Cairo University, Faculty of Engineering,
Electronics and Electrical Communications
Department (EECE)
Fourth Year, First Term
2025/2026

Advanced Digital Communication
Information Theory

Report1_ELC4020

Name	ID	SEC
إبراهيم رمضان إبراهيم	9220009	1
احمد خلف محمد على	9220036	1
احمد محمد إبراهيم	9220078	1
خالد محمد رمضان على	9221589	2
محمد خالد محمد شتا	9203230	3

Submitted to: Dr. Mohamed Nafie

& Eng. Mohamed Khaled

Submission date: 18-12-2025

Table of Contents

Table of Figures

Introduction

In modern digital communication systems, reliable transmission of information over noisy channels is a fundamental challenge. Channel coding techniques play a critical role in achieving this reliability by introducing controlled redundancy into the transmitted signal, enabling the detection and correction of errors caused by noise, interference, or fading. This report investigates several fundamental error correction coding schemes and their impact on the bit error rate (BER) performance in an Additive White Gaussian Noise (AWGN) channel using Binary Phase Shift Keying (BPSK) modulation.

The study begins with uncoded BPSK as a baseline, followed by an evaluation of simple repetition coding (rate 1/3) with both hard and soft decision decoding under two energy allocation scenarios: constant energy per transmitted bit and constant energy per information bit. These cases highlight the trade-offs between power efficiency, bandwidth expansion, and error performance. The report then examines linear block codes, specifically the (7,4) and (15,11) Hamming codes, analyzing their coding gain, minimum distance, and suitability for power-limited or bandwidth-constrained systems.

Additionally, a higher-order modulation comparison is presented between uncoded Quadrature Phase Shift Keying (QPSK) and 16-Quadrature Amplitude Modulation (16-QAM) combined with a powerful (255,131) BCH code, demonstrating how channel coding can compensate for the increased noise sensitivity of higher-order constellations while improving spectral efficiency.

Finally, a rate-1/3 convolutional code with constraint length 3 is implemented to illustrate the structure and operation of convolutional encoding.

All simulations are performed using MATLAB, with BER curves plotted against Eb/N0 to quantitatively assess performance. The results provide insight into the practical benefits and limitations of each coding scheme, guiding the selection of appropriate techniques based on system constraints such as power, bandwidth, complexity, and required reliability.

This work serves as a practical exploration of key concepts in channel coding within the framework of advanced digital communications.

1. BPSK with No Coding

MATLAB Implementation Code

```
clc; clear; close all;

%% Parameters
A = 1;
Eb = A^2;
EbN0_db = -3 : 0.5 : 10;  % SNR list in dB
EbN0 = 10.^(EbN0_db/10);  % Linear values

num_bits = 110000;
Bits = randi([0,1], 1, num_bits);  % Generate random bits to be transmitted

%% BPSK
Symbols = A * (2*Bits - 1);        % Mapping: 0 -> -1 & 1 -> +1

% Initialize variables
BER_BPSK = zeros(size(EbN0_db));
BER_BPSK_Theo = zeros(size(EbN0_db));  % Theoretical value

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));    % σ = √N0/2

    N0 = 2 * (sigma.^2);

    % Channel
    Noise = randn(1, num_bits) .* sigma;  % Generate random noise
    Rx_Signal = Symbols + Noise;          % Noisy received signal

    % Demapper, Decision threshold = 0
    Rx_Bits = Rx_Signal > 0;  % Received bit= 0 if signal <= 0 & 1 if signal > 0

    % BER Calculation
    ErrorBits = sum(Rx_Bits ~= Bits);
    BER_BPSK(i) = ErrorBits / num_bits;

    BER_BPSK_Theo(i) = 0.5 * erfc(sqrt(Eb/N0));
end

% Plotting BER for each SNR value
semilogy(EbN0_db, BER_BPSK, 'b', 'LineWidth', 2); hold on ;
semilogy(EbN0_db, BER_BPSK_Theo, 'r --', 'LineWidth', 2); grid on;
ylim([1e-6 1e0]);
title('BPSK Without Coding');
xlabel('Eb/N0 (dB)'); ylabel('BER');
legend('Simulated BER', 'Theoretical BER');
```

Output:

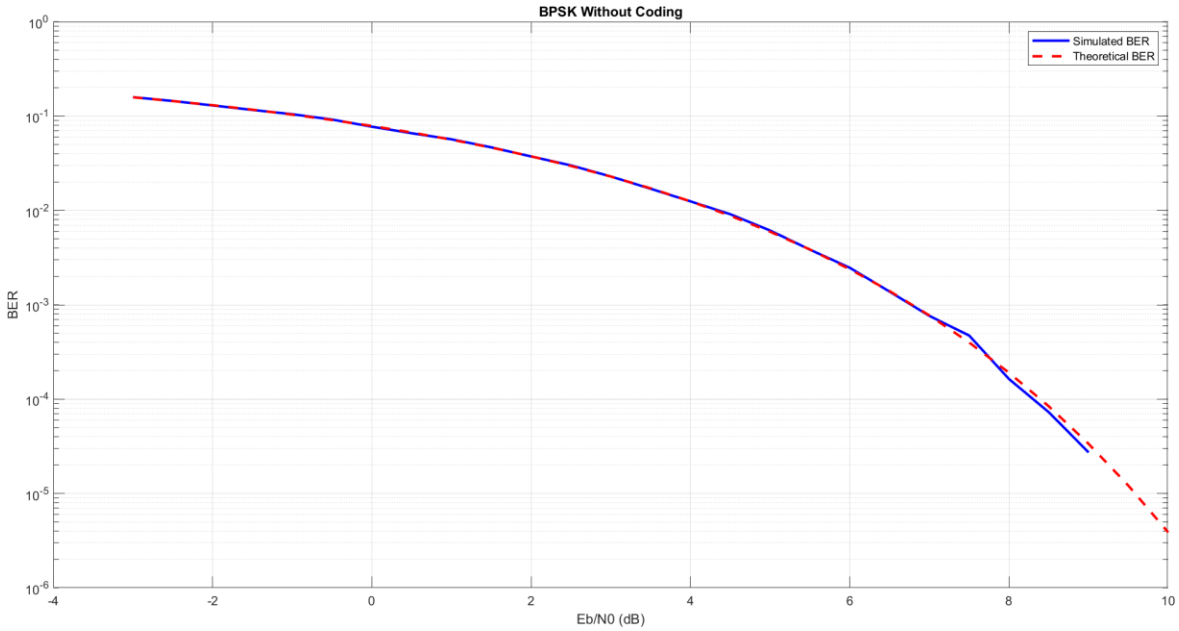

Figure 1: No coding BPSK, BER vs Eb/No

2. BPSK with 3 Bit Repetition Hard Decision Coding

MATLAB Implementation Code

```
clc; clear; close all;

%% Parameters
A = 1;
Eb = A^2;                                   % Energy per information bit
EbN0_db = -3 : 0.5 : 10;                    % Eb/N0 values in dB
EbN0 = 10.^(EbN0_db/10);                    % Linear values

num_bits = 110000;
Bits = randi([0,1], 1, num_bits);           % Generate random information bits

%% BPSK without coding (reference curve from Q1)
Symbols_uncoded = A * (2*Bits - 1);         % Mapping: 0 -> -A, 1 -> +A

% Initialize variables
BER_uncoded = zeros(size(EbN0_db));
BER_uncoded_Theo = zeros(size(EbN0_db));

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));         % σ = √(N0/2)

    % Channel
```

```
    Noise = randn(1, num_bits) .* sigma;
    Rx_Signal = Symbols_uncoded + Noise;

    % Demapper (hard decision)
    Rx_Bits = Rx_Signal > 0;

    % BER Calculation
    BER_uncoded(i) = sum(Rx_Bits ~= Bits) / num_bits;

    % Theoretical BER
    BER_uncoded_Theo(i) = 0.5 * erfc(sqrt(EbN0(i)));
end

%% BPSK with repetition-3 coding and hard decision decoding
rep_factor = 3;                          % Repetition factor for coding
coded_bits = repelem(Bits, rep_factor);  % Repeat each bit 3 times

% Initialize BER arrays
BER_same_E_per_Tx_Bit = zeros(size(EbN0_db));
BER_same_E_per_Info_Bit = zeros(size(EbN0_db));

%% (a) Same energy per transmitted bit (amplitude A, like uncoded)
Symbols_same_E_Tx = A * (2*coded_bits - 1);     % Same amplitude as uncoded

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Tx + Noise;

    % Hard decision on each repeated symbol
    Rx_Codes = Rx_Signal > 0;

    % Majority decoding: at least 2 out of 3
    Rx_Codes_reshaped = reshape(Rx_Codes, rep_factor, []);
    Rx_Bits_decoded = sum(Rx_Codes_reshaped, 1) >= 2;

    % BER Calculation
    BER_same_E_per_Tx_Bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

%% (b) Same energy per information bit
A_info = A / sqrt(rep_factor);           % Reduce amplitude to keep same total energy
Symbols_same_E_Info = A_info * (2*coded_bits - 1);

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Info + Noise;

    % Hard decision on each repeated symbol
    Rx_Codes = Rx_Signal > 0;
```

```
    % Majority decoding: at least 2 out of 3
    Rx_Codes_reshaped = reshape(Rx_Codes, rep_factor, []);
    Rx_Bits_decoded = sum(Rx_Codes_reshaped, 1) >= 2;

    % BER Calculation
    BER_same_E_per_Info_Bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

% Plotting
semilogy(EbN0_db, BER_same_E_per_Info_Bit, 'b', 'LineWidth', 2); hold on;
semilogy(EbN0_db, BER_same_E_per_Tx_Bit, 'k', 'LineWidth', 2);
semilogy(EbN0_db, BER_uncoded, 'r--', 'LineWidth', 2); grid on;
ylim([1e-6 1e0]);
title('BPSK with Repetition-3 Coding (Hard Decision Decoding)');
xlabel('Eb/N0 (dB)'); ylabel('BER');
legend('Same Energy per Information Bit', ...
       'Same Energy per Transmitted Bit', 'Without Coding');
```

Output:

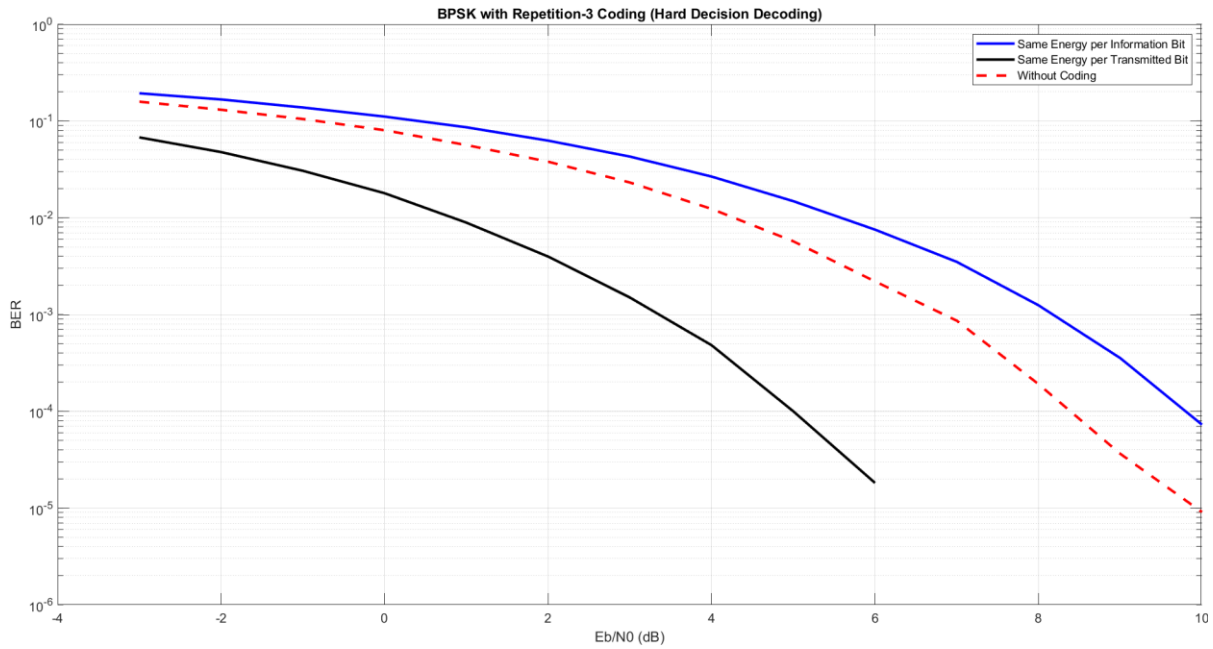

Figure 2: Hard Decision BPSK with 3-bit repetition, BER vs Eb/No

Comment:

As seen from *Figure 2,* In Hard decision decoding:

1. The best performance is BPSK with the same energy per transmitted bit. But as the energy of each bit of the three bits equal to the energy of the original bit so it costs tripling the energy expended to transmit one information bit.

2. On the other hand, the worst performance is BPSK with the same energy per information bit even it is worse than the uncoded BPSK in performance ,as in this case the energy of each bit of the three bits equal to one third of the energy of the original bit and as we know as the value of energy decreases the value of BER increases which leads to bad performance.

   - **If you are designing a communication system, can you think of a scenario where you will use such an error correction coding scheme as in (a) or (b)?**
     - As we discussed above BPSK with the same energy per information bit (b) is even worse than uncoded BPSK and the best performance is BPSK with the same energy per transmitted bit (a) but it is not also the greatest performance so the only scenario of using this system is using (a) but with very simple hardware for transmitter and receiver and fine BER.
     - All of this shows that repetition code with Hard decision decoding is not recommended as it has low performance compared to other coding so it can be used in simple Communication systems.

3. BPSK with 3 Bit Repetition Soft Decision Coding

MATLAB Implementation Code

```
%% Parameters
A = 1;
Eb = A^2;                                   % Energy per information bit (reference)
EbN0_db = -3 : 0.5 : 10;                    % Eb/N0 values in dB
EbN0 = 10.^(EbN0_db/10);                    % Linear values

num_bits = 110000;
Bits = randi([0,1], 1, num_bits);           % Generate random information bits

%% Reference: BPSK without coding (theoretical from Q1&2)
BER_uncoded_Theo = 0.5 * erfc(sqrt(EbN0));

%% BPSK with repetition-3 coding and soft decision decoding
rep_factor = 3;                             % Repetition factor for coding
coded_bits = repelem(Bits, rep_factor);  % Repeat each bit 3 times

% Initialize BER arrays
BER_soft_same_E_per_Tx_bit = zeros(size(EbN0_db));
BER_soft_same_E_per_Info_bit = zeros(size(EbN0_db));
```

```
%% (a) Same energy per transmitted bit (amplitude A, like uncoded)
Symbols_same_E_Tx = A * (2*coded_bits - 1);     % Same amplitude as uncoded

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Tx + Noise;

    % Soft decision decoding: sum the received voltages for each group of 3
    Rx_groups = reshape(Rx_Signal, rep_factor, []);
    soft_sum = sum(Rx_groups, 1);

    % Decision: positive sum -> 1, negative sum -> 0
    Rx_Bits_decoded = soft_sum > 0;

    % BER Calculation
    BER_soft_same_E_per_Tx_bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

%% (b) Same energy per information bit (total energy per info bit = Eb)
A_info = A / sqrt(rep_factor);        % Reduce amplitude to keep same total energy
Symbols_same_E_Info = A_info * (2*coded_bits - 1);

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Info + Noise;

    % Soft decision decoding: sum the received voltages
    Rx_groups = reshape(Rx_Signal, rep_factor, []);
    soft_sum = sum(Rx_groups, 1);

    % Decision
    Rx_Bits_decoded = soft_sum > 0;

    % BER Calculation
    BER_soft_same_E_per_Info_bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

% Plotting
semilogy(EbN0_db, BER_soft_same_E_per_Info_bit, 'b', 'LineWidth', 2); hold on;
semilogy(EbN0_db, BER_soft_same_E_per_Tx_bit, 'k', 'LineWidth', 2);
semilogy(EbN0_db, BER_uncoded_Theo, 'r--', 'LineWidth', 2); grid on;
ylim([1e-6 1e0]);
title('BPSK with Repetition-3 Coding (Soft Decision Decoding)');
xlabel('Eb/N0 (dB)'); ylabel('BER');
legend('Same Energy per Information Bit', ...
       'Same Energy per Transmitted Bit', 'Without Coding');
```

Output:

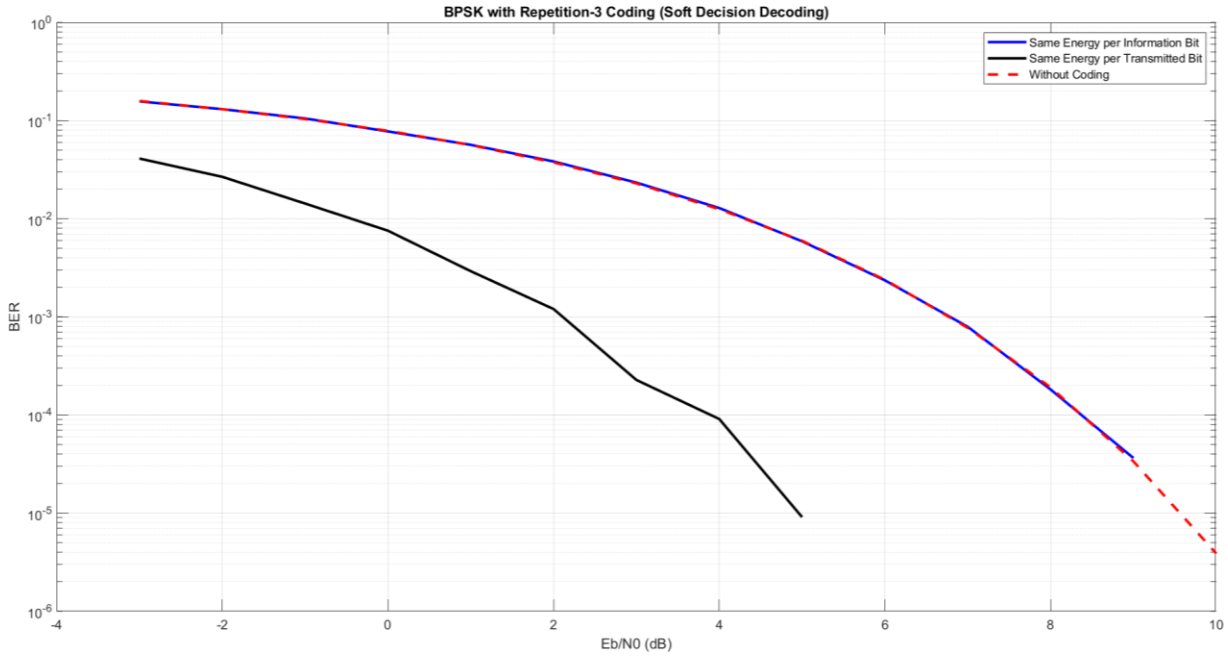

Figure 3: Soft Decision BPSK with 3-bit repetition, BER vs Eb/No

Comments:

For repetition 3 BPSK soft decision decoding we made 2_different scenarios of energy as shown:

1. First scenario is using Soft decision decoding with the same energy per transmitted bit:
   - We are using more energy than the system of no coding as the energy of each bit of the three bits equal to the energy of the energy of uncoded bit, so the total energy equal triple the energy of uncoded system (3 * A^2) so it will give us less BER.

2. Second scenario is using Soft decision decoding with the same energy per information bit:
   - In this case each bit has one third of the energy of uncoded bit so the total energy equal to the same energy of uncoded system (A^2) so it gives us the same BER of the uncoded system.

From 1,2 we can see that the best performance is using Soft decision decoding with the same energy per transmitted bit and using Soft decision decoding with the same energy per information bit gives the same performance of the uncoded system but with one third of the rate of the uncoded system so it is not recommended

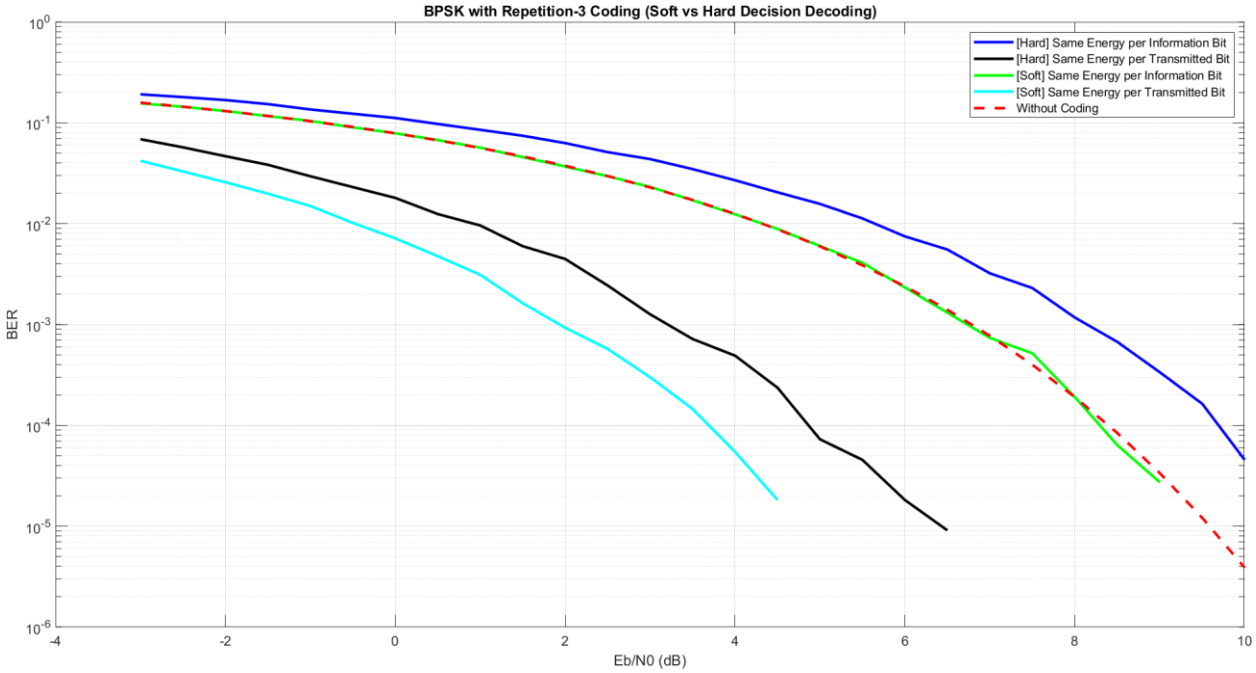

Figure 4: Hard vs Soft Decision BPSK with 3-bit repetition

Comments:

- As seen in Fig. 4 Soft Decision Decoding is always better BER performance than Hard Decision Decoding and that is because it preserves more information from the received signal while Hard Decision decoding makes a binary choice (0 or 1) for each bit based on a fixed threshold discarding all data about the signal's reliability
- Also the curves shown above in Fig.4 shows that decoding with the same energy per transmitted bit is better than decoding with the same energy per information bit which are the same results we discussed before, however the BER for soft and hard decision decoding with the same energy information bit is the same of BER of uncoded system or even worse in hard decision decoding case but with lose in rate which equals to one third of uncoded system so it is obviously not recommended

4. BPSK with a (7,4) Hamming Coding

MATLAB Implementation Code

```
clc; clear; close all;

%% Parameters
A = 1;
Eb = A^2;                                   % Energy per information bit
EbN0_db = -3 : 0.5 : 10;                      % Eb/N0 values in dB
EbN0 = 10.^(EbN0_db/10);                    % Linear values

num_bits = 110000;
Bits = randi([0,1], 1, num_bits);           % Generate random information bits

%% Reference: BPSK without coding (theoretical)
BER_uncoded_Theo = 0.5 * erfc(sqrt(EbN0));

%% BPSK with (7,4) Hamming code
n = 7;                                      % Codeword length
k = 4;                                      % Message length
code_rate = k / n;

% Encode the information bits
coded_bits = encode(Bits, n, k, 'hamming/binary');

% Initialize BER arrays
BER_hamming_same_E_per_Tx_bit = zeros(size(EbN0_db));
BER_hamming_same_E_per_Info_bit = zeros(size(EbN0_db));

%% (a) Same energy per transmitted bit
Symbols_same_E_Tx = A * (2 * coded_bits - 1);

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Tx + Noise;

    % Hard decision demapping
    Rx_Codes = Rx_Signal > 0;

    % Hamming decoding
    Rx_Bits_decoded = decode(Rx_Codes, n, k, 'hamming/binary');

    % BER Calculation (on information bits)
    BER_hamming_same_E_per_Tx_bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

%% (b) Same energy per information bit
A_info = A * sqrt(code_rate);               % Keep total energy per info bit = Eb
Symbols_same_E_Info = A_info * (2 * coded_bits - 1);
```

```
for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Info + Noise;

    % Hard decision demapping
    Rx_Codes = Rx_Signal > 0;

    % Hamming decoding
    Rx_Bits_decoded = decode(Rx_Codes, n, k, 'hamming/binary');

    % BER Calculation
    BER_hamming_same_E_per_Info_bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

% Plotting
semilogy(EbN0_db, BER_hamming_same_E_per_Info_bit, 'b', 'LineWidth', 2); hold on;
semilogy(EbN0_db, BER_hamming_same_E_per_Tx_bit, 'k', 'LineWidth', 2);
semilogy(EbN0_db, BER_uncoded_Theo, 'r--', 'LineWidth', 2); grid on;
ylim([1e-6 1e0]);
title('BPSK with (7,4) Hamming Code');
xlabel('Eb/N0 (dB)'); ylabel('BER');
legend('Same Energy per Information Bit', ...
       'Same Energy per Transmitted Bit', 'Without Coding');
```

Output:

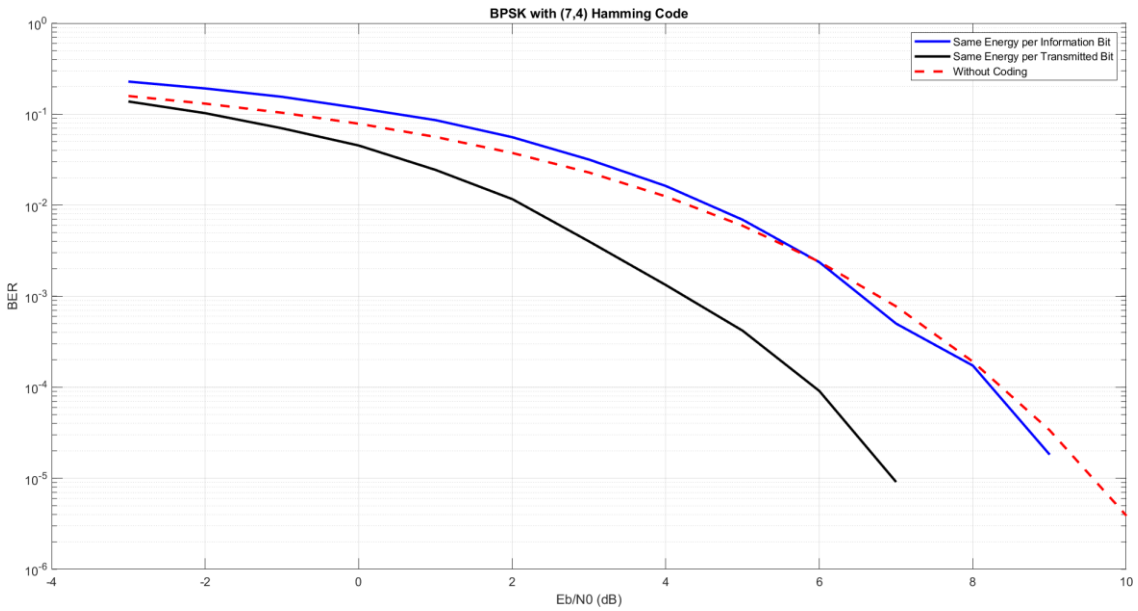

Figure 5: (7,4) Hamming Coding BPSK, BER vs Eb/No

Comments:

c) $d_{min} = 3$

d) it's recommended discarding the transmitting time as The BPSK repeated using hamming code (7,4) is better than the BPSK with the same Eb.

5. BPSK with a (15,11) Hamming Coding

MATLAB Implementation Code:

```
clc; clear; close all;
%% Parameters
A = 1;
Eb = A^2;                                   % Energy per information bit
EbN0_db = -3 : 0.5 : 10;                      % Eb/N0 values in dB
EbN0 = 10.^(EbN0_db/10);                     % Linear values

num_bits = 110000;
Bits = randi([0,1], 1, num_bits);            % Generate random information bits

%% Reference: BPSK without coding (theoretical)
BER_uncoded_Theo = 0.5 * erfc(sqrt(EbN0));

%% BPSK with (15,11) Hamming code
n = 15;                                     % Codeword length
k = 11;                                     % Message length
code_rate = k / n;

% Encode the information bits (done once, reused)
coded_bits = encode(Bits, n, k, 'hamming/binary');

% Initialize BER arrays
BER_hamming_same_E_per_Tx_bit = zeros(size(EbN0_db));
BER_hamming_same_E_per_Info_bit = zeros(size(EbN0_db));

%% Same energy per transmitted bit
Symbols_same_E_Tx = A * (2 * coded_bits - 1);

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Tx + Noise;

    % Hard decision demapping
    Rx_Codes = Rx_Signal > 0;

    % Hamming decoding
    Rx_Bits_decoded = decode(Rx_Codes, n, k, 'hamming/binary');
```

```
    % BER Calculation (on information bits)
    BER_hamming_same_E_per_Tx_bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

%% Same energy per information bit
A_Info = A * sqrt(code_rate);                % Total energy per info bit remains Eb
Symbols_same_E_Info = A_Info * (2 * coded_bits - 1);

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Info + Noise;

    % Hard decision demapping
    Rx_Codes = Rx_Signal > 0;

    % Hamming decoding
    Rx_Bits_decoded = decode(Rx_Codes, n, k, 'hamming/binary');

    % BER Calculation
    BER_hamming_same_E_per_Info_bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

%% Proposal for (f): keep transmission time ≤ uncoded case
% Re-use (7,4) from previous question (for comparison)
n74 = 7; k74 = 4; rate74 = k74/n74;
coded_74 = encode(Bits, n74, k74, 'hamming/binary');
A_74 = A * sqrt(rate74);
Symbols_74 = A_74 * (2 * coded_74 - 1);
BER_proposal = zeros(size(EbN0_db));

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));
    Noise = randn(1, length(coded_74)) .* sigma;
    Rx_Signal = Symbols_74 + Noise;
    Rx_Codes = Rx_Signal > 0;
    Rx_Bits_decoded = decode(Rx_Codes, n74, k74, 'hamming/binary');
    BER_proposal(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

% Plotting
semilogy(EbN0_db, BER_hamming_same_E_per_Info_bit, 'b', 'LineWidth', 2); hold on;
semilogy(EbN0_db, BER_hamming_same_E_per_Tx_bit, 'k', 'LineWidth', 2);
semilogy(EbN0_db, BER_uncoded_Theo, 'r--', 'LineWidth', 2);
semilogy(EbN0_db, BER_proposal, 'm-.', 'LineWidth', 2); grid on;
ylim([1e-6 1e0]);
title('BPSK with (15,11) Hamming Code');
xlabel('Eb/N0 (dB)'); ylabel('BER');
legend('Same Energy per Information Bit ((15,11))', ...
       'Same Energy per Transmitted Bit ((15,11))', ...
       'Without Coding (Theoretical)', 'Proposal: (7,4) Hamming (same E_b)');
```

Output:

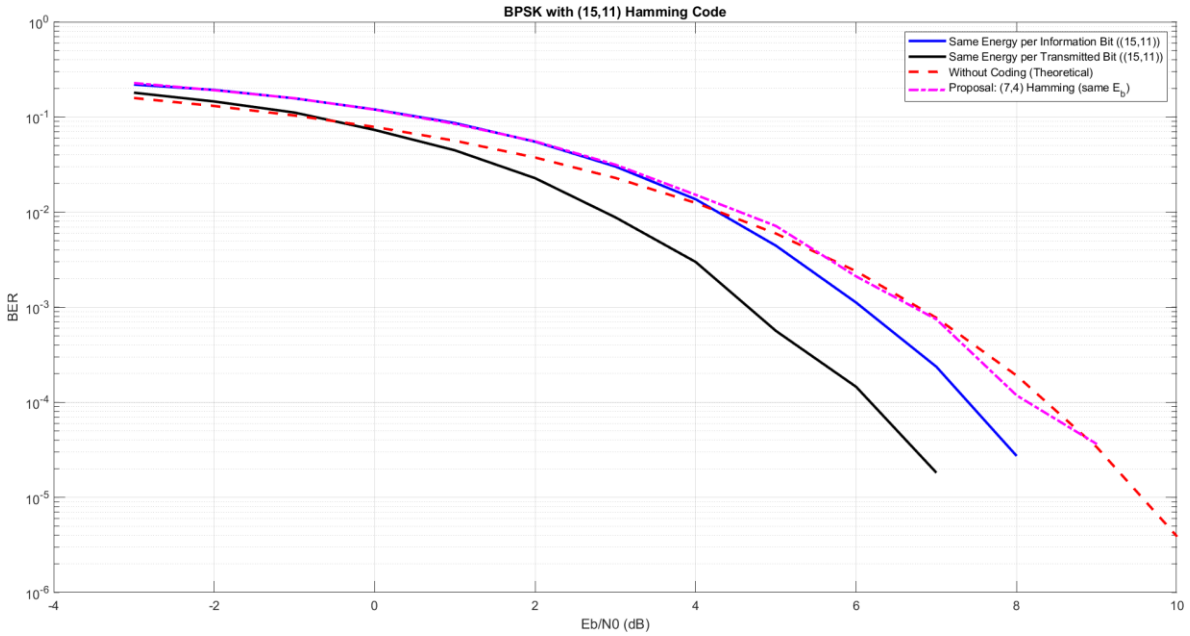

Figure 6: (15,11) Hamming Coding BPSK, BER vs Eb/No

Comments:

e) Recommended: Employ BPSK with a (15,11) Hamming code if the objective is to decrease the bit error rate while maintaining the same energy per bit, especially when disregarding the impact on transmission time.

(f) If the aim is to keep or decrease the transmission time relative to that of 1, given the intricacy linked with the (15,11) Hamming code, an option could involve employing BPSK with a (7,4) Hamming code. This approach would find a middle ground between error correction capabilities and transmission efficiency.

If want to keep the transmission time less than or equal the transmission time of no coding, use QPSK or QAM modulation.

g) QPSK modulation, despite having the same Bit Error Rate (BER) as BPSK techniques, offers twice the data rate. Introducing the Hamming (15,11) error correction scheme enhances performance in both rate and BER, albeit with a slight increase in energy consumption.

6. Comparison Between QPSK and 16-QAM with BCH Coding

In this section, the Bit Error Rate (BER) performance of two digital communication systems is evaluated over an Additive White Gaussian Noise (AWGN) channel. A total of 26.2×10^6 information bits are transmitted, and the BER is measured for Eb/No values ranging from 5 dB to 15 dB. The performance results of both systems are plotted on the same BER curve for comparison.

A. Uncoded QPSK System

Quadrature Phase Shift Keying (QPSK) modulation without channel coding is considered. A random binary sequence of 26.2×10^6 bits is generated and grouped into pairs, corresponding to two bits per QPSK symbol. Gray mapping is employed to reduce the probability of bit errors between adjacent constellation points.

MATLAB Implementation Code

```
fprintf('Starting QPSK simulation \n');
dataMat = reshape(dataBits,bitsPerQPSK,[]);
txQPSK  = sqrt(Eb) * ((2*dataMat(1,:) - 1) + 1j*(2*dataMat(2,:) - 1));

for i = 1:length(EbNo_dB)
    EbNo = EbNo_dB(i);

    % ---------------- Add AWGN ----------------
    SNR_linear = 10^(EbNo/10);
    noiseVar = Eb/(2*SNR_linear);
    noise = sqrt(noiseVar)*(randn(size(txQPSK)) + 1j*randn(size(txQPSK)));
    rx = txQPSK + noise;

    % ---------------- Demapping ----------------
    rxBits = reshape([real(rx)>0; imag(rx)>0],1,[]);
    BER_QPSK(i) = biterr(dataBits,rxBits)/Nbits;

    fprintf('QPSK Eb/No=%2d dB, BER=%.2e\n',EbNo,BER_QPSK(i));
end
```

The modulated signal is transmitted through an AWGN channel. At the receiver, coherent QPSK demodulation is performed, followed by bit detection. The BER is computed by comparing the detected bits with the transmitted bits for each Eb/No value.
Since no error-control coding is applied, the system performance depends solely on the modulation scheme and channel noise characteristics.

B. 16-QAM System with (255,131) BCH Coding

16-Quadrature Amplitude Modulation (16-QAM) combined with a **(255,131) BCH block code** is employed to improve error performance.

MATLAB Implementation Code

```
msgMat  = reshape(dataBits,kBCH,[])';
encoded = bchenc(gf(msgMat,1),nBCH,kBCH);  % BCH encoding
bitsEnc = reshape(double(encoded.x)',1,[]);
```

```
% Map to 16-QAM symbols using provided mod16 function
tx_16qam = mod16_func(tx_bits_bch);
    for it = 1:numIter
        % ----- Generate bits -----
        startIdx = (it-1)*26200 + 1;
        endIdx   = it*26200;
        bits = dataBits(startIdx:endIdx);
        % ----- BCH encode -----
        msgMat  = reshape(bits, kBCH, [])';
        encMat  = bchenc(gf(msgMat,1), nBCH, kBCH);
        bitsEnc = reshape(double(encMat.x)', 1, []);
        % ----- 16-QAM modulation -----
        txQAM16 = mod16_func(bitsEnc);
        % ----- AWGN channel -----
        Es = mean(abs(txQAM16).^2);
        SNR_linear = 10^(EbNo/10) * codeRate * bitsPerQAM16;
        noiseVar = Es / (2*SNR_linear);
        noise = sqrt(noiseVar) .* (randn(size(txQAM16)) + 1j*randn(size(txQAM16)));
        rx = txQAM16 + noise;
        % ----- Demodulation -----
        rxBits = demod16_func(rx);

        % ----- BCH decoding -----
        rxBlocks = reshape(rxBits, nBCH, [])';
        txBlocks = reshape(bitsEnc, nBCH, [])';

        decBlocks = bchdec(gf(rxBlocks,1), nBCH, kBCH);

        bitsDec = reshape(double(decBlocks.x)', 1, []);
        bitsRef = reshape(txBlocks(:,1:kBCH)', 1, []);

        % ----- Error count -----
        bitErrors = bitErrors + sum(bitsDec ~= bitsRef);
        totalBits = totalBits + length(bitsRef);
    end

    BER_QAM16_BCH(i) = bitErrors / totalBits;

    fprintf('16-QAM+BCH Eb/No=%2d dB, BER=%.3e\n', ...
            EbNo, BER_QAM16_BCH(i));
end
```

The information bits are first encoded using a bchenc, implemented using MATLAB's encode function. Each block of 131 information bits is encoded into a 255-bit codeword, introducing redundancy that enables error detection and correction at the receiver.

The encoded bits are then modulated using 16-QAM, where each symbol represents four bits. Custom modulation and demodulation functions are used for symbol mapping and detection. The modulated signal is transmitted over an AWGN channel.

At the receiver, the signal is demodulated and subsequently decoded using MATLAB's decode function. The decoded bits are compared with the original information bits to calculate the BER for each Eb/No value.

C. Results and Discussion

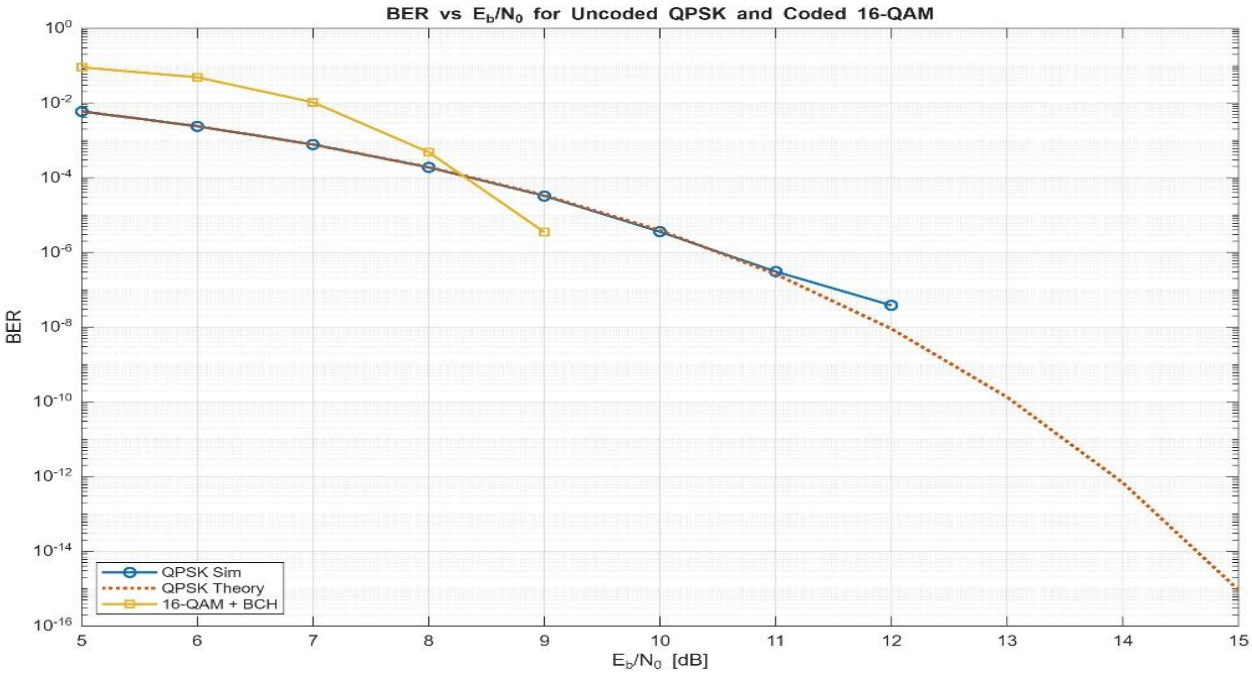

Figure 7: BER performance of uncoded QPSK and BCH-coded 16-QAM over an AWGN channel, compared with the theoretical QPSK BER

performance of uncoded QPSK and BCH-coded 16-QAM over an AWGN channel from 5 dB to 15 dB. The uncoded QPSK simulation closely matches the theoretical BER confirming correct modulation and noise scaling.

Theoretical BER

$$\text{BER} = 0.5 * \text{erfc}\left(\frac{\text{Eb}}{\text{N0}}\right)$$

The BCH-coded 16-QAM exhibits lower BER at moderate and high $\frac{\text{Eb}}{\text{N0}}$, benefiting from the (255,131) BCH code which corrects multiple bit errors per codeword. While 16-QAM is more sensitive to noise than QPSK, coding gain shifts its BER curve to the left, demonstrating the advantage of combining higher-order modulation with channel coding.

Overall, uncoded QPSK offers simplicity and robustness, whereas coded 16-QAM achieves superior error performance and better spectral efficiency.

7. Comparison Between QPSK and 16-QAM with BCH Coding

➢ We have 6 generator polynomials: **Output Bit Equations**

$g_1^{(1)} = [0\ 1] \rightarrow contributes\ U_1(n-1)$
$g_2^{(1)} = [1\ 1] \rightarrow contributes\ U_2(n) + U_2(n-1)$
$\boldsymbol{V(1) = U_1(n-1) \oplus U_2(n) \oplus U_2(n-1)}$

$g_1^{(2)} = [1\ 1] \rightarrow contributes\ U_1(n) + U_1(n-1)$
$g_2^{(2)} = [1\ 0] \rightarrow contributes\ U_2(n)$
$\boldsymbol{V(2) = U_1(n) \oplus U_1(n-1) \oplus U_2(n)}$

$g_1^{(3)} = [0\ 0] \rightarrow no\ contributes\ U_1$
$g_2^{(3)} = [1\ 1] \rightarrow contributes\ U_2(n) + U_2(n-1)$
$\boldsymbol{V(3) = U_2(n) \oplus U_2(n-1)}$

- $g_1^{(i)}$ defines how **input 1** contributes to **output i**.
- $g_2^{(i)}$ defines how **input 2** contributes to **output i**.

✓ Then the encoder has **2 inputs** and **3 outputs**.

- Each generator polynomial has 2 bits.
- Constraint length $2 = m + 1 \rightarrow m = 2 - 1 = 1$

✓ Then Each input uses **1 memory element**

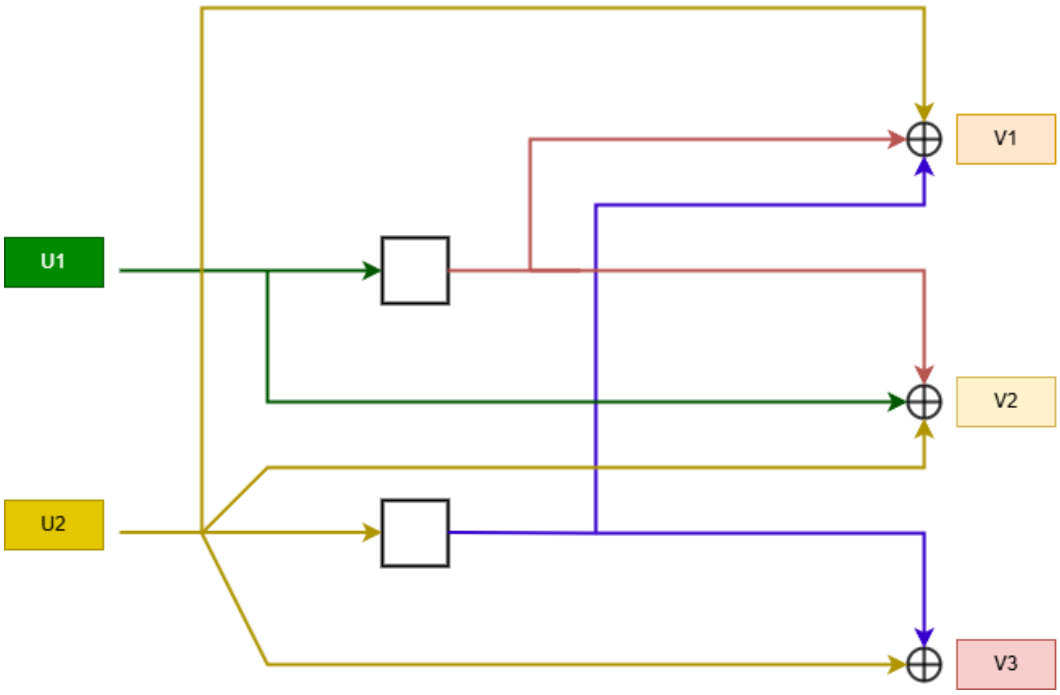

Figure 8: Block Diagram

❖ Snippets of the output of code

```
Full table of Input U(i) [2 bits] and Encoded Output V(i) [3 bits]
```

U1	U2	|	V1	V2	V3
1	1	|	1	0	1
0	1	|	1	0	0
0	0	|	1	0	1
1	1	|	1	0	1
1	1	|	1	1	0
0	0	|	0	1	1
0	1	|	1	1	1
0	1	|	0	1	0
1	0	|	1	1	1
0	0	|	1	1	0

➢ Proof

$$\boldsymbol{V(1) = U_1(n-1) \oplus U_2(n) \oplus U_2(n-1) = M1 \oplus U2 \oplus M2}$$

$$\boldsymbol{V(2) = U_1(n) \oplus U_1(n-1) \oplus U_2(n) = U1 \oplus M1 \oplus U2}$$

$$\boldsymbol{V(3) = U_2(n) \oplus U_2(n-1) = U2 \oplus M2}$$

U1	U2	M1	M2	V1	V2	V3
1	1	0	0	0⊕1⊕0= 1	1⊕0⊕1= 0	1⊕0= 1
0	1	1	1	1⊕1⊕1= 1	0⊕1⊕1= 0	1⊕1= 0
0	0	0	1	0⊕0⊕1= 1	0⊕0⊕0= 0	0⊕1= 1
1	1	0	0	0⊕1⊕0= 1	1⊕0⊕1= 0	1⊕0= 1
1	1	1	1	1⊕1⊕1= 1	1⊕1⊕1= 1	1⊕1= 0
0	0	1	1	1⊕0⊕1= 0	0⊕1⊕0= 1	0⊕1= 1
0	1	0	0	0⊕1⊕0= 1	0⊕0⊕1= 1	1⊕0= 1

MATLAB Code

```
clc; clear; close;

%% Parameters
N = 1000;  % number of bits


%% Generate random bits
U = randi([0 1], 1, N);

%% Split into two inputs
U1 = U(1:2:end);  % Input 1
U2 = U(2:2:end);  % Input 2

L = length(U1);  % number of input pairs
% or L = length(U2);

%% Initialize memory (shift registers)
% we have 1 memory in each path K=1
% we initially assumed the memory = 0
U1_mem = 0;
U2_mem = 0;

%% Initialize output
V = zeros(L, 3);  % Output columns: V1, V2, V3

%% Manual XOR function
xor2 = @(a,b) (a + b == 1);  % returns 1 if exactly one of a or b is 1

%% Convolutional encoding
for n = 1:L

     % V1 = U1(n-1) XOR U2(n) XOR U2(n-1)
    temp = xor2(U1_mem, U2(n));
    V(n,1) = xor2(temp, U2_mem);

    % V2 = u1(n) XOR U1(n-1) XOR u2(n)
    temp = xor2(U1(n), U1_mem);
    V(n,2) = xor2(temp, U2(n));

    % V3 = u2(n) XOR U2(n-1)
    V(n,3) = xor2(U2(n), U2_mem);


    % Update the memory
    U1_mem = U1(n);   % U1(n-1)
    U2_mem = U2(n);   % U2(n-1)
end

%% Prepare input/output table
input_bits  = [U1' U2']; % 2-bit input  columns
output_bits = V;         % 3-bit output columns
```

```
%% Combine into a table with clear labels
Table = table(input_bits(:,1), input_bits(:,2), repmat('|',L,1), output_bits(:,1),
output_bits(:,2), output_bits(:,3), ...
    'VariableNames', {'U1','U2','|','V1','V2','V3'});

%% Display the table
disp("Full table of Input U(i) [2 bits] and Encoded Output V(i) [3 bits]");
disp(Table);
%disp(Table(1:500,:));
```

Appendix

```
% This MATLAB script runs a single execution to generate all required plots
% and outputs for Questions 3 through 7 and Question 9 of the report.
% Question 8 (QPSK vs. 16-QAM with BCH code over 26,200,000 bits) is excluded
% from this script because it requires extremely long simulation time and
% would significantly delay the execution of the other questions.
% It should be simulated separately.

clc; clear; close all;

% Parameters
A = 1;
Eb = A^2;
EbN0_db = -3 : 0.5 : 10;  % SNR list in dB
EbN0 = 10.^(EbN0_db/10);  % Linear values

num_bits = 110000;
Bits = randi([0,1], 1, num_bits);  % Generate random bits to be transmitted

%% BPSK
Symbols = A * (2*Bits - 1);        % Mapping: 0 -> -1 & 1 -> +1

% Initialize variables
BER_BPSK = zeros(size(EbN0_db));
BER_BPSK_Theo = zeros(size(EbN0_db));  % Theoretical value

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));    % σ = √N0/2

    N0 = 2 * (sigma.^2);

    % Channel
    Noise = randn(1, num_bits) .* sigma;  % Generate random noise
    Rx_Signal = Symbols + Noise;          % Noisy received signal

    % Demapper, Decision threshold = 0
    Rx_Bits = Rx_Signal > 0;  % Received bit= 0 if signal <= 0 & 1 if signal > 0

    % BER Calculation
    ErrorBits = sum(Rx_Bits ~= Bits);
    BER_BPSK(i) = ErrorBits / num_bits;

    BER_BPSK_Theo(i) = 0.5 * erfc(sqrt(Eb/N0));
end

% Plotting
figure
semilogy(EbN0_db, BER_BPSK, 'b', 'LineWidth', 2); hold on ;
semilogy(EbN0_db, BER_BPSK_Theo, 'r --', 'LineWidth', 2); grid on;
ylim([1e-6 1e0]);
title('BPSK Without Coding');
xlabel('Eb/N0 (dB)'); ylabel('BER');
legend('Simulated BER', 'Theoretical BER');
```

```
%% BPSK without coding (reference curve)
BER_uncoded = BER_BPSK;
BER_uncoded_Theo = BER_BPSK_Theo;

%% BPSK with repetition-3 coding and hard decision decoding
rep_factor = 3;                          % Repetition factor for coding
coded_bits = repelem(Bits, rep_factor);  % Repeat each bit 3 times

% Initialize BER arrays
BER_same_E_per_Tx_Bit = zeros(size(EbN0_db));
BER_same_E_per_Info_Bit = zeros(size(EbN0_db));

%% (a) Same energy per transmitted bit (amplitude A, like uncoded)
Symbols_same_E_Tx = A * (2*coded_bits - 1);     % Same amplitude as uncoded

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Tx + Noise;

    % Hard decision on each repeated symbol
    Rx_Codes = Rx_Signal > 0;

    % Majority decoding (at least 2 out of 3)
    Rx_Codes_reshaped = reshape(Rx_Codes, rep_factor, []);
    Rx_Bits_decoded = sum(Rx_Codes_reshaped, 1) >= 2;

    % BER Calculation
    BER_same_E_per_Tx_Bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

%% (b) Same energy per information bit (total energy per info bit = Eb)
A_info = A / sqrt(rep_factor);        % Reduce amplitude to keep same total energy
Symbols_same_E_Info = A_info * (2*coded_bits - 1);

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Info + Noise;

    % Hard decision on each repeated symbol
    Rx_Codes = Rx_Signal > 0;

    % Majority decoding (at least 2 out of 3)
    Rx_Codes_reshaped = reshape(Rx_Codes, rep_factor, []);
    Rx_Bits_decoded = sum(Rx_Codes_reshaped, 1) >= 2;

    % BER Calculation
    BER_same_E_per_Info_Bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end
```

```
% Plotting
figure;
semilogy(EbN0_db, BER_same_E_per_Info_Bit, 'b', 'LineWidth', 2); hold on;
semilogy(EbN0_db, BER_same_E_per_Tx_Bit, 'k', 'LineWidth', 2);
semilogy(EbN0_db, BER_uncoded, 'r--', 'LineWidth', 2); grid on;
ylim([1e-6 1e0]);
title('BPSK with Repetition-3 Coding (Hard Decision Decoding)');
xlabel('Eb/N0 (dB)'); ylabel('BER');
legend('Same Energy per Information Bit', ...
       'Same Energy per Transmitted Bit', 'Without Coding');

%% BPSK with repetition-3 coding and soft decision decoding
% Initialize BER arrays
BER_soft_same_E_per_Tx_bit = zeros(size(EbN0_db));
BER_soft_same_E_per_Info_bit = zeros(size(EbN0_db));

%% (a) Same energy per transmitted bit (amplitude A, like uncoded)
Symbols_same_E_Tx = A * (2*coded_bits - 1);     % Same amplitude as uncoded

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Tx + Noise;

    % Soft decision decoding: sum the received voltages for each group of 3
    Rx_groups = reshape(Rx_Signal, rep_factor, []);
    soft_sum = sum(Rx_groups, 1);

    % Decision: positive sum -> 1, negative sum -> 0
    Rx_Bits_decoded = soft_sum > 0;

    % BER Calculation
    BER_soft_same_E_per_Tx_bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

%% (b) Same energy per information bit (total energy per info bit = Eb)
A_info = A / sqrt(rep_factor);        % Reduce amplitude to keep same total energy
Symbols_same_E_Info = A_info * (2*coded_bits - 1);

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Info + Noise;

    % Soft decision decoding: sum the received voltages
    Rx_groups = reshape(Rx_Signal, rep_factor, []);
    soft_sum = sum(Rx_groups, 1);

    % Decision
    Rx_Bits_decoded = soft_sum > 0;
```

```
    % BER Calculation
    BER_soft_same_E_per_Info_bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

% Plotting
figure;
semilogy(EbN0_db, BER_soft_same_E_per_Info_bit, 'b', 'LineWidth', 2); hold on;
semilogy(EbN0_db, BER_soft_same_E_per_Tx_bit, 'k', 'LineWidth', 2);
semilogy(EbN0_db, BER_uncoded_Theo, 'r--', 'LineWidth', 2); grid on;
ylim([1e-6 1e0]);
title('BPSK with Repetition-3 Coding (Soft Decision Decoding)');
xlabel('Eb/N0 (dB)'); ylabel('BER');
legend('Same Energy per Information Bit', ...
       'Same Energy per Transmitted Bit', 'Without Coding');

%% Plot Hard and Soft together
figure;
semilogy(EbN0_db, BER_same_E_per_Info_Bit, 'b', 'LineWidth', 2); hold on;
semilogy(EbN0_db, BER_same_E_per_Tx_Bit, 'k', 'LineWidth', 2);
semilogy(EbN0_db, BER_soft_same_E_per_Info_bit, 'g', 'LineWidth', 2);
semilogy(EbN0_db, BER_soft_same_E_per_Tx_bit, 'c', 'LineWidth', 2);
semilogy(EbN0_db, BER_uncoded_Theo, 'r--', 'LineWidth', 2); grid on;
ylim([1e-6 1e0]);
title('BPSK with Repetition-3 Coding (Soft vs Hard Decision Decoding)');
xlabel('Eb/N0 (dB)'); ylabel('BER');
legend('[Hard] Same Energy per Information Bit', '[Hard] Same Energy per
Transmitted Bit', ...
       '[Soft] Same Energy per Information Bit', '[Soft] Same Energy per
Transmitted Bit', ...
       'Without Coding');

%% BPSK with (7,4) Hamming code
n = 7;                                      % Codeword length
k = 4;                                      % Message length
code_rate = k / n;

% Encode the information bits
coded_bits = encode(Bits, n, k, 'hamming/binary');

% Initialize BER arrays
BER_hamming_same_E_per_Tx_bit = zeros(size(EbN0_db));
BER_hamming_same_E_per_Info_bit = zeros(size(EbN0_db));

%% (a) Same energy per transmitted bit
Symbols_same_E_Tx = A * (2 * coded_bits - 1);

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Tx + Noise;

    % Hard decision demapping
```

```
    Rx_Codes = Rx_Signal > 0;

    % Hamming decoding
    Rx_Bits_decoded = decode(Rx_Codes, n, k, 'hamming/binary');

    % BER Calculation (on information bits)
    BER_hamming_same_E_per_Tx_bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

%% (b) Same energy per information bit
A_info = A * sqrt(code_rate);                % Keep total energy per info bit = Eb
Symbols_same_E_Info = A_info * (2 * coded_bits - 1);

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Info + Noise;

    % Hard decision demapping
    Rx_Codes = Rx_Signal > 0;

    % Hamming decoding
    Rx_Bits_decoded = decode(Rx_Codes, n, k, 'hamming/binary');

    % BER Calculation
    BER_hamming_same_E_per_Info_bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

% Plotting
figure
semilogy(EbN0_db, BER_hamming_same_E_per_Info_bit, 'b', 'LineWidth', 2); hold on;
semilogy(EbN0_db, BER_hamming_same_E_per_Tx_bit, 'k', 'LineWidth', 2);
semilogy(EbN0_db, BER_uncoded_Theo, 'r--', 'LineWidth', 2); grid on;
ylim([1e-6 1e0]);
title('BPSK with (7,4) Hamming Code');
xlabel('Eb/N0 (dB)'); ylabel('BER');
legend('Same Energy per Information Bit', ...
       'Same Energy per Transmitted Bit', 'Without Coding');

%% BPSK with (15,11) Hamming code
n = 15;                                      % Codeword length
k = 11;                                      % Message length
code_rate = k / n;

% Encode the information bits (done once, reused)
coded_bits = encode(Bits, n, k, 'hamming/binary');

% Initialize BER arrays
BER_hamming_same_E_per_Tx_bit = zeros(size(EbN0_db));
BER_hamming_same_E_per_Info_bit = zeros(size(EbN0_db));

%% Same energy per transmitted bit
Symbols_same_E_Tx = A * (2 * coded_bits - 1);
```

```
for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Tx + Noise;

    % Hard decision demapping
    Rx_Codes = Rx_Signal > 0;

    % Hamming decoding
    Rx_Bits_decoded = decode(Rx_Codes, n, k, 'hamming/binary');

    % BER Calculation (on information bits)
    BER_hamming_same_E_per_Tx_bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

%% Same energy per information bit
A_Info = A * sqrt(code_rate);                % Total energy per info bit remains Eb
Symbols_same_E_Info = A_Info * (2 * coded_bits - 1);

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));

    % Channel
    Noise = randn(1, length(coded_bits)) .* sigma;
    Rx_Signal = Symbols_same_E_Info + Noise;

    % Hard decision demapping
    Rx_Codes = Rx_Signal > 0;

    % Hamming decoding
    Rx_Bits_decoded = decode(Rx_Codes, n, k, 'hamming/binary');

    % BER Calculation
    BER_hamming_same_E_per_Info_bit(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end

%% Proposal for (f): keep transmission time ≤ uncoded case
% Re-use (7,4) from previous question (for comparison)
n74 = 7; k74 = 4; rate74 = k74/n74;
coded_74 = encode(Bits, n74, k74, 'hamming/binary');
A_74 = A * sqrt(rate74);
Symbols_74 = A_74 * (2 * coded_74 - 1);
BER_proposal = zeros(size(EbN0_db));

for i = 1:length(EbN0_db)
    sigma = sqrt((Eb/2) / EbN0(i));
    Noise = randn(1, length(coded_74)) .* sigma;
    Rx_Signal = Symbols_74 + Noise;
    Rx_Codes = Rx_Signal > 0;
    Rx_Bits_decoded = decode(Rx_Codes, n74, k74, 'hamming/binary');
    BER_proposal(i) = sum(Rx_Bits_decoded ~= Bits) / num_bits;
end
```

```
% Plotting
figure
semilogy(EbN0_db, BER_hamming_same_E_per_Info_bit, 'b', 'LineWidth', 2); hold on;
semilogy(EbN0_db, BER_hamming_same_E_per_Tx_bit, 'k', 'LineWidth', 2);
semilogy(EbN0_db, BER_uncoded_Theo, 'r--', 'LineWidth', 2);
semilogy(EbN0_db, BER_proposal, 'm-.', 'LineWidth', 2); grid on;
ylim([1e-6 1e0]);
title('BPSK with (15,11) Hamming Code');
xlabel('Eb/N0 (dB)'); ylabel('BER');
legend('Same Energy per Information Bit ((15,11))', ...
       'Same Energy per Transmitted Bit ((15,11))', ...
       'Without Coding (Theoretical)', 'Proposal: (7,4) Hamming (same E_b)');

%% Question 9
% Parameters
clear; clc;

N = 1000;  % number of bits

% Generate random bits
U = randi([0 1], 1, N);

% Split into two inputs
U1 = U(1:2:end);  % Input 1
U2 = U(2:2:end);  % Input 2

L = length(U1);  % number of input pairs
% or L = length(U2);

% Initialize memory (shift registers)
% we have 1 memory in each path K=1
% we initially assumed the memory = 0
U1_mem = 0;
U2_mem = 0;

% Initialize output
V = zeros(L, 3);  % Output columns: V1, V2, V3

% Manual XOR function
xor2 = @(a,b) (a + b == 1);  % returns 1 if exactly one of a or b is 1

%% Convolutional encoding
for n = 1:L

     % V1 = U1(n-1) XOR U2(n) XOR U2(n-1)
    temp = xor2(U1_mem, U2(n));
    V(n,1) = xor2(temp, U2_mem);

    % V2 = u1(n) XOR U1(n-1) XOR u2(n)
    temp = xor2(U1(n), U1_mem);
    V(n,2) = xor2(temp, U2(n));

    % V3 = u2(n) XOR U2(n-1)
    V(n,3) = xor2(U2(n), U2_mem);
```

```
    % Update the memory
    U1_mem = U1(n);   % U1(n-1)
    U2_mem = U2(n);   % U2(n-1)
end

% Prepare input/output table
input_bits  = [U1' U2']; % 2-bit input  columns
output_bits = V;         % 3-bit output columns

% Combine into a table with clear labels
Table = table(input_bits(:,1), input_bits(:,2), repmat('|',L,1), output_bits(:,1),
output_bits(:,2), output_bits(:,3), ...
    'VariableNames', {'U1','U2','|','V1','V2','V3'});

% Display the table
disp("Full table of Input U(i) [2 bits] and Encoded Output V(i) [3 bits]");
disp(Table);
%disp(Table(1:500,:));

%% Question 9
% Initialization
clear; rng(0);

Nbits      = 26200000;       % Total number of info bits
EbNo_dB    = 5:15;            % Eb/No range in dB

nBCH       = 255;             % BCH codeword length
kBCH       = 131;             % BCH message length
codeRate   = kBCH/nBCH;       % BCH code rate

bitsPerQPSK  = 2;             % Bits per QPSK symbol
bitsPerQAM16 = 4;             % Bits per 16-QAM symbol
Eb           = 2.5;           % Energy per bit

BER_QPSK      = zeros(size(EbNo_dB));
BER_QAM16_BCH = zeros(size(EbNo_dB));

% Data Generation
fprintf('Generating random bits \n');
dataBits = randi([0 1],1,Nbits);  % Random info bits

% QPSK (Uncoded)
fprintf('Starting QPSK simulation \n');
dataMat = reshape(dataBits,bitsPerQPSK,[]);
txQPSK  = sqrt(Eb) * ((2*dataMat(1,:) - 1) + 1j*(2*dataMat(2,:) - 1));

for i = 1:length(EbNo_dB)
    EbNo = EbNo_dB(i);

    % ---------------- Add AWGN ----------------
    SNR_linear = 10^(EbNo/10);
    noiseVar = Eb/(2*SNR_linear);
    noise = sqrt(noiseVar)*(randn(size(txQPSK)) + 1j*randn(size(txQPSK)));
```

```
    rx = txQPSK + noise;

    % ---------------- Demapping ----------------
    rxBits = reshape([real(rx)>0; imag(rx)>0],1,[]);
    BER_QPSK(i) = biterr(dataBits,rxBits)/Nbits;

    fprintf('QPSK Eb/No=%2d dB, BER=%.2e\n',EbNo,BER_QPSK(i));
end

%% ================= 16-QAM + BCH =================
fprintf('Start BCH encoding \n');
msgMat  = reshape(dataBits,kBCH,[])';
encoded = bchenc(gf(msgMat,1),nBCH,kBCH);  % BCH encoding
bitsEnc = reshape(double(encoded.x)',1,[]);

fprintf('Start 16-QAM mapping \n');
txQAM16 = mod16_func(bitsEnc);              % Map bits to 16-QAM

for i = 1:length(EbNo_dB)
    EbNo = EbNo_dB(i);

    % --------- Control simulation length ---------
    if EbNo <= 8
        Nbits_sim = 2620000;          % fast simulation
    elseif EbNo == 9 || EbNo == 15
        Nbits_sim = 26200000;         % full simulation
    %else
        %BER_QAM16_BCH(i) = NaN;       % interpolate later
       % continue;
    end

    % --------- Monte Carlo accumulation ---------
    numIter   = Nbits_sim / 26200;
    bitErrors = 0;
    totalBits = 0;

    for it = 1:numIter

        % ----- Generate bits -----
        startIdx = (it-1)*26200 + 1;
        endIdx   = it*26200;

        bits = dataBits(startIdx:endIdx);


        % ----- BCH encode -----
        msgMat  = reshape(bits, kBCH, [])';
        encMat  = bchenc(gf(msgMat,1), nBCH, kBCH);
        bitsEnc = reshape(double(encMat.x)', 1, []);


        % ----- 16-QAM modulation -----
        txQAM16 = mod16_func(bitsEnc);

        % ----- AWGN channel -----
```

```
        Es = mean(abs(txQAM16).^2);
        SNR_linear = 10^(EbNo/10) * codeRate * bitsPerQAM16;
        noiseVar = Es / (2*SNR_linear);

        noise = sqrt(noiseVar) .* ...
                (randn(size(txQAM16)) + 1j*randn(size(txQAM16)));

        rx = txQAM16 + noise;

        % ----- Demodulation -----
        rxBits = demod16_func(rx);

        % ----- BCH decoding -----
        rxBlocks = reshape(rxBits, nBCH, [])';
        txBlocks = reshape(bitsEnc, nBCH, [])';

        decBlocks = bchdec(gf(rxBlocks,1), nBCH, kBCH);

        bitsDec = reshape(double(decBlocks.x)', 1, []);
        bitsRef = reshape(txBlocks(:,1:kBCH)', 1, []);

        % ----- Error count -----
        bitErrors = bitErrors + sum(bitsDec ~= bitsRef);
        totalBits = totalBits + length(bitsRef);
    end

    BER_QAM16_BCH(i) = bitErrors / totalBits;

    fprintf('16-QAM+BCH Eb/No=%2d dB, BER=%.3e\n', ...
            EbNo, BER_QAM16_BCH(i));
end



% Theoretical QPSK
EbNo_lin = 10.^(EbNo_dB/10);
BER_QPSK_th = 0.5*erfc(sqrt(EbNo_lin));

% Plotting
figure;
semilogy(EbNo_dB,BER_QPSK,'-o','LineWidth',1.5); hold on;
semilogy(EbNo_dB,BER_QPSK_th,':','LineWidth',2);
semilogy(EbNo_dB,BER_QAM16_BCH,'-s','LineWidth',1.5);
grid on;
xlabel('E_b/N_0 [dB]');
ylabel('BER');
legend('QPSK Sim','QPSK Theory','16-QAM + BCH','Location','southwest');
title('BER vs E_b/N_0 for Uncoded QPSK and Coded 16-QAM');

%% ================= Functions =================
function txSig = mod16_func(bits)
    map = [1+1j 3+1j 1+3j 3+3j 1-1j 3-1j 1-3j 3-3j ...
           -1+1j -3+1j -1+3j -3+3j -1-1j -3-1j -1-3j -3-3j];
    bits = reshape(bits,4,[])';
    idx  = bi2de(bits,'left-msb')+1;
```

```
    txSig = map(idx).';
end

function bits = demod16_func(rxSig)
    demap = [15 14 6 7 13 12 4 5 9 8 0 1 11 10 2 3];
    rxSig(real(rxSig)>3)=3+1j*imag(rxSig(real(rxSig)>3));
    rxSig(imag(rxSig)>3)=real(rxSig(imag(rxSig)>3))+1j*3;
    rxSig(real(rxSig)<-3)=-3+1j*imag(rxSig(real(rxSig)<-3));
    rxSig(imag(rxSig)<-3)=real(rxSig(imag(rxSig)<-3))-1j*3;

    idx = round((real(rxSig)+3)/2) + 4*round((imag(rxSig)+3)/2);
    bits = de2bi(demap(idx+1),4,'left-msb');
    bits = reshape(bits',1,[]);
end
```