ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Thiết kế luận lí với HDL (CO1025)

BÀI TẬP LỚN

Chủ đề:

MÃ HAMMING

Sinh viên thực hiện	Mã số sinh viên
Huỳnh Bùi Ngọc Khoa	2211589
Lương Văn Khanh	2211486
Nguyễn Phúc An	2210022
Bùi Xuân Bách	2210179

Thành phố Hồ Chí Minh – 2023

MỤC LỤC

1. Lời nói đầu:

Trong thời buổi công nghệ hiện nay, chúng ta đang đến với sự bùng nổ của công nghệ số, vì vậy nhu cầu truyền dẫn dữ liệu giữa các hệ thống là vô cùng lớn và sai số trong quá trình truyền dẫn là không thể tránh khỏi. Trong các hệ thống kỹ thuật số, dữ liệu được truyền khi giao tiếp có thể bị hỏng do nhiễu bên ngoài hoặc bất kì lỗi vật lý nào khác. Nếu dữ liệu được truyền không khớp với dữ liệu đầu vào đã cho, thì nó được gọi là **"lỗi"**. Việc truyền dữ liệu sẽ ở dạng bit (0 và 1) trong hệ thống kỹ thuật số. Nếu bit '1' bị thay đổi thành bit '0' hoặc ngược lại, thì nó được gọi là lỗi bit. Các lỗi dữ liệu hay lỗi bit có thể xóa dữ liệu quan trọng, trong nhiều trường hợp có thể gây hậu quả nghiêm trọng về thông tin. Có các loại lỗi khác nhau như lỗi bit đơn, nhiều lỗi và lỗi cụm. Do đó để giải quyết tình trạng này, chúng ta cần có các chương trình hoặc thuật giải mã hiệu quả. Một trong số những thuật giải mã đơn giản nhưng khá thông dụng hiện nay là mã Hamming (7,4), là một dạng mã cơ bản có thể truy vết và sửa chữa các lỗi bit đơn hiệu quả.

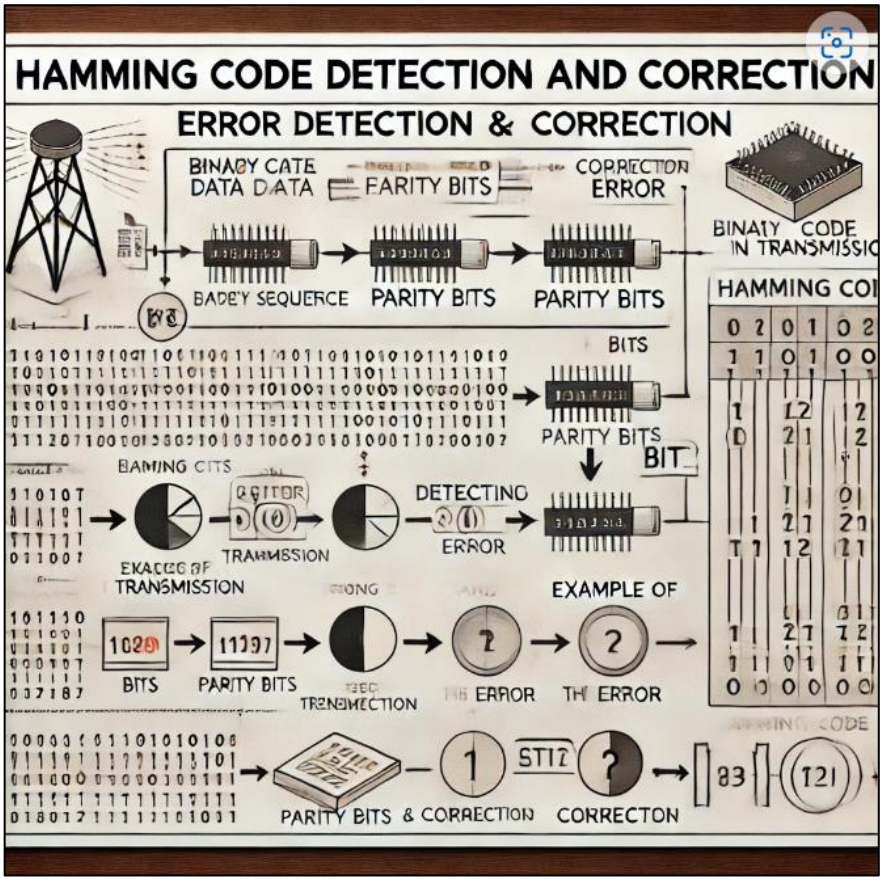
Ngày nay, với việc áp dụng toán tin vào việc xử lí các vấn đề toán học đã dần trở nên đơn giản và phổ biến. Trong số các phần mềm xử lí số, MATLAB là một phần mềm được sử dụng khá rộng rãi, có vai trò cung cấp môi trường tính toán số và lập trình, giúp xử lí các vấn đề toán học một cách dễ dàng hơn.

Bài báo cáo của chúng em xin được trình bày phương pháp xử lí lỗi bit dữ liệu bằng mã Hamming (7,4) thông qua các cơ sở lí thuyết và code (Verilog và MATLAB).

2. Giới thiệu:

Với đề tài mã Hamming, chúng em sẽ đi sơ qua về nguyên lý truyền dẫn và hoạt động của phương pháp sửa lỗi này. Sau đây là sơ lược về cách tổ chức của phương pháp sửa lỗi Hamming (cụ thể là **Hamming (7,4)**).

Trong quá trình truyền dẫn dữ liệu từ nơi truyền đến nơi nhận, sai số có thể xảy ra và với các hệ thống không bị nhiễu quá cao thì sai số chỉ thường xảy ra ở 1 bit. Với phương pháp Hamming, nó cho phép chúng ta phát hiện và sửa lỗi 1 bit này. Ta sẽ có 4 bit dữ liệu cần truyền đi và 3 bit dữ liệu kèm theo để có thể phát hiện và sửa lỗi, 3 bit dữ liệu này được tạo ra từ 4 bit dữ liệu gốc. Vậy khi muốn truyền dữ liệu 4 bit ta cần truyền tổng cộng 7 bit và bên nhận sẽ nhận 7 bit dữ liệu này và thực hiện kiểm tra, phát hiện và sửa lỗi để có được 4 bit dữ liệu cần nhận. Và chúng ta sẽ thiết kế bộ xử lý 7 tín hiệu nhận được này, thực hiện kiểm tra và sửa lỗi 1 bit (nếu có) để nhận được 4 bit dữ liệu thực tế cần được truyền tải.



3. Nguyên lý của mã Hamming:

Chúng ta sẽ xác định các bit lỗi bằng cách nhân các ma trận được gọi là ma trận Hamming (Hamming matrices) với nhau. Đối với mã (7,4), chúng ta sử dụng 2 ma trận liên quan gần gũi là H_e và H_d.

$$H_e := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

$$H_d := \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Các cột trong H_e là cơ sở nhân của H_d và 4 hàng đầu của H_e là một ma trận đơn vị. Sau khi làm phép tính nhân, 4 bit dữ liệu sẽ nằm ở 4 vị trí trên cùng. Khác với sử dụng ô mã với các bit kiểm tra ở các vị trí 2^k, khi sử dụng ma trận thì các bit dữ liệu nằm trên, các bit kiểm chẵn lẻ nằm dưới.

Chúng ta cộng 4 bit dữ liệu chủ chốt với 3 bit dữ liệu thừa lại là 7 bit (do đó mã mới có tên là mã (7, 4)). Để truyền gửi dữ liệu, ta nhóm các bit dữ liệu muốn gửi thành một vector. Cách làm cụ thể như sau:

Lấy ví dụ dữ liệu cần gửi là “1011” thì vector của nó là:

$$\boldsymbol{p} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

Chúng ta sẽ truyền gửi dữ liệu trên. Ta tìm tích của H_e và $\boldsymbol{p}$, sau đó lấy các giá trị ở 3 hàng dưới cùng của kết quả và modulo 2 (phép chia lấy phần dư cho 2) tương ứng với 3 bit kiểm tra:

$$H_e \times \boldsymbol{p} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 2 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \boldsymbol{r}$$

Máy thu sẽ nhân H_d với $\boldsymbol{r}$, để kiểm tra xem có lỗi hay không. Sau khi nhân xong tiếp tục lấy giá trị modulo 2:

$$H_d \times \boldsymbol{r} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Khi vector thu được là vector không ta kết luận là không có lỗi xảy ra. Sở dĩ vector không đồng nghĩa với không có lỗi, là do khi H_e nhân với vector dữ liệu thì sẽ tạo ra một vector nằm trong nhân của ma trận H_d. Nếu không có vấn đề xảy ra trong khi truyền thông, $\boldsymbol{r}$ sẽ nằm nguyên trong nhân của H_d và phép nhân trên sẽ cho ra vector không.

Bây giờ giả sử có lỗi xảy ra tại vị trí i, hay nói cách khác:

$$\boldsymbol{s} = \boldsymbol{r} + \boldsymbol{e}_i$$

với $\boldsymbol{e}_i$ là vector đơn vị thứ i

Kết quả của phép nhân $H_d \times \boldsymbol{r}$ sẽ ứng với cột thứ i của ma trận H_d, và đó cũng là vị trí bit lỗi, nhờ đó ta có thể biết lỗi ở đâu và sửa được nó.

Lấy ví dụ:

$$\boldsymbol{s} = \boldsymbol{r} + \boldsymbol{e}_2 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Phép nhân sẽ có kết quả là:

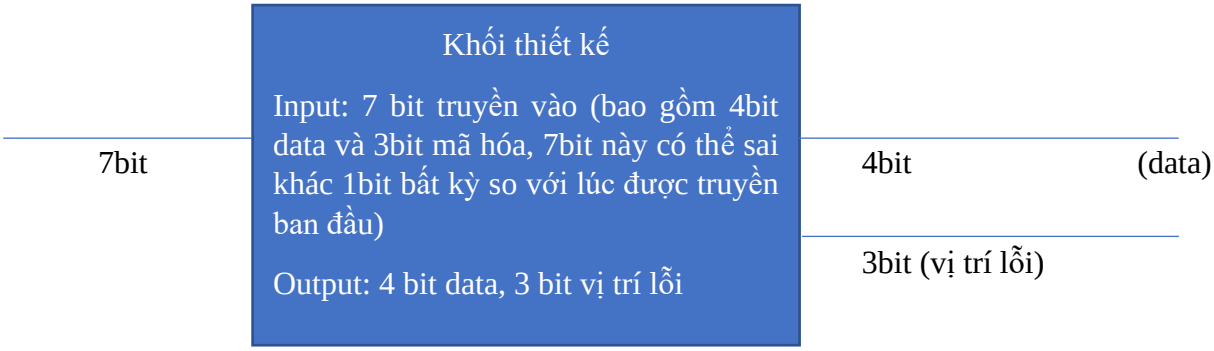
$$H_d \times \boldsymbol{s} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

Kết quả của phép nhân tương ứng với cột thứ 2 của H_d, do đó ta biết được có một lỗi xảy ra ở vị trí thứ 2 trong chuỗi bit. Mã Hamming có thể giúp phát hiện lỗi do 1 bit hoặc 2 bit bị đảo gây ra, song nó không thể thực hiện cả hai việc.

Từ các phép nhân ma trận và chia lấy phần dư cho 2 ta tìm ra mối tương quan để có thể hình dung và thiết kế mạch dựa trên nguyên lý trên.

4. Thiết kế và hiện thực mạch:

Sau khi nắm rõ được nguyên lý và cơ chế của mã Hamming, ta tìm ra mối liên quan giữa các phép toán trên và các cổng phần cứng (đó là cổng **XOR**) và cũng như hình dung các chân vào ra cho mạch mà chúng ta đang thiết kế.



Quy ước: bit [0:3] sẽ là 4 bit dữ liệu và bit [4:6] là 3 bit kiểm tra.

Chúng ta sẽ đi vào chi tiết của mạch dựa trên mối tương quan lý thuyết và nguyên lý ở trên:

+ Ta khởi tạo các chân vào 7 bit (input [6:0] received) và chân ra gồm 4 bit data (output [3:0] data) và 3 bit để xác định vị trí lỗi (output reg [2:0] error), nếu không có lỗi error=0. Một khi 7 bit input thay đổi tức thông tin mới đã được nạp vào thì chúng ta cần xác định lại data và xem xét lỗi, vậy nên chúng ta đặt các input trong khối always@().
+ Tương quan với ma trận H_e ta có các hàng 1, 2, 3, 4 tạo thành một ma trận đơn vị để sau khi nhân H_e với 4 bit data cần truyền (quá trình mã hóa trước khi truyền) ta được 4 bit data nằm ở ngay ở hàng 1, 2, 3, 4. Ba dòng tiếp theo trong ma trận H_e là để tạo ra 3 bit kiểm thử (quá trình mã hóa trước khi truyền), tương quan ta thấy đây là phép **XOR** bit thứ (1, 2, 3); (0, 2, 3); (0, 1, 3). Vậy trong 7 bit chúng ta nhận được sẽ có 4 bit đầu là data 3 bit sau là bit dùng để kiểm tra chẵn lẻ.

$$H_e := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

+ Tương quan với ma trận H_d (phần giải mã mà chúng ta thực hiện), H_d có các cột là các số nhị phân từ 1 đến 7 cho biết vị trí mà bit đó bị sai. Phép toán nhân H_d với vecto r (dãy dữ liệu 7 bit ta nhận được) ta sẽ thu được ba bit của output error.
+ Chúng lần lượt là phép **XOR** của các bit ở các vị trí (từ phải sang hoặc từ dưới lên theo dạng ma trận hay được thể hiện qua đoạn code mô tả mạch): (1, 2, 3, 6); (0, 2, 3, 5); (0, 1, 3, 4).

$$H_d := \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

+ Cuối cùng là phần kiểm tra xem error có bằng 0 hay không để có thể định lỗi hay không và sửa chúng.

5. Code Verilog:

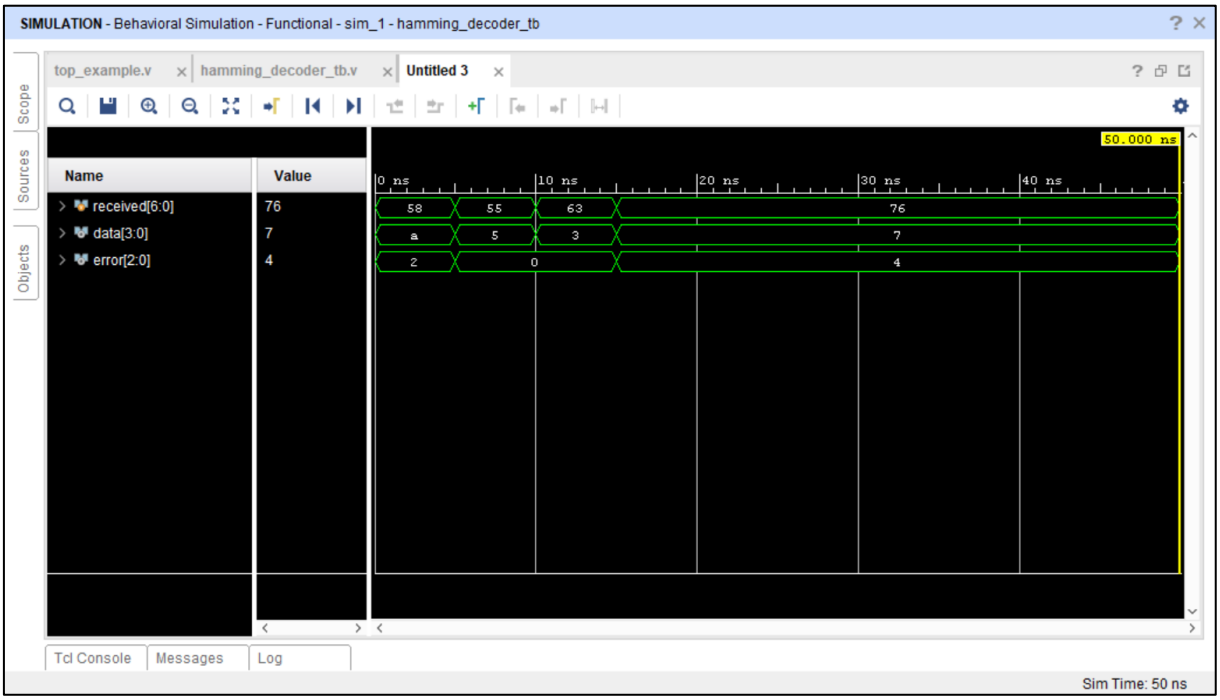
```
`timescale 1ns / 1ps

module hamming_decoder (
    input [6:0] received,
    output [3:0] data,
    output reg [2:0] error
);

    reg [6:0] corrected;
    reg [2:0] check;
    reg [2:0] index;

    always @(received) begin
        // Calculate check
        check[2] = received[1] ^ received[2] ^ received[3] ^ received[6];
        check[1] = received[0] ^ received[2] ^ received[3] ^ received[5];
        check[0] = received[0] ^ received[1] ^ received[3] ^ received[4];

        error = check;
        corrected = received;

        if (check != 3'b000) begin
        // If error != 0, there is a fault
            if (check == 3'b100) begin
                index = 0;
            end

            if (check == 3'b010) begin
                index = 1;
            end

            if (check == 3'b110) begin
                index = 2;
            end

            if (check == 3'b001) begin
                index = 3;
            end

            if (check == 3'b101) begin
                index = 4;
            end

            if (check == 3'b011) begin
                index = 5;
            end

            if (check == 3'b111) begin
                index = 6;
            end

            corrected[index] = ~received[index];
        end
    end

    assign data = corrected[3:0];
endmodule
```

6. Mô phỏng và kiểm tra mạch thiết kế:

Testbench:

```
`timescale 1ns/1ps

module hamming_decoder_tb;
    reg [6:0] received;
    wire [3:0] data;
    wire [2:0] error;

    hamming_decoder a1(received, data, error);

    initial
    $monitor(
        "time %t: data3=%b,data2=%b,data1=%b,data0=%b\n",
        $time,data[3],data[2],data[1],data[0]
    );

    initial begin
    received=7'b1011000; // false case
    #5 received=7'b1010101; // true case
    #5 received=7'b1100011; // true case
    #5 received=7'b1110110; // false case
    end

    initial #50 $stop;
endmodule
```

Simulation:


SIMULATION - Behavioral Simulation - Functional - sim_1 - hamming_decoder_tb
top_example.v
hamming_decoder_tb.v
Untitled 3
Scope
Sources
Objects
Name
Value
received[6:0]
76
data[3:0]
7
error[2:0]
4
50.000 ns
0 ns
10 ns
20 ns
30 ns
40 ns
58
55
63
76
a
5
3
7
2
0
4
Tcl Console
Messages
Log
Sim Time: 50 ns

7. Tài liệu tham khảo:

[1]. Wakerly, J. (2018). *Digital Design_Principles and Practices-Pearson.*

[2]. https://vi.wikipedia.org/wiki/M%C3%A3_Hamming