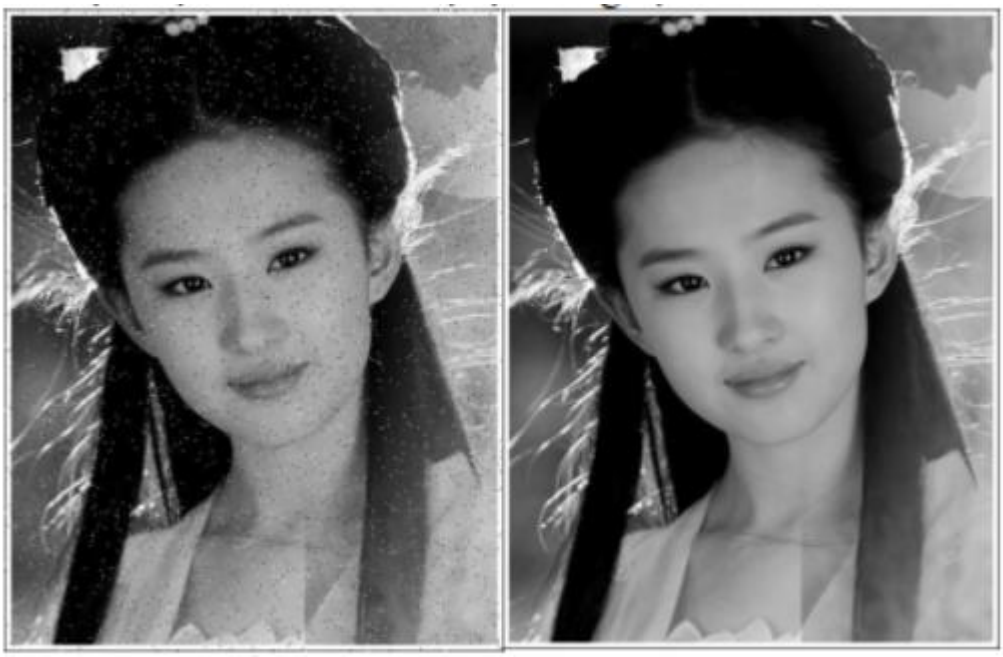
Họ và tên thành viên 1: **Nguyễn Việt Dũng**

MSSV thành viên 1: **23520338**

Lớp: **CE213.Q12**

LAB 2

<u>Bài tập 1</u> : Xây dựng bộ lọc trung vị (median filter) cho bức ảnh



I. Thuật toán lọc trung vị

Lọc trung vị là một kỹ thuật lọc nhiễu phi tuyến thường được sử dụng trong xử lý ảnh để loại bỏ nhiễu, đặc biệt là nhiễu muối tiêu (salt-and-pepper noise) trong khi vẫn bảo tồn được các cạnh (cạnh sắc) của hình ảnh.

1. Nguyên lý hoạt động

Nguyên lý cơ bản của lọc trung vị là thay thế giá trị của một điểm ảnh bằng giá trị trung vị (giá trị nằm ở giữa danh sách đã sắp xếp) của các điểm ảnh trong vùng lân cận (cửa sổ lọc).

Các bước thực hiện:

1. Xác định cửa sổ lọc (Window): Chọn một vùng bao quanh điểm ảnh hiện tại (thường là hình vuông kích thước 3 x 3, 5 x 5, 7 x 7).

2. Liệt kê: Lấy tất cả các giá trị mức xám của các điểm ảnh trong cửa sổ đó.
3. Sắp xếp: Sắp xếp các giá trị này theo thứ tự tăng dần hoặc giảm dần.
4. Chọn trung vị: Tìm giá trị nằm ở chính giữa danh sách đã sắp xếp.
5. Thay thế: Gán giá trị trung vị này cho điểm ảnh ở tâm cửa sổ.

2. Ưu điểm và Nhược điểm

Đặc điểm	**Chi tiết**
Ưu điểm	- Loại bỏ cực tốt nhiễu muối tiêu (các đốm trắng đen lỗi). - Không làm nhòe các cạnh sắc của ảnh như bộ lọc trung bình (Mean Filter). - Giữ lại được chi tiết biên.
Nhược điểm	- Tốn tài nguyên tính toán hơn (do phải thực hiện các thuật toán sắp xếp). - Nếu cửa sổ lọc quá lớn, ảnh có thể bị mất các chi tiết nhỏ hoặc bị biến dạng cấu trúc.

II. Chuyển ảnh nhiễu sang ảnh file pic_input.txt bằng python

CreateFileTxt.py - E:\HDL_ALL\LAB2\BaiTap1\CreateFileTxt.py (3.13.0)

File Edit Format Run Options Window Help

```python
from PIL import Image
import numpy as np
import os

# ================== CẤU HÌNH ==================
INPUT_IMAGE  = "baitap1_nhieu.jpg"
OUTPUT_FILE  = "pic_input.txt"
WIDTH  = 430
HEIGHT = 554
# ==============================================

if not os.path.exists(INPUT_IMAGE):
    raise FileNotFoundError("Không tìm thấy file ảnh!")

# Mở ảnh và chuyển grayscale
img = Image.open(INPUT_IMAGE).convert("L")

# Resize cho khớp Verilog
img = img.resize((WIDTH, HEIGHT))

# Lấy dữ liệu pixel (row-major)
pixels = np.array(img, dtype=np.uint8).flatten()

# Ghi ra file HEX
with open(OUTPUT_FILE, "w") as f:
    for p in pixels:
        f.write(f"{p:02X}\n")

print("✔ Tạo pic_input.txt thành công")
print("✔ Tổng pixel:", WIDTH * HEIGHT)
```

III. Code Verilog xử lý lọc trung vị

1. Xây dựng các module

a) median9 (module lọc trung vị theo cửa sổ 3 x 3)

```
module median9 (
    input  [7:0] p0,p1,p2,p3,p4,p5,p6,p7,p8,
    output [7:0] med
);
    reg [7:0] a0,a1,a2,b0,b1,b2,c0,c1,c2;
    reg [7:0] d0,d1,d2,e0,e1,e2,f0,f1,f2;
    reg [7:0] m0,m1,m2,m3,m4;
    reg [7:0] t;

    always @(*) begin
        // Stage 1: Sort Rows
        a0=p0; a1=p1; a2=p2; if(a0>a1)begin t=a0;a0=a1;a1=t;end if(a1>a2)begin t=a1;a1=a2;a2=t;end if(a0>a1)begin t=a0;a0=a1;a1=t;end
        b0=p3; b1=p4; b2=p5; if(b0>b1)begin t=b0;b0=b1;b1=t;end if(b1>b2)begin t=b1;b1=b2;b2=t;end if(b0>b1)begin t=b0;b0=b1;b1=t;end
        c0=p6; c1=p7; c2=p8; if(c0>c1)begin t=c0;c0=c1;c1=t;end if(c1>c2)begin t=c1;c1=c2;c2=t;end if(c0>c1)begin t=c0;c0=c1;c1=t;end

        // Stage 2: Sort Cols
        d0=a0; d1=b0; d2=c0; if(d0>d1)begin t=d0;d0=d1;d1=t;end if(d1>d2)begin t=d1;d1=d2;d2=t;end if(d0>d1)begin t=d0;d0=d1;d1=t;end
        e0=a1; e1=b1; e2=c1; if(e0>e1)begin t=e0;e0=e1;e1=t;end if(e1>e2)begin t=e1;e1=e2;e2=t;end if(e0>e1)begin t=e0;e0=e1;e1=t;end
        f0=a2; f1=b2; f2=c2; if(f0>f1)begin t=f0;f0=f1;f1=t;end if(f1>f2)begin t=f1;f1=f2;f2=t;end if(f0>f1)begin t=f0;f0=f1;f1=t;end

        // Stage 3: Cross
        m0=e0; m1=d1; m2=e1; m3=f1; m4=e2;
        if(m0>m1)begin t=m0;m0=m1;m1=t;end
        if(m3>m4)begin t=m3;m3=m4;m4=t;end
        if(m2>m4)begin t=m2;m2=m4;m4=t;end
        if(m2>m3)begin t=m2;m2=m3;m3=t;end
        if(m1>m4)begin t=m1;m1=m4;m4=t;end
        if(m0>m3)begin t=m0;m0=m3;m3=t;end
        if(m0>m2)begin t=m0;m0=m2;m2=t;end
        if(m1>m3)begin t=m1;m1=m3;m3=t;end
        if(m1>m2)begin t=m1;m1=m2;m2=t;end
    end
    assign med = m2;
endmodule
```

Giải thích:

- Module này tìm giá trị trung vị bằng cách thực hiện 3 bước lọc chính:
  - Sắp xếp các số trong từng hàng.
  - Sắp xếp các số trong từng cột (dựa trên kết quả hàng đã sắp xếp).
  - Tìm trung vị của các phần tử "ứng cử viên" còn lại.
- Từng giai đoạn:
  - Sắp xếp theo hàng: Code thực hiện sắp xếp tăng dần cho từng hàng riêng biệt:
    - Hàng 1 (a0, a1, a2) ← Sắp xếp (p0, p1, p2)
    - Hàng 2 (b0, b1, b2) ← Sắp xếp (p3, p4, p5)
    - Hàng 3 (c0, c1, c2) ← Sắp xếp (p6, p7, p8)
  - Sắp xếp theo cột: Code thực hiện sắp xếp tăng dần cho từng cột riêng biệt:
    - Hàng 1 (d0, d1, d2) ← Sắp xếp (a0, b0, c0)
    - Hàng 2 (e0, e1, e2) ← Sắp xếp (a1, b1, c1)
    - Hàng 3 (f0, f1, f2) ← Sắp xếp (a2, b2, c2)
  - Lọc chéo và chọn trung vị: Code chọn ra 5 ứng cử viên tiềm năng nhất để tìm trung vị cuối cùng:
    - m0 = e0
    - m1 = d1
    - m2 = e1 (Phần tử trung tâm của ma trận)
    - m3 = f1
    - m4 = e2

Sau đó, code thực hiện một loạt các phép so sánh và hoán đổi (Bubble sort/Insertion sort) trên 5 biến m này để đảm bảo chúng được sắp xếp.

- **Kết quả:** Biến nằm giữa (m2) sau khi sắp xếp chính là **giá trị trung vị của cả 9 pixel ban đầu**.

b) MedianFilter (module lọc ảnh)

i. Khai báo tín hiệu và tham số

```
1   module MedianFilter #(
2       parameter WIDTH = 430                   // Chiều rộng ảnh
3   )(
4       input  wire         clk,                // Tín hiệu Clock
5       input  wire         rst,                // Tín hiệu reset
6       input  wire         pixel_valid,        // Tín hiệu báo có dữ liệu gửi vào
7       input  wire [7:0]   pixel_in,           // Giá trị pixel đầu vào (0-255)
8
9       output reg          pixel_out_valid,    // Tín hiệu báo đang có output hợp lệ
10      output reg  [7:0]   pixel_out           // Giá trị pixel đầu ra
11  );
12
```

ii. Bộ nhớ đệm dòng

```
13      // Buffer lưu 2 dòng (Kích thước phải đúng bằng WIDTH)
14      reg [7:0] line1 [0:WIDTH-1];            // Lưu trữ dòng thứ N-2
15      reg [7:0] line2 [0:WIDTH-1];            // Lưu trữ dòng thứ N-1
```

iii. Biến đếm tọa độ và thanh ghi cửa sổ

```
16
17      reg [9:0] x;  // biến đếm vị trí cột
18      reg [9:0] y;  // biến đếm vị trí dòng
19
```

iv. Khai báo cửa sổ 3 x 3

```
20      // Cửa sổ 3x3
21      reg [7:0] p0, p1, p2;
22      reg [7:0] p3, p4, p5;
23      reg [7:0] p6, p7, p8;
24
```

v. Kết nối module tính toán

```
27      // Module tìm trung vị
28      median9 u_med (
29          .p0(p0), .p1(p1), .p2(p2),
30          .p3(p3), .p4(p4), .p5(p5),
31          .p6(p6), .p7(p7), .p8(p8),
32          .med(med)
33      );
34
```

vi. Phần xử lý lọc ảnh

a. Khởi tạo

```
35      always @(posedge clk or posedge rst) begin
36          if (rst) begin
37              x <= 0;
38              y <= 0;
39              pixel_out_valid <= 0;
40              pixel_out <= 0;
41              p0 <= 0; p1 <= 0; p2 <= 0;
42              p3 <= 0; p4 <= 0; p5 <= 0;
43              p6 <= 0; p7 <= 0; p8 <= 0;
```

b. Phần xử lý chính

- Dịch cửa sổ

```
44          end else if (pixel_valid) begin
45              // 1. Dịch cửa sổ
46              p0 <= p1;  p1 <= p2;  p2 <= line1[x];
47              p3 <= p4;  p4 <= p5;  p5 <= line2[x];
48              p6 <= p7;  p7 <= p8;  p8 <= pixel_in;
49
```

- Cập nhật bộ đệm

```
49
50              // 2. Cập nhật lại buffer
51              line1[x] <= line2[x];
52              line2[x] <= pixel_in;
53
```

- Logic xuất dữ liệu & Xử lý viền

```
53
54          // 3. Logic xuất dữ liệu
55          // p4 chứa dữ liệu tại dòng y-1.
56          // Ta bắt đầu xuất khi đã nạp đủ 1 dòng đầu tiên (y >= 1)
57          if (y >= 1) begin
58              pixel_out_valid <= 1'b1;
59
60              // Nếu ở vùng giữa (x>=2 và y>=2): Lọc Median
61              if (x >= 2 && y >= 2) begin
62                  pixel_out <= med;
63              end
64              // Nếu ở viền: Giữ nguyên giá trị gốc (p4)
65              else begin
66                  pixel_out <= p4;
67              end
68          end else begin
69              pixel_out_valid <= 1'b0;
70          end
71
```

- Xử lý bộ đếm tọa độ

```
72          // 4. Xử lý bộ đếm tọa độ
73          if (x == WIDTH-1) begin
74              x <= 0;
75              y <= y + 1;
76          end else begin
77              x <= x + 1;
78          end
```

2. Testbench

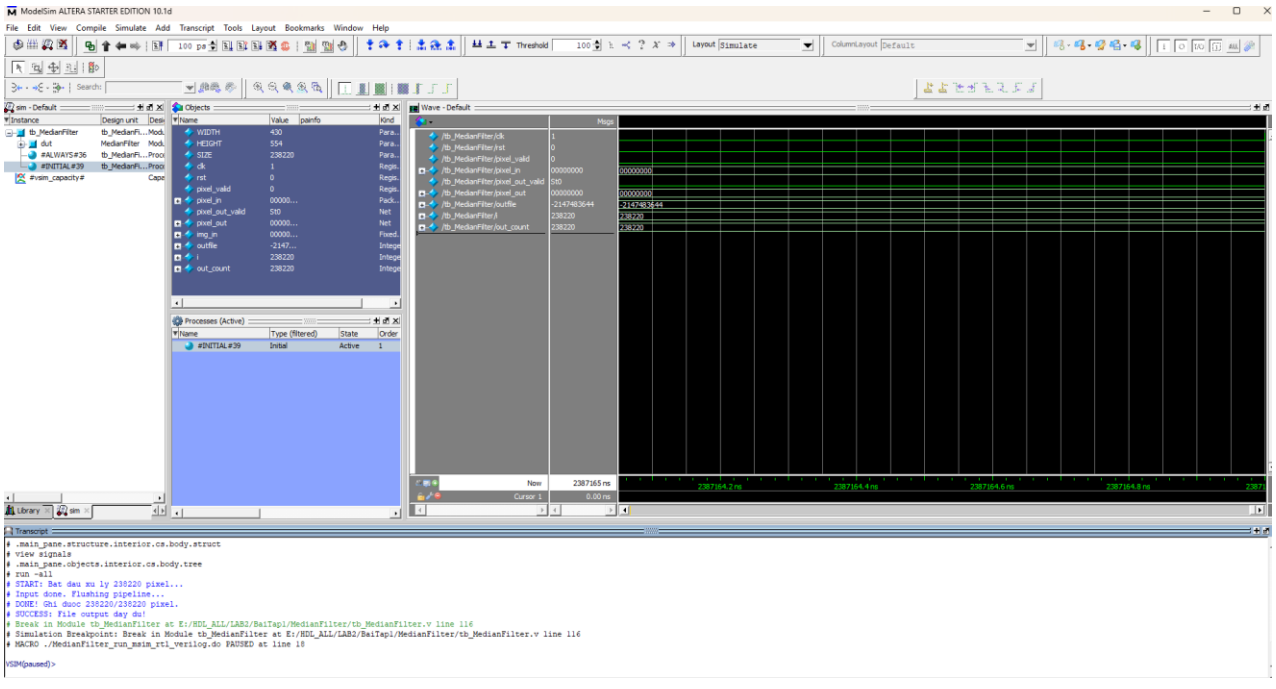
```
1   `timescale 1ns/1ps
2
3   module tb_MedianFilter;
4       // --- 1. CẤU HÌNH KÍCH THƯỚC ẢNH ---
5       parameter WIDTH  = 430; // Chiều rộng ảnh (số cột)
6       parameter HEIGHT = 554; // Chiều cao ảnh (số dòng)
7       parameter SIZE   = WIDTH * HEIGHT; // Tổng số pixel (238,220)
8
9       // --- 2. KHAI BÁO TÍN HIỆU ---
10      reg clk;
11      reg rst;
12      reg pixel_valid;    // Biến điều khiển: báo cho module biết đang gửi data
13      reg [7:0] pixel_in; // Dữ liệu pixel đầu vào (từng byte một)
14
15      wire pixel_out_valid; // Module báo lại: "Tao đã tính xong 1 pixel"
16      wire [7:0] pixel_out; // Giá trị pixel kết quả
17
18      // --- 3. KẾT NỐI MODULE (DUT) ---
19      MedianFilter #(.WIDTH(WIDTH)) dut (
20          .clk(clk),
21          .rst(rst),
22          .pixel_valid(pixel_valid),
23          .pixel_in(pixel_in),
24          .pixel_out_valid(pixel_out_valid),
25          .pixel_out(pixel_out)
26      );
27
28      // --- 4. BIẾN HỖ TRỢ ĐỌC/GHI FILE ---
29      reg [7:0] img_in [0:SIZE-1]; // Mảng lớn chứa toàn bộ ảnh input
30      integer outfile;             // Biến đại diện file output
31      integer i;                   // Biến đếm vòng lặp gửi input
32      integer out_count;           // Đếm số pixel đã ghi được ra file output
33
34      // --- 5. TẠO XUNG CLOCK ---
35      initial clk = 0;
36      always #5 clk = ~clk; // Chu kỳ 10ns (100MHz)
37
38      // --- 6. CHƯƠNG TRÌNH CHÍNH ---
39      initial begin
40          // A. Khởi tạo giá trị ban đầu
41          rst = 1; pixel_valid = 0; pixel_in = 0;
42          i = 0; out_count = 0;
43
44          // B. Load ảnh từ file txt vào bộ nhớ mảng
45
```

```
46          $readmemh("pic_input.txt", img_in);
47
48          // C. Tạo file để ghi kết quả
49          outfile = $fopen("pic_output.txt", "w");
50
51          // D. Reset hệ thống
52          #20 rst = 0;          // Thả reset sau 20ns
53          repeat(5) @(posedge clk); // Đợi vài chu kỳ cho ổn định
54
55          // --- GIAI ĐOẠN XỬ LÝ (PIPELINE) ---
56          $display("START: Bat dau xu ly %0d pixel...", SIZE);
57
58          // Bật cờ Valid lên 1 để báo hiệu bắt đầu truyền dữ liệu
59          pixel_valid = 1;
60
61          // Vòng lặp: Gửi từng pixel vào module
62          while (i < SIZE) begin
63              // 1. Đưa dữ liệu vào
64              pixel_in <= img_in[i]; // Lấy pixel thứ i gửi vào
65              i = i + 1;             // Tăng biến đếm input
66
67              // 2. Chờ cạnh lên của clock (đồng bộ hóa)
68              @(posedge clk);
69
70              // 3. Kiểm tra xem có Output văng ra chưa?
71              // Dùng #1 để lùi thời gian một chút sau cạnh lên, đảm bảo đọc đúng giá trị dây
72              #1;
73              if (pixel_out_valid) begin
74                  // Ghi vào file dưới dạng Hex (2 số), ví dụ: A5
75                  $fwrite(outfile, "%2h\n", pixel_out);
76                  out_count = out_count + 1;
77              end
78          end
79
80          // --- GIAI ĐOẠN FLUSH (ĐẨY DỮ LIỆU TỒN ĐỌNG) ---
81          // Khi gửi hết pixel cuối cùng, module vẫn còn giữ dữ liệu trong Line Buffer (2 dòng).
82          // Ta cần tiếp tục gửi tín hiệu (dù là rác) để đẩy kết quả ra ngoài.
83
84          $display("Input done. Flushing pipeline...");
85
86          pixel_valid = 1; // Vẫn giữ valid
87
88          // Lặp thêm khoảng 1 dòng + dư ra xíu để đảm bảo đẩy hết sạch
89          repeat(WIDTH + 50) begin
```

```
90              pixel_in <= 0; // Gửi số 0 vào (padding), không quan trọng giá trị này
91              @(posedge clk);
92
93              #1; // Kiểm tra output
94              if (pixel_out_valid) begin
95                  // Chỉ ghi nếu chưa đủ số lượng (đề phòng ghi dư)
96                  if (out_count < SIZE) begin
97                      $fwrite(outfile, "%2h\n", pixel_out);
98                      out_count = out_count + 1;
99                  end
100             end
101         end
102
103         // --- KẾT THÚC ---
104         pixel_valid = 0; // Tắt tín hiệu
105         repeat(10) @(posedge clk);
106
107         $fclose(outfile); // Đóng file output quan trọng để lưu dữ liệu
108         $display("DONE! Ghi duoc %0d/%0d pixel.", out_count, SIZE);
109
110         // Kiểm tra nhanh kết quả
111         if (out_count == SIZE)
112             $display("SUCCESS: File output day du!");
113         else
114             $display("WARNING: Thieu %0d pixel!", SIZE - out_count);
115
116         $stop; // Dừng mô phỏng
117     end
118 endmodule
```

Lưu ý: Chuyển thư mục pic_input.txt vào thư mục simulation/modelsim để chạy được testbench

3. Waveform chạy mô phỏng (Pre Simulation)



Lưu ý: File pic_output được sinh ra ở thư mục simulation/modelsim.

IV. Đánh giá ảnh tái tạo với ảnh gốc bằng python

Compare.py - E:\HDL_ALL\LAB2\BaiTap1\Compare.py (3.13.0)

File Edit Format Run Options Window Help

```python
import numpy as np
import matplotlib.pyplot as plt
from skimage.metrics import peak_signal_noise_ratio as psnr
from skimage.metrics import structural_similarity as ssim
import cv2

# --- CẤU HÌNH ---
WIDTH = 430
HEIGHT = 554

# Hàm đọc file Hex (từ bài trước)
def load_hex_image(filename):
    pixels = []
    try:
        with open(filename, 'r') as f:
            for line in f:
                line = line.strip()
                if not line or line.startswith('//'): continue
                for token in line.split():
                    try: pixels.append(int(token, 16))
                    except: continue

        expected = WIDTH * HEIGHT
        if len(pixels) != expected:
            print(f"Lưu ý: File {filename} có {len(pixels)} pixel (Mong đợi {expected}). Sẽ resize.")
            if len(pixels) < expected: pixels += [0]*(expected-len(pixels))
            else: pixels = pixels[:expected]

        return np.array(pixels, dtype=np.uint8).reshape((HEIGHT, WIDTH))
    except Exception as e:
        print(f"Lỗi đọc file {filename}: {e}")
        return None

# --- LOAD DỮ LIỆU ---

# 1. Load ảnh Output từ Verilog
img_verilog = load_hex_image('MedianFilter/simulation/modelsim/pic_output.txt')

# 2. Load ảnh Gốc (Reference)
try:
    # Đọc ảnh, chuyển sang grayscale
    img_ref = cv2.imread('baitap1_anhgoc.jpg', cv2.IMREAD_GRAYSCALE)
    # Resize về đúng kích thước 430x554 nếu cần
    if img_ref is not None:
        img_ref = cv2.resize(img_ref, (WIDTH, HEIGHT))
except:
    img_ref = None
    print("Không tìm thấy file ảnh gốc để so sánh.")


# --- TÍNH TOÁN VÀ HIỂN THỊ ---
if img_verilog is not None and img_ref is not None:

    # Tính PSNR
    score_psnr = psnr(img_ref, img_verilog)

    # Tính SSIM
    # data_range=255 vì ảnh 8-bit
    score_ssim = ssim(img_ref, img_verilog, data_range=255)
```

```
    print("-" * 30)
    print(f"KẾT QUẢ ĐÁNH GIÁ (Quantitative):")
    print(f"SIZE: {WIDTH}x{HEIGHT}")
    print(f"PSNR: {score_psnr:.4f} dB")
    print(f"SSIM: {score_ssim:.4f}")
    print("-" * 30)

    # Hiển thị ảnh để so sánh
    fig, ax = plt.subplots(1, 2, figsize=(10, 5))
    ax[0].imshow(img_ref, cmap='gray')
    ax[0].set_title("Ảnh Gốc (Reference)")
    ax[0].axis('off')

    ax[1].imshow(img_verilog, cmap='gray')
    ax[1].set_title(f"Ảnh Verilog Output\nPSNR: {score_psnr:.2f}dB - SSIM: {score_ssim:.2f}")
    ax[1].axis('off')

    plt.tight_layout()
    plt.show()

elif img_ref is None:
    print("Chưa load được ảnh gốc. Hãy cung cấp file 'original_image.jpg' hoặc 'pic_input.txt' để so sánh.")
```


```
>>>
================== RESTART: E:\HDL_ALL\LAB2\BaiTap1\Compare.py ==================
Lưu ý: File MedianFilter/simulation/modelsim/pic_output.txt có 238218 pixel (Mon
g đợi 238220). Sẽ resize.
------------------------------
KẾT QUẢ ĐÁNH GIÁ (Quantitative):
SIZE: 430x554
PSNR: 34.2257 dB
SSIM: 0.9625
------------------------------
```

V. Source code

Đường dẫn: [Link github]

Bài tập 2: Chuyển ảnh RGB sang ảnh grayscale

I. Chuyển ảnh RGB sang ảnh bitmap bằng python

ConvertRGB.py - E:\HDL_ALL\LAB2\BaiTap2\ConvertRGB.py (3.13.0)

File Edit Format Run Options Window Help

```
from PIL import Image

# 1. Mở ảnh và resize (nếu ảnh quá lớn, mô phỏng sẽ rất lâu)
img = Image.open("baitap2_anhgoc.jpg")
width, height = img.size
pixels = img.load()

print(f"Kích thước ảnh gốc: {width} x {height}")
print(f"Tổng số pixel: {width * height}")

# 2. Ghi ra file text (định dạng Hex: RR GG BB)
with open("image_in.hex", "w") as f:
    for y in range(height):
        for x in range(width):
            r, g, b = pixels[x, y]
            # Ghi mỗi dòng là 1 pixel gồm 24 bit (8 bit R, 8 bit G, 8 bit B)
            f.write(f"{r:02x}{g:02x}{b:02x}\n")

print(f"Đã tạo file image_in.hex với kích thước {width}x{height}")
```

IDLE Shell 3.13.0

File Edit Shell Debug Options Window Help

```
Python 3.13.0 (tags/v3.13.0:60403a5, Oct  7 2024, 09:38:07) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: E:\HDL_ALL\LAB2\BaiTap2\ConvertRGB.py ==========
Kích thước ảnh gốc: 2048 x 1365
Tổng số pixel: 2795520
Đã tạo file image_in.hex với kích thước 2048x1365
>>>
```

II. Xử lý trên Verilog

1. Xây dựng module

- median9 (module lọc trung vị theo cửa sổ 3 x 3)
  - Khai báo tín hiệu và tham số

```
module ConvertRGB (
    input [7:0] r,        // Tín hiệu Red (0-255)
    input [7:0] g,        // Tín hiệu Green (0-255)
    input [7:0] b,        // Tín hiệu Blue (0-255)
    input [3:0] level,    // Mức độ sáng người dùng chọn (ví dụ: 1 đến 10)

    output reg [7:0] gray_out // Kết quả ảnh xám cuối cùng (8-bit)
);

    // --- KHAI BÁO BIẾN TRUNG GIAN (REGISTERS) ---
    reg [15:0] mult_r, mult_g, mult_b;

    // Biến chứa tổng trọng số RGB (trước khi chia)
    reg [15:0] sum_gray;

    // Giá trị độ sáng được cộng thêm (tính toán từ level)
    reg [15:0] brightness_val;

    // Kết quả tạm thời sau khi cộng độ sáng (có thể vượt quá 255)
    reg [15:0] gray_final;
```

- **Phần xử lý chuyển đổi RGB sang grayscale**
  - **Chuyển đổi RGB sang grayscale**

```
    // ============================================================
    // BƯỚC 1: CHUYỂN ĐỔI RGB SANG GRAYSCALE
    // ============================================================
    // Công thức chuẩn mắt người: Y = 0.299R + 0.587G + 0.114B
    // Trong phần cứng (số nguyên), ta nhân cả 2 vế với 256 để khử số thập phân:
    // Y * 256 ≈ 77*R + 150*G + 29*B

    mult_r = r * 77;   // Trọng số cho màu Đỏ
    mult_g = g * 150;  // Trọng số cho màu Lục (Mắt nhạy nhất)
    mult_b = b * 29;   // Trọng số cho màu Lam (Mắt kém nhạy nhất)

    // Cộng tổng và chia cho 256 bằng cách dịch phải 8 bit (>> 8)
    sum_gray = (mult_r + mult_g + mult_b) >> 8;
```

  - **Tính toán độ sáng cộng thêm**

```
    // ============================================================
    // BƯỚC 2: TÍNH TOÁN ĐỘ SÁNG CỘNG THÊM
    // ============================================================
    // Quy đổi từ mức người dùng chọn (level 1-10) ra giá trị pixel thực tế.
    // Ví dụ: Level 5 -> cộng thêm 100 vào giá trị pixel.
    brightness_val = level * 20;
```

  - **Cộng độ sáng vào ảnh gốc**

```
    // ============================================================
    // BƯỚC 3: CỘNG ĐỘ SÁNG VÀO ẢNH GỐC
    // ============================================================
    gray_final = sum_gray + brightness_val;
```

  - **Chống tràn số**

```
        // ============================================================
        // BƯỚC 4: CHỐNG TRÀN SỐ (SATURATION / CLAMPING)
        // ============================================================
        // Vì gray_final là 16-bit, nó có thể chứa giá trị > 255 (ví dụ 300).
        // Nhưng output gray_out chỉ có 8-bit (max 255).
        // Nếu không có bước này, 300 sẽ bị cắt thành 44 (300 % 256), làm ảnh bị lốm đốm đen sai lệch.

        if (gray_final > 255)
            gray_out = 8'd255;  // Nếu vượt quá ngưỡng, gán bằng giá trị trắng nhất
        else
            gray_out = gray_final[7:0]; // Nếu an toàn, lấy 8 bit thấp
    end
endmodule
```

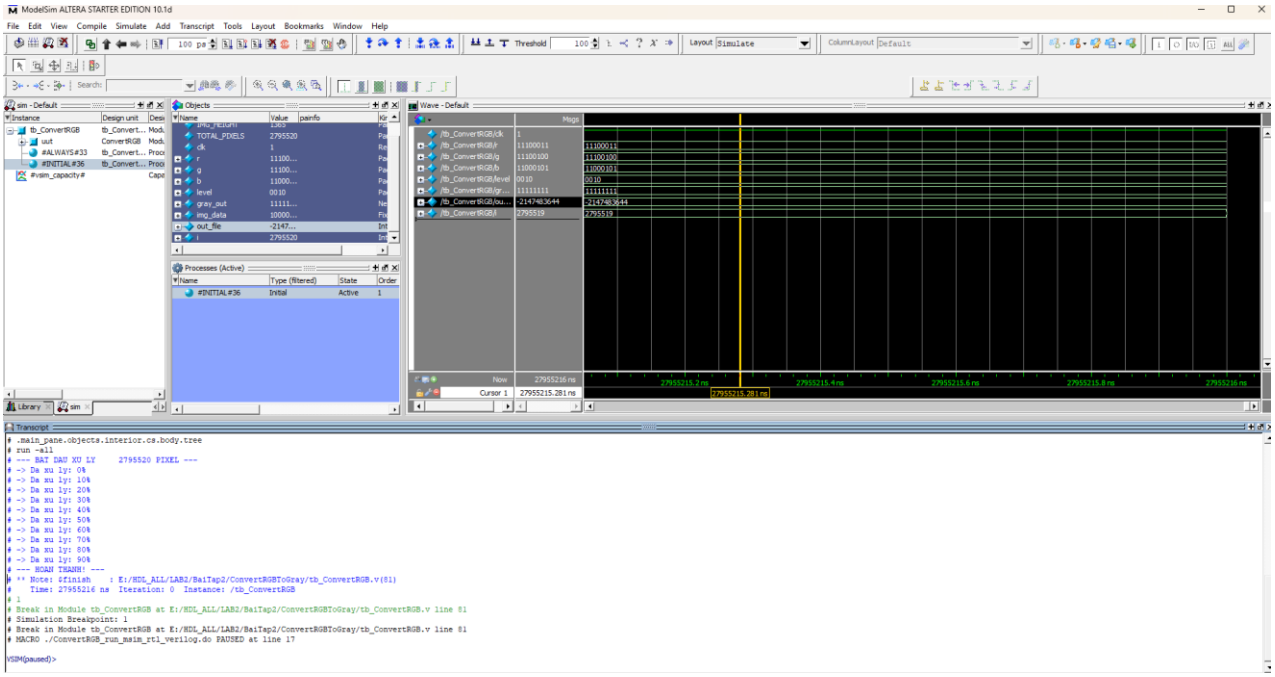
2. Testbench

```
`timescale 1ns/1ps

module tb_ConvertRGB;

    // --- 1. KHAI BÁO THAM SỐ & TÍN HIỆU ---
    parameter IMG_WIDTH  = 2048;
    parameter IMG_HEIGHT = 1365;
    parameter TOTAL_PIXELS = IMG_WIDTH * IMG_HEIGHT; // Tổng số lần lặp

    reg clk;
    reg [7:0] r, g, b;
    reg [3:0] level;
    wire [7:0] gray_out;

    // Bộ nhớ lưu ảnh Input
    reg [23:0] img_data [0:TOTAL_PIXELS-1];

    // Biến file và biến chạy vòng lặp
    integer out_file;
    integer i; // Biến dùng cho vòng lặp for

    // --- 2. KẾT NỐI MODULE (DUT) ---
    ConvertRGB uut (
        .r(r),
        .g(g),
        .b(b),
        .level(level),
        .gray_out(gray_out)
    );

    // --- 3. TẠO CLOCK ---
    initial clk = 0;
    always #5 clk = ~clk; // Chu kỳ 10ns

    // --- 4. CHƯƠNG TRÌNH CHÍNH  ---
    initial begin
        // A. KHỞI TẠO
        r = 0; g = 0; b = 0;
        level = 4'd2; // Chọn mức sáng

        // Load dữ liệu từ file Hex vào mảng RAM
        $readmemh("image_in.hex", img_data);

        // Mở file để ghi kết quả
        out_file = $fopen("image_out.hex", "w");
```

```
46
47          #20;
48
49          $display("--- BAT DAU XU LY %d PIXEL ---", TOTAL_PIXELS);
50
51          // ============================================================
52          // B. VÒNG LẶP FOR ĐỂ QUÉT ẢNH
53          // ============================================================
54          for (i = 0; i < TOTAL_PIXELS; i = i + 1) begin
55
56              // 1. Đồng bộ với Clock
57              @(posedge clk);
58
59              // 2. Đưa dữ liệu vào (Input Driving)
60              // Lấy 24-bit từ mảng, cắt ra R, G, B đưa vào module
61              r <= img_data[i][23:16];
62              g <= img_data[i][15:8];
63              b <= img_data[i][7:0];
64
65              // 3. Đợi kết quả (Output Sampling)
66              #1;
67
68              // 4. Ghi kết quả vào file
69              $fwrite(out_file, "%2h\n", gray_out);
70
71              // (Tùy chọn) In tiến độ mỗi khi xong 10% ảnh để đỡ sốt ruột
72              if (i % (TOTAL_PIXELS/10) == 0) begin
73                  $display("-> Da xu ly: %0d%%", (i * 100) / TOTAL_PIXELS);
74              end
75          end
76          // ============================================================
77
78          // C. KẾT THÚC
79          $display("--- HOAN THANH! ---");
80          $fclose(out_file); // Đóng file output
81          $finish;           // Dừng mô phỏng
82      end
83
84  endmodule
```

Lưu ý: Chuyển thư mục image_in.hex vào thư mục simulation/modelsim để chạy được testbench

3. Waveform chạy mô phỏng (Pre Simulation)



Lưu ý: File pic_output được sinh ra ở thư mục simulation/modelsim.

III. Chuyển ảnh từ file Hex về lại file jpg bằng python

ShowPic.py - E:\HDL_ALL\LAB2\BaiTap2\ShowPic.py (3.13.0)

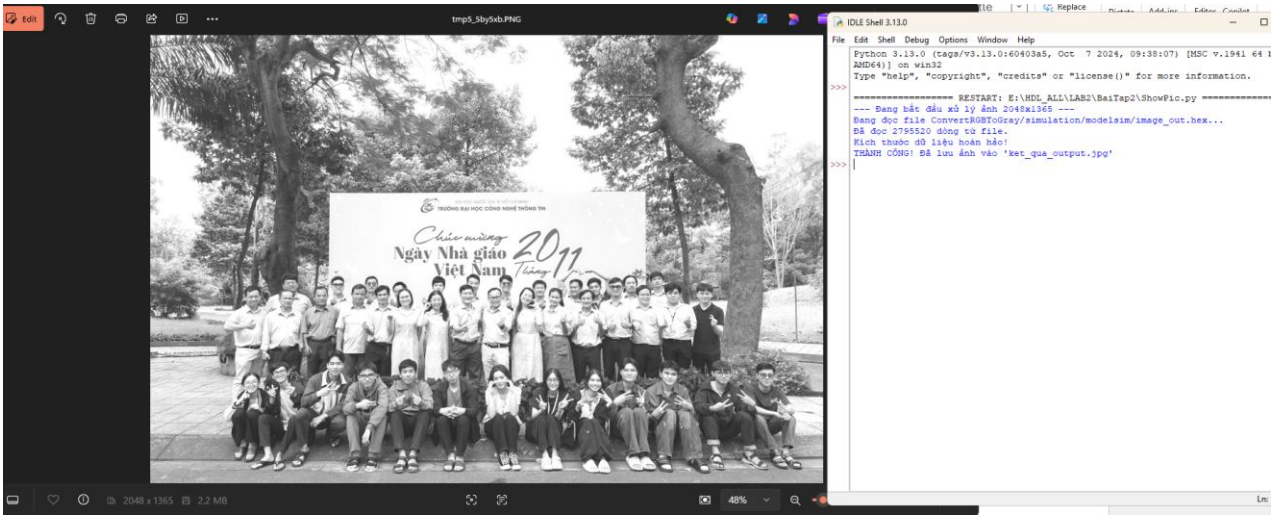
File Edit Format Run Options Window Help

```python
import numpy as np
from PIL import Image
import os

# ================ CẤU HÌNH ================
# 1. Kích thước ảnh (Phải khớp với ảnh gốc ban đầu)
WIDTH = 2048
HEIGHT = 1365

# 2. Tên file
INPUT_HEX_FILE = "ConvertRGBToGray/simulation/modelsim/image_out.hex"  # Tên file Verilog đã ghi ra
OUTPUT_IMG_FILE = "ket_qua_output.jpg"  # Tên file ảnh muốn lưu
# ==========================================

def main():
    print(f"--- Đang bắt đầu xử lý ảnh {WIDTH}x{HEIGHT} ---")

    # 1. Kiểm tra file tồn tại
    if not os.path.exists(INPUT_HEX_FILE):
        print(f"LỖI: Không tìm thấy file '{INPUT_HEX_FILE}'")
        print("Hãy kiểm tra lại xem Verilog đã chạy xong và ghi file chưa.")
        return

    # 2. Đọc dữ liệu từ file Hex
    print(f"Đang đọc file {INPUT_HEX_FILE}...")
    with open(INPUT_HEX_FILE, "r") as f:
        lines = f.readlines()

    print(f"Đã đọc {len(lines)} dòng từ file.")

    # 3. Chuyển đổi Hex sang số nguyên (Xử lý lỗi 'xx')
    pixel_data = []
    error_count = 0

    for line in lines:
        line = line.strip()
        # Bỏ qua dòng trống
        if not line:
            continue

        # Kiểm tra nếu dòng chứa 'x' (lỗi từ Verilog)
        if 'x' in line.lower() or 'z' in line.lower():
            pixel_data.append(0) # Gán màu đen cho điểm lỗi
            error_count += 1
        else:
            try:
                # Chuyển hex sang int
                val = int(line, 16)
                pixel_data.append(val)
            except ValueError:
                pixel_data.append(0)
                error_count += 1

    if error_count > 0:
        print(f"Cảnh báo: Có {error_count} pixels bị lỗi (xx) và đã được gán bằng 0.")

    # 4. Kiểm tra và điều chỉnh số lượng pixel
    expected_pixels = WIDTH * HEIGHT
    actual_pixels = len(pixel_data)
```

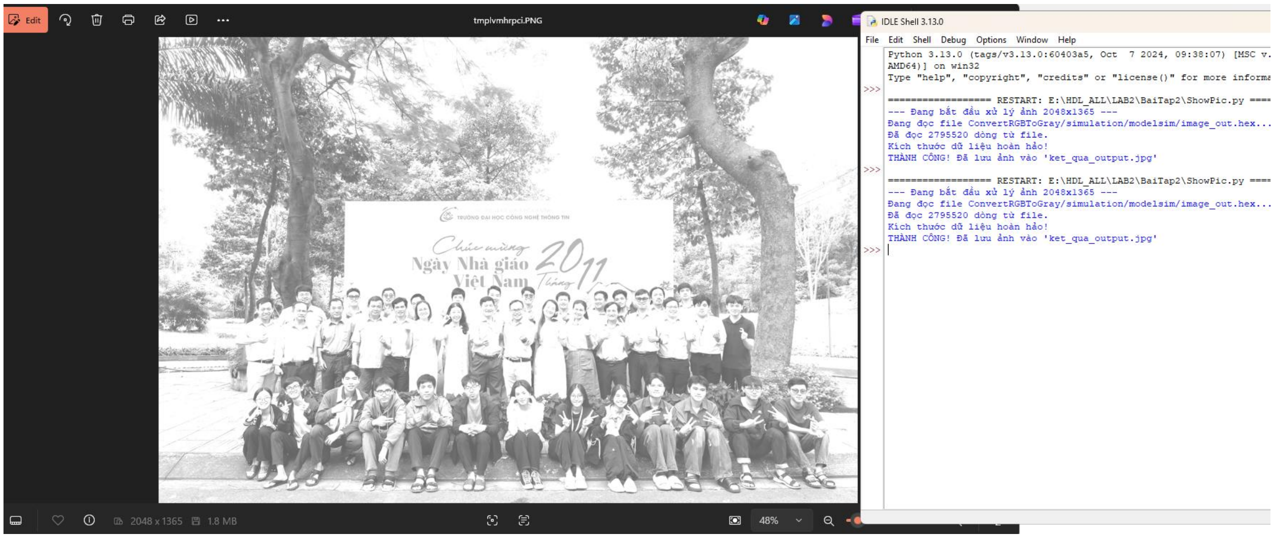
```
    # 4. Kiểm tra và điều chỉnh số lượng pixel
    expected_pixels = WIDTH * HEIGHT
    actual_pixels = len(pixel_data)

    if actual_pixels != expected_pixels:
        print(f"Lệch kích thước! Cần {expected_pixels}, thực tế có {actual_pixels}.")

        if actual_pixels > expected_pixels:
            print("-> Đang cắt bớt dữ liệu thừa...")
            pixel_data = pixel_data[:expected_pixels]
        else:
            print("-> Đang thêm dữ liệu trống (đen) vào cuối cho đủ...")
            pixel_data = pixel_data + [0] * (expected_pixels - actual_pixels)
    else:
        print("Kích thước dữ liệu hoàn hảo!")

    # 5. Tạo ảnh
    try:
        # Chuyển list thành mảng Numpy
        arr = np.array(pixel_data, dtype=np.uint8)

        # Reshape thành ma trận 2D (Hàng, Cột)
        matrix = arr.reshape((HEIGHT, WIDTH))

        # Tạo đối tượng ảnh từ mảng (Mode 'L' là Grayscale)
        img = Image.fromarray(matrix, 'L')

        # Hiển thị và lưu
        img.show()
        img.save(OUTPUT_IMG_FILE)
        print(f"THÀNH CÔNG! Đã lưu ảnh vào '{OUTPUT_IMG_FILE}'")

    except Exception as e:
        print("LỖI khi tạo ảnh:", e)

if __name__ == "__main__":
    main()
```

IV. Kết quả

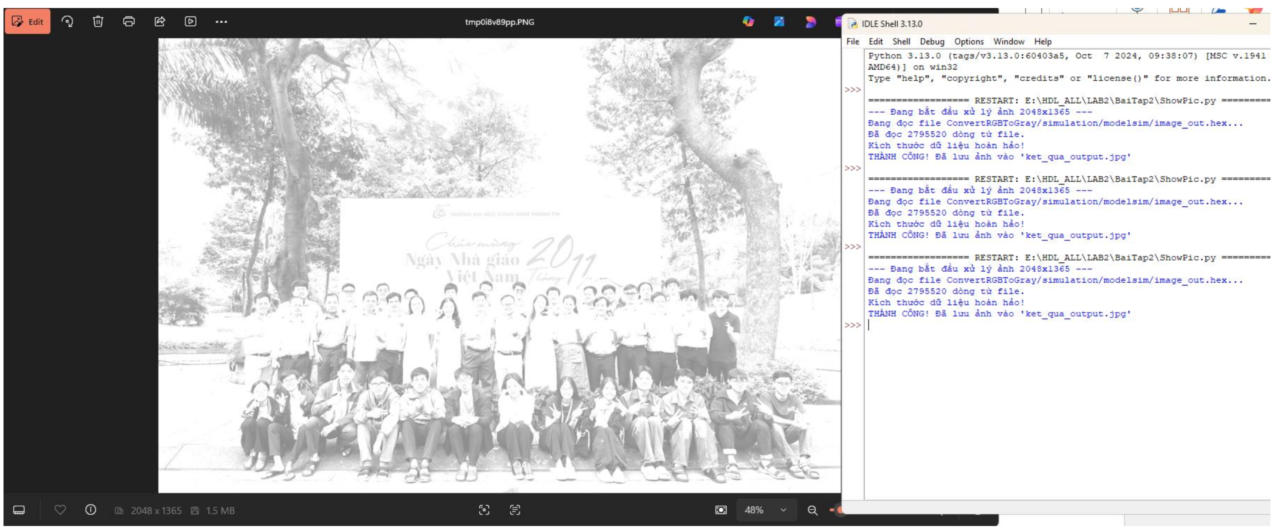
- Với level = 2



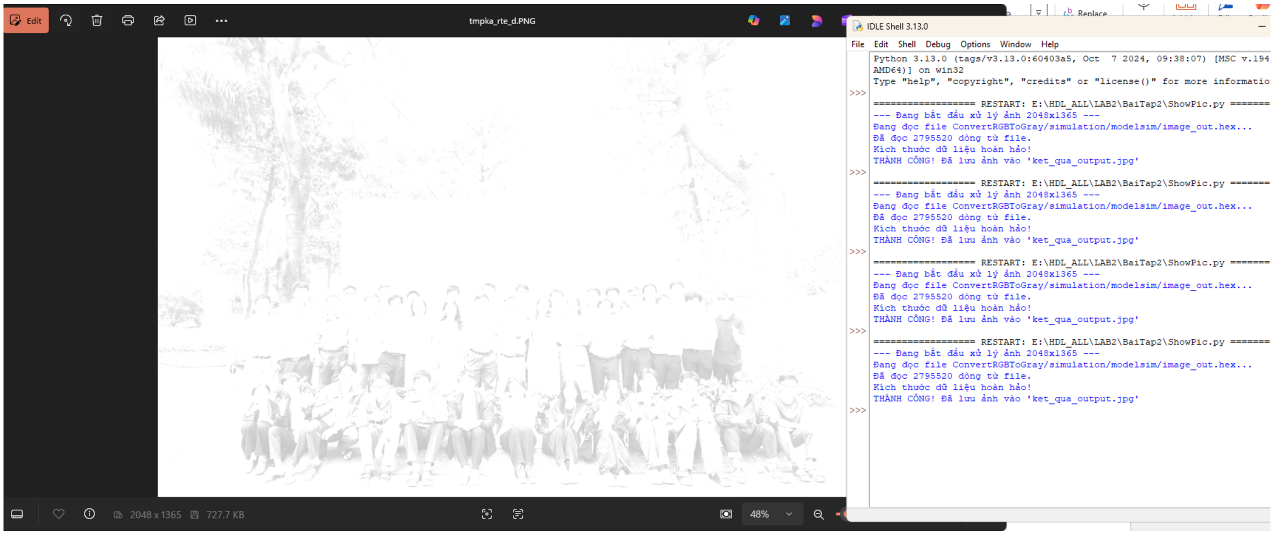
- Với level = 5



- Với level = 7



- Với level = 10



V. Source code

Đường dẫn: [Link github]