COL215P - Hardware Assignment 2 Report

October 15, 2023

1 Aim of the Assignment

The objective of *COL215 - Hardware Assignment 2* is to complete a three-part task related to memory generation, image processing, and image display using FPGA hardware. The assignment can be summarized as follows:

1.1 Programming Memories on Board

- Utilize Vivado's memory generator tool to create a ROM (Read-Only Memory) or RAM (Random-Access Memory).
- Populate the generated memory with the provided image file.
- Implement the ability to read the stored image file from the memory.

1.2 Image Gradient Operation

- Utilize the programmed memory to perform an image gradient operation on the loaded image.
- Calculate the gradient of the image to extract vertical edges.

1.3 Displaying Images

- Develop a display controller module in VHDL to drive an externally connected VGA display.
- Display the processed image on the VGA monitor.

In summary, the assignment aims to provide hands-on experience in memory generation, image processing, and FPGA-based image display.

2 ROM Testbench

2.1 Code

2.2 ROM Test Bench Explanation

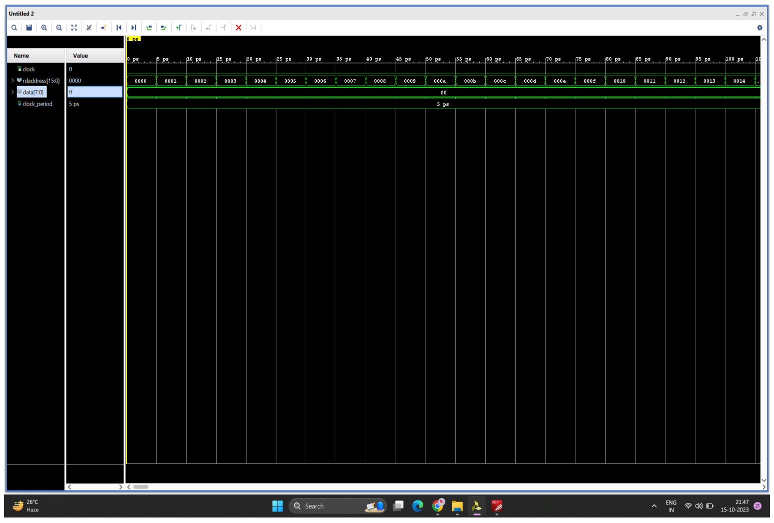
The given VHDL code represents a testbench for a ROM (Read-Only Memory) module. It serves to simulate and verify the behavior of the ROM. Below is a concise explanation of the code's key components:

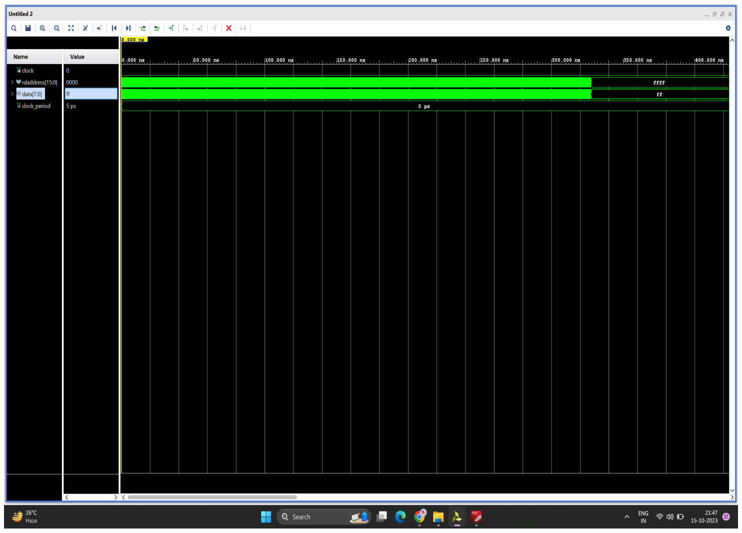
- The code imports various IEEE standard libraries for signal types and numeric operations.
- The testbench entity 'tb_ROM_block' is defined, indicating the start of the testbench.
- The architecture 'behavior' of the testbench instantiates a component 'dist mem gen_0,' which represents the ROM module.
- It declares signals for clock, read address ('rdaddress'), and data output ('data').
- The 'clock_period' constant specifies the clock's period at 5 picoseconds.
- Inside the 'stim_proc' process, a loop iterates through memory addresses (0 to 65535), updating 'rdaddress' for each address and waiting for 5 picoseconds between each address change.

This testbench is designed to stimulate the ROM module by sequentially accessing its memory locations and simulating its operation.

2.3 WaveForm

Attached Below the Waveform of the simulation





3 Gradient Generator

3.1 Code Explanation

Algorithm: Gradient Generation Stimulus Process

Initialize variables and signals for temporary storage (temp1, temp2, temp3, temp4), previous data (prev_prev, prev), current data (curren, current), gradient (grad), and counters (i, j, v).

Begin a process that triggers on the rising edge of clk25.

Check if the write_enable signal is active ('1').

If true, perform the following steps for gradient generation: If j is equal to 0:

Reset previous and previous-previous data to all zeros. Set the rom_address to the appropriate address based on i and j. Update the current variable with data from the ROM. If j is less than 256:

Store the previous data (prev) in prev_prev. Update prev with the current data (current). Set the rom_address accordingly. Get the new data from the ROM and store it in curren. Update current with curren. If j is 256 or greater:

Similar to the previous case, update previous and previous-previous data. But set curren and current to all zeros. Calculate gradient values for the current pixel:

Compute temp1 by adding zero-padded versions of curren and prev_prev. Compute temp2 by adding zero-padded prev to itself. Check whether temp1 is less than temp2.

If true, set the gradient (grad) to all zeros. If temp1 is not less than temp2, calculate the gradient value:

Compute temp3 by subtracting temp2 from temp1. Check if the most significant bit (temp3(8)) of temp3 is set. If the most significant bit is set, set the gradient to all ones. Otherwise, set the gradient to the 7 least significant bits of temp3.

If i is 0, update the ram_address based on the value of j:

If j is 1, set ram_address to all zeros. Otherwise, set ram_address to the previous address minus one. If i is greater than 0, set ram_address based on the previous address minus one.

Set the ram_data_in with the calculated gradient value.

Update the counter j:

If j is less than 255, increment it. If j is 255 and i is less than 256, reset j to 0 and increment i. If j is 255 and i is 256, set write_enable to '0', and reset ram_address to all zeros. If write_enable is '0', check if the current pixel position is within the active region of the display (defined by hPos and vPos). If it is, calculate the ram_address for the RAM based on the current pixel position. End the process.

4 VGA Code

4.1 Code Explanation

Provide a detailed explanation of the VGA code, broken down into the following parts:

4.1.1 Clock Divider

We have implemented Clock Divider using Counters. This way we have reduced the frequency of basys 3 board from 100 MHz to the required frequency 25MHz.

Listing 1: Your VHDL code

```
     clock_div: process(clk)
     begin
        if(rising_edge(clk)) then
            if(countkrnewawla < 1) then
                countkrnewawla <= countkrnewawla+1;
            elsif(countkrnewawla=1) then
                countkrnewawla <= 0;
                clk25<=not clk25;
             end if;
        end if;
     end process;
```

4.1.2 HSYNC Logic

HSync will only be activated in Display Area ,Front Porch and Back Porch.

Listing 2: Your VHDL code

```
Hor_Sync:process(clk25,reset,hpos)
     begin
        if(write_enable ='0') then
            if(reset='1')then
                HSYNC<='0';
            elsif(clk25'event and clk25='1')then
                if((hpos<=(HorHD+HorFP))OR(hpos>HorHD+HorFP+HorSP))
                   then
                    HSYNC<='1';
                else
                    HSYNC<='0';
                end if;
            end if;
        else
            HSYNC<='0';
        end if;
    end process;
```

4.1.3 VSYNC Logic

VSync will only be activated in Display Area ,Vertical Front Porch and Vertical Back Porch.

Listing 3: Your VHDL code

```
Ver_Sync:process(clk25,reset,vpos)
    begin
        if(write_enable ='0') then
            if(reset='1')then
                VSYNC<='0';
            elsif(clk25'event and clk25='1')then
                if((vpos<=(VerHD+VerFP)) OR
                   (vpos>VerHD+VerFP+VerSP))  then
                    VSYNC<='1';
                else
                    VSYNC<='0';
                end if;
            end if;
        else
            VSYNC <='0';
        end if;
    end process;
```

4.1.4 Horizontal Pixel Counter

Listing 4: Your VHDL code

```
Hpos_countkrnewawla:process(clk25,reset)
    begin
        if(write_enable='0') then
            if(reset='1')then
                hpos<=0;
            elsif(clk25'event and clk25='1') then
                if(hpos=(HorHD+HorFP+HorSP+HorBP)) then
                    hpos<=0;
                else
                    hpos<=hpos+1;
                end if;
            end if;
        else
            hpos<=0;
        end if;
    end process;
```

4.1.5 Vertical Pixel Counter

Explain the role of the vertical line counter and how it contributes to the VGA display.

Listing 5: Your VHDL code

```
Vpos_countkrnewawla:process(clk25,reset,hpos)
    begin
        if(write_enable='0') then
            if(reset ='1')then
                vpos<=0;
            elsif(clk25'event and clk25='1') then
                if(hpos = HorHD+ HorFP+HorSP+HorBP) then
                    if( vpos = (VerHD+VerFP+VerSP+VerBP)) then
                        vpos<=0;
                    else
                        vpos<=vpos+1;
                    end if;
                end if;
            end if;
        else
            vpos<=0;
        end if;
    end process;
```

4.1.6 Display Controller Logic

Detail the logic used in the display controller, including how it generates HSYNC, VSYNC signals, and handles image data.

Listing 6: Your VHDL code

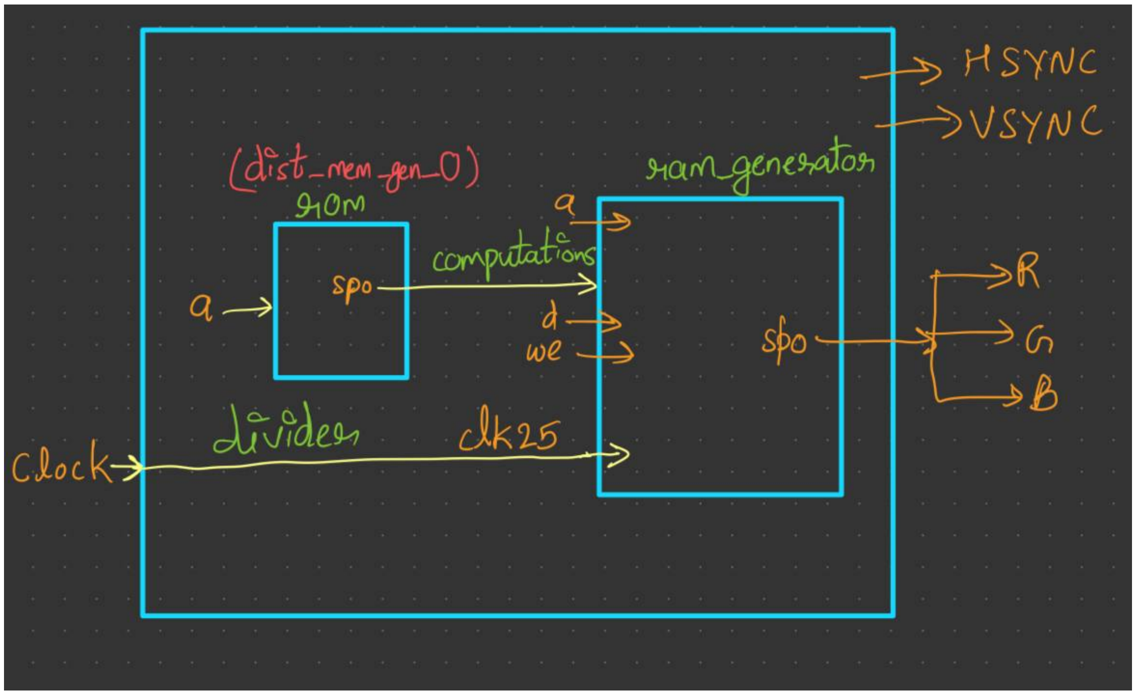
```
Display:process(vidOn,vpos,hpos,clk25,reset,r,ram_address)
      variable l: integer:=0;
    begin
        if(write_enable= '0') then
```

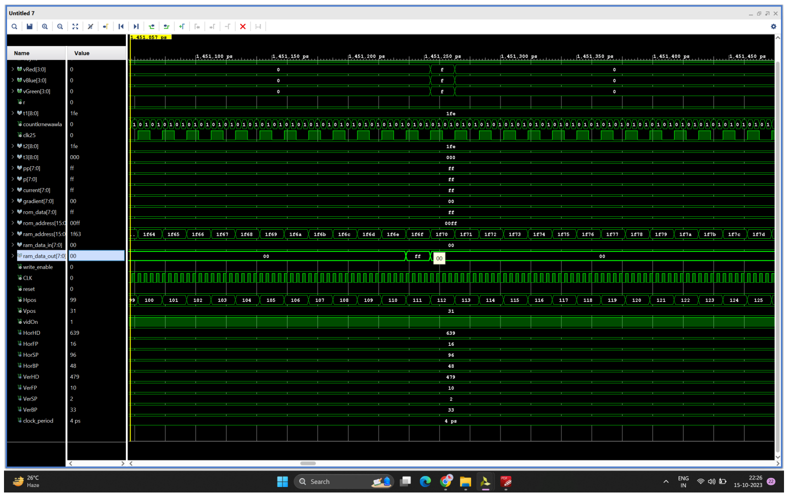
```
            if(reset = '1') then
                vRed <= "0000";
                vBlue <= "0000";
                vGreen <= "0000";
            elsif(rising_edge(clk25)) then
                if(vidOn ='1') then
                    if(vpos>=0 and vpos < 256 ) and (hpos >=0 and hpos <256 ) then
                        vRed <= ram_data_out(7 downto 4);
                        vBlue <= ram_data_out(7 downto 4);
                        vGreen <= ram_data_out(7 downto 4);
                    else
                        vRed <="0000";
                        vBlue <="0000";
                        vGreen <= "0000";
                    end if;
                end if;
            end if;
        else
                vRed <= "0000";
                vBlue <= "0000";
                vGreen <= "0000";
        end if;

    end process;
```

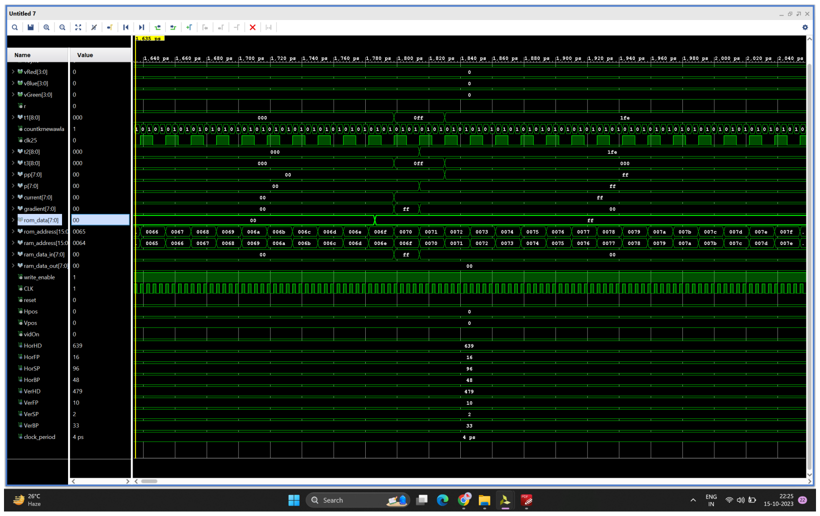
4.2 Block Diagram and Simulation

Include a block diagram illustrating the components and their connections in the VGA module.

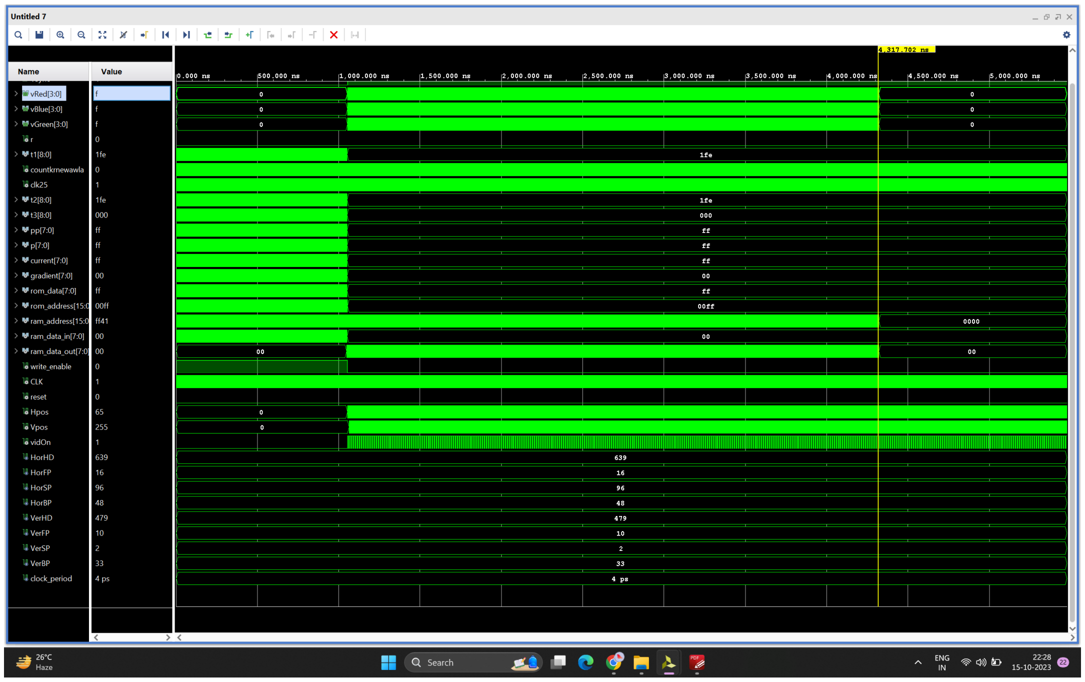




Name	Value
vRed[3:0]	0
vBlue[3:0]	0
vGreen[3:0]	0
r	0
t1[8:0]	1fe
countkrnewawla	0
clk25	0
t2[8:0]	1fe
t3[8:0]	000
pp[7:0]	ff
p[7:0]	ff
current[7:0]	ff
gradient[7:0]	00
rom_data[7:0]	ff
rom_address[15:0	00ff
ram_address[15:0	1f63
ram_data_in[7:0]	00
ram_data_out[7:0]	00
write_enable	0
CLK	0
reset	0
Hpos	99
Vpos	31
vidOn	1
HorHD	639
HorFP	16
HorSP	96
HorBP	48
VerHD	479
VerFP	10
VerSP	2
VerBP	33
clock_period	4 ps

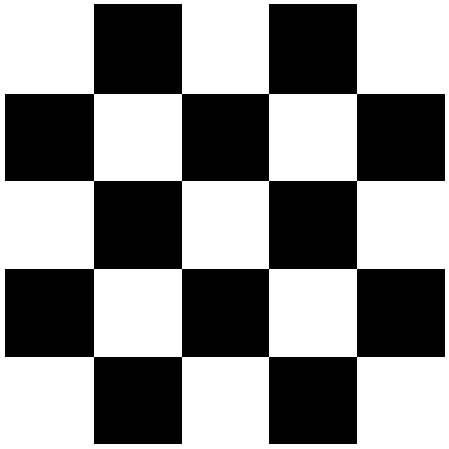


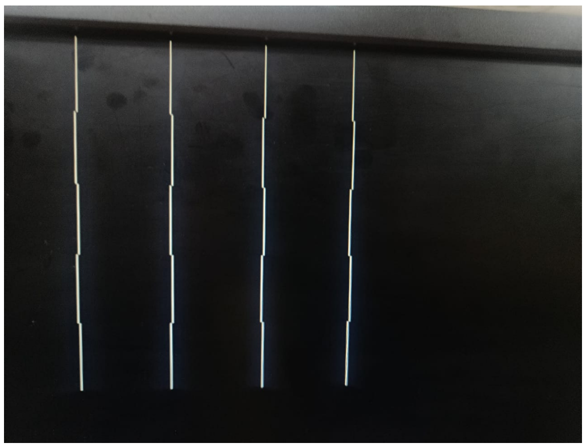
Name	Value
vRed[3:0]	0
vBlue[3:0]	0
vGreen[3:0]	0
r	0
t1[8:0]	000
countkrnewawla	1
clk25	0
t2[8:0]	000
t3[8:0]	000
pp[7:0]	00
p[7:0]	00
current[7:0]	00
gradient[7:0]	00
rom_data[7:0]	00
rom_address[15:0	0065
ram_address[15:0	0064
ram_data_in[7:0]	00
ram_data_out[7:0]	00
write_enable	1
CLK	1
reset	0
Hpos	0
Vpos	0
vidOn	0
HorHD	639
HorFP	16
HorSP	96
HorBP	48
VerHD	479
VerFP	10
VerSP	2
VerBP	33
clock_period	4 ps

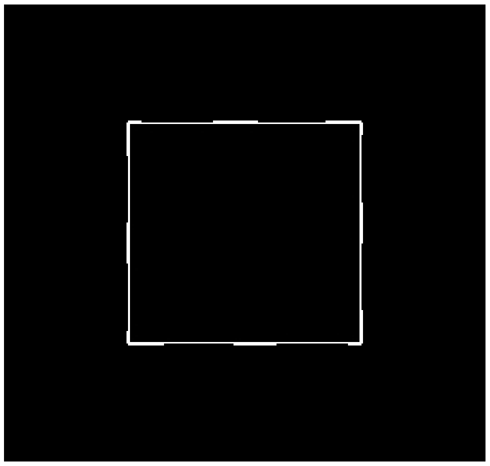


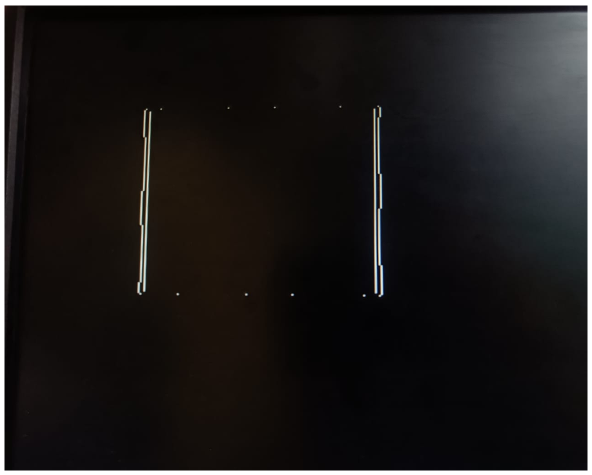
Name	Value
vRed[3:0]	f
vBlue[3:0]	f
vGreen[3:0]	f
r	0
t1[8:0]	1fe
countkrnewawla	0
clk25	1
t2[8:0]	1fe
t3[8:0]	000
pp[7:0]	ff
p[7:0]	ff
current[7:0]	ff
gradient[7:0]	00
rom_data[7:0]	ff
rom_address[15:0	00ff
ram_address[15:0	ff41
ram_data_in[7:0]	00
ram_data_out[7:0]	00
write_enable	0
CLK	1
reset	0
Hpos	65
Vpos	255
vidOn	1
HorHD	639
HorFP	16
HorSP	96
HorBP	48
VerHD	479
VerFP	10
VerSP	2
VerBP	33
clock_period	4 ps

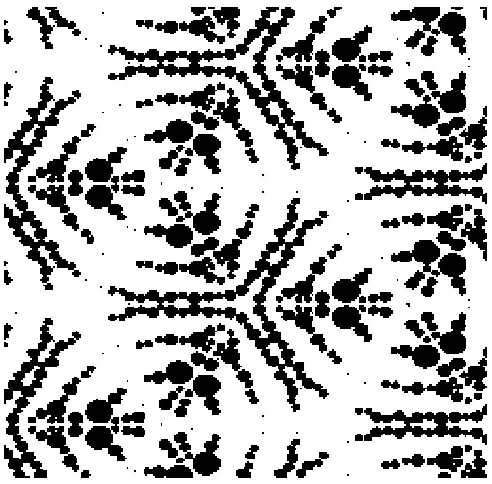
4.3 Chessboard Output and Other Test Cases

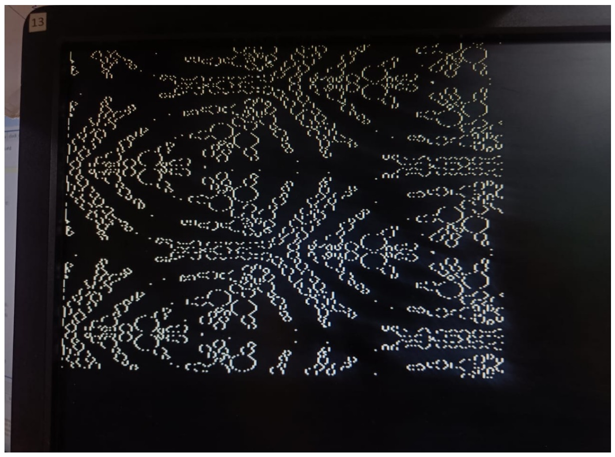












5 Conclusion

Summarize the key points of the report, emphasizing the completion of the assignment's objectives.