**ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH**

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN – ĐIỆN TỬ

**BỘ MÔN ĐIỆN TỬ**

---------------o0o---------------

****

**ĐỒ ÁN MÔN HỌC**

**THIẾT KẾ KHỐI NÉN ẢNH BẰNG GIẢI THUẬT DCT**

**GVHD: HOÀNG TRANG**

**SVTH: TRẦN HUỲNH XUÂN TRƯỜNG**

**MSSV: 1513795**

**TP. HỒ CHÍ MINH, THÁNG 06 NĂM 2018**

***LỜI CẢM ƠN***

***Em gửi lời cảm ơn sâu sắc đến thầy Hoàng Trang – GVHD đồ án môn học đã tận tình giảng dạy trong quá trình thực hiện đồ án môn học này. Em cảm ơn tất cả giảng viên giảng dạy các bộ môn đã truyền đạt cho em kiến thức để em thực hiện đồ án môn học này.***

*Tp. Hồ Chí Minh, ngày 05 tháng 06 năm 2018 .*

**Sinh viên**

**Trần Huỳnh Xuân Trường**

**TÓM TẮT ĐỒ ÁN**

Đồ án trình bày về thiết kế và mô phỏng khối nén ảnh sử dụng phép biến đổi Discrete Cosine Transform.

[1. GIỚI THIỆU 1](#_Toc516464642)

[1.1 Tổng quan 1](#_Toc516464643)

[1.2 Nhiệm vụ đề tài 1](#_Toc516464644)

[2. LÝ THUYẾT 1](#_Toc516464645)

[2.1 Miền màu: 1](#_Toc516464646)

[2.2 Kỹ thuật nén ảnh với phép biến đổi DCT trong JPEG: 2](#_Toc516464647)

[3. THIẾT KẾ HỆ THỐNG: 7](#_Toc516464648)

[3.1 Yêu cầu: 7](#_Toc516464649)

[3.1.1 Chức năng: 7](#_Toc516464650)

[3.1.2 I/O: 8](#_Toc516464651)

[3.2 Thiết kế hệ thống: 8](#_Toc516464652)

[3.3 Thực hiện hệ thống trên Matlab: 11](#_Toc516464653)

[3.4 Thiết kế RTL: 11](#_Toc516464654)

[3.5 Kiểm tra RTL, hệ thống: 11](#_Toc516464655)

[4. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 13](#_Toc516464656)

[4.1 Kết luận 13](#_Toc516464657)

[4.2 Hướng phát triển 13](#_Toc516464658)

[5. TÀI LIỆU THAM KHẢO 14](#_Toc516464659)

[6. PHỤ LỤC 14](#_Toc516464660)

[6.1 Code Matlab 14](#_Toc516464661)

[6.2 Sơ đồ RTL thiết kế 24](#_Toc516464662)

[6.3 Code Files: 26](#_Toc516464663)

[6.3.1 File image\_compression\_ip.v 26](#_Toc516464664)

[6.3.2 file dct\_2\_D.v 32](#_Toc516464665)

[6.3.3 file dct.v 35](#_Toc516464666)

DANH SÁCH HÌNH MINH HỌA

[Hình 1 :các I/O của khốiFigure 8](#_Toc516465180)

[Hình 2: sơ đồ các khối 9](#_Toc516465181)

[Hình 3: Kết quả sau mô phỏng a) ảnh tái tạo. b) ảnh gốc 12](#_Toc516465182)

[Hình 4: Kết quả sau mô phỏng a) ảnh tái tạo. b) ảnh gốc 12](#_Toc516465183)

# GIỚI THIỆU

## Tổng quan

Internet ngày nay đang dần trở nên quan trọng hơn với con người cho nhiều mục đích khác nhau. Lượng dữ liệu được tạo ra là rất lớn. Nhưng trong những hệ thống vừa và nhỏ thì việc lưu trữ và truyền dữ liệu lớn là một vấn đề. Chính vì vậy giải pháp nén dữ liệu trước khi truyền hoặc lưu đặc biệt hiệu quả trong những trường hợp này. Giới hạn của đề tài này chỉ xem xét việc nén ảnh để lưu trữ trong hệ thống.

## Nhiệm vụ đề tài

Thiết kế khối nén ảnh và đánh giá chất lượng ảnh sau nén qua những tiêu chí.

Nội dung 1: Tìm hiểu lý thuyết về nén ảnh với phép biến đổi DCT, những tiêu chí về đánh giá chất lượng ảnh sau hồi phục.

Nội dung 2: Mô phỏng hệ thống và các tiêu chí đánh giá về chất lượng ảnh sau nén trên Matlab.

Nội dung 3: Thiết kế RTL, mô phỏng và so sánh kết quả với hệ thống đã thiết kế bằng Matlab.

# LÝ THUYẾT

## Miền màu:

Ba màu cơ bản Red, Blue, Green (viết tắt lần lượt là R,G,B) hình thành ba trục tọa độ trong không gian màu mà mỗi màu là một điểm trong không gian đó. Một ảnh màu xám là ảnh mà chỉ sử dụng những điểm mà cả ba giá trị R, G, B đều bằng nhau. Trong hệ thống màu số, số lượng mức màu ở mỗi trục quy định bởi số lượng bits/pixel. Ví dụ: hệ thống RGB có 24 bpp tức có từ 0 – 255 mức màu cho mỗi hệ trục tọa độ.

Một không gian khác màu khác sử dụng những trục tọa độ thay thế là độ sáng (luminance), độ màu đỏ (red chrominance) và độ màu xanh (blue chrominance) hình thành nên không gian màu YCbCr. Công thức sau biểu thị sự chuyển đổi qua lại giữa không gian màu RGB và YCbCr trong hệ thống 24 bpp:

Chúng ta cần phải chuyển miền màu bởi vì mắt con người nhạy với độ sáng hơn độ màu. Thực tế cho thấy chuẩn nén JPEG đã tận dụng điều này để loại bỏ khoảng 3/4 thông tin về độ màu (chrominance) để nén, giúp giảm đáng kể dung lượng của ảnh.

## Kỹ thuật nén ảnh với phép biến đổi DCT trong JPEG:

Ảnh được biễu diễn bởi hàm trên mặt phẳng tọa độ tách rời 2 chiều . Mỗi mẫu trong không gian tọa độ tách rời 2-D được gọi là pixel. Mỗi pixel đó thường được lượng tử thành số nguyên không dấu để chỉ độ sáng trên mặt phẳng 2D. Với ảnh màu thì giá trị pixel ứng với vector trong không gian màu RGB .

Trong kỹ thuật nén ảnh, ta tìm một cách biễu diễn ảnh khác nhưng tốn ít tài nguyên hơn cách trên. Những kỹ thuật nén ảnh thường thỏa những tính chất sau:

* Tính tái tạo: Ảnh khi được biểu diễn dưới dạng nén có thể được biễu diễn ngược lại theo cách thông thường. Mặc dù quá trình này có thể làm cho ảnh sau khi tái tạo giống hoàn toàn hoặc gần giống với ảnh gốc.
* Ít dư thừa: Đối với ảnh thông thường những pixels gần nhau có mối tương quan về không gian điển hình như miền RGB. Chẳng hạn như dãy 10 các pixel có độ sáng từ trải từ 200 – 209 theo thang 0-255, thì chúng ta có thể thay thế dạng biểu diễn ảnh khác ít tốn tài nguyên hơn là cách hiển thị thông thường.
* Hệ số hóa thành các cấu trúc con: Việc phân tích đối tượng thành các thành phần hoặc cấu trúc con khác nhau của nó là một tính năng mong muốn khác của một kỹ thuật nén. Điều này trở nên đặc biệt hữu ích cho việc biểu diễn gần đúng hoặc mất dữ liệu của các đối tượng. Trong trường hợp này, các thành phần đóng góp không đáng kể vào việc xây dựng lại một đối tượng ở dạng ban đầu của nó, được cắt tỉa hoặc loại bỏ. Điều này làm giảm không gian lưu trữ của ảnh.

Một trong những kỹ thuật nén ảnh phổ biến đó là biễu diễn ảnh ở miền tần số bằng phép biến đổi Discrete Cosine Transform. Mục đích nhằm phân tích ảnh dưới dạng những tập hệ số để loại bỏ những thành phần tần số cao mà mắt con người không nhạy để tiết kiệm không gian lưu trữ. Miền màu RGB có thể được chuyển sang miền khác để những “giá trị pixels” trở nên ít tương quan hơn, đòi hỏi ít bit hơn để biểu diễn. Trong một số trường hợp, một thành phần màu sắc có thể được biểu diễn dưới dạng chức năng của các thành phần khác, yêu cầu ít dung lượng hơn để biểu thị các hàm hiển thị đó.

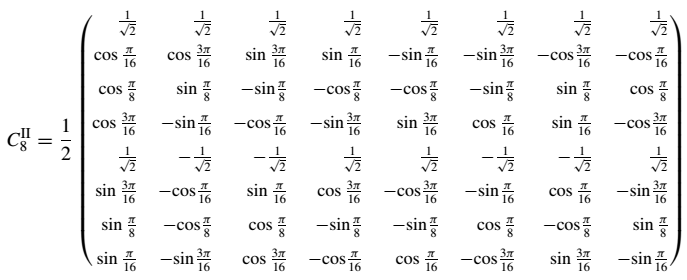
Trong chuẩn nén JPEG, ảnh được chia thành những khối 8x8 không chồng lên nhau và vì thế mỗi khối gần như xử lý độc lập. Sau đó mỗi giá trị miền màu trong khoảng 0- 255 đối với loại 8bpp. Sau đó, giá trị mỗi pixels được trừ cho 128 để để khoảng giá trị của ảnh thay đổi từ -128 – 127. Thực hiện biến đổi DCT 2-D []

Trong đó

Biến đổi ngược:

Từ hoàn toàn có thể được biến đổi trở lại bằng cách:

ma trận biến đổi DCT 1-D



Thực tế ta có thể biến đổi DCT 2-D dựa trên phép biến đổi DCT 1-D bằng việc biến đổi DCT 1-D lần lượt theo 2 chiều x, y.

Những hệ số trong ma trận trên là số thập phân, sẽ tốn rất nhiều tài nguyên để thiết kế. Trên thực tế, người ta sử dụng những phép xấp xỉ cùng với những biến đổi nhanh để tiết kiệm không gian thiết kế. Một trong những cách đó là xấp xỉ số nguyên [19, 22–24]

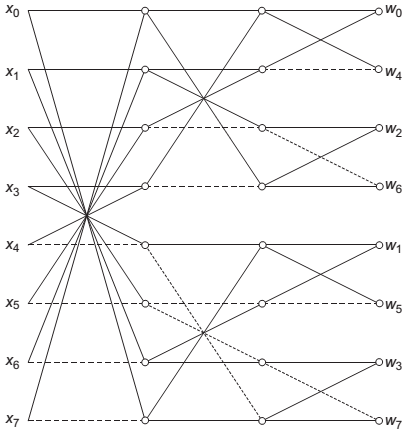
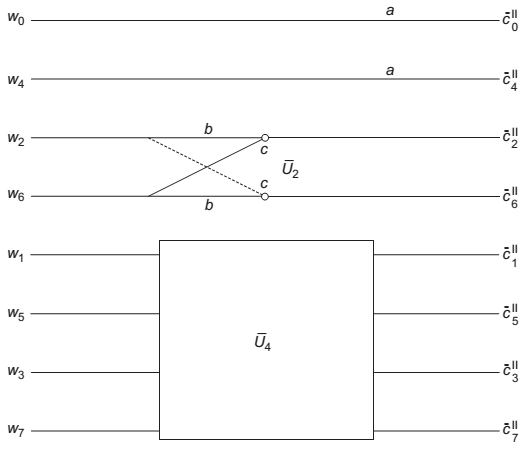
Ma trận biến đổi ở trên được phân tích thành tích của 2 ma trận với W và T là 2 ma trận 8x8

 và 

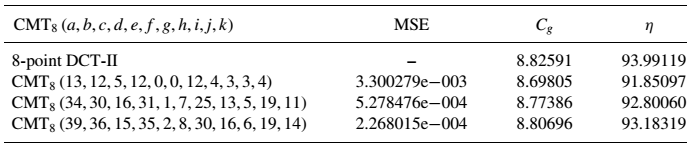


là ma trận T nhưng đổi chỗ hàng 2 với 5, hàng 7 với 4

Ta có sơ đồ tính nhanh của phép xấp xỉ này:



Các hệ số a, b, c, d, e, f, g, h, i, j được tính toán và so sánh sai số so với biến đổi gốc[]:

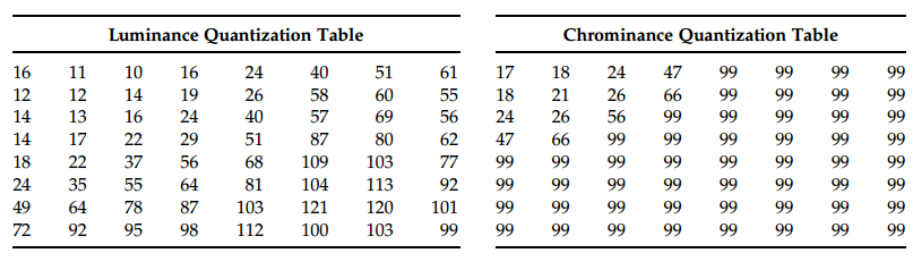


Các hệ số sau khi biến đổi DCT ở trên được lượng tử ở những mức khác nhau cho những thành phần tần số khác nhau. X(k,l) là hệ số của DCT thứ (k,l), và Q(k,l) là mức lượng tử tương ứng.

Trong đó round(x) là số nguyên gần nhất với x.

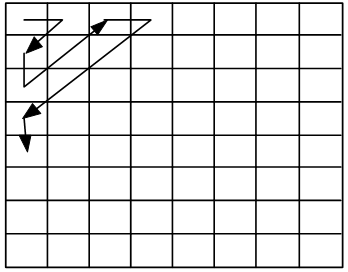
Ở quá trình ngược lại, hệ số DCT được xấp xỉ trở lại bằng việc nhân với Q(k,l)

Các hệ số Q(k,l) có thể do chúng ta tự quy định hoặc theo chuẩn JPEG theo bảng sau:



Bảng lượng tử chuẩn JPEG

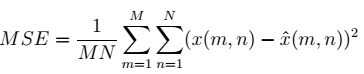
Quá trình lượng tử và giải lượng tử này làm cho phương pháp nén này xếp vào loại lossy. Tuy nhiên, quá trình lượng tử giúp làm giảm khoảng giá trị hệ số DCT 2D vì thế làm giảm số bit cần biểu diễn. Hơn nữa, nhiều hệ số DCT 2D sau lượng tử được xấp xỉ 0 nhất là những thành phần tần số cao, giúp chúng ta có thể biểu diễn chuỗi 0 này bằng những cách tiết kiệm bit hơn.

Quá trình mã hóa dữ liệu với mô hình Zig-zag để cho được chuỗi nhiều số 0 nhất và xuất ra output

xxx00000…. 🡪 xxx05….

Quá trình giải nén ngược lại các quá trình trên, ảnh thu được ta sẽ đánh giá theo những tiêu chí đối chiếu với ảnh chưa nén để đánh giá về hệ thống nén ảnh:

* Mean-squared error (MSE): MSE dựa trên sự bình phương khác nhau giữa ảnh nén và ảnh chưa nén.



* Peak signal-to-noise ratio (pSNR). pSNR chỉ ra tỉ lệ giữa độ sáng lớn nhất và công suất méo dạng, pSNR có giá trị càng cao thì ảnh càng tốt. (lớn hơn 25dB).



* Structural Similarity (SSIM). SSIM kết hợp cấu trúc hình ảnh cục bộ, độ chói và độ tương phản thành một điểm chất lượng cục bộ duy nhất. Trong chỉ số này, cấu trúc là các mẫu cường độ điểm ảnh, đặc biệt là giữa các pixel lân cận, sau khi chuẩn hóa độ sáng và độ tương phản. Vì hệ thống thị giác của con người có cấu trúc cảm nhận tốt, chỉ số chất lượng SSIM đồng ý chặt chẽ hơn với điểm chất lượng chủ quan.



# THIẾT KẾ HỆ THỐNG:

## Yêu cầu:

## Chức năng:

* Hệ thống nhận tín hiệu clock cạnh lên là tín hiệu đồng bộ, tín hiệu reset tích cực cao để khởi động hệ thống.
* Memory chứa dữ liệu theo cấu trúc từng pixel dưới dạng số thực 32 bits, theo thứ tự từ trái sang phải, từ trên xuống dưới của block 8x8 pixels, từng block từ trái sang phải, từ trên xuống dưới của ảnh. Memory tổ chức gồm 3 bộ nhớ tương ứng với 3 vùng màu Red, Green, Blue.
* Mỗi nhịp clock cạnh lên, hệ thống đọc 3 giá trị tương ứng 3 vùng màu ở Memory. Thực hiện quá trình nén xuất ra ouput 3 bus 8bits tương ứng với mỗi miền R, G, B. Dữ liệu nén xuất ra theo từng block, nếu phát hiện byte 0 thì byte liền sau nó chỉ số lượng các byte 0. Đồng thời có 1 tín hiệu đồng bộ xuất ra để báo hiệu bắt đầu xuất dữ liệu nén.
* Khối thực hiên giải nén ảnh rồi gửi ra output 3 bus 8bits là giá trị R, G, B theo thang 0 – 255. Các byte xuất theo lần lượt từng pixel từ trái sang phải, từ trên xuống dưới của từng block, từ trái sang phải, từ trên xuống dưới của ảnh.

Có 1 tín hiệu xuất ra để đồng bộ báo hiệu bắt đầu xuất dữ liệu giải nén.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

## I/O:

* Input: 1 bit clock, 1 bit reset tích cực cao, 1 Memory
* Output:
  + 3 bus 8bits tương ứng với 3 vùng màu Red, Green, Blue thể hiện giá trị pixel theo thang từ 0 – 255.
  + 3 bus 8bits thể hiện dữ liệu đã nén.
  + 2 tín hiệu đồng bộ: 1 cho ngõ ra 3 bus 8bits hồi phục và 1 cho ngõ ra 3bus 8bits dữ liệu nén.

|  |  |  |
| --- | --- | --- |
| Pin/bus | Loại | Chức năng |
| P1 | Input | Cấp clock. đồng bộ cạnh lên |
| P2 | Input | Reset tích cực mức cao |
| P3 | Output | Tín hiệu đồng bộ dữ liệu nén. Tích cực mức cao khi có dữ liệu xuất ra |
| P4 | Output | Tín hiệu đồng bộ dữ liệu giải nén. Tích cực mức cao khi có dữ liệu xuất ra |
| B1 | Input | Input mem R |
| B2 | Input | Input mem G |
| B3 | Input | Input mem B |
| B4 | Output | Dữ liệu nén miền Y |
| B5 | Output | Dữ liệu nén miền Cb |
| B6 | Output | Dữ liệu nén miền Cr |
| B7 | Output | Dữ liệu giải nén miền R |
| B8 | Output | Dữ liệu giải nén miền G |
| B9 | Output | Dữ liệu giải nén miền B |

8 bits

B4

B3

B2

B1

B5

P1

B6

Mem

P3

8 bits

B7

B8

B9

P2

P4

Hình :các I/O của khốiFigure

## Thiết kế hệ thống:

Sơ đồ các khối:

Mem RGB

to YCbCr

DCT-II

Quantization

Encoding

Normalize

Compressed Data

Decoding

Un-quantization

r-DCT

Un-Normalize

to RGB

Uncompressed Data

Hình : sơ đồ các khối

* Khối Mem RGB: dung lượng lưu trữ 4MB cho mỗi vùng màu R, G, B. Mỗi chu kỳ xung clock xuất ra 3 giá trị R,G,B lưu dưới đạng số thập phâp 32 bits.
* Khối chuyển sang YCbCr: Nhận 3 giá trị R, G, B từ khối Memory, khối chuyển vùng màu tính toán và đưa ra 3 giá trị của vùng màu Y, Cb, Cr dưới dạng số thập phân 32 bits.
  + Input: 3 số 32 bits.
  + Ouput: 3 số 32 bits.
* Khối Normalization: Khối thực hiện chuẩn hóa dữ liệu. Mỗi chu kỳ clock, khối lấy mỗi giá trị mỗi vùng Y, Cb, Cr trừ cho 128. Ngõ ra là số nguyên có dấu 32 bits. Thực tế khối này ghép vào trong khối chuyển vùng màu.
  + Input: 3 số 32 bits.
  + Ouput: 3 số 32 bits.
* Khối DCT: Khối DCT thực hiện tính toán DCT 2D. Mỗi chu kỳ clock lấy 3 giá trị ở khối chuẩn hóa làm input. Khối này tính toán và xuất ra kết quả trong mỗi chu kỳ clock dưới dạng số floating-point 32 bits sang 3 output tương ứng 3 vùng Y, Cb, Cr. Khối có delay ngõ vào và ra nên cần 1 tín hiệu đồng bộ báo bắt đầu xuất output tích cực mức cao.
  + Input: 3 số 32 bits, clock, reset
  + Ouput: 3 số 32 bits, 1 tín hiệu đồng bộ.
* Khối Quantization: Mỗi clock, lấy giá trị 3 ngõ ra khối DCT lượng tử giá trị này theo bảng lượng tử JPEG đã lưu trong những thanh ghi. Làm tròn những số sau lượng tử này thành số nguyên gần nhất rồi xuất ra số 8 bits có dấu ở mỗi chu kỳ xung clock.
  + Input: 3 số 32 bits, clock, reset.
  + Output: 3 số 8 bits.
* Khối Un-quantization: Mỗi clock lấy mỗi output của khối decoding nhân cho giá trị trong bảng lượng tử lưu sẵn trong những thanh ghi. Output là số floating-point 32 bits
  + Input: 3 số 8 bits, clock, reset.
  + Output: 3 số 32bits bits.
* Khối r-DCT: Khối r-DCT thực hiện biến đổi ngược DCT 2D. Mỗi chu kỳ clock lấy 3 giá trị ở khối Un-quantization làm input. Khối này tính toán và xuất ra kết quả trong mỗi chu kỳ clock dưới dạng số floating-point 32 bits sang 3 output tương ứng 3 vùng Y, Cb, Cr. Khối có delay ngõ vào và ra nên cần 1 tín hiệu đồng bộ báo bắt đầu xuất output tích cực mức cao.
  + Input: 3 số 32 bits, clock, reset.
  + Output: 3 số 32 bits, 1 tín hiệu đồng bộ.
* Khối Un-normalize: cộng mỗi giá trị output của khối r-DCT với 128.
  + Input: 3 số 32 bits.
  + Output: 3 số 32 bits.

Khối chuyển sang RGB: Lấy ngõ ra của khối thực hiện chuyển vùng màu YCbCr sang RGB.

* + Input: 3 số 32 bits.
  + Output: 3 số 8 bits.

## Thực hiện hệ thống trên Matlab:

Thực hiện hệ thống trên matlab để kiểm tra các thiết kế hệ thống, cùng các tiêu chí. Đồng thời, lấy những kết quả tính toán được bằng matlab làm cơ sở so sánh thiết kế RTL sau này.

Code matlab phần [phụ lục 6.1](#_Code_Matlab)

## Thiết kế RTL:

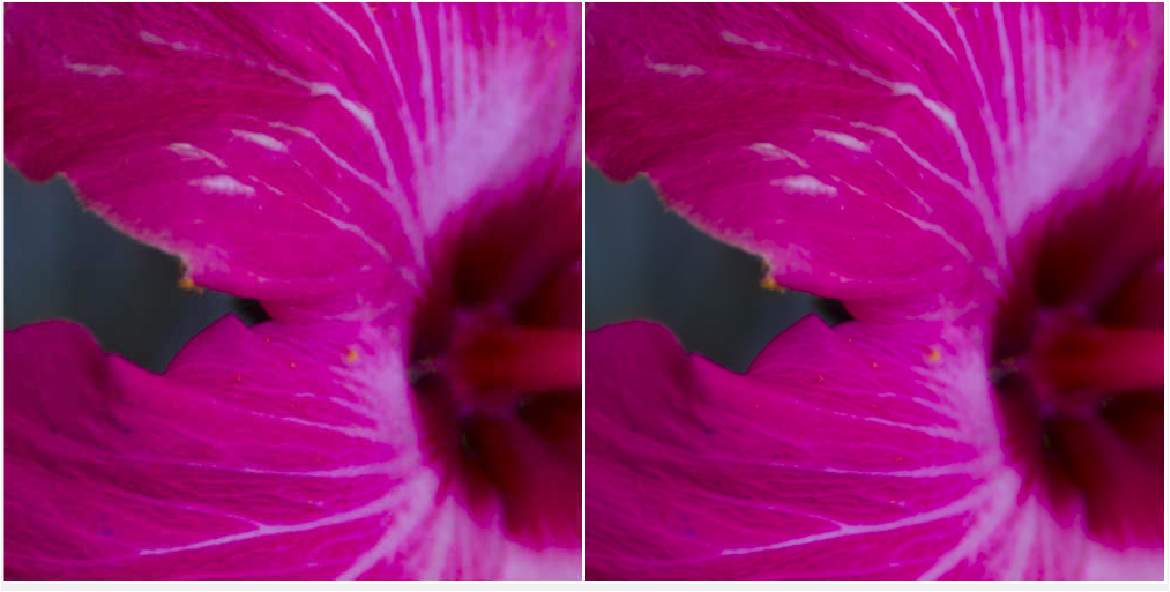
Sơ đồ thiết kế [phụ lục 6.2](#_Sơ_đồ_RTL)

Thiết kế bộ floating point phụ lục 6.3:

## Kiểm tra RTL, hệ thống:

* Kiểm tra thiết kế hệ thống bằng matlab:

Ảnh 1:



1. b)

Hình : Kết quả sau mô phỏng a) ảnh tái tạo. b) ảnh gốc

|  |  |
| --- | --- |
| pSNR | SSIM |
| 36.9296 | 0.9964 |

Ảnh 2:



1. b)

Hình : Kết quả sau mô phỏng a) ảnh tái tạo. b) ảnh gốc

|  |  |
| --- | --- |
| pSNR | SSIM |
| 34.12167 | 0.9679 |

Nhận xét: Những đường nét của ảnh tái tạo mờ đi so với ảnh gốc, các chỉ số trên sẽ càng cao đối với ảnh không có nhiều sự thay đổi màu sắc nhiều.

* Kiểm tra sự khác nhau giữa thiết kế bằng matlab và thiết kế RTL.

Một đoạn code matlab nhỏ để chạy so sánh dựa vào những file kết quả matlab và kêt quả chạy mô phỏng thiết kế:

clear; clc;

string = '%f';

file = fopen('A:\Truong\project\_term\_172\05\_Sim\_Log\result\_G.txt','r');

file\_pc = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\r\_G\_file.v','r');

data = fscanf(file, '%s', [168 1]);

data = fscanf(file, string);

data\_pc = fscanf(file\_pc, string);

[x y] = size(data);

for i=1:x

if(abs(data(i,1) - data\_pc(i,1)) > 0.1)

fprintf('=======\n %d ==> %f\n', i, data(i,1) - data\_pc(i,1));

end

end

Kết quả của sự so sánh này, ta thấy vẫn có sự sai khác +-1 do sai số khối làm tròn từ thực sang số nguyên

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết luận

Khối nén ảnh được thiết kế đảm bảo chất lượng ảnh sau nén (SSIM > 0.90), kết quả thiết kế RTL gần đúng so với thiết kế bằng matlab (sai số không nhiều). Tuy nhiên, đề tài chưa giới thiệu cách thức nén cũng như tỉ lệ nén của hệ thống.

## Hướng phát triển

Hệ thống này vẫn chưa tối ưu số clock thực thi, có thể giảm độ trễ ngõ vào, ra bằng việc thay thế những thuật toán bộ nhân high radix, bộ cộng cla, sử dụng kỹ thuật unfolding để giảm số clock thực thi cho khối chuyển vị ma trận. Bộ encoding và decoding nên được thiết kế bằng khối mã hóa đường truyền Huff-man sẽ cho kết quả nén tốt nhất.

# TÀI LIỆU THAM KHẢO

1. “Image and video compression for multimedia engineering\_ fundamentals, algorithms, and standards” (Image Processing Series) Yun Q. Shi, Huifang Sun CRC Press (2008)
2. “Discrete cosine and sine transforms general properties, fast algorithms and integer approximations” Vladimir Britanak, Patrick C. Yip, K. R Rao Academic (2007)
3. “Image and Video Compression\_ Fundamentals, Techniques, and Applications” Madhuri A. Joshi, Mehul S. Raval, Yogesh H. Dandawate, Kalyani R. Joshi, Shilpa P. Metkar Chapman and Hall\_CRC (2014)
4. “Image and Video Processing in the Compressed Domain”- Jayanta Mukhopadhyay Chapman and Hall \_CRC (2011)

# PHỤ LỤC

## Code Matlab

function C02\_System\_with\_compression();

clear variables; clear global;

close all force;

clc;

%----------------------------------------------------------------------------------------------------

%----- list all images

cd '.\rgb8bit\';

ls;

cd ..;

%----- choose an image and the part of image to compress

prompt = 'ten anh can nen\n';

str = input(prompt,'s');

str = strcat('.\rgb8bit\',str);

prompt = 'vi tri x:\n';

x = input(prompt);

prompt = 'vi tri y:\n';

y = input(prompt);

prompt = 'rong :\n';

width = input(prompt);

prompt = 'cao :\n';

height = input(prompt);

RGB = cutImage(x, y, width, height, str);

%----- get the size of image

RGB = double(RGB);

[M, N, O] = size(RGB);

up\_r = ceil(M/8);

up\_c = ceil(N/8);

%----- write the pixels value to files so that RTL system read files for its inputs

R\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\read\_R.v','w');

G\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\read\_G.v','w');

B\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\read\_B.v','w');

for y\_block = 1:up\_r

for x\_block = 1:up\_c

for r = (y\_block-1)\*8+1 : y\_block\*8

for c = (x\_block-1)\*8+1 : x\_block\*8

if c <= N && r <= M

fprintf(R\_file,'%s\n', num2hex(single(RGB(r,c,1))));

fprintf(G\_file,'%s\n', num2hex(single(RGB(r,c,2))));

fprintf(B\_file,'%s\n', num2hex(single(RGB(r,c,3))));

else

fprintf(R\_file,'%s\n', num2hex(single(255.0)));

fprintf(G\_file,'%s\n', num2hex(single(255.0)));

fprintf(B\_file,'%s\n', num2hex(single(255.0)));

end

end

end

end

end

%----- add white pixels if the image does not have suitable size

for y\_block = 1:up\_r

for x\_block = 1:up\_c

for r = (y\_block-1)\*8+1 : y\_block\*8

for c = (x\_block-1)\*8+1 : x\_block\*8

if r > M || c > N

RGB(r,c,1) = 255;

RGB(r,c,2) = 255;

RGB(r,c,3) = 255;

end

end

end

end

end

%----------------------------------------------------------------------------------------------------

%----- convert RGB space into YCbCr space

YCbCr(:,:,1) = 0.299\*RGB(:,:,1) + 0.587\*RGB(:,:,2) + 0.114\*RGB(:,:,3);

YCbCr(:,:,2) = -0.169\*RGB(:,:,1) - 0.331\*RGB(:,:,2) + 0.500\*RGB(:,:,3) + 128;

YCbCr(:,:,3) = 0.500\*RGB(:,:,1) - 0.419\*RGB(:,:,2) - 0.081\*RGB(:,:,3) + 128;

%----- write the result of conversion to files to check the one of RTL system

in\_Y\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\in\_Y.v','w');

in\_Cb\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\in\_Cb.v','w');

in\_Cr\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\in\_Cr.v','w');

for y\_block = 1:up\_r

for x\_block = 1:up\_c

for r = (y\_block-1)\*8+1 : y\_block\*8

for c = (x\_block-1)\*8+1 : x\_block\*8

fprintf(in\_Y\_file,'%f\n', YCbCr(r,c,1));

fprintf(in\_Cb\_file,'%f\n', YCbCr(r,c,2));

fprintf(in\_Cr\_file,'%f\n', YCbCr(r,c,3));

end

end

end

end

%----------------------------------------------------------------------------------------------------

%----- normalize matrix, subtract 128 from YCbCr matrix

normalization = YCbCr - 128;

Y = round(normalization(:,:,1));

Cb = round(normalization(:,:,2));

Cr = round(normalization(:,:,3));

%----------------------------------------------------------------------------------------------------

%-----DCT 2-D transform

%D = dctmtx(8); % uncomment to use a built-in 8x8 DCT matrix

a=39; b=36; c=15; d=35; e=2; f=8; g=30; h=16; y=6; j=19; k=14;

W = [

1 1 1 1 1 1 1 1;

1 -1 -1 1 1 -1 -1 1;

1 1 -1 -1 -1 -1 1 1;

1 -1 1 -1 -1 1 -1 1;

1 1 1 1 -1 -1 -1 -1;

1 -1 -1 1 -1 1 1 -1;

1 1 -1 -1 1 1 -1 -1;

1 -1 1 -1 1 -1 1 -1 ];

T = [

a 0 0 0 0 0 0 0;

0 a 0 0 0 0 0 0;

0 0 b c 0 0 0 0;

0 0 -c b 0 0 0 0;

0 0 0 0 d -e h y;

0 0 0 0 f g -j k;

0 0 0 0 -k j g f;

0 0 0 0 -y -h -e d ];

cv = T(2,:);

T(2,:) = T(5,:);

T(5,:) = cv;

cv = T(7,:);

T(7,:) = T(4,:);

T(4,:) = cv;

D = T\*W;

% D = [

% 39 39 39 39 39 39 39 39;

% 55 47 27 11 -11 -27 -47 -55;

% 51 21 -21 -51 -51 -21 21 51;

% 43 -11 -55 -33 33 55 11 -43;

% 39 -39 -39 39 39 -39 -39 39;

% 33 -55 11 43 -43 -11 55 -33;

% 21 -51 51 -21 -21 51 -51 21;

% 11 -27 47 -55 55 -47 27 -11];

D\_T = (T\*W)';

for i=0:up\_r-1

for j=0:up\_c-1

temp = Y(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8);

Y\_o(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8) = ((D\*(D\*temp)')') /8/39/39;

temp = Cb(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8);

Cb\_o(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8) = ((D\*(D\*temp)')') /8/39/39;

temp = Cr(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8);

Cr\_o(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8) = ((D\*(D\*temp)')') /8/39/39;

end

end

dct\_transform(:,:,1) = Y\_o;

dct\_transform(:,:,2) = Cb\_o;

dct\_transform(:,:,3) = Cr\_o;

%----- write the result of transformation to files to check the one of RTL system

dct\_Y\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\dct\_Y.v','w');

dct\_Cb\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\dct\_Cb.v','w');

dct\_Cr\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\dct\_Cr.v','w');

for y\_block = 1:up\_r

for x\_block = 1:up\_c

for c = (x\_block-1)\*8+1 : x\_block\*8

for r = (y\_block-1)\*8+1 : y\_block\*8

fprintf(dct\_Y\_file,'%f\n', dct\_transform(r,c,1));

fprintf(dct\_Cb\_file,'%f\n', dct\_transform(r,c,2));

fprintf(dct\_Cr\_file,'%f\n', dct\_transform(r,c,3));

end

end

end

end

%----------------------------------------------------------------------------------------------------

%----- quantize

Y\_quantization = [

16 11 10 16 24 40 51 61;

12 12 14 19 26 58 60 55;

14 13 16 24 40 57 69 56;

14 17 22 29 51 87 80 62;

18 22 37 56 68 109 103 77;

24 35 55 64 81 104 113 92;

49 64 78 87 103 121 120 101;

72 92 95 98 112 100 103 99 ];

Chrominance\_quantization = [

17 18 24 47 99 99 99 99;

18 21 26 66 99 99 99 99;

24 26 56 99 99 99 99 99;

47 99 99 99 99 99 99 99;

99 99 99 99 99 99 99 99;

99 99 99 99 99 99 99 99;

99 99 99 99 99 99 99 99;

99 99 99 99 99 99 99 99 ];

for i=0:up\_r-1

for j=0:up\_c-1

quantized\_Y(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8) = round(Y\_o(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8)./Y\_quantization);

quantized\_Cb(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8) = round(Cb\_o(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8)./Chrominance\_quantization);

quantized\_Cr(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8) = round(Cr\_o(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8)./Chrominance\_quantization);

end

end

%----- write the result of quantization to files to check the one of RTL system

qun\_Y\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\qun\_Y.v','w');

qun\_Cb\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\qun\_Cb.v','w');

qun\_Cr\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\qun\_Cr.v','w');

for y\_block = 1:up\_r

for x\_block = 1:up\_c

for c = (x\_block-1)\*8+1 : x\_block\*8

for r = (y\_block-1)\*8+1 : y\_block\*8

fprintf(qun\_Y\_file,'%f\n' , (quantized\_Y(r,c)));

fprintf(qun\_Cb\_file,'%f\n', (quantized\_Cb(r,c)));

fprintf(qun\_Cr\_file,'%f\n', (quantized\_Cr(r,c)));

end

end

end

end

%----------------------------------------------------------------------------------------------------

%----- un-quantize

for i=0:up\_r-1

for j=0:up\_c-1

un\_quantized\_Y(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8) = quantized\_Y(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8).\* Y\_quantization;

un\_quantized\_Cb(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8) = quantized\_Cb(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8) .\* Chrominance\_quantization;

un\_quantized\_Cr(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8) = quantized\_Cr(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8) .\* Chrominance\_quantization;

end

end

%----- write the result of un-quantization to files to check the one of RTL system

unqun\_Y\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\unqun\_Y.v','w');

unqun\_Cb\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\unqun\_Cb.v','w');

unqun\_Cr\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\unqun\_Cr.v','w');

for y\_block = 1:up\_r

for x\_block = 1:up\_c

for c = (x\_block-1)\*8+1 : x\_block\*8

for r = (y\_block-1)\*8+1 : y\_block\*8

fprintf(unqun\_Y\_file,'%f\n' , (un\_quantized\_Y(r,c)));

fprintf(unqun\_Cb\_file,'%f\n', (un\_quantized\_Cb(r,c)));

fprintf(unqun\_Cr\_file,'%f\n', (un\_quantized\_Cr(r,c)));

end

end

end

end

%----------------------------------------------------------------------------------------------------

%-----inverse DCT 2-D transform

for i=0:up\_r-1

for j=0:up\_c-1

temp = un\_quantized\_Y(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8);

Y\_i(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8) = ((D\_T\*(D\_T\*temp)')') /8/39/39;

temp = un\_quantized\_Cb(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8);

Cb\_i(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8) = ((D\_T\*(D\_T\*temp)')') /8/39/39;

temp = un\_quantized\_Cr(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8);

Cr\_i(i\*8+1:(i+1)\*8, j\*8+1:(j+1)\*8) = ((D\_T\*(D\_T\*temp)')') /8/39/39;

end

end

inverse\_dct\_transform(:,:,1) = Y\_i;

inverse\_dct\_transform(:,:,2) = Cb\_i;

inverse\_dct\_transform(:,:,3) = Cr\_i;

%----- write the result of r-DCT 2-D to files to check the one of RTL system

r\_dct\_Y\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\r\_dct\_Y\_file.v','w');

r\_dct\_Cb\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\r\_dct\_Cb\_file.v','w');

r\_dct\_Cr\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\r\_dct\_Cr\_file.v','w');

for y\_block = 1:up\_r

for x\_block = 1:up\_c

for r = (y\_block-1)\*8+1 : y\_block\*8

for c = (x\_block-1)\*8+1 : x\_block\*8

fprintf(r\_dct\_Y\_file,'%f\n' , (Y\_i(r,c)));

fprintf(r\_dct\_Cb\_file,'%f\n', (Cb\_i(r,c)));

fprintf(r\_dct\_Cr\_file,'%f\n', (Cr\_i(r,c)));

end

end

end

end

%----------------------------------------------------------------------------------------------------

%-----to un-normalize

un\_normalization = inverse\_dct\_transform + 128;

%----------------------------------------------------------------------------------------------------

%-----convert YCbCr space to RGB space

rgb\_matrix(:,:,1) = round(1\*un\_normalization(:,:,1)-0.000\*(un\_normalization(:,:,2)-128)+1.400\*(un\_normalization(:,:,3)-128));

rgb\_matrix(:,:,2) = round(1\*un\_normalization(:,:,1)-0.343\*(un\_normalization(:,:,2)-128)-0.711\*(un\_normalization(:,:,3)-128));

rgb\_matrix(:,:,3) = round(1\*un\_normalization(:,:,1)+1.765\*(un\_normalization(:,:,2)-128)+0.000\*(un\_normalization(:,:,3)-128));

%----- write the result to file

r\_R\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\r\_R\_file.v','w');

r\_G\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\r\_G\_file.v','w');

r\_B\_file = fopen('A:\Truong\project\_term\_172\02\_Simulate\test\_data\r\_B\_file.v','w');

for y\_block = 1:up\_c

for x\_block = 1:up\_r

for i = (x\_block-1)\*8+1 : x\_block\*8

for j = (y\_block-1)\*8+1 : y\_block\*8

% R

if (rgb\_matrix(i,j,1) > 255)

fprintf(r\_R\_file,'%d\n', 255);

else if (rgb\_matrix(i,j,1) < 0)

fprintf(r\_R\_file,'%d\n', 0);

else

fprintf(r\_R\_file,'%d\n', rgb\_matrix(i,j,1));

end

end

% G

if (rgb\_matrix(i,j,2) > 255)

fprintf(r\_G\_file,'%d\n', 255);

else if (rgb\_matrix(i,j,2) < 0)

fprintf(r\_G\_file,'%d\n', 0);

else

fprintf(r\_G\_file,'%d\n', rgb\_matrix(i,j,2));

end

end

% B

if (rgb\_matrix(i,j,3) > 255)

fprintf(r\_B\_file,'%d\n', 255);

else if (rgb\_matrix(i,j,3) < 0)

fprintf(r\_B\_file,'%d\n', 0);

else

fprintf(r\_B\_file,'%d\n', rgb\_matrix(i,j,3));

end

end

end

end

end

end

%----------------------------------------------------------------------------------------------------

%-----display images

im\_1 = figure;

im\_1.Name = 'Recreated Image';

rgb\_matrix=uint8(rgb\_matrix);

imshow(rgb\_matrix);

im\_2 = figure;

im\_2.Name = 'Original Image';

RGB=uint8(RGB);

imshow(RGB);

%----------------------------------------------------------------------------------------------------

%----------------------------------------------------------------------------------------------------

%-----mesure some metrics

%=====1. MD metric

format long;

r\_m=0.0; g\_m=0.0; b\_m=0.0;

R=RGB(:,:,1);

G=RGB(:,:,2);

B=RGB(:,:,3);

r\_m = max(max(abs(rgb\_matrix(:,:,1) - R)));

g\_m = max(max(abs(rgb\_matrix(:,:,2) - G)));

b\_m = max(max(abs(rgb\_matrix(:,:,3) - B)));

fprintf('\n================\nMD metric\n');

fprintf('R space: %f\nG space: %f\nB space: %f\n',r\_m,g\_m,b\_m);

%----------------------------------------------------------------------------------------------------

%=====2. MSE metric

fprintf('\n================\nMSE metric\n');

% MSE\_R = 0.0; MSE\_G = 0.0; MSE\_B = 0.0;

% r=rgb\_matrix(:,:,1);

% g=rgb\_matrix(:,:,2);

% b=rgb\_matrix(:,:,3);

%

% for i=1:M

% for j=1:N

% MSE\_R = MSE\_R + (double(r(i,j)) - double(R(i,j)))^2;

% MSE\_G = MSE\_G + (double(g(i,j)) - double(G(i,j)))^2;

% MSE\_B = MSE\_B + (double(b(i,j)) - double(B(i,j)))^2;

% end

% end

% MSE\_R = double(MSE\_R/M/N);

% MSE\_G = double(MSE\_G/M/N);

% MSE\_B = double(MSE\_B/M/N);

% fprintf('MSE Red space metric: %f\nMSE Green space metric: %f\nMSE Blue space metric: %f\n',MSE\_R,MSE\_G,MSE\_B);

MSE = immse(rgb\_matrix,RGB)

%----------------------------------------------------------------------------------------------------

%-----2. PSNR metric

fprintf('\n================\nPSNR metric\n');

% PSNR\_R = 0.0; PSNR\_G = 0.0; PSNR\_B = 0.0;

%

% PSNR\_R = 10\*log10((M^2/MSE\_R));

% PSNR\_G = 10\*log10((M^2/MSE\_G));

% PSNR\_B = 10\*log10((M^2/MSE\_B));

%

% fprintf('PSNR Red space metric: %f\nPSNR Green space metric: %f\nPSNR Blue space metric: %f\n',PSNR\_R,PSNR\_G,PSNR\_B);

pSNR = psnr(rgb\_matrix,RGB)

%----------------------------------------------------------------------------------------------------

%-----2. SSIM metric

fprintf('\n================\nSSIM metric\n');

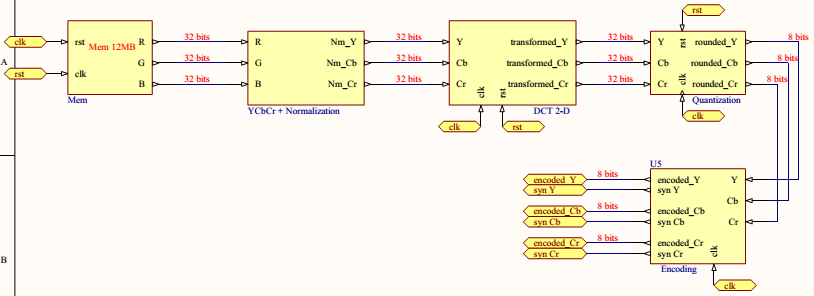
SSIM = ssim(rgb\_matrix,RGB)

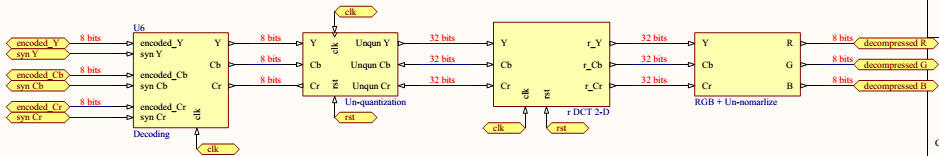
fclose('all');

end

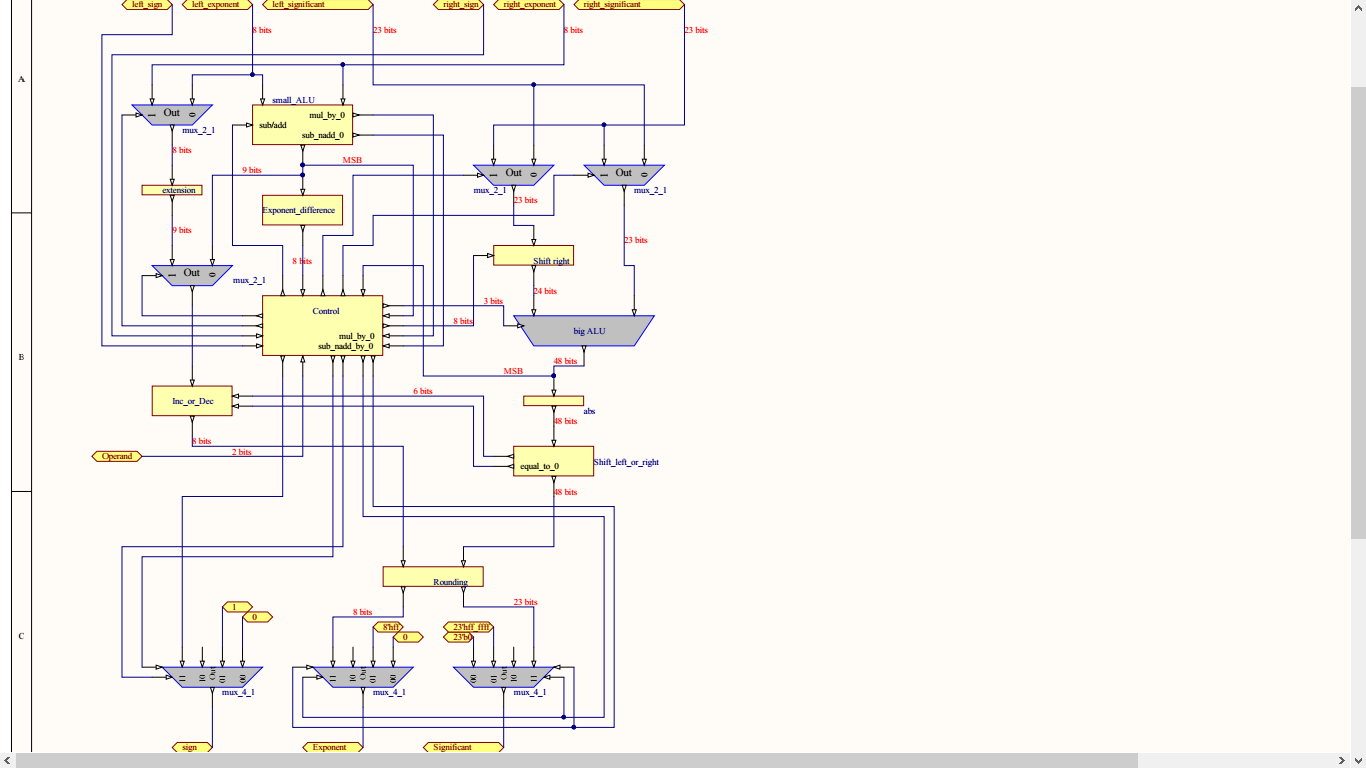
## Sơ đồ RTL thiết kế

Sơ đồ thiết kế





Sơ đồ thiết kế floating point:



## Code Files:

## File image\_compression\_ip.v

// Author: Truong Tran

// Date: 2018/05/10

//====================================================================================================//

//====================================================================================================//

`timescale 1ps/1ps

module image\_compression\_ip (R\_out, G\_out, B\_out, rst, clk);

output [7:0] R\_out, G\_out, B\_out;

input clk, rst;

wire mem\_start;

wire finish\_64, finish, enable, dct\_2\_D\_rst, reset\_quatization, reset\_unquatization, r\_dct\_2\_D\_rst;

wire [24:0] dct\_2D\_Y\_int, dct\_2D\_Cb\_int, dct\_2D\_Cr\_int, Y\_int, Cb\_int, Cr\_int;

wire [31:0] R\_O, G\_O, B\_O, Y\_in, Cb\_in, Cr\_in, Y\_O, Cb\_O, Cr\_O;

wire [31:0] float\_Y, float\_Cb, float\_Cr, dct\_2D\_Y\_float, dct\_2D\_Cb\_float, dct\_2D\_Cr\_float;

wire [31:0] quantized\_Y, quantized\_Cb, quantized\_Cr;

wire [24:0] rounded\_Y\_int, rounded\_Cb\_int, rounded\_Cr\_int;

wire [31:0] rounded\_Y\_float, rounded\_Cb\_float, rounded\_Cr\_float;

wire [31:0] unquantized\_Y, unquantized\_Cb, unquantized\_Cr, r\_Y, r\_Cb, r\_Cr;

wire [31:0] mem\_Y\_O, mem\_Cb\_O, mem\_Cr\_O;

wire [31:0] r\_dct\_2D\_Y\_float, r\_dct\_2D\_Cb\_float, r\_dct\_2D\_Cr\_float;

wire [31:0] R\_result\_float, G\_result\_float, B\_result\_float;

wire [24:0] R\_result\_int, G\_result\_int, B\_result\_int;

wire [7:0] scaled\_R, scaled\_G, scaled\_B;

wire [7:0] Y\_coding, Cb\_coding, Cr\_coding, decode\_Y, decode\_Cb, decode\_Cr;

wire reset\_coding, sync\_clk\_Y, sync\_clk\_Cb, sync\_clk\_Cr;

//----------------------------------------------------------------------------------------------------//

//----- change colour spaces

mem\_RGB RGB\_mem (.R\_O(R\_O), .G\_O(G\_O), .B\_O(B\_O), .finish\_64(finish\_64), .finish(finish), .size\_x(32'd512), .size\_y(32'd512), .en\_read(1'b1), .clk(clk));

convert\_color\_space\_Y convert\_into\_Y\_space (.Y(Y\_in) , .R(R\_O), .G(G\_O), .B(B\_O));

convert\_color\_space\_Cb convert\_into\_Cb\_space (.Cb(Cb\_in), .R(R\_O), .G(G\_O), .B(B\_O));

convert\_color\_space\_Cr convert\_into\_Cr\_space (.Cr(Cr\_in), .R(R\_O), .G(G\_O), .B(B\_O));

control\_convertion control\_mem\_YCbCr (.en\_mem\_YCbCr(enable), .clk(clk));

mem\_YCbCr YCbCr\_mem (.Y\_O(Y\_O), .Cb\_O(Cb\_O), .Cr\_O(Cr\_O), .Y\_in(Y\_in), .Cb\_in(Cb\_in), .Cr\_in(Cr\_in), .en\_write(1'b1), .en\_read(1'b1), .enable(enable), .clk(clk));

//----------------------------------------------------------------------------------------------------//

//----- calculate DCT\_2D

convert\_float\_into\_int float\_2\_int\_Y\_00 (Y\_int, Y\_O);

convert\_float\_into\_int float\_2\_int\_Cb\_00 (Cb\_int, Cb\_O);

convert\_float\_into\_int float\_2\_int\_Cr\_00 (Cr\_int, Cr\_O);

control\_DCT control\_dct (.dct\_2\_D\_reset(dct\_2\_D\_rst), .clk(clk));

dct\_2\_D dct\_2\_D\_Y (.O(dct\_2D\_Y\_int) , .in(Y\_int) , .clk(clk), .rst(dct\_2\_D\_rst));

dct\_2\_D dct\_2\_D\_Cb (.O(dct\_2D\_Cb\_int), .in(Cb\_int), .clk(clk), .rst(dct\_2\_D\_rst));

dct\_2\_D dct\_2\_D\_Cr (.O(dct\_2D\_Cr\_int), .in(Cr\_int), .clk(clk), .rst(dct\_2\_D\_rst));

convert\_int\_into\_float int\_2\_float\_Y\_00 (.O(float\_Y) , .in(dct\_2D\_Y\_int));

convert\_int\_into\_float int\_2\_float\_Cb\_00 (.O(float\_Cb) , .in(dct\_2D\_Cb\_int));

convert\_int\_into\_float int\_2\_float\_Cr\_00 (.O(float\_Cr) , .in(dct\_2D\_Cr\_int));

//----- divide by 8\*39\*39 --> finish calculating DCT\_2D

FP fp\_Y (.O(dct\_2D\_Y\_float) , .in0(float\_Y) , .in1(32'h38AC598B), .operand(2'h2));

FP fp\_Cb (.O(dct\_2D\_Cb\_float), .in0(float\_Cb), .in1(32'h38AC598B), .operand(2'h2));

FP fp\_Cr (.O(dct\_2D\_Cr\_float), .in0(float\_Cr), .in1(32'h38AC598B), .operand(2'h2));

//----------------------------------------------------------------------------------------------------//

//----- Quantization

control\_quatization ctrl\_quatization (.reset\_quatization(reset\_quatization), .clk(clk), .rst(rst));

Y\_quantization Y\_table (.quantized\_Y(quantized\_Y), .Y\_in(dct\_2D\_Y\_float), .clk(clk), .rst(reset\_quatization));

Chrominance\_quantization Cb\_table (.quantized\_Cb\_Cr(quantized\_Cb), .Cb\_Cr\_in(dct\_2D\_Cb\_float), .clk(clk), .rst(reset\_quatization));

Chrominance\_quantization Cr\_table (.quantized\_Cb\_Cr(quantized\_Cr), .Cb\_Cr\_in(dct\_2D\_Cr\_float), .clk(clk), .rst(reset\_quatization));

//----------------------------------------------------------------------------------------------------//

//----- Rounding

convert\_float\_into\_int float\_2\_int\_Y\_01 (rounded\_Y\_int, quantized\_Y);

convert\_float\_into\_int float\_2\_int\_Cb\_01 (rounded\_Cb\_int, quantized\_Cb);

convert\_float\_into\_int float\_2\_int\_Cr\_01 (rounded\_Cr\_int, quantized\_Cr);

convert\_int\_into\_float int\_2\_float\_Y\_01 (.O(rounded\_Y\_float) , .in(rounded\_Y\_int));

convert\_int\_into\_float int\_2\_float\_Cb\_01 (.O(rounded\_Cb\_float) , .in(rounded\_Cb\_int));

convert\_int\_into\_float int\_2\_float\_Cr\_01 (.O(rounded\_Cr\_float) , .in(rounded\_Cr\_int));

//----------------------------------------------------------------------------------------------------//

//----- Line Coding

reset\_code r (.O(reset\_coding), .clk(clk), .rst(rst));

coding code\_Y (.O(Y\_coding), .sync\_clk(sync\_clk\_Y), .in(rounded\_Y\_int), .clk(clk), .rst(reset\_coding));

coding code\_Cb (.O(Cb\_coding), .sync\_clk(sync\_clk\_Cb), .in(rounded\_Cb\_int), .clk(clk), .rst(reset\_coding));

coding code\_Cr (.O(Cr\_coding), .sync\_clk(sync\_clk\_Cr), .in(rounded\_Cr\_int), .clk(clk), .rst(reset\_coding));

//----------------------------------------------------------------------------------------------------//

//----- Inverse line coding

decoding decoding\_Y (.O(decode\_Y), .in(Y\_coding), .clk(sync\_clk\_Y), .rst(reset\_coding));

decoding decoding\_Cb (.O(decode\_Cb), .in(Cb\_coding), .clk(sync\_clk\_Cb), .rst(reset\_coding));

decoding decoding\_Cr (.O(decode\_Cr), .in(Cr\_coding), .clk(sync\_clk\_Cr), .rst(reset\_coding));

//----------------------------------------------------------------------------------------------------//

//----- Unquantization

assign reset\_unquatization = reset\_quatization;

Y\_unquantization recover\_Y (unquantized\_Y, rounded\_Y\_float, clk, reset\_unquatization);

Cb\_Cr\_unquantization recover\_Cb (unquantized\_Cb, rounded\_Cb\_float, clk, reset\_unquatization);

Cb\_Cr\_unquantization recover\_Cr (unquantized\_Cr, rounded\_Cr\_float, clk, reset\_unquatization);

//----------------------------------------------------------------------------------------------------//

//----- Reverse DCT 2D

control\_r\_DCT control\_r\_dct (.r\_dct\_2\_D\_reset(r\_dct\_2\_D\_rst), .clk(clk));

DCT\_2D\_reverse r\_DCT\_2D\_Y (r\_Y, unquantized\_Y, clk, r\_dct\_2\_D\_rst);

DCT\_2D\_reverse r\_DCT\_2D\_Cb (r\_Cb, unquantized\_Cb, clk, r\_dct\_2\_D\_rst);

DCT\_2D\_reverse r\_DCT\_2D\_Cr (r\_Cr, unquantized\_Cr, clk, r\_dct\_2\_D\_rst);

//----- divide by 8\*39\*39 --> finish calculating r\_DCT\_2D

FP fp\_divide\_Y (.O(r\_dct\_2D\_Y\_float) , .in0(r\_Y) , .in1(32'h38AC598B), .operand(2'h2));

FP fp\_divide\_Cb (.O(r\_dct\_2D\_Cb\_float), .in0(r\_Cb), .in1(32'h38AC598B), .operand(2'h2));

FP fp\_divide\_Cr (.O(r\_dct\_2D\_Cr\_float), .in0(r\_Cr), .in1(32'h38AC598B), .operand(2'h2));

//----------------------------------------------------------------------------------------------------//

//----- un-normalize

//----- change colour space from YCbCr to RGB

convert\_color\_space\_R convert\_into\_R (R\_result\_float, r\_dct\_2D\_Y\_float, r\_dct\_2D\_Cb\_float, r\_dct\_2D\_Cr\_float);

convert\_color\_space\_G convert\_into\_G (G\_result\_float, r\_dct\_2D\_Y\_float, r\_dct\_2D\_Cb\_float, r\_dct\_2D\_Cr\_float);

convert\_color\_space\_B convert\_into\_B (B\_result\_float, r\_dct\_2D\_Y\_float, r\_dct\_2D\_Cb\_float, r\_dct\_2D\_Cr\_float);

//----------------------------------------------------------------------------------------------------//

//----- take output

convert\_float\_into\_int R\_result (R\_result\_int, R\_result\_float);

convert\_float\_into\_int G\_result (G\_result\_int, G\_result\_float);

convert\_float\_into\_int B\_result (B\_result\_int, B\_result\_float);

mux\_2\_1 #(8) scaled\_R\_space (scaled\_R, R\_result\_int[7:0], 8'hff, (R\_result\_int[8] | R\_result\_int[9]));

mux\_2\_1 #(8) scaled\_G\_space (scaled\_G, G\_result\_int[7:0], 8'hff, (G\_result\_int[8] | G\_result\_int[9]));

mux\_2\_1 #(8) scaled\_B\_space (scaled\_B, B\_result\_int[7:0], 8'hff, (B\_result\_int[8] | B\_result\_int[9]));

mux\_2\_1 #(8) R\_output (R\_out, scaled\_R, 8'b0, R\_result\_int[24]);

mux\_2\_1 #(8) G\_output (G\_out, scaled\_G, 8'b0, G\_result\_int[24]);

mux\_2\_1 #(8) B\_output (B\_out, scaled\_B, 8'b0, B\_result\_int[24]);

endmodule

//====================================================================================================//

module control\_DCT (dct\_2\_D\_reset, clk);

output reg dct\_2\_D\_reset;

input clk;

reg [31:0] counter;

always @(posedge clk) begin

counter = counter + 32'b1;

if (counter == 32'h03 || dct\_2\_D\_reset == 1'b0) begin

dct\_2\_D\_reset = 1'b0;

end

else begin

dct\_2\_D\_reset = 1'b1;

end

end

initial begin

counter = 32'b0;

dct\_2\_D\_reset = 1'b1;

end

endmodule

//====================================================================================================//

module control\_quatization (reset\_quatization, clk, rst);

output reg reset\_quatization;

input clk, rst;

reg [6:0] counter;

always @(posedge clk or posedge rst) begin

if (rst == 1'b1) begin

counter = 7'b0;

reset\_quatization = 1'b1;

end

else begin

if(counter == 7'h53) begin

reset\_quatization = 1'b0;

end

counter = counter + 7'b1;

end

end

endmodule

//====================================================================================================//

module control\_r\_DCT (r\_dct\_2\_D\_reset, clk);

output reg r\_dct\_2\_D\_reset;

input clk;

reg [31:0] counter;

always @(posedge clk) begin

counter = counter + 32'b1;

if (counter == /\*32'h46\*/32'h56 || r\_dct\_2\_D\_reset == 1'b0) begin

r\_dct\_2\_D\_reset = 1'b0;

end

end

initial begin

counter = 32'b0;

r\_dct\_2\_D\_reset = 1'b1;

end

endmodule

//====================================================================================================//

module transpose (O, clk, rst);

output reg O;

input clk, rst;

reg [31:0] counter;

always @(posedge clk or posedge rst) begin

if (rst == 1'b1) begin

counter = 32'b0;

O = 1'b0;

end

else begin

if(counter == 32'h54) begin

O = 1'b1;

end

counter = counter + 32'b1;

end

end

endmodule

//====================================================================================================//

module reset\_code (O, clk, rst);

output reg O;

input clk, rst;

reg [7:0] counter;

always @(posedge clk) begin

if (rst) begin

counter = 8'b0;

O = 1'b1;

end

else begin

if(counter == 8'd85) begin

O = 1'b0;

end

counter = counter + 8'b1;

end

end

endmodule

## file dct\_2\_D.v

`timescale 1ps/1ps

//====================================================================================================//

module dct\_2\_D (O, in, clk, rst);

parameter bits = 25;

output [bits-1:0] O;

input [bits-1:0] in;

input clk, rst;

reg [6:0] state;

wire [bits-1:0] dct\_a\_O, dct\_b\_O, mem\_a\_O, mem\_b\_O, mux\_a\_O, mux\_b\_O, mux\_c\_O;

wire [bits-1:0] dct\_a\_in, dct\_b\_in, mem\_a\_in, mem\_b\_in, mux\_a\_0, mux\_a\_1, mux\_b\_0, mux\_b\_1, mux\_c\_0, mux\_c\_1;

wire mem\_a\_en, mem\_b\_en, mux\_a\_select, mux\_b\_select, mux\_c\_select;

wire Q, a, sensitivity, dct\_a\_finish, dct\_b\_finish, mem\_a\_start\_count\_state, mem\_b\_start\_count\_state;

wire reset, dieu\_chinh\_in;

//----- list all modules --------------------

dct dct\_a (dct\_a\_O, dct\_a\_finish, dct\_a\_in, clk, rst);

dieu\_chinh dc (reset, dieu\_chinh\_in, clk, rst);

dct dct\_b (dct\_b\_O, dct\_b\_finish, dct\_b\_in, clk, reset);

mem\_8x8 #(25) mem\_a (mem\_a\_O, mem\_a\_in, clk, mem\_a\_en, rst, mem\_a\_start\_count\_state);

mem\_8x8 #(25) mem\_b (mem\_b\_O, mem\_b\_in, clk, mem\_b\_en, rst, mem\_b\_start\_count\_state);

mux\_2\_1 #(bits) mux\_a (mux\_a\_O, mux\_a\_0, mux\_a\_1, mux\_a\_select);

mux\_2\_1 #(bits) mux\_b (mux\_b\_O, mux\_b\_0, mux\_b\_1, mux\_b\_select);

mux\_2\_1 #(bits) mux\_c (mux\_c\_O, mux\_c\_0, mux\_c\_1, mux\_c\_select);

counter\_8\_state C (sensitivity, dct\_a\_finish,rst);

D\_flipflop D (Q, a, a, sensitivity);

delay\_1\_clk stall (.O(mux\_c\_select), .in(~Q), .clk(clk), .rst(rst));

//----- connect wires --------------------

assign dct\_a\_in[bits-1:0] = in[bits-1:0];

assign mux\_a\_1[bits-1:0] = dct\_a\_O[bits-1:0];

assign mux\_b\_1[bits-1:0] = dct\_a\_O[bits-1:0];

assign mux\_a\_0[bits-1:0] = 25'b0;

assign mux\_b\_0[bits-1:0] = 25'b0;

assign mem\_a\_in[bits-1:0] = mux\_a\_O[bits-1:0];

assign mem\_b\_in[bits-1:0] = mux\_b\_O[bits-1:0];

assign mux\_c\_0[bits-1:0] = mem\_b\_O[bits-1:0];

assign mux\_c\_1[bits-1:0] = mem\_a\_O[bits-1:0];

assign dct\_b\_in[bits-1:0] = mux\_c\_O[bits-1:0];

assign mem\_a\_en = Q;

assign mem\_b\_en = ~Q;

assign mux\_a\_select = Q;

assign mux\_b\_select = ~Q;

// assign mux\_c\_select = ~Q;

assign mem\_a\_start\_count\_state = dct\_a\_finish;

assign mem\_b\_start\_count\_state = dct\_a\_finish;

assign dieu\_chinh\_in = dct\_a\_finish;

assign O[bits-1:0] = dct\_b\_O[bits-1:0];

endmodule

//====================================================================================================//

// D flip-flop

module D\_flipflop(Q, Q\_n, D, clk);

output reg Q, Q\_n;

input D, clk;

initial begin

Q\_n = 1'b1;

Q = 1'b0;

end

always @(posedge clk) begin

Q = D;

Q\_n = ~Q;

end

endmodule

//====================================================================================================//

// count 8 state

module counter\_8\_state(Q, clk, rst);

output reg Q;

input clk, rst;

reg [2:0] state;

reg [1:0] starting;

always @(posedge clk or posedge rst) begin

if(rst == 1'b1) state = 3'b000;

else begin

if(starting[1] == 1'b1) begin

case(state)

3'b000: begin state = 3'b001; Q = 1'b0; end

3'b001: state = 3'b010;

3'b010: state = 3'b011;

3'b011: state = 3'b100;

3'b100: state = 3'b101;

3'b101: state = 3'b110;

3'b110: state = 3'b111;

3'b111: begin state = 3'b000; Q = 1'b1; end

endcase

end

end

end

always @(posedge clk) begin

if(starting[0] == 1'b1) starting[1] = 1'b1;

starting[0] = 1'b1;

end

endmodule

//====================================================================================================//

//----- module dieu chinh clock -----

module dieu\_chinh(reset, in, clk, rst);

output reset;

input in, clk, rst;

reg [1:0] state;

always @(posedge clk) begin

if(rst == 1'b1) state = 2'b00;

else begin

case(state)

2'b00: if(in == 1'b1) state = 2'b01;

2'b01: state = 2'b10;

2'b10: state = 2'b11;

endcase

end

end

assign reset = (state == 2'b01);

endmodule

//====================================================================================================//

module delay\_1\_clk (O, in, clk, rst);

output reg O;

input in, clk, rst;

reg mem;

always @(posedge clk) begin

if(rst == 1'b1) begin

mem = 1'b0;

O = 1'b0;

end

else begin

mem = in;

O = mem;

end

end

endmodule

## file dct.v

`timescale 1ps/1ps

module dct(O, finish\_computing, in, clk, rst);

parameter bits = 25;

parameter S0=4'd0, S1=4'd1, S2=4'd2, S3=4'd3, S4=4'd4, S5=4'd5, S6=4'd6, S7=4'd7;

parameter S8=4'd8, S9=4'd9, S10=4'd10, S11=4'd11, S12=4'd12, S13=4'd13, S14=4'd14, S15=4'd15;

output reg [bits-1:0] O;

output reg finish\_computing;

input [bits-1:0] in;

input clk, rst;

reg [bits-1:0] Mem\_in [7:0];

reg [bits-1:0] Mem\_out [7:0];

reg [3:0] state;

wire [bits:0] na\_s\_a0\_0, na\_s\_a1\_0, na\_s\_a2\_0, na\_s\_a3\_0, na\_s\_a4\_0, na\_s\_a5\_0, na\_s\_a6\_0, na\_s\_a7\_0;

wire [bits:0] na\_s\_b0\_0, na\_s\_b1\_0, na\_s\_b2\_0, na\_s\_b3\_0, na\_s\_b4\_0, na\_s\_b5\_0, na\_s\_b6\_0, na\_s\_b7\_0;

wire [bits:0] na\_s\_w0\_0, na\_s\_w1\_0, na\_s\_w2\_0, na\_s\_w3\_0, na\_s\_w4\_0, na\_s\_w5\_0, na\_s\_w6\_0, na\_s\_w7\_0;

wire [bits:0] dct0, dct1, dct2, dct3, dct4, dct5, dct6, dct7;

wire [bits-1:0] m2w5, m2w6, m6w7, m6w4, m8w4, m8w7, m14w7, m14w4, m15w3, m15w2, m16w5, m16w6, m19w5, m19w6, m30w5, m30w6, m35w4, m35w7, m36w2, m36w3, m39w0, m39w1;

wire [bits:0] sub11\_0, add12\_0, add13\_0, sub8\_0, add9\_0, add10\_0, add3\_0, sub7\_0, add6\_0, add4\_0, sub5\_0, add0\_0, add1\_0, sub2\_0;

wire [bits-1:0] in\_25bits;

//----------

assign in\_25bits[bits-1:0] = in[bits-1:0];

sub\_nadd\_25\_sign\_bits na\_s\_a0 (na\_s\_a0\_0, Mem\_in[0], Mem\_in[7], 1'b0); // x0 + x7

sub\_nadd\_25\_sign\_bits na\_s\_a1 (na\_s\_a1\_0, Mem\_in[1], Mem\_in[6], 1'b0); // x1 + x6

sub\_nadd\_25\_sign\_bits na\_s\_a2 (na\_s\_a2\_0, Mem\_in[2], Mem\_in[5], 1'b0); // x2 + x5

sub\_nadd\_25\_sign\_bits na\_s\_a3 (na\_s\_a3\_0, Mem\_in[3], Mem\_in[4], 1'b0); // x3 + x4

sub\_nadd\_25\_sign\_bits na\_s\_a4 (na\_s\_a4\_0, Mem\_in[0], Mem\_in[7], 1'b1); // x0 - x7

sub\_nadd\_25\_sign\_bits na\_s\_a5 (na\_s\_a5\_0, Mem\_in[1], Mem\_in[6], 1'b1); // x1 - x6

sub\_nadd\_25\_sign\_bits na\_s\_a6 (na\_s\_a6\_0, Mem\_in[2], Mem\_in[5], 1'b1); // x2 - x5

sub\_nadd\_25\_sign\_bits na\_s\_a7 (na\_s\_a7\_0, Mem\_in[3], Mem\_in[4], 1'b1); // x3 - x4

sub\_nadd\_25\_sign\_bits na\_s\_b0 (na\_s\_b0\_0, {na\_s\_a0\_0[bits], na\_s\_a0\_0[bits-2:0]}, {na\_s\_a1\_0[bits], na\_s\_a1\_0[bits-2:0]}, 1'b0); // a0+a1

sub\_nadd\_25\_sign\_bits na\_s\_b1 (na\_s\_b1\_0, {na\_s\_a4\_0[bits], na\_s\_a4\_0[bits-2:0]}, {na\_s\_a5\_0[bits], na\_s\_a5\_0[bits-2:0]}, 1'b0); // a4+a5

sub\_nadd\_25\_sign\_bits na\_s\_b2 (na\_s\_b2\_0, {na\_s\_a2\_0[bits], na\_s\_a2\_0[bits-2:0]}, {na\_s\_a3\_0[bits], na\_s\_a3\_0[bits-2:0]}, 1'b0); // a2+a3

sub\_nadd\_25\_sign\_bits na\_s\_b3 (na\_s\_b3\_0, {na\_s\_a6\_0[bits], na\_s\_a6\_0[bits-2:0]}, {na\_s\_a7\_0[bits], na\_s\_a7\_0[bits-2:0]}, 1'b0); // a6+a7

sub\_nadd\_25\_sign\_bits na\_s\_b4 (na\_s\_b4\_0, {na\_s\_a0\_0[bits], na\_s\_a0\_0[bits-2:0]}, {na\_s\_a1\_0[bits], na\_s\_a1\_0[bits-2:0]}, 1'b1); // a0-a1

sub\_nadd\_25\_sign\_bits na\_s\_b5 (na\_s\_b5\_0, {na\_s\_a4\_0[bits], na\_s\_a4\_0[bits-2:0]}, {na\_s\_a5\_0[bits], na\_s\_a5\_0[bits-2:0]}, 1'b1); // a4-a5

sub\_nadd\_25\_sign\_bits na\_s\_b6 (na\_s\_b6\_0, {na\_s\_a2\_0[bits], na\_s\_a2\_0[bits-2:0]}, {na\_s\_a3\_0[bits], na\_s\_a3\_0[bits-2:0]}, 1'b1); // a2-a3

sub\_nadd\_25\_sign\_bits na\_s\_b7 (na\_s\_b7\_0, {na\_s\_a6\_0[bits], na\_s\_a6\_0[bits-2:0]}, {na\_s\_a7\_0[bits], na\_s\_a7\_0[bits-2:0]}, 1'b1); // a6-a7

sub\_nadd\_25\_sign\_bits na\_s\_w0 (na\_s\_w0\_0, {na\_s\_b0\_0[bits], na\_s\_b0\_0[bits-2:0]}, {na\_s\_b2\_0[bits], na\_s\_b2\_0[bits-2:0]}, 1'b0); // b0+b2

sub\_nadd\_25\_sign\_bits na\_s\_w1 (na\_s\_w1\_0, {na\_s\_b4\_0[bits], na\_s\_b4\_0[bits-2:0]}, {na\_s\_b6\_0[bits], na\_s\_b6\_0[bits-2:0]}, 1'b1); // b4-b6

sub\_nadd\_25\_sign\_bits na\_s\_w2 (na\_s\_w2\_0, {na\_s\_b0\_0[bits], na\_s\_b0\_0[bits-2:0]}, {na\_s\_b2\_0[bits], na\_s\_b2\_0[bits-2:0]}, 1'b1); // b0-b2

sub\_nadd\_25\_sign\_bits na\_s\_w3 (na\_s\_w3\_0, {na\_s\_b4\_0[bits], na\_s\_b4\_0[bits-2:0]}, {na\_s\_b6\_0[bits], na\_s\_b6\_0[bits-2:0]}, 1'b0); // b4+b6

sub\_nadd\_25\_sign\_bits na\_s\_w4 (na\_s\_w4\_0, {na\_s\_b1\_0[bits], na\_s\_b1\_0[bits-2:0]}, {na\_s\_b3\_0[bits], na\_s\_b3\_0[bits-2:0]}, 1'b0); // b1+b3

sub\_nadd\_25\_sign\_bits na\_s\_w5 (na\_s\_w5\_0, {na\_s\_b5\_0[bits], na\_s\_b5\_0[bits-2:0]}, {na\_s\_b7\_0[bits], na\_s\_b7\_0[bits-2:0]}, 1'b1); // b5-b7

sub\_nadd\_25\_sign\_bits na\_s\_w6 (na\_s\_w6\_0, {na\_s\_b1\_0[bits], na\_s\_b1\_0[bits-2:0]}, {na\_s\_b3\_0[bits], na\_s\_b3\_0[bits-2:0]}, 1'b1); // b1-b3

sub\_nadd\_25\_sign\_bits na\_s\_w7 (na\_s\_w7\_0, {na\_s\_b5\_0[bits], na\_s\_b5\_0[bits-2:0]}, {na\_s\_b7\_0[bits], na\_s\_b7\_0[bits-2:0]}, 1'b0); // b5+b7

/\* T\*Wx =

\* a\*w0

\* a\*w1

\* b\*w2 + c\*w3

\* -c\*w2 + b\*w3

\* d\*w4 - e\*w5 + h\*w6 + y\*w7

\* f\*w4 + g\*w5 - j\*w6 + k\*w7

\* -k\*w4 + j\*w5 + g\*w6 + f\*w7

\* -y\*w4 - h\*w5 - e\*w6 + d\*w7

\*

\* a=39; b=36; c=15; d=35; e=2; f=8; g=30; h=16; y=6; j=19; k=14;

\*/

multiplier\_2 m0 (m2w5 , {na\_s\_w5\_0[bits], na\_s\_w5\_0[bits-2:0]});

multiplier\_2 m1 (m2w6 , {na\_s\_w6\_0[bits], na\_s\_w6\_0[bits-2:0]});

multiplier\_6 m2 (m6w7 , {na\_s\_w7\_0[bits], na\_s\_w7\_0[bits-2:0]});

multiplier\_6 m3 (m6w4 , {na\_s\_w4\_0[bits], na\_s\_w4\_0[bits-2:0]});

multiplier\_8 m4 (m8w4 , {na\_s\_w4\_0[bits], na\_s\_w4\_0[bits-2:0]});

multiplier\_8 m5 (m8w7 , {na\_s\_w7\_0[bits], na\_s\_w7\_0[bits-2:0]});

multiplier\_14 m6 (m14w7 , {na\_s\_w7\_0[bits], na\_s\_w7\_0[bits-2:0]});

multiplier\_14 m7 (m14w4 , {na\_s\_w4\_0[bits], na\_s\_w4\_0[bits-2:0]});

multiplier\_15 m8 (m15w3 , {na\_s\_w3\_0[bits], na\_s\_w3\_0[bits-2:0]});

multiplier\_15 m9 (m15w2 , {na\_s\_w2\_0[bits], na\_s\_w2\_0[bits-2:0]});

multiplier\_16 m10 (m16w5 , {na\_s\_w5\_0[bits], na\_s\_w5\_0[bits-2:0]});

multiplier\_16 m11 (m16w6 , {na\_s\_w6\_0[bits], na\_s\_w6\_0[bits-2:0]});

multiplier\_19 m12 (m19w5 , {na\_s\_w5\_0[bits], na\_s\_w5\_0[bits-2:0]});

multiplier\_19 m13 (m19w6 , {na\_s\_w6\_0[bits], na\_s\_w6\_0[bits-2:0]});

multiplier\_30 m14 (m30w5 , {na\_s\_w5\_0[bits], na\_s\_w5\_0[bits-2:0]});

multiplier\_30 m21 (m30w6 , {na\_s\_w6\_0[bits], na\_s\_w6\_0[bits-2:0]});

multiplier\_35 m15 (m35w4 , {na\_s\_w4\_0[bits], na\_s\_w4\_0[bits-2:0]});

multiplier\_35 m16 (m35w7 , {na\_s\_w7\_0[bits], na\_s\_w7\_0[bits-2:0]});

multiplier\_36 m17 (m36w2 , {na\_s\_w2\_0[bits], na\_s\_w2\_0[bits-2:0]});

multiplier\_36 m18 (m36w3 , {na\_s\_w3\_0[bits], na\_s\_w3\_0[bits-2:0]});

multiplier\_39 m19 (m39w0 , {na\_s\_w0\_0[bits], na\_s\_w0\_0[bits-2:0]});

multiplier\_39 m20 (m39w1 , {na\_s\_w1\_0[bits], na\_s\_w1\_0[bits-2:0]});

//dct0

assign dct0[bits-1:0] = m39w0[bits-1:0];

//dct1

sub\_nadd\_25\_sign\_bits sub11 (sub11\_0 , m35w4 , m2w5, 1'b1);

sub\_nadd\_25\_sign\_bits add12 (add12\_0 , {sub11\_0[bits], sub11\_0[bits-2:0]}, m16w6, 1'b0);

sub\_nadd\_25\_sign\_bits add13 (add13\_0 , {add12\_0[bits], add12\_0[bits-2:0]}, m6w7, 1'b0);

assign dct1[bits-1:0] = {add13\_0[bits], add13\_0[bits-2:0]};

//dct2

sub\_nadd\_25\_sign\_bits add6 (add6\_0 , m36w2 , m15w3, 1'b0);

assign dct2[bits-1:0] = {add6\_0[bits], add6\_0[bits-2:0]};

//dct3

sub\_nadd\_25\_sign\_bits sub8 (sub8\_0 , m19w5 , m14w4, 1'b1);

sub\_nadd\_25\_sign\_bits add9 (add9\_0 , {sub8\_0[bits], sub8\_0[bits-2:0]}, m30w6, 1'b0);

sub\_nadd\_25\_sign\_bits add10 (add10\_0 , {add9\_0[bits], add9\_0[bits-2:0]}, m8w7, 1'b0);

assign dct3[bits-1:0] = {add10\_0[bits], add10\_0[bits-2:0]};

//dct4

assign dct4[bits-1:0] = m39w1[bits-1:0];

//dct5

sub\_nadd\_25\_sign\_bits add3 (add3\_0 , m8w4 , m30w5, 1'b0);

sub\_nadd\_25\_sign\_bits add4 (add4\_0 , {add3\_0[bits], add3\_0[bits-2:0]} , m14w7, 1'b0);

sub\_nadd\_25\_sign\_bits sub5 (sub5\_0 , {add4\_0[bits], add4\_0[bits-2:0]} , m19w6, 1'b1);

assign dct5[bits-1:0] = {sub5\_0[bits], sub5\_0[bits-2:0]};

//dct6

sub\_nadd\_25\_sign\_bits sub7 (sub7\_0 , m36w3, m15w2, 1'b1);

assign dct6[bits-1:0] = {sub7\_0[bits], sub7\_0[bits-2:0]};

//dct7

sub\_nadd\_25\_sign\_bits add0 (add0\_0 , m6w4, m16w5, 1'b0);

sub\_nadd\_25\_sign\_bits add1 (add1\_0 , {add0\_0[bits], add0\_0[bits-2:0]}, m2w6 , 1'b0);

sub\_nadd\_25\_sign\_bits sub2 (sub2\_0 , m35w7, {add1\_0[bits], add1\_0[bits-2:0]}, 1'b1);

assign dct7[bits-1:0] = {sub2\_0[bits], sub2\_0[bits-2:0]};

always @(posedge clk) begin

if(rst) begin

Mem\_in[0] = 8'b0;

Mem\_in[1] = 8'b0;

Mem\_in[2] = 8'b0;

Mem\_in[3] = 8'b0;

Mem\_in[4] = 8'b0;

Mem\_in[5] = 8'b0;

Mem\_in[6] = 8'b0;

Mem\_in[7] = 8'b0;

O = 8'b0;

finish\_computing = 1'b0;

state = 4'b0;

end

else begin

case (state)

S0: begin

Mem\_in[0] = in\_25bits; state = S1;

finish\_computing = 1'b1;

Mem\_out[0] = dct0;

Mem\_out[1] = dct1;

Mem\_out[2] = dct2;

Mem\_out[3] = dct3;

Mem\_out[4] = dct4;

Mem\_out[5] = dct5;

Mem\_out[6] = dct6;

Mem\_out[7] = dct7;

O = Mem\_out[0];

end

S1: begin

Mem\_in[1] = in\_25bits; state = S2;

finish\_computing = 1'b0;

O = Mem\_out[1];

end

S2: begin

Mem\_in[2] = in\_25bits; state = S3;

O = Mem\_out[2];

end

S3: begin

Mem\_in[3] = in\_25bits; state = S4;

O = Mem\_out[3];

end

S4: begin

Mem\_in[4] = in\_25bits; state = S5;

O = Mem\_out[4];

end

S5: begin

Mem\_in[5] = in\_25bits; state = S6;

O = Mem\_out[5];

end

S6: begin

Mem\_in[6] = in\_25bits; state = S7;

O = Mem\_out[6];

end

S7: begin

Mem\_in[7] = in\_25bits; state = S0;

O = Mem\_out[7];

end

/\*

S9: begin state = S10; end

S10: begin state = S11; end

S11: begin state = S12; end

S12: begin state = S13; end

S13: begin state = S14; end

S14: begin state = S15; end

S15: begin state = S0; end \*/

endcase

end

end

endmodule