TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN ĐIỆN TỬ - VIỄN THÔNG

BÁO CÁO BÀI TẬP LỚN

THÔNG TIN DI ĐỘNG

Đề tài:

THỰC HIỆN ĐIỀU CHẾ, MÃ HÓA, GIẢI MÃ HÓA LDPC CHO MẠNG DI ĐỘNG 4G

Sinh viên thực hiện: VŨ THÀNH TRUNG- 20186319
NGUYỄN VIỆT HOÀNG - 20186315
VŨ HIẾU TRUNG - 20182839
Nhóm: 1
Giảng viên hướng dẫn: PGS.TS. NGUYỄN VĂN ĐỨC

Hà Nội, 1-2022

LỜI NÓI ĐẦU

Hệ thống thông tin di động thế hệ thứ 4 (4G - 4th Generation) cần đạt được cả 3 tiêu chí chính là băng thông rộng, độ tin cậy cao và độ trễ thấp. Mã kiểm tra chẵn lẻ mật độ thấp (LDPC - Low Density Parity Check) đã được chấp nhận cho hệ thống thông tin di động 4G vì mã LDPC gần đạt được dung lượng Shannon. Trong báo cáo này, chúng em mô phỏng bộ mã LDPC bằng Matlab trên kênh truyền nhiễu trắng(AWGN) với phương pháp điều chế BPSK để chứng minh tính hiệu quả của bộ mã này trong cho hệ thống thông tin di động 4G.

MỤC LỤC

DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

Kí hiệu	Giải nghĩa
LDPC	Low - density Parity Check
AWGN	White Noise Gauss
DVB	Digital Video Broadcasting
NR	New Radio
BG	Base Graph
SPC	Single Parity Check
ML	Maximum Likelihood
BPSK	Binary Phase Shift Keying

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT

1.1 Khái niệm LDPC

Mã LDPC (Low - density Parity Check) mã kiểm tra chẵn lẻ mật độ thấ , hay còn gọi là mã Gallager, được đề xuất bởi Gallager vào năm 1962. Với ưu thế chính là khả năng sửa lỗi đạt gần giới hạn Shannon trên kênh đối xứng nhị phân (BSC) cũng như trên kênh White Noise Gauss (AWGN), mã LDPC đã thu hút được nhiều sự quan từ cộng đồng nghiên cứu lẫn giới công nghệ và được ứng dụng trong các công nghệ hiện nay, điển hình như là: Wireless, DVB (Digital Video Broadcasting), 5G NR (New Radio)... Về cơ bản, mã LDPC là một loại mã khối tuyến tính sử dụng ma trận kiểm tra chẵn lẻ các ma trận thưa (Sparse Parity-check Matrix), tức là hầu hết các phần tử là 0, chỉ một số ít là 1. Theo định nghĩa của Gallager, ma trận kiểm tra chẵn lẻ của mã LDPC có đặc điểm là mỗi hàng chứa đúng i phần tử 1 và mỗi cột chứa đúng j phần tử 1. Bằng việc sử dụng phương pháp truyền tin tưởng lặp (Iterative Belief Propagation) LDPC có thể được giải mã thời gian tuyến tính theo chiều dài khối của chúng.

Mã LDPC đang được sử dụng ngày càng nhiều trong các ứng dụng yêu cầu truyền thông tin đáng tin cậy và hiệu quả cao qua các liên kết giới hạn băng thông hoặc giới hạn kênh trả về khi có sự xuất hiện của nhiễu.

1.2 Cấu trúc của mã LDPC

Về cơ bản, một mã LDPC đều là 1 loại mã khối tuyến tính mà mã trận kiểm tra chẵn lẻ H(mxn) có trọng số cột g và trọng số hàng r sao cho r = g(n/m) và g«m. Nếu H có mật độ thấp và trọng số cột, hàng không phải là hằng số, thì mã đó sẽ là mã LDPC không đều. Ví dụ dưới đây là ma trận 6x12 đều có trọng số cột là 3, trọng số hàng là 6.

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Base graph (base matrix): Là ma trận cơ sở mà từng phần tử trong ma trận có thể được mở rộng ra dựa vào hệ số mở rộng Z. Nói cách khác, ma trận cơ sở sẽ chứa các giá trị dịch chuyển P_{i_j}, -1 $\leq P_{i_j} \leq$ Z-1, với mỗi giá trị dịch chuyển P_{i_j} sẽ có thể chuyển đổi thành ma trận đơn vị I có kích thước ZxZ dịch chuyển sang phải P_{i_j} lần đối với phần tử $(i,j)^{th} \neq 0$ trong ma trận cơ sở. Ma trận hoán vị tuần hoàn nhị phân này sẽ được ký hiệu

là $Q(P_{i_j})$.

$$Q(1) = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Ví dụ trên là ma trận Q(1) dịch chuyển sang phải 1 lần. Do đó, Q(-1) để hiển thị ma trận rỗng, tất cả các phần tử bằng 0.

Từ đây, ta có thể xác định được ma trận kiểm tra chẵn lẻ dựa trên ma trận cơ sở và hệ số dịch chuyển P_{i_j} trong ma trận cơ sở. Các phần tử trong ma trận cơ sở (có giá trị từ -1 -> Z – 1) sẽ được thay thế bằng các ma trận hoán vị tuần hoàn nhị phân tương ứng theo phương pháp mở rộng bên trên.

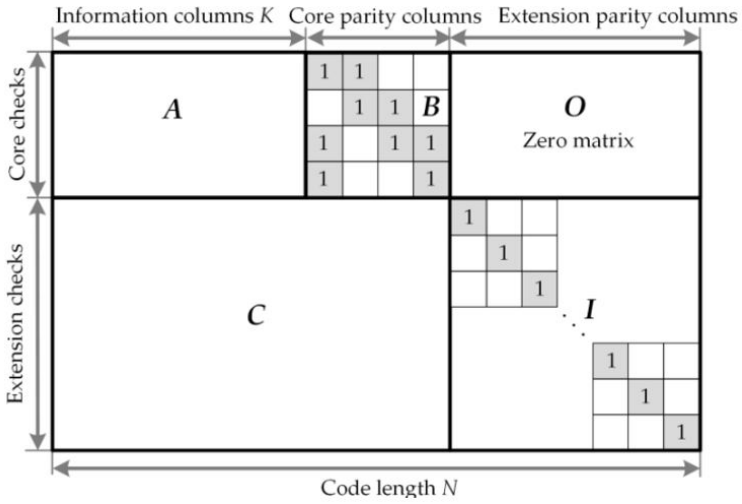
$$H = \begin{bmatrix} \mathrm{Q}(\mathrm{P}_{1,1}) & \mathrm{Q}(\mathrm{P}_{1,2}) & \cdots & \mathrm{Q}(\mathrm{P}_{1,n_b}) \\ \mathrm{Q}(\mathrm{P}_{2,1}) & \mathrm{Q}(\mathrm{P}_{2,2}) & \cdots & \mathrm{Q}(\mathrm{P}_{2,n_b}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Q}(\mathrm{P}_{m_b,1}) & \mathrm{Q}(\mathrm{P}_{m_b,2}) & \cdots & \mathrm{Q}(\mathrm{P}_{m_b,n_b}) \end{bmatrix}$$

H là ma trận kiểm tra chẵn lẻ cho ma trận có kích thước m_bxn_b

Ta có ma trận cơ sở của ma trận H như sau:

$$E(H) = \begin{bmatrix} \mathrm{P}_{1,1} & \mathrm{P}_{1,2} & \cdots & \mathrm{P}_{1,n_b} \\ \mathrm{P}_{2,1} & \mathrm{P}_{2,2} & \cdots & \mathrm{P}_{2,n_b} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{P}_{m_b,1} & \mathrm{P}_{m_b,2} & \cdots & \mathrm{P}_{m_b,n_b} \end{bmatrix}$$

Cấu trúc khối của ma trận cơ sở sẽ có dạng như hình dưới đây

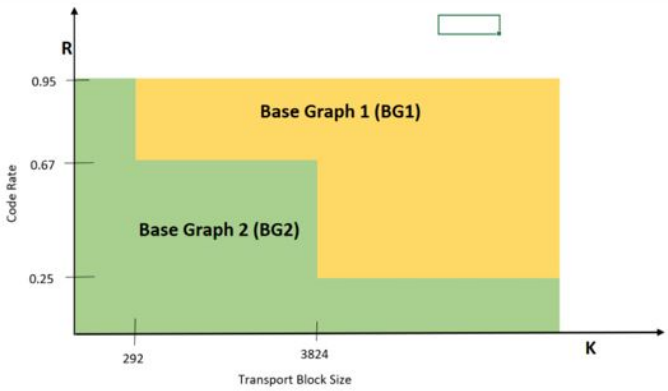


Hình 1.1 Cấu trúc của ma trận cơ sở

- Các cột trong ma trận cơ sở sẽ được chia làm 3 phần: Cột thông tin, cột chẵn lẻ cốt lõi (hay đường chéo kép) và cột chẵn lẻ mở rộng. Các hàng trong ma trận cơ sở được chia làm 2 phần: Hàng kiểm tra cốt lõi và hàng kiểm tra mở rộng. Như ở hình trên, ma trận cơ sở sẽ gồm các ma trận con A,B,O,C và I. Ma trận con A sẽ tương ứng với các bit có tính hệ thống. Ma trận con B tương ứng với tập hợp bit chẵn lẻ đầu tiên có cấu trúc đường chéo kép. Ma trận con O là ma trận 0. Các ma trận con C và I được dùng để hỗ trợ cho việc nhanh chóng yêu cầu lặp lại tự động, kết hợp gia tăng dự phòng. A và B được coi là hạt nhân, còn O, C và I được coi là phần mở rộng.

- 3GPP đưa ra 2 cấu trúc ma trận cơ sở chính cho việc mã hóa kênh là BG1 có ma trận H kích thước 46x68 và BG2 có ma trận H kích thước 42x52.

- BG1 có độ dài lớn hơn và tỷ lệ mã hóa R cao hơn. Nếu kích thước khối ≤ 292 hoặc ≤ 3824 và R $\leq$ 2/3 hoặc R $\leq$ 1/4 thì ma trận cơ sở 2, BG2, của mã LDPC được sử dụng; nếu không thì ma trận cơ sở 1, BG1, của mã LDPC được sử dụng.



Hình 1.2 Cách lựa chọn cấu trúc ma trận cơ sở theo 3GPP

- Đối với các ma trận BG1 và BG2 thì số lượng thiết kế hệ số dịch chuyển là 8. Tất cả các kích thước khác được chia thành 8 tập dựa trên tham số a, trong đó a được sử dụng để xác định hệ số mở rộng $Z = a \times 2^j$.

Bảng 1.1 Tập các hệ số mở rộng

Ma trận lũy thừa	Tập kích thước nâng
Tập 1	Z = 2 x $2^j, j = 0,1,2,3,4,5,6,7$
Tập 2	Z = 3 x $2^j, j = 0,1,2,3,4,5,6,7$
Tập 3	Z = 5 x $2^j, j = 0,1,2,3,4,5,6$
Tập 4	Z = 7 x $2^j, j = 0,1,2,3,4,5,$
Tập 5	Z = 9 x $2^j, j = 0,1,2,3,4,5$
Tập 6	Z = 11 x $2^j, j = 0,1,2,3,4,5$
Tập 7	Z = 13 x $2^j, j = 0,1,2,3,4$
Tập 8	Z = 15 x $2^j, j = 0,1,2,3,4$

- Giá trị hệ số dịch chuyển P_{ij} có thể được tính bằng cách sử dụng hàm $P_{ij} = f(V_{ij}, Z)$, trong đó V_{ij} là hệ số dịch chuyển của phần tử (i, j). Hàm f được định nghĩa như sau:

$$P_{i,j} = f(V_{i,j}, Z) = \begin{cases} -1, & \text{V}_{i,j}, Z = -1 \\ mod(\text{V}_{i,j}, Z), & \text{khác} \end{cases} \tag{1.1}$$

  Trong đó, mod là toán tử của phép module.

1.3 Cấu trúc của mã LDPC

- Thay vì sử dụng ma trận sinh G như với SPC (Single Parity Check), mã LDPC có thể được mã hóa sử dụng trực tiếp bằng ma trận chẵn lẻ H.

- Cho từ mã $C = [s, p_a, p_c]$, trong đó s biểu thị phần hệ thống, được chia thành kb nhóm gồm Z bits vì mô hình cơ sở có $k_b = n_b - m_b$ cột bit thông tin. Hơn nữa, $s = [s1, s2, ..., s_{kb}]$ trong đó mỗi phần tử của s là một vector có độ dài Z. Các bản tin nhận được bởi bộ mã hóa được lưu trữ trong các thanh ghi được sắp xếp theo khối k_b, ký hiệu là $s_i (i = 1, 2, ..., k_b)$, tương ứng với các khối hệ thống, trong đó mỗi khối bao gồm Z bit. Giả sử rằng phần chẵn lẻ của mỗi thông tin p được chia thành 2 thành phần như sau: g = 4 bit chẵn lẻ đầu tiên ,$p_a = [p_{a1}, p_{a2}, ..., p_{ag}]$ và phần còn lại gồm $(m_b - g)$ bit kiểm tra $p_c = [p_{c1}, p_{c2}, ..., p_{c(mb-g)}]$.

- Cụ thể, từ mã mã hóa được biểu diễn như sau:

$$C = [s_1, s_2, ..., skb, p_{a1}, p_{a2}, ..., p_{ag}, p_{c1}, p_{c2}, ..., p_{c(mb-g)}] \tag{1.2}$$

- Ma trận chẵn lẻ H của LDPC được chia thành 6 ma trận con:

$$H = \begin{bmatrix} A & B & 0 \\ C_1 & C_2 & I \end{bmatrix} \tag{1.3}$$

Trong đó: A là ma trận có kích thước g×kb, B là ma trận có kích thước $g \times g$, C1 là ma trận có kích thước $(m_b - g)k_b$ và C2 là ma trận có kích thước $(m_b - g)g$. Ngoài ra, I là một ma trận đơn vị có kích thước là $(m_b - g)(m_b - g)$. Việc mã hóa các mã LDPC được thực hiện bằng cách sử dụng phương trình sau:

$$HC^T = 0^T \tag{1.4}$$

- Phương trình trên được biểu thị như sau:

$$\begin{bmatrix} A & B & 0 \\ C_1 & C_2 & I \end{bmatrix} \begin{bmatrix} s \\ p_a \\ p_c \end{bmatrix} = 0^T \tag{1.5}$$

- Từ phương trình trên ta thu được hai phương trình sau:

$$As^T + Bp_a^T + 0p_a^T = 0^T \tag{1.6}$$

$$C_1 s^T + C_2 p_a^T + I p_a^T = 0^T \tag{1.7}$$

- Bước đầu tiên trong việc triển khai bộ mã hóa là xác định phần p_a. Trước tiên, phương trình (1.6) được viết lại ở dạng khối như sau:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,k_b} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,k_b} \\ a_{3,1} & a_{3,2} & \cdots & a_{3,k_b} \\ a_{4,1} & a_{4,2} & \cdots & a_{4,k_b} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ \cdots \\ s_{kb} \end{bmatrix} + \begin{bmatrix} 1 & 0 & -1 & -1 \\ 0 & 0 & 0 & -1 \\ -1 & -1 & 0 & 0 \\ 1 & -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} p_{a1} \\ p_{a2} \\ p_{a3} \\ p_{a4} \end{bmatrix} = 0 \tag{1.8}$$

- Sau đó, mở rộng phương trình (1.8) thành tập phương trình sau:

$$\sum_{j=1}^{k_b} a_{1,j} S_j + p_{a1}^{(1)} + p_{a2} = 0^T \tag{1.9}$$

$$\sum_{j=1}^{k_b} a_{2,j} S_j + p_{a1} + p_{a2} + p_{a3} = 0^T \tag{1.10}$$

$$\sum_{j=1}^{k_b} a_{3,j} S_j + p_{a3} + p_{a4} = 0^T \tag{1.11}$$

$$\sum_{j=1}^{k_b} a_{4,j} S_j + p_{a1}^{(1)} + p_{a4} = 0^T \tag{1.12}$$

- Trong đó $p_{a1}^{(1)}$ biểu thị phiên bản dịch chuyển sang phải 1 của p_{a1}.

- Bằng cách cộng tất cả các phương trình trên ta được:

$$p_{a1} = \sum_{i=1}^{4} \sum_{j=1}^{k_b} a_{(i,j)} S_j \tag{1.13}$$

- Ta có thể viết lại như sau:

$$\lambda_i = \sum_{j=1}^{k_b} a_{i,j} S_j \quad vi \quad i = 1,2,3,4 \tag{1.14}$$

$$p_{a1} = \sum_{i=1}^{4} \lambda_i \tag{1.15}$$

$$p_{a2} = \lambda_1 + p_{a1}^{(1)} \tag{1.16}$$

$$p_{a3} = \lambda_3 + p_{a4} \tag{1.17}$$

$$p_{a4} = \lambda_4 + p_{a1}^{(1)} \tag{1.18}$$

- Từ phương trình (1.14), mỗi giá trị của $lambda_i$ được tính bằng cách cộng dồn tất cả các giá trị của $a_(i,j)S_j$. Trong phép toán modulo, $lambda_i$ được tính bằng cách thực hiện phép toán XOR trên tất cả các phần tử của $a_(i,j)S_j$. Khối thứ nhất của bit chẵn lẻ p_{a1} được tính bằng cách tích lũy tất cả các giá trị $lambda_i$. Các bit chẵn lẻ còn lại có thể lấy được dựa vào phương trình (1.15)-(1.18). Tất cả các bit chẵn lẻ đầu tiên pa được lưu giữ trong thanh ghi dịch.

- Tiếp theo, ta xác định p_c dựa trên phương trình (7), trong đó ma trận C_1 và C_2 được cho bởi

$$C_1 = \begin{bmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,k_b} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,k_b} \\ \vdots & \vdots & \ddots & \vdots \\ C_{mb-g,1} & C_{mb-g,2} & \cdots & C_{mb-g,k_b} \end{bmatrix} \tag{1.19}$$

$$C_2 = \begin{bmatrix} C_{1,k_b+1} & C_{1,k_b+2} & \cdots & C_{1,k_b+g} \\ C_{2,k_b+1} & C_{2,k_b+2} & \cdots & C_{2,k_b+g} \\ \vdots & \vdots & \ddots & \vdots \\ C_{mb-g,k_b+1} & C_{mb-g,k_b+2} & \cdots & C_{mb-g,k_b+g} \end{bmatrix} \tag{1.20}$$

- Khi áp dụng phương trình (1.7), các phần tử của p_c sẽ có thể tính bằng các phương trình sau:

$$p_{c1} = \sum_{j=1}^{k_b} C_{(1,j)} S_j + \sum_{j=1}^{g} C_{(1,k_b+j)} p_{a1} \tag{1.21}$$

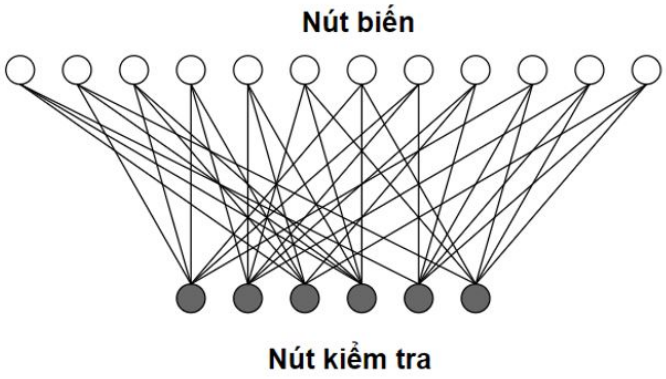
$$p_{c2} = \sum_{j=1}^{k_b} C_{(2,j)} S_j + \sum_{j=1}^{g} C_{(2,k_b+j)} p_{a1} \tag{1.22}$$

$$p_{cmb-g} = \sum_{j=1}^{k_b} C_{(mb-g,j)} S_j + \sum_{j=1}^{g} C_{(mb-g,k_b+j)} p_{a1} \tag{1.23}$$

- Sau khi tính toán xong, từ mã sẽ là sự kết hợp của bản tin ban đầu s và hai phần chẵn lẻ được tính toán p_a và p_c.

1.4 Giải mã (soft input - soft output decoder)

- Mã LDPC (N,K) là mã nhị phân được đặc trưng bởi ma trận kiểm tra chẵn lẻ thưa H_{MxN}, trong đó M = N – K có thể được biểu diễn bằng đồ hình Tanner của các nút biến $n \in \{1,...,N\}$ và các nút kiểm tra $m \in \{1,...,M\}$. Biểu thị tập hợp nút kiểm tra. $N\{m\}$ biểu thị tập hợp các nút biến được kết nối với một nút kiểm tra m nào đó. Một nút biến được kết nối với nút kiểm tra m nếu $n \in N\{m\}$. Ngoài ra, tập $N\{m\}$ \n biểu thị tập các nút biến được kết nối với nút kiểm tra m mà không bao gồm n. Tương tự, tập các nút kiểm tra đối với một nút biến nào đó n được ký hiệu là $M\{n\}$. Một nút kiểm tra được kết nối với nút biến n nào đó nếu $m \in M\{n\}$. Tập hợp $M\{n\}$ \m biểu thị tập hợp các nút kiểm tra được kết nối với nút biến n loại trừ m.



Hình 1.3 Tanner graph cho ma trận kiểm tra chẵn lẻ

- Sum-product là tên chung cho một lớp thuật toán giải mã Maximum Likelihood (ML). Thuật toán sử dụng thông tin kênh truyền và các giá trị từ kênh truyền. Thuật toán tạo ra một giá trị xác xuất cho mỗi bit nhận được và làm mới giá trị này sau nhiều lần lặp để tìm ước lượng cho bit đó.

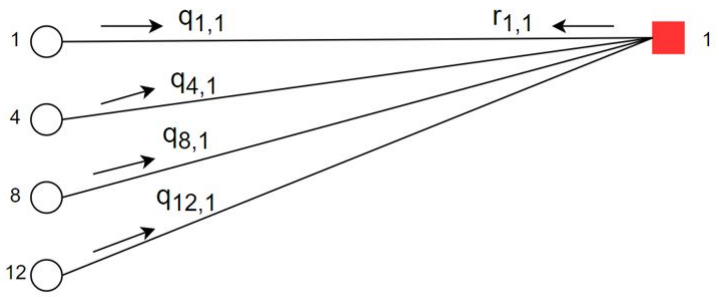
- Sum-product xử lý lặp đi lặp lại các bit nhận được theo các bước nối liền nhau có thể nhìn thấy trên đồ hình Tanner để cải thiện độ tin cậy mỗi bit được giải mã.

- Để minh họa, hãy xét tin độ cậy một bit được giải mã được đo bằng xác xuất $P(x_N|Y), 1 \leq n \leq N$. Sau đó Log-Likelihood Ratio (LLR) của mỗi bit mã được tính bởi công thức:

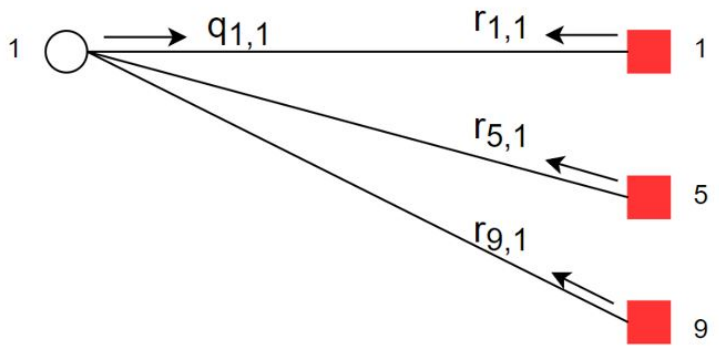
$$L(x_N) = log(\frac{P(x_N = 0|Y)}{P(x_N = 1|Y)}) \tag{1.24}$$

- Trong mỗi lần lặp lại, một giá trị $r_{m \to n}$ được tính tại mỗi nút kiểm tra m theo chiều ngang và được chuyển cho tất cả các nút biến n nếu $n \in N\{m\}$. Tương tự, mỗi nút biến n sẽ gửi một giá trị $q_{n \to m}$ được tính theo chiều dọc đến tất cả các nút kiểm tra m nếu $n \in M\{n\}$.

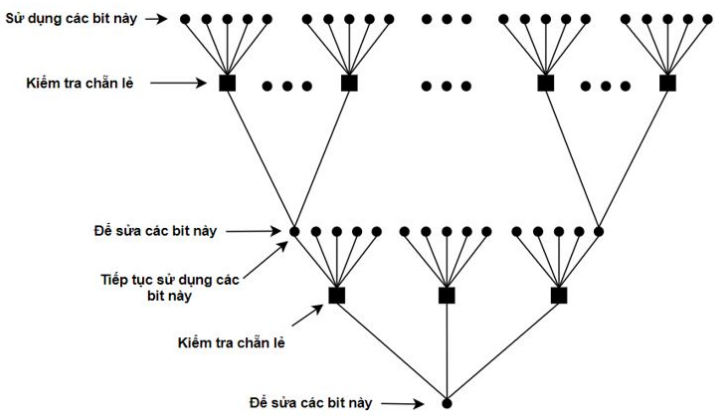
- Cần lưu ý rằng, việc tính toán sẽ được thực hiện tại các nút kiểm tra trước, rồi sau đó mới đến các bit node.



Hình 1.4 Nút kiểm tra 1 nhận các giá trị $q_{(n \to m)}$ từ các bit node và tính toán trả lại giá trị $r_{(m \to n)}$ đến các bit node tương ứng

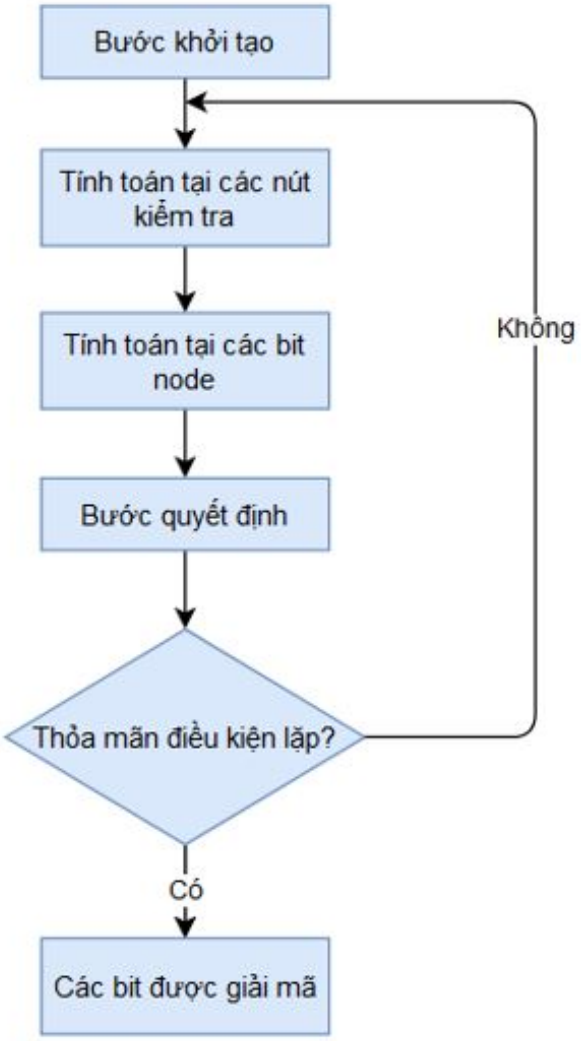


Hình 1.5 Bit node 1 nhận các giá trị $r_{(m \to n)}$ từ các node kiểm tra, rồi tính toán và tính toán lại các giá trị $q_{(n \to m)}$ đến các node kiểm tra tương ứng



Hình 1.6 Tổng quan hơn về việc sửa lỗi bit dựa trên Tanner graph (Soft Decision Decoding)

- Từ mã ta sẽ ký hiệu là $X = [x_1, x_2, ..., x_N]$, trong đó $x_N \in \{0, 1\}$. Các giá trị LLR của vector nhận được tương ứng được biểu thị bằng $Y = [y_1, y_2, ..., y_N]$.
- Quá trình giải mã sử dụng thuật toán sum-product có thể được thực hiện theo các bước liên tiếp theo hình dưới đây.



Hình 1.7 Giải thuật giải mã Sum-Product

1. Bước khởi tạo: Các giá trị ban đầu của LLR có thể nhận được từ đầu ra của bộ giải điều chế y_n. Các giá trị này được sử dụng làm $q_{n\to m}$ của lần lặp đầu tiên cho tới bước cập nhập nút kiểm tra.
2. Tính toán tại các nút kiểm tra: Tại từng nút kiểm tra m, xử lý các giá trị đến từ các nút biến (bit node) $q_{n\to m}$ để tính toán các giá trị trả lời $r_{m\to n}$ cho mọi $n \in N\{m\}$. Vì vậy, đối với nút kiểm tra m:

$$r_{(m\to n)} = (\prod_{n\in N(m)\backslash n} sgn(q_{n\to m}) \times 2tanh^{-1}(\prod_{n\in N(m)\backslash n} tanh(\frac{|q_{n\to m}|}{2})) \quad (1.25)$$

3. Tính toán tại các nút kiểm tra: Tại các nút biến n, xử lý các giá trị đến từ các nút kiểm tra $r_{m\to n}$ để tính toán các giá trị trả lời $q_{n\to m}$ cho mọi $m \in N\{n\}$. Vì vậy, đối với nút biến n:

$$q_{n\to m} = y_n + \sum_{m\in M(n)\backslash m} r_{m\to n}(x_n) \quad (1.26)$$

4. Bước quyết định: Đối với mỗi nút biến, các giá trị LLR được cập nhập theo công thức:

$$L(x_n) = y_n + \sum_{m \in M(n)} r_{m \to n}(x_n) \tag{1.27}$$

Các giá trị LLR được áp dụng cho được áp dụng quyết định cứng để quyết định giá trị trả về. Ví dụ với kênh truyền BPSK thì 0 sẽ là ngưỡng để quyết định xem đó là -1 hay 1. Các giá trị LLR này sẽ được lặp đi lặp lại cho đến khi thỏa mãn điều kiện lặp. Điều kiện lặp có thể đạt được khi tiến hành kiểm tra Syndrome Hx^T bằng 0 hoặc thỏa mãn điều kiện lặp tối đa. Như vậy quá trình lặp sẽ tiếp tục cho đến khi từ mã được giải thành công hoặc số lần lặp tối đa đã hết.

CHƯƠNG 2. MÔ PHỎNG MATLAB

2.1 Thuật toán

2.1.1 Giải thuật Min-sum Product

Để cải thiện tốc độ giải mã ta có thể chỉnh sửa lại thuật toán Sum-Product nhằm giảm độ phức tạp khi thực hiện của bộ giải mã.

Giải thuật Min-Sum Product sẽ có các bước và các phương trình giống như với giải thuật Sum-Product. Tuy nhiên, sẽ có sự thay đổi ở phương trình (1.24). Phương trình (1.24) sẽ được viết lại thành:

$$r_{m\to n} = (\prod_{n\in N(m)\backslash n} sgn(q_{n\to m}) \times \min|q_{n\to m}| \tag{2.28}$$

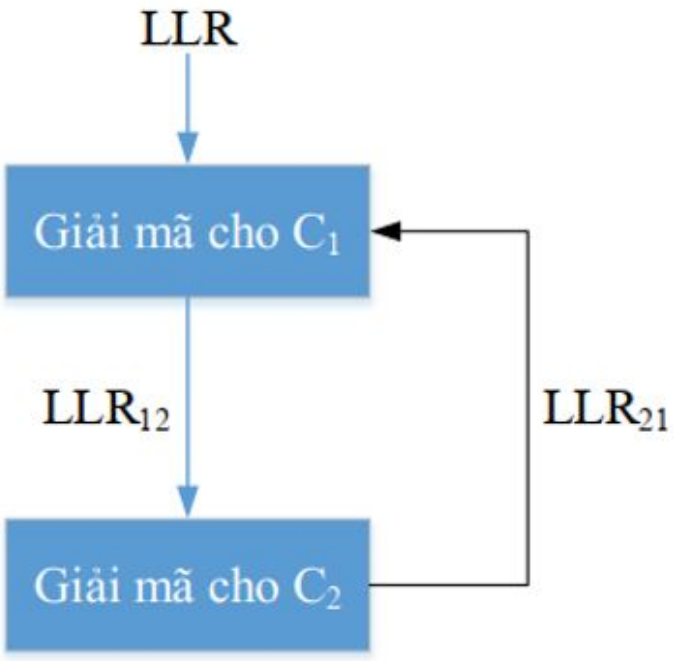
2.1.2 Kỹ thuật phân lớp

Để cải thiện hiệu năng giải mã của thuật toán sum-product, ta sẽ áp dụng kỹ thuật phân lớp. Kỹ thuật này giúp bộ giải mã của ta đạt được thông lượng giải mã hiệu quả cao với độ phức tạp tính toán thấp.

Đối với mỗi lần lặp, chúng ta sẽ tính toán nút kiểm tra và nút biến trong một lớp. Việc giải mã sau đó diễn ra tuần tự. Điều này có nghĩa là ta sẽ tập hợp một số hàng của ma trận kiểm tra chẵn lẻ thành một lớp. Ma trận H sẽ được phân lớp như sau:

$$H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_{N-K} \end{bmatrix} \tag{2.29}$$

trong đó mỗi hàng trong ma trận H là một lớp.

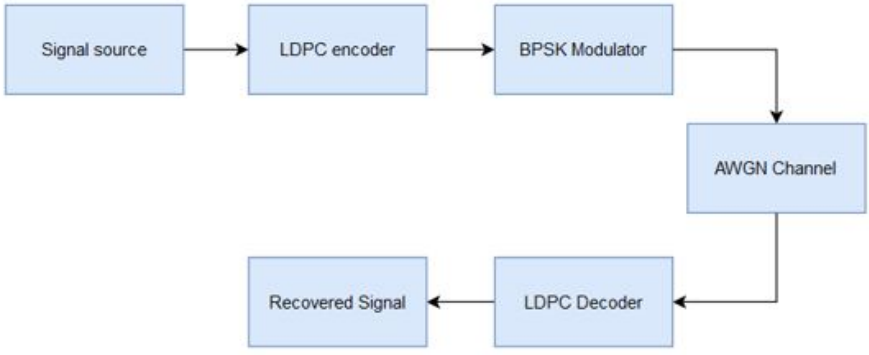


Hình 2.1 Thuật toán giải mã với kỹ thuật phân lớp

Hình trên minh họa sơ đồ giải mã của kỹ thuật phân lớp khi có 2 lớp: C_1 là từ mã được mã hóa từ H_1 và C_2 là từ mã được mã hóa từ H_2. Việc giải mã C1 sẽ sử dụng LLR trong vòng lặp đầu tiên, sau đó sẽ sử dụng LLR_{21} sau khi đã cập nhập cột.

2.2 Mô phỏng

Mô phỏng cho hệ thống truyền thông được điều chế BPSK (Binary Phase Shift Keying) với nhiễu Gaussian trắng cộng (Additive White Gaussian Noise – AWGN) theo sơ đồ dưới đây



Hình 2.2 Mô hình thực hiện việc điều chế, mã hóa, giải mã hóa với LDPC

- Ở đây ta sẽ thực hiện mô phỏng kênh truyền khi không thực hiện mã hóa LDPC và mã hóa LDPC với các lần lặp khác nhau.
- Các hình 14, hình 15 và hình 16 là kết quả của việc thực hiện mã hóa LDPC với lần

lặp là 8.

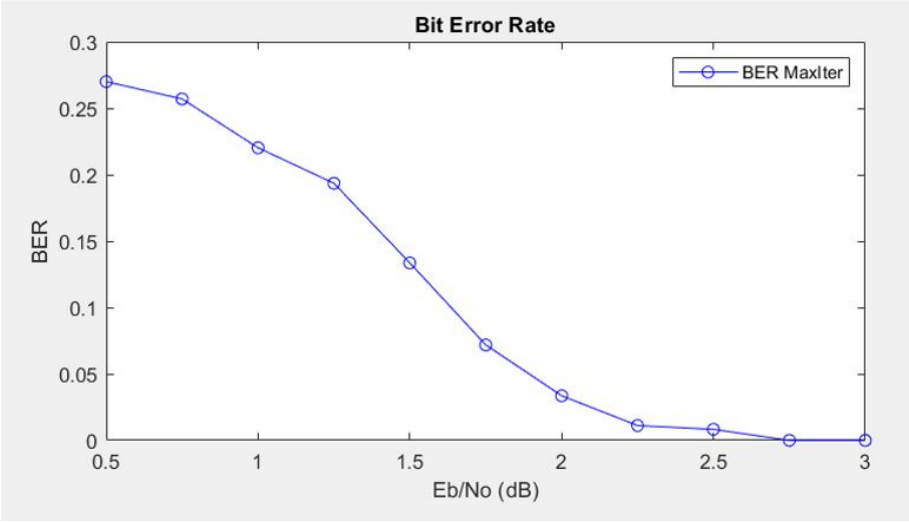
- Các hình 17 và hình 18 là kết quả so sánh giữa việc không thực hiện mã hóa LDPC và mã hóa LDPC với các lần lặp khác nhau.
- Việc mã hóa và giải mã hóa ở đây sẽ được thực hiện với ma trận BG1 với hệ số mở rộng Z là 16 và 100 khối bit truyền.

Code matlab được trình bày ở phần Phụ lục.

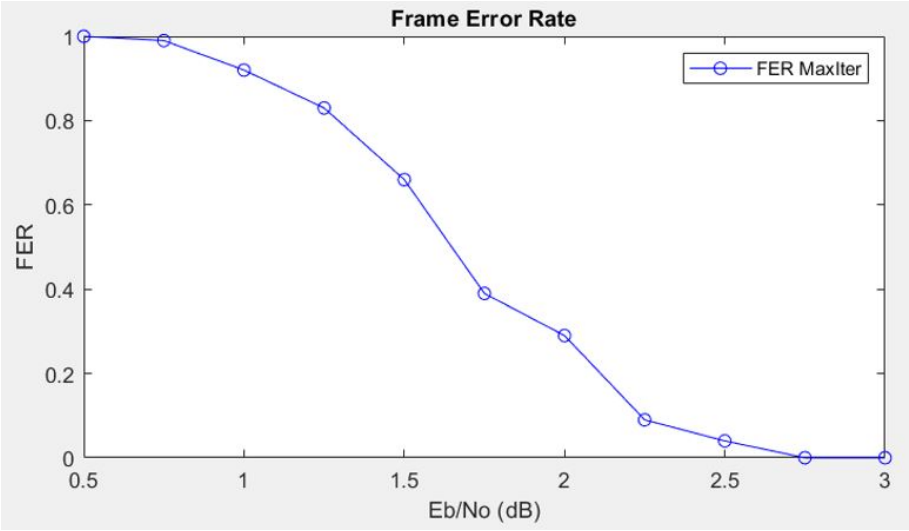
2.3 Kết quả

dB	FER	BER	Nblkerrs	Nbiterrs	Nblocks
0.5	1	0.27003	100	9505	100
0.75	0.99	0.25699	99	9046	100
1	0.92	0.22028	92	7754	100
1.25	0.83	0.19355	83	6813	100
1.5	0.66	0.13358	66	4702	100
1.75	0.39	0.07179	39	2527	100
2	0.29	0.033551	29	1181	100
2.25	0.09	0.011165	9	393	100
2.5	0.04	0.0082955	4	292	100
2.75	0	0	0	0	100

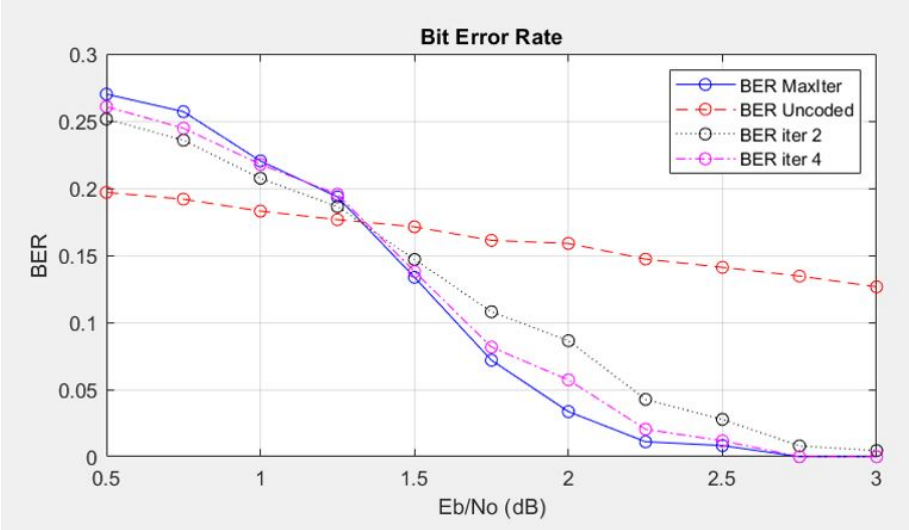
Hình 2.3 Kết quả mô phỏng với số lần lặp tối đa là 8



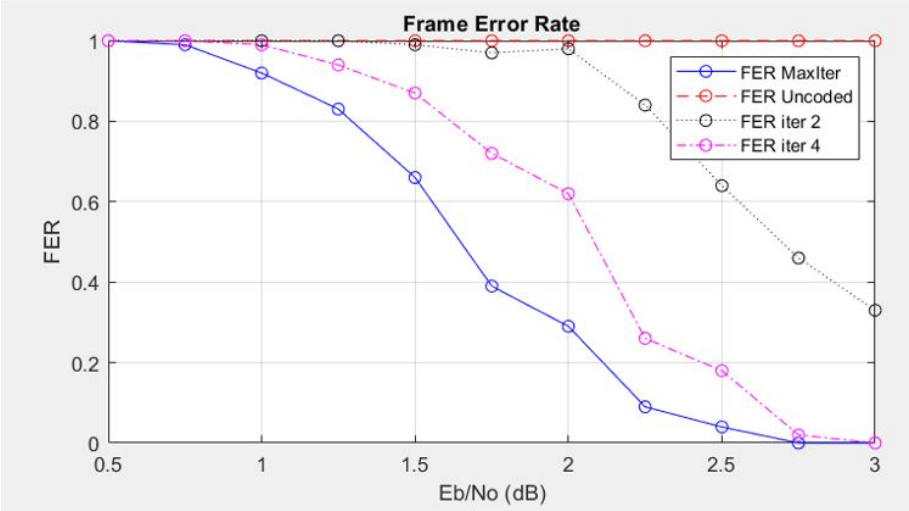
Hình 2.4 BER theo Eb/N0¬ với số lần lặp tối đa là 8



Hình 2.5 FER theo Eb/N0¬ với số lần lặp tối đa là 8



Hình 2.6 BER theo Eb/N0 với các trường hợp khác nhau



Hình 2.7 FER theo Eb/N0 với các trường hợp khác nhau

Từ việc mô phỏng trên, ta có thể đưa ra một số nhận xét cụ thể như sau:

- BER và FER sẽ giảm khi tỷ số Eb/N0 càng lớn
- Việc thực hiện mã hóa LDPC sẽ giúp cho tỷ lệ BER và FER giảm đi đáng kể so với việc không mã hóa.
- Số lần lặp càng lớn thì BER và FER càng giảm theo chiều tăng của Eb/N0.

THẢO LUẬN

Nhóm em xin được trả lời một số câu hỏi của thầy đưa ra đối với đề tài này

1. Tại sao phải sử dụng ma trận thưa thớt?
   → Sử dụng ma trận thưa thớt sẽ giúp tăng tốc độ xử lý và tăng khả năng lưu trữ của bộ nhớ bởi ma trận thưa thớt sẽ chỉ lưu trữ tọa độ của các giá trị khác 0 và việc tính toán cũng sẽ chỉ dựa trên trên các giá trị khác 0 này.

2. Mục đích của việc sử dụng ma trận cơ sở cùng với hệ số mở rộng?
   → Hệ số mở rộng giúp mở rộng ma trận cơ sở thành ma trận thưa thớt. Ma trận cơ sở khi mở rộng sẽ có không gian ma trận khá lớn, vì vậy để việc lưu trữ dễ hơn, người ta sử dụng ma trận cơ sở và hệ số mở rộng.

PHỤ LỤC

Mã code mô phỏng

```
1.        Hàm dịch ma trận theo khối vào:
function y = mul_sh(x,k)
%x: Khoi vao
%k: -1 hoac so lan dich
%y: Khoi ra
if (k==-1)
    y = zeros(1,length(x));
else
    y = [x(k+1:end) x(1:k)];
end
```

2. Hàm Encoding cho từng vector message đầu vào:

```
function cword = ldpc_encode(B,z,msg)
%B: Ma tran co so
%z: He so mo rong
%msg: message vector, length = (#cols(B)-#rows(B))*z
%cword: codeword vector, length = #cols(B)*z

[m,n] = size(B);

cword = zeros(1,n*z);
cword(1:(n-m)*z) = msg;

%Ma hoa duong cheo kep
temp = zeros(1,z);
for i = 1:4 %row 1 to 4
    for j = 1:n-m %message columns
        temp = mod(temp + mul_sh(msg((j-1)*z+1:j*z),B(i,j)),2);
    end
end
if B(2,n-m+1) == -1
    p1_sh = B(3,n-m+1);
else
    p1_sh = B(2,n-m+1);
end
cword((n-m)*z+1:(n-m+1)*z) = mul_sh(temp,z-p1_sh); %p1
%Tim p2, p3, p4
for i = 1:3
    temp = zeros(1,z);
    for j = 1:n-m+i
        temp = mod(temp + mul_sh(cword((j-1)*z+1:j*z),B(i,j)),2);
    end
    cword((n-m+i)*z+1:(n-m+i+1)*z) = temp;
end
%Cac bit chan le con lai
for i = 5:m
    temp = zeros(1,z);
    for j = 1:n-m+4
        temp = mod(temp + mul_sh(cword((j-1)*z+1:j*z),B(i,j)),2);
    end
    cword((n-m+i-1)*z+1:(n-m+i)*z) = temp;
end
```

```
3.        Decoding:
clear
close all
format short g

MaxItrs = 8;

%Basegraphnumber_j_z
load base_matrices/NR_1_0_16.txt
B = NR_1_0_16;
[mb,nb] = size(B);
z = 16;

Slen = sum(B(:)~=-1); %So luong cac gia tri khac -1 trong B
min_reg = zeros(max(sum(B ~= -1,2)),z); %Thanh ghi luu cac gia tri minsum
k = (nb-mb)*z; %So cot bit message trong B
n = nb*z; %So cot bit codeword trong B
Rate = k/n;  %Toc do


dB=[0.5:0.25:3];              % Khoang SNR chay theo dB
FER=zeros(1,length(dB));      % Mang luu tru Frame Error Rate
BER=zeros(1,length(dB));      % Mang luu tru Bit Error Rate

FER_uncoded=zeros(1,length(dB));% Mang luu tru Frame Error Rate khong ma hoa
BER_uncoded=zeros(1,length(dB));% Mang luu tru Bit Error Rate khong ma hoa

FER_iter_2=zeros(1,length(dB));% Mang luu tru Frame Error Rate voi lan lap 2
BER_iter_2=zeros(1,length(dB));% Mang luu tru Bit Error Rate voi lan lap 2

FER_iter_4=zeros(1,length(dB));% Mang luu tru Frame Error Rate voi lan lap 4
BER_iter_4=zeros(1,length(dB));% Mang luu tru Bit Error Rate voi lan lap 4

EbNo=10.^(dB/10);   % Eb/No doi tu dB thanh thap phan



Nblocks = 100;   % So luong Block Message mo phong
disp('  dB   FER   BER    Nblkerrs   Nbiterrs    Nblocks')
for g=1:length(EbNo) % Vong lap de test trong khoang SNR
Nbiterrs = 0; Nblkerrs = 0;
    Nbiterrs_uncoded = 0; Nblkerrs_uncoded = 0;
    Nbiterrs_iter_2 = 0; Nblkerrs_iter_2 = 0;
    Nbiterrs_iter_4 = 0; Nblkerrs_iter_4 = 0;
    sigma = sqrt(1/(2*Rate*EbNo(g)));
    for i = 1: Nblocks
        msg = randi([0 1],1,k); %Tao k-bit message ngau nhien
        %msg = zeros(1,k); %all-zero message
        %cword = zeros(1,n); %all-zero codeword

        %Encoding
        cword = ldpc_encode(B,z,msg);
        cword = cword(1:n);
        s = 1 - 2 * cword; %Ma hoa BPSK
        r = s + sigma * randn(1,n); %AWGN channel I
```

```
%Soft-decision, Layer decoding
        L = r; %Tong cac belief
        itr = 0; %so lan lap hien tai
        R = zeros(Slen,z); %Thanh ghi chua du lieu xu ly tren hang

        % Thuc hien Layer Decoding
        while itr < MaxItrs
            Ri = 0;
            for lyr = 1:mb %lyr la layer, mac dinh la 1
                ti = 0; %so luong cac gia tri khac -1 tai row=lyr
                for col = find(B(lyr,:) ~= -1)
                       ti = ti + 1;
                       Ri = Ri + 1;
                       %Subtraction (Thuc hien phep tru)
                       L((col-1)*z+1:col*z) = L((col-1)*z+1:col*z)-R(Ri,:);
                       %can chinh lai gia tri hang va luu tru trong min_reg
                     min_reg(ti,:) = mul_sh(L((col-1)*z+1:col*z),B(lyr,col));
                end
                %minsum tai min_reg: ti x z
                for i1 = 1:z %min_reg(1:ti,i1)
                    [min1,pos] = min(abs(min_reg(1:ti,i1)));
                    %Cuc tieu thu nhat theo gia tri tuyet doi
                    min2 = min(abs(min_reg([1:pos-1 pos+1:ti],i1)));
                    %Cuc tieu thu 2 theo gia tri tuyet doi
                    S = sign(min_reg(1:ti,i1)); %Luu tru dau cua 1 hang
                    parity = prod(S); %Tich cac dau cua 1 hang
                    min_reg(1:ti,i1) = min1;
                    %Thay the thanh gia tri tuyet doi min1
                    min_reg(pos,i1) = min2;
                %Thay the cho vi tri cua min1 thanh gia tri tuyet doi min2
                    min_reg(1:ti,i1) = parity*S.*min_reg(1:ti,i1);
                    %Nhan them dau cua hang
                end
                %Thuc hien hinh lai gia tri cot, addition va luu tru trong R
                Ri = Ri - ti; %reset Ri
                ti = 0;
                for col = find(B(lyr,:) ~= -1)
                       Ri = Ri + 1;
                       ti = ti + 1;
                       %Can chinh gia tri cot
                       R(Ri,:) = mul_sh(min_reg(ti,:),z-B(lyr,col));
                       %Addition (Thuc hien phep cong)
                       L((col-1)*z+1:col*z) = L((col-1)*z+1:col*z)+R(Ri,:);
                end
            end
            if itr == 2
               msg_iter_2 = L(1:k)<0;
            end
            if itr == 4
               msg_iter_4 = L(1:k)<0;
            end
```

```
            itr = itr + 1;
        end
        msg_cap = L(1:k) < 0; %Quyet dinh
        uncoded_msg = r(1:k) < 0;
        %Dem loi
        Nerrs_uncoded = sum(msg ~= uncoded_msg);
        Nerrs_iter_2 = sum(msg ~= msg_iter_2);
        Nerrs_iter_4 = sum(msg ~= msg_iter_4);
        Nerrs = sum(msg ~= msg_cap);
        if Nerrs > 0
            Nbiterrs = Nbiterrs + Nerrs;
            Nblkerrs = Nblkerrs + 1;
        end
        if Nerrs_uncoded > 0
            Nbiterrs_uncoded = Nbiterrs_uncoded + Nerrs_uncoded;
            Nblkerrs_uncoded = Nblkerrs_uncoded + 1;
        end
        if Nerrs_iter_2 > 0
            Nbiterrs_iter_2 = Nbiterrs_iter_2 + Nerrs_iter_2;
            Nblkerrs_iter_2 = Nblkerrs_iter_2 + 1;
        end
        if Nerrs_iter_4 > 0
            Nbiterrs_iter_4 = Nbiterrs_iter_4 + Nerrs_iter_4;
            Nblkerrs_iter_4 = Nblkerrs_iter_4 + 1;
        end
    end

    BER(g) = Nbiterrs/k/Nblocks;
    FER(g) = Nblkerrs/Nblocks;

    BER_uncoded(g) = Nbiterrs_uncoded/k/Nblocks;
    FER_uncoded(g) = Nblkerrs_uncoded/Nblocks;

    BER_iter_2(g) = Nbiterrs_iter_2/k/Nblocks;
    FER_iter_2(g) = Nblkerrs_iter_2/Nblocks;

    BER_iter_4(g) = Nbiterrs_iter_4/k/Nblocks;
    FER_iter_4(g) = Nblkerrs_iter_4/Nblocks;

    %disp([EbNodB FER_sim BER_sim Nblkerrs Nbiterrs Nblocks])
    disp([dB(g) FER(g) BER(g) Nblkerrs Nbiterrs Nblocks])
end

subplot(2,2,1)
plot(dB,BER,'b-o')
legend('BER MaxIter ');
title('Bit Error Rate')
ylabel('BER')
xlabel('Eb/No (dB)')
```

```
subplot(2,2,2)
plot(dB,BER,'b-o')
hold on;
plot(dB,BER_uncoded,'r--o')
plot(dB,BER_iter_2,'k:o')
plot(dB,BER_iter_4,'m-.o')
legend('BER MaxIter ','BER Uncoded','BER iter 2','BER iter 4');
title('Bit Error Rate')
ylabel('BER')
xlabel('Eb/No (dB)')
grid
%figure
hold off;

subplot(2,2,3)
plot(dB,FER,'b-o')
legend('FER MaxIter ');
title('Frame Error Rate')
ylabel('FER')
xlabel('Eb/No (dB)')

subplot(2,2,4)
plot(dB,FER,'b-o')
hold on;
plot(dB,FER_uncoded,'r--o')
plot(dB,FER_iter_2,'k:o')
plot(dB,FER_iter_4,'m-.o')
legend('FER MaxIter ','FER Uncoded','FER iter 2','FER iter 4');
title('Frame Error Rate')
ylabel('FER')
xlabel('Eb/No (dB)')
grid
```