US 20020051501A1

(19) United States

(12) Patent Application Publication

(10) Pub. No.: US 2002/0051501 A1

(43) Pub. Date: May 2, 2002

Demjanenko et al.

(54) USE OF TURBO-LIKE CODES FOR QAM MODULATION USING INDEPENDENT I AND Q DECODING TECHNIQUES AND APPLICATIONS TO XDSL SYSTEMS

(76) Inventors: **Victor Demjanenko**, Pendleton, NY (US); **Frederic J. Hirzel**, Clinton Township, MI (US); **Juan Alberto Torres**, Orchard Park, NY (US)

Correspondence Address:
David A. Blumenthal
FOLEY & LARDNER
Suite 3500
2029 Century Park East
Los Angeles, CA 90067-3021 (US)

(21) Appl. No.: **09/846,061**

(22) Filed: **Apr. 30, 2001**

Related U.S. Application Data

(63) Non-provisional of provisional application No. 60/200,369, filed on Apr. 28, 2000. Non-provisional of provisional application No. 60/248,099, filed on Nov. 13, 2000. Non-provisional of provisional application No. 60/242,393, filed on Oct. 20, 2000. Non-provisional of provisional application No. 60/244,550, filed on Oct. 31, 2000.

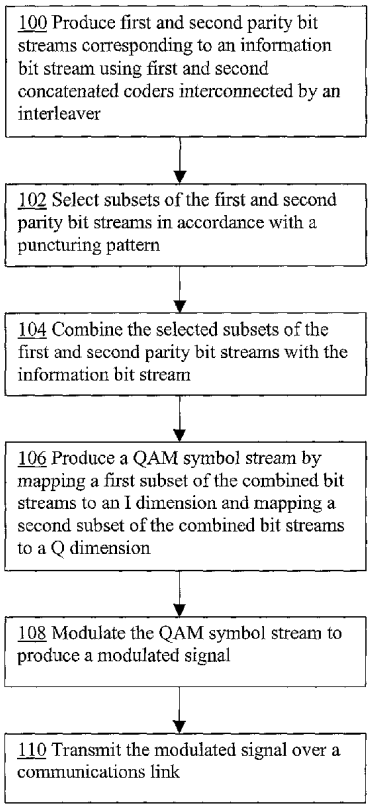
Publication Classification

(51) **Int. Cl.**[7] **H04L 27/36**; H03D 1/00; H04L 27/06

(52) **U.S. Cl.** **375/298**; 375/340

(57) **ABSTRACT**

A transmitter produces a modulated signal with forward error correction from an information bit stream in a QAM transmitter. The transmitter produces parity bit streams that correspond to an inputted information bit stream using first and second concatenated coders interconnected by an interleaver. Subsets of the first and second parity bit streams are selected in accordance with a puncturing pattern. A variety of novel puncturing patterns providing various coding rates for various constellations are disclosed. The transmitter combines the selected subsets of said first and second parity bit streams with said information bit stream. A QAM symbol stream is produced by mapping a first subset of the combined bit streams to an I dimension and mapping a second subset of the combined bit streams to a Q dimension. The QAM symbol stream is modulated to produce a modulated signal that is transmitted over a communication link. A complementary receiver is also disclosed. The puncturing pattern used in the transmitter may be adapted based on a performance metric determined in the receiver.



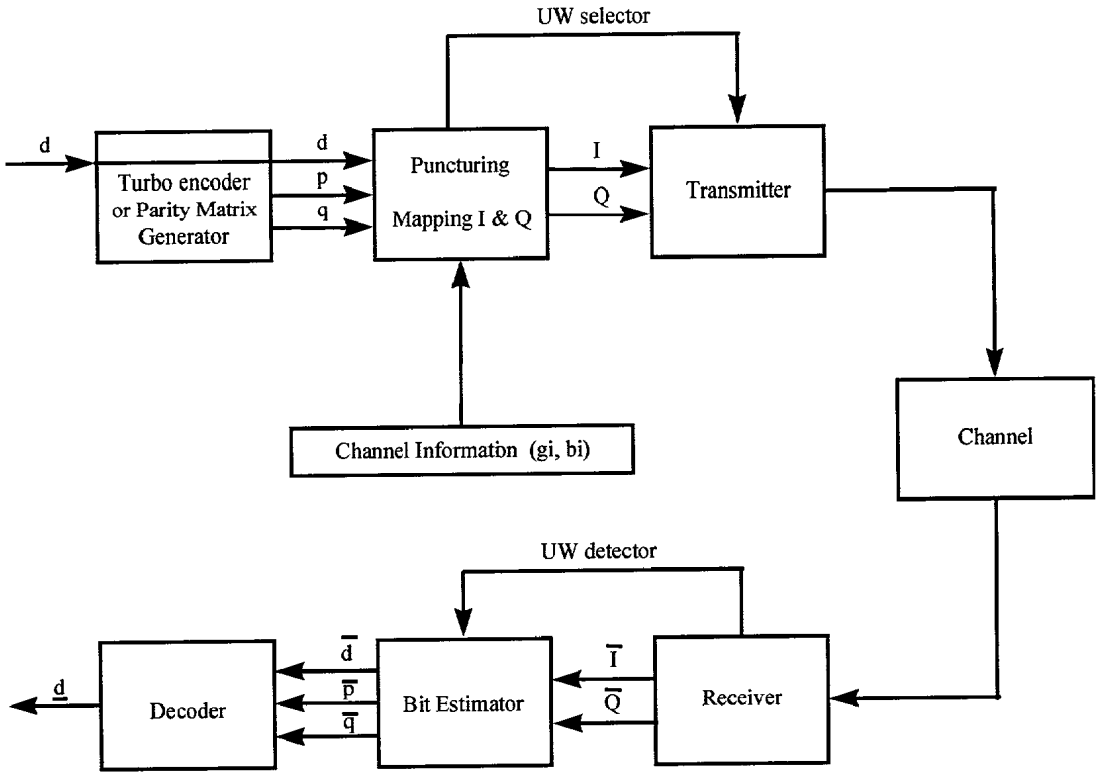

UW selector
d
Turbo encoder
or Parity Matrix
Generator
d
p
q
Puncturing
Mapping I & Q
I
Q
Transmitter
Channel
Channel Information (gi, bi)
UW detector
Decoder
d
Bit Estimator
Receiver

Figure 1

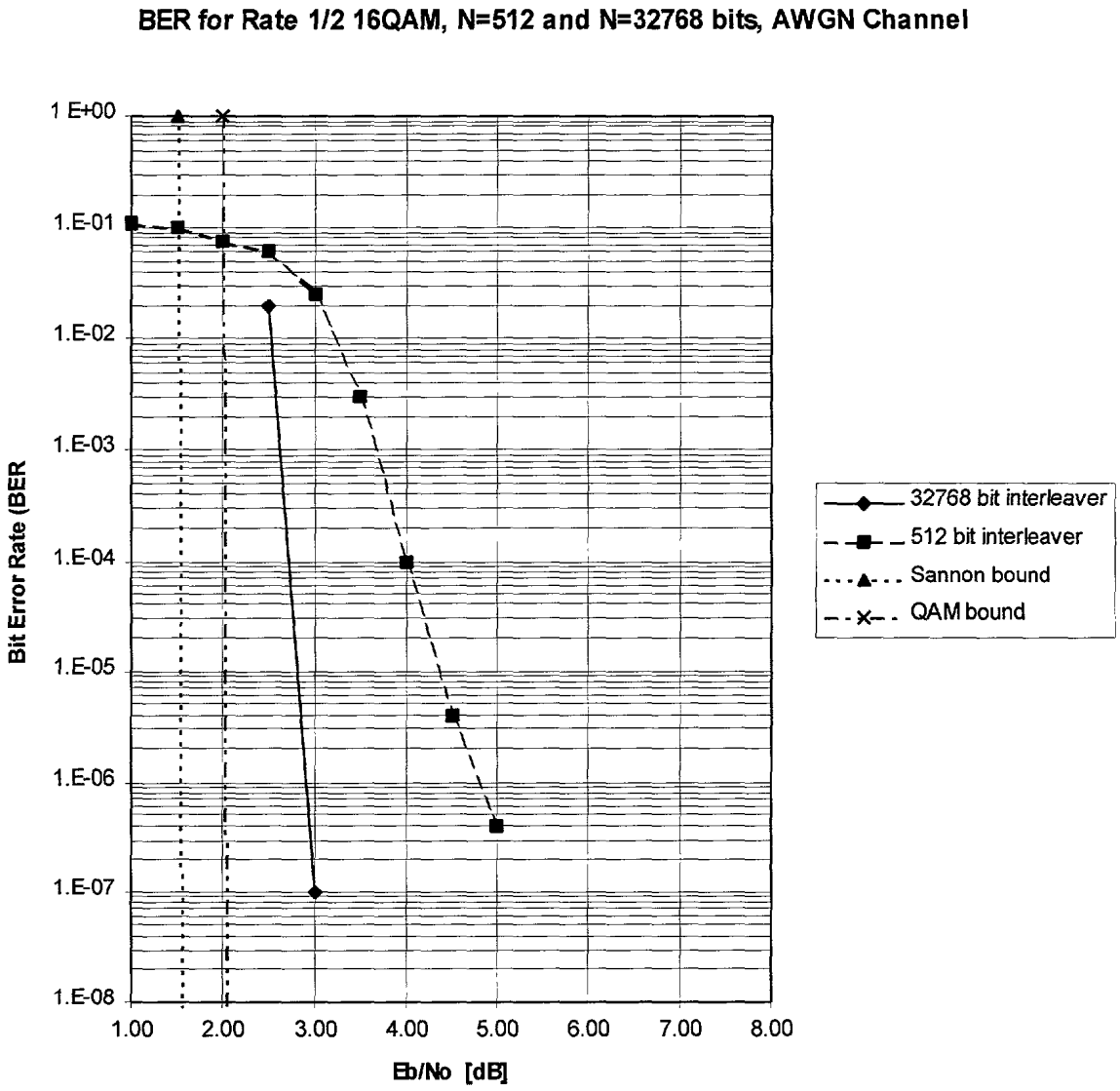

BER for Rate 1/2 16QAM, N=512 and N=32768 bits, AWGN Channel
1 E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
1.E-08
Bit Error Rate (BER
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
Eb/No [dB]
32768 bit interleaver
512 bit interleaver
Sannon bound
QAM bound

Figure 2

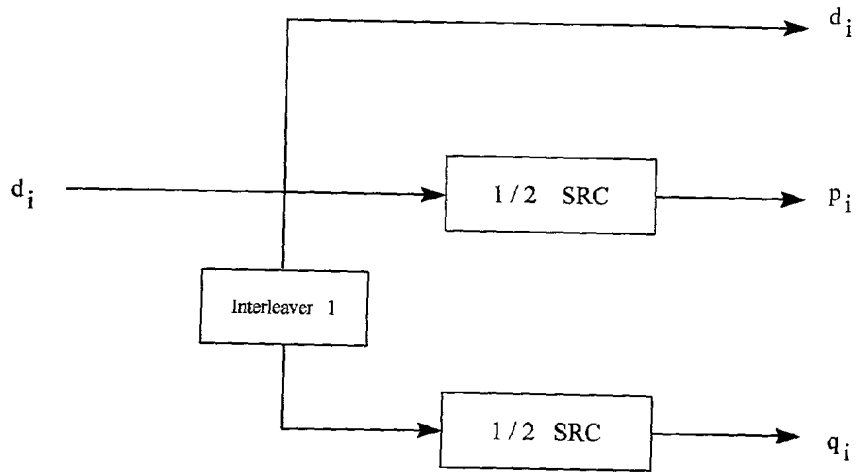


Figure 3

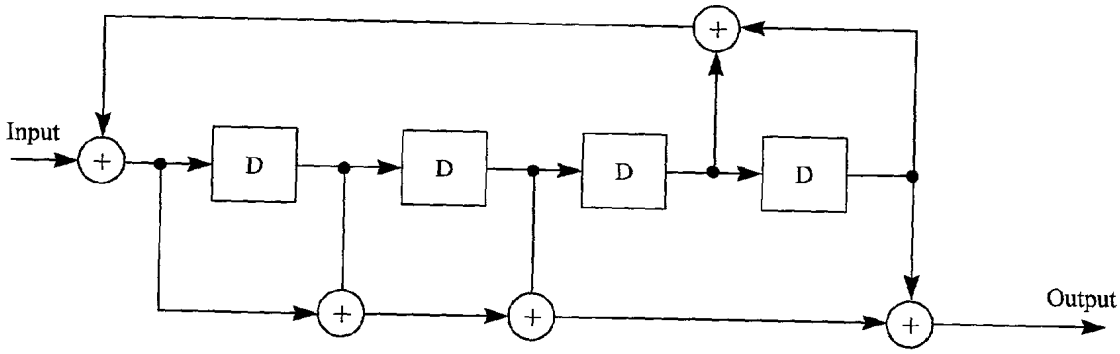


Figure 4

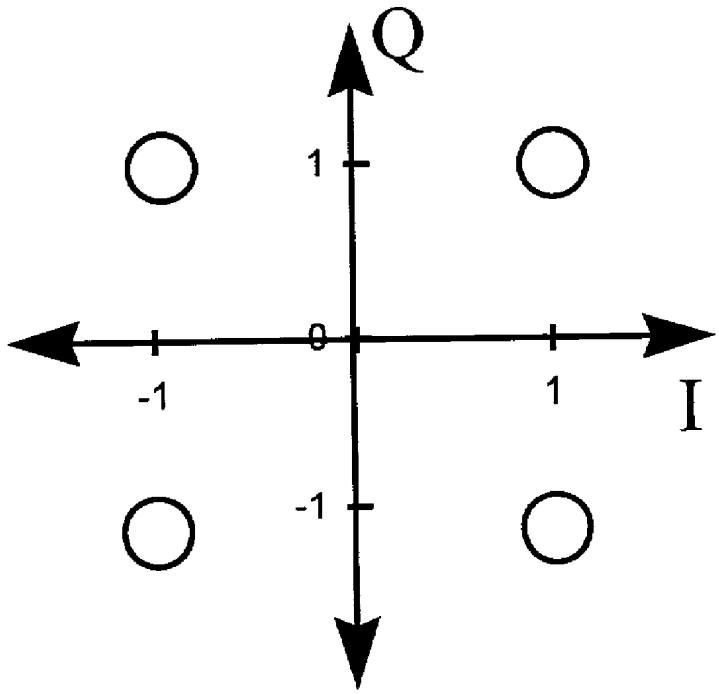


Figure 5

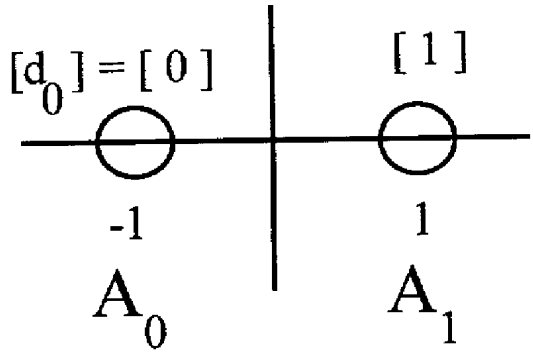


Figure 6

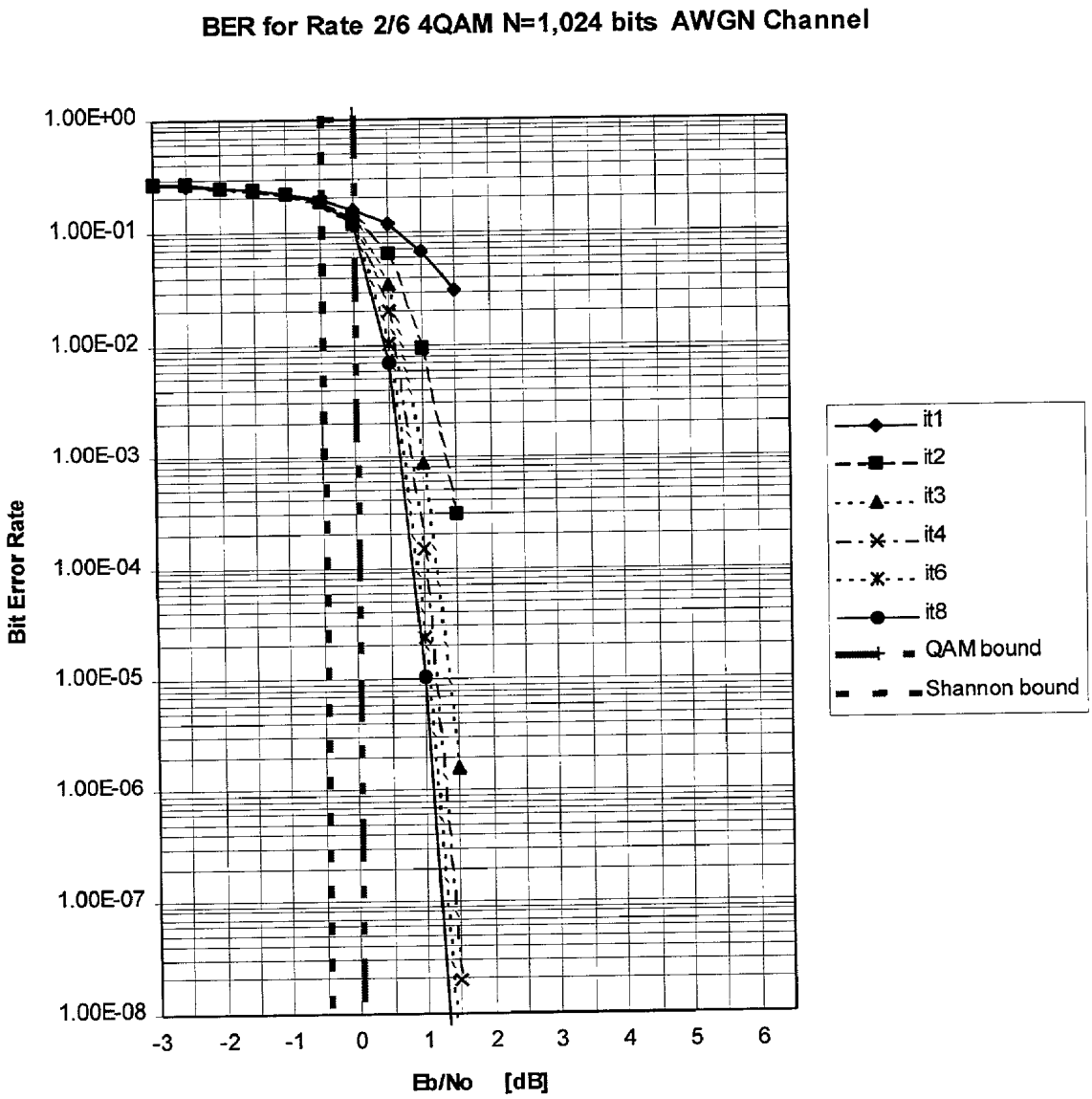

BER for Rate 2/6 4QAM N=1,024 bits AWGN Channel
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-3
-2
-1
0
1
2
3
4
5
6
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 7

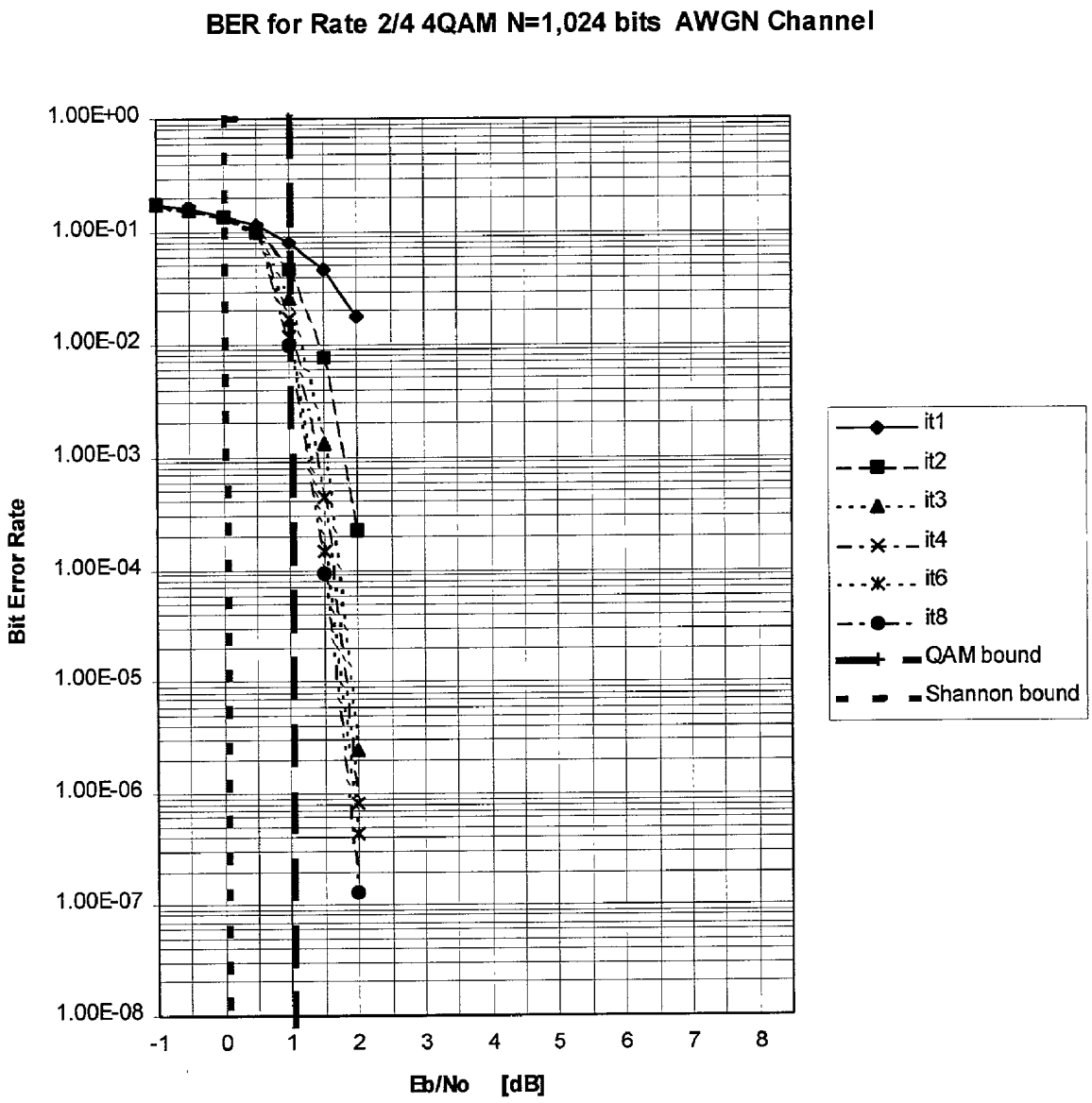

BER for Rate 2/4 4QAM N=1,024 bits AWGN Channel
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
0
1
2
3
4
5
6
7
8
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 8

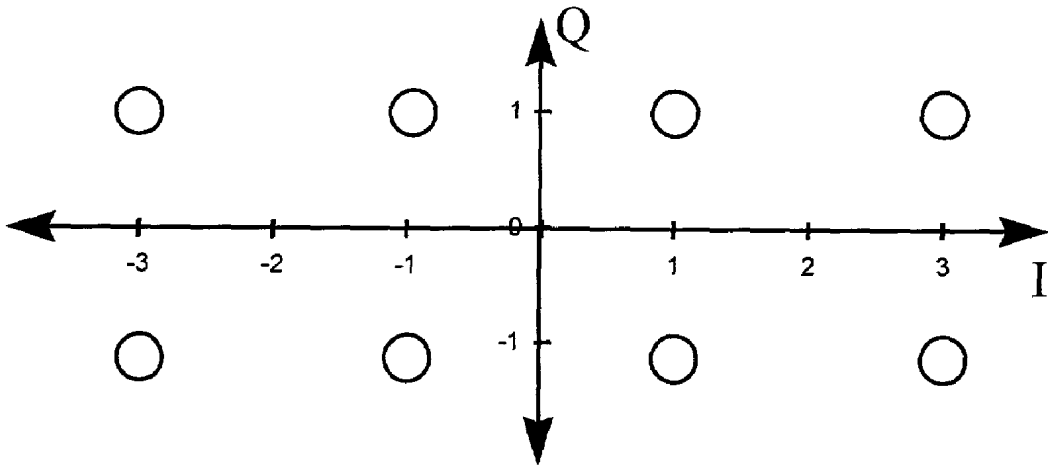
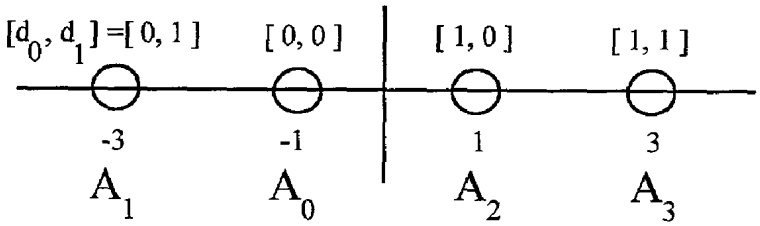


Figure 9



$E[|a\ 2\ |] = (\ 1+9+1+9)\ /\ 4 = 5$

Figure 10

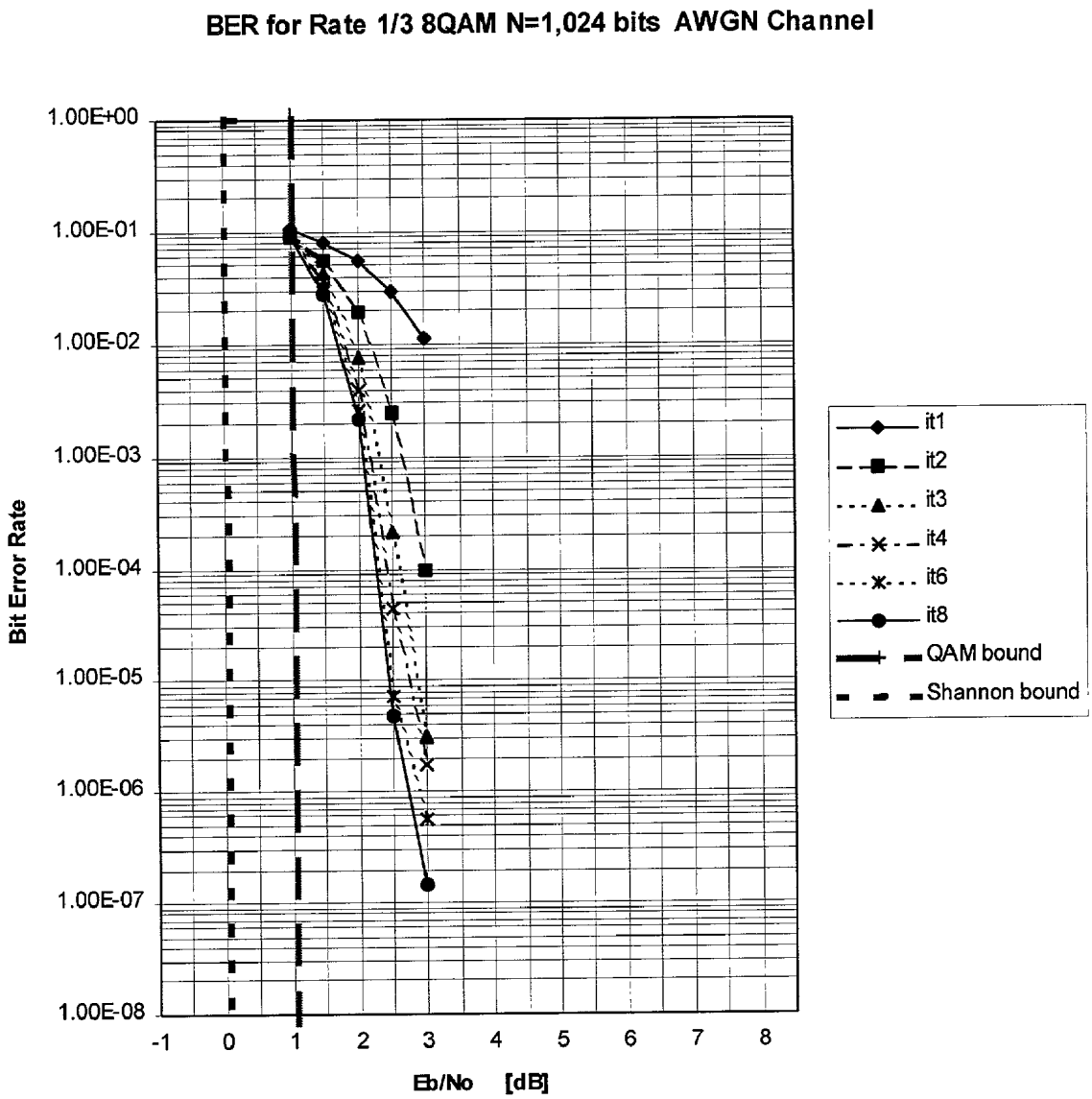

BER for Rate 1/3 8QAM N=1,024 bits AWGN Channel
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
0
1
2
3
4
5
6
7
8
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 11

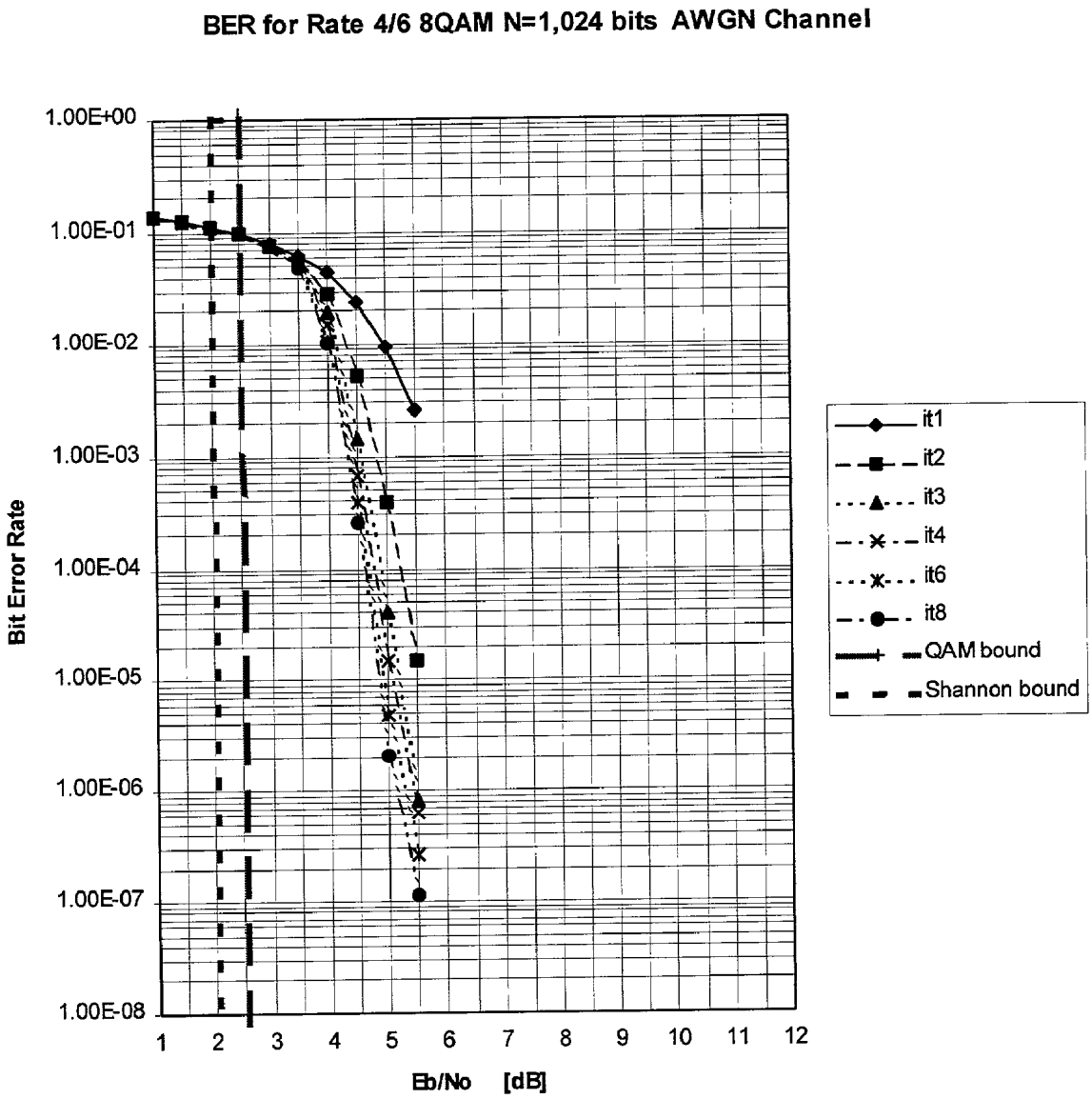

BER for Rate 4/6 8QAM N=1,024 bits AWGN Channel
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
1
2
3
4
5
6
7
8
9
10
11
12
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 12

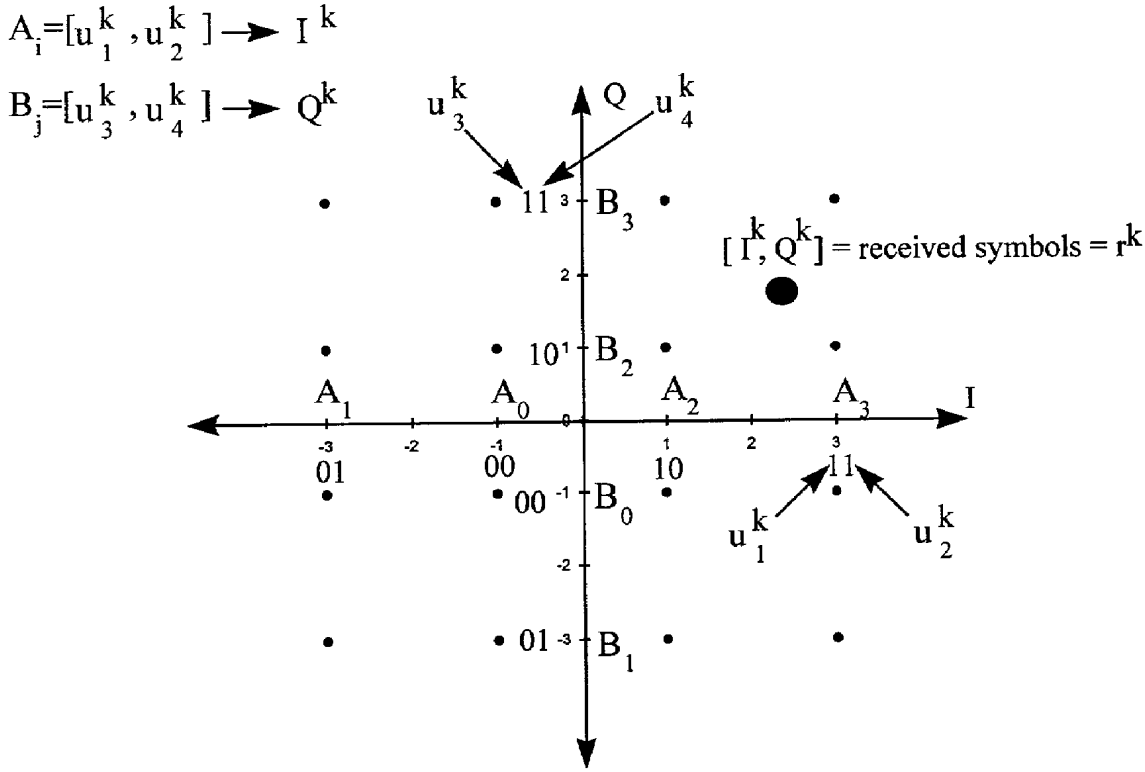

$A_i=[u_1^k, u_2^k] \rightarrow I^k$
$B_j=[u_3^k, u_4^k] \rightarrow Q^k$
u_3^k
u_4^k
Q
I
11
10
01
00
B_3
B_2
B_0
B_1
A_1
A_0
A_2
A_3
$[I^k, Q^k]$ = received symbols = r^k
u_1^k
u_2^k

Figure 13

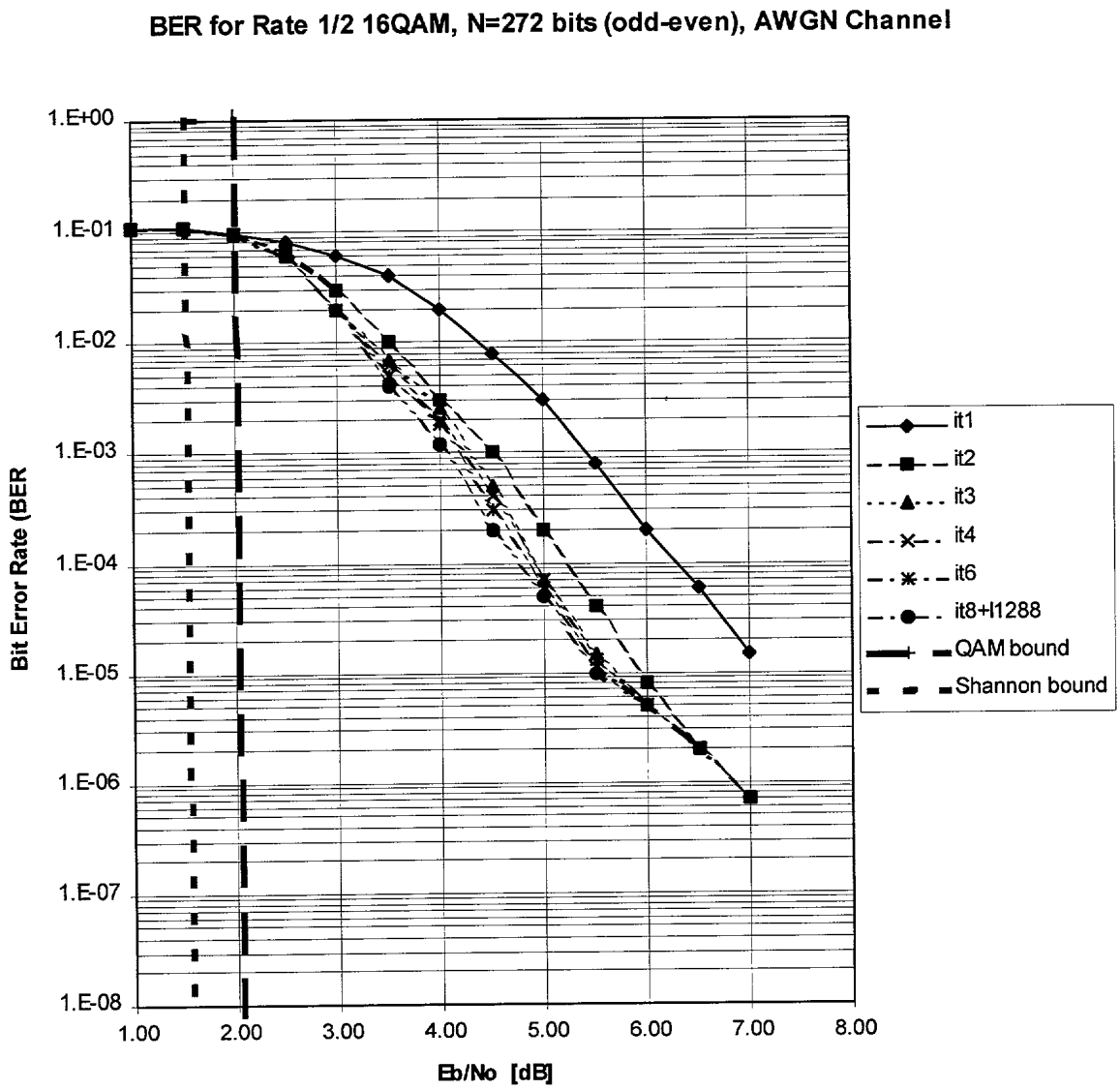

BER for Rate 1/2 16QAM, N=272 bits (odd-even), AWGN Channel
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
1.E-08
Bit Error Rate (BER
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
Eb/No [dB]
it1
it2
it3
it4
it6
it8+I1288
QAM bound
Shannon bound

Figure 14

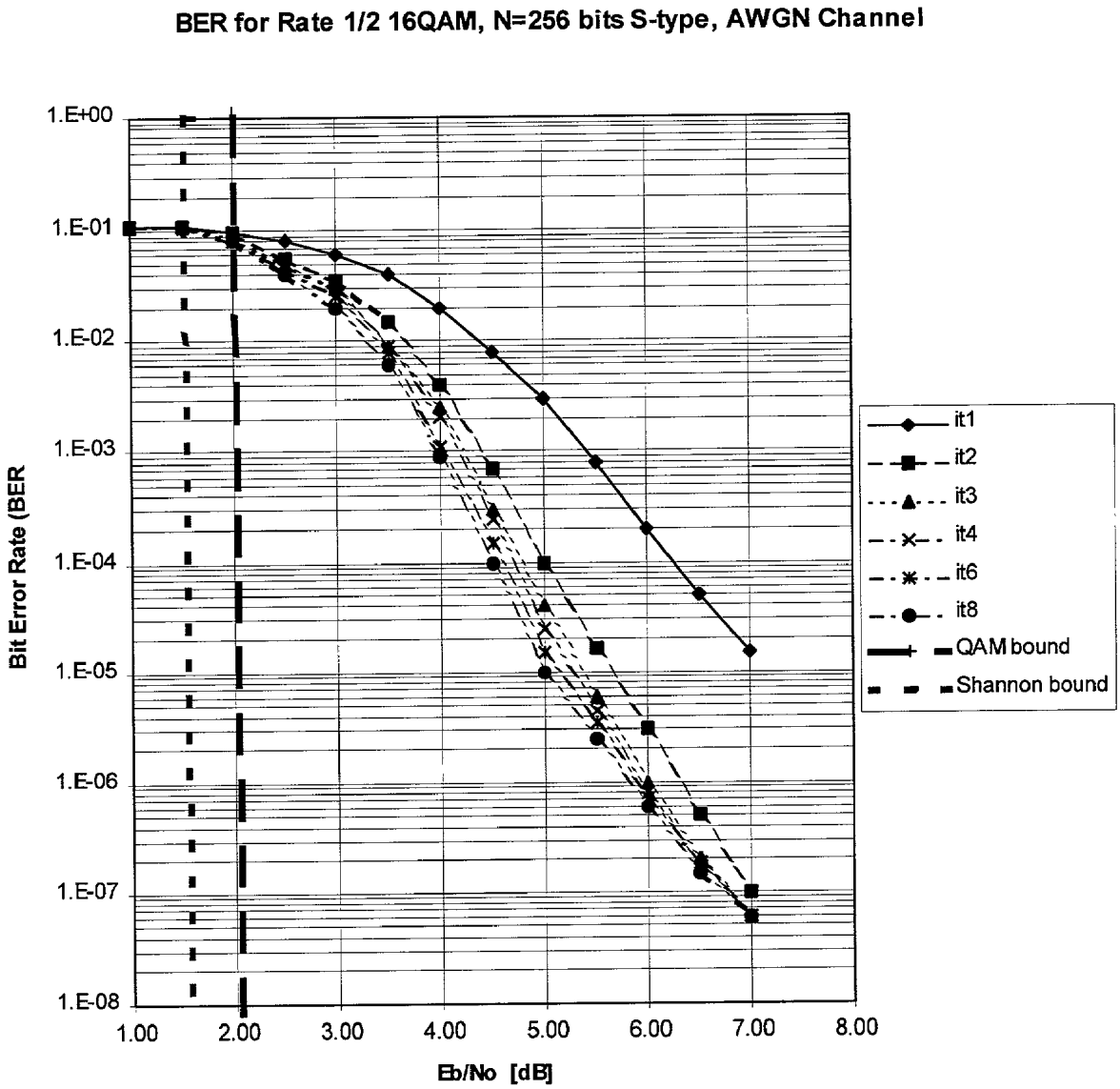

BER for Rate 1/2 16QAM, N=256 bits S-type, AWGN Channel
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
1.E-08
Bit Error Rate (BER
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 15

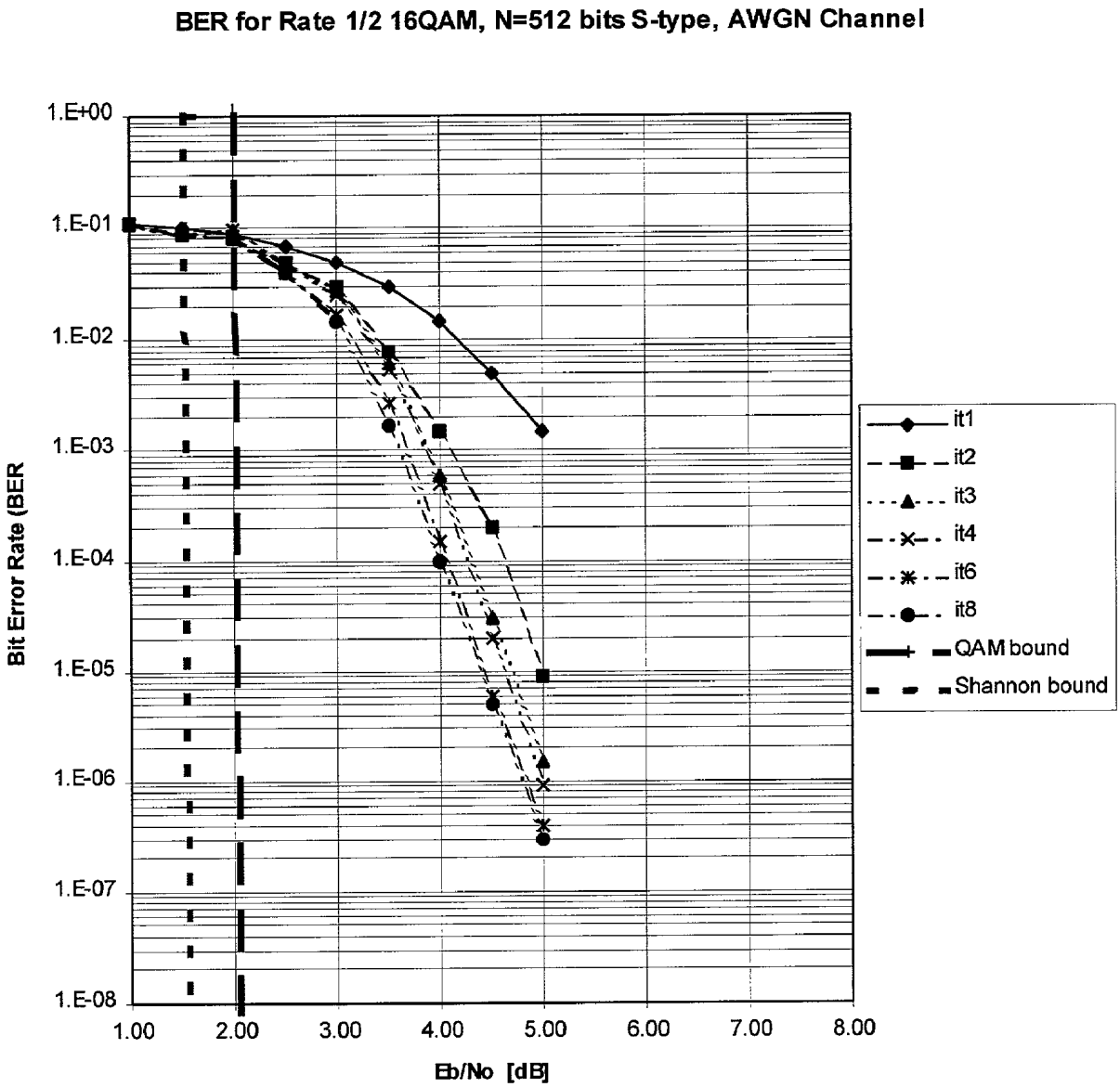

BER for Rate 1/2 16QAM, N=512 bits S-type, AWGN Channel
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
1.E-08
Bit Error Rate (BER
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 16

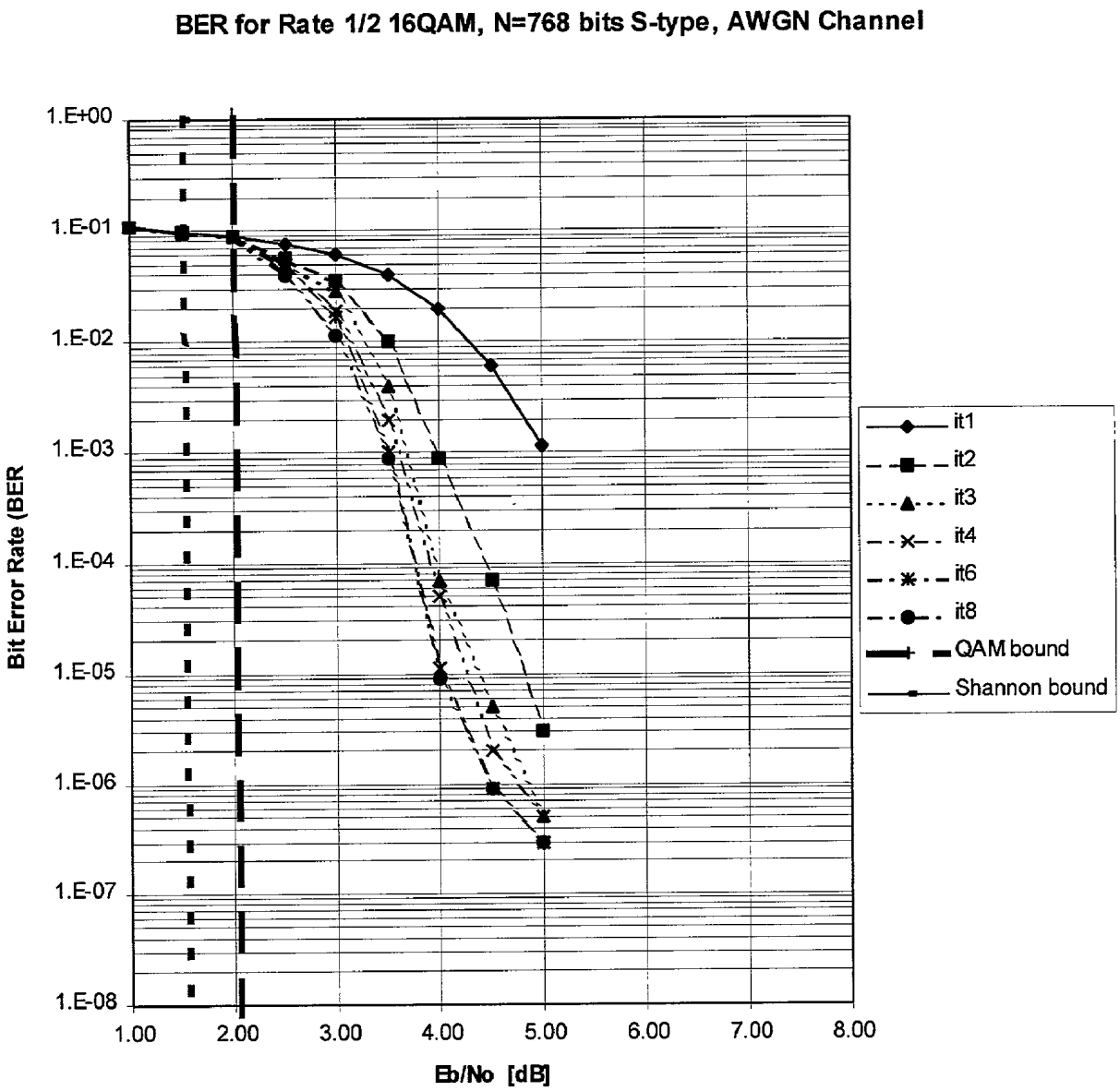

BER for Rate 1/2 16QAM, N=768 bits S-type, AWGN Channel
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
1.E-08
Bit Error Rate (BER
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 17

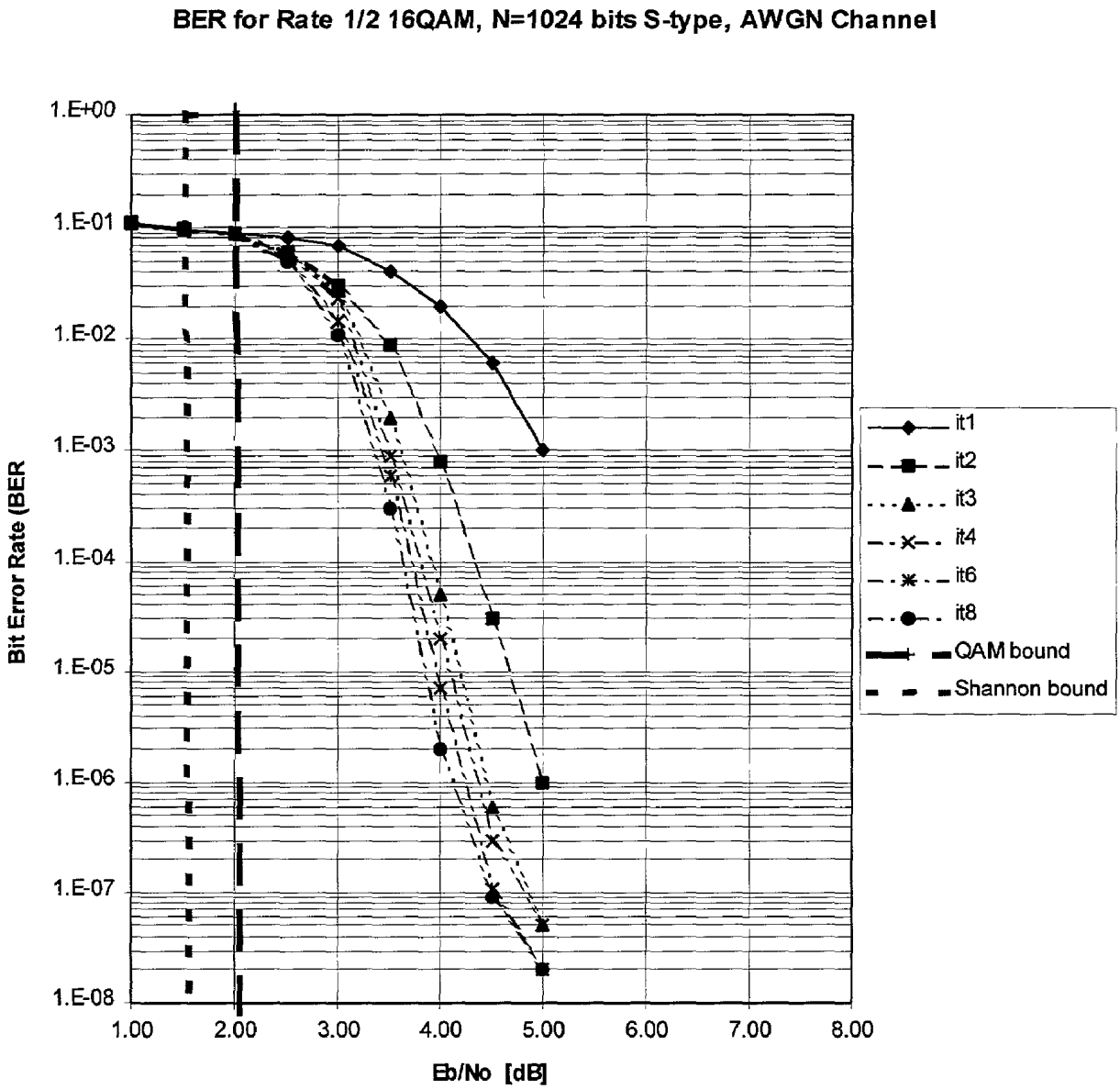

BER for Rate 1/2 16QAM, N=1024 bits S-type, AWGN Channel
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
1.E-08
Bit Error Rate (BER
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 18

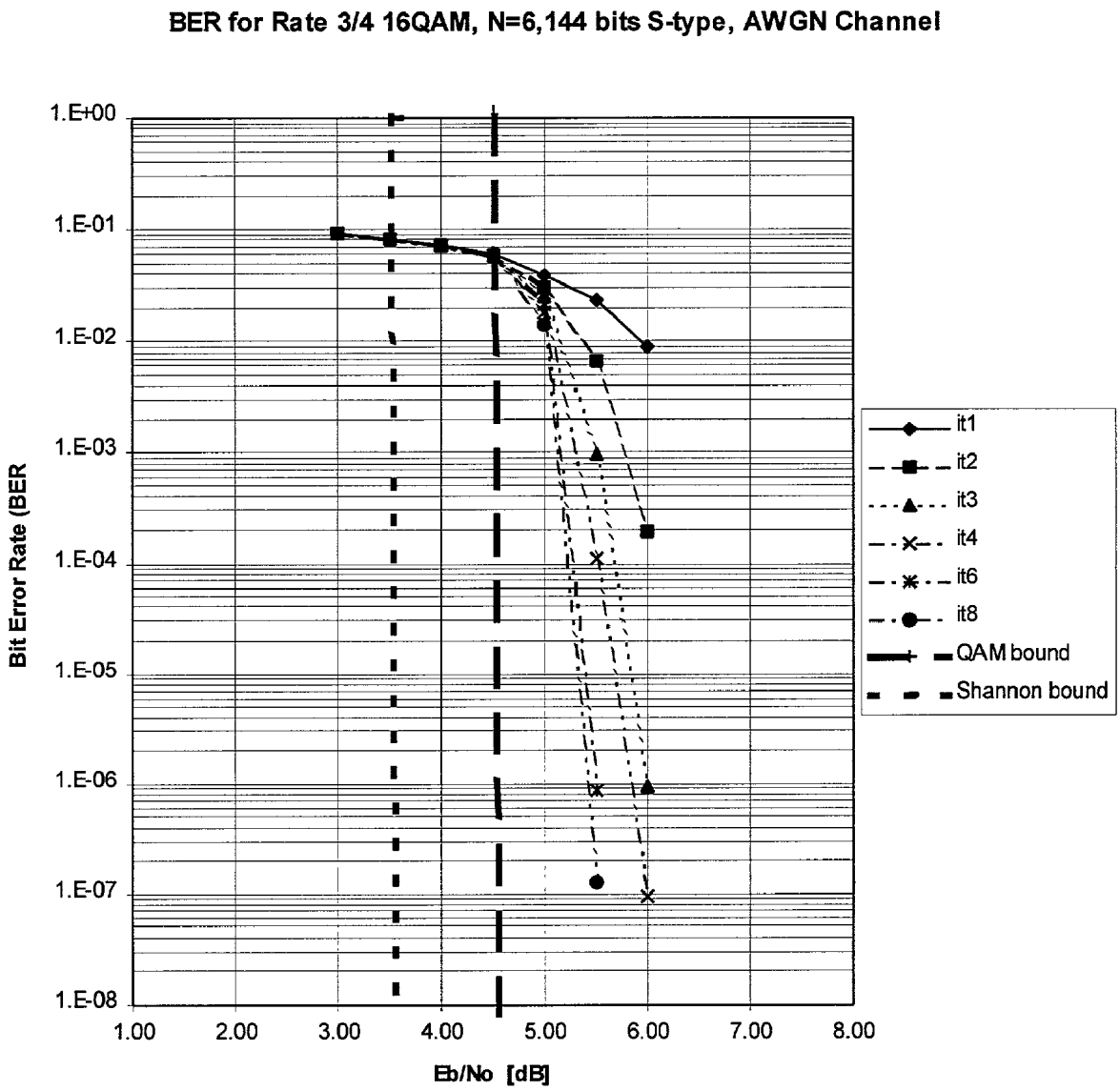

BER for Rate 3/4 16QAM, N=6,144 bits S-type, AWGN Channel
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
1.E-08
Bit Error Rate (BER
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 19

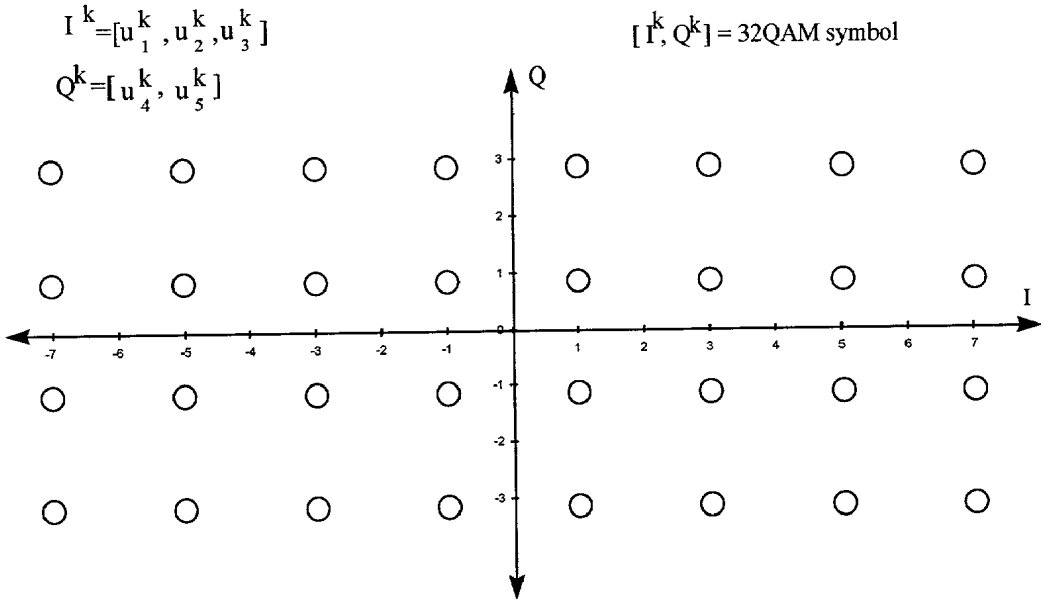
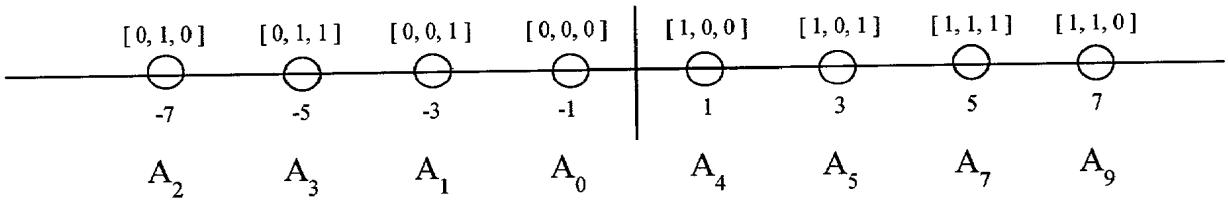


Figure 20



$$E[|a\ 2\ |] = (\ 1+9+25+49)\ /\ 4 = 21$$

Figure 21

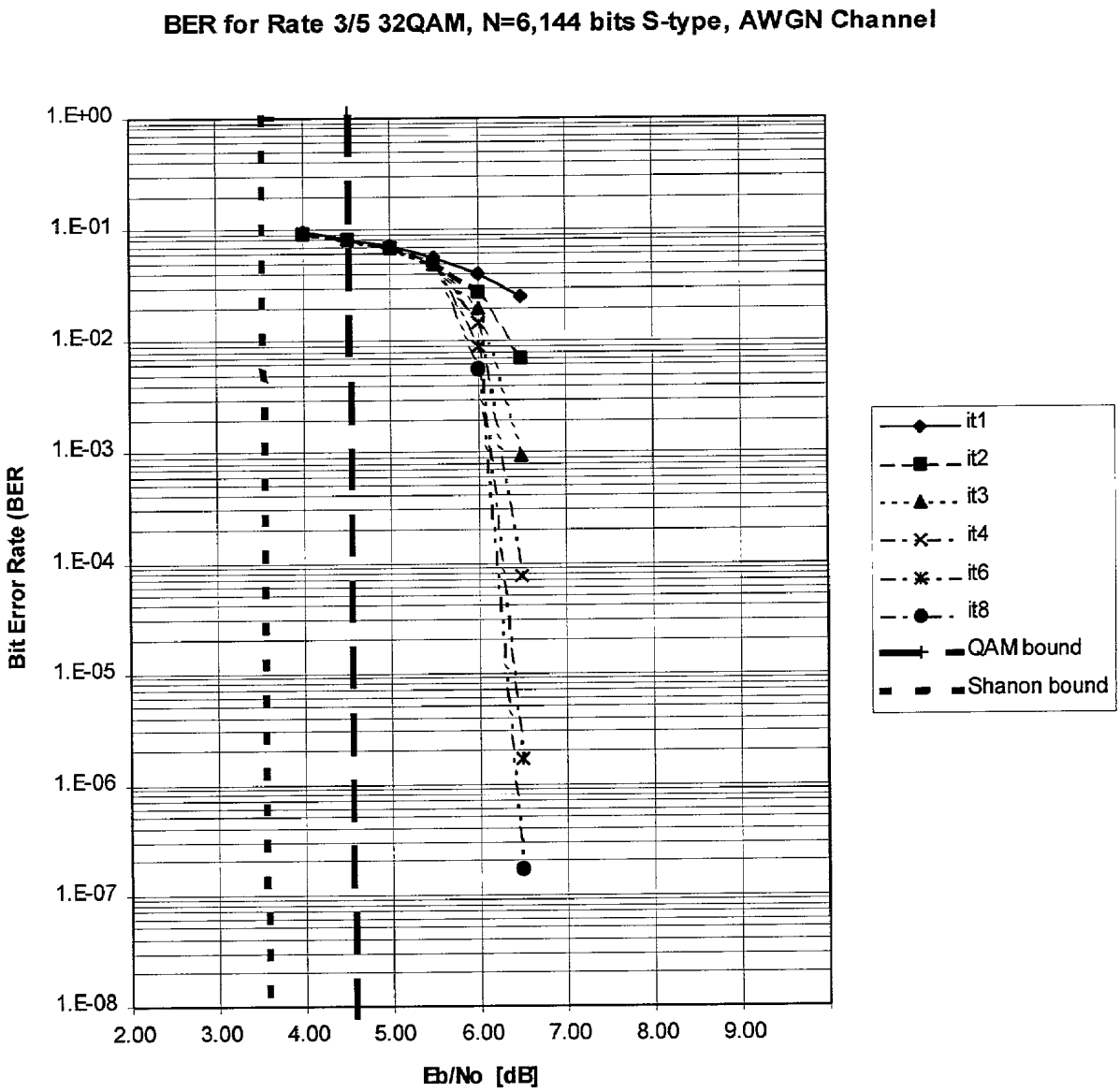

BER for Rate 3/5 32QAM, N=6,144 bits S-type, AWGN Channel
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
1.E-08
Bit Error Rate (BER
2.00
3.00
4.00
5.00
6.00
7.00
8.00
9.00
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shanon bound

Figure 22

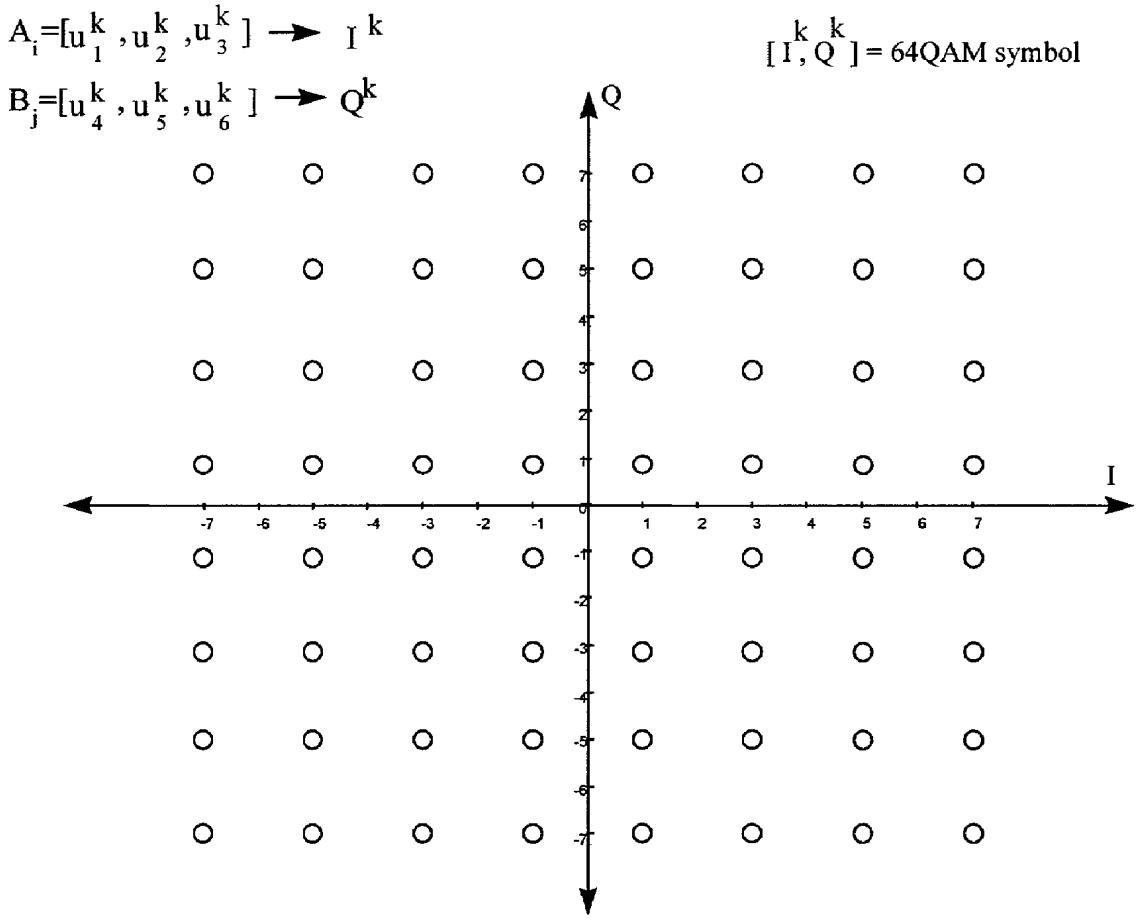

$A_i=[u_1^k, u_2^k, u_3^k] \rightarrow I^k$
$B_j=[u_4^k, u_5^k, u_6^k] \rightarrow Q^k$
$[I^k, Q^k]$ = 64QAM symbol
Q
I
-7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7

Figure 23

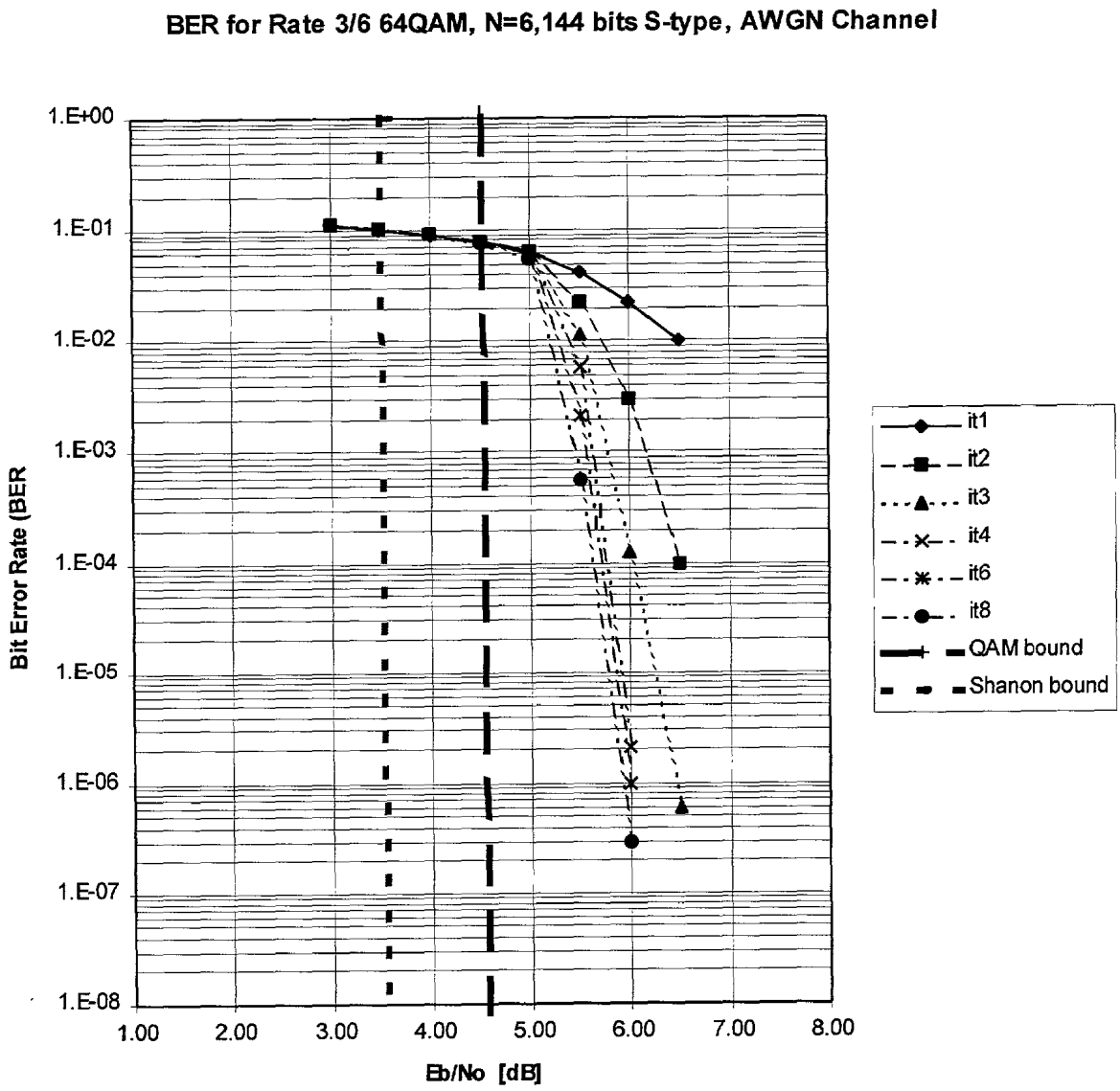

BER for Rate 3/6 64QAM, N=6,144 bits S-type, AWGN Channel
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
1.E-08
Bit Error Rate (BER
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shanon bound

Figure 24

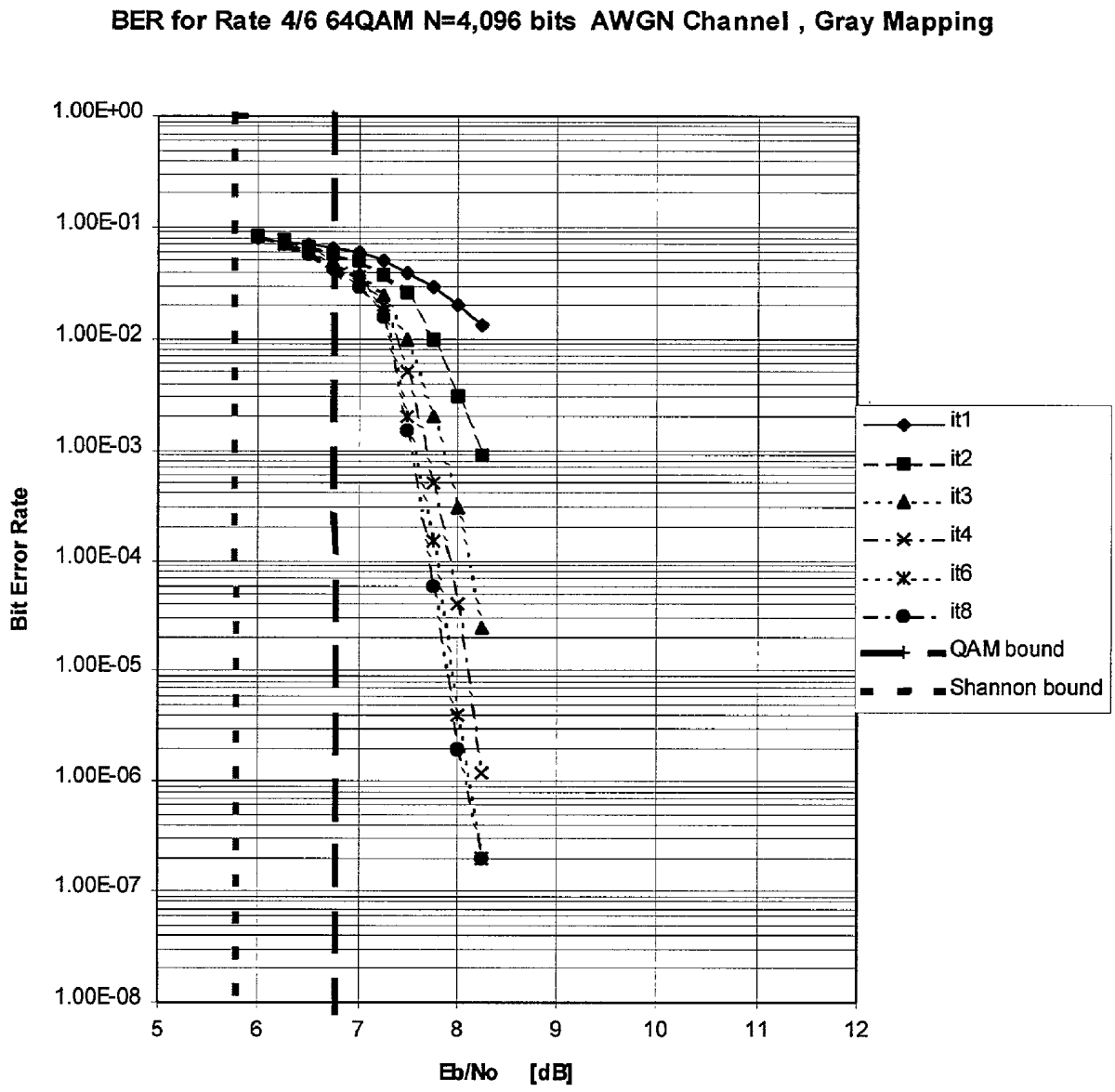
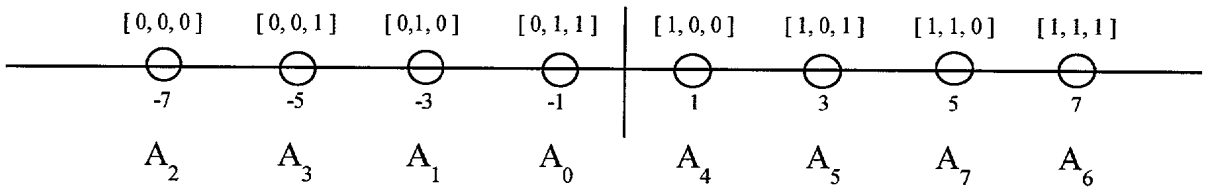

BER for Rate 4/6 64QAM N=4,096 bits AWGN Channel , Gray Mapping
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
5
6
7
8
9
10
11
12
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 25



E[|a 2 |] = (1 + 9 + 25 + 49) / 4 = 21

Figure 26

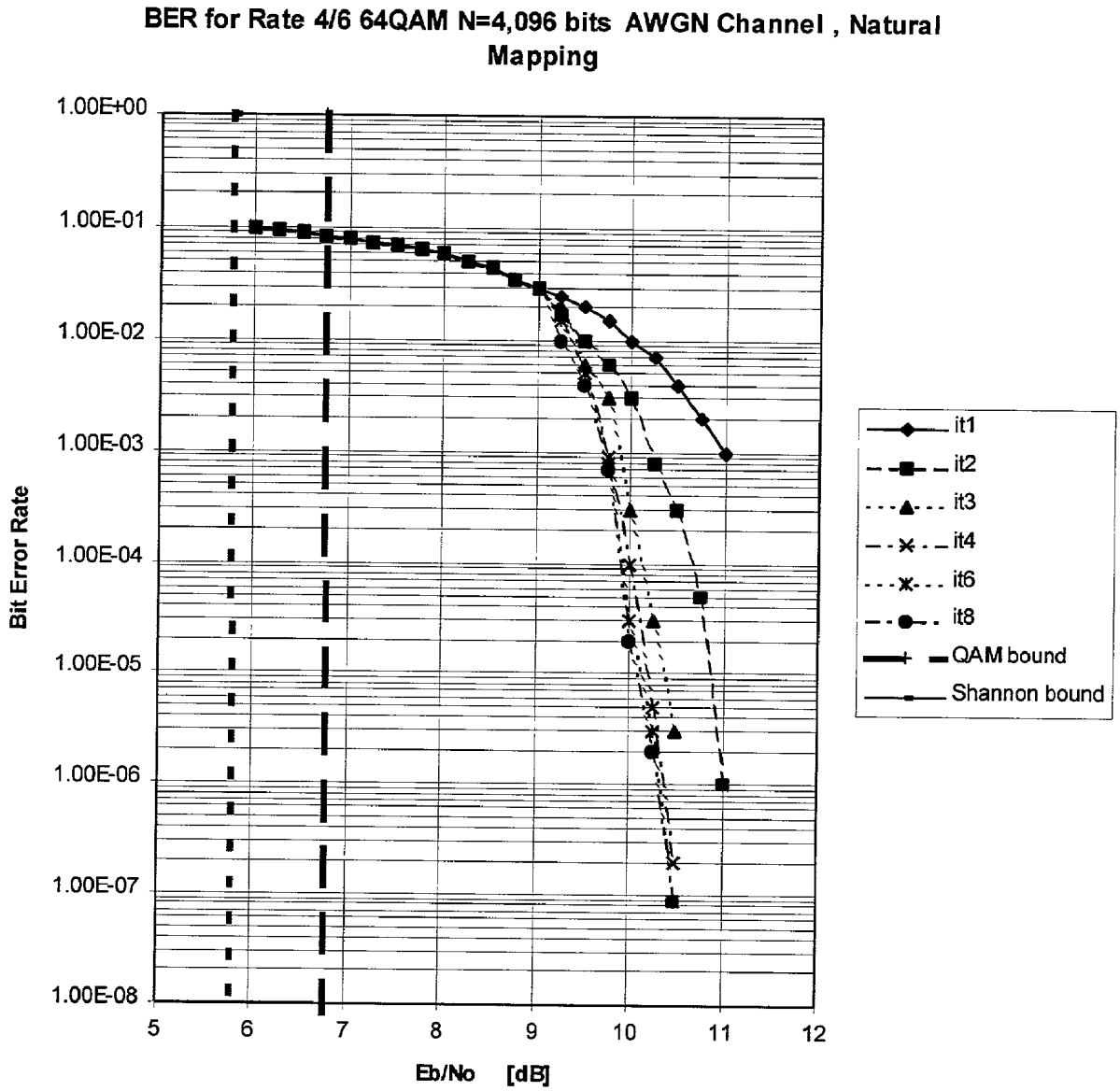

BER for Rate 4/6 64QAM N=4,096 bits AWGN Channel , Natural Mapping
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
5
6
7
8
9
10
11
12
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 27

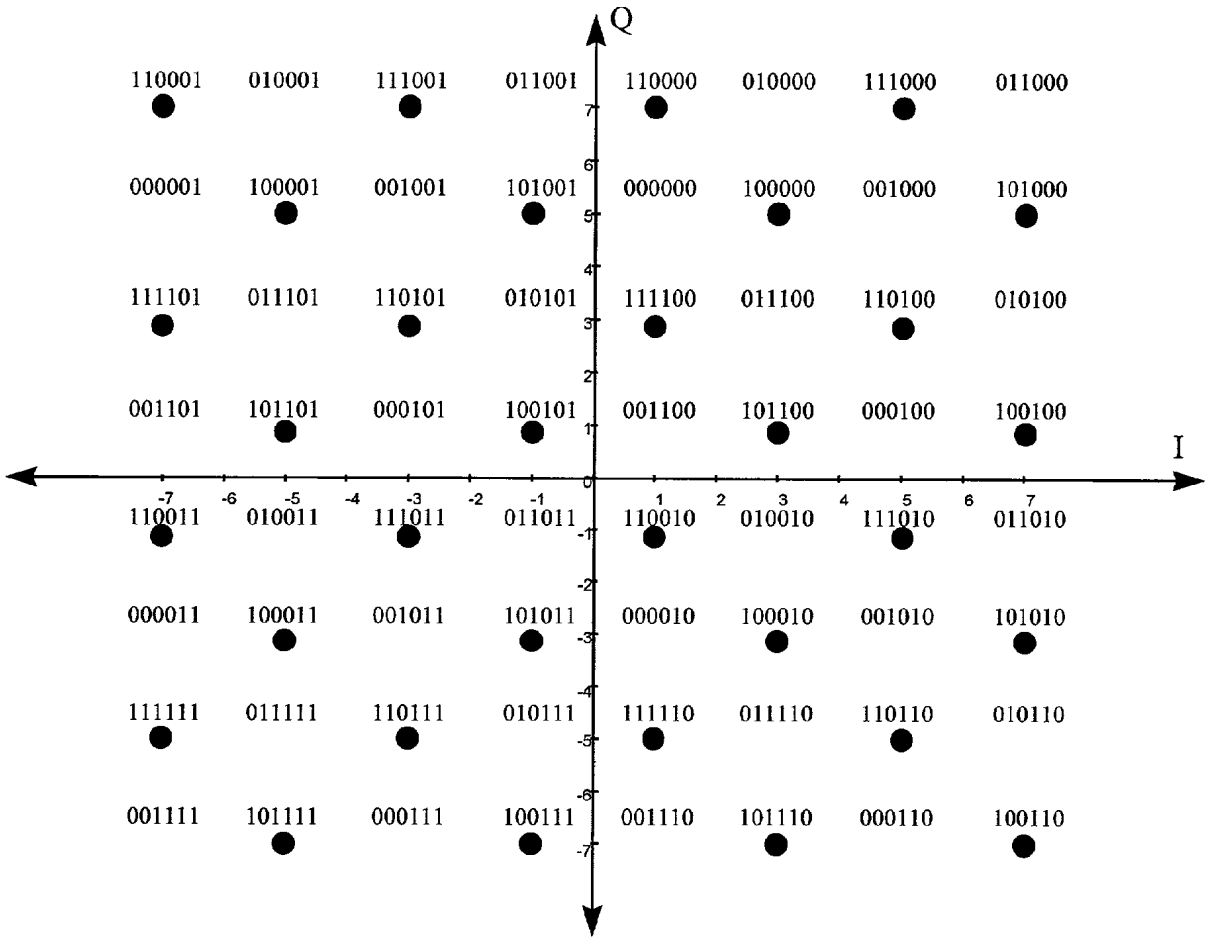

Q
I
110001
010001
111001
011001
110000
010000
111000
011000
000001
100001
001001
101001
000000
100000
001000
101000
111101
011101
110101
010101
111100
011100
110100
010100
001101
101101
000101
100101
001100
101100
000100
100100
110011
010011
111011
011011
110010
010010
111010
011010
000011
100011
001011
101011
000010
100010
001010
101010
111111
011111
110111
010111
111110
011110
110110
010110
001111
101111
000111
100111
001110
101110
000110
100110
-7
-6
-5
-4
-3
-2
-1
0
1
2
3
4
5
6
7

Figure 28

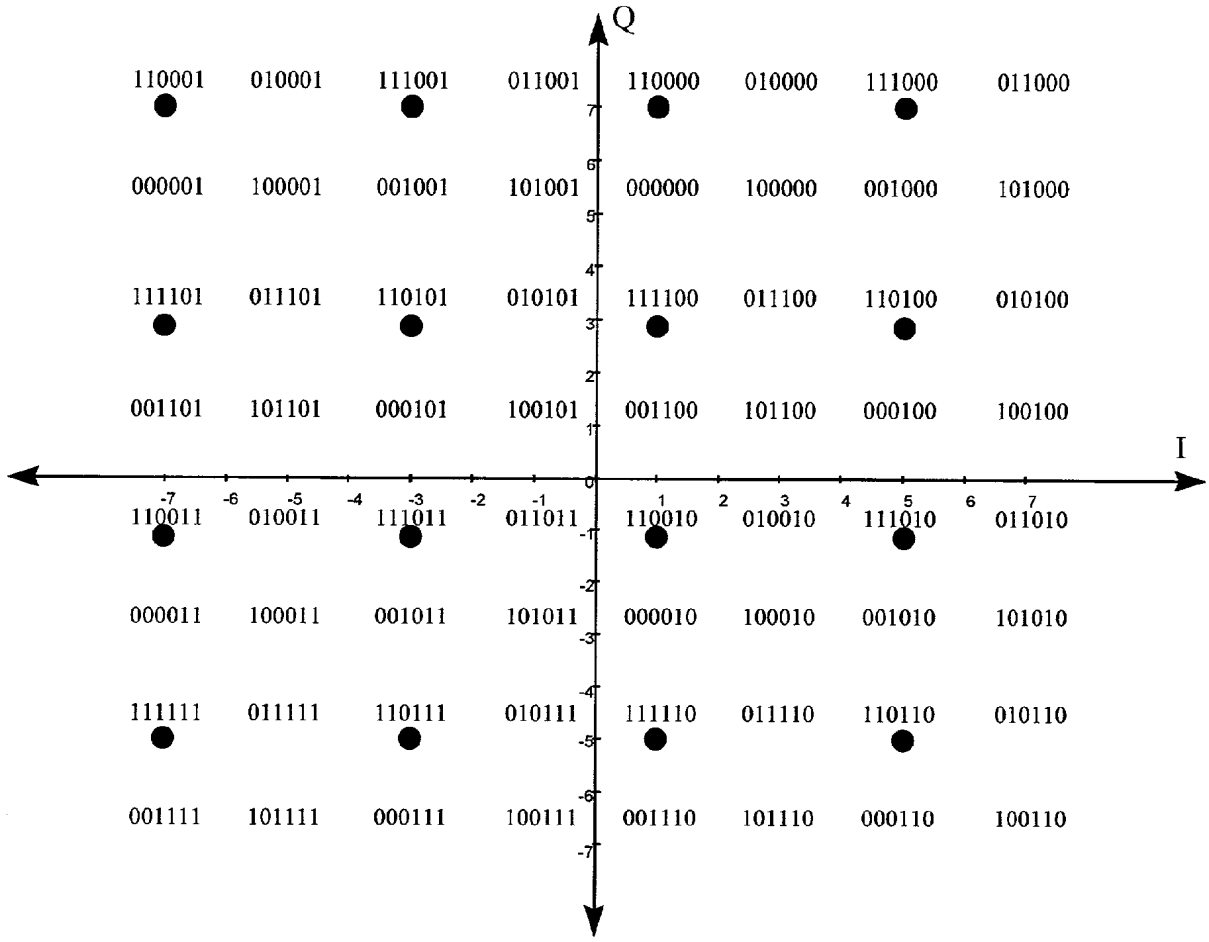

Q
I
110001 010001 111001 011001 110000 010000 111000 011000
000001 100001 001001 101001 000000 100000 001000 101000
111101 011101 110101 010101 111100 011100 110100 010100
001101 101101 000101 100101 001100 101100 000100 100100
110011 010011 111011 011011 110010 010010 111010 011010
000011 100011 001011 101011 000010 100010 001010 101010
111111 011111 110111 010111 111110 011110 110110 010110
001111 101111 000111 100111 001110 101110 000110 100110
-7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7
7 6 5 4 3 2 1 -1 -2 -3 -4 -5 -6 -7

Figure 29

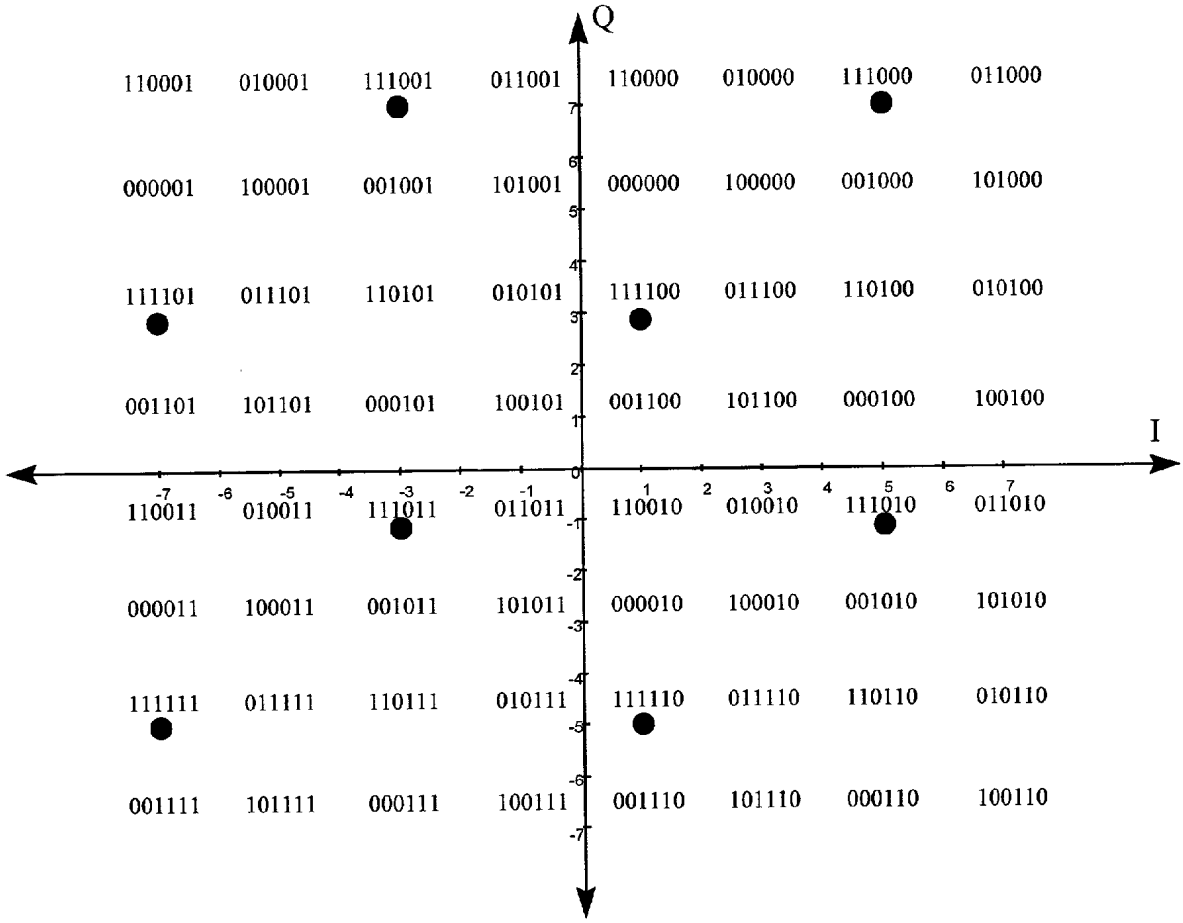

Q
I
110001 010001 111001 011001 110000 010000 111000 011000
000001 100001 001001 101001 000000 100000 001000 101000
111101 011101 110101 010101 111100 011100 110100 010100
001101 101101 000101 100101 001100 101100 000100 100100
110011 010011 111011 011011 110010 010010 111010 011010
000011 100011 001011 101011 000010 100010 001010 101010
111111 011111 110111 010111 111110 011110 110110 010110
001111 101111 000111 100111 001110 101110 000110 100110
-7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7

Figure 30

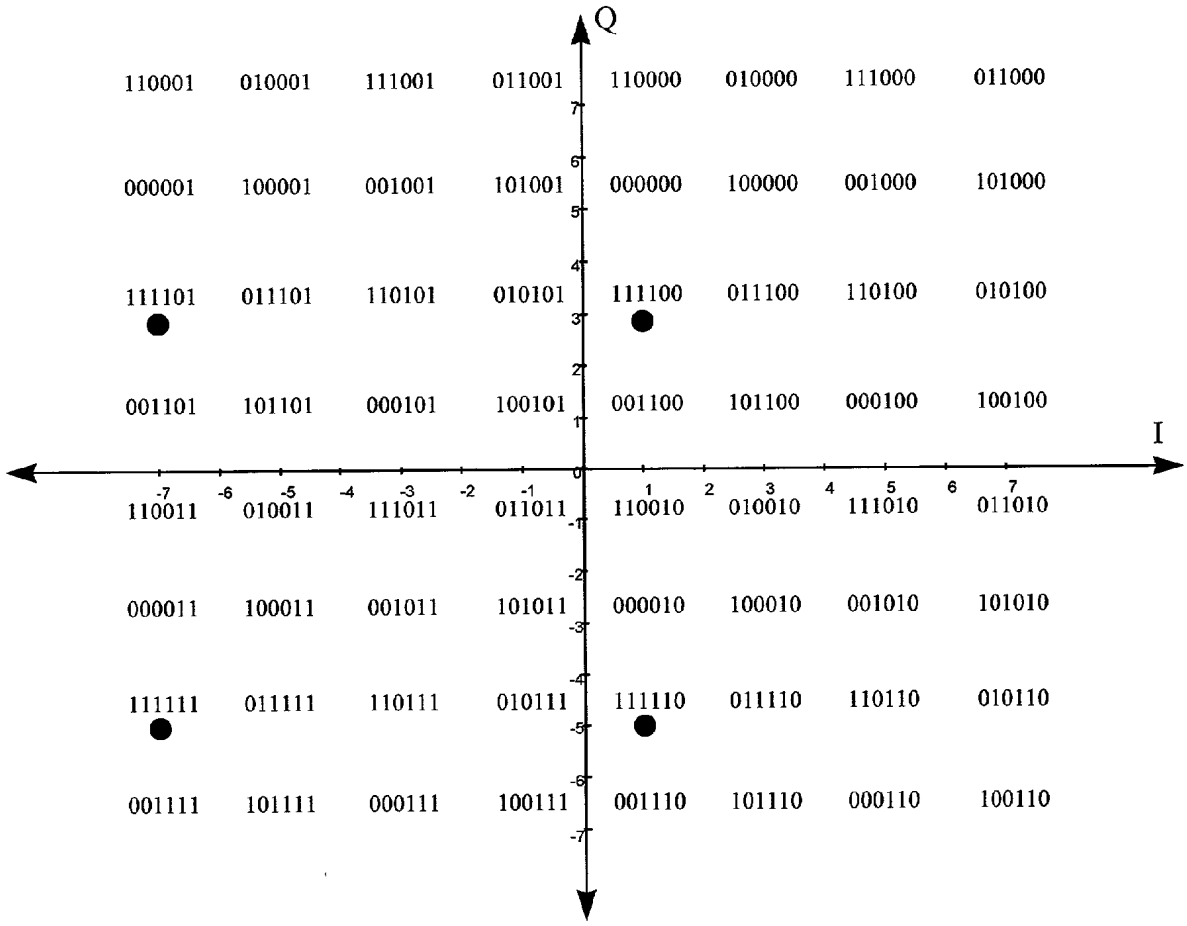

Q
I
110001 010001 111001 011001 110000 010000 111000 011000
000001 100001 001001 101001 000000 100000 001000 101000
111101 011101 110101 010101 111100 011100 110100 010100
001101 101101 000101 100101 001100 101100 000100 100100
110011 010011 111011 011011 110010 010010 111010 011010
000011 100011 001011 101011 000010 100010 001010 101010
111111 011111 110111 010111 111110 011110 110110 010110
001111 101111 000111 100111 001110 101110 000110 100110
7 6 5 4 3 2 1 0 -1 -2 -3 -4 -5 -6 -7
-7 -6 -5 -4 -3 -2 -1 1 2 3 4 5 6 7

Figure 31

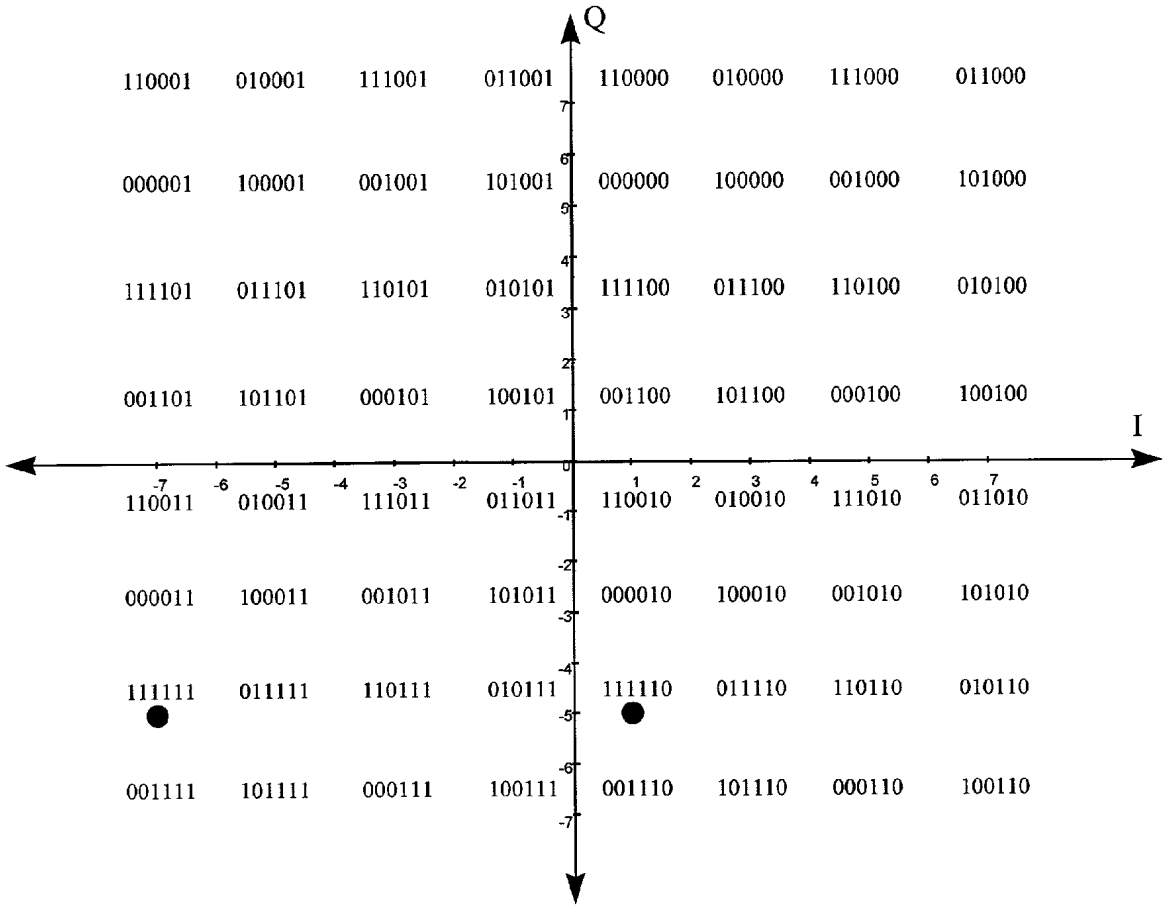

Q
I
110001 010001 111001 011001 110000 010000 111000 011000
000001 100001 001001 101001 000000 100000 001000 101000
111101 011101 110101 010101 111100 011100 110100 010100
001101 101101 000101 100101 001100 101100 000100 100100
110011 010011 111011 011011 110010 010010 111010 011010
000011 100011 001011 101011 000010 100010 001010 101010
111111 011111 110111 010111 111110 011110 110110 010110
001111 101111 000111 100111 001110 101110 000110 100110
7 6 5 4 3 2 1 0 -1 -2 -3 -4 -5 -6 -7
-7 -6 -5 -4 -3 -2 -1 1 2 3 4 5 6 7

Figure 32

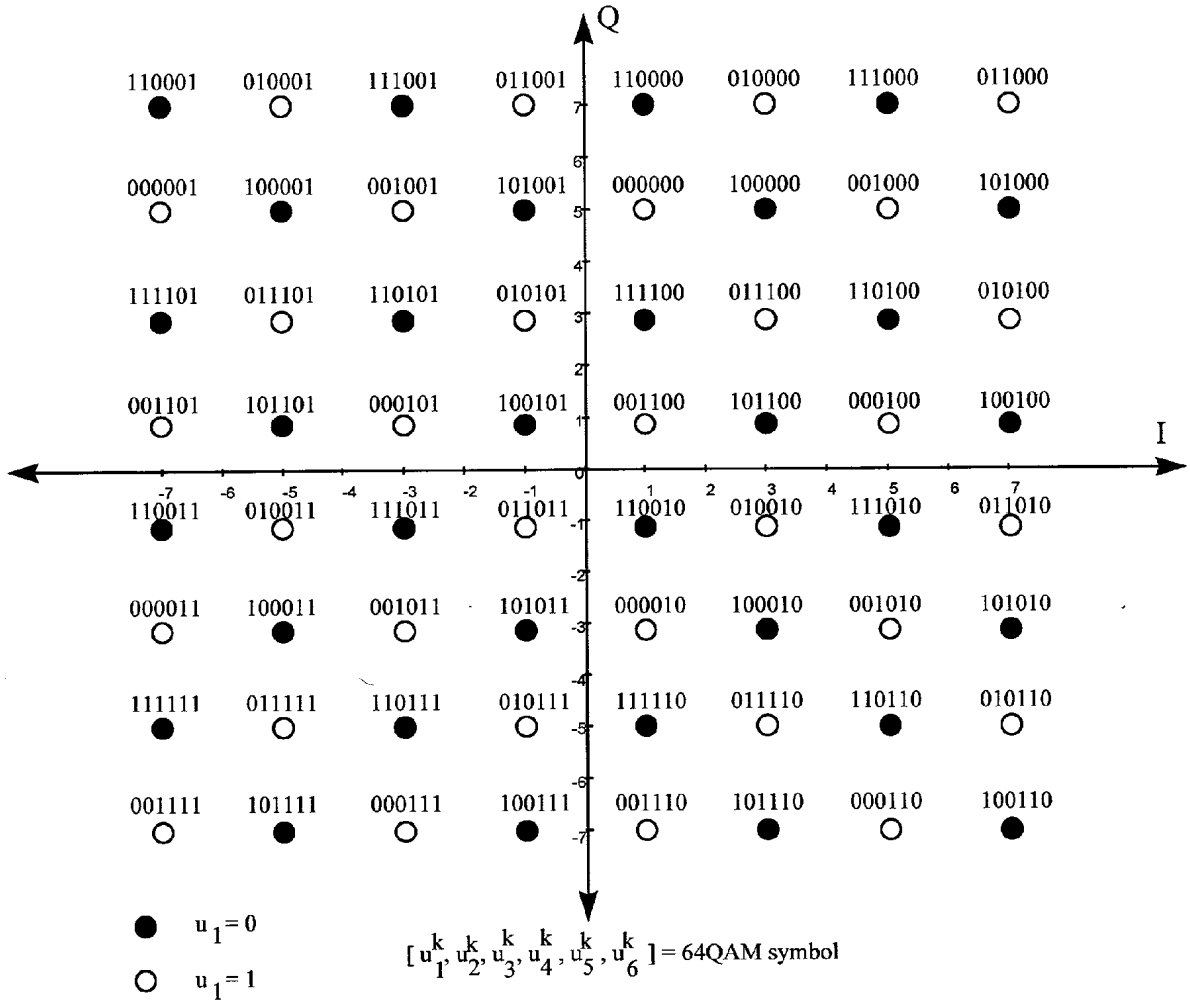

Q
I
110001 010001 111001 011001 110000 010000 111000 011000
000001 100001 001001 101001 000000 100000 001000 101000
111101 011101 110101 010101 111100 011100 110100 010100
001101 101101 000101 100101 001100 101100 000100 100100
110011 010011 111011 011011 110010 010010 111010 011010
000011 100011 001011 101011 000010 100010 001010 101010
111111 011111 110111 010111 111110 011110 110110 010110
001111 101111 000111 100111 001110 101110 000110 100110
-7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7
u1 = 0
u1 = 1
[u1k, u2k, u3k, u4k, u5k, u6k] = 64QAM symbol

Figure 33

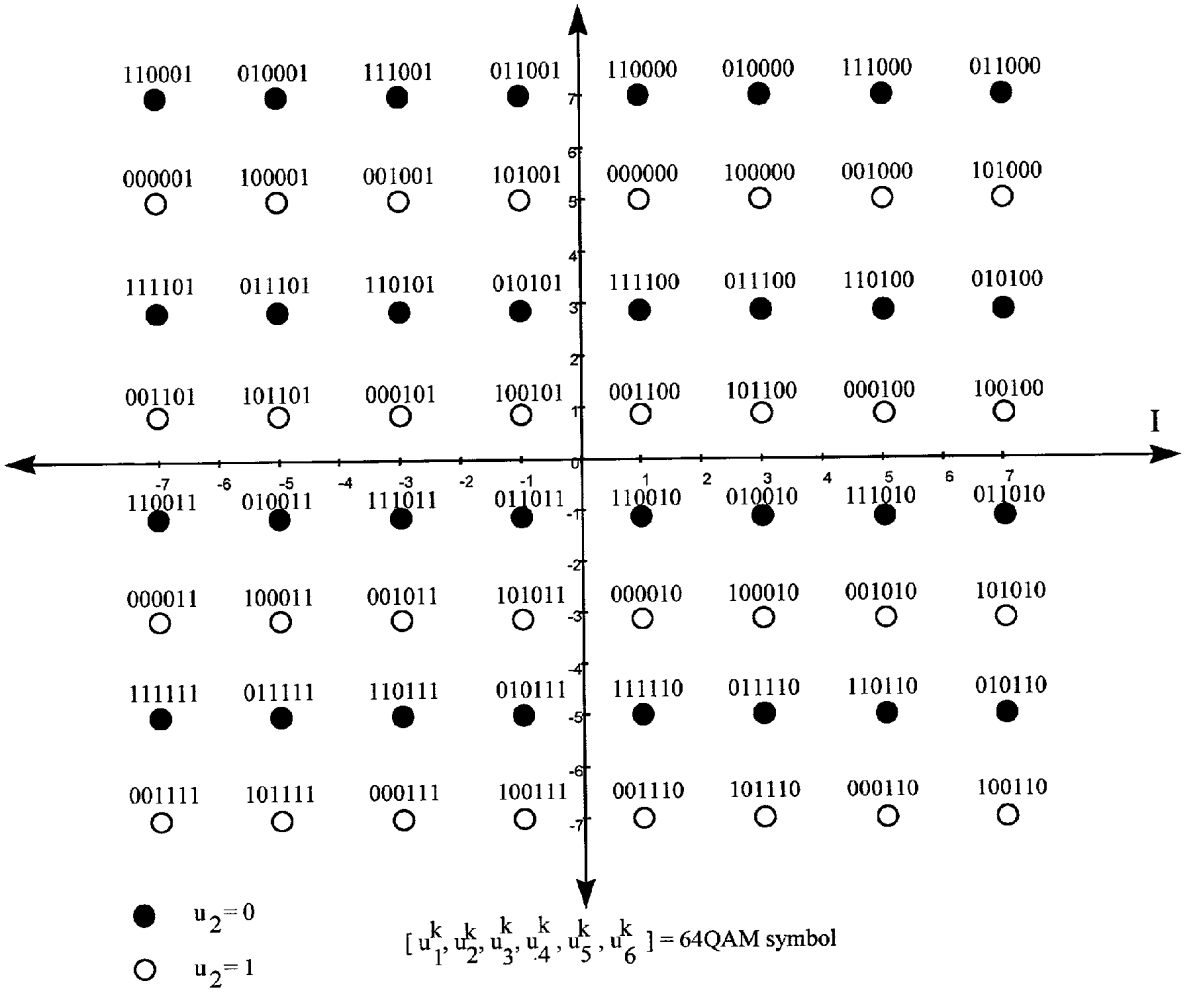

110001 010001 111001 011001 110000 010000 111000 011000
000001 100001 001001 101001 000000 100000 001000 101000
111101 011101 110101 010101 111100 011100 110100 010100
001101 101101 000101 100101 001100 101100 000100 100100
110011 010011 111011 011011 110010 010010 111010 011010
000011 100011 001011 101011 000010 100010 001010 101010
111111 011111 110111 010111 111110 011110 110110 010110
001111 101111 000111 100111 001110 101110 000110 100110
I
-7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7
7 6 5 4 3 2 1 -1 -2 -3 -4 -5 -6 -7
$u_2 = 0$
$u_2 = 1$
$[u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k]$ = 64QAM symbol

Figure 34

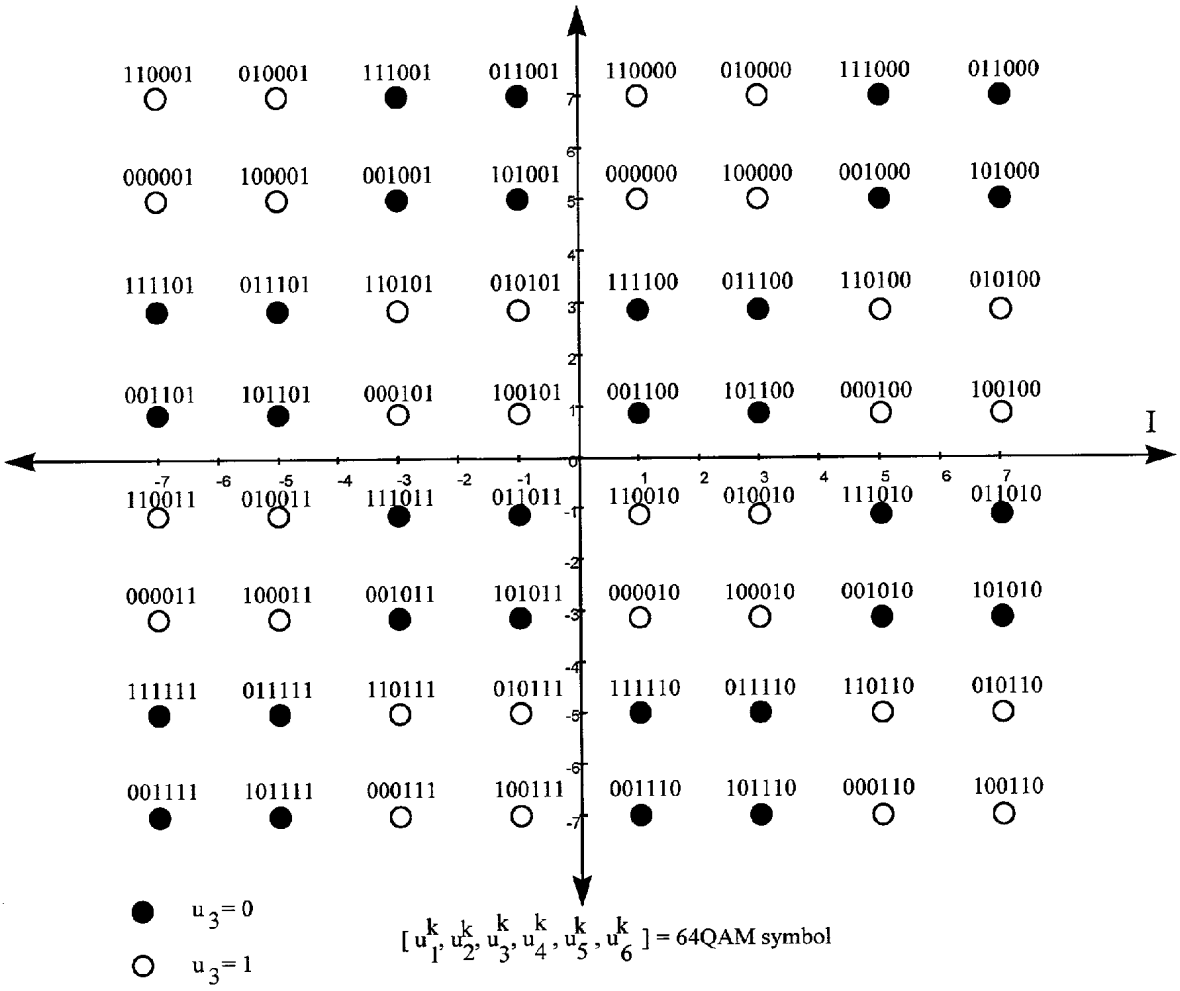

110001 010001 111001 011001 110000 010000 111000 011000
000001 100001 001001 101001 000000 100000 001000 101000
111101 011101 110101 010101 111100 011100 110100 010100
001101 101101 000101 100101 001100 101100 000100 100100
110011 010011 111011 011011 110010 010010 111010 011010
000011 100011 001011 101011 000010 100010 001010 101010
111111 011111 110111 010111 111110 011110 110110 010110
001111 101111 000111 100111 001110 101110 000110 100110
I
-7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7
7 6 5 4 3 2 1 -1 -2 -3 -4 -5 -6 -7
$u_3 = 0$
$u_3 = 1$
$[u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k]$ = 64QAM symbol

Figure 35

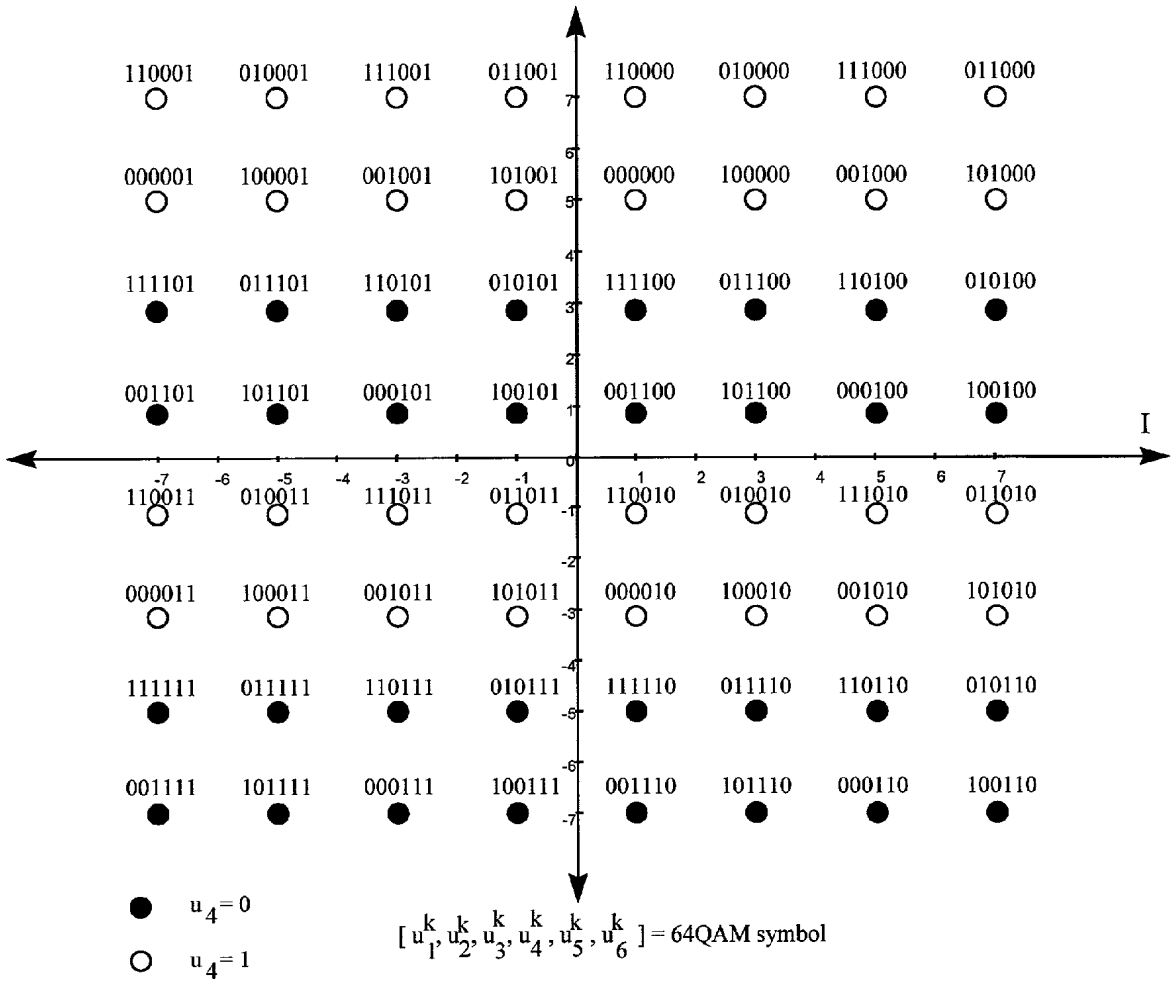

110001 010001 111001 011001 110000 010000 111000 011000
000001 100001 001001 101001 000000 100000 001000 101000
111101 011101 110101 010101 111100 011100 110100 010100
001101 101101 000101 100101 001100 101100 000100 100100
110011 010011 111011 011011 110010 010010 111010 011010
000011 100011 001011 101011 000010 100010 001010 101010
111111 011111 110111 010111 111110 011110 110110 010110
001111 101111 000111 100111 001110 101110 000110 100110
I
-7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7
7 6 5 4 3 2 1 -1 -2 -3 -4 -5 -6 -7
$u_4 = 0$
$u_4 = 1$
$[u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k]$ = 64QAM symbol

Figure 36

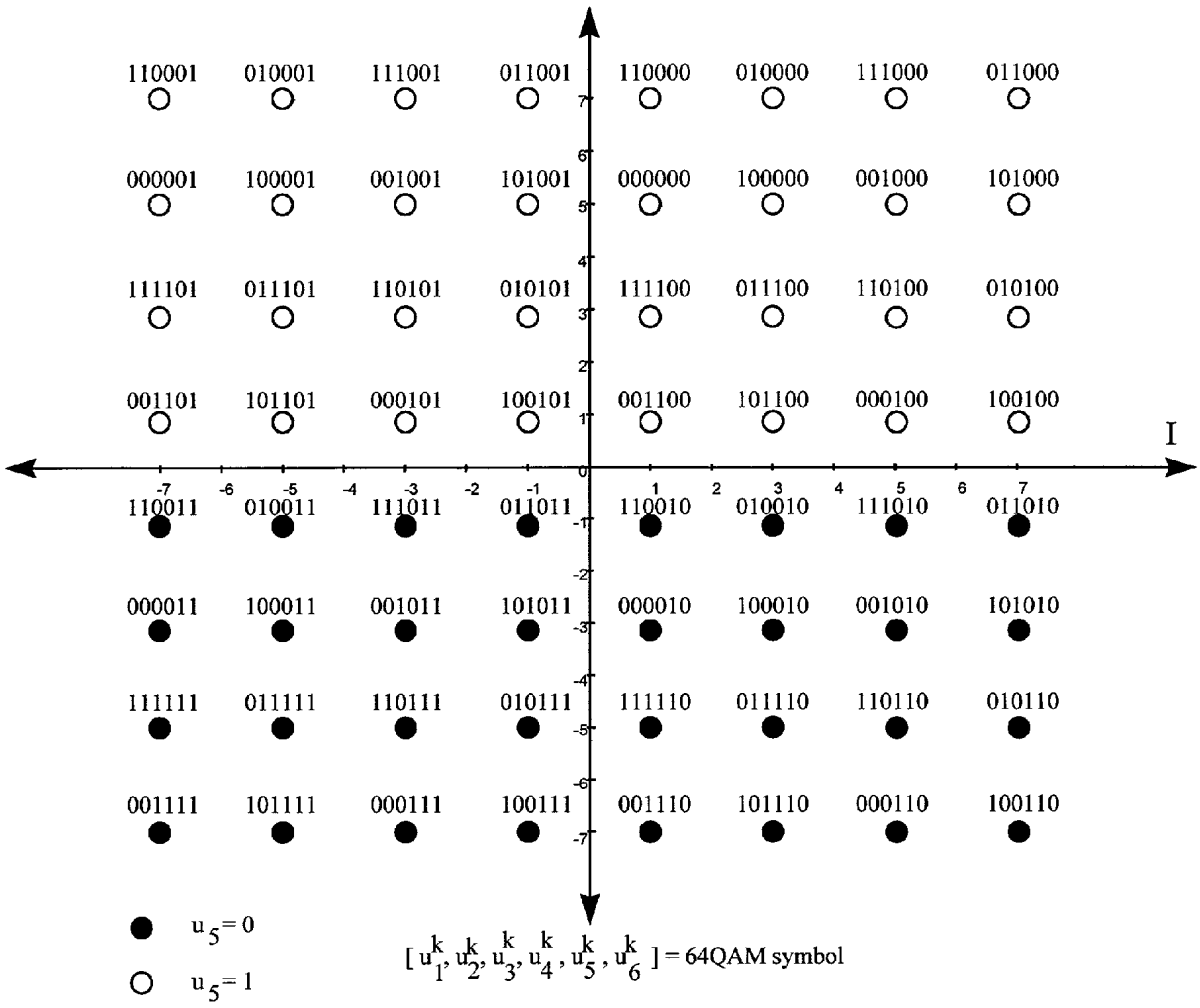

110001 010001 111001 011001 110000 010000 111000 011000
000001 100001 001001 101001 000000 100000 001000 101000
111101 011101 110101 010101 111100 011100 110100 010100
001101 101101 000101 100101 001100 101100 000100 100100
110011 010011 111011 011011 110010 010010 111010 011010
000011 100011 001011 101011 000010 100010 001010 101010
111111 011111 110111 010111 111110 011110 110110 010110
001111 101111 000111 100111 001110 101110 000110 100110
I
-7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7
7 6 5 4 3 2 1 -1 -2 -3 -4 -5 -6 -7
$u_5 = 0$
$u_5 = 1$
$[u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k]$ = 64QAM symbol

Figure 37

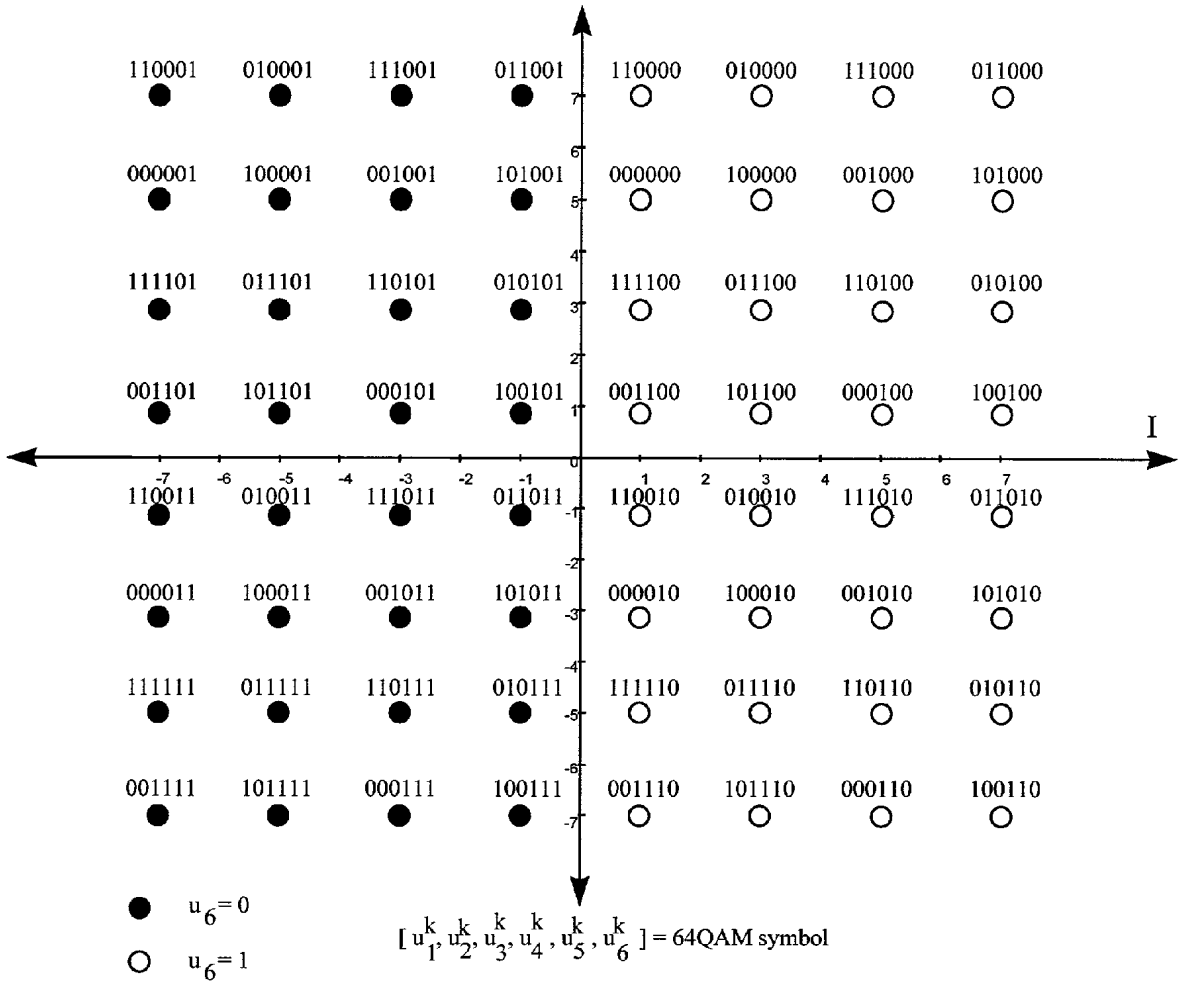

110001 010001 111001 011001 110000 010000 111000 011000
000001 100001 001001 101001 000000 100000 001000 101000
111101 011101 110101 010101 111100 011100 110100 010100
001101 101101 000101 100101 001100 101100 000100 100100
110011 010011 111011 011011 110010 010010 111010 011010
000011 100011 001011 101011 000010 100010 001010 101010
111111 011111 110111 010111 111110 011110 110110 010110
001111 101111 000111 100111 001110 101110 000110 100110
I
-7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7
7 6 5 4 3 2 1 -1 -2 -3 -4 -5 -6 -7
$u_6 = 0$
$u_6 = 1$
$[u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k]$ = 64QAM symbol

Figure 38

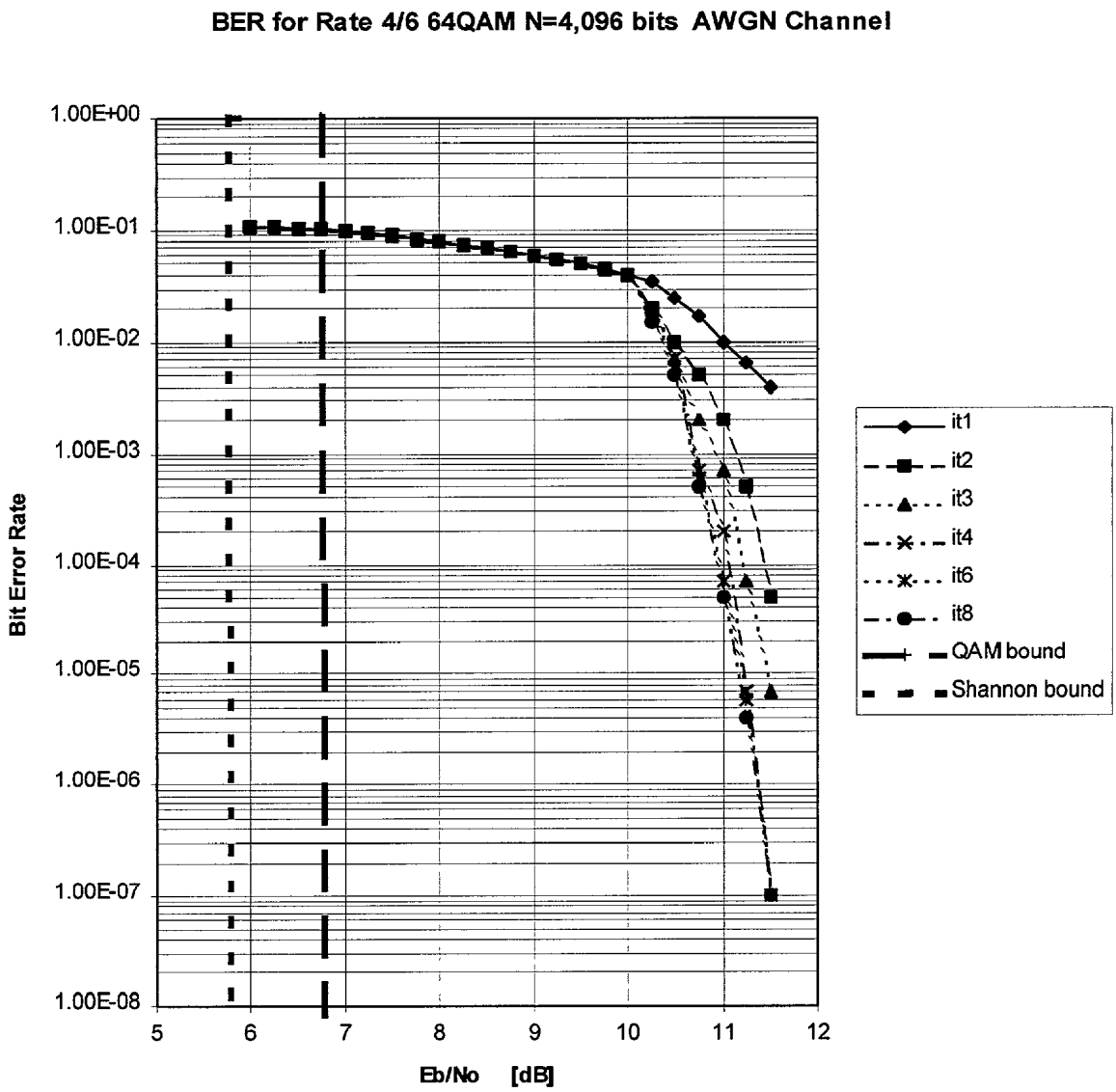
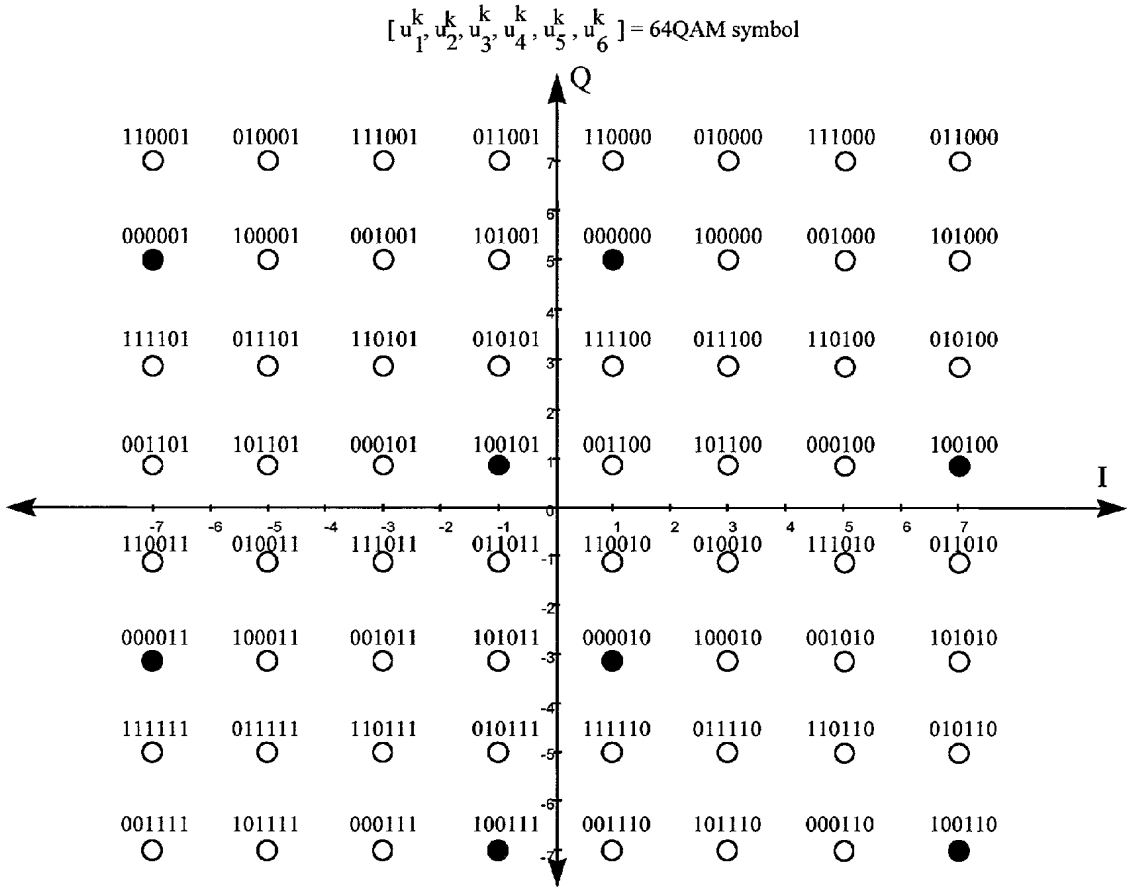

BER for Rate 4/6 64QAM N=4,096 bits AWGN Channel
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
5
6
7
8
9
10
11
12
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 39



● This 4 point constellation is selected by the four coded bits 0000

● Each of the four points of the 1001 constellation is selected by two uncoded bits using Gray mapping

Figure 40

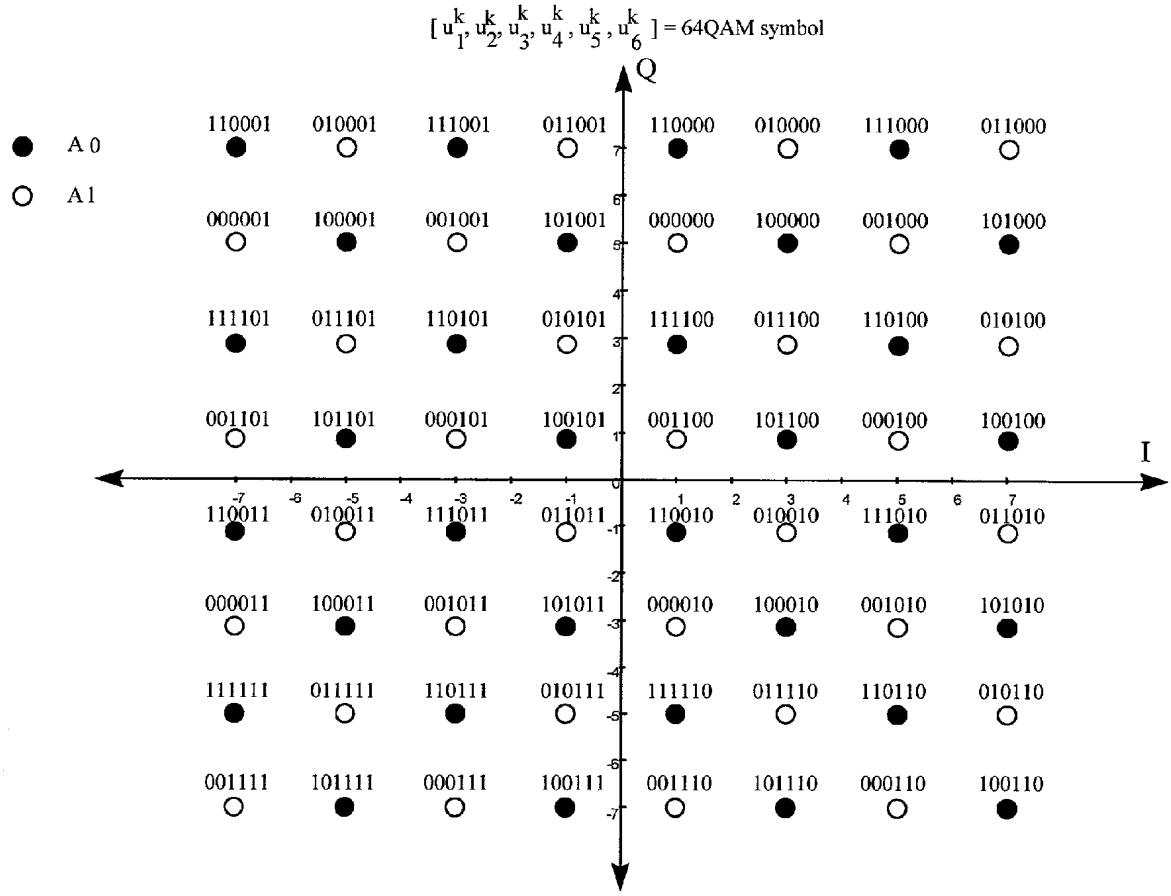

[u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k] = 64QAM symbol
Q
I
A 0
A 1
110001 010001 111001 011001 110000 010000 111000 011000
000001 100001 001001 101001 000000 100000 001000 101000
111101 011101 110101 010101 111100 011100 110100 010100
001101 101101 000101 100101 001100 101100 000100 100100
110011 010011 111011 011011 110010 010010 111010 011010
000011 100011 001011 101011 000010 100010 001010 101010
111111 011111 110111 010111 111110 011110 110110 010110
001111 101111 000111 100111 001110 101110 000110 100110

Figure 41

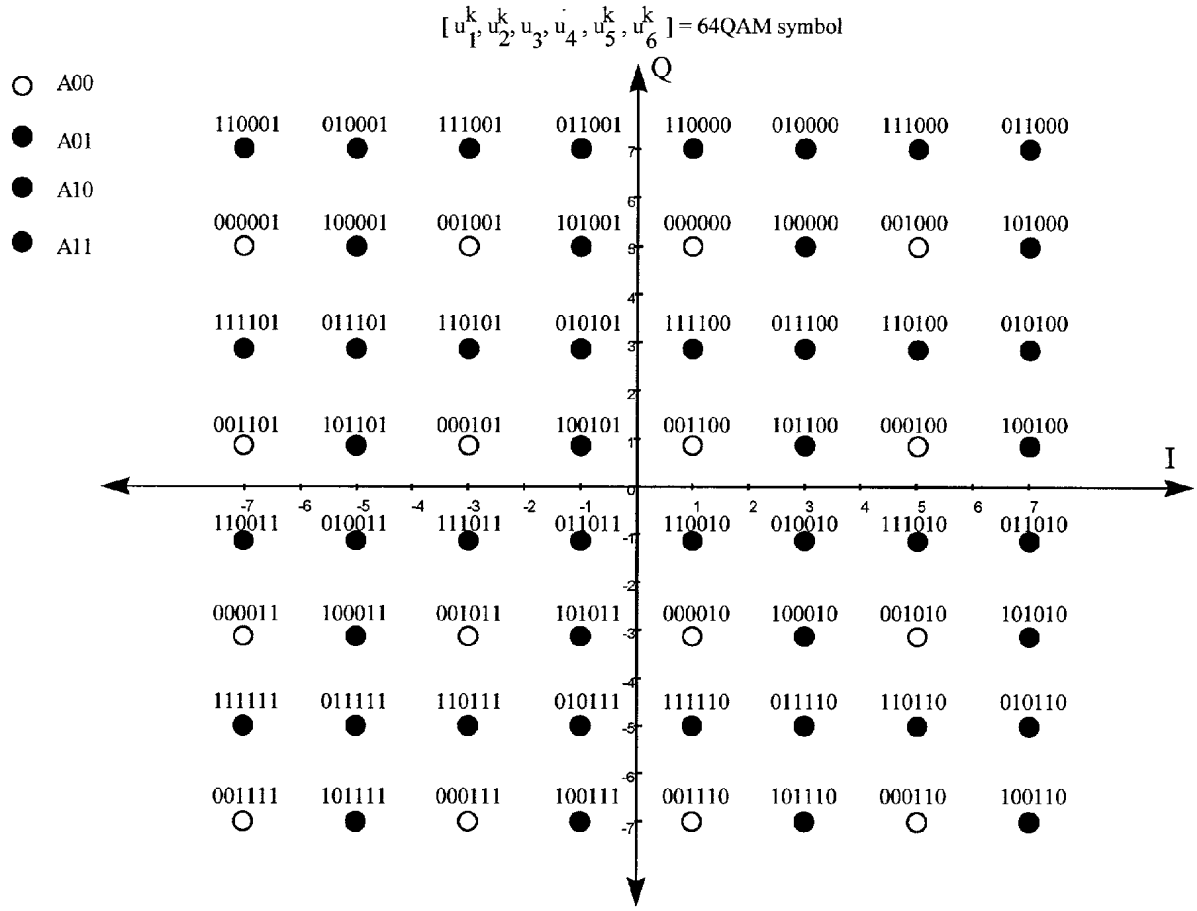

[u_1^k, u_2^k, u_3, u_4, u_5^k, u_6^k] = 64QAM symbol
A00
A01
A10
A11
Q
I
110001 010001 111001 011001 110000 010000 111000 011000
000001 100001 001001 101001 000000 100000 001000 101000
111101 011101 110101 010101 111100 011100 110100 010100
001101 101101 000101 100101 001100 101100 000100 100100
110011 010011 111011 011011 110010 010010 111010 011010
000011 100011 001011 101011 000010 100010 001010 101010
111111 011111 110111 010111 111110 011110 110110 010110
001111 101111 000111 100111 001110 101110 000110 100110

Figure 42

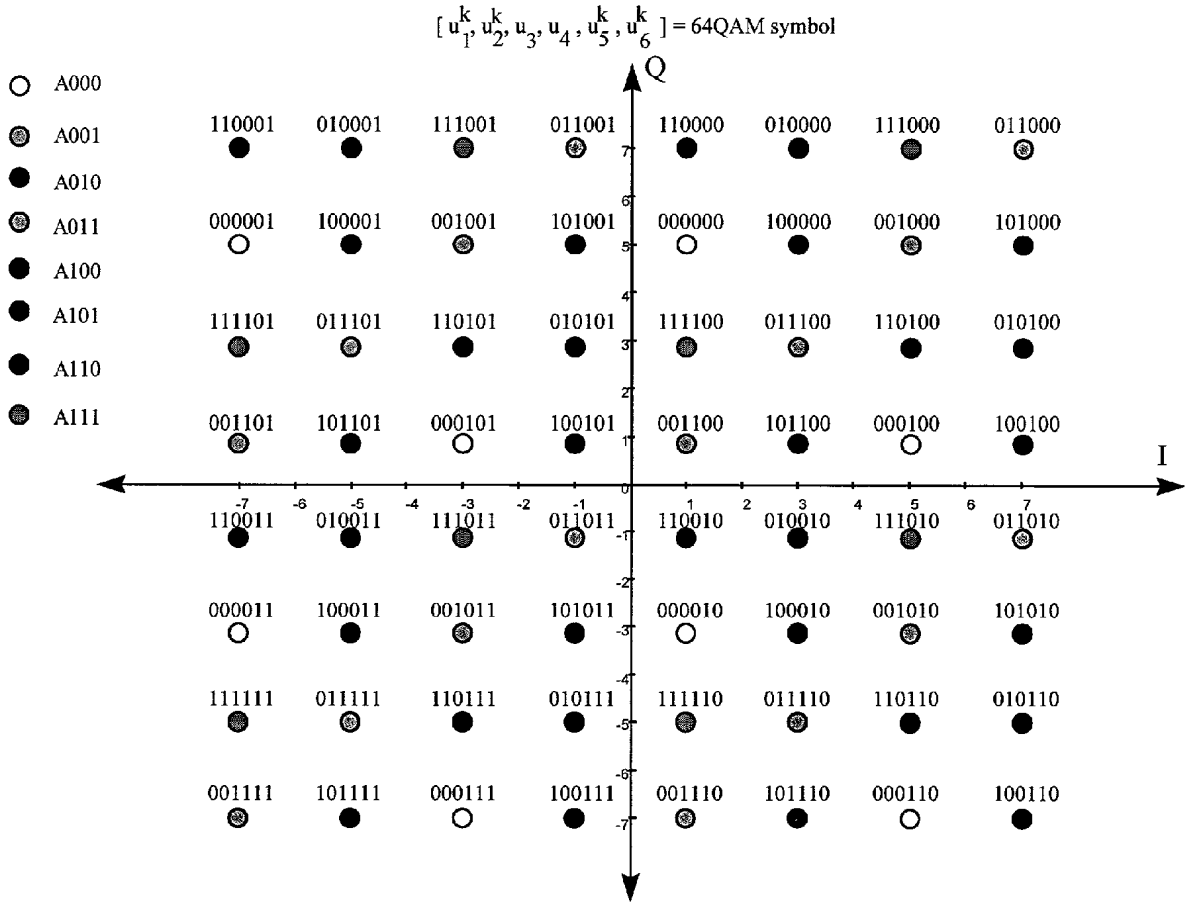

[u_1^k, u_2^k, u_3, u_4, u_5^k, u_6^k] = 64QAM symbol
A000
A001
A010
A011
A100
A101
A110
A111
Q
I
110001 010001 111001 011001 110000 010000 111000 011000
000001 100001 001001 101001 000000 100000 001000 101000
111101 011101 110101 010101 111100 011100 110100 010100
001101 101101 000101 100101 001100 101100 000100 100100
110011 010011 111011 011011 110010 010010 111010 011010
000011 100011 001011 101011 000010 100010 001010 101010
111111 011111 110111 010111 111110 011110 110110 010110
001111 101111 000111 100111 001110 101110 000110 100110

Figure 43

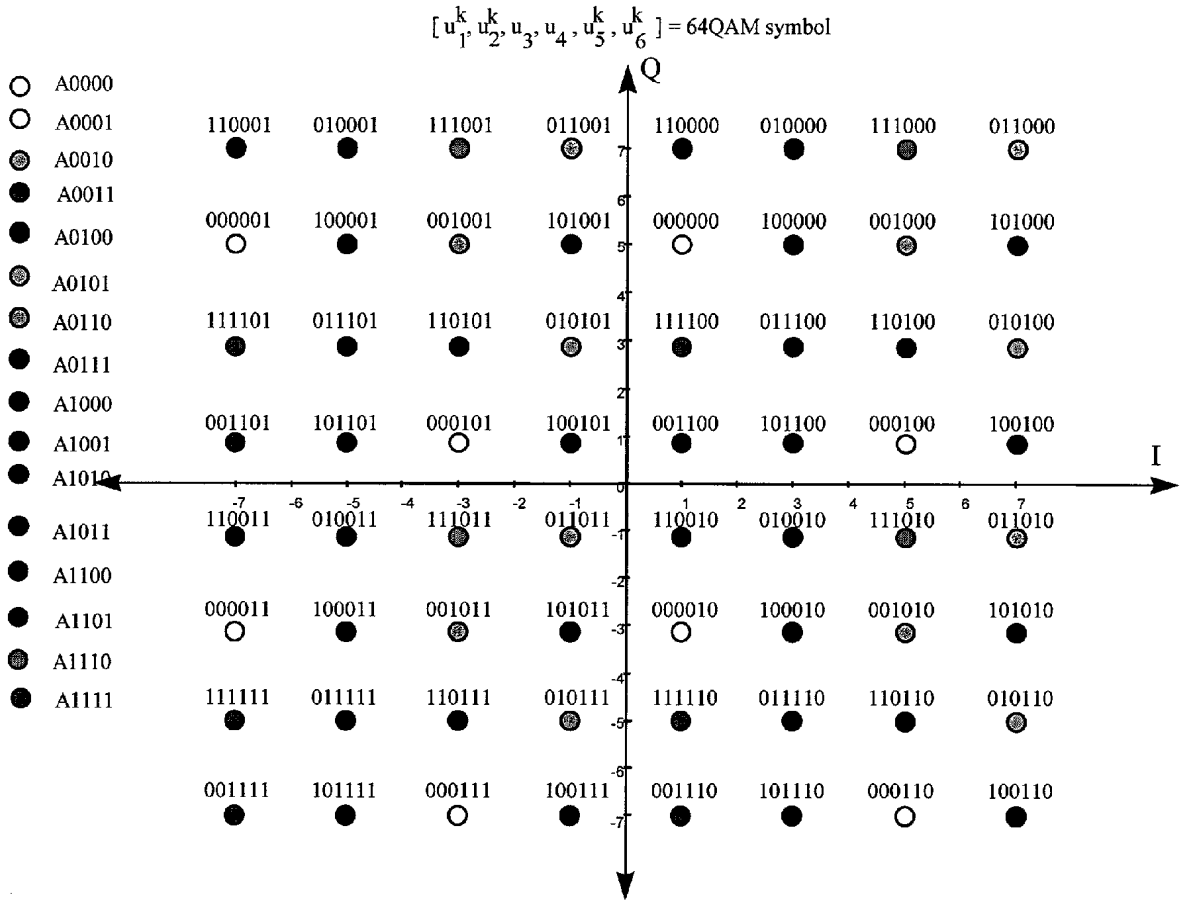

[u_1^k, u_2^k, u_3, u_4, u_5^k, u_6^k] = 64QAM symbol
A0000
A0001
A0010
A0011
A0100
A0101
A0110
A0111
A1000
A1001
A1010
A1011
A1100
A1101
A1110
A1111
Q
I
110001 010001 111001 011001 110000 010000 111000 011000
000001 100001 001001 101001 000000 100000 001000 101000
111101 011101 110101 010101 111100 011100 110100 010100
001101 101101 000101 100101 001100 101100 000100 100100
110011 010011 111011 011011 110010 010010 111010 011010
000011 100011 001011 101011 000010 100010 001010 101010
111111 011111 110111 010111 111110 011110 110110 010110
001111 101111 000111 100111 001110 101110 000110 100110

Figure 44

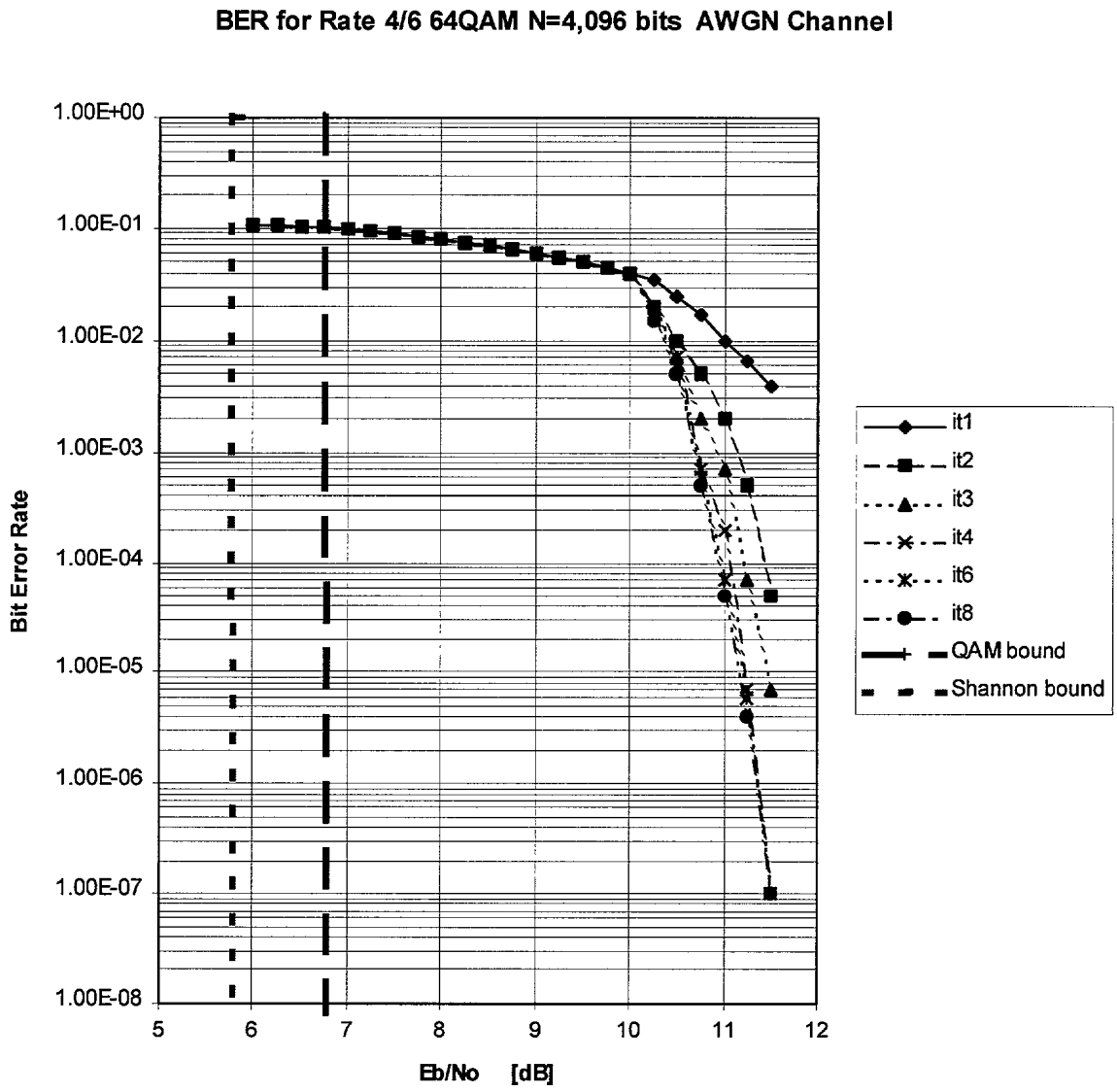

BER for Rate 4/6 64QAM N=4,096 bits AWGN Channel
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
5
6
7
8
9
10
11
12
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 45

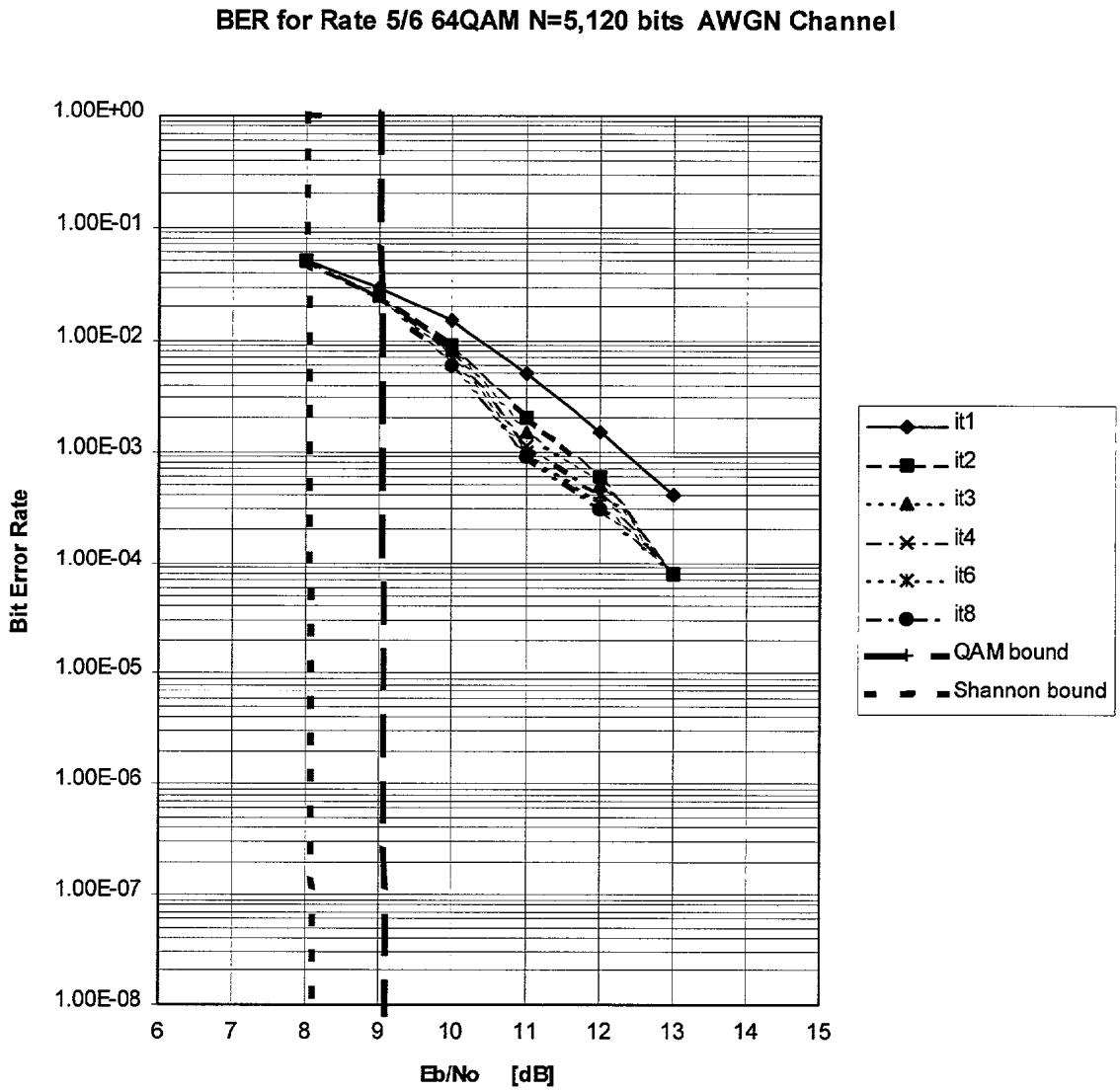

BER for Rate 5/6 64QAM N=5,120 bits AWGN Channel
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
6
7
8
9
10
11
12
13
14
15
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 46

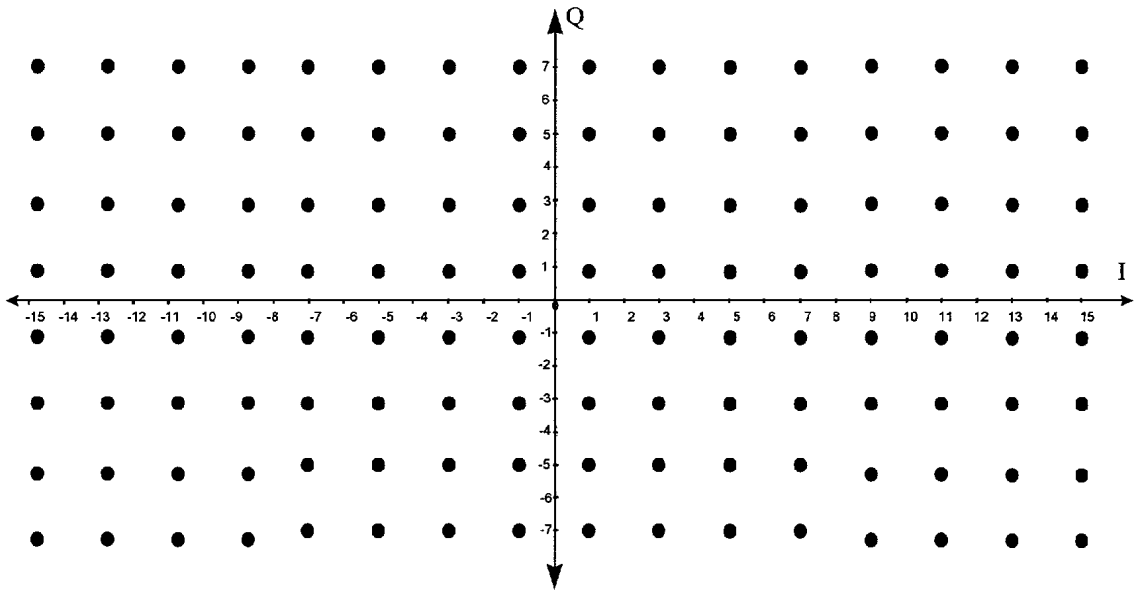


Figure 47

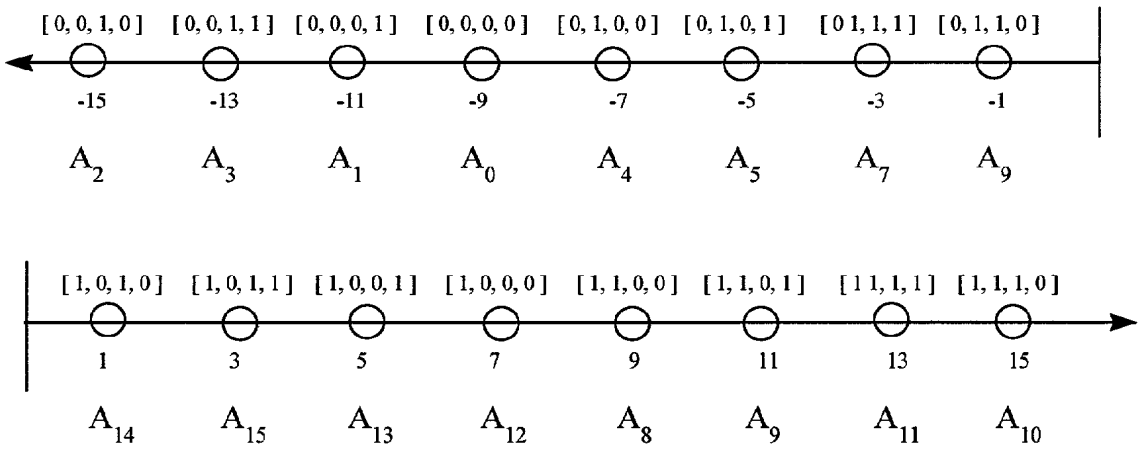


Figure 48

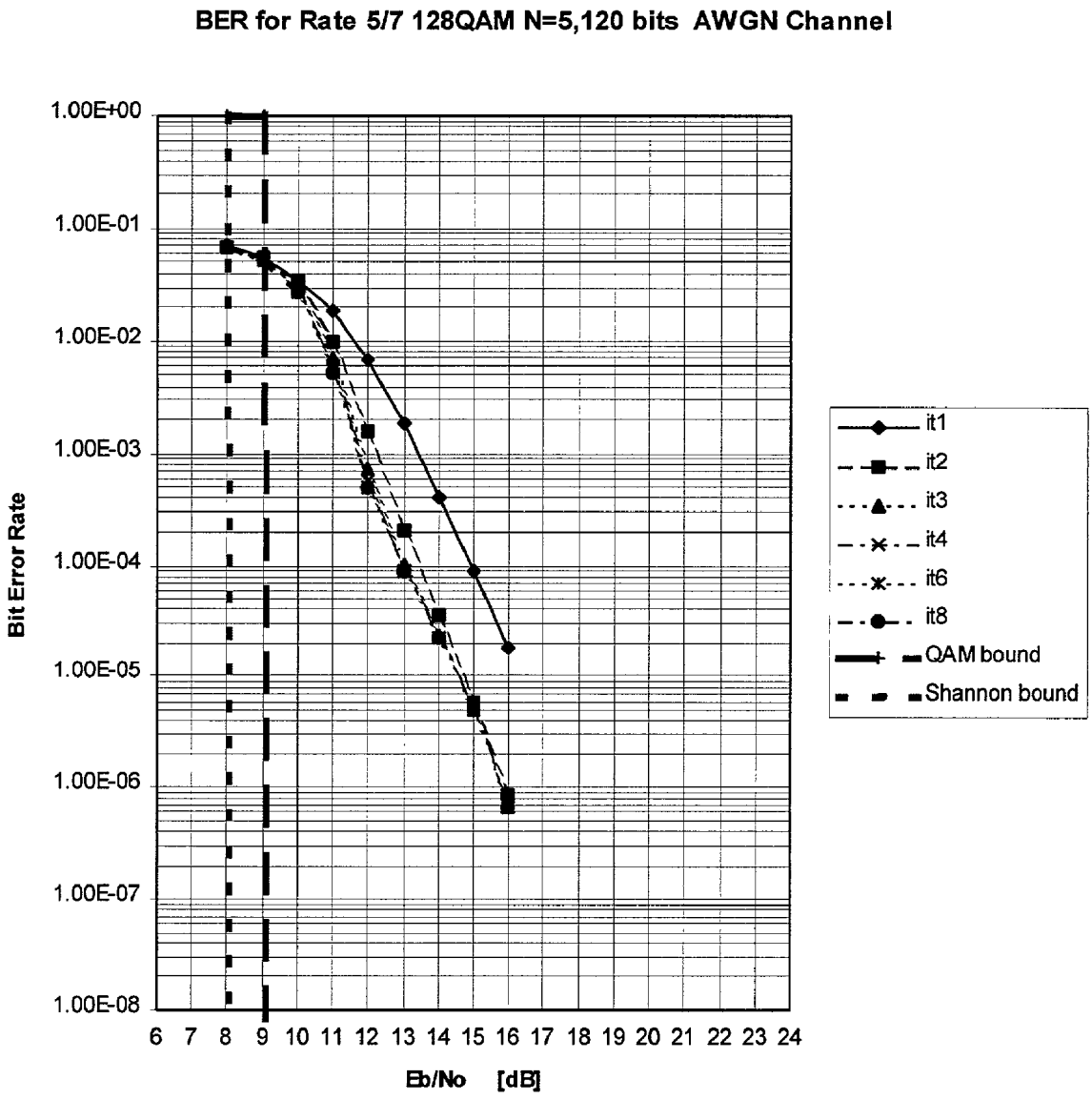

BER for Rate 5/7 128QAM N=5,120 bits AWGN Channel
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 49

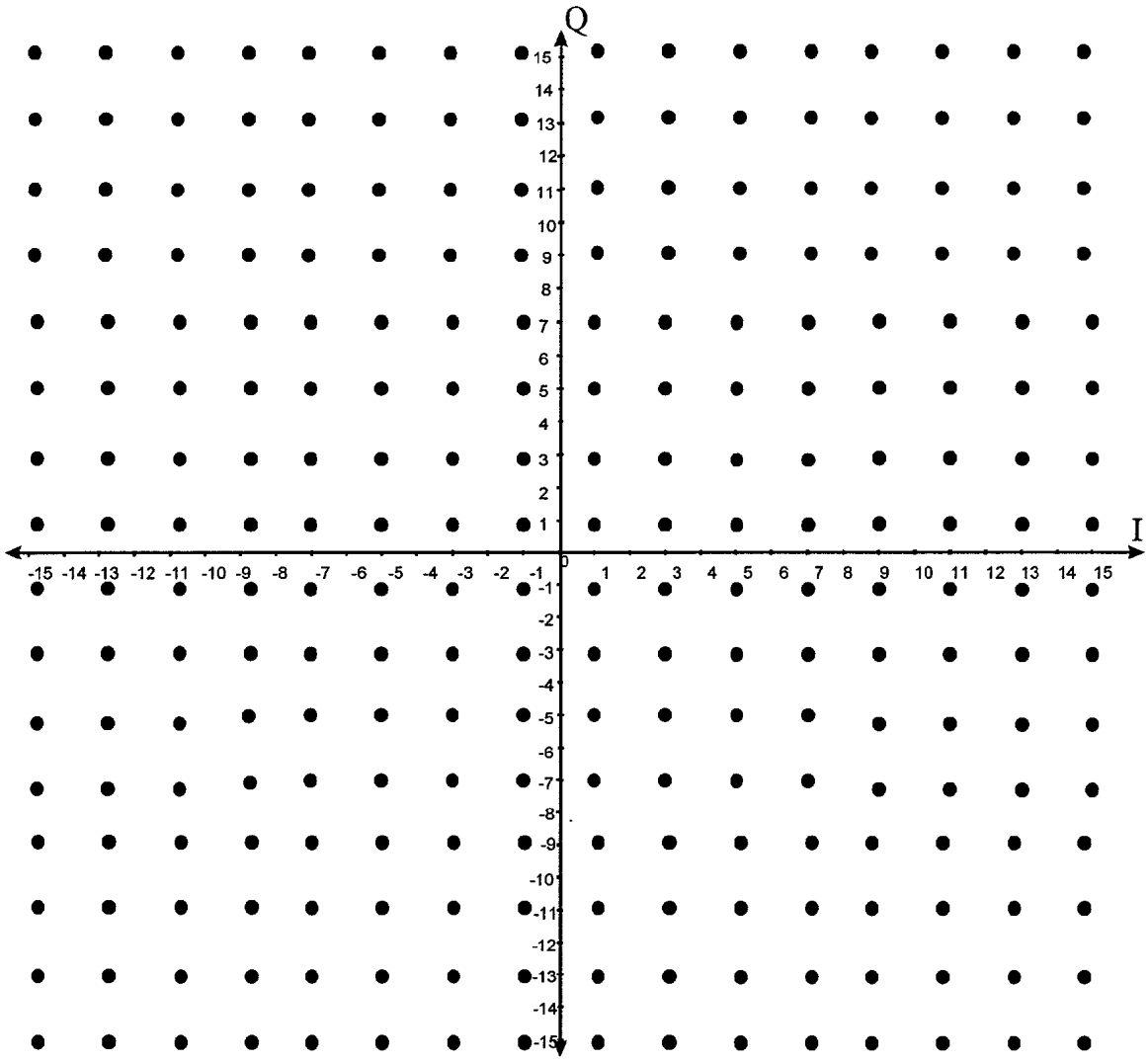

Q
I
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-7
-8
-9
-10
-11
-12
-13
-14
-15
-15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Figure 50

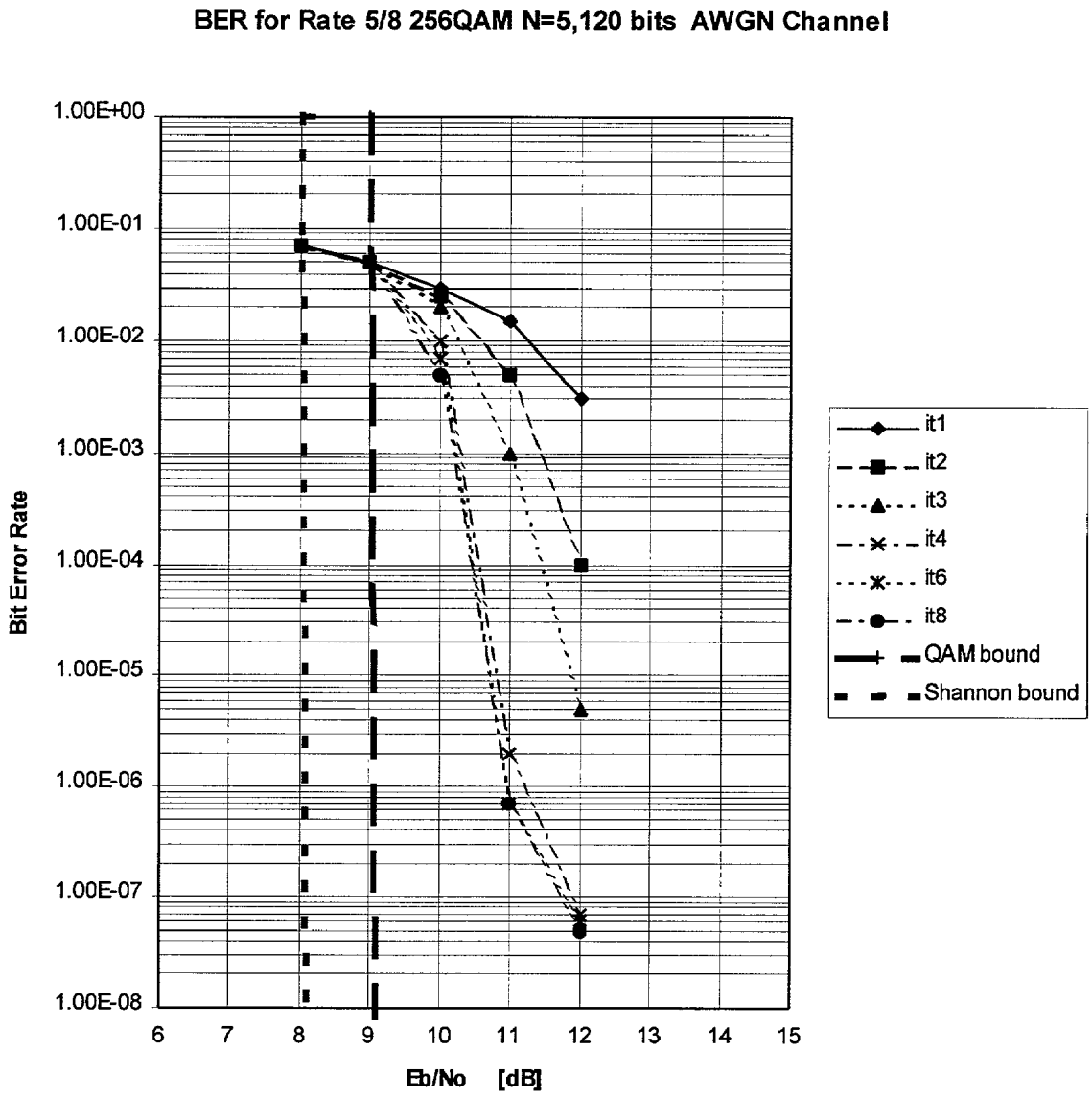

BER for Rate 5/8 256QAM N=5,120 bits AWGN Channel
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
6
7
8
9
10
11
12
13
14
15
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 51

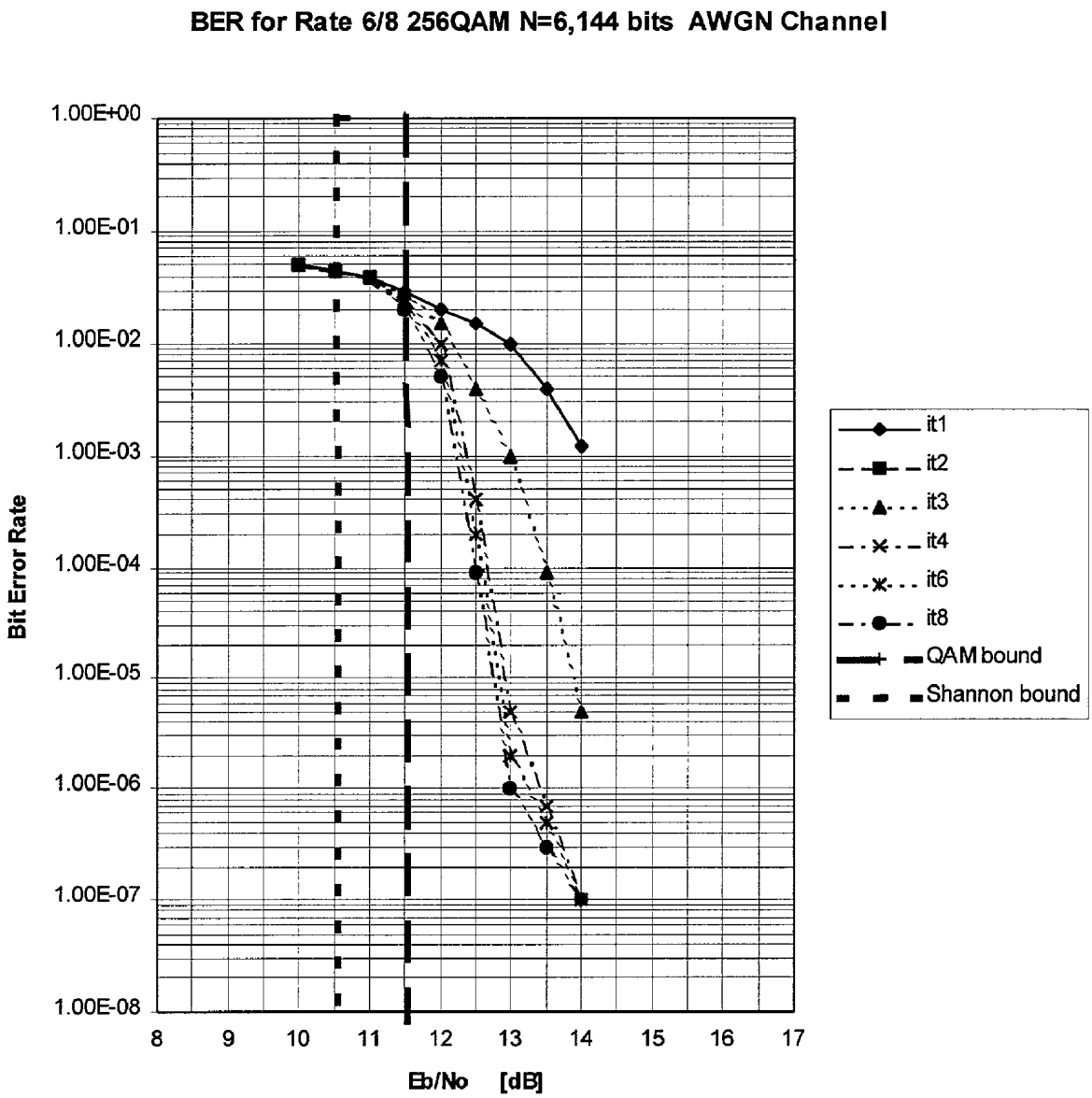

BER for Rate 6/8 256QAM N=6,144 bits AWGN Channel
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
8
9
10
11
12
13
14
15
16
17
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 52

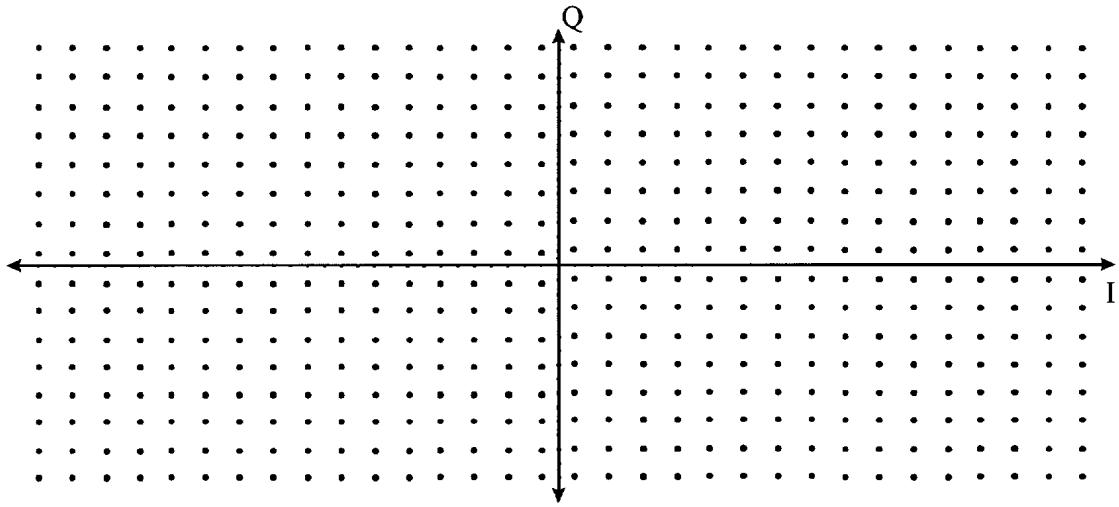


Figure 53

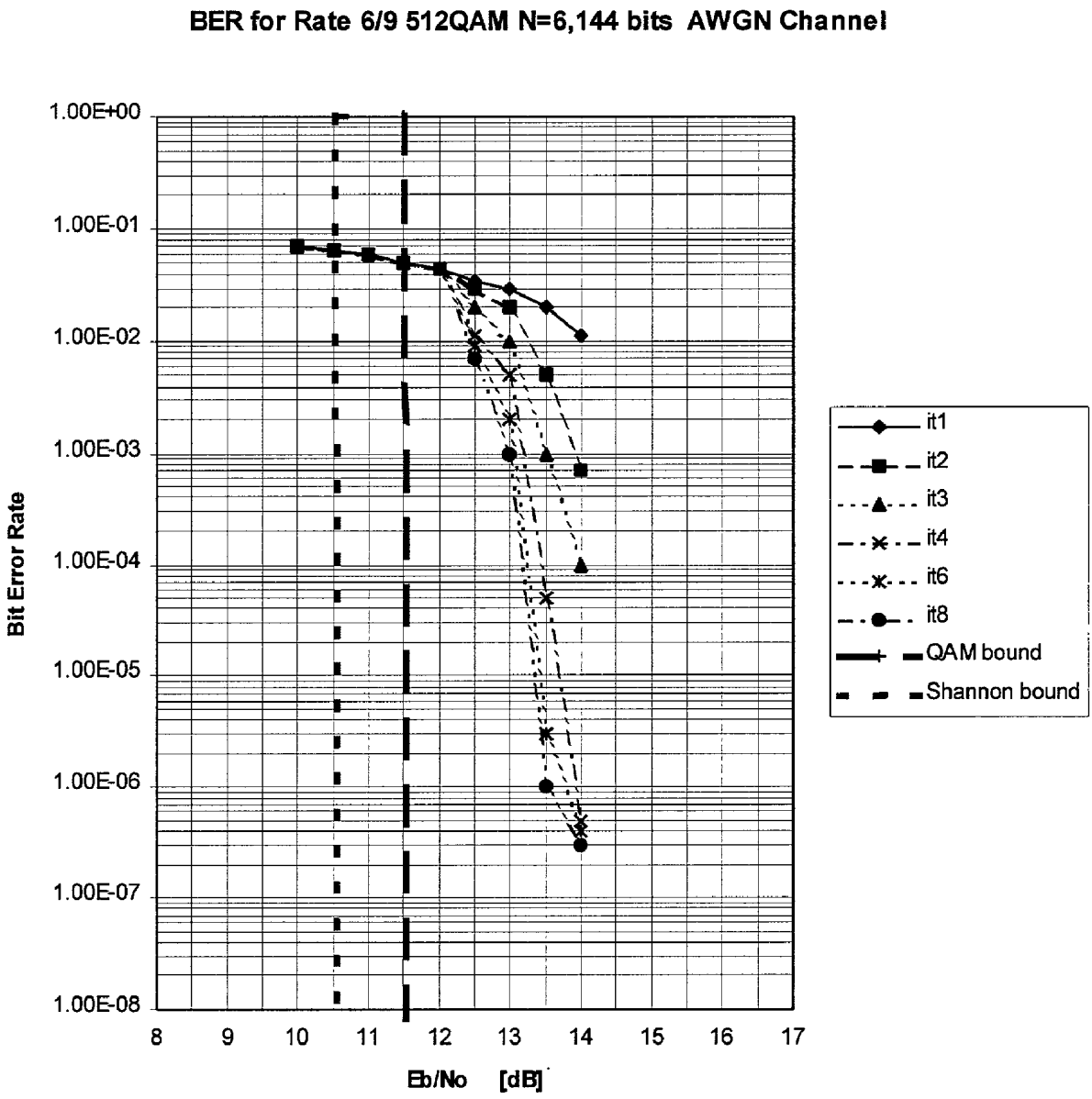

BER for Rate 6/9 512QAM N=6,144 bits AWGN Channel
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
8
9
10
11
12
13
14
15
16
17
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 54

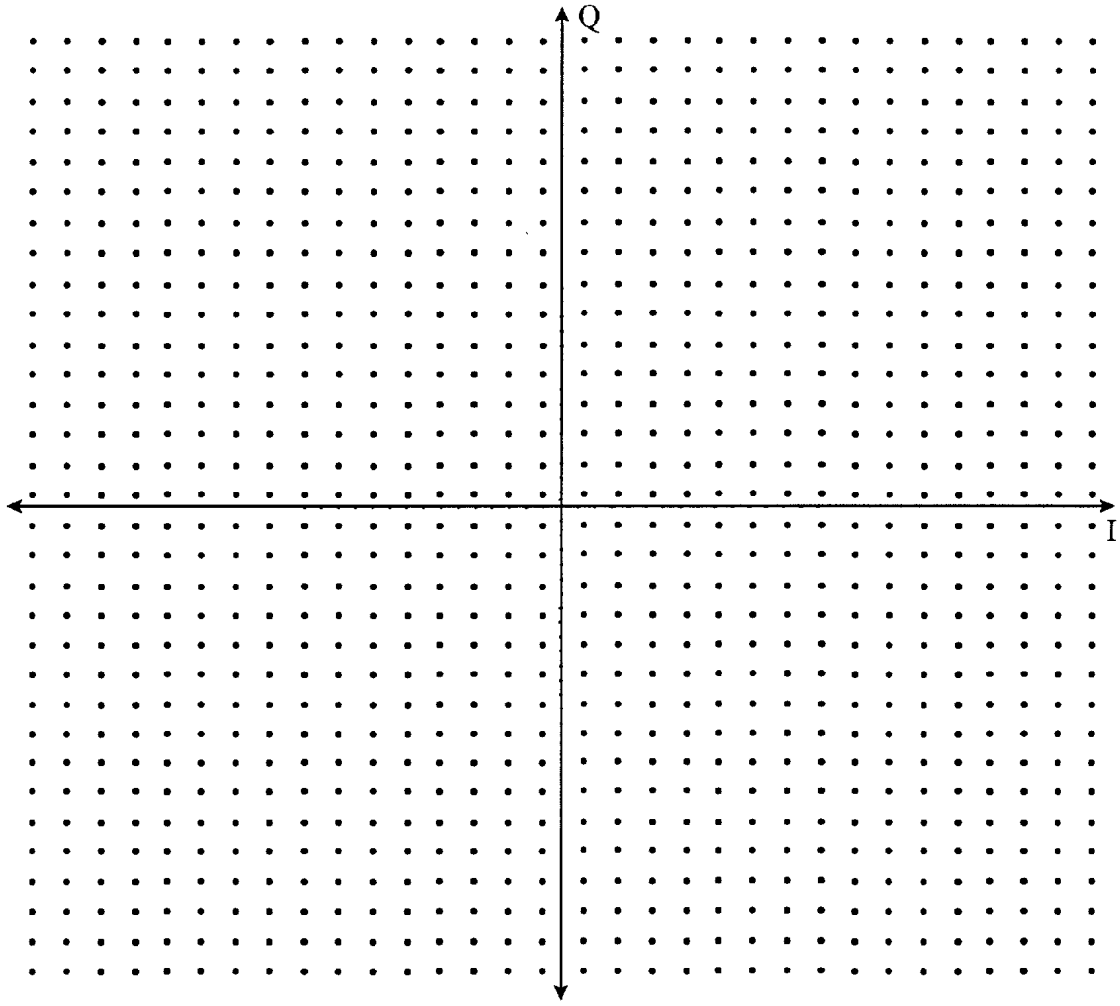


Figure 55

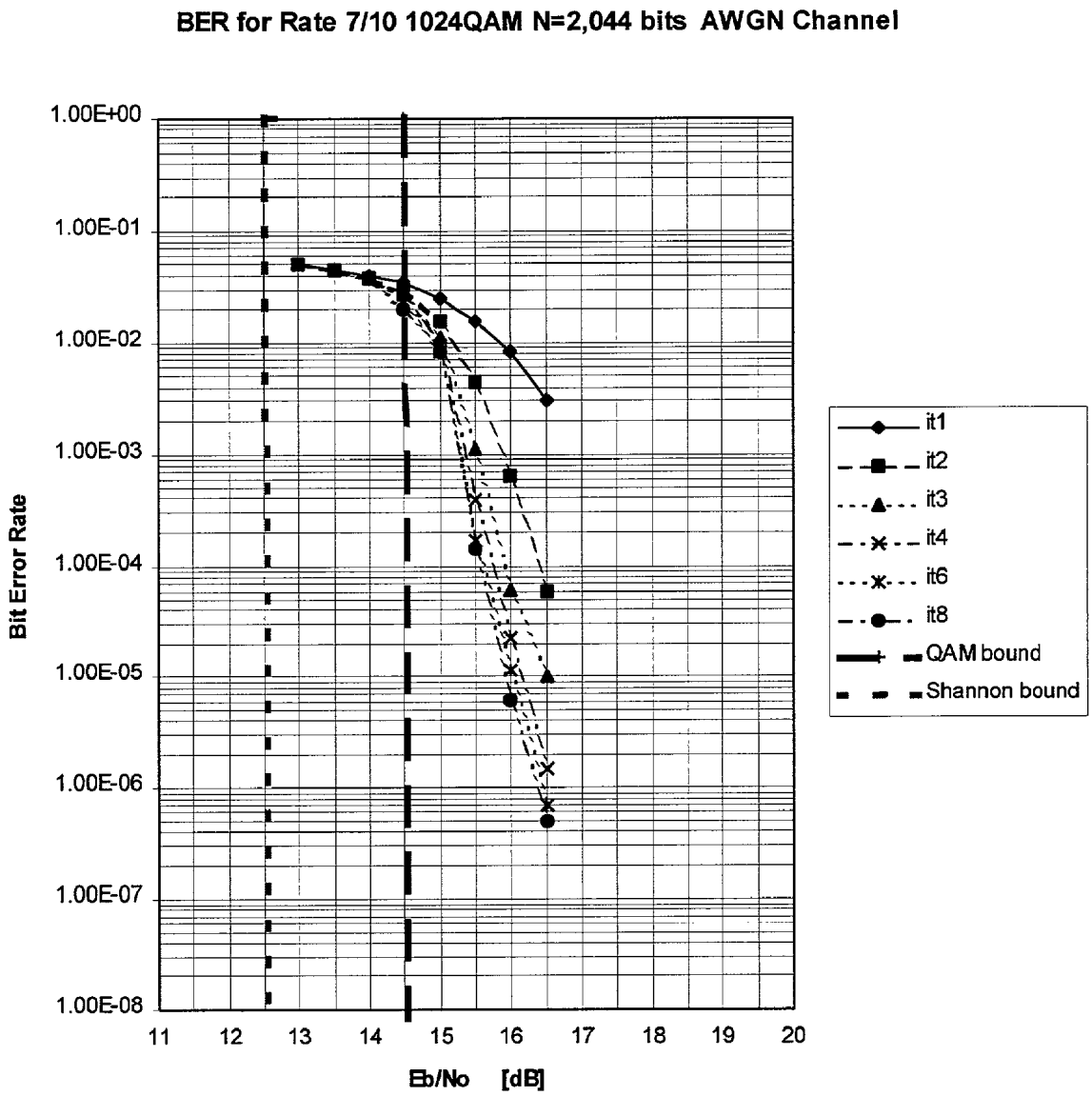

BER for Rate 7/10 1024QAM N=2,044 bits AWGN Channel
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
11
12
13
14
15
16
17
18
19
20
Eb/No [dB]
it1
it2
it3
it4
it6
it8
QAM bound
Shannon bound

Figure 56

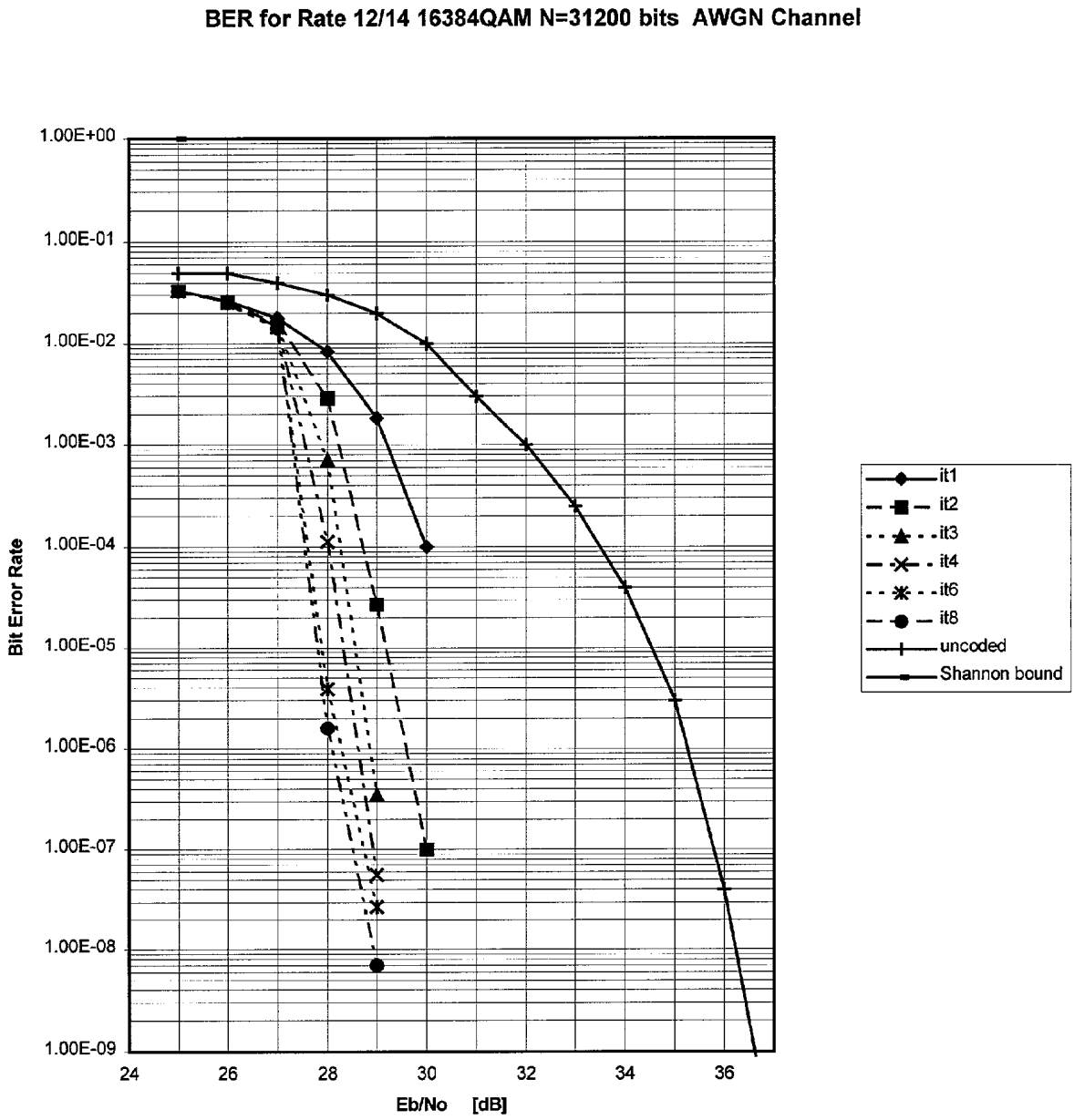

BER for Rate 12/14 16384QAM N=31200 bits AWGN Channel
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
1.00E-09
Bit Error Rate
24
26
28
30
32
34
36
Eb/No [dB]
it1
it2
it3
it4
it6
it8
uncoded
Shannon bound

Figure 57

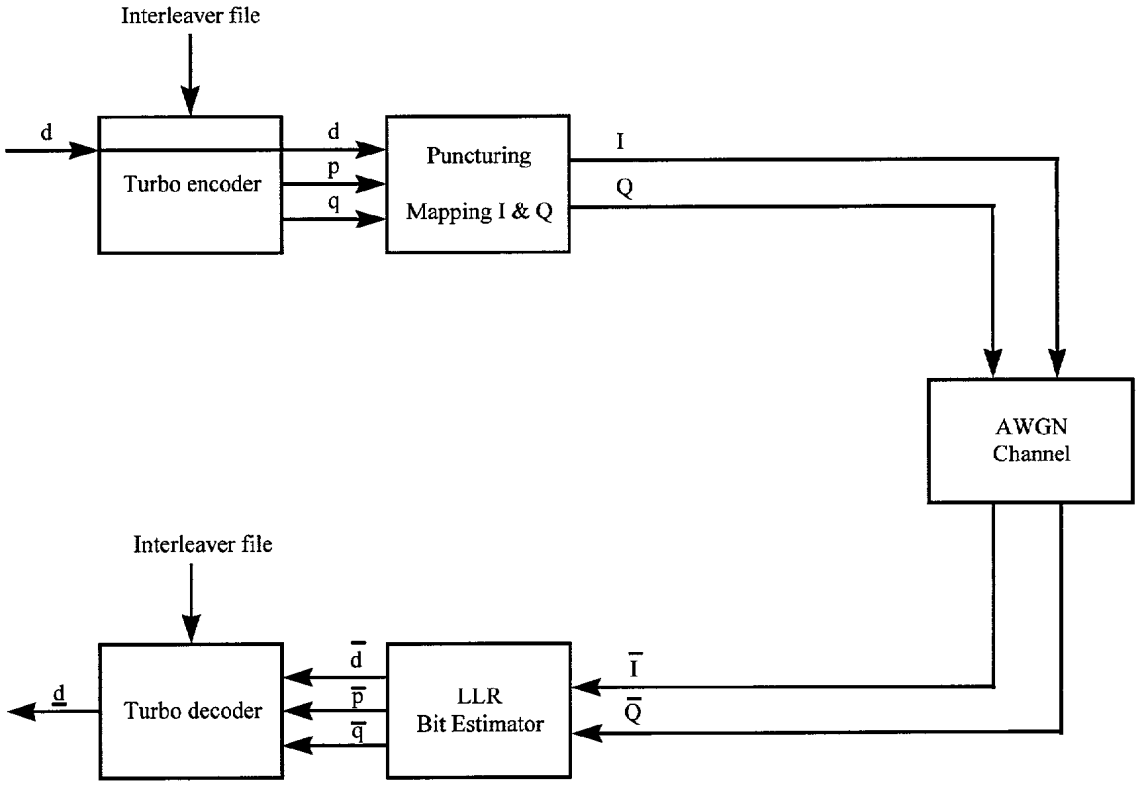

Interleaver file
d
Turbo encoder
d
p
q
Puncturing
Mapping I & Q
I
Q
AWGN
Channel
Interleaver file
d
Turbo decoder
d̄
p̄
q̄
LLR
Bit Estimator
Ī
Q̄

Figure 58

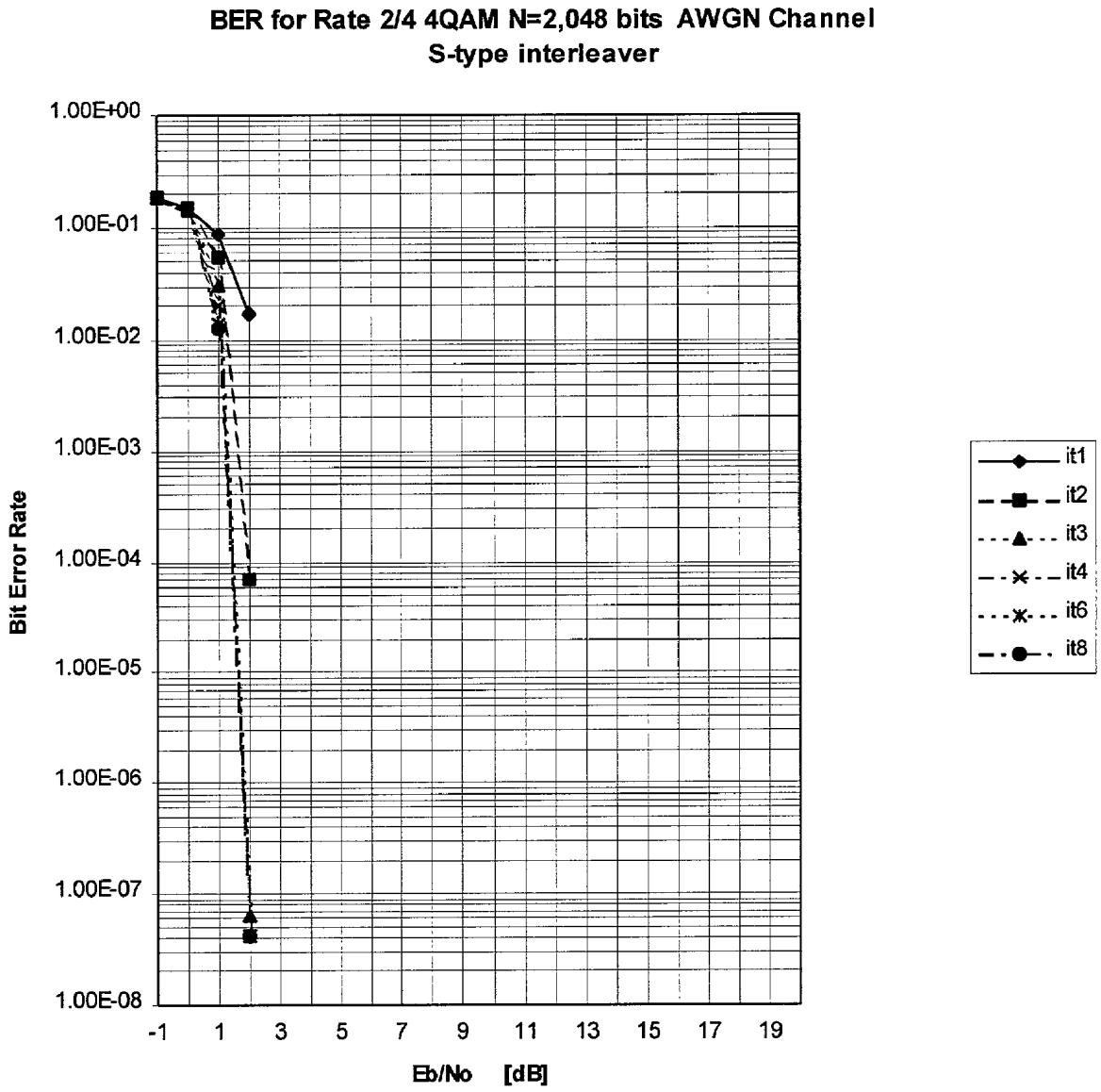

BER for Rate 2/4 4QAM N=2,048 bits AWGN Channel
S-type interleaver
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
1
3
5
7
9
11
13
15
17
19
Eb/No [dB]
it1
it2
it3
it4
it6
it8

Figure 59

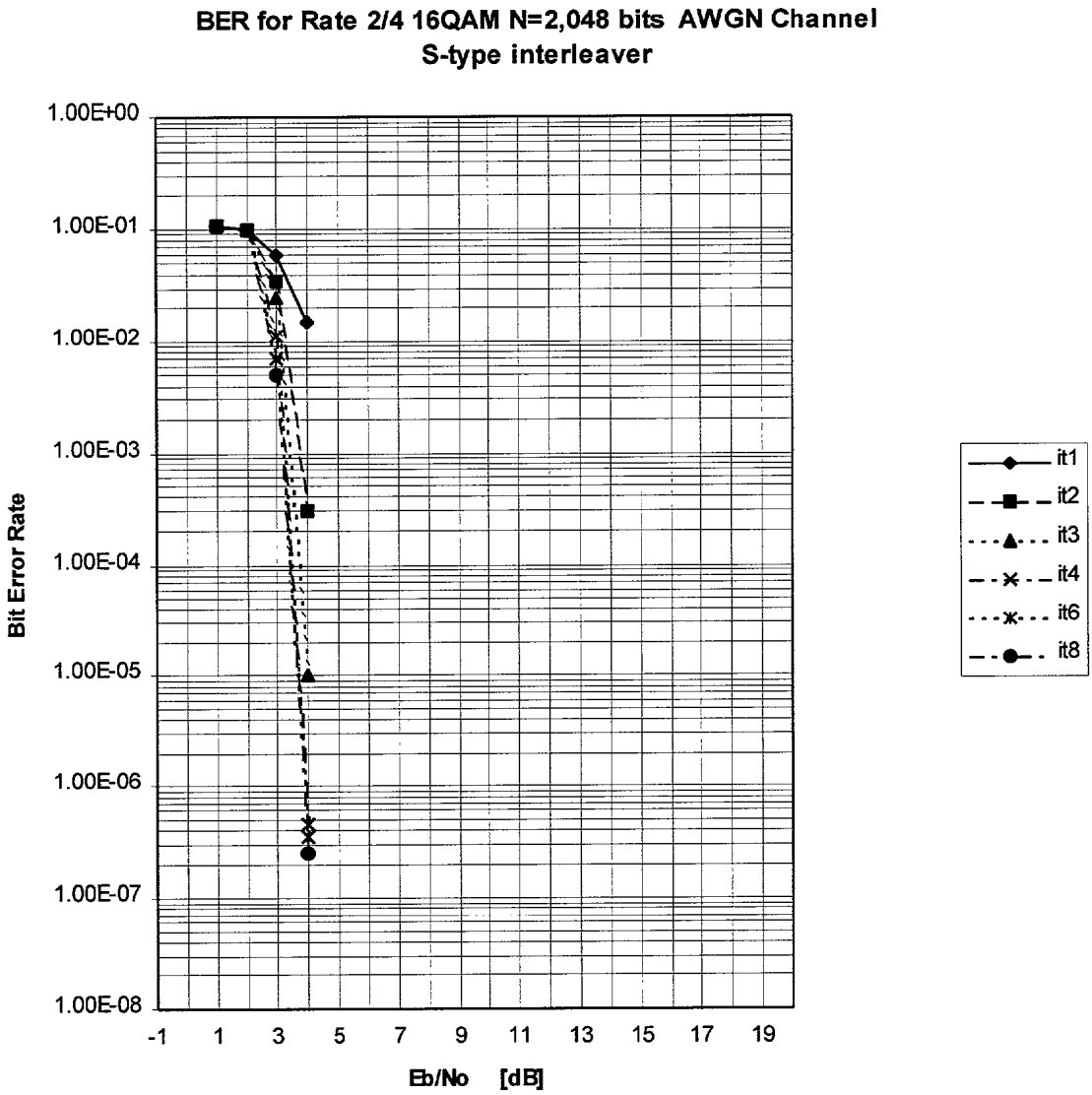

BER for Rate 2/4 16QAM N=2,048 bits AWGN Channel
S-type interleaver
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
1
3
5
7
9
11
13
15
17
19
Eb/No [dB]
it1
it2
it3
it4
it6
it8

Figure 60

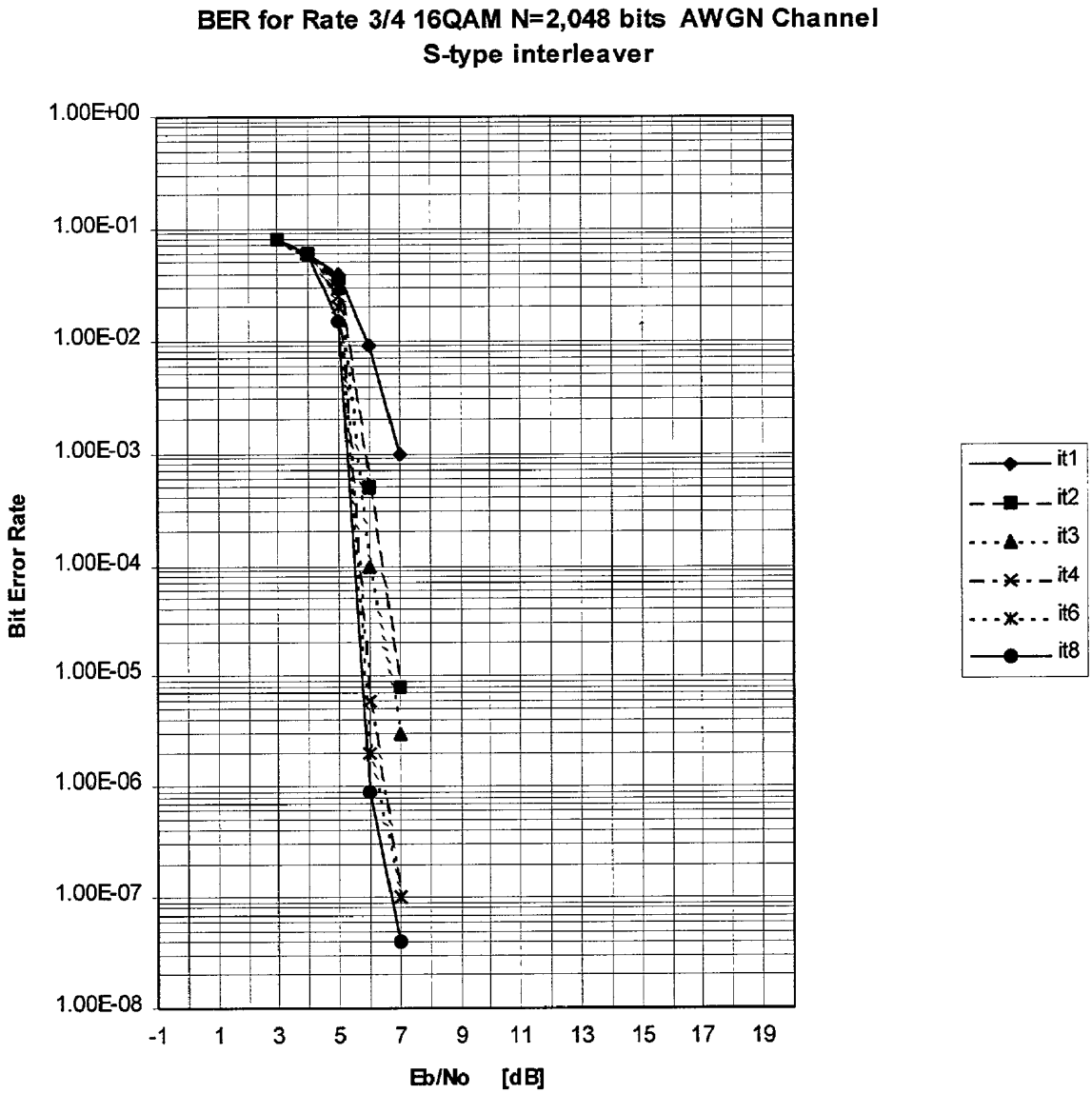

BER for Rate 3/4 16QAM N=2,048 bits AWGN Channel
S-type interleaver
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
1
3
5
7
9
11
13
15
17
19
Eb/No [dB]
it1
it2
it3
it4
it6
it8

Figure 61

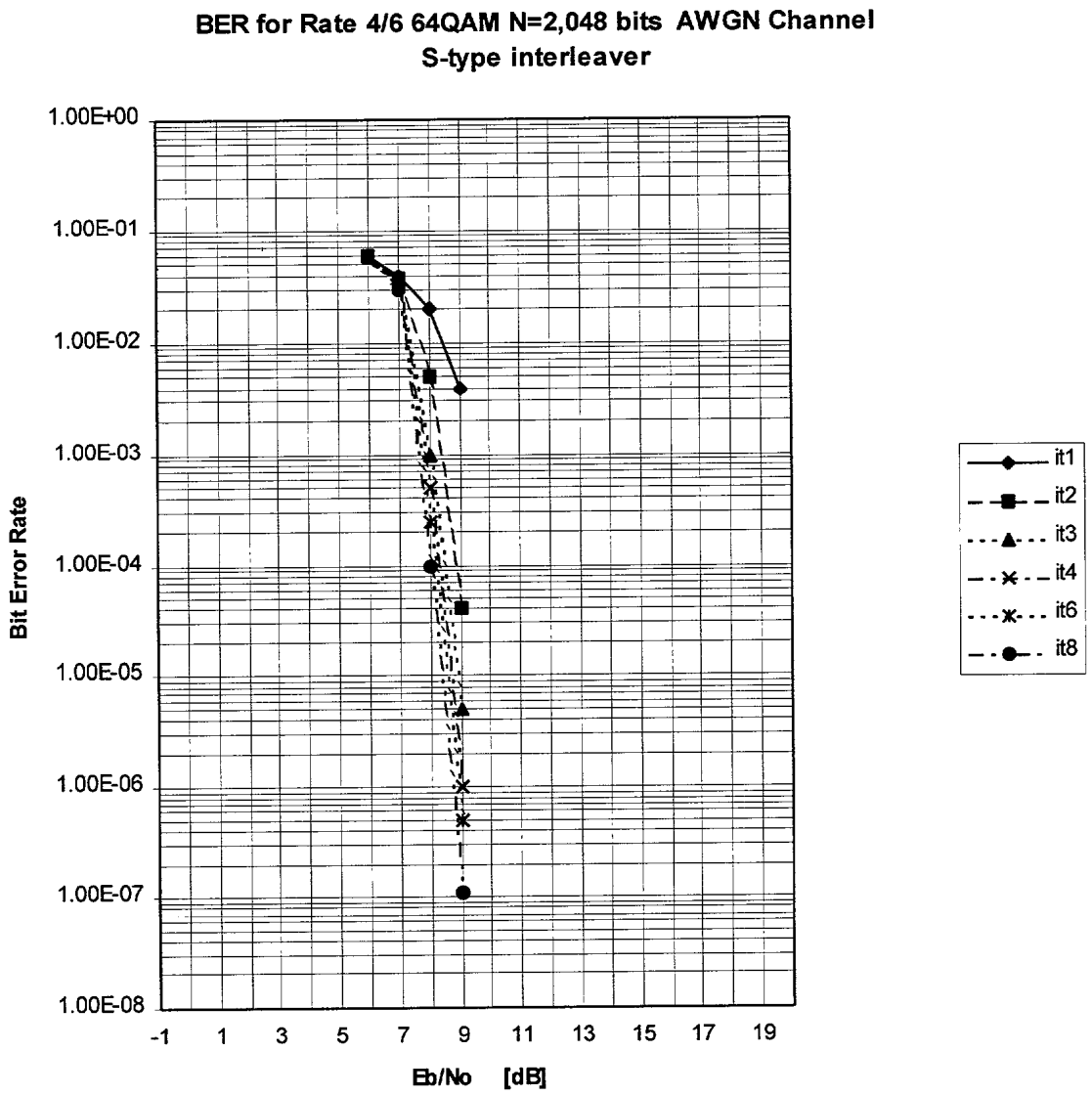

BER for Rate 4/6 64QAM N=2,048 bits AWGN Channel
S-type interleaver
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
1
3
5
7
9
11
13
15
17
19
Eb/No [dB]
it1
it2
it3
it4
it6
it8

Figure 62

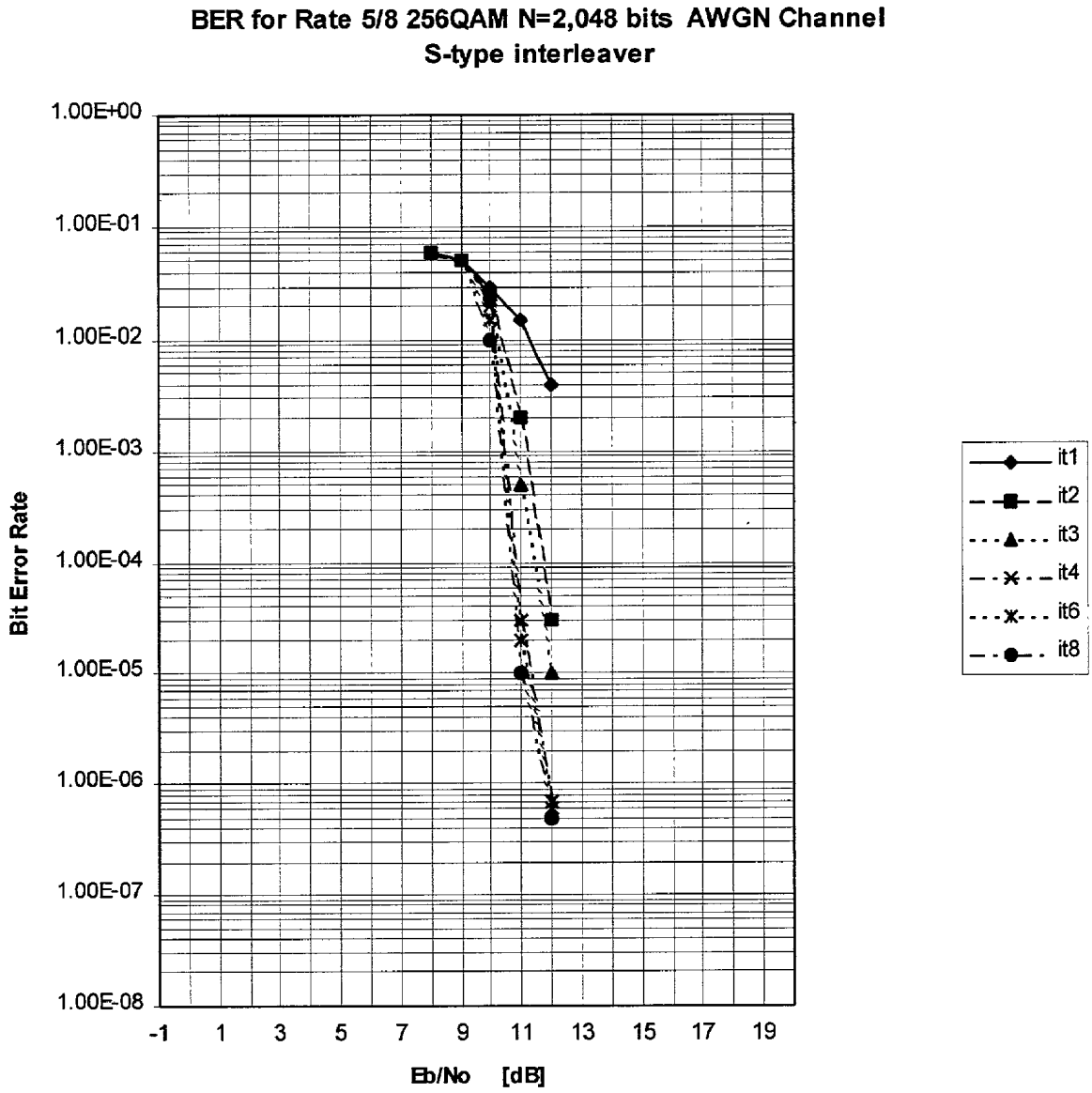

BER for Rate 5/8 256QAM N=2,048 bits AWGN Channel
S-type interleaver
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
1
3
5
7
9
11
13
15
17
19
Eb/No [dB]
it1
it2
it3
it4
it6
it8

Figure 63

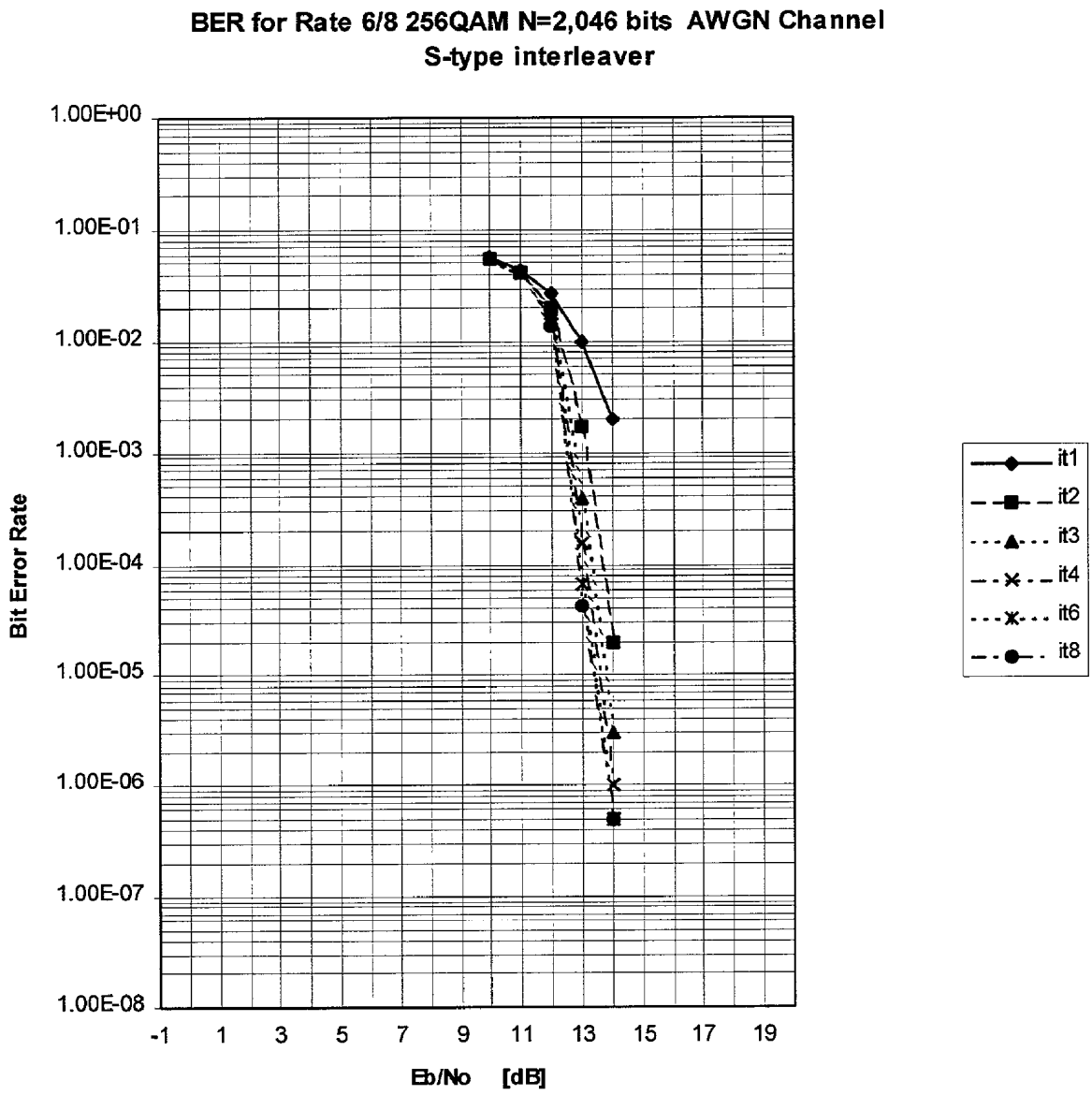

BER for Rate 6/8 256QAM N=2,046 bits AWGN Channel
S-type interleaver
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
1
3
5
7
9
11
13
15
17
19
Eb/No [dB]
it1
it2
it3
it4
it6
it8

Figure 64

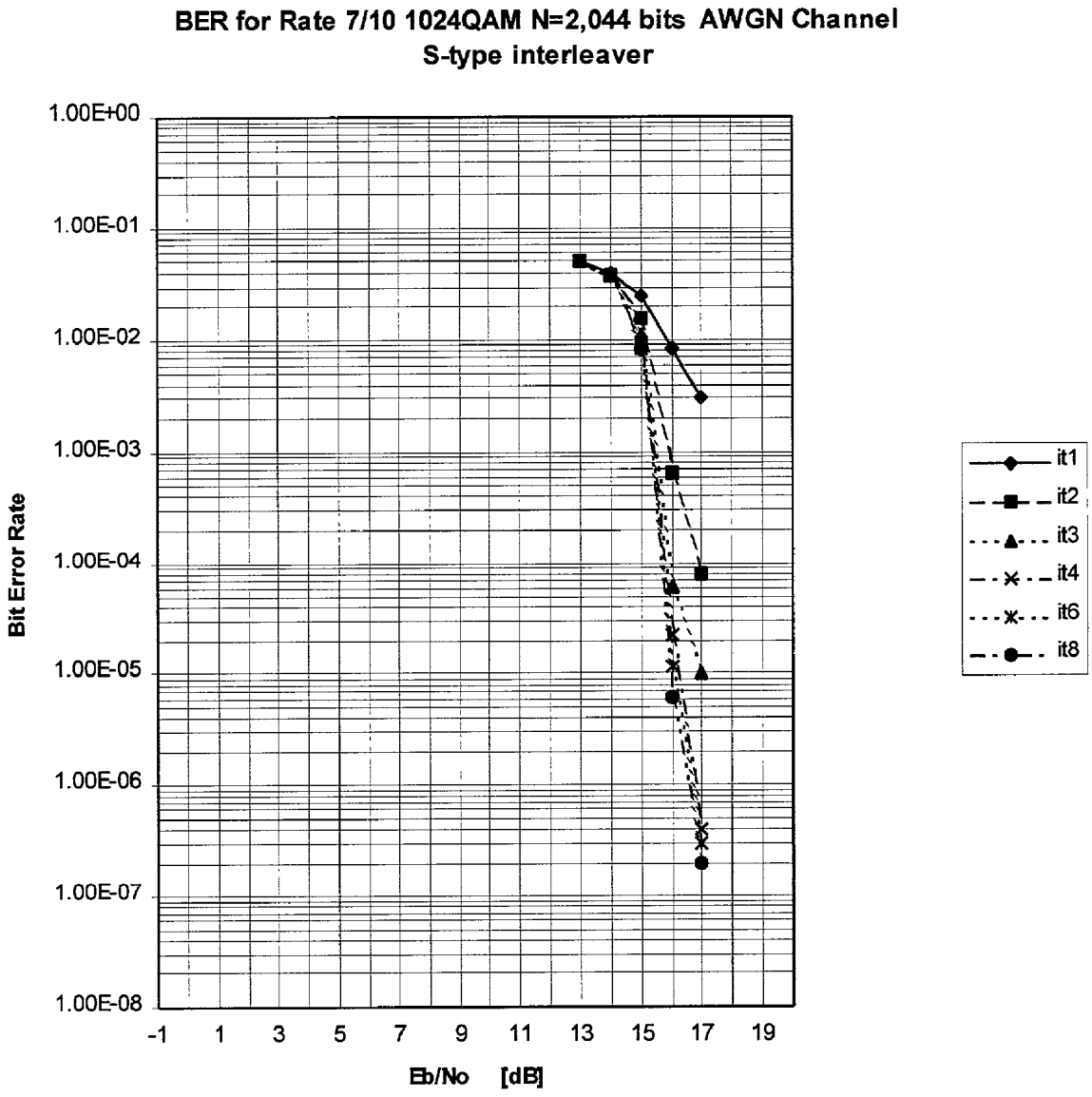

BER for Rate 7/10 1024QAM N=2,044 bits AWGN Channel
S-type interleaver
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
1
3
5
7
9
11
13
15
17
19
Eb/No [dB]
it1
it2
it3
it4
it6
it8

Figure 65

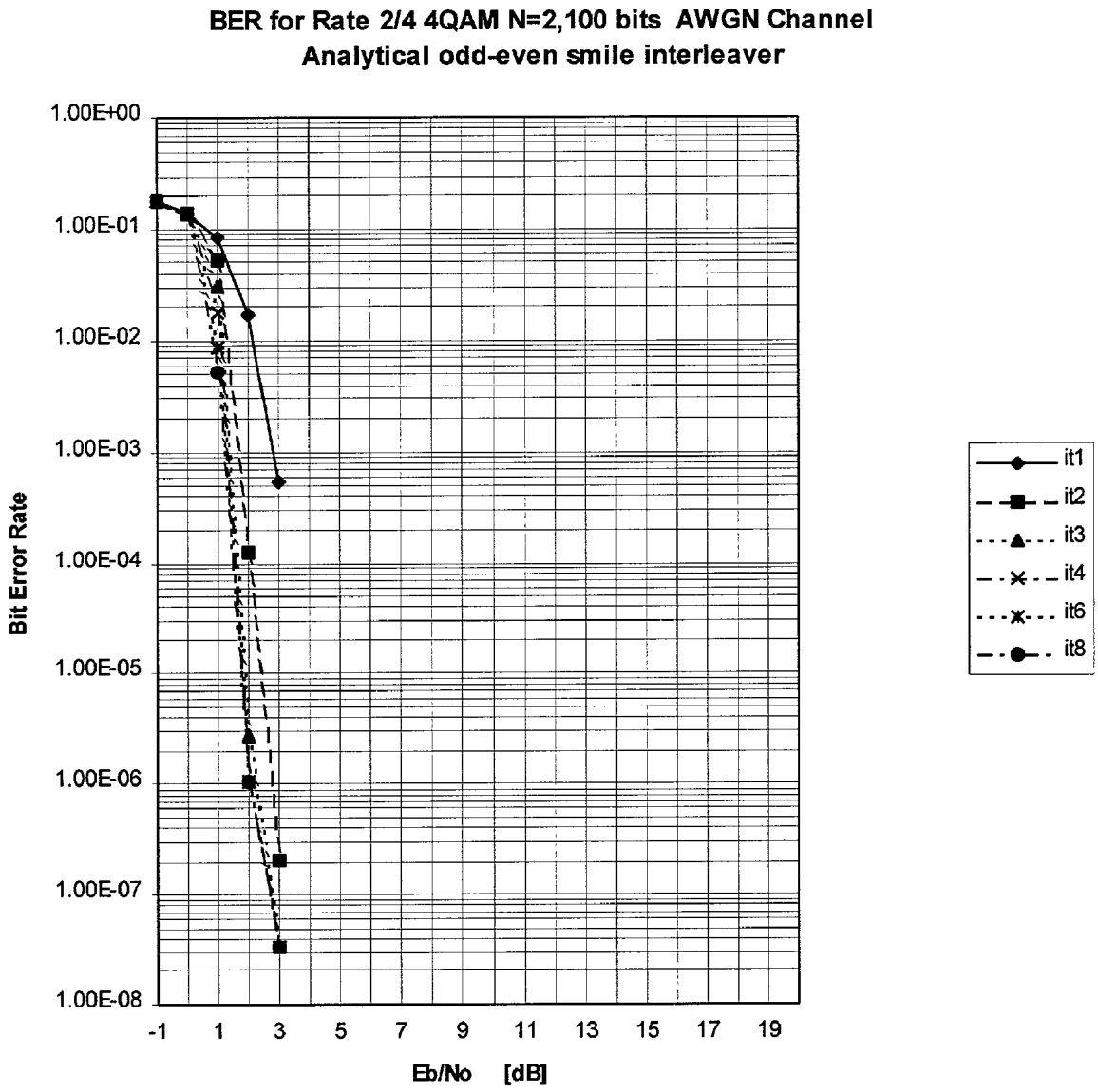

BER for Rate 2/4 4QAM N=2,100 bits AWGN Channel
Analytical odd-even smile interleaver
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
1
3
5
7
9
11
13
15
17
19
Eb/No [dB]
it1
it2
it3
it4
it6
it8

Figure 66

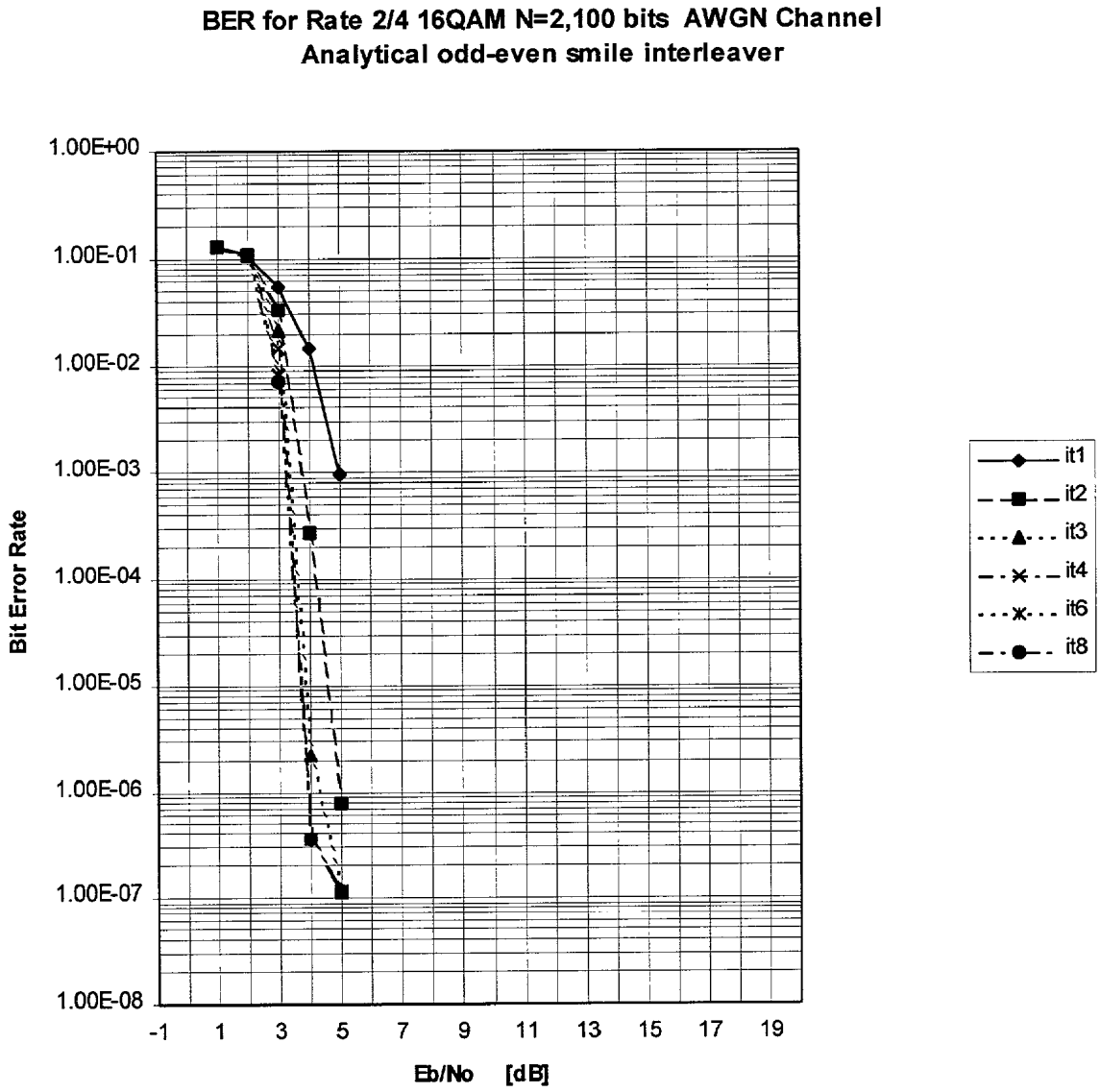

BER for Rate 2/4 16QAM N=2,100 bits AWGN Channel
Analytical odd-even smile interleaver
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
1
3
5
7
9
11
13
15
17
19
Eb/No [dB]
it1
it2
it3
it4
it6
it8

Figure 67

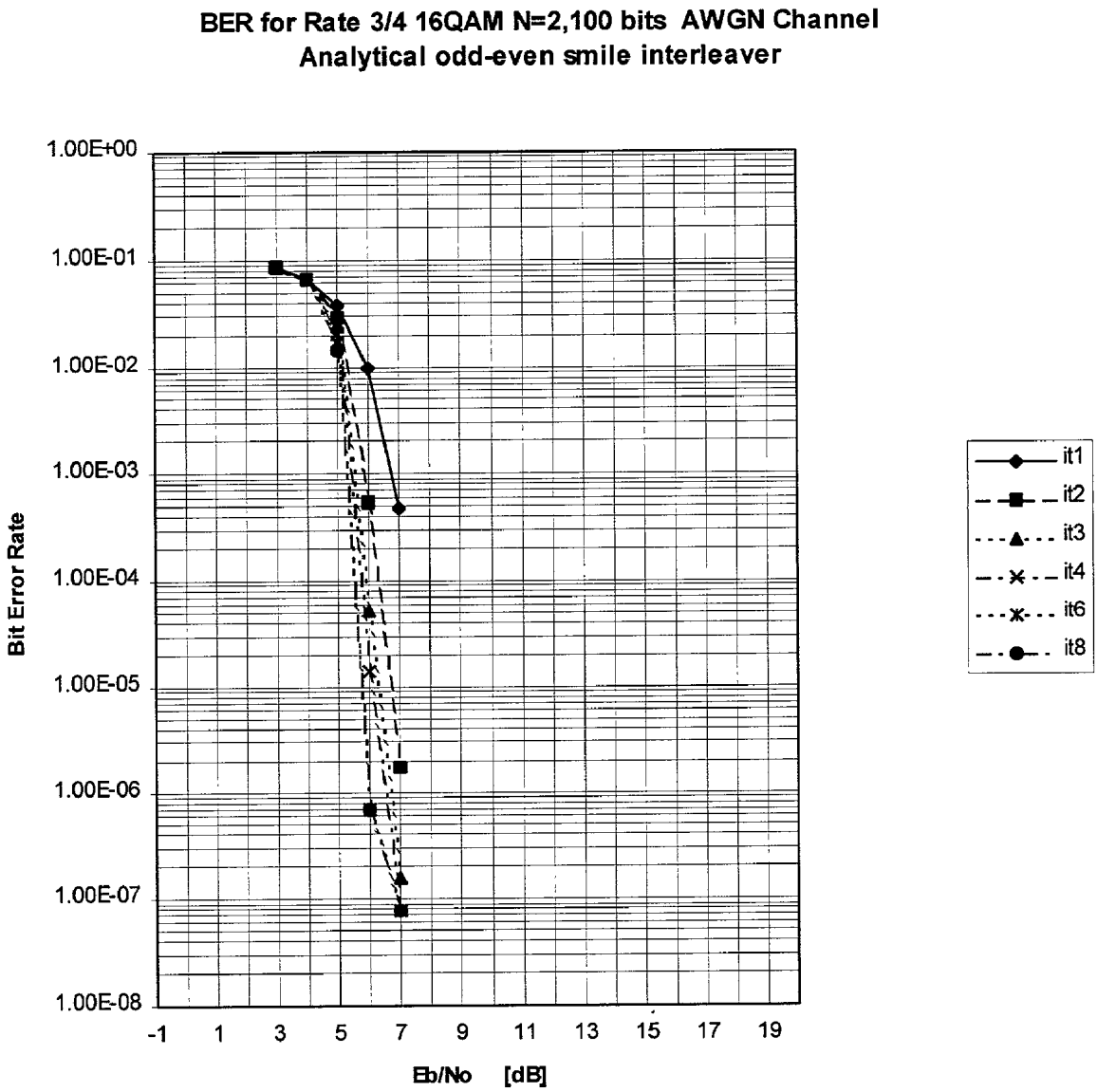

BER for Rate 3/4 16QAM N=2,100 bits AWGN Channel
Analytical odd-even smile interleaver
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
1
3
5
7
9
11
13
15
17
19
Eb/No [dB]
it1
it2
it3
it4
it6
it8

Figure 68

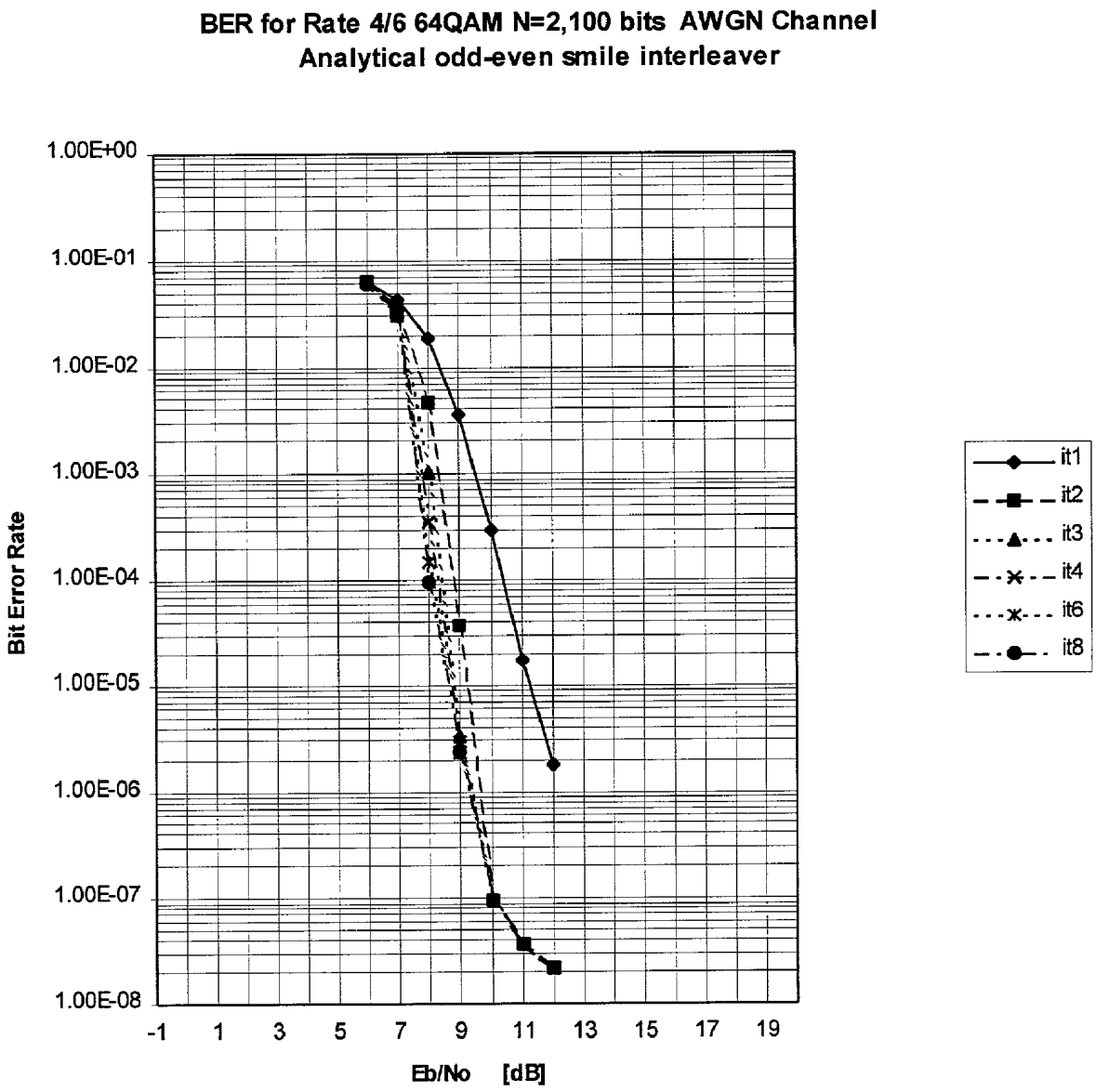

BER for Rate 4/6 64QAM N=2,100 bits AWGN Channel
Analytical odd-even smile interleaver
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
1
3
5
7
9
11
13
15
17
19
Eb/No [dB]
it1
it2
it3
it4
it6
it8

Figure 69

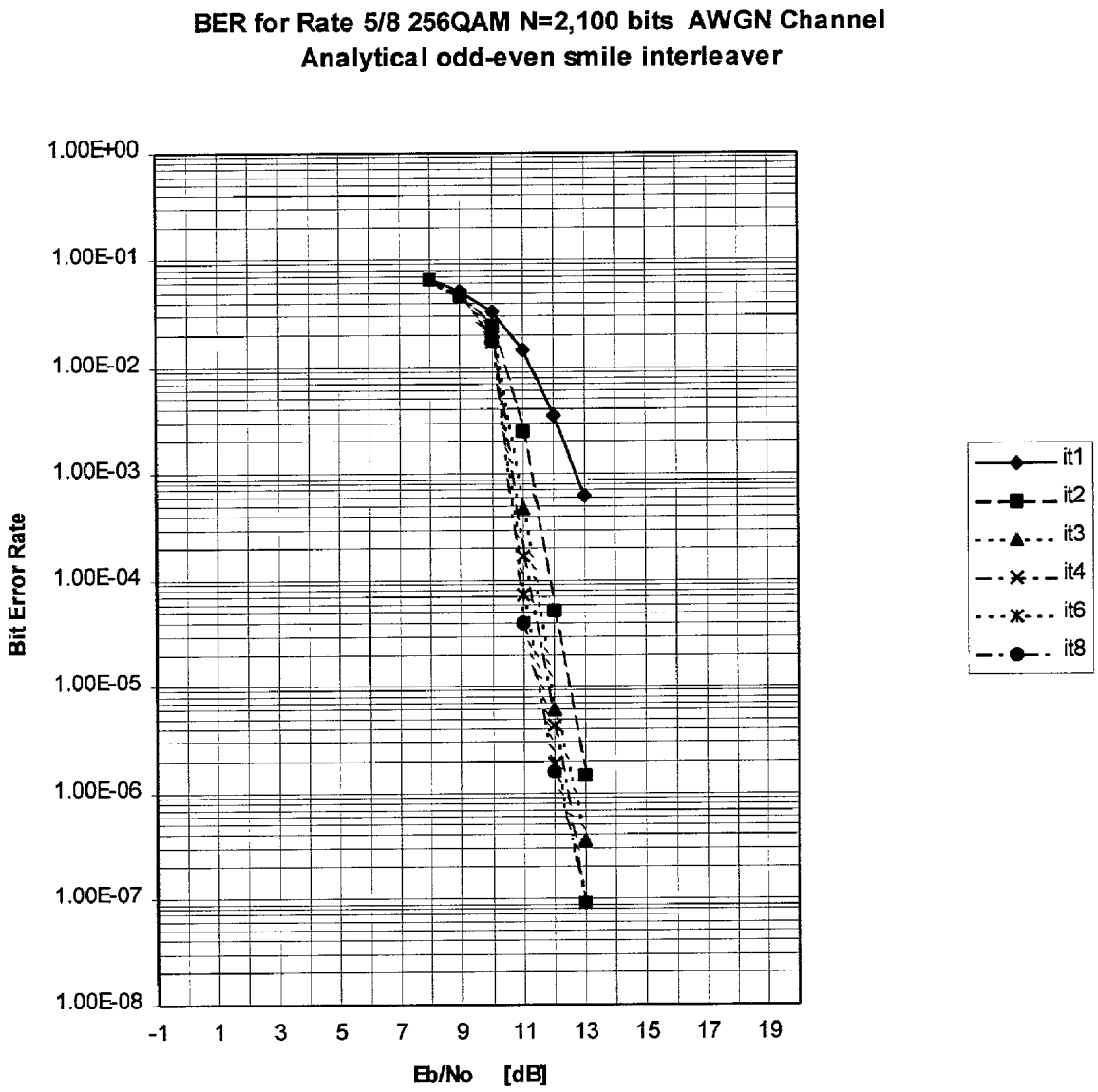

BER for Rate 5/8 256QAM N=2,100 bits AWGN Channel
Analytical odd-even smile interleaver
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
1
3
5
7
9
11
13
15
17
19
Eb/No [dB]
it1
it2
it3
it4
it6
it8

Figure 70

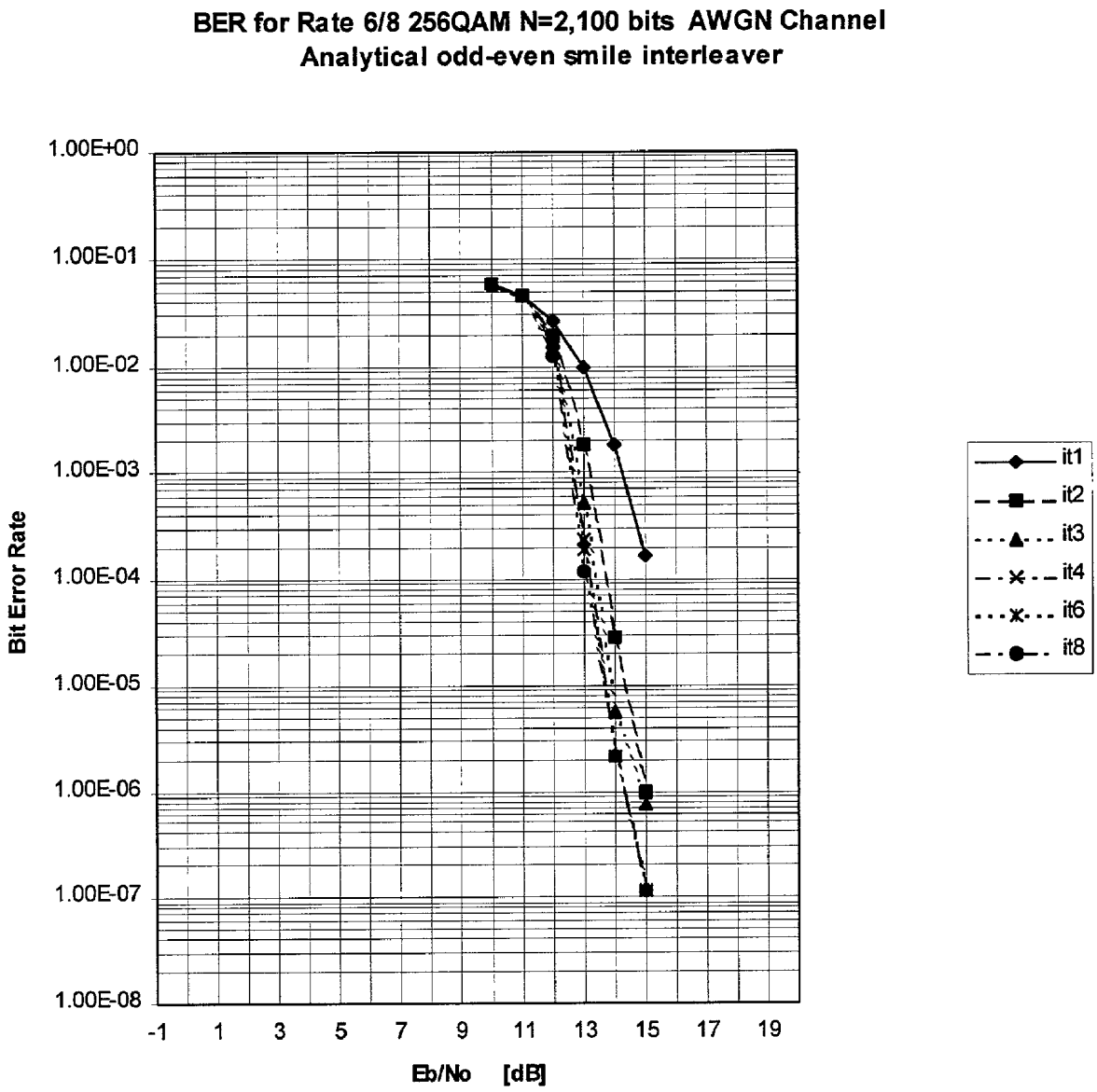

BER for Rate 6/8 256QAM N=2,100 bits AWGN Channel
Analytical odd-even smile interleaver
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
1
3
5
7
9
11
13
15
17
19
Eb/No [dB]
it1
it2
it3
it4
it6
it8

Figure 71

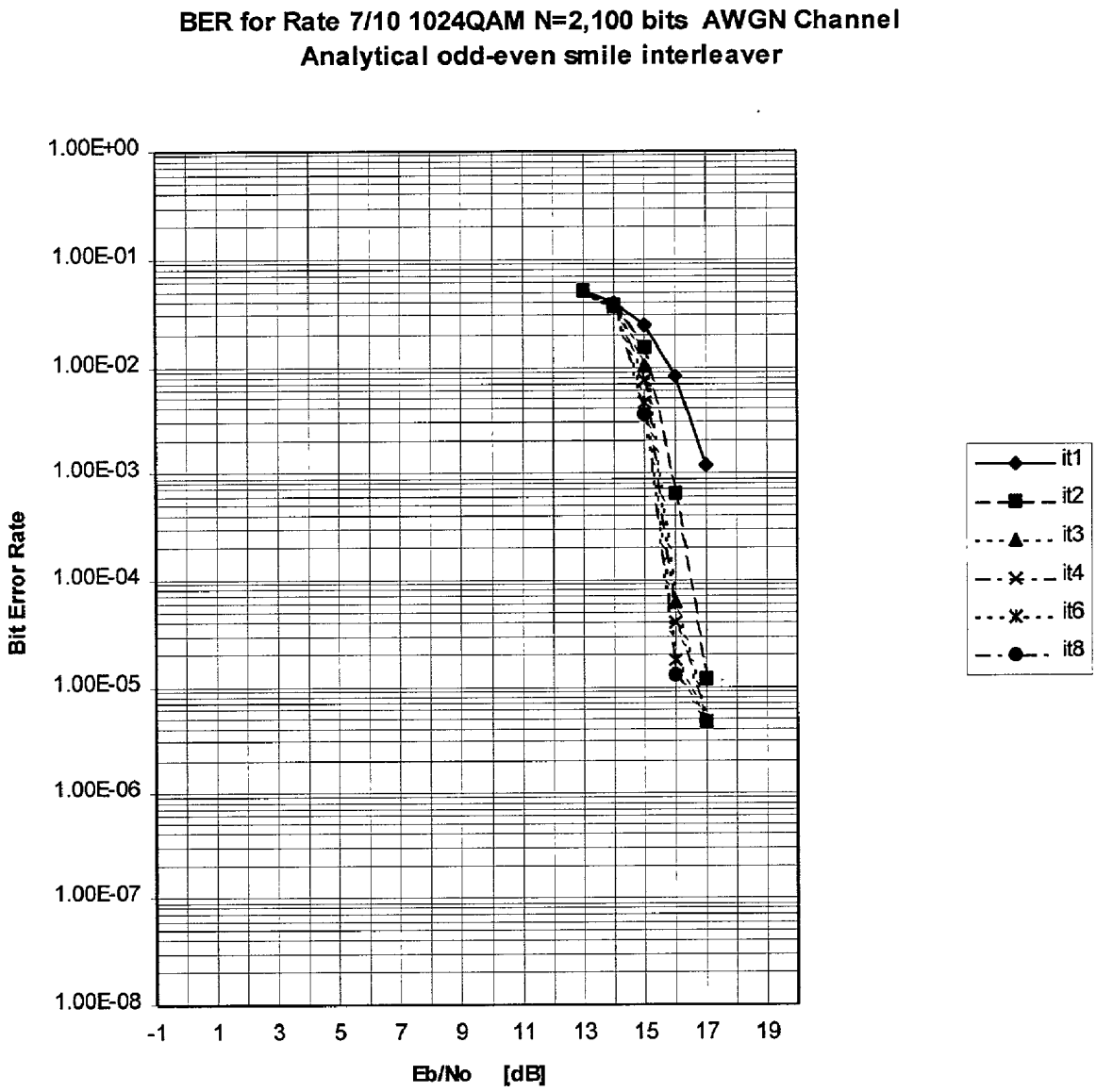

BER for Rate 7/10 1024QAM N=2,100 bits AWGN Channel
Analytical odd-even smile interleaver
1.00E+00
1.00E-01
1.00E-02
1.00E-03
1.00E-04
1.00E-05
1.00E-06
1.00E-07
1.00E-08
Bit Error Rate
-1
1
3
5
7
9
11
13
15
17
19
Eb/No [dB]
it1
it2
it3
it4
it6
it8

Figure 72

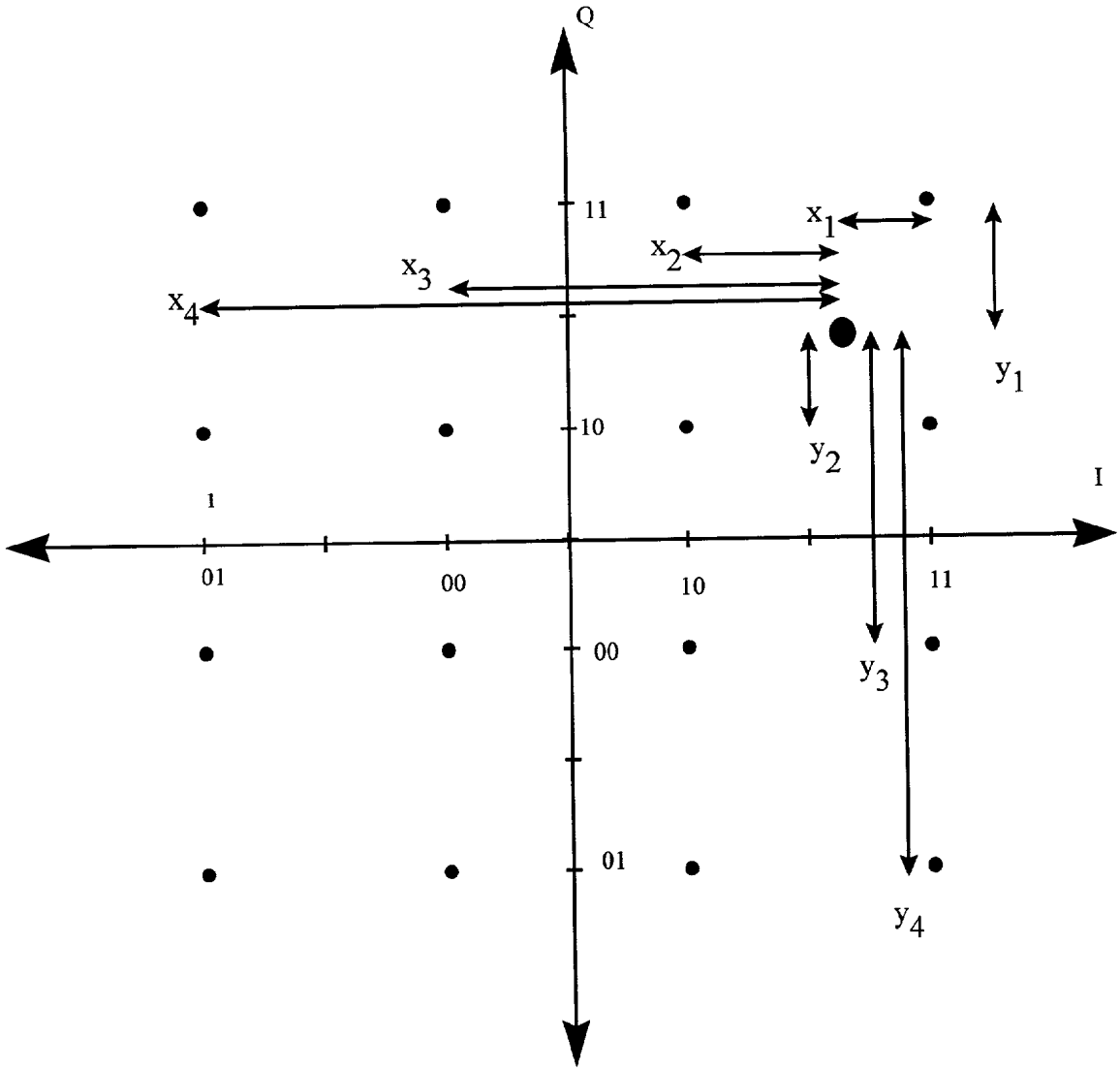


Figure 73

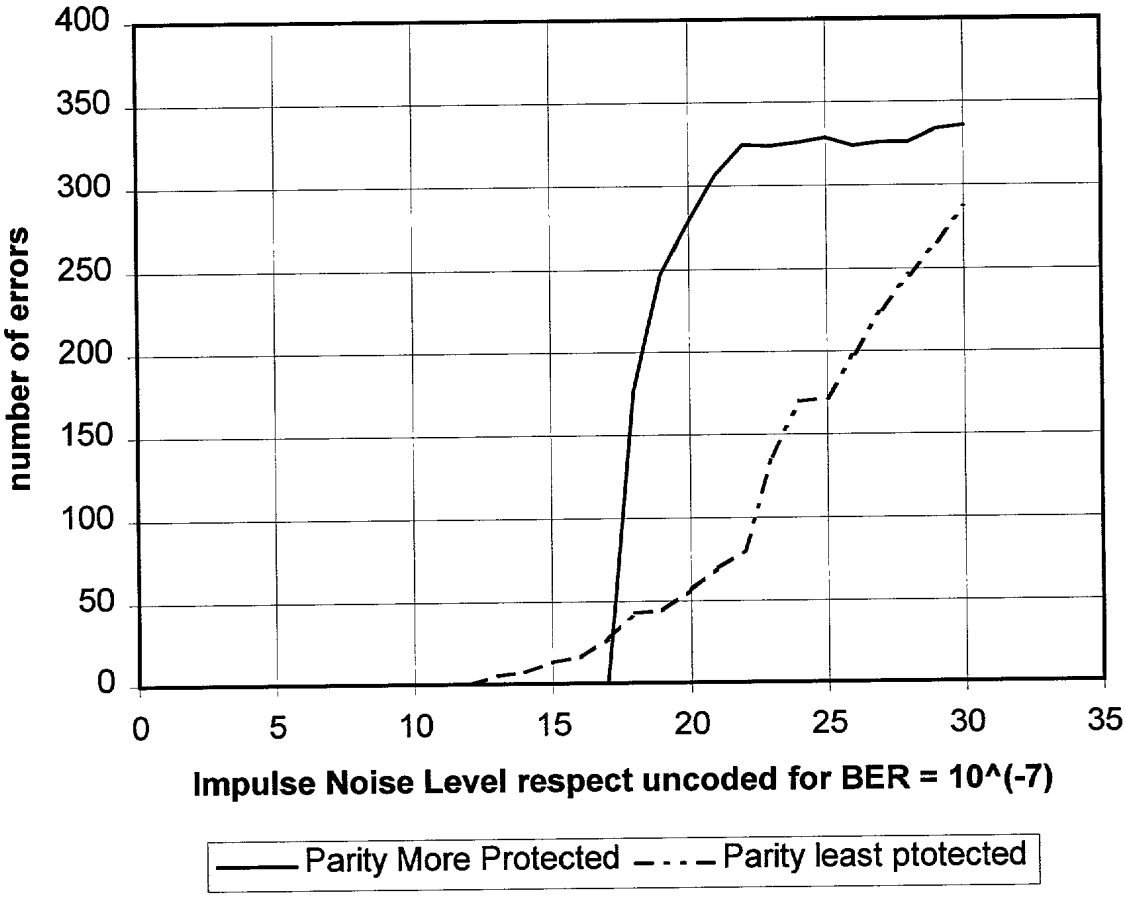

400
350
300
250
200
150
100
50
0
number of errors
0
5
10
15
20
25
30
35
Impulse Noise Level respect uncoded for BER = 10^(-7)
Parity More Protected
Parity least ptotected

Figure 74

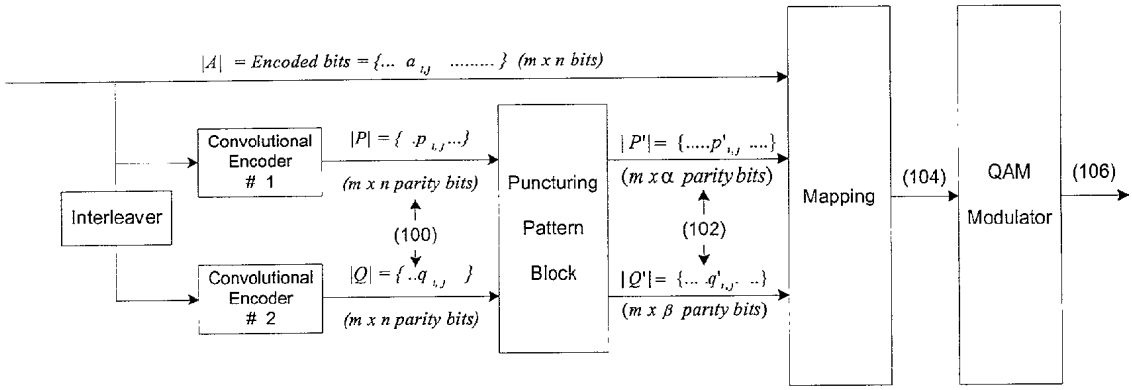


Figure 75

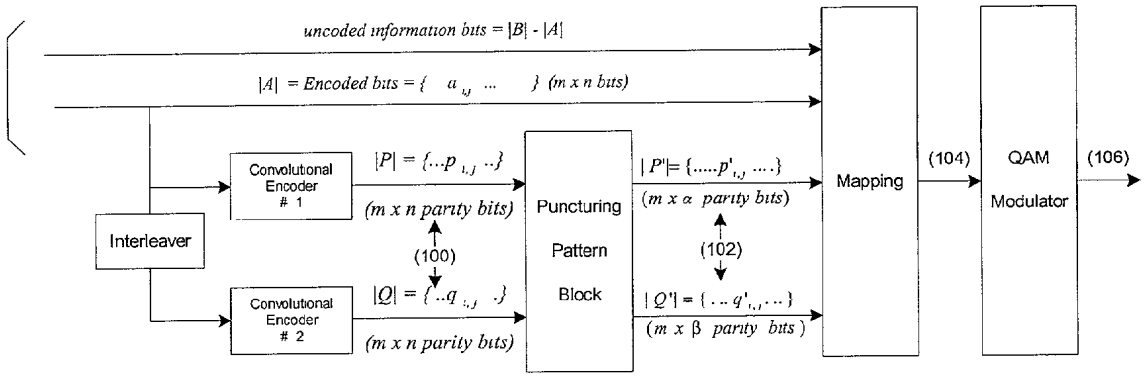


Figure 76

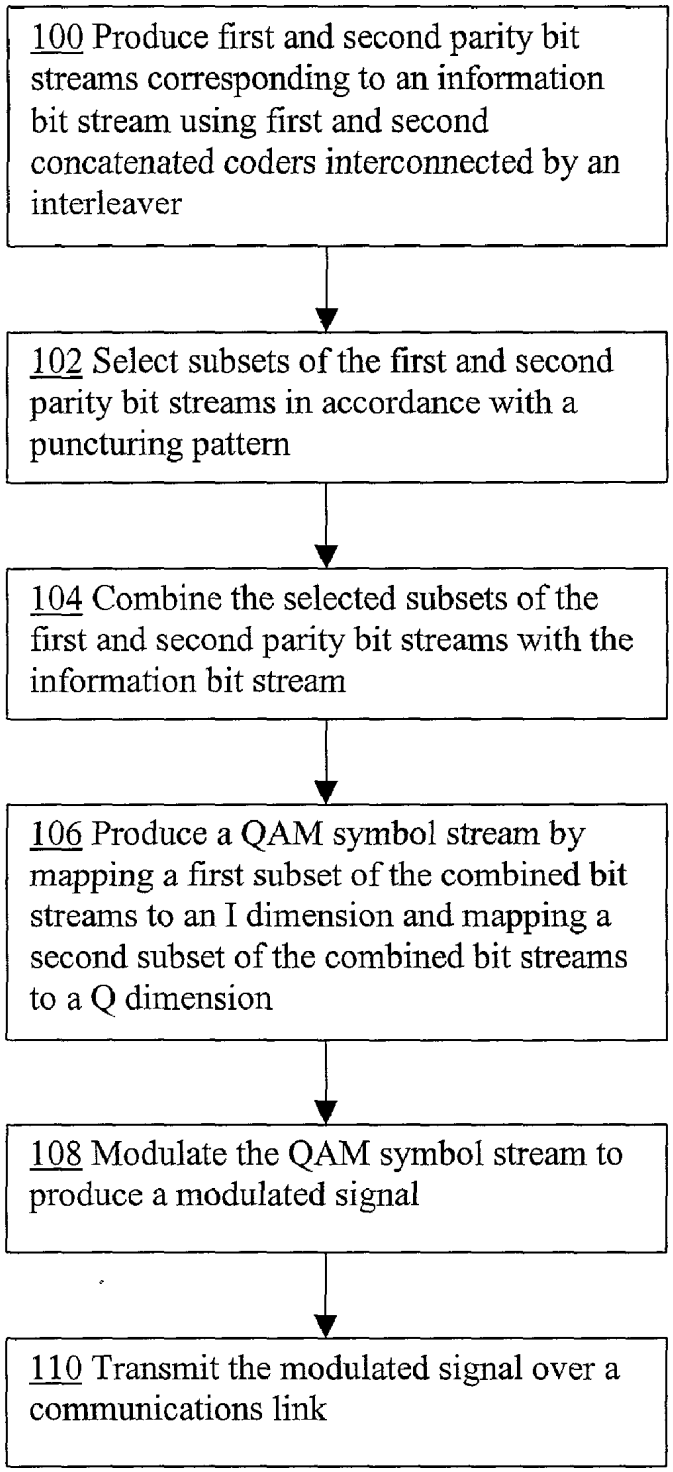

100 Produce first and second parity bit streams corresponding to an information bit stream using first and second concatenated coders interconnected by an interleaver
102 Select subsets of the first and second parity bit streams in accordance with a puncturing pattern
104 Combine the selected subsets of the first and second parity bit streams with the information bit stream
106 Produce a QAM symbol stream by mapping a first subset of the combined bit streams to an I dimension and mapping a second subset of the combined bit streams to a Q dimension
108 Modulate the QAM symbol stream to produce a modulated signal
110 Transmit the modulated signal over a communications link

Figure 77

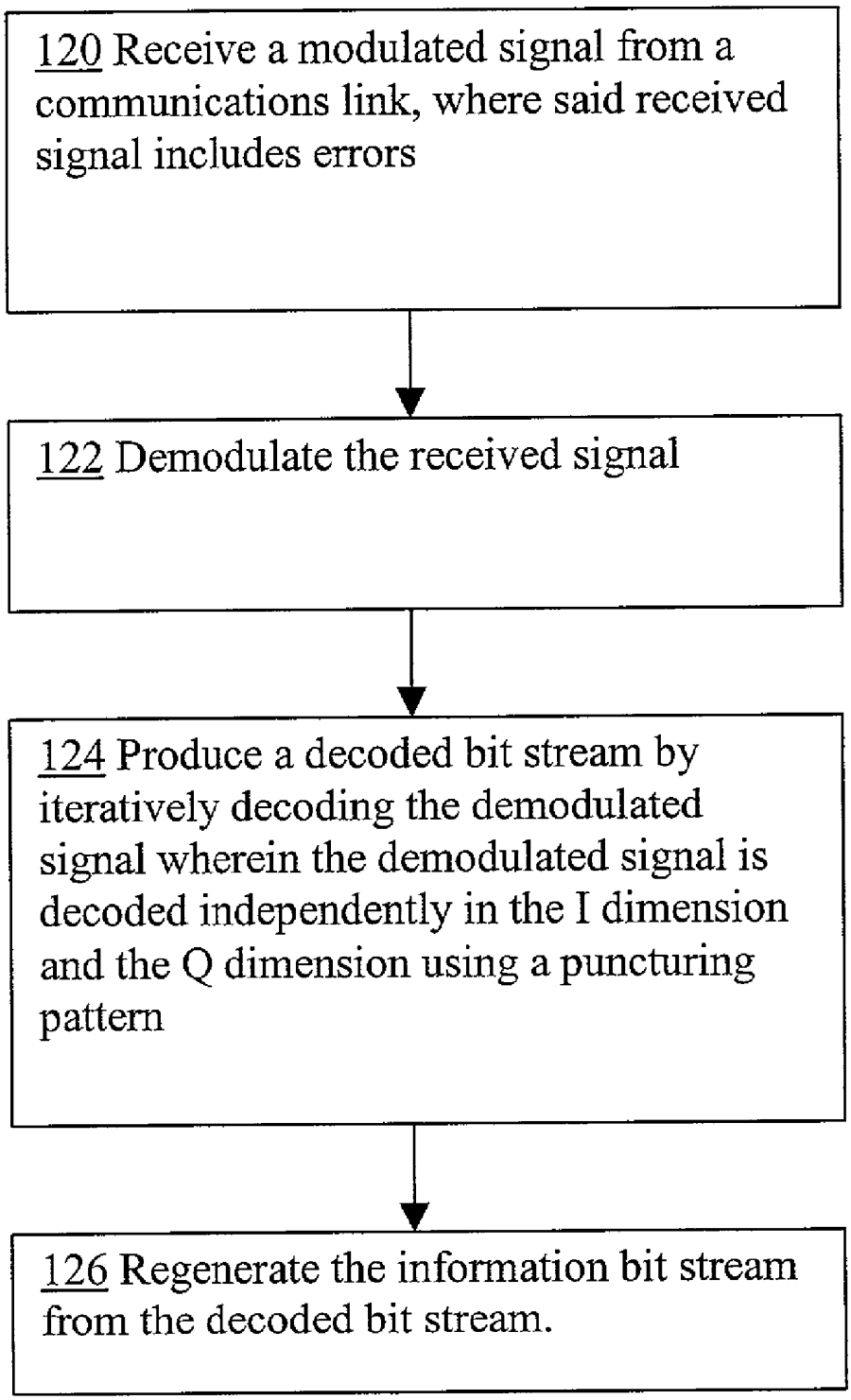

120 Receive a modulated signal from a communications link, where said received signal includes errors
122 Demodulate the received signal
124 Produce a decoded bit stream by iteratively decoding the demodulated signal wherein the demodulated signal is decoded independently in the I dimension and the Q dimension using a puncturing pattern
126 Regenerate the information bit stream from the decoded bit stream.

Figure 78

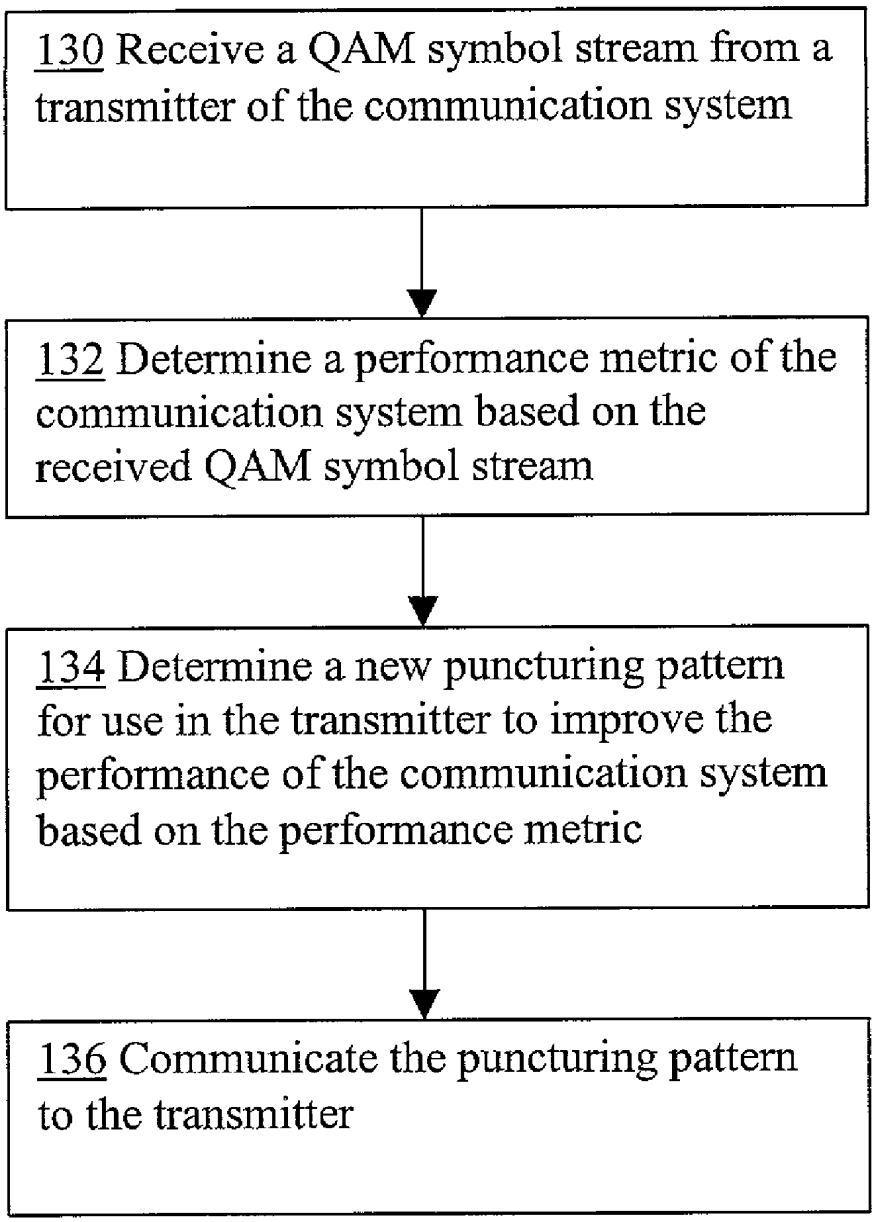

130 Receive a QAM symbol stream from a transmitter of the communication system
132 Determine a performance metric of the communication system based on the received QAM symbol stream
134 Determine a new puncturing pattern for use in the transmitter to improve the performance of the communication system based on the performance metric
136 Communicate the puncturing pattern to the transmitter

Figure 79

USE OF TURBO-LIKE CODES FOR QAM MODULATION USING INDEPENDENT I AND Q DECODING TECHNIQUES AND APPLICATIONS TO XDSL SYSTEMS

[0001] This non-provisional patent application claims the benefit under 35 U.S.C. Section 119(e) of U.S. Provisional Patent Application Ser. No. 60/200,369, filed on Apr. 28, 2000, Provisional Patent Application Ser. No. 60/248,099, filed on Nov. 13, 2000, Provisional Patent Application Ser. No. 60/242,393, filed on Oct. 20, 2000 and Provisional Patent Application Ser. No. 60/244,550, filed on Oct. 31, 2000, each of which is incorporated herein by references in its entirety.

FIELD OF THE INVENTION

[0002] The present invention relates to a techniques for coding and decoding signals used in DSL data transmission over wired and wireless systems that use Turbo Codes (TC) and any other receiver soft-decision extraction technique, such as Low Density Parity Check Codes (LDCP) (these techniques are called Turbo-like Codes), with QAM constellations using independent I and Q as modulation for error correction. The QAM constellations covered in this application are from 2 QAM up to 16384 QAM. The same technique can be use for higher order modulations. In this application we present results using an S-type and helicoidal odd-even smile analytical interleaver. An embodiment of the invention pertains particularly to xDSL systems, as a representative of DMT wired-based systems. The same technique can be used for multicarrier wireless DMT systems that using only one transmitter can send multi-channel information. The low order constellation insures a minimum acceptable data rate for very low Signal to Noise ratios that is not possible with any other known error correcting technique. The preferred mapping used is Gray mapping because of its better performance vs. Natural or Ungerboeck mapping, however the techniques disclosed herein may use Natural or Ungerboeck mapping.

BACKGROUND OF THE INVENTION

[0003] Turbo codes and other receiver soft-decision extraction techniques such as Low Density Parity Check Codes, are new and very powerful error control techniques, which allow communication very close to the channel capacity. These techniques are called as Turbo-like Codes. Low Density Parity Check Codes were introduced in 1962. Turbo codes were introduced in 1993.

[0004] A lot of research has been done in the application of Turbo-like Codes in deep space communications, mobile satellite/cellular communications, microwave links, paging, in OFDM and CDMA architectures. Turbo-like codes outperform all previously known coding schemes regardless of the targeted channel. The extra coding gain offered by these codes can be used either to save bandwidth or reduce power requirements in the link budget. Standards based in turbo codes have already been defined or are currently under investigation. For example:

[0005] Inmarsat's new multimedia service is based on turbo codes and 16 QAM that allows the user to communicate with existing Inmarsat 3 spot beam satellites from a notebook-sized terminal at 64 kbit/s.

[0006] The Third Generation Partnership Project (3 GPP) proposal for IMT-2000 includes turbo codes in the multiplexing and channel coding specification. The IMT-2000 represents the third generation mobile radio systems worldwide standard. The 3 GPP objective is to harmonize similar standards proposals from Europe, Japan, Korea and the United States.

[0007] NASA's next-generation deep-space transponder will support turbo codes and implementation of turbo decoders in the Deep Space Network is planned by 2003.

[0008] The new standard of the Consultative Committee for Space Data Systems (CCSDS) in based on turbo codes. The new standard outperforms with 1.5 to 2.8 dB the old CCSDS standard based on concatenated convolutional code and Reed-Solomon code.

[0009] The new European Digital Video Broadcasting (DVB) standard also adopted turbo codes for the return channel over satellite applications.

[0010] xDSL modems are designed to operate between a Central Office CO (or a similar point of presence) and a customer premises CPE. As such they use existing telephone network wiring between the CO and the CPE. There are several modems in this class which function in generally similar manners. Typically these modems transmit their signals above the voice band. As such, they are dependent on adequate frequency response above voice band. The modem may use a set of transmitters and receivers each at different frequencies. Alternatively, a transmitting modem may use the Inverse Discrete Fourier Transform (IDFT) that sends a signal in different subcarriers at the same time. A corresponding receiver uses the Discrete Fourier Transform (DFT) to detect the signal in the different subcarriers.

[0011] The International Telecommunication Union (ITU) has developed Recommendations G.992.1 and G.992.2 for the use of ADSL modems in the telephone network. G.992.1 allows the use of Trellis Code Modulation in the encoder of the transmitter to improve the performance of the ADSL modems. Trellis Code Modulation encodes the two least significant information bits using a convolutional encoder that provides an extra parity bit for Forward Error Correction. The extra parity bit produces an expansion of 0.5 bits per dimension in the DMT/QAM modulator. This Forward Error Correction provides an extra coding gain of around 3 dB with respect to the un-coded DMT/QAM system.

SUMMARY OF THE INVENTION

[0012] Embodiments of the present invention pertain to the use of Turbo Codes and any other receiver soft-decision extraction technique such as Low Density Parity Check Codes (these techniques are collectively referred to herein as Turbo-like Codes), for DMT systems with QAM modulation in a manner that reduces. computing requirements by treating the QAM signal as two AM modulations (one in the I dimension and one in the Q dimension) and using the probabilities of the I dimension values as an input (the probabilities of the Q dimension are not needed) to the Turbo-like Code. For non-square constellations (8 QAM, 32 QAM, 128 QAM, etc . . .), we show a method of rectangular constellations with spherical noise; a method of creating non-separable I and Q constellations by combining constitu-

ent separable I and Q constellations, as presented in Provisional Patent Application Ser. No. 09/248,099, filed on Nov. 13, 2000, that also reduces the computational complexity by the order of the square root of the number of constellation points ($O(N^{1/2})$); other methods with lower computational saving such as the diagonal regions method presented in the Provisional Patent Application Ser. No. 09/242,393, filed on Oct. 20, 2000 and Provisional Patent Application Ser. No. 09/244,550, filed on Oct. 31, 2000 can also be used.

[0013] Further embodiments of the invention improve the Turbo code, and other receiver soft-decision extraction techniques, by mapping the modulated signal in a manner providing different protection to the information bits and the parity bits as a function of the channel used and the performance needed. For Additive White Gaussian Noise (AWGN), more protection should be put in the information bits for applications with a target BER higher than 10^{-8}, and more protection is preferable put in the parity bits for applications with a target BER lower than 10^{-8}. More protection to the parity bits than to the information bits reduces the floor error of the Turbo code below 10^{-9}. For Impulsive Noise (IN) channels and high puncturing, the parity bits should be more protected than the information bits in all cases, obtaining an extra protection on the order of 25 dB.

[0014] Further embodiments of the invention employ novel puncturing patterns for each order of a QAM modulated signal, and a preferred puncturing pattern for each order is identified. The high puncturing used in the Turbo Code makes the Turbo Code and the Low Density Parity Check Code very similar in theoretical fundamentals and practical results.

[0015] Embodiments disclosed herein represent the first use of Turbo-like Codes for high order modulations techniques, up to 16384 QAM. Preferred embodiments achieve a target BER very close to the channel capacity for the respective spectral efficiency response. The schemes proposed here, are more power efficient than the trellis coded modulation schemes used traditionally with V.32.bis or V.34 standards. The higher the constellation, the more improvements that this technique provides (i.e. for 16 QAM the probabilities to compute in a classical turbo code are 16 points, with this technique the probabilities to compute are 4, so if we use 10 iterations, the computing saving is more than 40 times; for 128 QAM the computing saving is 10*1,042/186=60 times, using a method of creating non-separable I and Q constellations by combining constituent separable I and Q constellations).

[0016] Embodiments disclosed herein apply this technique to DMT xDSL modems as representative species of a wired-based system, where constellations up to 32,768 QAM are possible, and typically constellations of 1024 QAM are used. In the case of the G.992.1, this technique allows the system to work at 400 kbps with E_b/N_O below 2 dB (assuming 4 QAM and spectral efficiency of 1 bit/s/Hz 4 ksymbols/s and 100 tones). These embodiments use S-type interleavers and analytical helicoidal odd-even interleavers. The attached computer program code listing appendix presents the c code of simulation program, a program for the S-type interleaver, and a program for the analytical interleaver.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWING

[0017] **FIG. 1** shows the System Architecture for variable Symbol rates

[0018] **FIG. 2** shows the BER curves for the rate 2/4 16 QAM scheme with N=512 bits and N=32768 bits S-type

[0019] **FIG. 3** shows a Coding Scheme

[0020] **FIG. 4** shows a SRC Scheme

[0021] **FIG. 5** shows a 4 QAM constellation

[0022] **FIG. 6** shows a 2 AM constellation

[0023] **FIG. 7** shows the BER for Rate 2/6 4 QAM, N=1024 bits S-type AWGN Channel

[0024] **FIG. 8** shows the BER for Rate 2/4 4 QAM, N=1024 bits S-type AWGN Channel

[0025] **FIG. 9** shows a 8 QAM constellation

[0026] **FIG. 10** shows a 4 AM constellation

[0027] **FIG. 11** shows the BER for Rate 2/6 8 QAM, N=1024 bits S-type AWGN Channel

[0028] **FIG. 12** shows the BER for Rate 4/6 8 QAM, N=1024 bits S-type AWGN Channel

[0029] **FIG. 13** shows a 16 QAM constellation

[0030] **FIG. 14** shows the BER for Rate 1/2 16 QAM, N=272 bits (odd-even) AWGN Channel

[0031] **FIG. 15** shows the BER for Rate 1/2 16 QAM, N=256 bits S-type AWGN Channel

[0032] **FIG. 16** shows the BER for Rate 1/2 16 QAM, N=512 bits S-type AWGN Channel

[0033] **FIG. 17** shows the BER for Rate 1/2 16 QAM, N=768 bits S-type AWGN Channel

[0034] **FIG. 18** shows the. BER for Rate 1/2 16 QAM, N=1024 bits S-type AWGN Channel

[0035] **FIG. 19** shows the BER for Rate 3/4 16 QAM, N=4,096 bits S-type AWGN Channel

[0036] **FIG. 20** shows a 32 QAM constellation

[0037] **FIG. 21** shows an 8 AM constellation with Gray Mapping

[0038] **FIG. 22** shows the BER for Rate 3/5 32 QAM, N=6,144 bits S-type AWGN Channel

[0039] **FIG. 23** shows a 64 QAM constellation

[0040] **FIG. 24** shows a BER for Rate 3/6 64 QAM, N=4,096 bits S-type AWGN Channel

[0041] **FIG. 25** shows the BER for Rate 4/6 64 QAM, N=4,096 bits S-type AWGN Channel Gray Mapping

[0042] **FIG. 26** shows an 8 AM constellation with Natural Mapping

[0043] **FIG. 27** shows the BER for Rate 4/6 64 QAM, N=4,096 bits S-type AWGN Channel Natural Mapping

[0044] **FIG. 28** shows a 64 QAM constellation: First Partition Level based on $u_1=1$

[0045] **FIG. 29** shows a 64 QAM constellation: Second Partition Level based on $u_1=1$, $u_2=1$

[0046] **FIG. 30** shows a 64 QAM constellation: Third Partition Level based on $u_1=1$, $u_2=1$, $u_3=1$

[0047] **FIG. 31** shows a 64 QAM constellation: Fourth Partition Level based on $u_1=1$, $u_2=1$, $u_3=1$, $u_4=1$

[0048] **FIG. 32** shows a 64 QAM constellation: Fifth Partition Level based on $u_1=1$, $u_2=1$, $u_3=1$, $u_4=1$, $u_5=1$

[0049] **FIG. 33** shows a 64 QAM constellation: $LLR(u_1)$

[0050] **FIG. 34** shows a 64 QAM constellation: $LLR(u_2)$

[0051] **FIG. 35** shows a 64 QAM constellation: $LLR(u_3)$

[0052] **FIG. 36** shows a 64 QAM constellation: $LLR(u_4)$

[0053] **FIG. 37** shows a 64 QAM constellation: $LLR(u_5)$

[0054] **FIG. 38** shows a 64 QAM constellation: $LLR(u_6)$

[0055] **FIG. 39** shows the BER for Rate 4/6 64 QAM, N=4,096 bits S-type AWGN Channel TCM with four bits coded

[0056] **FIG. 40** shows a 64 QAM constellation with four point subsets

[0057] **FIG. 41** shows a First Partition of the 64 QAM constellation

[0058] **FIG. 42** shows a Second Partition of the 64 QAM constellation

[0059] **FIG. 43** shows a Third Partition of the 64 QAM constellation

[0060] **FIG. 44** shows a Fourth Partition of the 64 QAM constellation

[0061] **FIG. 45** shows the BER for Rate 4/6 64 QAM, N=2,048 bits S-type AWGN Channel TCM with two bits coded

[0062] **FIG. 46** shows the BER for Rate 5/6 64 QAM N=5,120 bits AWGN Channel

[0063] **FIG. 47** shows a 128 QAM constellation

[0064] **FIG. 48** shows a 16 AM constellation with Gray Mapping

[0065] **FIG. 49** shows the BER for Rate 5/7 128 QAM N=5,120 bits AWGN Channel

[0066] **FIG. 50** shows a 256 QAM constellation

[0067] **FIG. 51** shows the BER for Rate 5/8 256 QAM N=5,120 bits AWGN Channel

[0068] **FIG. 52** shows the BER for Rate 6/8 256 QAM N=6,144 bits AWGN Channel

[0069] **FIG. 53** shows a 512 QAM constellation

[0070] **FIG. 54** shows the BER for Rate 6/9 512 QAM N=6,144 bits AWGN Channel

[0071] **FIG. 55** shows a 1024 QAM constellation

[0072] **FIG. 56** shows the BER for Rate 7/10 1024 QAM N=2,044 bits AWGN Channel

[0073] **FIG. 57** shows the BER for Rate 12/14 16384 QAM N=31,200 bits AWGN Channel

[0074] **FIG. 58** shows a AWGN Channel model

[0075] **FIG. 59** shows the BER for Rate 2/4 4 QAM N=2,048 bits S-type interleaver AWGN Channel

[0076] **FIG. 60** shows the BER for Rate 2/4 16 QAM N=2,048 bits S-type interleaver AWGN Channel

[0077] **FIG. 61** shows the BER for Rate 3/4 16 QAM N=2,046 bits S-type interleaver AWGN Channel

[0078] **FIG. 62** shows the BER for Rate 4/6 64 QAM N=2,048 bits S-type interleaver AWGN Channel

[0079] **FIG. 63** shows the BER for Rate 5/8 256 QAM N=2,050 bits S-type interleaver AWGN Channel

[0080] **FIG. 64** shows the BER for Rate 6/8 256 QAM N=2,046 bits S-type interleaver AWGN Channel

[0081] **FIG. 65** shows the BER for Rate 7/10 1024 QAM N=2,044 bits S-type interleaver AWGN Channel

[0082] **FIG. 66** shows the BER for Rate 2/4 4 QAM N=2,100 bits Analytical odd-even smile interleaver AWGN Channel

[0083] **FIG. 67** shows the BER for Rate 2/4 16 QAM N=2,100 bits Analytical odd-even smile interleaver AWGN Channel

[0084] **FIG. 68** shows the BER for Rate 3/4 16 QAM N=2,100 bits Analytical odd-even smile interleaver AWGN Channel

[0085] **FIG. 69** shows the BER for Rate 4/6 64 QAM N=2,100 bits Analytical odd-even smile interleaver AWGN Channel

[0086] **FIG. 70** shows the BER for Rate 5/8 256 QAM N=2,100 bits Analytical odd-even smile interleaver AWGN Channel

[0087] **FIG. 71** shows the BER for Rate 6/8 256 QAM N=2,100 bits Analytical odd-even smile interleaver AWGN Channel

[0088] **FIG. 72** shows the BER for Rate 7/10 1024 QAM N=2,100 bits Analytical odd-even smile interleaver AWGN Channel

[0089] **FIG. 73** shows a 16 QAM constellation with indication of x1, x2, x3, x4 and y1, y2, y3, y4

[0090] **FIG. 74** shows the performance of the 6/4 rate 64 QAM for the case that the parity bits are more protected and for the case that the parity bits are least protected

[0091] **FIG. 75** shows a detailed explanation of the encoding and puncturing patterns in a case where all the information bits are encoded.

[0092] **FIG. 76** shows a detail explanation of the encoding and punctured procedures in a case where a subset of information bits are encoded.

[0093] **FIG. 77** shows a process in a transmitter in accordance with embodiments of the invention, and

[0094] **FIG. 78** shows a process in a receiver in accordance with embodiments of the invention.

DETAILED DESCRIPTION OF THE INVENTION

[0095] We herby incorporate by reference the following references, which are representative of the conventional knowledge in the field of the invention:

[0096] 1. R. G. Gallager, "Low Density Parity-Check Codes", IRE Trans. on Information Theory, pp. 21-28, 01/62.

[0097] 2. C. Berrou, V. Glavieux and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes", ICC 1993, Geneva, Switzerland, pp. 1064-1070, May 1993.

[0098] 3. H. Feldman and D. V. Ramana, "An introduction to Inmarsat's New Mobile Multimedia Service", The Sixth International Mobile Satellite Conference, Ottawa, pp. 226-229, June 1999.

[0099] 4. P. Chaudhury, W. Mohr and S. Onoe, "The 3GPP Proposal for IMT-2000", IEEE Communications Magazine, vol. 37, no 12, pp.72-81. December 1999.

[0100] 5. 3 GPP Standard "Multiplexing and channel coding: TS 25.212"

[0101] 6. C. D. Edwards, C. T. Stelzried, L. J. Deutsch and L. Swanson, "NASAS's deep-Space Telecommunications Road Map" TMO Progress Report 42-136, JPL, Pasadena, Calif. USA. PP. 1-20 February 1999.

[0102] 7. R. Pyndiah, A. Picard and A. Glavieux "Performance of Block Turbo Coded 16 QAM and 64 QAM modulations" Procedings of Globecom 95 pp.1039-1043.

[0103] 8. Rauschmayer, Dennis J. "ADSL/VDSL Principles", Macmillan Technical Publishing, 1999.

[0104] 9. ITU G.992.1 "ADSL Transceivers", ITU, 1999.

[0105] 10. ITU G.992.2 "Splitterless ADSL Transceivers". ITU 1999.

[0106] 11. ITU I.432 "B-ISDN user-network interface-physical layer specification", ITU, 1993.

[0107] 12. Benedetto, Divsalar, Montorsi and F. Pollara, "Serial Concatenation of Interleaved Codes: Performance Analysis, Design, and Iterative Decoding", The Telecommunications and Data Acquisition Progress Report 42-126, Jet Propulsion Laboratory, Pasadena, Calif., pp. 1-26, Aug. 15, 1996.

[0108] 13. Benedetto, Divsalar, Montorsi and F. Pollara, "A Soft-Output Maximum A Posteriori (MAP) Module to decode parallel and Serial Concatenated Codes", The Telecommunications and Data Acquisition Progress Report 42-127, Jet Propulsion Laboratory, Pasadena, Calif., pp. 1-20, Nov. 15, 1996.

[0109] 14. L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," IEEE Transactions on Information Theory, pp. 284-287, March 1974.

[0110] 15. Divsalar and F. Pollara, "Turbo Codes for PCS Applications", Proceedings of ICC'95, Seattle, Wash., pp. 54-59, June 1995.

[0111] 16. D. Divsalar and F. Pollara, "Multiple Turbo Codes", Proceedings of IEEE MILCOM95, San Diego, Calif., Nov. 5-8, 1995.

[0112] 17. D. Divsalar and F. Pollara, "Soft-Output Decoding Algorithms in iterative Decoding of Turbo Codes," The Telecommunications and Data Acquisition Progress Report 42-124, Jet Propulsion Laboratory, Pasadena, Calif., pp. 63-87, Feb. 15, 1995.

[0113] 18. Juan Alberto Torres, Frederic Hirzel and Victor Demjanenko, "Forward Error Correcting System With Encoders Configured in Parallel and/or Series", International Patent Application Ser. No. PCT/US99/17369 filed on Jul. 30, 1999.

[0114] 1. Introduction

[0115] Error correction can be used in communication systems to improve the amount of data that can be accurately communicated through a communication channel. Digital Subscriber Line (XDSL) technology is an example of communication using error correction. In xDSL, a transmitter and a receiver communicate over a copper twisted pair using Discrete Multi-Tone (DMT) modulation. In particular, ADSL (Asymmetric Digital Subscriber Line) provides multiple carriers in the frequency band between 20 kHz and 1.1 MHz, at intervals of 4.1325 kHz. Each carrier is modulated using quadrature amplitude modulation (QAM), in which a data value is represented by the phase and magnitude of the carrier. A given phase/magnitude combination is referred to as a "symbol", and the set of symbols that can be transmitted on a carrier is referred to as a "constellation." Typically a constellation contains 2^n symbols; in other words, each symbol represents n bits of data. An ADSL channel is theoretically capable of a 15 bit constellation (2^{15} discrete symbols). However, in practice, noise on the line will cause the magnitude and phase of transmitted symbols to vary slightly, making it uncertain which symbol a received phase/magnitude combination corresponds to. This typically requires the use of a constellation that provides more space between symbols so that variations in phase and magnitude create less uncertainty. As a result, a given carrier will typically use a constellation of less than 15 bits.

[0116] A type of error correction that is used in xDSL and other communication systems is called Forward Error Correction (FEC). Forward error correction involves transmitting one or more error correction bits, typically referred to as "parity bits," in conjunction with information bits, so that the information bits represented in the signal can be determined solely from the received signal itself using the parity bits, thereby eliminating the need for retransmission. One simple conventional form of forward error coding involves providing a parity bit that represents the least significant bit of a sum of all information bits. Upon receiving the bit string, if the least significant bit of the sum of the information bits matches the parity bit, the received bits are assumed to be correct.

[0117] In a more complex forward error coding system referred to as trellis coding, some input bits are coded by a convolutional coder that generates a parity bit for each group of n input bits, and the resulting group of n+1 output bits are used by QAM modulation to produce a signal for a digital to analog converter. The addition of the parity bit effectively doubles the number of symbols in the constellation, and the

value of the parity bit effectively places a given symbol in one of two subsets of the constellation, each of which contains every other symbol of the full constellation. As a result, all symbols of the constellation are available for use, but each individual symbol is discriminated within a subset of the constellation in which the symbols are twice as far apart as they would be in the full constellation.

[0118] Presently some xDSL modems use a forward error correction method known as Trellis Code Modulation. Trellis coded modulation (TCM) proposed by Ungerboeck in 1982 is now a well-established technique in digital communications. Since its first appearance, TCM has generated a continually growing interest, concerning its theoretical foundations as well as its numerous applications, spanning high-rate digital transmission over voice circuits, digital microwave radio relay links, and satellite communications.

[0119] Turbo-like Codes represent a more recent development in the coding research field (1993), which have generated a large interest in the coding community. Turbo Codes are Parallel Concatenated Convolutional Codes (PCCC) whose encoder is formed by two (or more) constituent systematic encoders joined through one or more interleavers. The input information bits feed the first encoder and, after having been scrambled by the interleaver, they enter the second encoder.

[0120] This document describes the performance of Turbo-like Codes used with square and non-square high order QAM modulation/demodulation schemes, from 4 QAM up to 16384 QAM. For higher order modulation the same procedure is applicable. This document also describes the application of this modulation schemes to xDSL modems presented in the ITU Recommendations G.992.1 and G.992.2. The use of squared and non-squared QAM constellations with novel puncturing patterns simplifies the design of the equalizer, and for this reason it is possible to utilize squared and non-squared QAM constellations with independent I and Q dimensions using blind equalizers. LDPC codes and other receiver soft-decision extraction techniques may be employed in a similar manner. The different puncturing patterns allow the system to work with low signal-to-noise signals (in the order of 1.25 dB) at the expense of reduced data rates. The Turbo-like Coding and modulation schemes disclosed herein are designed to accommodate a variable symbol rate, which could be adapted to a particular channel characteristic.

[0121] The performance of the coding and the modulation schemes disclosed herein are estimated through computer simulations, for different interleaver sizes. Bit probabilities are estimated from the received QAM symbol. Trade offs between power and bandwidth efficiency are also addressed.

[0122] 1.1 Objectives

[0123] The objectives of using higher order modulations and Turbo-like Codes with independent I and Q are:

[0124] 1. To define a flexible architecture to allow variable symbol rates function of the channel quality.

[0125] 2. To estimate the performance of 4 QAM, 8 QAM, 16 QAM, 32 QAM, 64 QAM, 128 QAM, 256 QAM 512 QAM, 1024 QAM and 16384 QAM schemes for different interleaver sizes and Additive Gaussian noise (AWGN) and Impulse noise channel (IN).

[0126] 3. To find the mapping technique to limit the constellation expansion to at most 0.5 bit/dimension.

[0127] 4. To evaluate the performance for higher order modulation schemes in an AWGN channel and in an Impulse Noise Channel.

[0128] 1.2 System Parameters

[0129] The system parameters are summarized in Table 1.

TABLE 1

System Parameters

Parameter	Value
Target bit error rate (BER)	10^{-4}, 10^{-5}, 10^{-6}, and 10^{-7}
Target Eb/No	As low as possible
Interleaver block sizes	variable to 14 DMT symbols for a latency of 10 ms.
Modulation	QAM
Coding rate	From 1/3 and up
Maximum Symbol Rate	1,024 ksym/s
Minimum Symbol Rate	128 ksym/s
Symbol Rate Step	4 ksym/s
Maximum Information Rate	6,144 kbit/s
Minimum Information Rate	160 kbit/s
Information Rate Step	32 kbit/s
Channel	AWGN and IN

[0130] 2. System Architecture.

[0131] 2.1 Introduction

[0132] The coding and modulation schemes of the embodiments disclosed herein are designed to accommodate variable symbol rates using the same encoder and decoder structure. Therefore, depending on the channel quality, the same data rate can be transmitted in a variable bandwidth or a different data rate can be transmitted, using the same structure but different puncturing patterns and mapping.

[0133] **FIG. 1** illustrates functional architectures of a transmitter and a receiver in a communication system. At the transmitter, incoming information bits, d, are encoded by a turbo-like coder such as a rate 1/3 systematic encoder or linear parity bit generator matrix H. The turbo-like encoder outputs the information bits d and a parity bit p and parity bit q corresponding to each information bit. The puncturing block performs a puncturing function that determines the actual coding rate of the parity generator matrix H. The puncturing function is a simple deletion of some parity bits from the p and q streams which results in creation of subsets of the p and q bit streams. No puncturing is applied to the information bits d. A mapping function then combines the information bit stream with the subsets of the parity bit streams selected by the puncturing pattern and produces a QAM symbol stream by mapping a first subset of the combined bit streams to an I dimension and mapping a second subset of the combined bit streams to a Q dimension. Each puncturing pattern and mapping is identified using a unique word (UW). The puncturing pattern and mapping may be varied adaptively. For example, transmission may initially employ the most powerful coding scheme that can be used in the available bandwidth. As a function of the Channel State Information (with the values of gi and bi for each tone for the case of a G.992.1 or G.992.2 modem), the puncturing and mapping can be adapted to reduce the transmitted signal band width to a value that allows com-

munication at the desired BER. The receiver is provided with an identification of the puncturing pattern and mapping.

[0134] At the receiver, after synchronization with the transmitter, a particular configuration of the bit estimator block corresponding to the puncturing pattern and mapping of the transmitter is established. The function of the bit estimator block is to compute the log-likelihood ratios (LLR) for each bit of the received QAM symbol in the Turbo-like Code. The outputs from this block are punctured streams of the estimate $\bar{d}$, $\bar{p}$ and $\bar{q}$ bits that are provided to a decoder. The decoder does not need to know anything about the transmitter mapping or modulation scheme. A maximum a posteriori (MAP) algorithm is used in the decoder to produce decoded bits d. In the case of LDPC the output d is the decoded bit d or an error flag indicating that the decoding process has an error.

[0135] 2.2 Power vs. Bandwidth.

[0136] The architecture of **FIG. 1** is flexible enough to select the most bandwidth efficient modulation given the available signal-to-noise ratio available for a particular channel. An example is shown in Table 2 for an information data rate of 2,048 kbps. The most powerful coding scheme that could be used is the rate 2/6 4 QAM. This scheme can achieve a BER of 10^{-6} at approximately E_b/N_O=1.2 dB and requires a symbol rate of 3,072 ksym/s. If the transmission occurs over a less noisy channel, them the bandwidth can be reduced by using higher spectral efficiency schemes. The minimum symbol rate that can be achieved is 292 ksym/s.

TABLE 2

Trade Off Power for Bandwidth for 2,048 kbit/s Information Data Rate

Spectral efficiency η [bits/s/Hz]	Coding Rate and Modulation	Symbol Rate [ksym/s]	E_b/N_O [dB]
2/3	2/6 and 4 QAM	3072	1.2
1	2/4 and 4 QAM	2048	2.1
2	2/4 and 16 QAM	1024	3.9
3	3/4 and 16 QAM	682	6.0
4	4/6 and 64 QAM	512	8.8
5	5/8 and 256 QAM	408	11.5
6	6/8 and 256 QAM	342	13.8
7	7/10 and 1024 QAM	292	16.4

[0137] The turbo decoder has to operate at a maximum speed of 6,144 kbit/s. Speeds of 20,048 kbps can be achieved with the current FPGA technology

[0138] 2.3 Optimization for Specific Applications

[0139] Another advantage of this architecture is that the system is re-configurable in order to achieve the Quality of Service requested by a particular application. For example, the LDPC code or the turbo code modem could be configured to operate in two modes:

[0140] Mode A: minimum transfer delay for delay sensitive applications.

[0141] In this mode, if a maximum transfer delay is defined, the block for the parity matrix size or the interleaver size could be changed in function of the data rate (to an integer number multiple of DMT symbols).

[0142] Mode B: maximum coding gain for data transfer applications.

[0143] In this mode, a fixed block parity matrix size or an interleaver size (i.e. 65,536 bits) could be used to achieve the best performance. A significant coding gain of almost 2 dB could be achieved at a BER of 10^{-6} if the larger interleaver is used.

[0144] 2.4 Why are Turbo-like Codes Different?

[0145] The performance of Turbo-like Codes depends on the delay that is allowed in the turbo-like encoder. In the case of Turbo Codes, the elementary decoders rely heavily on the interleaving/de-interleaving process to de-correlate their soft outputs. The larger the interleaver, the better the de-correlation that can be achieved, the larger the parity check matrix and the better the performance of the LDPC codes.

[0146] The sensitivity of the Turbo-like Codes to the correlation between data bits can also be noticed in higher order modulation schemes when two or more data bits are estimated from the same receiver symbol. Different mapping techniques are investigated for a spectral efficiency of four information bits per symbol using 64 QAM. Simulation results shows that Gray mapping performs better than Natural mapping, which in turn is better than the Ungerboeck set partitioning. A fourth case when two bits where sent uncoded also performed badly because of a smaller interleaver size.

[0147] The complexity of the bit estimator block in **FIG. 1** increases with the constellation size because more bits need to be estimated per symbol. The equivalent of this in a trellis coded modulation scheme with un-coded bits is that more parallel paths are added to the trellis, therefore there is an increased computational effort in the branch metric estimator.

[0148] 2.5 Capacity Bound

[0149] The minimum E_b/N_O values to achieve the Shannon bound, 4 QAM, 8 QAM, 16 QAM, 32 QAM, 64 QAM, 128 QAM, 256 QAM, 512 QAM and 1024 QAM bounds for spectral efficiencies form 2/3 up to 7 bits/s/Hz respectively are as in Table 3 for a BER=10^{-5}.

TABLE 3

Shannon and QAM bounds.

Spectral efficiency η [bit/s/Hz]	Shannon bound [dB]	4 QAM bound [dB]	8 QAM bound [dB]	16 QAM bound [dB]	32 QAM bound [dB]	64 QAM bound [dB]	128 QAM bound [dB]	256 QAM bound [dB]	512 QAM bound [dB]	1024 QAM bound [dB]
2/3	−0.5	0.3	0	−0.4	−0.4	−0.4	−0.4	−0.4	−0.4	−0.4
1	0	1.0	0.9	0.1	0.1	0.1	0.1	0.1	0.1	0.1

TABLE 3-continued

Shannon and QAM bounds.

Spectral efficiency η [bit/s/Hz]	Shannon bound [dB]	4 QAM bound [dB]	8 QAM bound [dB]	16 QAM bound [dB]	32 QAM bound [dB]	64 QAM bound [dB]	128 QAM bound [dB]	256 QAM bound [dB]	512 QAM bound [dB]	1024 QAM bound [dB]
2	1.75	∞	2.2	2.1	2.09	2.09	2.09	2.09	2.09	2.09
3	3.7	—	∞	4.6	4.3	4.3	4.3	4.3	4.3	4.3
4	5.6	—	—	∞	6.6	6.6	6.6	6.6	6.6	6.6
5	7.9	—	—	—	∞	9.1	9.0	9.0	9.0	9.0
6	10.3	—	—	—	—	∞	11.8	11.7	11.7	11.7
7	12.6	—	—	—	—	—	∞	14.5	14.5	14.5

[0150] The conversion for E_s/N_O to E_b/N_O is performed using the following relation

$$E_b/N_O[dB]=E_s/N_O[dB]-10\log_{10}(\eta)[dB] \quad (1)$$

[0151] where η is the number of information bits per symbol.

[0152] The required C/N_O given a certain E_b/N_O can be found using the following relation:

$$C/N_O[dB-Hz]=E_b/N_O[dB]+10\log_{10}(R_b)[dB-Hz] \quad (2)$$

[0153] where R_b is the information bit rate.

[0154] For a D-dimension modulation the following formulae are used:

$$SNR=\frac{E[|a_k^2|]}{E[|w_k^2|]}=\frac{E[|a_k^2|]}{D\sigma_N^2}=\frac{E_{av}}{D\sigma_N^2} \quad (3)$$

$$SNR=\frac{E_s}{D\frac{N_0}{2}}=\frac{\eta E_b}{D\frac{N_0}{2}} \quad (4)$$

[0155] where σ^2_N is the noise variance in each of the D dimension and η is the number of information bits per symbol. From the above relations:

$$\sigma_N^2=E_{av}\left(\frac{2\eta E_b}{N_0}\right)^{-1} \quad (5)$$

[0156] 3. Coding and Modulation for 2/3 bit/s/Hz Spectral Efficiency

[0157] The only option investigated in this section combines a rate 2/6 coding scheme with 4 QAM.

[0158] 3.1 Coding

[0159] A preferred embodiment uses a coding scheme as shown in **FIG. 3**. The encoder is formed by two 1/2 systematic recursive constituent Concatenated Convolutional Encoders (CCE) that receive two inputs every information input bit cycle, adding one parity bit per dimension. We propose the convolution encoder format (g1,g2)=(23o, 35o) joined through an interleaver and which can be generalized to a N constituent systematic encoders with N-1 interleavers.

[0160] **FIG. 4** shows a preferred convolutional encoder. g1 and g2 are the values in octal of the forward and feedback lines. In **FIG. 2**, the upper side the first line is feedback (value 1), the second line is also a feedback (value 2) and the fifth line as well (value 16). The feedback lines sum to a value of 19 in decimal or 23 in octal, and is represented as 23o. In **FIG. 2**, the lower side the first line is forward line (value 1), the same thing happen with the third (value 4), forth (value 8) and fifth (value 16), this makes the total forward line to a value of 29 in decimal or 35 in octal, and is represented as 35o.

[0161] 3.2 Puncturing

[0162] In order to obtain a code rate of 2/6, no puncturing is applied. The transmission pattern is given in

[0163] Table 4.

TABLE 4

Transmission pattern for Rate 2/6 4 QAM.

Information bit (d)		d_1	d_2
parity bit (p)		p1	p2
parity bit (q)		q1	q2
2 AM symbol (I)	$(u_1) = (d_1)$	$(u_1) = (d_2)$	$(u_1) = (q1)$
2 AM symbol (Q)	$(u_2) = (p1)$	$(u_2) = (q2)$	$(u_2) = (p2)$
4 QAM symbol (I, Q)	(I, Q) = (u_1, u_2) = $(d_1, p1)$	(I, Q) = (u_1, u_2) = $(d_2, q2)$	(I, Q) = (u_1, u_2) = (q1, p2)

[0164] 3.3 Modulation

[0165] A 4 QAM scheme is shown in **FIG. 5**. At time k, the symbol $u^k=(u_1^k)$ is sent through the channel and the point r^k in two dimensional space is received.

[0166] For a 4 QAM constellation with points at −A and A, The E_{av} is:

$$E_{av}=\frac{4(A^2+A^2)}{4}=2A^2 \quad (6)$$

[0167] For a rate 2/6 code and 4 QAM, the noise variance is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right) = 2A^2\left(\frac{2\times\frac{2}{3}\times E_b}{N_0}\right)^{-1} = \frac{3}{2}A^2\left(\frac{E_b}{N_0}\right)^{-1} \quad (7)$$

[0168] It is assumed that the time k, u_1^k modulates the I component and u_2^k modulates the Q component for a 4 QAM scheme

[0169] The symbol u^k symbol has the following mapping: $u^k=(u_1^k, u_2^k)=(d^i, q^i)$; $u^{k+1}=(u_1^{k+1}, u_2^{k+1})=(d^{i+1}, q^{i+1})$; $u^{k+2}=(u_1^{k+2}, u_2^{k+2})=(q^{i+1}, p^{i+2})$

[0170] Considering two independent Gaussian noises with identical variance σ^2_N, the LLR can be determined independently for each I and Q.

[0171] At the receiver, the I and Q signals are treated independently in order to take advantage of the simpler formulae for the 2 bit-LLR values.

[0172] In order to estimate the performance of this scheme, a 2 AM modulation is used, as it is shown in **FIG. 6**, instead of 4 QAM modulation.

[0173] The 4 QAM scheme will achieve a similar performance in terms of bit error rate (BER) at twice the spectral efficiency, assuming an ideal demodulator.

[0174] For a rate 2/6 code and a 2 AM scheme as shown in **FIG. 6**, the noise variance is:

$$E_{av_I}=(1+1)A^2/2=A^2 \quad (8)$$

[0175]

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right) = A^2\left(\frac{2e\frac{2}{6}\times E_b}{N_0}\right)^{-1} = \frac{3}{2}A^2\left(\frac{E_b}{N_0}\right)^{-1} \quad (9)$$

[0176] That is the same that the 4 QAM of equation (7)

[0177] 3.4 Bit Probabilities

[0178] For an AWGN channel the following expressions need to be evaluated:

$$LLR(u_1^k) = \log\left(\frac{\sum_{i=1}^{1}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{1i}^k)^2\right]}{\sum_{i=1}^{1}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_1)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0j}^k)^2\right]}\right) \quad (10)$$

-continued

$$LLR(u_2^k) = \log\left(\frac{\sum_{i=1}^{1}\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - a_{1i}^k)^2\right]}{\sum_{i=1}^{1}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - B_1)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - B_0)^2\right]}\right) \quad (11)$$

[0179] The above LLRs are used as inputs to the turbo encoder. There is no need to compute the 4 LLRs for all symbols because I and Q signals are treated independently. Also the simulations for the 2 bit-LLR values are reduced to one term each.

[0180] 3.5 Simulations Results

[0181] **FIG. 7** shows the performance of a turbo code which 1024 bit interleavers using S-type interleaver.

[0182] The target BER of 10^{-7} for a 1,024 information bit interleaver can be achieve at E_b/N_O=1.25 dB

[0183] 4. Coding and Modulation for 1 bit/s/Hz Spectral Efficiency

[0184] Two options are investigated in this section. The first scheme combines a rate 2/4 coding scheme with 4 QAM. The second scheme investigated combines 2/6 coding scheme with 8 QAM.

[0185] 4.1 Option 1: Rate 2/4 Turbo Code and 4 QAM

[0186] 4.1.1 Coding

[0187] The coding scheme is shown in **FIG. 3**.

[0188] The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0189] 4.1.2 Puncturing

[0190] In order to obtain a code rate of 2/4, every other bit of the parity bits p and q from **FIG. 3** are punctured. The puncturing pattern is given in Table 5.

TABLE 5

Puncturing and Mapping for Rate 2/4 4 QAM.

Information bit (d)	d_1	d_2
parity bit (p)	p_1	—
parity bit (q)	—	q_2
2 AM symbol (I)	$(u_1) = (d_1)$	$(u_1) = (d_2)$
2 AM symbol (Q)	$(u_2) = (p_1)$	$(u_2) = (q_2)$
4 QAM symbol (I,Q)	$(I,Q) = (u_1, u_2) = (d_1, p_1)$	$(I,Q) = (u_1, u_2) = (q_2)$

[0191] 4.1.3 Modulation

[0192] A 4 QAM scheme is shown in **FIG. 5**. At time k, the symbol $u^k=(u_1^k, u_2^k)$ is sent through the channel and the point r^k in two dimensional space is received.

[0193] For a 4 QAM constellation with points at −A and A, The E_{av} is:

$$E_{av} = \frac{4(A^2 + A^2)}{4} = 2A^2 \tag{12}$$

[0194] For a rate 2/4 code and 4 QAM, the noise variance is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right) = 2A^2\left(\frac{2\times 1\times E_b}{N_0}\right)^{-1} = A^2\left(\frac{E_b}{N_0}\right)^{-1} \tag{13}$$

[0195] It is assumed that the time k, u_1^k modulates the I component and u_2^k modulates the Q component for a 4 QAM scheme.

[0196] The symbol u^k symbol has the following mapping: $u^k=(u_1^k, u_2^k)=(d^i, p^i)$; $u^{k+1}=(u_1^{k+1}, u_2^{k+1})=(d^{i+1}, q^{i+1})$.

[0197] Considering two independent Gaussian noises with identical variance σ^2_N, the LLR can be determined independently for each I and Q.

[0198] At the receiver, the I and Q signal are treated independently in order to take advantage of the simpler formulae for the 2 bit-LLR values.

[0199] In order to estimate the performance of this scheme, rate 1/2 turbo codes a 2 AM modulation is used, as it is shown in **FIG. 6**, instead of 4 QAM modulation.

[0200] The 4 QAM scheme will achieve a similar performance in terms of bit error rate (BER) at twice the spectral efficiency, assuming an ideal demodulator.

[0201] For a rate 1/2 code and a 2 AM scheme as shown in **FIG. 6**, the noise variance is:

$$E_{av_1}=(1+1)A^2/2=A^2 \tag{14}$$

[0202]

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right) = A^2\left(\frac{2\times 0.5\times E_b}{N_0}\right)^{-1} = A^2\left(\frac{E_b}{N_0}\right)^{-1} \tag{15}$$

[0203] That is the same that the 4 QAM of equation (13)

[0204] 4.1.4 Bit Probabilities.

[0205] For an AWGN channel the following expressions need to be evaluated.

$$LLR(u_1^k) = \log\left(\frac{\sum_{i=1}^{1}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{1i}^k)^2\right]}{\sum_{i=1}^{1}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_1)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0j}^k)^2\right]}\right) \tag{16}$$

-continued

$$LLR(u_2^k) = \log\left(\frac{\sum_{i=1}^{1}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{1i}^k)^2\right]}{\sum_{i=1}^{1}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - B_1)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - B_0)^2\right]}\right) \tag{17}$$

[0206] The above LLRs are used as inputs to the turbo encoder. There is no need to compute the 4 LLRs for all symbols because I and Q signals are treated independently. Also the simulations for the 2 bit-LLR values are reduced to one term each. Due to the puncturing, with one in two parity bits being transmitted, the expected performance will be lower when compared with the non-punctured scheme.

[0207] For an Impulse Noise channel, for a determined period of time (the duration of the impulse) the E_b/N_O ratio changes to a worse value, and the errors occurs. The Reference Level of E_b/N_O is set to the value of E_b/N_O needed by the un-coded QAM constellation to reach the value of BER of 10^{-7}.

[0208] 4.1.5 Simulations Results

[0209] **FIG. 8** shows the performance of a turbo code which 1024 bit interleavers using S-type interleaver.

[0210] The target BER of 10^{-7} for a 1,024 information bit interleaver can be achieve at E_b/N_O=2.1 dB

[0211] 4.2 Option 1: Rate 1/3 Turbo Code and 8 QAM

[0212] 4.2.1 Coding

[0213] The coding scheme is shown in **FIG. 3**.

[0214] The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0215] 4.2.2 Puncturing

[0216] No puncturing is applying in this case. The pattern is given in Table 6.

TABLE 6

Pattern and Mapping for Rate 1/3 8 QAM.

Information bit (d)	d_1
parity bit (p)	p1
parity bit (q)	q1
4 AM symbol (I)	$(u_1, u_2) = (d_1, p1)$
2 AM symbol (Q)	$(u_3) = (q1)$
8 QAM symbol (I, Q)	$(I, Q) = (d_1, p1, q1)$

[0217] 4.2.3 Modulation

[0218] The 8 QAM scheme used is shown in **FIG. 9**. At time k, the symbol $u^k=(u_1^k, u_2^k, u_3^k)$ is sent through the channel and the point r^k in two dimensional space is received.

[0219] For an 8 QAM costellation with points at −3A, −A, A and 3A, in the I dimension and −A, and A in the Q dimension

The E_{av_I} is:

$$E_{av_I} = \frac{(1+9+1+9)A^2}{4} = 5A^2 \tag{18}$$

The E_{av_Q} is:

$$E_{av_Q} = \frac{(1+1)A^2}{2} = A^2 \tag{19}$$

[0220] Because the noise has to be spherical, the total value of E_{av} will be the addition of these two values, so:

$$E_{av}=E_{av_}I+E_{av_}Q=6A^2 \tag{20}$$

[0221] The value of the noise variance has to be the same in both dimensions and its value is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right) = 6A^2\left(\frac{2\times 1\times E_b}{N_0}\right)^{-1} = 3A^2\left(\frac{E_b}{N_0}\right)^{-1} \tag{21}$$

[0222] It is assumed that the time k, u_1^k and u_2^k modulates the I component and u_3^k modulates the Q component for an 8 QAM scheme.

[0223] The symbol u^k has the following mapping: $u^k=(u_1^k, u_2^k, u_3^k)=(d^i, p^i, q^i)$. The parity bits are mapped to the least protected bits of the QAM symbol. Note that k denotes the symbol time index and information bit time index.

[0224] Considering two independent Gaussian noises with identical variance σ^2_N, the LLR can be determined independently for each I and Q.

[0225] At the receiver, the I and Q signals are treated independently in order to take advantage of the simpler formulae for the LLR values. The mapping of the information bit is made to the most protected bit in each dimension (u_1^k for the I signal and u_3^k for the Q signal in **FIG. 9**).

[0226] In order to estimate the performance of this scheme, rate 1/3 turbo codes and 4 AM modulation (as it is shown in **FIG. 10**) is used in the I dimension, and a 2 AM modulation is used for the Q dimension, (as it is shown in **FIG. 6**), instead of 8 QAM modulation.

[0227] 4.2.4 Bit Probabilities

[0228] For an AWGN channel the following expressions need to be evaluated:

$$LLR(u_1^k) = \log\left(\frac{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_2)^2\right]+\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_3)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_0)^2\right]+\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_1)^2\right]}\right) \tag{22}$$

$$LLR(u_2^k) = \log\left(\frac{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_1)^2\right]+\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_3)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_0)^2\right]+\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_2)^2\right]}\right) \tag{23}$$

$$LLR(u_3^k) = \log\left(\frac{\sum_{i=1}^{1}\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{1}\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - B_1)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - B_0)^2\right]}\right) \tag{24}$$

[0229] The above LLRs are used as inputs to the turbo encoder. There is no need to compute the 8 LLRs for all symbols because I and Q signals are treated independently. Also the simulations for the LLR values are reduced to two terms for the 4 AM modulation and one term for the 2 AM modulation. Due to the puncturing, the expected performance will be lower when compared with the non-punctured scheme.

[0230] 4.2.5 Simulations Results

[0231] **FIG. 11** shows the performance of a turbo code, which used 1024 bit interleavers using S-type interleaver.

[0232] The target BER of 10^{-7} can be achieve at E_b/N_O= 3.1 dB.

[0233] 5. Coding and Modulation for 2 bit/s/Hz Spectral Efficiency

[0234] Two options are investigated in this section. The first scheme combines a rate 4/6 coding scheme with 8 QAM. The second scheme investigated combines 1/2 coding scheme with 16 QAM. These schemes have the advantage of a very efficient implementation without any significant compromise in performance.

[0235] 5.1 Option 1: Rate 4/6 Turbo Code and 8 QAM

[0236] 5.1.1 Coding

[0237] The coding scheme is shown in **FIG. 3**.

[0238] The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o. 5.1.2 Puncturing

[0239] In accordance with an embodiment of the invention, the puncturing pattern given in Table 7 is used in order to obtain a rate 4/6 code.

TABLE 7

Puncturing and Mapping for Rate 4/6 8 QAM.

Information bit (d)	d_1	d_2	d_3	d_4
parity bit (p)	p1	—	—	—
parity bit (q)	—	—	q3	—
4 AM symbol (I)	(u_1, u_2) = $(d_1, p1)$		(u_1, u_2) = $(d_3, q3)$	
2 AM symbol (Q)	$(u_3) = (d_2)$		$(u_3) = (d_4)$	
8 QAM symbol (I, Q)	(I, Q) = $(d_1, p1, d_2)$		(I, Q) = $(d_3, q3, d_4)$	

[0240] 5.1.3 Modulation

[0241] The 8 QAM scheme used is shown in **FIG. 9**. At time k, the symbol $u^k=(u_1^k, u_2^k, u_3^k)$ is sent through the channel and the point r^k in two dimensional space is received.

[0242] For an 8 QAM costellation with points at −3A, −A, A and 3A, in the I dimension and −A, and A in the Q dimension

[0243] The $E_{av_}I$ is:

$$E_{av_I} = \frac{(1+9+1+9)A^2}{4} = 5A^2 \tag{25}$$

$$\text{The } E_{av_Q} \text{ is: } E_{av_Q} = \frac{(1+1)A^2}{2} = A^2 \tag{26}$$

[0244] Because the noise has to be spherical, the total value of E_{av} will be the addition of these two values, so:

$$E_{av}=E_{av_}I+E_{av_}Q=6A^2 \tag{27}$$

[0245] The value of the noise variance has to be the same in both dimensions and its value is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right) = 6A^2\left(\frac{2\times 2\times E_b}{N_0}\right)^{-1} = \frac{3}{2}A^2\left(\frac{E_b}{N_0}\right)^{-1} \tag{28}$$

[0246] It is assumed that the time k, u_1^k and u_2^k modulates the I component and u_3^k modulates the Q component for an 8 QAM scheme.

[0247] The symbol u^k has the following mapping: $u^k=(u_1^k, u_2^k, u_3^k)=(d^1, p^1, d^{1+1})$. The parity bits are mapped to the least protected bits of the QAM symbol. Note that k denotes the symbol time index and information bit time index.

[0248] Considering two independent Gaussian noises with identical variance σ^2_N, the LLR can be determined independently for each I and Q.

[0249] At the receiver, the I and Q signals are treated independently in order to take advantage of the simpler formulae for the LLR values. The mapping of the information bit is made to the most protected bit in each dimension (u_1^k for the I signal and u_3^k for the Q signal in **FIG. 9**).

[0250] In order to estimate the performance of this scheme, rate 4/6 turbo codes and 4 AM modulation (as it is shown in **FIG. 10**) is used in the I dimension, and a 2 AM modulation is used for the Q dimension, (as it is shown in **FIG. 6**), instead of 8 QAM modulation.

[0251] 5.1.4 Bit Probabilities

[0252] For an AWGN channel the following expressions need to be evaluated:

$$LLR(_{u_1}^k) = \log\left(\frac{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_2)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_3)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_0)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_1)^2\right]}\right) \tag{29}$$

$$LLR(_{u_2}^k) = \log\left(\frac{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_1)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_3)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_0)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_2)^2\right]}\right) \tag{30}$$

$$LLR(_{u_3}^k) = \log\left(\frac{\sum_{i=1}^{1}\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{1}\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - B_1)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - B_0)^2\right]}\right) \tag{31}$$

[0253] The above LLRs are used as inputs to the turbo encoder. There is no need to compute the 8 LLRs for all symbols because I and Q signals are treated independently. Also the simulations for the LLR values are reduced to two terms for the 4 AM modulation and one term for the 2 AM modulation. Due to the puncturing, the expected performance will be lower when compared with the non-punctured scheme.

[0254] 5.1.5 Simulations Results

[0255] **FIG. 12** shows the performance of a turbo code, which used 1024 bit interleavers using S-type interleaver.

[0256] The target BER of 10^{-7} can be achieve at E_b/N_O= 5.5 dB.

[0257] 5.2 Option 2: Rate 1/2 Turbo Code and 16 QAM

[0258] 5.2.1 Coding

[0259] The coding scheme is shown in **FIG. 3**.

[0260] The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0261] 5.2.2 Puncturing

[0262] In order to obtain a rate 1/2 code, every other bit of the parity bits p and q from **FIG. 3** are punctured. The puncturing pattern is given in Table 8. This is the traditional puncturing pattern using alternating parity bits in each cycle.

TABLE 8

Puncturing and Mapping for Rate 1/2 16 QAM.

Information bit (d)		d_1	d_2
parity bit (p)		p1	—
parity bit (q)		—	q2
4 AM symbol (I)	$(u_1, u_2) = (d_1, p1)$		
4 AM symbol (Q)	$(u_3, u_4) = (d_2, q2)$		
16 QAM symbol (I, Q)	$(I, Q) = (u_1, u_2, u_3, u_4) = (d_1, p1, d_2, q2)$		

[0263] 5.2.3 Modulation

[0264] A 16 QAM scheme is shown in **FIG. 13**. At time k, the symbol $u^k=(u_1^k, u_2^k, u_3^k, u_4^k)$ is sent through the channel and the point r^k in two dimensional space is received.

[0265] For a 16 QAM constellation with points at −3A, −A,A and 3A, The E_{av} is:

$$E_{av} = \frac{4(A^2 + A^2) + 8(A^2 + 9A^2) + 4(9A^2 + 9A^2)}{16} = 10A^2 \tag{32}$$

[0266] For a rate 1/2 code and 16 QAM, the noise variance in each dimension is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right) = 10A^2\left(\frac{2\times 2\times E_b}{N_0}\right)^{-1} = 2.5A^2\left(\frac{E_b}{N_0}\right)^{-1} \tag{33}$$

[0267] It is assumed that the time k, u_1^k and u_2^k modulates the I component and u_3^k and u_4^k modulates the Q component for a 16 QAM scheme.

[0268] The symbol u^k symbol has the following mapping: $u^k=(u_1^k, u_2^k, u_3^k, u_4^k)=(d^i, p^i, d^{i+i}, q^{i+i})$. The parity bits are mapped to the least protected bits of the QAM symbol. Note that k denotes the symbol time index and i the information bit time index. This means a puncturing of one in two parity bits.

[0269] Considering two independent Gaussian noises with identical variance σ^2_N, the LLR can be determined independently for each I and Q.

[0270] At the receiver, the I and Q signals are treated independently in order to take advantage of the simpler formulae for the 4 bit-LLR values. The mapping of the information bit is made to the most protected bit in each dimension (u_1^k for the I signal and u_3^k for the Q signal in **FIG. 13**).

[0271] In order to estimate the performance of this scheme, rate 1/2 turbo code and 4 AM modulation is used (as it is shown in **FIG. 10**), instead of 16 QAM modulation.

[0272] The 16 QAM scheme will achieve a similar performance in terms of bit error rate (BER) at twice the spectral efficiency, assuming an ideal demodulator.

[0273] For a rate 1/2 code and a 4 AM scheme as shown in **FIG. 6**, the noise variance is:

$$E_{av_1}=(1+9)A^2/2=5A^2 \tag{34}$$

[0274]

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right) = 5A^2\left(\frac{2\times 1\times E_b}{N_0}\right)^{-1} = 2.5A^2\left(\frac{E_b}{N_0}\right)^{-1} \tag{35}$$

[0275] 5.2.4 Bit Probabilities

[0276] For an AWGN channel the following expressions need to be evaluated:

$$LLR(u_1^k) = \log\left(\frac{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_2)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_3)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_0)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_1)^2\right]}\right) \tag{36}$$

$$LLR(u_2^k) = \log\left(\frac{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_1)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_3)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_0)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_2)^2\right]}\right) \tag{37}$$

$$LLR(u_3^k) = \log\left(\frac{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - B_2)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(Q^k - B_3)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - B_0)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(Q^k - B_1)^2\right]}\right) \tag{38}$$

-continued

$$LLR(u_4^k) = \log\left(\frac{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - B_1)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(Q^k - B_3)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(Q^k - B_0)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(Q^k - B_2)^2\right]}\right) \quad (39)$$

[0277] The above LLRs are used as inputs to the turbo encoder. There is no need to computer the 16 LLR for all symbols because O and Q signals are treated independently. Also the simulations for the 4 bit-LLR values are reduced to 2 terms each. Due to the puncturing, with one in two parity bits being transmitted, the expected performance will be lower when compared with the non punctured scheme.

[0278] 5.2.5 Simulations Results

[0279] **FIG. 14** shows the performance of a turbo code that uses the odd-even block helical interleaver of 272 bits. **FIG. 15** shows the performance of a turbo code that uses the S-type interleaver for 256 bits.

[0280] The S-type interleaver performs better than the odd-even block helical interleaver with approximately 1 dB at a BER=10^{-6} in this case. Therefore, the rest of the simulations used the S-type interleaver.

[0281] FIGS. **16** to **18** show the simulation results for 512, 768 and 1024 bit interleavers using S-type interleaver.

[0282] A search for the best S-type interleavers was performed. For each interleaver size, the maximum S-factor was found and four different interleavers were generated using a random seed. Four simulations were generated for a target BER of 10^{-5}. The best result decided the preferred interleaver, complete simulations were started for BERs down to 10^{-7}.

[0283] In the case of N=768, a further interleaver search is required in order to avoid the flattening of the BER curve which started to show below 10^{-6} in **FIG. 17**.

[0284] 5.2.6 Recommended Solution

[0285] The rate 1/2 16 QAM scheme described in this chapter achieves the target BER at less than 1.5 dB from capacity. The implementation in hardware is feasible and it can be used at very high data rates.

[0286] The target BER of 10^{-7} can be achieve at the following values of E_b/N_O=6.8 dB for N=256 information bits and E_b/N_O=is 4.5 dB for N=1024 information bits.

[0287] 6. Coding and Modulation for 3 bit/s/Hz Spectral Efficiency

[0288] Three options are investigated in this section. The first scheme combines a rate 3/4 coding scheme with 16 QAM The second scheme combines a rate 3/5 coding scheme with 32 QAM and the third scheme combines a rates 3/6 Coding scheme with 64 QAM

[0289] 6.1 Option 1 Rate 3/4 Turbo Code and 16 QAM

[0290] 6.1.1 Coding

[0291] The coding used in this scheme is shown in **FIG. 3**. The two systematic recursive codes (SRC) used are identical and is defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0292] 6.1.2 Puncturing

[0293] In accordance with an embodiment of the invention, in order to obtain a rate 3/4 code, the puncturing pattern used is given in Table 9.

TABLE 9

Puncturing and Mapping for Rate 3/4 16 QAM

Information bit (d)	d_1	d_2	d_3	d_4	d_5	d_6
parity bit (p)	—	p2	—	—	—	—
parity bit (q)	—	—	—	—	q5	—
4 AM symbol (I)	(d_1, d_2)			(d_4, d_5)		
4 AM symbol (Q)	(d_3, p2)			(d_6, q5)		
16 QAM symbol (I, Q)	(I, Q) = (d_1, d_2, d_3, p2)			(I, Q) = (d_4, d_5, d_6, q5)		

[0294] 6.1.3 Modulation

[0295] A 16 QAM scheme is shown in **FIG. 13**. It is assumed that at time k u_1^k and u_2^k modulates the I component and U_3^k and u_4^k modulates the Q component of a 16 QAM scheme.

[0296] In order to estimate the performance of this scheme, rate 3/4 turbo codes and 4 AM modulation are used.

[0297] For a rate 3/4 code and 16 QAM, the noise variance in each dimension is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right)^{-1} = 10A^2\left(\frac{2\times 3\times E_b}{N_0}\right)^{-1} = \frac{10}{6}A^2\left(\frac{E_b}{N_0}\right)^{-1} \quad (40)$$

[0298] The puncturing and mapping scheme is shown in Table 9 for 6 consecutive information bits that are encoded into 8 coded bits, therefore two 16 QAM symbols.

[0299] For the 4 AM scheme as shown in **FIG. 13**, the noise variance is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right) = 10A^2\left(\frac{2\times 1.5\times E_b}{N_0}\right)^{-1} = \frac{10}{3}A^2\left(\frac{E_b}{N_0}\right)^{-1} \quad (41)$$

[0300] 6.1.4 Bit Probabilities

[0301] For each received symbol, the bit probabilities are computed as described in equations (36) to (39).

[0302] 6.1.5 Simulation Results

[0303] **FIG. 19** shows the simulation results for 6,144 information bits with S-type interleaver.

[0304] A BER of 10^{-7} can be achieved after 8 iterations at E_b/N_O=5.55 dB for N =6,144 information bits.

[0305] 6.2 Option 2 Rate 3/5 Turbo code and 32 QAM

[0306] 6.2.1 Coding

[0307] The coding scheme is shown in **FIG. 3**. The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0308] 6.2.2 Puncturing

[0309] In accordance with an embodiment of the invention, in order to obtain a rate 3/5 code, the puncturing pattern used is given in Table 10.

TABLE 10

Puncturing and Mapping for Rate 3/5 32 QAM

Information bit (d)	d_1	d_2	d_3	d_4	d_5	d_6
parity bit (p)	p1	—	—	p4	—	—
parity bit (q)	—	q2	—	—	q5	—
8 AM symbol (I)		(d_1, d_3, p1)			(d_4, d_6, q5)	
4 AM symbol (Q)		(d_2, q2)			(d_5, p4)	
32 QAM symbol (I, Q)	(I, Q) = (d_1, d_3, p1, d_2, q2)			(I, Q) = (d_4, d_6, q5, d_5, p4)		

[0310] 6.2.3 Modulation

[0311] The asymmetrical 32 QAM modulation scheme considered is shown in **FIG. 20**. At time k, the symbol $u^k=(u_1^k, u_2^k, u_3^k, u_4^k, u_5^k)$, is sent through the channel and the point r^k in two dimensional space is received.

[0312] It is assumed that at time k u_1^k, u_2^k and u_3^k modulates the I component and u_4^k and u_5^k modulates the Q component of the 32 QAM scheme.

[0313] In the I dimension, two information bits and one parity bit are transmitted using Gray mapping and 8 AM as shown in **FIG. 21**.

[0314] The $E_{av_}I$ is:

$$E_{av_I} = \frac{(1+9+25+49)A^2}{4} = 21A^2 \tag{42}$$

$$\text{The } E_{av_Q} \text{ is: } E_{av_Q} = \frac{(1+9)A^2}{2} = 5A^2 \tag{43}$$

[0315] Because the noise has to be spherical, the total value of E_{av} will be the addition of these two values, so:

$$E_{av}=E_{av_I}+E_{av_Q}=26\,A^2 \tag{44}$$

[0316] The value of the noise variance has to be the same in both dimensions and its value is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right) = 26A^2\left(\frac{2\times 3\times E_b}{N_0}\right)^{-1} = \frac{26}{6}A^2\left(\frac{E_b}{N_0}\right)^{-1} \tag{45}$$

[0317] The puncturing and mapping scheme is shown in Table 10 for 6 consecutive information bits that are encoded into 10 coded bits, therefore two 32 QAM symbols.

[0318] 6.2.4 Bit Probabilities

[0319] For an AWGN channel, the following expressions need to be evaluated for the I dimension:

$$LLR(u_1^k) = \log\left(\frac{\sum_{i=1}^{4}\exp\left[-\frac{1}{2\sigma_N^2}(I^k-a_{1,j}^k)^2\right]}{\sum_{i=1}^{4}\exp\left[-\frac{1}{2\sigma_N^2}(I^k-a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\begin{array}{c}\exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_4)^2\right]+\exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_5)^2\right]+\\ \exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_6)^2\right]+\exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_7)^2\right]\end{array}}{\begin{array}{c}\exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_0)^2\right]+\exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_1)^2\right]+\\ \exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_2)^2\right]+\exp\left[\frac{1}{2\sigma_N^2}(I^k-A_3)^2\right]\end{array}}\right) \tag{46}$$

$$LLR(u_2^k) = \log\left(\frac{\sum_{i=1}^{4}\exp\left[-\frac{1}{2\sigma_N^2}(I^k-a_{1,j}^k)^2\right]}{\sum_{i=1}^{4}\exp\left[-\frac{1}{2\sigma_N^2}(I^k-a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\begin{array}{c}\exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_2)^2\right]+\exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_3)^2\right]+\\ \exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_6)^2\right]+\exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_7)^2\right]\end{array}}{\begin{array}{c}\exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_0)^2\right]+\exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_1)^2\right]+\\ \exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_4)^2\right]+\exp\left[\frac{1}{2\sigma_N^2}(I^k-A_5)^2\right]\end{array}}\right) \tag{47}$$

$$LLR(u_3^k) = \log\left(\frac{\sum_{i=1}^{4}\exp\left[-\frac{1}{2\sigma_N^2}(I^k-a_{1,j}^k)^2\right]}{\sum_{i=1}^{4}\exp\left[-\frac{1}{2\sigma_N^2}(I^k-a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\begin{array}{c}\exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_1)^2\right]+\exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_5)^2\right]+\\ \exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_3)^2\right]+\exp\left[-\frac{1}{2\sigma_n^2}(I^k-A_7)^2\right]\end{array}}{\begin{array}{c}\exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_0)^2\right]+\exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_4)^2\right]+\\ \exp\left[-\frac{1}{2\sigma_N^2}(I^k-A_2)^2\right]+\exp\left[\frac{1}{2\sigma_N^2}(I^k-A_6)^2\right]\end{array}}\right) \tag{48}$$

[0320] For the Q dimension, only $LLR(u_4^k)$ and $LLR(u_5^k)$ are required to be evaluated:

$$LLR(u_4^k) = \log\left(\frac{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_2)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_3)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_0)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_1)^2\right]}\right) \quad (49)$$

$$LLR(u_5^k) = \log\left(\frac{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{2}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_1)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_3)^2\right]}{\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_0)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_2)^2\right]}\right) \quad (50)$$

[0321] 6.2.5 Simulation Results

[0322] **FIG. 22** shows the simulation results for 6,144 information bits with S-type interleaver.

[0323] A BER of 10^{-6} can be achieved after 8 iterations at E_b/N_O=6.5 dB for N=4,096 information bits

[0324] 6.3 Option 2 Rate 3/6 Turbo code and 64 QAM

[0325] 6.3 .1 Coding

[0326] The coding scheme is shown in **FIG. 3**. The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0327] 6.3.2 Puncturing

[0328] In order to obtain a rate 3/6 code, the puncturing pattern used is given in Table 11. This is the traditional puncturing pattern using alternating parity bits in each cycle.

TABLE 11

Puncturing and Mapping for Rate 3/6 64 QAM

Information bit (d)	d_1	d_2	d_3	d_4	d_5	d_6
parity bit (p)	p1	—	p3	—	p5	—
parity bit (q)	—	q2	—	q4	—	q6
8 AM symbol (I)		(d_1, d_2, p1)			(d_4, d_5, q4)	
8 AM symbol (Q)		(d_3, p3, q2)			(d_6, p5, q6)	
64 QAM symbol (I, Q)		(I, Q) = (d_1, d_2, p1, d_3, p3, q2)			(I, Q) = (d_4, d_5, q4, d_6, p5, q6)	

[0329] 6.3.3 Modulation

[0330] Let us consider 64 QAM modulation scheme as it is shown in **FIG. 23**. At time k, the symbol $u^k=(u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k)$, is sent through the channel and the point r^k in two dimensional space is received.

[0331] It is assumed that at time k u_1^k, u_2^k and u_3^k modulates the I component and u_4^k, u_5^k and u_6^k modulates the Q component of a 64 QAM scheme.

[0332] For 64 QAM constellations with points at −7A, −5A, −3A, −A, A, 3A, 5A, 7A The E_{av} is:

$$E_{av}=(8(49+25+9+1)+8(25+49+49+9+49+1)+8(25+9+25+1)+8(9+1))A^2/64=42\,A^2 \quad (51)$$

[0333] For a rate 3/6 code and 64 QAM, the noise variance in each dimension is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right)^{-1} = 42A^2\left(\frac{2\times 3\times E_b}{N_0}\right)^{-1} = 7A^2\left(\frac{E_b}{N_0}\right)^{-1} \quad (52)$$

[0334] In order to estimate the performance of this scheme, when rate 3/6 turbo codes and 8 AM modulation is used, as shown in **FIG. 21**.

[0335] The 64 QAM scheme will achieve a similar performance in terms of bit error rate (BER) at twice the spectral efficiency, assuming an ideal modulator.

[0336] For the 8 AM scheme shown in **FIG. 21**, the noise variance is

$$\sigma_N^2 = E[|a_k^2|]\left(\frac{2\eta E_b}{N_0}\right)^{-1} = 21A^2\left(\frac{2\times 1.5\times E_b}{N_0}\right)^{-1} = 7A^2\left(\frac{E_b}{N_0}\right)^{-1} \quad (53)$$

[0337] This is identical to the variance for the 64 QAM scheme.

[0338] The puncturing and mapping scheme is shown in Table 11 for 6 consecutive information bits that are encoded into 12 coded bits, therefore two 64 QAM symbols.

[0339] 6.3.4 Bit Probabilities

[0340] For an AWGN channel, the following expressions need to be evaluated for the I dimension:

$$LLR(u_1^k) = \log\left(\frac{\sum_{i=1}^{4}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{4}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0,j}^k)^2\right]}\right) = \log\left(\frac{\begin{array}{c}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_4)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_5)^2\right] + \\ \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_6)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_7)^2\right]\end{array}}{\begin{array}{c}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_0)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_1)^2\right] + \\ \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_2)^2\right] + \exp\left[\frac{1}{2\sigma_N^2}(I^k - A_3)^2\right]\end{array}}\right) \quad (54)$$

-continued

$$LLR(u_2^k) = \log\left(\frac{\sum_{i=1}^{4}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{4}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0,j}^k)^2\right]}\right) \tag{55}$$

$$= \log\left(\frac{\begin{array}{c}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_2)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_3)^2\right] + \\ \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_6)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_7)^2\right]\end{array}}{\begin{array}{c}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_0)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_1)^2\right] + \\ \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_4)^2\right] + \exp\left[\frac{1}{2\sigma_N^2}(I^k - A_5)^2\right]\end{array}}\right)$$

$$LLR(u_3^k) = \log\left(\frac{\sum_{i=1}^{4}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{1,j}^k)^2\right]}{\sum_{i=1}^{4}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - a_{0,j}^k)^2\right]}\right) \tag{56}$$

$$= \log\left(\frac{\begin{array}{c}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_1)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_5)^2\right] + \\ \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_3)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_7)^2\right]\end{array}}{\begin{array}{c}\exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_0)^2\right] + \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_4)^2\right] + \\ \exp\left[-\frac{1}{2\sigma_N^2}(I^k - A_2)^2\right] + \exp\left[\frac{1}{2\sigma_N^2}(I^k - A_6)^2\right]\end{array}}\right)$$

[0341] An identical computation effort is required for the Q dimension the I^k being replaced with the Q^k demodulated value in order to evaluate $LLR(u_4^k)$, $LLR(u_5^k)$ and $LLR(u_6^k)$.

[0342] 6.3.5 Simulation Results

[0343] **FIG. 24** shows the simulation results for 6,144 information bits with S-type interleaver. A BER of 10^{-7} can be achieved after 8 iterations at E_b/N_O=6.1 dB for N=4,096 information bits. This result is 0.3 dB worse than the performance of the rate 3/4 16 QAM scheme.

[0344] 6.4 Recommended Solution

[0345] Comparing **FIGS. 19, 22** and **24**, the recommended solution for 3 bits/Hz spectral efficiency is the rate 3/4 turbo code 16 QAM Scheme.

[0346] This scheme achieves a target BER of 10^{-7} at E_b/N_O=5.8 dB for N=6,144 information bits.

[0347] 7. Coding and Modulation for 4 bit/s/Hz Spectral Efficiency

[0348] This section investigated four schemes for transmission of 4 information bits in a 64 QAM symbol. The mapping used for 64 QAM constellation has a very significant impact on the performance of these schemes. The first scheme uses independent I and Q mapping, and also uses Gray mapping in each dimension. The second scheme uses independent I and Q mapping, but natural mapping in each dimension. The third scheme used a conventional trellis coded modulation approach based on Ungerboeck set partitioning of the 64 QAM set. This partitioning technique splits the constellation in sub-constellations with increased Euclidian distance. In these schemes all the information bits are coded. The fourth scheme used the same conventional trellis coded modulation approach based on Ungerboeck set partitioning of the 64 QAM set. However, only two information bits are encoded by a rate 1/2 code. The four coded bits select a sub-partition of four points. The other two information bits, which are sent uncoded, identify the transmitted point.

[0349] 7.1 Option 1 —Rate 4/6 64 QAM with independent I and Q and with Gray Mapping

[0350] 7.1.1 Coding

[0351] The coding scheme is shown in **FIG. 3**. The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0352] 7.1.2 Puncturing

[0353] In accordance with an embodiment of the invention, in order to obtain a rate 4/6 code, the puncturing pattern used is shown in Table 12.

TABLE 12

Puncturing and Mapping for Rate 4/6 64 QAM Option 1

Information bit (d)	d_1	d_2	d_3	d_4
parity bit (p)	p1	—	—	—
parity bit (q)	—	—	q3	—
8 AM symbol (I)		(d_1, d_2, p1)		
8 AM symbol (Q)		(d_3, d_4, q3)		
64 QAM symbol (I, Q)		(I, Q) = (d_1, d_2, p1, d_3, d_4, q3)		

[0354] 7.1.3 Modulation

[0355] A 64 QAM scheme is shown in **FIG. 23**. Gray mapping was used in each dimension as shown in **FIG. 21**.

[0356] Four information bits are required to be sent using a 64 QAM constellation.

[0357] For a rate 4/6 code and 64 QAM, the noise variance in each dimension is

[0358] For a rate 4/6 code and 64 QAM, the noise variance in each dimension is

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right)^{-1} = 42A^2\left(\frac{2\times 4\times E_b}{N_0}\right)^{-1} = 5.25A^2\left(\frac{E_b}{N_0}\right)^{-1} \tag{57}$$

[0359] Gray mapping was used in each dimension as shown in **FIG. 21**.

[0360] The puncturing and mapping scheme is shown in Table 12 for 4 consecutive information bits that are encoded into 6 coded bits, therefore one 64 QAM symbol.

[0361] The turbo encoder with the puncturing presented in Table 12 is a rate 4/6 turbo code which in conjunction with 64 QAM gives a spectral efficiency of 4 bits/s/Hz.

[0362] Considering two independent Gaussian noises with identical variance σ^2_N, the LLR can be determined independently for each I and Q.

[0363] It is assumed that at time u_1^k, u_2^k and u_3^k modulates the I component and u_4^k, u_5^k and u_6^k modulates the Q component of the 64 QAM scheme.

[0364] At the receiver, the I and Q signals are treated independently in order to take advantage of the simpler formulae for the LLR values.

[0365] 7.1.4 Bit Probabilities

[0366] From each received symbol, the bit probabilities are computed as described in equations (46) (47) and (48) for I dimension. An identical computation effort is required for the Q dimension, the I^k being replaced with the Q^k demodulated value in order to evaluate $LLR(u_4^k)$, $LLR(u_5^k)$ and $LLR(u_6^k)$.

[0367] 7.1.5 Simulation Results

[0368] **FIG. 25** shows the simulation results for 4,096 information bits with S-type interleaver.

[0369] A search for the best S-type interleaver was performed. For each interleaver size, the maximum S-factor was found and four different interleavers were generated using a random seed. Four simulations were generated for a target BER of 10^{-5}. The best result decided the preferred interleaver. Using the preferred interleaver, complete simulations were started for BERs down to 10^{-7}.

[0370] A BER of 10^{-7} can be achieved after 8 iterations at E_b/N_O=8.3 dB.

[0371] 7.2 Option 2 —Rate 4/6 64 QAM with independent I and Q and Natural Mapping

[0372] 7.2.1 Coding

[0373] The proposed coding scheme is shown in **FIG. 3**. The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0374] 7.2.2 Puncturing

[0375] In accordance with an embodiment of the invention, in order to obtain a rate 4/6 code, the puncturing pattern used is shown in Table 13.

TABLE 13

Puncturing and Mapping for Rate 4/6 64 QAM Option 2

Information bit (d)	d_1	d_2	d_3	d_4
parity bit (p)	p1	—	—	—
parity bit (q)	—	—	q3	—
8 AM symbol (I)		(d_1, d_2, p1)		
8 AM symbol (Q)		(d_3, d_4, q3)		
64 QAM symbol (I, Q)		(I, Q) = (d_1, d_2, p1, d_3, d_4, q3)		

[0376] 7.2.3 Modulation

[0377] A 64 QAM scheme is shown in **FIG. 23**. Natural mapping was used in each dimension as shown in **FIG. 26**. Four information bits are required to be sent using a 64 QAM constellation. This is equivalent to a rate 2/3 coding scheme.

[0378] For a rate 4/6 code and 64 QAM, the noise variance in each dimension is

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right)^{-1} = 42A^2\left(\frac{2\times 4\times E_b}{N_0}\right)^{-1} = 5.25A^2\left(\frac{E_b}{N_0}\right)^{-1} \quad (58)$$

[0379] Gray mapping is used in each dimension as shown in **FIG. 21**.

[0380] The puncturing and mapping scheme is shown in Table 13 for 4 consecutive information bits that are encoded into 6 coded bits, therefore one 64 QAM symbol.

[0381] The turbo encoder with the puncturing presented in Table 13 is a rate 4/6 turbo code which in conjunction with 64 QAM gives a spectral efficiency of 4 bits/s/Hz.

[0382] Considering two independent Gaussian noises with identical variance σ^2_N, the LLR can be determined independently for each I and Q.

[0383] It is assumed that at time k u_1^k, u_2^k, and u_3^k modulates the I component and u_4^k, u_5^k and u_6^k modulates the Q component of the 64 QAM scheme.

[0384] At the receiver, the I and Q signals are treated independently in order to take advantage of the simpler formulae for the LLR values.

[0385] 7.2.4 Bit Probabilities

[0386] From each received symbol, the bit probabilities are computed as described in equations (46) (47) and (48) for I dimension. An identical computation effort is required for the Q dimension, the I^k being replaced with the Q^k demodulated value in order to evaluate $LLR(u_4^k)$, $LLR(u_5^k)$ and $LLR(u_6^k)$.

[0387] 7.2.5 Simulation Results

[0388] **FIG. 27** shows the simulation results for 4,096 information bits with S-type interleaver.

[0389] A search for the best S-type interleaver was performed. For each interleaver size, the maximum S-factor was found and four different interlevers were generated using a random seed. Four simulations were generated for a target BER of 10^{-5}. The best result decided the preferred interleaver. Using the preferred interleaver, complete simulations were started for BERs down to 10^{-7}.

[0390] A BER of 10^{-7} can be achieved after 8 iterations at E_b/N_O=10.5 dB.

[0391] 7.3 Option 3 —Trellis Coded Modulation with 4 bits Coded

[0392] 7.3.1 Coding

[0393] In this scheme, all four information bits are coded by a rate 4/6 code. Only two parity bits are transmitted. The six bits produced select a point in the 64 QAM constellation. The coding scheme is shown in **FIG. 3**. The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0394] 7.3.2 Puncturing

[0395] In accordance with an embodiment of the invention, in order to obtain a rate 2/3 code, the puncturing pattern used is shown in Table 14.

TABLE 14

Puncturing and Mapping for Rate 4/6 64 QAM Option 3				
Information bit (d)	d_1	d_2	d_3	d_4
parity bit (p) p1	—	—	—	
parity bit (q)	—	—	q3	—
64 QAM symbol (I, Q)	(I, Q) = (d_1, d_2, d_3, d_4, p1, q3)			

[0396] 7.3.3 Modulation

[0397] A trellis coded modulation scheme is employed. The six bits divide the 64 QAM constellation based on increased Euclidean distance.

[0398] The 64QM constellation is partitioned in subsets as shown in FIGS. **28** to **32**.

[0399] The puncturing and mapping scheme is shown in Table 14 for 4 consecutive information bits that are encoded into 6 coded bits, therefore one 64 QAM symbol.

[0400] 7.3.4 Bit Probabilities

[0401] For an AWGN channel, the following expressions need to be evaluated for each received symbol before the turbo decoding process can start.

$$LLR(_{u_1}^{k}) = \log\left(\frac{\sum_{u_1^k=1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_i\|\right)}{\sum_{u_1^k=0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_j\|\right)}\right) \quad (59)$$

$$LLR(_{u_2}^{k}) = \log\left(\frac{\sum_{u_2^k=1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_i\|\right)}{\sum_{u_2^k=0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_j\|\right)}\right) \quad (60)$$

$$LLR(_{u_3}^{k}) = \log\left(\frac{\sum_{u_3^k=1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_i\|\right)}{\sum_{u_3^k=0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_j\|\right)}\right) \quad (61)$$

$$LLR(_{u_4}^{k}) = \log\left(\frac{\sum_{u_4^k=1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_i\|\right)}{\sum_{u_4^k=0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_j\|\right)}\right) \quad (62)$$

$$LLR(_{u_5}^{k}) = \log\left(\frac{\sum_{u_5^k=1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_i\|\right)}{\sum_{u_5^k=0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_j\|\right)}\right) \quad (63)$$

-continued

$$LLR(_{u_6}^{k}) = \log\left(\frac{\sum_{u_6^k=1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_i\|\right)}{\sum_{u_6^k=0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_j\|\right)}\right) \quad (51)$$

[0402] The $\|R^k-P_i\|$ represents the squared Euclidian distance between the received point R^k at the time k and a constellation point P_i. The actual set in which u_i is 0 or 1 are shown in FIGS. **33-38**.

[0403] 7.3.5 Simulation Results

[0404] This scheme required much higher computational effort than previous options and would be difficult to implement in hardware. **FIG. 39** shows the simulation results for 4,096 information bits using S-type interleaver.

[0405] A BER of 10^{-7} can be achieved after 8 iterations at E_b/N_O=11.5 dB.

[0406] 7.4 Option 4—Trellis Coded Modulation with 2 bits Coded.

[0407] 7.4.1 Coding

[0408] In this scheme, only two information bits are coded by a rate 2/4 code. The other two information bits are sent uncoded. The four coded bits (two information bits plus two parity bits) selects a four point constellation (16 constellations in total) and the two uncoded bits select a point in the constellation (four points in each constellation). The coding scheme is shown in **FIG. 3**. The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0409] 7.4.2 Puncturing

[0410] In accordance with an embodiment of the invention, in order to obtain a rate 1/2 code, every other parity bit is punctured as shown in Table 15.

TABLE 15

Puncturing and Mapping for Rate 1/2 64 QAM Option 4				
Information bit (d)	d_1	d_2	d_3	d_4
parity bit (p)	p1	—	—	—
parity bit (q)	—	q2	—	—
64 QAM symbol (I, Q)	(I, Q) = (d_1, p1, d_3, d_2, q2, d_4)			

[0411] 7.4.3 Modulation

[0412] A trellis coded modulation scheme is employed. The four coded bits divide the 64 QAM constellation based on increased Euclidean distance.

[0413] The 64 QM constellation shown in **FIG. 39** is partitioned by the four coded bits in 16 subsets with four point each. The two uncoded bits will select one of the four points of the subset. FIGS. **41** to **44** show the four steps of the partitioning process. Each 16 points constellation subset can now be further partitioned as show in FIGS. **41** to **44**.

[0414] The puncturing and mapping scheme is shown in Table 15 for 4 consecutive information bits that are encoded into 6 coded bits, therefore one 64 QAM symbol.

[0415] 7.4.4 Bit Probabilities

[0416] For an AWGN channel, the following expressions need to be evaluated for each received symbol before the turbo decoding process can start.

$$LLR(_{u_1}^{k}) = \log\left(\frac{\sum_{u_1^k=1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_i\|\right)}{\sum_{u_1^k=0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_j\|\right)}\right) \tag{64}$$

$$LLR(_{u_2}^{k}) = \log\left(\frac{\sum_{u_2^k=1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_i\|\right)}{\sum_{u_2^k=0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_j\|\right)}\right) \tag{65}$$

$$LLR(_{u_3}^{k}) = \log\left(\frac{\sum_{u_3^k=1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_i\|\right)}{\sum_{u_3^k=0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_j\|\right)}\right) \tag{66}$$

$$LLR(_{u_4}^{k}) = \log\left(\frac{\sum_{u_4^k=1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_i\|\right)}{\sum_{u_4^k=0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - P_j\|\right)}\right) \tag{67}$$

[0417] Each summation in equations (64) (65) (66) and (67) is over 32 points

[0418] The $\|R^k-P_i\|$ represents the squared Euclidian distance between the received point R^k at the time k and a constellation point P_i. The actual set in which u_i is 0 or 1 are shown in FIGS. **41-44**.

[0419] 7.4.5 Simulation Results

[0420] **FIG. 45** shows the simulation results for 4,096 bit blocks with 2,048 bit S-type interleaver.

[0421] A BER of 10^{-7} can be achieved after 8 iterations at E_b/N_O=11.5 dB.

[0422] 7.5 Recommended Solution

[0423] Comparing the complexity of each scheme and their performance, the recommended solution for 4 bit/s/Hz spectral efficiency is Option 1, the rate 4/6 turbo code 64 QAM scheme independent I and Q with gray mapping in each dimension. This achieves the target BER at E_b/N_O=8.3 dB. For N=4096 information bits.

[0424] 8. Coding and Modulation for 5 bit/s/Hz Spectral Efficiency.

[0425] This section investigated three schemes all use independent I and Q modulation. The first scheme combines a rate 5/6 coding scheme with 64 QAM. The second scheme combines a rate 5/7 coding scheme with 128 QAM. The third scheme combines a rate 5/8 coding scheme with 256 QAM.

[0426] 8.1 Option I—Rate 5/6 Turbo Code and 64 QAM.

[0427] 8.1.1 Coding

[0428] The coding scheme is shown in **FIG. 3**. The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0429] 8.1.2 Puncturing

[0430] In accordance with an embodiment of the invention, in order to obtain a rate 5/6 code, the puncturing pattern used is shown in Table 16.

TABLE 16

Puncturing and Mapping for Rate 5/6 64 QAM Option 1

Information bit (d)	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}
parity bit (p)	p1	—	—	—	—	—	—	—	—	—
parity bit (q)	—	—	—	—	—	q6	—	—	—	—
8 AM symbol (I)	(d_1, d_2, P1)					(d_6, d_7, q6)				
8 AM symbol (Q)	(d_3, d_4, d_5)					(d_8, d_9, d_{10})				
64 QAM symbol (I, Q)	(I, Q) = (d_1, d_2, p1, d_3, d_4, d_5)					(I, Q) = (d_6, d_7, q6, d_8, d_9, d_{10})				

[0431] 8.1.3 Modulation

[0432] A 64 QAM scheme is shown in **FIG. 23**.

[0433] Five information bits are required to be sent using a 64 QAM constellation. This is equivalent to a rate 5/6 coding scheme.

[0434] For a rate 5/6 code, the noise variance in each dimension is

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right)^{-1} = 42A^2\left(\frac{2\times 5\times E_b}{N_0}\right)^{-1} = 4.2A^2\left(\frac{E_b}{N_0}\right)^{-1} \tag{68}$$

[0435] The puncturing and mapping scheme is shown in Table 16 for 10 consecutive information bits that are encoded into 12 coded bits, therefore two 64 QAM symbols. The turbo encoder with the puncturing presented in Table 16 is a rate 5/6 turbo code which in conjunction with 64 QAM gives a spectral efficiency of 5 bits/s/Hz.

[0436] Considering two independent Gaussian noises with identical variance σ^2_N, the LLR can be determined independently for each I and Q.

[0437] It is assumed that at time k u_1^k, u_2^k and u_3^k modulates the I component and u_4^k, u_5^k and u_6^k modulates the Q component of the 64 QAM scheme.

[0438] At the receiver, the I and Q signals are treated independently in order to take advantage of the simpler formulae for the LLR values.

[0439] 8.1.4 Bit Probabilities

[0440] From each received symbol, the bit probabilities are computed as described in equations (54) (55) and (56) for I dimension. An identical computation effort is required

for the Q dimension, the I^k being replaced with the Q^k demodulated value in order to evaluate $LLR(u_4^k)$, $LLR(u_5^k)$ and $LLR(u_6^k)$.

[0441] 8.1.5 Simulation Results

[0442] **FIG. 46** shows the simulation results for 5,120 information bits (1,204 QAM symbols) with S-type interleaver.

[0443] The high puncturing rate makes the iterative decoding process to converge very slowly, showing a flattening of the BER curve. Therefore this option is not considered acceptable.

[0444] 8.2 Option 2—Rate 5/7 Turbo Code and 128 QAM.

[0445] 8.2.1 Coding

[0446] The coding scheme is shown in **FIG. 3**. The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0447] 8.2.2 Puncturing

[0448] In accordance with an embodiment of the invention, in order to obtain a rate 5/7 code, the puncturing pattern used is shown in Table 17.

TABLE 17

Puncturing and Mapping for Rate 5/7 128 QAM Option 2

Information bit (d)	d_1	d_2	d_3	d_4	d_5
parity bit (p)	p1	—	—	—	
parity bit (q)	—	—	q3	—	
16 AM symbol (I)	(d_3, d_4, d_5, q3)				
8 AM symbol (Q)	(d_1, d_2, p1)				
128 QAM symbol (I, Q)	(I, Q) = (d_3, d_4, d_5, q3, d_1, d_2, p1)				

[0449] 8.2.3 Modulation

[0450] Let us consider the 128 QAM scheme as shown in **FIG. 47**. It is assumed that at time k the symbol $u^k=(u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k)$ is sent though the channel. It is assumed that at time k the symbol u_1^k, u_2^k, u_3^k, and u_4^k modulate the I component and $u5^k u_6^k$ and u_7^k modulate the Q component of a 128 QAM scheme.

[0451] In the I dimension, three information bits and one parity bits are transmitted using Gray mapping and 16 QAM as show in **FIG. 48**.

[0452] The average energy The E_{av_I} for the 16 QAM shown in **FIG. 48** is:

$$\text{The } E_{av_I} \text{ is:} \quad E_{av_I} = \frac{(1+9+25+49+81+121+169+225)A^2}{8} = 85A^2 \tag{69}$$

$$\text{The } E_{av_Q} \text{ is:} \quad E_{av_Q} = \frac{(1+9+25+49)A^2}{4} = 21A^2 \tag{70}$$

[0453] Because the noise has to be spherical, the total value of E_{av} will be the addition of these two values, so:

$$E_{av}=E_{av_I}+E_{av_Q}=106A^2 \tag{71}$$

[0454] The value of the noise variance has to be the same in both dimensions and its value is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right) = 106A^2\left(\frac{2\times 5\times E_b}{N_0}\right)^{-1} = 10.6A^2\left(\frac{E_b}{N_0}\right)^{-1} \tag{72}$$

[0455] The puncturing and mapping scheme shown in Table 17 is for 5 consecutive information bits that are coded into 7 coded bits, therefore, one 128 QAM symbol. The turbo encoder is a rate 5/7 turbo code, which in conjunction with 128 QAM, gives a spectral efficiency of 5 bits/s/Hz.

[0456] 8.2.4 Bit Probabilities

[0457] The 16 QAM symbol is defined as $u^k=(u_1^k, u_2^k, u_3^k, u_4^k)$, where u_1^k is the most significant bit and u_4^k is the least significant bit. The following set can be defined. 1. bit-1-is-0=$\{A_0, A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$2. bit-1-is-1=$\{A_8, A_9, A_{10}, A_{11}, A_{12}, A_{13}, A_{14}, A_{15}\}$3. bit-2-is-0=$\{A_0, A_1, A_2, A_3, A_8, A_9, A_{10}, A_{11}\}$4. bit-2-is-1 =$\{A_4, A_5, A_6, A_7, A_{12}, A_{13}, A_{14}, A_{15}\}$5. bit-3-is-0=$\{A_0, A_1, A_4, A_5, A_8, A_9, A_{12}, A_{13}\}$6. bit-3-is-1=$\{A_2, A_3, A_6, A_7, A_{10}, A_{11}, A_{14}, A_{15}\}$7. bit-4-is-0=$\{A_0, A_2, A_4, A_6, A_8, A_{10}, A_{12}, A_{14}\}$8. bit-4-is-1=$\{A_1, A_3, A_5, A_7, A_9, A_{11}, A_{13}, A_{15}\}$

[0458] From each received symbol, R^k, the bit probabilities are computed as follows:

$$LLR(_{u_1}^k) = \log\left(\frac{\sum_{A_i\in bit-1-is-1}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_i\|\right)}{\sum_{A_j\in bit-1-is-0}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_j\|\right)}\right) \tag{73}$$

$$LLR(_{u_2}^k) = \log\left(\frac{\sum_{A_i\in bit-2-is-1}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_i\|\right)}{\sum_{A_j\in bit-2-is-0}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_j\|\right)}\right) \tag{74}$$

$$LLR(_{u_3}^k) = \log\left(\frac{\sum_{A_i\in bit-3-is-1}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_i\|\right)}{\sum_{A_j\in bit-3-is-0}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_j\|\right)}\right) \tag{75}$$

$$LLR(_{u_4}^k) = \log\left(\frac{\sum_{A_i\in bit-4-is-1}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_i\|\right)}{\sum_{A_j\in bit-4-is-0}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_j\|\right)}\right) \tag{76}$$

[0459] Similar formulae apply for the 8 AM symbol defined as $u^k=(u_5^k, u_6^k, u_7^k)$, where u_5^k is the most significant bit and u_7^k is the least significant bit.

[0460] 8.2.5 Simulation Results

[0461] **FIG. 49** shows the simulation results for 5,120 information bits (1,204 QAM symbols) with S-type interleaver.

[0462] The high puncturing rate makes the iterative decoding process to converge very slowly, showing a flattening of the BER curve.

[0463] 8.3 Option 3—Rate 5/8 Turbo Code and 256 QAM

[0464] 8.3.1 Coding

[0465] The coding scheme is shown in **FIG. 3**. The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0466] 8.3.2 Puncturing

[0467] In accordance with an embodiment of the invention, in order to obtain a rate 5/8 code, the puncturing pattern used is shown in Table 18.

TABLE 18

Puncturing and Mapping for Rate 5/8 256 QAM Option 3

Information bit (d)	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}
parity bit (p)	p1	—	—	—	p5	—	—	p8	—	—
parity bit (q)	—	—	q3	—		q6	—	—	—	q10
16 AM symbol (I)	(d_1, d_2, d_3, p1)					(d_6, d_7, d_8, q6)				
16 AM symbol (Q)	(d_4, d_5, q3, p5)					(d_9, d_{10}, p8, q10)				
256 QAM symbol (I, Q)	(I, Q) = (d_1,d_2, d_3, p1, d_4, d_5, q3, p5)					(I, Q) = (d_6, d_7, d_8, q6, d_9, d_{10}, p8, q10)				

[0468] 8.3.3 Modulation

[0469] Let us consider the 256 QAM scheme as shown in **FIG. 50**

[0470] For a 256 QAM constellation with points at −15A, −13A, −11A, −9A, −7A, −5A, −3A, −A, A, 3A, 5A 7A, 9A, 11A, 13A, 15A. E_{av} is:

$$E_{av}=(8(225+169+121+81+49+25+9+1)+8(225+169+225+121+225+81+225+49+225+25+225+9+225+1)+8(169+121+169+81+169+49+169+25+169+9+169+1)+8(121+81+121+49+121+25+121+9+121+1)+8(81+49+25+81+9+81+1)+8(49+25+49+9+49+1)+8(25+9+25+1)+8(9+1))A^2/256=170A^2 \quad (77)$$

[0471] It is assumed that at time k the symbol $u^k=(u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k, u_7^k, u_8^k)$ is sent though the channel.

[0472] It is assumed that at time k the symbol u_1^k, u_2^k, u_3^k and u_4^k modulate the I component and u_5^k, u_6^k, u_7^k and u_8^k modulate the Q component of a 256 QAM scheme.

[0473] For a rate 5/8 code and 256 QAM, the noise variance is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right)^{-1} = 170A^2\left(\frac{2\times 5\times E_b}{N_0}\right)^{-1} = 17A^2\left(\frac{E_b}{N_0}\right)^{-1} \quad (78)$$

[0474] In order to study the performance of this scheme, a rate 5/8 turbo code and a 16 AM is used as describe in **FIG. 50**. The 256 QAM scheme will achieve a similar performance in terms of bit error rate (BER) at twice the spectral efficiency, assuming an ideal demodulator.

[0475] The E_{av} for the 16 QAM shown in **FIG. 48** is:

$$E_{av_1}=(1+9+25+49+81+121+169+225)A^2/8=85A^2 \quad (79)$$

[0476] The noise variance is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right)^{-1} = 85A^2\left(\frac{2\times 2.5\times E_b}{N_0}\right)^{-1} = 17A^2\left(\frac{E_b}{N_0}\right)^{-1} \quad (80)$$

[0477] This is identical to the variance for 256 QAM scheme.

[0478] The puncturing and mapping scheme shown in Table 18 is for 10 consecutive information bits that are coded into 16 encoded bits, therefore, one 256 QAM symbol.

[0479] The turbo encoder is a rate 5/8 turbo code, which in conjunction with 256 QAM, gives a spectral efficiency of 5 bits/s/Hz.

[0480] 8.3.4 Bit Probabilities

[0481] The 16 AM symbol is defined as $u^k=(u_1^k, u_2^k, u_3^k, u_4^k)$, where u_1^k is the most significant bit and u_4^k is the least significant bit . The following set can be defined.

[0482] From each received symbol, R^k, the bit probabilities are computed as follows:

$$LLR(_{u_1}^k) = \log\left(\frac{\sum_{A_i\in bit-1-is-1}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_i\|\right)}{\sum_{A_j\in bit-1-is-0}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_j\|\right)}\right) \quad (81)$$

$$LLR(_{u_2}^k) = \log\left(\frac{\sum_{A_i\in bit-2-is-1}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_i\|\right)}{\sum_{A_j\in bit-2-is-0}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_j\|\right)}\right) \quad (82)$$

$$LLR(_{u_3}^k) = \log\left(\frac{\sum_{A_i\in bit-3-is-1}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_i\|\right)}{\sum_{A_j\in bit-3-is-0}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_j\|\right)}\right) \quad (83)$$

$$LLR(_{u_4}^k) = \log\left(\frac{\sum_{A_i\in bit-4-is-1}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_i\|\right)}{\sum_{A_j\in bit-4-is-0}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k-A_j\|\right)}\right) \quad (84)$$

[0483] 8.3.5 Simulation Results

[0484] **FIG. 51** shows the simulation results for 5,120 information bits (1,204 QAM symbols) with S-type interleaver.

[0485] A search for the best S-type interleaver was performed. For each interleaver size, the maximum S-factor was found and four different interleavers were generated using a random seed. Four simulations were generated for a target BER of 10^{-5}. The best result decided the preferred interleaver. Using the preferred interleaver, complete simu-

lations were started for BER down to 10^{-7}. A BER of 10^{-7} can be achieved after 8 iterations at E_b/N_O=11.8 dB.

[0486] 8.4 Recommended Solution

[0487] The recommended solution for 5 bit/s/Hz spectral efficiency is the rate 5/8 turbo code 256 QAM scheme independent I and Q with gray mapping in each dimension. This achieves the target BER at E_b/N_O=11.8 dB. For N=5, 120 information bits.

[0488] 9. Coding and Modulation for 6 bit/Hz Spectral Efficiency

[0489] This section investigated two schemes all use independent I and Q modulation. The first scheme combines a rate 6/8 coding scheme with 256 QAM. The second scheme combines a rate 6/9 coding scheme with 512 QAM.

[0490] 9.1 Option 1—Rate 6/8 Turbo Code and 256 OAM

[0491] 9.1.1 Coding

[0492] The coding scheme is shown in **FIG. 3**. The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0493] 9.1.2 Puncturing

[0494] In accordance with an embodiment of the invention, in order to obtain a rate 6/8 code, the puncturing pattern used is shown in Table 19.

TABLE 19

Puncturing and Mapping for Rate 6/8 256 QAM Option 1

Information bit (d)	d_1	d_2	d_3	d_4	d_5	d_6
parity bit (p)	p1	—	—	—	—	—
parity bit (q)	—	—	—	q4	—	—
16 AM symbol (I)	(d_1, d_2, d_3, p1)					
16 AM symbol (Q)	(d_4, d_5, d_6, q4)					
256 QAM symbol (I, Q)	(I, Q) = (d_1, d_2, d_3, p1, d_4, d_5, d_6, q4)					

[0495] 9.1.3 Modulation

[0496] Let us consider a 256 QAM scheme as shown in **FIG. 50**. It is assumed that at time k the symbol $u^k=(u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k, u_7^k, u_8^k)$ is sent though the channel. It is assumed that at time k the symbol u_1^k, u_2^k, u_3^k and u_4^k modulate the I component and u_5^k, u_6^k, u_7^k and u_8^k modulate the Q component of a 256 QAM scheme.

[0497] For a rate 6/8 code and 256 QAM, the noise variance is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right)^{-1} = 170A^2\left(\frac{2\times 6\times E_b}{N_0}\right)^{-1} = 14.16A^2\left(\frac{E_b}{N_0}\right)^{-1} \tag{85}$$

[0498] The E_{av} in the I and Q dimension is the 16 QAM shown in **FIG. 47** is:

$$E_{av_I}=(1+9+25+49+81+121+169+225)A^2/8=85A^2 \tag{86}$$

[0499] The noise variance is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right)^{-1} = 85A^2\left(\frac{2\times 3\times E_b}{N_0}\right)^{-1} = 14.16A^2\left(\frac{E_b}{N_0}\right)^{-1} \tag{87}$$

[0500] This is identical to the variance for 256 QAM scheme.

[0501] The puncturing and mapping scheme shown in Table 19 is for 6 consecutive information bits that are coded into 8 coded bits, therefore, one 256 QAM symbol. The turbo encoder is a rate 6/8 turbo code, which in conjunction with 256 QAM, gives a spectral efficiency of 6 bits/s/Hz.

[0502] 9.1.4 Bit Probabilities

[0503] The 16 AM symbol is defined as $u^k=(u_1^k, u_2^k, u_3^k, u_4^k)$, where u_1^k is the most significant bit and u_4^k is the least significant bit. The following set can be defined.

[0504] 1. bit-**1**-is-0={A_0, A_1, A_2, A_3, A_4, A_5, A_6, A_7}

[0505] 2. bit-**1**-is-1={A_8, A_9, A_{10}, A_{11}, A_{12}, A_{13}, A_{14}, A_{15}}

[0506] 3. bit-**2**-is-0={A_0, A_1, A_2, A_3, A_8, A_9, A_{10}, A_{11}}

[0507] 4. bit-**2**-is-1={A_4, A_5, A_6, A_7, A_{12}, A_{13}, A_{14}, A_{15}}

[0508] 5. bit-**3**-is-0={A_0, A_1, A_4, A_5, A_8, A_9, A_{12}, A_{13}}

[0509] 6. bit-**3**-is-1={A_2, A_3, A_6, A_7, A_{10}, A_{11}, A_{14}, A_{15}}

[0510] 7. bit-**4**-is-0={A_0, A_2, A_4, A_6, A_8, A_{10}, A_{12}, A_{14}}

[0511] 8. bit-**4**-is-1={A_1, A_3, A_5, A_7, A_9, A_{11}, A_{13}, A_{15}}

[0512] From each received symbol, R^k, the bit probabilities are computed as follows:

$$LLR(u_1^k) = \log\left(\frac{\sum_{A_i\in bit-1-is-1}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_i\in bit-1-is-0}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \tag{88}$$

$$LLR(u_2^k) = \log\left(\frac{\sum_{A_i\in bit-2-is-1}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_i\in bit-2-is-0}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \tag{89}$$

$$LLR(u_3^k) = \log\left(\frac{\sum_{A_i\in bit-3-is-1}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_i\in bit-3-is-0}\exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \tag{90}$$

-continued

$$LLR(u_4^k) = \log\left(\frac{\sum_{A_i \in bit-4-is-1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_i \in bit-4-is-0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \quad (91)$$

[0513] 9.1.5 Simulation Results

[0514] **FIG. 52** shows the simulation results for 6,144 information bits (1,024 QAM symbols) with S-type interleaver.

[0515] A search for the best S-type interleaver was performed. For each interleaver size, the maximum S-factor was found and four different interleavers were generated using a random seed. Four simulations were generated for a target BER of 10^{-5}. The best result decided the preferred interleaver. Using the preferred interleaver, complete simulations were started for BER down to 10^{-7}.

[0516] A BER of 10^{-7} can be achieved after 8 iterations at E_b/N_O=14.2 dB.

[0517] 9.2 Option 2—Rate 6/9 Turbo Code and 512 QAM

[0518] 9.2.1 Coding

[0519] The coding scheme is shown in **FIG. 3**. The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0520] 9.2.2 Puncturing

[0521] In accordance with an embodiment of the invention, in order to obtain a rate 6/9 code, the puncturing pattern used is shown in Table 20.

TABLE 20

Puncturing and Mapping for Rate 6/9 512 QAM Option 2

Information bit (d)	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}
parity bit (p)	p1	—	—	—	p5	—	—	—	p9	—	—	—
parity bit (q)	—	—	q3	—	—	—	q7	—	—	—	q11	—
32 AM symbol (I)	(d_1, d_2, d_3, d_4, p1)						(d_7, d_8, d_9, d_{10}, q7)					
16 AM symbol (Q)	(d_5, d_6, p5, q3)						(d_{11}, d_{12}, Q11, p9)					
512 QAM symbol (I, Q)	(I, Q) = (d_1, d_2, d_3, d_4, p1, d_5, d_6, p5, q3)						(I, Q) = (d_7, d_8, d_9, d_{10}, q7, d_{11}, d_{12}, q11, P9)					

[0522] 9.2.3 Modulation

[0523] Let us consider a 512 QAM scheme as shown in **FIG. 52**. It is assumed that at time k the symbol $u^k=(u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k, u_7^k, u_8^k, u_9^k)$ is sent though the channel. It is assumed that at time k the symbol u_1^k, u_2^k, u_3^k, u_4^k and u_5^k modulates the I component and u_6^k, u_7^k, u_8^k and u_9^k modulates the the Q component of a 512 QAM scheme.

[0524] In the I dimension, four information bits and one parity bit are transmitted using Gray mapping and 32 AM as shown in **FIG. 53**.

[0525] The 32 constellation points have amplitudes: −31A, −29A, −27A, −25A, −23A, −21A, −19A, −17A, −15A, −13A, −11A, −9A, −7A, −5A, −3A, −A, A, 3A, 5A, 7A, 9A, 11A, 13A, 15A, 17A, 19A, 21A, 23A, 25A, 27A, 29A, 31A. The E_{av_I} is:

$$E_{av_Q} = \frac{(1+9+25+49+81+121+169+225)A^2}{8} = 85_{A^2} \quad (93)$$

[0526] Because the noise has to be spherical, the total value of E_{av} will be the addition of these two values, so:

$$E_{av}=E_{av_I}+E_{av_Q}=426\,A^2 \quad (94)$$

[0527] The value of the noise variance has to be the same in both dimensions and its value is:

$$\sigma_{N_1}^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right)^{-1} = 341A^2\left(\frac{2\times 6\times E_b}{N_0}\right) = \frac{341}{12}A^2\left(\frac{E_b}{N_0}\right)^{-1} \quad (95)$$

[0528] The puncturing and mapping scheme shown in Table 20 is for 12 consecutive information bits that are coded into 18 coded bits, therefore, one 512 QAM symbol. The turbo encoder is a rate 6/9 turbo code, which in conjunction with 512 QAM, gives a spectral efficiency of 6 bits/s/Hz.

[0529] 9.2.4 Bit Probabilities

[0530] From each received symbol, R^k, the bit probabilities are computed as follows:

$$LLR(u_1^k) = \log\left(\frac{\sum_{A_i \in bit-1-is-1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_i \in bit-1-is-0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \quad (96)$$

$$LLR(u_2^k) = \log\left(\frac{\sum_{A_i \in bit-2-is-1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_i \in bit-2-is-0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \quad (97)$$

$$LLR(u_3^k) = \log\left(\frac{\sum_{A_i \in bit-3-is-1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_i \in bit-3-is-0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \quad (98)$$

$$LLR(u_4^k) = \log\left(\frac{\sum_{A_i \in bit-4-is-1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_i \in bit-4-is-0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \quad (99)$$

-continued

$$LLR(u_5^k) = \log\left(\frac{\sum_{A_i \in bit-5-is-1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_j \in bit-5-is-0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \quad (99)$$

[0531] 9.2.5 Simulation Results

[0532] **FIG. 54** shows the simulation results for 6,144 information bits (1,204 QAM symbols) with S-type interleaver.

[0533] A BER of 10^{-6} I can be achieved after 8 iterations at E_b/N_O=13.5 dB. A BER of 10^{-7} can be achieved after 8 iterations at E_b/N_O=14.5 dB.

[0534] 9.3 Recommended Solution

[0535] The recommended solution for 6 bit/s/Hz spectral efficiency is the rate 6/8 turbo code 256 QAM scheme. This scheme achieves a BER of 10^{-7} at E_b/N_O=14.2 dB.

[0536] 10. Coding and Modulation for 7 bit/Hz Spectral Efficiency

[0537] This section investigated one scheme that use independent I and Q modulation. The scheme combines a rate 7/10 coding scheme with 1024 QAM.

[0538] 10.1 Coding

[0539] The coding scheme is shown in **FIG. 3**. The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0540] 10.2 Puncturing

[0541] In accordance with an embodiment of the invention, in order to obtain a rate 7/10 code, the puncturing pattern used is shown in Table 21.

TABLE 21

Puncturing and Mapping for Rate 7/10 1024 QAM

Information bit (d)	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}
parity bit (p)	p1	—	—	—	—	p6	—	—	—	—	p11	—	—	—
parity bit (q)	—	—	q3	—	—	—	—	q8	—	—	—	—	q13	—
32 AM symbol (I)	(d_1, d_2, d_3, p1, q3)							(d_8, d_9, d_{10}, d_{11}, q8)						
32 AM symbol (Q)	(d_4, d_5, d_6, d_7, p6)							d_{12}, d_{13}, d_{14}, p11, q13)						
1024 QAM symbol (I, Q)	(I, Q) = (d_1, d_2, d_3, p1, q3, d_4, d_5, d_6, d_7, p6)							(I, Q) = (d_8, d_9, d_{10}, d_{11}, q8, d_{12}, d_{13}, d_{14}, p11, q13)						

[0542] 10.3 Modulation

[0543] Let us consider a 1024 QAM scheme as shown in **FIG. 55**. It is assumed that at time k the symbol $u^k=(u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k, u_7^k, u_8^k, u_9^k, u_{10}^k)$ is sent though the channel. It is assumed that at time k the symbol u_1^k, u_2^k, u_3^k, u_4^k and u_5^k modulates the I component and u_6^k, u_7^k, u_8^k, u_9^k and u_{10}^k modulates the Q component of a 1024 QAM scheme.

[0544] For a 1024 QAM constellation with points at −31A, −29A, −27A, −25A, −23A, −21A, −19A, −17A, −15A, −13A, −11A, −9A, −7A, −5A, −3A, −A, A, 3A, 5A, 7A, 9A, 11A, 13A, 15A, 17A, 19A, 21A, 23A, 25A, 27A, 29A, 31A. E_{av} is:

$$\begin{aligned}E_{av}=&8(961+841+729+625+529+441+361+289+225+\\&169+121+81+49+25+9+1)\\&+8(961+841+961+729+961+625+961+529+961+441+961+\\&361+961+289+961+225+961+169+961+121+961+81+961\\&+49+961+25+961+9+961+1)+8(841+729+841+625+841+\\&5\quad 29+841+441+841+361+841+289+841+225+841\\&+169+841+121+841+81+841+49+841+25+841+9+841+1)\\&8(729+625+729+529+729+441+729+361+729+289+729\\&+225+729+169+729+121+729+81+729+49+729+25+729+9+7\ 29+\\&1)8(625+529+625+441+625+361+625+289+625+225\\&+625+169+625+121+625+81+625+49+625+25+625+9+625+1)\\&+8(529+441+529+361+529+289+529+225+529+169+529+121+529\\&+81+529+49+529+25+529+9+529+1)+8(441+361+441+289+441+\\&225+441+169+441+121+441+81+441+49+441+25+441+9+441+1)\\&8(361+289+361+225+361+169+361+121+361+529+9+529+1)\\&+8(441+361+441+289+441+225+441+169+441+121+441+81+441\\&+49+441+25+441+9+441+1)8(361+289+361+225+361+169+361\\&+121+361+81+361+49+361+25+361+9+361+1)+8(289+225\\&+289+169+289+121+289+81+289+49+289+25+289+9+289+1)\\&8(225+169+225+121+225+81+225+49+225+25+225+9\\&+225+1)8(169+121+169+81+169+49+169+25+169+9+169\\&+1)+8(121+81+121+49+121+25+121+9+121+1)\\&+8(81+49+81+25+81+9+81+1)+8(49+25+49+9+49+\\&1)+8(25+9+25+1)+8(9+1)A^2/1024=682A^2\end{aligned} \quad (100)$$

[0545] For a rate 7/10 code and 512 QAM, the noise variance is:

$$\begin{aligned}\sigma_N^2 &= E_{av}\left(\frac{2\eta E_b}{N_0}\right)^{-1} = 682A^2\left(\frac{2\times 7\times E_b}{N_0}\right)^{-1} \\ &= 48.7A^2\left(\frac{E_b}{N_0}\right)^{-1}\end{aligned} \quad (101)$$

[0546] The 1024 QAM constellation is a product of two 32 constellation with points at: −31A, −29A, −27A, −25A, −23A, −21A, −19A, −17A, −15A, −13A, −11A, −9A, −9A, −7A, −5A, −3A, −A, A, 3A, 5A, 7A, 9A, 11A, 13A, 15A, 17A, 19A, 21A, 23A, 25A, 27A, 29A, 31A. The 32 QAM E_{av_I} is:

$$\begin{aligned}E_{av}=&(1+9+25+49+81+121+169+225+289+361+441+\\&529+625+729+841+961)\ A^2/16=341A^2\end{aligned} \quad (102)$$

[0547] For a rate 7/10 code and 1024 QAM, the noise variance is:

$$\begin{aligned}\sigma_N^2 &= E_{av}\left(\frac{2\eta E_b}{N_0}\right)^{-1} = 341A^2\left(\frac{2\times 3.5\times E_b}{N_0}\right)^{-1} \\ &= 48.7A^2\left(\frac{E_b}{N_0}\right)^{-1}\end{aligned} \quad (103)$$

[0548] This is identical to the variance for 1024 QAM scheme.

[0549] The puncturing and mapping scheme shown in Table 21 is for 14 consecutive information bits that are coded into 20 coded bits, therefore, two 1024 QAM symbols. The turbo encoder is a rate 7/10 turbo code, which in conjunction with 1024 QAM, gives a spectral efficiency of 7 bits/s/Hz.

[0550] 10.4 Bit Probabilities

[0551] The 32 AM symbol is defined as $u^k=(u_1^k, u_2^k, u_3^k, u_4^k, u_5^k)$, where u_1^k is the most significant bit and u_5^k is the least significant bit. The following set can be defined.

[0552] 1. bit-1-is-0={A_0, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9, A_{10}, A_{11}, A_{12}, A_{13}, A_{14}, A_{15}}

[0553] 2. bit-1-is-1={A_{16}, A_{17}, A_{18}, A_{19}, A_{20}, A_{21}, A_{22}, A_{23}, A_{24}, A_{25}, A_{26}, A_{27}, A_{28}, A_{29}, A_{30}, A_{31}}

[0554] 3. bit-2-is-0={A_0,A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_{16}, A_{17}, A_{18}, A_{19}, A_{20}, A_{21}, A_{22}, A_{23}}

[0555] 4. bit-2-is-1={A_8, A_9, A_{10}, A_{11}, A_{12}, A_{13}, A_{14}, A_{15}, A_{24}, A_{25}, A_{26}, A_{27}, A_{28}, A_{29}, A_{30}, A_{31}}

[0556] 5. bit-3-is-0={A_0, A_1, A_2, A_3, A_8, A_9, A_{10}, A_{11}, A_{16}, A_{17}, A_{18}, A_{19}, A_{24}, A_{25}, A_{26}, A_{27}}

[0557] 6. bit-3-is-1={A_4, A_5, A_6, A_7, A_{12}, A_{13}, A_{14}, A_{15}, A_{20}, A_{21}, A_{22}, A_{23}, A_{28}, A_{29}, A_{30}, A_{31}}

[0558] 7. bit-4-is-0={A_0, A_1, A_4, A_5, A_8, A_9, A_{12}, A_{13}, A_{16}, A_{17}, A_{20}, A_{21}, A_{24}, A_{25}, A_{28}, A_{29}}

[0559] 8. bit-4-is-1={A_2, A_3, A_6, A_7, A_{10}, A_{11}, A_{14}, A_{15}, A_{18}, A_{19}, A_{22}, A_{23}, A_{26}, A_{27}, A_{30}, A_{31}}

[0560] 9. bit-5-is-0={A_0, A_2, A_4, A_6, A_8, A_{10}, A_{12}, A_{14}, A_{16}, A_{18}, A_{20}, A_{22}, A_{24}, A_{26}, A_{28}, A_{30}}

[0561] 10. bit-5-is-1={A_1, A_3, A_5, A_7, A_9, A_{11}, A_{13}, A_{15}, A_{17}, A_{19}, A_{21}, A_{23}, A_{25}, A_{27}, A_{29}, A_{31}}

[0562] From each received symbol, R^k, the bit probabilities are computed as follows:

$$LLR(u_1^k) = \log\left(\frac{\sum_{A_i \in bit-1-is-1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_j \in bit-1-is-0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \quad (104)$$

$$LLR(u_2^k) = \log\left(\frac{\sum_{A_i \in bit-2-is-1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_j \in bit-2-is-0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \quad (105)$$

$$LLR(u_3^k) = \log\left(\frac{\sum_{A_i \in bit-3-is-1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_j \in bit-3-is-0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \quad (106)$$

$$LLR(u_4^k) = \log\left(\frac{\sum_{A_i \in bit-4-is-1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_j \in bit-4-is-0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \quad (107)$$

-continued

$$LLR(u_5^k) = \log\left(\frac{\sum_{A_i \in bit-5-is-1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_j \in bit-5-is-0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \quad (108)$$

[0563] 10.5 Simulation Results

[0564] **FIG. 56** shows the simulation results for 2,044 information bits with S-type interleaver.

[0565] A search for the best S-type interleaver was performed. For each interleaver size, the maximum S-factor was found and four different interleavers were generated using a random seed. Four simulations were generated for a target BER of 10^{-5}. The best result decided the preferred interleaver. Using the preferred interleaver, complete simulations were started for BER down to 10^{-7}.

[0566] A BER of 10^{-7} can be achieved after 8 iterations at E_b/N_O=17 dB.

[0567] 10.6 Recommended Solution

[0568] The recommended solution for 7 bit/s/Hz spectral efficiency is the rate 7/10 turbo code 1024 QAM scheme. This scheme achieves a BER of 10^{-7} at E_b/N_O=17 dB for N=2044 information bits.

[0569] 11. Coding and Modulation for 12 bit/Hz Spectral Efficiency

[0570] This section investigated one scheme that use independent I and Q modulation. The scheme combines a rate 12/14 coding scheme with 16384 QAM.

[0571] 11.1 Coding

[0572] The coding scheme is shown in **FIG. 3**. The two systematic recursive codes (SRC) used are identical and are defined in **FIG. 4**. The code is described by the generating polynomials 35o and 23o.

[0573] 11.2 Puncturing

[0574] In accordance with an embodiment of the invention, in order to obtain a rate 12/14 code, the puncturing pattern used is shown in Table 22.

TABLE 22

Puncturing and Mapping for Rate 12/14 16384 QAM

Information bit (d)	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}
parity bit (p)	p1	—	—	—	—	—	—	—	—	—	—	—
parity bit (q)	—	—	—	—	—	—	q7	—	—	—	—	—
128 AM symbol (I)	(d_1, d_2, d_3, d_4 d_5, d_6, p1)											
128 AM symbol (Q)	(d_7, d_8, d_9, d_{10}, d_{11}, d_{12}, q7)											
16384 QAM symbol (I, Q)	(d_1, d_2, d_3, d_4 d_5, d_6, p1, d_7, d_8, d_9, d_{10}, d_{11}, d_{12}, q7)											

[0575] 11.3 Modulation

[0576] For a 16384 QAM constellation with points at −127A, −125A, −123A, −121A, −119A, −117A, −115A,

−113A, −111A, −109A, −107A, −105A, −103A, −101A, −99A, −97A, −95A, −93A, −91A, −89A, −87A, −85A, −83A, −81A, −79A, −77A, −75A, −73A, −71A, −69A, −67A, −65A, −63A, −61A, −59A, −57A, −55A, −53A, −51A, −49A, −47A, −45A, −43A, −41A, −39A, −37A, −35A, −33A, −31A, −29A, −27A, −25A, −23A, −21A, −19A, −17A, −15A, −13A, −11A, −9A, −7A, −5A, −3A, −A, A, 3A, 5A, 7A, 9A, 11A, 13A, 15A, 17A, 19A, 21A, 23A, 25A, 27A, 29A, 31A, 33A, 35A, 37A, 39A, 41A, 43A, 45A, 47A, 49A, 51A, 53A, 55A, 57A, 59A, 61A, 63A, 65A, 67A, 69A, 71A, 73A, 75A, 77A, 79A, 81A, 83A, 95A, 87A, 89A, 91A, 93A, 95A, 97A, 99A, 101A, 103A, 105A, 107A, 109A, 111A, 113A, 115A, 117A, 119A, 121A, 123A, 125A, 127A. E_{av} is:

$$E_{av}=5461A^2 \tag{109}$$

[0577] It is assumed that at time k the symbol $u^k=(u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k, u_7^k, u_8^k, u_9^k, u_{10}^k, u_{11}^k, u_{12}^k, u_{13}^k, u_{14}^k)$ is sent though the channel. It is assumed that at time k the symbol u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k and u_7^k modulates the I component and u_8^k, u_9^k, u_{10}^k, u_{11}^k, u_{12}^k, u_{13}^k, u_{14}^k, modulates the Q component of a 16384 QAM scheme.

[0578] For a rate 12/14 code and 16384 QAM, the noise variance is:

$$\sigma_N^2 = E_{av}\left(\frac{2\eta E_b}{N_0}\right)^{-1} = 5461A^2\left(\frac{2\times 6\times E_b}{N_0}\right)^{-1} = 455.08A^2\left(\frac{E_b}{N_0}\right)^{-1} \tag{110}$$

[0579] In order to study the performance of this scheme, a rate 6/7 turbo code and a 128AM is used. The 16384 QAM scheme will achieve a similar performance in terms of bit error rate (BER) at twice the spectral efficiency, assuming an ideal demodulator. The puncturing and mapping scheme shown in Table 22 is for 12 consecutive information bits that are coded into 14 encoded bits, therefore, one 16384 QAM symbol. The turbo encoder is a rate 12/14 turbo code, which in conjunction with 16384 QAM, gives a spectral efficiency of 12 bits/s/Hz.

[0580] 11.4 Bit Probabilities

[0581] The 128AM symbol is defined as $u^k=(u_1^k, u_2^k, u_3^k, u_4^k, u_5^k, u_6^k, u_7^k)$, where u_1^k is the most significant bit and u_7^k is the least significant bit. The following set can be defined.

[0582] 1. bit-1-is-1={A_{64}, A_{65}, A_{66}, A_{67}, A_{68}, A_{69}, A_{70}, A_{71}, A_{72}, A_{73}, A_{74}, A_{75}, A_{76}, A_{77}, A_{78}, A_{79}, A_{80}, A_{81}, A_{82}, A_{83}, A_{84}, A_{85}, A_{86}, A_{87}, A_{88}, A_{89}, A_{90}, A_{91}, A_{92}, A_{93}, A_{94}, A_{95}, A_{96}, A_{97}, A_{98}, A_{99}, A_{100}, A_{101}, A_{102}, A_{103}, A_{104}, A_{105}, A_{106}, A_{107}, A_{108}, A_{109}, A_{110}, A_{111}, A_{112}, A_{113}, A_{114}, A_{115}, A_{116}, A_{117}, A_{118}, A_{119}, A_{120}, A_{121}, A_{122}, A_{123}, A_{124}, A_{125}, A_{126}, A_{127},}

[0583] 2. bit-2-is-1={A_{32}, A_{33}, A_{34}, A_{35}, A_{36}, A_{37}, A_{38}, A_{39}, A_{40}, A_{41}, A_{42}, A_{43}, A_{44}, A_{45}, A_{46}, A_{47}, A_{48}, A_{49}, A_{50}, A_{51}, A_{52}, A_{53}, A_{54}, A_{55}, A_{56}, A_{57}, A_{58}, A_{59}, A_{60}, A_{61}, A_{62}, A_{63}, A_{64}, A_{65}, A_{66}, A_{67}, A_{68}, A_{69}, A_{70}, A_{71}, A_{72}, A_{73}, A_{74}, A_{75}, A_{76}, A_{77}, A_{78}, A_{79}, A_{80}, A_{81}, A_{82}, A_{83}, A_{84}, A_{85}, A_{86}, A_{87}, A_{88}, A_{89}, A_{90}, A_{91}, A_{92}, A_{93}, A_{94}, A_{95},}

[0584] 3. bit-3-is-1={A_{16}, A_{17}, A_{18}, A_{19}, A_{20}, A_{21}, A_{22}, A_{23}, A_{24}, A_{25}, A_{26}, A_{27}, A_{28}, A_{29}, A_{30}, A_{31}, A_{32}, A_{33}, A_{34}, A_{35}, A_{36}, A_{37}, A_{38}, A_{39}, A_{40}, A_{41}, A_{42}, A_{43}, A_{44}, A_{45}, A_{46}, A_{47}, A_{80}, A_{81}, A_{82}, A_{83}, A_{84}, A_{85}, A_{86}, A_{87}, A_{88}, A_{89}, A_{90}, A_{91}, A_{92}, A_{93}, A_{94}, A_{95}, A_{96}, A_{97}, A_{98}, A_{99}, A_{100}, A_{101}, A_{102}, A_{103}, A_{104}, A_{105}, A_{106}, A_{107}, A_{108}, A_{109}, A_{110}, A_{111},}

[0585] 4. bit-4-is-1={A_8, A_9, A_{10}, A_{11}, A_{12}, A_{13}, A_{14}, A_{15}, A_{16}, A_{17}, A_{18}, A_{19}, A_{20}, A_{21}, A_{22}, A_{23}, A_{40}, A_{41}, A_{42}, A_{43}, A_{44}, A_{45}, A_{46}, A_{47}, A_{48}, A_{49}, A_{50}, A_{51}, A_{52}, A_{53}, A_{54}, A_{55}, A_{72}, A_{73}, A_{74}, A_{75}, A_{76}, A_{77}, A_{78}, A_{79}, A_{80}, A_{81}, A_{82}, A_{83}, A_{85}, A_{86}, A_{87}, A_{104}, A_{105}, A_{106}, A_{107}, A_{108}, A_{109}, A_{110}, A_{111}, A_{112}, A_{113}, A_{114}, A_{115}, A_{116}, A_{117}, A_{118}, A_{119},}

[0586] 5. bit-5-is-1={A_4, A_5, A_6, A_7, A_8, A_9, A_{10}, A_{11}, A_{20}, A_{21}, A_{22}, A_{23}, A_{24}, A_{25}, A_{26}, A_{27}, A_{36}, A_{37}, A_{38}, A_{39}, A_{40}, A_{41}, A_{42}, A_{43}, A_{52}, A_{53}, A_{54}, A_{55}, A_{56}, A_{57}, A_{58}, A_{59}, A_{68}, A_{69}, A_{70}, A_{71}, A_{72}, A_{73}, A_{74}, A_{75}, A_{84}, A_{85}, A_{86}, A_{87}, A_{88}, A_{89}, A_{90}, A_{91}, A_{100}, A_{101}, A_{102}, A_{103}, A_{104}, A_{105}, A_{106}, A_{107}, A_{116}, A_{117}, A_{118}, A_{119}, A_{120}, A_{121}, A_{122}, A_{123},}

[0587] 6. bit-6-is-1={A_0, A_1, A_6, A_7, A_8, A_9, A_{14}, A_{15}, A_{16}, A_{17}, A_{22}, A_{23}, A_{24}, A_{25}, A_{30}, A_{31}, A_{32}, A_{33}, A_{38}, A_{39}, A_{40}, A_{41}, A_{46}, A_{47}, A_{48}, A_{49}, A_{54}, A_{55}, A_{56}, A_{57}, A_{62}, A_{63}, A_{64}, A_{65}, A_{70}, A_{71}, A_{72}, A_{73}, A_{78}, A_{79}, A_{80}, A_{81}, A_{86}, A_{87}, A_{88}, A_{89}, A_{94}, A_{95}, A_{96}, A_{97}, A_{102}, A_{103}, A_{104}, A_{105}, A_{110}, A_{111}, A_{112}, A_{113}, A_{118}, A_{119}, A_{120}, A_{121}, A_{126}, A_{127},}

[0588] 7. bit-7-is-1={A_1, A_2, A_5, A_6, A_9, A_{10}, A_{13}, A_{14}, A_{17}, A_{18}, A_{21}, A_{22}, A_{25}, A_{26}, A_{29}, A_{30}, A_{33}, A_{34}, A_{37}, A_{38}, A_{42}, A_{45}, A_{46}, A_{49}, A_{50}, A_{53}, A_{54}, A_{57}, A_{58}, A_{61}, A_{62}, A_{65}, A_{66}, A_{69}, A_{70}, A_{73}, A_{74}, A_{77}, A_{78}, A_{81}, A_{82}, A_{85}, A_{86}, A_{89}, A_{90}, A_{93}, A_{94}, A_{97}, A_{98}, A_{101}, A_{102}, A_{105}, A_{106}, A_{109}, A_{110}, A_{113}, A_{114}, A_{117}, A_{118}, A_{121}, A_{122}, A_{125}, A_{126},}

[0589] From each received symbol, R^k, the bit probabilities are computed as follows:

$$LLR(u_n^k) = \log\left(\frac{\sum_{A_i\in bit-n-is-1} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_i\|\right)}{\sum_{A_j\in bit-n-is-0} \exp\left(-\frac{1}{2\sigma_N^2}\|R^k - A_j\|\right)}\right) \tag{111}$$

[0590] 11.5 Simulation Results

[0591] **FIG. 57** shows the simulation results for 31200 information bits. A BER of 10^{-7} can be achieved after 8 iterations at E_b/N_O=28.25 dB.

[0592] 12. Power vs. Bandwidth in an AWGN Channel

[0593] This section gives an estimate of the trade off which can be achieved between minimum required E_b/N_O and bandwidth efficiency. An information data rate of 2,044 Mbit/s and a maximum transmitter delay of 1 ms is considered. The corresponding interleaver size is 2,044 bits.

[0594] 12.1 Channel Model

[0595] All the simulations assumed the additive white Gaussian noise (AWGN) channel model, with independent I and Q signal. A block diagram of the system is shown in **FIG. 57**.

[0596] 12.2 Simulation Results

[0597] Simulations were run for bandwidth efficiencies from 1 to 7 bit/symbol using the recommended coding and modulation schemes. The results are shown in FIGS. **58** to **64**.

[0598] **12.3** Conclusions

[0599] Table 23 summarizes the minimum E_b/N_O required to achieve a BER of 10^{-7}.

TABLE 23

Minimum E_b/N_0 required to achieve a BER of 10^{-7}			
Spectral efficiency η [bits/s/Hz]	Coding Rate and Modulation	Symbol Rate [ksym/s]	E_b/N_0 For BER = 10^{-7} [dB]
1	2/4 and 4 QAM	2048	2.2
2	2/4 and 16 QAM	1024	4.2
3	3/4 and 16 QAM	682	6.5
4	4/6 and 64 QAM	512	9.1
5	5/8 and 256 QAM	408	12.3
6	6/8 and 256 QAM	342	14.5
7	7/10 and 1024 QAM	292	17.0

[0600] The results show the potential reduction in bandwidth for a given signal-to-noise ration for a particular channel. If an E_b/N_O of 17 dB or more is available, the symbol rate can be reduced from 2048 ksym/s to 292 ksym/s.

[0601] 13. Channel Simulation

[0602] In this paragraph we show numerical results of the simulations for the AWGN and Impulsive nose using or not and outer Reed-Solomon encoder.

[0603] 13.1 Without outer Reed-Solomon Encoder

[0604] 13.1.1 Net Coding Gain

[0605] The net coding gain of the spectral efficiency of 4 bit/tone and 12 bits per tone, protecting more the information bits than the parity bits is as shown in Table 24. The case that the parity bits are more protected than the information bits is shown in table 25. for a BER greater than 10^{-8} the net coding gain is 0.5 dB worse. For BER below 10^{-8} the coding gain improves respect the case that the information bits are more protected.

TABLE 24

Net Coding Gain Information bits more protected							
Bit/ Tone	Tones	Inter-leaver Size	# of DMT symbols	Latency (Tx + Rx) ms <	10^{-3}	10^{-7}	10^{-9} extrap.
4	100	5,200	13	10.0	4.60	7.42	7.94
		800	2	1.5	3.70	4.92	4.84
		400	1	0.7	3.30	3.62	3.84
	200	10,400	13	10.0	4.60	7.52	8.14
		1,600	2	1.5	4.10	6.42	6.64
		800	1	0.7	3.70	4.92	4.84
12	100	15,600	13	10.0	4.10	5.91	6.03
		2,400	2	1.5	3.60	5.51	5.63
		1,200	1	0.7	3.00	3.91	4.03
	200	31,200	13	10.0	4.10	6.81	7.53
		4,800	2	1.5	3.60	5.91	6.43
		2,400	1	0.7	3.60	5.51	5.63

[0606]

TABLE 25

Net Coding Gain parity more protected							
Bit/ Tone	Tones	Inter-leaver Size	# of DMT symbols	Latency (Tx + Rx) ms <	10^{-3}	10^{-7}	10^{-9} extrap.
4	100	5,200	13	10.0	4.10	7.22	8.04
		800	2	1.5	3.20	4.72	5.04
		400	1	0.7	2.80	3.42	4.04
	200	10,400	13	10.0	4.10	7.32	8.24
		1,600	2	1.5	3.60	6.22	6.84
		800	1	0.7	3.20	4.72	5.04
12	100	15,600	13	10.0	3.60	5.71	6.23
		2,400	2	1.5	3.10	5.31	5.83
		1,200	1	0.7	2.50	3.71	4.23
	200	31,200	13	10.0	3.60	6.61	7.73
		4,800	2	1.5	3.10	5.71	6.63
		2,400	1	0.7	3.10	5.31	5.83

[0607] 13.1.2 Errors due to Impulse noise (IN)

[0608] The impulse noise is defined as 2 consecutive DMT symbols with an increase AWGN respect to the reference noise level 10^{-7} a carrier-to-noise ratio of 21.5 dB (spectral efficiency of 4 bit/tone) and 45.5 dB (spectral efficiency of 12 bit/tone).

[0609] Table 26 shows the numerical results in the case that the information bits are more protected than the parity bits. Table 27 shows the numerical results in the case that the parity bits are more protected than the information bits.

TABLE 26

Error due to Impulse Noise. Information bits more protected										
Bit/ Tone	Tones	Interleaver Size	RL + 2.5 dB	RL + 5 dB	RL + 7.5 dB	RL + 10 dB	RL + 12.5 dB	RL + 15 dB	RL + 17.5 dB	RL + 20d B
4	100	5,200	0	0	0	0	0	0	0	4
		800	0	0	39	65	104	140	188	243
		400	0	0	10	50	89	127	161	214
	200	10,400	0	0	0	0	0	0	0	7
		1,600	0	0	0	127	189	267	363	448
		800	0	0	40	116	187	252	346	440

TABLE 26-continued

Error due to Impulse Noise. Information bits more protected

Bit/ Tone	Tones	Interleaver Size	RL + 2.5 dB	RL + 5 dB	RL + 7.5 dB	RL + 10 dB	RL + 12.5 dB	RL + 15 dB	RL + 17.5 dB	RL + 20d B
12	100	15,600	0	0	0	0	10	58	130	207
		2,400	0	0	40	78	121	171	21	295
		1,200	0	0	43	98	129	188	255	329
	200	31,200	0	0	0	0	90	175	313	482
		4,800	0	0	75	177	254	341	462	608
		2,400	0	0	80	166	244	345	457	598

[0610]

TABLE 27

Error due to Impulse Noise. Parity bits more protected

Bit/ Tone	Tones	Interleaver Size	RL + 10 dB	RL + 15 dB	RL + 20 dB	RL + 25 dB	RL + 30 dB	RL + 35 dB	RL + 40 dB	RL + 45 dB
4	100	5,200	0	0	0	0	0	0	0	2
		800	0	0	0	45	84	100	118	143
		400	0	0	0	20	69	87	101	114
	200	10,400	0	0	0	0	0	0	0	3
		1,600	0	0	0	27	49	67	83	118
		800	0	0	0	16	37	52	86	120
12	100	15,600	0	0	0	0	0	8	10	15
		2,400	0	0	0	18	31	51	60	125
		1,200	0	0	0	18	29	38	55	89
	200	31,200	0	0	0	0	0	15	33	43
		4,800	0	0	0	0	24	41	52	68
		2,400	0	0	0	16	24	34	45	59

[0611] 13.1.3 Error Statistics

[0612] 13.1.3.1 For AWGN

[0613] 10 Table 28 shows the statistics of the errors for the case of protecting more the information bits, for a AWGN channel.

TABLE 28

Error Statistics for AWGN. Information bits more protected

Bit/Tone	Tones	Interleaver Size	1 consec. error	2 consec errors	3 consec errors	4 consec errors	5 consec errors	6 consec errors
4	100	5,20	87.30%	10.81%	1.47%	0.29%	0.03%	0.10%
		80	94.35%	5.64%	0.00%	0.00%	0.00%	0.00%
		40	90.28%	9.72%	0.01%	0.00%	0.00%	0.00%
	200	10,40	89.90%	8.63%	1.21%	0.20%	0.06%	0.00%
		1,60	97.94%	2.06%	0.00%	0.00%	0.00%	0.00%
		80	90.28%	9.72%	0.01%	0.00%	0.00%	0.00%
12	100	15,60	99.79%	0.21%	0.00%	0.00%	0.00%	0.00%
		2,40	98.72%	1.28%	0.00%	0.00%	0.00%	0.00%
		1,20	97.94%	2.06%	0.00%	0.00%	0.00%	0.00%
	200	31,20	99.86%	0.14%	0.00%	0.00%	0.00%	0.00%
		4,80	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%
		2,40	98.72%	1.28%	0.00%	0.00%	0.00%	0.00%

[0614] 13.1.3.2 Impulse Noise

[0615] Table 29 shows the statistics of the errors for the case of protecting more the information bits for a Impulse noise channel.

TABLE 29

Error Statistics for Impulse noise. Information bits more protected

Bit/Tone	Tones	Interleaver Size	1 consec. error	2 consec errors	3 consec errors	4 consec errors	5 consec errors	6 consec errors
4	100	5,20	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%
		80	75.97%	18.99%	18.99%	3.36%	0.84%	0.84%
		40	79.89%	17.24%	1.72%	1.15%	0.00%	0.00%
	200	10,40	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%
		1,60	80.24%	13.05%	4.47%	1.68%	0.47%	0.00%
		80	79.03%	17.50%	2.46%	0.46%	0.46%	0.00%
12	100	15,60	95.19%	4.81%	0.00%	0.00%	0.00%	0.00%
		2,40	94.61%	5.28%	0.11%	0.00%	0.00%	0.00%
		1,20	93.63%	5.95%	93.63%	0.00%	0.00%	0.00%
	200	31,20	93.25%	6.65%	0.00%	0.00%	0.00%	0.00%
		4,80	94.89%	4.95%	0.16%	0.00%	0.00%	0.00%
		2,40	94.59%	5.36%	0.06%	0.00%	0.00%	0.00%

[0616] It is interesting that for the large turbo decoders the impulse errors still tends to stay within the 2 DMT symbols. This implies a moderately large turbo coder of 5 ms follow by a convolutional interleaver/Reed Solomon of 10 ms should create both robust performance and good impulse resistance.

[0617] 13.2 With Reed-Solomon

[0618] 13.2.1 Net Coding Gain

[0619] The coding gain of the spectral efficiency of 4 bit/tone and 12 bits per tone, protecting more the information bits than the parity bits is as shown in Table 30. The net coding gain is shown in table 31.

TABLE 30

Coding Gain with Reed-Solomon and information bits more protected

Bit/ Tone	Tones	Inter-leaver Size	# of DMT symbols	Latency (Tx + Rx) ms <	10^{-3}	10^{-7}	10^{-9} extrap.
4	100	5,200	13	10.0	5.00	8.62	9.64
		800	2	1.5	3.50	7.12	8.44
		400	1	0.7	3.50	6.42	7.44
	200	10,400	13	10.0	5.30	8.82	9.84
		1,600	2	1.5	4.60	7.72	8.74
		800	1	0.7	3.50	7.12	8.44
12	100	15,600	13	10.0	4.40	7.71	8.53
		2,400	2	1.5	4.60	7.41	8.33
		1,200	1	0.7	4.10	6.81	7.63
	200	31,200	13	10.0	4.40	7.71	8.53
		4,800	2	1.5	4.40	7.21	8.13
		2,400	1	0.7	4.60	7.41	8.33

[0620]

TABLE 31

Coding Gain with Reed-Solomon and information bits more protected

Bit/ Tone	Tones	Inter-leaver Size	# of DMT symbols	Latency (Tx + Rx) ms <	10^{-3}	10^{-7}	10^{-9} extrap.
4	100	5,200	13	10.0	3.42	7.04	8.06
		800	2	1.5	1.78	5.40	6.72
		400	1	0.7	1.94	4.86	5.88
	200	10,400	13	10.0	3.72	7.24	8.26
		1,600	2	1.5	2.88	6.00	7.02
		800	1	0.7	1.94	5.56	6.88
12	100	15,600	13	10.0	0.20	3.51	4.33
		2,400	2	1.5	0.02	2.83	3.75
		1,200	1	0.7	−0.24	2.47	3.29
	200	31,200	13	10.0	1.06	4.37	5.19
		4,800	2	1.5	0.76	3.57	4.49
		2,400	1	0.7	0.26	3.07	3.99

[0621] 13.2.2 Errors due to Impulse noise

[0622] The impulse noise is defined as 2 consecutive DMT symbols with an increase AWGN respect to the reference noise level 10^{-7} of a carrier-to-noise ratio of 21.5 dB (spectral efficiency of 4 bit/tone) and 45.5 dB (spectral efficiency of 12 bit/tone).

[0623] Table 32 shows the numerical results in the case that the information bits are more protected than the parity. Table 33 shows the numerical results in the case that the parity bits are more protected than the information bits.

TABLE 32

Error due to Impulse Noise. Information more protected

Bit/ Tone	Tones	Interleaver Size	RL + 2.5 dB	RL + 5 dB	RL + 7.5 dB	RL + 10 dB	RL + 12.5 dB	RL + 15 dB	RL + 17.5 dB	RL + 20 dB
4	100	5,200	0	0	0	0	0	0	0	0
		800	0	0	0	0	0	0	0	0
		400	0	0	0	0	0	0	0	0
	200	10,400	0	0	0	0	0	0	0	0
		1,600	0	0	0	0	0	0	0	0
		800	0	0	0	0	0	0	0	0
12	100	15,600	0	0	0	0	10	58	130	207
		2,400	0	0	0	0	0	0	0	0
		1,200	0	0	0	0	0	0	0	0
	200	31,200	0	0	0	0	90	175	313	482
		4,800	0	0	0	0	0	10	11	65
		2,400	0	0	0	0	9	10	24	115

[0624]
***** WRI file not found

[0625] 13.2.3 Error Statistics.

[0626] The statistics results obtained are practically the same than for the non Reed-Solomon case

[0627] 14 Simulation using an analytical interleaver

[0628] In this section we present the same simulation of the previous section, but using an analytical interleaver.

[0629] The analytical interleaver proposed here is a helical odd-even smile interleaver.

[0630] This section gives an estimate of the trade off which can be achieved between minimum required E_b/N_O and bandwidth efficiency. An information data rate of 2,100 kbps and a maximum transmitter delay of 1 ms is considered. The corresponding interleaver size is 2,100 bits (multiple of 2, 4, 6, 10, 14).

[0631] 14.1 Channel model

[0632] All the simulations assumed the additive white Gaussian noise (AWGN) channel model, with independent I and Q signals. A block diagram of the system is shown in **FIG. 57**.

[0633] 14.2 Simulation Results

[0634] Simulations were run for bandwidth efficiencies from 1 to 7 bit/symbol using the recommended coding and modulation schemes. The results are shown in FIGS. **65** to **71**.

[0635] 14.3 Conclusions

[0636] Table 24 summarizes the minimum E_b/N_O required to achieve a BER of 10^{-7}.

TABLE 24

Minimum E_b/N_0 required to achieve a BER of 10^{-7}

Spectral efficiency η [bits/s/Hz]	Coding Rate and Modulation	Symbol Rate [ksym/s]	E_b/N_0 For BER = $10^{31\ 7}$ [dB]
1	2/4 and 4 QAM	2048	2.7
2	2/4 and 16 QAM	1024	4.7

TABLE 24-continued

Minimum E_b/N_0 required to achieve a BER of 10^{-7}

Spectral efficiency η [bits/s/Hz]	Coding Rate and Modulation	Symbol Rate [ksym/s]	E_b/N_0 For BER = $10^{31\ 7}$ [dB]
3	3/4 and 16 QAM	682	6.75
4	4/6 and 64 QAM	512	9.6
5	5/8 and 256 QAM	408	12.8
6	6/8 and 256 QAM	342	15.0
7	7/10 and 1024 QAM	292	17.5
12	12/14 and 16384 QAM	170	28.5

[0637] The results show the potential reduction in bandwidth for a given signal-to-noise ration for a particular channel. If an E_b/N_O of 17 dB or more is available, the symbol rate can be reduced from 2048 ksym/s to 292 ksym/s.

[0638] Due to the savings in memory and the low degradation in this kind of interleaver for a BER of 10^{-6} below 0.5 dB, we recommend the use of this helical smile odd-even interleaver for these kind of applications.

[0639] 1.5 Conclusions

TABLE 25

Summary of Simulation Results

Spectral efficiency η [bits/s/Hz]	Coding Rate	Modulation	Interleaver size Information bits	Required E_b/N_0 [dB] S-type interleaver	Required E_b/N_0 [dB] Analytical interleaver
2/3	2/6	4 QAM	1,024	1.2	1.7
1	2/4	4 QAM	1,024	2.1	2.6
2	4/6	8 QAM	1,024	5.5	6.1
2	1/2	16 QAM	256	6.8	7.2
2	1/2	16 QAM	272	7.0	7.5
2	1/2	16 QAM	512	5.3	5.8
2	1/2	16 QAM	768	4.9	5.3
2	1/2	16 QAM	1,024	4.5	5.0
2	1/2	16 QAM	2,048	4.2	4.7
2	1/2	16 QAM	32,728	2.9	3.3
3	3/4	16 QAM	2,048	6.5	7.0
3	3/4	16 QAM	6,144	5.6	6.1
3	3/5	32 QAM	6,144	6.0	6.5
3	3/6	64 QAM	6,144	6.1	6.6

TABLE 25-continued

Summary of Simulation Results

Spectral efficiency η [bits/s/Hz]	Coding Rate	Modulation	Interleaver size Information bits	Required E_b/N_0 [dB] S-type interleaver	Required E_b/N_0 [dB] Analytical interleaver
4	4/6	64 QAM	2,048	9.1	9.6
4	4/6	64 QAM-1	4,096	8.3	8.8
4	4/6	64 QAM-2	4,096	10.5	11.0
4	4/6	64 QAM-3	4,096	11.5	12.0
4	4/6	64 QAM-4	4,096	11.5	12.0
5	5/6	64 QAM	5,120	13.0	13.5
5	5/7	128 QAM	5,120	13.0	13.5
5	5/8	256 QAM	2,048	12.3	12.8
5	5/8	256 QAM	5,120	11.8	12.3
6	6/8	256 QAM	2,048	14.5	15.0
6	6/8	256 QAM	6,144	14.2	14.7
6	6/9	512 QAM	6,144	14.5	15.0
7	7/10	1024 QAM	2,044	17.0	—
12	12/14	16384 QAM	31,200	28.0	28.5

[0640] 16. Computational Complexity saving

[0641] To fix ideas lets use the 16 QAM case.

[0642] Using independent I and Q with the Gray mapping, the constellation will looks like the one shown in **FIG. 13**.

[0643] The conventional technique for extraction LLR soft-decision information from the channel is create a value representing the probability of the received symbol being a one as:

$$\frac{\sum \text{ of the measures with the transmit symbol was } 0}{\sum \text{ of the measures with the transmit symbol was } 1} \tag{112}$$

[0644] where the measure is defined as:

$$e^{(-n*\text{metric})} \tag{113}$$

[0645] where:

[0646] metric=Euclidian distance (or square of the Euclidian distance) from the possible transmit symbol to the received symbol.

[0647] In Figure Receiving the point x if we do not use an independent I and Q technique, we have to compute the 8 exponentials (4 in each dimension) that is the distance of the point x to the four point in the I dimension and to the 4 points in the Q dimension as shown (x1, x2, x3, x4, y1, y2, y3, y4). To compute the 4 LLR probabilities we need 4*14 adds and 4*16 multiplications and 4 divisions. The total number of operations is 132.

$$\frac{\sum_{bit=1}}{\sum_{bit=0}} = \frac{e^{(x_1^2+y_1^2)} + e^{(x_1^2+y_2^2)} + e^{(x_1^2+y_3^2)} + e^{(x_1^2+y_4^2)} + e^{(x_4^2+y_1^2)} + e^{(x_4^2+y_2^2)} + e^{(x_4^2+y_3^2)} + e^{(x_4^2+y_4^2)}}{e^{(x_2^2+y_1^2)} + e^{(x_2^2+y_2^2)} + e^{(x_2^2+y_3^2)} + e^{(x_2^2+y_4^2)} + e^{(x_3^2+y_1^2)} + e^{(x_3^2+y_2^2)} + e^{(x_3^2+y_3^2)} + e^{(x_3^2+y_4^2)}} \tag{114}$$

-continued

$$= \frac{(e^{x_1^2} + e^{x_4^2})e^{(y_1^2+y_2^2+y_3^2+y_4^2)}}{(e^{x_2^2} + e^{x_3^2})e^{(y_1^2+y_2^2+y_3^2+y_4^2)}}$$

$$= \frac{e^{x_1^2} + e^{x_4^2}}{e^{x_2^2} + e^{x_3^2}}$$

[0648] With the independent I and Q if we want to compute the LLR of the least significant bit,

[0649] With the independent I and Q if we want to compute the LLR of the second-least significant bit,

$$\frac{\sum_{bit=1}}{\sum_{bit=0}} = \frac{(e^{x_1^2} + e^{x_2^2})e^{(y_1^2+y_2^2+y_3^2+y_4^2)}}{(e^{x_3^2} + e^{x_4^2})e^{(y_1^2+y_2^2+y_3^2+y_4^2)}} = \frac{e^{x_1^2} + e^{x_2^2}}{e^{x_3^2} + e^{x_4^2}} \tag{115}$$

$$\frac{\sum_{bit=1}}{\sum_{bit=0}} = \frac{(e^{y_1^2} + e^{y_2^2})e^{(x_1^2+x_2^2+x_3^2+x_4^2)}}{(e^{y_2^2} + e^{y_3^2})e^{(x_1^2+x_2^2+x_3^2+x_4^2)}} = \frac{e^{y_1^2} + e^{y_4^2}}{e^{y_2^2} + e^{y_3^2}} \tag{116}$$

[0650] With the independent I and Q if we want to compute the LLR of the most significant bit,

$$\frac{\sum_{bit=1}}{\sum_{bit=0}} = \frac{(e^{y_1^2} + e^{y_2^2})e^{(x_1^2+x_2^2+x_3^2+x_4^2)}}{(e^{y_3^2} + e^{y_4^2})e^{(x_1^2+x_2^2+x_3^2+x_4^2)}} = \frac{e^{y_1^2} + e^{y_2^2}}{e^{y_3^2} + e^{y_4^2}} \tag{117}$$

[0651] The same reduction of calculations occurs for all bits.

[0652] The number of exponentials to compute is the same, 8, the number of additions is 4*2=8, the number of multiplications is 0 and the number of divisions is 4. The total number of operations is 20.

[0653] When the constellation size increase the saving in computations also increase considerably.

[0654] For the case of 16 QAM, 64 QAM, 256 QAM and 16384 QAM the comparison is as follows:

[0655]

	#exp	#add	#mul	#div	TOTAL
16 QAM					
Full	8	4*14	16	4	84
Independent I & Q	8	4*2	0	4	20
64 QAM					
Full	16	6*62	64	6	458
Independent I & Q	16	6*6	0	6	58
128 QAM					
Full	32	7*126	121	7	1,042
Independent I & Q	32	7*14 + 2*14	14	14	186
256 QAM					
Full	32	8*254	256	8	2,328
Independent I & Q	32	8*14	0	8	152
16384 QAM					
Full	256	14*16382	16384	14	246,002
Independent I & Q	256	14*126	0	14	2,034
		N odd			
$N = 2^n$ QAM					
Full	$2(2N)^{1/2}$	$n(N-2)$	$N-n$	n	$2(2N)^{1/2} + (n+1)N - 2n$
Independent I & Q	$2(2N)^{1/2}$	$2n^2 + 4n$	$2n$	$2n$	$2(2N)^{1/2} + 2n^2 + 8n$
		N even			
$N = 2^n$ QAM					
Full	$2N^{1/2}$	$n(N-2)$	N	n	$2N^{1/2 + (n+1)N - n}$
Independent I & Q	$2N^{1/2}$	$n(N^{1/2} - 2)$	0	n	$(n+2)N^{1/2} + n$

[0656] The increase complexity for this type of constellation can be shown to be of $O((N)^{1/2})$ where N is the number of constellation points. For the odd constellation the procedure defined in Provisional Patent Application Serial No. 60/248,099 has been used.

[0657] 17. Use of Puncturing Patterns and Protection on Parity/Information Bits

[0658] In conventional Turbo Codes, when no comment about puncturing is made, it is understood that the puncturing pattern used selects one parity bit from the first encoder as one encoded bit and parity bit from the second encoder as the next encoded bit in an alternating fashion. Thus in the conventional Turbo Code using a four bit constellation, two parity bits are selected for every two information bits, and as such the number of parity bits represented in each symbol is equal to the number of information bits represented in the symbol.

[0659] In accordance with embodiments of the invention, for the constellations selected, one or two parity bits are sent in each dimension, providing, for this reason, the possibility to protect the parity bit(s) more than the information bit(s), or less than the information bit(s), because of the Gray mapping. The number of parity bits represented in each symbol is less than the number of information bits represented in the symbol.

[0660] In an AWGN channel, if the information bits are more protected, for BER higher than 10^{-7}, the performance is 0.5 dB better. This means that for the same BER, the E_b/N_O in the case that the information bits are more protected, is 0.5 dB lower than the E_b/N_o if the parity bits are more protected. For BER below 10^{-7}, this statement is reversed. For the same BER, the E_b/N_o, in the case that the information bits are more protected, is 0.5 dB higher that the E_b/N_o if the parity bits are more protected.

[0661] In an Impulse Noise environment, the greater protection of the parity bits provides more immunization against impulses (around 5 to 10 dB more). Once the impulse affects the more protected parity, they are more harmful than the case where the parity is less protected and the information is more protected.

[0662] **FIG. 74** shows the performance of the 6/4 rate 64 QAM for the case that the parity bits are most protected and for the case that the parity bits are least protected.

[0663] A detail explanation of the encoding and puncturing procedures is shown in **FIG. 75**, where matrix |A|, is the total set of bits that are encoded. The number of rows of this matrix is the number of input symbols to be transmitted, "m". The number of rows of this matrix is the number of bits encoded in each symbol, "n".

$$|A| = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}; A_j = \{a_{j,1}, a_{j,2}, \ldots, a_{j,n}\} \tag{118}$$

[0664] Each row of the matrix |A| is sent to the Turbo Encoder once every symbol time. For symbol j, the input to

the turbo encoder is $|A_j|=\{a_{j,1}, a_{j,2}, \ldots, a_{j,n}\}$. For each symbol, the output of the first convolutional encoder will be $|P_i|=p_{i,1}, p_{i,2}, \ldots, p_{i,n}$, and the output of the second convolutional encoder will be $|Q_j|=q_{j,1}, q_{j,2}, \ldots, p_{j,n}$. Each encoded bit $a_{j,i}$, produces one parity bit in the first encoder $p_{j,i}$, and one parity bit in the second encoder, $q_{j,i}$. Essentially the encoder runs "n"times for each symbol.

$$|P| = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ \vdots & \vdots & \vdots & \vdots \\ p_{m,1} & p_{m,2} & \cdots & p_{m,n} \end{pmatrix}; \quad P_j = \{p_{j,1}, p_{j,2}, \ldots, p_{j,n}\} \tag{119}$$

$$|Q| = \begin{pmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,n} \\ \vdots & \vdots & \vdots & \vdots \\ q_{m,1} & q_{m,2} & \cdots & q_{m,n} \end{pmatrix}; \quad Q_j = \{q_{j,1}, q_{j,2}, \ldots, q_{j,n}\} \tag{120}$$

[0665] The output of the first convolutional encoder and the output of the second convolutional encoders go to the puncturing pattern block, as shown in **FIG. 75**. This puncturing pattern block, selects the parity bits to be transmitted. The first output provides the parity bits to be transmitted from the first encoder $|P'_j|=P'_{j,1}P'_{j,2}, \ldots, P'_{j,a}$. The second output provides the parity bits to be transmitted from the second encoder $|Q'_j|=q'_{j,1}, q'_{j,2}, \ldots, q'_{j,\beta}$.

$$|P'| = \begin{pmatrix} p'_{1,1} & p'_{1,2} & \cdots & p'_{1,\alpha} \\ \vdots & \vdots & \vdots & \vdots \\ p'_{m,1} & p'_{m,2} & \cdots & p'_{m,\alpha} \end{pmatrix}; \quad P'_j = \{p'_{j,1}, p'_{j,2}, \ldots, p'_{j,\alpha}\} \tag{121}$$

$$|Q'| = \begin{pmatrix} q'_{1,1} & q'_{1,2} & \cdots & q'_{1,\beta} \\ \vdots & \vdots & \vdots & \vdots \\ q'_{m,1} & q'_{m,2} & \cdots & q'_{m,\beta} \end{pmatrix}; \quad Q'_j = \{q'_{j,1}, q'_{j,2}, \ldots, q'_{j,\beta}\} \tag{122}$$

[0666] In accordance with an embodiment of the invention, $\alpha+\beta<n$. The total number of parity bits sent in each symbol is $\alpha+\beta$. In traditional puncturing, $\alpha+\beta=n/2$.

[0667] The implementation shown in **FIG. 76** includes additional uncoded information bits, where matrix |B| is the total number of information bits. This is the general form for Multi-level Turbo Codes, where not all the information bits are coded. The number of rows of this matrix is the number of input symbols to be transmitted, "m". The number of rows of this matrix is the number of bits in each symbol, "N". Matrix |B| has a sub-matrix |A|, which represents the information bits to be encoded. The number of rows of this matrix is the number of input symbols to be transmitted, "m". The number of rows of this matrix is the number of bits that are encoded in each symbol, "n".

$$|B| = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,N} \\ \vdots & \vdots & \vdots & \vdots \\ b_{m,1} & b_{m,2} & \cdots & b_{m,N} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} & b_{1,n+1} \ldots b_{1,N} \\ \vdots & \vdots & \vdots & \vdots & \cdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} & b_{m,n+1} \ldots b_{m,N} \end{pmatrix} \tag{123}$$

[0668] 18. Puncturing Rate Adaptation

[0669] Using the Channel State Information (values of gi and bi for each tone for the case of a G.992.1 or G.992.2 modem), the puncturing and mapping can be adapted to alter the transmitted signal bandwidth, which allows communication at a desired BER.

[0670] If there are too many errors (i.e., BER is too high), this can be improved by increasing the constellation size and reducing the puncturing, thereby sending more parity bits. The puncturing of the system is reduced, increasing the values of α and β. If the values of α and β are increased to twice their original values, the redundancy of the communications is also increased to twice its original value.

[0671] Alternatively, if there are too many errors (i.e., BER is too high), this can be improved by reducing the information data rate and reducing the puncturing, thereby sending more parity bits. The puncturing of the system is reduced, increasing the values of α and β. If the values of α and β are increased to twice their original values, the redundancy of the communications is also increased to twice its original value.

[0672] If the channel is working with a very low number of errors, it is possible to send information with fewer parity bits, by increasing the puncturing which reduces the values of α and β.

[0673] In the case that the bandwidth of the system is not a concern (i.e. spread spectrum systems or low data rate situations), reducing the puncturing increases the performance. For this case a and β should use their maximum value, "n".

[0674] The monitoring of the error rate may be performed in the receiver using various techniques. Noise measurements based on the distance between each actual QAM signal point received and its nearest constellation points is a frequently used MSE (Mean-Squared Error)performance metric. Alternatively, the detection of errors in a bit-stream by a forward error-correction protocol, such as Reed Solomon or Turbo Decoder, may be used to determine performance metric. Higher-level protocol detection of errors, such as from an HDLC CRC checker, may also be used to determine a performance metric. Thus the actual error rate need not be measured. Rather, a performance metric may be derived in the receiver using any suitable technique and then characterized to determine when the receiver would benefit from a change in the puncturing pattern. Once the receiver decides to select a different operating puncturing pattern, it communicates this selection to the transmitter using a renegotiation process.

[0675] Taking into account these possibilities, it is clear that the system may adapt to different scenarios modifying

the puncturing pattern using a "puncturing pattern adaptation" based on the condition of the channel, increasing or reducing the values of α and β. The operation of the channel can be controlled using a α-mask and β-mask. The α-mask and β-mask are "n" bits long; each position can take a "0" value or a "1" value.

[0676] The "0" value in the position "i" in the α-mask means that the puncturing pattern will not include the parity bit "$p_{j,i}$"from the first encoder. The "1" value in the position "i" in the α-mask means that the puncturing pattern will include the parity bit "p_j"from the first encoder. The "0" value in the position "i" in the β-mask means that the puncturing pattern will not include the parity bit "$q_{j,i}$"from the second encoder. The "1" value in the position "i" in the β-mask means that the puncturing pattern will include the parity bit "$q_{j,i}$"from the second encoder. Equations (124) and (125) shows the values of these masks.

$$\alpha_j\text{mask}=\{\alpha_{j,1}, \alpha_{j,2}, \ldots, \alpha_{j,n}\}\alpha_{j,i}=0,1 \quad i=1 \ldots n \tag{124}$$

$$\beta_j\text{mask}=\{\beta_{j,1}, \beta_{j,2}, \ldots, \beta_{j,n}\}\beta_{j,i}=0,1 \quad i=1 \ldots n \tag{124}$$

[0677] As seen in the puncturing patterns disclosed in Tables 7-22, the α-mask and β-mask of a puncturing pattern typically utilize the same repetitive bit selection pattern, with the pattern in the β-mask being shifted by one or more bits with respect to the pattern in the α-mask. The patterns of the α-mask and β-mask are typically non-coincidental, in that the two patterns do not select a parity bit at the same time.

[0678] 19. Summarv of operation of transmitters and receivers in preferred embodiments of the invention

[0679] **FIG. 77** illustrates a basic process in a transmitter in accordance with an embodiment of the invention. The process produces a modulated signal with forward error correction from an information bit stream in a QAM transmitter. The transmitter produces 100 parity bit streams that correspond to an inputted information bit stream using first and second concatenated coders interconnected by an interleaver. The transmitter selects 102 subsets of the first and second parity bit streams in accordance with a puncturing pattern. The transmitter combines 104 the selected subsets of said first and second parity bit streams with said information bit stream. A QAM symbol stream is produced 106 by mapping a first subset of the combined bit streams to an I dimension and mapping a second subset of the combined bit streams to a Q dimension. The QAM symbol stream is modulated 108 to produce a modulated signal, and the modulated signal is transmitted 110 over a communication link.

[0680] As described above, any puncturing pattern may be implemented in accordance with the above process. However, it is preferred to implement one of the novel puncturing patterns disclosed herein. As described above, the novel puncturing patterns disclosed herein are characterized in that, for an c-bit constellation, wherein each symbol represents n information bits and c-n parity bits (also $\alpha+\beta$ parity bits using the terminology of Section **17** above), the number of parity bits c-n selected by the puncturing pattern is less than n. In other words, the number of parity bits represented in each symbol is less than the number of information bits represented in that symbol. Examples of such puncturing patterns are provided in Tables 7 and 9-22, and those of ordinary skill will recognize a variety of other patterns that may be implemented in accordance with the parameters described herein.

[0681] **FIG. 78** illustrates a basic process in a receiver that is complementary to a transmitter as described above. The process recovers an information bit stream from a noisy modulated signal with forward error correction in a QAM receiver. The receiver receives 120 a modulated signal from a communications link. The signal includes errors. The received signal is demodulated 122, and a decoded bit stream is produced 124 by iteratively decoding the demodulated signal using first and second concatenated coders connected by an interleaver. The demodulated signal is decoded independently in the I dimension and the Q dimension using a puncturing pattern. The puncturing pattern corresponds to a puncturing pattern implemented in the transmitter that produces the modulated signal. The receiver regenerates 126 the information bit stream from the decoded bit stream.

[0682] As noted above, embodiments in accordance with the invention may adapt the puncturing pattern used in the transmitter in accordance with conditions in the communication system. **FIG. 79** illustrates a basic process in a receiver for adapting to conditions in the communication system. The receiver receives 130 a QAM symbol stream from a transmitter of the communication system. The receiver determines 132 a performance metric of the communication system based on the received QAM symbol stream. The performance metric may be determined based on detection of errors in the QAM symbol stream in accordance with the error detection techniques discussed above or any other conventional error detection technique. If the performance metric is unsatisfactory, the receiver selects 134 a new puncturing pattern for use in the transmitter to improve the performance of the communication system. The puncturing pattern is selected based on the performance metric. The puncturing pattern is then communicated 136 to the transmitter. This may be accomplished by communicating the actual pattern to the transmitter, or by transmitting an identifier of a predefined puncturing pattern.

[0683] While the foregoing disclosure addressed a variety of preferred embodiments of the invention, those having ordinary skill in the art will recognize a variety of further embodiments that may be implemented in accordance with the novel principles disclosed herein.

Votc.h

```
/*
 ***************************************************************************
 * file:        tc_v2.h
 * date:        April 12, 2000
 * function:    tests turbo codes
 *              Modulation: QAM
 *              Decoder:    MAP algorithm
 ***************************************************************************
 */

#include <math.h>
#include <stdio.h>
#include <malloc.h>
#include <dos.h>


/* Definition of the first recursive systematic code (RSC1):                */
#define RSC1_ENC_MEM  4              /* encoder memory order                */
#define RSC1_STATES   (1 << RSC1_ENC_MEM)
#define RSC1_FP       035            /* forward polynomial in octal         */
#define RSC1_BP       023            /* backward polynomial in octal        */

/* Definition of the second recursive systematic code (RSC2):               */
#define RSC2_ENC_MEM  4              /* encoder memory order                */
#define RSC2_STATES   (1 << RSC2_ENC_MEM)
#define RSC2_FP       035            /* forward polynomial in octal         */
#define RSC2_BP       023            /* backward polynomial in octal        */

#define NR_ITER       8              /* nr. of iterative decoding stages    */
#define EBNO          6.0            /* Eb/No in dB                         */
#define MAX_ERRORS    1000           /* stop when this nr. is reached       */
#define INT_SIZE      6144           /* nr. of info bits to be ileaved      */

#define MAX           (exp(31.0))    /* limit soft outputs                  */
#define E_STEPS       1000           /* number of values for E_val          */
#define PRINT_BLOCKS  100            /* how often to print results          */
#define SEED1         13733          /* seeds for random nr. gen.           */
#define SEED2         1935791

#define SIGMA_12_4AM         sqrt(2.50 * pow(10.0,(-EBNO / 10.0))) /* A = 1.0 */
#define SIGMA_12_16QAM       sqrt(2.50 * pow(10.0,(-EBNO / 10.0)))
#define SIGMA_34_16QAM       sqrt((10.0/6.0) * pow(10.0,(-EBNO / 10.0)))

/* For 8AM, 64QAM, 256QAM, A = 0.5 => A*A = 0.25. Thus, Eav = 5.25*A*A = Eav/4 */
#define SIGMA_56_64QAM       sqrt(4.2/4 * pow(10.0,(-EBNO / 10.0)))
#define SIGMA_46_64QAM       sqrt(5.25/4 * pow(10.0,(-EBNO / 10.0)))
#define SIGMA_23_8AM         sqrt(5.25/4 * pow(10.0,(-EBNO / 10.0)))
#define SIGMA_12_8AM         sqrt(7.0/4 * pow(10.0,(-EBNO / 10.0)))
#define SIGMA_58_256QAM      sqrt(17.0/4 * pow(10.0,(-EBNO / 10.0)))
#define SIGMA_68_256QAM      sqrt((170.0/12)/4 * pow(10.0,(-EBNO / 10.0)))

/* For 4QAM (A = 0.5):*/
#define SIGMA_24_4QAM   sqrt(2.0/2.0/4 * pow(10.0,(-EBNO / 10.0))) /* 1 info */
#define SIGMA_26_4QAM   sqrt(2.0/(4.0/3)/4 * pow(10.0,(-EBNO / 10.0))) /* 2/3 info */
/* For 8QAM (A = 0.5):*/
#define SIGMA_4AM_of_46_8QAM   sqrt(6.0/4.0/4 * pow(10.0,(-EBNO / 10.0))) /* 2 info*/
#define SIGMA_2AM_of_46_8QAM   sqrt(6.0/4.0/4 * pow(10.0,(-EBNO / 10.0))) /* 2 info*/
#define SIGMA_4AM_of_26_8QAM   sqrt(6.0/2.0/4 * pow(10.0,(-EBNO / 10.0))) /* 1 info*/
#define SIGMA_2AM_of_26_8QAM   sqrt(6.0/2.0/4 * pow(10.0,(-EBNO / 10.0))) /* 1 info*/
#define SIGMA_4AM_of_13_8QAM   sqrt(6.0/2.0/4 * pow(10.0,(-EBNO / 10.0))) /* 1 info*/
#define SIGMA_2AM_of_13_8QAM   sqrt(6.0/2.0/4 * pow(10.0,(-EBNO / 10.0))) /* 1 info*/
#define SIGMA_36_64QAM   sqrt(42.0/6.0/4 * pow(10.0,(-EBNO / 10.0))) /* 3 info */

/* For 16QAM, 64QAM, 256QAM, 1024QAM (A = 0.5):*/
#define SIGMA_412_16QAM   sqrt(10.0/(8.0/3)/1 * pow(10.0,(-EBNO / 10.0))) /* 4/3 A=1 */
#define SIGMA_26_64QAM   sqrt(42.0/4.0/4 * pow(10.0,(-EBNO / 10.0))) /* 2 info */
#define SIGMA_824_256QAM   sqrt(170.0/(10.0/3)/4 * pow(10.0,(-EBNO / 10.0))) /* 10/3*/
#define SIGMA_1030_1024QAM   sqrt(341.0/(10.0/3)/4 * pow(10.0,(-EBNO / 10.0))) /*10/3 */
```

```
/* For 32QAM, 128QAM, 512QAM,(A = 0.5):*/
#define SIGMA_8AM_of_115_32QAM   sqrt(21.0/2/4 * pow(10.0,(-EBNO / 10.0))) /* 1 info */
#define SIGMA_4AM_of_115_32QAM   sqrt(5.00/(4/3)/4 * pow(10.0,(-EBNO / 10.0))) /*2/3*/
#define SIGMA_16AM_of_721_128QAM   sqrt(85.0/(8/3)/4 * pow(10.0,(-EBNO / 10.0))) /*4/3*/
#define SIGMA_8AM_of_721_128QAM    sqrt(21.0/2/4 * pow(10.0,(-EBNO / 10.0))) /* 1 info */
#define SIGMA_32AM_of_39_512QAM   sqrt(341.0/4/4*pow(10.0,(-EBNO / 10.0))) /* 2 info */
#define SIGMA_16AM_of_39_512QAM   sqrt(85.00/2/4*pow(10.0,(-EBNO / 10.0))) /* 1 info */

/* For 32QAM (A = 0.5):*/
#define SIGMA_8AM_of_32QAM   sqrt(26.0/6/4 * pow(10.0,(-EBNO / 10.0))) /* 3 info */
#define SIGMA_4AM_of_32QAM   sqrt(26.0/6/4 * pow(10.0,(-EBNO / 10.0))) /* 3 info */

/* For R57_128QAM (A = 0.5):*/
#define SIGMA_16AM_of_128QAM   sqrt(106.0/10/4 * pow(10.0,(-EBNO / 10.0))) /* 3 info */
#define SIGMA_8AM_of_128QAM    sqrt(106.0/10/4 * pow(10.0,(-EBNO / 10.0))) /* 2 info */

/* For R69_512QAM (A = 0.5):*/
#define SIGMA_32AM_of_512QAM   sqrt(426.0/12/4*pow(10.0,(-EBNO / 10.0))) /* 4 info */
#define SIGMA_16AM_of_512QAM   sqrt(426.0/12/4*pow(10.0,(-EBNO / 10.0))) /* 2 info */

/* For R710_1024QAM (A = 0.5):*/
#define SIGMA_710_1024QAM   sqrt((341.0/7)/4 * pow(10.0,(-EBNO / 10.0))) /* 3.5 info */

/* Define the particular coding and modulation case for simulation   */
#define R36_64QAM

#define BIT_HIST
#define THRESHOLD_ITER       10       /* record bit histogram for higher iterations */
#define MAX_BIT_HIST_ARRAY   (2 * INT_SIZE)
#define ERROR_FILE_NAME      "../results/R36_64QAM_6144_test_30.err"
#define FRAME_HIST_FILE_NAME "../results/test.fhist"
#define BIT_HIST_FILE_NAME   "../results/map.hist"
#define INTERLEAVER_FILE     "../results/6144/s6144"


 /*
 * Note1:
Make sure that for each simulation, the INT_SIZE represents the size of the interleaver
defined in INTERLEAVER_FILE
 */

/*
 * Note2:
 *
In rate 4/6 64QAM_TTCM only two bits out of four are coded rate half. Therefore,
the first half of the interleaver table used has a INT_SIZE/2 interleaver,
the rest is mapping the bits in the same position.
 */
```

Votc.c

```
/*
 ****************************************************************************
 * file:        tc_v2.c
 * date:        March 20, 2000
 * function:    tests turbo codes
 *              Modulation: QAM
 *              Decoder:    MAP algorithm
 ****************************************************************************
 */

#include "tc_v2.h"

typedef struct {
  int         enc_state;              /* encoder state                        */
  int         nr_states;              /* number of encoder states             */
  int         enc_mem;                /* encoder memory                       */
  int         bp;                     /* backward polynomial                  */
  int         fp;                     /* forward polynomial                   */
  int         *P0state;               /* previous state for i=0 branch        */
  int         *P1state;               /* previous state for i=1 branch        */
  int         *N0state;               /* next state for i=0 branch            */
  int         *N1state;               /* next state for i=1 branch            */
  int         *Coded0;                /* coded bit for i=0 branch             */
  int         *Coded1;                /* coded bit for i=1 branch             */
} jat_code;

void          jat_map1(jat_code *, double *, double *, double *, double *);
void          jat_map2(jat_code *, double *, double *, double *, double *);
void          jat_trellis_bp_fp(jat_code *);
int           jat_ps(jat_code *, int);
int           jat_enc_bp_fp(jat_code *, int);
void          r_ileav(double *, int *);
void          r_ileava(int *, int *);
void          r_deileav(double *, int *);
void          r_deileava(int *, int *);
double        nrgen();
int           nrgenbin();
double        gasdev();
int           errors(int *, double *, int, int);
int           print_err(int *, double *, int, int, int *);
double        find_tx_I(int);
double        find_tx_Q(int);


int           *frame_hist;            /* how many frames with how many errors*/
int           **bit_hist_array;       /* pointer to NR_ITER pointers
                                       to blocks of data organised as:
                                            block nr.,bit pos. in error,
                                            block nr.,bit pos. in error   */
int           *bit_hist_block;        /* current number of blocks in error
                                       for each iteration                 */
int           frame_err;              /* frame/block error rate             */
int           total_err;              /* total nr. of err. after NR_ITER    */
long          s1, s2;                 /* seed generators                    */


/****************************************************************************/
main()
{
jat_code      *jat_code1;
jat_code      *jat_code2;
int           u1, u2, u3, u4, u5, u6;    /* bits of a 64QAM symbol in TTCM   */
double        tx_I, tx_Q, rx_I, rx_Q;
double        v00_I, v00_Q, v01_I, v01_Q, v10_I, v10_Q, v11_I, v11_Q;
int           i, j, k, block, iteration;
int           *rule    ;          /* interleaver                              */
int           *data;              /* the information block of data            */
```

COMPUTER PROGRAM LISTING APPENDIX

```
int          *data_i;          /* the interleaved information block of data*/
int          *data_d;          /* the deinterleaved inf. block of data     */
int          *Enc1;            /* Encoder1 output                          */
int          *Enc2;            /* Encoder2 output                          */
int          *no_err;          /* stores nr. err. for each iteration       */
double       *D1_data;         /* Decoder1 input data                      */
double       *D1_parity;       /* Decoder1 input parity                    */
double       *D1_app;          /* Decoder1 input a priori information      */
double       *D1_exi;          /* Decoder1 output extrinsic information    */
double       *D2_data;         /* Decoder2 input data                      */
double       *D2_parity;       /* Decoder2 input parity                    */
double       *D2_app;          /* Decoder2 input a priori information      */
double       *D2_exi;          /* Decoder2 output extrinsic information    */
double       *Dec_data;        /* Decoded data                             */
double       *Zero_data;       /* zero data                                */
double       d0, d1, d2, d3, d4, L_d0, L_d1, L_d2, L_d3, tx, rx,K,noise1,n;
double       L_d4, L_d5;
double       L_u1, L_u2, L_u3, L_u4, L_u5, L_u6;
double       noise_I, noise_Q;
FILE         *out_file = NULL;

s1        = SEED1;  /* initialize the seeds for the noise generator          */
s2        = SEED2;
frame_err = 0;
total_err = 0;

/*
 * initialize the code structures:
 */
jat_code1            = (jat_code *)malloc(sizeof(jat_code));
jat_code1->enc_mem   = RSC1_ENC_MEM;
jat_code1->bp        = RSC1_BP;
jat_code1->fp        = RSC1_FP;
jat_code1->enc_state = 0;
jat_code1->nr_states = (1 << RSC1_ENC_MEM);
jat_code1->P0state   = (int *)malloc(sizeof(int)*jat_code1->nr_states);
jat_code1->P1state   = (int *)malloc(sizeof(int)*jat_code1->nr_states);
jat_code1->N0state   = (int *)malloc(sizeof(int)*jat_code1->nr_states);
jat_code1->N1state   = (int *)malloc(sizeof(int)*jat_code1->nr_states);
jat_code1->Coded0    = (int *)malloc(sizeof(int)*jat_code1->nr_states);
jat_code1->Coded1    = (int *)malloc(sizeof(int)*jat_code1->nr_states);
jat_trellis_bp_fp(jat_code1);
jat_code2            = (jat_code *)malloc(sizeof(jat_code));
jat_code2->enc_mem   = RSC2_ENC_MEM;
jat_code2->bp        = RSC2_BP;
jat_code2->fp        = RSC2_FP;
jat_code2->enc_state = 0;
jat_code2->nr_states = (1 << RSC2_ENC_MEM);
jat_code2->P0state   = (int *)malloc(sizeof(int)*jat_code2->nr_states);
jat_code2->P1state   = (int *)malloc(sizeof(int)*jat_code2->nr_states);
jat_code2->N0state   = (int *)malloc(sizeof(int)*jat_code2->nr_states);
jat_code2->N1state   = (int *)malloc(sizeof(int)*jat_code2->nr_states);
jat_code2->Coded0    = (int *)malloc(sizeof(int)*jat_code2->nr_states);
jat_code2->Coded1    = (int *)malloc(sizeof(int)*jat_code2->nr_states);
jat_trellis_bp_fp(jat_code2);


data = (int *)malloc(sizeof(int) * INT_SIZE);
if(data == 0)
  {
    printf("Couldn't allocate data memory!\n");
    exit(1);
  }

data_i = (int *)malloc(sizeof(int) * INT_SIZE);
if(data_i == 0)
  {
    printf("Couldn't allocate data_i memory!\n");
    exit(1);
  }
```

COMPUTER PROGRAM LISTING APPENDIX

```
data_d = (int *)malloc(sizeof(int) * INT_SIZE);
if(data_d == 0)
  {
    printf("Couldn't allocate data_d memory!\n");
    exit(1);
  }

no_err = (int *)malloc(sizeof(int) * NR_ITER);
if(no_err == 0)
  {
    printf("Couldn't allocate no_err memory!\n");
    exit(1);
  }
else
  for(i = 0; i < NR_ITER; i++)
    no_err[i] = 0;

Enc1 = (int *)malloc(sizeof(int) * INT_SIZE);
if(Enc1 == 0)
  {
    printf("Couldn't allocate Enc1 memory!\n");
    exit(1);
  }

Enc2 = (int *)malloc(sizeof(int) * INT_SIZE);
if(Enc2 == 0)
  {
    printf("Couldn't allocate Enc2 memory!\n");
    exit(1);
  }

D1_data = (double *)malloc(sizeof(double) * INT_SIZE);
if(D1_data == 0)
  {
    printf("Couldn't allocate D1_data memory!\n");
    exit(1);
  }

D1_parity = (double *)malloc(sizeof(double) * INT_SIZE);
if(D1_parity == 0)
  {
    printf("Couldn't allocate D1_parity memory!\n");
    exit(1);
  }

D1_app = (double *)malloc(sizeof(double) * INT_SIZE);
if(D1_app == 0)
  {
    printf("Couldn't allocate D1_app memory!\n");
    exit(1);
  }

D1_exi = (double *)malloc(sizeof(double) * INT_SIZE);
if(D1_exi == 0)
  {
    printf("Couldn't allocate D1_exi memory!\n");
    exit(1);
  }

D2_data = (double *)malloc(sizeof(double) * INT_SIZE);
if(D2_data == 0)
  {
    printf("Couldn't allocate D2_data memory!\n");
    exit(1);
  }

D2_parity = (double *)malloc(sizeof(double) * INT_SIZE);
if(D2_parity == 0)
  {
    printf("Couldn't allocate D2_parity memory!\n");
    exit(1);
```

COMPUTER PROGRAM LISTING APPENDIX

```
  }

D2_app = (double *)malloc(sizeof(double) * INT_SIZE);
if(D2_app == 0)
  {
    printf("Couldn't allocate D2_app memory!\n");
    exit(1);
  }

D2_exi = (double *)malloc(sizeof(double) * INT_SIZE);
if(D2_exi == 0)
  {
    printf("Couldn't allocate D2_exi memory!\n");
    exit(1);
  }

Dec_data = (double *)malloc(sizeof(double) * INT_SIZE);
if(Dec_data == 0)
  {
    printf("Couldn't allocate Dec_data memory!\n");
    exit(1);
  }

Zero_data = (double *)malloc(sizeof(double) * INT_SIZE);
if(Zero_data == 0)
  {
    printf("Couldn't allocate Zero_data memory!\n");
    exit(1);
  }
for(i = 0; i < INT_SIZE; i++)
  Zero_data[i] = 0.0;

frame_hist = (int *)malloc(sizeof(int) * (INT_SIZE+1) * NR_ITER);
if(frame_hist == 0)
  {
    printf("Couldn't allocate frame_hist memory!\n");
    exit(1);
  }
else
  {
    for(i = 0; i < (INT_SIZE+1)*NR_ITER; i++)
      frame_hist[i] = 0;
  }

bit_hist_array = (int **)malloc(sizeof(int *) * 2 * NR_ITER);
if(bit_hist_array == 0)
  {
    printf("Couldn't allocate bit_hist_array memory!\n");
    exit(1);
  }
else
  {
    for(i = THRESHOLD_ITER; i <= NR_ITER; i++)
      {
        bit_hist_array[i] = (int *)malloc(sizeof(int) * MAX_BIT_HIST_ARRAY);
        bit_hist_array[i+NR_ITER] = bit_hist_array[i]; /* store the original pointer */
        if(bit_hist_array[i] == 0)
          {
            printf("Couldn't allocate bit_hist_array[i] memory!\n");
            exit(1);
          }
      }
  }

bit_hist_block = (int *)malloc(sizeof(int) * (NR_ITER+1));
if(bit_hist_block == 0)
  {
    printf("Couldn't allocate bit_hist_block memory!\n");
    exit(1);
  }
else
```

COMPUTER PROGRAM LISTING APPENDIX

```
  {
    for(i = 0; i <= NR_ITER; i++)
      bit_hist_block[i] = 1;
  }

rule = (int *)malloc(sizeof(int) * INT_SIZE * 2);
if(rule == 0)
  {
    printf("Couldn't allocate rule memory!\n");
    exit(1);
  }

for(i = 0; i < INT_SIZE; i++)
  rule[2*i] = 0;

/*
 * read the interleaver file
 */

out_file = fopen(INTERLEAVER_FILE, "r");

if(!out_file)
  {
    printf("Error1: the output file could not be opened!\n");
    exit (1);
  }
for(i = 0; i < INT_SIZE; i++)
  fscanf(out_file, "%d%d", &i, &rule[2*i+1]);
fclose(out_file);


/*
 * initialize the noise generator seeds in order to have the same data and
 * noise for different random interleavers
 */
s1 = SEED1;
s2 = SEED2;

/*
 * start the big loop:
 */
for(block = 1; total_err < MAX_ERRORS; block++)
  {
    jat_code1->enc_state = 0;                    /* reset encoder1's state    */
    jat_code2->enc_state = 0;                    /* reset encoder2's state    */
    for(i = 0; i < INT_SIZE; i++)                /* no app for first decoder  */
      D1_app[i] = 1.0;

    /*
     * generate random data:
     */
    for(i = 0; i < INT_SIZE; i++)
      data[i] = nrgenbin();

    /*
     * encoder1:
     */
    for(i = 0; i < INT_SIZE; i++)
      Enc1[i] = jat_enc_bp_fp(jat_code1, data[i]);

    /*
     * interleave data:
     */
    for(i = 0; i < INT_SIZE; i++)
      data_i[i] = data[i];
    r_ileava(data_i, rule);

    /*
     * encoder2:
     */
    for(i = 0; i < INT_SIZE; i++)
```

COMPUTER PROGRAM LISTING APPENDIX

```
      Enc2[i] = jat_enc_bp_fp(jat_code2, data_i[i]);

    /*
     * deinterleave data:
     */
    for(i = 0; i < INT_SIZE; i++)
      data_d[i] = data[i];
    r_deileava(data_d, rule);

    /*
     * modulate and add AWGN noise:
     */


#ifdef R12_4AM
        /*
         * Channel:
         * d0 is MSB and d1 is LSB in a 4-AM:(d0,d1) = 01--00-|-10--11
         *                                              -3  -1    1   3
         */

    n   = (-1.0) / (2 * SIGMA_12_4AM * SIGMA_12_4AM);
    for(i = 0; i < INT_SIZE; i++)
      {
        d0 = data[i];
        if(i & 0x1)
          d1 = Enc1[i];
        else
          d1 = Enc2[i];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
        rx   = tx + SIGMA_12_4AM * gasdev();
        L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
                   (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));
        L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
                   (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));
        D1_data[i] = L_d0;
        if(i & 0x1)
          {
            D1_parity[i] = L_d1;
            D2_parity[i] = 0.0;
          }
        else
          {
            D1_parity[i] = 0.0;
            D2_parity[i] = L_d1;
          }
      }
#endif


#ifdef R13_8AM
    /*
     * Channel:
     * d0 is MSB and d2 is LSB in 8-AM:(d0,d1,d2):
     *       010---011---001---000---100---101---111---110
     *      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
     */

    n   = (-1.0) / (2 * SIGMA_13_8AM * SIGMA_13_8AM);
    for(i = 0; i < INT_SIZE; i++)
      {
        d0 = data[i];
        if(i & 0x1)
          {
            d1 = Enc1[i];
            d2 = Enc2[i];
          }
        else
          {
            d1 = Enc2[i];
            d2 = Enc1[i];
```

COMPUTER PROGRAM LISTING APPENDIX

```
          }

        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
        rx   = tx + SIGMA_13_8AM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                    exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                   (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                    exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

        L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                    exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                   (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                    exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

        L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                    exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                   (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                    exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

        D1_data[i] = L_d0;
        if(i & 0x1)
          {
            D1_parity[i] = L_d1;
            D2_parity[i] = L_d2;
          }
        else
          {
            D1_parity[i] = L_d2;
            D2_parity[i] = L_d1;
          }
      }
#endif

#ifdef R12_8AM
    /*
     * Channel:
     * d0 is MSB and d2 is LSB in 8-AM:(d0,d1,d2):
     *       010---011---001---000---100---101---111---110
     *      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
     */
    /*
     * Channel:  we transmit two 8AM symbols to emulate a 64QAM symbol.
     * 6 info bits and 6 parity bits are mapped to 2 64QAM symbols which in
     * turn are simulated as 4 8AM symbols to achieve 3bit/s/Hz
     *
     * INT_SIZE to be a multiple of 6
     */

    n   = (-1.0) / (2 * SIGMA_12_8AM * SIGMA_12_8AM);
    for(i = 0; i < INT_SIZE; i++)
      {
        /* symbol 1 */
        d0  = data[i];
        d1  = data[i+1];
        d2  = Enc1[i];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
        rx   = tx + SIGMA_12_8AM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                    exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                   (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                    exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

        L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                    exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                   (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                    exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

        L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                    exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                   (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                    exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
```

COMPUTER PROGRAM LISTING APPENDIX

```
D1_data[i]      = L_d0;
D1_data[i+1]    = L_d1;
D1_parity[i]    = L_d2;
D1_parity[i+1]  = 0;
D2_parity[i]    = 0;

/* symbol 2 */
d0   = data[i+2];
d1   = Enc1[i+2];
d2   = Enc2[i+1];
tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx   = tx + SIGMA_12_8AM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
          (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
          (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
          (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
D1_data[i+2]    = L_d0;
D1_parity[i+2]  = L_d1;
D2_parity[i+1]  = L_d2;
D2_parity[i+2]  = 0;

/* symbol 3 */
d0   = data[i+3];
d1   = data[i+4];
d2   = Enc2[i+3];
tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx   = tx + SIGMA_12_8AM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
          (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
          (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
          (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
D1_data[i+3]    = L_d0;
D1_data[i+4]    = L_d1;
D2_parity[i+3]  = L_d2;
D1_parity[i+3]  = 0;
D2_parity[i+4]  = 0;

/* symbol 4 */
d0   = data[i+5];
d1   = Enc1[i+4];
d2   = Enc2[i+5];
tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx   = tx + SIGMA_12_8AM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
          (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
          (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

        L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
               (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
        D1_data[i+5]   = L_d0;
        D1_parity[i+4] = L_d1;
        D2_parity[i+5] = L_d2;
        D1_parity[i+5] = 0;

        i = i+5;
      }
#endif

#ifdef R23_8AM
    /*
     * Channel:
     * d0 is MSB and d2 is LSB in 8-AM:(d0,d1,d2):
     *       010---011---001---000---100---101---111---110
     *      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
     */

    n   = (-1.0) / (2 * SIGMA_23_8AM * SIGMA_23_8AM);
    for(i = 0; i < INT_SIZE; i++)
      {
        d0 = data[i];
        d1 = data[i+1];
        if(i & 0x4)
          d2 = Enc1[i];
        else
          d2 = Enc2[i];


        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
        rx   = tx + SIGMA_23_8AM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

        L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

        L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
               (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

        D1_data[i]   = L_d0;
        D1_data[i+1] = L_d1;
        if(i & 0x4)
          {
            D1_parity[i]   = L_d2;
            D1_parity[i+1] = 0;
            D2_parity[i]   = 0;
            D2_parity[i+1] = 0;
          }
        else
          {
            D1_parity[i]   = 0;
            D1_parity[i+1] = 0;
            D2_parity[i]   = L_d2;
            D2_parity[i+1] = 0;
          }

        i++;
      }
#endif
```

COMPUTER PROGRAM LISTING APPENDIX

```
#ifdef R35_32QAM
    /*
     * I dimension:
     * d0 is MSB and d2 is LSB in 8-AM:(d0,d1,d2):
     *       010---011---001---000---100---101---111---110
     *      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
     * Q dimension:
     * d0 is MSB and d1 is LSB in a 4-AM:(d0,d1):
     *       01----00----10----11
     *     -1.5  -0.5   0.5   1.5
     *
     *
     * We transmit one 8AM symbol and one 4AM symbol to emulate a 32QAM symbol.
     * 6 info bits and 4 parity bits are mapped to 2 32QAM symbols.
     *
     * INT_SIZE to be a multiple of 6
     */

    for(i = 0; i < INT_SIZE; i++)
      {
        /* symbol 1: 8AM */
    d0  = data[i];
    d1  = data[i+2];
        d2  = Enc1[i];
        tx  = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx  = tx + SIGMA_8AM_of_32QAM * gasdev();
        n   = (-1.0) / (2 * SIGMA_8AM_of_32QAM * SIGMA_8AM_of_32QAM);
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

        L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

        L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                  (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
         D1_data[i]      = L_d0;
         D1_data[i+2]    = L_d1;
        D1_parity[i]    = L_d2;
        D1_parity[i+1] = 0.0;
        D1_parity[i+2] = 0.0;
        D2_parity[i]    = 0.0;
        D2_parity[i+2] = 0.0;

         /* symbol 2: 4AM */
        d0 = data[i+1];
        d1 = Enc2[i+1];
        tx   = d0 - d1 + 2*d0*d1 - 0.5;
        rx   = tx + SIGMA_4AM_of_32QAM * gasdev();
        n   = (-1.0) / (2 * SIGMA_4AM_of_32QAM * SIGMA_4AM_of_32QAM);
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                  (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
        L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                  (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
        D1_data[i+1]    = L_d0;
        D2_parity[i+1] = L_d1;

         /* symbol 3: 8AM */
    d0   = data[i+3];
    d1   = data[i+5];
        d2   = Enc2[i+4];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx  = tx + SIGMA_8AM_of_32QAM * gasdev();
        n   = (-1.0) / (2 * SIGMA_8AM_of_32QAM * SIGMA_8AM_of_32QAM);
```

COMPUTER PROGRAM LISTING APPENDIX

```
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

        L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

        L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                  (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
        D1_data[i+3]    = L_d0;
        D1_data[i+5]    = L_d1;
        D1_parity[i+4] = 0.0;
        D1_parity[i+5] = 0.0;
        D2_parity[i+3] = 0.0;
        D2_parity[i+4] = L_d2;
        D2_parity[i+5] = 0.0;

        /* symbol 4: 4AM */
        d0 = data[i+4];
        d1 = Enc1[i+3];
        tx   = d0 - d1 + 2*d0*d1 - 0.5;
        rx   = tx + SIGMA_4AM_of_32QAM * gasdev();
        n   = (-1.0) / (2 * SIGMA_4AM_of_32QAM * SIGMA_4AM_of_32QAM);
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                  (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
        L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                  (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
        D1_data[i+4]    = L_d0;
        D1_parity[i+3] = L_d1;

        i = i+5;
      }
#endif


#ifdef R46_64QAM_TTCM_VoCAL
    /* Option 4
     * Channel: I & Q defined as
     *       -|-----|-----|-----|-----|-----|-----|-----|-
     *      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
     * the 64QAM symbol is defined as: (u1, u2, u3, u4, u5, u6)
     * where u4 = d0
     *       u3 = d1
     *       u2 = p0 parity from ENC_H
     *       u1 = q1 parity from ENC_V
     *       u5 = d uncoded
     *       u6 = d uncoded
     * Use s4096v interleaver - only first 2048 bits are interleaved
     */

    n   = (-1.0) / (2 * SIGMA_46_64QAM * SIGMA_46_64QAM);
    for(i = 0; i < INT_SIZE/2 -1;)
      {
       /* Encode only first half of INT_SIZE
        * d0, d1, d2, d3,...    up to INT_SIZE/2 - 1
        * p0,  0, p2,  0,...
        *  0, q1,  0, q3,...
        */
       u4 = data[i];
       u3 = data[i+1];
       u2 = Enc1[i];
       u1 = Enc2[i+1];
       u5 = data[i+INT_SIZE/2];
       u6 = data[i+INT_SIZE/2+1];

       k = u6+2*u5+4*u4+8*u3+16*u2+32*u1;
```

COMPUTER PROGRAM LISTING APPENDIX

```
tx_I = find_tx_I(k);
tx_Q = find_tx_Q(k);
rx_I = tx_I + SIGMA_46_64QAM * gasdev();
rx_Q = tx_Q + SIGMA_46_64QAM * gasdev();

L_d0 = log((exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+3.5)*(rx_Q+3.5)))) /
          (exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+3.5)*(rx_Q+3.5)))));
```

COMPUTER PROGRAM LISTING APPENDIX

```
L_d1 = log((exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
           exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
           exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+2.5)*(rx_Q+2.5)))) /
          (exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
           exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
           exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
           exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
           exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
           exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+3.5)*(rx_Q+3.5)))));

L_d2 = log((exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
           exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
           exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+3.5)*(rx_Q+3.5)))) /
(exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+3.5)*(rx_Q+3.5)))));

L_d3 = log((exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+3.5)*(rx_Q+3.5)))) /
               (exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+0.5)*(rx_Q+0.5)))));

        D1_data[i]      = L_d3;
        D1_data[i+1]    = L_d2;
        D1_data[i + INT_SIZE/2]      = rx_I;
        D1_data[i + INT_SIZE/2 + 1] = rx_Q;
        D1_parity[i]    = L_d1;
        D1_parity[i+1] = 0;
        D2_parity[i]    = 0;
        D2_parity[i+1] = L_d0;

        i = i + 2;
      }
#endif


#ifdef R46_64QAM_TTCM_Ungerboeck_Map
    /* Option3: conventional set partitioning used in TCM
     * Channel: I & Q defined
     *       -|-----|-----|-----|-----|-----|-----|-----|-
```

COMPUTER PROGRAM LISTING APPENDIX

```
 *      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
 * the 64QAM symbol is defined as: (u1, u2, u3, u4, u5, u6)
 * where
 *       u6 = d0
 *       u5 = d1
 *       u4 = d2
 *       u3 = d3
 *       u2 = p0 parity from ENC_H
 *       u1 = q1 parity from ENC_V
 */
/*
 * deinterleave data:
 */
for(i = 0; i < INT_SIZE; i++)
  data_d[i] = data[i];
r_deileava(data_d, rule);
n   = (-1.0) / (2 * SIGMA_46_64QAM * SIGMA_46_64QAM);
for(i = 0; i < INT_SIZE;)
  {
    /* Puncturing patern is:
     * d0, d1, d2, d3,...
     * p0,  0,  0,  0,...
     *  0,  0, q2,  0,...
     */

    u1 = Enc1[i];
    u2 = Enc2[i];
    u6 = data_d[i+3];
    u5 = data_d[i+2];
    u4 = data_d[i+1];
    u3 = data_d[i];

    k = u6+2*u5+4*u4+8*u3+16*u2+32*u1;

    tx_I = find_tx_I(k);
    tx_Q = find_tx_Q(k);

    rx_I = tx_I + SIGMA_46_64QAM * gasdev();
    rx_Q = tx_Q + SIGMA_46_64QAM * gasdev();

    L_d0 = log((exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+3.5)*(rx_Q+3.5)))) /
           (exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+3.5)*(rx_Q+3.5)))));

L_d1 = log((exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+2.5)*(rx_Q+2.5)))) /
           (exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+3.5)*(rx_Q+3.5)))));

L_d2 = log((exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+3.5)*(rx_Q+3.5)))) /
           (exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+3.5)*(rx_Q+3.5)))));

L_d3 = log((exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+3.5)*(rx_Q+3.5)))) /
           (exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+0.5)*(rx_Q+0.5)))));

L_d4 = log((exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+2.5)*(rx_Q+2.5)))) /
               (exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
                exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
                exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
                exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
                exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
                exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-3.5)*(rx_Q-3.5)))));

L_d5 = log((exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I+3.5)*(rx_I+3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+2.5)*(rx_I+2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+1.5)*(rx_I+1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I+0.5)*(rx_I+0.5) + (rx_Q-3.5)*(rx_Q-3.5)))) /
           (exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-3.5)*(rx_Q-3.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-2.5)*(rx_Q-2.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-1.5)*(rx_Q-1.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q-0.5)*(rx_Q-0.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+0.5)*(rx_Q+0.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+1.5)*(rx_Q+1.5))) +
            exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+2.5)*(rx_Q+2.5))) +
            exp(n*((rx_I-0.5)*(rx_I-0.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
                exp(n*((rx_I-1.5)*(rx_I-1.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I-2.5)*(rx_I-2.5) + (rx_Q+3.5)*(rx_Q+3.5))) +
                exp(n*((rx_I-3.5)*(rx_I-3.5) + (rx_Q+3.5)*(rx_Q+3.5)))));
        D1_data[i]     = L_d2;
        D1_data[i+1]   = L_d3;
        D1_data[i+2]   = L_d4;
        D1_data[i+3]   = L_d5;
        D1_parity[i]   = L_d0;
        D1_parity[i+1] = 0.0;
        D1_parity[i+2] = 0.0;
        D1_parity[i+3] = 0.0;
        D2_parity[i]   = L_d1;
        D2_parity[i+1] = 0.0;
        D2_parity[i+2] = 0.0;
        D2_parity[i+3] = 0.0;

        i = i + 4;
      }
    /*
     * interleave data:
     */
    r_ileav(D1_data, rule);

#endif


#ifdef R46_64QAM_IQ_Natural_Map
    /* Option2
     * Channel: I = (u1, u2, u3), Q = (u4, u5, u6) defined using natural mapping:
     *       000   001   010   011   100   101   110   111
     *       -|-----|-----|-----|-----|-----|-----|-----|-
     *      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
     * the 64QAM symbol is defined as: (u1, u2, u3, u4, u5, u6)
     * where:
     *       u1 = d0
     *       u2 = d1
     *       u3 = p0 parity from ENC_V
     *       u4 = d2
     *       u5 = d3
     *       u6 = q0 parity from ENC_H
     *
     */

    /*
     * deinterleave data:
     */
    for(i = 0; i < INT_SIZE; i++)
      data_d[i] = data[i];
    r_deileava(data_d, rule);

    n   = (-1.0) / (2 * SIGMA_46_64QAM * SIGMA_46_64QAM);

    for(i = 0; i < INT_SIZE;)
      {
        /* Puncturing patern is:
         * d0, d1, d2, d3,...
         * p0,  0,  0,  0,...
         * q0,  0,  0,  0,...
         */

        u1 = data_d[i];
        u2 = data_d[i+1];
        u3 = Enc1[i];
        u4 = data_d[i+2];
        u5 = data_d[i+3];
        u6 = Enc2[i];

        tx_I = -3.5 + u3 + 2*u2 +4*u1;
        tx_Q = -3.5 + u6 + 2*u5 +4*u4;

        rx_I = tx_I + SIGMA_46_64QAM * gasdev();
```

COMPUTER PROGRAM LISTING APPENDIX

```
    rx_Q = tx_Q + SIGMA_46_64QAM * gasdev();

    L_u1 = log((exp(n*((rx_I-3.5)*(rx_I-3.5)))+
               exp(n*((rx_I-2.5)*(rx_I-2.5)))+
               exp(n*((rx_I-1.5)*(rx_I-1.5)))+
               exp(n*((rx_I-0.5)*(rx_I-0.5))))/
              (exp(n*((rx_I+3.5)*(rx_I+3.5)))+
               exp(n*((rx_I+2.5)*(rx_I+2.5)))+
               exp(n*((rx_I+1.5)*(rx_I+1.5)))+
               exp(n*((rx_I+0.5)*(rx_I+0.5)))));
    L_u2 = log((exp(n*((rx_I-3.5)*(rx_I-3.5)))+
               exp(n*((rx_I-2.5)*(rx_I-2.5)))+
               exp(n*((rx_I+1.5)*(rx_I+1.5)))+
               exp(n*((rx_I+0.5)*(rx_I+0.5))))/
              (exp(n*((rx_I+3.5)*(rx_I+3.5)))+
               exp(n*((rx_I+2.5)*(rx_I+2.5)))+
               exp(n*((rx_I-1.5)*(rx_I-1.5)))+
               exp(n*((rx_I-0.5)*(rx_I-0.5)))));
    L_u3 = log((exp(n*((rx_I-3.5)*(rx_I-3.5)))+
               exp(n*((rx_I+2.5)*(rx_I+2.5)))+
               exp(n*((rx_I-1.5)*(rx_I-1.5)))+
               exp(n*((rx_I+0.5)*(rx_I+0.5))))/
              (exp(n*((rx_I+3.5)*(rx_I+3.5)))+
               exp(n*((rx_I-2.5)*(rx_I-2.5)))+
               exp(n*((rx_I+1.5)*(rx_I+1.5)))+
               exp(n*((rx_I-0.5)*(rx_I-0.5)))));

    rx_I = rx_Q;

    L_u4 = log((exp(n*((rx_I-3.5)*(rx_I-3.5)))+
               exp(n*((rx_I-2.5)*(rx_I-2.5)))+
               exp(n*((rx_I-1.5)*(rx_I-1.5)))+
               exp(n*((rx_I-0.5)*(rx_I-0.5))))/
              (exp(n*((rx_I+3.5)*(rx_I+3.5)))+
               exp(n*((rx_I+2.5)*(rx_I+2.5)))+
               exp(n*((rx_I+1.5)*(rx_I+1.5)))+
               exp(n*((rx_I+0.5)*(rx_I+0.5)))));
    L_u5 = log((exp(n*((rx_I-3.5)*(rx_I-3.5)))+
               exp(n*((rx_I-2.5)*(rx_I-2.5)))+
               exp(n*((rx_I+1.5)*(rx_I+1.5)))+
               exp(n*((rx_I+0.5)*(rx_I+0.5))))/
              (exp(n*((rx_I+3.5)*(rx_I+3.5)))+
               exp(n*((rx_I+2.5)*(rx_I+2.5)))+
               exp(n*((rx_I-1.5)*(rx_I-1.5)))+
               exp(n*((rx_I-0.5)*(rx_I-0.5)))));
    L_u6 = log((exp(n*((rx_I-3.5)*(rx_I-3.5)))+
               exp(n*((rx_I+2.5)*(rx_I+2.5)))+
               exp(n*((rx_I-1.5)*(rx_I-1.5)))+
               exp(n*((rx_I+0.5)*(rx_I+0.5))))/
              (exp(n*((rx_I+3.5)*(rx_I+3.5)))+
               exp(n*((rx_I-2.5)*(rx_I-2.5)))+
               exp(n*((rx_I+1.5)*(rx_I+1.5)))+
               exp(n*((rx_I-0.5)*(rx_I-0.5)))));

    D1_data[i]     = L_u1;
    D1_data[i+1]   = L_u2;
    D1_data[i+2]   = L_u4;
    D1_data[i+3]   = L_u5;
    D1_parity[i]   = L_u3;
    D1_parity[i+1] = 0.0;
    D1_parity[i+2] = 0.0;
    D1_parity[i+3] = 0.0;
    D2_parity[i]   = L_u6;
    D2_parity[i+1] = 0.0;
    D2_parity[i+2] = 0.0;
    D2_parity[i+3] = 0.0;

    i = i + 4;
  }
/*
 * interleave data:
```

COMPUTER PROGRAM LISTING APPENDIX

```
     */
    r_ileav(D1_data, rule);

#endif


#ifdef R46_64QAM_IQ_Gray_Map
    /* Option1: used in "Parallel Concatenated Trellis Coded Modulation" ICC'96
     * Channel: I = (u1, u2, u3), Q = (u4, u5, u6) defined using Gray mapping:
     * u1 & u4 are MSBs and u3 & u6 are LSBs:
     *       010---011---001---000---100---101---111---110
     *      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
     * where:
     *       u1 = d0
     *       u2 = d1
     *       u3 = p0 parity from ENC_V
     *       u4 = d2
     *       u5 = d3
     *       u6 = q0 parity from ENC_H
     *
     * INT_SIZE = multiple of 4
     */

    n   = (-1.0) / (2 * SIGMA_23_8AM * SIGMA_23_8AM);

    /*
     * deinterleave data:
     */
    for(i = 0; i < INT_SIZE; i++)
      data_d[i] = data[i];
    r_deileava(data_d, rule);


    for(i = 0; i < INT_SIZE;)
      {
       /* Puncturing patern is:
        * d0, d1, d2, d3,...
        * p0,  0,  0,  0,...
        * q0,  0,  0,  0,...
        */

       u1   = data_d[i];
       u2   = data_d[i+1];
       u3   = Enc1[i];
       tx   = 2*u1 - 2*u2 + 4*u1*u2 - 1.0 + (((2*u1-1)*(2*u2-1))<0?(u3-0.5):(0.5-u3));
       rx   = tx + SIGMA_23_8AM * gasdev();
       L_u1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

       L_u2 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

       L_u3 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                  (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
       u1   = data_d[i+2];
       u2   = data_d[i+3];
       u3   = Enc2[i];
       tx   = 2*u1 - 2*u2 + 4*u1*u2 - 1.0 + (((2*u1-1)*(2*u2-1))<0?(u3-0.5):(0.5-u3));
       rx   = tx + SIGMA_23_8AM * gasdev();
       L_u4 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

       L_u5 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

        L_u6 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
               (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));

        D1_data[i]     = L_u1;
        D1_data[i+1]   = L_u2;
        D1_data[i+2]   = L_u4;
        D1_data[i+3]   = L_u5;
        D1_parity[i]   = L_u3;
        D1_parity[i+1] = 0.0;
        D1_parity[i+2] = 0.0;
        D1_parity[i+3] = 0.0;
        D2_parity[i]   = L_u6;
        D2_parity[i+1] = 0.0;
        D2_parity[i+2] = 0.0;
        D2_parity[i+3] = 0.0;

        i = i + 4;

      }
    /*
     * interleave data:
     */
    r_ileav(D1_data, rule);

#endif


/*mio*/
#ifdef R24_4QAM
    /*
     * Channel:  we transmit two 2-AM symbols to emulate a 4-QAM symbol.
     * 2 info bits and 2 parity bits are mapped to 2 4-QAM symbols which in
     * turn are simulated as 4 2-AM symbols to achieve 1bit/s/Hz
     *
     * INT_SIZE to be a multiple of 2
     */

    n   = (-1.0) / (2 * SIGMA_24_4QAM * SIGMA_24_4QAM);
    for(i = 0; i < INT_SIZE; i++)
      {
        /* symbol 1 */
        d0  = data[i];
        tx   = d0 - 0.5;
        rx   = tx + SIGMA_24_4QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                   (exp(n*(rx+0.5)*(rx+0.5))));
        D1_data[i]   = L_d0;

        /* symbol 2 */
        d0   = Enc1[i];
        tx   = d0 - 0.5;
        rx   = tx + SIGMA_24_4QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                   (exp(n*(rx+0.5)*(rx+0.5))));
        D1_parity[i] = L_d0;

        /* symbol 3 */
        d0   = data[i+1];
        tx   = d0 - 0.5;
        rx   = tx + SIGMA_24_4QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                   (exp(n*(rx+0.5)*(rx+0.5))));
        D1_data[i+1] = L_d0;

        /* symbol 4 */
```

```
        d0   = Enc2[i+1];
        tx   = d0 - 0.5;
        rx   = tx + SIGMA_24_4QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                  (exp(n*(rx+0.5)*(rx+0.5))));
        D2_parity[i+1]   = L_d0;
        D1_parity[i+1]   = 0.0;
        D2_parity[i]   = 0.0;
        i = i+1;
      }
#endif

/*mio*/
#ifdef R26_4QAM
    /*
     * Channel:  we transmit two 2-AM symbols to emulate a 4-QAM symbol.
     * 2 info bits and 4 parity bits are mapped to 3 4-QAM symbols which in
     * turn are simulated as 6 2-AM symbols to achieve 1bit/s/Hz
     *
     * INT_SIZE to be a multiple of 2
     */

    n   = (-1.0) / (2 * SIGMA_26_4QAM * SIGMA_26_4QAM);
    for(i = 0; i < INT_SIZE; i++)
      {
        /* symbol 1 */
        d0  = data[i];
        tx   = d0 - 0.5;
        rx   = tx + SIGMA_26_4QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                  (exp(n*(rx+0.5)*(rx+0.5))));
        D1_data[i]   = L_d0;

        /* symbol 2 */
        d0   = Enc1[i];
        tx   = d0 - 0.5;
        rx   = tx + SIGMA_26_4QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                  (exp(n*(rx+0.5)*(rx+0.5))));
        D1_parity[i] = L_d0;

        /* symbol 3 */
        d0   = Enc2[i];
        tx   = d0 - 0.5;
        rx   = tx + SIGMA_26_4QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                  (exp(n*(rx+0.5)*(rx+0.5))));
        D2_parity[i] = L_d0;

        /* symbol 4 */
        d0  = data[i+1];
        tx   = d0 - 0.5;
        rx   = tx + SIGMA_26_4QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                  (exp(n*(rx+0.5)*(rx+0.5))));
        D1_data[i+1]   = L_d0;

        /* symbol 5 */
        d0   = Enc1[i+1];
        tx   = d0 - 0.5;
        rx   = tx + SIGMA_26_4QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                  (exp(n*(rx+0.5)*(rx+0.5))));
        D1_parity[i+1] = L_d0;

        /* symbol 6 */
        d0   = Enc2[i+1];
        tx   = d0 - 0.5;
        rx   = tx + SIGMA_26_4QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                  (exp(n*(rx+0.5)*(rx+0.5))));
```

COMPUTER PROGRAM LISTING APPENDIX

```
        D2_parity[i+1] = L_d0;
        i = i+1;
      }
#endif

/*mio*/
#ifdef R46_8QAM
    /*
     * I dimension:
     * d0 is MSB and d1 is LSB in a 4-AM:(d0,d1):
     *       01----00----10----11
     *     -1.5  -0.5   0.5   1.5
     * Q dimension:
     * d0 is MSB in a 2-AM:(d0):
     *       0-----1
     *     -0.5   0.5
     *
     *
     * We transmit one 4A-M symbol and one 2-AM symbol to emulate a 32QAM symbol.
     * 4 info bits and 2 parity bits are mapped to 2 8QAM symbols.
     *
     * INT_SIZE to be a multiple of 4
     */

    for(i = 0; i < INT_SIZE; i++)
      {
       /* symbol 1: 4AM */
       d0 = data[i];
       d1 = Enc1[i];
       tx   = d0 - d1 + 2*d0*d1 - 0.5;
       rx   = tx + SIGMA_4AM_of_46_8QAM * gasdev();
       n   = (-1.0) / (2 * SIGMA_4AM_of_46_8QAM * SIGMA_4AM_of_46_8QAM);
       L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                  (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
       L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                  (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
       D1_data[i]    = L_d0;
       D1_parity[i] = L_d1;

       /* symbol 2: 2AM */
       d0 = data[i+1];
       tx   = d0 - 0.5;
       rx   = tx + SIGMA_2AM_of_46_8QAM * gasdev();
       n   = (-1.0) / (2 * SIGMA_2AM_of_46_8QAM * SIGMA_2AM_of_46_8QAM);
       L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                  (exp(n*(rx+0.5)*(rx+0.5))));
       D1_data[i+1]    = L_d0;

       /* symbol 3: 4AM */
       d0 = data[i+2];
       d1 = Enc2[i+2];
       tx   = d0 - d1 + 2*d0*d1 - 0.5;
       rx   = tx + SIGMA_4AM_of_46_8QAM * gasdev();
       n   = (-1.0) / (2 * SIGMA_4AM_of_46_8QAM * SIGMA_4AM_of_46_8QAM);
       L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                  (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
       L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                  (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
       D1_data[i+2]    = L_d0;
       D2_parity[i+2] = L_d1;

       /* symbol 2: 4AM */
       d0 = data[i+3];
       tx   = d0 - 0.5;
       rx   = tx + SIGMA_2AM_of_46_8QAM * gasdev();
       n   = (-1.0) / (2 * SIGMA_2AM_of_46_8QAM * SIGMA_2AM_of_46_8QAM);
       L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                  (exp(n*(rx+0.5)*(rx+0.5))));
       D1_data[i+3]    = L_d0;
     D1_parity[i+1]    = 0.0;
     D1_parity[i+2]    = 0.0;
```

COMPUTER PROGRAM LISTING APPENDIX

```
    D1_parity[i+3]   = 0.0;
    D2_parity[i]   = 0.0;
    D2_parity[i+1]   = 0.0;
    D2_parity[i+3]   = 0.0;

       i = i+3;
      }
#endif

/*mio*/
#ifdef R26_8QAM
    /*
     * I dimension:
     * d0 is MSB and d1 is LSB in a 4-AM:(d0,d1):
     *       01----00----10----11
     *     -1.5  -0.5   0.5   1.5
     * Q dimension:
     * d0 is MSB in a 2-AM:(d0):
     *       0-----1
     *     -0.5   0.5
     *
     *
     * We transmit one 4A-M symbol and one 2-AM symbol to emulate a 8QAM symbol.
     * 2 info bits and 4 parity bits are mapped to 2 8QAM symbols.
     *
     * INT_SIZE to be a multiple of 2
     */

    for(i = 0; i < INT_SIZE; i++)
      {
    /* symbol 1: 4AM */

        d0 = data[i];
        d1 = Enc1[i];
        tx   = d0 - d1 + 2*d0*d1 - 0.5;
        rx   = tx + SIGMA_4AM_of_26_8QAM * gasdev();
        n   = (-1.0) / (2 * SIGMA_4AM_of_26_8QAM * SIGMA_4AM_of_26_8QAM);
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                   (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
        L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                   (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
        D1_data[i]   = L_d0;
        D1_parity[i] = L_d1;

    /* symbol 2: 2AM */

        d0 = Enc2[i];
        tx   = d0 - 0.5;
        rx   = tx + SIGMA_2AM_of_26_8QAM * gasdev();
        n   = (-1.0) / (2 * SIGMA_2AM_of_26_8QAM * SIGMA_2AM_of_26_8QAM);
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                   (exp(n*(rx+0.5)*(rx+0.5))));
        D2_parity[i]   = L_d0;

        /* symbol 3: 4AM */

        d0 = Enc1[i+1];
        d1 = Enc2[i+1];
        tx   = d0 - d1 + 2*d0*d1 - 0.5;
        rx   = tx + SIGMA_4AM_of_26_8QAM * gasdev();
        n   = (-1.0) / (2 * SIGMA_4AM_of_26_8QAM * SIGMA_4AM_of_26_8QAM);
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                   (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
        L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                   (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
        D1_parity[i+1] = L_d0;
        D2_parity[i+1] = L_d1;

    /* symbol 4: 2AM */

        d0 = data[i+1];
```

COMPUTER PROGRAM LISTING APPENDIX

```
        tx   = d0 - 0.5;
        rx   = tx + SIGMA_2AM_of_26_8QAM * gasdev();
        n   = (-1.0) / (2 * SIGMA_2AM_of_26_8QAM * SIGMA_2AM_of_26_8QAM);
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                   (exp(n*(rx+0.5)*(rx+0.5))));
        D1_data[i+1]   = L_d0;
        i=i+1;
        }
#endif

/*mio*/
#ifdef R13_8QAM
    /*
     * I dimension:
     * d0 is MSB and d1 is LSB in a 4-AM:(d0,d1):
     *       01----00----10----11
     *     -1.5  -0.5   0.5   1.5
     * Q dimension:
     * d0 is MSB in a 2-AM:(d0):
     *       0-----1
     *     -0.5   0.5
     *
     *
     * We transmit one 4A-M symbol and one 2-AM symbol to emulate a 8QAM symbol.
     * 1 info bits and 2 parity bits are mapped to 1 8QAM symbols.
     *
     * INT_SIZE to be a multiple of 1
     */

    for(i = 0; i < INT_SIZE; i++)
      {
    /* symbol 1: 4AM */

        d0 = data[i];
        d1 = Enc1[i];
        tx   = d0 - d1 + 2*d0*d1 - 0.5;
        rx   = tx + SIGMA_4AM_of_13_8QAM * gasdev();
        n   = (-1.0) / (2 * SIGMA_4AM_of_13_8QAM * SIGMA_4AM_of_13_8QAM);
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                   (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
        L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                   (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
        D1_data[i]   = L_d0;
        D1_parity[i] = L_d1;

    /* symbol 2: 2AM */

        d0 = Enc2[i];
        tx   = d0 - 0.5;
        rx   = tx + SIGMA_2AM_of_13_8QAM * gasdev();
        n   = (-1.0) / (2 * SIGMA_2AM_of_13_8QAM * SIGMA_2AM_of_13_8QAM);
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))) /
                   (exp(n*(rx+0.5)*(rx+0.5))));
        D2_parity[i]   = L_d0;

        }
#endif

/*mio*/
#ifdef R412_16QAM
    /*
     * Channel:  we transmit two 4-AM symbols to emulate a 16-QAM symbol.
     * 4 info bits and 8 parity bits are mapped to 3 16-QAM symbols which in
     * turn are simulated as 6 4-AM symbols to achieve 3bit/s/Hz
     * d0 is MSB and d1 is LSB in a 4-AM:(d0,d1) =   01----00--|--10----11
     *                                              -1.5  -0.5    0.5   1.5
     * INT_SIZE to be a multiple of 4
     */

    n   = (-1.0) / (2 * SIGMA_412_16QAM * SIGMA_412_16QAM);
    for(i = 0; i < INT_SIZE; i++)
```

COMPUTER PROGRAM LISTING APPENDIX

```
{
  /* symbol 1 */
  d0  = data[i];
  d1  = Enc1[i];
  tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
  rx   = tx + SIGMA_412_16QAM * gasdev();
  L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
             (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));
  L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
             (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));
  D1_data[i]    = L_d0;
  D1_parity[i] = L_d1;

  /* symbol 2 */
  d0   = data[i+1];
  d1   = Enc1[i+1];
  tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
  rx   = tx + SIGMA_412_16QAM * gasdev();
  L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
             (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));
  L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
             (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));
  D1_data[i+1] = L_d0;
  D1_parity[i+1] = L_d1;

  /* symbol 3 */
  d0   = Enc2[i];
  d1   = Enc2[i+1];
  tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
  rx   = tx + SIGMA_412_16QAM * gasdev();
  L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
             (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));
  L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
             (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));
  D2_parity[i]    = L_d0;
  D2_parity[i+1] = L_d1;

  /* symbol 4 */
  d0   = data[i+2];
  d1   = Enc1[i+2];
  tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
  rx   = tx + SIGMA_412_16QAM * gasdev();
  L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
             (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));
  L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
             (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));
  D1_data[i+2]    = L_d0;
  D1_parity[i+2] = L_d1;

    /* symbol 5 */
  d0   = Enc2[i+2];
  d1   = Enc2[i+3];
  tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
  rx   = tx + SIGMA_412_16QAM * gasdev();
  L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
             (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));
  L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
             (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));
  D2_parity[i+2] = L_d0;
  D2_parity[i+3] = L_d1;

  /* symbol 6 */
  d0   = data[i+3];
  d1   = Enc1[i+3];
  tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
  rx   = tx + SIGMA_412_16QAM * gasdev();
  L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
             (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));
  L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
             (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));
  D1_data[i+3]    = L_d0;
```

COMPUTER PROGRAM LISTING APPENDIX

```
        D1_parity[i+3] = L_d1;

                i = i+3;
      }
#endif

/*mio*/
#ifdef R515_32QAM
    /*
     * I dimension:
     * d0 is MSB and d2 is LSB in 8-AM:(d0,d1,d2):
     *       010---011---001---000---100---101---111---110
     *      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
     * Q dimension:
     * d0 is MSB and d1 is LSB in a 4-AM:(d0,d1):
     *       01----00----10----11
     *     -1.5  -0.5   0.5   1.5
     *
     *
     * We transmit one 8AM symbol and one 4AM symbol to emulate a 32QAM symbol.
     * 5 info bits and 10 parity bits are mapped to 3 32QAM symbols.
     *
     * INT_SIZE to be a multiple of 5
     */

    for(i = 0; i < INT_SIZE; i++)
      {
        /* symbol 1: 8AM */
    d0  = data[i];
    d1  = Enc1[i];
        d2  = Enc2[i];
        tx  = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
        rx  = tx + SIGMA_8AM_of_515_32QAM * gasdev();
        n   = (-1.0) / (2 * SIGMA_8AM_of_515_32QAM * SIGMA_8AM_of_515_32QAM);
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                    exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                   (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                    exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

        L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                    exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                   (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                    exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

        L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                    exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                   (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                    exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
    D1_data[i]      = L_d0;
    D1_parity[i]    = L_d1;
        D2_parity[i]    = L_d2;

        /* symbol 2: 4AM */
        d0 = data[i+1];
        d1 = Enc1[i+1];
        tx   = d0 - d1 + 2*d0*d1 - 0.5;
        rx   = tx + SIGMA_4AM_of_515_32QAM * gasdev();
        n   = (-1.0) / (2 * SIGMA_4AM_of_515_32QAM * SIGMA_4AM_of_515_32QAM);
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                   (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
        L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5))) /
                   (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
        D1_data[i+1]    = L_d0;
        D2_parity[i+1] = L_d1;

        /* symbol 3: 8AM */
    d0   = data[i+2];
    d1   = Enc1[i+2];
        d2    = Enc2[i+1];
        tx    = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
        rx  = tx + SIGMA_8AM_of_515_32QAM * gasdev();
```

COMPUTER PROGRAM LISTING APPENDIX

```
    n   = (-1.0) / (2 * SIGMA_8AM_of_515_32QAM * SIGMA_8AM_of_515_32QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
               (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
D1_data[i+2]   = L_d0;
D1_parity[i+2] = L_d1;
    D2_parity[i+1] = L_d2;

    /* symbol 4: 4AM */
    d0 = data[i+3];
    d1 = Enc2[i+2];
    tx   = d0 - d1 + 2*d0*d1 - 0.5;
    rx   = tx + SIGMA_4AM_of__515_32QAM * gasdev();
    n   = (-1.0) / (2 * SIGMA_4AM_of_515_32QAM * SIGMA_4AM_of_515_32QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))) /
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
    L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5))) /
               (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
    D1_data[i+3]   = L_d0;
    D2_parity[i+2] = L_d1;

    /* symbol 5: 8AM */
d0   = data[i+4];
d1   = Enc1[i+3];
    d2   = Enc2[i+3];
    tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx   = tx + SIGMA_8AM_of_515_32QAM * gasdev();
    n   = (-1.0) / (2 * SIGMA_8AM_of_515_32QAM * SIGMA_8AM_of_515_32QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
               (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
D1_data[i+4]   = L_d0;
D1_parity[i+3] = L_d1;
    D2_parity[i+3] = L_d2;

    /* symbol 4: 4AM */
    d0 = Enc1[i+4];
    d1 = Enc2[i+4];
    tx   = d0 - d1 + 2*d0*d1 - 0.5;
    rx   = tx + SIGMA_4AM_of__515_32QAM * gasdev();
    n   = (-1.0) / (2 * SIGMA_4AM_of_515_32QAM * SIGMA_4AM_of_515_32QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))) /
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5))));
    L_d1 = log((exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx-1.5)*(rx-1.5))) /
               (exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx+0.5)*(rx+0.5))));
    D1_parity[i+4] = L_d0;
    D2_parity[i+4] = L_d1;

    i = i+4;
```

COMPUTER PROGRAM LISTING APPENDIX

```
      }
#endif

/*mio*/
#ifdef R26_64QAM
    /*
     * Channel:
     * d0 is MSB and d2 is LSB in 8AM:(d0,d1,d2):
     *       010---011---001---000---100---101---111---110
     *      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
     */
    /*
     * Channel:  we transmit two 8AM symbols to emulate a 64QAM symbol.
     * 2 info bits and 4 parity bits are mapped to 1 64QAM symbols which in
     * turn are simulated as 2 8AM symbols to achieve 2bit/s/Hz.
     *
     * INT_SIZE to be a multiple of 2
     */
    n   = (-1.0) / (2 * SIGMA_26_64QAM * SIGMA_26_64QAM);
    for(i = 0; i < INT_SIZE; i++)
      {
    /* symbol 1 */

        d0  = data[i];
    d1  = Enc1[i];
        d2  = Enc2[i];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx   = tx + SIGMA_26_64QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                    exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                   (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                    exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

        L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                    exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                   (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                    exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

        L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                    exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                   (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                    exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
    D1_data[i]      = L_d0;
    D1_parity[i]    = L_d1;
        D2_parity[i]    = L_d2;

    /* symbol 2 */

        d0   = data[i+1];
    d1   = Enc1[i+1];
        d2   = Enc2[i+1];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx   = tx + SIGMA_26_64QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                    exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                   (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                    exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

        L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                    exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                   (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                    exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

        L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                    exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                   (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                    exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
    D1_data[i+1]      = L_d0;
    D1_parity[i+1]    = L_d1;
    D2_parity[i+1]    = L_d2;
        i = i+1;
```

COMPUTER PROGRAM LISTING APPENDIX

```
      }
#endif

/*mio*/
#ifdef R36_64QAM
    /*
     * Channel:
     * d0 is MSB and d2 is LSB in 8AM:(d0,d1,d2):
     *       010---011---001---000---100---101---111---110
     *      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
     */
    /*
     * Channel:  we transmit two 8AM symbols to emulate a 64QAM symbol.
     * 6 info bits and 6 parity bits are mapped to 2 64QAM symbols which in
     * turn are simulated as 4 8AM symbols to achieve 2bit/s/Hz.
     *
     * INT_SIZE to be a multiple of 6
     */
    n   = (-1.0) / (2 * SIGMA_36_64QAM * SIGMA_36_64QAM);
    for(i = 0; i < INT_SIZE; i++)
      {
    /* symbol 1 */

        d0  = data[i];
    d1  = data[i+1];
        d2  = Enc1[i];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx   = tx + SIGMA_36_64QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

        L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

        L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                  (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
    D1_data[i]      = L_d0;
    D1_data[i+1]    = L_d1;
        D1_parity[i]    = L_d2;

    /* symbol 2 */

        d0   = data[i+2];
    d1   = Enc1[i+2];
        d2   = Enc2[i+1];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx   = tx + SIGMA_36_64QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

        L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

        L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                  (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
    D1_data[i+2]      = L_d0;
    D1_parity[i+2]    = L_d1;
    D2_parity[i+1]    = L_d2;
```

COMPUTER PROGRAM LISTING APPENDIX

```
/* symbol 3 */

    d0  = data[i+3];
d1  = data[i+4];
    d2  = Enc2[i+3];
    tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx   = tx + SIGMA_36_64QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
               (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
D1_data[i+3]      = L_d0;
D1_data[i+4]      = L_d1;
    D2_parity[i+3]   = L_d2;

/* symbol 2 */

    d0   = data[i+5];
d1   = Enc1[i+4];
    d2   = Enc2[i+5];
    tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx   = tx + SIGMA_36_64QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
               (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
D1_data[i+5]      = L_d0;
D1_parity[i+4]    = L_d1;
D2_parity[i+5]    = L_d2;
D2_parity[i]      = 0.0;
D1_parity[i+1]    = 0.0;
D2_parity[i+2]    = 0.0;
D1_parity[i+3]    = 0.0;
D2_parity[i+4]    = 0.0;
D1_parity[i+5]    = 0.0;
    i = i+5;
  }
#endif

/*mio*/
#ifdef R721_128QAM
  /*
   * Q dimension:
   * d0 is MSB and d2 is LSB in 8-AM:(d0,d1,d2):
   *       010---011---001---000---100---101---111---110
   *      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
   * I dimension:
   * d0 is MSB and d3 is LSB in 16AM:(d0,d1,d2,d3):
   *
   *       0010---0011---0001---0000---0100---0101---0111---0110
   *       -7.5   -6.5   -5.5   -4.5   -3.5   -2.5   -1.5   -0.5
```

COMPUTER PROGRAM LISTING APPENDIX

```
 *
 *       1010---1011---1001---1000---1100---1101---1111---1110
 *        7.5    6.5    5.5    4.5    3.5    2.5    1.5    0.5
 *
 * INT_SIZE to be a multiple of 7
 */

for(i = 0; i < INT_SIZE; i++)
  {

    /* symbol 1 Q dimension: 8AM */

    d0  = data[i+2];
d1  = Enc1[i+1];
    d2  = Enc2[i+1];
    tx  = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
    rx  = tx + SIGMA_8AM_of_721_128QAM * gasdev();
    n   = (-1.0) / (2 * SIGMA_8AM_of_721_128QAM * SIGMA_8AM_of_721_128QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
               (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
D1_data[i+2]      = L_d0;
D1_parity[i+1]    = L_d1;
    D2_parity[i+1]    = L_d2;

/* symbol 2 I dimension: 16AM */

    d0 = data[i];
d1 = data[i+1];
d2 = Enc1[i];
    d3 = Enc2[i];
    tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
    tx = (d0 == 0 ? (tx - 4): (4 - tx));
    rx = tx + SIGMA_16AM_of_721_128QAM * gasdev();
    n  = (-1.0) / (2 * SIGMA_16AM_of_721_128QAM * SIGMA_16AM_of_721_128QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

    L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

    L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
               (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i]      = L_d0;
D1_data[i+1]    = L_d1;
    D1_parity[i]    = L_d2;
    D2_parity[i]    = L_d3;

/* symbol 3 Q dimension: 8AM */

    d0  = data[i+4];
d1  = Enc1[i+4];
    d2  = Enc2[i+3];
    tx  = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx  = tx + SIGMA_8AM_of_721_128QAM * gasdev();
    n   = (-1.0) / (2 * SIGMA_8AM_of_721_128QAM * SIGMA_8AM_of_721_128QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
               (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
D1_data[i+4]      = L_d0;
D1_parity[i+4]    = L_d1;
    D2_parity[i+3]    = L_d2;

/* symbol 4 I dimension: 16AM */

    d0 = data[i+3];
d1 = Enc1[i+3];
d2 = Enc1[i+2];
    d3 = Enc2[i+2];
    tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
    tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx = tx + SIGMA_16AM_of_721_128QAM * gasdev();
    n  = (-1.0) / (2 * SIGMA_16AM_of_721_128QAM * SIGMA_16AM_of_721_128QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

    L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

    L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
               (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i+3]      = L_d0;
D1_parity[i+3]    = L_d1;
    D1_parity[i+2]    = L_d2;
    D2_parity[i+2]    = L_d3;

/* symbol 5 Q dimension: 8AM */

    d0  = data[i+6];
d1  = Enc1[i+6];
    d2  = Enc2[i+6];
    tx  = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx  = tx + SIGMA_8AM_of_721_128QAM * gasdev();
    n   = (-1.0) / (2 * SIGMA_8AM_of_721_128QAM * SIGMA_8AM_of_721_128QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

    L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
               (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
D1_data[i+6]      = L_d0;
D1_parity[i+6]    = L_d1;
    D2_parity[i+6]    = L_d2;

/* symbol 6 I dimension: 16AM */

    d0 = data[i+5];
d1 = Enc1[i+5];
d2 = Enc2[i+5];
    d3 = Enc2[i+4];
    tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
    tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx = tx + SIGMA_16AM_of_721_128QAM * gasdev();
    n  = (-1.0) / (2 * SIGMA_16AM_of_721_128QAM * SIGMA_16AM_of_721_128QAM);
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
```

COMPUTER PROGRAM LISTING APPENDIX

```
                    (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                     exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                     exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                     exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

        L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                     exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                     exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                     exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
                    (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                     exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                     exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                     exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

        L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                     exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                     exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                     exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
                    (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                     exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                     exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                     exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

    D1_data[i+5]      = L_d0;
    D1_parity[i+5]    = L_d1;
       D2_parity[i+5]    = L_d2;
       D2_parity[i+4]    = L_d3;
       i = i+6;
      }
#endif


/*mio*/
#ifdef R824_256QAM
    /*
     * Channel:
     * d0 is MSB and d3 is LSB in 16AM:(d0,d1,d2,d3):
     *
     *       0010---0011---0001---0000---0100---0101---0111---0110
     *       -7.5   -6.5   -5.5   -4.5   -3.5   -2.5   -1.5   -0.5
     *
     *       1010---1011---1001---1000---1100---1101---1111---1110
     *        7.5    6.5    5.5    4.5    3.5    2.5    1.5    0.5
     *
     * Channel:  we transmit two 16AM symbols to emulate a 256QAM symbol.
     * 8 info bits and 16 parity bits are mapped to 3 256QAM symbols which in
     * turn are simulated as 6 16AM symbols to achieve 8/3bit/s/Hz.
     *
     * INT_SIZE to be a multiple of 8
     */

    n   = (-1.0) / (2 * SIGMA_824_256QAM * SIGMA_824_256QAM);

    /*
     * deinterleave data:
     */
    for(i = 0; i < INT_SIZE; i++)
      data_d[i] = data[i];
    r_deileava(data_d, rule);

    for(i = 0; i < INT_SIZE; i++)
      {
        /* symbol 1 */
    d0 = data_d[i];
    d1 = data_d[i+1];
    d2 = Enc1[i];
       d3 = Enc2[i];
       tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
       tx = (d0 == 0 ? (tx - 4): (4 - tx));
       /* Test the mapping to the 16AM constellation:
        * if (i < 500)
```

COMPUTER PROGRAM LISTING APPENDIX

```
     *  printf("\n(%d%d%d%d) = %f", (int)d0, (int)d1, (int)d2, (int)d3, tx);
     */
rx    = tx + SIGMA_58_256QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

    L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

    L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
               (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i]      = L_d0;
D1_data[i+1]    = L_d1;
    D1_parity[i]    = L_d2;
    D2_parity[i]    = L_d3;

    /* symbol 2 */
d0 = data_d[i+2];
d1 = Enc1[i+2];
    d2 = Enc1[i+1];
    d3 = Enc2[i+1];
    tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
    tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx    = tx + SIGMA_824_256QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

    L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

    L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
               (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));
D1_data[i+2]    = L_d0;
D1_parity[i+2] = L_d1;
D1_parity[i+1] = L_d2;
D2_parity[i+1] = L_d3;

    /* symbol 3 */
d0   = data_d[i+3];
d1   = Enc1[i+3];
d2   = Enc2[i+3];
    d3   = Enc1[i+2];
    tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
    tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx   = tx + SIGMA_824_256QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

    L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

    L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
               (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));
D1_data[i+3]      = L_d0;
D1_parity[i+3]    = L_d1;
D2_parity[i+3]    = L_d2;
    D2_parity[i+2]    = L_d3;

/* symbol 4 */
d0   = data_d[i+4];
```

COMPUTER PROGRAM LISTING APPENDIX

```
d1   = data_d[i+5];
   d2   = Enc1[i+4];
   d3   = Enc2[i+4];
   tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
   tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx   = tx + SIGMA_824_256QAM * gasdev();
   L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
              exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
              exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
              exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
             (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
              exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
              exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
              exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

   L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
              exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
              exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
              exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
             (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
              exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
              exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
              exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

   L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
              exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
              exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
              exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
             (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
              exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
              exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
              exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

   L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
              exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
              exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
              exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
             (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
              exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
              exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
              exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));
D1_data[i+4]   = L_d0;
D1_data[i+5]   = L_d1;
   D1_parity[i+4] = L_d2;
   D2_parity[i+4] = L_d3;

   /* symbol 5 */

   d0   = data_d[i+6];
d1   = Enc1[i+6];
d2   = Enc1[i+5];
   d3   = Enc2[i+5];
   tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
   tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx   = tx + SIGMA_824_256QAM * gasdev();
   L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
              exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
              exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
              exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
             (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
              exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
              exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
              exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

   L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
              exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
              exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
              exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
             (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
              exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
              exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

    L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

    L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
               (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));
D1_data[i+6]      = L_d0;
D1_parity[i+6]    = L_d1;
D1_parity[i+5]    = L_d2;
    D2_parity[i+5]    = L_d3;

/* symbol 4 */
d0    = data_d[i+7];
d1    = Enc1[i+7];
    d2    = Enc2[i+7];
    d3    = Enc2[i+6];
    tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
    tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx    = tx + SIGMA_824_256QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

    L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

    L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

    L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
               (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));
D1_data[i+7]    = L_d0;
D1_parity[i+7]    = L_d1;
    D2_parity[i+7] = L_d2;
    D2_parity[i+6] = L_d3;
```

COMPUTER PROGRAM LISTING APPENDIX

```
        i = i+7;
      }
    /*
     * interleave data:
     */
    r_ileav(D1_data, rule);
#endif

/*mio*/

#ifdef R69_512QAM
    /*
     * Interleaver should be a multiple of 3, e.g., 6144
     *
     * Q dimension:
     * d0 is MSB and d4 is LSB in 32AM:(d0,d1,d2,d3,d4):
     *       00010--00011--00001--00000--00100--00101--00111--00110
     *       -15.5  -14.5  -13.5  -12.5  -11.5  -10.5  -9.5   -8.5
     *
     *       01010--01011--01001--01000--01100--01101--01111--01110
     *       -0.5   -1.5   -2.5   -3.5   -4.5   -5.5   -6.5   -7.5
     *
     *       11010--11011--11001--11000--11100--11101--11111--11110
     *       0.5    1.5    2.5    3.5    4.5    5.5    6.5    7.5
     *
     *       10010--10011--10001--10000--10100--10101--10111--10110
     *       15.5   14.5   13.5   12.5   11.5   10.5   9.5    8.5
     *
     * I dimension:
     * d0 is MSB and d3 is LSB in 16AM:(d0,d1,d2,d3):
     *
     *       0010---0011---0001---0000---0100---0101---0111---0110
     *       -7.5   -6.5   -5.5   -4.5   -3.5   -2.5   -1.5   -0.5
     *
     *       1010---1011---1001---1000---1100---1101---1111---1110
     *        7.5    6.5    5.5    4.5    3.5    2.5    1.5    0.5
     */

    /*
     * deinterleave data:
     */
    for(i = 0; i < INT_SIZE; i++)
      data_d[i] = data[i];
    r_deileava(data_d, rule);

    for(i = 0; i < INT_SIZE; i++)
      {
    /* symbol 1, Q dimension: 32AM */
    d0  = data_d[i];
    d1  = data_d[i+1];
        d2  = Enc1[i];
        d3  = Enc2[i];
        d4  = Enc1[i+1];
        tx  = 2*d2 - 2*d3 + 4*d2*d3 - 1.0 + (((2*d2-1)*(2*d3-1))<0?(d4-0.5):(0.5-d4));
        tx = (d1 == 0 ? (tx - 4): (4 - tx));
        tx = (d0 == 0 ? (tx - 8): (8 - tx));
    rx  = tx + SIGMA_32AM_of_39_512QAM * gasdev();
        n   = (-1.0) / (2 * SIGMA_32AM_of_39_512QAM * SIGMA_32AM_of_39_512QAM);
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                    exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                    exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                    exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                    exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
                    exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
                    exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
                    exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
                   (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                    exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                    exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                    exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
                    exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

L_d1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))})/
           (exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

L_d2 = log((exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)))/
           (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5))));

L_d3 = log((exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
           (exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-10.5)*(rx-10.5))));

L_d4 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-13.5)*(rx-13.5)))/
           (exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+8.5)*(rx+8.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-7.5)*(rx-7.5))+exp(n*(rx-8.5)*(rx-8.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-11.5)*(rx-11.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-15.5)*(rx-15.5))));
```

COMPUTER PROGRAM LISTING APPENDIX

```
    D1_data[i]      = L_d0;
    D1_data[i+1]    = L_d1;
       D1_parity[i]    = L_d2;
       D2_parity[i]    = L_d3;
       D1_parity[i+1] = L_d4;

       /* symbol 1, I dimension: 16AM */
    d0 = data_d[i+2];
    d1 = Enc1[i+2];
    d2 = Enc2[i+2];
       d3 = Enc2[i+1];
       tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
       tx = (d0 == 0 ? (tx - 4): (4 - tx));
    rx = tx + SIGMA_16AM_of_39_512QAM * gasdev();
       n  = (-1.0) / (2 * SIGMA_16AM_of_39_512QAM * SIGMA_16AM_of_39_512QAM);
       L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                   exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                   exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
                  (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                   exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                   exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

       L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                   exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                   exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                   exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                   exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

       L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                   exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
                  (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                   exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                   exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

       L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                   exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
                  (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                   exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                   exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

    D1_data[i+2]    = L_d0;
    D1_parity[i+2] = L_d1;
       D2_parity[i+2] = L_d2;
       D2_parity[i+1] = L_d3;
       i = i+2;
      }
    /*
     * interleave data:
     */
    r_ileav(D1_data, rule);

#endif


/*mio*/
#ifdef R12_16QAM
    /*
     * Channel:  we transmit two 4-AM symbols to emulate a 16-QAM symbol.
```

COMPUTER PROGRAM LISTING APPENDIX

```
     * 2 info bits and 2 parity bits are mapped to 1 16-QAM symbols which in
     * turn are simulated as 2 4-AM symbols to achieve 2bit/s/Hz
     * d0 is MSB and d1 is LSB in a 4-AM:(d0,d1) = 01--00-|-10--11
     *                                              -3  -1    1   3
     * INT_SIZE to be a multiple of 2
     */

    n   = (-1.0) / (2 * SIGMA_12_16QAM * SIGMA_12_16QAM);
    for(i = 0; i < INT_SIZE; i++)
      {
        /* symbol 1 */
        d0  = data[i];
        d1  = Enc1[i];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
        rx   = tx + SIGMA_12_16QAM * gasdev();
        L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
                   (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));
        L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
                   (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));
        D1_data[i]   = L_d0;
        D1_parity[i] = L_d1;
        D2_parity[i] = 0.0;

        /* symbol 2 */
        d0   = data[i+1];
        d1   = Enc2[i+1];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
        rx   = tx + SIGMA_12_16QAM * gasdev();
        L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
                   (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));
        L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
                   (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));
        D1_data[i+1]   = L_d0;
        D2_parity[i+1] = L_d1;
        D1_parity[i+1] = 0.0;

        i = i+1;
      }
#endif

#ifdef R34_16QAM
    /*
     * Channel:  we transmit two 4-AM symbols to emulate a 16-QAM symbol.
     * 6 info bits and 2 parity bits are mapped to 2 16-QAM symbols which in
     * turn are simulated as 4 4-AM symbols to achieve 3bit/s/Hz
     * d0 is MSB and d1 is LSB in a 4-AM:(d0,d1) = 01--00-|-10--11
     *                                              -3  -1    1   3
     * INT_SIZE to be a multiple of 6
     */

    n   = (-1.0) / (2 * SIGMA_34_16QAM * SIGMA_34_16QAM);
    for(i = 0; i < INT_SIZE; i++)
      {
        /* symbol 1 */
        d0  = data[i];
        d1  = data[i+1];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
        rx   = tx + SIGMA_34_16QAM * gasdev();
        L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
                   (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));
        L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
                   (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));
        D1_data[i]   = L_d0;
        D1_data[i+1] = L_d1;

        /* symbol 2 */
        d0   = data[i+2];
        d1   = Enc1[i+1];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
        rx   = tx + SIGMA_34_16QAM * gasdev();
        L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
```

COMPUTER PROGRAM LISTING APPENDIX

```
                (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));
        L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
                (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));
        D1_data[i+2] = L_d0;
        D1_parity[i+1] = L_d1;

        /* symbol 3 */
        d0   = data[i+3];
        d1   = data[i+4];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
        rx   = tx + SIGMA_34_16QAM * gasdev();
        L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
                (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));
        L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
                (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));
        D1_data[i+3] = L_d0;
        D1_data[i+4] = L_d1;

        /* symbol 4 */
        d0   = data[i+5];
        d1   = Enc2[i+4];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0;
        rx   = tx + SIGMA_34_16QAM * gasdev();
        L_d0 = log((exp(n*(rx-1)*(rx-1))+exp(n*(rx-3)*(rx-3))) /
                (exp(n*(rx+1)*(rx+1))+exp(n*(rx+3)*(rx+3))));
        L_d1 = log((exp(n*(rx+3)*(rx+3))+exp(n*(rx-3)*(rx-3))) /
                (exp(n*(rx-1)*(rx-1))+exp(n*(rx+1)*(rx+1))));
        D1_data[i+5]   = L_d0;
        D2_parity[i+4] = L_d1;
        D1_parity[i]   = 0.0;
        D1_parity[i+2] = 0.0;
        D1_parity[i+3] = 0.0;
        D1_parity[i+4] = 0.0;
        D1_parity[i+5] = 0.0;
        D2_parity[i]   = 0.0;
        D2_parity[i+1] = 0.0;
        D2_parity[i+2] = 0.0;
        D2_parity[i+3] = 0.0;
        D2_parity[i+5] = 0.0;
        i = i+5;
      }
#endif

#ifdef R56_64QAM
    /*
     * Channel:
     * d0 is MSB and d2 is LSB in 8AM:(d0,d1,d2):
     *       010---011---001---000---100---101---111---110
     *      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
     */
    /*
     * Channel:  we transmit two 8AM symbols to emulate a 64QAM symbol.
     * 10 info bits and 2 parity bits are mapped to 2 64QAM symbols which in
     * turn are simulated as 4 8AM symbols to achieve 5bit/s/Hz.
     *
     * INT_SIZE to be a multiple of 10
     */
    n   = (-1.0) / (2 * SIGMA_56_64QAM * SIGMA_56_64QAM);
    for(i = 0; i < INT_SIZE; i++)
      {
        /* symbol 1 */
        d0  = data[i];
        d1  = data[i+1];
        d2  = Enc1[i];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
        rx   = tx + SIGMA_56_64QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                  exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                 (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                  exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));
```

COMPUTER PROGRAM LISTING APPENDIX

```
L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
          (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
          (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
D1_data[i]     = L_d0;
D1_data[i+1]   = L_d1;
D1_parity[i]   = L_d2;
D1_parity[i+1] = 0;
D2_parity[i]   = 0;
D2_parity[i+1] = 0;

/* symbol 2 */
d0   = data[i+2];
d1   = data[i+3];
d2   = data[i+4];
tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx   = tx + SIGMA_56_64QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
          (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
          (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
          (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
D1_data[i+2]   = L_d0;
D1_data[i+3]   = L_d1;
D1_data[i+4]   = L_d2;
D1_parity[i+2] = 0;
D1_parity[i+3] = 0;
D1_parity[i+4] = 0;
D2_parity[i+2] = 0;
D2_parity[i+3] = 0;
D2_parity[i+4] = 0;

/* symbol 3 */
d0   = data[i+5];
d1   = data[i+6];
d2   = Enc2[i+5];
tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
rx   = tx + SIGMA_56_64QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
          (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
          (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
          (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
D1_data[i+5]   = L_d0;
D1_data[i+6]   = L_d1;
D2_parity[i+5] = L_d2;
D2_parity[i+6] = 0;
D1_parity[i+5] = 0;
```

COMPUTER PROGRAM LISTING APPENDIX

```
        D1_parity[i+6] = 0;

        /* symbol 4 */
        d0   = data[i+7];
        d1   = data[i+8];
        d2   = data[i+9];
        tx   = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
        rx   = tx + SIGMA_56_64QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

        L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

        L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
                  (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
        D1_data[i+7]   = L_d0;
        D1_data[i+8]   = L_d1;
        D1_data[i+9]   = L_d2;
        D1_parity[i+7] = 0;
        D1_parity[i+8] = 0;
        D1_parity[i+9] = 0;
        D2_parity[i+7] = 0;
        D2_parity[i+8] = 0;
        D2_parity[i+9] = 0;

        i = i+9;
      }
#endif


#ifdef R57_128QAM
    /*
     * Q dimension:
     * d0 is MSB and d2 is LSB in 8-AM:(d0,d1,d2):
     *       010---011---001---000---100---101---111---110
     *      -3.5  -2.5  -1.5  -0.5   0.5   1.5   2.5   3.5
     * I dimension:
     * d0 is MSB and d3 is LSB in 16AM:(d0,d1,d2,d3):
     *
     *       0010---0011---0001---0000---0100---0101---0111---0110
     *       -7.5   -6.5   -5.5   -4.5   -3.5   -2.5   -1.5   -0.5
     *
     *       1010---1011---1001---1000---1100---1101---1111---1110
     *        7.5    6.5    5.5    4.5    3.5    2.5    1.5    0.5
     *
     * INT_SIZE to be a multiple of 5
     */

    for(i = 0; i < INT_SIZE; i++)
      {
        /* Q dimension: 8AM */
        d0  = data[i];
        d1  = data[i+1];
        d2  = Enc1[i];
        tx  = 2*d0 - 2*d1 + 4*d0*d1 - 1.0 + (((2*d0-1)*(2*d1-1))<0?(d2-0.5):(0.5-d2));
        rx  = tx + SIGMA_8AM_of_128QAM * gasdev();
        n   = (-1.0) / (2 * SIGMA_8AM_of_128QAM * SIGMA_8AM_of_128QAM);
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5))));

        L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
```

COMPUTER PROGRAM LISTING APPENDIX

```
            (exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
             exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5))));

L_d2 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx-1.5)*(rx-1.5))+exp(n*(rx-2.5)*(rx-2.5)))/
           (exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5))));
 D1_data[i]      = L_d0;
 D1_data[i+1]    = L_d1;
D1_parity[i]    = L_d2;
D1_parity[i+1] = 0.0;
D1_parity[i+2] = 0.0;
D1_parity[i+3] = 0.0;
D1_parity[i+4] = 0.0;

 /* I dimension: 16AM */
 d0 = data[i+2];
 d1 = data[i+3];
 d2 = data[i+4];
d3 = Enc2[i];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4): (4 - tx));
 rx = tx + SIGMA_16AM_of_128QAM * gasdev();
n  = (-1.0) / (2 * SIGMA_16AM_of_128QAM * SIGMA_16AM_of_128QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
           (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
           (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
            exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
           (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
            exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
            exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
           (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
            exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

 D1_data[i+2]    = L_d0;
 D1_data[i+3]    = L_d1;
D1_data[i+4]    = L_d2;
D2_parity[i]    = L_d3;
D2_parity[i+1] = 0.0;
D2_parity[i+2] = 0.0;
D2_parity[i+3] = 0.0;
D2_parity[i+4] = 0.0;

i = i+4;
}
```

COMPUTER PROGRAM LISTING APPENDIX

```
#endif

#ifdef R58_256QAM
    /*
     * Channel:
     * d0 is MSB and d3 is LSB in 16AM:(d0,d1,d2,d3):
     *
     *       0010---0011---0001---0000---0100---0101---0111---0110
     *       -7.5   -6.5   -5.5   -4.5   -3.5   -2.5   -1.5   -0.5
     *
     *       1010---1011---1001---1000---1100---1101---1111---1110
     *        7.5    6.5    5.5    4.5    3.5    2.5    1.5    0.5
     *
     * Channel:  we transmit two 16AM symbols to emulate a 256QAM symbol.
     * 10 info bits and 6 parity bits are mapped to 2 256QAM symbols which in
     * turn are simulated as 4 16AM symbols to achieve 6bit/s/Hz.
     *
     * INT_SIZE to be a multiple of 10
     */

    n   = (-1.0) / (2 * SIGMA_58_256QAM * SIGMA_58_256QAM);

    /*
     * deinterleave data:
     */
    for(i = 0; i < INT_SIZE; i++)
      data_d[i] = data[i];
    r_deileava(data_d, rule);

    for(i = 0; i < INT_SIZE; i++)
      {
        /* symbol 1 */
        d0 = data_d[i];
        d1 = data_d[i+1];
        d2 = data_d[i+2];
        d3 = Enc1[i];
        tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
        tx = (d0 == 0 ? (tx - 4): (4 - tx));
        /* Test the mapping to the 16AM constellation:
         * if (i < 500)
         *  printf("\n(%d%d%d%d) = %f", (int)d0, (int)d1, (int)d2, (int)d3, tx);
         */
        rx   = tx + SIGMA_58_256QAM * gasdev();
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                    exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                    exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                    exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
                   (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                    exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                    exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                    exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

        L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                    exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                    exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                    exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                   (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                    exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                    exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                    exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

        L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                    exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                    exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                    exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
                   (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                    exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                    exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                    exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));
```

COMPUTER PROGRAM LISTING APPENDIX

```
L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
           exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
          (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
           exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i]     = L_d0;
D1_data[i+1]   = L_d1;
D1_data[i+2]   = L_d2;
D1_parity[i]   = L_d3;
D1_parity[i+1] = 0;
D1_parity[i+2] = 0;
D2_parity[i]   = 0;
D2_parity[i+1] = 0;

/* symbol 2 */
d0 = data_d[i+3];
d1 = data_d[i+4];
d2 = Enc2[i+2];
d3 = Enc1[i+4];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx   = tx + SIGMA_58_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
          (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
           exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
           exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
          (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
           exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
           exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
           exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
          (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
           exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
           exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
          (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
           exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));
D1_data[i+3]   = L_d0;
D1_data[i+4]   = L_d1;
D2_parity[i+2] = L_d2;
D1_parity[i+4] = L_d3;
D1_parity[i+3] = 0;
D2_parity[i+3] = 0;
D2_parity[i+4] = 0;

/* symbol 3 */
d0   = data_d[i+5];
```

COMPUTER PROGRAM LISTING APPENDIX

```
 d1   = data_d[i+6];
 d2   = data_d[i+7];
d3   = Enc2[i+5];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4): (4 - tx));
 rx   = tx + SIGMA_58_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
          (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
           exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
           exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
          (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
           exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
           exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
           exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
          (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
           exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
           exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
          (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
           exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));
 D1_data[i+5]   = L_d0;
 D1_data[i+6]   = L_d1;
 D1_data[i+7]   = L_d2;
D2_parity[i+5] = L_d3;
D2_parity[i+6] = 0;
D2_parity[i+7] = 0;
D1_parity[i+5] = 0;
D1_parity[i+6] = 0;

 /* symbol 4 */
 d0   = data_d[i+8];
 d1   = data_d[i+9];
d2   = Enc1[i+7];
d3   = Enc2[i+9];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4): (4 - tx));
 rx   = tx + SIGMA_58_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
          (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
           exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
           exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
```

COMPUTER PROGRAM LISTING APPENDIX

```
                (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                 exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                 exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

        L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                 exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
                (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                 exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                 exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                 exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

        L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                 exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                 exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                 exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
                (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                 exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                 exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                 exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));
        D1_data[i+8]   = L_d0;
        D1_data[i+9]   = L_d1;
        D1_parity[i+7] = L_d2;
        D2_parity[i+9] = L_d3;
        D1_parity[i+8] = 0;
        D1_parity[i+9] = 0;
        D2_parity[i+7] = 0;
        D2_parity[i+8] = 0;

        i = i+9;
      }
    /*
     * interleave data:
     */
    r_ileav(D1_data, rule);
#endif


#ifdef R68_256QAM
    /*
     * Channel:
     * d0 is MSB and d3 is LSB in 16AM:(d0,d1,d2,d3):
     *
     *       0010---0011---0001---0000---0100---0101---0111---0110
     *       -7.5   -6.5   -5.5   -4.5   -3.5   -2.5   -1.5   -0.5
     *
     *       1010---1011---1001---1000---1100---1101---1111---1110
     *        7.5    6.5    5.5    4.5    3.5    2.5    1.5    0.5
     *
     * Channel:  we transmit two 16AM symbols to emulate a 256QAM symbol.
     * 6 info bits and 2 parity bits are mapped to one 256QAM symbol which in
     * turn is simulated as 2 16AM symbols to achieve 6bit/s/Hz.
     *
     * INT_SIZE to be a multiple of 6
     */
    n   = (-1.0) / (2 * SIGMA_68_256QAM * SIGMA_68_256QAM);

    /*
     * deinterleave data:
     */
    for(i = 0; i < INT_SIZE; i++)
      data_d[i] = data[i];
    r_deileava(data_d, rule);

    for(i = 0; i < INT_SIZE; i++)
      {
        /* symbol 1 */
        d0 = data_d[i];
        d1 = data_d[i+1];
```

COMPUTER PROGRAM LISTING APPENDIX

```
d2 = data_d[i+2];
d3 = Enc1[i];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx   = tx + SIGMA_68_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
          (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
           exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
           exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
          (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
           exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
           exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
           exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
          (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
           exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
           exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
          (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
           exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

D1_data[i]     = L_d0;
D1_data[i+1]   = L_d1;
D1_data[i+2]   = L_d2;
D1_parity[i]   = L_d3;
D1_parity[i+1] = 0;
D1_parity[i+2] = 0;
D2_parity[i]   = 0;
D2_parity[i+1] = 0;
D2_parity[i+2] = 0;

/* symbol 2 */
d0 = data_d[i+3];
d1 = data_d[i+4];
d2 = data_d[i+5];
d3 = Enc2[i+3];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4): (4 - tx));
rx   = tx + SIGMA_68_256QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
          (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
           exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
           exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
           exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
```

```
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
               (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

        L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
               (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

        L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
               (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));
        D1_data[i+3]   = L_d0;
        D1_data[i+4]   = L_d1;
        D1_data[i+5]   = L_d2;
        D2_parity[i+3] = L_d3;
        D2_parity[i+4] = 0;
        D2_parity[i+5] = 0;
        D1_parity[i+3] = 0;
        D1_parity[i+4] = 0;
        D1_parity[i+5] = 0;

        i = i+5;
      }

    /*
     * interleave data:
     */
    r_ileav(D1_data, rule);

#endif


#ifdef R69_512QAM
    /*
     * Interleaver should be a multiple of 12, e.g., 6144
     *
     * Q dimension:
     * d0 is MSB and d4 is LSB in 32AM:(d0,d1,d2,d3,d4):
     *       00010--00011--00001--00000--00100--00101--00111--00110
     *       -15.5  -14.5  -13.5  -12.5  -11.5  -10.5  -9.5   -8.5
     *
     *       01010--01011--01001--01000--01100--01101--01111--01110
     *       -0.5   -1.5   -2.5   -3.5   -4.5   -5.5   -6.5   -7.5
     *
     *       11010--11011--11001--11000--11100--11101--11111--11110
     *       0.5    1.5    2.5    3.5    4.5    5.5    6.5    7.5
     *
     *       10010--10011--10001--10000--10100--10101--10111--10110
     *       15.5   14.5   13.5   12.5   11.5   10.5   9.5    8.5
     *
     * I dimension:
     * d0 is MSB and d3 is LSB in 16AM:(d0,d1,d2,d3):
     *
     *       0010---0011---0001---0000---0100---0101---0111---0110
     *       -7.5   -6.5   -5.5   -4.5   -3.5   -2.5   -1.5   -0.5
     *
     *       1010---1011---1001---1000---1100---1101---1111---1110
     *        7.5    6.5    5.5    4.5    3.5    2.5    1.5    0.5
```

COMPUTER PROGRAM LISTING APPENDIX

```
 */

/*
 * deinterleave data:
 */
for(i = 0; i < INT_SIZE; i++)
  data_d[i] = data[i];
r_deileava(data_d, rule);

for(i = 0; i < INT_SIZE; i++)
  {
    /* symbol 1, Q dimension: 32AM */
    d0  = data_d[i];
    d1  = data_d[i+1];
   d2  = data_d[i+2];
   d3  = data_d[i+3];
   d4  = Enc1[i];
   tx  = 2*d2 - 2*d3 + 4*d2*d3 - 1.0 + (((2*d2-1)*(2*d3-1))<0?(d4-0.5):(0.5-d4));
   tx = (d1 == 0 ? (tx - 4): (4 - tx));
   tx = (d0 == 0 ? (tx - 8): (8 - tx));
    rx  = tx + SIGMA_32AM_of_512QAM * gasdev();
   n   = (-1.0) / (2 * SIGMA_32AM_of_512QAM * SIGMA_32AM_of_512QAM);
   L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
              exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
              exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
              exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
              exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
              exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
              exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
              exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
             (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
              exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
              exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
              exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
              exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
              exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
              exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
              exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

   L_d1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
              exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
              exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
              exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
              exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
              exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
              exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
              exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)))/
             (exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
              exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
              exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
              exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)) +
              exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
              exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
              exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
              exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

   L_d2 = log((exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
              exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
              exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
              exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
              exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
              exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
              exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
              exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)))/
             (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
              exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
              exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
              exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
              exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
              exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
              exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5))));

L_d3 = log((exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
           (exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-10.5)*(rx-10.5))));

L_d4 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-13.5)*(rx-13.5)))/
           (exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+8.5)*(rx+8.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-7.5)*(rx-7.5))+exp(n*(rx-8.5)*(rx-8.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-11.5)*(rx-11.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-15.5)*(rx-15.5))));


 D1_data[i]     = L_d0;
 D1_data[i+1]   = L_d1;
D1_data[i+2]   = L_d2;
D1_data[i+3]   = L_d3;
D1_parity[i]   = L_d4;
D1_parity[i+1] = 0.0;
D1_parity[i+2] = 0.0;
D1_parity[i+3] = 0.0;
D1_parity[i+5] = 0.0;

 /* symbol 1, I dimension: 16AM */
 d0 = data_d[i+4];
 d1 = data_d[i+5];
 d2 = Enc1[i+4];
d3 = Enc2[i+2];
tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
tx = (d0 == 0 ? (tx - 4): (4 - tx));
 rx = tx + SIGMA_16AM_of_512QAM * gasdev();
n  = (-1.0) / (2 * SIGMA_16AM_of_512QAM * SIGMA_16AM_of_512QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
           (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
           (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
            exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
           (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
            exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
            exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
           (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
            exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

 D1_data[i+4]    = L_d0;
 D1_data[i+5]    = L_d1;
D1_parity[i+4] = L_d2;
D2_parity[i+2] = L_d3;
D2_parity[i]   = 0.0;
D2_parity[i+1] = 0.0;
D2_parity[i+3] = 0.0;
D2_parity[i+4] = 0.0;
D2_parity[i+5] = 0.0;

 /* symbol 2, Q dimension: 32AM */
 d0  = data_d[i+6];
 d1  = data_d[i+7];
d2  = data_d[i+8];
d3  = data_d[i+9];
d4  = Enc2[i+6];
tx  = 2*d2 - 2*d3 + 4*d2*d3 - 1.0 + (((2*d2-1)*(2*d3-1))<0?(d4-0.5):(0.5-d4));
tx = (d1 == 0 ? (tx - 4): (4 - tx));
tx = (d0 == 0 ? (tx - 8): (8 - tx));
 rx  = tx + SIGMA_32AM_of_512QAM * gasdev();
n   = (-1.0) / (2 * SIGMA_32AM_of_512QAM * SIGMA_32AM_of_512QAM);
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
           (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

L_d1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)))/
           (exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

L_d2 = log((exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)))/
           (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5))));

L_d3 = log((exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
           (exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-10.5)*(rx-10.5))));

L_d4 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-13.5)*(rx-13.5)))/
           (exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+8.5)*(rx+8.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-7.5)*(rx-7.5))+exp(n*(rx-8.5)*(rx-8.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-11.5)*(rx-11.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-15.5)*(rx-15.5))));


 D1_data[i+6]   = L_d0;
 D1_data[i+7]   = L_d1;
D1_data[i+8]   = L_d2;
D1_data[i+9]   = L_d3;
D2_parity[i+6] = L_d4;
D2_parity[i+7] = 0.0;
D2_parity[i+8] = 0.0;
D2_parity[i+9] = 0.0;
D2_parity[i+11] = 0.0;

 /* symbol 2, I dimension: 16AM */
 d0 = data_d[i+10];
 d1 = data_d[i+11];
 d2 = Enc2[i+10];
```

```
        d3 = Enc1[i+8];
        tx = 2*d1 - 2*d2 + 4*d1*d2 - 1.0 + (((2*d1-1)*(2*d2-1))<0?(d3-0.5):(0.5-d3));
        tx = (d0 == 0 ? (tx - 4): (4 - tx));
         rx = tx + SIGMA_16AM_of_512QAM * gasdev();
        n  = (-1.0) / (2 * SIGMA_16AM_of_512QAM * SIGMA_16AM_of_512QAM);
        L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                   exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                   exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
                  (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                   exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                   exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                   exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

        L_d1 = log((exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                   exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)))/
                  (exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                   exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                   exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                   exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))));

        L_d2 = log((exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                   exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)))/
                  (exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                   exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                   exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5))));

        L_d3 = log((exp(n*(rx+5.5)*(rx+5.5))+exp(n*(rx+6.5)*(rx+6.5)) +
                   exp(n*(rx+1.5)*(rx+1.5))+exp(n*(rx+2.5)*(rx+2.5)) +
                   exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                   exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)))/
                  (exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+4.5)*(rx+4.5)) +
                   exp(n*(rx+3.5)*(rx+3.5))+exp(n*(rx+0.5)*(rx+0.5)) +
                   exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                   exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-7.5)*(rx-7.5))));

         D1_data[i+10]    = L_d0;
         D1_data[i+11]    = L_d1;
        D2_parity[i+10] = L_d2;
        D1_parity[i+8]  = L_d3;
        D1_parity[i+6]  = 0.0;
        D1_parity[i+7]  = 0.0;
        D1_parity[i+9]  = 0.0;
        D1_parity[i+10] = 0.0;
        D1_parity[i+11] = 0.0;

        i = i+11;
      }
    /*
     * interleave data:
     */
    r_ileav(D1_data, rule);

#endif


#ifdef R710_1024QAM
    /*
     * Use S2044_33_1 interleaver (multiple of 14)
     *
     * I and Q dimensions:
     * d0 is MSB and d4 is LSB in 32AM:(d0,d1,d2,d3,d4):
     *       00010--00011--00001--00000--00100--00101--00111--00110
     *       -15.5  -14.5  -13.5  -12.5  -11.5  -10.5  -9.5   -8.5
     *
     *       01010--01011--01001--01000--01100--01101--01111--01110
```

COMPUTER PROGRAM LISTING APPENDIX

```
 *       -0.5    -1.5    -2.5    -3.5    -4.5    -5.5    -6.5    -7.5
 *
 *       11010--11011--11001--11000--11100--11101--11111--11110
 *       0.5     1.5     2.5     3.5     4.5     5.5     6.5     7.5
 *
 *       10010--10011--10001--10000--10100--10101--10111--10110
 *       15.5    14.5    13.5    12.5    11.5    10.5    9.5     8.5
 *
 */

n   = (-1.0) / (2 * SIGMA_710_1024QAM * SIGMA_710_1024QAM);
for(i = 0; i < INT_SIZE; i++)
  {
    /* symbol 1, Q dimension */
    d0  = data[i];
    d1  = data[i+1];
    d2  = data[i+2];
    d3  = data[i+3];
    d4  = Encl[i];
    tx  = 2*d2 - 2*d3 + 4*d2*d3 - 1.0 + (((2*d2-1)*(2*d3-1))<0?(d4-0.5):(0.5-d4));
    tx = (d1 == 0 ? (tx - 4): (4 - tx));
    tx = (d0 == 0 ? (tx - 8): (8 - tx));
    rx  = tx + SIGMA_710_1024QAM * gasdev();
    L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
                exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
                exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
                exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
                exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
                exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
                exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
                exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

    L_d1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5))})/
               (exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
                exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
                exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
                exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)) +
                exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
                exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
                exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
                exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

    L_d2 = log((exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
                exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
                exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
                exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
                exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
                exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
                exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
                exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)))/
               (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
                exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
                exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
                exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
                exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5))));

L_d3 = log((exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
           (exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-10.5)*(rx-10.5))));

L_d4 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-13.5)*(rx-13.5)))/
           (exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+8.5)*(rx+8.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-7.5)*(rx-7.5))+exp(n*(rx-8.5)*(rx-8.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-11.5)*(rx-11.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-15.5)*(rx-15.5))));


D1_data[i]     = L_d0;
D1_data[i+1]   = L_d1;
D1_data[i+2]   = L_d2;
D1_data[i+3]   = L_d3;
D1_parity[i]   = L_d4;
D1_parity[i+1] = 0.0;
D1_parity[i+2] = 0.0;
D1_parity[i+3] = 0.0;
D1_parity[i+4] = 0.0;
D1_parity[i+6] = 0.0;

/* symbol 1, I dimension */
d0 = data[i+4];
d1 = data[i+5];
d2 = data[i+6];
d3 = Enc1[i+5];
d4 = Enc2[i];
tx  = 2*d2 - 2*d3 + 4*d2*d3 - 1.0 + (((2*d2-1)*(2*d3-1))<0?(d4-0.5):(0.5-d4));
tx = (d1 == 0 ? (tx - 4): (4 - tx));
tx = (d0 == 0 ? (tx - 8): (8 - tx));
rx  = tx + SIGMA_710_1024QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
           (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
```

```
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

L_d1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)))/
           (exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

L_d2 = log((exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)))/
           (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5))));

L_d3 = log((exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
           (exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-10.5)*(rx-10.5))));

L_d4 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-13.5)*(rx-13.5)))/
           (exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+8.5)*(rx+8.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-7.5)*(rx-7.5))+exp(n*(rx-8.5)*(rx-8.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-11.5)*(rx-11.5)) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-15.5)*(rx-15.5))));

D1_data[i+4]   = L_d0;
D1_data[i+5]   = L_d1;
D1_data[i+6]   = L_d2;
D1_parity[i+5] = L_d3;
D2_parity[i]   = L_d4;
D2_parity[i+1] = 0.0;
D2_parity[i+2] = 0.0;
D2_parity[i+3] = 0.0;
D2_parity[i+4] = 0.0;
D2_parity[i+6] = 0.0;

/* symbol 2, Q dimension */
d0  = data[i+7];
d1  = data[i+8];
d2  = data[i+9];
d3  = data[i+10];
d4  = Enc2[i+5];
tx  = 2*d2 - 2*d3 + 4*d2*d3 - 1.0 + (((2*d2-1)*(2*d3-1))<0?(d4-0.5):(0.5-d4));
tx = (d1 == 0 ? (tx - 4): (4 - tx));
tx = (d0 == 0 ? (tx - 8): (8 - tx));
rx  = tx + SIGMA_710_1024QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
           exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
           exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
           exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
           exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
          (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
           exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
           exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
           exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
           exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
           exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
           exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

L_d1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
           exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
           exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
           exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)))/
          (exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
           exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
           exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
           exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)) +
           exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
           exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
           exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
           exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

L_d2 = log((exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
           exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
           exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
           exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
           exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
           exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
           exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
           exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)))/
          (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
           exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
           exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
           exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
           exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
           exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
```

COMPUTER PROGRAM LISTING APPENDIX

```
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5))));

L_d3 = log((exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
           (exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-10.5)*(rx-10.5))));

L_d4 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-13.5)*(rx-13.5)))/
           (exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+8.5)*(rx+8.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-7.5)*(rx-7.5))+exp(n*(rx-8.5)*(rx-8.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-11.5)*(rx-11.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-15.5)*(rx-15.5))));


 D1_data[i+7]   = L_d0;
 D1_data[i+8]   = L_d1;
D1_data[i+9]   = L_d2;
D1_data[i+10]  = L_d3;
D2_parity[i+5] = L_d4;
D2_parity[i+7] = 0.0;
D2_parity[i+8] = 0.0;
D2_parity[i+9] = 0.0;
D2_parity[i+11] = 0.0;
D2_parity[i+12] = 0.0;
D2_parity[i+13] = 0.0;

 /* symbol 2, I dimension */
 d0 = data[i+11];
 d1 = data[i+12];
 d2 = data[i+13];
 d3 = Enc2[i+10];
d4 = Enc1[i+10];
tx  = 2*d2 - 2*d3 + 4*d2*d3 - 1.0 + (((2*d2-1)*(2*d3-1))<0?(d4-0.5):(0.5-d4));
tx = (d1 == 0 ? (tx - 4): (4 - tx));
tx = (d0 == 0 ? (tx - 8): (8 - tx));
 rx  = tx + SIGMA_710_1024QAM * gasdev();
L_d0 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
           (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
```

```
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

L_d1 = log((exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)))/
           (exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5))));

L_d2 = log((exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-11.5)*(rx-11.5)))/
           (exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5))));

L_d3 = log((exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+7.5)*(rx+7.5)) +
            exp(n*(rx+8.5)*(rx+8.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-7.5)*(rx-7.5)) +
            exp(n*(rx-8.5)*(rx-8.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-15.5)*(rx-15.5)))/
           (exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-13.5)*(rx-13.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-10.5)*(rx-10.5))));

L_d4 = log((exp(n*(rx+2.5)*(rx+2.5))+exp(n*(rx+1.5)*(rx+1.5)) +
            exp(n*(rx+6.5)*(rx+6.5))+exp(n*(rx+5.5)*(rx+5.5)) +
            exp(n*(rx+10.5)*(rx+10.5))+exp(n*(rx+9.5)*(rx+9.5)) +
            exp(n*(rx+14.5)*(rx+14.5))+exp(n*(rx+13.5)*(rx+13.5)) +
            exp(n*(rx-2.5)*(rx-2.5))+exp(n*(rx-1.5)*(rx-1.5)) +
            exp(n*(rx-6.5)*(rx-6.5))+exp(n*(rx-5.5)*(rx-5.5)) +
            exp(n*(rx-10.5)*(rx-10.5))+exp(n*(rx-9.5)*(rx-9.5)) +
            exp(n*(rx-14.5)*(rx-14.5))+exp(n*(rx-13.5)*(rx-13.5)))/
           (exp(n*(rx+4.5)*(rx+4.5))+exp(n*(rx+3.5)*(rx+3.5)) +
            exp(n*(rx+7.5)*(rx+7.5))+exp(n*(rx+8.5)*(rx+8.5)) +
            exp(n*(rx+12.5)*(rx+12.5))+exp(n*(rx+11.5)*(rx+11.5)) +
            exp(n*(rx+0.5)*(rx+0.5))+exp(n*(rx+15.5)*(rx+15.5)) +
            exp(n*(rx-4.5)*(rx-4.5))+exp(n*(rx-3.5)*(rx-3.5)) +
            exp(n*(rx-7.5)*(rx-7.5))+exp(n*(rx-8.5)*(rx-8.5)) +
```

```
                exp(n*(rx-12.5)*(rx-12.5))+exp(n*(rx-11.5)*(rx-11.5)) +
                exp(n*(rx-0.5)*(rx-0.5))+exp(n*(rx-15.5)*(rx-15.5))));

        D1_data[i+11]   = L_d0;
        D1_data[i+12]   = L_d1;
        D1_data[i+13]   = L_d2;
       D2_parity[i+10] = L_d3;
       D1_parity[i+10] = L_d4;
       D1_parity[i+7]  = 0.0;
       D1_parity[i+8]  = 0.0;
       D1_parity[i+9]  = 0.0;
       D1_parity[i+11] = 0.0;
       D1_parity[i+12] = 0.0;
       D1_parity[i+13] = 0.0;

       i = i+13;
      }
#endif


    /****************************************/



   /*
    * At this moment we received the whole turbo code block:
    * D1_data[] stores the received information sequence,
    * D1_parity[] stores the received punctured parity P sequence and
    * D2_data[] stores the interleaved received information sequence,
    */

    for(i = 0; i < INT_SIZE; i++)
      D2_data[i] = D1_data[i];
    r_ileav(D2_data, rule);
   /*
    * D2_parity[] stores the received punctured parity Q sequence.
    */

    for(iteration = 1; iteration <= NR_ITER; iteration++)
      {
        /*
         * Start one iteration of the turbo decoder here:
         */
#ifdef R46_64QAM_TTCM_VoCAL
        jat_map2(jat_code1, D1_data, D1_parity, D1_app, D1_exi);
#else
        jat_map1(jat_code1, D1_data, D1_parity, D1_app, D1_exi);
#endif
        /*
         * Interleave the extrinsic information from Decoder1:
         */
        for(k = 0; k < INT_SIZE; k++)
          D2_app[k] = D1_exi[k];
        r_ileav(D2_app, rule);

        /*
         * Decoder2:
         */
#ifdef R46_64QAM_TTCM_VoCAL
        jat_map2(jat_code2, D2_data, D2_parity, D2_app, D2_exi);
#else
        jat_map1(jat_code2, D2_data, D2_parity, D2_app, D2_exi);
#endif

        /*
         * Deinterleave the extrinsic information from Decoder2:
         */
        r_deileav(D2_exi, rule);
        for(k = 0; k < INT_SIZE; k++)
          D1_app[k] = D2_exi[k];
```

COMPUTER PROGRAM LISTING APPENDIX

```
#ifdef R46_64QAM_TTCM_VoCAL
        for(k = 0; k < INT_SIZE/2; k++)
          Dec_data[k] = D1_data[k] + log(D1_exi[k]) + log(D2_exi[k]);

        /*
         * Re-encode with encoder1:
         */
        jat_code1->enc_state = 0;          /* reset encoder1's state    */
        for(k = 0; k < INT_SIZE/2; k++)
          Enc1[k] = jat_enc_bp_fp(jat_code1, ((Dec_data[k] > 0.0)?1:0));

        /*
         * interleave data:
         */
         for(k = 0; k < INT_SIZE/2; k++)
           data_i[k] = ((Dec_data[k] > 0.0)?1:0);
         r_ileava(data_i, rule);

         /*
          * Re-encode with encoder2:
          */
        jat_code2->enc_state = 0;          /* reset encoder2's state    */
         for(k = 0; k < INT_SIZE/2; k++)
            Enc2[k] = jat_enc_bp_fp(jat_code2, data_i[k]);

         /*
          * Find the closest point out of four in the sub-constellation
          */
        for(k = 0; k < INT_SIZE/2 - 1; )
          {
            u4 = ((Dec_data[k] > 0.0)?1:0);
            u3 = ((Dec_data[k+1] > 0.0)?1:0);
            u2 = Enc1[k];
            u1 = Enc2[k+1];

            rx_I = D1_data[k + INT_SIZE/2];
            rx_Q = D1_data[k + INT_SIZE/2 + 1];


            j = 4*u4+8*u3+16*u2+32*u1;
            v00_I = find_tx_I(j);
            v00_Q = find_tx_Q(j);
            v01_I = find_tx_I(j+1);
            v01_Q = find_tx_Q(j+1);
            v10_I = find_tx_I(j+2);
            v10_Q = find_tx_Q(j+2);
            v11_I = find_tx_I(j+3);
            v11_Q = find_tx_Q(j+3);

            Dec_data[k+INT_SIZE/2]  = log(
                    (exp(n*((rx_I-v11_I)*(rx_I-v11_I)+(rx_Q-v11_Q)*(rx_Q-v11_Q)))+
                     exp(n*((rx_I-v10_I)*(rx_I-v10_I)+(rx_Q-v10_Q)*(rx_Q-v10_Q))))/
                    (exp(n*((rx_I-v01_I)*(rx_I-v01_I)+(rx_Q-v01_Q)*(rx_Q-v01_Q)))+
                     exp(n*((rx_I-v00_I)*(rx_I-v00_I)+(rx_Q-v00_Q)*(rx_Q-v00_Q)))));

            Dec_data[k+INT_SIZE/2+1]  = log(
                    (exp(n*((rx_I-v11_I)*(rx_I-v11_I)+(rx_Q-v11_Q)*(rx_Q-v11_Q)))+
                     exp(n*((rx_I-v01_I)*(rx_I-v01_I)+(rx_Q-v01_Q)*(rx_Q-v01_Q))))/
                    (exp(n*((rx_I-v10_I)*(rx_I-v10_I)+(rx_Q-v10_Q)*(rx_Q-v10_Q)))+
                     exp(n*((rx_I-v00_I)*(rx_I-v00_I)+(rx_Q-v00_Q)*(rx_Q-v00_Q)))));

            k = k+2;
          }

#else
        for(k = 0; k < INT_SIZE; k++)
          Dec_data[k] = D1_data[k] + log(D1_exi[k]) + log(D2_exi[k]);
#endif

        /*
```

```
         * print errors:
         */
        k = print_err(data, Dec_data, iteration, block, no_err);
        if(k == 0)
          break;

        /*
         * End one iteration of the turbo decoder here.
         */
      }


  }
free(jat_code1->P0state);
free(jat_code1->P1state);
free(jat_code1->N0state);
free(jat_code1->N1state);
free(jat_code1->Coded0);
free(jat_code1->Coded1);
free(jat_code1);
free(jat_code2->P0state);
free(jat_code2->P1state);
free(jat_code2->N0state);
free(jat_code2->N1state);
free(jat_code2->Coded0);
free(jat_code2->Coded1);
free(jat_code2);
free(rule);
free(data);
free(data_i);
free(data_d);
free(no_err);
free(Enc1);
free(Enc2);
free(D1_data);
free(D1_parity);
free(D1_app);
free(D1_exi);
free(D2_data);
free(D2_parity);
free(D2_app);
free(D2_exi);
free(Dec_data);
free(frame_hist);
free(Zero_data);
for(i = THRESHOLD_ITER; i <= NR_ITER; i++)
        free(bit_hist_array[i]);
free(bit_hist_array);
free(bit_hist_block);
}


/*****************************************************************************/
/* jat_trellis_bp_fp() initializes the code structure                        */
void jat_trellis_bp_fp(jat_code *code_str)
{
  int i;
  for(i = 0; i < code_str->nr_states ; i++)
    {
      code_str->enc_state  = i;
      code_str->P0state[i] = jat_ps(code_str, 0);
      code_str->P1state[i] = jat_ps(code_str, 1);
      code_str->enc_state  = i;
      code_str->Coded0[i]  = jat_enc_bp_fp(code_str, 0);
      code_str->N0state[i] = code_str->enc_state;    /*next state i if d = 0 */
      code_str->enc_state  = i;
      code_str->Coded1[i]  = jat_enc_bp_fp(code_str, 1);
      code_str->N1state[i] = code_str->enc_state;    /*next state i if d = 1 */
```

COMPUTER PROGRAM LISTING APPENDIX

```
/*****************************************************************************/
/* jat_ps() returns the previous state, given the code structure & the input */
/*      bit for the previous state                                           */
/*           input:   current state, input bit for the previous state        */
/*           output:  previous state                                         */
int jat_ps(jat_code *code_st, int inp)
{
  int pr_state, pr_msb, i, j, k, l;

  if(code_st->enc_mem == 1)
    {
      pr_state = code_st->enc_state ^ inp;
    }
  else
    {
      /*find previous state:                                                 */
      pr_msb = (inp & 0x1) ^ (code_st->enc_state & 0x1);
      for(i=1,j=2,k=(1        <<        code_st->enc_mem-1),l=code_st->enc_mem-1;i<code_st-
>enc_mem;i++,l--)
        {
          pr_msb = pr_msb ^ (((code_st->enc_state&j)>>i) & ((code_st->bp&k)>>l));
          j = j << 1;
          k = k >> 1;
        }

      pr_state = ((code_st->enc_state >> 1) & ((1<<(code_st->enc_mem - 1)) - 1)) |
        (pr_msb << (code_st->enc_mem - 1));
    }
  return (pr_state);
}


/*****************************************************************************/
/* jat_enc_bp_fp() -  rate 1/2 systematic feedback convolutional enc.        */
/*          input:  input bit to be encoded                                  */
/*          output: the coded bit                                            */
/* Note:    the lsb of the enc_state will have the new input bit             */
/*          the msb of the enc_state matches the lsb of bp & fp              */
int jat_enc_bp_fp(jat_code *code_st, int data)
{
  int new_lsb, parity, i, j, k, l;

  new_lsb = data;
  if(code_st->enc_mem == 1)
    {
      parity = code_st->enc_state ^ data;
      code_st->enc_state = parity;
    }
  else
    {
      /* xor it with the bits of the enc_state1 for which bp1 is one          */
      for(i = 0, j = 1, k = (1 << code_st->enc_mem-1), l = code_st->enc_mem - 1; i <
code_st->enc_mem; i++, l--)
        {
          new_lsb = new_lsb ^ (((code_st->enc_state&j)>>i) & ((code_st->bp&k)>>l));
          j = j << 1;
          k = k >> 1;
        }

      /* find the parity bit                                                  */
      parity = new_lsb & ((code_st->fp&(1<<code_st->enc_mem)) >> code_st->enc_mem);
      for(i = 0, j = 1, k = (1 << code_st->enc_mem-1), l = code_st->enc_mem - 1; i <
code_st->enc_mem; i++, l--)
        {
          parity = parity ^ (((code_st->enc_state&j)>>i) & ((code_st->fp&k)>>l));
          j = j << 1;
          k = k >> 1;
        }
```

```
      /* update code_st->enc_state                                          */
      code_st->enc_state = ((code_st->enc_state & (code_st->nr_states/2 - 1)) << 1) |
new_lsb;
    }
  return (parity);
}


/*****************************************************************************/
void r_ileav(double *array, int *rule_i)
{
  double *i_wmem;
  int    k;

  i_wmem = (double *)malloc(sizeof(double) * INT_SIZE);
  if(i_wmem == 0)
    printf("\nCouldn't allocate i_wmem memory!");
  for(k = 0; k < INT_SIZE; k++)
    i_wmem[k] = array[k];
  for(k = 0; k < INT_SIZE; k++)
    array[k] = i_wmem[rule_i[2*k+1]];
  free(i_wmem);
}


/*****************************************************************************/
void r_ileava(int *array, int *rule_i)
{
  int   *i_wmem;
  int   k;

  i_wmem = (int *)malloc(sizeof(int) * INT_SIZE);
  if(i_wmem == 0)
    printf("\nCouldn't allocate i_wmem memory!");
  for(k = 0; k < INT_SIZE; k++)
    i_wmem[k] = array[k];
  for(k = 0; k < INT_SIZE; k++)
    array[k] = i_wmem[rule_i[2*k+1]];
  free(i_wmem);
}


/*****************************************************************************/
void r_deileav(double *array, int *rule_d)
{
  double *d_wmem;
  int    k;

  d_wmem = (double *)malloc(sizeof(double) * INT_SIZE);
  if(d_wmem == 0)
    printf("\nCouldn't allocate d_wmem memory!");
  for(k = 0; k < INT_SIZE; k++)
    d_wmem[rule_d[2*k+1]] = array[k];
  for(k = 0; k < INT_SIZE; k++)
    array[k] = d_wmem[k];
  free(d_wmem);
}


/*****************************************************************************/
void r_deileava(int *array, int *rule_d)
{
  int    *d_wmem;
  int    k;

  d_wmem = (int *)malloc(sizeof(int) * INT_SIZE);
  if(d_wmem == 0)
    printf("\nCouldn't allocate d_wmem memory!");
  for(k = 0; k < INT_SIZE; k++)
    d_wmem[rule_d[2*k+1]] = array[k];
  for(k = 0; k < INT_SIZE; k++)
```

COMPUTER PROGRAM LISTING APPENDIX

```
    array[k] = d_wmem[k];
  free(d_wmem);
}


/*****************************************************************************/
/* nrgen() returns a random number between 0 and 1                           */
/*         (uniform distribution generator)                                  */
double nrgen()
{
  long z, k;

  k = s1 / 53668;
  s1 = 40014 * (s1 - k * 53668) - k * 12211;
  if(s1 < 0)
    s1 += 2147483563;
  k = s2 / 52774;
  s2 = 40692 * (s2 - k * 52774) - k * 3791;
  if(s2 < 0)
    s2 += 2147483399;
  z = s1 - s2;
  if(z < 1)
    z += 2147483562;
  return ((double)z / 2147483563);
}


/*****************************************************************************/
/* nrgenbin() returns a 0 or 1 (uniform distribution)                        */
int nrgenbin()
{
  return ((nrgen() > 0.5)?1:0);
}


/*****************************************************************************/
/* gasdev() returns a normally distributed deviate                           */
/*          with zero mean and unit variance                                 */
double gasdev()
{
  static int     iset = 0;
  static double  gset;
  double         fac, r, v1, v2;

  if(iset == 0)
    {
      /* pick two uniform numbers in the square extending from               */
      /* -1 to +1 in each direction, see if they are in the                  */
      /* unit circle, and if they are not, try again.                        */
      do
        {
          v1 = 2.0 * nrgen() - 1.0;
          v2 = 2.0 * nrgen() - 1.0;
          r = v1 * v1 + v2 * v2;
        }
      while (r >= 1.0 || r == 0.0);
      fac = sqrt(-2.0 * log(r)/r);
      /* now make the Box-Muller transformation to get two normal            */
      /* deviates; return one and save the other for next time.              */
      gset = v1 * fac;
      iset = 1;              /* set flag                                     */
      return (v2 * fac);
    }
  else
    {
      /* we have an extra deviate handy, so unset the flag and               */
      /* return it.                                                          */
      iset = 0;
      return (gset);
    }
}
```

COMPUTER PROGRAM LISTING APPENDIX

```
/*****************************************************************************/
/* errors() returns the nr. of positions in which two blocks of data are     */
/*          different; it accepts a shift between the addresses               */
/*          inputs: the address of the first block of integers                */
/*                  the address of the second block of doubles                */
/*                  the size of the block (blocks are equal)                  */
/*          output: the number of positions in which the two blocks are dif. */
int errors(int *block1, double *block2, int size, int iter_nr)
{
  int  i;
  int  count = 0;

  for(i = 0; i < size; i++)
      if(block1[i] != ((block2[i] > 0.0)?1:0))
        {
          count++;
#if defined BIT_HIST
          if(iter_nr>=THRESHOLD_ITER)
            {
              *bit_hist_array[iter_nr] = bit_hist_block[iter_nr];
              bit_hist_array[iter_nr]++;
              *bit_hist_array[iter_nr] = i;
              bit_hist_array[iter_nr]++;
            }
#endif
        }
#if defined BIT_HIST
  if((count>0)&&(iter_nr >= THRESHOLD_ITER))
    bit_hist_block[iter_nr]++;
#endif
  return (count);
}


/*****************************************************************************/
/* print_err() append to the file the nr. of errors and BER                  */
/* returns:  number of bit errors in a block                                 */
int print_err(int *data1, double *data2, int iter_no, int block_no, int *err)
{
  int   i, j, nr;
  int   block_err = 0;
  char fname[] = BIT_HIST_FILE_NAME;
  char sss[] = {'0','1','2','3','4','5','6','7','8','9'};
  FILE  *out_file = NULL;
  int   *pi;

  if((iter_no == 1) && (block_no == 1))
    {
      out_file = fopen(ERROR_FILE_NAME, "a");
      if(!out_file)
        {
          printf("Error2: the output file could not be opened!\n");
          exit (1);
        }
      fprintf(out_file,  "ref_tc.c,  RSC1_enc_mem  =  %d,  RSC1_fp  =  %d,  RSC1_bp  =  %d,
RSC2_enc_mem = %d, RSC2_fp = %d, RSC2_bp = %d, s1 = %d, s2 = %d, int_size = %d, Limit soft
outputs to = %e, Eb/No = %fdB\n", RSC1_ENC_MEM, RSC1_FP, RSC1_BP, RSC2_ENC_MEM, RSC2_FP,
RSC2_BP, SEED1, SEED2, INT_SIZE, (double) MAX, (double) EBNO);
      fclose(out_file);
    }
  block_err = errors(data1, data2, INT_SIZE, iter_no);
  err[iter_no - 1] += block_err;

  ++frame_hist[(iter_no-1)*(INT_SIZE+1) + block_err];

  if((iter_no == NR_ITER) && (block_err != 0))
    ++frame_err;
```

COMPUTER PROGRAM LISTING APPENDIX

```
#if defined BIT_HIST
  if((block_err != 0) && (iter_no >= THRESHOLD_ITER))
    {
      fname[strlen(fname)-1]= sss[iter_no];
      out_file = fopen(fname, "a");
      if(!out_file)
        {
          printf("Error: the bit_hist file could not be opened!\n");
          exit (1);
        }
      for(pi = bit_hist_array[NR_ITER +iter_no]; pi < bit_hist_array[iter_no]; pi = pi+2)
        fprintf(out_file, "\n%06d %06d", *pi, *(pi+1));
      fclose(out_file);
      bit_hist_array[iter_no] = bit_hist_array[NR_ITER +iter_no];
    }
#endif

  if(((iter_no == NR_ITER) && (((block_no % PRINT_BLOCKS) == 0) || (err[NR_ITER - 1] >
MAX_ERRORS))) || (((block_no % PRINT_BLOCKS) == 0) && (block_err == 0)))


      {
        out_file = fopen(ERROR_FILE_NAME, "a");
        if(!out_file)
          {
            printf("Error3: the output file could not be opened!\n");
            exit (1);
          }
        nr = block_no * INT_SIZE;
        fprintf(out_file, "\n\nNr. of info bits:  %d (%d blocks)",
                nr, block_no);
        for(j = 0; j < NR_ITER; j++)
          fprintf(out_file, "\nIter: %02d, Errors: %06d, BER = %e",
                  j + 1, err[j],(double)err[j]/INT_SIZE/block_no);
        total_err = err[NR_ITER - 1];
        fprintf(out_file, "\nFrame error = %f(%f errors per block)\n",
                (double)frame_err/block_no,
                (frame_err == 0?0.0:(double)err[NR_ITER-1]/frame_err));
        fclose(out_file);

        if((block_no % 100000) == 0)
          {
            out_file = fopen(FRAME_HIST_FILE_NAME, "a");
            if(!out_file)
              {
                printf("Error4: the .fhist file could not be opened!\n");
                exit (1);
              }
            nr = block_no * INT_SIZE;
            fprintf(out_file, "\n\nNr. of info bits:  %d (%d blocks)",
                    nr, block_no);
            for(j = 0; j <= INT_SIZE; j++)
              fprintf(out_file, "\n%03d %03d",
                      j, frame_hist[(NR_ITER-1)*(INT_SIZE+1)+j]);
            fclose(out_file);
          }
      }
return (block_err);
}


/*****************************************************************************/
/* This is a MAP decoder for a cs->nr_states states jat_code.                */
/*   function:  decodes a block of received data of length INT_SIZE.         */
/*              It assumes that the encoder state starts from state zero     */
/*   input:     code structure, I address, Q address, L_in address           */
/*   output:    the extrinsic information in L_out                           */
/*   globals:   noise                                                        */
```

COMPUTER PROGRAM LISTING APPENDIX

```
/*
 ****************************************************************************
 * As jat_map but outputs probability and not log(probability)
 * It also can handle very large interleavers
 */
void jat_mapl(jat_code *cs, double *I, double *Q, double *L_in, double *L_out)
{
  double    sum, sum_0, sum_1, max;
  int       i, j, k, st;
  double    *alpha_old;
  double    *alpha_new;
  double    *beta0;
  double    *beta1;
  double    *probI;
  double    *probQ;

alpha_old = (double *)malloc(sizeof(double) * 2 * cs->nr_states);
alpha_new = (double *)malloc(sizeof(double) * 2 * cs->nr_states);

beta0 = (double *)malloc(sizeof(double) * INT_SIZE * cs->nr_states);
if(beta0 == 0)
  {
    printf("Couldn't allocate beta0 memory!\n");
    exit(1);
  }

beta1 = (double *)malloc(sizeof(double) * INT_SIZE * cs->nr_states);
if(beta1 == 0)
  {
    printf("Couldn't allocate beta1 memory!\n");
    exit(1);
  }

probI = (double *)malloc(sizeof(double) * INT_SIZE);
if(probI == 0)
  {
    printf("Couldn't allocate probI memory!\n");
    exit(1);
  }

probQ = (double *)malloc(sizeof(double) * INT_SIZE);
if(probQ == 0)
  {
    printf("Couldn't allocate probQ memory!\n");
    exit(1);
  }


  /* initialize the alpha_old metrics                                    */
  for(st = 0; st < cs->nr_states; st++)
    for(k = 0; k < 2; k++)
      *(alpha_old + k * cs->nr_states + st) = 0.0;

  *(alpha_old + cs->P0state[0]) = 1.0;
  *(alpha_old + cs->nr_states + cs->P1state[0]) = 1.0;

  /* initialize beta's                                                   */
  for(st = 0; st < cs->nr_states; st++)
    {
      beta0[(INT_SIZE - 1) * cs->nr_states + st] = 1.0;
      beta1[(INT_SIZE - 1) * cs->nr_states + st] = 1.0;
    }

  /* compute all beta's                                                  */
  for(i = INT_SIZE - 2; i >= 0; i--)
      {
        probI[i + 1] = exp(I[i + 1]) * L_in[i + 1];
        probQ[i + 1] = exp(Q[i + 1]);
        for(st = 0; st < cs->nr_states; st++)
          {
            /* compute beta0[i][st]:                                     */
```

COMPUTER PROGRAM LISTING APPENDIX

```
            beta0[i * cs->nr_states + st] = beta0[(i + 1) * cs->nr_states + cs-
>N0state[st]]*
                        ((cs->Coded0[cs->N0state[st]] == 0)?1:probQ[i + 1])+
                        beta1[(i + 1) * cs->nr_states + cs->N0state[st]]*probI[i + 1]*
                        ((cs->Coded1[cs->N0state[st]] == 0)?1:probQ[i + 1]);
            beta1[i * cs->nr_states + st] = beta0[(i + 1) * cs->nr_states + cs-
>N1state[st]]*
                        ((cs->Coded0[cs->N1state[st]] == 0)?1:probQ[i + 1])+
                        beta1[(i + 1) * cs->nr_states + cs->N1state[st]]*probI[i + 1]*
                        ((cs->Coded1[cs->N1state[st]] == 0)?1:probQ[i + 1]);
        }

      max = beta0[i * cs->nr_states];
      for(st = 1; st < cs->nr_states; st++)
        if(beta0[i * cs->nr_states + st] > max)
          max = beta0[i * cs->nr_states + st];
      for(st = 0; st < cs->nr_states; st++)
        if(beta1[i * cs->nr_states + st] > max)
          max = beta1[i * cs->nr_states + st];
      for(st = 0; st < cs->nr_states; st++)
        {
          beta0[i * cs->nr_states + st] = beta0[i * cs->nr_states + st] / max;
          beta1[i * cs->nr_states + st] = beta1[i * cs->nr_states + st] / max;
        }
    }

  /* now we have all beta's; we can compute alpha for all states for each    */
  /* data bit and using beta's we compute lambda                              */
  probI[0] = exp(I[0]) * L_in[0];
  probQ[0] = exp(Q[0]);
  for(k = 0; k < INT_SIZE; k++)
    {
      for(st = 0; st < cs->nr_states; st++)
        {
        sum = *(alpha_old + cs->P0state[st]) + *(alpha_old + cs->nr_states + cs-
>P1state[st]);
        *(alpha_new + st) = sum * ((cs->Coded0[st] == 0)?1:probQ[k]);
        *(alpha_new + cs->nr_states + st) = sum * probI[k] * ((cs->Coded1[st] ==
0)?1:probQ[k]);
        }

      /* find the max value and renormalize alpha's:                          */
      max = *alpha_new;
      for(st = 0; st < cs->nr_states; st++)
        for(j = 0; j < 2; j++)
          if(*(alpha_new + cs->nr_states * j + st) > max)
                max = *(alpha_new + cs->nr_states * j + st);
      for(st = 0; st < cs->nr_states; st++)
        for(j = 0; j < 2; j++)
          *(alpha_new + cs->nr_states * j + st) = *(alpha_new + cs->nr_states * j + st)/
max;

      /* find sum_0 and sum_1 over all states for L_out:                      */
      sum_0 = 0.0;
      sum_1 = 0.0;
      for(st = 0; st < cs->nr_states; st++)
        {
          sum_0 += *(alpha_new + st) * beta0[k * cs->nr_states + st];
          sum_1 += *(alpha_new + cs->nr_states + st) * beta1[k * cs->nr_states + st];
        }

      /* output the extrinsic information:                                    */
      L_out[k] = (sum_1 / sum_0) / exp(I[k]) / L_in[k];
      if(L_out[k] > MAX)
        L_out[k] = MAX;
      if(L_out[k] < 1/MAX)
        L_out[k] = 1/MAX;

      for(st = 0; st < cs->nr_states; st++)
        for(j = 0; j < 2; j++)/* update alphas            */
          *(alpha_old + cs->nr_states * j + st)=*(alpha_new + cs->nr_states * j + st);
```

COMPUTER PROGRAM LISTING APPENDIX

```
    }
free(beta0);
free(beta1);
free(probI);
free(probQ);
free(alpha_old);
free(alpha_new);
}



double find_tx_I(int k)
{
  double tx_I;
        switch(k)
          {
          case 0:
            tx_I = 0.5;
            break;
          case 1:
            tx_I = -3.5;
            break;
          case 2:
            tx_I = 0.5;
            break;
          case 3:
            tx_I = -3.5;
            break;
          case 4:
            tx_I = 2.5;
            break;
          case 5:
            tx_I = -1.5;
            break;
          case 6:
            tx_I = 2.5;
            break;
          case 7:
            tx_I = -1.5;
            break;
          case 8:
            tx_I = 2.5;
            break;
          case 9:
            tx_I = -1.5;
            break;
          case 10:
            tx_I = 2.5;
            break;
          case 11:
            tx_I = -1.5;
            break;
          case 12:
            tx_I = 0.5;
            break;
          case 13:
            tx_I = -3.5;
            break;
          case 14:
            tx_I = 0.5;
            break;
          case 15:
            tx_I = -3.5;
            break;
          case 16:
            tx_I = 1.5;
            break;
          case 17:
            tx_I = -2.5;
            break;
```

COMPUTER PROGRAM LISTING APPENDIX

```
case 18:
  tx_I = 1.5;
  break;
case 19:
  tx_I = -2.5;
  break;
case 20:
  tx_I = 3.5;
  break;
case 21:
  tx_I = -0.5;
  break;
case 22:
  tx_I = 3.5;
  break;
case 23:
  tx_I = -0.5;
  break;
case 24:
  tx_I = 3.5;
  break;
case 25:
  tx_I = -0.5;
  break;
case 26:
  tx_I = 3.5;
  break;
case 27:
  tx_I = -0.5;
  break;
case 28:
  tx_I = 1.5;
  break;
case 29:
  tx_I = -2.5;
  break;
case 30:
  tx_I = 1.5;
  break;
case 31:
  tx_I = -2.5;
  break;
case 32:
  tx_I = 1.5;
  break;
case 33:
  tx_I = -2.5;
  break;
case 34:
  tx_I = 1.5;
  break;
case 35:
  tx_I = -2.5;
  break;
case 36:
  tx_I = 3.5;
  break;
case 37:
  tx_I = -0.5;
  break;
case 38:
  tx_I = 3.5;
  break;
case 39:
  tx_I = -0.5;
  break;
case 40:
  tx_I = 3.5;
  break;
case 41:
  tx_I = -0.5;
```

COMPUTER PROGRAM LISTING APPENDIX

```
    break;
  case 42:
    tx_I = 3.5;
    break;
  case 43:
    tx_I = -0.5;
    break;
  case 44:
    tx_I = 1.5;
    break;
  case 45:
    tx_I = -2.5;
    break;
  case 46:
    tx_I = 1.5;
    break;
  case 47:
    tx_I = -2.5;
    break;
  case 48:
    tx_I = 0.5;
    break;
  case 49:
    tx_I = -3.5;
    break;
  case 50:
    tx_I = 0.5;
    break;
  case 51:
    tx_I = -3.5;
    break;
  case 52:
    tx_I = 2.5;
    break;
  case 53:
    tx_I = -1.5;
    break;
  case 54:
    tx_I = 2.5;
    break;
  case 55:
    tx_I = -1.5;
    break;
  case 56:
    tx_I = 2.5;
    break;
  case 57:
    tx_I = -1.5;
    break;
  case 58:
    tx_I = 2.5;
    break;
  case 59:
    tx_I = -1.5;
    break;
  case 60:
    tx_I = 0.5;
    break;
  case 61:
    tx_I = -3.5;
    break;
  case 62:
    tx_I = 0.5;
    break;
  case 63:
    tx_I = -3.5;
    break;
  }
return(tx_I);
```

COMPUTER PROGRAM LISTING APPENDIX

```
double find_tx_Q(int k)
{
  double tx_Q;
        switch(k)
          {
          case 0:
            tx_Q = 2.5;
            break;
          case 1:
            tx_Q = 2.5;
            break;
          case 2:
            tx_Q = -1.5;
            break;
          case 3:
            tx_Q = -1.5;
            break;
          case 4:
            tx_Q = 0.5;
            break;
          case 5:
            tx_Q = 0.5;
            break;
          case 6:
            tx_Q = -3.5;
            break;
          case 7:
            tx_Q = -3.5;
            break;
          case 8:
            tx_Q = 2.5;
            break;
          case 9:
            tx_Q = 2.5;
            break;
          case 10:
            tx_Q = -1.5;
            break;
          case 11:
            tx_Q = -1.5;
            break;
          case 12:
            tx_Q = 0.5;
            break;
          case 13:
            tx_Q = 0.5;
            break;
          case 14:
            tx_Q = -3.5;
            break;
          case 15:
            tx_Q = -3.5;
            break;
          case 16:
            tx_Q = 3.5;
            break;
          case 17:
            tx_Q = 3.5;
            break;
          case 18:
            tx_Q = -0.5;
            break;
          case 19:
            tx_Q = -0.5;
            break;
          case 20:
            tx_Q = 1.5;
            break;
          case 21:
            tx_Q = 1.5;
```

COMPUTER PROGRAM LISTING APPENDIX

```
  break;
case 22:
  tx_Q = -2.5;
  break;
case 23:
  tx_Q = -2.5;
  break;
case 24:
  tx_Q = 3.5;
  break;
case 25:
  tx_Q = 3.5;
  break;
case 26:
  tx_Q = -0.5;
  break;
case 27:
  tx_Q = -0.5;
  break;
case 28:
  tx_Q = 1.5;
  break;
case 29:
  tx_Q = 1.5;
  break;
case 30:
  tx_Q = -2.5;
  break;
case 31:
  tx_Q = -2.5;
  break;
case 32:
  tx_Q = 2.5;
  break;
case 33:
  tx_Q = 2.5;
  break;
case 34:
  tx_Q = -1.5;
  break;
case 35:
  tx_Q = -1.5;
  break;
case 36:
  tx_Q = 0.5;
  break;
case 37:
  tx_Q = 0.5;
  break;
case 38:
  tx_Q = -3.5;
  break;
case 39:
  tx_Q = -3.5;
  break;
case 40:
  tx_Q = 2.5;
  break;
case 41:
  tx_Q = 2.5;
  break;
case 42:
  tx_Q = -1.5;
  break;
case 43:
  tx_Q = -1.5;
  break;
case 44:
  tx_Q = 0.5;
  break;
case 45:
```

```
            tx_Q = 0.5;
            break;
          case 46:
            tx_Q = -3.5;
            break;
          case 47:
            tx_Q = -3.5;
            break;
          case 48:
            tx_Q = 3.5;
            break;
          case 49:
            tx_Q = 3.5;
            break;
          case 50:
            tx_Q = -0.5;
            break;
          case 51:
            tx_Q = -0.5;
            break;
          case 52:
            tx_Q = 1.5;
            break;
          case 53:
            tx_Q = 1.5;
            break;
          case 54:
            tx_Q = -2.5;
            break;
          case 55:
            tx_Q = -2.5;
            break;
          case 56:
            tx_Q = 3.5;
            break;
          case 57:
            tx_Q = 3.5;
            break;
          case 58:
            tx_Q = -0.5;
            break;
          case 59:
            tx_Q = -0.5;
            break;
          case 60:
            tx_Q = 1.5;
            break;
          case 61:
            tx_Q = 1.5;
            break;
          case 62:
            tx_Q = -2.5;
            break;
          case 63:
            tx_Q = -2.5;
            break;
          }
        return(tx_Q);
}


/*****************************************************************************/
/* This is a MAP decoder for a cs->nr_states states jat_code.                */
/*   function:  decodes a block of received data of length INT_SIZE/2.        */
/*              It assumes that the encoder state starts from state zero     */
/*   input:     code structure, I address, Q address, L_in address           */
/*   output:    the extrinsic information in L_out                           */
/*   globals:   noise                                                        */


/*
```

COMPUTER PROGRAM LISTING APPENDIX

```
 ****************************************************************************
 * As jat_map but outputs probability and not log(probability)
 * It also can handle very large interleavers
 */
void jat_map2(jat_code *cs, double *I, double *Q, double *L_in, double *L_out)
{
  double    sum, sum_0, sum_1, max;
  int       i, j, k, st;
  double    *alpha_old;
  double    *alpha_new;
  double    *beta0;
  double    *beta1;
  double    *probI;
  double    *probQ;

alpha_old = (double *)malloc(sizeof(double) * 2 * cs->nr_states);
alpha_new = (double *)malloc(sizeof(double) * 2 * cs->nr_states);

beta0 = (double *)malloc(sizeof(double) * INT_SIZE/2 * cs->nr_states);
if(beta0 == 0)
  {
    printf("Couldn't allocate beta0 memory!\n");
    exit(1);
  }

beta1 = (double *)malloc(sizeof(double) * INT_SIZE/2 * cs->nr_states);
if(beta1 == 0)
  {
    printf("Couldn't allocate beta1 memory!\n");
    exit(1);
  }

probI = (double *)malloc(sizeof(double) * INT_SIZE/2);
if(probI == 0)
  {
    printf("Couldn't allocate probI memory!\n");
    exit(1);
  }

probQ = (double *)malloc(sizeof(double) * INT_SIZE/2);
if(probQ == 0)
  {
    printf("Couldn't allocate probQ memory!\n");
    exit(1);
  }


  /* initialize the alpha_old metrics                                        */
  for(st = 0; st < cs->nr_states; st++)
    for(k = 0; k < 2; k++)
      *(alpha_old + k * cs->nr_states + st) = 0.0;

  *(alpha_old + cs->P0state[0]) = 1.0;
  *(alpha_old + cs->nr_states + cs->P1state[0]) = 1.0;

  /* initialize beta's                                                       */
  for(st = 0; st < cs->nr_states; st++)
    {
      beta0[(INT_SIZE/2 - 1) * cs->nr_states + st] = 1.0;
      beta1[(INT_SIZE/2 - 1) * cs->nr_states + st] = 1.0;
    }

  /* compute all beta's                                                      */
  for(i = INT_SIZE/2 - 2; i >= 0; i--)
      {
        probI[i + 1] = exp(I[i + 1]) * L_in[i + 1];
        probQ[i + 1] = exp(Q[i + 1]);
        for(st = 0; st < cs->nr_states; st++)
          {
            /* compute beta0[i][st]:                                         */
```

COMPUTER PROGRAM LISTING APPENDIX

```
            beta0[i * cs->nr_states + st] = beta0[(i + 1) * cs->nr_states + cs-
>N0state[st]]*
                        ((cs->Coded0[cs->N0state[st]] == 0)?1:probQ[i + 1])+
                        beta1[(i + 1) * cs->nr_states + cs->N0state[st]]*probI[i + 1]*
                        ((cs->Coded1[cs->N0state[st]] == 0)?1:probQ[i + 1]);
            beta1[i * cs->nr_states + st] = beta0[(i + 1) * cs->nr_states + cs-
>N1state[st]]*
                        ((cs->Coded0[cs->N1state[st]] == 0)?1:probQ[i + 1])+
                        beta1[(i + 1) * cs->nr_states + cs->N1state[st]]*probI[i + 1]*
                        ((cs->Coded1[cs->N1state[st]] == 0)?1:probQ[i + 1]);
        }

      max = beta0[i * cs->nr_states];
      for(st = 1; st < cs->nr_states; st++)
        if(beta0[i * cs->nr_states + st] > max)
          max = beta0[i * cs->nr_states + st];
      for(st = 0; st < cs->nr_states; st++)
        if(beta1[i * cs->nr_states + st] > max)
          max = beta1[i * cs->nr_states + st];
      for(st = 0; st < cs->nr_states; st++)
        {
          beta0[i * cs->nr_states + st] = beta0[i * cs->nr_states + st] / max;
          beta1[i * cs->nr_states + st] = beta1[i * cs->nr_states + st] / max;
        }
    }

  /* now we have all beta's; we can compute alpha for all states for each    */
  /* data bit and using beta's we compute lambda                              */
  probI[0] = exp(I[0]) * L_in[0];
  probQ[0] = exp(Q[0]);
  for(k = 0; k < INT_SIZE/2; k++)
    {
      for(st = 0; st < cs->nr_states; st++)
        {
        sum = *(alpha_old + cs->P0state[st]) + *(alpha_old + cs->nr_states + cs-
>P1state[st]);
        *(alpha_new + st) = sum * ((cs->Coded0[st] == 0)?1:probQ[k]);
        *(alpha_new + cs->nr_states + st) = sum * probI[k] * ((cs->Coded1[st] ==
0)?1:probQ[k]);
        }

      /* find the max value and renormalize alpha's:                          */
      max = *alpha_new;
      for(st = 0; st < cs->nr_states; st++)
        for(j = 0; j < 2; j++)
          if(*(alpha_new + cs->nr_states * j + st) > max)
                max = *(alpha_new + cs->nr_states * j + st);
      for(st = 0; st < cs->nr_states; st++)
        for(j = 0; j < 2; j++)
          *(alpha_new + cs->nr_states * j + st) = *(alpha_new + cs->nr_states * j + st)/
max;

      /* find sum_0 and sum_1 over all states for L_out:                      */
      sum_0 = 0.0;
      sum_1 = 0.0;
      for(st = 0; st < cs->nr_states; st++)
        {
          sum_0 += *(alpha_new + st) * beta0[k * cs->nr_states + st];
          sum_1 += *(alpha_new + cs->nr_states + st) * beta1[k * cs->nr_states + st];
        }

      /* output the extrinsic information:                                    */
      L_out[k] = (sum_1 / sum_0) / exp(I[k]) / L_in[k];
      if(L_out[k] > MAX)
        L_out[k] = MAX;
      if(L_out[k] < 1/MAX)
        L_out[k] = 1/MAX;

      for(st = 0; st < cs->nr_states; st++)
        for(j = 0; j < 2; j++)/* update alphas                    */
          *(alpha_old + cs->nr_states * j + st)=*(alpha_new + cs->nr_states * j + st);
```

```
    }
free(beta0);
free(beta1);
free(probI);
free(probQ);
free(alpha_old);
free(alpha_new);
}
```

COMPUTER PROGRAM LISTING APPENDIX

interlever.c

```
#define MAX_CINDEX 46
#define MAX_RINDEX 47
#define MAX_ELEMENT 2100
#include <stdio.h>
#include <stdlib.h>

void main (void)
{
        int ra, ca; //Ia sequence row and column indices
        int count; //Counter for each bit in DMT frame
        int element; //Element number used for finding if element within array
        FILE *output;


        output=fopen("interleaver","w");
        //Initial sequence indices

        ra=MAX_RINDEX-1;
        ca=0;

        //Adjust the initial indices for Ia if beyond ending element

        element=ra*MAX_CINDEX+ca;

        while (element >=MAX_ELEMENT) {
                ra--;
                ca++;
                if (ra<0) {
                        ra=MAX_RINDEX-1;
                        ca=ca+(MAX_RINDEX-1);
                }
                ca=ca%MAX_CINDEX;

                element= ra*MAX_CINDEX+ra;
        }

        //Fetch all elements in sequence Ia

        for (count = 0; count<MAX_ELEMENT; count++) {
                //Fetch array[ra][ca]
                element=ra*MAX_CINDEX+ca;
                fprintf(output,"%d %d\n",count,element);
                //Update indices for next access

                do {
                        ra--;
                        ca++;
                        if (ra<0) {
                                ra=MAX_RINDEX-1;
                                ca=ca+(MAX_RINDEX-1);
                        }
                        ca=ca%MAX_CINDEX;

                        element = ra * MAX_CINDEX+ca;
                } while (element >= MAX_ELEMENT);
        }
}
```

COMPUTER PROGRAM LISTING APPENDIX

S-type interleaver generatior

```
program int(input,output);
{This program generates mod-k S-random and symmetric mod-k S-random interleavers.

const Nmax = 65536; {maximum interleaver size}

var G,H,I,J,K,L,M,N,S,count,temp,prt,i_,j_,k_,im,jm:longint;
    inta,hat,deint:array[0..Nmax] of longint;
    pass,good:boolean;
    s1,s2:longint; {seeds for function uniform}
    into,deinto:text;
    sym:char;

function max(x,y:longint):longint;
{Finds the maximum of x and y}
begin{max}
  if x > y
    then max := x
    else max := y;
end;{max}

function min(x,y:longint):longint;
{Finds the maximum of x and y}
begin{min}
  if x < y
    then min := x
    else min := y;
end;{min}

function uniform(var s1,s2:longint):double;
{Generates a random number from 0.0 < x < 1.0}
const m0 = 2147483562;
      m1 = 2147483563;
      m2 = 2147483399;
      a1 = 40014;
      a2 = 40692;
      q1 = 53668;
      q2 = 52774;
      r1 = 12211;
      r2 =  3791;
var k:longint;
begin{uniform}
  k := s1 div q1;
  s1 := a1*(s1-k*q1) - k*r1;
  if s1 < 0 then s1 := s1+m1;

  k := s2 div q2;
  s2 := a2*(s2-k*q2) - k*r2;
  if s2 < 0 then s2 := s2+m2;

  k := s1-s2;
  if k < 1 then k := k+m0;
  uniform := k/m1;
end;{uniform}

procedure srandom;
{Generates mod-k S-random interleaver}
label 98;

procedure reject;
{reject random number}
begin{reject}
  count := count-1;
  if count = 0
    then begin{bad int}
           good := false;
           goto 98;
         end;{bad int}
```

```
  pass := false;
  for M := K to count-1 do
    hat[M] := hat[M+1];
  hat[count] := J;
end;{reject}

begin{S-random}
  repeat
    writeln('seed1 = ',s1:1,', seed2 = ',s2:1);
    good := true;
    for I := 0 to N-1 do
      hat[I] := I;
    for I := 0 to N-1 do
      begin{make int}
        count := N-I;
        i_ := I mod k_;
        im := min(i_,k_-i_);
        repeat
          pass := true;
          K := trunc(count*uniform(s1,s2));
          if K = count then K := K-1;
          J := hat[K];
          if k_ > 1 then
            begin{mod k test}
              j_ := J mod k_;
              jm := min(j_,k_-j_);
              if im <> jm then reject;
            end;{mod k test}
          if pass = true then
            begin{S-random test}
              for L := max(0,I-S) to I-1 do
                if (abs(J-inta[L]) <= S) and (pass = true) then reject;
            end{S-random test}
        until pass = true;
        for M := K to N-I-2 do
          hat[M] := hat[M+1];
        inta[I] := J;
      end;{make int}
    98:
  until good = true;
end{S-random};

procedure trandom;
{Generates symmetric mod-k S-random interleaver}
label 99;

procedure rejectS;
{reject random number}
begin{reject S}
  count := count-1;
  if count = 0 then
    begin{bad int}
      good := false;
      goto 99;
    end;{bad int}
  pass := false;
  inta[I] := -1;
  inta[J] := -1;
  for M := K to count-1 do
    hat[M] := hat[M+1];
  hat[count] := J;
end;{reject S}

procedure test;
{S-random test}
begin{test}
  if (inta[L] >= 0) and (abs(G-inta[L]) <= S) then rejectS;
  L := L+1;
end;{test}

begin{T-random}
```

COMPUTER PROGRAM LISTING APPENDIX

```
    repeat
      writeln('seed1 = ',s1:1,', seed2 = ',s2:1);
      good := true;
      for I := 0 to N-1 do
        begin{init}
          hat[I] := I;
          inta[I] := -1;
        end;{init}
      H := N;
      I := 0;
      repeat
        count := H;
        while (inta[I] >= 0) and (I < N) do I := I+1;
        i_ := I mod k_;
        im := min(i_,k_-i_);
        repeat
          pass := true;
          K := trunc(count*uniform(s1,s2));
          if K = count then K := K-1;
          J := hat[K];
          if k_ > 1 then
            begin{mod k test}
              j_ := J mod k_;
              jm := min(j_,k_-j_);
              if im <> jm then rejectS;
            end;{mod k test}
          if pass = true then
            begin{S-random test}
              inta[I] := J;
              inta[J] := I;
              G := J;
              L := max(0,I-S);
              while (pass = true) and (L < I) do test;
              L := I+1;
              while (pass = true) and (L < min(I+S,N)) do test;
              G := I;
              L := max(0,J-S);
              while (pass = true) and (L < J) do test;
              L := J+1;
              while (pass = true) and (L < min(J+S,N)) do test;
            end;{S-random test}
        until pass = true;

        H := H-1;
        for M := K to H-1 do
          hat[M] := hat[M+1];
        if I <> J then
          begin{sym}
            K := 0;
            while (hat[K] <> I) and (K < H) do K := K+1;
            H := H-1;
            for M := K to H-1 do
              hat[M] := hat[M+1];
          end;{sym}
      until H = 0;
      99:
    until good = true;
  end{T-random};

  begin{int}
    s1 := 12345; {initialise seeds for uniform}
    s2 := 67890;

    writeln;
    writeln('Random Interleaver Generator V1.01');
    writeln('Copyright (c) 1998 Small World Communications. All rights reserved.');
    writeln;
    write('Enter block size N <= ',Nmax:1,': ');
    readln(N);
    write('Enter S parameter (S=1 is random): ');
    readln(S);
```

COMPUTER PROGRAM LISTING APPENDIX

```
  write('Enter mod-k parameter (k=1 is normal): ');
  readln(k_);
  repeat
    write('Do you want a symmetric interleaver? ');
    readln(sym);
    pass := (sym = 'y') or (sym = 'Y') or (sym = 'n') or (sym = 'N');
    if pass = false then
      writeln('Invalid entry. Try again.');
  until pass = true;
  writeln;
  case sym of
    'y','Y': trandom;
    'n','N': srandom;
  end;{case}

  assign(into,'int.dat');
  rewrite(into);
  for I := 0 to N-1 do
    writeln(into,inta[I]:1);
  close(into);

  for I := 0 to N-1 do
    hat[I] := 0;
  for I := 0 to N-1 do
    begin{test}
      J := inta[I];
      hat[J] := hat[J] + 1;
    end;{test}
  pass := true;
  for I := 0 to N-1 do
    if hat[I] < 1 then pass := false;
  if pass = false
    then writeln('Bad interleaver!');

  pass := true;
  I := 0;
  repeat
    J := inta[I];
    for L := max(0,I-S) to I-1 do
      if (abs(J-inta[L]) <= S) and (pass = true)
        then begin
               pass := false;
               writeln('Interleaver failed S-test');
               writeln(I:1,' ',inta[I]:1,' ',L:1,' ',inta[L]:1);
             end;
    I := I+1;
  until (pass = false) or (I = N);

  K := 0;
  for I := 0 to N-1 do
    K := max(K,abs(I-inta[I]));
  writeln('Dmin = ',K:1);
  writeln('Interleaver table int.dat succussfully generated');
end.{int}
```

1. A method of producing a modulated signal with forward error correction from an information bit stream in a QAM transmitter comprising:

producing first and second parity bit streams corresponding to an information bit stream using first and second concatenated coders interconnected by an interleaver;

selecting respective subsets of said first and second parity bit streams in accordance with a puncturing pattern;

combining the selected subsets of said first and second parity bit streams with said information bit stream;

producing a QAM symbol stream by mapping a first subset of the combined bit streams to an I dimension and mapping a second subset of the combined bit streams to a Q dimension;

modulating said QAM symbol stream to produce a modulated signal; and, transmitting said modulated signal over a communications link.

2. The method recited in claim 1, wherein said puncturing pattern selects respective subsets of parity bits from said first and second parity bit stream such that the number of parity bits represented in each QAM symbol of said QAM symbol stream is less than the number of information bits represented by said each symbol.

3. The method recited in claim 1, wherein each of said mapping to an I dimension and said mapping to a Q dimension comprises:

placing one or more parity bits in least significant positions of said dimension; and, placing information bits in more significant positions of said dimension.

4. The method recited in claim 1, wherein each of said mapping to an I dimension and said mapping to a Q dimension comprises:

placing an information bit in a least significant position of said dimension;

placing one or more parity bits in a more significant position of said dimension; and,

placing additional information bits in remaining positions of said dimension.

5. The method recited in claim 1, where each of said mapping to an I dimension and said mapping to a Q dimension comprises:

placing one or more parity bits in a most significant position of said dimension; and,

placing additional information bits in less significant positions of said dimension.

6. A method of producing a recovered information bit stream from a noisy modulated signal with forward error correction in a QAM receiver comprising:

receiving a modulated signal representing a QAM symbol stream from a communications link, where said received signal includes errors;

demodulating said received signal;

producing a decoded bit stream by iteratively decoding said demodulated signal wherein said demodulated signal is decoded independently in the I dimension and the Q dimension using a puncturing pattern; and,

regenerating said information bit stream from said decoded bit stream.

7. The method recited in claim 6, wherein said puncturing pattern selects respective subsets of parity bits from first and second parity bit streams such that the number of parity bits represented in each QAM symbol of said QAM symbol stream is less than the number of information bits represented by said each symbol.

8. A method in a QAM receiver for adapting to performance conditions in a communication system, comprising:

receiving a QAM symbol stream from a transmitter;

determining a performance metric of the communication system;

determining, based on the performance metric, a puncturing pattern for use in the transmitter to improve performance of the communication system; and

communicating the puncturing pattern to the transmitter.

9. The method recited in claim 8, wherein said puncturing pattern selects respective subsets of parity bits from first and second parity bit streams such that the number of parity bits represented in each QAM symbol of said QAM symbol stream is less than the number of information bits represented by said each symbol.

* * * * *