COL215P - Hardware Assignment 2

Submission Deadline: 3rd September'23

1. Problem Statement

This assignment has three parts.

a) Using Vivado's memory generator, RAM/ROM has to be generated. This memory has to be populated with the image file provided and should be able to read the stored image file.

b) Using this memory you need to carry out image gradient operation.

c) You should be able to display the image stored in the FPGA memory before and after the transformation is carried out on the image .

2. Description

2.1 Programming Memories on Board

We would be using Vivado's memory generator to design programs and data memories. Instructions are as follows,

- In Vivado, go to the Project Manager window on the left side of the screen which looks like the screen-shot shown on right.
- Open IP catalog. You can also open IP Catalog through a drop down menu after clicking on Window in the main menu on the top of the screen. "IP" (Intellectual Property) refers to pre-designed modules available for use. The IP catalog lists various IPs category-wise.
- Select the category **Memories & Storage Elements** and sub-category **RAMs & ROMs**, as shown below.

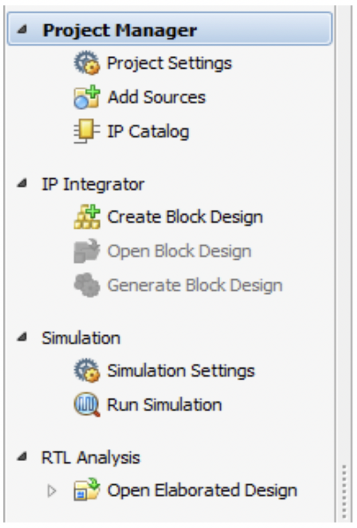
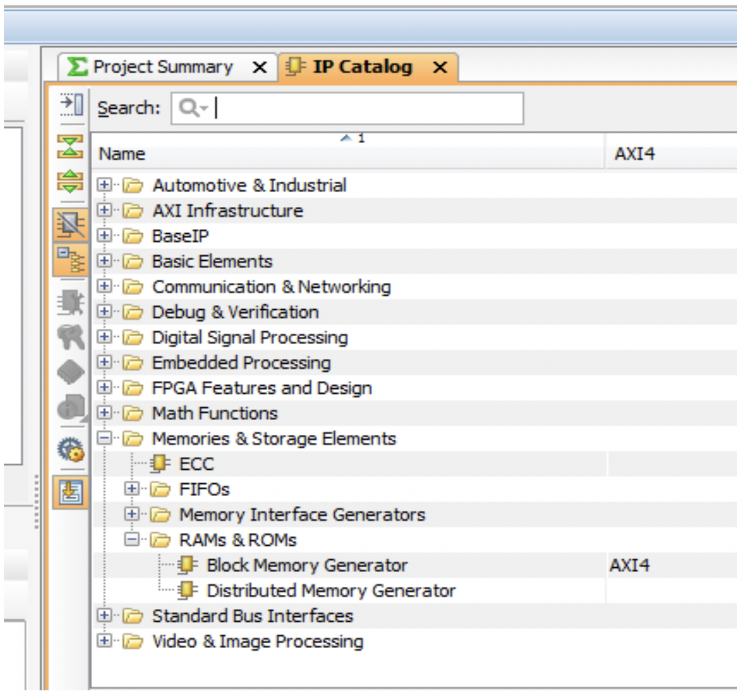


Figure 1a: Project Manager



Project Summary | IP Catalog

Search:

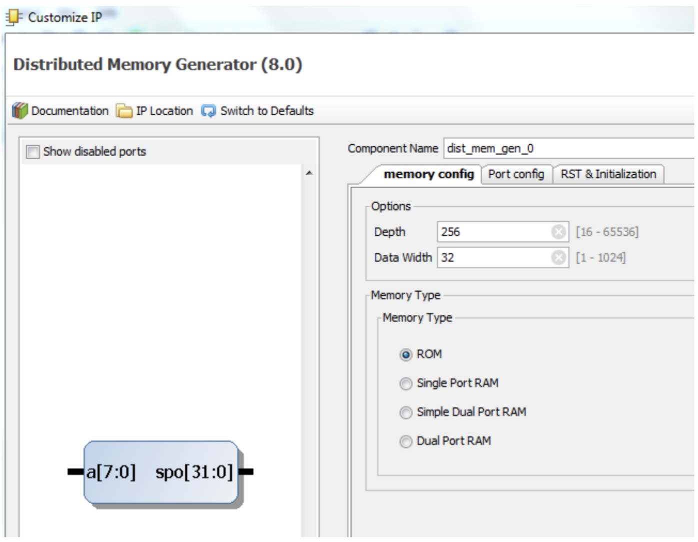
Name	AXI4
Automotive & Industrial	
AXI Infrastructure	
BaseIP	
Basic Elements	
Communication & Networking	
Debug & Verification	
Digital Signal Processing	
Embedded Processing	
FPGA Features and Design	
Math Functions	
Memories & Storage Elements	
ECC	
FIFOs	
Memory Interface Generators	
RAMs & ROMs	
Block Memory Generator	AXI4
Distributed Memory Generator	
Standard Bus Interfaces	
Video & Image Processing	

Figure 1b: IP Catalog

- Now choose ***Distributed Memory Generator***, to create a ROM / RAM as per your design requirement. This generator creates memory modules of specified sizes using LUTs of FPGA. The reason for using the Distributed Memory Generator here is that ***Block Memory Generator*** creates a memory with an address register since they use FPGA BRAMs, and are mostly preferred when we want large on-chip memories.

  The distributed RAMs are faster and more efficient because they use the LUTs in contrast to FPGA BRAMs.

  The Distributed Memory Generator opens a window with three tabs. In the ***memory config tab***, specify memory size and type. The screen-shot shown below is for a ROM with 256 words, each of size 32 bits.



Customize IP

Distributed Memory Generator (8.0)

Documentation | IP Location | Switch to Defaults

Show disabled ports

Component Name: dist_mem_gen_0

memory config | Port config | RST & Initialization

Options

Depth: 256 [16 - 65536]

Data Width: 32 [1 - 1024]

Memory Type

Memory Type

[✓ ROM] Single Port RAM Simple Dual Port RAM Dual Port RAM

Figure 2: Customize IP

- In the ***RST & Initialization*** tab, specify the name of the file that defines the ROM contents. A sample file named *sample.coe* is available on moodle.

  Now instead of passing the inputs from the test bench or using "force value" construct, please initialize the inputs using a *coe* file and pass input to the DUT by connecting it to the memory module.

Once you synthesize and implement your design, you will see a change in the number of LUTs used in your design.

- **TestBench to test the memory module:**

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;

ENTITY tb_ROM_block IS
END tb_ROM_block;
ARCHITECTURE behavior OF tb_ROM_block IS
   COMPONENT blk_mem_gen_0
   PORT(
      clka : IN STD_LOGIC;
      addra : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
      );
   END COMPONENT;
  --Inputs
  signal clock : std_logic := '0';
  signal rdaddress : std_logic_vector(15 downto 0) := (others => '0');
  --Outputs
  signal data : std_logic_vector(7 downto 0) := (others => '0');

  -- Clock period definitions
  constant clock_period : time := 10 ns;

  signal i: integer;
BEGIN
 -- Read image in VHDL
  uut: blk_mem_gen_0  PORT MAP (
       clka => clock,
       douta => data,
       addra => rdaddress
```

```
    );

  -- Clock process definitions
  clock_process :process
  begin
    clock <= '0';
    wait for clock_period/2;
    clock <= '1';
    wait for clock_period/2;
  end process;

  -- Stimulus process
  stim_proc: process
  begin
    for i in 0 to 65535 loop
      rdaddress <= std_logic_vector(to_unsigned(i, 16));
    wait for 20 ns;
    end loop;
    wait;
  end process;

END;
```

2.2 Overview of image gradient operation

Grayscale 8-bit image is a 2-D matrix storing pixel values between 0-255, as shown in figure1 below. Size of the image will be 256x256. Image should be stored in 1-D array format in block RAM, that from address 0 (0000_{16}) in row major format.

Objective is to perform image operation to extract the vertical edges in the given image. To get the edges in the image, the gradient of the image needs to be calculated. A 3x1 filter (shown in red in Figure 3) is used to calculate the gradient of the image.

2.2.1 Steps to calculate the gradient

- Filter stride along the row of the image to calculate the resultant pixels in the output image.
- Output pixel value is summation of element-wise multiplication between filter value and input image pixel as shown in the given equation. Input image pixel at location (i,j) is denoted by *I(i, j)*, output pixel as *O(i, j)*.

$$O(i,j) = 1 * I(i,j-1) - 2 * I(i,j) + 1 * I(i,j+1)$$

When the column index is j < 0 or j > 255, then assume the pixel value as 0, this will be applicable while calculating the border pixel value in the output image. Few illustrations:

– *Case I:*

$$O(0, 0) = 1 * I(0, -1) - 2 * I(0, 0) + 1 * I(0, 1)$$
$$O(0, 0) = 0 - 20 + 10 = -10.$$

– *Case II:*

$$O(0, 1) = 1 * I(0, 0) - 2 * I(0, 1) + 1 * I(0, 2)$$
$$O(0, 0) = 10 - 20 + 10 = 0$$

– *Case III:*

$$O(0, 2) = 1 * I(0, 1) - 2 * I(0, 2) + 1 * I(0, 3)$$
$$O(0, 0) = 10 - 20 + 220 = 210$$

In the final output image, negative pixel values shall be clamped to zero and any pixel value greater than 255 clamped to 255. This is done to ensure the output image is stored in 8-bit unsigned format.

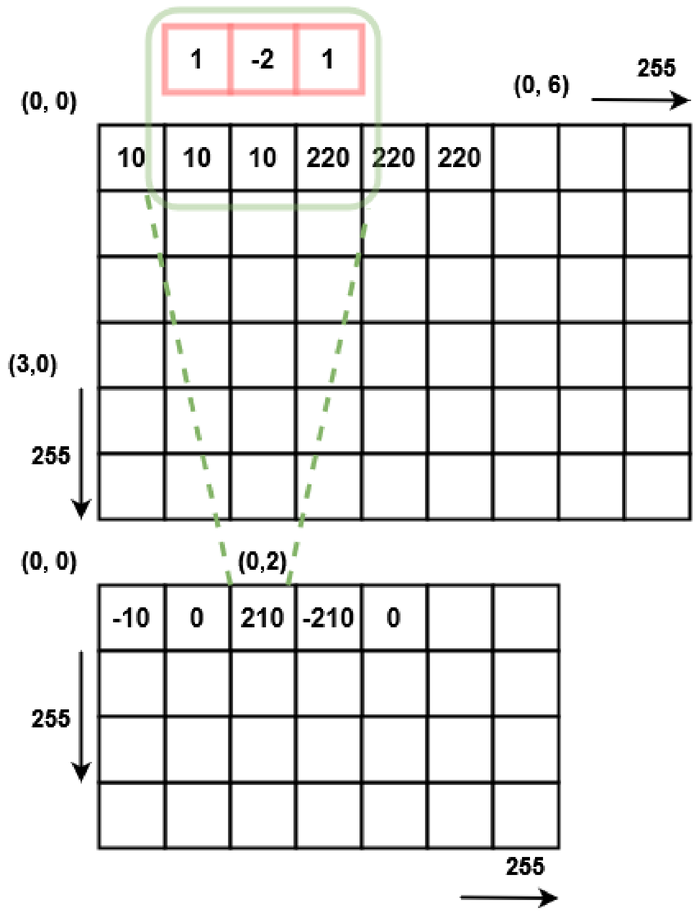


Figure 3: Illustration to perform 1D operation on grayscale image

2.2.2 Image gradient VHDL module

Task is to design a module which will read the images from the block RAM and calculate the image gradient as described in above section 2. The resultant image should be stored into new block RAM and displayed on the VGA monitor. All the components are described in Figure 4.

To calculate the output pixel, three input pixel values are required. But in one clock cycle you can read only one address from block RAM. Naive way is to read each input pixel in a 3 clock cycle and calculate the output. Another way is to use a temporary small storage element (Registers) to store all three pixel values and thus avoid extra clock cycles, this is shown in Figure 4 (highlighted in red box). **It is not compulsory to follow the above method, you are free to design your own module.**

- A Read-Only Memory (ROM) - You will be using this memory to store the input image.
- Registers - You will be using these to store temporary results across clock cycles.
- A Read-Write Memory (RWM, more popularly known as RAM or Random-Access Memory) - You will be using this memory to store output images.
- Adder - to perform summation of three resultant pixel values.
- VGA display controller: To drive the VGA port on basys 3 board.

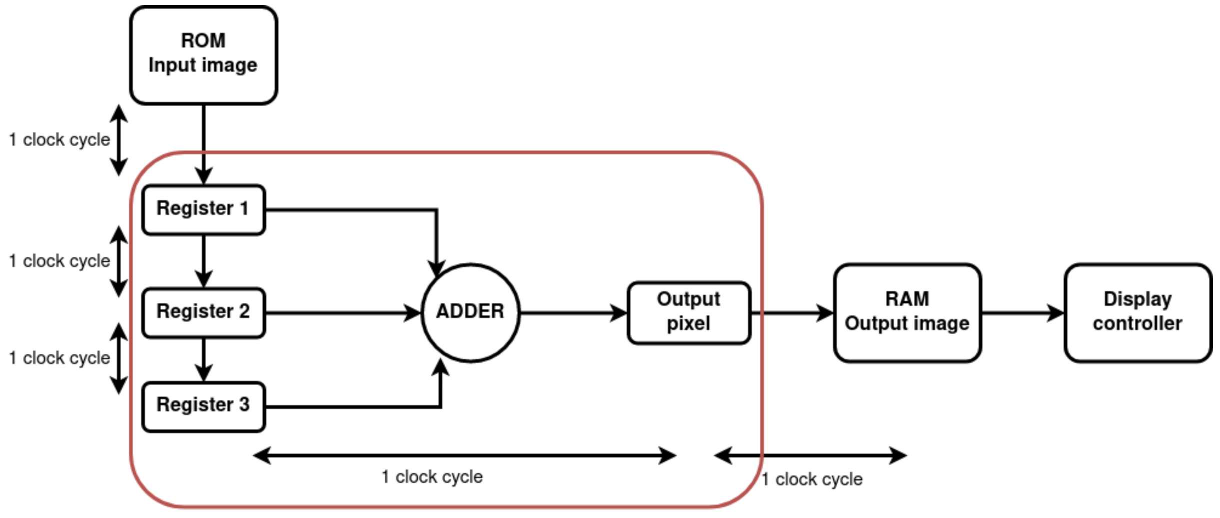


Figure 4: Block diagram component-wise

2.2.3 Memory Settings :

The design requires memories to store the input and the output image. We will store the input image in a read-only memory (ROM) and store the output image in random-access memory (RAM). The memory will be used to store the following:

- Input Image: 256x256 image with each pixel of 8 bits or 1B. For storing the input we need 65536 words, 8 bits wide. These will be stored at address 0 (0000_{16}) onwards.
- Output Image: 256x256 image with each pixel of 8 bits or 1B. For storing the output we need 65536 words, 8 bits wide. These will be stored at address 0 (0000_{16}) onwards.

You need to create testbench for this module as given in section 2.1

2.3 Setting up VGA display on Basys 3 board

In the current assignment, you will have to display the output image on a monitor. You will be designing a display controller module in VHDL, to drive externally connected VGA displays.

VGA is an acronym for video graphics array, connector has 15 pins, three lines for RGB, 2 for sync signals and rest are ground pins to regulate the current. Refer to Basys3 reference manual, section 7.1 for pin configuration (*Basys3 manual :* *https://digilent.com/reference/_media/basys3:basys3_rm.pdf*).

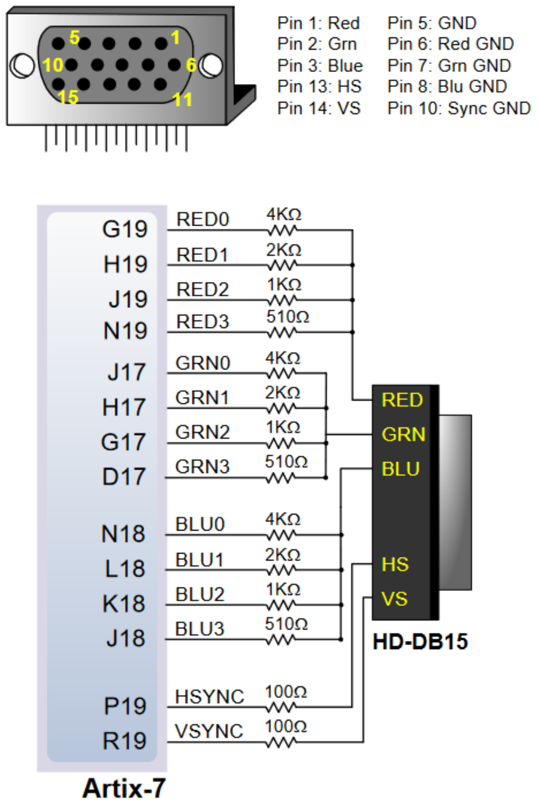
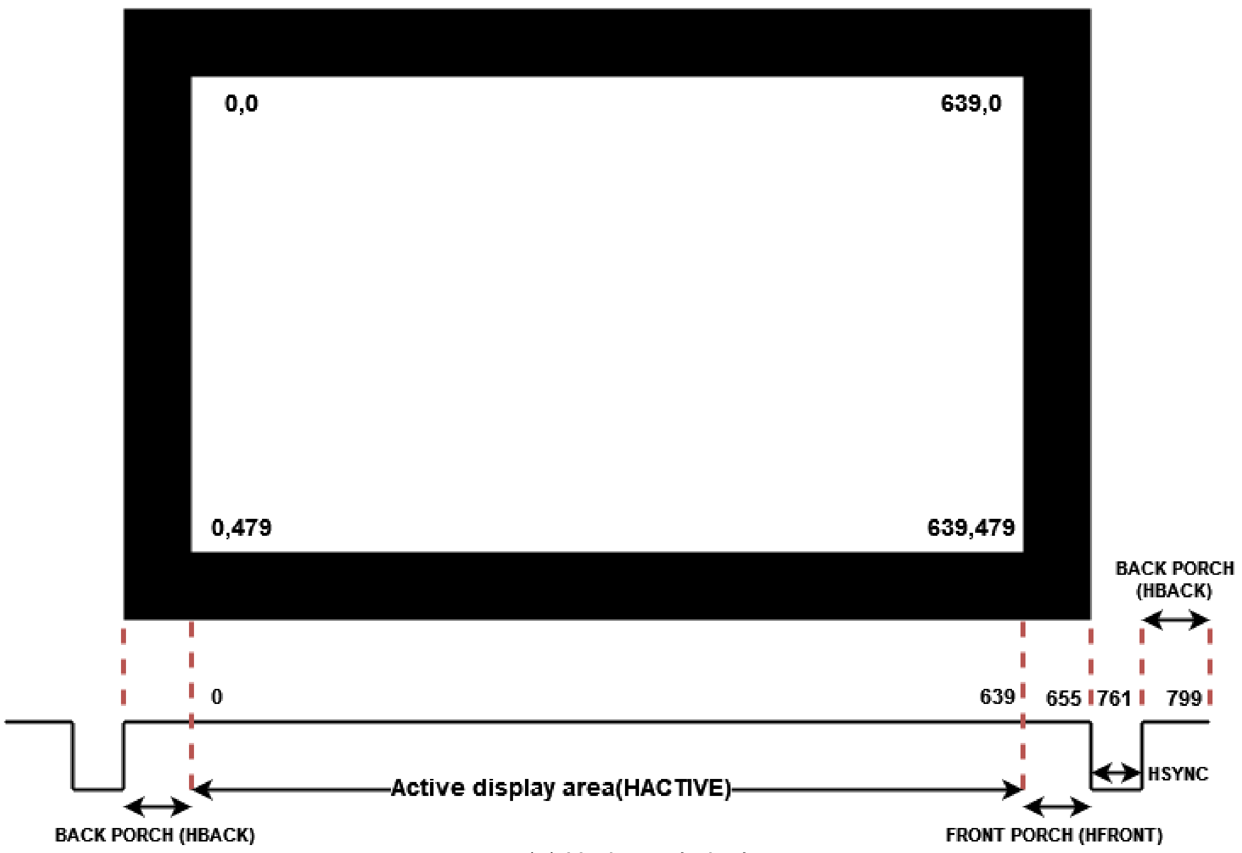
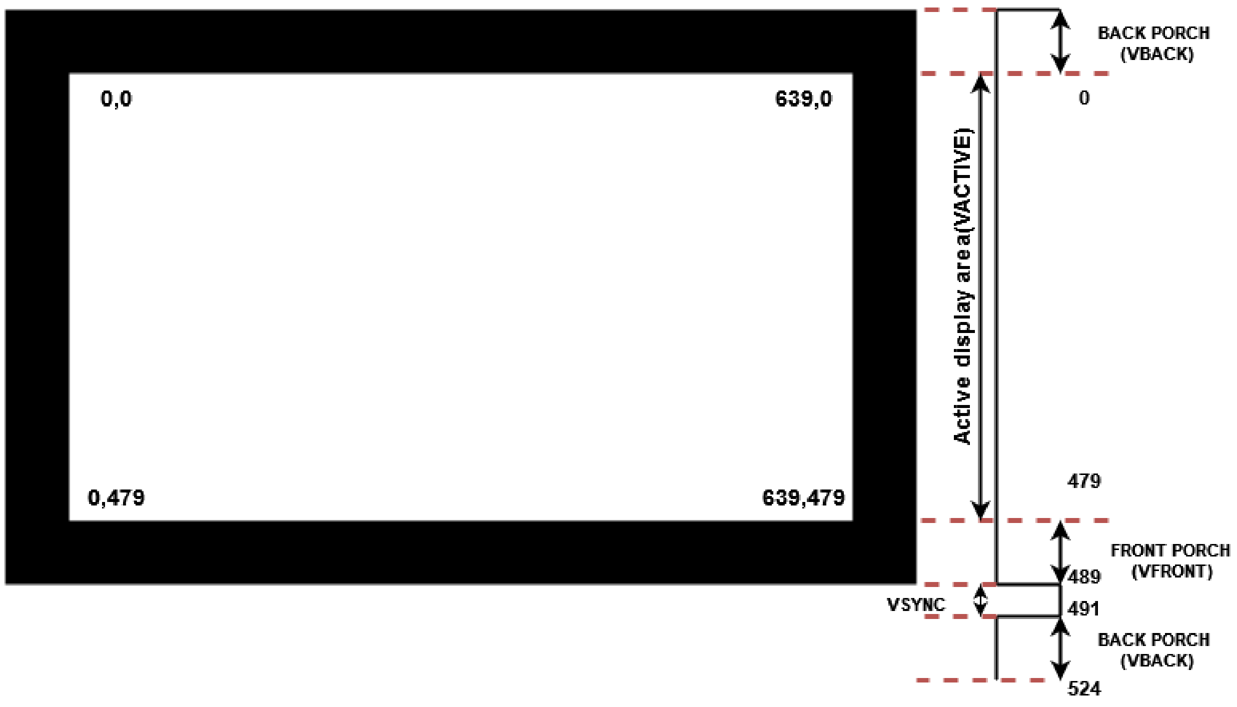


Figure 5: Pin connector for VGA basys 3 board

Each display has certain timings, as shown below in Figure 6. Timings are separated into horizontal and vertical segments. It consists of active video line, front and back porch and sync signals. To understand how the working of old CRT displays work, refer to the CRT link: https://learn.digilentinc.com/Documents/269 . Even though the devices have been upgraded, the timing specification remains the same. Video timing has an active area where the actual video/image data is seen and blank timings in which the internal circuit (electron beam in CRT) needs to be reset.



(a) Horizontal pixels



(b) Vertical lines

Figure 6: Display timings diagram

There are various resolution and timing supported via display, but for now only one display timing **(640*480@60Hz)** has to be implemented which is defined in the Table 1 given below.

Table 1: Timing specification for 640x480@60Hz

Parameter	Value
Pixel clock	25 MHz
HSYNC	96 pixels
HBACK	48 pixels
HFRONT	16 pixels
HACTIVE	640 pixels
VSYNC	2 lines
VBACK	33 lines
VFRONT	10 lines
VACTIVE	480 lines

Total number of pixels in one horizontal line (active+blanking) will be :

$$HTOT = HACTIVE + HBACK + HFRONT + HSYNC$$

$$800 = 640 + 48 + 16 + 96$$

Total number of lines in one frame will be :

$$VTOT = VACTIVE + VBACK + VFRONT + VSYNC$$

$$525 = 480 + 33 + 10 + 2$$

Total number of pixels in one frame are, **420000** which has to be displayed in a fixed amount of time that is **16.66 ms** (60 Hz or 60 frames per seconds). Thus, time period for the clock for one pixel will be (25.2 Mhz):

$$420000 * 60 = 25200000$$

Display controllers need an input clock of 25 MHz and generate corresponding HSYNC and VSYNC signals.

To accomplish this, three modules need to be made: (i) clock divider, (ii) Horizontal pixel counter and (iii) Vertical line counter. A block diagram of the modules with input and output signals is given in the Figure 7

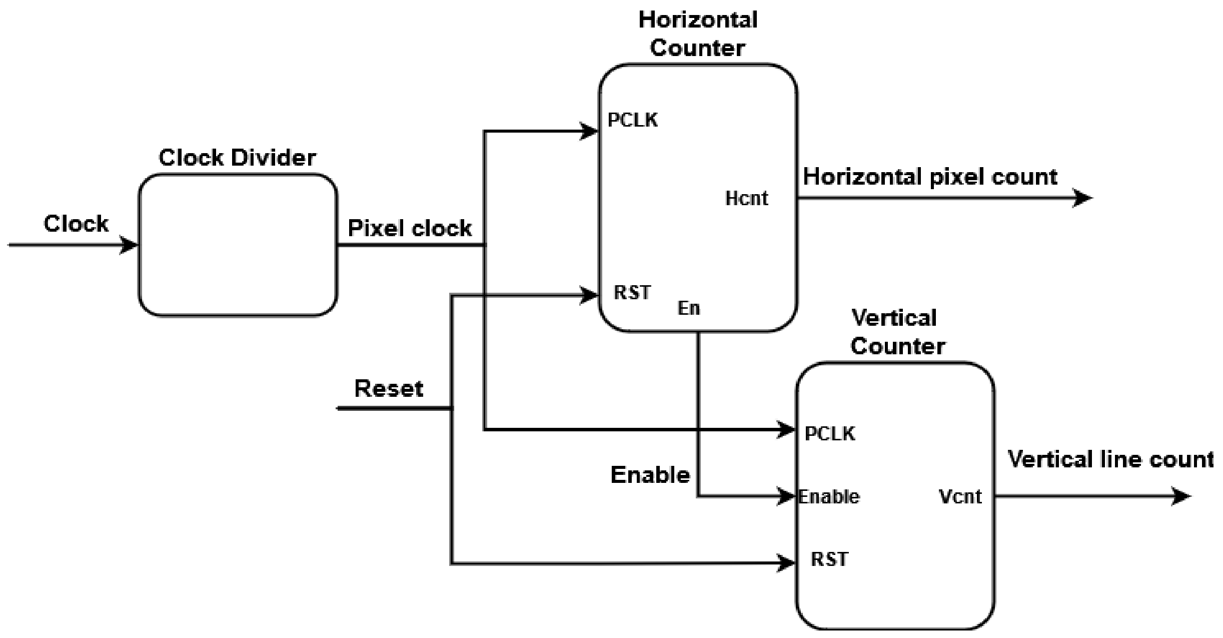


Figure 7: Module diagram for display controller

2.3.1 Clock divider

The clock divider will generate the 25 MHz pixel clock using the onboard board clock of 100 MHz. Essentially, the board clock needs to be divided to get the required pixel clock.

2.3.2 Horizontal pixel counter

The horizontal pixel counter will keep track of pixels in the line, input will be pixel clock and reset signal. Output will be pixel count (Hcnt) and line complete signal (indicating end of the line, which will serve as an enable signal to the vertical counter). Counter will start from 0 and increment till HTOT-1 then reset the count to 0.

2.3.3 Vertical line counter

The vertical line counter will keep track of lines in a frame, input will be pixel clock, reset and enable signal, Output will be line count (Vcnt). Counter will start from 0 and increment till VTOT-1 then reset the count to 0.

2.3.4 Display controller logic

Based on the **Hcnt**, **HSYNC** signal will be driven **HIGH** for the pixel count 0 to **HACTIVE** + **HFRONT** and changed to LOW for duration **HYSNC**. Similarly, based on **Vcnt**, **VSYNC** will be driven **HIGH** from line **0** to **VACTIVE** + **VFRONT** and **LOW** for **VSYNC** duration. The signal transition is shown in the figure.

Additionally, image data need to be sent during **HACTIVE** and **VACTIVE** period. Image pixel location will be calculated based on **Hcnt** and **Vcnt** signal from the counter. For example, if the image size is 256x256 then it can be displayed at any location in the frame 640x480 as shown in the Figure 8.

The given image is an 8-bit grayscale format, meaning the image contains 256 levels of grayscale colours. First, to display grayscale image using VGA RGB format, each colour line of RGB should be set to the same value that is R=G=B. But basys3 has 4-bit width per colour line, so image 8-bit value should be truncated to 4-bit value via discarding lower 4 bits (i.e. using 4 MSB bits). This will reduce the 256 colour level to 16 colour level. For example, if pixel at location (0,0) has 8-bit value **10110001** then video data in RGB format will be **1011 1011 1011** (12-bit RGB).

Above logic needs to be written in the main display controller module which will output HSYNC, VSYNC signals, video data signals.

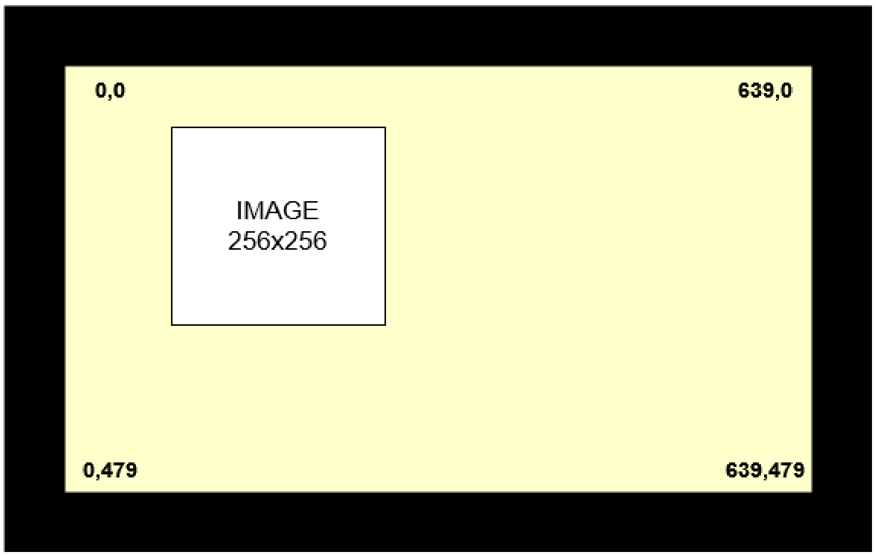


Figure 8: Positioning of image wrt display area

You need to create testbench for this module as given in section 2.1

3. Assignment Submission Instructions

General assignment instructions which need to be followed for each assignment. Only one partner needs to submit. Mention all team member names and entry IDs during the submission.

1. Name the submission file as entryNumber1_entryNumber2.zip

2. You are required to submit a zip file containing the following on Gradescope:

- VHDL files for all the designed modules.
- Constraint File (.xdc)
- Bit file (.bit)
- synthesis report (particularly resource counts: Flip-flops, LUTs, BRAMs, and DSPs)
- Simulated waveforms with test cases of each sub-component and complete design.
- A short report (1-2 pages) explaining your approach. Include block diagram depicting the modules