DIGITAL SYSTEM DESIGN APPLICATIONS

(CRN: 11275)

THE REPORT OF EXPERIMENT - 8

Faculty of Electrical and Electronics Engineering

Electronics and Communication Engineering

Yusuf Tekin - 040200043

Yusuf Tekin
040200043

1. Conv_unit

```
`timescale 1ns / 1ps
module conv_unit(
    input pixel_clk,
    input rst,
    input enable,
    input [11:0] pixel1,
    input [11:0] pixel2,
    input [11:0] pixel3,
    input [11:0] kernel1,
    input [11:0] kernel2,
    input [11:0] kernel3,
    output reg [3:0] pixel_out
    );

    wire signed [8:0] mult1_1, mult1_2, mult1_3, mult2_1, mult2_2, mult2_3, mult3_1, mult3_2, mult3_3;
    wire [12:0] product;
    wire signed [4:0] pixel11, pixel12, pixel13, pixel21, pixel22, pixel23, pixel31, pixel32, pixel33;
    wire signed [3:0] kernel11, kernel12, kernel13, kernel21, kernel22, kernel23, kernel31, kernel32, kernel33;

    assign pixel11 = {1'b0, pixel1[3:0]};
    assign pixel12 = {1'b0, pixel1[7:4]};
    assign pixel13 = {1'b0, pixel1[11:8]};
    assign pixel21 = {1'b0, pixel2[3:0]};
    assign pixel22 = {1'b0, pixel2[7:4]};
    assign pixel23 = {1'b0, pixel2[11:8]};
    assign pixel31 = {1'b0, pixel3[3:0]};
    assign pixel32 = {1'b0, pixel3[7:4]};
    assign pixel33 = {1'b0, pixel3[11:8]};

    assign kernel11 = kernel1[3:0];
    assign kernel12 = kernel1[7:4];
    assign kernel13 = kernel1[11:8];
    assign kernel21 = kernel2[3:0];
    assign kernel22 = kernel2[7:4];
    assign kernel23 = kernel2[11:8];
    assign kernel31 = kernel3[3:0];
    assign kernel32 = kernel3[7:4];
    assign kernel33 = kernel3[11:8];



    assign mult1_1 = pixel13 * kernel13;
    assign mult1_2 = pixel12 * kernel12;
    assign mult1_3 = pixel11 * kernel11;

    assign mult2_1 = pixel23 * kernel23;
    assign mult2_2 = pixel22 * kernel22;
    assign mult2_3 = pixel21 * kernel21;

    assign mult3_1 = pixel33 * kernel33;
    assign mult3_2 = pixel32 * kernel32;
    assign mult3_3 = pixel31 * kernel31;

    assign product = (((mult1_1 + mult1_2) + (mult1_3 + mult2_1)) + ((mult2_2 + mult2_3) + (mult3_1 + mult3_2))) + mult3_3;

    always @(posedge pixel_clk or posedge rst) begin
        if(rst) begin
            pixel_out <= 4'b0000;
        end else if(enable) begin
            if(product < 0) begin
                pixel_out <= 4'b0000;
            end else if(product > 15) begin
                pixel_out <= 4'b1111;
            end else begin
                pixel_out <= product[3:0];
            end
        end else begin
            pixel_out <= 4'b0000;
        end

    end
    endmodule
```

Table 1 - Convolution Unit Verilog Code

Yusuf Tekin
040200043

2. Controller Unit

```verilog
`timescale 1ns / 1ps
module controller(
    input pixel_clk,
    input rst,
    input enable,
    input [11:0] data_in,
    output reg done,
    output reg [16:0] address,
    output reg [11:0] kernel1,
    output reg [11:0] kernel2,
    output reg [11:0] kernel3,
    output reg [11:0] pixel1,
    output reg [11:0] pixel2,
    output reg [11:0] pixel3
    );

    parameter FIRST_LINE = 3'b000, SECOND_LINE = 3'b001,
              PROC1 = 3'b010, PROC2 = 3'b011, PROC3 = 3'b100,
              DONE = 3'b101;

    reg [2:0] state;
    reg [3:0] buffer1 [641:0];
    reg [3:0] buffer2 [641:0];
    reg [9:0] index;
    reg [11:0] previous_data;
    integer i;

    always @(posedge pixel_clk or posedge rst) begin
        if(rst) begin
            state <= FIRST_LINE;
            address <= 17'b0;
            done <= 1'b0;
            pixel1 <= 12'b0;
            pixel2 <= 12'b0;
            pixel3 <= 12'b0;
            index <= 10'b0;
            kernel1 <= 12'b1111_1111_1111; // -1
            kernel2 <= 12'b1111_1000_1111; // -1, 8, -1 (-113)
            kernel3 <= 12'b1111_1111_1111; // -1
            previous_data <= 12'b0;
            for (i=0; i<642; i = i +1) begin
                buffer1[i] <= 4'b0;
                buffer2[i] <= 4'b0;
            end
        end else if(enable) begin
            if(address == 103148) begin
                 state <= DONE;
            end
            case(state)
                FIRST_LINE: begin
                    if(index < 640) begin
                        buffer1[index] <= data_in[3:0];
                        buffer1[index + 1] <= data_in[7:4];
                        buffer1[index + 2] <= data_in[11:8];
                        index <= index + 3;
                        address <= address + 1;
                        state <= FIRST_LINE;
                    end else begin
                        index <= 10'b0;
                        state <= SECOND_LINE;
                    end
                end
                SECOND_LINE: begin
                    if(index < 640) begin
                        buffer2[index] <= data_in[3:0];
                        buffer2[index + 1] <= data_in[7:4];
                        buffer2[index + 2] <= data_in[11:8];
                        index <= index + 3;
                        address <= address + 1;
                        state <= SECOND_LINE;
                    end else begin
                        index <= 10'b0;
                        state <= PROC1;
                    end
                end
                PROC1: begin
                    pixel1 <= {buffer1[index],buffer1[index+1],buffer1[index+2]};
                    pixel2 <= {buffer2[index],buffer2[index+1],buffer2[index+2]};
                    pixel3 <= data_in;
                    buffer1[index] <= buffer2[index];
                    buffer2[index] <= data_in[3:0];
                    previous_data <= data_in;
                    index <= index +1;

                    if(index == 639) begin
                       state <= PROC1;
                       index <= 0;
                       address <= address + 1;
                    end else begin
                       address <= address + 1;
                       state <= PROC2;
                    end
                end
                PROC2: begin

                    pixel1 <= {buffer1[index],buffer1[index+1],buffer1[index+2]};
                    pixel2 <= {buffer2[index],buffer2[index+1],buffer2[index+2]};
                    pixel3 <= {previous_data[7:4],previous_data[11:8], data_in[3:0]};
                    buffer1[index] <= buffer2[index];
                    buffer2[index] <= previous_data[7:4];
                    index <= index +1;

                    if(index == 639) begin
                       state <= PROC1;
                       index <= 0;
                       address <= address + 1;
                    end else begin
                       state <= PROC3;
                    end
                end
                PROC3: begin
                    pixel1 <= {buffer1[index],buffer1[index+1],buffer1[index+2]};
                    pixel2 <= {buffer2[index],buffer2[index+1],buffer2[index+2]};
                    pixel3 <= {previous_data[11:8], data_in[3:0], data_in[7:4]};
                    buffer1[index] <= buffer2[index];
                    buffer2[index] <= previous_data[11:8];
                    index <= index + 1;

                    if(index == 639) begin
                       address <= address + 1;
                       state <= PROC1;
                       index <= 0;

                    end else begin
                       state <= PROC1;
                    end
                end
                DONE: begin
                    state <= DONE;
                    done <= 1;
                end
            endcase
        end
    end
endmodule
```

Table 2 - Controller Unit Verilog Code

3. Top_Module

```verilog
`timescale 1ns / 1ps


module top_module(
    input wire clk,
    input wire rst,
    output wire VGA_HS,
    output wire VGA_VS,
    output wire [3:0] VGA_R,
    output wire [3:0] VGA_G,
    output wire [3:0] VGA_B
    );

    wire clk_25MHz, data_en;
    reg [1:0] counter=0;
    assign clk_25MHz = counter[1];

    always @(posedge clk) begin
        counter <= counter + 1;
    end

    vga_driver VGA(
        .pixel_clk(clk_25MHz),
        .rst(rst),
        .VGA_HS(VGA_HS),
        .VGA_VS(VGA_VS),
        .data_en(data_en)
        );


 //-----------------------------------RED------------------------------

    wire [11:0] R_K1, R_K2, R_K3, R_P1, R_P2, R_P3;
    wire [16:0] R_addr;
    wire [11:0] R_data;

    blk_mem_red RAM_R(
        .clka(clk_25MHz),    // input wire clka
        .ena(data_en),      // input wire ena
        .wea(1'b0),      // input wire [0 : 0] wea
        .addra(R_addr),  // input wire [16 : 0] addra
        .dina(1'b0),    // input wire [11 : 0] dina
        .douta(R_data)
        );

    controller controller_R(
        .pixel_clk(clk_25MHz),
        .rst(rst),
        .enable(data_en),
        .data_in(R_data),
        .done(),
        .address(R_addr),
        .kernel1(R_K1),
        .kernel2(R_K2),
        .kernel3(R_K3),
        .pixel1(R_P1),
        .pixel2(R_P2),
        .pixel3(R_P3)
        );

    conv_unit conv_R(
        .pixel_clk(clk_25MHz),
        .rst(rst),
        .enable(data_en),
        .kernel1(R_K1),
        .kernel2(R_K2),
        .kernel3(R_K3),
        .pixel1(R_P1),
        .pixel2(R_P2),
        .pixel3(R_P3),
        .pixel_out(VGA_R)
        );

  //-----------------------------------GREEN------------------------------

    wire [11:0] G_K1, G_K2, G_K3, G_P1, G_P2, G_P3;
    wire [16:0] G_addr;
    wire [11:0] G_data;

    blk_mem_green RAM_G(
        .clka(clk_25MHz),    // input wire clka
        .ena(data_en),      // input wire ena
        .wea(1'b0),      // input wire [0 : 0] wea
        .addra(G_addr),  // input wire [16 : 0] addra
        .dina(1'b0),    // input wire [11 : 0] dina
        .douta(G_data)
        );

    controller controller_G(
        .pixel_clk(clk_25MHz),
        .rst(rst),
        .enable(data_en),
        .data_in(G_data),
        .done(),
        .address(G_addr),
        .kernel1(G_K1),
        .kernel2(G_K2),
        .kernel3(G_K3),
        .pixel1(G_P1),
        .pixel2(G_P2),
        .pixel3(G_P3)
        );

    conv_unit conv_G(
        .pixel_clk(clk_25MHz),
        .rst(rst),
        .enable(data_en),
        .kernel1(G_K1),
        .kernel2(G_K2),
        .kernel3(G_K3),
        .pixel1(G_P1),
        .pixel2(G_P2),
        .pixel3(G_P3),
        .pixel_out(VGA_G)
        );

 //-----------------------------------BLUE------------------------------

    wire [11:0] B_K1, B_K2, B_K3, B_P1, B_P2, B_P3;
    wire [16:0] B_addr;
    wire [11:0] B_data;

    blk_mem_blue RAM_B(
        .clka(clk_25MHz),    // input wire clka
        .ena(data_en),      // input wire ena
        .wea(1'b0),      // input wire [0 : 0] wea
        .addra(B_addr),  // input wire [16 : 0] addra
        .dina(1'b0),    // input wire [11 : 0] dina
        .douta(B_data)
        );

    controller controller_B(
        .pixel_clk(clk_25MHz),
        .rst(rst),
        .enable(data_en),
        .data_in(B_data),
        .done(),
        .address(B_addr),
        .kernel1(B_K1),
        .kernel2(B_K2),
        .kernel3(B_K3),
        .pixel1(B_P1),
        .pixel2(B_P2),
        .pixel3(B_P3)
        );

    conv_unit conv_B(
        .pixel_clk(clk_25MHz),
        .rst(rst),
        .enable(data_en),
        .kernel1(B_K1),
        .kernel2(B_K2),
        .kernel3(B_K3),
        .pixel1(B_P1),
        .pixel2(B_P2),
        .pixel3(B_P3),
        .pixel_out(VGA_B)
        );
endmodule
```

Table 3 - Top Module Verilog Code

```
`timescale 1ns / 1ns

module top_module_tb;

    // Clock and reset signals
    reg clk;
    reg rst;
    reg slw_clk;
    // Outputs from the DUT
    wire VGA_HS;
    wire VGA_VS;
    wire [3:0] VGA_R;
    wire [3:0] VGA_G;
    wire [3:0] VGA_B;

    // Instantiate the top_module
    top_module dut (
        .clk(clk),
        .rst(rst),
        .VGA_HS(VGA_HS),
        .VGA_VS(VGA_VS),
        .VGA_R(VGA_R),
        .VGA_G(VGA_G),
        .VGA_B(VGA_B)
    );

    // Clock generation (100 MHz)
    always #5 clk = ~clk;


    // File handling
    integer red_file, green_file, blue_file;
    integer i;

    // Test procedure
    initial begin
        // Initialize signals
        clk = 1'b0;
        rst = 1'b1;

        // Open output files
        red_file = $fopen("output_red.txt", "w");
        green_file = $fopen("output_green.txt", "w");
        blue_file = $fopen("output_blue.txt", "w");

        if (red_file == 0 || green_file == 0 || blue_file == 0) begin
            $display("ERROR: Could not open output files.");

        end

        // Apply reset
        #50 rst = 0;

        // Wait for some time to let the system run and process data
        //repeat (480 * 640) begin // Simulate 640x480 image
        for(i = 0; i <307200; i = i +1) begin
            $fwrite(red_file, "%d\n", VGA_R);    // Write Red channel to file
            $fwrite(green_file, "%d\n", VGA_G);  // Write Green channel to file
            $fwrite(blue_file, "%d\n", VGA_B);   // Write Blue channel to file
            #10;
        end

        $fclose(red_file);
        $fclose(green_file);
        $fclose(blue_file);

        $display("Simulation completed. Output files are ready.");

        // End simulation
        $finish();
    end
endmodule
```

Table 4 - Top Module Testbench Code

Yusuf Tekin
040200043

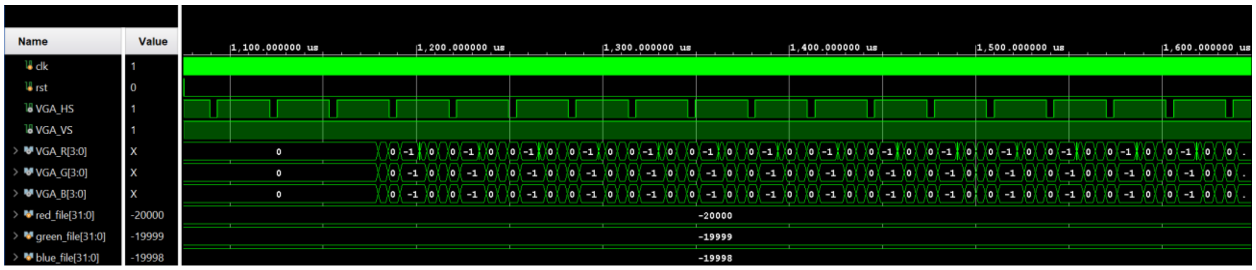


Figure 2 - Top Module Behavioral Simulation

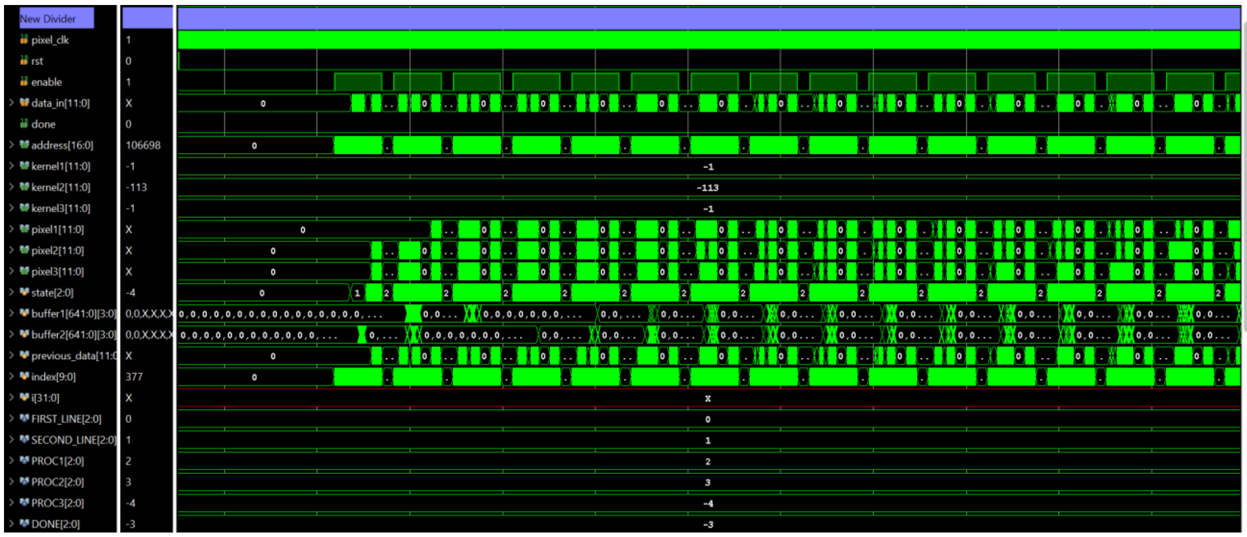


Figure 3 - Control Unit Behavioral Simulation

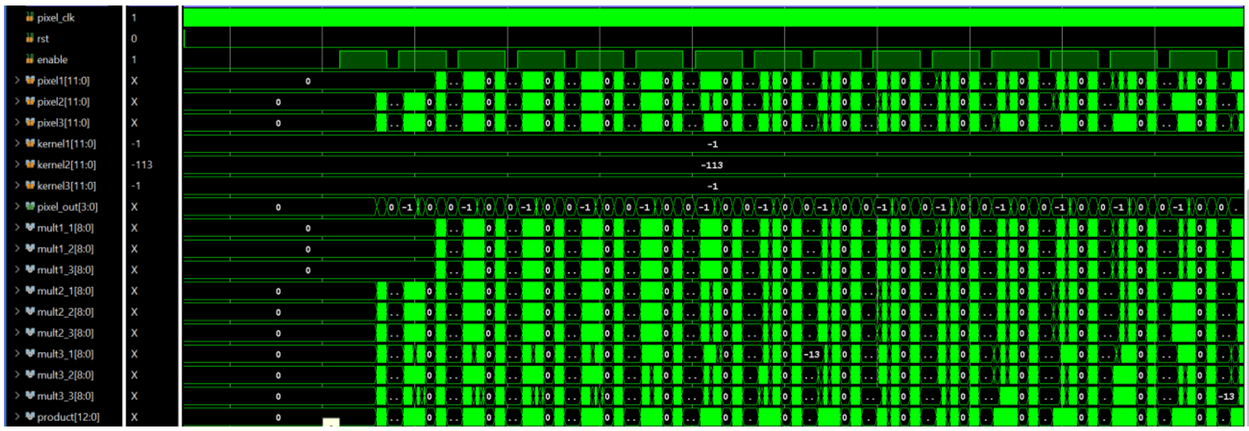


Figure 4 - Convolution Unit Behavioral Simulation

```
% Define the file names

r_file = 'output_red.txt';
g_file = 'output_green.txt';
b_file = 'output_blue.txt';

% Define the image size
width = 640;
height = 480;

% Read the R, G, B values from the text files
R = dlmread(r_file); % Reads all values into a vector
G = dlmread(g_file);
B = dlmread(b_file);

% Ensure the data matches the required dimensions
if length(R) ~= width * height || length(G) ~= width * height || length(B)
~= width * height
    error('The number of values in the files does not match the required
640x480 resolution.');
end

% Reshape the vectors into 2D matrices (height x width)
R_ = reshape(R, [width, height])'; % Transpose to get the correct
orientation
G = reshape(G, [width, height])';
B = reshape(B, [width, height])';

% Scale 4-bit values (0-15) to 8-bit values (0-255)
R_ = uint8(R_ * (255 / 15));
G = uint8(G * (255 / 15));
B = uint8(B * (255 / 15));

% Combine the R, G, B channels into an RGB image
RGB_image = cat(3, R_, G, B);

% Display the image
imshow(RGB_image);
title('RGB Image');

% Save the image to a file
imwrite(RGB_image, 'output_image.png');
disp('Image saved as output_image.png');
```

Table 5 - Output Image Checking MATLAB Code

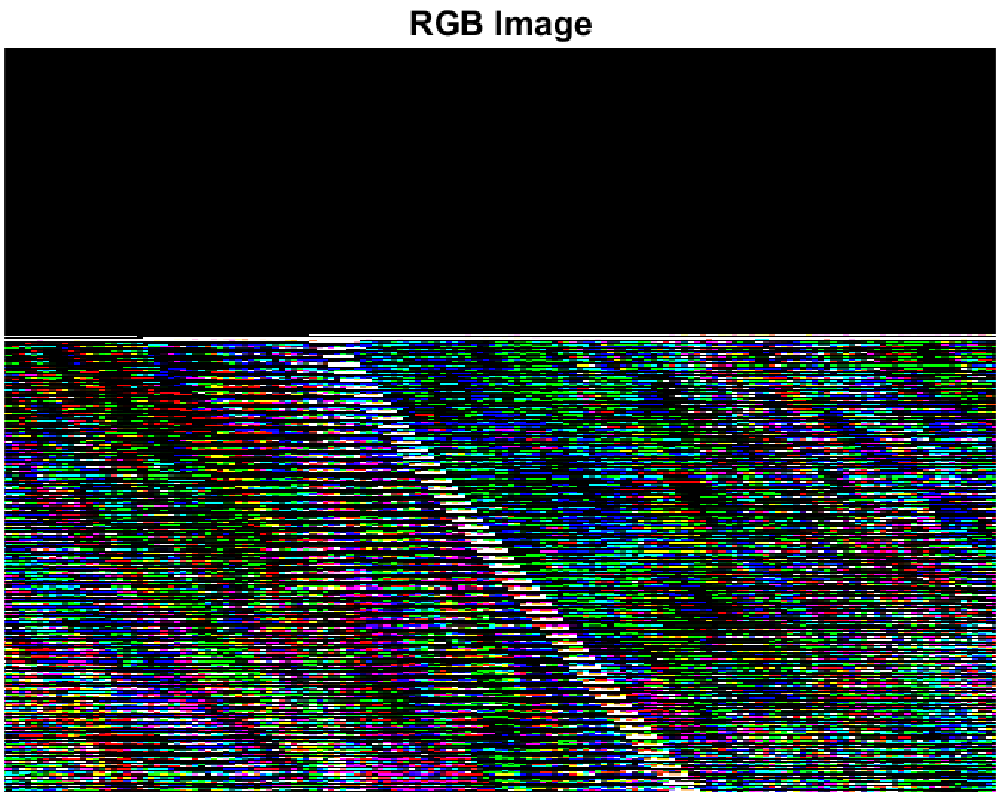
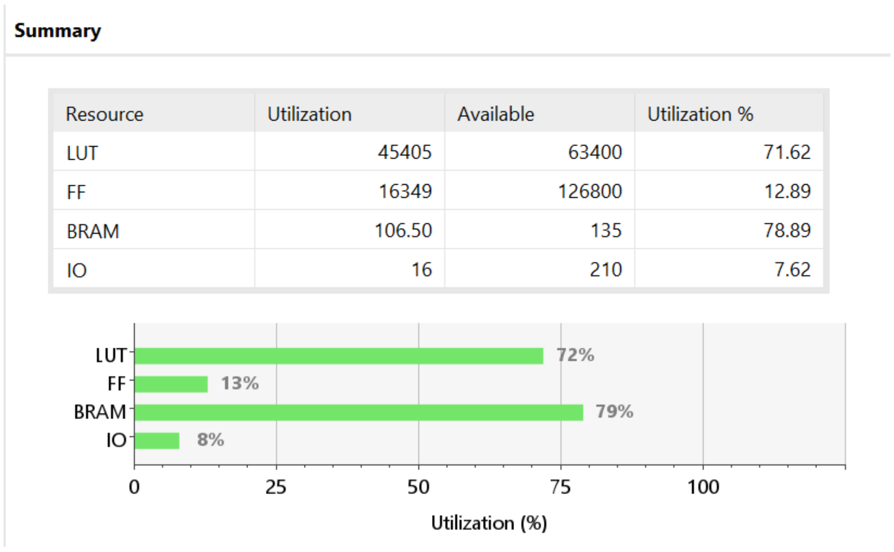


Figure 5 - MATLAB Output Image (Corrupted)



Summary

Resource	Utilization	Available	Utilization %
LUT	45405	63400	71.62
FF	16349	126800	12.89
BRAM	106.50	135	78.89
IO	16	210	7.62

Figure 6 - Utilization Summary

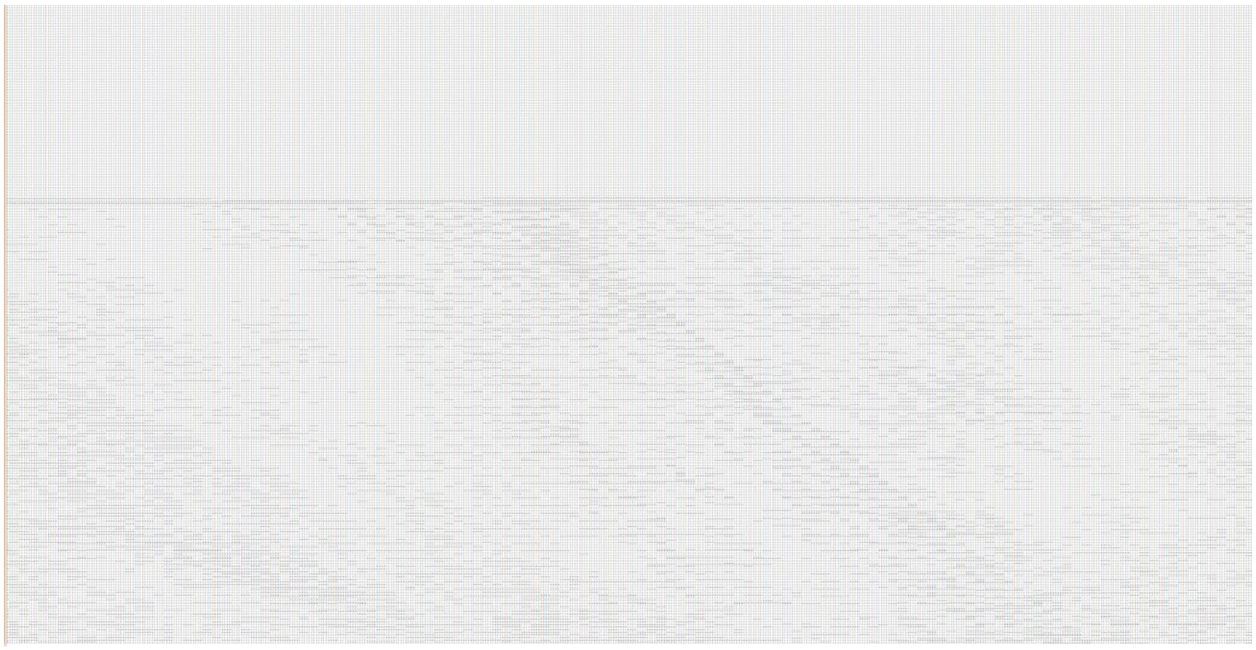
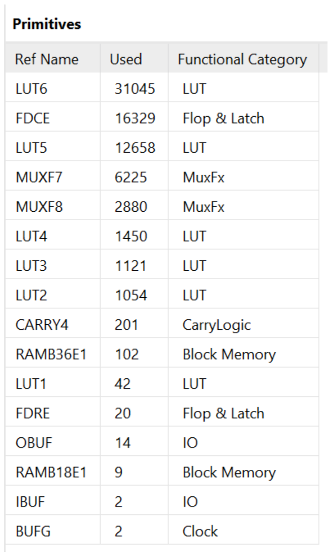


Figure 7 - Text Files in Line



Primitives

Ref Name	Used	Functional Category
LUT6	31045	LUT
FDCE	16329	Flop & Latch
LUT5	12658	LUT
MUXF7	6225	MuxFx
MUXF8	2880	MuxFx
LUT4	1450	LUT
LUT3	1121	LUT
LUT2	1054	LUT
CARRY4	201	CarryLogic
RAMB36E1	102	Block Memory
LUT1	42	LUT
FDRE	20	Flop & Latch
OBUF	14	IO
RAMB18E1	9	Block Memory
IBUF	2	IO
BUFG	2	Clock

Figure 8 - Primitives

Yusuf Tekin
040200043

Explanations

In the convolution unit, this design takes the inputs as 12-bit vectors and divides it by 3 to 4-bit for both pixels and kernels. After dividing the pixels, the fifth sign bit added to the divided pixel vectors but not the kernels. This division wires defined as signed bits for convolution process. Since the structure gives 13-bit product output after the convolution but it is needed to be 4-bit for the pixel_out, the output chosen to be the least significant 4-bit of the product. Later, the pixel_out is connected to VGA outputs in the top_module.

Block RAM is the most basic 1clk RAM from IP design. The initial values are given by the text files received from MATLAB.

In the control unit, the buffers serve a purpose as a memory block to store the first two lines of the image for the FIRST_LINE and SECOND_LINE states. However, after switching to PROC states, the processed values in buffer2 register to buffer1 and data_in to buffer2. The previous_data serves as a role to restore data_in values to be able to process the next states until the transition to the PROC1 from PROC3. This goes until the index == 639 because it is supposed to go 639 to make the [index + 2] equal to 642 which is the last pixel horizontally. However, the system does not shift vertically, it rather increases the address in PROC2. When the address == 642*482 the system goes to DONE state.