ECE 532 Digital Hardware: Final Project
Group Report
Photoshop Functionalities on FPGA

Group:
Pearl Liu (990975307)
George Ng (990857355)

Date: March 28, 2005

Table of Contents

System Design Overview:

Goals

The goal of the project was to implement a real-time video capture, Photoshop digital filter processing, and display system on FPGA. The implementation platform is the Virtex-II XC2V2000 FF896 Speed Grade –4 Development Board.

Original Design

The original design was to use a video capture core that would continuously capture video data from a digital camera to ZBT RAM. The captured image would then be processed by a Photoshop type of digital filter implemented in HDL code. The filtered image would then be displayed on a VGA monitor using a VGA display controller. The entire process would be performed in real-time.

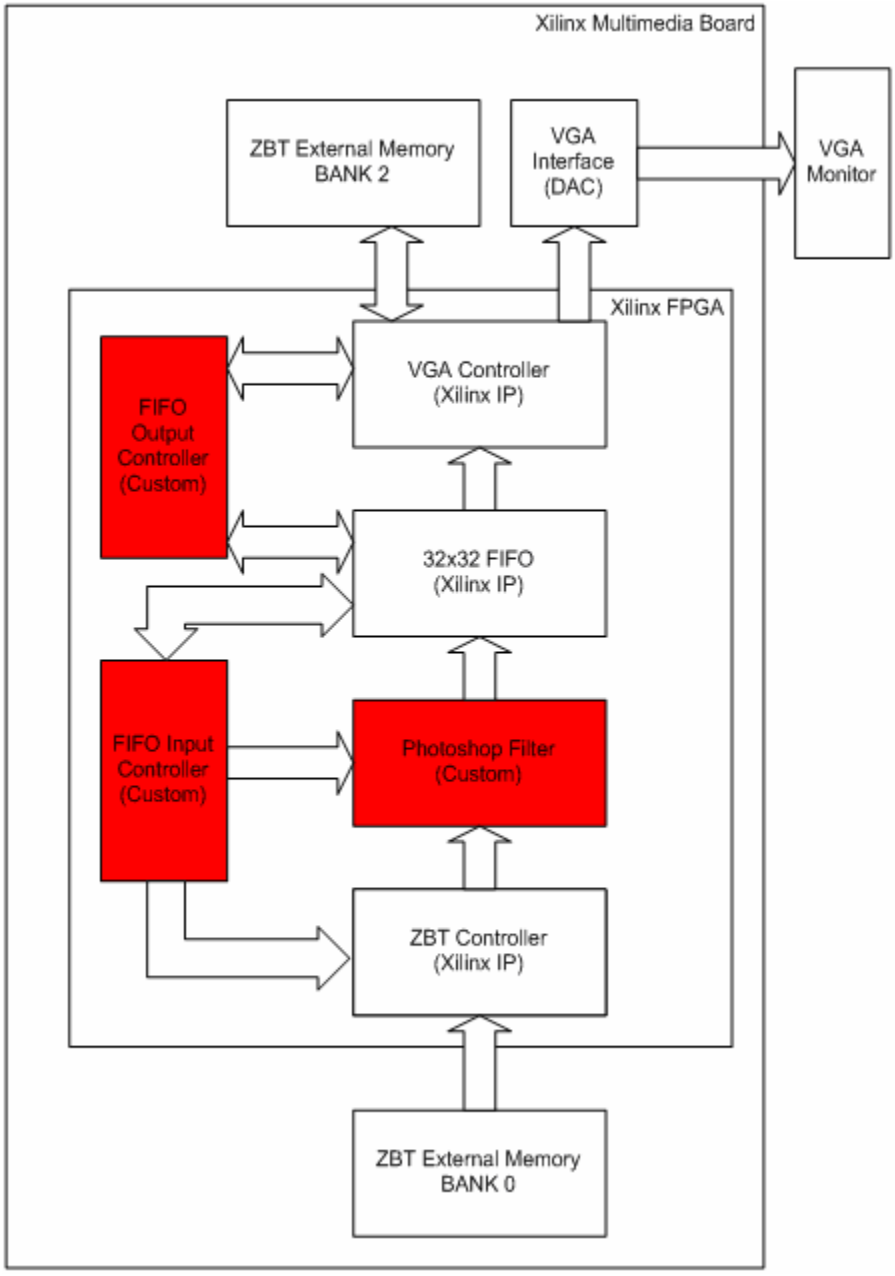
Final Design

All parts of the proposed project were implemented except for the real-time video capture component. This decision was taken to simplify our project. Instead of capturing the image in real-time, an image was loaded from a separate project into the ZBT RAM. A 32-bit (flipped row order) bitmap image file format was used. From our research we found that the pixel RGB (red, green, and blue) components in a bitmap are stored directly in consecutive bytes, where each byte represents the pixel color. To simplify the design of the memory controller, we decide to use a 32-bit bitmap file format over a 24-bit bitmap format (the 32-bit bitmap file format uses 24-bits to store the color data and pads the remaining 8-bits). This was done since the ZBT RAM reads out 32-bits at a time. In the case of 24-bit bitmaps, the RGB components of a pixel can potentially straddle a 32-bit boundary, as result; multiple reads may be needed performed to retrieve a single pixel, thus complicating the memory controller design.

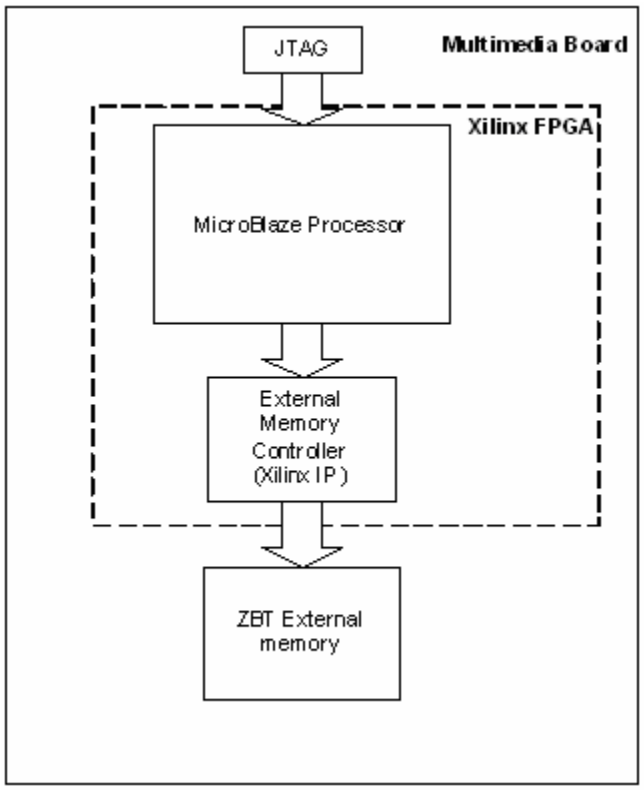
The way our design works is that once there is image data in the ZBT RAM, it will immediately send the data to the digital filter within the FPGA. The filtered data is then passed on to a 32 bit x 32 deep FIFO using a custom designed ZBT memory to FIFO controller. Once the FIFO becomes full, a FIFO to display controller is used to transfer the filtered data out of the FIFO and to the SVGA display controller. The SVGA controller provided by Xilinx handled the reads and writes from a ZBT video memory bank within the display block and the filtered image would be immediately displayed on the monitor.

Block diagram

Digital Photoshop Filter and Display Blocks



Bitmap image loader
Bitmap image loader is the direct implementation of lab m08. It is used to initially load image data into a ZBT RAM bank 0. No further description is provided.
Lab m08 can be found at:
http://www.eecg.toronto.edu/~pc/courses/edk/modules/6.3/m08.pdf



Description of IP Blocks:

IP Block	Functionality	Origin
Custom Blocks		
displaypattern.v	Top level hardware block which connects the FIFO, ZBT controllers, and custom controllers	Custom design by George and Pearl
mem2fifo_ctrl.v	Takes image data from ZBT RAM and inputs it through the filter and into the FIFO.	Custom Design by George

fifo2disp_ctrl.v	Takes image data from the FIFO and inputs it into display controller	Custom Design by George
gblur.v	Low pass filter code that blurs an image	Custom Design by Pearl
emboss.v	High pass filter code that embosses an image	Custom Design by Pearl
Xilinx Display Controller and ZBT RAM Controller		
ADDR_BUS_INTERFACE.v	Address bus interface for ZBT RAM controller	Xilinx SVGA IP
BM_MODE_SVGA_CTRL.v	SVGA controller	Xilinx SVGA IP modified by connecting the RAM and SVGA controller to the same pixel_clock
CTRL_BUS_INTERFACE.v	Control bus interface for the ZBT RAM controller	Xilinx SVGA IP
DATA_BUS_INTERFACE.v	Data bus interface for the ZBT RAM controller	Xilinx SVGA IP
DRIVE_DAC_DATA.v	Directs data to SVGA	Xilinx SVGA IP
MEMORY_CTRL.v	Memory controller module of ZBT RAM	Xilinx SVGA IP
PIPELINES.v	Provides pipelines for data WRITES and a latch for the data READS in ZBT	Xilinx SVGA IP
SVGA_TIMING_GENERATION.v	Generates timing and control signal for DAC & VGA output connector	Xilinx SVGA IP
ZBT_CONTROL.v	ZBT RAM controller top level interface	Xilinx SVGA IP
Xilinx CoreGen		
mem_fifo.v	CoreGen FIFO wrapper which is used as a buffer to store filtered data and send to ZBT RAM of display controller	Instantiated from CoreGen

Outcome

How Well It Works

Once a bitmap image was loaded into the ZBT RAM with the use of an external image loader, we were able to blur a bitmap image and display the resulting effect on the VGA monitor. We discovered that the VGA color output of the Xilinx board was not accurate. All colors suffered from a tinge of green, especially the color black. We suspect that this problem might be related to the DAC on the board. Although the primary colors of red, green and blue were displayed accurately, more complex 24-bit colors were not displayed accurately. To prove that we had not connected the color components incorrectly to the display controller, we created a bitmap with the reference RGB colors as well as more complex 24-bit colors on the same bitmap for display. When the bitmap was displayed on the monitor, the RGB colors displayed correctly while other more complicated colors were not.

Suggestion for Further Work or Improvement

Although a blur and an emboss Photoshop filter were created, they were both implemented in different projects, which meant the user was unable to select the filter of choice without reprogramming the FGPA. A C program was written to interface with the FSL to enable the user to select which Photoshop effect to be done on the bitmap image, but due to time constraint, the FSL was not integrated with the rest of the system. Thus, further work would be to integrate the FSL with the final design to allow filter selections.

Real-time image capture would enhance the "coolness" of the project. Due to time constraints in figuring out the display controller and ZBT RAM controllers, there was not sufficient time to investigate integration of a video capture core.

More advanced Photoshop filters could be implemented in hardware, potentially any Photoshop filter imaginable. Photoshop filters implemented in hardware would have a speed advantage over a software filter, especially on complex filter effects or large images. More advanced filters would include radial blur, twirl, and ocean ripple effect filters. These filters would probably involve the FFT core or large matrix multiplications.

Decompression algorithms for different image formats could be investigated to allow various types of pictures to be downloaded to the FPGA and processed. This would save time in the loading process of the image since the compressed image would be smaller than the uncompressed bitmap format we are currently using. This would allow jpeg, gif image file format to be filtered.

Description of the Blocks:

Custom Designed Blocks

displaypattern.v

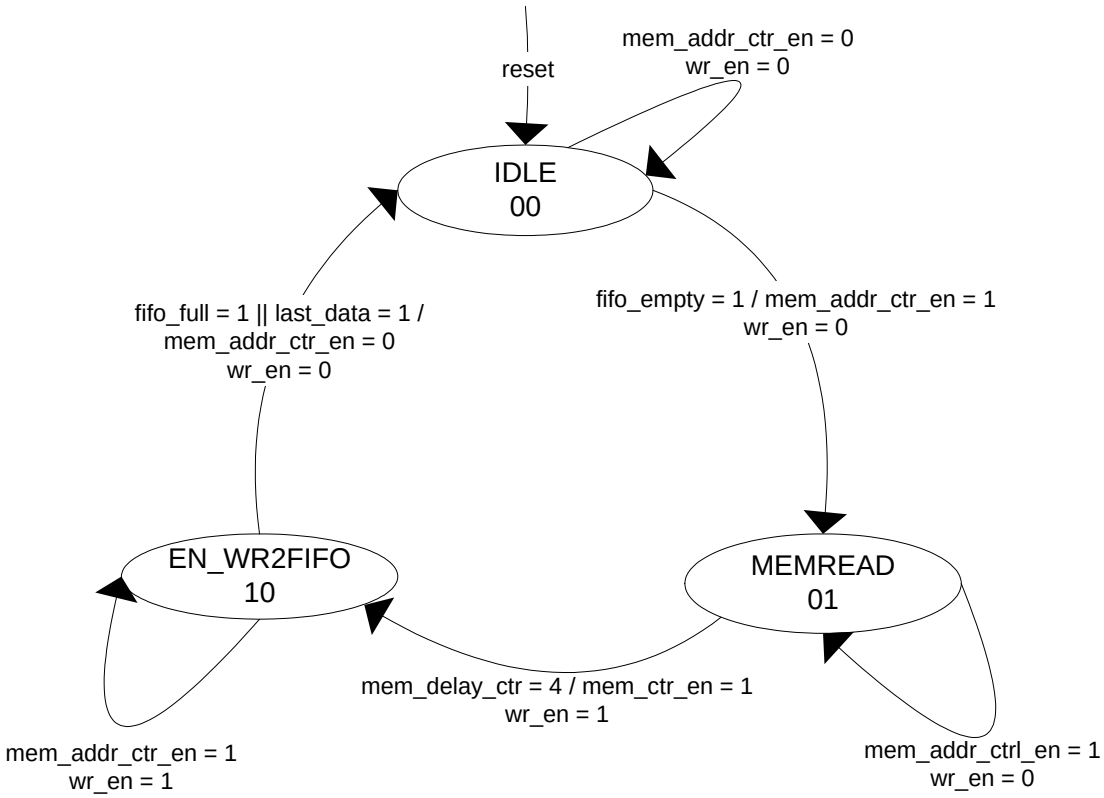
The displaypattern module is the top-level hardware block. This is the block that is downloaded into the FPGA. It instantiates the ZBT RAM bank 0 controller, the SVGA video display controller on bank 2, memory to FIFO controller, FIFO to display controller, the digital Photoshop filters, and the CoreGen FIFO. This block ties the signals of the ZBT controller for reads from the ZBT RAM. This block also drives the correct signals to the display controller to allow writing to the video RAM. This module has a video RAM address counter used to draw the pixels from the FIFO into the video RAM. The video RAM address counter stops when it reaches the corner of the screen at 640x480 = 0x4B000.

mem2fifo_ctrl.v

The mem2fifo_ctrl.v controls the transfer of data from the ZBT RAM, through the filter and into the FIFO. The controller is a Mealy type FSM. The behaviour was tested in ModelSim by using a testbench called mem_and_fifo.v.

Module Functional Description:
The controller is designed to burst data from the RAM into the digital filter, until the 32 bit x 32 deep FIFO is filled. The data read from the RAM is 32 bits wide. The FIFO write enable is raised after the initial four clock latency from the ZBT RAM to capture the incoming data. When the FIFO raises the full flag, the mem2fifo_ctrl stops the reads from the ZBT RAM and readjusts the ZBT RAM address counter. This is done since there is there is a five clock latency between the when the full flag is raised and when the RAM address counter knows to stop counting. The FSM continues this cycle until the last bitmap pixel is read out from the ZBT RAM, where upon it raises the last data flag to inform the fifo2disp_ctrl block to read out from the FIFO before it is full.

FSM State Diagram:

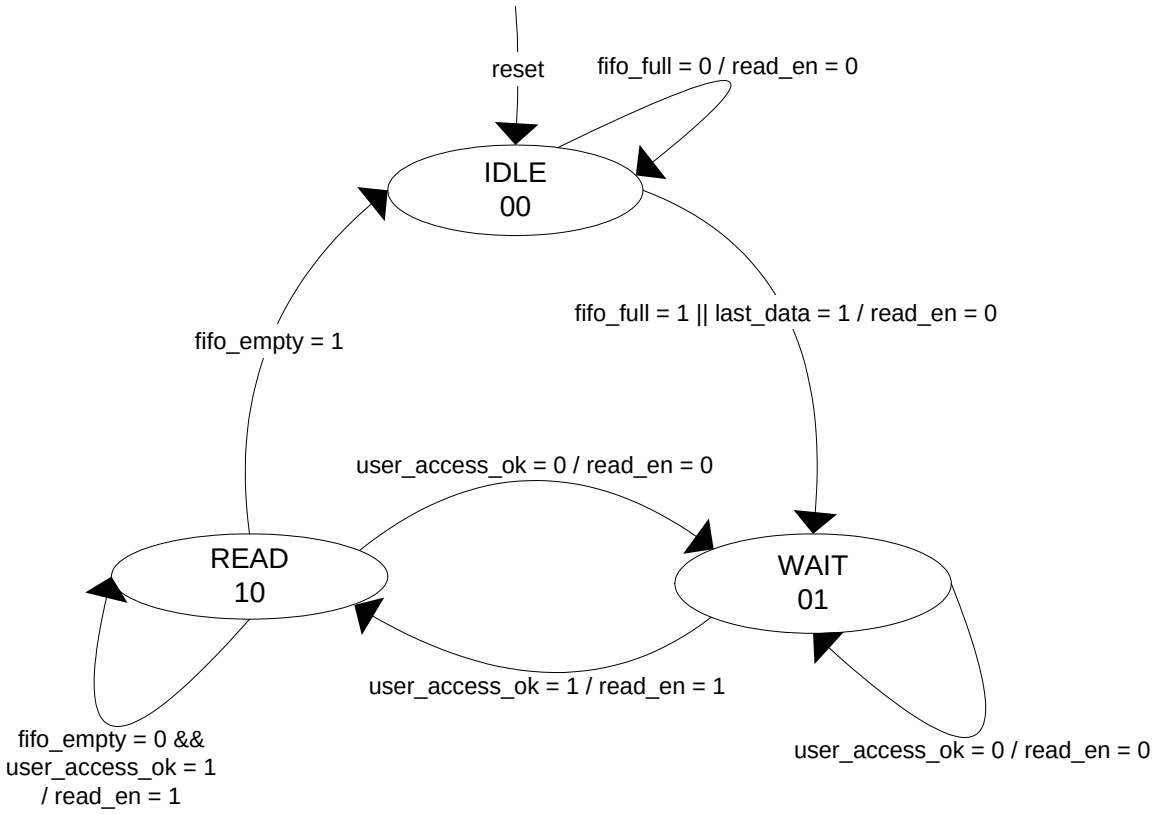


fifo2disp_ctrl.v

The fifo2disp_ctrl controls the transfer of data stored in the FIFO to the display controller. The controller is a Mealy type FSM. The behaviour was tested in ModelSim using a testbench called mem_and_fifo.v.

Module Functional Description:
The controller is designed to read data from the FIFO and transfer it to the display controller until the FIFO is empty. The FSM is triggered by either the full signal from the FIFO or last_data signals from the mem2fifo_ctrl block. Data is transferred out of the FIFO whenever the user_access_ok signal is high. A data_valid signal, which is a flopped version of the read_en is also generated, since it takes one clock cycle for the FIFO to output valid data after the read_en is raised.

FSM State Diagram:



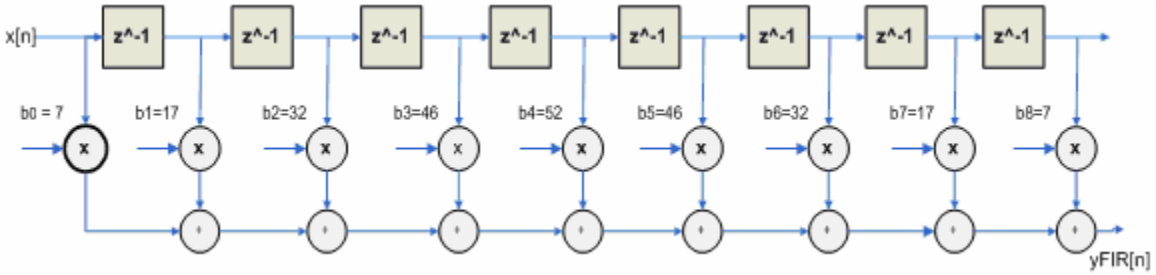
gblur.v

This is one of the digital Photoshop filters created. The gblur module is designed to provide a blurring effect on the bitmap image. The behaviour of the module was tested in ModelSim using a testbench called gblur_tb.v.

Module Functional Description:

The module coded represents an 8th order Gaussian Finite Impulse Response low pass filter to give the blur effect. The design is fully synchronous, with active-high synchronous reset. The coefficients of the filter are implemented as unsigned 8-bit words (unsigned integers) that are even-symmetric to guarantee that the phase characteristic will be linear. The coefficients were chosen to give the filter a Gaussian shape for its impulse response. The module processes 8 bits at a time – this would represent red, green, or blue components of the 24 bit pixel. Each bit is weighted by the coefficients that represent a Gaussian curve and summed to produce a filtered pixel. The functional block diagram for the blur filter is shown below. Three instances of this module are instantiated to process the RGB components of a single pixel. This block looks at the FIFO write enable signal to determine if it should process new data on the clock edge or hold the previous values in the registers until new data arrives to be processed.

Functional Block Diagram of Blur Filter:



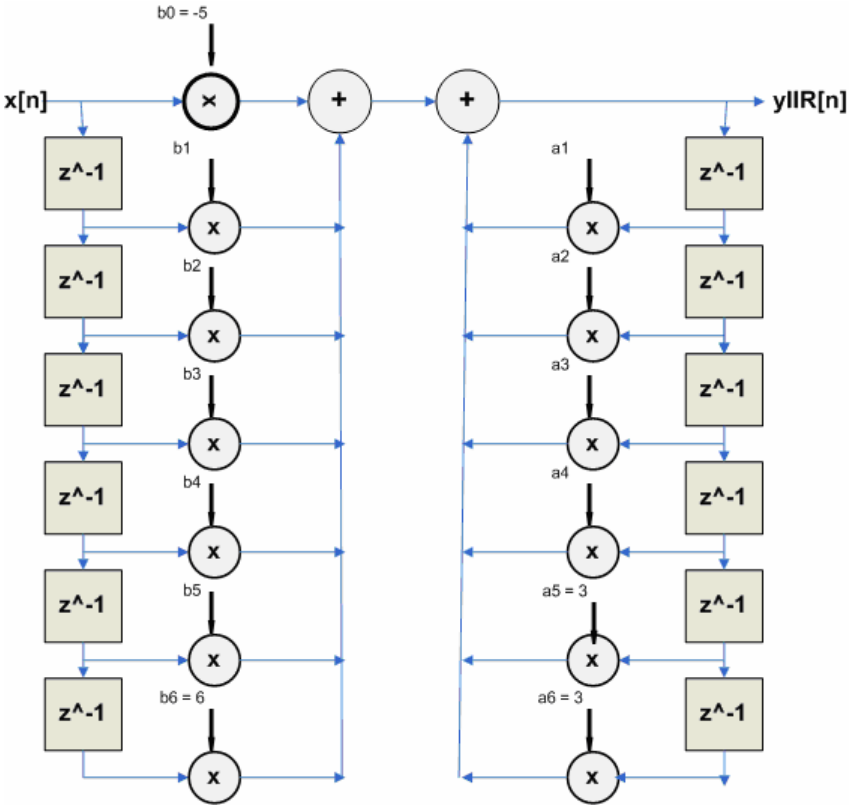
emboss.v

This is one of the digital Photoshop filter created. The emboss module is designed to provide an emboss effect on the bitmap image. The behaviour of the model was tested in ModelSim using a testbench called emboss_tb.v.

The module represents a 6th order high pass filter to emboss the picture. Similar to the blur filter, the emboss filter design is also fully synchronous with active-high synchronous reset. The module also processes 8 bits at a time, similar to the structure of the gblur module. However, a high pass filter has non-zero coefficients in the numerator and denominator in its transfer function and is not coded as a FIR filter because a FIR filter does not have feedback. The emboss module is coded as an Infinite Impulse Response filter due to the need to account for the coefficients in the numerator. The functional block diagram for the emboss filter is shown below. Similar to the gblur module, three instances of this module are instantiated to process the RGB components of a single pixel. This block looks at the FIFO write enable signal to determine if it should process new data on the clock edge or hold the previous values in the registers until new data arrives to be processed.

Functional Block Diagram of Emboss Filter:

Note: For coefficients that do not have numerical value implies the coefficient is 0. The variables are shown as a general representation of an IIR filter.



CoreGen Blocks

mem_fifo.v

The FIFO wrapper generated by CoreGen used to instantiate the 32 bit x 32 deep FIFO. A FIFO is required between the digital filter and the display controller to buffer the data since our filter can process data on every clock edge, while the write access to the video RAM through user_access_ok may not be available on every clock edge.

Xilinx SVGA Controller and ZBT RAM Controller Blocks

BM_MODE_SVGA_CTRL.v

The BM_MODE_SVGA_CTRL module comes from the BM_MODE_SVGA_CTRL.v file of the Bit Mapped Mode SVGA example design from the Xilinx website. The modifications involved removing the part related to the IO BUFG since we did not use this in our design. The user memory clock and user memory ram clock was modified to connect to the system clock of 27 MHz such that the ZBT memory was synchronized with the rest of the design. This module was tested using Modelsim by inputting the clock and seeing if the correct time information came out.

ADDR_BUS_INTERFACE.v

The ADDR_BUS_INTERFACE module comes from the Bit Mapped Mode SVGA example design from the Xilinx website. No modifications were made to this module. This module was tested by running it in the FPGA, as part of the test to see if memory contents could be displayed on a monitor. This module is also the same as the module provided in the Xilinx ZBT RAM controller package.

CTRL_BUS_INTERFACE.v

The CTRL_BUS_INTERFACE module comes from the CTRL_BUS_INTERFACE.v file of the Bit Mapped Mode SVGA example design from the Xilinx website. No modifications were needed. This module was tested by running it in the FPGA, as part of the test to see if memory contents could be displayed on a monitor. This module is also the same as the module provided in the Xilinx ZBT RAM controller package.

DATA_BUS_INTERFACE.v

The DATA_BUS_INTERFACE module comes from the DATA_BUS_INTERFACE.v file of the Bit Mapped Mode SVGA example design from the Xilinx website. No modification was made to this module. This module was tested by running it in the FPGA, as part of the test to see if memory contents could be displayed on a monitor. This module is also the same as the module provided in the Xilinx ZBT RAM controller package.

DRIVE_DAC_DATA.v

The DRIVE_DAC_DATA module comes from the DRIVE_DAC_DATA.v file of the Bit Mapped Mode SVGA example design from the Xilinx website. The modifications involved adding a clk and reset signal, and adding code to display colour bars for testing purposes. This module was tested using ModelSim, in conjunction with the BM_MODE_SVGA_CTRL module.

MEMORY_CTRL.v

The MEMORY_CTRL module comes from the MEMORY_CTRL.v file of the Bit Mapped Mode SVGA example design from the Xilinx website. This module was tested by running it in the FPGA, as part of the test to see if memory contents could be displayed on a monitor, and since it worked the first time no additional testing was needed.

PIPELINES.v

The PIPELINES module comes from the PIPELINES.v file of the Bit Mapped Mode SVGA example design from the Xilinx website. This module was tested by running it

in the FPGA, as part of the test to see if memory contents could be displayed on a monitor. This module is also the same as the module provided in the Xilinx ZBT RAM controller package.

SVGA_TIMING_GENERATION.v

The SVGA_TIMING_GENERATION module comes from the SVGA_TIMING_GENERATION.v file of the Bit Mapped Mode SVGA example design from the Xilinx website. This module was tested using ModelSim, in conjunction with the BM_MODE_SVGA_CTRL module.

ZBT_CONTROL.v

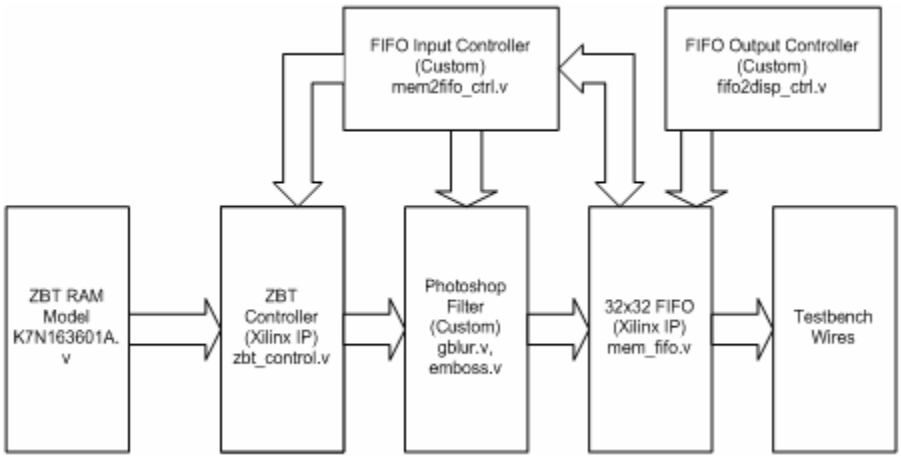
The ZBT_CONTROL module originated from the Bit Mapped Mode SVGA example design. This module is also the same as the module provided in the Xilinx ZBT RAM controller example. This module is also the same as the module provided in the Xilinx ZBT RAM controller package.

Testbenches and Models

mem_and_fifo_tb.v

Testbench used to test interaction of mem2fifo_ctrl, fifo2disp_ctrl, Xilinx ZBT controller, and CoreGen mem_fifo. This module instantiates mem2fifo_ctrl, fifo2disp_ctrl, mem_fifo, zbt_control, and K7N163601A. The testbench simulates the transfer of data from the ZBT RAM to the FIFO using mem2fifo_ctrl, and the transfer of data out from the FIFO using fifo2disp_ctrl. To run this testbench execute the mem_and_fifo_tb.do in the mem_and_fifo directory in ModelSim.

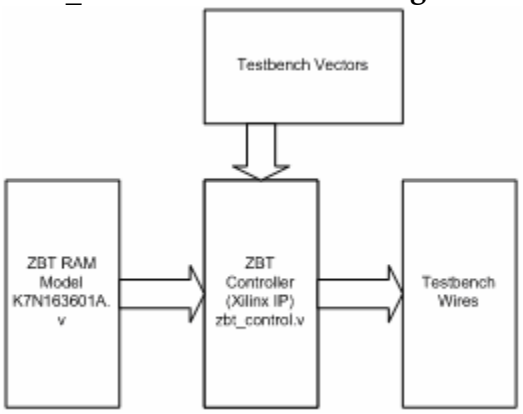
mem_and_fifo Testbench Block Diagram



ZBT_tb.v

The functionality of the RAM controller was tested for read and write transactions. This connects ZBT_CONTROL to the RAM model K7N163601A.v.

ZBT_tb Testbench Block Diagram



K7N163601A.v

Behavioral simulation RAM model found in module m08.zip, originally called MMboard_ZBT_behmod.v.

Description of the Design Tree

A file called PhotoshopFunctions.zip is submitted along with this report. Within the file, there are eight directories: doc, emboss, gblur, imgloader, mem_and_fifo, ViewSVGAbmp_emboss, ViewSVGAbmp_gblur, ViewSVGAbmp_orig. The tables below highlight the important contents within each directory.

doc Purpose: contains presentation slides	
File	**Purpose**
PhotoshopFunctions.pdf	Final group report
Presentation.pdf	Slides used for presentation during project demo

emboss Purpose: to testbench emboss filter Note: The same matlab files contained in gblur directory were used to test the emboss filter	
File	**Purpose**

emboss.v	Emboss filter HDL code
emboss_tb.v	Emboss filter HDL testbench code
orange.jpg	Original picture of an orange
orange_em.jpg	Embossed picture of an orange

gblur Purpose: to testbench blur filter	
File	**Purpose**
gblur.v	Blur filter HDL code
gblur_tb.v	Blur filter HDL testbench code
gblur.do	Do script to test filter
image.in	Input image text data for gblur_tb.v
image.out	Output image text data from gblur_tb.v
convimg2hex.m	Matlab script to convert image to hexadecimal value for testing
convhex2img.m	Matlab script to convert output text data to an image file for display
orange.jpg	Original picture of an orange
orange_blur.jpg	Blurred picture of an orange

imgloader Purpose: to download a bitmap picture to the ZBT memory	
File	**Purpose**
System.xmp	Project file setup to download bitmap file
System.mhs	Hardware specifications of imgloader – contains ZBT memory, EMC, microblaze
Zbt.ucf	Constraint file for ZBT memory and clock pins assignment
Subdirectories	
__xps	Directory created by XPS
clk_align_v1_00_a	Clock alignment files provided by module08
data	Contains system.ucf generated by BSB
etc	Generated by BSB
microblaze_0	microblaze directory generated by BSB
pcores	Provided by module08
zbtio	ZBT memory directory generated by Base System Builder – contains pao, mpd, HDL code for memory
mem_and_fifo Purpose: to testbench FIFO controllers with ZBT memory and FIFO	
File	**Purpose**

mem_and_fifo.npl	ISE Project file
mem_and_fifo_tb.v	Testbench file to test ZBT ram, mem2fifo_ctrl.v, fifo2disp_ctrl.v, and CoreGen 32x32 deep FIFO
mem2fifo_ctrl.v	Memory to FIFO controller module, instantiated by mem_and_fifo_tb.v
fifo2disp_ctrl.v	FIFO to display controller module, instantiated by mem_and_fifo_tb.v
mem_fifo.v	Xilinx Module for memory and FIFO connection
K7N163601A.v	Xilinx ZBT ram model, instantiated by mem_and_fifo_tb.v
ZBT_CONTROL.v	Xilinx ZBT controller, instantiated by mem_and_fifo_tb.v
Subdirectories	
__xps	Directory created by XPS
clk_align_v1_00_a	Clock alignment files provided by module08
data	Contains system.ucf generated by BSB
etc	Generated by BSB
microblaze_0	microblaze directory generated by BSB
pcores	Provided by module08
zbtio	ZBT memory directory generated by Base System Builder – contains pao, mpd, HDL code for memory

ViewSVGAbmp_orig Purpose: to display the original bitmap picture	
File	**Purpose**
view_svga.npl	ISE Project file
view_svga.ucf	Assigns system clock speed and pins for display controller (VGA and ZBT memory banks)
displaypattern.v	Top level design of the project
Rest of directory contents are described in Description of IP Block	All files need for the entire system integration
Subdirectories	
__projnav	Generated by ISE as part of project tree
doc	Generated by ISE
work	Generated by ISE for testbenching purposes

ViewSVGAbmp_gblur Purpose: to display the blurred bitmap picture	
File	**Purpose**
view_svga.npl	ISE Project file
view_svga.ucf	Assigns system clock speed and pins for display controller (VGA and ZBT memory banks)
displaypattern.v	Top level design of the project
Rest of directory contents are described in Description of IP Block	All files need for the entire system integration
Subdirectories	
__projnav	Generated by ISE as part of project tree
doc	Generated by ISE
work	Generated by ISE for testbenching purposes

ViewSVGAbmp_emboss Purpose: to display the embossed bitmap picture	
File	**Purpose**
view_svga.npl	ISE Project file
view_svga.ucf	Assigns system clock speed and pins for display controller (VGA and ZBT memory banks)
displaypattern.v	Top level design of the project
Rest of directory contents are described in Description of IP Block	All files need for the entire system integration
Subdirectories	
__projnav	Generated by ISE as part of project tree
doc	Generated by ISE
Work	Generated by ISE for testbenching purposes

ZBT_CONTROL_tb Purpose: performs simple read and write transactions using the Xilinx RAM controller	
File	**Purpose**
ZBT_tb.v	Top level testbench
ZBT_tb.do	ModelSim do file
img_data.mem	RAM initialization data

Instructions to run the system

To view the original bitmap image on the SVGA monitor.

1. Create a 32-bit “flip row order” bitmap in a program like Adobe Photoshop. Some sample bitmaps are included in the imgloader directory.
2. Program the FPGA using the imgloader project.
3. Open XMD and download a bitmap image to the ZBT RAM. i.e. dow –data intel.bmp 0x80600000. This will transfer the image over JTAG into the ZBT RAM.
4. Close XMD and XPS and open the ISE project ViewSVGAbmp_orig and generate the programming file for displaypattern.v
5. Open Impact and download displaypattern.bit to the FPGA
6. The original bitmap should appear on the screen. If it is distorted flip the reset switch on the multimedia board.

Note:
To view the Gaussian blur filter follow procedure 4 – 5 using the ViewSVGAbmp_gblur project.
To view the emboss filter follow procedure 4 – 5 using the ViewSVGAbmp_emboss project.

Instructions to run the Matlab and Verilog Testbench

To test the Verilog digital filters in Matlab.

1. Create a 48x48 size grayscale jpeg image. i.e. mypic.jpg
2. In Matlab:
   ```
   M = imread('mypic.jpg');
   convimg2hex(M);
   ```
   To create the input data file for the Verilog testbench
3. In ModelSim:
   ```
   do gblur.do
   ```
   This will run the filter on the image and create the an output data file for Matlab
4. In Matlab:
   ```
   A = convtxt2img(48,48);
   colormap('gray');
   imagesc(A);
   ```
   To reconstruct the filtered data from the digital filter

References

Michael D. Ciletti, Advanced Digital Design with Verilog HDL, 1st ed. , New Jersey: Prentice Hall, 2002.

Alan V. Oppenheim and R.W. Schafer with John Buck, Discrete Time Signal Processing, 2nd Edition, Prentice Hall Inc, 1998.

Create Bitmap Images Using a Text Editor
http://www.developeriq.com/articles/view_article.php?id=137

Bit Mapped Mode SVGA example source file
http://www.xilinx.com/products/boards/multimedia/docs/examples/BM_MODE_SVGA.zip.

Multimedia Board user guide
http://www.eecg.toronto.edu/~pc/courses/432/2004/handouts/Multimedia_UserGuide.pdf

Multimedia Board datasheet
http://www.eecg.toronto.edu/~pc/courses/432/2004/handouts/Multimedia_Schematics.pdf

ZBT RAM memory controller
http://www.xilinx.com/products/boards/multimedia/docs/examples/ZBT.zip

ZBT RAM behavioral simulation model
http://www.eecg.toronto.edu/~pc/courses/edk/modules/6.3/m08.zip

ZBT RAM data sheet
http://www.eecg.toronto.edu/~pc/courses/edk/doc/ZBT_k7n163601a.pdf
(Functional description of page 8 particularly useful)

Xilinx synchronous FIFO data sheet
http://www.xilinx.com/ipcenter/catalog/logicore/docs/sync_fifo.pdf

Connecting Customized IP to the MicroBlaze Soft Processor Using the Fast Simplex Link (FSL) Channel
http://www.xilinx.com/bvdocs/appnotes/xapp529.pdf