Green Book material on LDPC Codes extracted from:


October 15, 2008
1 LOW DENSITY PARITY CHECK CODE OPTIMIZED FOR NEAR EARTH APPLICATIONS

1.1 OVERVIEW

1.1.1 BACKGROUND

The mid-1990s were highlighted by the rediscovery of Low Density Parity Check codes (LDPC) 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 LDPC is the discovery of an iterative decoding algorithm, now called Belief Propagation (BP) decoding, which offers near-optimum performance for large linear LDPC at a manageable complexity. LDPC 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 LDPC, has come to be regarded as a special class of LDPC. Recent advances in LDPC code construction have resulted in the development of new codes with (arguably) improved performance over Gallager LDPC. One class of these codes, irregular LDPC (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 LDPC 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 LDPC (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.

1.1.2 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 1-1. 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.

![Circulant Matrix](image)

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

An example of a quasi-cyclic parity check matrix is shown in figure 1-2. 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.

![Quasi-Cyclic Matrix](image)

**Figure 1-2: 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.
1.2 BASE (8176,7156) LDPC CODE

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 $2 \times 2$ or four.

A scatter chart of the parity check matrix is shown in figure Figure 1-3: Scatter Chart of Parity Check Matrix where every ‘1’ bit in the matrix is represented by a point.

![Figure 1-3: Scatter Chart of Parity Check Matrix](image)

1.3 ENCODING

The encoder can be designed using the method given in reference [C1.6].
There are many ways to design the encoder based on the generator matrix in figure Error! Reference source not found.. These schemes have complexities that are proportional to the length of the codeword or parity check bits (see reference [C1.6]).

1.4 SHORTENED (8160, 7136) CODE

Using the generator matrix given by figure Error! Reference source not found., an encoder can be implemented using a circuit described in reference [C1.6].
2 LOW DENSITY PARITY CHECK CODE FAMILY OPTIMIZED FOR DEEP SPACE APPLICATIONS

2.1 OVERVIEW

2.1.1 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 [Error! Reference source not found.]), and were designed according to a list of requirements and evaluation criteria that reflect the needs of spacecraft applications (see references [C2.1] and [C2.2]).

- Requirements
  - **Code rates**: The family shall include codes of rate ≈ 0.5 and ≈ 0.8
  - **Block lengths**: The family shall cover \( k \approx 1000 \) to \( k \approx 16000 \) information bits spaced by multiples of \( \approx 4 \)
  - **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
  - **Description 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 [C2.6]). 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 $k=\{1024,4096,\infty\}$ are about uniformly spaced by 0.6 dB on the sphere-packing bound at WER=$10^{-8}$, and reducing the last value from $\infty$ 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.

The selected codes are systematic. A low-complexity encoding method is described (see reference [C2.5]). The parity check matrices have plenty of structure to facilitate decoder implementation (see reference [C2.8]). 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 [C2.3]). Naming conventions proposed in references [C2.1] and [C2.2] have designated these codes Accumulate-Repeat-4-Jagged-Accumulate (AR4JA).

2.1.2 DESCRIPTION OF THE CODE

In contrast to turbo codes, LDPC codes offer the prospect of much higher decoding speeds via highly parallelized decoder structures. Currently reference [Error! Reference source not found.] includes turbo codes of rates $1/2$ and lower and a convolutional code of 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 [Error! Reference source not found.] are comparable.
2.2 SPECIFICATION

The LDPC code rates $r$ are exactly as indicated in table Error! Reference source not found., unlike the case of turbo codes for which the precise code rates are slightly lower than the corresponding nominal rates due to termination bits.

An $H$ matrix for rate-3/4 is also specified since this rate naturally occurs via the column extension required to achieve rate-4/5.

The parity check matrix descriptions in conjunction with table 3-3 and table 3-4 describe 28 codes, one for each rate $r = \{1/2, 2/3, 3/4, 4/5\}$ and $M$ in the set above. Of these 28 codes, 9 are selected based on criteria provided earlier in this document (for instance spacing of $\sim$1dB for different rates and 0.6 dB for different lengths).

For example, the parity matrix for the $(n = 1280, k = 1024)$ rate-4/5 code is shown below with blue lines representing each non-zero circulant entry, and structure indicated by gridlines. Minor gridlines are spaced at intervals $m$ and major gridlines (not shown) at $M = 4m$.

![Figure 2-1: An $H$ Matrix for the $(n = 1280, k = 1024)$ Rate 4/5 Code](image)

NOTE – For this code $M=128$ and columns 1281 through 1408 are punctured.

2.3 ENCODING

1) Construct the matrix $G = [I_{MK} \ W]$, where $I_{MK}$ is the $MK \times MK$ identity matrix, and $W$ is a dense matrix of circulants of size $MK \times M(N-K)$. Note that we can define $N$ such that code block size is $n_{unpunc} = MN$. For AR4JA codes, $n_{unpunc}$ also equals
We can then therefore express the dimension of $W$ as $MK \times 3M$, or $MK \times 2M$ in the case where punctured variables are omitted from the generator matrix.

The matrix $G$ is block-circulant and is composed of sets of superimposed size $m = M/4$ circulants. These superimposed block-circulants have a compact description based on the summation of monomials of the form $x^i (i \in \{0, \ldots, m-1\})$ where $x^i$ represents a circulant matrix with a one in the $i$th column of the first row (row $j = 0$), the $i+1$th column of the second row (row $j = 1$), and the $((i+j) \mod m)$th row of the $j$th column. Note that block circulant $x^0$ is the identity matrix.

![Figure 2-2: A Quasi-Cyclic Encoder Using Feedback Shift Registers](image)

**Table 2-1:** Description of Rate 4/5 $k = 1024$ Quasi-Cyclic Encoding. Degree-32 Parity Generation Polynomials Expressed As Hexadecimal Characters (Punctured Columns are Omitted)

---

1 We distinguish $n_{unpunc}$, the total number of codesymbols in the parity-check matrix of an AR4JA code, from $n$, the number of transmitted codesymbols. Note that $n = M(K+2)$. 

Page 2-4 March 2008
As an example, table 3-5 lists every $M/4$'th row of $W$ (the first row of each set of circulants) for the generator matrix of the $(n = 1280, k = 1024)$ rate 4/5 code. For this code $(M = 128, K = 8)$ therefore block-circulants have size $M/4 = m = 32$. The table dimensions are $MK/m = 1024/32 = 32$ rows and $2M/m = 8$ columns of summed monomials (polynomials) of degree up to $m-1 = 31$ (Note that the last $M$ columns for the punctured symbols are not included). Rather than writing long polynomials into the table, we have instead used hexadecimal notation. For example the (1,1) element (0x8AD371E6) denotes polynomial $x^{31} + x^{27} + x^{25} + x^{23} + x^{22} + x^{20} + x^{17} + x^{16} + x^{14} + x^{13} + x^{12} + x^{8} + x^{7} + x^{6} + x^{5} + x^{4} + x^{1}$. A procedure for deriving AR4JA encoding matrices can be found in annex B.

Encoding of message $m$ requires computing $mG$. Because $G$ is block-circulant, this can be performed in an efficient bit-serial manner using $2M/m = 8$ (for all rates) linear feedback shift registers, each of length $M = M/4$, as shown in figure 2-2. Initially, the binary pattern
from the first row of circulants (or first row of table 2-1) is placed in the shift registers. Correct endianness occurs if higher order monomials (leftmost hexadecimal digits) are placed closer to the output than the input of each shift register. Also, in terms of $mG$, the first bit to be encoded by the circuit represents the bit in the leftmost element of row vector $m$. After $m$ bit arrivals (and cyclic shifts) then next row of circulants (second row of table 2-1) is loaded. Encoding is complete after $MK/m$ rows of circulants have been loaded. Each requires $m$ clock cycles to process for a total of $k = MK$ clock cycles to compute the parity for one codeword. Many architectural alternatives are possible. The main benefits of the architecture in figure 2-2 are conceptual simplicity and relatively high throughput ($n$ codeword bits are computed in $k$ clock cycles).

A1 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 A2 summarizes the test results.

**A2 FPGA TEST RESULTS**

![Hardware vs. Software Bit Error Probability Performance](image)

**Figure B-3: Bit Error Rate Test Results**

Figure B-3 shows the BER and figure B-4 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 $2 \times$ system clock while the decoder operated at $14 \times$ 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 B-4: Block Error Rate Test Results
ANNEX B

ANNEX TO SECTION 2,
LOW DENSITY PARITY CHECK CODE FAMILY
OPTIMIZED FOR DEEP SPACE APPLICATIONS

B1 COMPUTATION OF DENSE GENERATOR SUBMATRIX W

To clarify the procedure for computing $W$ (from section 2.3), we describe a step-by-step method for finding the generator matrix for a rate 1/2 AR4JA LDPC code with very short block length ($k = 32$). Consider the protograph of the rate 1/2 AR4JA LDPC code shown in figure B-1.

Expand the protograph by a factor of 4 to remove parallel edges. Assign circulant permutations to edges on the expanded graph. The expanded protograph has the following $H$ matrix.

$$H = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & x^1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & x^0 & x^0 & x^3 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & x^2 & 0 & x^3 & x^1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x^2 & x^2 & 0 & x^2 \\
1 & 0 & 0 & 0 & 0 & 0 & x^3 & x^2 & 0 & 0 & 0 & 0 & x^3 & x^3 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & x^0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & x^3 & x^3 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & x^3 & x^0 & 0 & 0 & 0 & 0 & 0 & 0 & x^3 & x^3 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & x^0 & x^2 & 0 & 0 & 0 & 0 & x^1 & 0 & 0 & x^2 & 0 & 0 & 0 & 1
\end{bmatrix}$$

Figure B-1: Protograph for Rate 1/2 AR4JA LDPC Code

Expand the protograph by a factor of 4 to remove parallel edges. Assign circulant permutations to edges on the expanded graph. The expanded protograph has the following $H$ matrix.
The first 4 rows correspond to check node number 1 in figure B-1, the second 4 rows and the last 4 rows correspond to check nodes 2 and 3, respectively. The first 4 columns correspond to variable node number 1 in figure B-1. The subsequent 4 groups of 4 columns correspond to variable node numbers 2, 3, 4 and 5 respectively in figure B-1. For this example $M = 16$ and the $m \times m$ circulant permutations have size $m = 4$. Each nonzero entry $x_i^j$ in the parity check matrix $H$ represents a circulant permutation (an $m \times m$ identity matrix where each row is circularly shifted to the right by $i$).

All operations are over the ring of polynomials with coefficients in GF(2) modulo $x^m + 1$.

Step 1: Denote the first 8 columns of $H$ as $Q$, and the last 12 columns as $P$:

$$Q = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & x^3 & x^2 \\
0 & 1 & 0 & 0 & x^0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & x^3 & x^0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & x^0 & x^2 & 0
\end{bmatrix}$$

$$P = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & x^1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & x^2 & x^2 & x^2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & x^0 & x^0 & x^0 & x^3 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & x^2 & 0 & x^3 & x^1 & x^1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & x^2 & x^2 & 0 & x^2 & x^2 \\
0 & 0 & 0 & 0 & x^3 & x^3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & x^3 & x^3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & x^3 & x^3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & x^1 & 0 & 0 & x^2 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

Step 2: Find the cofactor matrix of $P$ and denote it $P_c$. This is the transpose of the matrix composed of determinants of all sub-matrices of $P$ found by eliminating each row/column pair and forming a matrix from each set of remaining elements.

Step 3: Find the determinant of $P$, denote it by polynomial $d(x)$.

In this example $d(x) = x + x^2 + x^3$.

Step 4: Find the inverse of polynomial $d(x)$ using Euclid’s algorithm.
In this example $d^{\text{i}}(x) = x + x^2 + x^3$

Step 5: Multiply to find $P_i = d^{\text{i}}(x)P_c$

Step 6: Multiply to find $W_2 = (P_iQ)^T$

Step 7: Find the reciprocal polynomial of each entry of $W_2$ and denote the result $W$. This step serves to swap row/column ordering of all circulants in $W_2$.

Step 8: We then obtain the generator matrix $G = [I_{MK} \ W]$

$$W_P = \begin{bmatrix} 1 + x^2 & x^2 + x^3 & 1 + x^3 & x^3 & 1 + x + x^2 & 0 \\ 1 + x^3 & 1 + x & 1 + x^2 & 0 & 1 + x^2 + x^3 & 1 + x^2 \\ x^2 + x^3 & x + x^2 & x + x^3 & 0 & 1 + x + x^3 & x^3 \\ 1 + x & 0 & x^2 + x^3 & 1 + x^2 & 1 + x^2 + x^3 & x^3 \\ x + x^2 & 1 + x & x^3 & 1 + x + x^3 & 1 + x + x^2 & x^3 \\ 0 & 0 & 1 + x + x^2 + x^3 & x^3 & 1 + x + x^3 & x + x^3 \end{bmatrix}$$

The most compact representation of an encoding omits punctured columns as well as the leading $I_{MK}$ identity matrix. We denote such a representation as $W_p$ and express the $W_p$ embedded within the $G$ of Step 8 using hexadecimal digits as:

$$W_P = \begin{bmatrix} 5 & C & 1 & 2 & 9 & 8 & 7 & 0 \\ D & 3 & 5 & 8 & 6 & 0 & 7 & 4 \\ D & E & 6 & 0 & D & 5 & A & D \\ 3 & 2 & 7 & 6 & E & 8 & F & 6 \\ 0 & C & 0 & A & 9 & 8 & 7 & 2 \\ C & 3 & 5 & 3 & 4 & 6 & D & 8 \\ 6 & A & 9 & C & D & B & 9 & E \\ C & 0 & 0 & F & 8 & B & 2 & 6 \end{bmatrix}$$

**B2 PERFORMANCE**

Figure B-2 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-3 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-4 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. Finally, figure B-5 provides a composite plot of the performance of all nine AR4JA codes.
Figure B-2: Performance of Length $k=1024$ LDPC Codes: Rate 1/2, 2/3, 4/5 (Left to Right)
Figure B-3: Performance of Length $k=4096$ LDPC Codes: Rate 1/2, 2/3, 4/5 (Left to Right)
Figure B-4: 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 [C2.7]); performance for the $k = 16384$ codes are from software simulations (except rate 1/2 performance which was also found via hardware simulation). 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. The error floor that appears above FER $10^{-7}$ for the $(n=1280, k=1024)$ rate 4/5 code may be due in part to characteristics of the decoders used.
ANNEX C

INFORMATIVE REFERENCES

C1 INFORMATIVE REFERENCES FOR SECTION 1, LOW DENSITY PARITY CHECK CODE OPTIMIZED FOR NEAR EARTH APPLICATIONS


C2 INFORMATIVE REFERENCES FOR SECTION 2, LOW DENSITY PARITY CHECK CODE FAMILY OPTIMIZED FOR DEEP SPACE APPLICATIONS


NOTE – Normative references are contained in Error! Reference source not found.