**VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY**

**UNIVERSITY OF INFORMATION TECHNOLOGY**

**FACULTY OF COMPUTER ENGINEERING**

**Full name: Đỗ Thanh Sơn Full name: Trương Hữu Trường Sơn**

**Student ID: 21522551 Student ID: 21522559**

**Class: CE213.O11.MTCL Class: CE213.O11.MTCL**

**Full name: Nguyễn Anh Quốc**

**Student ID: 21522526**

**Class: CE213.O11.MTCL**

**Image Processing Using Verilog**

**Manipulate RGB Values with Varied Filters**

Logo, company name

Description automatically generated with medium confidence

**MENTOR**

**MSc. Hồ Ngọc Diễm**

TABLE OF CONTENTS

[ABSTRACT: 3](#_Toc154386827)

[Chapter 1. Introduction 3](#_Toc154386828)

[Chapter 2. Theory Basis 4](#_Toc154386829)

[2.1 RGB 4](#_Toc154386830)

[2.2 Practical Implementation 4](#_Toc154386831)

[2.2.1 Median filter 4](#_Toc154386832)

[2.2.2 Mean filter 6](#_Toc154386833)

[2.2 HSYNC, VSYNC 8](#_Toc154386834)

[Chapter 3. System Implementation 9](#_Toc154386835)

[3.1. MATLAB 9](#_Toc154386836)

[3.2 Verilog 10](#_Toc154386837)

[3.2.1) Image Read and Process 10](#_Toc154386838)

[3.2.2) FSM of VSYNC and HSYNC 15](#_Toc154386839)

[3.2.3) Image Write 16](#_Toc154386840)

[3.2.4) Testbench 18](#_Toc154386841)

[3.3 ModelSim 19](#_Toc154386842)

[Chapter 4. FPGA Implementation (Underdevelopment) 20](#_Toc154386843)

[Chapter 5. SUMMARY, RESTRICTIONS AND DEVELOPMENT ORIENTATIONS 22](#_Toc154386844)

[5.1 Summary 22](#_Toc154386845)

[5.2 Limitations 22](#_Toc154386846)

[5.3 Future Development Directions 22](#_Toc154386847)

# ABSTRACT:

This paper presents a unique approach to image processing through Verilog code, departing from the conventional use of Python OpenCV. Leveraging MATLAB, we convert an image into a hex file and integrate it into Verilog for processing operations. Utilizing Quartus II as the coding platform, the study focuses on three essential image operations: MEAN, MEDIAN, GRAYSCALE. The methodology encompasses reading an input bitmap image in Verilog, executing processing operations, and generating an output bitmap image. This innovative process offers insights into image processing within the Verilog framework, showcasing the potential for hardware description language applications in this domain.

# Chapter 1. Introduction

Image processing serves as a pivotal technique for enhancing or extracting pertinent information from images. This form of signal processing involves three fundamental steps:

1. Importing image hex files through an image acquisition tool.

2. Analyzing and manipulating the image.

3. Producing output, which may consist of an altered image or a report based on image analysis.

Leveraging the Verilog hardware description language enables direct coding at the hardware level, offering the advantage of hardware portability. Verilog facilitates the straightforward manipulation of hex files for image processing, providing hardware engineers with a robust platform for circuit synthesis. The inherent link between Verilog syntax and hardware structure, coupled with timing information, allows for specialized speed enhancements in a hypothetical hardware implementation.

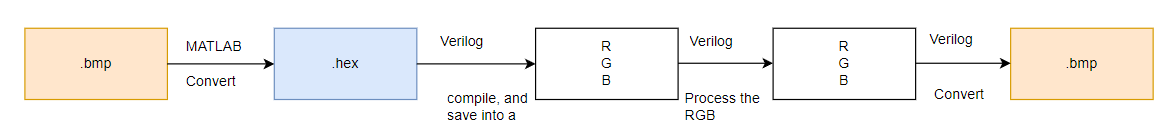


Figure . Image Processing Block Diagram

# Chapter 2. Theory Basis

## 2.1 RGB

The RGB color model is an additive color model in which the red, green and blue primary colors of light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

The main purpose of the RGB color model is for the sensing, representation, and display of images in electronic systems, such as televisions and computers, though it has also been used in conventional photography. Before the electronic age, the RGB color model already had a solid theory behind it, based in human perception of colors.

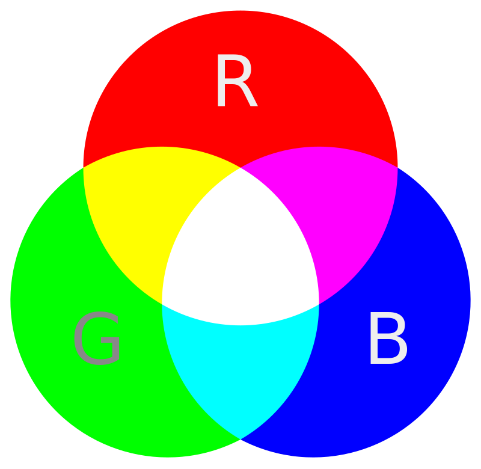


Figure . RGB

## Practical Implementation

### 2.2.1 Median filter

#### a) Definition

The median filter considers each pixel in the image in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. It replaces it with the median of those values. The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical order and then replacing the pixel being considered with the middle pixel value.

#### b) How It Works

Sort the values of pixels in the filter window in ascending order, then replace its value with the median of that sorted sequence.

A screenshot of a diagram

Description automatically generated

Figure . Median Filter Demonstration

#### c) Application

Detecting pixels that are "unusually different" and adjusting them to harmonize as closely as possible with the surrounding bits. Due to this characteristic, the median filter is often employed for images affected by salt-and-pepper noise. This serves as a crucial preprocessing step to enhance the accuracy of subsequent tasks with noisy images.

The drawback of this filtering method is its high computational cost and time consumption. Finding the median value requires sorting the array of values within the filter window, which is relatively slow.

#### d) Example

Below is an example of an image that has been processed using a median filter.

A person with a beard smiling

Description automatically generatedA person smiling for the camera

Description automatically generated

Figure . Before and After using Median Filter

### 2.2.2 Mean filter

#### a) Definition

Mean filtering is a simple, intuitive and easy to implement method of smooth-ing images, i.e. reducing the amount of intensity variation between one pixel and the next. It is often used to reduce noise in images.

#### b) How It Works

It operates by replacing the value at a pixel with the average of the values within its own filter window.

A math problem with numbers and arrows

Description automatically generated with medium confidence

Figure . Mean Filter Demonstration

#### c) Application

Mean filtering is most commonly used as a simple methpd for reducing noise in an image.

Disadvantages of mean filtering:

A single pixel with a very unrepresentative value can significantly affect the mean value of all the pixels in its neighborhood.

When the filter neighborhood straddles an edge, the filter will interpolate new values for pixels on the edge and so will blur that edge. This may be a problem if sharp edges are required in the output.

#### d) Example

Below is an example of an image that has been processed using a mean filter.

A person with a beard smiling

Description automatically generatedA person with a beard smiling

Description automatically generated

Figure . Before and After using Mean Filter

2.2.3 Grayscale filter

#### a) Definition

There are some image processing tasks that do not require consideration of color. In such cases, grayscale is used to convert the image to black and white, minimizing unnecessary processing work.

#### b) How it works

We need to convert the three R, G, B values at each pixel into a single value. There are various formulas to accomplish this; in this project, we use the formula: Y = 0.299R + 0.587G + 0.114B.

A screenshot of a math problem

Description automatically generated

Figure . Grayscal Filter Demonstration

#### c) Application

As mentioned in the definition section, the grayscale filter is a preprocessing step used to eliminate unnecessary colors from an image if they are not needed in subsequent processing steps. This helps minimize the workload, increasing image processing efficiency.

#### d) Example

Below is an example of an image that has been processed using grayscale filter.

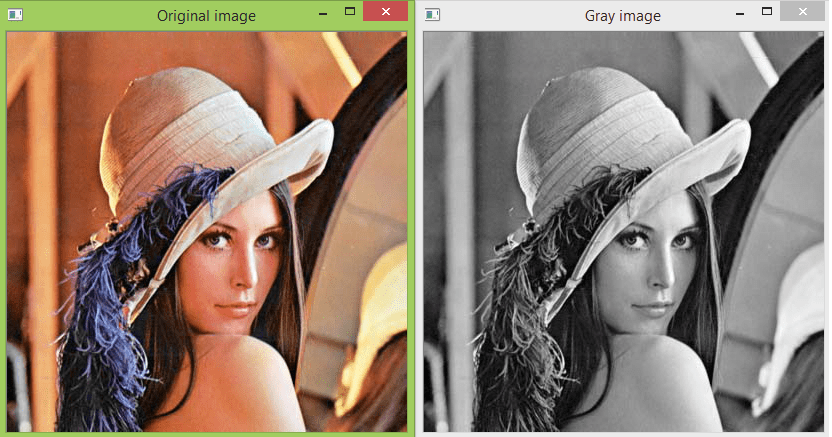
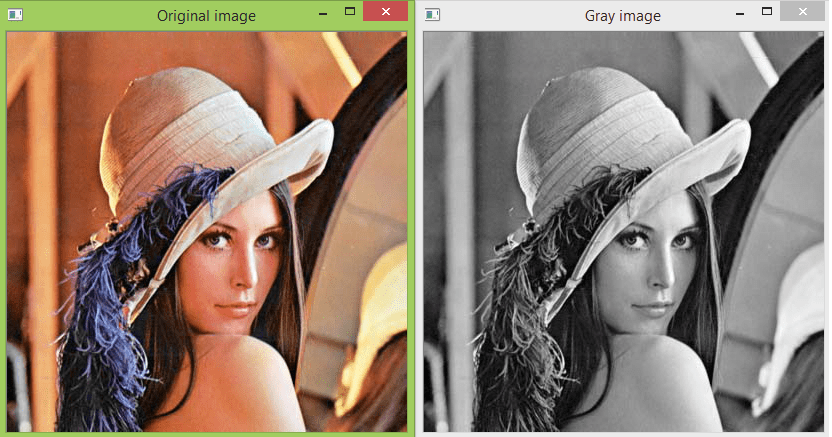


Figure . Before and After using Grayscale Filter

## HSYNC, VSYNC

A diagram of a diagram

Description automatically generated

Figure . HSYNC and VSYNC

HSYNC (horizontal sync) and VSYNC (vertical sync) are essential components of synchronization in a video system, ensuring stability and proper display on a 2D screen.

HSYNC maintains stability along the horizontal axis (each line) by defining the horizontal reference point, while VSYNC does the same for the vertical axis (each frame or field).

Together, they provide the necessary references for reconstructing the image on a display device.

CSYNC (composite sync) is a combined version of HSYNC and VSYNC, representing a unified signal used for transmission.

# Chapter 3. System Implementation

## 3.1. MATLAB

Images cannot be read directly by Verilog. The image must be transformed from bitmap to hexadecimalformat before it can be read in Verilog. To convert a bitmap image to a hex file, we use MATLAB the input image needs to be converted into 256x256 pixels, and the following files : input1.hex file contains the Original bitmap image, input2.hex file contains salt and pepper image, input3.hex file contains gaussian.

|  |
| --- |
| % Read your image (assuming b\_resized is your original image)  original\_image = imread('test\_img.bmp');  % Add salt and pepper noise  noisy\_image = imnoise(original\_image, 'salt & pepper', 0.02);  % Apply a mean filter  speckle\_noisy\_image = imnoise(original\_image, 'gaussian', 0, 0.01);  % Display the results  figure;  subplot(1, 3, 1), imshow(original\_image), title('Original Image');  subplot(1, 3, 2), imshow(noisy\_image), title('For Median filter');  subplot(1, 3, 3), imshow(speckle\_noisy\_image), title('For Mean filter');  % Initialize arrays to store pixel values  a = zeros(1, 3 \* 256 \* 256);  b = zeros(1, 3 \* 256 \* 256);  c = zeros(1, 3 \* 256 \* 256);  % Convert the image data to linear arrays (column-wise)  k = 1;  for i = 256:-1:1 % Image is written from the last row to the first row  for j = 1:256  a(k) = original\_image(i, j, 1);  a(k + 1) = original\_image(i, j, 2);  a(k + 2) = original\_image(i, j, 3);  b(k) = noisy\_image(i, j, 1);  b(k + 1) = noisy\_image(i, j, 2);  b(k + 2) = noisy\_image(i, j, 3);  c(k) = speckle\_noisy\_image(i, j, 1);  c(k + 1) = speckle\_noisy\_image(i, j, 2);  c(k + 2) = speckle\_noisy\_image(i, j, 3);  k = k + 3;  end  end  % Open text files for writing in hexadecimal format  fid1 = fopen('input1.hex', 'wt');  fid2 = fopen('input2.hex', 'wt');  fid3 = fopen('input3.hex', 'wt');  % Write the pixel values to the text files in hexadecimal format  fprintf(fid1, '%02X\n', a);  fprintf(fid2, '%02X\n', b);  fprintf(fid3, '%02X\n', c);  % Close the files  fclose(fid1);  fclose(fid2);  fclose(fid3);  disp('Text files write done'); |

Figure 10. Matlab Hex code file.

The image is named “test\_img.bmp” as input for MATLAB code to identify it in the same folder. The image is send through some iterations in for loop. Notice the loop is from 256 to 256 same as the pixels of the image. In the iterations the images RGB data is extracted and files name input1.hex, input2.hex, input3.hex are given out by using the commad fopen. The input hex file only contains RGB vectors for each pixel of the input image. The next section covers the Verilog part of the research.

## 3.2 Verilog

### 3.2.1) Image Read and Process

#### a) Reading Image

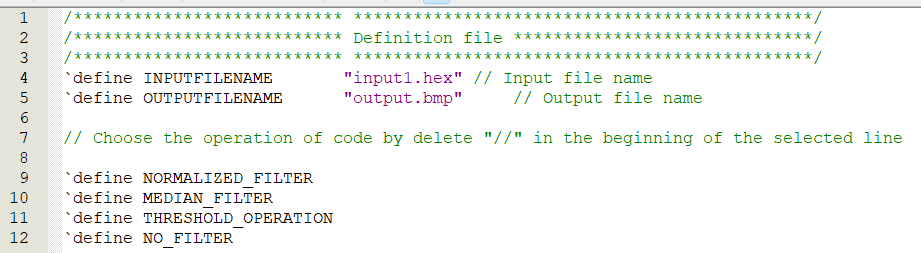


Figure 11. Parameters file from where operations will be defined

Image processing operations are now implemented in Verilog. The Verilog coding is divided into two parts, first is implementation where parameters for operations such as MEAN, MEDIAN, GRAYSCALE are declared. The second is simulation part where the above operations are defined and coded in details. We will be performing three types of operations on the image.

A screenshot of a computer program

Description automatically generated

A computer screen shot of text

Description automatically generated

Figure 12. Image Read Module.

In Verilog, the $readmemh command is employed to extract hexadecimal data from an image file “.hex “. Following this read operation, the RGB picture data is retained in memory, ready for subsequent processing. The module encapsulates variables to store the image's width and height, essential parameters for further manipulation. A dedicated array is established to house the RGB data after undergoing a sequence of operations. Notably, for each color, dual arrays are instantiated—one for even data and another for odd data. The choice of image processing operation resides in the "parameter.v" file, offering flexibility through a simple comment line switch. This modular approach facilitates seamless adjustment of processing operations, enhancing the adaptability and functionality of the Verilog image read module.

#### b) Image Process

A computer code with numbers and symbols

Description automatically generated with medium confidence

Figure 13. Grayscale operation Verilog code.

The threshold operation is performed by taking a weighted sum of the RGB components of each pixel and assigning the result to all three color components. This process effectively converts a color image to a grayscale image based on the specified weights for each color channel.

A screenshot of a computer program

Description automatically generated

Figure 14. Normalized Filter Verilog code.

The normalized filter operation smoothens the image by replacing each pixel's RGB values with the average RGB values of itself and its eight neighboring pixels. The division by 9 in the calculation is for normalization, ensuring that the resulting values are averaged over a consistent scale.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Figure 15. Median Filter Verilog code.

The median filter operation replaces each pixel's RGB values with the median values of itself and its eight neighboring pixels. This helps to reduce noise and preserve edges in the image. The sorting process ensures that the median value is selected from the sorted neighborhood values.

A screenshot of a computer program

Description automatically generated

Figure 16. Median Filter Verilog code.

When NO\_FILTER is defined, the code directly passes through the RGB values of each pixel without modification, effectively keeping the image unchanged.

### 3.2.2) FSM of VSYNC and HSYNC

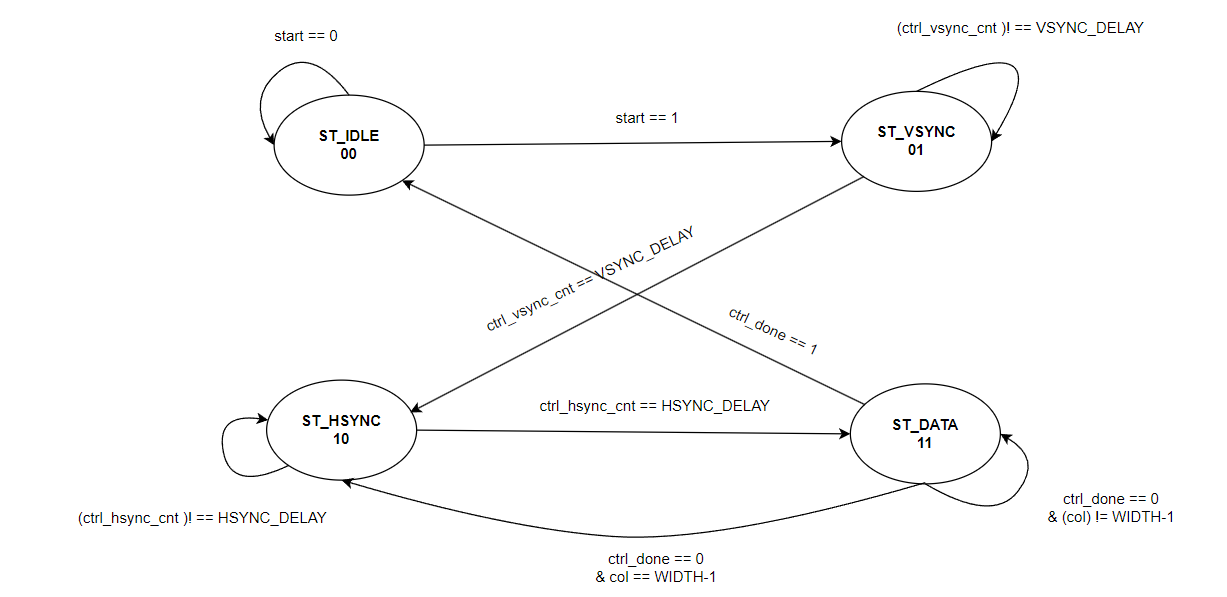


Figure 17. FSM Diagram

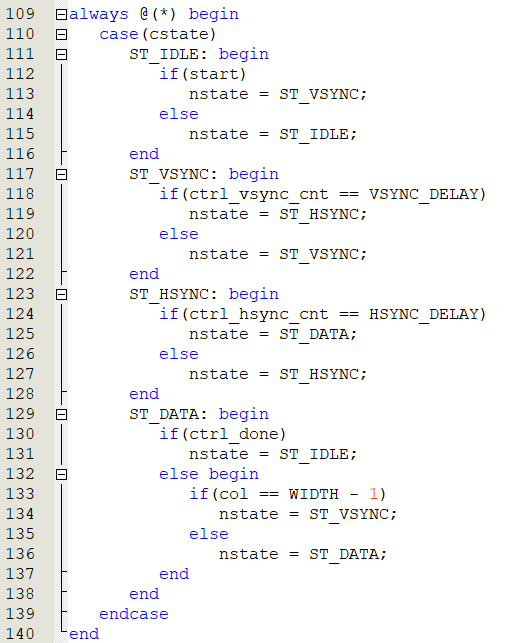


Figure 18. FSM Verilog code.

Within this Verilog FSM implementation, a crucial aspect involves temporarily simulating the Vertical Sync (VSYNC) and Horizontal Sync (HSYNC) pulses, pivotal components in video and image processing. In the VSYNC state (ST\_VSYNC), a condition checks if the vertical sync pulse delay (VSYNC\_DELAY) has been satisfied. Once this condition is met, the FSM transitions to the HSYNC state (ST\_HSYNC), simulating the vertical synchronization process.

Subsequently, in the HSYNC state, a similar mechanism is employed to introduce a delay (HSYNC\_DELAY) that mimics the horizontal sync pulse. Upon completion of this delay, the FSM transitions to the DATA state (ST\_DATA), indicating the readiness to handle and process pixel data.

### 3.2.3) Image Write

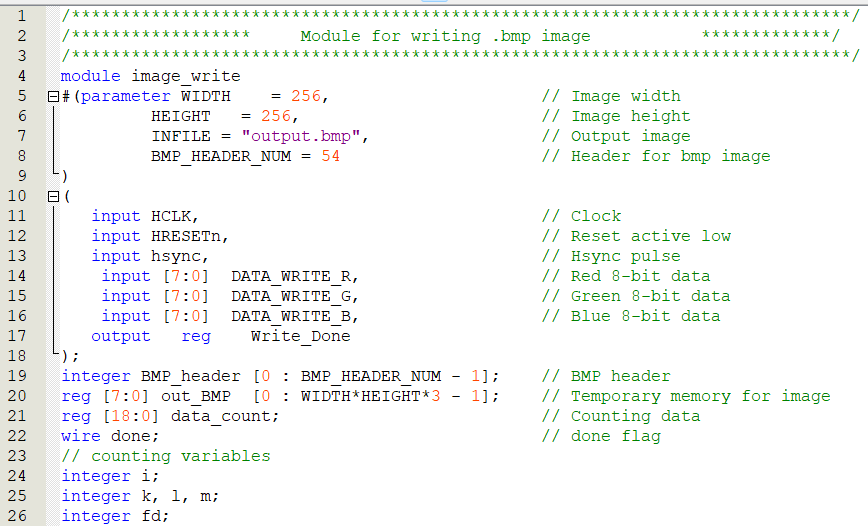


Figure 19. Image Read Module.

The "parameter.v" file, previously alluded to, serves a pivotal role in the image processing module by defining essential parameters such as input and output file directories and names. These specifications are instrumental in guiding the system to the correct data sources and destinations. Post-processing, it becomes imperative to validate the results, and to facilitate this, the processed image data is meticulously written to an output image file.

A screenshot of a computer

Description automatically generated

Figure 19. Initialie BMP Header File

The Verilog code features a crucial 54-byte header designed for bitmap images, providing essential information for proper image interpretation. Specifically tailored for 256x256-pixel images, this header ensures accurate rendering. Without it, the displayed picture may be compromised, highlighting the significance of this configuration in guaranteeing appropriate image presentation.

### 3.2.4) Testbench

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

Figure 20. Testbench Verilog code

This testbench simulates the processing of an image file through the image\_read module, encoding it, and then writing the processed image data to a BMP file using the image\_write module.

## 3.3 ModelSim

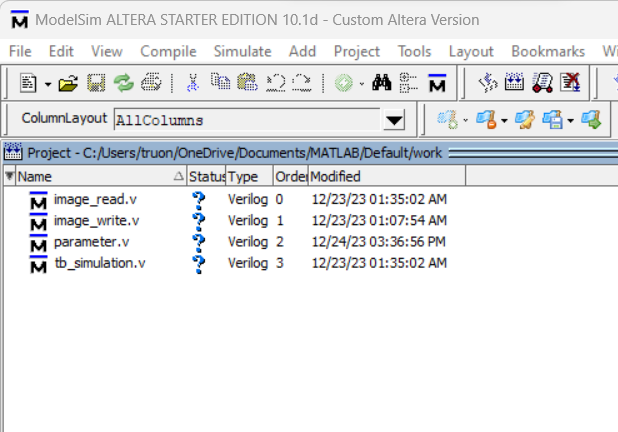


Figure 20. Compiling 4 different files

A screenshot of a computer

Description automatically generated

Figure 21. Run Simulation and see the results

Link google drive:

# Chapter 4. FPGA Implementation (Underdevelopment)

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

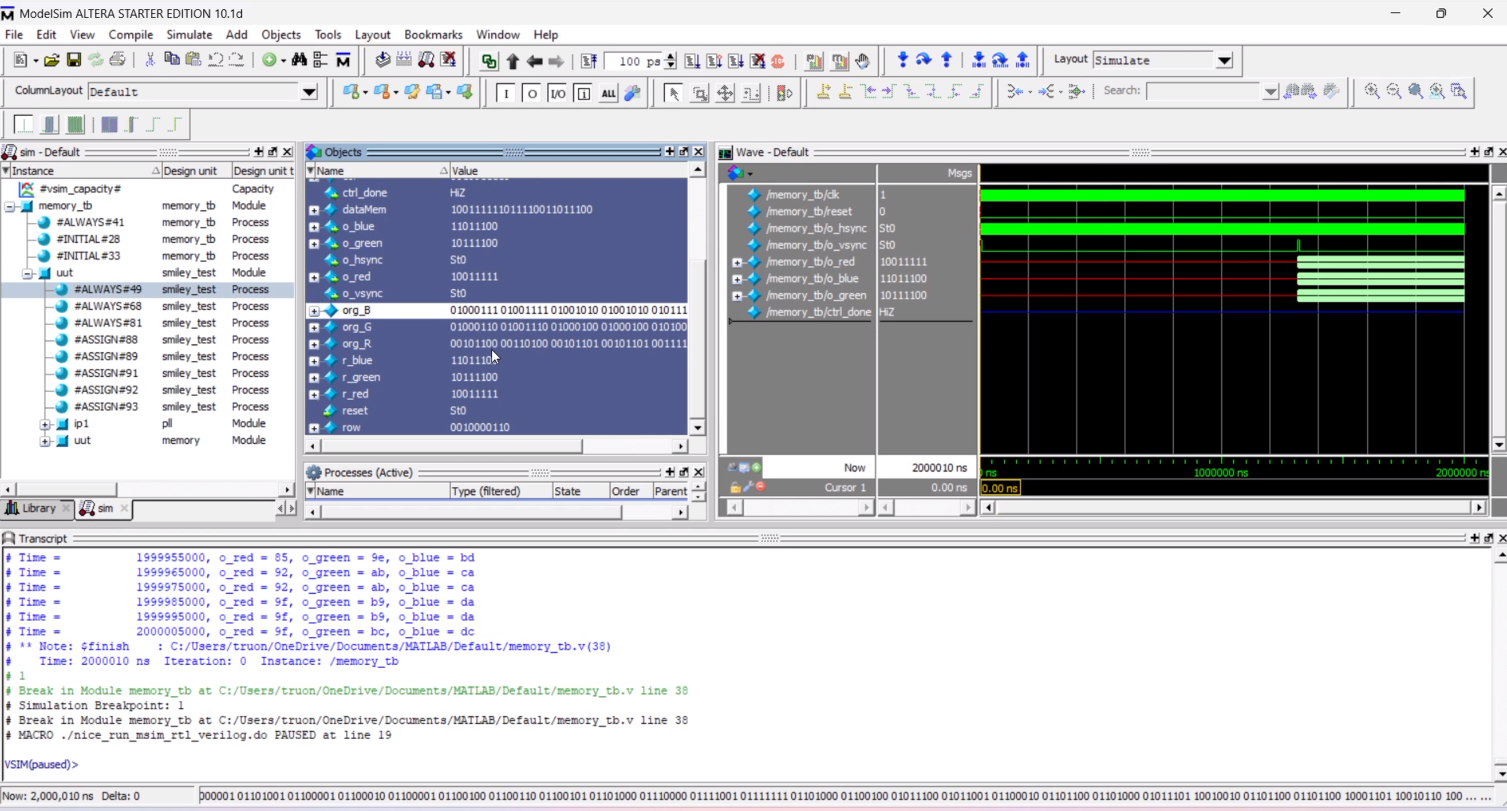
Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated



# Chapter 5. SUMMARY, RESTRICTIONS AND DEVELOPMENT ORIENTATIONS

## 5.1 Summary

The VGA display simulation code has demonstrated flawless performance in providing a realistic representation. The current implementation excels in accurately simulating VGA display functionalities.

## 5.2 Limitations

However, certain constraints exist within the current framework. Notably, the code is confined to execution exclusively on ModelSim, limiting its versatility. Additionally, the simulation accommodates only one filter at a time, imposing a restriction on the complexity of concurrent operations.

## 5.3 Future Development Directions

To enhance the applicability and practicality of the VGA display simulation, future development efforts should focus on creating a synthesizable module. This would enable seamless integration with real FPGA boards, facilitating a tangible implementation of the simulated environment. By transcending the limitations of ModelSim and accommodating multiple filters concurrently, the system can be optimized for a broader range of applications. This strategic expansion would contribute to the evolution of the project into a more versatile and scalable platform.