ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Đồ án môn học Thiết kế luận lý

Image Rotation and Mirroring

GVHD: thầy Huỳnh Phúc Nghị
Lớp: L02
Nhóm: 09

STT	Họ và tên	MSSV
1	Lê Quyết Trung Hiếu	2310952
2	Lê Kim Ngân	2312218
3	Đặng Gia Hưng	2311318

Thành phố Hồ Chí Minh, 12/2025

Mục lục

1 Bảng giải mã các ký tự viết tắt

Ký hiệu	Tên đầy đủ	Giải thích
AXI	Advanced eXtensible Interface	Giao thức bus chuẩn của ARM dùng trong các hệ thống SoC.
FSM	Finite State Machine	Máy trạng thái hữu hạn dùng để điều khiển luồng hoạt động.
UART	Universal Asynchronous Receiver/Transmitter	Giao tiếp nối tiếp bất đồng bộ dùng để truyền và nhận dữ liệu theo từng byte, phổ biến trong việc giao tiếp giữa FPGA/SoC và máy tính hoặc thiết bị ngoại vi.
AW	Address Write	Kênh truyền địa chỉ cho giao dịch ghi.
W	Write Data	Kênh truyền dữ liệu ghi.
B	Write Response	Kênh phản hồi cho giao dịch ghi.
AR	Address Read	Kênh truyền địa chỉ cho giao dịch đọc.
R	Read Data	Kênh truyền dữ liệu đọc.
VALID	Valid	Tín hiệu cho biết dữ liệu hoặc địa chỉ hợp lệ.
READY	Ready	Tín hiệu cho biết phía nhận sẵn sàng.
WLAST	Write Last	Tín hiệu đánh dấu beat cuối của burst ghi.
RLAST	Read Last	Tín hiệu đánh dấu beat cuối của burst đọc.
BRAM	Block RAM	Bộ nhớ khối bên trong FPGA.
DDR	Double Data Rate	Bộ nhớ ngoài tốc độ cao, cho phép truyền dữ liệu ở cả hai cạnh lên và xuống của xung clock, thường được dùng để lưu trữ dữ liệu lớn như ảnh hoặc video.
FIFO	First In First Out	Bộ đệm dữ liệu vào trước ra trước.
ACP	Accelerator Coherency Port	Cổng giao tiếp trong kiến trúc ARM dùng để kết nối các bộ gia tốc phần cứng với hệ thống bộ nhớ đệm của bộ xử lý, cho phép duy trì tính nhất quán dữ liệu (cache coherency) giữa CPU và các khối xử lý tăng tốc.
GPIO	General Purpose Input/Output	Khối vào/ra đa dụng dùng để giao tiếp với các thiết bị bên ngoài như LED, nút nhấn hoặc các tín hiệu điều khiển đơn giản.
S2MM	Stream-to-Memory-Mapped	Khối chuyển đổi dữ liệu từ AXI-Stream sang AXI Memory-Mapped, thường dùng để ghi dữ liệu stream vào bộ nhớ.
MM2S	Memory-Mapped-to-Stream	Khối chuyển đổi dữ liệu từ AXI Memory-Mapped sang AXI-Stream, thường dùng để đọc dữ liệu từ bộ nhớ và truyền đi dưới dạng stream.
HP0	High-Performance Port	Cổng AXI hiệu năng cao dùng để truyền dữ liệu tốc độ lớn giữa bộ xử lý và các khối ngoại vi.
PS	Processing System	Khối xử lý trung tâm của hệ thống SoC, bao gồm CPU, bộ điều khiển bộ nhớ và các ngoại vi tích hợp.
PL	Programmable Logic	Khối logic lập trình được (FPGA) dùng để hiện thực phần cứng tùy chỉnh và tăng tốc xử lý.

Bảng 5: *Bảng giải mã các ký tự viết tắt trong AXI*

2 Bảng tóm tắt

Mục	Nội dung rút gọn
Lý do, yêu cầu, ý nghĩa	Thiết kế khối phần cứng xoay ảnh 90° (CW/CCW) và lật ảnh (ngang/dọc) cho ảnh xám; đọc pixel từ bộ nhớ, ánh xạ tọa độ và ghi ảnh kết quả về bộ nhớ.
Kiến thức liên quan	Ảnh xám dạng ma trận `Width`×`Height`, gốc (0,0) góc trên-trái; địa chỉ hóa tuyến tính; ánh xạ $(old\_row, old\_col) \to (new\_row, new\_col)$; HDL + FSM + giao tiếp bộ nhớ.
Giải pháp và hiện thực	Duyệt (old_row, old_col), tính (new_row, new_col) theo `mode`, ghi sang vùng nhớ đích. – Mirror ngang (lật trái–phải): $new\_row = old\_row,\ new\_col = Width - old\_col - 1$ – Mirror dọc (lật trên–dưới): $new\_row = Height - old\_row - 1,\ new\_col = old\_col$ – Rotate 90° CW: $new\_row = old\_col,\ new\_col = Height - old\_row - 1$ – Rotate 90° CCW: $new\_row = Width - old\_col - 1,\ new\_col = old\_row$
Kết quả đạt/chưa đạt	Mô phỏng Vivado đúng start/valid/done và đúng luồng đọc–ánh xạ–ghi cho 4 chế độ; chưa benchmark trên kit FPGA; chưa hỗ trợ ảnh màu/ảnh lớn.
Hướng mở rộng	Hỗ trợ RGB, tối ưu truy cập bộ nhớ, triển khai/đo đạc trên FPGA thực, thêm thanh ghi cấu hình mode, tối ưu LUT/FF.

Tóm tắt đề tài: Image Rotation and Mirroring (Tiếng Việt)

Item	Condensed content
Motivation, requirements, significance	Design a hardware block for 90° rotation (CW/CCW) and mirroring (horizontal/vertical) on grayscale images. The block reads pixels from memory, remaps coordinates, and writes the transformed image back to memory.
Background	Grayscale image as a `Width`×`Height` matrix (top-left origin); linear addressing; $(old\_row, old\_col) \to (new\_row, new\_col)$ mapping; HDL modular design with an FSM and memory interface.
Solution and implementation	Scan all (old_row, old_col), compute (new_row, new_col) by `mode`, and write to destination. – Horizontal flip: $new\_row = old\_row,\ new\_col = Width - old\_col - 1$ – Vertical flip: $new\_row = Height - old\_row - 1,\ new\_col = old\_col$ – Rotate 90° CW: $new\_row = old\_col,\ new\_col = Height - old\_row - 1$ – Rotate 90° CCW: $new\_row = Width - old\_col - 1,\ new\_col = old\_row$
Achieved / not achieved	Verified in Vivado simulation: correct timing and correct read–map–write behavior for all four modes; not yet benchmarked on real FPGA hardware; no RGB support yet.
Future work	Add RGB support, optimize memory access for larger images, deploy and benchmark on FPGA board, add configuration registers, and reduce LUT/FF via resource sharing.

Concise summary: Image Rotation and Mirroring (English)

3 Giới thiệu

3.1 Bối cảnh và vấn đề

Trong các hệ thống xử lý ảnh hiện đại (camera giám sát, robot, thiết bị IoT, ứng dụng thị giác máy tính trên hệ nhúng), dữ liệu ảnh thu được thường cần qua các bước tiền xử lý trước khi hiển thị hoặc đưa vào các khối xử lý cao hơn (lọc nhiễu, trích đặc trưng, nhận dạng...). Một trong các thao tác tiền xử lý cơ bản nhưng xuất hiện rất thường xuyên là **biến đổi hình học ảnh**: xoay (rotation) và lật gương (mirroring). Lý do là ảnh từ cảm biến/camera có thể bị **lệch hướng** do cách lắp đặt, do cơ chế cầm thiết bị (portrait/landscape), hoặc do quy ước tọa độ giữa các mô-đun khác nhau. Nếu không chuẩn hóa hướng ảnh, các khối xử lý phía sau dễ cho kết quả sai hoặc giảm độ chính xác.
Trong thực tế, xoay ảnh 90° và lật ảnh tuy đơn giản nhưng có thể trở thành "nút thắt" nếu thực hiện bằng phần mềm thuần túy trên vi điều khiển/CPU hạn chế tài nguyên, đặc biệt khi độ phân giải tăng hoặc yêu cầu thời gian thực (real-time). Với mỗi pixel, hệ thống phải đọc dữ liệu, tính toán tọa độ mới và ghi lại, dẫn đến lượng truy cập bộ nhớ lớn và tốn nhiều chu kỳ xử lý. Do đó, việc **thiết kế khối phần cứng chuyên dụng** để thực hiện các phép biến đổi này là cần thiết nhằm:

- tăng tốc xử lý (throughput cao hơn),
- giảm tải cho CPU,

- chuẩn hóa hướng ảnh sớm trong pipeline,
- tạo nền tảng để mở rộng sang các phép biến đổi khác trong tương lai.

Từ nhu cầu trên, đồ án/bài tập lớn tập trung vào bài toán: **thiết kế một hardware block thực hiện xoay ảnh 90° theo chiều kim đồng hồ/ngược chiều kim đồng hồ và lật ảnh ngang/dọc**, làm việc trực tiếp với ảnh xám lưu trong bộ nhớ.

3.2 Vấn đề cụ thể cần giải quyết

Vấn đề cụ thể của đề tài là:

> Thiết kế và kiểm chứng một khối phần cứng có khả năng đọc ảnh xám từ bộ nhớ, thực hiện ánh xạ tọa độ $(x, y \to x', y')$ theo 4 chế độ (Rotate 90° CW, Rotate 90° CCW, Mirror ngang, Mirror dọc), và ghi ảnh kết quả về bộ nhớ đích một cách chính xác, đồng bộ và có thể mở rộng theo tham số kích thước ảnh.

3.3 Mục tiêu và nhiệm vụ

Mục tiêu tổng quát: Xây dựng khối phần cứng biến đổi hình học cơ bản (xoay 90° và mirror) cho ảnh xám, đảm bảo đúng chức năng và có thể mô phỏng/kiểm chứng bằng công cụ thiết kế phần cứng.
Các nhiệm vụ cụ thể:

1. Mô hình hóa dữ liệu ảnh và quy ước hệ tọa độ (gốc ảnh, chiều tăng của x và y), xác định công thức ánh xạ cho từng chế độ.
2. Thiết kế thuật toán remap theo pixel: duyệt từng pixel nguồn (x, y), tính (x', y') và xác định địa chỉ đọc/ghi tuyến tính trong bộ nhớ.
3. Thiết kế kiến trúc phần cứng gồm các khối:
   - Khối điều khiển (FSM/Control Unit) điều phối trình tự đọc–tính–ghi,
   - Bộ đếm tọa độ (x/y counters) để quét ảnh,
   - Khối ánh xạ tọa độ (Coordinate Mapper),
   - Khối tính địa chỉ (Address Generator) cho vùng nhớ nguồn/đích,
   - Giao tiếp bộ nhớ (Memory Interface) và tín hiệu ghi (write enable).
4. Tích hợp cơ chế điều khiển chế độ (mode) và tín hiệu trạng thái (start/valid/done hoặc tương đương).
5. Mô phỏng và kiểm chứng: chạy test với ảnh mẫu/ma trận mẫu, đối chiếu kết quả với kỳ vọng (golden/reference), đảm bảo đúng ở cả 4 chế độ.
6. (Tùy tiến độ) Đánh giá sơ bộ khả năng mở rộng theo kích thước ảnh và tổ chức bộ nhớ.

3.4 Phạm vi và giới hạn của đề tài

Để đảm bảo tính khả thi trong thời gian đồ án/bài tập lớn, nhóm đặt ra phạm vi và các giới hạn như sau:

- **Kiểu ảnh đầu vào:** ảnh xám (grayscale), mỗi pixel là một mẫu cường độ (thường 8-bit hoặc cấu hình tương đương).
- **Các phép biến đổi hỗ trợ:** Rotate 90° CW, Rotate 90° CCW, Mirror ngang (trái–phải), Mirror dọc (trên–dưới). Không xử lý góc xoay tùy ý và không nội suy.
- **Tổ chức dữ liệu:** ảnh lưu tuyến tính theo hàng (row-major), truy cập theo địa chỉ `base + y*Width + x`.
- **Ghi kết quả:** ưu tiên mô hình **out-of-place** (đọc vùng nhớ nguồn, ghi sang vùng nhớ đích) để tránh ghi đè khi xoay ảnh.

- **Môi trường kiểm chứng:** mô phỏng/kiểm chứng trên công cụ thiết kế phần cứng (ví dụ Vivado/ModelSim tùy triển khai). Việc benchmark chi tiết trên kit FPGA thực có thể chưa thực hiện toàn diện (tùy tiến độ).
- **Chưa mở rộng:** ảnh màu RGB/YUV, xử lý nhiều kênh đồng thời, pipeline đa pixel/chu kỳ.

3.5 Ý nghĩa thực tiễn

Ý nghĩa thực tiễn:

- Tạo khối tiền xử lý giúp chuẩn hóa hướng ảnh trước các bước xử lý sau, hữu ích trong hệ nhúng yêu cầu thời gian thực.
- Giảm tải CPU và tăng khả năng đáp ứng khi làm việc với ảnh lớn hoặc tốc độ khung hình cao.
- Có thể tái sử dụng như một IP core cơ bản trong các đồ án FPGA/SoC khác.

Ý nghĩa học thuật:

- Củng cố kiến thức về biểu diễn ảnh số, hệ tọa độ, ánh xạ rời rạc và cách hiện thực phép biến đổi hình học bằng phần cứng.
- Rèn luyện tư duy thiết kế hệ thống số: tách khối chức năng, FSM điều khiển, tính địa chỉ bộ nhớ, đồng bộ tín hiệu và kiểm chứng mô phỏng.

3.6 Cấu trúc của báo cáo

Báo cáo được tổ chức theo hướng từ tổng quan đến hiện thực và kiểm chứng:

- **Chương 1 – Giới thiệu:** bối cảnh, vấn đề, mục tiêu, phạm vi, ý nghĩa và cấu trúc báo cáo.
- **Chương 2 – Cơ sở lý thuyết và kiến thức liên quan:** biểu diễn ảnh xám, quy ước tọa độ, địa chỉ hóa tuyến tính và công thức ánh xạ cho 4 phép biến đổi.
- **Chương 3 – Thiết kế hệ thống/kiến trúc phần cứng:** sơ đồ khối, vai trò từng mô-đun (Control Unit, Mapper, Address Generator, Memory Interface...), luồng dữ liệu và tín hiệu điều khiển.
- **Chương 4 – Hiện thực và mô phỏng kiểm chứng:** triển khai HDL, testbench, kịch bản kiểm thử, kết quả mô phỏng và đối chiếu tính đúng.
- **Chương 5 – Kết luận và hướng phát triển:** tổng kết đạt/chưa đạt và đề xuất mở rộng (RGB, tối ưu bộ nhớ, triển khai thực nghiệm trên FPGA, tăng throughput...).

4 Kiến thức nền tảng

4.1 Tổng quan về xử lý ảnh kỹ thuật số

Xử lý ảnh kỹ thuật số là một lĩnh vực quan trọng trong khoa học máy tính và kỹ thuật điện tử, cho phép thực hiện các phép biến đổi và phân tích trên dữ liệu hình ảnh số hóa. Các ứng dụng của xử lý ảnh bao gồm nhận dạng đối tượng, y tế, viễn thám, và điều khiển tự động.

4.1.1 Biểu diễn ảnh số

Một ảnh số được biểu diễn dưới dạng ma trận hai chiều, trong đó mỗi phần tử được gọi là *pixel* (picture element). Đối với ảnh grayscale (ảnh xám), mỗi pixel được biểu diễn bởi một giá trị cường độ sáng, thường nằm trong khoảng $[0, 255]$ với biểu diễn 8-bit.
Một ảnh có kích thước $H \times W$ (chiều cao H hàng, chiều rộng W cột) được biểu diễn như sau:

$$I(x,y) = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,W-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,W-1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{H-1,0} & p_{H-1,1} & \cdots & p_{H-1,W-1} \end{bmatrix} \tag{1}$$

trong đó $p_{x,y}$ là giá trị pixel tại tọa độ (x, y) với $x \in [0, H-1]$ và $y \in [0, W-1]$.

4.1.2 Lưu trữ ảnh trong bộ nhớ

Trong hệ thống phần cứng, ảnh thường được lưu trữ theo thứ tự *row-major* (theo hàng), nghĩa là các pixel trên cùng một hàng được lưu liên tiếp trong bộ nhớ. Địa chỉ tuyến tính của pixel tại vị trí (x, y) được tính theo công thức:

$$\text{addr}(x,y) = x \times W + y \tag{2}$$

Phương pháp lưu trữ này tối ưu cho việc truy cập tuần tự theo hàng và phù hợp với kiến trúc bộ nhớ của FPGA. Cách lưu trữ row-major này phù hợp với cơ chế truyền dữ liệu tuần tự theo địa chỉ của AXI DMA, giúp đơn giản hóa việc ánh xạ giữa dữ liệu trong bộ nhớ DDR và luồng AXI Stream trong Chương 3.

4.2 Các phép biến đổi hình học trên ảnh

4.2.1 Phép quay ảnh (Image Rotation)

Phép quay ảnh là một phép biến đổi hình học cơ bản, trong đó mỗi pixel của ảnh được ánh xạ sang một vị trí mới theo một góc quay nhất định. Trong đề tài này, chúng ta xét hai trường hợp quay 90:

4.2.1.1 Quay thuận chiều kim đồng hồ 90° (Clockwise Rotation) Khi quay ảnh 90 theo chiều kim đồng hồ, công thức ánh xạ tọa độ như sau:

$$I_{out}[r][c] = I_{in}[c][W-1-r] \tag{3}$$

Trong đó:

- I_{in}: ảnh đầu vào có kích thước $H \times W$
- I_{out}: ảnh đầu ra có kích thước $W \times H$
- (r, c): tọa độ pixel trong ảnh đầu ra

Lưu ý rằng sau phép quay 90, kích thước ảnh đầu ra sẽ bị hoán đổi: chiều cao trở thành chiều rộng và ngược lại.

4.2.1.2 Quay ngược chiều kim đồng hồ 90° (Counter-Clockwise Rotation) Khi quay ảnh 90 ngược chiều kim đồng hồ, công thức ánh xạ như sau:

$$I_{out}[r][c] = I_{in}[H-1-c][r] \tag{4}$$

4.2.1.3 Minh họa phép quay 90° Xét ví dụ với ma trận 3×4:

Ảnh gốc (3×4):

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

Quay CW 90° (4×3):

$$\begin{bmatrix} 9 & 5 & 1 \\ 10 & 6 & 2 \\ 11 & 7 & 3 \\ 12 & 8 & 4 \end{bmatrix}$$

Hình 1: Minh họa phép quay thuận chiều kim đồng hồ 90°

4.2.2 Phép lật ảnh (Image Mirroring)

4.2.2.1 Lật ngang (Horizontal Mirror) Phép lật ngang đảo ngược ảnh theo trục dọc. Công thức ánh xạ:

$$I_{out}[r][c] = I_{in}[r][W-1-c] \tag{5}$$

Kích thước ảnh đầu ra giữ nguyên: $H \times W$.

4.2.2.2 Lật dọc (Vertical Mirror) Phép lật dọc đảo ngược ảnh theo trục ngang. Công thức ánh xạ:

$$I_{out}[r][c] = I_{in}[H-1-r][c] \tag{6}$$

Kích thước ảnh đầu ra giữ nguyên: $H \times W$.

4.2.3 Bảng tổng hợp các phép biến đổi

Phép biến đổi	Công thức	Kích thước đầu ra	Mode
Rotate CW 90°	$I_{out}[r][c] = I_{in}[c][W-1-r]$	$W \times H$	00
Rotate CCW 90°	$I_{out}[r][c] = I_{in}[H-1-c][r]$	$W \times H$	01
Mirror Horizontal	$I_{out}[r][c] = I_{in}[r][W-1-c]$	$H \times W$	10
Mirror Vertical	$I_{out}[r][c] = I_{in}[H-1-r][c]$	$H \times W$	11

Bảng 6: *Tổng hợp công thức ánh xạ tọa độ*

Các giá trị Mode trong bảng trên được sử dụng trực tiếp làm tín hiệu điều khiển trong hệ thống, được PS truyền tới AXI Custom IP Image Rotator thông qua AXI GPIO như trình bày trong Chương 3.

4.3 Giao thức AXI

4.3.1 Tổng quan về AXI

AXI (Advanced eXtensible Interface) là một giao thức bus chuẩn trong kiến trúc AMBA (Advanced Microcontroller Bus Architecture) do ARM phát triển. AXI được thiết kế để hỗ trợ truyền dữ liệu hiệu năng cao trong các hệ thống SoC hiện đại.
Có hai biến thể chính của AXI được sử dụng trong dự án:

- **AXI4 Memory-Mapped**: Dùng cho giao tiếp với bộ nhớ và các ngoại vi có địa chỉ.
- **AXI4-Stream**: Dùng cho truyền dữ liệu luồng (streaming) một chiều với throughput cao.

4.3.2 AXI4 Memory-Mapped

AXI4 Memory-Mapped sử dụng 5 kênh độc lập để thực hiện các giao dịch đọc/ghi:

Kênh	Chiều	Mô tả
Write Address (AW)	Master → Slave	Truyền địa chỉ và thông tin điều khiển cho giao dịch ghi
Write Data (W)	Master → Slave	Truyền dữ liệu ghi
Write Response (B)	Slave → Master	Phản hồi trạng thái giao dịch ghi
Read Address (AR)	Master → Slave	Truyền địa chỉ và thông tin điều khiển cho giao dịch đọc
Read Data (R)	Slave → Master	Truyền dữ liệu đọc về Master

Bảng 7: *Các kênh của AXI4 Memory-Mapped*

4.3.2.1 Cơ chế Burst Transfer

AXI hỗ trợ chế độ burst, cho phép truyền nhiều beat dữ liệu trong một giao dịch duy nhất. Điều này giảm overhead và tăng băng thông hiệu quả.
Các tham số burst quan trọng:

- `AWLEN/ARLEN`: Số lượng beat trong burst (0-255)
- `AWSIZE/ARSIZE`: Kích thước mỗi beat (bytes)
- `WLAST/RLAST`: Đánh dấu beat cuối cùng

4.3.3 AXI4-Stream

AXI4-Stream là giao thức đơn giản hơn, được tối ưu cho truyền dữ liệu luồng một chiều. Không có địa chỉ, chỉ có dữ liệu được truyền tuần tự.

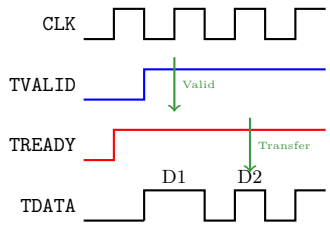
Tín hiệu	Chiều	Mô tả
`TDATA[N:0]`	Master → Slave	Dữ liệu truyền (N+1 bit)
`TVALID`	Master → Slave	Tín hiệu hợp lệ, dữ liệu trên TDATA là đúng
`TREADY`	Slave → Master	Tín hiệu sẵn sàng, Slave có thể nhận dữ liệu
`TLAST`	Master → Slave	Đánh dấu byte cuối cùng trong gói dữ liệu
`TKEEP[N:0]`	Master → Slave	(Tùy chọn) Chỉ định byte nào hợp lệ
`TSTRB[N:0]`	Master → Slave	(Tùy chọn) Chỉ định vị trí byte dữ liệu

Bảng 8: *Các tín hiệu AXI Stream*

Trong phạm vi đề tài, dữ liệu pixel được căn chỉnh theo byte, do đó toàn bộ byte trong `TDATA` luôn hợp lệ. Vì lý do này, các tín hiệu tùy chọn `TKEEP` và `TSTRB` không được sử dụng trong thiết kế AXI Custom IP ở Chương 3.

4.3.3.1 Cơ chế bắt tay (Handshaking)

Truyền dữ liệu chỉ xảy ra khi cả `TVALID` và `TREADY` đều ở mức cao. Cơ chế này cho phép điều khiển luồng dữ liệu linh hoạt và tránh mất dữ liệu.



Hình 2: Sơ đồ thời gian (timing diagram) của AXI Stream

4.4 Kiến trúc bộ nhớ BRAM trên FPGA

4.4.1 Block RAM (BRAM)

Block RAM (BRAM) là tài nguyên bộ nhớ on-chip được nhúng trong FPGA, có khả năng đọc/ghi nhanh với độ trễ thấp. BRAM thường được tổ chức thành các khối có kích thước cố định (ví dụ: 36Kb trên Xilinx 7-series).
Đặc điểm của BRAM:

- Truy cập đồng bộ với clock
- Có thể cấu hình single-port hoặc dual-port
- Hỗ trợ chiều rộng dữ liệu linh hoạt (1-72 bits)
- Độ trễ đọc: 1-2 cycles

4.4.2 Kỹ thuật phân chia bank

Để tối ưu hóa việc sử dụng BRAM và tránh vi phạm giới hạn kích thước của từng khối, ta có thể chia bộ nhớ thành nhiều *bank*. Trong thiết kế này, BRAM được chia thành 4 bank, mỗi bank lưu trữ 1/4 tổng số pixel.
Việc chọn bank được thực hiện dựa trên 2 bit cao nhất của địa chỉ:

$$\text{bank\_sel} = \text{addr}[31:30] \tag{7}$$

$$\text{bank\_addr} = \text{addr}[29:0] \tag{8}$$

Kiến trúc này cho phép truy cập song song và giảm xung đột khi đọc/ghi. Kiến trúc phân chia BRAM thành nhiều bank giúp tăng khả năng đáp ứng băng thông khi tiếp nhận các giao dịch burst từ AXI Master, đồng thời phù hợp với kiến trúc AXI Slave Burst được trình bày trong Chương 3.

4.5 Kiến trúc Zynq SoC

4.5.1 Tổng quan

Zynq SoC của Xilinx kết hợp hai phần chính:

- **Processing System (PS)**: Bao gồm ARM Cortex-A9 dual-core processor, bộ điều khiển bộ nhớ DDR, và các ngoại vi như UART, SPI, I2C.
- **Programmable Logic (PL)**: Phần FPGA có thể lập trình để hiện thực các IP core tùy chỉnh.

4.5.2 Giao tiếp PS-PL

PS và PL giao tiếp với nhau thông qua các cổng AXI:

- **GP (General Purpose)**: 4 cổng AXI4-Lite, dùng cho điều khiển và cấu hình
- **HP (High Performance)**: 4 cổng AXI4, hỗ trợ burst và throughput cao, dùng cho truyền dữ liệu lớn
- **ACP (Accelerator Coherency Port)**: Duy trì tính nhất quán cache

Cổng ACP không được sử dụng trong thiết kế này do hệ thống không yêu cầu duy trì tính nhất quán cache giữa PS và PL. Thay vào đó, các cổng HP được sử dụng để đạt throughput cao trong truyền dữ liệu ảnh thông qua AXI DMA.

4.6 AXI Direct Memory Access (DMA)

AXI DMA là một IP core quan trọng, cho phép truyền dữ liệu trực tiếp giữa bộ nhớ và các IP streaming mà không cần CPU can thiệp.

4.6.1 Chức năng chính

- **MM2S (Memory-Mapped to Stream)**: Đọc dữ liệu từ bộ nhớ DDR và chuyển thành AXI Stream
- **S2MM (Stream to Memory-Mapped)**: Nhận AXI Stream và ghi vào bộ nhớ DDR

4.6.2 Ưu điểm

- Giảm tải cho CPU
- Tăng throughput nhờ truyền dữ liệu trực tiếp
- Hỗ trợ scatter-gather DMA cho các buffer không liên tục
- Có thể hoạt động với interrupts hoặc polling

Trong thiết kế này, AXI DMA được cấu hình ở chế độ simple DMA với các buffer liên tục trong bộ nhớ. Tính năng scatter-gather DMA được xem như một khả năng mở rộng và không được khai thác trong phạm vi của đề tài.

Từ các kiến thức nền tảng đã trình bày, Chương 3 sẽ tập trung vào việc hiện thực kiến trúc phần cứng cụ thể cho hệ thống xử lý ảnh, bao gồm thiết kế AXI Custom IP, tích hợp AXI DMA và xây dựng luồng dữ liệu giữa PS và PL trên nền tảng Zynq SoC.

4.7 Công trình liên quan

4.7.1 Các nghiên cứu về xử lý ảnh trên FPGA

Nhiều nghiên cứu đã được thực hiện về xử lý ảnh trên FPGA nhờ vào khả năng xử lý song song và throughput cao của kiến trúc này.
[1] đề xuất các kỹ thuật tối ưu cho xử lý ảnh trên FPGA, bao gồm sử dụng pipeline và parallel processing. Tuy nhiên, nghiên cứu này tập trung vào các phép biến đổi phức tạp hơn và chưa tối ưu cho các phép quay/lật đơn giản.
[2] so sánh hiệu năng xử lý ảnh giữa GPU và FPGA, cho thấy FPGA có ưu thế về độ trễ thấp và tiêu thụ năng lượng thấp hơn, nhưng phức tạp hơn trong việc lập trình.

4.7.2 Các thiết kế sử dụng giao thức AXI

[3] trình bày thiết kế một hệ thống xử lý video sử dụng AXI4-Stream trên Zynq. Nghiên cứu này cung cấp nền tảng tốt về cách tích hợp PS và PL, nhưng không đề cập chi tiết đến việc xử lý các phép biến đổi hình học.
[4] khảo sát các ứng dụng của FPGA trong điều khiển công nghiệp, trong đó có sử dụng DMA để tăng tốc truyền dữ liệu.

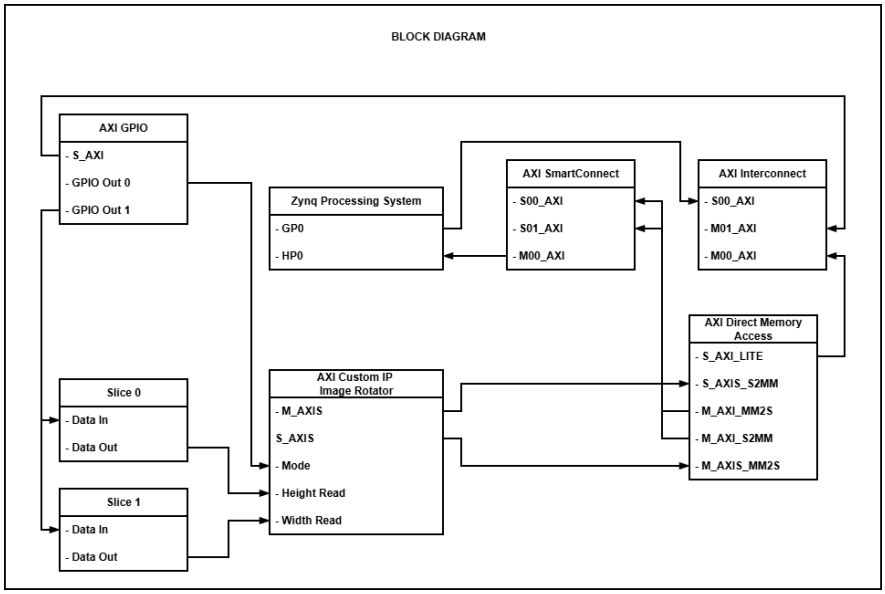
4.7.3 Đóng góp của đề tài

So với các công trình trên, đề tài này đóng góp:

- Thiết kế tối ưu cho các phép quay/lật ảnh với độ phức tạp $O(n)$
- Sử dụng kiến trúc multi-bank BRAM để hỗ trợ ảnh kích thước lớn
- Tích hợp hoàn chỉnh PS-PL với AXI DMA và AXI Stream
- Cung cấp testbench và phương pháp kiểm thử tự động

5 Phần thiết kế

5.1 Sơ đồ khối hệ thống



Hình 3: Sơ đồ khối tổng quan hệ thống

Các thành phần chính

1. **Zynq Processing System (PS)**
   - Bộ xử lý trung tâm của hệ thống, dựa trên CPU ARM.
   - Giao tiếp sử dụng GPIO cho tín hiệu I/O cơ bản và HP0 cho giao tiếp AXI băng thông cao giữa PS và PL.
   - Đóng vai trò là AXI Master, chịu trách nhiệm cấu hình hệ thống, phát sinh các tín hiệu điều khiển và khởi tạo các giao dịch truyền dữ liệu.
2. **AXI GPIO**
   - Kết nối với PS thông qua giao diện S_AXI qua AXI Interconnect.
   - Cung cấp hai ngõ ra GPIO Out:
     - GPIO Out 0: truyền tín hiệu chế độ xoay ảnh (Mode) tới AXI Custom IP.
     - GPIO Out 1: truyền thông tin chiều cao và chiều rộng ảnh tới các thanh ghi `Height Read` và `Width Read`.
3. **AXI SmartConnect**
   - Bao gồm các cổng slave (S00_AXI, S01_AXI) từ PS và cổng master (M00_AXI) tới các khối hạ lưu.
   - Đóng vai trò kết nối và định tuyến các giao dịch AXI, tối ưu hóa băng thông và giảm độ trễ trong hệ thống đa cổng.
4. **AXI Interconnect**
   - Nhận giao dịch từ SmartConnect thông qua cổng slave S00_AXI và phân phối tới các IP thông qua các cổng master.
   - Cho phép một AXI Master giao tiếp với nhiều AXI Slave khác nhau.
5. **AXI Custom IP Image Rotator**

- Giao tiếp dữ liệu thông qua AXI Stream, bao gồm S_AXIS (dữ liệu vào) và M_AXIS (dữ liệu ra).
- Nhận các tín hiệu điều khiển gồm Mode, Height Read và Width Read từ PS.
- Thực hiện chức năng xoay ảnh theo dòng dữ liệu thời gian thực.

6. **AXI Direct Memory Access (DMA)**
   - Sử dụng giao diện AXI Lite cho điều khiển và cấu hình.
   - Thực hiện truyền dữ liệu giữa bộ nhớ DDR và AXI Stream thông qua hai kênh:
     - MM2S (Memory-Mapped-to-Stream): đọc dữ liệu từ bộ nhớ.
     - S2MM (Stream-to-Memory-Mapped): ghi dữ liệu về bộ nhớ.
   - Giảm tải cho PS bằng cách cho phép truyền dữ liệu trực tiếp mà không cần CPU can thiệp.
7. **Slice 0 và Slice 1**
   - Dùng để cắt dữ liệu đọc từ GPIO Out 1.
   - 16 bit cao biểu diễn chiều cao ảnh, 16 bit thấp biểu diễn chiều rộng ảnh.
   - Cung cấp các tham số này cho AXI Custom IP Image Rotator.

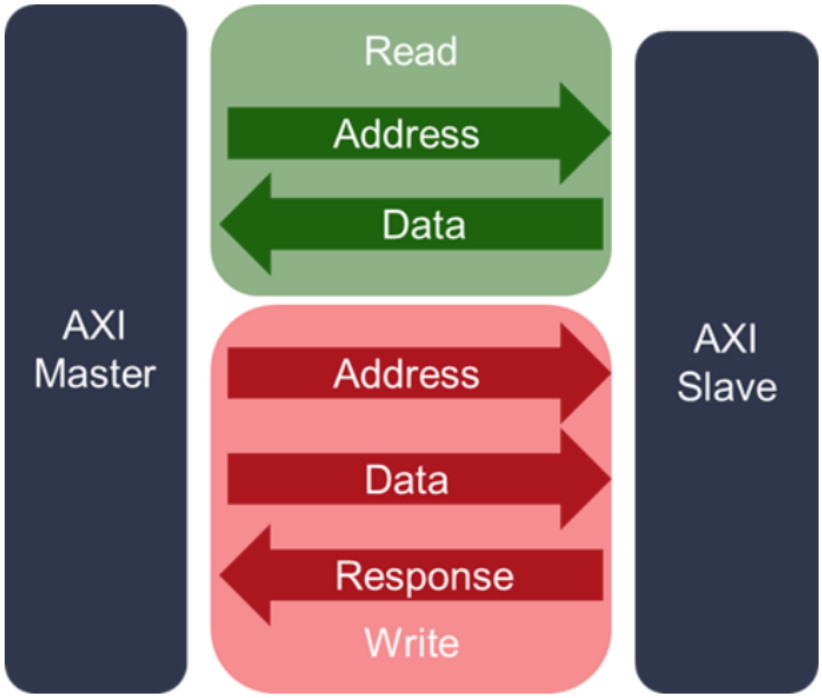
Cách hoạt động tổng thể

- **Khởi tạo từ PS**: PS cấu hình các tham số điều khiển (Mode, Height, Width) thông qua GPIO và khởi tạo AXI DMA để thực hiện truyền dữ liệu ảnh.
- **Dòng chảy dữ liệu hình ảnh**:
  1. DMA đọc dữ liệu ảnh từ bộ nhớ DDR và chuyển thành AXI Stream thông qua kênh MM2S.
  2. Image Rotator xử lý dữ liệu ảnh theo chế độ xoay đã cấu hình và xuất dữ liệu qua M_AXIS.
  3. DMA ghi dữ liệu đã xử lý trở lại bộ nhớ thông qua kênh S2MM.
- **Điều khiển và kết nối AXI**: SmartConnect và Interconnect đảm bảo định tuyến hiệu quả giữa PS, DMA và các IP trong hệ thống.

5.2 Kiến trúc Master–Slave của AXI

5.2.1 Giới thiệu

Hệ thống được thiết kế theo mô hình **AXI Master–Slave** dựa trên chuẩn **AXI4 Memory-Mapped**. AXI Master (Zynq PS hoặc AXI DMA) chịu trách nhiệm khởi tạo các giao dịch đọc và ghi, trong khi AXI Custom IP được hiện thực như một AXI Slave.



Hình 4: Sơ đồ kiến trúc AXI Master–Slave

5.2.2 Luồng hoạt động

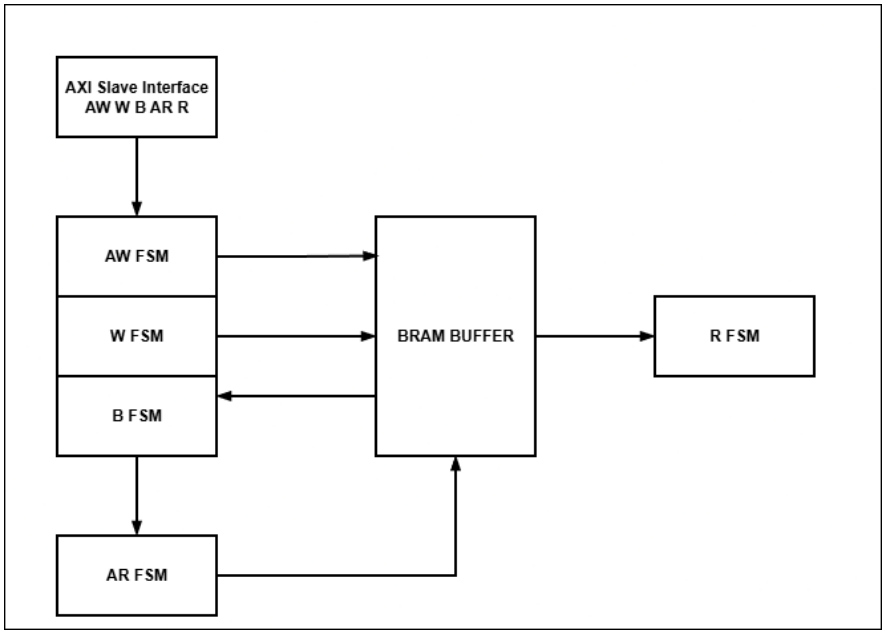
- **Giao dịch đọc (Read)**:
  1. Master gửi địa chỉ đọc qua kênh AR.
  2. Slave xử lý yêu cầu và gửi dữ liệu qua kênh R.
- **Giao dịch ghi (Write)**:
  1. Master gửi địa chỉ ghi qua kênh AW.
  2. Master gửi dữ liệu qua kênh W.
  3. Slave phản hồi trạng thái giao dịch qua kênh B.

5.3 Kiến trúc AXI Slave Burst

5.3.1 Giới thiệu

AXI Slave trong hệ thống được hiện thực để hỗ trợ các giao dịch **burst**, cho phép truyền hoặc nhận nhiều beat dữ liệu trong một giao dịch duy nhất, giúp tăng hiệu năng truyền dữ liệu so với các giao dịch đơn lẻ.
Để tuân thủ chuẩn AXI4 Memory-Mapped, mỗi kênh giao tiếp được hiện thực độc lập bằng các máy trạng thái hữu hạn (FSM), đảm bảo khả năng hoạt động song song và tách biệt giữa các kênh.
AXI Slave bao gồm năm kênh chính:

- **AW** (Write Address): tiếp nhận địa chỉ bắt đầu và thông tin burst ghi.
- **W** (Write Data): tiếp nhận dữ liệu từng beat, xác định beat cuối thông qua tín hiệu `WLAST`.
- **B** (Write Response): phản hồi trạng thái hoàn tất giao dịch ghi.
- **AR** (Read Address): tiếp nhận yêu cầu đọc burst.
- **R** (Read Data): truyền dữ liệu từng beat và phát tín hiệu `RLAST` tại beat cuối.



Hình 5: Sơ đồ kiến trúc bên trong AXI Slave Burst

Tên khối	Chức năng
AXI4 Slave Interface	Khối giao tiếp chuẩn AXI4 Memory-Mapped với AXI Master, chịu trách nhiệm tiếp nhận và phản hồi các giao dịch đọc/ghi dữ liệu dạng burst thông qua năm kênh AXI.
AW FSM	Máy trạng thái hữu hạn quản lý kênh Write Address, tiếp nhận địa chỉ bắt đầu (`AWADDR`) và độ dài burst (`AWLEN`) từ AXI Master.
W FSM	Máy trạng thái hữu hạn quản lý kênh Write Data, tiếp nhận dữ liệu theo từng beat và xác định beat cuối của burst thông qua tín hiệu `WLAST`.
B FSM	Máy trạng thái hữu hạn quản lý kênh Write Response, phát tín hiệu phản hồi khi giao dịch ghi burst hoàn tất.
AR FSM	Máy trạng thái hữu hạn quản lý kênh Read Address, tiếp nhận yêu cầu đọc burst từ AXI Master.
R FSM	Máy trạng thái hữu hạn quản lý kênh Read Data, truyền dữ liệu từng beat và phát tín hiệu `RLAST` tại beat cuối của burst.
BRAM Buffer	Bộ đệm dữ liệu trung tâm dùng để lưu trữ tạm thời dữ liệu burst, giúp tách biệt luồng điều khiển và luồng dữ liệu, từ đó tăng độ ổn định và khả năng mở rộng của hệ thống.

Bảng 9: *Chú thích các khối trong kiến trúc AXI Slave Burst*

5.3.2 Luồng hoạt động

Luồng hoạt động của giao thức AXI bao gồm hai loại giao dịch chính: đọc và ghi dữ liệu.

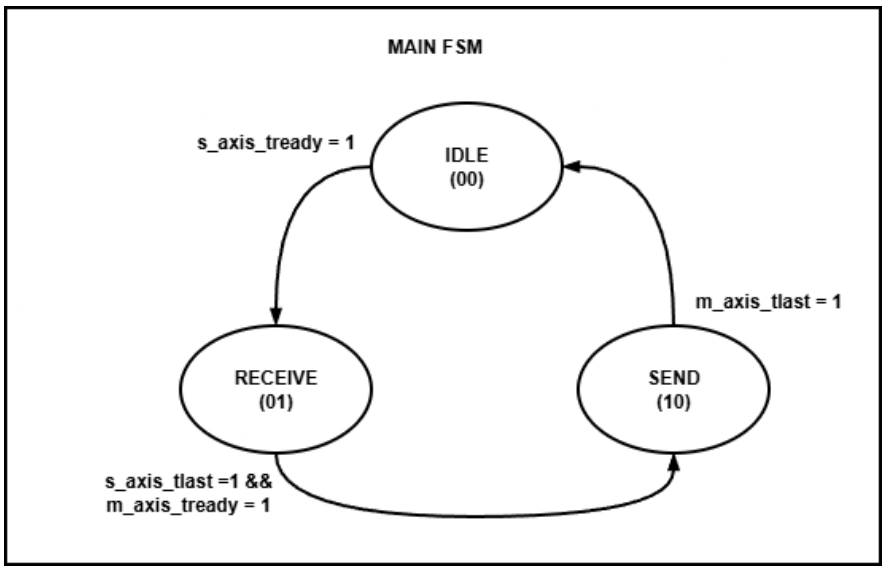
- **Giao dịch đọc (Read)**:
  1. AXI Master gửi địa chỉ đọc thông qua kênh **AR**.
  2. AXI Slave xử lý yêu cầu và trả dữ liệu về cho Master thông qua kênh **R**.
- **Giao dịch ghi (Write)**:
  1. AXI Master gửi địa chỉ ghi thông qua kênh **AW**.
  2. Dữ liệu ghi được truyền qua kênh **W**.
  3. AXI Slave phản hồi trạng thái hoàn thành giao dịch qua kênh **B**.

5.3.3 Luồng dữ liệu của AXI Slave Burst

Luồng dữ liệu trong AXI Slave Burst mô tả đường đi của dữ liệu từ AXI Master vào khối AXI Slave và ngược lại. Luồng dữ liệu được tách biệt hoàn toàn với luồng điều khiển, đảm bảo khả năng truyền dữ liệu liên tục và ổn định trong các giao dịch burst.

- **Luồng dữ liệu ghi (Write Data Flow)**:
  1. AXI Master gửi dữ liệu qua kênh **W** với các tín hiệu `WDATA`, `WVALID` và `WREADY`.
  2. Mỗi lần xảy ra bắt tay `WVALID`–`WREADY`, một beat dữ liệu được tiếp nhận và ghi vào **BRAM Buffer**.
  3. Khối W FSM điều khiển chỉ số beat và địa chỉ ghi dựa trên thông tin từ kênh AW.
  4. Khi tín hiệu `WLAST` được phát, quá trình ghi burst kết thúc.
- **Luồng dữ liệu đọc (Read Data Flow)**:
  1. Sau khi tiếp nhận yêu cầu đọc từ kênh AR, dữ liệu được truy xuất từ **BRAM Buffer**.
  2. Dữ liệu được truyền tới AXI Master qua kênh **R** khi xảy ra bắt tay `RVALID`–`RREADY`.
  3. Khối R FSM quản lý thứ tự các beat dữ liệu trong suốt burst.
  4. Tại beat cuối, tín hiệu `RLAST` được phát để kết thúc giao dịch đọc.

5.4 Sơ đồ máy trạng thái



Hình 6: Sơ đồ máy trạng thái chính (Main FSM)

5.4.1 Giới thiệu

Khối xử lý trung tâm của AXI Custom IP được điều khiển bởi một **máy trạng thái hữu hạn (Finite State Machine – FSM)**. FSM này chịu trách nhiệm điều phối luồng dữ liệu giữa giao diện **AXI Stream Slave (S_AXIS)** và **AXI Stream Master (M_AXIS)**, đảm bảo dữ liệu được tiếp nhận, xử lý và truyền đi đúng thứ tự.
FSM chính bao gồm **ba trạng thái**: `IDLE`, `RECEIVE` và `SEND`, tương ứng với các giai đoạn chờ, nhận dữ liệu và gửi dữ liệu.

5.4.2 Luồng hoạt động của FSM

Luồng hoạt động của FSM được mô tả chi tiết như sau:

1. **Trạng thái IDLE (00):**

   Đây là trạng thái khởi tạo của hệ thống. FSM chờ dữ liệu đầu vào từ giao diện `S_AXIS`. Khi tín hiệu `s_axis_tready = 1` được kích hoạt, FSM chuyển sang trạng thái `RECEIVE` để bắt đầu tiếp nhận dữ liệu.

2. **Trạng thái RECEIVE (01):**

   Ở trạng thái này, FSM tiếp nhận dữ liệu từ AXI Stream Slave. Dữ liệu được đọc theo từng beat khi xảy ra bắt tay `s_axis_tvalid` và `s_axis_tready`.

   Khi tín hiệu `s_axis_tlast = 1` (đánh dấu beat cuối) và đồng thời `m_axis_tready = 1`, FSM xác định rằng toàn bộ dữ liệu đầu vào đã được nhận đầy đủ và sẵn sàng chuyển sang giai đoạn truyền dữ liệu. FSM chuyển sang trạng thái `SEND`.

3. **Trạng thái SEND (10):**

   Trong trạng thái này, dữ liệu đã được xử lý sẽ được truyền ra qua giao diện `M_AXIS`. Dữ liệu được gửi từng beat dựa trên cơ chế bắt tay `m_axis_tvalid` và `m_axis_tready`.

   Khi tín hiệu `m_axis_tlast = 1` được phát, FSM xác định rằng quá trình truyền dữ liệu đã hoàn tất và quay trở lại trạng thái `IDLE`, sẵn sàng cho một chu kỳ xử lý mới.

6 Phần hiện thực

6.1 Hiện thực các tham số tùy chỉnh cho dự án

Để tăng tính linh hoạt và khả năng tái sử dụng của hệ thống, các tham số quan trọng được thiết kế dưới dạng tham số hóa. Các tham số chính bao gồm:

- **MAX_WIDTH**: Chiều rộng tối đa của hình ảnh đầu vào
- **MAX_HEIGHT**: Chiều dài tối đa của hình ảnh đầu vào
- **MAX_PIXELS**: Số lượng pixel tối đa của hình ảnh đầu vào, được tính bằng cách **MAX_PIXELS** = **MAX_WIDTH** × **MAX_HEIGHT**.
- **DATA_WIDTH**: độ rộng dữ liệu pixel trong quá trình xử lý ảnh. Trong thiết kế này, độ rộng dữ liệu được lựa chọn là **8 bit** cho mỗi pixel. Việc sử dụng độ rộng dữ liệu 8 bit đảm bảo tương thích với định dạng ảnh phổ biến và đơn giản hóa quá trình xử lý.
- **INPUT_FILE**, **OUTPUT_FILE**: tên file để hỗ trợ cho việc lưu file trong mô phỏng. Trong thiết kế này, định dạng của **INPUT_FILE** là `.mem`, còn **OUTPUT_FILE** là `.pgm`.
- **MODES**: các chế độ xử lý ảnh. Giá trị của từng chế độ:
  - **MODE_ROTATE_CW**: `00`
  - **MODE_ROTATE_CCW**: `01`
  - **MODE_MIRROR_H**: `10`
  - **MODE_MIRROR_V**: `11`

```
1   DATA_WIDTH 8
2   MAX_WIDTH  128
3   MAX_HEIGHT 128
4   MAX_PIXELS `MAX_HEIGHT * `MAX_WIDTH
5
6   INPUT_FILE "samples_in.mem"
7   OUTPUT_FILE "samples_out.pgm"
8
9   MODE_ROTATE_CW  2'b00'
10  MODE_ROTATE_CCW 2'b01'
11  MODE_MIRROR_H   2'b10'
12  MODE_MIRROR_V   2'b11'
```

6.2 Hiện thực logic xử lý ảnh

Logic xử lý ảnh được hiện thực bên trong AXI Custom IP và hoạt động dựa trên luồng dữ liệu AXI Stream kết hợp với bộ đệm BRAM nội bộ. Quá trình xử lý ảnh được thực hiện theo các bước sau:

Các bước hiện thực

- **Bước 1: Tiếp nhận dữ liệu ảnh**

  Ở trạng thái `RECEIVE`, AXI Slave tiếp nhận các pixel ảnh được truyền từ AXI Master thông qua giao diện AXI Stream đầu vào.

- **Bước 2: Lưu trữ dữ liệu vào BRAM**

  Mỗi pixel ảnh có giá trị trong khoảng 0–255 và được lưu trữ vào bộ nhớ BRAM nội bộ của khối Custom IP. BRAM được tổ chức dưới dạng mảng một chiều, trong đó mỗi phần tử có độ rộng dữ liệu **8 bit** và được đánh địa chỉ tuần tự từ $0, 1, 2, \ldots$.

- **Bước 3: Tính toán địa chỉ mới cho pixel**

  Địa chỉ mới của mỗi pixel được tính toán thông qua hàm `calc_addr`, dựa trên các tham số đầu vào:

  – Tọa độ pixel (x, y) sau khi biến đổi.
  – Tham số `Mode` xác định phép xử lý ảnh đang áp dụng.
  – Chiều rộng và chiều cao của ảnh đầu vào.

  Địa chỉ mới được ánh xạ vào mảng BRAM một chiều theo công thức:

  $$\texttt{new\_addr} = \texttt{new\_row} \times \texttt{width} + \texttt{new\_col}$$

- **Bước 4: Chuẩn bị dữ liệu đầu ra**

  Việc tính toán địa chỉ mới được thực hiện độc lập với xung nhịp `clk` và tín hiệu `reset`, cho phép dữ liệu đầu ra được truy xuất nhanh và sẵn sàng truyền tới AXI Master trong trạng thái `SEND`.

6.2.1 Mã nguồn mẫu hiện thực

```
function [31:0] calc_addr;
    input [15:0] x, y;
    input [1:0] mode;
    input [15:0] w, h;
    begin
        case(mode)
            `MODE_ROTATE_CW: calc_addr = (h - 1 - y) * w + x;
            `MODE_ROTATE_CCW: calc_addr = y * w + (w - 1 - x);
            `MODE_MIRROR_H: calc_addr = x * w + (w - 1 - y);
            `MODE_MIRROR_V: calc_addr = (h - 1 - x) * w + y;
            default:        calc_addr = (h - 1 - y) * w + x;
        endcase
    end
endfunction

always @(*) begin
    out_addr = calc_addr(in_x, in_y, i_mode, img_width, img_height);
end
```

6.3 Hiện thực BRAM Buffer khi xử lý dữ liệu ảnh lớn

6.3.1 Vấn đề

Nếu sử dụng một mảng BRAM một chiều duy nhất (`bram`) để lưu trữ toàn bộ dữ liệu ảnh, thiết kế chỉ phù hợp với các ảnh có kích thước nhỏ. Khi kích thước ảnh tăng lên, yêu cầu bộ nhớ sẽ vượt quá dung lượng BRAM khả dụng trên FPGA, đồng thời làm giảm hiệu suất truy cập bộ nhớ do số lượng phần tử cần xử lý quá lớn.

6.3.2 Giải pháp

Để khắc phục vấn đề này, có thể sử dụng các phương pháp như truy cập bộ nhớ ngoài (DDR) hoặc tổ chức lại bộ nhớ trong FPGA theo cấu trúc dễ quản lý hơn. Trong thiết kế này, nhóm lựa chọn giải pháp **chia nhỏ BRAM Buffer thành nhiều ngân hàng độc lập (multi-bank BRAM)** nhằm tối ưu việc lưu trữ và truy xuất dữ liệu ảnh lớn, đồng thời đảm bảo thời gian xử lý không quá lớn.

6.3.3 Các bước hiện thực

- **Bước 1: Chia BRAM thành các ngân hàng nhỏ**

  Khối BRAM ban đầu được chia thành 4 ngân hàng BRAM độc lập có cùng độ dài. Độ dài của mỗi ngân hàng được xác định như sau:

  $$\text{Chiều dài mỗi BRAM} = \frac{\texttt{MAX\_PIXELS}}{4}$$

  trong đó `MAX_PIXELS` là tham số xác định tổng số pixel tối đa mà hệ thống có thể xử lý, được khai báo trong phần hiện thực tham số.

  Việc chia BRAM giúp giảm kích thước mỗi bộ nhớ con, dễ dàng ánh xạ địa chỉ và tối ưu tài nguyên phần cứng.

- **Bước 2: Phân tách địa chỉ truy cập**

  Địa chỉ ghi và đọc dữ liệu được chia thành hai phần:

  - Hai bit cao nhất của địa chỉ dùng làm **bank selector** để lựa chọn một trong bốn ngân hàng BRAM.
  - Các bit còn lại dùng làm **địa chỉ bên trong ngân hàng** (intra-bank address).

  Cách phân tách này cho phép ánh xạ không gian địa chỉ tuyến tính của ảnh lên nhiều ngân hàng BRAM khác nhau mà không làm thay đổi luồng xử lý dữ liệu.

- **Bước 3: Ghi dữ liệu vào BRAM theo ngân hàng**

  Trong trạng thái **RECEIVE**, dữ liệu pixel từ kênh AXI Stream đầu vào được ghi tuần tự vào ngân hàng BRAM được chọn. Mỗi khi xảy ra bắt tay `VALID/READY`, một beat dữ liệu được lưu vào BRAM tại vị trí tương ứng.

- **Bước 4: Đọc dữ liệu từ BRAM để xử lý và xuất ra**

  Trong trạng thái **SEND**, dữ liệu được đọc từ BRAM dựa trên địa chỉ đầu ra đã tính toán. Ngân hàng BRAM cần truy cập được xác định thông qua bank selector, dữ liệu sau đó được đưa ra kênh AXI Stream đầu ra để tiếp tục xử lý hoặc lưu trữ.

6.3.4 Mã nguồn phần hiện thực

```
    wire [1:0] wr_bank_sel = wr_ptr[31:30];
    wire [29:0] wr_bank_addr = wr_ptr[29:0];

    wire [1:0] rd_bank_sel = out_addr[31:30];
    wire [29:0] rd_bank_addr = out_addr[29:0];

    always @(posedge aclk) begin
```

```
        if (s_axis_tvalid && s_axis_tready && state == S_RECEIVE) begin
            case(wr_bank_sel)
                2'b00': bram_bank0[wr_bank_addr] <= s_axis_tdata;
                2'b01': bram_bank1[wr_bank_addr] <= s_axis_tdata;
                2'b10': bram_bank2[wr_bank_addr] <= s_axis_tdata;
                2'b11': bram_bank3[wr_bank_addr] <= s_axis_tdata;
            endcase
        end
    end

    always @(*) begin
        case(rd_bank_sel)
            2'b00': read_data_reg <= bram_bank0[rd_bank_addr];
            2'b01': read_data_reg <= bram_bank1[rd_bank_addr];
            2'b10': read_data_reg <= bram_bank2[rd_bank_addr];
            2'b11': read_data_reg <= bram_bank3[rd_bank_addr];
        endcase
    end
```

6.4 Hiện thực logic máy trạng thái

6.4.1 Luồng hoạt động

Logic điều khiển của hệ thống được hiện thực dựa trên **máy trạng thái hữu hạn (Finite State Machine – FSM)**, nhằm quản lý luồng dữ liệu AXI Stream, đồng bộ quá trình lưu trữ và xử lý ảnh trong khối Custom IP.
FSM chính bao gồm ba trạng thái, được mô tả như sau:

- **S_IDLE**: Trạng thái khởi tạo và chờ. Tại trạng thái này, các thanh ghi điều khiển như con trỏ ghi (`wr_ptr`) và tọa độ xử lý ảnh (`in_x`, `in_y`) được reset về giá trị ban đầu. Slave sẵn sàng nhận dữ liệu mới bằng cách kích hoạt tín hiệu `s_axis_tready`.
- **S_RECEIVE**: Trạng thái tiếp nhận dữ liệu ảnh từ AXI Master thông qua kênh `S_AXIS`. Mỗi khi xảy ra bắt tay hợp lệ (`s_axis_tvalid` và `s_axis_tready`), một pixel được ghi vào BRAM và con trỏ `wr_ptr` được tăng lên. Khi nhận tín hiệu `s_axis_tlast` hoặc khi số pixel đạt đến `total_pixels`, FSM dừng nhận dữ liệu và chuyển sang trạng thái **S_SEND**.
- **S_SEND**: Trạng thái truyền dữ liệu ảnh đã xử lý ra AXI Master thông qua kênh `M_AXIS`. Dữ liệu được đọc từ BRAM và đưa lên bus `m_axis_tdata` khi `m_axis_tready` được kích hoạt. FSM sử dụng các biến `in_x` và `in_y` để duyệt lần lượt từng pixel theo thứ tự hàng – cột. Khi pixel cuối cùng được truyền, tín hiệu `m_axis_tlast` được phát và FSM quay trở về trạng thái **S_IDLE**.

Việc chuyển trạng thái của FSM được điều khiển bởi các tín hiệu `VALID`, `READY` và `LAST`, đảm bảo tuân thủ chuẩn giao tiếp AXI Stream và duy trì tính toàn vẹn của dữ liệu trong suốt quá trình truyền nhận.

6.4.2 Mã nguồn phần hiện thực

```
always @(posedge aclk or negedge aresetn) begin
    if (!aresetn) begin
        state <= S_IDLE;
        s_axis_tready <= 0;
        m_axis_tvalid <= 0;
        m_axis_tlast <= 0;
        m_axis_tdata <= 0;
        wr_ptr <= 0;
        in_x <= 0;
        in_y <= 0;
    end else begin
```

```
        case(state)
            S_IDLE: begin
                wr_ptr <= 0;
                in_x <= 0;
                in_y <= 0;
                s_axis_tready <= 1;
                m_axis_tvalid <= 0;
                m_axis_tlast <= 0;
                state <= S_RECEIVE;
            end

            S_RECEIVE: begin
                if (s_axis_tvalid && s_axis_tready) begin
                    if (s_axis_tlast || wr_ptr == total_pixels - 1) begin
                        s_axis_tready <= 0;
                        state <= S_SEND;
                        in_x  <= 0;
                        in_y  <= 0;
                    end else begin
                        wr_ptr <= wr_ptr + 1;
                    end
                end
            end
            S_SEND: begin
                if (m_axis_tready) begin
                    m_axis_tdata <= read_data_reg;
                    m_axis_tvalid <= 1;
                    m_axis_tlast <= (in_x == new_height - 1 && in_y == new_width - 1);
                    if (in_x == new_height - 1 && in_y == new_width - 1) begin
                        state <= S_IDLE;
                    end else if (in_y == new_width - 1) begin
                        in_y <= 0;
                        in_x <= in_x + 1;
                    end else begin
                        in_y <= in_y + 1;
                    end
                end
            end
            default: state <= S_IDLE;
        endcase
    end
end
```

6.5 Hiện thực phần mô phỏng

Phần mô phỏng được xây dựng nhằm kiểm tra tính đúng đắn của khối **AXI Image Rotator** trước khi triển khai trên phần cứng thực tế. Testbench mô phỏng đầy đủ giao tiếp **AXI-Stream Slave** ở ngõ vào và **AXI-Stream Master** ở ngõ ra, đồng thời kiểm tra các chế độ xử lý ảnh khác nhau.

6.5.1 Cấu trúc testbench

Testbench bao gồm các thành phần chính sau:

- Tạo xung clock và tín hiệu reset cho hệ thống.
- Mô phỏng giao tiếp AXI-Stream đầu vào (S_AXIS).
- Mô phỏng giao tiếp AXI-Stream đầu ra (M_AXIS).
- Nạp dữ liệu ảnh từ file và ghi kết quả ra file đầu ra.

Khối `axi_image_rotator` được khởi tạo như *Design Under Test (DUT)*, với các tham số `MAX_WIDTH` và `MAX_HEIGHT` được truyền vào thông qua file `defines.v`.

6.5.2 Tạo xung clock và reset

Xung clock được tạo với chu kỳ 10 ns (tần số 100 MHz), phù hợp với các thiết kế AXI thông dụng. Tín hiệu reset chủ động mức thấp (`aresetn`) được giữ ở trạng thái reset trong giai đoạn đầu mô phỏng để đảm bảo toàn bộ hệ thống khởi tạo đúng trạng thái ban đầu.

6.5.3 Nạp dữ liệu ảnh đầu vào

Dữ liệu ảnh được đọc từ file đầu vào thông qua lệnh `$readmemh`. Bộ đệm `buffer_din` được sử dụng để lưu trữ dữ liệu, trong đó:

- Phần tử đầu tiên chứa chiều cao ảnh.
- Phần tử thứ hai chứa chiều rộng ảnh.
- Các phần tử tiếp theo chứa giá trị pixel ảnh xám 8-bit.

Sau khi đọc header ảnh, testbench tự động tính toán tổng số pixel cần truyền, giúp mô phỏng linh hoạt với nhiều kích thước ảnh khác nhau mà không cần chỉnh sửa mã nguồn.

6.5.4 Mô phỏng luồng dữ liệu ghi (AXI Slave)

Dữ liệu ảnh được gửi tới DUT thông qua giao diện AXI-Stream Slave. Testbench chỉ phát tín hiệu `s_axis_tvalid` khi DUT phản hồi `s_axis_tready`, đảm bảo tuân thủ cơ chế bắt tay của chuẩn AXI-Stream.
Mỗi chu kỳ bắt tay tương ứng với một pixel được truyền. Tín hiệu `s_axis_tlast` được kích hoạt tại pixel cuối cùng của ảnh, đánh dấu kết thúc luồng dữ liệu đầu vào.

6.5.5 Mô phỏng luồng dữ liệu đọc (AXI Master)

Sau khi xử lý xong dữ liệu ảnh, DUT truyền kết quả ra giao diện AXI-Stream Master. Testbench luôn giữ `m_axis_tready` ở mức sẵn sàng, đảm bảo không xảy ra hiện tượng nghẽn dữ liệu.
Mỗi pixel đầu ra được ghi vào file kết quả thông qua lệnh `$fwrite`. Tín hiệu `m_axis_tlast` được sử dụng để xác định pixel cuối cùng của ảnh đầu ra, tại thời điểm đó mô phỏng sẽ kết thúc.

6.5.6 Mã nguồn phần hiện thực

```
`timescale 1ns/1ps
`include "defines.v"

module tb_axi_image_rotator;
    reg aclk, aresetn;
    reg [1:0] i_mode;

    reg [7:0] s_axis_tdata;
    reg s_axis_tvalid, s_axis_tlast;
    wire s_axis_tready;

    wire [7:0] m_axis_tdata;
    wire m_axis_tvalid, m_axis_tlast;
    reg m_axis_tready;

    reg [15:0] tb_dynamic_height;
    reg [15:0] tb_dynamic_width;

    axi_image_rotator #(
        .MAX_WIDTH(`MAX_WIDTH),
        .MAX_HEIGHT(`MAX_HEIGHT)
    ) dut (
        .aclk(aclk),
        .aresetn(aresetn),
```

```
            .i_mode(i_mode),
            .img_height(tb_dynamic_height),
            .img_width(tb_dynamic_width),

            .s_axis_tdata(s_axis_tdata),
            .s_axis_tvalid(s_axis_tvalid),
            .s_axis_tlast(s_axis_tlast),
            .s_axis_tready(s_axis_tready),

            .m_axis_tdata(m_axis_tdata),
            .m_axis_tvalid(m_axis_tvalid),
            .m_axis_tlast(m_axis_tlast),
            .m_axis_tready(m_axis_tready)
        );

        initial begin aclk=0; forever #5 aclk=~aclk; end

        reg [31:0] buffer_din [0:`MAX_WIDTH*`MAX_HEIGHT-1];

        integer i;
        integer pixel_count;
        integer total_pixels;

        integer fout;

        initial begin
            aresetn=0; m_axis_tready=1; s_axis_tvalid=0;

            i_mode=`MODE_ROTATE_CW;
            #50 aresetn=1;

            $readmemh(`INPUT_FILE, buffer_din);
            tb_dynamic_height = buffer_din[0];
            tb_dynamic_width = buffer_din[1];
            total_pixels = tb_dynamic_height * tb_dynamic_width;

            fout = $fopen(`OUTPUT_FILE,"w");
            $fwrite(fout,"P2\n%d %d\n255\n", tb_dynamic_width, tb_dynamic_height);
            //$fwrite(fout,"P2\n%d %d\n255\n" tb_dynamic_height,tb_dynamic_width);

            $display("=== Sending %0d pixels ===", total_pixels);
            for (i=0; i<total_pixels; i=i+1) begin
                @(posedge aclk);
                while (!s_axis_tready) @(posedge aclk);
                s_axis_tvalid <= 1;
                s_axis_tdata <= buffer_din[i + 2][7:0];
                s_axis_tlast <= (i==total_pixels - 1);
                $display("Sending pixel[%0d] = %0d", i, buffer_din[i + 2][7:0]);
            end
            @(posedge aclk);
            s_axis_tvalid <= 0;
            s_axis_tlast <= 0;

            pixel_count = 0;
            $display("=== Receiving pixels ===");
            while (pixel_count < total_pixels) begin
                @(posedge aclk);
                if (m_axis_tvalid && m_axis_tready) begin
                    $fwrite(fout,"%02X\n", m_axis_tdata);
                    $write("%3d ", m_axis_tdata);
                    if ((pixel_count+1)%tb_dynamic_height==0) $write("\n");
                    pixel_count = pixel_count + 1;
```

```
                if (m_axis_tlast) begin
                    $display("=== DONE, %s written ===", `OUTPUT_FILE);
                    $fclose(fout);
                    $finish;
                end
            end
        end
    end
endmodule
```

6.6 Hiện thực block design cho hệ thống

Block design của hệ thống được xây dựng trên nền tảng SoC, kết hợp giữa khối **Processing System (PS)** và **Programmable Logic (PL)** nhằm tận dụng ưu điểm của cả xử lý phần mềm và tăng tốc phần cứng.

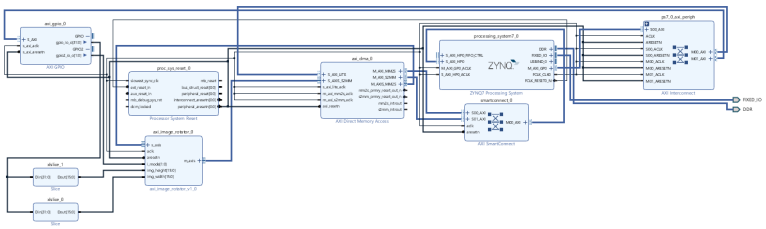
6.6.1 Kiến trúc tổng thể

- **PS (Processing System)**: Chịu trách nhiệm khởi tạo hệ thống, cấu hình các thanh ghi điều khiển, và quản lý luồng dữ liệu thông qua các cổng AXI.
- **AXI DMA**: Đóng vai trò trung gian truyền dữ liệu ảnh giữa bộ nhớ DDR trong PS và khối xử lý ảnh trong PL. DMA hoạt động với hai kênh chính: `MM2S` để truyền dữ liệu từ bộ nhớ ra PL và `S2MM` để ghi dữ liệu xử lý trở lại bộ nhớ.
- **Custom Image Rotator IP**: Khối IP do nhóm tự thiết kế, giao tiếp với AXI DMA thông qua chuẩn AXI Stream. IP này thực hiện việc nhận dữ liệu ảnh, lưu trữ vào BRAM nội bộ, xử lý ảnh theo mode cấu hình và truyền kết quả ra ngoài.
- **BRAM Buffer**: Bộ nhớ trong của Custom IP, dùng để lưu trữ tạm thời dữ liệu ảnh và hỗ trợ truy xuất nhanh trong quá trình xử lý.

6.6.2 Luồng dữ liệu trong hệ thống

Luồng dữ liệu chính của hệ thống diễn ra theo các bước sau:

1. Dữ liệu ảnh đầu vào được lưu trong bộ nhớ DDR và được PS cấu hình thông qua phần mềm SDK.
2. AXI DMA sử dụng kênh `MM2S` để đọc dữ liệu ảnh từ DDR và truyền dưới dạng AXI Stream tới khối Custom Image Processing IP trong PL.
3. Khối Custom IP tiếp nhận dữ liệu, lưu trữ vào BRAM Buffer, thực hiện xử lý ảnh theo thuật toán đã thiết kế và tạo ra dữ liệu ảnh đầu ra.
4. Dữ liệu sau xử lý được truyền ngược lại AXI DMA thông qua kênh `S2MM` và được ghi trở lại bộ nhớ DDR.
5. PS tiếp tục đọc dữ liệu kết quả từ DDR để hiển thị, lưu trữ hoặc phục vụ các tác vụ tiếp theo.



Hình 7: Block design được thiết kế trong Vivado

6.7 Hiện thực cho SDK

Phần mềm SDK được xây dựng trên nền tảng **Vitis SDK**, đóng vai trò điều khiển hệ thống, cấu hình phần cứng trong PL và quản lý quá trình truyền dữ liệu ảnh giữa bộ nhớ DDR và khối xử lý ảnh thông qua AXI DMA.

6.7.1 Khởi tạo hệ thống và ngoại vi

Chương trình bắt đầu bằng việc khởi tạo nền tảng và các driver cần thiết, bao gồm **AXI DMA** và **AXI GPIO**.
AXI GPIO được sử dụng để truyền các tham số cấu hình từ PS sang PL, bao gồm chế độ xử lý ảnh và kích thước ảnh đầu vào. AXI DMA được cấu hình ở chế độ *Simple Transfer*, với hai kênh:

- **MM2S** (Memory-Mapped to Stream): truyền dữ liệu ảnh từ DDR tới PL.
- **S2MM** (Stream to Memory-Mapped): nhận dữ liệu ảnh đã xử lý từ PL về DDR.

Các ngắt của DMA được vô hiệu hóa, thay vào đó chương trình sử dụng cơ chế polling để đơn giản hóa việc kiểm soát luồng dữ liệu.

6.7.2 Tự động phát hiện kích thước ảnh

Dữ liệu ảnh đầu vào được lưu dưới dạng một mảng byte, trong đó 8 byte đầu tiên chứa thông tin header của ảnh. SDK tự động đọc:

- 4 byte đầu: chiều cao ảnh (`Height`)
- 4 byte tiếp theo: chiều rộng ảnh (`Width`)

Từ hai thông số này, chương trình tính toán tổng số pixel cần xử lý. Cơ chế kiểm tra kích thước được sử dụng để đảm bảo dữ liệu không vượt quá vùng nhớ đệm đã cấp phát, tránh lỗi tràn bộ nhớ.

6.7.3 Cấu hình tham số xử lý cho phần cứng

Các tham số xử lý ảnh được truyền từ PS sang PL thông qua AXI GPIO:

- **Chế độ xử lý ảnh**: được mã hóa bằng 2 bit, bao gồm xoay ảnh theo chiều kim đồng hồ, ngược chiều kim đồng hồ, lật ngang và lật dọc.
- **Kích thước ảnh**: được đóng gói trong một thanh ghi 32-bit, với 16 bit cao biểu diễn chiều cao và 16 bit thấp biểu diễn chiều rộng ảnh.

Việc sử dụng GPIO giúp đơn giản hóa giao tiếp điều khiển, đồng thời tách biệt rõ ràng giữa luồng dữ liệu (AXI Stream) và luồng điều khiển.

6.7.4 Quản lý bộ nhớ và tính nhất quán dữ liệu

Trước khi kích hoạt DMA, bộ nhớ cache của CPU được xử lý nhằm đảm bảo tính nhất quán dữ liệu:

- Flush cache vùng chứa dữ liệu ảnh đầu vào, đảm bảo DMA đọc đúng dữ liệu từ DDR.
- Invalidate cache vùng bộ đệm nhận dữ liệu, tránh việc CPU đọc dữ liệu cũ sau khi DMA ghi kết quả mới.

Bước này là bắt buộc trong các hệ thống PS–PL để tránh lỗi dữ liệu do không đồng bộ cache.

6.7.5 Truyền dữ liệu bằng AXI DMA

Quá trình truyền dữ liệu được thực hiện theo thứ tự:

1. Kích hoạt kênh **S2MM** để sẵn sàng nhận dữ liệu từ PL.
2. Kích hoạt kênh **MM2S** để gửi dữ liệu ảnh từ DDR tới khối xử lý ảnh.

Cách cấu hình này đảm bảo PL có thể truyền dữ liệu trả về ngay khi sẵn sàng, tránh mất dữ liệu trong quá trình xử lý.
Chương trình sử dụng cơ chế polling để chờ cả hai kênh DMA hoàn tất truyền dữ liệu trước khi tiếp tục các bước xử lý tiếp theo.

6.7.6 Nhận và kiểm tra kết quả

Sau khi DMA hoàn tất, SDK xác định lại kích thước ảnh đầu ra. Trong trường hợp xoay ảnh 90°, chiều cao và chiều rộng của ảnh sẽ được hoán đổi.
Dữ liệu kết quả được đọc từ bộ đệm nhận trong DDR và in ra màn hình dưới dạng ma trận pixel, nhằm kiểm tra tính đúng đắn của thuật toán xử lý ảnh.

6.7.7 Mã nguồn hiện thực

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
#include "xaxidma.h"
#include "xgpio.h"
#include "xil_cache.h"
#include "in_data.h" // Include input data file

#define ROTATE_MODE_CW  0
#define ROTATE_MODE_CCW 1
#define MIRROR_MODE_H  2
#define MIRROR_MODE_V  3

#define MAX_HEIGHT 512
#define MAX_WIDTH 512

#define DMA_DEV_ID        XPAR_AXIDMA_0_DEVICE_ID
#define GPIO_DEV_ID       XPAR_AXI_GPIO_0_DEVICE_ID
#define MAX_BUFFER_SIZE   MAX_HEIGHT * MAX_WIDTH

u8 RxBuffer[MAX_BUFFER_SIZE] __attribute__ ((aligned (32)));

XAxiDma AxiDma;
XGpio Gpio;

int main() {
    init_platform();
    xil_printf("\r\n===== IMAGE ROTATION SDK =====\r\n");
```

```
30      XGpio_Initialize(&Gpio, GPIO_DEV_ID);
31      XGpio_SetDataDirection(&Gpio, 1, 0x00000000);
32      XGpio_SetDataDirection(&Gpio, 2, 0x00000000);
33
34      XAxiDma_Config *CfgPtr = XAxiDma_LookupConfig(DMA_DEV_ID);
35      XAxiDma_CfgInitialize(&AxiDma, CfgPtr);
36      XAxiDma_IntrDisable(&AxiDma, XAXIDMA_IRQ_ALL_MASK, XAXIDMA_DEVICE_TO_DMA);
37      XAxiDma_IntrDisable(&AxiDma, XAXIDMA_IRQ_ALL_MASK, XAXIDMA_DMA_TO_DEVICE);
38
39      u32 *header_ptr = (u32 *)raw_image_file;
40
41      u32 detected_h = header_ptr[0];
42      u32 detected_w = header_ptr[1];
43
44      xil_printf("[>]Detected Header: Height = %d, Width = %d\r\n", detected_h,
            detected_w);
45      u32 image_data_size = detected_h * detected_w;
46      if (image_data_size > MAX_BUFFER_SIZE) {
47          xil_printf("[x]Error: Image too big for RxBuffer!\r\n");
48          return -1;
49      }
50      u8 *pixel_data_ptr = (u8 *)(raw_image_file + 8);
51      u32 mode = 0;
52      XGpio_DiscreteWrite(&Gpio, 2, mode);
53      u32 size_config = (detected_h << 16) | detected_w;
54      XGpio_DiscreteWrite(&Gpio, 1, size_config);
55      Xil_DCacheFlushRange((UINTPTR)raw_image_file, 8 + image_data_size);
56      Xil_DCacheInvalidateRange((UINTPTR)RxBuffer, image_data_size);
57      XAxiDma_SimpleTransfer(&AxiDma, (UINTPTR)RxBuffer, image_data_size,
            XAXIDMA_DEVICE_TO_DMA);
58
59      XAxiDma_SimpleTransfer(&AxiDma, (UINTPTR)pixel_data_ptr, image_data_size,
            XAXIDMA_DMA_TO_DEVICE);
60      while (XAxiDma_Busy(&AxiDma, XAXIDMA_DMA_TO_DEVICE) || XAxiDma_Busy(&AxiDma,
            XAXIDMA_DEVICE_TO_DMA));
61      int out_h = detected_h;
62      int out_w = detected_w;
63      if (mode == 0 || mode == 1) {
64          out_h = detected_w;
65          out_w = detected_h;
66      }
67      xil_printf("====== Output Data (%dx%d) =====\r\n", out_h, out_w);
68      Xil_DCacheInvalidateRange((UINTPTR)RxBuffer, image_data_size);
69      for(int r = 0; r < out_h; r++){
70          for(int c = 0; c < out_w; c++){
71              xil_printf("%02d ", RxBuffer[r * out_w + c]);
72          }
73          xil_printf("\r\n");
74      }
75
76      xil_printf("===== Done Output Data ======\r\n");
77
78      cleanup_platform();
79      return 0;
80  }
```

6.8 Kiểm thử

6.8.1 Chuẩn bị các testcase mẫu

Để đánh giá tính đúng đắn và ổn định của hệ thống, nhóm xây dựng một tập **testcase mẫu** với các đặc điểm sau:

- Ảnh đầu vào dạng **ảnh xám (grayscale)** với độ sâu dữ liệu 8-bit, giá trị pixel nằm trong khoảng $0 \sim 255$.
- Kích thước ảnh đa dạng, bao gồm các ảnh nhỏ để dễ kiểm tra bằng mắt và các ảnh lớn để đánh giá khả năng xử lý dữ liệu lớn của hệ thống.
- Mỗi file ảnh được tổ chức theo định dạng:
  - 2 từ đầu: chiều cao (`Height`) và chiều rộng (`Width`) của ảnh.
  - Các từ tiếp theo: dữ liệu pixel được sắp xếp theo thứ tự dòng (row-major).
- Các testcase được kiểm tra với đầy đủ các chế độ xử lý ảnh: xoay ảnh 90° theo chiều kim đồng hồ (CW), xoay ảnh 90° ngược chiều kim đồng hồ (CCW), lật ảnh theo chiều ngang (Mirror Horizontal), và lật ảnh theo chiều dọc (Mirror Vertical).

Tập testcase này được sử dụng thống nhất cho cả mô phỏng và kiểm thử trên phần cứng nhằm đảm bảo tính nhất quán của kết quả.

6.8.2 Kiểm thử bằng mô phỏng

6.8.2.a Kiểm tra các chức năng cơ bản

Các chức năng được kiểm tra

- Kiểm tra khả năng đọc tuần tự các pixel từ ảnh đầu vào thông qua giao diện AXI Stream.
- Kiểm tra dữ liệu pixel sau xử lý được ghi đầy đủ và đúng thứ tự vào file ảnh đầu ra.
- Kiểm tra tính toàn vẹn dữ liệu, đảm bảo không xảy ra hiện tượng mất pixel trong quá trình truyền nhận dữ liệu giữa Master và Slave.
- Kiểm tra tín hiệu kết thúc khung hình (`TLAST`) được phát đúng tại pixel cuối cùng.
- Kiểm tra cơ chế bắt tay `VALID/READY` hoạt động đúng, đảm bảo không truyền dữ liệu khi phía nhận chưa sẵn sàng.

Hình 8: Ảnh đầu vào grayscale có kích thước 4×4

Ảnh đầu vào được lựa chọn có kích thước nhỏ nhằm thuận tiện cho việc kiểm tra các chức năng cơ bản của hệ thống, bao gồm quá trình truyền nhận pixel, xử lý địa chỉ và ghi dữ liệu đầu ra.

Hình 9: Nội dung file `.mem` được chuyển đổi từ ảnh grayscale

Định dạng file `.mem` được sử dụng trong mô phỏng như sau:

- Dòng đầu tiên là header nhằm nhận biết định dạng file `.mem`.
- Hai dòng tiếp theo lần lượt biểu diễn chiều cao và chiều rộng của ảnh, được mã hóa dưới dạng số HEX với độ rộng 8-bit.
- Các dòng tiếp theo là giá trị pixel của ảnh grayscale, mỗi pixel có độ rộng dữ liệu 8-bit và giá trị nằm trong khoảng từ 0 đến 255.

Sau khi chạy mô phỏng trên phần mềm Vivado 2018.3, thu được các kết quả như sau.

```
=== Sending 16 pixels ===
Sending pixel[0] = 0
Sending pixel[1] = 85
Sending pixel[2] = 170
Sending pixel[3] = 255
Sending pixel[4] = 0
Sending pixel[5] = 85
Sending pixel[6] = 170
Sending pixel[7] = 255
Sending pixel[8] = 0
Sending pixel[9] = 85
Sending pixel[10] = 170
Sending pixel[11] = 255
Sending pixel[12] = 0
Sending pixel[13] = 85
Sending pixel[14] = 170
Sending pixel[15] = 255
=== Receiving pixels ===
```

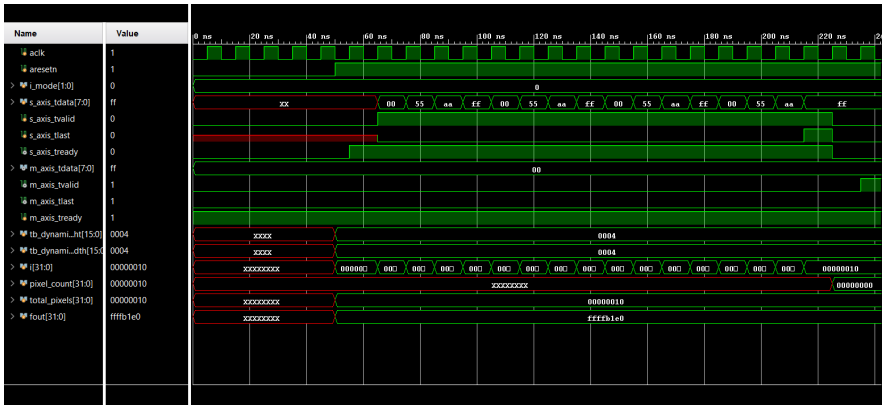
Hình 10: Các pixel được gửi từ Master đến Slave trên cửa sổ Tcl Console

Các pixel được truyền từ Master đến Slave với tổng số 16 pixel, đúng bằng kích thước ảnh đầu vào. Thứ tự các pixel được in ra trên màn hình console cho thấy dữ liệu được gửi tuần tự và không bị mất mát.

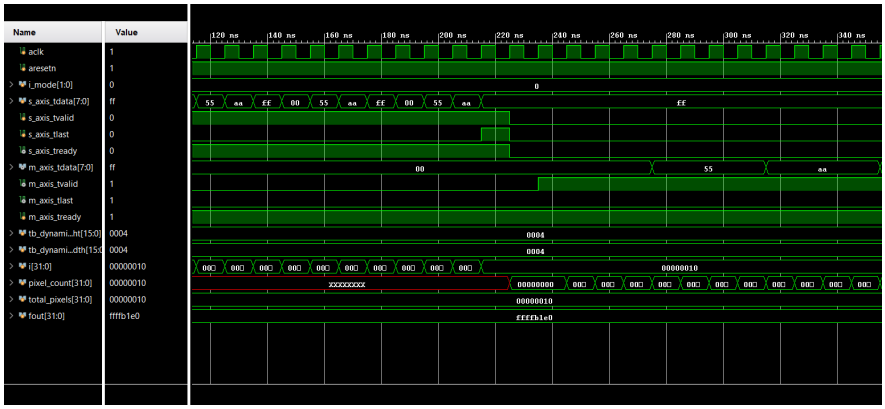
```
=== Receiving pixels ===
  0   0   0   0
 85  85  85  85
170 170 170 170
255 255 255 255
=== DONE, samples_out.pgm written ===
```

Hình 11: Các pixel được gửi từ Slave đến Master sau khi xử lý

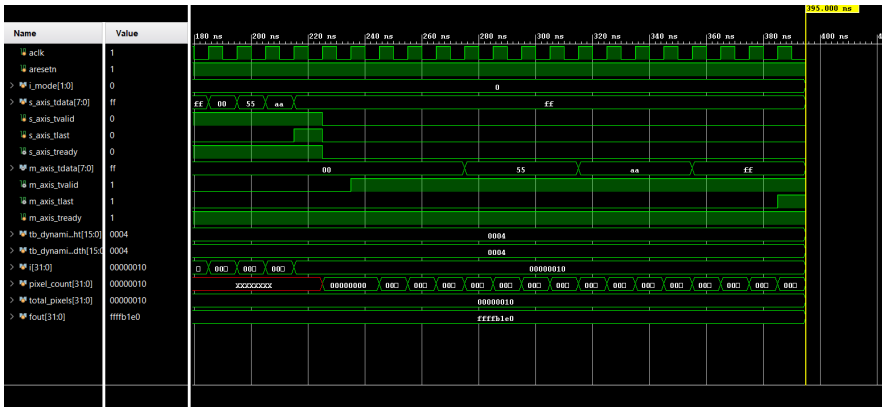
Sau khi hoàn tất quá trình xử lý ảnh tại Slave, các pixel được gửi ngược lại cho Master theo thứ tự mới tương ứng với chế độ xử lý đã chọn. Dữ liệu này cũng được hiển thị trên cửa sổ console để phục vụ kiểm tra.



Hình 12: Waveform mô phỏng quá trình truyền dữ liệu (1)

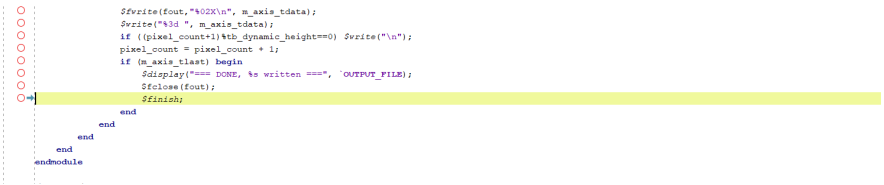


Hình 13: Waveform mô phỏng quá trình truyền dữ liệu (2)



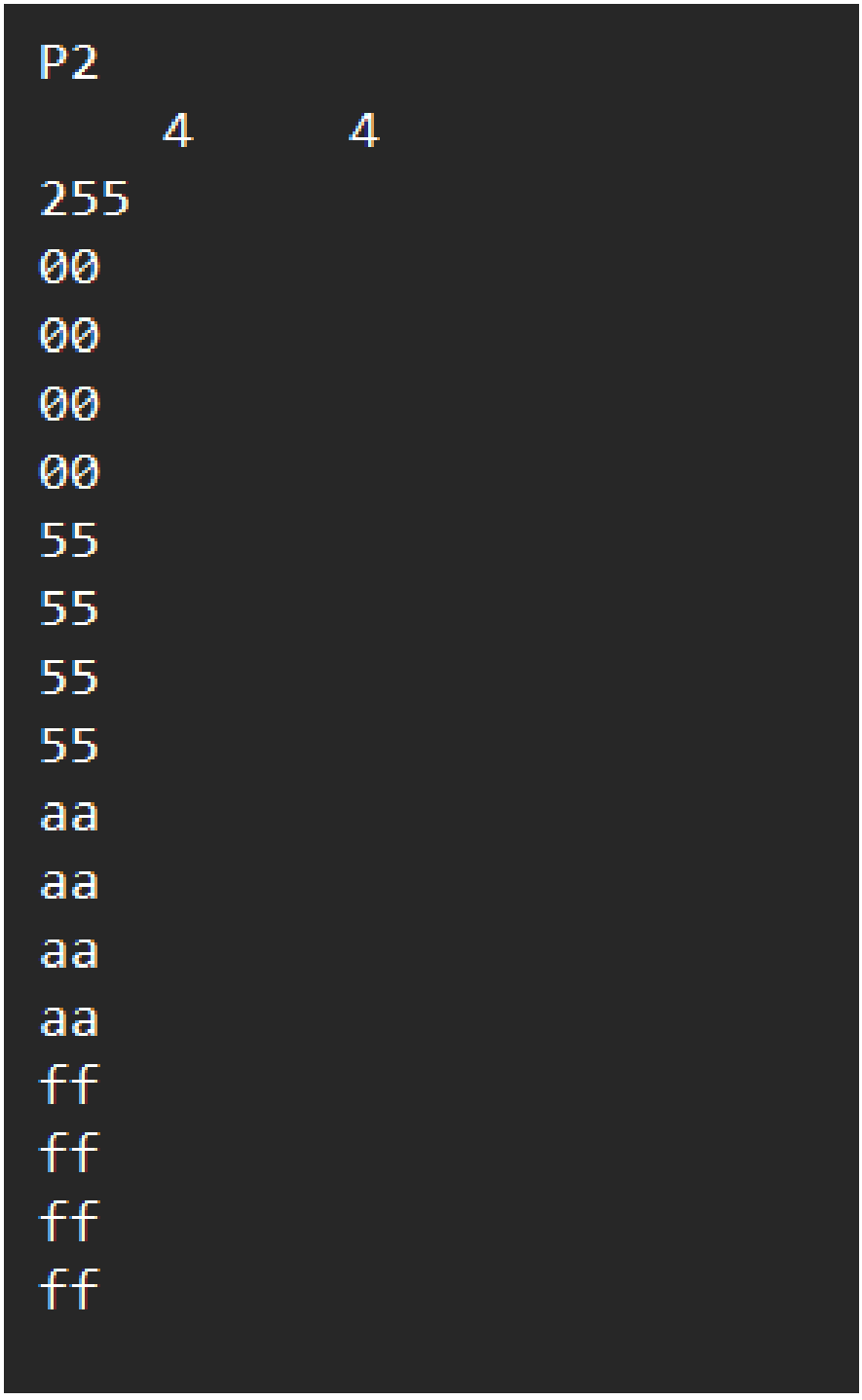
Hình 14: Waveform mô phỏng quá trình truyền dữ liệu (3)

Các waveform ghi lại toàn bộ quá trình mô phỏng cho thấy các tín hiệu `VALID`, `READY` và `TLAST` đều hoạt động đúng theo chuẩn giao thức AXI Stream và tại các thời điểm dự kiến.



Hình 15: Dòng tô đậm vàng trỏ tới `finish`

Quá trình diễn ra hoàn tất và không xảy ra lỗi.



Hình 16: Nội dung file ảnh `.pgm` sau khi hoàn tất xử lý

Kết quả cho thấy các pixel đã được ghi vào file ảnh đầu ra với vị trí mới phù hợp với thuật toán xử lý ảnh đã cài đặt, chứng minh quá trình xử lý ảnh được thực hiện thành công.
Nhận xét

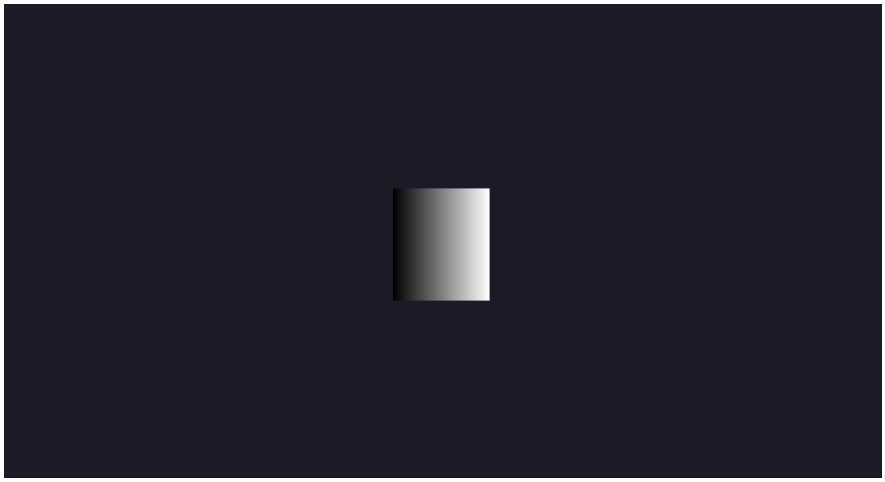
- Hệ thống truyền nhận dữ liệu hoạt động ổn định, không xảy ra mất hoặc trùng pixel trong quá trình mô phỏng.
- Cơ chế bắt tay `VALID/READY` và tín hiệu `TLAST` tuân thủ đúng chuẩn AXI Stream.
- Kết quả ảnh đầu ra phù hợp với mong đợi, xác nhận tính đúng đắn của logic xử lý ảnh và máy trạng thái đã thiết kế.

6.8.2.b Kiểm tra các chế độ xoay và lật

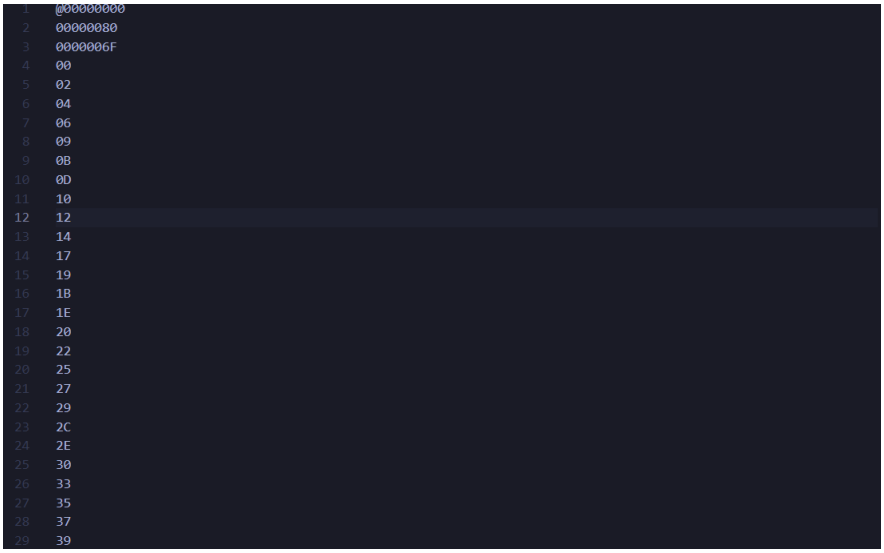
Các chức năng được kiểm tra

- Kiểm tra chức năng xoay ảnh 90 độ theo chiều kim đồng hồ (Rotate 90° CW).
- Kiểm tra chức năng xoay ảnh 90 độ ngược chiều kim đồng hồ (Rotate 90° CCW).
- Kiểm tra chức năng lật ảnh theo phương ngang (Horizontal Mirror).
- Kiểm tra chức năng lật ảnh theo phương dọc (Vertical Mirror).
- Kiểm tra thứ tự và vị trí các pixel ở ảnh đầu ra so với kết quả mong đợi.
- Kiểm tra khả năng hoạt động của hệ thống với các kích thước ảnh khác nhau trong giới hạn cho phép.

Xoay ảnh 90 độ theo chiều kim đồng hồ (Rotate 90° CW)
Trong trường hợp kiểm thử này, hệ thống được cấu hình ở chế độ **xoay ảnh 90° theo chiều kim đồng hồ (Rotate 90° CW)**. Dữ liệu đầu vào là file `.mem` được sinh tự động từ ảnh grayscale ban đầu, trong đó bao gồm phần header mô tả kích thước ảnh và các pixel được mã hóa dưới dạng dữ liệu 8-bit.

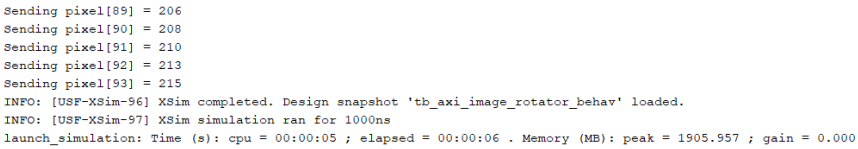


Hình 17: Ảnh đầu vào có kích thước 111×128



Hình 18: Nội dung file `.mem` được chuyển đổi từ ảnh grayscale

Trong quá trình mô phỏng, dữ liệu pixel được truyền từ Master đến Slave thông qua giao tiếp AXI Stream theo đúng thứ tự tuyến tính ban đầu. Số lượng pixel được gửi và nhận tại khối xử lý đúng bằng giá trị `height × width` của ảnh đầu vào, đảm bảo không xảy ra hiện tượng mất hoặc lặp dữ liệu trong pha `RECEIVE`.



Hình 19: Quá trình mô phỏng bị ngắt vì giới hạn hiển thị

```
# run 1000ns
=== Sending 14208 pixels ===
Sending pixel[0] = 0
Sending pixel[1] = 2
Sending pixel[2] = 4
Sending pixel[3] = 6
Sending pixel[4] = 9
Sending pixel[5] = 11
Sending pixel[6] = 13
Sending pixel[7] = 16
Sending pixel[8] = 18
Sending pixel[9] = 20
Sending pixel[10] = 23
Sending pixel[11] = 25
Sending pixel[12] = 27
Sending pixel[13] = 30
```

Hình 20: Dữ liệu pixel được gửi từ Master đến Slave trên cửa sổ Tcl

Do kích thước ảnh tương đối lớn, quá trình mô phỏng RTL trong Vivado bị giới hạn số lượng chu kỳ hiển thị mặc định. Vì vậy, các câu lệnh Tcl được sử dụng để tiếp tục và hoàn tất quá trình mô phỏng.

```
run 10000ns
```

Hình 21: Câu lệnh Tcl được sử dụng để tiếp tục mô phỏng

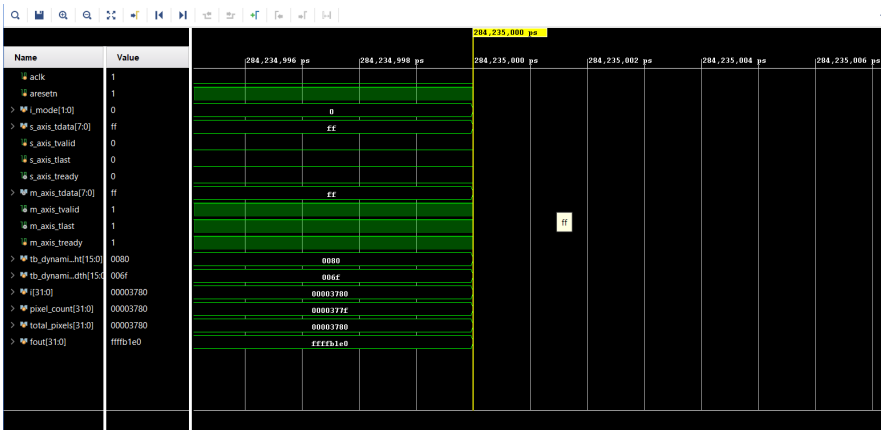
Sau khi toàn bộ dữ liệu ảnh được lưu vào BRAM buffer nội bộ, FSM chuyển sang trạng thái `SEND` để phát dữ liệu đã được xử lý ra kênh M_AXIS. Các pixel đầu ra được phát theo thứ tự mới tương ứng với phép biến đổi xoay ảnh 90° theo chiều kim đồng hồ.

```
210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210 210
213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213
215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215 215
217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217 217
220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220 220
222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222 222
224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224 224
227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227
229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229 229
231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231 231
234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234
236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236 236
238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238 238
241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241 241
243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243 243
245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245 245
248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248 248
250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250 250
252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
=== DONE, samples_out.pgm written ===
```

Hình 22: Quá trình mô phỏng hoàn tất và dữ liệu đầu ra được in ra màn hình

Quan sát waveform cho thấy các tín hiệu điều khiển `TVALID`, `TREADY` và `TLAST` hoạt động đúng thời

điểm, tuân thủ đầy đủ chuẩn AXI Stream. Tín hiệu `TLAST` chỉ được assert tại pixel cuối cùng của khung ảnh đầu ra, xác nhận ranh giới frame được xác định chính xác.

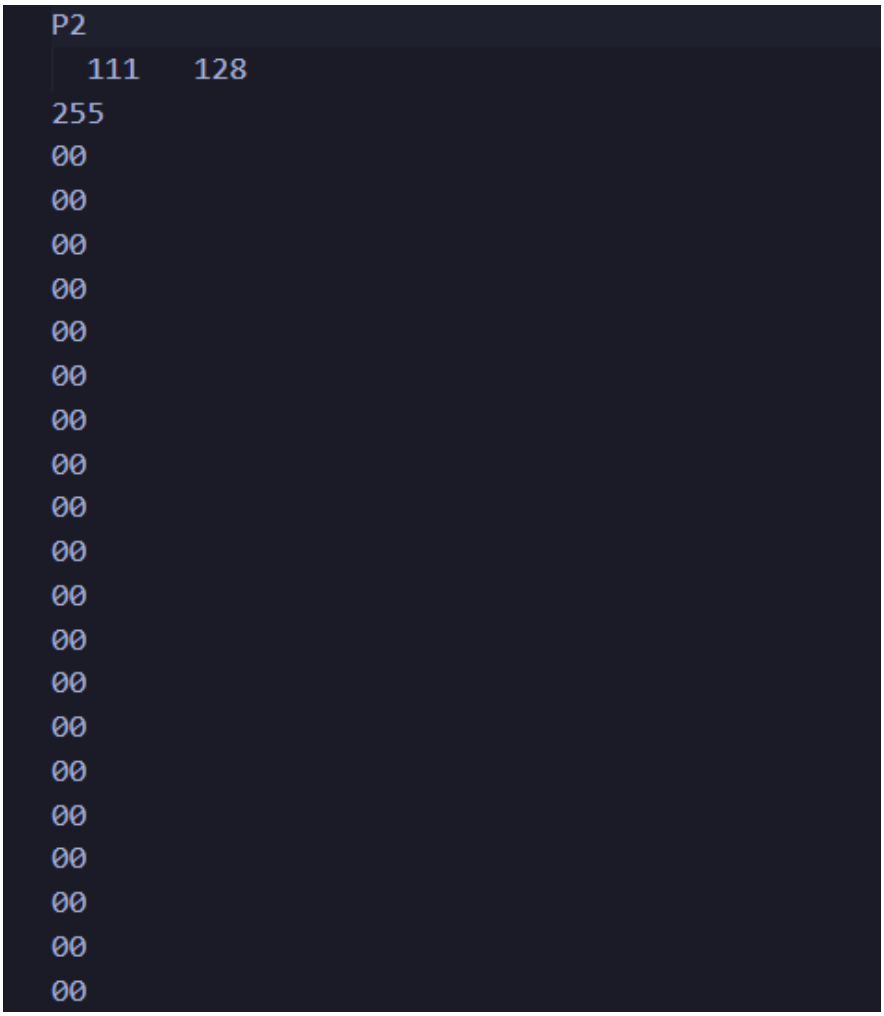


Hình 23: Waveform của quá trình mô phỏng chế độ Rotate 90° CW

Tổng thời gian mô phỏng cho testcase này là:

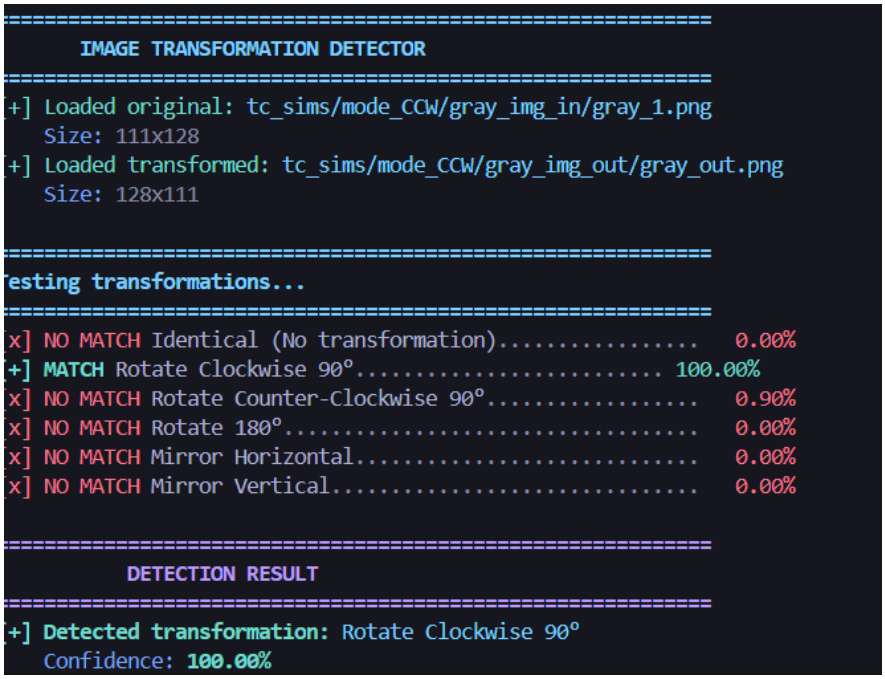
$$284{,}235{,}000 \text{ ps} \approx 2.84 \times 10^{-4} \text{ s}$$

Giá trị này lớn hơn đáng kể so với các testcase sử dụng ảnh kích thước nhỏ, phản ánh đặc trưng của mô phỏng RTL theo chu kỳ xung nhịp.



Hình 24: Nội dung file `.pgm` sau khi xử lý

Hình 25: Ảnh đầu ra sau khi chuyển đổi từ `.pgm` sang `.png`



Hình 26: So sánh ảnh đầu vào và ảnh đầu ra sau khi xoay

Xoay ảnh 90 độ ngược chiều kim đồng hồ (Rotate 90° CCW)
Trong trường hợp kiểm thử này, hệ thống được cấu hình ở chế độ **xoay ảnh 90° theo ngược chiều kim đồng hồ (Rotate 90° CW)**. Dữ liệu đầu vào là file `.mem` được sinh tự động từ ảnh grayscale ban đầu, trong đó bao gồm phần header mô tả kích thước ảnh và các pixel được mã hóa dưới dạng dữ liệu 8-bit.

Hình 27: Ảnh đầu vào có kích thước 63×20

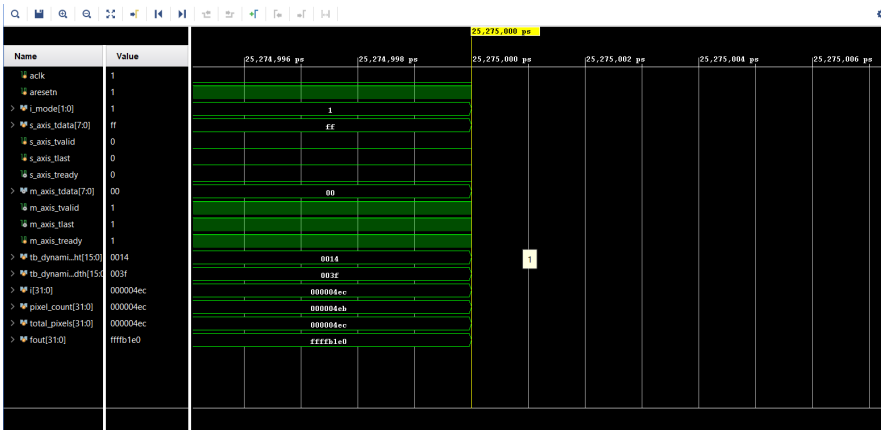
Hình 28: Nội dung file `.mem` được chuyển đổi từ ảnh grayscale

Sau khi toàn bộ dữ liệu ảnh được lưu vào BRAM buffer nội bộ, FSM chuyển sang trạng thái `SEND` để phát dữ liệu đã được xử lý ra kênh M_AXIS. Các pixel đầu ra được phát theo thứ tự mới tương ứng với phép biến đổi xoay ảnh 90° theo ngược chiều kim đồng hồ.

```
65  65  65  65  65  65  65  65  65  65  65  65  65  65  65  65  65  65  65  65
61  61  61  61  61  61  61  61  61  61  61  61  61  61  61  61  61  61  61  61
57  57  57  57  57  57  57  57  57  57  57  57  57  57  57  57  57  57  57  57
53  53  53  53  53  53  53  53  53  53  53  53  53  53  53  53  53  53  53  53
49  49  49  49  49  49  49  49  49  49  49  49  49  49  49  49  49  49  49  49
45  45  45  45  45  45  45  45  45  45  45  45  45  45  45  45  45  45  45  45
41  41  41  41  41  41  41  41  41  41  41  41  41  41  41  41  41  41  41  41
37  37  37  37  37  37  37  37  37  37  37  37  37  37  37  37  37  37  37  37
32  32  32  32  32  32  32  32  32  32  32  32  32  32  32  32  32  32  32  32
28  28  28  28  28  28  28  28  28  28  28  28  28  28  28  28  28  28  28  28
24  24  24  24  24  24  24  24  24  24  24  24  24  24  24  24  24  24  24  24
20  20  20  20  20  20  20  20  20  20  20  20  20  20  20  20  20  20  20  20
16  16  16  16  16  16  16  16  16  16  16  16  16  16  16  16  16  16  16  16
12  12  12  12  12  12  12  12  12  12  12  12  12  12  12  12  12  12  12  12
 8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8   8
 4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4   4
 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
=== DONE, samples_out.pgm written ===
```

Hình 29: Quá trình mô phỏng hoàn tất và dữ liệu đầu ra được in ra màn hình

Quan sát waveform cho thấy các tín hiệu điều khiển `TVALID`, `TREADY` và `TLAST` hoạt động đúng thời điểm, tuân thủ đầy đủ chuẩn AXI Stream. Tín hiệu `TLAST` chỉ được assert tại pixel cuối cùng của khung ảnh đầu ra, xác nhận ranh giới frame được xác định chính xác.



Hình 30: Waveform của quá trình mô phỏng chế độ Rotate 90° CCW

Tổng thời gian mô phỏng cho testcase này là:

$$25{,}275{,}000 \text{ ps} \approx 25.75 \times 10^{-4} \text{ s}$$

Hình 31: Ảnh đầu ra sau khi chuyển đổi từ `.pgm` sang `.png`

```
P2
   63    20
255
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
ff
fa
fa
```

Hình 32: Nội dung file `.pgm` sau khi xử lý

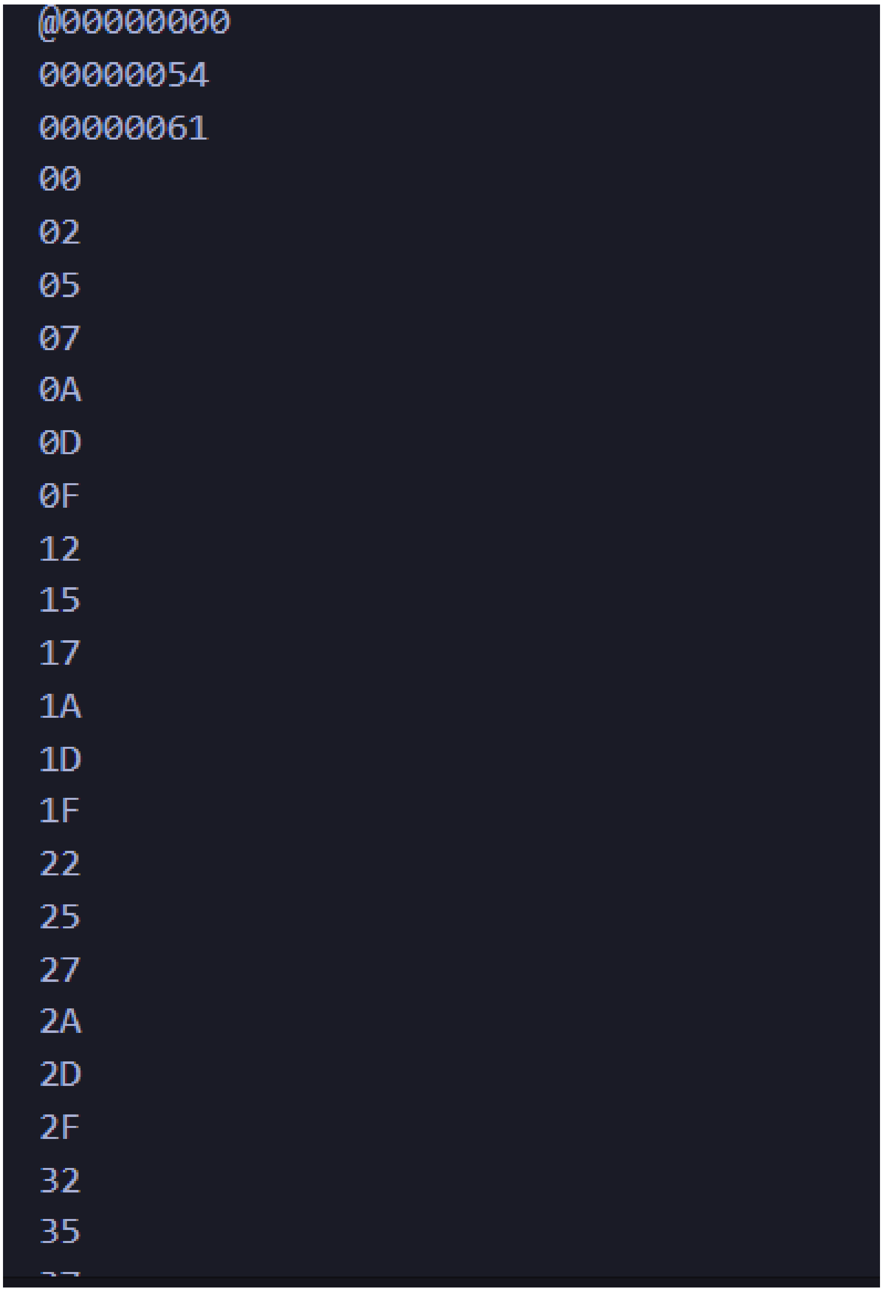
```
============================================================
       IMAGE TRANSFORMATION DETECTOR
============================================================
[+] Loaded original: mode_CCW/gray_img_in/gray_1.png
    Size: 63x20
[+] Loaded transformed: mode_CCW/gray_img_out/gray_out.png
    Size: 20x63

============================================================
Testing transformations...
============================================================
[x] NO MATCH Identical (No transformation).................   0.00%
[x] NO MATCH Rotate Clockwise 90°..........................   1.59%
[+] MATCH Rotate Counter-Clockwise 90°..................... 100.00%
[x] NO MATCH Rotate 180°...................................   0.00%
[x] NO MATCH Mirror Horizontal.............................   0.00%
[x] NO MATCH Mirror Vertical...............................   0.00%

============================================================
          DETECTION RESULT
============================================================
[+] Detected transformation: Rotate Counter-Clockwise 90°
    Confidence: 100.00%
    Mode value: 1
============================================================
```

Hình 33: So sánh ảnh đầu vào và ảnh đầu ra sau khi xoay

Lật ảnh theo phương ngang (Horizontal Mirror)

Trong trường hợp kiểm thử này, hệ thống được cấu hình ở chế độ **lật ảnh theo phương ngang (Horizonal Mirror)**. Dữ liệu đầu vào là file `.mem` được sinh tự động từ ảnh grayscale ban đầu, trong đó bao gồm phần header mô tả kích thước ảnh và các pixel được mã hóa dưới dạng dữ liệu 8-bit.

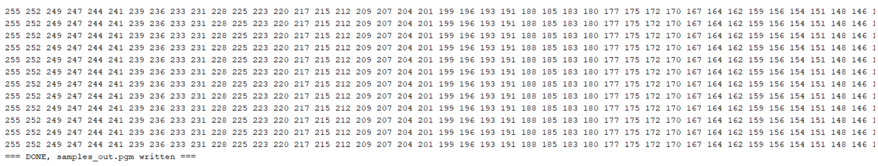
Hình 34: Ảnh đầu vào có kích thước 84x97



```
@00000000
00000054
00000061
00
02
05
07
0A
0D
0F
12
15
17
1A
1D
1F
22
25
27
2A
2D
2F
32
35
```

Hình 35: Nội dung file `.mem` được chuyển đổi từ ảnh grayscale

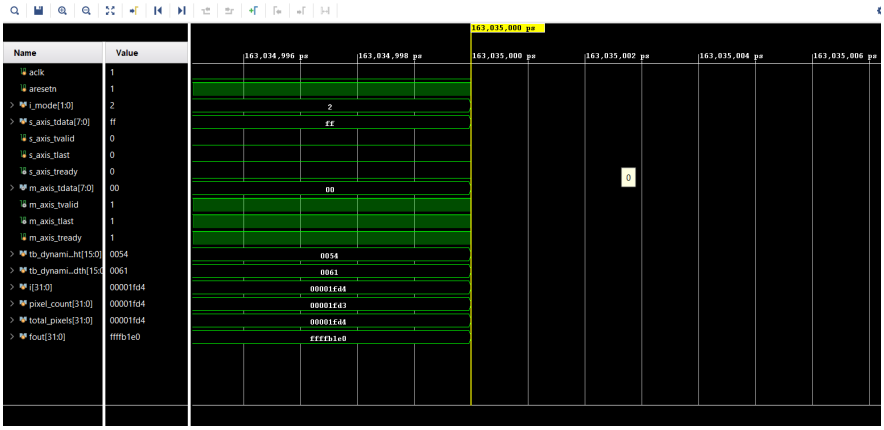
Sau khi toàn bộ dữ liệu ảnh được lưu vào BRAM buffer nội bộ, FSM chuyển sang trạng thái `SEND` để phát dữ liệu đã được xử lý ra kênh M_AXIS. Các pixel đầu ra được phát theo thứ tự mới tương ứng với phép biến đổi xoay ảnh 90° theo ngược chiều kim đồng hồ.



```
255 252 249 247 244 241 239 236 233 231 228 225 223 220 217 215 212 209 207 204 201 199 196 193 191 188 185 183 180 177 175 172 170 167 164 162 159 156 154 151 148 146
255 252 249 247 244 241 239 236 233 231 228 225 223 220 217 215 212 209 207 204 201 199 196 193 191 188 185 183 180 177 175 172 170 167 164 162 159 156 154 151 148 146
255 252 249 247 244 241 239 236 233 231 228 225 223 220 217 215 212 209 207 204 201 199 196 193 191 188 185 183 180 177 175 172 170 167 164 162 159 156 154 151 148 146
255 252 249 247 244 241 239 236 233 231 228 225 223 220 217 215 212 209 207 204 201 199 196 193 191 188 185 183 180 177 175 172 170 167 164 162 159 156 154 151 148 146
255 252 249 247 244 241 239 236 233 231 228 225 223 220 217 215 212 209 207 204 201 199 196 193 191 188 185 183 180 177 175 172 170 167 164 162 159 156 154 151 148 146
255 252 249 247 244 241 239 236 233 231 228 225 223 220 217 215 212 209 207 204 201 199 196 193 191 188 185 183 180 177 175 172 170 167 164 162 159 156 154 151 148 146
255 252 249 247 244 241 239 236 233 231 228 225 223 220 217 215 212 209 207 204 201 199 196 193 191 188 185 183 180 177 175 172 170 167 164 162 159 156 154 151 148 146
255 252 249 247 244 241 239 236 233 231 228 225 223 220 217 215 212 209 207 204 201 199 196 193 191 188 185 183 180 177 175 172 170 167 164 162 159 156 154 151 148 146
255 252 249 247 244 241 239 236 233 231 228 225 223 220 217 215 212 209 207 204 201 199 196 193 191 188 185 183 180 177 175 172 170 167 164 162 159 156 154 151 148 146
255 252 249 247 244 241 239 236 233 231 228 225 223 220 217 215 212 209 207 204 201 199 196 193 191 188 185 183 180 177 175 172 170 167 164 162 159 156 154 151 148 146
255 252 249 247 244 241 239 236 233 231 228 225 223 220 217 215 212 209 207 204 201 199 196 193 191 188 185 183 180 177 175 172 170 167 164 162 159 156 154 151 148 146
255 252 249 247 244 241 239 236 233 231 228 225 223 220 217 215 212 209 207 204 201 199 196 193 191 188 185 183 180 177 175 172 170 167 164 162 159 156 154 151 148 146
=== DONE, samples_out.pgm written ===
```

Hình 36: Quá trình mô phỏng hoàn tất và dữ liệu đầu ra được in ra màn hình

Quan sát waveform cho thấy các tín hiệu điều khiển `TVALID`, `TREADY` và `TLAST` hoạt động đúng thời điểm, tuân thủ đầy đủ chuẩn AXI Stream. Tín hiệu `TLAST` chỉ được assert tại pixel cuối cùng của khung ảnh đầu ra, xác nhận ranh giới frame được xác định chính xác.



Hình 37: Waveform của quá trình mô phỏng chế độ lật theo phương ngang

Tổng thời gian mô phỏng cho testcase này là:

$$163{,}035{,}000 \text{ ps} \approx 1.63 \times 10^{-4} \text{ s}$$

Hình 38: Ảnh đầu ra sau khi chuyển đổi từ `.pgm` sang `.png`

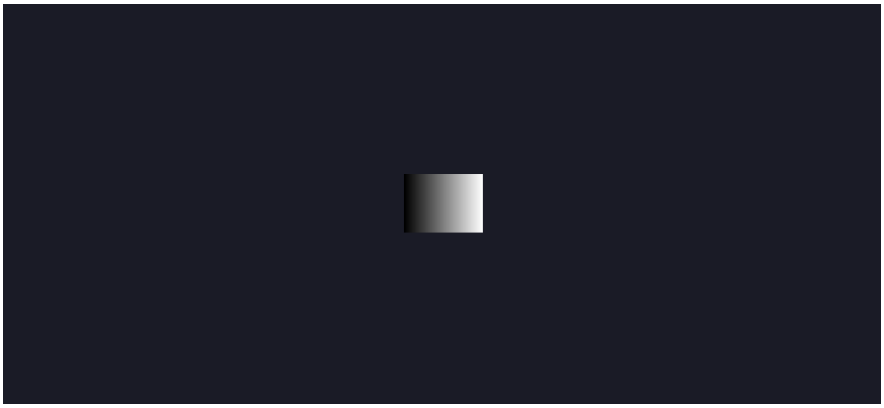
```
P2
 84 97
255
ff
fc
f9
f7
f4
f1
ef
ec
e9
e7
e4
e1
df
dc
d9
d7
d4
d1
cf
cc
```

Hình 39: Nội dung file `.pgm` sau khi xử lý

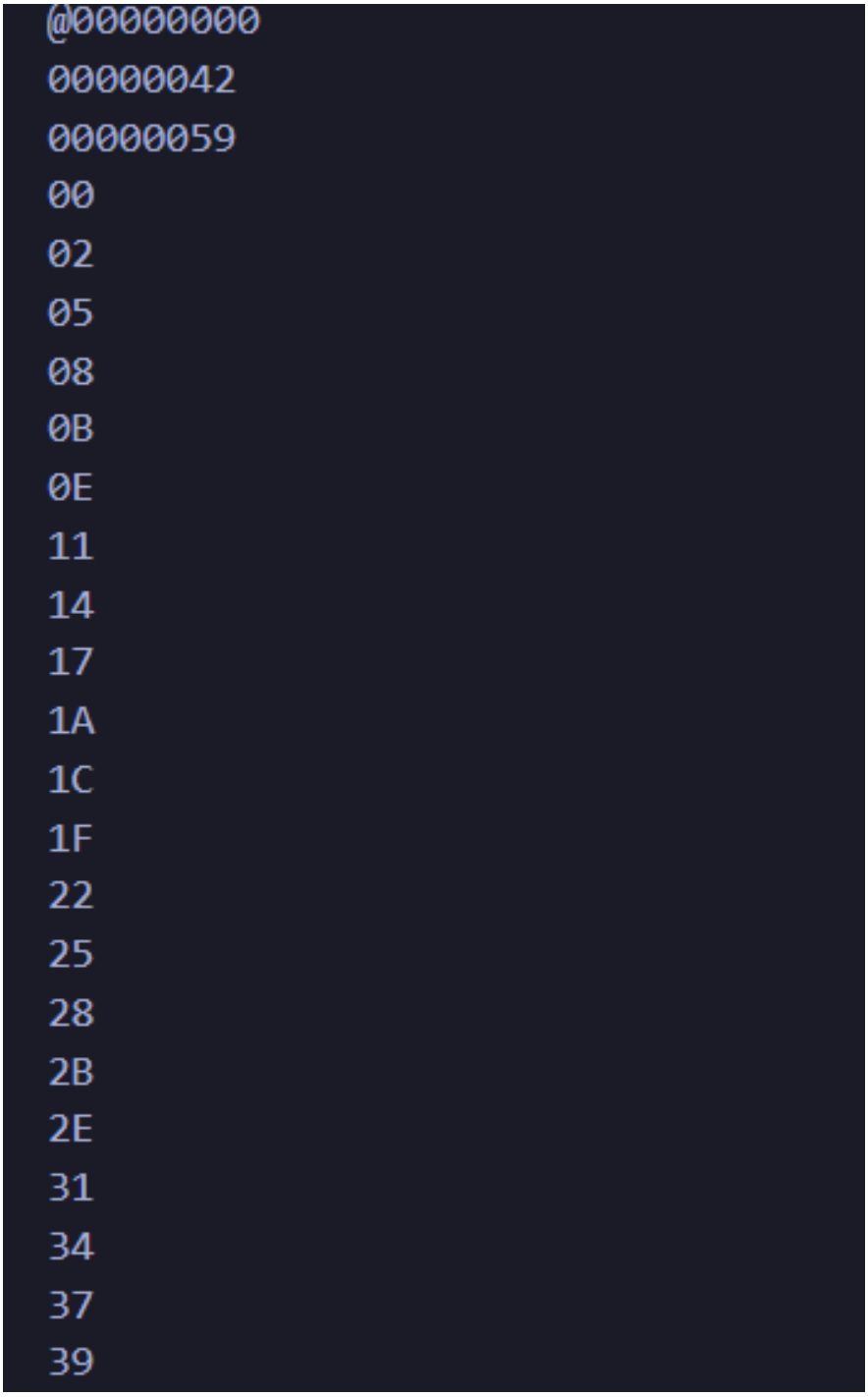
```
============================================================
      IMAGE TRANSFORMATION DETECTOR
============================================================
[+] Loaded original: mode_H/gray_img_in/gray_4.png
    Size: 97x84
[+] Loaded transformed: mode_H/gray_img_out/gray_out_4.png
    Size: 97x84

============================================================
Testing transformations...
============================================================
[x] NO MATCH Identical (No transformation).................   1.03%
[x] NO MATCH Rotate Clockwise 90°..........................   0.00%
[x] NO MATCH Rotate Counter-Clockwise 90°..................   0.00%
[+] MATCH Rotate 180°..................................... 100.00%
[+] MATCH Mirror Horizontal............................... 100.00%
[x] NO MATCH Mirror Vertical...............................   1.03%

============================================================
          DETECTION RESULT
============================================================
[+] Detected transformation: Rotate 180°
    Confidence: 100.00%
============================================================
```

Hình 40: So sánh ảnh đầu vào và ảnh đầu ra sau khi lật

Lật ảnh theo chiều dọc (Mirror Vertical)
Trong trường hợp kiểm thử này, hệ thống được cấu hình ở chế độ **lật ảnh theo chiều dọc (vertical Mirror)**. Dữ liệu đầu vào là file `.mem` được sinh tự động từ ảnh grayscale ban đầu, trong đó bao gồm phần header mô tả kích thước ảnh và các pixel được mã hóa dưới dạng dữ liệu 8-bit.

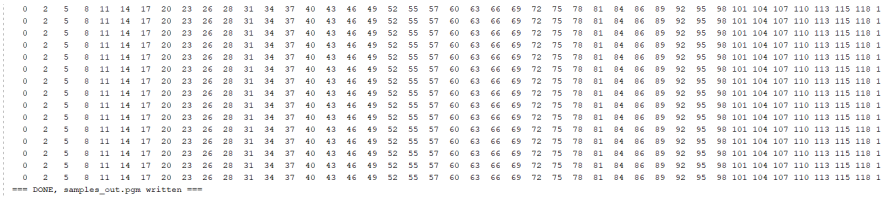


Hình 41: Ảnh đầu vào có kích thước 66x89



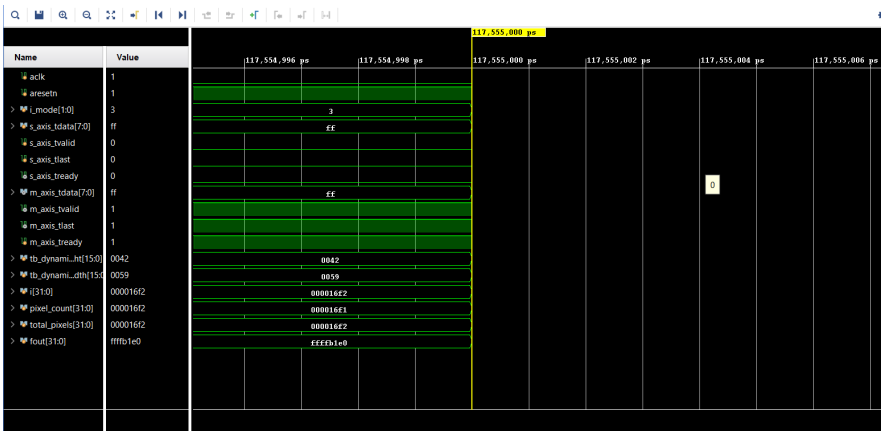
Hình 42: Nội dung file `.mem` được chuyển đổi từ ảnh grayscale

Sau khi toàn bộ dữ liệu ảnh được lưu vào BRAM buffer nội bộ, FSM chuyển sang trạng thái `SEND` để phát dữ liệu đã được xử lý ra kênh M_AXIS. Các pixel đầu ra được phát theo thứ tự mới tương ứng với phép biến đổi lật theo chiều dọc.



Hình 43: Quá trình mô phỏng hoàn tất và dữ liệu đầu ra được in ra màn hình

Quan sát waveform cho thấy các tín hiệu điều khiển `TVALID`, `TREADY` và `TLAST` hoạt động đúng thời điểm, tuân thủ đầy đủ chuẩn AXI Stream. Tín hiệu `TLAST` chỉ được assert tại pixel cuối cùng của khung ảnh đầu ra, xác nhận ranh giới frame được xác định chính xác.

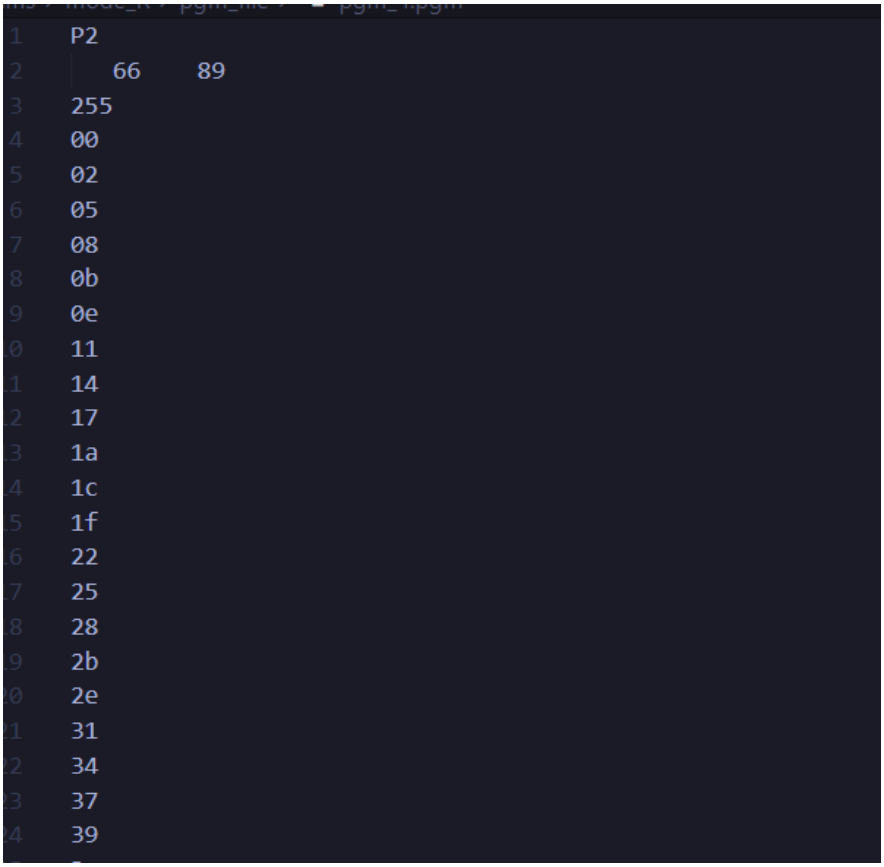


Hình 44: Waveform của quá trình mô phỏng chế độ lật theo chiều dọc

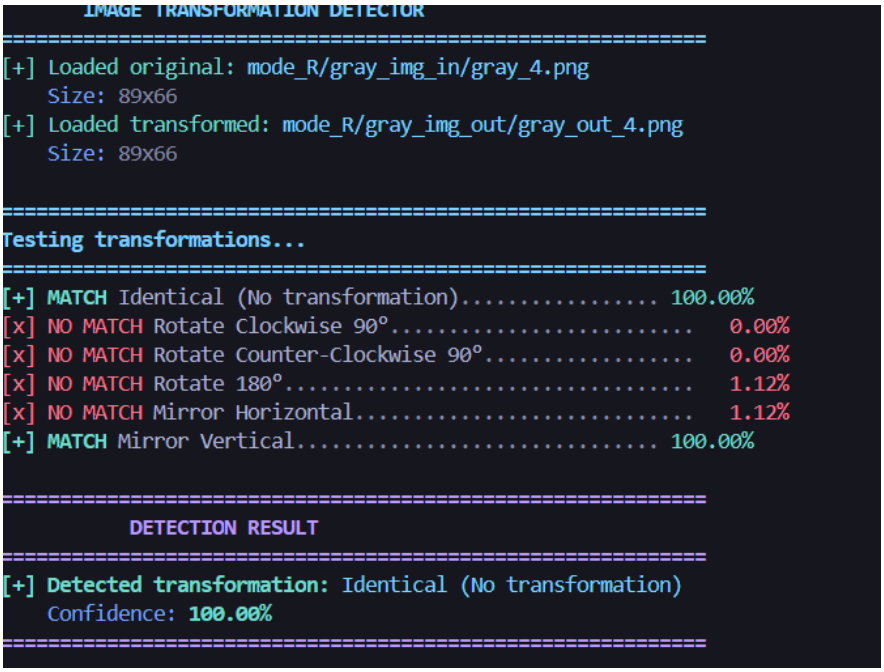
Tổng thời gian mô phỏng cho testcase này là:

$$117{,}555{,}000 \text{ ps} \approx 1.17 \times 10^{-4} \text{ s}$$

Hình 45: Ảnh đầu ra sau khi chuyển đổi từ `.pgm` sang `.png`



Hình 46: Nội dung file `.pgm` sau khi xử lý



Hình 47: So sánh ảnh đầu vào và ảnh đầu ra sau khi lật

Nhận xét

- Khối xử lý ảnh tiếp nhận đầy đủ và chính xác dữ liệu pixel từ file `.mem`, bao gồm cả thông tin kích thước ảnh và nội dung pixel.

- Giao tiếp AXI Stream giữa Master và Slave hoạt động ổn định, không xảy ra mất dữ liệu trong suốt quá trình mô phỏng.
- Máy trạng thái hữu hạn (FSM) điều khiển đúng luồng dữ liệu giữa các trạng thái `IDLE`, `RECEIVE` và `SEND`.
- Thuật toán xoay ảnh 90° theo chiều kim đồng hồ được hiện thực chính xác, thể hiện rõ qua kết quả ảnh đầu ra.
- Các tín hiệu điều khiển `TVALID`, `TREADY` và `TLAST` tuân thủ đúng chuẩn AXI Stream.
- Thời gian mô phỏng tăng theo số lượng pixel của ảnh đầu vào, cho thấy sự khác biệt rõ rệt giữa mô phỏng RTL và thực thi trên phần cứng thực tế.

6.8.2.c Kiểm tra ảnh với kích thước lớn

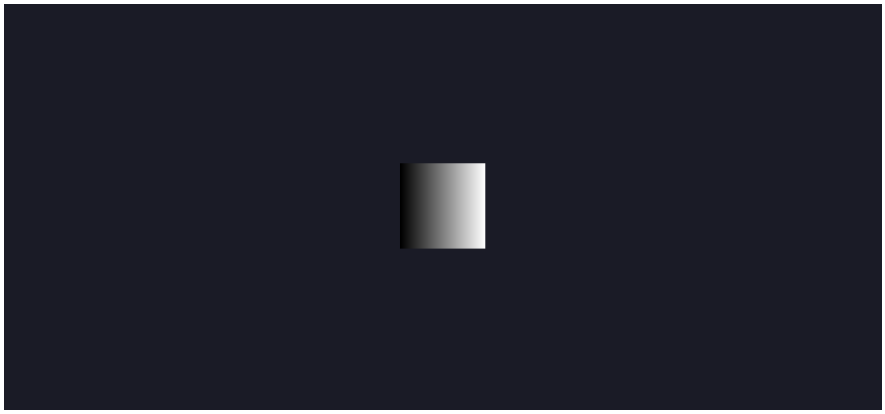
Mục đích của testcase này là đánh giá khả năng hoạt động của hệ thống khi xử lý ảnh có kích thước lớn, đồng thời đo và phân tích **thời gian phản hồi (response time)** của toàn bộ luồng xử lý.
Các chức năng cần kiểm tra

- Khả năng tiếp nhận và xử lý ảnh có kích thước lớn mà không xảy ra tràn bộ nhớ BRAM.
- Tính toàn vẹn của dữ liệu pixel trong suốt quá trình truyền và xử lý.
- Hoạt động ổn định của FSM khi số lượng pixel lớn.
- Đánh giá thời gian phản hồi của hệ thống từ lúc bắt đầu nhận dữ liệu đến khi hoàn tất truyền dữ liệu đầu ra.

Phương pháp đánh giá thời gian phản hồi
Thời gian phản hồi của hệ thống được xác định là khoảng thời gian tính từ:

- Thời điểm pixel đầu tiên được gửi từ AXI Master (bắt tay `S_AXIS_TVALID`–`S_AXIS_TREADY`),
- Cho đến thời điểm pixel cuối cùng được trả về tại AXI Master (tín hiệu `M_AXIS_TLAST` được assert).

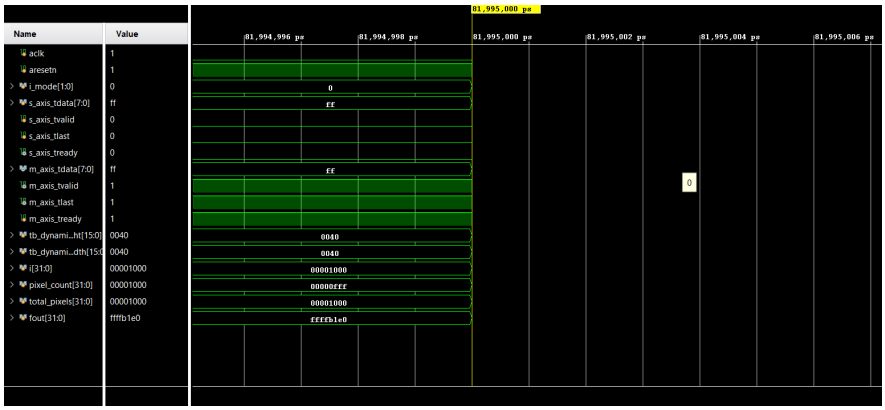
Giá trị thời gian được đo trực tiếp trên waveform mô phỏng RTL và được quy đổi từ đơn vị `ps` sang `s` để thuận tiện cho việc so sánh và phân tích.
Kích thước ảnh 64x64



Hình 48: Ảnh ban đầu có kích thước 64x64

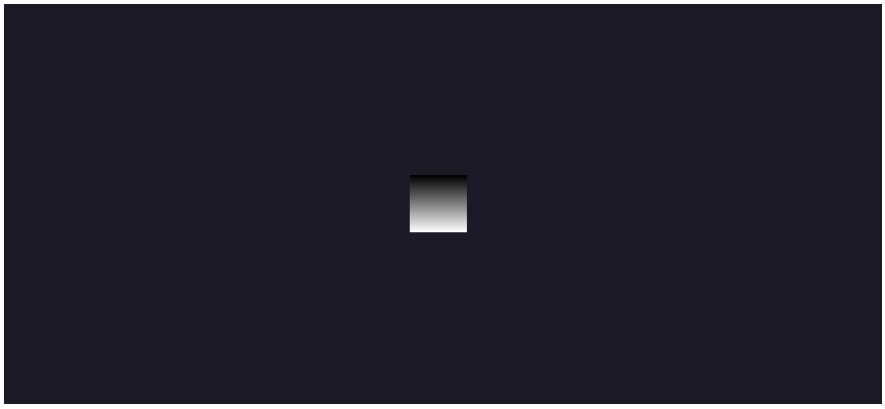
```
@00000000
00000040
00000040
00
04
08
0C
10
14
18
1C
20
24
28
2C
30
34
38
3C
40
44
48
4C
50
```

Hình 49: Nội dung file `.mem` được chuyển đổi từ ảnh grayscale



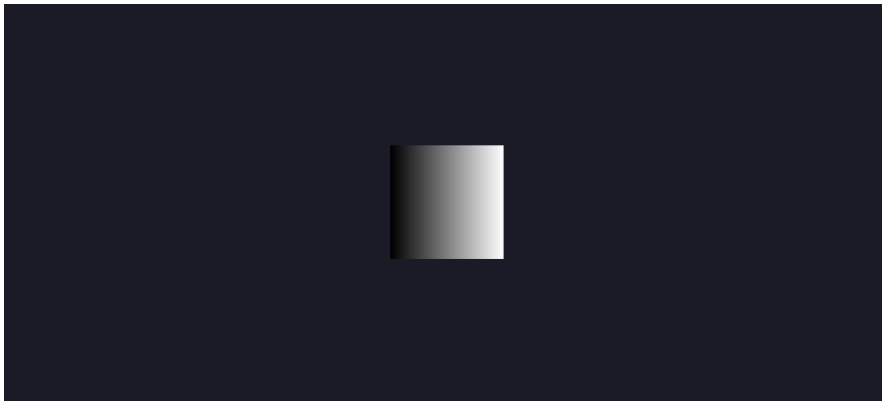
Hình 50: Waveform của quá trình mô phỏng trên

Hình 51: Nội dung file `.pgm` sau khi xử lý

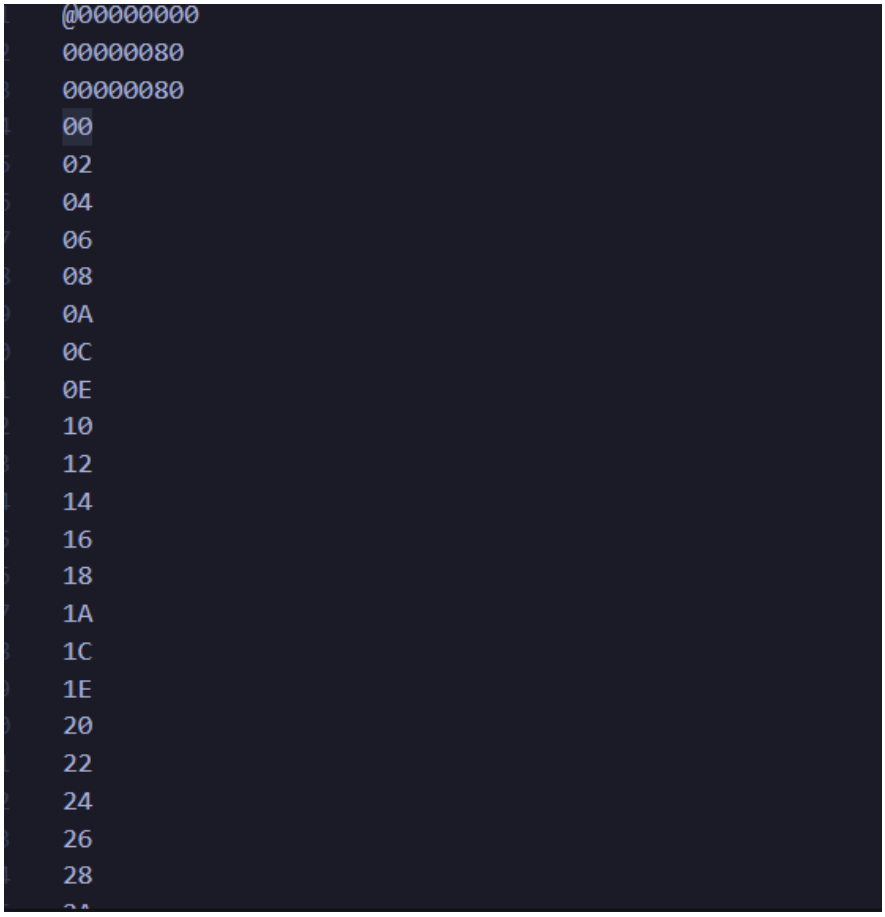


Hình 52: Ảnh đầu ra sau khi chuyển đổi từ `.pgm` sang `.png`

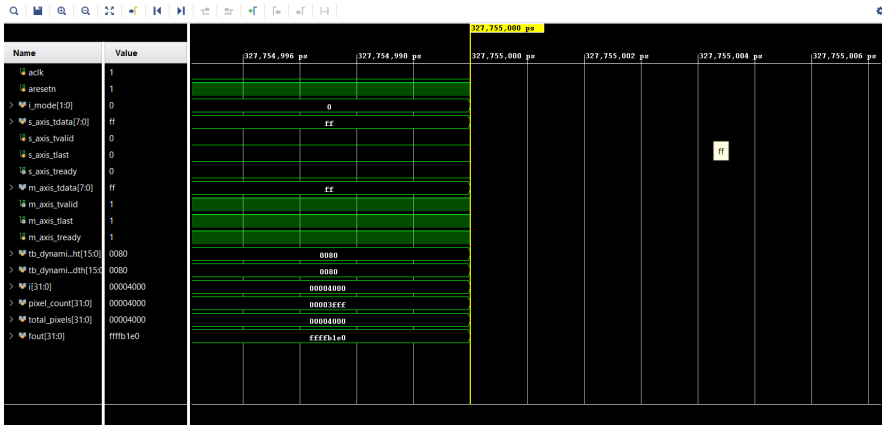
Kích thước ảnh 128x128



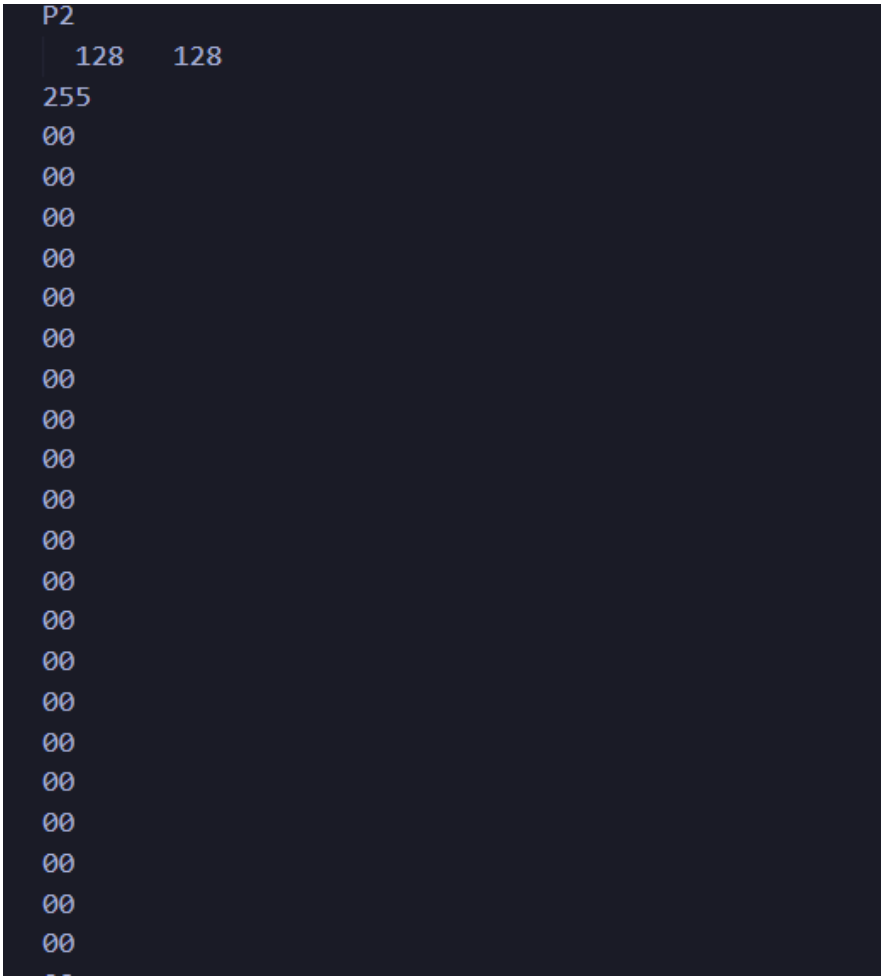
Hình 53: Ảnh ban đầu có kích thước 128x128



Hình 54: Nội dung file `.mem` được chuyển đổi từ ảnh grayscale



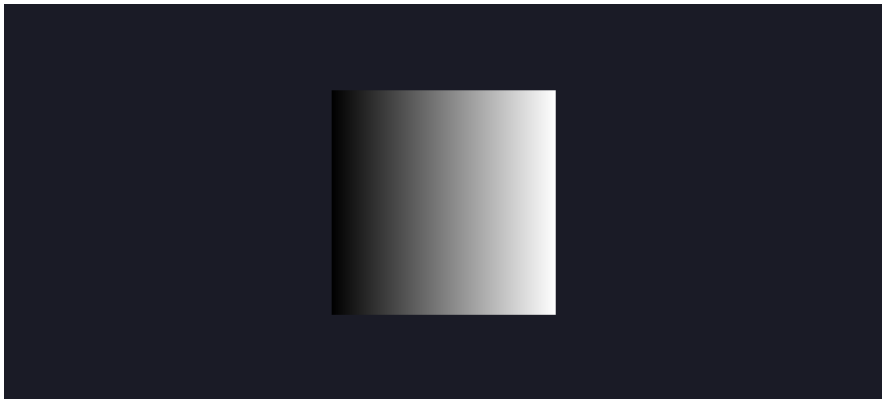
Hình 55: Waveform của quá trình mô phỏng trên



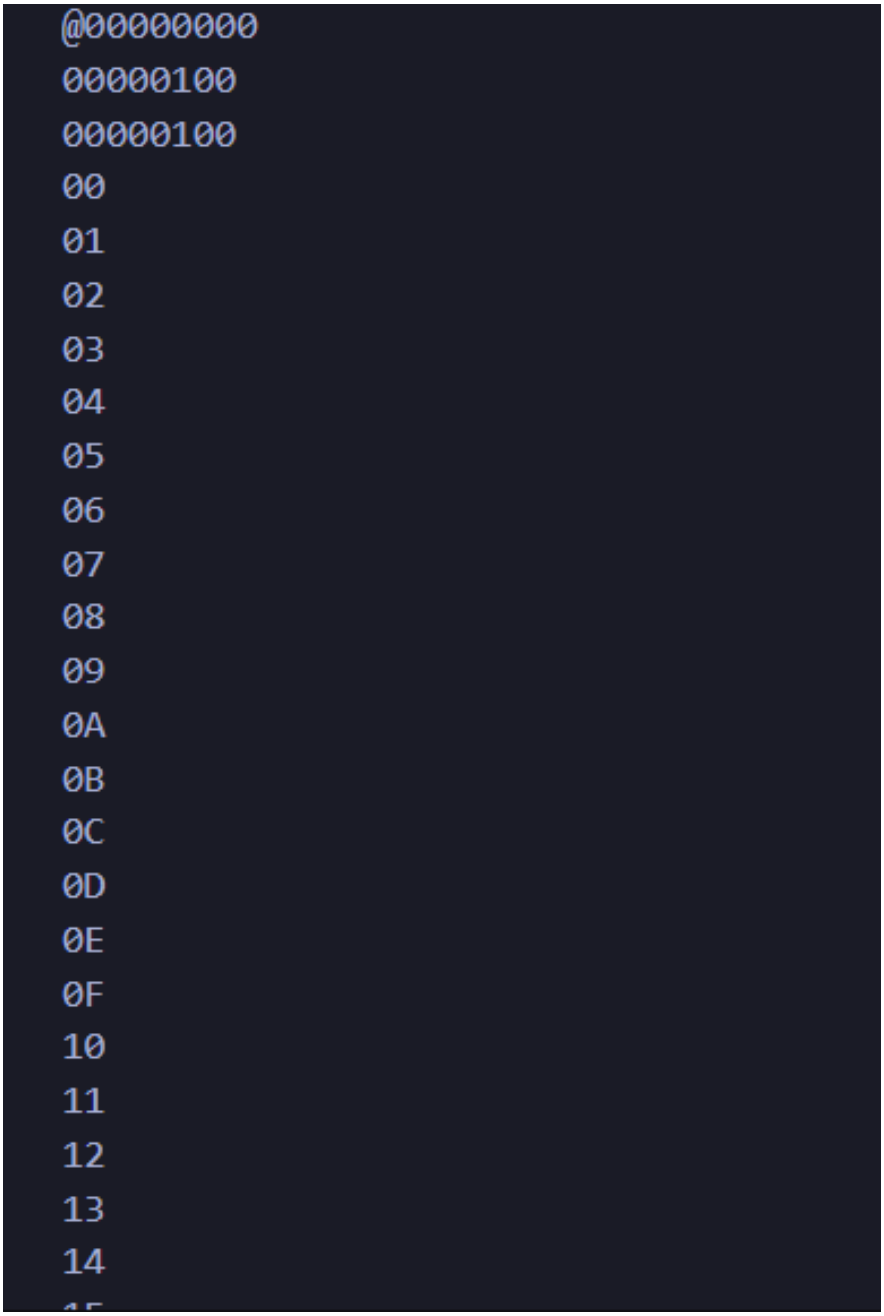
Hình 56: Nội dung file `.pgm` sau khi xử lý

Hình 57: Ảnh đầu ra sau khi chuyển đổi từ `.pgm` sang `.png`

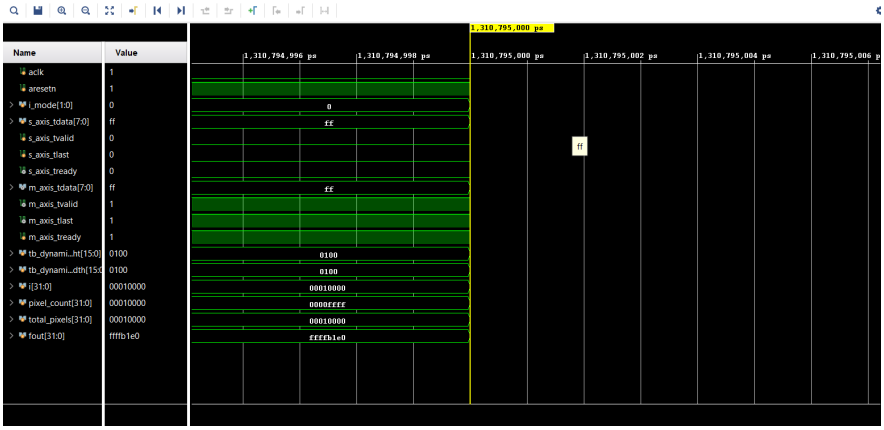
Kích thước ảnh 256x256



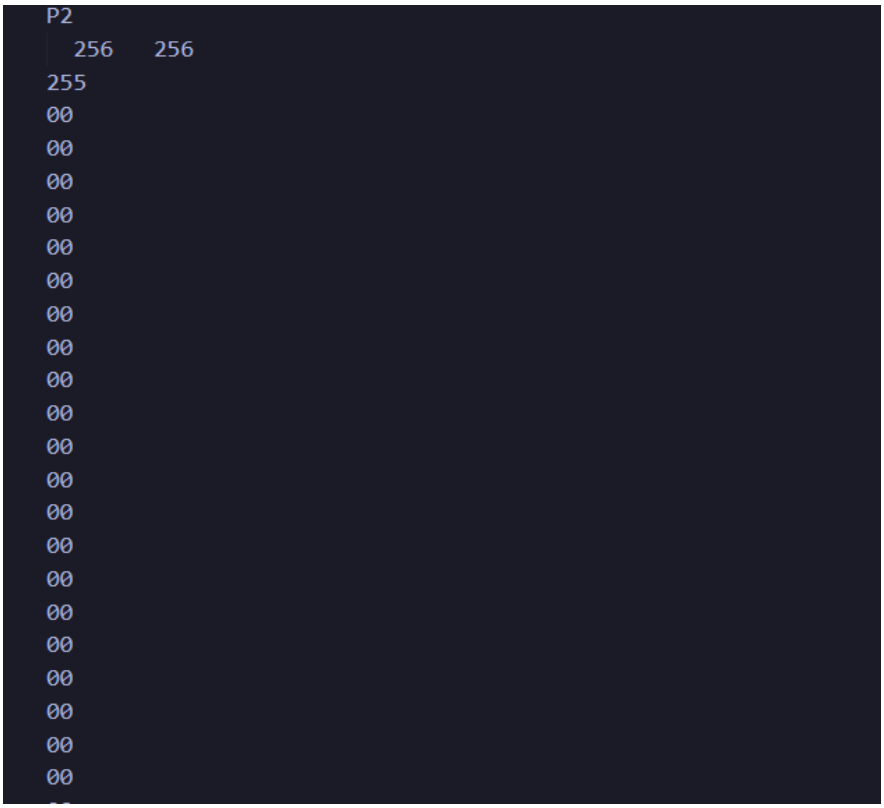
Hình 58: Ảnh ban đầu có kích thước 256x256



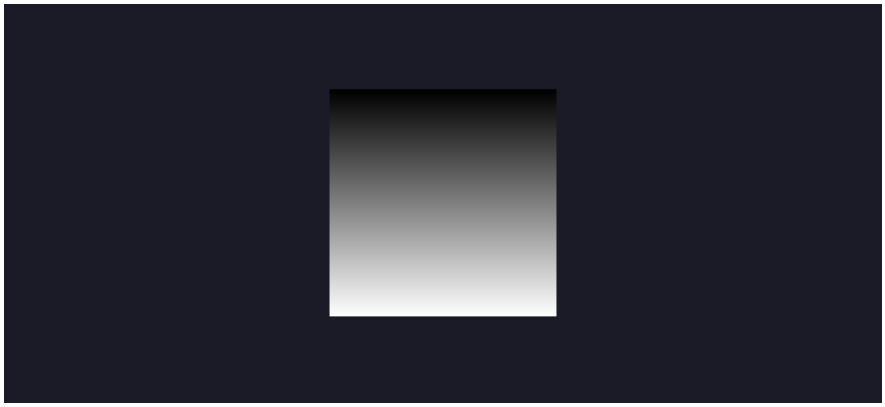
Hình 59: Nội dung file `.mem` được chuyển đổi từ ảnh grayscale



Hình 60: Waveform của quá trình mô phỏng trên

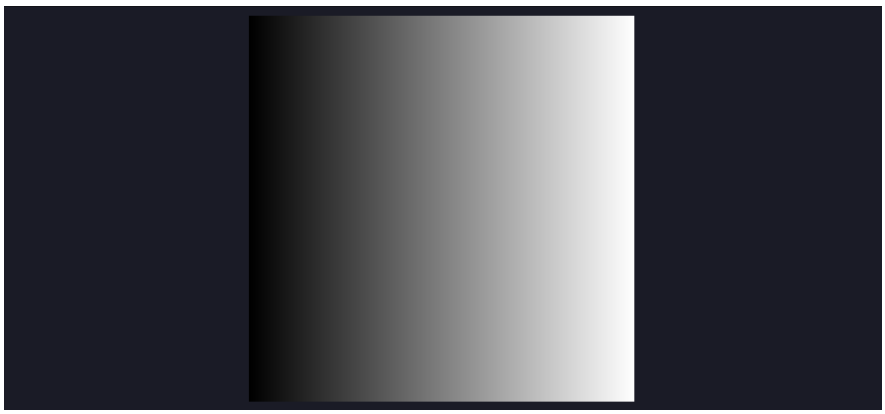


Hình 61: Nội dung file `.pgm` sau khi xử lý

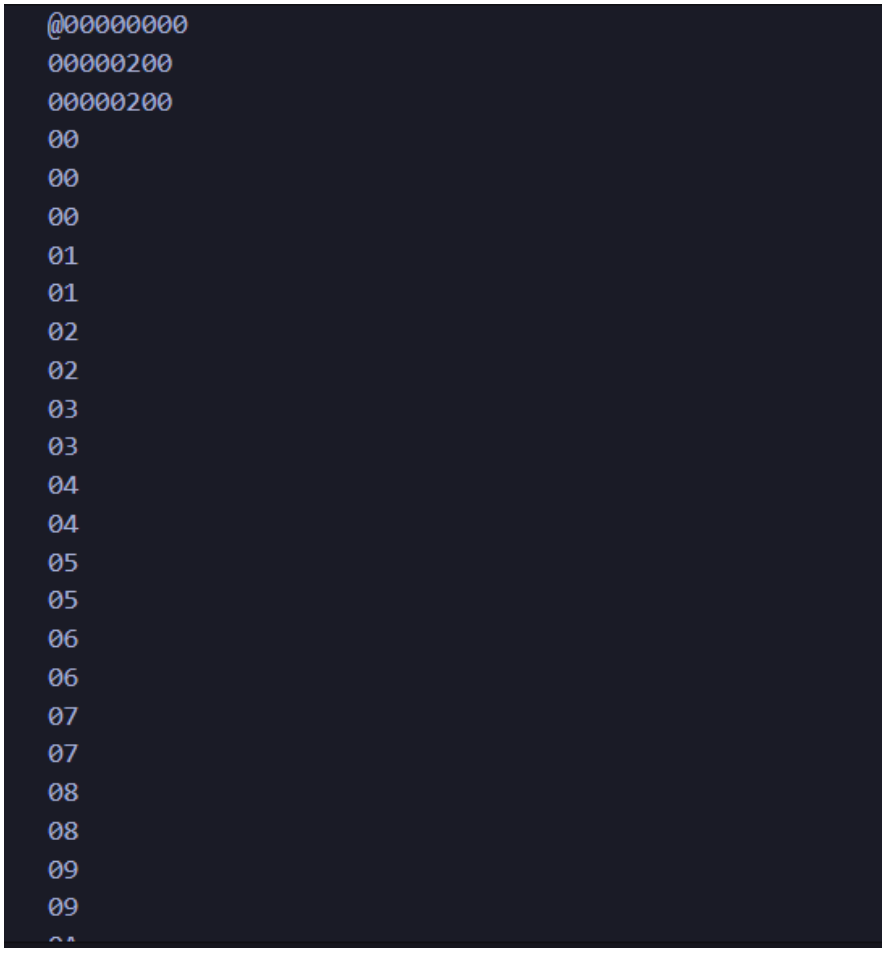


Hình 62: Ảnh đầu ra sau khi chuyển đổi từ `.pgm` sang `.png`

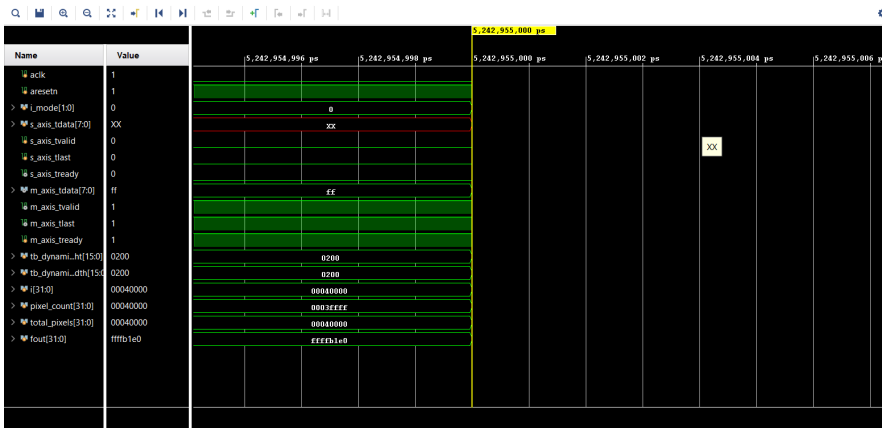
Kích thước ảnh 512x512



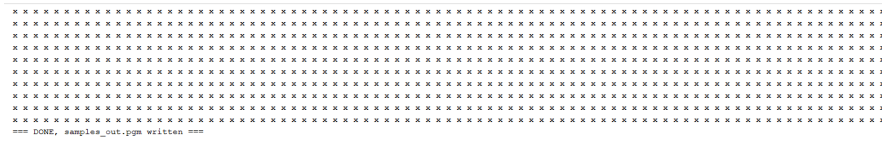
Hình 63: Ảnh ban đầu có kích thước 512x512



Hình 64: Nội dung file `.mem` được chuyển đổi từ ảnh grayscale



Hình 65: Waveform của quá trình mô phỏng trên



Hình 66: Dữ liệu pixel bị mất mát khi kích thước quá lớn

Testcase	Kích thước ảnh	Số pixel	Thời gian mô phỏng (ps)	Thời gian (s)	Kết quả mô phỏng
TC–64x64	64×64	4,096	8.2×10^6	8.2×10^{-6}	Thành công
TC–128x128	128×128	16,384	3.3×10^7	3.3×10^{-5}	Thành công
TC–256x256	256×256	65,536	1.3×10^8	1.3×10^{-4}	Thành công
TC–512x512	512×512	262,144	5.2×10^8	5.2×10^{-4}	Không thành công

Bảng 10: *Đánh giá thời gian phản hồi của hệ thống với các kích thước ảnh khác nhau*

Nhận xét

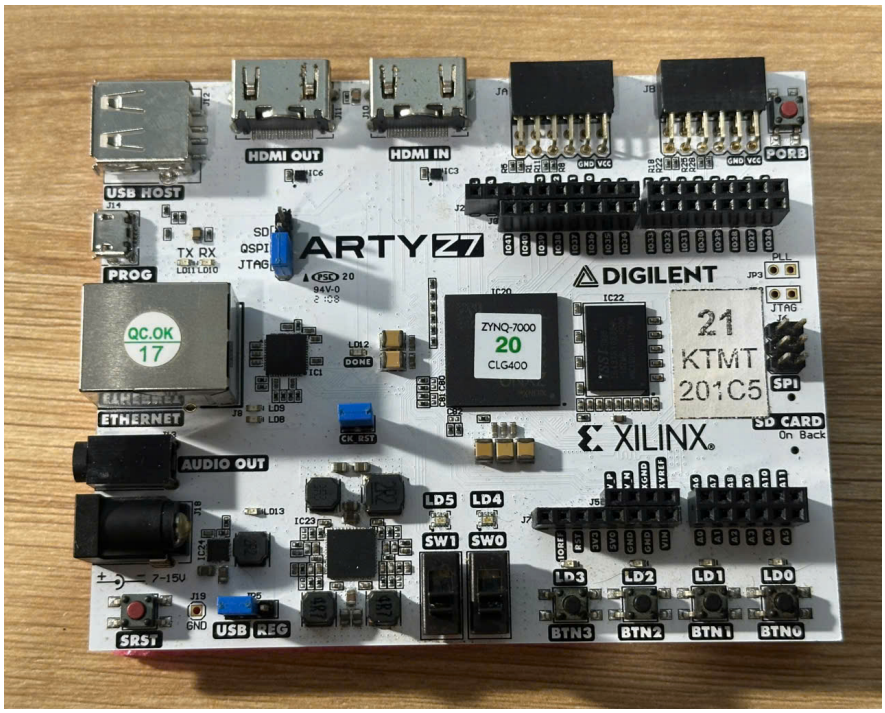
- Khi kích thước ảnh tăng, thời gian phản hồi của hệ thống tăng gần tuyến tính theo số lượng pixel.
- Mất mát dữ liệu xảy ra khi kích thước ảnh quá lớn, cụ thể là lớn hơn 512×512.
- Hệ thống vẫn đảm bảo đúng thứ tự dữ liệu và không phát sinh lỗi giao thức AXI Stream.
- Kết quả cho thấy thiết kế phù hợp cho các ảnh kích thước lớn trong giới hạn tài nguyên bộ nhớ được cấu hình.

6.8.3 Kiểm thử trên phần cứng thực Arty– Z7–20

6.8.3.a Cấu hình phần cứng

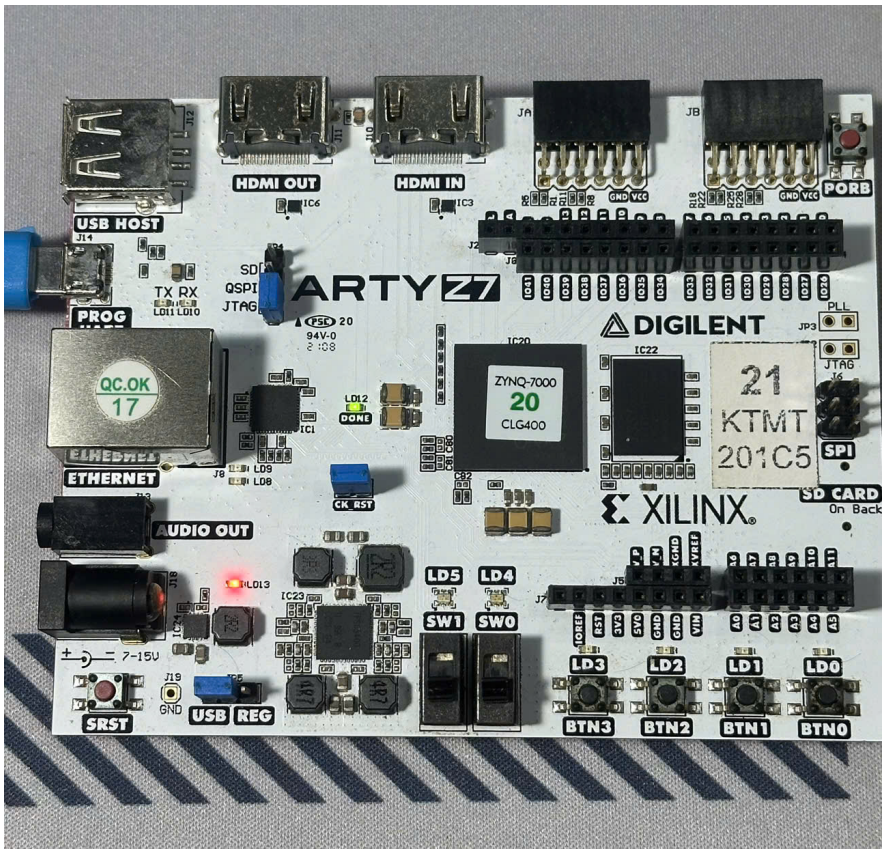
Hệ thống được triển khai và kiểm thử trên bo mạch phát triển **Arty–Z7–20**, sử dụng nền tảng SoC Zynq-7000 tích hợp **Processing System (PS)** và **Programmable Logic (PL)**.
Quy trình chuẩn bị phần cứng
Để đảm bảo quá trình kiểm thử trên phần cứng diễn ra ổn định và chính xác, các bước chuẩn bị phần cứng được thực hiện theo trình tự như sau:

- **Bước 1:** Kiểm tra trạng thái hoạt động của bo mạch, bao gồm nguồn cấp, các đèn báo trạng thái (LED) và các linh kiện ngoại vi, nhằm đảm bảo bo mạch không gặp lỗi phần cứng.
- **Bước 2:** Thiết lập jumper chế độ **JTAG/UART** để cho phép lập trình và giao tiếp dữ liệu giữa bo mạch và máy tính thông qua cổng USB.



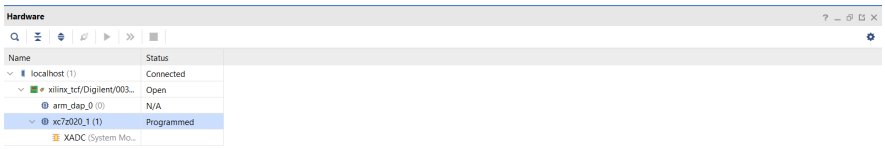
Hình 67: Thiết lập jumper JTAG/UART trên bo mạch

- **Bước 3:** Kết nối cáp USB giữa bo mạch và máy tính, trong đó đầu USB Type-A (hoặc Type-C tùy phiên bản) kết nối với máy tính, và đầu còn lại kết nối với bo mạch Arty–Z7–20.
- **Bước 4:** Cấp nguồn cho bo mạch và thực hiện reset hệ thống bằng nút **SRST** hoặc **PORB** để đảm bảo toàn bộ hệ thống khởi động lại ở trạng thái xác định.



Hình 68: Bo mạch sau khi cấp nguồn và reset thành công

- **Bước 5:** Nạp bitstream và chương trình điều khiển từ môi trường phát triển Vivado/SDK vào bo mạch, đồng thời xác nhận quá trình nạp diễn ra thành công thông qua thông báo trên phần mềm và tín hiệu trạng thái trên bo mạch.



Hình 69: Nạp chương trình thành công vào bo mạch Arty–Z7–20

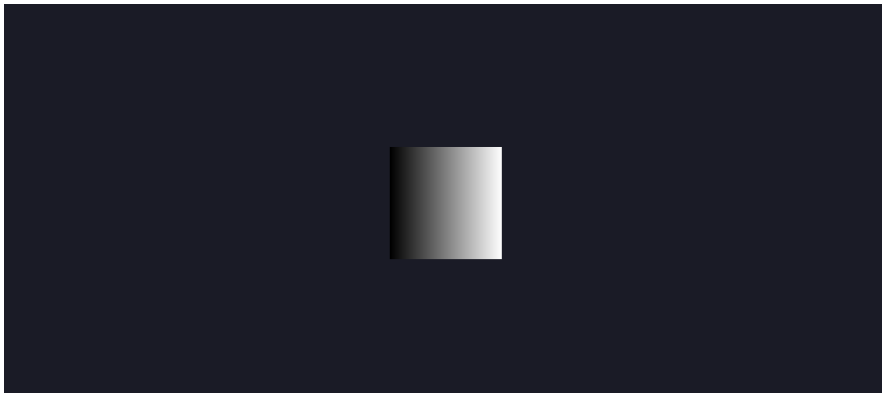
6.8.3.b Kiểm tra các chức năng xoay, lật ảnh và lưu vào bộ nhớ

Các chức năng cần kiểm tra

- Kiểm tra chức năng **xoay ảnh 90° theo chiều kim đồng hồ (Rotate CW)**.
- Kiểm tra chức năng **xoay ảnh 90° ngược chiều kim đồng hồ (Rotate CCW)**.
- Kiểm tra chức năng **lật ảnh theo phương ngang (Mirror Horizontal)**.
- Kiểm tra chức năng **lật ảnh theo phương dọc (Mirror Vertical)**.

- Kiểm tra tính chính xác của dữ liệu ảnh sau xử lý khi được **ghi trở lại bộ nhớ DDR** thông qua kênh S2MM của AXI DMA.
- Kiểm tra sự toàn vẹn dữ liệu trong suốt quá trình truyền, đảm bảo **không mất pixel, không trùng pixel** và đúng thứ tự theo chế độ xử lý đã chọn.

Kiểm tra chức năng xoay ảnh 90° theo chiều kim đồng hồ (Rotate CW)



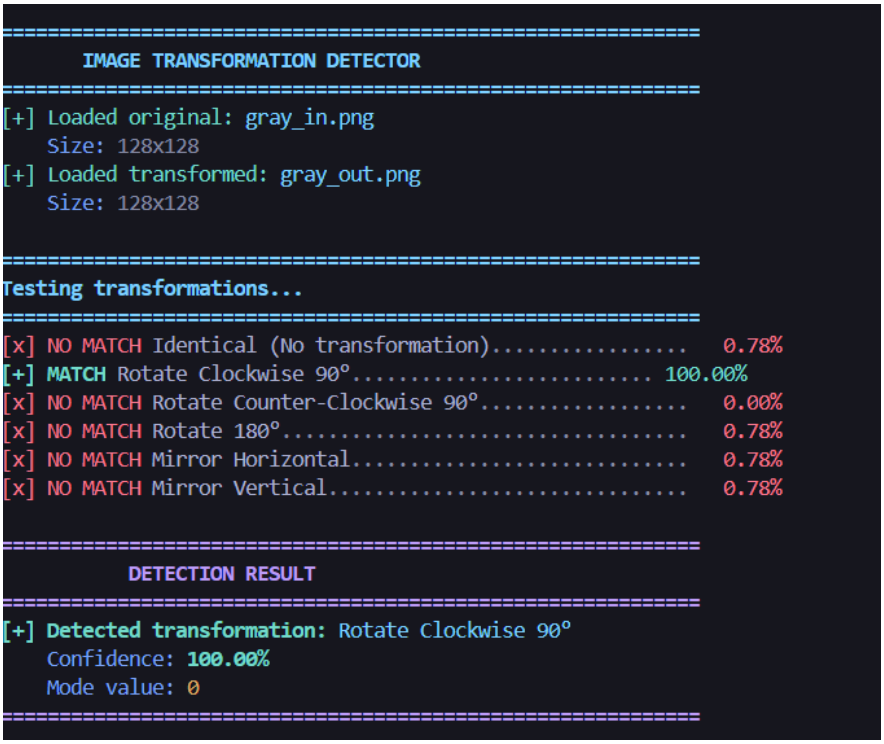
Hình 70: Ảnh đầu vào có kích thước 128x128

```
#ifndef IN_DATA_H
#define IN_DATA_H

typedef uint8_t u8;

// Image data with header (Little Endian format)
// Structure: [Height 4B][Width 4B][Pixel Data]
u8 raw_image_file[] __attribute__ ((aligned (32))) = {
    // --- HEADER (8 Bytes) ---
    // Height = 128
    0x80, 0x00, 0x00, 0x00,
    // Width = 128
    0x80, 0x00, 0x00, 0x00,

    // --- PIXEL DATA ---
    0x00, 0x02, 0x04, 0x06, 0x08, 0x0A, 0x0C, 0x0E, 0x10, 0x12, 0x14, 0x16, 0x18, 0x1A, 0x1C, 0x1E,
    0x20, 0x22, 0x24, 0x26, 0x28, 0x2A, 0x2C, 0x2E, 0x30, 0x32, 0x34, 0x36, 0x38, 0x3A, 0x3C, 0x3E,
    0x40, 0x42, 0x44, 0x46, 0x48, 0x4A, 0x4C, 0x4E, 0x50, 0x52, 0x54, 0x56, 0x58, 0x5A, 0x5C, 0x5E,
    0x60, 0x62, 0x64, 0x66, 0x68, 0x6A, 0x6C, 0x6E, 0x70, 0x72, 0x74, 0x76, 0x78, 0x7A, 0x7C, 0x7E,
    0x80, 0x82, 0x84, 0x86, 0x88, 0x8A, 0x8C, 0x8E, 0x90, 0x92, 0x94, 0x96, 0x98, 0x9A, 0x9C, 0x9E,
    0xA0, 0xA2, 0xA4, 0xA6, 0xA8, 0xAA, 0xAC, 0xAE, 0xB0, 0xB2, 0xB4, 0xB6, 0xB8, 0xBA, 0xBC, 0xBE,
    0xC0, 0xC2, 0xC4, 0xC6, 0xC8, 0xCA, 0xCC, 0xCE, 0xD0, 0xD2, 0xD4, 0xD6, 0xD8, 0xDA, 0xDC, 0xDE,
    0xE0, 0xE2, 0xE4, 0xE6, 0xE8, 0xEA, 0xEC, 0xEE, 0xF0, 0xF2, 0xF4, 0xF6, 0xF8, 0xFA, 0xFC, 0xFF,
    0x00, 0x02, 0x04, 0x06, 0x08, 0x0A, 0x0C, 0x0E, 0x10, 0x12, 0x14, 0x16, 0x18, 0x1A, 0x1C, 0x1E,
    0x20, 0x22, 0x24, 0x26, 0x28, 0x2A, 0x2C, 0x2E, 0x30, 0x32, 0x34, 0x36, 0x38, 0x3A, 0x3C, 0x3E,
```

Hình 71: Dữ liệu ảnh grayscale được chuyển đổi và lưu trữ trong file `in_data.h`

Hình 72: Các pixel mới được in ra màn hình virtual terminal

Hình 73: Ảnh mới được chuyển đổi từ pixel mới



Hình 74: So sánh ảnh giữa ảnh ban đầu và ảnh mới

Luồng hoạt động

Ảnh đầu vào được chuyển đổi sang dạng dữ liệu grayscale và lưu trữ dưới dạng mảng trong file `in_data.h`. Các pixel được hệ thống đọc tuần tự, xử lý và lưu tạm trong bộ nhớ đệm nội bộ. Sau đó, dữ liệu pixel đầu ra được truyền tuần tự qua giao tiếp **UART** và hiển thị trên *virtual terminal*. Cuối cùng, các pixel thu được từ UART được sử dụng để tái tạo ảnh đầu ra nhằm phục vụ việc so sánh và đánh giá kết quả xử lý.

Kiểm tra chức năng xoay ảnh 90° ngược chiều kim đồng hồ (Rotate CCW)

Hình 75: Ảnh đầu vào có kích thước 40x50

```
#ifndef IN_DATA_H
#define IN_DATA_H

#include <stdint.h>

typedef uint8_t u8;

// Image data with header (Little Endian format)
// Structure: [Height 4B][Width 4B][Pixel Data]
u8 raw_image_file[] __attribute__ ((aligned (32))) = {
    // --- HEADER (8 Bytes) ---
    // Height = 40
    0x28, 0x00, 0x00, 0x00,
    // Width = 50
    0x32, 0x00, 0x00, 0x00,

    // --- PIXEL DATA ---
    0x00, 0x05, 0x0A, 0x0F, 0x14, 0x1A, 0x1F, 0x24, 0x29, 0x2E, 0x34, 0x39, 0x3E, 0x43, 0x48, 0x4E,
    0x53, 0x58, 0x5D, 0x62, 0x68, 0x6D, 0x72, 0x77, 0x7C, 0x82, 0x87, 0x8C, 0x91, 0x96, 0x9C, 0xA1,
    0xA6, 0xAB, 0xB0, 0xB6, 0xBB, 0xC0, 0xC5, 0xCA, 0xD0, 0xD5, 0xDA, 0xDF, 0xE4, 0xEA, 0xEF, 0xF4,
    0xF9, 0xFF,
    0x00, 0x05, 0x0A, 0x0F, 0x14, 0x1A, 0x1F, 0x24, 0x29, 0x2E, 0x34, 0x39, 0x3E, 0x43, 0x48, 0x4E,
    0x53, 0x58, 0x5D, 0x62, 0x68, 0x6D, 0x72, 0x77, 0x7C, 0x82, 0x87, 0x8C, 0x91, 0x96, 0x9C, 0xA1,
    0xA6, 0xAB, 0xB0, 0xB6, 0xBB, 0xC0, 0xC5, 0xCA, 0xD0, 0xD5, 0xDA, 0xDF, 0xE4, 0xEA, 0xEF, 0xF4,
    0xF9, 0xFF,
```

Hình 76: Dữ liệu ảnh grayscale được chuyển đổi và lưu trữ trong file `in_data.h`

```
===== IMAGE ROTATION SDK =====
[>]Detected Header: Height = 40, Width = 50
====== Output Data (50x40) =====
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249 249
244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244 244
239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239 239

234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234 234
228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228 228
223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223 223
218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218 218
213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213 213
208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208 208
202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202 202
197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197 197
192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192 192
187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187 187
182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182 182
176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176 176
171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171 171
166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166 166
161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161 161
156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156 156
150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150 150
145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145 145
140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140 140
135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135 135
130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130 130
124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124 124
119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119 119
114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114 114
109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109 109
104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104 104
98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98
93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93 93
88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88
```

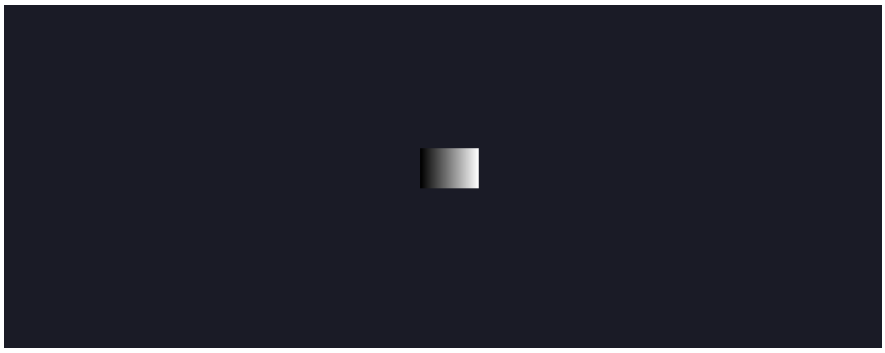
Hình 77: Các pixel mới được in ra màn hình virtual terminal

Hình 78: Ảnh mới được chuyển đổi từ pixel mới

```
============================================================
       IMAGE TRANSFORMATION DETECTOR
============================================================
[+] Loaded original: gray_in.png
    Size: 50x40
[+] Loaded transformed: gray_out.png
    Size: 40x50

============================================================
Testing transformations...
============================================================
[x] NO MATCH Identical (No transformation).................   0.00%
[x] NO MATCH Rotate Clockwise 90°..........................   0.00%
[+] MATCH Rotate Counter-Clockwise 90°..................... 100.00%
[x] NO MATCH Rotate 180°...................................   0.00%
[x] NO MATCH Mirror Horizontal.............................   0.00%
[x] NO MATCH Mirror Vertical...............................   0.00%

============================================================
           DETECTION RESULT
============================================================
[+] Detected transformation: Rotate Counter-Clockwise 90°
    Confidence: 100.00%
    Mode value: 1
------------------------------------------------------------
```

Hình 79: So sánh ảnh giữa ảnh ban đầu và ảnh mới

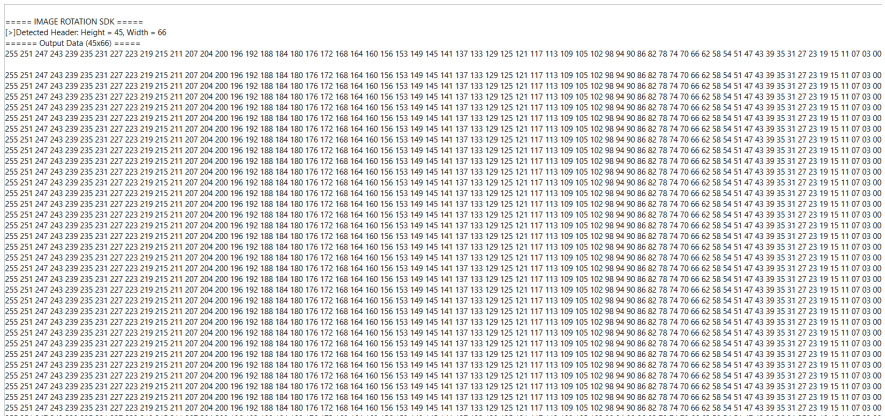
Kiểm tra chức năng lật ảnh theo phương ngang (Mirror Horizontal)



Hình 80: Ảnh đầu vào có kích thước 45x66

```c
#ifndef IN_DATA_H
#define IN_DATA_H

#include <stdint.h>

typedef uint8_t u8;

// Image data with header (Little Endian format)
// Structure: [Height 4B][Width 4B][Pixel Data]
u8 raw_image_file[] __attribute__ ((aligned (32))) = {
    // --- HEADER (8 Bytes) ---
    // Height = 45
    0x2D, 0x00, 0x00, 0x00,
    // Width = 66
    0x42, 0x00, 0x00, 0x00,

    // --- PIXEL DATA ---
    0x00, 0x03, 0x07, 0x0B, 0x0F, 0x13, 0x17, 0x1B, 0x1F, 0x23, 0x27, 0x2B, 0x2F, 0x33, 0x36, 0x3A,
    0x3E, 0x42, 0x46, 0x4A, 0x4E, 0x52, 0x56, 0x5A, 0x5E, 0x62, 0x66, 0x69, 0x6D, 0x71, 0x75, 0x79,
    0x7D, 0x81, 0x85, 0x89, 0x8D, 0x91, 0x95, 0x99, 0x9C, 0xA0, 0xA4, 0xA8, 0xAC, 0xB0, 0xB4, 0xB8,
    0xBC, 0xC0, 0xC4, 0xC8, 0xCC, 0xCF, 0xD3, 0xD7, 0xDB, 0xDF, 0xE3, 0xE7, 0xEB, 0xEF, 0xF3, 0xF7,
    0xFB, 0xFF,
    0x00, 0x03, 0x07, 0x0B, 0x0F, 0x13, 0x17, 0x1B, 0x1F, 0x23, 0x27, 0x2B, 0x2F, 0x33, 0x36, 0x3A,
    0x3E, 0x42, 0x46, 0x4A, 0x4E, 0x52, 0x56, 0x5A, 0x5E, 0x62, 0x66, 0x69, 0x6D, 0x71, 0x75, 0x79,
    0x7D, 0x81, 0x85, 0x89, 0x8D, 0x91, 0x95, 0x99, 0x9C, 0xA0, 0xA4, 0xA8, 0xAC, 0xB0, 0xB4, 0xB8,
    0xBC, 0xC0, 0xC4, 0xC8, 0xCC, 0xCF, 0xD3, 0xD7, 0xDB, 0xDF, 0xE3, 0xE7, 0xEB, 0xEF, 0xF3, 0xF7,
    0xFB, 0xFF,
    0x00, 0x03, 0x07, 0x0B, 0x0F, 0x13, 0x17, 0x1B, 0x1F, 0x23, 0x27, 0x2B, 0x2F, 0x33, 0x36, 0x3A,
    0x3E, 0x42, 0x46, 0x4A, 0x4E, 0x52, 0x56, 0x5A, 0x5E, 0x62, 0x66, 0x69, 0x6D, 0x71, 0x75, 0x79,
    0x7D, 0x81, 0x85, 0x89, 0x8D, 0x91, 0x95, 0x99, 0x9C, 0xA0, 0xA4, 0xA8, 0xAC, 0xB0, 0xB4, 0xB8,
```

Hình 81: Dữ liệu ảnh grayscale được chuyển đổi và lưu trữ trong file `in_data.h`



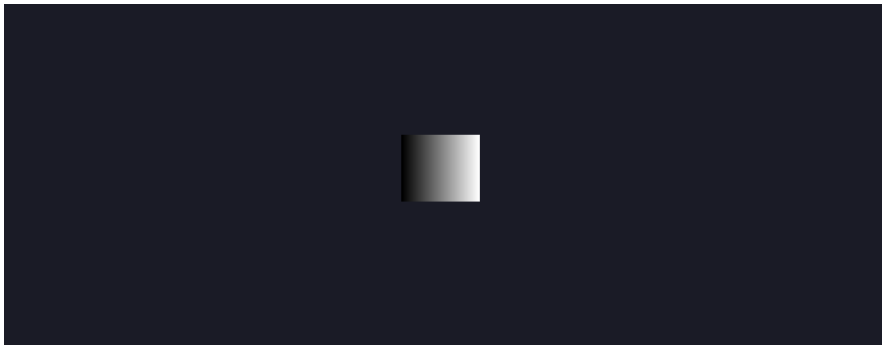
Hình 82: Các pixel mới được in ra màn hình virtual terminal

Hình 83: Ảnh mới được chuyển đổi từ pixel mới

```
============================================================
       IMAGE TRANSFORMATION DETECTOR
============================================================
[+] Loaded original: gray_in.png
    Size: 66x45
[+] Loaded transformed: gray_out.png
    Size: 66x45

============================================================
Testing transformations...
============================================================
[x] NO MATCH Identical (No transformation).................   0.00%
[x] NO MATCH Rotate Clockwise 90°..........................   0.00%
[x] NO MATCH Rotate Counter-Clockwise 90°..................   0.00%
[+] MATCH Rotate 180°..................................... 100.00%
[+] MATCH Mirror Horizontal............................... 100.00%
[x] NO MATCH Mirror Vertical...............................   0.00%

============================================================
           DETECTION RESULT
============================================================
[+] Detected transformation: Rotate 180°
    Confidence: 100.00%
============================================================
```

Hình 84: So sánh ảnh giữa ảnh ban đầu và ảnh mới

Kiểm tra chức năng lật ảnh theo chiều dọc (Mirror Vertical)



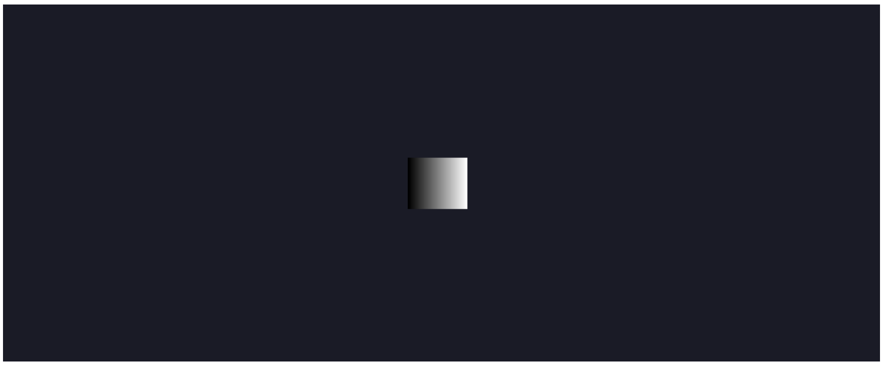
Hình 85: Ảnh đầu vào có kích thước 38x45

```c
#ifndef IN_DATA_H
#define IN_DATA_H

#include <stdint.h>

typedef uint8_t u8;

// Image data with header (Little Endian format)
// Structure: [Height 4B][Width 4B][Pixel Data]
u8 raw_image_file[] __attribute__ ((aligned (32))) = {
    // --- HEADER (8 Bytes) ---
    // Height = 38
    0x26, 0x00, 0x00, 0x00,
    // Width = 45
    0x2D, 0x00, 0x00, 0x00,

    // --- PIXEL DATA ---
    0x00, 0x05, 0x0B, 0x11, 0x17, 0x1C, 0x22, 0x28, 0x2E, 0x34, 0x39, 0x3F, 0x45, 0x4B, 0x51, 0x56,
    0x5C, 0x62, 0x68, 0x6E, 0x73, 0x79, 0x7F, 0x85, 0x8B, 0x90, 0x96, 0x9C, 0xA2, 0xA8, 0xAD, 0xB3,
    0xB9, 0xBF, 0xC5, 0xCA, 0xD0, 0xD6, 0xDC, 0xE2, 0xE7, 0xED, 0xF3, 0xF9, 0xFF,
    0x00, 0x05, 0x0B, 0x11, 0x17, 0x1C, 0x22, 0x28, 0x2E, 0x34, 0x39, 0x3F, 0x45, 0x4B, 0x51, 0x56,
```

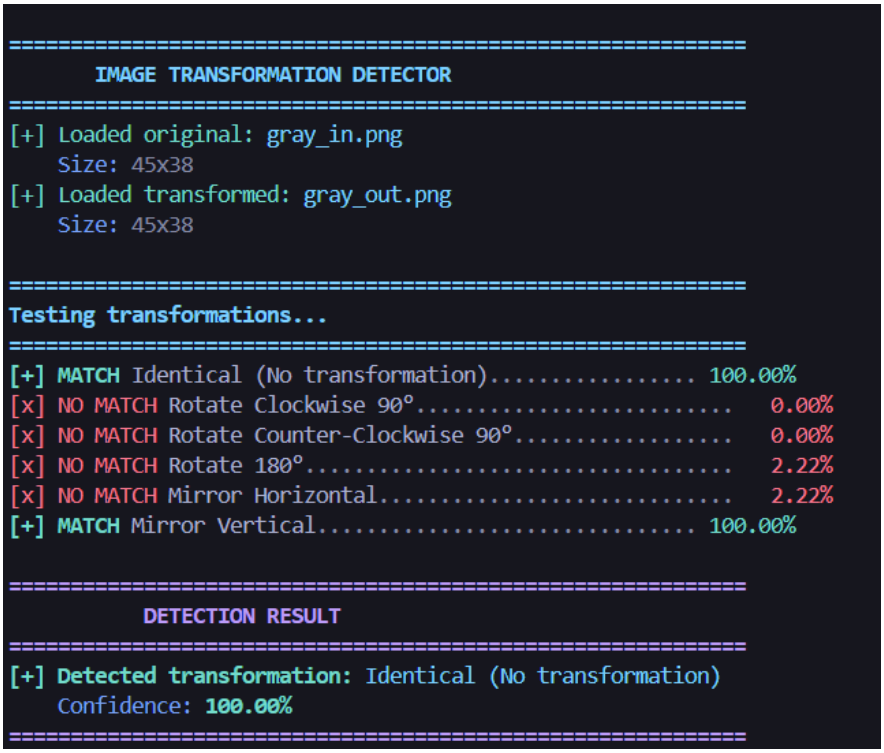
Hình 86: Dữ liệu ảnh grayscale được chuyển đổi và lưu trữ trong file `in_data.h`

```
===== IMAGE ROTATION SDK =====
[>]Detected Header: Height = 38, Width = 45
====== Output Data (38x45) =====
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
00 05 11 17 23 28 34 40 46 52 57 63 69 75 81 86 92 98 104 110 115 121 127 133 139 144 150 156 162 168 173 179 185 191 197 202 208 214 220 226 231 237 243 249 255
```

Hình 87: Các pixel mới được in ra màn hình virtual terminal



Hình 88: Ảnh mới được chuyển đổi từ pixel mới



Hình 89: So sánh ảnh giữa ảnh ban đầu và ảnh mới

Nhận xét

- Kết quả kiểm thử cho thấy các chức năng **xoay ảnh 90° theo chiều kim đồng hồ (Rotate CW)**, **xoay ảnh 90° ngược chiều kim đồng hồ (Rotate CCW)**, **lật ảnh theo phương ngang (Mirror Horizontal)** và **lật ảnh theo phương dọc (Mirror Vertical)** đều được thực hiện đúng theo đặc tả thiết kế.
- Ảnh đầu ra thu được sau xử lý có hình dạng, kích thước và nội dung pixel phù hợp với kết quả mong đợi khi so sánh trực quan với ảnh đầu vào.
- Dữ liệu pixel sau xử lý được truyền và ghi trở lại bộ nhớ **DDR** thông qua kênh **S2MM của AXI DMA** một cách chính xác, không phát hiện hiện tượng mất pixel, trùng pixel hoặc sai thứ tự dữ liệu.
- Quá trình truyền dữ liệu qua giao tiếp **UART** diễn ra ổn định, đảm bảo toàn vẹn dữ liệu phục vụ cho việc tái tạo ảnh đầu ra và đánh giá kết quả.
- Kết quả kiểm thử trên phần cứng thực nghiệm cho thấy hệ thống hoạt động ổn định và nhất quán với kết quả mô phỏng, đáp ứng yêu cầu chức năng đã đề ra.

6.8.3.c Kiểm tra với các hình ảnh có kích thước lớn

Các chức năng cần kiểm tra

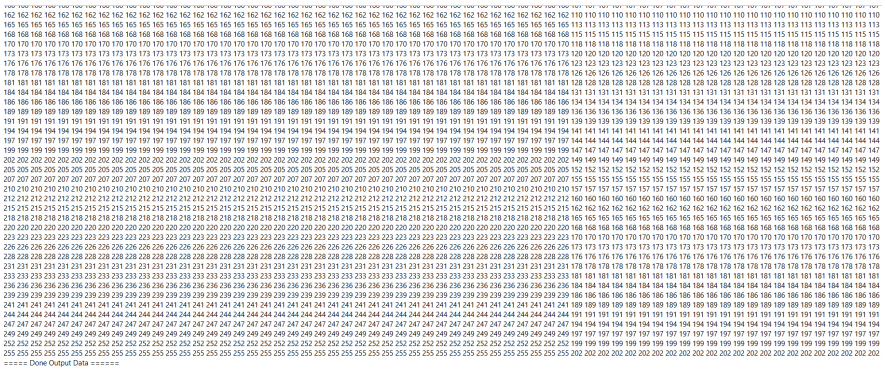
- Đánh giá khả năng xử lý của hệ thống đối với **ảnh có kích thước lớn** (số lượng pixel lớn).
- Kiểm tra **khả năng lưu trữ và quản lý dữ liệu ảnh** trong bộ nhớ, đảm bảo dữ liệu được ghi và đọc đúng địa chỉ, không xảy ra tràn bộ nhớ hoặc ghi đè ngoài vùng cho phép.
- Đánh giá **khả năng đáp ứng của giao tiếp UART** khi truyền dữ liệu ảnh có dung lượng lớn, bao gồm tính ổn định và độ tin cậy của quá trình truyền.
- So sánh và phân tích **thời gian xử lý tổng thể** của hệ thống khi làm việc với các ảnh có kích thước khác nhau, từ ảnh nhỏ đến ảnh lớn, nhằm đánh giá mức độ mở rộng và hiệu năng của thiết kế.

Kích thước ảnh 128x128

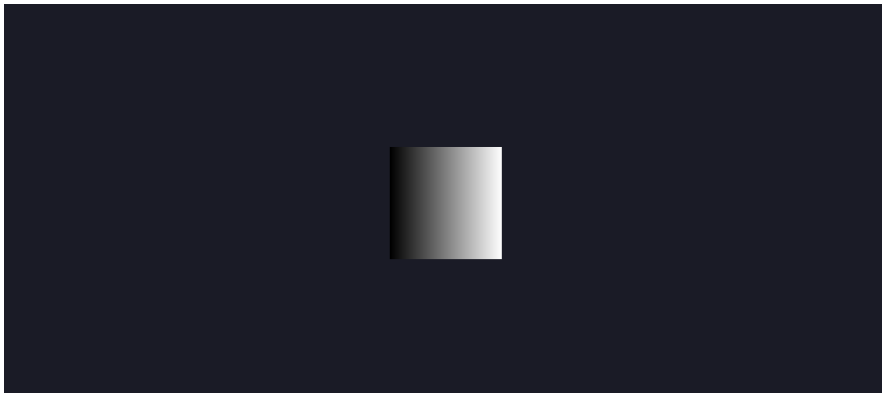
Hình 90: Ảnh đầu vào có kích thước 128x128

```
#ifndef IN_DATA_H
#define IN_DATA_H

typedef uint8_t u8;

// Image data with header (Little Endian format)
// Structure: [Height 4B][Width 4B][Pixel Data]
u8 raw_image_file[] __attribute__ ((aligned (32))) = {
    // --- HEADER (8 Bytes) ---
    // Height = 128
    0x80, 0x00, 0x00, 0x00,
    // Width = 128
    0x80, 0x00, 0x00, 0x00,

    // --- PIXEL DATA ---
    0x00, 0x02, 0x04, 0x06, 0x08, 0x0A, 0x0C, 0x0E, 0x10, 0x12, 0x14, 0x16, 0x18, 0x1A, 0x1C, 0x1E,
    0x20, 0x22, 0x24, 0x26, 0x28, 0x2A, 0x2C, 0x2E, 0x30, 0x32, 0x34, 0x36, 0x38, 0x3A, 0x3C, 0x3E,
    0x40, 0x42, 0x44, 0x46, 0x48, 0x4A, 0x4C, 0x4E, 0x50, 0x52, 0x54, 0x56, 0x58, 0x5A, 0x5C, 0x5E,
    0x60, 0x62, 0x64, 0x66, 0x68, 0x6A, 0x6C, 0x6E, 0x70, 0x72, 0x74, 0x76, 0x78, 0x7A, 0x7C, 0x7E,
    0x80, 0x82, 0x84, 0x86, 0x88, 0x8A, 0x8C, 0x8E, 0x90, 0x92, 0x94, 0x96, 0x98, 0x9A, 0x9C, 0x9E,
    0xA0, 0xA2, 0xA4, 0xA6, 0xA8, 0xAA, 0xAC, 0xAE, 0xB0, 0xB2, 0xB4, 0xB6, 0xB8, 0xBA, 0xBC, 0xBE,
    0xC0, 0xC2, 0xC4, 0xC6, 0xC8, 0xCA, 0xCC, 0xCE, 0xD0, 0xD2, 0xD4, 0xD6, 0xD8, 0xDA, 0xDC, 0xDE,
    0xE0, 0xE2, 0xE4, 0xE6, 0xE8, 0xEA, 0xEC, 0xEE, 0xF0, 0xF2, 0xF4, 0xF6, 0xF8, 0xFA, 0xFC, 0xFF,
    0x00, 0x02, 0x04, 0x06, 0x08, 0x0A, 0x0C, 0x0E, 0x10, 0x12, 0x14, 0x16, 0x18, 0x1A, 0x1C, 0x1E,
    0x20, 0x22, 0x24, 0x26, 0x28, 0x2A, 0x2C, 0x2E, 0x30, 0x32, 0x34, 0x36, 0x38, 0x3A, 0x3C, 0x3E,
```

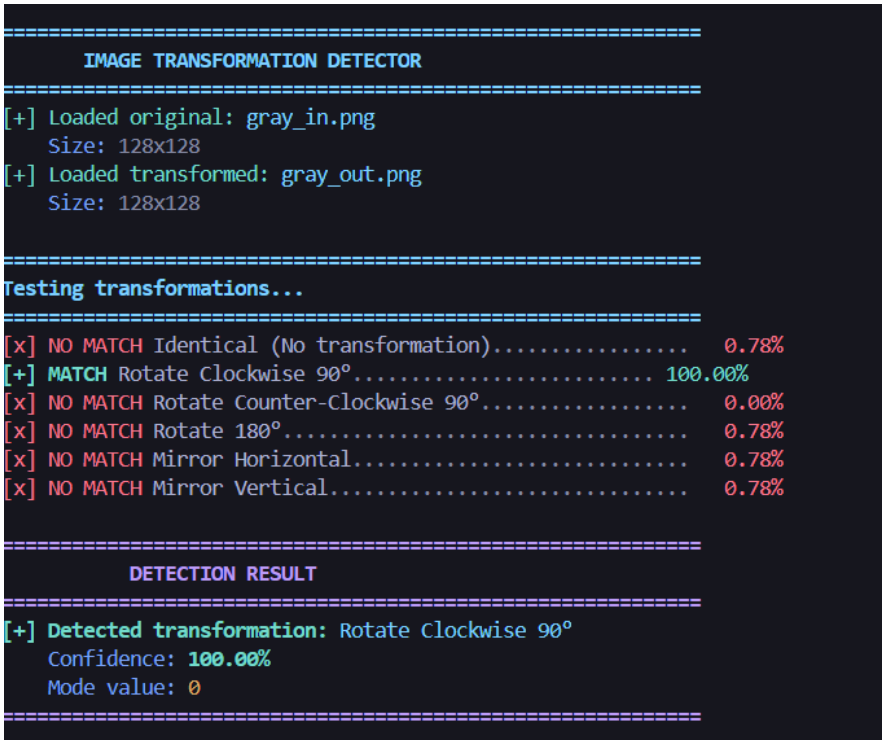
Hình 91: Dữ liệu ảnh grayscale được chuyển đổi và lưu trữ trong file `in_data.h`



Hình 92: Các pixel mới được in ra màn hình virtual terminal

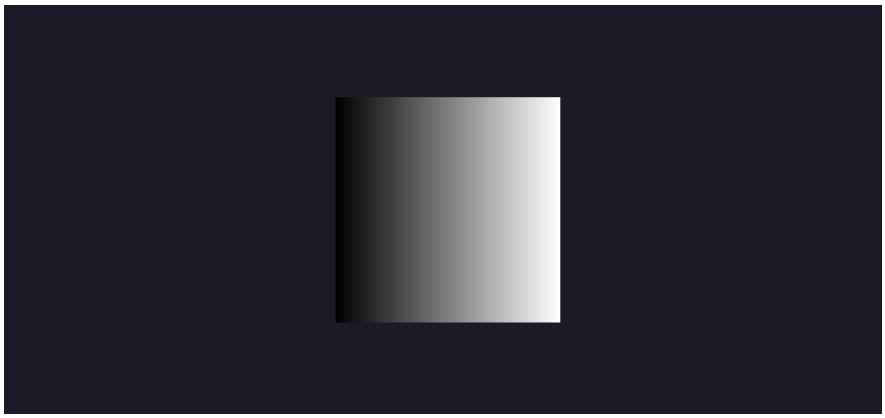


Hình 93: Ảnh mới được chuyển đổi từ pixel mới



Hình 94: So sánh ảnh giữa ảnh ban đầu và ảnh mới

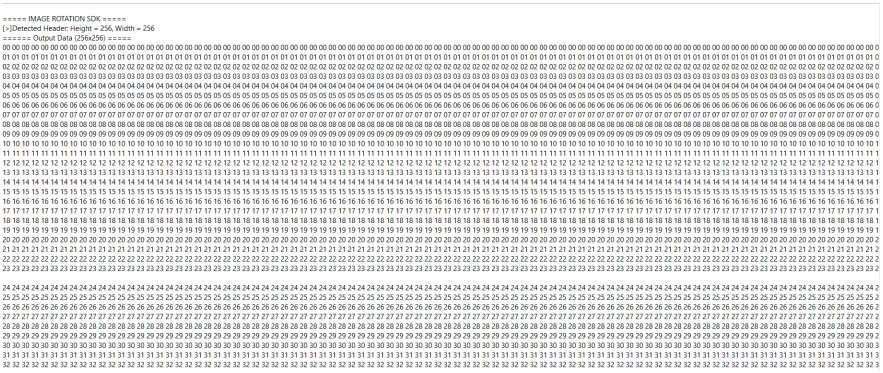
Kích thước ảnh 256x256



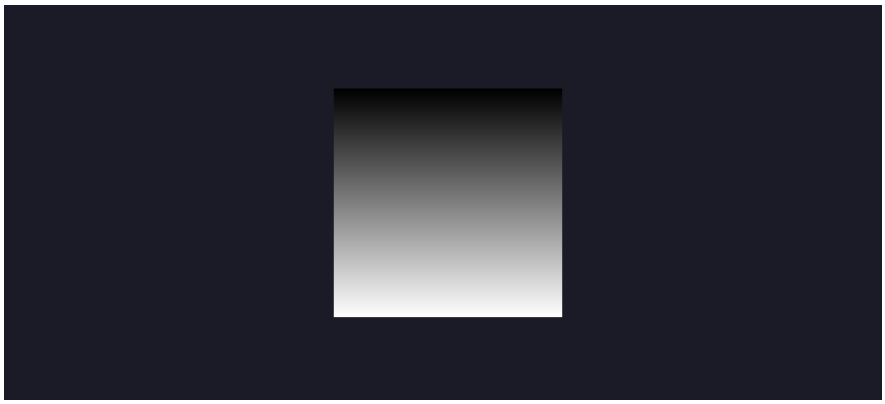
Hình 95: Ảnh đầu vào có kích thước 256x256

```h
#ifndef IN_DATA_H
#define IN_DATA_H

#include <stdint.h>

typedef uint8_t u8;

// Image data with header (Little Endian format)
// Structure: [Height 4B][Width 4B][Pixel Data]
u8 raw_image_file[] __attribute__ ((aligned (32))) = {
    // --- HEADER (8 Bytes) ---
    // Height = 256
    0x00, 0x01, 0x00, 0x00,
    // Width = 256
    0x00, 0x01, 0x00, 0x00,

    // --- PIXEL DATA ---
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
    0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
    0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F,
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
    0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x4D, 0x4E, 0x4F,
    0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5A, 0x5B, 0x5C, 0x5D, 0x5E, 0x5F,
    0x60, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6A, 0x6B, 0x6C, 0x6D, 0x6E, 0x6F,
    0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, 0x79, 0x7A, 0x7B, 0x7C, 0x7D, 0x7E, 0x7F,
```

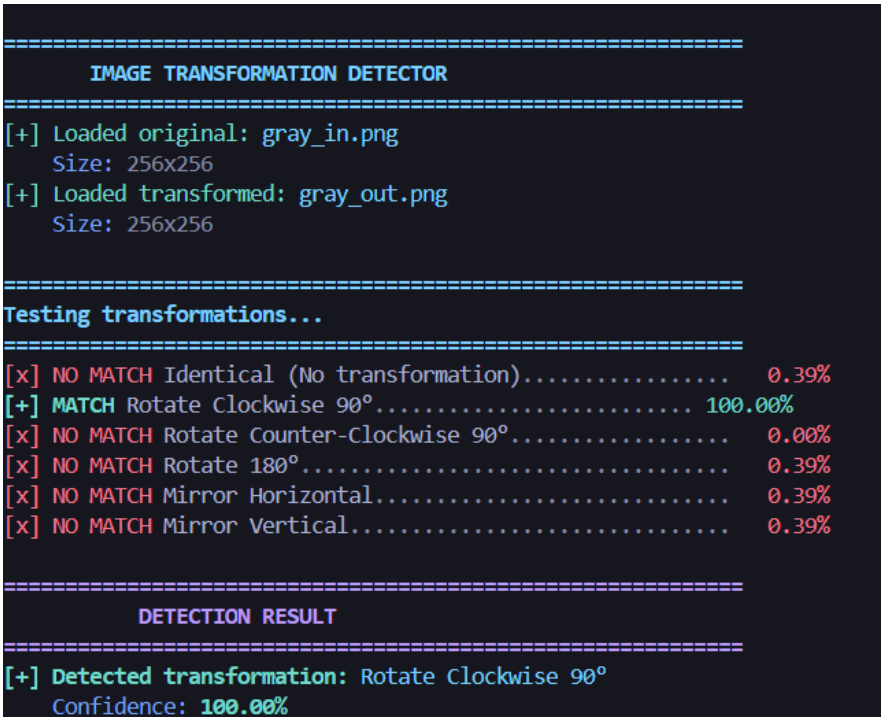
Hình 96: Dữ liệu ảnh grayscale được chuyển đổi và lưu trữ trong file `in_data.h`



Hình 97: Các pixel mới được in ra màn hình virtual terminal

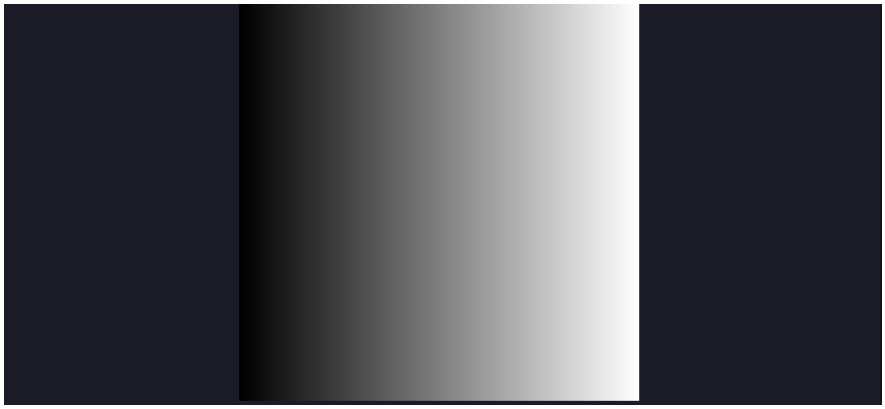


Hình 98: Ảnh mới được chuyển đổi từ pixel mới



Hình 99: So sánh ảnh giữa ảnh ban đầu và ảnh mới

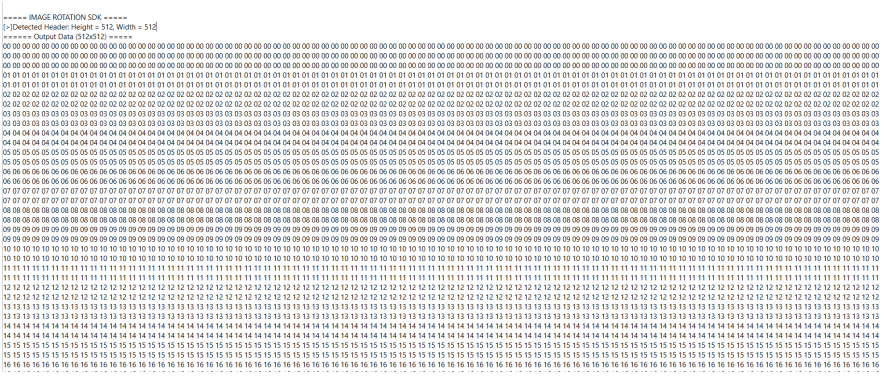
Kích thước ảnh 512x512



Hình 100: Ảnh đầu vào có kích thước 512x512

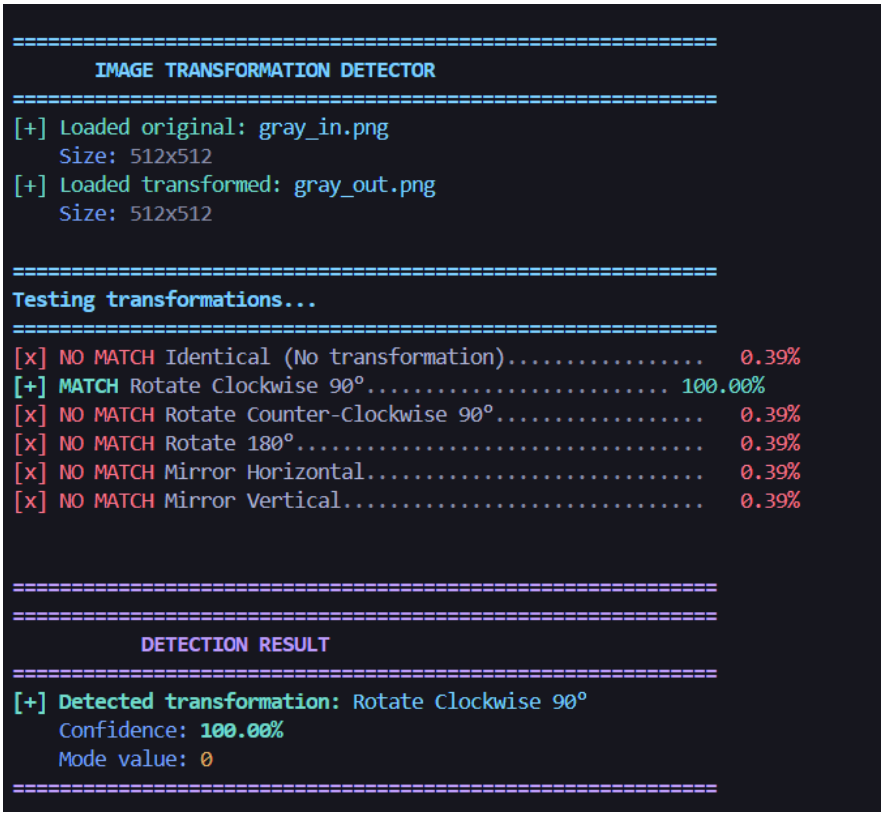
```c
#ifndef IN_DATA_H
#define IN_DATA_H

#include <stdint.h>

typedef uint8_t u8;

// Image data with header (Little Endian format)
// Structure: [Height 4B][Width 4B][Pixel Data]
u8 raw_image_file[] __attribute__ ((aligned (32))) = {
    // --- HEADER (8 Bytes) ---
    // Height = 512
    0x00, 0x02, 0x00, 0x00,
    // Width = 512
    0x00, 0x02, 0x00, 0x00,

    // --- PIXEL DATA ---
    0x00, 0x00, 0x00, 0x01, 0x01, 0x02, 0x02, 0x03, 0x03, 0x04, 0x04, 0x05, 0x05, 0x06, 0x06, 0x07,
    0x07, 0x08, 0x08, 0x09, 0x09, 0x0A, 0x0A, 0x0B, 0x0B, 0x0C, 0x0C, 0x0D, 0x0D, 0x0E, 0x0E, 0x0F,
    0x0F, 0x10, 0x10, 0x11, 0x11, 0x12, 0x12, 0x13, 0x13, 0x14, 0x14, 0x15, 0x15, 0x16, 0x16, 0x17,
    0x17, 0x18, 0x18, 0x19, 0x19, 0x1A, 0x1A, 0x1B, 0x1B, 0x1C, 0x1C, 0x1D, 0x1D, 0x1E, 0x1E, 0x1F,
    0x1F, 0x20, 0x20, 0x21, 0x21, 0x22, 0x22, 0x23, 0x23, 0x24, 0x24, 0x25, 0x25, 0x26, 0x26, 0x27,
    0x27, 0x28, 0x28, 0x29, 0x29, 0x2A, 0x2A, 0x2B, 0x2B, 0x2C, 0x2C, 0x2D, 0x2D, 0x2E, 0x2E, 0x2F,
    0x2F, 0x30, 0x30, 0x31, 0x31, 0x32, 0x32, 0x33, 0x33, 0x34, 0x34, 0x35, 0x35, 0x36, 0x36, 0x37,
    0x37, 0x38, 0x38, 0x39, 0x39, 0x3A, 0x3A, 0x3B, 0x3B, 0x3C, 0x3C, 0x3D, 0x3D, 0x3E, 0x3E, 0x3F,
```

Hình 101: Dữ liệu ảnh grayscale được chuyển đổi và lưu trữ trong file `in_data.h`



Hình 102: Các pixel mới được in ra màn hình virtual terminal

Hình 103: Ảnh mới được chuyển đổi từ pixel mới



Hình 104: So sánh ảnh giữa ảnh ban đầu và ảnh mới

Kích thước ảnh	Số pixel	Dung lượng dữ liệu	Thời gian xử lý (s)	Kết quả xử lý	Nhận xét
128×128	16,384	Nhỏ	56	Thành công	Hệ thống hoạt động ổn định
256×256	65,536	Trung bình	80	Thành công	Không xảy ra lỗi truyền dữ liệu
512×512	262,144	Lớn	180	Thành công	Thời gian xử lý tăng đáng kể
1024×1024	1,048,576	Rất lớn	Không xác định	Không thành công	Không chạy thành công trong việc lưu trữ và xử lý

Bảng 11: *So sánh thời gian xử lý với các kích thước ảnh khác nhau*

Nhận xét

- Qua kết quả kiểm thử với các kích thước ảnh từ nhỏ đến lớn, có thể nhận thấy rằng hiệu năng của hệ thống phụ thuộc mạnh vào số lượng pixel của ảnh đầu vào.
- Khi kích thước ảnh tăng, thời gian xử lý và truyền dữ liệu qua giao tiếp UART tăng đáng kể do cơ chế truyền tuần tự từng pixel.
- Đối với các ảnh có kích thước lên đến 512×512, hệ thống vẫn hoạt động ổn định, đảm bảo xử lý đúng chức năng xoay và lật ảnh, đồng thời giữ nguyên tính toàn vẹn dữ liệu trong suốt quá trình truyền và lưu trữ.
- Tuy nhiên, khi thử nghiệm với ảnh có kích thước lớn hơn, cụ thể là 1024×1024, hệ thống không còn phản hồi sau khi bắt đầu quá trình xử lý.
- Nguyên nhân chủ yếu được xác định là do giới hạn về băng thông của giao tiếp UART, dung lượng bộ nhớ đệm nội bộ, cũng như thời gian xử lý tăng vượt quá khả năng đáp ứng của kiến trúc hiện tại.
- Hiện tượng này cho thấy rằng giải pháp truyền dữ liệu và xuất kết quả thông qua UART chỉ phù hợp với các ảnh có kích thước nhỏ và trung bình. Đối với các ảnh có độ phân giải lớn, cần sử dụng các cơ chế truyền dữ liệu có băng thông cao hơn như **AXI DMA kết hợp với bộ nhớ DDR** để đảm bảo hiệu năng và khả năng mở rộng của hệ thống.

7 Kết luận, thảo luận và khuyến nghị

7.1 Tổng kết kết quả đạt được

Đề tài "Thiết kế hệ thống xoay và lật ảnh trên FPGA sử dụng giao thức AXI" đã được hoàn thành với các kết quả chính sau:

7.1.1 Về mặt thiết kế

- Thiết kế thành công khối AXI Custom IP Image Rotator hỗ trợ 4 chế độ xử lý:
  - Xoay ảnh 90° theo chiều kim đồng hồ (CW)
  - Xoay ảnh 90° ngược chiều kim đồng hồ (CCW)
  - Lật ảnh theo phương ngang (Mirror Horizontal)
  - Lật ảnh theo phương dọc (Mirror Vertical)
- Hiện thực giao thức AXI4-Stream đầy đủ với cơ chế handshaking `VALID/READY` và tín hiệu `TLAST` để đánh dấu khung ảnh.
- Thiết kế kiến trúc multi-bank BRAM (4 banks) giúp tối ưu tài nguyên và hỗ trợ xử lý ảnh lên đến 512×512 pixels.
- Xây dựng máy trạng thái hữu hạn (FSM) với 3 trạng thái (`IDLE`, `RECEIVE`, `SEND`) điều khiển luồng xử lý dữ liệu hiệu quả.
- Tích hợp hệ thống PS-PL hoàn chỉnh trên Zynq SoC, sử dụng AXI DMA để truyền dữ liệu giữa bộ nhớ DDR và khối xử lý ảnh.

7.1.2 Về mặt hiện thực

- Hiện thực thành công hệ thống trên Arty Z7-20 FPGA board.
- Phát triển phần mềm SDK để điều khiển và cấu hình hệ thống từ PS.
- Xây dựng bộ công cụ kiểm thử tự động bao gồm:
  - Script Python để sinh testcase và chuyển đổi format ảnh
  - Script Bash để quản lý test cases
  - Testbench Verilog để mô phỏng RTL
  - Script so sánh và xác nhận kết quả
- Kiểm thử đầy đủ với nhiều kích thước ảnh khác nhau từ 4×4 đến 512×512 pixels.

7.2 Phân tích ưu điểm và hạn chế

7.2.1 Ưu điểm

1. **Hiệu năng cao**
   - Xử lý song song trên phần cứng, throughput cao hơn nhiều so với xử lý tuần tự trên CPU
   - Độ trễ thấp nhờ sử dụng BRAM on-chip
   - Không cần CPU can thiệp trong quá trình xử lý nhờ DMA
2. **Tính linh hoạt**
   - Hỗ trợ kích thước ảnh động từ 1×1 đến 512×512
   - Dễ dàng mở rộng thêm các chế độ xử lý khác
   - Có thể tái sử dụng trong các hệ thống lớn hơn
3. **Tuân thủ chuẩn**

- Sử dụng giao thức AXI chuẩn công nghiệp
- Dễ tích hợp với các IP core khác
- Tương thích với công cụ Vivado và Vitis

4. **Tối ưu tài nguyên**
   - Sử dụng multi-bank BRAM hiệu quả
   - Không sử dụng DSP, tiết kiệm tài nguyên

7.2.2 Hạn chế

1. **Kích thước ảnh giới hạn**
   - Hiện tại chỉ hỗ trợ tối đa 512×512 pixels do giới hạn BRAM
   - Không thể xử lý ảnh rất lớn (megapixel) trực tiếp
   - Cần phải chia nhỏ ảnh hoặc sử dụng bộ nhớ ngoài (DDR) cho ảnh lớn hơn
2. **Chỉ hỗ trợ ảnh grayscale**
   - Chưa hỗ trợ ảnh màu (RGB, YUV)
   - Cần mở rộng độ rộng dữ liệu và logic xử lý cho ảnh màu
3. **Góc quay cố định**
   - Chỉ hỗ trợ quay 90° (CW/CCW)
   - Không hỗ trợ quay góc tùy ý (arbitrary rotation)
   - Không có tính năng scale, crop, hoặc các phép biến đổi phức tạp khác
4. **Chưa tối ưu throughput tối đa**
   - Chưa sử dụng pipeline giữa các stage
   - Chưa sử dụng dual-port BRAM để đọc/ghi đồng thời
   - Có thể cải thiện thêm bằng các kỹ thuật tối ưu phần cứng

7.3 Hướng phát triển trong tương lai

7.3.1 Mở rộng chức năng

1. **Hỗ trợ ảnh màu**
   - Mở rộng độ rộng dữ liệu từ 8-bit lên 24-bit (RGB) hoặc 32-bit (RGBA)
   - Thêm logic xử lý riêng cho từng kênh màu
   - Hỗ trợ nhiều color space: RGB, YUV, HSV
2. **Thêm các phép biến đổi**
   - Rotation góc tùy ý sử dụng interpolation
   - Scaling (zoom in/out) với thuật toán bilinear hoặc bicubic
   - Cropping và affine transformation
   - Các filter: blur, sharpen, edge detection
3. **Xử lý ảnh kích thước lớn**
   - Sử dụng DDR làm buffer cho ảnh lớn
   - Hiện thực thuật toán tiling để xử lý từng phần
   - Tối ưu memory bandwidth với burst access

7.3.2 Cải thiện hiệu năng

1. **Pipeline processing**
   - Thêm các stage pipeline giữa RECEIVE và SEND
   - Sử dụng dual-port BRAM cho đọc/ghi đồng thời
   - Mục tiêu: giảm thời gian xử lý xuống còn $O(H \times W)$ thay vì $O(2 \times H \times W)$
2. **Tối ưu băng thông**
   - Sử dụng burst transfer cho AXI DMA
   - Tăng độ rộng bus dữ liệu (16-bit, 32-bit)
   - Prefetching và buffering thông minh
3. **Multi-channel processing**
   - Xử lý nhiều ảnh song song bằng cách nhân bản IP core
   - Sử dụng AXI interconnect để phân chia băng thông

7.3.3 Tích hợp hệ thống lớn hơn

1. **Video processing pipeline**
   - Tích hợp với camera module (MIPI CSI-2, USB)
   - Xử lý video real-time với frame rate cao
   - Output qua HDMI với resolution 720p/1080p
2. **RISC-V integration**
   - Tích hợp PicoRV32 hoặc Ibex RISC-V core
   - Sử dụng RISC-V để điều khiển thay vì ARM PS
   - Hiện thực multi-core RISC-V cho xử lý phân tán
3. **ASIC design**
   - Chuyển đổi thiết kế sang ASIC sử dụng ASAP7 PDK
   - Sử dụng Yosys để tổng hợp RTL
   - Áp dụng OpenLane flow cho place & route
   - Mục tiêu: giảm tiêu thụ năng lượng và diện tích
4. **Machine Learning integration**
   - Kết hợp với CNN accelerator để nhận dạng đối tượng
   - Xử lý ảnh trước (preprocessing) cho neural network
   - Ứng dụng trong computer vision và robotics

7.4 Những thách thức đã gặp phải

7.4.1 Thách thức về thiết kế

1. **Công thức tính địa chỉ**
   - Khó khăn trong việc xác định công thức ánh xạ tọa độ chính xác
   - Nhiều lần debug để tìm ra lỗi nhầm lẫn giữa row/column và input/output coordinates
   - Giải pháp: Vẽ sơ đồ minh họa và test với ma trận nhỏ
2. **Bank addressing**
   - Ban đầu sử dụng sai bits để chọn bank
   - Dẫn đến lỗi 'x' (unknown) trong simulation
   - Giải pháp: Phân tích lại địa chỉ và sử dụng đúng bits [31:30] hoặc [13:12] tùy MAX_PIXELS

7.4.2 Thách thức về hiện thực

1. **AXI protocol compliance**
   - Đảm bảo tuân thủ đúng chuẩn AXI về handshaking và timing
   - Xử lý các trường hợp edge case (back-pressure, stall)
   - Giải pháp: Tham khảo AXI specification v2.0 và các IP reference design
2. **Cache coherency**
   - Gặp vấn đề dữ liệu không đồng bộ giữa CPU cache và DDR
   - Giải pháp: Flush và invalidate cache trước/sau DMA transfer
3. **Debugging trên hardware**
   - Khó khăn trong việc debug trên FPGA thực so với simulation
   - Giải pháp: Sử dụng ILA (Integrated Logic Analyzer) và UART debug output

7.4.3 Thách thức về kiểm thử

1. **Simulation time**
   - Mô phỏng RTL cho ảnh lớn mất rất nhiều thời gian
   - Giải pháp: Sử dụng Tcl script để extend simulation time và skip irrelevant cycles
2. **Testbench automation**
   - Xây dựng framework kiểm thử tự động mất thời gian
   - Giải pháp: Phát triển từng bước, bắt đầu với manual test rồi tự động hóa

7.5 Kết luận cuối cùng

Đề tài đã hoàn thành đầy đủ các mục tiêu đề ra, bao gồm:

- Thiết kế hardware block xoay và lật ảnh
- Sử dụng giao thức AXI4-Stream
- Tích hợp với Zynq PS và AXI DMA
- Kiểm thử đầy đủ trên simulation và hardware
- Xây dựng bộ công cụ test automation

Thông qua đề tài này, nhóm đã tích lũy được kinh nghiệm quý báu về:

- Thiết kế digital trên FPGA
- Giao thức AXI và tích hợp IP
- Lập trình embedded trên Zynq
- Quy trình kiểm thử và debug phần cứng
- Làm việc nhóm và quản lý dự án

Đề tài không chỉ đáp ứng yêu cầu học thuật mà còn có tính ứng dụng thực tế, có thể mở rộng và tích hợp vào các hệ thống lớn hơn như xử lý video, computer vision, và robotics.

8 Phụ lục

8.1 Kế hoạch thực hiện và phân công công việc

Giai đoạn	Thời gian	Nội dung công việc	Kết quả đạt được
1	Tuần 1	Nghiên cứu yêu cầu bài toán, tìm hiểu nguyên lý xoay ảnh và lật ảnh trong xử lý ảnh số	Xác định yêu cầu hệ thống và phạm vi bài toán
2	Tuần 2	Xây dựng khung chương trình và thuật toán cơ bản cho các phép xoay và lật ảnh	Hoàn thành khung xử lý ảnh ban đầu
3	Tuần 3 – 4	Thiết kế và hoàn thiện thuật toán xử lý ảnh (xoay 90°, lật ngang, lật dọc)	Thuật toán xử lý ảnh hoạt động đúng về mặt chức năng
4	Tuần 5	Kiểm tra logic xoay và lật ảnh với nhiều kích thước ảnh khác nhau bằng mô phỏng	Xác nhận tính đúng đắn của thuật toán
5	Tuần 6	Hoàn thiện báo cáo giữa kỳ, trình bày khung thiết kế ban đầu và ý tưởng xử lý ảnh	Báo cáo giữa kỳ được hoàn thiện
6	Tuần 7 – 8	Nghiên cứu mở rộng, tìm hiểu giao thức AXI Stream và AXI DMA để tích hợp vào hệ thống	Nắm vững cơ chế truyền dữ liệu AXI
7	Tuần 9 – 10	Kiểm thử AXI Stream với các bài toán đơn giản nhằm đánh giá luồng dữ liệu	Xác nhận luồng truyền dữ liệu hoạt động ổn định
8	Tuần 11	Thiết kế và phát triển hệ thống dựa trên khung ban đầu, kết hợp AXI Stream và FSM	Hệ thống xử lý ảnh trên FPGA được hình thành
9	Tuần 12 – 16	Hiện thực đầy đủ các chức năng xử lý ảnh trên FPGA	Hoàn thiện khối Custom IP
10	Tuần 15	Xây dựng testbench và thực hiện mô phỏng chức năng trên Vivado, đồng thời chỉnh sửa các chức năng chính phù hợp	Kết quả mô phỏng đạt yêu cầu
11	Tuần 15 – 16	Thiết kế Block Design, tích hợp Custom IP với AXI DMA và PS, đồng thời chỉnh sửa các chức năng chính phù hợp và chỉnh sửa testbench	Hệ thống sẵn sàng chạy trên phần cứng
12	Tuần 17	Sử dụng SDK để kiểm tra các chức năng xử lý ảnh trên bo mạch Arty–Z7–20	Xác nhận hệ thống hoạt động trên phần cứng
13	Tuần 18 – 19	Kiểm thử toàn bộ chức năng bằng mô phỏng và phần cứng, đánh giá hiệu năng hệ thống	Đánh giá thời gian xử lý và độ ổn định
14	Tuần 20 – 21	Hoàn thiện báo cáo cuối kỳ và tổng hợp kết quả thực nghiệm	Hoàn thành đồ án

Bảng 12: *Kế hoạch thực hiện đồ án theo từng giai đoạn*

8.2 Mã nguồn và tài nguyên liên quan

Toàn bộ mã nguồn, tài liệu thiết kế, tài nguyên và hướng dẫn sử dụng hệ thống được công bố công khai trên GitHub tại địa chỉ sau:

`https://github.com/DgHnG36/Logic-Design-Project-ImageRot-Mir-HCMUT-251.git`

Kho lưu trữ này bao gồm:

- **Thư mục** `main_project/`: chứa toàn bộ tệp tin và cấu trúc dự án chính, bao gồm mã nguồn HDL, cấu hình phần cứng và các thành phần thiết kế được xây dựng trong môi trường Vivado.
- **Thư mục** `reports/`: lưu trữ các báo cáo kỹ thuật, tài liệu mô tả thiết kế, kết quả mô phỏng và đánh giá hệ thống trong quá trình thực hiện dự án.
- **Tập tin** `.gitignore`: dùng để loại trừ các tệp tin trung gian và tệp tin phát sinh tự động bởi phần mềm Vivado, nhằm đảm bảo kho lưu trữ gọn nhẹ và dễ quản lý.
- **Tập tin** `README.md` **và** `LICENSE`: cung cấp thông tin tổng quan về dự án, hướng dẫn sử dụng, cũng như giấy phép phân phối và sử dụng mã nguồn.
- **Các tài liệu tham khảo liên quan**: bao gồm các tài liệu kỹ thuật, tiêu chuẩn giao tiếp, và nguồn tham khảo phục vụ cho việc nghiên cứu và phát triển hệ thống.

Việc chia sẻ mã nguồn giúp đảm bảo tính minh bạch, tái hiện kết quả và phục vụ cho quá trình đánh giá hoặc phát triển tiếp theo.

8.3 Mã nguồn hỗ trợ testcase và công cụ bổ sung

Ngoài mã nguồn chính của hệ thống, dự án còn sử dụng một repository phụ chứa các tệp tin phục vụ việc kiểm thử, mô phỏng và tự động hoá testcase:

`https://github.com/DgHnG36/Tc-Img-Rot-Script.git`

Repository này bao gồm:

- Các script tự động chuyển đổi ảnh đầu vào (ví dụ: từ định dạng `.png` sang dữ liệu nhị phân hoặc mã nguồn dùng cho testbench).
- Dữ liệu ảnh mẫu phục vụ kiểm thử các chế độ xoay và lật ảnh.
- Các tập tin hỗ trợ tạo file `in_data.h` cho SDK hoặc tạo file `.mem` cho mô phỏng.
- Hướng dẫn sử dụng để tái tạo testcase, chuẩn hóa pipeline kiểm thử và so sánh kết quả.

Việc sử dụng repository phụ này giúp:

- Tách biệt rõ ràng giữa mã nguồn chính của hệ thống và công cụ hỗ trợ kiểm thử.
- Đảm bảo quá trình kiểm thử có thể được tái lập và mở rộng cho các bộ dữ liệu khác nhau.
- Tăng tính tái lập cho hệ thống kiểm thử.

9 Các nguồn và tài liệu tham khảo

Tài liệu

[1] D. G. Bailey, *Design for Embedded Image Processing on FPGAs*, John Wiley & Sons, 2011.

[2] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing," in *2009 International Conference on Field Programmable Logic and Applications*, IEEE, 2009, pp. 126–131.

[3] C. Wang and X. Zhou, "Design and implementation of video processing system based on FPGA," in *2011 International Conference on Electronics, Communications and Control (ICECC)*, IEEE, 2011, pp. 2198–2201.

[4] E. Monmasson et al., "FPGAs in industrial control applications," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 2, pp. 224–243, 2011.

[5] ARM Limited, *AMBA AXI and ACE Protocol Specification*, ARM IHI 0022E, 2013.

[6] Xilinx Inc., *Zynq-7000 SoC Technical Reference Manual (UG585)*, v1.12.2, 2018.

[7] Xilinx Inc., *AXI DMA LogiCORE IP Product Guide (PG021)*, v7.1, 2019

[8] Digilent Inc., *Arty Z7 Reference Manual*, Available online: https://digilent.com/reference/programmable-logic/arty-z7/reference-manual, Accessed: 2025.

[9] Digilent Inc., *Vitis: Create Blinky Software Guide*, Available online: https://digilent.com/reference/programmable-logic/guides/vitis-create-blinky-software, Accessed: 2025.

[10] Digilent Inc., *Zedboard Creating Custom IP Cores Tutorial*, Available online: https://digilent.com/reference/learn/programmable-logic/tutorials/zedboard-creating-custom-ip-cores/start, Accessed: 2025.

[11] AMD Inc., *AXI4-Stream Interface (SDFEC Integrated Block)*, Available online: https://docs.amd.com/r/en-US/pg256-sdfec-integrated-block/AXI4-Stream-Interface, Accessed: 2025.

[12] J. Park, *image_rotator Repository*, Available online: https://github.com/jhpark1013/image_rotator/tree/master, Accessed: 2025.

[13] FPGA Developer, *Creating a Custom AXI Streaming IP in Vivado*, Available online: https://www.fpgadeveloper.com/2017/11/creating-a-custom-axi-streaming-ip-in-vivado.html, Accessed: 2025.