The Consultative Committee for Space Data Systems

Research and Development for
Space Data System Standards

LOW DENSITY PARITY CHECK CODES FOR USE IN NEAR-EARTH AND DEEP SPACE APPLICATIONS

EXPERIMENTAL SPECIFICATION

CCSDS 131.1-O-1

August 2006

AUTHORITY

Issue:	Orange Book, Issue 1
Date:	August 2006
Location:	Washington, DC, USA

This document has been approved for release by the Consultative Committee for Space Data Systems (CCSDS). The procedure for review and authorization of CCSDS documents is detailed in the *Procedures Manual for the Consultative Committee for Space Data Systems*.

This document is published and maintained by:

CCSDS Secretariat
Office of Space Communication (Code M-3)
National Aeronautics and Space Administration
Washington, DC 20546, USA

FOREWORD

This document is a CCSDS Experimental specification for a set of Low Density Parity Check (LDPC) codes. It was contributed to CCSDS by NASA.

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Experimental Specification is therefore subject to CCSDS document management and change control procedures, which are defined in the *Procedures Manual for the Consultative Committee for Space Data Systems*. Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/

Questions relating to the contents or status of this document should be addressed to the CCSDS Secretariat at the address indicated on page i.

PREFACE

This document is a CCSDS Experimental Specification. Its Experimental status indicates that it is part of a research or development effort based on prospective requirements, and as such it is not considered a Standards Track document. Experimental Specifications are intended to demonstrate technical feasibility in anticipation of a 'hard' requirement that has not yet emerged. Experimental work may be rapidly transferred onto the Standards Track should a hard requirement emerge in the future.

DOCUMENT CONTROL

Document	**Title**	**Date**	**Status**
CCSDS 131.1-O-1	Low Density Parity Check Codes for Use in Near-Earth and Deep Space Applications, Experimental Specification, Issue 1	August 2006	Original issue

CONTENTS

CONTENTS (continued)

1 INTRODUCTION

1.1 APPLICABILITY AND SCOPE

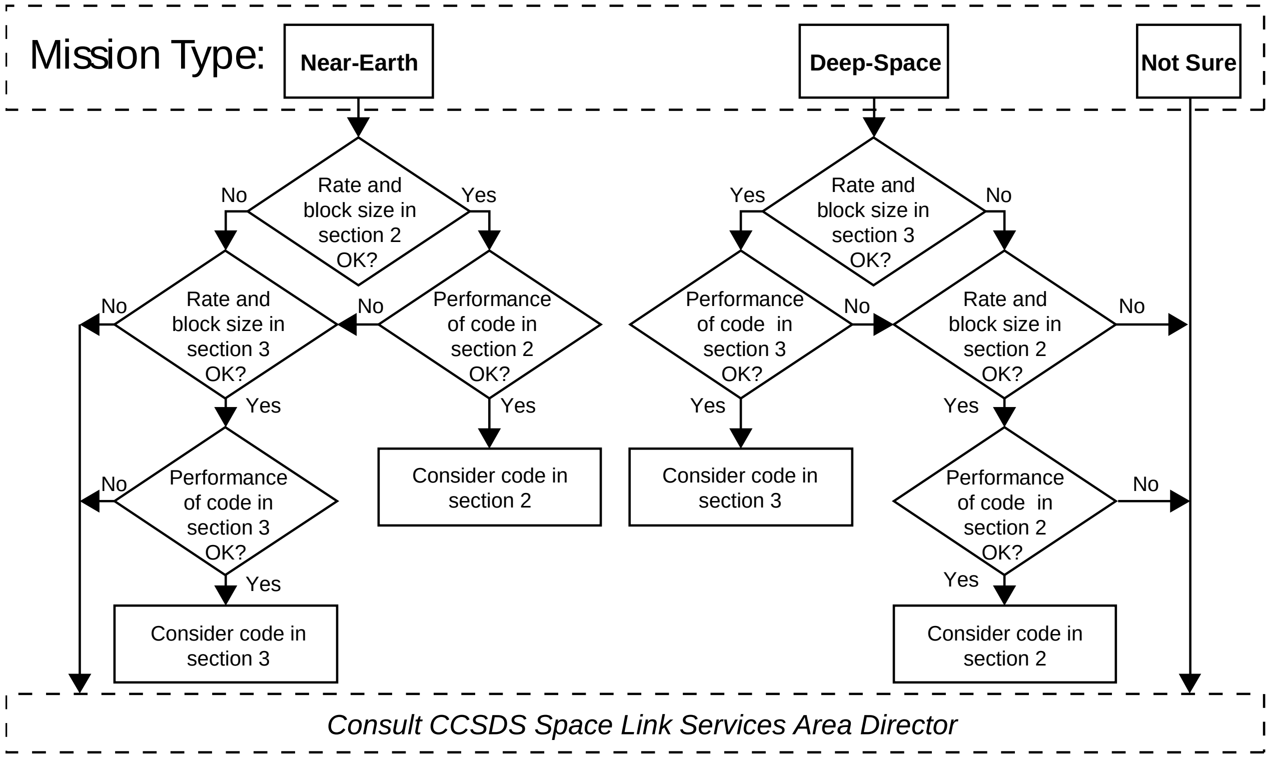
This CCSDS Experimental specification has been contributed to CCSDS by NASA. It describes a set of Low Density Parity Check (LDPC) codes including definition of their code structures, encoder implementations and experimental performance results.

This CCSDS Experimental specification is composed of two parts. The single rate 7/8 code described in section 2 was designed and optimized for the characteristics and requirements typical of many Near-Earth missions, whereas the set of codes described in section 3 was designed and optimized for the characteristics and requirements typical of many Deep-Space missions.

Users are cautioned that selection of the most appropriate code for a particular mission environment is a complex problem that involves evaluation of several characteristics, including:

– error rate performance, including error floors;
– encoder and decoder complexity, including the number of decoder iterations required to achieve desired performance;
– familial relationships between multiple codes;
– maturity and availability of implementations and implementation support;
– intellectual property issues;
– etc.

In addition, users may need to perform independent analysis and simulation. In order to begin the process of code selection, users are advised to approach the problem as follows:


Mission Type:
Near-Earth
Deep-Space
Not Sure
Rate and block size in section 2 OK?
No
Yes
Rate and block size in section 3 OK?
Performance of code in section 2 OK?
Performance of code in section 3 OK?
Consider code in section 2
Consider code in section 3
Rate and block size in section 3 OK?
Performance of code in section 3 OK?
Rate and block size in section 2 OK?
Performance of code in section 2 OK?
Consider code in section 3
Consider code in section 2
Consult CCSDS Space Link Services Area Director

1.2 NOMENCLATURE

The following conventions apply throughout this Specification:

a) the words 'shall' and 'must' imply a binding and verifiable specification;

b) the word 'should' implies an optional, but desirable, specification;

c) the word 'may' implies an optional specification;

d) the words 'is', 'are', and 'will' imply statements of fact.

1.3 CONVENTIONS

In this document, the following convention is used to identify each bit in an *N*-bit field. The first bit in the field to be transmitted (i.e., the most left justified when drawing a figure) is defined to be 'Bit 0', the following bit is defined to be 'Bit 1', and so on up to 'Bit *N*-1'. When the field is used to express a binary value (such as a counter), the Most Significant Bit (MSB) shall be the first transmitted bit of the field, i.e., 'Bit 0' (see figure 1-1).

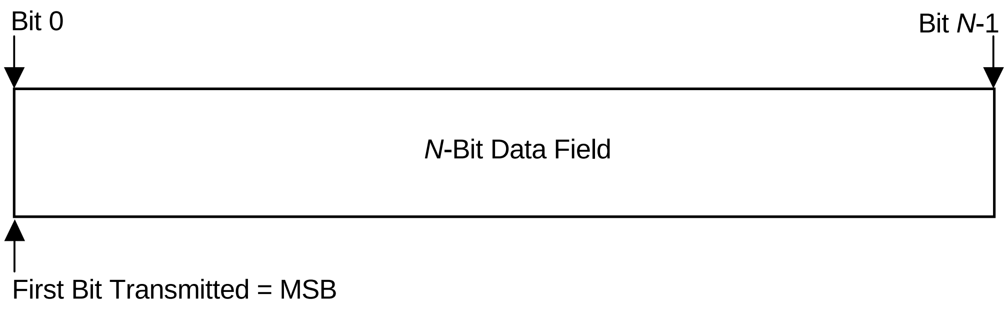


Figure 1-1: Bit Numbering Convention

The convention for matrices differs from that for bit fields. Matrices are indexed beginning with the number '1'.

In accordance with standard data-communications practice, data fields are often grouped into 8-bit 'words' which conform to the above convention. Throughout this Specification, such an 8-bit word is called an 'octet'.

The numbering for octets within a data structure starts with '0'.

1.4 REFERENCES

The following document contains provisions which, through reference in this text, constitute provisions of this Experimental Specification. At the time of publication, the edition indicated was valid. All documents are subject to revision, and users of this Experimental Specification are encouraged to investigate the possibility of applying the most recent edition of the document indicated below. The CCSDS Secretariat maintains a register of currently valid CCSDS documents.

[1] *TM Synchronization and Channel Coding*. Recommendation for Space Data System Standards, CCSDS 131.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, September 2003.

NOTE – Annex C contains lists of informative references.

2 LOW DENSITY PARITY CHECK CODE OPTIMIZED FOR NEAR EARTH APPLICATIONS

2.1 OVERVIEW

2.1.1 BACKGROUND

The mid-1990s were highlighted by the rediscovery of Low Density Parity Check codes (LDPCC) in the field of channel coding (reference [C1.1]). Originally invented by R. Gallager in his PhD thesis in 1961 (reference [C1.2]), this coding technique was largely forgotten for more than 30 years. The primary advance in LDPCC is the discovery of an iterative decoding algorithm, now called Belief Propagation (BP) decoding, which offers near-optimum performance for large linear LDPCC at a manageable complexity. LDPCC performance gains were difficult to realize technologically in the early 1960s. Several decades of VLSI development has finally made the implementation of these codes practical.

The original construction, now called Gallager LDPCC, has come to be regarded as a special class of LDPCC. Recent advances in LDPC code construction have resulted in the development of new codes with (arguably) improved performance over Gallager LDPCC. One class of these codes, irregular LDPCC (reference [C1.3]), demonstrates improved performance in the waterfall region. Disadvantages of irregular codes, however, include an increase, in general, in the number of iterations required for decoding convergence and an unequal error protection between code bits resulting from the irregular structure. Another class of LDPCC developed using algebraic construction based on finite geometries (reference [C1.4]) has been shown to provide very low error floors and very fast iterative convergence. These qualities make these codes a good fit for near Earth applications where very high data rates and high reliability are the driving requirements. A white paper based on Euclidean Geometry LDPCC (reference [C1.5]) was submitted to the CCSDS Channel Coding Subpanel P1B in the fall 2002 Houston meeting. That paper and subsequent research is the basis for the code presented in this Experimental Specification.

2.1.2 TECHNICAL FOUNDATION

A linear block code is designated in this Experimental Specification by (n, k) where n is the length of the codeword (or block) and k is the length of the information sequence. LDPC codes are linear block codes in which the ratio of the total number of '1's to the total number of elements in the parity check matrix is $<< 0.5$. The distribution of the '1's determine the structure and performance of the decoder. An LDPC code is defined by its parity check matrix. The $k \times n$ generator matrix which is used to encode a linear block code can be derived from the parity check matrix through linear operations. (The reader is encouraged to review reference [C1.8] for an overview of linear block codes).

The LDPC code considered in this specification is a member of a class of codes called Quasi-Cyclic codes. The construction of these codes involves juxtaposing smaller circulants (or cyclic submatrices) to form a larger parity check or base matrix.

An example of a circulant is shown in figure 2-1. Notice that every row is one bit right cyclic shift (where the end bit is wrapped around to the beginning bit) of the previous row. The entire circulant is uniquely determined and specified by its first row. For this example the first row has four ‘1’s or a row weight of four.

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

Figure 2-1: Example of a 15 × 15 Circulant Matrix

An example of a quasi-cyclic parity check matrix is shown in figure 2-2. In this case, a quasi-cyclic 10 × 25 matrix is formed by an array of 2 × 5 circulant submatrices of size 5 × 5. To unambiguously describe this matrix, only the position of the ‘1’s in the first row of every circulant submatrix and the location of each submatrix within the base matrix is needed.

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

Figure 2-2: Example of a Quasi-Cyclic Matrix

Constructing parity check matrices in this manner produces two positive features:

a) the encoding complexity can be made linear with the code length or parity bits using shift registers, and

b) encoder and decoder routing complexity in the interconnections of integrated circuits is reduced.

2.2 BASE (8176,7156) LDPC CODE

This section describes the base (8176, 7156) LDPC code. For reasons outlined below, implementations should shorten the base code according to the format described in subsection 2.5. The parity check matrix for the (8176, 7156) LDPC code is formed by using a 2 × 16 array of 511 × 511 square circulants. This creates a parity check matrix of dimension 1022 × 8176. The structure of the parity check base matrix is shown in figure 2-3.

$$\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} & A_{1,5} & A_{1,6} & A_{1,7} & A_{1,8} & A_{1,9} & A_{1,10} & A_{1,11} & A_{1,12} & A_{1,13} & A_{1,14} & A_{1,15} & A_{1,16} \\ A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} & A_{2,5} & A_{2,6} & A_{2,7} & A_{2,8} & A_{2,9} & A_{2,10} & A_{2,11} & A_{2,12} & A_{2,13} & A_{2,14} & A_{2,15} & A_{2,16} \end{bmatrix}$$

Figure 2-3: Base Parity Check Matrix of the (8176, 7156) LDPC Code

Each $A_{i,j}$ is a 511 × 511 circulant. The row weight of the each of the 32 circulants is two; i.e., there are two '1's in each row. The total row weight of each row in the parity check matrix is 2 × 16, or 32. The column weight of each circulant is also two; i.e., there are two '1's in each column. The total weight of each column in the parity check matrix is 2 × 2 or four. The position of the '1's in each circulant is defined in table 2-1. A scatter chart of the parity check matrix is shown in figure 2-4 where every '1' bit in the matrix is represented by a point.

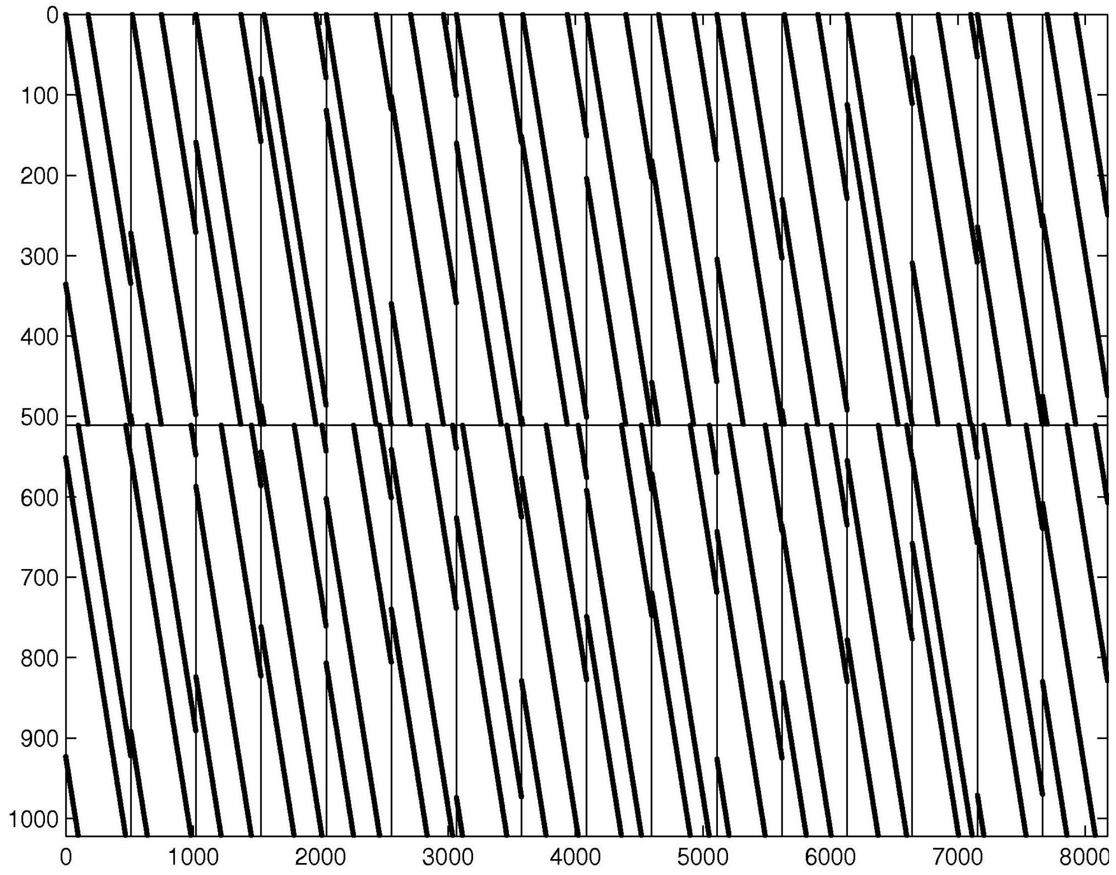


Figure 2-4: Scatter Chart of Parity Check Matrix

Table 2-1: Specification of Circulants

Circulant	'1's position in 1st row of circulant	Absolute '1's position in 1st row of Parity Check Matrix
$A_{1,1}$	0, 176	0, 176
$A_{1,2}$	12, 239	523, 750
$A_{1,3}$	0, 352	1022, 1374
$A_{1,4}$	24, 431	1557, 1964
$A_{1,5}$	0, 392	2044, 2436
$A_{1,6}$	151, 409	2706, 2964
$A_{1,7}$	0, 351	3066, 3417
$A_{1,8}$	9, 359	3586, 3936
$A_{1,9}$	0, 307	4088, 4395
$A_{1,10}$	53, 329	4652, 4928
$A_{1,11}$	0, 207	5110, 5317
$A_{1,12}$	18, 281	5639, 5902
$A_{1,13}$	0, 399	6132, 6531
$A_{1,14}$	202, 457	6845, 7100
$A_{1,15}$	0, 247	7154, 7401
$A_{1,16}$	36, 261	7701, 7926
$A_{2,1}$	99, 471	99, 471
$A_{2,2}$	130, 473	641, 984
$A_{2,3}$	198, 435	1220, 1457
$A_{2,4}$	260, 478	1793, 2011
$A_{2,5}$	215, 420	2259, 2464
$A_{2,6}$	282, 481	2837, 3036
$A_{2,7}$	48, 396	3114, 3462
$A_{2,8}$	193, 445	3770, 4022
$A_{2,9}$	273, 430	4361, 4518
$A_{2,10}$	302, 451	4901, 5050
$A_{2,11}$	96, 379	5206, 5489
$A_{2,12}$	191, 386	5812, 6007
$A_{2,13}$	244, 467	6376, 6599
$A_{2,14}$	364, 470	7007, 7113
$A_{2,15}$	51, 382	7205, 7536
$A_{2,16}$	192, 414	7857, 8079

Note that the numbers in the second column represent the relative column position of the ‘1’s in the first row of each circulant. Since there are only 511 possible positions, these numbers can range only from 0 to 510. The third column represents the absolute position of the ‘1’s in the parity-check matrix. There are exactly 8176 possible; therefore these numbers can range only from 0 to 8175.

2.3 ENCODING

The encoder can be designed using the method given in reference [C1.6]. The generator matrix of the (8176, 7156) code consists of two parts. The first part is a 7154×8176 submatrix in systematic-circulant form as shown in figure 2-5. It consists of a 7154×7154 identity matrix and two columns of 511×511 circulants $\mathrm{B}_{i,j}$s, each column consisting of 14 circulants. The Is are the 511×511 identity submatrices and the 0s are the all zero 511×511 submatrices. The second part consists of two independent rows. The first part generates a (8176, 7154) LDPC subcode of the (8176, 7156) code. Each codeword in the subcode consists of 7154 information bits and 1022 parity-check bits. For reason given in section 2.4, there are advantages in using the subcode implementation. The circulants $\mathrm{B}_{i,j}$s are constructed based on the algorithm given below:

1) From figure 2-3 and table 2-1, define $\mathrm{D} = \begin{bmatrix} \mathrm{A}_{1,15} & \mathrm{A}_{1,16} \\ \mathrm{A}_{2,15} & \mathrm{A}_{2,16} \end{bmatrix}$, which is a 1022×1022 matrix.
2) Let $u = (1\ 0\ 0\ 0\ \ldots.\ 0)$ be the unit 511 tuple, i.e., a vector quantity of length 511 with a ‘1’ at the leftmost position and ‘0’s in the rest.
3) Define $z_i = (b_{i,1} \quad b_{i,2})$, where i = 1, 2, ..., 14 and the $b_{i,j}$s are the first row of the $\mathrm{B}_{i,j}$ circulants.
4) Define $\mathrm{M}_i = \begin{bmatrix} \mathrm{A}_{1,i} \\ \mathrm{A}_{2,i} \end{bmatrix}$, where i = 1, 2, …, 14. (Note that the parity check matrix can now be represented as: $[\mathrm{M}_1\ \mathrm{M}_2 \ldots \mathrm{M}_{14}\ \mathrm{D}]$.)
5) Since the rank of D is 1020 and not 1022, there are two linearly dependent columns, 511th and 1022nd. Set the 511th and 1022nd elements of z_i to zero and solve $\mathrm{M}_i u^{\mathrm{T}} + \mathrm{D} z_i^{\mathrm{T}} = 0$ for z_i, where i = 1, 2, …, 14 and T superscript represents matrix transpose.
6) The $b_{i,j}$s can be extracted from the z_is. (They are numerically tabulated in annex A1.)

There are many ways to design the encoder based on the generator matrix in figure 2-5. These schemes have complexities that are proportional to the length of the codeword or parity check bits (see reference [C1.6]).

$$\begin{bmatrix}
I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{1,1} & B_{1,2} \\
0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{2,1} & B_{2,2} \\
0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{3,1} & B_{3,2} \\
0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{4,1} & B_{4,2} \\
0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{5,1} & B_{5,2} \\
0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{6,1} & B_{6,2} \\
0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{7,1} & B_{7,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & B_{8,1} & B_{8,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & B_{9,1} & B_{9,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & B_{10,1} & B_{10,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & B_{11,1} & B_{11,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & B_{12,1} & B_{12,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & B_{13,1} & B_{13,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & B_{14,1} & B_{14,2}
\end{bmatrix}$$

Figure 2-5: Systematic Circulant Generator Matrix

2.4 SHORTENED (8160, 7136) CODE

Using the generator matrix given by figure 2-5, an encoder can be implemented using a circuit described in reference [C1.6]. This encoder generates a (8176, 7154) LDPC subcode of the (8176, 7156) code. Current spacecraft and ground systems manipulate and process data at 32-bit computer word size. Neither (8176, 7154) nor (8176, 7156) is a multiple of 32. It is beneficial to shorten the codeword to the dimensions of (8160, 7136). In other words, by shortening the information sequence to 7136 through the use of 18 bits of virtual fill, the (8176, 7154) subcode encoder can be used. This is accomplished by encoding the virtual fill bits with zeros but not transmitting them; thus the total codeword length becomes 8158. Note that it is not necessary to add two independent rows to the generator matrix to encode the full (8176, 7156) code because these bits would be shortened anyway, and so the subcode is sufficient and less complicated for this application. Since the code length of 8158 is two bits shy of 8160, an exact multiple of 32, two bits of actual transmitted zero fill are appended to the end of the codeword to achieve a shortened code dimension of (8160, 7136) bits, or (1020, 892) octets, or (255, 223) 32-bit words. The shortened codeword is shown in figure 2-6.

The received shortened codeword would require the removal of the two zero fill bits prior to decoding. The decoder would then reproduce the 18 virtual fill zeros after processing but would, in general, not pass these 18 zeros on to the ground equipment.

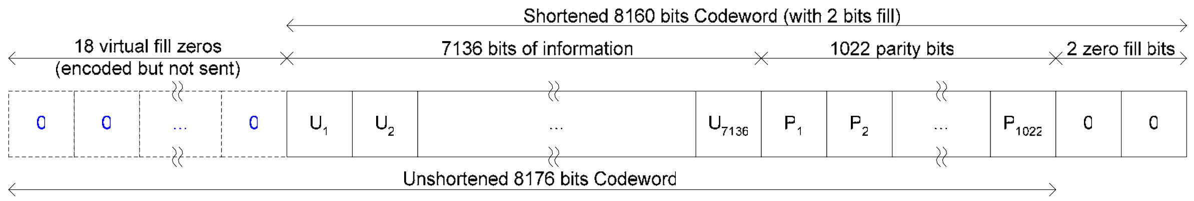

Shortened 8160 bits Codeword (with 2 bits fill)
18 virtual fill zeros (encoded but not sent)
7136 bits of information
1022 parity bits
2 zero fill bits
0
0
...
0
U_1
U_2
...
U_{7136}
P_1
P_2
...
P_{1022}
0
0
Unshortened 8176 bits Codeword

Figure 2-6: Shortened Codeword

2.5 RANDOMIZATION AND SYNCHRONIZATION

The use of the shortened (8160, 7136) LDPC code does not guarantee sufficient bit (symbol) transitions to acquire or maintain bit (symbol) synchronization. It is highly recommended that a pseudo-randomizer be used after encoding in accordance with section 7 of reference [1].

In addition, frame (codeword) synchronization is required so that the receiver can identify the beginning of the frame (codeword) for proper decoding. The use of an attached sync marker (ASM) as specified in section 6.6 of reference [1] is required. Note that the ASM is not pseudo-randomized.

3 LOW DENSITY PARITY CHECK CODE FAMILY OPTIMIZED FOR DEEP SPACE APPLICATIONS

3.1 OVERVIEW

3.1.1 BACKGROUND

The Low-Density Parity-Check (LDPC) codes presented in this document are intended to complement the current codes in the CCSDS Recommended Standard, *TM Synchronization and Channel Coding* (reference [1]), and were designed according to a list of requirements and evaluation criteria that reflect the needs of spacecraft applications (see reference [C2.2]).

- Requirements
  - *Code rates*: The family shall include codes of rate ≈ 0.5 and ≈ 0.8
  - *Block lengths*: The family shall cover $k\approx1000$ to $k\approx16000$ information bits spaced by multiples of ≈4
  - *Family*: A single hardware decoder shall be appropriate for all codes
  - *Intellectual property*: There must be no restrictions for CCSDS members
- Desired Properties
  - *Systematic encoders*: Systematic encoders are preferred
  - *Code rates*: One or two intermediate rates from {1/2,2/3,3/4,4/5} are desired
- Evaluation Criteria
  - *Decoder computation*: Codes requiring fewer decoder message computations are preferred
  - *Encoder computation*: Preferred encoders require fewer logic gates for a given speed
  - *Descriptional complexity*: The code description in a standards document should be short
  - *Code performance*: Codes requiring less E_b/N_0 at Word Error Rate (WER)=10^{-4} and 10^{-6} are preferred

The selected code rates are 1/2, 2/3, and 4/5, three values, which are about uniformly spaced by 1 dB on the rate-dependent capacity curve for the binary-input Additive White Gaussian Noise (AWGN) channel (reference [C2.4]). Near rate 1/2, a 1% improvement in bandwidth efficiency costs about 0.02 dB in power efficiency; near rate 7/8, a 1% improvement in bandwidth efficiency costs 0.1 dB in power efficiency. Hence, the use of a higher order modulation may be a more practical means for saving bandwidth than the use of a code with rate much above 0.8. The code rates are exact ratios of small integers to simplify implementation.

The selected block lengths are k=1024, k=4096, and k=16384. The three values k={1024,4096,∞} are about uniformly spaced by 0.6 dB on the sphere-packing bound at WER=10^{-8}, and reducing the last value from ∞ to 16384 makes the largest block size practical at a cost of about 0.3 dB. By choosing to keep k constant among family members, rather than n, the spacecraft's command and data handling system can generate data frames without knowledge of the code rate. Choosing powers of 2 may simplify implementation.

Implementers should be aware that many patents have been filed on LDPC codes; in particular, a procedure for parallelized decoding of LDPC codes is covered by a U.S. patent granted to T. Richardson and V. Novichkov.[1]

The selected codes are systematic. Two low-complexity encoding methods are described (see reference [C2.3]), and either can be used, depending on constraints of the chosen technology (software, FPGA, ASIC). The parity check matrices have plenty of structure to facilitate decoder implementation (see reference [C2.6]). The codes have irregular degree distributions, because this improves performance by about 0.5 dB at rate 1/2, compared to a regular (3,6) code (see reference [C2.1]).

3.1.2 DESCRIPTION OF THE CODE

Like turbo codes, low-density parity-check (LDPC) codes are binary block codes with large code blocks (hundreds or thousands of bits). They may be systematic or non-systematic, and they may be transparent or non-transparent. The nine LDPC codes defined here are systematic. All are non-transparent,[2] and phase ambiguities are resolved using frame markers, which are required for Codeblock synchronization.

Like turbo codes, LDPC codes may be used to obtain greater coding gain than those provided by concatenated coding systems. In contrast to turbo codes, LDPC codes offer the prospect of much higher decoding speeds via highly parallelized decoder structures. Currently reference [1] includes turbo codes of rates 1/2 and lower and a convolutional code rate 1/2. LDPC codes of rates 1/2 and higher are defined in this experimental specification. Therefore rate 1/2 is the only rate at which this specification and reference [1] are comparable.

[1] Richardson, Tom, and Novichkov, Vladimir. 2001. Methods and apparatus for decoding LDPC codes. U.S. Patent 6,633,856, filed October 10, 2001, and issued October 14, 2003.

[2]Differential encoding (i.e., NRZ-M signaling) after the LDPC encoder is not recommended since soft decoding would require the use of differential detection with considerable loss of performance. Differential encoding before the LDPC encoder cannot be used because the LDPC codes specified in this document are non-transparent. This implies that phase ambiguities have to be detected and resolved by the frame synchronizer.

NOTES

1 LDPC coding, by itself, cannot guarantee sufficient bit transitions to keep receiver symbol synchronizers in lock. Therefore, the Pseudo-Randomizer defined in section 3.5 is required unless the system designer verifies that sufficient symbol transition density is assured by other means when the Randomizer is not used.

2 While providing outstanding coding gain, LDPC codes generally may still leave some residual errors in the decoded output. For this reason, when CCSDS TM or AOS Transfer Frames are used, references [C2.7] and [C2.8], respectively, require that a Cyclic Redundancy Check (CRC) be used to validate the frame.

3.2 SPECIFICATION

An LDPC code is specified indirectly by a v-by-w parity-check matrix H consisting of v linearly independent rows. A coded sequence of w bits must satisfy all v parity-check equations corresponding to the v rows of H. Parity-check matrices may include additional linearly dependent rows without changing the code. An encoder maps an input frame of $k \leq w-v$ information bits uniquely into a codeblock of $n \leq w$ bits. If $n<w$, the remaining $w-n$ code symbols are punctured and are not transmitted. If $k<w-v$, the remaining dimensions of the code remain unused.

The codeblock lengths n and information block lengths k, and the corresponding rates $r=k/n$, are shown in table 3-1 for the suite of LDPC codes. The LDPC code rates r are exactly as indicated in table 3-1, unlike the case of turbo codes for which the precise code rates are slightly lower than the corresponding nominal rates due to termination bits.

Table 3-1: Codeblock Lengths for Supported Code Rates (Measured in Bits)

Information block length k	Code block length n		
	rate 1/2	rate 2/3	rate 4/5
1024	2048	1536	1280
4096	8192	6144	5120
16384	32768	24576	20480

Table 3-2: Values of Submatrix Size *M* for Supported Codes

Information block length k	Submatrix size M		
	rate 1/2	rate 2/3	rate 4/5
1024	512	256	128
4096	2048	1024	512
16384	8192	4096	2048

For each (n,k) in table 3-1, both the parity-check matrix H and the encoding rule for generating n code bits from k information bits are specified.

3.3 PARITY CHECK MATRICES

The *H* matrices are constructed from *M*×*M* submatrices, where the submatrix size is listed in table 3-2.

The *H* matrices for the rate-1/2 codes are specified as follows,

$$H_{1/2} = \begin{bmatrix} 0_M & 0_M & 0_M & I_M & I_M \oplus \Pi_1 \\ I_M \oplus \Pi_8 & I_M \oplus \Pi_7 \oplus \Pi_6 & 0_M & 0_M & I_M \\ 0_M & I_M & I_M \oplus \Pi_5 & 0_M & \Pi_4 \oplus \Pi_3 \oplus \Pi_2 \end{bmatrix}$$

where I_M and 0_M are the *M*×*M* identity and zero matrices, respectively, and Π_1 through Π_8 are permutation matrices. The *H* matrices for the rate-2/3 and rate-4/5 codes are specified with additional columns and permutation matrices as follows.

$$H_{2/3} = \left[\begin{array}{cc|c} 0_M & 0_M & \\ I_M & \Pi_{11} \oplus \Pi_{10} \oplus \Pi_9 & H_{1/2} \\ \Pi_{14} \oplus \Pi_{13} \oplus \Pi_{12} & I_M & \end{array}\right]$$

$$H_{4/5} = \left[\begin{array}{cccc|c} 0_M & 0_M & 0_M & 0_M & \\ I_M & \Pi_{23} \oplus \Pi_{22} \oplus \Pi_{21} & I_M & \Pi_{17} \oplus \Pi_{16} \oplus \Pi_{15} & H_{2/3} \\ \Pi_{26} \oplus \Pi_{25} \oplus \Pi_{24} & I_M & \Pi_{20} \oplus \Pi_{19} \oplus \Pi_{18} & I_M & \end{array}\right]$$

Permutation matrix Π_k has non-zero entries in row *i* and column $\pi_k(i)$ for $i \in \{0,..., M\text{-}1\}$ and

$$\pi_k(i) = \frac{M}{4}\left(\left(\theta_k + \lfloor 4i/M \rfloor\right) \bmod 4\right) + \left(\phi_k\left(\lfloor 4i/M \rfloor\right) + i\right) \bmod \frac{M}{4}$$

where the functions θ_k and $\phi_k(j)$ are defined by:

	θ_k	$\phi_k(0)$	$\phi_k(1)$	$\phi_k(2)$	$\phi_k(3)$
k=1	1	1787	1502	1887	1291
2	1	1077	602	521	301
3	2	1753	749	590	1353
4	3	697	1662	1775	1405
5	1	1523	1371	1738	997
6	1	5	9	2032	2032
7	2	2035	131	2047	11
8	3	331	1884	85	1995
9	1	1920	1268	1572	623
10	2	130	1784	78	73
11	3	4	19	26	1839
12	0	85	1839	298	2003
13	2	551	81	1177	2019
14	3	15	2031	1950	1841
15	0	1780	76	1806	167
16	2	1960	336	128	1087
17	3	3	529	1855	2032
18	0	145	74	129	388
19	1	1019	68	269	1385
20	3	691	186	1614	885
21	0	132	905	1467	707
22	1	42	1751	1533	1272
23	3	393	1516	925	7
24	0	502	1285	1886	1534
25	1	201	1597	2046	1965
26	2	1064	1712	1167	588

These parity check matrices have one redundant check equation: the last 2M rows sum to zero (modulo 2). The last M code symbols are punctured. One dimension of the code is not used.

For example, the parity check matrix for the (n=1536, k=1024) code is shown in figure 3-1 with dots representing each of the non-zero entries, and its structure is indicated by gridlines.

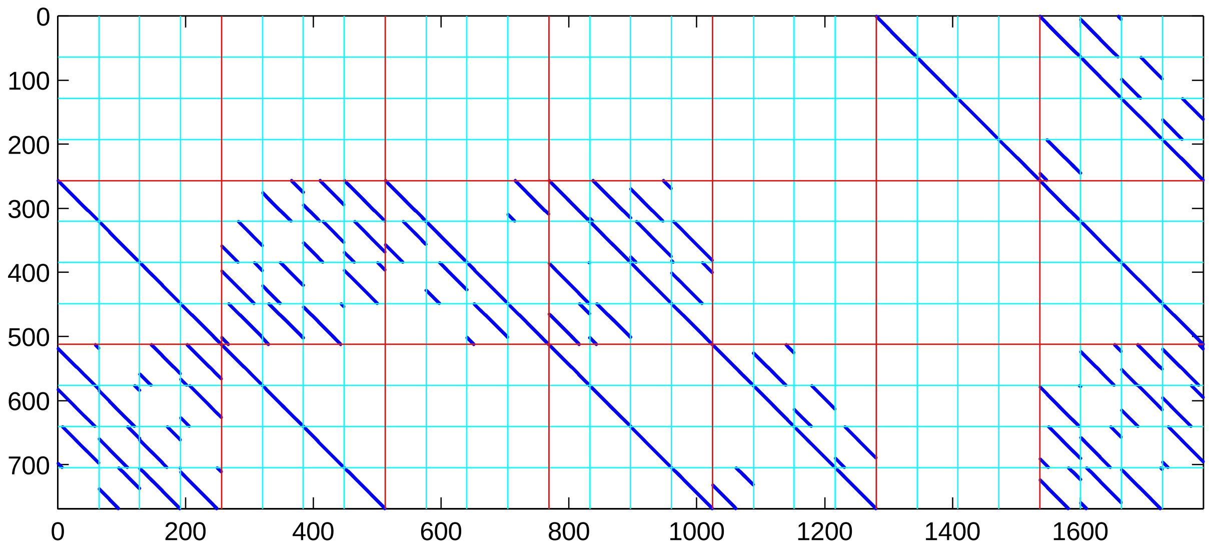


Figure 3-1: An *H* Matrix for the (*n*=1536, *k*=1024) Code

3.4 ENCODERS

The encoding rule for producing codeblocks consistent with these *H* matrices is as follows. The input message is $(m_0, m_1, ..., m_{k-1})$ and the symbol $\oplus$ denotes the exclusive OR operation. The submatrix size is *M* as listed in table 3-2.

The five encoding steps are:

1) For 0≤*j*≤7, define the vectors:

$$s^j = \begin{cases} (m_{jM}, ..., m_{(j+1)M-1}) & \text{if } j < k/M \\ (0,...,0), \text{ the length} - M \text{ zero vector} & \text{otherwise} \end{cases}$$

Thus, potentially non-zero vectors are s^0 and s^1 for rate 1/2 codes, s^0 through s^3 for rate 2/3 codes, and s^0 through s^7 for rate 4/5 codes.

2) Compute the vector p^2 of length *M* by sparse matrix multiplication:

$$p_i^2 = s_i^0 \oplus s_{\pi_6(i)}^0 \oplus s_{\pi 7(i)}^0 \oplus s_i^1 \oplus s_{\pi 8(i)}^1 \oplus s_{\pi_9(i)}^2 \oplus s_{\pi_{10}(i)}^2 \oplus s_{\pi_{11}(i)}^3 \oplus s_i^3$$
$$\oplus s_{\pi_{15}(i)}^4 \oplus s_{\pi_{16}(i)}^4 \oplus s_{\pi_{17}(i)}^4 \oplus s_i^5 \oplus s_{\pi_{21}(i)}^6 \oplus s_{\pi_{22}(i)}^6 \oplus s_{\pi_{23}(i)}^6 \oplus s_i^7$$

3) Compute the non-accumulated portion of the parity p^1 by sparse matrix multiplication:

$$p_i^1 = p_i^2 \oplus p_{\pi_1(i)}^2$$

4) Compute the accumulated portion of the parity, p^0, by initializing:

   $j_0 = 0$ and $p_0^0 = 0$

   For $i \in \{0,\ldots,M\text{-}1\}$, compute,

   $j_{i+1} = \pi_5(j_i)$

$$p^0_{j_{i+1}} = p^0_{j_i} \oplus p^2_{\pi_2(j_i)} \oplus p^2_{\pi_3(j_i)} \oplus p^2_{\pi_4(j_i)} \oplus s^0_{j_i} \oplus s^2_{j_i} \oplus s^3_{\pi_{12}(j_i)} \oplus s^3_{\pi_{13}(j_i)} \oplus s^3_{\pi_{14}(j_i)} \\ \oplus s^4_{j_i} \oplus s^5_{\pi_{18}(j_i)} \oplus s^5_{\pi_{19}(j_i)} \oplus s^5_{\pi_{20}(j_i)} \oplus s^6_{j_i} \oplus s^7_{\pi_{24}(j_i)} \oplus s^7_{\pi_{25}(j_i)} \oplus s^7_{\pi_{26}(j_i)}$$

5) Form the output codeword c:

$$c = \begin{cases} (s^1, s^0, p^0, p^1) & \text{if } r = 1/2 \\ (s^3, s^2, s^1, s^0, p^0, p^1) & \text{if } r = 2/3 \\ (s^7, s^6, s5, s^4, s^3, s^2, s^1, s^0, p^0, p^1) & \text{if } r = 4/5 \end{cases}$$

3.5 SYNCHRONIZATION

Codeblock synchronization is achieved by synchronization of an Attached Sync Marker associated with each LDPC Codeblock. The Attached Sync Marker (ASM) is a bit pattern specified in section 6 of CCSDS Recommended Standard CCSDS 131.0-B-1, *TM Synchronization and Channel Coding* (reference [1]), as an aid to synchronization, and it precedes the LDPC Codeblock. Frame synchronizers should be set to expect a marker at a recurrence interval equal to the length of the ASM plus that of the Turbo Codeblock. All codes in the LDPC family use the 64-bit ASM.

ANNEX A

ANNEX TO SECTION 2, LOW DENSITY PARITY CHECK CODE OPTIMIZED FOR NEAR-EARTH APPLICATIONS

A1 GENERATOR MATRIX CIRCULANT TABLE

Table A-1: Table of Circulants for the Generator Matrix

Circulant	1st row of circulant
$B_{1,1}$	55BF56CC55283DFEEFEA8C8CFF04E1EBD9067710988E25048D67525426939E206 8D2DC6FCD2F822BEB6BD96C8A76F4932AAE9BC53AD20A2A9C86BB461E43759C
$B_{1,2}$	6855AE08698A50AA3051768793DC238544AF3FE987391021AAF6383A6503409C3C E971A80B3ECE12363EE809A01D91204F1811123EAB867D3E40E8C652585D28
$B_{2,1}$	62B21CF0AEE0649FA67B7D0EA6551C1CD194CA77501E0FCF8C85867B9CF679C1 8BCF7939E10F8550661848A4E0A9E9EDB7DAB9EDABA18C168C8E28AACDDEAB1 E
$B_{2,2}$	64B71F486AD57125660C4512247B229F0017BA649C6C11148FB00B70808286F1A97 90748D296A593FA4FD2C6D7AAF7750F0C71B31AEE5B400C7F5D73AAF00710
$B_{3,1}$	681A8E51420BD8294ECE13E491D618083FFBBA830DB5FAF330209877D801F92B5 E07117C57E75F6F0D873B3E520F21EAFD78C1612C6228111A369D5790F5929A
$B_{3,2}$	04DF1DD77F1C20C1FB570D7DD7A1219EAECEA4B2877282651B0FFE713DF338A6 3263BC0E324A87E2DC1AD64C9F10AAA585ED6905946EE167A73CF04AD2AF9218
$B_{4,1}$	35951FEE6F20C902296C9488003345E6C5526C5519230454C556B8A04FC0DC642D 682D94B4594B5197037DF15B5817B26F16D0A3302C09383412822F6D2B234E
$B_{4,2}$	7681CF7F278380E28F1262B22F40BF3405BFB92311A8A34D084C086464777431DBF DDD2E82A2E6742BAD6533B51B2BDEE0377E9F6E63DCA0B0F1DF97E73D5CD8
$B_{5,1}$	188157AE41830744BAE0ADA6295E08B79A44081E111F69BBE7831D07BEEBF76232 E065F752D4F218D39B6C5BF20AE5B8FF172A7F1F680E6BF5AAC3C4343736C2
$B_{5,2}$	5D80A6007C175B5C0DD88A442440E2C29C6A136BBCE0D95A58A83B48CA0E7474 E9476C92E33D164BFF943A61CE1031DFF441B0B175209B498394F4794644392E
$B_{6,1}$	60CD1F1C282A1612657E8C7C1420332CA245C0756F78744C807966C3E132643887 8BD2CCC83388415A612705AB192B3512EEF0D95248F7B73E5B0F412BF76DB4
$B_{6,2}$	434B697B98C9F3E48502C8DBD891D0A0386996146DEBEF11D4B833033E05EDC28 F808F25E8F314135E6675B7608B66F7FF3392308242930025DDC4BB65CD7B6E
$B_{7,1}$	766855125CFDC804DAF8DBE3660E8686420230ED4E049DF11D82E357C54FE256E A01F5681D95544C7A1E32B7C30A8E6CF5D0869E754FFDE6AEFA6D7BE8F1B148
$B_{7,2}$	222975D325A487FE560A6D146311578D9C5501D28BC0A1FB48C9BDA173E869133 A3AA9506C42AE9F466E85611FC5F8F74E439638D66D2F00C682987A96D8887C
$B_{8,1}$	14B5F98E8D55FC8E9B4EE453C6963E052147A857AC1E08675D99A308E7269FAC5 600D7B155DE8CB1BAC786F45B46B523073692DE745FDF10724DDA38FD09B1C

Circulant	1st row of circulant
$B_{8,2}$	1B71AFFB8117BCF8B5D002A99FEEA49503C0359B056963FE5271140E626F6F8FC E9F29B37047F9CA89EBCE760405C6277F329065DF21AB3B779AB3E8C8955400
$B_{9,1}$	0008B4E899E5F7E692BDCE69CE3FAD997183CFAEB2785D0C3D9CAE510316D4BD 65A2A06CBA7F4E4C4A80839ACA81012343648EEA8DBBA2464A68E115AB3F4034
$B_{9,2}$	5B7FE6808A10EA42FEF0ED9B41920F82023085C106FBBC1F56B567A14257021BC5 FDA60CBA05B08FAD6DC3B0410295884C7CCDE0E56347D649DE6DDCEEB0C95E
$B_{10,1}$	5E9B2B33EF82D0E64AA2226D6A0ADCD179D5932EE1CF401B336449D0FF775754C A56650716E61A43F963D59865C7F017F53830514306649822CAA72C152F6EB2
$B_{10,2}$	2CD8140C8A37DE0D0261259F63AA2A420A8F81FECB661DBA5C62DF6C817B4A61 D2BC1F068A50DFD0EA8FE1BD387601062E2276A4987A19A70B460C54F215E184
$B_{11,1}$	06F1FF249192F2EAF063488E267EEE994E7760995C4FA6FFA0E4241825A7F5B65C 74FB16AC4C891BC008D33AD4FF97523EE5BD14126916E0502FF2F8E4A07FC2
$B_{11,2}$	65287840D00243278F41CE1156D1868F24E02F91D3A1886ACE906CE741662B40B4 EFDFB90F76C1ADD884D920AFA8B3427EEB84A759FA02E00635743F50B942F0
$B_{12,1}$	4109DA2A24E41B1F375645229981D4B7E88C36A12DAB64E91C764CC43CCEC188E C8C5855C8FF488BB91003602BEF43DBEC4A621048906A2CDC5DBD4103431DB8
$B_{12,2}$	2185E3BC7076BA51AAD6B199C8C60BCD70E8245B874927136E6D8DD527DF0693 DC10A1C8E51B5BE93FF7538FA138B335738F4315361ABF8C73BF40593AE22BE4
$B_{13,1}$	228845775A262505B47288E065B23B4A6D78AFBDDB2356B392C692EF56A35AB4A A27767DE72F058C6484457C95A8CCDD0EF225ABA56B7657B7F0E947DC17F972
$B_{13,2}$	2630C6F79878E50CF5ABD353A6ED80BEACC7169179EA57435E44411BC7D566136 DFA983019F3443DE8E4C60940BC4E31DCEAD514D755AF95A622585D69572692
$B_{14,1}$	7273E8342918E097B1C1F5FEF32A150AEF5E11184782B5BD5A1D8071E94578B0AC 722D7BF49E8C78D391294371FFBA7B88FABF8CC03A62B940CE60D669DFB7B6
$B_{14,2}$	087EA12042793307045B283D7305E93D8F74725034E77D25D3FF043ADC5F8B5B18 6DB70A968A816835EFB575952EAE7EA4E76DF0D5F097590E1A2A978025573E

Note that the numbers in the second column represent the hexadecimal representation of the first row of each circulant. Since there are only 511 possible positions, the leftmost bit is padded with a zero to allow a 128 digit hexadecimal number. Table A-1 cannot be as efficiently described as table 2-5 because the generator circulants do not have a low density of '1's.

A2 COMPLEXITY

The complexity of LDPC codes has been an area of research and discussion. For a Field Programmable Gate Array (FPGA) or Application Specific Integrated Circuit (ASIC) implementation, the encoder's complexity are dominated by two factors: 1) the total number of required logic gates and 2) the routing complexity. For the code presented in this Experimental Specification, the quasi-cyclic property allows for the use of shift registers whose required number of logic gates is proportional to $n–k$ (see reference [C1.6]) or 8176–7156 = 1020 (unshortened). With regard to the routing complexity, there is currently no way to predict this figure: it would depend on a number of factors, such as the choice of the FPGA or ASIC, the routing algorithm, and the layout of the device.

The decoder's complexity is larger than the encoder's and even more difficult to predict. The primary complexity factors (the total number of required logic gates and the routing complexity) are a function of the choice of BP decoding algorithm (there are many) as well as the architectural decisions (i.e., parallel or serial processing, number of bits of finite precision, fixed number of iterations or stopping rule, use of look-up tables, etc.) These choices also determine the decoder's Bit Error Rate (BER) performance.

For the development of the baselined (8176, 7156) code, an FPGA implementation was used to confirm the software simulations. A Xilinx 8000 Virtex-2 FPGA was used for the test. The device contained both the encoder and decoder. The decoder algorithm was a Scaled Min-Sum Parallel BP Decoder (SMSPD) described in reference [C1.7]. The encoder algorithm was a shift register based encoder described in reference [C1.6]. An architectural evaluation was performed prior to implementation to produce a quasi-optimal implementation based on routing, logic requirements, and BER performance.

The FPGA had the following statistics:

1) encoder used 2,535 logic slices out of 46,592 available or 5.4% and 4 memory blocks out of 168 available or 2.4%;

2) decoder used 21,803 logic slices out of 46,592 or 46.8% and 137 memory blocks out of 168 or 81.5%.

The number of logic slices is an aggregate measure of the number of logic gates required and the routing complexity, while the memory blocks figure is the number of dedicated FPGA memory blocks used. It is clear from these statistics that the encoder is of much lower complexity than the decoder, using only 5.4% of the logic slices resources while the decoder requires 46%.

Annex A3 summarizes the test results.

A3 FPGA TEST RESULTS

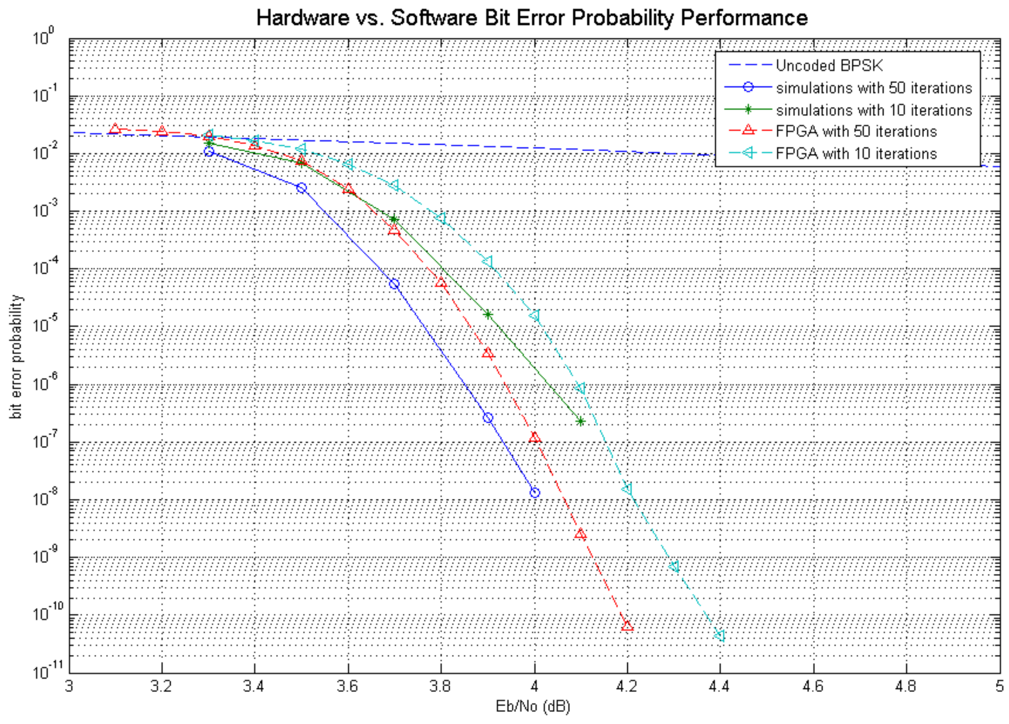


Figure A-1: Bit Error Rate Test Results

Figure A-1 shows the BER and figure A-2 shows the Block Error Rate (BLER) test results for 50 and 10 maximum iterations from an FPGA implementation of the baselined (8176, 7156) code. Note that for both cases the difference between simulations and hardware tests was 0.1 dB or less.

The encoder data rate was limited to 2 × system clock while the decoder operated at 14 × system clock / number of iterations. For testing, the system clock was set to 100 MHz, so for 10 iterations, the decoder operated at 140 Mbps. Although the shortened (8160, 7136) was not tested, it is reasonable to say that the baselined (8176, 7156) and the shortened (8160, 7136) codes will have similar results.

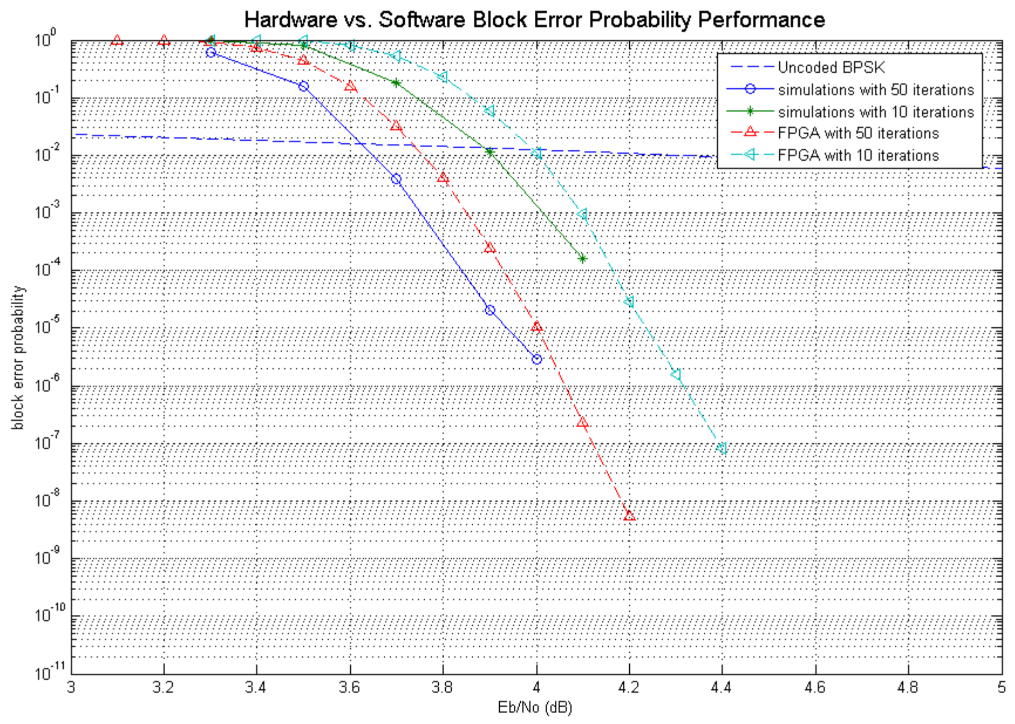


Figure A-2: Block Error Rate Test Results

ANNEX B

ANNEX TO SECTION 3, LOW DENSITY PARITY CHECK CODE FAMILY OPTIMIZED FOR DEEP SPACE APPLICATIONS

B1 QUASICYCLIC ENCODERS

The encoding algorithm described in section 3.4 can be appropriate for microprocessor-based encoders that operate on many bits in parallel. For a hardware bit-serial encoder, such as might be implemented in an FPGA or ASIC, matrix multiplication by a (dense) quasicyclic generator matrix G_{qc} may be more convenient.

For the codes of rate 1/2, 2/3, and 4/5, quasicyclic generator matrices G_{qc} may be constructed as follows. Let K=2 for a rate 1/2 code, K=4 for rate 2/3, and K=8 for rate 4/5, so G_{qc} will be of size $MK \times M(K+2)$.

1) Let P be the $3M$-1 $\times$ $3M$-1 submatrix of H consisting of the first $3M$-1 rows of the last $3M$-1 columns. Let Q be the $3M$-1 $\times$ MK submatrix of H consisting of the first $3M$-1 rows of the first MK columns.

2) Compute $W = (P^{-1}Q)^T$, where the arithmetic is performed modulo-2.

3) Construct the matrix $G = \left[I_{MK} \quad 0_{MK\times 11} \quad W \right]$, where I_{MK} is the $MK \times MK$ identity matrix, and $0_{MK\times 1}$ is the $MK \times 1$ all-zero column vector. This generator matrix defines the same encoder as that described in section 3.4 (when the last M columns are deleted, corresponding to the untransmitted vector p^2), and it is not quasicyclic. Note that the all-zero column in G defines a code symbol that is always zero, as does the initialization p_0^1=0 in Step 4) of section 3.4.

4) Let G_{qc} be the same size as G, with rows 1, 1+M/4, 1+2M/4, ..., 1+MK-M/4 identical to the corresponding rows of G. These are the first rows of circulants of size M/4; the remaining rows of G_{qc} complete the circulants. In implementation, the circulants in the last M columns are deleted, for these code symbols are punctured.

For example, the generator matrix G_{qc} for the (n=2048,k=1024) code is shown in figure B-1, including the last M columns for the punctured symbols.

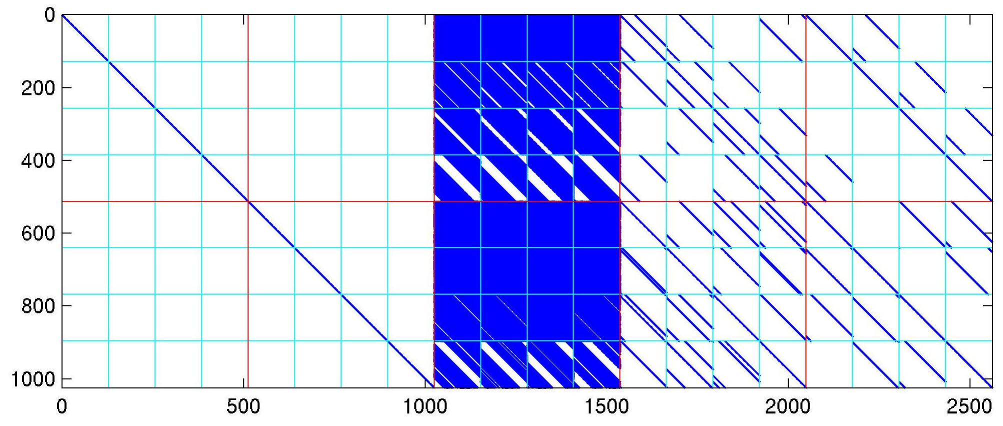


Figure B-1: A Quasicyclic Generator Matrix for the (2048,1024) Code

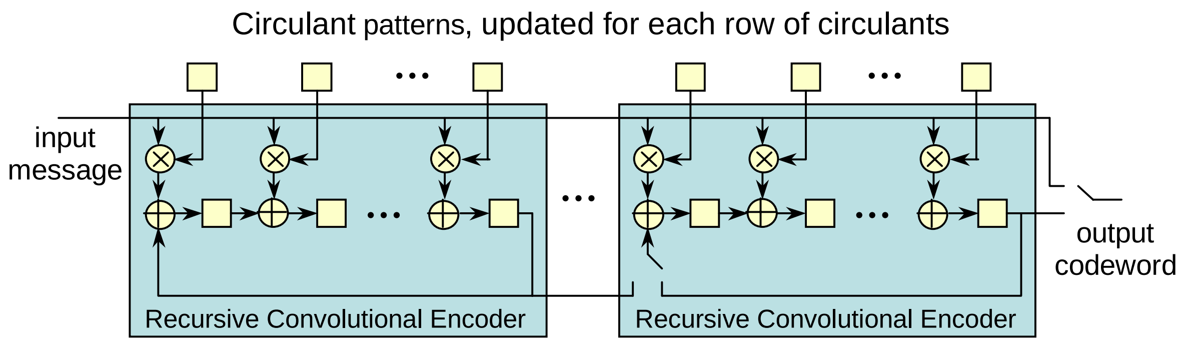

Circulant patterns, updated for each row of circulants
input message
output codeword
Recursive Convolutional Encoder
Recursive Convolutional Encoder

Figure B-2: A Quasicyclic Encoder Using Feedback Shift Registers

Encoding of message m requires computing mG_{qc}. Because G_{qc} is quasicyclic, this can be performed in an efficient bit-serial manner using $4n/M$ linear feedback shift registers, each of length $M/4$, as shown in figure B-2. Initially, the binary pattern from the first row of circulants is placed in the top row of small boxes in the figure. With the switches set as drawn, the k message bits are fed through the encoder one at a time, and the registers are updated and shifted once per bit. After each set of $M/4$ message bits are processed, the circulant patterns are updated for the next row of circulants. Then the switches are changed and the contents of the registers are read out sequentially as the parity portion of the codeword.

B2 PERFORMANCE

Figure B-3 shows the frame error rates (dashed) and symbol error rates (solid) for the short blocklength members of the code family. From left to right, these three codes have parameters (n=2048, k=1024) rate 1/2, (n=1536, k=1024) rate 2/3, and (n=1280,k=1024) rate 4/5. Similarly, figure B-4 shows performance curves for the midsize blocklength codes with parameters (n=8192, k=4096) rate 1/2, (n=6144, k=4096) rate 2/3, and (n=5120, k=4096) rate 4/5. Figure B-5 shows performance curves for the long blocklength codes with parameters (n=32768, k=16384) rate 1/2, (n=24576, k=16384) rate 2/3, and (n=20480, k=16384) rate 4/5.

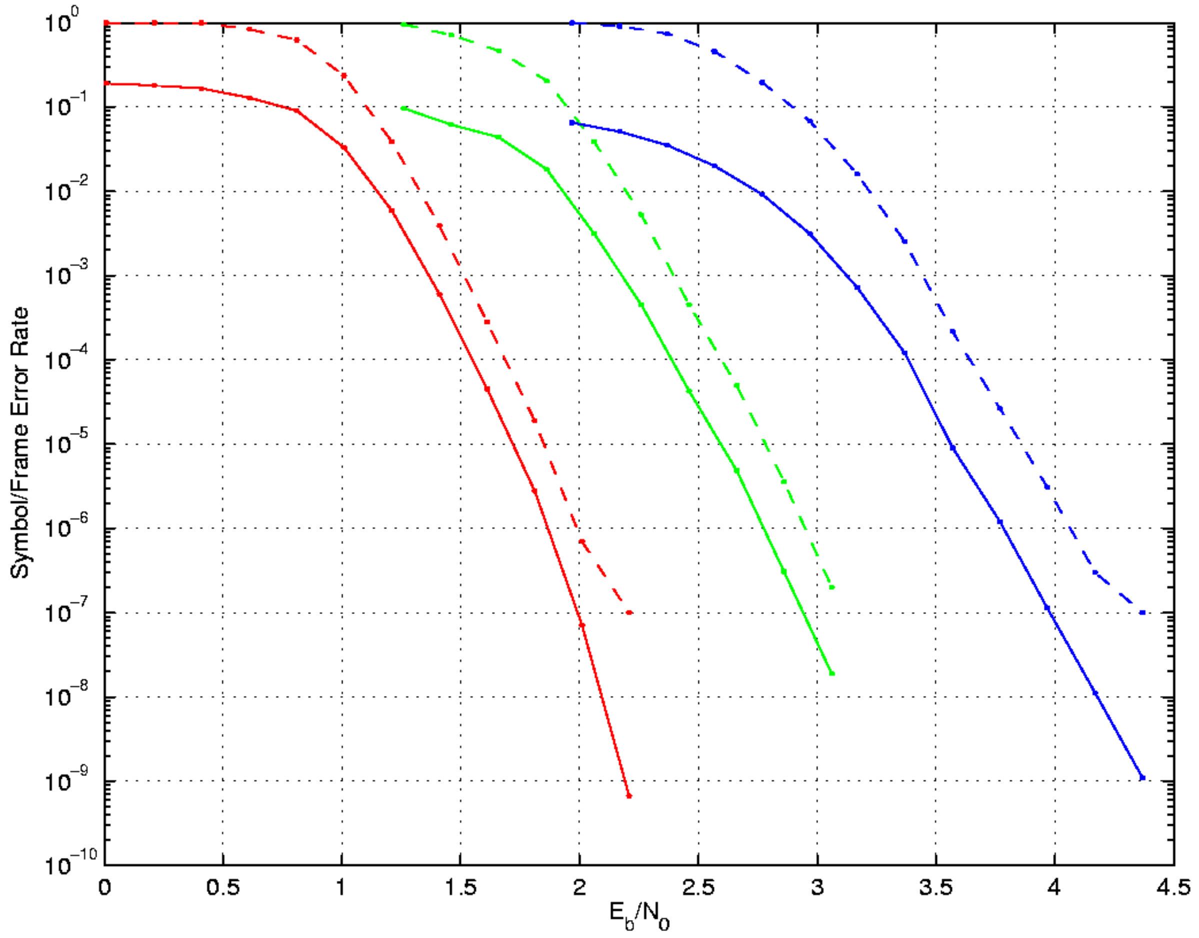


Figure B-3: Performance of Length k=1024 LDPC Codes: Rate 1/2, 2/3, 4/5 (Left to Right)

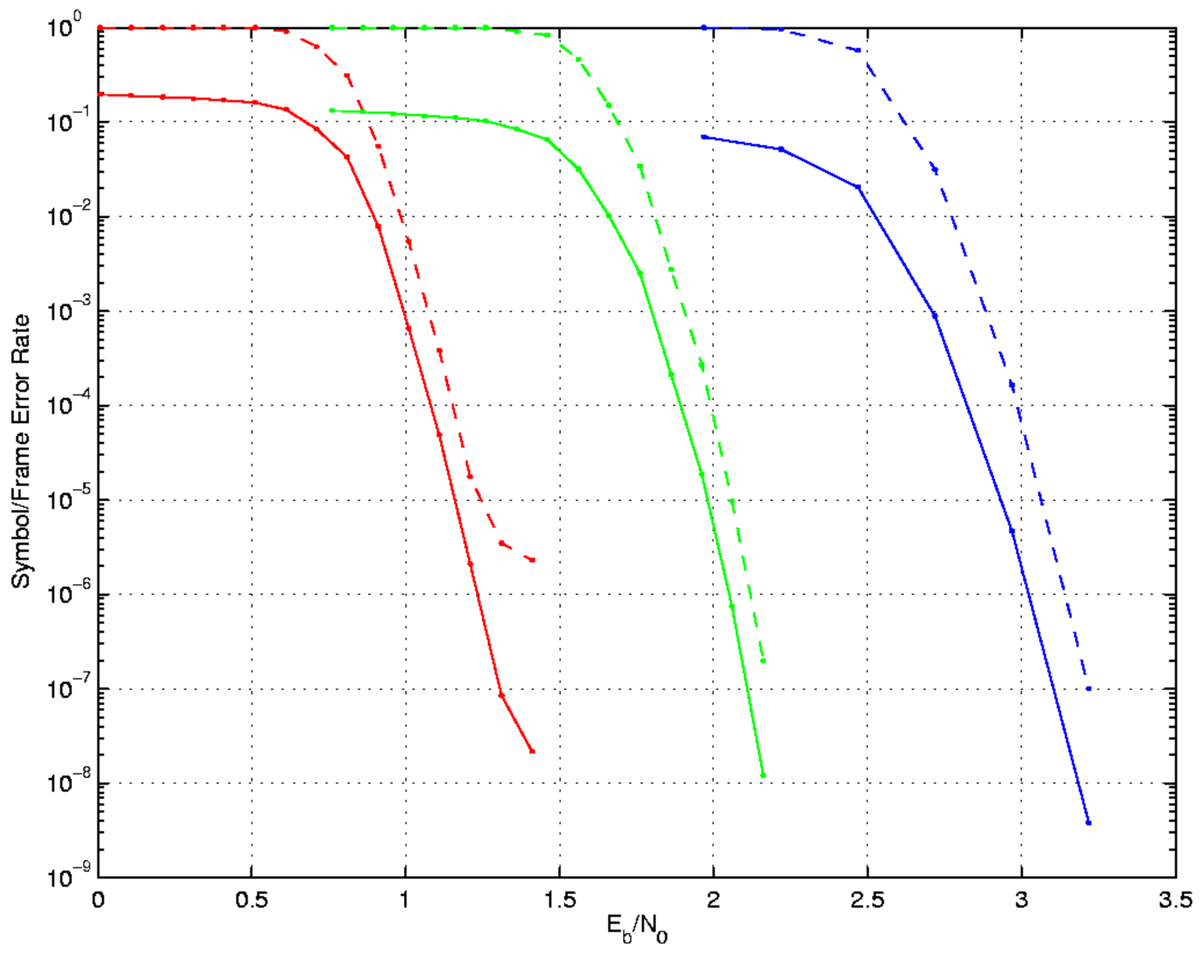


Figure B-4: Performance of Length *k*=4096 LDPC Codes: Rate 1/2, 2/3, 4/5 (Left to Right)

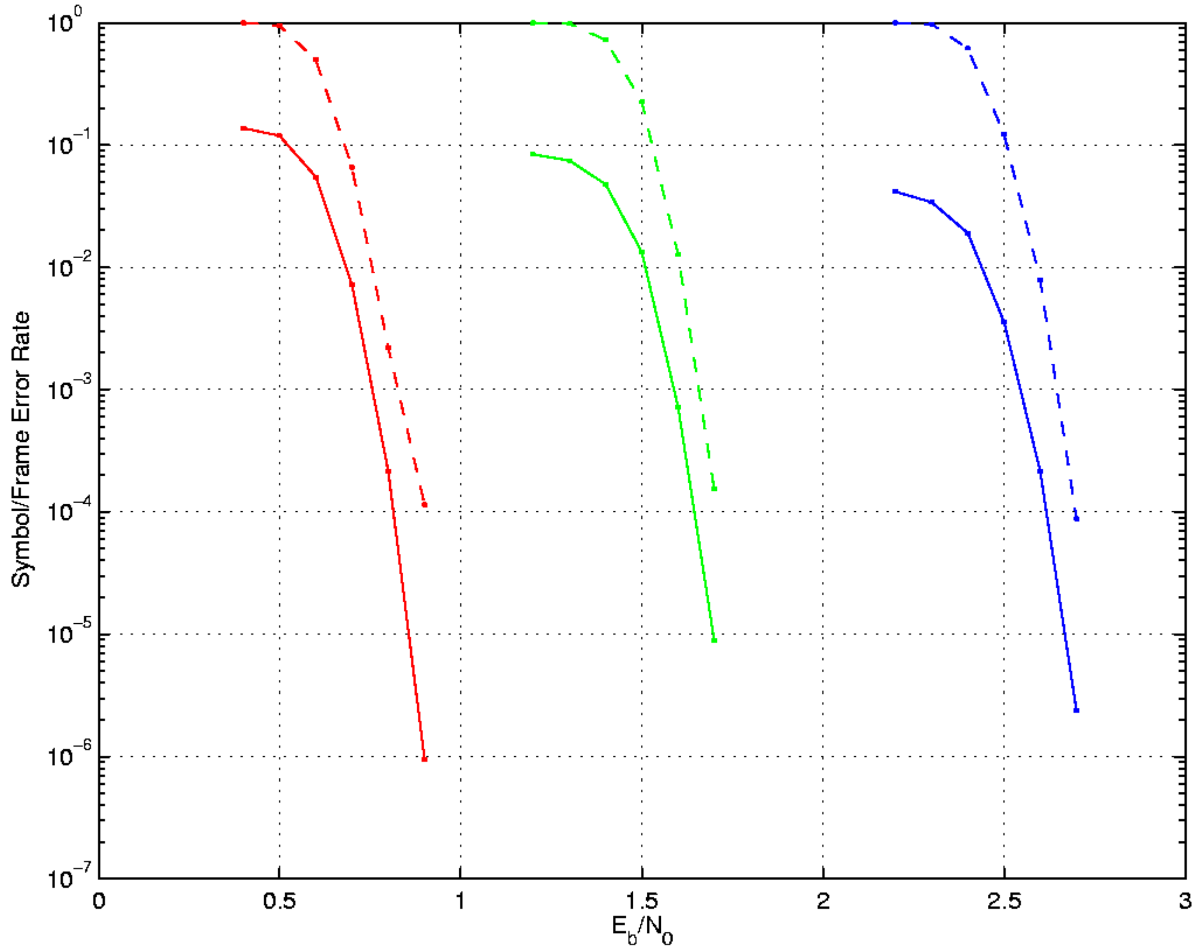


Figure B-5: Performance of Length *k*=16384 LDPC Codes: Rate 1/2, 2/3, 4/5 (Left to Right)

Performance curves for the codes with *k*=1024 and *k*=4096 were determined by hardware simulation on a Xilinx Virtex-II FPGA (see reference [C2.5]); performance for the *k*=16384 codes are from software simulations. In each case, a large maximum number of iterations was allowed, and a stopping rule was used so the average number of iterations required remained small.

It is well known that error floors can be caused either by the structure of the code, or by implementation details of the decoder. Except for the (*n*=1280,*k*=1024) rate 4/5 code, experiments indicate that each code has a minimum distance of at least 30. The error floors that appear in a few cases may be due in part to characteristics of the decoders used.

ANNEX C

INFORMATIVE REFERENCES

C1 INFORMATIVE REFERENCES FOR SECTION 2, LOW DENSITY PARITY CHECK CODE OPTIMIZED FOR NEAR EARTH APPLICATIONS

[C1.1] D. J. C. MacKay and R. M. Neal. “Near Shannon Limit Performance of Low Density Parity Check Codes.” *Electro. Lett.* 32 (August 1996): 1645-1646.

[C1.2] R. G. Gallager. “Low Density Parity Check Codes.” *IRE Trans. Inform. Theory* IT-8 (January 1962): 21-28.

[C1.3] T. Richardson and R. Urbanke. “Design of Capacity-Approaching Low Density Parity Check Codes.” *IEEE Trans. Inform. Theory* 47 (February 2001): 619-637.

[C1.4] Y. Kou, S. Lin, and M. P. C. Fossorier. “Low-Density Parity-Check Codes Based on Finite Geometries: A Rediscovery and New Results.” *IEEE Trans. Information Theory* 47 (November 2001): 2711-2736.

[C1.5] W. Fong, “White Paper for Low Density Parity Check (LDPC) Codes for CCSDS Channel Coding Blue Book.” CCSDS P1B Channel Coding Meeting, Houston, TX, October 2002.

[C1.6] Z. Li, et al. “Efficient Encoding of Quasi-Cyclic Low Density Parity Check Codes.” *IEEE Transactions on Communication* [to be published].

[C1.7] J. Heo. “Analysis of Scaling Soft Information on Low Density Parity Check Code.” *Electro. Lett.* 39 (January 2003): 219-221.

[C1.8] S. Lin and D. Costello, Jr. *Error Control Coding*. 2nd ed. New Jersey: Pearson Prentice Hall, 2004.

C2 INFORMATIVE REFERENCES FOR SECTION 3, LOW DENSITY PARITY CHECK CODE FAMILY OPTIMIZED FOR DEEP SPACE APPLICATIONS

[C2.1] A. Abbasfar, D. Divsalar, and K. Yao. “Accumulate Repeat Accumulate Codes.” In *Proceedings of GLOBECOM '04 (Dallas, Texas)*, 1-509–1-513. Piscataway, NJ: IEEE, 29 Nov.-3 Dec. 2004.

[C2.2] K. Andrews, et al. “Design of Low-Density Parity-Check (LDPC) Codes for Deep Space Applications.” *IPN Progress Report* 42-159 (November 2004). <http://tmo.jpl.nasa.gov/progress_report/42-159/159K.pdf>

[C2.3] K. Andrews, S. Dolinar, and J. Thorpe. “Encoders for Block-Circulant LDPC Codes.” In *Proceedings of the IEEE International Symposium on Information Theory (Adelaide, Australia)*, 2300-2304. Piscataway, NJ: IEEE, September 2005.

[C2.4] S. Dolinar, D. Divsalar, and F. Pollara. “Code Performance as a Function of Block Size.” *TMO Progress Report* 42-133 (January-March 1998). <http://tmo.jpl.nasa.gov/progress_report/42-159/159K.pdf>

[C2.5] C. Jones, et al. “Approximate-MIN* Constraint Node Updating for LDPC Code Decoding.” In *Proceedings of MILCOM 2003 (Boston, Massachusetts)*, 1-157-1-162. Piscataway, NJ: IEEE, October 2003.

[C2.6] J. Lee and J. Thorpe. “Memory-Efficient Decoding of LDPC Codes.” In Proceedings of the IEEE International Symposium on Information Theory (Adelaide, Australia), 459-463. Piscataway, NJ: IEEE, September 2005.

[C2.7] *TM Space Data Link Protocol*. Recommendation for Space Data System Standards, CCSDS 132.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, September 2003.

[C2.8] *AOS Space Data Link Protocol*. Recommendation for Space Data System Standards, CCSDS 732.0-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, July 2006.

NOTE – Normative references are contained in 1.4.