Technion

Electrical Engineering Department

High Speed Digital System Lab

Picture Manipulation using Hardware

Version: 1.03

Last update: 10.6.2013

Written By: Ran Mizrahi , Uri Tsipin

Guided by: Moshe Porian

Table of Contents

Table of Figures

1 Introduction

1.1 Abstract

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

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

1.2 Applications

1.2.1 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 rotating the image and zooming in/out of the image.

Figure 1- Pilot Helmet

1.2.2 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 the surveillance more comfortable.

Figure 2- Intelligence Devices

1.2.3 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.

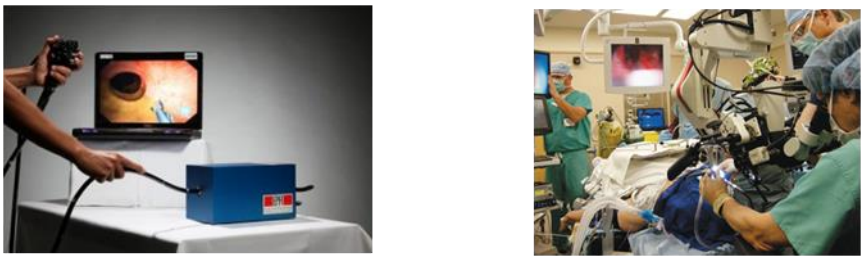


Figure 3- Mrdical Applications

1.2.4 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- Printer Application

2 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.

3 Project requirements

- Input Image resolution must be 600x800 and monochromatic.
- Output image resolution will be 512x512.
- 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.

4 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.

4.1 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.

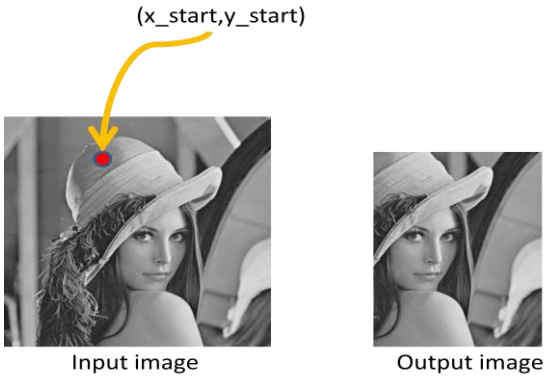


Figure 5 – Example Of Image Cropping

4.2 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.

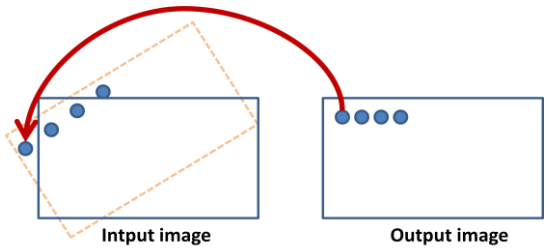


Figure 6 - Example of 30 degree rotation

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

4.2.1 Calculation of source pixel address

Assume the pixel coordinate in the original image matrix is (x, y)

For a rotation over an angle θ, (x, y) in the original image is mapped onto the point (x', y') in the output image.

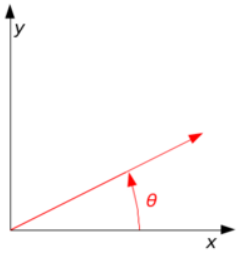


Figure 7 - A counterclockwise rotation of a vector through angle θ

The relation between the points is:

$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

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:

$$x = x'\cos(\theta) - y'\sin(\theta)$$
$$y = -x'\sin(\theta) + y'\cos(\theta)$$

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 (x, y) values and evaluate the grey level using bilinear interpolation.

4.2.2 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 surrounding 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

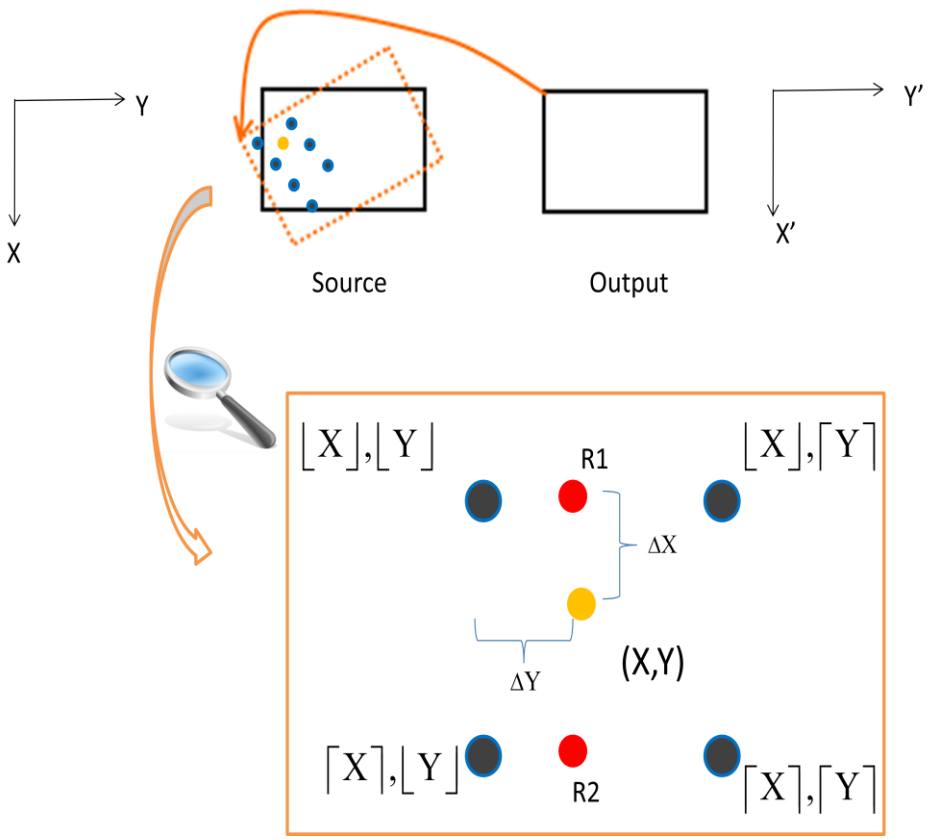


Figure 8- bilinear illustration

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

$$R1 = \text{Source}\left(\lfloor X \rfloor, \lfloor Y \rfloor\right) \times (1 - \Delta Y) + \text{Source}\left(\lfloor X \rfloor, \lceil Y \rceil\right) \times \Delta Y$$
$$R2 = \text{Source}\left(\lceil X \rceil, \lfloor Y \rfloor\right) \times (1 - \Delta Y) + \text{Source}\left(\lceil X \rceil, \lceil Y \rceil\right) \times \Delta Y$$
$$\text{Output}\left(X', Y'\right) = R1 \times (1 - \Delta X) + R2 \times (\Delta X)$$

4.3 Zoom

4.3.1 Mathematical background

The zoom function is achieved by using the mathematical operation of scaling the axis. For example- in one dimension: $f(x) = \sin(x)$

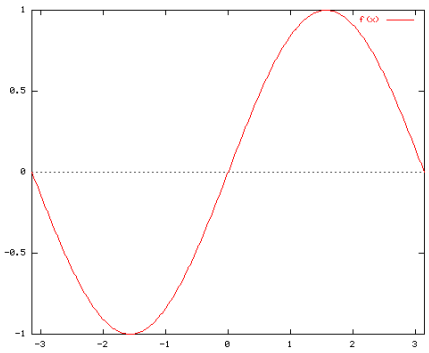


Figure 9- sine function

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

$$g(x) = f(ax) \qquad a > 1$$

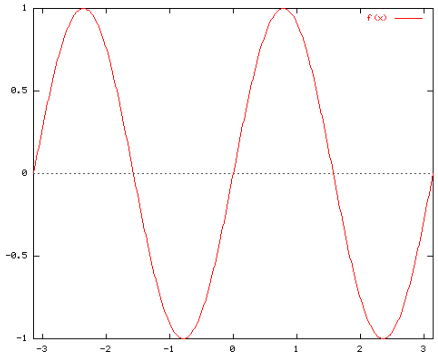


Figure 10- scaling function (1)

$$g(x) = f(ax) \qquad a < 1$$

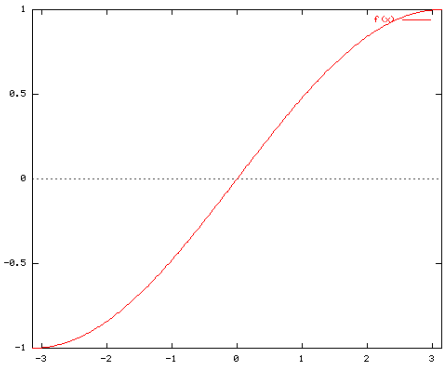


Figure 11- scaling function (2)

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

4.3.2 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:

$$\text{Output}\left(X',Y'\right)=\text{Source}\left(\frac{X}{\text{zoom factor}},\frac{Y}{\text{zoom factor}}\right)$$

4.4 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:

$$\text{Output}\left(X',Y'\right)=\text{Source}\left(\frac{X}{x\_factor},\frac{Y}{y\_factor}\right)$$

$$x\_factor=\frac{x\ \ axis\ \ source\ \ size-x\_start}{x\_output\_resolution}$$

and the same for Y axis.

5 General description

5.1 Project's Top Blocks Scheme

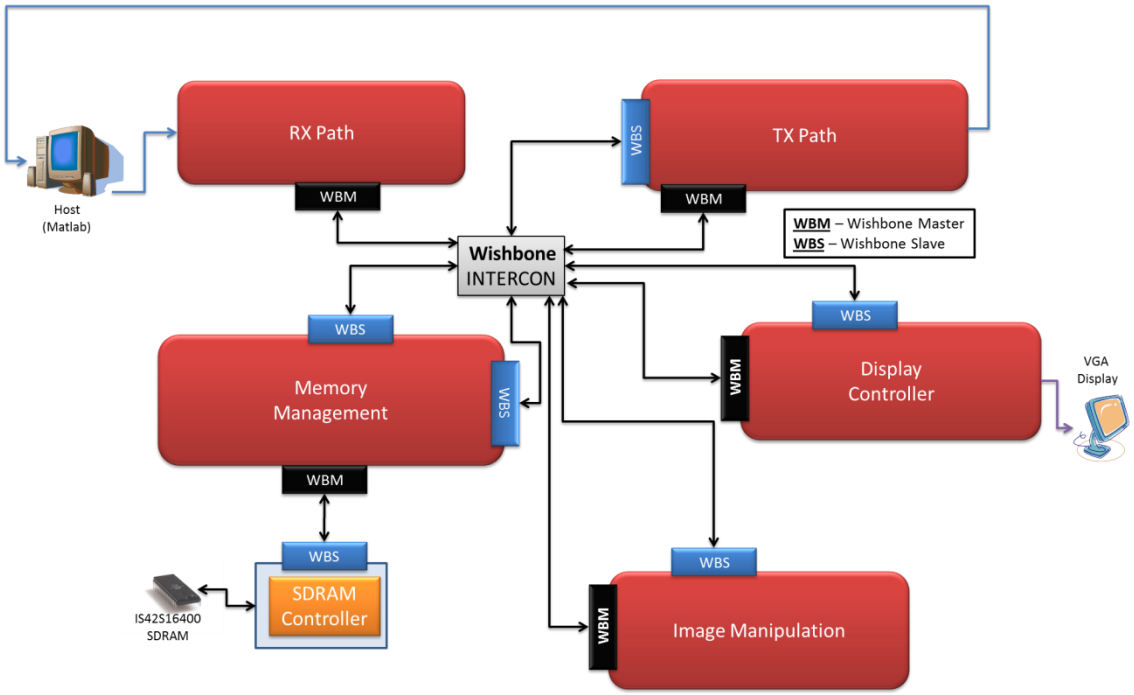


Figure 12- project main diagram

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

All changes from the previous project is explained in the blocks sections below.

5.2 Block Descriptions

5.2.1 Rx Path

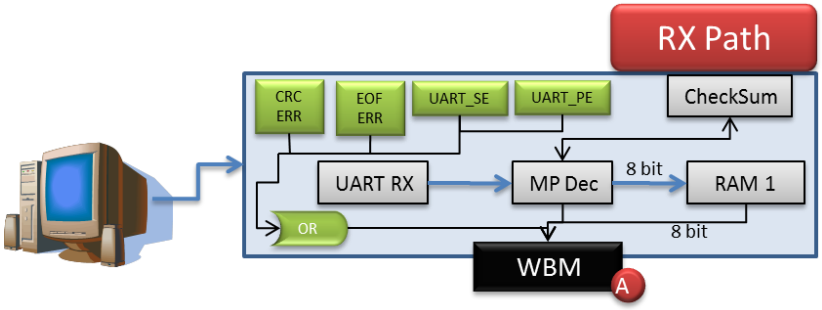


Figure 13- RX path block

Implemented with UART protocol.
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.

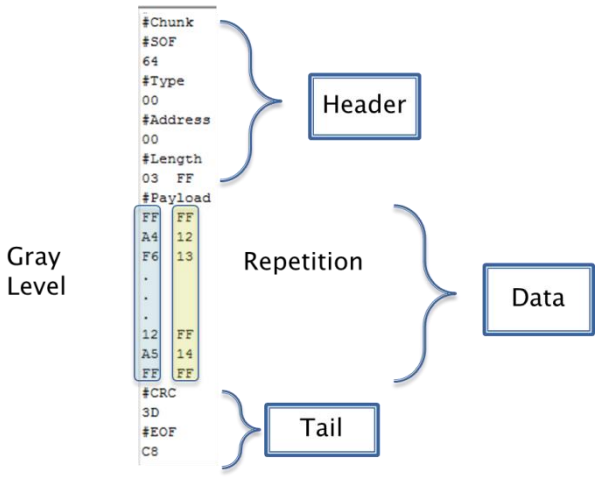
The UART message has the following structure:

1. **SOF** – Start of Frame
2. **Type** – Message type
3. **Address** – Address for the data
4. **Length** – Data length. Data length CANNOT be less than 1.
5. **Data (Payload)** – The wrapped data
6. **CRC** – CRC of Type, Address, Length and Data blocks
7. **EOF** – End of Frame

Figure 14- UART Message Pack Structure

5.2.1.1 Changes

The next change was made in RX_PATH block:



The original UART txt message included pixels repetitions counters, which are not needed on the current project. On the figure above, the repetition column (yellow) was removed. In addition CRC and length registers were updated.

5.2.2 Tx Path

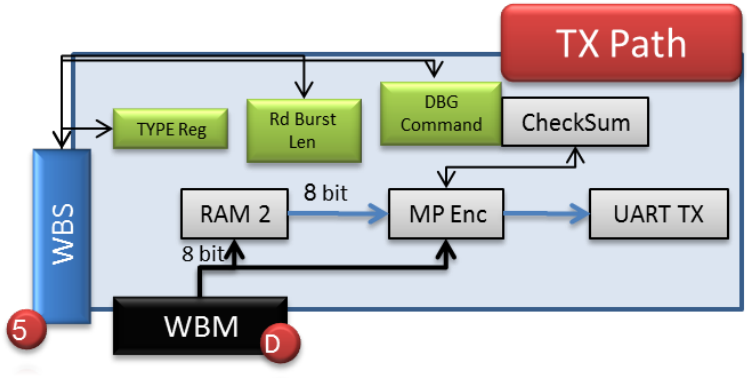


Figure 15- TX path block

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.

5.2.3 Memory Management

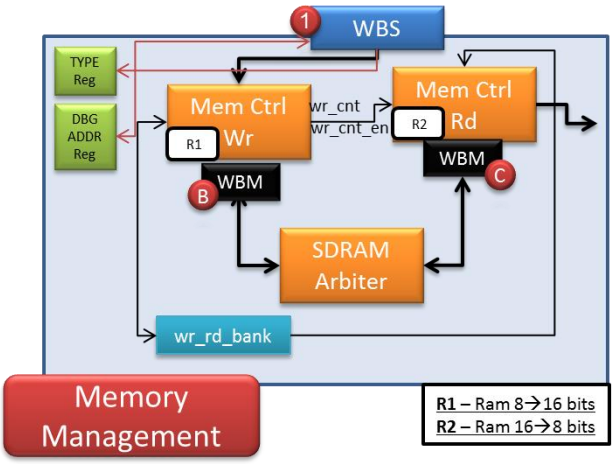


Figure 16- memory management block

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

Memory Management block includes two important registers:

1. **Type Register (address 0xD)**
   The relevant bits in this register are:
   - Bit 0: '0' for Normal Mode, '1' for Debug Mode
   - Bit 7: '0' for Data Transmission, '1' for Registers Transmission

2. **Debug Address Register (address 0x2-0x4)**
   These 3 registers have the address to write/read from in SDRAM at Debug Mode (that is controlled with the Type Register).

5.2.3.1 Changes

- Every bank was divided into 2 sections according to the 2nd MSB in address (1st MSB describes the bank, 2nd MSB is the bank divider).
- 2nd MSB='0' holds original image (no rotation)
  2nd MSB='1' holds manipulated image
- This change was made in mem_ctrl_rd_wbm under wbm_adr_out_proc

5.2.4 Display Controller

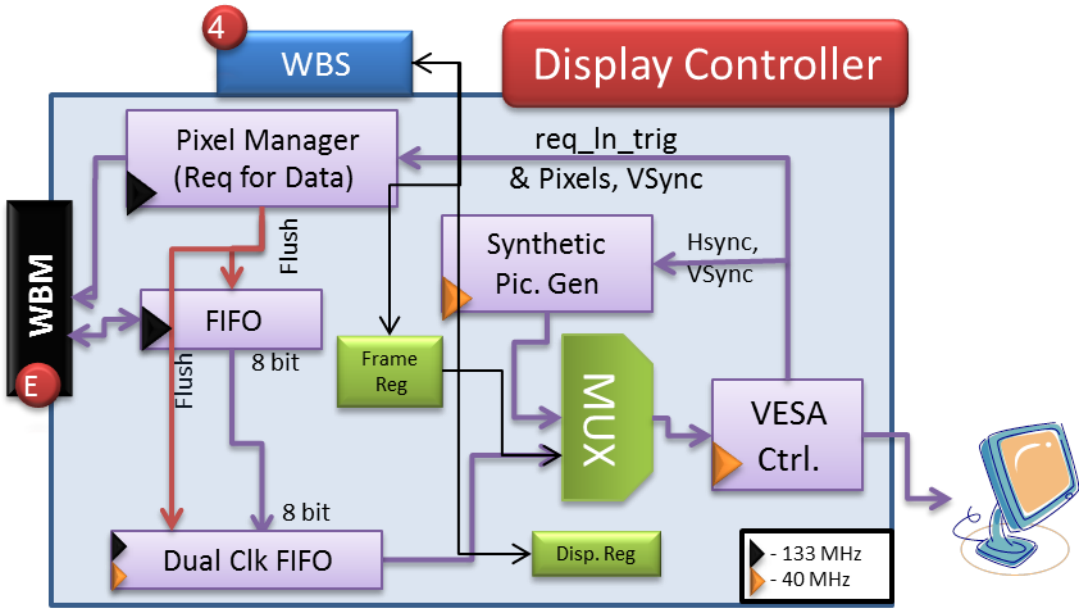


Figure 17- Display Controller block

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.

5.2.4.1 Changes

1. Pixel manager-
   - requested repetition parameters were disabled (pixel_mng.vhd)
   - counters of total requested pixels were updated
2. DC FIFO direct connection to FIFO- Runlen extractor was removed.
3. Top block- requested lines parameter was updated.

5.2.5 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)

5.2.5.1 Changes

- **Intercom Z**- WBM was added to the intercom to support read request from image manipulation block.
  WBS was added to support write requests to image manipulation block (registers).
- **Intercom Y**- WBM was added to the intercom to support write request from image manipulation block.

5.2.6 Image Manipulation

This is a general description of the new block in the system. More detailed description in section 6.

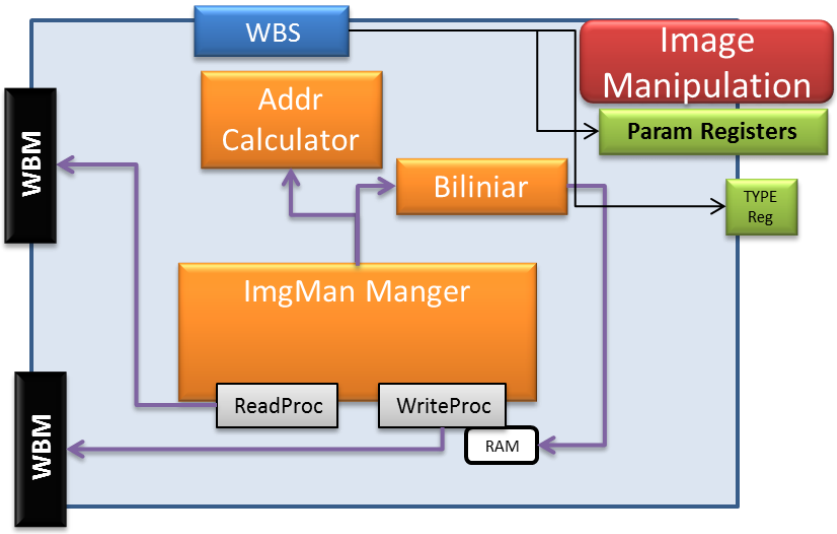


Figure 18- Image Manipulation block

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.

5.3 Data flow

Host transmits wrapped message of data packets, composed of bitmap image and user parameters to the **RX Path (1)**. Message is decoded, transmitted to the **Memory Management** block **(2)**, and stored into **Storage Device** (parameters into registers, image to the SDRAM) **(3).** Data is read from the memory, sent to image manipulation block **(4)**, and rewritten back to the SDRAM **(5)**. Then the manipulated image is transmitted to the **Display** through the **Display Controller (6)**.

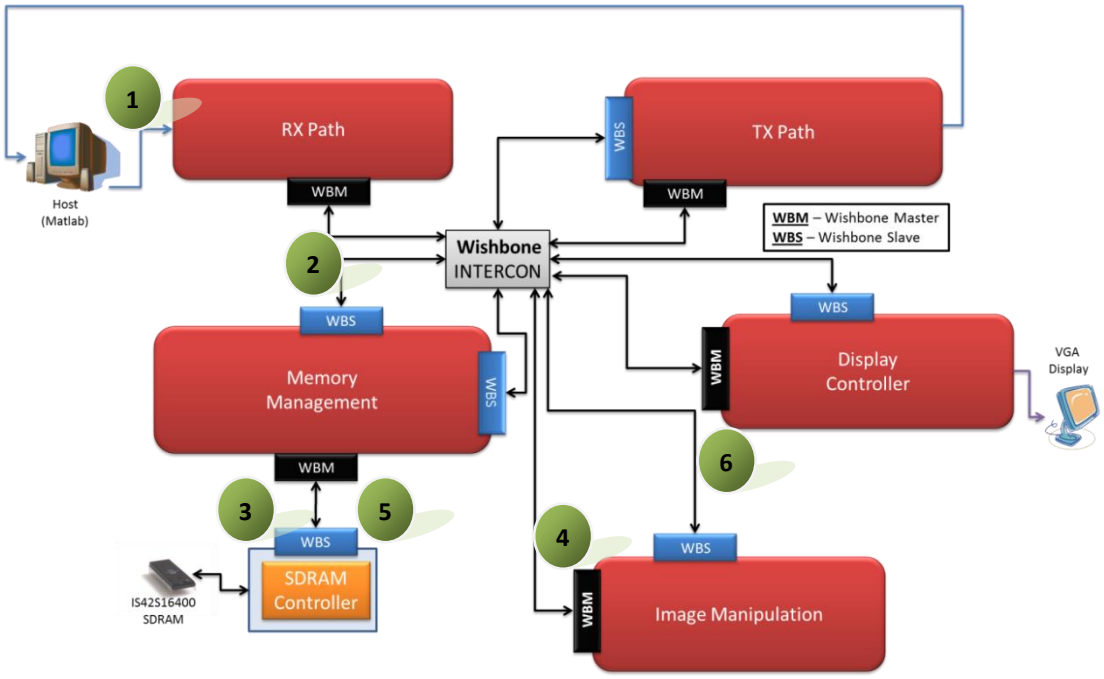


Figure 19- Data Flow

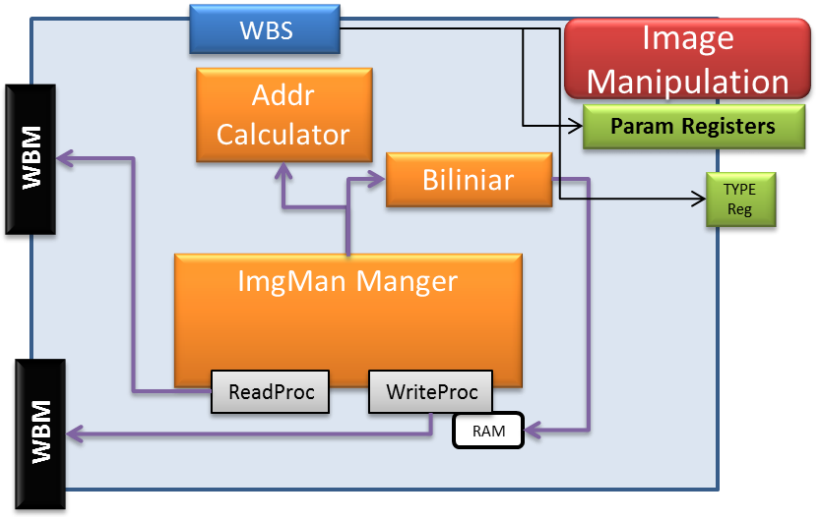
6 Implementation

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.

6.1 Image Manipulation – Top Architecture

6.1.1 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.

6.1.2 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.

6.1.3 Inputs

Name	description	type	Size(bit)
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	As described in Wishbone protocol	Std_logic_vector	10
wbs_tga_i		Std_logic_vector	10
wbs_dat_i		Std_logic_vector	8
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		Std_logic_vector	8
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		Std_logic_vector	8
rd_wbm_stall_i		Std_logic	1
rd_wbm_ack_i		Std_logic	1
rd_wbm_err_i		Std_logic	1

6.1.4 Outputs

Name	description	type	Size(bit)
wbs_dat_o	As described in Wishbone protocol	Std_logic_vector	8
wbs_stall_o		Std_logic	1
wbs_ack_o		Std_logic	1
wbs_err_o		Std_logic	1
wr_wbm_adr_o		Std_logic_vector	10
wr_wbm_tga_o		Std_logic_vector	10
wr_wbm_dat_o		Std_logic_vector	8
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		Std_logic_vector	10
rd_wbm_tga_o		Std_logic_vector	10
rd_wbm_cyc_o		Std_logic	1
rd_wbm_tgc_o		Std_logic	1
rd_wbm_stb_o		Std_logic	1

6.1.5 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.

6.2 Address calculator

6.2.1 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.

6.2.2 Illustration

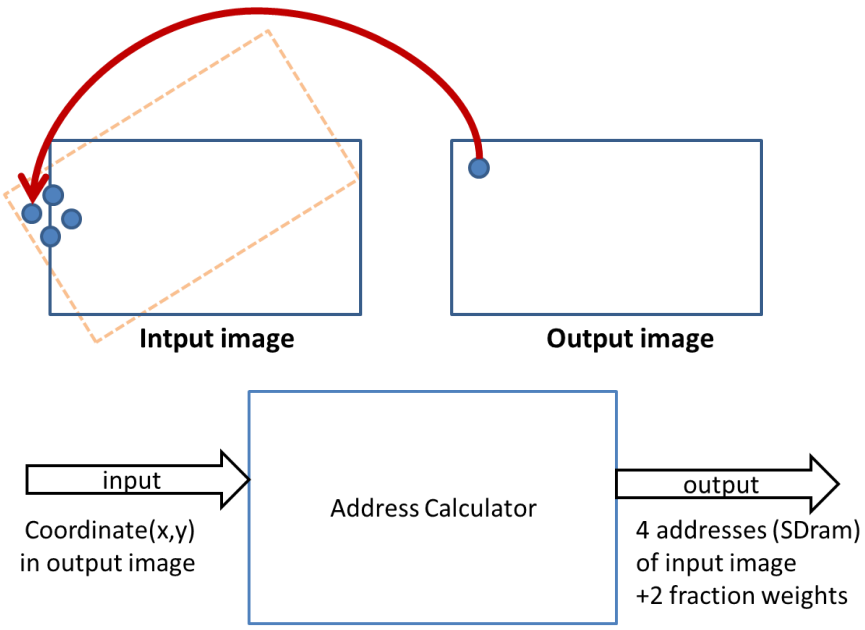


Figure 20- Addr_Calc illustration

6.2.3 The algorithm

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

$$X_{out} = Zoom\_Factor \cdot \left(X_{in} - \frac{Ver\_Output\_\mathrm{Re}solution}{2} \right) \cdot \cos(\theta) +$$

$$+ Zoom\_Factor \cdot \left(Y_{in} - \frac{Hor\_Output\_\mathrm{Re}solution}{2} \right) \cdot \sin(\theta) + \frac{1}{2} new\_frame\_x\_size$$

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:

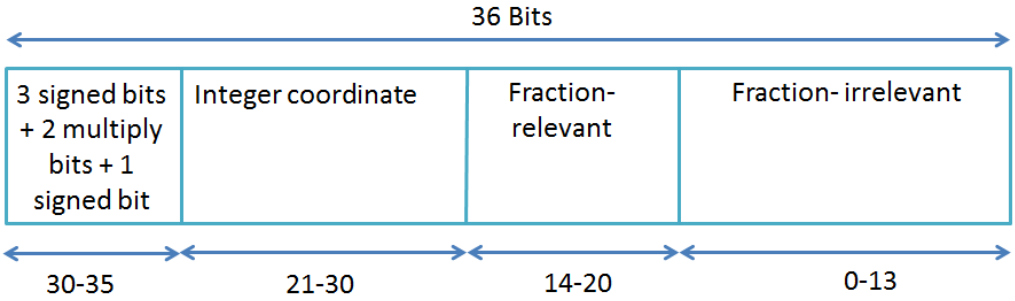


Figure 21- Output size vector

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).

6.2.4 µArchitecture

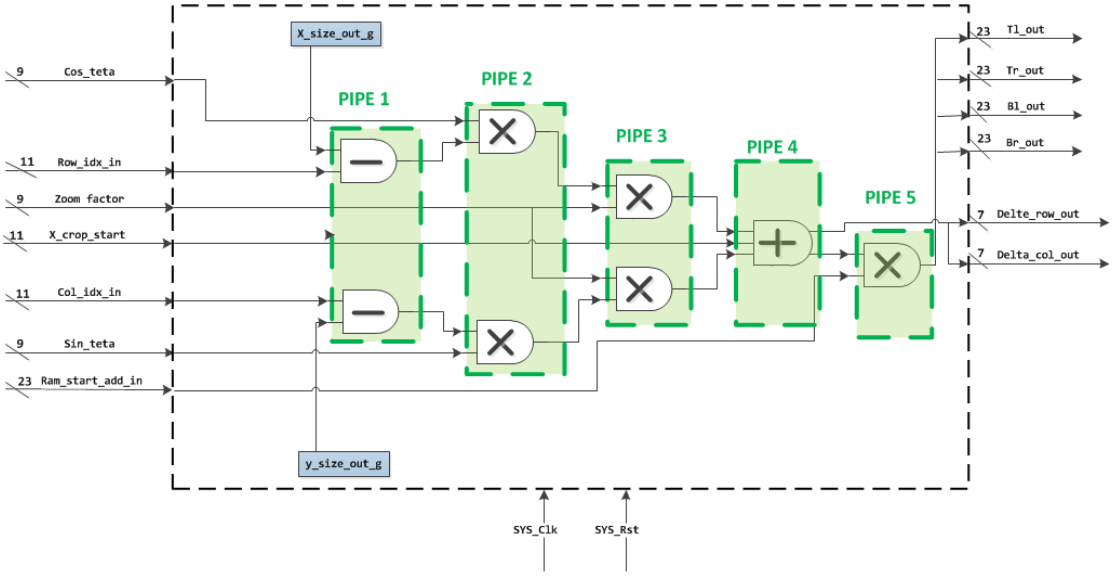


Figure 22- Micro Arch. – Addr_Calc

6.2.5 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)

6.2.6 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.

6.2.7 Simulations

first valid output

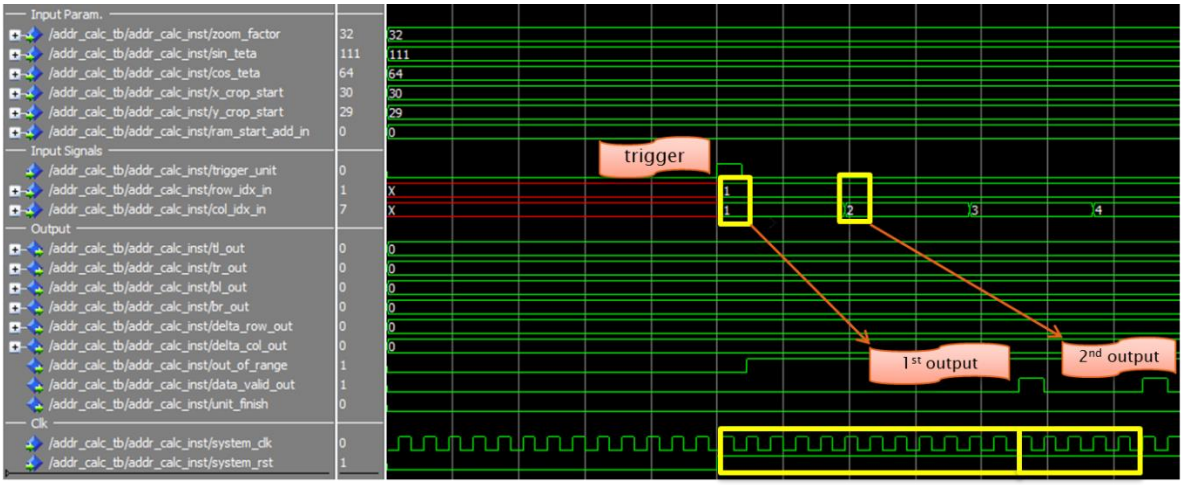


Figure 23- Addr_Calc simulation (1): first valid data

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

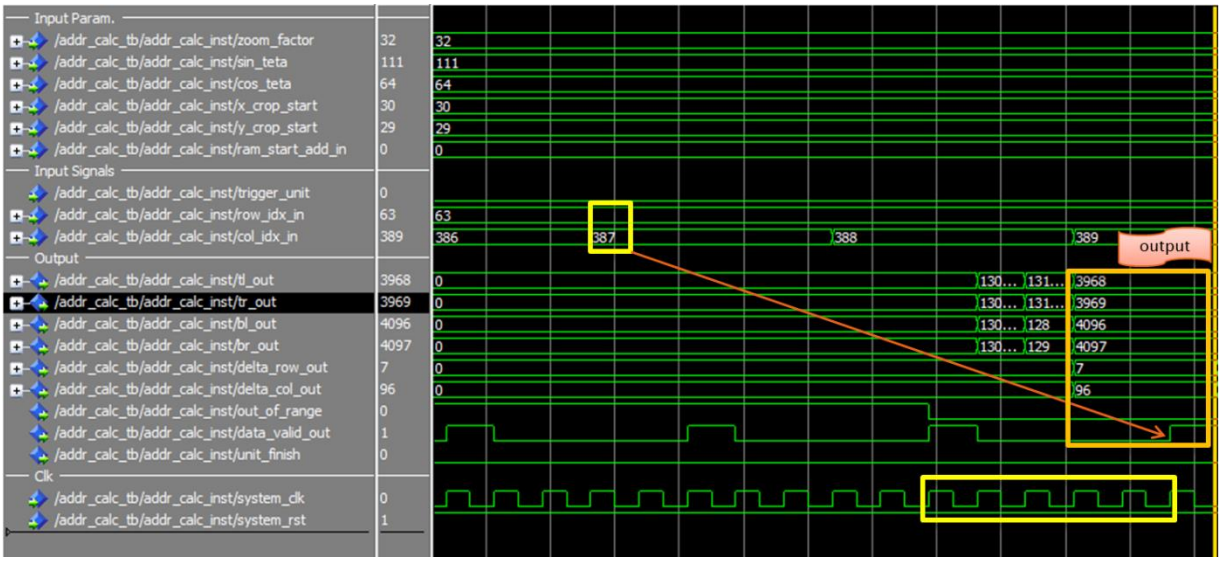


Figure 24- Addr_Calc simulation (2)- midway output

5 cycles between every adjacent result is maintained.

End of calculation process output

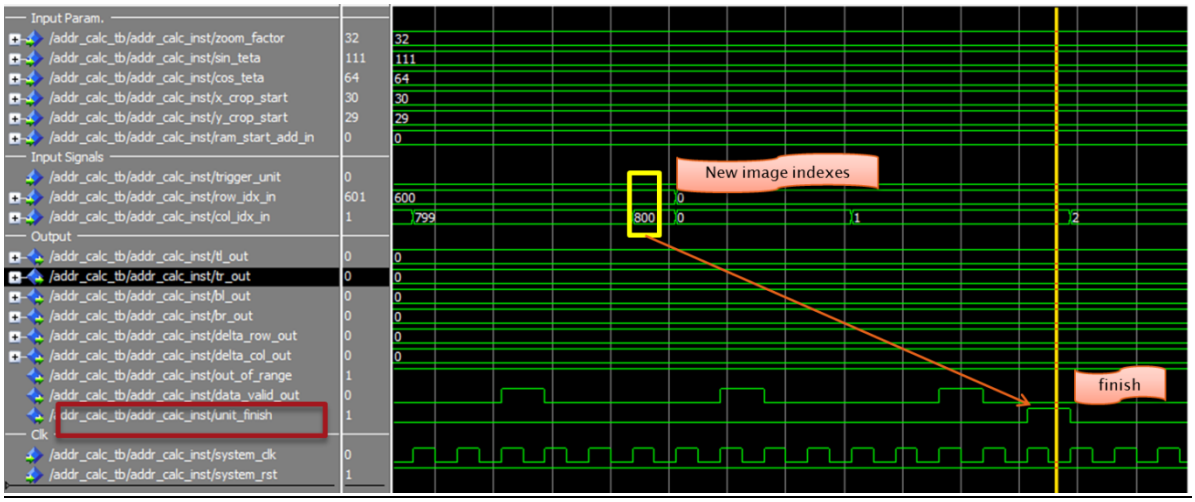


Figure 25- Addr_Calc simulation (3)- final procedure

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.

6.2.8 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:

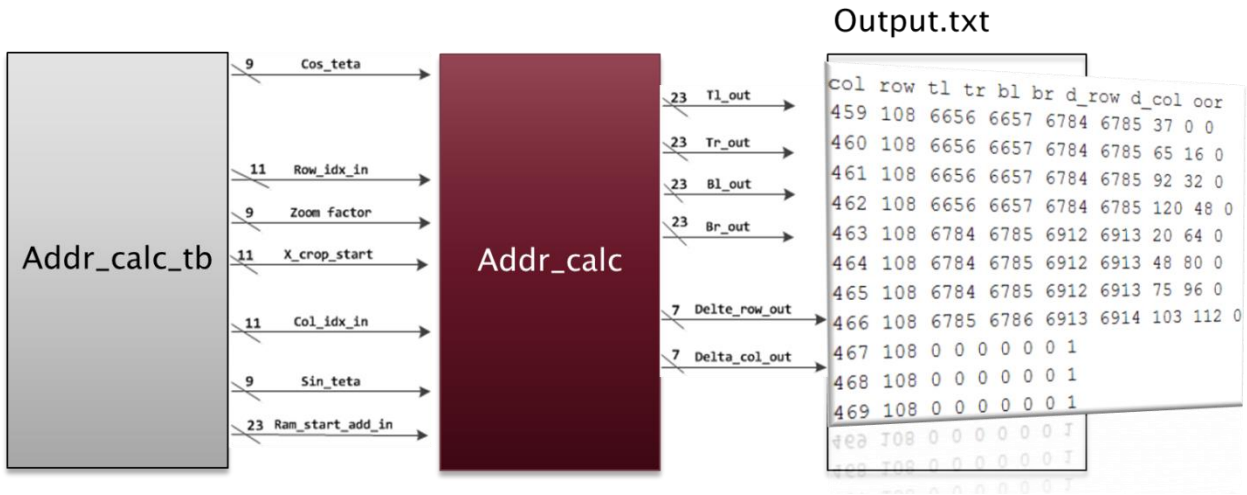


Figure 26- Addr_Calc Testing Process

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

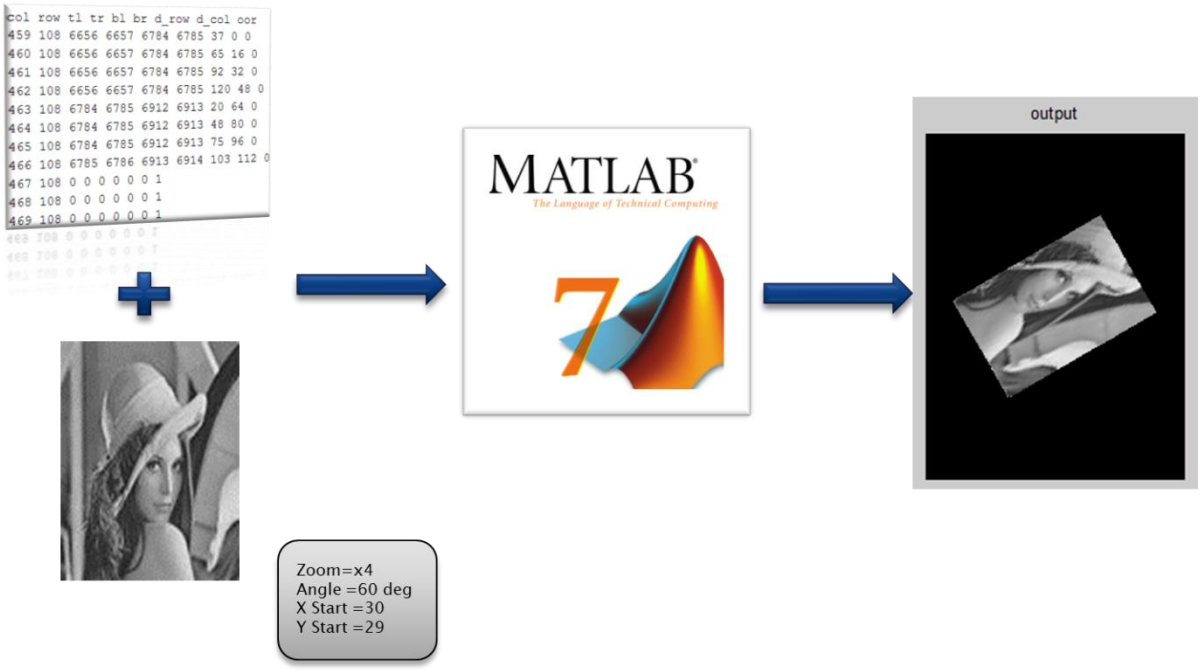


Figure 27- View the results of Addr_calc with Matlab

6.2.9 Matlab Test Results

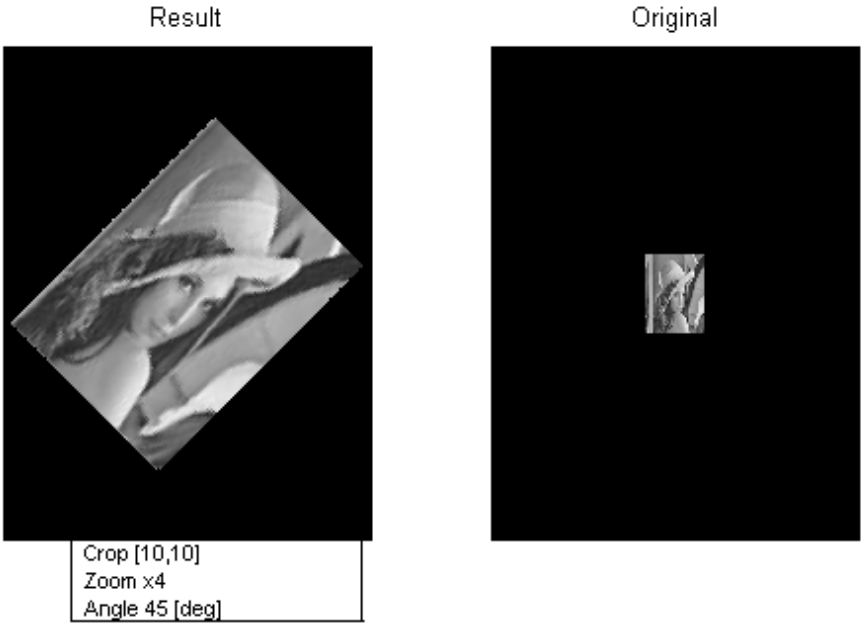


Figure 28 - Simulation result 1

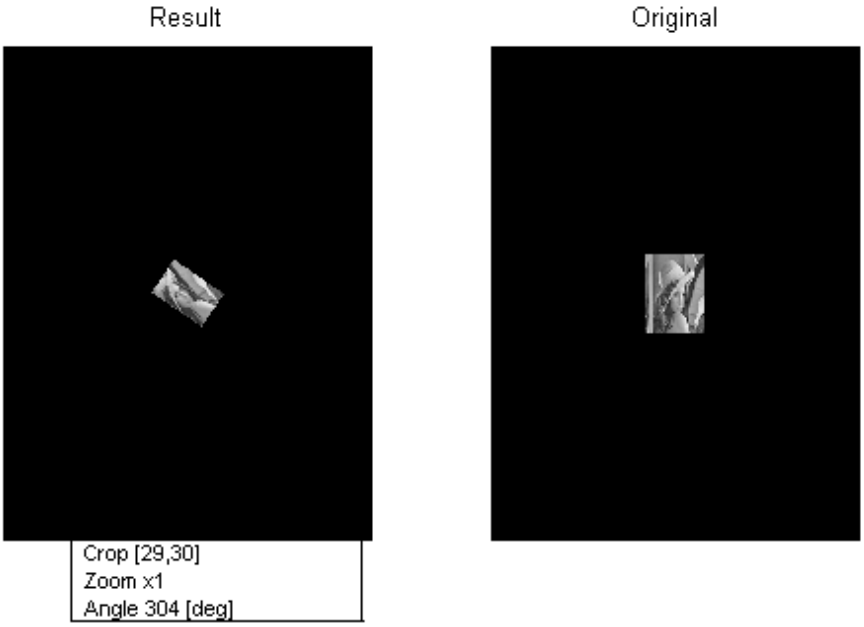


Figure 29 - Simulation result 2

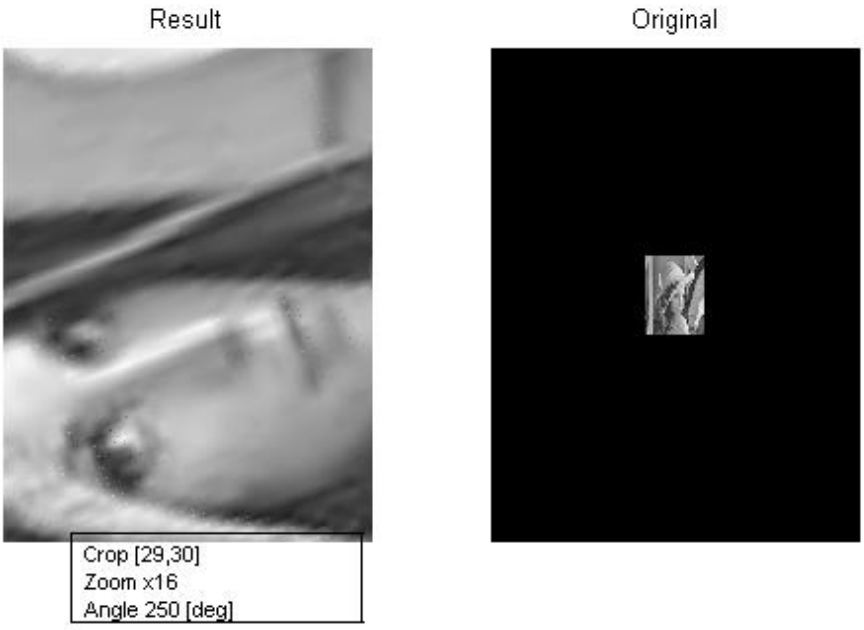


Figure 30 - Simulation result 3

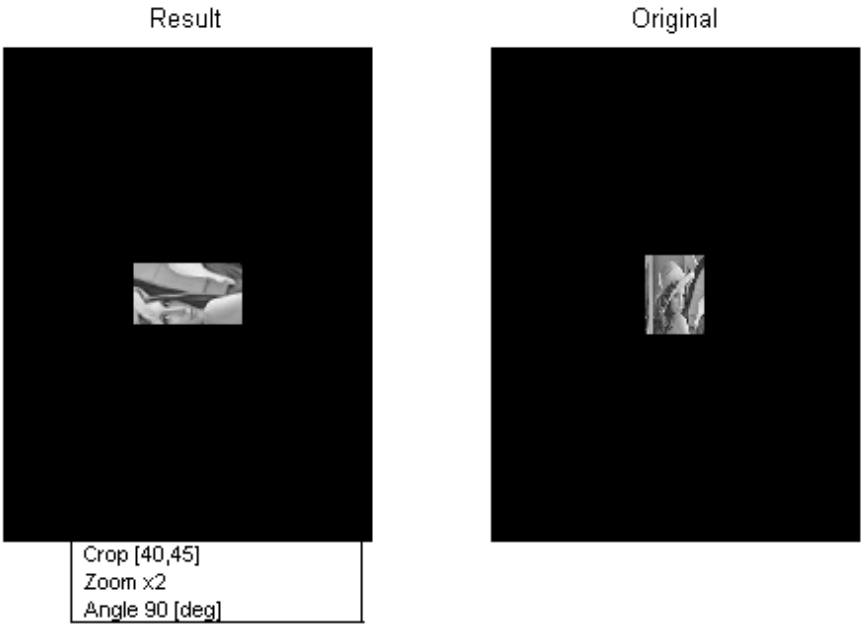


Figure 31 - Simulation result 4

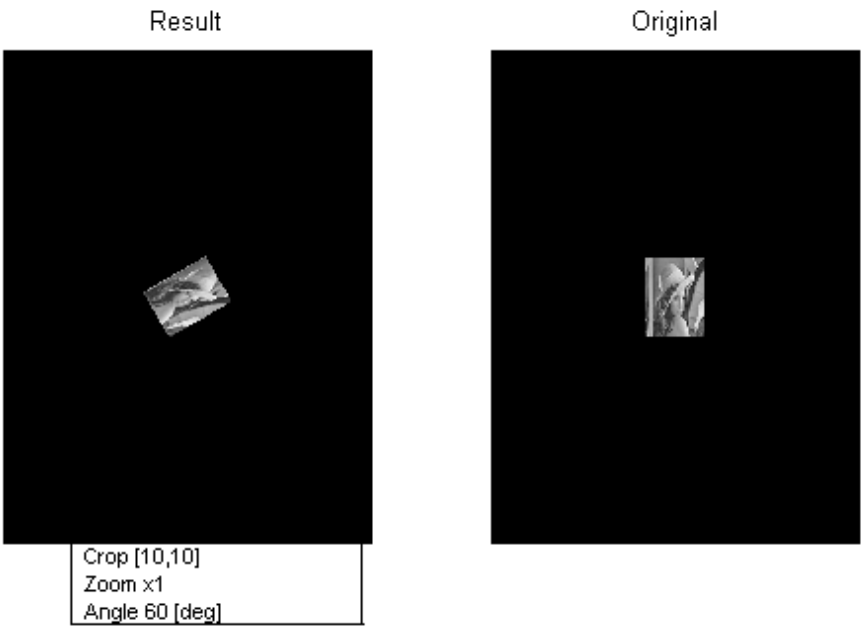


Figure 32 - Simulation result 5

6.3 Bilinear Interpolator

6.3.1 General description

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

Figure 33- Bilinear Interpolator illustration

6.3.2 The algorithm

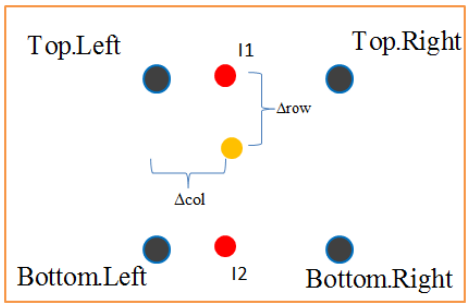


Figure 34- Bilinear algorithm scheme

$$I1 = Top.Left \times (1 - \Delta col) + Top.Right \times \Delta col$$

$$I2 = Bottom.Left \times (1 - \Delta col) + Bottom.Right \times \Delta col$$

$$result = I1 \times (1 - \Delta row) + I2 \times \Delta row$$

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

6.3.3 µArchitecture

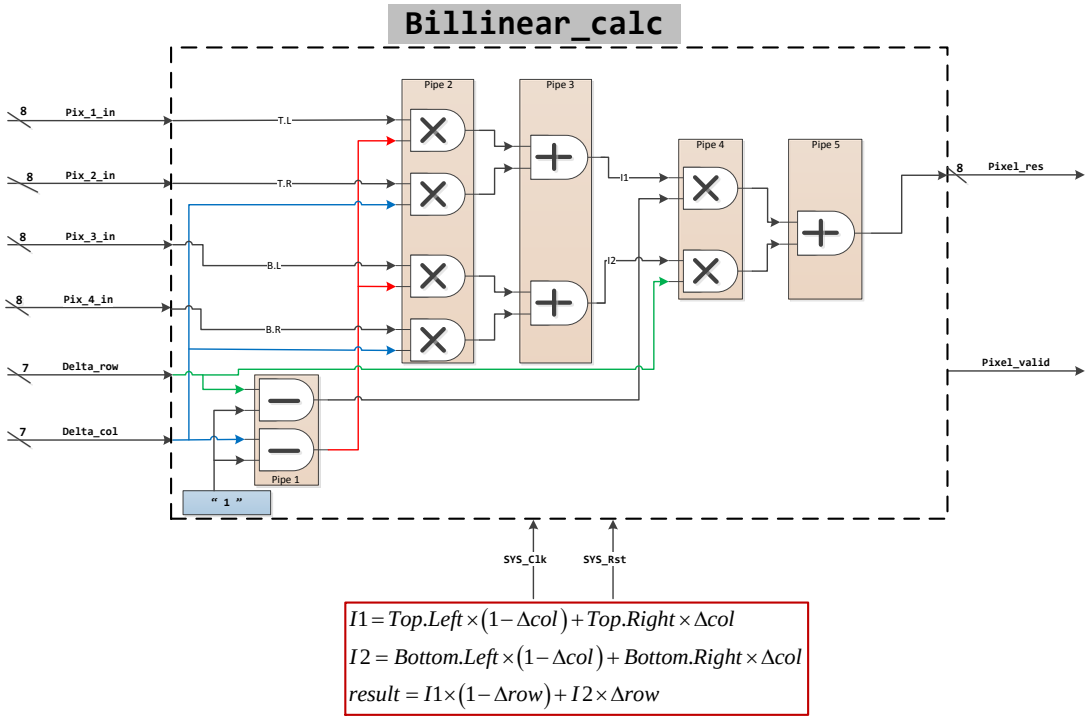


Figure 35- Micro Arch. Bilinear Interpolator

6.3.4 Inputs

Name	description	type	size	Received from
Pix_1_in	Value of 1st requested pixel	Std_logic_vector	8	SDRAM
Pix_2_in	Value of 2nd requested pixel	Std_logic_vector	8	SDRAM
Pix_3_in	Value of 3rd requested pixel	Std_logic_vector	8	SDRAM
Pix_4_in	Value of 4th requested pixel	Std_logic_vector	8	SDRAM
delta_row	Holds the delta of row, for bilinear-interpolation	Std_logic_vector	7	Addr_calc
delta_col	Holds the delta of col, for bilinear-interpolation	Std_logic_vector	7	Addr_calc

6.3.5 Outputs

Name	description	type	size	Destination
Pixel_res	Gray scale value for output image	Std_logic_vector	8	WB to Sdram
Pixel_valid	Valid signal for result	Std_logic	1	manager

6.3.6 Simulation

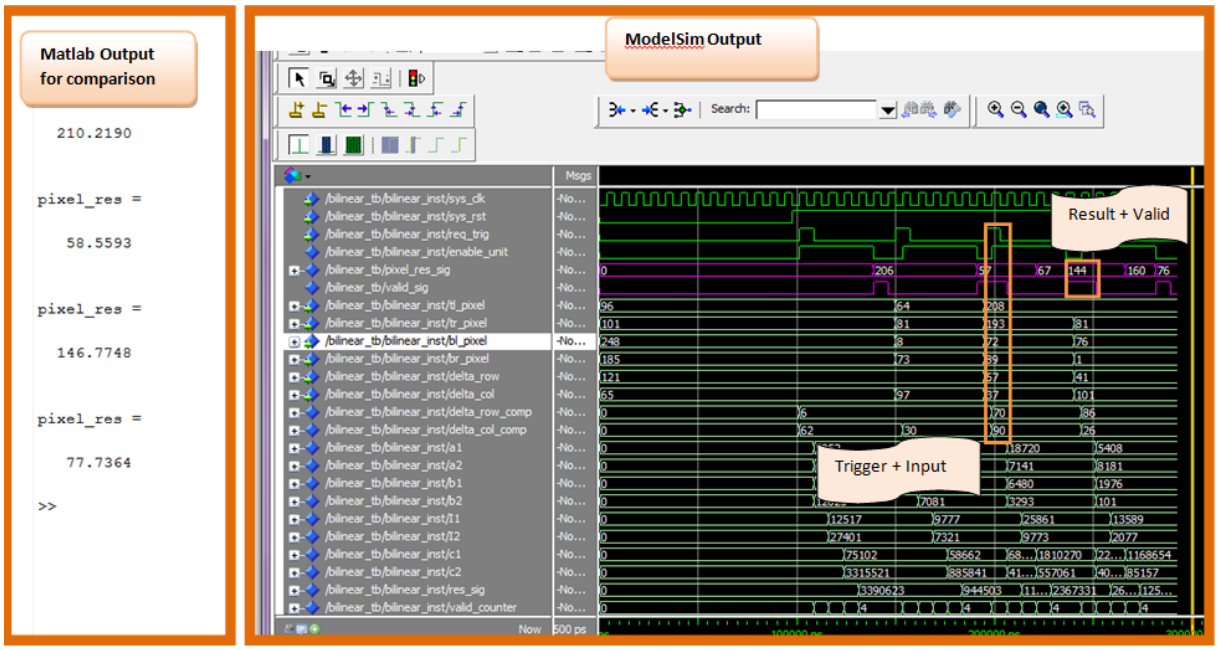


Figure 36- Bilinear simulation

- 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

6.4 Image Manipulation Manager

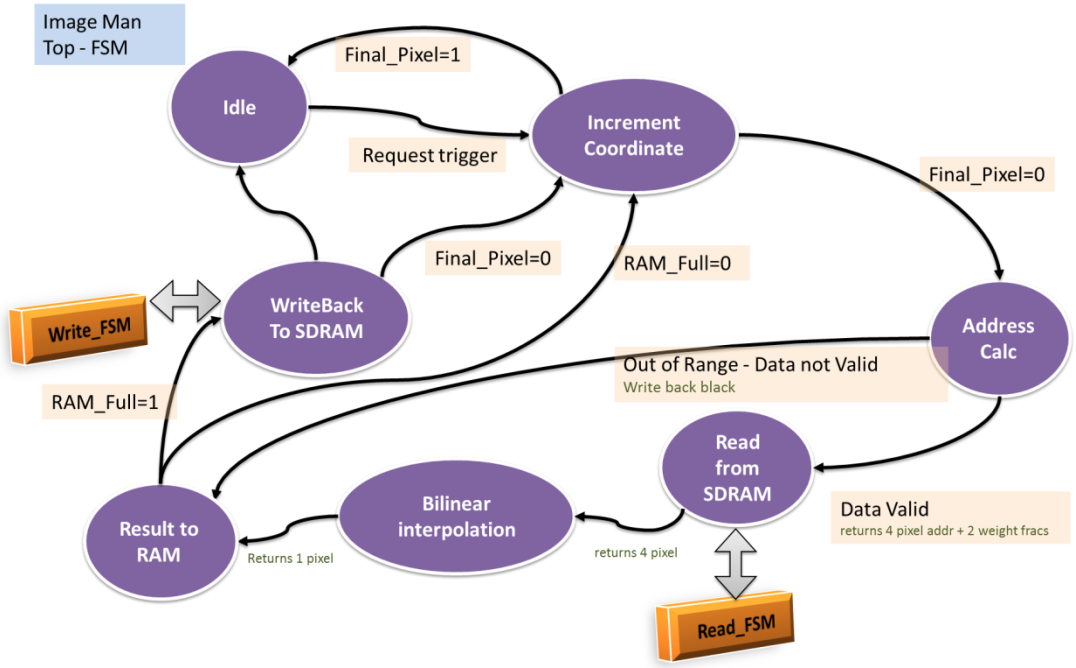
6.4.1 General Description

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

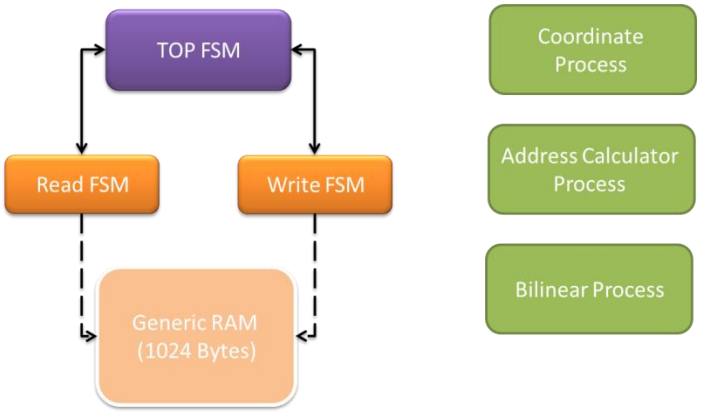
1. Top FSM- responsible for the general image manipulation process.
2. Write FSM – send write requests to memory management block.
3. Read FSM – sends read requests to memory management block.

All read and write transactions are executed in DEBUG mode.

6.4.2 Top Finite State Machine



6.4.3 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:
  $n \times wb\_burst\_\text{int} = \text{img\_hor\_pixels\_g} \times \text{img\_ver\_lines\_g}$.
  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.

6.5 Timing

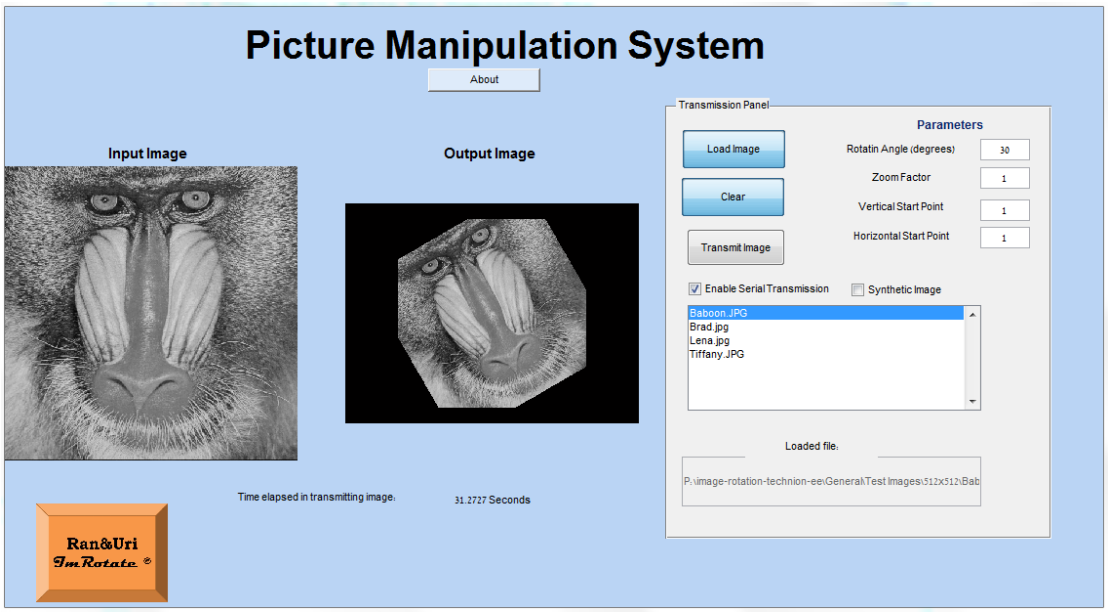
The latency of the system, which includes: read image, execute the manipulation, write back to memory and finally ready for display it to VESA: 0.4354 sec (435422720 ns).

7 Graphical User Interface

The system requires a graphical user interface in order to send data to the DE2 board. Current GUI implemented by MATLAB, and includes the following fill-in fields:

1. Load Image- Upload image from selected location
2. Processing parameters- angle, zoom, crop.
3. Serial transmission enable option
4. Transmit data- start to send image to DE2
5. Synthetic image- for debug purpose

The GUI also includes preview of the input image and the output image.



Link to the GUI user manual file:

http://image-rotation-technion-ee.googlecode.com/svn/Docs/GUI%20Tutorial.docx

8 Testing

8.1 Lab tests

8.1.1 Components

The full system includes:

1. Altera DE2 board with Cyclone II FPGA and SDRAM
2. Computer
3. MATLAB GUI
4. JTAG connector to the computer, to burn the FPGA
5. RS-232 connector to serial port in the computer
6. VGA connector between the board and the screen for display
7. Standard computer display

The figure below includes the listed components:

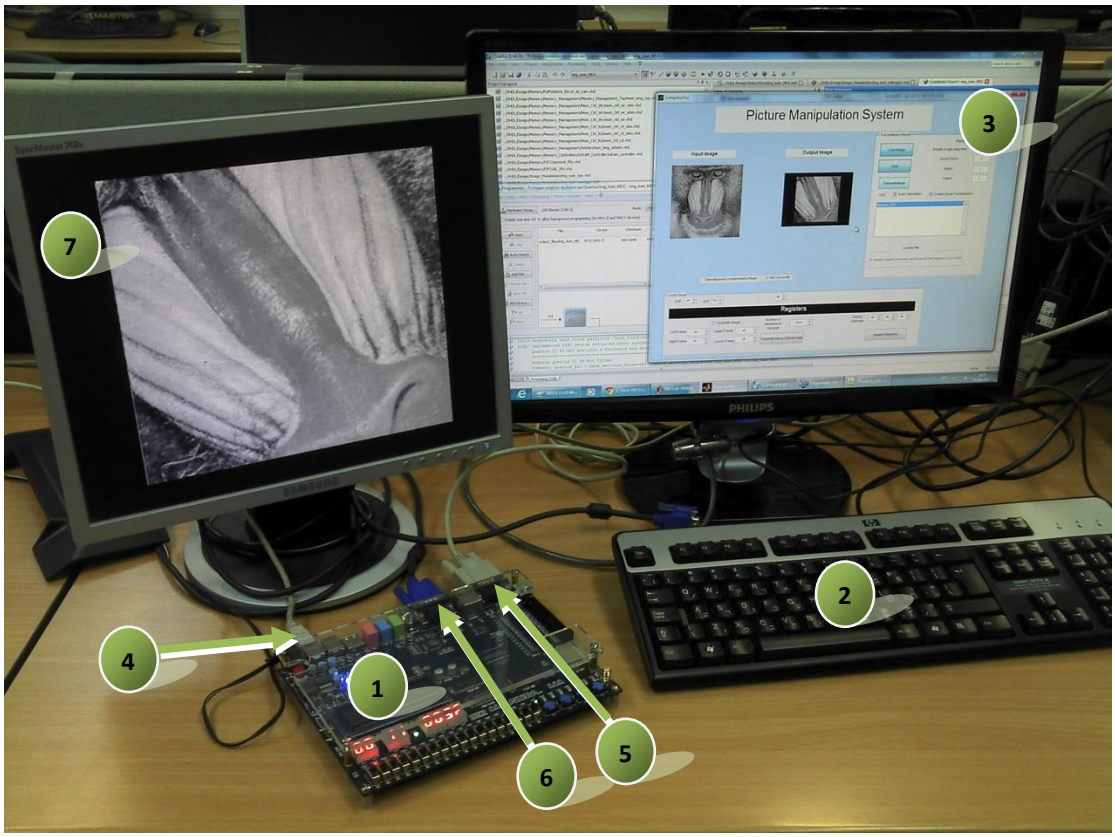


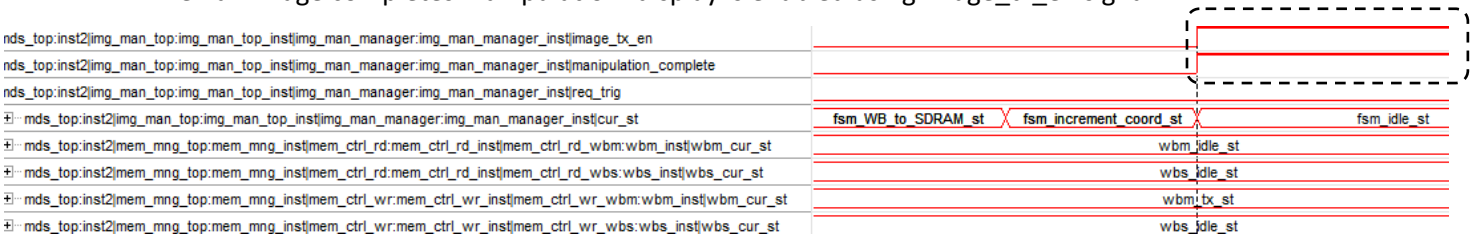
Figure 37- Lab testing components

8.1.2 Signal Tap

After completing the simulation stage, the system was ready to be burn on the DE2. Burn success might take few attempts, and a debug tool is needed. Therefore, on this phase of the project, signal tap (Quartus built-in tool) was used.

The figure below demonstrates the use of this tool:

When an image completes manipulation- display is enabled using image_tx_en signal.



9 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.

9.1 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.

9.2 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:

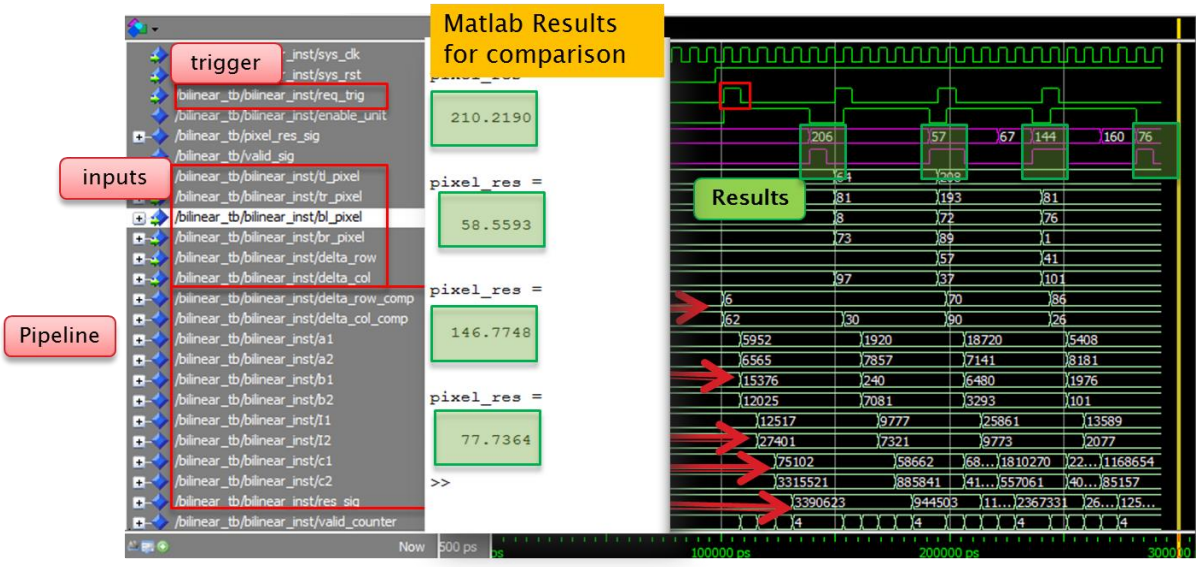


Figure 38- Bilinear results- compare with Matlab

9.3 Resolution changes

High resolution (512X512) simulations with signal waves (Modelsim) requires more than 3 hours of running. Hence, changes were made in the design in order to support low resolution images (96X128). These adjustment allows short simulation time and an effective way to debug the manipulation block and the main system as well.

9.4 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.

9.5 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.

9.6 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.

9.7 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.

9.8 Working with generics parameters

In order to change resolutions (as mentioned in 7.3), generic parameters were implemented in the design. This method allows easily change parameters at single file, without editing other files.

10 Summery

10.1 Problems during the process

10.1.1 Working with fractures

First version of addr_calc used a fixed point package. Problems occurred during synthesis with the package.

Solution - work with regular std_logic_Vector, with relevant adjustments.

Example – representing (sin(60))

$$\sin(60) = 0.866025$$

$$\lceil 0.866025 \times 128 \rceil = 111 = 001101111_{binary}$$

10.1.2 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.

10.1.3 Timing issues

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

	Requested Starting Clock	Estimated Frequency	Requested Frequency	Estimated Period	Clock Period	Clock Slack	Clock Type	Group
addr_calc|clk_133	100.0 MHz	53.3 MHz	10.000	18.754	-8.754	inferred	Inferred_clkgroup_0	

Figure 39- 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

Starting Clock	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Slack	Clock Type	Clock Group
addr_calc|system_clk	100.0 MHz	129.3 MHz	10.000	7.732	2.268	inferred	Inferred_clkgroup_0

Figure 40- Synthesis timing report (2)

10.2 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.

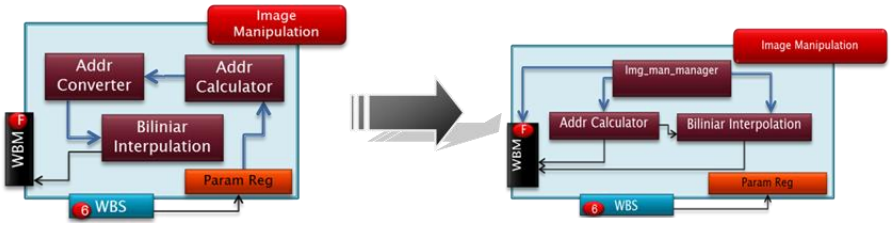


Figure 41- Arch modification

10.3 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.

10.4 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.

11 Future improvements

- Change the algorithm from bilinear interpolation to other interpolation method such as "nearest neighbor", "polynomial" etc.
- Replace UART transmission to a more modern and fast protocol, such as USB, Firewire etc.
- Replace VGA with a modern display port, such as DVI, HDMI etc.
- Work with optional external memory, to support larger images.

12 Appendix

The next section includes useful links to the project and other helpful information.

12.1 Main Image Manipulation System

http://image-rotation-technion-ee.googlecode.com/svn/

Includes all the VHD files, Matlab files, test images, Docs, presentations, Quartus files.

12.2 Digital Laboratory

http://diglab.technion.ac.il

12.3 Blocks description documents

http://image-rotation-technion-ee.googlecode.com

13 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