TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN, ĐHQG-HCM

KHOA KỸ THUẬT MÁY TÍNH

BÁO CÁO LAB2

ỨNG DỤNG VERILOG TRONG XỬ LÝ ẢNH

Môn học: CE213 - Thiết kế hệ thống số với HDL

Giảng viên hướng dẫn: Ngô Hiếu Trường

Thực hiện bởi:

1. Vũ Thành Lam 23520840

LAB2 ỨNG DỤNG VERILOG TRONG XỬ LÝ ẢNH 4
I. ĐỀ BÀI 4
1. Xây dựng bộ lọc trung vị (median filter) (không dùng for/while…) 4
2. Chuyển ảnh RGB sang ảnh grayscale (không dùng for/while…) 4
II. MỤC TIÊU BÀI LAB 5
III. KIẾN THỨC CẦN CHUẨN BỊ 5
IV. THIẾT KẾ VÀ TRIỂN KHAI PHẦN CỨNG 7
1. Xử lý nhiễu muối tiêu 7
1.1. Finite state machine cho mạch 7
1.2. Code matlab phục vụ việc chuyển đổi 8
1.3. Code verilog (design và testbench) 15
a) Module Controller 15
b) Module Datapath 19
c) Module parameter 25
d) Module top 25
e) Module testbench 27
f) Kết quả compile trên quartus 28
g) Kết quả waveform 28
2. Chuyển ảnh RGB sang ảnh grayscale 32
2.1. Finite state machine cho mạch 32
2.2. Code matlab phục vụ việc chuyển đổi 32
a) Code chuyển jpg sang hex. 32
b) Code chuyển hex sang jpg 35
2.3. Code verilog (design , testbench, kết quả) 37
a) Module controller 37
b) Module BGR_to_Gray 39
c) Module Datapath 39
d) Module Top 42
e) Module testbench 44
f) Kết quả chạy synthesis trên quartus. 46

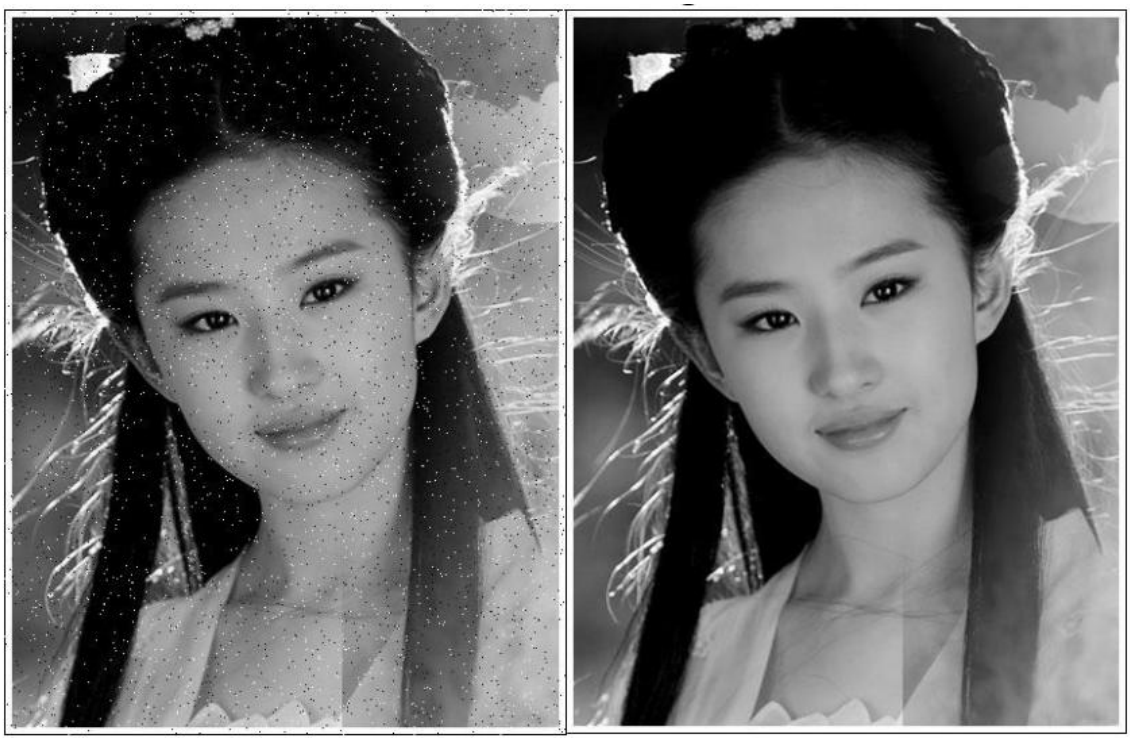
g) Kết quả mô phỏng waveform 46
h) Gen ảnh ra từ file 1 lần nữa. 48
V. KẾT QUẢ VÀ NHẬN XÉT 48
1. Xữ lý nhiễu muối tiêu 48
2. Chuyển ảnh RGB sang ảnh grayscale 48
VI. TÀI LIỆU THAM KHẢO 48

LAB2 ỨNG DỤNG VERILOG TRONG XỬ LÝ ẢNH

I. ĐỀ BÀI

1. Xây dựng bộ lọc trung vị (median filter) (không dùng for/while…)

a) Sinh viên thực hiện trên matlab bộ lọc trung vị



Hình bên trái là ảnh đầu vào, hình bên phải là ảnh sau khi lọc

b) Sinh viên tìm hiểu thuật toán lọc trung vị.
c) Chuyển ảnh grayscale sang ảnh file pic_input.txt.
d) Viết code verilog đọc file pic_input.txt và hiện thực thuật toán lọc trung vị. Kết quả xuất ra file pic_output.txt.
e) Sử dụng matlab/python để chuyển ảnh từ pic_output.txt thành ảnh grayscale và so sánh theo định tính. Thuật toán verilog tốt sinh viên sẽ nhận được được ảnh như hình bên phải.
f) Đánh giá ảnh tái tạo với ảnh gốc qua 2 thông số: PSNR, SSIM (dùng python/matlab)

2. Chuyển ảnh RGB sang ảnh grayscale (không dùng for/while…)

a) Chuyển ảnh RGB sang ảnh bitmap bằng matlab/python rồi xử lý trên verilog
b) Sử dụng công thức chuẩn :

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

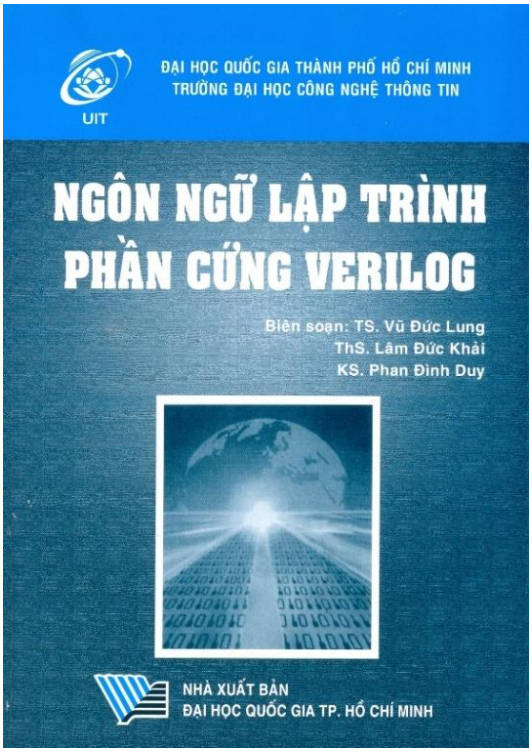
c) Phải có tham số chỉnh độ sáng trong Verilog code

II. MỤC TIÊU BÀI LAB

- Giúp sinh viên cũng cố kiến thức cơ bản về HDL – verilog: module, always, blocking ,…
- Sử dụng chức năng *Pre-simulation và Post-Simulaiton.*
- Vận dụng Verilog để hiện thực các thuật toán xử lý ảnh.

III. KIẾN THỨC CẦN CHUẨN BỊ

- Sinh viên sử dụng tài liệu/giáo trình được học ở lớp lý thuyết :

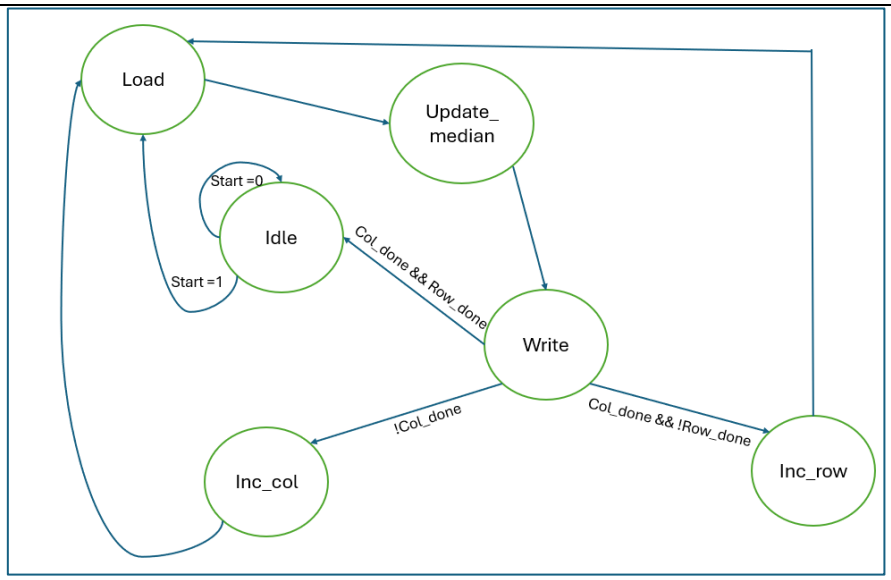


- Matlab/python để chuyển ảnh sang grayscale/hoặc các phần phềm có cùng chức năng :

IV. THIẾT KẾ VÀ TRIỂN KHAI PHẦN CỨNG

1. Xử lý nhiễu muối tiêu

1.1. Finite state machine cho mạch

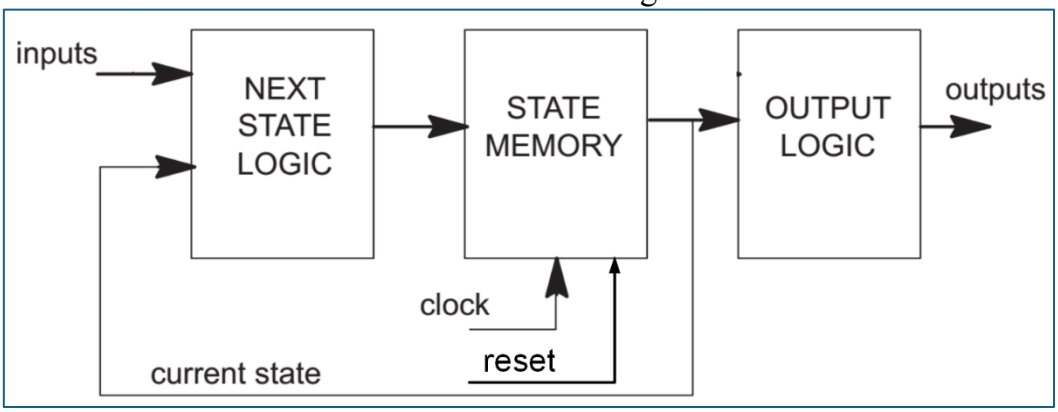


Moore_FSM

Bảng giải thích chức năng và ý nghĩa của từng state

State	Chức năng và ý nghĩa
Idle	Không làm gì, chỉ nạp giá trị 0 cho 2 thanh ghi Row và Col.
Load	Load giá trị window 3x3 vào 9 thanh ghi 8 bit.
Update_median	Lấy giá trị 9 thanh ghi vừa load trước đó để tính ra giá trị median bằng cách sort và cho phép thanh thế vào thanh ghi trung vị. Thanh ghi thứ 5. Phần tử chính giữa của ma trận 3x3
Write	Cập nhật giá trị từ thanh ghi trung vị median vào bộ nhớ buffer.
Inc_col	Tăng giá trị của thanh ghi Col lên 1 đơn vị.
Inc_row	Tăng giá trị của thanh ghi Row lên 1 đơn vị và cho thanh ghi Col về 0.

Kiến trúc FSM - controller gồm 3 khối



1.2. Code matlab phục vụ việc chuyển đổi

a) Code chuyển đổi jpg sang hex

```
% Đọc ảnh grayscale
img = imread('baitap1_anhgoc.jpg');

if size(img, 3) == 3
    img = rgb2gray(img);
end
[rows, cols] = size(img);
hex_array = cell(rows, cols);
for i = 1:rows
    for j = 1:cols
        pixel_value = img(i, j);
        hex_array{i, j} = dec2hex(pixel_value, 2); % 2 ký tự hex
    end
end
disp('Pixel [1,1]:');
fprintf('Decimal: %d, Hex: %s\n', img(1,1), hex_array{1,1});

disp('Pixel [1,2]:');
fprintf('Decimal: %d, Hex: %s\n', img(1,2), hex_array{1,2});
```

```
fid = fopen('anhgoc.txt', 'w');
for i = 1:rows
    for j = 1:cols
        fprintf(fid, '%s ', hex_array{i, j});
    end
    fprintf(fid, '\n');
end
fclose(fid);
disp('Done! Saved to anhgoc.txt');
```

- **Ảnh nhiễu :**

- Link file pic_input.txt chuyển từ ảnh nhiễu sang : https://drive.google.com/file/d/1u-e2QZvvVg9mfPdAR2eKbbXpewdZe9YX/view?usp=drive_link

b) Code chuyển đổi hex sang jpg

```
function fast_writememh_to_jpg(input_file, output_file, width, height)
    % Đọc file nhanh bằng textscan
    fid = fopen(input_file, 'r');

    % Đọc tất cả các giá trị hex
    data = textscan(fid, '%s', 'CommentStyle', '//');
    fclose(fid);

    % Chuyển hex sang decimal
    hex_values = data{1};
```

```
    pixels = cellfun(@hex2dec, hex_values);

    fprintf('Read %d pixels\n', length(pixels));

    % Reshape
    img = reshape(pixels(1:width*height), [width, height])';
    img = uint8(img);

    % Lưu và hiển thị
    imwrite(img, output_file, 'jpg', 'Quality', 95);
    figure; imshow(img);
    title(sprintf('%s (%dx%d)', output_file, height, width));

    fprintf('✓ Saved: %s\n', output_file);
end

% Sử dụng
fast_writememh_to_jpg('out_2.txt', 'test2.jpg', 2048, 1365);
```

- **Link file pic_output.txt sinh ra từ code verilog xử lý ảnh của em :** https://drive.google.com/file/d/1kMpe6n7JnSZUJKi7-IkqQAF-F9VXtYYp/view?usp=drive_link

- Ảnh gen ra từ file txt :

c) Code đánh giá 2 thông số PSNR, SSIM.

```
function [psnr_value, ssim_value] = evaluate_image(original_img,
reconstructed_img)
    % Đánh giá ảnh tái tạo với ảnh gốc qua PSNR và SSIM
    % original_img: ảnh gốc (path hoặc matrix)
    % reconstructed_img: ảnh tái tạo (path hoặc matrix)

    % Đọc ảnh nếu là đường dẫn
    if ischar(original_img)
        img1 = imread(original_img);
    else
        img1 = original_img;
    end

    if ischar(reconstructed_img)
        img2 = imread(reconstructed_img);
    else
        img2 = reconstructed_img;
    end

    % Chuyển sang grayscale nếu là ảnh màu
    if size(img1, 3) == 3
        img1 = rgb2gray(img1);
    end
    if size(img2, 3) == 3
        img2 = rgb2gray(img2);
    end

    % Đảm bảo cùng kích thước
    if ~isequal(size(img1), size(img2))
        error('Hai ảnh phải có cùng kích thước!');
    end

    % Chuyển sang double
    img1 = double(img1);
    img2 = double(img2);

    % Tính PSNR
    mse = mean((img1(:) - img2(:)).^2);
    if mse == 0
        psnr_value = Inf;
    else
        max_pixel = 255.0;
        psnr_value = 20 * log10(max_pixel / sqrt(mse));
    end

    % Tính SSIM
```

```
ssim_value = ssim(uint8(img1), uint8(img2));

% Hiển thị kết quả
fprintf('=== ĐÁNH GIÁ CHẤT LƯỢNG ẢNH ===\n');
fprintf('PSNR: %.4f dB\n', psnr_value);
fprintf('SSIM: %.4f\n', ssim_value);
fprintf('\n');

% Phân loại chất lượng
fprintf('Đánh giá PSNR:\n');
if psnr_value >= 40
   fprintf('  → Xuất sắc (≥40 dB)\n');
elseif psnr_value >= 30
   fprintf('  → Tốt (30-40 dB)\n');
elseif psnr_value >= 20
   fprintf('  → Trung bình (20-30 dB)\n');
else
   fprintf('  → Kém (<20 dB)\n');
end

fprintf('\nĐánh giá SSIM:\n');
if ssim_value >= 0.95
   fprintf('  → Xuất sắc (≥0.95)\n');
elseif ssim_value >= 0.85
   fprintf('  → Tốt (0.85-0.95)\n');
elseif ssim_value >= 0.70
   fprintf('  → Trung bình (0.70-0.85)\n');
else
   fprintf('  → Kém (<0.70)\n');
end

% Hiển thị ảnh so sánh
figure('Position', [100, 100, 1200, 400]);

subplot(1, 3, 1);
imshow(uint8(img1));
title('Ảnh gốc');

subplot(1, 3, 2);
imshow(uint8(img2));
title('Ảnh tái tạo');

subplot(1, 3, 3);
diff_img = abs(img1 - img2);
imshow(diff_img, []);
title(sprintf('Sai số\nPSNR: %.2f dB, SSIM: %.4f', psnr_value, ssim_value));
```

```
    colormap('jet');
    colorbar;

    % Histogram so sánh
    figure('Position', [100, 600, 1200, 300]);

    subplot(1, 2, 1);
    hold on;
    histogram(img1(:), 50, 'FaceColor', 'b', 'FaceAlpha', 0.5);
    histogram(img2(:), 50, 'FaceColor', 'r', 'FaceAlpha', 0.5);
    legend('Ảnh gốc', 'Ảnh tái tạo');
    title('Histogram so sánh');
    xlabel('Giá trị pixel');
    ylabel('Số lượng');
    hold off;

    subplot(1, 2, 2);
    histogram(diff_img(:), 50, 'FaceColor', 'g');
    title('Histogram sai số');
    xlabel('Sai số pixel');
    ylabel('Số lượng');
end

% Sử dụng
[psnr, ssim_val] = evaluate_image('baitap1_anhgoc.jpg', 'output.jpg');
```

- **Thông số PSNR, SSIM khi so sánh giữa ảnh từ verilog với ảnh gốc :**

```
>> compare_anh
=== ĐÁNH GIÁ CHẤT LƯỢNG ẢNH ===
PSNR: 33.2088 dB
SSIM: 0.9581

Đánh giá PSNR:
  → Tốt (30-40 dB)

Đánh giá SSIM:
  → Xuất sắc (≥0.95)
```

- Ảnh gốc :

- Ảnh xử lý nhiễu :

1.3. Code verilog (design và testbench)

a) Module Controller

```
module controller
(
   input wire clk, rst, start, Col_done,Row_done,
   output reg [1:0] Col_ctrl, Row_ctrl, median_ctrl, data_ctrl, enable_wr,
   output reg [2:0] State
);
   localparam  Idle = 3'd0,
               Load = 3'd1,
               Update_median = 3'd5,
               Write = 3'd2,
               Inc_col = 3'd3,
               Inc_row = 3'd4;
   reg [2:0] cur_state,n_state;

   // reg
   always @(posedge clk or posedge rst)
   begin
      if(rst)
         cur_state <= Idle;
      else
         cur_state <= n_state;
   end


   //Next_state update
   always @(*)
   begin
      case(cur_state)
         Idle : n_state = (start) ? Load : Idle;
         Load : n_state = Update_median ;
         Update_median : n_state = Write;
         Write : begin
            if(Col_done && Row_done)
               n_state = Idle;
            else if(Col_done)
               n_state = Inc_row;
            else
               n_state = Inc_col;
         end
         Inc_col : n_state = Load;
         Inc_row : n_state = Load;
         default : n_state = Idle;
```

```
    endcase
  end

  always @(*)
  begin
    State = cur_state;
    Row_ctrl = 2'd0;
    Col_ctrl = 2'd0;
    data_ctrl = 2'd0;
    median_ctrl = 2'd0;
    enable_wr = 2'd0;
   // update_memory = 2'd0;
    case(cur_state)
      Idle : begin
        Row_ctrl = 2'd1;
        Col_ctrl= 2'd1;
      end
      Load : begin
        data_ctrl = 2'd1; //load
        median_ctrl = 2'd1;// load
      end
      Update_median : begin
        median_ctrl = 2'd2;//Change

      end
      Write : begin
        enable_wr = 2'd1 ; // cho phep ghi vo memory
      end
      Inc_col : begin
        Col_ctrl = 2'd2 ; // increase 1
      end
      Inc_row : begin
        Row_ctrl = 2'd2 ; // increase 1
        Col_ctrl = 2'd1; // reload 0
      end
    endcase
  end


endmodule
```

- Module controller chỉ đơn giản là được thiết kế dựa trên fsm đã được trình bày ở phần trước đó .
- Tín hiệu Start cho phép chương trình bắt đầu
- Tín hiệu rst cho phép reset lại hệ thống.

- Tín hiệu Col_done = 1 khi nó đã duyệt hết cột của dòng đó . Tức là thanh ghi Col = Width(430) – 3, do xét từ index 0 nên chỉ xét đến 429, và xét 1 lần là 3 cột nên phải trừ 2, 429 -2 tức 430 -3 .

```
assign Col_done = (Col == (WIDTH - 3 )) ? 1 : 0;
```

- Tín hiệu Row _done =1 khi nó đã duyệt hết hàng của ma trận pixel. Tương tự với Col done :

```
assign Row_done = (Row ==(HEIGHT - 3)) ? 1 : 0;
```

Bảng output signal dùng để điều khiển các khối trong datapath

Output signals	Ý nghĩa
Row_ctrl	Chọn chế độ ghi vô thanh ghi Row
Col_ctrl	Chọn chế độ ghi vô thanh ghi Col
data_ctrl	Chọn chế độ ghi vô 8 thanh ghi biên của window3x3
median_ctrl	Chọn chế độ ghi vô thanh ghi trung tâm của window3x3
enable_wr	Cho phép cập nhật giá trị median vào bộ buffer.

Bảng output signal theo từng State của Fsm

State	Row_Ctrl	Col_Ctrl	data_ctrl	median_Ctrl	enable_wr
Idle	2'd1	2'd1	0	0	0
Load	0	0	2'd1	2'd1	0
Update_median	0	0	0	2'd2	0
Write	0	0	0	0	2'd1
Inc_col	0	2'd2	0	0	0
Inc_row	2'd2	2'd1	0	0	0

Giải thích ý nghĩa của từng signal control

1. Row_ctrl

```
mux4to1_10 a0
(
    .a(Row),
    .b(10'd0),
    .c(Row + 10'd1),
    .d(10'd0),
    .sel(Row_ctrl),
    .out(Row_in)
);
```

- Trên đây là bộ mux4to1_10bit chọn chế độ ghi cho reg Row.
- Row_ctrl là 2 bit select :
  - 2'd0 : Row giữ nguyên giá trị.
  - 2'd1 : Row nạp giá trị 0.

- 2’d2 : Row <= Row + 1. Tăng 1 để duyệt hàng kế.
- 2’d3 : Row nạp giá trị 0.

2. Col_ctrl

```
mux4to1_10 a1(
    .a(Col),
    .b(10'd0),
    .c(Col + 10'd1),
    .d(10'd0),
    .sel(Col_ctrl),
    .out(Col_in)
);
```

- Tương tự như Row_ctrl.

3. Data_ctrl

```
mux4to1_8 a2(
    .a(arr[0]),
    .b(total_memory[Row * WIDTH + Col]),
    .c(8'd0),
    .d(8'd0),
    .sel(data_ctrl),
    .out(arr_in[0])
);
```

- Data-ctrl :
  - 2’d0 : giá trị của 8 thanh ghi giữ nguyên.
  - 2’d1 : Nạp giá trị theo row và col từ memory chính.
  - 2’d2 : Nạp 0.
  - 2’d3 : Nạp 0.

4. Median_ctrl

```
mux4to1_8 a6(
    .a(arr[4]),
    .b(total_memory[Row * WIDTH + Col + WIDTH + 1]),
    .c(median),
    .d(8'd0),
    .sel(median_ctrl),
    .out(arr_in[4])
);
```

- Median_Ctrl :
  - 2’d0 : giá trị thanh ghi trung tâm này giữ nguyên.
  - 2’d1 : Nạp giá trị từ bộ memory chính.
  - 2’d2 : Cập nhật giá trị cho thanh ghi theo bộ lọc median.
  - 2’d3 : Nạp 0.

5. Enable_wr:

```
always @(posedge clk) begin
    if (enable_wr) begin
        tmp_mem[Row * WIDTH + Col + WIDTH + 1] <= median;
    end
end
```

- Cho phép cập nhật giá trị từ thanh ghi median vô bộ đệm buffer.

b) Module Datapath

- Module median9 :

```
module median9(
   input  wire [7:0] a0,a1,a2,a3,a4,a5,a6,a7,a8,
   output wire [7:0] median
);

   reg [7:0] x[0:8];

   task sort2;
      inout [7:0] p, q;
      reg [7:0] t;
      begin
         if (p > q) begin
            t = p; p = q; q = t;
         end
      end
   endtask

   always @(*) begin
      x[0]=a0; x[1]=a1; x[2]=a2;
      x[3]=a3; x[4]=a4; x[5]=a5;
      x[6]=a6; x[7]=a7; x[8]=a8;

      sort2(x[1],x[2]);
      sort2(x[4],x[5]);
      sort2(x[7],x[8]);

      sort2(x[0],x[1]);
      sort2(x[3],x[4]);
      sort2(x[6],x[7]);

      sort2(x[1],x[2]);
      sort2(x[4],x[5]);
      sort2(x[7],x[8]);

      sort2(x[0],x[3]);
```

```
            sort2(x[5],x[8]);
            sort2(x[4],x[7]);

            sort2(x[3],x[6]);
            sort2(x[1],x[4]);
            sort2(x[2],x[5]);

            sort2(x[4],x[7]);
            sort2(x[4],x[2]);
            sort2(x[6],x[4]);
            sort2(x[4],x[2]);
        end

        assign median = x[4];
endmodule
```

- Module sử dụng kiến trúc như sorting network.
- Module median9 không thực hiện sắp xếp toàn bộ 9 phần tử, mà sử dụng mạng so sánh–hoán đổi (comparison network) được thiết kế sẵn để đảm bảo chỉ phần tử ở vị trí giữa (x[4]) là trung vị.
- Nó không sắp xếp 9 phần tử mà chỉ đảm bảo vị trí x[4] có 4 phần tử lớn hơn ở sau nó và 4 phần tử bé hơn ở trước nó.

Datapath :

```
    `include "parameter.v"
    module datapath #(
        parameter   WIDTH = 430,
                    HEIGHT = 554,
                    INFILE = "pic_input.txt",
                    OUTFILE = "pic_output.txt"
    )(
        input wire [1:0]  Col_ctrl, Row_ctrl, median_ctrl, data_ctrl,
    enable_wr,
        input wire clk,
        output wire  Row_done,Col_done,
        output [9:0] Row_o,
        output [9:0] Col_o,
        output [7:0] D0,D1,D2,D3,D4,D5,D6,D7,D8
    );

        parameter sizeOfLengthReal = 238220;                    // image data
    430*554 = 238220

        reg [7 : 0]   total_memory [0 : sizeOfLengthReal-1];
        reg [7:0] tmp_mem [0 : sizeOfLengthReal-1];
        initial begin
```

```
    $readmemh(INFILE,total_memory,0,sizeOfLengthReal-1); // read
file from INFILE
    $readmemh(INFILE,tmp_mem,0,sizeOfLengthReal-1);
    end


    reg [9:0] Row;
    reg [9:0] Col;
    wire [9:0] Row_in;
    wire [9:0] Col_in;

    reg [7:0] arr [0:8];
    wire [7:0] arr_in[0:8];

    mux4to1_10 a0
    (
        .a(Row),
        .b(10'd0),
        .c(Row + 10'd1),
        .d(10'd0),
        .sel(Row_ctrl),
        .out(Row_in)
    );

    mux4to1_10 a1(
        .a(Col),
        .b(10'd0),
        .c(Col + 10'd1),
        .d(10'd0),
        .sel(Col_ctrl),
        .out(Col_in)
    );

    mux4to1_8 a2(
        .a(arr[0]),
        .b(total_memory[Row * WIDTH + Col]),
        .c(8'd0),
        .d(8'd0),
        .sel(data_ctrl),
        .out(arr_in[0])
    );

    mux4to1_8 a3(
        .a(arr[1]),
        .b(total_memory[Row * WIDTH + Col + 1]),
        .c(8'd0),
```

```
    .d(8'd0),
    .sel(data_ctrl),
    .out(arr_in[1])
);

 mux4to1_8 a4(
    .a(arr[2]),
    .b(total_memory[Row * WIDTH + Col + 2]),
    .c(8'd0),
    .d(8'd0),
    .sel(data_ctrl),
    .out(arr_in[2])
);


 mux4to1_8 a5(
    .a(arr[3]),
    .b(total_memory[Row * WIDTH + Col + WIDTH]),
    .c(8'd0),
    .d(8'd0),
    .sel(data_ctrl),
    .out(arr_in[3])
);
wire [7:0] median;
median9 a7 (
    .a0(arr[0]),
    .a1(arr[1]),
    .a2(arr[2]),
    .a3(arr[3]),
    .a4(arr[4]),
    .a5(arr[5]),
    .a6(arr[6]),
    .a7(arr[7]),
    .a8(arr[8]),
    .median(median)
);


 mux4to1_8 a6(
    .a(arr[4]),
    .b(total_memory[Row * WIDTH + Col + WIDTH + 1]),
    .c(median),
    .d(8'd0),
    .sel(median_ctrl),
    .out(arr_in[4])
);
```

```
  mux4to1_8 a9(
    .a(arr[5]),
    .b(total_memory[Row * WIDTH + Col + WIDTH + 2]),
    .c(8'd0),
    .d(8'd0),
    .sel(data_ctrl),
    .out(arr_in[5])
  );
  mux4to1_8 a8(
    .a(arr[6]),
    .b(total_memory[Row * WIDTH + Col + WIDTH*2]),
    .c(8'd0),
    .d(8'd0),
    .sel(data_ctrl),
    .out(arr_in[6])
  );
   mux4to1_8 a10(
    .a(arr[7]),
    .b(total_memory[Row * WIDTH + Col + WIDTH*2 + 1]),
    .c(8'd0),
    .d(8'd0),
    .sel(data_ctrl),
    .out(arr_in[7])
  );

  mux4to1_8 a11(
    .a(arr[8]),
    .b(total_memory[Row * WIDTH + Col + WIDTH*2 + 2]),
    .c(8'd0),
    .d(8'd0),
    .sel(data_ctrl),
    .out(arr_in[8])
  );




 assign Row_done = (Row ==(HEIGHT - 3)) ? 1 : 0;
 assign Col_done = (Col == (WIDTH - 3 )) ? 1 : 0;

/* integer infile, outfile;
 initial begin
    outfile = $fopen(OUTFILE, "w");
 end*/
```

```
    always @(posedge clk) begin
        if (enable_wr) begin
            tmp_mem[Row * WIDTH + Col + WIDTH + 1] <= median;
        end
    end


    always @(posedge clk) begin
        if (Row_done && Col_done ) begin
            $writememh(OUTFILE,tmp_mem,0,sizeOfLengthReal-1);
        end
    end

    always @(posedge clk) begin
        Row <= Row_in;
        Col <= Col_in;
        arr[0] <= arr_in[0];
        arr[1] <= arr_in[1];
        arr[2] <= arr_in[2];
        arr[3] <= arr_in[3];
        arr[4] <= arr_in[4];
        arr[5] <= arr_in[5];
        arr[6] <= arr_in[6];
        arr[7] <= arr_in[7];
        arr[8] <= arr_in[8];
    end

    // Assign outputs
    assign Row_o = Row;
    assign Col_o = Col;
    assign D0 = arr[0];
    assign D1 = arr[1];
    assign D2 = arr[2];
    assign D3 = arr[3];
    assign D4 = arr[4];
    assign D5 = arr[5];
    assign D6 = arr[6];
    assign D7 = arr[7];
    assign D8 = arr[8];


endmodule
```

- Đa số các thành phần của datapath như mux và reg quan trọng đã được trình bày ở phần của controller .
- Bên cạnh đó có dùng chức năng readmemh để đọc file txt vào bộ nhớ

```
reg [7 : 0]   total_memory [0 : sizeOfLengthReal-1];
reg [7:0] tmp_mem [0 : sizeOfLengthReal-1];
initial begin
$readmemh(INFILE,total_memory,0,sizeOfLengthReal-1); // read file from INFILE
$readmemh(INFILE,tmp_mem,0,sizeOfLengthReal-1);
end
```

- Tạo 2 mảng thanh ghi này giống nhau lúc ban đầu,total_memory thì chỉ có read_only sau khi đọc từ filetxt. Còn tmp_mem sẽ được update các giá trị median từ việc tính toán.
- Và khi duyệt xong tất cả các hàng và cột, tức là mảng tmp- mem đã hoàn thành thì sẽ writememh vào file txt :

```
always @(posedge clk) begin
    if (Row_done && Col_done ) begin
        $writememh(OUTFILE,tmp_mem,0,sizeOfLengthReal-1);
    end
end
```

c) Module parameter

```
`define INPUTFILENAME          "pic_input.txt" // Input file name
`define OUTPUTFILENAME         "pic_output2.txt"          // Output
```

- Module này chỉ đơn giản khai báo file in và file out.
- Lưu ý : để được modelsim xử lý thì 2 file này cần nằm cùng thư mục với project modelsim.
- Pic_output2 là vì em đã chạy thêm 1 lần nữa coi nó có ra giống cái pic_output không.

d) Module top

```
`include "parameter.v"
module top #(
   parameter   WIDTH = 430,
               HEIGHT = 554,
               INFILE = "pic_input.txt",
               OUTFILE = "pic_output.txt"
)(
   input wire clk,
   input wire rst,
   input wire start,
   output wire [2:0] State,
   output wire [9:0] Row_o,
   output wire [9:0] Col_o,
   output wire [7:0] D0, D1, D2, D3, D4, D5, D6, D7, D8,
   output wire done
```

```
);

    // ========== INTERNAL SIGNALS ==========
    wire [1:0] Col_ctrl;
    wire [1:0] Row_ctrl;
    wire [1:0] median_ctrl;
    wire [1:0] data_ctrl;
    wire [1:0] enable_wr;

    wire Col_done;
    wire Row_done;

    // Done signal
    assign done = (State == 3'd0) && (Row_done && Col_done);

    // ========== CONTROLLER INSTANCE ==========
    controller u_controller (
        .clk(clk),
        .rst(rst),
        .start(start),
        .Col_done(Col_done),
        .Row_done(Row_done),
        .Col_ctrl(Col_ctrl),
        .Row_ctrl(Row_ctrl),
        .median_ctrl(median_ctrl),
        .data_ctrl(data_ctrl),
        .enable_wr(enable_wr),
        .State(State)
    );

    // ========== DATAPATH INSTANCE ==========
    datapath #(
        .WIDTH(WIDTH),
        .HEIGHT(HEIGHT),
        .INFILE(INFILE),
        .OUTFILE(OUTFILE)
    ) u_datapath (
        .clk(clk),
        .Col_ctrl(Col_ctrl),
        .Row_ctrl(Row_ctrl),
        .median_ctrl(median_ctrl),
        .data_ctrl(data_ctrl),
        .enable_wr(enable_wr),
        .Row_o(Row_o),
        .Col_o(Col_o),
        .D0(D0),
```

```
        .D1(D1),
        .D2(D2),
        .D3(D3),
        .D4(D4),
        .D5(D5),
        .D6(D6),
        .D7(D7),
        .D8(D8),
        .Row_done(Row_done),
        .Col_done(Col_done)
    );


endmodule
```

- Module top này chỉ đơn giản là nối module controller với datapath.

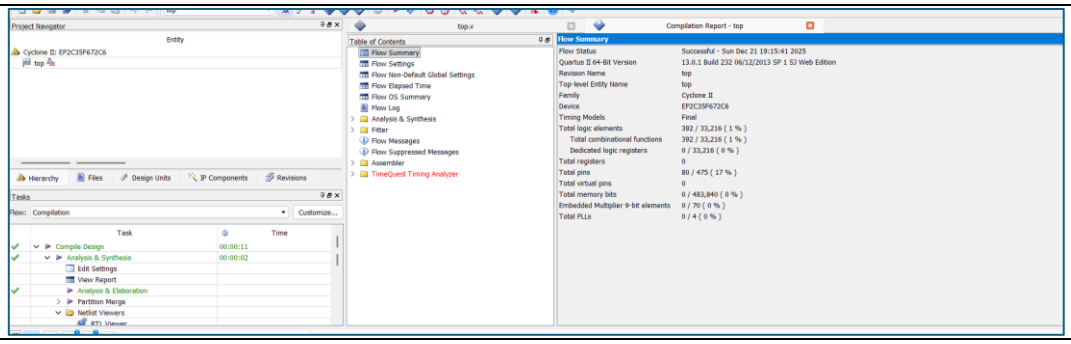
e) Module testbench

```
`timescale 1ns/1ps
`include "parameter.v"
module tb_median_filter;

    reg clk, rst, start;
    wire [2:0] State;
    wire [9:0] Row_o;
    wire [9:0] Col_o;
    wire [7:0] D0, D1, D2, D3, D4, D5, D6, D7, D8;
    wire done;

    // DUT
    top #(
        .INFILE(`INPUTFILENAME),
        .OUTFILE(`OUTPUTFILENAME)
    ) u_top (
        .clk(clk), .rst(rst), .start(start),
        .State(State), .Row_o(Row_o), .Col_o(Col_o),
        .D0(D0), .D1(D1), .D2(D2), .D3(D3), .D4(D4),
        .D5(D5), .D6(D6), .D7(D7), .D8(D8), .done(done)
    );

    // Clock
    initial begin
        clk = 0;
        forever #1 clk = ~clk;
    end
```

```
// Test
initial begin
    rst = 1;
    start = 0;
    #5 rst = 0;
    #2 start = 1;
    #2 start = 0;
end

endmodule
```

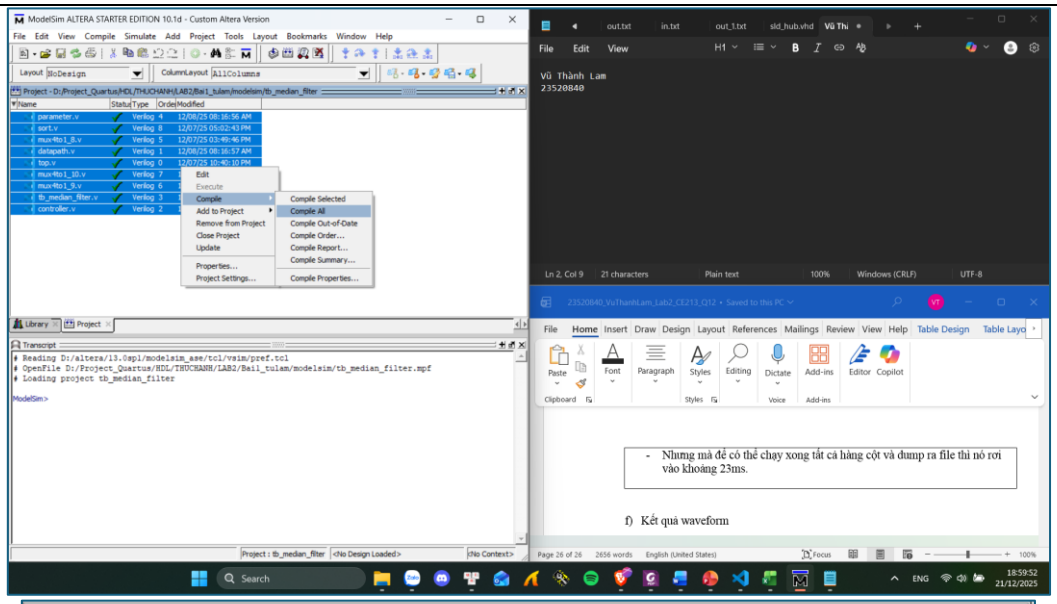
- Module này cũng chỉ khởi tạo clk.
- Cho rst lúc đầu và cho start =1 để mạch chạy.
- Đặc điểm tb này không có finish vì em sẽ dùng run -xns trên modelsim để muốn chạy bao nhiêu cũng được.
- Nhưng mà để có thể chạy xong tất cả hàng cột và dump ra file thì nó rơi vào khoảng 18ms.

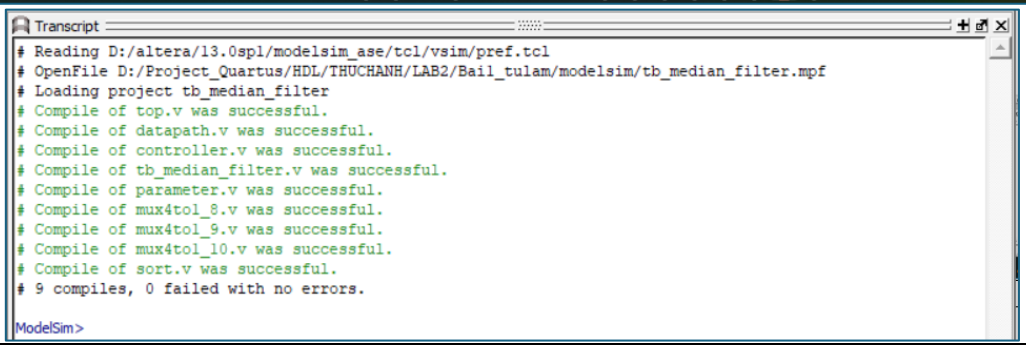
f) Kết quả compile trên quartus



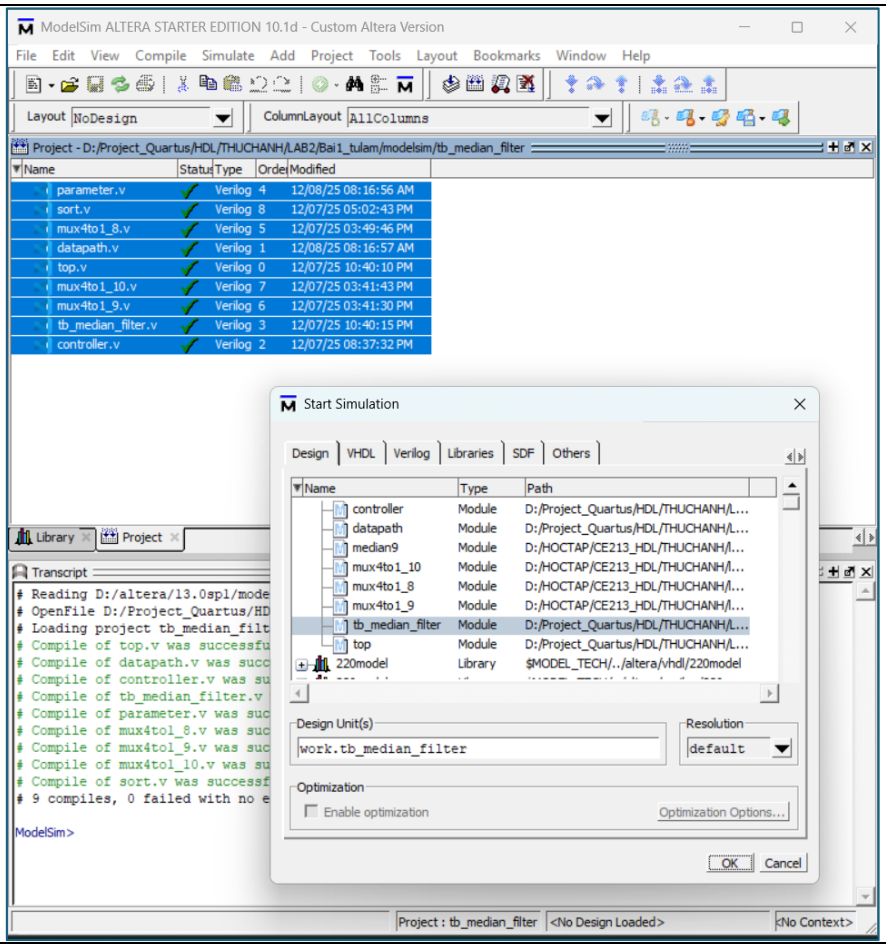
g) Kết quả waveform

Compile tất cả các file:





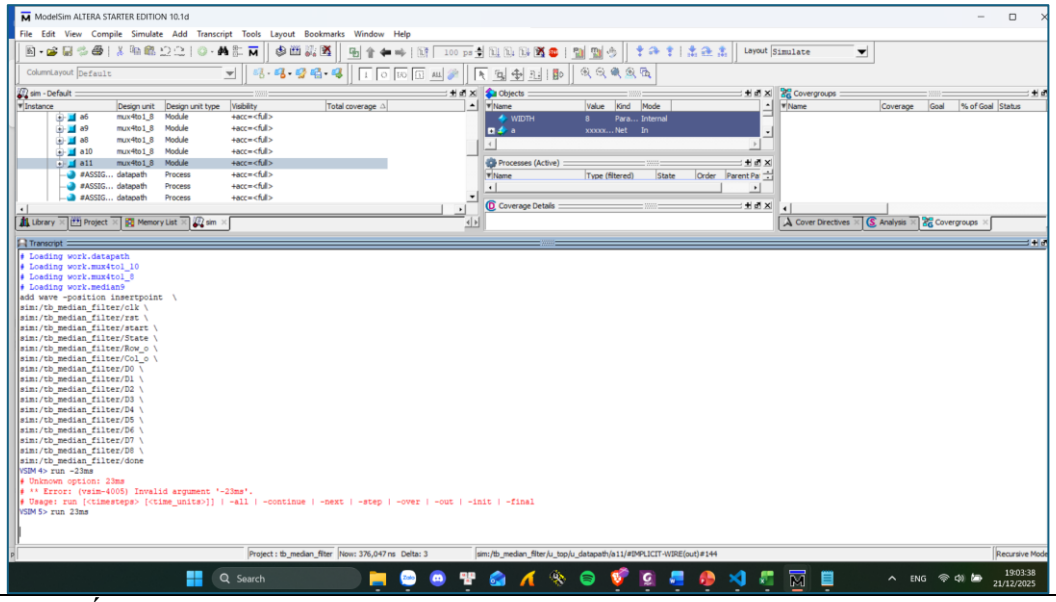
- Set top là tb_median_filter.v



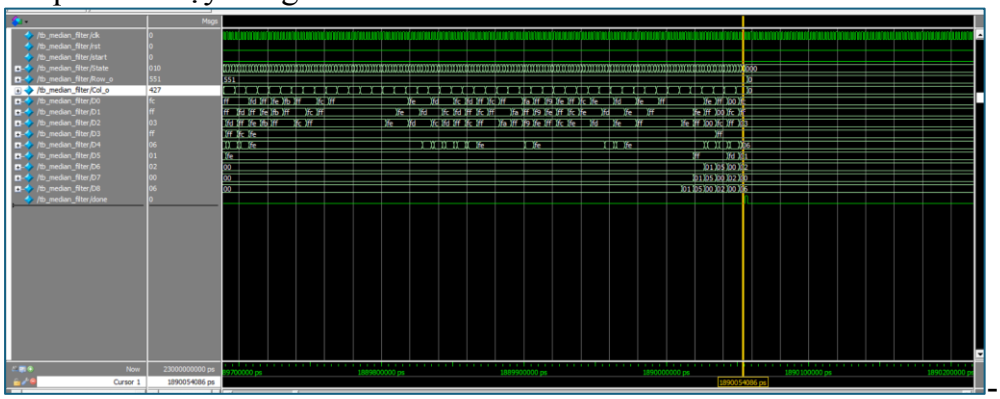
- Add Wave :

```
ModelSim> vsim -gui work.tb_median_filter
# vsim -gui work.tb_median_filter
# Loading work.tb_median_filter
# Loading work.top
# Loading work.controller
# Loading work.datapath
# Loading work.mux4to1_10
# Loading work.mux4to1_8
# Loading work.median9
add wave -position insertpoint  \
sim:/tb_median_filter/clk \
sim:/tb_median_filter/rst \
sim:/tb_median_filter/start \
sim:/tb_median_filter/State \
sim:/tb_median_filter/Row_o \
sim:/tb_median_filter/Col_o \
sim:/tb_median_filter/D0 \
sim:/tb_median_filter/D1 \
sim:/tb_median_filter/D2 \
sim:/tb_median_filter/D3 \
sim:/tb_median_filter/D4 \
sim:/tb_median_filter/D5 \
sim:/tb_median_filter/D6 \
sim:/tb_median_filter/D7 \
sim:/tb_median_filter/D8 \
sim:/tb_median_filter/done

VSIM 4>
```

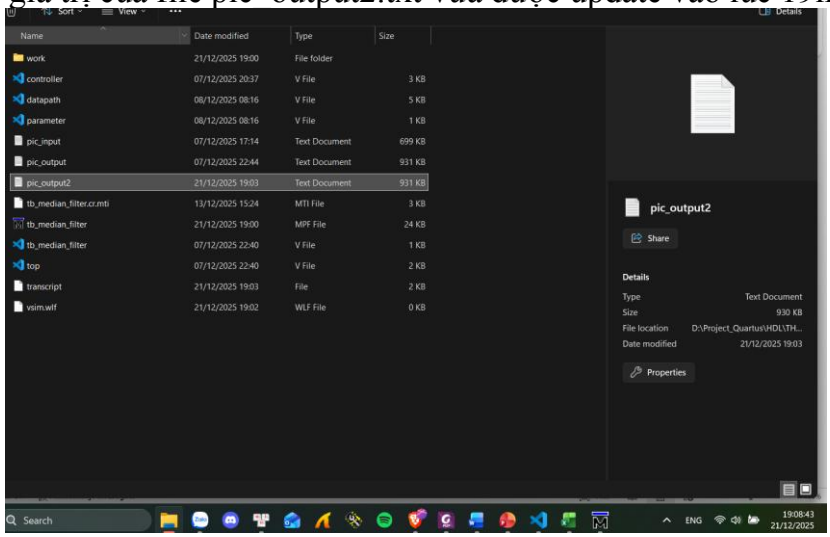
- Run 23ms



- Kết quả khi chạy xong :

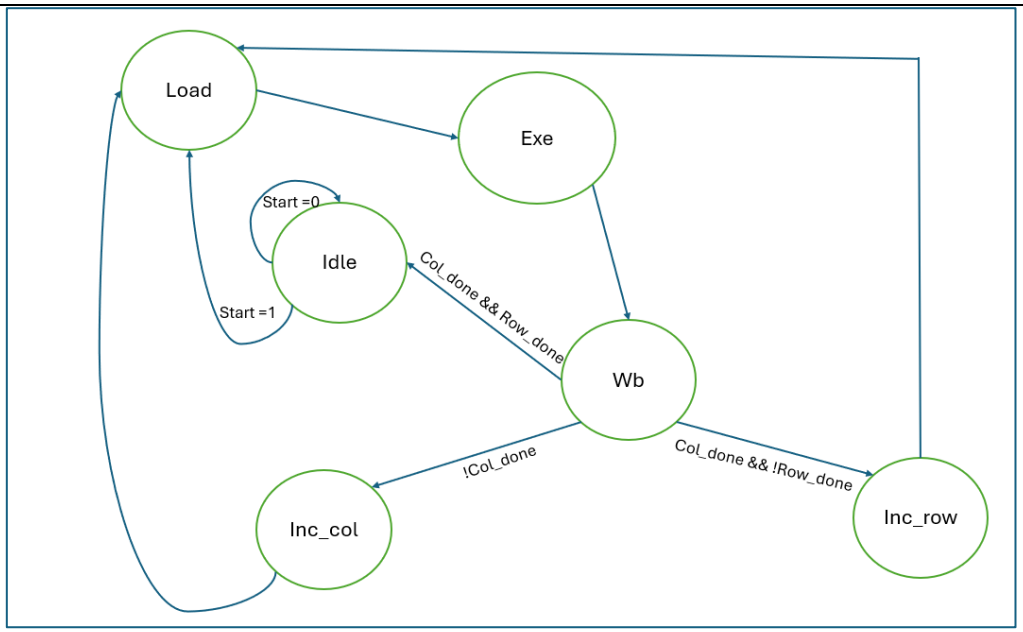


- Cụ thể thì nó chạy khoảng 18ms là xong.
- Hoàn thành khi Row = 427 và Col = 551.
- Cũng như giá trị của file pic_output2.txt vừa được update vào lúc 19h03p.



2. Chuyển ảnh RGB sang ảnh grayscale

2.1. Finite state machine cho mạch



Moore_fsm

Bảng giải thích ý nghĩa của các state trong fsm

State	Chức năng và ý nghĩa
Idle	Không làm gì, chỉ nạp giá trị 0 cho 2 thanh ghi Row và Col.
Load	Load 3 giá trị B G R vào 3 thanh ghi từ bộ nhớ.
Exe	Tính ra giá trị gray tương ứng với 3 giá trị B G R.
Wb	Cập nhật giá trị từ thanh ghi gray vào bộ nhớ buffer.
Inc_col	Tăng giá trị của thanh ghi Col lên 1 đơn vị.
Inc_row	Tăng giá trị của thanh ghi Row lên 1 đơn vị và cho thanh ghi Col về 0.

2.2. Code matlab phục vụ việc chuyển đổi

a) Code chuyển jpg sang hex.

```
% Chuyển JPG sang BGR hex (không header)
clear; clc;

INPUT_JPG = 'baitap2_anhgoc.jpg';
OUTPUT_HEX = 'in.txt';

fprintf('=====================================\n');
fprintf('JPG TO HEX CONVERTER\n');
fprintf('=====================================\n');
fprintf('Input:  %s\n', INPUT_JPG);
fprintf('Output: %s\n', OUTPUT_HEX);
```

```
% Đọc ảnh
fprintf('\nĐọc ảnh...\n');
img = imread(INPUT_JPG);
[height, width, channels] = size(img);

fprintf('Kích thước: %d x %d x %d\n', height, width, channels);

if channels ~= 3
   error('Ảnh phải là RGB (3 channels)!');
end

% Tách RGB
R = img(:,:,1);
G = img(:,:,2);
B = img(:,:,3);

% Tạo BGR data (từ TRÊN xuống DƯỚI, TRÁI sang PHẢI)
fprintf('\nChuyển RGB → BGR...\n');
num_pixels = height * width;
bgr_data = zeros(num_pixels * 3, 1, 'uint8');

idx = 1;
for row = 1:height
   for col = 1:width
      bgr_data(idx)   = B(row, col);  % Blue
      bgr_data(idx+1) = G(row, col);  % Green
      bgr_data(idx+2) = R(row, col);  % Red
      idx = idx + 3;
   end

   % Progress
   if mod(row, 100) == 0
      fprintf('  %.1f%%\n', row/height*100);
   end
end

fprintf('Tổng bytes: %d\n', length(bgr_data));

% Ghi file hex
fprintf('\nGhi file hex...\n');
fid = fopen(OUTPUT_HEX, 'w');

for i = 1:length(bgr_data)
   fprintf(fid, '%02X\n', bgr_data(i));
```

```
        % Progress
        if mod(i, 100000) == 0
            fprintf('  %.1f%%\n', i/length(bgr_data)*100);
        end
    end

    fclose(fid);

    fprintf('\n==================================\n');
    fprintf('✓ Hoàn thành!\n');
    fprintf('==================================\n');
    fprintf('File:   %s\n', OUTPUT_HEX);
    fprintf('Bytes:  %d\n', length(bgr_data));
    fprintf('Pixels: %d\n', num_pixels);

    % Verify - hiển thị 9 bytes đầu
    fprintf('\n9 bytes đầu (3 pixels BGR):\n');
    for i = 1:3
        fprintf('Pixel %d: %02X %02X %02X (B=%d, G=%d, R=%d)\n', ...
            i-1, ...
            bgr_data(i*3-2), bgr_data(i*3-1), bgr_data(i*3), ...
            bgr_data(i*3-2), bgr_data(i*3-1), bgr_data(i*3));
end
```

- Code này giúp chuyển jpg thành hex và thứ tự pixel là **B G R.** Code này còn giúp loại bỏ header của jpg.
- File ảnh gốc (ảnh màu) :

- File input sau khi chuyển sang hex : https://drive.google.com/file/d/111GrXiDWtHaUhyG96PV37iPtOSKo2Npb/view?usp=sharing

b) Code chuyển hex sang jpg

```
function fast_writememh_to_jpg(input_file, output_file, width,
height)
    % Đọc file nhanh bằng textscan
    fid = fopen(input_file, 'r');

    % Đọc tất cả các giá trị hex
    data = textscan(fid, '%s', 'CommentStyle', '//');
    fclose(fid);

    % Chuyển hex sang decimal
    hex_values = data{1};
    pixels = cellfun(@hex2dec, hex_values);

    fprintf('Read %d pixels\n', length(pixels));

    % Reshape
    img = reshape(pixels(1:width*height), [width, height])';
    img = uint8(img);

    % Lưu và hiển thị
    imwrite(img, output_file, 'jpg', 'Quality', 95);
    figure; imshow(img);
    title(sprintf('%s (%dx%d)', output_file, height, width));

    fprintf('✓ Saved: %s\n', output_file);
end

% Sử dụng
fast_writememh_to_jpg('out_2.txt', 'test2.jpg', 2048, 1365);
```

- Code này giúp chuyển file hex sang jpg.
- Em có chuẩn bị ba phiên bản của ảnh.
- Ảnh 1 không tăng giảm độ sáng :

- File txt của ảnh 1 : https://drive.google.com/file/d/1U0R8-N9N88sx4L3_EbM_9w8PhePbORtz/view?usp=drive_link
- Ảnh 2 tăng độ sáng lên 100 (cộng với value=100) :

- File txt của ảnh 2 : https://drive.google.com/file/d/1CqrELgGHoSC0M0-hLjJe3dUTWOzGfgRR/view?usp=drive_link

- Ảnh 3 là giảm độ sáng đi 100(trừ value 100) :

- File txt của ảnh 3 : https://drive.google.com/file/d/1aBpVG3OETGtNlbigPbKKH2qlsPaMS0Uh/view?usp=drive_link

2.3. Code verilog (design , testbench, kết quả)

a) Module controller

```
module controller (
   input wire start,clk,rst,Col_done,Row_done,
   output reg [1:0] Col_ctrl, Row_ctrl,enable_wr, data_ctrl,gray_ctrl,
   output reg [2:0] State
);
   localparam  Idle = 3'd0,
               Exe = 3'd2,
               Wb  = 3'd3,
               Inc_col = 3'd4,
               Inc_row = 3'd5 ,
               Load = 3'd1;

   reg [2:0] cur_state, n_state;

   always @(posedge clk or posedge rst) begin
      if(rst)
         cur_state <= Idle;
      else begin
         cur_state <= n_state;
      end
   end
```

```
always @(*) begin
   case(cur_state)
      Idle : n_state = (start) ? Load : Idle;
      Exe : n_state = Wb;
      Wb : begin
         if(Col_done && Row_done)
            n_state = Idle;
         else if(Col_done)
            n_state = Inc_row;
         else
            n_state = Inc_col;
      end
      Load : n_state = Exe;
      Inc_col : n_state = Load;
      Inc_row : n_state = Load;
      default : n_state = Idle;
   endcase
end

always @(*)  begin
   State = cur_state ;
   enable_wr = 2'd0;
   data_ctrl = 2'd0;
   Col_ctrl = 2'd0;
   Row_ctrl = 2'd0;
   gray_ctrl = 2'd0;
   case (cur_state)
      Idle : begin
         Col_ctrl = 2'd1 ;// Load 0
         Row_ctrl = 2'd1 ;
      end
      Load : begin
         data_ctrl = 2'd1; // Load Data to Reg
      end
      Exe : begin
         gray_ctrl = 2'd1 ; // Load for gray
      end
      Wb : begin
         enable_wr = 2'd1 ;
      end
      Inc_col : begin
         Col_ctrl = 2'd2 ; // increase 1
         //data_ctrl = 2'd1;
      end
      Inc_row : begin
         Row_ctrl = 2'd2 ; // increase 1
```

```
            Col_ctrl = 2'd1; // reload 0
            //data_ctrl = 2'd1;
         end
      endcase
   end

endmodule
```

- Code này khá tương đồng với bài 1, chỉ khác đôi phần là không xử lý cửa sổ 3x3 mà chỉ load ra 3 giá trị lần lượt trên 1 hàng để xử lý.

b) Module BGR_to_Gray

```
// Y = 0.299 * R + 0.587*G + 0.114 * B
// Y = (299 * R + 587 * G + 114 * B) /1000
// xấp xỉ Y =  (299 * R + 587 * G + 114 * B) >> 10bit
module BGR_to_Gray
(
   input wire [7:0] B,G,R,Value,
   input wire op, // 1 la cong, 0 la tru
   output reg [7:0] Gray
);
   wire [8:0] temp = (299 * R + G * 587 + 114 * B) >> 10 ;
   wire signed [9:0] tmp = (op ) ? (temp + Value ) : (temp -Value);
   always @(*) begin
      if(tmp > 255)
         Gray = 8'd255;
      else if( tmp < 0)
         Gray = 8'd0;
      else
         Gray = tmp[7:0];
   end
endmodule
```

- Từ công thức chuẩn của thầy cho em biến đổi thành 1 công thức xấp xỉ để cho mạch đơn giản.
- Bên cạnh đó còn có op là chọn trừ hay cộng value độ sáng.
- Cộng thì nó sẽ sáng thêm và trừ thì nó sẽ tối đi.
- Value là giá trị biến điều chỉnh độ sáng.
- Em tạo biến tmp có dấu để có thể kiểm tra nếu nó bị vượt qua 255 thì sẽ gán nó là 255 và nếu nó bé hơn 0 thì sẽ gán nó là 0. Mở rộng bit để giá trị chính xác hơn.

c) Module Datapath

```
// Thu tu : B- G - R
//`include "parameter.v"
module datapath#(
```

```
    parameter   WIDTH = 2048,
                HEIGHT = 1365,
                INFILE = "in.txt",
                OUTFILE = "out_2.txt"
)(
    input wire clk,
    input wire [1:0] Col_ctrl, Row_ctrl,enable_wr, data_ctrl,gray_ctrl,
    output wire Col_done,Row_done,
    input wire [7:0] Value,
    input wire op,
    output [7:0] Red_o, Green_o, Blue_o, Gray_o,
    output [10:0] Col_o, Row_o
);
    parameter size_a = 8386560 ; // 2048 * 1365 *3
    parameter size_b = 2795520 ; //2048 * 1365
    reg [7:0] Red,Green,Blue,Gray;
    reg [7:0] mem [0: size_a-1];

    reg [7:0] mem_gray [0: size_b - 1] ;
    reg [10:0] Row,Col;
    wire [10:0] Row_in, Col_in;
    wire [7:0] Red_in, Green_in, Blue_in , Gray_in;

    initial begin
        $readmemh(INFILE,mem,0,size_a-1); // read file from INFILE
    end


    mux4to1_11 a0(
        .a(Row),
        .b(11'd0),
        .c(Row + 11'd1),
        .d(11'd0),
        .sel(Row_ctrl),
        .out(Row_in)
    );

    mux4to1_11 a7(
        .a(Col),
        .b(11'd0),
        .c(Col + 11'd1),
        .d(11'd0),
        .sel(Col_ctrl),
        .out(Col_in)
    );
```

```
mux4to1_8 a4 (
    .a(Blue),
    .b(mem[Row*WIDTH*3 + Col*3  ]),
    .c(8'd0),
    .d(8'd0),
    .sel(data_ctrl),
    .out(Blue_in)
);
mux4to1_8 a2 (
    .a(Red),
    .b(mem[Row*WIDTH*3 + Col*3 + 2 ]),
    .c(8'd0),
    .d(8'd0),
    .sel(data_ctrl),
    .out(Red_in)
);
mux4to1_8 a3 (
    .a(Green),
    .b(mem[Row*WIDTH*3 + Col*3 + 1]),
    .c(8'd0),
    .d(8'd0),
    .sel(data_ctrl),
    .out(Green_in)
);

wire [7:0] new_Gray;
BGR_to_Gray a6 (
    .B(Blue),
    .G(Green),
    .R(Red),
    .Value(Value),
    .op(op),
    .Gray(new_Gray)
);
mux4to1_8 a5 (
    .a(Gray),
    .b(new_Gray),
    .c(8'd0),
    .d(8'd0),
    .sel(gray_ctrl),
    .out(Gray_in)
);

always @(posedge clk)
begin
    if(enable_wr) begin
```

```
            mem_gray[Row*WIDTH + Col] <= Gray;
        end
        Gray <= Gray_in;
        Red <= Red_in;
        Blue <= Blue_in;
        Green <= Green_in;
        Col <= Col_in;
        Row <= Row_in;
        if (Row_done && Col_done ) begin
            $writememh(OUTFILE,mem_gray,0,size_b-1);
        end
    end

    assign Row_done = (Row ==(HEIGHT -1 )) ? 1 : 0;
    assign Col_done = (Col == (WIDTH -1 )) ? 1 : 0;
    assign Red_o  = Red;
    assign Green_o = Green;
    assign Gray_o = Gray ;
    assign Blue_o = Blue;
    assign Row_o = Row;
    assign Col_o = Col;

endmodule
```

- Mảng 1 chiều lưu file txt thì có độ rộng là Width * Height * 3 vì có 3 mảng màu B G R.
- Nhưng khi chuyển về Gray thì nó kích thước chỉ còn Width * Height

```
parameter size_a = 8386560 ; // 2048 * 1365 *3
parameter size_b = 2795520 ; //2048 * 1365
reg [7:0] Red,Green,Blue,Gray;
reg [7:0] mem [0: size_a-1];

reg [7:0] mem_gray [0: size_b - 1] ;
```

- Cơ bản code bài này cũng khá tương tự với bài kia nhưng thay vì ghi median thì code này ghi giá trị Gray của thanh ghi Gray vô mảng bufer mem_gray thôi.

d) Module Top

```
`timescale 1ns / 1ps

module top #(
    parameter WIDTH = 2048,
    parameter HEIGHT = 1365,
    parameter INFILE = "in.txt",
    parameter OUTFILE = "out_2.txt"
```

```
)(
    input wire clk,
    input wire rst,
    input wire start,
    input wire op,
    input wire [7:0] Value,
    output wire [1:0] enable_wr,
    output wire [2:0] State,
    output wire [7:0] Red_o, Green_o, Blue_o, Gray_o,
    output wire [10:0] Col_o, Row_o
);

    // Internal signals
    wire [1:0] Col_ctrl, Row_ctrl,  data_ctrl, gray_ctrl;
    wire Col_done, Row_done;

    // Instantiate Controller
    controller u_controller (
        .clk(clk),
        .rst(rst),
        .start(start),
        .Col_done(Col_done),
        .Row_done(Row_done),
        .Col_ctrl(Col_ctrl),
        .Row_ctrl(Row_ctrl),
        .enable_wr(enable_wr),
        .data_ctrl(data_ctrl),
        .gray_ctrl(gray_ctrl),
        .State(State)
    );

    // Instantiate Datapath
    datapath #(
        .WIDTH(WIDTH),
        .HEIGHT(HEIGHT),
        .INFILE(INFILE),
        .OUTFILE(OUTFILE)
    ) u_datapath (
        .clk(clk),
        .Col_ctrl(Col_ctrl),
        .Row_ctrl(Row_ctrl),
        .enable_wr(enable_wr),
        .data_ctrl(data_ctrl),
        .gray_ctrl(gray_ctrl),
        .Col_done(Col_done),
        .Row_done(Row_done),
```

```
        .Red_o(Red_o),
        .Green_o(Green_o),
        .Blue_o(Blue_o),
        .Gray_o(Gray_o),
        .Col_o(Col_o),
        .Row_o(Row_o),
        .Value(Value),
        .op(op)
    );

endmodule
```

e) Module testbench

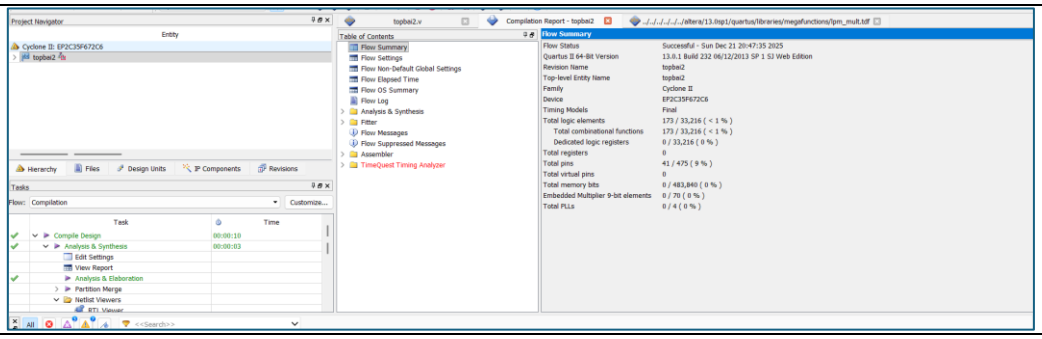
```
`timescale 1ns / 1ps

module tb_top();
    // Parameters
    parameter WIDTH = 2048;
    parameter HEIGHT = 1365;
    parameter INFILE = "in.txt";
    parameter OUTFILE = "out_2.txt";
    parameter CLOCK_PERIOD = 2; // 10ns = 100MHz

    // Signals
    reg clk;
    reg rst;
    reg start;
    reg [7:0] Value;
    reg op;
    wire [2:0] State;
    wire enable_wr;
    wire [7:0] Red_o, Green_o, Blue_o, Gray_o;
    wire [10:0] Row_o,Col_o;

    // Clock generation
    initial begin
        clk = 0;
        forever #(CLOCK_PERIOD/2) clk = ~clk;
    end

    // Instantiate Top module
    top #(
        .WIDTH(WIDTH),
        .HEIGHT(HEIGHT),
        .INFILE(INFILE),
```

```
        .OUTFILE(OUTFILE)
    ) dut (
        .clk(clk),
        .rst(rst),
        .start(start),
        .State(State),
        .Red_o(Red_o),
        .Value(Value),
        .op(op),
        .Green_o(Green_o),
        .Blue_o(Blue_o),
        .Gray_o(Gray_o),
        .Row_o(Row_o),
        .Col_o(Col_o),
        .enable_wr(enable_wr)
    );

    // Test stimulus
    initial begin
        // Initialize
        rst = 0;
        start = 0;

        // Display header
        $display("========================================");
        $display("BGR to Grayscale Conversion Testbench");
        $display("Image Size: %0d x %0d", WIDTH, HEIGHT);
        $display("Total Pixels: %0d", WIDTH * HEIGHT);
        $display("========================================");

        // Reset
        #2;
        rst = 1;
        #2;
        rst = 0;
        #2;

        // Start processing
        $display("Starting conversion at time %0t", $time);
        start = 1;
        #(CLOCK_PERIOD);
        start = 0;
        Value = 8'd100;
        op = 0;
        // Wait for completion
```

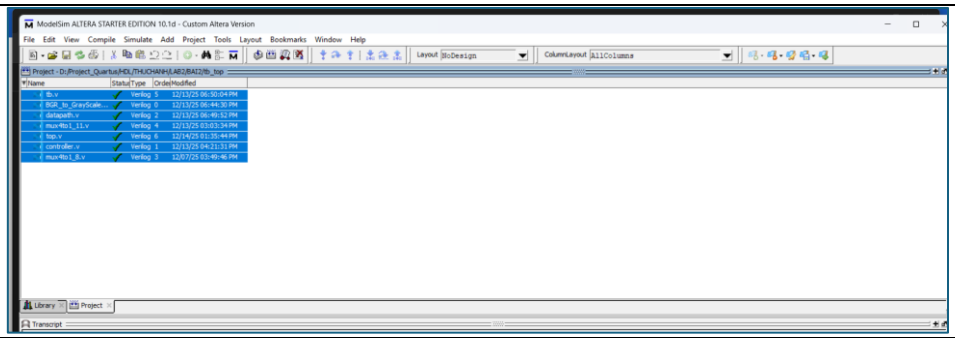
```
        wait(State == 3'd0 && dut.u_datapath.Row_done &&
dut.u_datapath.Col_done);
        #100;

        $display("Conversion completed at time %0t", $time);
        $display("Output file: %s", OUTFILE);
        $display("==========================================");

        $finish;
    end
endmodule
```

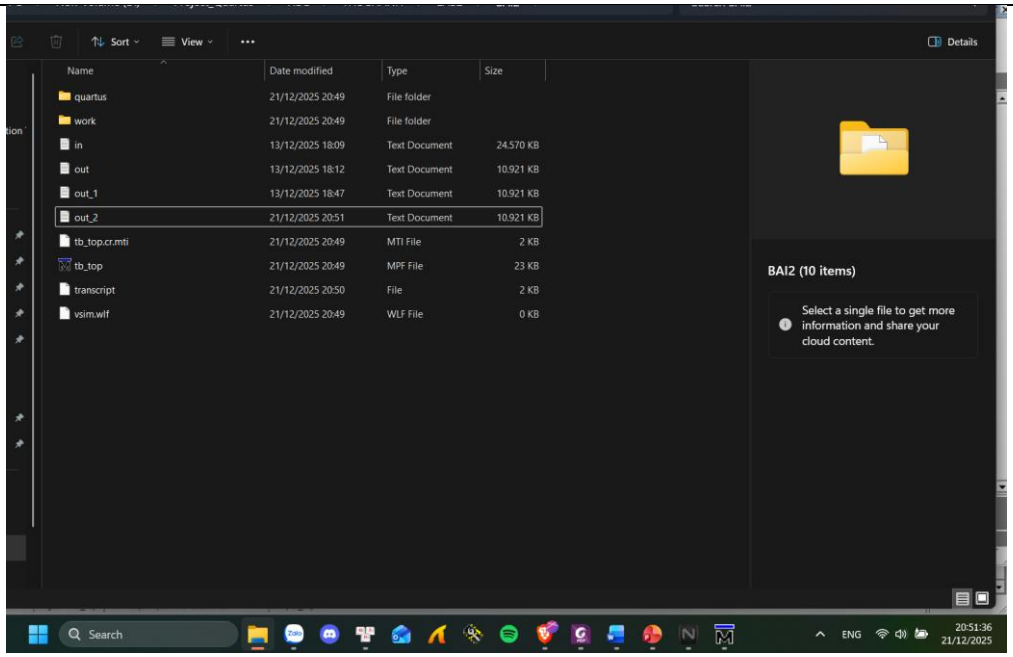
- Đây là testbench của việc tạo ra ảnh tối thứ 3.
- Tương tự như bài trước nên em sẽ chỉ chạy lại 1 waveform của nó vì nó tận 3 lần wf khác nhau.

f) Kết quả chạy synthesis trên quartus.

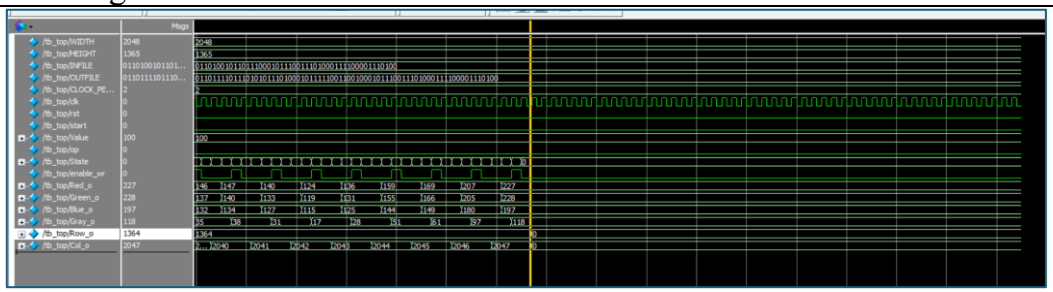


g) Kết quả mô phỏng waveform



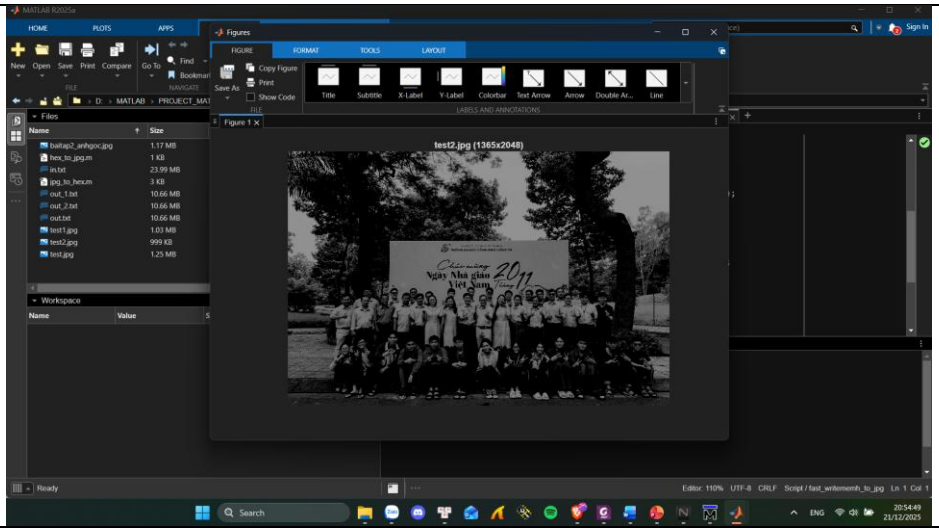


- Out_2 đã được update vào 20h51 khi dump file của chạy mô phỏng thành công.



- Chạy hoàn thành khi Row = 1364 = 1365 – 1. Và Col = 2047 = 2048-1.
- Này là duyệt theo index rồi không còn duyệt theo cột nữa, bởi một index tận 3 cột.
- Chạy khoảng 22ms.

h) Gen ảnh ra từ file 1 lần nữa.



V. KẾT QUẢ VÀ NHẬN XÉT

1. Xữ lý nhiễu muối tiêu

Ảnh xử lý sau nhiễu có thông số **SSIM** bằng : 95,81% và **PSNR** 33. 2088 dB.

2. Chuyển ảnh RGB sang ảnh grayscale

Ảnh sau xử lý đã là ảnh grayscale và có thể tùy chỉnh độ sáng theo 1 biến value và op.

VI. TÀI LIỆU THAM KHẢO

[1] FPGA4student.com, “Image processing on FPGA using Verilog HDL,” FPGA4student.com, Nov. 2016. [Online]. Available: https://www.fpga4student.com/2016/11/image-processing-on-fpga-verilog.html.

[2] D. System Design, “FPGA Implementation Median Filter for De-Noising,” Digital System Design, Feb. 11, 2021. [Online]. Available: https://digitalsystemdesign.in/fpga-implementation-median-filter-for-de-noising/.

[3] MS_IT, “[Xử lý ảnh] – Lọc trung vị (Median filter),” YouTube, 2-Mar-2019. [Online]. Available: https://www.youtube.com/watch?v=jD_FxL5zsS0.

[4] Thiết kế khối xử lý ảnh - Image Processing Verilog, YouTube, 11-Oct-2021. [Online]. Available: https://www.youtube.com/watch?v=4WwUw7aAp3I.