An analysis of four error detection and correction schemes for the proposed Federal Standard 1024 (Land Mobile Radio)

Lohrmann, Carol A.

Monterey, California. Naval Postgraduate School
AN ANALYSIS OF FOUR ERROR DETECTION AND CORRECTION SCHEMES FOR THE PROPOSED FEDERAL STANDARD 1024 (LAND MOBILE RADIO)

by

Carol A. Lohrmann

March 1990

Thesis Advisor T. A. Schwendtner

Approved for public release; distribution is unlimited.
AN ANALYSIS OF ERROR DETECTION AND CORRECTION SCHEMES FOR THE PROPOSED FEDERAL STANDARD 1024 (LAND MOBILE RADIO)

Personal Author(s): Carol A. Lohmann

Type of Report: Master's Thesis

supported by

Land Mobile Radio (LMR), Federal Standard (FS), Error Detection and Correction (EDAC), Golay Codes, Hamming Codes, Quadratic Residue Codes (QRC).

This thesis surveys one area of the proposed FS 1024 for LMRs: namely, the error detection and correction (EDAC) of the message indicator (MI) bits used for cryptographic synchronization. Several EDAC codes are examined (Hamming, Quadratic Residue, hard decision Golay and soft decision Golay), tested on three FORTRAN programmed channel simulations (INMARSAT, Gaussian and constant burst width), compared and analyzed (based on bit error rates and percent of error-free superframe runs) so that a "best" code can be recommended. Out of the four codes under study, the soft decision Golay code (24,12) is evaluated to be the best. This finding is based on the code's ability to detect and correct errors as well as the relative ease of implementation of the algorithm.

Unclassified

Abstract security classification of this page

REPORT DOCUMENTATION PAGE

1. Title (Include security classification): AN ANALYSIS OF ERROR DETECTION AND CORRECTION SCHEMES FOR THE PROPOSED FEDERAL STANDARD 1024 (LAND MOBILE RADIO)

12. Personal Author(s): Carol A. Lohmann

13a. Type of Report: Master's Thesis

13b. Date: March 1990

15. Page Count: 142

17. Cost/Date

Interoperability of commercial Land Mobile Radios (LMR) and the military's tactical LMR is highly desirable if the U.S. government is to respond effectively in a national emergency or in a joint military operation. This ability to talk securely and immediately across agency and military service boundaries is often overlooked. One way to ensure interoperability is to develop and promote federal communications standards (FS).

This thesis surveys one area of the proposed FS 1024 for LMRs: namely, the error detection and correction (EDAC) of the message indicator (MI) bits used for cryptographic synchronization. Several EDAC codes are examined (Hamming, Quadratic Residue, hard decision Golay and soft decision Golay), tested on three FORTRAN programmed channel simulations (INMARSAT, Gaussian and constant burst width), compared and analyzed (based on bit error rates and percent of error-free superframe runs) so that a "best" code can be recommended. Out of the four codes under study, the soft decision Golay code (24,12) is evaluated to be the best. This finding is based on the code's ability to detect and correct errors as well as the relative ease of implementation of the algorithm.

Unclassified

Abstract security classification of this page

REPORT DOCUMENTATION PAGE

1. Title (Include security classification): AN ANALYSIS OF ERROR DETECTION AND CORRECTION SCHEMES FOR THE PROPOSED FEDERAL STANDARD 1024 (LAND MOBILE RADIO)

12. Personal Author(s): Carol A. Lohmann

13a. Type of Report: Master's Thesis

13b. Date: March 1990

15. Page Count: 142

17. Cost/Date

Interoperability of commercial Land Mobile Radios (LMR) and the military's tactical LMR is highly desirable if the U.S. government is to respond effectively in a national emergency or in a joint military operation. This ability to talk securely and immediately across agency and military service boundaries is often overlooked. One way to ensure interoperability is to develop and promote federal communications standards (FS).

This thesis surveys one area of the proposed FS 1024 for LMRs: namely, the error detection and correction (EDAC) of the message indicator (MI) bits used for cryptographic synchronization. Several EDAC codes are examined (Hamming, Quadratic Residue, hard decision Golay and soft decision Golay), tested on three FORTRAN programmed channel simulations (INMARSAT, Gaussian and constant burst width), compared and analyzed (based on bit error rates and percent of error-free superframe runs) so that a "best" code can be recommended. Out of the four codes under study, the soft decision Golay code (24,12) is evaluated to be the best. This finding is based on the code's ability to detect and correct errors as well as the relative ease of implementation of the algorithm.

Unclassified

Abstract security classification of this page
An Analysis of Four Error Detection and Correction Schemes for the proposed Federal Standard 1024 (Land Mobile Radio)

by

Carol A. Lohrmann
Civilian, GGE-12, Dept of Defense
B.S.E.E. Valparaiso University, 1984

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN SYSTEM TECHNOLOGY
(Command, Control, and Communications)

from the

NAVAL POSTGRADUATE SCHOOL
March 1990

Author:

Carol A. Lohrmann

Approved by:

T. A. Schwendtner, Thesis Advisor
Herschel Loomis, Second Reader

Carl Jones, Chairman,
Command, Control, and Communications Academic Group
ABSTRACT

Interoperability of commercial Land Mobile Radios (LMR) and the military's tactical LMR is highly desirable if the U.S. government is to respond effectively in a national emergency or in a joint military operation. This ability to talk securely and immediately across agency and military service boundaries is often overlooked. One way to ensure interoperability is to develop and promote federal communications standards (FS).

This thesis surveys one area of the proposed FS 1024 for LMRs; namely, the error detection and correction (EDAC) of the message indicator (MI) bits used for cryptographic synchronization. Several EDAC codes are examined (Hamming, Quadratic Residue, hard decision Golay and soft decision Golay), tested on three FORTRAN programmed channel simulations (INMARSAT, Gaussian and constant burst width), compared and analyzed (based on bit error rates and percent of error-free superframe runs) so that a "best" code can be recommended. Out of the four codes under study, the soft decision Golay code (24,12) is evaluated to be the best. This finding is based on the code's ability to detect and correct errors as well as the relative ease of implementation of the algorithm.
THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.
# TABLE OF CONTENTS

I. INTRODUCTION ................................................................. 1
   A. BACKGROUND .......................................................... 1
      1. Why Land Mobile Radio Standards are Needed ............. 1
      2. Current and Proposed LMR Standards ...................... 2
      3. Advantages and Disadvantages of FS 1024 ................. 3
   B. DESCRIPTION OF LAND MOBILE RADIO FS 1024 .............. 3
   C. SCOPE AND GOAL OF THESIS STUDY ............................ 4
   D. ORGANIZATION OF THESIS ........................................ 5
   E. THESIS COLLABORATION AND SUPPORT .......................... 5

II. TRANSMISSION FRAME STRUCTURE AND INTERLEAVING ............... 6
   A. FS 1024 SUPERFRAME AND FRAME ............................... 6
   B. PROPOSED INTERLEAVING .......................................... 9

III. EDAC CODES EXAMINED .................................................. 14
   A. INTRODUCTION .................................................... 14
   B. GOLAY CODES ..................................................... 14
   C. HAMMING CODES .................................................. 15
   D. QUADRATIC RESIDUE CODE (QRC) .............................. 16

IV. TESTS AND SIMULATION .................................................. 19
   A. DESCRIPTION OF THE PROGRAMS PROVIDED .................... 19
      1. Introduction .................................................. 19
      2. Modifications ................................................ 19
   B. SIMULATION DESCRIPTION ....................................... 19
   C. CHANNEL SIMULATIONS ......................................... 22
      1. INMARSAT Burst (mode = 1, vary = 0) ...................... 23
      2. AWGN Channel (mode = 0, vary = 0) ....................... 24
      3. Constant Burst Width (mode = 1, vary = 1) ............. 24

V. COMPARISON AND ANALYSIS OF TESTS .................................. 27
A. INMARSAT BURST CHANNEL ......................................................... 27
  1. BER Results ........................................................................ 27
  2. Run Success Failure Results .................................................. 28
  3. Additional Tests ................................................................... 30
B. GAUSSIAN NOISE CHANNEL ......................................................... 30
  1. BER Results ........................................................................ 30
  2. Run Success Failure Results .................................................. 31
C. CONSTANT BURST WIDTH CHANNEL ............................................. 34
  1. BER Results ........................................................................ 34
  2. Run Success Failure Results .................................................. 37
D. COMPARISON OF ERROR POSSIBILITIES ......................................... 44

VI. CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER STUDY 46
A. CONCLUSIONS ........................................................................ 46
B. RECOMMENDATIONS FOR FURTHER STUDY ............................. 47
  1. Study Continuation ............................................................... 47
  2. Implementation .................................................................... 47
  3. Other Codes ......................................................................... 47

APPENDIX A. FORTRAN CODE ......................................................... 49

APPENDIX B. SIMULATION DATA ...................................................... 108

APPENDIX C. INTERLEAVING TABLES ............................................... 122

LIST OF REFERENCES ................................................................ 125

INITIAL DISTRIBUTION LIST ......................................................... 128
LIST OF TABLES

Table 1. SUPERFRAME BIT COUNT (420 MS) ........................................... 6
Table 2. QUADRATIC RESIDUES FOR P = 19 ........................................... 17
Table 3. BIT, BAUD EQUIVALENT .............................................................. 21
Table 4. SIMPLIFIED FADE DURATION PDF [REF. 20] ................................. 23
Table 5. A FUNCTION OF INMARSAT BURST NOISE, ALL CODES ............. 27
Table 6. INMARSAT BURST - #SUCCESSES/#FAILURES ............................. 28
Table 7. BER AS A FUNCTION OF GAUSSIAN NOISE LEVEL, ALL CODES ...... 30
Table 8. #SUCCESSES %SUCCESFUL RUNS VS. GAUSSIAN NOISE LEVEL, ALL CODES ............................................................. 31
Table 9. BER AS A FUNCTION OF CONSTANT BURST WIDTHS, ALL CODES ................................................................. 37
Table 10. #SUCCESSES %SUCCESFUL RUNS VS CONSTANT BURST WIDTHS, ALL CODES ......................................................... 40
LIST OF FIGURES

Figure 1. Data Rate Allocation ........................................... 7
Figure 2. Transmission Format ........................................... 9
Figure 3. Transmission Structure Breakdown .......................... 10
Figure 4. Superframe Composition ...................................... 13
Figure 5. Confidence Constellation ..................................... 16
Figure 6. Simulation Block Diagram ................................... 20
Figure 7. Simulation Design Approach ................................. 22
Figure 8. INMARSAT Channel Performance ......................... 25
Figure 9. Varying Fade Width Signals ................................. 26
Figure 10. INMARSAT BER Performance ............................... 29
Figure 11. Gaussian BER Performance (line graph) .................. 32
Figure 12. Gaussian BER Performance (bar chart) .................... 33
Figure 13. #Successes % Successful for AWGN channel (line graph) ............. 35
Figure 14. #Successes % Successful runs for AWGN channel (bar chart) ....... 36
Figure 15. Constant Burst Width Performance (line graph) .......... 38
Figure 16. Constant Burst Width Performance (bar chart) .......... 39
Figure 17. Constant Burst Width - S f (line graph) .................. 41
Figure 18. Constant Burst Width - S f (bar chart) ................... 42
Figure 19. Constant Burst Width - S f (comparison) .................. 43
Figure 20. Possible Errors Corrected ................................. 45
# LIST OF ACRONYMS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>APCO</td>
<td>Association of Public Safety Commission Officers</td>
</tr>
<tr>
<td>AWGN</td>
<td>Additive White Gaussian Noise</td>
</tr>
<tr>
<td>A/D</td>
<td>analog to digital modulation (usually infers D A)</td>
</tr>
<tr>
<td>BER</td>
<td>bit error rate</td>
</tr>
<tr>
<td>C3</td>
<td>Command Control and Communications</td>
</tr>
<tr>
<td>CELP</td>
<td>Codebook Excited Linear Predictive Coding</td>
</tr>
<tr>
<td>COMSEC</td>
<td>Communications Security</td>
</tr>
<tr>
<td>dB</td>
<td>decibels (relative unit of measure, usually for noise)</td>
</tr>
<tr>
<td>DCA</td>
<td>Defense Communications Agency</td>
</tr>
<tr>
<td>DIRNSA</td>
<td>Director of NSA</td>
</tr>
<tr>
<td>DOJ</td>
<td>Department of Justice</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processing</td>
</tr>
<tr>
<td>EDAC</td>
<td>Error Detection and Correction</td>
</tr>
<tr>
<td>FM</td>
<td>Frequency Modulation</td>
</tr>
<tr>
<td>FS</td>
<td>Federal Standard</td>
</tr>
<tr>
<td>FSK</td>
<td>Frequency Shift Keying</td>
</tr>
<tr>
<td>FTSC</td>
<td>Federal Telecommunications Standards Council</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>ICASSP</td>
<td>International Conference on Acoustics, Speech and Signal Processing</td>
</tr>
<tr>
<td>ICEP</td>
<td>INMARSAT Codec Evaluation Proposal</td>
</tr>
<tr>
<td>INMARSAT</td>
<td>International Maritime Satellite</td>
</tr>
<tr>
<td>LMR</td>
<td>Land Mobile Radio</td>
</tr>
<tr>
<td>lsb</td>
<td>least significant bit</td>
</tr>
<tr>
<td>MI</td>
<td>Message Indicator</td>
</tr>
<tr>
<td>msh</td>
<td>most significant bit</td>
</tr>
<tr>
<td>NCS</td>
<td>National Communications System</td>
</tr>
<tr>
<td>NSA</td>
<td>National Security Agency</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>NSEP</td>
<td>National Security Emergency Plan</td>
</tr>
<tr>
<td>PDF</td>
<td>Probability Distribution Function</td>
</tr>
<tr>
<td>OOS</td>
<td>Out-of-Synchronization</td>
</tr>
<tr>
<td>QDPSK</td>
<td>Quadrature Differential Phase Shift Keying</td>
</tr>
<tr>
<td>QRC</td>
<td>Quadratic Residue Code</td>
</tr>
<tr>
<td>PN</td>
<td>Pseudorandom Noise</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>S/N</td>
<td>Signal to Noise Ratio (usually measured in dB)</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

The author wishes to express her deepest appreciation and gratitude to the following people:

- Doug Rahikka, without whose help this thesis would not have been possible. He supplied the majority of the information and knowledge for this thesis as well as many hours in consultation.

- Captain Tom Schwendtner -- for his enthusiasm, advice and encouragement which enabled the achievement of this goal!

- Professor Loomis, for his expertise and advice.

- Bob Limes and Elaine Kodres, for software support with a smile.

- Mr. Doug Stoneburger, Mr. Al Crawley, Mr. Dale Learn, and Bob Feinichel, whose support I could not have done without.

- My family and friends who believed this achievement was possible and prayed for this miracle that was realized only by God’s Grace.
I. INTRODUCTION

A. BACKGROUND

1. Why Land Mobile Radio Standards are Needed

Interoperability, connectivity and security of the U.S. land mobile radio (LMR) communications systems are important objectives at all levels - federal, state, and local. In many ways the U.S. falls short in meeting these communications objectives. "This situation is even more serious when viewed in the National Security Emergency Preparedness (NSEP) context" [Ref. 1]. During such an emergency, private users of a land mobile public safety system may need to communicate with federal defense and law enforcement agencies to effectively execute emergency operations. This is not currently possible.

Within the defense community, interoperability between various services is a longstanding problem. An example that illustrates the extent of this problem is provided by the communication problems encountered during the Grenada conflict. Many rumors have circulated about a soldier who used his AT&T card to call back to the continental United States by phone, since he could not communicate with soldiers on the other side of the island due to lack of radio interoperability.¹ For all future joint military conflicts, interoperability between the services is a necessity.

The absence of a federal communications standard for the LMR, until most recently, has encouraged the proliferation of a diverse group of technically incompatible LMRs. The fact that the commercial market has produced a wide variety of state-of-the-art equipment is good in one sense -- there is much to choose from and the competition in the commercial market place keeps the cost low. However, each manufacturer produces his own "best" product with no standards or guidelines to follow. Modulation schemes, for example, may be different. This makes communication possible only with someone who has pur-

¹ This story has never been substantiated.
chased a radio from the same vendor. But if the overall objective is to promote LMR interoperability across agency and service boundaries, then more thorough standardization is necessary.


There is presently a LMR standard in place -- Federal Standard (FS) 1023. Because of its late adoption (September 1989), it did not have an impact on existing defense or civilian LMR systems; many incompatible LMRs were already fielded. FS 1023’s near-term objective was to prevent further proliferation of incompatible systems until a new standard for future upgraded systems could be agreed upon.

FS 1023 will be obsolete in the near future, in part because “...increasing demands on the fixed spectrum availability along with overseas evolution towards narrower channels have caused the need for efficient spectrum use.” [Ref. 2] Today, new technological advancements enable more efficient use of the frequency spectrum. FS 1023 is based on 25 kHz allocated channel spacing; the proposed standard, FS 1024, will require a narrower allocation of frequency channel spacing -- 12.5 or 6.25 kHz.

LMR interoperability, through the adoption of FS 1024, is an achievable mid-term goal. Such an LMR standard would help to alleviate both interoperability and spectral congestion concerns.

Because of the interest and justified need for a future federal standard, the National Communications System (NCS) office of Technology and Standards, the National Security Agency (NSA) and the Land Mobile Radio (LMR) subcommittee of the Federal Telecommunication Standards Committee (FTSC) are working together to develop Federal Standard 1024 to meet the future needs of all LMR users -- federal (civil and defense), state, and local. The official draft FS 1024 is expected to be released in Spring 1990 for 90 day industry coordination and comment [Ref. 3]. Other correspondents involved include the U.S. Department of Justice (DOJ), the Assistant Secretary of Defense for C3I, and the General Electric Company [Refs. 1, 4, 5, 6, 7, 8, and 9].
3. Advantages and Disadvantages of FS 1024

There are three clear advantages with the adoption of FS 1024. First, full and open competition between commercial LMR vendors will become healthier. Clearly the company with the best product for the money will win a competitive procurement. Second, a healthy market will produce a greater availability of LMR products. Third, it is likely that the standard will extend beyond the bounds of the intended commercial LMR market to the tactical defense radio market. This "status quo" effect, driven by the adoption of FS 1024, would have a positive influence on intra-service communications compatibility and interoperability.

Most often, each service provides the communications for its own units. There is a wide variety of tactical radios fielded for dozens of unique applications. It is more of an exception when these radios are able to interoperate. FS 1024 may be seen, then, as a first step in enhancing cross-service interoperability requirements for tactical (land mobile) units.

The main disadvantage of FS 1024 is that standardized radios would not be backwards compatible with existing radios that comply with FS 1023. An upgrading of all existing systems, a high-cost and highly unlikely proposition, would be necessary before full interoperability could be obtained. Over time this disadvantage will be lessened since an in-place Federal Standard for new LMR systems will ultimately drive the replacement of existing systems.

B. DESCRIPTION OF LAND MOBILE RADIO FS 1024

The narrowband digital LMR standard includes three criteria to ensure interoperability. These three criteria are:

- voice coding (digitization) technique,
- the radio frequency (RF) modulation technique,
- the cryptographic algorithm and cryptographic synchronization.
The voice coding technique refers to another federal standard - FS 1016. This standard specifies CELP (codebook excited linear predictive) coding as the requirement. CELP coding will be implemented with a 8 KHz sampling rate. The algorithm itself contains three functions: short term spectral prediction, long delay adaptive codebook “pitch” searches, and innovative stochastic codebook search.

The second criterion, RF modulation, is still under consideration. Possibilities include 4-ary frequency shift keying (FSK), tamed frequency modulation (FM), quadrature differential phase shift keying (QDPSK) and $\pi/4$ shift QDPSK. Final selection will be based on spectral efficiency (minimizing adjacent channel interference) and power efficiency.

C. SCOPE AND GOAL OF THESIS STUDY

The third criterion, cryptographic algorithm and synchronization is the portion of FS 1024 on which this thesis will concentrate. The focus is on the error detection and correction (EDAC) scheme for the message indicator (MI) portion of the synchronization bits, within the allocated transmission format.

The goal of this study is to determine the most suitable EDAC code, among four - Hamming, Quadratic Residue Code (QRC), Golay hard decision and Golay soft decision - for cryptographically securable LMRs. Factors affecting the findings are also discussed. Analyses and findings are based on a computer simulation. The thesis only addresses block codes. Convolutional codes were eliminated from consideration by DIRNSA based on a study by NASA [Ref. 11]. Longer processing delays - characteristic of convolutional codes - are considered unacceptable for use in LMR, where a more immediate response is required.

2 FS 1016 is proceeding through the FTSC approval process [Ref. 10: p. 3] and is expected to be approved by the Summer of 1990.

3 In an analysis performed by NSA, coherent and mainly noncoherent $\pi/4$ shift QDPSK were used. [Ref. 2]
D. ORGANIZATION OF THESIS

Chapter II discusses the transmission frame (bit) structure and proposed interleaving scheme. Interleaving and EDAC serve to mitigate noise induced errors.

Chapter III discusses and describes the EDAC codes considered.

Chapter IV describes the simulation and tests performed.

Chapter V presents the test results.

Chapter VI provides conclusions and recommendations for further study.

The appendix contains the FORTRAN code for the computer simulations used (Appendix A), the selected output data of the simulation runs (Appendix B), and the codeword interleave tables for each code type (Appendix C). Golay hard and soft decision use the same interleave table, therefore, there are only three algorithm 'types' out of the four codes tested.

E. THESIS COLLABORATION AND SUPPORT

This thesis was performed in consonance with DIRNSA R556 as an aid in their study and analysis of the FS 1024 issues for DIRNSA V2 (secure voice acquisition) and the National Communications System (NCS). Also, GTE Government Systems Corporation - Electronic Defense Communications Division in Waltham, Massachusetts is currently evaluating portions of the new FS 1024 for DIRNSA under contract number MSA904-90-C-6014.
II. TRANSMISSION FRAME STRUCTURE AND INTERLEAVING

A. FS 1024 SUPERFRAME AND FRAME

Figure 1 on page 7 shows the bit allocation for the FS 1024 data rate of 8000 bits/s. Table 1 on page 8 summarizes bit allocation and the respective bit rate for each superframe. For each second, 2400 are Error Detection and Correction (EDAC, or “parity”) for the information bits, 4800 of the bits are information bits (CELP processed) and the remaining 800 bits are used for overhead processing. Bits allocated for overhead processing include bits for:

- framing
- mode control
- system control
- cryptographic synchronization (i.e., message indicator (MI) bits)
Figure 1. Data Rate Allocation (Source: GTE Government Systems Corp., LMR program review, Waltham, MA)
Table 1.  SUPERFRAME BIT COUNT (420 MS)

<table>
<thead>
<tr>
<th>Description</th>
<th>SF Bits</th>
<th>s Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voice EDAC (parity)</td>
<td>1008</td>
<td>2400.00</td>
</tr>
<tr>
<td>Voice (CELP bits)</td>
<td>2016</td>
<td>4800.00</td>
</tr>
<tr>
<td>Framing</td>
<td>48</td>
<td>114.28</td>
</tr>
<tr>
<td>Mode Control (MC)</td>
<td>4</td>
<td>9.52</td>
</tr>
<tr>
<td>MC EDAC</td>
<td>10</td>
<td>23.80</td>
</tr>
<tr>
<td>Msg Indicator (MI)</td>
<td>72</td>
<td>171.42</td>
</tr>
<tr>
<td>MI EDAC</td>
<td>72</td>
<td>171.42</td>
</tr>
<tr>
<td>Reserved bits</td>
<td>6</td>
<td>14.29</td>
</tr>
<tr>
<td>System Control</td>
<td>124</td>
<td>295.24</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>3360</strong></td>
<td><strong>8000</strong></td>
</tr>
</tbody>
</table>

Adapted from GTE - Government Systems Corporation

For the recommended superframe transmission and interleaving format, there are 8 modes of operation defined by 4 bits in the 800-bit overhead control field: four are for the encrypted mode and four for the plain text mode. Only the encrypted voice mode, designated as 0100 in the mode control block, will be addressed in this thesis since the message indicator (MI) bits are not used in the unencrypted or plain text frame structure.

Figure 2 on page 9 shows the proposed frame structure for both the encrypted and the plain text mode. The shaded portion, the information superframes, is the only section of the message considered. The plain text mode does not incorporate EDAC or MI bits; thus, plain text superframes are half the duration.

Figure 3 on page 10 provides a one second snapshot of 8000 bits. Each superframe contains fourteen 30 msec frames. A total of 72 MI bits plus 72 MI parity bits will be interleaved throughout these 9 of these 14 frames, in a process
explained below. Three levels are depicted in Figure 3 on page 10. The top level depicts 8000 bits or 8000/3360 = 2.38 superframes. The next level shows the superframe broken down into the fourteen 30 msec frames. Finally in the lowest level, one of these 240 bit frames shows the arrangement of the 16 interleaved bits within one frame. Of the 14 frames, frames 4 through 12 (only) contain the MI bits and MI EDAC (or "parity") bits.

B. PROPOSED INTERLEAVING

The process of interleaving involves dispersing critical bits over the superframe, in regular fashion, so that there is a smaller chance of these bits being corrupted by noise bursts. Interleaving ensures, for example, that the 72 MI bits and the 72 MI parity bits are spread in a pre-determined arrangement throughout the superframe. In this way a long noise burst will not corrupt these critical portions of the message. If the bits were grouped together, even a 10 ms burst (80 bits) could destroy the usefulness of the entire MI, and prevent successful cryptographic synchronization. Instead, by using an interleaving technique, only
One Second Snapshot (8000 bits)

Information Portion of Message

Figure 3. Transmission Structure Breakdown

partial superframe 2320 bits

1 superframe = 3360 bits

partial superframe 2320 bits

290 ms

420 ms

290 ms

14 frames/superframe

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

bit position 2880/3360

interleaved frame = 240 bits

bit position 960/3360

30 ms

superframe bit positions {721 751 781 811 841 871 901 931 961} composed of MI codewords

after interleaving {731 761 791 821 851 881 911 941} (bits + parity)

16 interleaved bits/frame x 9 frames = 144 bits
a few of the critical MI bits may be corrupted; these should be detected and corrected by the interleaved MI parity bits. 4

An evaluation was performed by GTE on two proposed interleaving schemes (I and II) for the FS 1024 superframe format [Ref. 12]. The testing was performed using mobile vehicles communicating in an urban environment. Scheme II outperformed Scheme I for vehicle speeds approximately three times slower in the five LMR frequency bands (132-172MHz, 406-420MHz, 450-512MHz, 806-512MHz and 851-866MHz). 5 This is because the EDAC codewords (MI bits + parity bits) are more evenly interspersed over each frame in the Scheme II interleaver. The subsequent description and analysis address interleaver scheme II only, since FS 1024 will likely incorporated that scheme.

Figure 3 on page 10 also shows the interleaving scheme used by the EDAC computer simulation described in Chapter IV. Frames 1 through 3 are reserved for framing and mode control bits; Frames 13 and 14 contain system control bits. Every frame contains the voice - CELP processed information bits and its parity bits. The interleaving of MI bits and MI parity bits begins at the beginning of frame #4 or bit 721 and alternates skipping first 10 bits then 20 bits then 10 bits. etc. all the way through the frame #12 ending at bit 2680. Appendix C includes three codeword interleaving tables, one each for the Golay (24,12) code, the QRC (48,24) code and for the Hamming (8,4) code. The MI bits plus their parity bits (or "codewords") are broken up differently for the Golay, QRC, and Hamming codes to achieve maximum spread over frames 9 thru 12, the interleaving frames. For QRC the codewords are broken up into 4 bit chunks, for Golay the codewords are broken up into 2 bit chunks and for Hamming 1 bit chunks.

Figure 4 on page 13 illustrates frame composition for the QRC (48,24). For the QRC there are 3 codewords required (3 x 48 bits = 144 bits). Each codeword

---

4 More critical, however, is the interleaving of the MI parity bits. For if non-interleaved parity bits were corrupted by a noise burst, then MI bits also corrupted by another random noise burst would have no chance of being detected and corrected by the chosen EDAC scheme. At the very least, then, MI parity must be interleaved. Because of the random nature of noise however, both MI bits and their parity bits are both, optimally, interleaved.

5 For both schemes the lowest frequency band had the worst performance.
block pictured is a piece of one of the 3 codewords taken 4 bits at a time. All bits other than these 144 out of 3360 bits' superframe are treated as "don't care" within the analysis program. The EDAC protection of the information bits (CELP processed voice) was analyzed by NSA.R556 using Golay (hard and soft decision) code [Ref. 2].

In Chapter III, block codes and the codewords used for each code will be discussed in more detail. Most importantly, each code will be examined in terms of its ability to detect and correct errors.
Figure 4. Superframe Composition (Source: GTE Government Systems Corp., LMR program review, Waltham, MA)
III. EDAC CODES EXAMINED

A. INTRODUCTION

All of the codes examined in this thesis -- Golay(24,12), QRC(48,24), and Hamming(8,4) are either in the class of (n,k) block repetition codes or (n,k) cyclic block codes. For n total bits/block and k information bits/block, the number of parity bits equals (n-k), and code rate R = k/n. All of the codes have a code rate R = 1/2, which lends itself to easier bit manipulation.

For block codes, a generating polynomial, \( g(x) \), is used to determine the parity checks performed. The number of parity bits determine the number of checks performed. This parity check group is formed into a check matrix or \( H \) matrix for ease of manipulation.

The n-bit blocks (or "codewords"), formed by the product of the encoder (the parity bits) and information bits, becomes the transmit vector, \( V(x) \). The received vector, before decoding, is designated \( R(x) \).

Prior to decoding, a syndrome is used as a binary number block by the code algorithm to mark errors and the position of the errors for correction. If the syndrome equals zero, then there are no errors. The decoding and correction of the bits are algorithm dependent. Several different decoding schemes may be possible for each code.\(^6\)

The number of possible errors corrected for a code is determined not only by the codeword size (n) and the information stream length (k), but by the minimum Hamming distance, \( d_{mn} \). This distance is the minimum number of bits that differ among all pairs of codewords in the code set [Ref. 16: p. 25]. The code can correct \( 1, 2(d_{mn}) \) number of errors if \( d_{mn} \) is even and \( 1/2(d_{mn} - 1) \) errors if \( d_{mn} \) is odd.

B. GOLAY CODES

The perfect cyclic Golay (23,12) code is a triple-error correcting code, and can be implemented in software. Typically, one extra bit is added to parity yielding

\(^6\) For derivations and more study on linear block codes refer to [Refs. 13, 14, and 15].
the Golay (24,12) code. This is done to maintain code rate = 1/2 and ease of algorithmic manipulation. The generating polynomial used is:

\[ g(x) = x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1 \]

The Golay (24,12) code can be improved by utilizing analog information associated with the channel bits. In this way the bit error rate in white Gaussian noise can be improved about 2 dB. [Ref. 17] This “soft” decision-making uses the value of the previous baud to calculate confidence intervals. These confidence values are \textit{real} values rather than actual baud values. This concept is illustrated by the constellation. Figure 5 on page 16 [Ref. 2].

The X and Y axes of Figure 5 on page 16 represent the least significant bit (lsb) and the most significant bit (msb) decision boundaries of the two baud being compared. The dot represents the unquantized phase difference between present baud and previous baud and the circle represents the magnitude at the present baud relative to the low-pass filter receive baud magnitude. The confidence for each sb is then the phase difference from the dot to its corresponding nearest boundary. Fades are flagged by scaling the amplitudes of the confidence values.

Sixteen iterations (2^4 bit patterns) are performed to search for and invert the bits designated by the Golay decoder as the lowest four confidence bits in the codeword. The errors are counted for each iteration and the bit pattern with the lowest cumulative confidence is selected as valid.

For both Golay decisions, there are 6 codewords (24 bits codeword x 6 codewords superframe = 144 bits) generated for every superframe for the simulation.

C. HAMMING CODES

The Hamming (7,4) codes is a single error correcting code where all combinations of two errors are detected; only one error may be both detected and corrected. This code was the first error detection and correction code discovered, in

\footnote{Bauds 10, 00, 01, and 11 correspond to phase changes 45, 135, -135, and -45 degrees, respectively.}
Figure 5. Confidence Constellation [Ref. 2]

1949 by R.W. Hamming [Ref. 18: p. 13-29]. The modified (8,4) Hamming code is used for this analysis, where, 1 parity bit is added to maintain code rate \( R = 1/2 \).

For the simulation, there are 18 Hamming codewords (8 bits/codeword \( \times \) 18 codewords = 144 bits/superframe) generated for every superframe.

D. QUADRATIC RESIDUE CODE (QRC)

Quadratic residues are used to specify the roots of the code generator polynomial [Ref. 19: pp. 92,93]. They are defined by the numbers,

\[ 1^2, 2^2, 3^2, \ldots, \left( \frac{p-1}{2} \right)^2 \]

which are mod \( p \) reduced, where \( p \) is an odd prime number. Nonresidues are the numbers in the quadratic group that are not included by the residues. For ex-
ample, if \( p = 19 \), this would include all the mod 19 residues for \( 1^2, 2^2, 3^2, \ldots, 9^2 \). The following table, Table 2 on page 17, shows these values.

**Table 2. QUADRATIC RESIDUES FOR \( p = 19 \)**

<table>
<thead>
<tr>
<th>residues</th>
<th>non-resid</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1^2 = 1 )</td>
<td>1</td>
</tr>
<tr>
<td>( 2^2 = 4 )</td>
<td>4</td>
</tr>
<tr>
<td>( 3^2 = 9 )</td>
<td>9</td>
</tr>
<tr>
<td>( 4^2 = 16 )</td>
<td>16</td>
</tr>
<tr>
<td>( 5^2 = 25 )</td>
<td>6</td>
</tr>
<tr>
<td>( 6^2 = 36 )</td>
<td>17</td>
</tr>
<tr>
<td>( 7^2 = 49 )</td>
<td>11</td>
</tr>
<tr>
<td>( 8^2 = 64 )</td>
<td>7</td>
</tr>
<tr>
<td>( 9^2 = 81 )</td>
<td>5</td>
</tr>
</tbody>
</table>

The QRC investigated by GTE is the (48.24) code, with \( p = 47 \). This code is capable of detecting and correcting 100% of 5 (or fewer) bit errors and 62% of 6-bit errors. The probability of correcting 6-bit errors is calculated with combinatorial logic as follows.

\[(47.24) + 1 \text{ parity bit} = (48,24)\]

The number of 5 or less correctable errors is:

\[C(47,5) + C(47,4) + C(47,3) + C(47,2) + C(47,1) = 1,729,647\]

Also, the codeword with one error must be accounted for, \( C(47,0) = 1 \). Since there are \( 2^{23} \) possible parity vector values (8,388,608).

\[8,388,608 - 1,729,647 - 1 = 6,658,900\]
is the number of possible parity combinations remaining to map into 6 or more error combinations. Next, the number of 6-bit errors possible within the 48 bit codeword is:

\[ C(47,6) = 10,737,573. \]

Therefore,

\[ 6,658,960 \div 10,737,573 = .6202 \]

This implies that the QRC will correct the sixth error in a 48 bit codeword 62.02% of the time.

There are 3 QRC codewords (48 bits/codeword x 3 codewords = 144 bits/superframe) generated for every superframe in the simulation.

The implementation of these EDAC schemes into a transmission simulation model will be discussed in Chapter IV. Also, the testing design approach and channel simulation models are introduced the next chapter.
IV. TESTS AND SIMULATION

A. DESCRIPTION OF THE PROGRAMS PROVIDED

1. Introduction

The program used in this analysis was provided by DIRNSA/R556, and was previously used by DIRNSA to evaluate the use of Golay (hard and soft decision) coding for the protection of the CELP information bits. A separate subroutine of the Hamming code was also provided, for integration into the main program. Program subroutines neatly divided most of the program chores (i.e. bit generation, encoding, modulation, decoding, bit error count, etc.).

2. Modifications

The program code was loaded and run on a VAX 11/785 mainframe in the ECE Department at the Naval Postgraduate School after conversion from Sun III FORTRAN to Berkley FORTRAN 77. The program was further modified to incorporate the Hamming code simulation and a bit error counter for the QRC. The encode decode functions of the QRC are not included in the simulation - only a counter based on the code's probability of error detection and correction within a codeword.\(^8\)

B. SIMULATION DESCRIPTION

Figure 6 outlines the simulation process in block fashion. The process begins as a pseudorandom (PN) bit stream, representing the MI bits, is produced for each superframe. Next, the 72 MI bits are encoded by the chosen EDAC, with "no code" representing the default control. The encoder generates the 72 parity bits to formulate a completed codeword of 144 bits.

The 144 bits are then hashed. This hashing process is in reality a pre-interleaving step that enhances the spread of the codeword bits throughout the

\(^8\) At this writing GTE is looking at simulating the QRC. However, the algorithmic implementation is much more complicated than even the Golay codes. Therefore, actual QRC encoding and decoding is not attempted in this analysis.
superframe. For each codeword the first bit is stacked, next the second, the third and so on. These realigned bits are then interleaved in Scheme II fashion, as described in Chapter II. To fill the rest of the 3360 bit superframe, an arbitrary PN bit stream is produced. The codeword bits are interleaved among these fill bits.

Figure 6. Simulation Block Diagram

Before modulation and transmission, the bits are changed to baud using a dibit representation as shown in Table 3 on page 21. The entire superframe is modulated and transmitted, then sent over one of three user-chosen simulation

---

9 This process is not reflected in the codeword interleaving table in Appendix C.
channels. The transmitted bits, now with simulated errors, are received and de-modulated, deinterleaved, stripped of the 3216 non-applicable bits, de-hashed, decoded and checked for errors.

<table>
<thead>
<tr>
<th>Table 3. BIT/BAUD EQUIVALENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>dibit</td>
</tr>
<tr>
<td>00</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
</tbody>
</table>

Figure 7 illustrates the simulation design approach. Each section of the matrix represents one simulation run, for a total of 80 runs (5 codes x (1 INMARSAT CH + 8 AWGN CH + 7 constant burst CH)). A simulation run consists of 200 consecutive superframes. In all cases, the first superframe was expended for synchronization purposes. Thus, there are 199 x 420 ms or 83.58 seconds of established continuous information flow, which is likely much greater than an average message transmission. During the 83.58 second period, continuity of the digital signal is maintained unless bit synchronization is lost due to fade or noise.
Figure 7. Simulation Design Approach

All simulation runs record the number of errors (within the 144 EDAC bits/superframe - location of an error is unknown) out of the total number of EDAC bits processed. Also, the number of successful and failed (S/F) superframes are recorded. Even one uncorrected error results in a superframe failure.

C. CHANNEL SIMULATIONS

The "mode" and "vary" variables provided for each channel are simulation selectable variables which determine the channel simulation mode.

For the simulation model, during fading or burst mode sequence the signal was subjected to -24 dB S/N and the signal strength divided by 200. This was
injected to ensure a 50% Bit Error Rate (BER) during burst error as required by the INMARSAT model. During non-burst error intervals, the S/N could be at any level. This parameter is user specified. For the two burst channel simulations, the S/N was set equal to 99.0 dB to ensure a clean signal. Therefore, all of the errors are a result of the signal fade. In the Gaussian channel the S/N parameter was varied from 0 to 7 dB. For the Gaussian channel, simulation errors were a result of poor S/N levels only.

1. INMARSAT Burst (mode = 1, vary = 0)

For the random burst noise (signal fade) channel simulation, the “simplified Land Mobile Radio channel model for IMARSAT Codes Evaluation Proposal (ICEP)” was used [Ref. 20: pp. 9,10]. The model is described as follows:

“The model is derived from realistic empirical propagation measurements such that performance of voice codecs can be tested in the laboratory with replicable situations. 90% is chosen as the link availability figure in the simplified model. The model is basically an ON-OFF model in which fading is in a binary state. The ON-state should correspond to no transmission whereas the OFF-state corresponds to 50% BER.”

The discrete probability density function (PDF) for the permissible noise burst widths are shown in Table 4 on page 24. The distribution of the burst widths for the INMARSAT channel is base on a random process. Therefore, the number of bursts for each burst width varied. For all but 200 msec burst widths, the actual number of bursts for each burst width were not significantly different from that predicted by the PDF.
Table 4. SIMPLIFIED FADE DURATION

<table>
<thead>
<tr>
<th>Fade (ms)</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>10ms</td>
<td>0.8</td>
</tr>
<tr>
<td>20ms</td>
<td>0.1</td>
</tr>
<tr>
<td>40ms</td>
<td>0.05</td>
</tr>
<tr>
<td>100ms</td>
<td>0.04</td>
</tr>
<tr>
<td>200ms</td>
<td>0.01</td>
</tr>
</tbody>
</table>

For the burst-noise channel simulation, one 200 superframe run was performed for each of the four codes and one run with no coding. An example of 3-second Golay-encoded signals, transmitted over an INMARSAT burst-noise channel, is shown in Figure 8 on page 25.

2. AWGN Channel (mode = 0, vary = 0)

For the additive white Gaussian noise (AWGN) channel simulation, signal-to-noise ratio (S/N) was varied from 0 to 7.0 decibels (dB) in 1.0 dB increments. Thus forty simulation runs (5 code modes x 8 S/N levels) were performed.

3. Constant Burst Width (mode = 1, vary = 1)

For the constant burst width channel simulation, a constant fade depth (or burst width) was inserted at random time intervals. Seven discrete fade widths -- 5, 10, 20, 40, 100, 200, and 400 milliseconds -- were run for each code. Thus, thirty-five simulation runs (5 codes x 7 burst widths) were performed, each having 200 superframes. The 5 and the 400 msec burst widths, not included in the INMARSAT model, were added to observe the limits of the model and the performance of the codes.
Figure 8. INMARSAT Channel Performance [Ref. 2]

Figure 9 shows constant amplitude signals subjected to varying fade widths. One second translates to 2.38 superframes.

Using the same outline as Chapter IV, the test results are presented in Chapter V by channel simulation type. The BER results and S/F results for each simulated code over each of the three channels is displayed and compared.
<table>
<thead>
<tr>
<th>VARYING FADE WIDTH SIGNALS (1 SEC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>burst width, ms</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

Figure 9. Varying Fade Width Signals [Ref. 2]
V. COMPARISON AND ANALYSIS OF TESTS

For each of the three simulated noise channels, the codes are rated in terms of BER performance and in terms of number or percentage of successful (or "error-free") superframe transmissions. BER results are expressed as the BER exponent.\(^{10}\) This number is calculated by taking the \(\log_{10}\) of the number of errors counted divided by the number of bits transmitted.

The code abbreviations used in data tables are:
- \(g(s)\) - soft decision Golay (24,12)
- \(g(h)\) - hard decision Golay (24,12)
- \(\text{none}\) - no channel coding
- \(\text{ham}\) - Hamming (8,4)
- \(\text{QRC}\) - quadratic residue code (48,24)

A. INMARSAT BURST CHANNEL

1. BER Results

Table 5 on page 27, reflects BER results as the log of the quotient (number of errors counted divided by the number of bits transmitted). All BERs are greater than \(10^{-1.2}\) for this simulation, which implies greater than 1 error out of 100 bits. These performance values are intuitively reasonable for LMR operations in an urban environment.

<table>
<thead>
<tr>
<th></th>
<th>(g(s))</th>
<th>(g(h))</th>
<th>none</th>
<th>ham</th>
<th>QRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BER exp</td>
<td>-1.5003</td>
<td>-1.4342</td>
<td>-1.3363</td>
<td>-1.5243</td>
<td>-1.8153</td>
</tr>
</tbody>
</table>

Also, the difference between the hard and soft Golay is only about 0.5\%, while the QRC outperforms the soft Golay by more than 1.6\%. This may be a reflection of the number of "long" bursts. For the pseudo-random process, the

\(^{10}\) The convention in [Ref. 2] was followed.
g(s) suffered eight 200 msec fades while the QRC was only forced to process 2. If these additional bursts happened to span the critical MI bits in the g(s) codewords, the BER would have increased. Looking at the INMARSAT PDF, Table 4 page 24, the 200 msec bursts only have a 1% probability of occurrence. The difference between the number of 200 msec fades for the g(s) and the QRC runs is therefore, insignificant.

Figure 10 on page 29, derived from table 5, shows the BER performance advantage that the QRC has over the other codes for this simulation. The QRC detected and corrected 1.5% more errors than the g(s).

2. Run Success/Failure Results

Table 6 on page 28 reports the number of successful runs in the simulated INMARSAT channel, for each code. While BER performance favors the QRC in an INMARSAT noise channel, the QRC only ran an additional two successful superframes out of 199. That is less than a 1% performance difference. Nevertheless, the QRC and g(s) are noticeably superior than the other codes and the uncoded channel.

<table>
<thead>
<tr>
<th>Table 6. INMARSAT BURST - #SUCCESSES/#FAILURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>g(s)</td>
</tr>
<tr>
<td>#successes</td>
</tr>
<tr>
<td>#failures</td>
</tr>
<tr>
<td>#total runs</td>
</tr>
<tr>
<td>% successful</td>
</tr>
</tbody>
</table>

11 This simulation did not allow the control of the number of bursts of each width. Therefore, it was impossible to inject the codes with the same error occurrences.

12 If the number of successes and the number of failures do not add up to 199, the runs not accounted for are out-of-synchronization (OOS) superframes. If there are too many errors the transmission falls OOS and retries until a success or a failure is achieved. In any of the channel simulations the OOS runs are not counted toward the total number runs.
Figure 10. INMARSAT BER Performance
3. Additional Tests

Further testing was performed to check the validity of the INMARSAT simulation results. An additional 1000 superframes were run for both the g(s) and the QRC. Again, the QRC outperformed the g(s) in BER by .62%. The QRC also provided 2% more successful superframe runs than the QRC.

There seems to be a trend, in that the performance difference between the two codes appears to narrow with more superframe runs. This hypothesis could be tested with even more superframe runs. Nevertheless, the BER difference of less than 1% leads to the conclusion that there is no significant difference between the INMARSAT channel performance of the two codes -- g(s) and QRC.

B. GAUSSIAN NOISE CHANNEL

1. BER Results

Table 7 on page 31, tabulates BER exponent for the 40, 200 superframe runs performed in the simulated Gaussian noise channel. Figure 11 on page 32 is derived from Table 7, and shows that g(s) outperforms QRC at SN levels greater than 5 dB. The figure also shows that the g(h) outperforms the QRC at SN levels greater than 6 dB. At SN levels less than or equal to 5 dB, the g(s) offers a slight improvement over that of the QRC and the g(h).

---

13 No errors were counted at 7 dB for g(s) out of codeword bits; the BER exponent of -4.0 was used as a graph limit.

14 For the Hamming code, extra data points were taken; they are reflected in the graph only and not the table.
Table 7. BER AS A FUNCTION OF GAUSSIAN NOISE LEVEL, ALL CODES

<table>
<thead>
<tr>
<th></th>
<th>g(s)</th>
<th>g(h)</th>
<th>none</th>
<th>ham</th>
<th>QRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-0.6055</td>
<td>-0.5638</td>
<td>-0.6421</td>
<td>-0.6108</td>
<td>-0.654</td>
</tr>
<tr>
<td>1</td>
<td>-0.7447</td>
<td>-0.6615</td>
<td>-0.719</td>
<td>-0.7072</td>
<td>-0.7525</td>
</tr>
<tr>
<td>2</td>
<td>-0.9547</td>
<td>-0.8125</td>
<td>-0.8153</td>
<td>-0.829</td>
<td>-0.9014</td>
</tr>
<tr>
<td>3</td>
<td>-1.2388</td>
<td>-1.0526</td>
<td>-0.9281</td>
<td>-1.0036</td>
<td>-1.1537</td>
</tr>
<tr>
<td>4</td>
<td>-1.6126</td>
<td>-1.3487</td>
<td>-1.0605</td>
<td>-1.2069</td>
<td>-1.556</td>
</tr>
<tr>
<td>5</td>
<td>-2.2676</td>
<td>-1.8297</td>
<td>-1.2269</td>
<td>-1.4724</td>
<td>-2.1586</td>
</tr>
<tr>
<td>6</td>
<td>-3.2343</td>
<td>-2.8761</td>
<td>-1.4056</td>
<td>-1.7447</td>
<td>-2.6198</td>
</tr>
<tr>
<td>7</td>
<td>-4.0</td>
<td>-3.1549</td>
<td>-1.6234</td>
<td>-2.1707</td>
<td>-2.8339</td>
</tr>
</tbody>
</table>

Figure 12 on page 33 is another representation of the same data. Code performance is compared at each $S_N$ level measured. For each code, as the $S_N$ increases the BER decreases. There are no surprising trends for any code. These two factors -- $S_N$ and BER -- are inversely proportional and the comparison chart best illustrates this response. Also, the difference between the codes response at various $S_N$ levels is more easily observed on this comparison chart than on the previous line graph.

2. Run Success/Failure Results

Table 8 on page 34 displays the number and the percent of successful runs with the OOS runs excluded from the total. These calculations were made using the total number of successful runs over the total number of runs performed. The g(s) code outperforms other codes in the gaussian environment, especially at lower $S_N$, which are more realistic in an LMR environment.
Figure 11. Gaussian BER Performance (line graph)
Figure 12. Gaussian BER Performance (bar chart)
Table 8. #SUCCESSES/%SUCCESSFUL RUNS VS. GAUSSIAN NOISE LEVEL, ALL CODES

<table>
<thead>
<tr>
<th>dB</th>
<th>g(s) # %</th>
<th>g(h) # %</th>
<th>none # %</th>
<th>ham # %</th>
<th>QRC # %</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>2.1</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>19.96</td>
<td>31.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>62.315</td>
<td>26.132</td>
<td>0.0</td>
<td>2.13</td>
<td>35.176</td>
</tr>
<tr>
<td>4</td>
<td>122.613</td>
<td>72.369</td>
<td>1.0</td>
<td>7.36</td>
<td>107.538</td>
</tr>
<tr>
<td>5</td>
<td>175.879</td>
<td>140.710</td>
<td>21.0</td>
<td>41.0</td>
<td>177.889</td>
</tr>
<tr>
<td>6</td>
<td>194.975</td>
<td>186.935</td>
<td>7.35</td>
<td>92.462</td>
<td>194.975</td>
</tr>
<tr>
<td>7</td>
<td>197.990</td>
<td>194.975</td>
<td>31.156</td>
<td>149.749</td>
<td>198.995</td>
</tr>
<tr>
<td>totals</td>
<td>48.6%</td>
<td></td>
<td></td>
<td></td>
<td>45.2%</td>
</tr>
</tbody>
</table>

The line graph, Figure 13 on page 35 compares the percent success of each code in the Gaussian noise channel. For this figure of merit, the g(s) code outperforms both the g(h) and QRC, especially at S/N levels between 2-5 dB. The QRC outperforms g(h) at S/N levels between 2-7 dB.

The stacked bar comparison graph, Figure 14 on page 36 compares the overall success of each code, combining all S/N level runs. For total number of runs, g(s) out performs QRC by more than 50 successes.

C. CONSTANT BURST WIDTH CHANNEL

1. BER Results

Table 9 on page 37 represents BER exponents for 35, 200 superframe runs. Figure 15 on page 38 shows the BER performance recorded in table 9, with the area of critical code performance highlighted on the graph. The g(s)

---

\[ P(10 \text{ msec burst}) = 0.80; \]
\[ P(10-40 \text{ msec burst}) = 0.95. \]

---

From Table 4 page 21), INMARSAT Noise Burst Width: P(10 msec burst) = 0.80; P(10-40 msec burst) = 0.95.
Figure 13. #Successes/%Successful for AWGN channel (line graph)
Figure 14. \#successes/199 runs for AWGN channel (bar chart)
code outperforms the QRC at burst widths from 10-40 msec; the advantage diminishes at burst widths greater than 40 msecs.

Table 9. BER AS A FUNCTION OF CONSTANT BURST WIDTHS, ALL CODES

<table>
<thead>
<tr>
<th>burst widths</th>
<th>g(s)</th>
<th>g(h)</th>
<th>none</th>
<th>ham</th>
<th>QRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>-4.</td>
<td>-2.6021</td>
<td>-1.8297</td>
<td>-2.7375</td>
<td>-2.8447</td>
</tr>
<tr>
<td>10</td>
<td>-3.1759</td>
<td>-2.4461</td>
<td>-1.5331</td>
<td>-2.0706</td>
<td>-2.6716</td>
</tr>
<tr>
<td>20</td>
<td>-2.0301</td>
<td>-1.9281</td>
<td>-1.2612</td>
<td>-1.5567</td>
<td>-1.9431</td>
</tr>
<tr>
<td>40</td>
<td>-1.3516</td>
<td>-1.0516</td>
<td>-1.0419</td>
<td>-1.0956</td>
<td>-1.2048</td>
</tr>
<tr>
<td>100</td>
<td>-0.7595</td>
<td>-0.6655</td>
<td>-0.7721</td>
<td>-0.7022</td>
<td>-0.7404</td>
</tr>
<tr>
<td>200</td>
<td>-0.585</td>
<td>-0.5229</td>
<td>-0.5544</td>
<td>-0.5089</td>
<td>-0.5786</td>
</tr>
<tr>
<td>400</td>
<td>-0.4789</td>
<td>-0.4815</td>
<td>-0.5086</td>
<td>-0.4619</td>
<td>-0.5396</td>
</tr>
</tbody>
</table>

Figure 16 on page 39 also displays the effect that the varying burst widths have on each code. At the 20 msec burst width all three of the codes -- g(s), QRC, and g(h) -- begin to display a similar performance. For a 100 msec burst width condition, all codes perform equally poorly. Code response trends -- a consistent increase in the BER exponent as the burst width increases -- are as expected.

2. Run Success/Failure Results

Table 10 on page 40 displays the number and percent successful runs as a function of constant burst widths, for all codes tested. At narrower levels of burst width, the g(s) code outperforms the other coded or uncoded channels. The advantage becomes less significant as the width of noise burst exceeds 20 msecs. Again, the g(s) has the overall advantage of 4.9% more successful runs.
Figure 15. Constant Burst Width Performance (line graph)
Figure 16. Constant Burst Width Performance (bar chart)
Table 10. #SUCCESSES/%SUCCESSFUL RUNS VS CONSTANT BURST WIDTHS, ALL CODES

<table>
<thead>
<tr>
<th>burst widths</th>
<th>g(s) #,%</th>
<th>g(h) #,%</th>
<th>none #,%</th>
<th>ham #,%</th>
<th>QRC #,%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>197/99.5</td>
<td>196/99.5</td>
<td>98/49.2</td>
<td>187/94.0</td>
<td>198/100</td>
</tr>
<tr>
<td>10</td>
<td>197/99.0</td>
<td>188/94.5</td>
<td>93/46.7</td>
<td>152/76.4</td>
<td>195/98.0</td>
</tr>
<tr>
<td>20</td>
<td>178/89.4</td>
<td>158/79.4</td>
<td>85/42.7</td>
<td>118/59.3</td>
<td>166/83.4</td>
</tr>
<tr>
<td>40</td>
<td>139/69.8</td>
<td>97/48.7</td>
<td>74/37.6</td>
<td>77/38.7</td>
<td>107/53.8</td>
</tr>
<tr>
<td>100</td>
<td>66/33.2</td>
<td>48/24.1</td>
<td>62/31.5</td>
<td>48/24.1</td>
<td>41/20.6</td>
</tr>
<tr>
<td>200</td>
<td>39/19.6</td>
<td>19/12.5</td>
<td>39/20.0</td>
<td>26/13.1</td>
<td>26/13.1</td>
</tr>
<tr>
<td>400</td>
<td>1/5.9</td>
<td>0</td>
<td>8/30.0</td>
<td>28/14.1</td>
<td>19/11.0</td>
</tr>
</tbody>
</table>

The line graph, Figure 17 on page 41, compares the relative number of successes for each code. A more robust code with greater interleaver depth may be needed to satisfy performance objectives at these burst widths. The g(s) code performs best, but it's percent success degrades as burst widths exceed 40 msecs. It is interesting to note that the uncoded channel surpasses all codes in performance except the g(s) for burst widths of 100 and 200 msecs.

Figure 18 on page 42 also compares the percent success for each code. Again, the performance advantage of the g(s) and QR codes degrades beyond a 40 msec burst to the 100 msec burst. Figure 19 on page 43 shows the performance trends as burst widths lengthen. With the exception of the g(s) and QRC, all codes achieve less than 50% success at burst widths greater than or equal to 40 msec.

16 More analysis would needed to reach a decisive conclusion.
Figure 17. Constant Burst Width - S/F (line graph)
Figure 18. Constant Burst Width - S/F (bar chart)
Figure 19. Constant Burst Width - S/F (comparison)
D. COMPARISON OF ERROR POSSIBILITIES

Given the larger codeword, by probabilistic calculations, the QRC leads in error correction capability since there are a greater number of possible errors corrected. See Figure 20 on page 45 for an understanding of the possibility of 6 errors corrected out of 48 bits for each code. Notice that the Golay calculation considers the hard decision number of possible errors only. The soft-decision Golay code yielded a 5% better success rate for the INMARSAT channel (see Table 6 page 28). It was also observed that the number of possible errors corrected is much closer to the QRC calculation - up to seven errors can be corrected per codeword.

In Chapter VI, the preceding results will be summarized and recommendations presented.
COMBINATIONS of 6 errors/48 bits corrected

62% \times P\left(C(48,6),1\right) = 7,610,512

**QRC**

\[ P(C(24,3),2) + C(24,3)C(2,2)P(C(24,3),0) = 4,096,576 \]

**Golay**

24 bits

\[ P(C(8,1),6) + C(8,1)C(6,2)P(7,4) + \ldots + C(8,1)C(6,6)P(7,0) = 159,944 \]

**Hamming**

8 bits

48 bits

Figure 20. Possible Errors Corrected
VI. CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER STUDY

A. CONCLUSIONS

Three simulated noise channels -- INMARSAT random burst, AWGN, and constant burst -- were used to analyze four EDAC codes; Golay hard decision, Golay soft decision, Hamming and QRC. Before the study commenced, the INMARSAT random burst channel model was determined to be the most realistic type of LMR noise channel. It was believed by NSA that the soft Golay and QRC codes would be very close performance competitors. The results obtained show that the soft decision Golay slightly outperforms the QRC based on the following:

- For the Gaussian channel, the BER exponent for the g(s) is significantly less than that of the QRC at S/N of 5 dB and below. The g(s) code also had greater percentage of successful runs than the QRC for S/N levels from 2-4 dB.

- For the Constant Burst Width channel, the g(s) marginally outperforms the QRC when subjected to 10 msec burst noise. For each code, percentage of successful runs is similar at this burst width; the g(s) achieves 1% greater success. Over all burst widths, the g(s) achieves 4.9% more successful superframe runs.

- For the INMARSAT random burst channel the results are very close. The QRC outperformed the g(s) by detecting and correcting 1.5% more errors and running almost 1% more successful superframes. For the additional 1000 superframe runs, the results are even closer. Since the performance margin is so close, the conclusion is that there is no significant difference between the two codes for this simulation. More superframe runs may further support this conclusion.

The margin of performance is very close, as expected, between the soft decision Golay and the QRC. The biggest disadvantage in QRC implementation versus soft decision Golay is the difficulty of algorithmic implementation and the larger number of codeword and parity combinations to choose from. This translates into greater processing time and greater memory storage.
Based on these two considerations, soft decision Golay is recommended as the best code for FS 1024.

**B. RECOMMENDATIONS FOR FURTHER STUDY**

1. **Study Continuation**

From experience it was learned that a more concentrated level of effort should have been paid to the 10-40 msec burst width region (95% of the burst fades for the INMARSAT PDF). More superframes should be subjected to these burst widths to provide more conclusive findings.

Also, additional INMARSAT channel iterations along with a statistical analysis\(^\text{17}\) may shed light on the difference that the long burst widths may have on the performance of the g(s) versus the QRC.

2. **Implementation**

Another consideration and topic for further study is the feasibility and architecture of hardware implementation of candidate EDAC algorithms. Several Digital Signal Processor (DSP) Integrated Circuits (ICs), such as the DSP32C and the TMS320C30, are being considered for hardware implementation. Microprocessor (assembly language) code of the soft decision Golay is already under development. DIRNSA.\(^\text{18}\) Since the QRC codebook is much larger than the soft decision Golay, IC memory (itself limited by chip size) may be a limiting factor for hardware implementation of the QRC.

3. **Other Codes**

The scope of the thesis was narrowed to the four block codes; however, other possibilities should be considered in future studies. Simulation and testing of convolutional, other linear block, and concatenated codes may result in the discovery of an even better code for LMR application. A comparison of appropriate tradeoffs -- such as processing time, difficulty of algorithmic implementa-

\(^{17}\) The program must be modified to obtain the necessary statistical information, such as the bit position of the error, position of the fade, the number of errors each fade caused, etc.

\(^{18}\) This EDAC was chosen as the code to protect the digitized voice (CELP processed) information bits.
tion, and code performance -- should be weighed in order to fairly judge the best code for the M1 bits.
APPENDIX A: FORTRAN CODE

A. MAIN DRIVER

c  14*240=3360 bits per superframe (sf)
c  72+72=144 bits per mi
c  3360-144=3216 bits of fill per sf

c  3360/2=1680 bauds per sf

c*****************************************************************************
c  This program generates a superframe of 3360 bits.
c  72 bits are of interest in the detection and correction
c  of errors. Three different EDAC schemes simulated here

c  for comparison of the best scheme under various conditions.
c The three codes are Golay (24,12) using soft and hard
c decision logic, Hamming (8,4), and Quadratic Residue Code

c (QRC) (48,24). Also, a run with no coding is performed as

c  a control case.

c  The entire superframe is interleaved, transmitted over a

c  simulated AWGN channel, received, deinterleaved, and

c  finally - checked for errors. The run is considered a

c  failure if there are ANY errors.

c  Options for the simulated channel include variance of the

c  s/n during non-fade (the fade s/n was held at -24.0 dB), and

c  two burst modes. The first burst mode used the INMARSAT

c  values of varying length from 10-200ms for LMR. The second

c  burst mode allows for a constant burst length input.

c*****************************************************************************

program mainedac

*****************************************************************************

integer*4 tbaud(1680)
integer*4 fillbits(3216)
integer*4 parity(72)
integer*4 rbaud(1680)
double precision rbcnt,tbcnt,decodcnt1,decodcnt2
integer*4 modegolay,ham,parnum,qrc,sf,nummi
integer*4 index,prevoutdibit,table(16)
integer*4 tbits, bits, txbits, rbits
real xdisp, ydisp, confh, confl
real samples(3), noise, noisef, bprob, probq
integer*4 mode, iseed, vary
integer*4 testcnt, tbit(3360), rbit(3360), intlvtable(144)
integer*4 mibits(72), codeword(8), hmatrix(8), syndrometable(8)
integer*4 testcnt, paritybit
real intlvconf(3360), rcodeconf(144), rhashconf(144)
real rfillconf(3216)
real conf(24), gaus(256)
integer*4 code(144), hashcode(144), qrcbits(144), hambits(72)
integer*4 hashtable(144)
integer*4 rcode(144)
integer*4 itoterr, qrcwerr, qrcsferr, bwidth, myerror, hamerr
integer*4 iallerr, itotbitct
logical insync
integer*4 success, bits, fail, nerror, intlvrcnt, ifillrcnt, ihashrcnt
integer*4 rhashcode(144), rfillbits(3216)
integer*4 isym(2047)
integer*4 idata(24), kdata(24), ierr(5), qdata(48), hdata(4)
common /blk5/idata
common /blk7/conf, isym, kdata, ierr

*******************************************************************************DATA and TABLES*******************************************************************************

myerror = 0
data init, initfill /1,1/
data table /2,0,3,1,3,2,1,0,0,1,2,3,1,3,0,2/
if ((ham.eq.0).and.(qrc.eq.0)) then
data hashtable /
  1, 25, 49, 73, 97, 121, 2, 26, 50, 74, 98, 122,
  1, 3, 27, 51, 75, 99, 123, 4, 28, 52, 76, 100, 124,
  1, 5, 29, 53, 77, 101, 125, 6, 30, 54, 78, 102, 126,
  1, 7, 31, 55, 79, 103, 127, 8, 32, 56, 80, 104, 128,
  1, 9, 33, 57, 81, 105, 129, 10, 34, 58, 82, 106, 130,
  1, 11, 35, 59, 83, 107, 131, 12, 36, 60, 84, 108, 132,
  1, 13, 37, 60, 85, 109, 133,
  1, 14, 38, 61, 86, 110, 134,
  1, 15, 39, 62, 87, 111, 135,
  1, 16, 40, 63, 88, 112, 136,
  1, 17, 41, 64, 89, 113, 137,
  1, 18, 42, 65, 90, 114, 138,
  1, 19, 43, 66, 91, 115, 139,
  1, 20, 44, 67, 92, 116, 140,
  1, 21, 45, 68, 93, 117, 141,
1 22,46,69,94,118,142,
1 23,47,70,95,119,143,
1 24,48,71,96,120,144/
end if

if (qrc.eq.1) then
  data hashtable /
  1 1,49,97,2,50,98,3,51,99,4,52,100,
  1 5,53,101,6,54,102,7,55,103,8,56,104,
  1 9,57,105,10,58,106,11,59,107,12,60,108,
  1 13,61,109,14,62,110,15,63,111,16,64,112,
  1 17,65,113,18,66,114,19,67,115,20,68,116,
  1 21,69,117,22,70,118,23,71,119,24,72,120,
  1 25,73,121,26,74,122,27,75,123,28,76,124,
  1 29,77,125,30,78,126,31,79,127,32,80,128,
  1 33,81,129,34,82,130,35,83,131,36,84,132,
  1 37,85,133,38,86,134,39,87,135,40,88,136,
  1 41,89,137,42,90,138,43,91,139,44,92,140,
  1 45,93,141,46,94,142,47,95,143,48,96,144/
end if

if (ham.eq.1) then
  data hashtable /
  1 1,9,17,25,33,41,49,57,65,73,81,89,97,105,113,121,129,137,
  1 2,10,18,26,34,42,50,58,66,74,82,90,98,106,114,122,130,138,
  1 3,11,19,27,35,43,51,59,67,75,83,91,99,107,115,123,131,139,
  1 4,12,20,28,36,44,52,60,68,76,84,92,100,108,116,124,132,140,
  1 5,13,21,29,37,45,53,61,69,77,85,93,101,109,117,125,133,141,
  1 6,14,22,30,38,46,54,62,70,78,86,94,102,110,118,126,134,142,
  1 7,15,23,31,39,47,55,63,71,79,87,95,103,111,119,127,135,143,
  1 8,16,24,32,40,48,56,64,72,80,88,96,104,112,120,128,136,144/
end if

c read in table for golay decoder from disk file

open (3, file='isytab.dat', status='old', form='formatted')
read (3,20)isym
20 format(10(I6))
close(3)

c read in gaussian distribution from disk file

open (4, file='gaussian.dat', status='old', form='formatted')
read (4,25)gaus
25 format(8(f6.0,1x))
write(6,*) 'how many 420ms superframes would you like to run?'
read(5,*) sf
write(6,*) 'what golay mode? (1=soft, 2=hard, 3=nogolay) = '
read(5,*) modegolay
if (modegolay.eq.3) then
  write(6,*) 'would you like to try the Hamming code?
  *(1=yes, 0=no) . . . '
  read(5,*) ham
  if (ham.eq.1) qrc=0
  if (ham.eq.0) then
    write(6,*) 'would you like to perform a QRC error count?
    *(1=yes, 0=no) . . . '
    read(5,*) qrc
  end if
else
  ham=0
  qrc=0
end if
write(6,*) 'what is s/n ratio in dB during fade = ? (real #)
should be -24dB
read(5,*) noisef
write(6,*) 'what is s/n ratio in dB during no fade = ? (real #)
read(5,*) noise
write(6,*) 'what is mode? (0 for non-fading, 1 for fading) = '
read(5,*) mode
if (mode.eq.1) then
  write(6,*) 'what is the time seed? (any positive integer) = '
  read(5,*) iseed
  write(6,*) 'would you like to vary burst prob and width?
  *(1=yes, 0=no) . . . '
  read(5,*) vary
  if (vary.eq.1) then
    write(6,*) 'what burst prob?(r .94056 INMARSAT) = '
    read(5,*) bprob
    write(6,*) 'what burst width?(r 120=10ms, INMARSAT) = '
    read(5,*) bwidth
  end if
else
  iseed = 1
end if
**c****** write the superframe interleave table for the 144 codeword bits
**c****** only throughout frames 4 through 14 (frame 4 starts at bit 721)

```fortran
  do 35 m=1,9
    do 30 mm=1,8
      intlvtable(((m-1)*16)+((mm-1)*2)+1)=
        1 721+((m-1)*240)+((mm-1)*30)
      intlvtable(((m-1)*16)+((mm-1)*2)+2)=
        1 721+((m-1)*240)+((mm-1)*30)+10
    30 continue
  35 continue
```

**c*********** Initialize ****************************

```fortran
  itoterr = 0
  qrcsferr = 0
  qrccwcnt = 0
  success = 0
  fail = 0
  ifirst=1
  tbcnt=1680
  rbcnt=1680-10
```

**c***** Begin for sf# of superframes**************************
**c***** don't start counting results until n=about 3 so that error counters
**c***** are in sync (test if insync and insyncfill .eq..true)

```fortran
  do 500 n=1,sf
    loop for 1680 baud/sf - generate, tx, sim, rcv, etc.
      do 400 mk=1,1680
        tbcnt=tbcnt+1
        if(tbcnt.eq.1681)then
          tbcnt=1
      400 continue
    c***** fill mibits
      if(qrc.eq.1)nummi=72
      if(qrc.eq.0)nummi=36
      do 40 m=1,nummi
        call bitgen11(2,init,tbits)
        init=0
        if((qrc.eq.0).and.(ham.eq.0)) then
          mibits(((m-1)*2)+1)=0
          mibits(((m-1)*2)+2)=0
        if((tbits.eq.1).or.(tbits.eq.3))then
          mibits(((m-1)*2)+1)=1
```

53
end if
if((tbits.eq.2).or.(tbits.eq.3))then
  mibits(((m-l)*2)+2)=1
end if
end if
if(hani.eq.1) then
  hambits(((C(m-l)*2)+1)=0
  hambits( ((C(m-l)*2)+2)=0
  if((tbits.eq.1).or.(tbits.eq.3))then
    hazbits(((m-l)*2)+1)=1
  end if
  if((tbits.eq.2).or.(tbits.eq.3))then
    hambits(((m-l)*2)+2)=1
  end if
end if
if(qrc.eq.1) then
  qrcbits(((m-l)*2)+1)=0
  qrcbits( ((m-1)*2)+2)=0
  if((tbits.eq.1) .or. (tbits.eq.3))then
    qrcbits( ((m-l)*2)+1)=1
  end if
  if((tbits.eq.2) .or. (tbits.eq.3))then
    qrcbits( ((m-1)*2)+2)=1
  end if
end if
continue
if (qrc.eq.1) go to 95

c***** fill parbits
if (ham.eq.1) parnum=18
if ( (ham.eq.0).and.(qrc.eq.0)) parnum=6
  call matrixgen(8,4,hmatrix,syndrometable)
do 70 m=1,parnum
  c***** bit encoder for hamming
  if(ham.eq.1)then
    do 48 mm=1,2
      codeword(((mm-1)*2)+1)=hambits(((mm-1)*4)
                  +(((mm-1)*2)+1))
      codeword(((mm-1)*2)+2)=hambits(((mm-1)*4)
                  +(((mm-1)*2)+2))
c      codeword(((mm-1)*2)+5)=0
      codeword(((mm-1)*2)+6)=0
    continue
    call encodeham(8,4,hmatrix,paritybit,codeword)
  end if
40 continue
c***** load hamming codewords (parity + data) ***********************
do 50 mm=1,8
   code(((m-1)*8)+mm)=codeword(mm)
50 continue
else

   c***** generate parity bauds for golay here
   if (qrc.eq.0) then
      do 55 mm=1,6
         idata(((mm-1)*2)+1)=mibits(((m-1)*12)+(((mm-1)*2)+1))
         idata(((mm-1)*2)+2)=mibits(((m-1)*12)+(((mm-1)*2)+2))
      55 continue
   end if
   call golenc
   do 60 mm=1,12
      parity(((m-1)*12)+mm)=idata(12+mm)
   60 continue
   continue

   c***** load Golay and QRC codewords (bits)*************************
   if((ham.eq.0).and.(qrc.eq.0)) then
      do 90 m=1,6
         do 80 mm=1,12
            code(((rn-i)*24)+m)=mibits((C(m-l)*12).mm)
            codeC(((m-l)*24)+mm+12)=parity(((m-l)*12)+mm)
         80 continue
      90 continue
   end if
   continue
   if(qrc.eq.1) then
      do 97 m=1,144
         code(m)=qrcbits(m)
      97 continue
   end if

   c***** scramble the codewords
   do 100 m=1,144
      hashcode(m)=code(hashtable(m))
   100 continue
   do 110 m=1,1608

   continue
c get the rest of the superframe's bits
call bitgenifill(2,initfill,tbits)
c same as bitgen11 but a separate routine

initfill=0
fillbits(((m-1)*2)+1)=0
fillbits(((m-1)*2)+2)=0
if((tbits.eq.1).or.(tbits.eq.3))then
  fillbits(((m-1)*2)+1)=1
  end if
if((tbits.eq.2).or.(tbits.eq.3))then
  fillbits(((m-1)*2)+2)=1
  end if
110 continue

c***** load the superframe with codewords and fill
intlvcnt=1
ihashcnt=1
ifillcnt=1
do 130 m=1,3360
  if(m.eq.(intlvtable(intlvcnt)))then
    tbit(m)=hashcode(ihashcnt)
    ihashcnt=ihashcnt+1
    intlvcnt=intlvcnt+1
  else
    tbit(m)=fillbits(ifillcnt)
    ifillcnt=ifillcnt+1
  end if
130 continue

c***** bits to bauds
  do 140 m=1,1680
    if(tbit(((m-1)*2)+1).eq.0)then
      if(tbit(((m-1)*2)+2).eq.0)tbau(m)=0
      if(tbit(((m-1)*2)+2).eq.1)tbau(m)=2
      else
        if(tbit(((m-1)*2)+2).eq.0)tbau(m)=1
        if(tbit(((m-1)*2)+2).eq.1)tbau(m)=3
      end if
140 continue
end if

txbits=tbau(mk)
c***** differential phase encoding
index = (4*txbits) + prevoutdibit
txbit = table (index+1)
prevoutdibit = txbits

c***** transmit over simulated channel and receive
   call tx(txbits,samples)
   if (vary.eq.1) then
      call chsim(samples,noise,noisef,mode,iseed,gaus,
                 bprob,bwidth)
   else
      call sim(samples,noise,noisef,mode,iseed,gaus,n, sf,mk)
   end if
   call rcv(samples,rbits,xdisp,ydisp,confh,conf1)

c***** The confidence calculations are used for softgolay only *****
   rbcnt = rbcnt + 1
   rbaud(rbcnt-1) = rbits
   intlvconf(((rbcnt-2)*2)+1) = confl
   intlvconf(((rbcnt-2)*2)+2) = confh
   if (rbcnt.eq.1681) then
      testcnt = testcnt + 1
      rbcnt = 1
      if (ifirst.eq.1) then
         ifirst = 0
      else
         c***** unpack and decode the bits and count any errors in the mi
         c this would be 18 calls to hamming or 6 calls to golay
         c if any of the 72 mi bits are in error then we fail and
         c***** bauds to bits
         do 150 m = 1,1680
            rbit(((m-1)*2)+1) = 0
            rbit(((m-1)*2)+2) = 0
            if((rbaud(m).eq.1).or.(rbaud(m).eq.3))
               rbit(((m-1)*2)+1) = 1
            if((rbaud(m).eq.2).or.(rbaud(m).eq.3))
               rbit(((m-1)*2)+2) = 1
         150 continue

c***** deinterleave bits from the superframe
   intlvrcnt = 1
   ihashrcnt = 1
   ifillrcnt = 1
   do 160 m = 1,3360
      if (m .eq. (intlvtable(intlvrcnt))) then

rhashcode(ihashrcnt)=rbit(m)
rhashconf(ihashrcnt)=intlvconf(m)

ihashrcnt=ihashrcnt+1
intlvrcnt=intlvrcnt+1

else
rfillbits(ifillrcnt)=rbit(m)
rfillconf(ifillrcnt)=intlvconf(m)
ifillrcnt=ifillrcnt+1
end if

continue

C***** descramble the codeword bits

do 170 m=1,144
rcode(hashtable(m))=rhashcode(m)
rcodeconf(hashtable(m))=rhashconf(m)

170 continue

C***** decode the codewords (baud)***********************

if(qrc.eq.1)then
decodcnt1=3
decodcnt2=24
end if
if(ham.eq.1)then
decodcnt1=18
decodcnt2=2
end if
if((qrc.eq.0).and.(ham.eq.0))then
decodcnt1=6
decodcnt2=6
end if

do 200 m=1,decodcnt1
if((ham.eq.0).and.(qrc.eq.0))then
do 180 mm=1,24
idata(mm)=rcode(((m-1)*24)+mm)
conf(mm)=rcodeconf(((m-1)*24)+mm)

180 continue
end if
if(ham.eq.1)then
do 181 mm=1,8
codeword(mm)=rcode(((m-1)*8)+mm)

181 continue
end if
if(qrc.eq.1)then
do 182 mm=1,48

58
qdata(mm) = rcode(((m-1)*48)+mm)

182
continue
end if

if(modegolay.eq.1) call soft
if(modegolay.eq.2) call hard
if(ham.eq.1) then
    call decodeham(8,4,paritybit,myerror,hmatrix,
syndrometable,codeword)

continue

183

if(myerror.eq.1) then
    hamerr = hamerr + 2
    write(6,*),'decodeham detected errors'
end if
end if

c***** strip off the first 4 bits of each codeword MI bits.**********0

do 183 mm=1,4
    hdata(mm) = codeword(mm)
183
continue

if(myerror.eq.1) then
    hamerr = hamerr + 2
    write(6,*),'decodeham detected errors'
end if
end if

c***** count errors in 72 MI bits (golay=12/24,hamming=4/8)
c***** or all 144 bits (qrc=48/48)
c***** before the errors are counted they are changed to baud

c************************* qrcwerr=0

do 190 k=1,decodcnt2

190
    if((qrc.eq.0).and.(ham.eq.0)) then
        if(idata(((k-1)*2)+1).eq.0) then
            if(idata(((k-1)*2)+2).eq.0) bits=0
            if(idata(((k-1)*2)+2).eq.1) bits=2
        else
            if(idata(((k-1)*2)+2).eq.0) bits=1
            if(idata(((k-1)*2)+2).eq.1) bits=3
        end if
    end if
end if

if(ham.eq.1) then
    if(hdata(((k-1)*2)+1).eq.0) then
        if(hdata(((k-1)*2)+2).eq.0) bits=0
        if(hdata(((k-1)*2)+2).eq.1) bits=2
    else
        if(hdata(((k-1)*2)+2).eq.0) bits=1
        if(hdata(((k-1)*2)+2).eq.1) bits=3
    end if
end if
End if

59
if(qrc.eq.1) then 
  if(qdata(((k-1)*2)+1).eq.0) then 
    if(qdata(((k-1)*2)+2).eq.0) bits=0 
    if(qdata(((k-1)*2)+2).eq.1) bits=2 
  else 
    if(qdata(((k-1)*2)+2).eq.0) bits=1 
    if(qdata(((k-1)*2)+2).eq.1) bits=3 
  end if 
end if 
call biterr1la(bits,2,insync,nerror) 
iioterr=iioterr+nerror 
if(qrc.eq.1)qrccwerr=qrccwerr+nerror 
190 continue

c***** This is the QRC BER counter section ****************************
c errors/codeword are checked (for decodcnti=3 cws) 
c QRC(48,24) will correct 5 errors/cw and 62.02% of the 6th errors 
c**************************************************

if(qrc.eq.1) then 
  if(qrccwerr.eq.0) then 
    write(6,*)'there were no errors in QRC cw' 
  end if 
  if((qrccwerr.le.5).and.(qrccwerr.gt.0)) then 
    write(6,*)'# errors corrected = ',qrccwerr 
    qrccwerr=0 
  end if 
  if(qrccwerr.eq.6) then 
    probq = rand(0) 
    if(probq.le..6202)then 
      write(6,*)'qrc 6th error corrected' 
      qrccwerr=0 
    end if 
  end if 
  qrcsferr=qrccwerr+qrcsferr 
end if 
195 continue 
200 continue

c***** print error results 
c if any errors in total error count , the run is a failure 
c**************************************************

if(qrc.eq.1) then 
  iioterr = qrcsferr 
  iallerr = iallerr + qrcsferr 
  itotbitct = itotbitct + 144
end if
if(itoterr.eq.0)then
  success=success+1
  write(6,*)'success = ',success
end if

if(itoterr.ne.0)then
  fail=fail+1
  write(6,*)'fail = ',fail
  write(6,*)'total errors in sf = ',itoterr
end if

c***** reinitialize for new superframe *******************
qrcs ferr=0
itoterr=0
hamerr=0
if(.not.insync)then
  fail=0
  success=0
  testcnt=0
  itotbitct=0
  iallerr=0
end if

c***** or use the fill bits to measure ber in non-fading noise
ca call biterrfill(bit,2,insyncfill,nerror)
end if

400 continue
write(6,*)' you just completed superframe # ',n
if((qrc.eq.1) .and. (mod(n,50) .eq.0))write(6,*)
  1 'i allerr = ',i allerr,' itotbitct = ',itotbitct,
  1 'n = ',n
500 continue

c***** print final counts and results
write(6,*)' the total number of failures = ',fail
write(6,*)' the total number of successes = ',success
if(modegolay.eq.1) write(6,*)' This was a soft golay run'
if(modegolay.eq.2) write(6,*)' This was a hard golay run'
if(modegolay.eq.3) write(6,*)' This was a nogolay run'
if(ham.eq.1)write(6,*)' This was a Hamming run'
if(qrc.eq.1)write(6,*)' This was a QRC BE run'
write(6,*)' This run used fading (1=yes,0=no) ',mode
if(vary.eq.1)write(6,*)'fading width, bwidth = ',bwidth
write(6,*)' s/n during fade (dB) = ', noise
write(6,*)' s/n during non-fade (dB) = ', noise

c*****QUIT!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
end
B. SUBROUTINES

subroutine biterrlla(data,nbits,insync,nerror)

integer*4 data, nbits, nerror, rxdata, shindex, error
integer*4 index, errcnt, bitcnt, i, itemp
integer*4 nblck, btest, lshift, rshift, xor
logical insync

data errcnt, bitcnt / 0, 0 /
data rxdata, index / 0, 1 /
data nblck /100/

rxdata = rshift(rxdata,nbits)
call mvbits(data,0,nbits,rxdata,11-nbits)

shindex = lshift(index,2)
itemp = xor(index,shindex)
index = rshift(index,nbits)
call mvbits(itemp,2,nbits,index,11-nbits)

nerror = 0
error = xor(index,rxdata)
if (error .ne. 0) then
   do 9 i =11-nbits, 10
      if (btest(error,i).eq.1) nerror = nerror +1
9 continue
end if

errcnt = errcnt + nerror
bitcnt = bitcnt + nbits

if (insync) then
   if (bitcnt .ge. nbJ.-k) then
      if (float(errcnt) .gt. .35 * float(bitcnt)) then
         insync = .false.
         nblck=100
      else
         write(6,*) 'errors in ',nblck, ' bits = ', errcnt
         nblck=3000
         to span fades
         errcnt = 0
         bitcnt = 0
      end if
   else
      write(6,*) 'errors in ',nblck, ' bits = ', errcnt
      nblck=3000
      to span fades
      errcnt = 0
      bitcnt = 0
   end if

end if

63
if (.not. insync) then
   if (bitcnt .ge. 20) then
      if (float(errcnt) .le. .2 * float(bitcnt)) then
         insync = .true.
         write(6,'(a)') 'insync = true'
      else
         if (bitcnt .eq. 20) write(6,'(a)') 'not yet in sync'
            index = rxdata
            if (index .eq. 0) index = 1
         end if
      end if
      errcnt = 0
      bitcnt = 0
   end if
end if
return
end

---------------------------------------------------------------------

subroutine biterrllafill(data,nbits,insync,nerror)

   integer*4 data, nbits, nerror, rxdata, shindex, error
   integer*4 index, errcnt, bitcnt, i, itemp
   integer*4 nblck, btest, lshift, rshift, xor
   logical insync
   data errcnt, bitcnt / 0, 0 /
   data rxdata, index / 0, 1 /
   data nblck /100/

   rxdata = rshift(rxdata,nbits)
   call mvbits(data,0,nbits,rxdata,11-nbits)

   shindex = lshift(index,2)
   itemp = xor(index,shindex)
   index = rshift(index,nbits)
   call mvbits(itemp,2,nbits,index,11-nbits)

   nerror = 0
   error = xor(index,rxdta)
   if (error .ne. 0) then
      do i =11-nbits, 10
         if (btest(error,i).ne.0) nerror = nerror +1
      end do
   end if
64
continue
end if

errcnt = errcnt + nerror
bitcnt = bitcnt + nbits

if (insync) then
  if (bitcnt .ge. nblck) then
    if (float(errrant) .gt. .35 * float(bitcnt)) then
      insync = .false.
    else
      write(6,*) 'errors in ', nblck, ' bits = ', errcnt
    end if
  end if
end if

if (.not. insync) then
  if (bitcnt .ge. 20) then
    if (float(errcnt) .le. .2 * float(bitcnt)) then
      insync = .true.
      write(6,*) 'insync = true'
    else
      if (bitcnt.eq.20) write(6,*) 'not yet in sync'
      index = rxdata
      if (index .eq. 0) index = 1
    end if
    errcnt = 0
    bitcnt = 0
  end if
end if

return
end

C******************************************************************************************

function btest(i1, i2)

C *** this function returns 1 if bit position a2 of a1 is 1 and
C *** returns 0 if bit position a2 of a1 is 0

65
integer*4 t1,t2,btest,i1,i2

  t1=i1
  t2=i2
  t1 = rshift(t1,t2)
  t1 = and(t1,1)
  btest = t1

  return
end

C******************************************************************************************

subroutine mvbits(isource,ibitfrom,length,idest,ibitto)

integer*4 isource, ibitfrom, length, idest, ibitto, n
integer*4 k, ks, btest, xor, lshift

  do 12 n = 1, length
  
    k = ibitto + n - 1
    ks = lshift(1,k)

    if (btest(isource,(ibitfrom+n-1)).ne.0) then
      c **set**
        if (btest(idest,k).eq.0)
          1
        idest = xor(idest,ks)
      c **clear**
      else
        if (btest(idest,k).ne.0)
          1
        idest = xor(idest,ks)
      end if
  12  continue

  return
end

C-------------------------------------------------------------------------------------------------

c soft decision decoder for golay(24,12)
c uses chase ii algorithm with table look-up decoder

subroutine soft
common/blk5/idata
common/blk7/conf,isym,kdata,ierr
integer*4 idata(24),kdata(24),ierr(5)
integer*4 isym(2047)
n integer*4 nb(4),mask(4,24),jb(16,24),ib(24),nerr
real conf(24)
n
nerr=1

C

C

w is analog weight of most likely error pattern
set w initially to a very large value
w=1000000.0

identify 4 bits with lowest confidence values
call low4(nb)
wg4=0.

gen 4 masks identifying 4 bits
call mgen(mask,nb)

gen 2**4=16 patterns
call patgen(mask,jb)

test each of 16 patterns for the SOFT decoder
do 19 i=1,16

clear ierr
do 13 j=1,5
   ierr(j)=0
continue

add data and test error pattern, store in kdata
do 14 j=1,24
   call exor(kdata(j),idata(j),jb(i,j))
continue

binary decoder
c accepts input data in kdata
c returns data uncorrected
c returns no. of errors in ierr(5)
c returns location of errors in ierr(1) to ierr(4)
call golay

c test if decoded word was error free
if (ierr(5).eq.0) then
  c test word was error free
  c accept test word as best estimate
  w=0.0
  do 15 j=1,24
    idata(j)=kdata(j)
  15 continue
  return
else
  c test word contained errors
  c get final error pattern, calculate its analog wgt
  wgt=0.0
  n=ierr(5)
  c add detected errors to test error pattern
  do 16 j=1,n
    ix=ierr(j)
    call exor(jb(i,ix),jb(i,ix),nerr)
  16 continue
  do 17 j=1,24
    if(jb(i,j).eq.1)wgt=wgt+conf(j)
  17 continue
  c compare wgt and w
  c save error pattern with lowest analog weight
  if(wgt.lt.w)then
    c store new value
    c save error pattern
    w=wgt
    do 18 j=1,24
      ib(j)=jb(i,j)
    18 continue
  end if
end if
19 continue
  c decode data with most likely error pattern
  do 21 i=1,24
    call exor(idata(i),idata(i),ib(i))
hard decision decoder for golay(24,12)
uses chase ii algorithm with table look-up decoder

subroutine hard

integer*4 idata(24),kdata(24),ierr(5)
integer*4 isym(2047)
integer*4 nb(4),mask(4,24),jb(16,24),ib(24),nerr
real conf (24)
common/blk5/idata
common/blk7/conf,isym,kdata,ierr

nerr=1

w is analog weight of most likely error pattern
set w initially to a very large value
w=1000000.0

identify 4 bits with lowest confidence values
call low4(nb)
wgt4=0.

generate 4 masks identifying 4 bits
call mgen(mask,nb)

generate 2**4=16 patterns
call patgen(mask,jb)

test 1 pattern --> hard
do 28 i=1,1

clear ierr
do 22 j=1,5
   ierr(j)=0
22 continue

add data and test error pattern, store in kdata
do 23 j=1,24
call exor(kdata(j),idata(j),jb(i,j))

23            continue

c            binary decoder
c            accepts input data in kdata
c            returns data uncorrected
c            returns no. of errors in ierr(5)
c            returns location of errors in ierr(1) to ierr(4)

call golay

c            test if decoded word was error free
if (ierr(5).eq.0) then
  c            test word was error free
  c            accept test word as best estimate
  w=0.0

do 24 j=1,24
   idata(j)=kdata(j)
24            continue

return

else
  c            test word contained errors
  c            get final error pattern, calculate its analog wgt
  wgt=0.0
  n=ierr(5)

c            add detected errors to test error pattern
do 25 j=1,n
   ix=ierr(j)
   call exor(jb(i,ix),jb(i,ix),nerr)
25            continue

do 26 j=1,24
   if(jb(i,j).eq.1)wgt=wgt+conf(j)
26            continue

c            compare wgt and w

c            save error pattern with lowest analog weight

if(wgt.lt.w)then
  c            store new value
  c            save error pattern
  w=wgt
  do 27 j=1,24

!b(j)=jb(i,j)
27    continue
    end if
    end if
28    continue

c    decode data with most likely error pattern
    do 29 i=1,24
      call exor(idata(i),idata(i),ib(i))
29    continue
    return
end

c--------------------------------------------------------

subroutine low4(nb)

c    to identify in nb the location of 4 lowest conf values
dimension d(4)
integer*4 kdata(24),ierr(5),nb(4)
integer*4 isym(2047)
real conf(24)
common/blk7/conf,isym,kdata,ierr-

c    assume first 4 are lowest
    do 31 i=1,4
      d(i)=conf(i)
      nb(i)=i
31    continue

c    compare remaining values
    do 33 i=5,24
      a=conf(i)
      n=i
      j=1
      if (j.gt.4) go to 33
    if (a.lt.d(j)) then
      c      a is < d(j)
      c      replace d(j) with a
      c      save d(j) to compare with other "values in d
      c      save nb(j)
      b=d(j)
      nx=nb(j)
33  continue
c replace
d(j)=a
nb(j)=n
c rename the saved values
a=b
n=nx
j=1
end if
c incr j
j=j+1
33 continue
57 return
end

-------------------------------------------------------------------------------------------------------------------

c mask generator for soft decision golay decoder
c generate 4 masks with 1 bit in location nb(i)

subroutine mgen(mask,nb)

dimension mask(4,24),nb(4)
integer*4 mask,nb

do 35 i=1,4
do 34 j=1,24
   mask(i,j)=0
34 continue
n=nb(i)
mask(i,n)=1
35 continue

return
end

-------------------------------------------------------------------------------------------------------------------

c generate 16 patterns based on 4 bits

subroutine patgen(mask,jb)

integer*4 mask(4,24),jb(16,24)

do 36 i=1,24


\[
\begin{align*}
\text{jb}(1,i) &= 0 \\
\text{jb}(2,i) &= \text{mask}(1,i) \\
\text{jb}(3,i) &= \text{mask}(2,i) \\
\text{jb}(4,i) &= \text{mask}(3,i) \\
\text{jb}(5,i) &= \text{mask}(4,i) \\
\text{jb}(6,i) &= \text{jb}(2,i) + \text{mask}(2,i) \\
\text{jb}(7,i) &= \text{jb}(2,i) + \text{mask}(3,i) \\
\text{jb}(8,i) &= \text{jb}(2,i) + \text{mask}(4,i) \\
\text{jb}(9,i) &= \text{jb}(3,i) + \text{mask}(3,i) \\
\text{jb}(10,i) &= \text{jb}(3,i) + \text{mask}(4,i) \\
\text{jb}(11,i) &= \text{jb}(4,i) + \text{mask}(4,i) \\
\text{jb}(12,i) &= \text{jb}(6,i) + \text{mask}(3,i) \\
\text{jb}(13,i) &= \text{jb}(6,i) + \text{mask}(4,i) \\
\text{jb}(14,i) &= \text{jb}(7,i) + \text{mask}(4,i) \\
\text{jb}(15,i) &= \text{jb}(9,i) + \text{mask}(4,i) \\
\text{jb}(16,i) &= \text{jb}(12,i) + \text{mask}(4,i)
\end{align*}
\]

36 continue

\[
\begin{align*}
do 56 &\text{ j}=1,16 \\
do 37 &\text{ i}=1,24 \\
\text{jb}(j,i) &= \text{and} (\text{jb}(j,i), 1) \\
37 &\text{ continue} \\
56 &\text{ continue} \\
\end{align*}
\]

return

end

---

c exclusive or of two integer*4s

c one bit per integer*4

subroutine exor(i,j,k)

integer*4 i,j,k

i=and(j+k,1)

return

end

---

binary table look-up decoder for golay (24,12) code
subroutine golay

integer*4 idata(24), ierr(5), kdata(24), is(11)
integer*4 isym(247)
real conf(24)
common/blk5/idata
common/blk7/conf,isym,kdata,ierr

n=23
ierr(5)=0

c generate syndrome
write(6,*) 'generating golay syndrome'
call encode(kdata, is,n)

c use syndrome to generate index to table
ir=0
do 38 j=1,11
   ir=ir*2+is(j)
38 continue

if(ir.ne.0)then
   look up error pattern
   possibility of up to 3 errors
   recorded as three 5-bit symbols in a 16-bit word
   ix=isym(ir)
   identify error locations and count errors
   do 39 j=1,3
      ierr(j)=and(ix,31)
      if(ierr(j).gt.0) ierr(5)=ierr(5)+1
      ix=ix/32
   c if(ierr(j).gt.0) write(6,*) 'ierr(j)=',ierr(j)
39 continue
end if

c test overall parity
ir=ierr(5)
do 41 j=1,24
   ir=ir+kdata(j)
41 continue

ir=and(ir,1)

c test for even parity
if (ir.ne.0) then
 parity check failed
force overall parity
increment error count

ir= ierr(5)+1
 ierr(ir)=24
 ierr(5)=ir

end if

return
end

golay(23,42) encoder, syndrome generator

subroutine encode(idata, ip, n)

integer*4 idata(12),ip(11),ix

g(x)=x**11+x**9+x**7+x**6+x**5+x+1

clear ip

do 42 i=1,11
ip(i)=0
42 continue

read in n bits

do 43 i=1,n
    call exor(ix,idata(i),ip(1))
    ip(1)=ip(2)
    call exor(ip(2),ix,ip(3))
    ip(3)=ip(4)
    call exor(ip(4),ix,ip(5))
    call exor(ip(5),ix,ip(6))
    call exor(ip(6),ix,ip(7))
    ip(7)=ip(8)
    ip(8)=ip(9)
    ip(9)=ip(10)
    call exor(ip(10),ix,ip(11))
    ip(11)=ix
43 continue

return
end
subroutine bitgen11(nbits, ival, data)

c this routine generates a pseudorandom sequence of bits using an 11 bit

c shift register, exoring the 9th and 11th bits to generate the new 1st bit

c before shifting. the length of the pseudorandom sequence is (2**11)-1

c or 2047 bits.

integer*4 data, shindex, nbits, ival, index, temp
integer*4 ishift, rshift, xor

if (ival .gt. 0 .and. ival .lt. 2048) index = ival

data = 0

c ensure that all bits in data are 0

call mvbits(index, 0, nbits, data, 0)
c move bits from index to data

shindex = lshift(index, 2)
c shindex is index shifted left 2

temp = xor(index, shindex)
c temp contains new bits for index

index = rshift(index, nbits)
c shift index right by nbits

call mvbits(temp, 2, nbits, index, 11-nbits)
c move new bits, temp to index

return
derend

c---------------------------------------------------------------
c
golay encoder

subroutine golenc

integer*4 idata(24), id(12), is(11)
common/blk5/idata

n=12
c load 12 information bits
c sum inf bits for overall parity
do 44 j=1,12
   id(j)=idata(j)
44 continue
c encode using n-k type shift register
call encod(id,is,n)
c store parity bits in same array as inf bits
do 45 j=1,11
   idata(j+12)=is(j)
45 continue
c generate over all parity bit(even parity)
c store as 24th bit in array
   m=0
do 46 j=1,23
      m=m+idata(j)
46 continue
idata(24)=and(m,1)
return
end

------------------------------------------------------------------------------
c golay(23,12) encoder, syndrome generator
c x11+x9+x7+x6+x5+x+1

subroutine encod(idata, ip,n)
integer*4 idata(12),ip(11),ix

c clear ip
do 47 i=1,11
   ip(i)=0
47 continue
c read in n bits
do 48 i=1,n
   call exor(ix,idata(i),ip(1))
   ip(1)=ip(2)
   call exor(ip(2),ix,ip(3))
ip(3)=ip(4)
call exor(ip(4),ix,ip(5))
call exor(ip(5),ix,ip(6))
call exor(ip(6),ix,ip(7))

ip(7)=ip(8)
ip(8)=ip(9)

ip(9)=ip(10)
call exor(ip(10),ix,ip(11))

ip(11)=ix

48 continue

return

derend

-----------------------------------------------------------------------------

subroutine bitgen11fill(nbits,ival,data)

c this routine generates a pseudorandom sequence of bits using an 11 bit
c shift register, exoring the 9th and 11th bits to generate the new 1st bit
c before shifting. the length of the pseudorandom sequence is (2**11)-1
c or 2047 bits.

integer*4 data, shindex, nbits, ival, index, temp
integer*4 lshift, rshift, xor

if (ival .gt. 0 .and. ival .lt. 2048) index = ival

data = 0
c ensure that all bits in data are 0

call mvbits(index,0,nbits,data,0)
c move bits from index to data

shindex = lshift(index,2)
c shindex is index shifted left 2

temp = xor(index,shindex)
c temp contains new bits for index

index = rshift(index,nbits)
c shift index right by nbits

call mvbits(temp,2,nbits,index,11-nbits)
c move new bits, temp to index
subroutine tx(dibit,txsamp)

dimension txbuff(32), txsamp(3)
integer*4 dibit, index, i
real txbuff, txsamp
data txbuff, index / 32*0., 1 /

call modulate(dibit,index,txbuff)
c modulate one baud or symbol

c add symbol to transmit buffer

c note: each symbol is filtered by the modulator to be wider than

c one baud so as to restrict the bandwidth to about 4000 Hz for

c reduction of ACI (adjacent channel interference), so that the resulting

c baud pulses overlap and must be added together

   do 49 i = 1, 3
      txsamp(i) = txbuff(index)
      txbuff(index) = 0
      index = mod(index,32) + 1
   49 continue

return
end

c subroutine modulate(mbit,index,buffer)

integer*4 mbit, phasec, cphase, tphase
integer*4 point, wlength, i, index, modangle
real s, pi, buffer, w
dimension phasec(4), w(29), buffer(*)

data phasec / 225,135,315,45 /
data wlength, modangle, cphase, pi / 29, 0, 0, 3.141593 /

   c this is the transmit window to shape the output spectrum to about
   c 4000 Hz wide
* * *

data w / -.005696,.062601,.028191,-.021020,-.060997,
* -.035666,.049659,.106424,.043728,-.114226,-.203894,
* -.049151,.358925,.805832,.999756,.805832,.358925,
* -.049151,-.203894,-.114226,.043728,.106424,.049659,
* -.035666,.060997,-.021020,.028191,.062601,-.005696/

modangle = phasec(mbit + 1)
c modulation phase change angle

cphase = mod(cphase + 270, 360)
c continuous unmodulated carrier phase

tphase = mod(cphase + modangle, 360)
c actual transmit phase

point = index

c take the sine wave of the actual transmit phase and window it it
do 51 i = 1, wlength
   s = .364 * w(i) * sin(pi / 180. * (tphase + 90 * i))
   buffer(point) = buffer(point) + s
   point = mod(point, 32) + 1
51 continue

return
end

* * *

subroutine rcv(samples, dbits, xdisp, ydisp, confh, confl)

c ,purpose: this routine demodulates an 8000 b/s 4 phase dpsk modem signal.
c it is called once to demodulate each baud.

c inputs: sigin - contains samples of the signal to be demodulated.
c sigin(1) is first in time.

c outputs: dbits - an integer which contains the 2 demodulated bits.
c The data is stored in the least significant bits;
c the lsb is first in time.

c brief description of subroutine functions:

c equalize - calculates the output of the adaptive equalizer.
c decode - uses the equalizer output to determine the data bits.
a phase reference is calculated for internal use.

\[
\text{integer*4 dbits, i}
\]
\[
\text{real confh,confl}
\]
\[
\text{real signal, x, y}
\]
\[
\text{real xdisp,ydisp}
\]
\[
\text{real signal(48), samples(3)}
\]
\[
\text{real xcoef(48),ycoef(48)}
\]

\[\text{common /adequ/ xcoef, ycoef}\]

\[\text{c compromise equalizer coefficients for removing the intersymbol}\]
\[\text{c interference introduced by the transmitter windowing :}\]
\[\text{data xcoef / 0.00047277, 0.00018969, 0.00152612,-0.00026510,}\]
\[1 0.00288611,-0.00077746, 0.00057433,-0.00176436,\]
\[1 -0.00299245,-0.00031443,-0.01333420, 0.00245006,\]
\[1 0.00073868, 0.00575836, 0.00552585, 0.00128170,\]
\[1 0.01571514,-0.00732426, 0.00609350,-0.01291209,\]
\[1 -0.03999998,-0.00832545,-0.1046436, 0.00526107,\]
\[1 -0.14213182, 0.01429809, 0.38911167, 0.03028025,\]
\[1 -0.0619568, 0.01810114,-0.00429513, 0.00713205,\]
\[1 0.01294118, 0.00074462, 0.0006572, 0.0038059,\]
\[1 -0.01742128, 0.00060559,-0.00858223, 0.00059077,\]
\[1 0.00697080,-0.00119318, 0.00344639,-0.00073849,\]
\[1 0.00324149,-0.00039312, 0.00125349,-0.00045128/\]

\[\text{data ycoef / -0.00016221, 0.00070496,-0.00036193,-0.00072445,}\]
\[1 -0.00059857,-0.00178603,-0.0014448,-0.00418214,\]
\[1 0.00070030, 0.00100549,-0.00036193,-0.00072445,\]
\[1 0.00079199, 0.00792035,-0.00374299, 0.01238339,\]
\[1 -0.00632491,-0.01299071, 0.00042162,-0.03951978,\]
\[1 0.01424530,-0.02844719, 0.02507262, 0.05758527,\]
\[1 0.02317356, 0.35042825, 0.01940719,-0.24283291,\]
\[1 -0.00569436,-0.01806669,-0.01134070,-0.00663402,\]
\[1 -0.00711616,-0.02175912,-0.0007578, -0.01645218,\]
\[1 0.00256463,-0.00212019, 0.00061785, 0.0215092,\]
\[1 -0.00076241, 0.00839603,-0.00143184,-0.00148724,\]
\[1 -0.00056496,-0.00310774,-0.00173826/\]

\[\text{data signal / 48*0.0 /}\]

\[\text{c signal buffer is as long as the equalizer that it will multiply against}\]
\[\text{c or 48 samples long}\]

\[\text{do 52 i = 1, 45}\]

\[\text{c shift signal buffer to accomodate new samples}\]
signal(i) = signal(i+3)
continue

do 53 i = 46, 48
  put new samples into signal buffer
  signal(i) = samples(i-45)
continue

call equalize(signal,x,y)
call decode(x,y,dbits,xdisp,ydisp,confh,conf1)
return
end

c-----------------------------------------------

subroutine equalize(signal,x,y)

c purpose: this routine generates the output of the equalizer; i.e., it
  calculates the output value of each fir equalizer filter.
c input:   signal - a real array which contains the received signal samples.
c         signal(1) is first in time.
c outputs: x - the x-coordinate of the equalizer output.
c         y - the y-coordinate of the equalizer output.

integer*4 i
real x, y
real xcoef(48),ycoef(48)
real signal(48)

common /adequ/ xcoef, ycoef

y = 0.
x = 0.

do 54 i = 1,48
   x = x + signal(i) * xcoef(i)
   y = y + signal(i) * ycoef(i)
continue

x=2.*x
y=2.*y
subroutine decode(x,y,dibit,xdisp,ydisp,confh,conf1)
c purpose: this routine determines the data bits given the x,y coordinates
c from the equalizer.
c inputs: x - the x-coordinate of the demodulated baud.
c y - the y-coordinate of the demodulated baud.
c outputs: dibit - the 2 data bits for the current baud; lsb is first in time.
integer*4 dibit
real mag, phaser, x, y, qang, preang, phasech, arctan
real angle, decang, angdisp, xdisp, ydisp, error
real angtab
dimension angtab(4)
real predecang,diffang
real confh,confl
real desmag

   data angtab / -.75, -.25, .25, .75 /
data preang / 0. /

   angle = arctan(y,x)
c convert from rectangular to polar coordinates

   decang = angle - phaser
c decang is the angle to be decoded

call adjust(decang)
c calculate dibit:
c    preang=qang
c    previously quantized angle

c    qang = angtab(int(2.*decang+2.)+1)
c    quantized angle

c    phasech = qang - preang

return
end
c quantized phase change

c if(phasech.lt.0.0)phasech=phasech+2.
c if(phasech.eq.0.0)dibit=2

c phase change of 0

c if(phasech.eq.0.5)dibit=0

c phase change of 90

c if(phasech.eq.1.0)dibit=1

c phase change of 180, or -180

c if(phasech.eq.1.5)dibit=3

c phase change of -90

diffang=decang-predecang

c unquantized phase change

call adjust(diffang)

predecang=decang

c previous decoded angle update

c compute confidence values:
if(diffang.lt.-.75)then
    dibit=1
    confl=(diffang+1.25)*2.
    confh=-(diffang+.75)*2.
else if(diffang.lt.-.25)then
    dibit=3
    confl=-(diffang+.25)*2.
    confh=(diffang+.75)*2.
else if(diffang.lt.0.25)then
    dibit=2
    confl=(diffang+.25)*2.
    confh=-(diffang-.25)*2
else if(diffang.lt.0.75)then
    dibit=0
    confl=-(diffang-.75)*2.
    confh=(diffang-.25)*2.
else
    dibit=1
    confl=(diffang-.75)*2.
    confh=-(diffang-1.25)*2.
end if

c calculate x and y values for constellation display:
mag = sqrt(x*x + y*y)
angdisp = (diffang+.25)
call adjust(angdisp)
angdisp = angdisp * 3.141593

desmag=.707/2
c scale the confidence by the radius (lower confidence in fade):
if(mag.lt.desmag)then
   confl=confl*mag/desmag
   confh=confh*mag/desmag
end if

c calculate error angle:
c
   error = decang - qang
c
   call adjust(error)

c update phase reference for next baud:

   phaser = phaser - .5
   the .5 comes from .25 (times 180 degrees)
c
   of carrier advance per baud plus .25
   c (or 45 degrees) thus shifting the phase computations
   c from 135,45,-45,-135 to the 90,0,-90,-180 domain
   call adjust(phaser)

   return
   end

function arctan(y,x)

   c 4 quadrant arctangent -1 <= arctan < 1
   c -1 corresponds to -180 degrees

   real arctan, y, x

   arctan = 0.0
   if (x .ne. 0.0 .or. y .ne. 0.0) arctan = atan2(y,x)/3.141593

   return
end

c---------------------------------------------------------------

subroutine adjust (value)
  c routine to ensure value is kept in range -1 to 1.

  real value

  value = amod(value,2.)
  if (value .lt. -1.) value = value + 2.
  if (value .ge. 1.) value = value - 2.

  return
end


  c---------------------------------------------------------------

subroutine sim (samples, noise, noisef, mode, iseed, gaus, n, sf, mk)
  c--------------------------variables and parameters-----------------
  c noise=signal/noise in dB
  c mode = 0 for non-fading, 1 for fading

  real samples(3)
      real gaus(256)
  c gaussian distribution of noise table
  real noise, noisef
  real sv(120), nv(120)
  real sdb, ndb
  integer*4 sinn, nin, n, sf
  integer*4 fade, mode, fadecnt
  integer*4 iseed
      integer*4 ms10, ms20, ms40, ms100, ms200

  c--------------------------variable initialization-----------------

  if (ifirst.eq.0) then
    rgn = (10.**(-noise/20.))
  c compute actual noise scaler from dB
    rgnf = (10.**(-noisef/20.))
  c compute noisef scaler from dB
    ifirst=1
test = rand(iseed)
    fade = 0
    fadecnt = 0
    ms10 = 0
    ms20 = 0
    ms40 = 0
    ms100 = 0
    ms200 = 0

end if

c-------------------------------------------------------------------------------
do 55 m=1,3

c measure the RMS of past 120 samples of signal
    s = samples(m)
    sinn = mod(sinn+1,120)
    sdb = (sdb**2.)-sv(mod(sinn+1,120)+1)
    sv(sinn+1) = (s**2.)/120.
    sdb = sdb + sv(sinn+1)
    if (sdb.lt.0.) sdb = 0.
    sdb = sqrt(sdb)

c measure the RMS of past 120 samples of noise (direct from table)
    nin = mod(nin+1,120)
    ndb = (ndb**2.)-nv(mod(nin+1,120)+1)
    k = (255.*rand(0)) + 1
    nv(nin+1) = ((gaus(k)/25295.)**2.)/120.
    ndb = ndb + nv(nin+1)
    if (ndb.lt.0.) ndb = 0.
    ndb = sqrt(ndb)

if(mode.eq.1)then
    if(fadecnt.eq.0)then
        if(rand(0).gt..94056)then
            fade=1
        test=rand(0)
        if(test.lt.0.8)then
            fadecnt=120
            ms10 = ms10 + 1
        else if(test.lt.0.9)then
            fadecnt=240
            ms20 = ms20 + 1
        else if(test.lt.0.95)then
            fadecnt=480

87
ms40 = ms40 + 1
else if(test.lt.0.99) then
    fade cnt = 1200
    ms100 = ms100 + 1
else
    fade cnt = 2400
    ms200 = ms200 + 1
end if
else
    fade = 0
    fade cnt = 120
end if
else
    fade cnt = fade cnt - 1
end if

if((n.eq.sf) .and. (mk.eq.1680) .and. (m.eq.3)) then
    write(6,*)' number of 10ms fades in ',sf,
    ' runs = ',ms10
    write(6,*)' number of 20ms fades in ',sf,
    ' runs = ',ms20
    write(6,*)' number of 40ms fades in ',sf,
    ' runs = ',ms40
    write(6,*)' number of 100ms fades in ',sf,
    ' runs = ',ms100
    write(6,*)' number of 200ms fades in ',sf,
    ' runs = ',ms200
end if

if((mode.eq.0) .or. ((mode.eq.1) .and. (fade.eq.0))) then
    if (rand(0).ge.0.5) then
        if (ndb.ne.0.) samples(m) = samples(m) +
            1 (gaus(k)/25295.)*(sdb/ndb)*(rgn)
        else
            if (ndb.ne.0.) samples(m) = samples(m) -
                1 (gaus(k)/25295.)*(sdb/ndb)*(rgn)
        end if
    end if
end if

if((mode.eq.1) .and. (fade.eq.1)) then
    if (rand(0).ge.0.5) then
        c make positive noise
end if
if (ndb.ne.0.) samples(m) = samples(m) + 
   1
   (gaus(k)/25295.)*(sdb/ndb)*(rgnf)
else
   c make negative noise
   if (ndb.ne.0.) samples(m) = samples(m) - 
     1
     (gaus(k)/25295.)*(sdb/ndb)*(rgnf)
end if
end if

c scale signal down in fade :
   if((mode.eq.1).and.(fade.eq.1))samples(m)=samples(m)/10.

   continue
   return
end

subroutine chsim(samples,noise,noisef,mode,iseed,gaus,bprob,bwidth)

   c------------------
   variables and parameters------------
   c noisesignal/noise in dB
   c mode = 0 for non-fading, 1 for fading
   real samples(3)
      real gaus(256)
   c***** gaussian distribution of noise table
   real noise,noisef,bprob
   real sv(120),nv(120)
   real sdb,ndb,rtest
   integer*4 sinn,nin
   integer*4 fade,mode,fadecnt
   integer*4 iseed,bwidth

   c----------------------variable initialization------------------

   if (ifirst.eq.0) then
      rgn = (10.**(-noise/20.))
   c***** compute actual noise scaler from dB

      rgnf = (10.**(-noisef/20.))
   c***** compute noisef scaler from dB
   ifirst=1
      test=rand(iseed)
fadecnt = 0
rtest = 1. - (1. - bprob)/120
end if

do 55 m=1,3

C***** measure the RMS of past 120 samples of signal
  s = samples(m)
  sinn = mod(sinn+1,120)
  sdb = (sdb**2.)-sv(mod(sinn+1,120)+1)
  sv(sinn+1) = (s**2.)/120.
  sdb = sdb + sv(sinn+1)
  if (sdb.lt.0.) sdb = 0.
  sdb = sqrt(sdb)

C***** measure the RMS of past 120 samples of noise (direct from table)
  nin = mod(nin+1,120)
  ndb = (ndb**2.)-nv(mod(nin+1,120)+1)
  k = (255.*rand(0)) + 1
  nv(nin+1) = ((gaus(k)/25295.)**2.)/120.
  ndb = ndb + nv(nin+1)
  if (ndb.lt.0.) ndb = 0.
  ndb = sqrt(ndb)

if(mode.eq.1)then
  if(fadecnt.eq.0)then
    if(rand(0).gt.rtest)then
      fade=1
      fadecnt = bwidth
    else
      fade=0
      fadecnt=0
    end if
  else
    fadecnt=fadecnt-1
  end if
else
  fadecnt=fadecnt-1
end if

C***** not in fade
  if((mode.eq.0).or.((mode.eq.1).and.(fade.eq.0)))then
    if (rand(0).ge.0.5) then
      if (ndb.ne.0.) samples(m) = samples(m) +
1 \quad (\text{gaus}(k)/25295.)*(sdb/ndb)*\text{rgn}

\text{else}
\quad \text{if} \quad (\text{ndb}.ne.0.) \quad \text{samples}(m) = \text{samples}(m) - 1
\quad (\text{gaus}(k)/25295.)*(sdb/ndb)*\text{rgn}
\text{end if}
\text{end if}

c***** we are in fade
\text{if}\quad ((\text{mode}.eq.1). \text{and}. (\text{fade}.eq.1))\text{then}
c***** make positive noise
\quad \text{if} \quad (\text{rand}(0).ge.0.5) \text{then}
\quad \quad \text{if} \quad (\text{ndb}.ne.0.) \quad \text{samples}(m) = \text{samples}(m) + 1\quad (\text{gaus}(k)/25295.)*(sdb/ndb)*\text{rgnf}
\quad \text{c***** make negative noise}
\quad \text{else}
\quad \quad \text{if} \quad (\text{ndb}.ne.0.) \quad \text{samples}(m) = \text{samples}(m) - 1\quad (\text{gaus}(k)/25295.)*(sdb/ndb)*\text{rgnf}
\quad \text{end if}
\quad \text{end if}

c***** scale signal down in fade :
\text{if}\quad ((\text{mode}.eq.1). \text{and}. (\text{fade}.eq.1))\text{samples}(m) = \text{samples}(m)/200.

55 \quad \text{continue}

\text{return}
\text{end}

c******************************************************************************************************************

c \quad \text{encodeham}

c \quad \text{FUNCTION}

c \quad \text{This subroutine calculates the parity bits necessary}

c \quad \text{to form the codeword.}

c \quad \text{c}

c \quad \text{SYNOPSIS}

c \quad \text{encodeham(codelength1,codelength2,hmatrix,}

c \quad \text{paritybit,codeword)}

c \quad \text{c}

c \quad \text{formal}

c \quad \text{data} \quad \text{I/O}

c \quad \text{name} \quad \text{type} \quad \text{type} \quad \text{function
DESCRIPTION

This subroutine is part of a set of subroutines which perform a Generalized Hamming Code. As you know, Hamming codes are perfect codes and can only detect and correct one error. We added an overall parity checkbit, which allows us to detect 2 errors. When 2 errors are detected, (in subroutine decodeham.f) no correction attempt is made. This would most likely result in more errors. Instead, a flag is sent to the calling program notifying it of multiple errors so that smoothing may be attempted. The Hamming codes presently supported by the routines are (63,57), (31,26), (15,11), and shortened variations thereof. It could be made even more general by making minor modifications to the dectobin.f subroutine. This routine at present will calculate a maximum of 6 bits.

Hamming routines consist of the following files:

matrixgen - generates the hmatrix and sydrometable.

dectobin - does a simple decimal to binary conversion.

cencodeham - generates the codeword and overall paritybit.

cdecodeham - recovers infobits, checks for errors, corrects 1 error, and sends out flag for smoothing.

This subroutine performs the Hamming encode function. It will calculate the necessary parity bits, depending on which code is requested, and will add the overall parity bit to the end of the codeword generated.

REFERENCES

Lin and Costello: Error Control Coding

Berlekamp: Algebraic Coding Theory
C
C*****************************************************************************
C subroutine encodeham(codelength1,codelength2,hmatrix, 
  1  paritybit,codeword)
C
integer*4 codelength1,codelength2,paritybit
integer*4 codeword(codelength1),hmatrix(codelength1)
integer*4 parityflag,temp1,temp2,i,temp3
C
paritybit=0
parityflag=0
C *** parityflag = 0 if not using the extra parity bit.
temp1=codelength1-codelength2
temp2=0
temp3=0
C
C First generate the parity bits for the Hamming codeword. This is 
C relatively straightforward. hmatrix was generated in matrixgen.f, 
C which is called as part of the Hamming initialization routines.
C
do 10 i=1, codelength2
   if(codeword(i).ne.0) temp2= xor(temp2,hmatrix(i))
10 continue
C
C since the hmatrix is stored in a packed decimal format, the parity 
C bits must be unpacked and appended to the end of the bitsteam. 
C after this routine you will have the complete codeword.
C
call dectobin(temp1,temp2,codeword(codelength2+1))
C
C Now I check to see if the parityflag is set, indicating the user 
C requests an overall parity bit be generated. Normally this will 
C be the case.
C
temp2=0
if (parityflag.eq.1)then
  do 20 i=1,codelength1
     temp2 = xor(temp2,codeword(i))
     if (codeword(i).ne.0) temp3=temp3+1
20 continue
paritybit=temp2
end if
C
return
This subroutine decodes the bitstream generated by
encodeham. It will correct a single error, and detect 2
errors.

SUBROUTINE decodeham(codelength1, codelength2, hmatrix,
syndrometable, paritybit, codeword, myerror)

DATA I/O
name type type function
------------------------------------------------------------------------
codelength1 int i number of data bits
codelength2 int i number of information bits
hmatrix int i vector to encode an decode by
syndrometable int i errormasks used to correct single
c errors
paritybit int i overall parity bit
codeword int i/o encoded/decoded stream
myerror log o flag for 2 error detect
synflag int o value 0 or 1, 1 if syndrome .ne. 0
c

This subroutine is part of a set of subroutines which perform
a Generalized Hamming Code. As you know, Hamming codes are perfect
codes and can only detect and correct one error. We added an overall
cparity checkbit, which allows us to detect 2 errors. When 2 errors
care detected, (in subroutine decodeham.f) no correction attempt is
cmade. This would most likely result in more errors. Instead, a flag
cis sent to the calling program notifying it of multiple errors so
cthat smoothing may be attempted. The Hamming codes presently supported
by the routines are (63,57), (31,26), (15,11), and shortened variations thereof. It could be made even more general by making minor modifications to the dectobin.f subroutine. This routine at present will calculate a maximum of 6 bits.

Hamming routines consist of the following files:

matrixgen - generates the hmatrix and syndrometable.
dectobin - does a simple decimal to binary conversion.
encodeham - generates the codeword and overall paritybit.
decodeham - recovers infobits, checks for errors, corrects 1 error, and sends out flag for smoothing.

This subroutine, decodeham, is responsible for checking for errors, correcting the error if there is only one, and sending a smoothing flag to the calling routine if there is more than one.

REFERENCES

Lin and Costello: Error Control Coding
Berlekamp: Algebraic Coding Theory

---------------------------------------------------------------

subroutine decodeham(codelength1,codelength2,paritybit,myerror,
  hmatrix,syndrometable,codeword)

integer*4 codelength1,codelength2,hmatrix(codelength1)
integer*4 syndrometable(codelength1),paritybit
  integer*4 codeword(codelength1),myerror
integer*4 errorflag,parityflag
integer*4 synflag, i, j, temp3

myerror=0
errorflag=0
parityflag=0

*** parity flag = 0 if not using the extra parity bit

This part of the routine checks the overall parity of the codeword and compares it with the overall paritybit sent. If they are not the same that means there is at least one error. If, later on in the routine, the syndrome check indicates that there is an error and the parity is
c correct in this part of the routine, that indicates there are two errors.
c One of the weaknesses of this method is that there is no way of knowing
c if we have 3,5,7,... errors. We always smooth if there are 2,4,6,...
c errors.
c if (parityflag.eq.1) then
  synflag=0
  do 10 i=1,codelength1
    synflag= xor(synflag,codeword(i))
  10  continue
  if (paritybit.ne.synflag)errorflag=errcrflag+1
end if
c
 This part of the routine generates the syndrome. The syndrome will
c equal zero if there are no errors. synflag accumulates the syndrome
c and is used as the offset in the syndrome table, which tells the
c routine which bit is in error.
C
  synflag=0
temp3=0
  do 30 i=1,codelength1
    if(codeword(i).ne.0)synflag = xor(synflag,hmatrix(i))
    if(codeword(i).ne.0)temp3 = temp3 + 1
  30  continue

 *** Check to see if the parityflag is set and if it is then check
c to see if the parity bit was in error.
c If the parityflag was set and there was an error in the syndrome,
c the errorflag should equal 1.
c If it doesn't, then there are more errors than can be corrected
and the infobits are passed on unchanged.
c
  if (synflag.ne.0) then
    if((errorflag.ne.1) .and. (parityflag.eq.1))then
      myerror = 1
      go to 20
    end if
    j=syndrometable(synflag)
    codeword(j)=xor(codeword(j),1)
  end if

 *** If the syndrome is equal to zero and the errorflag is set
(c not likely, but must be checked) then more than one error has
c occurred, but it cannot be corrected, so I pass on the infobits
c the same as if there were no errors.

96
20       continue
return
end

This subroutine converts decimal numbers into a binary output vector.

SYNOPSIS
dectobin(vectorsize, decinteger, binaryvector)

formal

data I/O
name type type function

vectorsize int i output vector length
decinteger int i decimal number (< 2^32-1)
binaryvector int o vector containing binary number

This subroutine is part of a set of subroutines which perform a Generalized Hamming Code. As you know, Hamming codes are perfect codes and can only detect and correct one error. We added an overall parity check bit, which allows us to detect 2 errors. When 2 errors are detected, (in subroutine decodeham.f) no correction attempt is made. This would most likely result in more errors. Instead, a flag is sent to the calling program notifying it of multiple errors so that smoothing may be attempted. The Hamming codes presently supported by the routines are (63,57), (31,2^r), (15,11), and shortened variations thereof. It could be made even more general by making minor modifications to the dectobin.f subroutine. This routine at present will calculate a maximum of 6 bits.

Hamming routines consist of the following files:
matrixgen - generates the hmatrix and sydrometable.
dectobin - does a simple decimal to binary conversion.
encodeham - generates the codeword and overall paritybit.
decodeham - recovers infobits, checks for errors, corrects 1
error, and sends out flag for smoothing.

This routine is used by encodeham to convert the packed decinteger
into the Hamming paritybits.

REFERENCES

Lin and Costello: Error Control Coding
Berlekamp: Algebraic Coding Theory

subroutine dectobin(vectorsize, decinteger, binaryvector)
integer*4 vectorsize, decinteger, binaryvector(vectorsize)
integer*4 i, temp1, twostable(6)
data twostable/1,2,4,8,16,32/
c
Check to see if the decimal integer is larger than the routine can
c convert. This can be easily extended by adding to the twostable.
if (decinteger .gt. 63)
   print *, 'dectobin: decinteger too large', decinteger
temp1=vectorsize
do 10 i=1,vectorsize
   if (decinteger .ge. twostable(temp1)) then
      binaryvector(temp1)=1
      decinteger=decinteger-twostable(temp1)
   else
      binaryvector(temp1)=0
   end if
   temp1=temp1-1
10 continue
return
end

ROUTINE
This routine is used to generate the H matrix and
syndrome table necessary for Hamming encode and decode. This
routine should be called once before calling encodeham and
decodeham.

SYNOPSIS
subroutine matrixgen(codelength1,codelength2,
hmatrix,syndrometable)

formal
data I/O
name type type function

- codelength1 int i number of data bits (63)
codelength2 int i number of information bits (57)
hmatrix int o vector to encode an decode by
syndrometable int o table containing error masks

DESCRIPTION
This subroutine is part of a set of subroutines which perform
a Generalized Hamming Code. As you know, Hamming codes are perfect
codes and can only detect and correct one error. We added an overall
c parity checkbit, which allows us to detect 2 errors. When 2 errors
c are detected, (in subroutine decodeham.f) no correction attempt is
c made. This would most likely result in more errors. Instead, a flag
c is sent to the calling program notifying it of multiple errors so
c that smoothing may be attempted. The Hamming codes presently supported
c by the routines are (63,57), (31,26), (15,11), and shortened variations
c thereof. It could be made even more general by making minor modifications
c to the dectobin.f subroutine. This routine at present will calculate
c a maximum of 6 bits.

Hamming routines consist of the following files:

matrixgen - generates the hmatrix and syndrometable.
dectobin - does a simple decimal to binary conversion.
encodeham - generates the codeword and overall paritybit.
c decodeham - recovers infobits, checks for errors, corrects 1
c error, and sends out flag for smoothing.
c
c This routine is initializes all of the tables necessary to perform
c the Hamming code (G Matrix, Syndrome Table).
c
c=================================================================}
c
c REFERENCES
c
c Lin and Costello : Error Control Coding
c Berlekamp : Algebraic Coding Theory
c
c=================================================================

c subroutine matrixgen(codelength1,codelength2,hmatrix,syndrometable)
c
integer*4 codelength1,codelength2
    integer*4 hmatrix(codelength1),syndrometable(codelength1)
integer*4 itemplate(6),ptemplate(57),i,temp1

c This is the data necessary to construct the G Matrix and the Syndrome
Table. If a larger code is desired, this table can be easily added to.
c All other routines, except the syndrome table construction,
c are general enough to calculate any size Hamming Code.
c
data itemplate/1,2,4,8,16,32/
data ptemplate/3,5,6,7,9,10,11,12,13,15,17,18,19,
    + 20,21,22,23,24,25,26,27,28,29,30,31,33,34,35,36,37,38,39,40,41,
    + 42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,
    + 63/
c
c Construct the parity portion of the hmatrix
c
do 30 i=1,codelength2
    hmatrix(i)=ptemplate(i)
30 continue
c
c Construct the identity portion of the hmatrix.
c
do 20 i=1,(codelength1-codelength2)
    hmatrix((codelength2+i))=itemplate(i)
20 continue
c
c Construct the syndrometable. This routine is rather simple because
I chose to arrange my G matrix sequentially (Berlekamp method).
I placed the parity bits in front in ascending order then added the
c bits left over in ascending order. Since our code is linear I can get
c away with this. If a larger Hamming code is needed, then a new
c exception must be generated for each parity bit.

c
temp1=1

do 10 i=1,codelength1
   if(i.eq.1)then
      syndrometable(i)=codelength2+1
      goto 10
   end if
   if(i.eq.2)then
      syndrometable(i)=codelength2+2
      goto 10
   end if
   if(i.eq.4)then
      syndrometable(i)=codelength2+3
      goto 10
   end if
   if(i.eq.8)then
      syndrometable(i)=codelength2+4
      goto 10
   end if
   if(i.eq.16)then
      syndrometable(i)=codelength2+5
      goto 10
   end if
   if(i.eq.32)then
      syndrometable(i)=codelength2+6
      goto 10
   end if
   syndrometable(i)=temp1
   temp1=temp1+1
10 continue

c
return

c
end
### C. DATA INPUT

************** Gaussian Input **************

<p>| | | | | | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>60</td>
<td>100</td>
<td>140</td>
<td>181</td>
<td>221</td>
<td>261</td>
<td>301</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>341</td>
<td>381</td>
<td>421</td>
<td>462</td>
<td>502</td>
<td>542</td>
<td>582</td>
<td>622</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>662</td>
<td>702</td>
<td>743</td>
<td>783</td>
<td>824</td>
<td>864</td>
<td>904</td>
<td>944</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>985</td>
<td>1025</td>
<td>1065</td>
<td>1106</td>
<td>1146</td>
<td>1187</td>
<td>1227</td>
<td>1268</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1309</td>
<td>1350</td>
<td>1390</td>
<td>1431</td>
<td>1472</td>
<td>1513</td>
<td>1553</td>
<td>1594</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1635</td>
<td>1676</td>
<td>1717</td>
<td>1758</td>
<td>1798</td>
<td>1840</td>
<td>1881</td>
<td>1922</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1963</td>
<td>2005</td>
<td>2046</td>
<td>2088</td>
<td>2129</td>
<td>2170</td>
<td>2212</td>
<td>2254</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2296</td>
<td>2337</td>
<td>2379</td>
<td>2421</td>
<td>2463</td>
<td>2505</td>
<td>2546</td>
<td>2589</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2631</td>
<td>2673</td>
<td>2716</td>
<td>2758</td>
<td>2800</td>
<td>2843</td>
<td>2886</td>
<td>2928</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2972</td>
<td>3014</td>
<td>3057</td>
<td>3100</td>
<td>3144</td>
<td>3186</td>
<td>3230</td>
<td>3273</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3316</td>
<td>3360</td>
<td>3404</td>
<td>3448</td>
<td>3492</td>
<td>3535</td>
<td>3579</td>
<td>3624</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3668</td>
<td>3712</td>
<td>3757</td>
<td>3801</td>
<td>3846</td>
<td>3891</td>
<td>3936</td>
<td>3981</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4026</td>
<td>4072</td>
<td>4117</td>
<td>4163</td>
<td>4209</td>
<td>4254</td>
<td>4300</td>
<td>4346</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4392</td>
<td>4439</td>
<td>4485</td>
<td>4532</td>
<td>4579</td>
<td>4626</td>
<td>4673</td>
<td>4720</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4767</td>
<td>4815</td>
<td>4863</td>
<td>4911</td>
<td>4959</td>
<td>5007</td>
<td>5056</td>
<td>5104</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5153</td>
<td>5201</td>
<td>5251</td>
<td>5301</td>
<td>5350</td>
<td>5400</td>
<td>5450</td>
<td>5500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5550</td>
<td>5600</td>
<td>5651</td>
<td>5702</td>
<td>5754</td>
<td>5805</td>
<td>5857</td>
<td>5908</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5961</td>
<td>6013</td>
<td>6066</td>
<td>6118</td>
<td>6171</td>
<td>6225</td>
<td>6279</td>
<td>6333</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6387</td>
<td>6442</td>
<td>6496</td>
<td>6551</td>
<td>6606</td>
<td>6662</td>
<td>6719</td>
<td>6774</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6831</td>
<td>6888</td>
<td>6945</td>
<td>7003</td>
<td>7061</td>
<td>7119</td>
<td>7178</td>
<td>7237</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7296</td>
<td>7356</td>
<td>7417</td>
<td>7477</td>
<td>7538</td>
<td>7599</td>
<td>7661</td>
<td>7724</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7787</td>
<td>7850</td>
<td>7914</td>
<td>7978</td>
<td>8042</td>
<td>8108</td>
<td>8173</td>
<td>8240</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8306</td>
<td>8373</td>
<td>8441</td>
<td>8510</td>
<td>8579</td>
<td>8649</td>
<td>8719</td>
<td>8790</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8862</td>
<td>8935</td>
<td>9007</td>
<td>9081</td>
<td>9156</td>
<td>9231</td>
<td>9306</td>
<td>9383</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9461</td>
<td>9540</td>
<td>9619</td>
<td>9700</td>
<td>9781</td>
<td>9863</td>
<td>9947</td>
<td>10031</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10117</td>
<td>10203</td>
<td>10290</td>
<td>10380</td>
<td>10470</td>
<td>10561</td>
<td>10654</td>
<td>10748</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10843</td>
<td>10941</td>
<td>11039</td>
<td>11140</td>
<td>11241</td>
<td>11345</td>
<td>11451</td>
<td>11558</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11668</td>
<td>11780</td>
<td>11893</td>
<td>12010</td>
<td>12129</td>
<td>12250</td>
<td>12374</td>
<td>12500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12631</td>
<td>12764</td>
<td>12901</td>
<td>13041</td>
<td>13186</td>
<td>13334</td>
<td>13488</td>
<td>13646</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13809</td>
<td>13978</td>
<td>14153</td>
<td>14335</td>
<td>14524</td>
<td>14721</td>
<td>14928</td>
<td>15143</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15371</td>
<td>15610</td>
<td>15864</td>
<td>16134</td>
<td>16423</td>
<td>16734</td>
<td>17071</td>
<td>17439</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17847</td>
<td>18305</td>
<td>18827</td>
<td>19440</td>
<td>20186</td>
<td>21150</td>
<td>22545</td>
<td>25295</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

102
Golay Decoder Input

11 10 362 9 361 329 11593 8 360 328
11592 296 11560 10536 22946 7 359 327 11591 295
11559 10535 16869 263 11527 10503 18563 9479 24198 21889
20014 6 353 326 11590 294 11558 10534 20033 262
11526 10502 17932 9478 24199 15812 21667 230 11494 10470
21965 9446 24200 17506 22916 8422 24201 23141 15425 24173
756 18957 24202 5 357 325 11589 293 11557 10533
16871 261 11525 10501 20929 9477 19844 24145 21827 21859
11493 10469 16873 9445 16874 16875 527 8421 22050 23142
23980 14755 23105 10501 20929 9477 19844 24145 21827 21859
23086 20908 21763 8399 18957 23143 21795 16449 21827 21859
675 7365 12385 23144 21073 22084 19874 24001 16870 23146
16836 723 23147 17900 24197 23145 21731 4 356 324
11588 292 11556 10532 22161 260 11524 10500 18659 9476
19845 15814 24065 228 11492 10468 18691 9444 14401 24173
22918 8420 18755 18877 579 23088 21997 20642 18723 196
11460 10436 23714 9412 16803 15816 22919 8388 23201 15817
21101 15818 18978 494 15819 7364 20015 20993 22921 22085
22922 22923 716 13698 16837 24241 18627 19553 24196 15815
22920 164 11428 10404 23746 9380 19848 22625 18893 8356
19849 22029 23087 19851 620 20706 19850 332 23181 17868
22113 22086 24099 20738 16868 24033 16838 20770 18595 20802
19847 642 20834 6308 23874 23906 738 22087 20961 20016
23842 21027 16839 18817 23810 24269 19846 15813 21635 22089
16840 15779 23778 690 22091 22090 22917 16843 526 23140
16842 22088 16841 20674 17825 3 355 323 11587 291
11555 10531 24012 259 11523 10499 18660 9475 17889 21104
21701 227 11491 10467 16892 9443 23219 17602 20897 8419
18756 18788 580 14757 16770 24271 18724 195 11459 10435
23183 9411 16804 17634 21765 8387 19906 23969 21797 23116
21829 21861 677 7363 12449 17698 24176 17730 18926 546
17762 22031 23085 20940 16828 19585 24195 17666 21733 163
11427 10403 20013 9379 21058 22657 21766 8355 24272 15746
21798 14759 21830 21862 678 7331 12481 24244 22978 14760
24100 20044 16867 14761 21103 17921 18596 461 14763 14762
21734 6307 12513 18958 21800 24175 21832 21864 680 21028
21833 21865 681 21866 682 683 21 12641 385 15780
12609 23184 12577 17570 21767 24130 12545 23139 21799 14758
21831 21863 679 131 11395 10371 18695 9347 16806 22689
19938 8323 18759 18791 583 24226 23182 17836 18727 7299
18760 18792 584 20972 24101 22030 18728 18794 586 587

103
<table>
<thead>
<tr>
<th>23873</th>
<th>23905</th>
<th>737</th>
<th>16644</th>
<th>20962</th>
<th>21925</th>
<th>23841</th>
<th>16676</th>
<th>13507</th>
<th>18818</th>
</tr>
</thead>
<tbody>
<tr>
<td>23809</td>
<td>516</td>
<td>16740</td>
<td>16708</td>
<td>23150</td>
<td>721</td>
<td>23083</td>
<td>23082</td>
<td>23777</td>
<td>23081</td>
</tr>
<tr>
<td>14533</td>
<td>19948</td>
<td>18947</td>
<td>23080</td>
<td>21106</td>
<td>21956</td>
<td>15524</td>
<td>16612</td>
<td>24236</td>
<td>20673</td>
</tr>
<tr>
<td>17826</td>
<td>3137</td>
<td>22026</td>
<td>22027</td>
<td>688</td>
<td>19884</td>
<td>24262</td>
<td>18888</td>
<td>22025</td>
<td>23175</td>
</tr>
<tr>
<td>13476</td>
<td>18899</td>
<td>22024</td>
<td>18990</td>
<td>17890</td>
<td>590</td>
<td>18891</td>
<td>23176</td>
<td>20049</td>
<td>23940</td>
</tr>
<tr>
<td>22023</td>
<td>21989</td>
<td>14467</td>
<td>17601</td>
<td>21039</td>
<td>24265</td>
<td>21093</td>
<td>22022</td>
<td>24267</td>
<td>758</td>
</tr>
<tr>
<td>16769</td>
<td>18887</td>
<td>24165</td>
<td>18916</td>
<td>24265</td>
<td>21093</td>
<td>22022</td>
<td>24267</td>
<td>758</td>
<td>17633</td>
</tr>
<tr>
<td>24266</td>
<td>22060</td>
<td>19905</td>
<td>23970</td>
<td>15587</td>
<td>16547</td>
<td>24264</td>
<td>18886</td>
<td>20868</td>
<td>16845</td>
</tr>
<tr>
<td>12450</td>
<td>17697</td>
<td>15619</td>
<td>17729</td>
<td>24263</td>
<td>545</td>
<td>17761</td>
<td>23174</td>
<td>15683</td>
<td>15715</td>
</tr>
<tr>
<td>483</td>
<td>19586</td>
<td>22093</td>
<td>17656</td>
<td>15651</td>
<td>24110</td>
<td>13572</td>
<td>21094</td>
<td>22021</td>
<td>21991</td>
</tr>
<tr>
<td>21057</td>
<td>22658</td>
<td>17795</td>
<td>13668</td>
<td>420</td>
<td>15745</td>
<td>13636</td>
<td>16579</td>
<td>13604</td>
<td>18885</td>
</tr>
<tr>
<td>24167</td>
<td>21993</td>
<td>12482</td>
<td>18851</td>
<td>22977</td>
<td>687</td>
<td>21995</td>
<td>21994</td>
<td>24168</td>
<td>23173</td>
</tr>
<tr>
<td>13540</td>
<td>17922</td>
<td>24169</td>
<td>21992</td>
<td>24170</td>
<td>24171</td>
<td>755</td>
<td>21098</td>
<td>12514</td>
<td>659</td>
</tr>
<tr>
<td>21099</td>
<td>16643</td>
<td>24261</td>
<td>21097</td>
<td>15821</td>
<td>16675</td>
<td>13508</td>
<td>21096</td>
<td>23121</td>
<td>515</td>
</tr>
<tr>
<td>16739</td>
<td>16707</td>
<td>21569</td>
<td>12642</td>
<td>386</td>
<td>21095</td>
<td>12610</td>
<td>21990</td>
<td>12578</td>
<td>17569</td>
</tr>
<tr>
<td>18948</td>
<td>24129</td>
<td>12546</td>
<td>21956</td>
<td>15523</td>
<td>16611</td>
<td>21038</td>
<td>22956</td>
<td>24166</td>
<td>18918</td>
</tr>
<tr>
<td>13573</td>
<td>23943</td>
<td>22020</td>
<td>21040</td>
<td>14563</td>
<td>22690</td>
<td>19937</td>
<td>13669</td>
<td>421</td>
<td>20003</td>
</tr>
<tr>
<td>13637</td>
<td>24225</td>
<td>13605</td>
<td>18884</td>
<td>20870</td>
<td>23946</td>
<td>14627</td>
<td>748</td>
<td>23947</td>
<td>14691</td>
</tr>
<tr>
<td>451</td>
<td>23945</td>
<td>14659</td>
<td>23172</td>
<td>13541</td>
<td>23944</td>
<td>18497</td>
<td>19650</td>
<td>14595</td>
<td>16877</td>
</tr>
<tr>
<td>23217</td>
<td>591</td>
<td>18923</td>
<td>18922</td>
<td>17858</td>
<td>18921</td>
<td>24260</td>
<td>21923</td>
<td>20872</td>
<td>18920</td>
</tr>
<tr>
<td>13509</td>
<td>23041</td>
<td>20873</td>
<td>19682</td>
<td>20874</td>
<td>20875</td>
<td>652</td>
<td>18919</td>
<td>22145</td>
<td>23942</td>
</tr>
<tr>
<td>23149</td>
<td>19714</td>
<td>14531</td>
<td>17537</td>
<td>18949</td>
<td>19746</td>
<td>24112</td>
<td>21957</td>
<td>15491</td>
<td>610</td>
</tr>
<tr>
<td>19810</td>
<td>19778</td>
<td>20871</td>
<td>13672</td>
<td>424</td>
<td>22818</td>
<td>13640</td>
<td>22850</td>
<td>13608</td>
<td>706</td>
</tr>
<tr>
<td>22882</td>
<td>427</td>
<td>13</td>
<td>13674</td>
<td>426</td>
<td>13673</td>
<td>425</td>
<td>22786</td>
<td>13641</td>
<td>19969</td>
</tr>
<tr>
<td>13575</td>
<td>23941</td>
<td>21039</td>
<td>21988</td>
<td>14499</td>
<td>22754</td>
<td>18950</td>
<td>13671</td>
<td>423</td>
<td>21958</td>
</tr>
<tr>
<td>13639</td>
<td>18988</td>
<td>13607</td>
<td>20577</td>
<td>24164</td>
<td>18917</td>
<td>13574</td>
<td>21092</td>
<td>23649</td>
<td>14721</td>
</tr>
<tr>
<td>22129</td>
<td>22722</td>
<td>18951</td>
<td>13670</td>
<td>422</td>
<td>21959</td>
<td>13638</td>
<td>16515</td>
<td>13606</td>
<td>24111</td>
</tr>
<tr>
<td>20869</td>
<td>23075</td>
<td>12418</td>
<td>21960</td>
<td>18953</td>
<td>24205</td>
<td>18954</td>
<td>18955</td>
<td>592</td>
<td>21962</td>
</tr>
<tr>
<td>13542</td>
<td>686</td>
<td>21963</td>
<td>19618</td>
<td>23009</td>
<td>21961</td>
<td>18952</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

107
APPENDIX B: Simulation Data

A. INMARSAT Channel

************* QRC *************

# errors corrected = 2
there were no errors in QRC cw
# errors corrected = 2
success = 763
you just completed superframe # 999
# errors corrected = 2
# errors corrected = 4
fail = 236
total errors in sf = 8
number of 10ms fades in 1000 runs = 1842
number of 20ms fades in 1000 runs = 254
number of 40ms fades in 1000 runs = 103
number of 100ms fades in 1000 runs = 87
number of 200ms fades in 1000 runs = 24
you just completed superframe # 1000
iallerr = 3955 itotbitct = 143856 n = 1000
the total number of failures = 236
the total number of successes = 763
This was a nogolay run
This was a QRC BE count run
This run used fading (1=yes, 0=no) 1
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 99.0000

************* Soft Decision Golay *************

success = 746
you just completed superframe # 999
success = 747
number of 10ms fades in 1000 runs = 1909
number of 20ms fades in 1000 runs = 239
number of 40ms fades in 1000 runs = 129
number of 100ms fades in 1000 runs = 86
number of 200ms fades in 1000 runs = 32
you just completed superframe # 1000
the total number of failures = 252
the total number of successes = 747
This was a soft golay run
This run used fading (1=yes, 0=no)  1
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) =  99.0000
B. Gaussian Channel

*************** No Code ***************

errors in 100 bits = 18
errors in 3000 bits = 337
errors in 3000 bits = 352
errors in 3000 bits = 364
errors in 3000 bits = 367
you just completed superframe # 200
the total number of failures = 198
the total number of successes = 0
This was a nogolay run
This run used fading (1=yes, 0=no) 0
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 3.00000

errors in 100 bits = 11
errors in 3000 bits = 249
errors in 3000 bits = 268
errors in 3000 bits = 265
errors in 3000 bits = 262
you just completed superframe # 200
the total number of failures = 196
the total number of successes = 1
This was a nogolay run
This run used fading (1=yes, 0=no) 0
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 4.00000

errors in 100 bits = 7
errors in 3000 bits = 179
errors in 3000 bits = 182
errors in 3000 bits = 175
errors in 3000 bits = 176
you just completed superframe # 200
the total number of failures = 197
the total number of successes = 2
This was a nogolay run
This run used fading (1=yes, 0=no) 0
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 5.00000
errors in 100 bits = 4
errors in 3000 bits = 125
errors in 3000 bits = 128
errors in 3000 bits = 103
errors in 3000 bits = 116
you just completed superframe # 200
the total number of failures = 192
the total number of successes = 7
This was a nogolay run
This run used fading (i=yes,0=no) 0
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 6.00000
********** Hard Decision Golay **********

errors in 100 bits = 14
errors in 3000 bits = 240
errors in 3000 bits = 277
errors in 3000 bits = 268
errors in 3000 bits = 278

you just completed superframe # 200
the total number of failures = 171
the total number of successes = 26
This was a hard golay run
This run used fading (1=yes,0=no) 0
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 3.00000

errors in 100 bits = 1
errors in 3000 bits = 111
errors in 3000 bits = 146
errors in 3000 bits = 145
errors in 3000 bits = 136

you just completed superframe # 200
the total number of failures = 123
the total number of successes = 72
This was a hard golay run
This run used fading (1=yes,0=no) 0
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 4.00000

errors in 100 bits = 0
errors in 3000 bits = 41
errors in 3000 bits = 52
errors in 3000 bits = 48
errors in 3000 bits = 36

you just completed superframe # 200
the total number of failures = 57
the total number of successes = 140
This was a hard golay run
This run used fading (1=yes,0=no) 0
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 5.00000

112
errors in 100 bits = 0
errors in 3000 bits = 5
errors in 3000 bits = 0
errors in 3000 bits = 4
errors in 3000 bits = 7

you just completed superframe # 200
the total number of failures = 13
the total number of successes = 186
This was a hard golay run
This run used fading (1=yes, 0=no) 0
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 6.00000

113
********** Soft Decision Golay ******

errors in 100 bits = 10
errors in 3000 bits = 146
errors in 3000 bits = 192
errors in 3000 bits = 172
errors in 3000 bits = 182
you just completed superframe # 200
the total number of failures = 135
the total number of successes = 62
This was a soft golay run
This run used fading (1=yes,0=no) 0
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 3.00000

errors in 100 bits = 0
errors in 3000 bits = 62
errors in 3000 bits = 75
errors in 3000 bits = 88
errors in 3000 bits = 68
you just completed superframe # 200
the total number of failures = 77
the total number of successes = 122
This was a soft golay run
This run used fading (1=yes,0=no) 0
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 4.00000

errors in 100 bits = 0
errors in 3000 bits = 18
errors in 3000 bits = 13
errors in 3000 bits = 24
errors in 3000 bits = 10
you just completed superframe # 200
the total number of failures = 24
the total number of successes = 175
This was a soft golay run
This run used fading (1=yes,0=no) 0
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 5.00000
<table>
<thead>
<tr>
<th>Errors in bits</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 bits</td>
<td>0</td>
</tr>
<tr>
<td>3000 bits</td>
<td>0</td>
</tr>
<tr>
<td>3000 bits</td>
<td>0</td>
</tr>
<tr>
<td>3000 bits</td>
<td>7</td>
</tr>
</tbody>
</table>

You just completed superframe # 200
the total number of failures = 5
the total number of successes = 194
This was a soft golay run
This run used fading (1=yes, 0=no) 0

s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 6.00000
C. Constant Burst Width Channel

KEY:

Fade widths (120, 240, 480, 1200, 2400) correspond to burst widths (10, 20, 40, 100, 200) milliseconds.

************ No Code ************

errors in 100 bits = 4
errors in 3000 bits = 101
errors in 3000 bits = 95
errors in 3000 bits = 86
errors in 3000 bits = 70
you just completed superframe # 200
the total number of failures = 106
the total number of successes = 93
This was a nogolay run
This run used fading (1=yes, 0=no) 1
fading width, bwidth = 120
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 99.0000

errors in 100 bits = 7
errors in 3000 bits = 178
errors in 3000 bits = 190
errors in 3000 bits = 188
errors in 3000 bits = 102
you just completed superframe # 200
the total number of failures = 114
the total number of successes = 85
This was a nogolay run
This run used fading (1=yes, 0=no) 1
fading width, bwidth = 240
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 99.0000

errors in 100 bits = 0
errors in 3000 bits = 283
errors in 3000 bits = 265
errors in 3000 bits = 324
errors in 3000 bits = 218
you just completed superframe # 200

116
the total number of failures = 123
the total number of successes = 74
This was a nogolay run
This run used fading (1=yes,0=no) 1
fading width, bwidth = 480
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 99.0000

errors in 100 bits = 29
errors in 3000 bits = 472
errors in 3000 bits = 606
errors in 3000 bits = 482
errors in 3000 bits = 462
you just completed superframe # 200
the total number of failures = 135
the total number of successes = 62
This was a nogolay run
This run used fading (1=yes,0=no) 1
fading width, bwidth = 1200
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 99.0000

errors in 100 bits = 26
errors in 3000 bits = 798
errors in 3000 bits = 816
errors in 3000 bits = 852
errors in 3000 bits = 883
you just completed superframe # 200
the total number of failures = 156
the total number of successes = 39
This was a nogolay run
This run used fading (1=yes,0=no) 1
fading width, bwidth = 2400
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 99.0000
************* Hard Decision Golay *************

errors in 100 bits = 0
errors in 3000 bits = 10
errors in 3000 bits = 13
errors in 3000 bits = 7
errors in 3000 bits = 13

you just completed superframe # 200
the total number of failures = 11
the total number of successes = 188

This was a hard golay run
This run used fading (I=yes,0=no) 1
fading width, bwidth = 120
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 99.0000

errors in 100 bits = 0
errors in 3000 bits = 25
errors in 3000 bits = 40
errors in 3000 bits = 58
errors in 3000 bits = 18

you just completed superframe # 200
the total number of failures = 41
the total number of successes = 158

This was a hard golay run
This run used fading (I=yes,0=no) 1
fading width, bwidth = 240
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 99.0000

errors in 100 bits = 0
errors in 3000 bits = 270
errors in 3000 bits = 257
errors in 3000 bits = 234
errors in 3000 bits = 305

you just completed superframe # 200
the total number of failures = 102
the total number of successes = 97

This was a hard golay run
This run used fading (I=yes,0=no) 1
fading width, bwidth = 480
s/n during fade (dB) = -24.0000

118
s/n during non-fade (dB) = 99.0000

errors in 100 bits = 0
errors in 3000 bits = 698
errors in 3000 bits = 543
errors in 3000 bits = 733
errors in 3000 bits = 616

you just completed superframe # 200
the total number of failures = 151
the total number of successes = 48
This was a hard golay run
This run used fading (1=yes,0=no) 1
fading width, bwidth = 1200
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 99.0000

errors in 100 bits = 14
errors in 3000 bits = 739
errors in 3000 bits = 972
errors in 3000 bits = 986

you just completed superframe # 200
the total number of failures = 133
the total number of successes = 19
This was a hard golay run
This run used fading (1=yes,0=no) 1
fading width, bwidth = 2400
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 99.0000
*********** Soft Decision Golay ***********

errors in 100 bits = 0
errors in 3000 bits = 0
errors in 3000 bits = 0
errors in 3000 bits = 0
errors in 3000 bits = 0
you just completed superframe # 200
the total number of failures = 2
the total number of successes = 197
This was a soft golay run
This run used fading (1=yes,0=no) 1
fading width, bwidth = 120
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 99.0000

errors in 100 bits = 0
errors in 3000 bits = 0
errors in 3000 bits = 76
errors in 3000 bits = 10
errors in 3000 bits = 26
you just completed superframe # 200
the total number of failures = 21
the total number of successes = 178
This was a soft golay run
This run used fading (1=yes,0=no) 1
fading width, bwidth = 240
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 99.0000

errors in 100 bits = 0
errors in 3000 bits = 111
errors in 3000 bits = 137
errors in 3000 bits = 133
errors in 3000 bits = 153
you just completed superframe # 200
the total number of failures = 60
the total number of successes = 139
This was a soft golay run
This run used fading (1=yes,0=no) 1
fading width, bwidth = 480
s/n during fade (dB) = -24.0000

120
s/n during non-fade (dB) = 99.0000

errors in 100 bits = 0
errors in 3000 bits = 412
errors in 3000 bits = 521
errors in 3000 bits = 647
errors in 3000 bits = 509
you just completed superframe # 200
the total number of failures = 133
the total number of successes = 66
This was a soft golay run
This run used fading (1=yes, 0=no) 1
fading width, bwidth = 1200
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 99.0000

errors in 100 bits = 8
errors in 3000 bits = 769
errors in 3000 bits = 711
errors in 3000 bits = 976
errors in 3000 bits = 757
you just completed superframe # 200
the total number of failures = 160
the total number of successes = 39
This was a soft golay run
This run used fading (1=yes, 0=no) 1
fading width, bwidth = 2400
s/n during fade (dB) = -24.0000
s/n during non-fade (dB) = 99.0000
Hamming Codeword Interleaving Table

<table>
<thead>
<tr>
<th>CW1</th>
<th>CW2</th>
<th>CW3</th>
<th>CW4</th>
<th>CW5</th>
<th>CW6</th>
<th>CW7</th>
<th>CW8</th>
<th>CW9</th>
</tr>
</thead>
<tbody>
<tr>
<td>721</td>
<td>731</td>
<td>751</td>
<td>761</td>
<td>781</td>
<td>791</td>
<td>811</td>
<td>821</td>
<td>841</td>
</tr>
<tr>
<td>991</td>
<td>1001</td>
<td>1021</td>
<td>1031</td>
<td>1051</td>
<td>1061</td>
<td>1081</td>
<td>1091</td>
<td>1111</td>
</tr>
<tr>
<td>1261</td>
<td>1271</td>
<td>1291</td>
<td>1301</td>
<td>1321</td>
<td>1331</td>
<td>1351</td>
<td>1361</td>
<td>1381</td>
</tr>
<tr>
<td>1531</td>
<td>1541</td>
<td>1561</td>
<td>1571</td>
<td>1591</td>
<td>1601</td>
<td>1621</td>
<td>1631</td>
<td>1651</td>
</tr>
<tr>
<td>1801</td>
<td>1811</td>
<td>1831</td>
<td>1841</td>
<td>1861</td>
<td>1871</td>
<td>1891</td>
<td>1901</td>
<td>1921</td>
</tr>
<tr>
<td>2071</td>
<td>2081</td>
<td>2101</td>
<td>2111</td>
<td>2131</td>
<td>2141</td>
<td>2161</td>
<td>2171</td>
<td>2191</td>
</tr>
<tr>
<td>2341</td>
<td>2351</td>
<td>2371</td>
<td>2381</td>
<td>2401</td>
<td>2411</td>
<td>2431</td>
<td>2441</td>
<td>2461</td>
</tr>
<tr>
<td>2611</td>
<td>2621</td>
<td>2641</td>
<td>2651</td>
<td>2671</td>
<td>2681</td>
<td>2701</td>
<td>2711</td>
<td>2731</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CW10</th>
<th>CW11</th>
<th>CW12</th>
<th>CW13</th>
<th>CW14</th>
<th>CW15</th>
<th>CW16</th>
<th>CW17</th>
<th>CW18</th>
</tr>
</thead>
<tbody>
<tr>
<td>851</td>
<td>871</td>
<td>881</td>
<td>901</td>
<td>911</td>
<td>931</td>
<td>941</td>
<td>961</td>
<td>971</td>
</tr>
<tr>
<td>1121</td>
<td>1141</td>
<td>1151</td>
<td>1171</td>
<td>1181</td>
<td>1201</td>
<td>1211</td>
<td>1231</td>
<td>1241</td>
</tr>
<tr>
<td>1391</td>
<td>1411</td>
<td>1421</td>
<td>1441</td>
<td>1451</td>
<td>1471</td>
<td>1481</td>
<td>1501</td>
<td>1511</td>
</tr>
<tr>
<td>1661</td>
<td>1681</td>
<td>1691</td>
<td>1711</td>
<td>1721</td>
<td>1741</td>
<td>1751</td>
<td>1771</td>
<td>1781</td>
</tr>
<tr>
<td>2201</td>
<td>2221</td>
<td>2231</td>
<td>2251</td>
<td>2261</td>
<td>2281</td>
<td>2291</td>
<td>2311</td>
<td>2321</td>
</tr>
<tr>
<td>2471</td>
<td>2491</td>
<td>2501</td>
<td>2521</td>
<td>2531</td>
<td>2551</td>
<td>2561</td>
<td>2581</td>
<td>2591</td>
</tr>
<tr>
<td>2741</td>
<td>2761</td>
<td>2771</td>
<td>2791</td>
<td>2801</td>
<td>2821</td>
<td>2831</td>
<td>2851</td>
<td>2861</td>
</tr>
</tbody>
</table>

122
Golay Codeword Interleaving Table

<table>
<thead>
<tr>
<th>CW1</th>
<th>CW2</th>
<th>CW3</th>
<th>CW4</th>
<th>CW5</th>
<th>CW6</th>
</tr>
</thead>
<tbody>
<tr>
<td>721</td>
<td>751</td>
<td>781</td>
<td>811</td>
<td>841</td>
<td>871</td>
</tr>
<tr>
<td>731</td>
<td>761</td>
<td>791</td>
<td>821</td>
<td>851</td>
<td>881</td>
</tr>
<tr>
<td>901</td>
<td>931</td>
<td>961</td>
<td>991</td>
<td>1021</td>
<td>1051</td>
</tr>
<tr>
<td>911</td>
<td>941</td>
<td>971</td>
<td>1001</td>
<td>1031</td>
<td>1061</td>
</tr>
<tr>
<td>1081</td>
<td>1111</td>
<td>1141</td>
<td>1171</td>
<td>1201</td>
<td>1231</td>
</tr>
<tr>
<td>1091</td>
<td>1121</td>
<td>1151</td>
<td>1181</td>
<td>1211</td>
<td>1241</td>
</tr>
<tr>
<td>1261</td>
<td>1291</td>
<td>1321</td>
<td>1351</td>
<td>1381</td>
<td>1411</td>
</tr>
<tr>
<td>1271</td>
<td>1301</td>
<td>1331</td>
<td>1361</td>
<td>1391</td>
<td>1421</td>
</tr>
<tr>
<td>1441</td>
<td>1471</td>
<td>1501</td>
<td>1531</td>
<td>1561</td>
<td>1591</td>
</tr>
<tr>
<td>1451</td>
<td>1481</td>
<td>1511</td>
<td>1541</td>
<td>1571</td>
<td>1601</td>
</tr>
<tr>
<td>1621</td>
<td>1651</td>
<td>1681</td>
<td>1711</td>
<td>1741</td>
<td>1771</td>
</tr>
<tr>
<td>1631</td>
<td>1661</td>
<td>1691</td>
<td>1721</td>
<td>1751</td>
<td>1781</td>
</tr>
<tr>
<td>1801</td>
<td>1831</td>
<td>1861</td>
<td>1891</td>
<td>1921</td>
<td>1951</td>
</tr>
<tr>
<td>1811</td>
<td>1841</td>
<td>1871</td>
<td>1901</td>
<td>1931</td>
<td>1961</td>
</tr>
<tr>
<td>1981</td>
<td>2011</td>
<td>2041</td>
<td>2071</td>
<td>2101</td>
<td>2131</td>
</tr>
<tr>
<td>1991</td>
<td>2021</td>
<td>2051</td>
<td>2081</td>
<td>2111</td>
<td>2141</td>
</tr>
<tr>
<td>2161</td>
<td>2191</td>
<td>2221</td>
<td>2251</td>
<td>2281</td>
<td>2311</td>
</tr>
<tr>
<td>2171</td>
<td>2201</td>
<td>2231</td>
<td>2261</td>
<td>2291</td>
<td>2321</td>
</tr>
<tr>
<td>2341</td>
<td>2371</td>
<td>2401</td>
<td>2431</td>
<td>2461</td>
<td>2491</td>
</tr>
<tr>
<td>2351</td>
<td>2381</td>
<td>2411</td>
<td>2441</td>
<td>2471</td>
<td>2501</td>
</tr>
<tr>
<td>2521</td>
<td>2551</td>
<td>2581</td>
<td>2611</td>
<td>2641</td>
<td>2671</td>
</tr>
<tr>
<td>2531</td>
<td>2561</td>
<td>2591</td>
<td>2621</td>
<td>2651</td>
<td>2681</td>
</tr>
<tr>
<td>2701</td>
<td>2731</td>
<td>2761</td>
<td>2791</td>
<td>2821</td>
<td>2851</td>
</tr>
<tr>
<td>2711</td>
<td>2741</td>
<td>2771</td>
<td>2801</td>
<td>2831</td>
<td>2861</td>
</tr>
</tbody>
</table>
## QRC Codeword Interleaving Table

<table>
<thead>
<tr>
<th>CW1</th>
<th>CW2</th>
<th>CW3</th>
</tr>
</thead>
<tbody>
<tr>
<td>721</td>
<td>781</td>
<td>841</td>
</tr>
<tr>
<td>731</td>
<td>791</td>
<td>851</td>
</tr>
<tr>
<td>751</td>
<td>811</td>
<td>871</td>
</tr>
<tr>
<td>761</td>
<td>821</td>
<td>881</td>
</tr>
<tr>
<td>1081</td>
<td>1141</td>
<td>1201</td>
</tr>
<tr>
<td>1091</td>
<td>1151</td>
<td>1211</td>
</tr>
<tr>
<td>1111</td>
<td>1171</td>
<td>1231</td>
</tr>
<tr>
<td>1121</td>
<td>1181</td>
<td>1241</td>
</tr>
<tr>
<td>1441</td>
<td>1501</td>
<td>1561</td>
</tr>
<tr>
<td>1451</td>
<td>1511</td>
<td>1571</td>
</tr>
<tr>
<td>1471</td>
<td>1531</td>
<td>1591</td>
</tr>
<tr>
<td>1481</td>
<td>1541</td>
<td>1601</td>
</tr>
<tr>
<td>1801</td>
<td>1861</td>
<td>1921</td>
</tr>
<tr>
<td>1811</td>
<td>1871</td>
<td>1931</td>
</tr>
<tr>
<td>1831</td>
<td>1891</td>
<td>1951</td>
</tr>
<tr>
<td>1841</td>
<td>1901</td>
<td>1961</td>
</tr>
<tr>
<td>2161</td>
<td>2221</td>
<td>2281</td>
</tr>
<tr>
<td>2171</td>
<td>2231</td>
<td>2291</td>
</tr>
<tr>
<td>2191</td>
<td>2311</td>
<td>2311</td>
</tr>
<tr>
<td>2201</td>
<td>2311</td>
<td>2321</td>
</tr>
<tr>
<td>2521</td>
<td>2581</td>
<td>2641</td>
</tr>
<tr>
<td>2531</td>
<td>2591</td>
<td>2651</td>
</tr>
<tr>
<td>2551</td>
<td>2611</td>
<td>2671</td>
</tr>
<tr>
<td>2561</td>
<td>2621</td>
<td>2681</td>
</tr>
</tbody>
</table>
LIST OF REFERENCES


2. Rahikka, D.J., Tremain, T.E., Welch, and V.C., Cambell, J.P. Jr., “CELP Coding For LMR Applications”, paper to be presented at the International Conference on Acoustics, Speech and Signal Processing (ICASSP) in Albuquerque, NM on 4-6 April 1990 (no page numbers).


<table>
<thead>
<tr>
<th>No.</th>
<th>Copies</th>
<th>Distribution List</th>
</tr>
</thead>
</table>
| 1.  | 2      | Defense Technical Information Center  
                  Cameron Station  
                  Alexandria, VA  22304-6145 |
| 2.  | 2      | Library, Code 0142  
                  Naval Postgraduate School  
                  Monterey, CA  93943-5002 |
| 3.  | 1      | Director  
                  National Security Agency  
                  Attn: R556, Mr. Doug Rahikka  
                  9800 Savage Rd  
                  Ft. George G. Meade, MD  20755-6000 |
| 4.  | 1      | Director  
                  National Security Agency  
                  Attn: R556, Mr. John Lee  
                  9800 Savage Rd  
                  Ft. George G. Meade, MD  20755-6000 |
| 5.  | 1      | Director  
                  National Security Agency  
                  Attn: R55, Mr. Thomas Tremain  
                  9800 Savage Rd  
                  Ft. George G. Meade, MD  20755-6000 |
| 6.  | 1      | Director  
                  National Security Agency  
                  Attn: V236, Mr. Doug Stonebarger  
                  9800 Savage Rd  
                  Ft. George G. Meade, MD  20755-6000 |
| 7.  | 1      | Director  
                  National Security Agency  
                  Attn: V23, Mr. Alton Crawley  
                  9800 Savage Rd  
                  Ft. George G. Meade, MD  20755-6000 |
8. Director
National Security Agency
Attn: V2, Mr. Dale Learn
9800 Savage Rd
Ft. George G. Meade, MD 20755-6000

9. National Communications System
Attn: Mr. Robert M. Fenichel
Washington, DC 20305-2010

10. Alan H. Levesque
GTE Government Systems Corporation
100 First Avenue, N S EDCD-33
Waltham, MA 02254-1191

11. Professor T.A. Schwendtner, Code EC SC
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5000

12. Curricular Office, Code 39
Naval Postgraduate School
Monterey, CA 93943-5000

13. C³ Academic Group
Attn: Professor Carl Jones, Code CC
Naval Postgraduate School
Monterey, CA 93943-5000

14. Professor Herschal Loomis, Code EC LM
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5000

15. Carol A. Lohrmann
517 N. Chapelgate Lane
Baltimore, MD 21229