FUNDAMENTAL OF DIGITAL SYSTEM FINAL PROJECT REPORT
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITAS INDONESIA

Image Grayscaller

GROUP A7

ARKA BRIAN DEWARA	**2106731421**
AZZAH AZKIYAH ANGELI	**2106731390**
HANDANESWARI P.	**2106731346**
MIRANTI ANGGUNSARI	**2106731472**

Kata Pengantar

Puji dan syukur kami panjatkan kehadirat Allah SWT atas segala rahmat dan karunia-Nya sehingga kami dapat menyelesaikan laporan proyek akhir Praktikum. Tidak lupa kami ucapkan terima kasih kepada para asisten Laboratorium Digital yang membantu kami dalam mengerjakan proyek akhir ini, terutama kepada bang Raihan Azhari sebagai pendamping kami dalam mengerjakan proyek Perancangan Sistem Digital yang berjudul "VHDL Implementation with Image Compressor".

Laporan ini berisi tentang hasil pengimplementasian VHDL pada alat yang kami buat yaitu Image Compressor. Selain itu, laporan ini juga mencakup cara kerja, pengujian, dan analisis selama percobaan pengimplementasiannya.

Perlu disadari bahwa laporan proyek akhir ini masih jauh dari sempurna. Oleh karena itu, kritik dan saran dari berbagai pihak sangat kami harapkan agar laporan ini dapat menjadi lebih baik.

Terakhir, kami berharap laporan proyek akhir yang kami buat dapat bermanfaat bagi banyak orang, serta juga bisa menjadi referensi yang berguna bagi kemajuan teknologi di masa mendatang.

Jakarta , 10 Desember 2022

Kelompok A7

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Berkembangnya teknologi seiring dengan meningkatnya kebutuhan manusia secara dinamis mampu dirasakan dengan pasti dan tak dapat dihindari. Sejak awal kemunculannya, teknologi dirancang untuk mempermudah pekerjaan manusia. Banyaknya inovasi dan perkembangan dalam bidang teknologi, seperti dalam bidang komunikasi, transportasi, kesehatan, pertahanan, dan lain-lain menjadikan teknologi sebagai bagian integral dari kehidupan kita sehari-hari, memudahkan pekerjaan dan meningkatkan produktivitas.

Salah satu contoh teknologi yang digunakan secara masif pada beberapa bidang profesional adalah *image processing. Image processing* adalah jenis teknologi yang melibatkan manipulasi dan peningkatan gambar digital untuk meningkatkan kualitasnya atau untuk mengekstrak informasi yang berguna darinya. Teknologi ini melibatkan berbagai algoritma dan teknik, seperti pemfilteran gambar, deteksi tepi, dan pengenalan pola, untuk memproses dan menganalisa gambar. Image processing digunakan dalam berbagai aplikasi, termasuk pencitraan medis, keamanan dan pengawasan, robotika, dan visi komputer. Selain itu juga dapat digunakan untuk mendeteksi kelainan pada gambar medis, mengidentifikasi objek dan wajah dalam rekaman keamanan, dan membantu robot menavigasi dan berinteraksi dengan lingkungannya.

Dalam pengolahannya, *image processing* yang dilakukan tanpa *grayscaller* akan memiliki keterbatasan dalam kemampuannya menganalisis dan memanipulasi gambar secara akurat. Konversi *grayscale* membantu dalam pemrosesan gambar dengan mengurangi jumlah data dan kompleksitas dalam suatu gambar. Reduksi kompleksitas yang dilakukan akan mensimplifikasi algoritma yang berkaitan dengan kebutuhan komputasi sehingga daoat meningkatkan visualisasi. Hal ini mempermudah penerapan berbagai teknik pemrosesan gambar, seperti deteksi tepi, thresholding, dan pengurangan noise pada gambar. Selain itu, *grayscale image* lebih cocok untuk analisis dan interpretasi karena memberikan representasi data gambar yang lebih intuitif.

Mengacu pada penjelasan diatas, kelompok kami memutuskan untuk mendeskripsikan sebuah *image grayscaler* dalam VHDL untuk dapat memecahkan permasalahan yang ada sehingga dapat meningkatkan fungsionalitas dari image processing.

1.2 PROJECT DESCRIPTION

VHDL grayscaler adalah sebuah sirkuit digital yang mengubah input gambar dari warna ke skala abu-abu. Grayscaler menggunakan jumlah terberat dari saluran warna merah, hijau, dan biru gambar masukan untuk menghitung nilai skala abu-abu untuk setiap piksel. Bobot yang diterapkan pada setiap saluran ditentukan oleh algoritma skala abu-abu yang ditentukan, yang dapat disesuaikan untuk menghasilkan efek visual yang berbeda.

VHDL grayscaler diimplementasikan menggunakan kombinasi kode VHDL behavioral dan structural. Kode perilaku mendefinisikan operasi keseluruhan dari grayscaler, termasuk perhitungan nilai skala abu-abu untuk setiap piksel. Kode struktural mendefinisikan komponen dan koneksi yang spesifik digunakan untuk menerapkan grayscaler, seperti register, multiplekser, dan penjumlah.

VHDL grayscaler dirancang untuk dapat disintesis, artinya dapat ditransformasikan menjadi deskripsi perangkat keras yang dapat digunakan untuk membuat sirkuit fisik. Grayscaler dapat digunakan dalam berbagai aplikasi, seperti pemrosesan video, manipulasi gambar, dan penglihatan komputer.

1.3 OBJECTIVES

1. Mendeskripsikan sebuah *image grayscaler* ke dalam bentuk VHDL.
2. Mengimplementasikan pemodelan Finite State Machine (FSM) pada cara kerja *image grayscaler.*
3. Melakukan simulasi terhadap “Image Grayscaler” dengan menggunakan testbench dan ModelSim.

1.4 ROLES AND RESPONSIBILITIES

Adapun peran dan tanggung jawab yang kami bagikan dengan anggota kelompok :

Roles	Responsibilities	Person
Ketua	Pembacaan file dan testbench testing	Arka Brian Dewara
Anggota 1	Riset mekanisme dan test grayscale	Azzah Azkiyah Angeli Syahwa
Anggota 2	Riset mekanisme dan testbench	Handaneswari Pramudhyta Imanda
Anggota 3	Laporan dan top level	Miranti Anggunsari

Table 1. Roles and Responsibilities

CHAPTER 2

IMPLEMENTATION

2.1 EQUIPMENT

The tools that are going to be used in this project are as follows:

- VSCODE
- Source code VHDL (main coding dan test bench)
- ModelSim
- Figma (FSM)

2.2 IMPLEMENTATION

Grayscale Maker merupakan sebuah program yang tujuannya mengubah gambar yang dimasukkan user menjadi berwarna hitam putih. Rangkaian akan menerima input berupa file bmp kemudian menghitung besar dan panjangnya (width and height) dari gambar tersebut. Setelah itu, program akan membaca rgb value dari gambar yang diinput dan menyimpannya. RGB value tersebut kemudian mengubahnya menjadi grayscale image.

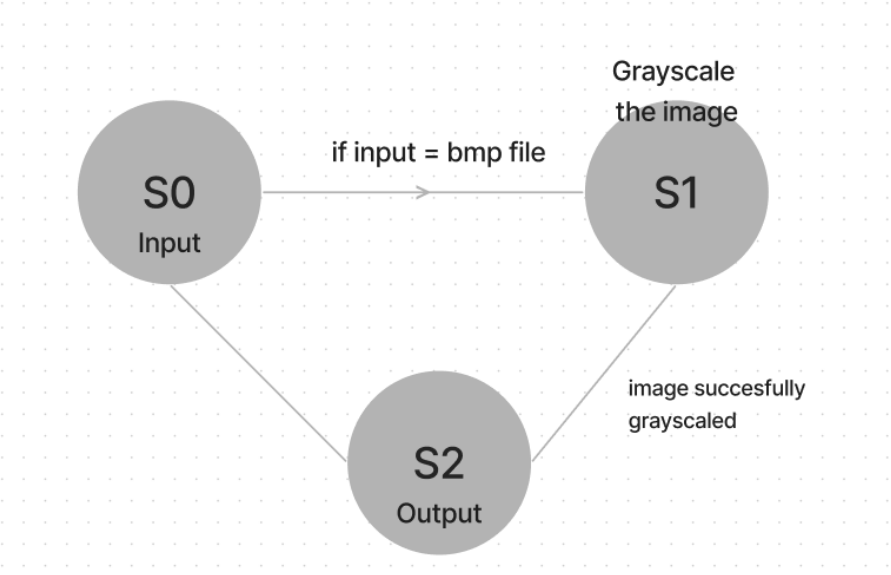


Fig 1. Finite State Machine

Program bekerja dengan mengubah warna gambar yang awalnya berupa RGB (red-green-blue) menjadi abu-abu. Pertama-tama, program akan melakukan pengecekan pada

header file, apakah dia dalam format bmp atau bukan. Jika bukan, maka program akan mengalami error :

```
--Baca header
for i in header_type'range loop
  read(bmp_file, header(i));
end loop;
```

Jika input yang dimasukan dalam bentuk bmp, maka selanjutnya program akan mengecek offset pada file. Pada program, offset pada header di set 54 bytes. Jika gambar yang dimasukkan tidak memenuhi syarat, maka program akan berhenti bekerja:

```
--Cek header, jika bukan 54-byte, keluar
assert character'pos(header(10)) = 54 and
  character'pos(header(11)) = 0 and
  character'pos(header(12)) = 0 and
  character'pos(header(13)) = 0
  report "Header is not 54 bytes"
  severity failure;
```

Kemudian dilakukan lagi pengecekan untuk mengecek apakah file yang dimasukkan merupakan file bmp dengan mengecek apakah gambar tersebut memiliki DIB header. DIB header merupakan sebuah header yang hanya dimiliki oleh file berbentuk bmp:

```
--Cek header, jika bukan 40-byte dib header, keluar
assert character'pos(header(14)) = 40 and
  character'pos(header(15)) = 0 and
  character'pos(header(16)) = 0 and
  character'pos(header(17)) = 0
  report "DIB headers size is not 40 bytes"
  severity failure;
```

```
            --Baca pixel biru
            read(bmp_file, char);
            row(col_i).blue :=
              std_logic_vector(to_unsigned(character'pos(char), 8));

            --Baca pixel hijau
            read(bmp_file, char);
            row(col_i).green :=
              std_logic_vector(to_unsigned(character'pos(char), 8));

            --Baca pixel merah
            read(bmp_file, char);
            row(col_i).red :=
              std_logic_vector(to_unsigned(character'pos(char), 8));

          end loop;
```

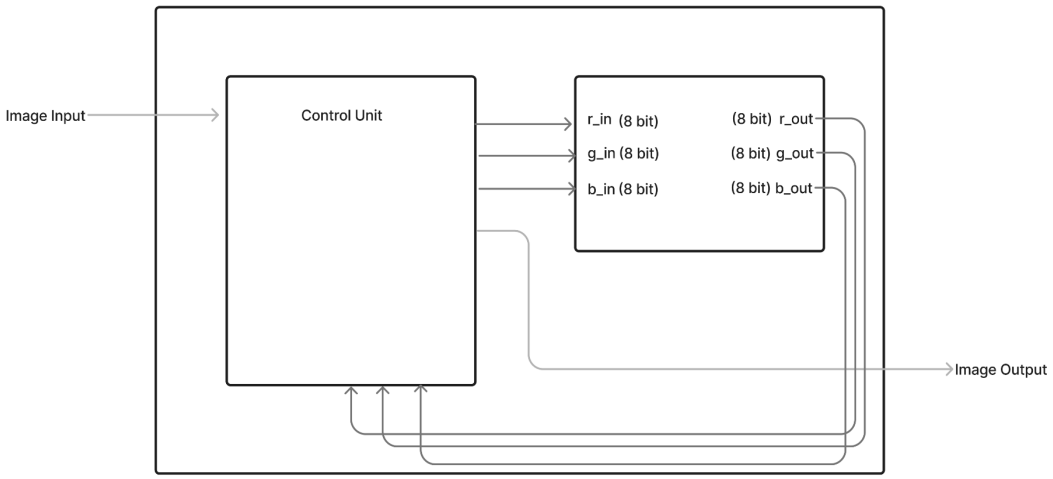
```
        -- RGB input
        r_in : in std_logic_vector(7 downto 0);
        g_in : in std_logic_vector(7 downto 0);
        b_in : in std_logic_vector(7 downto 0);
```

Program kemudian akan membaca rgb value dan menjadikannya input 8-bit untuk. Input tersebut berupa red, green, dan blue. Kemudian program memiliki 8-bit output untuk merepresentasikan nilai grayscale dari gambar Program kemudian menggunakan sebuah mekanisme yang digunakan untuk mengubah sebuah gambar menjadi hitam putih, yaitu

dengan menjumlahkan red, green, dan blue untuk menghasilkan nilai grayscale sejumlah 8-bit untuk setiap pixel. Formula yang digunakan adalah:

```
luma <= std_logic_vector(r_weighted + g_weighted + b_weighted);
```

Formula tersebut merupakan cara untuk mengubah sebuah RGB image menjadi grayscale. Formula tersebut merupakan perkiraan dari titik tetap dari komponen luma (yang merepresentasikan proporsi hitam-putih dari suatu gambar).



CHAPTER 3

TESTING AND ANALYSIS

3.1 TESTING

Rangkaian Grayscale maker yang telah diimplementasikan ke VHDL sesuai dengan cara kerja seluruh komponennya selanjutnya diuji agar mendapatkan output yang akurat sesuai dengan input yang diberikan. Pengujian dilakukan dengan menggunakan test bench dan sebuah image dalam bentuk bmp. Pengujian dilakukan untuk melihat apakah program dapat menghasilkan output berupa image dengan versi hitam putih dari input yang diberikan.

Dalam melakukan simulasi program ini, kami menggunakan ModelSim untuk dapat melihat sinyal-sinyal dari input maupun output program ini dengan bentuk wave. Dari sinyal wave ini juga akan membantu untuk melakukan analisis terhadap hasil yang didapatkan sehingga kami dapat mengetahui apakah program yang dibuat ini sudah sesuai dengan cara kerja yang sudah dideskripsikan dalam FSM.

Program yang digunakan dalam melakukan pengujian adalah sebagai berikut :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;
use std.env.finish;


entity grayscale_tb is
end grayscale_tb;
architecture sim of grayscale_tb is
  signal PresentState, NextState : integer range 0 to 2;
  type header_type  is array (0 to 53) of character;
  type pixel_type is record
    red : std_logic_vector(7 downto 0);
    green : std_logic_vector(7 downto 0);
    blue : std_logic_vector(7 downto 0);
  end record;
  type row_type is array (integer range <>) of pixel_type;
  type row_pointer is access row_type;
  type image_type is array (integer range <>) of row_pointer;
  type image_pointer is access image_type;
  -- DUT signals
  signal r_in : std_logic_vector(7 downto 0);
  signal g_in : std_logic_vector(7 downto 0);
```

```
  signal b_in : std_logic_vector(7 downto 0);
  signal r_out : std_logic_vector(7 downto 0);
  signal g_out : std_logic_vector(7 downto 0);
  signal b_out : std_logic_vector(7 downto 0);

  -- Clock signal
  signal clk : std_logic := '0';

  -- Clock generator
  component clock_gen is
    port (
      clk : out std_logic
    );
  end component;

begin
  DUT :entity work.grayscale(rtl)
  port map (
    clk => clk,
    r_in => r_in,
    g_in => g_in,
    b_in => b_in,
    r_out => r_out,
    g_out => g_out,
    b_out => b_out
  );
  process
    type char_file is file of character;
    file bmp_file : char_file open read_mode is "input.bmp";
```

```
  file out_file : char_file open write_mode is "output.bmp";
  variable header : header_type;
  variable image_width : integer;
  variable image_height : integer;
  variable row : row_pointer;
  variable image : image_pointer;
  variable padding : integer;
  variable char : character;
begin
  for i in header_type'range loop
    read(bmp_file, header(i));
  end loop;

  PresentState <= 0;
  NextState <= 1;

  assert header(0) = 'B' and header(1) = 'M'
    report "NOT A BMP FILE"
    severity failure;

  assert character'pos(header(10)) = 54 and
    character'pos(header(11)) = 0 and
    character'pos(header(12)) = 0 and
    character'pos(header(13)) = 0
    report "Header is not 54 bytes"
    severity failure;

  assert character'pos(header(14)) = 40 and
    character'pos(header(15)) = 0 and
```

```
    character'pos(header(16)) = 0 and
    character'pos(header(17)) = 0
    report "DIB headers size is not 40 bytes"
    severity failure;

  assert character'pos(header(26)) = 1 and
    character'pos(header(27)) = 0
    report "Color planes is not 1" severity failure;

  assert character'pos(header(28)) = 24 and
    character'pos(header(29)) = 0
    report "Bits per pixel is not 24" severity failure;

  image_width := character'pos(header(18)) +
    character'pos(header(19)) * 2**8 +
    character'pos(header(20)) * 2**16 +
    character'pos(header(21)) * 2**24;

  image_height := character'pos(header(22)) +
    character'pos(header(23)) * 2**8 +
    character'pos(header(24)) * 2**16 +
    character'pos(header(25)) * 2**24;

  report "image_width: " & integer'image(image_width) &
    ", image_height: " & integer'image(image_height);

  padding := (4 - image_width*3 mod 4) mod 4;

  image := new image_type(0 to image_height - 1);
```

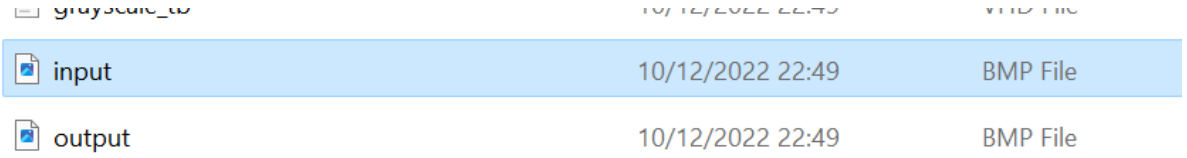
```
PresentState <= 1;
NextState <= 2;
for row_i in 0 to image_height - 1 loop
  row := new row_type(0 to image_width - 1);
  for col_i in 0 to image_width - 1 loop
    read(bmp_file, char);
    row(col_i).blue :=
      std_logic_vector(to_unsigned(character'pos(char), 8));
    read(bmp_file, char);
    row(col_i).green :=
      std_logic_vector(to_unsigned(character'pos(char), 8));
    read(bmp_file, char);
    row(col_i).red :=
      std_logic_vector(to_unsigned(character'pos(char), 8));
  end loop;

  for i in 1 to padding loop
    read(bmp_file, char);
  end loop;
  image(row_i) := row;
end loop;

for row_i in 0 to image_height - 1 loop
  row := image(row_i);
  for col_i in 0 to image_width - 1 loop
    r_in <= row(col_i).red;
    g_in <= row(col_i).green;
    b_in <= row(col_i).blue;
```

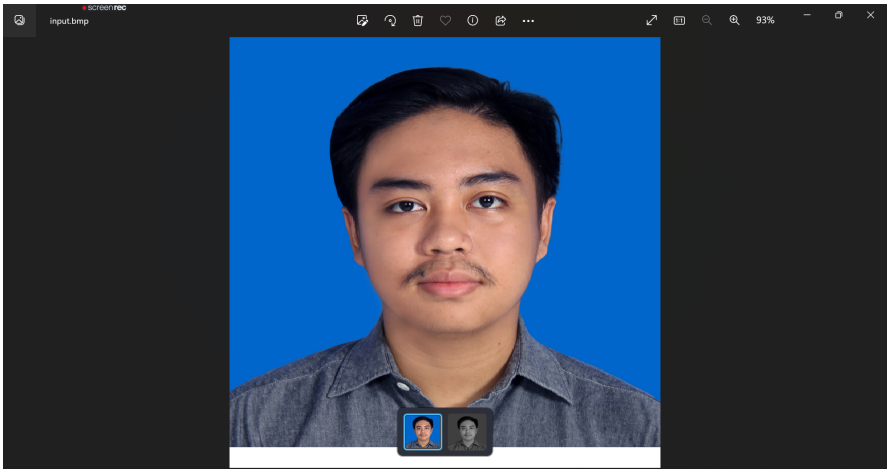
```
      wait for 10 ns;
      row(col_i).red := r_out;
      row(col_i).green := g_out;
      row(col_i).blue := b_out;
    end loop;
  end loop;

  PresentState <= 2;
  nextstate <= 0;

  for i in header_type'range loop
    write(out_file, header(i));
  end loop;

  for row_i in 0 to image_height - 1 loop
    row := image(row_i);
    for col_i in 0 to image_width - 1 loop
      write(out_file,
        character'val(to_integer(unsigned(row(col_i).blue))));
      write(out_file,
        character'val(to_integer(unsigned(row(col_i).green))));
      write(out_file,
        character'val(to_integer(unsigned(row(col_i).red))));
    end loop;

    deallocate(row);

    for i in 1 to padding loop
      write(out_file, character'val(0));
```

```
      end loop;
    end loop;


    deallocate(image);
    file_close(bmp_file);
    file_close(out_file);
    report "Simulation done. Check ""output.bmp"" image.";
    finish;
  end process;
end architecture;
```

3.2 RESULT

Dari hasil pengujian, seluruh komponen terlah berjalan semestinya. Pada program diterima input berupa BMP file dengan nama input, kemudian akan dihasilkan gambar yang sudah di grayscale bernama output dengan bentuk BMP juga. berikut foto yang dihasilkan:





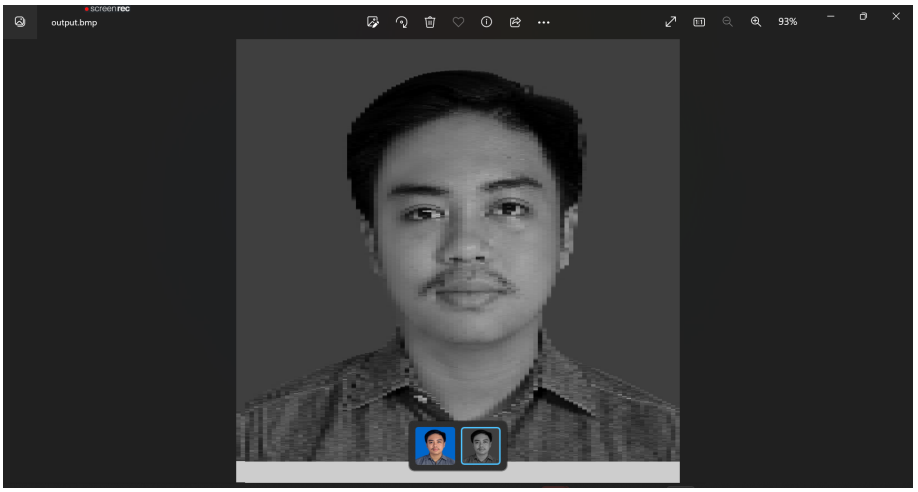
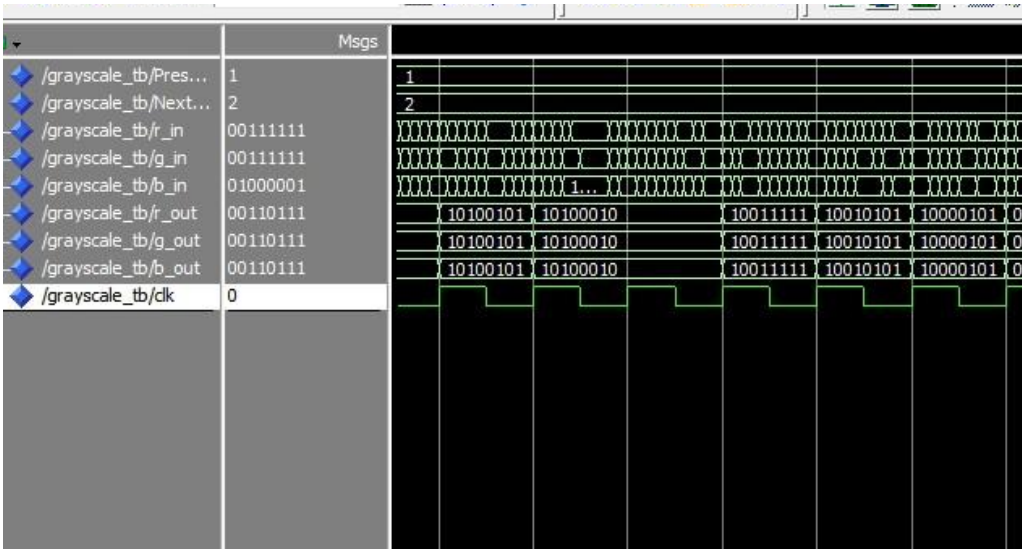


Foto berupa screenshot karena google docs tidak mensupport file dengan bentuk bmp.

3.2.1 Hasil Percobaan



Berikut merupakan wave dari testbench yang dilakukan pada modelsim dengan nilai clock 100.000, clock yang diberikan harus besar agar proses mendapat outputnya cepat dilakukan. Jika clock yang diberikan hanya senilai 100 maka prosesnya bisa memakan waktu dengan estimasi kurang lebih 5 menit.

3.3 ANALYSIS

Dari pengujian yang dilakukan, seluruh komponen telah berjalan semestinya, dan telah berhasil mengolah file input berupa bmp menjadi output berupa image dengan warna black and white dalam bentuk bmp, sesuai dengan input yang kita inginkan.

Program telah berhasil menganalisis apakah gambar yang diinput merupakan bmp atau tidak, kemudian membaca pixel yang ada pada input. Setelah itu menyimpan rgb value dari gambar, dan mengubahnya sesuai rumus yang ada pada program menjadi sebuah value yang menjadikan gambarnya hitam putih.

CHAPTER 4

CONCLUSION

Image Scaler yang dideskripsikan pada VHDL hanya akan menerima input dengan format BMP. Biasanya digunakan untuk menyimpan gambar yang memilki banyak detail, seperti foto beresolusi tinggi. File BMP sering digunakan untuk membuat dan mengedit grafik, serta untuk menampilkan gambar di halaman web dan aplikasi lain. Penggunaan *image grayscaler* dengan format BMP ditujukan untuk mengubah citra berwarna menjadi versi grayscale yang berguna untuk berbagai tujuan, seperti mengurangi ukuran file gambar, membuatnya lebih mudah untuk dimanipulasi, atau meningkatkan daya tarik visualnya.

REFERENCES

[1] Isahit (2022) *Why to use grayscale conversion during image processing?*, *Isahit*. isahit. Available at: https://www.isahit.com/blog/why-to-use-grayscale-conversion-during-image-processing (Accessed: December 10, 2022).

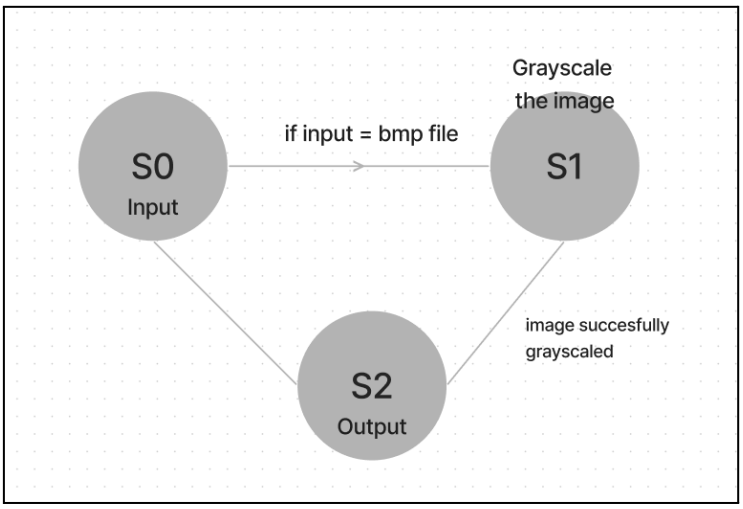
[2] Wilhelm Burger, Mark J. Burge (2010). Principles of Digital Image Processing Core Algorithm. (Accessed: December 10, 2022)

[3] *Grayscale to RGB conversion* (no date) *Tutorials Point*. Available at: https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.html.(Accessed: December 10, 2022).

APPENDICES

Appendix A: Project Schematic

Finite state machine yang dibuat secara manual adalah sebagai berikut :



Appendix B: Documentation