V.32 AND V.33 MODEM MODEL AND ANALYSIS

by

Joseph John Pisula

Project Report Submitted to the Faculty of the

Virginia Polytechnic Institute and State University

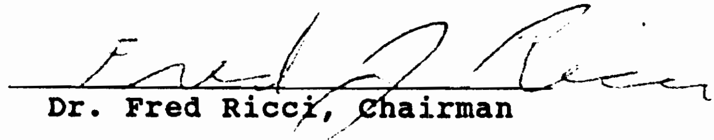
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in

Electrical Engineering

APPROVED:



Dr. Fred Ricci, Chairman

Dr. Daniel Schaefer

Dr. William Lawrence

April, 1992

Blacksburg, Virginia

c.2

LD
5655
V851
1992
P578
C.2

V.32 AND V.33 MODEM MODEL AND ANALYSIS

by
Joseph John Pisula
Committee Chairman: Dr. Fred Ricci
Electrical Engineering

(ABSTRACT)

Over the past ten years, modem technology has evolved to reliably handle 9.6 kbps on general switched telephone networks (GSTN's) and 14.4 kbps on special quality leased circuits largely due to the maturation of Trellis Code Modulation (TCM). A TCM modem can tolerate more than twice the noise power that a Quadrature Amplitude Modulation (QAM) modem can while achieving the same block error rates. The schemes proposed by the The International Telegraph and Telephone Consultative Committee (CCITT) in recommendations V.32 and V.33 take advantage of what TCM has to offer for high-speed modem applications. Further studies into the areas of Trellis Shaping and Trellis Precoding potentially offer even higher gains, potentially approaching the theoretical limit for TCM applications on limited bandwidth channels.

The purpose of this project was to develop a malleable model that would simulate modem transmissions of V.32 and V.33 type standardized modems. The model was to allow the user control over the channel noise and the viterbi memory length so that the limits of noise tolerance and the viterbi decoder could

be studied in reference to TCM. The model was also to be modular so further extensions could easily be developed to enhance the model's abilities. For ease of use, the model was made to operate on standard ASCII text messages so users could operate the model without needing to create binary files beforehand.

This report includes a section on the history of modem technologies and an introduction to TCM. It also contains a detailed description of the model that was developed to aide in the teaching and research of TCM. The model includes controls for the channel noise and viterbi memory length and allows the user to choose what file they would like to transmit, what standard they would like to follow, and what they would like the report file to be called. Upon running, the model generates the following statistics on the transmission simulation made:

- Viterbi Memory Length Used
- Noise Reduction Factor Used
- # of Bits Received In Error
- # of Symbols Received in Error
- # of ASCII Characters in Error
- Mean X-Coordinate Error
- # of Bits Transmitted
- # of Symbols Transmitted
- # ASCII Characters Sent
- # of Demodulation Faults
- Data Rate Achieved
- Mean Y-Coordinate Error

These statistics are written in an easy to read format to the

file that is specified by the user. Many detailed analysescould be completed by using the above statistics to study a particular element of TCM.

To demonstrate the type of analysis that could be completed, this report also includes analyses of "Noise Tolerance" and "Viterbi Memory Length". These analyses were performed using the software model to illustrate the value of such a modeling tool. The final section of this report includes a section on enhancements that could be made to the software model that would further widen the range of analyses that could be conducted by using it. The Appendix of this report includes the source code that was written to generate this model. The code is well documented so it can be easily modified to include whatever elements future students and researchers desire.

TABLE OF CONTENTS

1. Modem Background 1
1.0 Introduction 1
1.1 Quadrature Amplitude Modulation (QAM) 3
1.2 Error Control 8

2. Trellis-Coded Modulation(TCM) 10
2.0 TCM Description 10
2.1 CCITT TCM Standardization Efforts and Schemes . 16
2.2 Trellis Shaping 25
2.3 Trellis Precoding 28

3. The Software Model 30
3.0 High-Level Description 30
3.1 Binary-to-Text Converter 33
3.2 Binary Reader 34
3.3 Differential Encoder 37
3.4 Convolutional Encoder 39
3.5 TCM Mapping Module 43
3.6 Channel\Noise Simulator 44
3.7 Decoder 47
3.7.1 Convolutional Encoder State Analysis . . . 47
3.7.2 The Viterbi Decoder Implementation 53
3.7.3 Decoding The Differential Encoding 57
3.8 Binary To Text Converter 58

4. Example Analyses 60
4.0 V.32 and V.33 Analysis 60
4.1 Transmission In The Presence Of Noise 60
4.1.1 Signal Constellation Noise Tolerance . . . 62
4.1.2 V.32 vs. V.33 in Gaussian Noise 65
4.2 Viterbi Window Length 69

5. Model Enhancements 74
5.0 Potential Extensions To The Model 74
5.1 Modify Channel Simulation 74
5.1.1 More Robust Noise Control 74
5.1.2 Burst Error Simulation 75
5.2 Supply Additional Modulation Techniques 76
5.2.1 ADD QAM 76
5.2.2 Add Other Modulation Schemes 77
5.3 Enhance The Decoder 78
5.3.1 Handling Larger Transmissions 78
5.3.2 Add Class Manipulation 79
5.3.3 Add Multi-Dimensional Options 80
5.4 Add Variable Block Sizes 81
5.5 Add Redundant Modeling Shell 82
5.6 Add Differential Decoding Statistics 83

LIST Of FIGURES

Figure 1: Four-State DPSK 2

Figure 2: 9.6 Kbits/s QAM Signal Constellation 4

Figure 3: 16-Point Nonredundant V.32 Signal Coding Structure 7

Figure 4: TCM Signal Constellation with 256 Points . . 12

Figure 5: Coarse Recognition, The Distance to Each Class 12

Figure 6: The Eight-Dimensional Constellation Proposed by Codex Corp. 15

Figure 7: Convolutional Encoder Recommended for 9.6 kbps Use by V.32 20

Figure 8: Signal Space Mapping Diagram for 9.6 kbps Recommended in V.32 20

Figure 9: The Signal Point Mapping Constellation for 14.4 kbps 22

Figure 10: The Convolutional Encoder Recommend for 14.4 kbps in V.33 23

Figure 11: Signal Point Mapping for 12.0 kbps Fallback Rate 23

Figure 12: The Multiplexer Configurations from V.33 for 14.4 kbps 24

Figure 13: The Multiplexer Configurations for the 12.0 kbps 25

Figure 14: Conventional Binary TCM Encoder 26

Figure 15: Coded Modulation Encoder with Trellis Shaping 27

Figure 16: TCM Software Model Block Diagram 31

Figure 17: Differential Encoder Specified in Both V.32 and V.33 39

Figure 18: Convolutional Encoder With State Reduction Points Marked 41

Figure 19: Convolutional Encoder State Analysis Points 48

Figure 20: Trellis Diagram For V.32/V.33 Convolutional Encoder 51

Figure 21: An Example V.32 Model Report File 61

Figure 22: An Example V.33 Model Report File 61

Figure 23: Data Rate vs. Constellation Point Error For V.32 63

Figure 24: Data Rate vs. Constellation Point Error For V.33 64

Figure 25: Noise Reduction Factor vs. Data Rate For V.32 65

Figure 26: Noise Reduction Factor vs. Data Rate For V.33 66

Figure 27: V.32 and V.33 Noise To Data Rate Plots . . . 68

Figure 28: Noise Reduction Factor vs. Average Constellation Point Error 69

Figure 29: Viterbi Window Length vs. Demodulation Faults, V.32 71

Figure 30: Viterbi Window Length vs. Demodulation Faults, V.33 73

LIST Of TABLES

Table 1: Differential Encoding for 9.6 kbps Nonredundant Coding 7

Table 2: Tablarized Signal State Mapping for 9.6 kbps Nonredundant Coding 8

Table 3: Signal Space Mapping Table for 9.6 kbps Recommended in V.32 19

Table 4: Differential Encoding for 14.4 kbps Recommend in V.33 21

Table 5: State Table For V.32/V.33 Convolutional Encoder 50

Table 6: Differential Decoder Look-Up Table 58

APPENDIX

FUNCTION MAIN(ARGC,AGRV) 86

FUNCTION TEXTCONVERTER(CHAR PATHFILE[25]) 87

FUNCTION BINARYREADER33
(VIT_MEM_LENGTH,ENCODER_TYPE,NOISE,TFILE,RFILE) 89

FUNCTION DIFFENCODER(INT Q1, INT Q2) 127

FUNCTION ENCODER() 129

FUNCTION ENCODER2() 130

FUNCTION DE_DIFFENCODER(BIT_1,BIT_2,DEC1,DEC2) 136

FUNCTION TOTEXT() 139

FUNCTION BINARYREADER32
(VIT_MEM_LENGTH,ENCODER_TYPE,NOISE,TFILE,RFILE) 140

FUNCTION MAPPERVDOT33(X,Y) 154

FUNCTION MAPPERVDOT32(X,Y) 170

FUNCTION STATE_FROM1
(POSS_SYMBOL,STATE_TO,STATE_FROM_INDEX) 175

FUNCTION STATE_FROM2
(POSS_SYMBOL,STATE_TO,STATE_FROM_INDEX) 187

FUNCTION STATE_FROMV32
(POSS_SYMBOL,STATE_TO,STATE_FROM_INDEX 199

FUNCTION REPORT32
(VML,NRF,XCE,YCE,DF,NUMXY,INPUT_FILE,REPORT_FILE) . . . 206

FUNCTION REPORT33
(VML,NRF,XCE,YCE,NUMXY,INPUT_FILE,REPORT_FILE) 209

1. Modem Background

1.0 Introduction

The first modems developed used frequency-shift keying (FSK) to encode the data. Using this scheme, the transmitted signal shifts back and forth between two frequencies: one representing a 1 and the other representing a 0. Modems of this type are crude and are not capable of running at more than 450 bps.

To increase the data rate, the next wave of modems used phase-shift keying (PSK) to encode the data. At first, PSK was used similarly to FSK. One phase would represent a 1 and 180 degrees opposite of it would represent a 0. The problem that was encountered with this scheme was the high dependency it had on the synchronicity of the clocks. If the clocks were not precise, the receiver would have no way of knowing which phase represented which symbol. This problem was avoided by the introduction of differential phase-shift keying (DPSK). DPSK is a method by which a phase transition, not the actual phase, is used to indicate a logic level. No phase change represents one level and a phase transition represents the other level. This solved the synchronization problem, but does not increase the data rate of the modem. Assuming a 2400 Hz bandwidth and a full-duplex channel, a two-state DPSK modem is only capable of supporting 600 bps. To increase the data rate, four-state DPSK was introduced.

Four-state DPSK uses 4 phase values in 90 degree increments to encode two-bit codewords, hence doubling the data rate for a given clock rate. The four-state DPSK state diagram is illustrated in Figure 1. This scheme is capable of transmitting 2400 bps (1200 bps in a full-duplex mode. From here on, half-duplex rates are referred to). To achieve even higher data rates, more states need to be created. The next obvious step would be to implement an eight-state DPSK modulation scheme. However, this is rarely ever implemented

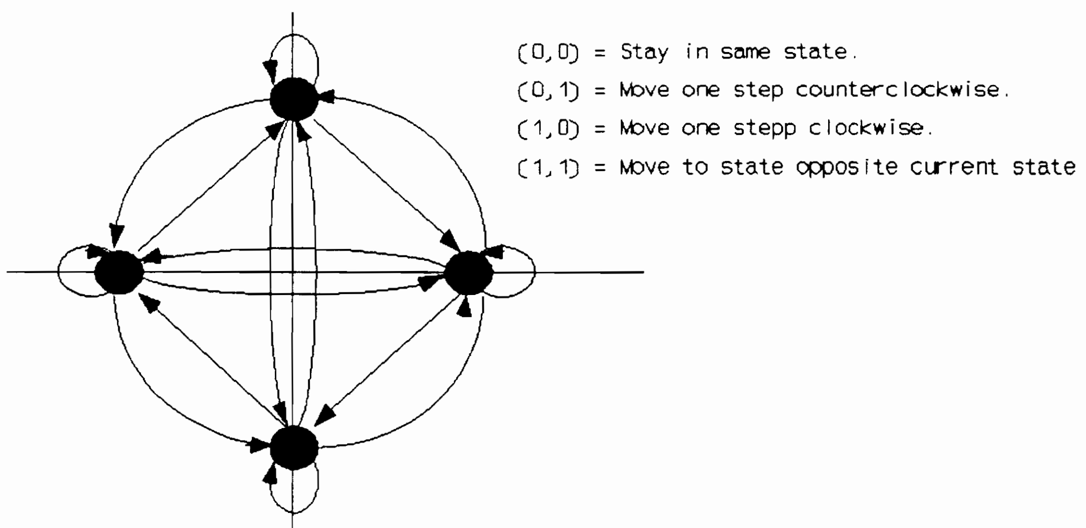


Figure 1: Four-State DPSK

because of the difficulties involved in recognizing small differences in phase. For these reasons Quadrature Amplitude Modulation (QAM) was introduced, and Trellis Coding Modulation

(TCM) techniques were added to further extend QAM's data rate. It is these techniques and the standards that govern them that will be discussed and analyzed in the body of this paper.

1.1 Quadrature Amplitude Modulation (QAM)

All modern, nonredundant high-speed modems use a form of QAM. In QAM, two carriers in quadrature are amplitude modulated and combined to produce the transmitter output. The receiver separates and demodulates the two in-quadrature signals thus recovering the transmitted information. The modulation for each symbol is then characterized by two amplitudes known as a signal point. Each signal point can be represented by a point in the X-Y plane. A QAM modem that sends 4 bits per symbol supporting 9.6 kbps (2400 baud) is illustrated in Figure 2 by it's signal constellation. This particular constellation is the one recommended by CCITT V.29 for 9.6 kbps leased-line modems.

At the receiver, the received words are mapped onto the constellation to demodulate the message word. Due to line impairments such as noise, phase jitter, and nonlinear distortion, a "best guess" must be made in attempting to ascertain which word was transmitted. If the transmission characteristics were ideal, the received signal point would directly map onto the signal point that was transmitted. This, however, is obviously not the case. Line impairments are inherent in the 3002 Hz General Switched Telephone Network

(GSTN) most often implemented in modem applications. The best strategy is simply to choose the signal point that is closest in euclidean distance to the received point.

Another limiting stipulation put upon modem modulation techniques is the signal power. The telephone companies limit the signal power that can be transmitted onto the line (the signal power is proportional to the average squared amplitude

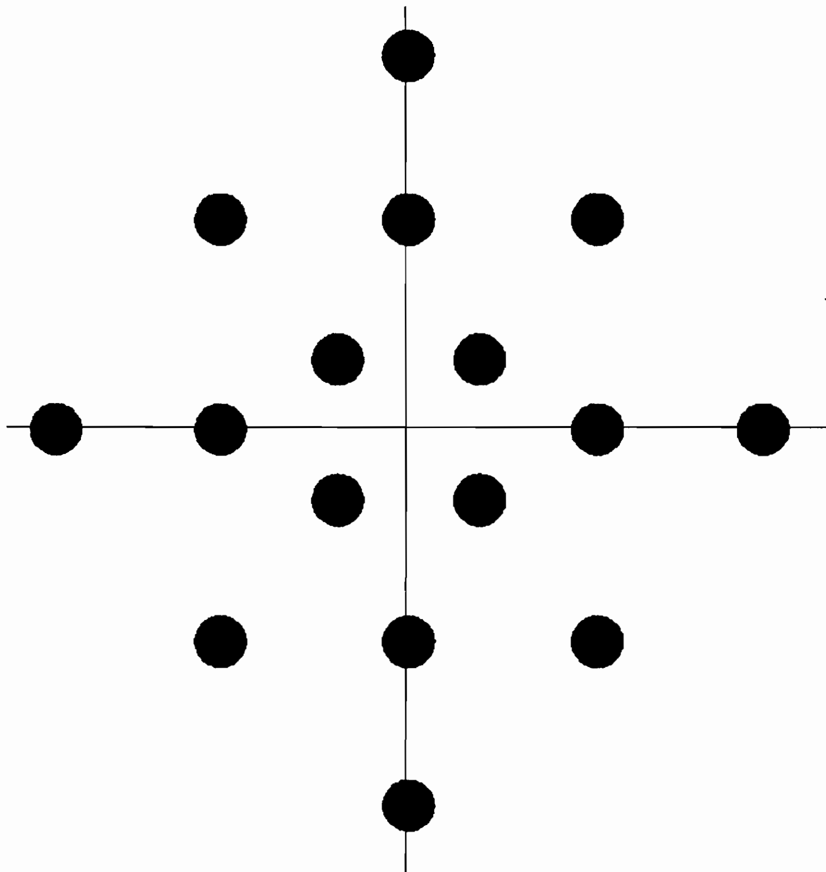


Figure 2: 9.6 Kbits/s QAM Signal Constellationof of the

signal points). The combination of the line impairments and the power stipulations invoked make communicating at higher data rates by increasing the number states by a factor of 2 very susceptible to error. Increasing the number of states causes the signal points to be pushed closer together, and it is not permissible to increase the signal power in an effort to alleviate this problem. Therefore, the modem's immunity to noise is reduced each time more states are added to the constellation. Typically, a noise power of N/2 in a 2^{m+1} state constellation will cause as many errors as a 2^m state constellation with N noise power. Therefore, to achieve data rates higher than 9.6 kbps, it is necessary to consider error control in the performance evaluation of a modem.

Modem performance is characterized by the error rate as a function of the signal to noise (S/N) ratio expressed in decibels (dB). High-speed synchronous modems typically use synchronous link protocols that use error detection and retransmission techniques usually referred to as automatic repeat requested (ARQ). The data is usually transmitted in blocks (1,000 bits for example), each time one or more bits in a block is in error, the whole block must be retransmitted. Therefore, as the number of signal points (or equivalently the bits/symbol) increases past 16 (4 bits/symbol) the inherent noise immunity causes QAM to be unacceptable without more rigorous error control techniques. Before more rigorous error

control techniques are discussed, the QAM-variant signal constellation scheme recommended in CCITT V.32 for 9.6 kbps over the GSTN will be discussed and illustrated. In accordance with recommendation V.32, the nonredundant coding scheme for 9.6 kbps is as follows.

The scrambled data stream is divided into groups of 4 consecutive data bits, each the scrambled representation of a particular symbol or signal point. The scrambler/ descrambler operate as is discussed in the following section on standardization efforts and TCM. The first two bits in time, $Q1_n$ and $Q2_n$ are then differentially encoded into $Y1_n$ and $Y2_n$ according to Table 1, where the subscript n denotes the sequence number of the group. The four bits $Y1_n$, $Y2_n$, $Q3_n$, and $Q4_n$ are then mapped into the coordinates of the signal state to be transmitted according to the signal space diagram illustrated in Figure 3. This mapping is tablarized in Table 2 for ease of mapping and reading.

The first two-bits are differentially encoded to assure that every rotation of 90 degrees of a codeword is another codeword. This way a temporary interruption in carrier-phase synchronization will not effect the demodulated data stream. This has the effect of making end-to-end transmission transparent to 90 degree offsets of the carrier phase.

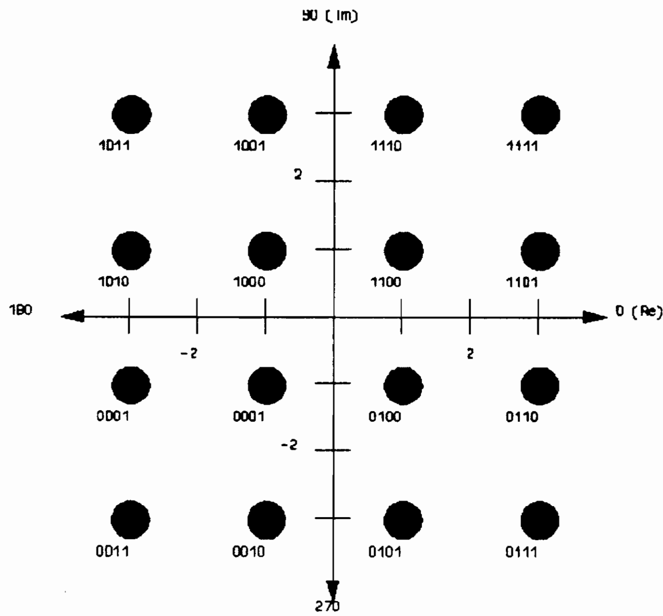


Figure 3: 16-Point Nonredundant V.32 Signal Coding Structure

Table 1: Differential Encoding for 9.6 kbps Nonredundant Coding

Inputs		Previous outputs		Phase quadrant change	Outputs	
$Q1_n$	$Q2_n$	$Y1_{n-1}$	$Y2_{n-1}$		$Y1_n$	$Y2_n$
0	0	0	0	+ 90°	0	1
0	0	0	1		1	1
0	0	1	0		0	0
0	0	1	1		1	0
0	1	0	0	0°	0	0
0	1	0	1		0	1
0	1	1	0		1	0
0	1	1	1		1	1
1	0	0	0	+ 180°	1	1
1	0	0	1		1	0
1	0	1	0		0	1
1	0	1	1		0	0
1	1	0	0	+ 270°	1	0
1	1	0	1		0	0
1	1	1	0		1	1
1	1	1	1		0	1

Table 2: Tablarized Signal State Mapping for 9.6 kbps Nonredundant Coding

Coded inputs				Nonredundant coding	
Y1	Y2	Q3	Q4	Re	Im
0	0	0	0	-1	-1
0	0	0	1	-3	-1
0	0	1	0	-1	-3
0	0	1	1	-3	-3
0	1	0	0	1	-1
0	1	0	1	1	-3
0	1	1	0	3	-1
0	1	1	1	3	-3
1	0	0	0	-1	1
1	0	0	1	-1	3
1	0	1	0	-3	1
1	0	1	1	-3	3
1	1	0	0	1	1
1	1	0	1	3	1
1	1	1	0	1	3
1	1	1	1	3	3

1.2 Error Control

Besides using ARQ to solely control error, binary forward error correction (FEC) can be used to avoid retransmissions by reducing receiver error. This scheme generates extra code bit(s) from the incoming data stream and then transmits the data and error control bits together. These extra control bits provide a means to further separate the data, i.e. provide a greater Hamming distance between adjacent streams of bits. The greater the Hamming distance the greater its ability to correct corrupt messages on the receiving end.

Unfortunately, traditional FEC techniques encode the data stream before it reaches the transmitter, hence the transmitter does not differentiate between the redundant bit(s) and the message bit(s). Because of this ordering, the redundant bits cannot be used in a systematic way to select the sequence of the transmitted signal points. In other words, there is not one code bit that can be associated with each symbol. To combat this problem, a more sophisticated modulation and coding method was developed. This method is called Trellis-Coded Modulation (TCM). TCM uses redundant bits that are generated from and associated with each symbol to allow systematic handling on the receiving end.

2. Trellis-Coded Modulation(TCM)

2.0 TCM Description

TCM is a method that uses the redundant bits(s) generated by a convolutional encoder to select the sequence of transmitted signal points. Adding a code bit to each symbol achieves a coding gain without reducing the data rate or requiring more bandwidth. Coding gains on the order of 6 dB are obtainable. Another advantage that TCM offers is the ability to separate signal points into classes. The N bits output by the convolutional encoder can be used to classify the signal points into one of 2^N classes. This technique improves the noise characteristics by providing a means to further separate the signal points. The constellation contains twice as many points due to the extra bit, hence has more symbols than are necessary to represent all of the bit combinations. These 'extra' symbols are used to rule out some transitions and allow information from previous symbols to be used by introducing dependencies between successively transmitted points. These dependencies are used to rule out illegal symbol strings and choose the closest symbol that remains at the decoding end.

A general TCM scheme takes N bit symbols and transforms them into N+1 bit symbols by using a convolutional encoder to add a redundancy bit. The N+1 bits are then mapped onto one of the 2^{N+1} signal points that make up the constellation. The

convolutional encoder is designed to transmit certain sequences and the subset mapping rule ensures that the sequences are unique. The extra signal points enable the encoder to introduce dependencies between successively transmitted symbols. Thus, only certain patterns or sequences of the signal points are valid. Other sequences, which are not valid, are never transmitted. This forced dependency adds separation between allowable sequences which more than makes up for the constellation compression that is caused by the additional symbols. Another way signal point separation is introduced is through decoding by breaking up the symbols into classes. The output of the convolutional encoder is used to specify the class of the particular signal point adding a further separation of points. Figure 4 illustrates a typical signal constellation and the classes of points for a scheme with N=7 making 256 ($2^{N+1} = 256$) constellation points. Line impairments cause the signal point to drift from the transmitted value making proper decoding essential. The generic process of decoding with TCM consists of two steps, coarse recognition and fine recognition. The coarse step calculates the distance to the nearest point of each class and marks them as the most likely candidates to be transmitted as illustrated in Figure 5. The fine recognition step uses the recorded information (past received points) to examine the dependencies introduced by the encoder. Using past and

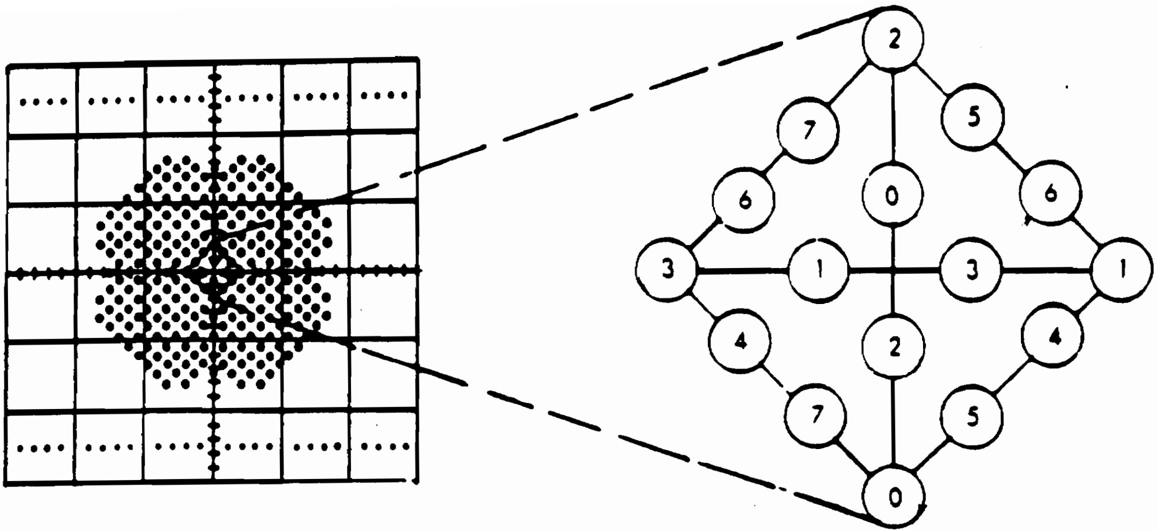


Figure 4: TCM Signal Constellation with 256 Points

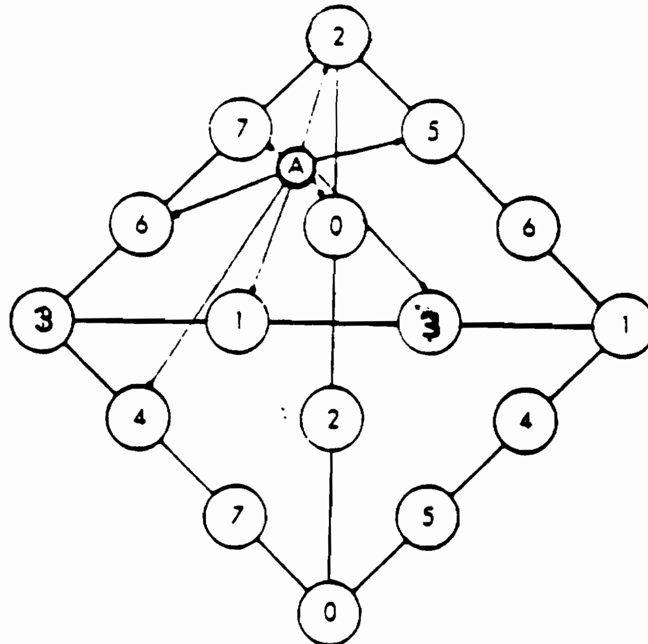


Figure 5: Coarse Recognition, The Distance to Each Class

subsequently received points, the decoder decides which signal point was most likely transmitted. The decoding algorithm looks at many successively received signal points before drawing a conclusion. It typically uses between 5 and 7 constraint lengths of memory to make its best guess to the identity of a single signal point. The constraint length is defined to be N of a 2^N point signal constellation.

The decoding algorithm used is the viterbi algorithm. The viterbi algorithm recursively searches all of the possible encoded sequences efficiently. It stores a path history of the most likely state transitions leading up to each possible state. As each new symbol is received, all possible extensions of the stored paths are considered. Minimum code distance from valid sequences is the deciding factor. The code distance measure used is the sum of the squared absolute distances between individual signal points that make up two signal point sequences that are being compared. For example, if X_1, X_2, X_3 is one sequence and Y_1, Y_2, Y_3 is another, the distance between them is $[(X_1-Y_1)^2 + (X_2-Y_2)^2 + (X_3-Y_3)^2]$. TCM modems of this general type typically show a 3 dB improvement over their QAM counterparts. For even higher gains, Codex Corp. has researched Multidimensional TCM.

Multidimensional TCM has been designed with the goal of achieving reliable data transmissions at a rate of up to 19.2 kbps over voice grade lines. Multidimensional TCM is

considered the state-of-the-art modulation technology today. The most recognized coding scheme was developed by Codex Corp., and it utilizes 64 states and 8 dimensions. Essentially, they have concatenated 4 two-dimensional signal constellations into one eight-dimensional constellation. There are 4 classes of signal points in each of the four two-dimensional constellations for a total of 16 classes and 64 states. Equivalent TCM encoding techniques are used to introduce dependencies between the eight-dimensional signal points. Only one redundant bit is added per eight-dimensional signal point rather than one per each two-dimensional signal point. Using less redundancy reduces the signal power and number of points to be transmitted because proportionally fewer bits are used. Even with less encoding bits, the eight-dimensional scheme uses these redundant bits in a more efficient manner via the more powerful 64-state trellis code. Symbol rates higher than the traditional 2400 baud can be used in conjunction with eight-dimensional TCM to improve noise immunity and reduce the number of signal points from 256 to 160 in each of the four concatenated two-dimensional constellations. An image of the eight-dimensional constellation that accomplishes this is depicted in Figure 6. Eight-dimensional TCM appears much more complicated than does two-dimensional, but it is only slightly more difficult

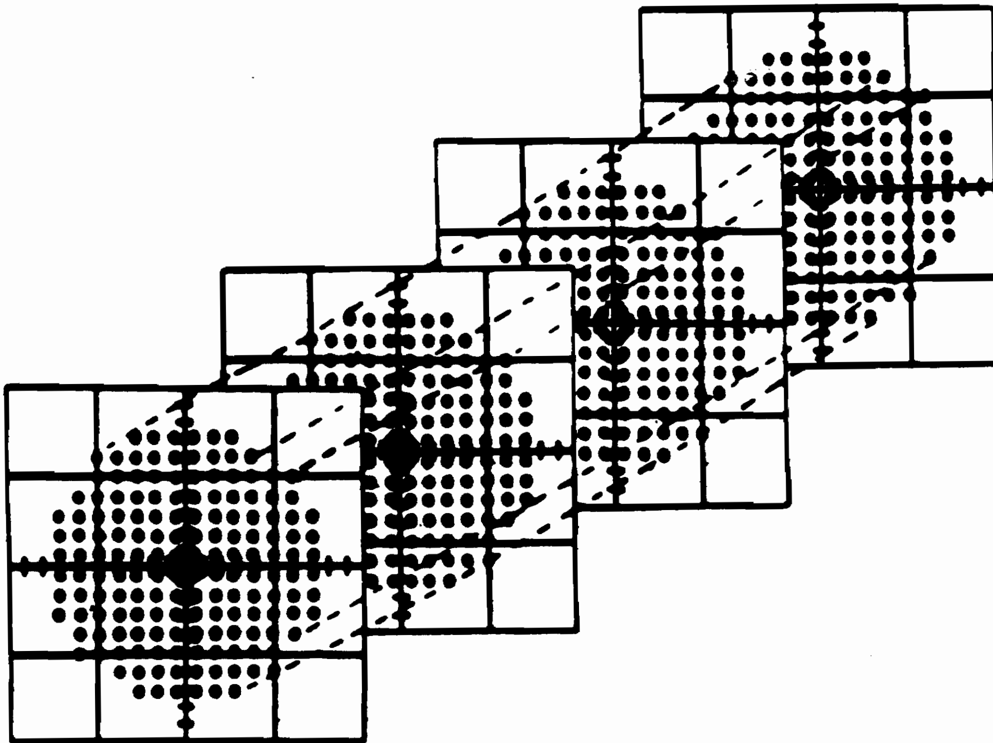


Figure 6: The Eight-Dimensional Constellation Proposed by Codex Corp.

because the TCM constructs are the same. The signal point recognition is essentially the same as is the backward looking mechanism in the decoder. The exploitation of the successive signal point dependencies is still evident. The advantage of eight-dimensional TCM (and potentially other multidimensional techniques) is the ability to approach the theoretical limit of the communication channel. Eight-dimensional TCM shows a 5.4 dB improvement over the uncoded 128 signal point QAM

constellation that 256 point two-dimensional TCM shows a 3 dB improvement over.

The CCITT has 2 standards currently in place that deal with standardizing TCM use over the GSTN and leased telephone lines for high-speed modem applications. Recommendation V.32 details TCM use over standard GSTN lines at 9.6 kbps and Recommendation V.33 details TCM over leased telephone circuits at 14.4 kbps with a fallback rate of 12.0 kbps. Currently, the CCITT has no formal recommendation that includes an eight-dimensional application, that technique is the property of Codex Corp.. The details of V.32 and V.33 as they pertain to TCM and high-speed modem applications are discussed in the following section.

2.1 CCITT TCM Standardization Efforts and Schemes

Along with specifying nonredundant coding schemes, Recommendation V.32 presents a TCM scheme that supports 9.6 kbps over GSTN lines (or leased-lines) that utilizes 32 carrier states. V.32 stipulates that the data will first be scrambled using the following generating polynomials, one for the calling modem and another for the answering modem.

Call Mode Modem Generating Polynomial (GPC) = $1 + X^{-18} + X^{-23}$

Answer Mode Modem Generating Polynomial (GPA) = $1 + X^{-5} + X^{-23}$

The transmitter effectively divides the message data sequence

by the GPC and the receiver multiplies the received data sequence by the same polynomial. The scrambled data stream that is to be transmitted is then split into groups of 4 consecutive data bits. The first two data bits in time, denoted $Q1_n$ and $Q2_n$ in each group are differentially encoded into $Y1_n$ and $Y2_n$ according to Table 3 where n denotes the sequence number of the group. The two differentially encoded bits are then fed into a systematic convolutional encoder which produces the redundant bit $Y0_n$. A block representation of the convolutional encoder used to do this is depicted in Figure 7. $Y0_n$, $Y1_n$, $Y2_n$, $Q3_n$, and $Q4_n$ are then mapped into the coordinates of the signal element to be transmitted in accordance with the signal space diagram illustrated in Figure 8. This is tabulated for ease of reading in Table 3.

The decoding process then takes place in the reverse order utilizing the viterbi algorithm to minimize error. The received bits are decoded using the reverse of the mapping procedure depicted in Figure 8 and then descrambled using the appropriate generating polynomial to recover the original data.

CCITT Recommendation V.33 makes recommendations for the use of 14.4 kbps rate modems with a fallback rate of 12.0 kbps over leased telephone lines. This scheme could also potentially be used on good quality GSTN lines.

The V.33 standard states that the scrambled data stream to

be transmitted be divided into groups of 6 consecutive data bits. (The generating polynomial used is the same GPC that is stated above for the V.32 9.6 kbps case.) The first 2 bits in time are then differentially encoded into $Y1_n$ and $Y2_n$ according to Table 4. The 2 differentially encoded bits are then input to a systematic convolutional encoder that generates the redundant bit $Y0_n$ where n again represents the sequence number. The redundant bit $Y0_n$ along with the six information carrying bits $Y1_n$, $Y2_n$, $Q3_n$, $Q4_n$, $Q5_n$, and $Q6_n$ are then mapped into the coordinate space depicted in Figure 9. The convolutional encoder recommended is illustrated in Figure 10 along with a reference to the signal mapping element.

The 12.0 kbps fallback rate that is recommended has a data stream that is divided into groups of 5 consecutive data bits. The trellis coding scheme that is illustrated in Figure 10 is utilized with the exception of $Q6_n$ being removed. The revised signal element mapping is illustrated in Figure 11. The decoding process works in reverse the same way as was stated for the V.32 case. The viterbi algorithm is used to assure minimal error propagation, and the reverse of the mapping and scrambling is performed.

V.33 also allows for the optional inclusion of a multiplexer that could be used for combining rates of multiples of 2400 bps up to the 14.4 kbps limit that the recommendation handles. The modulator bits are chosen such that the first bit in time

Table 3: Signal Space Mapping Table for 9.6 kbps Recommended in V.32

Coded inputs					Trellis coding	
(Y0)	Y1	Y2	Q3	Q4	Re	Im
0	0	0	0	0	-4	1
	0	0	0	1	0	-3
	0	0	1	0	0	1
	0	0	1	1	4	1
	0	1	0	0	4	-1
	0	1	0	1	0	3
	0	1	1	0	0	-1
	0	1	1	1	-4	-1
	1	0	0	0	-2	3
	1	0	0	1	-2	-1
	1	0	1	0	2	3
	1	0	1	1	2	-1
	1	1	0	0	2	-3
	1	1	0	1	2	1
	1	1	1	0	-2	-3
	1	1	1	1	-2	1
1	0	0	0	0	-3	-2
	0	0	0	1	1	-2
	0	0	1	0	-3	2
	0	0	1	1	1	2
	0	1	0	0	3	2
	0	1	0	1	-1	2
	0	1	1	0	3	-2
	0	1	1	1	-1	-2
	1	0	0	0	1	4
	1	0	0	1	-3	0
	1	0	1	0	1	0
	1	0	1	1	1	-4
	1	1	0	0	-1	-4
	1	1	0	1	3	0
	1	1	1	0	-1	0
	1	1	1	1	-1	4

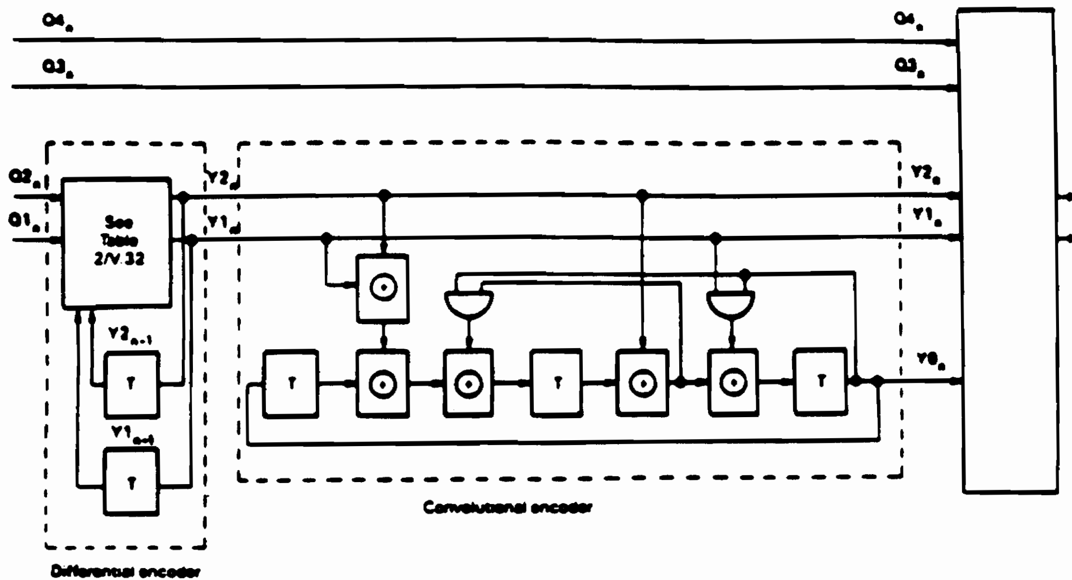


Figure 7: Convolutional Encoder Recommended for 9.6 kbps Use by V.32

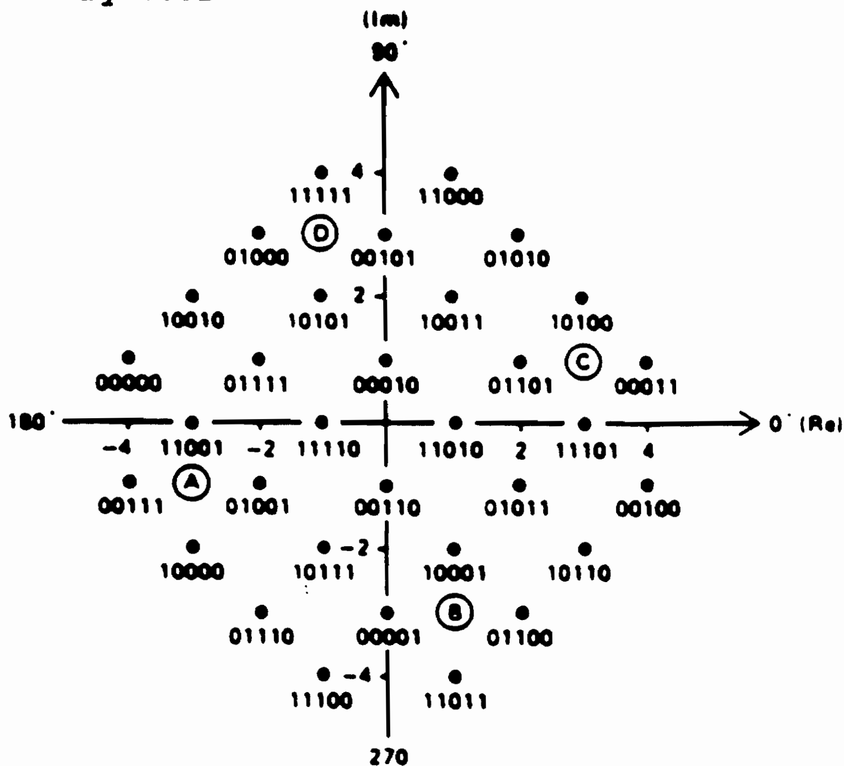


Figure 8: Signal Space Mapping Diagram for 9.6 kbps Recommended in V.32

Table 4: Differential Encoding for 14.4 kbps Recommend in V.33

Inputs		Previous outputs		Outputs	
$Q1_n$	$Q2_n$	$Y1_{n-1}$	$Y2_{n-1}$	$Y1_n$	$Y2_n$
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	0	0
0	1	1	0	1	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	1	1
1	0	1	0	0	1
1	0	1	1	0	0
1	1	0	0	1	1
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	1	0	1

is the sub-channel and is assigned to the first available bit in time of the modulator. Care is taken to separate the modulated bits of each multiplexed channel as far apart from each other as possible. The multiplex configurations suggested for the 14.4 and 12.0 kbps rates are illustrated respectively in Figures 12 and 13.

Driving future efforts to try and boost the transmission rates closer to the theoretical limits are studies into the areas of Trellis Shaping and Trellis Precoding. It is possible that in the near future, extensions to V.32 and V.33

will utilize these leading edge trellis techniques to boost standard use to 19.2 kbps and even beyond with varying baud rates.

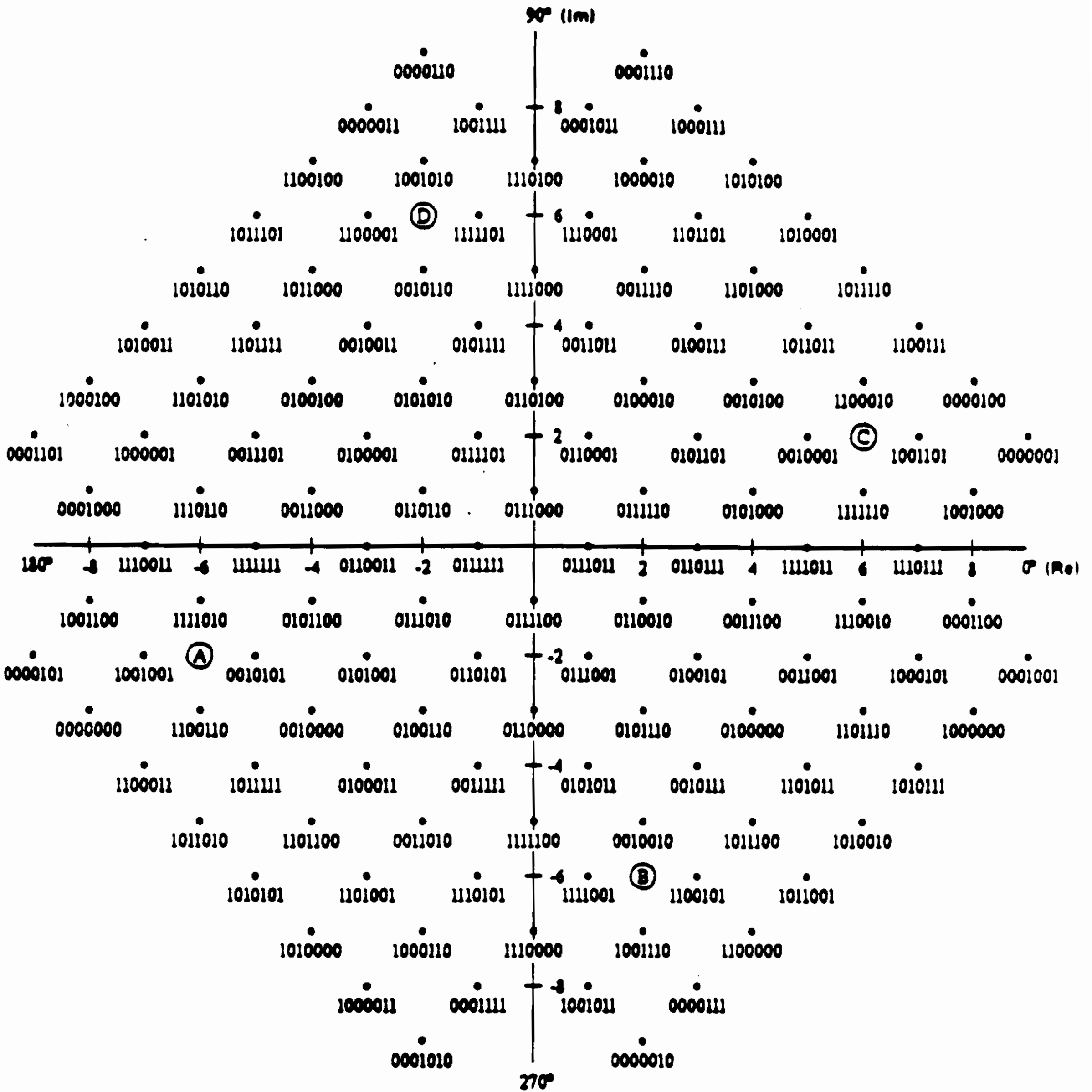


Figure 9: The Signal Point Mapping Constellation for 14.4 kbps

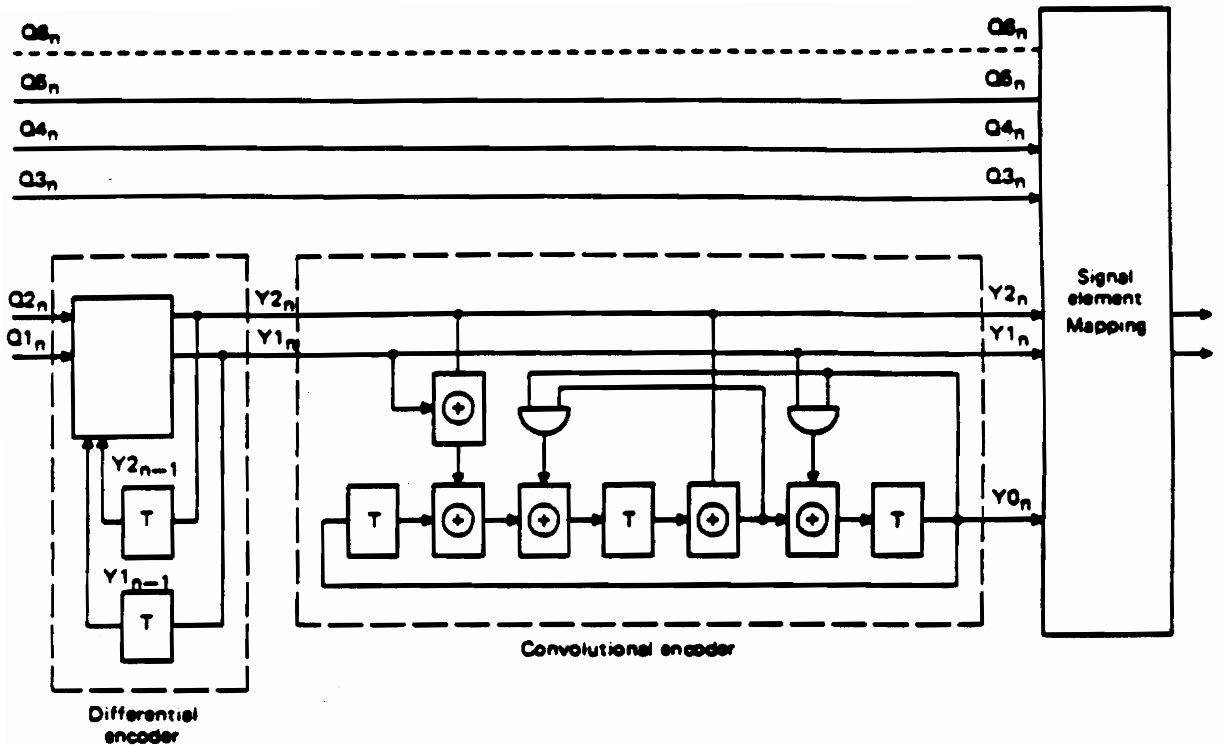


Figure 10: The Convolutional Encoder Recommend for 14.4 kbps in V.33

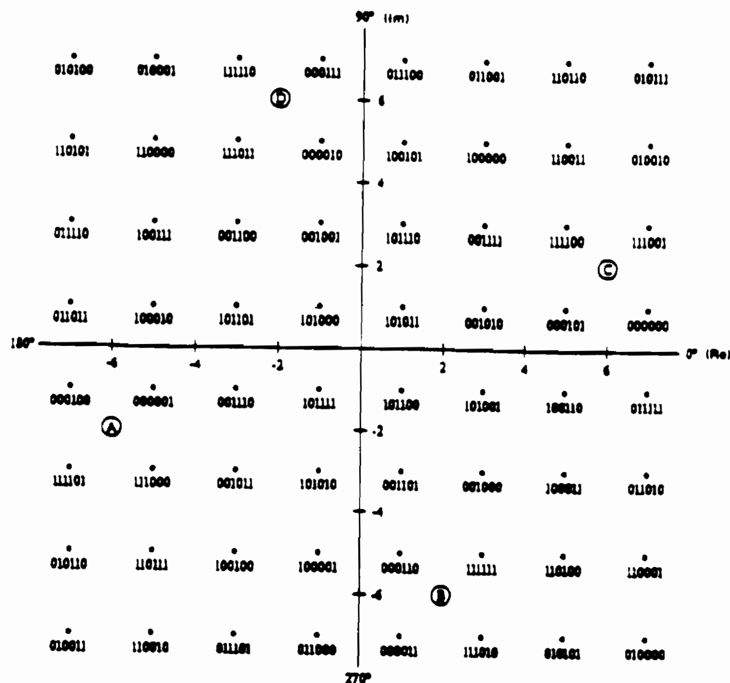


Figure 11: Signal Point Mapping for 12.0 kbps Fallback Rate

Aggregate data rate (bit/s)	Multiplex configuration	Sub-channel data rate (bit/s)	Multiplex channel	Modulator bits					
				Q1	Q2	Q3	Q4	Q5	Q6
14 400	1	14 400	A	X	X	X	X	X	X
14 400	2	12 000	A	X	X	X	X	X	
		2 400	B						X
14 400	3	9 600	A	X	X		X	X	
		4 800	B			X			X
14 400	4	9 600	A	X	X		X	X	
		2 400	B			X			
		2 400	C						X
14 400	5	7 200	A	X		X		X	
		7 200	B		X		X		X
14 400	6	7 200	A	X		X		X	
		4 800	B		X		X		
		2 400	C						X
14 400	7	7 200	A	X		X		X	
		2 400	B		X				
		2 400	C				X		
		2 400	D						X
14 400	8	4 800	A	X			X		
		4 800	B		X			X	
		4 800	C			X			X
14 400	9	4 800	A	X			X		
		4 800	B		X			X	
		2 400	C			X			
		2 400	D						X
14 400	10	4 800	A	X			X		
		2 400	B		X				
		2 400	C			X			
		2 400	D					X	
		2 400	E						X
14 400	11	2 400	A	X					
		2 400	B		X				
		2 400	C			X			
		2 400	D				X		
		2 400	E					X	
		2 400	F						X

Figure 12: The Multiplexer Configurations from V.33 for 14.4 kbps

Aggregate data rate (bit/s)	Multiplex configuration	Sub-channel data rate (bit/s)	Multiplex channel	Modulator bits					
				Q1	Q2	Q3	Q4	Q5	Q6
12 000	1	12 000	A	X	X	X	X	X	
12 000	2	9 600 2 400	A B	X	X	X	X	X	
12 000	3	7 200 4 800	A B	X	X	X	X	X	
12 000	4	7 200 2 400 2 400	A B C	X	X	X	X	X	
12 000	5	4 800 4 800 2 400	A B C	X	X	X	X	X	
12 000	6	4 800 2 400 2 400 2 400	A B C D	X	X	X	X	X	
12 000	7	2 400 2 400 2 400 2 400 2 400	A B C D E	X	X	X	X	X	

Figure 13: The Multiplexer Configurations for the 12.0 kbps

2.2 Trellis Shaping

Trellis Shaping is a method that selects the minimum weight sequence from an equivalence class of possible transmitted sequences. This method is performed by searching through a trellis diagram of a given convolutional code. Shaping gains on the order of 1 dB are obtainable with simple 4-state convolutional codes. Shaping gains of up to the 1.53 dB

theoretical limit can be achieved if more complicated codes are used to expand the constellation. Presently, most coded modulation schemes are generally designed for ease implementation. They only depend a small amount on the number of signal points and the specific choice of constellation to represent that number of signal points.

The general coded modulation encoder from which currently applied TCM is a subset of can be pictured as illustrated in Figure 14. As discussed previously, this encoder uses the coded bits to choose a subset of points from which to be a member. (subset and class is the same principle)

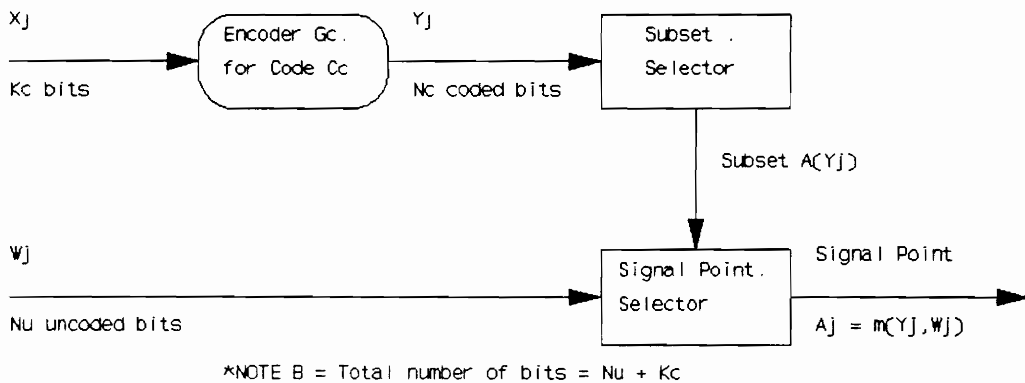


Figure 14: Conventional Binary TCM Encoder

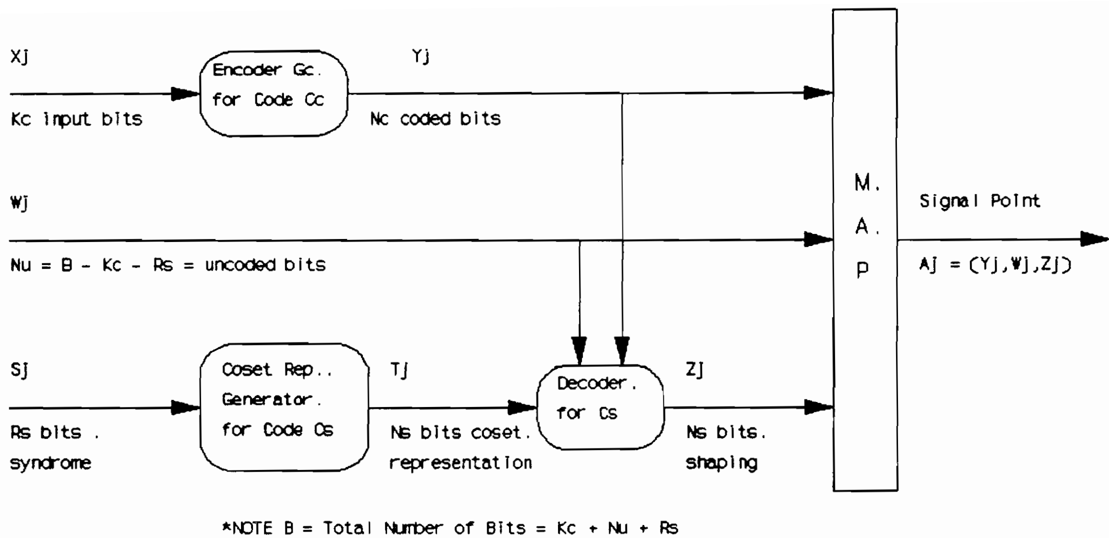


Figure 15: Coded Modulation Encoder with Trellis Shaping

consideration is given to the shaping of the constellation and choosing optimal groupings (classes) of the signal points. To take these factors into consideration, trellis shaping must be introduced into the encoding scheme. Figure 15 illustrates a general coded modulation scheme of an encoder that uses trellis shaping.

The key difference in this encoder is the shaping operation which takes place in the encoder block for the convolutional code C_s. The coset representative generator $(H_s-1)T$ is used as a third input to the encoding scheme which adds information about the N dimensional space being mapped to. Using this information, signal points are mapped more efficiently into

the aj = $m(y_j, w_j, z_j)$ space. Simulations have been run that show a 1 dB improvement for simple 1/2-rate 4-state convolutional codes. Trellis shaping is compatible with an coded modulation scheme and can be used with a variety of convolutional codes. It should therefore be adaptable to current CCITT high-speed modem standard, and current studies are under way towards this goal. An important extension to trellis shaping is to intersymbol interference (ISI) channels. This extension introduces trellis precoding which will be discussed in the following section.

2.3 Trellis Precoding

Trellis precoding is a method that adapts the previously discussed coded modulation methods for use on non-Gaussian channels with ISI. This is necessary because of the major advances that have been made in the efficient use of band-limited channels. Current voice-band modems use linear equalizers in the receiver and this works fine as long as the ISI is not severe. However, once the maximum bandwidth of a band-limited channel is being utilized, spectral nulls or near nulls are often present and linear equalization is not able to handle it.

Trellis precoding is a method that achieves the equalization performance on an ideal decision-feedback equalizer, the coding gain of any known lattice code, and shaping gain. By correlating these three elements, trellis precoding should

bridge the gap to the theoretical channel capacities. It has been analytically proven that at high rates, trellis precoding can achieve essentially the same gains on ISI channels as on ideal channels. The capacity of ideal channels can be approached with ISI channels as long as the signal-to-noise ratio is suitably high.

Along with theoretically, trellis precoding has been successfully implemented by the Codex Corporation. Codex developed a voiceband modem (Codex 3600 Premium) that uses the widest possible bandwidth and can handle the severe distortion that is present at the band edges. Future applications into the areas of coded modulation and high-speed modem technologies is over ordinary subscriber lines which have nulls and near nulls as well as crosstalk above the standard voice band to contend with. The specifics of trellis precoding are highly mathematical and beyond the scope of this project on TCM and high-speed modems. For an in-depth look at the mathematical theory involved, refer to the CCITT working paper referenced in the bibliography.

3. The Software Model
3.0 High-Level Description

The software model is comprised of all the modules necessary to model the transmission of a text message end-to-end using either V.32 or V.33 CCITT standards. The user specified text message is converted to a bit stream, differentially encoded, convolutionally encoded, mapped to the signal constellation, injected with noise, decoded, and converted back to an ASCII text format. The model generates an output file that reports the statistics of the results of the transmission simulation.

The model allows for the line-noise, viterbi memory length, encoder type, CCITT Standard to follow, text file to be transmitted and the report file to be generated to be controllable at the input of the model. The software model is functionally composed of the following modules:

- Text-to-Binary Converter
- Binary Reader
- Differential Encoder
- Convolutional Encoder
- TCM Mapping Module
- Channel/Noise Simulator
- Viterbi Decoder
- Binary-to-Text Converter

The connectivity of these modules is illustrated in Figure 16.

The model is capable of running against any ASCII text file. The Text-to-Binary converter takes the ASCII input file and converts it to a stream of representative 1's and 0's. The

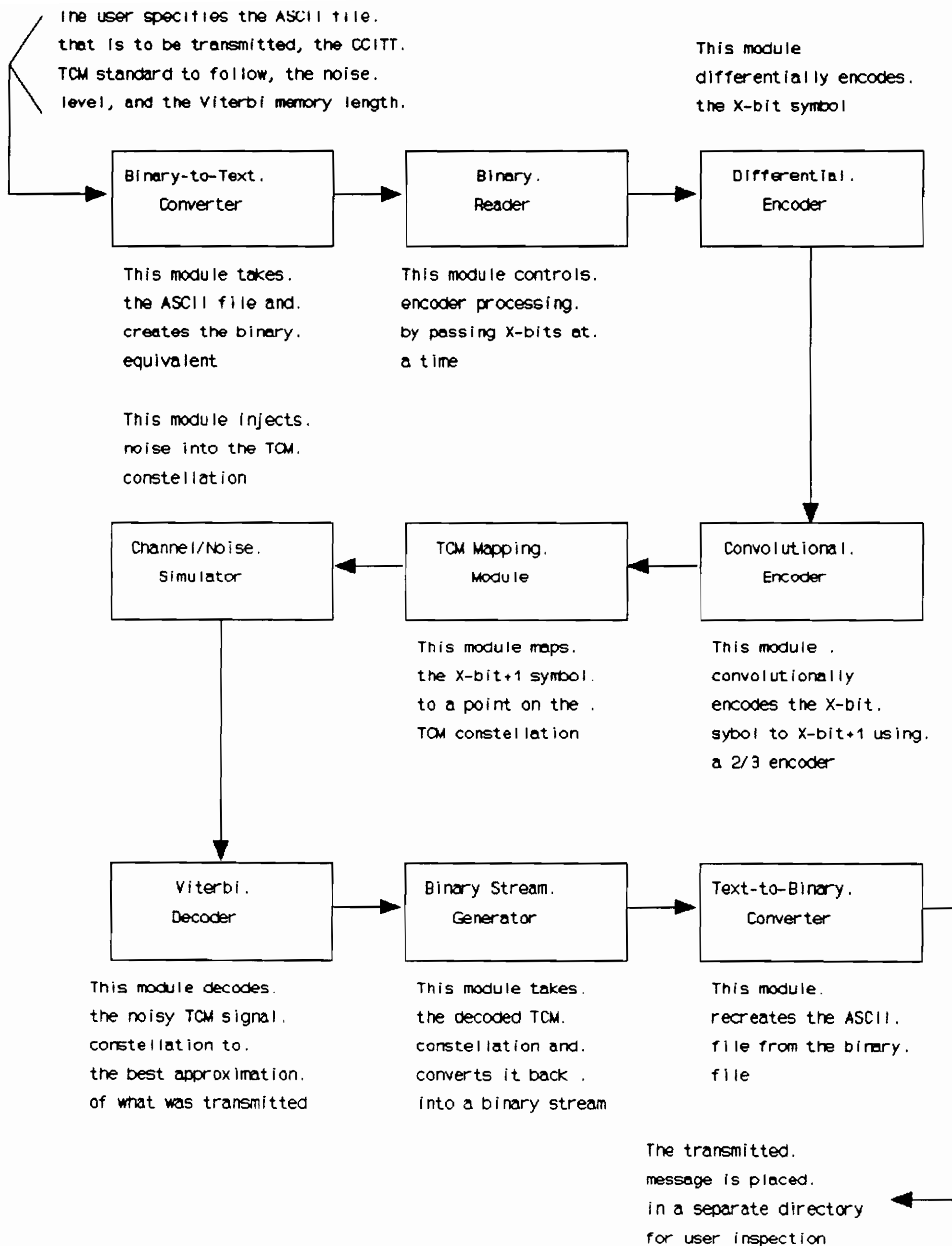


Figure 16: TCM Software Model Block Diagram

binary reader then grabs X-bit chunks (dependent upon the standard followed) of the binary file and passes it through the differential encoder. The output of the differential encoder is passed to the convolutional encoder where the extra bit is generated for each symbol. The output of the convolutional encoder is then passed to the mapping module that translates the X-bit string to the X-Y coordinate plane. The mapping function employed is dependent upon the CCITT standard that is being implemented. V.32 uses a 32-point signal space and V.33 uses a 128-point signal space. The signal space is then passed to the channel simulator where noise is injected into the system via two independent Gaussian Random Variables (GRVs). A GRV is defined for each direction in the coordinate plane. The channel output is then passed to the decoder where the signal space is reconstructed from the noisy signal. Once the decoding of a particular symbol is complete, it is appended to the binary stream. Once decoding is complete, the binary stream that is generated is fed to the Binary-to-Text converter where the ASCII text message is reconstructed.

This software model can be used to study TCM and the viterbi decoding algorithm for many purposes and from many points of view. The control that the user has at the beginning of a TCM simulation allows them to set the parameters to study whatever interests them. For example, it can be used to study:

- The limits on noise tolerance that TCM can handle in reference to the signal constellations.

- How the amount of decoding memory effects the decoding of a message and the number of demodulation faults encountered.

- How Differential Encoding extends decoding errors.

- The relationships between achieved data rate and line noise for both V.32 and V.33 transmissions.

The software model can be used from a research point-of-view to study the relationships of TCM parameters, or from an educational point-of-view to aid in the understanding of the underlying principles by the use of example. Interested students and researchers could also easily expand upon the model as the model is very modular and well commented. The final section of this report contains potential extensions to the model that would increase its power.

3.1 Binary-to-Text Converter

The Binary-to-Text converter reads the ASCII text file character by character converting each ASCII character to its 8-bit binary equivalent. The binary equivalence of carriage returns and line-feeds are also translated to make certain the file can be properly reconstructed to it's ASCII text message format. The text-to-binary translation occurs completely before the binary reader begins passing X-bit chunks to the Differential Encoder. This is facilitated by creating a new file (binary.txt) that represents the user's message with a

continuous stream of 1's and 0's.

3.2 Binary Reader

The binary reader is the controlling function of the model. There is a separate binary reader module for each CCITT standard, one for V.32 and another for V.33. This module feeds the input to the modem with X-bit chunks of data that are read from the binary.txt file that is generated in the Binary-to-Text converter. The X is dependent upon the rate of transmission which is dependent on the CCITT standard being modeled. X=4 for V.32 transmissions at 9,600 bits/sec and X=6 for V.33 transmissions at 14,400 bits/sec. The model allows the user to specify the transmission rate and TCM scheme to be implemented prior to running the model.

This module controls the flow of the program in the following manner. (The flow is the same for V.32 and V.33 implementations.) It first stores a copy of the all the possible states of the trellis to memory so the states can be accessed quickly. It then feeds the first 2 bits of the X-bit chunk to the differential encoder. The two bits output from the differential encoder are input to the convolutional encoder. The output of the convolutional encoder which includes the redundant bit that has been generated for that symbol is recoupled with the (X-2) remaining bits of the X-bit chunk that was not encoded. These (X+1) bits are then input to the signal constellation mapper which maps the entire

symbol to the appropriate X-Y coordinate pair. X-bit chunks are grabbed and stored until all of the binary.txt file is processed and the entire transmission is stored in an array of X-Y coordinate pairs.

Once the entire message has been encoded and mapped to a signal point and stored, the noise function is accessed from within the binary reader. The noise function adds gaussian noise to each coordinate pair, the magnitude of the noise used is also controllable at the input to the model. The specifics of the noise generator are discussed in a separate section below.

After the line noise has been simulated, the message is said to have been transmitted and is now ready to be decoded. A straight Viterbi decoder has been implemented. A 32-state decoder for V.32 and a 128-state decoder for V.33. The memory length of the decoder is controllable at the input of the program. Due to memory space on the PC, a maximum of 50 was set on this memory length. This maximum should not effect the model since it is over 7 constraint lengths larger than the constraint length of the V.33 modem. Once a symbol is decoded its binary equivalent is output to the binary file binary_o.txt. The internals of the decoding algorithm are described in a separate section below.

Once the entire transmitted message has been decoded as best as possible, the resulting file binary_o.txt is converted back

to an ASCII format by calling the Binary-to-Text converter. The resulting ASCII file is called text.txt and it can be compared to the transmitted text file to see if it is recognizable.

The last function that the binary reader calls is the reporting function. This reporting function compares the two binary files and uses internal variables that are generated during the encoding and decoding processes to generate the following statistics on the transmission:

- Viterbi Memory Length Used
- Noise Reduction Factor Used
- # of Bits Received In Error
- # of Symbols Received in Error
- # of ASCII Characters in Error
- Mean X-Coordinate Error
- # of Bits Transmitted
- # of Symbols Transmitted
- # ASCII Characters Sent
- # of Demodulation Faults
- Data Rate Achieved
- Mean Y-Coordinate Error

These statistics are written to the screen and to a user specified file. The report file is the last line argument accepted at the input of the model. The reporting function and all of the other functions mentioned in this section are further broken out in the following sections.

3.3 Differential Encoder

The differential encoder is the first step in the encoding process. It encodes the first two bits of the X-bit chunk which are in turn input to the convolutional encoder. These bits are differentially encoded to assure that every rotation of 90 degrees in phase is another codeword. This allows a temporary interruption in carrier-phase synchronization to have no effect on the demodulated data stream. Although phase is not an issue in the transmission scheme of the model, the differential encoder and decoder is included for completeness. It is also important to note that the decoding of the differential encoder extends errors of improperly decoded signal points. The omission of the differential encoder and decoder would give improper throughput results because the error extension phenomena would not be modeled.

A block diagram of the differential encoder used for V.32 and V.33 transmissions is illustrated in Figure 17. It is implemented in the model by reconstructing the truth table that is illustrated in Table 1.

Upon studying the truth table, one will notice that the first 8 rows can be generated using a simple exclusive or (XOR) operation bit wise on the inputs and previous outputs.

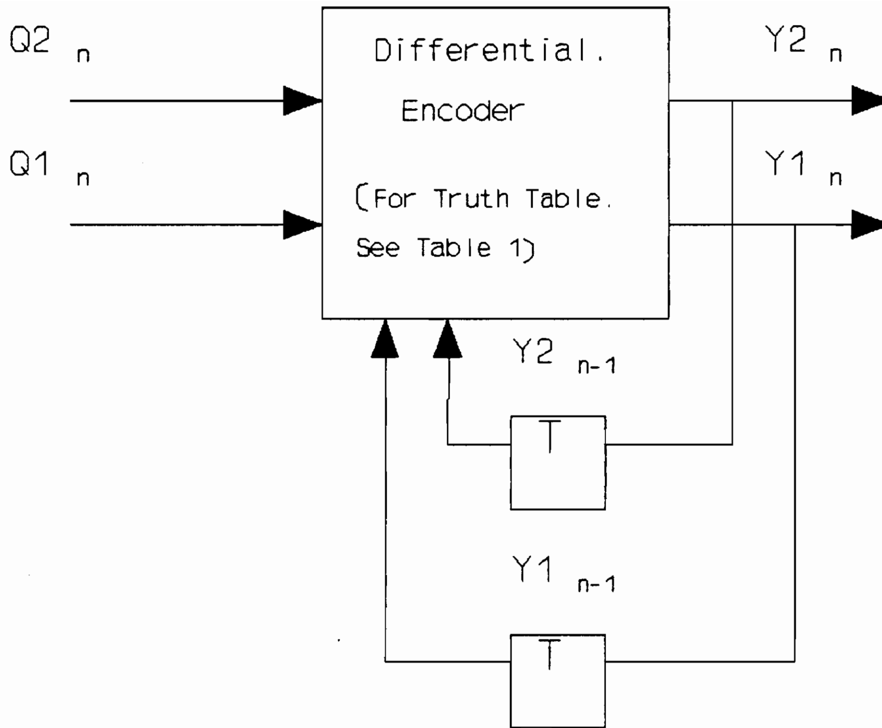


Figure 17: Differential Encoder Specified in Both V.32 and V.33

For example:

$$Y1_n = (Q1_n \text{ XOR } Y1_{n-1})$$

$$Y2_n = (Q2_n \text{ XOR } Y2_{n-1})$$

This is exactly how the first 8 rows of the truth tables are coded into the Differential Encoder. The remaining 8 rows did not possess such a convenient pattern, so they were hardcoded into the program using if-then logic.

This module is called from the binary reader and is sent the two bits it operates on ($Q1_n$ and $Q2_n$) which are inputs to the function call (Refer to Figures 7 & 10). The differential encoder returns to binary reader $Y1_n$ and $Y2_n$, which are the output of the differential encoder. For each run of the model, the initial previous output is set to $Y1_{n-1} = 0$ and $Y2_{n-1} = 0$ to ensure erratic initial states do not exist. The output of the differential encoder is the input the convolutional encoder which is discussed in the following section.

3.4 Convolutional Encoder

The convolutional encoder is the second step in the encoding process. The output of the differential encoder (both bits) is fed directly into the convolutional encoder. The convolutional (2/3 rate) encoder is used to generate the redundant bit that is appended to the symbol representation. This redundant bit doubles the number of signal points but allows a coding gain on the order of 6 db because there are more symbols than are necessary to represent the data. The extra symbols are used on the encoding and decoding end to separate the data. This more than makes up for the constellation compression that takes place because of the extra bit per symbol.

The convolutional encoder that is implemented in both V.32 and V.33 is illustrated in Figure 18. In order to implement

the encoder in the software model I had to derive equations that would represent the encoder. I developed these equations by a state analysis of the convolutional encoder. Each point after a delay plus point X2 was studied. These points are marked in Figure 18. The only variable of interest is the output, $Y0_n$. The purpose of the analysis was to boil down the equations to an algorithm that would allow me to calculate $Y0_n$ from what is known in a systematic way that could be implemented in 'C'.

Upon inspection of the block diagram, the following equations were evident:

1) $X0n = [Y0_{n-1}]$

2) $X1_n = [(X0_{n-1} \text{ XOR } (Y1_{n-1} \text{ XOR } Y2_{n-1})) \text{ XOR } (Y0_{n-1} \text{ AND } X2_{n-1})]$

3) $X2_n = [(X1_n \text{ XOR } Y2_n)]$

4) $Y0_n = [(X2_{n-1} \text{ XOR } (Y0_{n-1} \text{ AND } Y1_{n-1})]$

Since $X0n = [Y0_{n-1}]$ and $X2_n = [(X1_n \text{ XOR } Y2_n)]$, the above 4 equations can be reduced to the following 2 equations through algebraic manipulation:

1) $X1_n = [(Y0_{n-2} \text{ XOR } (Y1_{n-1} \text{ XOR } Y2_{n-1})) \text{ XOR } (Y0_{n-1} \text{ AND } (X1_{n-1} \text{ XOR } Y2_{n-1}))]$

2) $Y0_n = [(X1_{n-1} \text{ XOR } Y2_{n-1}) \text{ XOR } (Y0_{n-1} \text{ AND } Y1_{n-1})]$

These two equations can be used to equate $Y0_n$ from the inputs $Y1_n$ and $Y2_n$ at any given point in time. Sequentially setting the following variables completely models the convolutional encoder displayed in Figure 18. The algorithm is illustrated by the following set of algebraic equations.

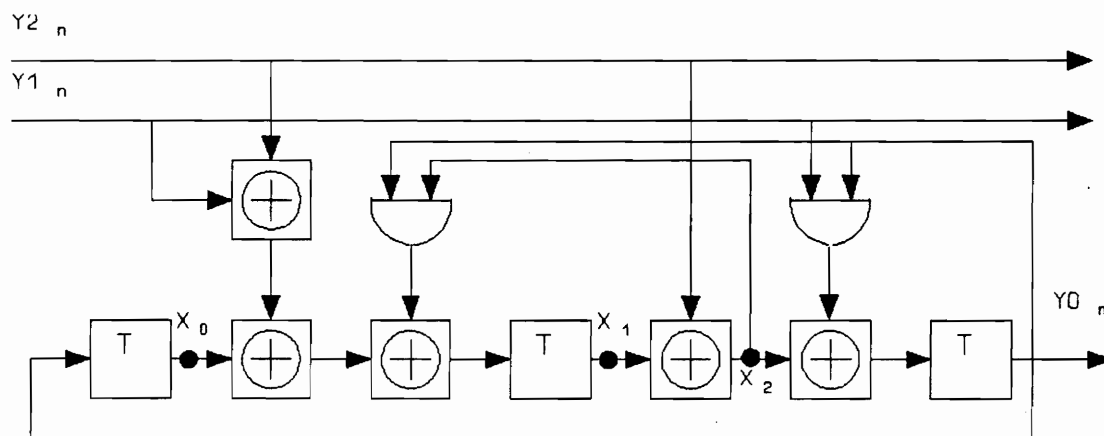


Figure 18: Convolutional Encoder With State Reduction Points Marked

Convolutional Encoder Algorithm

Initially: $Y0_{n-1} = 0$, $Y1_{n-1} = 0$, $Y2_{n-1} = 0$, $Y0_{n-2} = 0$ and $X0_{n-1} = 0$

Then:

$X0_n$ = [($Y0_{n-2}$ XOR ($Y1_{n-1}$ XOR $Y2_{n-1}$)) XOR ($Y0_{n-1}$ AND ($X0_{n-1}$ XOR $Y2_{n1}$))]

$Y1_n$ = Input bit Y1 from differential encoder.

$Y2_n$ = Input bit Y2 from differential encoder.

$Y0_n$ = [($X0_{n-1}$ XOR $Y2_{n-1}$) XOR ($Y0_{n-1}$ and $Y1_{n-1}$)]

$X0_{n-1}$ = $X0_n$

$Y1_{n-1}$ = $Y1_n$

$Y2_{n-1}$ = $Y2_n$

$Y0_{n-2}$ = $Y0_{n-1}$

$Y0_{n-1}$ = $Y0_n$

Starting this algorithm initialized as illustrated and feeding bits $Y1_n$ and $Y2_n$ from the output of the differential encoder to the convolutional encoder will generate the desired results. The output of the convolutional encoder ($Y2_n$, $Y1_n$ and $Y0_n$) can then be joined with the remaining symbol components which were not altered and passed to the TCM Mapping Module (refer to Figures 7 & 10).

Also, as an alternative convolutional encoding method and to verify that the actual V.32/V.33 encoder was properly implemented, an encoder mapping function was also implemented. The trellis that is defined by a state analysis of the encoder was used to hard-code the relationships of bits input to the encoder to bits output from the encoder dependent upon the previous bits output from the encoder. This state analysis is explained in detail in the section on decoding below. The

encoder mentioned first dynamically encodes the data as it is received, the latter merely uses a look-up table. The user can specify which encoding scheme they would like to use upon running the model. (encoder = 1 is the dynamic encoder, encoder = 2 is the hard coded encoder) The encoders are functional equivalents, the choice made has no effect on the output of the simulation. It was only implemented as a means of verification.

3.5 TCM Mapping Module

The TCM Mapping Module takes the (X-bit)+1 symbol and maps it to an X-Y coordinate point on the constellation. The mapping algorithms are defined in the V.32 and V.33 standards and are illustrated in Figures 8 and 9. These mapping algorithms are designed to take full advantage of the redundant bit per symbol that is generated by the convolutional encoder. The mapping algorithms take full advantage of the 8 classes by separating points in the same class as far apart as possible. This maximum distance allows for more noise tolerance by giving the decoding algorithm room to work by separating consecutively transmitted symbols as far apart from each other as possible.

The mapping algorithms were implemented in 'C' by brute force. After trying to engineer a more efficient way to do the mapping for some time, I realized a lengthy but simple If-Then structure would be the best approach. Once the

convolutional encoders output is joined with the other bits of the (X-bit)+1 symbol, that bit combination is matched to a specific X-Y coordinate by checking the value of every bit in the symbol. There are 32 possible combinations to check for in V.32 and 128 possible combinations in V.33. Each bit combination represents a different constellation point. Once the entire message has been encoded to signal points and stored in an array, the Channel/Noise Simulator is called.

3.6 Channel\Noise Simulator

The Channel\Noise Simulator was implemented by taking the mapped signal points and passing them through a noise generator that uses two random Gaussian variables to modify the constellation point. One is generated for the X-coordinate and one is generated for the Y-coordinate. The Gaussian Random Variable (GRV) generator used is described below.

Since the cumulative form of the normal distribution cannot be expressed in closed form using X-Y coordinates, the bivariate form was transformed. The bivariate turns out to be ideal for the case of this model since two GRVs are required for each constellation point. The inversion method that I implemented was developed by Box and Muller. The GRVs of X_1 and X_2 and defined as follows:

$X_1 = [-2\ln U_1]^{1/2}\sin(2(PI)U_2)$

$X_2 = [-2\ln U_1]^{1/2}\cos(2(PI)U_2)$

Algorithmically this can be described as follows:

U1 = U(0,1)   U2 = U(0,1)

(U(0,1) represents an RV between 0 and 1 inclusive.)

$R = (-2\ln U_1)^{1/2}$

$A = 2(PI)U_2$

$X_1 = RSin(a)$

$X_2 = Rcos(a)$

To apply this to constellation error injection I added another factor that allows the user to control the mean level of gaussian noise injected into the channel. I did this by allowing the input of a Noise_Reduction_Factor at the input of the model. This Noise_Reduction_Factor impacts the GRVs in the following manner:

$X_1 = RSin(a)/Noise_Reduction_Factor$

$X_2 = Rcos(a)/Noise_Reduction_Factor$

The signal point after having noise injected is then represented by:

$$\text{X-Coord(From Mapper)} = \text{X-Coord} + X_1$$

$$\text{Y-Coord(From Mapper)} = \text{Y-Coord} + X_2$$

Since the user specifies what the Noise_Reduction_Factor is, they can control the average amount of signal constellation disturbance that occurs. Included in the report that is generated after each run of the model is the mean signal point noise injected on the X and Y coordinate pairs. The user can use this feedback to gauge how to set the Noise_Reduction_Factor to generate the level of noise that they are interested in studying. It is also important to remember that the report only gives the mean disturbance for that simulation. Some of the signal points may likely have been injected with noise far from the mean, either on the high or low side. Therefore, errors could potentially be injected even if a large noise reduction factor is used. The use of a Noise_Reduction_Factor of greater than 25 should statistically rule out the likelihood of X_1 or X_2 having a significant absolute value. The user also has the option of using a Noise_Reduction_Factor < 1.0 to study cases of extreme noise.

3.7 Decoder

The decoder is composed of decoding algorithms for decoding the convolutional code and separate ones for decoding the differential code. Before the decoding can be discussed it is important to discuss the convolutional encoder that was invoked and to do a state analysis on it so the viterbi algorithm can be properly applied. With that in mind, this section begins with a subsection on the state analysis of the convolutional encoder that had to be completed before the decoder could be implemented. The implementation sections on decoding follow the state analysis.

3.7.1 Convolutional Encoder State Analysis

To accomplish a state analysis of the convolutional encoder, all possible states of the encoder had to be defined. The states of an encoder are completely defined if the point after each delay is made an element of the state and each possible input combination is considered an instance of the state. The state element points of the V.32/V.33 encoder are marked in Figure 19. There are 3 delays for a possible 8 states, and there are 2 input bits for 4 different instances of the 8 states. This means that there are 32 potential combinations the input and encoder state could be in.

The goal of the state analysis of this convolutional encoder was to quantify the allowable state traversal scheme and the input and output stream associated with each legal state

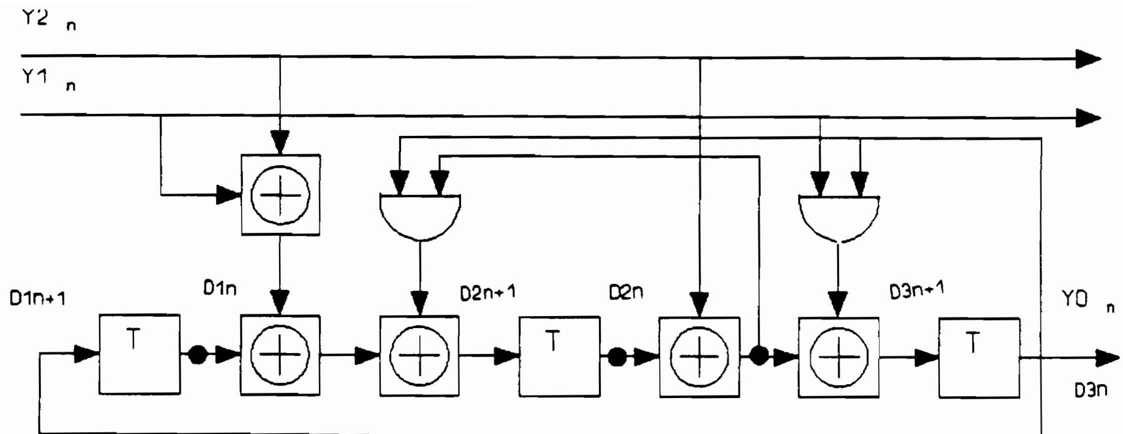


Figure 19: Convolutional Encoder State Analysis Points

traversal. With this in mind, the following analysis was conducted.

Since there are 3 delays and 2 input bits, there are 32 potential input states that cause a given output. This table of potential occurrences can be constructed by calculating the next state and encoder output for each of the 32 potential input instances. Furthermore, since the 2 input bits are directly mapped to the output without change (Refer to Figure 19) only the redundant bit and next state need be calculated. Y_{1n} and Y_{2n} are a direct mapping of the Q_{1n} and Q_{2n} that were input to the encoder. Also, $D3_n = Y0_n$ since $Y0_n$ is immediately following a delay. Therefore, all that has to be characterized is the next state that is traversed to by each

given input instance. To calculate the next state, the point preceding each delay is analyzed since it this state that will next inhabit the current state. Upon studying Figure 19, the following relationships can be derived:

1) $Y0_n = D3_n$

2) $D1_{n+1} = D3_n$

3) $D2_{n+1} = ((D1_n \text{ XOR } (Y1_n \text{ XOR } Y2_n)) \text{ XOR } ((D2_n \text{ XOR } Y2_n) \text{ AND } D3_n))$

4) $D3_{n+1} = ((D2_n \text{ XOR } Y2_n) \text{ XOR } (D3_n \text{ AND } Y1_n))$

Using these relationships, the convolutional encoder can be completely characterized. To investigate all the possibilities, a short 'C' program was written to calculate the next state of each potential input. This program was used to fill in Table 5 which completely characterizes the convolutional encoder used for V.32/V.33 modem transmissions. Also from this table, the trellis diagram for the convolutional encoder can be drawn, and is illustrated in Figure 20.

Figure 20 contains the nucleus of what is necessary to implement the viterbi decoding algorithm to V.32 and V.33 modem transmissions. However, it is necessary to extend these

Table 5: State Table For V.32/V.33 Convolutional Encoder

INPUT INSTANCES					OUTPUT INSTANCES					
$Y1_n$	$Y2_n$	$D1_n$	$D2_n$	$D3_n$	$D1_{n+1}$	$D2_{n+1}$	$D3_{n+1}$	$Y0_n$	$Y1_{n+1}$	$Y2_{n+1}$
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	1	0	0
0	0	0	1	0	0	0	1	0	0	0
0	0	0	1	1	1	1	1	1	0	0
0	0	1	0	0	0	1	0	0	0	0
0	0	1	0	1	1	1	0	1	0	0
0	0	1	1	0	0	1	1	0	0	0
0	0	1	1	1	1	0	1	1	0	0
0	1	0	0	0	0	1	1	0	0	1
0	1	0	0	1	1	0	1	1	0	1
0	1	0	1	0	0	1	0	0	0	1
0	1	0	1	1	1	1	0	1	0	1
0	1	1	0	0	0	0	1	0	0	1
0	1	1	0	1	1	1	1	1	0	1
0	1	1	1	0	0	0	0	0	0	1
0	1	1	1	1	1	0	0	1	0	1
1	0	0	0	0	0	1	0	0	1	0
1	0	0	0	1	1	1	1	1	1	0
1	0	0	1	0	0	1	1	0	1	0
1	0	0	1	1	1	0	0	1	1	0
1	0	1	0	0	0	0	0	0	1	0
1	0	1	0	1	1	0	1	1	1	0
1	0	1	1	0	0	0	1	0	1	0
1	0	1	1	1	1	1	0	1	1	0
1	1	0	0	0	0	0	1	0	1	1
1	1	0	0	1	1	1	0	1	1	1
1	1	0	1	0	0	0	0	0	1	1
1	1	0	1	1	1	0	1	1	1	1
1	1	1	0	0	0	1	1	0	1	1
1	1	1	0	1	1	0	0	1	1	1
1	1	1	1	0	0	1	0	0	1	1
1	1	1	1	1	1	1	1	1	1	1

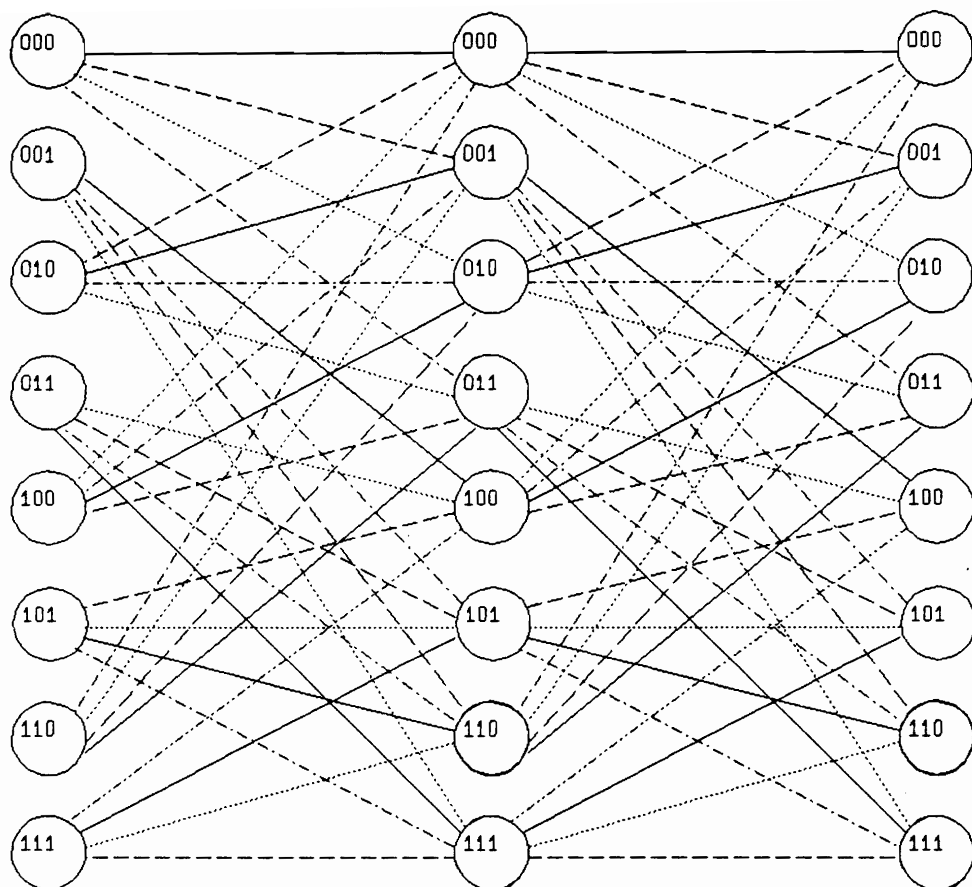


Figure 20: Trellis Diagram For V.32/V.33 Convolutional Encoder

results to obtain the results necessary to properly decode the transmissions. This is because the entire X-bit chunk was not convolutionally encoded, only the first 2 bits in time. Therefore, the trellis has to be expanded to cover all of the instances of the additional bits that were not convolutionallyencoded. This means 4 instances for V.32 since 2 bits were not encoded, and 16 instances for V.33 since 4 bits were not convolutionally encoded.

This extension is easy to visualize since the unencoded bits are the same on the output as they are on the input. To extend to V.32's trellis just stack 4 of the trellises in Figure 20 on top of each other and append one of the 2-bit combinations to the rear of each of the trellises to quadruple the number of states to 32. The states will then be numbered in binary (00000 to 11111, or 0 to 31). The input along the trellis to each other state remains the same. (Just keep in mind that only the left-most 3 bits represent the state, the other 2 bits represent the input which is equivalent to the output since these bits are not encoded.) Therefore the complete extension for V.32 has 32 states with 16 trellis extensions coming out of and going into each state. The left-most three places in the state represent the state type (or class) as it is shown in Figure 20, and the right-most 2 places represent the input and output of that trellis extension. The other 2 bits which are input to the encoder

and the generated redundant bit remains the same for each type, just as they are displayed in Figure 20. The extension to V.33 is similar, just 4 times larger in magnitude. It has 132 states with 64 trellis extensions in and out of each state. The 3 left-most places still represent the state type (or class), but now there are 4 un-encoded bits so there are 16 4-place combinations to be append to the end of each state type for a total of 132 states. The method applied to extending the input and output bits is the same. With the knowledge of this state analysis, it is now possible to discuss the viterbi decoder that was implemented.

3.7.2 The Viterbi Decoder Implementation

The decoder that was implemented is a straight viterbi decoder. The extensions to the trellis diagram in Figure 20 that were discussed in the previous section were used. The explanation to follow will be done generically because it applies to both V.32 and V.33 implementations. There is a separate decoder for each configuration implemented in the program. The discussion below is quite involved, it is recommended that Appendix A be referenced so the actual algorithms implemented in 'C' can be referenced to gain a full understanding of the decoding process.

The viterbi algorithm searches the trellis for the most likely path to each state in the trellis at each frame. It also keeps a path history of what states were traversed to

arrive at that frame along with a cumulative likelihood that that path was traversed. This cumulative likelihood is referred to as the discrepancy of the path. The viterbi decoder waits b frames into the future to decode the symbol from the current frame. If b is chosen large enough, all of the path histories should begin with the same initial state. The path to this state is then the decoded symbol. If all path histories do not begin with the same state, then a demodulation fault has occurred and a guess must be made. If b is chosen sufficiently large, the number of demodulation faults incurred should remain small. It is this length b that is the window length of the decoder, which is referred to as the viterbi memory length in the model which is a user specified parameter.

To implement this decoder, a couple of other considerations had to be made. The first being how to get the last b frames out of viterbi memory once the end of the message is reached. Since there are not b frames to come in the future, they must be forced out of memory. This was done in the model by actually append a string of b states of zero to the message that was transmitted. This way the last real data point is pushed out of the viterbi decoder and only the dummy zero states remain once the decoding process is complete.

The second consideration that needed to be made was with respect to the metric that was to be used to compare potential

state traversals to determine which was the most likely. The metric that was implemented was euclidian distance from the output of the current state to the next state. Euclidian distance was chosen because it is a simple calculation when Cartesian coordinates are used.

The actual decoder implemented works in the following manner. It reads the current symbol, which consists of an X-Y coordinate pair, and iteratively calculates the most likely path it could have traversed to arrive at each state. To further explain this iterative process, one iteration will be broken out further.

For every possible state there are X (16 for V.32, 64 for V.33) possible ways to get there. The most likely one is the one in which the euclidian distance between the output and the state plus the cumulative discrepancy from that state is the smallest. This is implemented in the model by using multiple arrays to keep track of the cumulative discrepancy of each state and a look-up table to determine where each state came from so the proper cumulative discrepancy is applied. The look-up tables that were implemented are actually separate functions in the model. Whenever it is necessary to know what state a potential symbol arriving at a given state came from, a call is made to this function which passes the potential state and destination state and is returned a pointer to the state it came from.

Once all the most likely paths to each state have been calculated for the current symbol (X-Y coordinate pair), the total discrepancy arrays are updated to reflect the new values and control is passed to the next phase of the decoding of that frame. The next phase updates the path histories for all the states just traversed to and pops the last symbol in viterbi memory out since it will be lost when the next iteration is completed. If the last symbol in each path is the same symbol then that is the decoded symbol and it is written to the output file binary_o.txt in a binary format. If all the path histories do not match then a demodulation fault has occurred and a guess needs to be made upon the identity of the symbol. The current model merely takes the last symbol in the first path as the decoded symbol when a demodulation fault occurs. Care is also taken to ensure that the initial viterbi states are not output to the file. This is taken care of by also checking to make sure that b (Viterbi Memory Length) iterations have passed when comparing path histories before the first symbol is written to the output file.

The above process is completed for each symbol that is received. Once all of the symbols have been decoded and written to the binary_o.txt file, all of the decoding for the convolutional encoder is complete. However, the decoding that is necessary to undo the encoding of the differential encoder

must still be accomplished. This decoding is explained in the section below.

3.7.3 Decoding The Differential Encoding

The decoding of a differential code is much simpler than that of a convolutional code. All that is necessary is a look-up table that does the inverse of the look-up that was used to encode the data. Such a look-up table is illustrated in Table 6. Upon comparing Table 6 to Table 4, one will notice that the tables form inverse processes.

The differential encoding process was done using a separate function that served as a look-up table. This function is passed the 2 bits to encode, and uses these 2 bits along with the last output of the differential encoder (which is a global variable) to map what the current output should be.

The differential decoding process is accomplished by calling another function that accomplishes the inverse. This function is passed the 2 bits to decode, and uses these 2 bits along with the 2 bits that were previously put into the decoder (which is a global variable) to map what the decoded 2 bit word should be.

The differential decoding process described is simple enough to where a separate decoding algorithm is not necessary. For this reason, the differential decoder was incorporated with the convolutional decoder for the model implementation. Immediately before a convolutionally decoded symbol is written

Table 6: Differential Decoder Look-Up Table

Bits In	Previous Bits In	Bits Out
00	00	00
00	01	01
00	10	11
00	11	10
01	00	01
01	01	00
01	10	10
01	11	11
10	00	10
10	01	11
10	10	00
10	11	01
11	00	11
11	01	10
11	10	01
11	11	00

to the output file, the differential decoder is called so the completely decoded symbol is written to the output file.

3.8 Binary To Text Converter

The binary to text converter reads the binary_o.txt file and creates the ASCII text equivalent of that file. It grabs 8 bits at a time and converts them into their ASCII text equivalent.

This is implemented by using the text conversion features that are imbedded in 'C'. The ASCII equivalent for a given 8 bit stream is obtained by calculating the decimal equivalent of the 8 bit binary string and printing it out as a character. 'C' handles this by printing out the character that is

referenced by that decimal, not the decimal itself. Since the character is what this function is after, 'C' has done much of the work for us.

4. Example Analyses

4.0 V.32 and V.33 Analysis

Many analyses could be conducted using the model in its current form. The user specified switches allow for many transmission parameters to be isolated and studied. This particular analysis will focus on the transmission rates achievable by each scheme in the presence of gaussian noise, the amount of noise the signal constellations can tolerate, and the decoding window size necessary to limit decoding errors due to demodulation faults.

To facilitate this analysis, multiple runs of the model were made varying the input parameters to achieve an ample data spread. Each model run produced an output file. Examples of the data files produced by the model are displayed below. The results of all of the runs included in this analysis were compiled using a spread sheet. The pertinent results from each run were stripped from the reports and put into a spead sheet format for ease of manipulation and display.

4.1 Transmission In The Presence Of Noise

To study the effects that channel noise has on the modem performance, the actual signal constellation perturbations that are caused by noise were studied. Advantage was taken of the control the model allows for the amount of deviation from the transmitted constellation point that the received constellation point is due to the gaussian noise. Multiple

```
V32 MODEM TRANSMISSION STATISTICAL REPORT
FOR: c:\modem\run_02.txt
-----------------------------------------

The Viterbi Memory Length Used: 30
The Noise Reduction Factor Used: 0.80
Mean Error From X Coordinate: 0.956
Mean Error From Y Coordinate: 1.018
Number Of Symbols Transmitted: 1348
Number Of Bits Transmitted: 5392
Number Of Symbols Received In Error: 289
Number Of Bits Received In Error: 588
Data Rate Achieved 7.542
Number Of ASCII Characters Transmitted: 674
Number Of ASCII Characters In Error: 181
Number Of Demodulation Faults: 5
```

Figure 21: An Example V.32 Model Report File

```
V33 MODEM TRANSMISSION STATISTICAL REPORT
FOR: c:\modem\run_02.txt
-----------------------------------------

The Viterbi Memory Length Used: 30
The Noise Reduction Factor Used: 2.75
Mean Error From X Coordinate: 0.297
Mean Error From Y Coordinate: 0.298
Number Of Symbols Transmitted: 898
Number Of Bits Transmitted: 5388
Number Of Symbols Received In Error: 37
Number Of Bits Received In Error: 87
Data Rate Achieved 13.807
Number Of ASCII Characters Transmitted: 675
Number Of ASCII Characters In Error: 44
Number Of Demodulation Faults: 4
```

Figure 22: An Example V.33 Model Report File

runs of the model were made with the Noise Reduction Factor varying from .6 to 4.0 and the viterbi memory length varying from 10 to 50 for both the V.32 and V.33 modulation schemes.

To allow a direct comparison between V.32 and V.33 to be

made, the signal constellations needed to be normalized to a nominal signal power. For the purpose of the study, a +/- 9v power was implemented. To normalize the signal constellations, the V.32 constellation points were multiplied by 9/4. This caused the 32-point V.32 constellation to be much more spread out than the 128-point V.33 constellation. Figures 8 shows the V.32 signal constellation on a +/- 4v scale. Comparing this constellation to the +/- 9v V.33 constellation illustrated in Figure 9, one can visualize the spreading that occurs when the V.32 constellation is normalized to +/- 9v. Instead of the 1v signal power separating signal points as in V.33, there are (9/4) volts separating constellation points.

4.1.1 Signal Constellation Noise Tolerance

The data generated by the model for constellation point error is displayed graphically in Figures 23 and 24. The general shape of each curve indicates the declining performance that is expected when the constellation point perturbation increases.

It is also instructional to study the knee of each the V.32 and V.33 curves. To the left of the knee, the data rate is only effected slightly by rising constellation point error, but to the right of the knee, the data drops off quickly when the constellation point error continues to rise linearly. In both cases, the knee of the curve corresponds to approximately

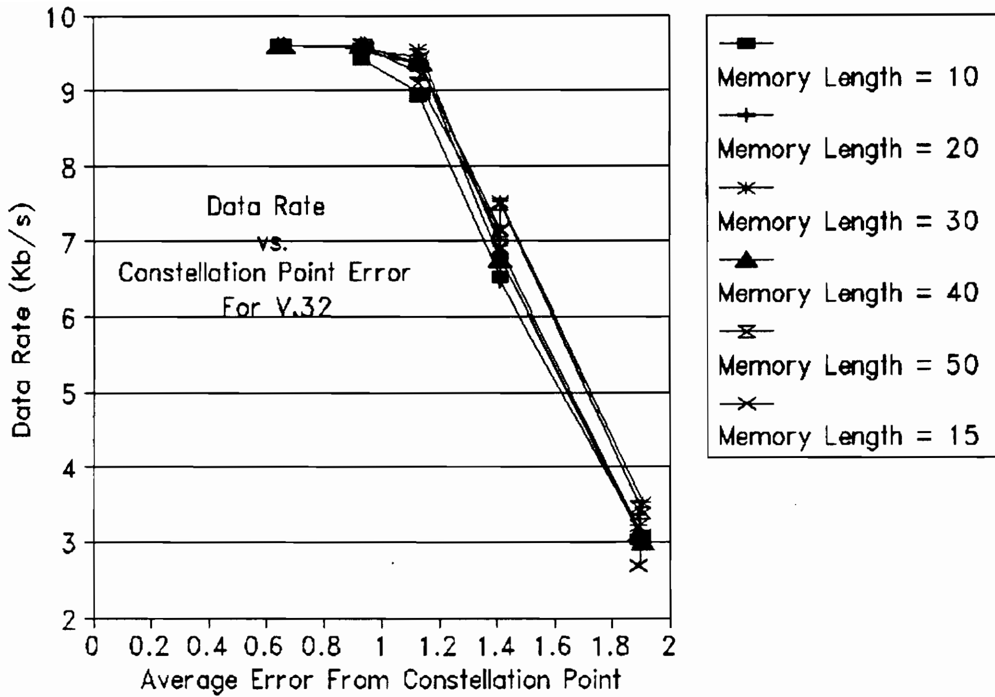


Figure 23: Data Rate vs. Constellation Point Error For V.32

half the distance between adjacent signal points. (.5 for V.33 and 9/8 for V.32) This makes sense since the received signal point is actually closer to an incorrect signal point than it is to the one actually transmitted. Once this occurs, it is much more difficult for the decoder to determine the correct sequence. This also makes the decoder display errors that are not correlated with demodulation faults. The decoder reaches

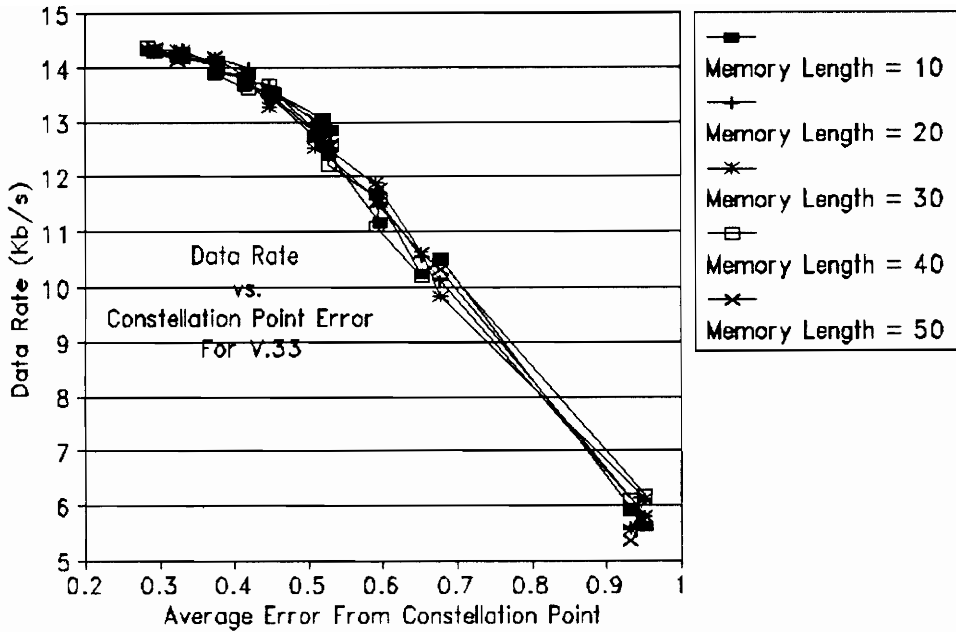


Figure 24: Data Rate vs. Constellation Point Error For V.33

a satisfactory decision that is incorrect due to the high level of noise present.

It is also pertinent to note that the current noise generating function has the potential to inject an occasional large error on a given constellation point due to its pure gaussian nature. A point far out on one of the tails of the normal curve will occasionally occur even though the chances of its occurrence are slim. To further study this effect,

recommendations are given in section 5.1.1.

4.1.2 V.32 vs. V.33 in Gaussian Noise

The curves illustrated in Figure 25 and 26 show the effect that the change of Noise Reduction Factor has on the achieved data rates of the respective schemes. The same relationship observed in Figures 23 and 24 is noticed, but in the opposite direction due to the inverse relationship of the Noise Reduction Factor and constellation point error.

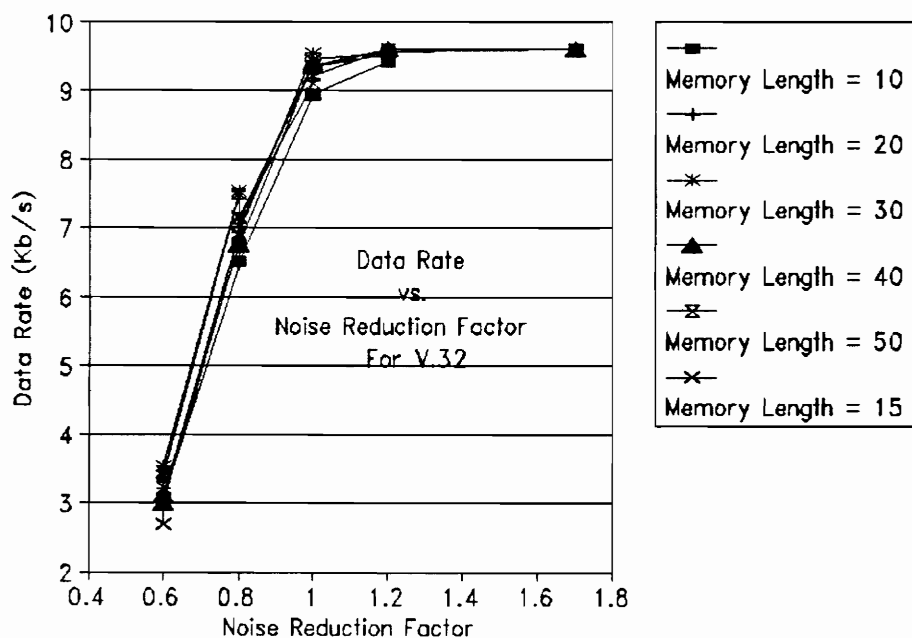


Figure 25: Noise Reduction Factor vs. Data Rate For V.32

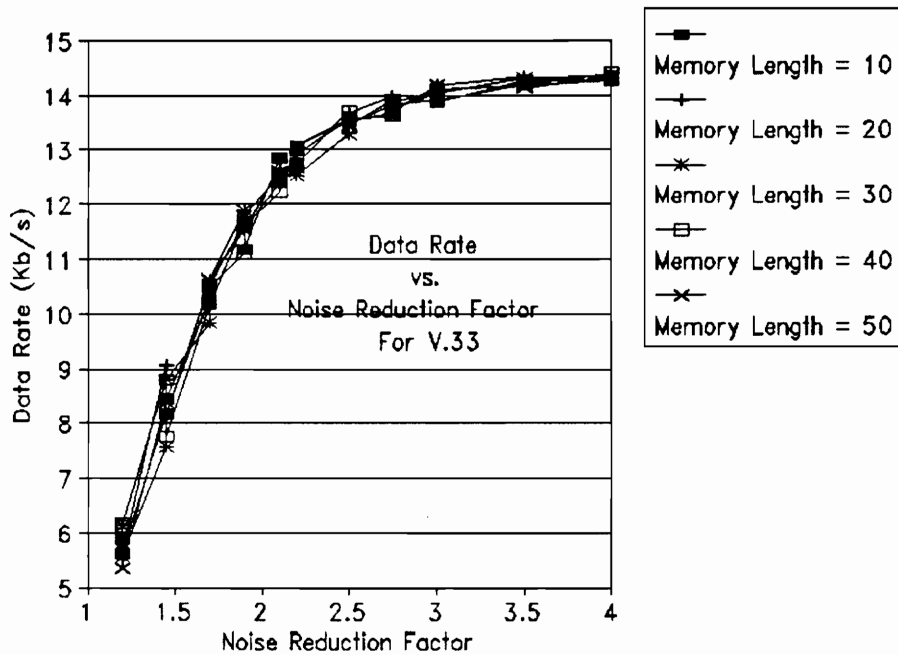


Figure 26: Noise Reduction Factor vs. Data Rate For V.33

Merging the V.32 and V.33 data onto one plot reveals the relationship the two schemes have in reference to noise tolerance. This is illustrated in Figure 27. The two plots intersect at a data rate of 9.6 Kbit/s and a Noise Reduction Factor of 1.6. This implies that the noise level that causes a 33% degradation in performance for V.33 does not effect V.32 transmissions at all. This noise level causes the effective data rate of V.33 to drop from 14.4 Kbits/s to 9.6 Kbits/s and

leaves the V.32 at its peak rate of 9.6 Kbits/s.

A Noise Reduction Factor of 1.6 corresponds to an average signal point error of .7 as is illustrated by the Noise Reduction Factor vs. Average Signal Point Error plot in Figure 28. Up until this point V.33 achieves a higher data than V.32. This allows us to specify that V.33 is a preferable scheme to V.32 when the average signal point error is less than .7 under certain pretenses. Those pretenses are that the reply for retransmit time is negligible and one symbol block lengths are used. This result would change if multiple symbol block lengths were used or if the time to reply a retransmit is not negligible. Higher block lengths would cause a higher number of V.33 blocks to be in error, hence cause more retransmissions and lower the effective data rate. If the request retransmit time is not negligible the effective V.33 data rate would have to be scaled accordingly. This would also further reduce V.33's noise tolerance. The issues involved in studying multi-symbol block lengths and ARQ protocols are addressed in the Potential Enhancements section, particularly in section 5.4.

The results that were obtained go along well with what would be expected. The V.32 scheme is much more tolerant of noise because of the more spread out signal constellation. The signal constellation compression of V.33 makes it much more susceptible to noise just as was discussed in the introduction

section of the report. For the assumptions made, the intersection point was .7, however, no matter what assumptions, block lengths, or other parameters are imposed, there will always be such a crossover point. The V.33 scheme will always be better than the V.32 scheme up to specific noise level.

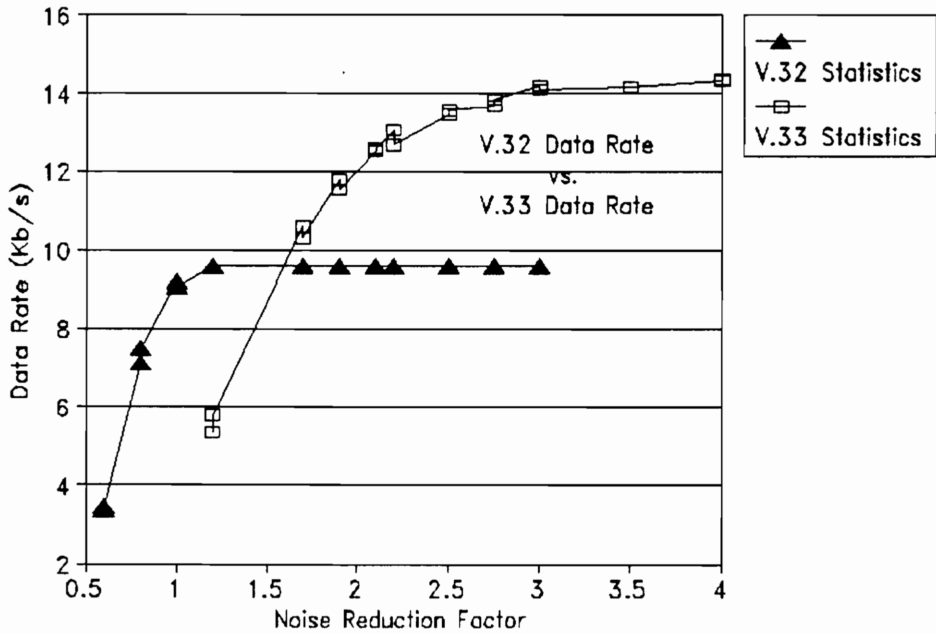


Figure 27: V.32 and V.33 Noise To Data Rate Plots

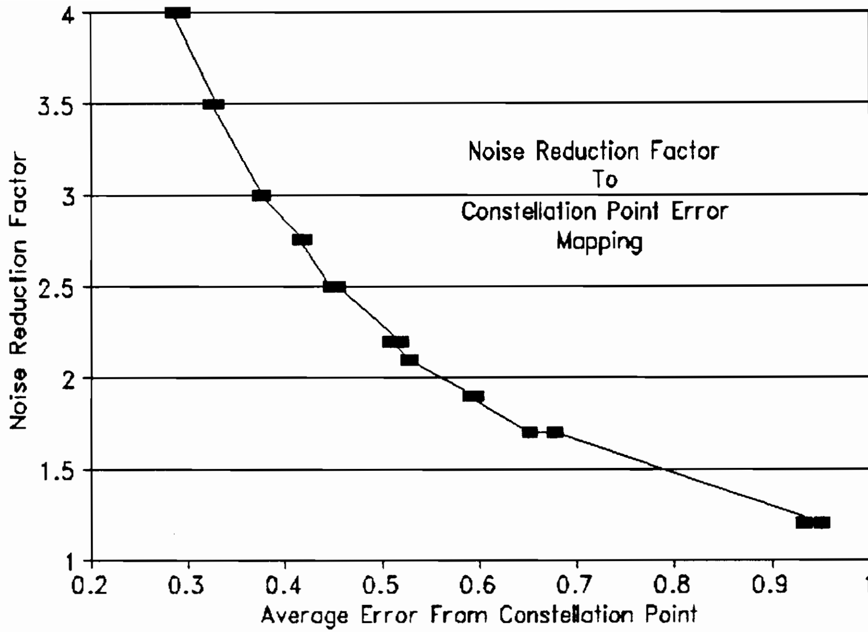


Figure 28: Noise Reduction Factor vs. Average Constellation Point Error

4.2 Viterbi Window Length

The ideal viterbi window length would be infinite, but since this is obviously not practical, one of a manageable finite length is necessary for implementation. The choice of memory length is dependent upon the decoding scheme used, particularly the number of bits per symbol which translates to the number of states that exist in the signal constellation. Even though the delay for an optimum solution

is unbounded, little degradation occurs when the algorithm chooses a fixed window length width that is sufficiently large.

The model allows the window length to be studied as a function of the number of demodulation faults it causes. With this model as a tool, the fixed length necessary for implementation can be studied. The effect that the window length has on the number of demodulation faults is illustrated for both V.32 and V.33 in Figures 29 and 30. Different plots are included for different noise reduction factors to show how the window length's size improves the efficiency of noisy channels.

The plots reveal that the choice of memory length is particularly important for noisy channels. The percentage of demodulation faults drops dramatically for noisy channels when the memory length is raised. The effect is also present for less noisy transmissions, but not nearly as dramatic. It is also instructive to note that increasing the memory length always improves the performance, but to a lesser degree with larger window lengths. The law of diminishing returns is definitely applicable when contemplating the amount of memory to implement. The plots show this effect by approaching the viterbi memory length axis asymptotically.

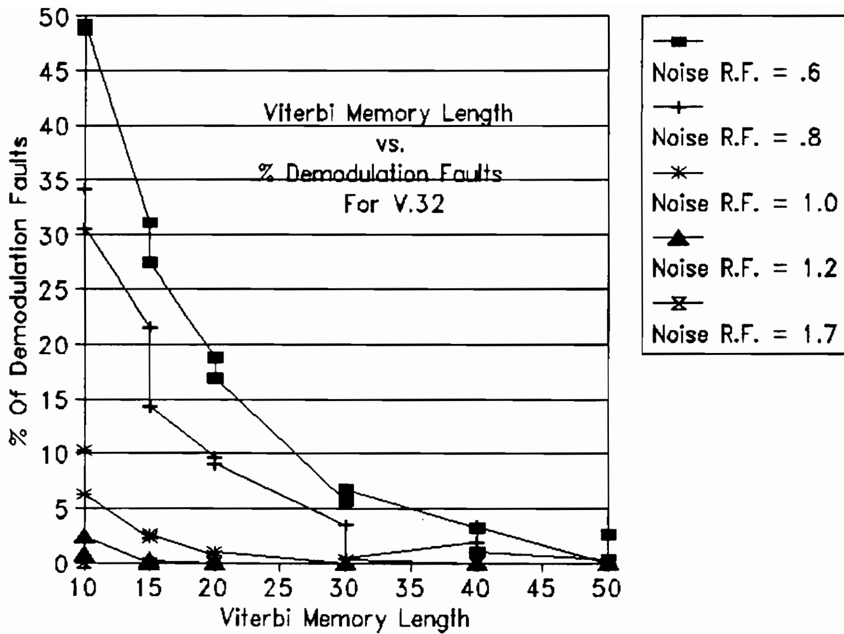


Figure 29: Viterbi Window Length vs. Demodulation Faults, V.32

Comparably, the V.32 scheme requires a smaller window length than does the V.33 scheme to achieve the same percentage of demodulation faults. This is as would be expected since there are 2 additional bits per symbol and 4 times as many states for V.33. V.32 operates within acceptable limits with window lengths of 30-35 and V.33 gets comparable performance with

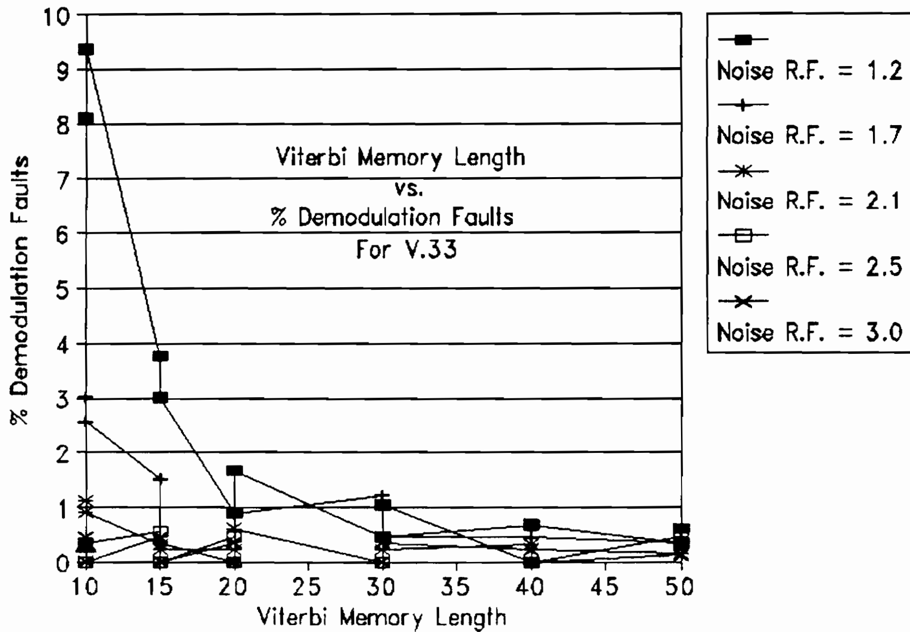


Figure 30: Viterbi Window Length vs. Demodulation Faults, V.33

window lengths of 40-45. Both schemes only gain a very small amount by extending the windows past these lengths.

It is important to keep in mind that demodulation faults do not directly translate to decoding errors. More times than not, demodulation faults result in correctly decoded symbols. Also, no demodulation fault does not necessarily mean that the symbol was decoded properly. Decoding errors are still possible. The lack of a demodulation fault only means that a unanimous decision was made, not necessarily a correct one.

As is evident by the plots, many incorrect decisions were made for transmissions over noisy channels even though demodulation faults were not detected. To further study these phenomena, refer to section 5 where model enhancements to identify error cause are discussed.

5. Model Enhancements

5.0 Potential Extensions To The Model

The model's modular design allows it to be easily upgraded to include more robust functionality. Individual modules can be modified and/or added to the model one at time as they are developed without a loss in functionality of current capabilities. This section includes a list of potential modifications that could be made to enhance the model. Each potential enhancement is described from a modeling and functional point of view. This list is by no means an exhaustive one. The limit on the amount of enhancement is only limited by the desires of the future developers.

5.1 Modify Channel Simulation

The current noise simulation algorithm is quite accurate for studying a pure gaussian distributed channel. However, a pure gaussian distribution is not the most accurate way to represent a real voice band channel. A more realistic noise simulation algorithm would finely tune the results of the model and allow it to be beneficial for studies into other areas such as that of burst error and the effects it has on decoding algorithms.

5.1.1 More Robust Noise Control

The noise generation algorithm could be modified so the variance of the distribution could also be controlled. Measures could also be taken to eliminate gaussian samples

that are far out on the tail of the normal curve. Such points represent short spikes (one symbol in length) of high intensity noise that would better be modeled by spikes of longer duration. High intensity spikes do occur in voice grade channels, but are commonly longer in duration that one symbol. A more accurate noise model would allow users to control the variance, clip-off point and burstiness of the errors.

The variance and clip-off point control would allow the user to more tightly specify the noise to be studied. This type of noise generator modification could be made by simply piping more control arguments from the command line to the noise generator. These additional arguments could be used to modify the resultant RV's that perturb the transmitted constellation points. Once the more sophisticated RVs are generated, they would simply be added to the X-Y coordinates of the signal points just like the GRVs.

5.1.2 Burst Error Simulation

Burst errors, which could be caused by lightning among other sources, are of variable length and intensity. Allowing the user to control the existence, intensity and length of burst errors would allow them to study the effects such errors have on the system performance as a whole. Burst error control could be added to the model by piping additional user specified variables to the noise simulation module. One

variable each would be necessary for the duration, periodicity, and intensity of the burst error. Given these variables, a separate controlling function could be incorporated into the model that would disperse burst error into the transmitted constellation.

5.2 Supply Additional Modulation Techniques

The current model only handles the modulation techniques that are specified in the CCITT standards regarding V.32 and V.33 modem transmissions. Providing additional modulation techniques to the model would provide a wider comparison basis to all studies. In particular, the addition of QAM would provide an excellent baseline for TCM benchmark comparisons. QAM is the modulation technique that previous generation modems were based upon, and text book references often compare coding gains with QAM as the baseline.

5.2.1 ADD QAM

The addition of QAM to the list of modulation techniques handled (currently V.32 and V.33) would allow users to compare the performance of redundant codes to that of nonredundant codes. Both schemes that are currently implemented use TCM to produce one redundant bit per codeword that has the effect of signal constellation expansion due to the extra states. The addition of the nonredundant scheme of QAM would allow the benefits of TCM to be fully explored.

QAM could be introduced into the model by adding it as an

option at the command prompt. Control code could be added to accept QAM as a modulation type. (V.32 and V.33 are currently the only legal values) The new section should be made to call a new function that encodes the binary.txt file into a 16-state QAM constellation. (One is provided in CCITT standard V.29) To facilitate this, new mapping algorithms would also have to be put into place. However, these mapping functions would not be nearly as complex as the ones for TCM. A convolutional encoder is not necessary and their are only 16 states in the signal constellation. The decoding would be handled exactly the same.

5.2.2 Add Other Modulation Schemes

Other modulation schemes could also be incorporated into the model. Schemes such as QPSK could be easily added for historical comparisons and TCM hybrids could be developed and modeled for research purposes.

The implementation of adding multiple modulation techniques to a model that already supports multiple schemes would not be complex. Each would be a separately developed set of function calls that is triggered from the main program through user specified command arguments. The decoding of most schemes could implemented using the current implementation of the viterbi algorithm, although other decoding options may be preferable.

5.3 Enhance The Decoder

The current decoder uses a straight implementation of the viterbi decoding algorithm. The minimum discrepancy (using euclidian distance as the metric) to each state is calculated and the subsequent path histories are stored in a queue equal in length to the specified memory length. The decoder could be enhanced to handle longer messages, take advantage of the classes produced by the convolutional encoder, and to work with multi-dimensional encoding schemes. (Such as the one mentioned in Section 1.) These enhancements would allow decoder complexity and cost to be compared to efficiency, and allow new decoding approaches to be studied in the presence of noise.

5.3.1 Handling Larger Transmissions

The statistics on the transmissions would be more precise if they were calculated using longer transmit messages. The decoder limits the current message length because of the growing discrepancies at each state. In order to keep the discrepancies from reaching out of range values, they have to be periodically reduced.

Implementing this minor enhancement would not be complex. Periodically, (once every 500 symbols or so) the minimum total discrepancy for all the paths needs to be reset to zero and subtracted off of all the other discrepancies. This will have no effect on the outcome of any of the decisions, but will

keep the state's discrepancies from growing proportionally with the message length.

5.3.2 Add Class Manipulation

The decoder could be augmented to allow the study of decoding techniques other than straight viterbi. One possible approach is to try and take advantage of the 8 classes of constellation points that the convolutional encoder produces. Upon studying either the V.32 or V.33 signal constellation, one will notice that the first three bits (which is the output of the convolutional encoder) represents one of eight classes and each class is spread proportionally throughout the constellation. There are 16 signal points per class using V.33 and 4 signal points per class using V.32.

One potential method of taking advantage of the class presence would be to modify the decoding approach. The decoder could be modified to only consider the nearest points from each class as the potential signal point. These 8 points only would then be used as next state paths in the viterbi algorithm. This would have the effect of pruning a large part of the trellis before viterbi decoding even begins. If implemented this would drastically reduce the number of calculations necessary for decoding each symbol. It would reduce the V.32 implementation by approximately 75% and the V.33 implementation by approximately 93%. This calculation savings is reflected by the 32 states currently calculated per

symbol for V.32 and the 128 states currently calculated per symbol for V.33. It would be interesting to see the cost in performance paid for this calculation savings. Once implemented, the model would be capable of bench marking the decoding approaches.

Implementing such a decoder would involve adding a separate module that could be optionally called from within the Binary Reader function. An additional command argument would also have to be added to allow the user to specify the decoding method to be used. The additional module would be very similar to the current decoding functions, but would have additional lines to find the closest point in each class for each point in the transmit signal constellation. The possible paths to these eight states would be calculated and stored in the same manner that the entire trellis is for the straight Viterbi implementation.

5.3.3 Add Multi-Dimensional Options

Multi-dimensional encoders and decoders, such as the eight-dimensional scheme developed by Codex Corp., could be implemented for study by developing a new Binary Reader function that could be optionally called by the user.

The implementation of such a scheme would be fairly involved. A substantial amount of work would have to be done to define the modulation scheme, reduce to it recursive algorithms and and write the necessary code. The encoding

scheme would likely be more complex than that of V.32 or V.33 and is not currently available in a standard format. There is no limit on the number of multi-dimensional schemes that could be added to the model.

5.4 Add Variable Block Sizes

The current model only studies blocks of length equal to one symbol. Different data rates would be achieved by using different size transmit blocks because the probability of block error goes up proportionally with the size of the block. Allowing the user to control the block sizes would open another degree of freedom to the analyses that could be conducted using the model. Studies would be capable of investigating the effects of block length on the modems functional and output performance.

Implementing controllable block length would involve modifying the statistics and reporting features of the model in addition to adding another command argument for user input. It is not the actual encoding and decoding algorithms that would have to be modified, but the way the statistics are calculated. Instead of calculating achieved data rate as a percentage of error free symbols to symbols transmitted, it would be calculated as a percentage of error free blocks to blocks transmitted using the user specified block length. It is easy to see that the larger block lengths would be more impacted by substantial noise rates than smaller block

lengths, assuming that a block error occurs when any bit in the block is in error. To fully study block length effects on modem efficiency, Acknowledgement Requested (ARQ) protocol simulation may also be necessary depending upon the configuration and accuracy desired.

5.5 Add Redundant Modeling Shell

The current implementation of the model only allows for one run of the model to be specified at a time. More results could be obtained if the model were capable of running multiple scenarios in batch without intermediate user interaction. Since the command arguments control the modem model's specifications, this would allow the user to obtain results from many different model scenarios with only one interaction with the model. This increase in data output would make the model's ability to analyze multiple scenarios in a timely manner much greater.

There are many possible approaches to implementing such a shell. The most straight forward one would be to use the operating system to create an interface shell for the user. This could be done using MS-DOS or UNIX.(The executable code can be run on either platform.) A Commercial-Off-The-Shelf (COTS) text editor could be integrated with custom shell script to allow the user to enter an infinite number of scenarios. This shell could also be made to allow the user to run a given scenario X (user specified) times in order to

gain more precise results. Also, since multiple scenarios could take many hours to run, an escape sequence should also be implemented to allow the user to break out of a simulation. This could be implemented in the shell script using the standard OS break sequences.

5.6 Add Differential Decoding Statistics

The current system uses CCITT standard differential encoding and decoding, but no statistics are calculated separating the errors caused by noise from the error extensions to those caused by noise due to the differential decoder. In cases where the differentially encoded bits are convolutionally decoded in error, the differential decoder extends the error due to its cyclical nature (caused by the feedback). The decoder currently implemented could proliferate any where from 1 to 4 extra errors before correcting itself. The size of the error extension is dependent upon the bit(s) in error and the timing of the error. Each extension length of the error is equiprobable. Allowing the user to determine which errors were caused by the differential decoder would enable them to study its impact on performance and seek alternative solutions, or at least the optimal scenario. For a more in-depth discussion of differential decoding, reference the bibliography and other communications theory sources.

To enhance the model to allow the user to differentiate errors, more rigorous error monitoring and calculating will have to be implemented. This could be done by adding a section to the reporting module that checks for decode errors in the differentially encoded bits ($Y1_n$ and $Y2_n$) and calculates the number of errors immediately proceeding that were caused. In order to ascertain which adjacent errors were caused by the differential decoder, additional look-up tables would have to be generated. To populate these new tables properly, an in-depth understanding of differential encoding patterns would have to be gained. This understanding could be gotten from writing a small program that exhaustively searches for the possible combinations.

Also, it is possible to study a model of a modem without differential encoding/decoding by simply piping $Q1_n$ and $Q2_n$ directly to $Y1_n$ and $Y2_n$ bypassing the Diffencoder function. This would allow comparisons to be made on the price paid to remain synchronous. Another user specified switch is all that would be necessary to allow turning on and off the differential encoder/decoder.

BIBLIOGRAPHY

Blahut, Richard E. Digital Transmission of Information. Addison-Wesley Publishing Company, Reading Massachusetts, 1990.

CCITT Recommendation V.32: A Family of 2-wire, Duplex Modems Operating at Data Signalling Rates of Up to 9600 bps for Use on the General Switched Telephone Network and on Leased Telephone-Type Circuits. Malaga-Torremolinos, 1984, ammended at Melboure, 1988.

CCITT Recommendation V.33: 14.400 kbps Modem Standardized for Use on Point-to-Point 4-Wire Leased Telephone-Type Circuits. Melbourne, 1988.

CCITT Study Group XVII & Working Parties Trellis Precoding: Combined Coding, Precoding and Shaping for Intersymbol Interference Channels. Geneva, Special Rapporteur on V.32 Enhancements (R.L. Stuart, Penril DataComm) 15-23 October, 1990.

CCITT Study Group XVII & Working Parties Trellis Shaping. Geneva, Special Rapporteur on V.32 Enhancements (R.L. Stuart, Penril DataComm) 15-23 October, 1990.

Douglass, Jack and Hayes, Jack The ABCs of Trellis Coding. Telephony, December 21, 1987.

Glass, Brett L. Modern Modem Methods. BYTE, June 1989.

Keiser, Bernard E. Broadband Coding, Modulation, and Transmission Engineering. Prentice Hall, Englewood Cliffs, N.J. 07632.

Payton, John and Qureshi, Shahid Codex Corporation Trellis Encoding: What It Is and How It Affects Data Transmission. Data Communications, May 1985.

Tang, Wilson H. Probability Concepts in Engineering Planning and Design. John Wiley & Sons, Inc. New York, 1975.

von Taube, Eugene The Frontier of High-Speed Modulation. Telecommunications, March 1986.

APPENDIX

```
/*   Program Name: Modem.c                                      */
/*                                                              */
/*   Developed By: Joseph J. Pisula                             */
/*                                                              */
/*      Purpose:  To read ascii text files and simulate */
/*                V.32 and V.33 modem transmissions.    */
/*                The overall purpose is to allow the   */
/*                study of noise, decoder, and TCM      */
/*                techniques and the effect each has on */
/*                transmission efficiency.              */
/*                                                      */
/*   Global Variables:                                  */
/*      y0old = y0(n-1),y1old = y1(n-1),y2old = y2(n-1),*/
/*   y0oldest = y0(n-2), xnew = x1(n), xold = x1(n-1),  */
/*      y0new = (redundant bit, encoded y0),            */
/*      y1new = (diffencoded bit q1, encoded y1),       */
/*      y2new = (diffencoded bit q2, encoded y2),       */
/*   y3new = (bit 3 of X-bit chunk),                    */
/*   y4new = (bit 4 of X-bit chunk),                    */
/*   y5new = (bit 5 of X-bit chunk,V.33 14.4 rate only),*/
/*   y6new = (bit 6 of X-bit chunk,V.33 14.4 rate only) */
/*                                                      */
/*                                                      */

int y0old = 0, y1old = 0, y2old = 0, y0oldest = 0,d1 = 0,
d2 = 0,d3 = 0;
int xnew, xold, y1new, y2new, y0new, y3new, y4new, y5new,
y6new;
float x, y;
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
/* FUNCTION MAIN(ARGC,AGRV)    */
main(argc,argv)
int argc;
char *argv[];
{int vit_mem_length;
float noise;

if (argc < 7)
{
printf("\nExample Usage:C>Data_File Modem Enc_Type(1,2)
Vit_Mem_Lgth Noise_R_F Rpt_P_Name\n");
     exit(1);
}
noise = atof(argv[5]);
vit_mem_length = atoi(argv[4]);
/* THIS CONVERTS THE FILE FROM ASCII TO BINARY          */
```

```
textconverter(argv[1]);
/* THIS CHECKS TO SEE WHAT CCITT STANDARD WAS CHOSEN    */
/* AND KICKS OFF THE APPROPRIATE ENCODING\DECODING      */
if (strcmp(argv[2],"v32")==0 ¦¦ strcmp(argv[2],"V32")==0)

binaryreader32((int)vit_mem_length,argv[3],(float)noise,argv
[1],argv[6]);
else
     if (strcmp(argv[2],"v33")==0 ¦¦
strcmp(argv[2],"V33")==0)

binaryreader33((int)vit_mem_length,argv[3],(float)noise,
     argv[1],argv[6]);
     else
     {    printf("\n**Invalid Modem Type (Argument 2)**\n");
          exit(1);
     }
/* ONCE ALL THE ENCODING/DECODING AND REPORT WRITING IS
*/
/* COMPLETED, CONVERT BINARY BACK TO TEXT
*/
totext();
return 0;
}
/*textconverter*/
/*   This function converts ascii text to binary for modem
*/ /*     transmission
/* FUNCTION TEXTCONVERTER(CHAR PATHFILE[25]) */
textconverter(char pathfile[25])
{
FILE *fptr,*fptr2;
int output,mask,j,bit;
mask = 0x80;
/*   THE FILE TO TRANSMIT IS SPECIFIED BY THE USER, THE
*/
/*   BINARY FILE WHICH IS OPERATED AGAINST IS ALWAYS CALLED
*/
/*   BINARY.TXT
*/
fptr = fopen(pathfile,"r");
fptr2 = fopen("c:\\modem\\binary.txt","w");
while( (output=getc(fptr)) != EOF)
{
     mask = 0x80;
     for (j=0; j<8; j++)
          {
          bit = (mask & output) ? 1 : 0;
          fprintf(fptr2,"%d",bit);
          mask >>= 1;
```

```
            }
}
fclose(fptr);
fclose(fptr2);
return 0;
}
```

```
/*   binaryreader33 */
/*FUNCTION
BINARYREADER33(VIT_MEM_LENGTH,ENCODER_TYPE,NOISE,TFILE,RFILE
)*/
/*   THIS IS THE FUNCTION THAT CONTROLS V.33 TRANSMISSIONS
*/
/*   VIT_MEM_LENGTH = VITERBI_MEMORY_LENGTH
*/
/*   ENCODER_TYPE = (1 OR 2) TO SPECIFY DYNAMIC OR LOOK-
UP   */
/*   NOISE = NOISE_REDUCTION_FACTOR USED
*/
/*   TFILE = PATH/NAME OF TEXT FILE TO OPERATE AGAINST
*/
/*   RFILE = PATH/NAME OF REPORT FILE TO GENERATE
*/
/*   MY_XY = STRUCTURE OF TRANSMITTED SIGNAL POINTS
*/
/*   PATH_OLD = AN ARRAY OF STRUCTURES TO TRACK PATH
HISYORY*/
/*   PATH_NEW = AN ARRAY OF STRUCTURES TO TRACK PATH
HISTORY*/
/*   V33_TABLE = AN ARRAY OF STRUCTURES FOR S/P LOOK-UP
*/
/*   FROM_NODEM = AN ARRAY THAT TRACKS WHAT STATE THAT THE
*/
/*                   CURRENT STATE CAME FROM
*/
/*   DECODED = ARRAY THAT TRACKS CURRENT SYMBOL FOR
DECODING*/
/*   BEST_PATH = AN ARRAY TO STORE THE NEXT NODE IN PATH
*/
/*   TO_D_O = AN ARRAY TO HOLD TOTAL DISCREP FOR LAST STATE
*/
/*   TO_D_N = AN ARRAY TO HOLD TOTAL DISCREP FOR NEXT STATE
*/
/*   MIN_DISCREP = MINIMUM DISCREP TO LAST STATE
*/
/*   OUTPUT =THE STRING USED TO TAKE SYMBOLS FROM
BINARY.TXT*/
binaryreader33(vit_mem_length,encoder_type,noise,tfile,rfile
)
float noise;
char encoder_type,tfile[25],rfile[25];
int vit_mem_length;
{
struct my_xy {
 float xval;
 float yval;
```

```
} xy[3000];
struct path1 {
int path_o[50];
} path_old[128];
struct path2 {
int path_c[50];
} path_new[128];
struct v33 {
float xcoord;
float ycoord;
int bit_1;
int bit_2;
int bit_3;
int bit_4;
int bit_5;
int bit_6;
int bit_7;
int index;
} v33_table[128];
int  q1, q2,dec1,dec2;
int ct = 0, eoffnd = 0;
int j,bit,iteration_count=0,zero_pad;
int from_node[128];
int i,l,poss_symbol;
int state_to,current,state_from_index,demod_fault=0;
int best_path[128];
int decoded[128];
int numxy = 0,check=1;
int maxxy = 3000;
float tot_d_o[128];
float tot_d_n[128];
float discrep;
float min_discrep[128];
float u1,u2,r,a,grv1,grv2,rvtot1=0,rvtot2=0;
float xv,yv;
char output[6];
FILE *fptr,*fptr8;
fptr = fopen("c:\\modem\\binary.txt","r");
fptr8 = fopen("c:\\modem\\binary_o.txt","w");
randomize();
/* INITIALIZE ALL NECESSARY TO 0                              */
for (i=0; i<128; i++)
tot_d_o[i] = 0.0;
for (i=0; i<128; i++)
tot_d_n[i] = 0.0;
for (i=0; i<128; i++)
min_discrep[i] = 30000;
for (i=0; i<128; i++)
for (l=0; l<vit_mem_length; l++)
```

```
path_old[i].path_o[l] = 0;
for (i=0; i<128; i++)
for (l=0; l<vit_mem_length; l++)
path_new[i].path_c[l] = 0;
/* LOAD THE V.33 LOOK-UP TABLE TO MEMORY FOR EASY ACCESS
*/
/* V33 Look-Up Table */
v33_table[0].xcoord = -8;
v33_table[0].ycoord = -3;
v33_table[0].bit_1 = 0;
v33_table[0].bit_2 = 0;
v33_table[0].bit_3 = 0;
v33_table[0].bit_4 = 0;
v33_table[0].bit_5 = 0;
v33_table[0].bit_6 = 0;
v33_table[0].bit_7 = 0;
v33_table[0].index = 0;

v33_table[1].xcoord = 9;
v33_table[1].ycoord = 2;
v33_table[1].bit_1 =  0;
v33_table[1].bit_2 =  0;
v33_table[1].bit_3 =  0;
v33_table[1].bit_4 =  0;
v33_table[1].bit_5 =  0;
v33_table[1].bit_6 = 0;
v33_table[1].bit_7 = 1;
v33_table[1].index =  1;

v33_table[2].xcoord = 2;
v33_table[2].ycoord =-9;
v33_table[2].bit_1 = 0;
v33_table[2].bit_2 = 0;
v33_table[2].bit_3 = 0;
v33_table[2].bit_4 = 0;
v33_table[2].bit_5 = 0;
v33_table[2].bit_6 = 1;
v33_table[2].bit_7 = 0;
v33_table[2].index = 2;

v33_table[3].xcoord =-3;
v33_table[3].ycoord = 8;
v33_table[3].bit_1 = 0;
v33_table[3].bit_2 = 0;
v33_table[3].bit_3 = 0;
v33_table[3].bit_4 = 0;
v33_table[3].bit_5 = 0;
v33_table[3].bit_6 = 1;
v33_table[3].bit_7 = 1;
```

```
v33_table[3].index = 3;

v33_table[4].xcoord = 8;
v33_table[4].ycoord = 3;
v33_table[4].bit_1 = 0;
v33_table[4].bit_2 = 0;
v33_table[4].bit_3 = 0;
v33_table[4].bit_4 = 0;
v33_table[4].bit_5 = 1;
v33_table[4].bit_6 = 0;
v33_table[4].bit_7 = 0;
v33_table[4].index = 4;

v33_table[5].xcoord =-9;
v33_table[5].ycoord =-2;
v33_table[5].bit_1 = 0;
v33_table[5].bit_2 = 0;
v33_table[5].bit_3 = 0;
v33_table[5].bit_4 = 0;
v33_table[5].bit_5 = 1;
v33_table[5].bit_6 = 0;
v33_table[5].bit_7 = 1;
v33_table[5].index = 5;

v33_table[6].xcoord =-2;
v33_table[6].ycoord = 9;
v33_table[6].bit_1 =  0;
v33_table[6].bit_2 =  0;
v33_table[6].bit_3 =  0;
v33_table[6].bit_4 =  0;
v33_table[6].bit_5 =  1;
v33_table[6].bit_6 = 1;
v33_table[6].bit_7 = 0;
v33_table[6].index =  6;

v33_table[7].xcoord =  3;
v33_table[7].ycoord = -8;
v33_table[7].bit_1 = 0;
v33_table[7].bit_2 = 0;
v33_table[7].bit_3 = 0;
v33_table[7].bit_4 = 0;
v33_table[7].bit_5 = 1;
v33_table[7].bit_6 = 1;
v33_table[7].bit_7 = 1;
v33_table[7].index = 7;

v33_table[8].xcoord = -8;
v33_table[8].ycoord = 1;
v33_table[8].bit_1 = 0;
```

```
v33_table[8].bit_2 = 0;
v33_table[8].bit_3 = 0;
v33_table[8].bit_4 = 1;
v33_table[8].bit_5 = 0;
v33_table[8].bit_6 = 0;
v33_table[8].bit_7 = 0;
v33_table[8].index = 8;

v33_table[9].xcoord = 9;
v33_table[9].ycoord = -2;
v33_table[9].bit_1 = 0;
v33_table[9].bit_2 = 0;
v33_table[9].bit_3 = 0;
v33_table[9].bit_4 = 1;
v33_table[9].bit_5 = 0;
v33_table[9].bit_6 = 0;
v33_table[9].bit_7 = 1;
v33_table[9].index = 9;

v33_table[10].xcoord =-2;
v33_table[10].ycoord =-9;
v33_table[10].bit_1 = 0;
v33_table[10].bit_2 = 0;
v33_table[10].bit_3 = 0;
v33_table[10].bit_4 = 1;
v33_table[10].bit_5 = 0;
v33_table[10].bit_6 = 1;
v33_table[10].bit_7 = 0;
v33_table[10].index = 10;

v33_table[11].xcoord = 1;
v33_table[11].ycoord = 8;
v33_table[11].bit_1 = 0;
v33_table[11].bit_2 = 0;
v33_table[11].bit_3 = 0;
v33_table[11].bit_4 = 1;
v33_table[11].bit_5 = 0;
v33_table[11].bit_6 = 1;
v33_table[11].bit_7 = 1;
v33_table[11].index = 11;

v33_table[12].xcoord = 8;
v33_table[12].ycoord = -1;
v33_table[12].bit_1 =  0;
v33_table[12].bit_2 =  0;
v33_table[12].bit_3 =  0;
v33_table[12].bit_4 =  1;
v33_table[12].bit_5 =  1;
v33_table[12].bit_6 = 0;
```

```
v33_table[12].bit_7 = 0;
v33_table[12].index =  12;

v33_table[13].xcoord =-9;
v33_table[13].ycoord = 2;
v33_table[13].bit_1 = 0;
v33_table[13].bit_2 = 0;
v33_table[13].bit_3 = 0;
v33_table[13].bit_4 = 1;
v33_table[13].bit_5 = 1;
v33_table[13].bit_6 = 0;
v33_table[13].bit_7 = 1;
v33_table[13].index = 13;

v33_table[14].xcoord = 2;
v33_table[14].ycoord = 9;
v33_table[14].bit_1 = 0;
v33_table[14].bit_2 = 0;
v33_table[14].bit_3 = 0;
v33_table[14].bit_4 = 1;
v33_table[14].bit_5 = 1;
v33_table[14].bit_6 = 1;
v33_table[14].bit_7 = 0;
v33_table[14].index = 14;

v33_table[15].xcoord =  -1;
v33_table[15].ycoord = -8;
v33_table[15].bit_1 = 0;
v33_table[15].bit_2 = 0;
v33_table[15].bit_3 = 0;
v33_table[15].bit_4 = 1;
v33_table[15].bit_5 = 1;
v33_table[15].bit_6 = 1;
v33_table[15].bit_7 = 1;
v33_table[15].index = 15;

v33_table[16].xcoord = -4;
v33_table[16].ycoord = -3;
v33_table[16].bit_1 = 0;
v33_table[16].bit_2 = 0;
v33_table[16].bit_3 = 1;
v33_table[16].bit_4 = 0;
v33_table[16].bit_5 = 0;
v33_table[16].bit_6 = 0;
v33_table[16].bit_7 = 0;
v33_table[16].index = 16;

v33_table[17].xcoord = 5;
v33_table[17].ycoord = 2;
```

```
v33_table[17].bit_1 = 0;
v33_table[17].bit_2 = 0;
v33_table[17].bit_3 = 1;
v33_table[17].bit_4 = 0;
v33_table[17].bit_5 = 0;
v33_table[17].bit_6 = 0;
v33_table[17].bit_7 = 1;
v33_table[17].index = 17;

v33_table[18].xcoord = 2;
v33_table[18].ycoord =-5;
v33_table[18].bit_1 = 0;
v33_table[18].bit_2 = 0;
v33_table[18].bit_3 = 1;
v33_table[18].bit_4 = 0;
v33_table[18].bit_5 = 0;
v33_table[18].bit_6 = 1;
v33_table[18].bit_7 = 0;
v33_table[18].index = 18;

v33_table[19].xcoord =-3;
v33_table[19].ycoord = 4;
v33_table[19].bit_1 = 0;
v33_table[19].bit_2 = 0;
v33_table[19].bit_3 = 1;
v33_table[19].bit_4 = 0;
v33_table[19].bit_5 = 0;
v33_table[19].bit_6 = 1;
v33_table[19].bit_7 = 1;
v33_table[19].index = 19;

v33_table[20].xcoord = 4;
v33_table[20].ycoord = 3;
v33_table[20].bit_1 = 0;
v33_table[20].bit_2 = 0;
v33_table[20].bit_3 = 1;
v33_table[20].bit_4 = 0;
v33_table[20].bit_5 = 1;
v33_table[20].bit_6 = 0;
v33_table[20].bit_7 = 0;
v33_table[20].index = 20;

v33_table[21].xcoord = -5;
v33_table[21].ycoord =-2;
v33_table[21].bit_1 = 0;
v33_table[21].bit_2 = 0;
v33_table[21].bit_3 = 1;
v33_table[21].bit_4 = 0;
v33_table[21].bit_5 = 1;
```

```
v33_table[21].bit_6 = 0;
v33_table[21].bit_7 = 1;
v33_table[21].index = 21;

v33_table[22].xcoord =-2;
v33_table[22].ycoord = 5;
v33_table[22].bit_1 = 0;
v33_table[22].bit_2 = 0;
v33_table[22].bit_3 = 1;
v33_table[22].bit_4 = 0;
v33_table[22].bit_5 = 1;
v33_table[22].bit_6 = 1;
v33_table[22].bit_7 = 0;
v33_table[22].index = 22;

v33_table[23].xcoord = 3;
v33_table[23].ycoord = -4;
v33_table[23].bit_1 = 0;
v33_table[23].bit_2 = 0;
v33_table[23].bit_3 = 1;
v33_table[23].bit_4 = 0;
v33_table[23].bit_5 = 1;
v33_table[23].bit_6 = 1;
v33_table[23].bit_7 = 1;
v33_table[23].index = 23;

v33_table[24].xcoord =-4;
v33_table[24].ycoord = 1;
v33_table[24].bit_1 = 0;
v33_table[24].bit_2 = 0;
v33_table[24].bit_3 = 1;
v33_table[24].bit_4 = 1;
v33_table[24].bit_5 = 0;
v33_table[24].bit_6 = 0;
v33_table[24].bit_7 = 0;
v33_table[24].index = 24;

v33_table[25].xcoord =  5;
v33_table[25].ycoord =-2;
v33_table[25].bit_1 = 0;
v33_table[25].bit_2 = 0;
v33_table[25].bit_3 = 1;
v33_table[25].bit_4 = 1;
v33_table[25].bit_5 = 0;
v33_table[25].bit_6 = 0;
v33_table[25].bit_7 = 1;
v33_table[25].index = 25;

v33_table[26].xcoord =-2;
```

```
v33_table[26].ycoord =-5;
v33_table[26].bit_1 = 0;
v33_table[26].bit_2 = 0;
v33_table[26].bit_3 = 1;
v33_table[26].bit_4 = 1;
v33_table[26].bit_5 = 0;
v33_table[26].bit_6 = 1;
v33_table[26].bit_7 = 0;
v33_table[26].index = 26;

v33_table[27].xcoord = 1;
v33_table[27].ycoord = 4;
v33_table[27].bit_1 = 0;
v33_table[27].bit_2 = 0;
v33_table[27].bit_3 = 1;
v33_table[27].bit_4 = 1;
v33_table[27].bit_5 = 0;
v33_table[27].bit_6 = 1;
v33_table[27].bit_7 = 1;
v33_table[27].index = 27;

v33_table[28].xcoord = 4;
v33_table[28].ycoord = -1;
v33_table[28].bit_1 = 0;
v33_table[28].bit_2 = 0;
v33_table[28].bit_3 = 1;
v33_table[28].bit_4 = 1;
v33_table[28].bit_5 = 1;
v33_table[28].bit_6 = 0;
v33_table[28].bit_7 = 0;
v33_table[28].index = 28;

v33_table[29].xcoord =-5;
v33_table[29].ycoord = 2;
v33_table[29].bit_1 = 0;
v33_table[29].bit_2 = 0;
v33_table[29].bit_3 = 1;
v33_table[29].bit_4 = 1;
v33_table[29].bit_5 = 1;
v33_table[29].bit_6 = 0;
v33_table[29].bit_7 = 1;
v33_table[29].index = 29;

v33_table[30].xcoord = 2;
v33_table[30].ycoord = 5;
v33_table[30].bit_1 = 0;
v33_table[30].bit_2 = 0;
v33_table[30].bit_3 = 1;
v33_table[30].bit_4 = 1;
```

```
v33_table[30].bit_5 = 1;
v33_table[30].bit_6 = 1;
v33_table[30].bit_7 = 0;
v33_table[30].index = 30;

v33_table[31].xcoord = -1;
v33_table[31].ycoord =-4;
v33_table[31].bit_1 = 0;
v33_table[31].bit_2 = 0;
v33_table[31].bit_3 = 1;
v33_table[31].bit_4 = 1;
v33_table[31].bit_5 = 1;
v33_table[31].bit_6 = 1;
v33_table[31].bit_7 = 1;
v33_table[31].index = 31;

v33_table[32].xcoord = 4;
v33_table[32].ycoord = -3;
v33_table[32].bit_1 = 0;
v33_table[32].bit_2 = 1;
v33_table[32].bit_3 = 0;
v33_table[32].bit_4 = 0;
v33_table[32].bit_5 = 0;
v33_table[32].bit_6 = 0;
v33_table[32].bit_7 = 0;
v33_table[32].index =32;

v33_table[33].xcoord =-3;
v33_table[33].ycoord = 2;
v33_table[33].bit_1 =  0;
v33_table[33].bit_2 =  1;
v33_table[33].bit_3 =  0;
v33_table[33].bit_4 =  0;
v33_table[33].bit_5 =  0;
v33_table[33].bit_6 = 0;
v33_table[33].bit_7 = 1;
v33_table[33].index = 33;

v33_table[34].xcoord = 2;
v33_table[34].ycoord = 3;
v33_table[34].bit_1 = 0;
v33_table[34].bit_2 = 1;
v33_table[34].bit_3 = 0;
v33_table[34].bit_4 = 0;
v33_table[34].bit_5 = 0;
v33_table[34].bit_6 = 1;
v33_table[34].bit_7 = 0;
v33_table[34].index =34;
```

```
v33_table[35].xcoord =-3;
v33_table[35].ycoord =-4;
v33_table[35].bit_1 = 0;
v33_table[35].bit_2 = 1;
v33_table[35].bit_3 = 0;
v33_table[35].bit_4 = 0;
v33_table[35].bit_5 = 0;
v33_table[35].bit_6 = 1;
v33_table[35].bit_7 = 1;
v33_table[35].index =35;

v33_table[36].xcoord =-4;
v33_table[36].ycoord = 3;
v33_table[36].bit_1 = 0;
v33_table[36].bit_2 = 1;
v33_table[36].bit_3 = 0;
v33_table[36].bit_4 = 0;
v33_table[36].bit_5 = 1;
v33_table[36].bit_6 = 0;
v33_table[36].bit_7 = 0;
v33_table[36].index =36;

v33_table[37].xcoord = 3;
v33_table[37].ycoord =-2;
v33_table[37].bit_1 = 0;
v33_table[37].bit_2 = 1;
v33_table[37].bit_3 = 0;
v33_table[37].bit_4 = 0;
v33_table[37].bit_5 = 1;
v33_table[37].bit_6 = 0;
v33_table[37].bit_7 = 1;
v33_table[37].index =37;

v33_table[38].xcoord =-2;
v33_table[38].ycoord = -3;
v33_table[38].bit_1 =  0;
v33_table[38].bit_2 =  1;
v33_table[38].bit_3 =  0;
v33_table[38].bit_4 =  0;
v33_table[38].bit_5 =  1;
v33_table[38].bit_6 = 1;
v33_table[38].bit_7 = 0;
v33_table[38].index = 38;

v33_table[39].xcoord = 3;
v33_table[39].ycoord = 4;
v33_table[39].bit_1 = 0;
v33_table[39].bit_2 = 1;
v33_table[39].bit_3 = 0;
```

```
v33_table[39].bit_4 = 0;
v33_table[39].bit_5 = 1;
v33_table[39].bit_6 = 1;
v33_table[39].bit_7 = 1;
v33_table[39].index =39;

v33_table[40].xcoord = 4;
v33_table[40].ycoord = 1;
v33_table[40].bit_1 = 0;
v33_table[40].bit_2 = 1;
v33_table[40].bit_3 = 0;
v33_table[40].bit_4 = 1;
v33_table[40].bit_5 = 0;
v33_table[40].bit_6 = 0;
v33_table[40].bit_7 = 0;
v33_table[40].index =40;

v33_table[41].xcoord = -3;
v33_table[41].ycoord = -2;
v33_table[41].bit_1 = 0;
v33_table[41].bit_2 = 1;
v33_table[41].bit_3 = 0;
v33_table[41].bit_4 = 1;
v33_table[41].bit_5 = 0;
v33_table[41].bit_6 = 0;
v33_table[41].bit_7 = 1;
v33_table[41].index =41;

v33_table[42].xcoord =-2;
v33_table[42].ycoord = 3;
v33_table[42].bit_1 = 0;
v33_table[42].bit_2 = 1;
v33_table[42].bit_3 = 0;
v33_table[42].bit_4 = 1;
v33_table[42].bit_5 = 0;
v33_table[42].bit_6 = 1;
v33_table[42].bit_7 = 0;
v33_table[42].index = 42;

v33_table[43].xcoord = 1;
v33_table[43].ycoord = -4;
v33_table[43].bit_1 = 0;
v33_table[43].bit_2 = 1;
v33_table[43].bit_3 = 0;
v33_table[43].bit_4 = 1;
v33_table[43].bit_5 = 0;
v33_table[43].bit_6 = 1;
v33_table[43].bit_7 = 1;
v33_table[43].index = 43;
```

```
v33_table[44].xcoord =-4;
v33_table[44].ycoord = -1;
v33_table[44].bit_1 =  0;
v33_table[44].bit_2 =  1;
v33_table[44].bit_3 =  0;
v33_table[44].bit_4 =  1;
v33_table[44].bit_5 =  1;
v33_table[44].bit_6 = 0;
v33_table[44].bit_7 = 0;
v33_table[44].index =  44;

v33_table[45].xcoord = 3;
v33_table[45].ycoord = 2;
v33_table[45].bit_1 = 0;
v33_table[45].bit_2 = 1;
v33_table[45].bit_3 = 0;
v33_table[45].bit_4 = 1;
v33_table[45].bit_5 = 1;
v33_table[45].bit_6 = 0;
v33_table[45].bit_7 = 1;
v33_table[45].index = 45;

v33_table[46].xcoord = 2;
v33_table[46].ycoord = -3;
v33_table[46].bit_1 = 0;
v33_table[46].bit_2 = 1;
v33_table[46].bit_3 = 0;
v33_table[46].bit_4 = 1;
v33_table[46].bit_5 = 1;
v33_table[46].bit_6 = 1;
v33_table[46].bit_7 = 0;
v33_table[46].index = 46;

v33_table[47].xcoord =  -1;
v33_table[47].ycoord =  4;
v33_table[47].bit_1 = 0;
v33_table[47].bit_2 = 1;
v33_table[47].bit_3 = 0;
v33_table[47].bit_4 = 1;
v33_table[47].bit_5 = 1;
v33_table[47].bit_6 = 1;
v33_table[47].bit_7 = 1;
v33_table[47].index = 47;

v33_table[48].xcoord = 0;
v33_table[48].ycoord =-3;
v33_table[48].bit_1 = 0;
v33_table[48].bit_2 = 1;
```

```
v33_table[48].bit_3 = 1;
v33_table[48].bit_4 = 0;
v33_table[48].bit_5 = 0;
v33_table[48].bit_6 = 0;
v33_table[48].bit_7 = 0;
v33_table[48].index = 48;

v33_table[49].xcoord = 1;
v33_table[49].ycoord = 2;
v33_table[49].bit_1 = 0;
v33_table[49].bit_2 = 1;
v33_table[49].bit_3 = 1;
v33_table[49].bit_4 = 0;
v33_table[49].bit_5 = 0;
v33_table[49].bit_6 = 0;
v33_table[49].bit_7 = 1;
v33_table[49].index = 49;

v33_table[50].xcoord = 2;
v33_table[50].ycoord =-1;
v33_table[50].bit_1 = 0;
v33_table[50].bit_2 = 1;
v33_table[50].bit_3 = 1;
v33_table[50].bit_4 = 0;
v33_table[50].bit_5 = 0;
v33_table[50].bit_6 = 1;
v33_table[50].bit_7 = 0;
v33_table[50].index = 50;

v33_table[51].xcoord =-3;
v33_table[51].ycoord = 0;
v33_table[51].bit_1 = 0;
v33_table[51].bit_2 = 1;
v33_table[51].bit_3 = 1;
v33_table[51].bit_4 = 0;
v33_table[51].bit_5 = 0;
v33_table[51].bit_6 = 1;
v33_table[51].bit_7 = 1;
v33_table[51].index = 51;

v33_table[52].xcoord = 0;
v33_table[52].ycoord = 3;
v33_table[52].bit_1 = 0;
v33_table[52].bit_2 = 1;
v33_table[52].bit_3 = 1;
v33_table[52].bit_4 = 0;
v33_table[52].bit_5 = 1;
v33_table[52].bit_6 = 0;
v33_table[52].bit_7 = 0;
```

```
v33_table[52].index = 52;

v33_table[53].xcoord = -1;
v33_table[53].ycoord =-2;
v33_table[53].bit_1 = 0;
v33_table[53].bit_2 = 1;
v33_table[53].bit_3 = 1;
v33_table[53].bit_4 = 0;
v33_table[53].bit_5 = 1;
v33_table[53].bit_6 = 0;
v33_table[53].bit_7 = 1;
v33_table[53].index = 53;

v33_table[54].xcoord =-2;
v33_table[54].ycoord = 1;
v33_table[54].bit_1 = 0;
v33_table[54].bit_2 = 1;
v33_table[54].bit_3 = 1;
v33_table[54].bit_4 = 0;
v33_table[54].bit_5 = 1;
v33_table[54].bit_6 = 1;
v33_table[54].bit_7 = 0;
v33_table[54].index = 54;

v33_table[55].xcoord =  3;
v33_table[55].ycoord =  0;
v33_table[55].bit_1 = 0;
v33_table[55].bit_2 = 1;
v33_table[55].bit_3 = 1;
v33_table[55].bit_4 = 0;
v33_table[55].bit_5 = 1;
v33_table[55].bit_6 = 1;
v33_table[55].bit_7 = 1;
v33_table[55].index = 55;

v33_table[56].xcoord = 0;
v33_table[56].ycoord = 1;
v33_table[56].bit_1 = 0;
v33_table[56].bit_2 = 1;
v33_table[56].bit_3 = 1;
v33_table[56].bit_4 = 1;
v33_table[56].bit_5 = 0;
v33_table[56].bit_6 = 0;
v33_table[56].bit_7 = 0;
v33_table[56].index = 56;

v33_table[57].xcoord =  1;
v33_table[57].ycoord =-2;
v33_table[57].bit_1 = 0;
```

```
v33_table[57].bit_2 = 1;
v33_table[57].bit_3 = 1;
v33_table[57].bit_4 = 1;
v33_table[57].bit_5 = 0;
v33_table[57].bit_6 = 0;
v33_table[57].bit_7 = 1;
v33_table[57].index = 57;

v33_table[58].xcoord =-2;
v33_table[58].ycoord =-1;
v33_table[58].bit_1 = 0;
v33_table[58].bit_2 = 1;
v33_table[58].bit_3 = 1;
v33_table[58].bit_4 = 1;
v33_table[58].bit_5 = 0;
v33_table[58].bit_6 = 1;
v33_table[58].bit_7 = 0;
v33_table[58].index = 58;

v33_table[59].xcoord = 1;
v33_table[59].ycoord =  0;
v33_table[59].bit_1 = 0;
v33_table[59].bit_2 = 1;
v33_table[59].bit_3 = 1;
v33_table[59].bit_4 = 1;
v33_table[59].bit_5 = 0;
v33_table[59].bit_6 = 1;
v33_table[59].bit_7 = 1;
v33_table[59].index = 59;

v33_table[60].xcoord =  0;
v33_table[60].ycoord =-1;
v33_table[60].bit_1 = 0;
v33_table[60].bit_2 = 1;
v33_table[60].bit_3 = 1;
v33_table[60].bit_4 = 1;
v33_table[60].bit_5 = 1;
v33_table[60].bit_6 = 0;
v33_table[60].bit_7 = 0;
v33_table[60].index = 60;

v33_table[61].xcoord =-1;
v33_table[61].ycoord = 2;
v33_table[61].bit_1 = 0;
v33_table[61].bit_2 = 1;
v33_table[61].bit_3 = 1;
v33_table[61].bit_4 = 1;
v33_table[61].bit_5 = 1;
v33_table[61].bit_6 = 0;
```

```
v33_table[61].bit_7 = 1;
v33_table[61].index = 61;

v33_table[62].xcoord = 2;
v33_table[62].ycoord = 1;
v33_table[62].bit_1 = 0;
v33_table[62].bit_2 = 1;
v33_table[62].bit_3 = 1;
v33_table[62].bit_4 = 1;
v33_table[62].bit_5 = 1;
v33_table[62].bit_6 = 1;
v33_table[62].bit_7 = 0;
v33_table[62].index = 62;

v33_table[63].xcoord = -1;
v33_table[63].ycoord = 0;
v33_table[63].bit_1 = 0;
v33_table[63].bit_2 = 1;
v33_table[63].bit_3 = 1;
v33_table[63].bit_4 = 1;
v33_table[63].bit_5 = 1;
v33_table[63].bit_6 = 1;
v33_table[63].bit_7 = 1;
v33_table[63].index = 63;

v33_table[64].xcoord = 8;
v33_table[64].ycoord = -3;
v33_table[64].bit_1 = 1;
v33_table[64].bit_2 = 0;
v33_table[64].bit_3 = 0;
v33_table[64].bit_4 = 0;
v33_table[64].bit_5 = 0;
v33_table[64].bit_6 = 0;
v33_table[64].bit_7 = 0;
v33_table[64].index = 64;

v33_table[65].xcoord = -7;
v33_table[65].ycoord =  2;
v33_table[65].bit_1 = 1;
v33_table[65].bit_2 = 0;
v33_table[65].bit_3 = 0;
v33_table[65].bit_4 = 0;
v33_table[65].bit_5 = 0;
v33_table[65].bit_6 = 0;
v33_table[65].bit_7 = 1;
v33_table[65].index = 65;

v33_table[66].xcoord = 2;
v33_table[66].ycoord = 7;
```

```
v33_table[66].bit_1 = 1;
v33_table[66].bit_2 = 0;
v33_table[66].bit_3 = 0;
v33_table[66].bit_4 = 0;
v33_table[66].bit_5 = 0;
v33_table[66].bit_6 = 1;
v33_table[66].bit_7 = 0;
v33_table[66].index = 66;

v33_table[67].xcoord = -3;
v33_table[67].ycoord =-8;
v33_table[67].bit_1 = 1;
v33_table[67].bit_2 = 0;
v33_table[67].bit_3 = 0;
v33_table[67].bit_4 = 0;
v33_table[67].bit_5 = 0;
v33_table[67].bit_6 = 1;
v33_table[67].bit_7 = 1;
v33_table[67].index = 67;

v33_table[68].xcoord =-8;
v33_table[68].ycoord = 3;
v33_table[68].bit_1 = 1;
v33_table[68].bit_2 = 0;
v33_table[68].bit_3 = 0;
v33_table[68].bit_4 = 0;
v33_table[68].bit_5 = 1;
v33_table[68].bit_6 = 0;
v33_table[68].bit_7 = 0;
v33_table[68].index = 68;

v33_table[69].xcoord = 7;
v33_table[69].ycoord = -2;
v33_table[69].bit_1 = 1;
v33_table[69].bit_2 = 0;
v33_table[69].bit_3 = 0;
v33_table[69].bit_4 = 0;
v33_table[69].bit_5 = 1;
v33_table[69].bit_6 = 0;
v33_table[69].bit_7 = 1;
v33_table[69].index = 69;

v33_table[70].xcoord = -2;
v33_table[70].ycoord =-7;
v33_table[70].bit_1 = 1;
v33_table[70].bit_2 = 0;
v33_table[70].bit_3 = 0;
v33_table[70].bit_4 = 0;
v33_table[70].bit_5 = 1;
```

```
v33_table[70].bit_6 = 1;
v33_table[70].bit_7 = 0;
v33_table[70].index = 70;

v33_table[71].xcoord = 3;
v33_table[71].ycoord = 8;
v33_table[71].bit_1 = 1;
v33_table[71].bit_2 = 0;
v33_table[71].bit_3 = 0;
v33_table[71].bit_4 = 0;
v33_table[71].bit_5 = 1;
v33_table[71].bit_6 = 1;
v33_table[71].bit_7 = 1;
v33_table[71].index = 71;

v33_table[72].xcoord = 8;
v33_table[72].ycoord = 1;
v33_table[72].bit_1 = 1;
v33_table[72].bit_2 = 0;
v33_table[72].bit_3 = 0;
v33_table[72].bit_4 = 1;
v33_table[72].bit_5 = 0;
v33_table[72].bit_6 = 0;
v33_table[72].bit_7 = 0;
v33_table[72].index = 72;

v33_table[73].xcoord = -7;
v33_table[73].ycoord =-2;
v33_table[73].bit_1 = 1;
v33_table[73].bit_2 = 0;
v33_table[73].bit_3 = 0;
v33_table[73].bit_4 = 1;
v33_table[73].bit_5 = 0;
v33_table[73].bit_6 = 0;
v33_table[73].bit_7 = 1;
v33_table[73].index = 73;

v33_table[74].xcoord =-2;
v33_table[74].ycoord =  7;
v33_table[74].bit_1 = 1;
v33_table[74].bit_2 = 0;
v33_table[74].bit_3 = 0;
v33_table[74].bit_4 = 1;
v33_table[74].bit_5 = 0;
v33_table[74].bit_6 = 1;
v33_table[74].bit_7 = 0;
v33_table[74].index = 74;

v33_table[75].xcoord =  1;
```

```
v33_table[75].ycoord = -8;
v33_table[75].bit_1 = 1;
v33_table[75].bit_2 = 0;
v33_table[75].bit_3 = 0;
v33_table[75].bit_4 = 1;
v33_table[75].bit_5 = 0;
v33_table[75].bit_6 = 1;
v33_table[75].bit_7 = 1;
v33_table[75].index = 75;

v33_table[76].xcoord =-8;
v33_table[76].ycoord =-1;
v33_table[76].bit_1 = 1;
v33_table[76].bit_2 = 0;
v33_table[76].bit_3 = 0;
v33_table[76].bit_4 = 1;
v33_table[76].bit_5 = 1;
v33_table[76].bit_6 = 0;
v33_table[76].bit_7 = 0;
v33_table[76].index = 76;

v33_table[77].xcoord = 7;
v33_table[77].ycoord = 2;
v33_table[77].bit_1 = 1;
v33_table[77].bit_2 = 0;
v33_table[77].bit_3 = 0;
v33_table[77].bit_4 = 1;
v33_table[77].bit_5 = 1;
v33_table[77].bit_6 = 0;
v33_table[77].bit_7 = 1;
v33_table[77].index = 77;

v33_table[78].xcoord = 2;
v33_table[78].ycoord =-7;
v33_table[78].bit_1 = 1;
v33_table[78].bit_2 = 0;
v33_table[78].bit_3 = 0;
v33_table[78].bit_4 = 1;
v33_table[78].bit_5 = 1;
v33_table[78].bit_6 = 1;
v33_table[78].bit_7 = 0;
v33_table[78].index = 78;

v33_table[79].xcoord =-1;
v33_table[79].ycoord =  8;
v33_table[79].bit_1 = 1;
v33_table[79].bit_2 = 0;
v33_table[79].bit_3 = 0;
v33_table[79].bit_4 = 1;
```

```
v33_table[79].bit_5 = 1;
v33_table[79].bit_6 = 1;
v33_table[79].bit_7 = 1;
v33_table[79].index = 79;

v33_table[80].xcoord = -4;
v33_table[80].ycoord =-7;
v33_table[80].bit_1 = 1;
v33_table[80].bit_2 = 0;
v33_table[80].bit_3 = 1;
v33_table[80].bit_4 = 0;
v33_table[80].bit_5 = 0;
v33_table[80].bit_6 = 0;
v33_table[80].bit_7 = 0;
v33_table[80].index = 80;

v33_table[81].xcoord = 5;
v33_table[81].ycoord = 6;
v33_table[81].bit_1 = 1;
v33_table[81].bit_2 = 0;
v33_table[81].bit_3 = 1;
v33_table[81].bit_4 = 0;
v33_table[81].bit_5 = 0;
v33_table[81].bit_6 = 0;
v33_table[81].bit_7 = 1;
v33_table[81].index = 81;

v33_table[82].xcoord = 6;
v33_table[82].ycoord =-5;
v33_table[82].bit_1 = 1;
v33_table[82].bit_2 = 0;
v33_table[82].bit_3 = 1;
v33_table[82].bit_4 = 0;
v33_table[82].bit_5 = 0;
v33_table[82].bit_6 = 1;
v33_table[82].bit_7 = 0;
v33_table[82].index = 82;

v33_table[83].xcoord = -7;
v33_table[83].ycoord = 4;
v33_table[83].bit_1 = 1;
v33_table[83].bit_2 = 0;
v33_table[83].bit_3 = 1;
v33_table[83].bit_4 = 0;
v33_table[83].bit_5 = 0;
v33_table[83].bit_6 = 1;
v33_table[83].bit_7 = 1;
v33_table[83].index = 83;
```

```
v33_table[84].xcoord = 4;
v33_table[84].ycoord =  7;
v33_table[84].bit_1 = 1;
v33_table[84].bit_2 = 0;
v33_table[84].bit_3 = 1;
v33_table[84].bit_4 = 0;
v33_table[84].bit_5 = 1;
v33_table[84].bit_6 = 0;
v33_table[84].bit_7 = 0;
v33_table[84].index = 84;

v33_table[85].xcoord = -5;
v33_table[85].ycoord = -6;
v33_table[85].bit_1 = 1;
v33_table[85].bit_2 = 0;
v33_table[85].bit_3 = 1;
v33_table[85].bit_4 = 0;
v33_table[85].bit_5 = 1;
v33_table[85].bit_6 = 0;
v33_table[85].bit_7 = 1;
v33_table[85].index = 85;

v33_table[86].xcoord =-6;
v33_table[86].ycoord = 5;
v33_table[86].bit_1 = 1;
v33_table[86].bit_2 = 0;
v33_table[86].bit_3 = 1;
v33_table[86].bit_4 = 0;
v33_table[86].bit_5 = 1;
v33_table[86].bit_6 = 1;
v33_table[86].bit_7 = 0;
v33_table[86].index = 86;

v33_table[87].xcoord =  7;
v33_table[87].ycoord =-4;
v33_table[87].bit_1 = 1;
v33_table[87].bit_2 = 0;
v33_table[87].bit_3 = 1;
v33_table[87].bit_4 = 0;
v33_table[87].bit_5 = 1;
v33_table[87].bit_6 = 1;
v33_table[87].bit_7 = 1;
v33_table[87].index = 87;

v33_table[88].xcoord =-4;
v33_table[88].ycoord = 5;
v33_table[88].bit_1 = 1;
v33_table[88].bit_2 = 0;
v33_table[88].bit_3 = 1;
```

```
v33_table[88].bit_4 = 1;
v33_table[88].bit_5 = 0;
v33_table[88].bit_6 = 0;
v33_table[88].bit_7 = 0;
v33_table[88].index = 88;

v33_table[89].xcoord = 5;
v33_table[89].ycoord = -6;
v33_table[89].bit_1 = 1;
v33_table[89].bit_2 = 0;
v33_table[89].bit_3 = 1;
v33_table[89].bit_4 = 1;
v33_table[89].bit_5 = 0;
v33_table[89].bit_6 = 0;
v33_table[89].bit_7 = 1;
v33_table[89].index = 89;

v33_table[90].xcoord = -6;
v33_table[90].ycoord =-5;
v33_table[90].bit_1 = 1;
v33_table[90].bit_2 = 0;
v33_table[90].bit_3 = 1;
v33_table[90].bit_4 = 1;
v33_table[90].bit_5 = 0;
v33_table[90].bit_6 = 1;
v33_table[90].bit_7 = 0;
v33_table[90].index = 90;

v33_table[91].xcoord = 5;
v33_table[91].ycoord = 4;
v33_table[91].bit_1 = 1;
v33_table[91].bit_2 = 0;
v33_table[91].bit_3 = 1;
v33_table[91].bit_4 = 1;
v33_table[91].bit_5 = 0;
v33_table[91].bit_6 = 1;
v33_table[91].bit_7 = 1;
v33_table[91].index = 91;

v33_table[92].xcoord = 4;
v33_table[92].ycoord =-5;
v33_table[92].bit_1 = 1;
v33_table[92].bit_2 = 0;
v33_table[92].bit_3 = 1;
v33_table[92].bit_4 = 1;
v33_table[92].bit_5 = 1;
v33_table[92].bit_6 = 0;
v33_table[92].bit_7 = 0;
v33_table[92].index = 92;
```

```
v33_table[93].xcoord = -5;
v33_table[93].ycoord = 6;
v33_table[93].bit_1 = 1;
v33_table[93].bit_2 = 0;
v33_table[93].bit_3 = 1;
v33_table[93].bit_4 = 1;
v33_table[93].bit_5 = 1;
v33_table[93].bit_6 = 0;
v33_table[93].bit_7 = 1;
v33_table[93].index = 93;

v33_table[94].xcoord = 6;
v33_table[94].ycoord =  5;
v33_table[94].bit_1 = 1;
v33_table[94].bit_2 = 0;
v33_table[94].bit_3 = 1;
v33_table[94].bit_4 = 1;
v33_table[94].bit_5 = 1;
v33_table[94].bit_6 = 1;
v33_table[94].bit_7 = 0;
v33_table[94].index = 94;

v33_table[95].xcoord = -5;
v33_table[95].ycoord = -4;
v33_table[95].bit_1 = 1;
v33_table[95].bit_2 = 0;
v33_table[95].bit_3 = 1;
v33_table[95].bit_4 = 1;
v33_table[95].bit_5 = 1;
v33_table[95].bit_6 = 1;
v33_table[95].bit_7 = 1;
v33_table[95].index = 95;

v33_table[96].xcoord = 4;
v33_table[96].ycoord =-7;
v33_table[96].bit_1 = 1;
v33_table[96].bit_2 = 1;
v33_table[96].bit_3 = 0;
v33_table[96].bit_4 = 0;
v33_table[96].bit_5 = 0;
v33_table[96].bit_6 = 0;
v33_table[96].bit_7 = 0;
v33_table[96].index = 96;

v33_table[97].xcoord = -3;
v33_table[97].ycoord = 6;
v33_table[97].bit_1 = 1;
v33_table[97].bit_2 = 1;
```

```
v33_table[97].bit_3 = 0;
v33_table[97].bit_4 = 0;
v33_table[97].bit_5 = 0;
v33_table[97].bit_6 = 0;
v33_table[97].bit_7 = 1;
v33_table[97].index = 97;

v33_table[98].xcoord = 6;
v33_table[98].ycoord = 3;
v33_table[98].bit_1 = 1;
v33_table[98].bit_2 = 1;
v33_table[98].bit_3 = 0;
v33_table[98].bit_4 = 0;
v33_table[98].bit_5 = 0;
v33_table[98].bit_6 = 1;
v33_table[98].bit_7 = 0;
v33_table[98].index = 98;

v33_table[99].xcoord =-7;
v33_table[99].ycoord = -4;
v33_table[99].bit_1 = 1;
v33_table[99].bit_2 = 1;
v33_table[99].bit_3 = 0;
v33_table[99].bit_4 = 0;
v33_table[99].bit_5 = 0;
v33_table[99].bit_6 = 1;
v33_table[99].bit_7 = 1;
v33_table[99].index = 99;

v33_table[100].xcoord = -4;
v33_table[100].ycoord = 7;
v33_table[100].bit_1 = 1;
v33_table[100].bit_2 = 1;
v33_table[100].bit_3 = 0;
v33_table[100].bit_4 = 0;
v33_table[100].bit_5 = 1;
v33_table[100].bit_6 = 0;
v33_table[100].bit_7 = 0;
v33_table[100].index =100;

v33_table[101].xcoord = 3;
v33_table[101].ycoord =-6;
v33_table[101].bit_1 = 1;
v33_table[101].bit_2 = 1;
v33_table[101].bit_3 = 0;
v33_table[101].bit_4 = 0;
v33_table[101].bit_5 = 1;
v33_table[101].bit_6 = 0;
v33_table[101].bit_7 = 1;
```

```
v33_table[101].index =101;

v33_table[102].xcoord =-6;
v33_table[102].ycoord =-3;
v33_table[102].bit_1 = 1;
v33_table[102].bit_2 = 1;
v33_table[102].bit_3 = 0;
v33_table[102].bit_4 = 0;
v33_table[102].bit_5 = 1;
v33_table[102].bit_6 = 1;
v33_table[102].bit_7 = 0;
v33_table[102].index =102;

v33_table[103].xcoord =  7;
v33_table[103].ycoord = 4;
v33_table[103].bit_1 = 1;
v33_table[103].bit_2 = 1;
v33_table[103].bit_3 = 0;
v33_table[103].bit_4 = 0;
v33_table[103].bit_5 = 1;
v33_table[103].bit_6 = 1;
v33_table[103].bit_7 = 1;
v33_table[103].index =103;

v33_table[104].xcoord = 4;
v33_table[104].ycoord =  5;
v33_table[104].bit_1 = 1;
v33_table[104].bit_2 = 1;
v33_table[104].bit_3 = 0;
v33_table[104].bit_4 = 1;
v33_table[104].bit_5 = 0;
v33_table[104].bit_6 = 0;
v33_table[104].bit_7 = 0;
v33_table[104].index =104;

v33_table[105].xcoord = -3;
v33_table[105].ycoord = -6;
v33_table[105].bit_1 = 1;
v33_table[105].bit_2 = 1;
v33_table[105].bit_3 = 0;
v33_table[105].bit_4 = 1;
v33_table[105].bit_5 = 0;
v33_table[105].bit_6 = 0;
v33_table[105].bit_7 = 1;
v33_table[105].index =105;

v33_table[106].xcoord =-6;
v33_table[106].ycoord = 3;
v33_table[106].bit_1 = 1;
```

```
v33_table[106].bit_2 = 1;
v33_table[106].bit_3 = 0;
v33_table[106].bit_4 = 1;
v33_table[106].bit_5 = 0;
v33_table[106].bit_6 = 1;
v33_table[106].bit_7 = 0;
v33_table[106].index =106;

v33_table[107].xcoord =  5;
v33_table[107].ycoord =-4;
v33_table[107].bit_1 = 1;
v33_table[107].bit_2 = 1;
v33_table[107].bit_3 = 0;
v33_table[107].bit_4 = 1;
v33_table[107].bit_5 = 0;
v33_table[107].bit_6 = 1;
v33_table[107].bit_7 = 1;
v33_table[107].index =107;

v33_table[108].xcoord =-4;
v33_table[108].ycoord =-5;
v33_table[108].bit_1 = 1;
v33_table[108].bit_2 = 1;
v33_table[108].bit_3 = 0;
v33_table[108].bit_4 = 1;
v33_table[108].bit_5 = 1;
v33_table[108].bit_6 = 0;
v33_table[108].bit_7 = 0;
v33_table[108].index =108;

v33_table[109].xcoord = 3;
v33_table[109].ycoord =  6;
v33_table[109].bit_1 = 1;
v33_table[109].bit_2 = 1;
v33_table[109].bit_3 = 0;
v33_table[109].bit_4 = 1;
v33_table[109].bit_5 = 1;
v33_table[109].bit_6 = 0;
v33_table[109].bit_7 = 1;
v33_table[109].index =109;

v33_table[110].xcoord =  6;
v33_table[110].ycoord =-3;
v33_table[110].bit_1 = 1;
v33_table[110].bit_2 = 1;
v33_table[110].bit_3 = 0;
v33_table[110].bit_4 = 1;
v33_table[110].bit_5 = 1;
v33_table[110].bit_6 = 1;
```

```
v33_table[110].bit_7 = 0;
v33_table[110].index =110;

v33_table[111].xcoord =-5;
v33_table[111].ycoord = 4;
v33_table[111].bit_1 = 1;
v33_table[111].bit_2 = 1;
v33_table[111].bit_3 = 0;
v33_table[111].bit_4 = 1;
v33_table[111].bit_5 = 1;
v33_table[111].bit_6 = 1;
v33_table[111].bit_7 = 1;
v33_table[111].index =111;

v33_table[112].xcoord = 0;
v33_table[112].ycoord =-7;
v33_table[112].bit_1 = 1;
v33_table[112].bit_2 = 1;
v33_table[112].bit_3 = 1;
v33_table[112].bit_4 = 0;
v33_table[112].bit_5 = 0;
v33_table[112].bit_6 = 0;
v33_table[112].bit_7 = 0;
v33_table[112].index =112;

v33_table[113].xcoord =  1;
v33_table[113].ycoord = 6;
v33_table[113].bit_1 = 1;
v33_table[113].bit_2 = 1;
v33_table[113].bit_3 = 1;
v33_table[113].bit_4 = 0;
v33_table[113].bit_5 = 0;
v33_table[113].bit_6 = 0;
v33_table[113].bit_7 = 1;
v33_table[113].index =113;

v33_table[114].xcoord = 6;
v33_table[114].ycoord = -1;
v33_table[114].bit_1 = 1;
v33_table[114].bit_2 = 1;
v33_table[114].bit_3 = 1;
v33_table[114].bit_4 = 0;
v33_table[114].bit_5 = 0;
v33_table[114].bit_6 = 1;
v33_table[114].bit_7 = 0;
v33_table[114].index =114;

v33_table[115].xcoord = -7;
v33_table[115].ycoord =  0;
```

```
v33_table[115].bit_1 = 1;
v33_table[115].bit_2 = 1;
v33_table[115].bit_3 = 1;
v33_table[115].bit_4 = 0;
v33_table[115].bit_5 = 0;
v33_table[115].bit_6 = 1;
v33_table[115].bit_7 = 1;
v33_table[115].index =115;

v33_table[116].xcoord = 0;
v33_table[116].ycoord = 7;
v33_table[116].bit_1 = 1;
v33_table[116].bit_2 = 1;
v33_table[116].bit_3 = 1;
v33_table[116].bit_4 = 0;
v33_table[116].bit_5 = 1;
v33_table[116].bit_6 = 0;
v33_table[116].bit_7 = 0;
v33_table[116].index =116;

v33_table[117].xcoord = -1;
v33_table[117].ycoord =-6;
v33_table[117].bit_1 = 1;
v33_table[117].bit_2 = 1;
v33_table[117].bit_3 = 1;
v33_table[117].bit_4 = 0;
v33_table[117].bit_5 = 1;
v33_table[117].bit_6 = 0;
v33_table[117].bit_7 = 1;
v33_table[117].index =117;

v33_table[118].xcoord =-6;
v33_table[118].ycoord = 1;
v33_table[118].bit_1 = 1;
v33_table[118].bit_2 = 1;
v33_table[118].bit_3 = 1;
v33_table[118].bit_4 = 0;
v33_table[118].bit_5 = 1;
v33_table[118].bit_6 = 1;
v33_table[118].bit_7 = 0;
v33_table[118].index =118;

v33_table[119].xcoord = 7;
v33_table[119].ycoord =  0;
v33_table[119].bit_1 = 1;
v33_table[119].bit_2 = 1;
v33_table[119].bit_3 = 1;
v33_table[119].bit_4 = 0;
v33_table[119].bit_5 = 1;
```

```
v33_table[119].bit_6 = 1;
v33_table[119].bit_7 = 1;
v33_table[119].index =119;

v33_table[120].xcoord =  0;
v33_table[120].ycoord = 5;
v33_table[120].bit_1 = 1;
v33_table[120].bit_2 = 1;
v33_table[120].bit_3 = 1;
v33_table[120].bit_4 = 1;
v33_table[120].bit_5 = 0;
v33_table[120].bit_6 = 0;
v33_table[120].bit_7 = 0;
v33_table[120].index =120;

v33_table[121].xcoord = 1;
v33_table[121].ycoord =-6;
v33_table[121].bit_1 = 1;
v33_table[121].bit_2 = 1;
v33_table[121].bit_3 = 1;
v33_table[121].bit_4 = 1;
v33_table[121].bit_5 = 0;
v33_table[121].bit_6 = 0;
v33_table[121].bit_7 = 1;
v33_table[121].index =121;

v33_table[122].xcoord =-6;
v33_table[122].ycoord =-1;
v33_table[122].bit_1 = 1;
v33_table[122].bit_2 = 1;
v33_table[122].bit_3 = 1;
v33_table[122].bit_4 = 1;
v33_table[122].bit_5 = 0;
v33_table[122].bit_6 = 1;
v33_table[122].bit_7 = 0;
v33_table[122].index =122;

v33_table[123].xcoord =  5;
v33_table[123].ycoord = 0;
v33_table[123].bit_1 = 1;
v33_table[123].bit_2 = 1;
v33_table[123].bit_3 = 1;
v33_table[123].bit_4 = 1;
v33_table[123].bit_5 = 0;
v33_table[123].bit_6 = 1;
v33_table[123].bit_7 = 1;
v33_table[123].index =123;

v33_table[124].xcoord = 0;
```

```
v33_table[124].ycoord = -5;
v33_table[124].bit_1 = 1;
v33_table[124].bit_2 = 1;
v33_table[124].bit_3 = 1;
v33_table[124].bit_4 = 1;
v33_table[124].bit_5 = 1;
v33_table[124].bit_6 = 0;
v33_table[124].bit_7 = 0;
v33_table[124].index =124;

v33_table[125].xcoord = -1;
v33_table[125].ycoord =  6;
v33_table[125].bit_1 = 1;
v33_table[125].bit_2 = 1;
v33_table[125].bit_3 = 1;
v33_table[125].bit_4 = 1;
v33_table[125].bit_5 = 1;
v33_table[125].bit_6 = 0;
v33_table[125].bit_7 = 1;
v33_table[125].index =125;

v33_table[126].xcoord = 6;
v33_table[126].ycoord = 1;
v33_table[126].bit_1 = 1;
v33_table[126].bit_2 = 1;
v33_table[126].bit_3 = 1;
v33_table[126].bit_4 = 1;
v33_table[126].bit_5 = 1;
v33_table[126].bit_6 = 1;
v33_table[126].bit_7 = 0;
v33_table[126].index =126;

v33_table[127].xcoord = -5;
v33_table[127].ycoord = 0;
v33_table[127].bit_1 = 1;
v33_table[127].bit_2 = 1;
v33_table[127].bit_3 = 1;
v33_table[127].bit_4 = 1;
v33_table[127].bit_5 = 1;
v33_table[127].bit_6 = 1;
v33_table[127].bit_7 = 1;
v33_table[127].index =127;

/* Begin reading binary.txt,the binary file to be
*/
/* transmitted
*/
while (!eoffnd)
  {
```

```
  ct=0;
  while( ((ct < 6) && (output[ct]=getc(fptr)) != EOF) )
  {
       ct++;
  }
  if(ct != 6)         /* got to eof before we read six bits
*/
  {
    eoffnd = 1;
    while(ct< 6)
    {
      output[ct] = '0';
      ct++;
    }
  }
  for(ct=0; ct<6; ct++)
  {
    if (output[ct] == EOF)
      eoffnd = 1;
  }
/* SET THE VALUES OF THE BITS TO THE APPROPRIATE VARIABLE
AS*/
/* REFERENCED IN CCITT STANDART V.33
*/
   q1 = output[0]-48;
   q2 = output[1]-48;
   y3new = output[2]-48;
   y4new = output[3]-48;
   y5new = output[4]-48;
   y6new = output[5]-48;

/* DIFFERENTIALLY ENCODE THE TWO BITS IN TIME
*/
diffencoder(q1, q2);

/* ENCODE THE BITS USING THE ENCODING SCHEME SPECIFIED
*/
/* THE SCHEMES ARE FUNCTIONALLY EQUIVALENT, THERE ARE TWO
*/
/* TO VERIFY THAT THE ENCODING PROCESS IS CORRECT.
*/
if (strcmp(encoder_type,"1")==0)
     encoder();
else
     if (strcmp(encoder_type,"2")==0)
          encoder2();
     else
     {    printf("\n**Invalid Encoder Type (Argument
3)**\n");
```

```
          exit(1);
      }
/* TAKE THE ENCODED BITS AND MAP THEM TO THE PROPER SIGNAL
*/
/* POINT ON THE V.33 SIGNAL CONSTELLATION
*/
mappervdot33(&x,&y);

     if(numxy < maxxy)
   {
/* LOG THE SYMBOL IN THE TRANSMITTED ARRAY
*/
     xy[numxy].xval = x;
     xy[numxy].yval = y;
     numxy++;
   }
   else
   printf("Too many xy bits for array\n");

/*CHANNEL NOISE SECTION */
/* GENERATE TO RV'S SCALED BY USER SPECIFICATION  */
u1 = ((float)random(20000))/20000.0;
u2 = ((float)random(20000))/20000.0;
r=sqrt(-2*log(u1));
a = 2*3.1415926*u2;
grv1 = r*sin(a)/(float)noise;
grv2 = r*cos(a)/(float)noise;
rvtot1 = rvtot1 + sqrt(grv1*grv1);
rvtot2 = rvtot2 + sqrt(grv2*grv2);
/* MODIFY THE THE TRANSMITTED ARRAY DUE TO NOISE INJECTION
*/
xy[numxy-1].xval = grv1+xy[numxy-1].xval;
xy[numxy-1].yval = grv2+xy[numxy-1].yval;

}
/*channel noise ends*/

/* APPEND 0's TO END OF MESSAGE TO ACCOMODATE FOR MEMORY
*/
for (zero_pad = 0; zero_pad < vit_mem_length; zero_pad++)
{xy[numxy + zero_pad].xval = 0;
xy[numxy + zero_pad].yval = 0;}
y1old = 0;
y2old = 0;
fclose(fptr);

/* DECODER SECTION */
/* LOOP THROUGH ONCE FOR EACH TRANSMITTED SYMBOL
*/
```

```
for (current = 0; current<numxy-1+vit_mem_length; current++)
{
iteration_count++;
/* CHECK FOR MOST LIKELY PATH TO EACH OF 0-63 STATES
*/
     for (state_to = 0; state_to < 64; state_to++)
     {
/* CHECK FOR BEST SYMBOL INTO EACH OF ABOVE NODES
*/
          for (poss_symbol = 0; poss_symbol < 64;
poss_symbol++)
          {
          discrep = sqrt((xy[current].xval -
v33_table[poss_symbol].xcoord)*
          (xy[current].xval - v33_table[poss_symbol].xcoord)
          + (xy[current].yval -
v33_table[poss_symbol].ycoord)*
          (xy[current].yval -
v33_table[poss_symbol].ycoord));

/* GET THE STATE THAT THE SYMBOL CAME FROM   */
state_from1(poss_symbol,state_to,&state_from_index);

     if ((discrep+tot_d_o[state_from_index]) > 29999)
     {
     printf("Discrepancy Has Exceeded 30000");
     exit(1);
   }

     if ((discrep +
tot_d_o[state_from_index])<=min_discrep[state_to])
          {
          min_discrep[state_to] =
(discrep+tot_d_o[state_from_index]);
          best_path[state_to] = poss_symbol;
          from_node[state_to] = state_from_index;
          }
     } /*for 0-63 possible symbols*/

     tot_d_n[state_to] = min_discrep[state_to];
}/* for 0-63 possible states to go to */

/* DO THE SAME FOR STATES 64-127   */
     for (state_to = 64; state_to < 128; state_to++)
     {
          for (poss_symbol = 64; poss_symbol < 128;
poss_symbol++)
          {
          discrep = sqrt((xy[current].xval -
```

```
v33_table[poss_symbol].xcoord)*(xy[current].xval -
v33_table[poss_symbol].xcoord)
          + (xy[current].yval -
v33_table[poss_symbol].ycoord)*(xy[current].yval -
v33_table[poss_symbol].ycoord));

state_from2(poss_symbol,state_to,&state_from_index);

     if ((discrep+tot_d_o[state_from_index]) > 29999)
     {
     printf("Discrepancy Has Exceeded 30000");
     exit(1);
   }

     if ((discrep +
tot_d_o[state_from_index])<=min_discrep[state_to])
          {
          min_discrep[state_to] =
(discrep+tot_d_o[state_from_index]);
          best_path[state_to] = poss_symbol;
          from_node[state_to] = state_from_index;
          }
     } /*for 64-127 possible symbols*/

     tot_d_n[state_to] = min_discrep[state_to];
}/* for 64-127 possible states to go to */

for (i=0; i<128; i++)
tot_d_o[i] = tot_d_n[i];
for (i=0; i<128; i++)
min_discrep[i] = 30000;
for (i=0; i<128; i++)
{
     for (l=0; l<vit_mem_length; l++)
/* UPDATE PATH AND DECODE LAST SYMBOL IN MEMORY   */
     path_new[i].path_c[l] =
path_old[from_node[i]].path_o[l];
   decoded[i] = path_new[i].path_c[0];
     for (j=1; j<vit_mem_length; j++)
   {
      path_new[i].path_c[j-1] = path_new[i].path_c[j];
   }
     path_new[i].path_c[vit_mem_length - 1] = best_path[i];
}
for (i=0; i<128; i++)
     for (l=0; l<vit_mem_length; l++)
     path_old[i].path_o[l] = path_new[i].path_c[l];

/* CHECK FOR DEMODULATION FAULTS                  */
```

```
if
((decoded[0]==decoded[1])&&(decoded[2]==decoded[3])&&(decode
d[0]==decoded[2])
&&(decoded[4]==decoded[5])&&(decoded[6]==decoded[7])&&(decod
ed[4]==decoded[6])
&&(decoded[8]==decoded[9])&&(decoded[10]==decoded[11])&&(dec
oded[8]==decoded[10])
&&(decoded[12]==decoded[13])&&(decoded[14]==decoded[15])&&(d
ecoded[12]==decoded[14])
&&(decoded[16]==decoded[17])&&(decoded[18]==decoded[19])&&(d
ecoded[16]==decoded[18])
&&(decoded[20]==decoded[21])&&(decoded[22]==decoded[23])&&(d
ecoded[20]==decoded[22])
&&(decoded[24]==decoded[25])&&(decoded[26]==decoded[27])&&(d
ecoded[24]==decoded[26])
&&(decoded[28]==decoded[29])&&(decoded[30]==decoded[31])&&(d
ecoded[28]==decoded[30])
&&(decoded[32]==decoded[33])&&(decoded[34]==decoded[35])&&(d
ecoded[32]==decoded[34])
&&(decoded[36]==decoded[37])&&(decoded[38]==decoded[39])&&(d
ecoded[36]==decoded[38])
&&(decoded[40]==decoded[41])&&(decoded[42]==decoded[43])&&(d
ecoded[40]==decoded[42])
&&(decoded[44]==decoded[45])&&(decoded[46]==decoded[47])&&(d
ecoded[44]==decoded[46])
&&(decoded[48]==decoded[49])&&(decoded[50]==decoded[51])&&(d
ecoded[48]==decoded[50])
&&(decoded[52]==decoded[53])&&(decoded[54]==decoded[55])&&(d
ecoded[52]==decoded[54])
&&(decoded[56]==decoded[57])&&(decoded[58]==decoded[59])&&(d
ecoded[56]==decoded[58])
&&(decoded[60]==decoded[61])&&(decoded[62]==decoded[63])&&(d
ecoded[60]==decoded[62])
&&(decoded[64]==decoded[65])&&(decoded[66]==decoded[67])&&(d
ecoded[64]==decoded[66])
&&(decoded[68]==decoded[69])&&(decoded[70]==decoded[71])&&(d
ecoded[68]==decoded[70])
&&(decoded[72]==decoded[73])&&(decoded[74]==decoded[75])&&(d
ecoded[72]==decoded[74])
&&(decoded[76]==decoded[77])&&(decoded[78]==decoded[79])&&(d
ecoded[76]==decoded[78])
&&(decoded[80]==decoded[81])&&(decoded[82]==decoded[83])&&(d
ecoded[80]==decoded[82])
&&(decoded[84]==decoded[85])&&(decoded[86]==decoded[87])&&(d
ecoded[84]==decoded[86])
&&(decoded[88]==decoded[89])&&(decoded[90]==decoded[91])&&(d
ecoded[88]==decoded[90])
&&(decoded[92]==decoded[93])&&(decoded[94]==decoded[95])&&(d
ecoded[92]==decoded[94])
```

```
&&(decoded[96]==decoded[97])&&(decoded[98]==decoded[99])&&(d
ecoded[96]==decoded[98])
&&(decoded[100]==decoded[101])&&(decoded[102]==decoded[103])
&&(decoded[100]==decoded[102])
&&(decoded[104]==decoded[105])&&(decoded[106]==decoded[107])
&&(decoded[104]==decoded[106])
&&(decoded[108]==decoded[109])&&(decoded[110]==decoded[111])
&&(decoded[108]==decoded[110])
&&(decoded[112]==decoded[113])&&(decoded[114]==decoded[115])
&&(decoded[112]==decoded[114])
&&(decoded[116]==decoded[117])&&(decoded[118]==decoded[119])
&&(decoded[116]==decoded[118])
&&(decoded[120]==decoded[121])&&(decoded[122]==decoded[123])
&&(decoded[120]==decoded[122])
&&(decoded[124]==decoded[125])&&(decoded[126]==decoded[127])
&&(decoded[124]==decoded[126]))
{ check = 1;
}

if
((decoded[0]==decoded[4])&&(decoded[4]==decoded[8])&&(decode
d[8]==decoded[12])
&&(decoded[12]==decoded[16])&&(decoded[16]==decoded[20])&&(d
ecoded[20]==decoded[24])
&&(decoded[24]==decoded[28])&&(decoded[28]==decoded[32])&&(d
ecoded[32]==decoded[36])
&&(decoded[36]==decoded[40])&&(decoded[40]==decoded[44])&&(d
ecoded[44]==decoded[48])
&&(decoded[48]==decoded[52])&&(decoded[52]==decoded[56])&&(d
ecoded[56]==decoded[60])
&&(decoded[60]==decoded[64])&&(decoded[64]==decoded[68])&&(d
ecoded[68]==decoded[72])
&&(decoded[72]==decoded[76])&&(decoded[76]==decoded[80])&&(d
ecoded[80]==decoded[84])
&&(decoded[84]==decoded[88])&&(decoded[88]==decoded[92])&&(d
ecoded[92]==decoded[96])
&&(decoded[96]==decoded[100])&&(decoded[100]==decoded[104])&
&(decoded[104]==decoded[108])
&&(decoded[108]==decoded[112])&&(decoded[112]==decoded[116])
&&(decoded[116]==decoded[120])
&&(decoded[120]==decoded[124])&&(iteration_count
>vit_mem_length)&&(check==1))
   {

/* DIFFERENTIALLY DECODE AND WRITE TO OUTPUT BINARY_O.TXT */
de_diffencoder(v33_table[decoded[0]].bit_2,v33_table[decoded
[0]].bit_3,&dec1,&dec2);
     fprintf(fptr8,"%d%d%d%d%d",dec1,dec2,
```

```
v33_table[decoded[0]].bit_4,v33_table[decoded[0]].bit_5,

v33_table[decoded[0]].bit_6,v33_table[decoded[0]].bit_7);
     }

     else if(iteration_count>vit_mem_length)
     {
/* DIFFERENTIALLY DECODE AND GUESS (DEMOD_FAULT)       */
de_diffencoder(v33_table[decoded[0]].bit_2,v33_table[decoded
[0]].bit_3,&dec1,&dec2);
     fprintf(fptr8,"%d%d%d%d%d%d",dec1,dec2,

v33_table[decoded[0]].bit_4,v33_table[decoded[0]].bit_5,

v33_table[decoded[0]].bit_6,v33_table[decoded[0]].bit_7);
   demod_fault++;
     }
check = 0;
}
fclose(fptr8);
report33(vit_mem_length,noise,rvtot1,rvtot2,demod_fault,numx
y,tfile,rfile);
return 0;
}
```

```
/* diffencoder */
/*  This function differentially encodes input to the
convolutional encoder */
/*FUNCTION DIFFENCODER(INT Q1, INT Q2) */
diffencoder(int q1, int q2)
{extern int y1new,y2new,y1old,y2old;
int input;
if (q1 == 0 && q2 == 0)
     {
     y1new = (q1 ^ y1old);
     y2new = (q2 ^ y2old);
     }
if (q1 == 0 && q2 == 1)
     {
     y1new = (q1 ^ y1old);
     y2new = (q2 ^ y2old);
     }
if (q1 == 1 && q2 == 0)
{
     if (y1old == 0 && y2old == 0)
     {
          y1new = 1;
          y2new = 0;
     }
     if (y1old == 0 && y2old == 1)
     {
          y1new = 1;
          y2new = 1;
     }
     if (y1old == 1 && y2old == 0)
     {
          y1new = 0;
          y2new = 1;
     }
     if (y1old == 1 && y2old == 1)
     {
          y1new = 0;
          y2new = 0;
     }
}
if (q1 == 1 && q2 == 1)
{
     if (y1old == 0 && y2old == 0)
     {
          y1new = 1;
          y2new = 1;
     }
     if (y1old == 0 && y2old == 1)
     {
```

```
            y1new = 1;
            y2new = 0;
      }
      if (y1old == 1 && y2old == 0)
      {
            y1new = 0;
            y2new = 0;
      }
      if (y1old == 1 && y2old == 1)
      {
            y1new = 0;
            y2new = 1;
      }
}
       return 0;
}
```

```
/* encoder()  */
/* This function is the convolutional encoder */
/* FUNCTION ENCODER() */
encoder()
{extern int
y0oldest,xnew,y1old,y2old,y0old,y1new,y2new,y0new,xold;
xnew = ((y0oldest^(y1old^y2old))^(y0old&(xold^y2old)));
y0new = ((xold^y2old)^(y0old&y1old));
xold = xnew;
y1old = y1new;
y2old = y2new;
y0oldest = y0old;
y0old = y0new;
/*printf("Conv. Output y0new = %d  y1new = %d  y2new =
%d",y0new,y1new,y2new);*/
return 0;
}
```

```
/* encoder2()   */
/* This function is the convolutional encoder */
/* FUNCTION ENCODER2()*/
encoder2()
{extern int d1,d2,d3,y1new,y2new,y0new;
int d1t=0,d2t=0,d3t=0;
if (y1new == 0 && y2new == 0)
{
     if (d1 == 0 && d2 == 0 && d3 == 0)
     {
     y0new = 0;
     d1t = 0;
     d2t = 0;
     d3t = 0;
     }

     if (d1 == 0 && d2 == 0 && d3 == 1)
     {
     y0new = 1;
     d1t = 1;
     d2t = 0;
     d3t = 0;
     }

     if (d1 == 0 && d2 == 1 && d3 == 0)
     {
     y0new = 0;
     d1t = 0;
     d2t = 0;
     d3t = 1;
     }

     if (d1 == 0 && d2 == 1 && d3 == 1)
     {
     y0new = 1;
     d1t = 1;
     d2t = 1;
     d3t = 1;
     }

     if (d1 == 1 && d2 == 0 && d3 == 0)
     {
     y0new = 0;
     d1t = 0;
     d2t = 1;
     d3t = 0;
     }

     if (d1 == 1 && d2 == 0 && d3 == 1)
```

```
        {
        y0new = 1;
        d1t = 1;
        d2t = 1;
        d3t = 0;
        }

        if (d1 == 1 && d2 == 1 && d3 == 0)
        {
        y0new = 0;
        d1t = 0;
        d2t = 1;
        d3t = 1;
        }

        if (d1 == 1 && d2 == 1 && d3 == 1)
        {
        y0new = 1;
        d1t = 1;
        d2t = 0;
        d3t = 1;
        }
}

if (y1new == 0 && y2new == 1)
{
        if (d1 == 0 && d2 == 0 && d3 == 0)
        {
        y0new = 0;
        d1t = 0;
        d2t = 1;
        d3t = 1;
        }

        if (d1 == 0 && d2 == 0 && d3 == 1)
        {
        y0new = 1;
        d1t = 1;
        d2t = 0;
        d3t = 1;
        }

        if (d1 == 0 && d2 == 1 && d3 == 0)
        {
        y0new = 0;
        d1t = 0;
        d2t = 1;
        d3t = 0;
        }
```

```
    if (d1 == 0 && d2 == 1 && d3 == 1)
    {
    y0new = 1;
    d1t = 1;
    d2t = 1;
    d3t = 0;
    }

    if (d1 == 1 && d2 == 0 && d3 == 0)
    {
    y0new = 0;
    d1t = 0;
    d2t = 0;
    d3t = 1;
    }

    if (d1 == 1 && d2 == 0 && d3 == 1)
    {
    y0new = 1;
    d1t = 1;
    d2t = 1;
    d3t = 1;
    }

    if (d1 == 1 && d2 == 1 && d3 == 0)
    {
    y0new = 0;
    d1t = 0;
    d2t = 0;
    d3t = 0;
    }

    if (d1 == 1 && d2 == 1 && d3 == 1)
    {
    y0new = 1;
    d1t = 1;
    d2t = 0;
    d3t = 0;
    }
}

if (y1new == 1 && y2new == 0)
{
    if (d1 == 0 && d2 == 0 && d3 == 0)
    {
    y0new = 0;
    d1t = 0;
    d2t = 1;
```

```
d3t = 0;
}

if (d1 == 0 && d2 == 0 && d3 == 1)
{
y0new = 1;
d1t = 1;
d2t = 1;
d3t = 1;
}

if (d1 == 0 && d2 == 1 && d3 == 0)
{
y0new = 0;
d1t = 0;
d2t = 1;
d3t = 1;
}

if (d1 == 0 && d2 == 1 && d3 == 1)
{
y0new = 1;
d1t = 1;
d2t = 0;
d3t = 0;
}

if (d1 == 1 && d2 == 0 && d3 == 0)
{
y0new = 0;
d1t = 0;
d2t = 0;
d3t = 0;
}

if (d1 == 1 && d2 == 0 && d3 == 1)
{
y0new = 1;
d1t = 1;
d2t = 0;
d3t = 1;
}

if (d1 == 1 && d2 == 1 && d3 == 0)
{
y0new = 0;
d1t = 0;
d2t = 0;
d3t = 1;
```

```
        }

        if (d1 == 1 && d2 == 1 && d3 == 1)
        {
        y0new = 1;
        d1t = 1;
        d2t = 1;
        d3t = 0;
        }
}

if (y1new == 1 && y2new == 1)
{
        if (d1 == 0 && d2 == 0 && d3 == 0)
        {
        y0new = 0;
        d1t = 0;
        d2t = 0;
        d3t = 1;
        }

        if (d1 == 0 && d2 == 0 && d3 == 1)
        {
        y0new = 1;
        d1t = 1;
        d2t = 1;
        d3t = 0;
        }

        if (d1 == 0 && d2 == 1 && d3 == 0)
        {
        y0new = 0;
        d1t = 0;
        d2t = 0;
        d3t = 0;
        }

        if (d1 == 0 && d2 == 1 && d3 == 1)
        {
        y0new = 1;
        d1t = 1;
        d2t = 0;
        d3t = 1;
        }

        if (d1 == 1 && d2 == 0 && d3 == 0)
        {
        y0new = 0;
        d1t = 0;
```

```
      d2t = 1;
      d3t = 1;
      }

      if (d1 == 1 && d2 == 0 && d3 == 1)
      {
      y0new = 1;
      d1t = 1;
      d2t = 0;
      d3t = 0;
      }

      if (d1 == 1 && d2 == 1 && d3 == 0)
      {
      y0new = 0;
      d1t = 0;
      d2t = 1;
      d3t = 0;
      }

      if (d1 == 1 && d2 == 1 && d3 == 1)
      {
      y0new = 1;
      d1t = 1;
      d2t = 1;
      d3t = 1;
      }
}
d1 = d1t;
d2 = d2t;
d3 = d3t;
y1old = y1new;
y2old = y2new;
return 0;
}
```

```
/* de_diffencoder */
/* FUNCTION DE_DIFFENCODER(BIT_1,BIT_2,DEC1,DEC2) */
de_diffencoder(bit_1,bit_2,dec1,dec2)
int bit_1,bit_2;
int *dec1,*dec2;
{
if (bit_1 == 0 && bit_2 == 0)
{
	if (y1old == 0 && y2old == 0)
	{
		*dec1 = 0;
		*dec2 = 0;
	}
	if (y1old == 0 && y2old == 1)
	{
		*dec1 = 0;
		*dec2 = 1;
	}
	if (y1old == 1 && y2old == 0)
	{
		*dec1 = 1;
		*dec2 = 1;
	}
	if (y1old == 1 && y2old == 1)
	{
		*dec1 = 1;
		*dec2 = 0;
	}
}
if (bit_1 == 0 && bit_2 == 1)
{
	if (y1old == 0 && y2old == 0)
	{
		*dec1 = 0;
		*dec2 = 1;
	}
	if (y1old == 0 && y2old == 1)
	{
		*dec1 = 0;
		*dec2 = 0;
	}
	if (y1old == 1 && y2old == 0)
	{
		*dec1 = 1;
		*dec2 = 0;
	}
	if (y1old == 1 && y2old == 1)
	{
		*dec1 = 1;
```

```
            *dec2 = 1;
      }
}
if (bit_1 == 1 && bit_2 == 0)
{
      if (y1old == 0 && y2old == 0)
      {
            *dec1 = 1;
            *dec2 = 0;
      }
      if (y1old == 0 && y2old == 1)
      {
            *dec1 = 1;
            *dec2 = 1;
      }
      if (y1old == 1 && y2old == 0)
      {
            *dec1 = 0;
            *dec2 = 0;
      }
      if (y1old == 1 && y2old == 1)
      {
            *dec1 = 0;
            *dec2 = 1;
      }
}
if (bit_1 == 1 && bit_2 == 1)
{
      if (y1old == 0 && y2old == 0)
      {
            *dec1 = 1;
            *dec2 = 1;
      }
      if (y1old == 0 && y2old == 1)
      {
            *dec1 = 1;
            *dec2 = 0;
      }
      if (y1old == 1 && y2old == 0)
      {
            *dec1 = 0;
            *dec2 = 1;
      }
      if (y1old == 1 && y2old == 1)
      {
            *dec1 = 0;
            *dec2 = 0;
      }
}
```

```
y1old = bit_1;
y2old = bit_2;
return 0;
}
```

```
/* TOTEXT  */
/* This subroutine takes a bit stream and converts it back
to ASCII text */
/* FUNCTION TOTEXT()*/
totext()
{
FILE *fptr, *fptr2;
char textchar[9];
int decimal;
fptr =fopen("c:\\modem\\binary_o.txt","r");
fptr2 = fopen ("c:\\modem\\text.txt","w");
while( fgets(textchar,9,fptr) != NULL)
{decimal = 0;
 if (textchar[0] == '1')
     decimal = 128;
 if (textchar[1] == '1')
     decimal = decimal + 64;
 if (textchar[2] == '1')
     decimal = decimal + 32;
 if (textchar[3] == '1')
     decimal = decimal + 16;
 if (textchar[4] == '1')
     decimal = decimal + 8;
 if (textchar[5] == '1')
     decimal = decimal + 4;
 if (textchar[6] == '1')
     decimal = decimal + 2;
 if (textchar[7] == '1')
     decimal = decimal + 1;
fprintf(fptr2,"%c",decimal);

       }
fclose(fptr);
fclose(fptr2);
return 0;
       }
```

```
/*    Program Name: v32Modem.c
*/
/*
*/
/*    Developed By: Joseph J. Pisula
*/
/*
*/
/*       Purpose:  To read ascii text files and simulate
*/
/*                 V.32 modem transmissions.  The overall
*/ /*                   purpose is to allow the study
     */
/*                 of noise, decoder, and TCM techniques and
*/ /*                   the effect each has on transmission
     */ /*                    efficiency.
          */ /*
               */
/*
*/
/*    Global Variables:
*/
/*       y0old = y0(n-1), y1old = y1(n-1), y2old = y2(n-1),
*/
/*    y0oldest = y0(n-2), xnew = x1(n), xold = x1(n-1),
*/
/*       y0new = (redundant bit, encoded y0),
*/
/*       y1new = (diffencoded bit q1, encoded y1),
*/
/*       y2new = (diffencoded bit q2, encoded y2),
*/
/*    y3new = (bit 3 of X-bit chunk),
*/
/*    y4new = (bit 4 of X-bit Chunk)
*/
/*
*/
/*
*/
#include <math.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
/*FUNCTION
BINARYREADER32(VIT_MEM_LENGTH,ENCODER_TYPE,NOISE,TFILE,RFILE
)*/
/*binaryreader32*/
/*
```

```
*/
/*   THIS IS THE FUNCTION THAT CONTROLS V.33 TRANSMISSIONS
*/
/*   VIT_MEM_LENGTH = VITERBI_MEMORY_LENGTH
*/
/*   ENCODER_TYPE = (1 OR 2) TO SPECIFY DYNAMIC OR LOOK-
UP   */
/*   NOISE = NOISE_REDUCTION_FACTOR USED
*/
/*   TFILE = PATH/NAME OF TEXT FILE TO OPERATE AGAINST
*/
/*   RFILE = PATH/NAME OF REPORT FILE TO GENERATE
*/
/*   MY_XY = STRUCTURE OF TRANSMITTED SIGNAL POINTS
*/
/*   PATH_OLD = AN ARRAY OF STRUCTURES TO TRACK PATH
HISYORY*/
/*   PATH_NEW = AN ARRAY OF STRUCTURES TO TRACK PATH
HISTORY*/
/*   V33_TABLE = AN ARRAY OF STRUCTURES FOR S/P LOOK-UP
*/
/*   FROM_NODEM = AN ARRAY THAT TRACKS WHAT STATE THAT THE
*/
/*                CURRENT STATE CAME FROM
*/
/*   DECODED = ARRAY THAT TRACKS CURRENT SYMBOL FOR
DECODING*/
/*   BEST_PATH = AN ARRAY TO STORE THE NEXT NODE IN PATH
*/
/*   TO_D_O = AN ARRAY TO HOLD TOTAL DISCREP FOR LAST STATE
*/
/*   TO_D_N = AN ARRAY TO HOLD TOTAL DISCREP FOR NEXT STATE
*/
/*   MIN_DISCREP = MINIMUM DISCREP TO LAST STATE
*/
/*   OUTPUT =THE STRING USED TO TAKE SYMBOLS FROM
BINARY.TXT*/
binaryreader32(vit_mem_length,encoder_type,noise,tfile,rfile
)
float noise;
char encoder_type,tfile[25],rfile[25];
int vit_mem_length;
{extern float x,y;
extern int
y0new,y1new,y2new,y3new,y4new,y1old,y2old,xnew,xold,d1,d2,d3
,y0oldest;
struct my_xy {
 float xval;
 float yval;
```

```
} xy[3000];
struct path1 {
int path_o[50];
} path_old[32];/*stores the path up to previous node*/
struct path2{
int path_c[50];
} path_new[32];/*stores the current path to node (index)*/
struct v32 {
float ycoord;
float xcoord;
int bit_1;
int bit_2;
int bit_3;
int bit_4;
int bit_5;
int index; }
v32_table[32];
int numxy = 0;
int maxxy = 3000;
int j,bit,iteration_count=0,zero_pad;
int  q1, q2,dec1,dec2;
int ct = 0, eoffnd = 0;
int from_node[32];
int i,l,poss_symbol;
int state_to,current,state_from_index,demod_fault=0;
int best_path[32];/*stores previous node value, array index
is current node*/
int decoded[32];
float tot_d_o[32];/*stores total discrepancy for that path*/
float tot_d_n[32];
float discrep;
float min_discrep[32];
float u1,u2,r,a,grv1,grv2,rvtot1=0,rvtot2=0;
float xv,yv;
char output[6];
FILE *fptr,*fptr2;
fptr = fopen("c:\\modem\\binary.txt","r");
fptr2 = fopen("c:\\modem\\binary_o.txt","w");
randomize();
for (i=0; i<32; i++)
tot_d_o[i] = 0.0;
for (i=0; i<32; i++)
tot_d_n[i] = 0.0;
for (i=0; i<32; i++)
min_discrep[i] = 30000;
for (i=0; i<32; i++)
for (l=0; l<vit_mem_length; l++)
path_old[i].path_o[l] = 0;
for (i=0; i<32; i++)
```

```
for (l=0; l<vit_mem_length; l++)
path_new[i].path_c[l] = 0;

/* THIS LOADS THE V.32 SIGNAL CONSTELLATION TO MEMORY  */
/* V32 Look-up Table */
v32_table[0].xcoord = 2.250  *      -4.0;
v32_table[0].ycoord = 2.250  *      1.0;
v32_table[0].bit_1 = 0;
v32_table[0].bit_2 = 0;
v32_table[0].bit_3 = 0;
v32_table[0].bit_4 = 0;
v32_table[0].bit_5 = 0;
v32_table[0].index = 0;

v32_table[1].xcoord = 2.250  *      0.0;
v32_table[1].ycoord = 2.250  *      -3.0;
v32_table[1].bit_1 =  0;
v32_table[1].bit_2 =  0;
v32_table[1].bit_3 =  0;
v32_table[1].bit_4 =  0;
v32_table[1].bit_5 =  1;
v32_table[1].index =  1;

v32_table[2].xcoord = 2.250  *      0.0;
v32_table[2].ycoord = 2.250  *      1.0;
v32_table[2].bit_1 = 0;
v32_table[2].bit_2 = 0;
v32_table[2].bit_3 = 0;
v32_table[2].bit_4 = 1;
v32_table[2].bit_5 = 0;
v32_table[2].index = 2;

v32_table[3].xcoord = 2.250  *      4.0;
v32_table[3].ycoord = 2.250  *      1.0;
v32_table[3].bit_1 = 0;
v32_table[3].bit_2 = 0;
v32_table[3].bit_3 = 0;
v32_table[3].bit_4 = 1;
v32_table[3].bit_5 = 1;
v32_table[3].index = 3;

v32_table[4].xcoord = 2.250  *      4.0;
v32_table[4].ycoord = 2.250  *      -1.0;
v32_table[4].bit_1 = 0;
v32_table[4].bit_2 = 0;
v32_table[4].bit_3 = 1;
v32_table[4].bit_4 = 0;
v32_table[4].bit_5 = 0;
v32_table[4].index = 4;
```

```
v32_table[5].xcoord = 2.250   *       0.0;
v32_table[5].ycoord = 2.250   *       3.0;
v32_table[5].bit_1 = 0;
v32_table[5].bit_2 = 0;
v32_table[5].bit_3 = 1;
v32_table[5].bit_4 = 0;
v32_table[5].bit_5 = 1;
v32_table[5].index = 5;

v32_table[6].xcoord = 2.250   *       0.0;
v32_table[6].ycoord = 2.250   *       -1.0;
v32_table[6].bit_1 =  0;
v32_table[6].bit_2 =  0;
v32_table[6].bit_3 =  1;
v32_table[6].bit_4 =  1;
v32_table[6].bit_5 =  0;
v32_table[6].index =  6;

v32_table[7].xcoord = 2.250   *       -4.0;
v32_table[7].ycoord = 2.250   *       -1.0;
v32_table[7].bit_1 = 0;
v32_table[7].bit_2 = 0;
v32_table[7].bit_3 = 1;
v32_table[7].bit_4 = 1;
v32_table[7].bit_5 = 1;
v32_table[7].index = 7;

v32_table[8].xcoord = 2.250   *       -2.0;
v32_table[8].ycoord = 2.250   *       3.0;
v32_table[8].bit_1 = 0;
v32_table[8].bit_2 = 1;
v32_table[8].bit_3 = 0;
v32_table[8].bit_4 = 0;
v32_table[8].bit_5 = 0;
v32_table[8].index = 8;

v32_table[9].xcoord = 2.250   *       -2.0;
v32_table[9].ycoord = 2.250   *       -1.0;
v32_table[9].bit_1 = 0;
v32_table[9].bit_2 = 1;
v32_table[9].bit_3 = 0;
v32_table[9].bit_4 = 0;
v32_table[9].bit_5 = 1;
v32_table[9].index = 9;

v32_table[10].xcoord = 2.250   *      2.0;
v32_table[10].ycoord = 2.250   *      3.0;
v32_table[10].bit_1 = 0;
```

```
v32_table[10].bit_2 = 1;
v32_table[10].bit_3 = 0;
v32_table[10].bit_4 = 1;
v32_table[10].bit_5 = 0;
v32_table[10].index = 10;

v32_table[11].xcoord = 2.250  *      2.0;
v32_table[11].ycoord = 2.250  *      -1.0;
v32_table[11].bit_1 = 0;
v32_table[11].bit_2 = 1;
v32_table[11].bit_3 = 0;
v32_table[11].bit_4 = 1;
v32_table[11].bit_5 = 1;
v32_table[11].index = 11;

v32_table[12].xcoord = 2.250  *      2.0;
v32_table[12].ycoord = 2.250  *      -3.0;
v32_table[12].bit_1 =  0;
v32_table[12].bit_2 =  1;
v32_table[12].bit_3 =  1;
v32_table[12].bit_4 =  0;
v32_table[12].bit_5 =  0;
v32_table[12].index =  12;

v32_table[13].xcoord = 2.250  *      2.0;
v32_table[13].ycoord = 2.250  *      1.0;
v32_table[13].bit_1 = 0;
v32_table[13].bit_2 = 1;
v32_table[13].bit_3 = 1;
v32_table[13].bit_4 = 0;
v32_table[13].bit_5 = 1;
v32_table[13].index = 13;

v32_table[14].xcoord = 2.250  *      -2.0;
v32_table[14].ycoord = 2.250  *      -3.0;
v32_table[14].bit_1 = 0;
v32_table[14].bit_2 = 1;
v32_table[14].bit_3 = 1;
v32_table[14].bit_4 = 1;
v32_table[14].bit_5 = 0;
v32_table[14].index = 14;

v32_table[15].xcoord = 2.250  *         -2.0;
v32_table[15].ycoord = 2.250  *         1.0;
v32_table[15].bit_1 = 0;
v32_table[15].bit_2 = 1;
v32_table[15].bit_3 = 1;
v32_table[15].bit_4 = 1;
v32_table[15].bit_5 = 1;
```

```
v32_table[15].index = 15;

v32_table[16].xcoord = 2.250   *      -3.0;
v32_table[16].ycoord = 2.250   *      -2.0;
v32_table[16].bit_1 = 1;
v32_table[16].bit_2 = 0;
v32_table[16].bit_3 = 0;
v32_table[16].bit_4 = 0;
v32_table[16].bit_5 = 0;
v32_table[16].index = 16;

v32_table[17].xcoord = 2.250   *      1.0;
v32_table[17].ycoord = 2.250   *      -2.0;
v32_table[17].bit_1 = 1;
v32_table[17].bit_2 = 0;
v32_table[17].bit_3 = 0;
v32_table[17].bit_4 = 0;
v32_table[17].bit_5 = 1;
v32_table[17].index = 17;

v32_table[18].xcoord = 2.250   *      -3.0;
v32_table[18].ycoord = 2.250   *      2.0;
v32_table[18].bit_1 = 1;
v32_table[18].bit_2 = 0;
v32_table[18].bit_3 = 0;
v32_table[18].bit_4 = 1;
v32_table[18].bit_5 = 0;
v32_table[18].index = 18;

v32_table[19].xcoord = 2.250   *      1.0;
v32_table[19].ycoord = 2.250   *      2.0;
v32_table[19].bit_1 = 1;
v32_table[19].bit_2 = 0;
v32_table[19].bit_3 = 0;
v32_table[19].bit_4 = 1;
v32_table[19].bit_5 = 1;
v32_table[19].index = 19;

v32_table[20].xcoord = 2.250   *      3.0;
v32_table[20].ycoord = 2.250   *      2.0;
v32_table[20].bit_1 = 1;
v32_table[20].bit_2 = 0;
v32_table[20].bit_3 = 1;
v32_table[20].bit_4 = 0;
v32_table[20].bit_5 = 0;
v32_table[20].index = 20;

v32_table[21].xcoord = 2.250   *      -1.0;
v32_table[21].ycoord = 2.250   *      2.0;
```

```
v32_table[21].bit_1 = 1;
v32_table[21].bit_2 = 0;
v32_table[21].bit_3 = 1;
v32_table[21].bit_4 = 0;
v32_table[21].bit_5 = 1;
v32_table[21].index = 21;

v32_table[22].xcoord = 2.250   *       3.0;
v32_table[22].ycoord = 2.250   *       -2.0;
v32_table[22].bit_1 = 1;
v32_table[22].bit_2 = 0;
v32_table[22].bit_3 = 1;
v32_table[22].bit_4 = 1;
v32_table[22].bit_5 = 0;
v32_table[22].index = 22;

v32_table[23].xcoord = 2.250   *       -1.0;
v32_table[23].ycoord = 2.250   *       -2.0;
v32_table[23].bit_1 = 1;
v32_table[23].bit_2 = 0;
v32_table[23].bit_3 = 1;
v32_table[23].bit_4 = 1;
v32_table[23].bit_5 = 1;
v32_table[23].index = 23;

v32_table[24].xcoord = 2.250   *       1.0;
v32_table[24].ycoord = 2.250   *       4.0;
v32_table[24].bit_1 = 1;
v32_table[24].bit_2 = 1;
v32_table[24].bit_3 = 0;
v32_table[24].bit_4 = 0;
v32_table[24].bit_5 = 0;
v32_table[24].index = 24;

v32_table[25].xcoord = 2.250   *       -3.0;
v32_table[25].ycoord = 2.250   *       0.0;
v32_table[25].bit_1 = 1;
v32_table[25].bit_2 = 1;
v32_table[25].bit_3 = 0;
v32_table[25].bit_4 = 0;
v32_table[25].bit_5 = 1;
v32_table[25].index = 25;

v32_table[26].xcoord = 2.250   *       1.0;
v32_table[26].ycoord = 2.250   *       0.0;
v32_table[26].bit_1 = 1;
v32_table[26].bit_2 = 1;
v32_table[26].bit_3 = 0;
v32_table[26].bit_4 = 1;
```

```
v32_table[26].bit_5 = 0;
v32_table[26].index = 26;

v32_table[27].xcoord = 2.250  *      1.0;
v32_table[27].ycoord = 2.250  *      -4.0;
v32_table[27].bit_1 = 1;
v32_table[27].bit_2 = 1;
v32_table[27].bit_3 = 0;
v32_table[27].bit_4 = 1;
v32_table[27].bit_5 = 1;
v32_table[27].index = 27;

v32_table[28].xcoord = 2.250  *      -1.0;
v32_table[28].ycoord = 2.250  *      -4.0;
v32_table[28].bit_1 = 1;
v32_table[28].bit_2 = 1;
v32_table[28].bit_3 = 1;
v32_table[28].bit_4 = 0;
v32_table[28].bit_5 = 0;
v32_table[28].index = 28;

v32_table[29].xcoord = 2.250  *      3.0;
v32_table[29].ycoord = 2.250  *      0.0;
v32_table[29].bit_1 = 1;
v32_table[29].bit_2 = 1;
v32_table[29].bit_3 = 1;
v32_table[29].bit_4 = 0;
v32_table[29].bit_5 = 1;
v32_table[29].index = 29;

v32_table[30].xcoord = 2.250  *      -1.0;
v32_table[30].ycoord = 2.250  *      0.0;
v32_table[30].bit_1 = 1;
v32_table[30].bit_2 = 1;
v32_table[30].bit_3 = 1;
v32_table[30].bit_4 = 1;
v32_table[30].bit_5 = 0;
v32_table[30].index = 30;

v32_table[31].xcoord = 2.250  *      -1.0;
v32_table[31].ycoord = 2.250  *      4.0;
v32_table[31].bit_1 = 1;
v32_table[31].bit_2 = 1;
v32_table[31].bit_3 = 1;
v32_table[31].bit_4 = 1;
v32_table[31].bit_5 = 1;
v32_table[31].index = 31;
/* Begin Reading binary.txt, the binary file to be
transmitted*/
```

```
while (!eoffnd)
{
  ct=0;
  while( ((ct < 4) && (output[ct]=getc(fptr)) != EOF) )
  {
       ct++;
  }
  if(ct != 4)          /* got to eof before we read four
bits */
  {
    eoffnd = 1;
    while(ct< 4)
    {
      output[ct] = '0';
      ct++;
     }
  }
  for(ct=0; ct<4; ct++)
  {
    if (output[ct] == EOF)
      eoffnd = 1;
   }
   q1 = output[0]-48;
   q2 = output[1]-48;
   y3new = output[2]-48;
   y4new = output[3]-48;

/* DIFFERENTIALLY ENCODE FIRST TWO BITS IN TIME        */
     diffencoder(q1, q2);

/* CONVOLUTIONALLY ENCODE THE DIFF ENCODED BITS        */
if (strcmp(encoder_type,"1")==0)
     encoder();
else
     if (strcmp(encoder_type,"2")==0)
          encoder2();
     else
     {    printf("\n**Invalid Encoder Type (Argument
3)**\n");
          exit(1);
     }

/* MAP THE ENCODED SYMBOL THE V.32 SIGNAL CONSTELLATION */
     mappervdot32(&x,&y);

     if(numxy < maxxy)
   {
       xy[numxy].xval = x;
       xy[numxy].yval = y;
```

```
        numxy++;
    }
    else
    printf("Too many xy bits for array\n");

/*channel noise*/
/* GENERATE GRVs                    */
u1 = ((float)random(20000))/20000.0;
u2 = ((float)random(20000))/20000.0;
r=sqrt(-2*log(u1));
a = 2*3.1415926*u2;
grv1 = r*sin(a)/(float)noise;
grv2 = r*cos(a)/(float)noise;
rvtot1 = rvtot1 + sqrt(grv1*grv1);
rvtot2 = rvtot2 + sqrt(grv2*grv2);
/* INJECT NOISE INTO TRANSMIT ARRAY     */
xy[numxy-1].xval = grv1+xy[numxy-1].xval;
xy[numxy-1].yval = grv2+xy[numxy-1].yval;

}
/* ZERO PAD TRANSMIT ARRAY TO ACCOUNT FOR MEMORY LENGTH
*/
for (zero_pad = 0; zero_pad <vit_mem_length; zero_pad++)
{xy[numxy + zero_pad].xval = 0;
xy[numxy + zero_pad].yval = 0;}
y1old = 0;
y2old = 0;
fclose(fptr);

/*decoder */
/* LOOP FOR EACH SYMBOL TRANSMITTED                    */
/* SAME PROCESS AS IN BINARY_READER33 BIT WITH 32 STATES
*/
for (current = 0; current<numxy-1+vit_mem_length; current++)
{iteration_count++;
for (state_to = 0; state_to <16; state_to++)
{
    for ( poss_symbol=0; poss_symbol<16; poss_symbol++)
    {
    discrep = sqrt((xy[current].xval -
v32_table[poss_symbol].xcoord)*(xy[current].xval -
v32_table[poss_symbol].xcoord)
    + (xy[current].yval -
v32_table[poss_symbol].ycoord)*(xy[current].yval -
v32_table[poss_symbol].ycoord));

state_fromv32(poss_symbol,state_to,&state_from_index);

     if ((discrep+tot_d_o[state_from_index]) > 29999)
```

```
     {
     printf("Discrepancy Has Exceeded 30000");
     exit(1);
   }
     if
((discrep+tot_d_o[state_from_index])<=min_discrep[state_to])
     {    min_discrep[state_to] =
(discrep+tot_d_o[state_from_index]);
     best_path[state_to] = poss_symbol;
     from_node[state_to] = state_from_index;
     }
   }



   tot_d_n[state_to] = min_discrep[state_to];
}

for (state_to = 16; state_to <32; state_to++)
{

   for ( poss_symbol=16; poss_symbol<32; poss_symbol++)
   {
   discrep = sqrt((xy[current].xval -
v32_table[poss_symbol].xcoord)*(xy[current].xval -
v32_table[poss_symbol].xcoord)
   + (xy[current].yval -
v32_table[poss_symbol].ycoord)*(xy[current].yval -
v32_table[poss_symbol].ycoord));

state_fromv32(poss_symbol,state_to,&state_from_index);

     if ((discrep+tot_d_o[state_from_index]) > 29999)
     {
     printf("Discrepancy Has Exceeded 30000");
     exit(1);
   }

   if
((discrep+tot_d_o[state_from_index])<=min_discrep[state_to])
     {    min_discrep[state_to] =
(discrep+tot_d_o[state_from_index]);
     best_path[state_to] = poss_symbol;
     from_node[state_to] = state_from_index;
     }
   }


   tot_d_n[state_to] = min_discrep[state_to];
```

```
}
for (i=0; i<32; i++)
tot_d_o[i] = tot_d_n[i];
for (i=0; i<32; i++)
min_discrep[i] = 30000;
for (i=0; i<32; i++)
{
     for (l=0; l<vit_mem_length; l++)
     path_new[i].path_c[l] =
path_old[from_node[i]].path_o[l];
   decoded[i] = path_new[i].path_c[0];
   for (j=1; j<vit_mem_length; j++)
   {
      path_new[i].path_c[j-1] = path_new[i].path_c[j];
   }
   path_new[i].path_c[vit_mem_length-1] = best_path[i];
}
for (i=0; i<32; i++)
     for (l=0; l<vit_mem_length; l++)
     path_old[i].path_o[l] = path_new[i].path_c[l];

if
((decoded[0]==decoded[1])&&(decoded[2]==decoded[3])&&(decode
d[0]==decoded[2])

&&(decoded[4]==decoded[5])&&(decoded[6]==decoded[7])&&(decod
ed[4]==decoded[6])

&&(decoded[8]==decoded[9])&&(decoded[10]==decoded[11])&&(dec
oded[8]==decoded[10])

&&(decoded[12]==decoded[13])&&(decoded[14]==decoded[15])&&(d
ecoded[12]==decoded[14])

&&(decoded[16]==decoded[17])&&(decoded[18]==decoded[19])&&(d
ecoded[16]==decoded[18])

&&(decoded[20]==decoded[21])&&(decoded[22]==decoded[23])&&(d
ecoded[20]==decoded[22])

&&(decoded[24]==decoded[25])&&(decoded[26]==decoded[27])&&(d
ecoded[24]==decoded[26])

&&(decoded[28]==decoded[29])&&(decoded[30]==decoded[31])&&(d
ecoded[28]==decoded[30])

&&(decoded[0]==decoded[4])&&(decoded[4]==decoded[8])&&(decod
ed[12]==decoded[8])
```

```
&&(decoded[16]==decoded[12])&&(decoded[20]==decoded[16])&&(d
ecoded[24]==decoded[20])
      &&(decoded[28]==decoded[24])&&(iteration_count
>vit_mem_length))
{

de_diffencoder(v32_table[decoded[0]].bit_2,v32_table[decoded
[0]].bit_3,&dec1,&dec2);
      fprintf(fptr2,"%d%d%d%d",dec1,dec2,

v32_table[decoded[0]].bit_4,v32_table[decoded[0]].bit_5);
}
   else if(iteration_count>vit_mem_length)
   { printf("\n**Demodulation Fault**\n");

de_diffencoder(v32_table[decoded[0]].bit_2,v32_table[decoded
[0]].bit_3,&dec1,&dec2);
      fprintf(fptr2,"%d%d%d%d",dec1,dec2,

v32_table[decoded[0]].bit_4,v32_table[decoded[0]].bit_5);
   demod_fault++;
      }

}
fclose(fptr2);
report32(vit_mem_length,noise,rvtot1,rvtot2,demod_fault,numx
y,tfile,rfile);
return 0;
}
```

```
/*mappervdot33() */
/* THIS FUNCTION MAPS THE SYMBOLS TO SIGNAL POINTS FOR
*/
/* THE V.33 TRANSMISSION SCHEME
*/
/* FUNCTION MAPPERVDOT33(X,Y) */
mappervdot33(x,y)
float *x,*y;
{extern y0new,y1new,y2new,y3new,y4new,y5new,y6new;
float  dummy;
 if (y0new==0 && y1new==0 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 0)
  {
     *x  = -8;
     *y  = -3;
  }
 if (y0new==0 && y1new==0 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 1)
  {
     *x  = 9;
     *y  = 2;
  }
 if (y0new==0 && y1new==0 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 0)
  {
     *x  = 2;
     *y  = -9;
  }
 if (y0new==0 && y1new==0 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 1)
  {
     *x  = -3;
     *y  = 8;
  }
  if (y0new==0 && y1new==0 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 0)
  {
     *x  = 8;
     *y  = 3;
  }
  if (y0new==0 && y1new==0 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 1)
  {
     *x  = -9;
     *y  = -2;
  }
  if (y0new==0 && y1new==0 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 0)
  {
```

```
      *x   = -2;
      *y   =  9;
   }
   if (y0new==0 && y1new==0 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 1)
   {
      *x   =  3;
      *y   = -8;
   }
   if (y0new==0 && y1new==0 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 0)
   {
      *x   = -8;
      *y   =  1;
   }
   if (y0new==0 && y1new==0 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 1)
   {
      *x   =  9;
      *y   = -2;
   }
   if (y0new==0 && y1new==0 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 0)
   {
      *x   = -2;
      *y   = -9;
   }
   if (y0new==0 && y1new==0 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 1)
   {
      *x   =  1;
      *y   =  8;
   }
   if (y0new==0 && y1new==0 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 0)
   {
      *x   =  8;
      *y   = -1;
   }
   if (y0new==0 && y1new==0 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 1)
   {
      *x   = -9;
      *y   =  2;
   }
   if (y0new==0 && y1new==0 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 0)
   {
      *x   =  2;
```

```
      *y  =  9;
   }
   if (y0new==0 && y1new==0 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 1)
   {
      *x  = -1;
      *y  = -8;
   }
   if (y0new==0 && y1new==0 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 0)
   {
      *x  = -4;
      *y  = -3;
   }
   if (y0new==0 && y1new==0 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 1)
   {
      *x  =  5;
      *y  =  2;
   }
   if (y0new==0 && y1new==0 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 0)
   {
      *x  =  2;
      *y  = -5;
   }
   if (y0new==0 && y1new==0 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 1)
   {
      *x  = -3;
      *y  =  4;
   }
   if (y0new==0 && y1new==0 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 0)
   {
      *x  =  4;
      *y  =  3;
   }
   if (y0new==0 && y1new==0 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 1)
   {
      *x  = -5;
      *y  = -2;
   }
    if (y0new==0 && y1new==0 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 0)
   {
      *x  = -2;
      *y  =  5;
```

```
   }
   if (y0new==0 && y1new==0 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 1)
   {
      *x  =  3;
      *y  = -4;
   }
   if (y0new==0 && y1new==0 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 0)
   {
      *x  = -4;
      *y  =  1;
   }
   if (y0new==0 && y1new==0 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 1)
   {
      *x  =  5;
      *y  = -2;
   }
   if (y0new==0 && y1new==0 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 0)
   {
      *x  = -2;
      *y  = -5;
   }
   if (y0new==0 && y1new==0 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 1)
   {
      *x  =  1;
      *y  =  4;
   }
   if (y0new==0 && y1new==0 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 0)
   {
      *x  =  4;
      *y  = -1;
   }
   if (y0new==0 && y1new==0 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 1)
   {
      *x  = -5;
      *y  =  2;
   }
   if (y0new==0 && y1new==0 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 0)
   {
      *x  =  2;
      *y  =  5;
   }
```

```
  if (y0new==0 && y1new==0 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 1)
  {
     *x  = -1;
     *y  = -4;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 0)
  {
     *x  =  4;
     *y  = -3;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 1)
  {
     *x  = -3;
     *y  = 2;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 0)
  {
     *x  =  2;
     *y  =  3;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 1)
  {
     *x  = -3;
     *y  = -4;
  }
   if (y0new==0 && y1new==1 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 0)
  {
     *x  = -4;
     *y  =  3;
  }
   if (y0new==0 && y1new==1 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 1)
  {
     *x  =  3;
     *y  = -2;
  }
   if (y0new==0 && y1new==1 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 0)
  {
     *x  = -2;
     *y  = -3;
  }
   if (y0new==0 && y1new==1 && y2new==0 && y3new==0 &&
```

```
y4new==1 && y5new ==1 && y6new == 1)
  {
     *x  =  3;
     *y  =  4;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 0)
  {
     *x  =  4;
     *y  =  1;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 1)
  {
     *x  = -3;
     *y  = -2;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 0)
  {
     *x  = -2;
     *y  =  3;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 1)
  {
     *x  =  1;
     *y  = -4;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 0)
  {
     *x  = -4;
     *y  = -1;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 1)
  {
     *x  =  3;
     *y  =  2;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 0)
  {
     *x  =  2;
     *y  = -3;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 1)
```

```
   {
      *x  = -1;
      *y  =  4;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 0)
   {
      *x  =  0;
      *y  = -3;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 1)
   {
      *x  =  1;
      *y  =  2;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 0)
   {
      *x  =  2;
      *y  = -1;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 1)
   {
      *x  = -3;
      *y  =  0;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 0)
   {
      *x  =  0;
      *y  =  3;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 1)
   {
      *x  = -1;
      *y  = -2;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 0)
   {
      *x  = -2;
      *y  =  1;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 1)
   {
```

```
      *x  =  3;
      *y  =  0;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 0)
   {
      *x  =  0;
      *y  =  1;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 1)
   {
      *x  =  1;
      *y  = -2;
   }
    if (y0new==0 && y1new==1 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 0)
   {
      *x  = -2;
      *y  = -1;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 1)
   {
      *x  =  1;
      *y  =  0;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 0)
   {
      *x  =  0;
      *y  = -1;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 1)
   {
      *x  = -1;
      *y  =  2;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 0)
   {
      *x  =  2;
      *y  =  1;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 1)
   {
      *x  = -1;
```

```
      *y   =  0;
   }
   if (y0new==1 && y1new==0 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 0)
   {
      *x   =  8;
      *y   = -3;
   }
   if (y0new==1 && y1new==0 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 1)
   {
      *x   = -7;
      *y   = 2;
   }
   if (y0new==1 && y1new==0 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 0)
   {
      *x   =  2;
      *y   =  7;
   }
   if (y0new==1 && y1new==0 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 1)
   {
      *x   = -3;
      *y   = -8;
   }
    if (y0new==1 && y1new==0 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 0)
   {
      *x   = -8;
      *y   =  3;
   }
    if (y0new==1 && y1new==0 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 1)
   {
      *x   =  7;
      *y   = -2;
   }
    if (y0new==1 && y1new==0 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 0)
   {
      *x   = -2;
      *y   = -7;
   }
    if (y0new==1 && y1new==0 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 1)
   {
      *x   =  3;
      *y   =  8;
```

```
    }
     if (y0new==1 && y1new==0 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 0)
    {
        *x  =  8;
        *y  =  1;
    }
    if (y0new==1 && y1new==0 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 1)
    {
        *x  = -7;
        *y  = -2;
    }
    if (y0new==1 && y1new==0 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 0)
    {
        *x  = -2;
        *y  =  7;
    }
    if (y0new==1 && y1new==0 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 1)
    {
        *x  =  1;
        *y  = -8;
    }
    if (y0new==1 && y1new==0 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 0)
    {
        *x  = -8;
        *y  = -1;
    }
    if (y0new==1 && y1new==0 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 1)
    {
        *x  =  7;
        *y  =  2;
    }
    if (y0new==1 && y1new==0 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 0)
    {
        *x  =  2;
        *y  = -7;
    }
    if (y0new==1 && y1new==0 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 1)
    {
        *x  = -1;
        *y  =  8;
    }
```

```
  if (y0new==1 && y1new==0 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 0)
  {
     *x  = -4;
     *y  = -7;
  }
        if (y0new==1 && y1new==0 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 1)
  {
     *x  =  5;
     *y  =  6;
  }
  if (y0new==1 && y1new==0 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 0)
  {
     *x  =  6;
     *y  = -5;
  }
  if (y0new==1 && y1new==0 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 1)
  {
     *x  = -7;
     *y  =  4;
  }
  if (y0new==1 && y1new==0 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 0)
  {
     *x  =  4;
     *y  =  7;
  }
  if (y0new==1 && y1new==0 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 1)
  {
     *x  = -5;
     *y  = -6;
  }
  if (y0new==1 && y1new==0 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 0)
  {
     *x  = -6;
     *y  =  5;
  }
  if (y0new==1 && y1new==0 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 1)
  {
     *x  =  7;
     *y  = -4;
  }
  if (y0new==1 && y1new==0 && y2new==1 && y3new==1 &&
```

```
y4new==0 && y5new ==0 && y6new == 0)
  {
     *x  = -4;
     *y  =  5;
  }
  if (y0new==1 && y1new==0 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 1)
  {
     *x  =  5;
     *y  = -6;
  }
  if (y0new==1 && y1new==0 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 0)
  {
     *x  = -6;
     *y  = -5;
  }
  if (y0new==1 && y1new==0 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 1)
  {
     *x  =  5;
     *y  =  4;
  }
  if (y0new==1 && y1new==0 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 0)
  {
     *x  =  4;
     *y  = -5;
  }
  if (y0new==1 && y1new==0 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 1)
  {
     *x  = -5;
     *y  =  6;
  }
  if (y0new==1 && y1new==0 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 0)
  {
     *x  =  6;
     *y  =  5;
  }
  if (y0new==1 && y1new==0 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 1)
  {
     *x  = -5;
     *y  = -4;
  }
  if (y0new==1 && y1new==1 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 0)
```

```
  {
     *x  =  4;
     *y  = -7;
  }
  if (y0new==1 && y1new==1 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 1)
  {
     *x  = -3;
     *y  = 6;
  }
  if (y0new==1 && y1new==1 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 0)
  {
     *x  =  6;
     *y  =  3;
  }
  if (y0new==1 && y1new==1 && y2new==0 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 1)
  {
     *x  = -7;
     *y  = -4;
  }
   if (y0new==1 && y1new==1 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 0)
  {
     *x  = -4;
     *y  =  7;
  }
  if (y0new==1 && y1new==1 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 1)
  {
     *x  =  3;
     *y  = -6;
  }
  if (y0new==1 && y1new==1 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 0)
  {
     *x  = -6;
     *y  = -3;
  }
   if (y0new==1 && y1new==1 && y2new==0 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 1)
  {
     *x  =  7;
     *y  =  4;
  }
   if (y0new==1 && y1new==1 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 0)
  {
```

```
      *x  =  4;
      *y  =  5;
   }
    if (y0new==1 && y1new==1 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 1)
   {
      *x  = -3;
      *y  = -6;
   }
    if (y0new==1 && y1new==1 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 0)
   {
      *x  = -6;
      *y  =  3;
   }
    if (y0new==1 && y1new==1 && y2new==0 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 1)
   {
      *x  =  5;
      *y  = -4;
   }
    if (y0new==1 && y1new==1 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 0)
   {
      *x  = -4;
      *y  = -5;
   }
    if (y0new==1 && y1new==1 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 1)
   {
      *x  =  3;
      *y  =  6;
   }
   if (y0new==1 && y1new==1 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 0)
   {
      *x  =  6;
      *y  = -3;
   }
    if (y0new==1 && y1new==1 && y2new==0 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 1)
   {
      *x  = -5;
      *y  =  4;
   }
    if (y0new==1 && y1new==1 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 0)
   {
      *x  =  0;
```

```
      *y   = -7;
   }
   if (y0new==1 && y1new==1 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==0 && y6new == 1)
   {
      *x   =  1;
      *y   =  6;
   }
   if (y0new==1 && y1new==1 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 0)
   {
      *x   =  6;
      *y   = -1;
   }
   if (y0new==1 && y1new==1 && y2new==1 && y3new==0 &&
y4new==0 && y5new ==1 && y6new == 1)
   {
      *x   = -7;
      *y   =  0;
   }
   if (y0new==1 && y1new==1 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 0)
   {
      *x   =  0;
      *y   =  7;
   }
   if (y0new==1 && y1new==1 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==0 && y6new == 1)
   {
      *x   = -1;
      *y   = -6;
   }
   if (y0new==1 && y1new==1 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 0)
   {
      *x   = -6;
      *y   =  1;
   }
   if (y0new==1 && y1new==1 && y2new==1 && y3new==0 &&
y4new==1 && y5new ==1 && y6new == 1)
   {
      *x   =  7;
      *y   =  0;
   }
   if (y0new==1 && y1new==1 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 0)
   {
      *x   =  0;
      *y   =  5;
```

```
  }
  if (y0new==1 && y1new==1 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==0 && y6new == 1)
  {
     *x  =  1;
     *y  = -6;
  }
   if (y0new==1 && y1new==1 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 0)
  {
     *x  = -6;
     *y  = -1;
  }
  if (y0new==1 && y1new==1 && y2new==1 && y3new==1 &&
y4new==0 && y5new ==1 && y6new == 1)
  {
     *x  =  5;
     *y  =  0;
  }
  if (y0new==1 && y1new==1 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 0)
  {
     *x  =  0;
     *y  = -5;
  }
  if (y0new==1 && y1new==1 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==0 && y6new == 1)
  {
     *x  = -1;
     *y  =  6;
  }
  if (y0new==1 && y1new==1 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 0)
  {
     *x  =  6;
     *y  =  1;
  }
  if (y0new==1 && y1new==1 && y2new==1 && y3new==1 &&
y4new==1 && y5new ==1 && y6new == 1)
  {
     *x  = -5;
     *y  =  0;
  }
       return 0;
 }
```

```
/*mappervdot32() */
/* THIS FUNCTION MAPS THE SYMBOLS TO SIGNAL POINTS FOR
*/
/* V.32 TRANSMISSIONS
*/
/* FUNCTION MAPPERVDOT32(X,Y) */
mappervdot32(x,y)
float *x,*y;
{extern int y0new,y1new,y2new,y3new,y4new;

float  dummy;
  if (y0new==0 && y1new==0 && y2new==0 && y3new==0 &&
y4new==0)
  {
     *x = 2.250  * -4.0;
     *y = 2.250  * 1.0;
  }
  if (y0new==0 && y1new==0 && y2new==0 && y3new==0 &&
y4new==1)
  {
     *x = 2.250  * 0.0;
     *y = 2.250  * -3.0;
  }
  if (y0new==0 && y1new==0 && y2new==0 && y3new==1 &&
y4new==0)
  {
     *x = 2.250  * 0.0;
     *y = 2.250  * 1.0;
  }
  if (y0new==0 && y1new==0 && y2new==0 && y3new==1 &&
y4new==1)
  {
     *x = 2.250  * 4.0;
     *y = 2.250  * 1.0;
  }
  if (y0new==0 && y1new==0 && y2new==1 && y3new==0 &&
y4new==0)
  {
     *x = 2.250  * 4.0;
     *y = 2.250  * -1.0;
  }
  if (y0new==0 && y1new==0 && y2new==1 && y3new==0 &&
y4new==1)
  {
     *x = 2.250  * 0.0;
     *y = 2.250  * 3.0;
  }
  if (y0new==0 && y1new==0 && y2new==1 && y3new==1 &&
y4new==0)
```

```
  {
     *x = 2.250  * 0.0;
     *y = 2.250  * -1.0;
  }
  if (y0new==0 && y1new==0 && y2new==1 && y3new==1 &&
y4new==1)
  {
     *x = 2.250  * -4.0;
     *y = 2.250  * -1.0;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==0 &&
y4new==0)
  {
     *x = 2.250  * -2.0;
     *y = 2.250  * 3.0;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==0 &&
y4new==1)
  {
     *x = 2.250  * -2.0;
     *y = 2.250  * -1.0;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==1 &&
y4new==0)
  {
     *x = 2.250  * 2.0;
     *y = 2.250  * 3.0;
  }
  if (y0new==0 && y1new==1 && y2new==0 && y3new==1 &&
y4new==1)
  {
     *x = 2.250  * 2.0;
     *y = 2.250  * -1.0;
  }
  if (y0new==0 && y1new==1 && y2new==1 && y3new==0 &&
y4new==0)
  {
     *x = 2.250  * 2.0;
     *y = 2.250  * -3.0;
  }
  if (y0new==0 && y1new==1 && y2new==1 && y3new==0 &&
y4new==1)
  {
     *x = 2.250  * 2.0;
     *y = 2.250  * 1.0;
  }
  if (y0new==0 && y1new==1 && y2new==1 && y3new==1 &&
y4new==0)
  {
```

```
      *x = 2.250  * -2.0;
      *y = 2.250  * -3.0;
   }
   if (y0new==0 && y1new==1 && y2new==1 && y3new==1 &&
y4new==1)
   {
      *x = 2.250  * -2.0;
      *y = 2.250  * 1.0;
   }
   if (y0new==1 && y1new==0 && y2new==0 && y3new==0 &&
y4new==0)
   {
      *x = 2.250  * -3.0;
      *y = 2.250  * -2.0;
   }
   if (y0new==1 && y1new==0 && y2new==0 && y3new==0 &&
y4new==1)
   {
      *x = 2.250  * 1.0;
      *y = 2.250  * -2.0;
   }
   if (y0new==1 && y1new==0 && y2new==0 && y3new==1 &&
y4new==0)
   {
      *x = 2.250  * -3.0;
      *y = 2.250  * 2.0;
   }
   if (y0new==1 && y1new==0 && y2new==0 && y3new==1 &&
y4new==1)
   {
      *x = 2.250  * 1.0;
      *y = 2.250  * 2.0;
   }
   if (y0new==1 && y1new==0 && y2new==1 && y3new==0 &&
y4new==0)
   {
      *x = 2.250  * 3.0;
      *y = 2.250  * 2.0;
   }
   if (y0new==1 && y1new==0 && y2new==1 && y3new==0 &&
y4new==1)
   {
      *x = 2.250  * -1.0;
      *y = 2.250  * 2.0;
   }
   if (y0new==1 && y1new==0 && y2new==1 && y3new==1 &&
y4new==0)
   {
      *x = 2.250  * 3.0;
```

```
      *y = 2.250  * -2.0;
   }
   if (y0new==1 && y1new==0 && y2new==1 && y3new==1 &&
y4new==1)
   {
      *x = 2.250  * -1.0;
      *y = 2.250  * -2.0;
   }
   if (y0new==1 && y1new==1 && y2new==0 && y3new==0 &&
y4new==0)
   {
      *x = 2.250  * 1.0;
      *y = 2.250  * 4.0;
   }
   if (y0new==1 && y1new==1 && y2new==0 && y3new==0 &&
y4new==1)
   {
      *x = 2.250  * -3.0;
      *y = 2.250  * 0.0;
   }
   if (y0new==1 && y1new==1 && y2new==0 && y3new==1 &&
y4new==0)
   {
      *x = 2.250  * 1.0;
      *y = 2.250  * 0.0;
   }
   if (y0new==1 && y1new==1 && y2new==0 && y3new==1 &&
y4new==1)
   {
      *x = 2.250  * 1.0;
      *y = 2.250  * -4.0;
   }
   if (y0new==1 && y1new==1 && y2new==1 && y3new==0 &&
y4new==0)
   {
      *x = 2.250  * -1.0;
      *y = 2.250  * -4.0;
   }
   if (y0new==1 && y1new==1 && y2new==1 && y3new==0 &&
y4new==1)
   {
      *x = 2.250  * 3.0;
      *y = 2.250  * 0.0;
   }
   if (y0new==1 && y1new==1 && y2new==1 && y3new==1 &&
y4new==0)
   {
      *x = 2.250  * -1.0;
      *y = 2.250  * 0.0;
```

```
  }
  if (y0new==1 && y1new==1 && y2new==1 && y3new==1 &&
y4new==1)
  {
     *x = 2.250  * -1.0;
     *y = 2.250  * 4.0;
  }

  return 0;
}
```

```
/* state_from mapper*/
/* CALCULATES THE STATE A SYMBOL CAME FROM, FOR USE WITH
*/
/* STATE_TO BEING OF VALUE 0 - 63
*/
/* FUNCTION
STATE_FROM1(POSS_SYMBOL,STATE_TO,STATE_FROM_INDEX)*/
state_from1(poss_symbol,state_to,state_from_index)
int poss_symbol,state_to;
int *state_from_index;
{
     if (state_to == 0 || state_to == 1 || state_to == 2 ||
state_to == 3 ||
           state_to == 4 || state_to == 5 || state_to == 6
|| state_to == 7 ||
           state_to == 8 || state_to == 9 || state_to ==10
|| state_to ==11 ||
           state_to ==12 || state_to ==13 || state_to ==14
|| state_to ==15)
     {
          if (poss_symbol == 0)
          *state_from_index = 0;
          if (poss_symbol == 1)
          *state_from_index = 1;
          if (poss_symbol == 2)
          *state_from_index = 2;
          if (poss_symbol == 3)
          *state_from_index = 3;
          if (poss_symbol == 4)
          *state_from_index = 4;
          if (poss_symbol == 5)
          *state_from_index = 5;
          if (poss_symbol == 6)
          *state_from_index = 6;
          if (poss_symbol == 7)
          *state_from_index = 7;
          if (poss_symbol == 8)
          *state_from_index = 8;
          if (poss_symbol == 9)
          *state_from_index = 9;
          if (poss_symbol == 10)
          *state_from_index = 10;
          if (poss_symbol == 11)
          *state_from_index = 11;
          if (poss_symbol == 12)
          *state_from_index = 12;
          if (poss_symbol == 13)
          *state_from_index = 13;
          if (poss_symbol == 14)
```

```
*state_from_index = 14;
if (poss_symbol == 15)
*state_from_index = 15;
if (poss_symbol == 16)
*state_from_index = 96;
if (poss_symbol == 17)
*state_from_index = 97;
if (poss_symbol == 18)
*state_from_index = 98;
if (poss_symbol == 19)
*state_from_index = 99;
if (poss_symbol == 20)
*state_from_index = 100;
if (poss_symbol == 21)
*state_from_index = 101;
if (poss_symbol == 22)
*state_from_index = 102;
if (poss_symbol == 23)
*state_from_index = 103;
if (poss_symbol == 24)
*state_from_index = 104;
if (poss_symbol == 25)
*state_from_index = 105;
if (poss_symbol == 26)
*state_from_index = 106;
if (poss_symbol == 27)
*state_from_index = 107;
if (poss_symbol == 28)
*state_from_index = 108;
if (poss_symbol == 29)
*state_from_index = 109;
if (poss_symbol == 30)
*state_from_index = 110;
if (poss_symbol == 31)
*state_from_index = 111;
if (poss_symbol == 32)
*state_from_index = 64;
if (poss_symbol == 33)
*state_from_index = 65;
if (poss_symbol == 34)
*state_from_index = 66;
if (poss_symbol == 35)
*state_from_index = 67;
if (poss_symbol == 36)
*state_from_index = 68;
if (poss_symbol == 37)
*state_from_index = 69;
if (poss_symbol == 38)
*state_from_index = 70;
```

```
if (poss_symbol == 39)
*state_from_index = 71;
if (poss_symbol == 40)
*state_from_index = 72;
if (poss_symbol == 41)
*state_from_index = 73;
if (poss_symbol == 42)
*state_from_index = 74;
if (poss_symbol == 43)
*state_from_index = 75;
if (poss_symbol == 44)
*state_from_index = 76;
if (poss_symbol == 45)
*state_from_index = 77;
if (poss_symbol == 46)
*state_from_index = 78;
if (poss_symbol == 47)
*state_from_index = 79;
if (poss_symbol == 48)
*state_from_index = 32;
if (poss_symbol == 49)
*state_from_index = 33;
if (poss_symbol == 50)
*state_from_index = 34;
if (poss_symbol == 51)
*state_from_index = 35;
if (poss_symbol == 52)
*state_from_index = 36;
if (poss_symbol == 53)
*state_from_index = 37;
if (poss_symbol == 54)
*state_from_index = 38;
if (poss_symbol == 55)
*state_from_index = 39;
if (poss_symbol == 56)
*state_from_index = 40;
if (poss_symbol == 57)
*state_from_index = 41;
if (poss_symbol == 58)
*state_from_index = 42;
if (poss_symbol == 59)
*state_from_index = 43;
if (poss_symbol == 60)
*state_from_index = 44;
if (poss_symbol == 61)
*state_from_index = 45;
if (poss_symbol == 62)
*state_from_index = 46;
if (poss_symbol == 63)
```

```
            *state_from_index = 47;
      }
      if (state_to ==16 || state_to == 17|| state_to == 18||
state_to == 19||
             state_to ==20 || state_to == 21|| state_to ==22
|| state_to == 23||
             state_to ==24 || state_to == 25|| state_to ==26
|| state_to ==27 ||
             state_to ==28 || state_to ==29 || state_to ==30
|| state_to ==31)
      {
            if (poss_symbol == 0)
            *state_from_index = 32;
            if (poss_symbol == 1)
            *state_from_index = 33;
            if (poss_symbol == 2)
            *state_from_index = 34;
            if (poss_symbol == 3)
            *state_from_index = 35;
            if (poss_symbol == 4)
            *state_from_index = 36;
            if (poss_symbol == 5)
            *state_from_index = 37;
            if (poss_symbol == 6)
            *state_from_index = 38;
            if (poss_symbol == 7)
            *state_from_index = 39;
            if (poss_symbol == 8)
            *state_from_index = 40;
            if (poss_symbol == 9)
            *state_from_index = 41;
            if (poss_symbol == 10)
            *state_from_index = 42;
            if (poss_symbol == 11)
            *state_from_index = 43;
            if (poss_symbol == 12)
            *state_from_index = 44;
            if (poss_symbol == 13)
            *state_from_index = 45;
            if (poss_symbol == 14)
            *state_from_index = 46;
            if (poss_symbol == 15)
            *state_from_index = 47;
            if (poss_symbol == 16)
            *state_from_index = 64;
            if (poss_symbol == 17)
            *state_from_index = 65;
            if (poss_symbol == 18)
            *state_from_index = 66;
```

```
if (poss_symbol == 19)
*state_from_index = 67;
if (poss_symbol == 20)
*state_from_index = 68;
if (poss_symbol == 21)
*state_from_index = 69;
if (poss_symbol == 22)
*state_from_index = 70;
if (poss_symbol == 23)
*state_from_index = 71;
if (poss_symbol == 24)
*state_from_index = 72;
if (poss_symbol == 25)
*state_from_index = 73;
if (poss_symbol == 26)
*state_from_index = 74;
if (poss_symbol == 27)
*state_from_index = 75;
if (poss_symbol == 28)
*state_from_index = 76;
if (poss_symbol == 29)
*state_from_index = 77;
if (poss_symbol == 30)
*state_from_index = 78;
if (poss_symbol == 31)
*state_from_index = 79;
if (poss_symbol == 32)
*state_from_index = 96;
if (poss_symbol == 33)
*state_from_index = 97;
if (poss_symbol == 34)
*state_from_index = 98;
if (poss_symbol == 35)
*state_from_index = 99;
if (poss_symbol == 36)
*state_from_index = 100;
if (poss_symbol == 37)
*state_from_index = 101;
if (poss_symbol == 38)
*state_from_index = 102;
if (poss_symbol == 39)
*state_from_index = 103;
if (poss_symbol == 40)
*state_from_index = 104;
if (poss_symbol == 41)
*state_from_index = 105;
if (poss_symbol == 42)
*state_from_index = 106;
if (poss_symbol == 43)
```

```
        *state_from_index = 107;
        if (poss_symbol == 44)
        *state_from_index = 108;
        if (poss_symbol == 45)
        *state_from_index = 109;
        if (poss_symbol == 46)
        *state_from_index = 110;
        if (poss_symbol == 47)
        *state_from_index = 111;
        if (poss_symbol == 48)
        *state_from_index = 0;
        if (poss_symbol == 49)
        *state_from_index = 1;
        if (poss_symbol == 50)
        *state_from_index = 2;
        if (poss_symbol == 51)
        *state_from_index = 3;
        if (poss_symbol == 52)
        *state_from_index = 4;
        if (poss_symbol == 53)
        *state_from_index = 5;
        if (poss_symbol == 54)
        *state_from_index = 6;
        if (poss_symbol == 55)
        *state_from_index = 7;
        if (poss_symbol == 56)
        *state_from_index = 8;
        if (poss_symbol == 57)
        *state_from_index = 9;
        if (poss_symbol == 58)
        *state_from_index = 10;
        if (poss_symbol == 59)
        *state_from_index = 11;
        if (poss_symbol == 60)
        *state_from_index = 12;
        if (poss_symbol == 61)
        *state_from_index = 13;
        if (poss_symbol == 62)
        *state_from_index = 14;
        if (poss_symbol == 63)
        *state_from_index = 15;
    }

    if (state_to ==32 ¦¦ state_to == 33¦¦ state_to == 34¦¦
state_to == 35¦¦
          state_to ==36 ¦¦ state_to == 37¦¦ state_to ==38
¦¦ state_to == 39¦¦
          state_to ==40 ¦¦ state_to == 41¦¦ state_to ==42
¦¦ state_to ==43 ¦¦
```

```
            state_to ==44 || state_to ==45 || state_to ==46
|| state_to ==47)
     {
          if (poss_symbol == 0)
          *state_from_index = 64;
          if (poss_symbol == 1)
          *state_from_index = 65;
          if (poss_symbol == 2)
          *state_from_index = 66;
          if (poss_symbol == 3)
          *state_from_index = 67;
          if (poss_symbol == 4)
          *state_from_index = 68;
          if (poss_symbol == 5)
          *state_from_index = 69;
          if (poss_symbol == 6)
          *state_from_index = 70;
          if (poss_symbol == 7)
          *state_from_index = 71;
          if (poss_symbol == 8)
          *state_from_index = 72;
          if (poss_symbol == 9)
          *state_from_index = 73;
          if (poss_symbol == 10)
          *state_from_index = 74;
          if (poss_symbol == 11)
          *state_from_index = 75;
          if (poss_symbol == 12)
          *state_from_index = 76;
          if (poss_symbol == 13)
          *state_from_index = 77;
          if (poss_symbol == 14)
          *state_from_index = 78;
          if (poss_symbol == 15)
          *state_from_index = 79;
          if (poss_symbol == 16)
          *state_from_index = 32;
          if (poss_symbol == 17)
          *state_from_index = 33;
          if (poss_symbol == 18)
          *state_from_index = 34;
          if (poss_symbol == 19)
          *state_from_index = 35;
          if (poss_symbol == 20)
          *state_from_index = 36;
          if (poss_symbol == 21)
          *state_from_index = 37;
          if (poss_symbol == 22)
          *state_from_index = 38;
```

```
if (poss_symbol == 23)
*state_from_index = 39;
if (poss_symbol == 24)
*state_from_index = 40;
if (poss_symbol == 25)
*state_from_index = 41;
if (poss_symbol == 26)
*state_from_index = 42;
if (poss_symbol == 27)
*state_from_index = 43;
if (poss_symbol == 28)
*state_from_index = 44;
if (poss_symbol == 29)
*state_from_index = 45;
if (poss_symbol == 30)
*state_from_index = 46;
if (poss_symbol == 31)
*state_from_index = 47;
if (poss_symbol == 32)
*state_from_index = 0;
if (poss_symbol == 33)
*state_from_index = 1;
if (poss_symbol == 34)
*state_from_index = 2;
if (poss_symbol == 35)
*state_from_index = 3;
if (poss_symbol == 36)
*state_from_index = 4;
if (poss_symbol == 37)
*state_from_index = 5;
if (poss_symbol == 38)
*state_from_index = 6;
if (poss_symbol == 39)
*state_from_index = 7;
if (poss_symbol == 40)
*state_from_index = 8;
if (poss_symbol == 41)
*state_from_index = 9;
if (poss_symbol == 42)
*state_from_index = 10;
if (poss_symbol == 43)
*state_from_index = 11;
if (poss_symbol == 44)
*state_from_index = 12;
if (poss_symbol == 45)
*state_from_index = 13;
if (poss_symbol == 46)
*state_from_index = 14;
if (poss_symbol == 47)
```

```
            *state_from_index = 15;
            if (poss_symbol == 48)
            *state_from_index = 96;
            if (poss_symbol == 49)
            *state_from_index = 97;
            if (poss_symbol == 50)
            *state_from_index = 98;
            if (poss_symbol == 51)
            *state_from_index = 99;
            if (poss_symbol == 52)
            *state_from_index = 100;
            if (poss_symbol == 53)
            *state_from_index = 101;
            if (poss_symbol == 54)
            *state_from_index = 102;
            if (poss_symbol == 55)
            *state_from_index = 103;
            if (poss_symbol == 56)
            *state_from_index = 104;
            if (poss_symbol == 57)
            *state_from_index = 105;
            if (poss_symbol == 58)
            *state_from_index = 106;
            if (poss_symbol == 59)
            *state_from_index = 107;
            if (poss_symbol == 60)
            *state_from_index = 108;
            if (poss_symbol == 61)
            *state_from_index = 109;
            if (poss_symbol == 62)
            *state_from_index = 110;
            if (poss_symbol == 63)
            *state_from_index = 111;
        }

        if (state_to ==48 || state_to == 49|| state_to == 50||
state_to == 51||
            state_to ==52 || state_to == 53|| state_to ==54
|| state_to == 55||
            state_to ==56 || state_to == 57|| state_to ==58
|| state_to ==59 ||
            state_to ==60 || state_to ==61 || state_to ==62
|| state_to ==63)
        {
            if (poss_symbol == 0)
            *state_from_index = 96;
            if (poss_symbol == 1)
            *state_from_index = 97;
            if (poss_symbol == 2)
```

```
*state_from_index = 98;
if (poss_symbol == 3)
*state_from_index = 99;
if (poss_symbol == 4)
*state_from_index = 100;
if (poss_symbol == 5)
*state_from_index = 101;
if (poss_symbol == 6)
*state_from_index = 102;
if (poss_symbol == 7)
*state_from_index = 103;
if (poss_symbol == 8)
*state_from_index = 104;
if (poss_symbol == 9)
*state_from_index = 105;
if (poss_symbol == 10)
*state_from_index = 106;
if (poss_symbol == 11)
*state_from_index = 107;
if (poss_symbol == 12)
*state_from_index = 108;
if (poss_symbol == 13)
*state_from_index = 109;
if (poss_symbol == 14)
*state_from_index = 110;
if (poss_symbol == 15)
*state_from_index = 111;
if (poss_symbol == 16)
*state_from_index = 0;
if (poss_symbol == 17)
*state_from_index = 1;
if (poss_symbol == 18)
*state_from_index = 2;
if (poss_symbol == 19)
*state_from_index = 3;
if (poss_symbol == 20)
*state_from_index = 4;
if (poss_symbol == 21)
*state_from_index = 5;
if (poss_symbol == 22)
*state_from_index = 6;
if (poss_symbol == 23)
*state_from_index = 7;
if (poss_symbol == 24)
*state_from_index = 8;
if (poss_symbol == 25)
*state_from_index = 9;
if (poss_symbol == 26)
*state_from_index = 10;
```

```
if (poss_symbol == 27)
*state_from_index = 11;
if (poss_symbol == 28)
*state_from_index = 12;
if (poss_symbol == 29)
*state_from_index = 13;
if (poss_symbol == 30)
*state_from_index = 14;
if (poss_symbol == 31)
*state_from_index = 15;
if (poss_symbol == 32)
*state_from_index = 32;
if (poss_symbol == 33)
*state_from_index = 33;
if (poss_symbol == 34)
*state_from_index = 34;
if (poss_symbol == 35)
*state_from_index = 35;
if (poss_symbol == 36)
*state_from_index = 36;
if (poss_symbol == 37)
*state_from_index = 37;
if (poss_symbol == 38)
*state_from_index = 38;
if (poss_symbol == 39)
*state_from_index = 39;
if (poss_symbol == 40)
*state_from_index = 40;
if (poss_symbol == 41)
*state_from_index = 41;
if (poss_symbol == 42)
*state_from_index = 42;
if (poss_symbol == 43)
*state_from_index = 43;
if (poss_symbol == 44)
*state_from_index = 44;
if (poss_symbol == 45)
*state_from_index = 45;
if (poss_symbol == 46)
*state_from_index = 46;
if (poss_symbol == 47)
*state_from_index = 47;
if (poss_symbol == 48)
*state_from_index = 64;
if (poss_symbol == 49)
*state_from_index = 65;
if (poss_symbol == 50)
*state_from_index = 66;
if (poss_symbol == 51)
```

```
          *state_from_index = 67;
          if (poss_symbol == 52)
          *state_from_index = 68;
          if (poss_symbol == 53)
          *state_from_index = 69;
          if (poss_symbol == 54)
          *state_from_index = 70;
          if (poss_symbol == 55)
          *state_from_index = 71;
          if (poss_symbol == 56)
          *state_from_index = 72;
          if (poss_symbol == 57)
          *state_from_index = 73;
          if (poss_symbol == 58)
          *state_from_index = 74;
          if (poss_symbol == 59)
          *state_from_index = 75;
          if (poss_symbol == 60)
          *state_from_index = 76;
          if (poss_symbol == 61)
          *state_from_index = 77;
          if (poss_symbol == 62)
          *state_from_index = 78;
          if (poss_symbol == 63)
          *state_from_index = 79;
     }
return 0;
}
```

```
/* state_from mapper*/
/* CALCULATES THE STATE A SYMBOL CAME FROM, FOR USE WITH
*/
/* STATE_TO BEING OF VALUE 64 - 127
*/
/* FUNCTION
STATE_FROM2(POSS_SYMBOL,STATE_TO,STATE_FROM_INDEX)*/
state_from2(poss_symbol,state_to,state_from_index)
int poss_symbol,state_to;
int *state_from_index;
{

     if (state_to ==64 || state_to == 65|| state_to == 66||
state_to == 67||
           state_to ==68 || state_to == 69|| state_to ==70
|| state_to == 71||
           state_to ==72 || state_to == 73|| state_to ==74
|| state_to ==75 ||
           state_to ==76 || state_to ==77 || state_to ==78
|| state_to ==79)
     {
          if (poss_symbol == 64)
          *state_from_index = 16;
          if (poss_symbol == 65)
          *state_from_index = 17;
          if (poss_symbol == 66)
          *state_from_index = 18;
          if (poss_symbol == 67)
          *state_from_index = 19;
          if (poss_symbol == 68)
          *state_from_index = 20;
          if (poss_symbol == 69)
          *state_from_index = 21;
          if (poss_symbol == 70)
          *state_from_index = 22;
          if (poss_symbol == 71)
          *state_from_index = 23;
          if (poss_symbol == 72)
          *state_from_index = 24;
          if (poss_symbol == 73)
          *state_from_index = 25;
          if (poss_symbol == 74)
          *state_from_index = 26;
          if (poss_symbol == 75)
          *state_from_index = 27;
          if (poss_symbol == 76)
          *state_from_index = 28;
          if (poss_symbol == 77)
          *state_from_index = 29;
```

```
if (poss_symbol == 78)
*state_from_index = 30;
if (poss_symbol == 79)
*state_from_index = 31;
if (poss_symbol == 80)
*state_from_index = 112;
if (poss_symbol == 81)
*state_from_index = 113;
if (poss_symbol == 82)
*state_from_index = 114;
if (poss_symbol == 83)
*state_from_index = 115;
if (poss_symbol == 84)
*state_from_index = 116;
if (poss_symbol == 85)
*state_from_index = 117;
if (poss_symbol == 86)
*state_from_index = 118;
if (poss_symbol == 87)
*state_from_index = 119;
if (poss_symbol == 88)
*state_from_index = 120;
if (poss_symbol == 89)
*state_from_index = 121;
if (poss_symbol == 90)
*state_from_index = 122;
if (poss_symbol == 91)
*state_from_index = 123;
if (poss_symbol == 92)
*state_from_index = 124;
if (poss_symbol == 93)
*state_from_index = 125;
if (poss_symbol == 94)
*state_from_index = 126;
if (poss_symbol == 95)
*state_from_index = 127;
if (poss_symbol == 96)
*state_from_index = 48;
if (poss_symbol == 97)
*state_from_index = 49;
if (poss_symbol == 98)
*state_from_index = 50;
if (poss_symbol == 99)
*state_from_index = 51;
if (poss_symbol == 100)
*state_from_index = 52;
if (poss_symbol == 101)
*state_from_index = 53;
if (poss_symbol == 102)
```

```
*state_from_index = 54;
if (poss_symbol == 103)
*state_from_index = 55;
if (poss_symbol == 104)
*state_from_index = 56;
if (poss_symbol == 105)
*state_from_index = 57;
if (poss_symbol == 106)
*state_from_index = 58;
if (poss_symbol == 107)
*state_from_index = 59;
if (poss_symbol == 108)
*state_from_index = 60;
if (poss_symbol == 109)
*state_from_index = 61;
if (poss_symbol == 110)
*state_from_index = 62;
if (poss_symbol == 111)
*state_from_index = 63;
if (poss_symbol == 112)
*state_from_index = 80;
if (poss_symbol == 113)
*state_from_index = 81;
if (poss_symbol == 114)
*state_from_index = 82;
if (poss_symbol == 115)
*state_from_index = 83;
if (poss_symbol == 116)
*state_from_index = 84;
if (poss_symbol == 117)
*state_from_index = 85;
if (poss_symbol == 118)
*state_from_index = 86;
if (poss_symbol == 119)
*state_from_index = 87;
if (poss_symbol == 120)
*state_from_index = 88;
if (poss_symbol == 121)
*state_from_index = 89;
if (poss_symbol == 122)
*state_from_index = 90;
if (poss_symbol == 123)
*state_from_index = 91;
if (poss_symbol == 124)
*state_from_index = 92;
if (poss_symbol == 125)
*state_from_index = 93;
if (poss_symbol == 126)
*state_from_index = 94;
```

```
            if (poss_symbol == 127)
            *state_from_index = 95;
        }

        if (state_to ==80 || state_to == 81|| state_to == 82||
state_to == 83||
             state_to ==84 || state_to == 85|| state_to ==86
|| state_to == 87||
             state_to ==88 || state_to == 89|| state_to ==90
|| state_to ==91 ||
             state_to ==92 || state_to ==93 || state_to ==94
|| state_to ==95)
        {
            if (poss_symbol == 64)
            *state_from_index = 112;
            if (poss_symbol == 65)
            *state_from_index = 113;
            if (poss_symbol == 66)
            *state_from_index = 114;
            if (poss_symbol == 67)
            *state_from_index = 115;
            if (poss_symbol == 68)
            *state_from_index = 116;
            if (poss_symbol == 69)
            *state_from_index = 117;
            if (poss_symbol == 70)
            *state_from_index = 118;
            if (poss_symbol == 71)
            *state_from_index = 119;
            if (poss_symbol == 72)
            *state_from_index = 120;
            if (poss_symbol == 73)
            *state_from_index = 121;
            if (poss_symbol == 74)
            *state_from_index = 122;
            if (poss_symbol == 75)
            *state_from_index = 123;
            if (poss_symbol == 76)
            *state_from_index = 124;
            if (poss_symbol == 77)
            *state_from_index = 125;
            if (poss_symbol == 78)
            *state_from_index = 126;
            if (poss_symbol == 79)
            *state_from_index = 127;
            if (poss_symbol == 80)
            *state_from_index = 16;
            if (poss_symbol == 81)
            *state_from_index = 17;
```

```
if (poss_symbol == 82)
*state_from_index = 18;
if (poss_symbol == 83)
*state_from_index = 19;
if (poss_symbol == 84)
*state_from_index = 20;
if (poss_symbol == 85)
*state_from_index = 21;
if (poss_symbol == 86)
*state_from_index = 22;
if (poss_symbol == 87)
*state_from_index = 23;
if (poss_symbol == 88)
*state_from_index = 24;
if (poss_symbol == 89)
*state_from_index = 25;
if (poss_symbol == 90)
*state_from_index = 26;
if (poss_symbol == 91)
*state_from_index = 27;
if (poss_symbol == 92)
*state_from_index = 28;
if (poss_symbol == 93)
*state_from_index = 29;
if (poss_symbol == 94)
*state_from_index = 30;
if (poss_symbol == 95)
*state_from_index = 31;
if (poss_symbol == 96)
*state_from_index = 80;
if (poss_symbol == 97)
*state_from_index = 81;
if (poss_symbol == 98)
*state_from_index = 82;
if (poss_symbol == 99)
*state_from_index = 83;
if (poss_symbol == 100)
*state_from_index = 84;
if (poss_symbol == 101)
*state_from_index = 85;
if (poss_symbol == 102)
*state_from_index = 86;
if (poss_symbol == 103)
*state_from_index = 87;
if (poss_symbol == 104)
*state_from_index = 88;
if (poss_symbol == 105)
*state_from_index = 89;
if (poss_symbol == 106)
```

```
        *state_from_index = 90;
        if (poss_symbol == 107)
        *state_from_index = 91;
        if (poss_symbol == 108)
        *state_from_index = 92;
        if (poss_symbol == 109)
        *state_from_index = 93;
        if (poss_symbol == 110)
        *state_from_index = 94;
        if (poss_symbol == 111)
        *state_from_index = 95;
        if (poss_symbol == 112)
        *state_from_index = 48;
        if (poss_symbol == 113)
        *state_from_index = 49;
        if (poss_symbol == 114)
        *state_from_index = 50;
        if (poss_symbol == 115)
        *state_from_index = 51;
        if (poss_symbol == 116)
        *state_from_index = 52;
        if (poss_symbol == 117)
        *state_from_index = 53;
        if (poss_symbol == 118)
        *state_from_index = 54;
        if (poss_symbol == 119)
        *state_from_index = 55;
        if (poss_symbol == 120)
        *state_from_index = 56;
        if (poss_symbol == 121)
        *state_from_index = 57;
        if (poss_symbol == 122)
        *state_from_index = 58;
        if (poss_symbol == 123)
        *state_from_index = 59;
        if (poss_symbol == 124)
        *state_from_index = 60;
        if (poss_symbol == 125)
        *state_from_index = 61;
        if (poss_symbol == 126)
        *state_from_index = 62;
        if (poss_symbol == 127)
        *state_from_index = 63;
    }

    if (state_to ==96 || state_to == 97|| state_to == 98||
state_to == 99||
          state_to ==100|| state_to == 101| state_to
==102|| state_to ==103||
```

```
            state_to ==104¦¦ state_to ==105¦¦ state_to
==106¦¦ state_to ==107¦¦
            state_to ==108¦¦ state_to ==109¦¦ state_to
==110¦¦ state_to ==111)
      {
            if (poss_symbol == 64)
            *state_from_index = 80;
            if (poss_symbol == 65)
            *state_from_index = 81;
            if (poss_symbol == 66)
            *state_from_index = 82;
            if (poss_symbol == 67)
            *state_from_index = 83;
            if (poss_symbol == 68)
            *state_from_index = 84;
            if (poss_symbol == 69)
            *state_from_index = 85;
            if (poss_symbol == 70)
            *state_from_index = 86;
            if (poss_symbol == 71)
            *state_from_index = 87;
            if (poss_symbol == 72)
            *state_from_index = 88;
            if (poss_symbol == 73)
            *state_from_index = 89;
            if (poss_symbol == 74)
            *state_from_index = 90;
            if (poss_symbol == 75)
            *state_from_index = 91;
            if (poss_symbol == 76)
            *state_from_index = 92;
            if (poss_symbol == 77)
            *state_from_index = 93;
            if (poss_symbol == 78)
            *state_from_index = 94;
            if (poss_symbol == 79)
            *state_from_index = 95;
            if (poss_symbol == 80)
            *state_from_index = 48;
            if (poss_symbol == 81)
            *state_from_index = 49;
            if (poss_symbol == 82)
            *state_from_index = 50;
            if (poss_symbol == 83)
            *state_from_index = 51;
            if (poss_symbol == 84)
            *state_from_index = 52;
            if (poss_symbol == 85)
            *state_from_index = 53;
```

```
if (poss_symbol == 86)
*state_from_index = 54;
if (poss_symbol == 87)
*state_from_index = 55;
if (poss_symbol == 88)
*state_from_index = 56;
if (poss_symbol == 89)
*state_from_index = 57;
if (poss_symbol == 90)
*state_from_index = 58;
if (poss_symbol == 91)
*state_from_index = 59;
if (poss_symbol == 92)
*state_from_index = 60;
if (poss_symbol == 93)
*state_from_index = 61;
if (poss_symbol == 94)
*state_from_index = 62;
if (poss_symbol == 95)
*state_from_index = 63;
if (poss_symbol == 96)
*state_from_index = 112;
if (poss_symbol == 97)
*state_from_index = 113;
if (poss_symbol == 98)
*state_from_index = 114;
if (poss_symbol == 99)
*state_from_index = 115;
if (poss_symbol == 100)
*state_from_index = 116;
if (poss_symbol == 101)
*state_from_index = 117;
if (poss_symbol == 102)
*state_from_index = 118;
if (poss_symbol == 103)
*state_from_index = 119;
if (poss_symbol == 104)
*state_from_index = 120;
if (poss_symbol == 105)
*state_from_index = 121;
if (poss_symbol == 106)
*state_from_index = 122;
if (poss_symbol == 107)
*state_from_index = 123;
if (poss_symbol == 108)
*state_from_index = 124;
if (poss_symbol == 109)
*state_from_index = 125;
if (poss_symbol == 110)
```

```
        *state_from_index = 126;
        if (poss_symbol == 111)
        *state_from_index = 127;
        if (poss_symbol == 112)
        *state_from_index = 16;
        if (poss_symbol == 113)
        *state_from_index = 17;
        if (poss_symbol == 114)
        *state_from_index = 18;
        if (poss_symbol == 115)
        *state_from_index = 19;
        if (poss_symbol == 116)
        *state_from_index = 20;
        if (poss_symbol == 117)
        *state_from_index = 21;
        if (poss_symbol == 118)
        *state_from_index = 22;
        if (poss_symbol == 119)
        *state_from_index = 23;
        if (poss_symbol == 120)
        *state_from_index = 24;
        if (poss_symbol == 121)
        *state_from_index = 25;
        if (poss_symbol == 122)
        *state_from_index = 26;
        if (poss_symbol == 123)
        *state_from_index = 27;
        if (poss_symbol == 124)
        *state_from_index = 28;
        if (poss_symbol == 125)
        *state_from_index = 29;
        if (poss_symbol == 126)
        *state_from_index = 30;
        if (poss_symbol == 127)
        *state_from_index = 31;
    }

    if (state_to ==112¦¦ state_to ==113¦¦ state_to ==114¦¦
state_to ==115¦¦
         state_to ==116¦¦ state_to ==117¦¦ state_to
==118¦¦ state_to ==119¦¦
         state_to ==120¦¦ state_to ==121¦¦ state_to
==122¦¦ state_to ==123¦¦
         state_to ==124¦¦ state_to ==125¦¦ state_to
==126¦¦ state_to ==127)
    {
        if (poss_symbol == 64)
        *state_from_index = 48;
        if (poss_symbol == 65)
```

```
*state_from_index = 49;
if (poss_symbol == 66)
*state_from_index = 50;
if (poss_symbol == 67)
*state_from_index = 51;
if (poss_symbol == 68)
*state_from_index = 52;
if (poss_symbol == 69)
*state_from_index = 53;
if (poss_symbol == 70)
*state_from_index = 54;
if (poss_symbol == 71)
*state_from_index = 55;
if (poss_symbol == 72)
*state_from_index = 56;
if (poss_symbol == 73)
*state_from_index = 57;
if (poss_symbol == 74)
*state_from_index = 58;
if (poss_symbol == 75)
*state_from_index = 59;
if (poss_symbol == 76)
*state_from_index = 60;
if (poss_symbol == 77)
*state_from_index = 61;
if (poss_symbol == 78)
*state_from_index = 62;
if (poss_symbol == 79)
*state_from_index = 63;
if (poss_symbol == 80)
*state_from_index = 80;
if (poss_symbol == 81)
*state_from_index = 81;
if (poss_symbol == 82)
*state_from_index = 82;
if (poss_symbol == 83)
*state_from_index = 83;
if (poss_symbol == 84)
*state_from_index = 84;
if (poss_symbol == 85)
*state_from_index = 85;
if (poss_symbol == 86)
*state_from_index = 86;
if (poss_symbol == 87)
*state_from_index = 87;
if (poss_symbol == 88)
*state_from_index = 88;
if (poss_symbol == 89)
*state_from_index = 89;
```

```
if (poss_symbol == 90)
*state_from_index = 90;
if (poss_symbol == 91)
*state_from_index = 91;
if (poss_symbol == 92)
*state_from_index = 92;
if (poss_symbol == 93)
*state_from_index = 93;
if (poss_symbol == 94)
*state_from_index = 94;
if (poss_symbol == 95)
*state_from_index = 95;
if (poss_symbol == 96)
*state_from_index = 16;
if (poss_symbol == 97)
*state_from_index = 17;
if (poss_symbol == 98)
*state_from_index = 18;
if (poss_symbol == 99)
*state_from_index = 19;
if (poss_symbol == 100)
*state_from_index = 20;
if (poss_symbol == 101)
*state_from_index = 21;
if (poss_symbol == 102)
*state_from_index = 22;
if (poss_symbol == 103)
*state_from_index = 23;
if (poss_symbol == 104)
*state_from_index = 24;
if (poss_symbol == 105)
*state_from_index = 25;
if (poss_symbol == 106)
*state_from_index = 26;
if (poss_symbol == 107)
*state_from_index = 27;
if (poss_symbol == 108)
*state_from_index = 28;
if (poss_symbol == 109)
*state_from_index = 29;
if (poss_symbol == 110)
*state_from_index = 30;
if (poss_symbol == 111)
*state_from_index = 31;
if (poss_symbol == 112)
*state_from_index = 112;
if (poss_symbol == 113)
*state_from_index = 113;
if (poss_symbol == 114)
```

```
            *state_from_index = 114;
            if (poss_symbol == 115)
            *state_from_index = 115;
            if (poss_symbol == 116)
            *state_from_index = 116;
            if (poss_symbol == 117)
            *state_from_index = 117;
            if (poss_symbol == 118)
            *state_from_index = 118;
            if (poss_symbol == 119)
            *state_from_index = 119;
            if (poss_symbol == 120)
            *state_from_index = 120;
            if (poss_symbol == 121)
            *state_from_index = 121;
            if (poss_symbol == 122)
            *state_from_index = 122;
            if (poss_symbol == 123)
            *state_from_index = 123;
            if (poss_symbol == 124)
            *state_from_index = 124;
            if (poss_symbol == 125)
            *state_from_index = 125;
            if (poss_symbol == 126)
            *state_from_index = 126;
            if (poss_symbol == 127)
            *state_from_index = 127;
        }
        return 0;
}
```

```
/* state_from mapper */
/* CALCULATES THE STATE A SYMBOL CAME FROM, FOR USE WITH
*/
/* STATE_TO BEING OF VALUE 0 - 31 V.32 USE ONLY
*/
/* FUNCTION
STATE_FROMV32(POSS_SYMBOL,STATE_TO,STATE_FROM_INDEX*/
state_fromv32(poss_symbol,state_to,state_from_index)
int poss_symbol,state_to;
int *state_from_index;
{
     if (state_to == 0 || state_to == 1 || state_to == 2 ||
state_to ==3)
     {
          if (poss_symbol == 0)
          *state_from_index = 0;
          if (poss_symbol == 1)
          *state_from_index = 1;
          if (poss_symbol == 2)
          *state_from_index = 2;
          if (poss_symbol == 3)
          *state_from_index = 3;
          if (poss_symbol == 4)
          *state_from_index = 24;
          if (poss_symbol == 5)
          *state_from_index = 25;
          if (poss_symbol == 6)
          *state_from_index = 26;
          if (poss_symbol == 7)
          *state_from_index = 27;
          if (poss_symbol == 8)
          *state_from_index = 16;
          if (poss_symbol == 9)
          *state_from_index = 17;
          if (poss_symbol == 10)
          *state_from_index = 18;
          if (poss_symbol == 11)
          *state_from_index = 19;
          if (poss_symbol == 12)
          *state_from_index = 8;
          if (poss_symbol == 13)
          *state_from_index = 9;
          if (poss_symbol == 14)
          *state_from_index = 10;
          if (poss_symbol == 15)
          *state_from_index = 11;
     }
     if (state_to == 4 || state_to == 5 || state_to == 6 ||
state_to ==7)
```

```
	{
		if (poss_symbol == 0)
		*state_from_index = 8;
		if (poss_symbol == 1)
		*state_from_index = 9;
		if (poss_symbol == 2)
		*state_from_index = 10;
		if (poss_symbol == 3)
		*state_from_index = 11;
		if (poss_symbol == 4)
		*state_from_index = 16;
		if (poss_symbol == 5)
		*state_from_index = 17;
		if (poss_symbol == 6)
		*state_from_index = 18;
		if (poss_symbol == 7)
		*state_from_index = 19;
		if (poss_symbol == 8)
		*state_from_index = 24;
		if (poss_symbol == 9)
		*state_from_index = 25;
		if (poss_symbol == 10)
		*state_from_index = 26;
		if (poss_symbol == 11)
		*state_from_index = 27;
		if (poss_symbol == 12)
		*state_from_index = 0;
		if (poss_symbol == 13)
		*state_from_index = 1;
		if (poss_symbol == 14)
		*state_from_index = 2;
		if (poss_symbol == 15)
		*state_from_index = 3;
	}
	if (state_to == 8 || state_to == 9 || state_to == 10 ||
state_to ==11)
	{
		if (poss_symbol == 0)
		*state_from_index = 16;
		if (poss_symbol == 1)
		*state_from_index = 17;
		if (poss_symbol == 2)
		*state_from_index = 18;
		if (poss_symbol == 3)
		*state_from_index = 19;
		if (poss_symbol == 4)
		*state_from_index = 8;
		if (poss_symbol == 5)
		*state_from_index = 9;
```

```
          if (poss_symbol == 6)
          *state_from_index = 10;
          if (poss_symbol == 7)
          *state_from_index = 11;
          if (poss_symbol == 8)
          *state_from_index = 0;
          if (poss_symbol == 9)
          *state_from_index = 1;
          if (poss_symbol == 10)
          *state_from_index = 2;
          if (poss_symbol == 11)
          *state_from_index = 3;
          if (poss_symbol == 12)
          *state_from_index = 24;
          if (poss_symbol == 13)
          *state_from_index = 25;
          if (poss_symbol == 14)
          *state_from_index = 26;
          if (poss_symbol == 15)
          *state_from_index = 27;
     }
     if (state_to == 12 || state_to == 13 || state_to == 14
|| state_to ==15)
     {
          if (poss_symbol == 0)
          *state_from_index = 24;
          if (poss_symbol == 1)
          *state_from_index = 25;
          if (poss_symbol == 2)
          *state_from_index = 26;
          if (poss_symbol == 3)
          *state_from_index = 27;
          if (poss_symbol == 4)
          *state_from_index = 0;
          if (poss_symbol == 5)
          *state_from_index = 1;
          if (poss_symbol == 6)
          *state_from_index = 2;
          if (poss_symbol == 7)
          *state_from_index = 3;
          if (poss_symbol == 8)
          *state_from_index = 8;
          if (poss_symbol == 9)
          *state_from_index = 9;
          if (poss_symbol == 10)
          *state_from_index = 10;
          if (poss_symbol == 11)
          *state_from_index = 11;
          if (poss_symbol == 12)
```

```
            *state_from_index = 16;
            if (poss_symbol == 13)
            *state_from_index = 17;
            if (poss_symbol == 14)
            *state_from_index = 18;
            if (poss_symbol == 15)
            *state_from_index = 19;
      }

      if (state_to == 16 || state_to == 17 || state_to == 18
|| state_to ==19)
      {
            if (poss_symbol == 16)
            *state_from_index = 4;
            if (poss_symbol == 17)
            *state_from_index = 5;
            if (poss_symbol == 18)
            *state_from_index = 6;
            if (poss_symbol == 19)
            *state_from_index = 7;
            if (poss_symbol == 20)
            *state_from_index = 28;
            if (poss_symbol == 21)
            *state_from_index = 29;
            if (poss_symbol == 22)
            *state_from_index = 30;
            if (poss_symbol == 23)
            *state_from_index = 31;
            if (poss_symbol == 24)
            *state_from_index = 12;
            if (poss_symbol == 25)
            *state_from_index = 13;
            if (poss_symbol == 26)
            *state_from_index = 14;
            if (poss_symbol == 27)
            *state_from_index = 15;
            if (poss_symbol == 28)
            *state_from_index = 20;
            if (poss_symbol == 29)
            *state_from_index = 21;
            if (poss_symbol == 30)
            *state_from_index = 22;
            if (poss_symbol == 31)
            *state_from_index = 23;
      }
      if (state_to == 20 || state_to == 21 || state_to == 22
|| state_to ==23)
      {
            if (poss_symbol == 16)
```

```
            *state_from_index = 28;
            if (poss_symbol == 17)
            *state_from_index = 29;
            if (poss_symbol == 18)
            *state_from_index = 30;
            if (poss_symbol == 19)
            *state_from_index = 31;
            if (poss_symbol == 20)
            *state_from_index = 4;
            if (poss_symbol == 21)
            *state_from_index = 5;
            if (poss_symbol == 22)
            *state_from_index = 6;
            if (poss_symbol == 23)
            *state_from_index = 7;
            if (poss_symbol == 24)
            *state_from_index = 20;
            if (poss_symbol == 25)
            *state_from_index = 21;
            if (poss_symbol == 26)
            *state_from_index = 22;
            if (poss_symbol == 27)
            *state_from_index = 23;
            if (poss_symbol == 28)
            *state_from_index = 12;
            if (poss_symbol == 29)
            *state_from_index = 13;
            if (poss_symbol == 30)
            *state_from_index = 14;
            if (poss_symbol == 31)
            *state_from_index = 15;
        }
        if (state_to == 24 || state_to == 25 || state_to == 26
|| state_to ==27)
        {
            if (poss_symbol == 16)
            *state_from_index = 20;
            if (poss_symbol == 17)
            *state_from_index = 21;
            if (poss_symbol == 18)
            *state_from_index = 22;
            if (poss_symbol == 19)
            *state_from_index = 23;
            if (poss_symbol == 20)
            *state_from_index = 12;
            if (poss_symbol == 21)
            *state_from_index = 13;
            if (poss_symbol == 22)
            *state_from_index = 14;
```

```
            if (poss_symbol == 23)
            *state_from_index = 15;
            if (poss_symbol == 24)
            *state_from_index = 28;
            if (poss_symbol == 25)
            *state_from_index = 29;
            if (poss_symbol == 26)
            *state_from_index = 30;
            if (poss_symbol == 27)
            *state_from_index = 31;
            if (poss_symbol == 28)
            *state_from_index = 4;
            if (poss_symbol == 29)
            *state_from_index = 5;
            if (poss_symbol == 30)
            *state_from_index = 6;
            if (poss_symbol == 31)
            *state_from_index = 7;
      }
      if (state_to == 28 || state_to == 29 || state_to == 30
|| state_to ==31)
      {
            if (poss_symbol == 16)
            *state_from_index = 12;
            if (poss_symbol == 17)
            *state_from_index = 13;
            if (poss_symbol == 18)
            *state_from_index = 14;
            if (poss_symbol == 19)
            *state_from_index = 15;
            if (poss_symbol == 20)
            *state_from_index = 20;
            if (poss_symbol == 21)
            *state_from_index = 21;
            if (poss_symbol == 22)
            *state_from_index = 22;
            if (poss_symbol == 23)
            *state_from_index = 23;
            if (poss_symbol == 24)
            *state_from_index = 4;
            if (poss_symbol == 25)
            *state_from_index = 5;
            if (poss_symbol == 26)
            *state_from_index = 6;
            if (poss_symbol == 27)
            *state_from_index = 7;
            if (poss_symbol == 28)
            *state_from_index = 28;
            if (poss_symbol == 29)
```

```
            *state_from_index = 29;
            if (poss_symbol == 30)
            *state_from_index = 30;
            if (poss_symbol == 31)
            *state_from_index = 31;
      }
return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
/* THIS IS THE REPORTING FUNCTION FILE IT CONTAINS
*/
/* THE REPORT GENERATION FOR V.32 AND V.33REPORTS
*/
/*FUNCTION
REPORT32(VML,NRF,XCE,YCE,DF,NUMXY,INPUT_FILE,REPORT_FILE) */
report32(vml,nrf,xce,yce,df,numxy,input_file,report_file)
int vml,df,numxy;
float nrf,xce,yce;
char input_file[25],report_file[25];
{
char output[8];
char input[8];
float data_rate;
int bit_count = 0,bit_error_count =0,char_count =
0,char_error_count = 0;
int symbol_error_count=0,symbol_count=0,ct;
FILE *fptr1,*fptr2,*fptr3;
fptr1 = fopen("c:\\modem\\binary.txt","r");
fptr2 = fopen("c:\\modem\\binary_o.txt","r");
fptr3 = fopen(report_file,"w");
while(numxy/2 > char_count)
{
for (ct =0; ct<8; ct++)
{
input[ct]=getc(fptr1);
output[ct] = getc(fptr2);
bit_count++;
}

char_count++;
if (output[0] != input[0])
bit_error_count++;
if (output[1] != input[1])
bit_error_count++;
if (output[2] != input[2])
bit_error_count++;
if (output[3] != input[3])
bit_error_count++;
if (output[4] != input[4])
bit_error_count++;
if (output[5] != input[5])
bit_error_count++;
if (output[6] != input[6])
bit_error_count++;
if (output[7] != input[7])
bit_error_count++;
```

```
if ( ((output[0] != input[0])||(output[1] != input[1]))
||((output[2] != input[2])||(output[3] != input[3])) )
symbol_error_count++;
if ( ((output[4] != input[4])||(output[5] != input[5]))
||((output[6] != input[6])||(output[7] != input[7])) )
symbol_error_count++;
if ( ((output[0] != input[0])||(output[1] != input[1]))
||((output[2] != input[2])||(output[3] != input[3]))
||((output[4] != input[4])||(output[5] != input[5]))
||((output[6] != input[6])||(output[7] != input[7])) )
char_error_count++;

}
fclose(fptr1);
fclose(fptr2);

symbol_count = numxy - 1;
data_rate=(9.6*((float)(symbol_count)-(float)symbol_error_co
unt))/(float)symbol_count;

fprintf(fptr3,"\n\tV32 MODEM TRANSMISSION STATISTICAL
REPORT");
fprintf(fptr3,"\n\tFOR: %s",input_file);
fprintf(fptr3,"\n\t-----------------------------------------
");
fprintf(fptr3,"\n\n\tThe Viterbi Memory Length Used:
%d",vml);
fprintf(fptr3,"\n\tThe Noise Reduction Factor Used:
%.2f",nrf);
fprintf(fptr3,"\n\tMean Error From X Coordinate:
%.3f",xce/(numxy-1));
fprintf(fptr3,"\n\tMean Error From Y Coordinate:
%.3f",yce/(numxy-1));
fprintf(fptr3,"\n\tNumber Of Symbols Transmitted:
%d",numxy-1);
fprintf(fptr3,"\n\tNumber Of Bits Transmitted:
%d",bit_count);
fprintf(fptr3,"\n\tNumber Of Symbols Received In Error:
%d",symbol_error_count);
fprintf(fptr3,"\n\tNumber Of Bits Received In Error:
%d",bit_error_count);
fprintf(fptr3,"\n\tData Rate Achieved %.3f",data_rate);
fprintf(fptr3,"\n\tNumber Of ASCII Characters Transmitted:
%d",char_count);
fprintf(fptr3,"\n\tNumber Of ASCII Characters In Error:
%d",char_error_count);
fprintf(fptr3,"\n\tNumber Of Demodulation Faults:
%d\n\n\n",df);
printf("\n\tV32 MODEM TRANSMISSION STATISTICAL REPORT");
```

```
printf("\n\tFOR: %s",input_file);
printf("\n\t----------------------------------------");
printf("\n\n\tThe Viterbi Memory Length Used: %d",vml);
printf("\n\tThe Noise Reduction Factor Used: %.2f",nrf);
printf("\n\tMean Error From X Coordinate:
%.3f",xce/(numxy-1));
printf("\n\tMean Error From Y Coordinate:
%.3f",yce/(numxy-1));
printf("\n\tNumber Of Symbols Transmitted: %d",numxy-1);
printf("\n\tNumber Of Bits Transmitted: %d",bit_count);
printf("\n\tNumber Of Symbols Received In Error:
%d",symbol_error_count);
printf("\n\tNumber Of Bits Received In Error:
%d",bit_error_count);
printf("\n\tData Rate Achieved: %.3f kbits/s",data_rate);
printf("\n\tNumber Of ASCII Characters Transmitted:
%d",char_count);
printf("\n\tNumber Of ASCII Characters In Error:
%d",char_error_count);
printf("\n\tNumber Of Demodulation Faults: %d",df);
fclose(fptr3);
return 0;
}
```

```
/* FUNCTION
REPORT33(VML,NRF,XCE,YCE,NUMXY,INPUT_FILE,REPORT_FILE)*/
report33(vml,nrf,xce,yce,df,numxy,input_file,report_file)
int vml,df,numxy;
float nrf,xce,yce;
char input_file[25],report_file[25];
{
char output[8];
char input[8];
float data_rate;
int bit_count = 0,bit_error_count =0,char_count =
0,char_error_count = 0;
int symbol_error_count=0,symbol_count=0,ct,cnt;
FILE *fptr1,*fptr2,*fptr3;
fptr1 = fopen("c:\\modem\\binary.txt","r");
fptr2 = fopen("c:\\modem\\binary_o.txt","r");
fptr3 = fopen(report_file,"w");
for (cnt=0; cnt<numxy-1; cnt++)
{
for (ct =0; ct<6; ct++)
{
input[ct]=getc(fptr1);
output[ct] = getc(fptr2);
bit_count++;
}


if (output[0] != input[0])
bit_error_count++;
if (output[1] != input[1])
bit_error_count++;
if (output[2] != input[2])
bit_error_count++;
if (output[3] != input[3])
bit_error_count++;
if (output[4] != input[4])
bit_error_count++;
if (output[5] != input[5])
bit_error_count++;

if ( ((output[0] != input[0])||(output[1] != input[1]))
||((output[2] != input[2])||(output[3] != input[3]))
||((output[4] != input[4])||(output[5] != input[5])) )
symbol_error_count++;

}
fseek(fptr1,0,0);
fseek(fptr2,0,0);
```

```
while(numxy*.75 >= char_count)
{
     for (ct =0; ct<8; ct++)
     {
     input[ct]=getc(fptr1);
     output[ct] = getc(fptr2);
     }
   char_count++;
     if ( ((output[0] != input[0])||(output[1] != input[1]))
     ||((output[2] != input[2])||(output[3] != input[3]))
     ||((output[4] != input[4])||(output[5] != input[5]))
     ||((output[6] != input[6])||(output[7] != input[7])) )
     char_error_count++;
}
fclose(fptr1);
fclose(fptr2);

if (char_error_count == 1)
char_error_count = 0;

symbol_count = numxy - 1;
data_rate=(14.4*((float)symbol_count-(float)symbol_error_cou
nt))/(float)symbol_count;

fprintf(fptr3,"\n\tV33 MODEM TRANSMISSION STATISTICAL
REPORT");
fprintf(fptr3,"\n\tFOR: %s",input_file);
fprintf(fptr3,"\n\t-----------------------------------------
");
fprintf(fptr3,"\n\n\tThe Viterbi Memory Length Used:
%d",vml);
fprintf(fptr3,"\n\tThe Noise Reduction Factor Used:
%.2f",nrf);
fprintf(fptr3,"\n\tMean Error From X Coordinate:
%.3f",xce/(numxy-1));
fprintf(fptr3,"\n\tMean Error From Y Coordinate:
%.3f",yce/(numxy-1));
fprintf(fptr3,"\n\tNumber Of Symbols Transmitted:
%d",numxy-1);
fprintf(fptr3,"\n\tNumber Of Bits Transmitted:
%d",bit_count);
fprintf(fptr3,"\n\tNumber Of Symbols Received In Error:
%d",symbol_error_count);
fprintf(fptr3,"\n\tNumber Of Bits Received In Error:
%d",bit_error_count);
fprintf(fptr3,"\n\tData Rate Achieved %.3f",data_rate);
fprintf(fptr3,"\n\tNumber Of ASCII Characters Transmitted:
%d",char_count);
fprintf(fptr3,"\n\tNumber Of ASCII Characters In Error:
```

```
%d",char_error_count);
fprintf(fptr3,"\n\tNumber Of Demodulation Faults:
%d\n\n\n",df);
printf("\n\tV33 MODEM TRANSMISSION STATISTICAL REPORT");
printf("\n\tFOR: %s",input_file);
printf("\n\t----------------------------------------");
printf("\n\n\tThe Viterbi Memory Length Used: %d",vml);
printf("\n\tThe Noise Reduction Factor Used: %.2f",nrf);
printf("\n\tMean Error From X Coordinate:
%.3f",xce/(numxy-1));
printf("\n\tMean Error From Y Coordinate:
%.3f",yce/(numxy-1));
printf("\n\tNumber Of Symbols Transmitted: %d",numxy-1);
printf("\n\tNumber Of Bits Transmitted: %d",bit_count);
printf("\n\tNumber Of Symbols Received In Error:
%d",symbol_error_count);
printf("\n\tNumber Of Bits Received In Error:
%d",bit_error_count);
printf("\n\tData Rate Achieved: %.3f kbits/s",data_rate);
printf("\n\tNumber Of ASCII Characters Transmitted:
%d",char_count);
printf("\n\tNumber Of ASCII Characters In Error:
%d",char_error_count);
printf("\n\tNumber Of Demodulation Faults: %d",df);
fclose(fptr3);
return 0;
}
```