|  |
| --- |
| Image processing using verilog  By Omar Elhasan & Faris Ayyash |
| Instructor: Dr. Awos kanan |



Graphical user interface

Description automatically generated with medium confidence

Abstract  
The purpose of this project is to write a Verilog code that is capable of taking an image and filtering it, changing its pixel properties, and allowing us to apply a number of different effects to said image for this project is capable of applying up to eight different filters changing a base image in a multitude of ways

Table of Contents

[1 Introduction **2.**](#_Toc500411309)

[2 Design **2.**](#_Toc500411313)

[3 Design Analysis **3.**](#_Toc500411314)

[3.1 Test run and discussion **3.**](#_Toc500411315)

[3.2 Performance Analysis **5.**](#_Toc500411316)

[4 Conclusions **5.**](#_Toc500411320)

[5 References **5.**](#_Toc500411321)

Introduction  
Pipeline methodology where a process is divided into multiple smaller steps with the end goal of enhancing productivity and decrease the time needed for the processes to be complete.  
  
This project is based on pipeline methodology, where it is segmented into three major stages which are, stage one Fetch/Decoding; where instructions are fetched and decoded, stage two filter/processing where one of the eight filters is applied to the data, lastly stage three Output in where the filtered data is ready to be reprocessed into an image.

Design

Stage one (Read in input pixel values): - pixels from the Pixel\_input which is a twenty-four-bit buffer are assigned to Tblue, Tred and Tgreen based on their location where the first eight bits are assigned to Tred, the following eight are assigned to Tgreen and the last eight are assigned to Tblue

Stage two (Apply selected Filter): - One of the eight filters would be applied to the data; the user can choose on of the following eight filters  
  
Filter One (No Modification): - This filter does no modify anything but simply process the base RGB values.  
  
Filter two (Increase Brightness): - this filter would increase the brightness according to the value (val) entered by the user, it also insures that the RGB values do no exceed 255 and then returns the values to their Hex values by dividing by 16

Filter three (decrease Brightness): - this filter would decrease the brightness according to the value (val) entered by the user, it also insures that the RGB values do no exceed 255 and then returns the values to their Hex values by dividing by 16

Filter four (Grayscale): - This filter is used to give an image a monochrome look by combining the red, blue and green values then dividing by 3 changing their colors to shades of grey.  
  
Filter Five (Color inversion): - This filter is used to change the RGB values to their opposite counterparts on the color wheel by subtracting 0xff from the input values (Tred, Tblue and Tgreen).  
  
Filter six (Red Filter): - This filter subtracts the value(val) from Tred and then returns all RGB values to their Hex form.  
  
Filter six (Blue Filter): - This filter subtracts the value(val) from Tblue and then returns all RGB values to their Hex form.  
  
Filter six (Green Filter): - This filter subtracts the value(val) from Tgreen and then returns all RGB values to their Hex form.

Stage three: - this is the final stage where the Red\_o, Blue\_o and green\_o values are assigned to red, blue and green.

Design Analysis

During implementation of the pipeline some issues were faced, initially a decision was made to implement a buffer the contains all the RGB data before dividing them into Tred, Tblue and Tgreen, In stage two a switch system had to be implemented to allow the user to choose between the eight filters this design choice was limiting when creating the software implementation effectively forcing the use of a switch system in the software as well.

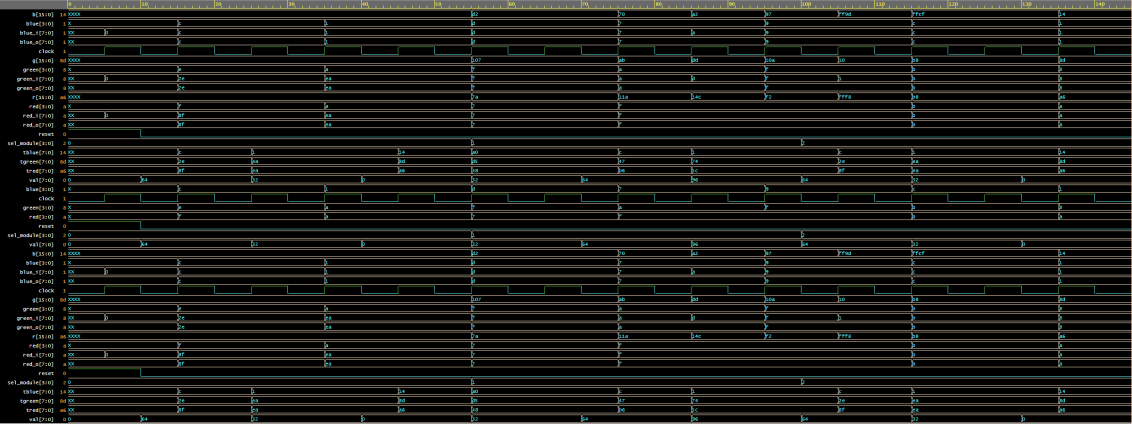
Test runs and Discussion   
In Figure 1, we use a small amount of data, and it took the pipelined version of the code 135 seconds to apply two filters to it.  


Figure 1 With two filters 135 seconds

In Figure 2, we use a mediocre amount of data, and it took the pipelined version of the code 205 seconds to apply four filters to it.

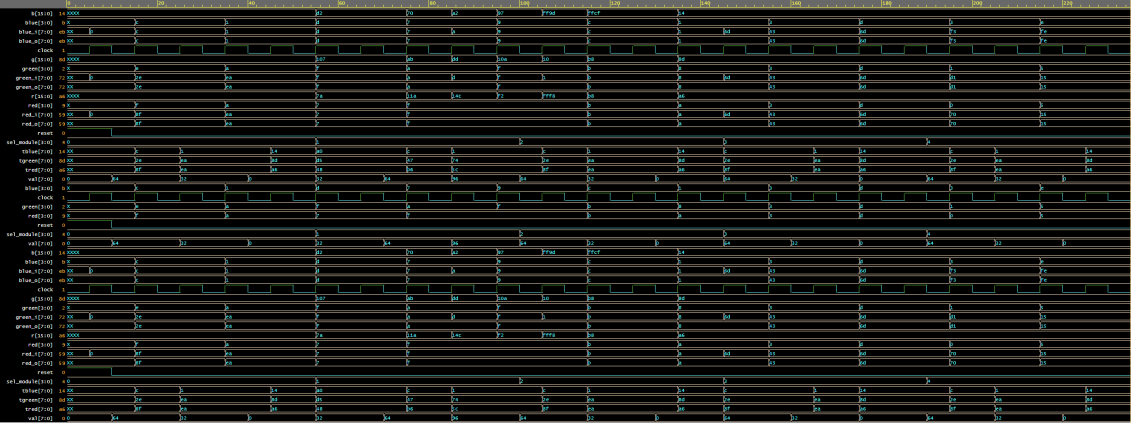


Figure 2 With four filters 205 seconds

In Figure 3, we use a large amount of data, and it took the pipelined version of the code 355 seconds to apply eight filters to it.  
A picture containing outdoor

Description automatically generated

Figure 3 With eight Filters 355 seconds

Graphical user interface, text, application, email

Description automatically generatedText

Description automatically generated

Figure 4 Verlog Implementation Figure 5 pipelined Software Implementation using c++

Performance Analysis

While Comparing between the pipelined and none-pipelined versions of this project, it was apparent that the pipelined version was much more effective, without a pipeline the first figures looked fairly similar, however when moving to figure three it was evident that the pipelined version took much less to complete, without a pipeline it took approximately 550 seconds to apply to all eight filters while the pipelined version only took 355 seconds giving us an apparent speedup with a ratio of 1:2  
  
Conclusion

In conclusion, using the data provided it is apparent to conclude that the pipelined version of this program is much faster than the none-pipelined version. By using the software (c++) implementation the same results were reproduced,  
  
References

- Computer Organization & Design RISC-V Edition: The Hardware/Software Interface, 2nd Ed. David Patterson & John Hennessy, Morgan Kaufmann, 2020

- https://github.com/Gowtham1729/Image Processing/tree/master/Final%20Project