**California State University, Fresno**
**Lyles College of Engineering**
**Electrical and Computer Engineering Department**

**TECHNICAL REPORT**

Final Report
Experiment Title: Photo Manipulation
Course Title: ECE 178 (Embedded Systems)
Instructor: Dr. Reza Raeisi

Prepared by: Anthony Herrick
Student ID #300144976
Date Submitted: 12/11/2022

**INSTRUCTOR SECTION**

Comments: _____
_____
_____
_____
_____
_____
_____

Final Grade: _____

TABLE OF CONTENTS

LIST OF FIGURES

## Project Objective

The objective of this project was to create a soft-core embedded system and accompanying software to be able to read bitmap images from an SD Card, manipulate them, and display them onto a monitor via the VGA port. Specifically, for this project, we will be implementing the following types of photo manipulation: Brightness, Color Shifting, Image Mirroring, Wave, Red/Green/Blue Isolation, Next Image, and Redisplaying the photo.

## Hardware Requirements

- Computer with Intel FPGA Monitor program 16.1
- Computer with Quartus Prime 16.1.
- DE2-115 Board
- A-B USB Cable
- VGA Cable
- SD Card
- Monitor

## Software Requirements

- Quartus Prime with Qsys, version 16.1.
- FAT16 SD Card Formatting

## Background

24Bit Map Image format

The 24 bit map image (.bmp) has several key components in which understanding of them is vital to the success of this project. The file begins with a 54 byte header. For our purposes, this Header is not important to our project. Each pixel of the photo consists of 3 bytes, the first three bytes correspond to the bottom left of the image, and sequentially the pixels move left to right, and bottom to top.

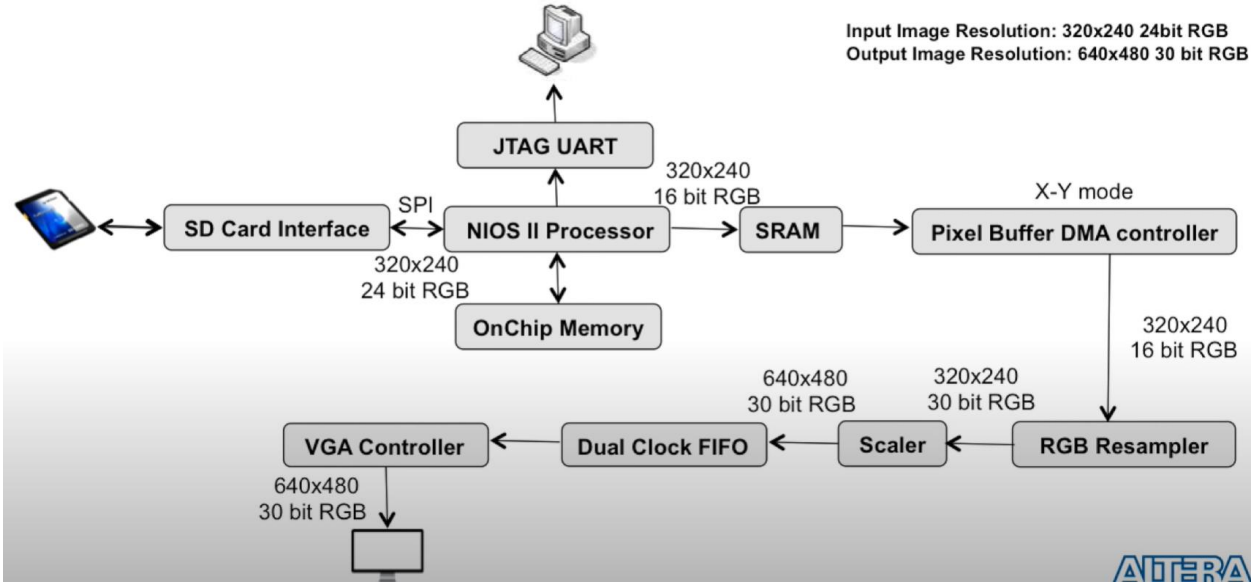
| Bitmap File Structure | | | |
|---|---|---|---|
| **Block** | **Field** | **Width** | **Description** |
| **BITMAPFILEHEADER**<br>**Fields:** 5<br>**Width:** 14 bytes | FileType | 2 bytes | A 2 character string value in ASCII. It must be 'BM' or '0x42 0x4D' |
| | FileSize | 4 bytes | An integer (unsigned) representing entire file size in bytes (number of bytes in a BMP image file ) |
| | Reserved | 2 bytes | To be utilized by an image processing application. Initialized to '0' integer (unsigned) value. |
| | Reserved | 2 bytes | To be utilized by an image processing application. Initialized to '0' integer (unsigned) value. |
| | PixelDataOffset | 4 bytes | An integer (unsigned) representing the offset of actual pixel data in bytes. |
| **BITMAPINFOHEADER**<br>**Fields:** 11<br>**Width:** 40 bytes | HeaderSize | 4 bytes | An integer (unsigned) representing the size of the header in bytes. It should be '40' in decimal. |
| | ImageWidth | 4 bytes | An integer (signed) representing the width of the final image in pixels. |
| | ImageHeight | 4 bytes | An integer (signed) representing the height of the final image in pixels. |
| | Planes | 2 bytes | An integer (unsigned) representing the number of color planes. Should be '1' in decimal. |
| | BitsPerPixel | 2 bytes | An integer (unsigned) representing the number of bits a pixel takes to represent a color. |
| | Compression | 4 bytes | An integer (unsigned) representing the value of compression to use. Should be '0' in decimal. |
| | ImageSize | 4 bytes | An integer (unsigned) representing the final size of the compressed image. Should be '0' in decimal. |
| | XpixelsPerMeter | 4 bytes | An integer (signed). Should be set to '0' in decimal to indicate no preference of the target device. |
| | YpixelsPerMeter | 4 bytes | An integer (signed). Should be set to '0' in decimal to indicate no preference of the target device. |
| | TotalColors | 4 bytes | An integer (unsigned) representing the number of colors in the color pallet. |
| | ImportantColors | 4 bytes | An integer (unsigned) representing the number of important colors. Ignored by setting '0' in decimal. |
| **COLOR TABLE**<br>**Fields:** 4 x entries<br>**Width:** 4 x entries | Red | 1 bytes | An integer (unsigned) representing Red color channel intensity. |
| | Green | 1 bytes | An integer (unsigned) representing Green color channel intensity. |
| | Blue | 1 bytes | An integer (unsigned) representing Blue color channel intensity. |
| | Reserved | 1 bytes | An integer (unsigned) reserved for other uses. Should be set to '0' in decimal |
| **PIXEL DATA** | | | An array of pixel values with padding bytes. A pixel value defines the color of the pixel. |

semi-optional

1. BMP Image Format

VGA Hardware Components

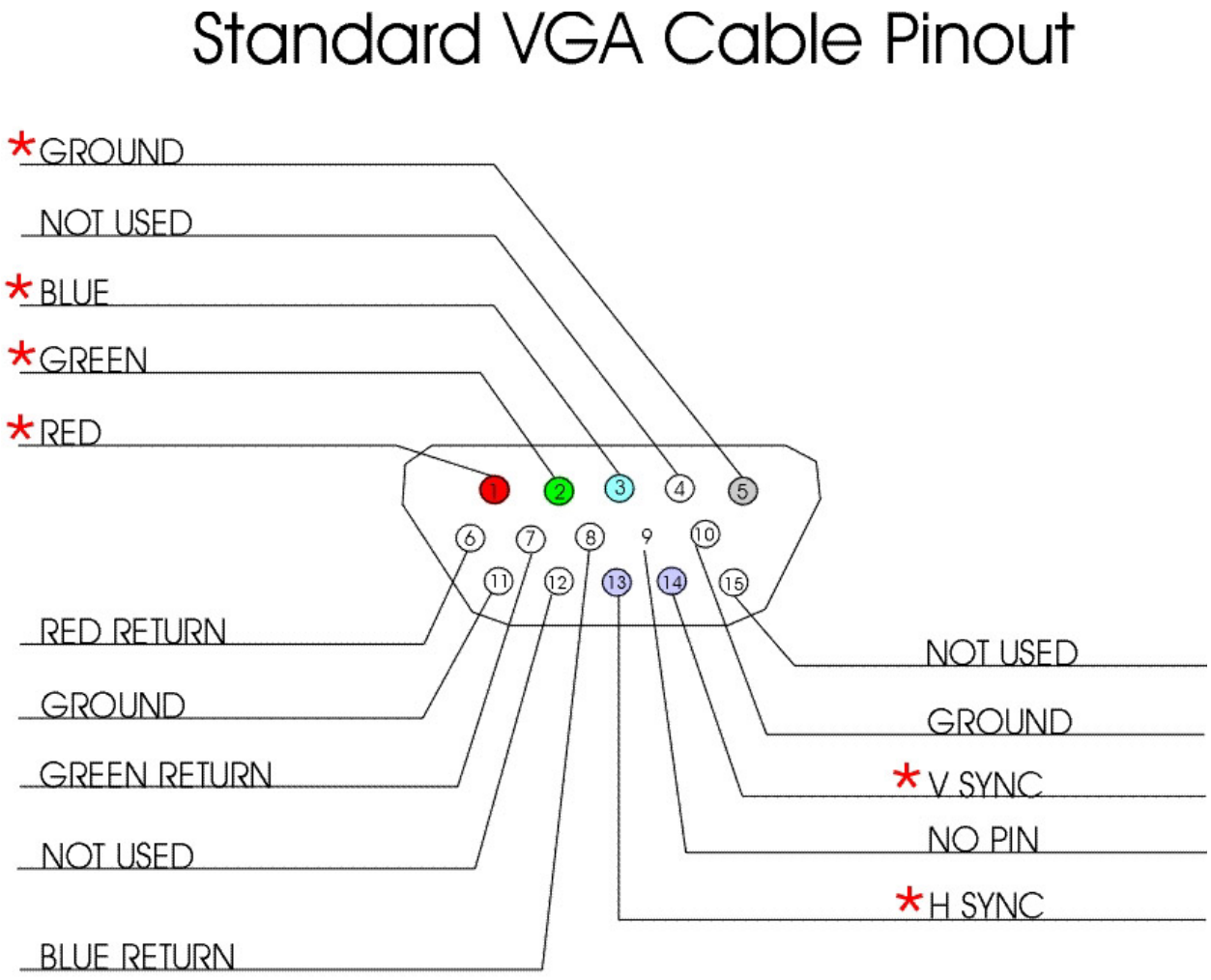
The VGA Design flow begins with the pixel buffer dma, the pixel buffer here the image is in the form of a 320x240 16-bit RGB. The 16-bit RBG format consists of 5 red bits, 6 green bits, and 5 blue bits in the form: RRRRR:GGGGGG:BBBBB. The buffer then sends the image to the RGB resampler, which will take a 16Bit RGB input, and output it as a 30Bit RGB. The resampler then connects to the scaler which converts the image to 640x480. This resolution is the native resolution of the VGA communication, so this can then be outputted to the monitor.




2. Hardware for VGA Components

VGA Communication

VGA uses I2C communication in order to display images to the monitor. This communication uses a serial clock and serial data pins. The 15 pin VGA port consists of as seen below, a red green blue, 3 ID bits, and red green and blue shield pins.




Needed pins for VGA - RGB Cable conversion 1,2,3,5,13,14

3. VGA Pinout

Interrupt Generation

In order to generate an interrupt, there is three main steps that must be accomplished. First is the initialization. Here we set the edge capture pointer, the IRQ Mask, reset the edge capture register, and register the ISR. Setting the IRQ mask is done with the IOWR_ALTERA_AVALON_PIO_IRQ_MASK function. This function takes two arguments, the base address and the mask. So for the keys, using the base address and 0b1100 will enable key 3 and 2. Registering the IRQ is also important which is set with alt_irq_register function

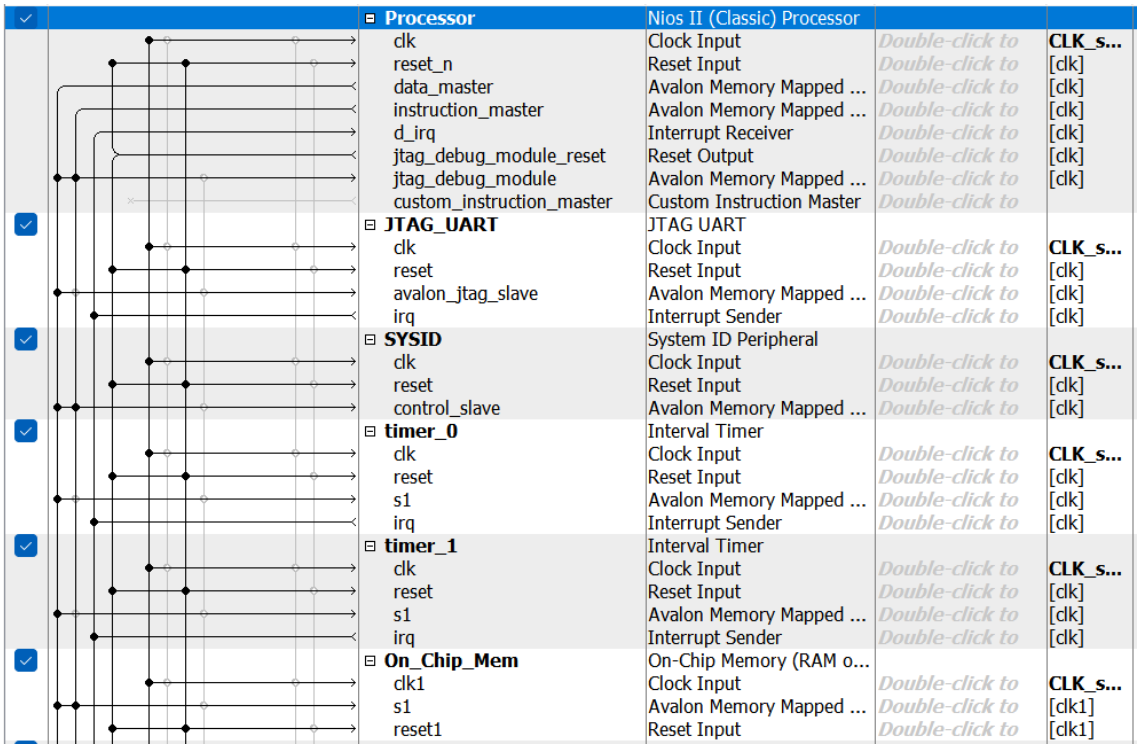
which takes the IRQ, edge capture pointer, and the function name that we are trying to call with the interrupt. We then have the interrupt handler, and the function for the ISR program.

## Project Overview

For this project, we will be implementing many photo manipulation techniques, reading data from an SD card, as well as displaying images to a monitor via the VGA port. The first part of this project is to be able to read a bitmap image from the SD card. After this step is successful, we can create the program to display the image to the monitor. Once these are working properly, we can begin implementing our photo manipulation techniques. These manipulations are, increasing the brightness of the image, color shifting, image mirroring, wave, red/blue/green isolation, next image, and redisplay. Brightness is simple, it simply makes the image brighter. Color shifting will make the red values in the photo blue, the blue values green, and the green values red. The next shift will then make the red values green, the green values blue, and the blue values red. Image mirroring will display the image upside down. The wave manipulation will offset each row by a pixel in order to create an interesting effect. The color isolation will only display one of the LED values. Lastly, next image will load the next image on the SD, and redisplay is used to redisplay the image to the monitor this will be used to visualize manipulations.

## Project Procedure

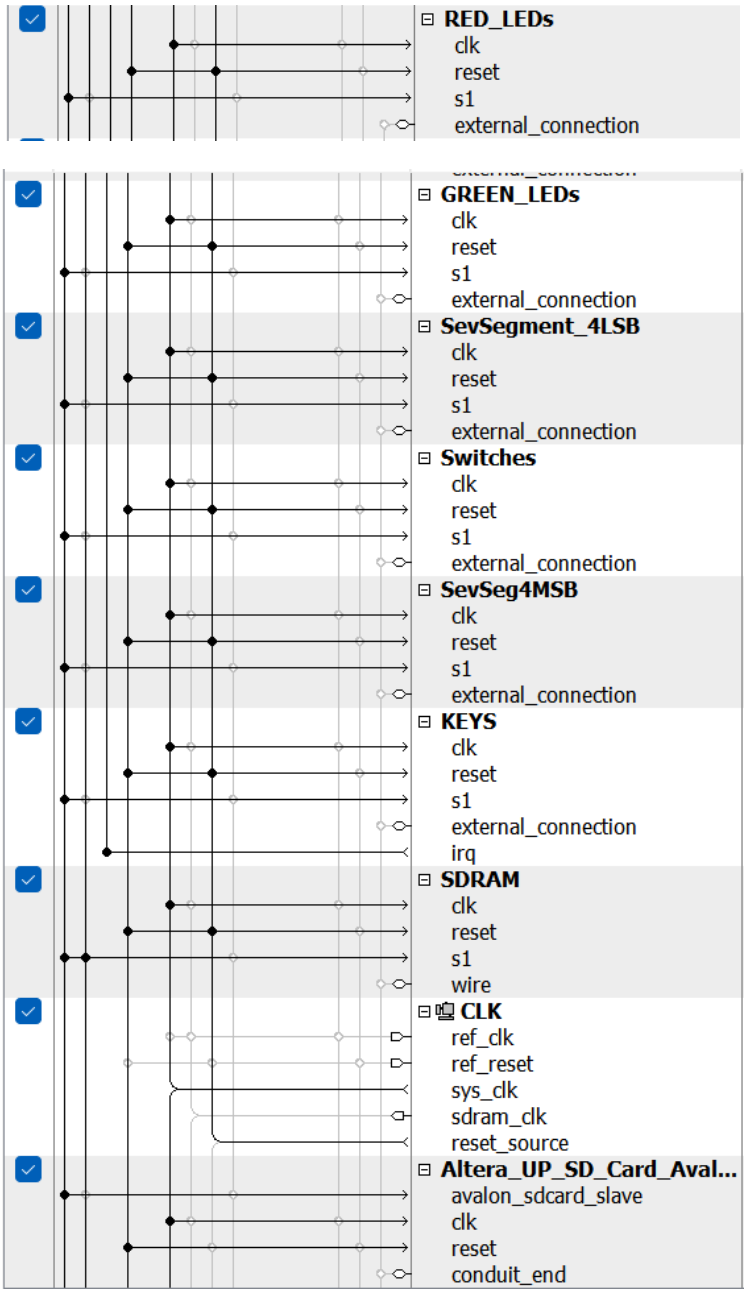
To begin this project, we create the Qsys system. This Qsys system is fairly similar to what has been seen in many past labs. We have the processor, JTAG, System ID, Timers, and on chip memory peripherals. These peripherals can be seen in the figure below.



| | | | |
|---|---|---|---|
| **Processor** | Nios II (Classic) Processor | | |
| clk | Clock Input | *Double-click to* | **CLK_s...** |
| reset_n | Reset Input | *Double-click to* | [clk] |
| data_master | Avalon Memory Mapped ... | *Double-click to* | [clk] |
| instruction_master | Avalon Memory Mapped ... | *Double-click to* | [clk] |
| d_irq | Interrupt Receiver | *Double-click to* | [clk] |
| jtag_debug_module_reset | Reset Output | *Double-click to* | [clk] |
| jtag_debug_module | Avalon Memory Mapped ... | *Double-click to* | [clk] |
| custom_instruction_master | Custom Instruction Master | *Double-click to* | |
| **JTAG_UART** | JTAG UART | | |
| clk | Clock Input | *Double-click to* | **CLK_s...** |
| reset | Reset Input | *Double-click to* | [clk] |
| avalon_jtag_slave | Avalon Memory Mapped ... | *Double-click to* | [clk] |
| irq | Interrupt Sender | *Double-click to* | [clk] |
| **SYSID** | System ID Peripheral | | |
| clk | Clock Input | *Double-click to* | **CLK_s...** |
| reset | Reset Input | *Double-click to* | [clk] |
| control_slave | Avalon Memory Mapped ... | *Double-click to* | [clk] |
| **timer_0** | Interval Timer | | |
| clk | Clock Input | *Double-click to* | **CLK_s...** |
| reset | Reset Input | *Double-click to* | [clk] |
| s1 | Avalon Memory Mapped ... | *Double-click to* | [clk] |
| irq | Interrupt Sender | *Double-click to* | [clk] |
| **timer_1** | Interval Timer | | |
| clk | Clock Input | *Double-click to* | **CLK_s...** |
| reset | Reset Input | *Double-click to* | [clk] |
| s1 | Avalon Memory Mapped ... | *Double-click to* | [clk] |
| irq | Interrupt Sender | *Double-click to* | [clk] |
| **On_Chip_Mem** | On-Chip Memory (RAM o... | | |
| clk1 | Clock Input | *Double-click to* | **CLK_s...** |
| s1 | Avalon Memory Mapped ... | *Double-click to* | [clk1] |
| reset1 | Reset Input | *Double-click to* | [clk1] |

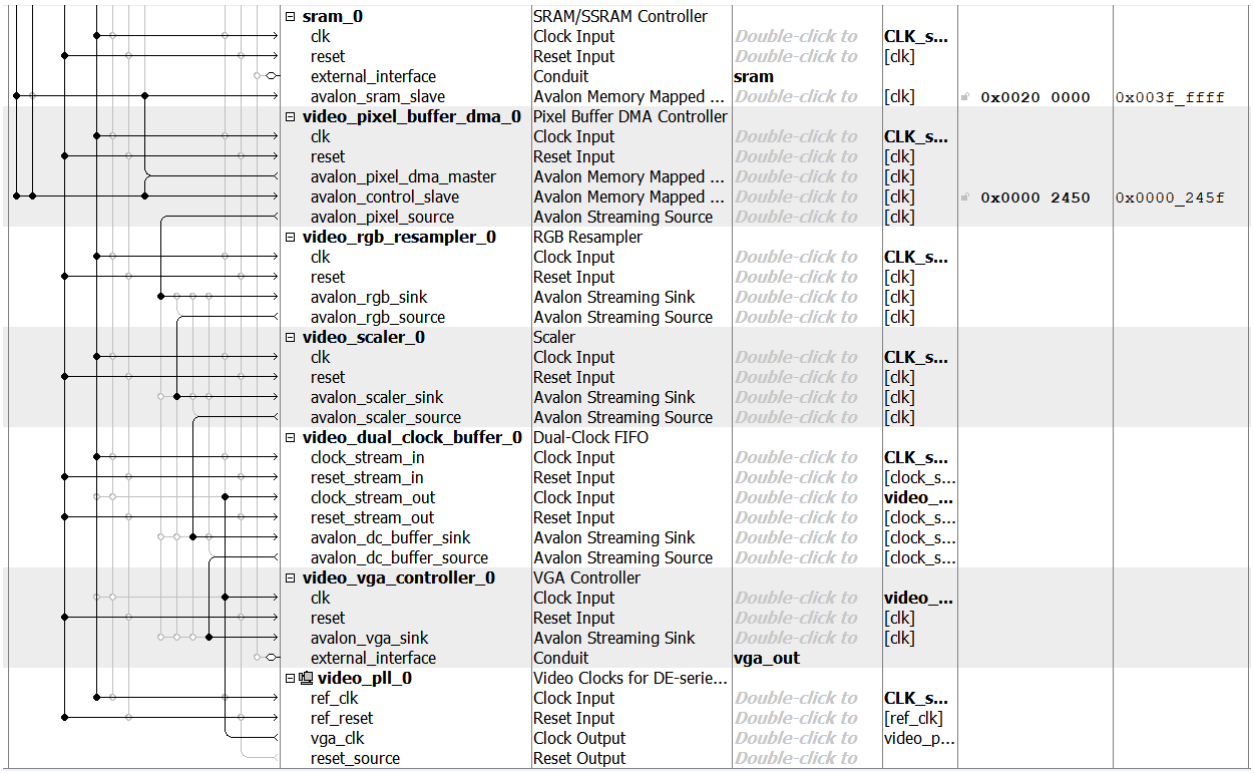
4. Main System Peripherals

After the main peripherals seen above we have our I/O peripherals which include the Red Leds, green leds, seven segment display, switches, pushbuttons. Along with the I/O peripherals we have the SD card, as well as the clock.




5. Qsys I/O Peripherals and SD Card

The final peripherals that need to be added to Qsys is the hardware for the VGA communication. These include SDRAM, video pixel buffer dma, RGB resampler, scaler, dual clock buffer, VGA controller, and the video pll. These are connected in a cascading fashion and the actual connections can be seen below.




6. VGA Communication in Qsys

The SRAM controller is used to address the pixel bugger and communicate with the rest of the VGA system. Here, the input bitmap image is a 24-bit RGB, 320x240 resolution. Our resolution to the monitor must be a 30-bit RGB with a resolution of 640x480. To get there, we input 16-bit RGB photo with a 16bit RGB to the buffer, the resampler then converts the 16-bit RGB to 30-Bit RGB. The next main step in this is the scaler. For the scaler, the resolution is doubled, and the output is finally a 30-bit RGB with a resolution of 640x480. After generating the HDL for the system, we use the instantiation in Appendix A, to instantiate the system. After creating the system, we create a new project in Nios2 Software Build tools for Eclipse. We can then begin coding. The overall code can be seen in Appendix B project code, but first, the code for opening and reading a file from the SD Card. The important sections for this process can be seen below.

```
device_reference = alt_up_sd_card_open_dev(ALTERA_UP_SD_CARD_AVALON_INTERFACE_0_NAME);
if (device_reference != NULL)
{
    if ((connected == 0) && (alt_up_sd_card_is_Present()))
    {
        if (alt_up_sd_card_is_FAT16())
        {
            if (FileSelect == 0){
                fileHandle = alt_up_sd_card_fopen("smile.bmp",false);
            }
            else if (FileSelect == 1){
                fileHandle = alt_up_sd_card_fopen("tayeb2.bmp",false);
            }
            else if (FileSelect == 2){
                fileHandle = alt_up_sd_card_fopen("buko.bmp",false);
            }
            att = alt_up_sd_card_get_attributes(fileHandle);

            VGA_box(0,0, 321, 241, 0xFFFF);

            for (j=0; j<54; j++) //gets past header
            {
                att1 = alt_up_sd_card_read(fileHandle);
            }
            i = 0, j = 0;
            printf("File handle: %i\n", fileHandle);
            for (i = height; i >= 0; i = i-1){
                for (j = 0; j < width; j = j+1){
                    att1 = (unsigned char)alt_up_sd_card_read(fileHandle);
                    att2 = (unsigned char)alt_up_sd_card_read(fileHandle);
                    att3 = (unsigned char)alt_up_sd_card_read(fileHandle);
                    pixel = ((att3>>3)<<11) | ((att2>>2)<<5) | (att1 >> 3);
                    //pixel = ((att3)<<16) + ((att2)<<8) + (att1);
                VGA_box(j,i,j,i,pixel);
                }
            }
            if ((Value & 0xF000) == 0x1000){
                        invertPhoto();
            }
            alt_up_sd_card_fclose(fileHandle);
        }
        else
        {
        printf("Unknown file system.\n");
        }

        connected = 1;
    }
    else if ((connected == 1) && (alt_up_sd_card_is_Present() == false))
    {
        printf("Card disconnected.\n");
        connected = 0;
    }
```

7. Important Section of SD Card Implmemntations

This program begins by using the alt_up_sd_card_open_dev command with the sd interface name as its parameter to set the device reference. Next, everything is pretty straight forward, the program ensures the card is FAT 16, then opens the file depending on which file select is being

used. This will be discussed more in detail later. The program whites the screen out, using the VGA Box function, which will also be discussed more in detail, and then reads the file handle three bytes at a time. In doing this, each pixel is printed to the monitor using the VGAbox function, the important code for this function can be seen below.

```
void VGA_box(int x1, int y1, int x2, int y2, int Color){
    int offset, row, col, StateFlag, NewValue;
    int R, G, B;
    StateFlag = Value & 0xF00; //Get State Flag
    NewValue = Value & 0x0001F; //Remask Value

    volatile short *pixel_buffer = (short *) 0x00200000;
    for (row = y1; row <= y2; row++){
        col = x1;
        while (col <= x2){
            offset = (row << 9) + col;
            *(pixel_buffer + offset) = Color;
            ++col;
        }

    }
}
```

8. Important sections of VGA_box Function

Disregarding the manipulations that occur within this block, we can see that the VGA_box function has an x1, y1, x2, t2, and color as inputs. We then ensure the pointer to the pixel buffer is the correct address and matches with that of the one in Qsys, lastly, we iterate through each row and column sending the 16-bit RGB color integer to the pixel buffer. This will then use the hardware to output that pixel to the monitor. Next, we can analyze how the photo manipulation selectors were set up. First the ISR for key0.

```
void key0_isr(){
    //Brightness, Increase Pixel Color by Switch Value
    printf("KEY 0");
    Value = IORD(SWITCHES_BASE, 0);
    if ((Value & 0b100000000000000000) > 0){
        //If Value of 17 is high, Width Adjustment go crazy
        Value = Value & 0x001F;
        Value = Value + 0x900;
    }
    else if ((Value & 0b010000000000000000) > 0){ //Switch 16 is high, Mirror the image up and down
        Value = Value & 0x001F;
        Value = Value + 0x1000;
    }
    else if ((Value & 0b001000000000000000) > 0) { //Switch 15 is high brightness changes
    Value = Value & (0x001F);
    Value = Value + (0x100); //Add the State Flag
    }
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
    IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    return;
}
```

9. Key0_isr functon

This is one of our major selector pushbuttons. Value is a global integer that is used throughout the project to perform manipulations. So, here, we begin by reading the switches, if 17 is high, we adjust the width by the value of the first 5 switches this is done by using the state flag 0x900, and masking the first five switch values. If key 16 high, we use the flag 0x1000, and we will delve into what this does later. Brightness works similarly to the width adjustment, however, the state flag is 0x100 for changing brightness. There are more elements that can be controlled, we will look at Key1's ISR to explore this some more.

```
void key1_isr(){
    //INVERT RGB
    printf("KEY 2\n");
    if (Value == 0x200){
        Value = 0x300;
    }
    else if(Value == 0x300){
        Value = 0x000;
    }
    else Value = (0x200);

    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
    IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    return;
}
```

10. Key1_ISR function

Key1's ISR inverts the RGB values. What this means is that every time the key is pressed the RGB colors shift. So, the first press, the red values become blue, blue becomes green and green becomes red. This process is accomplished by implementing more state flags within value. So for the red values to become blue, one press is required, and the state will become 2, next shift will be state 3, and it will then return to normal with a state of 0. This is very similar to the Key2_isr function which can be seen below.

```
void key2_isr(){
    //Just Red Values, Just Blue Values, Just Green Values
    if (Value == 0x400){
        Value = 0x500;
    }
    else if (Value == 0x500){
        Value = 0x600;
    }
    else Value = 0x400;

    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
    IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    return;
}
```

11. Key2_isr function

As the description of key1, key2 is very similar, however these states correspond to being a monochromatic, either just red, just blue or just green values. For our last ISR, we have the ISR for key3 which has only two options.

```
void key3_isr(){
    // Reload Image if switch 1 is high change to next photo
    int Sw = IORD(SWITCHES_BASE, 0);
    if ((Sw & 0x1) == 1){
        if (FileSelect == 0){
            FileSelect = 1;
        }
        else if (FileSelect == 1){
            FileSelect = 2;
        }
        else FileSelect = 0;
    }
    printf("KEY 3");
    initCard();
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
    IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
    return;
}
```

12. Key3_isr Function

Key 3 has two possibilities, it will either just reload the current image, or change the image to the next file. It changes which file is being used by changing the file select variable, and will always call the function to redisplay the image. Next, in order to implement each of the photo manipulations, the following techniques were used.

Brightness:

```
case(1):
        R = R + (NewValue << 11);
        G = (Color & 0b0000011111100000) + (NewValue << 6);
        B = B + NewValue;
        if (R >= 0b1000000000000000){
            R = 0b11111 << 11;
        }
        if (G >= 0b0000100000000000){
            G = 0b111111 << 5;
        }
        if (B >= 0b100000){
            B = 0b11111;
        }
        Color = R | G | B;
        break;
```

13. Code for Manipulating Brightness

For Brightness, we add the first 5 bits of the value variable which is loaded into “newValue” to each of the colors. Before updating the color, and printing the new color to the desired pixel, this will make the pixels more and more white until the photo is completely white.

Width Adjustment:

```
if ((Value & 0xF00) == 0x900){
    width = width + (Value & 0x0FF);
}
```

14. Code For Width Adjustment

Simply, the code for the width adjustment does exactly that, it changes the width of which the image is read to be the width plus the value of the first 5 switches.

Image Inversion:

Image inversion begins by first displaying the image normally, then displaying upside down. This is accomplished with the code below.

```
    i = 0, j = 0;
    printf("File handle: %i\n", fileHandle);
    for (i = 0; i <= 240; i = i+1){
        for (j = 0; j < width; j = j+1){
            att1 = (unsigned char)alt_up_sd_card_read(fileHandle);
            att2 = (unsigned char)alt_up_sd_card_read(fileHandle);
            att3 = (unsigned char)alt_up_sd_card_read(fileHandle);
            pixel = ((att3>>3)<<11) | ((att2>>2)<<5) | (att1 >> 3);
            //pixel = ((att3)<<16) + ((att2)<<8) + (att1);
        VGA_box(j,i,j,i,pixel);
        }
    }
    alt_up_sd_card_fclose(fileHandle);
}
```

15. Image Inversion Code

The image inversion code is essentially the regular display code, however instead of displaying from bottom to top, we display from top to bottom.

RGB color Shift:

```
case(2): //Red becomes Blue, Blue Becomes Green, Green Becomes Red
        Color = ((R >> 11) | (G << 5) | (B << 6));
        break;
case(3): //Red becomes Green, Green becomes Blue, Blue becomes Red
        Color = ((R >> 5) | (G >> 5) | (B << 11));
        break;
```

16. RGB Color Shift Code

The color shifting code is very straightforward, after having a 16-bit RGB value, where R is the masked 5 bits, green is the 5 most significant bits of the Green value, and Blue is the 5 final bits. This shifts the color by shifting each of the bits to their new location and oring them together. For example for the Red becoming blue, blue to green and green to red, the red is shifted right 11, green left 5, and blue left 6, so the LSB of green is ignored. All of these values are Ord together and create the new 16-bit color value. Lastly is the monochromatic images.

Monochromatic Images:

```
case(4):
        Color = Color & 0b1111100000000000;
        break;
case(5):
        Color = Color & 0b0000011111100000;
        break;
case(6):
        Color = Color & 0b0000000000011111;
        break;
default:
```

17. Code for Monochromatic Images

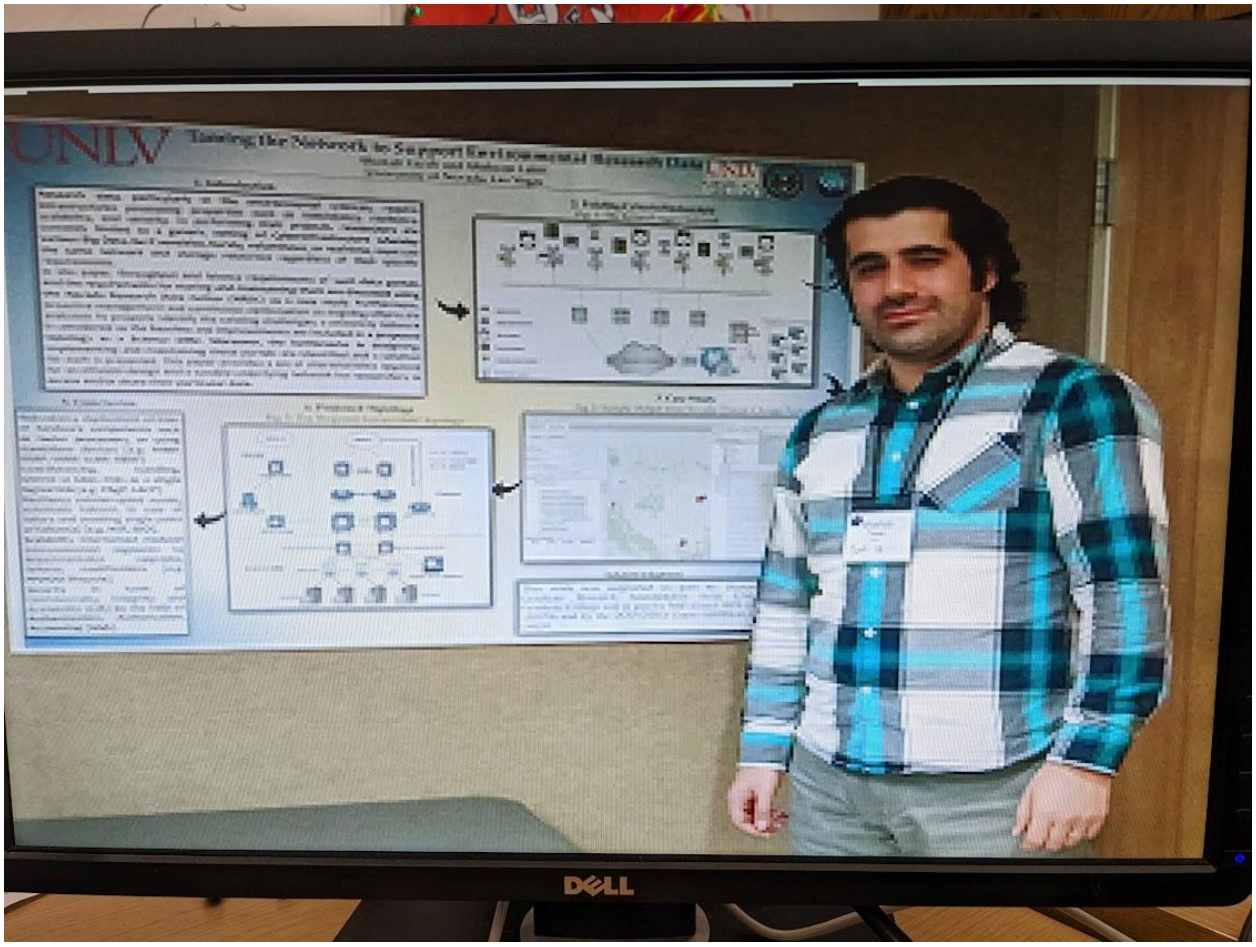
The monochromatic images are very straightforward, for example, state 4 simply masks the color value to only be the red values, which exist in the first 5 bits. The next state is just the green value which masks the color to be just the 6 green btis.

## Data Analysis:

For starters, when we set up and run the program the photo displayed on the monitor is the default photo with no manipulations, this can be seen below.


18. Photo 1 Displayed on Monitor

This is as expected, as the main program calls the function to initialize the SD card and display the image, since no changes have been made yet, the default photo is displayed. After this photo is displayed we can observe the function for switching photos, after hitting Key3 with sw0 high, we can see the next photo on the SD Card below.




19. Second Photo Displayed to Monitor

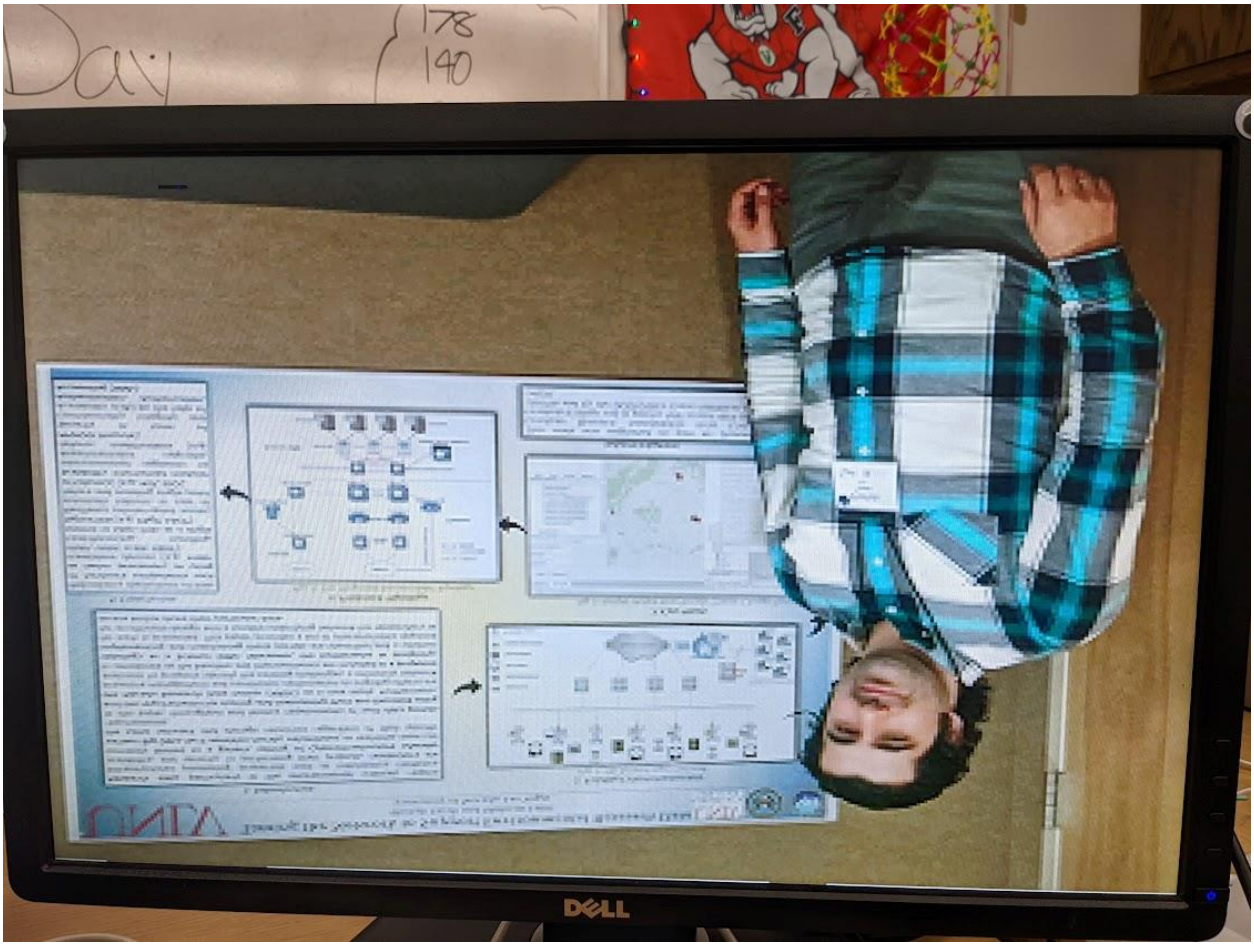
Next, after doing this operation one more time, we can see the third, and last photo that is stored on the SD Card with no modifications.


20. Third Photo Displayed to monitor

We can then see the brightness feature. This feature is more apparent when comparing the photo one figure to the figure seen below. We trigger this manipulation with key 0 and sw16 high with the sw0-5 being the adjustment switches. After pressing key3, and redisplaying the image we can see the image below.

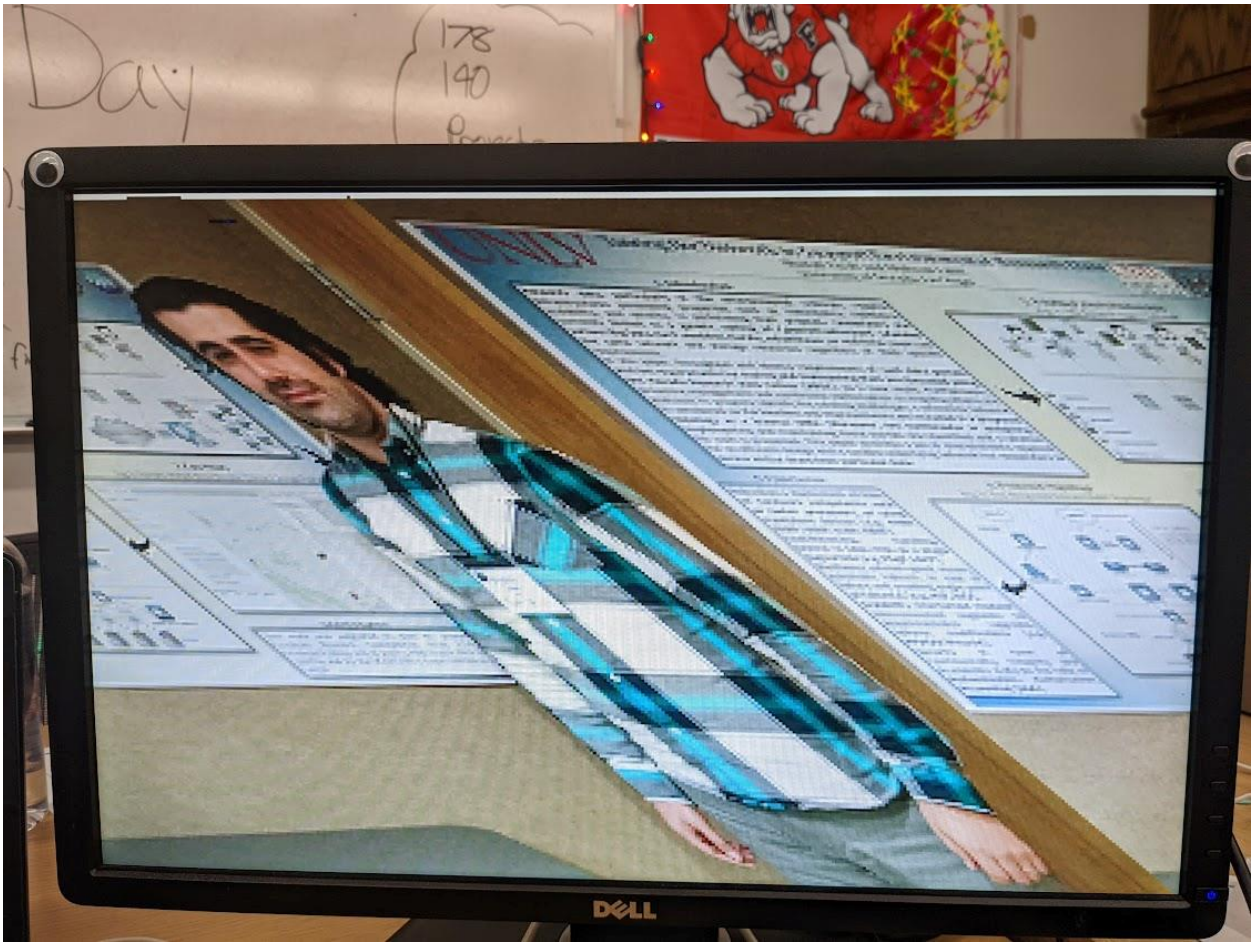

21. Brightness adjustment on photo 1

Another of our photo manipulation techniques is the image inversion, this can be seen below inverting the second photo. The image was first displayed right side up before the screen cleared and the image was displayed upside down. This was as expected and how we intended the program to display the inversion manipulation.




22. Image Inversion on Photo 2

Next, the wave, or width manipulation comes out with a very interesting outcome. With only a Sw0 of high, only 1 pixel shift a row, the following photo is displayed after manipulation. This manipulation was successful, as the image has a left skew.




23. Width Manipulation on Photo 2

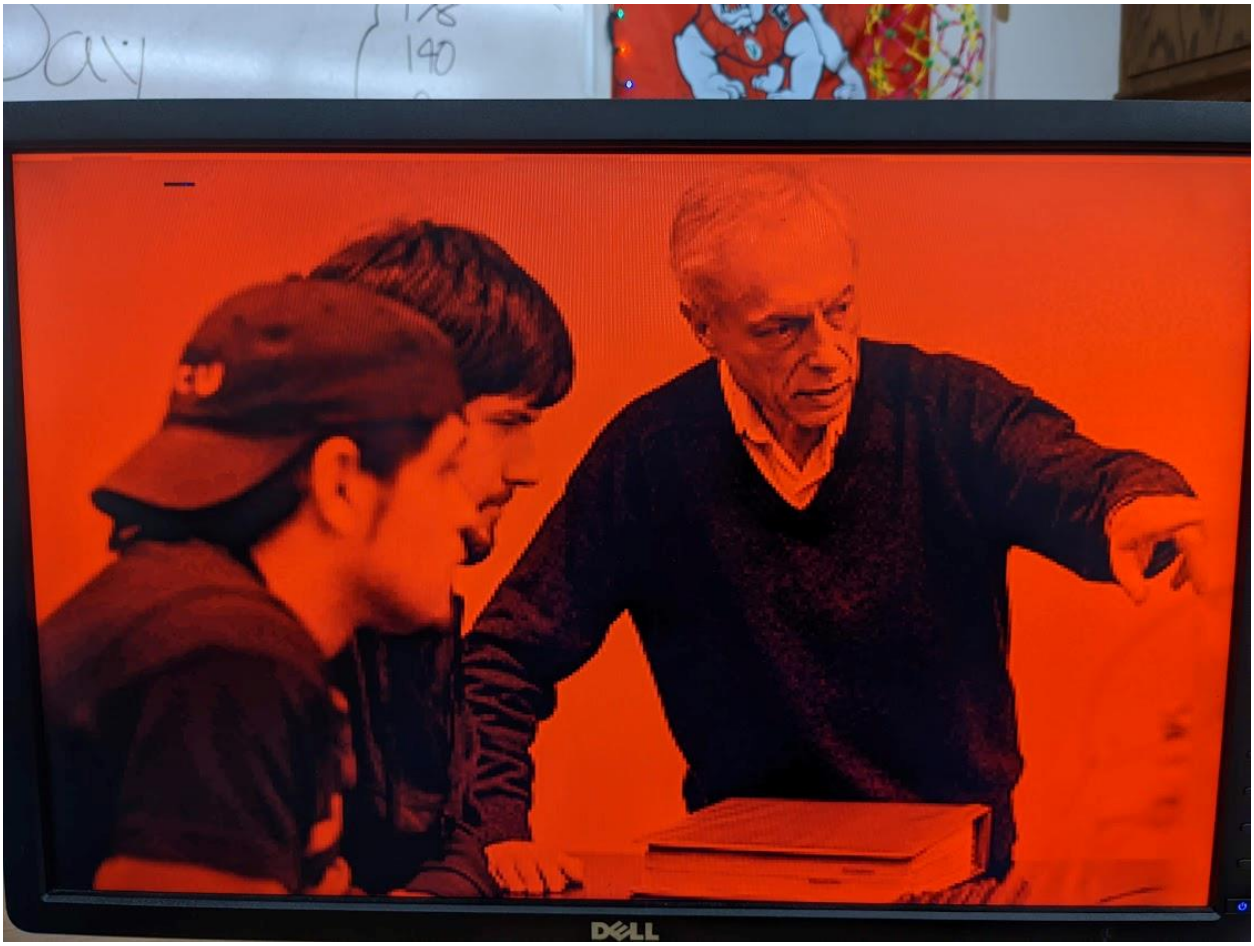
Next, we have the key 1 which implements the color shift. This color shift can be seen in the following figure, this particular manipulation is red becoming blue, blue to green, and green to red. We can see this is a correct manipulation as the students shirt that was previously green, is now a red.


24. Color Shfiting, RGB to GBR

We can then examine the next color shift, which is red green blue to blue red green. The shift can be seen in the figure below. This photo manipulation also worked as it was intended to.

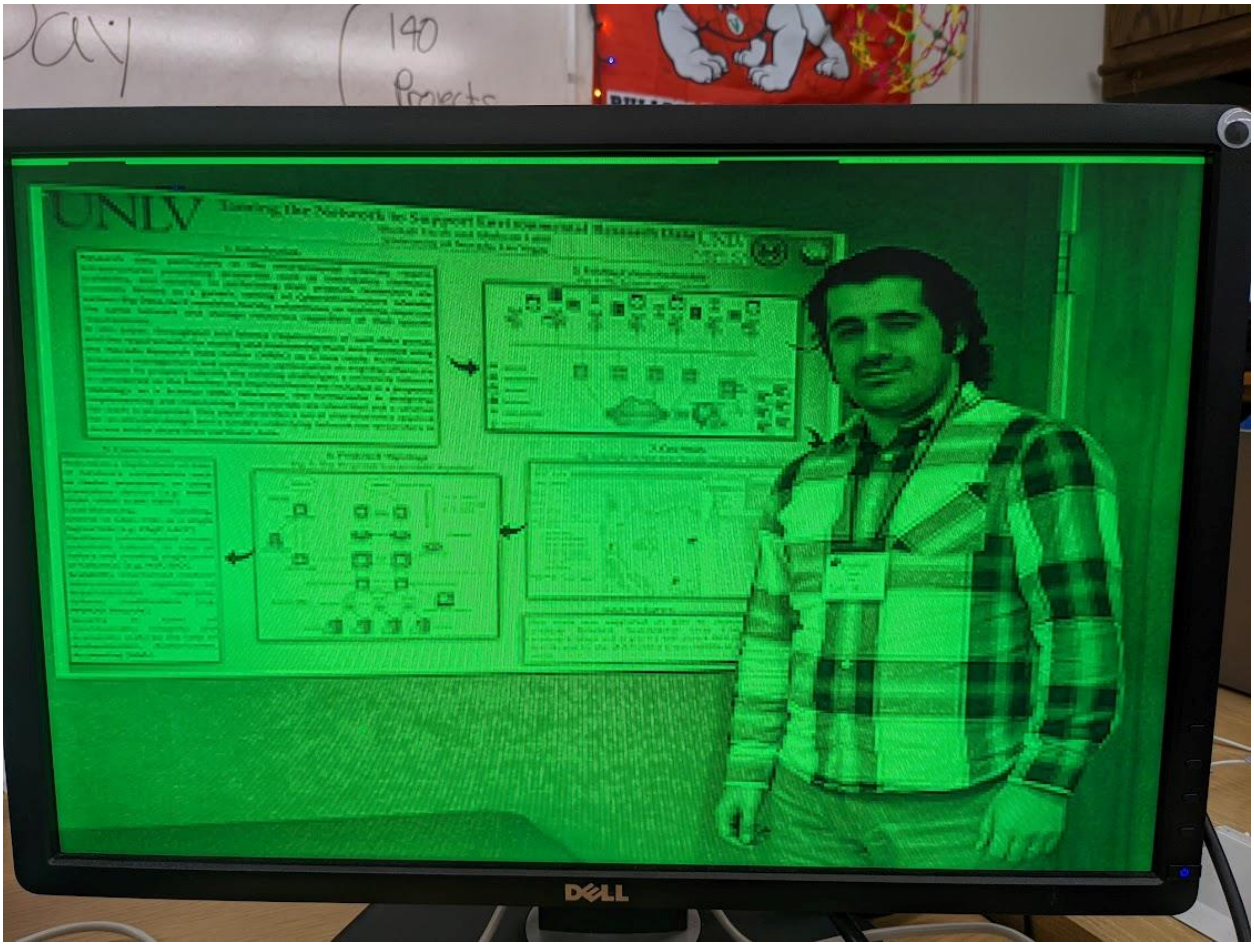

25. Color Shifting, RGB to BRG

The last Photo Manipulation technique that we will be examining is the color isolation. Here we are first manipulating Photo 3 and isolating just the red LEDs. This works, as the color is now rescaled. Simply the same image with just the red values present.



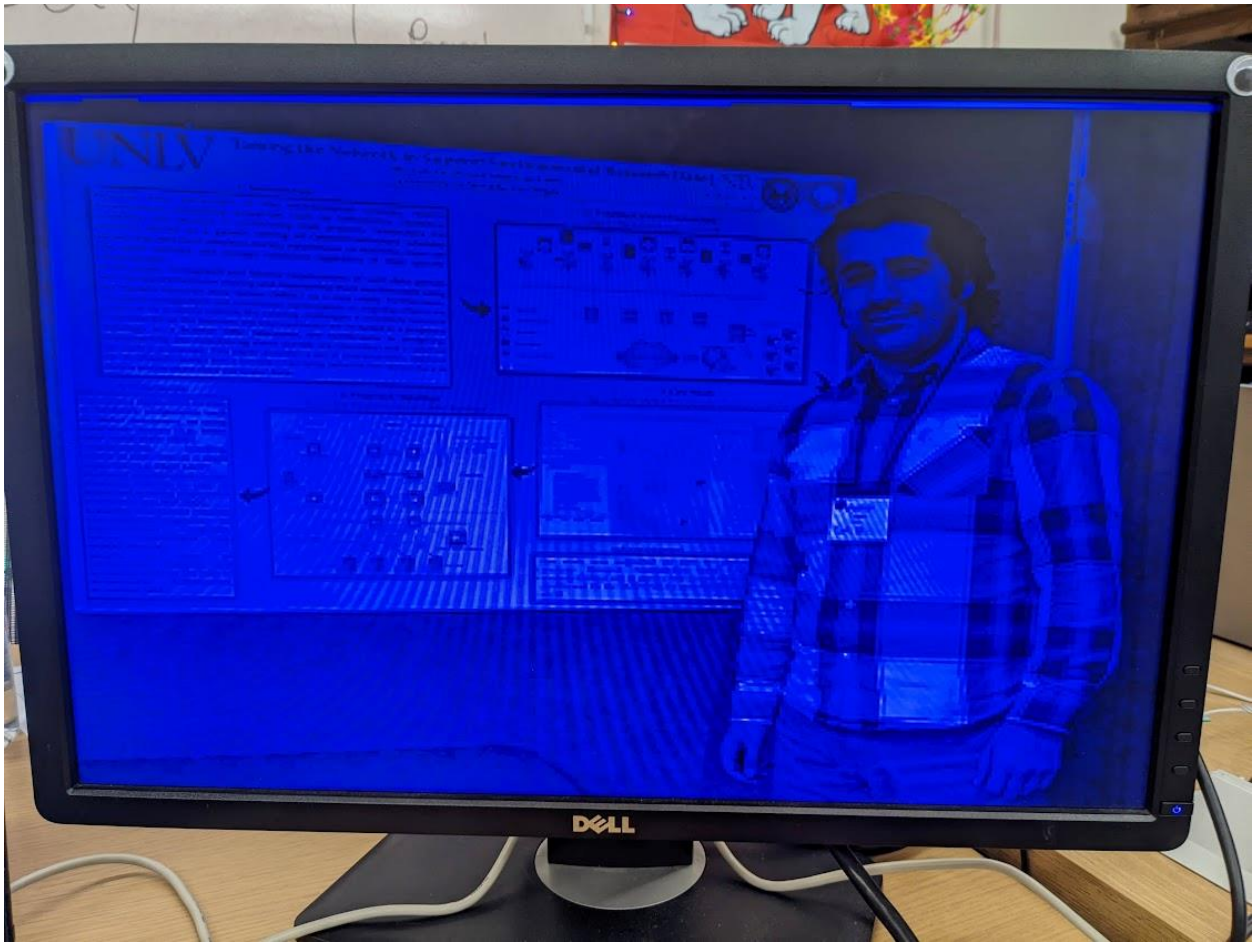

26. Red Color Isolation on Photo 3

Next, by pressing key2 again we turn to green isolation, we also change the photo back to photo 2 during this manipulation and the green color isolation can be seen below.




27. Green Color Isolation on Photo 2

Lastly, we have the blue color isolation, which we also perform on photo 2. This can be seen below, and with the color isolation working correctly, we can now confidently say that the manipulation techniques were successfully implemented and work correctly.




28. Blue Color Isolation on Photo 3

## Conclusion

This project was essential in moving forward with knowledge in embedded systems development. It allowed for me to use the knowledge I have gained over the semester, as well as expand on this knowledge into new peripherals. Along with peripherals we already had prior knowledge we gained new knowledge in how to establish a VGA port, SD Card implementation with SPI, and different photo manipulation techniques. With this, we have been able to take everything learned compiled into one big project representing everything learned over the semester, and expanding knowledge for future career opportunities.

## Appendix A (Top Level Instantiation)

```
module FinalProject (CLOCK_50, SW, KEY, LEDR, LEDG, DRAM_CLK,
SevMSB, SevLSB, sdram_wire_addr, sdram_wire_ba,

sdram_wire_cas_n, sdram_wire_cke, sdram_wire_cs_n,
sdram_wire_dq, sdram_wire_dqm, sdram_wire_ras_n,
sdram_wire_we_n,

SD_CMD, SD_CLK, SD_DAT, SD_DAT3, sram_DQ, sram_ADDR, sram_LB_N,
sram_UB_N,

sram_CE_N, sram_OE_N, sram_WE_N, vga_out_CLK, vga_out_HS,
vga_out_VS, vga_out_BLANK, vga_out_SYNC, vga_out_R,vga_out_G,

vga_out_B);


input CLOCK_50;

input [17:0] SW;

input [3:0] KEY;

output [7:0] LEDG;

output [17:0] LEDR;

output [31:0] SevMSB;

output [31:0] SevLSB;

output [12:0] sdram_wire_addr;

output [1:0]  sdram_wire_ba;

output sdram_wire_cas_n;

output sdram_wire_cke;

output sdram_wire_cs_n;

inout  [31:0] sdram_wire_dq;

output DRAM_CLK;

output [3:0] sdram_wire_dqm;

output sdram_wire_ras_n;
```

```
output sdram_wire_we_n;
output SD_CLK;
inout SD_CMD;
inout SD_DAT;
inout SD_DAT3;
inout [15:0] sram_DQ;
output [19:0] sram_ADDR;
output sram_LB_N;
output sram_UB_N;
output sram_CE_N;
output sram_OE_N;
output sram_WE_N;
output vga_out_CLK;
output vga_out_HS;
output vga_out_VS;
output vga_out_BLANK;
output vga_out_SYNC;
output [7:0]  vga_out_R;
output [7:0]  vga_out_G;
output [7:0]  vga_out_B;

QsysSystem Softcore (
.clk_clk(CLOCK_50),
.reset_reset(),
.green_leds_external_connection_export(LEDG),
.red_leds_external_connection_export(LEDR),
.switches_external_connection_export(SW),
.sevseg4msb_external_connection_export(SevMSB),
```

```
.sevsegment_4lsb_external_connection_export(SevLSB),
.keys_external_connection_export(KEY),
.sdram_wire_addr(sdram_wire_addr),
.sdram_wire_ba(sdram_wire_ba),
.sdram_wire_cas_n(sdram_wire_cas_n),
.sdram_wire_cke(sdram_wire_cke),
.sdram_wire_cs_n(sdram_wire_cs_n),
.sdram_wire_dq(sdram_wire_dq),
.sdram_wire_dqm(sdram_wire_dqm),
.sdram_wire_ras_n(sdram_wire_ras_n),
.sdram_wire_we_n(sdram_wire_we_n),
.sd_b_SD_cmd(SD_CMD),
.sd_b_SD_dat(SD_DAT),
.sd_b_SD_dat3(SD_DAT3),
.sd_o_SD_clock(SD_CLK),
.sram_DQ(sram_DQ),
.sram_ADDR(sram_ADDR),
.sram_LB_N(SRAM_LB_N),
.sram_UB_N(sram_UB_N),
.sram_CE_N(sram_CE_N),
.sram_OE_N(sram_OE_N),
.sram_WE_N(sram_WE_N),
.vga_out_CLK(vga_out_CLK),
.vga_out_HS(vga_out_HS),
.vga_out_VS(vga_out_VS),
.vga_out_BLANK(vga_out_BLANK),
.vga_out_SYNC(vga_out_SYNC),
.vga_out_R(vga_out_R),
```

```
.vga_out_G(vga_out_G),
.vga_out_B(vga_out_B),
.sdram_clk_clk(DRAM_CLK));

endmodule
```

## Appendix B (Project Code)

```
#include <stdio.h>
#include <system.h>
#include "sys/alt_stdio.h"
#include <io.h>
#include <altera_up_sd_card_avalon_interface.h>
#include <altera_up_avalon_video_pixel_buffer_dma.h>
#include <sys/alt_irq.h>
#include <altera_avalon_timer_regs.h>
#include "altera_avalon_pio_regs.h"



volatile int i, j, Value;
void pio_init();
void initCard();
void VGA();
void VGA_box(int, int, int, int, int);
void handle_key_interrupts(void*);
void invertPhoto();
int FileSelect = 0;

int main()
{
  printf("Hello from Nios II!\n");
  pio_init();
  VGA();
  initCard();
  printf("out\n");
  while (1){
  }
  return 0;
}

void VGA(){
     printf("VGALOOP");
     VGA_box(0,0, 321, 241, 0x8000);
     //VGA_box(0, 0, 100, 100, 0xF011);
     //VGA_box(20, 20, 300, 200, 0x4002);
     //VGA_box(30,30,600, 1079, 0x4002);

}

void initCard()
     {
```

```
          short att1 = 0, att2 = 0, att3 = 0, att;
          int width = 320;
          int height = 240;
          int pixel;
         alt_up_sd_card_dev *device_reference = NULL;
         int connected = 0;
         short int fileHandle;

         if ((Value & 0xF00) == 0x900){
          width = width + (Value & 0x0FF);
         }

         device_reference =
alt_up_sd_card_open_dev(ALTERA_UP_SD_CARD_AVALON_INTERFACE_0_NAM
E);
         if (device_reference != NULL)
         {
              if ((connected == 0) &&
(alt_up_sd_card_is_Present()))
              {
                  if (alt_up_sd_card_is_FAT16())
                  {
                    if (FileSelect == 0){
                         fileHandle =
alt_up_sd_card_fopen("smile.bmp",false);
                    }
                    else if (FileSelect == 1){
                         fileHandle =
alt_up_sd_card_fopen("tayeb2.bmp",false);
                    }
                    else if (FileSelect == 2){
                              fileHandle =
alt_up_sd_card_fopen("buko.bmp",false);
                    }
                       att =
alt_up_sd_card_get_attributes(fileHandle);

                       VGA_box(0,0, 321, 241, 0xFFFF);

                       for (j=0; j<54; j++) //gets past header
                       {
                         att1 = alt_up_sd_card_read(fileHandle);
                       }
                       i = 0, j = 0;
                       printf("File handle: %i\n", fileHandle);
                       for (i = height; i >= 0; i = i-1){
                         for (j = 0; j < width; j = j+1){
```

```
                                att1 = (unsigned
char)alt_up_sd_card_read(fileHandle);
                                att2 = (unsigned
char)alt_up_sd_card_read(fileHandle);
                                att3 = (unsigned
char)alt_up_sd_card_read(fileHandle);
                                pixel = ((att3>>3)<<11) |
((att2>>2)<<5) | (att1 >> 3);
                                //pixel = ((att3)<<16) +
((att2)<<8) + (att1);
                          VGA_box(j,i,j,i,pixel);
                          }
                      }
                      if ((Value & 0xF000) == 0x1000){
                              invertPhoto();
                      }
                      alt_up_sd_card_fclose(fileHandle);
                  }
                 else
                 {
                  printf("Unknown file system.\n");
                 }

                 connected = 1;
             }
             else if ((connected == 1) &&
(alt_up_sd_card_is_Present() == false))
             {
                 printf("Card disconnected.\n");
                 connected = 0;
             }
         }
         else
         {
          printf("Initialization failed.\n");
         }

     }

void invertPhoto(){
     short att1 = 0, att2 = 0, att3 = 0, att;
     int pixel;
     int width = 320;
     alt_up_sd_card_dev *device_reference = NULL;
    int connected = 0;
    short int fileHandle;
```

```
     if ((Value & 0xF00) == 0x900){
               width = width + (Value & 0x0FF);
          }

               device_reference =
alt_up_sd_card_open_dev(ALTERA_UP_SD_CARD_AVALON_INTERFACE_0_NAM
E);
               if (device_reference != NULL)
               {
                    if ((connected == 0) &&
(alt_up_sd_card_is_Present()))
                    {
                        if (alt_up_sd_card_is_FAT16())
                        {
                          if (FileSelect == 0){
                                    fileHandle =
alt_up_sd_card_fopen("smile.bmp",false);
                               }
                          else if (FileSelect == 1){
                               fileHandle =
alt_up_sd_card_fopen("tayeb2.bmp",false);
                               }
                          else if (FileSelect == 2){
                                    fileHandle =
alt_up_sd_card_fopen("buko.bmp",false);
                          }

                             att =
alt_up_sd_card_get_attributes(fileHandle);

                             VGA_box(0,0, 321, 241, 0xFFFF);

                             for (j=0; j<54; j++) //gets past
header
                             {
                               att1 =
alt_up_sd_card_read(fileHandle);
                             }
                             i = 0, j = 0;
                             printf("File handle: %i\n",
fileHandle);
                             for (i = 0; i <= 240; i = i+1){
                               for (j = 0; j < width; j = j+1){
                                    att1 = (unsigned
char)alt_up_sd_card_read(fileHandle);
                                    att2 = (unsigned
char)alt_up_sd_card_read(fileHandle);
```

```
                                    att3 = (unsigned
char)alt_up_sd_card_read(fileHandle);
                                    pixel = ((att3>>3)<<11) |
((att2>>2)<<5) | (att1 >> 3);
                                    //pixel = ((att3)<<16) +
((att2)<<8) + (att1);
                                  VGA_box(j,i,j,i,pixel);
                                  }
                                }
                                alt_up_sd_card_fclose(fileHandle);
                             }
                            else
                            {
                             printf("Unknown file system.\n");
                            }

                            connected = 1;
                        }
                        else if ((connected == 1) &&
(alt_up_sd_card_is_Present() == false))
                        {
                            printf("Card disconnected.\n");
                            connected = 0;
                        }
                    }
                    else
                    {
                     printf("Initialization failed.\n");
                    }

}

void VGA_box(int x1, int y1, int x2, int y2, int Color){
     int offset, row, col, StateFlag, NewValue;
     int R, G, B;
     StateFlag = Value & 0xF00; //Get State Flag
     NewValue = Value & 0x0001F; //Remask Value

     R = Color & (0b1111100000000000);
     G = Color & (0b0000011111000000);
     B = Color & (0b0000000000011111);
     switch (StateFlag >> 8){
          case(1):
                    R = R + (NewValue << 11);
                    G = (Color & 0b0000011111100000) + (NewValue
<< 6);
                    B = B + NewValue;
```

```
                    if (R >= 0b1000000000000000){
                         R = 0b11111 << 11;
                    }
                    if (G >= 0b0000100000000000){
                         G = 0b111111 << 5;
                    }
                    if (B >= 0b100000){
                         B = 0b11111;
                    }
                    Color = R | G | B;
                    break;
          case(2): //Red becomes Blue, Blue Becomes Green, Green
Becomes Red
                    Color = ((R >> 11) | (G << 5) | (B << 6));
                    break;
          case(3): //Red becomes Green, Green becomes Blue, Blue
becomes Red
                    Color = ((R >> 5) | (G >> 5) | (B << 11));
                    break;
          case(4):
                    Color = Color & 0b1111100000000000;
                    break;
          case(5):
                    Color = Color & 0b0000011111100000;
                    break;
          case(6):
                    Color = Color & 0b0000000000011111;
                    break;
          default:
               break;
     }

     volatile short *pixel_buffer = (short *) 0x00200000;
     for (row = y1; row <= y2; row++){
          col = x1;
          while (col <= x2){
               offset = (row << 9) + col;
               *(pixel_buffer + offset) = Color;
               ++col;
          }

     }
}

void key3_isr(){
     // Reload Image if switch 1 is high change to next photo
     int Sw = IORD(SWITCHES_BASE, 0);
```

```
	if ((Sw & 0x1) == 1){
		if (FileSelect == 0){
			FileSelect = 1;
		}
		else if (FileSelect == 1){
			FileSelect = 2;
		}
		else FileSelect = 0;
	}
	printf("KEY 3");
	initCard();
	IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
	IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
	return;
}

void key2_isr(){
	//Just Red Values, Just Blue Values, Just Green Values
	if (Value == 0x400){
		Value = 0x500;
	}
	else if (Value == 0x500){
		Value = 0x600;
	}
	else Value = 0x400;

	IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
	IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
	return;
}

void key1_isr(){
	//INVERT RGB
	printf("KEY 2\n");
	if (Value == 0x200){
		Value = 0x300;
	}
	else if(Value == 0x300){
		Value = 0x000;
	}
	else Value = (0x200);

	IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
	IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
	return;
}
```

```
void key0_isr(){
     //Brightness, Increase Pixel Color by Switch Value
     printf("KEY 0");
     Value = IORD(SWITCHES_BASE, 0);
     if ((Value & 0b1000000000000000000) > 0){
          //If Value of 17 is high, Width Adjustment go crazy
          Value = Value & 0x001F;
          Value = Value + 0x900;
     }
     else if ((Value & 0b0100000000000000000) > 0){ //Switch 16
is high, Mirror the image up and down
          Value = Value & 0x001F;
          Value = Value + 0x1000;
     }
     else if ((Value & 0b0010000000000000000) > 0) { //Switch 15
is high brightness changes
     Value = Value & (0x001F);
     Value = Value + (0x100); //Add the State Flag
     }
     IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0);
     IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
     return;
}

void handle_key_interrupts(void* context){
     volatile int *edge_capture_ptr = (volatile int*) context;
     *edge_capture_ptr =
IORD_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE);
     if (*edge_capture_ptr & 0x8){
          key3_isr();
     }
     else if (*edge_capture_ptr & 0x4){
          key2_isr();
          }
     else if (*edge_capture_ptr & 0x2){
          key1_isr();
     }
     else if (*edge_capture_ptr & 0x1){
          key0_isr();
     }
     return;
}

void pio_init(){
     void* edge_capture_ptr = KEYS_EDGE_TYPE;
     IOWR_ALTERA_AVALON_PIO_IRQ_MASK(KEYS_BASE, 0xF);
     IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEYS_BASE, 0x0);
```

```
    alt_irq_register(KEYS_IRQ, edge_capture_ptr,
handle_key_interrupts);

    return;
    }
```