DIGITAL SYSTEM DESIGN APPLICATION

EHB436E CRN: 11280

Salih Ömer Ongün
040220780

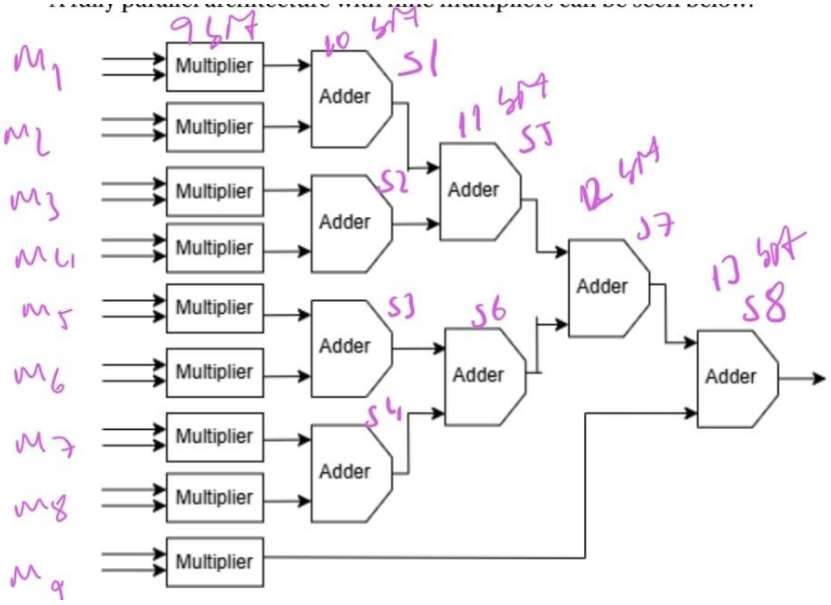
Experiment 8

Salih Ömer Ongün
040220780

CONVOLUTION MODULE

Design Source

```
module conv_unit
(
    input pixel_clk,
    input rst,
    input enable,
    input signed [4:0] pixel11,
    input signed [4:0] pixel12,
    input signed [4:0] pixel13,
    input signed [4:0] pixel21,
    input signed [4:0] pixel22,
    input signed [4:0] pixel23,
    input signed [4:0] pixel31,
    input signed [4:0] pixel32,
    input signed [4:0] pixel33,
    input signed [3:0] kernel11,
    input signed [3:0] kernel12,
    input signed [3:0] kernel13,
    input signed [3:0] kernel21,
    input signed [3:0] kernel22,
    input signed [3:0] kernel23,
    input signed [3:0] kernel31,
    input signed [3:0] kernel32,
    input signed [3:0] kernel33,
    output [3:0] pixel_out
);
    reg [3:0] pixel_out_reg;
    reg signed [8:0] m1,m2,m3,m4,m5,m6,m7,m8,m9;
    wire signed [9:0] sum1,sum2,sum3,sum4;
    wire signed [10:0] sum5,sum6;
    wire signed [11:0] sum7;
    wire signed [12:0] sum8;

    always @(posedge pixel_clk) begin
        if (rst == 1'b1) begin
            pixel_out_reg <= 4'b0;
        end
        else if (enable == 1'b1) begin
            m1 <= pixel11 * kernel11;
            m2 <= pixel12 * kernel12;
            m3 <= pixel13 * kernel13;
            m4 <= pixel21 * kernel21;
            m5 <= pixel22 * kernel22;
            m6 <= pixel23 * kernel23;
            m7 <= pixel31 * kernel31;
            m8 <= pixel32 * kernel32;
            m9 <= pixel33 * kernel33;
            if(sum8>15) begin
                pixel_out_reg <= 4'd15;
            end
            else if(sum8<0) begin
                pixel_out_reg <= 4'b0;
            end
            else begin
                pixel_out_reg <= sum8;
            end
        end
        else begin
            pixel_out_reg <= 4'b0;
        end
    end
    assign sum1 = m1 + m2;
    assign sum2 = m3 + m4;
    assign sum3 = m5 + m6;
    assign sum4 = m7 + m8;
    assign sum5 = sum1 + sum2;
    assign sum6 = sum3 + sum4;
    assign sum7 = sum5 + sum6;
    assign sum8 = -1*(sum7 + m9);
    assign pixel_out = pixel_out_reg;
endmodule
```

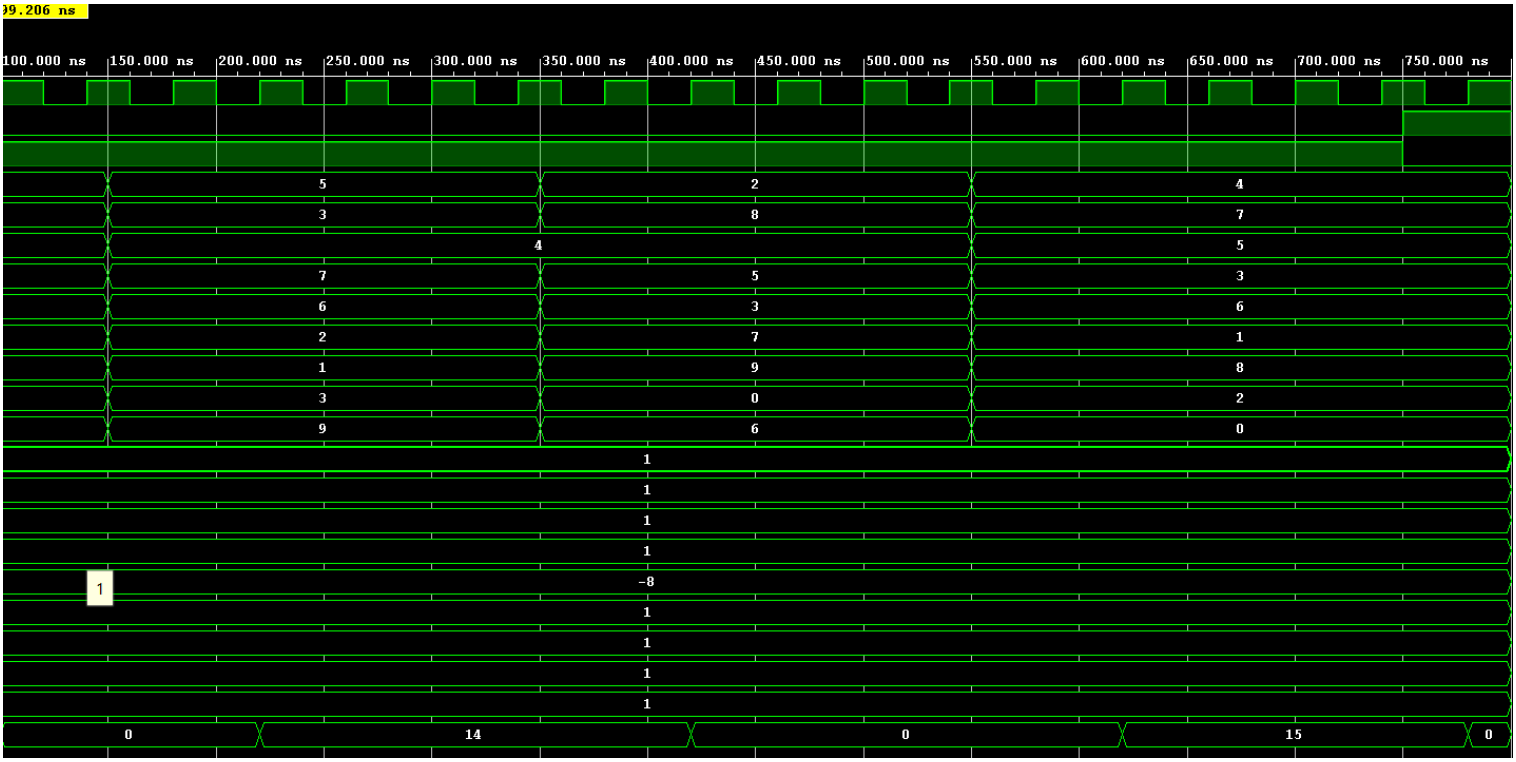


I set all multiplication calculations to be dependent on the clk signal. If the addition operation was in the always block, it would give incorrect results at certain intervals due to the clk signal. Because in the always block, which is dependent on the clk signal, all operations are performed simultaneously. That's why I did the addition operations with assign. I did the process in the code in the image above.

Simulation Source

```
module conv_unit_tb();
    reg pixel_clk = 1'b0;
    reg rst = 1'b0;
    reg enable = 1'b0;
    reg signed [4:0] pixel11, pixel12, pixel13;
    reg signed [4:0] pixel21, pixel22, pixel23;
    reg signed [4:0] pixel31, pixel32, pixel33;
    reg signed [3:0] kernel11, kernel12, kernel13;
    reg signed [3:0] kernel21, kernel22, kernel23;
    reg signed [3:0] kernel31, kernel32, kernel33;
    wire [3:0] pixel_out;

    conv_unit uut (
        .pixel_clk(pixel_clk),
        .rst(rst),
        .enable(enable),
        .pixel11(pixel11), .pixel12(pixel12), .pixel13(pixel13),
        .pixel21(pixel21), .pixel22(pixel22), .pixel23(pixel23),
        .pixel31(pixel31), .pixel32(pixel32), .pixel33(pixel33),
        .kernel11(kernel11), .kernel12(kernel12), .kernel13(kernel13),
        .kernel21(kernel21), .kernel22(kernel22), .kernel23(kernel23),
        .kernel31(kernel31), .kernel32(kernel32), .kernel33(kernel33),
        .pixel_out(pixel_out)
    );

    always begin
        #20 pixel_clk = ~pixel_clk;
    end
    initial begin
        rst = 1;
        enable = 0;
        pixel11 = 0; pixel12 = 0; pixel13 = 0;
        pixel21 = 0; pixel22 = 0; pixel23 = 0;
        pixel31 = 0; pixel32 = 0; pixel33 = 0;
        kernel11 = 1; kernel12 = 1; kernel13 = 1;
        kernel21 = 1; kernel22 = -8; kernel23 = 1;
        kernel31 = 1; kernel32 = 1; kernel33 = 1;
        #50;
        rst = 0;
        enable = 1;
        #100;
        pixel11 = 5; pixel12 = 3; pixel13 = 4;
        pixel21 = 7; pixel22 = 6; pixel23 = 2;
        pixel31 = 1; pixel32 = 3; pixel33 = 9;
        kernel11 = 1; kernel12 = 1; kernel13 = 1;
        kernel21 = 1; kernel22 = -8; kernel23 = 1;
        kernel31 = 1; kernel32 = 1; kernel33 = 1;


        // İkinci test verisi
        #200;
        pixel11 = 2; pixel12 = 8; pixel13 = 4;
        pixel21 = 5; pixel22 = 3; pixel23 = 7;
        pixel31 = 9; pixel32 = 0; pixel33 = 6;
        kernel11 = 1; kernel12 = 1; kernel13 = 1;
        kernel21 = 1; kernel22 = -8; kernel23 = 1;
        kernel31 = 1; kernel32 = 1; kernel33 = 1;


        // Üçüncü test verisi
        #200;
        pixel11 = 4; pixel12 = 7; pixel13 = 5;
        pixel21 = 3; pixel22 = 6; pixel23 = 1;
        pixel31 = 8; pixel32 = 2; pixel33 = 0;
        kernel11 = 1; kernel12 = 1; kernel13 = 1;
        kernel21 = 1; kernel22 = -8; kernel23 = 1;
        kernel31 = 1; kernel32 = 1; kernel33 = 1;
        #200;
        rst = 1;
        enable = 0;
        #50;
        $finish;
    end
endmodule
```

Simulation Waveform



The upper values show the pixels and the lower values show the kernels. Since we expressed the kernel in a negative way, we took the negative of the result while calculating the result.

The code works correctly as seen in the waveform.

BRAM MODULE

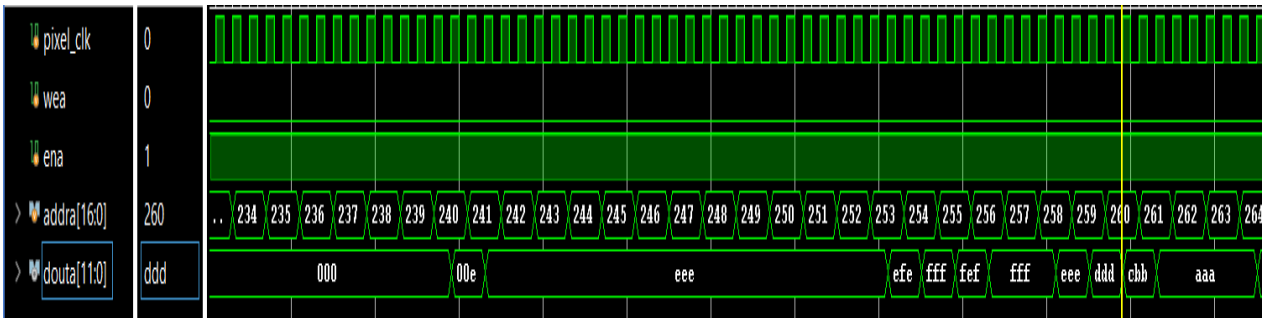
Design Source

```
module b_ram_red
(
    input pixel_clk,
    input wea,
    input ena,
    input [16:0]  addra,
    input [11:0] dina,
    output [11:0] douta
);

    blk_mem_gen_0 red_bram (  // red
      .clka(pixel_clk),    // input wire clka
      .ena(ena),      // input wire ena
      .wea(wea),      // input wire [0 : 0] wea
      .addra(addra),  // input wire [16 : 0] addra
      .dina(dina),    // input wire [11 : 0] dina
      .douta(douta)  // output wire [11 : 0] douta
    );

endmodule
```

Simulation Source

```
module b_ram_red_tb();

    reg pixel_clk = 1'b0;
    reg wea = 1'b0;
    reg ena = 1'b0;
    reg [16:0]  addra = 16'b0;
    reg [11:0] dina  = 12'b0;
    wire [11:0] douta;
    integer i;

    b_ram_red uut
    (
        .pixel_clk(pixel_clk),
        .wea(wea),
        .ena(ena),
        .addra(addra),
        .dina(dina),
        .douta(douta)
    );

    always begin
        #20 pixel_clk <= ~pixel_clk;
    end

    initial begin
        #40
        ena = 1;
        $display("we are reading");
        #300; // ilk baslangıcta LOCKED oluyor
        for(i = 0; i<103148; i=i+1) begin
            addra = i;
            $display("Address: %d, Data Read: %h", addra, douta);
            #80;
        end

        #50;
        $finish;
    end
endmodule
```

Simulation Waveform



I loaded the Bram code into a red input file and checked whether it was reading sequentially.

As can be seen in the graph, the code works correctly. Bram is serving the data correctly and in order.

CONTROLLER MODULE

Design Source

```
module controller
(                                   //
    input pixel_clk,
    input rst,
    input enable,
    input [11:0] data_in,
    output reg done,
    output reg [16:0] address,
    output  signed [11:0] kernel1,
    output  signed [11:0] kernel2,
    output  signed [11:0] kernel3,
    output reg  [11:0] pixel1,
    output reg  [11:0] pixel2,
    output reg  [11:0] pixel3
);
    assign kernel1 = 12'b0001_0001_0001;
    assign kernel2 = 12'b0001_1000_0001;
    assign kernel3 = 12'b0001_0001_0001;
    reg [3:0] buffer1[641:0];
    reg [3:0] buffer2[641:0];
    reg [7:0] counter;
    reg [9:0] sel_column;
    reg [7:0] column;
    reg [8:0] row;
    reg [11:0] prev_data;
    localparam FIRST_LINE = 3'b000;
    localparam SECOND_LINE = 3'b001;
    localparam SHIFT1 = 3'b010;
    localparam SHIFT2 = 3'b011;
    localparam SHIFT3 = 3'b100;
    localparam DONE = 3'b101;
    reg [2:0] state;
```

```
always @(posedge pixel_clk) begin

    if (rst == 1'b1) begin
        state <= FIRST_LINE;
        counter <= 10'd0;
        sel_column <= 12'd0;
        column <= 8'd0;
        row <= 9'd2;
        done <= 1'b0;
        address <= 17'd0;
    end
    else begin
        if(enable  == 1'b1) begin
            case(state)
                FIRST_LINE: begin
                    done <= 1'b0;
                    buffer1[counter * 3] <= data_in[11:8];
                    buffer1[(counter * 3) + 1] <= data_in[7:4];
                    buffer1[(counter * 3) + 2] <= data_in[3:0];
                    if(counter == 10'd213) begin
                        counter <= 10'd0;
                        state <= SECOND_LINE;
                        address <= 17'd214;
                    end
                    else begin
                        counter <= counter + 1 ;
                        address <= counter + 1;
                    end
                end

                SECOND_LINE: begin
                    buffer2[counter * 3] <= data_in[11:8];
                    buffer2[(counter * 3) + 1] <= data_in[7:4];
                    buffer2[(counter * 3) + 2] <= data_in[3:0];
                    if(counter == 10'd213) begin
                        counter <= 10'd0;
                        state <= SHIFT1;
                        address <= (214 * row);
                    end
                    else begin
                        counter <= counter + 1 ;
                        address <= counter + 1;
                    end
                end


                SHIFT1: begin
                    pixel1 <= {buffer1[sel_column],buffer1[sel_column+1],buffer1[sel_column + 2]};
                    pixel2 <= {buffer2[sel_column],buffer2[sel_column + 1],buffer2[sel_column + 2]};
                    pixel3 <= data_in;
                    if(sel_column == 10'd639) begin
                        if(row == 9'd481) begin
                            state <=DONE;
                        end
                        else begin
                            row <= row + 1;
                            state <= SHIFT1;
                            sel_column <= 10'd0;
                            column <= 8'd0;
                            address <= (214 * row);
                        end
                        sel_column <= 10'd0;
                        column <= 8'd0;

                    end
                    else begin
                        prev_data <= data_in;
                        column <= column + 1;
                        address <= (214 * row) + column + 1;
                        sel_column <= sel_column + 1;
                        state <= SHIFT2;
                    end
                end

                SHIFT2: begin
                    //address <= (214 * row) + column; // 3 0,
                    pixel1 <= {buffer1[sel_column],buffer1[sel_column + 1],buffer1[sel_column + 2]};
                    pixel2 <= {buffer2[sel_column],buffer2[sel_column + 1],buffer2[sel_column + 2]};
                    pixel3 <= {prev_data[7:0],data_in[11:8]};
                    state <= SHIFT3;
                    sel_column <= sel_column + 1;

                        end
```

```
                    SHIFT3: begin
                        pixel1 <= {buffer1[sel_column],buffer1[sel_column +
1],buffer1[sel_column + 2]};
                        pixel2 <= {buffer2[sel_column],buffer2[sel_column +
1],buffer2[sel_column + 2]};
                        pixel3 <= {prev_data[3:0],data_in[11:4]};
                        buffer2[sel_column] <= prev_data[3:0];
                        buffer2[sel_column - 1] <= prev_data[7:4];
                        buffer2[sel_column - 1] <= prev_data[7:4];
                        buffer2[sel_column - 2] <= prev_data[11:8];
                        buffer1[sel_column]     <= buffer2[sel_column];
                        buffer1[sel_column - 1] <= buffer2[sel_column - 1];
                        buffer1[sel_column - 2] <= buffer2[sel_column - 2];

                        state <= SHIFT1;
                        sel_column <= sel_column + 1;

                    end

                    DONE: begin
                        state <= FIRST_LINE;
                        counter <= 10'd0;
                        sel_column <= 12'd0;
                        column <= 8'd0;
                        row <= 9'd2;
                        done <= 1'b1;

                    end


                    default: begin
                        state <= FIRST_LINE;
                        counter <= 10'd0;
                        sel_column <= 12'd0;
                        column <= 8'd0;
                        row <= 9'd2;
                        done <= 1'b0;
                    end

                endcase
            end
        end
    end
endmodule
```

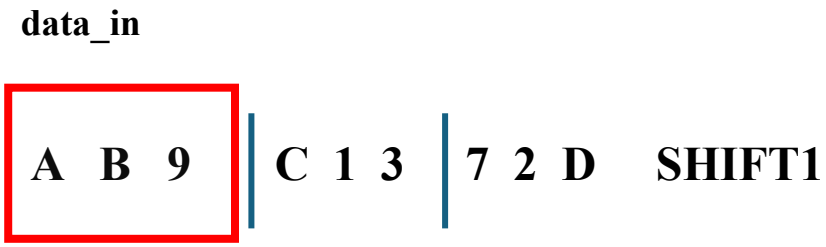
In FIRST_LINE and SECOND_LINE states, I assigned the value of the first two lines to buffer1 and buffer2. Before going to SHIFT1 state, I gave the address variable the address of line 3, column 1 in BRAM. Thus, data_in obtained the first 12 pixels of line 3.

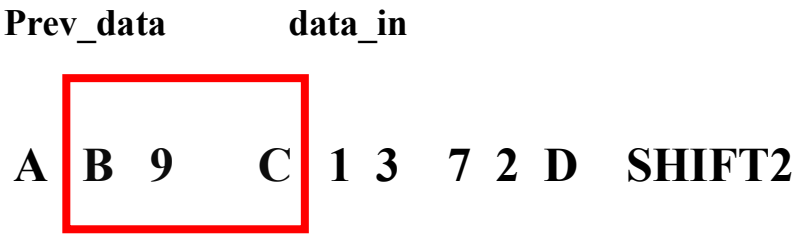
In SHIFT1, I set the buffers to pixel outputs with the data coming from BRAM. With the If block, I checked whether the index came to the last column element on the line. If it did, I increased the line by one and gave the BRAM address of the bottom line to the address and made data_in receive data from the bottom line. If the last line is where I am, I made it go to the DONE state. Before moving to SHIFT2, I shifted the BRAM address one step to the right and assigned the data to the prev_data variable.

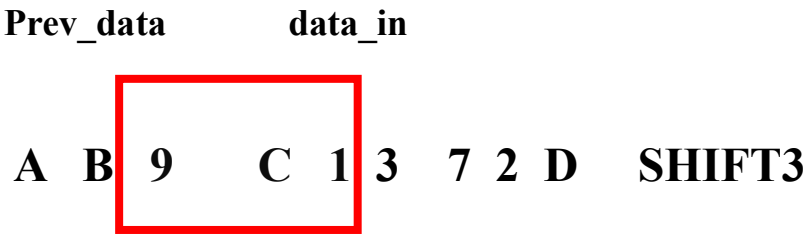
In SHIFT2, I received data from the buffers shifted one pixel to the right. For Pixel 3, I used the first 8 bits of prev_data, which is the previous data, and the last 4 bits of the new data.

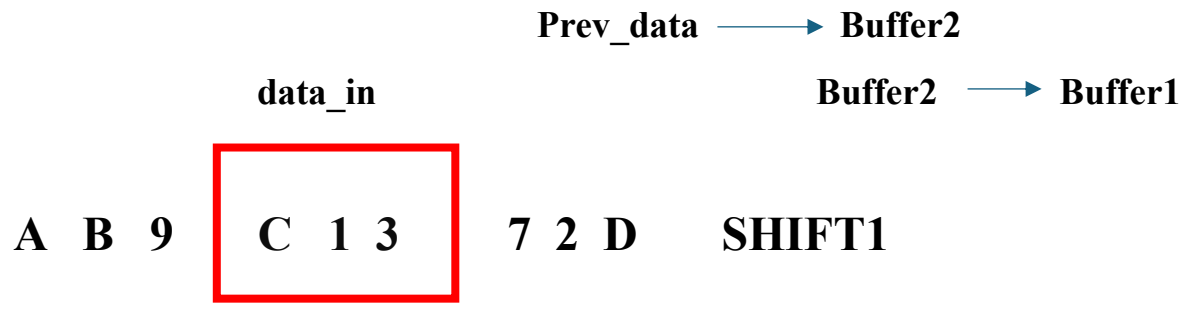
In SHIFT3, as in SHIFT2, I shifted one pixel to the right and assigned it to the outputs. Without starting a 3-pixel process again, I transferred the data to the next line and returned to SHIFT1.

When the whole picture convolves, it goes from SHIFT1 to DONE.









Simulation Source

I simulate controller module in top module.

Salih Ömer Ongün
040220780

TOP MODULE

```
module top_module
(
    input clk,
    input rst,
    output VGA_HS,
    output VGA_VS,
    output data_en,
    output [3:0] VGA_R,
    output [3:0] VGA_G,
    output [3:0] VGA_B
);
    wire locked;
    wire [16:0] address_r,address_b,address_g;
    wire [11:0] data_in_r,data_in_b,data_in_g;
    wire dina;
    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;

    wire signed [4:0] pixel11_b,pixel12_b,pixel13_b,pixel21_b,pixel22_b,pixel23_b,pixel31_b,pixel32_b,pixel33_b;

    wire signed [4:0] pixel11_g,pixel12_g,pixel13_g,pixel21_g,pixel22_g,pixel23_g,pixel31_g,pixel32_g,pixel33_g;

    wire wea;
    wire done;
    wire [11:0] kernel1,kernel2,kernel3,pixel1,pixel2,pixel3,pixel1_b,pixel2_b,pixel3_b,pixel1_g,pixel2_g,pixel3_g;
    //wire [3:0] pixel_out;
    assign wea = 1'b0;
    assign pixel11 = {1'b0,pixel1[11:8]};
    assign pixel12 = {1'b0,pixel1[7:4]};
    assign pixel13 = {1'b0,pixel1[3:0]};

    assign pixel21 = {1'b0,pixel2[11:8]};
    assign pixel22 = {1'b0,pixel2[7:4]};
    assign pixel23 = {1'b0,pixel2[3:0]};

    assign pixel31 = {1'b0,pixel3[11:8]};
    assign pixel32 = {1'b0,pixel3[7:4]};
    assign pixel33 = {1'b0,pixel3[3:0]};

    assign pixel11_b = {1'b0,pixel1_b[11:8]};
    assign pixel12_b = {1'b0,pixel1_b[7:4]};
    assign pixel13_b = {1'b0,pixel1_b[3:0]};

    assign pixel21_b = {1'b0,pixel2_b[11:8]};
    assign pixel22_b = {1'b0,pixel2_b[7:4]};
    assign pixel23_b = {1'b0,pixel2_b[3:0]};

    assign pixel31_b = {1'b0,pixel3_b[11:8]};
    assign pixel32_b = {1'b0,pixel3_b[7:4]};
    assign pixel33_b = {1'b0,pixel3_b[3:0]};

    assign pixel11_g = {1'b0,pixel1_g[11:8]};
    assign pixel12_g = {1'b0,pixel1_g[7:4]};
    assign pixel13_g = {1'b0,pixel1_g[3:0]};

    assign pixel21_g = {1'b0,pixel2_g[11:8]};
    assign pixel22_g = {1'b0,pixel2_g[7:4]};
    assign pixel23_g = {1'b0,pixel2_g[3:0]};

    assign pixel31_g = {1'b0,pixel3_g[11:8]};
    assign pixel32_g = {1'b0,pixel3_g[7:4]};
    assign pixel33_g = {1'b0,pixel3_g[3:0]};
```

```
    assign kernel11 = kernel1[11:8];
    assign kernel12 = kernel1[7:4];
    assign kernel13 = kernel1[3:0];
    assign kernel21 = kernel2[11:8];
    assign kernel22 = kernel2[7:4];
    assign kernel23 = kernel2[3:0];
    assign kernel31 = kernel3[11:8];
    assign kernel32 = kernel3[7:4];
    assign kernel33 = kernel3[3:0];
    parameter HPULSE  = 96;         // Hsync pulse
    parameter HBP     = 48;             // Back Porch
    parameter HACTIVE = 640;            // Horizontal pixels
    parameter HFP     = 16;         // Front Porch
    parameter H1      = HPULSE;
    parameter H2      = HPULSE+HBP;
    parameter H3      = HPULSE+HBP+HACTIVE;
    parameter H4      = HPULSE+HBP+HACTIVE+HFP;
    // vertical timing parameters (default 640x480x60)
    parameter VPULSE  = 2;              // Vsync pulse
    parameter VBP     = 33;             // Back Porch
    parameter VACTIVE = 480;            // Vertical pixels
    parameter VFP     = 10;             // Front Porch
    parameter V1      = VPULSE;
    parameter V2      = VPULSE+VBP;
    parameter V3      = VPULSE+VBP+VACTIVE;
    parameter V4      = VPULSE+VBP+VACTIVE+VFP;

    clk_wiz_0 pixel_clk_gen
   (
    .pixel_clk(pixel_clk),        // ok
    .reset(rst),                        // ok
    .locked(locked),            // ok
    .clk_in1(clk)       // ok
    );
    conv_unit red_conv
    (
        .pixel_clk(pixel_clk),   // ok
        .rst(rst),                      // ok
        .enable(data_en),           // ok
        .pixel11(pixel11), .pixel12(pixel12), .pixel13(pixel13),      // ok
        .pixel21(pixel21), .pixel22(pixel22), .pixel23(pixel23),    // ok
        .pixel31(pixel31), .pixel32(pixel32), .pixel33(pixel33),    // ok
        .kernel11(kernel11), .kernel12(kernel12), .kernel13(kernel13), //ok
        .kernel21(kernel21), .kernel22(kernel22), .kernel23(kernel23), //ok
        .kernel31(kernel31), .kernel32(kernel32), .kernel33(kernel33), //ok
        .pixel_out(VGA_R)                                           //ok
    );
    conv_unit blue_conv
    (
        .pixel_clk(pixel_clk),   // ok
        .rst(rst),                      // ok
        .enable(data_en),           // ok
        .pixel11(pixel11_b), .pixel12(pixel12_b), .pixel13(pixel13_b),      // ok
        .pixel21(pixel21_b), .pixel22(pixel22_b), .pixel23(pixel23_b),  // ok
        .pixel31(pixel31_b), .pixel32(pixel32_b), .pixel33(pixel33_b),  // ok
        .kernel11(kernel11), .kernel12(kernel12), .kernel13(kernel13), //ok
        .kernel21(kernel21), .kernel22(kernel22), .kernel23(kernel23), //ok
        .kernel31(kernel31), .kernel32(kernel32), .kernel33(kernel33), //ok
        .pixel_out(VGA_B)                                                   //ok
     );
    conv_unit green_conv
    (
        .pixel_clk(pixel_clk),   // ok
        .rst(rst),                      // ok
        .enable(data_en),           // ok
        .pixel11(pixel11_g), .pixel12(pixel12_g), .pixel13(pixel13_g),      // ok
        .pixel21(pixel21_g), .pixel22(pixel22_g), .pixel23(pixel23_g),  // ok
        .pixel31(pixel31_g), .pixel32(pixel32_g), .pixel33(pixel33_g),  // ok
        .kernel11(kernel11), .kernel12(kernel12), .kernel13(kernel13), //ok
        .kernel21(kernel21), .kernel22(kernel22), .kernel23(kernel23), //ok
        .kernel31(kernel31), .kernel32(kernel32), .kernel33(kernel33), //ok
        .pixel_out(VGA_G)                                           //ok
     );
```

```
b_ram_red red_bram
(
    .pixel_clk(pixel_clk), // ok
    .wea(wea),          // ok
    .ena(data_en),       // ok
    .addra(address_r),  // ok
    .dina(dina),    // ok
    .douta(data_in_r)   //ok
);
b_ram_blue blue_bram
(
    .pixel_clk(pixel_clk), // ok
    .wea(wea),          // ok
    .ena(data_en),       // ok
    .addra(address_b),  // ok
    .dina(dina),    // ok
    .douta(data_in_b)   //ok
);
b_ram_green green_bram
(
    .pixel_clk(pixel_clk), // ok
    .wea(wea),          // ok
    .ena(data_en),       // ok
    .addra(address_g),  // ok
    .dina(dina),    // ok
    .douta(data_in_g)   //ok
);
controller red_control
(
    .pixel_clk(pixel_clk), // ok
    .rst(rst),              // ok
    .enable(data_en),       // ok
    .data_in(data_in_r),        // ok
    .done(done),            // ok
    .address(address_r),    // ok
    .kernel1(kernel1),          // ok
    .kernel2(kernel2),       // ok
    .kernel3(kernel3),       // ok
    .pixel1(pixel1),       // ok
    .pixel2(pixel2),       // ok
    .pixel3(pixel3)       // ok
);
controller blue_control
(
    .pixel_clk(pixel_clk), // ok
    .rst(rst),              // ok
    .enable(data_en),       // ok
    .data_in(data_in_b),        // ok
    .done(done),            // ok
    .address(address_b),    // ok
    .kernel1(kernel1),          // ok
    .kernel2(kernel2),       // ok
    .kernel3(kernel3),       // ok
    .pixel1(pixel1_b),       // ok
    .pixel2(pixel2_b),       // ok
    .pixel3(pixel3_b)       // ok
);
controller green_control
(
    .pixel_clk(pixel_clk), // ok
    .rst(rst),              // ok
    .enable(data_en),       // ok
    .data_in(data_in_g),        // ok
    .done(done),            // ok
    .address(address_g),    // ok
    .kernel1(kernel1),          // ok
    .kernel2(kernel2),       // ok
    .kernel3(kernel3),       // ok
    .pixel1(pixel1_g),       // ok
    .pixel2(pixel2_g),       // ok
     .pixel3(pixel3_g)       // ok
);
```

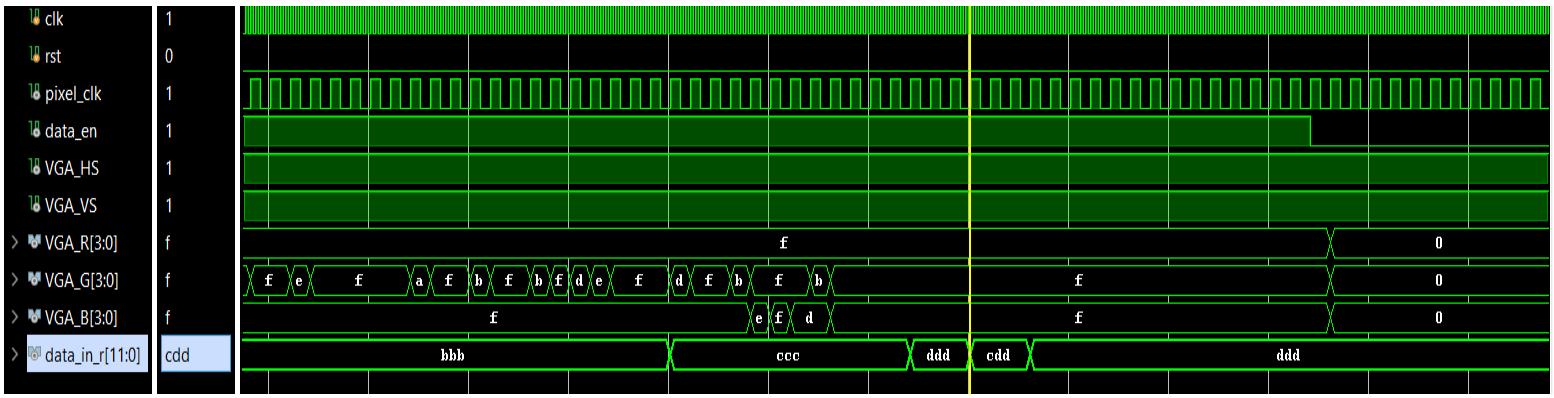
```
wire [9:0] sel_column = red_control.sel_column;
wire [9:0] counter = red_control.counter;
wire [8:0] row = red_control.row;
wire [7:0] column = red_control.column;
wire [2:0] state = red_control.state;
wire [11:0] prev_data = red_control.prev_data;

vga_driver #(
.HPULSE(HPULSE),
.HBP(HBP),
.HACTIVE(HACTIVE),
.HFP(HFP),
.VPULSE(VPULSE),
.VBP(VBP),
.VACTIVE(VACTIVE),
.VFP(VFP)
)
vga_gen (
.pixel_clk(pixel_clk),
.rst(rst),
.VGA_HS(VGA_HS),
.VGA_VS(VGA_VS),
.data_en(data_en)
);
endmodule
```

I created 3 bram, 3 controller and 3 conv_unit, 1 VGA modules. I adjusted required connection. I created sel_column, counter, row, column, state, prev_data to pursue change of controller.

Simulation Source

```
module top_module_tb;

  reg clk;
  reg rst;
  wire data_en;
  wire VGA_HS;
  wire VGA_VS;
  wire [3:0] VGA_R;
  wire [3:0] VGA_G;
  wire [3:0] VGA_B;
  integer file_r, file_g, file_b;
  integer row_count = 0;
  integer col_count = 0;

  initial clk = 0;
  always #5 clk = ~clk;

  top_module uut (
    .clk(clk),
    .rst(rst),
    .VGA_HS(VGA_HS),
    .VGA_VS(VGA_VS),
    .data_en(data_en),
    .VGA_R(VGA_R),
    .VGA_G(VGA_G),
    .VGA_B(VGA_B)
  );
  initial begin
    file_r = $fopen("vga_r_output.txt", "w");
    file_g = $fopen("vga_g_output.txt", "w");
    file_b = $fopen("vga_b_output.txt", "w");

    if (file_r == 0 || file_g == 0 || file_b == 0) begin
      $display("Error: Unable to open files for writing.");
      $stop;
    end

    rst = 1;
    #100;
    rst = 0;
    #5000000;

    $fclose(file_r);
    $fclose(file_g);
    $fclose(file_b);
    $stop;
  end

  always @(posedge clk) begin
    if (!rst && data_en) begin
      $fwrite(file_r, "%X", VGA_R);
      $fwrite(file_g, "%X", VGA_G);
      $fwrite(file_b, "%X", VGA_B);

      col_count = col_count + 1;

      if (col_count == 642) begin
        $fwrite(file_r, "\n");
        $fwrite(file_g, "\n");
        $fwrite(file_b, "\n");

        col_count = 0;
        row_count = row_count + 1;

        if (row_count == 482) begin
          $fclose(file_r);
          $fclose(file_g);
          $fclose(file_b);
          $stop;
        end
       end
     end
   end

endmodule
```

Simulation Waveform



I could not reach the same value with the output R,G,B pixels. I have checked my controller and other modules but I can't find where the problem is.

Utilization Report

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	1134	63400	1.79
FF	768	126800	0.61
BRAM	106.50	135	78.89
DSP	6	240	2.50
IO	16	210	7.62
BUFG	2	32	6.25
MMCM	1	6	16.67