FPGA Image Processing Final Project Report

1. Introduction

This report describes the implementation of image processing using the FPGA DE1-SOC board and Quartus by coding in verilog (HDL). Our goal in the project was to successfully import a menu, select a preloaded RAM, and implement various effects such as inverting, red, red filter, green filter, blue filter, greyscale, changing the brightness, box blur, gaussian blur, sobel edge detection and embossing the image.

We believed this would be the best project for us because this project is a parallel project, i.e. if one of the modules isn't working (eg. greyscale), this doesn't stop us from working on the other modules (eg. brightness)and implementing them, which we believe is an important factor for efficiency and excellence of a project.

Moreover we wanted to do a project, which along with being complex, is also interesting. We were intrigued by the fact that a device like the DE1-SOC board can implement functionalities on an image like a photoshop tool.

Due to all these reasons, we believed image processing was the best project option for us.

2. Description of the Design

This part will explain inputs and outputs of our design on a high-level perspective. And some of the key components of the design including top-level module, control path, data path, and some of the key low-level modules.

2.1.1 Functions:

1) To process a 160x120 pixel, 8-bit-colour image to yield filter effects, for example, Gaussian blur, box blur, RGB filters,
2) Be able to process the user inputs (i.e. control signals from DE1-SOC board) and respond accordingly

2.1.2 Objectives:

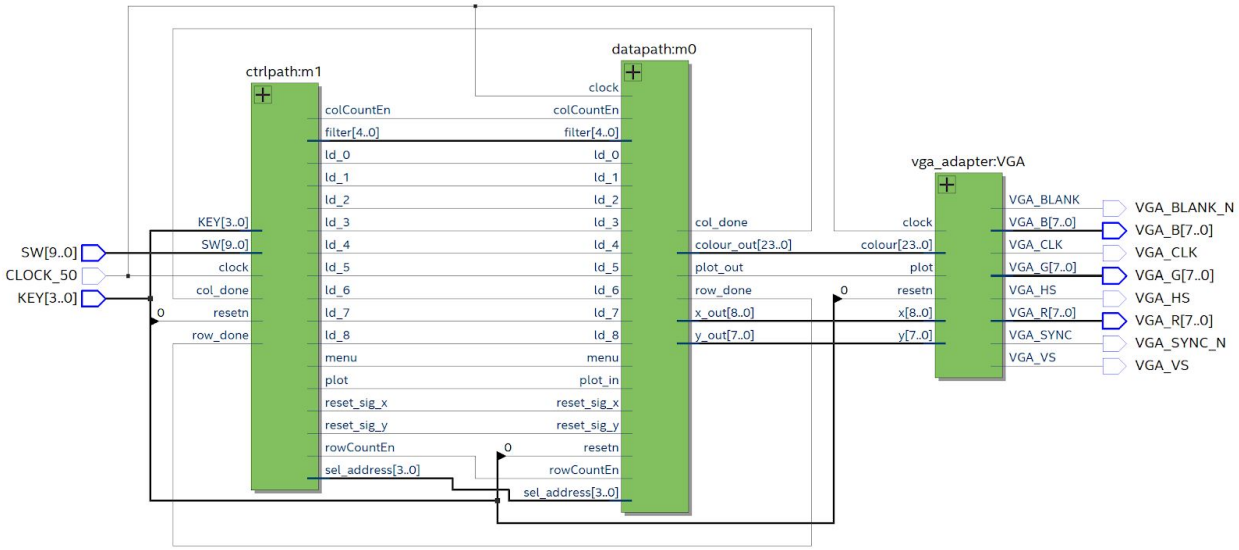
1)To use 9 registers to read the centre pixel (based on x, y counter outputs) and the eight neighbouring pixels in order to implement effects on a 3x3 kernel of the image to yield filter effects such as Gaussian blur, box blur, Sobel edge detection, and emboss.
2) To display the after effect using on a monitor using a VGA cable.

3) Inputting specific combinations of switches and keys on the DE1-SOC board to effectively apply the mentioned filters on the original image and revert back to the original image.

2.1.3 Constraints:

1) FPGA memory only allowed us to fit two 8-bit-coloured images and one 3-bit-coloured menu.
2) We had to use nine clock cycles to read 9 pixels of data from the RAM. (24x19200)

2.2 Top-level module Display:



The top level modules includes three main components: ctrl path:m0, data path:m1, and vga_adpator:VGA. The VGA adaptor module generally has 4 major inputs: x coordinate, y coordinate, 24-bit colour, and plot signal.

2.3 Data path (see appendix A):

1. a 8-bit x counter and a 7-bit y counter
    a. to traverse through the 320x240 resolution VGA display and display the processed 160x120 image at the centre of the VGA display,
2. Nine 24-bit registers for storing the pixel data
    a. Each register has an enable signal (i.e. ld_#) that the control path controls.
    b. Register 0 - 8 will store the pixel data read from image RAM in the following form. (#4 is the pixel which process and draw):

r0	r1	r2
r3	r4	r5
r6	r7	r8

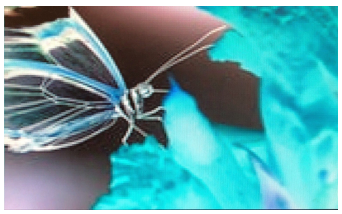
3. *Image processing module
    a. Core of the design, responsible for taking the 24-bit coloured data from the registers and process into one 24-bit colour data.
    b. Contains different sub-modules for filter effects.
    c. Outputs the processed 24-bit colour data to VGA adaptor module.
4. Address adaptor
    a. Takes the x, y coordinates and sel_address as inputs and outputs altered pixel locations in the 3x3 kernel

2.4 Control path (see AppendixB):

1. ld# signals – enables different registers to store pixel data .
2. Row_done – Triggered after we have traversed through all the pixels of the row.
3. Col_done – Triggered after we have traversed through all the pixels of the column.
4. Sel_address – controls which pixel is read from the RAM in the kernel.

3. Report of success

ORIGINAL IMAGE



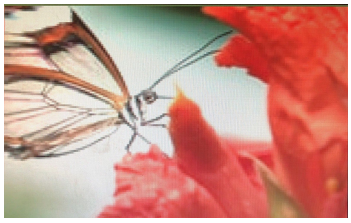
INVERTING

GREEN FILTER

RED FILTER

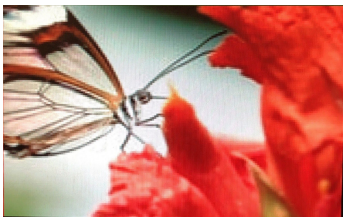
BLUE FILTER

GREYSCALE

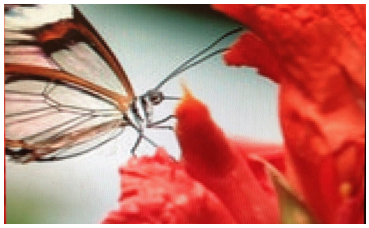


BRIGHTNESS INCREASE

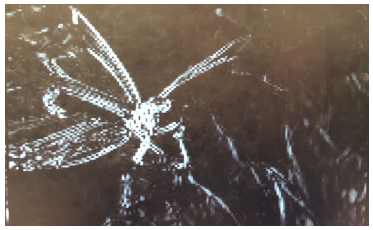
BRIGHTNESS DECREASE



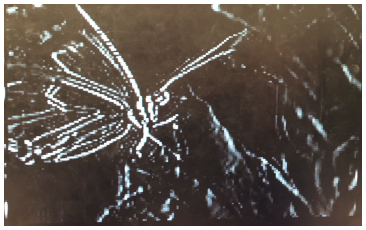
BOX BLUR



GAUSSIAN BLUR



SOBEL EDGE DETECTION



EMBOSSING

Parts That Didn't Work:

We were considering connecting a video camera to our project to implement the effects on them. But, another group who was doing a similar project ended up doing that before us. So we decided, instead of using a camera, to implement complex functionalities such as gaussian blur, edge detection, embossing etc

4. Lessons Learned

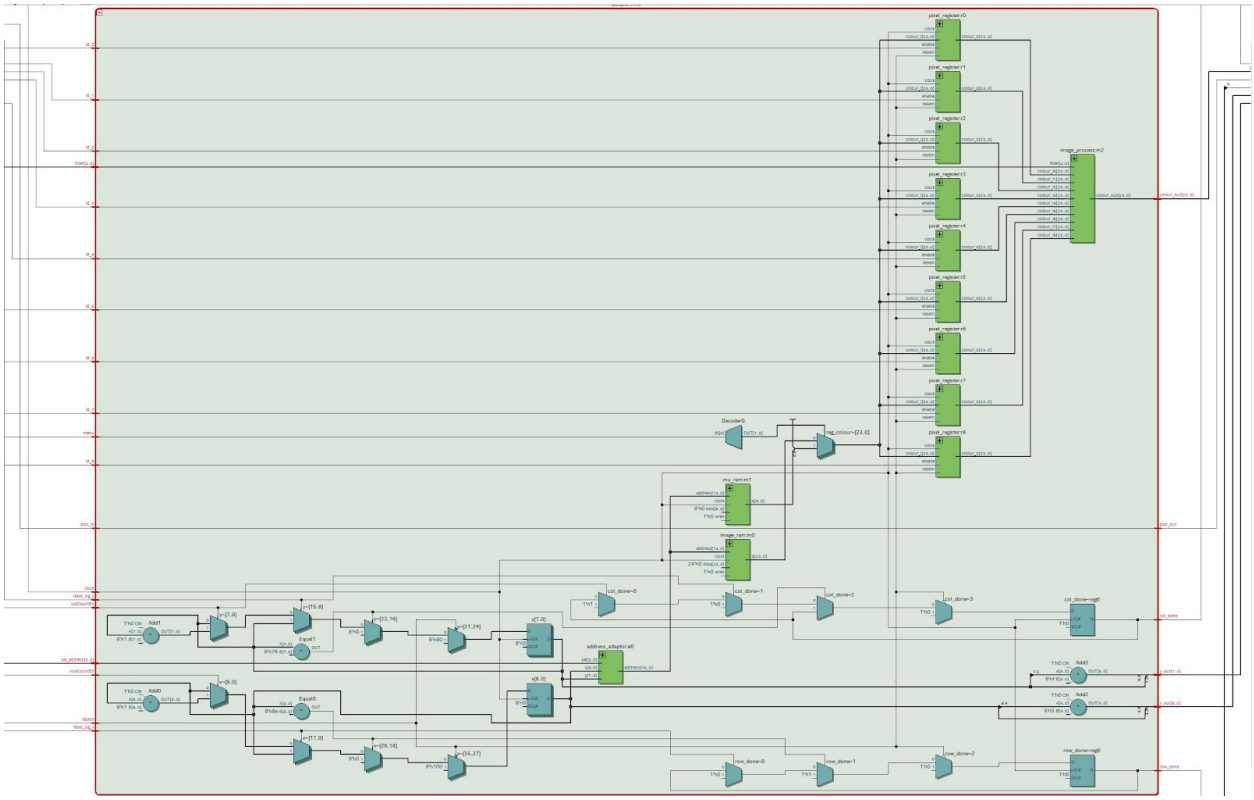
The main factor which messed up our time management in this project was disorganisation of code and improper planning on the long term by not utilizing github in the early phase of the project development.

After typing the code, sometimes a module which was previously working, wouldn't work after certain changes were implemented. As a result, we would have to debug a previously working code. If we had used github where we would keep versions of our code which we would update after every major change, a lot of time would've been saved, and it would also create a clean repository of all our progress.

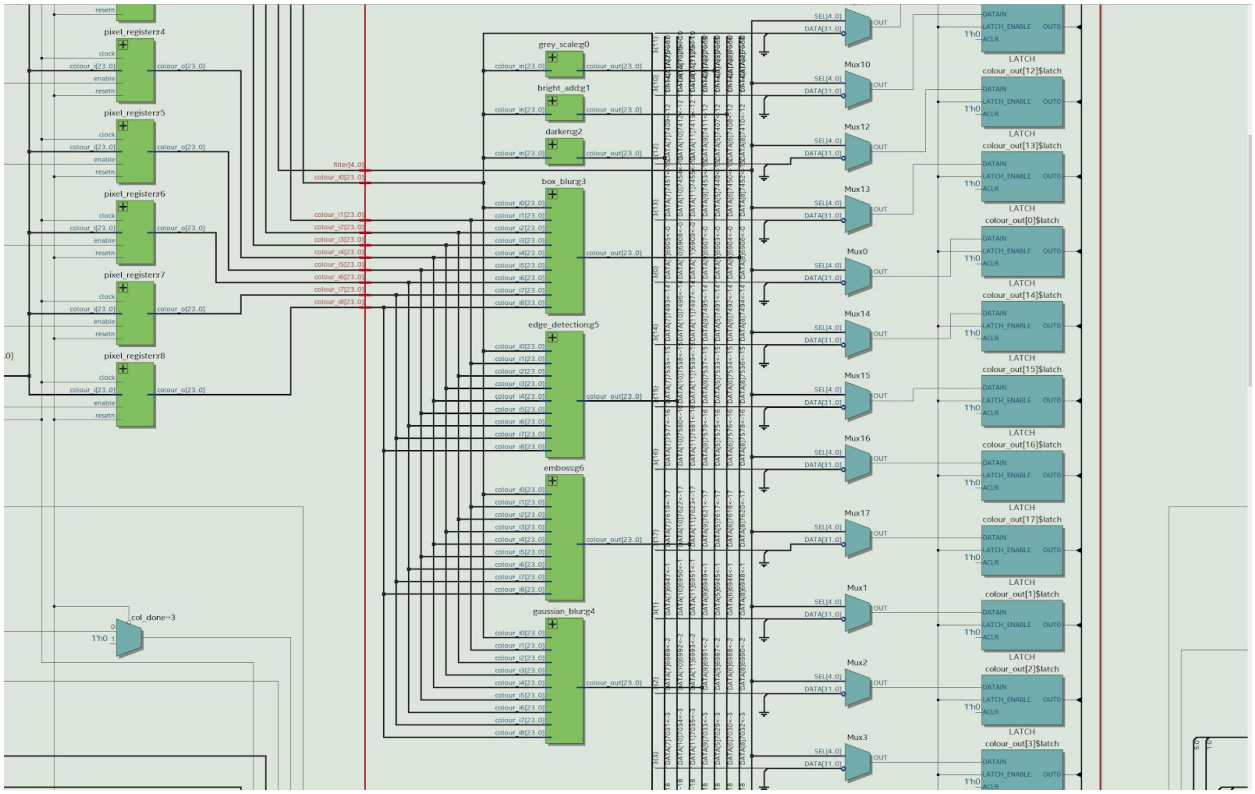
Secondly, once the image was displayed, whenever any change was done in the code, we used to compile and check the output from the VGA cable which would take around 3 and a half minutes. Instead, if we had used modelsim for faster compilation, along with saving time, it would also help us to easily detect what part of the code is creating an error by individually and collectively simulating the modules.

Third, we would use resources like emailing the TA and piazza more often. A few times, we were overcomplicating simple ideas or thinking from the wrong point of view just for the TA to tell us a simple idea to do this project thus pointlessly wasting our time.

Appendix



Appendix A (Data path module)



Appendix B (Image processing module)

Verilog code:

**********Display.v**********
****

```
module display
	(
		CLOCK_50,				//	On Board 50 MHz
		// Your inputs and outputs here
		KEY, SW,	// On Board Keys
		// The ports below are for the VGA output.  Do not change.
		VGA_CLK,				//	VGA Clock
		VGA_HS,					//	VGA H_SYNC
		VGA_VS,					//	VGA V_SYNC
		VGA_BLANK_N,				//	VGA BLANK
		VGA_SYNC_N,				//	VGA SYNC
		VGA_R,				//	VGA Red[9:0]
		VGA_G,					//	VGA
Green[9:0]
		VGA_B				//	VGA Blue[9:0]
	);
```

```
	input			CLOCK_50;				//	50 MHz
	input	[3:0]	KEY;
	input [9:0] SW;
	// Declare your inputs and outputs here
	// Do not change the following outputs
	output			VGA_CLK;				//	VGA Clock
	output			VGA_HS;					//	VGA H_SYNC
	output			VGA_VS;					//	VGA V_SYNC
	output			VGA_BLANK_N;				//	VGA BLANK
	output			VGA_SYNC_N;				//	VGA SYNC
	output	[7:0]	VGA_R;				//	VGA Red[7:0] Changed from
10 to 23-bit DAC
	output	[7:0]	VGA_G;					//	VGA Green[7:0]
	output	[7:0]	VGA_B;				//	VGA Blue[7:0]

	wire resetn;
	assign resetn = KEY[0];

	// Create the colour, x, y and writeEn wires that are inputs to the controller.

	wire [23:0] colour;
	wire [8:0] x;
	wire [7:0] y;
	wire writeEn;

	// Create an Instance of a VGA controller - there can be only one!
	// Define the number of colours as well as the initial background
	// image file (.MIF) for the controller.
	vga_adapter VGA(
			.resetn(resetn),
			.clock(CLOCK_50),
			.colour(colour),
			.x(x),
			.y(y),
			.plot(writeEn),
			/* Signals for the DAC to drive the monitor. */
			.VGA_R(VGA_R),
			.VGA_G(VGA_G),
			.VGA_B(VGA_B),
			.VGA_HS(VGA_HS),
			.VGA_VS(VGA_VS),
			.VGA_BLANK(VGA_BLANK_N),
```

```
            .VGA_SYNC(VGA_SYNC_N),
            .VGA_CLK(VGA_CLK));
        defparam VGA.RESOLUTION = "320x240";
        defparam VGA.MONOCHROME = "FALSE";
        defparam VGA.BITS_PER_COLOUR_CHANNEL = 8;
        defparam VGA.BACKGROUND_IMAGE = "black.mif";

// Put your code here. Your code should produce signals x,y,colour and writeEn
// for the VGA controller, in addition to any other functionality your design may require.

wire plot, rowCountEn, colCountEn, reset_sig_x, reset_sig_y, row_done, col_done;
wire ld_0, ld_1, ld_2, ld_3, ld_4, ld_5, ld_6, ld_7, ld_8;
wire [2:0] sel_im;
wire [4:0] filter;
wire [3:0] sel_address;

datapath2 m0(.clock(CLOCK_50),
                .resetn(resetn),
                .rowCountEn(rowCountEn),
                .colCountEn(colCountEn),
                .plot_in(plot),
                .filter(filter),
                .reset_sig_x(reset_sig_x),
                .reset_sig_y(reset_sig_y),
                .ld_0(ld_0),
                .ld_1(ld_1),
                .ld_2(ld_2),
                .ld_3(ld_3),
                .ld_4(ld_4),
                .ld_5(ld_5),
                .ld_6(ld_6),
                .ld_7(ld_7),
                .ld_8(ld_8),
                .sel_address(sel_address),
                .sel_im(sel_im),
                .row_done(row_done),
                .col_done(col_done),
                .x_out(x),
                .y_out(y),
                .colour_out(colour),
                .plot_out(writeEn));
```

```
	ctrlpath m1(.clock(CLOCK_50),
					.resetn(resetn),
					.SW(SW[9:0]),
					.KEY(KEY[3:0]),
					.row_done(row_done),
					.col_done(col_done),
					.rowCountEn(rowCountEn),
					.colCountEn(colCountEn),
					.plot(plot),
					.reset_sig_x(reset_sig_x),
					.reset_sig_y(reset_sig_y),
					.filter(filter),
					.ld_0(ld_0),
					.ld_1(ld_1),
					.ld_2(ld_2),
					.ld_3(ld_3),
					.ld_4(ld_4),
					.ld_5(ld_5),
					.ld_6(ld_6),
					.ld_7(ld_7),
					.ld_8(ld_8),
					.sel_address(sel_address),
					.sel_im(sel_im));


endmodule



/*
0 1 2
3 4 5
6 7 8
*/

// Data path2 for Guassian blur, Sobel edge detection
module datapath2(
	input clock, resetn,
	input rowCountEn,
	input colCountEn,
	input plot_in,
	input reset_sig_x,
	input reset_sig_y,
```

```
input ld_0, ld_1, ld_2, ld_3, ld_4, ld_5, ld_6, ld_7, ld_8, // register enable signals
input [2:0]sel_im,
input [3:0]sel_address,
input [4:0]filter,
output reg row_done,
output reg col_done,
output [8:0]x_out,
output [7:0]y_out,
output [23:0]colour_out,
output plot_out);

reg [8:0]x;
reg [7:0]y;
wire [23:0]c0, c1, c2, c3, c4, c5, c6, c7, c8;
wire [16:0]address;
reg [23:0]ram_colour;
wire [23:0]colour_im, colour_im2;
wire [8:0]colour_mu;


// x-8bit counter
always@(posedge clock)begin
	if(!resetn)begin
		x <= 9'd1;
		row_done = 1'b0;
	end
	else if(x == 9'd158)begin
		x <= 9'd0;
		row_done = 1'b1;
	end
	else if(reset_sig_x)begin
		row_done = 1'b0;
	end
	else if(rowCountEn)begin
		x <= x + 1'b1;
	end
	else
		x <= x;
end

// y-7bit counter
always@(posedge clock)begin
	if(!resetn)begin
```

```
                y <= 8'd1;
                col_done = 1'b0;
            end
            else if(y == 8'd118)begin
                y <= 8'd0;
            end
            else if(reset_sig_y)begin
                col_done = 1'b0;
            end
            else if(colCountEn)begin
                y <= y + 1'b1;
                col_done = 1'b1;
            end
            else
                y <= y;
        end



        //      input x, y count into the address_adaptor
        address_adaptor a0(.x(x), .y(y), .sel(sel_address), .address(address));

        // read the colour from the ram and store in ram_colour wire
        image_ram m0(.address(address), .clock(clock), .data(24'd0), .wren(1'b0),
.q(colour_im));

        image2_ram m1(.address(address), .clock(clock), .data(24'd0), .wren(1'b0),
.q(colour_im2));

        mu_ram m2(.address(address), .clock(clock), .data(24'd0), .wren(1'b0), .q(colour_mu));

        // case statement for select read colour from which RAMs
        always@(*)begin
            case(sel_im)
            3'd0: ram_colour = colour_im; //000
            3'd1: ram_colour = colour_im2; //001
            3'd2: ram_colour = {15'd0, colour_mu}; //010
            default: ram_colour = colour_im;
            endcase
        end



        // 9 registers while will hold the 9 pixels of colour
```

```
	pixel_register r0(.resetn(resetn), .clock(clock), .enable(ld_0), .colour_i(ram_colour),
.colour_o(c0));
	pixel_register r1(.resetn(resetn), .clock(clock), .enable(ld_1), .colour_i(ram_colour),
.colour_o(c1));
	pixel_register r2(.resetn(resetn), .clock(clock), .enable(ld_2), .colour_i(ram_colour),
.colour_o(c2));
	pixel_register r3(.resetn(resetn), .clock(clock), .enable(ld_3), .colour_i(ram_colour),
.colour_o(c3));
	pixel_register r4(.resetn(resetn), .clock(clock), .enable(ld_4), .colour_i(ram_colour),
.colour_o(c4));
	pixel_register r5(.resetn(resetn), .clock(clock), .enable(ld_5), .colour_i(ram_colour),
.colour_o(c5));
	pixel_register r6(.resetn(resetn), .clock(clock), .enable(ld_6), .colour_i(ram_colour),
.colour_o(c6));
	pixel_register r7(.resetn(resetn), .clock(clock), .enable(ld_7), .colour_i(ram_colour),
.colour_o(c7));
	pixel_register r8(.resetn(resetn), .clock(clock), .enable(ld_8), .colour_i(ram_colour),
.colour_o(c8));

	// new image process module
	image_process m3(
		.filter(filter),
		.colour_i0(c0),
		.colour_i1(c1),
		.colour_i2(c2),
		.colour_i3(c3),
		.colour_i4(c4),
		.colour_i5(c5),
		.colour_i6(c6),
		.colour_i7(c7),
		.colour_i8(c8),
		.colour_out(colour_out));

	// direct assignments
	assign x_out = x + 80;
	assign y_out = y + 60;
	assign plot_out = plot_in;

endmodule


/*
0 1 2
```

```
3 4 5
6 7 8
*/

module address_adaptor(
	input [8:0]x,
	input [7:0]y,
	input [3:0]sel,
	output reg [16:0]address);

	wire[16:0] p0, p1, p2, p3, p4, p5, p6, p7, p8;
	assign p0 = 160*y + x - 161;
	assign p1 = 160*y + x - 160;
	assign p2 = 160*y + x - 159;
	assign p3 = 160*y + x - 1;
	assign p4 = 160*y + x;
	assign p5 = 160*y + x + 1;
	assign p6 = 160*y + x + 159;
	assign p7 = 160*y + x + 160;
	assign p8 = 160*y + x + 161;

	always@(*)begin
		case(sel[3:0])
			4'd0: address = p0;
			4'd1: address = p1;
			4'd2: address = p2;
			4'd3: address = p3;
			4'd4: address = p4;
			4'd5: address = p5;
			4'd6: address = p6;
			4'd7: address = p7;
			4'd8: address = p8;
		default: address = p0;
		endcase
	end

endmodule

// Control Path module
module ctrlpath(
	input clock, resetn,
	input [9:0]SW,
	input [3:0]KEY,
```

```
input row_done,
input col_done,
output reg rowCountEn,
output reg colCountEn,
output reg plot,
output reg reset_sig_x,
output reg reset_sig_y,
output reg ld_0, ld_1, ld_2, ld_3, ld_4, ld_5, ld_6, ld_7, ld_8,
output reg [3:0]sel_address,
output [2:0]sel_im,
output [4:0]filter);


reg [5:0] current_state, next_state;

localparam
        // states for reading pixels from the image
        S_IDLE = 6'd0,
        S_WAIT_KEY = 6'd1,
        S_LD0 = 6'd2,
        S_WAIT_C0 = 6'd3,
        S_LD1 = 6'd4,
        S_WAIT_C1 = 6'd5,
        S_LD2 = 6'd6,
        S_WAIT_C2 = 6'd7,
        S_LD3 = 6'd8,
        S_WAIT_C3 = 6'd9,
        S_LD4 = 6'd10,
        S_WAIT_C4 = 6'd11,
        S_LD5 = 6'd12,
        S_WAIT_C5 = 6'd13,
        S_LD6 = 6'd14,
        S_WAIT_C6 = 6'd15,
        S_LD7 = 6'd16,
        S_WAIT_C7 = 6'd17,
        S_LD8 = 6'd18,
        S_WAIT_C8 = 6'd19,
        S_DISPLAY = 6'd20,
        S_INCR_X = 6'd21,
        S_RESET_SIG = 6'd22,
        S_INCR_Y = 6'd23,
        S_WAIT_STABLE = 6'd24;
```

```
// State table
always@(*)begin

        case(current_state)


        S_IDLE: next_state = KEY[1] ? S_IDLE : S_WAIT_KEY;

                S_WAIT_KEY: next_state = KEY[1] ? S_LD0 : S_WAIT_C0;
                S_WAIT_C0: next_state =  S_LD0;

                S_LD0: next_state = S_WAIT_C1;
                S_WAIT_C1: next_state = S_LD1;

                S_LD1: next_state = S_WAIT_C2;
                S_WAIT_C2: next_state = S_LD2;

                S_LD2: next_state = S_WAIT_C3;
                S_WAIT_C3: next_state = S_LD3;

                S_LD3: next_state = S_WAIT_C4;
                S_WAIT_C4: next_state = S_LD4;

                S_LD4: next_state = S_WAIT_C5;
                S_WAIT_C5: next_state = S_LD5;

                S_LD5: next_state = S_WAIT_C6;
                S_WAIT_C6: next_state = S_LD6;

                S_LD6: next_state = S_WAIT_C7;
                S_WAIT_C7: next_state = S_LD7;

                S_LD7: next_state = S_WAIT_C8;
                S_WAIT_C8: next_state = S_LD8;

                S_LD8: next_state = S_WAIT_STABLE;
                S_WAIT_STABLE: next_state = S_DISPLAY;
                S_DISPLAY: next_state = S_INCR_X;
                S_INCR_X: next_state = row_done ? S_INCR_Y : S_LD0;
                S_INCR_Y: next_state = col_done ? S_IDLE : S_RESET_SIG;
                S_RESET_SIG: next_state = S_LD0;
                default: next_state = S_IDLE;
```

```
        endcase
end

// Output logic
always@(*)begin

        rowCountEn = 1'b0;
        colCountEn = 1'b0;
        plot = 1'b0;
        reset_sig_x = 1'b0;
        reset_sig_y = 1'b0;
        ld_0 = 1'b0;
        ld_1 = 1'b0;
        ld_2 = 1'b0;
        ld_3 = 1'b0;
        ld_4 = 1'b0;
        ld_5 = 1'b0;
        ld_6 = 1'b0;
        ld_7 = 1'b0;
        ld_8 = 1'b0;
        sel_address = 4'd0;

        case(current_state)
        S_IDLE:begin
        reset_sig_x = 1'b1;
        reset_sig_y = 1'b1;
        end


        // for drawing the pic
        S_LD0:begin
        ld_0 = 1'b1;
        sel_address = 4'd0;
        end


        S_LD1:begin
        ld_1 = 1'b1;
        sel_address = 4'd1;
        end


        S_LD2:begin
```

```
ld_2 = 1'b1;
sel_address = 4'd2;
end


S_LD3:begin
ld_3 = 1'b1;
sel_address = 4'd3;
end

S_LD4:begin
ld_4 = 1'b1;
sel_address = 4'd4;
end

S_LD5:begin
ld_5 = 1'b1;
sel_address = 4'd5;
end

S_LD6:begin
ld_6 = 1'b1;
sel_address = 4'd6;
end


S_LD7:begin
ld_7 = 1'b1;
sel_address = 4'd7;
end

S_LD8:begin
ld_8 = 1'b1;
sel_address = 4'd8;
end

S_DISPLAY:begin
plot = 1'b1;
end

S_INCR_X:begin
rowCountEn = 1'b1;
end
```

```
                S_RESET_SIG:begin
                reset_sig_x = 1'b1;
                reset_sig_y = 1'b1;
                end

                S_INCR_Y:begin
                colCountEn = 1'b1;
                end

                endcase
        end



        // Direct assignment
        assign filter = SW[4:0];
        assign sel_im = SW[9:7];

        // current_state registers
    always@(posedge clock)
    begin
        if(!resetn)
            current_state <= S_IDLE;
        else
            current_state <= next_state;
    end // state_FFS

endmodule


****************************************image_process.v****************************************
************************************************************************************************
// Image processing module
// SW[9:7] filter option
module image_process(
        input [4:0]filter,
        input [23:0]colour_i0,
        input [23:0]colour_i1,
        input [23:0]colour_i2,
        input [23:0]colour_i3,
        input [23:0]colour_i4,
        input [23:0]colour_i5,
```

```
input [23:0]colour_i6,
input [23:0]colour_i7,
input [23:0]colour_i8,
output reg[23:0]colour_out);


// Filter options (USER MANUAL!) SW[4:0]
localparam
	ORIGIN = 5'd0, // 00000
	INVERT = 5'd1, // 00001
	R_FILTER = 5'd2, // 00010
	G_FILTER = 5'd3, // 00011
	B_FILTER = 5'd4, // 00100
	GREY_SCALE = 5'd5, // 00101
	BRIGHTEN = 5'd6, // 00110
	DARKEN = 5'd7, //00111
	BLUR = 5'd8, // 01000
	GBLUR = 5'd9, // 01001
	SOBEL_EDGE = 5'd10, // 01010
	EMBOSS = 5'd11; // 01011


wire [23:0]colour_grey;
wire [23:0]bright, dark;
wire [23:0]box_blur, gaussian_blur, edge_detection, emboss;

always@(*)begin
	case(filter[4:0])
		ORIGIN: colour_out = colour_i0;
		INVERT: colour_out = ~colour_i0;
		R_FILTER: colour_out = {colour_i0[23:16], 16'd0};
		G_FILTER: colour_out = {8'd0, colour_i0[15:8], 8'd0};
		B_FILTER: colour_out = {16'd0, colour_i0[7:0]};
		GREY_SCALE: colour_out = colour_grey;
		BLUR: colour_out = box_blur;
		BRIGHTEN: colour_out = bright;
		DARKEN: colour_out = dark;
		GBLUR: colour_out = gaussian_blur;
		SOBEL_EDGE: colour_out = edge_detection;
		EMBOSS: colour_out = emboss;
	endcase
end
```

```
grey_scale g0(.colour_in(colour_i0), .colour_out(colour_grey));
bright_add g1(.colour_in(colour_i0), .colour_out(bright));
darken g2(.colour_in(colour_i0), .colour_out(dark));

box_blur g3(
        .colour_i0(colour_i0),
        .colour_i1(colour_i1),
        .colour_i2(colour_i2),
        .colour_i3(colour_i3),
        .colour_i4(colour_i4),
        .colour_i5(colour_i5),
        .colour_i6(colour_i6),
        .colour_i7(colour_i7),
        .colour_i8(colour_i8),
        .colour_out(box_blur));

gaussian_blur g4(
        .colour_i0(colour_i0),
        .colour_i1(colour_i1),
        .colour_i2(colour_i2),
        .colour_i3(colour_i3),
        .colour_i4(colour_i4),
        .colour_i5(colour_i5),
        .colour_i6(colour_i6),
        .colour_i7(colour_i7),
        .colour_i8(colour_i8),
        .colour_out(gaussian_blur));

edge_detection g5(
        .colour_i0(colour_i0),
        .colour_i1(colour_i1),
        .colour_i2(colour_i2),
        .colour_i3(colour_i3),
        .colour_i4(colour_i4),
        .colour_i5(colour_i5),
        .colour_i6(colour_i6),
        .colour_i7(colour_i7),
        .colour_i8(colour_i8),
        .colour_out(edge_detection));

emboss g6(
```

```
                .colour_i0(colour_i0),
                .colour_i1(colour_i1),
                .colour_i2(colour_i2),
                .colour_i3(colour_i3),
                .colour_i4(colour_i4),
                .colour_i5(colour_i5),
                .colour_i6(colour_i6),
                .colour_i7(colour_i7),
                .colour_i8(colour_i8),
                .colour_out(emboss));

endmodule



module grey_scale(
        input [23:0]colour_in,
        output [23:0]colour_out);

        wire [7:0]r_in, g_in, b_in;
        wire [7:0]r_out, g_out, b_out;

        assign r_in = colour_in[23:16];
        assign g_in = colour_in[15:8];
        assign b_in = colour_in[7:0];

        assign r_out   = (r_in   != 0) ? (299*r_in/1000)+(587*g_in/1000)+(114*b_in/1000) : 0;
        assign g_out = (g_in != 0) ? (299*r_in/1000)+(587*g_in/1000)+(114*b_in/1000) : 0;
        assign b_out  = (b_in  != 0) ? (299*r_in/1000)+(587*g_in/1000)+(114*b_in/1000) : 0;

        assign  colour_out = {r_out, g_out, b_out};

endmodule


module  bright_add    (
   input [23:0]colour_in,
         output [23:0]colour_out
         );

   parameter black = 0;
   parameter white = 255;
        parameter threshold = 126;
```

```
parameter value1 = 50;

wire [7:0]r_in, g_in, b_in;
assign r_in = colour_in[23:16];
assign g_in = colour_in[15:8];
assign b_in = colour_in[7:0];


reg [7:0]r_out, g_out, b_out;
reg [15:0] tempR1, tempG1, tempB1;




always@(*)
begin

        tempR1 = r_in + value1;


if (tempR1 > 255)
        r_out = white;
else
        r_out = tempR1;


        tempG1 = g_in + value1;


if (tempG1 > 255)
        g_out = white;
else
        g_out = tempG1;


        tempB1 = b_in + value1;


if (tempB1 > 255)
        b_out = white;
else
        b_out = tempB1;
```

```
        end

        assign  colour_out = {r_out, g_out, b_out};

endmodule

module  darken    (
   input [23:0]colour_in,
         output [23:0]colour_out
         );

   parameter black = 0;
   parameter white = 255;
        parameter threshold = 126;
        parameter value1 = 50;

        wire [7:0]r_in, g_in, b_in;
        assign r_in = colour_in[23:16];
        assign g_in = colour_in[15:8];
        assign b_in = colour_in[7:0];


        reg [7:0]r_out, g_out, b_out;
        reg [15:0] tempR1, tempG1, tempB1;




        always@(*)
        begin

                tempR1 = r_in - value1;
        if (tempR1[8] == 1)
        r_out = 0;
        else
        r_out = tempR1;

        tempG1 = g_in - value1;
        if (tempG1[8] == 1)
        g_out = 0;
        else
        g_out = tempG1;
```

```
        tempB1 = b_in - value1;
        if (tempB1[8] == 1)
        b_out = 0;
        else
        b_out = tempB1;


        end

        assign  colour_out = {r_out, g_out, b_out};

endmodule


/*
0 1 2
3 4 5
6 7 8

*/

module box_blur(
        input [23:0]colour_i0,
        input [23:0]colour_i1,
        input [23:0]colour_i2,
        input [23:0]colour_i3,
        input [23:0]colour_i4,
        input [23:0]colour_i5,
        input [23:0]colour_i6,
        input [23:0]colour_i7,
        input [23:0]colour_i8,
        output [23:0]colour_out);

        wire[7:0]red0, red1, red2, red3, red4, red5, red6, red7, red8;
        wire[7:0]green0, green1, green2, green3, green4, green5, green6, green7, green8;
        wire[7:0]blue0, blue1, blue2, blue3, blue4, blue5, blue6, blue7, blue8;

        wire[31:0] red_x, green_x, blue_x;

        assign red0 = colour_i0[23:16];
        assign red1 = colour_i1[23:16];
        assign red2 = colour_i2[23:16];
```

```
        assign red3 = colour_i3[23:16];
        assign red4 = colour_i4[23:16];
        assign red5 = colour_i5[23:16];
        assign red6 = colour_i6[23:16];
        assign red7 = colour_i7[23:16];
        assign red8 = colour_i8[23:16];

        assign green0 = colour_i0[15:8];
        assign green1 = colour_i1[15:8];
        assign green2 = colour_i2[15:8];
        assign green3 = colour_i3[15:8];
        assign green4 = colour_i4[15:8];
        assign green5 = colour_i5[15:8];
        assign green6 = colour_i6[15:8];
        assign green7 = colour_i7[15:8];
        assign green8 = colour_i8[15:8];

        assign blue0 = colour_i0[7:0];
        assign blue1 = colour_i1[7:0];
        assign blue2 = colour_i2[7:0];
        assign blue3 = colour_i3[7:0];
        assign blue4 = colour_i4[7:0];
        assign blue5 = colour_i5[7:0];
        assign blue6 = colour_i6[7:0];
        assign blue7 = colour_i7[7:0];
        assign blue8 = colour_i8[7:0];


        assign red_x = red0 + red1 + red2 + red3 + red4 + red5 + red6 + red7 + red8;
        assign green_x = green0 + green1 + green2 + green3 + green4 + green5 + green6 +
green7 + green8;
        assign blue_x = blue0 + blue1 + blue2 + blue3 + blue4 + blue5 + blue6 + blue7 + blue8;

        assign colour_out[23:16] = red_x / 9;
        assign colour_out[15:8] = green_x / 9;
        assign colour_out[7:0] = blue_x / 9;

endmodule


module gaussian_blur(
        input [23:0]colour_i0,
        input [23:0]colour_i1,
```

```
input [23:0]colour_i2,
input [23:0]colour_i3,
input [23:0]colour_i4,
input [23:0]colour_i5,
input [23:0]colour_i6,
input [23:0]colour_i7,
input [23:0]colour_i8,
output [23:0]colour_out);

wire[7:0]red0, red1, red2, red3, red4, red5, red6, red7, red8;
wire[7:0]green0, green1, green2, green3, green4, green5, green6, green7, green8;
wire[7:0]blue0, blue1, blue2, blue3, blue4, blue5, blue6, blue7, blue8;

wire[31:0] red_x, green_x, blue_x;

assign red0 = colour_i0[23:16];
assign red1 = colour_i1[23:16];
assign red2 = colour_i2[23:16];
assign red3 = colour_i3[23:16];
assign red4 = colour_i4[23:16];
assign red5 = colour_i5[23:16];
assign red6 = colour_i6[23:16];
assign red7 = colour_i7[23:16];
assign red8 = colour_i8[23:16];

assign green0 = colour_i0[15:8];
assign green1 = colour_i1[15:8];
assign green2 = colour_i2[15:8];
assign green3 = colour_i3[15:8];
assign green4 = colour_i4[15:8];
assign green5 = colour_i5[15:8];
assign green6 = colour_i6[15:8];
assign green7 = colour_i7[15:8];
assign green8 = colour_i8[15:8];

assign blue0 = colour_i0[7:0];
assign blue1 = colour_i1[7:0];
assign blue2 = colour_i2[7:0];
assign blue3 = colour_i3[7:0];
assign blue4 = colour_i4[7:0];
assign blue5 = colour_i5[7:0];
assign blue6 = colour_i6[7:0];
assign blue7 = colour_i7[7:0];
```

```
	assign blue8 = colour_i8[7:0];

	/*
	|0 1 2| 1 2 1
	|3 4 5| 2 4 2
	|6 7 8| 1 2 1
	*/
	assign red_x = red0 + 2*red1 + red2 + 2*red3 + 4*red4+ 2*red5+ red6 + 2*red7+ red8;
	assign green_x = green0 + 2*green1 + green2 + 2*green3 + 4*green4 + 2*green5 +
green6 + 2*green7 + green8;
	assign blue_x = blue0 + 2*blue1 + blue2 + 2*blue3 + 4*blue4 + 2*blue5 + blue6 +2*blue7
+ blue8;

	assign colour_out[23:16]= red_x/16;
	assign colour_out[15:8]= green_x/16;
	assign colour_out[7:0]= blue_x/16;

endmodule


// sobel edge detection
//	sobel edge detection

//	| 1 0 -1 |	| 1 2 1 |
//	| 2 0 -2 |	| 0 0 0 |
//	| 1 0 -1 |	|-1 -2 -1 |


module edge_detection(
	input [23:0]colour_i0,
	input [23:0]colour_i1,
	input [23:0]colour_i2,
	input [23:0]colour_i3,
	input [23:0]colour_i4,
	input [23:0]colour_i5,
	input [23:0]colour_i6,
	input [23:0]colour_i7,
	input [23:0]colour_i8,
	output [23:0]colour_out);

	wire[7:0]red0, red1, red2, red3, red4, red5, red6, red7, red8;
	wire[7:0]green0, green1, green2, green3, green4, green5, green6, green7, green8;
	wire[7:0]blue0, blue1, blue2, blue3, blue4, blue5, blue6, blue7, blue8;
```

```
        wire[31:0] red_x, blue_x;
        reg[31:0] green_x;

        assign red0 = colour_i0[23:16];
        assign red1 = colour_i1[23:16];
        assign red2 = colour_i2[23:16];
        assign red3 = colour_i3[23:16];
        assign red4 = colour_i4[23:16];
        assign red5 = colour_i5[23:16];
        assign red6 = colour_i6[23:16];
        assign red7 = colour_i7[23:16];
        assign red8 = colour_i8[23:16];

        assign green0 = colour_i0[15:8];
        assign green1 = colour_i1[15:8];
        assign green2 = colour_i2[15:8];
        assign green3 = colour_i3[15:8];
        assign green4 = colour_i4[15:8];
        assign green5 = colour_i5[15:8];
        assign green6 = colour_i6[15:8];
        assign green7 = colour_i7[15:8];
        assign green8 = colour_i8[15:8];

        assign blue0 = colour_i0[7:0];
        assign blue1 = colour_i1[7:0];
        assign blue2 = colour_i2[7:0];
        assign blue3 = colour_i3[7:0];
        assign blue4 = colour_i4[7:0];
        assign blue5 = colour_i5[7:0];
        assign blue6 = colour_i6[7:0];
        assign blue7 = colour_i7[7:0];
        assign blue8 = colour_i8[7:0];

//      | 1 0 -1 |        | 1 2 1 |
//      | 2 0 -2 |        | 0 0 0 |
//      | 1 0 -1 |        |-1 -2 -1 |
//   | 0 1 2 |
//   | 3 4 5 |
//   | 6 7 8 |
        assign red_x = red0 - red2 +2*red3 -2*red5 + red6 - red8;
        assign blue_x = red0 +2*red1 + red2 - red6 - 2*red7 - red8;
```

```
        always@(*)begin

         if(red_x > 1024 & blue_x > 1024)begin
                green_x = -(red_x + blue_x)/2;
   end else if(red_x > 1024 & blue_x < 1024)begin
     green_x = (-red_x  + blue_x)/2;
   end else if(red_x < 1024 & blue_x < 1024)begin
     green_x = (red_x + blue_x)/2;
   end else begin
     green_x = (red_x - blue_x)/2;
   end

        end

        assign colour_out[23:16] = green_x;
        assign colour_out[15:8] = green_x;
        assign colour_out[7:0] = green_x;


endmodule



module emboss(
 input [23:0]colour_i0,
 input [23:0]colour_i1,
 input [23:0]colour_i2,
 input [23:0]colour_i3,
 input [23:0]colour_i4,
 input [23:0]colour_i5,
 input [23:0]colour_i6,
 input [23:0]colour_i7,
 input [23:0]colour_i8,
 output reg [23:0]colour_out);

 wire[7:0]red0, red1, red2, red3, red4, red5, red6, red7, red8;
 wire[7:0]green0, green1, green2, green3, green4, green5, green6, green7, green8;
 wire[7:0]blue0, blue1, blue2, blue3, blue4, blue5, blue6, blue7, blue8;

 wire[31:0] red_x;

 assign red0 = colour_i0[23:16];
 assign red1 = colour_i1[23:16];
```

```
assign red2 = colour_i2[23:16];
assign red3 = colour_i3[23:16];
assign red4 = colour_i4[23:16];
assign red5 = colour_i5[23:16];
assign red6 = colour_i6[23:16];
assign red7 = colour_i7[23:16];
assign red8 = colour_i8[23:16];

assign green0 = colour_i0[15:8];
assign green1 = colour_i1[15:8];
assign green2 = colour_i2[15:8];
assign green3 = colour_i3[15:8];
assign green4 = colour_i4[15:8];
assign green5 = colour_i5[15:8];
assign green6 = colour_i6[15:8];
assign green7 = colour_i7[15:8];
assign green8 = colour_i8[15:8];

assign blue0 = colour_i0[7:0];
assign blue1 = colour_i1[7:0];
assign blue2 = colour_i2[7:0];
assign blue3 = colour_i3[7:0];
assign blue4 = colour_i4[7:0];
assign blue5 = colour_i5[7:0];
assign blue6 = colour_i6[7:0];
assign blue7 = colour_i7[7:0];
assign blue8 = colour_i8[7:0];

/*
|0 1 2| -2 -1 0
|3 4 5| -1 1 1
|6 7 8|  0 1 2
*/
 assign red_x = red4 + red5 -red3 -red1 + 2*red8 - 2*red0;

 always@(*)begin

                    if (red_x > 1280)  begin
                    colour_out<=0;
                    end
                    else begin
                    colour_out[23:16] <= red_x;
                    colour_out[15:8] <= red_x;
```

```
                colour_out[7:0] <= red_x;
                end
 end


 endmodule
```

**********register.v**********

```
// register
module pixel_register(
        input resetn, clock,
        input enable,
        input [23:0] colour_i,
        output reg [23:0]colour_o);

        always@(posedge clock)begin

                if(!resetn)
                        colour_o <= 23'd0;
                else if(enable)
                        colour_o <= colour_i;
                else
                        colour_o <= colour_o;

        end

endmodule
```