The Consultative Committee for Space Data Systems

Report Concerning Space Data System Standards

TM SYNCHRONIZATION AND CHANNEL CODING — SUMMARY OF CONCEPT AND RATIONALE

INFORMATIONAL REPORT

CCSDS 130.1-G-1

GREEN BOOK
June 2006

AUTHORITY

Issue:	Informational Report, Issue 1
Date:	June 2006
Location:	Washington, DC, USA

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and reflects the consensus of technical panel experts from CCSDS Member Agencies. The procedure for review and authorization of CCSDS Reports 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 Report which contains background and explanatory material to support the CCSDS Recommended Standard, *TM Synchronization and Channel Coding* (reference [3]).

Through the process of normal evolution, it is expected that expansion, deletion, or modification to this Report may occur. This Report will therefore be subject to CCSDS document management and change control procedures which are defined in reference [1]. Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/ccsds/

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

At time of publication, the active Member and Observer Agencies of the CCSDS were:

Member Agencies

- – Agenzia Spaziale Italiana (ASI)/Italy.
- – British National Space Centre (BNSC)/United Kingdom.
- – Canadian Space Agency (CSA)/Canada.
- – Centre National d'Etudes Spatiales (CNES)/France.
- – Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)/Germany.
- – European Space Agency (ESA)/Europe.
- – Federal Space Agency (Roskosmos)/Russian Federation.
- – Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- – Japan Aerospace Exploration Agency (JAXA)/Japan.
- – National Aeronautics and Space Administration (NASA)/USA.

Observer Agencies

- – Austrian Space Agency (ASA)/Austria.
- – Belgian Federal Science Policy Office (BFSPO)/Belgium.
- – Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- – Centro Tecnico Aeroespacial (CTA)/Brazil.
- – Chinese Academy of Space Technology (CAST)/China.
- – Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- – Danish Space Research Institute (DSRI)/Denmark.
- – European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- – European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- – Hellenic National Space Committee (HNSC)/Greece.
- – Indian Space Research Organization (ISRO)/India.
- – Institute of Space Research (IKI)/Russian Federation.
- – KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- – Korea Aerospace Research Institute (KARI)/Korea.
- – MIKOMTEK: CSIR (CSIR)/Republic of South Africa.
- – Ministry of Communications (MOC)/Israel.
- – National Institute of Information and Communications Technology (NICT)/Japan.
- – National Oceanic & Atmospheric Administration (NOAA)/USA.
- – National Space Organization (NSPO)/Taipei.
- – Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- – Swedish Space Corporation (SSC)/Sweden.
- – United States Geological Survey (USGS)/USA.

DOCUMENT CONTROL

Document	Title	Date	Status
CCSDS 130.1-G-1	TM Synchronization and Channel Coding—Summary of Concept and Rationale, Informational Report, Issue 1	June 2006	Current issue
EC 1	Editorial Correction	July 2007	Corrects miscellaneous typographical errors.

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

CONTENTS (continued)

CONTENTS (continued)

1   DOCUMENT PURPOSE, SCOPE, AND ORGANIZATION

1.1   PURPOSE

This report contains the concept and supporting rationale for *TM Synchronization and Channel Coding* developed by the Consultative Committee for Space Data Systems (CCSDS). It has been prepared to serve two major purposes:

a) to provide an introduction and overview for the Channel Coding concept upon which the detailed CCSDS *TM Synchronization and Channel Coding* specifications (reference [3]) are based;

b) to describe and explain the codes considered and to supply the supporting rationale.

Supporting performance information along with illustrations are also included. This report provides a broad tutorial overview of the CCSDS *TM Synchronization and Channel Coding* and is aimed at helping first-time readers to understand the Recommended Standard. It is not intended to provide all necessary knowledge for successfully designing telemetry communication links.

In no event will CCSDS or its members be liable for any incidental, consequential, or indirect damages, including any lost profits, lost savings, or loss of data, or for any claim by another party related to errors or omissions in this report. This document is a CCSDS informational Report and is therefore not to be taken as a CCSDS Recommended Standard. The actual Recommended Standard is in reference [3].

1.2   SCOPE

The concepts, protocols and data formats developed for the *TM Synchronization and Channel Coding* described herein are designed for space communications links, primarily between spacecraft and ground elements. Data formats are designed with efficiency as a primary consideration; i.e., format overhead is minimized. The results reflect the consensus of experts from many space agencies.

This document provides supporting and descriptive material only: **it is not part of the Recommended Standard**. In the event of any conflict between the *TM Synchronization and Channel Coding* Recommended Standard (reference [3]) and the material presented herein, the Recommended Standard shall prevail.

1.3   ORGANIZATION

An overview of the CCSDS Telemetry System is presented in section 2, which introduces the notion of architectural layering to achieve transparent and reliable delivery of scientific and engineering sensor data (generated aboard remote space vehicles) to the users located in space or on Earth.

Section 3 introduces the elements of TM Synchronization and Channel Coding and the specific codes considered in the CCSDS *TM Synchronization and Channel Coding* Recommended Standard (reference [3]).

Subsequent sections describe in detail the characteristics, performance, and rationale of the four classes of codes considered: convolutional, Reed-Solomon, concatenated, and turbo codes.

Annex A presents a Glossary in order to familiarize the reader with the terminology used throughout the CCSDS Telemetry System. Annex B is a list of acronyms and abbreviations. Annex C presents some rationale for turbo code parameter selection.

1.4   REFERENCES

[1]   *Procedures Manual for the Consultative Committee for Space Data Systems*. CCSDS A00.0-Y-9. Yellow Book. Issue 9. Washington, D.C.: CCSDS, November 2003.

[2]   *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.

[3]   *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.

[4]   *Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model*. International Standard, ISO/IEC 7498-1. 2nd ed. Geneva: ISO, 1994.

[5]   *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.

[6]   *Lossless Data Compression*. Report Concerning Space Data System Standards, CCSDS 120.0-G-1. Green Book. Issue 1. Washington, D.C.: CCSDS, May 1997.

[7]   *Space Packet Protocol*. Recommendation for Space Data System Standards, CCSDS 133.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, September 2003.

[8]   C. E. Shannon. "A Mathematical Theory of Communication," *Bell System Technical Journal* 27 (July and October, 1948): 379-423, 623-656.

[9]   J. P. Odenwalder. *Concatenated Reed-Solomon/Viterbi Channel Coding for Advanced Planetary Missions, Final Report*. Contract 953866, December 1, 1974.

[10] K. Y. Liu. *The Effects of Receiver Tracking Phase Error on the Performance of Concatenated Reed-Solomon/Viterbi Channel Coding System*. JPL Publication 81-62. Pasadena, California: NASA-Jet Propulsion Laboratory, September 1, 1981.

[11] J. P. Odenwalder et al. *Hybrid Coding Systems Study, Final Report*. NASA-Ames Research Center Contract NAS2-6722. San Diego, California: Linkabit Corporation, September 1972.

[12] M. Perlman and J. J. Lee. *Reed-Solomon Encoders—Conventional vs. Berlekamp's Architecture*. JPL Publication 82-71. Pasadena, California: Jet Propulsion Laboratory, December 1, 1982.

[13] U. Cheng. "Node Synchronization of Viterbi Decoders Using State Metrics." *TDA Progress Report* 42-94, April-June 1988 (August 15, 1988): 201-209. <http://tmo.jpl.nasa.gov/tmo/progress_report/42-94/94P.PDF>

[14] D. Divsalar and F. Pollara. "Turbo Codes for Deep-Space Communications." *TDA Progress Report* 42-120, October-December 1994 (February 15, 1995): 29-39. <http://tmo.jpl.nasa.gov/tmo/progress_report/42-120/120D.pdf>

[15] D. Divsalar, S. Dolinar, F. Pollara, R.J. McEliece. "Transfer Function Bounds on the Performance of Turbo Codes." *TDA Progress Report* 42-122, April-June 1995 (August 15, 1995): 44-55. <http://tmo.jpl.nasa.gov/tmo/progress_report/42-122/122A.pdf>

[16] S. Dolinar, D. Divsalar, and F. Pollara . "Code Performance as a Function of Block Size." *TMO Progress Report* 42-133, January-March 1998 (May 15, 1998): 1-23. <http://tmo.jpl.nasa.gov/tmo/progress_report/42-133/133K.pdf >

[17] C. Berrou, A. Glavieux, and P. Thitimajshima. "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo codes." *Proceedings of IEEE International Conference on Communications*, 1064-1070. Geneva: IEEE, 1993.

[18] S. Benedetto et al. "Soft-Output Decoding Algorithms in Iterative Decoding of Turbo Codes." *TDA Progress Report* 42-124, October-December 1995 (February 15, 1996): 63-87. <http://tmo.jpl.nasa.gov/tmo/progress_report/42-124/124G.pdf>

[19] S. Benedetto et al. "A Soft-Input Soft-Output Maximum A Posteriori (MAP) Module to Decode Parallel and Serial Concatenated Codes." *TDA Progress Report* 42-127, July-September 1996 (November 15, 1996): 1-20. <http://tmo.jpl.nasa.gov/tmo/progress_report/42-127/127H.pdf>

[20] J. Hamkins and D. Divsalar. "Coupled Receiver-Decoders for Low Rate Turbo Codes." *Proceedings of IEEE Inernational Symposium on Information Theory*, 381–381. Geneva: IEEE, 2003.

[21] A. J. Viterbi and J. K. Omura. *Principles of Digital Communication and Coding*. New York: McGraw-Hill, 1979.

[22] I. Reed and G. Solomon. "Polynomial Codes Over Certain Finite Fields." *SIAM Journal on Applied Mathematics* 8 no. 2: 300-304.

[23] R. J. McEliece and L. Swanson. "On the Decoder Error Probability for Reed-Solomon Codes." *TDA Progress Report* 42-84, October-December 1985 (February 15, 1986): 66-72. <http://tmo.jpl.nasa.gov/tmo/progress_report/42-84/84F.PDF>

[24] R. J. McEliece. "The Decoding of Reed-Solomon Codes." *TDA Progress Report* 42-95, July-September 1988 (November 15, 1988): 153-157. <http://tmo.jpl.nasa.gov/tmo/progress_report/42-95/95O.PDF>

[25] G. D. Forney, Jr.. "The Viterbi algorithm." *Proceedings of the IEEE* 61 (1973): 268-278.

[26] G. D. Forney, Jr. *Concatenated Codes*. Cambridge: MIT Press, 1966.

[27] R. L. Miller, L. J. Deutsch, and S. A. Butman. *On the Error Statistics of Viterbi Decoding and the Performance of Concatenated Codes*. JPL Publication 81-9. Pasadena, California: Jet Propulsion Laboratory, September 1, 1981.

[28] K.-M. Cheung and S. J. Dolinar, Jr. "Performance of Galileo's Concatenated Codes With Nonideal Interleaving." *TDA Progress Report* 42-95, July-September 1988 (November 15, 1988): 148-152. <http://tmo.jpl.nasa.gov/tmo/progress_report/42-95/95N.PDF>

[29] D. Divsalar. "A Simple Tight Bound on Error Probability of Block Codes with Application to Turbo Codes." *TMO Progress Report* 42-139, July-September 1999 (November 15, 1999): 1-35. <http://tmo.jpl.nasa.gov/tmo/progress_report/42-139/139L.pdf>

[30] R. Garello, P. Pierleoni, and S. Benedetto. "Computing the Free Distance of Turbo Codes and Serially Concatenated Codes with Interleavers: Algorithms and Applications." *Journal on Selected Areas in Communications* 19, no. 5 (May 2001): 800-812.

[31] L. Deutsch, F. Pollara, and L. Swanson. "Effects of NRZ-M Modulation on Convolutional Codes Performance." *TDA Progress Report* 42-77, January-March 1984 (May 15, 1984): 33-40. <http://tmo.jpl.nasa.gov/tmo/progress_report/42-77/77E.PDF>

[32] Gian Paolo Calzolari, et al. "Turbo Code Applications on Telemetry and Deep Space Communications." In *Turbo Code Applications: A Journey from a Paper to Realization*, edited by Keattisak Sripimanwat, 321-344. Dordrecht: Springer, 2005.

The latest issues of CCSDS documents may be obtained from the CCSDS Secretariat at the address indicated on page i.

2   OVERVIEW OF CCSDS TELEMETRY SYSTEM

2.1   INTRODUCTION

The purpose of a telemetry system is to reliably and transparently convey measurement information from a remotely located data generating source to users located in space or on Earth. Typically, data generators are scientific sensors, science housekeeping sensors, engineering sensors and other subsystems on-board a spacecraft.

The advent of capable microprocessor based hardware will result in data systems with demands for greater throughput and a requirement for corresponding increases in spacecraft autonomy and mission complexity. These facts, along with the current technical and fiscal environments, create a need for greater telemetering capability and efficiency with reduced costs.

In the past, most of the telemetry resources used by a science mission have been wholly contained within a cognizant Project office and, with the exception of the tracking network, are completely dedicated to that mission. The lack of effective standardization among various missions forces the 'multi-mission' tracking network to implement the lowest level of telemetry transport service, i.e., bit transport. Higher level data delivery services, oriented more toward computer-to-computer transfers and typical of modern day commercial and military networks, had to be custom designed and implemented on a mission-to-mission basis.

The intent of the CCSDS Telemetry System is not only to ease the transition toward greater automation within individual space agencies, but also to ensure harmony among the agencies, thereby resulting in greater cross-support opportunities and services.

The CCSDS Telemetry System is broken down into two major conceptual categories: a 'TM Space Data Link Protocol' concept (references [2] and [7]) and a 'TM Synchronization and Channel Coding' concept (reference [3]).

a)   *TM Space Data Link Protocol* is a concept which facilitates the transfer of space-acquired data from source to user in a standardized and highly automated manner. TM Space Data Link Protocol provides a mechanism for implementing common data structures and protocols which can enhance the development and operation of space mission systems. TM Space Data Link Protocol addresses the following two processes:

1)   The end-to-end transport of space mission data sets from source application processes located in space to distributed user application processes located in space or on Earth.

2)   The intermediate transfer of these data sets through space data networks; more specifically, those elements which contain spacecraft, radio links, tracking stations and mission control centers as some of their components.

The *TM Space Data Link Protocol* Recommended Standard contained in references [2] and [7] is primarily concerned with describing the telemetry formats which are generated by spacecraft in order to execute their roles in the above processes.

b) *TM Synchronization and Channel Coding* (reference [3]) is a method by which data can be sent from a source to a destination by processing it in such a way that distinct messages are created which are easily distinguishable from one another. This allows reconstruction of the data with low error probability, thus improving the performance of the channel. The *TM Synchronization and Channel Coding* Recommended Standard contained in reference [3] describes several space TM Synchronization and Channel Coding schemes. The characteristics of the codes are specified only to the extent necessary to ensure interoperability and cross-support.

Together, TM Space Data Link Protocol and TM Synchronization and Channel Coding services provide to the user reliable and transparent delivery of telemetry information.

Figure 2-1 illustrates the CCSDS Telemetry System in terms of a layered service model. It should be noted that the CCSDS *TM Space Data Link Protocol* and *TM Synchronization and Channel Coding* Recommended Standards only address the five lower layers of this model.

LAYER	SERVICE PROVIDED BY LAYER
APPLICATION PROCESS LAYER	PROVIDES USERS A METHOD TO INVESTIGATE PHYSICAL PHENOMENA BY USING THEIR INSTRUMENTS IN SPACE FOR DATA COLLECTION AND THEIR APPLICATION PROCESSES FOR ANALYSIS.
PHYSICAL MEASUREMENTS	
SYSTEM MGMT LAYER	PROVIDES TRANSLATION OF PHYSICAL MEASUREMENTS INTO SETS OF APPLICATION DATA UNITS.
TLM APPLICATION DATA	
PACKETIZATION LAYER	PROVIDES END-TO-END DELIVERY OF APPLICATION DATA UNITS.
PACKET	
SEGMENTATION LAYER	(OPTIONAL) PREPARES LONGER PACKETIZED DATA UNITS FOR MULTIPLEXING AND TRANSFER THROUGH A SPACE DATA CHANNEL.
SEGMENT	
TRANSFER LAYER	PROVIDES RELIABLE TRANSFER OF PACKETS AND SEGMENTS IN A COMMON STRUCTURE FOR THEIR TRANSPORT THROUGH THE SPACECRAFT-TO-GROUND COMMUNICATION LINK.
TRANSFER FRAME	
CODING LAYER	PROTECTS TRANSFER FRAMES AGAINST ERRORS INDUCED DURING TRANSMISSION THROUGH THE NOISY PHYSICAL COMMUNICATIONS CHANNEL.
TLM BIT STREAM	
PHYSICAL LAYER	PROVIDES THE PHYSICAL CONNECTION, VIA RADIO FREQUENCY SIGNALS, BETWEEN A TRANSMITTING SPACECRAFT AND THE RECEIVING STATION.
PHYSICAL WAVEFORM	

Figure 2-1: Layered Telemetry Service Model

2.2 TELEMETRY SYSTEM CONCEPT

2.2.1 GENERAL

The system design technique known as layering was found to be a very useful tool for transforming the Telemetry System concept into sets of operational and formatting procedures. The layering approach is patterned after the International Organization for Standardization's Open Systems Interconnection layered network model (reference [4]), which is a seven layer architecture that groups functions logically and provides conventions for connecting functions at each layer. Layering allows a complex procedure such as the telemetering of spacecraft data to the users to be decomposed into sets of peer functions residing in common architectural strata.

Within each layer, the functions exchange data according to established standard rules or 'protocols'. Each layer draws upon a well defined set of services provided by the layer below, and provides a similarly well defined set of services to the layer above. As long as these service interfaces are preserved, the internal operations within a layer are unconstrained and transparent to other layers. Therefore, an entire layer within a system may be removed and replaced as dictated by user or technological requirements without destroying the integrity of the rest of the system. Further, as long as the appropriate interface protocol is satisfied, a customer (user) can interact with the system/service at any of the component layers. Layering is therefore a powerful tool for designing structured systems which change due to the evolution of requirements or technology.

A companion standardization technique that is conceptually simple, yet very robust, is the encapsulation of data within an envelope or 'header'. The header contains the identifying information needed by the layer to provide its service while maintaining the integrity of the envelope contents.

2.2.2 PACKETIZATION LAYER

Within *TM Space Data Link Protocol*, spacecraft generated application data are formatted into end-to-end transportable data units called 'TM Source Packets'. These data are encapsulated within a primary header which contains identification, sequence control and packet length information. A TM Source Packet is the basic data unit telemetered to the user by the spacecraft and generally contains a meaningful quantity of related measurements from a particular source.

2.2.3 TRANSFER FRAME LAYER

The *TM Transfer Frame* is used to reliably transport Source Packets (and Segments) through the telemetry channel to the receiving telecommunications network. As the heart of the CCSDS Telemetry System, the TM Transfer Frame protocols offer a range of delivery service options. An example of such a service option is the multiplexing of TM Transfer Frames into 'Virtual Channels' (VCs).

The TM Transfer Frame is a fixed length unit which was chosen to improve the ability to synchronize the frame with weak signals such as those found on space-ground links, and for compatibility with certain block oriented channel coding schemes. The (primary) header contains frame identification, channel frame count information and frame data field status information. An attached synchronization marker (ASM) signals the start of the TM Transfer Frame.

The transfer frame *data field* may be followed by an optional trailer containing an operational control field and/or a frame error control field. The first of these fields provides a standard mechanism for incorporating a small number of real-time functions (e.g., telecommand verification or spacecraft clock calibration). The error control field provides the capability for detecting errors which may have been introduced into the frame during the data handling process.

The delivery of transfer frames requires the services provided by the lower layers (e.g., carrier, modulation/detection, and coding/decoding) to accomplish its role.

2.2.4 CHANNEL CODING LAYER

TM Synchronization and Channel Coding is used to protect the transfer frames against telemetry channel noise-induced errors. Reference [3] describes the CCSDS Recommended Standard for *TM Synchronization and Channel Coding*, including specification of a convolutional code, a Reed-Solomon block-oriented code, a concatenated coding system consisting of a convolutional inner code and a Reed-Solomon outer code, and of turbo codes. The basic data units of the CCSDS TM Synchronization and Channel Coding which interface with the physical layer below are the Channel Symbols output by the channel encoder.

The RF channel physically modulates the channel symbols into RF signal patterns. Within the error detecting and correcting capability of the channel code chosen, errors which occur as a result of the physical transmission process may be detected and corrected by the receiving entity.

Full advantage of all CCSDS Telemetry System services could be realized if a Project complied with all CCSDS Recommended Standards. Alternatively, Projects can interface with any layer of the Telemetry System as long as they meet the interface requirements as specified in the Recommended Standards (references [2], [3], and [5]).

Figure 2-2 illustrates how the various telemetry data structures map into one another. There is presently no attempt to define the data structures of the top two layers of the telemetry system; i.e., the Application Process layer and the System Management layer. The Source Packets are placed into the data field of the Transfer Frame. An attached synchronization marker is always used, as shown in figure 2-2.
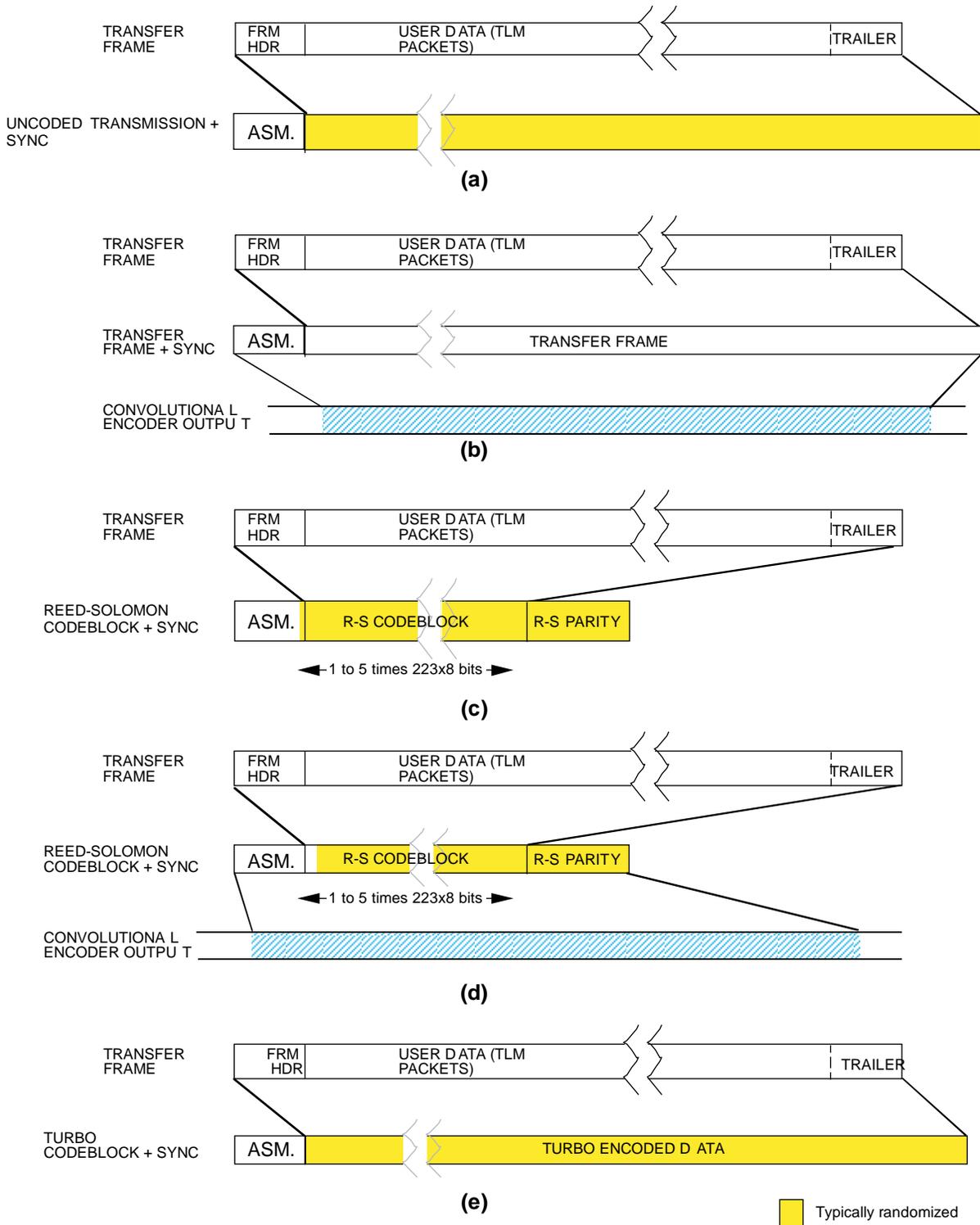
(a) Uncoded transmission
(b) Convolutional code only
(c) Reed-Solomon code only
(d) Concatenated Reed-Solomon and convolutional
(e) Turbo code

Figure 2-2: Telemetry Data Structures

3   TM SYNCHRONIZATION AND CHANNEL CODING

3.1   OVERVIEW

This section describes the CCSDS TM Synchronization and Channel Coding systems, and is divided into the four main coding methods: Convolutional Code, Reed-Solomon Code, Concatenated Code, and Turbo Codes.

3.2   INTRODUCTION

Channel coding[1] is a method by which data can be sent from a source to a destination by processing data so that distinct messages are easily distinguishable from one another. This allows reconstruction of the data with low error probability.

In spacecraft, the data source is usually digital, with the data represented as a string of zeroes and ones. A channel encoder (or simply 'encoder') is then a device that takes this string of binary data and produces a modulating waveform as output. If the channel code is chosen correctly for the particular channel in question, then a properly designed decoder will be able to reconstruct the original binary data even if the waveforms have been corrupted by channel noise. If the characteristics of the channel are well understood, and an appropriate coding scheme is chosen, then channel coding provides higher overall data throughput at the same overall quality (bit error rate) as uncoded transmission - but with less energy expended per information bit. Equivalently, channel coding allows a lower overall bit error rate than the uncoded system using the same energy per information bit.

There are other benefits that may be expected from coding. First, the resulting 'clean' channel can benefit the transmission of compressed data. The purpose of data compression schemes is to map a large amount of data into a smaller number of bits. Adaptive compressors will continually send information to direct a ground decompressor how to treat the data that follows. An error in these bits could result in improper handling of subsequent data. Consequently, compressed data is generally far more sensitive to communication errors than uncompressed data. The combination of efficient low error rate channel coding and sophisticated adaptive data compression can result in significant improvement in overall performance (reference [6]).

Second, a low bit error rate is also required when adaptive (or self-identified) telemetry is used. Adaptive telemetry is much like adaptive data compression in that information on how various ground processors should treat the transmitted data is included as part of the data. An error in these instructions could cause improper handling of subsequent data and the possible loss of much information.

---

[1]The method is called 'channel' coding because it is adapted to the statistical behavior of the channel and it applies to the overall transmitted data stream, not to specific sources only.

Third, low error probability telemetry may allow a certain amount of unattended mission operations. This is principally because the operations systems will know that any anomalies detected in the downlink data are extremely likely to be real and not caused by channel errors. Thus, operators may not be required to try to distinguish erroneous data from genuine spacecraft anomalies.

In a typical space channel, the principal signal degradations are due to the loss of signal energy with distance, and to the thermal noise in the receiving system. The codes described in reference [3] can usually provide good communication over this channel.

3.3 RECOMMENDED CODES

If interagency cross support requires one agency to decode the telemetry of another, then the codes recommended in reference [3] should be used. The recommended codes consist of: a constraint length 7, rate 1/2 convolutional code, and various punctured versions of it; (255,223) and (255,239) Reed-Solomon codes and arbitrary shortenings of them; codes formed by concatenating any of the recommended Reed-Solomon codes with any of the recommended convolutional codes; and a series of turbo codes of different rates and block sizes. A block diagram of the recommended coding system using concatenated codes appears in figure 3-1. A block diagram of the recommended coding system using turbo codes appears in figure 3-2.
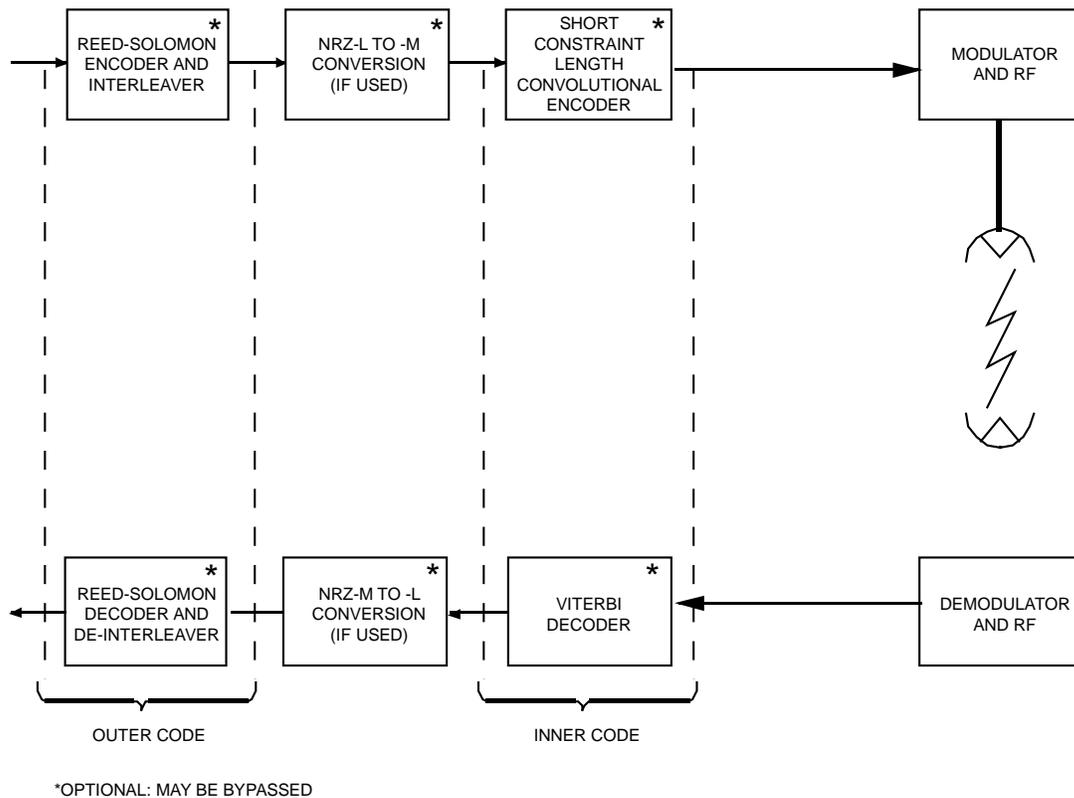
Figure 3-1: Coding System Block Diagram: Concatenated Codes
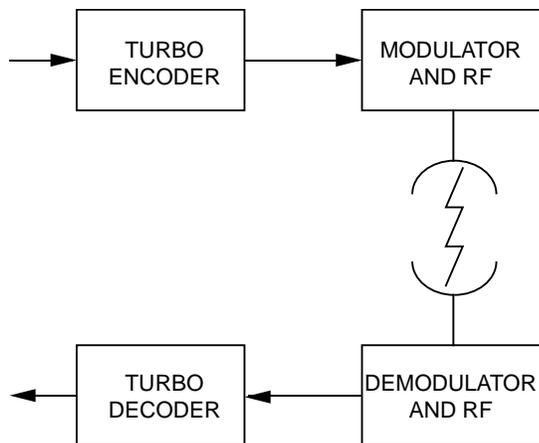
Figure 3-2: Coding System Block Diagram: Turbo Codes

These codes are included in the CCSDS Recommended Standard because they provide substantial coding gain over an uncoded system. They have already been incorporated, or are planned to be incorporated, into nearly all missions of member agencies of the CCSDS.

3.4   CHANNEL CODING PERFORMANCE

3.4.1   MEASURES OF PERFORMANCE

Performance of any channel code is measured by its error rate, relative to the amount of resources required to make the channel good enough to achieve that error rate. This Green Book shows the performance of the recommended codes on the additive white Gaussian (AWGN) channel, for which the relevant measure of required channel resources is given by a single parameter E_b/N_0, the ratio of the received signal energy per information bit to the (one-sided) spectral density of the white Gaussian noise. This channel parameter E_b/N_0 is commonly called the bit signal-to-noise ratio, or bit-SNR.

The error rates achieved by the recommended codes are measured and reported in this Green Book in three different ways. The bit error rate (BER) measures the error rate for individual bits; the word error rate (WER) measures the error rate for individual codewords;[2] and the frame error rate (FER) measures the error rate for individual frames. These three error rates are well correlated with each other for any given code, but one error rate cannot generally be derived from another without an assumption of independence of errors. As an example, if a frame comprises L independent bits, then FER $= 1 - (1 - \text{BER})^L$; this assumption is valid for uncoded frames on the AWGN channel, but not for frames subjected to any of the nontrivial recommended coding schemes.

---

[2]There is a slight impreciseness in this definition of WER. The output of a decoder is generally an estimate of the information bits that were encoded, not an estimate of the actual encoded codeword. Such a decoder makes a 'codeword error' when at least one of its decoded information bits is incorrect. This interpretation is consistent with the term 'codeword error' because re-encoding the information sequence will produce the correct codeword if and only if the entire sequence of information bits is correct.

In some cases, some of these error rates are synonymous or uninformative. For example, WER=BER for uncoded data because in this case each 'codeword' consists of one bit. Similarly, FER=WER for CCSDS turbo codes, because in this case the CCSDS transfer frame consists of the information bits from one turbo codeblock. A codeword for unterminated convolutional codes is theoretically infinitely long, so WER=1 (except on an error-free channel) and thus WER is not a very interesting measure of performance in this case. It is natural to define WER for terminated convolutional codes. Even for unterminated convolutional codes it is valid to compute FER on a segment (defining the frame) of the convolutional codeword.

3.4.2 FUNDAMENTAL LIMITS ON CODE PERFORMANCE

Good channel codes lower the error rate in the data, or equivalently they can achieve desired error rates more efficiently as a function of the bit-SNR E_b/N_0 on the channel. Shannon (see reference [8]) derived fundamental limits on the performance of all codes. There are code-rate-dependent channel capacity limits on the minimum E_b/N_0 required for reliable communication that are theoretically achievable by codes of a given rate in the limit of infinite block sizes. In addition, there are block-size-dependent limits that preclude capacity-attaining performance when the code's block size is also constrained.

Code-Rate-Dependent Capacity Limits — Figure 3-3 shows the Shannon-limit performance curves for a binary-input additive white Gaussian noise (AWGN) channel for rates 1/6, 1/4, 1/3, and 1/2. These curves show the lowest possible bit-energy-to-noise ratio E_b/N_0 required to achieve a given BER over the binary-input AWGN channel using codes of these rates.
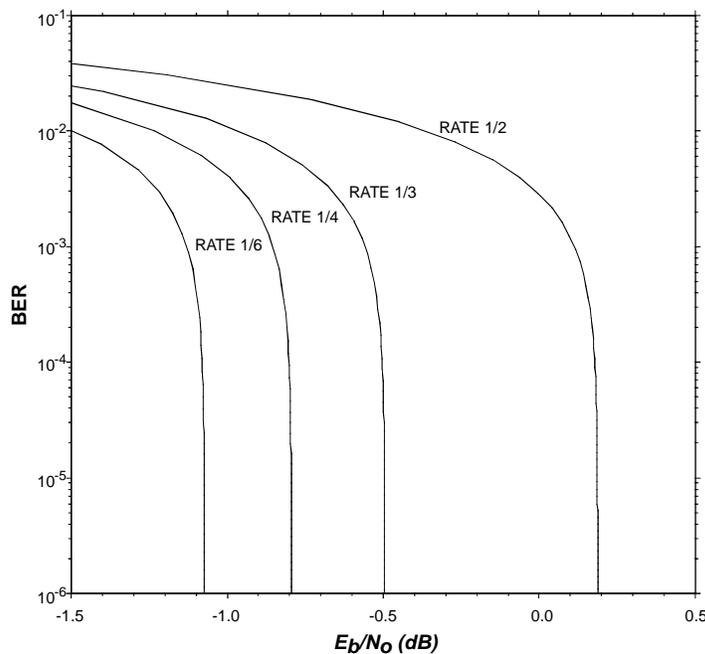
Figure 3-3: Capacity Limits on the BER Performance for Codes with Rates 1/2, 1/3, 1/4 and 1/6 Operating over a Binary Input AWGN Channel

For low BER, each of these capacity-limited performance curves approaches a vertical asymptote dependent on the code rate. The asymptotes are at 1.1 dB for rate 2/3, 0.2 dB for rate 1/2, -0.5 dB for rate 1/3, and -0.8 dB for rate 1/4. The vertical asymptote for the ultimate Shannon limit on performance (i.e., rate $\rightarrow$0) is -1.6 dB. A comparison of these limits shows the improvement that is theoretically possible as a result of lowering the code rate. For example, for a binary-input AWGN channel, rate-1/2 codes suffer an inherent 0.7 dB disadvantage relative to rate-1/3 codes, a 1.0 dB disadvantage relative to rate-1/4 codes, and a 1.8 dB disadvantage relative to the ultimate limit (rate $\rightarrow$0).

Block-Size-Dependent Limits on Code Performance — Just as a constraint on code rate raises the minimum threshold for reliable communication above the ultimate unconstrained capacity limit, so does a constraint on codeblock length. The theoretical limits shown in figure 3-3 assume no constraint on block size. Approaching these limits requires that block sizes grow arbitrarily large.

Figure 3-4 shows some classic Shannon sphere packing lower bounds on the performance of arbitrary codes of a given block size and code rate on the additive white Gaussian noise channel with unconstrained input (i.e., not necessarily binary-input as in figure 3-3). The curves labeled 'bound' are the block-size-dependent bounds for each code rate. The horizontal asymptotes labeled 'capacity' are the rate-dependent capacity limits. These asymptotes are slightly different from the vertical asymptotes in figure 3-3 because they represent capacity limits for an unconstrained-input channel instead of a binary-input channel.
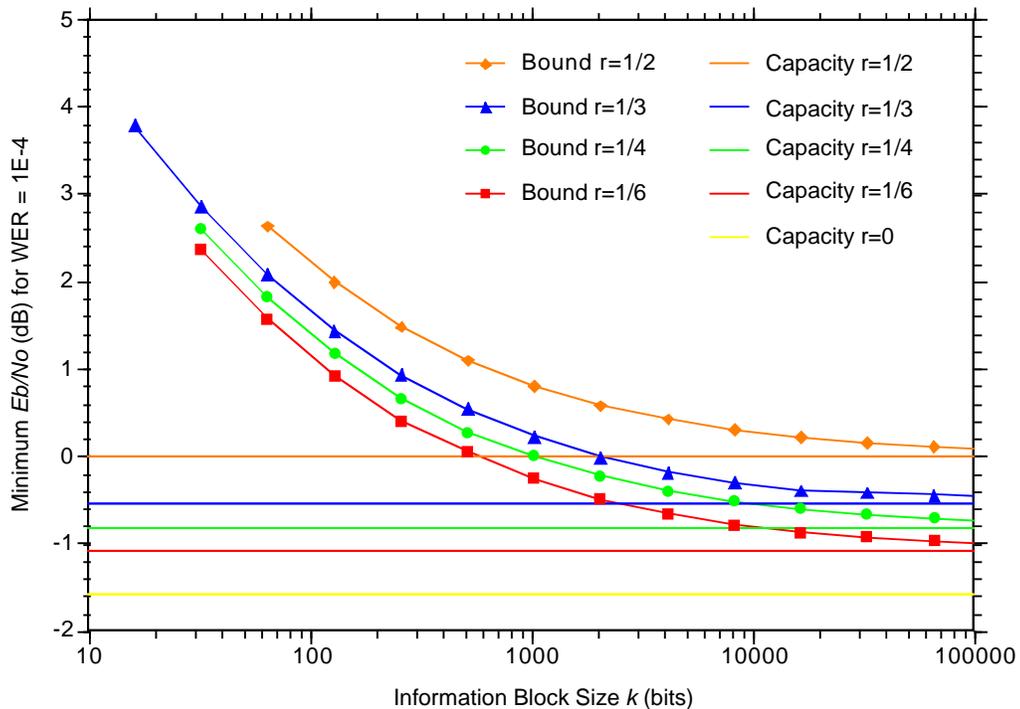
Figure 3-4: Shannon Sphere-Packing Lower Bounds on the WER Performance for Codes with Varying Information Block Length k and Rates 1/6, 1/4, 1/3, 1/2, Operating over an Unconstrained-Input AWGN Channel

This figure shows that, for any given code rate, the minimum threshold for reliable communication is significantly higher than the corresponding ultimate limit for that code rate, if the codeblock length is constrained to a given finite size. For example, 1000-bit blocks have an inherent advantage of about 1.3 dB compared to 100-bit blocks for each of the four code rates plotted. An additional gain of just over 0.5 dB is potentially obtained by going from 1000-bit blocks to 10000-bit blocks, and another 0.2 dB by going to 100000-bit blocks. After that, there is less than another 0.1 dB of improvement available before the ultimate capacity limit for unlimited block sizes is reached.

3.4.3   EXAMPLES OF PERFORMANCE OF RECOMMENDED CODES

The relative performance of various recommend (non-punctured, non-shortened) codes on a Gaussian channel is shown in figure 3-5. Here, the input is constrained to be chosen from between two levels, because biphase modulation is assumed throughout the Recommended Standard.[3] These performance data were obtained by software simulation and assume that there are no synchronization losses (see reference [10] for a discussion on the effect of receiver tracking losses). The channel symbol errors were assumed to be independent: this is a good assumption for the deep space channel, and an approximation for near-Earth links which ignores impulsive noise and RFI. In this introductory comparison of code performance, infinite interleaving is assumed in the concatenated code and bit error rate (BER) only is used. Specific results with finite interleaving depth are given in 6.3; results for frame error rate (FER) are given in later Sections discussing specific codes. It is clear from the figure that the convolutional code offers a coding gain of about 5.5 dB over an uncoded system at decoded bit error rate of 10^{-5}. Concatenation of this code with the outer Reed-Solomon code results in an additional 2.0 dB of coding gain. Turbo codes can provide even higher coding gains, as illustrated in the figure for the turbo code with rate 1/2 and block size 8920 bits. This code approaches within 1 dB the ultimate Shannon limit for codes with rate 1/2 and improves on the recommended concatenated code's performance by about 1.5 dB.

These codes are included in the CCSDS Recommended Standard because they provide substantial coding gain over an uncoded system. They have already been incorporated, or are planned to be incorporated, into nearly all missions of member agencies of the CCSDS.

The next four sections describe the parameters and the performance of each recommended code in more detail, along with brief descriptions of their encoder and decoder realizations.

---

[3] Biphase modulation is appropriate for power-limited links, where bandwidth efficiency is not particularly important.
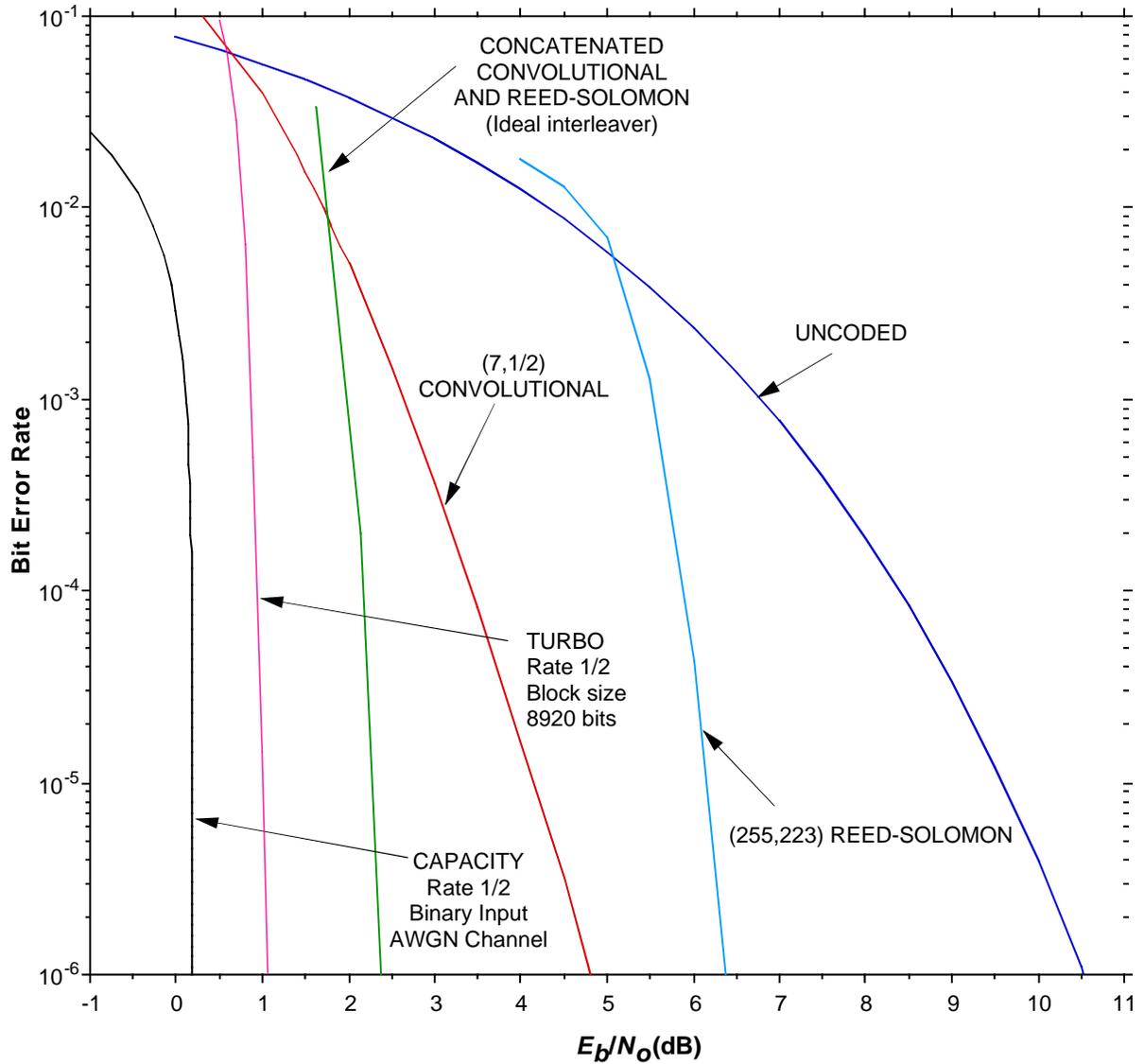
Figure 3-5: Performance Comparison of Selected Convolutional, Reed-Solomon, Concatenated, and Turbo Codes

4 CONVOLUTIONAL CODES

4.1 INTRODUCTION

A rate $r=1/n$ convolutional encoder is a linear finite-state machine with one binary input, n outputs and an m-stage shift register, where m is the *memory* of the encoder. Such a finite-state encoder has 2^m possible states. The *constraint length K* of the convolutional code is defined as $K=m+1$, and the code is referred to as a $(K,1/n)$ code. In comparison to block codes, convolutional codes encode the input data bits continuously rather than in blocks.

In general, a rate $r=l/n$ convolutional encoder is a linear finite-state machine with l binary inputs and n binary outputs. A rate $r=l/n$ code can also be produced by puncturing a convolutional code of rate $r=1/n$.

4.2 ENCODER FOR THE (7,1/2) RECOMMENDED CODE

A (7,1/2) convolutional code selected for space applications in the 1970s was a standout performer for its time. Exhaustive search over all convolutional codes with $r=1/2$ and $K\leq7$ found that only this code (not counting a few symmetric equivalents) was able to achieve a *free distance* $d_{\mathrm{free}}=10$. By comparison, the best (6,1/2) code can only achieve $d_{\mathrm{free}}=8$, and the best (8,1/2) code can only match the recommended (7,1/2) code's $d_{\mathrm{free}}=10$. Maximizing the free distance was an important consideration because a convolutional code's bit error rate with maximum likelihood decoding falls off exponentially with d_{free} at low error rates. It was also important to achieve a good d_{free} at a reasonably low value of constraint length K, because every unit increase in K doubles the number of encoder states and therefore doubles the complexity of maximum likelihood decoding. Existing technology at the time this code was selected allowed maximum likelihood decoding of convolutional codes with constraint length $K=7$ but not much higher. Thus, the recommended code was an obvious local optimum based on its d_{free}.

Convolutional codes with longer constraint lengths than $K=7$ were also used in the early days of space applications, but never standardized. Maximum likelihood decoding of these codes was infeasible; instead they were decoded by *sequential decoding* at a significant penalty in performance.

The recommended (7,1/2) code has another feature that makes it useful for space applications: it is *transparent*. Transparency means that at steady-state, if the input sequence to the encoder is inverted the output will be inverted also. Similarly, if the input sequence to the decoder is inverted, at steady-state the output sequence of the decoder will be inverted too. This feature is useful because with BPSK modulation there is often a 180-degree phase ambiguity, and the demodulator can produce the inverse of the transmitted symbols even when it is in lock. With a transparent code, when the demodulator produces the inverse of the transmitted symbols, the decoder produces the inverse of the encoded bits. Since packetized telemetry includes various known headers, it is easy to recognize if the decoded bits have been inverted and to invert them back if necessary.

A diagram of an encoder for the recommended convolutional code of rate 1/2 and K=7 is shown in figure 4-1. The particular encoder structure depends on the manner in which the adders are connected to the shift register. These connections are denoted by a set of vectors

$$\mathbf{g}_i = (g_{i,1}, g_{i,2}, \cdots, g_{i,m}) \; i = 1, 2, \cdots, n \tag{1}$$

where g_{il}= 1 denotes a connection between the ith stage of the shift register and the lth adder, and g_{il}= 0 denotes the absence of a connection. The complete set of the $\mathbf{g}_i$s defines the code.
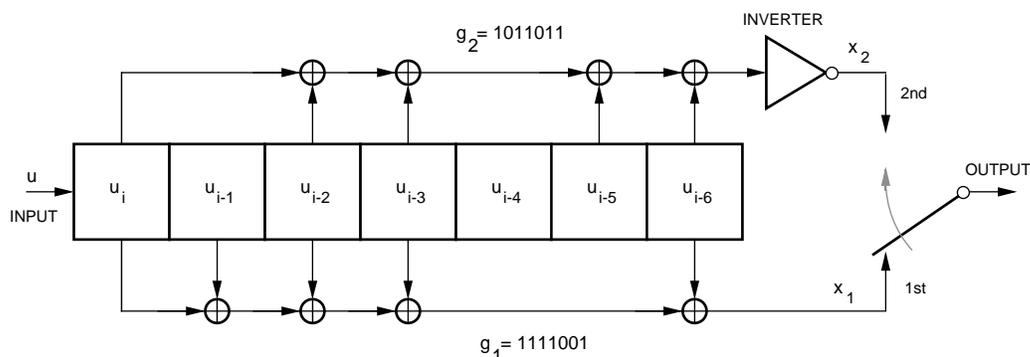
Figure 4-1: Example of Convolutional Encoder: Constraint Length K=7, Rate 1/2, CCSDS Standard Convolutional Code

The encoder for the CCSDS standard code is extremely simple, as shown in figure 4-1. It consists of a shift register and some exclusive OR gates that implement the two parity checks. The two checks are then multiplexed into one line. This means that the encoder can be made small and that it dissipates very little power. These are good attributes for spacecraft hardware.

It has been customary to invert one or the other parity check in the encoder. This operation makes the recommended code into a coset of a pure linear convolutional code. The inversion is performed to ensure that there are sufficient transitions in the channel stream for the symbol synchronizer to work in the case of a steady state (all zeroes or all ones) input to the encoder.[4] Although alternate symbol inversion may increase or decrease the average transition density, depending on the data source model, it does limit the number of contiguous symbols without transition for a particular class of convolutional codes, independent of the data source model. Further, this limit is sufficiently small to guarantee acceptable symbol synchronizer performance for typical applications. The maximum number of contiguous symbols without transition for the convolutional code of figure 4-1 is 14.

Historically, ESA, NASA-GSFC and NASA-JPL have each used a different ordering of the two parity checks or has inverted a different parity check. Performance is not affected by these minor differences. But, to reduce the number of options, CCSDS has adopted only one convolutional code for cross-support: all agencies are encouraged to adopt for all facilities the single convention described in reference [3].

_____

[4] See further discussion in section 8.

4.3 ENCODER FOR THE RECOMMENDED PUNCTURED CONVOLUTIONAL CODES

The CCSDS standard convolutional code, with constraint length K=7, has rate 1/2. The code rate can be increased by using a puncturing pattern, thus achieving an increase in bandwidth efficiency. Puncturing removes some of the encoded symbols before transmission, leading to a higher code rate and a lower bandwidth expansion than the original code, but with reduced error correcting performance. A block diagram of the punctured encoder is shown in figure 4-2.
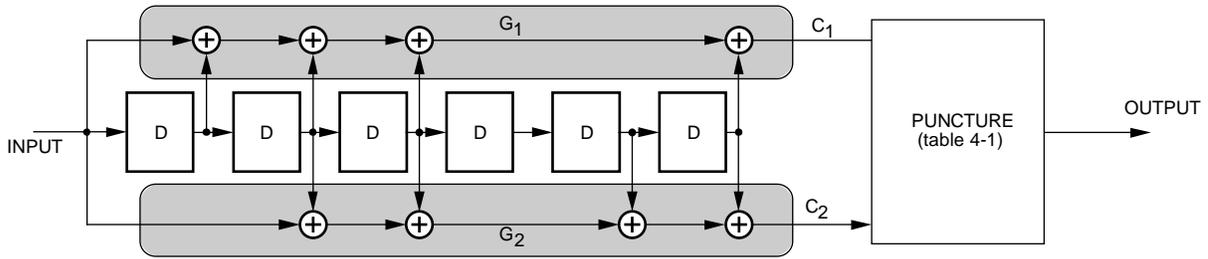
Figure 4-2: Encoder Block Diagram for the Punctured CCSDS Convolutional Codes

Starting from the CCSDS rate-1/2 convolutional code, the recommended punctured codes are obtained with fixed puncturing patterns yielding code rates 2/3, 3/4, 5/6 and 7/8, as reported in table 4-1.

Table 4-1: Puncturing Patterns for the CCSDS Punctured Convolutional Code Rates

Puncturing Pattern 1 = transmitted symbol 0 = non-transmitted symbol	Code Rate	Output $C_1(t)$, $C_2(t)$ denote values at bit time t (t=1,2,3,...)
C_1: 1 0 C_2: 1 1	2/3	$C_1(1)\ C_2(1)\ C_2(2)$ ...
C_1: 1 0 1 C_2: 1 1 0	3/4	$C_1(1)\ C_2(1)\ C_2(2)\ C_1(3)$ ...
C_1: 1 0 1 0 1 C_2: 1 1 0 1 0	5/6	$C_1(1)\ C_2(1)\ C_2(2)\ C_1(3)\ C_2(4)\ C_1(5)$ ...
C_1: 1 0 0 0 1 0 1 C_2: 1 1 1 1 0 1 0	7/8	$C_1(1)\ C_2(1)\ C_2(2)\ C_2(3)\ C_2(4)\ C_1(5)\ C_2(6)\ C_1(7)$ ...

4.4 SOFT MAXIMUM LIKELIHOOD DECODING OF CONVOLUTIONAL CODES

Soft, maximum likelihood decoding of convolutional codes can be accomplished by using the Viterbi algorithm (see references [21] and [25]), which will be illustrated for rate 1/n codes. The same decoding algorithm is applicable to both non-punctured and punctured codes, provided that the received symbol stream is 'depunctured' by inserting zero-symbols (i.e., neutral symbol values that do not favor either a received '0' or '1' bit) at the positions where encoded symbols were removed during the encoding of the punctured code.

Before proceeding to the Viterbi algorithm, a discussion of the trellis representation of the convolutional encoder is desirable. For a constraint length K, code rate $r=1/n$, (K, r) convolutional encoder, the state is defined by the $(K–1)=m$ most recent bits in the shift register. Figure 4-3 shows an encoder for a (3,1/2) convolutional code. (Note that this is just an illustrative example, and is not the CCSDS recommended code.) The output bits and transitions between states can be recorded by the trellis diagram of figure 4-4.
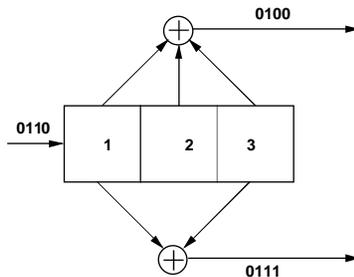
Figure 4-3: (3,1/2) Convolutional Encoder
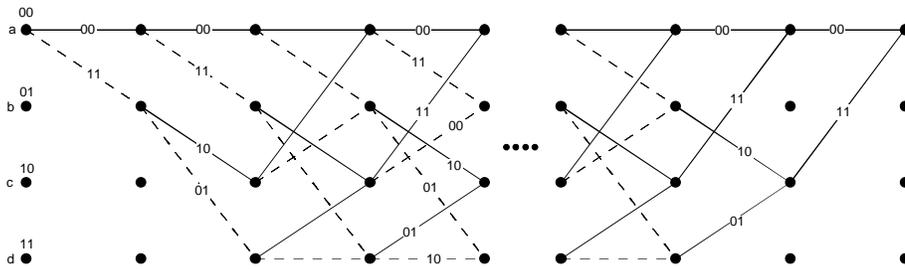
Figure 4-4: Trellis Representation of (3,1/2) Convolutional Code

The diagram starts in the all-zero state, node *a*, and makes transitions corresponding to the next data bit. These transitions are denoted by a solid line (branch) for a '0' and by a dotted line for a '1'. Thus node *a* proceeds to node *a* or *b* with outputs bits '00' or '11'. A branch weight is the number of '1's in the *n* code symbols in the branch.

It has been shown (see reference [25]) that the Viterbi algorithm implements, in fact, maximum-likelihood decoding. An exhaustive search maximum-likelihood decoder would calculate the likelihood of the received data for code symbol sequences on all paths through the trellis. The path with the largest likelihood would then be selected, and the information bits corresponding to that path would form the decoder output. Unfortunately, the number of paths for an L bit information sequence is 2^L; thus, this exhaustive search decoding quickly becomes impractical as L increases.

With Viterbi decoding, it is possible to greatly reduce the effort required for maximum-likelihood decoding by taking advantage of the special structure of the code trellis. Referring to figure 4-4, it is clear that the trellis assumes a fixed periodic structure after trellis depth K is reached.

The paths are said to have diverge d at some state, and some depth j, if at depth $j+1$, their information bit disagree. Later, paths can remerge after $(K-1)$ consecutive identical information bits. The maximum-likelihood sequence estimation problem is formally identical to the problem of finding the shortest route through a certain graph. The Viterbi algorithm then arises as a natural recursive solution. Consider a rate $1/n$ convolutional code. Let $u_0 \ldots u_{t-1}u_t u_{t+1} \ldots$ denote the information bits input to the encoder. At time t define the encoder state as

$$s_t = u_t \ldots u_{t-K+1} \qquad\qquad (2)$$

Given a sequence of observations $y_1, y_1, \ldots y_L$, where $y_1 = (y_{i1} \ldots y_{in})$, every path may be assigned a 'length' proportional to metric $-\log p(y|s)$, where $p(y|s)$ is the likelihood function and $s = (s_0, \ldots, s_L)$ is the state sequence associated with that path.

The Viterbi algorithm solves the problem of finding the state sequence for which $p(y|s)$ is maximum, or equivalently of finding the path whose length $-\log p(y|s)$ is minimum. Note that to every possible state sequence s there corresponds a unique path through the trellis, and vice versa. If the channel is memoryless, then

$$-\log p(y|s) = \sum_{t=1}^{L} \lambda(s_t, s_{t-1})$$

where

$$\lambda(s_t, s_{t-1}) = -\log p(y_t|s_t, s_{t-1}) = -\log p(y_t|s_t)$$

is the branch 'length' or metric. $T_t(s_t, s_{t-1})$ denotes the transition from state s_{t-1} to s_t associated with branch symbols $x_t = (x_{t1} \ldots x_{tn})$, which correspond to the information sequence

$$u_t \ldots u_{t-K}$$

Therefore, we can define the state transition as $T_t(s_t,s_{t-1})=u_t\ldots u_{t-K}$. We denote by $\mathbf{s}(s_t)$ a segment $(s_0, s_1, \ldots, s_t)$ consisting of the states up to time t of the state sequence $\mathbf{s}$. In the trellis, $\mathbf{s}(s_t)$ corresponds to a path segment starting at the state s_0 and terminating at state s_t. For any particular time t and state s_t, there will in general be several such path segments, each with some length

$$\lambda(\mathbf{s}(s_t)) = \sum_{i=1}^{t} \lambda(s_i, s_{i-1})$$

The shortest such path segment is called the *survivor*, corresponding to the state s_t, and is denoted $\hat{\mathbf{s}}(s_t)$. For any time $t>0$, there are 2^m survivors in all, one for each s_t.

Thus at any time t we need remember only the 2^m survivors $\hat{\mathbf{s}}(s_t)$ and their lengths $\Gamma(s_t)=\lambda(\mathbf{s}(s_t))$. To get to time $t+1$, we need only extend all time t survivors by one time unit, compute the lengths of the extended path segments, and for each state s_{t+1} select the shortest extended path segment terminating in s_{t+1} as the corresponding time $t+1$ survivor. Recursion proceeds indefinitely without the number of survivors ever exceeding 2^m.

The great advantage of the Viterbi maximum-likelihood decoder is that the number of decoder operations performed in decoding L bits is only $L2^m$, which is linear in L. Of course, Viterbi decoding as a practical technique is limited to relatively short constraint-length codes due to the exponential dependence of decoder operations, per decoder operations, per decided bit, on K. Recent convolutional codes for deep space communications have used constraint lengths up to 15. Constraint lengths of 24, 32 and even 40 have been used in the past for sequential decoders which have suboptimal performance with respect to maximum-likelihood decoders.

4.5 PERFORMANCE OF THE RECOMMENDED (7,1/2) CONVOLUTIONAL CODE

Figure 4-5 shows the simulated bit error rate performance of the CCSDS rate-1/2 convolutional code. Different quantization strategies have been considered, from unquantized soft decision to hard decision (corresponding to 1-bit quantization). It is shown that 8-bit quantization provides nearly ideal performance (less than 0.2 dB penalty with respect to unquantized curves), while hard decision suffers a loss greater than 2 dB.
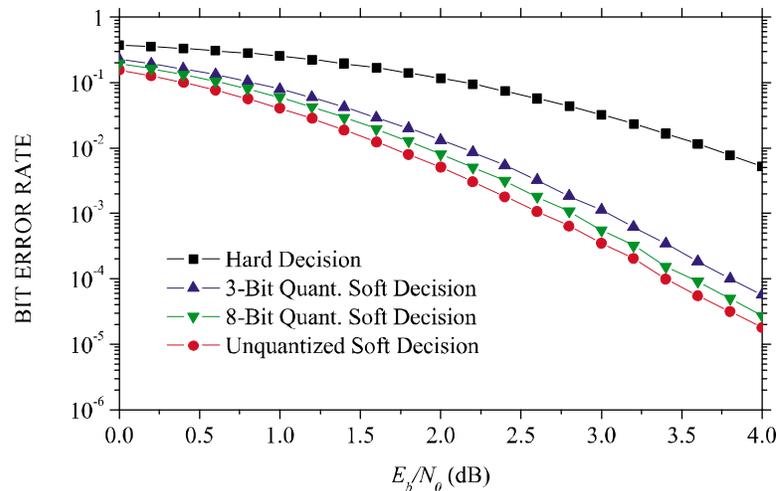
Figure 4-5: Bit Error Rate Performance of the CCSDS Rate 1/2 Convolutional Code with Different Quantizers

In principle, the Viterbi decoder should operate on the entire received sequence. This, however, would result in unacceptably long decoding delays (latency) and excessive memory storage for the survivor sequences. In fact, since all survivor paths tend to merge into one single path when exploring the trellis at sufficient depth, practical implementations use a truncated Viterbi algorithm that forces the decision on the oldest symbol of the minimum metric path after a fixed and sufficiently long delay or truncation length D. Computer simulations show that using a delay on the order of 5 times the constraint length (i.e., $D=5K$) is enough to obtain negligible degradations.

For the CCSDS rate-1/2 convolutional code, the dependence of the bit error rate on the decoding delay is shown in figure 4-6. Using a delay of only $D=30$ bits, i.e., 5 times the memory m, the performance exhibits a very small degradation. Using $D=60$ bits nearly optimum performance is obtained. (All the curves have been obtained with unquantized soft decision.)
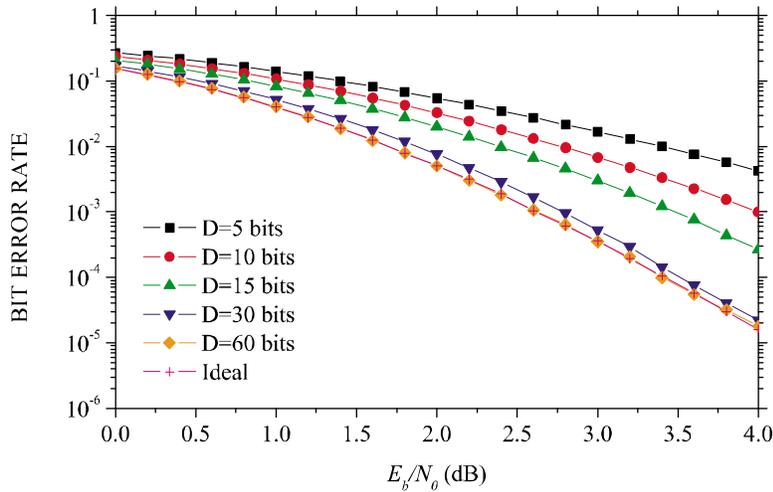
Figure 4-6: Bit Error Rate Performance of the CCSDS Rate-1/2 Convolutional Code with Different Decoding Delays D

Telemetry data are collected in packets and transmitted in frames (see reference [2]). In principle, any frame length L up to 16384 bits could be acceptable. In figure 4-7 the Frame Error Rate (FER) at the output of the Viterbi decoder is reported for different frame lengths corresponding to those used for the concatenated (Reed-Solomon (255,223) + convolutional code) CCSDS code. A frame is in error if any of its constituent bits is in error. These curves have been obtained with unquantized soft decision and decoding delay $D = 60$ bits. Since the Viterbi decoder's errors occur in bursts, the FER curves in figure 4-7 cannot be directly derived from the BER curve for $D = 60$ bits in figure 4-6 by assuming independent bit errors.
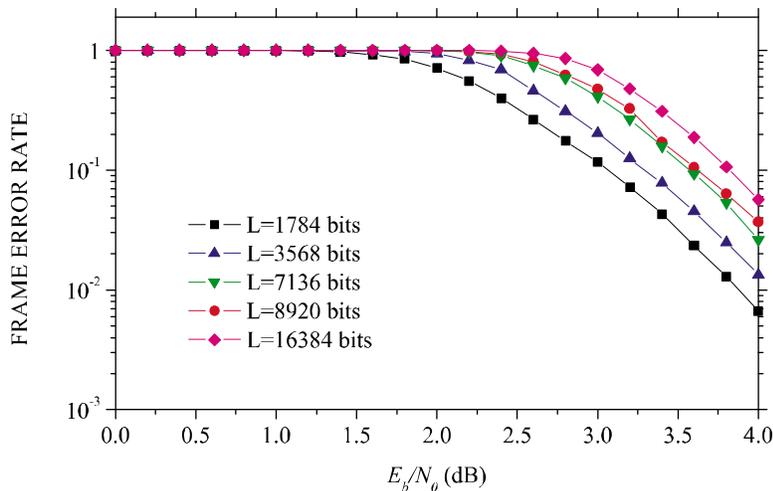
Figure 4-7: Frame Error Rate Performance of the CCSDS Rate-1/2 Convolutional Code with Different Frame Lengths and Decoding Delay D=60

4.6 PERFORMANCE OF THE RECOMMENDED PUNCTURED CONVOLUTIONAL CODES

The bit error rate performance of the CCSDS punctured convolutional codes is reported in figure 4-8. The curve relative to the non-punctured rate-1/2 CCSDS code is also reported for the sake of comparison. The expected performance degradation is confirmed (there is a gap of about 2.4 dB between the case of rate 1/2 and the case of rate 7/8), due to reduced bandwidth expansion. (All the curves have been obtained with unquantized soft decision and decoding delay equal to 60 bits.)

The frame error rate performance of the CCSDS punctured convolutional codes is reported in figure 4-9 for frame size 8920 bits.
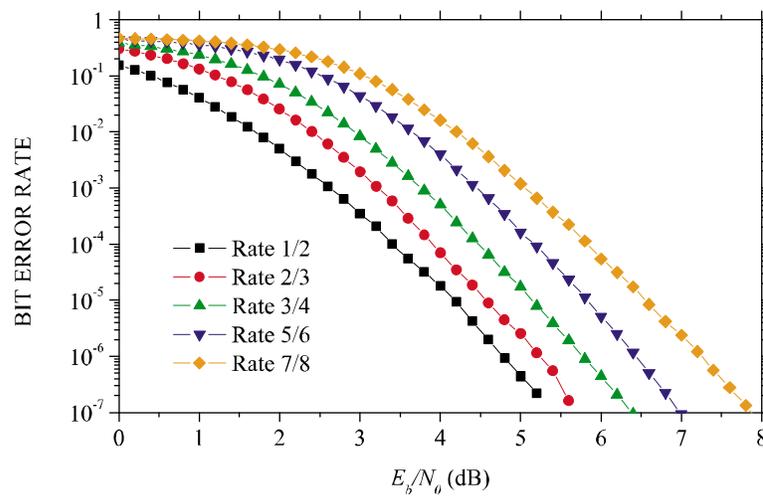
NOTE – The performance of the original rate 1/2 code is reported for comparison.

Figure 4-8: Bit Error Rate Performance of the CCSDS Punctured Convolutional Codes
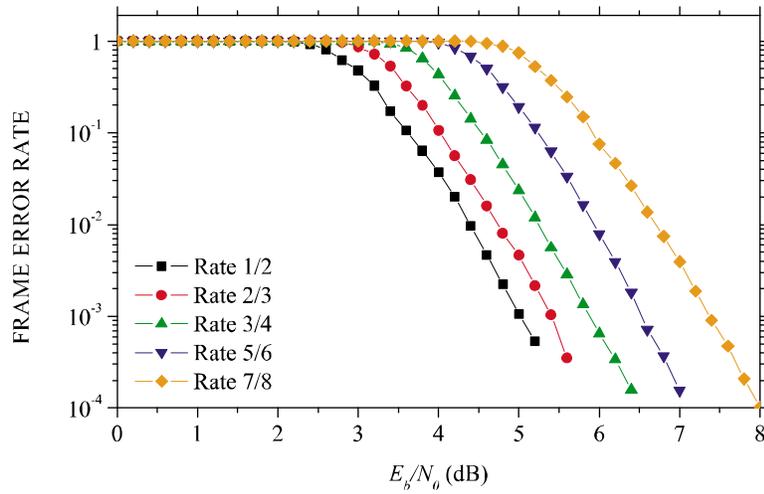
NOTE   –   The performance of the original rate 1/2 code is reported for comparison.

Figure 4-9: Frame Error Rate Performance of the CCSDS Punctured Convolutional Codes with Frame Length *L*=8920

5 REED-SOLOMON CODE

5.1 INTRODUCTION

Reed-Solomon (RS) codes (see reference [22]) are a particularly interesting and useful class of linear block codes. The block length n of an RS code is $q–1$, with $q = 2^J$ being the alphabet size of the symbols. RS codes with k information symbols and block length n have a minimum distance $d = n–k+1$. These codes have been used effectively in a concatenated code scheme (see section 6), where the symbols in an 'outer' RS code are further encoded by an 'inner' convolutional code. The error probability is an exponentially decreasing function of the block length, and the decoding complexity is proportional to a small power of $n–k$. Reed-Solomon codes can be used directly on a channel with a small input alphabet by representing each letter in a codeword by a sequence of channel letters. Such a technique is useful on channels where the errors are clustered, since the decoder operation depends only on the number of sequences of channel outputs that contain errors.

Using symbols with $q = 2^J$ for some J, the block length is $n = 2^J–1$. For an arbitrarily chosen odd minimum distance d, the number of information symbols is $k = n–d+1$ and any combination of $E = (d–1)/2 = (n–k)/2$ errors can be corrected. If we represent each letter in a codeword by J binary digits, then we can obtain a binary code with kJ information bits and block length nJ bits. Any noise sequence that alters at most E of these n binary J-tuples can be corrected, and thus the code can correct all bursts of length $J(E–1)+1$ or less, and many combinations of multiple shorter bursts. Therefore RS codes are very appropriate on burst noisy channels such as a channel consisting of a convolutional encoder-AWGN channel-Viterbi decoder. RS codes are less appropriate for direct application to the AWGN channel where their performance is poorer than that of convolutional codes (see figure 3-5).

The Reed-Solomon code, like the convolutional code, is a transparent code. This means that if the channel symbols have been inverted somewhere along the line, the decoders will still operate. The result will be the complement of the original data (except, usually, for the codeblock in which the inversion occurs). However, the Reed-Solomon code loses its transparency if virtual zero fill is used. For this reason it is mandatory that the sense of the data (i.e., true or complemented) be resolved before Reed-Solomon decoding, as specified in the Recommended Standard (reference [3]).

Two RS codes are recommended by CCSDS, both having codeblock size $n = 255$ symbols and symbol size $J = 8$ bits or alphabet size $2^J = 256$. The first code has information block size $k = 223$, minimum distance $d = 33$, and can correct $E = 16$ errors. The second code has $k = 239$, $d = 17$, and can correct $E = 8$ errors. The recommended RS codes are non-binary codes. Each member of the coding alphabet is one of 256 elements of a finite field rather than zero or one. A string of eight bits is used to represent each element in the field so that the output of the encoder still looks like binary data.

A Reed-Solomon symbol size of eight bits was chosen because the decoders for larger symbol sizes would be less suitable to implementation with current technology, and because telemetry transfer frames are octet-based. This choice forces the longest codeword length to

be 255 symbols. The recommended RS code with $E=16$ was chosen as this was shown to have the best performance when concatenated with the (7, 1/2) convolutional inner code (see references [9] and [11]). Since two check symbols are required for each symbol error to be corrected, this results in a total of 32 check symbols and 223 information symbols per codeword. The RS code with $E=8$ was added later to the Recommended Standard (reference [3]) to allow another coding option with higher code rate.

The same encoding and decoding hardware can implement a shortened $(n',n'-2E)$ Reed-Solomon code, where $n' = 33, 34, ... , 254$, as well as the non-shortened code with $n'=n=255$. This is accomplished by assuming that the remaining symbols are fixed: in the Recommended Standard (reference [3]), they are assumed to be all zero. This virtual zero fill allows the frame length to be tailored, if necessary, to suit a particular mission or situation. The shortened codes can correct the same number of errors (E) as the non-shortened code, but the overall code performance (energy efficiency per bit) generally (but not always) gets worse as the code rate is decreased due to shortening.

5.2 ENCODER

Reed-Solomon codes are block codes. This means that a fixed block of input data is processed into a fixed block of output data. In the case of the $(255,k)$ code, $k=255-2E$ Reed-Solomon input symbols (each eight bits long) are encoded into 255 output symbols. The Reed-Solomon code in the Recommended Standard (reference [3]) is systematic. This means that a portion of the codeword contains the input data in unaltered form. In the Recommended Standard (reference [3]), the first $k=223$ or 239 symbols are the input data for the two recommended codes, respectively.

A very simple block diagram of an (n,k) Reed-Solomon block encoder is shown in figure 5-1, where $n=2^J-1$ and $k=n-2E$. An RS symbol consists of a sequence of J bits so that there are 2^J possible RS symbols. All coding and decoding operations involve RS symbols, not individual bits. The input of the encoder consists of a block of $k=2^J-1-2E$ information symbols (or kJ information bits) from some data source. The result of the encoding operations is a codeword of length $n=2^J-1$ symbols, of which the first k are the same symbols as those entering to the left. This makes the code *systematic*. The remainder of the codeword is filled in with $2E$ *parity symbols*, where E is the number of correctable RS symbol errors in an RS codeword. An RS symbol is in error if one or more of the J bits making up the symbol are in error.
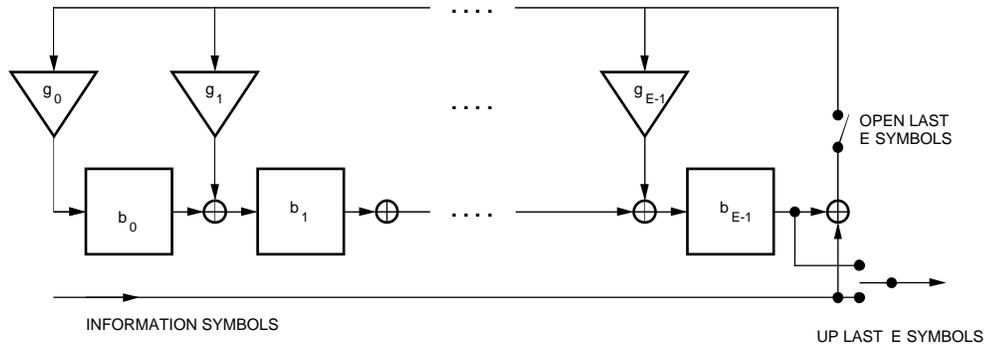
Figure 5-1: Block Diagram of an (*n*,*k*) Reed-Solomon Encoder

We put attention on the specific recommended RS code with $J=8$, $E=16$, i.e., the (255,223) code. The basic codeword structure of this specific code with $J=8$, $E=16$, is given in figure 5-2. If desired, a 'quick look' at the data (information bits) would still be possible since the code is systematic. Note that the overhead associated with the parity symbols is only around 15 percent. This percentage increases if the code is shortened.
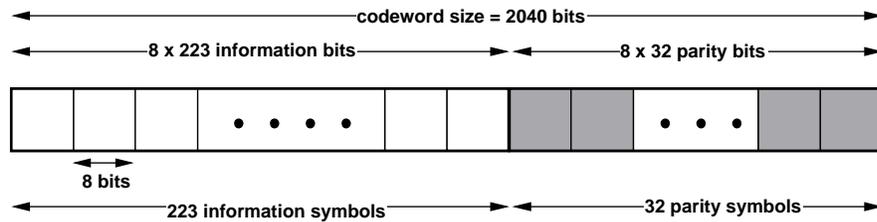
Figure 5-2: RS Codeword Structure, *J*=8, *E*=16

There are two polynomials that define each of the recommended Reed-Solomon codes in 4.2 (4) and (5) of reference [3] (also see reference [16]): a code generator polynomial over $GF(2^8)$ and a field generator polynomial over $GF(2)$. The field generator polynomial $F(x)=x^8+x^7+x^2+x+1$ is the same for both codes. The code generator polynomial $g(x)$ has degree $2E=32$ for the (255,223) code and degree $2E=16$ for the (255,239) code. The particular polynomials that define the recommended codes were chosen to minimize the encoder hardware. The code generator polynomials are palindromes (self-reciprocal polynomials) so that only half as many multipliers are required in the encoder circuits. The particular primitive element 'α' (and hence the field generator polynomial) was chosen to make these multipliers as simple as possible. An encoder using the 'dual basis' representation requires for implementation only a small number of integrated circuits or a single VLSI chip.

Figure 5-3 illustrates the construction of shortened RS codewords using virtual fill.
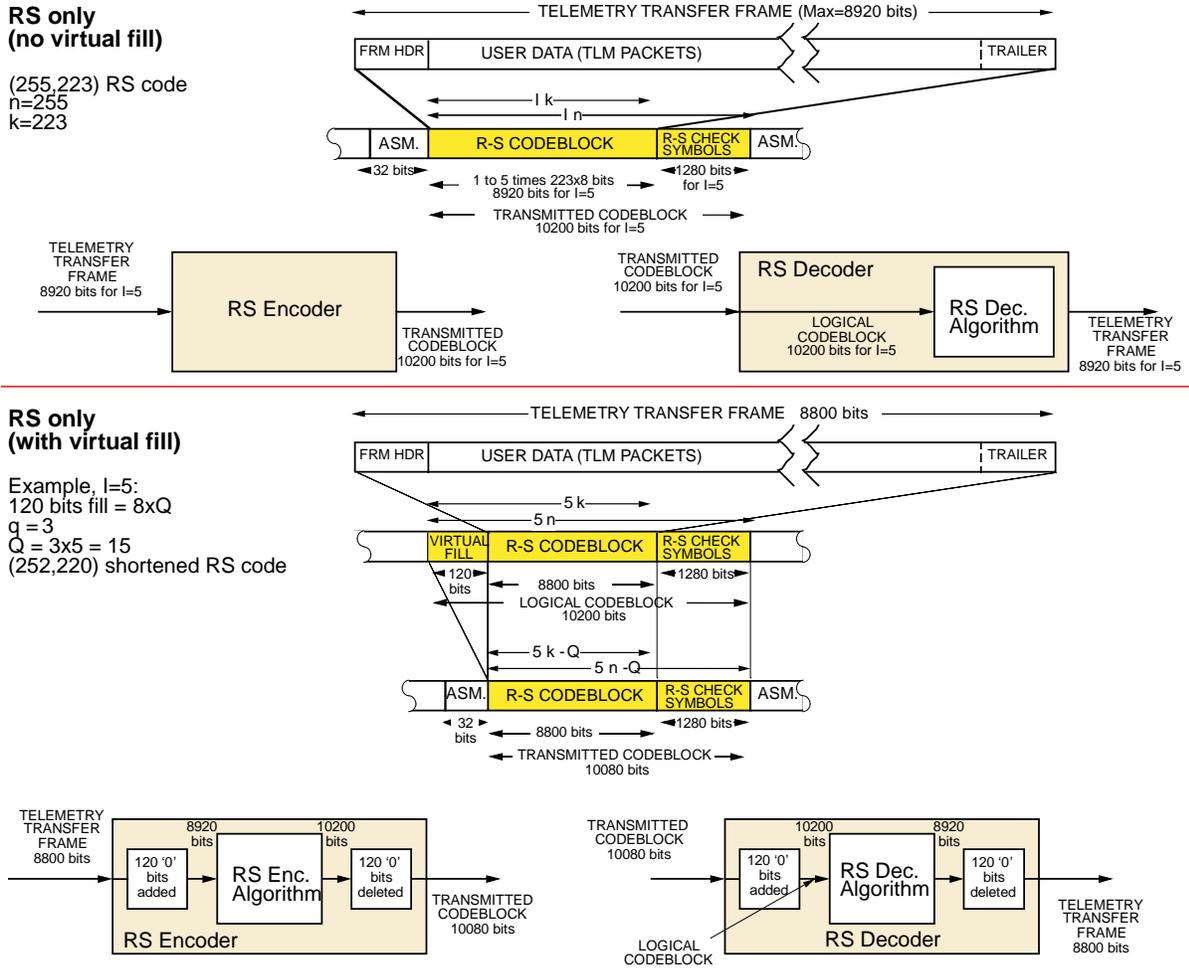
Figure 5-3: Illustration of RS Codeword Structure, with and without Virtual Fill

5.3 INTERLEAVING OF THE REED-SOLOMON SYMBOLS

When concatenated coding is used, or when the RS code is used without concatenation on a bursty channel, interleaving of the RS code symbols improves code performance. Without interleaving, burst error events would tend to occur within one RS codeword, and one codeword would have to correct all of these errors. Thus over a period of time there would be a tendency for some codewords to have 'too many' errors to correct (i.e., greater than E). The purpose of interleaving and de-interleaving is to make the RS symbol errors, at the input of the RS decoder, independent of each other and to distribute the RS symbol errors uniformly; in other words, to distribute the burst errors among several codewords. The performance of the RS decoder is severely degraded by highly correlated errors among several successive symbols.

Rectangular block interleaving of the RS symbols maximally spreads a burst of symbols with errors over a number of codewords equal to the 'interleaving depth' I. The interleaving depth is the number of RS codewords involved in a single interleaving and de-interleaving operation. Interleaving and de-interleaving operations over a channel can be described simply by considering two $I \times n$ matrices, one at the input of the channel and one at the output

(see figure 5-4). For interleaving, put the *I* codewords, each with length *n*, into rows 1,2,...,*I* of the matrix, then transmit the symbols of columns 1,2,...,*n* through the channel. For de-interleaving, do the reverse operation.
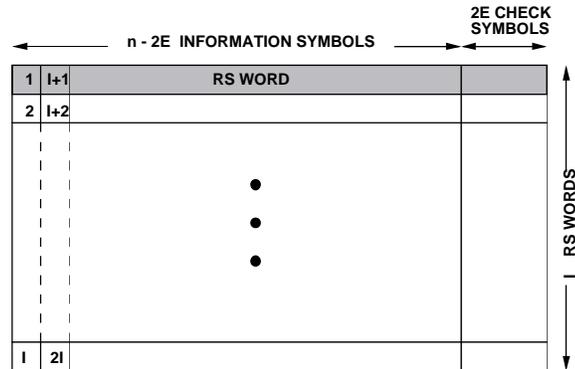
Figure 5-4: Matrix Used for Interleaving

Figure 5-4 illustrates the matrix used for interleaving *I* RS codewords (*interleaving depth I*). Note that this matrix, by itself, does not specify in which order the input information symbols should fill up the matrix cells not reserved for parity. If successive information symbols are written into the matrix in the 'natural' ordering, *row by row*, so as to fill up codewords one at a time, this requires holding *I*–1 full codewords before any of the columns of the matrix can be read out. On the other hand, if successive information symbols are written into the matrix *column by column*, there is no need to store the entire array of code symbols because each column of *I* newly written symbols can be immediately read out as the next *I* symbols of the RS codeblock, as soon as the encoder computes the (linear) contribution of each of these *I* symbols to its corresponding set of RS parity symbols. This is equivalent to the method specified in the Recommended Standard (reference [3]). One potential disadvantage of the recommended method is that it spreads individual RS codeword errors across more source blocks than the 'natural' ordering.

Interleaving of *I* RS codewords multiplies the length of the RS codeblock by *I*. The entire package of *I* RS codewords constitutes one codeblock. However, it is customary to compute WER for individual RS codewords rather than for the whole interleaved codeblock. The error rate on the interleaved codeblock is the FER for CCSDS frames.

5.4   HARD ALGEBRAIC DECODING OF REED-SOLOMON CODES

Unlike the 'soft' channel symbol values that are input to a Viterbi decoder for convolutional codes, the symbols input to the Reed-Solomon decoder are 'hard', which means that the RS decoder operates on symbols drawn from exactly the same alphabet as that used in producing the encoded symbols. This generation of hard symbol inputs to the RS decoder happens automatically when these symbols are generated by a Viterbi decoder for an inner convolutional code. In this case, the Viterbi decoder generates hard bit-by-bit decisions, and eight consecutive bits from the Viterbi decoder are grouped to form one symbol from the

256-ary RS alphabet. When the RS code is used without an inner convolutional code, hard decisions should be made on each group of channel symbols corresponding to one RS octet.

It is possible to allow the decisions on channel symbols to possess a little bit of 'softness', in that a Reed-Solomon decoder may also accept 'erasures' in addition to hard symbols from its native alphabet. An erasure is appropriate whenever there is substantial decision uncertainty between two or more hard symbols, because the Reed-Solomon code is capable of correcting twice as many erasures as errors. In the case of Reed-Solomon/convolutional concatenated coding, erasures are never produced by the standard Viterbi algorithm, but they may be generated by some modified versions of it .

The 'errors-only' Reed-Solomon decoder is somewhat simpler than the 'errors-and-erasures' version, but it is convenient to describe the more general case. The basic idea behind all RS decoding algorithms was developed by Berlekamp as described in reference [12], but there are dozens of variants of his basic algorithm in current use. A very detailed discussion on Reed-Solomon decoding algorithms can be found in reference [24].

Unlike the Viterbi decoder for convolutional codes, which always obtains a maximum likelihood decision for each bit, the Reed-Solomon decoder is an 'incomplete, bounded distance' decoder. The 'errors-only' decoder produces unflagged decoded output if and only if the sequence of received, corrupted symbols differs from a valid codeword by no more than E symbols. For the 'errors-and-erasures' version, the corresponding condition is that $2t+e \leq 2E$, where e is the number of erased symbols and t is the number of discrepancies between non-erased received symbols and those of a valid codeword. For both types of decoders, there are error sequences that move the sequence of received symbols outside the 'bounded-distance' decoding radius around the true codeword, yet also leave it outside the bounded-distance decoding radius of all other codewords. In this case, the RS decoder is incomplete, because it knows that the received sequence has been corrupted beyond its guaranteed correction capability, and it does not attempt to guess how to fix such corruptions. In fact, this type of 'detectable' corruption is much more likely to occur than an error sequence that moves the received symbol sequence inside the decoding radius of an incorrect codeword. For this reason the Reed-Solomon decoder almost always knows when there are too many errors to correct a word. Whenever this happens, the decoder can flag the 'detected' error and inform the user of this fact.

5.5   PERFORMANCE OF THE RECOMMENDED REED-SOLOMON CODES

In decoding the RS codewords, essentially three events may happen.

    a) The first event (*correct decoding*) happens if there are E or fewer RS symbol errors in a codeword. In this case the decoder successfully corrects the errors and outputs the correct information block.

    b) The second event (*detected error*) happens if the number of RS symbol errors in a codeword is more than E, but the corrupted codeword is not close to any other codeword within the distance of E symbols. In this case the RS decoder *fails to*

decode and may (if desired) output the first *k undecoded* information symbols that in all likelihood contain some symbol errors.

c) The third event (*undetected error*) happens if the number of RS symbol errors in a codeword is more than *E*, and the corrupted codeword is closer to some other codeword within the distance of *E* symbols. In this case the decoder is fooled, decodes incorrectly, and outputs a wrong information block. In other words, it claims the decoded block as a correct one and by doing this it may create up to *E* additional symbol errors (compared to the number of errors in the uncoded information block).

Fortunately for most of the RS codes of interest with large alphabet size, in particular for the (255, 223) RS code, the probability that the third event happens is very small (see reference [23]). This probability has very little effect on the error probability performance of an RS code in the range of interest. In reference [23] it has been shown that the probability of the third event, i.e., an incorrect decoding event, is less than $\frac{1}{E!}$. Therefore, for the practical range of interest in error probability performance, it almost surely can be assumed that only the first and second events happen. This conclusion is much less sure for the recommended (255,239) RS code with $E = 8$.

If it can be assumed that symbol errors occur independently with probability V_s at the RS decoder input, then the probability P_w of undecodable word error at the output of the RS decoder is given by

$$P_w(n, E) = \sum_{j=E+1}^{n} \binom{n}{j} V_s^j (1 - V_s)^{n-j}, \tag{3}$$

where $E = \frac{n-k}{2}$ is the number of correctable errors. This expression for P_w counts codeword errors for every occurrence of either the second or third event above.

The RS decoder output symbol error probability can be approximated by

$$P_s \approx V_s P_w(n-1, E-1) = V_s \sum_{i=E}^{n-1} \binom{n-1}{i} V_s^i (1 - V_s)^{n-i-1}. \tag{4}$$

This approximate expression for P_s assumes that nearly all of the symbol errors come from the second event above, and in this case it counts all of the erroneous symbols in the raw (undecoded) information portion of the RS codeblock.

Finally, the bit error probability at the RS decoder output is given approximately by

$$P_b \approx \frac{V_b}{V_s} P_s$$

where V_b is the bit error probability on the channel. On the AWGN channel, $V_s = 1 - (1 - V_b)^J$, and $V_b = Q\left(\sqrt{2E_s/N_0}\right)$, where $Q(x) = \frac{1}{2}\mathrm{erfc}(x/\sqrt{2})$ is the unit Gaussian complementary cumulative distribution function and E_b/N_0 is the channel symbol signal-to-noise ratio. This expression for P_b relies on the same assumptions as for P_s, and also on the assumption that the density of bit errors inside an erroneous undecodable J-bit RS symbol is the same as the density of bit errors inside any J-bit RS symbol regardless of whether the RS codeword is decodable or not and whether the particular RS symbol is erroneous or not.

The performance of the recommended RS codes with $E = 16$ and $E = 8$ is shown in figures 5-5 and 5-6, respectively, as a function of the channel symbol error probability V_s at the input of the decoder. This figure shows the bit, symbol, and word error probabilities, P_b, P_s, and P_w, respectively, at the output of the decoder, as computed from the formulas above.
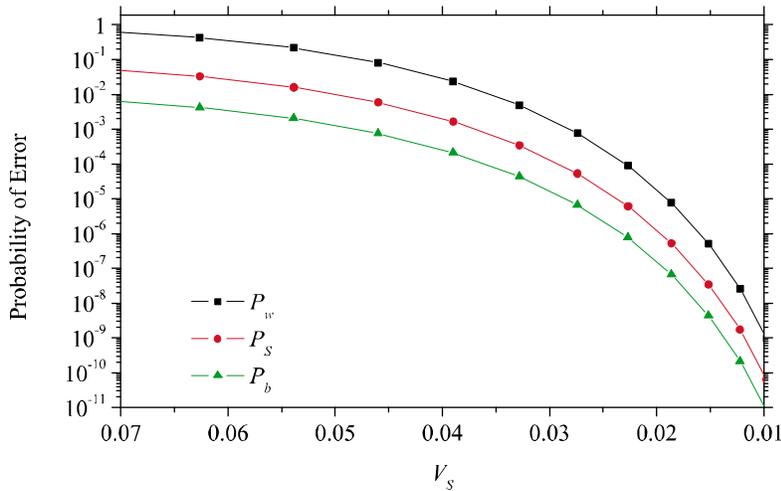
Figure 5-5: *P_w, P_s and P_b for the (255,223) RS Code with E=16*
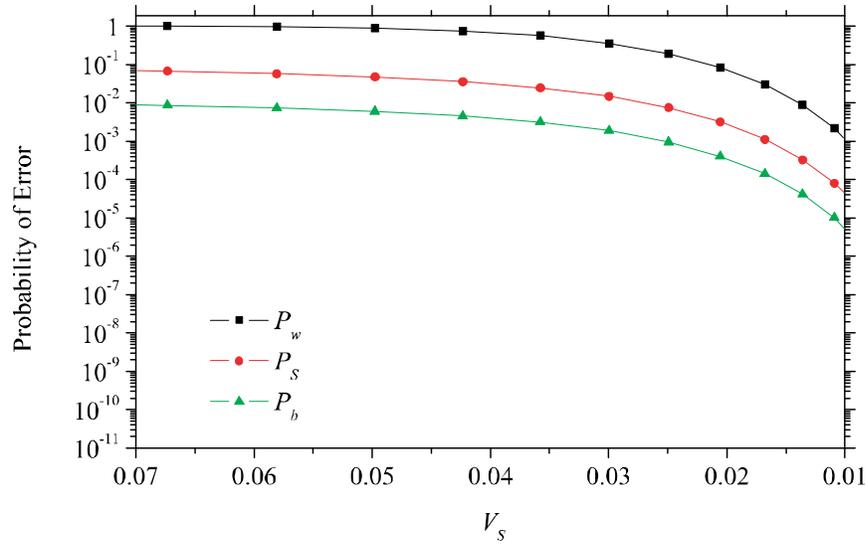
Figure 5-6: P_w, P_s and P_b for the (255,239) RS Code with E=8

Figures 5-7 and 5-8 show BER and WER performance curves for the recommended RS codes as a function of the normalized bit signal-to-noise ratio E_b/N_0 on the AWGN channel. Note that the WER curve for RS codes on the AWGN channel does not depend on the interleaving depth I, but for concatenated systems WER does depend on I. The WER curves in Figures 5-7 and 5-8 are the same as FER curves for interleaving depth I=1.
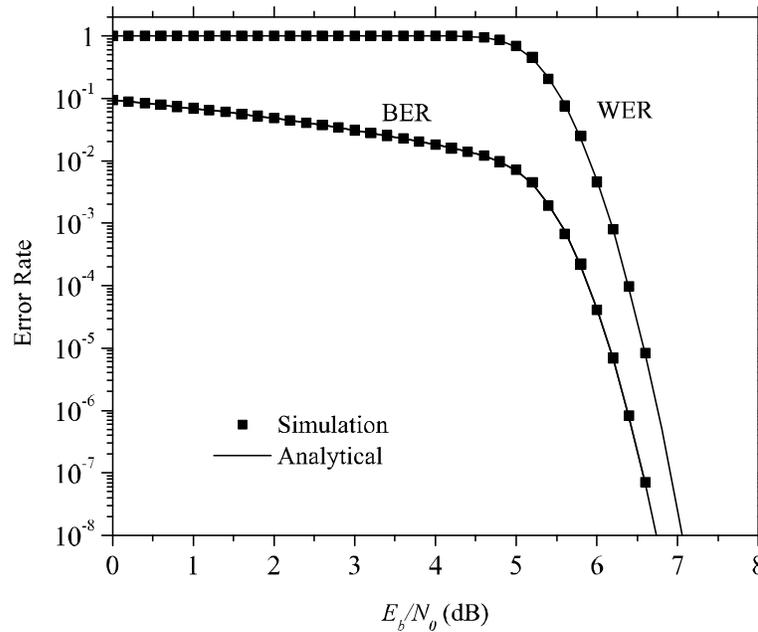
Figure 5-7: BER and WER Performance of the CCSDS E=16 Reed-Solomon Code (255,223): Simulated and Analytical Results for the AWGN Channel
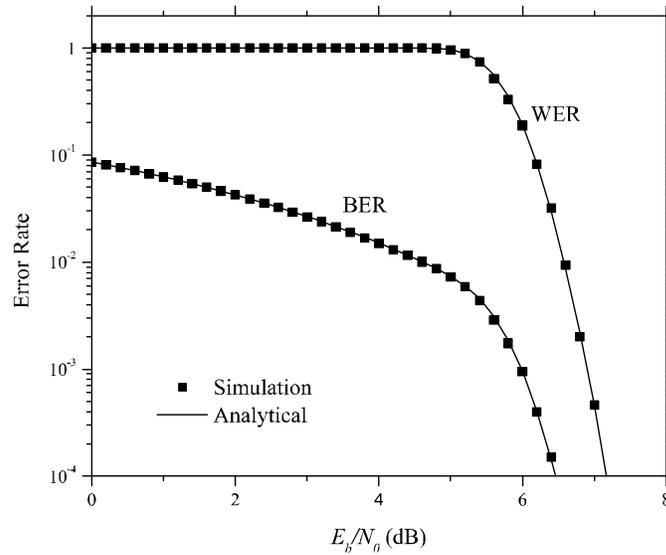
Figure 5-8: BER and WER Performance of the CCSDS E=8 Reed-Solomon Code (255,239): Simulated and Analytical Results for the AWGN Channel

Finally figure 5-9 illustrates the effects of shortening the recommended E=16 and E=8 Reed-Solomon codes. On the AWGN channel shortening may actually improve the performance (This is not the case for the recommended concatenated system). The best performance on the AWGN channel is achieved by a non-standard (255,173) RS code with E=41.
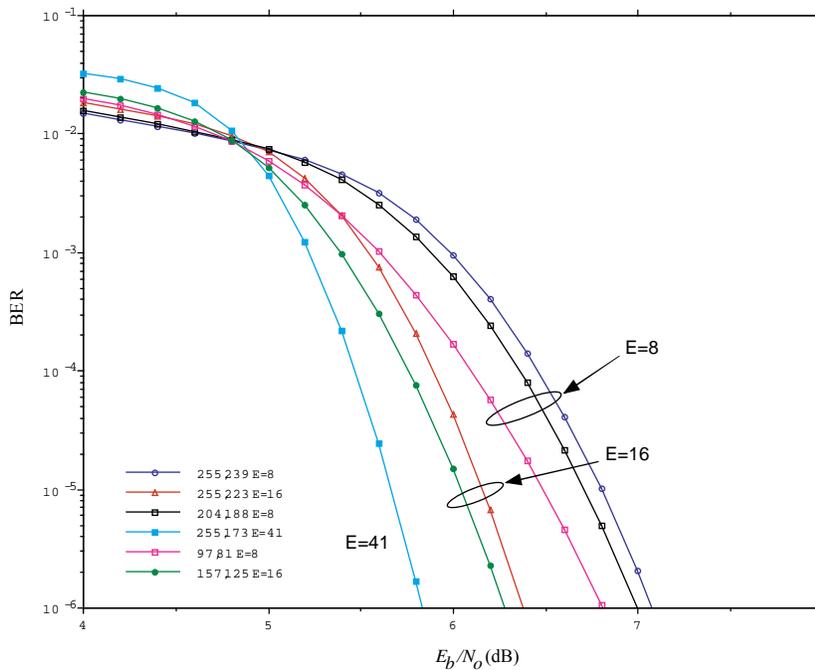
Figure 5-9: BER Performance Comparison of Shortened and Non-Shortened Reed-Solomon Codes on the AWGN Channel

6 CONCATENATED CODES: REED-SOLOMON AND CONVOLUTIONAL

6.1 INTRODUCTION

One method to build a strong code while maintaining manageable decoding complexity is to concatenate two codes, an 'outer code' and an 'inner code'. This section discusses a particular concatenated coding scheme of importance to space communications (low SNR). The recommended concatenated coding system consists of a Reed-Solomon outer code and a convolutional inner code (which is Viterbi decoded). Typically, the inner convolutional code corrects enough errors so that a high-code-rate outer code can reduce the error probability to the desired level. The reader may wish to consult reference [26] for the theory of concatenated coding and references [9] and [27] for more information on the Reed-Solomon/Viterbi concatenated code.

The concatenated code in the Recommended Standard (reference [3]) uses either of the recommended RS codes (or shortened versions) together with any of the recommended convolutional codes (either of which may also be used separately under the Recommended Standard (reference [3])). A block diagram of this concatenated coding system is given in figure 6-1. The binary input data sequence is divided into 8-bit sequences to form symbols over a $2^8 = 256$-ary alphabet. The Reed-Solomon (RS) code then encodes the symbols such that any combination of E or fewer symbol errors per RS word (255 symbols per word) can be corrected.

The reason that the recommended concatenated code operates as an effective teaming of its outer and inner codes stems from the nature of Viterbi decoding. The decoded bit errors made by the constraint-length-7 convolutional decoder tend to clump together in reasonably short bursts. In a concatenated coding system that uses a convolutional inner code, the outer code should be tailored to the burst error environment created by the convolutional decoder. A $(255,255–2E)$ Reed-Solomon outer code is a good match for the convolutional inner code with constraint length 7 because the bursts of errors from the convolutional decoder typically have burst lengths ranging from a few bits to several constraint lengths. This corresponds to only a small number of 8-bit symbols in the outer code, and hence only a moderate amount of interleaving is required to prevent a few long bursts from exceeding the error correction capability of the Reed-Solomon decoder. On the other hand, it is advantageous for Viterbi decoder errors to be clustered within individual RS symbols, because an RS symbol is equally wrong to the RS decoder whether it contains one bit error or eight bit errors. Because the Viterbi decoder errors occur in bursts comparable in length to the RS symbol size, 3 or 4 Viterbi decoder bit errors will typically be packed into a single RS symbol, and these cause much less damage than isolated bit errors to the error correction abilities of the outer code, at a given bit error rate of the inner code. In summary, the typical error bursts from a constraint-length-7 convolutional decoder are long enough to take advantage of packing Viterbi-decoded bit errors into single 8-bit RS symbols, but not so long as to require an inordinate amount of interleaving to keep the Reed-Solomon code from being overwhelmed by overly lengthy error bursts.
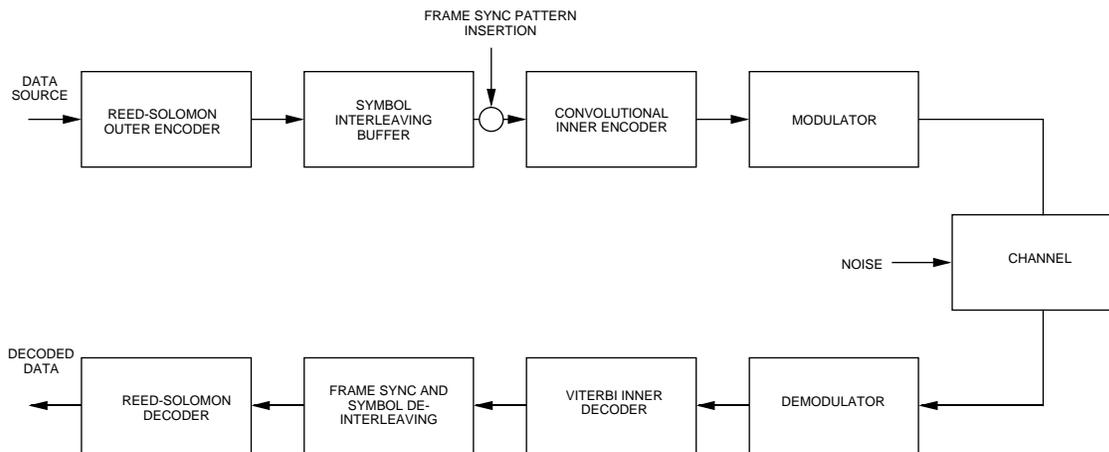
Figure 6-1: Concatenated Coding System Block Diagram

6.2   ENCODING AND DECODING A CONCATENATED CODE

Encoding or decoding of a concatenated code is a simple matter of encoding or decoding the two codes in sequence.

Interleaving between the Outer and Inner Codes — When concatenated coding is used, interleaving is recommended because the inner Viterbi decoder errors tend to occur in bursts, which occasionally are as long as several constraint lengths (see figure 6-2). Without interleaving, Viterbi decoder burst error events would tend to occur within one RS codeword, so that one codeword would have to correct all of these errors. Thus there would be a tendency for some codewords to have 'too many' errors to correct (i.e., greater than E).
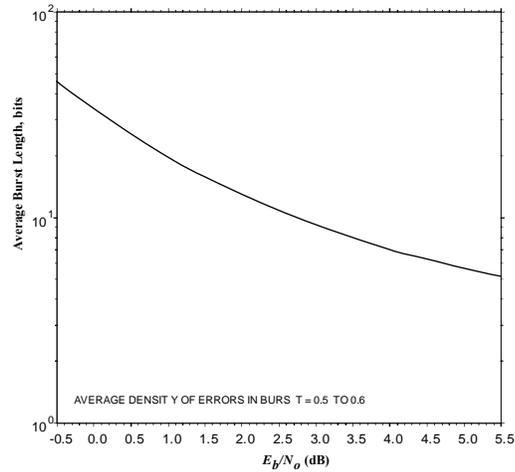
Figure 6-2: Average Burst Length vs. SNR, at the Viterbi Decoder Output, K=7 CCSDS Convolutional Code

Table 6-1 shows the frame lengths for all the recommended interleaving depths for the two (non-shortened) RS codes.

Table 6-1: Frame Lengths for All Interleaving Depths

Interleaver depth I	Frame length L, bits	
	E=16	E=8
1	1784	1912
2	3568	3824
3	5352	5736
4	7136	7648
5	8920	9560
8	14272	15296

6.3 PERFORMANCE OF THE RECOMMENDED CONCATENATED CODING SYSTEMS

Consider a concatenated coding system consisting of a (K,r) convolutional inner code of rate r and constraint length K, and an (n,k) Reed-Solomon outer code. It is assumed that the symbols are interleaved at a sufficient depth to insure that symbol errors are independent at the RS decoder input. Then the bit, symbol, and word error probabilities, P_b, P_s, and P_w, respectively, are given by the formulas in the previous section, in terms of V_s, the symbol error probability at the input of the RS decoder or equivalently at the output of the Viterbi decoder, and V_b, the bit error probability at the output of the Viterbi decoder. The ratio V_b/V_s is estimated empirically and depends on the burst statistics of the inner decoder's error events at its typical operating SNR.

Figures 6-3 and 6-4 show the BER performance of the non-shortened (255,223) and (255,239) RS codes with $E=16$ and $E=8$, respectively, concatenated with punctured and non-punctured convolutional codes, with infinite interleaving assuming that interleaving produces independent RS symbol errors. Performance curves for the non-concatenated convolutional codes and for the RS code alone are also shown for comparison. Note that for bandwidth efficiency it is better to use concatenations of RS and punctured convolutional codes than the Reed-Solomon code alone.
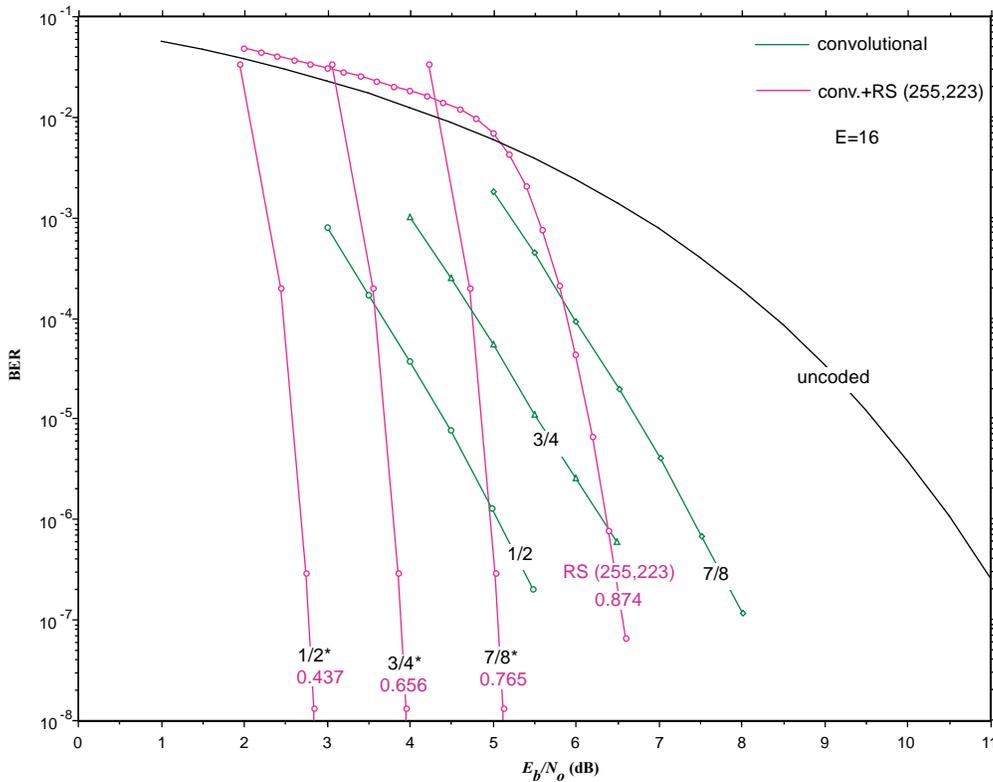
Figure 6-3: Performance of Concatenated Coding Systems with Infinite Interleaving, $E=16$, Punctured Codes
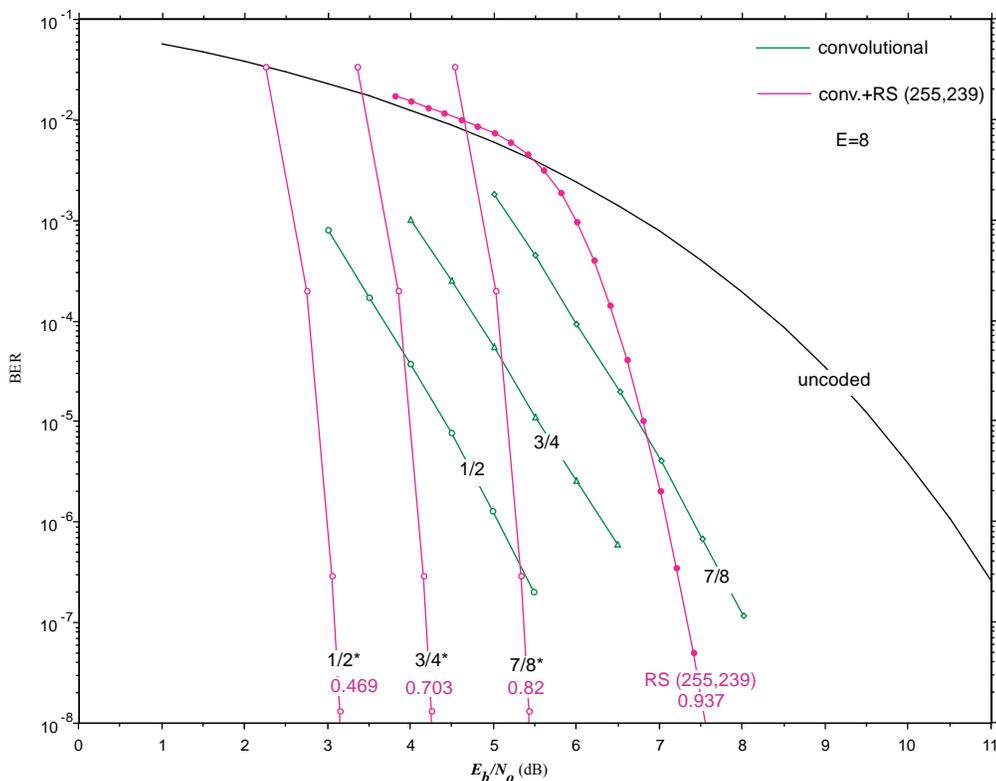
Figure 6-4: Performance of Concatenated Coding Systems with Infinite Interleaving, *E*=8, Punctured Codes

The convolutional decoder used to calculate the performance curves for all of the figures in this section operated with an unquantized maximum likelihood soft decision algorithm, corresponding to the 'unquantized soft decision' curve in figure 6-5. Note that, in order to compare the performance of concatenated and non-concatenated codes, the E_b/N_0 values on the *x*-axis in all figures in this section refer to the information bit SNR.

Effects of Finite Interleaving — When the interleaving depth *I* is not large enough, the errors at the output of the Viterbi decoder cannot be considered as independent since this decoder tends to produce errors in bursts. The performance under finite interleaving must therefore take into account the statistics of these bursts either by devising a plausible model or by simulation. A possible model for burst lengths and arrival times was developed in reference [27] and is called the *geometric* model. This model provides an approximate estimate of the performance under finite interleaving, but ignores the actual structure of the error patterns within the bursts. On the other hand, simulation is also problematic since very large amounts of Viterbi decoded data is necessary to provide reasonable confidence in the estimates of performance. A detailed description of methods to obtain performance estimates is given in reference [28].

BER and WER results for finite interleaving are shown in figures 6-5 and 6-6 respectively for the recommended concatenated system consisting of the non-shortened (255,223) RS

code with $E=16$ and the non-punctured $(7, 1/2)$ convolutional code, with different interleaving depths ranging from $I=1$ to $I=16$.
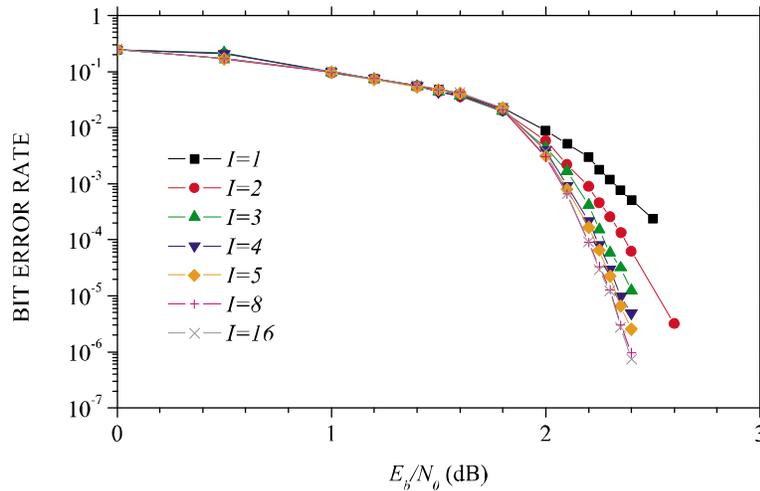
Figure 6-5: Bit Error Rate Simulated Performance of the CCSDS Concatenated Scheme with Outer $E=16$ Reed-Solomon Code (255,223) and Inner Rate-1/2 Convolutional Code as a Function of Interleaving Depth
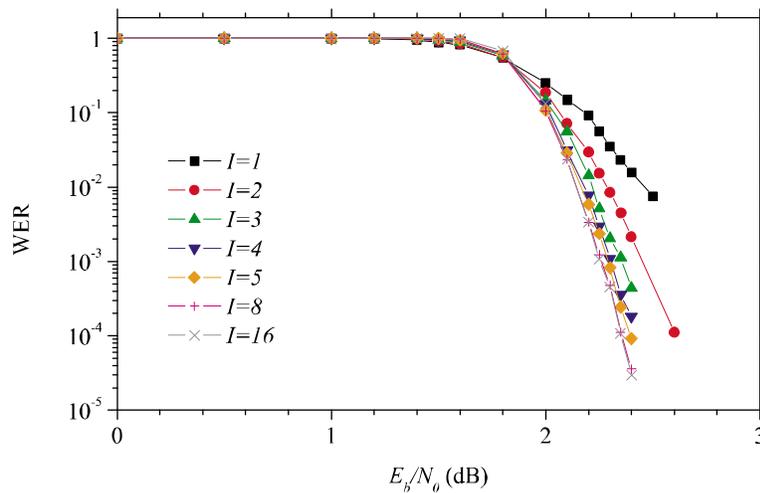
Figure 6-6: Word Error Rate Simulated Performance of the CCSDS Concatenated Scheme with Outer $E=16$ Reed-Solomon Code (255,223) and Inner Rate-1/2 Convolutional Code as a Function of Interleaving Depth

Figures 6-5 and 6-6 illustrate how interleaving depth $I=5$ obtains near-ideal performance. This amount of interleaving is also sufficient to obtain near-ideal performance for most other combinations of recommended RS and convolutional codes.

Figures 6-7 and 6-8 show BER and WER for the recommended concatenated system consisting of the non-shortened (255,239) RS code with $E = 8$ and the non-punctured (7, 1/2) convolutional code, with different interleaving depths ranging from $I = 1$ to $I = 16$.
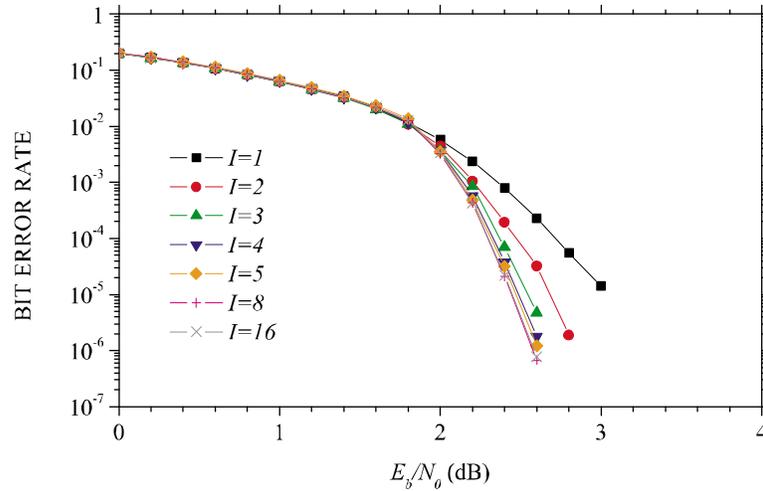
Figure 6-7: **Bit Error Rate Simulated Performance of the CCSDS Concatenated Scheme with Outer E=8 Reed-Solomon Code (255,239) and Inner Rate-1/2 Convolutional Code as a Function of Interleaving Depth**
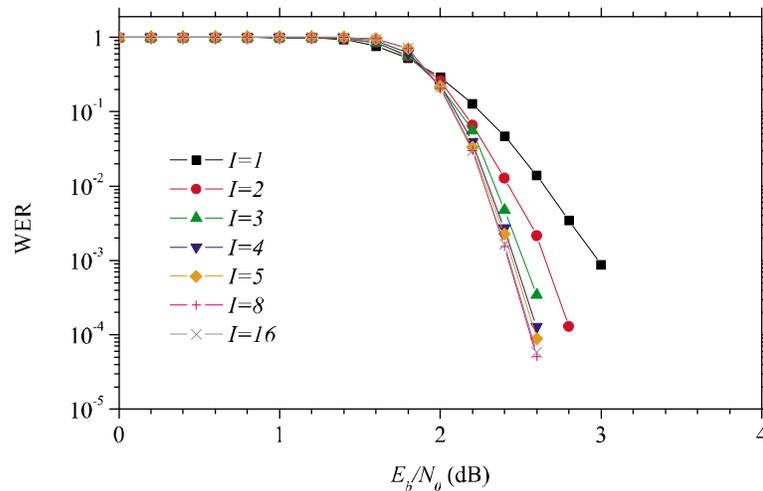
Figure 6-8: **Word Error Rate Simulated Performance of the CCSDS Concatenated Scheme with Outer E=8 Reed-Solomon Code (255,239) and Inner Rate-1/2 Convolutional Code as a Function of Interleaving Depth**

Figures 6-9 and 6-10 show BER and WER curves for the concatenated codes consisting of the non-shortened (255,223) RS code with $E = 16$ concatenated with any of the recommended punctured or non-punctured (7, 1/2) convolutional codes, with interleaving depth $I = 5$ (which gives a close approximation to ideal performance on the AWGN channel).
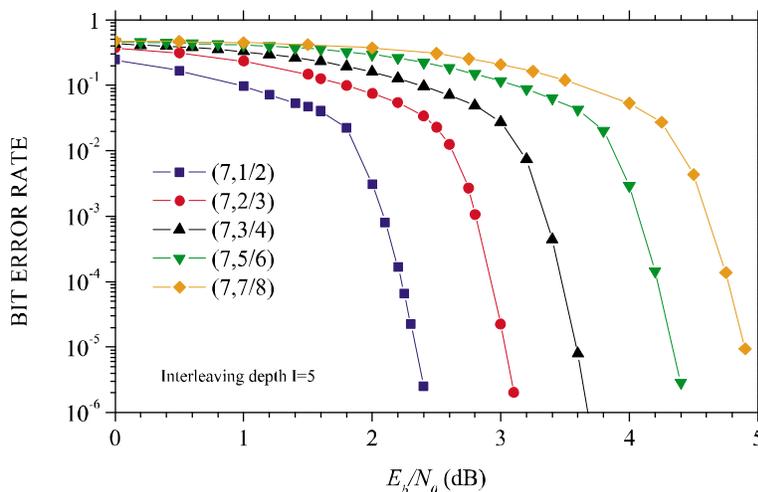
Figure 6-9: **Bit Error Rate Simulated Performance of the CCSDS Concatenated Scheme with Outer E=16 Reed-Solomon Code (255,223) and Inner Punctured Convolutional Codes, Using Finite Interleaving with I=5**
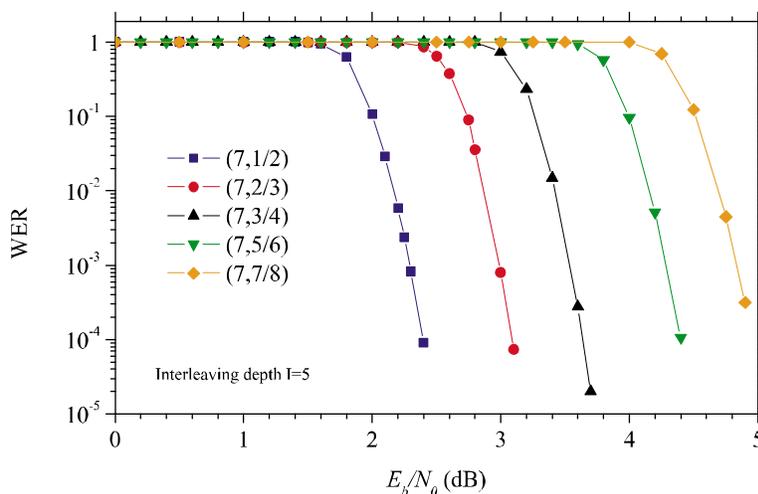
Figure 6-10: **Word Error Rate Simulated Performance of the CCSDS Concatenated Scheme with Outer E=16 Reed-Solomon Code (255,223) and Inner Punctured Convolutional Codes, Using Finite Interleaving with I=5**

Figures 6-11 and 6-12 show BER and WER curves for the concatenated codes consisting of the non-shortened (255,223) RS code with $E = 16$ concatenated with any of the recommended punctured or non-punctured (7, 1/2) convolutional codes, with interleaving depth $I = 5$ (which gives a close approximation to ideal performance on the AWGN channel).
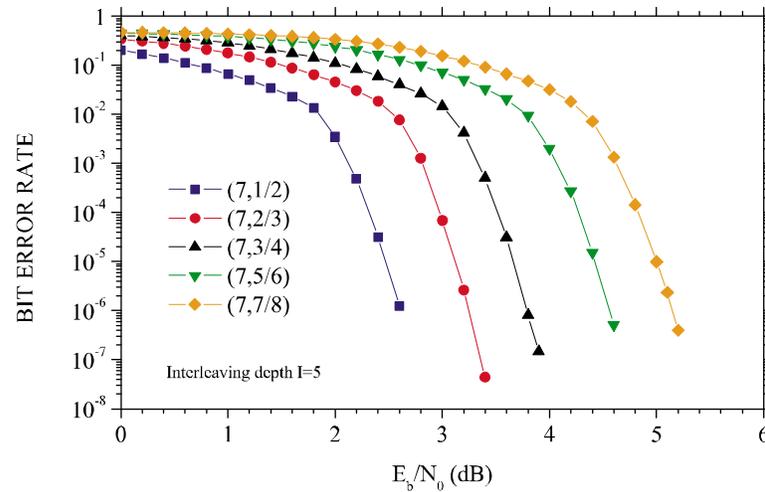
Figure 6-11: Bit Error Rate Simulated Performance of the CCSDS Concatenated Scheme with Outer *E*=8 Reed-Solomon Code (255,239) and Inner Punctured Convolutional Codes, Using Finite Interleaving with *I*=5
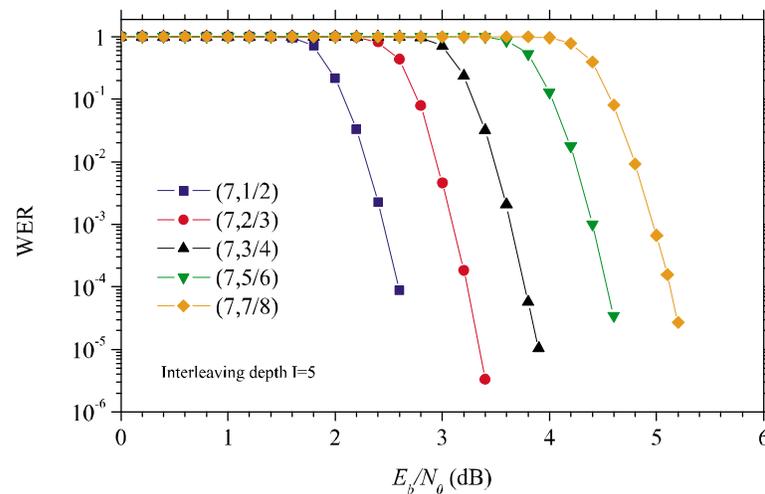

Figure 6-12: Word Error Rate Simulated Performance of the CCSDS Concatenated Scheme with Outer *E*=8 Reed-Solomon Code (255,239) and Inner Punctured Convolutional Codes, Using Finite Interleaving with *I*=5

7   TURBO CODES

7.1   INTRODUCTION

In 1993 a new class of concatenated codes called 'turbo codes' was introduced. These codes can achieve near-Shannon-limit error correction performance with reasonable decoding complexity. Turbo codes outperform even the most powerful codes known to date, but more importantly they are much simpler to decode. It was found that good turbo codes can come within approximately 0.8 dB of the theoretical limit at a bit error rate (BER) of 10^{-6}. In applying this rule of thumb, it is important to keep in mind that the limiting performance depends on the code rate.

A turbo code is a combination of two simple recursive convolutional codes, each using a small number of states. These simple convolutional codes are in fact 'terminated' convolutional codes and hence block codes. For a block of k information bits, each constituent code generates a set of parity bits. The turbo code consists of the information bits and both sets of parity, as shown in figure 7-1.
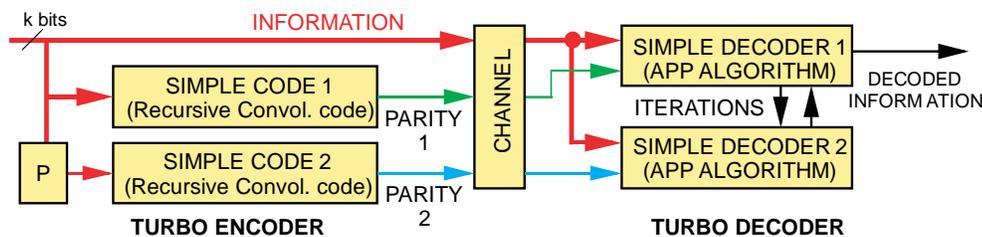
Figure 7-1:   Example of Turbo Encoder/Decoder

The key innovation is an interleaver P, which permutes the original k information bits before encoding the second code. If the interleaver is well-chosen, information blocks that correspond to error-prone codewords in one code will correspond to error-resistant codewords in the other code. The resulting code achieves performance similar to that of Shannon's well-known 'random' codes, but random codes approach optimum performance only at the price of a prohibitively complex decoder.

Turbo decoding uses two simple decoders individually matched to the simple constituent codes. Each decoder sends likelihood estimates of the decoded bits to the other decoder, and uses the corresponding estimates from the other decoder as *a priori* likelihoods. The constituent decoders use the 'APP' (*a posteriori* probability) bitwise decoding algorithm, which requires the same number of states as the well-known Viterbi algorithm. The turbo decoder iterates between the outputs of the two decoders until reaching satisfactory convergence. The final output is a hard-quantized version of the likelihood estimates of either of the decoders.

To achieve maximum performance, turbo codes use large block lengths and correspondingly large interleavers. The size of the interleaver affects buffer requirements and decoding delay,

but has little impact on decoding speed or decoder complexity. More recently, it was discovered that turbo codes with shorter blocks also perform amazingly well with respect to the theoretical performance bounds on codes constrained to have a given block length. Thus, turbo codes can also offer good performance for applications requiring small block sizes on the order of a few hundreds of bits (but these block sizes are not within the scope of the Recommended Standard (reference [3])).

7.2 TURBO ENCODER

A turbo encoder is a combination of two simple encoders. The input is a frame of k information bits. The two component encoders generate parity symbols from two simple recursive convolutional codes, each with a small number of states. The information bits are also sent uncoded. An interleaver permutes bit-wise the original k information bits before input to the second encoder. A generic implementation block diagram for a turbo encoder is shown in figure 7-2. The specific turbo encoder in the CCSDS Recommended Standard (reference [3]) is shown in more detail in figure 7-3.
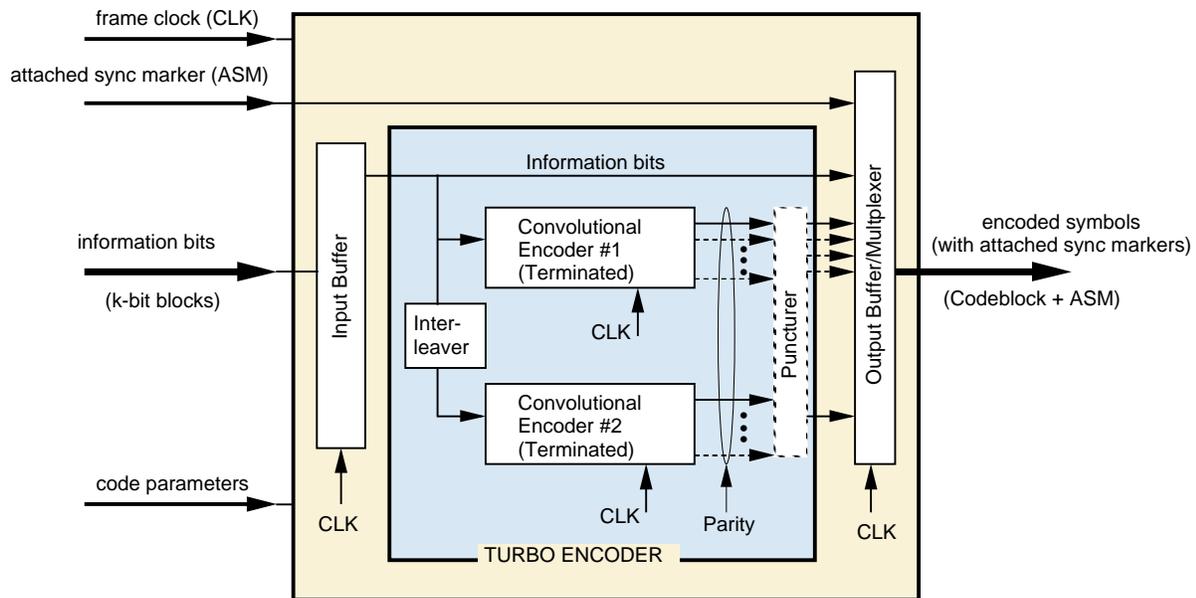
Figure 7-2: Block Diagram of Turbo Encoder
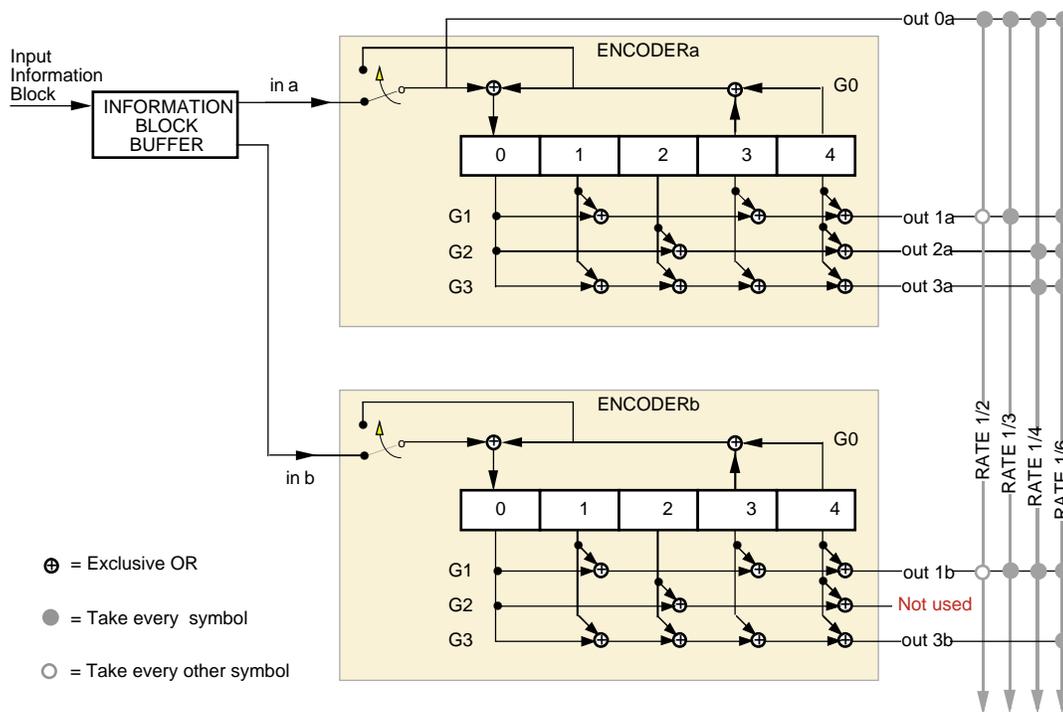
Figure 7-3: Turbo Encoder Block Diagram

The two convolutional encoders in the Recommended Standard (reference [3]) are recursive with constraint length $K = 5$, and are realized by feedback shift registers. However, unlike the encoder for the recommended plain convolutional code in section 4, the turbo codeblock is terminated by running each encoder for an additional K-1 bit times beyond the end of the information bit frame. After encoding the last bit in the frame, the leftmost adder in each component encoder receives two copies of the same feedback bit, causing it to zero its output. After K-1 more bit times, all 4 memory cells become filled with zeros, but in the interim the encoder continues to output nonzero encoded symbols.

The Recommended Standard (reference [3]) allows options for non-punctured codes with rates between 1/3, 1/4, and 1/6. The puncturer is used only for code rate 1/2.

The interleaver in the Recommended Standard (reference [3]) is based on a permutation rule which can be computed on-the-fly or pre-computed and stored in a look-up table, for all allowable frame lengths (1784 to 16384 bits).

In figure 7-2, CLK indicates the frame clock. It is used: (1) by the input buffer to determine when to empty and refill the buffer; (2) by the output buffer/multiplexer to determine when to insert the frame sync marker; (3) by each of the convolutional encoders to determine when to terminate the codeblock. Note that an entire information block of k bits must be read in before the encoding can proceed, because some of the bits in the tail end of block will be permuted to the front and need to be encoded first. Thus, there is a fundamental encoding latency of at least k bits in the encoding process.

The turbo code introduces a couple of unique encoder complexity issues. The information block needs to be buffered and read out in a permuted order as part of the encoding process. This buffering has no analog in the plain convolutional encoder, but the size of this buffer is comparable to that required for an interleaved Reed-Solomon codeblock of the same size. The difference is that the traditional concatenated coding architecture completely separates the Reed-Solomon encoder (with its associated buffer) from the convolutional encoder. Thus, the turbo encoder cannot be regarded as a plug-in replacement for the convolutional encoder hardware. The turbo encoder actually replaces the Reed-Solomon/convolutional encoder combination.

Another complexity consideration is how to implement the permutation. The best permutations for turbo codes look very random, but this requires specifying a random-looking readout order via a ROM (Table look-up). An alternative is to use a permutation that can be generated by a simple rule rather than from a lookup table, with minor performance sacrifice. The Recommended Standard (reference [3]) specifies a permutation based on a simple rule, because it was preferred in terms of implementation on the spacecraft.

7.3 TURBO DECODER

A turbo decoder uses an iterative decoding algorithm based on simple decoders individually matched to the two simple constituent codes. Each constituent decoder makes likelihood estimates derived initially without using any received parity symbols not encoded by its corresponding constituent encoder. The (noisy) received uncoded information symbols are available to both decoders for making these estimates. Each decoder sends its likelihood estimates to the other decoder, and uses the corresponding estimates from the other decoder to determine new likelihoods by extracting the 'extrinsic information' contained in the other decoder's estimates based on the parity symbols available only to it. Both decoders use the '*a posteriori* probability' (APP) bitwise decoding algorithm, which requires the same number of states as the well-known Viterbi algorithm. The turbo decoder iterates between the outputs of the two constituent decoders until reaching satisfactory convergence. The final output is a hard-quantized version of the likelihood estimates of either of the decoders.

The Recommended Standard (reference [3]) does not include a detailed description of the specific turbo decoding algorithm. However, the performance curves in 7.4 for the turbo code family in the Recommended Standard (reference [3]) were obtained using a decoding algorithm with the following characteristics:

a) decoder type: Iterative 'turbo' decoding using two 16-state component decoders (see reference [18]);

b) type of component decoders: Soft-input, soft-output APP decoders (see reference [19]);

c) quantization of channel symbols: At least 6 bits/symbol;

d) quantization of decoder metrics: At least 8 bits;

e) number of decoder iterations: variable depending on signal-to-noise ratio.

Variations from this algorithm will result in performance tradeoffs.

The overall turbo decoding procedure is depicted in figure 7-1 and described earlier. The 'simple decoders 1 and 2' each compute likelihood estimates (APP estimates) based on a version of the APP or log-APP algorithm,[5] as described in reference [14]. A diagram showing the structure of the turbo decoder in more detail is shown in figure 7-4. Figure 7-5 shows the basic circuits needed to implement the log-APP algorithm.
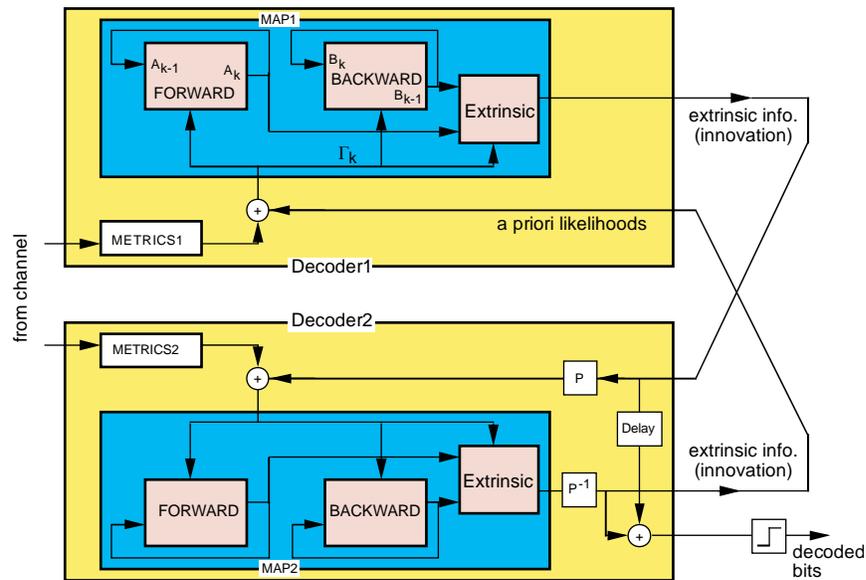
Figure 7-4: Structure of the Turbo Decoder

---

[5]In the early turbo coding literature the APP algorithm was designated as the MAP (maximum a posteriori) algorithm because it was derived from an homonymous algorithm for making optimum bit-wise hard decisions on plain convolutionally encoded symbols.

Basic Structure for Forward and Backward
Computation in the Log-APP Algorithm

Basic Structure for Bit Reliability Computation
in the Log-APP Algorithm.
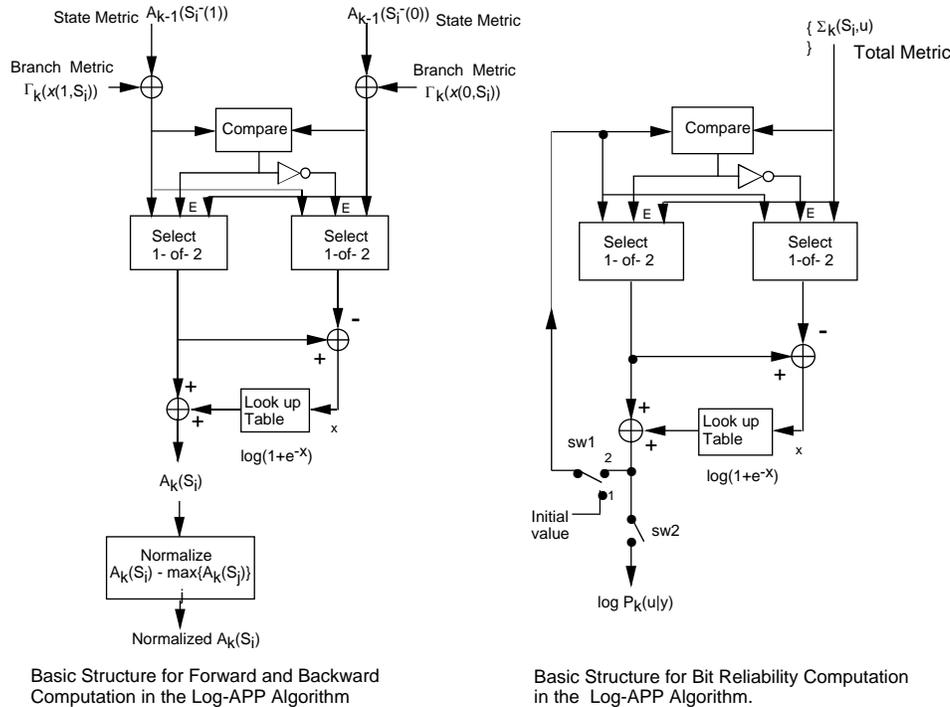
Figure 7-5: Basic Circuits to Implement the Log-APP Algorithm

Because the decoder processes whole blocks of k bits at a time, there is a minimum decoding delay of k bits. This latency is further increased by the time required for the decoder to process each block. If parallel decoders are used to increase decoding throughput, the latency increases in proportion to the number of parallel decoders.

To first order, the decoding complexity of a turbo decoder relative to that of a convolutional decoder using the *same* number of trellis states and branches can be estimated by multiplying several factors: (a) a factor of 2 because the turbo code uses two component decoders; (b) another factor of 2 because the individual decoders use forward and backward recursions compared to the Viterbi decoder's forward-only recursion; (c) another small factor because the turbo decoder's recursions require somewhat more complex calculations than the Viterbi decoder's; and (d) a factor to account for the turbo decoder's multiple iterations compared to the Viterbi decoder's single iteration. The relative decoding complexity for two *different* turbo codes or two *different* convolutional codes can be estimated by multiplying two additional factors: (e) the number of trellis states; and (f) the number of trellis branches per input bit into each state. Factor (c) can be reduced to one by implementing an approximate log-MAP algorithm at a small sacrifice in performance. Factors (b) and (d) might be reduced on the average by using a more advanced turbo decoding algorithm, using stooping rules or different iteration schedules. Such an algorithm might allow the decoder to stop its iterations early if a given codeword can already be decoded reliably, or to skip over portions of the forward and backward recursions for some iterations. Factors (a) through (d) are 1 for Viterbi decoders of convolutional codes. For the CCSDS standard constraint-length-7 convolutional decoder, factor (e) is $2^6 = 64$, and factor (f) is $2/1 = 2$. For the Cassini/Pathfinder constraint-

length 15, rate 1/6 convolutional decoder, factor (e) is $2^{14} = 16384$ and factor (f) is 6/1=6. For the turbo codes specified in 7.2, factor (e) is 2^4=16 and factor (f) ranges from 2/1=2 to 6/1=6.

A basic form of turbo decoder stops iterating after a predetermined number of iterations. For some codewords (or sections of codewords), the predetermined number of iterations may be too many or too few. A more efficient turbo decoder can employ a stopping rule to stop the decoder's iterations when convergence is satisfactory, i.e., without wasting iterations when the decoder has already converged, and without halting iterations prematurely when the decoder needs a little more time. Such a rule reduces the *average* number of iterations and increase the average decoding throughput. This comes at the expense of a slightly more complicated decoding algorithm and increased decoder buffering requirements to accommodate variable decoding times.

7.4 PERFORMANCE OF THE RECOMMENDED TURBO CODES

7.4.1 SIMULATED TURBO CODE PERFORMANCE CURVES

Figures 7-6, 7-7, 7-8, and 7-9, show the simulated performance of the recommended turbo codes of rates 1/2, 1/3, 1/4, and 1/6, constructed for information block lengths of 1784, 3568, 7136, and 8920 bits. For all of the results in these figures, the decoder used a fixed-iteration stopping rule and stopped after 10 iterations.

To achieve a bit error rate (BER) of 10^{-6}, threshold bit-SNRs of approximately -0.1 dB, 0.15 dB, 0.4 dB , and 1.1 dB, are required by the turbo codes of rates 1/6, 1/4, 1/3, and 1/2, respectively. Approximately the same threshold bit-SNRs achieve a word error rate (WER) or frame error rate (FER) of 10^{-4} for these codes. (Note that WER = FER for the CCSDS turbo codes because the turbo code's information block corresponds to one CCSDS frame).
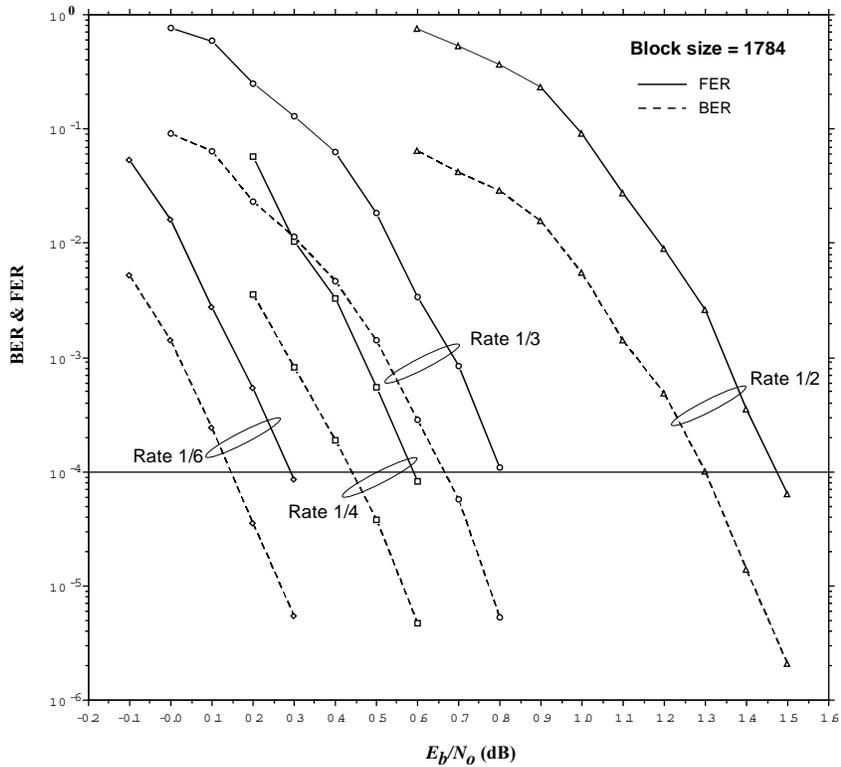
Figure 7-6: BER and FER Performance for Rate 1/2, 1/4, 1/3 and 1/6 Turbo Codes with Block Size 1784 Bits, Measured from JPL DSN Turbo Decoder, 10 Iterations
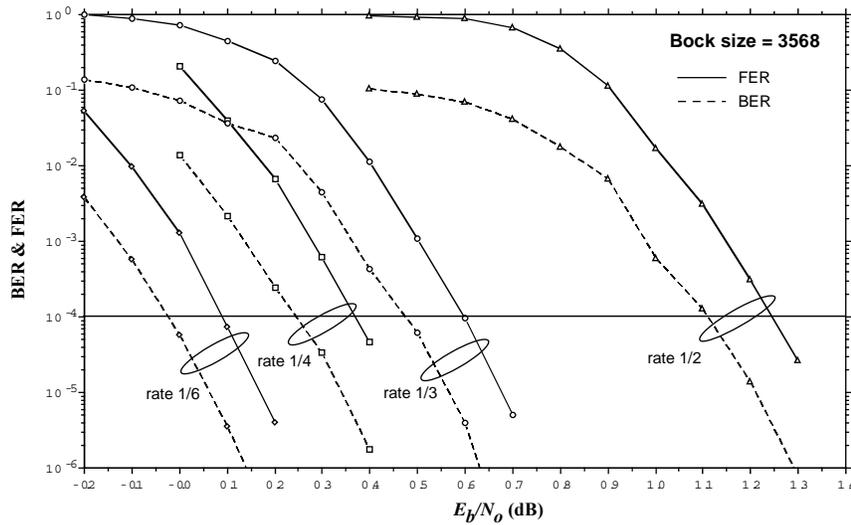
Figure 7-7: BER & FER Performance for Rate 1/2, 1/4, 1/3 and 1/6 Turbo Codes with Block Size 3568 Bits, Software Simulation, 10 Iterations[6]

---

[6] Performance of hardware decoder not available.
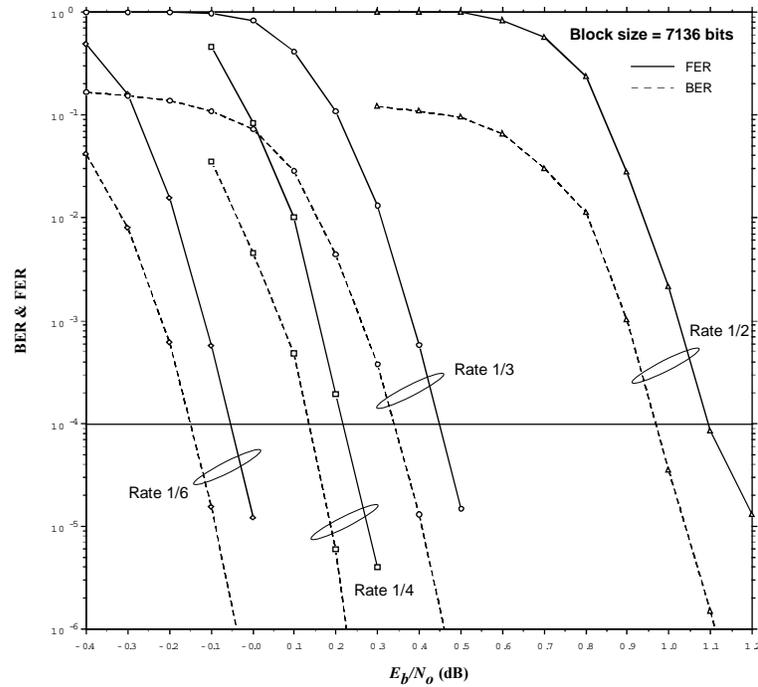
Figure 7-8: BER & FER Performance for Rate 1/2, 1/4, 1/3 and 1/6 Turbo Codes with Block Size 7136 bits, Software Simulation, 10 Iterations[6]
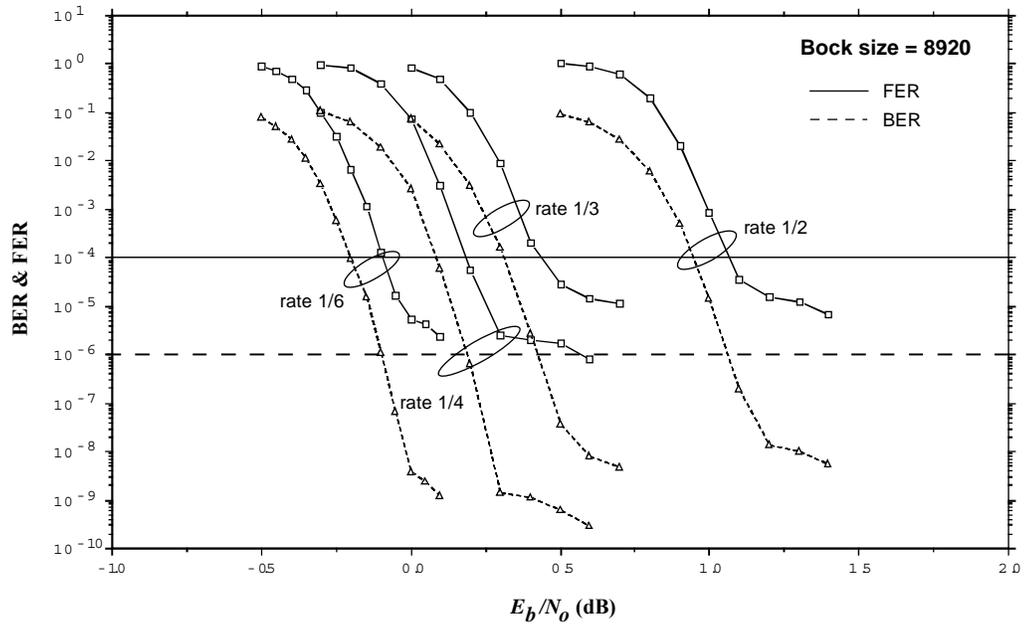
Figure 7-9: BER & FER Performance for Rate 1/2, 1/4, 1/3 and 1/6 Turbo Codes with Block Size 8920 Bits, Measured from JPL DSN Turbo Decoder, 10 Iterations

Figure 7-10 shows the simulated performance of turbo codes of rates 1/2, 1/3, 1/4, and 1/6 with an information block length of 16384 bits. These performance curves do not necessarily reflect the performance of the CCSDS codes for this block length since the recommended interleaver for this block length has not been specified yet.
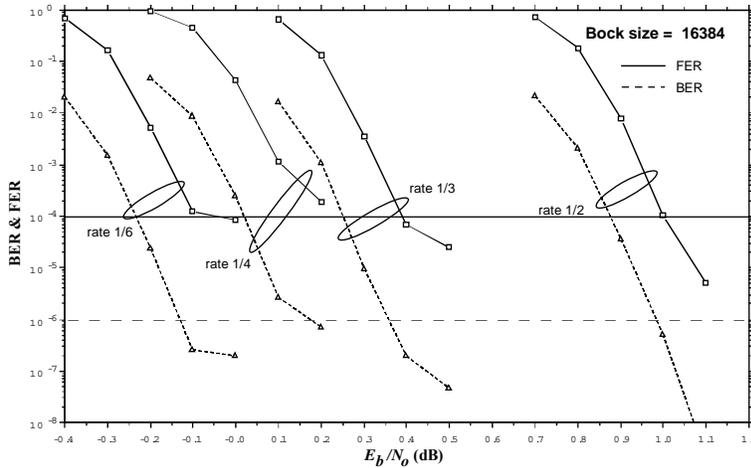

Figure 7-10: BER & FER Performance for Rate 1/2, 1/4, 1/3 and 1/6 Turbo Codes, Block Size 16384 Bits, Software Simulation, 10 Iterations

Figure 7-11 illustrates how the decoder's average speed can be increased through the use of stopping rules.
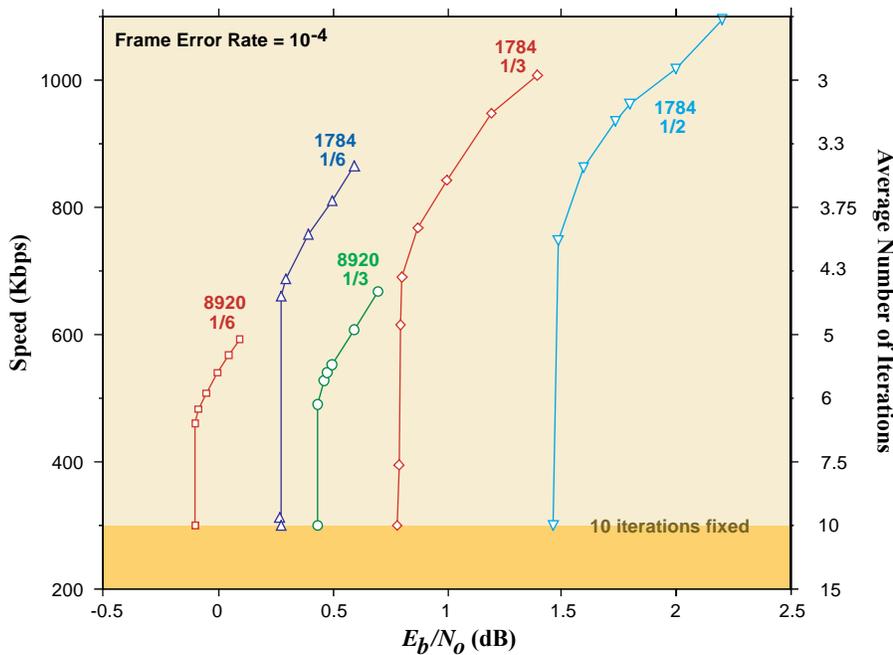
Figure 7-11: Illustration of Decoder Speedup Using Stopping Rules

The *x*-axis shows the threshold value of E_b/N_0 required to reach a WER of 10^{-4}. The *y*-axis shows the average decoding speed, or reciprocally the average number of iterations. In this figure a decoder using a fixed 10 iterations achieves a speed of 300 Kbps, and the decoder's average speed increases inversely as the average number of iterations is reduced by application of the stopping rule. The results in this figure are for a selection of recommended turbo codes with block lengths 1784 and 8920. The figure shows that effective stopping rules can increase the decoder speed on the order of 50% to 100% with virtually no compromise in the required value of E_b/N_0; further increases in speed can also be obtained by trading off additional SNR for increased speed.

7.4.2 COMPARISON TO TRADITIONAL CONCATENATED CODES

Turbo codes gain a significant performance improvement over the traditional Reed-Solomon and convolutional concatenated codes currently recommended by CCSDS. For example, to achieve an overall BER of 10^{-6} with a block length of 8920 bits (depth-5 interleaving), the required bit-SNRs are approximately 0.8 dB, 1.0 dB, and 2.6 dB for the DSN's standard codes consisting of the (255,223) Reed-Solomon code concatenated with the (15,1/6) convolutional code, the (15,1/4) convolutional code, and the (7,1/2) convolutional code, respectively. The performance gains achieved by the corresponding-rate turbo codes in figures 7-6, 7-7, 7-8, 7-9, and 7-10 range from 0.9 dB to 1.6 dB.

Figure 7-12 compares the performance of the recommended turbo codes of block length 1784 bits and rates 1/3 and 1/6 with the performance of the CCSDS concatenated code used by Voyager and that of the non-CCSDS concatenated code used by Cassini and Mars Pathfinder. The Voyager code consists of the recommended concatenation of the (255, 223) Reed-Solomon code with the (7,1/2) convolutional code. The Cassini/Pathfinder code consists of the same Reed-Solomon code concatenated with a (15, 1/6) convolutional code for which the Viterbi decoder requires $2^8 = 256$ times as many states as for the (7, 1/2) code. Performance for both concatenated codes is obtained using an interleaving depth of $I = 1$, not the actual interleaving depths used in the Voyager/Cassini/Pathfinder missions, in order to provide a fair comparison with the performance of the two turbo codes with block length 1784. In other words, a frame length of 1784 bits is assumed for all four curves in this figure.
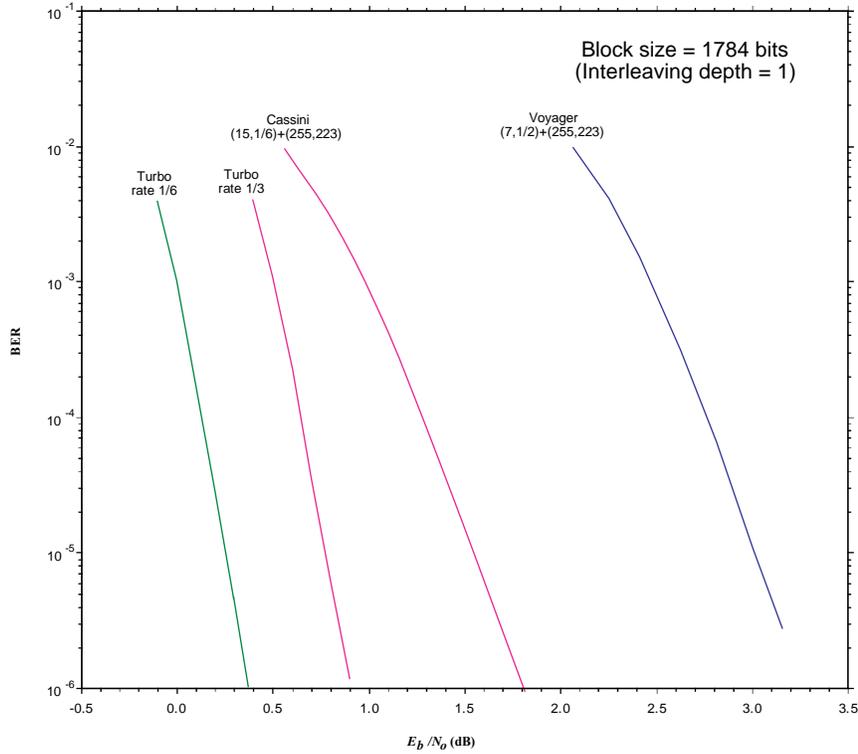
Figure 7-12: BER Performance of Turbo Codes Compared to Older CCSDS Codes (Except Cassini/Pathfinder Code: Reed-Solomon (255,223) + (15,1/6) Convolutional Code), Block Size 1784 Bits (Interleaving Depth = 1), Software Simulation, 10 Iterations

Figure 7-13 compares the performance of the recommended turbo codes of block length 8920 bits and rates 1/3 and 1/6 with the performance of the Voyager and Cassini/Pathfinder concatenated codes, now allowed to have interleaving depth $I = 5$ in order to produce equal-length frames of 8920 bits for all codes shown.
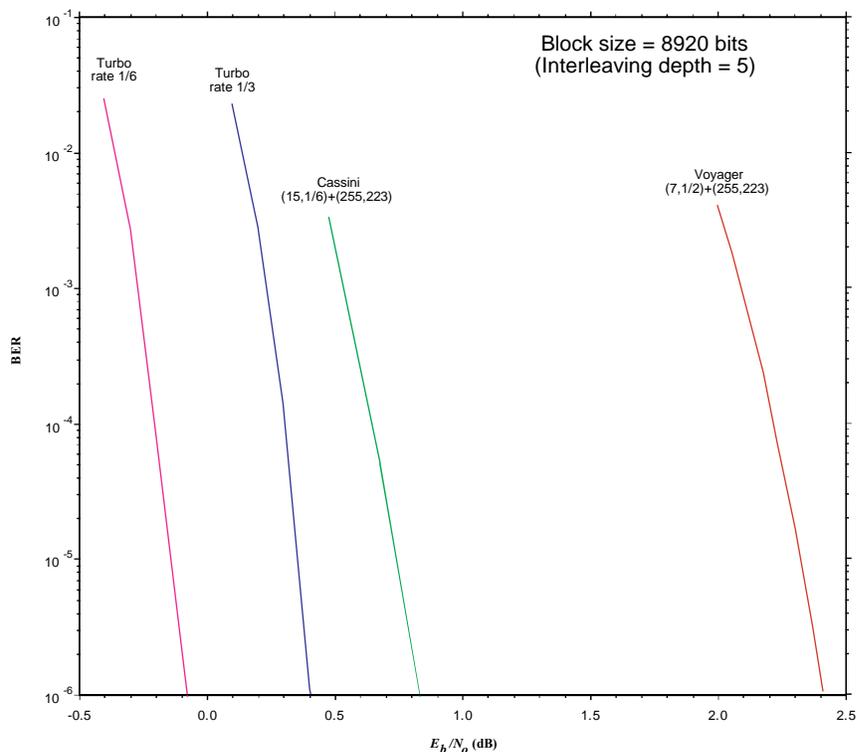
Figure 7-13: BER Performance of Turbo Codes Compared to Older CCSDS Codes (Except Cassini/Pathfinder Code: Reed-Solomon (255,223) + (15,1/6) Convolutional Code), Block Size 8920 Bits (Interleaving Depth = 5), Software Simulation, 10 Iterations

7.4.3 THE TURBO DECODER ERROR FLOOR

Although turbo codes can be found to approach the Shannon-limiting performance at very small required bit error rates, the turbo code's performance curve does not stay steep forever as does that of a convolutional/Reed-Solomon concatenated code. When it reaches the so-called 'error floor', the curve flattens out considerably and looks from that point onward like the performance curve for a weak convolutional code. In the error floor region, the weakness of the constituent codes takes charge, and the performance curve flattens out from that point onward. The error floor is not an absolute lower limit on achievable error rate, but it is a region where the slope of the turbo code's error rate curve becomes dramatically poorer.

There exist transfer function bounds on turbo code performance (reference [15]) that accurately predict the actual turbo decoder's performance in the error floor region above the so-called 'computational cutoff rate' threshold, below which the bounds diverge and are useless. More advanced bounds which are tight at lower values of bit SNR were developed in reference [29]. These bounds are computed from the code's weight enumerator which is not readily available for the recommended turbo codes. Approximations valid in the error floor region can be obtained from considering only codewords of the lowest weight(s). Reference [30] gives a method for calculating the minimum distance of the recommended

codes and the corresponding estimates of BER on the error floor. Other details on algorithms for computing CCSDS turbo code minimum distance and error floors can be found in reference [32].

Figure 7-14 provides an illustration of the transition of a turbo code performance curve from a steep 'waterfall' region into a much flatter 'error floor' region for two turbo codes analyzed as an example. This figure shows the actual simulated turbo code performance compared with bounds approximating the error floor.

The original turbo codes of Berrou et al. (reference [17]) had error floors starting at a BER of about 10^{-5}. By using theoretical predictors as guides, it was possible to design the turbo codes in the Recommended Standard (reference [3]) so as to lower the error floor to possibly insignificant levels (e.g., as low as 10^{-9} bit error rate).
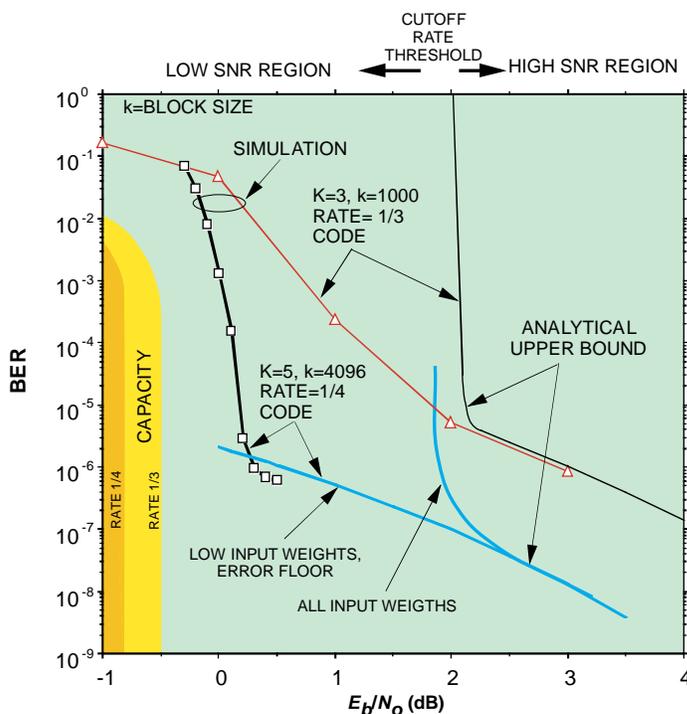
Figure 7-14: Illustration of Turbo Code Error Floor

8 IMPORTANT ANCILLARY ASPECTS OF THE CODING SYSTEM

8.1 GENERAL

The preceding four sections have described how one would encode and decode each of the recommended codes, and their corresponding performance, under ideal circumstances. CCSDS Recommended Standards (references [3] and [2]) also impose certain ancillary conditions on the coding system in order to approach this ideal performance in a practical system. Chief among these ancillary requirements addressed in Recommended Standards (references [3] or [2]) are the following:

    a) the coded output of all codes (or of uncoded data) must be sufficiently random to ensure proper receiver operation;

    b) there must be a method for synchronizing the received data with the codeblock boundaries;

    c) there must be a way to certify the validity of decoded data with high certainty.

There are a couple of additional ancillary issues associated with the recommended codes:

    a) some of the recommended codes are 'transparent' to inversion of the received data, and some are not;

    b) 1:1 remappings of the information or coded bits may be permitted but may affect performance.

8.2 RANDOMIZATION OF THE CODED OUTPUT

8.2.1 GENERAL

Randomization of the data stream provides three useful functions. It aids in achieving:

    – signal acquisition;

    – bit synchronization;

    – ambiguity resolution for convolutional decoder operation.

Receiver acquisition performance is often impaired by short periodic data patterns. Randomizing the data avoids this.

In order to acquire and maintain symbol synchronization with the coded symbol boundaries, a bit synchronizer requires a sufficient symbol transition density. The recommended non-punctured (7,1/2) convolutional code contains an inverter on one of its outputs, which assures a sufficient symbol transition density when this code is used with BPSK modulation. Although this inverter may be sufficient for proper operation of the bit synchronizer, it does not guarantee that the receiver and decoder will work correctly. In contrast, when the

recommended Reed-Solomon code is used alone, or the data is uncoded, there may be no symbol transitions, e.g., if all-zero data is sent.

While alternate symbol inversions solve the symbol synchronization problem for the case of convolutional codes with BPSK modulation, it is desirable to offer a universal solution for all three issues and any of the recommended codes. The pseudo-randomizer defined in section 7 of reference [3] gives such a solution. This randomizer adds (modulo-2) a pseudo-random sequence to the coded symbols. The result is a maximally random sequence of 0s and 1s regardless of the transition density characteristic of the particular code's output. The pseudo-randomizer is likely to solve the issues that can arise from non-random data for all combinations of CCSDS-recommended modulation and coding.

8.2.2   DESCRIPTION OF THE RECOMMENDED PSEUDO-RANDOMIZER

The method for ensuring sufficient transitions is to exclusive-OR each bit of the Codeblock or Transfer Frame with a standard pseudo-random sequence. If the Pseudo-Randomizer is used, on the sending end it is applied to the Codeblock or Transfer Frame after turbo encoding or RS encoding (if either is used), but before convolutional encoding (if used). On the receiving end, it is applied to derandomize the data after convolutional decoding (if used) and codeblock synchronization but before Reed-Solomon decoding or turbo decoding (if either is used).

The configuration at the sending end is shown in figure 8-1.
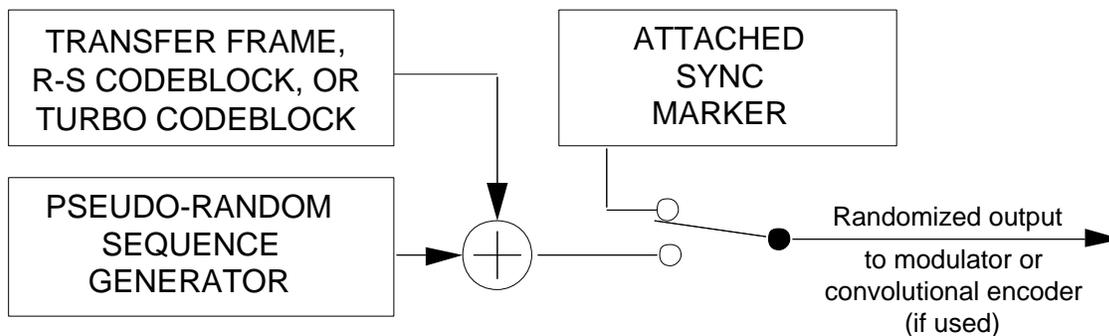
Figure 8-1: Block Diagram of the Recommended Pseudo-Randomizer

The Attached Sync Marker (ASM) is already optimally configured for synchronization purposes and it is therefore used for synchronizing the Pseudo-Randomizer. The pseudo-random sequence is applied starting with the first bit of the Codeblock or Transfer Frame. On the sending end, the Codeblock or Transfer Frame is randomized by exclusive-ORing the first bit of the Codeblock or Transfer Frame with the first bit of the pseudo-random sequence, followed by the second bit of the Codeblock or Transfer Frame with the second bit of the pseudo-random sequence, and so on. On the receiving end, the original Codeblock or Transfer Frame is reconstructed using the same pseudo-random sequence. After locating the ASM in the received data stream, the pseudo-random sequence is exclusive-ORed with the data bits

immediately following the ASM. The pseudo-random sequence is applied by exclusive-ORing the first bit following the ASM with the first bit of the pseudo-random sequence, followed by the second bit of the data stream with the second bit of the pseudo-random sequence, and so on.

The pseudo-random sequence used in the CCSDS standard is generated by using the following polynomial:

$$h(x) = x^8 + x^7 + x^5 + x^3 + 1$$

This sequence begins at the first bit of the Codeblock or Transfer Frame and repeats after 255 bits, continuing repeatedly until the end of the Codeblock or Transfer Frame. The sequence generator is initialized to the all-ones state at the start of each Codeblock or Transfer Frame.

The first 40 bits of the pseudo-random sequence from the generator are shown below; the leftmost bit is the first bit of the sequence to be exclusive-ORed with the first bit of the Codeblock or Transfer Frame; the second bit of the sequence is exclusive-ORed with the second bit of the Codeblock or Transfer Frame, and so on.

1111 1111 0100 1000 0000 1110 1100 0000 1001 1010 ...

8.2.3 USAGE CIRCUMSTANCES FOR THE RECOMMENDED PSEUDO-RANDOMIZER

The Recommended Standard (reference [3]) does not always require the use of the universal solution provided by the pseudo-randomizer. As we have seen, its use would be superfluous in the case of convolutional coding with alternate symbol inversions and BPSK modulation. Less conclusively, turbo codes might inherently provide a sufficient coded symbol transition density due to their recursive convolutional encoding of non-zero data headers at the beginning of each data block. Other codes might obtain sufficient transitions if their input information bits are guaranteed to be sufficiently random. I&T project personnel may prefer un-randomized data so that during testing, they can read the binary data that they are familiar with. One answer is to implement the recommended pseudo-randomizer but make it switchable so that during early testing it can be turned off.

While the recommended pseudo-randomizer is not strictly required, the system engineer must take all necessary steps to ensure that the coded symbols have sufficient transition density. Several projects have encountered unexpected problems with their telemetry links because this pseudo-randomizer was not used and sufficient randomness was not ensured by other means and properly verified. These problems are traced to a lack of randomization at the data or modulation symbol level. In many communication system designs, the receiver, bit/symbol synchronizer and convolutional decoder all have specific requirements that are met by using randomized data. Details may change depending on modulation type, data format (NRZ-L vs. Bi Phase L) and signal to noise ratio. If the implementer can adequately prove that a symbol stream with the proper randomness and balance of 1s and 0s can be achieved without the use of the recommended pseudo-randomizer to 1) ensure a high probability of receiver acquisition and lock in the presence of data, 2) eliminate DC offset

problems in PM systems, 3) ensure sufficient bit transition density to maintain bit (or symbol) synchronization, and 4) to handle special coding implementations (i.e., data that is multiplexed into multiple convolutional encoders), then the recommended Pseudo-Randomizer may be omitted.

The presence or absence of Pseudo-Randomization is fixed for a physical channel and is managed (i.e., its presence or absence is not signaled in the telemetry but must be known a priori) by the ground system.

8.3   CODEBLOCK SYNCHRONIZATION

8.3.1   GENERAL

Each of the recommended codes requires a method for aligning the sequence of received code symbols with the boundaries of its codeblocks (or code symbol periods in the case of convolutional codes). Otherwise, the decoder would fail because it would be applying the correct decoding algorithm to an incorrect subset of received code symbols. The synchronization requirements are different for each of the recommended codes, as described in the next four subsections.

8.3.2   SYNCHRONIZATION FOR CONVOLUTIONAL CODES

For a rate $1/n$ convolutional code, the encoding rule, and hence the decoding rule, are 'time-invariant' in that the same rule is applied at each bit time. Thus, even though the convolutional codeword is indefinitely long, the only requirement for proper synchronization is to correctly establish the identity of the starting symbol of any group of n symbols produced in one bit time. This procedure is commonly called 'node synchronization'. For the recommended rate-1/2 non-punctured convolutional code, as well as the entire series of recommended punctured convolutional codes derived from the rate-1/2 code, node synchronization is a relatively simple matter of distinguishing between two possible 'phases' of the received symbol stream. This can be accomplished with or without the aid of frame synchronization markers in the data. For example, the Viterbi decoder may determine the correct phase by monitoring the rate of growth of its own internal metrics. Some useful techniques for node synchronization are described in reference [13]. Alternatively, for the recommended rate-1/2 convolutional code, node synchronization and frame synchronization can be established simultaneously by locating the (52-symbol invariant part of the) convolutionally encoded synchronization marker within the received symbol stream.

8.3.3   SYNCHRONIZATION FOR REED-SOLOMON CODES

A Reed-Solomon decoder will only decode properly if the starting symbol of each codeword is identified; i.e., the decoder requires accurate codeword synchronization. If interleaving is used, further resolution is necessary to determine the starting symbol of each codeblock (interleaved set of I codewords), or else the de-interleaver will fail to work properly.

The recommended method for synchronizing the codeblock is to look for an attached synchronization marker of 32 bits. This procedure is commonly called 'frame synchronization', because, in the absence of Reed-Solomon coding, the same 32-bit synchronization marker is attached directly to the Transfer Frame and is used to locate the start of the frame. When Reed-Solomon coding is used, the 32-bit marker is attached to the beginning of the Reed-Solomon codeblock and is used in the same way to identify the starting symbol of a codeblock. In this case, the synchronization procedure is properly called 'codeblock synchronization', but the term 'frame synchronization' is often used indiscriminately to cover both cases.

It is important to note that the codeblock synchronization marker is not encoded by the Reed-Solomon encoder. Thus, even though the same 32-bit marker is attached to the same block of information bits, whether they occur in an uncoded Transfer Frame or as the data bits in a systematic Reed-Solomon codeblock, the Reed-Solomon coding cannot be considered a totally separate layer that follows the attachment of the marker to the Transfer Frame. If the coding layer should receive a Transfer Frame with frame synchronization marker already attached, it must detach the marker, encode the Transfer Frame only, and reattach the marker to the encoded codeblock.

8.3.4 SYNCHRONIZATION FOR CONCATENATED CODES

Synchronization for concatenated codes requires finding proper alignment with the boundaries of both constituent codes. The Recommended Standard (reference [3]) requires that the same 32-bit synchronization marker be attached to the recommended Reed-Solomon code, regardless of whether it is concatenated with an inner convolutional code. At the receiving end, the two levels of synchronization can be established by first node-synchronizing the inner convolutional code, and then locating the 32-bit synchronization marker after convolutionally decoding. Alternatively, when the inner code is the recommended rate-1/2 convolutional code, node synchronization and frame synchronization can be established simultaneously by locating the (52-symbol invariant part of the) convolutionally encoded synchronization marker within the received symbol stream.

8.3.5 SYNCHRONIZATION FOR TURBO CODES

Codeblock synchronization is necessary for proper decoding of turbo codeblocks. Synchronization of the turbo codeblocks is achieved by using an attached sync marker. The code symbols comprising the sync marker for the turbo code are attached directly to the encoder output without being encoded. Thus, the transmitted sync marker pattern remains static for each codeblock.

Synchronization is acquired on the receiving end by recognizing the specific bit pattern of the sync marker in the raw (undecoded) telemetry channel data stream. Synchronization is then confirmed by making further checks. Frame synchronizers should be set to expect a marker at a recurrence interval equal to the length of the sync marker plus that of the turbo codeblock.

A generic block diagram for generating a turbo codeblock with attached sync marker was already shown in figure 7-2. A diagram of the resulting codeblock with attached marker is shown in figure 8-2. Note that the lengths of the turbo codeblock and the sync marker are both inversely proportional to the nominal code rate r. This yields roughly equivalent synchronization performance independent of code rate.

Rate-Dependent
Attached Sync
Marker

Turbo Codeblock

| 32/r bits | K/r bits | 4/r bits |

r = 1/2, 1/3, 1/4, or 1/6 (nominal code rate)
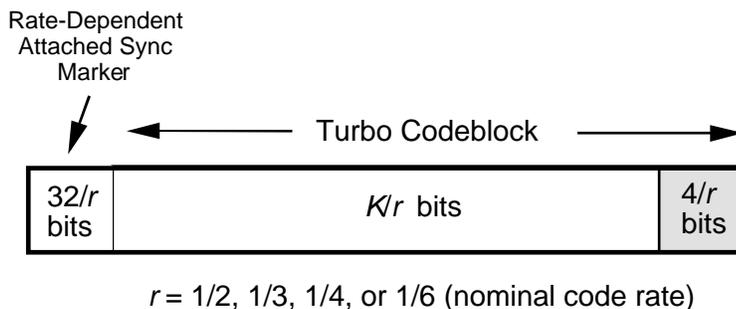
K = Telemetry Transfer Frame Length or Information Block Length

Figure 8-2: Turbo Codeblock with Attached Sync Marker

Note that frame sync for the recommended Reed-Solomon/convolutional concatenated code can be acquired using a sync marker defined in the information bit domain rather than the encoded symbol domain, and detected after Viterbi decoding. This method relies on the fact that frame sync is not required for successful operation of the Viterbi decoder but is necessary for decoding the Reed-Solomon code. The Viterbi decoder is capable of finding its own 'node sync' with or without the aid of known sync markers in the data stream. The Reed-Solomon decoder has no effective method (other than trial and error) for determining frame sync on its own, and so it must be presented with externally synchronized codeblocks. It is irrelevant to the performance of the RS decoder whether this synchronization is determined from the channel symbols or from Viterbi decoded bits.

In a similar way, the turbo decoder relies on being handed externally synchronized codeblocks, but a bit-domain approach does not work effectively for turbo decoders, because each constituent convolutional decoder is too weak by itself to detect a reasonable size marker reliably, and because the powerful combined turbo decoding operation needs to know the codeblock boundaries before it can iterate between permuted and unpermuted data domains. Therefore, turbo code applications need to use channel-symbol-domain frame sync methods as specified in the Recommended Standard (reference [3]).

Note that, for equivalent performance, channel-symbol-domain frame synchronization requires longer sync markers and faster processing (at the channel symbol rate rather than the Viterbi decoded bit rate).

8.4 CERTIFICATION OF THE DECODED DATA (FRAME INTEGRITY CHECKS)

8.4.1 GENERAL

The CCSDS applications are packet-oriented, which means that data are collected and transmitted in frames. With all coding options, and also for uncoded data, it is important to have a reliable indication whether the decoded data is correct. A frame integrity check can be used at the receiver side to validate the received frame or, when suitable, for requiring retransmission in case of check failure.

As with the problem of randomizing the coded output, a universal solution to this data validation problem exists in the form of a cyclic redundancy check (CRC) code, as specified in the *TM Space Data Link Protocol* Blue Book (reference [2]).

8.4.2 DESCRIPTION OF THE RECOMMENDED CRC CODE

Generally speaking, a binary CRC code is an (N,k) code obtained by shortening a cyclic code, capable of detecting the following error patterns:

a) all error bursts of length $N–k$ or less;

b) a fraction of error bursts of length equal to $N–k+1$; this fraction equals $1–2^{-(N–k–1)}$;

c) a fraction of error bursts of length greater than $N–k+1$; this fraction equals $1–2^{-(N–k)}$;

d) all error patterns containing $d_{min} – 1$ (or fewer) errors, d_{min} being the minimum distance of the CRC code;

e) all error patterns with an odd number of errors if the generator polynomial $G(D)$ for the code has an even number of nonzero coefficients.

The circuits for coding and syndrome computation are simple feedback shift registers with $r = N–k$ cells.

In the CRC code used for the CCSDS *TM Space Data Link Protocol* Recommended Standard, 16 parity check bits are added to every information frame consisting of $(N–16)$ information bits, according to the following generator polynomial:

$$G(D) = D^{16} + D^{12} + D^5 + 1$$

Thus the rate of this CRC code is $(N–16)/N$.

The CRC circuit in the Recommended Standard (reference [2]) is preset to an all '1' state prior to encoding; this is a peculiarity of the CCSDS CRC, which implies that the 16 parity check bits are inverted with respect to the usual CRC encoding $(N–16)$ all '0' information bits generate 16 all '1' parity check bits).

Serial concatenation of the CRC and a turbo code with nominal rate 1/3 is shown in figure 8-3.
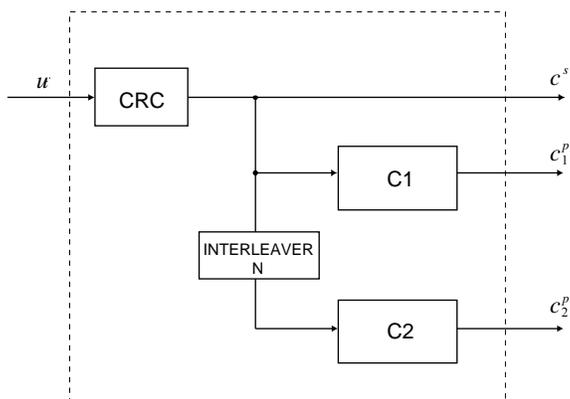
Figure 8-3: Turbo-CRC Encoder

The information frame is encoded by the CRC before entering the turbo encoder, and the CRC syndrome is used to check the integrity of the decoded frame produced by the turbo decoder at the receiver side.

8.4.3   USAGE CIRCUMSTANCES FOR THE RECOMMENDED CRC CODE

The recommended CRC code is included in the Telemetry Frame and consists of 16 check bits computed from the remainder of the frame contents. This code can reliably detect incorrect frames with an undetected error rate of around $2^{-15} \approx 10^{-5}$. This CRC code achieves approximately the same undetected error rate for any of the recommended telemetry channel codes.

A much lower undetected error rate is achieved when the RS code with $E = 16$ is used, either by itself or concatenated with an inner convolutional code. In this case, the undetected error rate of the RS decoder is on the order of $1/E! \approx 10^{-13}$, which is many orders of magnitude better than the validation offered by the CRC code. Thus, the error detection capability of the CRC code is superfluous when the RS code with $E = 16$ is used.

The RS code with $E = 8$ offers much lower error detection capability, on the same order as that provided by the 16-bit CRC code. Similarly, a turbo decoder equipped with a smart stopping rule that notes whether the decoder's iterations converge to a valid codeword can achieve some degree of error detectability and somewhat alleviate the need for the 16-bit CRC code. However, in these borderline cases the CRC code is still required. It is also required for uncoded data or convolutionally coded data, which offer absolutely no capability for error detection on their own.

If a lower detected error rate is desired than that offered by the recommended 16-bit CRC code, and RS coding is not used, then one option is to use a 32-bit or 48-bit CRC code (not in the CCSDS Recommended Standards).

8.5   CODE TRANSPARENCY

Rotationally invariant (transparent) coding schemes are used to overcome the phase ambiguity inherent in usual coherent demodulation techniques. Let us consider the transmission over a band-limited channel using phase-coherent demodulation. To estimate the carrier phase, the receiver uses its knowledge of the signal set S, which is the set of points produced by the modulator. By examining the pattern of received signal points, the receiver can infer the carrier phase up to an ambiguity corresponding to a rotational symmetry of S.

Let us denote a counterclockwise rotation of x degrees about the origin by ρ. A rotational symmetry of the signal set S is a rotation ρ mapping S into itself. The set of all the rotational symmetry of S is called the rotational symmetry group ϕ. If ϕ has n elements then it is a cyclic group generated by the rotation ρ of $x = 360/n$ degrees (the smallest non-zero rotation belonging to it).

As an example, an M-PSK constellation has M rotational symmetries. In particular, a 2-PSK constellation has 2 rotational symmetries: $\phi = \{\rho_0, \rho_{180}\}$, while a 4-PSK constellation has 4 rotational symmetries $\phi = \{\rho_0, \rho_9, \rho_{180}, \rho_{270}\}$, as a square QAM constellation (16-QAM, 64-QAM, 256-QAM). For non-square QAM constellations, ϕ depends on the signal choice.

When used in a modulation scheme with coherent demodulation, the carrier phase is estimated from the ensemble of the received signal points. However, an ambiguity corresponding to a rotation of ϕ cannot be solved without external reference. For example, if a 2-PSK is used, the demodulator observes the two received points and estimates a carrier phase which can be correct, or wrong by 180 degrees.

The receiver can handle the n-way phase ambiguity in several ways. One way to resolve the phase ambiguity is through training. At the start of the transmission, and within it, the transmitter sends a predetermined sequence of signal points which the receiver uses to correct its phase estimation.

Another method uses transparent coding schemes to solve the problem. In this case, the receiver does not try to resolve the possible phase error but uses transparent schemes able to cope with it. Let us consider an uncoded signal set S=2-PSK transmitted over a channel without noise. If a 180-degree error occurs at the receiver side, all the transmitted bits are received inverted. We observe that this is equivalent to sum an all-one sequence to the transmitted sequences. A simple differential precoder at the transmitter side, followed by a differential postcoder at the receiver can cope with this situation (see figure 3-1 and figure 8-4 below). In fact, the constant all-one sequence is eliminated by the differential devices.

Now, let us consider a binary code C mapped over S=2-PSK. The precoder/postcoder operation could still be applied to cope with possible phase errors. However, it is essential that a rotation of 180 degrees maps the code into itself (otherwise, in case of phase error, the decoder would work over a different set of codewords). In this case we say that C is rotationally invariant (transparent): for any code sequences $c \in C$, its inverted version still belongs to C: a differential precoder/postcoder pair is able to solve the phase ambiguity of the coded sequences.

8.6   REMAPPINGS OF THE BITS

In figure 3-1 there is an optional 'NRZ-L to NRZ-M conversion' block at the transmitter and, inversely, an 'NRZ-M to NRZ-L conversion' block at the receiver. NRZ-L is a modulation format that represents a data '1' by one of two levels, and a data '0' by the other level. On the other hand, NRZ-M represents a data '1' by a change in level and a data '0' by no change in level. The conversion from NRZ-L to NRZ-M is a form of differential precoding that can be used to resolve the ambiguity between true and complemented data. figure 8-4 shows a block diagram for implementing the 'NRZ-L to NRZ-M conversion' and its inverse.


(a)   (b)
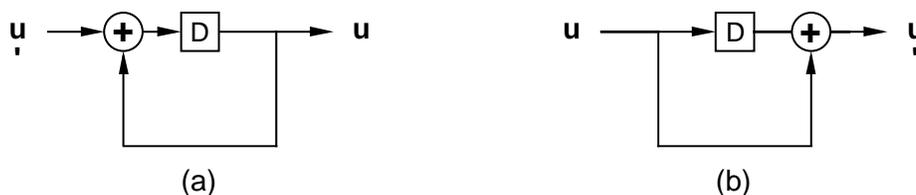
Figure 8-4: Block Diagrams for Implementing the (Optional) (a) 'NRZ-L to NRZ-M Conversion' and (b) Its Inverse

When all three elements of the coding system depicted in figure 3-1 are used, the 'NRZ-L to NRZ-M conversion' is actually just a form of 1:1 mapping applied to the binary data, not a conversion of modulation formats, since the modulation of the data occurs after the convolutional encoding stage. Any invertible mapping may be applied to the binary data without apparent consequence as long as all the data bits are correct; however, performance is affected by 1:1 mappings when errors enter the system. For example, there is a large performance penalty if one puts an NRZ-L to NRZ-M mapping at the output of a convolutional code (see reference [31]). For this reason figure 3-1 does not include an option that allows an 'NRZ-L to NRZ-M conversion' block to serve as a true modulation conversion at the output of the convolutional encoder.

The CCSDS Recommended Standards (references [3] and [2]) do not regulate whether a user's source data might be subjected to a 1:1 mapping (or any other form of data processing) before being packaged as information bits in a Telemetry Transfer Frame prior to coding. Thus any form of 1:1 mapping of the source data that precedes any of the recommended CCSDS codes is implicitly allowed by Recommended Standards (references [3] and [2]). In this case, the code performance curves shown in this Green Book pertain only to the error rates for the remapped data presented to the encoder. The user has the responsibility to determine whether these errors might propagate or multiply throughout the original source data as a result of the 1:1 premapping. For example, the discussion following figure 5-4 mentioned two methods (row-by-row and column-by-column) for reading the source data into the matrix used for interleaving Reed-Solomon codewords; this choice affects the characteristics of errors in the decoded source data.

The 'NRZ-L to NRZ-M conversion' block in figure 3-1 can be viewed simply as an implicitly permitted 1:1 remapping of the source data in the case when the Reed-Solomon code is not used. Curiously, however, the figure also indicates that this mapping may be placed *between* the two components of a concatenated code. This placement makes sense from a performance standpoint: unlike an NRZ-L to NRZ-M mapping at the output of a convolutional code, the same mapping applied to Reed-Solomon coded bits has only minor effects on the code's performance . However, in this position this remapping in fact makes the overall code a concatenation of three codes, not two, when all three elements of the coding system depicted in figure 3-1 are used. The Blue Book (reference [3]) does not clearly state that such an arrangement is permitted. Also, figure 3-1 fails to show where the optional 'NRZ-L to NRZ-M conversion' block between the Reed-Solomon and convolutional codes fits with respect to the interleaving of Reed-Solomon codewords.

ANNEX A

GLOSSARY

Block Encoding: A one-to-one transformation of sequences of length k of elements of a source alphabet to sequences of length n of elements of a code alphabet, $n>k$.

Channel Symbol: The unit of output of the innermost encoder which is a serial representation of bits, or binary digits, which have been encoded to protect against transmission induced errors.

Clean Data (Bits): Data (bits) which are error free within the error detection and optional error correction capabilities of the TM System.

Codeblock: A codeblock of an (n,k) block code is a sequence of n channel symbols which were produced as a unit by encoding a sequence of k information symbols, and will be decoded as a unit. Code Rate: The average ratio of the number of binary digits at the input of an encoder to the number binary digits at its output.

Codeword: In a block code, one of the sequences in the range of the one-to-one transformation (see Block Encoding).

Command Link Control Word: The Telecommand System Transfer Layer protocol data unit for Telecommand reporting via the TM Transfer Frame Operational Control Field.

Concatenation: The use of two or more codes to process data sequentially with the output of one encoder used as the input of the next.

Constraint Length: In convolutional coding, the number of consecutive input bits that are needed to determine the value of the output symbols at any time.

Convolutional Code: As used in this document, a code in which a number of output symbols are produced for each input information bit. Each output symbol is a linear combination of the current input bit as well as some or all of the previous k-1 bits, where k is the constraint length of the code.

Fill Bit(s): Additional bit(s) appended to enable a 'data entity' to exactly fit an integer number of octets or symbols.

Inner Code: In a concatenated coding system, the last encoding algorithm that is applied to the data stream. The data stream here consists of the codewords generated by the outer decoder. Modulating Waveform: A way of representing data bits ('1' and '0') by a particular waveform.

NRZ-L: A modulating waveform in which a data 'one' is represented by one of two levels, and a data 'zero' is represented by the other level.

NRZ-M: A modulating waveform in which a data 'one' is represented by a change in level and a data 'zero' is represented by no change in level.

Octet: An 8-bit word consisting of eight contiguous bits.

Outer Code: In a concatenated coding system, the first encoding algorithm that is applied to the data stream.

Packet: An efficient application-oriented protocol data unit that facilitates the transfer of source data to users located in space or on Earth.

Protocol: A set of procedures and their enabling format conventions that define the orderly exchange of information between entities within a given layer of the TM System.

Reed-Solomon ('R-S') Symbol: A set of J bits that represents an element in the Galois field $GF(2^J)$, the code alphabet of a J-bit Reed-Solomon code.

Reliable: Meets the quality, quantity, continuity and completeness criteria which are specified by the TM System.

Segment: A protocol data unit which facilitates telemetry flow control through the breaking of long source packets into communications-oriented data structures.

Systematic Code: A code in which the input information sequence appears in unaltered form as part of the output codeword.

Telemetry System: The end-to-end system of layered data handling services which exist to enable a spacecraft to send measurement information, in an error-controlled environment, to receiving elements (application processes) in space or on Earth.

Transfer Frame: A communication oriented protocol data unit that facilitates the transfer of application oriented protocol data units through the space-to-ground link.

Transparent: The invisible and seemingly direct (virtual) transfer of measurement information from the spacecraft source application process to the user (receiving application process).

Transparent Code: A code that has the property that complementing the input of the encoder or decoder results in complementing the output.

User: A human or machine-intelligent process which directs and analyzes the progress of a space mission.

Virtual Channel: A given sequence of Transfer Frames, which are assigned a common identification code (in the Transfer Frame header), enabling all Transfer Frames who are members of that sequence to be uniquely identified. It allows a technique for multiple source application processes to share the finite capacity of the physical link (i.e., through multiplexing).

Virtual Fill: In a systematic block code, a codeword can be divided into an information part and a parity (check) part. Suppose that the information part is N symbols long (symbol is defined here to be an element of the code's alphabet) and that the parity part is M symbols long. A 'shortened' code is created by taking only S ($S<N$) information symbols as input, appending a fixed string of length N-S and then encoding in the normal way. This fixed string is called 'fill'. Since the fill is a predetermined sequence of symbols, it need not be transmitted over the channel. Instead, the decoder appends the same fill sequence before decoding. In this case, the fill is called 'Virtual Fill'.

Connection Vector (Forward): In convolutional and turbo coding, a vector used to specify one of the parity checks to be computed by the shift register(s) in the encoder. For a shift register with s stages, a connection vector is an s-bit binary number. A bit equal to one in position i (counted from the left) indicates that the output of the ith stage of the shift register is to be used in computing that parity check.

Connection Vector (Backward): In turbo coding, a vector used to specify the feedback to the shift registers in the encoder. For a shift register with s stages, a backward connection vector is an s-bit binary number. A bit equal to one in position i (counted from the left) indicates that the output of the ith stage of the shift register is to be used in computing the feedback value, except for the leftmost bit which is ignored.

Trellis Termination: The operation of filling with zeros the s stages of each shift register used in the turbo encoder, after the end of the information block. During trellis termination the encoders continue to output encoded symbols for s-1 additional clock cycles.

Turbo Code: As used in this document, a block code formed by combining two component recursive convolutional codes. A turbo code takes as input a block of k information bits. The input block is sent unchanged to the first component code and bit-wise interleaved (see TURBO CODE PERMUTATION) to the second component code. The output is formed by the parity symbols contributed by each component code plus a replica of the information bits.

Turbo Code Permutation: A fixed bit-by-bit permutation of the entire input block of information bits performed by a permuter or interleaver, used in turbo codes.

ANNEX B

ACRONYMS AND ABBREVIATIONS

AOS — Advanced Orbiting System

APP — A posteriori probability

ASM — Attached Synchronization Marker

AWGN — Additive White Gaussian Noise

BCH — Bose-Chaudury-Hoquenheim

BER — Bit Error Rate

BPSK — Binary Phase Shift Keying

BSNR — bit SNR

CCSDS — Consultative Committee on Space Data Systems

CRC — Cyclic Redundancy Code

DSN — Deep Space Network

ESA — European Space Agency

FEC — Forward Error Correction

FER — Frame Error Rate

GF — Galois Field

GSFC — Goddard Space Flight Center

JPL — Jet Propulsion Laboratory

MAP — Maximum a posteriori probability

NASA — National Aeronautic and Space Administration

NRZ — Non-Return to Zero

PM — Phase Modulated

PSK — Phase Shift Keying

QAM — Quadrature Amplitude Modulation

RF — Radio Frequency

ROM — Read Only Memory

RS — Reed-Solomon

SNR — Signal to Noise Ratio

SSNR — Symbol SNR

TM — Telemetry

VC — Virtual Channel

WER — Word Error Rate

ANNEX C

RATIONALE FOR TURBO CODE PARAMETER SELECTIONS

C1   GENERAL

Because turbo codes can achieve great performance over a wide range of parameter values, the selection of reasonable code parameters is a major systems issue. The system design must assess all the parameter-space tradeoffs as they affect both the performance of the code and systems-related considerations. Turbo codes give the system designer vast flexibility to choose any desirable combination of parameters without sacrificing performance more than intrinsically necessary.

C2   CODE RATE

The code rate of the recommended turbo encoder is selectable from 1/2, 1/3, 1/4, or 1/6. Lower code rates are also possible to achieve even better performance if the receivers can work at the correspondingly lower channel-symbol SNR (E_b/N_0). The rule of thumb is that the potential coding gain for using lower code rates pretty much follows the corresponding gain for the ultimate code-rate-dependent theoretical limits.

For deep-space applications, turbo codes are intended for use with BPSK modulation, with code rate < 1 bit/channel symbol (spectral efficiency < 1 bit/sec/Hz). The same codes can be used with QPSK modulation with Gray coding signal assignment to achieve higher spectral efficiency, as typically required in near-Earth applications.[7]

C3   BLOCK SIZE

Figure 3-4 shows how some fundamental theoretical lower bounds on the performance of arbitrary codes on the additive white Gaussian noise channel vary with codeblock length. Amazingly, this variation is mirrored by the empirically determined dependence on block length of the performance of a large family of good turbo codes (see also reference [16]).

Figure C-1 shows simulation results compared to the lower bound for a family of rate-1/3 turbo codes with different block lengths (using the generator polynomials specified in 7.2). Note that the range of block lengths in this figure, from 256 bits up to 49152 bits, spans both larger and smaller block lengths than the five specific CCSDS recommended block lengths. Although there is a 2 dB performance differential between the simulation results for 256-bit blocks and 49152-bit blocks, the difference between the simulations and the lower bounds remains approximately the same. The simulation results are about 0.5 dB to 1.0 dB from the

---

[7] Additional turbo codes with matched modulation signal set have been designed for even higher spectral efficiencies. These codes would require 8PSK or higher level modulations and are not covered in this document.

theoretical limits for all code rates ranging from 1/6 to 1/2 and at all codeblock sizes ranging from 256 to 49152 information bits. Similar results were obtained for turbo codes in the same family with rates 1/2, 1/4, and 1/6. The significance of these results is that turbo codes appear to be uniformly good over the entire span of block sizes shown, including all of the CCSDS recommended block lengths.
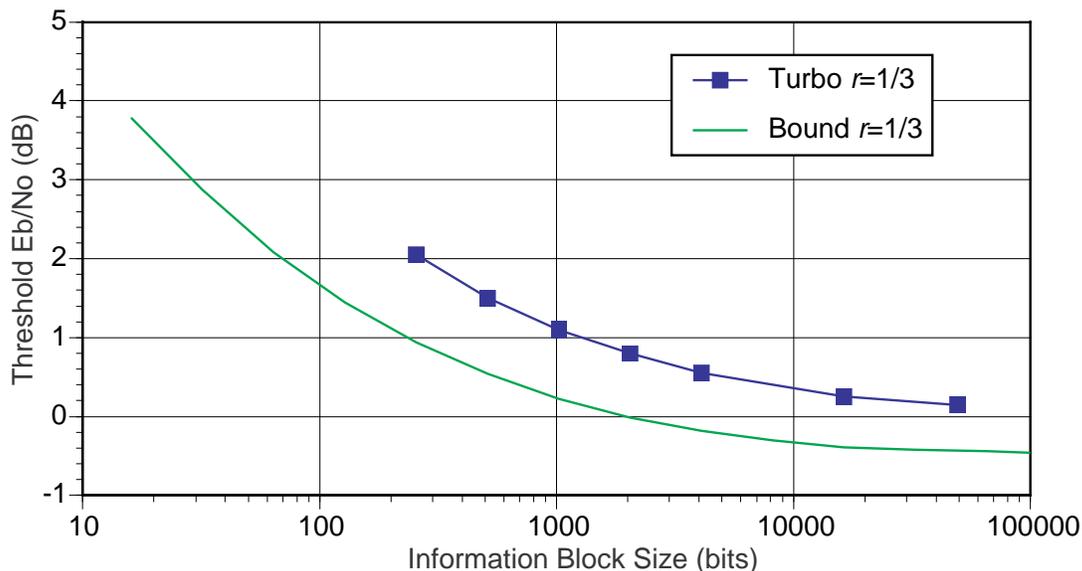
NOTE – Bound is calculated for word error rate of 10^{-4}, while turbo code simulations were for bit error rate of 10^{-6}.

Figure C-1: Comparison of Turbo Code Performance with Blocklength-Constrained Lower Bound

C4   CONSTITUENT CODES

Effective turbo codes can be constructed from a wide variety of constituents. Here are some of the factors underlying the choice of constituent codes that led to the recommended CCSDS turbo codes.

Number and Type of Constituent Codes — Turbo codes with more than two constituent codes are feasible in principle, but to this point they have not been well studied — mainly because two-component turbo codes already perform so well. The best performing and best understood constituent codes discovered thus far are the class of recursive convolutional codes, as recommended in 7.2 and in the original turbo code paper by Berrou *et al*.

Constraint Length — The recommended turbo code is formed from two recursive convolutional codes with constraint length $K = 5$. Higher constraint lengths are more complex to decode, and they seem to offer negligible performance improvement. In the other direction, constituent codes with constraint lengths less than 5 sacrifice some performance to achieve higher decoding speeds.

Code Generator Polynomials — Considerable theory has been developed to guide the choice of constituent code generator polynomials. This theory is based on the transfer function bounds that are used to predict the turbo decoder error floor. The error floor can be lowered the most if the divisor polynomial (G0 in figure 7-3) is a primitive polynomial. Additional theoretical considerations guide the choice of the remaining polynomials.

Code Transparency — Turbo codes are inherently non-transparent, meaning that a totally inverted codeblock cannot be an exact codeword. However, a turbo code can be made 'approximately transparent' except near the edges of the codeblock. It is a system design issue to decide whether an approximately transparent turbo code would be preferred, at some sacrifice of performance, to one designed without any transparency constraints. The turbo codes in the Recommended Standard (reference [3]) are *not* constructed to be approximately transparent.

C5   PERMUTATION

The (analytically) best-understood permutations for turbo codes are completely random. The best-performing permutations are manually optimized for each block size, and they also look very random. Manually optimized permutations generally outperform purely random permutations by only a small amount, except that they may significantly lower the error floor. However, such a permutation needs to be stored in ROM as a lookup table, because it is infeasible to recompute it on the fly for every codeword. The permutation in the Recommended Standard (reference [3]) can be generated on-the-fly by applying a simple rule. It also looks very random and performs nearly as well (within 0.1 dB, see figure C-2) as the manually optimized permutation. The recommended permutation gives the implementer an option to calculate the permutation on-the-fly in preference to using a look-up table. Note that a simple rectangular interleaver, such as the interleaver recommended for Reed-Solomon codes (see 5.3), is not suitable for turbo codes.

The interpretation of the permutation numbers in the Recommended Standard (reference [3]) is such that the sth bit read out on line 'in b' (in figure 7-3) is the $\pi(s)$th bit of the input information block, as shown in figure C-3.

C6   SOME SYSTEM ISSUES PERTINENT TO THE USE OF TURBO CODES

Lower symbol SNR — To take advantage of the improved performance of turbo codes, the receiving system must operate at a significantly lower symbol signal-to-noise ratio (SSNR) than that of a less powerful code with the same code rate. This imposes more stringent demands on the receiver's ability to perform symbol synchronization. The performance advantages of turbo coding may be negated if the receiver cannot lock onto the lower-SSNR symbols.

Since the threshold SSNR drops in direct proportion with the code rate, whereas the threshold bit signal-to-noise ratio (BSNR) converges to a fixed limit as code rate $\rightarrow 0$ (see 3.4.2), lowering the code rate too far toward 0 produces diminishing returns in overall code performance while continuing to tax the receiver heavily. It is a systems issue to decide on the code rate that provides the best tradeoffs. For turbo codes, the variation of code performance with code rate more or less mirrors that of the ultimate limits on performance, as given in 3.4.2.
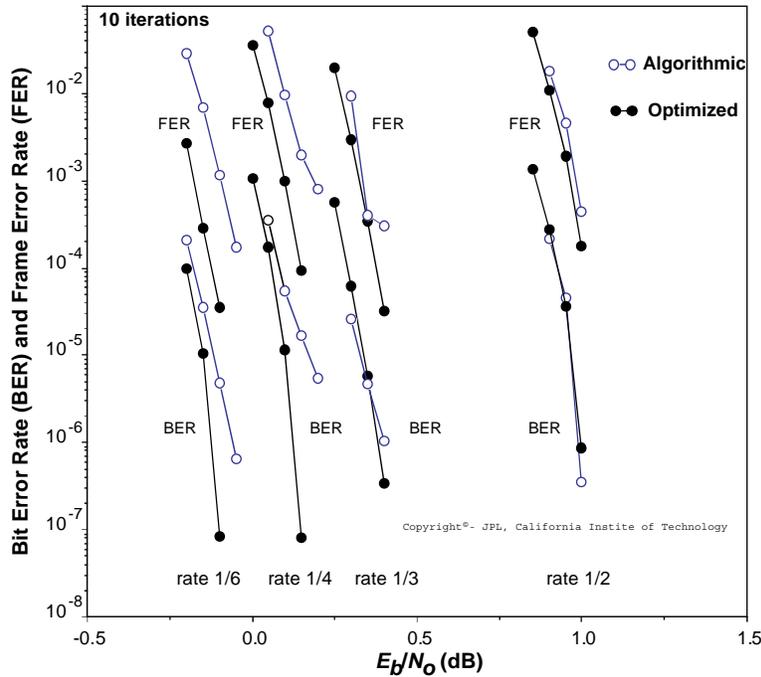
Figure C-2: Performance Comparison for Pseudo-Random and Algorithmic Permutations
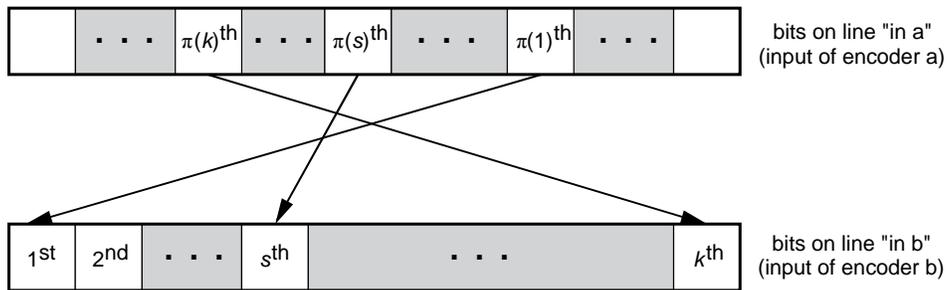

Figure C-3: Interpretation of Permutation

Performance with Non-Ideal Tracking Loops —Any decoder's performance degrades when there are small errors in tracking and detecting the received symbols. However, with turbo codes, there is also a possibility to improve the receiver's tracking performance by feeding back soft information from the decoding process to assist the receiver's tracking loops. Preliminary assessments (see reference [20]) of potential improvements are encouraging.

Residual Error Correction — In applications requiring extremely low error rates, the error rate of a turbo code in the error floor region may be unacceptable despite best efforts to lower it. The solution may be to add an outer code to work in conjunction with the turbo code as the inner code. The outer code would ideally be a binary code such as a BCH code rather than a nonbinary Reed-Solomon code. Because of the sparseness of errors on the error floor (typically a handful of bit errors per block), the outer code could have a very high code-

rate and would shift the required E_b/N_0 by just a tiny amount. However, an outer code will provide very little benefit at signal-to-noise ratios below the error floor region, because in this region there are frequently codewords for which the turbo decoding algorithm fails to converge and the resulting number of bit errors is beyond the error correction capability of any reasonable outer code. Unfortunately, these errors due to non-convergence of the decoding algorithm do not completely disappear in the error floor region, where they are similarly immune to being corrected by a reasonable outer code. Thus, even in the error floor region, an outer code is only effective at fixing the dominant error events, but the rarer events in this region (due to non-convergence) may still exceed the desired error rate if they are not rare enough.

Detecting Turbo Decoding Errors with an Outer CRC Code — Turbo decoders (like Viterbi decoders) are *complete* decoders, in that they always produce a decoded sequence. Currently these decoders do not detect and mark unreliable sequences, though in principle they could be modified to do so. Alternatively, a separate error detection code, such as a cyclic redundancy check (CRC) code, can be concatenated as an outer code with an inner turbo code, in order to flag unreliable decoded sequences. Let us define by ℓ the redundancy of the error detection code (CRC). The $\ell = 16$ CRC code used for the CCSDS standard detects every possible error sequence **e** with the lowest weights $|\mathbf{e}| = 1, 2,$ or 3. An undetected codeword error occurs whenever the error pattern **e** of the sequence decoded by the turbo code equals one of the nonzero codewords of the CRC code. The CRC/turbo code combination will produce a typical conditional undetected error probability of about $2^{-\ell} = 2^{-16} \approx 1.5 \times 10^{-5}$. This value must be multiplied by the probability of a codeword error to obtain the (unconditional) undetected error probability.

Lowering the Turbo Code's Error Floor — Even without using an outer BCH code, we have been able to design good turbo codes that lower the error floor to possibly insignificant levels (e.g., 10^{-9} bit error rate). Such performance may be sufficiently good for space applications to obviate the need for an outer error-correcting code. In that case, a simpler outer code (such as a CRC code) may still be desirable for error detection only.