Bora Kıran

040210069

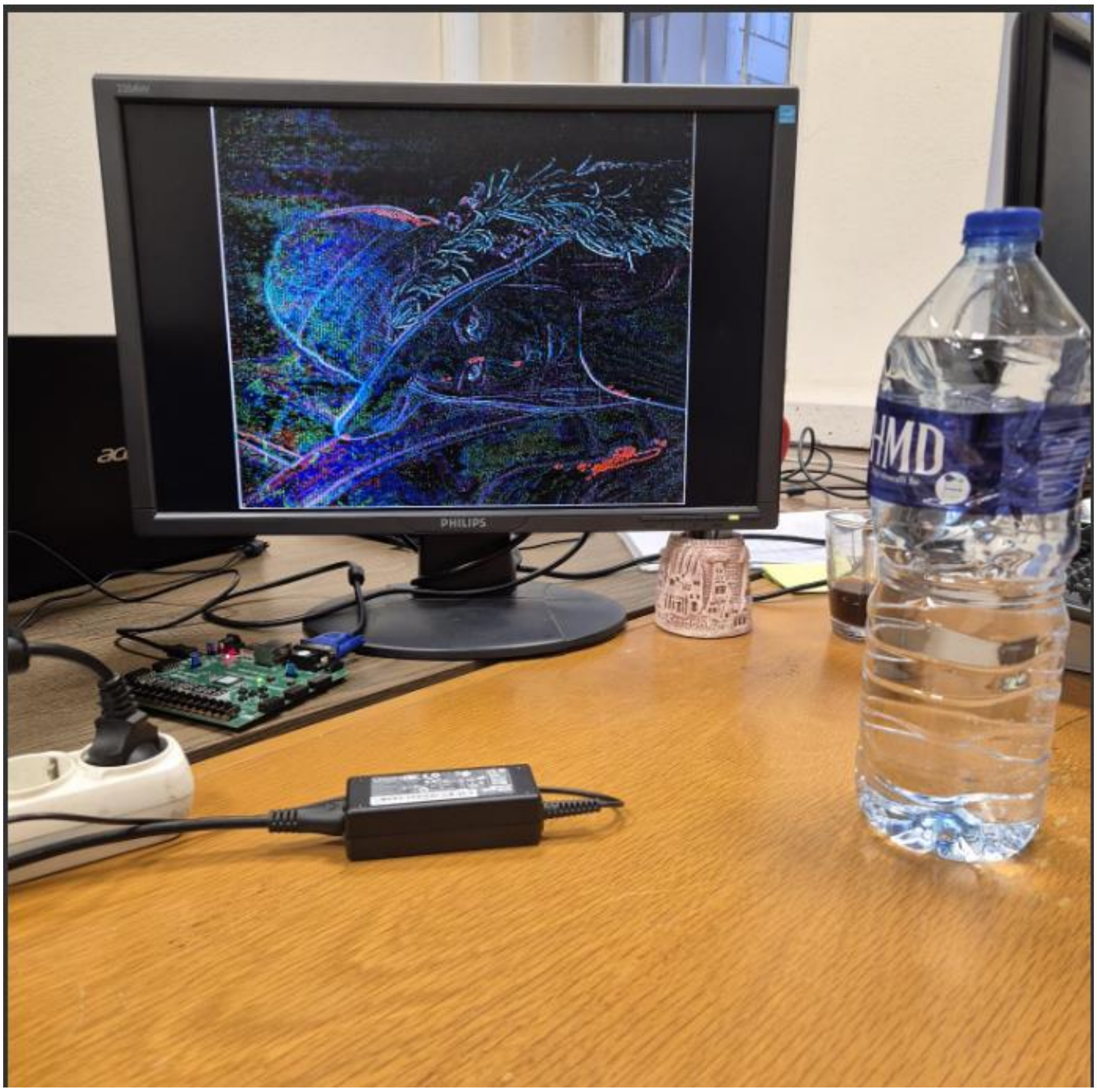
Experiment 8

```
`timescale 1ns / 1ps

module top(
input clk, //100mhz
input rst,
//output clk25,
output VGA_HS,
output VGA_VS,
output [3:0] VGA_R,
output [3:0] VGA_G,
output [3:0] VGA_B
    );

    wire sys_clk;
    //assign clk25 = sys_clk;
    wire pll_clk;
    wire en_for_controllers;
    reg enRed,enGreen,enBlue;

    wire [16:0] addr_R;
    wire [16:0] addr_G;
    wire [16:0] addr_B;

    wire [11:0] data_R;
    wire [11:0] data_G;
    wire [11:0] data_B;

    wire [11:0] RedKernel1;
    wire [11:0] RedKernel2;
    wire [11:0] RedKernel3;
    wire [11:0] Redpixel1;
    wire [11:0] Redpixel2;
    wire [11:0] Redpixel3;

    wire [11:0] GreenKernel1;
    wire [11:0] GreenKernel2;
    wire [11:0] GreenKernel3;
    wire [11:0] Greenpixel1;
    wire [11:0] Greenpixel2;
    wire [11:0] Greenpixel3;

    wire [11:0] BlueKernel1;
    wire [11:0] BlueKernel2;
    wire [11:0] BlueKernel3;
    wire [11:0] Bluepixel1;
    wire [11:0] Bluepixel2;
    wire [11:0] Bluepixel3;
    reg reset;

    controller Red (
    .pixel_clk(sys_clk),
    .rst(reset),
    .enable(en_for_controllers),
    .data_in(data_R),
    .address(addr_R),
    .kernel1(RedKernel1),
    .kernel2(RedKernel2),
    .kernel3(RedKernel3),
    .pixel1(Redpixel1),
    .pixel2(Redpixel2),
    .pixel3(Redpixel3),
    .done(done)
);
    conv convRed(
      .enable(en_for_controllers),
      .pixel1(Redpixel1),
      .pixel2(Redpixel2),
      .pixel3(Redpixel3),
      .kernel1(RedKernel1),
      .kernel2(RedKernel2),
      .kernel3(RedKernel3),
      .pixel_out(VGA_R)
    );
    controller Green (
    .pixel_clk(sys_clk),
    .rst(reset),
    .enable(en_for_controllers),
    .data_in(data_G),
    .address(addr_G),
    .kernel1(GreenKernel1),
    .kernel2(GreenKernel2),
    .kernel3(GreenKernel3),
    .pixel1(Greenpixel1),
    .pixel2(Greenpixel2),
    .pixel3(Greenpixel3),
    .done(done)
);
    conv convGreen (
    .enable(en_for_controllers),
    .pixel1(Greenpixel1),
    .pixel2(Greenpixel2),
    .pixel3(Greenpixel3),
    .kernel1(GreenKernel1),
    .kernel2(GreenKernel2),
    .kernel3(GreenKernel3),
    .pixel_out(VGA_G)
);

    controller Blue (
    .pixel_clk(sys_clk),
    .rst(reset),
    .enable(en_for_controllers),
    .data_in(data_B),
    .address(addr_B),
    .kernel1(BlueKernel1),
    .kernel2(BlueKernel2),
    .kernel3(BlueKernel3),
    .pixel1(Bluepixel1),
    .pixel2(Bluepixel2),
    .pixel3(Bluepixel3),
    .done(done)
);
    conv convBlue (
    .enable(en_for_controllers),
    .pixel1(Bluepixel1),
    .pixel2(Bluepixel2),
    .pixel3(Bluepixel3),
    .kernel1(BlueKernel1),
    .kernel2(BlueKernel2),
    .kernel3(BlueKernel3),
    .pixel_out(VGA_B)
);
    vga_driver vga(
        .pixel_clk(sys_clk),         // 25 MHz
          .rst(reset),
          .VGA_HS(VGA_HS),
          .VGA_VS(VGA_VS),
          .data_en(en_for_controllers)
    );

    clk_wiz_0 pll
   (
    .clk_out(sys_clk),
    .reset(rst),
    .locked(locked),
    .clk(clk)
   );

   blk_mem_gen_0 RedMem (
  .clka(sys_clk),    // input wire clka
  .ena(enRed),      // input wire ena
  .addra(addr_R),  // input wire [16 : 0] addra
  .douta(data_R)  // output wire [11 : 0] douta
   );
   blk_mem_gen_1 GreenMem (
  .clka(sys_clk),    // input wire clka
  .ena(enGreen),      // input wire ena
  .addra(addr_G),  // input wire [16 : 0] addra
  .douta(data_G)  // output wire [11 : 0] douta
  );

   blk_mem_gen_2 BlueMem (
  .clka(sys_clk),    // input wire clka
  .ena(enBlue),      // input wire ena
  .addra(addr_B),  // input wire [16 : 0] addra
  .douta(data_B)  // output wire [11 : 0] douta
  );




    always @(posedge clk or posedge rst) begin
    if (rst) begin
        reset <= 1;
        enRed<=0;
        enGreen<=0;
        enBlue<=0;
    end else if (!locked) begin
        reset <= 1;
        enRed<=0;
        enGreen<=0;
        enBlue<=0;
    end else begin
        reset <= 0;
        enRed<=1;
        enGreen<=1;
        enBlue<=1;
    end
    end

endmodule
```
Top Module

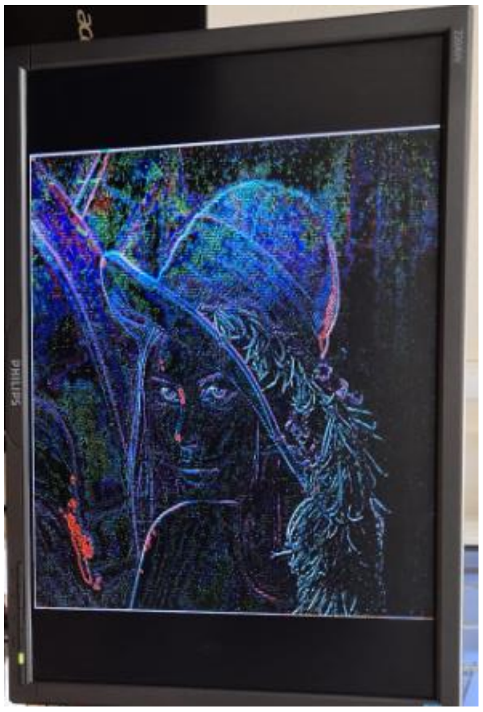
```
`timescale 1ns / 1ps

module conv(
    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 [12:0] result,
    output reg [3:0] pixel_out
);

    reg signed [4:0] px1, px2, px3, px4, px5, px6, px7, px8, px9;
    reg signed [3:0] kr1, kr2, kr3, kr4, kr5, kr6, kr7, kr8, kr9;


    reg signed [12:0] res;


    always @(*) begin
    px1 = $signed({0,pixel1[11:8]});
    px2 = $signed({0,pixel1[7:4]});
    px3 = $signed({0,pixel1[3:0]});
    px4 = $signed({0,pixel2[11:8]});
    px5 = $signed({0,pixel2[7:4]});
    px6 = $signed({0,pixel2[3:0]});
    px7 = $signed({0,pixel3[11:8]});
    px8 = $signed({0,pixel3[7:4]});
    px9 = $signed({0,pixel3[3:0]});

    kr1 = $signed(kernel1[11:8]);
    kr2 = $signed(kernel1[7:4]);
    kr3 = $signed(kernel1[3:0]);
    kr4 = $signed(kernel2[11:8]);
    kr5 = $signed(kernel2[7:4]);
    kr6 = $signed(kernel2[3:0]);
    kr7 = $signed(kernel3[11:8]);
    kr8 = $signed(kernel3[7:4]);
    kr9 = $signed(kernel3[3:0]);
end


    always @(*) begin
        if (enable) begin
            res = -(((px1 * kr1) + (px2 * kr2) + (px3 * kr3) + (px4 * kr4) +
(px5 * kr5) + (px6 * kr6) + (px7 * kr7) + (px8 * kr8) + (px9 * kr9)));
            //result = res;
            if (res > 15)
                pixel_out = 15;
            else if (res < 0)
                pixel_out = 0;
            else
                pixel_out = res[3:0];
        end else begin
            pixel_out = 0;
        end
    end
endmodule
```
Convolution Unit Verilog Code

```
`timescale 1ns / 1ps

module controller(
    input pixel_clk,rst,enable,
    input [11:0] data_in,

    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,

    output reg done
);

/*
             -------             -------
            /       \           /
           /         \         /
          /           \       /
    ------              ------
    address
    change
    on clock edge               data comes here
*/

reg signed [3:0] buffer1[641:0]/* synthesis ram_style = "block" */;
reg signed [3:0] buffer2[641:0]/* synthesis ram_style = "block" */;

reg signed [11:0] data_prev;
reg [9:0] index; // for the buffers

reg [16:0] addr; // for the rom
reg w_for_en;

localparam idle = 10'b000000000 ;       //0

localparam firstLine1 = 10'b000000001; //1
localparam firstLine2 = 10'b000000010; // 2
localparam firstLine3 = 10'b000000100;  // 4

localparam SecondLine1 = 10'b000001000;  // 8
localparam SecondLine2 = 10'b000010000; // 16
localparam SecondLine3 = 10'b000100000; // 32

localparam st1 = 10'b001000000; // 64
localparam st2 = 10'b010000000; // 128
localparam st3 = 10'b100000000;    // 256

reg [9:0] state;

    always @(posedge pixel_clk or posedge rst) begin
        if (rst) begin
            done <= 0;
            state <= idle;
            addr <= 0;
            pixel1 <= 0;
            pixel2 <= 0;
            pixel3 <= 0;
            index <= 0;
            addr <= 0;
            data_prev <= 0;
        end else begin
         case(state)
            idle: begin
                done <= 0;
                pixel1 <= 0;
                pixel2 <= 0;
                pixel3 <= 0;
                index <= 0;
                addr <= 0;
                data_prev <= 0;
                state <= firstLine1;
            end

            firstLine1: begin
                pixel1 <= 0;
                pixel2 <= 0;
                pixel3 <= 0;
                buffer1[index]  <= data_in[11:8];
                index <= index + 1;
                state <= firstLine2;
            end

            firstLine2: begin
                pixel1 <= 0;
                pixel2 <= 0;
                pixel3 <= 0;
                addr <= addr + 1;
                index <= index + 1;
                buffer1[index]  <= data_in[7:4];
                state <= firstLine3;
            end

            firstLine3: begin
                pixel1 <= 0;
                pixel2 <= 0;
                pixel3 <= 0;
                buffer1[index]  <= data_in[3:0];
                if(index == 641) begin
                    index <= 0;
                    state <=SecondLine1;
                end else begin
                    index <= index + 1;
                    state <=firstLine1;
                end
            end

            SecondLine1: begin
                pixel1 <= 0;
                pixel2 <= 0;
                pixel3 <= 0;
                buffer2[index]  <= data_in[11:8];
                index <= index + 1;
                state <= SecondLine2;
            end

            SecondLine2: begin
                pixel1 <= 0;
                pixel2 <= 0;
                pixel3 <= 0;
                addr <= addr + 1;
                index <= index + 1;
                buffer2[index]  <= data_in[7:4];
                state <= SecondLine3;
                w_for_en <= 0;
            end
            SecondLine3: begin
                pixel1 <= 0;
                pixel2 <= 0;
                pixel3 <= 0;

                if(index == 641 && enable ) begin
                    buffer2[index]  <= data_in[3:0];
                    index <= 0;
                    state<=st1;
                    w_for_en <= 0;
                end else if(index == 641 && w_for_en) begin
                    state<=SecondLine3;
                    w_for_en <= 1;
                end else if(index == 641 && !enable) begin
                    buffer2[index]  <= data_in[3:0];
                    w_for_en <= 1;
                    state<=SecondLine3;
                end else begin
                    buffer2[index]  <= data_in[3:0];
                    index <= index + 1;
                    state <=SecondLine1;
                    w_for_en <= 0;
                end
            end

            st1: begin
                if (addr == 103148) begin //done checking condition
                        done<=1;
                        state <= idle;
                        addr <= 0;
                end else if (enable) begin
                    buffer1[index]<=buffer2[index];
                    buffer2[index]<=data_in[11:8];
                    pixel1 <=
{buffer1[index],buffer1[index+1],buffer1[index+2]};
                    pixel2 <=
{buffer2[index],buffer2[index+1],buffer2[index+2]};
                    pixel3 <= data_in;

                    if(index == 639) begin
                        index <= 0;
                        state <= st1; // new line has been reached
                    end else begin
                        buffer1[index]<=buffer2[index];
                        buffer2[index]<=data_in[11:8];
                        index <= index + 1;
                        state <= st2;
                        data_prev <= data_in;
                    end
                end else
                    state <= st1;
            end

            st2: begin


                buffer1[index]<=buffer2[index];
                buffer2[index]<=data_prev[7:4];
                pixel1 <= {buffer1[index],buffer1[index+1],buffer1[index+2]};
                pixel2 <= {buffer2[index],buffer2[index+1],buffer2[index+2]};
                pixel3 <= {data_prev[11:4],data_in[3:0]};
                index <= index + 1;
                addr <= addr + 1;
                state <= st3;


            end

            st3: begin

                buffer1[index]<=buffer2[index];
                buffer2[index]<=data_prev[3:0];
                index <= index + 1;
                pixel1 <= {buffer1[index],buffer1[index+1],buffer1[index+2]};
                pixel2 <= {buffer2[index],buffer2[index+1],buffer2[index+2]};
                pixel3 <= {data_prev[11:8],data_in[7:0]};
                if(index == 638) begin // if next st1 will be the end of the
line
                    // at the end of line st3 -> st1 -> st1 -> st2...
                    addr <= addr + 1; // we know that st1 will return to
itself thus we need to increment the address here to get the data ready for
st1 after st1;
                end
                state <= st1;

            end

            default: begin
                state <= idle;
            end


         endcase
        end
    end
    always @(*) begin
        address = addr;
        kernel1 = 12'b000100010001;
        kernel2 = 12'b000110000001;
        kernel3 = 12'b000100010001;
    end

endmodule
```

Controller Verilog Code



Output of the code



Because I confused rows with columns in the matlab code the output is mirrored