BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN THIẾT KẾ HỆ THỐNG VÀ VI MẠCH TÍCH HỢP

PROJECTS CUỐI KỲ

****

HAMMING ENCODER AND DECODER

****

MÃ MÔN HỌC: ISCD336764_22_2_11
LỚP: Thứ 5 (Tiết 3-5)
GVHD: TS.ĐỖ DUY TÂN

SVTH: Phan Thị Châu Pha 21161170
Phan Thị Miên 21161406
Lê Thị Thi 21161194
Lê Như Tuân 21161418
Nguyễn Phạm Anh Tùng 21161420

Tp. Hồ Chí Minh, Tháng 5 năm 2023

DANH SÁCH THÀNH VIÊN THAM GIA VIẾT BÁO CÁO

HỌC KÌ II NĂM HỌC 2022-2023

Nhóm: (Lớp thứ 5 – tiết 3-5)

Tên đề tài: Hamming encoder and decoder (both modules)

STT	HỌ VÀ TÊN SINH VIÊN	MÃ SỐ SINH VIÊN	TỈ LỆ % HOÀN THÀNH
1	Phan Thị Châu Pha	21161170	100%
2	Phan Thị Miên	21161406	100%
3	Lê Thị Thi	21161194	100%
4	Lê Như Tuấn	21161418	100%
5	Nguyễn Phạm Anh Tùng	21161420	100%

Ghi chú:

- Tỉ lệ % = 100%: Mức độ phần trăm của từng sinh viên tham gia
- Trưởng nhóm: Lê Như Tuấn
- SĐT: 0352769212

Nhận xét của giáo viên:

.....

.....

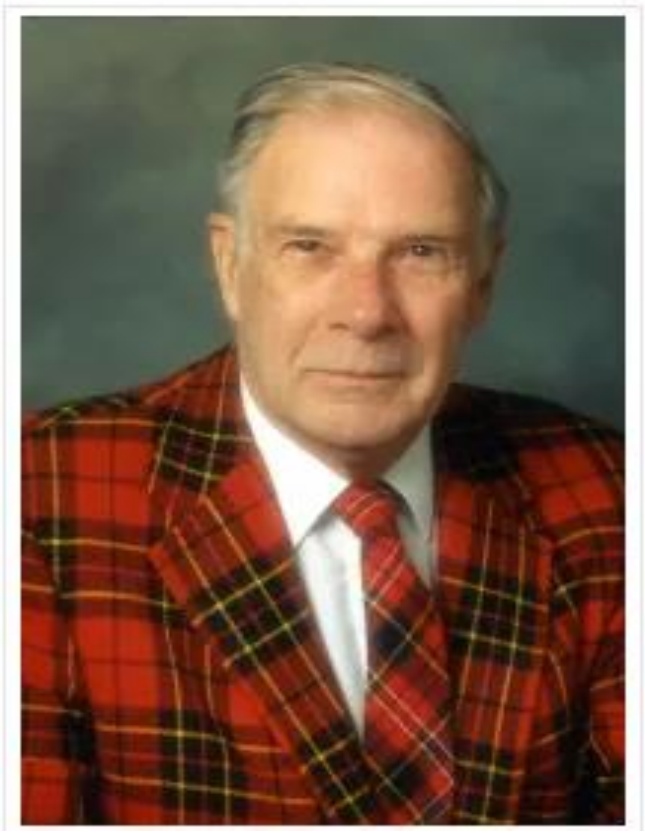
.....

.....

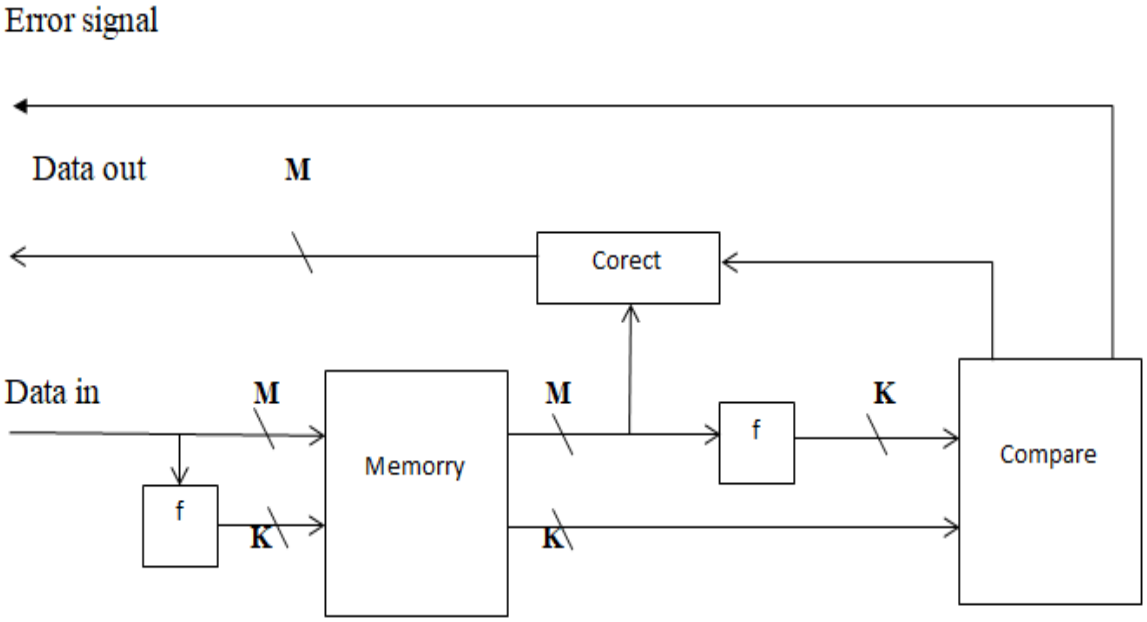
Ngày tháng năm 2023

Giáo viên chấm điểm

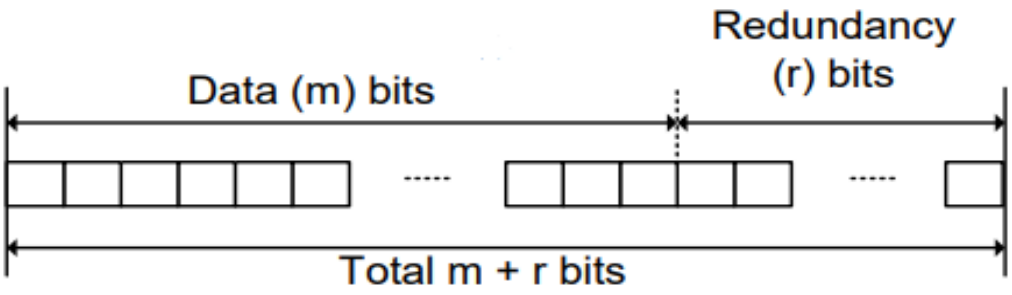
PHỤ LỤC HÌNH ẢNH



Hình 1 : Richard Wesley Hamming (1915- 1998)

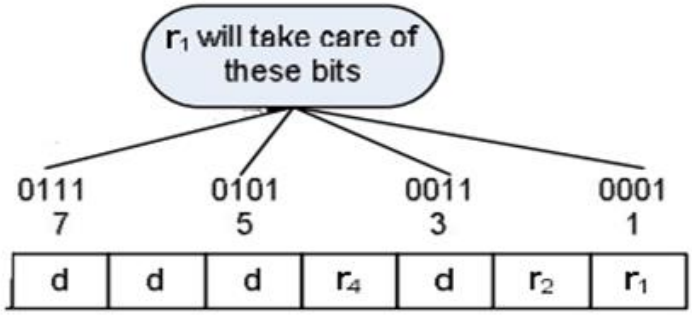


Hình 2: Error – correcting Code function

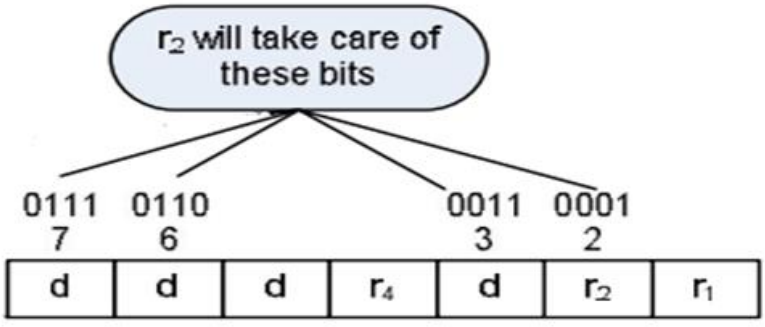


Hình 3: Độ dài của mã

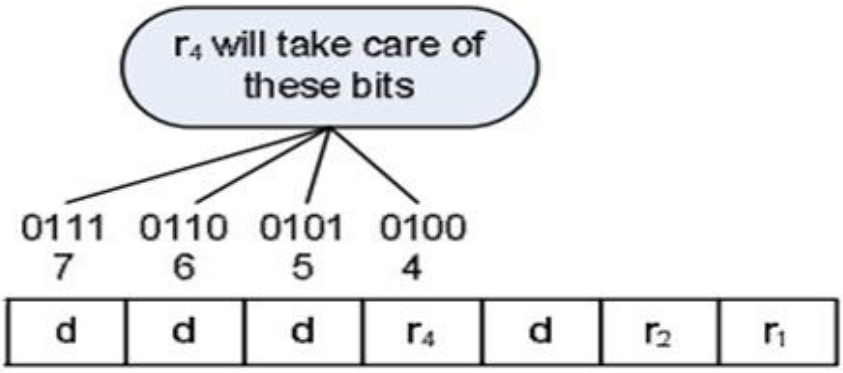
nhị phân có 1 trong vị trí thứ hai bên phải và tiếp tục như vẽ trong hình sau:



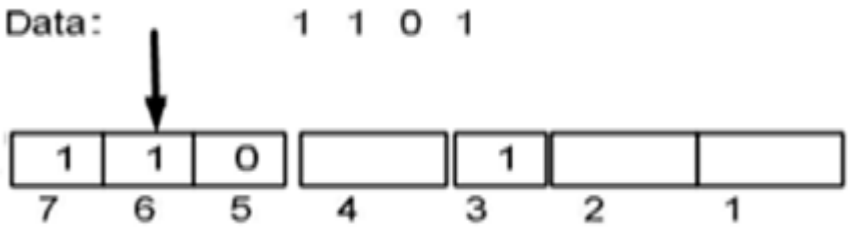
Hình 4: Cách biểu diễn bit r_1



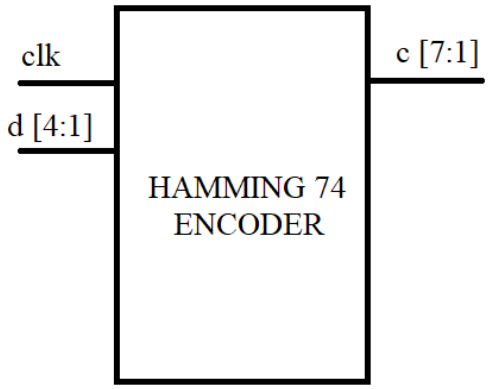
Hình 5: Cách biểu diễn bit r_2



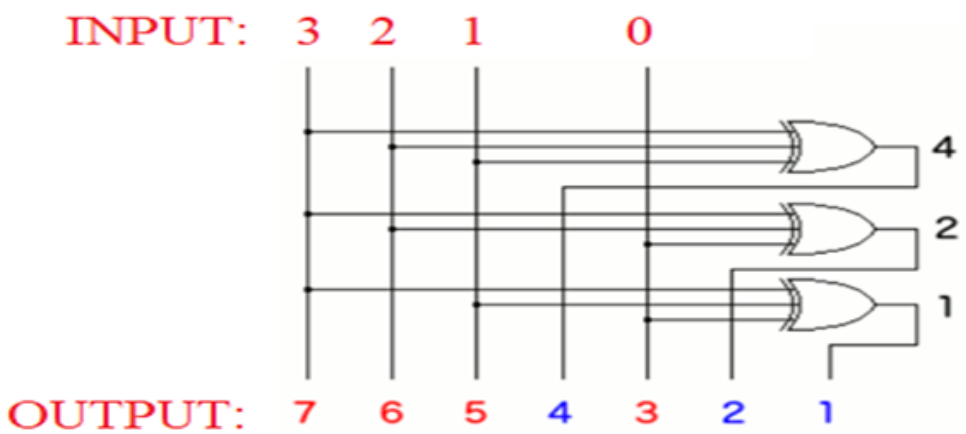
Hình 6: Cách biểu diễn bit r_4



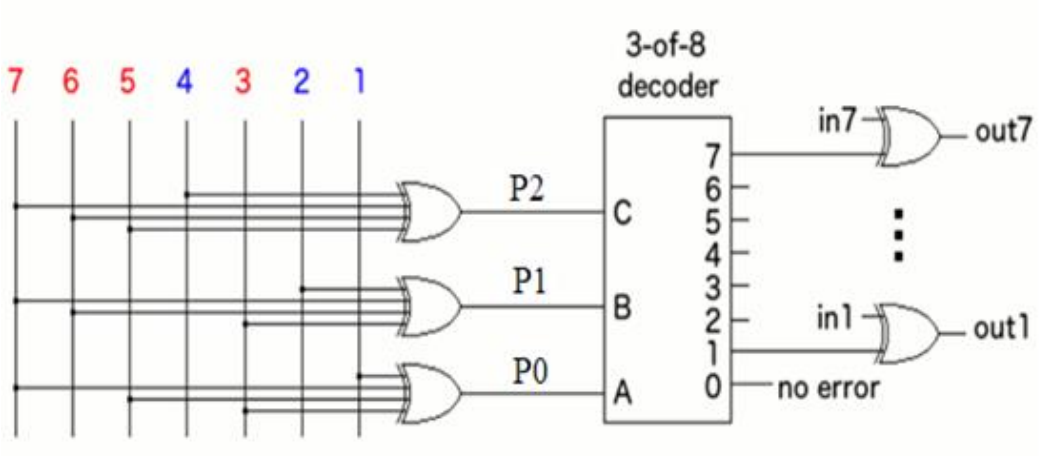
Hình 7: Đặt mỗi bit của ký tự gốc vào vị trí thích hợp



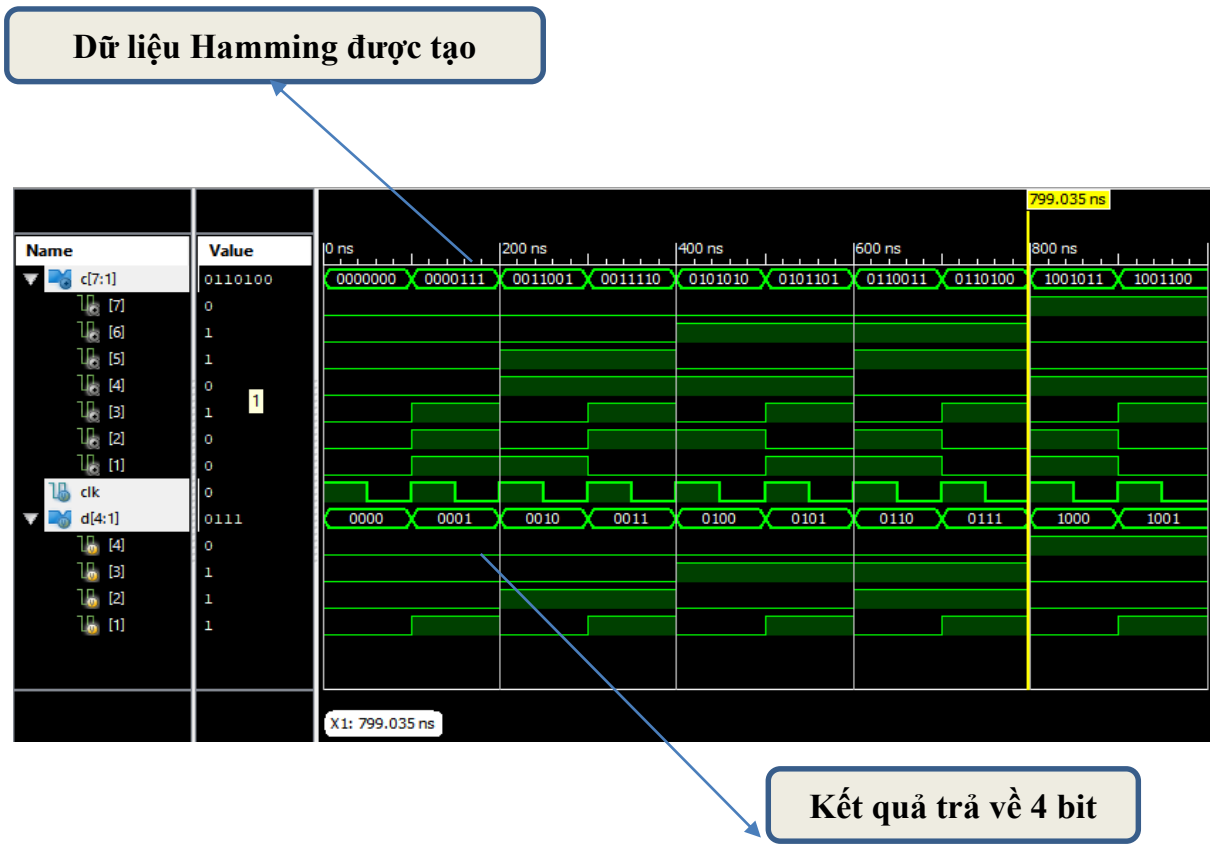
Hình 8: Khối hamming (7,4) encoder



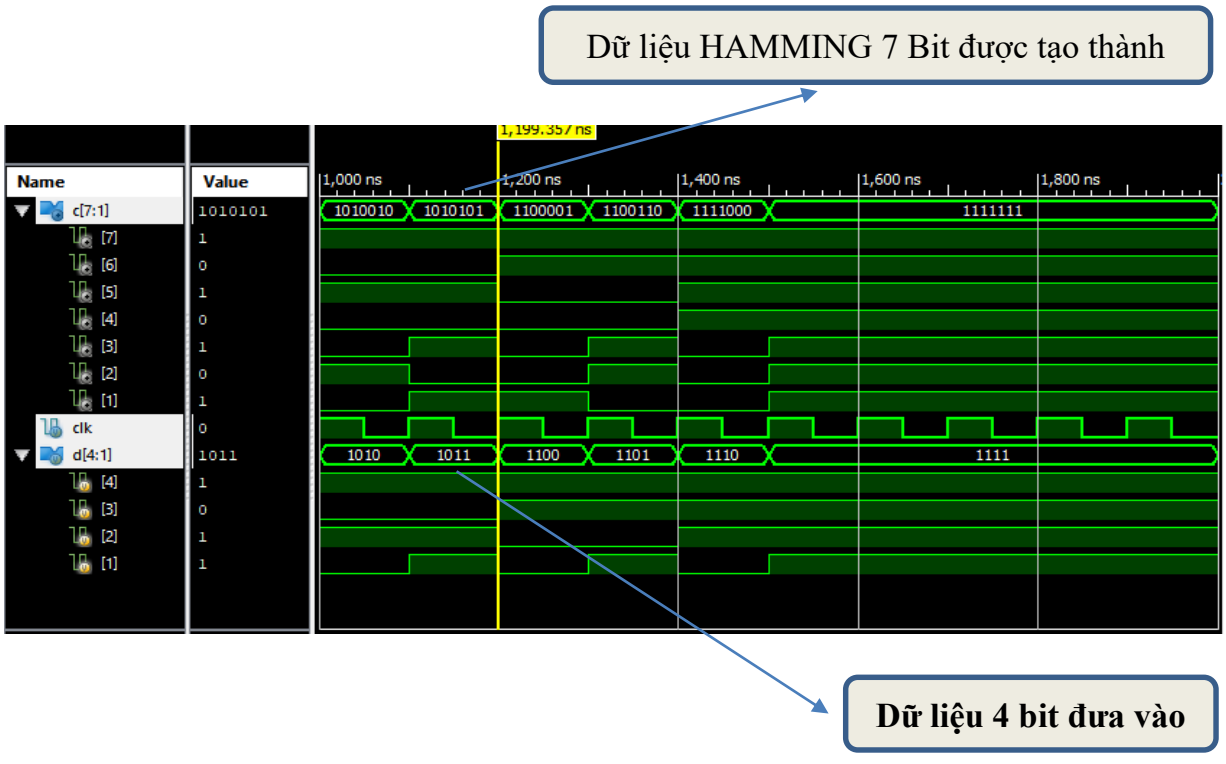
Hình 9: Sơ đồ scientific hamming (7,4) encoder



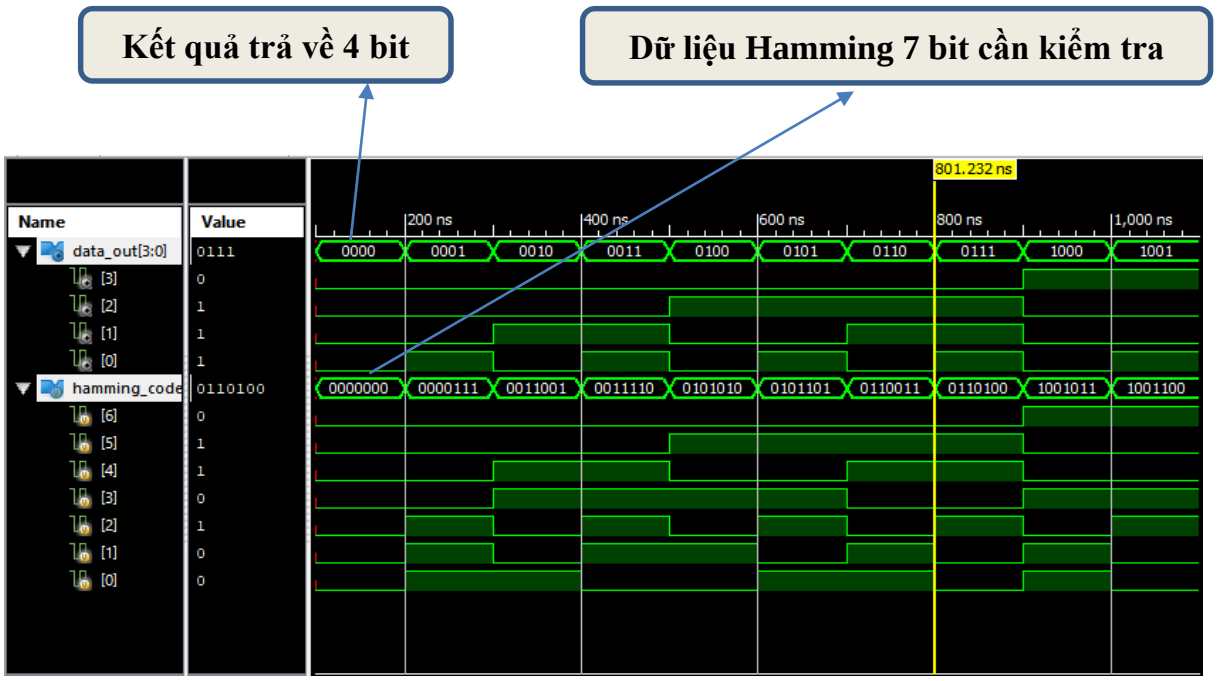
Hình 10: Sơ đồ scientific hamming (7,4) Decoder



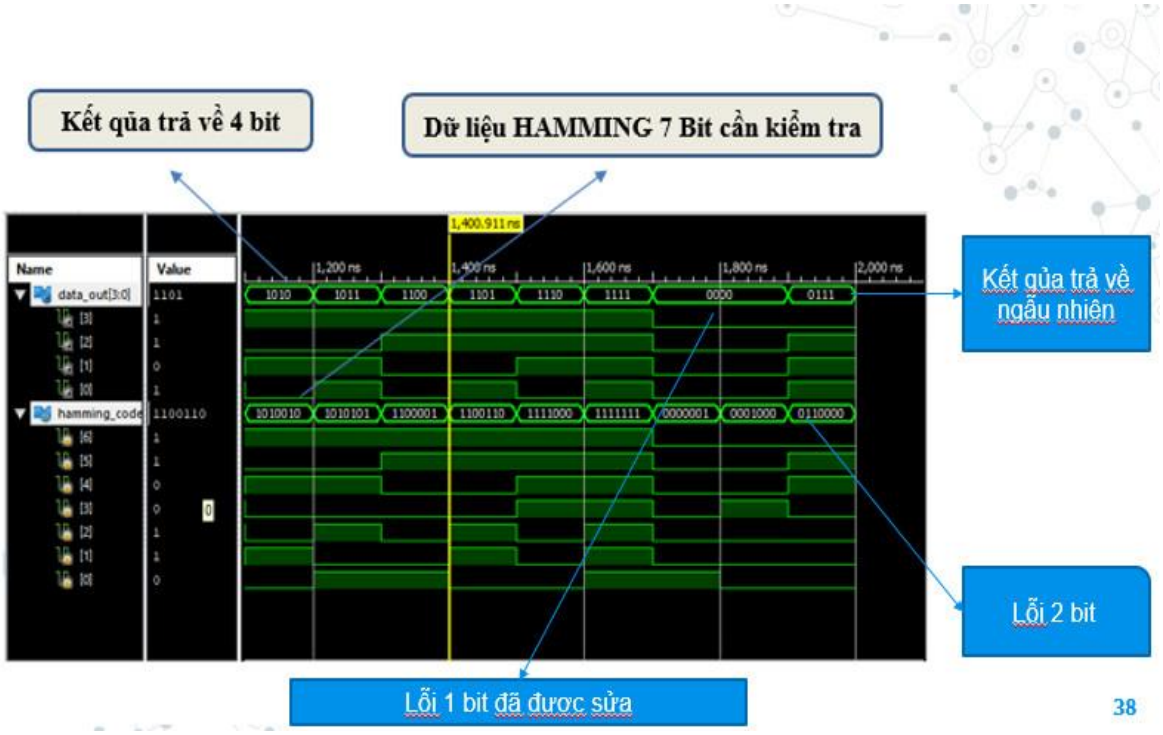
Hình 11: Kết quả mô phỏng testcase 1- 10



Hình 12: Kết quả mô phỏng testcase 11-16



Hình 13: Kết quả mô phỏng testcase 1-10



Hình 14: Kết quả mô phỏng testcase 11-1

MỤC LỤC

LỜI MỞ ĐẦU

Ngày nay công nghệ thông tin đang ngày một phát triển, việc liên lạc cũng như trao đổi thông tin với nhau dường như đã dần được số hóa .Trong kỹ thuật truyền dẫn thông tin, mã hóa kênh có vai trò vô cùng quan trọng, nhằm giảm xác suất sai thông tin do kênh truyền gây ra, giảm công suất phát, tiết kiệm năng lượng, thuận lợi hơn cho việc bảo mật, trải phổ và tăng độ chính xác của thông tin nhận. Mã hóa chập và mã hóa khối là hai dạng chính của mã hóa kênh truyền. Trong mã hóa khối bao gồm nhiều loại mã khác nhau, phần lớn các họ mã khối trước đây còn tồn tại nhiều mặt hạn chế như để giảm lượng tính toán và tăng tỷ lệ mã hóa thì phải đánh đổi chất lượng giải mã hoặc để đạt được chất lượng như mong muốn thì phải tăng độ phức tạp tính toán cũng như giảm tỷ lệ mã hóa.

Mã Hamming là một thuật toán mã hóa và kiểm tra lỗi được phát minh bởi Richard W. Hamming vào năm 1950. Thuật toán này hoạt động dựa trên nguyên lý thêm các bit kiểm tra vào dữ liệu ban đầu để kiểm tra lỗi và sửa lỗi nếu có. So với các loại mã khác thuộc loại mã khối, mã hamming có thể xem như là vượt trội khi nó có thể phát hiện và sửa được lỗi 1 bit, đồng thời có thể phát hiện được lỗi nhiều bit. Nhờ tính chất đơn giản của thuật toán mã hóa và giải mã, mã Hamming được ứng dụng khá rộng rãi trong các hệ thống truyền tin số và lưu trữ dữ liệu.

Trong bài báo cáo này, chúng ta sẽ tìm hiểu về cách hoạt động của mã Hamming thông qua các cơ sở lí thuyết và mô phỏng dựa trên code mẫu.

Mặc dù có tìm hiểu từ nhiều nguồn thông tin khác nhau, tuy nhiên trong quá trình làm báo cáo sẽ có những sai sót khó tránh khỏi, chúng em rất mong nhận được những nhận xét, góp ý của thầy và các bạn để chúng em có thể rút kinh nghiệm và hoàn thiện hơn bài bài báo cáo của mình. Chúng em xin chân thành cảm ơn!

CHƯƠNG 1: TỔNG QUAN

1. Mục tiêu

Tìm hiểu những kiến thức cơ bản về đặc điểm và cách thức hoạt động của mã Hamming, giúp chúng ta hiểu rõ hơn và có thể vận dụng chúng vào trong quá trình thực nghiệm.

Tìm hiểu các thuật toán sử dụng để tạo ra mã Hamming, cách áp dụng mã Hamming vào truyền thông dữ liệu số nhằm áp dụng cải thiện chất lượng truyền thông dữ liệu và tăng độ bảo mật của các hệ thống truyền thông.

Đánh giá hiệu suất của mã Hamming trong việc phát hiện và sửa lỗi trong truyền thông số.

2. Nội dung nghiên cứu

Đề tài tập trung tìm hiểu về khái niệm của mã Hamming, đặc điểm, cách tạo mã, phát hiện lỗi cũng như cách sửa lỗi.

Thực hiện thiết kế, mô phỏng, nhận xét đánh giá thông qua code mẫu và phần mềm mô phỏng.

3. Bố cục

Chương 1: Tổng quan

Chương 2: Cơ sở lí thuyết về mã Hamming

Chương 3: Tìm hiểu và thiết kế khối Hamming (7,4) encoder và hamming (7,4) decoder

Chương 4: Đánh giá qua test bench

Chương 5: Kết luận và hướng phát triển

4. Giới hạn đề tài

Đề tài tập trung nghiên cứu lí thuyết, phân tích và tổng hợp những kiến thức cơ bản về mã Hamming.

Thực hiện thiết kế, mô phỏng và đánh giá thông qua code mẫu và phần mềm mô phỏng Xilinx ISE.

Nghiên cứu cách triển khai mã hóa và giải mã Hamming trên phần mềm.

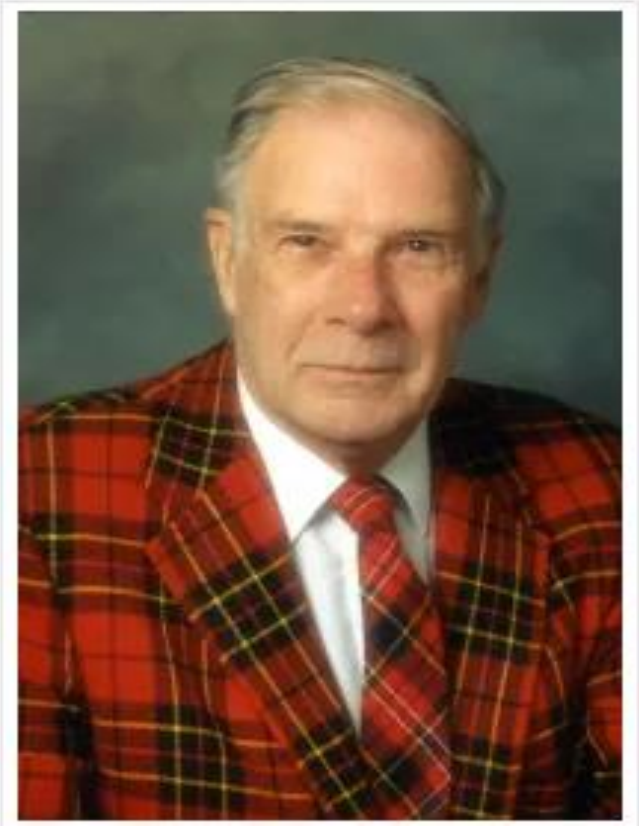
CHƯƠNG 2: CƠ SỞ LÍ THUYẾT VỀ HAMMING CODE

Hamming code là một thuật toán mã hóa và phát hiện lỗi được phát minh bởi Richard Hamming vào những năm 1950. Nó được sử dụng trong việc truyền thông số và lưu trữ để giảm thiểu lỗi truyền thông.

1.Khái niệm

Mã Hamming là một mã sửa lỗi tuyến tính (linear error-correcting code), được đặt tên theo tên của người phát minh ra nó là Richard Wesley Hamming. Mã Hamming có thể phát hiện một bit hoặc hai bit bị lỗi (single and double-bit errors). Mã Hamming còn có thể sửa các lỗi do một bit bị sai gây ra. Ngược lại với mã của ông, mã chẵn lẻ (parity code) đơn giản vừa không có khả năng phát hiện các lỗi khi 2 bit cùng một lúc bị hoán vị (0 thành 1 và ngược lại), vừa không thể giúp để sửa được các lỗi mà nó phát hiện thấy.

2.Lịch sử



Hình 1 : **Richard Wesley Hamming (1915- 1998)**

Vào những năm của thập niên kỷ 1940, Hamming làm việc tại Bell Labs trên máy tính Bell Model V, một máy điện cơ (electromechanical) dùng rơ-le (relay-based), với

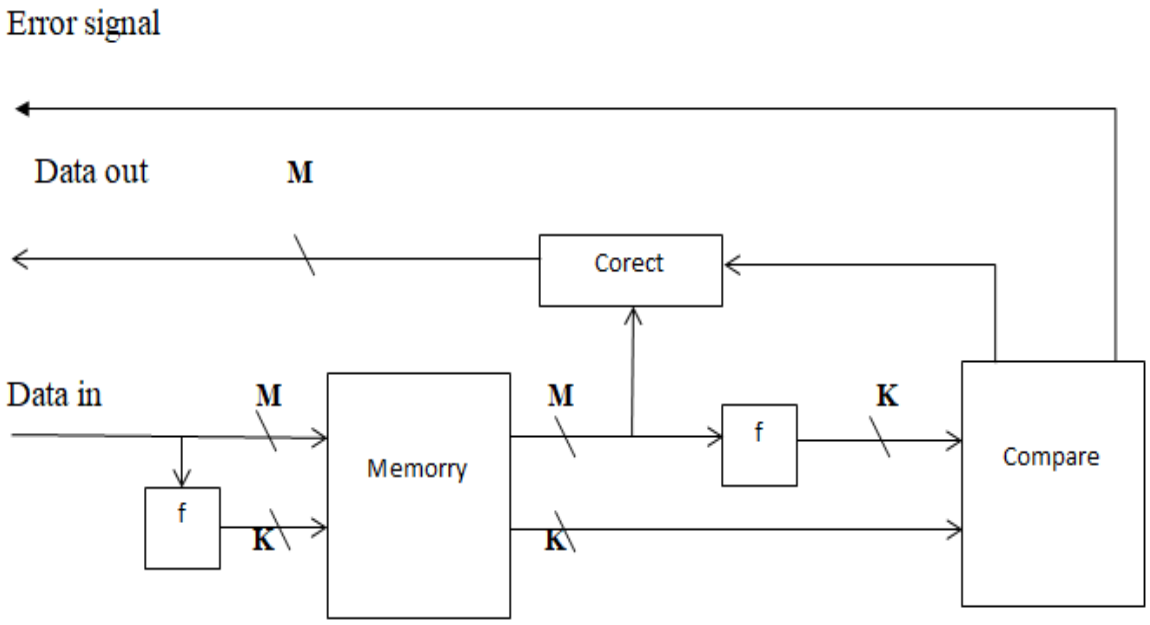
tốc độ rất chậm, mấy giây đồng hồ một chu kỳ máy. Nhập liệu được cho vào máy bằng các thẻ đục lỗ (punch cards), và hầu như máy luôn luôn gây ra các lỗi trong khi đọc. Những ngày làm việc trong tuần, những mã đặc biệt được dùng để tìm ra lỗi và mỗi khi tìm được, nó sẽ nhấp nháy đèn báo hiệu, báo cho người điều khiển biết để họ sửa, điều chỉnh máy lại. Trong thời gian ngoài giờ làm việc hoặc trong những ngày cuối tuần, khi người điều khiển máy không có mặt ở đó, mỗi khi mà lỗi xảy ra, máy tính tự động bỏ qua chương trình đương chạy và chuyển sang những công việc khác.

Richard Hamming thường làm việc trong những ngày cuối tuần và ông càng ngày càng trở nên bực tức mỗi khi phải khởi động lại các chương trình ứng dụng từ đầu, do chất lượng kém, không đáng tin cậy (unreliability) của bộ máy đọc các thẻ đục lỗ. Mấy năm tiếp theo đó, ông dồn tâm lực vào việc xây dựng hàng loạt các thuật toán có hiệu quả cao để giải quyết vấn đề sửa lỗi này. Năm 1950, ông đã công bố một phương pháp được biết là Mã Hamming. Một số chương trình ứng dụng hiện thời vẫn còn đang sử dụng mã này của ông.

3.Đặc điểm

- Hamming code sử dụng các bit kiểm tra để phát hiện và sửa chữa các lỗi truyền thông. Các bit kiểm tra được tính toán dựa trên các bit dữ liệu được mã hóa. Để đảm bảo tính toàn vẹn của dữ liệu, các bit kiểm tra được thêm vào các khung dữ liệu trước khi chúng được gửi đi.
- Hamming code có thể được thực hiện bằng cách sử dụng các phép toán đơn giản như XOR và AND để tính toán các bit kiểm tra. Các bit kiểm tra này sẽ được sử dụng để kiểm tra tính toàn vẹn của khung dữ liệu khi nó được nhận.
- Một trong những đặc điểm quan trọng của Hamming code là khả năng sửa chữa các lỗi truyền thông. Nếu một bit bị sai lệch trong quá trình truyền thông, Hamming code có thể sửa chữa bit đó. Nếu có nhiều hơn một bit bị sai lệch, Hamming code sẽ phát hiện được lỗi và báo cho người nhận biết.

4.Cách thức thực hiện quá trình sửa lỗi



Hình 2: Error – correcting Code function

Hình minh họa về cách thức thực hiện quá trình sửa lỗi. Khi nào dữ liệu được ghi vào bộ nhớ, tính toán, miêu tả như một hàm f, được thực hiện trên dữ liệu để tạo ra một mã. Cả mã và dữ liệu được lưu trữ. Vì vậy một từ M bit dữ liệu được lưu và mã có chiều dài K bit sau đó kích thước thực tế của từ được lưu trữ là M+ K bit. Khi từ lưu trữ trước đó được đọc, mã sử dụng để phát hiện và sửa lỗi. Một bộ bit mã K mới được tạo ra từ các bit dữ liệu M và so sánh với các bit mã lấy được. So sánh này mang lại một trong ba kết quả:

- Không phát hiện lỗi. Các bit dữ liệu được tìm nạp sẽ được gửi đi.
- Phát hiện lỗi và có thể sửa lỗi. Các bit dữ liệu cộng các bit điều chỉnh lỗi được đưa đến bộ điều chỉnh, tạo ra một bộ điều chỉnh M bit được gửi đi.
- Phát hiện lỗi, nhưng không thể sửa lỗi. Tình trạng này được báo cáo. Mã hoạt động theo cách này được gọi là các mã sửa lỗi. Một mã được đặc trưng bởi số lỗi bit trong một từ mà nó có thể chính và phát hiện.

5.Khoảng cách Hamming

- Khoảng cách Hamming giữa 2 từ cùng độ dài : là số bit (ở cùng vị trí) khác nhau
- Khoảng cách Hamming tối thiểu: là khoảng cách nhỏ nhất giữa 2 cặp bất kỳ có trong một tập từ:

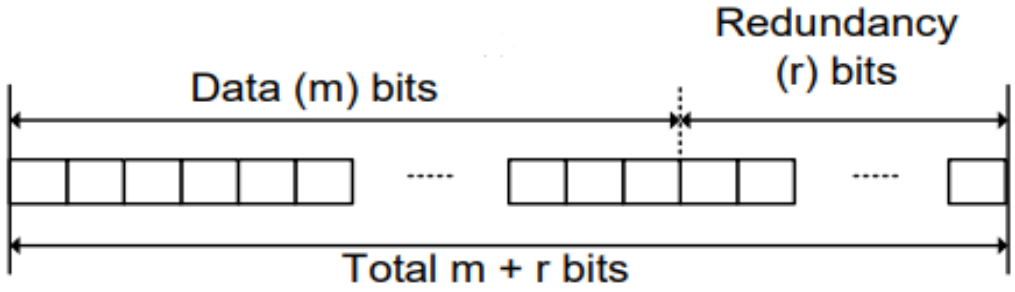
d(000, 011) là 2

Định lý 1: Để đảm bảo phát hiện lỗi của s bit thì khoảng cách Hamming cực tiểu trong các cặp từ mã phải là $d_{min} = s+1$

Định lý 2: để sửa t lỗi , khoảng cách Hamming cực tiểu trong các từ mã phải là $d_{min} = 2t +1$

6. Các bit dư

Để tính số bit dư (r) cần có để có thể sửa lỗi một số bit dữ liệu (m), ta cần tìm ra quan hệ giữa m và r. Trong hình sau cho thấy m bit dữ liệu và r bit dư. Độ dài của mã có được là m+r.



Hình 3: Độ dài của mã

Nếu tổng số các bit trong một đơn vị được truyền đi là m+r, thì r phải có khả năng chỉ ra ít nhất m+r+1 trạng thái khác nhau. Trong đó, một trạng thái là không có lỗi và m+r trạng thái chỉ thị vị trí của lỗi trong mỗi vị trí m+r.

Điều đó, tức là m+r+1 trạng thái phải được r bit phát hiện ra được; và r bit có chể chỉ được 2^r trạng thái khác nhau. Như thế, 2^r phải lớn hơn hay bằng m+r+1:

$$\mathbf{2^r \geq m+r+1.}$$

Giá trị của r có thể được xác định từ cách gắn vào trong giá trị của m (chiều dài ban đầu của đơn vị dữ liệu cần gởi đi).

Thí dụ, nếu giá trị của m là 4 (trường hợp 4 bit của mã ASCII), thì giá trị bé nhất của r cần thỏa mãn phương trình là 3:

$$\mathbf{2^r \geq 4+r+1;\ chọn\ r=3}$$

$$\mathbf{2^3 \geq 7+3+1.}$$

Một số khả năng của các giá trị m và r tương ứng:

Số lượng bit dữ liệu (m)	Số lượng bit dư (r)	Tổng số bit (m+r)
1	2	3
2	3	5
3	3	6
4	3	7
5	4	9
6	4	10
7	4	11

7. Mã Hamming (7,4)

Ngày nay, khi nói đến mã Hamming chúng ta thực ra là đang muốn nói đến mã (7,4) mà Richard Hamming công bố năm 1950. Với mỗi nhóm 4 bit dữ liệu, mã Hamming sẽ thêm 3 bit kiểm tra. Thuật toán (7, 4) của Hamming có thể dùng để sửa chữa bất cứ một bit lỗi nào, và phát hiện tất cả lỗi của 1 bit, và lỗi của 2 bit gây ra. Điều này đồng nghĩa với việc đối với tất cả các phương tiện truyền thông không có chùm lỗi đột phát (*burst errors*) xảy ra, mã (7, 4) của Hamming rất có hiệu quả (trừ phi phương tiện truyền thông có độ nhiễu rất cao thì nó mới có thể gây cho 2 bit trong số 7 bit truyền bị đảo lộn).

8. Định vị của các bit dư

7	6	5	4	3	2	1
d	d	d	r	d	r	r

Mã Hamming có thể được áp dụng vào đơn vị dữ liệu có chiều dài bất kỳ dùng quan hệ giữa dữ liệu và các bit dư đã được khảo sát trước đây.

Thí dụ, mã 4 bit ASCII cần có 3 bit dư được thêm vào phần cuối đơn vị dữ liệu hay phân bố vào bên trong các bit gốc. Các bit này được đặt ở các vị trí 0, 1, 2,…. (2n). Ta gọi các bit này lần lượt là r1, r2, r4.

Trong mã Hamming, mỗi bit r là bit VRC của một tổ hợp các bit dữ liệu; r1 là bit VRC của một tổ hợp bit; r2 là một bit trong một tổ hợp bit khác và cứ thế tiếp tục. Tổ hợp được dùng để tính toán mỗi giá trị trong ba bit r này trong chuỗi bảy bit được tính toán như sau:

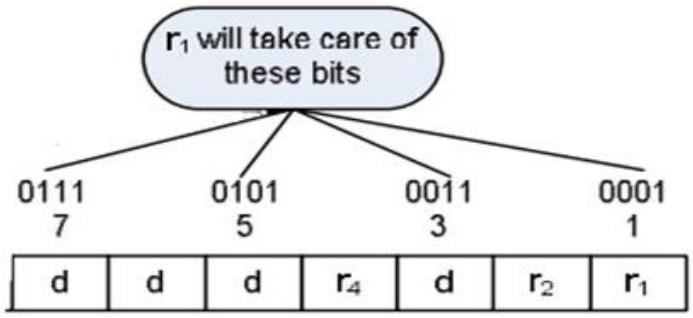
R1 (bit 1), 3, 5, 7; tổng số bit 1 là một số chẵn

R2 (bit 2), 3, 6, 7; tổng số bit 1 là một số chẵn

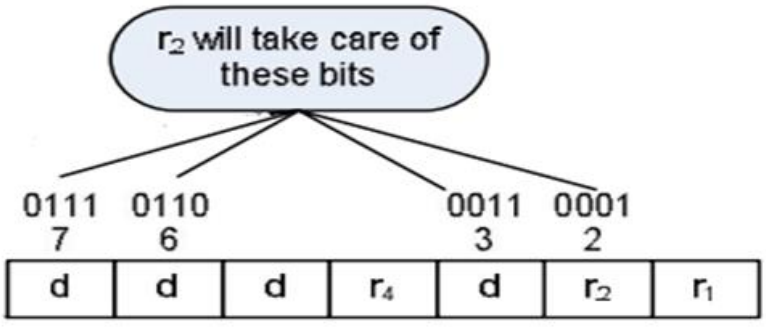
R4 (bit 4), 5, 6, 7; tổng số bit 1 là một số chẵn

Mỗi bit dữ liệu có thể tính đến trong nhiều hơn một lần tính VRC. Thí dụ, trong chuỗi trên, mội bit dữ liệu gốc được tính đến trong ít nhất hai tập, trong khi *r* chỉ được tính một lần

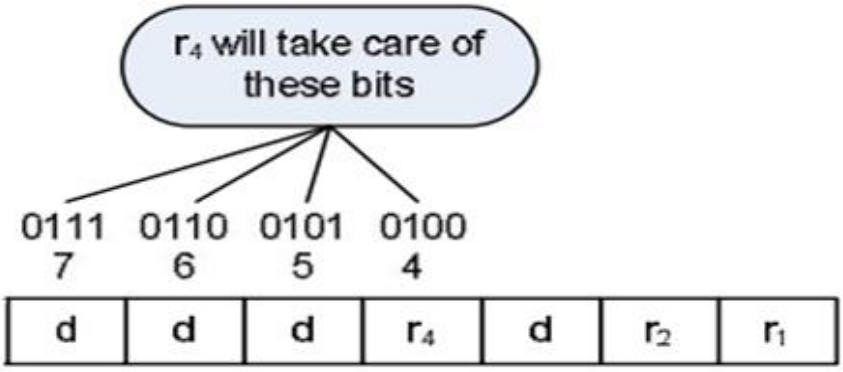
Để tìm các mẫu trong chiến lược tính toán này, hãy xem cách biểu diễn của mỗi vit trí bit. Bit *r1* được tính dùng tất cả các vị trí bit có cách biểu diễn nhị phân có 1 trong vị trí tận cùng bên phải. Bit *r2* được tính dùng tất cả các vị trí bit có cách biểu diễn nhị phân có 1 trong vị trí thứ hai bên phải và tiếp tục như vẽ trong hình sau:



Hình 4: Cách biểu diễn bit r_1

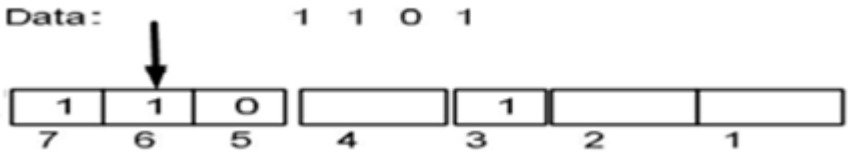


Hình 5: Cách biểu diễn bit r_2



Hình 6: Cách biểu diễn bit r_4

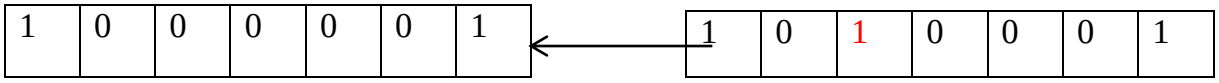
❖ **Tính toán các giá trị r:**



Hình 7: Đặt mỗi bit của ký tự gốc vào vị trí thích hợp

Bước đầu tiên, ta đặt mỗi bit của ký tự gốc vào vị trí thích hợp trong đơn vị 7 bit. Trong bước kế tiếp, ta tính các parity chẵn với nhiều tổ hợp bit khác nhau. Giá trị parity của mỗi tổ hợp là giá trị bit *r* tương ứng.Thí dụ, giá trị của *r1* được tính để cung cấp parity chẵn cho tổ hợp các bit 3, 5, 7. Giá trị của *r2* được tính để cung cấp parity bit cho các bit 3, 6, 7 và giá trị của *r2* được tính để cung cấp parity bit cho các bit 5, 6, 7. Mã 7 bit sau cùng được gởi đi qua đường truyền.

9. Phát hiện và sửa lỗi



❖ Giả sử trong lúc truyền tín hiệu đi, bit thứ 7 đã thay đổi từ 1→0.

❖ Máy thu nhận và tính lại bốn số dư r ở bên thu (VRC):

- r1 bên thu, 1, 3, 5, 7 ; tổng số bit 1 là một số chẵn
- r2 bên thu, 2, 3, 6, 7; tổng số bit 1 là một số chẵn
- r4 bên thu, 4, 5, 6, 7 ; tổng số bit 1 là một số chẵn

=>Vị trí bit sai của dữ liệu thu là giá trị thập phân của số nhị phân r4 r2 r1.

Ví dụ: Giả sử máy thu nhận được một dữ liệu 1100110 đã được mã hoá dưới dạng

Hamming. Hãy cho biết chuỗi dữ liệu nhận được đúng hay sai.

7	6	5	4	3	2	1	Vị trí
1	1	0	0	1	1	0	

- r1 bên thu, 0, 1, 0, 1 ; tổng số bit 1 là một số chẵn → r1 =0
- r2 bên thu, 1, 1, 1, 1; tổng số bit 1 là một số chẵn → r2 =0
- r4 bên thu, 0, 0, 1, 1; tổng số bit 1 là một số chẵn → r4 =0

=>r4 r2 r1 =0000_2= 0_{10}, *Không có bit sai*

Ví dụ: Giả sử máy thu nhận được một dữ liệu 1101110 đã được mã hoá dưới dạng Hamming. Hãy cho biết chuỗi dữ liệu nhận được đúng hay sai.

7	6	5	4	3	2	1	Vị trí
1	1	0	1	1	1	0	

- r1 bên thu, 0, 1, 0, 1 ; tổng số bit 1 là một số chẵn → r1 =0
- r2 bên thu, 1, 1, 1, 1; tổng số bit 1 là một số chẵn → r2 =0
- r4 bên thu, 1, 0, 1, 1; tổng số bit 1 là một số chẵn → r4 =1

Vậy vị trí sai là giá trị thập phân của số nhị phân r4 r2 r1 bên thu, r4 r2 r1 =100_2= 4_{10}, Vậy vị trí sai là 2, sửa bit ở vị trí 4: '1' → '0'

❖ **Các lưu ý trong việc sửa lỗi:**

- Sửa các lỗi được phát hiện thông thường yêu cầu truyền lại khối dữ liệu
- Không thích hợp cho các ứng dụng trao đổi dữ liệu không dây
  - BER cao
  - Truyền lại nhiều
- Thời gian trễ truyền nhiều hơn so với thời gian truyền dữ liệu (VD: truyền vệ tinh..)
  - Khối dữ liệu được truyền lại bị lỗi và nhiều khối dữ liệu khác tiếp theo
- Cần thiết phải sửa lỗi dự vào các dữ liệu nhận được

Tuy nhiên, Hamming code cũng có một số hạn chế. Nó yêu cầu thêm các bit kiểm tra, do đó làm tăng chiều dài của khung dữ liệu. Ngoài ra, việc sửa chữa lỗi có thể làm giảm tốc độ truyền thông và tăng độ trễ.

Tóm lại, Hamming code là một thuật toán mã hóa và phát hiện lỗi rất hiệu quả được sử dụng rộng rãi trong các ứng dụng truyền thông số và lưu trữ. Nó cho phép phát hiện và sửa chữa lỗi truyền thông một cách đáng tin cậy, nhờ đó giúp đảm bảo tính toàn vẹn của dữ liệu.

CHƯƠNG 3: TÌM HIỂU VÀ THIẾT KẾ KHỐI HAMMING (7,4) ENCODER VÀ HAMMING (7,4) DECODER

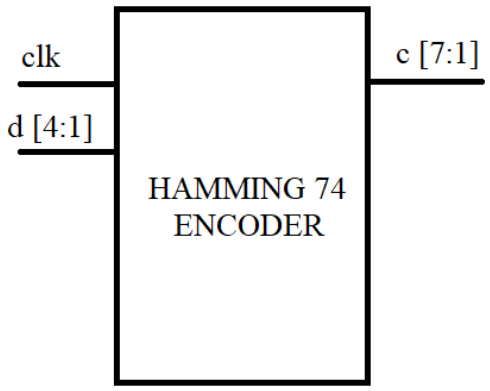
1. Tổng quan về encoder và decoder

Encoder (mã hóa) là một thiết bị hoặc mạch điện tử được sử dụng để chuyển đổi tín hiệu từ dạng không mã hóa sang dạng mã hóa.

Decoder (giải mã) là một thiết bị hoặc mạch điện tử được sử dụng để chuyển đổi tín hiệu từ dạng mã hóa sang dạng không mã hóa.

2. Hamming (7,4) Encoder

2.1 Sơ đồ khối



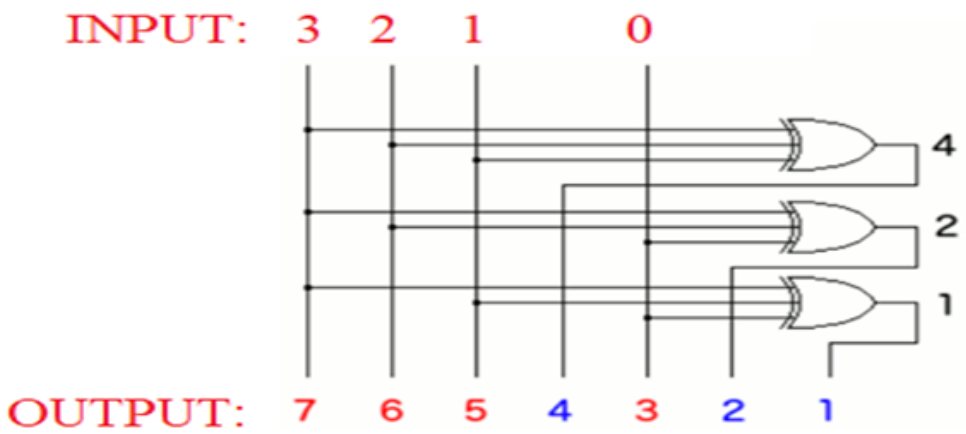
Hình 8: Khối hamming (7,4) encoder

Hamming (7,4) encoder có dữ liệu vào là 4 bit và dữ liệu ngõ ra (mã hamming) là 7 bit. Có 3 bit kiểm tra (p) được thêm vào chuỗi dữ liệu để tạo thành mã hamming.

Số lượng bit kiểm tra (r) được tính theo công thức $2^p \geq m+p+1$

- p: số bit kiểm tra
- m: số bit dữ liệu

→ Ở đây ta thiết kế khối hamming (7,4) encoder nên đã cố định số bit kiểm tra cũng như số bit dữ liệu vào (m=7, p=3).



Hình 9: Sơ đồ scientific hamming (7,4) encoder

2.2 Thiết kế giải thuật

Hamming (7,4) encoder

3	2	1		0			Vị trí bit dữ liệu vào
7	6	5	4	3	2	1	Vị trí bit ngõ ra
d4	d3	d2	p_2	d1	p_1	p_0	Mã hamming (7,4)

❖ ***Vị trí các bit tại ngõ ra:***

HAMMING (7,4) ENCODER

3	2	1		0			Vị trí bit dữ liệu vào
7	6	5	4	3	2	1	Vị trí bit ngõ ra
d4	d3	d2	p_2	d1	p_1	p_0	Mã hamming (7,4)

- 4 bit dữ liệu (d_4 d_3 d_2 d_1) và 3 bit kiểm tra (p_2 p_1 p_0).

- Bit kiểm tra p_i được đặt tại vị trí 2^i, 3 bit kiểm tra (p_2, p_1, p_0) lần lượt được điền vào các vị trí 4, 2, 1 và 4 bit dữ liệu được điền vào các vị trí 7, 6, 5, 3.

❖ ***Tính toán các bit kiểm tra:***

Dựa vào cách tính toán ở lý thuyết, từ đó có thể viết ra được các hàm để tính toán các bit kiểm tra như sau:

- Bit kiểm tra p_2 được tính bằng cách thực hiện các phép XOR giữa 3 bit dữ liệu d4, d3, d2

  $p2 = d4 \oplus d3 \oplus d2$
- Bit kiểm tra p_1 được tính bằng cách thực hiện các phép XOR giữa 3 bit dữ liệu d4, d3, d1

  $p1 = d4 \oplus d3 \oplus d1$
- Bit kiểm tra p_0 được tính bằng cách thực hiện các phép XOR giữa 3 bit dữ liệu d4, d2, d1

  $p0 = d4 \oplus d2 \oplus d1$

Ví dụ: Cho dữ liệu 4 bit: 1001 vào khối hamming (7,4) encoder, mã hamming tại ngõ ra là?

7	6	5	4	3	2	1	Vị trí bit ngõ ra
1	0	0	p_2	1	p_1	p_0	Mã hamming (7,4)

- 4 bit dữ liệu lần lượt được điền vào các vị trí của bit ngõ ra (7, 6, 5, 3).
- Các bit kiểm tra được tính toán theo công thức:
  - $p_2 = 1 \oplus 0 \oplus 0 = 1$
  - $p_1 = 1 \oplus 0 \oplus 1 = 0$
  - $p_0 = 1 \oplus 0 \oplus 1 = 0$

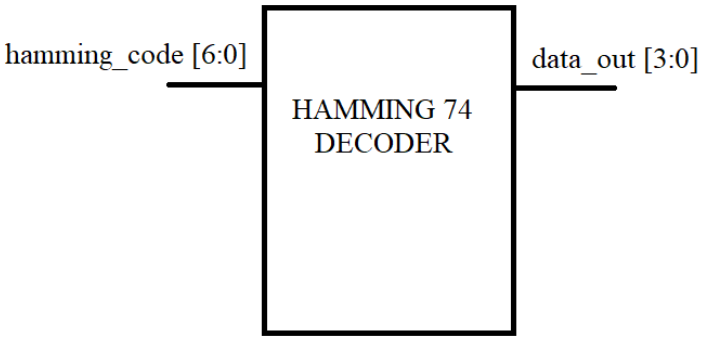
→ Các bit kiểm tra sẽ được điền vào các vị trí của bit ngõ ra

7	6	5	4	3	2	1	Vị trí bit ngõ ra
1	0	0	1	1	0	0	Mã hamming (7,4)

→ Mã hamming (7,4) tại ngõ ra: 1001100 khi dữ liệu vào là: 1001

3. Hamming (7,4) decoder

3.1 Sơ đồ khối

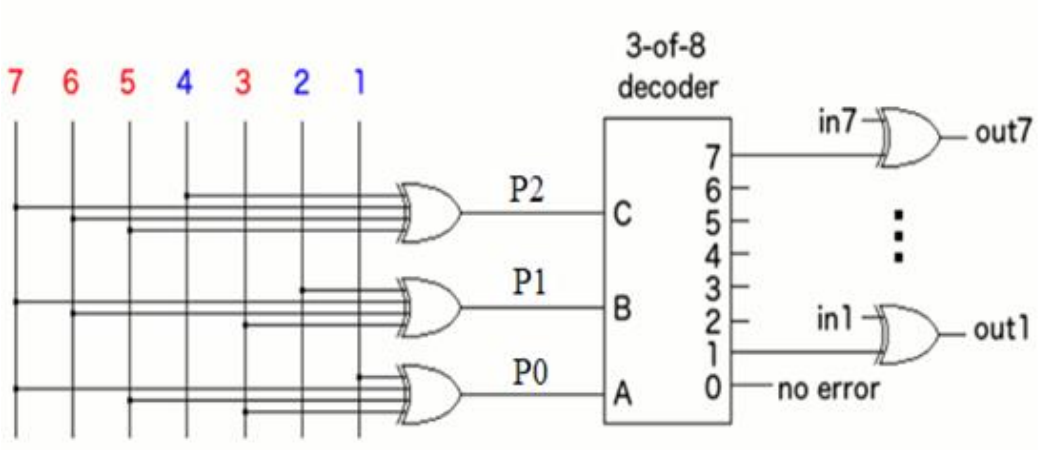


Hình 9: Khối hamming (7,4) Decoder

Khối hamming 74 decoder trên có thể phát hiện và sửa được lỗi 1 bit từ bất kỳ vị trí nào trong mã hamming 74 đầu vào.

Sau khi phát hiện và sửa lỗi 1 bit (nếu có), khối tiến hành loại bỏ các bit kiểm tra và đưa dữ liệu 4 bit đã được giải mã sang đầu ra.

3.2 Thiết kế giải thuật



Hình 10: Sơ đồ scientific hamming (7,4) Decoder

❖ ***Kiểm tra lỗi***

- Các bit kiểm tra của mã hamming được tính toán theo công thức ở khối encoder.
- Các bit kiểm tra ở vị trí 4, 2, 1.
- Tiến hành tính toán lại các bit kiểm tra và so sánh với bit kiểm tra cũ bằng phép XOR, nếu bất kỳ bit nào trong đó bị thay đổi thì P_i sẽ nhận giá trị 1.
    - P_2 được tính bằng cách thực hiện các phép XOR giữa 4 bit 7, 6, 5 và 4
    - $P_2 = 7 \oplus 6 \oplus 5 \oplus 4$
    - P_1 được tính bằng cách thực hiện các phép XOR giữa 4 bit 7, 6, 3 và 2
    - $P_1 = 7 \oplus 6 \oplus 3 \oplus 2$
    - P_0 được tính bằng cách thực hiện các phép XOR giữa 4 bit 7, 5, 3 và 1
    - $P_0 = 7 \oplus 5 \oplus 3 \oplus 1$

❖ ***Phát hiện lỗi***

Sau khi tính được các bit kiểm tra (P_2 , P_1, P_0), các bit này được qua bộ giải mã 3 sang 8 để tìm ra được vị trí bit bị lỗi (chỉ có thể phát hiện được 1 bit lỗi), sau đó sẽ xuất ra từ mã 7 bit để tiến hành sửa lỗi.

C	**B**	**A**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	
0	**0**	**0**	0	0	0	0	0	0	0	**No error**
0	**0**	**1**	0	0	0	0	0	0	1	**Lỗi bit 1**
0	**1**	**0**	0	0	0	0	0	1	0	**Lỗi bit 2**
0	**1**	**1**	0	0	0	0	1	0	0	**Lỗi bit 3**
1	**0**	**0**	0	0	0	1	0	0	0	**Lỗi bit 4**
1	**0**	**1**	0	0	1	0	0	0	0	**Lỗi bit 5**
1	**1**	**0**	0	1	0	0	0	0	0	**Lỗi bit 6**
1	**1**	**1**	1	0	0	0	0	0	0	**Lỗi bit 7**

❖ ***Sửa lỗi***

- Việc sửa lỗi này được hiện bằng cách sử dụng phép XOR giữa mã hamming đầu vào với từ mã 7 bit nhận từ bộ giải mã.
- Bit bị lỗi sẽ được lật lại thành bit đúng thông qua cổng XOR.

- Sau khi sửa được lỗi, khối tiến hành loại bỏ các bit kiểm tra và xuất ra 4 bit dữ liệu đã được giải mã sang đầu ra.

Vd: Mã hamming (7,4) đúng được truyền tới là: 1001100, giả sử trong quá trình truyền mã hamming (7,4) bị sai ở vị trí số 7: 0001100, mã được đưa qua khối decoder, tìm dữ liệu giải mã ở ngõ ra?

**Bước 1:*

- $P_2 = 0 \oplus 0 \oplus 0 \oplus 1 = 1$
- $P_1 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$
- $P_0 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$

**Bước 2:*

C	**B**	**A**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	
1	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**Lỗi bit 7**

**Bước 3:*

- Mã hamming (7,4) vào: 0001100
- Phép XOR: $\oplus$
- Từ mã phát hiện lỗi: 1000000
- Mã hamming (7,4) được sửa:1001100

→ ***Dữ liệu 4 bit được giải mã là:*** 1001

CHƯƠNG 4: ĐÁNH GIÁ QUA TEST BENCH

1. Encoder Test Bench

1.1 Dự đoán lý thuyết.

Đánh giá test bench với code Encoder, nhằm kiểm tra việc mã sửa lỗi Hamming đưa ra chuỗi dữ liệu Hamming 7 bit, khi dữ liệu đưa vào là 4 bit.

Quá trình đánh giá sẽ bao gồm 16 testcase tương ứng với 16 kiểu dữ liệu 4 bit sau:

1, d=4'b0000;
2, d=4'b0001;
3, d=4'b0010;
4, d=4'b0011;
5, d=4'b0100;
6, d=4'b0101;
7, d=4'b0110;
8, d=4'b0111;
9, d=4'b1000;
10, d=4'b1001;
11, d=4'b1010;
12, d=4'b1011;
13, d=4'b1100;
14, d=4'b1101;
15, d=4'b1110;
16, d=4'b1111;

Bởi vì Encoder là quá trình mã Hamming nhận dữ liệu 4 bit từ nguồn phát và đưa ra chuỗi chữ liệu hamming 7 bit khi kết thúc quá trình. Để có thể hình dung rõ hơn về cách thức tạo mã Hamming, ta sẽ áp dụng tính toán lý thuyết để có thể đưa ra dự đoán lý thuyết về những kết quả hiển thị sau quá trình test bench.

Ví dụ 1:

Dữ liệu 4 bit được đưa vào là d= “ 0000 ”

Xác định số bit dư: số bit của dữ liệu là m=4;

=> Suy ra số bit dư r theo bất đẳng thức: $2^r \geq$ m+r+1

❖ Với m= 4 ; $2^r \geq$ m+r+1 ; chọn r = 3

❖ Như vậy số bit cần thêm vào sẽ có trọng số R0, R1, R2 tương ứng với các vị trí :

$$2^0 = 1$$

$$2^1 = 2$$

$$2^2 = 4$$

- Ta có bảng chuyển hóa vị trí:

0	0	0	R2	0	R1	R0
7	5	5	4	3	2	1

Ta cần chuyển các bit 1 có trong chuỗi mã về dạng nhị phân, tuy nhiên trong trường hợp này, chỉ có bit 0, nên ta có thế quy ra mã VRC trực tiếp là "000'' tương ứng với giá trị của R0, R1, R2.

Như vậy chuỗi dữ liệu 7 bit sau khi hoàn thành quá trình sẽ là:

0	0	0	0	0	0	0

Ví dụ 2:

Dữ liệu 4 bit đưa vào có dạng: d= " 0110 "

Xác định số bit dư: số bit của dữ liệu là m= 4;

=>Suy ra số bit dư r theo bất đẳng thức: 2^r $\geq$ m+r+1

❖ Với m= 4 ; $2^r \geq$ m+r+1 ; chọn r = 3

Như vậy số bit cần thêm vào sẽ có trọng số R0, R1, R2 tương ứng với các vị trí :

$$2^0 = 1$$

$$2^1 = 2$$

$$2^2 = 4$$

- Ta có bảng chuyển hóa vị trí:

0	1	1	R2	0	R1	R0
7	6	5	4	3	2	1

Ta cần chuyển vị trí của các bit 1 về dạng mã nhị phân 3 bit để tìm mã VRC. Trong trường hợp này, bit 1 xuất hiện ở vị trí số 5 và số 6. Lập bảng tạo mã VRC với mode chẵn:

Nhị phân 5	1	0	1
Nhị phân 6	1	1	0
VRC	0	1	1

Mã VRC thu được là ‘‘ 011’’ tương ứng với giá trị của R0, R1, R2. Thay các giá trị vào chuỗi dữ liệu 7 bit ta được chuỗi dữ liệu Hamming khi kết thúc quá trình encoder:

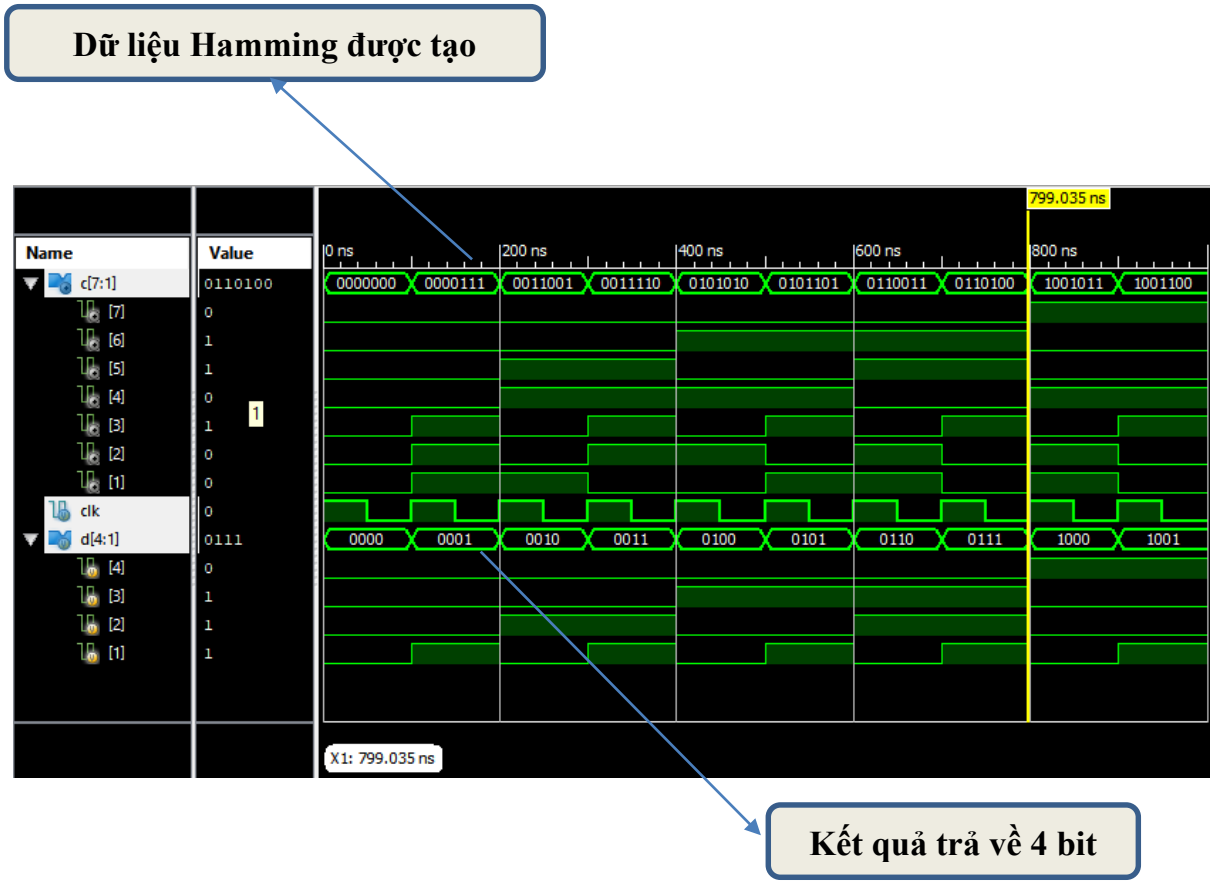
0	1	1	0	0	1	1

Như vậy, sau khi áp dụng tính toán lý thuyết cho tất cả các testcase, ta có được bảng dự đoán kết quả test bench cho quá trình encoder như sau:

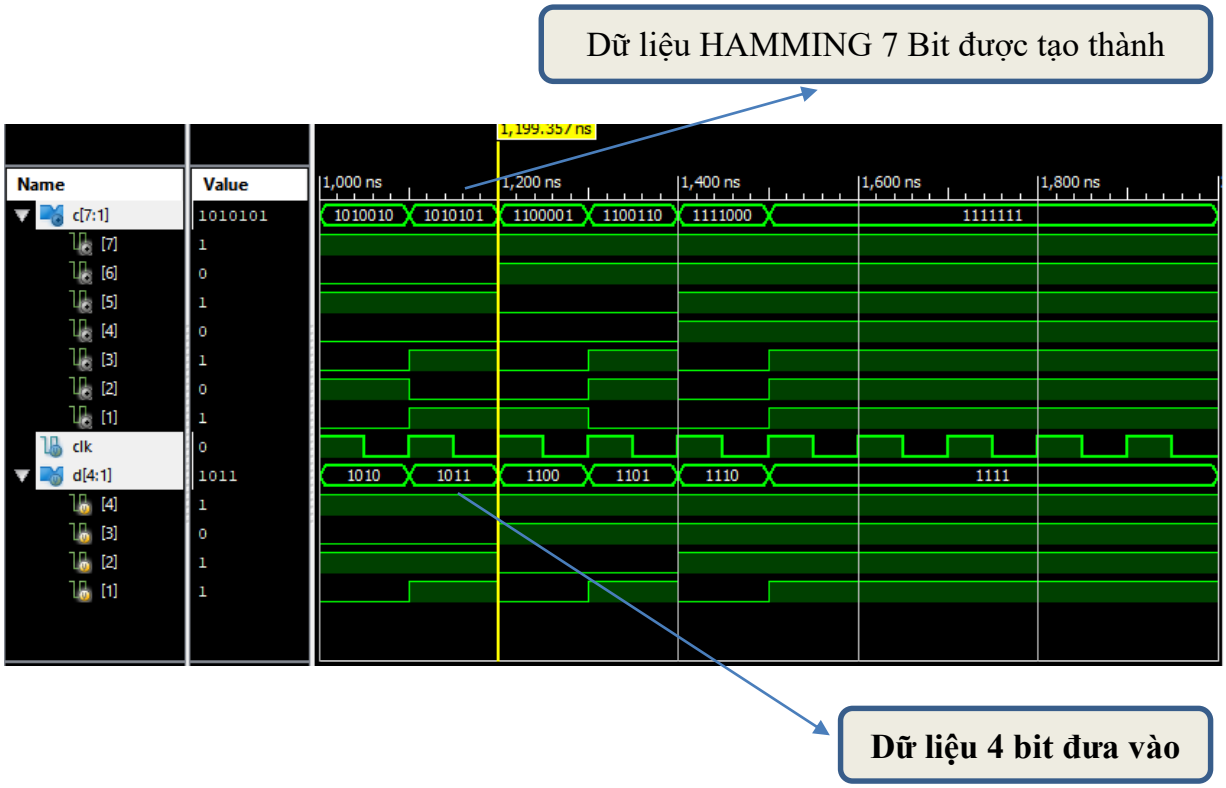
Testcase	1	2	3	4	5	6	7	8
7 Bit	0000	0001	0010	0011	0100	0101	0110	0111
4 Bit	0000000	0000111	0011001	0011110	0101010	0101101	0110011	0110100

9	10	11	12	13	14	15	16
1000	1001	1010	1011	1100	1101	1110	1111
1001011	1001100	1010010	1010101	1100001	1100110	1111000	1111111

1.2 Kết quả mô phỏng test bench.



Hình 11: Kết quả mô phỏng testcase 1- 10



Hình 12: Kết quả mô phỏng testcase 11-16

- Kết quả mô phỏng testcase hoàn hoàn trùng khớp với dự đoán lý thuyết ban đầu, có thể thấy quá trình encoder diễn ra theo đúng trình tự và độ chính xác cao.
- Xung clk xuất hiện song song cùng với quá trình encoder, mỗi chu kì tương ứng với 1 testcase, và mỗi lần xung clk được tích cực mức cao thì sẽ bắt đầu một testcase mới.

2. Decoder Test Bench

2.1 Dự đoán lý thuyết

Đánh giá test bench với code Decoder, nhằm kiểm tra việc sửa lỗi của mã Hamming nếu như trong chuỗi dữ liệu 7 bit có xuất hiện lỗi một bit, hoặc nhiều bit. Quá trình đánh giá sẽ bao gồm 19 testcase sau:

1; hamming_code = 7'b000_0_0_00
2; hamming_code = 7'b000_0_1_11
3; hamming_code = 7'b001_1_0_01
4; hamming_code = 7'b001_1_1_10
5; hamming_code = 7'b010_1_0_10
6; hamming_code = 7'b010_1_1_01
7; hamming_code = 7'b011_0_0_11
8; hamming_code = 7'b011_0_1_00
9; hamming_code = 7'b100_1_0_11
10; hamming_code = 7'b100_1_1_00
11; hamming_code = 7'b101_0_0_10
12; hamming_code = 7'b101_0_1_01
13; hamming_code = 7'b110_0_0_01
14; hamming_code = 7'b110_0_1_10
15; hamming_code = 7'b111_1_0_00
16; hamming_code = 7'b111_1_1_11

// gia su so 0 thap phan bi loi tai bit 1
17; hamming_code = 7'b000_0_0_01;
// gia su so 0 thap phan bi loi tai bit 4
18; hamming_code = 7'b000_1_0_00;
// gia su so 0 thap phan bi loi 2 bit 5 6
19; hamming_code = 7'b011_0_0_00;

Để có thể dự đoán dễ hơn về kết quả mô phỏng test bench ở quá trình decoder ta sẽ xét một vài testcase tiêu biểu như testcase số 2 và testcase số 17.

Vd: hamming_code = 7'b000_0_1_11

**Bước 1:*

- $P_2 = 0 \oplus 0 \oplus 0 \oplus 0 = 0$
- $P_1 = 0 \oplus 0 \oplus 1 \oplus 1 = 0$
- $P_0 = 0 \oplus 0 \oplus 1 \oplus 1 = 0$

**Bước 2:*

C	B	A	7	6	5	4	3	2	1	
0	0	0	0	0	0	0	0	0	0	**Không có lỗi**

**Bước 3:*

- Mã hamming (7,4) vào: 0000111
- Phép XOR: $\oplus$
- Từ mã phát hiện lỗi: 0000000
- Mã hamming (7,4) được sửa:0000111

→ ***Dữ liệu 4 bit được giải mã là:*** 0001

Vd: hamming_code = 7'b000_1_0_00 // hamming code bị lỗi ở vị trí bit 4

**Bước 1:*

- $P_2 = 0 \oplus 0 \oplus 0 \oplus 1 = 1$
- $P_1 = 0 \oplus 0 \oplus 0 \oplus 0 = 0$
- $P_0 = 0 \oplus 0 \oplus 0 \oplus 0 = 0$

**Bước 2:*

C	B	A	7	6	5	4	3	2	1	
1	0	0	0	0	0	1	0	0	0	**Lỗi bit 4**

**Bước 3:*

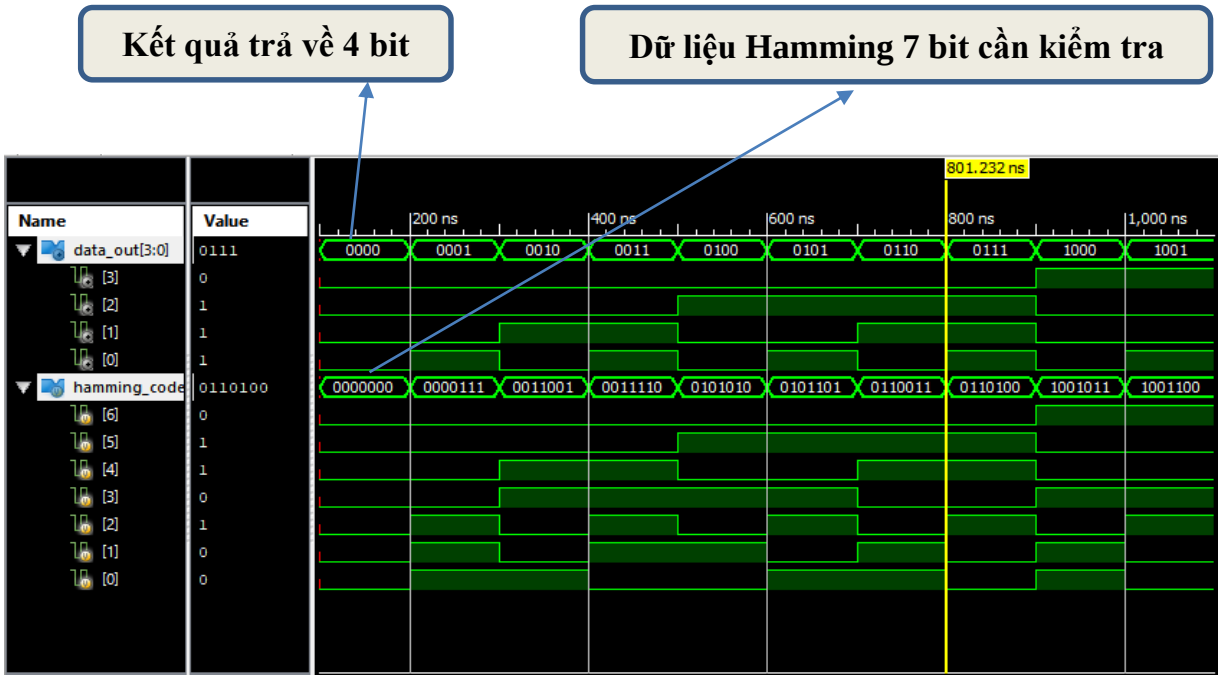
- Mã hamming (7,4) vào: 0001000
- Phép XOR: $\oplus$
- Từ mã phát hiện lỗi: 0001000
- Mã hamming (7,4) được sửa:0000000

→ ***Dữ liệu 4 bit được giải mã là:*** 0000

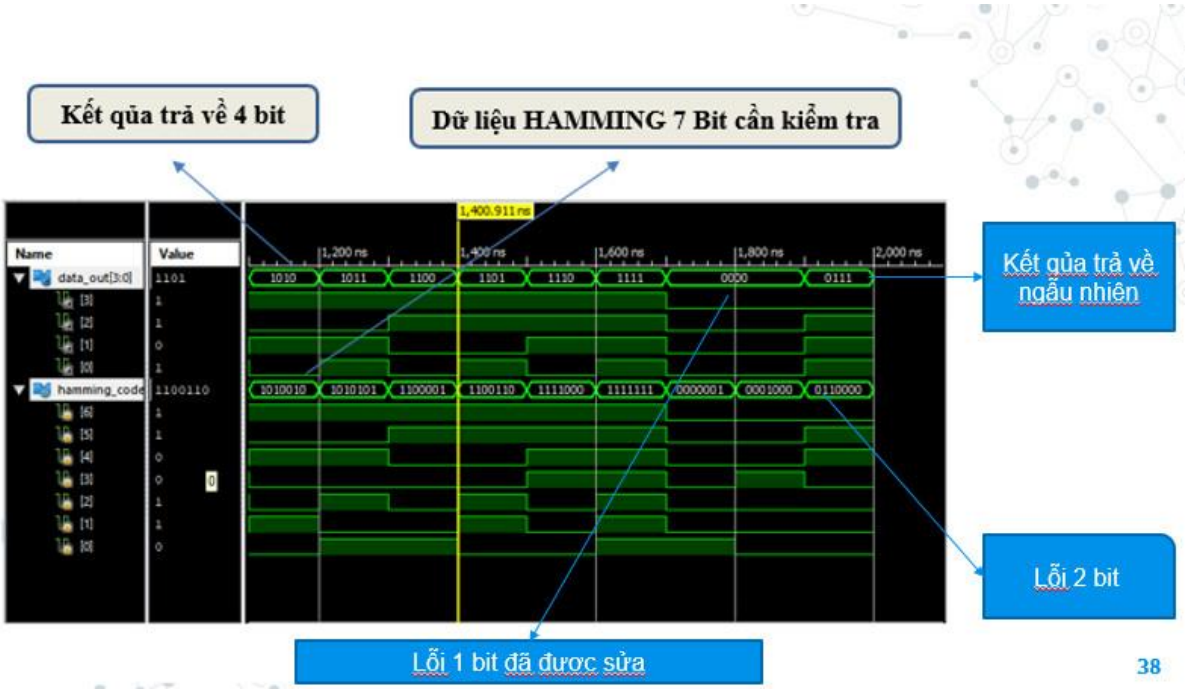
Với số lượng 16 testcase từ 1-16, mã sửa lỗi Hamming sẽ thực hiện quá trình trả về dữ liệu 4 bit từ chuỗi dữ liệu Hamming 7 bit được nhận. Nếu dự đoán đúng thì, kết quả mô phỏng sẽ ngược lại với quá trình Encoder, và những dữ liệu 4 bit sẽ hoàn toàn trùng khớp với dữ liệu 4 bit được nhắc đến trong quá trình encoder, nghĩa là chuỗi dữ liệu Hamming 7 bit nhận được không có lỗi bit.

Với testcase 17,18,19 ta sẽ đưa vào chuỗi dữ liệu Hamming 7 bit có trường hợp bị lỗi 1 bit và lỗi 2 bit, để kiểm tra tính sửa lỗi của mã Hamming. Nếu dự đoán đúng thì với lỗi 1 bit, mã Hamming sẽ phát hiện lỗi đồng thời sửa lỗi tại vị trí đó và trả lại kết quả đúng theo lý thuyết. Còn với lỗi 2 bit thì mã Hamming không thể sửa lỗi nên sẽ trả về kết quả bất kì không đúng với lý thuyết.

2.2 Kết quả mô phỏng test bench



Hình 13: Kết quả mô phỏng testcase 1-10



Hình 14: Kết quả mô phỏng testcase 11-19

❖ Với kết quả hiển thị trên mô phỏng, ta có thể dễ dàng kiểm tra độ chính xác của dự đoán khi tính toán lý thuyết và mô phỏng là hoàn toàn trùng khớp.

- Dễ dàng nhận thấy mã sửa lỗi Hamming hoạt động tốt khi trả về dữ liệu ban đầu từ chuỗi dữ liệu Hamming 7 bit được tạo trùng khớp với mã Hamming 4 bit ở quá trình encoder.
- Với lỗi 1 bit thì ở bất cứ vị trí nào trong chuỗi, mã Hamming đều có thể phát hiện và sửa lỗi (ví dụ ở testcase 17, 18). Còn đối với testcase 19, dữ liệu đầu vào xuất hiên lỗi 2 bit thì mã Hamming không thể sửa lỗi, lúc này dữ liệu trả về sẽ là bất kì.

CHƯƠNG 5: ỨNG DỤNG VÀ KẾT LUẬN

❖ Ứng dụng

Hiện nay mã Hamming được sử dụng trong nhiều ứng dụng, chẳng hạn như nghiên cứu khoa học, ứng dụng y tế và giao dịch tài chính. Trong nghiên cứu khoa học, mã được sử dụng để truyền và lưu trữ dữ liệu từ các thí nghiệm và mô phỏng, trong đó độ chính xác là rất quan trọng. Trong các ứng dụng y tế, mã được sử dụng để lưu trữ và truyền dữ liệu bệnh nhân, chẳng hạn như hồ sơ y tế và kết quả xét nghiệm, phải chính xác và không có lỗi. Trong các giao dịch tài chính, mã được sử dụng để đảm bảo rằng các giao dịch được xử lý chính xác và không xảy ra lỗi trong quá trình chuyển tiền.

❖ Kết luận

Mã Hamming là một loại mã sửa lỗi cụ thể được sử dụng rộng rãi trong các hệ thống truyền thông và lưu trữ kỹ thuật số. Đó là một mã nhị phân, có nghĩa là nó chỉ sử dụng hai ký hiệu, 0 và 1. Mã này hoạt động bằng cách thêm các bit chẵn lẻ vào thông báo, cho phép người nhận phát hiện và sửa bất kỳ lỗi nào có thể xảy ra trong quá trình truyền hoặc lưu trữ.

Mã Hamming có thể được triển khai bằng nhiều ngôn ngữ lập trình khác nhau, chẳng hạn như C ++, Python và Java. Mã này có thể được triển khai như một chức năng nhận một thông báo làm đầu vào và trả về thông báo với các bit chẵn lẻ được thêm vào. Chức năng này cũng có thể bao gồm các kỹ thuật phát hiện và sửa lỗi để đảm bảo rằng mọi lỗi đều được phát hiện và sửa chữa.

Và trong đó Hamming(7,4) là phương pháp mã hóa đơn giản và nó cho phép chúng ta phát hiện lỗi trong 1 bit và sửa nó hoặc phát hiện lỗi trong 2 bit (không đủ thông tin để sửa).Thuật toán Hamming (7,4) có thể sửa bất kỳ lỗi bit đơn nào hoặc phát hiện tất cả lỗi bit đơn và lỗi hai bit. Nói cách khác, khoảng cách Hamming tối thiểu giữa hai từ mã chính xác bất kỳ là 3 và các từ nhận được có thể được giải mã chính xác nếu chúng ở khoảng cách tối đa một từ từ mã được người gửi truyền đi. Điều này có nghĩa là đối với các trường hợp phương tiện truyền dẫn không xảy ra lỗi chùm , mã của Hamming (7,4) có hiệu quả (vì phương tiện sẽ phải cực kỳ nhiễu để hai trong số bảy bit bị lật).

Mã Hamming là mã sửa lỗi hiệu quả cao được sử dụng rộng rãi trong các hệ thống lưu trữ và truyền thông kỹ thuật số. Mã này có khả năng phát hiện và sửa lỗi với số lượng dự phòng tối thiểu, làm cho nó có hiệu quả cao. Nó được sử dụng trong nhiều ứng

dụng, chẳng hạn như nghiên cứu khoa học, ứng dụng y tế và giao dịch tài chính. Mặc dù mã Hamming có một số hạn chế nhưng nó vẫn là một trong những mã sửa lỗi đáng tin cậy nhất hiện nay.

Cho đến ngày nay Mã Hamming vẫn được sử dụng trong các ứng dụng như bộ nhớ ECC. Một số mã phát hiện lỗi đơn giản đã được sử dụng trước mã Hamming, nhưng không mã nào hiệu quả như mã Hamming mã trong cùng một chi phí của không gian.

***Encoder:**

```
module hamming_encoder(clk,d,c);
  input clk;
  input [4:1] d;
  output reg [7:1] c;
 always @ (posedge clk)
  begin
        c[7]=d[4];
        c[6]=d[3];
        c[5]=d[2];
        c[4]=d[2]^d[3]^d[4];
        c[3]=d[1];
        c[2]=d[1]^d[3]^d[4];
        c[1]=d[1]^d[2]^d[4];
  end
endmodule
```

***Encoder:**

```
module test_en;
        reg clk;
        reg [4:1] d;
        wire [7:1] c;

  hamming_encoder encoder_test(clk,d,c);
  initial
  begin
    forever
    begin
```

```
    clk=1;
    #50 clk=0;
    #50 clk=1;
    end
  end
 initial
 begin
 d=4'b0000;
 #100 d=4'b0001;
 #100 d=4'b0010;
 #100 d=4'b0011;
 #100 d=4'b0100;
 #100 d=4'b0101;
 #100 d=4'b0110;
 #100 d=4'b0111;
 #100 d=4'b1000;
 #100 d=4'b1001;
 #100 d=4'b1010;
 #100 d=4'b1011;
 #100 d=4'b1100;
 #100 d=4'b1101;
 #100 d=4'b1110;
 #100 d=4'b1111;
 end
 initial
 begin
    $monitor($time,"clk=%b,d=%b,c=%b",clk,d,c);
 end
endmodule
```

***Decoder:**

```
module hamming74_decoder(
   input [6:0] hamming_code,
   output [3:0] data_out
);

reg [2:0] parity;
reg [6:0] error_pattern;
wire [6:0] corrected_hamming_code;

always @(*) begin
   parity[0] = hamming_code[0] ^ hamming_code[2] ^ hamming_code[4] ^ hamming_code[6];
   parity[1] = hamming_code[1] ^ hamming_code[2] ^ hamming_code[5] ^ hamming_code[6];
   parity[2] = hamming_code[3] ^ hamming_code[4] ^ hamming_code[5] ^ hamming_code[6];
end

always @(*) begin
   case(parity)
            3'b000: error_pattern = 7'b0000000;
            3'b001: error_pattern = 7'b0000001;
            3'b010: error_pattern = 7'b0000010;
            3'b011: error_pattern = 7'b0000100;
            3'b100: error_pattern = 7'b0001000;
            3'b101: error_pattern = 7'b0010000;
            3'b110: error_pattern = 7'b0100000;
            3'b111: error_pattern = 7'b1000000;
      endcase
   end
```

```
assign corrected_hamming_code = error_pattern ^ hamming_code;
assign  data_out={corrected_hamming_code[6:4],
corrected_hamming_code[2]};
endmodule
```

***Decoder**

```
module TEST_DECODER;

                // Inputs
                reg [6:0] hamming_code;

                // Outputs
                wire [3:0] data_out;

                // Instantiate the Unit Under Test (UUT)
                hamming74_decoder uut (
                  .hamming_code(hamming_code),
                  .data_out(data_out)
                );

                initial begin
                $display($time, "TEST START");
                $monitor($time, " hamming_coder = %b, data_out = %d ",
hamming_code, data_out);

                // data chay tu 0 den 15 --> hamming_code
                #1; hamming_code = 7'b000_0_0_00;     // so 0 thap phan
                #1; hamming_code = 7'b000_0_1_11;
                #1; hamming_code = 7'b001_1_0_01;
                #1; hamming_code = 7'b001_1_1_10;
```

```
                    #1; hamming_code = 7'b010_1_0_10;
                    #1; hamming_code = 7'b010_1_1_01;
                    #1; hamming_code = 7'b011_0_0_11;
                    #1; hamming_code = 7'b011_0_1_00;
                    #1; hamming_code = 7'b100_1_0_11;
                    #1; hamming_code = 7'b100_1_1_00;
                    #1; hamming_code = 7'b101_0_0_10;
                    #1; hamming_code = 7'b101_0_1_01;
                    #1; hamming_code = 7'b110_0_0_01;
                    #1; hamming_code = 7'b110_0_1_10;
                    #1; hamming_code = 7'b111_1_0_00;
  #1; hamming_code = 7'b111_1_1_11;    // so 15 thap phan

                    #1 $display($time, "1 bit error - bit 1");  // gia su so 0 thap phan bi loi tai bit 1
                    hamming_code = 7'b000_0_0_01;
                    #1 $display($time, "1 bit error - bit 1");  // gia su so 0 thap phan bi loi tai bit 4
                    hamming_code = 7'b000_1_0_00;
                    #1 $display($time, "1 bit error - bit 1");  // gia su so 0 thap phan bi loi 2 bit 5 6
                    hamming_code = 7'b011_0_0_00;
                    #1; hamming_code = 7'b000_0_0_00;

//                  $display($time, "TEST STOP");
//                  $stop;
                    end

endmodule
```

PHỤ LỤC

KẾ HOẠCH PHÂN CÔNG VIẾT TIỂU LUẬN

Nội dung hoàn thành	**Sinh viên hoàn thành**	**Mức độ hoàn thành**
Lời mở đầu	Phan Thị Miên	Tốt
CHƯƠNG 1: TỔNG QUAN		
Nội dung 1: Lý do chọn đề tài, mục tiêu, phương pháp nghiên cứu bố cục, giới hạn đề tài.	Phan Thị Miên	Tốt
CHƯƠNG 2: CƠ SỞ LÝ THUYẾT VỀ HAMMING CODE		
Nội dung 2: Khái niệm,đặc điểm,lịch sử.	Lê Thị Thi	Tốt
Nội dung 3: Cách thức thực hiện quá trình sửa lỗi, khoảng cách hamming ,các bit dư **.**	Lê Thị Thi	Tốt
Nội dung 4: mã hamming (7,4), định vị của các bit dư, phát hiện và sửa lỗi.	Lê Thị Thi	Tốt
CHƯƠNG 3: TÌM HIỂU VÀ THIẾT KẾ KHỐI HAMMING (7,4) ENCODER VÀ HAMMING (7,4) DECODER		
Nội dung 5: Sơ đồ khối	Nguyễn Phạm Anh Tùng	Tốt
Nội dung 6:Thiết kế giải thuật	Nguyễn Phạm Anh Tùng	Tốt
CHƯƠNG 4: ĐÁNH GIÁ QUA TEST BENCH		
Nội dung 8: Encoder Test Bench	Lê Như Tuấn	Tốt
Nội dung 9: Decoder Test Bench	Lê Như Tuấn	Tốt

CHƯƠNG 5: ỨNG DỤNG VÀ KẾT LUẬN		
Nội dung 10: Ứng dụng.	Phan Thị Châu Pha	Tốt
Nội dung 11: Kết luận.	Phan Thị Châu Pha	Tốt
TỔNG HỢP NỘI DUNG BÀI BÁO CÁO	Phan Thị Miên	Tốt
CHỈNH SỬA HOÀN THIỆN	Phan Thị Châu Pha	Tốt

TÀI LIỆU THAM KHẢO

[1] Youtube channel: Ovisign Verilog HDL Tutorials, How to design a Hamming74 Encoder and Decoder for FPGA using Verilog. Ngày truy cập: 12/05/2023, tại:

Link: https://www.youtube.com/watch?v=doU4sFQpp4A

Link: https://www.youtube.com/watch?v=8VXA021mHcM&t=68s

[2] Wikipedia, Hamming (7,4). Ngày truy cập: 12/05/2023, tại:

Link: https://en.wikipedia.org/wiki/Hamming_code#Encoding

Link: https://en.wikipedia.org/wiki/Hamming(7,4)

[3]Phát hiện và sửa lỗi (15/5.2003), truy cập lại:

Link:https://nguyenvanquangcse.files.wordpress.com/2016/09/slide-4_errordetection-ver2.pdf

[4] Mã hamming-wikipedia (15/5/2023), truy cập lại:

Link :https://vi.wikipedia.org/wiki/M%C3%A3_Hamming

[5] Chương 9 giáo trình kỹ thuật truyền số liệu

[6] Tài liệu xanh –Mã hamming (15/5/2023), truy cập lại:

Link : https://tailieuxanh.com/vn/tlID658153_ma-hamming.html