Report

COL215 Hardware Assignment 3

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

Problem Statements

- Implementation of a 3x3 image filtering operation.
- Implement FSM (simple or optimised) and integrate it with the hardware design.

Task 1

ROM

Firstly, we have created a **filter_rom** of depth 16 and data width of 8. This contains the image kernel which can be used to calculate the gradient. Secondly, we have created a **gen_rom** of depth 4096 and data width of 8. This contains the image input for which the image operation has to be performed.

RAM

We have created a **gen_ram** of depth 4096 and data width of 8. This has been used to store the final image which is computed after the gradient calculation.

For the Part 1 of the assignment as we were only supposed to compute the gradient, we had created a single module GRADIENT.vhd. This process is done when the filter_ready signal is '0'. This reads 9 values of filter kernel and image in 18 clock cycles. Then using various registers we compute the gradient. And write it on RAM in 2 clock cycles.

Filter Read

We defined variable f1,f2,f3,f4,f5,f6,f7,f8,f9 and stored the value from the given image gradient 3x3 matrix containing negative values in 2's complement form (8 bit binary) by reading one by one by using clock cycles. We defined a signal counter which is integer. In the first cycle we gave the address of the first element from the matrix and in the next cycle we stored the data in the defined variable f1, similarly it continues till all 9 values are stored. Then the signal filter_ready is set to '1' after the filter values are read.

Gradient

The output pixel value O(i, j) at location (i, j) is computed as the sum of element-wise multiplications between kernel value and input image pixel, as **shown in the equation below. The input image pixel at location (i, j) is denoted I(i, j).**

O(i, j) = a*I(i − 1, j − 1) + b*I(i − 1, j) + c*I(i − 1, j + 1) +d*I(i, j − 1) + e*I(i, j) + f*I(i, j + 1) +g*I(i + 1, j − 1) + h*I(i + 1, j) + i*I(i + 1, j + 1)

For finding output pixel value here we have defined v1, v2, v3, v4, v5, v6, v7, v8, v9 variables and also max_v and min_v which will be used in the normalization part for outer pixel value.

Here v1,v2 … represent the corresponding I(i-1, j-1) , I(i-1,j)… values . we started the process from i=0 and when counter is 1 we assign the address of I(i-1,j-1) if it exists then in the next cycle we assign the rom data value to v1 if it exists else it is set as 0, this is repeated till all the nine values are obtained then in the next value we calculate the output pixel value by the above given formula.Then in the next cycle we updated the max_v and the min_v values. This process is continued until i=4095 then at the end we get the maximum minimum value of outer pixel values for the given image. Stimulation for coins .coe file

Here the output pixel value is stored in signal g when i changes to one the g value becomes 510 and this happens when counter is 19 when counter is 20 min_v and max_v is updated then after counter become 21 it set to 0 and this continue and we can see g changes to 256 this is for i=1.
Here in the above image i=4095 means it has calculated all the output pixels of the given image and we got the max_v and min_v pixel values in the given image.

Image Normalisation

Output value is clamped between 0-255. This is done to successfully write it on RAM.
It uses the following formula provided inorder to do this.

New_I (i, j) = (I (i, j) − min) 255 / (max – min)

When trig_normalize is ‘1’ then the normalization process will be triggered here, the variable j goes from j=0 to j=4095 and depends on counter_2. Here, we calculate the gradient again by reading the kernel and input image values and when counter_2 is 20 we apply the normalization to the pixel. we enable the ‘we’ (write enable) and give the j value to ram address and wait for 2 cycles to allow the data to be written on ram, then j is incremented.

Here ng signal represents the final pixel value. When the counter_2 is 20 it is updated to 253 for j=0 then we have waited for three cycle to write in ram then the process continuous till j=4095

Here ng is the final output pixel for corresponding j value and the difference of three clock cycles because ng is calculated when counter_2 is 20 and j is incremented when counter_2 is 23.

Task 2

We have created 6 modules namely

1. MAC.vhd
2. FMMN_MODULE.vhd
3. FSM.vhd
4. CLOCK_DIVIDER.vhd
5. COUNTER.vhd
6. VGA.vhd

MAC

A Multiplier-Accumulator block (MAC) has been created to perform element wise multiplication and accumulation to get one output pixel. This takes the 9 input values and 9 values of kernel one by one and then Multiplier (M) multiples the input value and kernel value and stores it in M and Accumulator (A) adds the value to itself. A CTRL has been established to control this unit using the FSM we have created and RST to reset the state of the MAC unit whenever it is required.

Find Minimum & Maximum and Normalize (FMMN_MODULE)

This unit performs 2 operations depending upon the state in which this unit is. One of the operations of the unit is to find out the maximum and minimum value of the gradient which will be later normalized. Another function is to normalize the gradient to fit it in 0-255 as mentioned in the assignment. So, initially it operates in state = ‘1’ where FSM operates in FMMN state in which we find the max and min. This module is directly connected to the output of the MAC unit, when the MAC unit has successfully calculated its value, this unit compares the value to the current maximum and minimum and updates it accordingly. Secondly it operates during the WRITE_RAM state of FSM when this same unit performs the normalization operation of the value that is received from the MAC unit. This final value is sent as an output to the FSM where it is mapped to the input of the ram and is written on the RAM.

Finite State Machine (FSM)

We have controlled the entire program using a single FSM only. This FSM consists of 3 states, FMMN, WRITE_RAM and VGA_DISPLAY. Initially the program starts with FMMN, in this state as mentioned above we calculate the Maximum and Minimum using the FMMN_MODULE module. We do so by using a counter which reads 9 values over 18 clock cycles from both the ROMs. here we have improved our code from the previous gradient. We have also considered the edge cases which lie towards the edges of the image. After this state of FSM is changed to WRITE_RAM. During the second state WRITE_RAM. We again read the values from the two rom's and this time in addition to calculation of gradient using the MAC unit, we use the FMMN_MODULE unit to normalize the value using the previously calculated maximum and minimum value. This value is sent to the input of the RAM and when the Write Enable (WE) turns '1' then this value is written on the RAM. Here we have also considered edge cases while calculating the pixels. After this the state of FSM is changed to VGA_DISPLAY so that we are able to display the output on Monitor using the VGA controller. Here we change the state of RAM so that it can be read and send the output value, corresponding to the ADDRESS which is input in the FSM module.

Display

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, pixel counter (horizontal and vertical) , 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 64x64. 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

- This module connects all the modules and is the root file of the program.
- It contains horizontal and vertical synchronisation processes which use hcount and vcount to generate hsync, vsync and video on signals.
- 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.
- 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 processes 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.FSM.vhd

- This is the FSM which controls the entire logic.
- It contains 3 states where FMMN state calculates the initial gradient and finds the maximum and minimum
- WRITE_RAM calculates the normalised gradient and VGA_DISPLAY allows us to display the values using the VGA Controller
- This contains the logic for calculating the gradient of the pixel by using the above mathematical formulation.

5. MAC.vhd

- This unit is a basic Multiplier and Accumulator unit which contains the logic for performing the addition and multiplication using a minimum amount of registers.

6.FMMN_MODULE.vhd

- This module contains two logics, one is the logic for finding the maximum and minimum gradient.
- Then this module normalises the gradient using these max and min values to calculate the 8 bit output value based upon the state of FSM.

Top level module for the code is **vga.vhd** which contains the rest of the modules and uses them as components.

MAC.vhd

```
ENTITY MAC IS
  PORT (
     CLK : IN STD_LOGIC;
     INP_VAL : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
     KERNEL : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
     RST : IN STD_LOGIC;
     CTRL : IN STD_LOGIC;
     OUT_MAC : OUT INTEGER
  );
END MAC;
```

FMMN_MODULE.vhd

```
ENTITY FMMN_MODULE IS
   PORT (
       CLK : IN STD_ULOGIC;
       CTRL : IN STD_LOGIC;
       RST : IN STD_LOGIC;
       STATE : IN STD_LOGIC;
       INP_VAL : IN INTEGER;
       OUT_VAL : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
   );
END ENTITY;
```

FSM.vhd

```
ENTITY FSM IS
   PORT (
       CLK : IN STD_ULOGIC;
       ADDRESS : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
       FSM_OUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
   );
END FSM;
```

CLOCK_DIVIDER.vhd

```
ENTITY CLK_DIV IS
   PORT (
       CLK : IN STD_LOGIC;
       PIXEL_CLK : OUT STD_LOGIC
   );
END ENTITY;
```

COUNTER.vhd

```
ENTITY COUNTER IS
   PORT (
       PIXEL_CLK : IN STD_LOGIC;
       RST : IN STD_LOGIC;
       HCOUNT : OUT INTEGER;
       VCOUNT : OUT INTEGER
   );
END COUNTER;
```

VGA.vhd

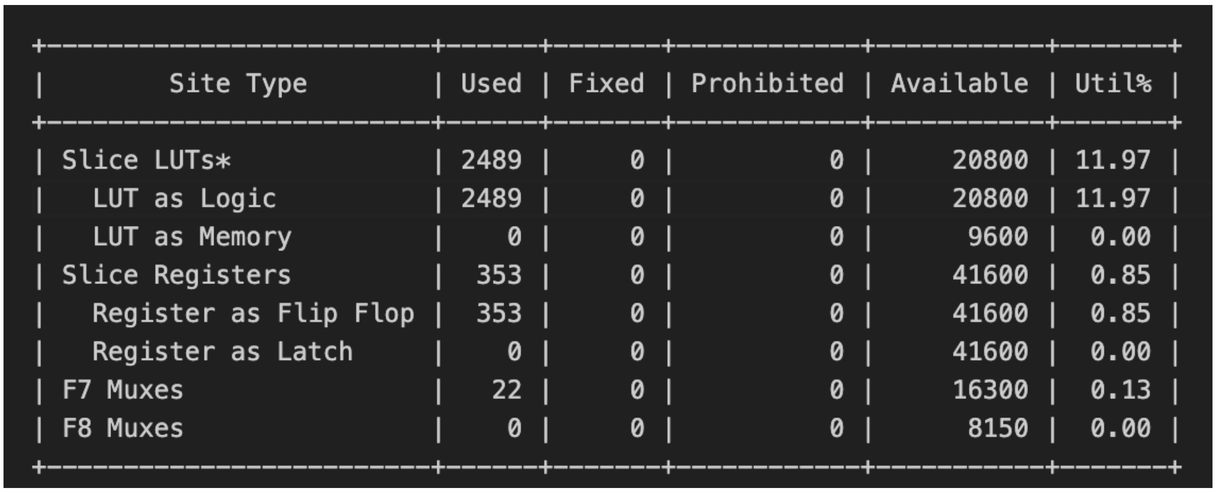
```
ENTITY VGA IS
      PORT (
          clk : IN STD_LOGIC;
          vgaRed : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
          vgaBlue : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
          vgaGreen : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
          hsync : OUT STD_LOGIC;
          vsync : OUT STD_LOGIC
      );
END VGA;
```

Output

Synthesis Report Analysis

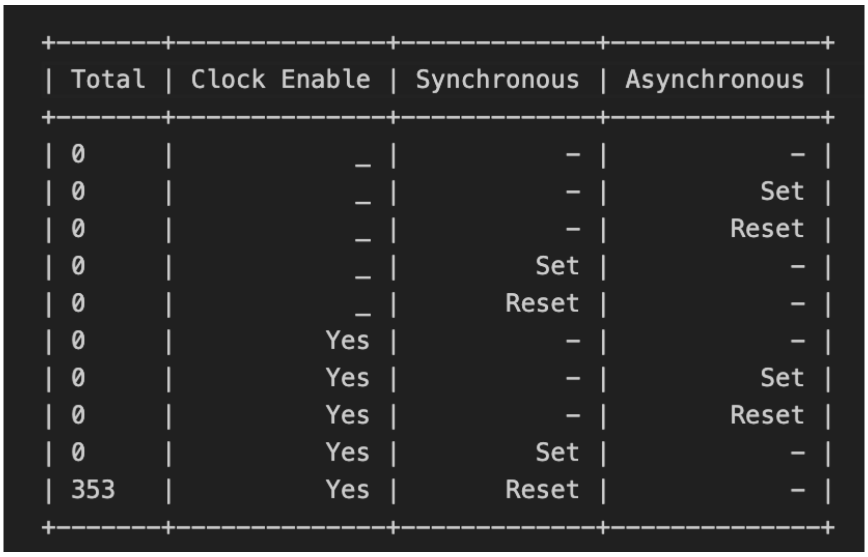
We have attached the synthesis rpt files for all the three sample input files in the submission.

Slice Logic



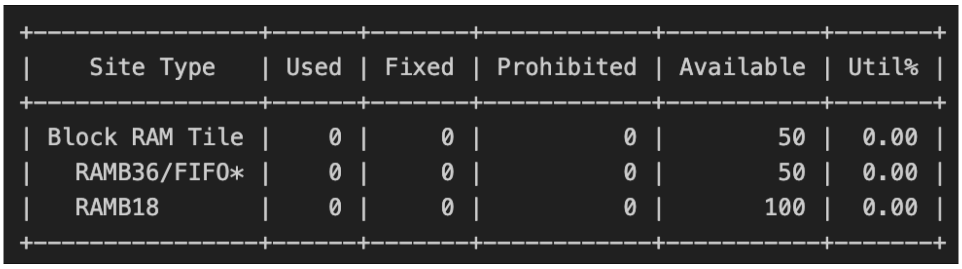
```
+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             | 2489 |     0 |          0 |     20800 | 11.97 |
|   LUT as Logic          | 2489 |     0 |          0 |     20800 | 11.97 |
|   LUT as Memory         |    0 |     0 |          0 |      9600 |  0.00 |
| Slice Registers         |  353 |     0 |          0 |     41600 |  0.85 |
|   Register as Flip Flop |  353 |     0 |          0 |     41600 |  0.85 |
|   Register as Latch     |    0 |     0 |          0 |     41600 |  0.00 |
| F7 Muxes                |   22 |     0 |          0 |     16300 |  0.13 |
| F8 Muxes                |    0 |     0 |          0 |      8150 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

Registers by Type



```
+-------+--------------+-------------+--------------+
| Total | Clock Enable | Synchronous | Asynchronous |
+-------+--------------+-------------+--------------+
| 0     |            _ |           - |            - |
| 0     |            _ |           - |          Set |
| 0     |            _ |           - |        Reset |
| 0     |            _ |         Set |            - |
| 0     |            _ |       Reset |            - |
| 0     |          Yes |           - |            - |
| 0     |          Yes |           - |          Set |
| 0     |          Yes |           - |        Reset |
| 0     |          Yes |         Set |            - |
| 353   |          Yes |       Reset |            - |
+-------+--------------+-------------+--------------+
```

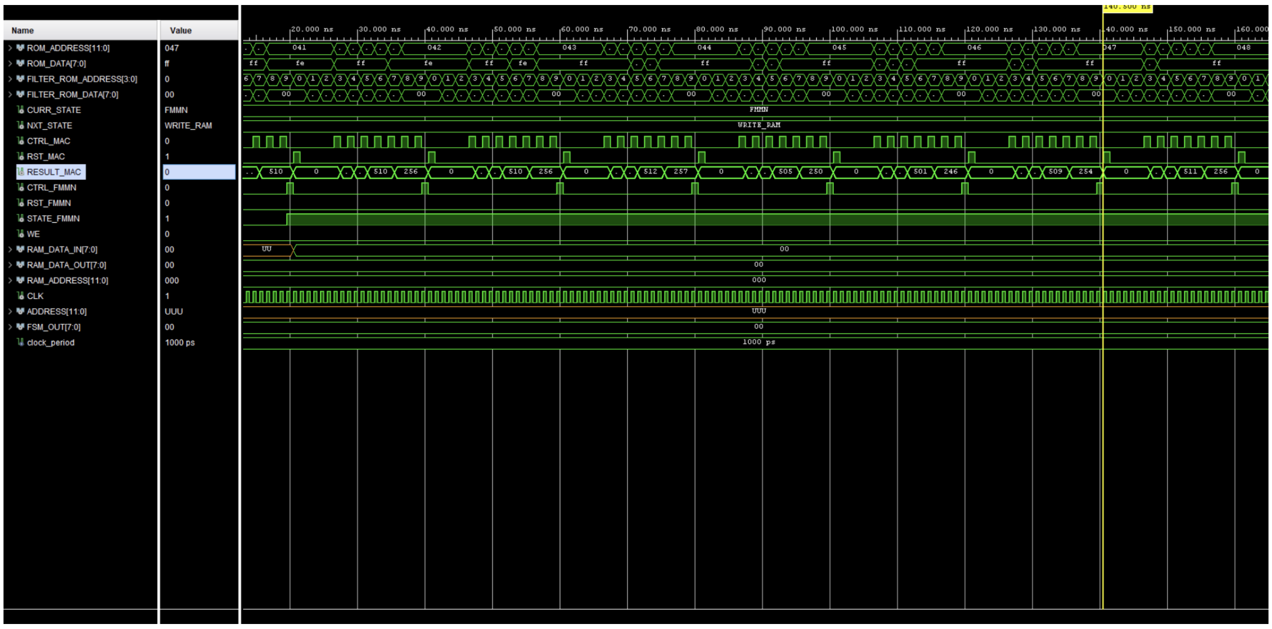
Memory



```
+-----------------+------+-------+------------+-----------+-------+
|    Site Type    | Used | Fixed | Prohibited | Available | Util% |
+-----------------+------+-------+------------+-----------+-------+
| Block RAM Tile  |    0 |     0 |          0 |        50 |  0.00 |
|   RAMB36/FIFO*  |    0 |     0 |          0 |        50 |  0.00 |
|   RAMB18        |    0 |     0 |          0 |       100 |  0.00 |
+-----------------+------+-------+------------+-----------+-------+
```

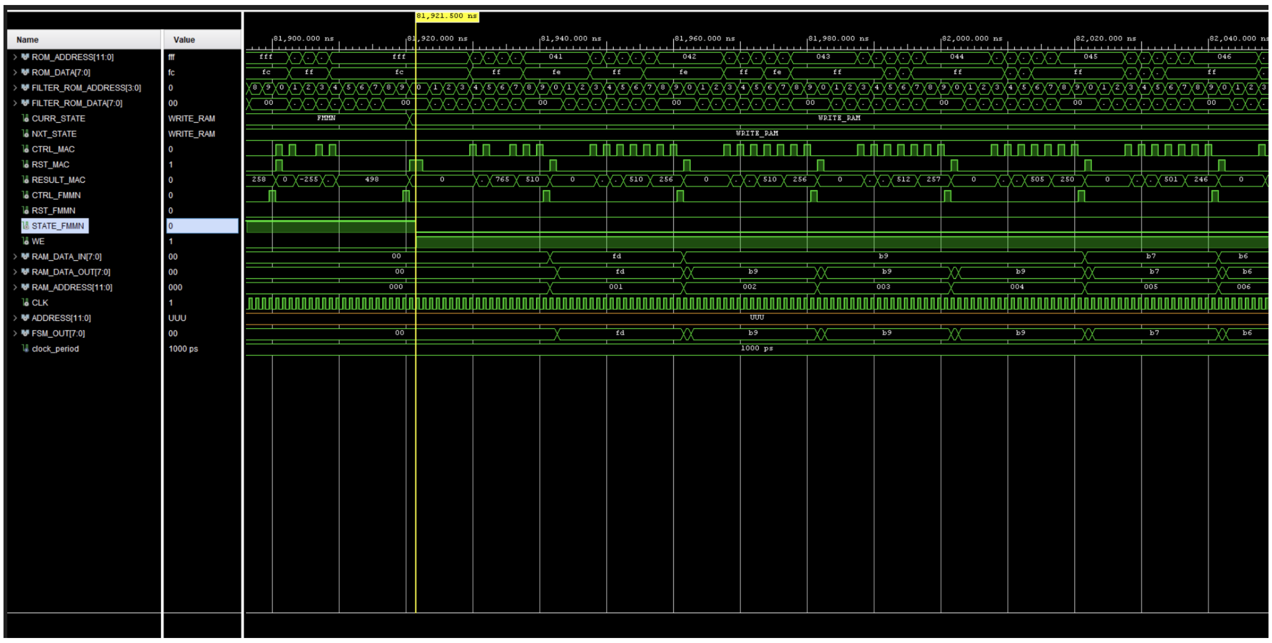
Simulation Results

FSM.vhd - FMMN State



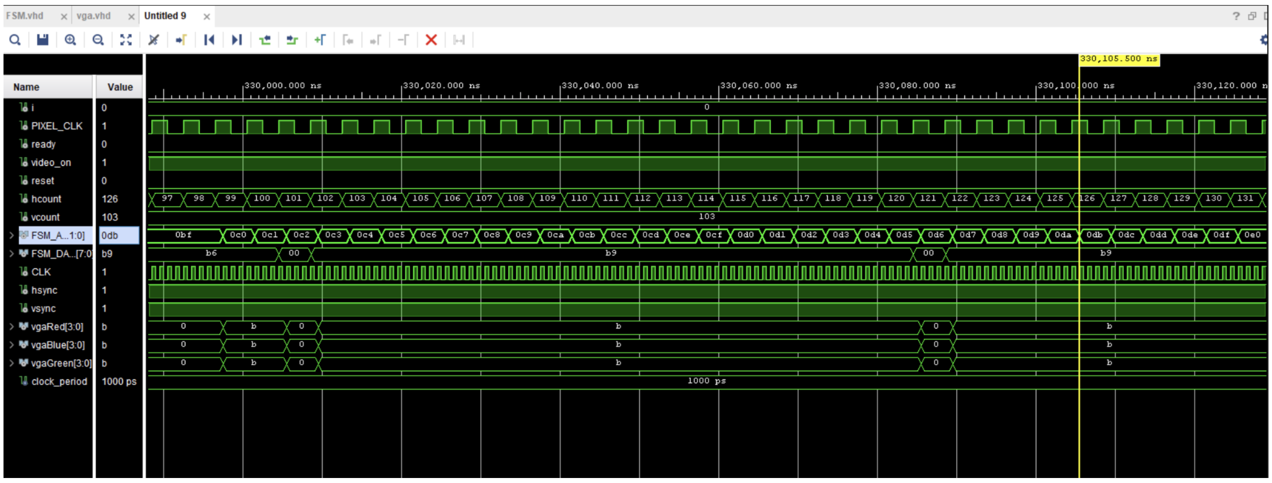
Name	Value
ROM_ADDRESS[11:0]	047
ROM_DATA[7:0]	ff
FILTER_ROM_ADDRESS[3:0]	0
FILTER_ROM_DATA[7:0]	00
CURR_STATE	FMMN
NXT_STATE	WRITE_RAM
CTRL_MAC	0
RST_MAC	1
RESULT_MAC	0
CTRL_FMMN	0
RST_FMMN	0
STATE_FMMN	1
WE	0
RAM_DATA_IN[7:0]	00
RAM_DATA_OUT[7:0]	00
RAM_ADDRESS[11:0]	000
CLK	1
ADDRESS[11:0]	UUU
FSM_OUT[7:0]	00
clock_period	1000 ps

FSM.vhd - WRITE_RAM State



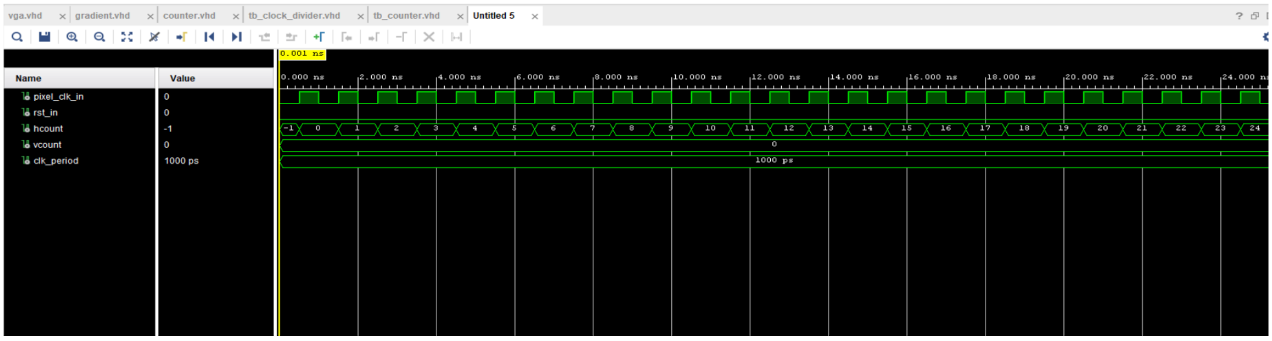
Name	Value
ROM_ADDRESS[11:0]	fff
ROM_DATA[7:0]	fc
FILTER_ROM_ADDRESS[3:0]	0
FILTER_ROM_DATA[7:0]	00
CURR_STATE	WRITE_RAM
NXT_STATE	WRITE_RAM
CTRL_MAC	0
RST_MAC	1
RESULT_MAC	0
CTRL_FMMN	0
RST_FMMN	0
STATE_FMMN	0
WE	1
RAM_DATA_IN[7:0]	00
RAM_DATA_OUT[7:0]	00
RAM_ADDRESS[11:0]	000
CLK	1
ADDRESS[11:0]	UUU
FSM_OUT[7:0]	00
clock_period	1000 ps

VGA.vhd / FSM.vhd VGA_DISPLAY state



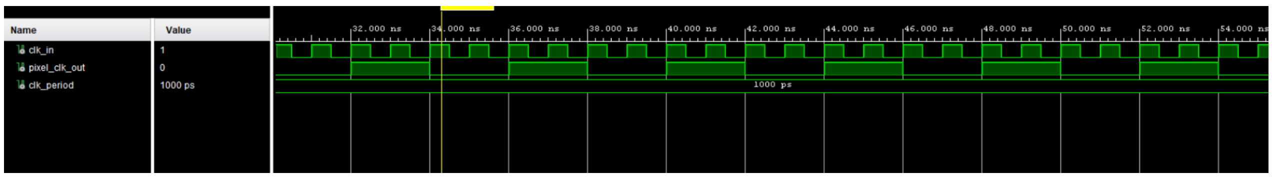
Name	Value
i	0
PIXEL_CLK	1
ready	0
video_on	1
reset	0
hcount	126
vcount	103
FSM_A...1:0]	0db
FSM_DA...[7:0]	b9
CLK	1
hsync	1
vsync	1
vgaRed[3:0]	b
vgaBlue[3:0]	b
vgaGreen[3:0]	b
clock_period	1000 ps

CLOCK_DIVIDER.vhd

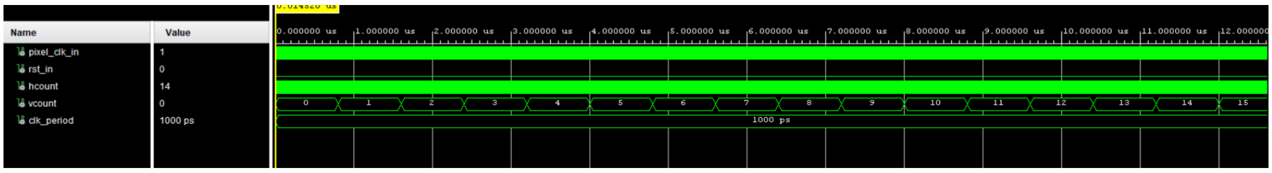


Name	Value
pixel_clk_in	0
rst_in	0
hcount	-1
vcount	0
clk_period	1000 ps

COUNTER.VHD



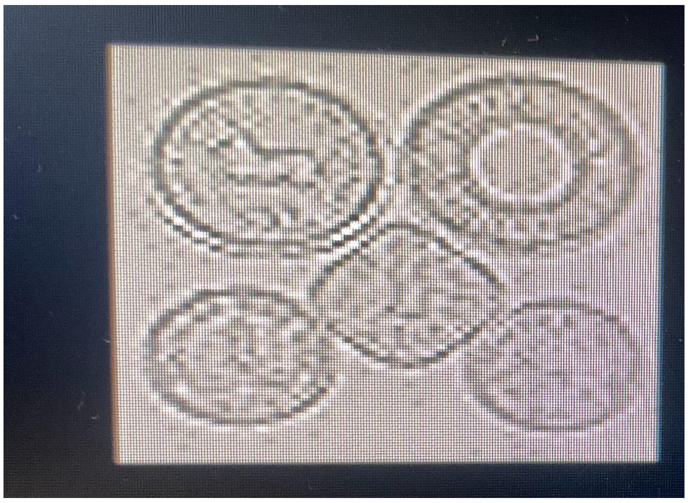
Name	Value
clk_in	1
pixel_clk_out	0
clk_period	1000 ps



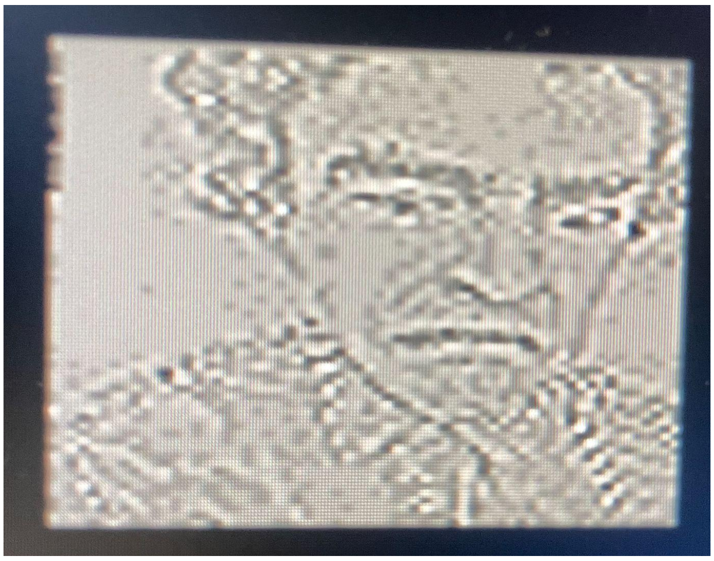
Name	Value
pixel_clk_in	1
rst_in	0
hcount	14
vcount	0
clk_period	1000 ps

Output Images

Coins.coe



Face.coe



Lighthouse.coe