Hardware Assignment 2 – REPORT

Team Members : Nikhil Zawar – 2022CS11106
Amaan Ali Sayed – 2022CS11618

OBJECTIVES :

1. To generate RAM/ROM using vivado's memory generator and then to populate the ROM with the given image file, then to read the stored image in the ROM.
2. Using this memory implement the gradient operation. Store the result of the gradient operation in the RAM.
3. To display the image through a VGA cable on an external display, both before and after the transformation.

IMPLEMENTATION :

PART 1 :

a. Using vivado's IP catalog, we first generated a ROM memory block with the size of 65536 and data width of each unit to be 8. This memory is a distributed memory block.
b. While generating the ROM in the RST & Initialization tab we load the .coe file which has the information about the input image.
c. There are two predefined ports while generating the RAM:
    a. "a" : std_logic_vector type with a size of 16 bits : which stores the address of a particular location in the RAM. This is an input port.
    b. "spo" : std_logic_vector type with a size of 8 bits : which stores the data value stored at the address "a". This is an output port.
d. Reading the stored image in the ROM
    a. We define a clock with a fixed time of event to read from the ROM and stimulate our results.
    b. Since we are not having any input or output ports on the Basys board, we will be initializing signals to work with and will use them while simulations and in other processes.
    c. We then define a uut which consists of the port map of the ports of the component to the signals that we have defined.

PART 2

a. For the gradient operation we defined multiple signals and variables to aid in our implementation.
Initialization of signals and constants :

```
signal clka : std_logic := '0';
signal address_rom : std_logic_vector(15 DOWNTO 0) := (others => '0');
signal data_rom : std_logic_vector(7 DOWNTO 0) := (others => '0');
signal address_ram : std_logic_vector(15 DOWNTO 0) := (others => '0');
signal address_ram_vga : std_logic_vector(15 downto 0) := (others => '0');
signal data_ram_in : std_logic_vector(7 DOWNTO 0) := (others => '0');
signal data_ram_out : std_logic_vector(7 DOWNTO 0) := (others => '0');
signal wr : std_logic := '1';
constant clock_period : time := 10 ps;
signal i : integer := 0;
signal counter : integer:=0;
```

b. Image Gradient Operation :
We are performing the image gradient calculation in one clock cycle. We have defined three specific variables – prev, curr, next. Also, we store the output in val and the pixel value in output_pix.
Now what we are doing is – in a single delta delay we will first put the value of curr in prev; put the value of next in curr; now in the next clock cycle access the new value of next. Calculate the gradient operation and then store it in val. We are synchronizing this part of our implementation using a counter. We have used conditional statements to take care of the boundary situations where we reach the first or last bit, or when the val becomes negative.
All of this we are doing in a 25 MHz clock, which is named as clk25. The definition of our clock process is written further in the VGA driver section.
The following is our implementation for the gradient operation :

```
grad_proc : process(clk25)
    variable prev : integer:=0;
    variable curr : integer:=0;
    variable nex : integer:=0;
    variable val : integer;
    variable out_pix : std_logic_vector(7 downto 0);
    begin
        if (rising_edge(clk25) and i<65536 and wr = '1') then
            if (counter=0) then
                if (i mod 256 = 0) then
                 address_rom <= std_logic_vector(to_unsigned(i, 16));
                end if;
                if (i mod 256 /=0) then
                    prev:=curr;
                else
                    prev:=0;
                end if;
                counter <= 1;
            elsif (counter=1) then
                if (i mod 256 = 0) then
                    curr := to_integer(unsigned(data_rom));
                else
                    curr:=nex;
                end if;
                if ((i mod 256)/=255) then
                 address_rom<=std_logic_vector(to_unsigned(i+1, 16));
                end if;
                counter<=2;
            elsif (counter=2) then
                if ((i mod 256)/=255) then
                    nex:=to_integer(unsigned(data_rom));
                else
                    nex:=0;
                end if;
                val := prev + nex - 2*curr;
                if (val>255) then
                    val:=255;
                elsif (val < 0) then
                    val:=0;
                end if;
                out_pix:= std_logic_vector(to_unsigned(val, 8));
                data_ram_in<=out_pix;
                counter<=0;
                i<=i+1;
            end if;
        end if;
```

```
    end process;
```

After we store the value of the pixel in RAM, we need to update its address. For this we have defined another process, which is common for PART 2 and PART 3(VGA), which will modify the address of the RAM. The following is the process. In this process we use another std_logic type signal called as “wr”, kind of a enable swtich. When wr = ‘1’, I can write in the RAM, and when wr = ‘0’, then the writing in the RAM would have finished and we will extract the information from the RAM so that we can display using the VGA cable.

```
ramadd : process(wr, clk25)
begin
    if(wr='0')then
      address_ram <= std_logic_vector(to_unsigned(256*vpos+hpos, 16));
    elsif(wr='1' and counter = 2) then
      address_ram <= std_logic_vector(to_unsigned(i, 16));
    end if;
end process;
```

PART 3 :

a. ***Clock Divider :***

The inbuilt clock that we have is opf 100 MHz, and we need a clock of 25 MHZ, so we define process which halves the frequency of the clock signal. First we convert the main clk into a clock with frequency 50 MHz(clk50) and then convert clk50 into a clock signal with frequency 25 MHz(clk25). To aid with this we have used two dummy variables – c, d_var. This is done as there are a total of 420000 pixels in one single frame and they have to be displayed for a fixed time which is 60 Hz. Hence we calculate the frequency of the clock for one pixel as 25.2MHz.

```
    clk_div : process(clka)
    begin
        if(clka'event and clka = '1') then
            c <= not c;
            if (c='0') then
                clk50<='0';
            else
```

```
                clk50<='1';
            end if;
        end if;
    end process;
```

```
clk_div1 : process(clk50)
    begin
        if(clk50'event and clk50 = '1') then
            d_var <= not d_var;
            if (d_var='0') then
                clk25<='0';
            else
                clk25<='1';
            end if;
        end if;
  end process;
```

b. <u>Processes to handle horizontal and vertical positions :</u>

The horizontal coordinate traverses from 0 to 799. And when it reaches 799, then again it is set to 0, which is the next line. Similar goes for the vertical coordinate which traverses from 0 to 524. The vertical coordinate updation is conditional on the horizontal coordinate as well.

```
horizontal_position_counter : process (clk25, rst)
begin
    if(rst = '1') then
        hpos <= 0;
    elsif(rising_edge(clk25) and wr = '0') 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' and wr = '0') 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;
```

c. ***Hsync and Vsync process :***

The following processes are set to define the values of hsync and vsync ports at different times. This is to ensure the continuity in the display and we return to the next line when terminated on a specific line :

```
horizontal_sync : process (clk25, rst, hpos)
begin
    if(rst = '1') then
        hsync <= '0';
    elsif(clk25'event and clk25 = '1' and wr = '0')
then
--        if((hpos <= 655) or (hpos >= 751)) then
        if((hpos <= (hd+hfp)) or (hpos >=
(hd+hfp+hsp))) then
            hsync <= '1';

        else
            hsync <= '0' ;
        end if;
```

```
    end if;
end process;


vertical_sync : process (clk25, rst, vpos)
begin
    if(rst = '1') then
        vsync <= '0';
    elsif(clk25'event and clk25 = '1' and wr = '0')
then
--        if((vpos <= 489) or (vpos > 491)) then
        if((vpos <= (vd + vfp)) or (vpos >
(vd+vfp+vsp))) then
            vsync <= '1';
        else
            vsync <= '0';
        end if;
    end if;
end process;
```

d. <u>Video on process and Read enable process</u>

In the video on process – we define a std_logic type signal video_on which is set to 1 when the video is to be displayed and 0 otherwise. This is 1 only in the display region, when the horizontal coordinate or vertical coordinate are in front porch area, back porch area and sync porch area then video on is set to 0.

Read Enable process – when we are doing the gradient process this is set to 1 and otherwise it is set to 1 so that we know when to start the vga process. Following is its implementation :

```
video_proc : process (clk25, rst, hpos, vpos)
begin
    if(rst = '1') then
        video_on <= '0';
    elsif(clk25'event and clk25 = '1' and wr = '0') then
        if(hpos <= hd and vpos <= vd) then
            video_on <= '1';
```

```
        else
            video_on <= '0';
        end if;
    end if;
end process;


 i_proc : process (i)
    begin
        if (i>65536) then
            wr <= '0';
        end if;
    end process;
```

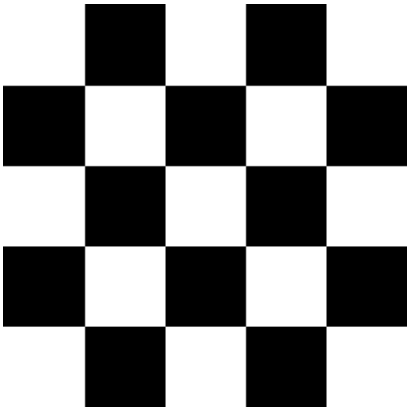
e. <u>Draw process – to display using the VGA cable</u>

This is the final process and the most important process. Here we take the input from the RAM and then display accordingly through the VGA cable. The data from the RAM is of 8 bit size, we reduce this to a 4 bit size data. This is done because the depth of our R, B, G ports is set to 4 bits. We consider the 4 MSBs and then pass them into the R, B, G ports. The implementation is as follows :

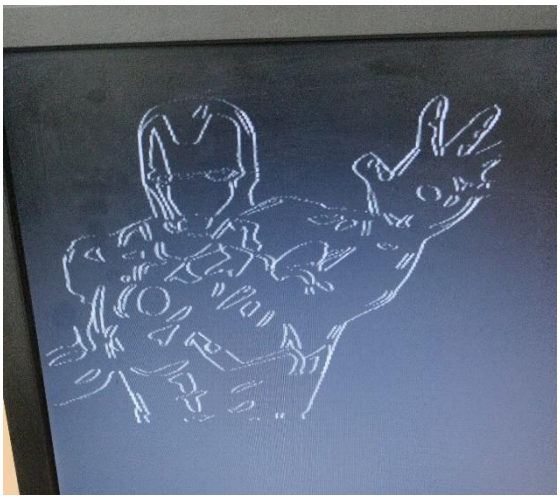
```
draw : process(clk25, rst, hpos, vpos, video_on)
begin
    if(rst = '1') then
        R <= "0000";
        G <= "0000";
        B <= "0000";
    elsif(rising_edge(clk25) and wr = '0') then
        if(video_on = '1') then
            if((hpos >= 0 and hpos <= 255) and (vpos >=
0 and vpos <= 255)) then
                R(3) <= data_ram_out(7);
                R(2) <= data_ram_out(6);
```

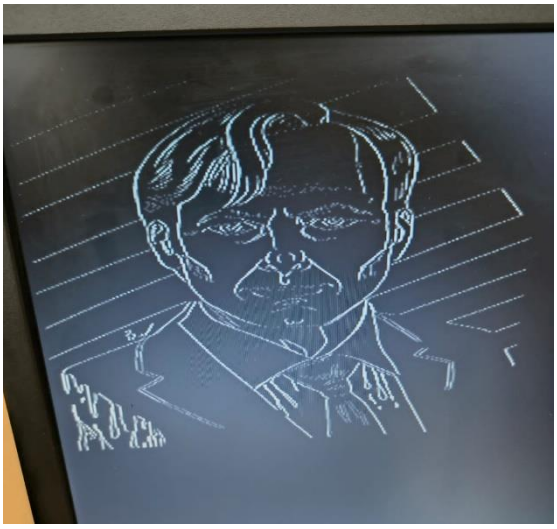
```
                R(1) <= data_ram_out(5);
                R(0) <= data_ram_out(4);
                G(3) <= data_ram_out(7);
                G(2) <= data_ram_out(6);
                G(1) <= data_ram_out(5);
                G(0) <= data_ram_out(4);
                B(3) <= data_ram_out(7);
                B(2) <= data_ram_out(6);
                B(1) <= data_ram_out(5);
                B(0) <= data_ram_out(4);
            else
                R <= "0000";
                G <= "0000";
                B <= "0000";
            end if;
        else
            R <= "0000";
            G <= "0000";
            B <= "0000";
        end if;
    end if;
end process;
```

OUTPUT & BONUS PART



After completing with the various processes and successfully being able to calculate the gradient value and displaying the output, we tried doing the same for more images. The following are our results :





Note :

Respected Sir/Mam,

Since at the end of the lab, because of time constraint we were not able to extract all the files from the lab, so please excuse us. We will be ready to present all the other files in the lab.

Thanking you,

Yours sincerely,

Nikhil & Amaan.