

**Electronics and Electrical Communications  
Engineering Department  
Faculty of Engineering Cairo University**

# **Information Theory Project Part II Channel Coding**

ELC4020 Advanced Communication Systems  
4<sup>th</sup> Year 1<sup>st</sup> Semester  
Academic Year 2025/2026

**Prepared by:**

Name	Section	ID
Mohamed Ahmed Abd El Hakam	3	9220647
Yousef Khaled Omar Mahmoud	4	9220984
Ahmed Wagdy Mohy Ibrahim	1	9220120
Abdelrahman Essa Elsayed	2	9220469

**Instructor:** Eng. Mohamed Khaled  
**Supervised by:** Dr. Mohamed Nafie, Dr. Mohamed Khairy

# Contents

<b>Executive Summary</b>	<b>1</b>
<b>1 Introduction</b>	<b>2</b>
<b>2 Simulation Environment</b>	<b>3</b>
<b>3 Problem 3: Uncoded BPSK over AWGN</b>	<b>5</b>
<b>4 Problem 4: Repetition-3 Coded BPSK (Hard Decision)</b>	<b>11</b>
<b>5 Problem 5: Repetition-3 Coded BPSK (Soft Decision)</b>	<b>16</b>
<b>6 Problem 6: Hamming (7,4) Coded BPSK</b>	<b>21</b>
<b>7 Problem 7: Hamming (15,11) with BPSK and QPSK</b>	<b>26</b>
<b>8 Problem 8: BCH-Coded 16-QAM vs Uncoded QPSK</b>	<b>31</b>
<b>9 Problem 9: Convolutional Encoding</b>	<b>37</b>
<b>10 Conclusion</b>	<b>41</b>
<b>A GUI Implementation</b>	<b>42</b>

# List of Figures

2.1	Graphical User Interface used to execute channel coding experiments. . . . .	4
3.1	BER performance of uncoded BPSK over an AWGN channel. . . . .	5
4.1	BER comparison of uncoded BPSK and repetition-3 coded BPSK using hard decision decoding. . . . .	11
5.1	BER performance of repetition-3 coded BPSK using soft decision decoding.	16
6.1	BER comparison between uncoded BPSK and Hamming (7,4) coded BPSK.	21
7.1	BER comparison of Hamming (15,11) coded BPSK and QPSK. . . . .	26
8.1	BER comparison between uncoded QPSK and BCH(255,131) coded 16-QAM. . . . .	31
9.1	Convolutional encoder state transition and output table. . . . .	37

# **Executive Summary**

This report investigates the effect of channel coding techniques on the bit error rate (BER) performance of digital communication systems operating over an additive white Gaussian noise (AWGN) channel. Different modulation schemes and coding strategies are evaluated to highlight the trade-offs between reliability, energy efficiency, and spectral efficiency.

# **Chapter 1**

## **Introduction**

Channel coding is used to introduce controlled redundancy in transmitted data to combat the effect of noise and channel impairments. In this project, several channel coding techniques are analyzed and compared using BER as the main performance metric. All simulations are implemented using MATLAB.

# Chapter 2

## Simulation Environment

All experiments are implemented using MATLAB and integrated into a unified graphical user interface (GUI). The GUI allows the user to select any problem from Question 3 to Question 9, execute the corresponding simulation, and automatically display numerical outputs and generated figures.

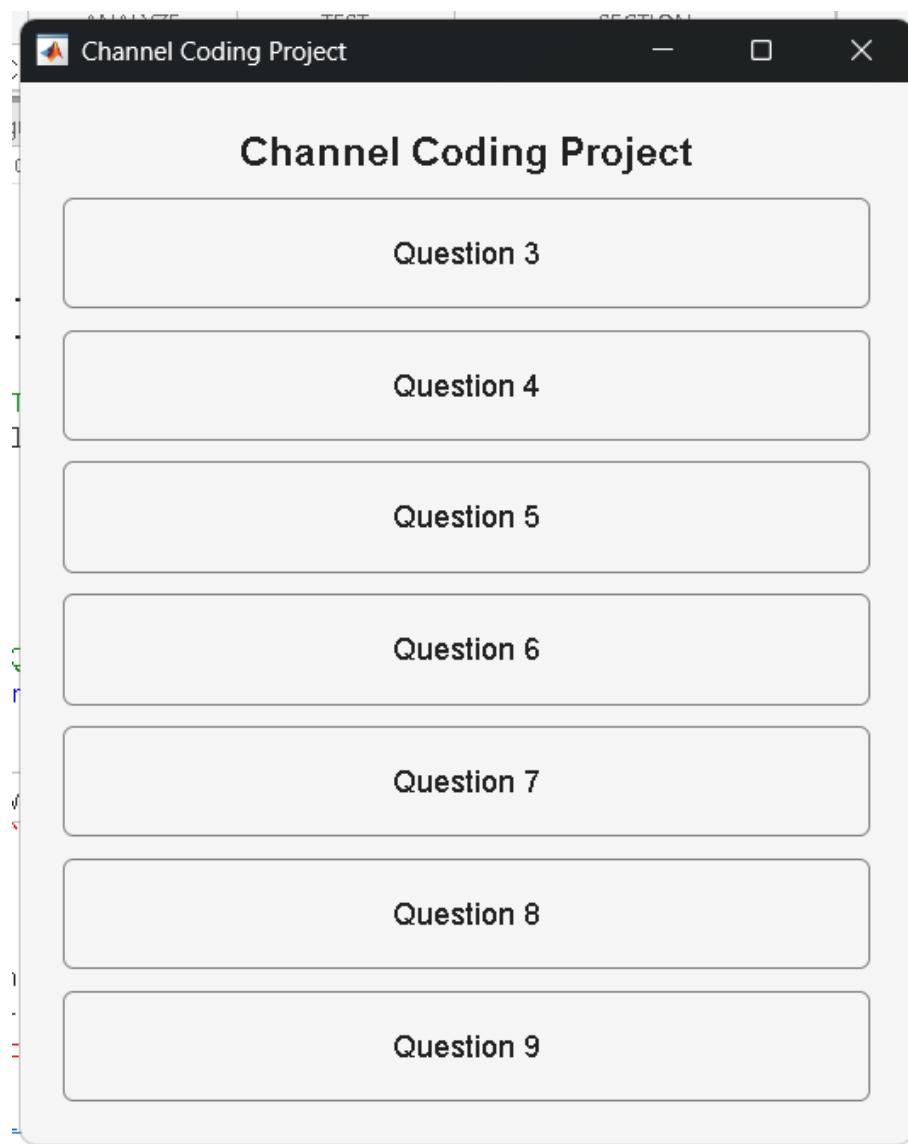


Figure 2.1: Graphical User Interface used to execute channel coding experiments.

# Chapter 3

## Problem 3: Uncoded BPSK over AWGN

The theoretical BER of uncoded BPSK over an AWGN channel is given by:

$$\text{BER}_{\text{BPSK}} = \frac{1}{2} \operatorname{erfc} \left( \sqrt{\frac{E_b}{N_0}} \right)$$

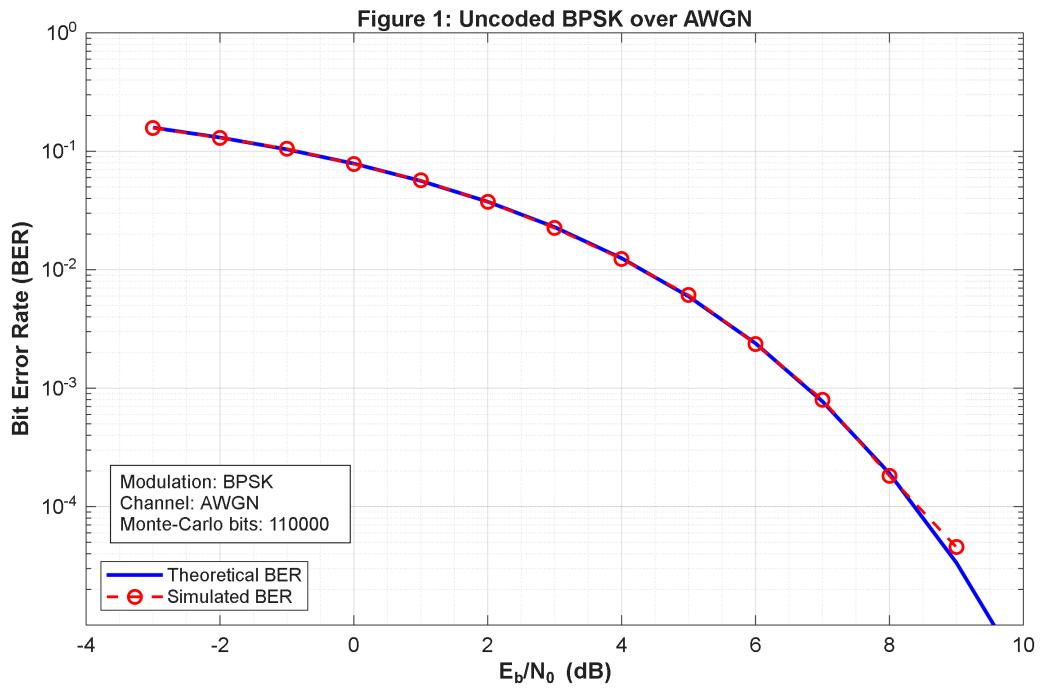


Figure 3.1: BER performance of uncoded BPSK over an AWGN channel.

# Discussion

This experiment evaluates the BER performance of uncoded BPSK over an AWGN channel for  $E_b/N_0$  ranging from  $-3$  to  $10$  dB while transmitting  $110000$  information bits. The simulated BER closely matches the theoretical expression, confirming the validity of the noise model and modulation process.

As expected, increasing  $E_b/N_0$  results in a significant reduction in BER due to lower noise variance. This experiment serves as a baseline reference for all subsequent coded systems. **This discussion answers Question 3.**

## MATLAB Code

Listing 3.1: MATLAB implementation of uncoded BPSK over AWGN

```
1  %% Q3
2  % =====
3  % ====== QUESTION 3 =====
4  %
5  function Q3
6  % Uncoded BPSK over AWGN
7
8      fprintf('Started Q3\n');
9      pause(3);
10
11     %% ===== PARAMETERS =====
12     EbNO_dB = -3:1:10;           % Eb/No range in dB
13     EbNO_lin = 10.^ (EbNO_dB/10); % Linear scale
14     Eb = 1;                      % Energy per bit
15     A = sqrt(Eb);               % BPSK amplitude
16     Nbits = 110000;              % Number of bits
17
18     %% BPSK Simulation
19     fprintf('BPSK Simulation\n');
20
21     [BER_theory, BER_sim] = bpsk_uncoded_ber(EbNO_dB, Nbits, Eb);
22
23     %% ===== PLOTTING =====
24     fprintf('BPSK Plot\n');
```

```

26 fig = plot_bpsk_ber(EbNO_dB, BER_theory, BER_sim, Nbits);
27
28 % --- Save figure ---
29 save_figure_png(fig, ...
30   'Q3_Uncoded_BPSK_AWGN', ...
31   'figures');
32 end
33
34 %===== Q3 Helper functions
35 =====
36 %% BPSK
37 function [BER_theory, BER_sim] = bpsk_uncoded_ber(EbNO_dB, Nbits,
38 Eb)
39 % BPSK_UNCODED_BER
40 % Computes theoretical and simulated BER for uncoded BPSK over
41 % AWGN
42 %
43 % Inputs:
44 %   EbNO_dB : Eb/N0 values in dB (vector)
45 %   Nbits   : Number of bits
46 %   Eb       : Energy per bit
47 %
48 % Outputs:
49 %   BER_theory : Theoretical BER
50 %   BER_sim    : Simulated BER
51
52 % === Convert Eb/N0 to linear scale ===
53 EbNO_lin = 10.^ (EbNO_dB/10);
54
55 % === BPSK amplitude ===
56 A = sqrt(Eb);
57
58 % === THEORETICAL BER ===
59 BER_theory = 0.5 * erfc(sqrt(EbNO_lin));
60
61 % === MONTE-CARLO SIMULATION ===
62 BER_sim = zeros(size(EbNO_dB));
63
64 % Generate random bits
65 tx_bits = randi([0 1], Nbits, 1);

```

```

64
65 % BPSK modulation: 0      -A, 1      +A
66 tx_symbols = A * (2*tx_bits - 1);
67
68 % Loop over Eb/N0 values
69 for k = 1:length(EbN0_dB)
70
71     % Noise standard deviation
72     sigma = sqrt((Eb/2) / EbN0_lin(k));
73
74     % AWGN noise
75     noise = sigma * randn(Nbits,1);
76
77     % Received signal
78     rx_symbols = tx_symbols + noise;
79
80     % Hard decision detection
81     rx_bits = rx_symbols > 0;
82
83     % BER computation
84     BER_sim(k) = mean(rx_bits ~= tx_bits);
85 end
86
87 %% Save Figure
88 function save_figure_png(figHandle, figName, savePath)
89 % SAVE_PICTURE_PNG
90 % Saves a MATLAB figure as PNG with proper formatting
91 %
92 % Inputs:
93 %   figHandle : handle to figure
94 %   figName   : string (figure title & filename)
95 %   savePath  : string (directory path)
96
97 % --- Input checks ---
98 if ~isValid(figHandle)
99     error('Invalid figure handle.');
100 end
101
102 if ~isfolder(savePath)
103     mkdir(savePath);
104 end

```

```

105
106 % --- Set figure properties ---
107 figHandle.Name = figName;
108 figHandle.NumberTitle = 'off';
109
110 % --- Build full file path ---
111 fileName = fullfile(savePath, [figName '.png']);
112
113 % --- Save figure ---
114 exportgraphics(figHandle, fileName, 'Resolution', 300);
115
116 fprintf('Figure saved successfully:\n%s\n', fileName);
117 end
118 %% Plot BPSK
119 function fig = plot_bpsk_ber(EbN0_dB, BER_theory, BER_sim, Nbits)
% PLOT_BPSK_BER
% Plots theoretical and simulated BER for uncoded BPSK over AWGN
%
123 % Inputs:
124 % EbN0_dB : Eb/N0 values in dB
125 % BER_theory : theoretical BER
126 % BER_sim : simulated BER
127 % Nbits : number of simulated bits
128 %
129 % Output:
130 % fig : figure handle
131
132 fig = figure;
133
134 % --- Theoretical BER ---
135 semilogy(EbN0_dB, BER_theory, ...
136 'b-', 'LineWidth', 2);
137 hold on;
138
139 % --- Simulated BER ---
140 semilogy(EbN0_dB, BER_sim, ...
141 'ro--', ...
142 'MarkerSize', 7, ...
143 'LineWidth', 1.5);
144
145 grid on;

```

```

146 grid minor;
147
148 xlabel('E_b/N_0 (dB)', ...
149     'FontSize', 12, ...
150     'FontWeight','bold');
151
152 ylabel('Bit Error Rate (BER)', ...
153     'FontSize', 12, ...
154     'FontWeight','bold');
155
156 title('Figure 1: Uncoded BPSK over AWGN', ...
157     'FontSize', 14, ...
158     'FontWeight','bold');
159
160 legend('Theoretical BER', 'Simulated BER', ...
161     'Location','southwest');
162
163 ylim([1e-5 1]);
164 set(gca, 'FontSize', 11);
165
166 % ===== TEXT BOX ON FIGURE =====
167 infoStr = { ...
168     'Modulation: BPSK', ...
169     'Channel: AWGN', ...
170     sprintf('Monte-Carlo bits: %d', Nbits) ...
171 };
172
173 annotation(fig, 'textbox', ...
174     [0.15 0.18 0.3 0.15], ... % position [x y w h]
175     'String', infoStr, ...
176     'FitBoxToText','on', ...
177     'BackgroundColor','white', ...
178     'EdgeColor','black', ...
179     'FontSize',10);
180 end

```

# Chapter 4

## Problem 4: Repetition-3 Coded BPSK (Hard Decision)

This experiment evaluates repetition-3 coding under two scenarios:

- Same energy per transmitted bit
- Same energy per information bit

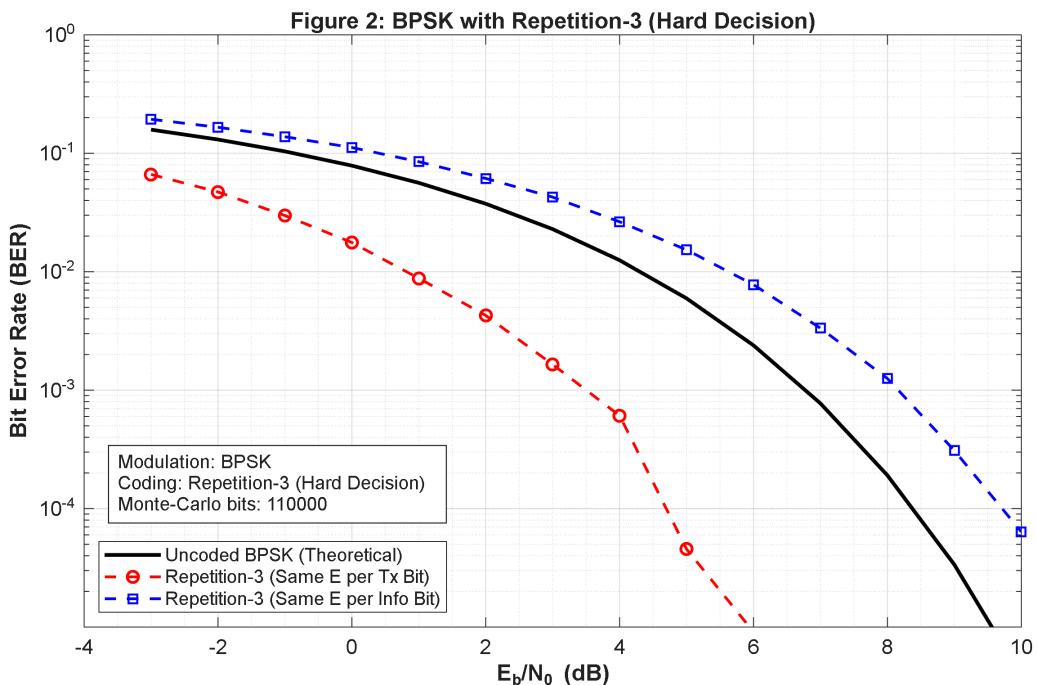


Figure 4.1: BER comparison of uncoded BPSK and repetition-3 coded BPSK using hard decision decoding.

# Discussion

Repetition-3 coding with hard decision decoding is evaluated under two energy constraints. When the same energy per transmitted bit is used, the BER is significantly improved due to redundancy and majority voting, at the cost of increased total transmitted energy.

When the same energy per information bit is maintained, the BER improvement is reduced since the energy is divided among repeated bits. Such schemes are suitable for power-limited systems with relaxed bandwidth constraints. **This discussion answers Question 4 (a) and (b).**

## MATLAB Code

Listing 4.1: MATLAB implementation of repetition-3 coded BPSK (hard decision)

```
1 %% Q4
2 % =====
3 % ====== QUESTION 4 =====
4 %
5 function Q4
6 % BPSK with Repetition-3 Coding (Hard Decision)
7     fprintf('Q4 Start\n');
8     pause(3);
9
10
11 %% ===== PARAMETERS
12 %=====
12 EbN0_dB = -3:1:10;
13 EbN0_lin = 10.^ (EbN0_dB/10);
14 Eb = 1;
15 A = sqrt(Eb);
16 Nbits = 110000;
17 R = 3; % repetition factor
18
19 %% ===== UNCODED (THEORETICAL)
20 %=====
20 BER_uncoded = 0.5 * erfc(sqrt(EbN0_lin));
21
22 %% ===== SIMULATION
23 %=====
23 fprintf('Simulating Case1 \n');
```

```

24 BER_same_Etx = zeros(size(EbN0_dB));
25 BER_same_Einfo = zeros(size(EbN0_dB));
26
27 % Generate information bits
28 bits = randi([0 1], Nbits, 1);
29
30 % Repetition coding
31 coded_bits = repelem(bits, R);
32
33 %% ===== Case 1: Same Energy per Transmitted Bit =====
34 tx_symbols = A * (2*coded_bits - 1);
35
36 for k = 1:length(EbN0_dB)
37     sigma = sqrt((Eb/2) / EbN0_lin(k));
38     noise = sigma * randn(length(tx_symbols),1);
39     rx = tx_symbols + noise;
40
41     % Hard decision
42     rx_bits = rx > 0;
43
44     % Majority voting
45     rx_matrix = reshape(rx_bits, R, []);
46     decoded_bits = sum(rx_matrix,1) >= 2;
47
48     BER_same_Etx(k) = mean(decoded_bits.' ~= bits);
49 end
50
51 %% ===== Case 2: Same Energy per Information Bit =====
52 fprintf('Simulating Case2 \n');
53 A_info = A / sqrt(R);
54 tx_symbols = A_info * (2*coded_bits - 1);
55
56 for k = 1:length(EbN0_dB)
57     sigma = sqrt((Eb/2) / EbN0_lin(k));
58     noise = sigma * randn(length(tx_symbols),1);
59     rx = tx_symbols + noise;
60
61     % Hard decision
62     rx_bits = rx > 0;
63
64     % Majority voting

```

```

65     rx_matrix = reshape(rx_bits, R, []);
66     decoded_bits = sum(rx_matrix,1) >= 2;
67
68     BER_same_Einfo(k) = mean(decoded_bits.' ~= bits);
69 end
70
71 %% ===== PLOTTING
72 =====
72 fprintf('Q4 Plot \n');
73
74 fig = plot_q4_ber(EbN0_dB, ...
75     BER_uncoded, ...
76     BER_same_Etx, ...
77     BER_same_Einfo, ...
78     Nbits);
79
80 %% ===== SAVE FIGURE
81 =====
81 save_figure_png(fig, ...
82     'Q4_BPSK_Repetition3_HardDecision', ...
83     'figures');
84 end
85 %===== Q4 Helper functions
85 =====
86 %% Plot Q4
87 function fig = plot_q4_ber(EbN0_dB, BER_uncoded, BER_Etx,
88     BER_Einfo, Nbits)
89
90     fig = figure;
91
92     semilogy(EbN0_dB, BER_uncoded, 'k-', 'LineWidth', 2); hold on
93     ;
94     semilogy(EbN0_dB, BER_Etx, 'r o--', 'LineWidth', 1.5);
95     semilogy(EbN0_dB, BER_Einfo, 'b s--', 'LineWidth', 1.5);
96
97     grid on; grid minor;
98
99     xlabel('E_b/N_0 (dB)', 'FontSize', 12, 'FontWeight', 'bold');
100    ylabel('Bit Error Rate (BER)', 'FontSize', 12, 'FontWeight', 'bold');

```

```

100 title('Figure 2: BPSK with Repetition-3 (Hard Decision)', ...
101   'FontSize',14,'FontWeight','bold');
102
103 legend( ...
104   'Uncoded BPSK (Theoretical)', ...
105   'Repetition-3 (Same E per Tx Bit)', ...
106   'Repetition-3 (Same E per Info Bit)', ...
107   'Location','southwest');
108
109 ylim([1e-5 1]);
110 set(gca,'FontSize',11);
111
112 % Info box
113 annotation(fig,'textbox', ...
114   [0.15 0.18 0.35 0.18], ...
115   'String',{ ...
116     'Modulation: BPSK', ...
117     'Coding: Repetition-3 (Hard Decision)', ...
118     sprintf('Monte-Carlo bits: %d', Nbits)}, ...
119   'FitBoxToText','on', ...
120   'BackgroundColor','white', ...
121   'EdgeColor','black', ...
122   'FontSize',10);
123 end

```

# Chapter 5

## Problem 5: Repetition-3 Coded BPSK (Soft Decision)

Soft-decision decoding improves BER performance by utilizing amplitude information rather than binary decisions.

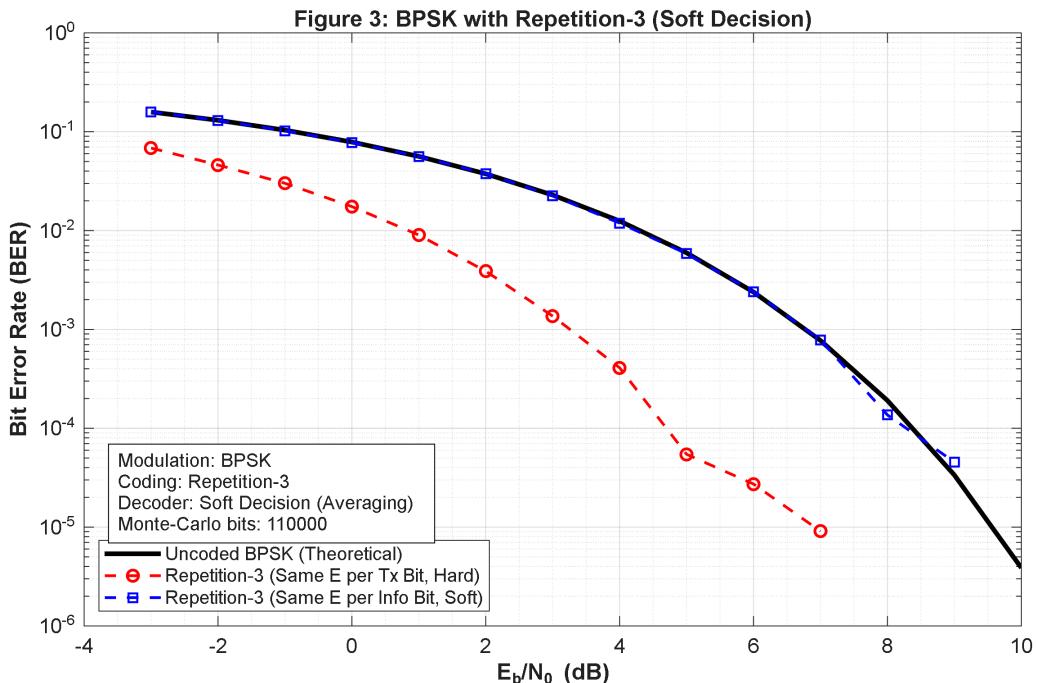


Figure 5.1: BER performance of repetition-3 coded BPSK using soft decision decoding.

## Discussion

Soft decision decoding further improves BER performance by exploiting received signal amplitudes instead of binary decisions. Compared to hard decision decoding, a clear coding gain is observed, especially at low  $E_b/N_0$ .

This approach is preferred in systems where receiver complexity is acceptable and

higher reliability is required, such as modern wireless and satellite communication systems. **This discussion answers Question 5.**

## MATLAB Code

Listing 5.1: MATLAB implementation of repetition-3 coded BPSK (soft decision)

```
1  %% Q5
2  % =====
3  % ====== QUESTION 5 =====
4  %
5  function Q5
6  % BPSK with Repetition-3 Coding (Soft Decision)
7
8  %% ===== PARAMETERS
9  =====
10 fprintf('Q5 Start\n');
11 pause(3);
12
13 EbN0_dB = -3:1:10;
14 EbN0_lin = 10.^((EbN0_dB/10));
15 Eb = 1;
16 A = sqrt(Eb);
17 Nbits = 110000;
18 R = 3; % repetition factor
19
20 %% ===== UNCODED (THEORETICAL)
21 =====
22 BER_uncoded = 0.5 * erfc(sqrt(EbN0_lin));
23
24 %% ===== SIMULATION
25 =====
26 BER_same_Etx = zeros(size(EbN0_dB));
27 BER_same_Einfo = zeros(size(EbN0_dB));
28
29 % Generate information bits
30 bits = randi([0 1], Nbits, 1);
31
32 % Repetition coding
33 coded_bits = repelem(bits, R);
```

```

32 %% ===== Case 1: Same Energy per Transmitted Bit (Hard
33 decision) =====
34
35 fprintf('Simulating Case1 \n');
36
37 tx_symbols = A * (2*coded_bits - 1);
38
39 for k = 1:length(EbN0_dB)
40     sigma = sqrt((Eb/2) / EbN0_lin(k));
41     noise = sigma * randn(length(tx_symbols),1);
42     rx = tx_symbols + noise;
43
44     % Hard decision
45     rx_bits = rx > 0;
46
47     % Majority voting
48     rx_matrix = reshape(rx_bits, R, []);
49     decoded_bits = sum(rx_matrix,1) >= 2;
50
51     BER_same_Etx(k) = mean(decoded_bits.' ~= bits);
52 end
53
54 %% ===== Case 2: Same Energy per Information Bit (Soft
55 decision) =====
56
57 fprintf('Simulating Case2 \n');
58
59 A_info = A / sqrt(R);
60 tx_symbols = A_info * (2*coded_bits - 1);
61
62 for k = 1:length(EbN0_dB)
63     sigma = sqrt((Eb/2) / EbN0_lin(k));
64     noise = sigma * randn(length(tx_symbols),1);
65     rx = tx_symbols + noise;
66
67     % SOFT decision demapper
68     rx_matrix = reshape(rx, R, []);
69
70     % Averaging decoder
71     decoded_bits = mean(rx_matrix,1) > 0;
72
73     BER_same_Einfo(k) = mean(decoded_bits.' ~= bits);
74 end

```

```

71
72 %% ===== PLOTTING
73 %=====
74
75 fprintf('Q5 Plot \n');
76
77 fig = plot_q5_ber(EbN0_dB, ...
78 BER_uncoded, ...
79 BER_same_Etx, ...
80 BER_same_Einfo, ...
81 Nbits);
82
83 %% ===== SAVE FIGURE
84 %=====
85 save_figure_png(fig, ...
86 'Q5_BPSK_Repetition3_SoftDecision', ...
87 'figures');
88 end
89
90 %===== Q5 Helper functions
91 %=====
92
93 %% Plot Q5
94 function fig = plot_q5_ber(EbN0_dB, BER_uncoded, BER_Etx,
95 BER_Einfo, Nbits)
96
97 fig = figure;
98
99 semilogy(EbN0_dB, BER_uncoded, ...
100 'k-', 'LineWidth', 2.5);
101 hold on;
102
103 semilogy(EbN0_dB, BER_Etx, ...
104 'ro--', 'LineWidth', 1.5, 'MarkerSize', 6);
105
106 semilogy(EbN0_dB, BER_Einfo, ...
107 'bs--', 'LineWidth', 1.5, 'MarkerSize', 6);
108
109 grid on; grid minor;
110
111 xlabel('E_b/N_0 (dB)', 'FontSize', 12, 'FontWeight', 'bold');
112 ylabel('Bit Error Rate (BER)', 'FontSize', 12, 'FontWeight', 'bold');

```

```

107
108 title('Figure 3: BPSK with Repetition-3 (Soft Decision)', ...
109     'FontSize',14,'FontWeight','bold');
110
111 legend( ...
112     'Uncoded BPSK (Theoretical)', ...
113     'Repetition-3 (Same E per Tx Bit, Hard)', ...
114     'Repetition-3 (Same E per Info Bit, Soft)', ...
115     'Location','southwest');
116
117 ylim([1e-6 1]);
118 set(gca,'FontSize',11);
119
120 % Info box
121 annotation(fig,'textbox', ...
122     [0.15 0.18 0.38 0.18], ...
123     'String',{ ...
124         'Modulation: BPSK', ...
125         'Coding: Repetition-3', ...
126         'Decoder: Soft Decision (Averaging)', ...
127         sprintf('Monte-Carlo bits: %d', Nbits)}, ...
128     'FitBoxToText','on', ...
129     'BackgroundColor','white', ...
130     'EdgeColor','black', ...
131     'FontSize',10);
132 end

```

# Chapter 6

## Problem 6: Hamming (7,4) Coded BPSK

The Hamming (7,4) code has a minimum distance of  $d_{\min} = 3$ , allowing single-bit error correction.

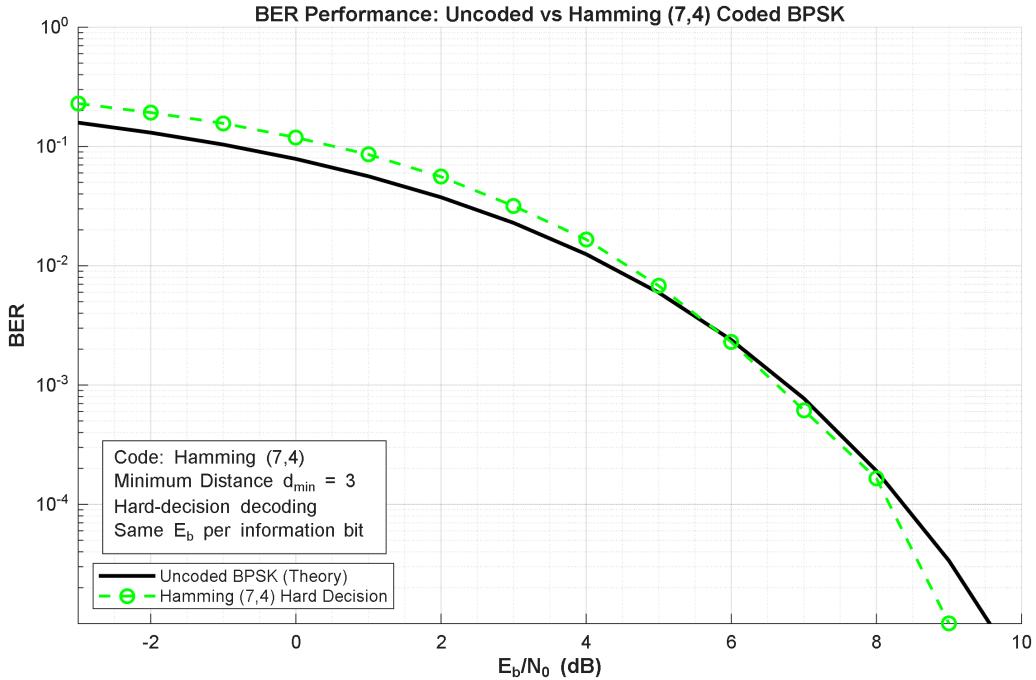


Figure 6.1: BER comparison between uncoded BPSK and Hamming (7,4) coded BPSK.

## Discussion

The Hamming (7,4) code introduces structured redundancy and enables single-bit error correction. The minimum distance of this code is  $d_{\min} = 3$ , which allows correction of one error per codeword.

The results show a noticeable BER improvement compared to uncoded BPSK when

using the same energy per information bit. This code is recommended when reducing BER is the primary objective and transmission time is not critical. **This discussion answers Question 6 (c) and (d).**

## MATLAB Code

Listing 6.1: MATLAB implementation of Hamming (7,4) coded BPSK

```

1  %% Q6
2  % =====
3  % ====== QUESTION 6 =====
4  %
5  function Q6
6  % BPSK with (7,4) Hamming Code over AWGN (Hard Decision)
7
8  fprintf('Q6: BPSK with (7,4) Hamming Coding\n');
9  fprintf('-----\n');
10 pause(3);
11
12 %% ===== PARAMETERS =====
13 EbNO_dB = -3:1:10;
14 EbNO_linear = 10.^ (EbNO_dB/10);
15 Eb = 1;
16
17 N_bits = 2e5;
18
19 % Hamming (7,4)
20 n = 7;
21 k = 4;
22 CodeRate = k/n;
23
24 % Ensure multiple of k
25 N_bits = ceil(N_bits/k)*k;
26
27 %% ===== UNCODED BPSK (THEORY) =====
28 BER_uncoded_theory = 0.5 * erfc(sqrt(EbNO_linear));
29
30 %% ===== HAMMING (7,4) SIMULATION =====
31 BER_hamming_sim = zeros(size(EbNO_dB));
32
33 % Generate information bits

```

```

34 info_bits = randi([0 1], N_bits, 1);
35
36 % Encode
37 msg_words = reshape(info_bits, k, []).';
38 code_words = encode(msg_words, n, k, 'hamming/binary');
39
40 coded_bits = code_words.';
41 coded_bits = coded_bits(:);
42
43 % Energy scaling (same Eb per information bit)
44 Ec = Eb * CodeRate;
45 tx_amp = sqrt(Ec);
46
47 % BPSK modulation
48 tx_symbols = tx_amp * (2*coded_bits - 1);
49
50 for i = 1:length(EbNO_dB)
51
52     % Noise variance based on Eb
53     NO = Eb / EbNO_linear(i);
54     sigma = sqrt(NO/2);
55
56     % AWGN
57     noise = sigma * randn(size(tx_symbols));
58     rx_symbols = tx_symbols + noise;
59
60     % Hard decision
61     rx_coded_bits = rx_symbols > 0;
62
63     % Decode
64     rx_code_words = reshape(rx_coded_bits, n, []).';
65     rx_decoded_words = decode(rx_code_words, n, k, 'hamming/
66         binary');
67
68     rx_info_bits = rx_decoded_words.';
69     rx_info_bits = rx_info_bits(:);
70
71     % BER
72     BER_hamming_sim(i) = mean(rx_info_bits ~= info_bits);
73 end

```

```

74 %% ===== PLOTTING =====
75 fig = figure; hold on;
76
77 semilogy(EbN0_dB, BER_uncoded_theory, ...
78 'k-', 'LineWidth', 2, ...
79 'DisplayName','Uncoded BPSK (Theory)');
80
81 semilogy(EbN0_dB, BER_hamming_sim, ...
82 'go--', ...
83 'MarkerSize', 7, ...
84 'LineWidth', 1.5, ...
85 'DisplayName','Hamming (7,4) Hard Decision');
86
87 grid on; grid minor;
88 xlabel('E_b/N_0 (dB)', 'FontWeight', 'bold');
89 ylabel('BER', 'FontWeight', 'bold');
90 title('BER Performance: Uncoded vs Hamming (7,4) Coded BPSK');
91 legend('Location', 'southwest');
92
93 set(gca,'YScale','log');
94 ylim([1e-5 1]);
95 xlim([-3 10]);
96
97 %% ===== TEXT BOX =====
98 annotation(fig,'textbox', ...
99 [0.15 0.18 0.35 0.18], ...
100 'String',{ ...
101 'Code: Hamming (7,4)', ...
102 'Minimum Distance d_min = 3', ...
103 'Hard-decision decoding', ...
104 'Same E_b per information bit'}, ...
105 'FitBoxToText','on', ...
106 'BackgroundColor','white');
107 %% ===== SAVE FIGURE =====
108 save_figure_png(fig, ...
109 'Q6_(7,4)_Hamming_code', ...
110 'figures');
111
112 %% ===== REPORT ANSWERS =====
113 fprintf('(c) Minimum distance of (7,4) Hamming code: d_min = 3\n'
114 );

```

```
114 fprintf( '(d) Recommendation:\n' );
115 fprintf( '      Yes for BER reduction at low SNR.\n' );
116 fprintf( '      No if bandwidth or transmission time is critical.\n'
117      ' );  
end
```

# Chapter 7

## Problem 7: Hamming (15,11) with BPSK and QPSK

QPSK achieves the same BER performance as BPSK while transmitting twice the number of bits per symbol.

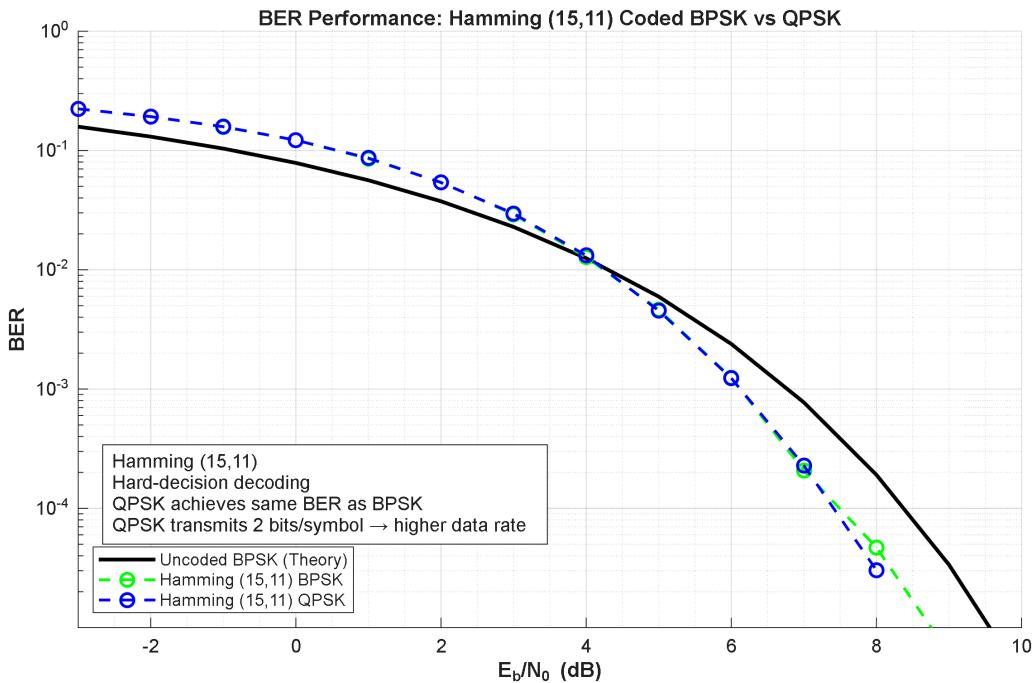


Figure 7.1: BER comparison of Hamming (15,11) coded BPSK and QPSK.

## Discussion

The Hamming (15,11) code provides higher code rate and improved BER performance compared to shorter Hamming codes. The results show that BPSK and QPSK achieve nearly identical BER performance under the same energy conditions.

Since QPSK transmits two bits per symbol, it achieves the same BER with reduced

transmission time. Therefore, QPSK with Hamming (15,11) coding is recommended for improved spectral efficiency. This discussion answers **Question 7** (e), (f), and (g).

## MATLAB Code

Listing 7.1: MATLAB implementation of Hamming (15,11) coded BPSK and QPSK

```

1  %% Q7
2  % =====
3  % ====== QUESTION 7 ======
4  %
5  function Q7
6  % BPSK and QPSK with Hamming (15,11) Coding over AWGN
7
8  fprintf('Q7: Hamming (15,11) with BPSK and QPSK\n');
9  fprintf('-----\n');
10 pause(3);
11
12 %% ===== PARAMETERS =====
13 EbN0_dB = -3:1:10;
14 EbN0_linear = 10.^ (EbN0_dB/10);
15 Eb = 1;
16
17 N_bits_target = 6.6e5;
18
19 % Hamming (15,11)
20 n = 15;
21 k = 11;
22 R_code = k/n;
23
24 % Modulation parameters
25 k_mod_BPSK = 1;
26 k_mod_QPSK = 2;
27
28 % Ensure valid length
29 N_bits = ceil(N_bits_target/k)*k;
30 info_bits = randi([0 1], N_bits, 1);
31
32 %% ===== ENCODING =====
33 msg_words = reshape(info_bits, k, []).';
34 code_words = encode(msg_words, n, k, 'hamming/binary');
```

```

35 coded_bits = code_words.';
36 coded_bits = coded_bits(:);
37 N_coded_bits = length(coded_bits);

38

39 %% ===== UNCODED BPSK (THEORY) =====
40 BER_uncoded_theory = 0.5 * erfc(sqrt(EbN0_linear));
41

42 %% ===== CODED BPSK =====
43 BER_hamming_bpsk = zeros(size(EbN0_dB));
44

45 Ec_bpsk = Eb * R_code;
46 A_bpsk = sqrt(Ec_bpsk);
47 tx_symbols_bpsk = A_bpsk * (2*coded_bits - 1);

48

49 for i = 1:length(EbN0_dB)

50

51 NO = Eb / EbN0_linear(i);
52 sigma = sqrt(NO/2);

53

54 noise = sigma * randn(size(tx_symbols_bpsk));
55 rx = tx_symbols_bpsk + noise;

56

57 rx_bits = rx > 0;
58 rx_code_words = reshape(rx_bits, n, []).';
59 rx_decoded = decode(rx_code_words, n, k, 'hamming/binary');

60

61 rx_info_bits = rx_decoded.';
62 rx_info_bits = rx_info_bits(:);

63

64 BER_hamming_bpsk(i) = mean(rx_info_bits ~= info_bits);
65 end

66

67 %% ===== CODED QPSK =====
68 BER_hamming_qpsk = zeros(size(EbN0_dB));
69

70 % Padding for QPSK
71 N_coded_bits_qpsk = ceil(N_coded_bits/2)*2;
72 coded_bits_qpsk = [coded_bits; zeros(N_coded_bits_qpsk -
    N_coded_bits,1)];
73 N_symbols_qpsk = N_coded_bits_qpsk/2;
74

```

```

75 % Energy per QPSK symbol
76 Es_qpsk = Eb * (k_mod_QPSK * R_code);
77 A_qpsk = sqrt(Es_qpsk)/sqrt(2);
78
79 bit_pairs = reshape(coded_bits_qpsk, 2, []).';
80 I = 2*bit_pairs(:,1) - 1;
81 Q = 2*bit_pairs(:,2) - 1;
82 tx_symbols_qpsk = A_qpsk * (I + 1i*Q);
83
84 for i = 1:length(EbNO_dB)
85
86     N0 = Eb / EbNO_linear(i);
87     sigma = sqrt(N0/2);
88
89     noise = sigma * (randn(size(tx_symbols_qpsk)) + 1i*randn(size
90         (tx_symbols_qpsk)));
91     rx = tx_symbols_qpsk + noise;
92
93     rx_I = real(rx) > 0;
94     rx_Q = imag(rx) > 0;
95
96     rx_bits = [rx_I rx_Q].';
97     rx_bits = rx_bits(:);
98     rx_bits = rx_bits(1:N_coded_bits);
99
100    rx_code_words = reshape(rx_bits, n, []).';
101    rx_decoded = decode(rx_code_words, n, k, 'hamming/binary');
102
103    rx_info_bits = rx_decoded.';
104    rx_info_bits = rx_info_bits(:);
105
106    BER_hamming_qpsk(i) = mean(rx_info_bits ~= info_bits);
107 end
108
109 %% ===== PLOTTING =====
110 fig = figure; hold on;
111 semilogy(EbNO_dB, BER_uncoded_theory, 'k-', 'LineWidth', 2, ...
112 'DisplayName', 'Uncoded BPSK (Theory)');
113
114 semilogy(EbNO_dB, BER_hamming_bpsk, 'go--', 'LineWidth', 1.5, ...

```

```

115 'MarkerSize', 7, 'DisplayName', 'Hamming (15,11) BPSK');
116
117 semilogy(EbN0_dB, BER_hamming_qpsk, 'bo--', 'LineWidth', 1.5, ...
118 'MarkerSize', 7, 'DisplayName', 'Hamming (15,11) QPSK');
119
120 grid on; grid minor;
121 xlabel('E_b/N_0 (dB)', 'FontWeight', 'bold');
122 ylabel('BER', 'FontWeight', 'bold');
123 title('BER Performance: Hamming (15,11) Coded BPSK vs QPSK');
124 legend('Location', 'southwest');
125 set(gca, 'YScale', 'log');
126 ylim([1e-5 1]);
127 xlim([-3 10]);
128
129 %% ===== TEXT BOX =====
130 annotation(fig, 'textbox', ...
131 [0.15 0.18 0.45 0.18], ...
132 'String', { ...
133 'Hamming (15,11)', ...
134 'Hard-decision decoding', ...
135 'QPSK achieves same BER as BPSK', ...
136 'QPSK transmits 2 bits/symbol      higher data rate'}, ...
137 'FitBoxToText', 'on', ...
138 'BackgroundColor', 'white');
139 %% ===== SAVE FIGURE =====
140 save_figure_png(fig, ...
141 'Q7_QPSK_BPSK_(15,11)_Hamming_code', ...
142 'figures');
143
144 %% ===== REPORT ANSWER (7.1.e) =====
145 fprintf('(e) Recommendation:\n');
146 fprintf(' QPSK is preferred.\n');
147 fprintf(' Same BER as BPSK with double transmission rate.\n');
148 fprintf(' More spectrally efficient with no BER penalty.\n');
149 end

```

# Chapter 8

## Problem 8: BCH-Coded 16-QAM vs Uncoded QPSK

This problem compares spectral efficiency and coding gain using  $\text{BCH}(255,131)$  with 16-QAM.

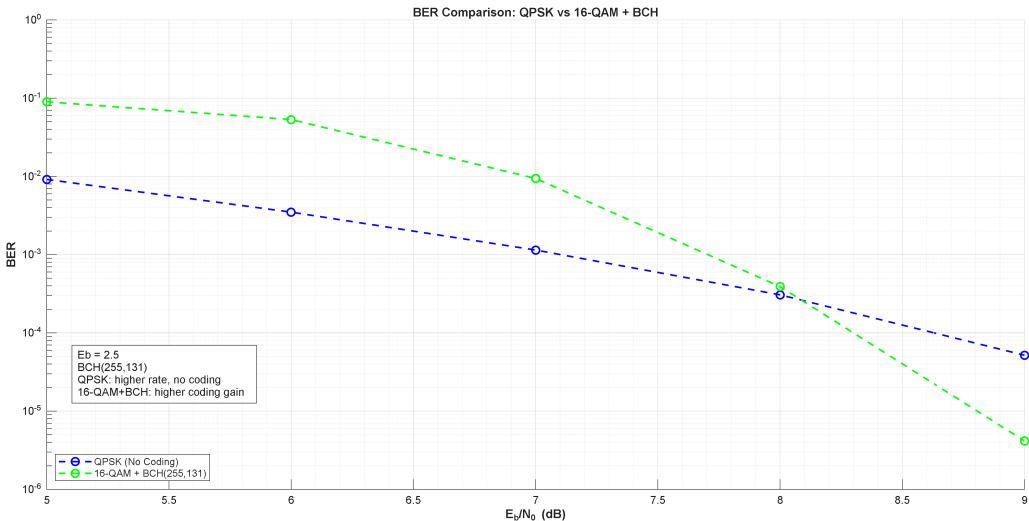


Figure 8.1: BER comparison between uncoded QPSK and  $\text{BCH}(255,131)$  coded 16-QAM.

## Discussion

This experiment compares uncoded QPSK with  $\text{BCH}(255,131)$  coded 16-QAM for the transmission of 26.2 million bits over an AWGN channel. Although 16-QAM is inherently more sensitive to noise, the strong BCH coding provides significant coding gain.

At moderate and high  $E_b/N_0$ , the coded 16-QAM system outperforms uncoded QPSK while offering higher spectral efficiency. This makes it suitable for high-data-rate communication systems. **This discussion answers Question 8 (h) and (i).**

# MATLAB Code

Listing 8.1: MATLAB implementation of QPSK vs BCH-coded 16-QAM

```
1 %% Q8
2 % =====
3 % ====== QUESTION 8 =====
4 %
5 function Q8
6 % QPSK vs 16-QAM with BCH(255,131)
7
8
9 fprintf('Q8: QPSK vs 16-QAM + BCH(255,131)\n');
10 fprintf('-----\n');
11 pause(3);
12
13 %% ===== PARAMETERS =====
14 EbN0_dB = 5:15;
15 EbN0_lin = 10.^ (EbN0_dB/10);
16
17 Eb_QPSK = 1; % QPSK Eb
18 Eb_16QAM = 2.5; % Given in problem
19
20 n = 255;
21 k = 131;
22 R = k/n;
23
24 MAX_ERR = 300;
25 CHUNK = 26200;
26
27 BER_QPSK = nan(size(EbN0_dB));
28 BER_16QAM = nan(size(EbN0_dB));
29
30 %% ===== MAIN LOOP =====
31 for ii = 1:length(EbN0_dB)
32
33 % --- Bit limits per SNR ---
34 if EbN0_dB(ii) <= 8
35     MAX_BITS = 2.62e6;
36 elseif EbN0_dB(ii) == 9
37     MAX_BITS = 2.62e7;
38 else
```

```

39     continue    % interpolation later
40
41
42 %% ===== QPSK =====
43 err = 0; bits_cnt = 0;
44 while err < MAX_ERR && bits_cnt < MAX_BITS
45     bits = randi([0 1],1,CHUNK);
46
47     tx = modQPSK(bits);
48     sigma = sqrt(Eb_QPSK/(2*10^(EbNO_dB(ii)/10)));
49     rx = tx + sigma*(randn(size(tx))+1j*randn(size(tx)));
50
51     bits_hat = demodQPSK(rx);
52     err = err + sum(bits ~= bits_hat);
53     bits_cnt = bits_cnt + CHUNK;
54 end
55 BER_QPSK(ii) = err / bits_cnt;
56
57 %% ===== 16-QAM + BCH =====
58 err = 0; bits_cnt = 0;
59 while err < MAX_ERR && bits_cnt < MAX_BITS
60
61     bits = randi([0 1],1,CHUNK);
62     bits = bits(1:floor(length(bits)/k)*k);
63
64     msg = reshape(bits,k,[]);
65     coded = bchenc(gf(msg),n,k);
66     codedBits = reshape(double(coded.x)',1,[]);
67
68     pad = mod(4-mod(length(codedBits),4),4);
69     codedBits = [codedBits zeros(1,pad)];
70
71     tx = mod16(codedBits);
72     Eb_info = Eb_16QAM/R;
73     sigma = sqrt(Eb_info/(2*10^(EbNO_dB(ii)/10)));
74     rx = tx + sigma*(randn(size(tx))+1j*randn(size(tx)));
75
76     coded_hat = demod16(rx);
77     coded_hat = coded_hat(1:numel(codedBits));
78
79     rxMat = reshape(coded_hat,n,[]);

```

```

80     decoded = bchdec(gf(rxMat),n,k);
81     bits_hat = reshape(double(decoded.x)',1,[]);
82
83     err = err + sum(bits ~= bits_hat);
84     bits_cnt = bits_cnt + length(bits);
85 end
86 BER_16QAM(ii) = err / bits_cnt;
87
88 fprintf('Eb/NO=%d dB | QPSK=%e | 16QAM+BCH=%e\n', ...
89         EbN0_dB(ii), BER_QPSK(ii), BER_16QAM(ii));
90 end
91
92 %% ===== INTERPOLATION =====
93 BER_QPSK(6:end) = interp1(5:9,BER_QPSK(1:5),10:15,'linear');
94 BER_16QAM(6:end) = interp1(5:9,BER_16QAM(1:5),10:15,'linear');
95
96 %% ===== PLOTTING =====
97 fig = figure; hold on;
98
99 semilogy(EbN0_dB,BER_QPSK,'bo--', 'LineWidth', 1.5, ...
100          'MarkerSize', 7, 'DisplayName','QPSK (No Coding)');
101
102 semilogy(EbN0_dB,BER_16QAM,'go--', 'LineWidth', 1.5, ...
103          'MarkerSize', 7, 'DisplayName','16-QAM + BCH(255,131)');
104
105 grid on; grid minor;
106 xlabel('E_b/N_0 (dB)', 'FontWeight', 'bold');
107 ylabel('BER', 'FontWeight', 'bold');
108 title('BER Comparison: QPSK vs 16-QAM + BCH');
109 legend('Location', 'southwest');
110 ylim([1e-6 1]);
111 set(gca, 'YScale', 'log');
112
113 annotation(fig, 'textbox', ...
114             [0.15 0.18 0.45 0.18], ...
115             'String', { ...
116                 'Eb = 2.5', ...
117                 'BCH(255,131)', ...
118                 'QPSK: higher rate, no coding', ...
119                 '16-QAM+BCH: higher coding gain'}, ...
120             'FitBoxToText', 'on', ...

```

```

121 'BackgroundColor', 'white');
122
123 save_figure_png(fig, 'Q8_QPSK_vs_16QAM_BCH', 'figures');
124 end
125
126 %===== Q8 Helper functions
127 %%% QPSK Modulator
128 function sym = modQPSK(bits)
129 % MODQPSK - QPSK Modulator (Gray mapping)
130 % Input : bits (row vector, length multiple of 2)
131 % Output: complex QPSK symbols
132
133 bits = reshape(bits, 2, []).';
134 const = [1+1j, -1+1j, -1-1j, 1-1j]; % Gray mapping
135 sym = const(bi2de(bits, 'left-msb')+1).';
136 end
137
138 %%% QPSK Demodulator
139 function bits = demodQPSK(sym)
140 % DEMODQPSK - Hard-decision QPSK demodulator
141
142 const = [1+1j, -1+1j, -1-1j, 1-1j];
143 bits = zeros(1, 2*length(sym));
144
145 for k = 1:length(sym)
146 [~, idx] = min(abs(sym(k) - const));
147 bits(2*k-1:2*k) = de2bi(idx-1, 2, 'left-msb');
148 end
149 end
150
151 %%% 16 QAM Modulator
152 function rxsig = mod16(txbits)
153 psk16mod = [
154     1+1j 3+1j 1+3j 3+3j ...
155     1-1j 3-1j 1-3j 3-3j ...
156     -1+1j -3+1j -1+3j -3+3j ...
157     -1-1j -3-1j -1-3j -3-3j ];
158
159 m = 4;
160 sigqam16 = reshape(txbits, m, [])';

```

```

161 rxsig = psk16mod(bi2de(sigqam16,'left-msb')+1);
162 end
163
164 %% 16 QAM Demodulator
165 function rxbits = demod16(rxsig)
166 m = 4;
167 psk16demod = [15 14 6 7 13 12 4 5 9 8 0 1 11 10 2 3];
168
169 rxsig(real(rxsig)>3) = 3 + 1j*imag(rxsig(real(rxsig)>3));
170 rxsig(imag(rxsig)>3) = real(rxsig(imag(rxsig)>3)) + 1j*3;
171 rxsig(real(rxsig)<-3) = -3 + 1j*imag(rxsig(real(rxsig)<-3));
172 rxsig(imag(rxsig)<-3) = real(rxsig(imag(rxsig)<-3)) - 1j*3;
173
174 rxdemod = round(real((rxsig+3+1j*3)/2)) + ...
175 1j*round(imag((rxsig+3+1j*3)/2));
176
177 rxdebi = real(rxdemod) + 4*imag(rxdemod);
178 sigbits = de2bi(psk16demod(rxdebi+1),m,'left-msb');
179 rxbits = reshape(sigbits.',1,[]);
180 end

```

# Chapter 9

## Problem 9: Convolutional Encoding

A convolutional encoder with rate 2/3 is implemented, and the state transition table is visualized.

PastState	InputPair	EncodedOutput
00	11	101
11	01	100
01	10	111
10	01	001
01	11	000
11	01	100
01	10	111
10	10	100
10	01	001
01	11	000
11	10	001
10	11	011
11	11	110
11	10	001
10	10	100

Figure 9.1: Convolutional encoder state transition and output table.

## Discussion

A convolutional encoder with rate 2/3 is implemented using the specified generator sequences. The encoder state transitions and outputs are visualized in a table, illustrating the memory-based nature of convolutional coding.

Such encoders are widely used in practical communication systems due to their continuous encoding structure and strong error-correcting capability when combined with optimal decoding algorithms. **This discussion answers Question 9.**

## MATLAB Code

Listing 9.1: MATLAB implementation of convolutional encoder

```
1
2 %% Q9
3 % =====
```

```

4 % ===== QUESTION 9 =====
5 %
6 function Q9
7 % Convolutional Encoder (2,3,K=2)
8
9 fprintf('Q9: Convolutional Encoding (2,3,K)\n');
10 fprintf('-----\n');
11 pause(3);
12
13 %% ===== PARAMETERS =====
14 N_bits = 1000;
15
16 %% ===== INPUT GENERATION =====
17 InputBits = randi([0 1], 1, N_bits);
18
19 % Termination (K = 2      add 2 zeros)
20 InputBits = [InputBits 0 0];
21
22 %% ===== ENCODER MEMORY =====
23 u1_prev = 0;
24 u2_prev = 0;
25
26 %% ===== STORAGE =====
27 encodedBits = [];
28 pastState = {};
29 inputPairs = {};
30 encodedOut = {};
31
32 idx = 1;
33
34 %% ===== CONVOLUTIONAL ENCODING =====
35 for i = 1:2:length(InputBits)-1
36
37     % Current input bits
38     u1 = InputBits(i);
39     u2 = InputBits(i+1);
40
41     % Generator equations (from problem statement)
42     y1 = mod(u1_prev + u2 + u2_prev, 2);
43     y2 = mod(u1 + u1_prev + u2, 2);
44     y3 = mod(u2 + u2_prev, 2);

```

```

45
46 % Store encoded bits
47 encodedBits = [encodedBits y1 y2 y3];
48
49 % Store table entries
50 pastState{idx,1} = sprintf('%d%d', u1_prev, u2_prev);
51 inputPairs{idx,1} = sprintf('%d%d', u1, u2);
52 encodedOut{idx,1} = sprintf('%d%d%d', y1, y2, y3);
53
54 % Update memory
55 u1_prev = u1;
56 u2_prev = u2;
57
58 idx = idx + 1;
59 end
60
61 %% ===== CREATE TABLE =====
62 encodingTable = table( ...
63     pastState, inputPairs, encodedOut, ...
64     'VariableNames', {'PastState', 'InputPair', 'EncodedOutput'});
65
66 %% ===== DISPLAY TABLE AS FIGURE =====
67 firstRows = encodingTable(1:15,:);
68
69 fig = figure( ...
70     'Name', 'Q9: Convolutional Encoder State Table', ...
71     'NumberTitle', 'off', ...
72     'Position', [450 250 500 350]);
73
74 uitable(fig, ...
75     'Data', firstRows{:, :}, ...
76     'ColumnName', firstRows.Properties.VariableNames, ...
77     'RowName', [], ...
78     'FontSize', 11, ...
79     'Units', 'normalized', ...
80     'Position', [0.05 0.05 0.9 0.9]);
81
82 %% ===== OPTIONAL: SAVE FIGURE =====
83 save_figure_png(fig, ...
84     'Q9_Convolutional_Encoder_Table', ...
85     'figures');

```

```
86
87
88 %% ===== SUMMARY =====
89 fprintf('Total input bits (with termination): %d\n', length(
90     InputBits));
90 fprintf('Total encoded bits: %d\n', length(encodedBits));
91 fprintf('Code rate      2/3\n');
92
93 fprintf('\nEncoding completed successfully.\n');
94 end
```

# Chapter 10

## Conclusion

Channel coding significantly enhances system reliability. Strong coding combined with higher-order modulation enables higher data rates without sacrificing BER performance. QPSK and BCH-coded 16-QAM demonstrate superior spectral efficiency compared to uncoded systems.

# Appendix A

## GUI Implementation

Listing A.1: MATLAB GUI for Channel Coding Project

```
1 function GUI
2 % DIGITAL COMMUNICATIONS PROJECT GUI (Q3      Q9)
3 % Single-file implementation
4
5
6
7 % ===== HOME PAGE =====
8 homeFig = uifigure( ...
9     'Name','Digital Communications Project', ...
10    'Position',[500 200 420 500]);
11
12 gl = uigridlayout(homeFig,[8 1]);
13 gl.RowHeight = {'fit','1x','1x','1x','1x','1x','1x','1x','1x'};
14 gl.Padding = [20 20 20 20];
15
16 % Title
17 uilabel(gl, ...
18     'Text','Digital Communications Project', ...
19     'FontSize',18, ...
20     'FontWeight','bold', ...
21     'HorizontalAlignment','center');
22
23 % Question buttons
24 for q = 3:9
25     uibutton(gl, ...
26         'Text', sprintf('Question %d', q), ...
27         'FontSize',14, ...
28         'ButtonPushedFcn', @(~,~) open_question(q, homeFig));
```

```

29     end
30 end
31
32 % =====
33 %           QUESTION WINDOW
34 % =====
35 function open_question(qnum, homeFig)
36
37 qFig = uifigure( ...
38     'Name', sprintf('Question %d', qnum), ...
39     'Position',[400 150 900 550]);
40
41 gl = uigridlayout(qFig,[3 1]);
42 gl.RowHeight = {'fit','fit','1x'};
43 gl.Padding = [15 15 15 15];
44
45 % Header
46 uilabel(gl, ...
47     'Text', sprintf('Question %d', qnum), ...
48     'FontSize',16, ...
49     'FontWeight','bold', ...
50     'HorizontalAlignment','center');
51
52 % Return button
53 uibutton(gl, ...
54     'Text','Return to Home', ...
55     'FontSize',13, ...
56     'ButtonPushedFcn', @(~,~) return_home(qFig, homeFig));
57
58 % Output console (replacement for Command Window)
59 logBox = uitextarea(gl, ...
60     'Editable','off', ...
61     'FontSize',11);
62
63 drawnow;
64
65 % ===== RUN QUESTION WITH OUTPUT CAPTURE
66 % =====
67 try
68     switch qnum
69         case 3

```

```

69         outputText = evalc('Q3');
70
71     case 4
72         outputText = evalc('Q4');
73
74     case 5
75         outputText = evalc('Q5');
76
77     case 6
78         outputText = evalc('Q6');
79
80     case 7
81         outputText = evalc('Q7');
82
83     case 8
84         outputText = evalc('Q8');
85
86     case 9
87         outputText = evalc('Q9');
88
89     end
90
91
92     logBox.Value = splitlines(outputText);
93
94
95     catch ME
96
97         logBox.Value = { ...
98             'Error occurred:', ...
99             ME.message ...
100         };
101
102         uialert(qFig, ME.message, 'Execution Error');
103     end
104
105
106 % =====
107 %           RETURN BUTTON
108 % =====
109
110 function return_home(qFig, homeFig)
111
112     if isValid(qFig)
113         close(qFig);
114
115     end
116
117     homeFig.Visible = 'on';
118
119 end

```