the delta of col, for bilinear-interpolation | Std\_logic\_vector | 7 (trig\_frac\_size\_g-1 downto 0) | Addr\_calc |

### Outputs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **description** | **type** | **size** | **Destination** |
| Pixel\_res | Gray scale value for output image | Std\_logic\_vector | 16/**8**? | WB to Sdram |
| Pixel\_valid | Valid signal for result | Std\_logic | 1 | manager |

### Simulation

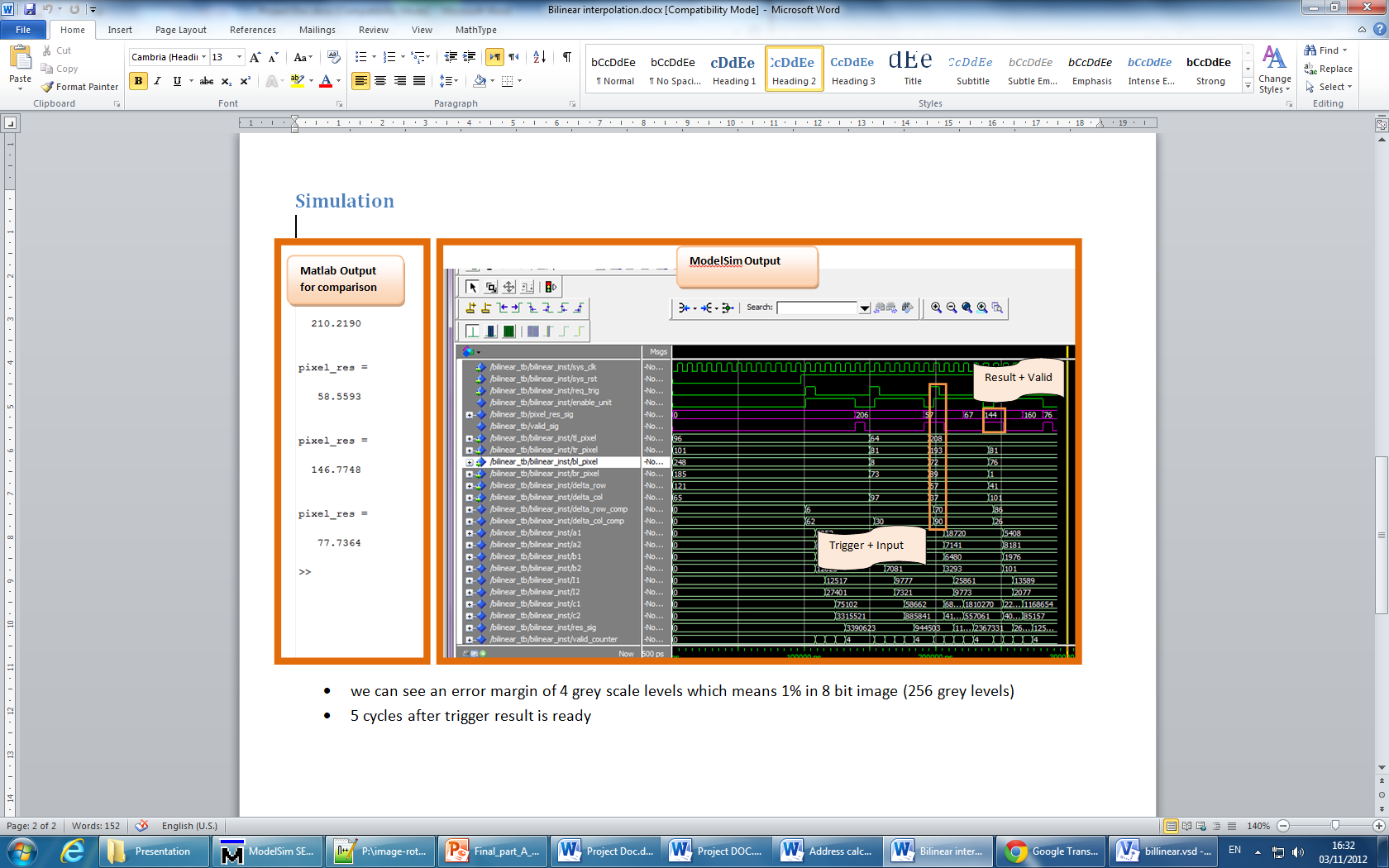


Figure - Bilinear simulation

* we can see an error margin of 4 grey scale levels which means 1.5% in 8 bit image (256 grey levels)
* 5 cycles after trigger result is ready

## Image Manipulation Manager

### General Description

The unit is responsible for executing the image manipulation algorithm. It is implemented with three finite state machines:

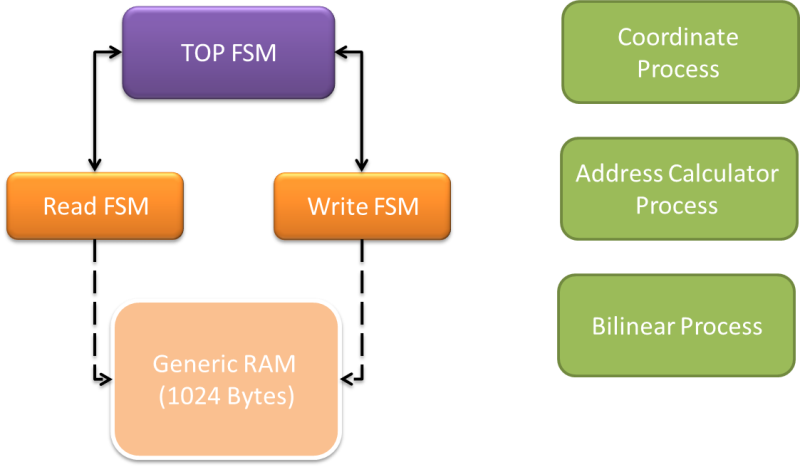
1. Top FSM- responsible for the general image manipulation process.
2. Write FSM – send write requests to memory management block.
3. Read FSM – sends read requests to memory management block.

All read and write transactions are executed in DEBUG mode.

### Top Finite State Machine



### µArchitecture



* TOP FSM – responsible for the general control of the unit (control and data flow).  
  Activates read/write state-machines and other process according to chart above.
* Coordinate Process - counter process, which increments the coordinates of the result image, triggered by top FSM. Counts only within region of interest as defined by row\_start, col\_start signals and inner constants and generics.
* Address Calculator process – triggers the address calculator into action.
* Bilinear process – triggers the bilinear into action.
* Read FSM – reads two pairs of pixels from the memory (in DEBUG mode) according to addresses given by address calculator. Each pair is read in a different transaction since they are located in different addresses in the memory.
* Write FSM – when RAM is full, the process writes the contents of the RAM to SDRAM with debug transaction to memory block.
* Generic RAM – 8 bit x 1024 ram. 8 bit for each pixel (256 grey-scale) and 1024 is the write back burst length, can be altered by *wb\_burst\_int* in the code.  
  *wb\_burst\_int* should such that for some n:  
  .  
  the condition demands that the image will have an integer number of bursts.

The SDRAM on DE2 has 4 banks, the current system configuration uses only two of them(bank 0 and bank 1). Each bank was divided into two such that:

Bank 0 :

* bottom addresses("msb" = 0) – holds original image that was received from software(via rx\_path). This image is currently used by image manipulation block for manipulation purposes
* top addresses("msb" = 1) – holds manipulated image that was received from image manipulation block.

Bank 1 :

* bottom addresses("msb" = 0) – holds new image that was received from software(via rx\_path)
* top addresses("msb" = 1) – holds manipulated image that was received from image manipulation block.

### Inputs

### Outputs

### Simulation

# Working methods

During the design, few design and management technics were implemented. Some of them were planned (advised by project leader and other students), and some discovered during the process.

## Top down design

As recommended, this method was implemented from the beginning of the design.

Especially in this kind of project (integrate into other top project), it was necessary that first one major block would connect to the top block. Later, all the other sub-components are edged and can be design.

## Pipeline

On this project, two major arithmetic calculations are operated:

* Coordinate translation in addr\_calc
* Grey level calculations on bilinear\_interpolator

These two operations are "heavy consumers" in terms of hardware, thus using pipeline design is very effective.

Each multiplying operation was separated (since the multiply operation is the longer operation) and was done on the next clock.

The result- throughput was improved and was compatible with the project requirements.

The next figure might visualize the method:

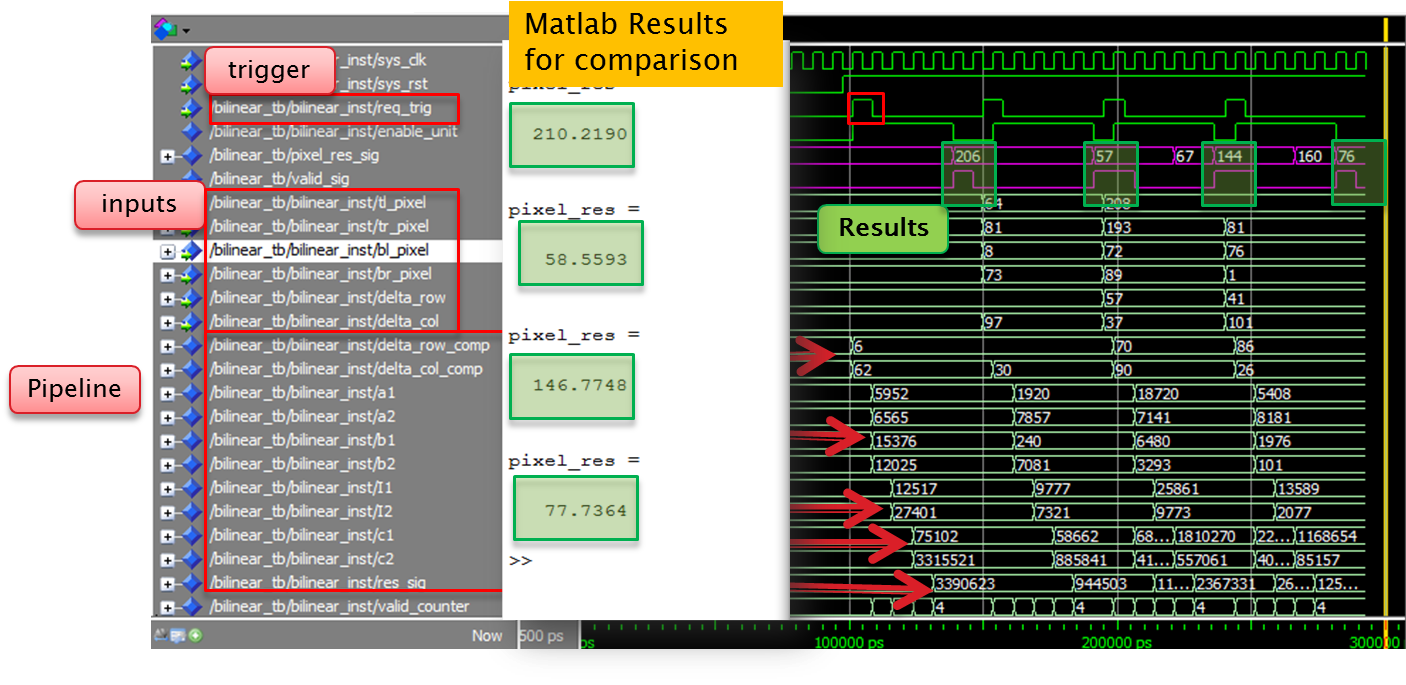


Figure - Bilinear results- compare with Matlab

## Test bench

In order to test the components, test benches were used for each component.

The purpose of this method is to inject all the possible inputs to the tested component, and record the outputs.

## Component documenting

On this project few components were created, in time intervals of weeks.

It seems that during the design it is the best time to document all the calculations, ideas, technics ext.

For each component, a document file was created in order to retrieve ideas and calculations.

Moreover, it turned out that this method made the presentations and the project document very easy to implement.

## Result comparison with Matlab

As mentioned before, 2 arithmetic calculations were operated. Each operation deals with 600x800 pixels, which means many output data.

In order to confirm the results, theoretical calculations were made by Matlab.

Since manual comparison is not practical, a Notepad++ compare plugin was used.

## Working with SVN

Synchronize file via SVN allowed working on the project from different end-points.

The option of code restore was used as well in several cases.

In addition, the SVN tags versions in every sync process.

# Summery

## Problems during the process

### Working with fractures

First version of addr\_calc used a fixed point package. Problemsoccurred during synthesis with the package.  
**Solution -** work with regular std\_logic\_Vector, with relevant adjustments.

Example – representing (sin(60))



### Trigonometric calculations

Cosine and Sine calculation were planned to be executed by hardware, but unfortunately they consumed a lot of hardware resources. **Solution-** calculate Cos/Sin by software (Matlab) and save it in a register.

### Timing issues

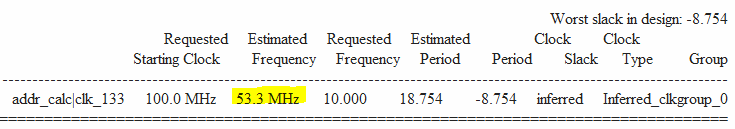
After initial synthesis, timing results did not meet the requirements.  


Figure - Synthesis timing report (1)

**Solution-** break arithmetic calculations into parts (piping).

Each complex calculation was divided to several stages in several processes in order to increase the frequency

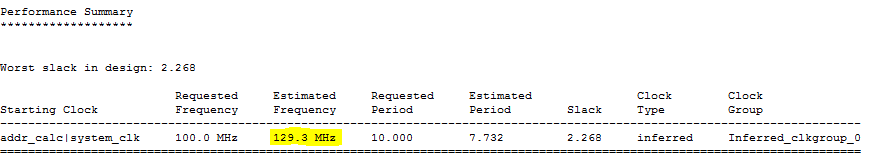


Figure 38- Synthesis timing report (2)

## Architectural changes

In the original design, the top block of image manipulation included addr\_calc and addr\_converter. During the design, the 2 blocks were combined into one block seemed inevitable.

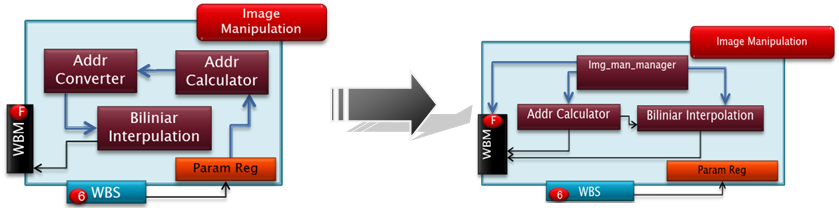


Figure - Arch modification

## Coding conclusions

* Working with fractions is simpler using the Fixed Point Package, but the downside is with synthesis. And therefore it is only useful for simulations
* Divide the design into several processes makes the design more readable, organized, efficient and manageable.
* Using pipeline implementation improves throughput but is harder to implement, and require more space and power.

## Project Conclusions

The project goal was designing a system which is able to implement the following features:

* Image rotation
* Image zoom
* Crop image

The main guidelines during the design were:

* Frequency requirements- 100 MHz
* Image quality- minimum distortion

So far, the goals were achieved and even more than expected (e.g. frequency up to 129 MHz, image quality can be controlled by generics- fraction precision).

The next stage would be integrating our top block within the MDS project (Beeri), including simulations and synthesis.

# Abbreviations

* SDRAM – Synchronous Dynamic Random Access Memory
* RAM – Random Access Memory
* TX – Transmission
* RX – Receive
* FIFO – First in First out
* PLL – Phased Locked Loop
* TB – Test bench
* SOF – Start of frame
* EOF – End of frame
* CRC - Cyclic redundancy check
* MP – Message Pack
* TB – Test Bench
* UART – Universal Asynchronous Receiver Transmitter
* VESA - Video Electronics Standards Association
* VGA - Video Graphics Array
* DVI - Digital Visual Interface
* IP – Intellectual Property