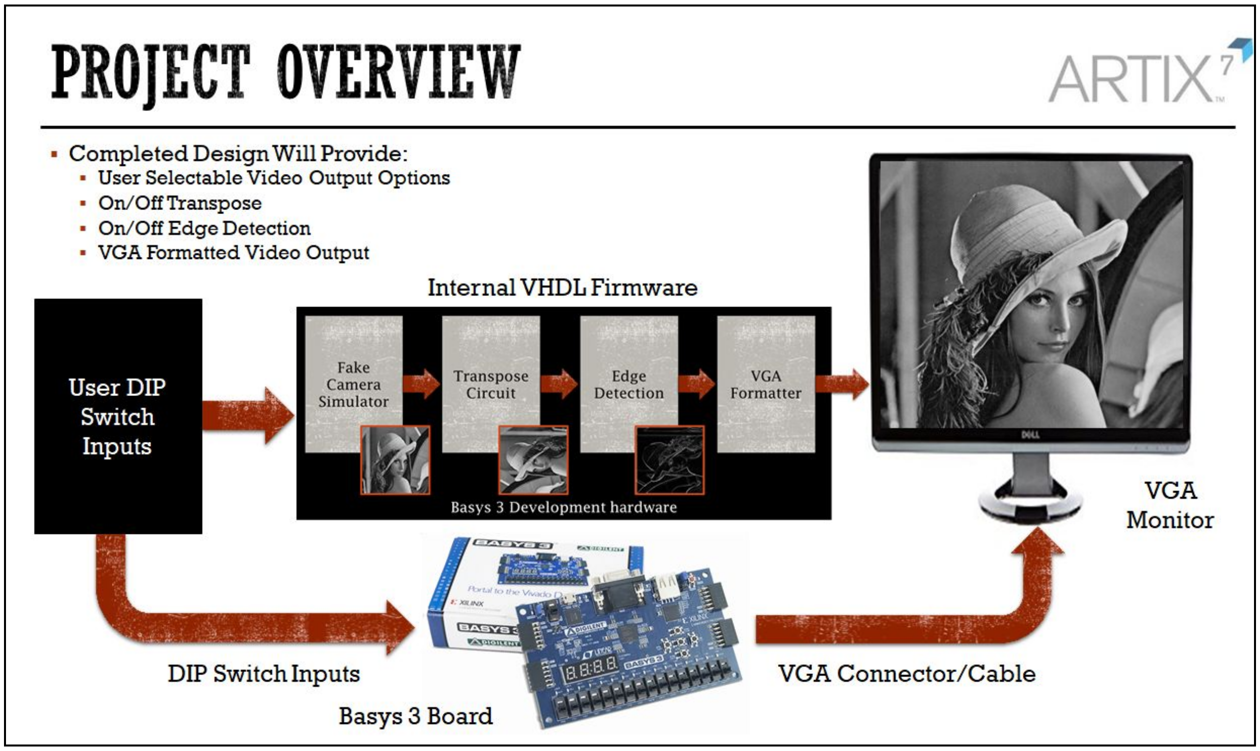
VGA Transpose and Edge Detection

Team 7 Distance Learning

Bryce Williams	- GTID: 903097901
Zachary Boe	- GTID: 903124261
Gregory Walls	- GTID: 903289298
Benjamin Sullins	- GTID: 903232988

INTRODUCTION

This project generates a synthetic video output with available image transposition, edge detection, and VGA output. The design is implemented on a Basys3 Development Board with user DIP switch inputs controlling each sub-module and a VGA video output for display purposes.



PROJECT OVERVIEW

- Completed Design Will Provide:
  - User Selectable Video Output Options
  - On/Off Transpose
  - On/Off Edge Detection
  - VGA Formatted Video Output

Figure 1: Project Overview

The sub-modules for the design include Fake Camera Generation, Transpose Circuit, Edge Detection, VGA Converter, and (additionally) Color Mapping. The first six switches on the Basys3 board are used in this design to toggle different settings in the first three blocks. The end result is a large number of combinations for operation.

	Fake Camera			**Transpose Circ.**		**Edge Detection**				**Color Mapping**	
Switch [5:0]	xxxx00	xxxx01	xxxx1x	xxx0xx	xxx1xx	x00xxx	x01xxx	x10xxx	x11xxx	0xxxxx	1xxxxx
Notes	00 - Horizontal Pattern 01 - Vertical Pattern 1x - Pre-Loaded Image			0 - Disabled 1 - Enabled		00 - Disabled 01 - Enabled - Sobel Sum 10 - Enabled - Sobel Vertical 11 - Enabled - Sobel Horizontal				0 - Grayscale 1 - Pink	

Table 1: DIP Switch Options

ARCHITECTURE

Fake Camera Simulator:

The Fake Camera Simulator is responsible for generating the synthetic video data used throughout the design. There are three available options: Horizontal Test Pattern, Vertical Test Pattern, and a Pre-Loaded Image. Each of the options can be selected based on the Basys3 DIP Switch inputs shown in Table 1. Figure 2 shows example output options.

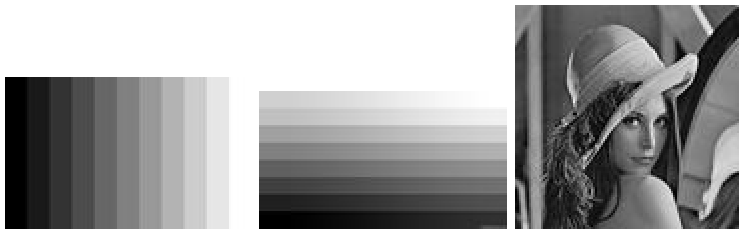


Figure 2: Fake Camera Simulator Outputs

An MMCM is fed to the 100 MHz Basys3 onboard clock to generate a 50 MHz pixel clock. This pixel clock was chosen as it falls within the CameraLink specs and also represents an actual clock rate for a typical camera (sensor). A 160x120 image size with 16 invalid column pixels was selected to limit the BRAM utilization throughout the design while retaining coherence to a real-life sensor output. A 50 Hz frame rate was also selected to retain similarity to actual cameras.

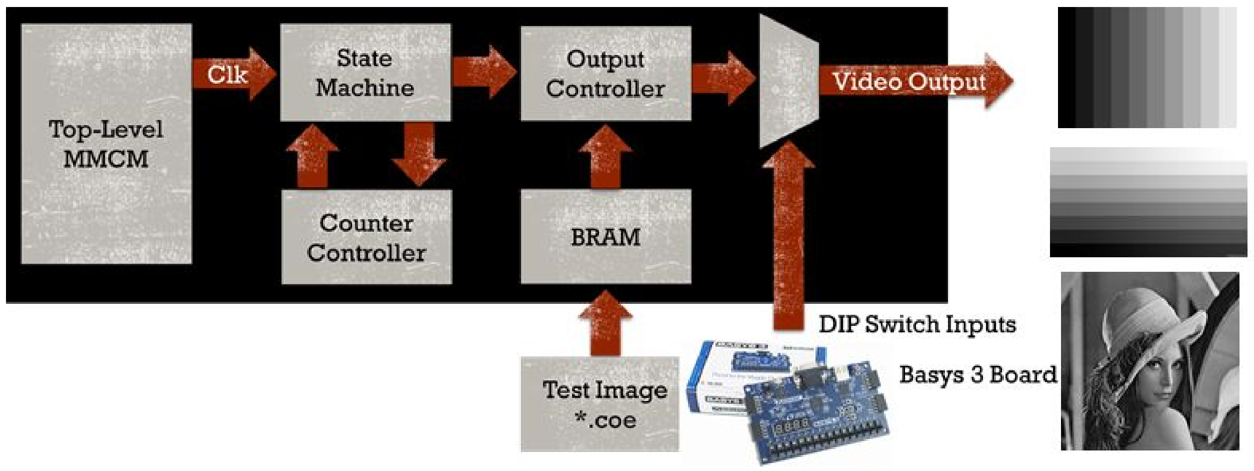
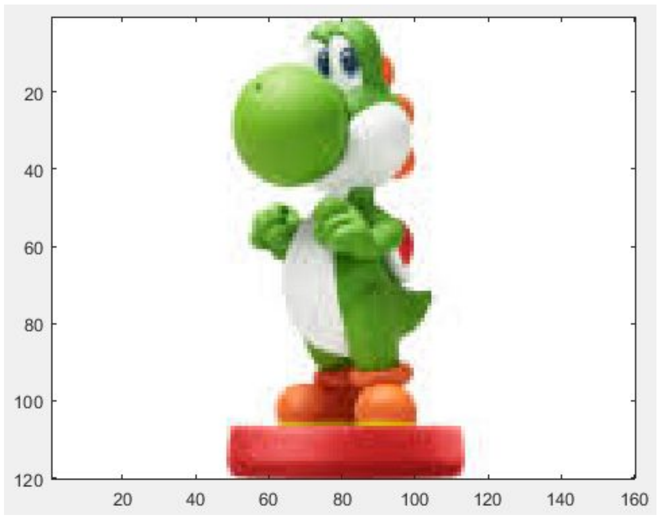
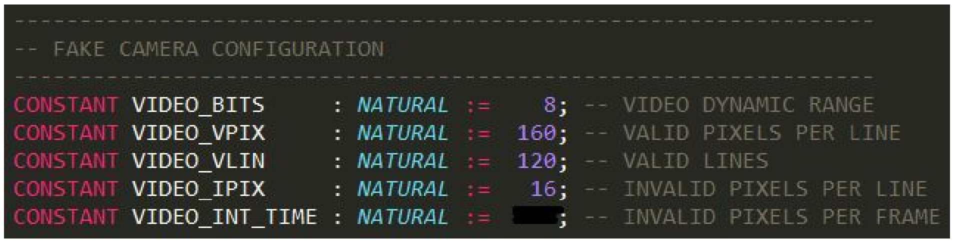


Figure 3: Fake Camera Simulator Overview

A high level overview of the Fake Camera Simulator is shown in Figure 3 above. The State Machine and Counter Controller handle the frame generation based on generics applied at the top level of the design. These generics represent the image size characteristics and flow down to these two modules for signal sizing and array generation.





```
----------------------------------------------------------------
-- FAKE CAMERA CONFIGURATION
----------------------------------------------------------------
CONSTANT VIDEO_BITS     : NATURAL :=    8; -- VIDEO DYNAMIC RANGE
CONSTANT VIDEO_VPIX     : NATURAL :=  160; -- VALID PIXELS PER LINE
CONSTANT VIDEO_VLIN     : NATURAL :=  120; -- VALID LINES
CONSTANT VIDEO_IPIX     : NATURAL :=   16; -- INVALID PIXELS PER LINE
CONSTANT VIDEO_INT_TIME : NATURAL := [redacted]; -- INVALID PIXELS PER FRAME
```

Figure 4: Image Sizing Generics

The Output Controller handles the strobe signals (FVAL/LVAL) and data outputs for the Fake Camera. The Output Controller ensures the proper alignment and data output which is needed for correlating to the exact location within the synthetic image frame. The Pre-Loaded Image requires address to be fed to an internal BRAM for proper readout and alignment with the other Test Patterns. This BRAM stores a *.coe file which was generated from a Matlab script. The *.coe file simply contains a preloaded image which is compatible for Xilinx's BRAMs. Finally, the user selectable DIP switches handle the video output mux.

Transpose Circuit:

The transpose circuit is responsible for receiving data and synchronization signals from the camera simulator and outputting the data as either the original image or the transpose of the original image. This requires outputting both data and synchronization signals in the proper format. Additionally, the transpose circuit has an asynchronous active high reset switch. When the reset switch is set to high, all operations cease, all outputs and valid signals are set to 0, and any inputs are not stored in memory. Normal operations continue after the reset signal is set to low.

The first stage of the transpose circuit buffers the input received from the camera simulator into Block RAM (BRAM). The contents of the BRAM are continuously overwritten with the input received from the camera simulator. This stage of the circuit reads *lval* and *fval* signals from the camera simulator and only stores data into memory when both signals are high. The buffer stage has a built-in valid signal that switches from low to high after the first full frame has been buffered. This valid signal is only set back to low when the reset switch is active. The valid signal is necessary to ensure that the transpose image is not output before valid pixels are stored in BRAM. This will become apparent when the transpose output structure is discussed.

The second stage of the transpose circuit reads a select signal. When the select signal is low, the circuit outputs the originally received image. When the select signal is high, the circuit outputs the transpose of the originally received image. This stage of the circuit outputs *lval* and *fval* signals that indicate "end of line" and "end of frame," respectively. The *lval* signal is set to low at the end of each line and the *fval* signal is set to low at the end of each frame. The edge detection circuit will use these signals to determine when valid pixels are available. Two counters are used to track which line of the frame is being output as well as which pixel of that line is being output.

The complexity of the circuit arises from outputting the transpose image. Figure 5 demonstrates the difference between the structure of the original image and the structure of the transpose image.

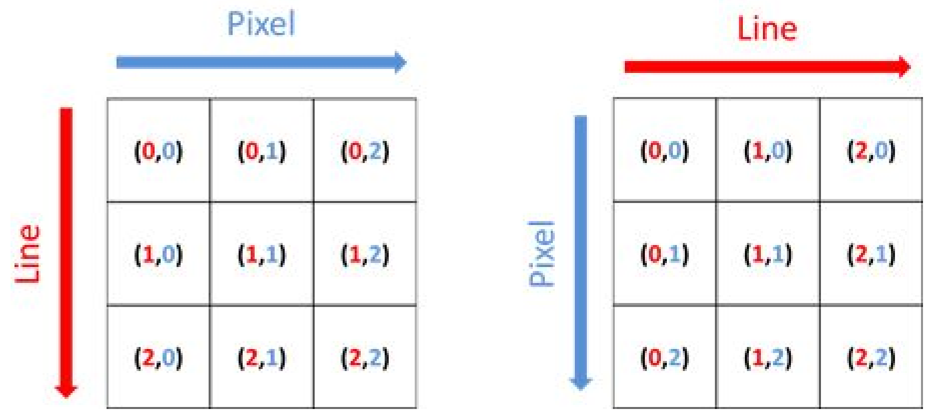
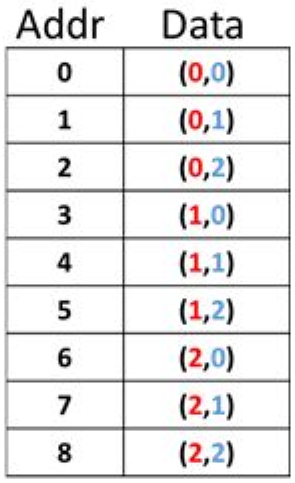


Figure 5: Original Image (left) and Transpose Image (right) Structures

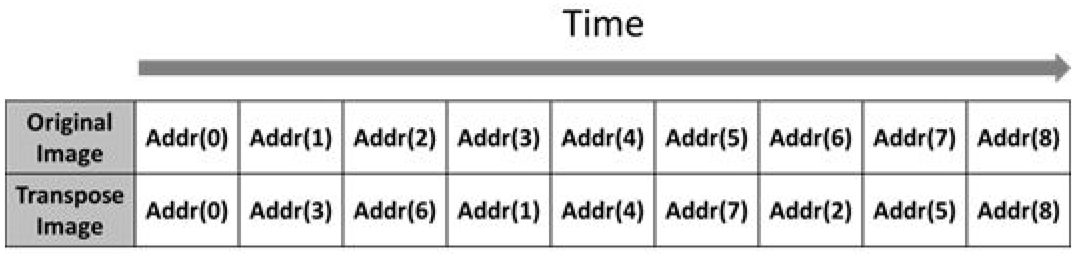
The input image is stored in BRAM as shown in Figure 6 whether the original or transpose image is output.



Addr	Data
0	(0,0)
1	(0,1)
2	(0,2)
3	(1,0)
4	(1,1)
5	(1,2)
6	(2,0)
7	(2,1)
8	(2,2)

Figure 6: BRAM Structure

When outputting the original image, the BRAM address can simply be incremented every time a pixel is output. When outputting the transpose image, the BRAM address must be calculated from the line and pixel counter values. The sequence of BRAM address values for the original and transpose images is visualized in Figure 7.



Original Image	Addr(0)	Addr(1)	Addr(2)	Addr(3)	Addr(4)	Addr(5)	Addr(6)	Addr(7)	Addr(8)
Transpose Image	Addr(0)	Addr(3)	Addr(6)	Addr(1)	Addr(4)	Addr(7)	Addr(2)	Addr(5)	Addr(8)

Figure 7: BRAM Address Timing

Equation 1 was derived to calculate the BRAM address value for the transpose image.

$$Addr = Pixel \times FrameWidth + Line \qquad (1)$$

In Equation 1, *Line* is the current value of the line counter, *Pixel is* the current value of the pixel counter, and *FrameWidth* is the horizontal length of the frame.

The need for the buffer valid signal is now apparent. If a full frame is not buffered before outputting the transpose image, then then the pixels stored in Addr(3), Addr(6), Addr(4), Addr(7), Addr(5), and Addr(8) would not be available in time.

Sobel Architecture:

Detection of edges in images is commonly accomplished using two-dimensional convolution filters. Many filters exist to accomplish this feat, with the Sobel Operator being one of the most common.

The Sobel Operator is a Two-Dimensional Convolution Kernel that calculates the differential gradient across a single pixel from one region to another. Being that edges in images arise from sharp changes in contrast of different objects, the Sobel is able to determine where these edges appear. Regions showing little to no change in scale appear very dark at the output, while large changes, indicating an edge, appear as light grey to white. A Sobel Filter is comprised of two of these two-dimensional filters, one to detect edges in the horizontal direction and the other to detect edges in the vertical direction. The final result is determined from the vector magnitude or summed approximation of the two outputs. The convolution is given for the Vertical and Horizontal Gradient Kernels below, as well as the Gradient Magnitude:

$$\boldsymbol{G_x} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * \boldsymbol{A} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * ([1 \quad 0 \quad -1] * \boldsymbol{A})$$

$$\boldsymbol{G_y} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \boldsymbol{A} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} * ([1 \quad 2 \quad 1] * \boldsymbol{A})$$

$$\boldsymbol{G} = \sqrt{G_x^{\,2} + G_y^{\,2}}$$

As can be seen in the equations above, the matrices can be implemented as separable matrices reducing the number of computations, leading to a reduction in allocation during implementation.

The architecture of the Sobel Filter is shown in the Figure 8 below. It is comprised of two 3x3 filter kernels, vertical and horizontal, and two BRAM line buffers each of depth equal to the image width (z^{-c}). The absolute value and shift blocks handle rounding and saturation and final truncation due to bit growth through the filter.

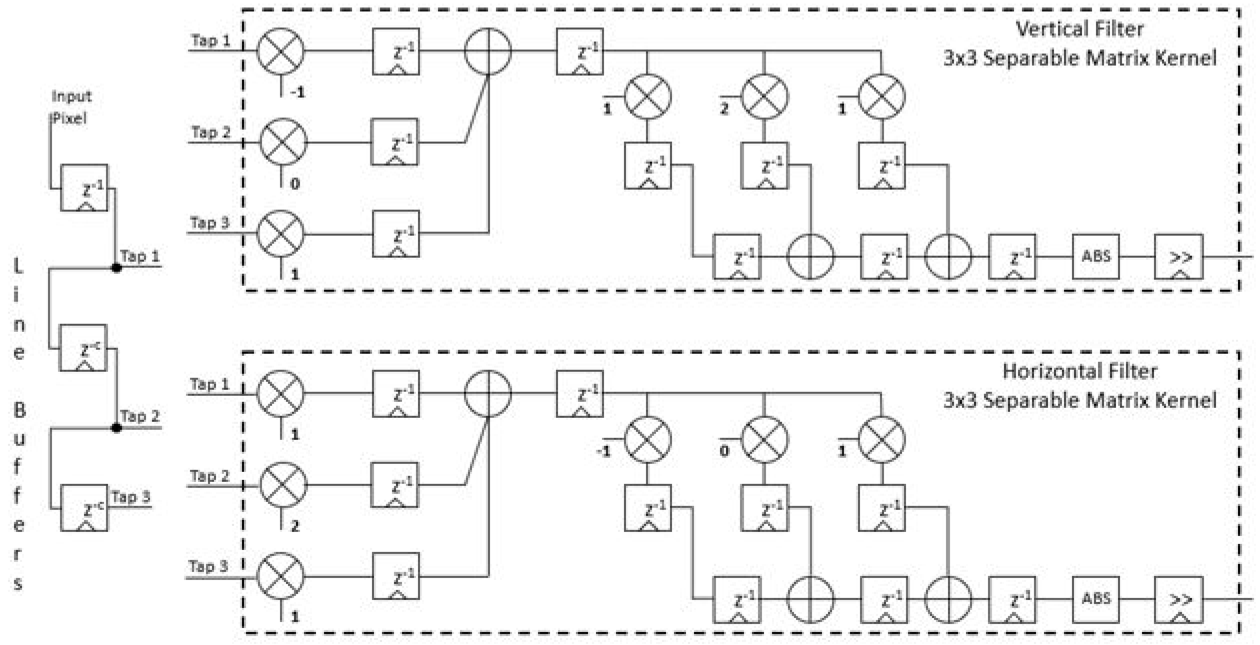


Figure 8: Sobel Filter

The block diagram shown below in Figure 9, shows the implemented Sobel Filter in a wrapper containing additional logic blocks for selection of the desired output, being: summed output, vertical filter output, horizontal filter output, or original/unfiltered output.

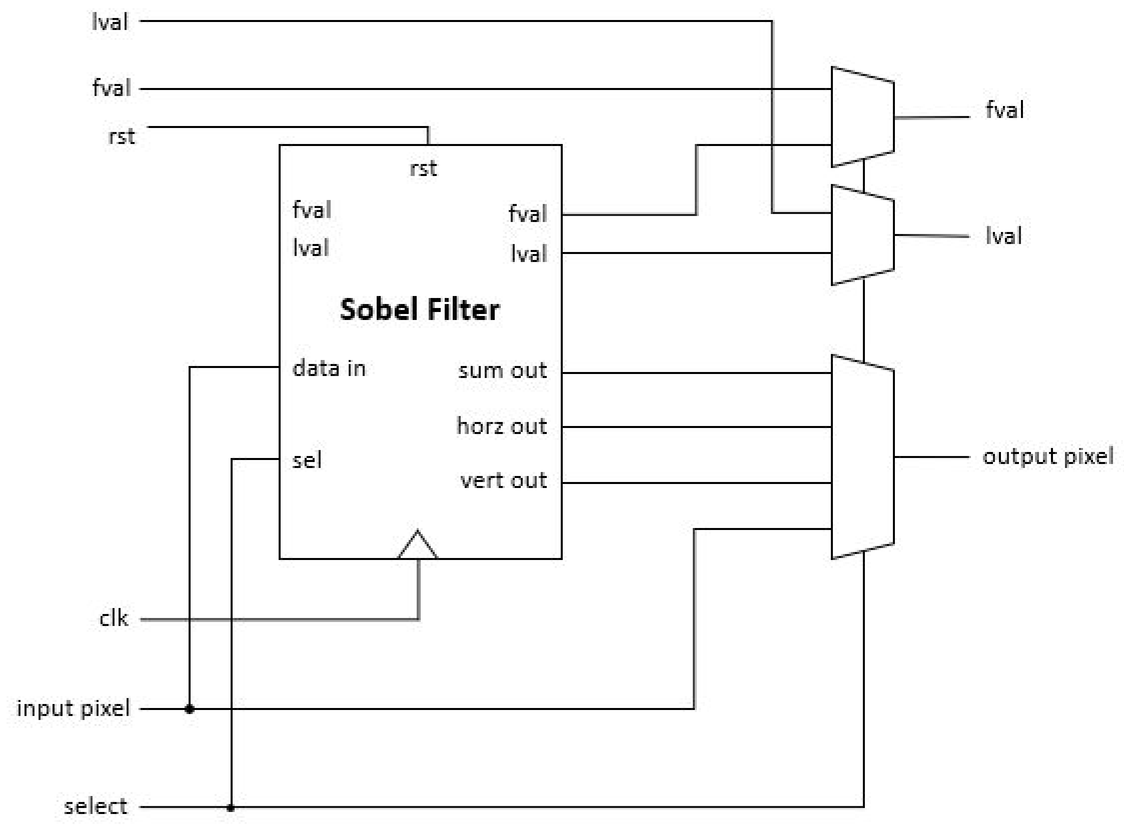


Figure 9: Sobel Wrapper

VGA CONVERSION

The VGA Conversion block is used to meet the standards set forth by VESA for 640x480 VGA video at 60 frames per second. The standard defines a clock rate and durations for front porch, sync pulse, and back porch durations. These values were created originally as physical constraints for CRT displays and are still supported by most monitors made today. As we have noticed, the VGA decoding and displaying implementation works differently for most monitor manufacturers, so sometimes their displays will differ somewhat for the same input.

The VGA standard dictates a 25.175MHz pixel clock for our resolution and frame rate. Because of limitations of clock synthesis tools on the FPGA, we are generating a 25.174MHz clock. This is extremely close to our target clock rate, and definitely close enough as some people have reported success using even a 25MHz clock. The horizontal front porch, back porch, and sync pulse are 16, 48, and 96 clock cycles, respectively. The vertical front porch, back porch, and sync pulse are 10, 33, and 2 clock cycles, respectively.

A Block RAM (BRAM) is placed at the input of the conversion block. It's depth is 320x240 = 76,800 elements, each being 8 bits for a total of 614,400 bits, or 600Kb of

BRAM space. In this project, the block never receives anything more than a 160x120 image due to available BRAM space on the part, but it would work with a 320x240 image as well. This BRAM is constantly overwritten at the board clock rate (50MHz) with only valid data (fval and lval high) from the previous block. The BRAM output address and values are driven at the 25.174MHz clock that was synthesized earlier.

Counters are kept to keep track of when the vertical and horizontal front porch, back porch, and sync pulses should occur. For simplicity and ease of verification, actual pixel data is output at counts 0 to 640 for horizontal and 0 to 480 for vertical. This means the first frame will be invalid since it wasn't preceded by a sync pulse, but the block likely hasn't received much valid data anyway and it will take the monitor a few frames to sync up to our pixel clock.

This block also handles upscaling from 160x120 or 320x240 to 640x480. It does this by extending the number of bits in the BRAM output address by the base two logarithm of the upscaling factor. The upscaling factor is defined by the horizontal resolution of the output over the horizontal resolution of the input, i.e. 640/160=4. These bottom bits added are then ignored when using the counter as the BRAM output address. This results in each address being used the same number of times as the upscale factor when the counter is incremented every clock cycle. The same upscaling must occur in the vertical direction, so another counter is kept to reset the BRAM output counter back to the start of the current row until it has been repeated the same number of times as each pixel.

This block will output 8 bit values for the pixel values. Unfortunately, the DAC on the Basys3 board only accepts 4 bit values. There is a separate block after the VGA Conversion block that does the scaling from 8 bits down to 4. This is made separate because it's functionality has nothing to with VGA timings, and so it can be swapped out very easily without messing with the complicated functionality of the VGA block.

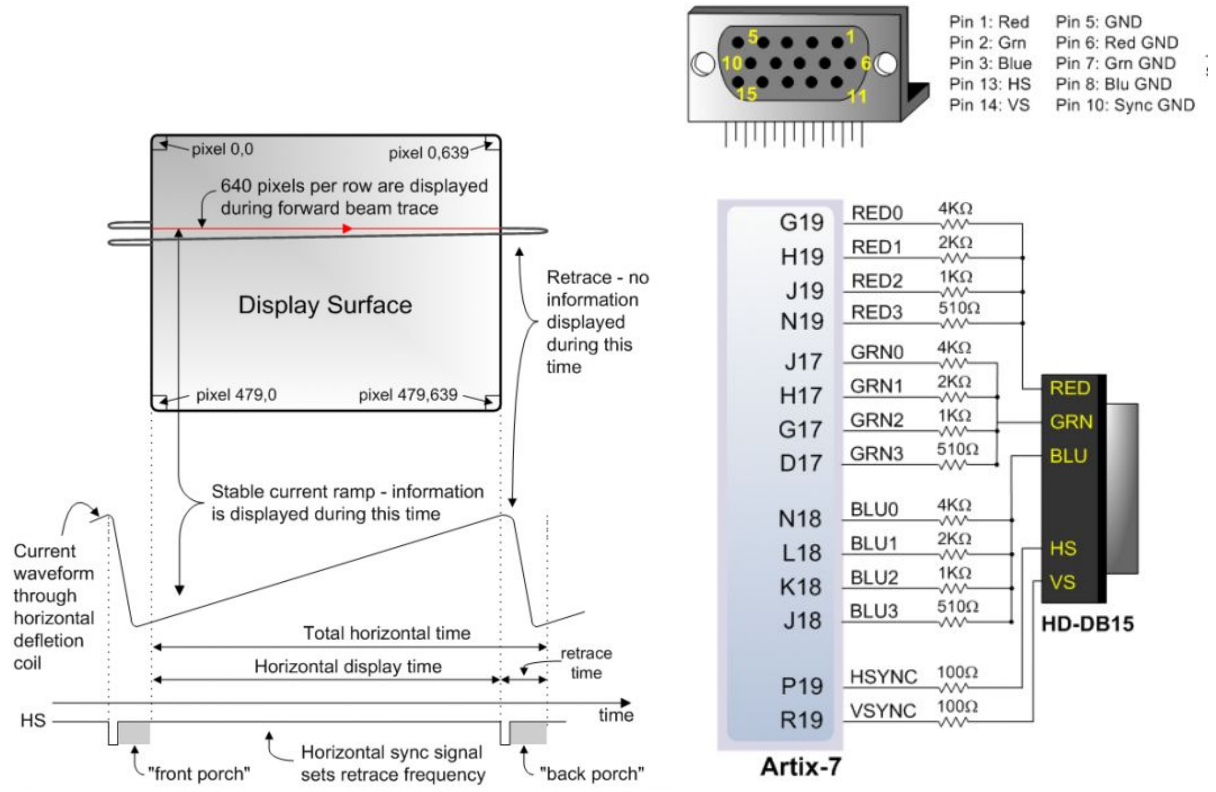


Figure 10: VGA Timing and Signal Diagrams
(*https://reference.digilentinc.com/basys3/refmanual*)

IMPLEMENTATION CHALLENGES

FAKE CAMERA SIMULATOR

There were very few implementation challenges associated with generating the synthetic image data. The only challenge which presented itself was creating the *.coe file needed for the PreLoaded Image within the internal BRAM. Initial attempts to create the *.coe file produces transposed results which were out of order when exiting the BRAM. This was a result of Matlabs column-major image interpretation as opposed to the expected row-major readout direction. An transpose on the image before converting it to the desired *.coe file was needed in order to solve the issue.

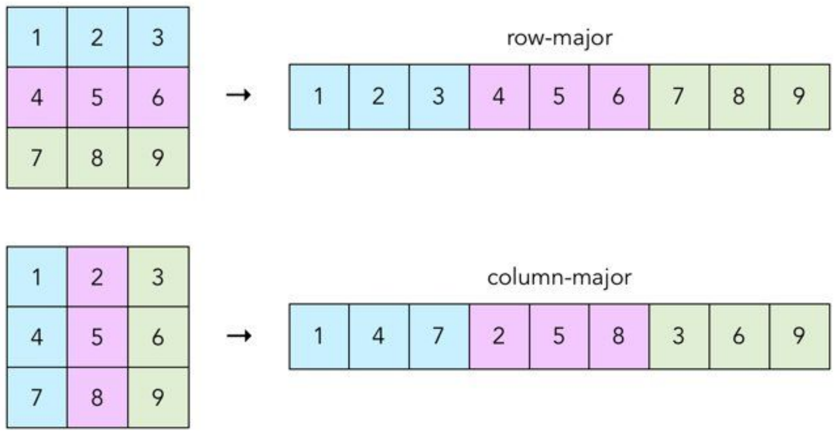


Figure 11: Matlab Readout vs. Expected

Sobel Filter

The Sobel Filter was a straightforward algorithm to implement in hardware. Because of the separable nature of the filter kernel matrices and identical structure of the vertical and horizontal filters, a single kernel was easy to design, with generics used to configure the two generated filters.

The main challenges in the design of the Sobel was ensuring data alignment through the pipeline, ensuring the matrix multiplications resulted in the expected fashion. The only other challenge was managing data representation and bit growth. Initial design lead to outputs that were half of the expected value. Through debugging it was quick and simple to identify that the bit growth was one bit less than needed. Correction of the bit growth to the proper size corrected the issue, with outputs being the appropriate value. IP BRAMS aided in reducing complexity of design by offering standard memory structures with all addressing and read/write functions needed.

TRANSPOSE CIRCUIT

The example in the architecture section of this document used a square frame for simplicity but the final design uses a rectangular frame. Implementing transposition of a rectangular frame initially caused issues because the transpose circuit was outputting a 120x160 frame but the edge detection circuit was expecting a 160x120 frame. The solution was to resize the transpose image. This was accomplished by designing the line and pixel counters to behave the same as they would to output the original image. On the short edge of the VGA monitor, the extra pixels at the bottom of the transpose image are not output. On the long edge of the VGA monitor, blank pixels (0x00) are output to the right of the transpose image. A MATLAB model was created to visualize this, shown in Figure 12.

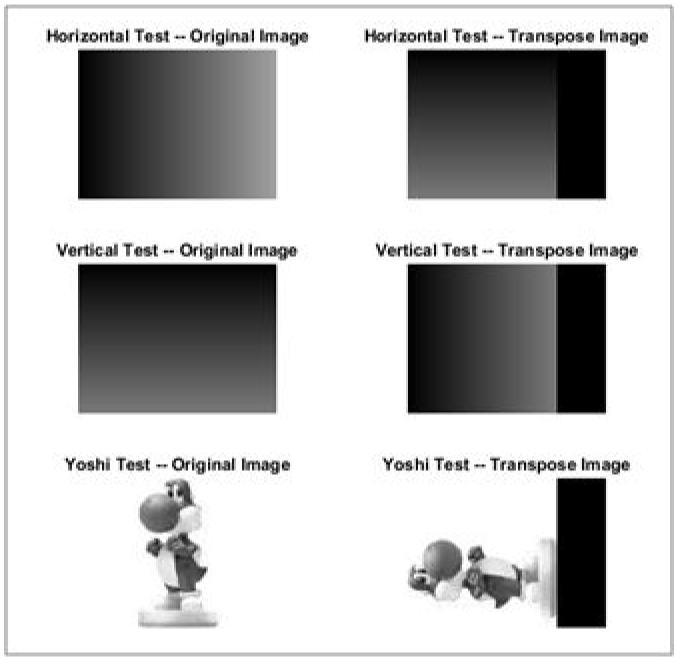


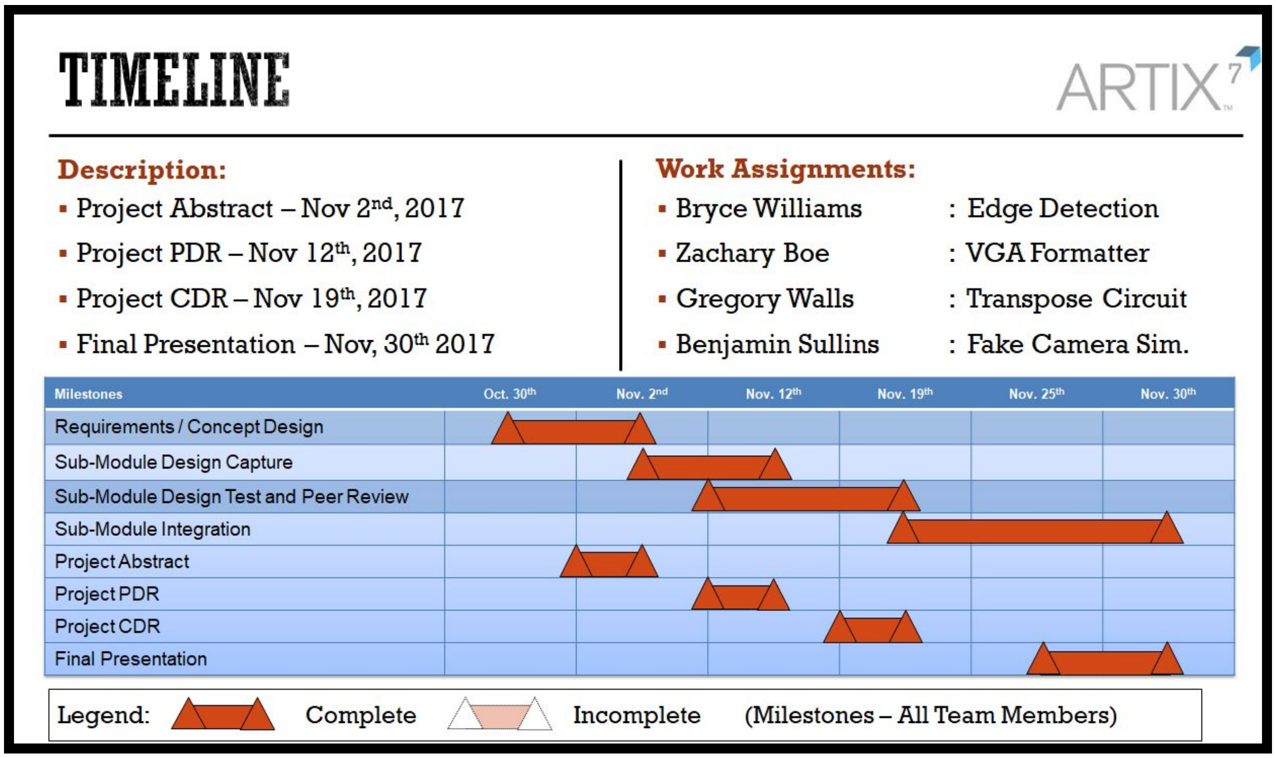
Figure 12: Rectangular Transpose MATLAB Model

VGA CONVERSION

We had a problem during development where one team member's monitor gave drastically different results than others. The problem stemmed from the fact that we couldn't find any resources that specified what value the RGB output lines should have during the front porch, back porch, and sync pulse. The monitor really only cares about the value of these lines during the active portion of the signal so it was thought to be okay to leave the value unconstrained during the inactive portion. After much debugging, an active signal was added and the RGB lines are set low when not in the active portion of the signal. This corrected the issue for the team member and had no effect on the other. This is thought to be one of the differences that different monitor manufacturers have in their VGA implementations.

WORK DISTRIBUTION

The following figure below provides the assignments for each team member. Significant emphasis was put on individual completion of each sub-module for providing additional time during integration. Although some sub-modules might have been slightly more/less difficult than another, each team member completed their task and went on to assist other teammates in completing other design modules. Overall, the work was distributed evenly amongst all project members.



TIMELINE

ARTIX 7

Description:

- Project Abstract – Nov 2nd, 2017
- Project PDR – Nov 12th, 2017
- Project CDR – Nov 19th, 2017
- Final Presentation – Nov, 30th 2017

Work Assignments:

Bryce Williams	: Edge Detection
Zachary Boe	: VGA Formatter
Gregory Walls	: Transpose Circuit
Benjamin Sullins	: Fake Camera Sim.

Milestones	Oct. 30th	Nov. 2nd	Nov. 12th	Nov. 19th	Nov. 25th	Nov. 30th
Requirements / Concept Design						
Sub-Module Design Capture						
Sub-Module Design Test and Peer Review						
Sub-Module Integration						
Project Abstract						
Project PDR						
Project CDR						
Final Presentation						

Legend: Complete Incomplete (Milestones – All Team Members)

Figure 13: Timeline and Works Assignments

TEST BENCH DETAILS

The top-level test bench is very straight forward. It instantiates all submodules and sets all of the select signals. Additionally, it sets the reset signal high for 1000 nsec.

Sobel Test Bench:

The Sobel Test Bench is comprised of three parts: a MATLAB preprocessor, a HDL test bench, and a MATLAB postprocessor. The test bench flow is shown in Figure 14, below.

MATLAB Preprocessor:

The MATLAB Preprocessor is responsible for converting a stock image to grayscale and resizing the image to the appropriate length and width for which the Sobel filter is configured. After grayscale and resizing is applied, the preprocessor generates a text file comprised of 8-bit binary values for each of the pixel intensities in the image. This file serves as the data input file for the HDL test bench.

HDL Test Bench:

The HDL Test Bench is responsible for generating and setting up a Sobel Filter, generating clock cycles, generating control signals, opening the data input file, and passing input data to the filter. The test bench produces three text files, as its output, comprised of the Sobel filter outputs, specifically the Horizontal Filter Gradient 8-bit binary values, the Vertical Filter 8-bit Gradient values, and the Summed 8-bit Gradient values.

MATLAB Postprocessor:

The MATLAB Postprocessor is responsible for rendering output images generated by the HDL test bench. It opens the three text file outputs from the HDL test bench, that is the Vertical, Horizontal, and Summed filter data, and reads each file into working arrays. MATLAB functions are used to convert the text data into decimal values before being placed into its relative array and location. MATLAB functions are then used to render each array as an image, saving each as a reference file in an output file. In conjunction with saving the processed images, the original and each of the filtered images are displayed in a figure window.

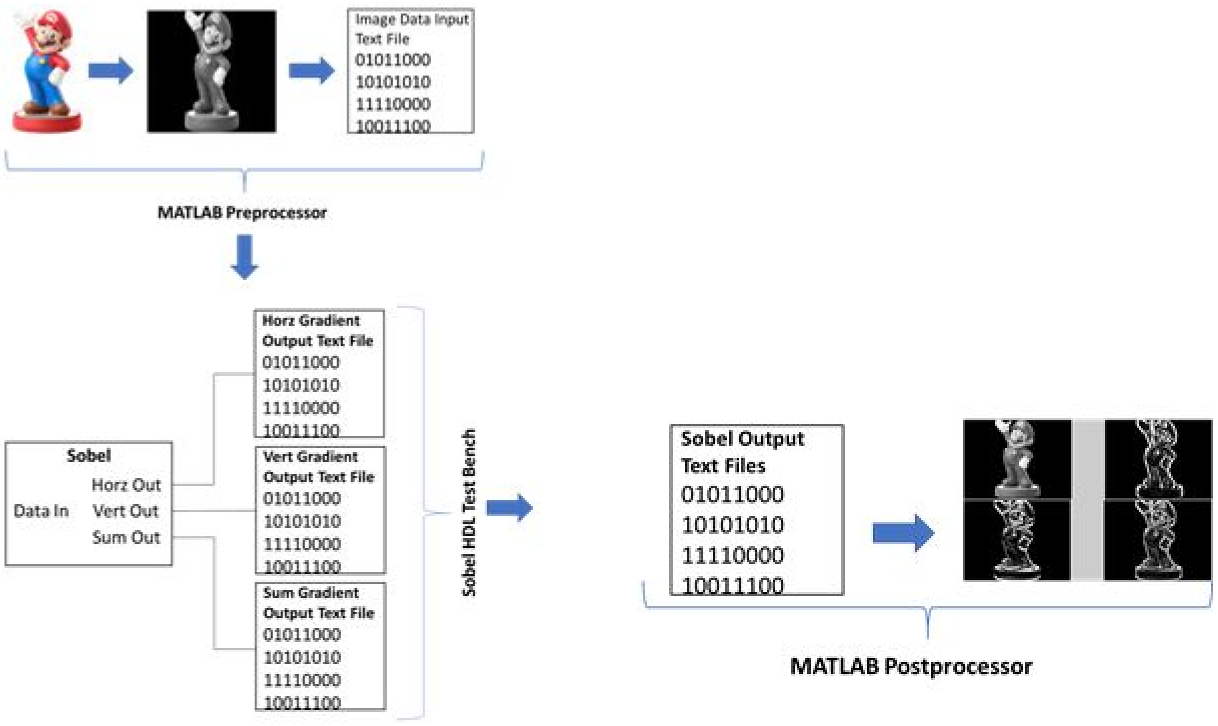


Figure 14: Sobel Test Bench Flow

SIMULATION RESULTS

Fake Camera Simulator:

Vivado's ISIM tool was used for simulating the results of the synthetic image data. The timing was reflective of the generics used to generate the 160x120 image along with the invalid pixels, clock rate, and integration time needed for a 50 Hz image. Shown below in Figure 15 - 17 is the simulation results, depicting the correct implementation and expected output.

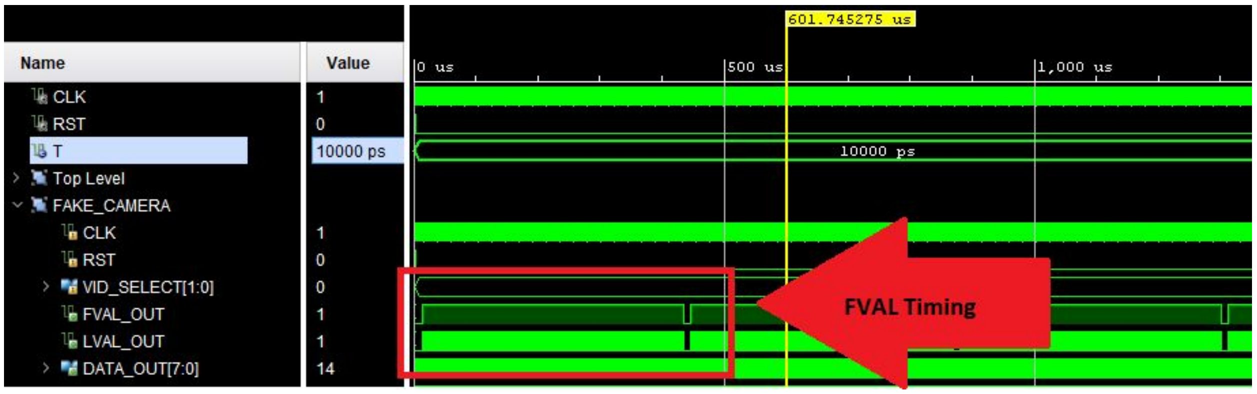


Figure 15: Fake Camera FVAL Timing

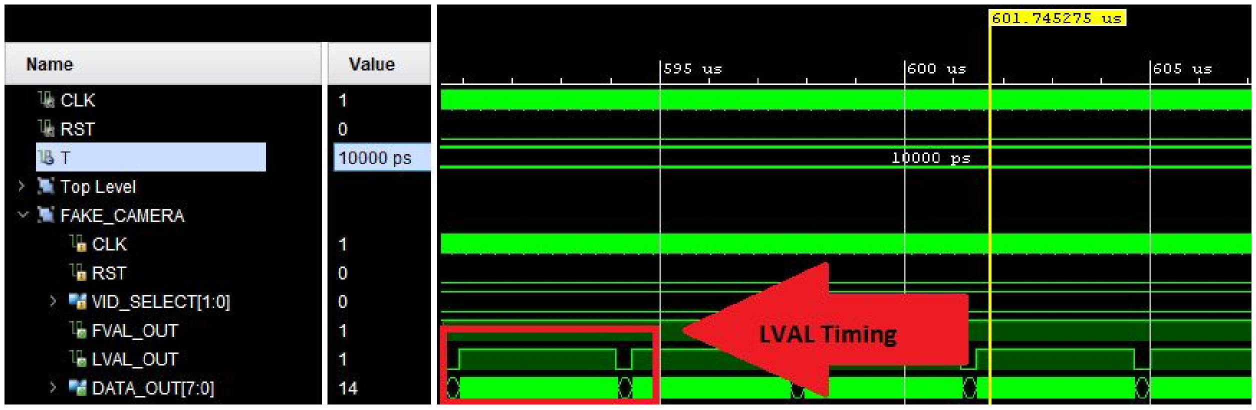


Figure 16: Fake Camera LVAL Timing

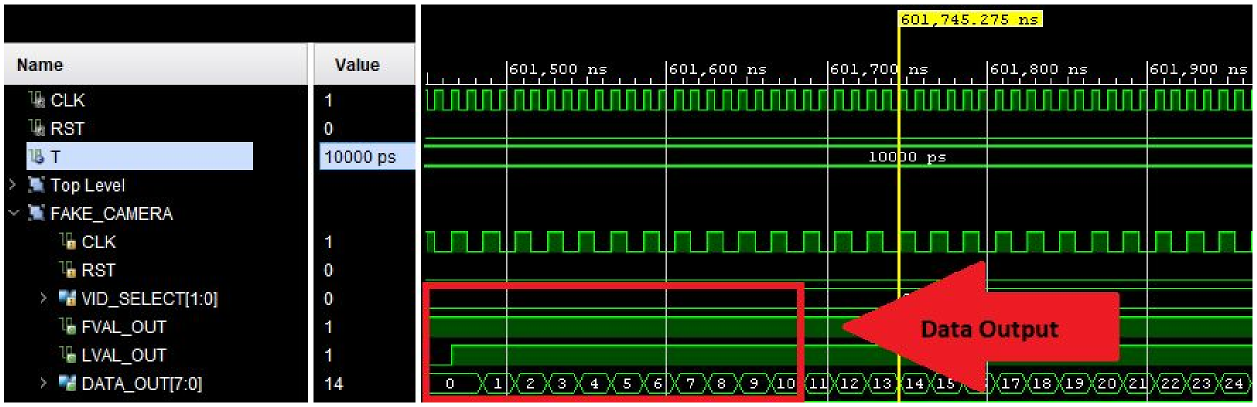


Figure 17: Fake Camera Data Output (Horizontal Ramp)

Transpose Circuit:

Vivado's ISIM tool was used to simulate transpose circuit. Figure 18 shows a simulation run where the first input frame is buffered and then two frames are output.

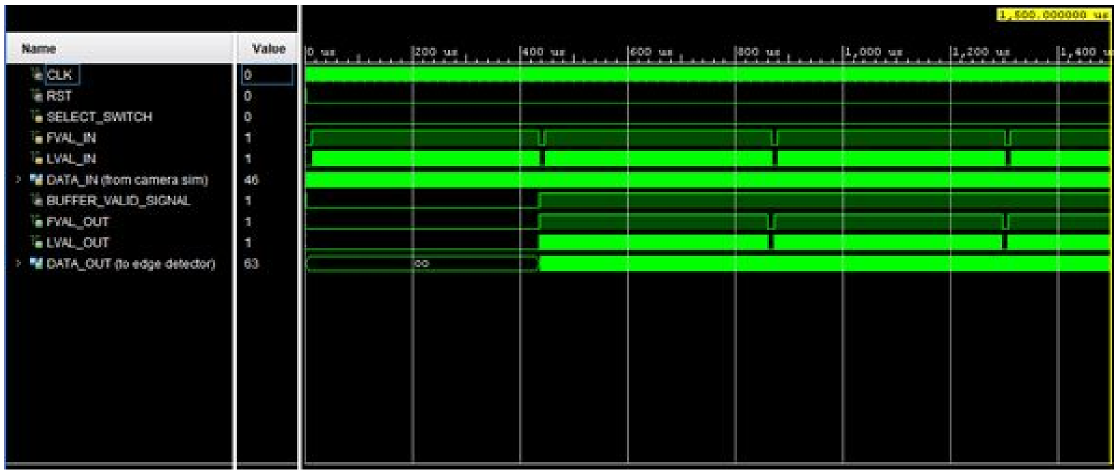


Figure 18: Transpose Circuit Simulation of Three Full Input Frames

Figure 19 shows a close-up of a simulation run when the original image is output.

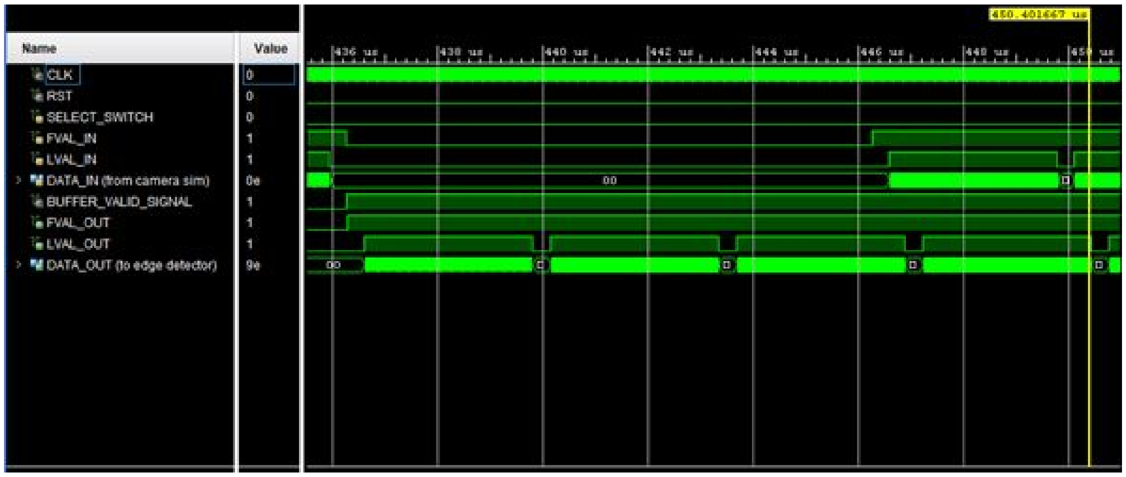


Figure 19: Several Lines of the Original Image Output in Simulation

Figure 20 shows a close-up of a simulation run when the transpose image is output.

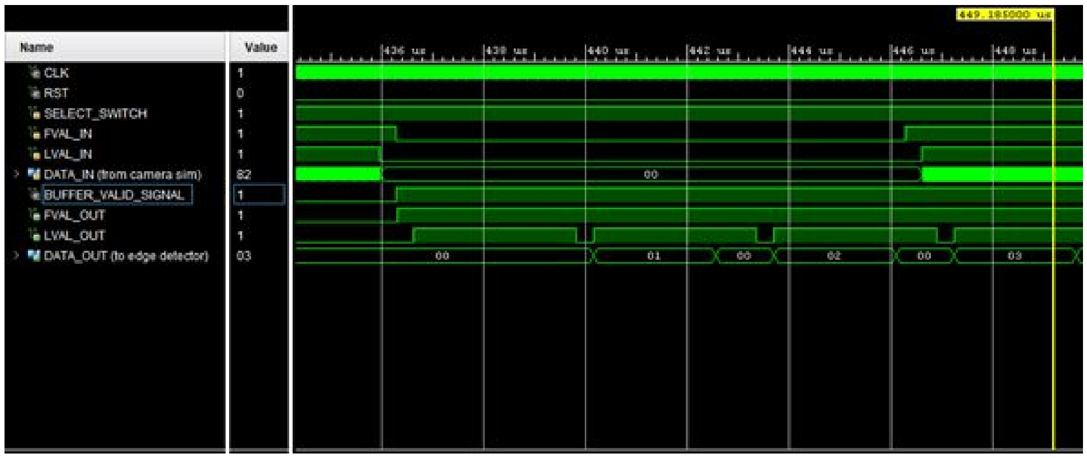


Figure 20: Several Lines of the Transpose Image Output in Simulation

By observing the timing of the signals and data outputs in simulation, the transpose circuit was debugged and verified before implementation.

Sobel Simulation Results:

Figures 21 and 22 below show the results of VHDL simulation, post-processed in MATLAB to render the images. The leftmost image in each set is the original image. To its right is the Horizontal Filter Output which captures vertical edges. The next image in each set is the Vertical Filter Output which captures horizontal edges, and the final image in each set is the averaged sum of the vertical and horizontal filters resulting in the complete edge detection image.

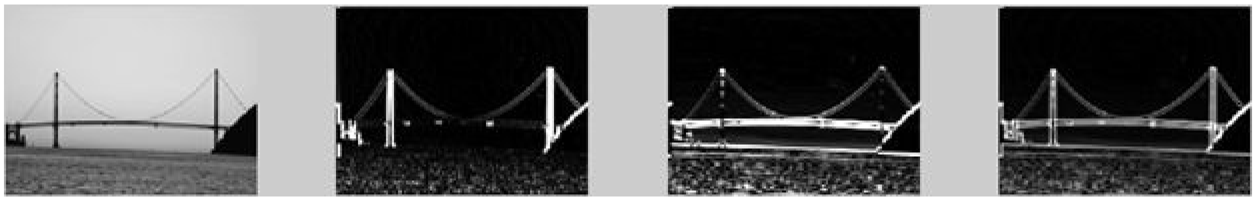


Figure 21: Simulation Output - Original Image, Horizontal Filter Output, Vertical Filter Output, Summed Outputs

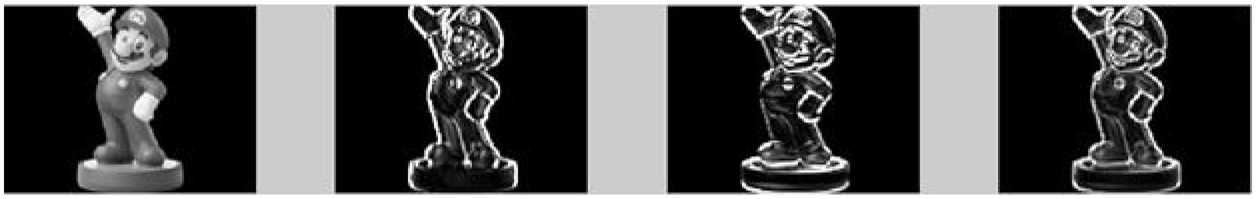


Figure 22: Simulation Output - Original Image, Horizontal Filter Output, Vertical Filter Output, Summed Outputs

VGA CONVERSION

A modelling module was created in Matlab to simulate the resolution upscaling and pixel value downconversion from 8 bits to 4 to meet the VGA DAC requirements. The verification of the porches, sync signals, BRAM addresses, and output values was verified in simulation.

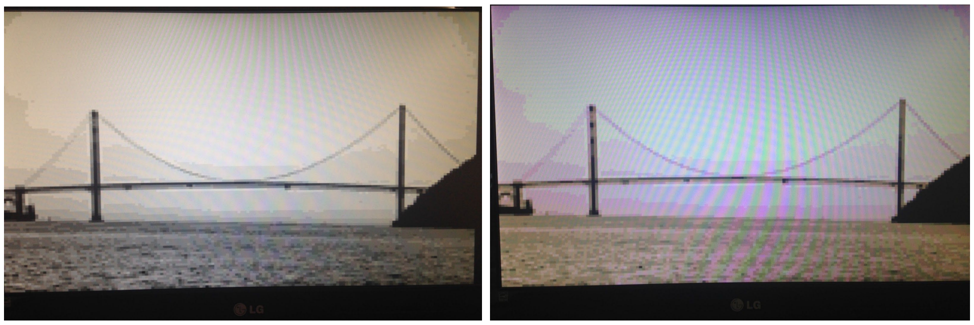


Figure 23: Actual VGA Output W/ Color Mapping

AREA AND POWER RESULTS

Fake Camera Simulator:

Slice Logic					
Site Type		Used	Fixed	Available	Util%
Slice LUTs		74	0	20800	0.36
	LUT as Logic	74	0	20800	0.36
	LUT as Memory	0	0	9600	0
Slice Registers		77	0	41600	0.19
	Register as Flip Flop	77	0	41600	0.19
	Register as Latch	0	0	41600	0
F7 Muxes		8	0	16300	0.05
F8 Muxes		0	0	8150	0

Power	
Total On-Chip Power (W)	0.199
Dynamic (W)	0.127
Device Static (W)	0.072
Effective TJA (C/W)	5
Max Ambient (C)	84
Junction Temperature (C)	26
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

Design Timing Summary		
WNS(ns)	WHS(ns)	WPWS(ns)
13.938	0.06	3

Memory				
Site Type	Used	Fixed	Available	Util%
Block RAM Tile	5	0	50	10
RAMB36/FIFO*	5	0	50	10
RAMB36E1 only	5			
RAMB18	0	0	100	0

Table 2: Fake Camera Simulator Resource Utilization

Transpose Circuit:

Slice Logic					
Site Type		Used	Fixed	Available	Util%
Slice LUTs		131	0	20800	0.63
	LUT as Logic	131	0	20800	0.63
	LUT as Memory	0	0	9600	0.00
Slice Registers		82	0	41600	0.20
	Register as Flip Flop	82	0	41600	0.20
	Register as Latch	0	0	41600	0.00
F7 Muxes		8	0	16300	0.05
F8 Muxes		0	0	8150	0.00

Power	
Total On-Chip Power (W)	0.079
Dynamic (W)	0.009
Device Static (W)	0.070
Effective TJA (C/W)	5.0
Max Ambient (C)	84.6
Junction Temperature (C)	25.4
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

Design Timing Summary		
WNS(ns)	WHS(ns)	WPWS(ns)
4.362	0.086	4.500

Memory				
Site Type	Used	Fixed	Available	Util%
Block RAM Tile	5	0	50	10.00
RAMB36/FIFO*	5	0	50	10.00
RAMB36E1 only	5			
RAMB18	0	0	100	0.00

Table 3: Transpose Circuit Resource Utilization

Edge Detection:

Slice Logic					
Site Type		Used	Fixed	Available	Util%
Slice LUTs		292	0	20800	1.40
	LUT as Logic	280	0	20800	1.35
	LUT as Memory	12	0	9600	0.13
Slice Registers		535	0	41600	1.29
	Register as Flip Flop	535	0	41600	1.29
	Register as Latch	0	0	41600	0.00
F7 Muxes		8	0	16300	0.05
F8 Muxes		0	0	8150	0.00

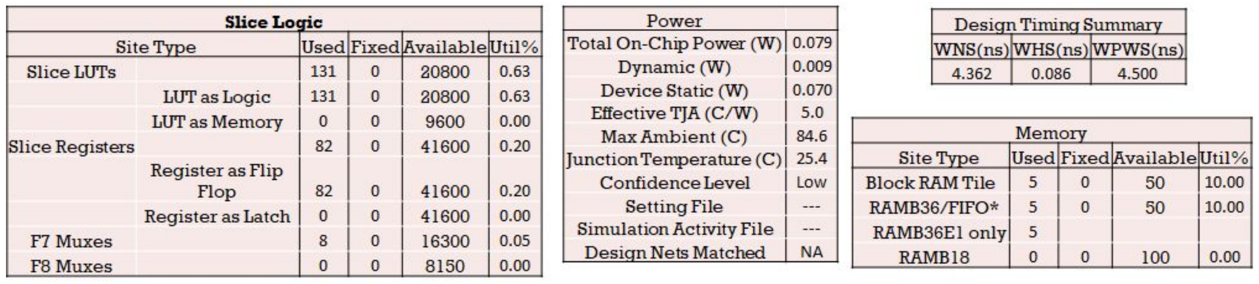
Power	
Total On-Chip Power (W)	0.193
Dynamic (W)	0.121
Device Static (W)	0.072
Effective TJA (C/W)	5.0
Max Ambient (C)	84.0
Junction Temperature (C)	26.0
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

Design Timing Summary		
WNS(ns)	WHS(ns)	WPWS(ns)
14.530	0.034	3.000

Memory				
Site Type	Used	Fixed	Available	Util%
Block RAM Tile	6	0	50	12.00
RAMB36/FIFO*	5	0	50	10.00
RAMB36E1 only	5	---	---	---
RAMB18	2	0	100	2.00

Table 4: Edge Detection Resource Utilization

VGA Conversion:



Slice Logic					
Site Type		Used	Fixed	Available	Util%
Slice LUTs		131	0	20800	0.63
	LUT as Logic	131	0	20800	0.63
	LUT as Memory	0	0	9600	0.00
Slice Registers		82	0	41600	0.20
	Register as Flip Flop	82	0	41600	0.20
	Register as Latch	0	0	41600	0.00
F7 Muxes		8	0	16300	0.05
F8 Muxes		0	0	8150	0.00

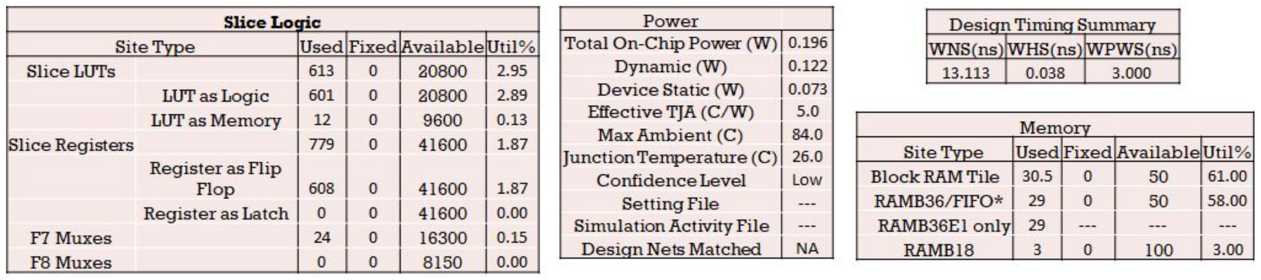
Power	
Total On-Chip Power (W)	0.079
Dynamic (W)	0.009
Device Static (W)	0.070
Effective TJA (C/W)	5.0
Max Ambient (C)	84.6
Junction Temperature (C)	25.4
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

Design Timing Summary		
WNS(ns)	WHS(ns)	WPWS(ns)
4.362	0.086	4.500

Memory				
Site Type	Used	Fixed	Available	Util%
Block RAM Tile	5	0	50	10.00
RAMB36/FIFO*	5	0	50	10.00
RAMB36E1 only	5			
RAMB18	0	0	100	0.00

Table 5: VGA Conversion Resource Utilization

Overall Project:



Slice Logic					
Site Type		Used	Fixed	Available	Util%
Slice LUTs		613	0	20800	2.95
	LUT as Logic	601	0	20800	2.89
	LUT as Memory	12	0	9600	0.13
Slice Registers		779	0	41600	1.87
	Register as Flip Flop	608	0	41600	1.87
	Register as Latch	0	0	41600	0.00
F7 Muxes		24	0	16300	0.15
F8 Muxes		0	0	8150	0.00

Power	
Total On-Chip Power (W)	0.196
Dynamic (W)	0.122
Device Static (W)	0.073
Effective TJA (C/W)	5.0
Max Ambient (C)	84.0
Junction Temperature (C)	26.0
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

Design Timing Summary		
WNS(ns)	WHS(ns)	WPWS(ns)
13.113	0.038	3.000

Memory				
Site Type	Used	Fixed	Available	Util%
Block RAM Tile	30.5	0	50	61.00
RAMB36/FIFO*	29	0	50	58.00
RAMB36E1 only	29	---	---	---
RAMB18	3	0	100	3.00

Table 6: Overall Project Resource Utilization

APPENDIX

Below is the file folder structure:

```
[Local Path]\ECE-6276
    \Abstract    - Repository for Abstract Products
    \CDR         - Repository for CDR Products
    \PDR         - Repository for PDR Products
    \ReadMe.ppt  - Instructions On Running Program
    \matlab      - Repository for Matlab Products
        \coe_generator      - Repository for COE File Generation
        \input_files        - Repository for Test Images
        \output_files       - Repository for Output Images and Test Results
        \reference_design   - Repository for Sub-Module Design Constructs
    \firmware - Repository For All Firmware Products
        \constraints                        - XDC File Location
        \core_src                           - Generic IP Location
        \docs                               - Firmware Documents
        \implement                          - Bitstream Locations
            \PROJECT_TOP_Gate.bit               - Golden Gate Bridge
            \PROJECT_TOP_Mario.bit              - Mario
            \PROJECT_TOP_Yoshi.bit              - Yoshi
            \TB_FILTER_behav.wcfg               - ISIM Waveform Configuration
        \packages                           - Library Files
            \generic_utilities                  - Custom Library File
        \vhdl_src
            \project_top.vhd                    - Top Level Project File
            \fake_camera.vhd                    - Fake Camera Simulator
            \video_transpose.vhd                - Transpose Circuit
            \sobel_wrapper.vhd                  - Edge Detection Wrapper
            \sobel.vhd                          - Sobel Filter
            \filter_mask_3x3.vhd                - Sobel Filter SubModule
            \vga_converter.vhd                  - VGA Circuit
            \image_scaling.vhd                  - Color Mapping Circuit
            \tb_*                               - Test Bench Files
        \project.tcl    - Main Project TCL File
```