

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ

-----o0o-----



Research Project

Design of a Grayscale/Halftone Converter

TP. HỒ CHÍ MINH, THÁNG 8 NĂM 2025

LỜI CẢM ƠN

Em xin cảm ơn anh Lâm Hoàng Nhật (anh Leo) đã hướng dẫn em hoàn thành đề tài. Nhờ thực hiện đề tài “Design of a grayscale/halftone image converter”, em đã có cơ hội hiểu rõ hơn về quá trình và nguyên lý hoạt động của một bộ xử lý ảnh trên phần cứng. Trong quá trình thực hiện đề tài, em đã tích lũy được những kinh nghiệm đáng quý trong việc sử dụng Verilog cũng như các công cụ mô phỏng khác để thực hiện full flow Digital IC design.

Tuy giai đoạn đầu làm có hơi khó khăn do không biết phải bắt đầu từ đâu, nhưng sau những ngày vừa qua em đã vượt qua được. Tuy chưa pass được synthesis do chưa có tool nhưng em cũng đã hiểu được hơn tư duy thiết kế phần cứng là như thế nào, tuân tự các quy trình thực hiện ra sao. Em xin chân thành cảm ơn anh và Thầy Trang đã tạo điều kiện cho em được thực hiện đề tài này ạ

MỤC LỤC

1. Cơ sở lý thuyết	3
1.1 Giới thiệu đề tài	3
1.1.1 Grayscale	4
1.1.2 Halftone Image	5
2. Thiết kế	6
2.1. Block Design Explanation.....	6
2.2.1 ROM (Read only Memory)	6
2.2.2 RAM (Random Access Memory).....	7
2.2.3 Grayscale	8
2.2.4 Error Diffusion	10
2.2.5 Grayscale and Halftone Image Converter	12
2.2.6 Controller.....	13
2.2 Giải thích chi tiết thuật toán	15
2.2.1 Grayscale	15
2.2.2 Error Diffusion	17

DANH SÁCH HÌNH MINH HỌA

Hình 1.1: Khái quát về chức năng của hệ thống	4
Hình 1.2: Minh họa về các điểm ảnh của RGB	4
Hình 2.1: Cấu tạo của ROM	6
Hình 2.2: Cấu tạo của RAM	7
Hình 2.3: Blockdiagram của Grayscale	8
Hình 2.4: Pass Grayscale testbench	9
Hình 2.5: Cấu tạo của một khối Error Diffusion	10
Hình 2.6: Sơ đồ FSM cho Error Diffusion.....	11
Hình 2.7: Pass testbench pattern 1	11
Hình 2.8: Pass testbench pattern 2	11
Hình 2.9: Pass testbench pattern 3	12
Hình 2.10: Pass testbench pattern 4	12
Hình 2.11: Block Diagram của top design.....	12
Hình 2.12: Sơ đồ FSM Controller của Thiết kế.....	13
Hình 2.13: Đã pass Grayscale converter testbench.....	13
Hình 2.14: Đã pass Grayscale lẫn Halftone converter testbench.....	14
Hình 2.15: Minh họa cơ chế của một khối grayscale.....	16
Hình 2.16: Cấu tạo fixed point.....	16
Hình 2.17: Minh họa về fixed - point	16
Hình 2.18: Minh họa về giải thuật rounding.....	17
Hình 2.19: Cấu tạo mặt nạ của giải thuật.....	18

1. Cơ sở lý thuyết

1.1 Giới thiệu đề tài

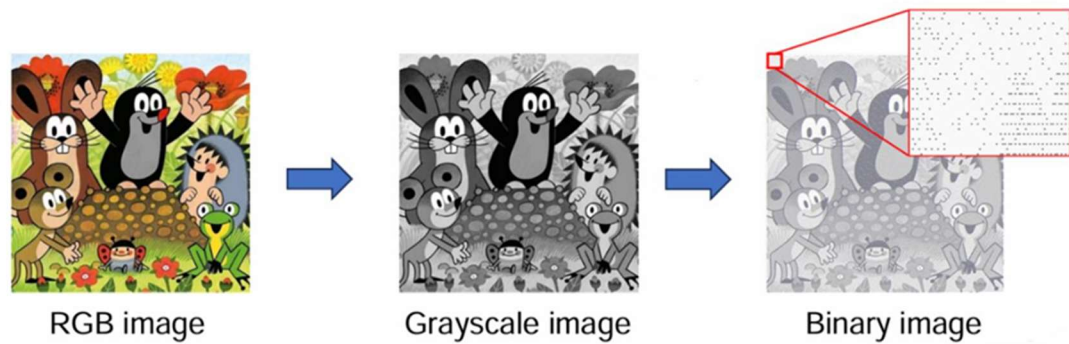
Xử lý ảnh số là một lĩnh vực quan trọng trong công nghệ thông tin và điện tử, với nhiều ứng dụng trong thị giác máy tính, nén ảnh, và hiển thị đa phương tiện. Một trong những bước cơ bản của xử lý ảnh là chuyển đổi ảnh màu (RGB) sang ảnh mức xám (grayscale), nhằm giảm độ phức tạp tính toán và dung lượng lưu trữ trong khi vẫn bảo toàn thông tin trực quan quan trọng. Quá trình này thường sử dụng công thức tuyến tính dựa trên các hệ số trọng số cho ba kênh màu,

Bên cạnh đó, để biểu diễn ảnh nhị phân từ ảnh mức xám, các kỹ thuật tạo ảnh halftone được áp dụng. Trong số đó, thuật toán khuếch tán sai số (error diffusion) là phương pháp phổ biến, cho phép phân bố sai số lượng tử hóa của mỗi điểm ảnh sang các điểm ảnh lân cận, từ đó tạo nên hình ảnh trực quan có độ mịn cao hơn. Điển hình, thuật toán Floyd–Steinberg sử dụng mặt nạ phân phối sai số cho các điểm ảnh bên phải và phía dưới của điểm đang xét mà ta sẽ đi sâu hơn ở phần giải thích thuật toán bên dưới.

Trong các hệ thống phần cứng số, việc hiện thực hóa các bộ chuyển đổi ảnh này đòi hỏi thiết kế mạch số tối ưu về mặt tốc độ, tài nguyên và khả năng mở rộng. Bộ chuyển đổi ảnh mức xám (Grayscale Image Converter) và bộ tạo ảnh halftone (Halftone Image Generator) cần được thiết kế dưới dạng mạch tổ hợp và tuần tự, kết hợp với bộ nhớ ROM/RAM, nhằm thực hiện tuần tự các bước: đọc dữ liệu ảnh, tính toán, và ghi kết quả.

Tóm lại, nghiên cứu này tập trung vào thiết kế và mô phỏng hệ thống phần cứng thực hiện chuỗi xử lý ảnh số gồm:

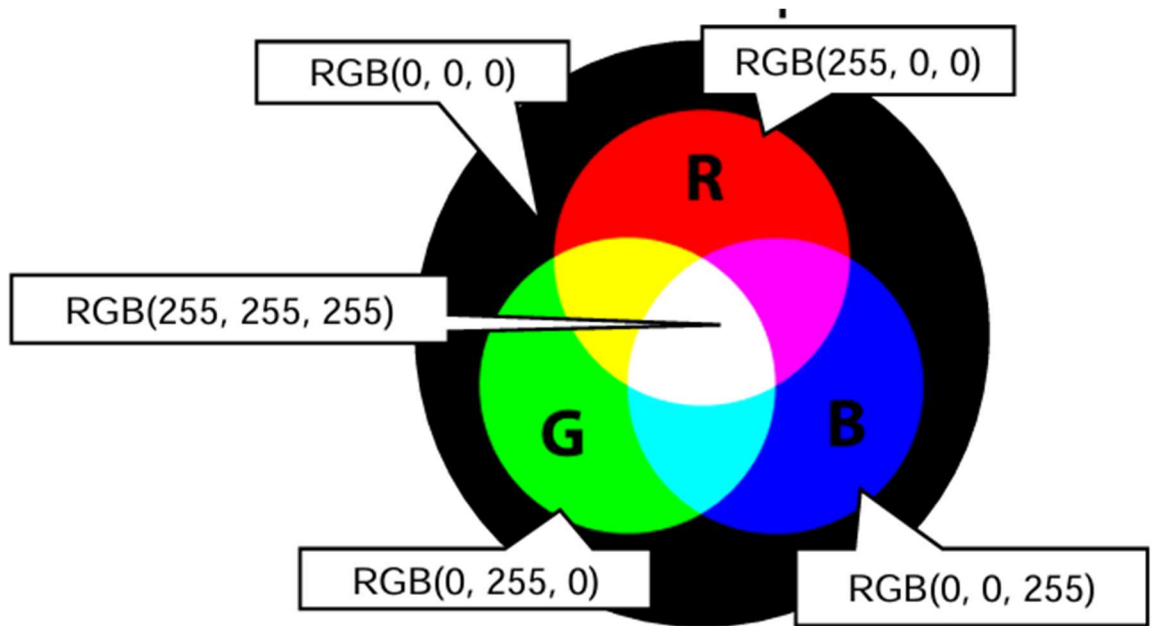
- ❖ *Chuyển đổi ảnh RGB sang ảnh mức xám.*
- ❖ *Khuếch tán sai số để sinh ảnh nhị phân halftone.*



Hình 1.1: Khái quát về chức năng của hệ thống

1.1.1 Grayscale

Trong xử lý ảnh số, hầu hết các ảnh được biểu diễn dưới dạng ảnh màu theo mô hình **RGB (Red, Green, Blue)**. Mỗi điểm ảnh (pixel) trong ảnh màu được đặc trưng bởi ba thành phần: giá trị kênh đỏ (R), xanh lá (G) và xanh dương (B). Trên các hệ thống máy tính phổ biến, mỗi kênh thường được mã hóa bằng 8 bit, cho phép 256 mức cường độ khác nhau (0–255). Do đó, một điểm ảnh màu chuẩn chiếm 24 bit và toàn bộ không gian màu có tới $256^3 = 16,777,216$ khả năng kết hợp.



Hình 1.2: Minh họa về các điểm ảnh của RGB

Tuy nhiên, trong nhiều ứng dụng, việc sử dụng ảnh màu là không cần thiết do dung lượng lưu trữ lớn và độ phức tạp tính toán cao. Để đơn giản hóa xử lý, ảnh thường được chuyển đổi

sang ảnh mức xám (grayscale), trong đó mỗi điểm ảnh chỉ mang một giá trị duy nhất biểu diễn cường độ sáng trong khoảng $[0, 255]$.

Phương pháp chuyển đổi phổ biến dựa trên đặc tính cảm nhận thị giác của con người, vốn nhạy cảm hơn với màu xanh lá so với đỏ và xanh dương. Công thức chuẩn được sử dụng là:

$$y=0.299r+0.587g+0.114b$$

Trong đó:

- r, g, b lần lượt là giá trị thành phần màu đỏ, xanh lá và xanh dương,
- y là giá trị mức xám của điểm ảnh (0–255).

Trong một số ứng dụng phần cứng số, để thuận tiện cho việc cài đặt bằng số cố định (fixed-point), các hệ số này được làm tròn về dạng phân số nhị phân. Một ví dụ điển hình được sử dụng trong thiết kế này là:

$$y=0.3125r+0.5625g+0.125b$$

Kết quả đầu ra y sẽ là giá trị 8 bit đại diện cho điểm ảnh mức xám.

Đáng chú ý, quá trình chuyển đổi từ RGB sang grayscale là **một chiều**. Nói cách khác, từ một giá trị mức xám, ta không thể khôi phục lại đầy đủ thông tin ba kênh màu ban đầu.

Tóm lại, việc chuyển đổi sang ảnh mức xám mang lại các lợi ích chính:

- Giảm dung lượng lưu trữ (từ 24 bit/pixel xuống 8 bit/pixel).
- Đơn giản hóa xử lý trong các bài toán thị giác máy tính và nhận dạng.
- Bảo toàn thông tin trực quan quan trọng về cấu trúc và độ sáng của ảnh.

1.1.2 Halftone Image

Ảnh Halftone dựa vào giải thuật Floyd-Steinberg cho phép chuyển đổi một ảnh mức xám hoặc ảnh màu liên tục thành ảnh nhị phân chỉ bao gồm hai mức đen và trắng. Phương pháp này đặc biệt hữu ích trong các hệ thống in ấn và hiển thị, nơi thiết bị đầu ra bị giới hạn chỉ thể hiện hai mức cường độ sáng. Ý tưởng chính của halftoning là tận dụng đặc tính thị giác của con người: khi quan sát ở khoảng cách đủ xa, mắt người sẽ cảm nhận sự phân bố các điểm đen và trắng như những sắc thái xám khác nhau.

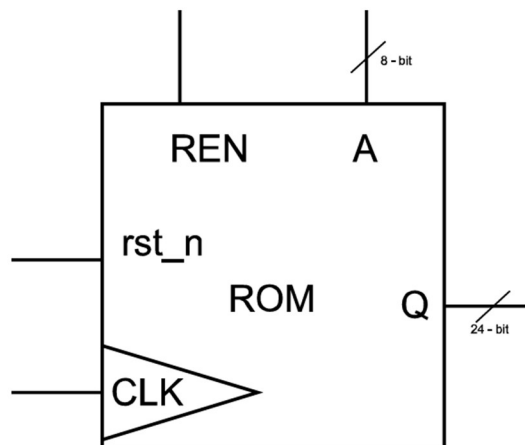
Trong số các thuật toán khuếch tán sai số, phương pháp Floyd–Steinberg được sử dụng rộng rãi nhất. Thuật toán này sử dụng một mặt nạ để phân bố sai số đến bốn điểm ảnh lân cận gồm: điểm bên phải, điểm dưới trái, điểm dưới giữa và điểm dưới phải, với các trọng số lần lượt cho trước. Quá trình quét ảnh được thực hiện tuần tự từ trái sang phải và từ dưới lên trên, đảm bảo rằng sai số được lan truyền hợp lý trong toàn bộ ảnh.

Kết quả của quá trình halftoning là một ảnh nhị phân trong đó mỗi điểm ảnh chỉ nhận giá trị 0 hoặc 255. Mặc dù ảnh này không còn giữ được toàn bộ dải mức xám, nhưng nhờ sự phân bố sai số có kiểm soát, ảnh halftone vẫn thể hiện được các chi tiết và sắc thái xám một cách thuyết phục. Nhờ vậy, kỹ thuật này vừa giúp giảm đáng kể dung lượng lưu trữ và yêu cầu tính toán, vừa đảm bảo chất lượng thị giác cao, phù hợp với các hệ thống phần cứng số trong in ấn và xử lý ảnh thời gian thực.

2. Thiết kế

2.1 Block Design Explanation

2.2.1 ROM (Read only Memory)



Hình 2.1: Cấu tạo của ROM

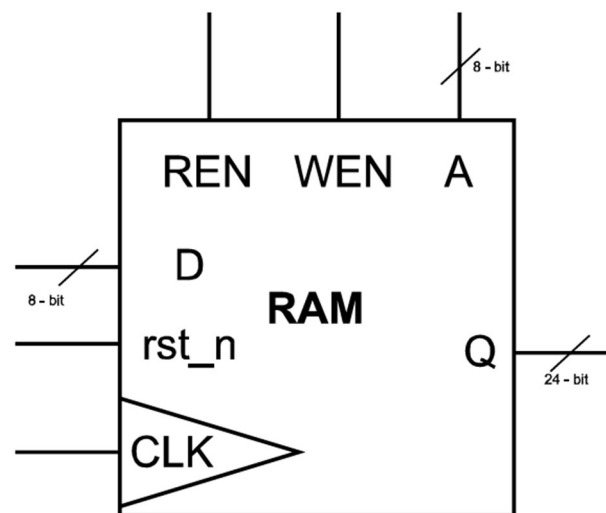
ROM		Kích thước	Ý nghĩa
CLK	Input	1 - bit	Clock signal
REN	Input	1 - bit	Chân Enable read.
A	Input	8 - bit	Pixel address to be read
rst_n	Input	1 - bit	Chân reset (tích cực thấp)
Q	Output	24 - bit	24-bits RGB pixel value

Chức năng: Lưu trữ dữ liệu ảnh đầu vào, thường ở định dạng bitmap hoặc RGB, phục vụ cho quá trình xử lý hình ảnh. ROM dùng để lưu trữ dữ liệu RGB của ảnh, mỗi địa chỉ sẽ lưu trữ mỗi pixel (Do mỗi giá trị $R - G - B$ đều có kích thước là 8 bit, nên data address là 8bit)

Vai trò: Đóng vai trò nguồn cung cấp dữ liệu điểm ảnh (pixel data), cho phép truy cập tuần tự đến từng điểm ảnh trong suốt chuỗi xử lý hình ảnh.

Nguyên lý hoạt động: Bộ điều khiển tạo ra các địa chỉ đọc để truy xuất dữ liệu điểm ảnh được lưu trong ROM. Mỗi lần đọc sẽ trả về các thành phần màu (R, G, B) của điểm ảnh tương ứng, sau đó được truyền đến khối Grayscale block để xử lý tiếp theo

2.2.2 RAM (Random Access Memory)



Hình 2.2: Cấu tạo của RAM

RAM		Kích thước	Ý nghĩa
CLK	Input	1 – bit	Clock signal
REN	Input	1 – bit	Chân Enable Read.
WEN	Input	1 – bit	Chân Enable Write
A	Input	8 – bit	Pixel address to be read
rst_n	Input	1 – bit	Chân reset (tích cực thấp)
D	Input	8 – bit	Data từ GrayScale hoặc Erro Diffusion đã write
Q	Output	24 – bit	24-bits RGB pixel value

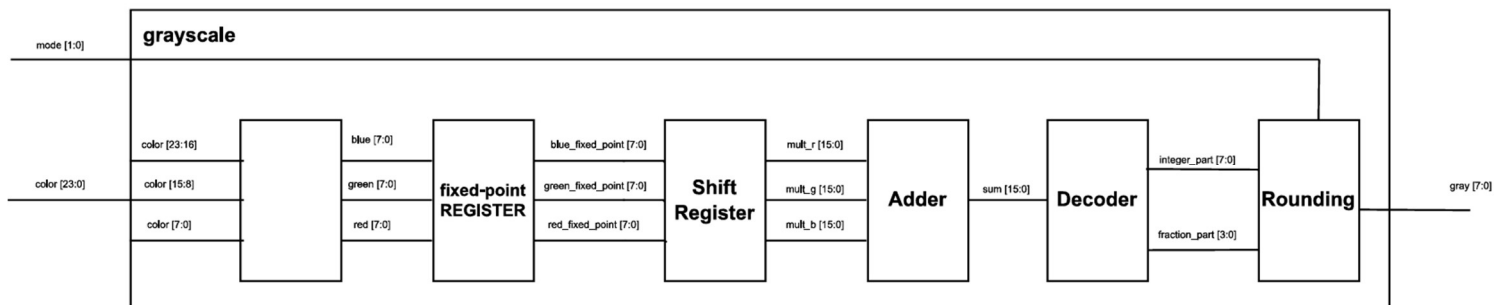
Chức năng: Lưu trữ tạm thời ảnh grayscale trong quá trình xử lý, và lưu trữ kết quả halftone cuối cùng sau quá trình xử lý xong

Vai trò: Cung cấp khả năng truy xuất dữ liệu điểm ảnh theo chế độ đọc–ghi (read–write) nhằm hỗ trợ cho các phép tính trung gian cũng như lưu trữ ảnh đầu ra.

Nguyên lý hoạt động:

- Sau khi đi qua khối Chuyển đổi thang xám (Grayscale Conversion block), các giá trị mức xám của điểm ảnh được ghi vào bộ nhớ RAM như dữ liệu trung gian.
- Trong giai đoạn Error Diffusion process, RAM được truy xuất để đọc và cập nhật giá trị điểm ảnh thông qua việc cộng thêm sai số lượng tử hóa.
- Khi quá trình xử lý kết thúc, RAM chứa ảnh halftone cuối cùng, có thể được xuất ra hoặc sử dụng cho các bước xử lý tiếp theo.

2.2.3 Grayscale



Hình 2.3: Blockdiagram của Grayscale

Grayscale		Kích thước	Ý nghĩa
color	Input	24 – bit	Chứa các data từ // [23:16] Blue, [15:8] Green, [7:0] Red
mode	Input	2 – bit	Rounding mode: 2'b00 for round-up 2'b01 for round-down 2'b10 for round-to-even <i>Các giá trị khác: chỉ lấy phần nguyên</i>
gray	Output	8 – bit	Giá trị GrayScale
clk (optional nếu là sequential)	Input	1 – bit	Clock signal
rst_n (optional nếu là sequential)	Input	1 – bit	Chân reset (tích cực thấp)

Chức năng: Chuyển đổi giá trị 24-bit ([23:16] Blue, [15:8] Green, [7:0] Red) thành giá trị 8-bit bằng công thức $y = r + 0.3125 \cdot 0.5625 \cdot 0.125 \cdot g + b$

Với r, g, b là các giá trị 8 bit **red, green, blue** từ chuỗi RGB ở ngõ vào và y là giá trị grayscale ở ngõ ra

Vai trò: Biến đổi điểm ảnh RGB thành điểm ảnh mức xám, phục vụ cho giai đoạn xử lý halftoning.

Nguyên lý hoạt động:

- Trích xuất giá trị điểm ảnh thành ba thành phần Red, Green và Blue.
- Thực hiện dịch bit (bit-shifting) các thành phần này theo công thức trọng số đã xác định, thay cho phép nhân số học trực tiếp.
- Cộng dồn các giá trị đã dịch, sau đó thực hiện làm tròn theo chế độ được lựa chọn trong *mode*

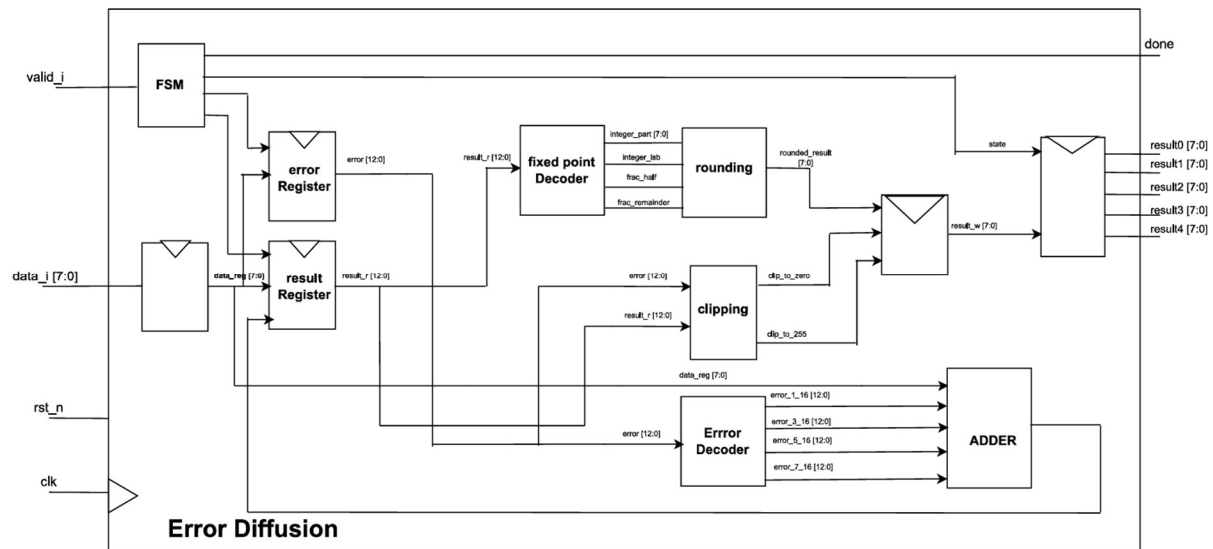
```

# ----- Pattern 10: 74478c -----
# Round-up is correct.
# Round-down is correct.
# Round-to-even is correct.
#
# *****
# ** Lab2 - RTL **
# *****
#
# ** Congratulations !! **
#
# ** SIMULATION PASS !! **
#
# *****
#
#
#
# ** Note: $finish : C:/Users/Ms Khanh/OneDrive - hcmut.edu.vn/Desktop/img_proj/grayscale_tb.v(110)
# Time: 15 ns Iteration: 0 Instance: /grayscale_tb
#

```

Hình 2.4: Pass Grayscale testbench

2.2.4 Error Diffusion



Hình 2.5: Cấu tạo của một khối Error Diffusion

Error Diffusion		Kích thước	Ý nghĩa
clk	Input	1 – bit	Clock signal
rst_n	Input	1 – bit	Chân reset tích cực thấp
valid_i	Input	1 – bit	Chân valid cho phép hoạt động của FSM
data_i	Input	8 – bit	Giá trị GrayScale
result0	Output	8 – bit	Quantized Center Pixel
result1	Output	8 – bit	Diffused Right Pixel
result2	Output	8 – bit	Diffused Upper Right Pixel
result3	Output	8 – bit	Diffused Upper Center Pixel
result4	Output	8 – bit	Diffused Upper Left Pixel
done	Output	1 – bit	Diffusion done

Chức năng: Áp dụng kỹ thuật error diffusion halftoning, điển hình là thuật toán Floyd–Steinberg, để chuyển đổi ảnh mức xám sang ảnh halftone.

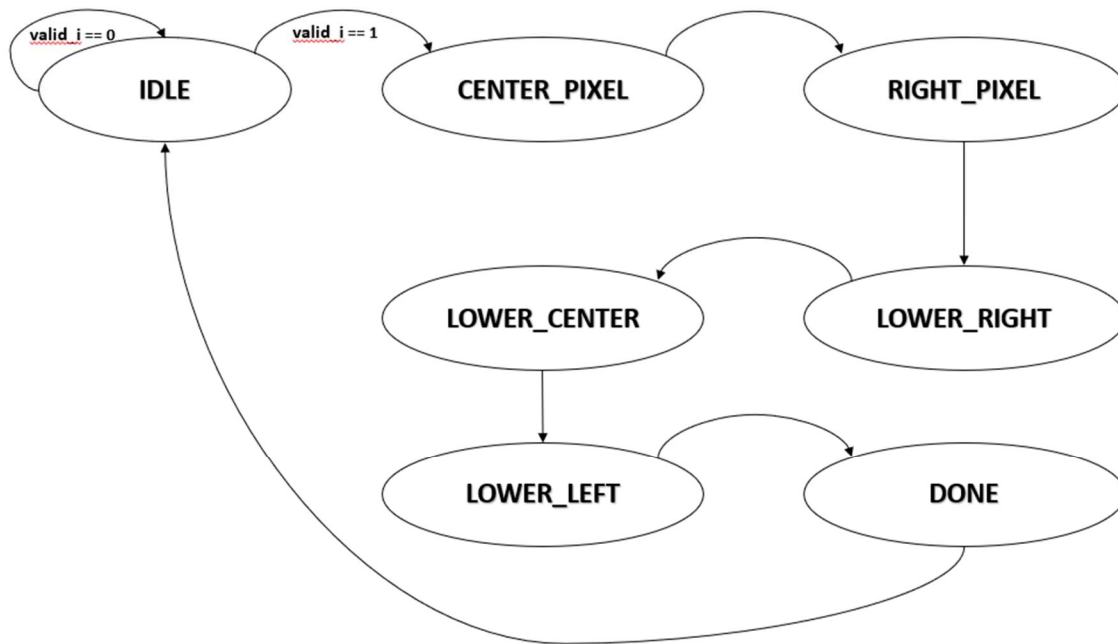
Vai trò: Sinh ra giá trị biểu diễn nhị phân (hoặc reduced-level) của ảnh nhằm mô phỏng continuous tone đồng thời bảo toàn những điểm chi tiết cần thiết.

Nguyên lý hoạt động:

1. So sánh giá trị mức xám hiện tại với ngưỡng (threshold).
2. Xuất ra giá trị **0** hoặc **255** tùy theo kết quả so sánh. (Đồng thời đảm bảo giá trị luôn nằm trong ngưỡng này)

3. Tính sai số lượng tử giữa giá trị gốc và giá trị đã nhị phân hoá.
4. Phân bố sai số này đến các điểm ảnh lân cận dựa theo bộ trọng số xác định trước (Floyd–Steinberg kernel: 7/16, 3/16, 5/16, 1/16).

Sơ đồ FSM



Hình 2.6: Sơ đồ FSM cho Error Diffusion

```

# ----- Pattern 1 start simulation -----
# Center pixel is correct!
# Right pixel is correct!
# Lower right pixel is correct!
# Lower center pixel is correct!
# Lower left pixel is correct!
#
# *****
# **                                     ** |__| |
# ** Congratulations !! ** / 0.0 |
# ** SIMULATION PASS !! ** / ^ ^ ^ \ |
# **                                     ** | ^ ^ ^ |w|
# ** ***** \m__m_|_|
# Your cycle count:      8.
#
#
#

```

Hình 2.7: Pass testbench pattern 1

```

# ----- Pattern 2 start simulation -----
# Center pixel is correct!
# Right pixel is correct!
# Lower right pixel is correct!
# Lower center pixel is correct!
# Lower left pixel is correct!
#
# *****
# **                                     ** |__| |
# ** Congratulations !! ** / 0.0 |
# ** SIMULATION PASS !! ** / ^ ^ ^ \ |
# **                                     ** | ^ ^ ^ |w|
# ** ***** \m__m_|_|
# Your cycle count:      8.
#
#
#

```

Hình 2.8: Pass testbench pattern 2

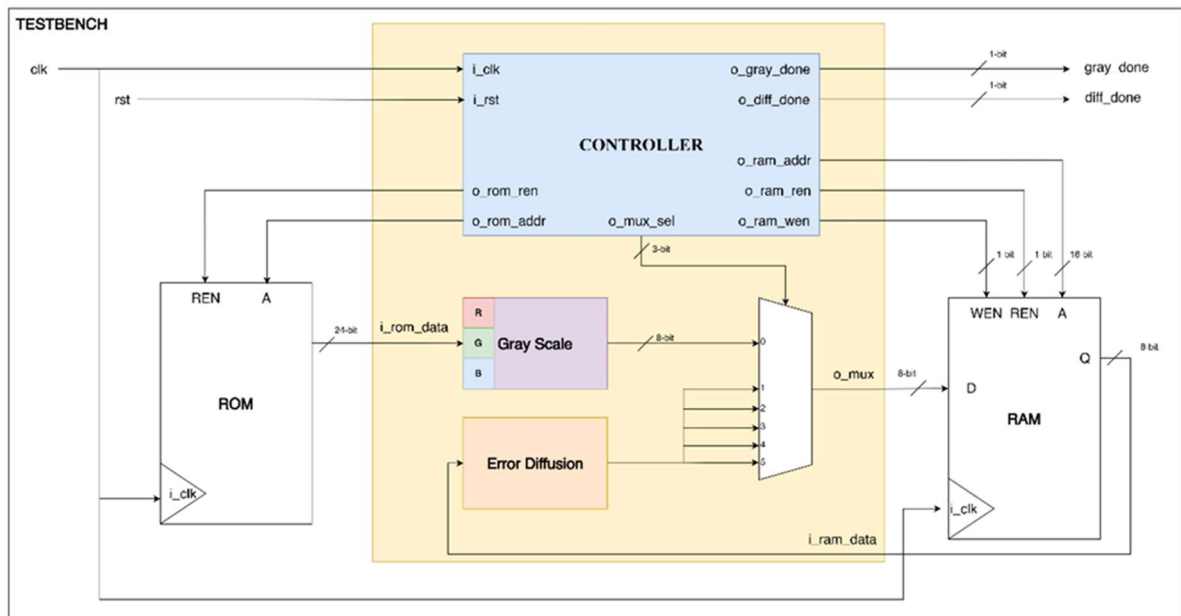
```
# ----- Pattern 3 start simulation -----
# Center pixel is correct!
# Right pixel is correct!
# Lower right pixel is correct!
# Lower center pixel is correct!
# Lower left pixel is correct!
#
# *****
# **                                     |_| |
# ** Congratulations !!                / 0.0 |
# **                                     / ^ ^ \ |
# ** SIMULATION PASS !!                / ^ ^ ^ \ |
# **                                     | ^ ^ ^ ^ |w|
# **                                     \m__m__|_|
# *****
# Your cycle count:      8.
#
```

Hình 2.9: Pass testbench pattern 3

```
# ----- Pattern 4 start simulation -----
# Center pixel is correct!
# Right pixel is correct!
# Lower right pixel is correct!
# Lower center pixel is correct!
# Lower left pixel is correct!
#
# *****
# **                                     |_| |
# ** Congratulations !!                / 0.0 |
# **                                     / ^ ^ \ |
# ** SIMULATION PASS !!                / ^ ^ ^ \ |
# **                                     | ^ ^ ^ ^ |w|
# **                                     \m__m__|_|
# *****
# Your cycle count:      8.
#
```

Hình 2.10: Pass testbench pattern 4

2.2.5 Grayscale and Halftone Image Converter



Hình 2.11: Block Diagram của top design

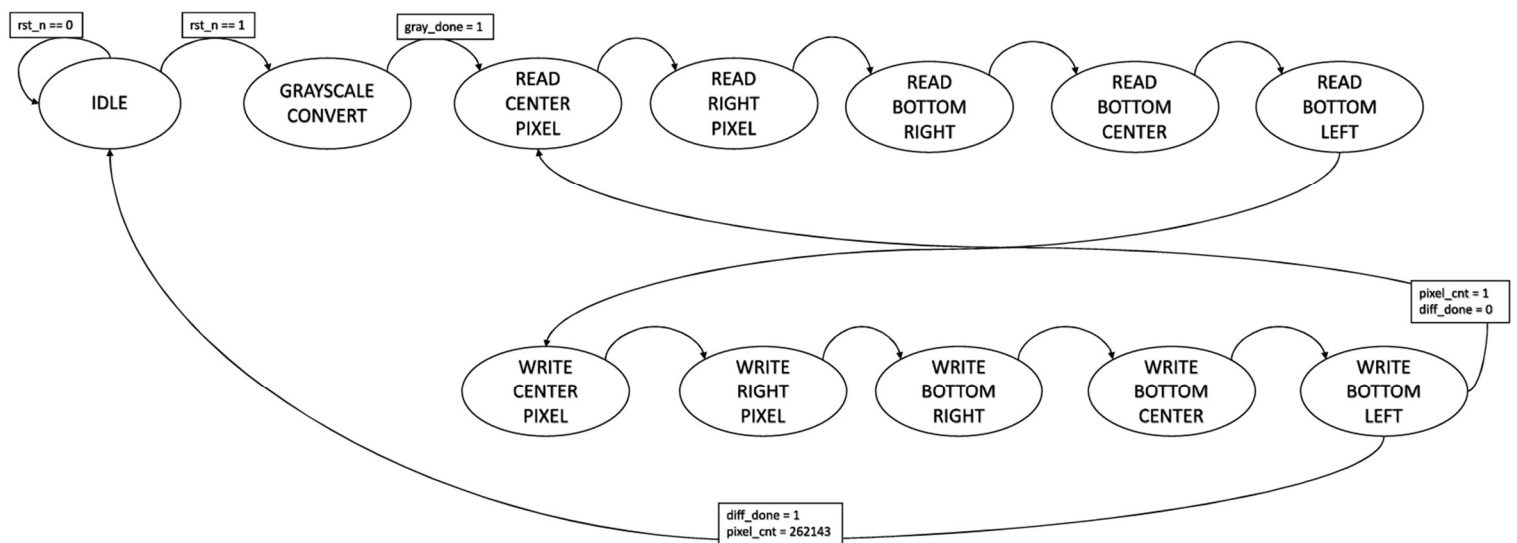
Thiết kế bao gồm 1 bộ nhớ ROM để lưu ảnh RGB, 1 khối grayscale để chuyển ảnh màu sang ảnh xám, 1 khối error diffusion để chuyển ảnh xám sang ảnh halftone, 1 bộ nhớ ROM để chứa ảnh sau khi xử lý và một controller để điều khiển toàn bộ hoạt động của hệ thống.

Quy trình hoạt động của hệ thống được mô tả như sau:

1. Input hình ảnh gốc sẽ được lưu trữ sẵn trong ROM.
2. Tín hiệu ảnh từ ROM sẽ được đưa vào khối xử lý Grayscale, chuyển đổi ảnh màu sang ảnh xám.

3. Ảnh xám sau khi xử lý sẽ được ghi vào RAM.
4. Tiếp theo, ảnh xám được đọc lại từ RAM và đưa vào khối xử lý Error Diffusion để tạo ảnh halftone.
5. Quá trình xử lý sẽ được điều khiển bởi Controller. Controller trong hệ thống được thiết kế theo kiểu Single Cycle, nghĩa là mỗi tác vụ sẽ được thực hiện trong một chu kỳ clock.

2.2.6 Controller



Hình 2.12: Sơ đồ FSM Controller của Thiết kế



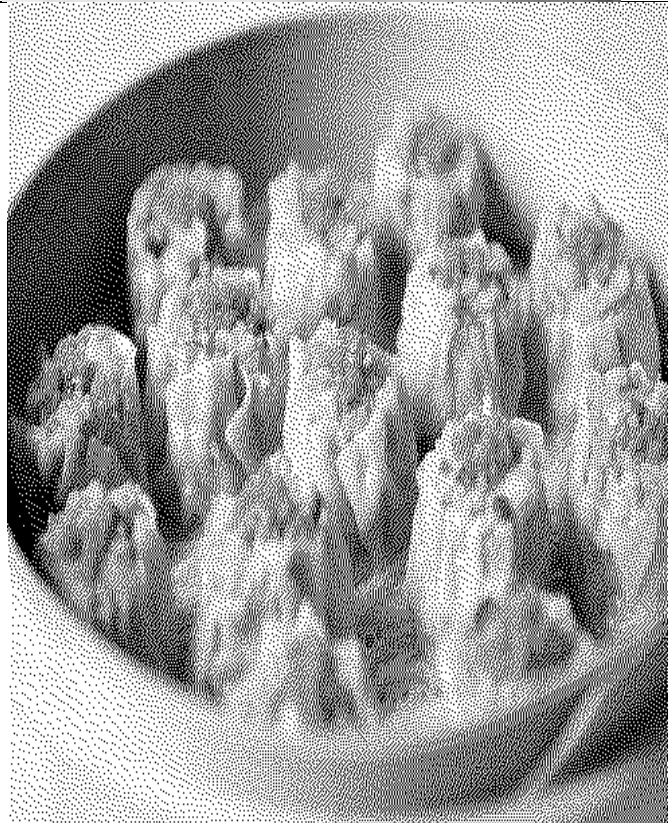

```
# Plot grayscale image ...
# The grayscale image had been generated in the path "../sim/image/gray_Tux.bmp"
#
#
# Grayscale is OK
#
#
#
#
# *****
#          Lab6 - RTL          **
# *****
#          **                  **
#          **                  **      |__||
#          ** Congratulations !! **      / 0.0 |
#          **                  **      /_____|
#          ** Simulation PASS!! **      / ^ ^ ^ \ |
#          **                  **      | ^ ^ ^ ^ |w|
#          **                  **      \m__m_|_|
#          *****
#
#
#
```

Hình 2.13: Đã pass Grayscale converter testbench


```
#
#
# Grayscale is OK
# Waiting the design assert the diff_done signal ....
# Plot dot image ...
# The dot image had been generated in the file "sim/image"
#
#
# Error diffusion is OK
#
#
#
#
# *****
#          Lab6 - RTL          **
# *****
#
#          **
# Congratulations !!          **
#
#          **
# Simulation PASS!!          **
#          **
# *****
#          \m__m_|_|
#          / 0.0 |
#          /_____|
#          / ^ ^ ^ \ |
#          | ^ ^ ^ ^ |w|
#          \m__m_|_|
```

Hình 2.14: Đã pass Grayscale lẫn Hafitone converter testbench

	Image1	Image2
Original		

Grayscale		
Halftone		

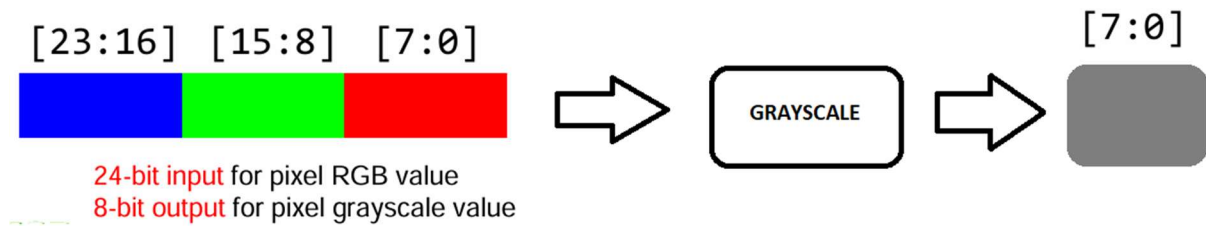
2.2 Giải thích chi tiết thuật toán

2.2.1 Grayscale

Để thuận tiện cho việc cài đặt bằng số cố định (fixed-point), các hệ số này được làm tròn về dạng phân số nhị phân. Một ví dụ điển hình được sử dụng trong thiết kế này là:

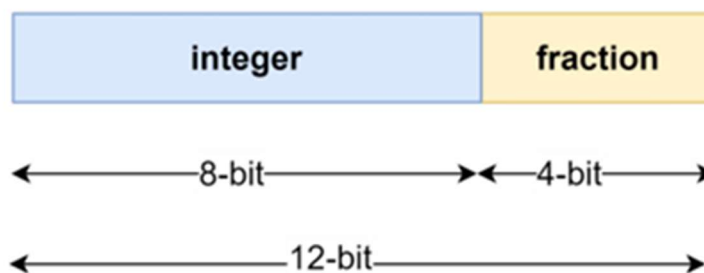
$$y=0.3125r+0.5625g+0.125b$$

Kết quả đầu ra y sẽ là giá trị **8 bit đại diện cho điểm ảnh mức xám**



Hình 2.15: Minh họa cơ chế của một khối grayscale

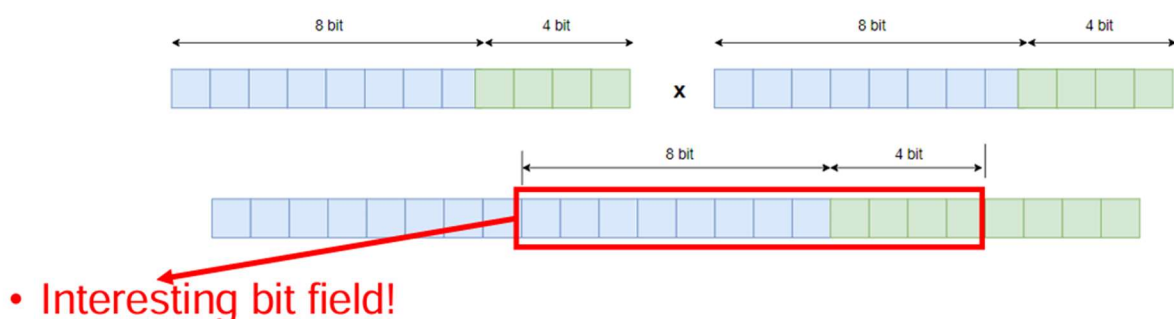
Nhưng trong tư duy thiết kế phần cứng ta không dùng phép nhân “*” vì sẽ gây tốn tài nguyên không cần thiết và gây khó khăn cho quá trình mapping ở giai đoạn synthesis. Mà ta sẽ dùng đến fixed – point format



Hình 2.16: Cấu tạo fixed point

Giải thuật:

Ta sử dụng phép dịch bit, nhưng vì y max chỉ là 255 do (r: 8bit, g: 8bit, b: 8bit) nên ta chỉ quan tâm đến 8bit thấp integer và 4 bit cao ở fraction



Hình 2.17: Minh họa về fixed - point

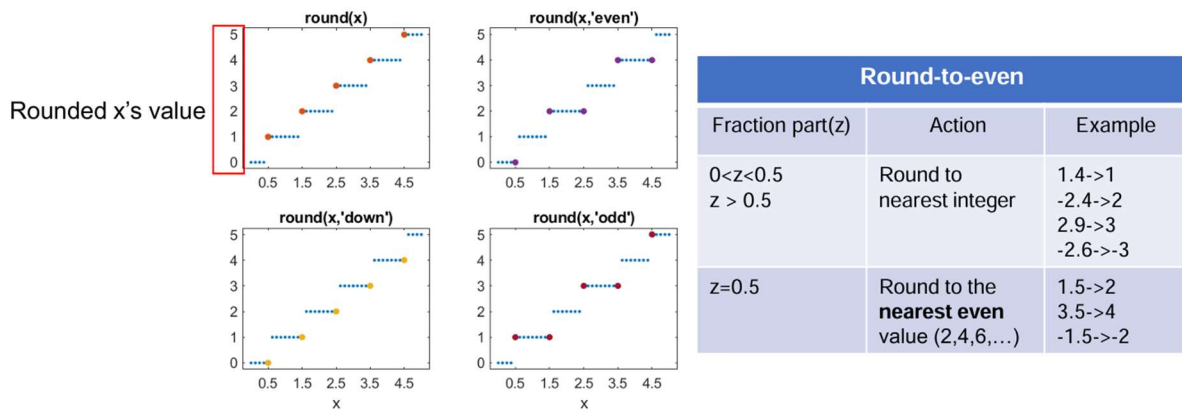
Ví dụ: Để thực hiện 0.3125_r

Bằng với $red \times \frac{5}{16} = \frac{4+1}{16} = \frac{1}{4} + \frac{1}{16}$ tương ứng với giải thuật

$$(red_fixed_point \gg 2) + (red_fixed_point \gg 4)$$

do phép chia với cơ số 2 là phép dịch bit sang phải

Và ta sử dụng giải thuật rounding (làm tròn)



Hình 2.18: Minh họa về giải thuật rounding

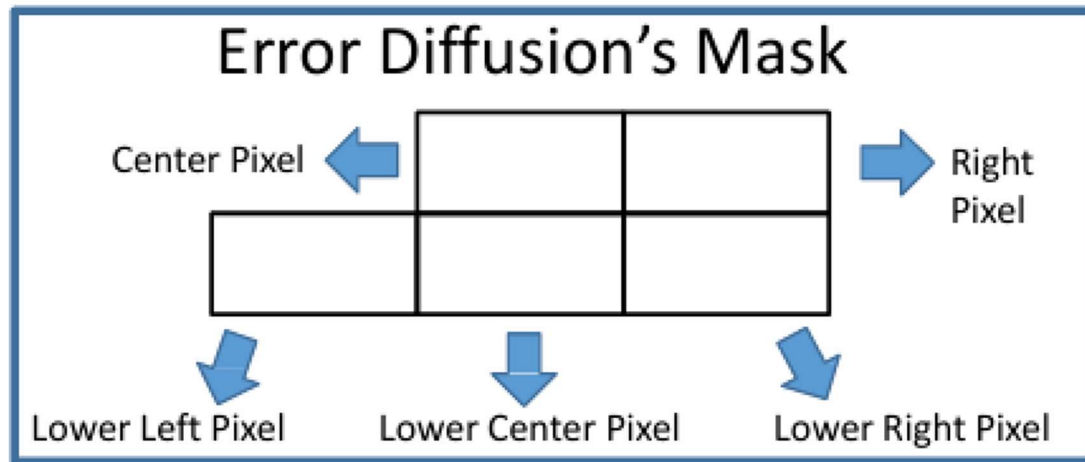
Làm tròn lên (round up): khi phần thập phân khác 0, kết quả sẽ được làm tròn lên số nguyên lớn hơn liền kề, ngược lại kết quả được giữ nguyên.

Làm tròn xuống (round down): khi phần thập phân khác 0, kết quả sẽ được làm tròn xuống số nguyên bé hơn liền kề, ngược lại kết quả được giữ nguyên.

Làm tròn đến số chẵn gần nhất (round to even): khi phần thập phân lớn hơn 0.5, kết quả được làm tròn lên số nguyên lớn hơn liền kề, khi phần thập phân nhỏ hơn 0.5, kết quả được làm tròn xuống số nguyên bé hơn liền kề, khi phần thập phân bằng 0.5, kết quả sẽ được làm tròn đến số chẵn gần nhất

2.2.2 Error Diffusion

Thuật toán error diffusion được sử dụng là Floyd Steinberg, việc phân tán lỗi từ một pixel trung tâm các pixel khác xung quanh nó được thực hiện bằng một mặt nạ:



Hình 2.19: Cấu tạo mặt nạ của giải thuật

Thuật toán Floyd Steinberg hoạt động bằng cách phân tán sai số làm tròn nhân với một ma trận hệ số của một pixel sang các pixel lân cận chưa được xử lý. Cụ thể, đối với mỗi pixel:

$$\text{Center pixel} = \begin{cases} 255 & \text{Nếu } \text{Center old pixel} \geq 128 \\ 0, & \text{Nếu } \text{Center old pixel} < 128 \end{cases}$$

$$\text{Error diffusion} = \text{Center old pixel} - \text{Center pixel}$$

$$\text{Right pixel} = \text{Right pixel data} + \frac{7}{16} \text{Error diffusion}$$

$$\text{Lower left pixel} = \text{Lower left pixel data} + \frac{3}{16} \text{Error diffusion}$$

$$\text{Lower pixel} = \text{Lower pixel data} + \frac{5}{16} \text{Error diffusion}$$

$$\text{Lower right pixel} = \text{Lower right pixel data} + \frac{1}{16} \text{Error diffusion}$$

Để thực hiện các phép tính trên, ta thực hiện các phép tính trên bằng phép dịch bit như ở Grayscale dưới dạng fixed point.

Và phải đảm bảo các giá trị pixel luôn nằm trong khoảng [0;255]. Ta dùng giải thuật như sau:

- Kiểm tra xem bit MSB của các pixel đó. Nếu âm (Clipping về 0): `result_r[12] && error[12]`
- Nếu bit MSB của pixel đó và bit [11:4] vượt quá 255: `(result_r[12] || (result_r[11:4] == 8'd255))`: clip về 255
- Nếu cờ `over_flow` của bit âm bị tràn: `&& !error[12]`: clip về 255