Report
COL215 Hardware Assignment 2

Jaskaran Singh Bhalla (2021TT11139) **Saurabh Arge (2022CS11610)**

Problem Statements

- Generating ram and rom using the vivado's memory generator. This memory has to be populated with the image file provided and should be able to read the stored image file.
- Using the memory to carry out the gradient operation.
- Displaying the image stored in the fpga memory before applying the gradient operation and after applying it.

Task 1

We generated the ROM and RAM by using a distributed memory generator in the ip catalogue. We used the distributed memory generator in memories and storage elements and then RAMs & ROMs category.

ROM

For ROM generation, we selected a memory port ROM with depth as 65536 as the image is 256x256 pixels. We initialised it using the given sample file in the RST & initialization tab and named it as **gen_rom.** Width of data was chosen to be 8.
We created a test bench to be able to read the data from the ROM and we were able to see it by running a simulation, analysing the result to ensure that the image is read properly and ensuring that the ROM worked properly.

RAM

FOR RAM generation we selected the single port RAM with depth as 65536 as per the image size. RAM was named **gen_ram.**

Task 2

We used the gradient operation formula given in the assignment to generate cases for calculation of the gradient of a coe image in terms of continuous bitstream of 65536 bits

Logic for gradient operation and implementation

The image size is 256 x 256 pixels. This makes a total of 65536 pixels and this image is stored as a single array of size 65536.

Let I(i) be pixel at i th index. Gradient operation on a pixel is given by
$$O(i, j) = 1 * I(i, j - 1) - 2 * I(i, j) + 1 * I(i, j + 1)$$

There are Three cases :

Case 1:
Whenever there is a left edge of the image , mathematically (i mod 256) = 0.
$$gradient = -2 * I(i) + I(i+1)$$

Case 2 :
Whenever there is a left edge of the image , mathematically (i mod 256) = 255
$$gradient = I(i - 1) - 2 * I(i)$$

Case 3 :
Whenever it is not at the edges.
$$gradient = I(i - 1) - 2 * I(i) + I(i+1)$$

We have sequentially read the ram data in three variables curr_val, nxt_val and prev_val and these are used to evaluate the gradient for each pixel.
We have implemented this logic in the gradient module where we read the data at address i and corresponding next and prev. These are read sequentially using the counter variable t.

Task 3

We were supposed to display the data on the VGA display using the vga display controller on the basys3 board. We have implemented clock divider, horizontal pixel and vertical pixel counter, video on signal and these are used to evaluate hsync and vsync and sendthe vga_red, vga_blue, vga_green and 5 signals sent to the vga to display the image.

Clock divider

We have made a clock divider that converts the given 100MHz to 25MHz. For this, we first made a clock of 50MHz then converted it to 25MHz.

Horizontal pixel counter

This decides the horizontal position on the screen.This will take input the pixel clock and the reset signal. This will count the horizontal count(what is the number of pixels in horizontal line) and the counter will start from 0 until 799 then reset the count to 0. This will be repeated continuously.

Vertical pixel counter

This decides the vertical position on the screen. This will take input to the pixel clock and the reset signal. This will count the vertical count and the counter will start from 0 until 524 then reset the count to 0.This will be repeated continuously. On reaching the end of the counter it will be reset to the beginning of the screen.

Display controller logic

Based on the horizontal count got from the horizontal pixel counter the HSYNC signal is set to high from hcount value 0 to 655 (HACTIVE + HFRONT) then it is 0 from 656 to 751 (HSYNC) then again high from 752 to 799(HBACK).

Similarly for the vertical pixel count ,VSYNC signal is set high from vcount 0 to 489(VACTIVE + VFRONT) then from 490 to 491(VSYNC) then again high from 492 to 524(VBACK).

Similarly the active area for the image is 640x480. We have kept a video on signal which will be active for hcount 0 to 639 and count 0 to 479 and we will display our image in this region.

Structure of code

This section of the report tells us about the structure of the code, modules with the various logics they have. We have made 4 modules :

1. vga.vhd
    a. This module connects all the modules and is the root file of the program
    b. It contains horizontal and vertical synchronisation processes which use hcount and vcount to generate hsync, vsync and video on signals
    c. Display process uses hcount, vcount, clock to display the ram and rom on the screen. ROM display process has been commented and ram can be used.
    d. CLK processes have been used to allow the ram to properly read and write the data.
2. counter.vhd
    - It uses the pixel_clk generated by the clk_divider module to evaluate hcount and vcount signals which decide the position of the pointer on screen.
    - It contains process for both horizontal pixel counter and vertical pixel counter
3. clock_divider.vhd
    - Clock divider creates a 25 MHz clock from the basys3 clock of 100 Mhz
    - It helps to implement the pixel_clk which is passed to the counter module.
4. gradient.vhd
    - This contains the logic for calculating the gradient of the pixel by using the above mathematical formulation. It calculates the gradient for the pixel whose address is passed as an input.

Top level module for the code is **vga.vhd** which uses the rest of the modules.

vga.vhd

```
ENTITY vga IS
PORT (
clk : IN STD_LOGIC;
vga_red : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
vga_blue : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
vga_green : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
hsync : OUT STD_LOGIC;
vsync : OUT STD_LOGIC
);
END vga;
```

gradient.vhd

```
ENTITY gradient IS
PORT (
clk : IN STD_LOGIC;
trig : IN STD_LOGIC;
address : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
data : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
ready : OUT STD_LOGIC
);
END gradient;
```

counter.vhd

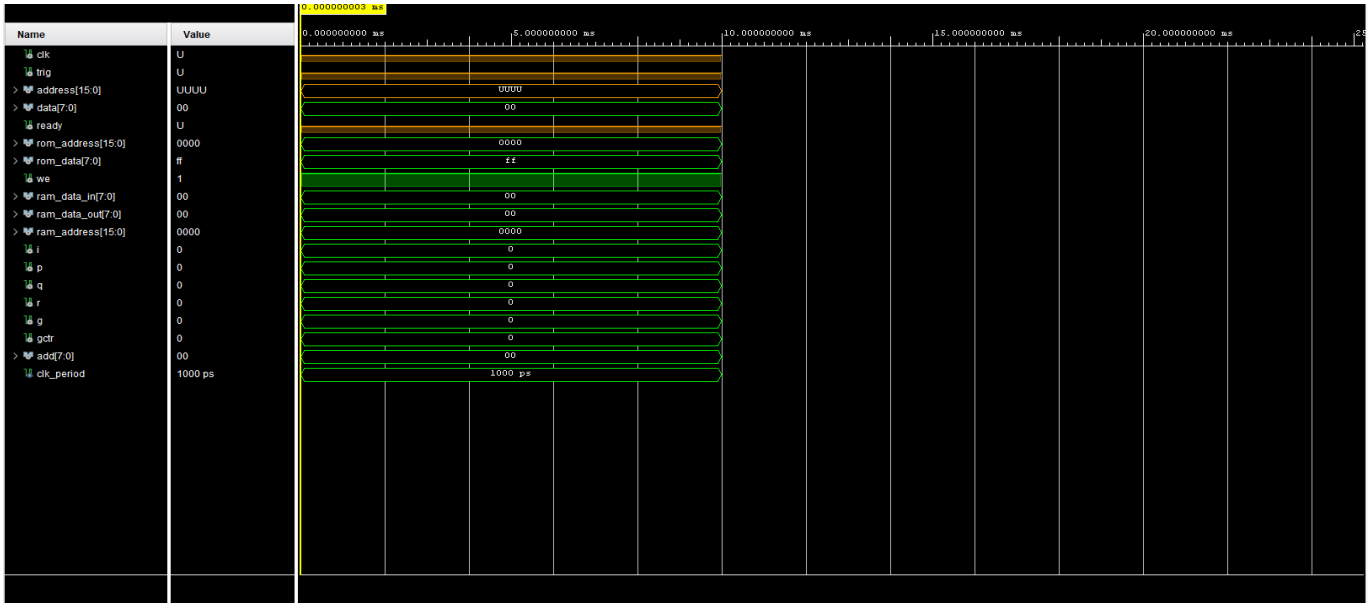
```
ENTITY counter IS
PORT (
pixel_clk : IN STD_LOGIC;
rst : IN STD_LOGIC;
hcount : OUT INTEGER;
vcount : OUT INTEGER
);
END counter;
```

clock_divider.vhd

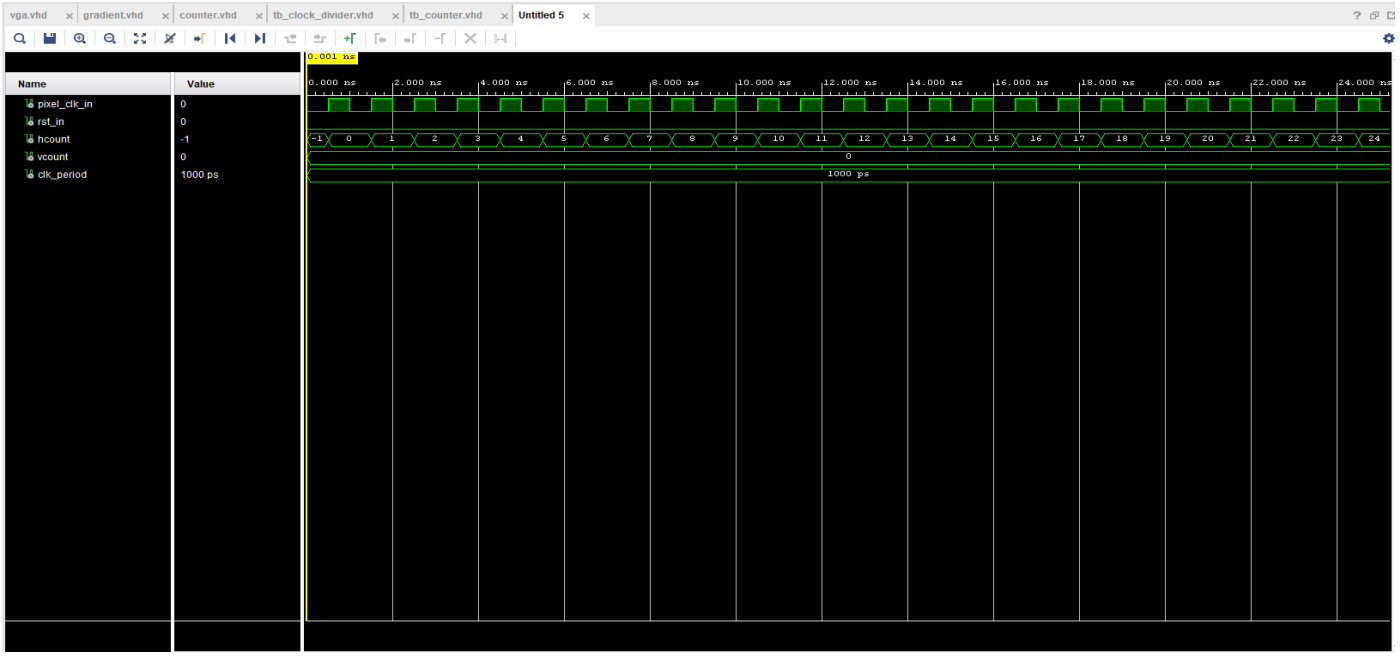
```
ENTITY clock_divider IS
PORT (
clk : IN STD_LOGIC;
pixel_clk : OUT STD_LOGIC
);
END ENTITY;
```

Simulation Diagram

gradient.vhd

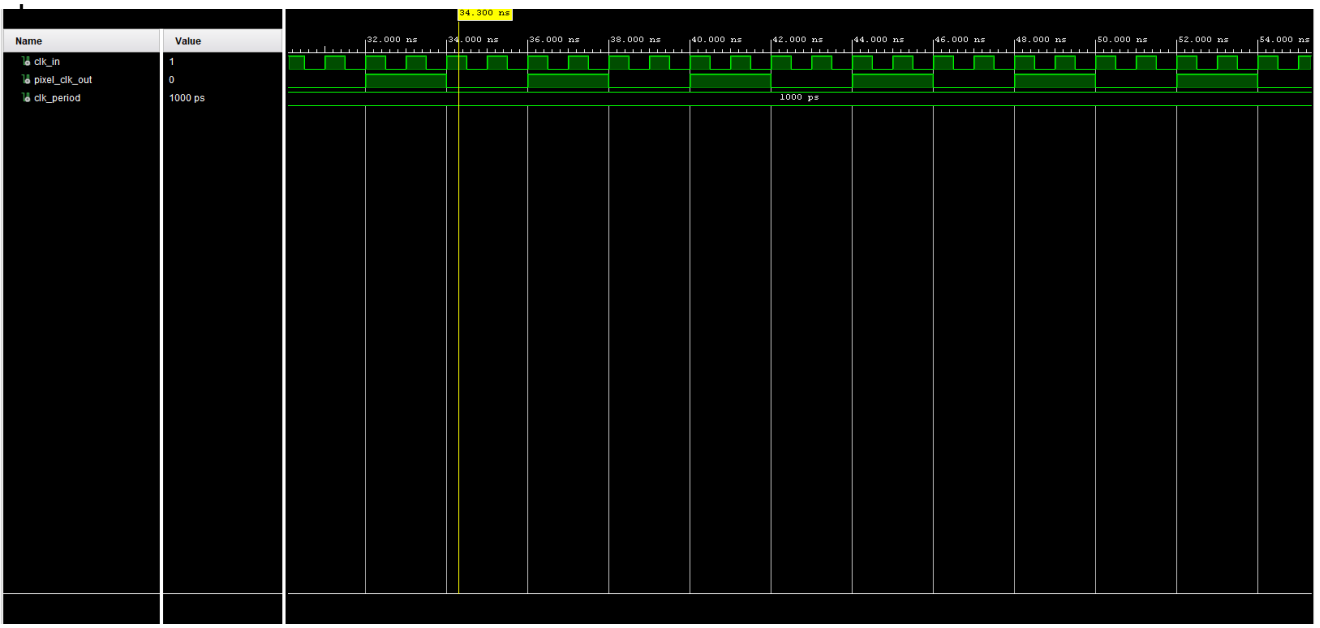


clock_divider.vhd

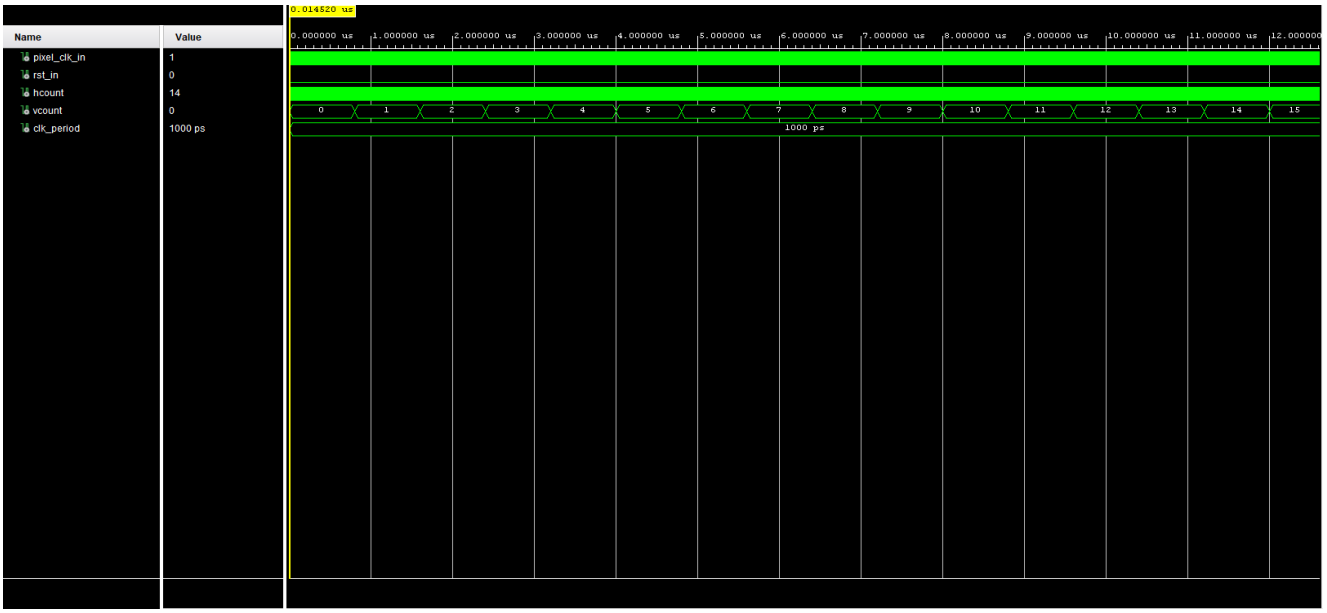


counter.vhd

hcounter.vhd

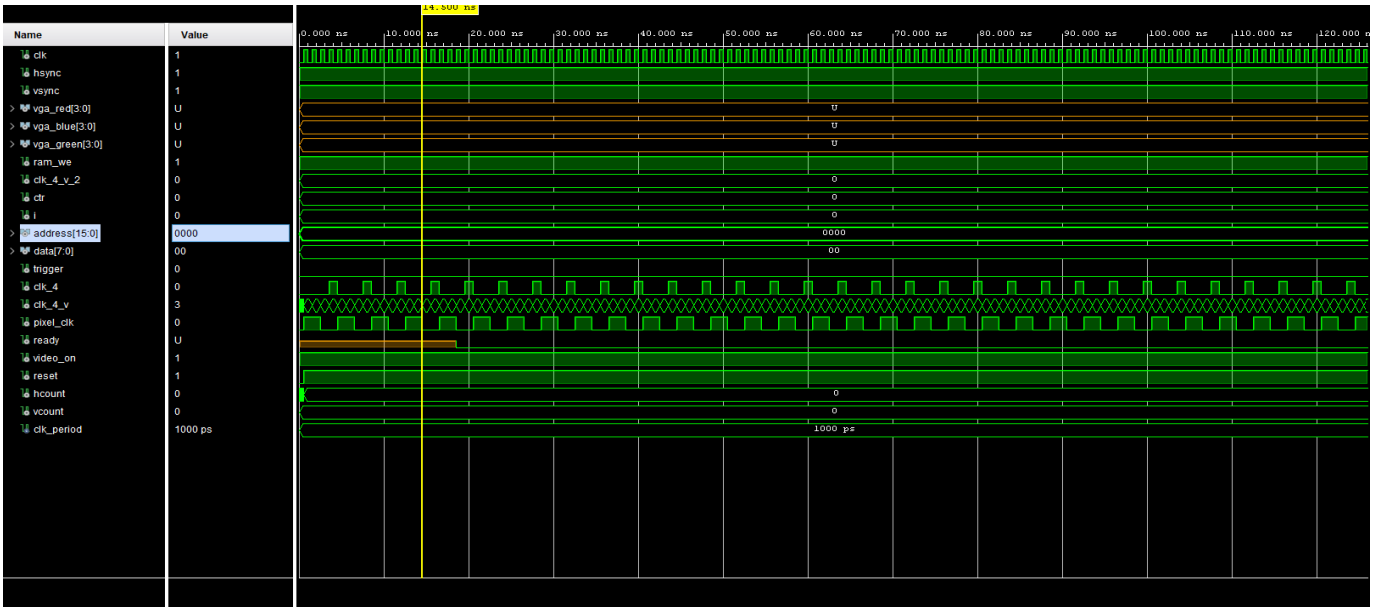


vcounter.vhd

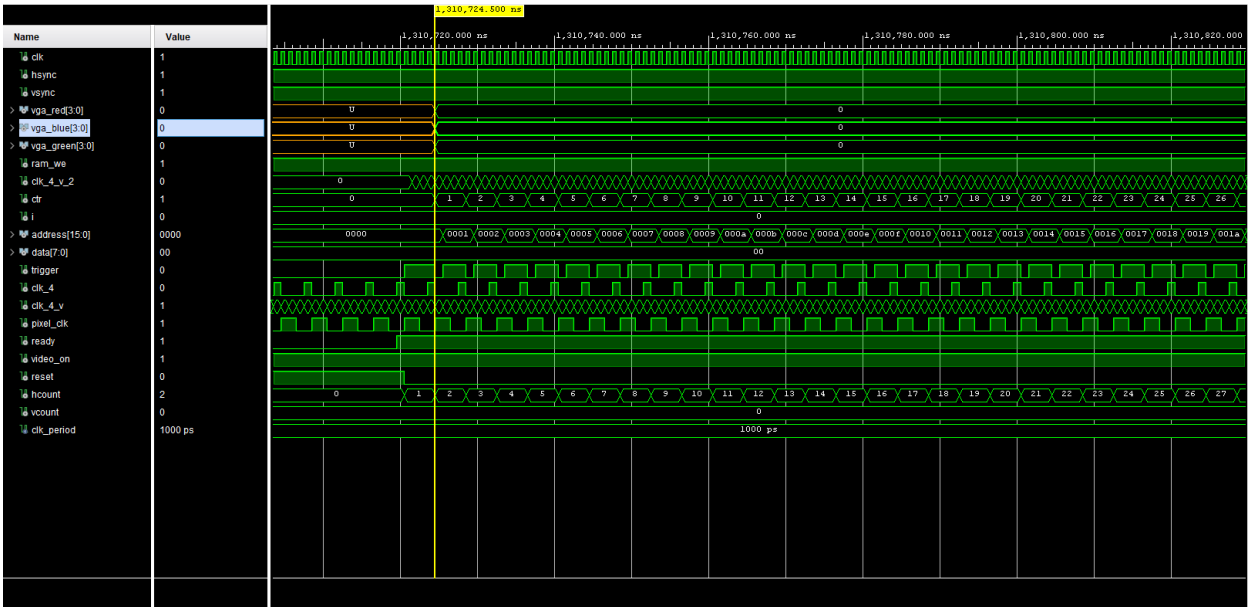


vga.vhd

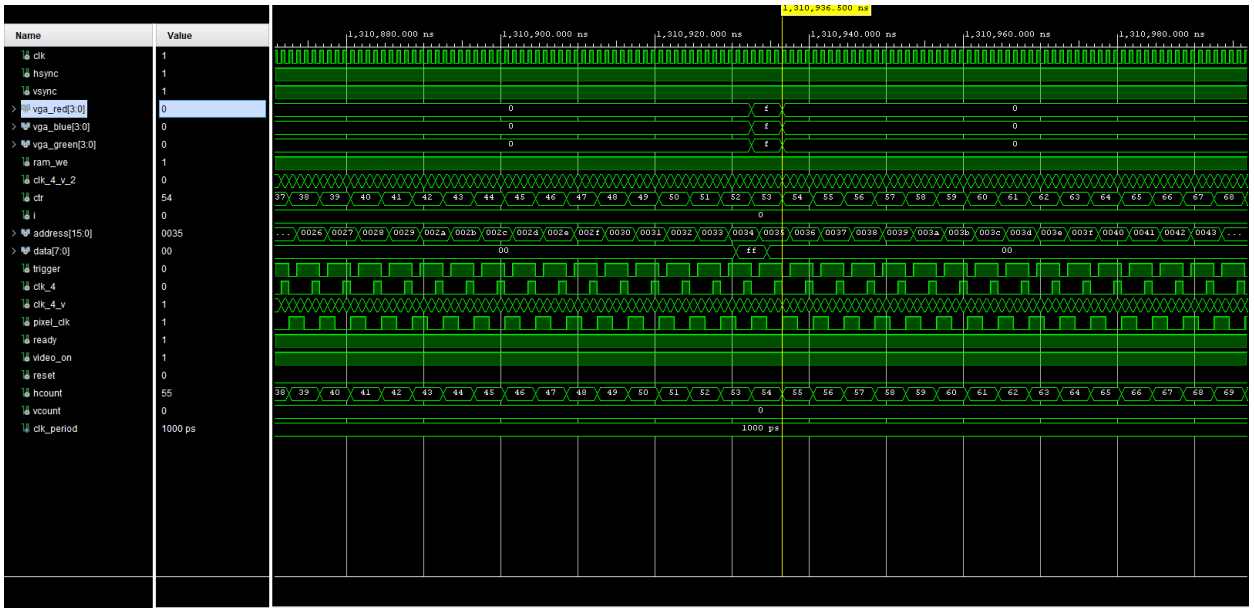
During gradient calculation



After gradient calculation, displaying the data from RAM



Displaying the data from RAM



Displaying the data from ROM

The stimulation for the vga of rom data and clock divider can be seen as pxl_clk in the above diagram

The stimulation for the vga file which includes the gradient and writing to the ram.

Synthesis Report Analysis

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	417	0	0	20800	2.00
LUT as files ic	417	0	0	20800	2.00
LUT as Memory	0	0	0	9600	0.00
Slice Registers	333	0	0	41600	0.80
Register as Flip Flop	333	0	0	41600	0.80
Register as Latch	0	0	0	41600	0.00
F7 Muxes	0	0	0	16300	0.00
F8 Muxes	0	0	0	8150	0.00

1.1 Summary of Registers by Type

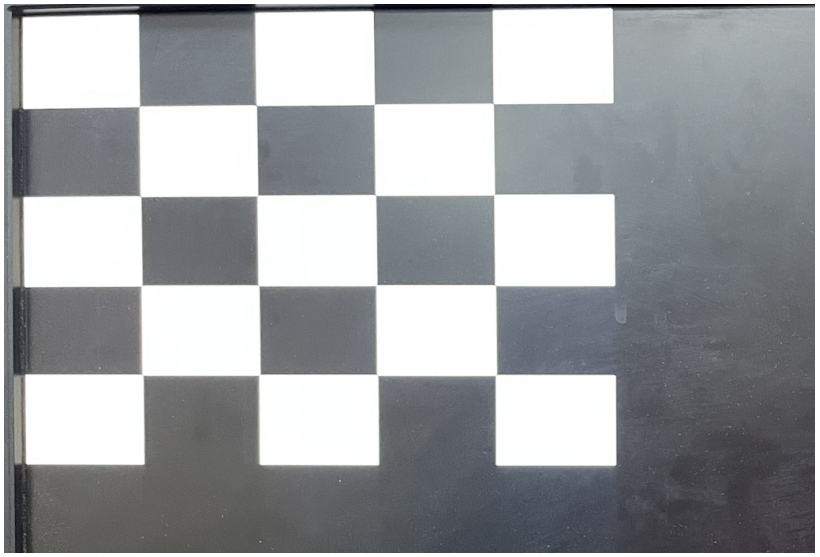
Total	Clock Enable	Synchronous	Asynchronous
0	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
0	Yes	-	Set
96	Yes	-	Reset
8	Yes	Set	-
229	Yes	Reset	-

Rest of the information can be found in the vga_utilization_synth.rpt attached.

Output

ROM

This is the vga controller output of rom data before calculation of gradient.



RAM

This is vga controller output of ram data after calculation of gradient.