UIT

TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

HIỆN THỰC GIẢI THUẬT NÉN ẢNH HUFFMAN CODING

Giáo viên phụ trách: LÂM ĐỨC KHẢI

Thành viên	Họ và tên	Mã số sinh viên	Đóng góp
1	Mai Anh Phong	18520127	Tìm kiếm tài liệu liên quan, thiết kế, soạn nội dung, thuyết trình.
2	Huỳnh Quang Thái	18521380	Tìm kiếm tài liệu liên quan

MỤC LỤC

CHƯƠNG 1: LỜI MỞ ĐẦU

Tài liệu này nói về hiện thực thuật toán nén ảnh HuffmanCoding bằng ngôn ngữ Verilog.

Sẽ qua các phần khái niện, ứng dụng, cách thức hoạt động, hướng giải quyết, thiết kế.

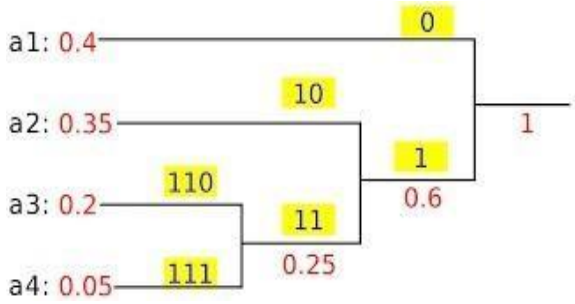
Hướng dẫn sử dụng

Bài báo cáo bài tập nhằm mục đích tìm hiểu và nghiên cứu. Các thông tin chỉ mang tính chất tham khảo và cá nhân. Người đọc cần chú ý cân nhắc.

CHƯƠNG 2: Giới thiệu

2.1. Giải thuật Huffman Coding:

Huffman Coding nén dữ liệu không mất dữ liệu. Giải thuật nén ảnh Huffman Coding **dựa trên tần suất xuất hiện của một mục dữ liệu tức là pixel trong hình ảnh** . Các pixel xuất hiện nhiều sẽ được mã hóa với độ sâu bit thấp và ngược lại các pixel xuất hiện ít hơn sẽ được mã hóa với độ sâu bit cao hơn. [1]



Hình 2.1 Ví dụ về cây Huffman

2.2. Tại sao cần thiết:

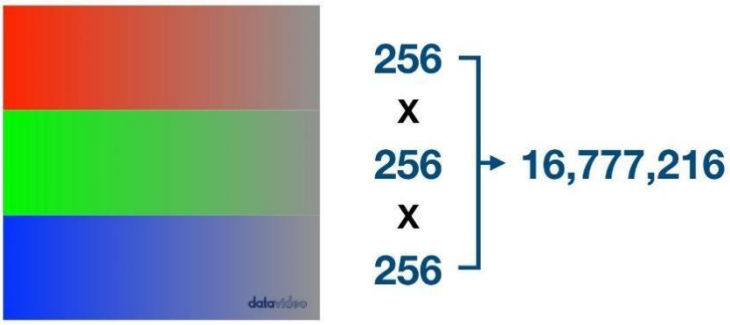
Khi nói đến sản xuất video kỹ thuật số, chúng ta thường thấy 8-bit, 10-bit hoặc thậm chí 12-bit thể hiện thông số xử lý hình ảnh.

Điều đầu tiên để cần phân tích là độ sâu bit:

Độ sâu màu còn được gọi là độ sâu bit đề cập đến số lượng bit được sử dụng để xác định các kênh màu, đỏ, lục hoặc lam cho mỗi pixel.

Hầu hết chúng ta đều nhận thức được rằng pixel là yếu tố cơ bản của bất kỳ hình ảnh nào. Cụ thể, bất kỳ màu nào trong hình ảnh kỹ thuật số được thể hiện bằng sự kết hợp của các sắc thái đỏ, lục và lam. Các sự kết hợp như vậy được sử dụng trên mỗi pixel và từ hàng triệu pixel tạo nên một hình ảnh. Chính vì lý do này mà độ sâu bit còn được gọi là độ sâu màu.

Ví dụ: Một hình ảnh màu 24bit có dãi màu trải dài từ 0000.0000. đến 1111.1111 hay là từ 0-255 cho mỗi dãi màu đỏ, xanh, lam.
Màu đỏ thuần được biểu thị bằng các số “255, 0, 0” Màu xanh lá cây thuần túy là 0, 255, 0 và màu xanh lam thuần túy là 0, 0, 255. Và “255, 100, 150” cho một màu hồng. Điều này có nghĩa Ta chỉ cần thay đổi thông số trong 3 thông số màu này sẽ ra được màu mới. Vậy có nghãi có tổng cộng 256x256x256 hoặc 16.777.216 màu trong một bức ảnh 24bit. [2]



Hình 1.2: Dải màu của 1 pixel

Hình ảnh cho thấy độ liên qua giữa độ sâu bit ảnh hưởng đến chất lượng và kích thước ảnh:[3]

2 màu
4 KB (−96%)

4 màu
6 KB (−94%)

256 màu
37 KB (−62%)

16,777,216 màu
98 KB

Vì vậy một hình ảnh có độ sâu bit cao sẽ có kích thước lớn và sẽ khó khăn trong việc truyền và lưu trữ.

Mục đích của việc nén hình ảnh là giảm lượng dữ liệu cần thiết để biểu diễn hình ảnh kỹ thuật số được lấy mẫu và do đó giảm chi phí cho việc lưu trữ và truyền tải. Nén hình ảnh đóng một vai trò quan trọng trong nhiều ứng dụng quan trọng, bao gồm cơ sở dữ liệu hình ảnh, truyền thông hình ảnh, viễn thám (việc sử dụng hình ảnh vệ tinh cho thời tiết và các ứng dụng tài nguyên trái đất). Và một trong nhưng giải thuật nén ảnh là Huffman Coding.

CHƯƠNG 3: Ứng dụng

3.1. Các ứng dụng của giải thuật nén ảnh Huffman Coding

Kỹ thuật mã hóa Huffman đã được sử dụng hiệu quả trong hệ thống nén văn bản, hình ảnh, video và hội nghị như JPEG, MPEG-2, MPEG-4 và H.263, v.v. Thường được sử dụng trong quá trình lưu trữ, truyền và phân tích hình ảnh mà không làm mất dữ liệu ảnh ban đầu.[4]

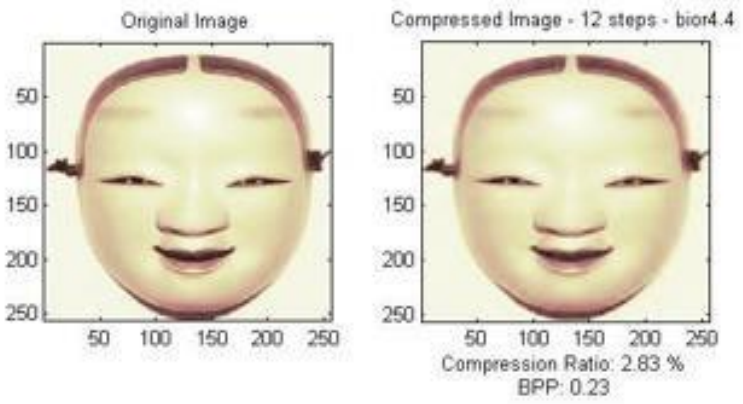
Đây là một ví dụ sau khi thực hiện nén ảnh:

Tỷ lệ nén cuối cùng (2,8%) và tỷ lệ Bit-Per-Pixel (0,23).

Tỷ lệ nén (CR) được định nghĩa là số bit để biểu thị kích thước của ảnh gốc và số bit để biểu thị kích thước của ảnh nén. Tức là bức ảnh sau khi thực hiện giải thuật chỉ được lưu trữ bằng 2,8%. [1]

Khi các khả năng tính toàn biên mới xuất hiện, chúng ta thấy một mô hình điện toán đang thay đổi – một mô hình không còn nhất thiết bị rang buộc bởi nhu cầu xây dựng các trung tâm dữ liệu tập trung. Thay vào đó, đối với một số ứng dụng nhất định, điện toán đám mây sẽ học các bài toán về ảo hóa và điện toán đám mây, đồng thời tạo ra khả năng có hàng nghìn nút phân tán rộng lớn tiềm năng có thể được áp dụng cho các trường hợp sử dụng đa dạng, chẳng hạn như IoT hoặc thậm chí các mạng giám sát từ xa để theo dõi việc sữ dụng tài nghiên nước theo thời gian thực trên hàng nghìn hoặc hàng triệu vị trí.

Edge computing được phát triên do sự phát triển theo cấp số nhân của các thiết bị IoT, kết nối với internet để nhận thông tin từ đám mây hoặc cung cấm dữ liệu trở lại đám mây. Và nhiều thiết bị IoT tạo ra lượng dữ liệu khổng lồ trong quá trình hoạt động của chúng. Hình 3 sẽ minh họa cái nhìn tổng quan về Edge computing.



Hình 3.1: Ảnh trước và sau khi nén

CHƯƠNG 4: Thuật toán

4.1. Static Huffman coding: [5]

Sử dụng tần suất xuất hiện của dữ liệu để nén dữ liệu. Những dữ liệu nào có tần suất cao thì dùng ít bits còn những dữ diệu nào có tần suất thấp thì dùng nhiều bits.
Cây Huffman [Huffman, 1952] là một cây nhị phân đầy đủ:

- Về ký tự
  - ✦ Nút lá chứa một ký tự
  - ✦ Nút cha sẽ chứa các ký tự của những nút con
  - ✦ Nút con trái sẽ có thứ tự từ điển trước nút con phải
- Về trọng số: mỗi nút sẽ được gán một trọng số
  - ✦ Nút lá có trọng số bằng số lần xuất hiện của ký tự trong dữ liệu
  - ✦ Nút cha có trọng số bằng tổng trọng số của các nút con
  - ✦ Nút con trái có giá trị trọng số nhỏ hơn hoặc bằng trọng số của nút con trái phải
- Mỗi cung sẽ được gán một giá trị: cung trái là 0 và cung phải là 1

Thuật toán nén Huffman Coding: [6] [7]

- Bước 1: Duyệt dữ liệu để lập bảng thống kê số lần xuất hiện của mỗi ký tự
- Bước 2: Tạo cây Huffman từ bảng thống kê
- Bước 3: Tạo bảng mã cho các ký tự
- Bước 4: Duyệt dữ liệu để thay thế các ký tự bằng mã bit tương ứng
- Bước 5: Lưu lại thông tin của cây Huffman dùng để giải nén

Thuật toán tạo cây Huffman: [6] [7]
Từ bảng thông kê tần suất xuất hiện của các ký tự ta khởi tạo cây T gồm có n nút là các ký tự với các trọng số của chúng

- Bước 1: Chọn hai phần tử có trọng số thấp nhất x và y
- Bước 2: Tạo một phần tử z từ x và y sao cho trọng số của z là tổng của x và y và chuỗi của z là tổng của x và y
- Bước 3: Loại bỏ x và y khỏi bảng thống kê
- Bước 4: Thêm z vào bảng thống kê và thêm nút z vào cây T với x và y là nút con trái và phải
- Lặp lại các bước trên cho đến khi chỉ còn một phần tử

Hạn chế:

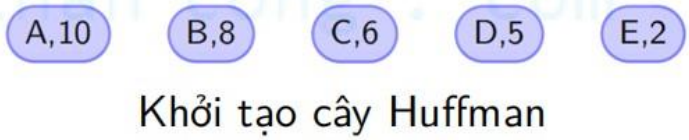
- Duyệt dữ liệu hai lần (thống kế và mã hóa) ⇒ chi phí cao
- Phải lưu trữ cây Huffman ⇒ tăng kích thước dữ liệu nén

- Dữ liệu cần nén phải có sẵn đầy đủ ⇒ không nén được trên dữ liệu phát sinh theo thời gian thực.

Minh họa:[5]
Hãy xây dựng cây Huffman cho chuỗi ký tự
"ADDAABBCCBAAABBCCCBBBCDAADDEEAA"

Tính bảng bảng tần suất cho các ký tự

Chuỗi ký tự	Tần suất
A	10
B	8
C	6
D	5
E	2



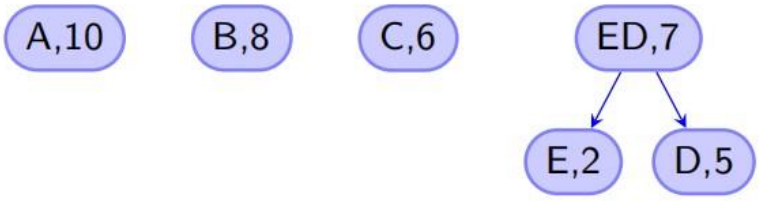
Khởi tạo cây Huffman

Hình 4.1: Khởi tạo

Chuỗi ký tự	Tần suất
A	10
B	8
C	6
D	5
E	2

Loại bỏ D, E và thêm ED rồi sắp xếp lại bảng tần suất

Chuỗi ký tự	Tần suất
A	10
B	8
ED	7
C	6



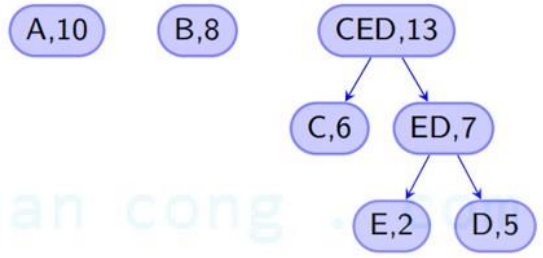
Thêm nút ED cho cây Huffman

Hình 4.2: Chọn 2 phần tử trọng số thấp nhất để tạo nút mới

Chuỗi ký tự	Tần suất
A	10
B	8
ED	7
C	6

Loại bỏ C, ED và thêm CED rồi sắp xếp lại bảng tần suất

Chuỗi ký tự	Tần suất
CED	13
A	10
B	8



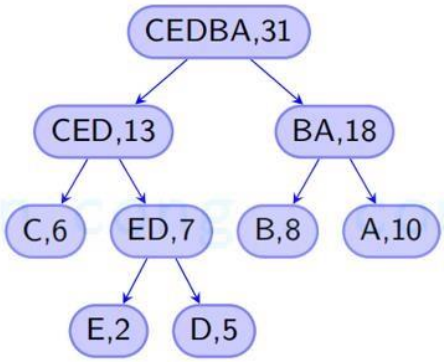
Thêm nút CED cho cây Huffman

Hình 4.3: Tiếp tục cho đến khi chỉ còn 1 nút.

Chuỗi ký tự	Tần suất
BA	18
CED	13

Loại bỏ BA, CED và thêm CEDBA vào bảng tần suất

Chuỗi ký tự	Tần suất
CEDBA	31



Hình 4.4: Cây Huffman hoàn chỉnh.

4.2. Adaptive Huffman coding: (Dynamic Huffman coding): [5]

Là một kỹ thuật mã hóa thích ứng dựa trên mã hóa Huffman . Nó cho phép xây dựng mã khi các ký hiệu đang được truyền đi, không có kiến thức ban đầu về phân phối nguồn, cho phép mã hóa một lần và thích ứng với các điều kiện thay đổi trong dữ liệu.

Lợi ích của thủ tục một lần là nguồn có thể được mã hóa trong thời gian thực, mặc dù nó trở nên nhạy cảm hơn với các lỗi truyền dẫn, vì chỉ một mất mát duy nhất sẽ làm hỏng toàn bộ mã. Cây Adaptive Huffman (Adaptive Huffman Tree) là

- Cây nhị phân đầy đủ
- Mỗi nút có trọng số không âm
- Mỗi nút lá chứa một ký tự và trọng số của nó chính là số lần xuất hiện của ký tự tính đến thời điểm đang xét
- Nút không phải là nút lá trọng số của nó bằng tổng trọng số các nút con của nó
- Có một nút đặc biệt chứa ký hiệu NYT (Not Yet Transmitted Symbol) luôn có trọng số bằng 0 (tương ứng với các ký tự chưa xuất hiện trên cây đến thời điểm đang xét)
- Mỗi cung được gán một giá trị: cung trái 0 và cung phải là 1

Cây Adaptive Huffman phải có tính chất anh em (sibling property)

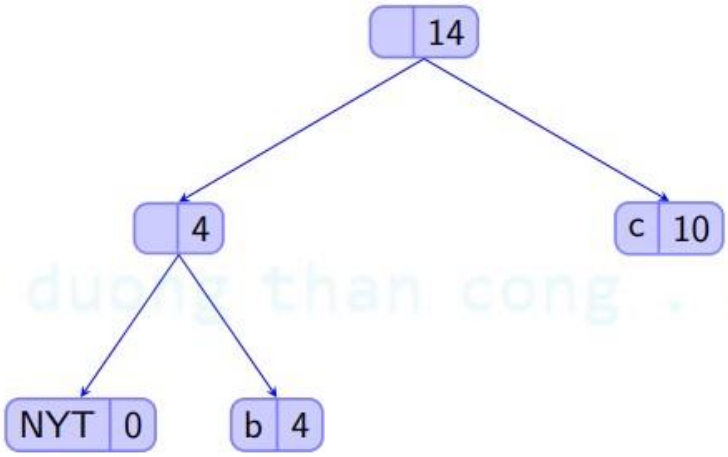
- Các nút không phải nút gốc phải có nút anh em (vì là cây nhị phân đầy đủ)
- Trọng số tăng dần từ dưới → trên, và từ trái → phải Ưu điểm:
- Không cần tính trước số lần xuất hiện của các ký tự
- Quá trình nén chỉ cần 1 lần duyệt dữ liệu
- Không cần lưu thông tin phục vụ cho việc giải nén
- Nén “on-line” dựa trên dữ liệu phát sinh theo thời gian thực Thuật toán giải nén Adaptive Huffman:
- Bước 1: Khởi tạo cây Adaptive Huffman
- Bước 2: Lặp nếu còn dữ liệu
    - ✦ Bước 2.1: Đọc ký tự
    - ✦ Bước 2.2: Mã hóa ký tự
    - ✦ Bước 2.3: Cập nhật cây Adaptive Huffman

Minh họa:
Bước 1: Khởi tạo cây Adaptive Huffman

Hình 4.5: Cây Adaptive Huffman khởi tạo

Bước 2.2: Mã hóa ký tự c. Kiểm tra ký tự c có trong cây Adaptive Huffman hay chưa?

- Nếu có: Phát sinh mã bit cho c (giống như cây Huffman thông thường)
- Nếu chưa có: Phát sinh mã bit bao gồm "mã bit của nút NYT + mã ASCII 8 bit của ký tự c" (?)



Hình 4.6: Mã hóa ký tự đã có trong cây: mã của b là 01, mã của c là 1

Mã hóa ký tự chưa có trong cây: mã của d là : 0001100100
Bước 2.3: Cập nhật cây

- Nếu c có trong cây? Gọi p là nút lá chứa c
- Nếu c chưa có trong cây? tách nút NYT thành hai nút
  - ✦ nút NYT (mới) trọng số 0
  - ✦ và nút lá p chứa c có trọng số khởi tạo là 0

Lặp nếu p khác NULL

1. Nếu p lá nút gốc tăng trọng số lên 1

2. Hoán đổi p với nút xa nhất (tính từ nút p theo hướng trái → phải và dưới → trên) có cùng trọng số (không tính nút cha của p)

3. Tăng trọng số của p lên 1

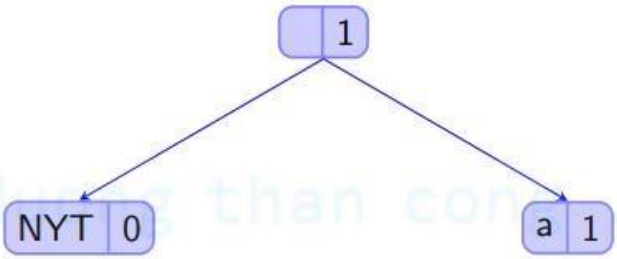
4. Di chuyển p lên nút cha của nó

Minh họa cập nhật dữ liệu

Cho cây Adaptive ban đầu hãy cập nhật cây với chuỗi ký tự “aad”

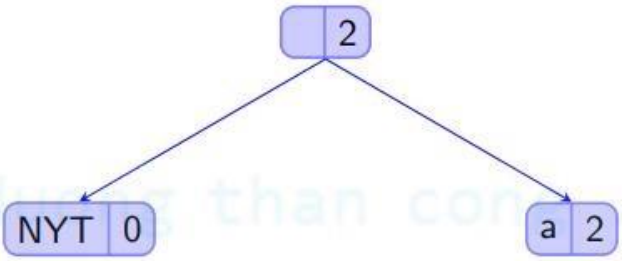
Hình 4.7: Cây Adaptive Huffman khởi đầu

Đọc dữ liệu “aad”



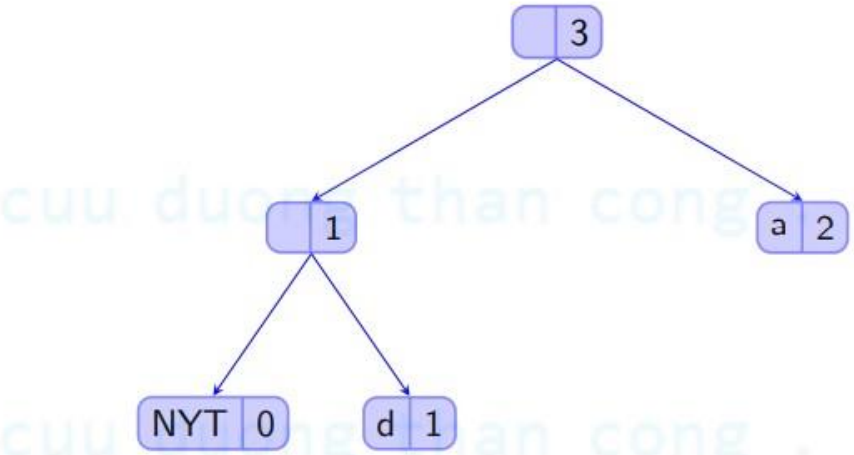
Hình 4.8: Cập nhật cây sau khi thêm a

Đọc dữ liệu “aad”



Hình 4.9: Cập nhật cây sau khi thêm a

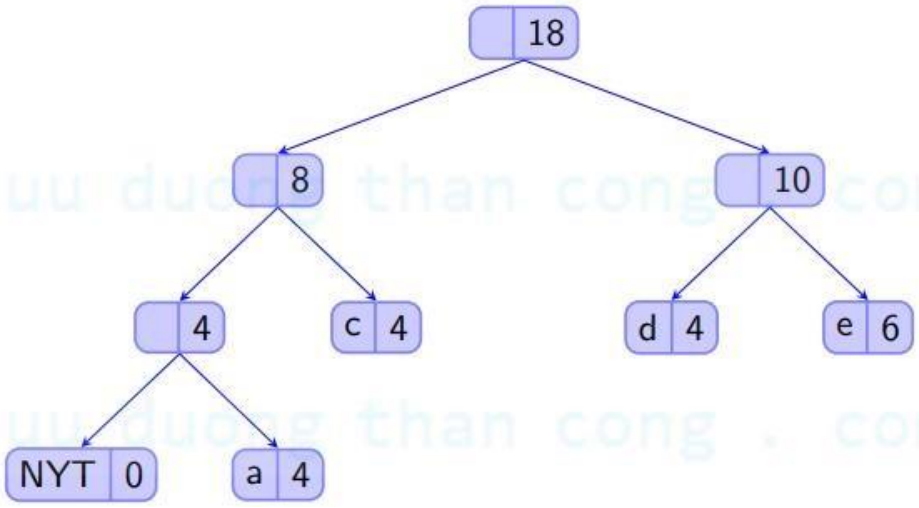
Đọc dữ liệu “aad”



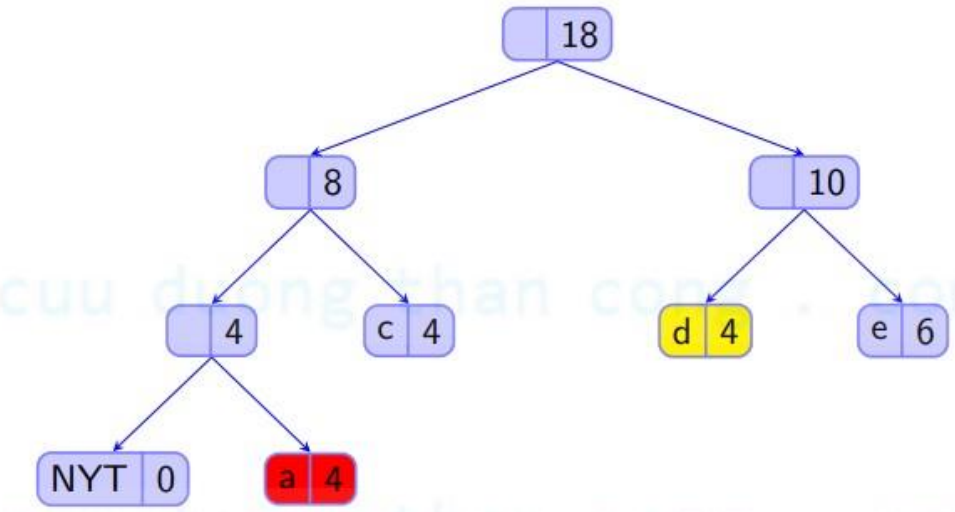
Hình 4.10: Cập nhật cây sau khi thêm d

Minh họa cập nhật trọng số

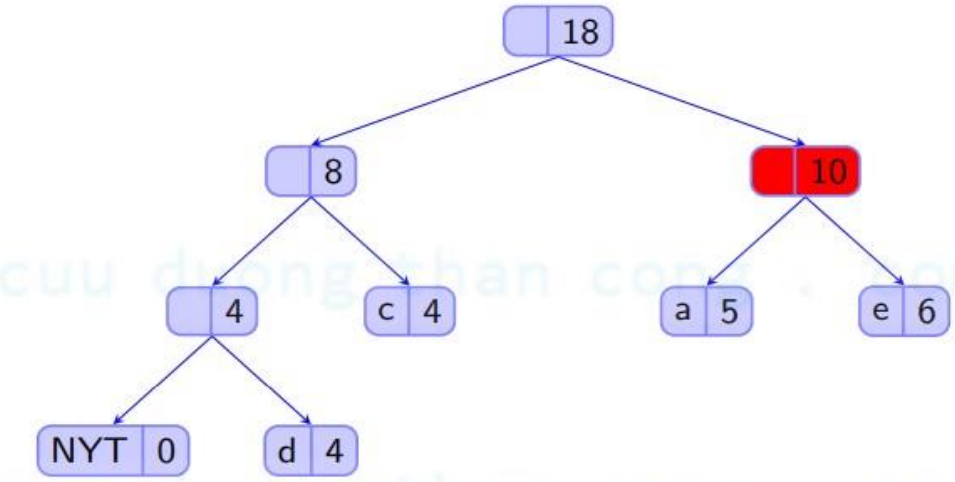
Cho một cây Adaptive Huffman. Và thêm a vào cây



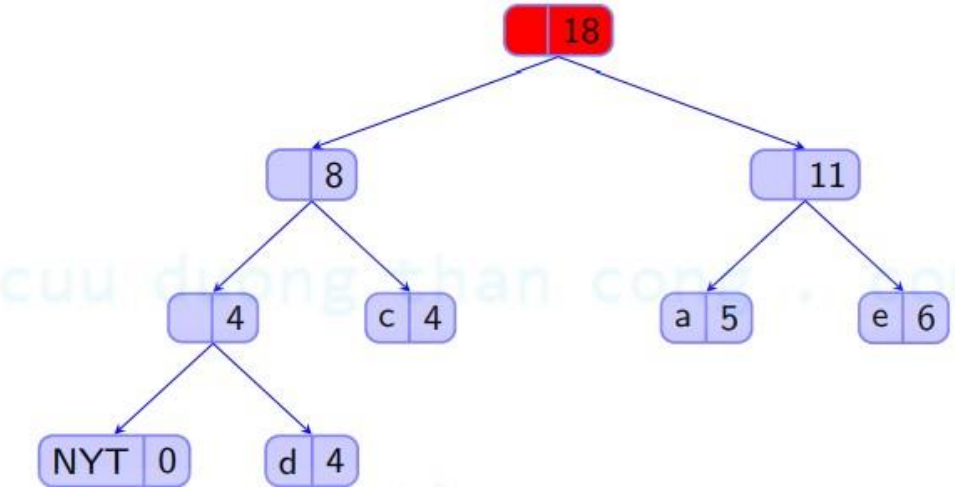
Hình 4.11: Cây Adaptive Huffman



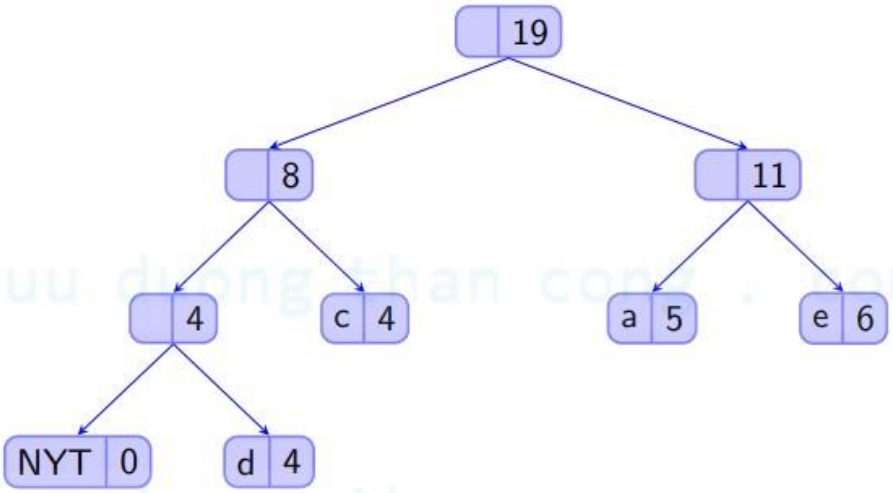
Hình 4.12: Tìm nút xa nhất có trọng số bằng với a và chuẩn bị hoán đổi



Hình 4.13: Cập nhật trọng số và di chuyển lên nút cha



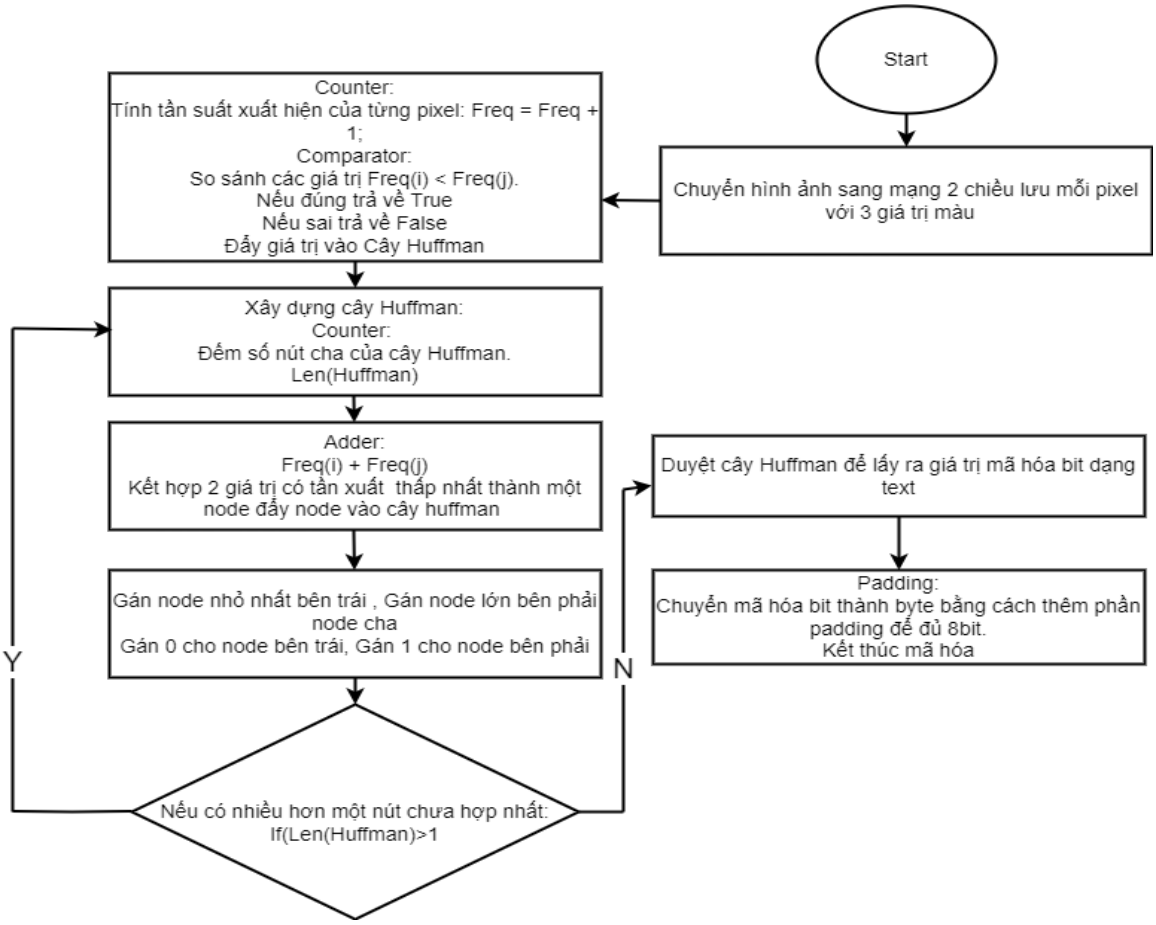
Hình 4.14: Cập nhật trọng số và di chuyển lên nút cha



Hình 4.15: Cập nhật trọng số và kết thúc

CHƯƠNG 5: Hiện thực giải thuật trên phần cứng

5.1. Sơ đồ giải thuật : [6] [7]

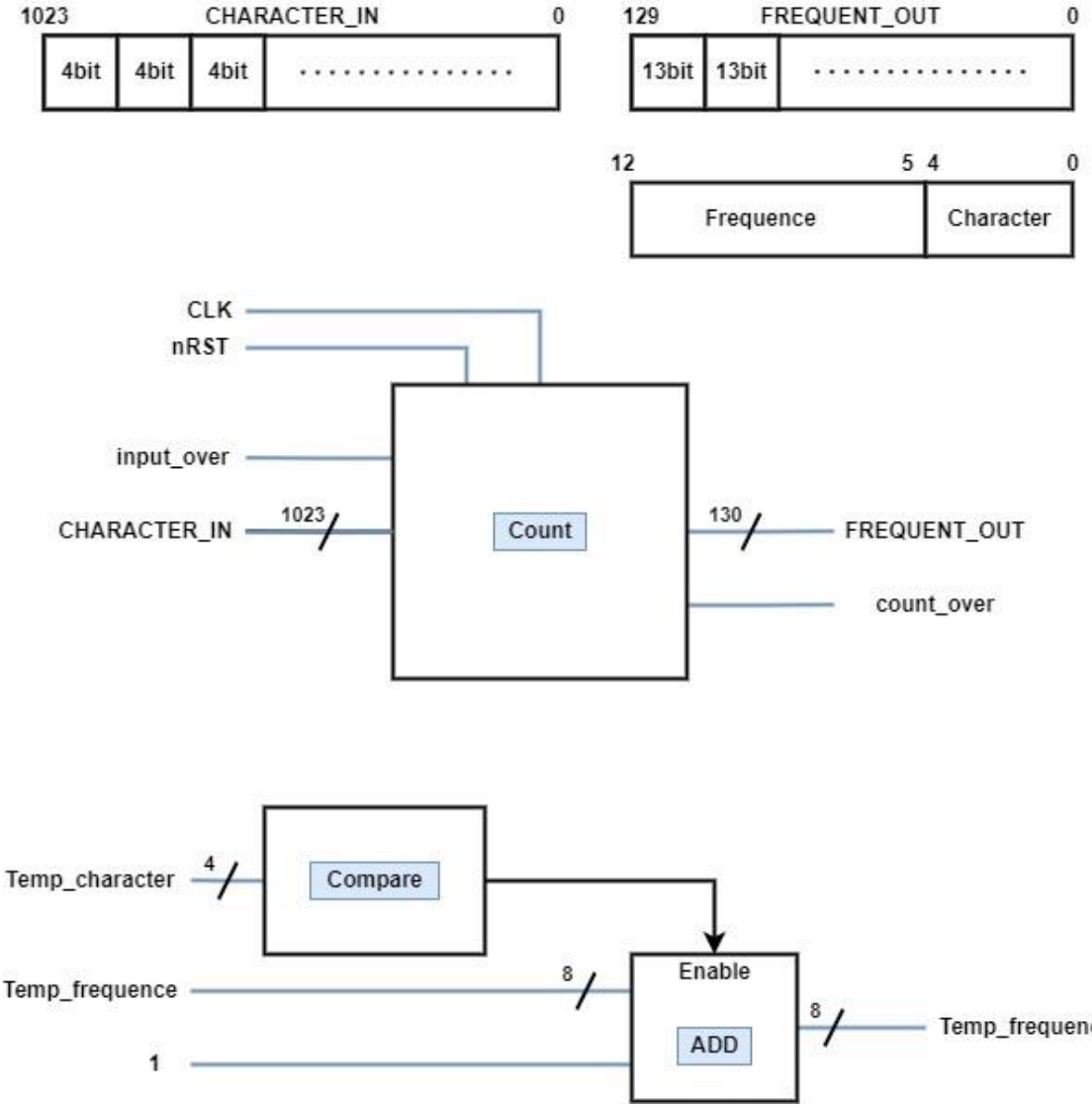


5.2. Module Count :[8]

Chức năng: Mô-đun đếm, chịu trách nhiệm đếm 256 dữ liệu đầu vào và đếm tần số 0 ~ 9 chữ số.

Đầu vào và đầu ra: Đầu vào là 256 dữ liệu 1024 bit và đầu ra là 10 số sau khi đếm 130 bit.

Thiết kế logic: 5 chữ số đầu tiên của mỗi dữ liệu đầu ra đại diện cho số lượng và 8 chữ số cuối thể hiện tần số từ.

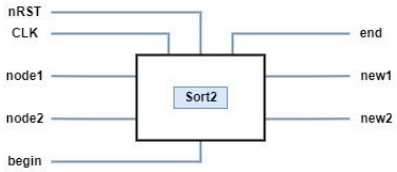


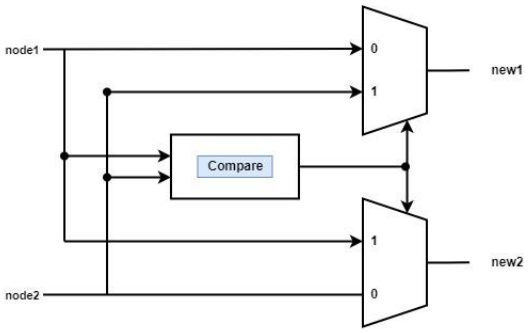
5.3. Module Sort :[8]

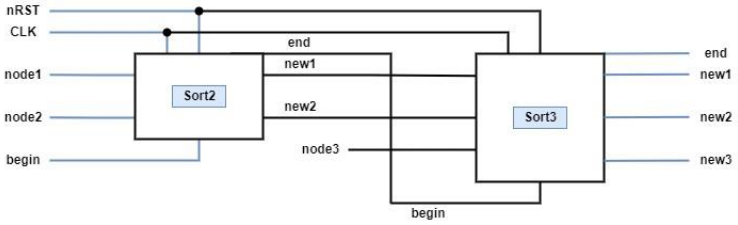
Chức năng: Mô-đun sắp xếp bong bóng, sắp xếp trọng số của 10 số

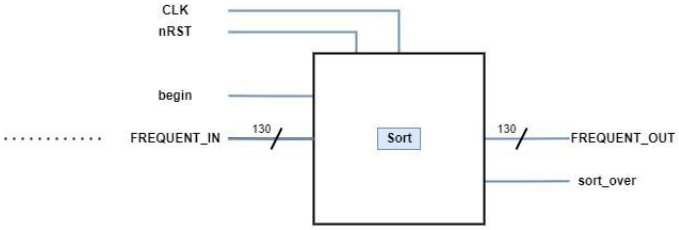
Đầu vào và đầu ra: Đầu vào là trọng số của 10 dữ liệu và đầu ra là trọng lượng của 10 dữ liệu đã được sắp xếp.

Thiết kế logic: Đầu tiên sắp xếp dữ liệu 9, sau đó chèn dữ liệu thứ 10 vào 9 số đầu tiên bằng cách so sánh. Việc sắp xếp dữ liệu 9 tương tự như việc sắp xếp dữ liệu 10, do đó sắp xếp đệ quy. Cuối cùng là sắp xếp hai dữ liệu.







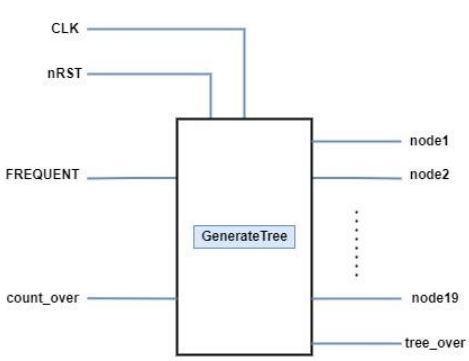


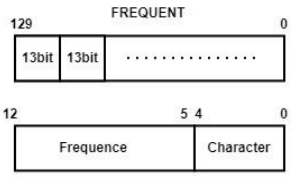
5.4. Module GenerateTree :[8]

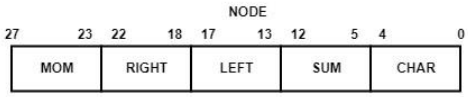
Chức năng: chịu trách nhiệm xây dựng số Huffman

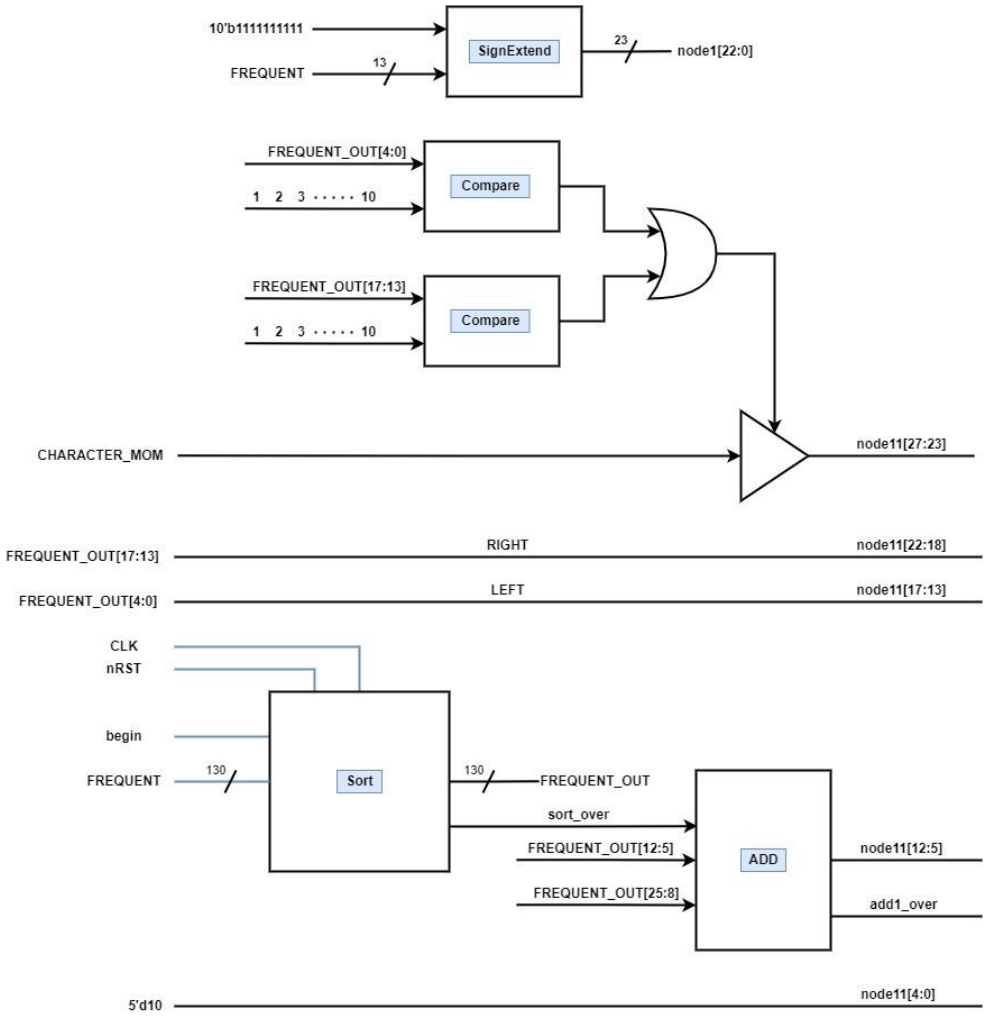
Đầu vào và đầu ra: Đầu vào là trọng số của mỗi số sau khi đếm và đầu ra là thông tin của 19 nút của cây Huffman. Mỗi nút bao gồm số nút của chính nó, trọng số, số nút bên trái cây con, số nút của cây con bên phải và số nút mẹ..

Thiết kế logic: Đầu tiên hãy sắp xếp trọng lượng của mỗi số. Sau đó, chọn hai nút thấp nhất từ các trọng số đã được sắp xếp, thực hiện phép cộng và tính trọng số của nút mới. Số lượng nút mới bắt đầu từ 10 và tăng lên 19, xác định các cây con bên trái và bên phải của nút mới và hai nút có trọng số thấp nhất tương ứng. Chỉ định rằng nút cha của nút con là số của nút mới. Sau đó sắp xếp các trọng số mới ... Sau 9 thao tác sắp xếp và thêm vào, cây Huffman được xây dựng.







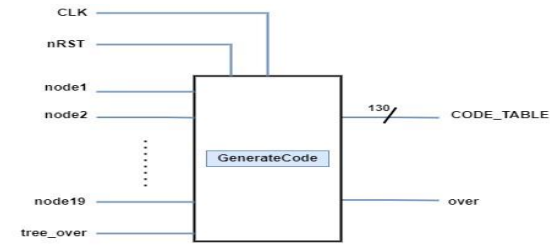


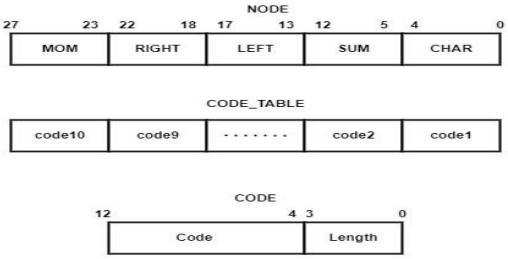
5.5. Module GenerateCode : [8]

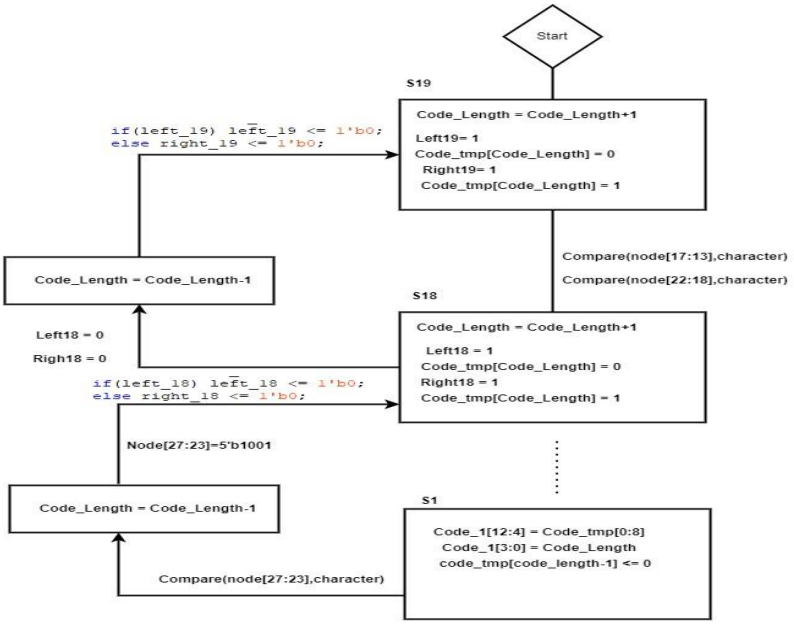
Chức năng: chịu trách nhiệm đánh số các số 0 ~ 9 theo cây Huffman đã tạo

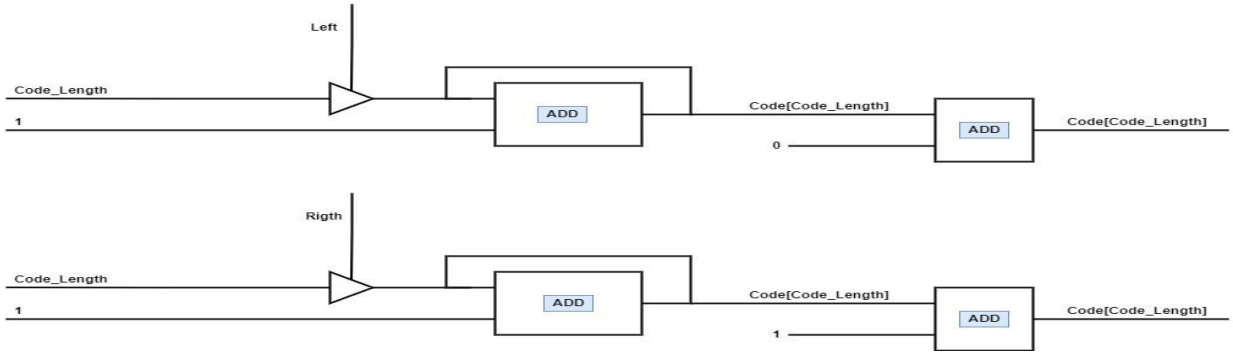
Đầu vào và đầu ra: Đầu vào là thông tin của 19 nút của cây Huffman và đầu ra là bảng mã của 10 số.

Thiết kế logic: bắt đầu từ nút thứ 19 của nút gốc, trạng thái được chuyển sang, mỗi nút tương ứng với một trạng thái, tổng cộng có 19 nút. Đầu tiên lướt qua cây con bên trái của nó, chuyển sang trạng thái tiếp theo và ghi lại độ dài của mã hóa, mã hóa được đặt thành 0. Ở trạng thái tiếp theo, nếu đó là nút lá, hãy lưu mã hiện tại, quay lại nút cha và đặt cây con bên trái của nút cha đã được duyệt qua. Sau đó, đi ngang qua cây con bên phải. Nếu cả hai cây con trái và phải đã được duyệt qua, hãy quay lại nút trước đó và đặt cờ đã duyệt; nếu trạng thái tiếp theo không phải là nút lá, hãy duyệt qua cây con bên trái trước, tăng độ dài mã hóa và đặt Được mã hóa là 0. Nếu cây con bên trái đã được mã hóa, hãy chuyển qua cây con bên phải.







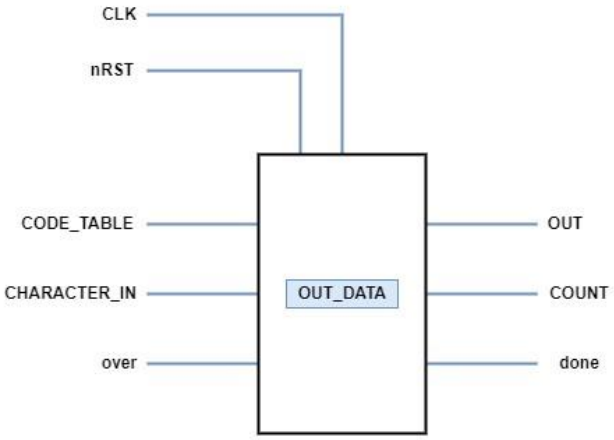


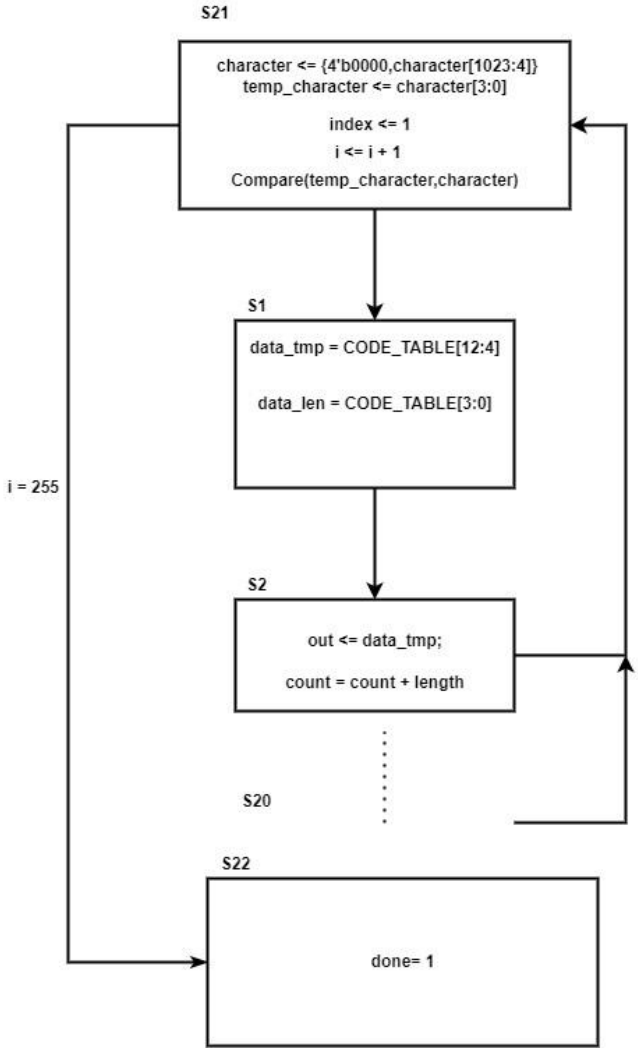
5.6. Module OutData :[8]

Chức năng: mô-đun đầu ra dữ liệu

Đầu vào và đầu ra: Đầu vào là mã hóa và độ dài mã hóa của 10 chữ số, và đầu ra là mã hóa 1024 bit và độ dài hiệu dụng của mã hóa.

Thiết kế logic: đầu tiên xuất ra lần lượt 10 mã kỹ thuật số, sau đó xuất ra 256 mã dữ liệu

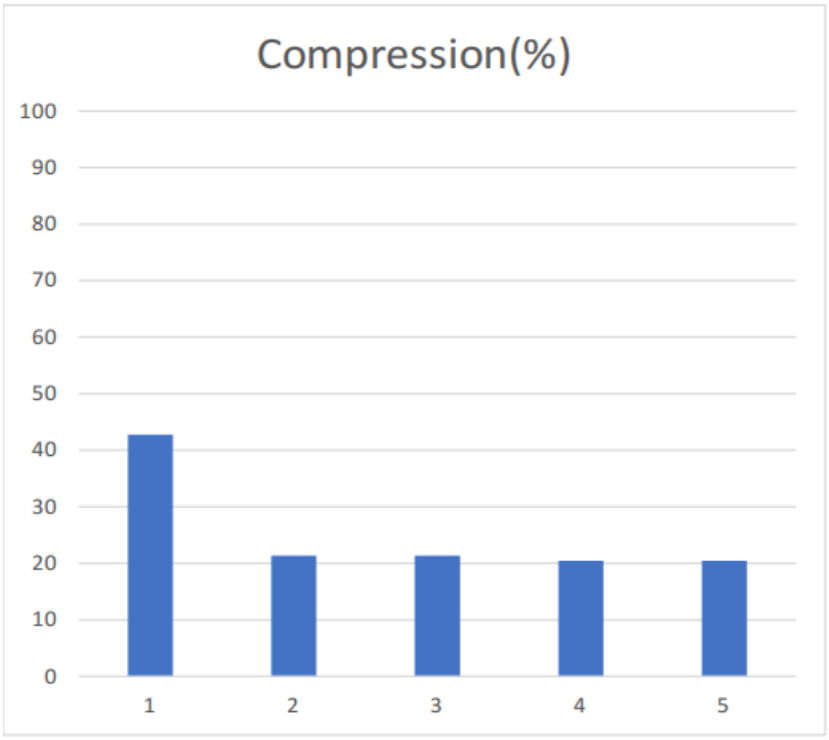


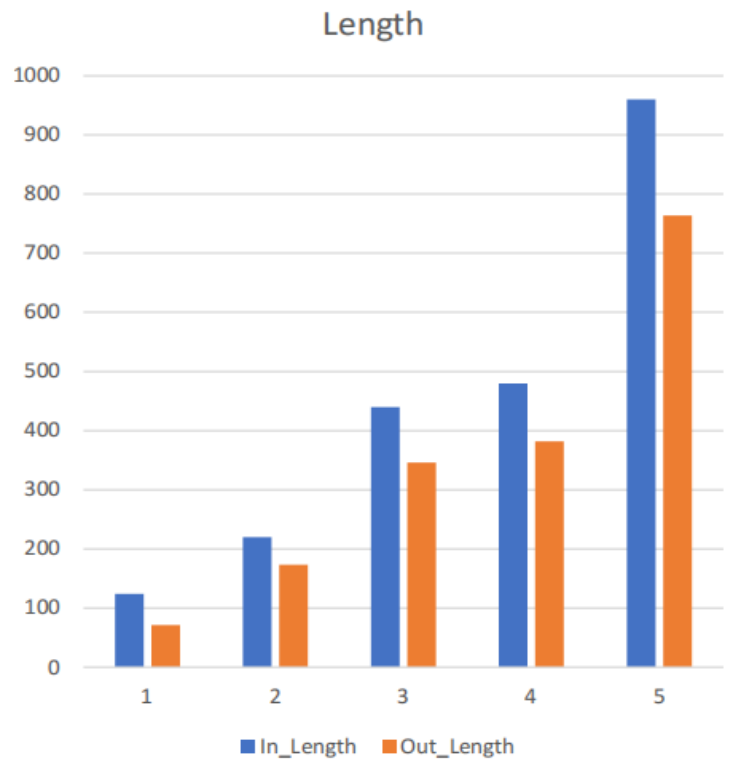


CHƯƠNG 6: Kết

6.1. Kết quả :

$$\text{Compression} = \frac{Lengthin - Lengthout * 100}{\text{Lengthin}}$$





Compilation Report :

Compilation Report - a

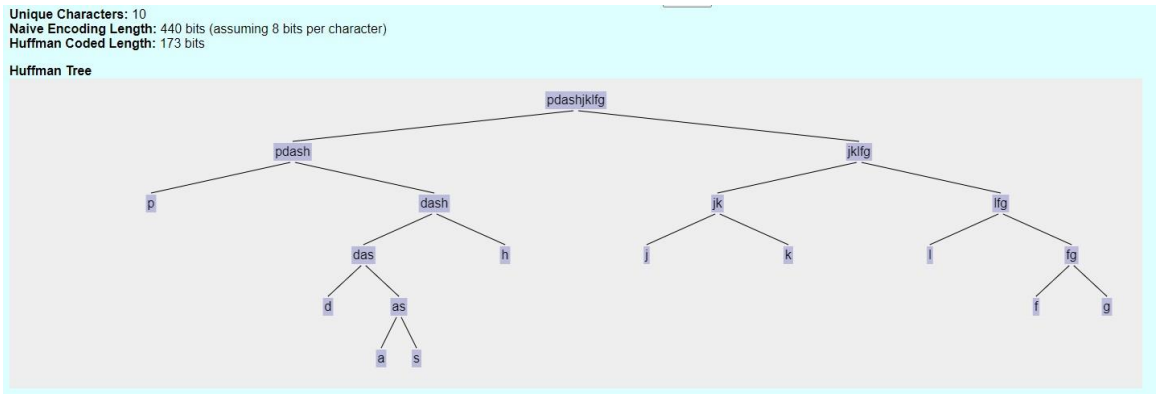
Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	157.38 MHz	157.38 MHz	CLK	

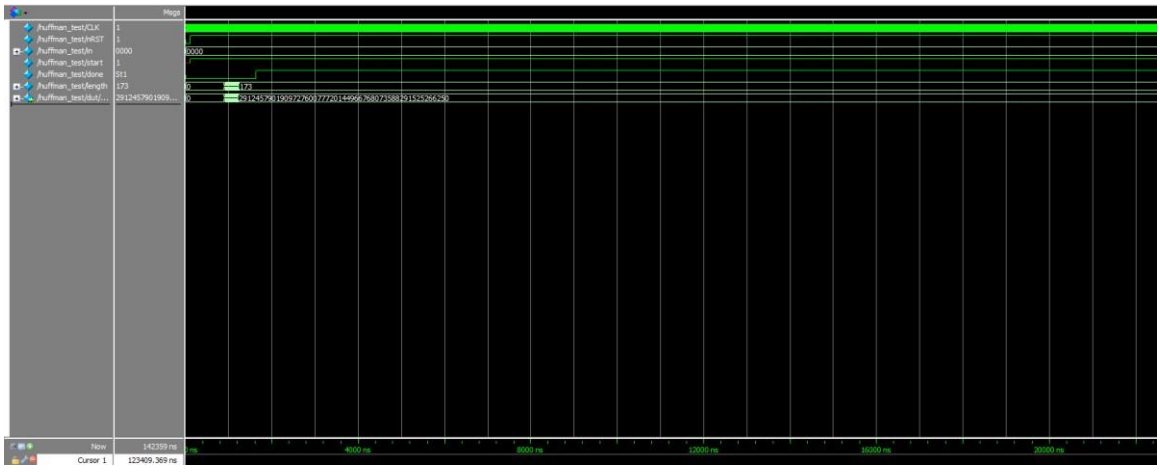
So sánh simulation hardware và software:

Test 1 :

Software:

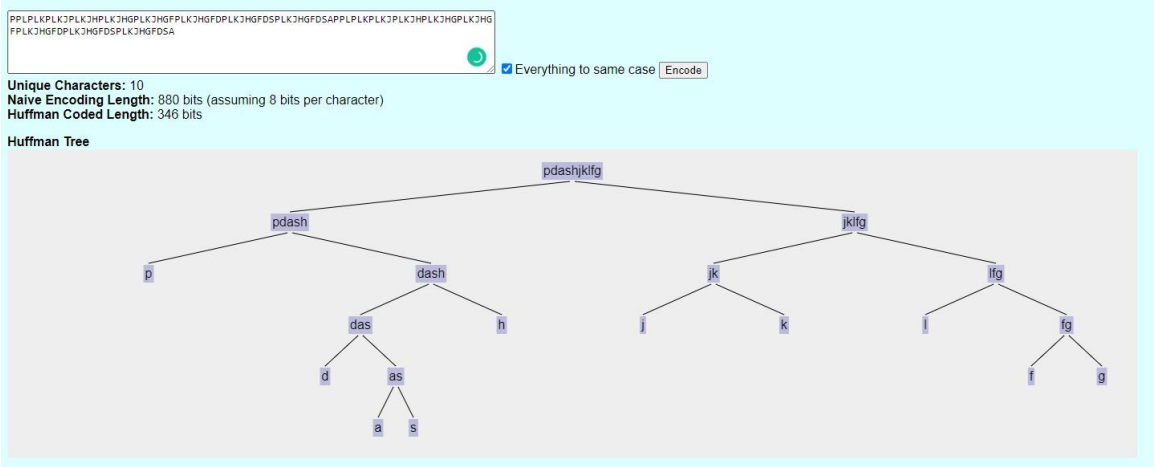


Hardware:

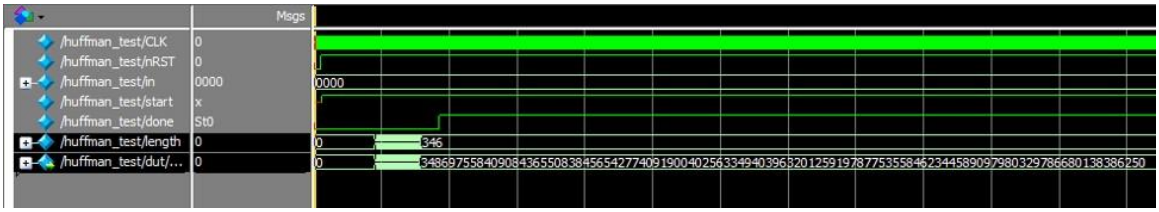


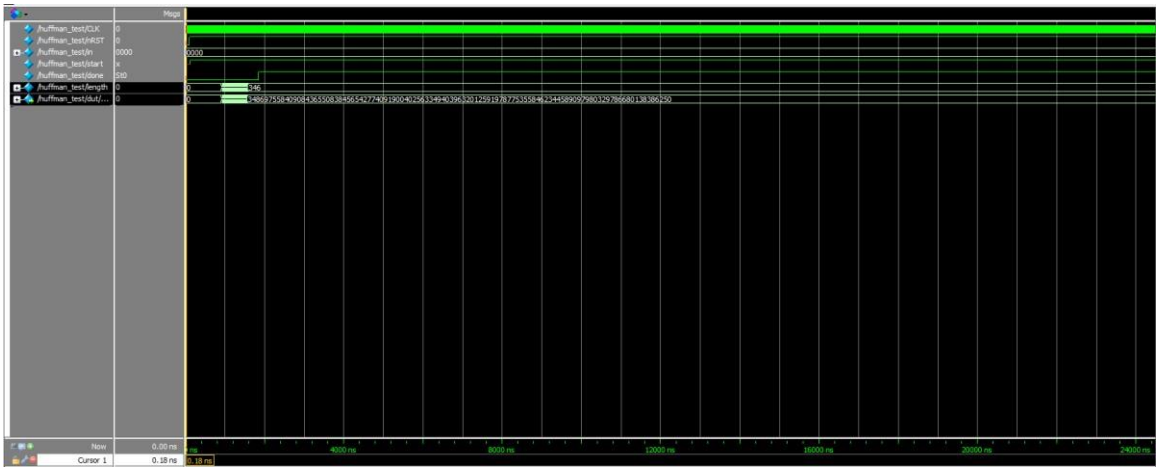
Test 2 :

Software:



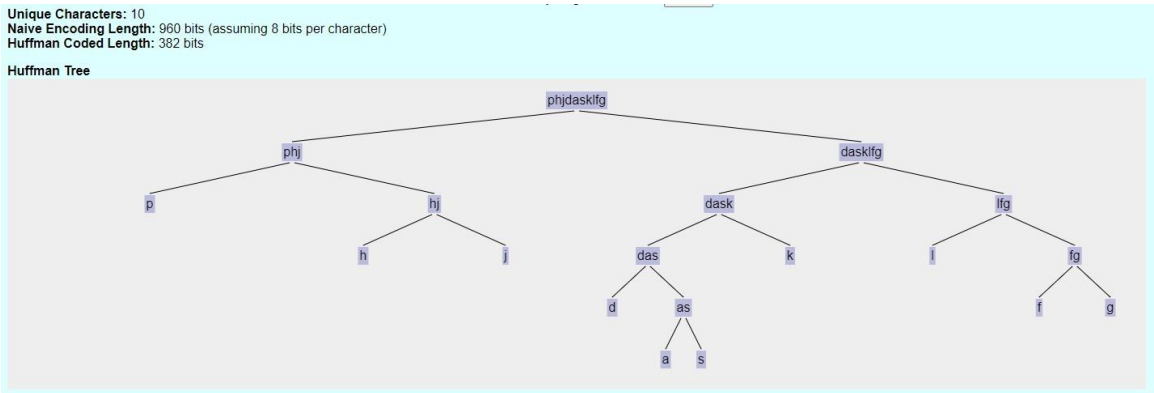
Hardware:



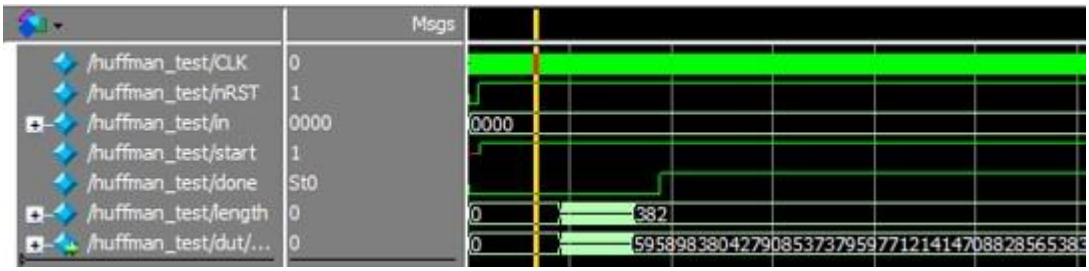


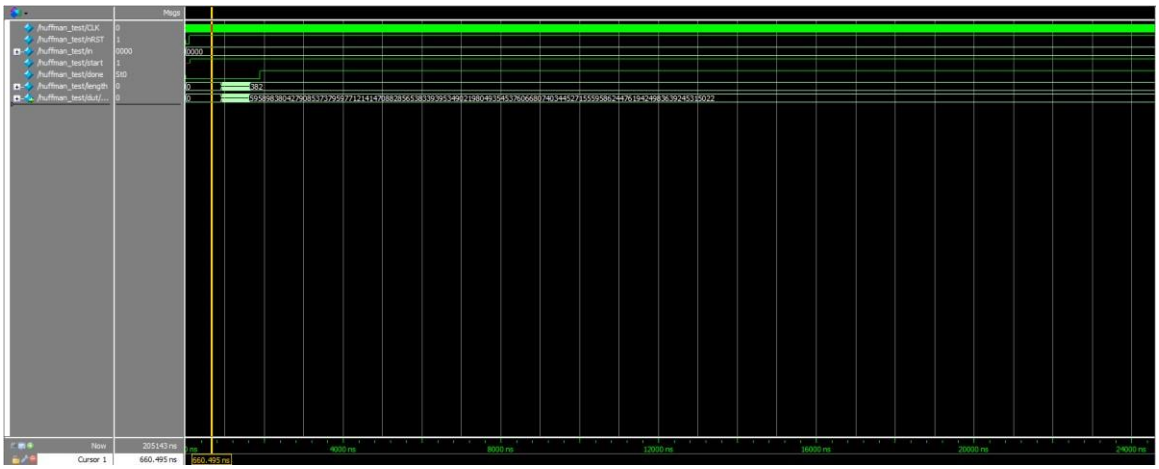
Test 3 :

Software:



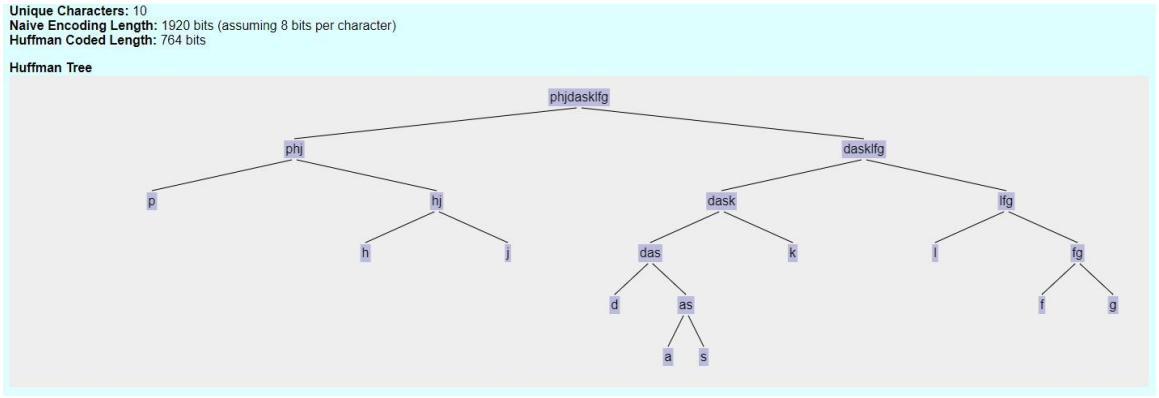
Hardware:



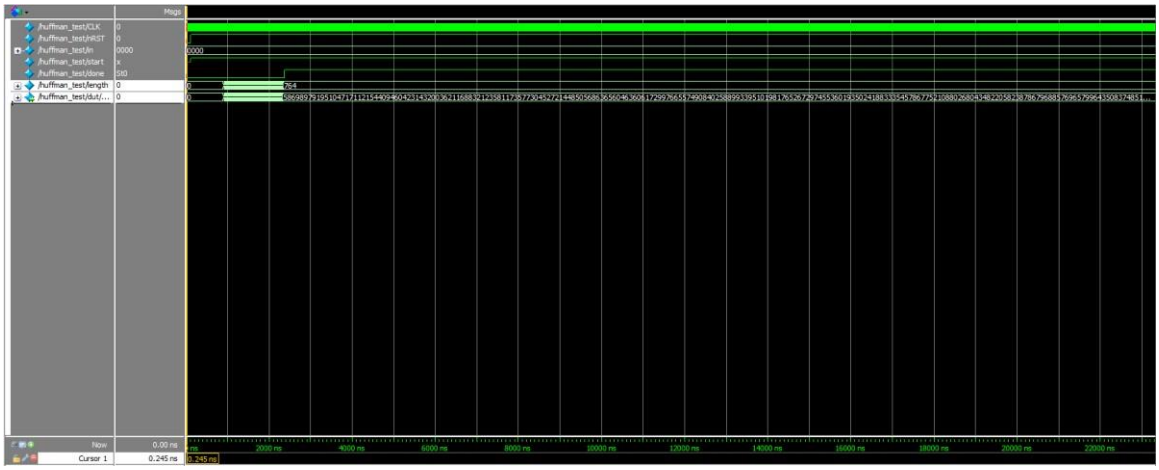


Test 4 :

Software :



Hardware :



6.2. Kết luận:

Hoàn thành:

- Hiểu được cách thức hoạt động của giải thuật HuffmanCoding.
- Hiện thực được thuật toán bằng hardware.

Hướng phát triển:

- Xây dựng mạch kích thước lớn hơn, với khả năng nén được ảnh với số lượng điểm ảnh khác nhau lớn hơn 10. - Tối ưu thiết kế.

KẾT BÁO CÁO BÀI TẬP

Xin chân thành cảm ơn đến các tác giả đã cung cấp tài những thông tin trong suốt bài làm của em. Sau đây là những tài liệu mà em đã tham khảo trong quá trình làm bài.

REFERENCE

[1]Jasmi, R. P., Perumal, B., & Rajasekaran, M. P. (2015, January). Comparison of image compression techniques using huffman coding, DWT and fractal algorithm. In 2015 International Conference on Computer Communication and Informatics (ICCCI) (pp. 1-5). IEEE. https://ieeexplore.ieee.org/document/7218137

[2]https://www.datavideo.com/eu/article/412/what-are-8-bit-10-bit-12-bit-4-4-4-4-2-2and-4-2-0

[3]https://en.wikipedia.org/wiki/Color_depth

[4]A NEW LOSSLESS METHOD OF IMAGE COMPRESSION AND DECOMPRESSION USING HUFFMAN CODING TECHNIQUES. Journal of Theoretical and Applied Information Technology © 2005 - 2010 JATIT. All rights reserved. JAGADISH H. PUJAR, LOHIT M. KADLASKAR. http://www.jatit.org/volumes/research-papers/Vol15No1/3Vol15No1.pdf

[5]Cấu trúc dữ liệu và giải thuật -ĐH Khoa Học Tự Nhiên HCM - TS. Bùi Tiến Lên https://cuuduongthancong.com/sjdt/cau-truc-du-lieu-va-giai-thuat/bui-tien-len/dh-khoahoc-tu-nhien-hcm

[6] Wang, H., Babacan, S. D., & Sayood, K. (2007). Lossless hyperspectral-image compression using context-based conditional average. IEEE transactions on geoscience and remote sensing, 45(12), 4187-193. https://ieeexplore.ieee.org/document/4378564 [7] Buro, M. (1993). On the maximum length of Huffman codes. Information processing letters, 45(5), 219-223. https://www.sciencedirect.com/science/article/abs/pii/002001909390207P [8]Lee, Taeyeon, and Jaehong Park. "Design and implementation of static Huffman encoding hardware using a parallel shifting algorithm." IEEE Transactions on Nuclear Science 51.5 (2004): 2073-2080. https://ieeexplore.ieee.org/document/1344287