

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**  
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**  
**KHOA KỸ THUẬT MÁY TÍNH**



**BÁO CÁO ĐỒ ÁN**  
**MÔN THIẾT KẾ HỆ THỐNG SỐ VỚI HDL – CE213.O11**

**ĐỀ TÀI: ỨNG DỤNG THUẬT TOÁN XỬ LÝ ẢNH LÊN FPGA**  
**SỬ DỤNG VERILOG**

**(AN APPLICATION OF IMAGE PROCESSING ON FPGA USING VERILOG)**

**HỌ VÀ TÊN:**

**MSSV:**

**ĐẶNG TẤN ĐẠT**

**21521927**

**NGUYỄN TUẤN KIẾT**

**21521036**

**NGUYỄN HOÀNG KHÁNH DUY**

**21522002**

**GIẢNG VIÊN HƯỚNG DẪN:**

**HỒ NGỌC DIỄM**

**TP. HỒ CHÍ MINH – Tháng 12 năm 2023**

# MỤC LỤC

DANH MỤC HÌNH ẢNH .....	II
I. TỔNG QUAN VỀ ĐỀ TÀI .....	1
I.1. Giới thiệu đề tài.....	1
I.2. Cơ sở lý thuyết .....	1
I.2.1. FPGA.....	1
I.2.2. Verilog.....	2
I.2.3. Xử lý ảnh và thuật toán .....	2
I.2.3.1. Khái niệm ảnh và xử lý ảnh.....	2
I.2.3.2. Các thuật toán xử lý ảnh.....	3
II. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG.....	6
II.1. Sơ đồ tổng quan .....	6
II.2. Thiết kế khối read unit.....	6
II.2.1. Thiết kế khối đọc dữ liệu từ file hex (mô phỏng).....	7
II.2.2. Thiết kế máy trạng thái và tín hiệu điều khiển.....	8
II.2.3. Thiết kế khối xử lý ảnh .....	9
II.3. Thiết kế khối write unit.....	10
II.3.1. Thiết lập 54-byte header bit mapfile .....	10
II.3.2. Thiết kế khối ghi dữ liệu ra file hex và bmp.....	11
II.3.3. Testbench .....	12
III. KIỂM TRA VÀ MÔ PHỎNG.....	13
IV. LINK PROJECT VÀ TÀI LIỆU THAM KHẢO .....	15

## DANH MỤC HÌNH ẢNH

Hình 1 Hàm Gauss .....	4
Hình 2 Kernel được sử dụng trong thuật toán Sobel.....	5
Hình 3 Sơ đồ khối tổng quan.....	6
Hình 4 Khai báo chân, biến cho module img_read .....	6
Hình 5 Hàm đọc file vào một biến .....	7
Hình 6 Module điều khiển tín hiệu start.....	7
Hình 7 Module chia ảnh thành R, G, B và padding .....	7
Hình 8 Module máy trạng thái .....	8
Hình 9 Module tín hiệu cho khối xử lý và tín hiệu kết thúc .....	8
Hình 10 Module xử lý ảnh .....	9
Hình 11 Khai báo chân, biến cho module img_write.....	10
Hình 12 Gán giá trị 54 byte header bmp file.....	10
Hình 13 Module duyệt ảnh theo dòng và cột .....	11
Hình 14 Module lưu các giá trị RGB vào mảng .....	11
Hình 15 Module đếm để xác định khi nào hoàn tất việc lưu vào mảng.....	11
Hình 16 Module truyền mảng ra file hex và bmp .....	11
Hình 17 Testbench.....	12
Hình 18 Ảnh đầu vào .....	13
Hình 19 Ảnh sau khi qua thuật toán Grayscale.....	13
Hình 20 Ảnh qua thuật toán Sobel từ ảnh xám .....	14
Hình 21 Ảnh qua thuật toán Gaussian blur .....	14

## PHẦN TRẢM ĐÓNG GÓP & PHÂN CHIA CÔNG VIỆC

ĐẶNG TẤN ĐẠT	34%
NGUYỄN TUẤN KIẾT	33%
NGUYỄN HOÀNG KHÁNH DUY	33%

# I. TỔNG QUAN VỀ ĐỀ TÀI

## I.1. Giới thiệu đề tài

Trong thời đại số hóa và công nghệ phát triển nhanh chóng hiện nay, xử lý ảnh đã trở thành một lĩnh vực quan trọng và đa dạng trong nhiều ứng dụng. Việc phân tích và xử lý hình ảnh không chỉ hỗ trợ trong lĩnh vực y tế, an ninh, nhận dạng khuôn mặt, mà còn trong nhiều ngành công nghiệp khác như ô tô tự lái, thị trường nông nghiệp thông minh và thực tế ảo. Tuy nhiên, xử lý ảnh đòi hỏi khá nhiều tài nguyên tính toán và thời gian, đặc biệt khi xử lý ảnh thời gian thực. Để đáp ứng yêu cầu này, sử dụng FPGA (Field-Programmable Gate Array) kết hợp với ngôn ngữ mô tả phần cứng Verilog đã trở thành một phương pháp phổ biến để thực hiện xử lý ảnh nhanh chóng và hiệu quả.

Cụ thể ở đây, nhóm đã quyết định lựa chọn thuật toán làm xám (grayscale), thuật toán làm mờ (gaussian) và thuật toán tìm cạnh Sobel (sobel edge detection) để tìm hiểu và áp dụng vào bằng verilog.

## I.2. Cơ sở lý thuyết

### I.2.1. FPGA

Field-programmable gate array (FPGA) là một loại mạch tích hợp cỡ lớn dùng cấu trúc mảng phần tử logic mà người dùng có thể lập trình được. Chữ field ở đây muốn chỉ đến khả năng tái lập trình "bên ngoài" của người sử dụng, không phụ thuộc vào dây chuyền sản xuất phức tạp của nhà máy bán dẫn. Vi mạch FPGA được cấu thành từ các bộ phận:

- Các khối logic cơ bản lập trình được (logic block)
- Hệ thống mạch liên kết lập trình được
- Khối vào/ra (IO Pads)
- Phần tử thiết kế sẵn khác như DSP slice, RAM, ROM, nhân vi xử lý...

FPGA có nhiều ứng dụng trong các lĩnh vực khác nhau bao gồm:

- Xử lý tín hiệu số: FPGA rất phù hợp để triển khai các thuật toán xử lý tín hiệu như xử lý hình ảnh, xử lý âm thanh và giao thức truyền thông như Wi-Fi và 5G.
- Hệ thống nhúng: FPGA có thể được sử dụng như một nền tảng phần cứng linh hoạt cho các hệ thống nhúng, kết hợp vi xử lý hoặc bộ điều khiển nhúng với các trình tăng tốc phần cứng tùy chỉnh.
- Tạo mẫu và xác minh: FPGA được sử dụng rộng rãi để tạo mẫu và xác minh các hệ thống số phức tạp trước khi hoàn thiện thiết kế cho sản xuất. Chúng cho phép kiểm tra và xác minh nhanh chóng các thiết kế.
- Trading: FPGA được sử dụng trong các hệ thống giao dịch tần số cao, nơi độ trễ thấp và hiệu năng cao là quan trọng để thực hiện giao dịch nhanh chóng và hiệu quả.
- Hàng không vũ trụ và quốc phòng: FPGA được sử dụng trong các ứng dụng hàng không vũ trụ và quốc phòng như hệ thống radar, truyền thông vệ tinh, mật mã hóa và chiến thuật điện tử.

FPGA mang lại nhiều lợi ích như:

- Linh hoạt: FPGA cho phép người thiết kế tạo ra các thiết kế phần cứng số tùy chỉnh để đáp ứng các yêu cầu cụ thể. Chúng có thể triển khai các chức năng, thuật toán và giao thức phức tạp trong một thiết bị duy nhất.

- Hiệu năng: FPGA có thể cung cấp hiệu năng cao nhờ khả năng xử lý song song. Chúng có thể xử lý nhiều nhiệm vụ cùng một lúc, phù hợp cho các ứng dụng yêu cầu xử lý thời gian thực và độ trễ thấp.
- Tối ưu thời gian ra thị trường: FPGA giúp rút ngắn thời gian ra thị trường vì cho phép người thiết kế triển khai và kiểm tra ý tưởng nhanh chóng. Chúng có chu kỳ phát triển ngắn hơn so với ASIC vì không cần gia công phức tạp.
- Hiệu quả chi phí: FPGA có thể mang lại lợi ích về chi phí cho sản xuất ở số lượng sản xuất thấp đến trung bình. Chúng loại bỏ nhu cầu về các khuôn mẫu chế tạo, giảm chi phí ban đầu liên quan đến sản xuất chip tùy chỉnh.

### **I.2.2. Verilog**

Verilog, được tiêu chuẩn hóa thành IEEE 1364, là ngôn ngữ mô tả phần cứng (hardware description language, viết tắt: HDL) được sử dụng để mô hình hóa các hệ thống điện tử. Nó được sử dụng phổ biến nhất trong thiết kế và xác minh các mạch kỹ thuật số ở trừu tượng mức chuyển thanh ghi. Nó cũng được sử dụng trong việc xác minh các mạch tương tự và mạch tín hiệu hỗn hợp, cũng như trong thiết kế các mạch di truyền. Vào năm 2009, tiêu chuẩn Verilog (IEEE 1364-2005) đã được hợp nhất vào tiêu chuẩn SystemVerilog, tạo ra tiêu chuẩn IEEE 1800-2009. Kể từ đó, Verilog chính thức là một phần của ngôn ngữ SystemVerilog.

### **I.2.3. Xử lý ảnh và thuật toán**

#### **I.2.3.1. Khái niệm ảnh và xử lý ảnh**

Trước khi bắt tay vào xử lý hình ảnh, trước tiên chúng ta cần hiểu chính xác những gì tạo nên một hình ảnh. Ảnh thường được xem như là một tập hợp các điểm hình ảnh (pixels) sắp xếp theo một cấu trúc. Mỗi pixel đại diện cho một đơn vị nhỏ nhất của hình ảnh và chứa thông tin về màu sắc và độ sáng của điểm ảnh tương ứng trên hình ảnh. Sự kết hợp của các pixel này tạo thành một hình ảnh hoàn chỉnh. Một hình ảnh được thể hiện bằng kích thước (chiều cao và chiều rộng) dựa trên số lượng pixel. Pixel này là một điểm trên hình ảnh có sắc thái, độ mờ hoặc màu sắc cụ thể. Nó thường được thể hiện ở một trong những điều sau đây:

- Thang độ xám - Pixel là một số nguyên có giá trị từ 0 đến 255 (0 hoàn toàn đen và 255 hoàn toàn trắng).
- RGB - Một pixel được tạo thành từ 3 số nguyên từ 0 đến 255 (các số nguyên biểu thị cường độ của màu đỏ, lục và lam).
- RGBA - Đây là phần mở rộng của RGB có thêm trường alpha, đại diện cho độ mờ của hình ảnh.

Xử lý ảnh là quá trình thay đổi và cải thiện các tính chất của một hình ảnh để đáp ứng các mục tiêu cụ thể. Nó bao gồm các phương pháp và kỹ thuật để thực hiện các thay đổi này, từ việc cải thiện chất lượng hình ảnh đến phân tích và trích xuất thông tin từ hình ảnh.

### **I.2.3.2. Các thuật toán xử lý ảnh**

#### **• Grayscale**

Là quá trình chuyển đổi hình ảnh từ các không gian màu khác, ví dụ: RGB, CMYK, HSV, v.v. sang các sắc thái xám. Nó khác nhau giữa màu đen hoàn toàn và màu trắng hoàn toàn

Tầm quan trọng của thang độ xám:

- Giảm kích thước: Ví dụ: Trong hình ảnh RGB có ba kênh màu và ba chiều trong khi hình ảnh thang độ xám là một chiều.
- Giảm độ phức tạp của mô hình: Cân nhắc việc đào tạo các bài viết thần kinh về hình ảnh RGB có kích thước  $10 \times 10 \times 3$  pixel. Lớp đầu vào sẽ có 300 nút đầu vào. Mặt khác, cùng một mạng lưới thần kinh sẽ chỉ cần 100 nút đầu vào cho hình ảnh thang độ xám.
- Để các thuật toán khác hoạt động: Nhiều thuật toán được tùy chỉnh để chỉ hoạt động trên các hình ảnh thang độ xám.

Cách thức hoạt động của thuật toán:

- Đầu tiên, hình ảnh màu ban đầu được chia thành các thành phần màu cơ bản, chẳng hạn như đỏ (R), xanh lá cây (G), và xanh dương (B), nếu hình ảnh được biểu diễn dưới dạng không gian màu RGB (Red-Green-Blue).
- Tiếp theo, một giá trị trung bình của các thành phần màu được tính toán cho mỗi pixel. Có nhiều cách tính giá trị trung bình, như trung bình cộng đơn giản hoặc trung bình có trọng số, tùy thuộc vào yêu cầu cụ thể của ứng dụng.
- Giá trị trung bình tính toán được gán cho cả ba thành phần màu (R, G, B) của pixel tương ứng, tạo ra một giá trị duy nhất đại diện cho độ sáng của pixel đó.
- Quá trình trên được lặp lại cho tất cả các pixel trong hình ảnh, cho đến khi tạo ra hình ảnh xám hoàn chỉnh.

#### **• Gaussian blur**

Trong xử lý hình ảnh, độ mờ Gaussian (còn được gọi là làm mịn Gaussian) là kết quả của việc làm mờ hình ảnh bằng hàm Gaussian (được đặt theo tên của nhà toán học và nhà khoa học Carl Friedrich Gauss).

Đây là một hiệu ứng được sử dụng rộng rãi trong phần mềm đồ họa, thường là để giảm nhiễu hình ảnh và giảm chi tiết. Hiệu ứng hình ảnh của kỹ thuật làm mờ này là hiệu ứng mờ mượt mà giống như xem hình ảnh qua màn hình mờ, khác biệt rõ rệt với hiệu ứng mờ ảo do ống kính mất nét hoặc bóng của vật thể tạo ra dưới ánh sáng thông thường.

Làm mịn Gaussian cũng được sử dụng như một giai đoạn tiền xử lý trong các thuật toán thị giác máy tính nhằm nâng cao cấu trúc hình ảnh ở các tỷ lệ khác nhau—xem biểu diễn không gian tỷ lệ và triển khai không gian tỷ lệ.

Cách thức hoạt động của thuật toán:

- Hình ảnh ban đầu được chuyển đổi thành một ma trận các giá trị pixel, trong đó mỗi pixel chứa thông tin về màu sắc và độ sáng của điểm ảnh tương ứng.
- Tiếp theo, một bộ lọc Gaussian được tạo ra. Bộ lọc này là một ma trận hai chiều có kích thước lớn (3x3, 5x5, 7x7), với các giá trị trọng số được tính toán dựa trên hàm Gauss (Hình 1). Hàm Gauss là một hàm mô tả phân phối xác suất Gauss, có dạng hình chuông và đối xứng.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Hình 1 Hàm Gauss

- Mỗi pixel trong hình ảnh ban đầu được duyệt qua từng vị trí, và áp dụng bộ lọc Gaussian lên các pixel và các pixel lân cận của nó. Quá trình này bao gồm việc tính tổng trọng số của các pixel trong vùng lân cận, và sau đó chia tổng đó cho tổng trọng số của bộ lọc Gaussian.
- Giá trị pixel mới được tính toán bằng cách lấy tổng trọng số của các pixel nhân với giá trị màu sắc hoặc độ sáng tương ứng và chia cho tổng trọng số. Kết quả là một pixel mới với màu sắc hoặc độ sáng được làm mờ dựa trên các pixel lân cận.
- Quá trình trên được lặp lại cho tất cả các pixel trong hình ảnh, tạo ra hình ảnh mới với hiệu ứng mờ theo phân phối Gauss.

- **Sobel edge detection**

Cạnh là những thay đổi cường độ cục bộ đáng kể trong một hình ảnh kỹ thuật số. Một cạnh có thể được định nghĩa là một tập hợp các pixel được kết nối tạo thành ranh giới giữa hai vùng rời nhau. Có ba loại cạnh:

- Cạnh ngang
- Cạnh dọc
- Các cạnh chéo

Phát hiện cạnh (edge detection) là phương pháp phân đoạn hình ảnh thành các vùng không liên tục. Nó là một kỹ thuật được sử dụng rộng rãi trong xử lý ảnh số như nhận dạng mẫu, hình thái hình ảnh, khai thác tính năng.

Phát hiện cạnh (edge detection) cho phép người dùng quan sát các đặc điểm của hình ảnh để biết sự thay đổi đáng kể về mức độ xám. Kết cấu này cho biết điểm cuối của một vùng trong ảnh và điểm bắt đầu của vùng khác. Nó làm giảm lượng dữ liệu trong hình ảnh và bảo toàn các thuộc tính cấu trúc của hình ảnh.

Toán tử Sobel (Sobel operator): Là toán tử vi phân rời rạc. Nó tính toán xấp xỉ độ dốc của hàm cường độ hình ảnh để phát hiện cạnh hình ảnh. Tại các pixel của một hình ảnh, toán tử Sobel tạo ra vector bình thường hoặc vector gradient tương ứng. Nó sử dụng hai hạt nhân

hoặc mặt nạ 3 x 3 được tích hợp với hình ảnh đầu vào (Hình 2) để tính toán các phép tính gần đúng đạo hàm dọc và ngang tương ứng

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad M_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Hình 2 Kernel được sử dụng trong thuật toán Sobel

Thuận lợi:

- Tính toán đơn giản và hiệu quả về thời gian.
- Rất dễ dàng tìm kiếm các cạnh mịn.

Hạn chế:

- Các điểm hướng chéo không phải lúc nào cũng được bảo toàn.
- Rất nhạy cảm với nhiễu.
- Không chính xác lắm trong việc phát hiện cạnh.
- Phát hiện với các cạnh dày và thô không cho kết quả phù hợp.

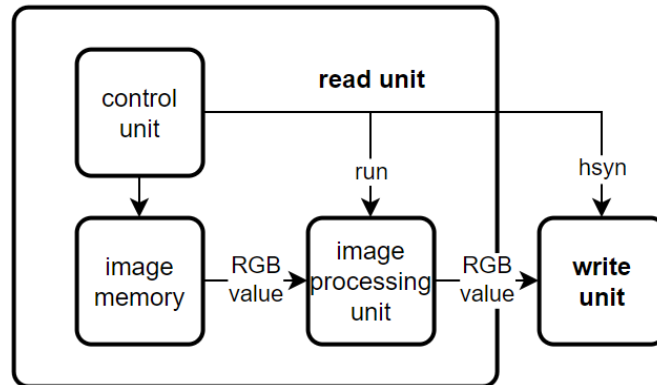
Cách thức hoạt động của thuật toán:

- Hình ảnh ban đầu được chuyển đổi thành một ma trận các giá trị pixel, trong đó mỗi pixel chứa thông tin về màu sắc hoặc độ sáng của điểm ảnh tương ứng.
- Bộ lọc Sobel được áp dụng lần lượt theo hai hướng: dọc theo trục x và dọc theo trục y. Các bộ lọc Sobel là một ma trận hai chiều có kích thước 3x3 hoặc 5x5, với các giá trị trọng số được thiết kế để phát hiện biên cạnh dọc theo hướng tương ứng.
- Đối với phát hiện biên cạnh theo hướng x (dọc theo chiều ngang), bộ lọc Sobel dọc theo trục x được áp dụng lên hình ảnh. Quá trình này bao gồm tính tổng trọng số của các pixel trong vùng lân cận của mỗi pixel và sau đó chia tổng đó cho tổng trọng số của bộ lọc Sobel x.
- Đối với phát hiện biên cạnh theo hướng y (dọc theo chiều dọc), bộ lọc Sobel dọc theo trục y được áp dụng lên hình ảnh. Cũng giống như trên, tổng trọng số của các pixel trong vùng lân cận được tính toán và chia cho tổng trọng số của bộ lọc Sobel y.
- Sau khi tính toán các giá trị gradient theo hướng x và y, gradient tổng hợp (độ lớn gradient) của mỗi pixel được tính toán bằng cách tính căn bậc hai của tổng bình phương độ lớn gradient theo hướng x và theo hướng y tại điểm ảnh tương ứng.
- Giá trị gradient tổng hợp cho mỗi pixel được so sánh với một ngưỡng (threshold) đã được xác định trước. Nếu giá trị gradient vượt qua ngưỡng, pixel đó được coi là một điểm biên cạnh. Ngược lại, nếu giá trị gradient không vượt qua ngưỡng, pixel đó được coi là nền.
- Kết quả là hình ảnh mới, trong đó các điểm biên cạnh được đánh dấu và phân biệt với nền.



## II. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

### II.1. Sơ đồ tổng quan



Hình 3 Sơ đồ khối tổng quan

### II.2. Thiết kế khối read unit.

Khai báo các input, output, biến và bộ nhớ. (Các mảng để lưu ảnh, biến index, ...)

```

`define SOBEL_OPERATION
//`define GAUSSIAN_BLUR_OPERATION
`define GRAYSCALE_OPERATION
module image_read
(
    parameter WIDTH      = 300,                // Image width
    parameter HEIGHT     = 400,                // Image height
    parameter INFILE     = "input.hex",        // image file
    parameter VALUE      = 100,                // value for Brightness operation
    parameter THRESHOLD  = 90,                // Threshold value for Threshold operation
    parameter SIGN       = 1,                // Sign value using for brightness operation
    // SIGN = 0: Brightness subtraction
    // SIGN = 1: Brightness addition
)
(
    input HCLK,                                // clock
    input HRESETn,                            // Reset (active low)
    input mode,                                // Horizontal synchronous pulse
    output reg HSYNC,                          // Horizontal synchronous pulse

    // An HSYNC indicates that one line of the image is transmitted.
    // Used to be a horizontal synchronous signals for writing bmp file.
    output reg [7:0] DATA_R,                  // 8 bit Red data
    output reg [7:0] DATA_G,                  // 8 bit Green data
    output reg [7:0] DATA_B,                  // 8 bit Blue data
    // Process and transmit 2 pixels in parallel to make the process faster, you can modify to transmit 1 pixels or more if needed
    output ctrl_done                           // Done flag
);

localparam sizeOfLengthReal = WIDTH*HEIGHT*3;
// local parameters for FSM
localparam ST_IDLE          = 2'b00,          // idle state
localparam ST_DATA          = 2'b11;          // state for data processing
reg [1:0] cstate,            // current state
           nstate;           // next state
reg start;                  // start signal: trigger Finite state machine beginning to operate
reg HRESETn_d;              // delayed reset signal: use to create start signal
reg ctrl_data_run;          // control signal for data processing
reg [7:0] total_memory [0 : sizeOfLengthReal-1]; // memory to store 8-bit data image
// temporary memory to save image data : size will be WIDTH*HEIGHT*3
reg [7:0] org_R [0 : WIDTH*HEIGHT - 1];        // temporary storage for R component
reg [7:0] org_G [0 : WIDTH*HEIGHT - 1];        // temporary storage for G component
reg [7:0] org_B [0 : WIDTH*HEIGHT - 1];        // temporary storage for B component

reg [7:0] img_pad [0:(WIDTH*2)*(HEIGHT*2)-1];
// counting variables
integer i, j;
// temporary signals for calculation: details in the paper.

integer temp1,temp2,temp3,a,b,value;
reg [ 9:0] row;                // temporary variables in invert and threshold operation
reg [10:0] col;                // row index of the image
reg [18:0] data_count;         // column index of the image
// data counting for entire pixels of the image

```

Hình 4 Khai báo chân, biến cho module img\_read

### II.2.1. Thiết kế khối đọc dữ liệu từ file hex (mô phỏng)

Hàm đọc file hex bao gồm giá trị RGB của mỗi điểm ảnh vào biến `total_memory` với kích thước `sizeofLengthReal - 1`

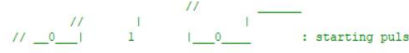
```
initial begin
    $readmemh(INFILE,total_memory,0,sizeofLengthReal-1); // read file from INFILE
    $display("load file successfully");
end
```

Hình 5 Hàm đọc file vào một biến

Module điều khiển tín hiệu start để bắt đầu đọc dữ liệu.

Tín hiệu start bằng 1 khi chân HRESETN kích cạnh lên.

```
//-----//
// ---Begin to read image file once reset was high ---//
// ---by creating a starting pulse (start)-----//
//-----//
always@(posedge HCLK, negedge HRESETn)
begin
    if(!HRESETn) begin
        start <= 0;
        HRESETn_d <= 0;
    end
    else begin
        HRESETn_d <= HRESETn;
        if(HRESETn == 1'b1 && HRESETn_d == 1'b0)
            start <= 1'b1;
        else
            start <= 1'b0;
    end
end
end
```



Hình 6 Module điều khiển tín hiệu start

Sau khi start tích cực mức cao và ảnh được truyền vào `total_memory`, chia `total_memory` vào 3 mảng `org_R`, `org_G`, `org_B`. Nếu `mode = 1` nghĩa là chọn thuật toán sobel hoặc gaussian thì thêm 1 bước padding.

Nguyên lý của việc padding là chèn thêm 1 viền bao quanh ảnh để tránh việc kích thước ảnh thay đổi khi thực hiện các thuật toán có sử dụng kernel.

```
always@(start) begin
    if(start == 1'b1) begin
        for(i=0; i<HEIGHT; i=i+1) begin
            for(j=0; j<WIDTH; j=j+1) begin
                org_R[WIDTH*i+j] = total_memory[WIDTH*3*i+3*j+0]; // save Red component
                org_G[WIDTH*i+j] = total_memory[WIDTH*3*i+3*j+1]; // save Green component
                org_B[WIDTH*i+j] = total_memory[WIDTH*3*i+3*j+2]; // save Blue component
            end
        end
        if(mode) begin
            for(i=0; i<HEIGHT+2; i=i+1) begin
                for(j=0; j<WIDTH+2; j=j+1) begin
                    if( 0<i && i<HEIGHT+1 && 0<j && j<WIDTH+1)
                        img_pad[(WIDTH+2)*i + j] = org_R[WIDTH*(i-1) + (j-1)];
                    else
                        img_pad[(WIDTH+2)*i + j] = 100;
                end
            end
        end
    end
end
end
```

Hình 7 Module chia ảnh thành R, G, B và padding

## II.2.2. Thiết kế máy trạng thái và tín hiệu điều khiển

Thiết kế máy trạng thái, gồm 2 trạng thái chờ và xử lý dữ liệu.

```
always@(posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
        cstate <= SI_IDLE;
    end
    else begin
        cstate <= nstate; // update next state
    end
end

//----- State Transition -----//
//----- IDLE . DATA -----//
always @(*) begin
    case(cstate)
        SI_IDLE: begin
            if(start)
                nstate = SI_DATA;
            else
                nstate = SI_IDLE;
            end
        SI_DATA: begin
            if(ctrl_done)
                nstate = SI_IDLE;
            else
                nstate = SI_DATA;
            end
    endcase
end
```

Hình 8 Module máy trạng thái

Thiết kế bộ tín hiệu cho khối xử lý ảnh, và tín hiệu để biết khi nào duyệt ảnh xong

```
//----- control signal -----//
//----- //
//----- //
always @(*) begin
    ctrl_data_run = 0;
    case(cstate)
        SI_DATA: ctrl_data_run = 1; // trigger counting for data processing
    endcase
end

// counting data, column and row index for reading memory
always@(posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
        data_count <= 0;
        row <= 0;
        col <= 0;
    end
    else begin
        if(ctrl_data_run) begin
            data_count <= data_count + 1;
            if(col == WIDTH - 1) begin
                row <= row + 1;
                col <= 0;
            end
            else
                col <= col + 1;
        end
    end
end

assign ctrl_done = (data_count >= WIDTH*HEIGHT-1)? 1'b1: 1'b0; // done flag
//-----//
```

Hình 9 Module tín hiệu cho khối xử lý và tín hiệu kết thúc

## II.2.3. Thiết kế khối xử lý ảnh

```
//----- Image processing -----//
//-----//
always @(*) begin
    HSYNC = 1'b0;
    DATA_R = 0;
    DATA_G = 0;
    DATA_B = 0;
    if(ctrl_data_run) begin

        HSYNC = 1'b1;
        if(~mode) begin
            /* *****
            /* GRAYSCALE_OPERATION */
            /* *****
            `ifdef GRAYSCALE_OPERATION
                value = (org_B[WIDTH * row + col] + org_R[WIDTH * row + col] + org_G[WIDTH * row + col]) / 3;
                DATA_R <= value;
                DATA_G <= value;
                DATA_B <= value;
            `endif
        end
        else begin

            /* *****
            /* GAUSSIAN_BLUR_OPERATION */
            /* *****
            `ifdef GAUSSIAN_BLUR_OPERATION
                temp1 = 94742*img_pad[(WIDTH+2) * row + col] + 118318*img_pad[(WIDTH+2) * row + col + 1] + 94742*img_pad[(WIDTH+2) * row + col + 2]
                + 118318*img_pad[(WIDTH+2) * (row+1) + col] + 147761*img_pad[(WIDTH+2) * (row+1) + col + 1] + 118318*img_pad[(WIDTH+2) * (row+1) + col + 2]
                + 94742*img_pad[(WIDTH+2) * (row+2) + col] + 118318*img_pad[(WIDTH+2) * (row+2) + col + 1] + 94742*img_pad[(WIDTH+2) * (row+2) + col + 2];
                value = temp1/1000000;
                temp2 = temp1/1000000;
                if (temp2 > 499999) begin
                    value = value + 1;
                end
                else begin
                    value = value;
                end
                DATA_R = value;
                DATA_G = value;
                DATA_B = value;
            `endif

            /* *****
            /* SOBEL_OPERATION */
            /* *****
            `ifdef SOBEL_OPERATION
                temp1 = (-1)*img_pad[(WIDTH+2) * row + col] + img_pad[(WIDTH+2) * row + col + 2]
                + (-2)*img_pad[(WIDTH+2) * (row+1) + col] + (2)*img_pad[(WIDTH+2) * (row+1) + col + 2]
                + (-1)*img_pad[(WIDTH+2) * (row+2) + col] + img_pad[(WIDTH+2) * (row+2) + col + 2];

                temp2 = (-1)*img_pad[(WIDTH+2) * row + col] + (-2)*img_pad[(WIDTH+2) * row + col + 1] + (-1)*img_pad[(WIDTH+2) * row + col + 2]
                + img_pad[(WIDTH+2) * (row+2) + col] + (2)*img_pad[(WIDTH+2) * (row+2) + col + 1] + img_pad[(WIDTH+2) * (row+2) + col + 2];

                if (temp1 < 0) temp1 = 0;
                else if (temp1 > 255) temp1 = 255;
                if (temp2 < 0) temp2 = 0;
                else if (temp2 > 255) temp2 = 255;
                a = (temp1 > temp2) ? temp1 : temp2;
                b = (temp1 > temp2) ? temp2 : temp1;
                temp3 = a*7/8 + b/2;
                if (a*7/8 > 4 || b/2 == 1) temp3 = temp3 + 1;
                value = (temp3 > a) ? temp3 : a;
                DATA_R = value;
                DATA_G = value;
                DATA_B = value;
            `endif
        end
    end
end
endmodule
```

Hình 10 Module xử lý ảnh

## II.3. Thiết kế khối write unit.

Trong khối write unit, ta khai báo các mảng để lưu giá trị header, giá trị điểm ảnh và các biến dùng trong việc đọc xuất ảnh.

```

/*****
Module for writing .bmp image
*****/
module image_write
#(parameter WIDTH
    = 300,                                // Image width
    HEIGHT = 400,                        // Image height
    INFILE1 = "output.bmp",              // Output image
    INFILE2 = "out.hex",                 // Output hex file
    BMP_HEADER_NUM = 54                  // Header for bmp image
)
(
    input HCLK,                          // Clock
    input HRESETn,                       // Reset active low
    input hsync,                         // Hsync pulse
    input [7:0] DATA_WRITE_R,           // Red 8-bit data (odd)
    input [7:0] DATA_WRITE_G,           // Green 8-bit data (odd)
    input [7:0] DATA_WRITE_B,           // Blue 8-bit data (odd)
    output reg Write_Done,
    output reg File_Closed = 0
);
reg [7:0] BMP_header [0 : BMP_HEADER_NUM - 1]; // BMP header
reg [7:0] out_BMP [0 : WIDTH*HEIGHT*3 - 1]; // Temporary memory for image
integer data_count;                          // Counting data
wire done;                                    // done flag
// counting variables
integer i;
integer k, row, col;
integer fd;

```

Hình 11 Khai báo chân, biến cho module img\_write

### II.3.1. Thiết lập 54-byte header bit mapfile

```

//-----//
// Windows BMP files begin with a 54-byte header:
// Check the website to see the value of this header: http://www.fastgraph.com/help/bmp\_header\_format.html
initial begin
    BMP_header[ 0] = 66;BMP_header[28] =24;
    BMP_header[ 1] = 77;BMP_header[29] = 0;
    BMP_header[ 2] = 54;BMP_header[30] = 0;
    BMP_header[ 3] = 0;BMP_header[31] = 0;
    BMP_header[ 4] = 0;BMP_header[32] = 0;
    BMP_header[ 5] = 0;BMP_header[33] = 0;
    BMP_header[ 6] = 0;BMP_header[34] = 0;
    BMP_header[ 7] = 0;BMP_header[35] = 0;
    BMP_header[ 8] = 0;BMP_header[36] = 0;
    BMP_header[ 9] = 0;BMP_header[37] = 0;
    BMP_header[10] = 54;BMP_header[38] = 0;
    BMP_header[11] = 0;BMP_header[39] = 0;
    BMP_header[12] = 0;BMP_header[40] = 0;
    BMP_header[13] = 0;BMP_header[41] = 0;
    BMP_header[14] = 40;BMP_header[42] = 0;
    BMP_header[15] = 0;BMP_header[43] = 0;
    BMP_header[16] = 0;BMP_header[44] = 0;
    BMP_header[17] = 0;BMP_header[45] = 0;
    BMP_header[18] = 44;BMP_header[46] = 0;
    BMP_header[19] = 1;BMP_header[47] = 0;
    BMP_header[20] = 0;BMP_header[48] = 0;
    BMP_header[21] = 0;BMP_header[49] = 0;
    BMP_header[22] =144;BMP_header[50] = 0;
    BMP_header[23] = 1;BMP_header[51] = 0;
    BMP_header[24] = 0;BMP_header[52] = 0;
    BMP_header[25] = 0;BMP_header[53] = 0;
    BMP_header[26] = 1;
    BMP_header[27] = 0;
end

```

Hình 12 Gán giá trị 54 byte header bmp file

Ở đây để các trường mặt định, chỉ khai báo về kích thước của file bmp ở byte 21:18 và 25:22 bằng cách chuyển giá trị 300 và 400 về số nhị phân rồi nạp vào.

## II.3.2. Thiết kế khối ghi dữ liệu ra file hex và bmp

```
// row and column counting for temporary memory of image
always@(posedge HCLK, negedge HRESETn) begin
    if(!HRESETn) begin
        row <= 0;
        col <= 0;
    end else begin
        if(hsync) begin
            if(col == WIDTH-1) begin
                col <= 0;
                row <= row + 1; // count to obtain row index of the out_BMP temporary memory to save image data
            end else begin
                col <= col + 1; // count to obtain column index of the out_BMP temporary memory to save image data
            end
        end
    end
end
end
```

Hình 13 Module duyệt ảnh theo dòng và cột

```
// Writing RGB888 even and odd data to the temp memory
always@(posedge HCLK, negedge HRESETn) begin
    if(!HRESETn) begin
        for(k=0;k<WIDTH*HEIGHT*3;k=k+1) begin
            out_BMP[k] <= 0;
        end
    end else begin
        if(hsync) begin
            out_BMP[WIDTH*3*row+3*col+2] <= DATA_WRITE_R;
            out_BMP[WIDTH*3*row+3*col+1] <= DATA_WRITE_G;
            out_BMP[WIDTH*3*row+3*col ] <= DATA_WRITE_B;
        end
    end
end
end
```

Hình 14 Module lưu các giá trị RGB vào mảng

```
// data counting
always@(posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
        data_count <= 0;
    end
    else begin
        if(hsync)
            data_count <= data_count + 1; // pixels counting for create done flag
    end
end
assign done = (data_count == WIDTH*HEIGHT-1)? 1'b1: 1'b0; // done flag once all pixels were processed
always@(posedge HCLK, negedge HRESETn)
begin
    if(~HRESETn) begin
        Write_Done <= 0;
    end
    else begin
        Write_Done <= done;
    end
end
end
```

Hình 15 Module đếm để xác định khi nào hoàn tất việc lưu vào mảng

```
//-----//
//-----Write .bmp file -----//
//-----//
initial begin
    fd = $fopen(INFILE1, "wb+");
end
always@(Write_Done) begin // once the processing was done, bmp image will be created
    if(Write_Done == 1'b1) begin
        $writememh(INFILE2, out_BMP);
        for(i=0; i<BMP_HEADER_NUM; i=i+1) begin
            $fwrite(fd, "%c", BMP_header[i][7:0]); // write the header
        end

        for(i=0; i<WIDTH*HEIGHT*3; i=i+3) begin
            // write RGB
            $fwrite(fd, "%c", out_BMP[i ][7:0]);
            $fwrite(fd, "%c", out_BMP[i+1][7:0]);
            $fwrite(fd, "%c", out_BMP[i+2][7:0]);
        end
        $fclose(fd);
        $display("Write file successfully!");
        File_Closed = 1;
    end
end
endmodule
```

Hình 16 Module truyền mảng ra file hex và bmp

### II.3.3. Testbench

Để lựa chọn thuật toán Grayscale hay Sobel ta có thể chọn bằng cách thay đổi giá trị mode bằng 0 để cho ra ảnh xám hay 1 để cho ra ảnh cạnh trắng.

Và chương trình mô phỏng sẽ chạy cho tới khi có tín hiệu hoàn thành việc xuất file của khối write unit.

```

/***** Definition file *****/
/***** Definition file *****/
/***** Definition file *****/
`define INPUTFILENAME      "seaG.hex" // Input file name
`define OUTPUTFILENAME     "seaSB.bmp" // Output file name
`define OUTPOTHEXFILE      "seaSB.hex"
// Choose the operation of code by delete // in the beginning of the selected line

`define SOBEL_OPERATION
//`define GAUSSIAN_BLUR_OPERATION
`define GRAYSCALE_OPERATION

module tb_simulation;

//-----
// Internal Signals
//-----

reg HCLK, HRESETn;
reg mode;
wire      hsync;
wire [ 7 : 0] data_R;
wire [ 7 : 0] data_G;
wire [ 7 : 0] data_B;
wire File_Closed;

image_read
#(.INFILE(`INPUTFILENAME))
u_image_read
(
    .HCLK          (HCLK      ),
    .HRESETn       (HRESETn   ),
    .HSYNC         (hsync     ),
    .mode          (mode      ),
    .DATA_R        (data_R    ),
    .DATA_G        (data_G    ),
    .DATA_B        (data_B    )
);

image_write
#(.INFILE1(`OUTPUTFILENAME), .INFILE2(`OUTPOTHEXFILE))
u_image_write
(
    .HCLK(HCLK),
    .HRESETn(HRESETn),
    .hsync(hsync),
    .DATA_WRITE_R(data_R),
    .DATA_WRITE_G(data_G),
    .DATA_WRITE_B(data_B),
    .File_Closed(File_Closed)
);

..
initial begin
    HCLK = 0;
    mode = 1;
    forever #10 HCLK = ~HCLK;
end

initial begin
    HRESETn = 0;
    #25 HRESETn = 1;
end

always @ (*)
    if(File_Closed)
        #10 $finish;

endmodule

```

Hình 17 Testbench

### III. KIỂM TRA VÀ MÔ PHỎNG

Ảnh được chọn là ảnh RGB có kích thước 300x400 như hình.



*Hình 18 Ảnh đầu vào*

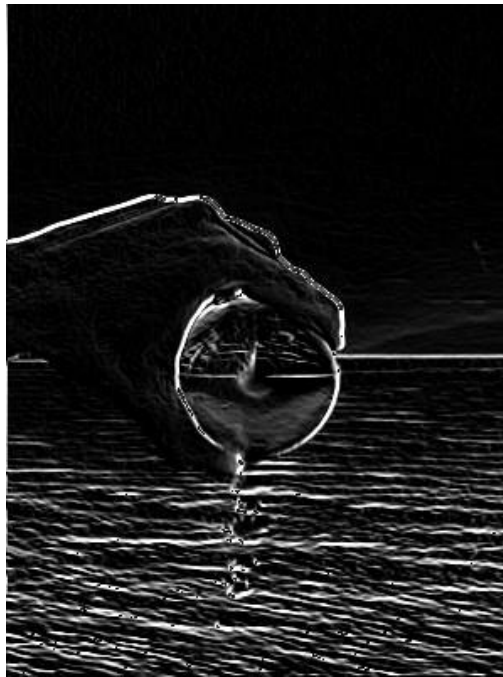
Kết quả ảnh sau khi chọn chế độ chuyển ảnh sang ảnh xám.



*Hình 19 Ảnh sau khi qua thuật toán Grayscale*



Kết quả khi chuyển từ ảnh xám sang ảnh chỉ gồm các cạnh trắng.



*Hình 20 Ảnh qua thuật toán Sobel từ ảnh xám*

Kết quả khi chuyển từ ảnh xám sang ảnh làm mờ, lọc nhiễu.



*Hình 21 Ảnh qua thuật toán Gaussian blur*

#### IV. LINK PROJECT VÀ TÀI LIỆU THAM KHẢO

- Link project: <https://github.com/DgTanDat/Verilog-image-processing>
- Tài liệu tham khảo:
  - <https://www.fpga4student.com/2016/11/image-processing-on-fpga-verilog.html>
  - [https://en.wikipedia.org/wiki/Sobel\\_operator](https://en.wikipedia.org/wiki/Sobel_operator)
  - <https://www.geeksforgeeks.org/image-edge-detection-operators-in-digital-image-processing/>
  - [https://nttuan8.com/bai-5-gioi-thieu-ve-xu-ly-anh/#:~:text=th%C3%AC%20c%C3%A0ng%20s%C3%A1ng.-,Chuy%E1%BB%83n%20h%E1%BB%87%20m%C3%A0u%20c%E1%BB%A7a%20%E1%BA%A3nh,%200.587%20%2B%20b%20\\*%200.114.](https://nttuan8.com/bai-5-gioi-thieu-ve-xu-ly-anh/#:~:text=th%C3%AC%20c%C3%A0ng%20s%C3%A1ng.-,Chuy%E1%BB%83n%20h%E1%BB%87%20m%C3%A0u%20c%E1%BB%A7a%20%E1%BA%A3nh,%200.587%20%2B%20b%20*%200.114.)
  - <https://vinbigdata.com/camera-ai/xu-ly-hinh-anh-trong-python-tu-thuat-toan-den-cong-cu.html>