HF ALE
SIMULATOR

August 1991

produced by
Johnson Research
Las Cruces, NM

sponsored by
Institute for Telecommunication Sciences
National Telecommunications and Information Administration
Boulder, CO

HF ALE SIMULATOR

August 1991

produced by
Johnson Research
Las Cruces, NM

sponsored by
Institute for Telecommunication Sciences
National Telecommunications and Information Administration
Boulder, CO

HF Simulator
FEC and ALE Modules

Eric E. Johnson
Johnson Research
Las Cruces, NM

6 August 1991

1 INTRODUCTION

This manual describes the design and use of a software simulator package for HF radios employing FED-STD-1045/MIL-STD-188-141A Automatic Link Establishment (ALE). This package employs the simulators for narrow-band HF channels and the standard ALE modem described elsewhere[1]. This software was developed by Johnson Research under the sponsorship of the Institute for Telecommunication Sciences, National Telecommunications and Information Administration, U.S. Department of Commerce, Boulder, Colorado. The initial work leading to the development of these simulators was sponsored by the U.S. Army Information Systems Engineering Command, Ft. Huachuca, Arizona.

The simulators described here are found at a higher level in the ISO OSI model than the channel and modem simulators, as seen in Figure 1. Due to the modular nature of this simulator package, the Linking Protection (LP) simulator can be incorporated with only minor changes in the other modules.

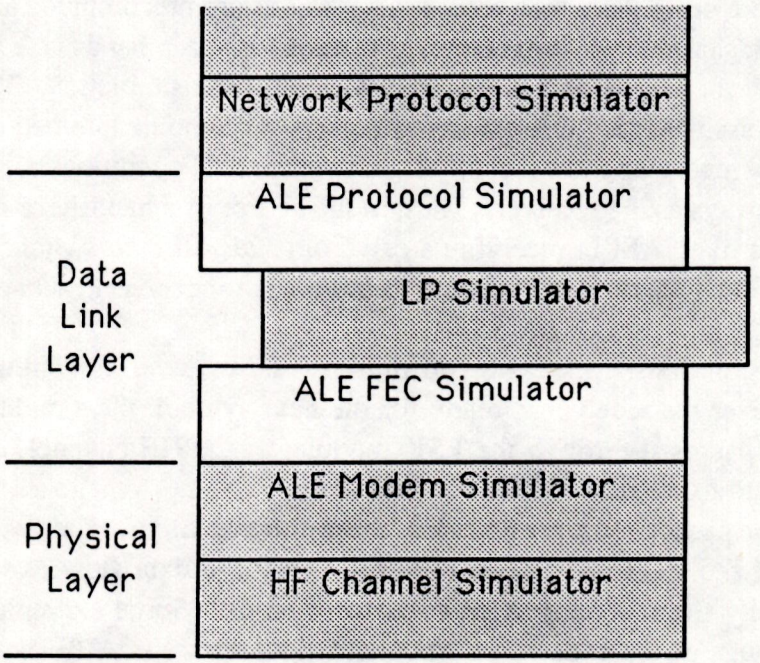


Figure 1: Simulator Stack

1 E.E. Johnson, "HF Simulator: Channel and Modem Modules," Johnson Research, Las Cruces, NM, 2 July 1991.

This simulator package includes the following features of the ALE standard:

- An address data base for self, net, and other addresses.
- Automatic measurement of a pseudo bit error ratio (PBER) for each successfully-received transmission.
- An LQA (Link Quality Analysis) matrix which records the PBER of channels between the simulator and other stations, including both measured PBER from received transmissions, and the PBER in the other direction, as reported by the other stations.
- Individual and net calls.
- LQA handshakes.
- Automatic Message Display (AMD).

The implementation of the simulator described in this report is limited in the following dimensions:

- Only single-word addresses (three characters) are supported.
- Anycalls, allcalls, and group calls are not supported.
- Data text mode (DTM) and data block mode (DBM) protocols are not supported.
- Sounds are correctly interpreted, but do not update the LQA matrix and are not sent.

The overall operating concept of this simulator package is to simulate in detail those operations and decisions made by an actual ALE radio which could have a first-order effect on the performance of the network. Other functions are either "hard coded" into the simulator or precomputed during the initialization phase. This leads to a design in which most transmissions are precomputed and then used repeatedly for each simulated handshake, while the receiver is reset before each handshake and reacts to whatever it receives, regardless of the type of transmissions selected for the simulation. The exception to the rule of precomputed transmissions is LQA reports, which must be computed immediately prior to transmission so that they correctly report the most recently measured channel conditions.

The role of the simulator is always as the receiver. Thus, in the course of a handshake between JOE and SAM, the simulator first acts like SAM in receiving a call from JOE. It then switches to become JOE to receive the response from SAM, and finally becomes SAM again to receive an acknowledgement from JOE.

Given the precomputed transmissions, it is easiest to think of the receiver as "pulling" symbols through the HF channel. Whenever the receiver is ready for the next symbol, the simulator finds the next tone to be transmitted and passes it through the FSK modulator, the HF channel, and the FSK demodulator, and delivers the result to the receive FEC module.

The modules of the simulator package are presented in the following order: first we discuss the operation and data structures of the transmit side. Then, following the data flow, we discuss the receiving FEC module and finally the receiving ALE protocol module. Some example results are presented next, followed by listings of the code in an Appendix (including the ALE modem and HF channel simulators, for reference).

2 SIMULATED TRANSMISSIONS

The simulation of ALE transmissions occurs in two phases:

1. precomputation of the frames to be transmitted, and creation of a data structure which supports efficient retrieval of the precomputed symbols (handshake creation)
2. retrieval of the symbols, including modulation of the transmitted power level to reflect periods when no transmitter is operating (handshake sequencing).

We first discuss the data structure, followed by examination of the functions which create it and read it. The heading for each function or data structure includes the file and page number of relevant C code.

2.1 Frames Data Structure — ale.c (L-1)

Each simulated handshake consists of a sequence of ALE frames, one per transmission. Each frame consists of several phases: for example, a scanning call contains a scanning call of length T_{sc}, a leading call of length T_{lc}, zero or more commands in a message section, and a frame termination of length T_x. The `frames` data structure shown in Figure 2 parallels this hierarchical structure.

Each phase of a frame is represented by a `WordRec`, a structure having the four fields shown in Figure 2: a `next` field which points to the WordRec for the next phase, a `Repetitions` field which indicates the number of times the ALE word for that phase is to be repeated on the air, a `Word` field which points to a pre-encoded array of 49 symbols for the associated ALE word, and a `TxLevel` field which sets the SNR of that word. In Figure 2, we see a simplified version of a scanning call, with $T_{sc} = 20\ T_{rw}$, $T_{lc} = 2\ T_{rw}$, and $T_x = 1\ T_{rw}$. Note that pre-encoded words, and even WordRecs, may be shared among frames. For example, the acknowledgement in this handshake (frame 2) can reuse the T_{lc} and T_x from the scanning call, as shown. The other variables shown are discussed later.

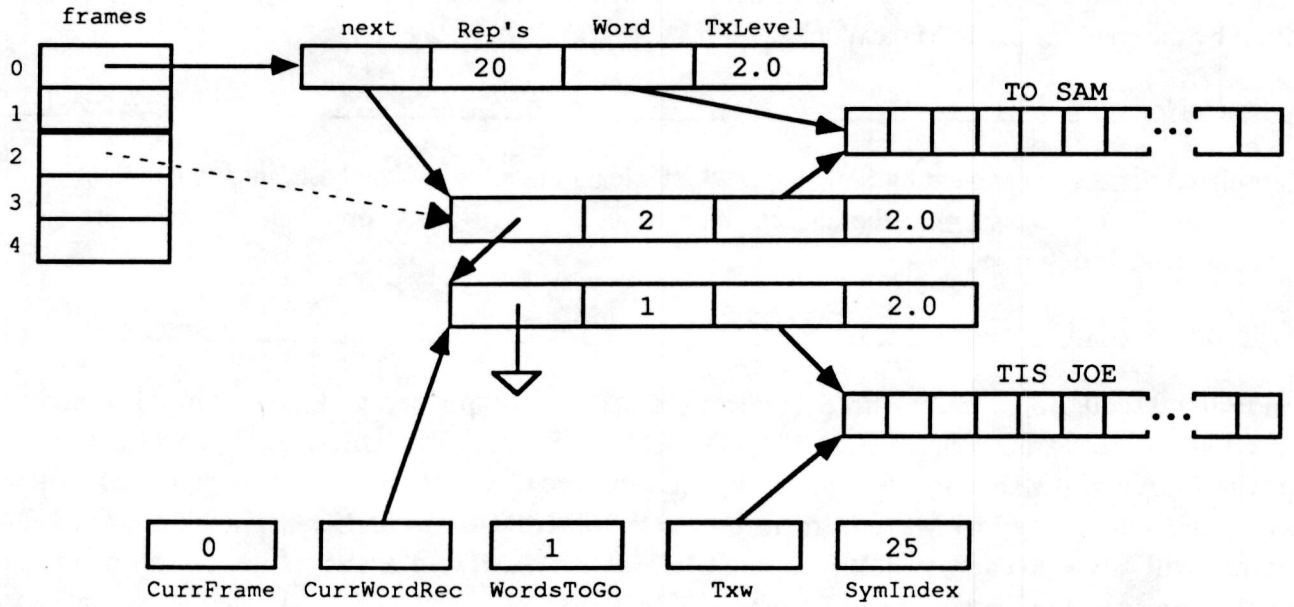


Figure 2: Frames Data Structure

2.2 Handshake Creation

2.2.1 Tx (Transmit FEC) — fec.c (L–22)

Creation of the pre-encoded ALE words used in the frames described above is accomplished by the Tx function. Given a 24-bit ALE word and a pointer to an array of 49 bytes where the tones produced are to be stored, Tx encodes the two halves of the ALE word to produce two 24-bit Golay words in G[0] and G[1], interleaves these Golay words and adds a stuff bit for tone diversity to produce a 49-bit word, and stores the 49-bit word three times as 49 three-bit symbols in the Txw array. (If the pointer Txw passed to Tx is NULL, a 49-byte array is allocated).

2.2.2 Golay Encoder — fec.c (L-21) golay.c (L–26)

Two Golay encode routines are used. The first, `Golay` (in `fec.c`; see page L-21), breaks the 24-bit ALE word into two 12-bit halves, and calls `encode` (in `golay.c`; see page L–26) to process each half. `encode` directly uses the generator matrix for the (24, 12, 3) code to produce a 24-bit Golay code word for each 12-bit argument presented in up to 12 steps (one for each 1 bit in the argument). A table look-up could achieve the encoding in one step, but time is not critical on the transmit side of the simulator because each word is only encoded once.

2.2.3 MakeWord — ale.c (L–1)

Creation of the word records used in the frames data structure is performed by MakeWord. MakeWord is given a pointer to a 49-byte array produced by Tx, a repetition count to be used, and the address of the pointer to a WordRec which is to receive the address of the new WordRec. A WordRec is allocated, and the Word and Repetitions fields are filled in from the arguments. The value of TxLevel set by GetParms (in `ale.c`; see page L–13) is assigned to every WordRec created; if necessary, this value may be modified by the routine calling MakeWord.

2.2.4 SetUpNet — ale.c (L–2)

The simulated network is created by SetUpNet, which clears the address data base, inserts a collection of self, net, and other addresses into the address data base, and then links some stations into nets. The address data base is described in 4.1.1.

2.2.5 SetUpIndivCall — ale.c (L–3)

An individual call consists of three frames: a call, a response, and an acknowledgement. SetUpIndivCall creates the four ALE words (TO SAM, TIS JOE, TO JOE, and TIS SAM) needed, along with LQA and AMD words if requested by the user (see GetParms in `ale.c`; page L–13). These Tx words are then passed to MakeWord to create the three frames. If LQA is enabled, the LQA command will always carry a CMD preamble. If an AMD message is to be sent (in the acknowledgement), the first word will carry a CMD preamble unless it is preceded by an LQA command, in which case it will start with a REP preamble. The following words alternate between DATA and REP in either case.

The first phase of the response and acknowledgement is the portion of the turnaround time between the last word wait time (T_{lww}) and the beginning of the transmission of the frame, and contains one word of noise. A random amount of this word is actually used in the simulation.

2.2.6 SetUpSound — ale.c (L-3)

SetUpSound creates the single frame needed for a sound. If it is desired to simulate only sounds, this function is sufficient as written. If sounds are to be interspersed with other calls, an anchor for this frame other than `frame[0]` must be used, and the handshake sequencing functions must be redirected to this alternate frame as required.

2.2.7 SetUpNetCall — ale.c (L-4)

The net call of FED-STD-1045 differs from the individual call principally in the use of slotted responses. The net call simulation includes a call to the net from the NCS, followed by a response from the first self address found in the called net, and concluded with an acknowledgement to the net from the NCS. The simulated net is determined entirely by the code in SetUpNet: the net members are assigned slots in the order that they are added to the net. SetUpNetCall searches the list of net members to find the first self address, and sets the slot number for the response to that slot. The timing parameters are computed in a fashion similar to that required in an actual ALE controller.

The period for slotted responses is filled with noise, except for the actual response. This provides ample opportunity for the odd sound, etc.

If LQA is enabled, the NCS will request LQA from the net members. The response will report the LQA measured by the respondent from the net call. The acknowledgement will not contain an LQA report. Therefore, the AMD message, if present, will always start with a CMD preamble.

2.3 Handshake Sequencing

The handshake sequencing functions are concerned with simulating the delivery of symbols to the transmitting modem in the order that an ALE controller would produce. The variables shown at the bottom of Figure 2 may be viewed as the state variables of the transmitting state machine: `CurrFrame` indexes the `frames` array; `CurrWordRec` points to the WordRec for the current phase; `WordsToGo` counts the repetitions of the ALE word for that phase; `Txw` points to the array of symbols for that word, and `SymIndex` counts through the symbols in the array.

2.3.1 NextSym — ale.c (L-5)

The "front end" of the symbol sequencer is NextSym, which finds the next symbol to be sent, and returns the result of passing this symbol through the modem and the HF channel. Most of the time, the next symbol is simply the next symbol in the Txw array. When SymIndex increments past the end of the array, NextSym determines whether to repeat the current word, go on to the next phase, or simply send noise (when the frame is complete).

NextSym also maintains the real time in the simulator by counting the symbols sent in `BaudCount`. This variable is cleared by StartHandshake, and used to find the end of the slotted response period in a net call.

The shareCPU function should be redefined in localdef.h to whatever operating system call is needed to support multitasking or background processing in the host environment. For Unix systems, no call is needed and shareCPU should be redefined to the null statement. For the Macintosh OS under System 6.0.x, an approach using WaitNextEvent will permit background processing; under System 7, no call should be needed. A more complicated approach would be needed to implement background processing under MS-DOS.

2.3.2 NextTxWord — ale.c (L–5)

NextTxWord is called to update the transmitter state variables after CurrWordRec is changed.

2.3.3 NoiseWord — ale.c (L–5)

NoiseWord is called instead of NextTxWord when the frame has been concluded and the channel is to carry only noise. WordsToGo is set to -1, which should not decrement to 0 before the receiver stops listening.

2.3.4 StartHandshake — ale.c (L–6)

StartHandshake resets the transmitter state variables to begin a new handshake. CurrChan is set to a random channel so that a scanning receiver arrives on channel with random word phase (see scan in ale.c, page L–6).

2.3.5 NextFrame — ale.c (L–6)

NextFrame sequences the transmitter state variables to the next frame, and selects a random turnaround time (less T_{lww}) between one and two T_w.

3 RECEIVE FEC SIMULATOR

3.1 Data Structures — fec.c (L-21)

The receive FEC module maintains three buffers: a 49-symbol shift register (SR1) that holds the last 49 received symbols for majority voting (a shorter shift register could be used at the expense of slower operation), a 16-entry circular buffer (ubuffer) that's used to store the counts of unanimous votes in the last 16 symbols output from the majority voter, and a pair of shift registers (g1 and g2) that hold the deinterleaved outputs of the majority voting process.

3.2 initRxFEC — fec.c (L-23)

The initRxFEC function is called to simulate the initial processing performed when a receiver lands on a channel carrying ALE signalling: the buffers are cleared and the first 48 symbols received from the modem (i.e., via NextSym) are processed through the majority voting block to produce 16 majority symbols (48 bits) and a count of unanimous votes in the first 48 bits. No evaluation is made of this data in initRxFEC.

The dwell argument passed to initRxFEC specifies how many symbols may be processed in attempting to achieve WordSync. The static variable SymbolsToGo is left equal to dwell – 48. Word sync has not been achieved yet, so the Sync flag is cleared.

3.3 RxFEC — fec.c (L-24)

RxFEC performs majority voting, deinterleaving, and Golay decoding of received symbols. The actions performed by RxFEC depend upon the state of the Sync flag: if set, word sync is assumed and RxFEC simply processes the next 49 symbols into the buffers and returns the result of Golay decoding the 48 bits produced by majority vote, discarding the stuff bit (returning -1 if Golay decoding fails).

If the Sync flag is cleared, RxFEC enters word sync acquisition mode. After each received symbol is shifted into the buffers, the Golay decoder if fed the most recent 48 majority bits; if the both Golay words are correctable and the unanimous votes for those bits exceeds the unanimous vote threshold, the decoded candidate ALE word is returned, and the Sync flag is speculatively set. Otherwise, the next symbol is shifted in, until the SymbolsToGo count expires.

Both majority voting and unanimous vote counting employ a fast table-lookup technique: the three symbols involved are concatenated to form a 9-bit index, which is used to find a pre-computed majority symbol and count of unanimous votes.

3.4 Golay Decoder — fec.c (L-21) golay.c (L–26)

Two Golay decode routines are used. The first, DeGolay (in `fec.c`; see page L-21), takes two 24-bit Golay words and the desired error-correcting power, and calls `decode` (in `golay.c`; see page L–26) to process each Golay word. `decode` implements a modified version of the decoder specified in an earlier report, an returns either the corrected 12 information bits of its argument, or –1 if an uncorrectable error was detected. If either decode fails, DeGolay returns –1.

4 ALE PROTOCOL SIMULATOR

4.1 Data Structures

4.1.1 Address Memory Data Structure — addr.c (L–15)

The address memory data base incorporates the relationships among self, net and other addresses in a single data structure, as shown in Figure 3. This data base is organized as a hash-indexed array of lists of address records, with net structure encoded using independent sets of pointers.

Addresses in the simulator are limited to one ALE word, which can hold three 7-bit characters, or 21 bits. Each address record includes six fields: `name`, which holds a 21-bit address; the `index` into the LQA matrix for that name; the `class` of the name (SELF, NET, or OTHER); a `member` pointer, which points to the next net member in sequence (the first net member if the record is a net address); a `net` pointer, which points to the address record of the net to which `name` belongs; and a `next` pointer, which links address records into a list within each hash bin.

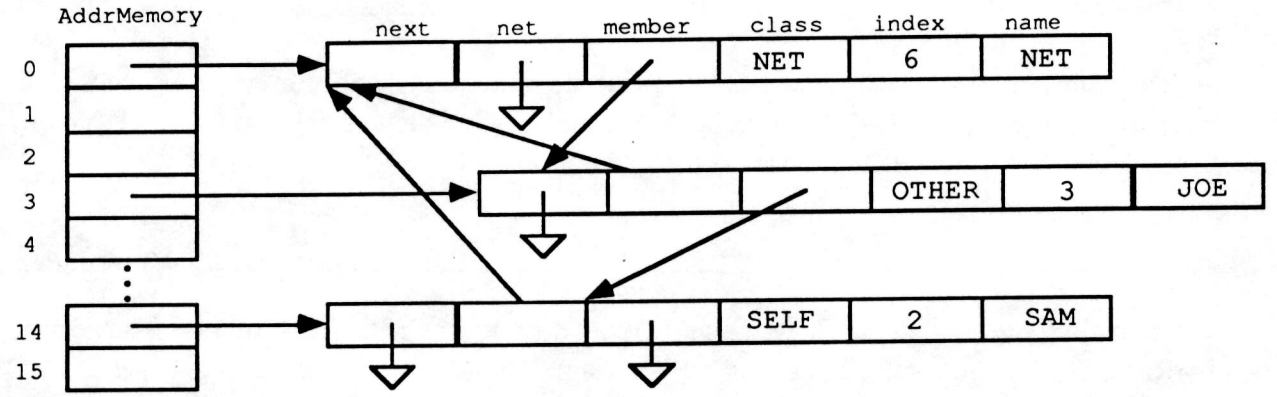


Figure 3: Address Memory Data Structure

In searching for an address in the data base, the address sought is hashed trivially by using the four lsbs of the 21-bit address as an index into the AddrMemory array of pointers to address records. Within each bin, address records are listed in increasing order of `name`.

In the Figure, SAM is a self address, JOE is an other address, and NET is a net address. Both JOE and SAM are members of NET. JOE will respond in slot 1 and SAM will respond in slot 2.

4.1.2 LQA Matrix — message.c (L–18)

The LQA matrix stores the most recent BER data for each address listed in the address data base. Each entry contains two fields: one for the BER measured on the last frame received from the addressee, and one for the last BER reported by that addressee from its measurement of our own transmissions.

4.2 Simulated ALE Receiver

4.2.1 RxALE — ale.c (L–12)

RxALE is called by the main simulator loop to simulate an ALE controller entering the available state. It attempts to handle whatever it finds on the active channel, and counts the links that it has achieved.

4.2.2 StopScan — ale.c (L–7)

StopScan scans to the active channel and tries to achieve word sync. If it reads a TO word, it looks up the address contained in the TO word and returns a pointer to the address record to RxALE. If it gets a TIS or TWAS word instead, it calls LogSound. Otherwise, it continues to scan and try to read a word until the transmission ceases. The variable `retries` counts the number of times StopScan has returned to scan and attempted to re-acquire the signal. When scanning at 5 channels/sec, it is sometimes possible to achieve word sync in T_{sc} after 3 retries.

4.2.3 RcvCall — ale.c (L–8)

RcvCall follows the remainder of a call after StopScan has found a valid individual or net call. Up to 3 words can be lost during T_{sc}. If a message section starts, FinishMessage is called to process the commands. Either a TIS or a TWAS conclusion prompts the storage of the measured BER value. If a TIS conclusion is received, the call is successfully completed, and a pointer to the address record of the caller is returned after T_{lww} expires.

4.2.4 Handshake — ale.c (L–9)

The Handshake function is given a pointer to the address record of the expected TO addressee. If it achieves word sync, it calls LogSound for a TIS or TWAS preamble, or checks the addressee in the case of a TO preamble. If the addressee is correct, the second word received is compared to the first (it must match, because we should be in T_{lc}). Further repetitions of the TO word are permitted, although this may be disabled simply. Messages and termination are handled as in RcvCall.

If an LQA command is to be included in the frame, the appropriate word is computed by MakeLQAword and placed in LQAword, which was linked into the appropriate frames when the frames were set up.

4.2.5 SlottedResp — ale.c (L–10)

initRxFEC is invoked with a dwell time equal to the entire period reserved for slotted responses. WordSync is called repeatedly until the slotted response time expires; each transmission received is processed similarly to the Handshake discussion above, with the following exceptions:

- Calling cycles longer than 2 T_{rw} are rejected.
- Rejected receptions do not abort the entire handshake.

Note that WordSync is expected to fail at the end of the slotted response period so that the handshake can continue.

4.3 Supporting Receiver Functions

4.3.1 Scan — ale.c (L-6)

The active channel is channel 0. Scan leaves CurrChan and dwells the specified DwellTime on each channel until it reaches channel 0.

4.3.2 WordSync — ale.c (L-7)

WordSync calls RxFEC to achieve FEC-level synchronization (i.e., unanimous vote threshold and successful Golay decodes). If RxFEC doesn't report failure (–1), WordSync checks for three ASCII-38 characters and a valid preamble (TO, TIS, or TWAS for this simulation). If RxFEC delivered a candidate which failed these tests, the Sync flag is cleared and RxFEC is re-invoked to continue seeking word sync until the dwell time set in initRxFEC expires.

If word sync is achieved, WordCount and TotalUcount are initialized for accumulation of the total number of words and the total number of unanimous votes in the frame. These are used upon successful conclusion of the frame to compute the pseudo BER.

4.3.3 ASCII_OK — ale.c (L-2)

ASCII_OK uses an ASCII_Set table to find the most restrictive ASCII subset to which each of the three characters passed to it belongs. If all three characters are in the set specified by the set argument, the word is returned unchanged. Otherwise, 0 is returned.

4.3.4 PreambleOK — ale.c (L-7)

PreambleOK uses a boolean table indexed by preamble to determine whether the preamble received is acceptable during word sync acquisition. TO is valid for individual and net calls; TIS and TWAS are valid for sounds. THRU, DATA, and REP would be valid if group calls (all three) or multi-word addresses (DATA and REP) were supported in the simulator (they're not).

4.3.5 NextWord — ale.c (L-2)

NextWord is called after word sync is achieved to return the next ALE word received. After calling RxFEC, NextWord accumulates unanimous vote statistics and returns the received word. If RxFEC reported failure (bad Golay decodes), no unanimous votes are recorded for the word.

4.3.6 LogSound — ale.c (L-6)

LogSound compares the next word to the last, and prints a "sound" message if they match. Storing the BER measured from a sound is not implemented, but it would be straightforward to do so.

4.3.7 IgnoreFrame, FinishTransmissions — ale.c (L-6)

IgnoreFrame is called to pull the rest of an aborted frame through the HF channel, so that the same sequence of random numbers is used whether or not links succeed. FinishTransmissions pulls *all* remaining frames through the channel when an entire handshake is aborted.

4.4 Address Management

4.4.1 initAddrMemory — addr.c (L–15)

The AddrMemory bins are all set to NULL pointers.

4.4.2 InsertAddr — addr.c (L–16)

The appropriate AddrMemory bin is searched for the name to be inserted. If it isn't found, a new address record is created and inserted into the list in that bin.

4.4.3 AddMember — addr.c (L–16)

Adds a new member to a net at the end of the list of members for that net. Slot numbers are implicitly assigned by the position of each address record in the list of net members. An address record can belong to at most one net.

4.4.4 FindAddr — addr.c (L–15)

An AddrMemory bin is found by hashing the name sought. The linked list in that bin is searched for a name match.

4.4.5 SelfAddr, OtherAddr, and NetAddr — ale.c (L–11)

Given a pointer to an address record, these functions return TRUE if the designated address record is of class SELF, OTHER, or NET, respectively.

4.4.6 AddrString — addr.c (L–15)

AddrString converts a 21-bit address into a NULL-terminated character string. It returns a pointer to a private long, which is assumed to be 4 bytes. Because new storage is not allocated for each string returned, the result of the function should be presumed to have an ephemeral life, and is best used for `printfs` and similar purposes.

4.4.7 ThreeChar, AMDchar — addr.c (L–15)

These functions are the inverse of AddrString: they convert a character string argument into a 21-bit address word. ThreeChar pads on the right with '@' (as specified for addresses), while AMDchar pads on the right with spaces (as specified for AMD). These help to create ALE words from user-specified character strings at run time.

4.4.8 PrintAddrMemory — addr.c (L–17)

The contents of non-empty bins are listed, including the class and LQA matrix index of each record, and the name of the net to which each belongs, if any. Each net address is followed by a list of the net members, in slot order.

4.5 Message Handling

4.5.1 initLQAmatrix — message.c (L–18)

All entries are set to NoReport (MP = 111; SINAD = 11111; BER = 11111). A character array is also allocated for LQAword, which may be pointed to by several WordRecs. LQAword must be filled with a current LQA command before each use.

4.5.2 MakeLQAword — message:c (L–18)

MakeLQAword creates a pre-computed LQA command in LQAword. The following algorithm is followed to determine the contents of the command:

1. The preamble is always CMD, and the first character is always 'a' (i.e., an LQA command).
2. If no destination is specified, the command will be an LQA request, with no LQA report.
3. If a destination is specified,
   a. If an LQA report was requested, the most recent MEASURED LQA value for that station is reported.
   b. If an LQA report has not been received recently from that station, an LQA report is requested.

4.5.3 FinishMessage — message.c (L–19)

FinishMessage continuously reads words until it receives a frame conclusion or the message section protocol is violated. AMD words are displayed as received. LQA reports are stored in a variable LQAreported until the sender of the report is known (after frame conclusion). Other commands are ignored, and are assumed to be one-word commands (alternating CMD and REP preambles).

4.5.4 StoreBER — message.c (L–20)

StoreBER is called when a frame is concluded. The PBER for the concluded frame is calculated and stored in the LQA matrix location assigned to the address received in the terminating word as a MEASURED value. If a value is found in LQAreported, it is stored as a REPORTED value for the same address, and LQAreported is reset to NoReport.

4.5.5 DisplayAMD — message.c (L–18)

The three characters contained in the argument word are converted to a character string and displayed. Words with uncorrectable errors (i.e., word = –1) are displayed as *** (no "block" character is available on many displays).

4.6 ALE Simulator

4.6.1 GetParms — ale.c (L–14)

Five parameters are required to specify the simulation to be run: multipath delay, fading frequency spread, and SNR; and the number and type of handshakes to simulate. In systems supporting command line arguments, these may be entered on the command line. GetParms prompts for any parameters not specified as command line arguments.

The following global variables are set by GetParms: the flags NetCalls, LQA, and AMD; SigLev and TxLevel; and DlyTime. The main routine passes a pointer to `attempts` so that GetParms can set the number of links to be attempted.

4.6.2 main — ale.c (L–14)

The main routine initializes the various modules, times the simulation, and launches the requested number of simulated handshakes. The transmission is initiated via StartHandshake; then RxALE is called to start the receiver.

5 EXAMPLE RESULTS

5.1 Verbose Output

Both the FEC and ALE simulators contain sections of code which are conditionally compiled depending upon whether the flag VERBOSE is defined or undefined at compile time. If VERBOSE is defined, the detailed operation of the simulator is described on the standard output. Three different combinations of this flag are useful:

- VERBOSE undefined for both modules — produces a compact output suitable for observing long simulations; a period is printed for a successful link, and an asterisk for a linking failure. The standard makefile produces this combination, and the executable file is named `ale`.
- VERBOSE defined in the ALE simulator but undefined in the FEC module — describes the events at the ALE protocol level. The makefile `monitor` produces this combination, and names the executable file `monale`.
- VERBOSE defined in both modules — prints a line of output each time the Golay decoder is called; these lines are interspersed among the ALE protocol outputs. The makefile `verbose` produces this combination, and names the executable file `vale`.

If the host system doesn't support command line definition of such flags, VERBOSE may be defined or undefined explicitly in the respective files `ale.c` and `fec.c`.

Listings for three situations are on the following pages:

- `Monale` output for a poor channel, 6 dB SNR, showing an individual call (no LQA or AMD).
- `Monale` output for a poor channel, 3 dB SNR, showing a net call with LQA and AMD.
- `Vale` output for a good channel, 6 dB SNR, showing an individual call (no LQA or AMD).

The command line arguments to all three programs are as follows:

- Multipath delay in ms.
- Fading frequency spread in Hz.
- SNR in dB.
- Number of handshakes to simulate
- Type of handshake:
  0: Individual calls, no LQA or AMD
  1: Net calls, no LQA or AMD
  2: Individual calls with LQA
  3: Net calls with LQA
  4: Individual calls with AMD in the acknowledgement
  5: Net calls with AMD in the acknowledgement
  6: Individual calls with LQA and AMD
  7: Net calls with LQA and AMD

```
% monale 2 2 6 1 0

Individual Calls
Delay =  2.0000 ms, Signal Level =   1.9953
Doppler spread =  2.0000 Hz; Run length = 1

Inserting 5160315 (SAM) in bin 13
Inserting 4523705 (JOE) in bin 5
Inserting 5020315 (PAM) in bin 13
Inserting 4663705 (MOE) in bin 5
Inserting 5120315 (RAM) in bin 13
Inserting 4721324 (NET) in bin 4
Inserting 4720723 (NCS) in bin 3
Inserting 5221304 (TED) in bin 4
Adding member "JOE" to net "NET"
Adding member "PAM" to net "NET"
Adding member "SAM" to net "NET"
Adding member "RAM" to net "NET"

Non-empty bins of Address Memory

   3     (NCS, 2, 6)

   4     (NET, 1, 5) -> (JOE, 2) -> (PAM, 2) -> (SAM, 0) -> (RAM, 2)
         (TED, 2, 7)

   5     (JOE, 2, 1)  <in net NET>
         (MOE, 0, 3)

  13     (PAM, 2, 2)  <in net NET>
         (RAM, 2, 4)  <in net NET>
         (SAM, 0, 0)  <in net NET>

Scanning .....
Looking for word sync....
      RxFEC returns 25160315   ASCII_OK   preamble OK    TO SAM
Finishing calling cycle....
    PBER = 4
    Storing PBER values for JOE.  Measured: 4, Reported: 31    OTHER address
(JOE)

Next frame...
Looking for word sync....
      RxFEC returns 24523705   ASCII_OK   preamble OK    TO JOE
    PBER = 8
    Storing PBER values for SAM.  Measured: 8, Reported: 31

Next frame...
Looking for word sync....
      RxFEC returns 25160315   ASCII_OK   preamble OK    TO SAM
    PBER = 6
    Storing PBER values for JOE.  Measured: 6, Reported: 31

Linked with JOE
```

```
% monale 2 2 3 1 7

Net Calls with LQA and AMD
Delay =  2.0000 ms, Signal Level =   1.4125
Doppler spread =  2.0000 Hz; Run length = 1

Inserting 5160315 (SAM) in bin 13
Inserting 4523705 (JOE) in bin 5
Inserting 5020315 (PAM) in bin 13
Inserting 4663705 (MOE) in bin 5
Inserting 5120315 (RAM) in bin 13
Inserting 4721324 (NET) in bin 4
Inserting 4720723 (NCS) in bin 3
Inserting 5221304 (TED) in bin 4
Adding member "JOE" to net "NET"
Adding member "PAM" to net "NET"
Adding member "SAM" to net "NET"
Adding member "RAM" to net "NET"

Non-empty bins of Address Memory
   3     (NCS, 2, 6)
   4     (NET, 1, 5) -> (JOE, 2) -> (PAM, 2) -> (SAM, 0) -> (RAM, 2)
         (TED, 2, 7)
   5     (JOE, 2, 1)  <in net NET>
         (MOE, 0, 3)
  13     (PAM, 2, 2)  <in net NET>
         (RAM, 2, 4)  <in net NET>
         (SAM, 0, 0)  <in net NET>

SAM will respond in slot 3.
4 slots total (excl. slot 0)

LQAword = 66077777
Scanning .....
Looking for word sync....
      RxFEC returns 24721324   ASCII_OK   preamble OK    TO NET
Finishing calling cycle....
  Message(s): LQA request
    PBER = 7
    Storing PBER values for NCS.  Measured: 7, Reported: 31   OTHER address (NCS)

Slotted Responses...
(requesting LQA report)    (LQA was requested)    LQAword = 66060007
Looking for word sync....
      RxFEC returns 24720723   ASCII_OK   preamble OK    TO NCS
  Message(s): LQA = 7
    PBER = 7
    Storing PBER values for SAM.  Measured: 7, Reported: 7    Response from SAM
Looking for word sync....WordSync failed.
(WordSync is expected to fail after the last slotted response.)
Responses = 1

Next frame...
(requesting LQA report)    (LQA was requested)    LQAword = 66077777
Looking for word sync....
      RxFEC returns 24721324   ASCII_OK   preamble OK    TO NET
  Message(s): AMD message "GOOD WORK"
    PBER = 5
    Storing PBER values for NCS.  Measured: 5, Reported: 31

Linked with NCS
```

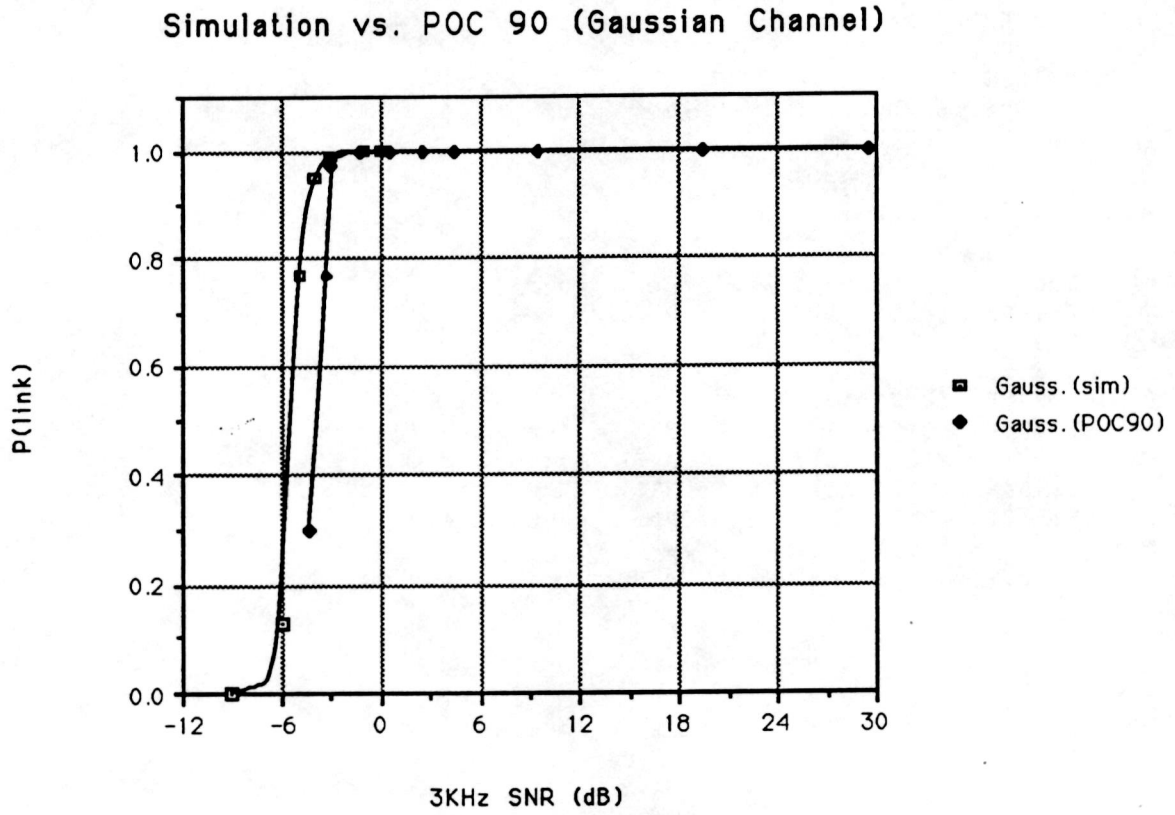
```
% vale .5 .1 6 1 0

Individual Calls
Delay =  0.5000 ms, Signal Level =   1.9953
Doppler spread =  0.1000 Hz; Run length = 1

Inserting 5160315 (SAM) in bin 13
Inserting 4523705 (JOE) in bin 5
Inserting 5020315 (PAM) in bin 13
Inserting 4663705 (MOE) in bin 5
Inserting 5120315 (RAM) in bin 13
Inserting 4721324 (NET) in bin 4
Inserting 4720723 (NCS) in bin 3
Inserting 5221304 (TED) in bin 4
Adding member "JOE" to net "NET"
Adding member "PAM" to net "NET"
Adding member "SAM" to net "NET"
Adding member "RAM" to net "NET"

Non-empty bins of Address Memory

   3     (NCS, 2, 6)

   4     (NET, 1, 5) -> (JOE, 2) -> (PAM, 2) -> (SAM, 0) -> (RAM, 2)
         (TED, 2, 7)

   5     (JOE, 2, 1)  <in net NET>
         (MOE, 0, 3)

  13     (PAM, 2, 2)  <in net NET>
         (RAM, 2, 4)  <in net NET>
         (SAM, 0, 0)  <in net NET>

Inputs to Golay:  00002516 00000315
New Tx word 25160315 at 17d68
Inputs to Golay:  00005452 00003705
New Tx word 54523705 at 17da8
Inputs to Golay:  00002452 00003705
New Tx word 24523705 at 17de8
Inputs to Golay:  00005516 00000315
New Tx word 55160315 at 17e28
Inputs to Golay:  00006436 00003717
New Tx word 64363717 at 17e68
Inputs to Golay:  00000421 00000127
New Tx word 04210127 at 17ea8
Inputs to Golay:  00007476 00004513
New Tx word 74764513 at 17ee8
Scanning .
Looking for word sync....
        13140315  77342542
        76712472  54606312
        31403153  73455426
        67124722  46013120
        14031537  34524264
        71247226  60141201
        40315375  45232646
        12472263  01462010
        03153756  52356463
        24722630  14650106
        31537562  23514637
        47226300  46571065
        15375625  35136374
```

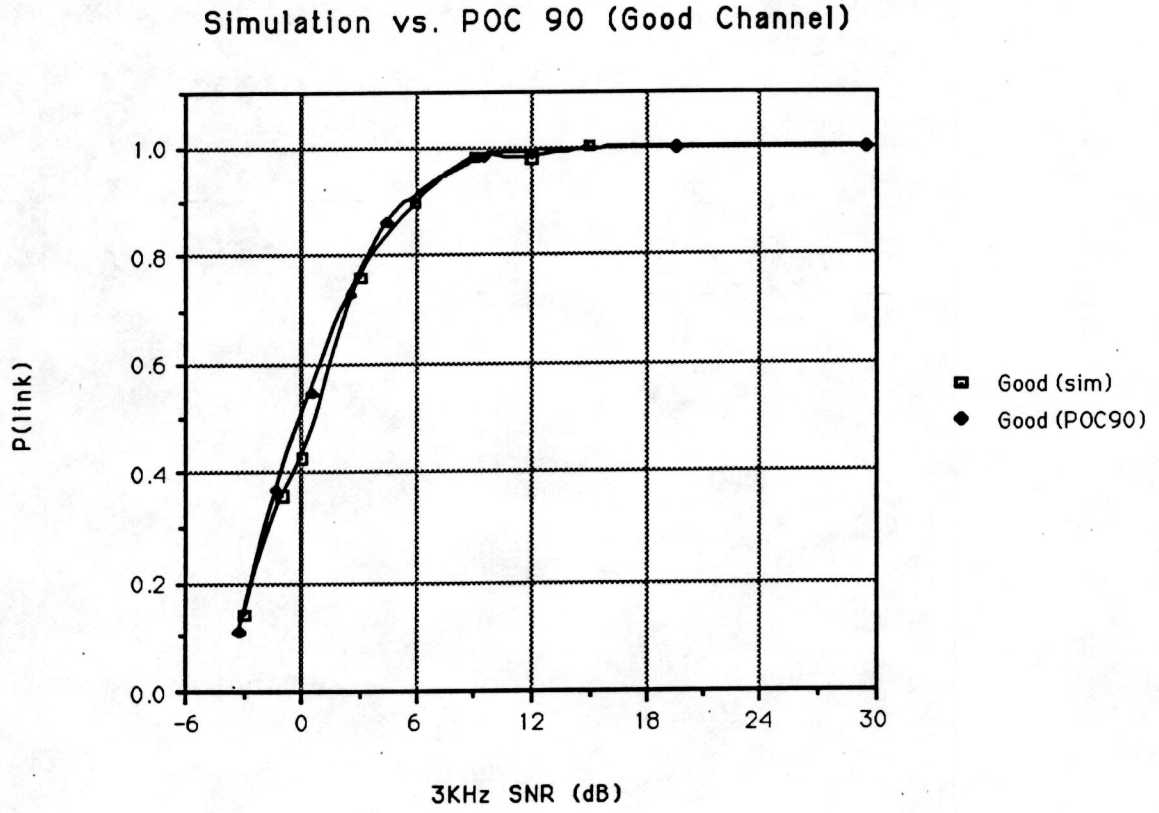
```
        72263006  65760653
        53756251  51313746
        22630063  57676530
        37562516  13147462
        26300632  76715305
        75625164  31404624
        63006327  67123055
        56251645  14036240
        30063277  71240551
        62516454  40312402
        00632773  12475514
        25164546  03154021    25160315    Ucount = 37
      RxFEC returns 25160315   ASCII_OK   preamble OK    TO SAM
Finishing calling cycle....
        25164546  03154021    25160315    Ucount = 40
        25164546  03154021    25160315    Ucount = 40
        25164546  03154021    25160315    Ucount = 45
        25164546  03154021    25160315    Ucount = 47
        25164546  03154021    25160315    Ucount = 44
        25164546  03154021    25160315    Ucount = 43
        25164546  03154021    25160315    Ucount = 48
        25164546  03154021    25160315    Ucount = 45
        25164546  03154021    25160315    Ucount = 47
        25164546  03154021    25160315    Ucount = 46
        25164546  03154021    25160315    Ucount = 41
        27160546  03154021    25160315    Ucount = 31
        25164546  03154023    25160315    Ucount = 41
        25164546  03154021    25160315    Ucount = 39
        25164506  03154061    25160315    Ucount = 39
        25144546  03154021    25160315    Ucount = 36
        25164546  03154021    25160315    Ucount = 46
        25164546  03154021    25160315    Ucount = 43
        25164504  03154021    25160315    Ucount = 39
        25564746  03154221    25160315    Ucount = 34
        54127270  37051542    54523705    Ucount = 35
    PBER = 7
    Storing PBER values for JOE.  Measured: 7, Reported: 31   OTHER address (JOE)

Next frame...
Looking for word sync....
        34436056  45276521
        12562535  62177506
        44360562  52715210
        25625356  21705066
        43605622  27122107
        56253561  17020667
        36056221  71251073
        62535610  70276672
        60562212  12560734
        25356107  02716725
        05622124  25677340
        53561076  27117255
        56221245  56703407
        35610761  71102552
        62212452  67044072
        56107613  11055524
        22124527  70430721
        61076134  10525242
        21245273  04377215
        10761344  05222421
        12452734  43702155
        07613445  52254216
```

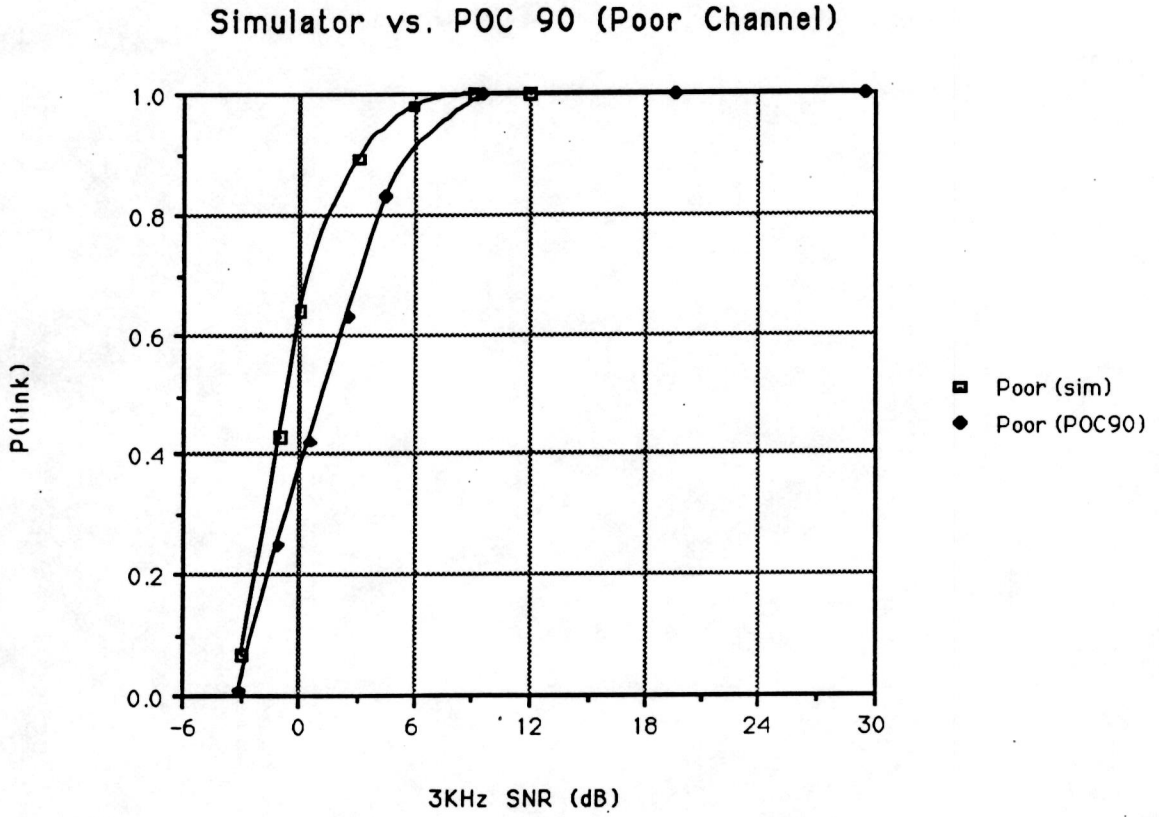
```
        24527342  37051552    24523705    Ucount = 42
      RxFEC returns 24523705   ASCII_OK   preamble OK    TO JOE
        24527342  37051542    24523705    Ucount = 42
        55160474  03154021    55160315    Ucount = 48
    PBER = 4
    Storing PBER values for SAM.  Measured: 4, Reported: 31

Next frame...
Looking for word sync....
        76712472  44606312
        11403153  73455426
        67124722  46013120
        14031537  34524264
        71247226  60141201
        40315375  45232646
        12472263  01462010
        03153756  52356463
        24722630  14650106
        31537562  23514637
        47226300  46571065
        15375625  35136374
        72263006  65760653
        53756251  51313746
        22630063  57676530
        37562516  13147462
        26300632  76715305
        75625164  31404624
        63006327  67123055
        56251645  14036240
        30063277  71240551
        62516454  40312402
        00632773  12475514
        25164546  03154021    25160315    Ucount = 48
      RxFEC returns 25160315   ASCII_OK   preamble OK    TO SAM
        25164546  03154021    25160315    Ucount = 48
        54523270  37051542    54523705    Ucount = 48
    PBER = 0
    Storing PBER values for JOE.  Measured: 0, Reported: 31

Linked with JOE
```

5.2 Comparison to POC '90

The data plotted in the following figures was generated using individual calls, no LQA or AMD, and scanning 5 channels per second. As can be seen, the simulator produces results similar to those obtained in the POC '90 tests. The better performance of the simulated radio for the Gaussian and poor channels may be due to the ability of the simulated radio to try to achieve word sync up to four times during T_{sc}.




Simulation vs. POC 90 (Good Channel)
P(link)
1.0
0.8
0.6
0.4
0.2
0.0
-6
0
6
12
18
24
30
3KHz SNR (dB)
Good (sim)
Good (POC90)


Simulator vs. POC 90 (Poor Channel)
P(link)
1.0
0.8
0.6
0.4
0.2
0.0
-6
0
6
12
18
24
30
3KHz SNR (dB)
Poor (sim)
Poor (POC90)

LISTINGS

```
/*      ale.c           eej             8/4/91          */
/*                      simulator for FED-STD-1045 ALE  */

#define GOLAY_SCAN      1
#define GOLAY_RUN       3
/*#define VERBOSE*/

#include "hfdef.h"      /*  constants defined; flags for other header files */
#include <stdio.h>
#include <math.h>

#include "hflib.h"      /*  tone-level HF propagation simulator   */
#include "ALEmodem.h"   /*  ALE modem simulator                   */
#include "ALEwords.h"
#include "fec.h"        /*  FEC routines                          */


/************************  ALE Handshake Creation  ************************/

struct WordRec {
    struct WordRec *next;
    int             Repetitions;
    char           *Word;
    float           TxLevel;
} ;

#define MaxFrames 5
static struct WordRec *frames[MaxFrames];
static short NetCalls, /* 0 for indiv, 1 for net */
             LQA = 0,  /* Use LQA? */
             AMD = 0;  /* Use AMD? */
static char *LQAword;          /* scratchpad for current LQA word */
static int ValidFrames;
static long WordCount, TotalUcount;  /* used for PBER measurement */

static struct WordRec *CurrWordRec; /*  points to current WordRec. */

static char *Txw;       /*  points to current word.     */
static int  CurrFrame,  /*  which transmission?         */
            WordsToGo,  /*  counts down.                */
            SymIndex;   /*  which symbol in word?       */

#define NumChan 10      /*  number of channels scanned  */
#define DwellTime 0.2   /*  time per channel in seconds */

static int CurrChan;  /*  current receiver channel (Tx always on chan 0) */

struct WordRec *MakeWord(word, reps, WordRecPtrAddr)
char *word;
int reps;
struct WordRec **WordRecPtrAddr;
{
    *WordRecPtrAddr = (struct WordRec *) malloc(sizeof(struct WordRec));
    (*WordRecPtrAddr)->TxLevel = TxLevel;     /* TxLevel is global */
    (*WordRecPtrAddr)->Word = word;
    (*WordRecPtrAddr)->Repetitions = reps;
    (*WordRecPtrAddr)->next = (struct WordRec *) 0;
    return *WordRecPtrAddr;    /* return pointer to new WordRec */
}
```

```
long NextWord()
{
    long w = RxFEC(GOLAY_RUN);

    WordCount++;
    if (w != -1) TotalUcount += Ucount;
    return w;
}

#include "ASCII_Set.h"

long ASCII_OK(w, set)
long w;
char set;
{
    if (ASCII_Set[w & 0177] >= set)
      if (ASCII_Set[(w >> 7) & 0177] >= set)
        if (ASCII_Set[(w >> 14) & 0177] >= set)
          return w;
    return 0;
}

#include "addr.c"        /* address memory data base and functions     */
#include "message.c"     /* LQA matrix and message handling functions  */

void SetUpNet()
{
    initLQAmatrix();
    initAddrMemory();
    InsertAddr(_SAM, SELF);
    InsertAddr(_JOE, OTHER);
    InsertAddr(_PAM, OTHER);
    InsertAddr(_MOE, SELF);
    InsertAddr(_RAM, OTHER);
    InsertAddr(_NET, NET);
    InsertAddr(_NCS, OTHER);
    InsertAddr(ThreeChar("TED"), OTHER); /* here's how to use other names */

    AddMember(_NET, _JOE);
    AddMember(_NET, _PAM);
    AddMember(_NET, _SAM);
    AddMember(_NET, _RAM);

#ifdef VERBOSE
    PrintAddrMemory();
#endif
}
```

```
/* only set up one of the following:  individual call */
/*                                    net call        */
/*                                    sound           */

void SetUpIndivCall()
{
    static char *TO_SAM, *TIS_JOE, *TO_JOE, *TIS_SAM, *AMD1, *AMD2, *AMD3;
    struct WordRec *cursor;

    TO_SAM = Tx(_TO+_SAM, (char *) 0);
    TIS_JOE = Tx(_TIS+_JOE, (char *) 0);
    TO_JOE = Tx(_TO+_JOE, (char *) 0);
    TIS_SAM = Tx(_TIS+_SAM, (char *) 0);
    if (LQA)
        AMD1 = Tx(_REP + AMDchar("GOO"), (char *) 0); /* LQA will be CMD */
    else AMD1 = Tx(_CMD + AMDchar("GOO"), (char *) 0);
    AMD2 = Tx(_DATA + AMDchar("D W"), (char *) 0);
    AMD3 = Tx(_REP + AMDchar("ORK"), (char *) 0);

    frames[0] = MakeWord(TO_SAM, 20, &cursor);          /*  CALL  */
       cursor = MakeWord(TO_SAM,  2, cursor);
       if (LQA) cursor = MakeWord(LQAword, 1, cursor);
       cursor = MakeWord(TIS_JOE, 1, cursor);

    frames[1] = MakeWord(TO_JOE, 1, &cursor);           /*  RESP  */
       cursor->TxLevel = 0.0;  /* part of Twr */
       cursor = MakeWord(TO_JOE, 2, cursor);
       if (LQA) cursor = MakeWord(LQAword, 1, cursor);
       cursor = MakeWord(TIS_SAM,1, cursor);

    frames[2] = MakeWord(TO_SAM, 1, &cursor);           /*  ACK   */
       cursor->TxLevel = 0.0;  /* part of Twr */
       cursor = MakeWord(TO_SAM, 2, cursor);
       if (LQA) cursor = MakeWord(LQAword, 1, cursor);
       if (AMD) {
           cursor = MakeWord(AMD1, 1, cursor);
           cursor = MakeWord(AMD2, 1, cursor);
           cursor = MakeWord(AMD3, 1, cursor);
       }
       cursor = MakeWord(TIS_JOE,1, cursor);

    ValidFrames = 3;
}


void SetUpSound()
{
    static char *TIS_MOE;
    struct WordRec *cursor;

    TIS_MOE = Tx(_TIS+_MOE, (char *) 0);

    frames[0] = MakeWord(TIS_MOE, 22, &cursor);      /*  SCANNING SOUND  */

    ValidFrames = 1;
}
```

```
/*  The following timing parameters are initialized by SetUpNetCall */

static double Tw, Tsw, Twrn, Tswt;  /* defined in FED-STD-1045 */
static int NS;                      /* number of slots         */
#define Trw 49              /* symbols (tones) in redundant word */

void SetUpNetCall()
{
    static char *TO_NET, *TIS_NCS, *TO_NCS, *response, *AMD1, *AMD2, *AMD3;
    struct WordRec *cursor;
    int slot;
    struct addr *memb, *net;

    TO_NET = Tx(_TO + _NET, (char *) 0);
    TIS_NCS = Tx(_TIS + _NCS, (char *) 0);
    TO_NCS = Tx(_TO + _NCS, (char *) 0);
    AMD1 = Tx(_CMD + AMDchar("GOO"), (char *) 0);
    AMD2 = Tx(_DATA + AMDchar("D W"), (char *) 0);
    AMD3 = Tx(_REP + AMDchar("ORK"), (char *) 0);

    /* Determine slot and address for response */
    if (!(net = FindAddr(_NET))) {
        printf("\nfailed to find net %s\n", AddrString(_NET));
        exit();
    }
    slot = 1;
    if (memb = net->member) do {    /* find first self address */
        if ((memb->class) == SELF) break;
        slot++;
    } while (memb = memb->member);
    if (!memb) {
        printf("\nNo self addresses in net %s\n", AddrString(_NET));
        exit();
    }
    response = Tx(_TIS+memb->name, (char *) 0);  /* make response ALE word */

#ifdef VERBOSE
    printf("\n%s will respond in slot %d.", AddrString(memb->name), slot);
#endif

    NS = slot;
    while (memb = memb->member) NS++;  /* count remaining net members */

#ifdef VERBOSE
    printf("\n%d slots total (excl. slot 0)\n", NS);
#endif

    Tw = 49.0 / 3.0;        /* word time in terms of symbol times */
    Tsw = 14 * Tw;          /* slot width */
    if (LQA) Tsw += 3 * Tw;
    Twrn = (NS + 1) * Tsw; /* total time for all slots (incl. slot 0) */
    Tswt = slot * Tsw;      /* beginning of our slot */

    frames[0] = MakeWord(TO_NET, 20, &cursor);          /*  NET CALL  */
       cursor = MakeWord(TO_NET,  2, cursor);
       if (LQA) cursor = MakeWord(LQAword, 1, cursor);
       cursor = MakeWord(TIS_NCS, 1, cursor);
```

```
                                                        /*  SLOTTED RESP  */
    frames[1] = MakeWord(TO_NCS, (int)(Tswt/49.0) + 1, &cursor);
       cursor->TxLevel = 0.0;  /* just noise in preceding slots */
       cursor = MakeWord(TO_NCS, 2, cursor);
       if (LQA) cursor = MakeWord(LQAword, 1, cursor);
       cursor = MakeWord(response, 1, cursor);
       /* any remaining slots will automatically be */
       /* filled with noise words by NextSym().     */

                                                      /*  NET ACK   */
    frames[2] = MakeWord(TO_NET, 1, &cursor);
       cursor->TxLevel = 0.0;  /* NCS Tta */
       cursor = MakeWord(TO_NET, 2, cursor);
       if (AMD) {
           cursor = MakeWord(AMD1, 1, cursor);
           cursor = MakeWord(AMD2, 1, cursor);
           cursor = MakeWord(AMD3, 1, cursor);
       }
       cursor = MakeWord(TIS_NCS,1, cursor);

    ValidFrames = 3;
}

/**********************   ALE Handshake Sequencing   ***********************/

void NextTxWord()
{
    Txw = CurrWordRec->Word;
    WordsToGo = CurrWordRec->Repetitions;
    SymIndex = 0;
    SigLev = CurrWordRec->TxLevel;
}

static char NoiseTones[49] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

void NoiseWord()
{
    Txw = NoiseTones;
    WordsToGo = -1;
    SymIndex = 0;
    SigLev = 0.0;
}

static long BaudCount;     /* used to measure (simulated) real time */
                           /* cleared by StartHandshake             */
                           /* incremented by NextSym                */

char NextSym()   /* returns next symbol received from modem */
{
    shareCPU();      /* O/S dependent routine for background processing */
    BaudCount++;
    if (SymIndex >= 49) {        /* start next redundant word ? */
        SymIndex = 0;
        if (!(--WordsToGo)) {      /* time for a new word ? */
            if (CurrWordRec)
              if (CurrWordRec = CurrWordRec->next) NextTxWord();
              else NoiseWord();
        }
    }
    return (ALEmodem(Txw[SymIndex++])); /* NOTE: POST-incremented! */
}
```

```
void NextFrame()
{
    CurrWordRec = frames[++CurrFrame];
    if (CurrFrame >= ValidFrames) {
        printf("\n\007**** ERROR: ran out of frames!\n");  exit();
    }
    NextTxWord();
    SymIndex = 16 + RandInt(17);  /* Tta between 1 and 2 Tw */
}

void StartHandshake()
{
    CurrWordRec = frames[CurrFrame = 0];
    NextTxWord();
    CurrChan = RandInt(NumChan); /* start receiver at a random channel */
    BaudCount = 0;
}

void Scan()
{
    register float dwell;
#ifdef VERBOSE
    printf("\nScanning ");
#endif
    while (++CurrChan < NumChan) {
#ifdef VERBOSE
        printf(".");
#endif
        for (dwell=0.0; dwell <= DwellTime; dwell += Ts)
          NextSym();
    }
    CurrChan = 0;
}

void LogSound(w) /************************ STUB ***************************/
long w;
{
    if (NextWord() == w)
      printf("\nSound from \"%s\"\n", AddrString(w));
}

void IgnoreFrame()
{
    int i;

    while (SigLev > 0.0) NextSym();  /* finish current frame */
    for (i=0; i<49; i++) NextSym();  /* let Tlww expire */
}

void FinishTransmissions()
{
    IgnoreFrame();
    while (CurrFrame < ValidFrames-1) {
        NextFrame();                        /* go on to next frame */
        IgnoreFrame();
    }
}

#define FAIL { FinishTransmissions();  return ((struct addr *) 0); }
```

```
int PreambleOK(w)
long w;
{
    static int PreambleMask[8] = {0, 0, 1, 1, 0, 1, 0, 0};

    if (w) {
#ifdef VERBOSE
        printf("ASCII_OK   ");
#endif
        return PreambleMask[(w >> 21) & 7];
    }
    return 0;
}

long WordSync()
{
    long w;

#ifdef VERBOSE
    printf("\nLooking for word sync....");
#endif
    while ((w=RxFEC(GOLAY_SCAN)) != -1) { /* keep trying until RxFEC gives up */
#ifdef VERBOSE
        printf("\n      RxFEC returns %08lo   ", w);
#endif
        if (PreambleOK(ASCII_OK(w, ASCII_38))) {
#ifdef VERBOSE
            printf("preamble OK    ");
#endif
            WordCount = 1;
            TotalUcount = Ucount;
            return(w);
        }
        Sync = 0;
        printf("!");
    }
#ifdef VERBOSE
    printf("WordSync failed.\n");
#endif
    return (-1);  /* word sync failed */
}

long retries = 0;

struct addr *StopScan()
{
    long word, name;
    if (LQA) MakeLQAword((struct addr *) 0);
    while (SigLev) {
        Scan();
        initRxFEC(2*Trw);     /* allow 2 Trw for word sync */
        if (((word = WordSync()) >> 21) == TO) {
#ifdef VERBOSE
            printf("TO ");
#endif
            return FindAddr(word);
        }
        if (word != -1) {  LogSound(word);  FAIL  }
        retries++;
    }
    FAIL
}
```

```
#define MaxLostWords 3

struct addr *RcvCall(callee)
struct addr *callee;
{
    int LostWords = 0, preamble, CallingCycle = 1, i;
    long word;

#ifdef VERBOSE
    printf("\nFinishing calling cycle....");
#endif
    while (CallingCycle) {
        word = NextWord();
        if (word == -1) {
            if (++LostWords > MaxLostWords)
              FAIL
        }
        else {
            LostWords = 0;
            switch (preamble = (word >> 21)) {
              case TIS: StoreBER(word);
                        for (i=0; i<49; i++) NextSym();    /* let Tlww expire */
                        return FindAddr(word);             /* conclusion: OK  */
              case TWAS: StoreBER(word);
                         /* doesn't count as a link */
                         FAIL
              case TO:  if ((word & 07777777) != callee->name)
                            FAIL                           /* changed addressee */
                        break;                    /* continue if same addressee */
              case CMD: CallingCycle = 0;                  /* begin msg section */
                        break;
              default: FAIL                       /* reject all other preambles */
            }
        }
    }
    /* must be in message section if get here */
    word = FinishMessage(word);   /* give word to message handler; it */
    /* returns first non-message word.  */
    if (word == -1) FAIL
    switch (preamble = (word >> 21)) {
      case TIS: StoreBER(word);
                for (i=0; i<49; i++) NextSym();    /* let Tlww expire */
                return FindAddr(word);             /* conclusion: OK  */
      case TWAS: StoreBER(word);
                 /* doesn't count as a link */
      default:  FAIL                    /* reject all other preambles */
    }
}
```

```
struct addr *Handshake(other)
struct addr *other;
{
    int preamble, CallingCycle = 1, i;
    long word1, word2, word3;

#ifdef VERBOSE
    printf("\n\nNext frame...");
#endif
    NextFrame();

    if (LQArequested) MakeLQAword(other); /* make LQA word using measured BER */

    initRxFEC(2*Trw);
    if (((word1 = WordSync()) >> 21) != TO) { /* get first word of resp. */
        if (word1 != -1)
          LogSound(word1);
        FAIL
    }
#ifdef VERBOSE
    printf("TO %s", AddrString(word1));
#endif
    if ((word1 & 07777777) != other->name) FAIL   /* wrong addressee */

    if ((word2 = NextWord()) != word1) /* get second word of response... */
      FAIL                             /* FAIL if it doesn't match first */

    while (CallingCycle && ((word3 = NextWord()) != -1)) {
        /* the 'while' construction allows a long calling cycle, */
        /* e.g., for tune time */
        switch (preamble = (word3 >> 21)) {
          case TIS: StoreBER(word3);
                    for (i=0; i<49; i++) NextSym();     /* let Tlww expire */
                    return FindAddr(word3);             /* conclusion: OK  */
          case TWAS: StoreBER(word3);
                     /* doesn't count as a link */
                     FAIL
          case TO:  if ((word3) != other->name)
                    FAIL                                /* changed addressee */
                    break;                    /* continue if same addressee */
          case CMD: CallingCycle = 0;
                    break;                              /* go handle message */
          default:  FAIL          /* reject all other preambles */
        }
    }

    /* must be in message section if get here */

    word3 = FinishMessage(word3); /* give word to message handler; it */
                                  /* returns first non-message word.  */
    if (word3 == -1) FAIL
    switch (preamble = (word3 >> 21)) {
      case TIS: StoreBER(word3);
                for (i=0; i<49; i++) NextSym();     /* let Tlww expire */
                return FindAddr(word3);             /* conclusion: OK  */
      case TWAS: StoreBER(word3);
                 /* doesn't count as a link */
      default:  FAIL                    /* reject all other preambles */
    }
}
```

```
#define NetMem(net, mem)   (net == mem->net)

int SlottedResp(net, caller)
     struct addr *net, *caller;
{
    int preamble, CallingCycle = 1, i, responses = 0;
    long word1, word2, word3, StartSlots;

#ifdef VERBOSE
    printf("\n\nSlotted Responses...");
#endif
    NextFrame();

    if (LQArequested) MakeLQAword(caller); /* make LQA word using measured BER */

    StartSlots = BaudCount;
    initRxFEC(Twrn);

    while (BaudCount < StartSlots + Twrn) {
        if (((word1 = WordSync()) >> 21) != TO) {     /* wait for a response */
            if (word1 == -1) break;
            LogSound(word1);
            IgnoreFrame();
            continue;
        }
#ifdef VERBOSE
        printf("TO %s", AddrString(word1));
#endif
        if ((word1 & 07777777) != caller->name) {
            IgnoreFrame();    /* not a response to net call */
            continue;
        }

        if ((word2 = NextWord()) != word1) { /* get second word of response..*/
            IgnoreFrame();            /* FAIL if it doesn't match first word */
            continue;
        }

        if (CallingCycle && ((word3 = NextWord()) != -1)) {
            /* NOTE:  calling cycle > 2 Trw fails */
            switch (preamble = (word3 >> 21)) {
              case TIS: StoreBER(word3);
                        for (i=0; i<49; i++) NextSym(); /* let Tlww expire */
                        if (NetMem(net, FindAddr(word3))) {
                            responses++;
#ifdef VERBOSE
                            printf("Response from %s", AddrString(word3));
#endif
                        }
                case TWAS: StoreBER(word3);
                           /* doesn't count as a link */
                           IgnoreFrame();
                           break;                  /* wait for more responses */
                case TO:  IgnoreFrame();
                          break;                   /* wait for more responses */
                case CMD: CallingCycle = 0;
                          break;                        /* go handle message */
                default:  IgnoreFrame();        /* reject all other preambles */
            }
        }
```

```
        if (preamble == CMD) {
            word3 = FinishMessage(word3); /* give word to message handler; */
                                          /* it returns first non-message word.*/
            if (word3 == -1) IgnoreFrame();
            switch (preamble = (word3 >> 21)) {
              case TIS: StoreBER(word3);
                        for (i=0; i<49; i++) NextSym(); /* let Tlww expire */
                        if (NetMem(net, FindAddr(word3))) {
                            responses++;
#ifdef VERBOSE
                            printf("Response from %s", AddrString(word3));
#endif
                        }
                        break;
              case TWAS: StoreBER(word3);
                         /* doesn't count as a link */
              default:  IgnoreFrame();
            }
        }
    }
#ifdef VERBOSE
    printf("(WordSync is expected to fail after the last slotted response.)");
    printf("\nResponses = %d", responses);
#endif
    return responses;
}

int SelfAddr(a)
struct addr *a;
{
    if (!a) return 0;
#ifdef VERBOSE
    if (a->class == SELF)
      printf("%s", AddrString(a->name));
#endif
    return (a->class == SELF);
}

int OtherAddr(a)
struct addr *a;
{
    if (!a) return 0;
#ifdef VERBOSE
    if (a->class == OTHER)
      printf("OTHER address (%s)", AddrString(a->name));
    else printf("not OTHER address");
#endif
    return (a->class == OTHER);
}

int NetAddr(a)
struct addr *a;
{
    if (!a) return 0;
#ifdef VERBOSE
    if (a->class == NET)
      printf("%s", AddrString(a->name));
#endif
    return (a->class == NET);
}
```

```
long links = 0;

void RxALE()
{
    struct addr *callee, *caller;
    int failed = 1;

    if (SelfAddr(callee = StopScan())) {
        if (OtherAddr(caller = RcvCall(callee)))
          if (Handshake(caller) == callee)
            if (Handshake(callee) == caller) {
                links++;
                failed = 0;
#ifdef VERBOSE
                printf("\007\n\nLinked with %s", AddrString(caller->name));
                if (retries) printf(" after %ld retries", retries);
                printf("\n");
#else
                if (retries) printf("%ld", retries);
                else printf(".");
#endif
            }
    }
    else if (NetAddr(callee)) {
        if (OtherAddr(caller = RcvCall(callee)))
          if (SlottedResp(callee, caller))  /* callee = NET; caller = NCS */
            if (Handshake(callee) == caller) {
                links++;
                failed = 0;
#ifdef VERBOSE
                printf("\007\n\nLinked with %s", AddrString(caller->name));
                if (retries) printf(" after %ld retries", retries);
                printf("\n");
#else
                if (retries) printf("%ld", retries);
                else printf(".");
#endif
            }
    }
    else {
#ifdef VERBOSE
        printf("***");
#endif
        FinishTransmissions();
    }
#ifdef VERBOSE
    if (failed) printf("\nLink failed.");
#else
    if (failed) printf("*");
#endif
    retries = 0;
}
```

```
#define NetCallBit 1
#define LQAbit 2
#define AMDbit 4

void GetParms(argc, argv, attempts) /* delay, spread, SNR, length */
int argc;
char **argv;
long *attempts;
{
    int SimSpecs;    /* bit 0 => net; else indiv  bit 1 => LQA  bit 2 => AMD */
    char c;

    if (argc > 1) sscanf(argv[1], "%lf", &DlyTime);
    else {
        printf("Enter multipath delay (ms): ");
        scanf("%lf", &DlyTime);
    }
    if (argc > 2) sscanf(argv[2], "%lf", &FreqSpread);
    else {
        printf("Enter Doppler spread (Hz): ");
        scanf("%lf", &FreqSpread);
    }
    if (argc > 3) sscanf(argv[3], "%lf", &SigLev);
    else {
        printf("Enter SNR (dB): ");
        scanf("%lf", &SigLev);
    }
    if(argc > 4) sscanf(argv[4], "%ld", attempts);
    else {
        printf("Number of links to attempt: ");
        scanf("%ld", attempts);
    }
    if(argc > 5) sscanf(argv[5], "%d", &SimSpecs);
    else {
        printf("Individual (i) or Net (n) calls? ");
        do {
            scanf("%c", &c);
            switch (c) {
              case 'I':
              case 'i': SimSpecs = 0; break;
              case 'N':
              case 'n': SimSpecs = 1; break;
              default:  if (c >= ' ') printf("\008");
                        c = '\000';
            }
        } while (!c);
        printf("Use LQA? (y or n): ");
        do {
            scanf("%c", &c);
            switch (c) {
              case 'y': SimSpecs |= LQAbit;
              case 'n': break;
              case 'Y': SimSpecs |= LQAbit;
              case 'N': break;
              default:  if (c >= ' ') printf("\008");
                        c = '\000';
            }
        } while (!c);
```

```
        printf("Use AMD? (y or n): ");
        do {
            scanf("%c", &c);
            switch (c) {
              case 'y': SimSpecs |= AMDbit;
              case 'n': break;
              case 'Y': SimSpecs |= AMDbit;
              case 'N': break;
              default:  if (c >= ' ') printf("\008");  c = '\000';
            }
        } while (!c);
    }
    NetCalls = SimSpecs & NetCallBit;
    LQA = SimSpecs & LQAbit;
    AMD = SimSpecs & AMDbit;

    switch (SimSpecs) {
      case 0: printf("\nIndividual Calls"); break;
      case 1: printf("\nNet Calls"); break;
      case 2: printf("\nIndividual Calls with LQA"); break;
      case 3: printf("\nNet Calls with LQA"); break;
      case 4: printf("\nIndividual Calls with AMD"); break;
      case 5: printf("\nNet Calls with AMD"); break;
      case 6: printf("\nIndividual Calls with LQA and AMD"); break;
      case 7: printf("\nNet Calls with LQA and AMD"); break;
      default: printf("\nBad combination of options."); exit();
    }
    SigLev = pow(10.0, SigLev/20.0);/* convert from dB to voltage ratio */
    TxLevel = SigLev;
    printf("\nDelay = %7.4f ms, Signal Level = %8.4f", DlyTime, SigLev);
    printf("\nDoppler spread = %7.41f Hz; Run length = %ld\n",
              FreqSpread, *attempts);
}

extern long time();

main(argc, argv)
     int argc;
     char **argv;
{
    long attempts, i, StartTime, EndTime;

    GetParms(argc, argv, &attempts);
    Initialize();
    InitModem();
    SetUpNet();
    if (NetCalls) SetUpNetCall();
    else SetUpIndivCall();

    StartTime = time( (long *) 0);
    for (i=0; i<attempts; i++) {
#ifndef VERBOSE
        if (!(i%10)) printf("\n");
#endif
        StartHandshake();
        RxALE();
    }
    EndTime = time( (long *) 0);
    printf("\nP(link) = %lf\n", (double) links / attempts);
    printf("\007Time per linking attempt = %lf seconds.\n",
           (double)(EndTime - StartTime)/attempts);
}
```

```
/*  addr.c          eej          8/4/91  */
/*  1045 simulator addr memory data base  */

struct addr {
    struct addr *next, *net, *member;
    long name;
    short class, index; } ;

#define HashBins 16
#define HashMask 0x0f
#define SELF     0
#define NET      1
#define OTHER    2
#define MaxAddresses 20

static struct addr *AddrMemory[HashBins];
static int LQAindex = 0;

char *AddrString(w)  /* returns a four-byte string containing the three */
     long w;         /* ASCII characters in the ALE word, followed by 0 */
{
    static long NameWord;

    NameWord = ((w & 07740000) << 10) | ((w & 037600) << 9) | ((w & 0177) << 8);
    return (char *) &NameWord;
}

long ThreeChar(s)  /* returns an ALE word containing the first three */
     char *s;      /* ASCII characters in the char string argument   */
{                  /*  (pads with @ characters if string too short)  */
    if (!(s[0] &= 0177)) return ((long)'@' << 14) + ((long)'@' << 7) + '@';
    else if (!(s[1]&=0177)) return ((long)s[0] << 14) + ((long)'@' << 7) + '@';
    else if (!(s[2]&=0177)) return ((long)s[0] << 14) + ((long)s[1] << 7) + '@';
    else return ((long)s[0] << 14) + ((long)s[1] << 7) + s[2];
}

long AMDchar(s)    /* returns an ALE word containing the first three */
     char *s;      /* ASCII characters in the char string argument   */
{                  /*  (pads with blanks if string too short)        */
    if (!(s[0] &= 0177)) return ((long)' ' << 14) + ((long)' ' << 7) + ' ';
    else if (!(s[1]&=0177)) return ((long)s[0] << 14) + ((long)' ' << 7) + ' ';
    else if (!(s[2]&=0177)) return ((long)s[0] << 14) + ((long)s[1] << 7) + ' ';
    else return ((long)s[0] << 14) + ((long)s[1] << 7) + s[2];
}

void initAddrMemory()
{
    int i;
    for (i=0; i<HashBins; i++) AddrMemory[i] = (struct addr *) 0;
}

struct addr *FindAddr(name)
     long name;
{
    register struct addr *cursor;
    name &= 07777777;  /* strip off preamble */
    if (cursor = AddrMemory[name & HashMask])
      do {
          if (cursor->name == name) break;
      } while (cursor = cursor->next);
    return cursor;
}
```

```
void InsertAddr(name, class)        /* add a new address to data base */
     long name;                     /* (bins are lexically ordered)   */
     short class;                   /*    OK   7/29/91                */
{
    register struct addr *cursor, **prev;
    long w = name;

#ifdef VERBOSE
    printf("\nInserting %lo (%s) in bin %ld",
           name, AddrString(w), name & HashMask);
#endif

    prev = AddrMemory + (name & HashMask);
    if (cursor = *prev)                    /* if bin not empty */
      do {
          if (cursor->name == name) return;      /* found name?           */
          if (cursor->name > name) break;        /* past insertion point? */
          prev = cursor;
      } while (cursor = cursor->next);
    *prev = (struct addr *) malloc(sizeof(struct addr));
    (*prev)->next = cursor;
    cursor = *prev;
    cursor->name = name;
    cursor->class = class;
    if ((cursor->index = LQAindex++) >= MaxAddresses) {
        printf("\nToo many addresses.\n");
        LQAindex--;
    }
    cursor->member = cursor->net = (struct addr *) 0;
}

int AddMember(NetName, NewMember)
     long NetName, NewMember;
{
    struct addr *net, *cursor;
    register struct addr *temp;

#ifdef VERBOSE
    printf("\nAdding member \"%s\"", AddrString(NewMember));
    printf(" to net \"%s\"", AddrString(NetName));
#endif

    if ((net = FindAddr(NetName)) == (struct addr *) 0) return 0;
    cursor = net;
    while (temp = cursor->member)     /* find last member */
      cursor = temp;
    if (cursor->member = FindAddr(NewMember)) {  /* if NewMember exists */
        temp = cursor->member;
        temp->member = (struct addr *) 0;
        temp->net = net;
    }
}
```

```
void PrintAddrMemory()
{
    int i;
    struct addr *cursor, *member;
    long w;

    printf("\n\nNon-empty bins of Address Memory\n");
    for (i=0; i<HashBins; i++) {
        if (cursor = AddrMemory[i]) {
          printf("\n%4d    ", i);
          while (cursor) {
            w = cursor->name;
            printf("(%s, %d, %d)", AddrString(w), cursor->class, cursor->index);
            if (cursor->class == NET) {
                member = cursor;
                while (member = member->member)
                  printf(" -> (%s, %d)", AddrString(member->name),
                                         member->class);
            }
            if (cursor->net) printf("  <in net %s>",
                                    AddrString(cursor->net->name));
            cursor = cursor->next;
            printf("\n        ");
          }
        }
    }
}
```

```
/*  message.c    eej     8/4/91  */
/*      handle message sections  */
/*      in ALE handshakes        */
/*                               */
/*  addr.c must be #included     */
/*   before this file            */

#define MEASURED 0    /* used to select appropriate entry in LQA matrix */
#define REPORTED 1
#define NoReport 017777

static short LQAmatrix[MaxAddresses][2];
static short LQArequested = 0,
             LQAreported = NoReport ;  /* LQA reports stored here until */
                                       /* sender address known          */
void initLQAmatrix()
{
    int i;
    for (i=0; i<MaxAddresses; i++)
      LQAmatrix[i][MEASURED] = LQAmatrix[i][REPORTED] = NoReport;
    /* LQAword is external, defined in ale.c */
    LQAword = (char *) malloc(49 * sizeof(char));
}

void DisplayAMD(word)
     long word;
{
    if (word == -1) printf("***");     /* no "block" character available */
    else printf("%s", AddrString(word));
}

void MakeLQAword(other)
     struct addr *other; /* address of station to receive LQA report */
{                        /* LQAword is external, declared in ale.c   */
    long CMDword;

#ifdef VERBOSE
    printf("\n");
#endif
    CMDword = _CMD + _LQA;
    if (other) {
        if (LQAmatrix[other->index][REPORTED] == NoReport) {
#ifdef VERBOSE
            printf("(requesting LQA report)    ");
#endif
            CMDword += _LQAreq;  /* request LQA if don't have recent data */
        }
        if (LQArequested) {
#ifdef VERBOSE
            printf("(LQA was requested)    ");
#endif
            CMDword += LQAmatrix[other->index][MEASURED];
            LQArequested = 0;
        }
        else CMDword += NoReport;
    }
    else CMDword += NoReport + _LQAreq;
#ifdef VERBOSE
    printf("LQAword = %08lo    ", CMDword);
#endif
    LQAword = Tx(CMDword, LQAword);
}
```

```
long FinishMessage(word)
     long word;
{
    char command, CmdPreamble = CMD;

#ifdef VERBOSE
    printf("\n  Message(s): ");
#endif
    while (((word >> 21) & 7) == CmdPreamble) {
        command = (word >> 14) & 0x7f;
        if (ASCII_Set[command] >= ASCII_64) { /* Automatic Msg Display mode  */
            printf("AMD message \"");
            CmdPreamble = DATA ^ 7;  /* second AMD word uses DATA preamble */
            do {
                DisplayAMD(word);
                CmdPreamble ^= 7;    /* alternate preambles btwn DATA and REP */
                word = NextWord();
            } while ((word == -1) | (((word >> 21) & 7) == CmdPreamble));
            CmdPreamble = CMD;  /* prepare for another command */
            printf("\"    ");
            continue;  /* skip to outer 'while' condition evaluation */
        }
        else switch (command) {
          case 'a': LQArequested = word & _LQAreq;
            LQAreported = word & 017777;
#ifdef VERBOSE
            if (LQAreported != NoReport) printf("LQA = %d    ", LQAreported);
            else if (LQArequested) printf("LQA request    ");
#endif
            CmdPreamble ^= 1;  /* alternate preambles btwn CMD and REP */
            word = NextWord();
            break;
          default:  /* ignore */
#ifdef VERBOSE
            printf("??? (%08lo)    ", word);
#endif
            CmdPreamble ^= 1;  /* alternate preambles btwn CMD and REP */
            word = NextWord();
            break;
        }
    }
    return word;
}
```

```
void StoreBER(word)    /* word should contain the conclusion of a frame, */
     long word;        /*  including the address of the sending station  */
{
    struct addr *sender;
    int PBER;

    PBER = 48 - (TotalUcount/WordCount);
    if (PBER > 30) PBER = 30;
#ifdef VERBOSE
    printf("\n    PBER = %d    ", PBER);
#endif
    if (sender = FindAddr(word)) {
#ifdef VERBOSE
      printf("\n    Storing PBER values for %s.  Measured: %d, Reported: %d    ",
              AddrString(sender->name), PBER, LQAreported & 037);
#endif
        LQAmatrix[sender->index][MEASURED] = PBER;
        if (LQAreported != NoReport)
          LQAmatrix[sender->index][REPORTED] = LQAreported;
    }
    LQAreported = NoReport;
}
```

```
/*  fec.c    eej, rsm    rev. 8/4/91  */

#include "localdef.h"
#include "unanimous.h"
#include "majority.h"

/*#define VERBOSE*/

#include "golay.c"       /*  Golay encode and decode routines      */

#define SR1LEN  49

int Ucount;
char Sync;

static int SymbolsToGo;
static int maj = 0;
static short sr1[SR1LEN];                          /* Symbol buffer */
static int ptr0, ptr1a, ptr1b, ptr2a, ptr2b; /* pointers to sr1 */
static int ubuffer[16];                            /* buffer of unanimous votes */
static int uptr;                                   /* pointer in Ubuffer */
static short pingpong;                             /* deinterleaving buffer choice */
static long g1,g2;                                 /* Golay words */

extern char NextSym();  /*  in ale.c  */

void Golay(w, G)
long w, *G;
{
#ifdef VERBOSE
    printf("\nInputs to Golay:  %08lo %08lo", w>>12, w&07777);
#endif
    G[0] = encode(w >> 12);
    G[1] = encode(w & 07777);
}

long DeGolay (G0, G1, power)
     long G0, G1;
     int power;
{
#ifdef VERBOSE
    printf("\n        %08lo  %08lo    ", G0, G1);
#endif
    if ((G0 = decode(G0, power)) == -1) return (-1);
    if ((G1 = decode(G1, power)) == -1) return (-1);
#ifdef VERBOSE
    printf("%08lo    Ucount = %d    ", (G0<<12) | G1, Ucount);
#endif
    return (G0<<12) | G1;
}
```

```
char *Tx(ALE_WORD, Txw)          /* new 7/14/91 */
     long ALE_WORD;
     char *Txw;
{
    long G[2];          /*      two 24-bit Golay code words      */
    register int i;
    register long a, b;

    Golay(ALE_WORD, G);

    a = (G[0]) << 2;
    b = (G[1] ^= 07777) << 1;  /*        invert B word check bits */

    if (!Txw) Txw = (char *) malloc(49 * sizeof(char)); /* allocate if needed */
#ifdef VERBOSE
    printf("\nNew Tx word %08lo at %lx", ALE_WORD, Txw);
#endif

    /*****  INTERLEAVING AND REDUNDANCY  *****/

    Txw[48] = a&4 | b&2;
    for (i=46; i>33; i -= 2) {
        Txw[i] = a&040 | b&020;
        a >>= 1;         b >>= 1;
        Txw[i] = (Txw[i] | a&010) >> 3;
        Txw[i+1] = b&4;
        a >>= 1;         b >>= 1;
        Txw[i+1] |= a&2 | b&1;
        a >>= 1;         b >>= 1;
    }
    a |= (G[1] << 5);
    b |= (G[0] << 5)     /*      stuff bit for free       */;
    for (i=32; i>16; i -= 2) {
        Txw[i] = a&040 | b&020;
        a >>= 1;         b >>= 1;
        Txw[i] = (Txw[i] | a&010) >> 3;
        Txw[i+1] = b&4;
        a >>= 1;         b >>= 1;
        Txw[i+1] |= a&2 | b&1;
        a >>= 1;         b >>= 1;
    }
    a |= (G[0] << 6);
    b |= (G[1] << 5)     /*      stuff bit for free       */;
    for (i=16; i>=0; i -= 2) {
        Txw[i] = a&040 | b&020;
        a >>= 1;         b >>= 1;
        Txw[i] = (Txw[i] | a&010) >> 3;
        Txw[i+1] = b&4;
        a >>= 1;         b >>= 1;
        Txw[i+1] |= a&2 | b&1;
        a >>= 1;         b >>= 1;
    }
    return Txw;
}
```

```
initRxFEC(dwell)          /* Initialize Routine */
     int dwell;           /* simulates initial receiver actions in arriving */
{                         /* on a channel carrying ALE tones and timing     */
    int sr1end  =  SR1LEN;
    int i, tmp, index;

    pingpong = 0;                  /* initialize pointers */
    ptr0 = 0;
    ptr1a = 16;
    ptr1b = 17;
    ptr2a = 32;
    ptr2b= 33;
    uptr = 0;
    g1 = g2 = 0.0;
    for (i=0;i<16;i++)    /* initialize unanimous vote circular buffer */
      ubuffer[i] = 0;     /*   (stores unanimous vote counts for 16    */
                          /*    most recent majority votes.)           */
    Ucount = 0;

    for (i=0;i<SR1LEN;i++) sr1[i] = 0; /* init received-symbol shift rgtr */

    SymbolsToGo = dwell;   /* number of symbols allowed for word sync acq. */

    for (i=0;i<SR1LEN-1;i++) {  /* read first 48 symbols into shift register */
        SymbolsToGo--;
        index = ((*(sr1+ptr1a) & 3) << 7) | ((*(sr1+ptr1b) & 4) << 4) |
          ((*(sr1+ptr2a) & 1) << 5) | ((*(sr1+ptr2b) & 6) << 2) |
            (*(sr1+ptr0) = NextSym()); /* concatenate three redundant symbols */
                                       /* to make an index for table accesses */

        /* Unanimous Votes */
        tmp = ubuffer[uptr];                    /* copy value to be shifted out*/
        Ucount += utable[index]-ubuffer[uptr]; /* update unanimous vote count */
        ubuffer[uptr] = utable[index];          /* shift old value out, new in */

        /* Majority Votes and Deinterleaving */
        maj = mtable[index];
        if (pingpong) {
            g1 = ((g1 << 2) | (((maj&4) >> 1) | (maj&1)));
            g2 = ((g2 << 1) | ((maj&2) >> 1));
        }
        else {
            g2 = ((g2 << 2) | (((maj&4) >> 1) | (maj&1)));
            g1 = ((g1 << 1) | ((maj&2) >> 1));
        }

        /* update pointers to circular buffers */
        pingpong = (++pingpong > 1) ? 0 : pingpong;
        ptr0  = (++ptr0  >= sr1end) ? 0 : ptr0;
        ptr1a = (++ptr1a >= sr1end) ? 0 : ptr1a;
        ptr1b = (++ptr1b >= sr1end) ? 0 : ptr1b;
        ptr2a = (++ptr2a >= sr1end) ? 0 : ptr2a;
        ptr2b = (++ptr2b >= sr1end) ? 0 : ptr2b;
        uptr = ++uptr&0xf;
    }
    Sync = 0;                      /* clear word sync flag */
    return 1;
}
```

```
#define THRESHOLD 0        /* this needs to be tuned */

long RxFEC(power)
     int power;
{
    int index;
    int srlend  =  SR1LEN;
    int i, j, tmp;
    long gol;              /* Golay decoded word */

    if (Sync) {                    /* just get next 49 symbols */
        for (i=0;i<SR1LEN;i++) {
            index = ((*(sr1+ptr1a) & 3) << 7) | ((*(sr1+ptr1b) & 4) << 4) |
              ((*(sr1+ptr2a) & 1) << 5) | ((*(sr1+ptr2b) & 6) << 2) |
                (*(sr1+ptr0) = NextSym());

            /* Unanimous Votes */
            tmp = ubuffer[uptr];
            Ucount += utable[index] - ubuffer[uptr];
            ubuffer[uptr] = utable[index];

            /* Majority Votes and Deinterleaving */
            maj = mtable[index];
            if (pingpong) {
                g1 = ((g1 << 2) | (((maj&4) >> 1) | (maj&1)));
                g2 = ((g2 << 1) | ((maj&2) >> 1));
            }
            else {
                g2 = ((g2 << 2) | (((maj&4) >> 1) | (maj&1)));
                g1 = ((g1 << 1) | ((maj&2) >> 1));
            }

            /* update pointers to circular buffers */
            pingpong = (++pingpong > 1) ? 0 : pingpong;
            ptr0  = (++ptr0  >= srlend) ? 0 : ptr0;
            ptr1a = (++ptr1a >= srlend) ? 0 : ptr1a;
            ptr1b = (++ptr1b >= srlend) ? 0 : ptr1b;
            ptr2a = (++ptr2a >= srlend) ? 0 : ptr2a;
            ptr2b = (++ptr2b >= srlend) ? 0 : ptr2b;
            uptr = ++uptr&0xf;
        }
        if (pingpong) /* These are reversed due to pingpong being inc */
          return(DeGolay(((g2>>1)&0xffffff),((g1^0xfff)&0xffffff),power));
        else
          return(DeGolay(((g1>>1)&0xffffff),((g2^0xfff)&0xffffff),power));
    }
```

```
    else  /* looking for word sync */
      while(!Sync && (SymbolsToGo--)) {

          /* Compute Index into majority vote and unanimous vote tables */
          index = ((*(sr1+ptr1a) & 3) << 7) | ((*(sr1+ptr1b) & 4) << 4) |
            ((*(sr1+ptr2a) & 1) << 5) | ((*(sr1+ptr2b) & 6) << 2) |
              (*(sr1+ptr0) = NextSym());

          /* Unanimous Votes */
          tmp = ubuffer[uptr];
          Ucount += utable[index] - ubuffer[uptr];
          ubuffer[uptr] = utable[index];

          /* Majority Votes and Deinterleaving */
          maj = mtable[index];
          if (pingpong) {
              g1 = ((g1 << 2) | (((maj&4) >> 1) | (maj&1)));
              g2 = ((g2 << 1) | ((maj&2) >> 1));
              gol = DeGolay(((g1>>1)&0xffffff),((g2^0xfff)&0xffffff),power);
          }
          else {
              g2 = ((g2 << 2) | (((maj&4) >> 1) | (maj&1)));
              g1 = ((g1 << 1) | ((maj&2) >> 1));
              gol = DeGolay(((g2>>1)&0xffffff),((g1^0xfff)&0xffffff),power);
          }
          pingpong = (++pingpong > 1) ? 0 : pingpong;

          /* update pointers to circular buffers */
          ptr0  = (++ptr0  >= sr1end) ? 0 : ptr0;
          ptr1a = (++ptr1a >= sr1end) ? 0 : ptr1a;
          ptr1b = (++ptr1b >= sr1end) ? 0 : ptr1b;
          ptr2a = (++ptr2a >= sr1end) ? 0 : ptr2a;
          ptr2b = (++ptr2b >= sr1end) ? 0 : ptr2b;
          uptr = ++uptr&0xf;

          if ((gol != (long) -1) && (Ucount > THRESHOLD)){
              Sync = 1;           /* got good golay; word sync is possible */
              return (gol);
          }
      }
    return (-1);                          /* No good golay word recieved */
}
```

```
/*  golay.c     eej     rev. 7/28/91                                    */
/*  routines to execute the FED-STD-1045 / MIL-STD-188-141A */
/*  Golay encode and decode process                                     */

#include <stdio.h>

/*********************   GOLAY ENCODER MODULE   **************************/


long encode(w)
long w;
{
    long i, word = 0;
        static long G[12] = { 040005343, 020007622, 010006453,    /* generator
*/
                              004006166, 002006331, 001003155,    /*  matrix   */
                              000401467, 000205570, 000102674,
                              000041336, 000025615, 000012707 };

    if (w & 1) word ^= G[11];
    for (i = 10; i >= 0; i--)
        if ((w >>= 1) & 1) word ^= G[i];
    return (word);
}


/*********************  GOLAY DECODER MODULE    **************************/

#include "errwt.h"   /*  error weight matrix   "int wt[4096] = { ... }"  */
#include "errpat.h"  /*  error pattern matrix  "int  e[4096] = { ... }"  */

long decode(w, power)
long w, power;
{
  register int syndrome;

  if ((wt[syndrome = (encode(w>>12) ^ w) & 07777]) > power) return (-1);
  else return (w>>12 ^ e[syndrome]);
}
```

```
/*  ALEmodem.c          eej      6/13/91          */
/*          8-ary FSK modem for HF ALE simulator */

#include <math.h>
#include "hfdef.h"
#define m_ary 8

static double tone[m_ary][2][NPTS];

void InitModem()
{
  register int i, iprime, j;
  double f[m_ary], dt;
  register double x, dx;

  f[0] = -(f[7] = 875.0);        /* 8-ary FSK tone frequencies */
  f[1] = -(f[6] = 625.0);
  f[2] = -(f[5] = 375.0);
  f[3] = -(f[4] = 125.0);

  dt = Ts/NPTS;

  for (i=0; i<m_ary/2; i++) {    /* generate tone vectors    */
      iprime = m_ary-i-1;        /* index of conjugate frequency */
      x = 0.0;
      dx = TPi*f[iprime]*dt;
      for (j=0; j<NPTS; j++) {
          tone[i][0][j] = tone[iprime][0][j] = cos(x);  /* real part of tone */
          tone[i][1][j] = -(tone[iprime][1][j] = sin(x));/* imag part of tone*/
          x += dx;       if (x >= TPi) x -= TPi;
      }
  }
}

SymVect *FSKmod(InSym)           /*  use: SymVect *XmitSig          */
     int InSym;                  /*       XmitSig = FSKmod(InSym)   */
{                                /*            ... (*XmitSig)[i][j] */
  return (tone[InSym]);
}

int FSKdemod(RcvSig)
     SymVect *RcvSig;
{
  register int i, j, out;
  register double sumi, sumq, sum, max=0.0;
  register SymVect *in;

  in = RcvSig;
  for (i=0; i<m_ary; i++)  {
      sumi = sumq = 0.0;
      for (j=NPTS/4; j<(3*NPTS)/4; j++)  {
          sumi += (*in)[0][j]*tone[i][0][j] + (*in)[1][j]*tone[i][1][j];
          sumq += (*in)[1][j]*tone[i][0][j] - (*in)[0][j]*tone[i][1][j];
      }
      if ((sum = sumi*sumi + sumq*sumq) >= max) {
          max = sum;
          out = i;
      }
  }
  return (out);
}
```

```
/*      hflib.c          routines for HF channel simulator   */
/*      eej                  rev. 8/3/91                     */

#include "hfdef.h"
#include <stdio.h>
#include <math.h>

#define sqrt2 1.414213562


/*****************************  UTILITIES  ******************************/

/*  uniform.c       eej         8/12/89                     */
/*  random number generator with uniform distribution       */
/*  from CACM June 1988, Vol 31, No 6, pp. 742-749+         */

double uniform()
{
    static long s1 = 123456787, s2 = 987654323;
    long z, k;

    k = s1 / 53668;
    s1 = 40014 * (s1 - k * 53668) - (k * 12211);
    if (s1 < 0) s1 += 2147483563;

    k = s2 / 52774;
    s2 = 40692 * (s2 - k * 52774) - (k * 3791);
    if (s2 < 0) s2 += 2147483399;

    z = s1 - s2;
    if (z < 1) z += 2147483562;

    return (z * 4.656613E-10);
}

int RandInt(i)  /* Generates random integers in the range 0 to i-1.     */
    int i;      /* In run of 100,000 with 32 bins, had variation of     */
{               /* 6% of mean from max bin to min bin.                  */
    return (i*uniform());
}

int hammingwt3(w)
int w;
{
    static int wt[8] = {0, 1, 1, 2, 1, 2, 2, 3};

    return (wt[w]);
}
```

```
/*********************  DELAY LINE MODULE  *********************************/

double DlyTime;          /* global so it can be manipulated in main routine */
static int DelayLineLen;         /* if more than one set of delay lines is  */
static double DlyTF;             /* needed, must pass these as parms        */

static int iptr, qptr;           /* these are passed as parms */
static double IDLine[65], QDLine[65];

double DelayLine(InSig, Ptr, LName)
     double InSig;     /* current sample: put into delay line */
     int *Ptr;         /* current offset into delay line      */
     double *LName;    /* ptr to delay line to use            */
{
    register double *tempptr, temp;
    register int nt;

    if (DelayLineLen > 1) {  /* use the delay line */
        nt = (++(*Ptr)+1) % DelayLineLen;
        tempptr = LName + (*Ptr %= DelayLineLen);
        temp = *tempptr * DlyTF + (*(LName+nt)) * (1.0 - DlyTF);
        *tempptr = InSig;
    }
    else {  /* use only first element of delay line */
        temp = InSig * (1.0 - DlyTF) + *LName * DlyTF;
        *LName = InSig;
    }
    return (temp);
}


/*********************  RAYLEIGH (NOISE) GENERATOR  ***************************/

void Rayleigh (x, y)
     double *x, *y;
{
    register double r, a;

    r = sqrt(-2.0*log(uniform()));
    a = TPi * uniform();
    *x = r * cos(a);
    *y = r * sin(a);
}
```

```
/********************  FADING GAINS MODULE  *********************************/

double FreqSpread;          /* global so it can be manipulated in main routine */
static double  g, a0, a1, a2;
static double  IFade0[6], QFade0[6], /* direct-path fading filter state vars */
               IFade1[6], QFade1[6];/* delayed-path fading filter state vars */

void Gauss_Filter(Fade)
     double *Fade;
{
    /* Fade array of input/output data:              */
    /*  Fade[0] is the current filter output         */
    /*  Fade[0-2] are the current and past outputs   */
    /*  Fade[3-5] are the current and past inputs    */

    /* Gaussian filter:  2-pole, 2-zero IIR */
    Fade[0]=(g*(Fade[3]+2*Fade[4]+Fade[5])-a1*Fade[1]-a2*Fade[2])/a0;

    /* adjust the history terms */
    Fade[2]=Fade[1];
    Fade[1]=Fade[0];
    Fade[5]=Fade[4];
    Fade[4]=Fade[3];
}

void FadeGains(i, q)
     double *i, *q;
{
    /* generate Rayleigh-distributed fade gain perturbations */
    Rayleigh(i+3, q+3); /* assign to current input positions */

    /* Run through gaussian filter */
    Gauss_Filter(i);
    Gauss_Filter(q);
}

void CompMult (Xi, Xq, Yi, Yq, Zi, Zq)            /*  Z = X * Y    */
     double Xi, Xq, Yi, Yq, *Zi, *Zq;
{
    *Zi = Xi*Yi - Xq*Yq;
    *Zq = Xi*Yq + Xq*Yi;
}
```

```
/*********************  HF CHANNEL SIMULATION  ******************************/

double SigLev, TxLevel;            /* global so it can be set by main routine */
static SymVect RcvSig;             /* used to return resulting symbol vector  */

SymVect *HFchan(XmitSig)
     SymVect *XmitSig;
{
    register int i;
    register SymVect *in, *out;
    double   ni, nq, Ri, Rq, RDi, RDq, tmp1, tmp2;
    register double Xi, Xq, Di, Dq;  /* in-phase and quadrature main and   */
                                     /* delayed path samples, respectively */
    in = XmitSig;
    out = RcvSig;                    /* RcvSig is global */
    for (i=0; i<NPTS; i++) {
        Rayleigh(&ni, &nq);  /*                 GENERATE NOISE              */
                             /* Rayleigh generates in-phase and quadrature */
                             /* components, jointly normal, with each      */
                             /* component having RMS amplitude of unity;   */
                             /* RMS noise power is also unity.             */

        if (FreqSpread > 0.0) {          /*     GENERATE FADING GAINS        */
            FadeGains(IFade0, QFade0);    /* generate direct-path fade gains  */
            FadeGains(IFade1, QFade1);    /* generate delayed-path fade gains */
        }
        else {
            Rayleigh(&tmp1, &tmp2);       /* don't care (to draw RNs)         */
            Rayleigh(&tmp1, &tmp2);      /* leave fade gains at initialized  */
                                         /*   values (phase shift only)       */
        }

        if (DelayLineLen + DlyTF > 1.0) {       /*  multipath?  */
            Di = DelayLine(Xi=SigLev*(*in)[0][i], &iptr, IDLine);/* generate */
            Dq = DelayLine(Xq=SigLev*(*in)[1][i], &qptr, QDLine);/* delayed  */
                                                                 /* signals  */

            CompMult(Xi, Xq, *IFade0, *QFade0, &Ri, &Rq);/* introduce fading */
            CompMult(Di, Dq, *IFade1, *QFade1, &RDi, &RDq);
        }
        else {                      /* all power in single path */
            CompMult(SigLev*(*in)[0][i], SigLev*(*in)[1][i],
                     *IFade0, *QFade0, &Ri, &Rq);
            RDi = RDq = 0.0;
            Ri *= sqrt2;
            Rq *= sqrt2;
        }

        (*out)[0][i] = Ri + RDi + ni;       /* compute output */
        (*out)[1][i] = Rq + RDq + nq;
    }
    return (out);
}
```

```
/*********************  INITIALIZATION ***********************************/

void Initialize()
{
    register int i;
    double a, c, A, C, dt;

    /******************** SET UP DELAY LINES *********************/
/*    do {
      printf("Enter delay time (ms):  ");
      scanf("%lf", &DlyTime);
    } while (DlyTime < 0.0); */

    dt = Ts/NPTS;              /*  dt = sample period  */
    DlyTime /= (dt * 1e3);  /* scale DlyTime from milliseconds to samples */
    DlyTF = DlyTime - (DelayLineLen = (int)DlyTime);
    DelayLineLen++;
    iptr = qptr = 0;
    for (i=0; i<65; i++) IDLine[i] = QDLine[i] = 0.0;


    /*****************  SET UP FADING GENERATOR  *****************/
/*    do {
      printf("Enter frequency spread (Hz):  ");
      scanf("%lf", &FreqSpread);
    } while (FreqSpread < 0.0); */

    if (FreqSpread > 0.0) {
      FreqSpread *= dt*TPi;   /*  scale from Hz to radians per sample  */

      /* magic numbers for Gaussian filter */
      a = sqrt(TPi);
      c = 1.5;
      A = a/FreqSpread;
      C = c/FreqSpread;    C *= C;

      /* Gaussian filter coefficients */
      g = sqrt(0.5*sqrt(TPi)/FreqSpread);
      a0 = A + C + 1.0;
      a1 = 2*(1.0 - C);
      a2 = C + 1.0 - A;

      for (i=0; i<6; i++)  /* clear fading filter state variables . . . */
        *(IFade0+i) = *(QFade0+i) = *(IFade1+i) = *(QFade1+i) = 0.0;

      for (i=0; i<1.0/FreqSpread; i++) {         /* and initialize them */
        FadeGains(IFade0, QFade0);
        FadeGains(IFade1, QFade1);
      }
    }
    else         /* fading generator disabled; set fixed gains for taps */
      *(IFade0) = *(QFade0) = *(IFade1) = *(QFade1) = sqrt2/2.0;
}
```

```
/*  ALEmodem.h          eej      6/16/91          */
/*        8-ary FSK modem for HF ALE simulator */

extern void        InitModem();
extern SymVect     *FSKmod();
extern int                 FSKdemod();

#define ALEmodem(x) FSKdemod(HFchan(FSKmod(x)))
```

```
/*        ALEwords.h                eej              8/4/91 */
/*        defines some useful ALE words     */

#define _TO         020000000       /* use these to build ALE words */
#define _TIS        050000000
#define _TWAS       030000000
#define _DATA       000000000
#define _REP        070000000
#define _CMD        060000000
#define _THRU       010000000
#define _FROM       040000000

#define TO          02              /* use these for tests and indices */
#define TIS         05
#define TWAS        03
#define DATA        00
#define REP         07
#define CMD         06
#define THRU        01
#define FROM        04

#define _SAM        005160315
#define _UEL        005261314

#define _JOE        004523705
#define _JOS        004523723
#define _EPH        004264110

#define _NET        004721324
#define _NCS        004720723

#define _MOE     004663705
#define _PAM     005020315
#define _RAM     005120315

#define _LQA     ('a' << 14)
#define _LQAreq 000020000
```

```
/*        ASCII_Set.h     eej     7/22/91  table indexed by ASCII character  */
/*                 to determine most restrictive character set it belongs to  */

#define ASCII_128  0
#define ASCII_64   1
#define ASCII_38   2

static char ASCII_Set[128] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                              1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                              2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 2,
                              2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                              2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1,
                              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

```
/*  fec.h    eej    7/26/91  */

extern char Sync;
extern int Ucount;
extern char *Tx();
extern long RxFEC();
extern void initRxFEC();
```

```
/*  hfdef.h     definitions for HF channel simulator routines   */
/*  eej         8/3/91                                          */

#include "localdef.h"
#define Pi  3.141592654
#define TPi 6.283185307
#define Ts  0.008
#define NPTS 32

typedef double  SymVect[2][NPTS];

extern double DlyTime, FreqSpread, SigLev, TxLevel;  /* declared in hflib.c */
```

```
/*  hflib.h      header file for HF propagation simulator routines  */
/*                      eej          7/30/91                          */

/* hfdef.h must be #included before this file */

extern double      uniform();
extern int         RandInt();
extern int         hammingwt3();

extern void        Initialize();
extern SymVect     *HFchan();
```

```
/*  majority.h     rsm      7/28/91      table of majority votes  */
/*                              indexed by concatenated tri-bits  */

short mtable[512] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1,
                      0, 0, 2, 2, 0, 0, 2, 2, 0, 1, 2, 3, 0, 1, 2, 3,
                      0, 0, 0, 0, 4, 4, 4, 4, 0, 1, 0, 1, 4, 5, 4, 5,
                      0, 0, 2, 2, 4, 4, 6, 6, 0, 1, 2, 3, 4, 5, 6, 7,
                      0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                      0, 1, 2, 3, 0, 1, 2, 3, 1, 1, 3, 3, 1, 1, 3, 3,
                      0, 1, 0, 1, 4, 5, 4, 5, 1, 1, 1, 1, 5, 5, 5, 5,
                      0, 1, 2, 3, 4, 5, 6, 7, 1, 1, 3, 3, 5, 5, 7, 7,
                      0, 0, 2, 2, 0, 0, 2, 2, 0, 1, 2, 3, 0, 1, 2, 3,
                      2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 3, 2, 3, 2, 3,
                      0, 0, 2, 2, 4, 4, 6, 6, 0, 1, 2, 3, 4, 5, 6, 7,
                      2, 2, 2, 2, 6, 6, 6, 6, 2, 3, 2, 3, 6, 7, 6, 7,
                      0, 1, 2, 3, 0, 1, 2, 3, 1, 1, 3, 3, 1, 1, 3, 3,
                      2, 3, 2, 3, 2, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                      0, 1, 2, 3, 4, 5, 6, 7, 1, 1, 3, 3, 5, 5, 7, 7,
                      2, 3, 2, 3, 6, 7, 6, 7, 3, 3, 3, 3, 7, 7, 7, 7,
                      0, 0, 0, 0, 4, 4, 4, 4, 0, 1, 0, 1, 4, 5, 4, 5,
                      0, 0, 2, 2, 4, 4, 6, 6, 0, 1, 2, 3, 4, 5, 6, 7,
                      4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 4, 5, 4, 5, 4, 5,
                      4, 4, 6, 6, 4, 4, 6, 6, 4, 5, 6, 7, 4, 5, 6, 7,
                      0, 1, 0, 1, 4, 5, 4, 5, 1, 1, 1, 1, 5, 5, 5, 5,
                      0, 1, 2, 3, 4, 5, 6, 7, 1, 1, 3, 3, 5, 5, 7, 7,
                      4, 5, 4, 5, 4, 5, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5,
                      4, 5, 6, 7, 4, 5, 6, 7, 5, 5, 7, 7, 5, 5, 7, 7,
                      0, 0, 2, 2, 4, 4, 6, 6, 0, 1, 2, 3, 4, 5, 6, 7,
                      2, 2, 2, 2, 6, 6, 6, 6, 2, 3, 2, 3, 6, 7, 6, 7,
                      4, 4, 6, 6, 4, 4, 6, 6, 4, 5, 6, 7, 4, 5, 6, 7,
                      6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 6, 7, 6, 7, 6, 7,
                      0, 1, 2, 3, 4, 5, 6, 7, 1, 1, 3, 3, 5, 5, 7, 7,
                      2, 3, 2, 3, 6, 7, 6, 7, 3, 3, 3, 3, 7, 7, 7, 7,
                      4, 5, 6, 7, 4, 5, 6, 7, 5, 5, 7, 7, 5, 5, 7, 7,
                      6, 7, 6, 7, 6, 7, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7 };
```

```
/*  localdef.h        eej             7/29/91                                  */
/*     definitions for the MAC environment using Think C  (System 6.0x) */

#ifndef _MC68881_
        #define _MC68881_
#endif

#undef _ERRORCHECK_

#define shareCPU(); {EventRecord ev; if (WaitNextEvent((int) everyEvent, &ev,
(long) 0, (long) 0)) StdEvent(&ev); }

#include <storage.h>
```

```
/*  unanimous.h     rsm     7/28/91     table of unanimous votes  */
/*                              indexed by concatenated tri-bits  */

short utable[512] = {3, 2, 2, 1, 2, 1, 1, 0, 2, 2, 1, 1, 1, 1, 0, 0,
                     2, 1, 2, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0,
                     2, 1, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0,
                     1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                     2, 2, 1, 1, 1, 1, 0, 0, 2, 3, 1, 2, 1, 2, 0, 1,
                     1, 1, 1, 1, 0, 0, 0, 0, 1, 2, 1, 2, 0, 1, 0, 1,
                     1, 1, 0, 0, 1, 1, 0, 0, 1, 2, 0, 1, 1, 2, 0, 1,
                     0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1,
                     2, 1, 2, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0,
                     2, 1, 3, 2, 1, 0, 2, 1, 1, 1, 2, 2, 0, 0, 1, 1,
                     1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                     1, 0, 2, 1, 1, 0, 2, 1, 0, 0, 1, 1, 0, 0, 1, 1,
                     1, 1, 1, 1, 0, 0, 0, 0, 1, 2, 1, 2, 0, 1, 0, 1,
                     1, 1, 2, 2, 0, 0, 1, 1, 1, 2, 2, 3, 0, 1, 1, 2,
                     0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1,
                     0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 2, 0, 1, 1, 2,
                     2, 1, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0,
                     1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                     2, 1, 1, 0, 3, 2, 2, 1, 1, 1, 0, 0, 2, 2, 1, 1,
                     1, 0, 1, 0, 2, 1, 2, 1, 0, 0, 0, 0, 1, 1, 1, 1,
                     1, 1, 0, 0, 1, 1, 0, 0, 1, 2, 0, 1, 1, 2, 0, 1,
                     0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1,
                     1, 1, 0, 0, 2, 2, 1, 1, 1, 2, 0, 1, 2, 3, 1, 2,
                     0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 2, 1, 2,
                     1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                     1, 0, 2, 1, 1, 0, 2, 1, 0, 0, 1, 1, 0, 0, 1, 1,
                     1, 0, 1, 0, 2, 1, 2, 1, 0, 0, 0, 0, 1, 1, 1, 1,
                     1, 0, 2, 1, 2, 1, 3, 2, 0, 0, 1, 1, 1, 1, 2, 2,
                     0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1,
                     0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 2, 0, 1, 1, 2,
                     0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 2, 1, 2,
                     0, 0, 1, 1, 1, 1, 2, 2, 0, 1, 1, 2, 1, 2, 2, 3 };
```

```
/*  errwt.h    eej    7/28/91    error weights for Golay decoder  */

int wt[4096] = {    0,    1,    1,    2,    1,    2,    2,    3,
                    1,    2,    2,    3,    2,    3,    3,    4,
                    1,    2,    2,    3,    2,    3,    3,    4,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    1,    2,    2,    3,    2,    3,    3,    4,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    1,    2,    2,    3,    2,    3,    3,    4,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    4,    3,    3,    2,    3,    4,    4,    3,
                    1,    2,    2,    3,    2,    3,    3,    4,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    3,    4,    4,    3,    4,    3,    3,    2,
                    4,    3,    3,    4,    3,    4,    4,    3,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    4,    3,    3,    4,    3,    4,    2,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    4,    3,    3,    4,    3,    4,    4,    3,
                    4,    3,    3,    4,    3,    2,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    1,    2,    2,    3,    2,    3,    3,    4,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    3,    4,    4,    3,    4,    3,    3,    2,
                    4,    3,    3,    4,    3,    4,    4,    3,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    4,    3,    3,    4,    3,    2,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    4,    3,    3,    4,    3,    4,    2,    3,
                    4,    3,    3,    4,    3,    4,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    2,    3,    3,    4,    3,    4,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    4,    3,    3,    2,    3,    4,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
                    4,    3,    3,    4,    3,    4,    4,    3,
                    4,    3,    3,    4,    3,    4,    4,    3,
                    3,    4,    4,    3,    2,    3,    3,    4,
                    3,    4,    4,    3,    4,    3,    3,    2,
                    4,    3,    3,    4,    3,    4,    4,    3,
                    4,    3,    3,    4,    3,    4,    4,    3,
                    3,    4,    4,    3,    4,    3,    3,    4,
```

```
4,  3,  3,  4,  3,  4,  4,  3,
3,  2,  4,  3,  4,  3,  3,  4,
3,  4,  2,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
1,  2,  2,  3,  2,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  2,
4,  3,  3,  4,  3,  4,  4,  3,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  2,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  2,  3,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  2,  3,  2,  3,  1,  2,
4,  3,  3,  2,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  2,  3,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  2,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  2,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  2,  3,  2,  2,  1,
3,  4,  4,  3,  4,  3,  3,  2,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  2,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  2,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  2,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
```

```
3,  2,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  2,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
1,  2,  2,  3,  2,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  2,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  4,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  2,  3,  4,  4,  3,
3,  4,  4,  3,  2,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  2,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  2,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  2,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  2,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  2,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  2,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  2,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  2,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  2,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  2,  3,  3,  4,
3,  4,  4,  3,  2,  3,  3,  4,
2,  3,  3,  4,  1,  2,  2,  3,
```

```
4,  3,  3,  2,  3,  2,  2,  1,
3,  4,  4,  3,  4,  3,  3,  2,
3,  4,  4,  3,  4,  3,  3,  2,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  2,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  2,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  2,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
4,  3,  3,  2,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
3,  2,  4,  3,  2,  1,  3,  2,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
4,  3,  3,  4,  3,  2,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  2,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  2,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  2,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  2,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  2,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  2,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  2,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
```

```
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  2,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  2,
2,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
1,  2,  2,  3,  2,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  2,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  4,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  2,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  2,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  2,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
3,  4,  4,  3,  2,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  2,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
```

```
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  2,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  2,  3,  4,  3,  3,  4,
3,  4,  4,  3,  2,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  2,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  2,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  2,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  2,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  2,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  2,  3,
3,  2,  2,  1,  4,  3,  3,  2,
4,  3,  3,  2,  3,  4,  4,  3,
4,  3,  3,  2,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  2,
2,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
2,  3,  3,  4,  3,  4,  4,  3,
1,  2,  2,  3,  2,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
3,  2,  4,  3,  2,  1,  3,  2,
```

```
3,  4,  2,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  2,  3,
3,  2,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  2,  3,  2,  3,  1,  2,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  2,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  2,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  2,  4,  3,  4,  3,  3,  4,
3,  2,  4,  3,  4,  3,  3,  4,
2,  1,  3,  2,  3,  2,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  2,  3,
3,  2,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
4,  3,  3,  4,  3,  2,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
3,  2,  2,  1,  4,  3,  3,  2,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  2,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  4,  3,  3,  4,
```

```
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  2,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  2,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  2,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  2,  4,  3,  4,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  2,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  2,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  2,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  2,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  2,  4,  3,
4,  3,  3,  4,  3,  4,  2,  3,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  2,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
2,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  2,
3,  4,  4,  3,  2,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  2,  4,  3,  4,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  4,  2,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  2,  3,  3,  4,
4,  3,  3,  2,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
3,  2,  4,  3,  4,  3,  3,  4,
4,  3,  3,  4,  3,  4,  4,  3,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  2,
4,  3,  3,  4,  3,  4,  4,  3,
3,  4,  4,  3,  4,  3,  3,  4,
```

```
3,    4,    4,    3,    4,    3,    3,    4,
2,    3,    3,    4,    3,    4,    4,    3,
3,    4,    2,    3,    4,    3,    3,    4,
4,    3,    3,    4,    3,    2,    4,    3,
2,    3,    1,    2,    3,    4,    2,    3,
3,    4,    2,    3,    4,    3,    3,    4,
4,    3,    3,    4,    3,    4,    4,    3,
3,    4,    4,    3,    4,    3,    3,    4,
3,    4,    2,    3,    4,    3,    3,    4,
4,    3,    3,    4,    3,    4,    4,    3,
4,    3,    3,    4,    3,    4,    4,    3,
3,    4,    4,    3,    4,    3,    3,    4,
3,    4,    2,    3,    4,    3,    3,    4,
4,    3,    3,    4,    3,    4,    4,    3,
3,    4,    4,    3,    4,    3,    3,    4,
4,    3,    3,    4,    3,    4,    2,    3,
4,    3,    3,    4,    3,    2,    4,    3,
3,    4,    4,    3,    4,    3,    3,    4 };
```

```
/*  errpat.h    eej    7/28/91    error patterns for Golay decoder  */

int  e[4096] = {00000, 00000, 00000, 00000, 00000, 00000, 00000, 00000,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00000,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00000,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 02002,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00000,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00600,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00040,
                00000, 00000, 00000, 00011, 00000, 04004, 01120, 00011,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00000,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00060,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 04300,
                00000, 00000, 00000, 00011, 00000, 01400, 00004, 00004,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 03004,
                00000, 00000, 00000, 00011, 00000, 00100, 04002, 00011,
                00000, 00000, 00000, 00011, 00000, 00022, 00400, 00011,
                00000, 00011, 00011, 00011, 02240, 00011, 00004, 00011,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00000,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00600,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 01010,
                00000, 00000, 00000, 04020, 00000, 00141, 00004, 00004,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00600,
                00000, 00000, 00000, 00600, 00000, 00600, 00600, 00600,
                00000, 00000, 00000, 00104, 00000, 00022, 06001, 00022,
                00000, 03000, 00042, 00011, 00010, 00010, 00004, 00600,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00001,
                00000, 00000, 00000, 01102, 00000, 06010, 00004, 00001,
                00000, 00000, 00000, 02440, 00000, 00022, 00004, 00001,
                00000, 00200, 00004, 00004, 00004, 00004, 00004, 00004,
                00000, 00000, 00000, 04000, 00000, 00022, 00150, 00001,
                00000, 00044, 02020, 00011, 01001, 00022, 00004, 00600,
                00000, 00022, 01200, 00011, 00022, 00022, 00004, 00022,
                04500, 00011, 00004, 00011, 00004, 00022, 00004, 00004,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00000,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00114,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00040,
                00000, 00000, 00000, 04020, 00000, 01400, 00201, 00040,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00040,
                00000, 00000, 00000, 01000, 00000, 02021, 04002, 00040,
                00000, 00000, 00000, 00040, 00000, 00040, 00040, 00040,
                00000, 00302, 02404, 00011, 00010, 00010, 00010, 00040,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00001,
                00000, 00000, 00000, 02200, 00000, 01400, 04002, 00001,
                00000, 00000, 00000, 00006, 00000, 01400, 02030, 00001,
                00000, 01400, 00140, 00006, 01400, 01400, 00004, 01400,
                00000, 00000, 00000, 00520, 00000, 00210, 04002, 00001,
                00000, 00044, 04002, 00011, 04002, 00044, 04002, 04002,
                00000, 06000, 01200, 00006, 00105, 00022, 00040, 00040,
                00020, 00011, 00011, 00011, 00010, 01400, 04002, 00011,
                00000, 00000, 00000, 00000, 00000, 00000, 00000, 00001,
                00000, 00000, 00000, 04020, 00000, 00002, 03040, 00001,
                00000, 00000, 00000, 04020, 00000, 02204, 00502, 00001,
                00000, 04020, 04020, 04020, 00010, 00002, 00004, 04020,
                00000, 00000, 00000, 02012, 00000, 05100, 00024, 00001,
                00000, 00044, 00101, 00044, 00010, 00002, 00010, 00600,
                00000, 00401, 01200, 00040, 00010, 00010, 00010, 00040,
                00010, 00010, 00010, 04020, 00010, 00010, 00010, 00010,
                00000, 00000, 00000, 00001, 00000, 00001, 00001, 00001,
                00000, 00044, 00410, 00001, 00320, 00001, 00001, 00001,
                00000, 00110, 01200, 00001, 04040, 00001, 00001, 00001,
                02003, 00044, 00004, 04020, 00004, 01400, 00004, 00001,
```

```
00000, 00044, 01200, 00001, 02400, 00001, 00001, 00001,
00044, 00044, 00044, 00044, 00010, 00044, 04002, 00001,
01200, 00022, 01200, 01200, 00010, 00022, 01200, 00001,
00010, 00044, 01200, 00011, 00010, 00010, 00004, 02100,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00000,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 05001,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00040,
00000, 00000, 00000, 00500, 00000, 00230, 00004, 00004,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00040,
00000, 00000, 00000, 00026, 00000, 00100, 02010, 00026,
00000, 00000, 00000, 00040, 00000, 00040, 00040, 00040,
00000, 03000, 04200, 00011, 00403, 00040, 00004, 00040,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00412,
00000, 00000, 00000, 02200, 00000, 00100, 00004, 00004,
00000, 00000, 00000, 01020, 00000, 02001, 00004, 00004,
00000, 04042, 00004, 00004, 00004, 00004, 00004, 00004,
00000, 00000, 00000, 04000, 00000, 00100, 00221, 00040,
00000, 00100, 01440, 00011, 00100, 00100, 00004, 00100,
00000, 00604, 02102, 00011, 05010, 00022, 00004, 00040,
00020, 00011, 00004, 00011, 00004, 00100, 00004, 00004,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 02120,
00000, 00000, 00000, 00050, 00000, 00002, 00004, 00002,
00000, 00000, 00000, 00203, 00000, 04400, 00004, 00004,
00000, 03000, 00004, 00004, 00004, 00002, 00004, 00004,
00000, 00000, 00000, 04000, 00000, 00015, 01002, 00015,
00000, 03000, 00101, 00026, 04060, 00002, 00004, 00600,
00000, 03000, 00430, 00040, 00300, 00015, 00004, 00040,
03000, 03000, 00004, 03000, 00004, 03000, 00004, 00004,
00000, 00000, 00000, 04000, 00000, 01240, 00004, 00001,
00000, 00421, 00004, 00004, 00004, 00002, 00004, 00004,
00000, 00110, 00004, 00004, 00004, 00004, 00004, 00004,
00004, 00004, 00004, 00004, 00004, 00004, 00004, 00004,
00000, 04000, 04000, 04000, 02400, 00015, 00004, 04000,
00212, 00044, 00004, 04000, 00004, 00100, 00004, 00004,
00041, 00022, 00004, 04000, 00004, 00022, 00004, 00004,
00004, 03000, 00004, 00004, 00004, 00004, 00004, 00004,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00040,
00000, 00000, 00000, 02200, 00000, 00002, 00420, 00002,
00000, 00000, 00000, 00040, 00000, 00040, 00040, 00040,
00000, 00005, 01012, 00005, 06100, 00002, 00004, 00040,
00000, 00000, 00000, 00040, 00000, 00040, 00040, 00040,
00000, 04410, 00101, 00026, 01204, 00002, 00040, 00040,
00000, 00040, 00040, 00040, 00040, 00040, 00040, 00040,
00020, 00005, 00020, 00040, 00010, 00040, 00040, 00040,
00000, 00000, 00000, 02200, 00000, 04024, 01100, 00001,
00000, 02200, 02200, 02200, 00051, 00002, 00004, 02200,
00000, 00110, 04401, 00006, 00202, 00040, 00004, 00040,
00020, 00005, 00004, 02200, 00004, 01400, 00004, 00004,
00000, 01003, 00014, 00014, 02400, 00040, 00014, 00040,
00020, 00020, 00014, 02200, 00020, 00100, 04002, 00040,
00020, 00020, 00014, 00040, 00020, 00040, 00040, 00040,
00020, 00020, 00020, 00011, 00020, 00020, 00004, 00040,
00000, 00000, 00000, 01404, 00000, 00002, 04210, 00001,
00000, 00002, 00101, 00002, 00002, 00002, 00002, 00002,
00000, 00110, 02000, 00040, 01021, 00002, 00004, 00040,
00640, 00002, 00004, 04020, 00002, 00002, 00004, 00002,
00000, 00220, 00101, 00040, 02400, 00002, 00024, 00040,
00101, 00002, 00101, 00101, 00002, 00002, 00101, 00002,
04006, 00040, 00040, 00040, 00010, 00040, 00040, 00040,
00010, 03000, 00101, 00040, 00010, 00002, 00004, 00040,
00000, 00110, 00062, 00001, 02400, 00001, 00001, 00001,
05000, 00002, 00004, 02200, 00002, 00002, 00004, 00001,
```

```
00110, 00110, 00004, 00110, 00004, 00110, 00004, 00001,
00004, 00110, 00004, 00004, 00004, 00002, 00004, 00004,
02400, 00044, 00014, 04000, 02400, 02400, 02400, 00001,
00020, 00044, 00101, 00044, 02400, 00002, 00004, 01030,
00020, 00110, 01200, 00040, 02400, 00022, 00004, 00040,
00020, 00020, 00004, 00402, 00004, 04201, 00004, 00004,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00000,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 02002,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 02002,
00000, 00000, 00000, 02002, 00000, 02002, 02002, 02002,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 04030,
00000, 00000, 00000, 01000, 00000, 00100, 00045, 00045,
00000, 00000, 00000, 00104, 00000, 01201, 00400, 00040,
00000, 00460, 04200, 00011, 00010, 00010, 00010, 02002,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00001,
00000, 00000, 00000, 04404, 00000, 00100, 01210, 00001,
00000, 00000, 00000, 01020, 00000, 00054, 00400, 00001,
00000, 00200, 00140, 00011, 04021, 00054, 00004, 02002,
00000, 00000, 00000, 00242, 00000, 00100, 00400, 00001,
00000, 00100, 02020, 00011, 00100, 00100, 00045, 00100,
00000, 06000, 00400, 00011, 00400, 00022, 00400, 00400,
01006, 00011, 00011, 00011, 00010, 00100, 00400, 00011,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00001,
00000, 00000, 00000, 00050, 00000, 01024, 04100, 00001,
00000, 00000, 00000, 00104, 00000, 04400, 00260, 00001,
00000, 00200, 01401, 00050, 00010, 00010, 00004, 02002,
00000, 00000, 00000, 00104, 00000, 02040, 01002, 00001,
00000, 04003, 02020, 00050, 00010, 00010, 00010, 00600,
00000, 00104, 00104, 00104, 00010, 00010, 00010, 00104,
00010, 00010, 00010, 00104, 00010, 00010, 00010, 00010,
00000, 00000, 00000, 00001, 00000, 00001, 00001, 00001,
00000, 00200, 02020, 00001, 00442, 00001, 00001, 00001,
00000, 00200, 04012, 00001, 03100, 00001, 00001, 00001,
00200, 00200, 00004, 00200, 00004, 00200, 00004, 00001,
00000, 01410, 02020, 00001, 04204, 00001, 00001, 00001,
02020, 00044, 02020, 02020, 00010, 00100, 02020, 00001,
00041, 00022, 00041, 00104, 00010, 00022, 00400, 00001,
00010, 00200, 02020, 00011, 00010, 00010, 00004, 05040,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00001,
00000, 00000, 00000, 01000, 00000, 04240, 00420, 00001,
00000, 00000, 00000, 00610, 00000, 00120, 05004, 00001,
00000, 00005, 00140, 00005, 00010, 00005, 00010, 02002,
00000, 00000, 00000, 01000, 00000, 00406, 02300, 00001,
00000, 01000, 01000, 01000, 00010, 00010, 00010, 01000,
00000, 06000, 00023, 00023, 00010, 00010, 00010, 00040,
00010, 00005, 00010, 01000, 00010, 00010, 00010, 00010,
00000, 00000, 00000, 00001, 00000, 00001, 00001, 00001,
00000, 00032, 00140, 00001, 02004, 00001, 00001, 00001,
00000, 06000, 00140, 00001, 00202, 00001, 00001, 00001,
00140, 00005, 00140, 00140, 00010, 01400, 00140, 00001,
00000, 06000, 00014, 00001, 01060, 00001, 00001, 00001,
00601, 00032, 00014, 01000, 00010, 00100, 04002, 00001,
06000, 06000, 00014, 06000, 00010, 06000, 00400, 00001,
00010, 06000, 00140, 00011, 00010, 00010, 00010, 00224,
00000, 00000, 00000, 00001, 00000, 00001, 00001, 00001,
00000, 02500, 00206, 00001, 00010, 00001, 00001, 00001,
00000, 01042, 02000, 00001, 00010, 00001, 00001, 00001,
00010, 00005, 00010, 04020, 00010, 00010, 00010, 00001,
00000, 00220, 04440, 00001, 00010, 00001, 00001, 00001,
00010, 00010, 00010, 01000, 00010, 00010, 00010, 00001,
00010, 00010, 00010, 00104, 00010, 00010, 00010, 00001,
00010, 00010, 00010, 00010, 00010, 00010, 00010, 00010,
```

```
00000, 00001, 00001, 00001, 00001, 00001, 00001, 00001,
05000, 00001, 00001, 00001, 00001, 00001, 00001, 00001,
00424, 00001, 00001, 00001, 00001, 00001, 00001, 00001,
00010, 00200, 00140, 00001, 00010, 00001, 00001, 00001,
00102, 00001, 00001, 00001, 00001, 00001, 00001, 00001,
00010, 00044, 02020, 00001, 00010, 00001, 00001, 00001,
00010, 06000, 01200, 00001, 00010, 00001, 00001, 00001,
00010, 00010, 00010, 00402, 00010, 00010, 00010, 00001,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00204,
00000, 00000, 00000, 00050, 00000, 00100, 00420, 00050,
00000, 00000, 00000, 01020, 00000, 04400, 00111, 00040,
00000, 00005, 04200, 00005, 01040, 00005, 00004, 02002,
00000, 00000, 00000, 02401, 00000, 00100, 01002, 00040,
00000, 00100, 04200, 00026, 00100, 00100, 00045, 00100,
00000, 00012, 04200, 00012, 02024, 00012, 00040, 00040,
04200, 00005, 04200, 04200, 00010, 00100, 04200, 00021,
00000, 00000, 00000, 01020, 00000, 00100, 06040, 00001,
00000, 00100, 00003, 00003, 00100, 00100, 00003, 00100,
00000, 01020, 01020, 01020, 00202, 00054, 00004, 01020,
02410, 00005, 00003, 01020, 00004, 00100, 00004, 00004,
00000, 00100, 00014, 00014, 00100, 00100, 00014, 00100,
00100, 00100, 00003, 00100, 00100, 00100, 00100, 00100,
00041, 00012, 00014, 01020, 00041, 00100, 00400, 00040,
00020, 00100, 04200, 00011, 00100, 00100, 00004, 00100,
00000, 00000, 00000, 00050, 00000, 04400, 01002, 00001,
00000, 00050, 00050, 00050, 02201, 00002, 00004, 00050,
00000, 04400, 02000, 00050, 04400, 04400, 00004, 04400,
00122, 00005, 00004, 00050, 00004, 04400, 00004, 00004,
00000, 00220, 01002, 00050, 01002, 00015, 01002, 01002,
00404, 00050, 00050, 00050, 00010, 00100, 01002, 00021,
00041, 00012, 00041, 00104, 00010, 04400, 01002, 00021,
00010, 03000, 04200, 00021, 00010, 00010, 00004, 00021,
00000, 02006, 00700, 00001, 00030, 00001, 00001, 00001,
05000, 00050, 00003, 00050, 00004, 00100, 00004, 00001,
00041, 00041, 00004, 01020, 00004, 04400, 00004, 00001,
00004, 00200, 00004, 00004, 00004, 00004, 00004, 00004,
00041, 00041, 00014, 04000, 00030, 00100, 01002, 00001,
00041, 00100, 02020, 00050, 00100, 00100, 00004, 00100,
00041, 00041, 00041, 00041, 00041, 00022, 00004, 02210,
00041, 00041, 00004, 00402, 00004, 00100, 00004, 00004,
00000, 00000, 00000, 04102, 00000, 03010, 00420, 00001,
00000, 00005, 00420, 00005, 00420, 00002, 00420, 00420,
00000, 00005, 02000, 00005, 00202, 00005, 00040, 00040,
00005, 00005, 00005, 00005, 00005, 00005, 00420, 00005,
00000, 00220, 00014, 00014, 04001, 00040, 00014, 00040,
02042, 00005, 00014, 01000, 00010, 00100, 00420, 00040,
01500, 00005, 00014, 00040, 00010, 00040, 00040, 00040,
00005, 00005, 04200, 00005, 00010, 00005, 00010, 00040,
00000, 00440, 00014, 00001, 00202, 00001, 00001, 00001,
05000, 00005, 00003, 02200, 00051, 00100, 00420, 00001,
00202, 00005, 00014, 01020, 00202, 00202, 00202, 00001,
00005, 00005, 00140, 00005, 00202, 00005, 00004, 04010,
00014, 00014, 00014, 00014, 00014, 00100, 00014, 00001,
00014, 00100, 00014, 00014, 00100, 00100, 00014, 00100,
00014, 06000, 00014, 00014, 00202, 00040, 00014, 00040,
00020, 00005, 00014, 00402, 00010, 00100, 03001, 00040,
00000, 00220, 02000, 00001, 00144, 00001, 00001, 00001,
05000, 00002, 00050, 00050, 00002, 00002, 00420, 00001,
02000, 00005, 02000, 02000, 00010, 04400, 02000, 00001,
00005, 00005, 02000, 00005, 00010, 00002, 00004, 01300,
00220, 00220, 00014, 00220, 00010, 00220, 01002, 00001,
00010, 00220, 00101, 00050, 00010, 00002, 00010, 06004,
```

```
00010, 00220, 02000, 00040, 00010, 00010, 00010, 00040,
00010, 00005, 00010, 00402, 00010, 00010, 00010, 00010,
05000, 00001, 00001, 00001, 00001, 00001, 00001, 00001,
05000, 05000, 05000, 00001, 05000, 00001, 00001, 00001,
00041, 00110, 02000, 00001, 00202, 00001, 00001, 00001,
05000, 00005, 00004, 00402, 00004, 02060, 00004, 00001,
00014, 00220, 00014, 00001, 02400, 00001, 00001, 00001,
05000, 00044, 00014, 00402, 00010, 00100, 00240, 00001,
00041, 00041, 00014, 00402, 00010, 01004, 04120, 00001,
00010, 00402, 00402, 00402, 00010, 00010, 00004, 00402,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00000,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00060,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 01010,
00000, 00000, 00000, 00500, 00000, 04004, 00201, 00060,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00103,
00000, 00000, 00000, 01000, 00000, 04004, 02010, 00060,
00000, 00000, 00000, 02220, 00000, 04004, 00400, 00040,
00000, 04004, 00042, 00011, 04004, 04004, 00042, 04004,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00060,
00000, 00000, 00000, 00060, 00000, 00060, 00060, 00060,
00000, 00000, 00000, 00006, 00000, 02001, 00400, 00006,
00000, 00200, 07000, 00006, 00112, 00060, 00004, 00060,
00000, 00000, 00000, 04000, 00000, 00210, 00400, 00060,
00000, 02402, 00304, 00011, 01001, 00060, 00060, 00060,
00000, 01140, 00400, 00006, 00400, 00022, 00400, 00400,
00020, 00011, 00011, 00011, 00020, 04004, 00400, 00011,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 01010,
00000, 00000, 00000, 02005, 00000, 00002, 04100, 00002,
00000, 00000, 00000, 01010, 00000, 01010, 01010, 01010,
00000, 00200, 00042, 00042, 02420, 00002, 00004, 01010,
00000, 00000, 00000, 04000, 00000, 02040, 00024, 00024,
00000, 00130, 00042, 00042, 01001, 00002, 00024, 00600,
00000, 00401, 00042, 00042, 00300, 00022, 00024, 01010,
00042, 00042, 00042, 00042, 00010, 04004, 00042, 00021,
00000, 00000, 00000, 04000, 00000, 00504, 02202, 00001,
00000, 00200, 00410, 00060, 01001, 00002, 00004, 00060,
00000, 00200, 00121, 00006, 04040, 00022, 00004, 01010,
00200, 00200, 00004, 00200, 00004, 00200, 00004, 00004,
00000, 04000, 04000, 04000, 01001, 00022, 00024, 04000,
01001, 00044, 00042, 04000, 01001, 01001, 01001, 00016,
02014, 00022, 00042, 04000, 00022, 00022, 00400, 00022,
00020, 00200, 00042, 00011, 01001, 00022, 00004, 02100,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 06400,
00000, 00000, 00000, 01000, 00000, 00002, 00201, 00002,
00000, 00000, 00000, 00006, 00000, 00120, 00201, 00006,
00000, 02050, 00201, 00006, 00201, 00002, 00201, 00201,
00000, 00000, 00000, 01000, 00000, 00210, 00024, 00024,
00000, 01000, 01000, 01000, 00540, 00002, 00024, 01000,
00000, 00401, 04110, 00006, 03002, 00040, 00024, 00040,
00020, 00020, 00020, 01000, 00010, 04004, 00201, 00040,
00000, 00000, 00000, 00006, 00000, 00210, 01100, 00001,
00000, 04101, 00410, 00006, 02004, 00002, 00060, 00060,
00000, 00006, 00006, 00006, 04040, 00006, 00006, 00006,
00020, 00006, 00006, 00006, 00020, 01400, 00201, 00006,
00000, 00210, 02041, 00006, 00210, 00210, 00024, 00210,
00020, 00020, 00020, 01000, 00020, 00210, 04002, 00060,
00020, 00006, 00006, 00006, 00020, 00210, 00400, 00006,
00020, 00020, 00020, 00006, 00020, 00020, 00020, 02100,
00000, 00000, 00000, 00340, 00000, 00002, 00024, 00001,
00000, 00002, 00410, 00002, 00002, 00002, 00002, 00002,
00000, 00401, 02000, 00006, 04040, 00002, 00024, 01010,
01104, 00002, 00042, 04020, 00002, 00002, 00201, 00002,
```

```
00000, 00401, 00024, 00024, 00024, 00002, 00024, 00024,
06200, 00002, 00024, 01000, 00002, 00002, 00024, 00002,
00401, 00401, 00024, 00401, 00010, 00401, 00024, 00024,
00010, 00401, 00042, 00042, 00010, 00002, 00010, 02100,
00000, 03020, 00410, 00001, 04040, 00001, 00001, 00001,
00410, 00002, 00410, 00410, 00002, 00002, 00410, 00001,
04040, 00006, 00006, 00006, 04040, 04040, 04040, 00001,
00020, 00200, 00410, 00006, 04040, 00002, 00004, 02100,
00102, 00044, 00024, 04000, 00024, 00210, 00024, 00001,
00020, 00044, 00410, 00044, 01001, 00002, 00024, 02100,
00020, 00401, 01200, 00006, 04040, 00022, 00013, 02100,
00020, 00020, 00020, 02100, 00010, 02100, 02100, 02100,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00204,
00000, 00000, 00000, 00500, 00000, 00002, 02010, 00002,
00000, 00000, 00000, 00500, 00000, 02001, 04022, 00040,
00000, 00500, 00500, 00500, 01040, 00002, 00004, 00500,
00000, 00000, 00000, 04000, 00000, 01420, 02010, 00040,
00000, 00241, 02010, 00026, 02010, 00002, 02010, 02010,
00000, 00012, 01005, 00012, 00300, 00012, 00040, 00040,
00020, 00012, 00020, 00500, 00020, 04004, 02010, 00021,
00000, 00000, 00000, 04000, 00000, 02001, 01100, 00060,
00000, 01014, 00003, 00003, 04600, 00002, 00003, 00060,
00000, 02001, 00250, 00006, 02001, 02001, 00004, 02001,
00020, 00020, 00003, 00500, 00004, 02001, 00004, 00004,
00000, 04000, 04000, 04000, 00046, 00046, 00046, 04000,
00020, 00020, 00003, 04000, 00020, 00100, 02010, 00060,
00020, 00012, 00020, 04000, 00020, 02001, 00400, 00040,
00020, 00020, 00020, 00011, 00020, 00020, 00004, 01202,
00000, 00000, 00000, 04000, 00000, 00002, 00441, 00002,
00000, 00002, 01220, 00002, 00002, 00002, 00002, 00002,
00000, 00064, 02000, 00064, 00300, 00002, 00004, 01010,
04011, 00002, 00004, 00500, 00002, 00002, 00004, 00002,
00000, 04000, 04000, 04000, 00300, 00002, 00024, 04000,
00404, 00002, 00042, 04000, 00002, 00002, 02010, 00002,
00300, 00012, 00042, 04000, 00300, 00300, 00300, 00021,
00020, 03000, 00042, 00021, 00300, 00002, 00004, 00021,
00000, 04000, 04000, 04000, 00030, 00002, 00004, 04000,
02140, 00002, 00003, 04000, 00002, 00002, 00004, 00002,
01402, 00064, 00004, 04000, 00004, 02001, 00004, 00004,
00004, 00200, 00004, 00004, 00004, 00002, 00004, 00004,
04000, 04000, 04000, 04000, 00030, 04000, 04000, 04000,
00020, 04000, 04000, 04000, 01001, 00002, 00004, 04000,
00020, 04000, 04000, 04000, 00300, 00022, 00004, 04000,
00020, 00020, 00004, 04000, 00004, 00450, 00004, 00004,
00000, 00000, 00000, 00031, 00000, 00002, 01100, 00002,
00000, 00002, 04044, 00002, 00002, 00002, 00002, 00002,
00000, 05200, 02000, 00006, 00414, 00002, 00040, 00040,
00020, 00002, 00020, 00500, 00002, 00002, 00201, 00002,
00000, 02104, 00602, 00031, 04001, 00002, 00024, 00040,
00020, 00002, 00020, 01000, 00002, 00002, 02010, 00002,
00020, 00012, 00020, 00040, 00020, 00040, 00040, 00040,
00020, 00020, 00020, 00020, 00020, 00002, 00020, 00040,
00000, 00440, 01100, 00006, 01100, 00002, 01100, 01100,
00020, 00002, 00003, 02200, 00002, 00002, 01100, 00002,
00020, 00006, 00006, 00006, 00020, 02001, 01100, 00006,
00020, 00020, 00020, 00006, 00020, 00002, 00004, 04010,
00020, 00020, 00014, 04000, 00020, 00210, 01100, 00040,
00020, 00020, 00020, 00020, 00020, 00002, 00020, 00405,
00020, 00020, 00020, 00006, 00020, 00020, 00013, 00040,
00020, 00020, 00020, 00020, 00020, 00020, 00020, 00020,
00000, 00002, 02000, 00002, 00002, 00002, 00002, 00002,
00002, 00002, 00002, 00002, 00002, 00002, 00002, 00002,
```

```
02000, 00002, 02000, 02000, 00002, 00002, 02000, 00002,
00002, 00002, 02000, 00002, 00002, 00002, 00002, 00002,
01050, 00002, 00024, 04000, 00002, 00002, 00024, 00002,
00002, 00002, 00101, 00002, 00002, 00002, 00002, 00002,
00020, 00401, 02000, 00040, 00300, 00002, 00013, 00040,
00020, 00002, 00020, 00214, 00002, 00002, 05400, 00002,
00205, 00002, 00062, 04000, 00002, 00002, 01100, 00001,
00002, 00002, 00410, 00002, 00002, 00002, 00002, 00002,
00020, 00110, 02000, 00006, 04040, 00002, 00004, 00620,
00020, 00002, 00004, 01041, 00002, 00002, 00004, 00002,
00020, 04000, 04000, 04000, 02400, 00002, 00013, 04000,
00020, 00002, 00020, 04000, 00002, 00002, 00240, 00002,
00020, 00020, 00013, 04000, 00013, 01004, 00013, 00013,
00020, 00020, 00020, 00020, 00020, 00002, 00004, 02100,
00000, 00000, 00000, 00000, 00000, 00000, 00000, 00204,
00000, 00000, 00000, 01000, 00000, 00411, 04100, 00060,
00000, 00000, 00000, 04041, 00000, 00120, 00400, 00120,
00000, 00200, 00034, 00034, 01040, 00120, 00034, 02002,
00000, 00000, 00000, 01000, 00000, 02040, 00400, 00103,
00000, 01000, 01000, 01000, 00222, 00100, 00045, 01000,
00000, 00012, 00400, 00012, 00400, 00012, 00400, 00400,
02101, 00012, 00034, 01000, 00010, 04004, 00400, 00021,
00000, 00000, 00000, 02110, 00000, 05002, 00400, 00001,
00000, 00200, 00003, 00003, 02004, 00060, 00003, 00060,
00000, 00200, 00400, 00006, 00400, 00054, 00400, 00400,
00200, 00200, 00003, 00200, 00112, 00200, 00400, 00060,
00000, 00025, 00400, 00025, 00400, 00025, 00400, 00400,
04050, 00025, 00003, 01000, 00100, 00100, 00400, 00016,
00400, 00012, 00400, 00400, 00400, 00400, 00400, 00400,
00020, 00200, 00400, 00011, 00400, 00043, 00400, 00400,
00000, 00000, 00000, 00422, 00000, 02040, 04100, 00001,
00000, 00200, 04100, 00050, 04100, 00002, 04100, 04100,
00000, 00200, 02000, 00104, 00007, 00007, 00007, 01010,
00200, 00200, 00034, 00200, 00007, 00200, 04100, 00021,
00000, 02040, 00211, 00104, 02040, 02040, 00024, 02040,
00404, 00130, 00042, 01000, 00010, 02040, 04100, 00016,
05020, 00012, 00042, 00104, 00007, 02040, 00400, 00021,
00010, 00200, 00042, 00021, 00010, 00010, 00010, 00021,
00000, 00200, 01044, 00001, 00030, 00001, 00001, 00001,
00200, 00200, 00003, 00200, 00030, 00200, 04100, 00001,
00200, 00200, 00121, 00200, 00007, 00200, 00400, 00001,
00200, 00200, 00200, 00200, 00200, 00200, 00004, 00200,
00102, 00025, 00102, 04000, 00030, 02040, 00400, 00001,
00102, 00200, 02020, 00016, 01001, 00016, 00016, 00016,
00041, 00200, 00400, 00070, 00400, 00022, 00400, 00400,
00200, 00200, 00042, 00200, 00010, 00200, 00400, 00016,
00000, 00000, 00000, 01000, 00000, 00120, 00052, 00001,
00000, 01000, 01000, 01000, 02004, 00002, 00052, 01000,
00000, 00120, 02000, 00006, 00120, 00120, 00052, 00120,
04402, 00005, 00034, 01000, 00010, 00120, 00201, 00120,
00000, 01000, 01000, 01000, 04001, 00120, 00024, 01000,
01000, 01000, 01000, 01000, 00010, 01000, 01000, 01000,
00244, 00012, 00023, 01000, 00010, 00120, 00400, 00040,
00010, 01000, 01000, 01000, 00010, 00010, 00010, 01000,
00000, 00440, 04220, 00001, 02004, 00001, 00001, 00001,
02004, 00032, 00003, 01000, 02004, 02004, 02004, 00001,
01011, 00006, 00006, 00006, 00120, 00120, 00400, 00001,
00020, 00200, 00140, 00006, 02004, 00043, 00140, 04010,
00102, 00025, 00014, 01000, 00102, 00210, 00400, 00001,
00020, 01000, 01000, 01000, 02004, 00043, 00131, 01000,
00020, 06000, 00400, 00006, 00400, 00043, 00400, 00400,
00020, 00020, 00020, 01000, 00010, 00043, 00400, 00043,
```

```
00000, 04014, 02000, 00001, 01600, 00001, 00001, 00001,
00061, 00002, 00061, 01000, 00002, 00002, 04100, 00001,
02000, 00120, 02000, 02000, 00007, 00120, 02000, 00001,
00010, 00200, 02000, 00113, 00010, 00002, 00010, 00444,
00102, 00102, 00024, 01000, 00010, 02040, 00024, 00001,
00010, 01000, 01000, 01000, 00010, 00002, 00010, 01000,
00010, 00401, 02000, 00070, 00010, 00010, 00010, 04202,
00010, 00010, 00010, 01000, 00010, 00010, 00010, 00010,
00102, 00001, 00001, 00001, 00001, 00001, 00001, 00001,
00061, 00200, 00410, 00001, 02004, 00001, 00001, 00001,
00102, 00200, 02000, 00001, 04040, 00001, 00001, 00001,
00200, 00200, 00140, 00200, 00010, 00200, 01022, 00001,
00102, 00102, 00102, 00001, 00102, 00001, 00001, 00001,
00102, 00044, 00102, 01000, 00010, 04420, 00240, 00001,
00102, 00070, 00070, 00070, 00010, 01004, 00400, 00001,
00010, 00200, 04005, 00070, 00010, 00010, 00010, 02100,
00000, 00000, 00000, 00204, 00000, 00204, 00204, 00204,
00000, 06020, 00003, 00003, 01040, 00002, 00003, 00204,
00000, 00012, 02000, 00012, 01040, 00012, 00111, 00204,
01040, 00005, 00003, 00500, 01040, 01040, 01040, 00021,
00000, 00012, 00160, 00012, 04001, 00012, 00160, 00204,
00404, 00012, 00003, 01000, 00100, 00100, 02010, 00021,
00012, 00012, 00012, 00012, 00012, 00012, 00400, 00012,
00012, 00012, 04200, 00012, 01040, 00012, 00021, 00021,
00000, 00440, 00003, 00003, 00030, 00030, 00003, 00204,
00003, 00003, 00003, 00003, 00003, 00100, 00003, 00003,
04104, 00012, 00003, 01020, 00030, 02001, 00400, 00142,
00003, 00200, 00003, 00003, 01040, 00100, 00003, 04010,
03200, 00012, 00003, 04000, 00030, 00100, 00400, 00100,
00003, 00100, 00003, 00003, 00100, 00100, 00003, 00100,
00012, 00012, 00400, 00012, 00400, 00012, 00400, 00400,
00020, 00012, 00003, 02044, 00020, 00100, 00400, 00021,
00000, 01101, 02000, 00050, 00030, 00002, 00030, 00204,
00404, 00002, 00003, 00050, 00002, 00002, 04100, 00002,
02000, 00012, 02000, 02000, 00007, 04400, 02000, 00021,
00122, 00200, 02000, 00021, 01040, 00002, 00004, 00021,
00404, 00012, 00160, 04000, 00030, 02040, 01002, 00021,
00404, 00404, 00404, 00021, 00404, 00002, 00021, 00021,
00012, 00012, 02000, 00012, 00300, 00012, 00021, 00021,
00404, 00012, 00021, 00021, 00010, 00021, 00021, 00021,
00030, 00030, 00003, 04000, 00030, 00030, 00030, 00001,
00003, 00200, 00003, 00003, 00030, 00002, 00003, 03400,
00030, 00200, 02000, 00142, 00030, 00030, 00004, 00142,
00200, 00200, 00003, 00200, 00004, 00200, 00004, 00004,
00030, 04000, 04000, 04000, 00030, 00030, 00030, 04000,
00404, 00100, 00003, 04000, 00030, 00100, 00240, 00016,
00041, 00012, 00041, 04000, 00030, 01004, 00400, 00021,
00020, 00200, 01110, 00021, 06002, 00021, 00004, 00021,
00000, 00440, 02000, 00031, 04001, 00002, 00052, 00204,
00310, 00002, 00003, 01000, 00002, 00002, 00420, 00002,
02000, 00005, 02000, 02000, 00120, 00120, 02000, 00040,
00005, 00005, 02000, 00005, 01040, 00002, 00106, 04010,
04001, 00012, 00014, 01000, 04001, 04001, 04001, 00040,
00020, 01000, 01000, 01000, 04001, 00002, 00106, 01000,
00012, 00012, 02000, 00012, 04001, 00012, 00040, 00040,
00020, 00005, 00020, 01000, 00010, 02600, 00106, 00021,
00440, 00440, 00003, 00440, 00030, 00440, 01100, 00001,
00003, 00440, 00003, 00003, 02004, 00002, 00003, 04010,
00020, 00440, 02000, 00006, 00202, 00120, 00065, 04010,
00020, 00005, 00003, 04010, 00020, 04010, 04010, 04010,
00014, 00440, 00014, 00014, 04001, 00100, 00014, 02022,
00020, 00020, 00003, 01000, 00020, 00100, 00240, 00100,
```

```
00020, 00012, 00014, 00301, 00020, 01004, 00400, 00040,
00020, 00020, 00020, 00020, 00020, 00020, 00020, 04010,
02000, 00002, 02000, 02000, 00002, 00002, 02000, 00001,
00002, 00002, 02000, 00002, 00002, 00002, 00002, 00002,
02000, 02000, 02000, 02000, 02000, 00002, 02000, 02000,
02000, 00002, 02000, 02000, 00002, 00002, 02000, 00002,
00102, 00220, 02000, 00047, 04001, 00002, 00024, 00510,
00404, 00002, 00101, 01000, 00002, 00002, 00240, 00002,
02000, 00012, 02000, 02000, 00010, 01004, 02000, 00021,
00010, 04140, 02000, 00021, 00010, 00002, 00010, 00021,
00030, 00440, 02000, 00001, 00030, 00001, 00001, 00001,
05000, 00002, 00003, 00124, 00002, 00002, 00240, 00001,
02000, 00110, 02000, 02000, 00030, 01004, 02000, 00001,
00020, 00200, 02000, 00124, 00501, 00002, 00004, 04010,
00102, 00102, 00014, 04000, 00030, 01004, 00240, 00001,
00020, 02011, 00240, 00124, 00240, 00002, 00240, 00240,
00020, 01004, 02000, 00070, 01004, 01004, 00013, 01004,
00020, 00020, 00020, 00402, 00010, 01004, 00240, 00021 };
```