



### *Change Log*

<b>Rev</b>	<b>Issue Date</b>	<b>Paragraphs Affected</b>	<b>Change Summary</b>
Initial	9/15/2000	All	
Chg 1	03/31/2004	Figures 3, 17	Corrects third generator polynomial hexadecimal representation in Figure 3. Removes implication in Figure 17 that Turbo code performance can be extrapolated below FER= $10^{-5}$

### **Note to Readers**

There are two sets of document histories in the 810-005 document that are reflected in the header at the top of the page. First, the entire document is periodically released as a revision when major changes affect a majority of the modules. For example, this module is part of 810-005, Revision E. Second, the individual modules also change, starting as an initial issue that has no revision letter. When a module is changed, a change letter is appended to the module number on the second line of the header and a summary of the changes is entered in the module's change log.

## *Contents*

<u>Paragraph</u>	<u>Page</u>
1 Introduction .....	5
1.1 Purpose .....	5
1.2 Scope .....	5
2 General Information .....	5
2.1 Convolutional Codes .....	6
2.2 Decoding of Convolutional Codes .....	6
2.2.1 B2MCD- and B3MCD-Supported Encoder Connections .....	6
2.2.2 Galileo Telemetry Decoder .....	7
2.2.3 Telemetry System Performance with Viterbi Decoding .....	13
2.2.4 MCD Synchronization Time .....	13
2.2.4.1 B2MCD Node Synchronization .....	13
2.2.4.2 B3MCD Node Synchronization .....	14
2.2.5 $E_b/N_0$ Estimation .....	15
2.3 Telemetry Data Formats .....	15
2.4 Frame Synchronization of Telemetry .....	16
2.4.1 Frame Synchronizer Operation .....	16
2.4.2. Frame Synchronizer Performance .....	18
2.5 Decoding of Concatenated Convolutional and Reed–Solomon Code .....	18
2.5.1. Interleaving .....	21
2.5.2. Reed–Solomon Encoder .....	21
2.5.3. Virtual Fill .....	23
2.5.4. Frame Error Rate of Concatenated Codes .....	23
2.5.5 Variable Redundancy and Feedback Concatenated Decoding .....	23
3 Proposed Capabilities .....	28
3.1 Parallel Concatenated Convolutional Codes (Turbo Codes) .....	28
3.1.1 Turbo Code Encoder .....	28
3.1.2 Permutations .....	30
3.1.3 Synchronization of Turbo Codes .....	30
3.1.4 Turbo Code Decoder Implementation .....	30
3.1.5 Turbo Code Latency .....	31
3.1.6 Turbo Code Error Floor .....	31
3.1.7 Selection and Performance of Turbo Codes .....	31
Appendix A, Reference .....	34

## *Illustrations*

<u>Figure</u>	<u>Page</u>
1. Baseline Performance for Several Convolutional Codes Showing Effect of Symbol Quantization for B2MCD ( $Q = 3$ ) and B3MCD ( $Q = 8$ ) .....	8
2. Rate-1/2 Connection Vector Schematics, Constraint Length = 7.....	9
3. Cassini (15, 1/6) Convolutional Encoder Connection Diagram.....	10
4. Concatenated (11,1/2) and (7,1/2) Galileo Convolutional Encoder Connection Diagram.....	12
5. Equivalent (14,1/4) Galileo Convolutional Encoder Connection Diagram.....	12
6. Telemetry Data Formats .....	15
7. Probability of Frame Acquisition (in Four Frames) as a Function of $BET_S$ Threshold and the Number of Check Frames ( $V$ ) with MED Enabled.....	19
8. Probability of Frame Acquisition (in Four Frames) Versus Bit SNR for Several Values of $BET_S$ . Frame Length = 5120, $V=2$ , MED Enabled, ASM = 32 bits, Coded (7,1/2) Data .....	20
9. Probability of Frame Out of Lock for the Number of Flywheels ( $F$ ) Set at One or Two Versus Bit SNR for Various Values of Bit Error Tolerance ( $BET_L$ ). ASM = 32 bits, Coded (7,1/2) Data .....	20
10. Reed-Solomon Symbol Arrangement for Interleave Factor ( $I$ ) of 5.....	22
11. Berlekamp Architecture Reed–Solomon Encoder.....	24
12. Reed–Solomon Code (255,223) Performance for Standard Convolutional Inner Codes Showing the Effect of Interleave Factor ( $I$ ).....	25
13. Virtual Fill .....	26
14. Modeled Frame Error Rate for several DSN Concatenated Codes and Interleave Factors ( $I$ ) .....	27
15. CCSDS Compliant Rate=1/3 and Rate=1/6 Turbo Encoder .....	29
16. Turbo Code Structure in the Information Channel.....	31
17. Modeled Performance of DSN Turbo Codes Versus Information Bit $E_b/N_0$ .....	33

## *Tables*

<u>Table</u>	<u>Page</u>
1. Maximum- Likelihood Convolutional Decoder Characteristics.....	7
2. B2MCD and B3MCD Connection Vectors and Impulse Responses.....	11
3. Frame Synchronizer Characteristics .....	17
4. Turbo Code Characteristics.....	32

## *1 Introduction*

### *1.1 Purpose*

The purpose of this module is to describe the capabilities and performance of the telemetry decoding and frame synchronization equipment used by the Deep Space Network (DSN) in order to assist the telecommunications engineer to design compatible spacecraft equipment.

### *1.2 Scope*

The detailed discussion in this module is limited to equipment that is currently installed at the Deep Space Communications Complexes (DSCCs) and performs data decoding and frame synchronization in real time. Additional factors that affect telemetry performance such as imperfect residual or suppressed carrier synchronization (radio loss), imperfect subcarrier and symbol synchronization, and waveform distortion are discussed in module 207. Formatting of data for delivery to the customer is discussed in the companion document, 810-007.

## *2 General Information*

Most spacecraft employ data encoding to make more efficient use of the data channel. Currently, the coding used for deep space spacecraft is one of several convolutional codes with or without concatenated Reed–Solomon (RS) code. The following paragraphs discuss the performance and provide recommendations for the use of codes that can be decoded by the DSN telemetry processing equipment.

## 2.1 Convolutional Codes

Convolutional codes are used on spacecraft because they achieve significant coding gain with simple, highly reliable encoders and their decoders are of reasonable complexity. Convolutional codes are specified by their *constraint length* ( $K$ ) and *rate* ( $r$ ). Constraint length is the number of sequential input bits required to define the output symbols at any point in time. Rate is the number of symbols produced for each input bit. In general, the performance of a convolutional code increases with both  $K$  and  $r$ , but codes must be selected carefully because channel bandwidth varies directly with  $r$  and decoder complexity increases exponentially with  $K$ .

## 2.2 Decoding of Convolutional Codes

The decoding of convolutional codes is performed by one of two Maximum-Likelihood Convolutional Decoders (MCDs): the Block 2 MCD (B2MCD) and the Block 3 MCD (B3MCD) that are hardware implementations of the Viterbi decoding algorithm. The B2MCD is intended for decoding constraint length 7, rate 1/2 codes whereas the B3MCD is a general purpose decoder subject to the limitations discussed below. An additional Viterbi decoder is implemented in software for decoding low rate data from the Galileo spacecraft. The characteristics of the three decoders are provided in Table 1.

The B3MCD uses the full 8 bits of input symbol quantization provided by the Block V Receiver (BVR). This provides decoding performance superior to the B2MCD, which must map the 8-bit symbols from the BVR into 3-bit symbols. Unfortunately, there are a limited number of B3MCDs at each complex, so the link designer is encouraged to accept the reduced performance of the B2MCD whenever this is possible. The effect of 3-bit versus 8-bit input symbol quantization on the  $K = 7$ ,  $r = 1/2$  code can be seen in the ideal performance curves shown in Figure 1. This figure also shows the performance improvement that can be expected from using the longer constraint length codes.

### 2.2.1 B2MCD- and B3MCD-Supported Encoder Connections

Two encoder configurations for  $K = 7$ ,  $r = 1/2$  (7,1/2) codes are supported by the DSN. These are the NASA–DSN convention and the Consultative Committee for Space Data Systems (CCSDS) convention, also known as the NASA-Goddard Space Flight Center (GSFC) convention. The two codes utilize the same connection vectors but differ in symbol ordering. Encoder diagrams for these two codes are shown in Figure 2. The DSN (7,1/2) code was most recently used on Mars Global Surveyor, while the CCSDS code has been used on almost all Earth orbiter spacecraft. Both codes are said to be *transparent*. That is, they can be decoded before the ambiguity associated with Binary Phase Shift Keying (BPSK) modulation of the subcarrier is resolved. The result of decoding a transparent code with every input symbol inverted will be an output with every bit inverted.

The B3MCD presently supports two configurations of  $K = 15$ ,  $r = 1/6$  (15,1/6) codes. One of these 15,1/6 codes is used for the Cassini spacecraft and is diagrammed in Figure 3. The other 15,1/6 code was used for the Mars Pathfinder spacecraft and is not diagrammed.

Table 1. Maximum-Likelihood Convolutional Decoder Characteristics

Parameter	B2MCD	B3MCD	Galileo
Constraint Length	7	3 to 15	14
Frame Length	Variable	Variable	65,536 symbols
Code Rate	1/2	1/2 to 1/6	1/4
Connection Vectors	Fixed	Programmable	Fixed
Symbol (Input) Rate	10 Ms/s (max.)	6.6 Ms/s (max.)	640 s/s (max.)
Bit (Output) Rate	5 Mb/s (max.)	2.2 Mb/s (max.)	160 b/s 9max.)
Input Quantization	3 bits (8 levels)	8 bits	8 bits
Node Synchronization	Automatic	Automatic or External	Automatic
Frame Synchronization	External	Automatic	Automatic

The difference between the two codes is the order in which their connection vectors deliver symbols to the communications channel. Connection vectors and impulse responses for the supported codes (including both 15,1/6 codes) are provided in Table 2. The impulse responses include the effects of alternate symbol inversion, which is normally used to provide a sufficient symbol transition density to ensure adequate symbol synchronizer performance. Alternate symbol inversion can be disabled if not required by the telecommunications link.

The B3MCD has the capability to support other codes, subject to the constraint that the most significant and least significant bits of all connection vectors be equal to one. This is not a significant limitation as it is a property of all the best codes. It also includes the capability to perform input symbol inversion as part of the node synchronization process and can therefore process both transparent and non-transparent codes. It must be recognized, however, that implementation of an additional code in the B3MCD would require significant resources for implementation and performance verification.

### 2.2.2 *Galileo Telemetry Decoder*

The  $K = 15$ ,  $r = 1/4$  code that was flown on Galileo as an experiment for the X-band downlink (and not available for the S-band downlink) could not be used when the high gain antenna failed to fully deploy. The search for a code that would improve S-band telemetry performance was complicated by the fact that the standard  $K = 7$ ,  $r = 1/2$  encoder could not be bypassed in the S-band downlink. This resulted in the selection of a  $K = 11$ ,  $r = 1/2$  software encoder programmed onboard the spacecraft and placed in series with the standard encoder,

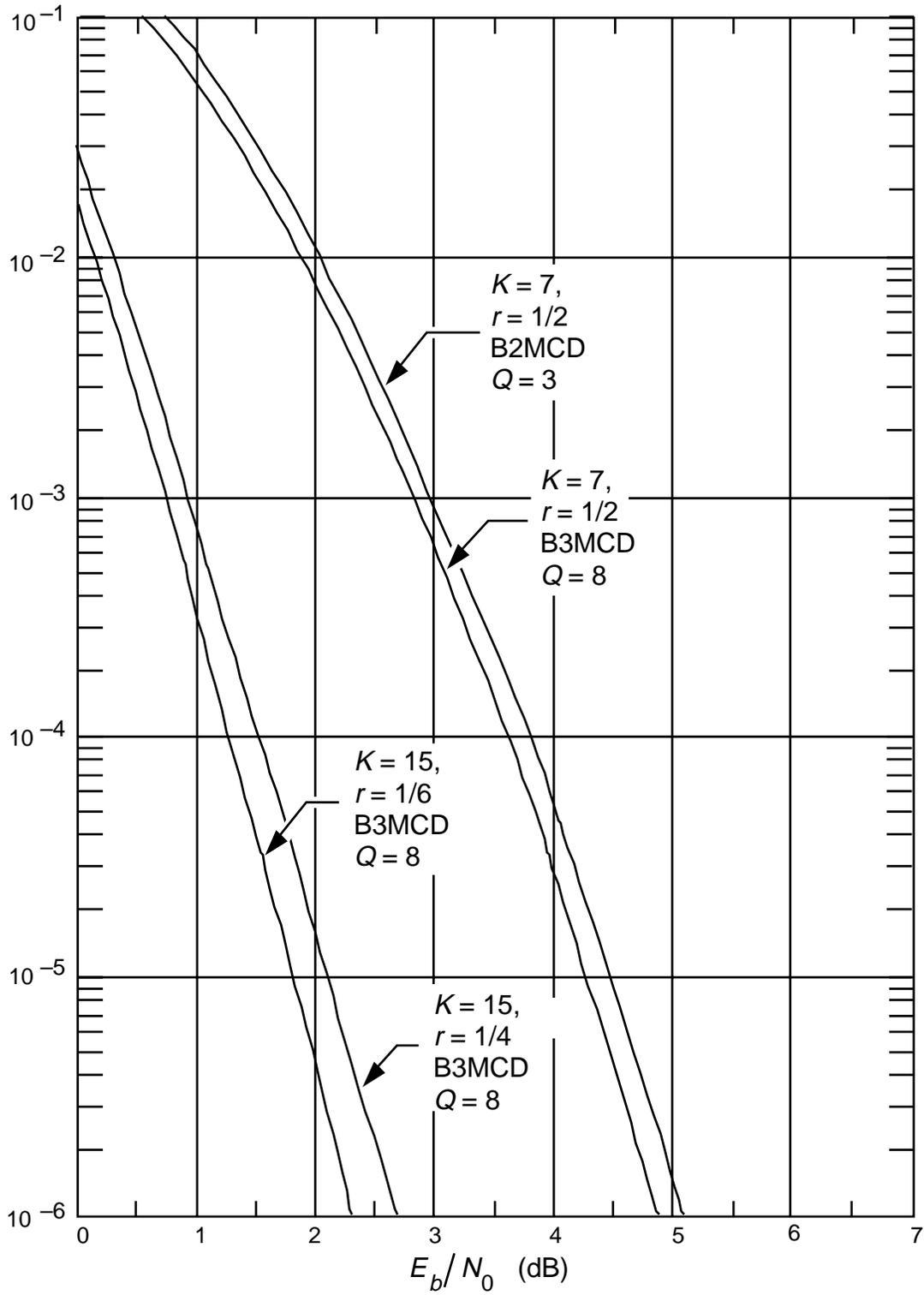
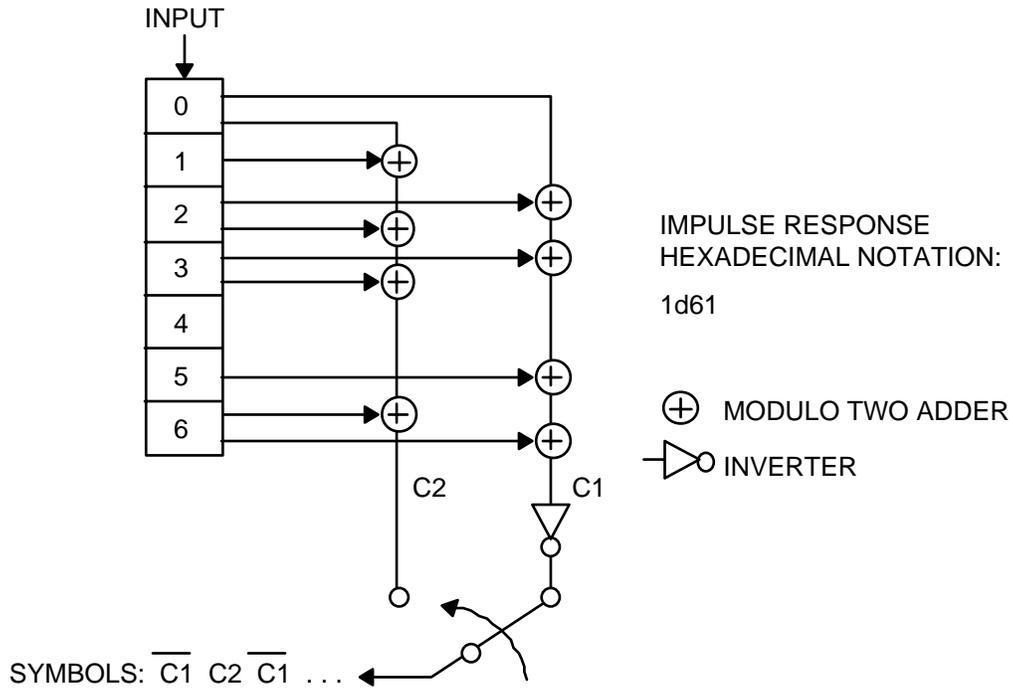
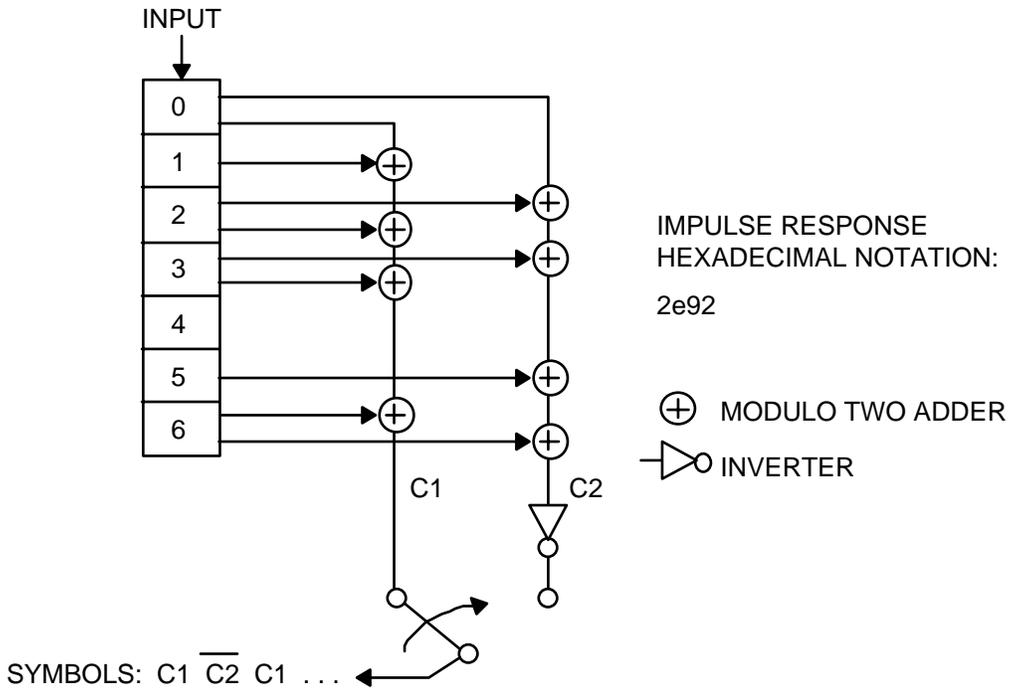


Figure 1. Baseline Performance for Several Convolutional Codes Showing Effect of Symbol Quantization for B2MCD ( $Q = 3$ ) and B3MCD ( $Q = 8$ )

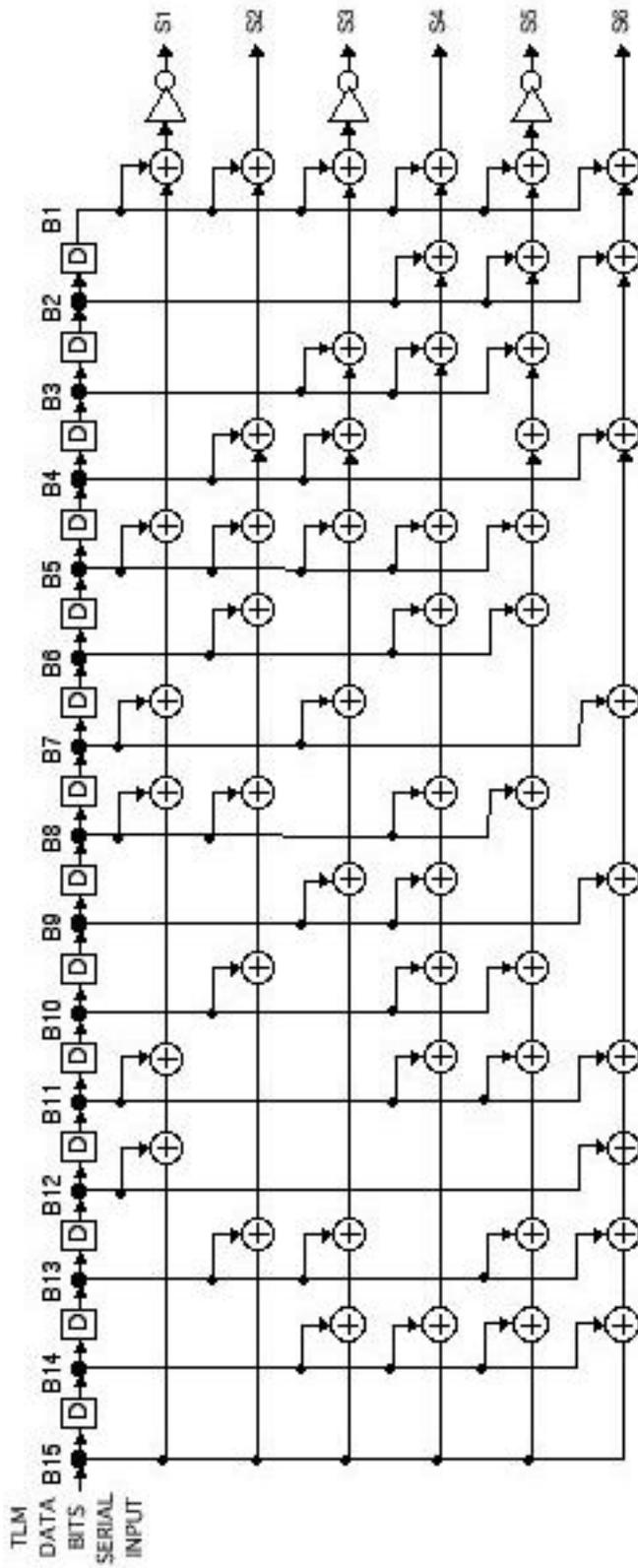


a) NASA-DSN CONVENTION



b) CCSDS (NASA-GSFC) CONVENTION

Figure 2. Rate-1/2 Connection Vector Schematics, Constraint Length = 7



CASSINI (15, 1/6) CONVOLUTIONAL CODER CONNECTION VECTORS

- 1st symbol S1 =  $\overline{4cd1}$  HEXADECIMAL:  $(B15 \oplus B12 \oplus B11 \oplus B8 \oplus B7 \oplus B5 \oplus B1)$
- 2nd symbol S2 =  $52b9$  HEXADECIMAL:  $(B15 \oplus B13 \oplus B10 \oplus B8 \oplus B6 \oplus B5 \oplus B4 \oplus B1)$
- 3rd symbol S3 =  $\overline{715d}$  HEXADECIMAL:  $(B15 \oplus B14 \oplus B13 \oplus B9 \oplus B7 \oplus B5 \oplus B4 \oplus B3 \oplus B1)$
- 4th symbol S4 =  $67b7$  HEXADECIMAL:  $(B15 \oplus B14 \oplus B11 \oplus B10 \oplus B9 \oplus B8 \oplus B6 \oplus B5 \oplus B3 \oplus B2 \oplus B1)$
- 5th symbol S5 =  $\overline{76bf}$  HEXADECIMAL:  $(B15 \oplus B14 \oplus B13 \oplus B11 \oplus B10 \oplus B8 \oplus B6 \oplus B5 \oplus B4 \oplus B3 \oplus B2 \oplus B1)$
- 6th symbol S6 =  $7d4b$  HEXADECIMAL:  $(B15 \oplus B14 \oplus B13 \oplus B12 \oplus B11 \oplus B9 \oplus B7 \oplus B4 \oplus B2 \oplus B1)$

Figure 3. Cassini (15,1/6) Convolutional Encoder Connection Diagram

Table 2. B2MCD and B3MCD Connection Vectors and Impulse Responses

Name	Constraint Length, Rate	Connection Vectors (HEX) <sup>1</sup>	Impulse Response (HEX) <sup>1</sup>
DSN Standard <sup>2</sup>	(7,1/2)	CV1 = 5b CV2 = 79	1d61
CCSDS, GSFC Standard <sup>2</sup>	(7,1/2)	CV1 = 79 CV2 = 5b	2e92
Cassini <sup>3</sup>	(15,1/6)	CV1 = 4cd1 CV2 = 52b9 CV3 = 715d CV4 = 67b7 CV5 = 76bf CV6 = 7d4b	0159, 712c, df27, 703f, 14c6, 4b55
Mars Pathfinder <sup>3</sup>	(15,1/6)	CV1 = 4cd1 CV2 = 52b9 CV3 = 67b7 CV4 = 715d CV5 = 76bf CV6 = 7d4b	0159, 7d2c, 1c27, 40fc, 14f6, 4855

1. Connection vectors and impulse responses are right justified within the hexadecimal numbers, which have been divided into groups of four for readability.
2. These codes can be supported by the B2MCD and B3MCD.
3. These codes can be supported only by the B3MCD.

producing a  $K = 14$ ,  $r = 1/4$  code. Figures 4 and 5 show the connection diagrams for the  $K = 11$ ,  $r = 1/2$  software encoder, the  $K = 7$ ,  $r = 1/2$  encoder, and the  $K = 14$ ,  $r = 1/4$  encoder that resulted from concatenating these two codes. Because the C0 connection vector has a 0 as its least significant bit, this code is not compatible with the B3MCD; however, the low data rate made it possible to construct a software decoder. This had additional advantages in that it provided flexibility to perform feedback concatenated decoding (FCD)—a scheme in which multiple passes are made through a Viterbi decoder constrained by information obtained from Reed–Solomon decoding.



### 2.2.3 *Telemetry System Performance with Viterbi Decoding*

Ideal reference curves for the (7,1/2) codes at the 3-bit quantization level of the B2MCD and the 8-bit quantization level of the B3MCD were shown in Figure 1. This figure also showed the performance of the (15,1/4) Galileo experimental convolutional code (which was not used) and the (15,1/6) Cassini convolutional code.

### 2.2.4 *MCD Synchronization Time*

The appropriate MCD is connected directly to the output of the receiver as part of the telemetry equipment configuration and will attempt to decode data as soon as it is enabled and the receiver delivers a symbol clock. Because of the possibility of false lock, the operators do not normally enable decoding until they are certain that the receiver lock is correct. This places a small but indeterminate time delay between the reported symbol lock acquisition and the start of decoding.

When the MCD is enabled, it must first obtain *node synchronization*. That is, it must determine which symbol from each group of  $r$  symbols represents the first one in the group. A different method is used by each MCD.

#### 2.2.4.1 *B2MCD Node Synchronization*

Node synchronization for the B2MCD requires only a determination of which is the first symbol out of each pair of symbols. The determination of waveform polarity and frame synchronization is left to later equipment because the rate-1/2 codes are transparent.

When enabled, the MCD begins calculating metrics for the 64 possible paths through its decoding trellis. Eleven bit periods later, an output is available, but its quality is uncertain as the amount of information obtained is insufficient to determine whether the node synchronization is correct or if the correct path has been followed. As the metrics grow, they are continuously normalized with respect to the value of the shortest path. The design of the decoder is such that a normalization rate of less than one normalization for every ten output bits is an indication that the decoder is operating correctly. A rate in excess of this is an indication that synchronization may not be correct. If acquisition has not been accomplished, the decoder is instructed to select the alternate node and start again. The process continues until the normalization rate is below the required threshold. Both the number of bit periods used to test for node synchronization and the threshold used are selectable to a maximum of 65,535 with the default values being 2,048 and 256.

MCD acquisition time is given by

$$T_{\text{MCD}} = B \times n/\text{bit-rate}$$

or

$$T_{\text{MCD}} = B \times (n + 1)/\text{bit-rate}, \text{ with equal probability}$$

where:

- $B$  = normalization monitor period (default = 2048 bits)
- $n$  = number of normalization periods that must be observed without a node synchronization change before *in-lock* is declared (selectable from 1 to 255, with a default value of 3)

The normalization rate is continuously monitored after *in-lock* is declared, and the number of consecutive periods exceeding the threshold is counted. If this number exceeds a specified value (selectable from 1 to 32,767 with a default value of 3), the MCD is declared out-of-lock and commanded to re-acquire.

Although the default values of  $B$  and  $n$  are usually used, at strong signal levels, the normalization monitor period can be decreased with a corresponding adjustment of threshold. At weak signal levels, the number of normalizations required for the best path increases and a larger number of bit periods is required to differentiate it from the other paths. For frame synchronized telemetry (discussed below) having a Bit Signal-to-Noise ratio (Bit SNR), also referred to as the Energy-per-bit to Noise ratio ( $E_b/N_0$ ), within 0.5 dB of theoretical, the continued recognition of the synchronization marker is a more reliable indication of MCD node synchronization and contrary indications from the MCD should be ignored.

#### **2.2.4.2 B3MCD Node Synchronization**

The B3MCD is capable of decoding both transparent and non-transparent codes at rates up to 1/6. This means that as many as twelve possibilities must be investigated if a rate-1/6, non-transparent code is being decoded.

The B3MCD uses a soft-symbol correlation technique to determine node synchronization. The input soft symbols to the decoder, appropriately delayed, are correlated against the re-encoded bits from the decoder output. This gives almost-known symbols to correlate against the input symbols without waiting for a frame marker. When the decoder is synchronized, the decoded output bits are a very good estimate of the true bits, and hence the re-encoded bits are a good estimate of the true symbols. If the decoder is out of synchronization, the decoded bits should be almost random, and the re-encoded bits will correlate very poorly with the input symbols.

To acquire node synchronization, the decoder must test all possible starting offsets for the received symbols. Both polarities also must be tested for non-transparent codes. As observed previously, a non-transparent code, such as the Cassini (15,1/6) code, has twelve possibilities. The B3MCD is capable of buffering the input soft-symbol stream during acquisition so that the re-examination process at low code rates does not take an excessive amount of time and does not cause a loss of data.

The B3MCD requires 1000 bit periods to determine node synchronization. After synchronization is declared, it correlates 3000 bit segments of the input data stream against the re-encoded output and compares the result against a calculated threshold. The result of this approach is to cause declaration of loss of lock to be delayed from 3000 to 6000 bit times after the actual event. This time may be significant at low bit rates.

### 2.2.5 $E_b/N_0$ Estimation

Both MCDs use the technique of re-encoding the output bits and comparing them to a suitably delayed version of the input symbol stream to provide an accurate count of channel symbol errors. This is possible because the probability of the decoder falsely decoding a bit is at least two orders of magnitude less than the the probability of a channel symbol error. This symbol error count is used to provide an estimate of the  $E_b/N_0$  with an accuracy of 0.1 dB.

## 2.3 Telemetry Data Formats

The DSN Telemetry System converts Non-Return to Zero-Mark (NRZ-M) and Non-Return to Zero-Space (NRZ-S) differentially encoded telemetry data into Non-Return to Zero-Level (NRZ-L) format subsequent to convolutional decoding. The relationship between these data formats is illustrated in Figure 6.

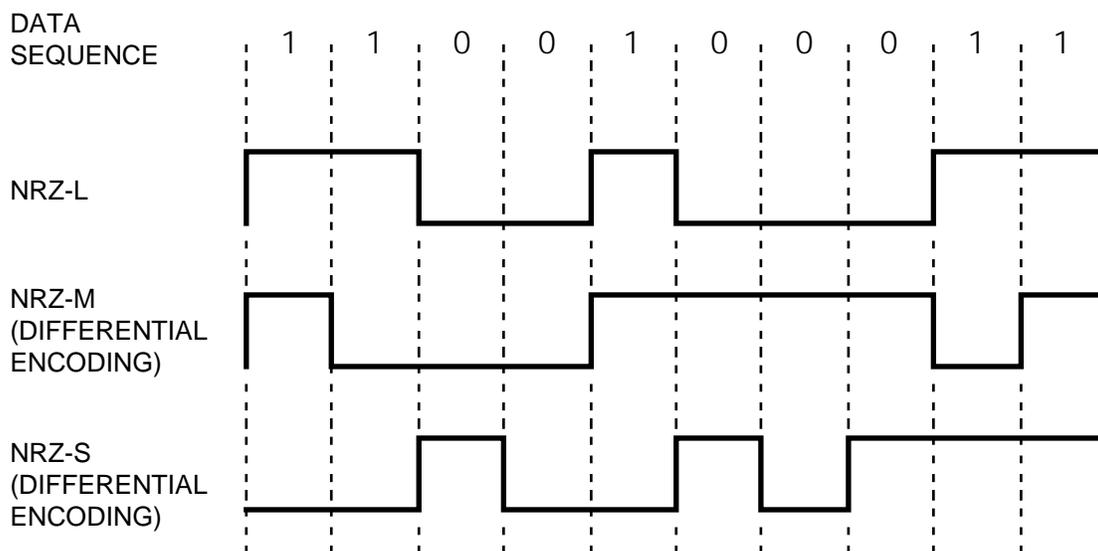
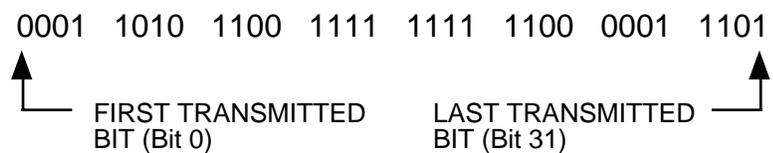


Figure 6. Telemetry Data Formats

## 2.4 *Frame Synchronization of Telemetry*

Frame synchronization is available for all telemetry streams and must be established prior to operation of an RS decoder or, when no outer code is being used, before processing information contained in transfer frames. Synchronization is accomplished by preceding each codeblock or transfer frame with a fixed-length Attached Synchronization Marker (ASM). This known bit pattern can be recognized to determine the start of the codeblocks or transfer frames. It also can be used to resolve the phase ambiguity associated with BPSK modulation if this was not done by the convolutional decoder.

The Consultative Committee for Space Data Systems (CCSDS) has adopted a 32-bit ASM with a pattern of



represented in hexadecimal as 1ACFFC1D, but other ASM lengths and patterns can be accommodated. Characteristics of the frame synchronizer are provided in Table 3.

Frame synchronization is an optional capability. When it is not being used, the received or received-and-decoded bits are placed into telemetry transport blocks in the order received and annotated in the block header with status data consisting of mode identifiers and data quality indicators.

### 2.4.1 *Frame Synchronizer Operation*

When frame synchronization is enabled, the frame synchronizer establishes and maintains synchronization for the incoming data at either of two specified frame lengths. The operation of the synchronizer is defined by four operating modes: Search, Verify, Lock, and Flywheel.

In the Search mode, the synchronizer attempts to find a pattern in the data that differs from the known ASM by a specified number of bit errors. The specified number of bit errors from the synchronization marker is called the Search Bit Error Tolerance (BET<sub>S</sub>). As the synchronizer does this, it collects the received bits in chronological order and divides them into data blocks using the specified frame length (or the longer of the two specified lengths) so they can be forwarded to the user.

Table 3. Frame Synchronizer Characteristics

Parameter	B2MCD
Maximum Data Rate	2.2 Mb/s
Frame Lengths (1 or 2)	Up to 32,766 bits
ASM	8 to 64 bits
Acceptable Bit Errors in ASM ( $BET_S$ and $BET_L$ )	0 to 15 bits, Independently Specifiable for $BET_S$ and $BET_L$
Minimum Error Detection (MED)	Enabled or Disabled
ASM Polarity	True or Inverted
Polarity Correction	Enabled or Disabled
Bit Slip	0 to -3 bits
Check Frames Before In-lock	0 to 15 (frames)
Maximum Number of Flywheel Frames	0 to 15 (frames)
Output Data Rate	800 frames/s (maximum)
Force Out of Synchronization	Yes

There are two user-selectable alternatives when an ASM match is found. Either it can be accepted (Minimum Error Detection (MED) disabled) or the search can be continued for one frame length past the first potential match (MED enabled) to find a potential ASM that has a lower number of bit errors. When the first or best match is found, the current data block is terminated, annotated with frame length and status information, and forwarded to the next stage of telemetry processing (either transfer frame formatting or Reed—Solomon decoding).

In the Verify mode, the synchronizer examines the data stream for an acceptable ASM within a 0 to 3-bit window of either of its possible locations. An acceptable ASM is defined as one having no more than  $BET_S$  errors. In this mode, the data bits that were recognized as the ASM, followed by the data bits up to the beginning of the next ASM, are collected, annotated with frame length and status, and forwarded to the next stage of telemetry processing. The synchronizer continues in the Verify mode for a specified number of frames or until no ASM is detected. If the requisite number of frames occur, the synchronizer advances to the Lock mode. If no ASM is detected, the synchronizer reverts to the Search mode.

In the Lock mode, the synchronizer places an ASM at the beginning of each frame followed by data bits in the order received and annotated with frame length and status information. It continues to examine the data stream for an acceptable ASM within a bit slip window, the Lock Bit Error Tolerance ( $BET_L$ ), that may be specified independently of  $BET_S$ .

The synchronizer remains in the Lock mode until no acceptable ASM is detected. Should this occur, the synchronizer places itself in the Flywheel mode.

In the Flywheel mode, the synchronizer substitutes the received bits instead of the ASM, followed by data bits until the end of the frame is reached (as was done in the verify mode). The synchronizer remains in this mode for a specified number of frames or until an ASM having fewer than  $BET_L$  errors is found and positioned within the bit slip window. If frame synchronization can be re-established within the specified number of frames, the synchronizer returns to Lock mode. If an acceptable ASM is not found, the synchronizer returns to Search mode.

#### **2.4.2. *Frame Synchronizer Performance***

There are no obvious best choices in the selection of frame synchronization parameters because each one gives a trade-off between the probability of incorrectly declaring lock and incorrectly not declaring lock. A computer simulation has been performed, and some results are provided in the following figures.

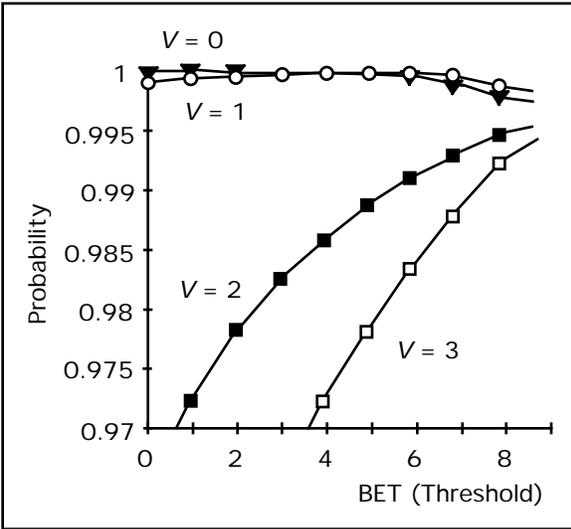
The probability of acquiring frame synchronization within a given number of frames is a function of Bit SNR, burst-error statistics, frame size, ASM length,  $BET_S$ , the number of frames selected to verify synchronization ( $V$ ), and whether MED is used. Figure 7 shows the probability of frame acquisition within four frames versus  $BET_S$  for different frame and ASM lengths with MED enabled and using 0 to 3 check frames ( $V = 0$  to 3).

Figure 8 shows the probability that correct frame synchronization is not found within four frames as a function of Bit SNR for values of  $BET_S$  of 0, 5, and 10. The figure assumes that the MED is enabled and that a 32-bit ASM is used. The probability of dropping lock (once acquired) is shown by Figure 9. The effects of the number of flywheels ( $F$ ),  $BET_L$ , and Bit SNR can be seen from the figure. The probability of dropping lock is independent of the frame length.

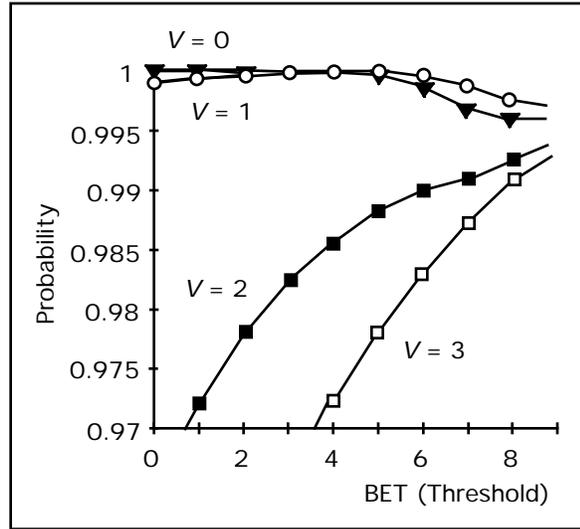
### **2.5 *Decoding of Concatenated Convolutional and Reed—Solomon Code***

Errors in convolutionally coded channels tend to occur in bursts that result when noise causes the decoder to momentarily follow the wrong path through the decoding trellis. The combination of an outer Reed—Solomon (RS) code with an inner convolutional code provides good burst-error correction with minimal bandwidth expansion.

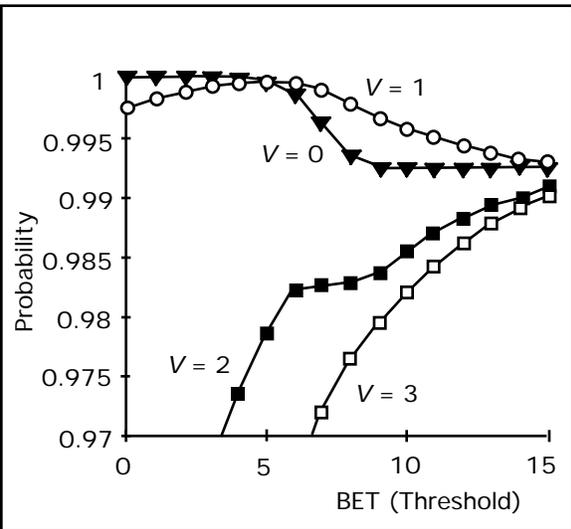
A Reed—Solomon code takes the input data and divides it into  $J$  bit sequences to form symbols. Although other symbol sizes are feasible, the ability of computers to handle information as 8-bit bytes has led to a standardization of  $J = 8$ . The RS encoder creates parity symbols from the data that enable the decoder to correct any combination of  $E$  or fewer symbol errors per RS codeword. The value  $E$  is referred to as the code redundancy. The length of an RS codeword is  $2^J - 1$  symbols or 255 symbols when  $J = 8$ . The output of an RS encoder is a



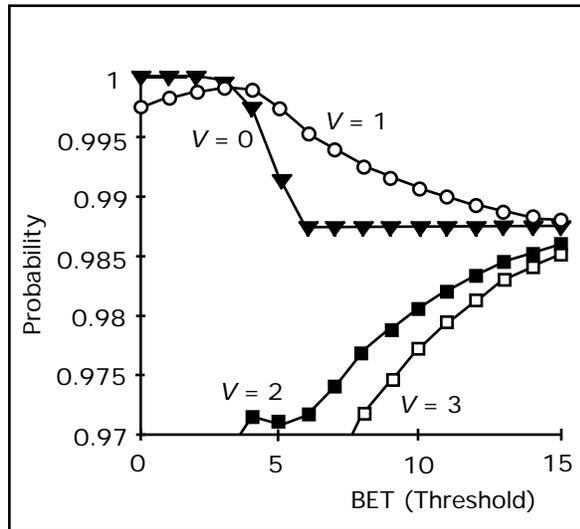
a. Frame transport length = 256 bits  
ASM length = 16 bits  
Coded (7,1/2) Bit SNR = 2.1 dB



b. Frame transport length = 1024 bits  
ASM length = 16 bits  
Coded (7,1/2) Bit SNR = 2.1 dB



c. Frame transport length = 480 bits  
ASM length = 32 bits  
Coded (7,1/2) Bit SNR = 2.1 dB



d. Frame transport length = 16320 bits  
ASM length = 32 bits  
Coded (7,1/2) Bit SNR = 2.1 dB

Figure 7. Probability of Frame Acquisition (in Four Frames) as a Function of  $BET_T$  Threshold and the Number of Check Frames ( $V$ ) with MED Enabled

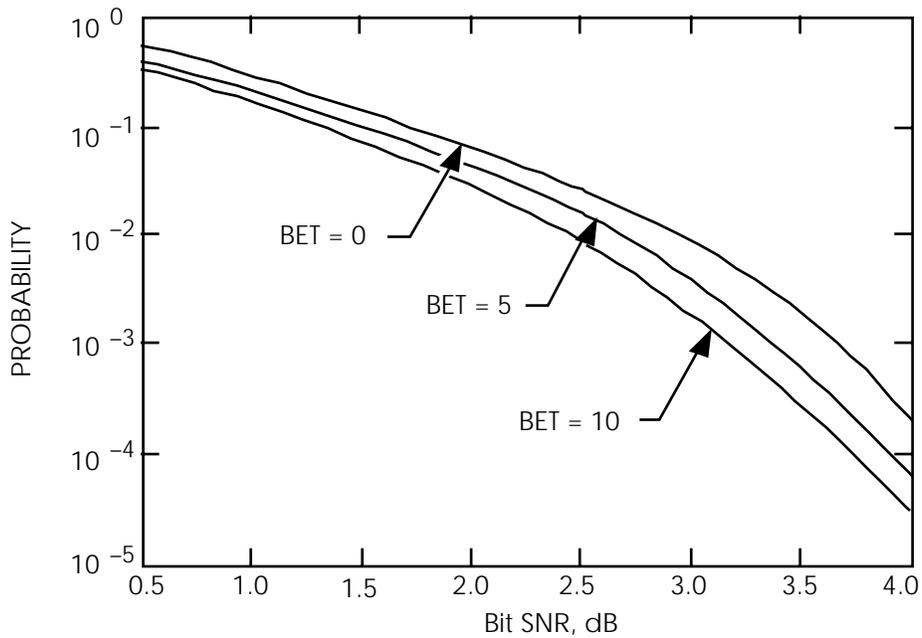


Figure 8. Probability of Frame Acquisition (in Four Frames) Versus Bit SNR for Several Values of BET<sub>S</sub>. Frame Length = 5120, V=2, MED Enabled, ASM = 32 bits, Coded (7,1/2) Data

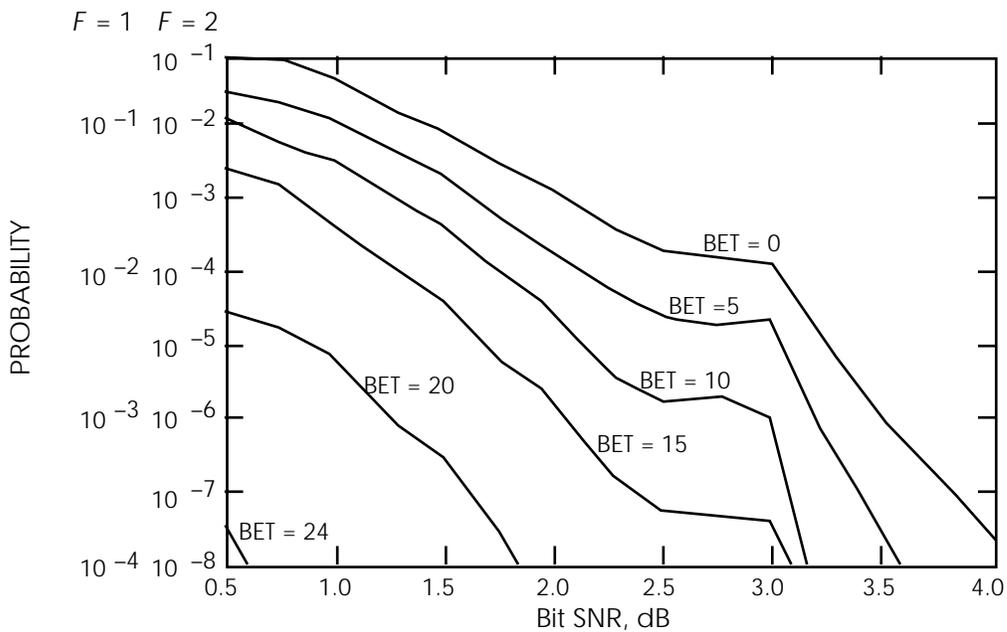


Figure 9. Probability of Frame Out of Lock for the Number of Flywheels ( $F$ ) Set at One or Two Versus Bit SNR for Various Values of Bit Error Tolerance (BET<sub>L</sub>). ASM = 32 bits, Coded (7,1/2) Data

*systematic* code (one in which the input symbols are contained in the output) containing  $2^J - (1 + 2E)$  information symbols followed by  $2E$  parity symbols. The standard RS code supported by the DSN is capable of correcting 16 or less RS symbol errors ( $E = 16$ ) and is referred to as the RS (255,223) code.

### **2.5.1. Interleaving**

The burst errors associated with Viterbi decoding can be as long as several constraint lengths. Thus, several closely spaced error bursts could exceed the decoder's error correction capability—especially with longer constraint length codes. Interleaving is a technique that spreads the effects of burst errors across several RS codewords. The input data are collected eight bits at a time to form RS symbols from which parity is calculated. The partially completed parity symbols are stored in  $8I$ -bit-long shift registers, where  $I$  is the *interleave factor*. After  $8I$  bit times, the output of the shift registers is available for use in calculating the next parity symbol. The result is to build the parity symbols from consecutive 8-bit data symbols when the interleave factor is one but from every  $I$ th 8-bit data symbol when  $I$  is other than one.

The input data are passed directly to the convolutional encoder as the parity symbols are being calculated. Thus, the code remains systematic—independent of the interleave factor. When all information symbols have been processed, the parity symbols are shifted out of the registers and passed to the convolutional encoder. This takes the 32 parity symbols from the first 223 information symbols and disperses them across the entire  $32I$  parity symbol portion of the codeblock at  $I$ -symbol intervals. Figure 10 illustrates the symbol arrangement for an interleave factor of 5.

When the data are received, they are written into an array from which the parity symbols associated with each of the  $I$  RS codewords can be separated. DSN supports interleaving for values of  $I$  between 1 (no interleaving) and 16.

### **2.5.2. Reed–Solomon Encoder**

The conventional architecture for an RS encoder employs binary multipliers and read-only memories to enable the log of the information symbols to be quickly accessed for calculation and to enable the results of these calculations to be converted back to parity symbols via a table of antilogs. In 1968, a simplified method was developed (named the Berlekamp Architecture, after its inventor) that, in combination with appropriate selection of the RS code generator polynomial, enables parity symbols to be calculated using bit-serial multipliers constructed with a matrix of exclusive OR gates. The DSN and CCSDS have adopted an RS code that is compatible with the Berlekamp Architecture and has the following specifications:

- (1)  $J = 8$  bits per RS symbol.
- (2)  $E = 16$  RS symbol error correction capability.

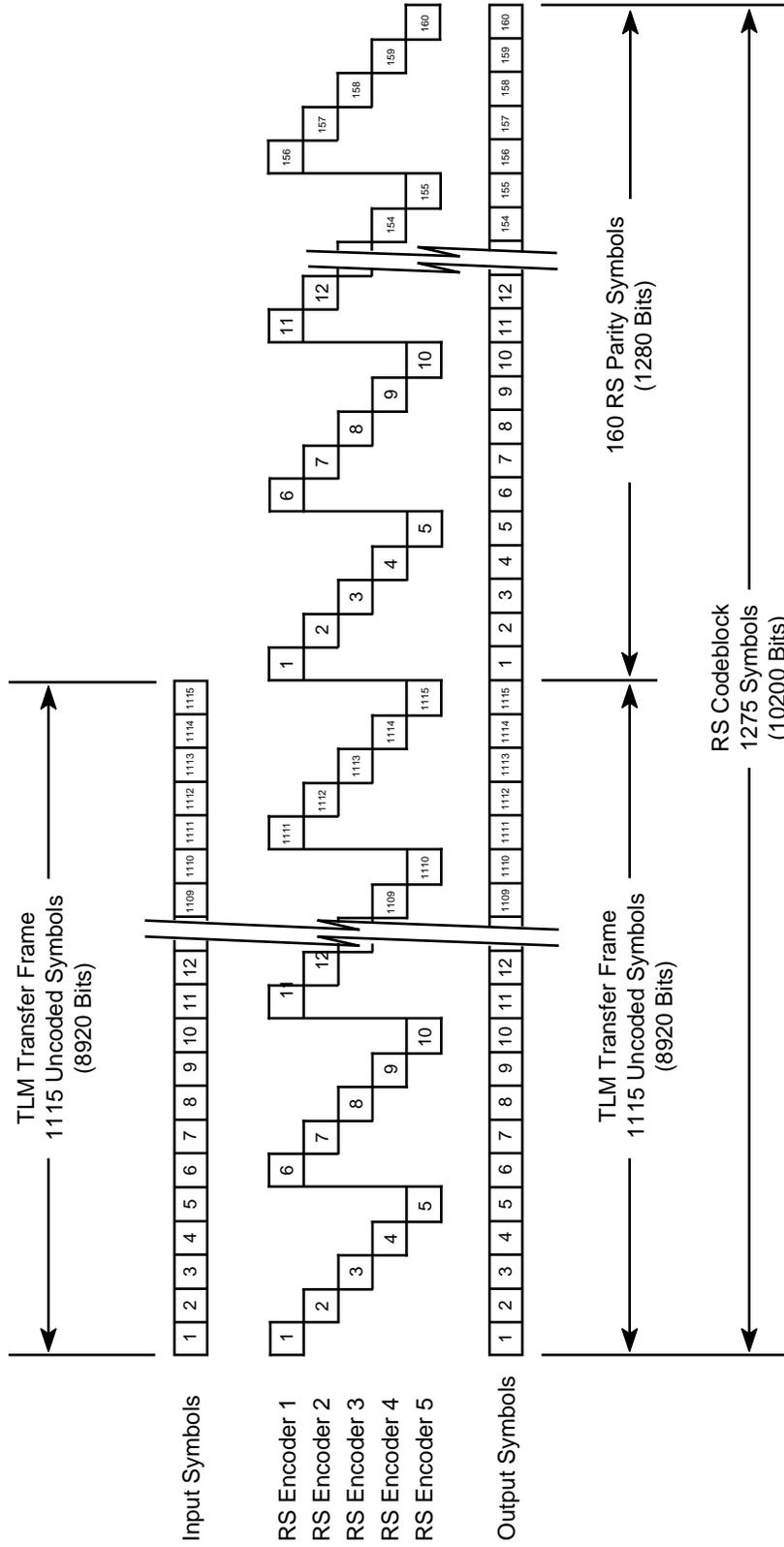


Figure 10. Reed-Solomon Symbol Arrangement for Interleave Factor ( $I$ ) of 5.

- (3) Field generator polynomial:

$$F(x) = x^8 + x^7 + x^2 + x + 1$$

over the Galois Field,  $GF(2^8)$ .

- (4) Code generator polynomial:

$$g(x) = \prod_{j=112}^{143} (x - \alpha^{11j}) = \sum_{i=0}^{32} G_i x^i$$

over the Galois Field,  $GF(2^8)$ , where  $F(\alpha) = 0$ .

It should be recognized that  $\alpha^{11}$  is a primitive element of  $GF(2^8)$  and that  $F(x)$  and  $g(x)$  characterize a (255,223) Reed–Solomon code.

Figure 11 shows the design of a Berlekamp encoder for producing the DSN/CCSDS standard RS code. The performance of RS codes at several interleave factors and for several of the supported convolutional inner codes is shown in Figure 12.

### 2.5.3. *Virtual Fill*

The maximum amount of input data that can be transmitted in a codeblock varies from 1784 bits (with no interleaving) to 28,554 bits (with an interleaving depth of 16). If a transfer frame has less data than 1784 bits, the codeblock can be completed by inserting virtual fill (all-zero RS symbols) between the ASM and the start of the input data. The amount of virtual fill (in units of 8-bits) must be fixed for a tracking pass and is inserted by the encoder and decoder but not transmitted. The efficiency of RS coding will decrease as the amount of virtual fill increases because the number of parity symbols remains fixed while the number of data symbols decreases. An illustration of virtual fill is shown in Figure 13.

### 2.5.4. *Frame Error Rate of Concatenated Codes*

Coding performance has traditionally been expressed in terms of bit-error rate (BER). This may not be the most appropriate measure for all applications. For example, the failure to correct an RS codeword is an indication that it contains at least 17 RS (7.6%) symbol errors. If the information has been RS coded in order to provide an error rate that is lower than can be obtained with convolutional coding, a block with an error rate of  $7.6 \times 10^{-2}$  is probably useless. Simulations have been performed to predict the Frame Error Rate (FER) as a function of interleaving depth. Figure 14 illustrates the modeled frame error rate for several DSN concatenated codes and interleave factors.

### 2.5.5 *Variable Redundancy and Feedback Concatenated Decoding*

The feedback concatenated decoding technique implemented for the Galileo low-rate telemetry mission utilizes a frame of eight codewords having four different redundancies ( $E$ ) and arranged with  $E = 47, 5, 15, 5, 30, 5, 15, \text{ and } 5$ . The average redundancy of these frames is

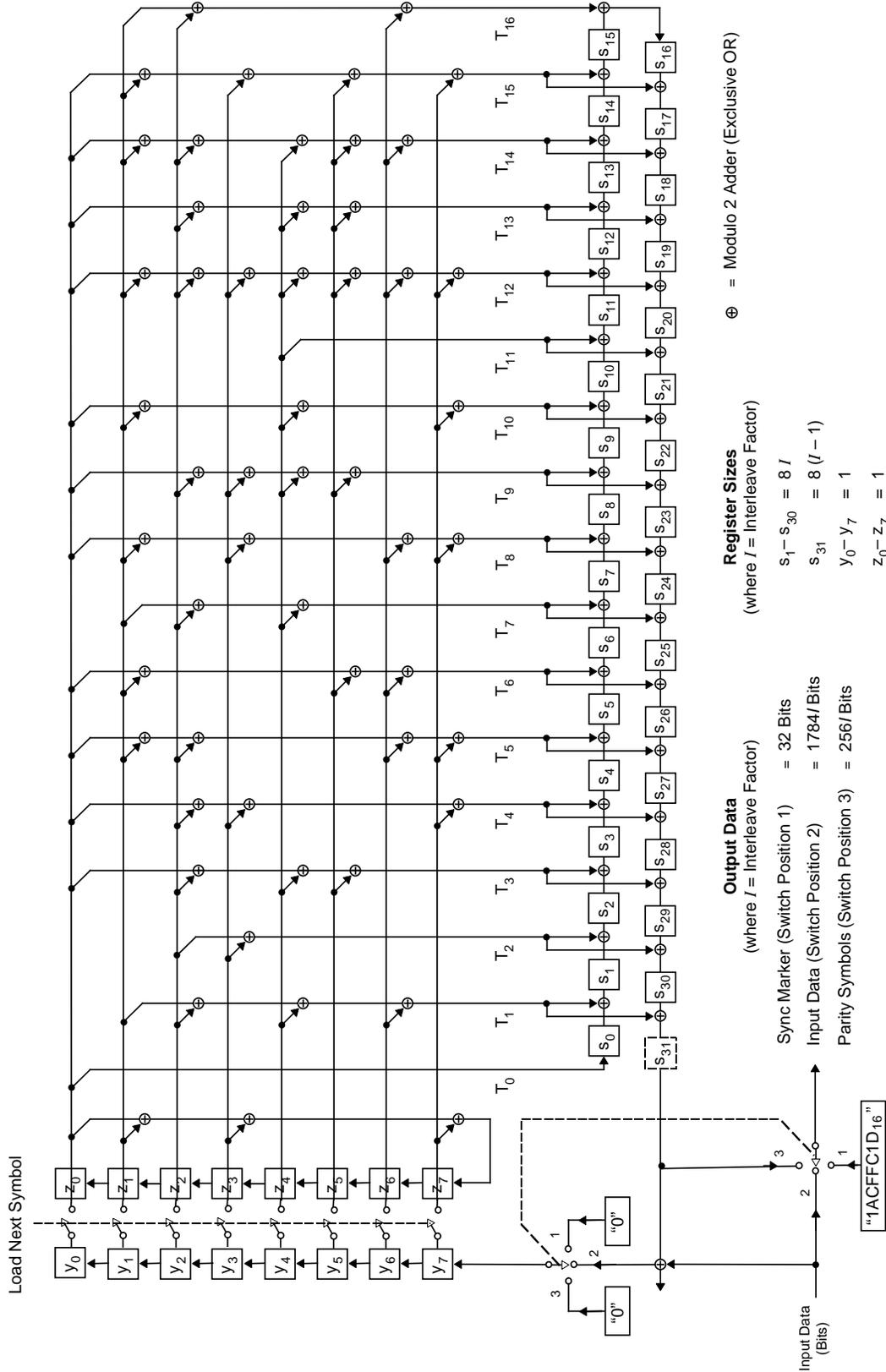


Figure 11. Berlekamp Architecture Reed-Solomon Encoder

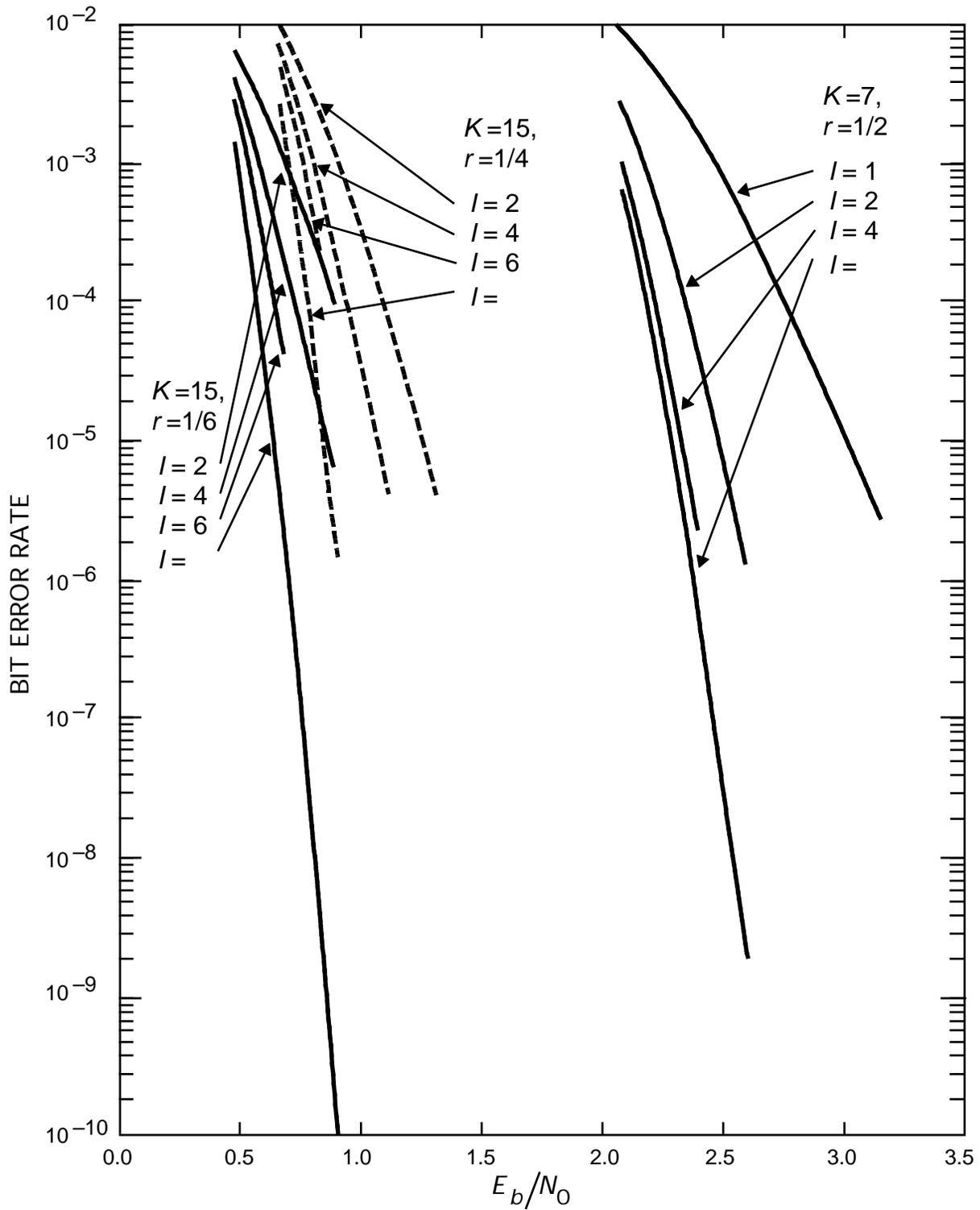


Figure 12. Reed-Solomon Code (255,223) Performance for Standard Convolutional Inner Codes Showing the Effect of Interleave Factor ( $l$ )

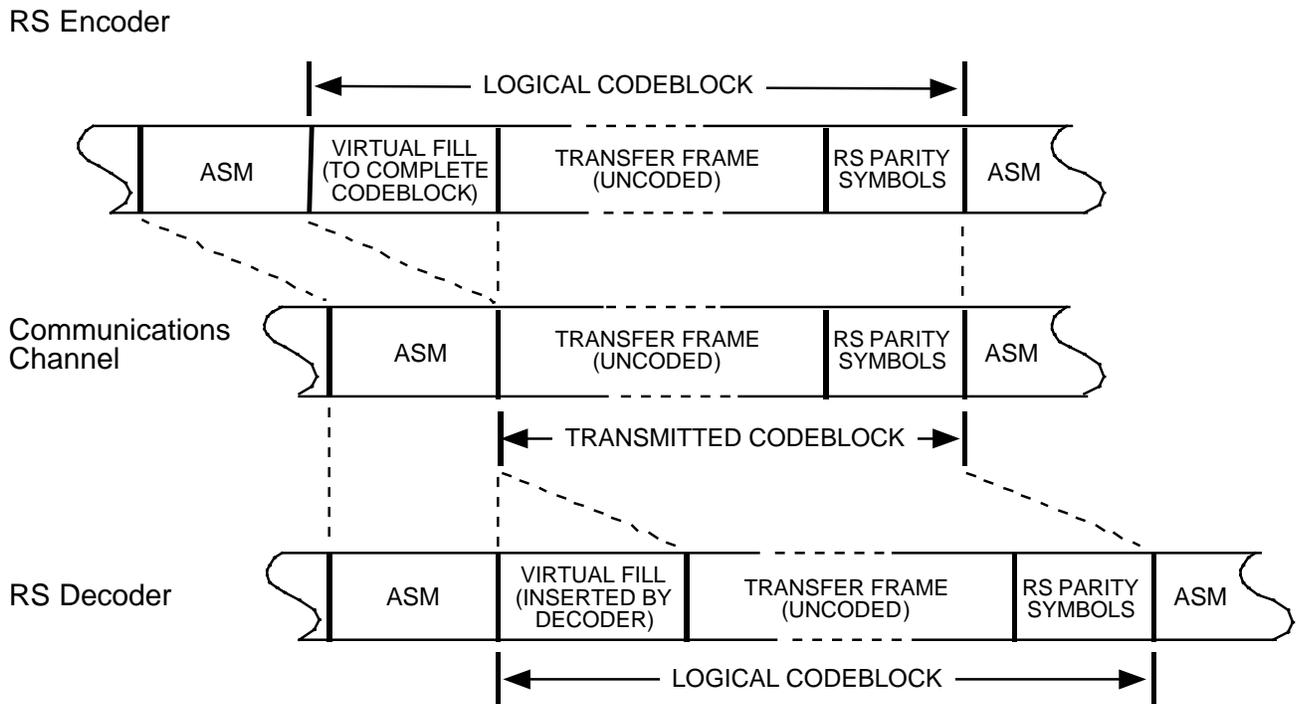


Figure 13. Virtual Fill

approximately the same as a standard RS (255,223) code, but the code becomes non-systematic after 1288 ( $[255 - 2 \times 47] \times 8$ ) symbols as parity symbols become interspersed with the data symbols.

The frames are decoded by making from two to four passes through a Viterbi decoder with each pass being constrained by known symbols from earlier passes. On the first pass, the Viterbi decoder is unconstrained, and decoding of the four codewords with the highest redundancies ( $E = 47, 30,$  and  $15$ ) is attempted. Depending on how many of these codewords are successfully decoded, either one or two additional passes are made with additional constraints from earlier passes. When these four codewords have been decoded, a final pass is made to decode the four codewords with the lowest redundancy ( $E=5$ ).

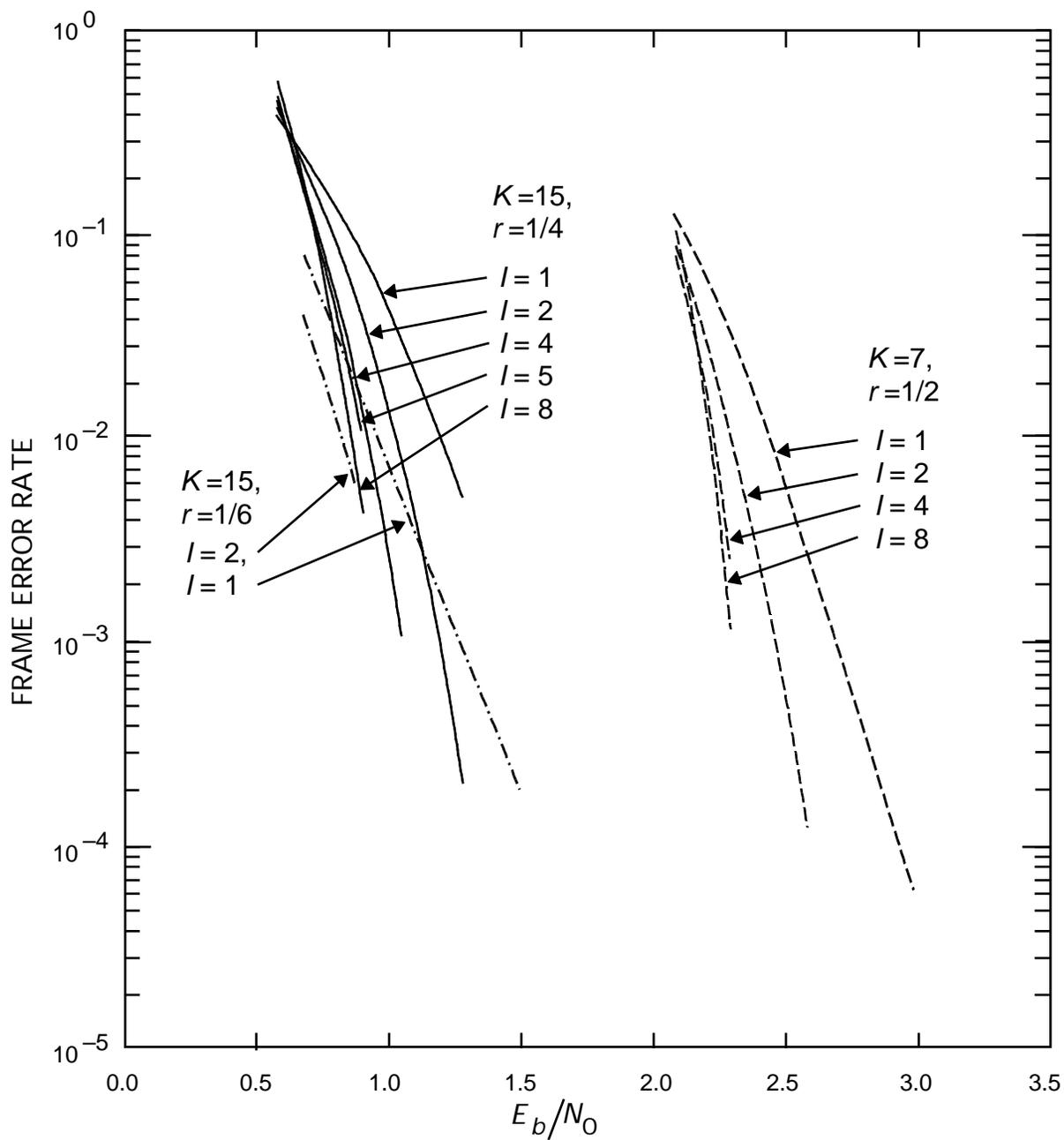


Figure 14. Modeled Frame Error Rate for several DSN Concatenated Codes and Interleave Factors ( $I$ )

### **3**            ***Proposed Capabilities***

The following paragraphs discuss capabilities that have not yet been implemented by the DSN but have adequate maturity to be considered for spacecraft mission and equipment design. Telecommunications engineers are advised that any capabilities discussed in this section cannot be committed to except by negotiation with the Telecommunications and Mission Operations Directorate (TMOD) Plans and Commitments Program Office.

#### **3.1**            ***Parallel Concatenated Convolutional Codes (Turbo Codes)***

Turbo codes provide near-Shannon-limit error-correction performance with reasonable decoding complexity. A turbo encoder consists of an interleaver that is used to permute a block of input data in a random fashion and two simple recursive convolutional encoders that produce sets of parity bits from the information bits and the permuted information bits. A second interleaver is used to combine the information bits and the two sets of parity bits. The resulting turbo code symbol stream is systematic and non-transparent.

The DSN is in the process of implementing a decoder for the turbo code specified in CCSDS Recommendation 101.0-B-4. The recommendation permits information block lengths ( $k$ ) of 1,784, 3568, 7136, 8920, and 16,384 bits, and nominal code rates ( $r$ ) of 1/2, 1/3, 1/4, and 1/6. The first four of these block lengths are the same as would be required for Reed-Solomon encoding using an interleave factor ( $I$ ) of 1, 2, 4, or 5.

##### **3.1.1**            ***Turbo Code Encoder***

A practical encoder includes two input buffer/interleavers to ensure a continuous stream of output symbols. A switching arrangement allows one buffer to be filled while the second is being read simultaneously in the normal and permuted orders. The interleaver responsible for the combination of information and parity bits becomes the third interleaver.

Figure 15 illustrates the design of a CCSDS compliant turbo for nominal code rates of 1/3 and 1/6. Switch SW1 is used to alternately route a block of input data bits into buffers 1 and 2. Switches SW2 and SW3 operate simultaneously to route the normal and permuted outputs of the buffers into the coders. At the end of each block, SW2 and SW3 move to position 1 for four additional bit times before selecting the opposite buffer/interleaver. This causes the encoders to be filled with zeroes in anticipation of processing the next block of data. The symbol output that occurs during the zeroing of the encoders is called the trellis termination sequence and provides a known final state (zero) to the turbo decoder.

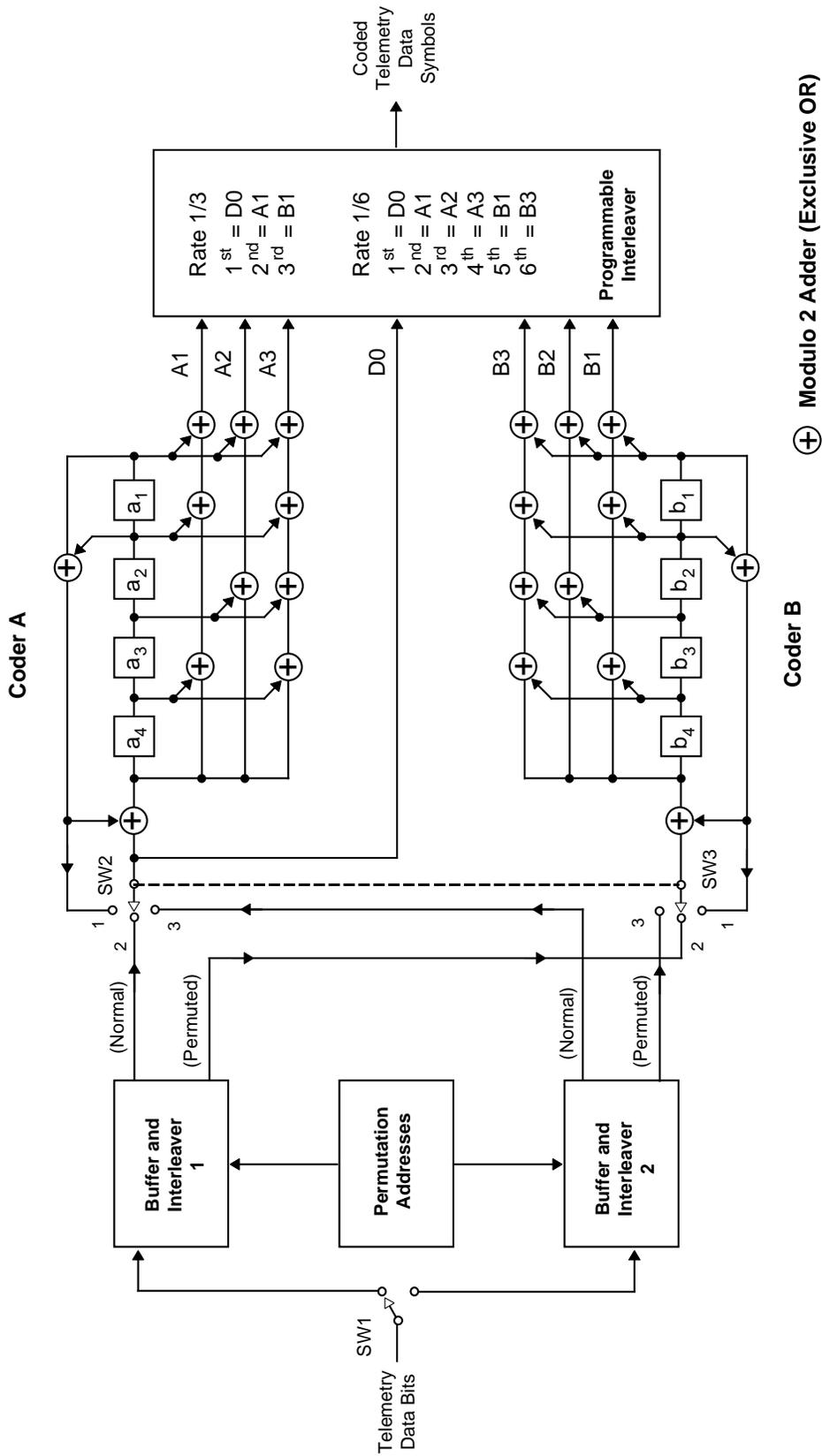


Figure 15. CCSDS Compliant Rate=1/3 and Rate=1/6 Turbo Encoder

### **3.1.2        *Permutations***

Turbo codes have been shown to provide good performance using randomly chosen permutations. However, implementation constraints require that a limited number be considered and that these be selected from the ones offering the best performance. Having discovered the characteristics of good random permutations, it is possible to develop an algorithm for a permutation that approaches their performance. Such an algorithm has been adopted by the CCSDS and is described in the previously cited recommendation. An algorithmic approach has the advantage of ease of delivery to flight and ground equipment at the possible expense of a small performance penalty.

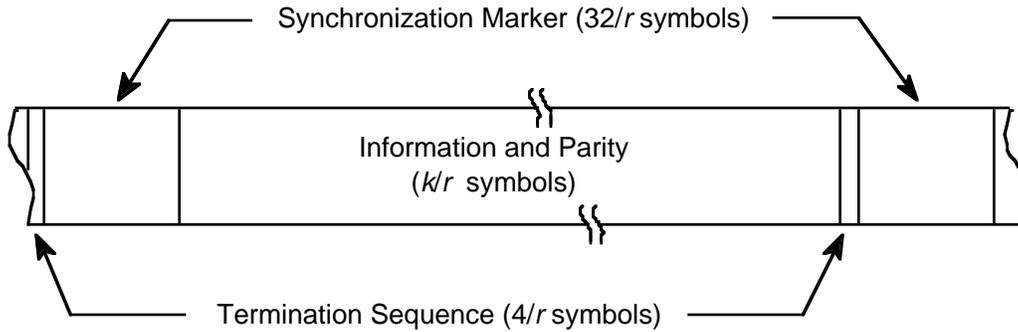
The alternative is to select the best permutation for a given information block size and deliver it to the flight and ground equipment as a file of numbers. This approach has the possibility of providing slightly improved performance at the expense of less convenient delivery of the permutation. The performance estimates in this document employ permutations that appear random but were manually optimized for the two selected block lengths.

### **3.1.3        *Synchronization of Turbo Codes***

Codeblock synchronization is necessary for proper decoding of turbo codes because the decoding operation needs to know the codeblock boundaries before it can iterate between the unpermuted and permuted data domains. Unlike the frame synchronization for Reed–Solomon codes, which is performed after convolutional decoding, synchronization for turbo codes must be done in the channel-symbol domain. This requires a rate-dependent attached synchronization marker and synchronization algorithms that operate at the symbol rate as opposed to the data rate. Operation of the synchronizer is anticipated to be similar to that of the RS frame synchronizer and involves recognition of the marker in the coded symbol stream and anticipation of a recurring marker at an interval corresponding to the length of the turbo codeblock, the trellis termination sequence, and the synchronization marker. Figure 16 illustrates the structure of the turbo code as it appears in the information channel.

### **3.1.4        *Turbo Code Decoder Implementation***

The turbo decoder uses an iterative decoding algorithm based on simple decoders individually matched to the two constituent codes. Each constituent decoder makes a maximum *a posteriori* probability (MAP) estimate for each bit from the uncoded information symbols (in normal or permuted form, as appropriate) and the parity symbols generated by its corresponding encoder. The MAP algorithm requires that the first and last states of the of the code sequence be known, which is most easily accomplished by transmitting the trellis termination sequence over the information channel. The decoders exchange their MAP estimates via the permutation matrices to be used by the opposite decoder as *a priori* estimates for a second iteration. The exchange of MAP estimates and iterations continues for a fixed number of times, depending on SNR, or until a satisfactory convergence is reached. The final output is a hard-quantized version of the likelihood estimates from either one of the decoders.



$k$  = information block size (1784, 3568, 7136, 8920, or 16384 bits)

$r$  = code rate (1/2, 1/3, 1/4, or 1/6)

Figure 16. Turbo Code Structure in the Information Channel

### 3.1.5 *Turbo Code Latency*

Because the decoder processes information one block at a time, there is a minimum decoding delay of  $k + 4r$  symbols (where  $k$  is the block size and  $r$  is the code rate). This latency is further increased by the (to be determined) iteration time of the decoders.

### 3.1.6 *Turbo Code Error Floor*

A turbo code's performance curve does not stay steep forever as does that of a convolutional/RS concatenated code. It eventually reaches an error floor, flattens out considerably, and looks, from that point onward, like the performance curve for its weak constituent convolutional codes. As a result, Projects using highly compressed data may benefit from the addition of an outer code that they can remove after the data are delivered by the DSN. The outer code can have a low code rate because only a few errors per block are expected at the error floor and the code will, therefore, have little effect on the required SNR.

### 3.1.7 *Selection and Performance of Turbo Codes*

The DSN intends to implement the complete set of turbo codes adopted by the CCSDS. However, performance has been simulated only for the subset of codes shown in Table 4 and identified by their block sizes and code rate. Figure 17 shows the modeled performance for these four codes as a function of their information bit rate. That is, the small overhead associated with the ASM and termination sequence is included to make them comparable to the other performance curves published in this document. Note that these curves are for frame error rate because a decoding failure affects an entire frame. Also note that these simulations employed optimized permutations rather than those produced by the CCSDS algorithm.

Table 4. Turbo Code Characteristics

Block Size (Information Bits)	Rate ( $r$ )	ASM Length (Symbols)	Trellis Termination (Symbols)
1784	1/3	96	12
1784	1/6	192	24
8920	1/3	96	12
8920	1/6	192	24

Selection of which of the four codes should be used depends on several factors but primarily on data rate. For example, a rate 1/6 code is not the best choice at low data rates because it has a smaller  $E_s/N_0$  than does a rate 1/3 code for a given  $E_b/N_0$ . This leads to increased receiver losses in the symbol, subcarrier, and Costas carrier (if used) loops. Thus, a rate 1/6 code can give poorer performance than a rate 1/3 code. A complete discussion of receiver losses is contained in module 207 of this document. The smaller block size of 1784 is also attractive at lower data rates because it leads to shorter acquisition times

At high data rates, the better baseline performance of the rate 1/6 codes makes them more attractive. The (8920,1/6) code often will be used for the higher data rates because it has the best baseline performance. The remaining two codes, (1784,1/6) and (8920,1/3), represent compromises between the extremes of the (1784,1/3) and (8920,1/6) codes.

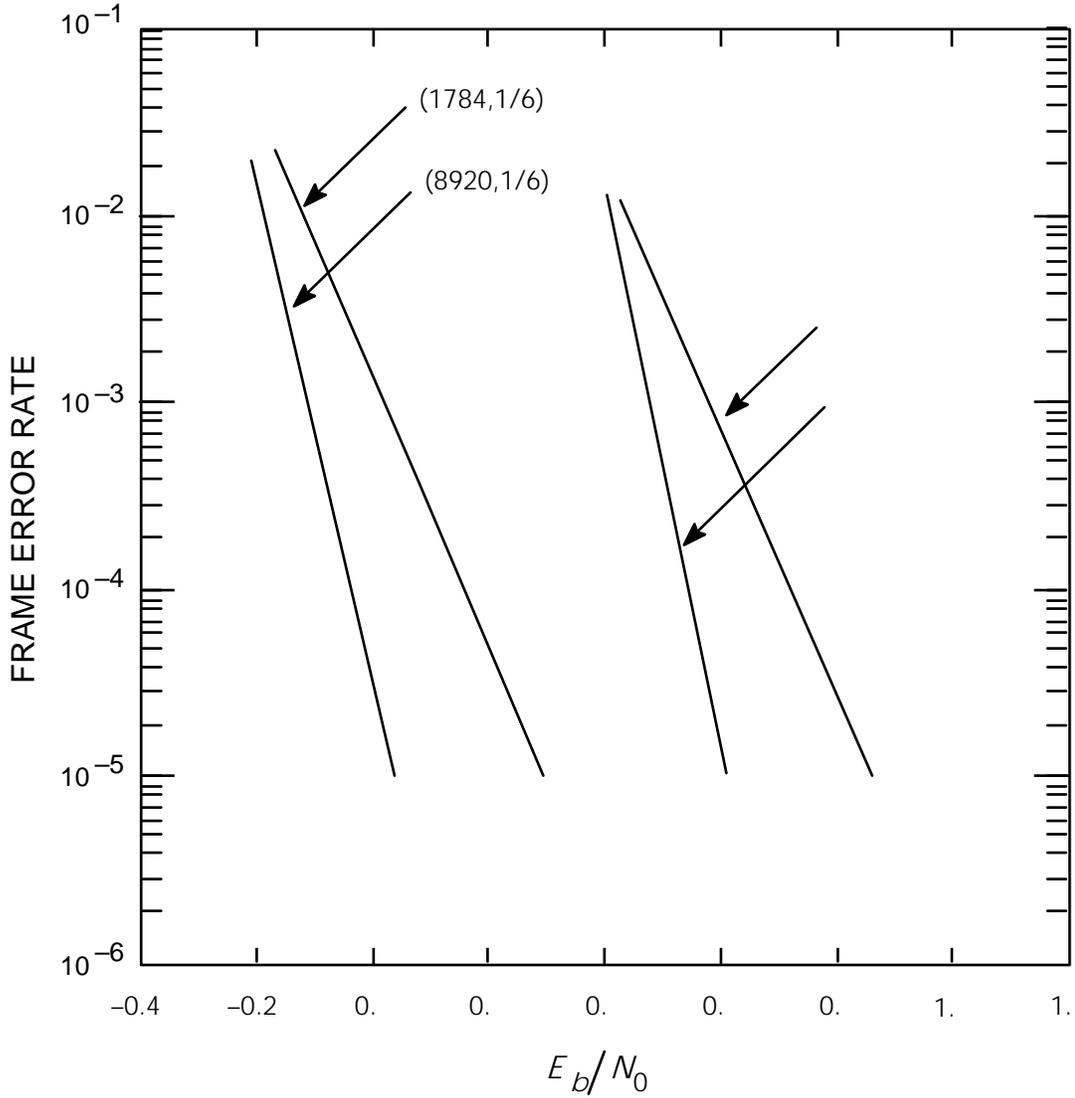


Figure 17. Modeled Performance of DSN Turbo Codes Versus Information Bit  $E_b/N_0$ .

***Appendix A***  
***Reference***

- 1 CCSDS 101.0-B-4, Telemetry Channel Coding, Blue Book. Issue 4, May 1999.