COL215: Digital Logic and System Design

Hardware Assignment - 2

Image Gradient Calculator and VGA Displayer

Yash Bansal : (2022CS51133)

Soumyaprabha Dey : (2022CS11107)

PROBLEM DESCRIPTION

- To generate RAM and ROM using Vivado's memory generator and populate the ROM with the image file.
- To perform image gradient operation on the stored image and populate the RAM with the generated image file.
- To display the images stored in the FPGA memory before and after carrying out the gradient operation.

VHDL CODE STRUCTURE

1. **VGA_DRIVER1 entity declaration:-** Initialising clk, rst, hsync, vsync and the colours r, b and g.
2. **Architecture formation of VGA_DRIVER1:-** Behavioral architecture of VGA_DRIVER1 is designed using various components, signals, port mappings and processes as follows:-

- **Component dist_mem_gen_0:-** Component for Vivado's ROM memory.
- **Component dist_mem_gen_1:-** Component for Vivado's RAM.
- Declaring and initialising all the signals and constants for upcoming processes.
- **clk_div process:-** divides the onboard 100 MHz clock **clk** (10 ns) and generates a 25 MHz clock **clk25** (40 ns). **clk25** is the required pixel clock.

```
clk_div : PROCESS (clk)
    VARIABLE cycle2 : INTEGER := 0;
BEGIN
    IF (rising_edge(clk)) THEN
        IF (cycle2 = 0) THEN
            clk25 <= '1';
            cycle2 := cycle2 + 1;
        ELSIF (cycle2 = 1) THEN
            cycle2 := cycle2 + 1;
        ELSIF (cycle2 = 2) THEN
            clk25 <= '0';
            cycle2 := cycle2 + 1;
        ELSE
            cycle2 := 0;
        END IF;
```

```
    END IF;
END PROCESS;
```

- **Port mapping** for dist_mem_gen_0 (ROM) and dist_mem_gen_1 (RAM).
- **mux process:-** changes the ram_address after every 40 ns while writing with wr (write_enable) as one and then changes wr to 0 while displaying to the VGA display from RAM

```
mux : PROCESS (clk25)
BEGIN
  IF (rising_edge(clk25)) THEN
    IF (count < 65536) THEN
        IF (i >= 0) THEN
            ram_address <= STD_LOGIC_VECTOR(TO_UNSIGNED(i, 16));
        END IF;
        wr <= '1';
    ELSE
        ram_address <= STD_LOGIC_VECTOR(TO_UNSIGNED(display_counter, 16));
        wr <= '0';
    END IF;
  END IF;
END PROCESS;
```

- **image_grad_oper process:-** Takes data from ROM, performs the gradient operation and stores the data in RAM.

```
image_grad_oper : PROCESS (clk25)
    VARIABLE sum : INTEGER := 0;
    VARIABLE reg1, reg2, reg3 : INTEGER := 0;
    --reg1 prev val reg2 curr val and reg3 next val
BEGIN
    IF (rising_edge(clk25)) THEN
        IF (i =- 1) THEN
            reg1 := 0;
            reg2 := 0;
            rom_address <= STD_LOGIC_VECTOR(TO_UNSIGNED(0, 16));
            reg3 := TO_INTEGER(unsigned(data_out_rom));
            i <= i + 1;
        ELSIF (i < 65536) THEN
            IF (i MOD 256 > 0 AND i MOD 256 < 255) THEN
                reg1 := reg2;


                reg2 := reg3;
                rom_address <= STD_LOGIC_VECTOR(TO_UNSIGNED(i + 1, 16));
                reg3 := TO_INTEGER(unsigned(data_out_rom));
```

```
                sum := - 2 * reg2 + reg1 + reg3;
                IF (sum < 0) THEN
                    sum := 0;
                ELSIF (sum > 255) THEN
                    sum := 255;
                END IF;
                data_in <= STD_LOGIC_VECTOR(TO_UNSIGNED(sum, 8));
            ELSIF (i MOD 256 = 0) THEN
                reg1 := 0;
                reg2 := reg3;
                rom_address <= STD_LOGIC_VECTOR(TO_UNSIGNED(i + 1, 16));
                reg3 := TO_INTEGER(unsigned(data_out_rom));
                sum := - 2 * reg2 + reg1 + reg3;
                IF (sum < 0) THEN
                    sum := 0;
                ELSIF (sum > 255) THEN
                    sum := 255;
                END IF;
                data_in <= STD_LOGIC_VECTOR(TO_UNSIGNED(sum, 8));
            ELSIF (i MOD 256 = 255) THEN
                reg1 := reg2;
                reg2 := reg3;
                rom_address <= STD_LOGIC_VECTOR(TO_UNSIGNED(i + 1, 16));
                reg3 := TO_INTEGER(unsigned(data_out_rom));
                sum := - 2 * reg2 + reg1;
                IF (sum < 0) THEN
                    sum := 0;
                ELSIF (sum > 255) THEN
                    sum := 255;
                END IF;
                data_in <= STD_LOGIC_VECTOR(TO_UNSIGNED(sum, 8));
            END IF;
            i <= i + 1;
            count <= count + 1;
        END IF;
    END IF;
END PROCESS;
```

Logic for Image Gradient Operation:-

1. Variables **reg1**, **reg2**, and **reg3** store the current, previous, and following pixel values, while the variable **sum** holds the value to be written to RAM after applying the gradient operation.
2. Integer **i** keeps track of the operation count, initially set to -1. Initially, reg1 and reg2 are initialised to 0, while reg3 takes the value of the first pixel.
3. Each operation cycle, including reading ROM input, generating pixel output, and writing to RAM, takes 40 ns based on a 40 ns clock (clk25).
4. At the start of each frame line, reg1 and reg2 are initialised to 0, while reg3 takes the first pixel value of each line.

5. During each 40 ns clock cycle, reg1 takes reg2's value, reg2 takes reg3's value, and reg3 updates with the next pixel from ROM. The required operation is applied, and the result is stored in sum, which is then written to RAM. i increments by 1
6. If i is a multiple of 256, indicating the end of a line, reg1 resets to 0, reg2 takes reg3's value, and reg3 assumes the next pixel. These steps continue.
7. Operations continue until i reach 65536, at which point the complete gradient is stored in RAM for display on the VGA screen.

- **horizontal_position_counter process, vertical_position_counter process, horizontal_synchronisation process and vertical_synchronization process:-** The horizontal counter keeps track of pixels in a line, and the vertical counter keeps track of lines in a frame. The horizontal counter goes from 0 to 799, and the vertical counter goes from 0 to 639. When the horizontal counter becomes 799, it returns to 0 and the vertical counter increments by 1. When the vertical counter goes to 639, it returns to 0. . hsync is always kept high except when hpos >= 656 and hpos <= 761. vsync is always kept high except when vpos = 490 or vpos = 491.

```
horizontal_position_counter : PROCESS (clk25, rst)
BEGIN
    IF (rst = '1') THEN
        hpos <= 0;
    ELSIF (clk25'event AND clk25 = '1') THEN
        IF (hpos = hd + hfp + hsp + hbp) THEN
            hpos <= 0;
        ELSE
            hpos <= hpos + 1;
        END IF;
    END IF;
END PROCESS;

vertical_position_counter : PROCESS (clk25, rst, hpos)
BEGIN
    IF (rst = '1') THEN
        vpos <= 0;
    ELSIF (clk25'event AND clk25 = '1') THEN
        IF (hpos = hd + hfp + hsp + hbp) THEN
            IF (vpos = vd + vfp + vsp + vbp) THEN
                vpos <= 0;
            ELSE
                vpos <= vpos + 1;
            END IF;
        END IF;
    END IF;
END PROCESS;
```

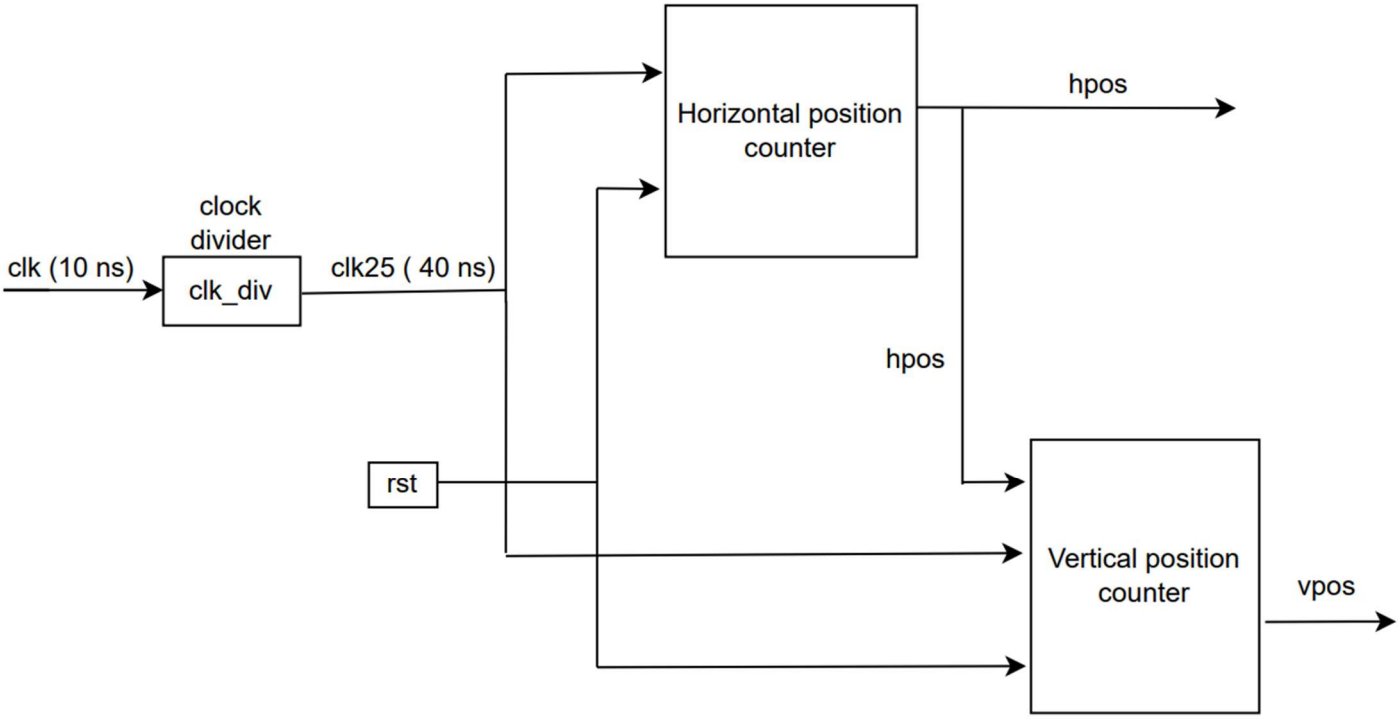
- **video_on process:-** controls the enabling and disabling the video output on the screen. video_on is high when hpos <= 639 and vpos <= 479.

- **draw process:-** When video_on is enabled, it updates the r,b, and g values of the pixel by reading the four most significant bits of the data from the RAM. The output image is displayed in the [10, 265] × [10, 265] portion of the screen.

```
draw : PROCESS (clk25, RST, hPos, vPos, videoOn)
BEGIN
   IF (count = 65536) THEN
      IF (rst = '1') THEN
         R <= "0000";
         G <= "0000";
         B <= "0000";
      ELSIF (clk25'event AND clk25 = '1') THEN
         IF (videoon = '1') THEN
            IF ((hpos >= 10 AND hpos <= 265) AND (vpos >= 10 AND vpos <=
                                                  265)) THEN
                r(3) <= data_out_ram(7);
                g(3) <= data_out_ram(7);
                b(3) <= data_out_ram(7);
                r(2) <= data_out_ram(6);
                g(2) <= data_out_ram(6);
                b(2) <= data_out_ram(6);
                r(1) <= data_out_ram(5);
                g(1) <= data_out_ram(5);
                b(1) <= data_out_ram(5);
                r(0) <= data_out_ram(4);
                g(0) <= data_out_ram(4);
                b(0) <= data_out_ram(4);
                IF (hpos = 265 AND vpos = 265) THEN
                    display_counter <= 0;
                ELSE
                    display_counter <= display_counter + 1;
                END IF;
            ELSE
                R <= "0000";
                G <= "0000";
                B <= "0000";
            END IF;
         ELSE
            R <= "0000";
            G <= "0000";
            B <= "0000";
         END IF;
      END IF;
   END IF;
END PROCESS;
```

BASYS3 File mappings

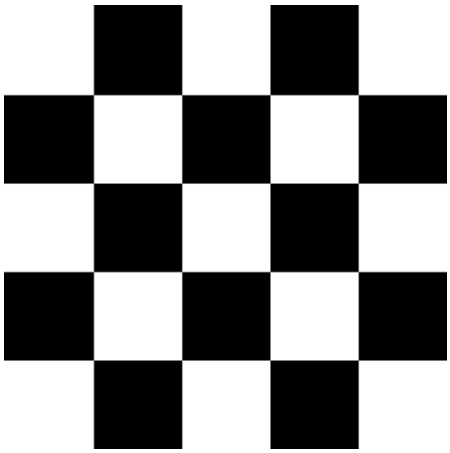
```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
        set_property IOSTANDARD LVCMOS33 [get_ports clk]
        create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
                                                        [get_ports clk]

## Switches
set_property PACKAGE_PIN V17 [get_ports {rst}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rst}]


##VGA Connector
set_property PACKAGE_PIN G19 [get_ports {r[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {r[3]}]
set_property PACKAGE_PIN H19 [get_ports {r[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {r[2]}]
set_property PACKAGE_PIN J19 [get_ports {r[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {r[1]}]
set_property PACKAGE_PIN N19 [get_ports {r[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {r[0]}]
set_property PACKAGE_PIN N18 [get_ports {g[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {g[3]}]
set_property PACKAGE_PIN L18 [get_ports {g[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {g[2]}]
set_property PACKAGE_PIN K18 [get_ports {g[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {g[1]}]
set_property PACKAGE_PIN J18 [get_ports {g[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {g[0]}]
set_property PACKAGE_PIN J17 [get_ports {b[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {b[3]}]
set_property PACKAGE_PIN H17 [get_ports {b[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {b[2]}]
set_property PACKAGE_PIN G17 [get_ports {b[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {b[1]}]
set_property PACKAGE_PIN D17 [get_ports {b[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {b[0]}]
set_property PACKAGE_PIN P19 [get_ports hsync]
        set_property IOSTANDARD LVCMOS33 [get_ports hsync]
set_property PACKAGE_PIN R19 [get_ports vsync]
        set_property IOSTANDARD LVCMOS33 [get_ports vsync]
```



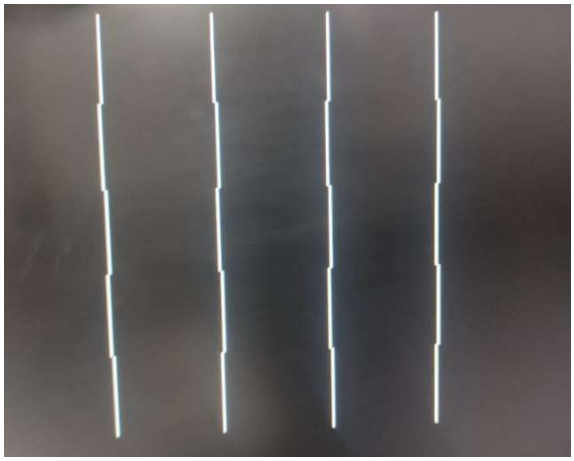
Block diagram

TEST CASES

Testcase 1:-



Input image



Output image

Testcase 2:-

Input image

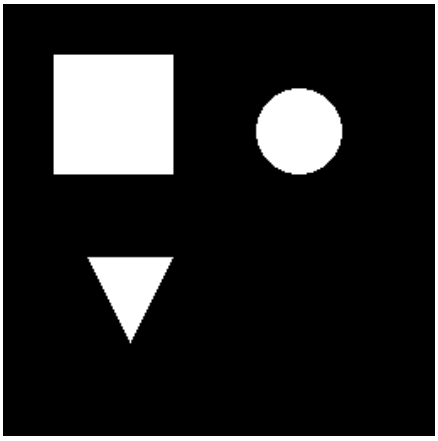
Output image

Testcase 3:-

Input image

Output image

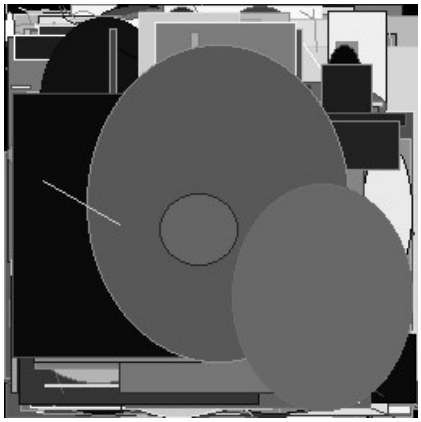
Testcase 4:-



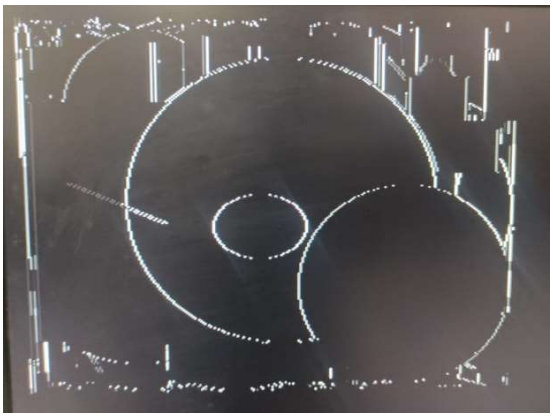
Input image

Output image

Testcase 5:-



Input image



Output image

- All the test cases, i.e. input image, expected output image, and the output image we got, along with the Python codes to generate the coe files for input images and to generate the expected output image, have been enclosed in the submission. Also, simulation snapshots of the first test case are also enclosed.