Bert Zhao, Shangrong Li
EE 371
Final Project Lab Report

Abstract:
This is the lab report for the final project of EE 371. This report provides an overview of the project, along with a procedure and specific breakdown of the inner functionality of the system. An analysis of the project is provided at the end of the report along with some sample images of the working program and screenshots of simulations and other information related to program compilation. Key pieces of code are included in the appendix.

Introduction:
The goal of the final project is to create an image/video filtering system. We introduce the D8M-GPIO camera in this final project, and it serves as the main feature of our project. Our system implements multiple filtering options to the video output, including grey scale, inverted colors, and other features. We also implement a keyboard peripheral to select between the different filtering modes. The purpose of this lab is to serve as a conceptual wrap-up for the course, by implementing most of the features we have used in previous labs.

Materials:
- **Altera Terasic DE1-SoC Board with power and USB cables**
- **PC with Altera Quartus Prime Lite installed**
- **LEDs for GPIO implementation**
- **Keyboard using PS2 interface**
- **Terasic D8M-GPIO camera**

Procedure:
In completing this lab, we must first implement the D8M_GPIO camera and a keyboard that has a default PS2 interface. The drivers required for the basic functionality of these hardware components was provided to us. The camera sends information such as the RGB values of pixels, and image sync properties, such as horizontal and vertical sync. The keyboard drivers send a make break signal, which indicates the status of a key press and an 8-bit code that represents the unique key that was pressed.

The primary functionality for the image processing done in this lab requires the use of a filter module that directly manipulates the RGB values. The image processing options in this project include six different brightness levels, inverted colors, grayscale, and increased contrast. The number keys one through nine on the keyboard are used to select between these settings. When no keys are pressed, the camera operates on default image settings.

The second GPIO slot is used for the LEDs. There are a total of nine LEDs, one for each of the image filtering modes. The LEDs correspond to keys one through nine as you go from left to right. For example, the leftmost LED is active if 1 is pressed on the keyboard, and the rightmost LED is active if 9 is pressed on the keyboard. Each one will remain active even if the corresponding button is released, effectively indicating the last selected mode. Also, only one filtering option may be selected at once, applying only the most recently selected mode.

The final output is sent to a monitor via VGA port from the FPGA. Images are not stored, therefore the output is essentially a filtered direct stream of video from the camera.

Modules:
The primary functionality of the image processing is driven by the filter module. This module handles nine distinct calculations for the image filtering. All pixel information is stored in a set of three 8-bit numbers. The RGB values with each ranging from 0 to 255, with 0 representing the complete absence of color and 255 representing complete saturation. To select between the nine modes, the filter module takes in the outcode from the keyboard and appropriately assigns the filtering mode with its respective key.

To give the appearance of modifying brightness, we must increase or decrease the "blackness" of each pixel. In mode one, the RGB values of each pixel is divided by two. This brings each set of RGB values closer to black. In modes two through six, each RGB value is multiplied by a factor greater than one, increasing its value. Since the values are brought closer to 255, the overall image will become more white. In mode 2, the multiplier is 1.2 and this increases in even increments to mode 6, in which the multiplier is 2. Since each RGB value cannot be greater than 255, if the post processed value were to be greater than 255, the post processed value is not used and 255 is used instead.

For inverting colors, each RGB value is subtracted from 255. This creates an easily observable effect where light colors become dark and dark colors become light.

For greyscale, each RGB value is assigned the average of the RGB values. This means that everything is pushed to the lower end of spectrum, and that the pixels will be overall more uniform in color.

For increasing contrast, each RGB value is first checked to see if its higher or lower than 127. Values less than 127 are multiplied by 0.6 and values greater than 127 are multiplied by 1.6. When increasing values with the higher multiplier, they are first checked if the new value would be greater than 255. If so, the high value just becomes 255.

For the LEDs, we used second GPIO and controlled them with the out codes similarly to how we controlled the selection of the different filtering modes. We use GPIO pins 35 to 27 with GPIO[35] for key one down to GPIO[27] for key nine.

The top level module of this project was adapted from the provided camera drivers. It incorporates the keyboard drivers, the filter module, and the LED selection from the second GPIO.

Simulation:
The simulation for the filter is show below. We can see how the RGB values are changed based on the outcode that was most recently selected.

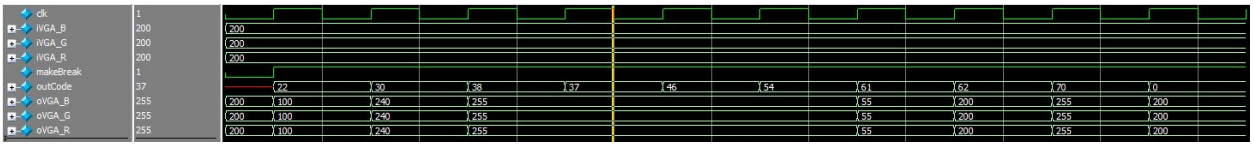

Figure 1: Filter module ModelSim screenshot

Results and Analysis:
Some sample images of the different image processing modes are shown below. The flow summary from the final compilation of the project is also included. Some minor issues of note while working on this project include issues with adapters on the ports. The PS2 to usb adapter did not work because keyboards that do not have a default PS2 interface do not draw power from the DE1-SoC. Likewise for the VGA to HDMI adapter, the extra adapter we had purchased did not draw the necessary power through the VGA for it to work. The other minor issue regarded the GPIO, as using the lower GPIO pins prevented the use of the camera. This was solved in our project by using the higher pins for the LEDs. In our specific case, we used pins 35 through 27.

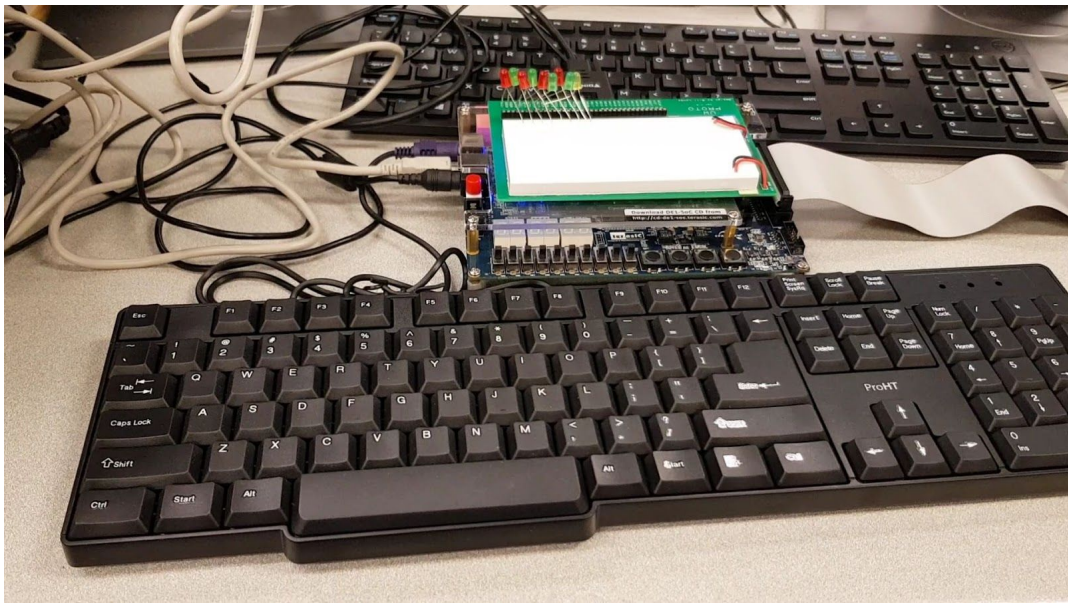

Figure 2: Overview of DE1_SoC setup

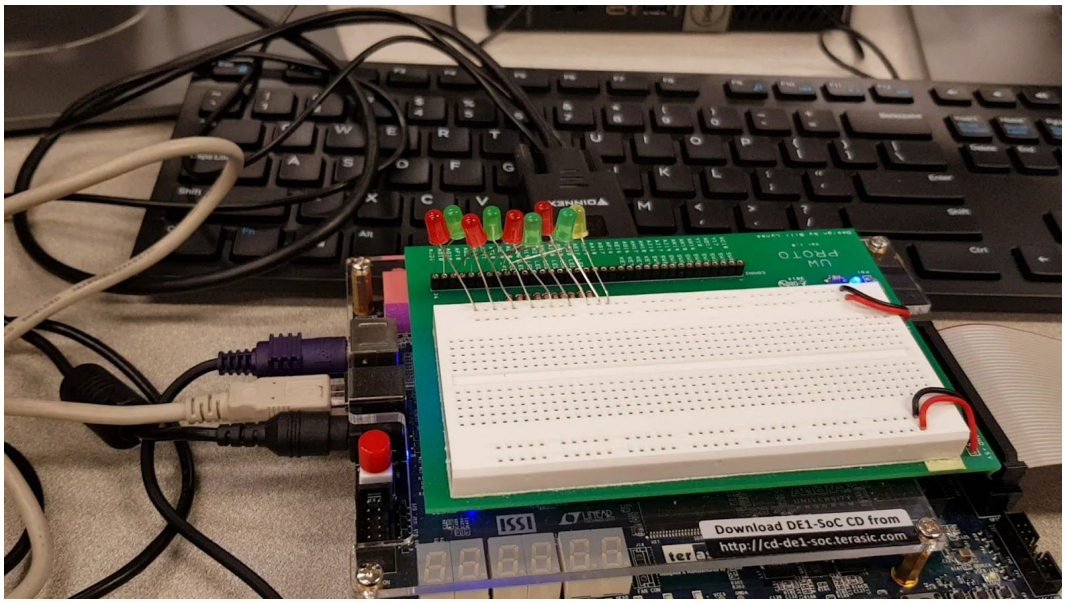

Figure 3: GPIO LED setup

Flow Summary

<<Filter>>

Revision Name	DE1_SOC_D8M_RTL
Top-level Entity Name	DE1_SOC_D8M_RTL
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	2,673 / 32,070 (8 %)
Total registers	2625
Total pins	226 / 457 (49 %)
Total virtual pins	0
Total block memory bits	88,448 / 4,065,280 (2 %)
Total DSP Blocks	2 / 87 (2 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	3 / 6 (50 %)
Total DLLs	0 / 4 (0 %)

Figure 4: Flow Summary

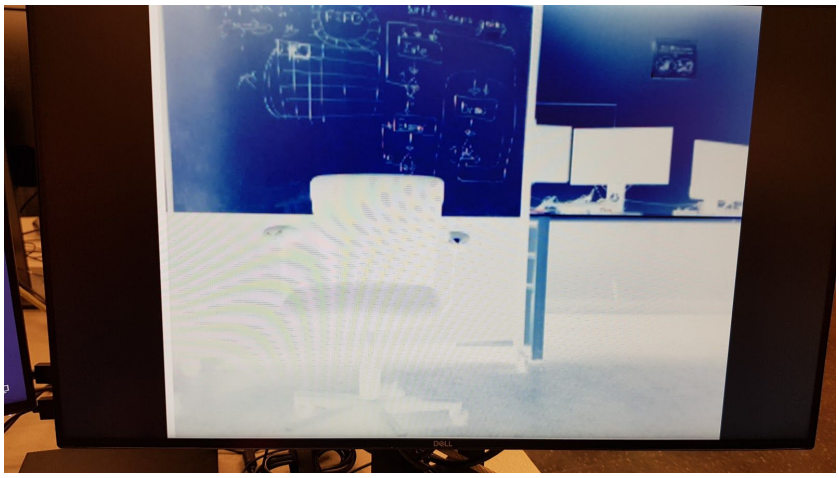

Figure 5: Inverted colors output

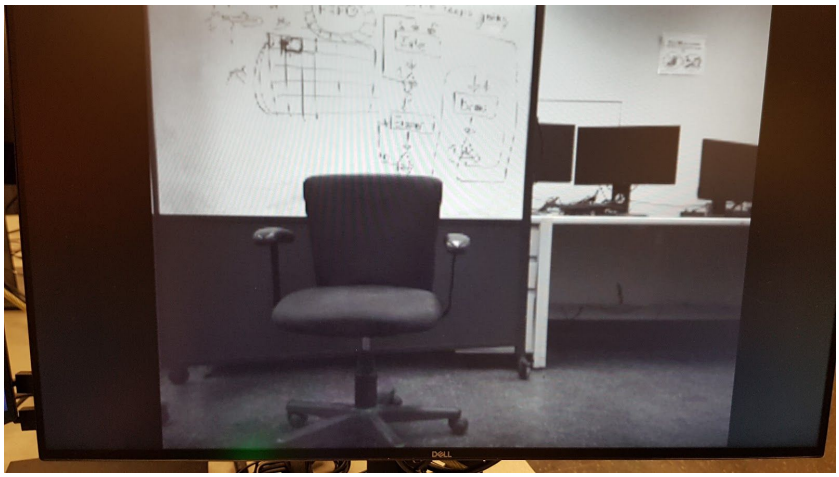

Figure 6: Grayscale output

Figure 7: Increased contrast output

Figure 8: Lowest brightness

Figure 9: Highest brightness

Conclusion:

For this final project, we combined many different features from previous labs. Additionally, we implemented a camera and keyboard as additional inputs to the system. We created an image filtering system with multiple modes of processing. This final project was a three week process and the entirety of the project took approximately 25 hours to complete.

Appendix:

```
// Warning: the Terasic VGA controller appears to have a few off-by-one errors.  If your code is very
// sensitive to the EXACT number of pixels per line, you may have issues.  You have been warned!

module Filter #(parameter WIDTH = 640, parameter HEIGHT = 480)
(
   input logic                  VGA_CLK, // 25 MHz clock

   // *** Incoming VGA signals ***
   // Colors.  0 if iVGA_BLANK_N is false.  Higher numbers brighter
   input logic         [7:0]    iVGA_B, // Blue
   input logic         [7:0]    iVGA_G, // Green
   input logic         [7:0]    iVGA_R, // Red
   // Horizontal sync.  Low between horizontal lines.
   input logic                  iVGA_HS,
   // Vertical sync.  Low between video frames.
   input logic                  iVGA_VS,
   // Always zero
   input logic                  iVGA_SYNC_N,
   // True in area not shown, false during the actual image.
   input logic                  iVGA_BLANK_N,

   // *** Outgoing VGA signals ***
   output logic        [7:0]    oVGA_B,
   output logic        [7:0]    oVGA_G,
   output logic        [7:0]    oVGA_R,
   output logic                 oVGA_HS,
   output logic                 oVGA_VS,
   output logic                 oVGA_SYNC_N,
   output logic                 oVGA_BLANK_N,

   // *** Board outputs ***
   output logic           [6:0]    HEX0,
   output logic           [6:0]    HEX1,
   output logic           [6:0]    HEX2,
   output logic           [6:0]    HEX3,
   output logic           [6:0]    HEX4,
   output logic           [6:0]    HEX5,
   output logic           [9:0]    LEDR,

   // *** User inputs ***
   input logic            [1:0]    KEY, // Key[2] reserved for reset, key[3] for auto-focus.
   input logic            [8:0]    SW,   // SW[9] reserved for auto-focus mode.
   input logic             [9:0]   keyboardNum,
   input logic makeBreak,
   input logic [7:0] outCode
);
```

```
    always_comb begin
        if (makeBreak) begin
            if (outCode == 8'b00010110) begin // when the 1 key is pressed on the keyboard; half brightness level 1
                oVGA_R = iVGA_R / 2;
                oVGA_G = iVGA_G / 2;
                oVGA_B = iVGA_B / 2;
            end else if (outCode == 8'b00011110) begin // 2; brightness level 2
                oVGA_R = iVGA_R * 6 / 5 > 255 ? 255 : iVGA_R * 6 / 5;
                oVGA_G = iVGA_G * 6 / 5 > 255 ? 255 : iVGA_G * 6 / 5;
                oVGA_B = iVGA_B * 6 / 5 > 255 ? 255 : iVGA_B * 6 / 5;
            end else if (outCode == 8'b00100110) begin // 3; brightness level 3
                oVGA_R = iVGA_R * 7 / 5 > 255 ? 255 : iVGA_R * 7 / 5;
                oVGA_G = iVGA_G * 7 / 5 > 255 ? 255 : iVGA_G * 7 / 5;
                oVGA_B = iVGA_B * 7 / 5 > 255 ? 255 : iVGA_B * 7 / 5;
            end else if (outCode == 8'b00100101) begin // 4; brightness level 4
                oVGA_R = iVGA_R * 8 / 5 > 255 ? 255 : iVGA_R * 8 / 5;
                oVGA_G = iVGA_G * 8 / 5 > 255 ? 255 : iVGA_G * 8 / 5;
                oVGA_B = iVGA_B * 8 / 5 > 255 ? 255 : iVGA_B * 8 / 5;
            end else if (outCode == 8'b00101110) begin // 5; brightness level 5
                oVGA_R = iVGA_R * 9 / 5 > 255 ? 255 : iVGA_R * 9 / 5;
                oVGA_G = iVGA_G * 9 / 5 > 255 ? 255 : iVGA_G * 9 / 5;
                oVGA_B = iVGA_B * 9 / 5 > 255 ? 255 : iVGA_B * 9 / 5;
            end else if (outCode == 8'b00110110) begin // 6; brightness level 6
                oVGA_R = iVGA_R * 10 / 5 > 255 ? 255 : iVGA_R * 10 / 5;
                oVGA_G = iVGA_G * 10 / 5 > 255 ? 255 : iVGA_G * 10 / 5;
                oVGA_B = iVGA_B * 10 / 5 > 255 ? 255 : iVGA_B * 10 / 5;
            end else if (outCode == 8'b00111101) begin // 7; invert colors
                oVGA_R = 255 - iVGA_R;
                oVGA_G = 255 - iVGA_G;
                oVGA_B = 255 - iVGA_B;
            end else if (outCode == 8'b00111110) begin // 8; grayscale
                oVGA_R = (iVGA_R + iVGA_G + iVGA_B) / 3;
                oVGA_G = (iVGA_R + iVGA_G + iVGA_B) / 3;
                oVGA_B = (iVGA_R + iVGA_G + iVGA_B) / 3;
            end else if (outCode == 8'b01000110) begin // 9; increase contrast
                oVGA_R = (iVGA_R < 127) ? (iVGA_R * 3 / 5) : ((iVGA_R * 8 / 5) > 255 ? 255 : (iVGA_R * 8 / 5));
                oVGA_G = (iVGA_G < 127) ? (iVGA_G * 3 / 5) : ((iVGA_G * 8 / 5) > 255 ? 255 : (iVGA_G * 8 / 5));
                oVGA_B = (iVGA_B < 127) ? (iVGA_B * 3 / 5) : ((iVGA_B * 8 / 5) > 255 ? 255 : (iVGA_B * 8 / 5));
            end else begin // default
                oVGA_R = iVGA_R;
                oVGA_G = iVGA_G;
                oVGA_B = iVGA_B;
            end
        end else begin // no keys are pressed
                oVGA_R = iVGA_R;
                oVGA_G = iVGA_G;
                oVGA_B = iVGA_B;
        end
        oVGA_HS = iVGA_HS;
        oVGA_VS = iVGA_VS;
        oVGA_SYNC_N = iVGA_SYNC_N;
        oVGA_BLANK_N = iVGA_BLANK_N;
    end

    assign HEX0 = '1;
    assign HEX1 = '1;
    assign HEX2 = '1;
    assign HEX3 = '1;
    assign HEX4 = '1;
    assign HEX5 = '1;
    assign LEDR = '0;
endmodule

module Filter_tb();
    logic clk;

    // VGA in
    logic [7:0] iVGA_B, iVGA_G, iVGA_R;
    logic iVGA_HS, iVGA_VS, iVGA_SYNC_N, iVGA_BLANK_N;

    // VGA out
    logic [7:0] oVGA_B, oVGA_G, oVGA_R;
    logic oVGA_HS, oVGA_VS, oVGA_SYNC_N, oVGA_BLANK_N;

    // *** Board outputs ***
    logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
    logic [9:0] LEDR;

    // *** User inputs ***
    logic [1:0] KEY;
    logic [8:0] SW;
    logic [9:0] keyboardNum;
    logic makeBreak;
    logic [7:0] outCode;
```

```
    Filter dut (.VGA_CLK(clk), .*);

    // Set up the clock.
    parameter CLOCK_PERIOD = 100;
    initial begin
        clk <= 0;
        forever #(CLOCK_PERIOD/2) clk <= ~clk;
    end

    initial begin
        iVGA_B <= 200; iVGA_G <= 200; iVGA_R <= 200; makeBreak <= 0; @(posedge clk);
        makeBreak <= 1; outCode <= 8'b00010110; @(posedge clk);
        outCode <= 8'b00011110; @(posedge clk);
        outCode <= 8'b00100110; @(posedge clk);
        outCode <= 8'b00100101; @(posedge clk);
        outCode <= 8'b00101110; @(posedge clk);
        outCode <= 8'b00110110; @(posedge clk);
        outCode <= 8'b00111101; @(posedge clk);
        outCode <= 8'b00111110; @(posedge clk);
        outCode <= 8'b01000110; @(posedge clk);
        outCode <= 0; @(posedge clk);
        $stop;
    end
endmodule
```

Code 1: Filter.sv

```
    Filter #(.WIDTH(640), .HEIGHT(480))
       filter (.VGA_CLK(VGA_CLK),
                .iVGA_B(pre_VGA_B), .iVGA_G(pre_VGA_G), .iVGA_R(pre_VGA_R),
                .iVGA_HS(pre_VGA_HS), .iVGA_VS(pre_VGA_VS),
                .iVGA_SYNC_N(pre_VGA_SYNC_N), .iVGA_BLANK_N(pre_VGA_BLANK_N),
                .oVGA_B(post_VGA_B), .oVGA_G(post_VGA_G), .oVGA_R(post_VGA_R),
                .oVGA_HS(post_VGA_HS), .oVGA_VS(post_VGA_VS),
                .oVGA_SYNC_N(post_VGA_SYNC_N), .oVGA_BLANK_N(post_VGA_BLANK_N),
                .HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2), .HEX3(HEX3), .HEX4(HEX4), .HEX5(HEX5),
                .LEDR(KEDR), .KEY(KEY[1:0]), .keyboardNum(keyboardNum), .SW(SW), .outCode(outCode),
                .makeBreak(makeBreak));

    assign VGA_BLANK_N = post_VGA_BLANK_N;
    assign VGA_B = post_VGA_B;
    assign VGA_G = post_VGA_G;
    assign VGA_HS = post_VGA_HS;
    assign VGA_R = post_VGA_R;
    assign VGA_SYNC_N = post_VGA_SYNC_N;
    assign VGA_VS = post_VGA_VS;
    wire [9:0] keyboardNum;
    wire valid, makeBreak;
    wire [7:0] outCode;
    assign LEDR = keyboardNum;
    keyboard_press_driver d (.CLOCK_50(CLOCK_50), .valid(valid), .makeBreak(makeBreak),
    .outCode(outCode), .PS2_DAT(PS2_DAT), .PS2_CLK(PS2_CLK), .reset(0));

    keyboard k (.reset(0), .CLOCK_50(CLOCK_50),
    .PS2_DAT(PS2_DAT), .PS2_CLK(PS2_CLK), .keyboardNum(keyboardNum));

// LED_select leds (.outCode(outCode), .LEDs(GPIO[35:27]));

    always begin
       GPIO[35:27] = 0;
       if (outCode == 8'b00010110) begin // key 1
          GPIO[35] = 1;
       end else if (outCode == 8'b00011110) begin // key 2
          GPIO[34] = 1;
       end else if (outCode == 8'b00100110) begin // key 3
          GPIO[33] = 1;
       end else if (outCode == 8'b00100101) begin // key 4
          GPIO[32] = 1;
       end else if (outCode == 8'b00101110) begin // key 5
          GPIO[31] = 1;
       end else if (outCode == 8'b00110110) begin // key 6
          GPIO[30] = 1;
       end else if (outCode == 8'b00111101) begin // key 7
          GPIO[29] = 1;
       end else if (outCode == 8'b00111110) begin // key 8
          GPIO[28] = 1;
       end else if (outCode == 8'b01000110) begin // key 9
          GPIO[27] = 1;
       end
    end
```

Code 2: DE1_SOC_D8M_RTL.v (implementation of Filter and LED selection only)