[[1]](#footnote-1)

Image Processing on FGPA

(May 2019)

Harrison Hsieh, Sergio Minera, Samuel Holt, Ivan Garcia, and Alexander Ulrichsen, Mohamed Aly

*Abstract*—There are many known resources and libraries for processing images through software applications. Instead, we developed a hardware implementation of an image processor which can manipulate images to perform various alterations such as adjusting brightness, encrypting the image, inverting the image, and changing the image to greyscale. The image can then be output directly from the FPGA to a monitor through VGA output.

*Index Terms*—Digital Images, FPGA, Image Processing, Nexys

# INTRODUCTION

T

HIS project was designed to be a full hardware implementation of an image processor. We wanted to be able to perform many traditional image processing functions without relying on software. The main driver for this project is the Nexys 4 DDR board which is based on the Artix FPGA created by Xilinx. We chose this board because it has many useful components such as switches and output ports. It also has good compatibility with the Verilog hardware design language and the Vivado IDE.

# Project Description

## Images

The Nexys board has enough memory to support two different source images simultaneously on the board. While one image could be used as a live feed from a computer or camera, we have supplied two images loaded directly into memory on the board. These two images are image files which are in the resolution of 640 pixels wide and 360 pixels high. This is because those are the dimensions which are supported by the video RAM on the Nexys board. The color of the image is then loaded via the palette memory files which are generated on a per-image basis. These files are separate from the image memory files and are what allow us to perform the manipulation on the image itself. Using the Nexys switches, it is possible to switch between the two images at any time without building and synthesizing again. This is accomplished via a case statement which is controlled by the first two switches on the Nexys board. Table 1 shows the table for each case of the two bits representing the switches. Images are displayed by writing to the red, green, and blue outputs of the VGA module.

TABLE I

Truth table of image selection switches

|  |  |
| --- | --- |
| Case | Result |
| 00 | Load image one with its palette. |
| *01* | Load image two with its palette. |
| *10* | Load image one with the encrypted palette. |
| *11* | Load image two with the encrypted palette. |

The encryption of states 10 and 11 will be mentioned in the encryption section of the palette. Shown in Figures 1, 2, 3, and 4 below are the two example images along with the states on the switches for loading the corresponding images.



Figure - Image One

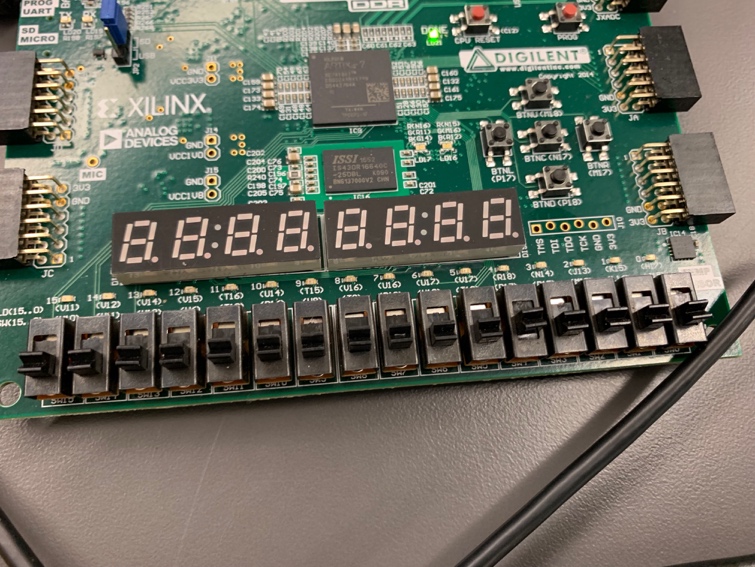


Figure - Switch Position for Image One



Figure - Image Two

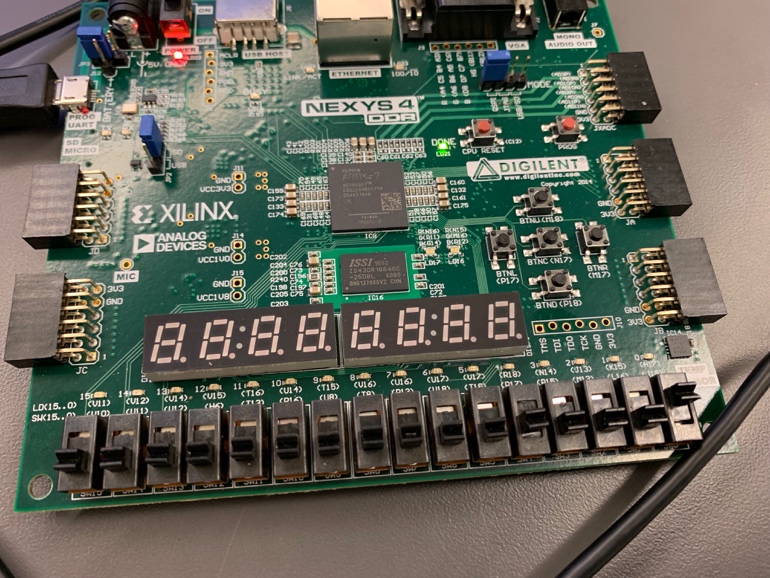


Figure - Switch Position for Image Two

## Encryption

Image encryption is accomplished by loading a different palette than the one generated for the image. This ensures that the overall structure of the image is preserved, however the image itself is not a reflection of the original image. In this case, another random palette is used to draw the image which results in an image that looks like the structure of the original image with a color map of a different image. This is one method of image encryption that is used because it makes the image look like a completely new image. Figures 5, 6, 7, and 8 show the switch positions and output of the encrypted images.

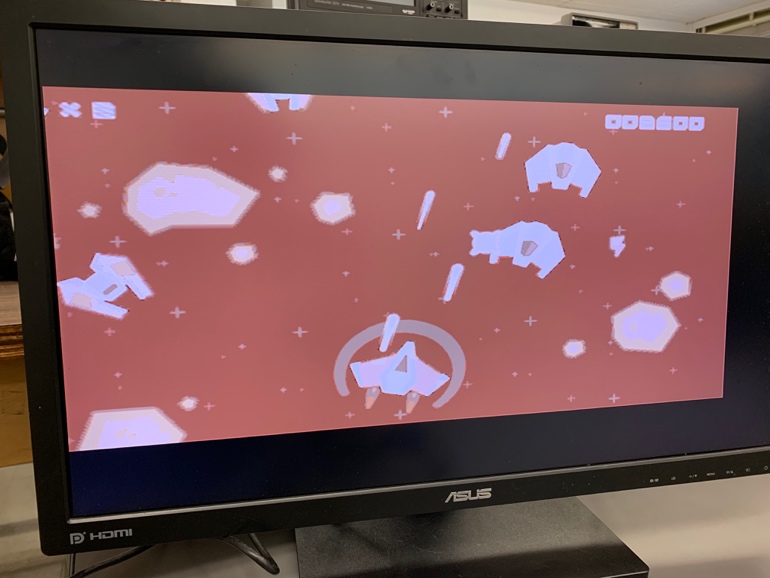


Figure - Encrypted Image One

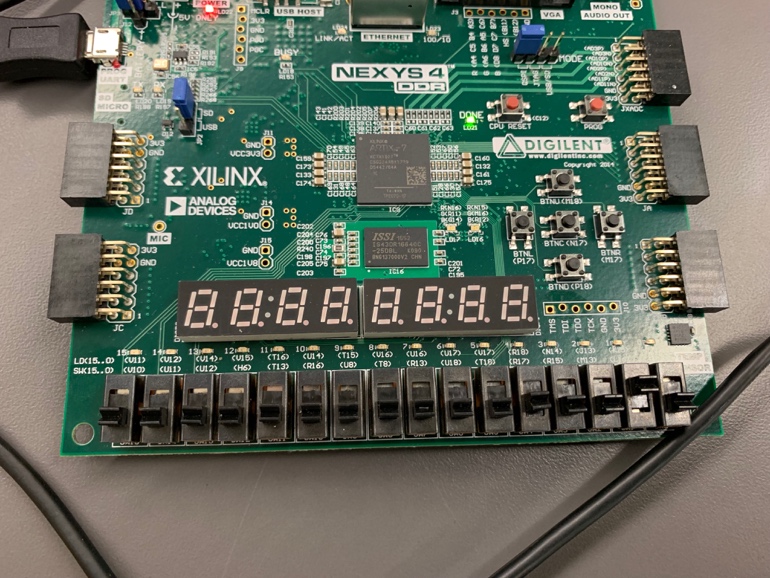


Figure - Switch Position of Encrypted Image One

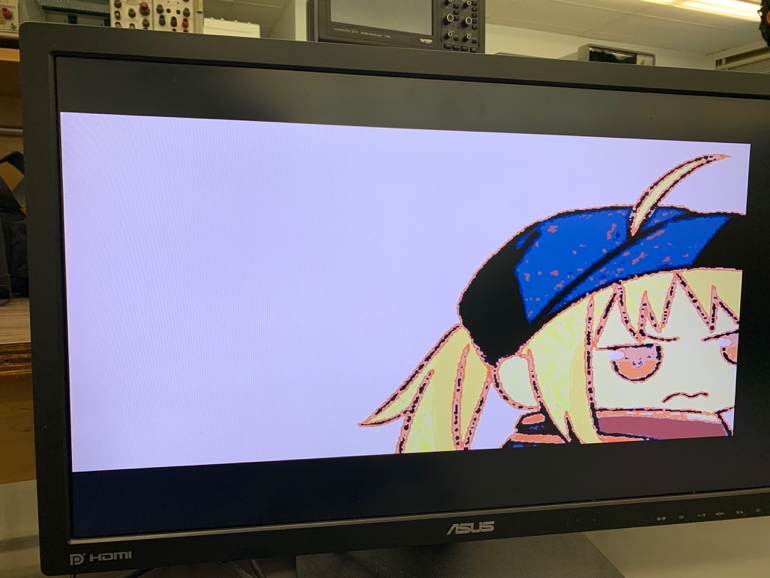


Figure - Encrypted Image Two

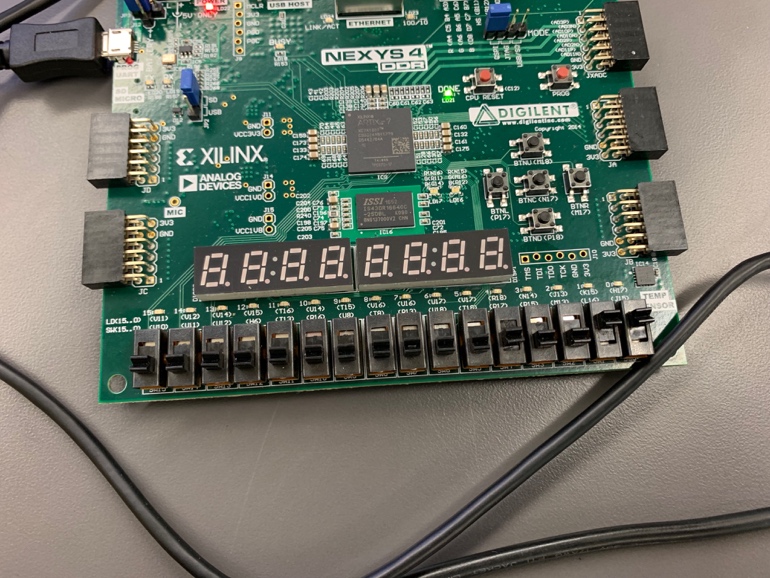


Figure - Switch Position of Encrypted Image Two

## Color Inversion

Color inversion is accomplished by subtracting the original red, green, and blue bit values from a value of 16. Since the images are in 4-bit color, the inverted version of the colors is taking the original value and subtracting that number from 16 which is one more than the highest possible 4-bit value. Figures 9, 10, 11, and 12 show the inverted versions of the example images along with the switch positions to generate this output.

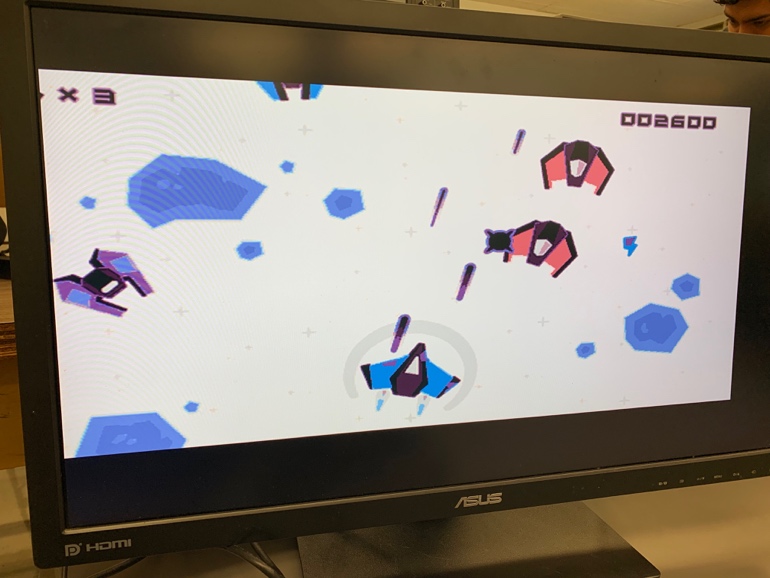


Figure 9 - Inverted Image One

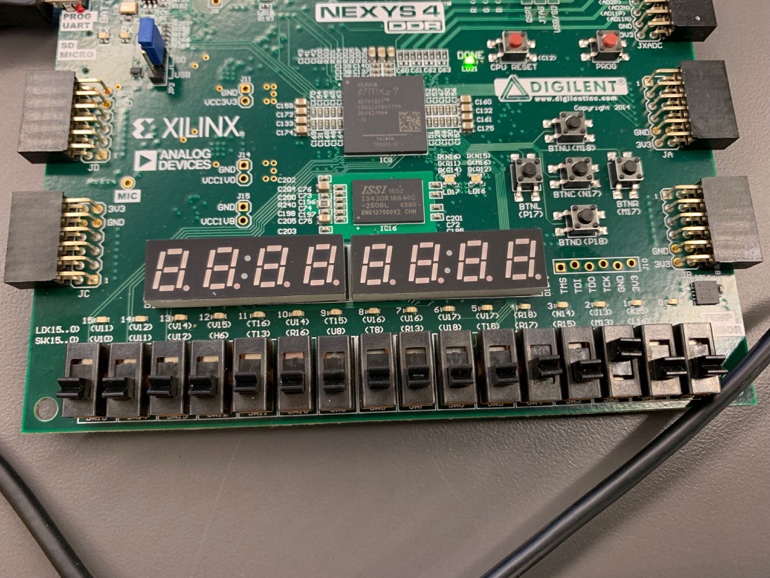


Figure 10 - Switch Position of Inverted Image One



Figure 11 - Inverted Image Two

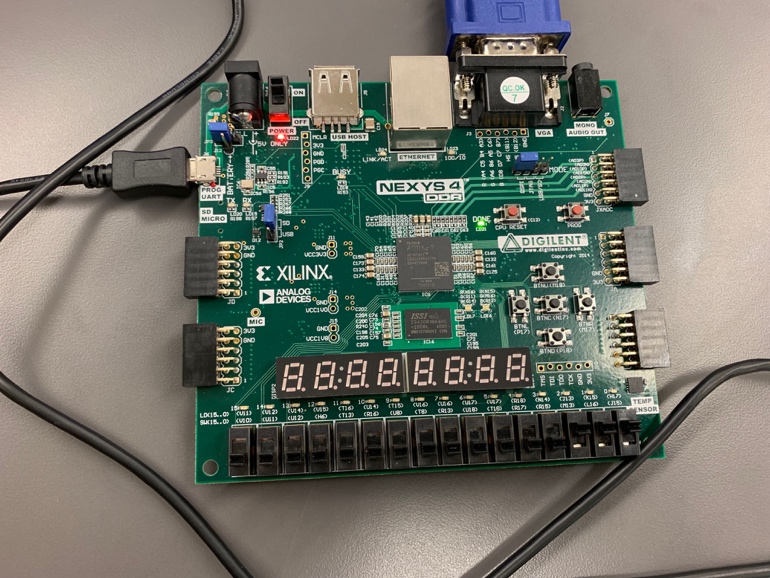


Figure 12 - Switch Position of Inverted Image Two

## Greyscale

Changing the images to greyscale is accomplished by adding all of the colors of each of the red, green, and blue sections and dividing the result by three. This ensures each of the bit values is either fully white or fully black. Figure 13 shows the greyscale version of image one and Figure 14 shows the switch position to achieve this output.



Figure 13 - Greyscale Image One

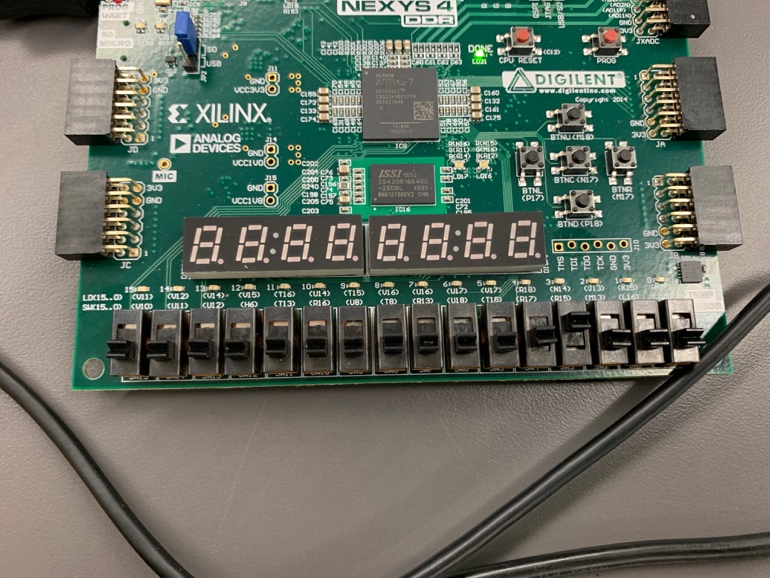


Figure 14 - Switch Position of Greyscale Image One

## Adjusting Brightness

Adjusting the brightness of images is slightly different from the previous processing effects because brightness has a modifier value. In the case of our brightness modifier, we multiple the original values of the palette by a user input value which is retrieved from switches on the FPGA. We handle this by first checking if the multiplier is zero. If it is, we simply show the original color palette. If the multiplier does not equal zero, we check to make sure the value of each palette section multiplied by the multiplier is less than 16. If It is less than 16, we show the modified color palette according to the user input. If the value is 16 or greater, we show the maximum value for each section of the color palette which is 15 since each color is represented with four bits. Figures 15-20 show examples of adjusted brightness on image one as well as the switch positions to achieve this output.



Figure 15 - Image One with Brightness Multiplier of Two

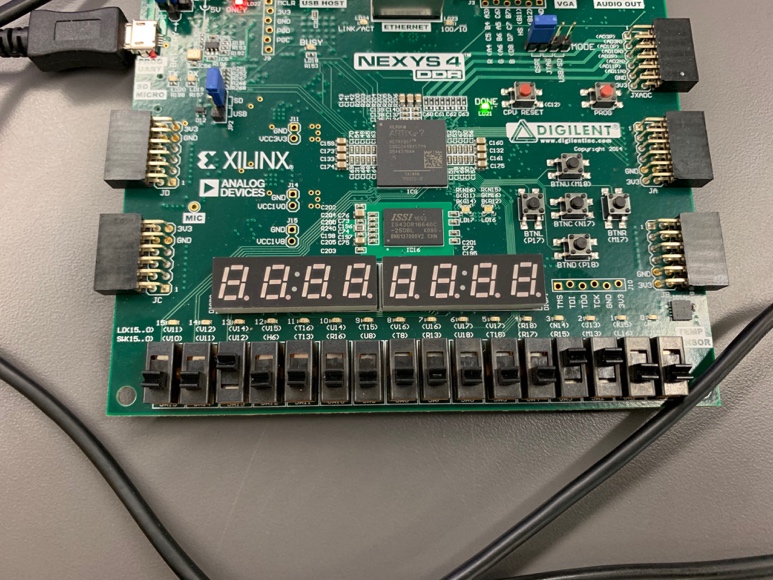


Figure 16 - Switch Position for Figure 15

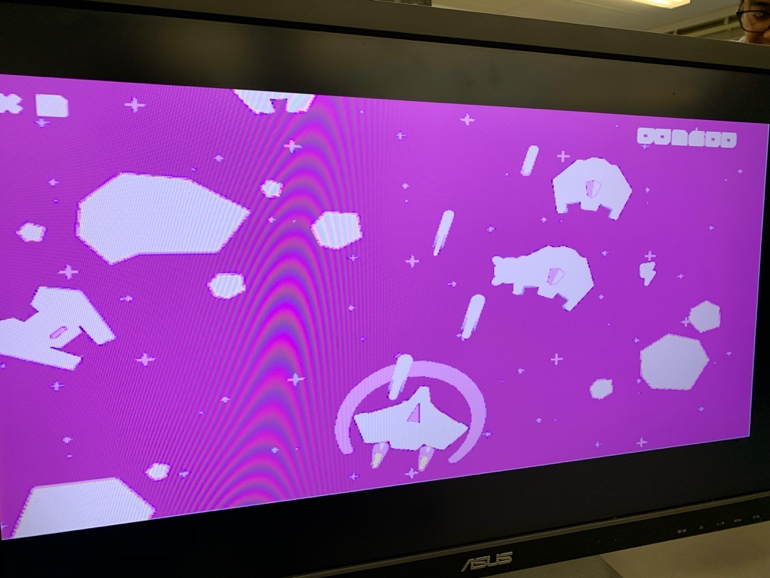


Figure 17 - Image One with Brightness Multiplier of Four

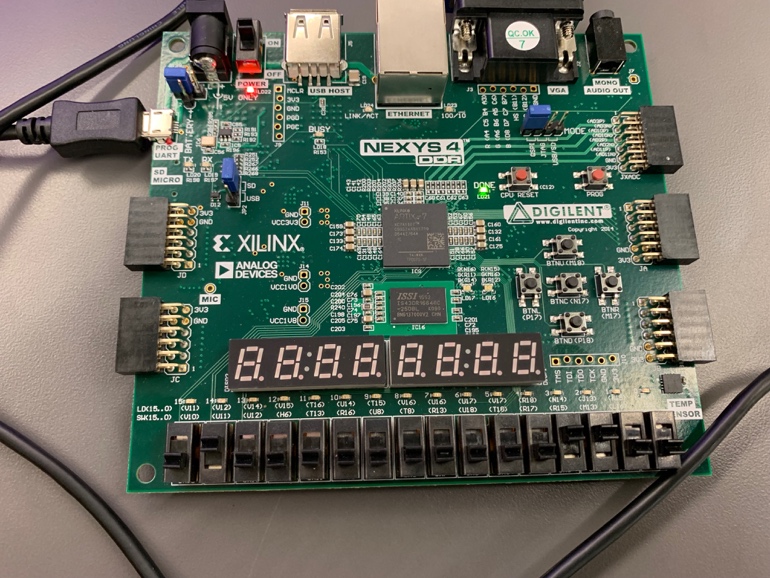


Figure 18 - Switch Position for Figure 17



Figure 19 - Image One with Brightness Multiplier of 8

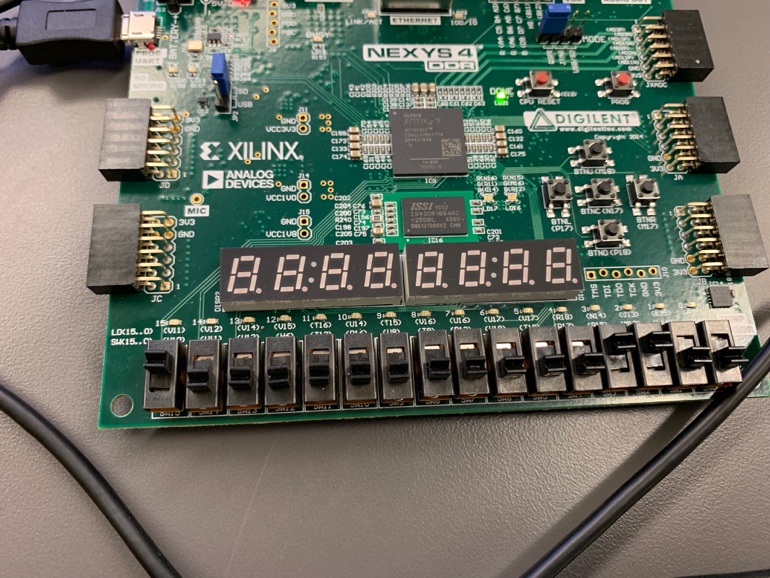


Figure 20 - Switch Position for Figure 19

# Conclusion

The goal of this project was to implement an image processor using techniques learned in the class. We were able to create a hardware implementation of an image processor which is able to change various aspects of the supplied images and display them using a VGA output. Further extensions of this project could be to incorporate a live video feed and apply the same output and processing effects on the live video. Another possibility would be to incorporate the ability to load an image file from another input such as a computer or another external device. This project could be extended to incorporate these aspects without changing much of the image processing design since that design only relies on modifying the bitmaps of the images.

Implementing RISC in our project would be difficult mostly due to failure on our end to foresee implementation difficulties. Our timeline was originally to simulate the image processing via Verilog, design the image processing algorithm on the FPGA without RISC, then implement the working algorithm with RISC. After successfully designing it, we soon realized implementing RISC would mean an entire overhaul of our program. The current build does not have a separate data path and control unit module. All instructions are hard coded in based on what we want the program to do. All the arithmetic functions are coded directly without a separate module. Implementing RISC means we would have to separate all of our instructions from the data while also assigning opcodes to the instructions we are using and using a separate ALU module to do the arithmetic operations in our image processing algorithm.

When we attempted to implement RISC, we ran into many errors that became very difficult to debug because of the sizable increase in modules added to our program. The idea was to design a RISC processor only the instructions we needed (and combination of instructions needed to be executed) so our case statement navigating the instructions would be minimal because of how few operations we needed. Many of our always statements utilizing if statements would need to be reorganized into using branches such as BNE and BNZ which was very difficult considering we had multiple nested if statements. The only thing we were able to implement remotely close to RISC was the SRAM module that loaded images to memory. The SRAM module is designed so that it could be taken and used in any program because of how the parameters are setup.

Acknowledgment

Thank you to Dr. Aly for his direct help and for his instruction in the Computer Architecture class. Also thank you to Cal Poly Pomona for the learning resources and study materials which aided in our completion of this project.

References

1. ECE 4300 Course Material. Dr. Mohamed Aly. 2019.
2. Helland, Tanner. “Seven Grayscale Conversion Algorithms (with Pseudocode and VB6 Source Code).” *Tanner Helland (Dot) Com*, 4 June 2012, www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/.

**Group Member Contributions**

**Harrison Hsieh** (012169313) Researching and implementing the goals of the project. Solved the main issues encountered while working on the project and brought all components of the project together.

**Sergio Minera** (011613446) Researching and implementing the project. Attempting to implement the RISC-V processor. Experimenting with various image processing effects and making it possible to synthesize them on an FPGA.

**Samuel Holt** (009871238) Researched the initial goals of the project. Helped test the project. Created images and structure of the report. Wrote down much of the information in the report.

**Ivan Garcia** (011965083) Assisted in the development of the project. Researched methods to implement different image processing effects. Worked on obtaining mem files and palette files by manipulating resolution of images.

**Alexander Ulrichsen** Assisted in the development of the project and generated test files.

**Mohamed Aly** (Professor of Computer Architecture Class) Helped with questions regarding the design of the project. Instructed all course material of the Computer Architecture course.

1. . [↑](#footnote-ref-1)