Portland State
UNIVERSITY

ECE 540

PORTLAND STATE UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SOC Design with FPGAs

BOX DETETCTION AND IMAGE REPLACEMENT REPORT

Atharva Mahidrakar
atharva2@pdx.edu

Jagir Charla
jcharla@pdx.edu

Hamed Mirlohi
mirlohi@pdx.edu

Prasanna Kulkarni
prasanna@pdx.edu

Date: March 22, 2019

Contents

List of Figures

1 Introduction

The project is simplified green screen system. It will detect a colored box in a image and then will replace the box with an image. The output of the system will be shown on a VGA screen. A simplified representation of the system is as follows:

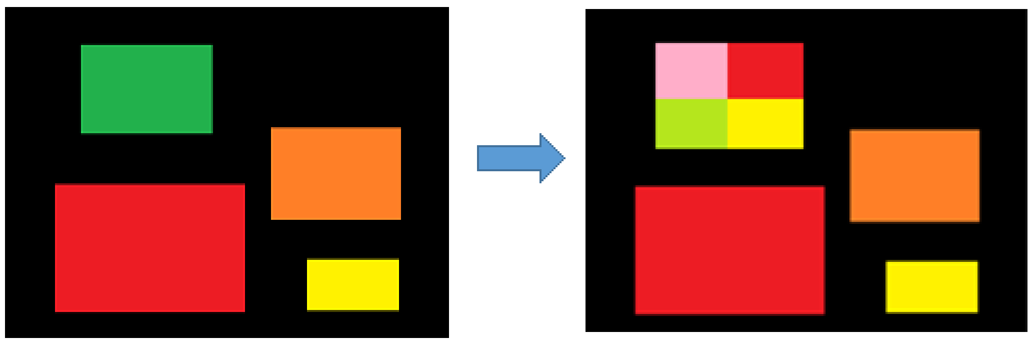


Figure 1: Basic Project Concept

The extendiblity of this project comes in the form of the OV 7670 camera. Where we can use the camera for replacing a colored box within the live video feed. We proposed that this be our stretch goal. The plan of the project then was

- **Backup Plan**: Replace a colored box in a predefined image. The backup plan proposed having only a few colored boxes which would make detection easier.
- **Main Plan**: Replace a colored box on a preloaded image
- **Stretch Goal**: Use the OV 7670 camera and replace the colored box in a live feed in real time.

The image in the backup plan is given as follows

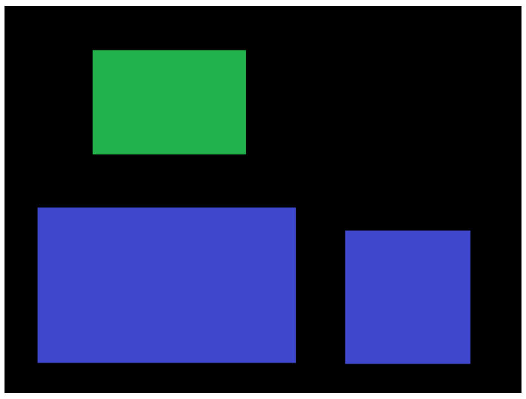


Figure 2: Backup Plan Concept

The reason this image is easier is that there are less colors to detect in the image.

2 Architecture

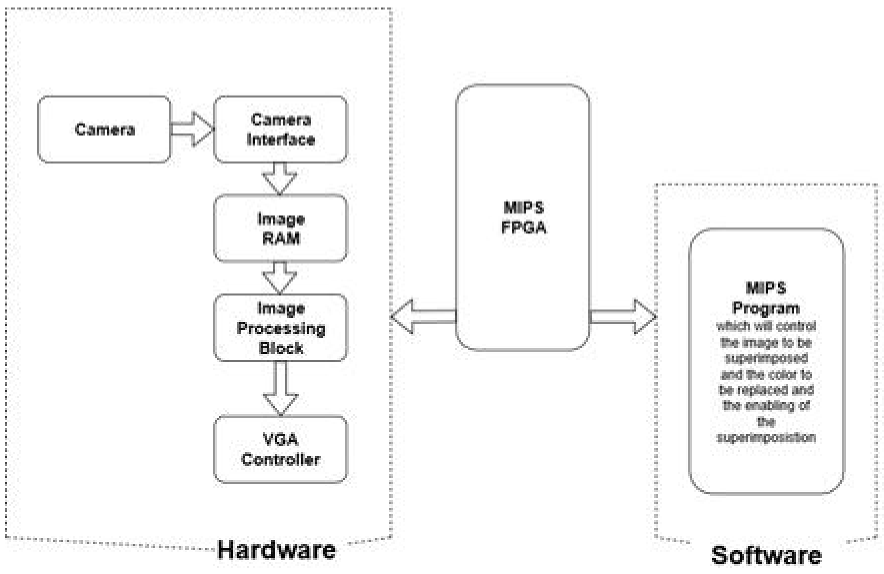

Camera
Camera Interface
Image RAM
Image Processing Block
VGA Controller
MIPS FPGA
MIPS Program which will control the image to be superimposed and the color to be replaced and the enabling of the superimposistion
Hardware
Software

Figure 3: Architecture

The architecture of the system is broadly divided into Hardware and Software sections. The hardware portion is the part that gets written in Verilog and integrated into the MIPSfpga hierarchy. The software part is the one that we write in the MIPS Assembly language.

2.1 Hardware

- **Image RAM**: Will hold the predefined image that will be super imposed. In terms of the example given above the multi-color boxes image will be held in this RAM.
- **Image Processing**: This will have the image processing algorithm that we will use to detect the colored box.
- **VGA Controller** This will control the VGA Display and write our image to the display.

2.2 Software

- **Enabling**: This will start or stop the image superimposition.
- **Selection**: This will select the color that we need to detect and superimpose. In the above example this color will be green. We have support for green and blue.

3 Flow of Project

3.1 Stationary Image

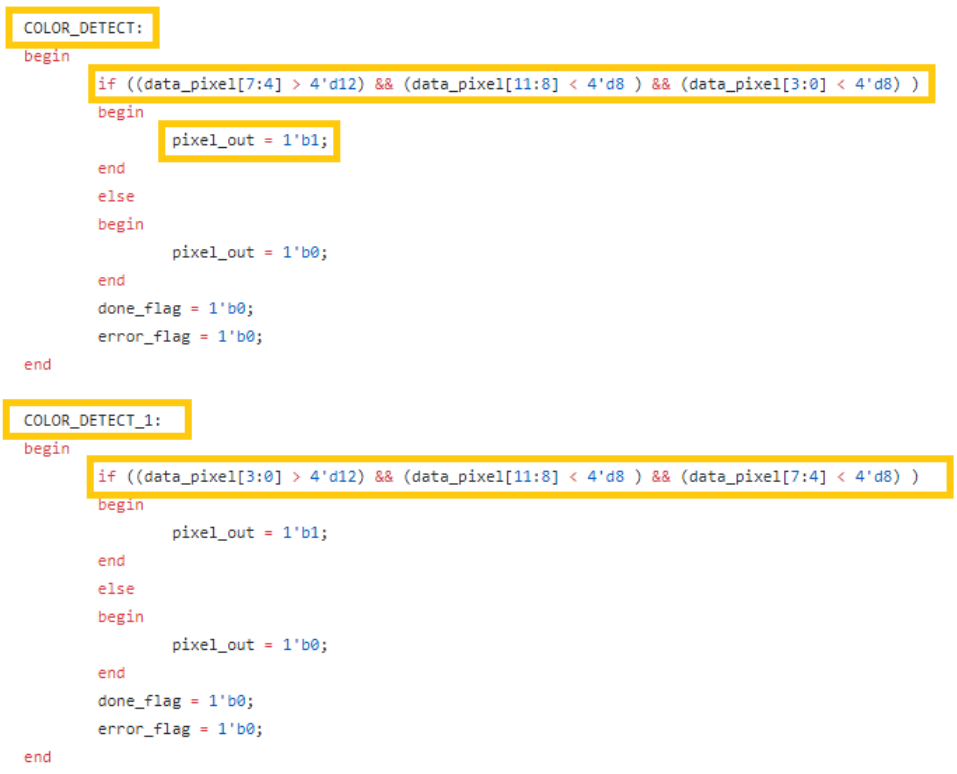
The first step in the flow of the project was to implement the image replacement system on a preloaded image. The steps that we followed in this stage were

- Load image in BRAM
- Detect a colour using filtering
- Generate the min-max coordinates

3.1.1 Loading image in BRAM

The image was loaded into a BRAM using the Xilinx Memory Generator. We provided a .COE file to the RAM in order to get the data in the RAM. The .COE file itself was generated using a Python script.

3.1.2 Detect a colour using filtering



```
COLOR_DETECT:
begin
        if ((data_pixel[7:4] > 4'd12) && (data_pixel[11:8] < 4'd8 ) && (data_pixel[3:0] < 4'd8) )
        begin
                pixel_out = 1'b1;
        end
        else
        begin
                pixel_out = 1'b0;
        end
        done_flag = 1'b0;
        error_flag = 1'b0;
end

COLOR_DETECT_1:
begin
        if ((data_pixel[3:0] > 4'd12) && (data_pixel[11:8] < 4'd8 ) && (data_pixel[7:4] < 4'd8) )
        begin
                pixel_out = 1'b1;
        end
        else
        begin
                pixel_out = 1'b0;
        end
        done_flag = 1'b0;
        error_flag = 1'b0;
end
```

Figure 4: Detecting the Color

This module is used to detect desired colored box and find out 4 min-max x and y coordinates of that box. We decided to use this approach because it is comparatively easier to implement in Verilog. The complex algorithms like Harris corner detection would give us better result but are time consuming to implement in Verilog. The basic version of this module was used on static image for the box detection, but in due to the strech goal of interfacing the camera this module got modified. This modified module is explained below.

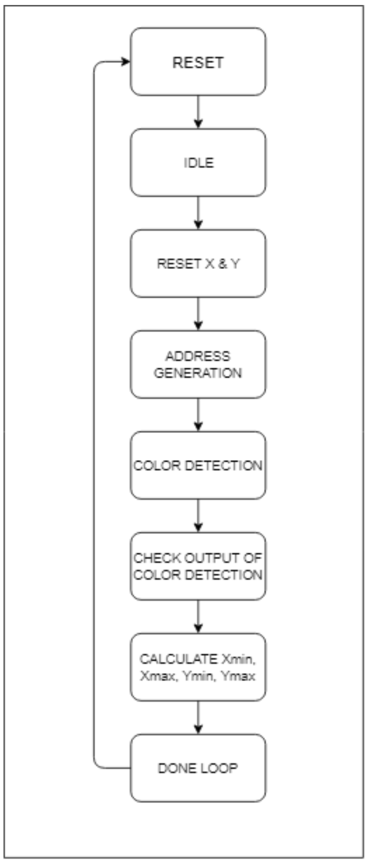

RESET
IDLE
RESET X & Y
ADDRESS GENERATION
COLOR DETECTION
CHECK OUTPUT OF COLOR DETECTION
CALCULATE Xmin, Xmax, Ymin, Ymax
DONE LOOP

Figure 5: Filter Algorithm

- RESET: This state gets called when user presses reset button. Over here the output flags like done flag and error flags are reset to zero. As well the variables x_count & y_count used to traverse the image are reset to zero. After that we switch to IDLE state.11
- IDLE: In this state the temporary variables x_min ,x_max,y_min & y_max are assigned to 319, 0, 319 & 0 respectively. After this it waits for the start flag to turn 1. Unless start flag turns one, it loops back to this state otherwise it jumps to next state Reset x & y.
- RESET X and Y: In this state, again the x & y variables used in traversing image are reset to zero. Also the variable address is reset to zero. After this it jumps to next state ADDRESS GENERATION.
- ADDRESS GENERATION: In this state we generate an address which would be used to access the pixels data of an image stored in BRAM. As memory

sequentially addresses from 0 to (320 x 240) 1, so to map our 2 D image in terms of X & Y coordinates we are using x_count & y_count variables and to get address of particular pixel on image we use following expression to get the sequential address

$$address = (y\_count * 320) + x\_count \tag{1}$$

Using this method, we can access all the pixels from (0,0) to (319,239) mapped sequentially on range 0 to 76799. After generating the address of particular pixel location then this data is sent to memory block which in return gives RGB data of that pixel. After this we switch to COLOR DETECTION state.

- COLOR DETECTION: For this project we are detecting colored boxes pf green & blue having highest contrast values. For example, for detection of green colored box either preloaded on the memory or shown in front of camera has 12 bit [R G B] data as [0 15 0]. This was done for ease of detection at real time.Now for the detection part, after doing running lot of trials we figured out that for green color detection the RGB threshold given must be [8 12 8] and for blue color detection it must be [8 8 12]. If the current pixels RGB data exceeds the threshold then it is counted as correct detection otherwise it would be treated as not detected. Depending upon the detection, color detecting flag is set to one or zero and then we switch to next state CHECK OUTPUT OF COLOR DETECTION.
- CHECK OUTPUT OF COLOR DETECTION: In this state, is the color detecting flag is set to 1 then we jump to CALCUTATE x_min, x_max, y_min, y_max state otherwise we jump to ADDRESS_GENERATION state.
- CALCUTATE x_min, x_max, y_min, y_max: At reset we have assigned 319, 0, 319 & 0 these values to temporary variables x_min ,x_max,y_min & y_max respectively. After each correct detection, the current X & Y coordinates saved in x_count & y_count are run through this block. The pseudo code for the min max algorithm is as follows

```
if(x\_count<x\_mintemp)
        x\_mintemp = x\_count
else
        x\_mintemp = x\_mintemp

if(y\_count<y\_mintemp)
        y\_mintemp = y\_count
else
        y\_mintemp = y\_mintemp

if(x\_count > x\_maxtemp)
        x\_maxtemp = x\_count
else
        x\_maxtemp = x\_maxtemp
```

```
if(y\_count > y\_maxtemp)
        x\_maxtemp = x\_count
else
        y\_maxtemp = y\_maxtemp
```

This part gets executed after each correct detection and after that we jump to DONE LOOP

- DONE LOOP: Here we check whether we have reached at the end of memory location or not. If we are not at end of memory we jump to ADDRESS GENERATION state. Otherwise we wait for acknowledge signal. If the acknowledge signal is 0 then we keep looping in this state. If acknowledge state is 1 then we forward the min max data to `overlap_image.v` module and jump to IDLE state.

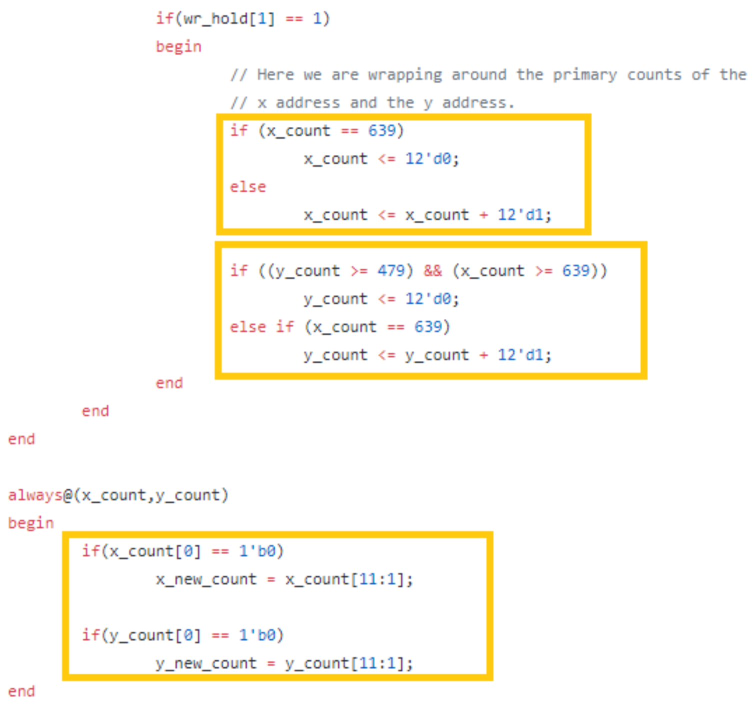
3.2 Camera Interfacing

The camera that we used was a OV 7670 camera. The interfacing of the camera was done using the PMOD ports. The camera needs to have two biasing resistors at the SOIC and the SOID pins which are the SCCB (Serial Camera Control Bus). The SCCB is an I2C-like bus that the OV 7670 uses for communication. The camera configuration and the SCCB interfacing modules taken, with permission from the Camera Interfacing project by Maxwell Cui. We edited the `verilog_capture.v` module to fit our usage better.
The start pulse that the camera needs was provided using the 15th switch on the Nexys DDR4 board especially during the first run when the camera is not configured. The camera transmits the image data in the 640x480 resolution. To make the camera work properly and still have some space for the processing of the camera stream, we have to scale down the image to 320 x 240. Doing this requires storing only the even numbered but not the odd numbered pixels. This is done in the `verilog_capture.v` module.
The above figure is our algorithm to scale down the image. In the first and second highlighted snippet we generate counts that wrap around at the upper bounds of the image. For instance a 640 x 480 image the maximum x coordinate count will be 640 and the y coordinate count will be 480. The last highlighted snippet divides the count by two if the count value is even. The address for storing the image data from the camera in a BRAM is given by:

$$address = (y\_new\_count * 320) + x\_new\_count \tag{2}$$

3.3 Box Detection on Video Feed

This section is the where we'll explain the working of the live feed image replacement. We have basically used two BRAM memories one to store the live feed data called the Frame Buffer and another which is a clone of the frame buffer and is used



```
        if(wr_hold[1] == 1)
        begin
            // Here we are wrapping around the primary counts of the
            // x address and the y address.
            if (x_count == 639)
                x_count <= 12'd0;
            else
                x_count <= x_count + 12'd1;

            if ((y_count >= 479) && (x_count >= 639))
                y_count <= 12'd0;
            else if (x_count == 639)
                y_count <= y_count + 12'd1;
        end
    end
end

always@(x_count,y_count)
begin
    if(x_count[0] == 1'b0)
        x_new_count = x_count[11:1];

    if(y_count[0] == 1'b0)
        y_new_count = y_count[11:1];
end
```

Figure 6: Downscaling the Image

for the processing of the image. We've also defined a state machine to control the setting and resetting of flags which drive other smaller state machines like the color filtering one which was explained above. This control state machine is referred to as the main state machine henceforth. The various parts of the main state machine are explained as follows:

3.3.1 SM_TAKE_PHOTO_START

This state starts the execution of another state machine called the `photo_sm` which has been defined in `photo_sm.v`. This is done by giving the Photo_SM a `start_flag`. The Photo_SM is where we define a single frame of the camera feed. This is done by waiting between two consecutive signal highs of `vsync`. In between these two signals the Photo_SM will give a `write_enable` signal as an output. This `write_enable` is used in the upper module to determine the writing on the memory. This writing process needs to be done once per frame on each frame and the Main state machine takes care of that.The writing process is us computing a new image based on the location of the green box and will be explained further.

3.3.2 SM_TAKE_PHOTO_START

This state provide the `start_flag` to the Photo_SM which starts it's execution.

3.3.3 SM_TAKE_PHOTO_STARTED

At the first occurrence of `vsync` this state will transition letting us know that the execution of our box detection and image replacement has to start.

3.3.4 SM_TAKE_PHOTO_EXEC

This state waits till the Photo_SM sends a done flag, which it sets during its final stage. When this state receives the done flag it transitions

3.3.5 SM_TAKE_PHOTO_DONE

The Photo_SM will stay in its final stage till it receives an acknowledgement flag. The ack flag is given by this stage.

This part of the Main_SM is primarily devoted to the working of the Photo_SM. As mentioned earlier the purpose of this routine is to detect a whole frame getting written into the frame-buffer (BRAM storing the camera feed). The next part of the state machine is the Min-Max computation.

3.3.6 SM_MIN_MAX

This part of the state machine is similar to the one above, there are different 4 different states in this part which are, SM_MIN_MAX_START, SM_MIN_MAX_EXEC, SM_MIN_MAX_DONE, SM_MIN_MAX_WAIT these states control the execution of the filter state machine given above. The min-max values of a single frame in the video feed are calculated exactly like the still-image min-max calculation process.

The last step of this state machine sends the min-max data to the `overlap_image.v` module.

3.3.7 Overlap_Image and Colorizer

overlap_image module takes bounding box information like min max and center of the detected box with pixel_row and pixel_column to help the the colorizer to decide what to display on the VGA screen. The pixel_row and pixel_column are generated using the DTG module given in the RojoBot release, which has been slightly tweaked. Based on the bounding information this module divides the detected region into 4 quadrants and outputs a 3 bit signal. according to this 3 bit signal colorizer decides whether to show background image (i.e. live feed) or whether to display color of 1st quadrant or 2nd or 3rd or 4th
Due to space constraint we are considering image of 2x2 which is equivalent to 4 quadrants. so overlap_image is like a scalar block for our 2x2 image. Thus we can replace a colored box in the live feed with an image.

3.4 Software

In the box detection and color replacement program, seven segment display is used for Graphic User Interface (GUI). For this purpose, the user has two options to control the box detection menu. Push buttons and switches are utilized to set and read the menu options.

3.4.1 Buttons

For the first option, the seven segment display is initially turned off and nothing is shown. In order for the user to modify the settings and enter the settings mode, they can press the left button. Upon pressing the left button the enable/disable menu option appears and the user knows they entered the settings mode. If the user decides to switch to an image selection menu, he/she presses the up button to pick the color (Blue or Green) to be superimposed. The menu selection direction goes both ways. Pressing the down button takes the user to reverse direction. If the user wishes to modify either setting, he/she presses the right push button. At any given time if the user desires to exit the menu, he can press the left button and all changes will be saved and seven segment turns off.

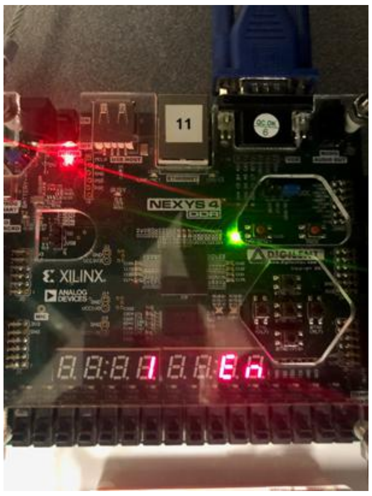


Figure 7: Button GUI

3.4.2 Switches

For switches, the menu works differently. At the beginning when the program is uploaded, the seven segments are automatically on and all the settings are displayed. The following are the assignments for the switches. Far right switch is dedicated to enabling/disable the camera. Next two switches are used to select the image to superimpose on the targeted color and finally the 4th button is dedicated to setting the color the user wants to target.
For this project, extra characters are required to be added to the mfp_ahb_sevensegdec module to support the extra letters. For example, In order to add p character, a, b, f,

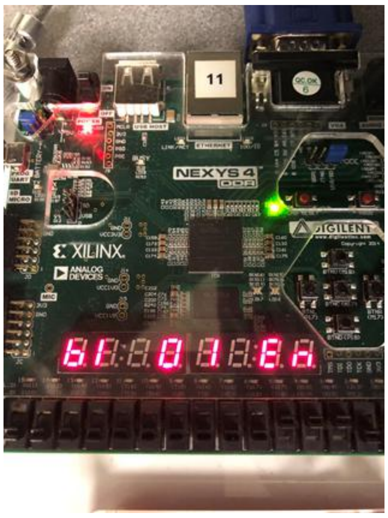
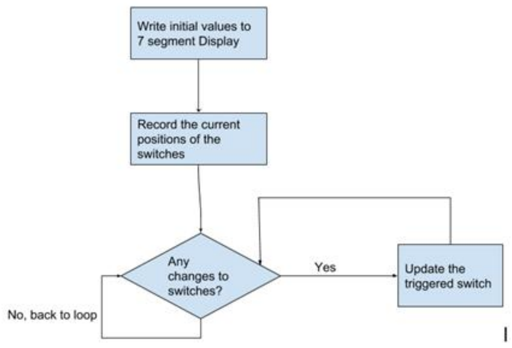


Figure 8: Switch GUI


Write initial values to 7 segment Display
Record the current positions of the switches
Any changes to switches?
Yes
Update the triggered switch
No, back to loop
Figure 9: GUI Flowchart

g, and e segment need to be turned on. To turn on those segments value of zero is inserted. Therefore, (abcdefg) = 0011000 corresponds to letter p where the seven segment display is given as follows:

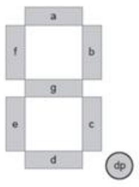


Figure 10: Seven Segment Display

4 Goals Achieved

- The still-image replacement was achieved.
- We successfully interfaced the camera.
- We successfully achieved our stretch goal of image replacement on a live feed.
- Results with live feed:

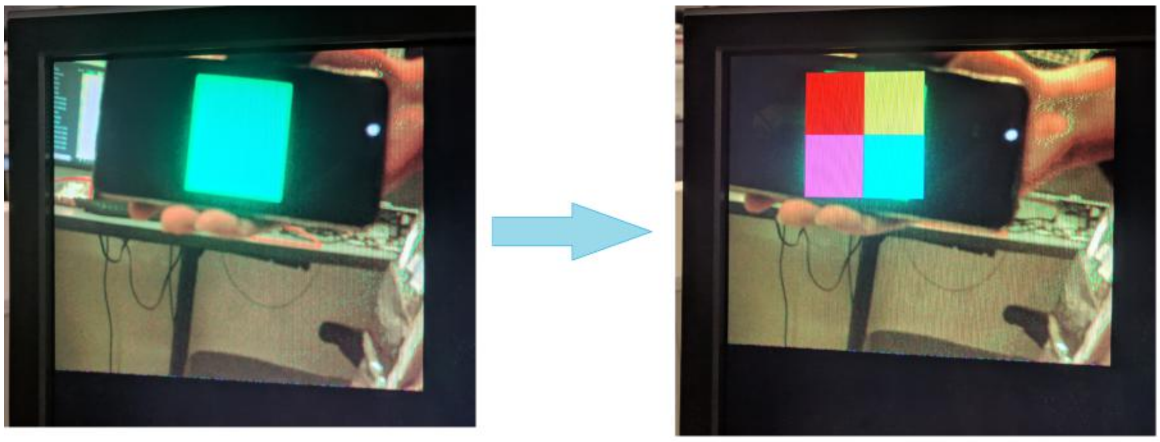


Figure 11: Final Result

5 Challenges

- **Generation of min and max coordinates**: This was one of the most important parts of the project.Coming up with an algorithm that was easy to synthesize was a problem.
- **Memory** This was another problem that we faced in this stage.If the image to be taken was of 640 x 480, which is the resolution that the camera would work at, most of the BRAM space was used up between the image and MIPSfpga Solution: Image was taken at a resolution of 320 x 240
- **Flicker** This was a nasty bug that took up a lot of time in the design process. It caused the superimposed image to appear on the screen but continuously flicker as if being constantly re written. Solution: Remove inferred latches. The source of the bug was a series of latches that had been inferred, we found them in the Elaborated Schematic.

6 Scope

- Finding a way to utilize more BRAM, with two 320x240 images and the MIPSfpga almost 90% of the BRAMs are used up.
- If more memory is available, then replacing the image with a more complex image, like the Mario wallpaper.
- Using HSV implementation for better filtering of the color.
- Using Affine transform to orient the superimposed image according to the source box.

7 Workload Division

7.1 Atharva

Color Filtering and Detection

7.2 Prasanna

Camera State Machine and Video Feed working

7.3 Jagir

Main State Machine Logic and Upper-level integration

7.4 Hamed

Software with the necessary hardware Integration

8 Refrences

- ECE-540 RojoBot project release
- The Math Coursework Template on Overleaf by Qiao Han.Liscenced under Creative Commons CC BY 4.0
- The MIPS Green Sheet by UC Berkeley at https://inst.eecs.berkeley.edu/ cs61c/resources/MIPS_Green_Sheetṗdf
- The MIPS Code Reference from the University of Idaho at http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSirḣtml
- OV7670 Interfacing repo at https://github.com/maxwellcui/OV7670-NEXYS4-VGA by Maxwell Cui.

- https://www.voti.nl/docs/OV7670.pdf.