MOTOROLA

APR6/D

Convolutional Encoding and Viterbi Decoding

using the

DSP56001 with a V.32 Modem Trellis Example

Digital Signal Processors

Motorola's High-Performance DSP Technology

dsp

Table of Contents

Illustrations

List of Tables

SECTION 1

Introduction

"The DSP56001 has many features that make it possible to perform Viterbi decoding quickly and efficiently. . ."

Coding techniques have long been used for error correction to decrease the bit error rate (BER) in data transmission systems. This decrease in BER is accomplished by adding redundant data bits to the transmitted data bits and, in some cases, scrambling the order of the original data bits. There are many types of coding techniques (Hamming, BCH, and Reed-Solomon) used to correct different error phenomena that occur during data transmission (see Reference 1). This discussion is limited to convolutional encoding, a good method for correcting errors that occur during data transmission.

Convolutional encoders are implemented in the form of a shift register type of circuit with particular locations of the shift register exclusively ORed together to produce an output. Figure 1-1 shows one such implementation. The locations that are exclusively ORed together may be referred to as taps. The placement of these taps defines possible state transitions where the number of states for a particular code is defined by $2^{(k-1)}$ (see Reference 2). In this case, k is the constraint length of the code and is also the length of the shift register. The state transitions may also be represented by a trellis diagram. The trellis for the encoder of Figure 1-1 is shown in Figure 1-2 (see Reference 2).

Many data transmission systems use convolutional encoding when transmitting data; however, some systems use a method of transmission known as trellis coded modulation (TCM). TCM is a technique in which modulation and coding are combined (see Reference 3). One such application is phone-line channels being used to transmit higher and higher data rates on a power-limited 3kHz band-limited line.

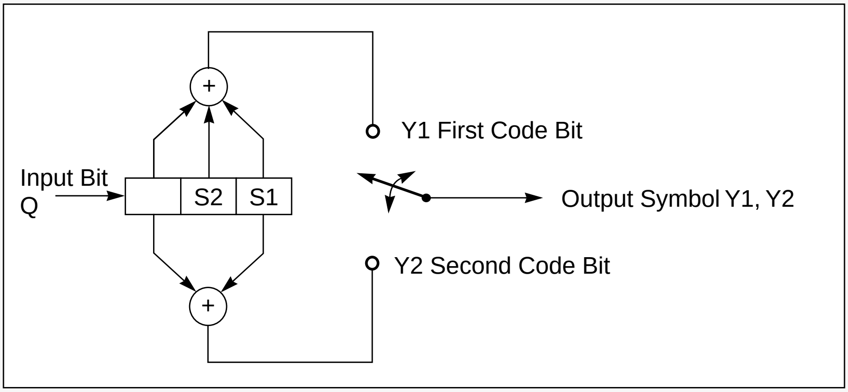


Figure 1-1 Convolutional Encoder Shift Register Implementation

There are several different methods for decoding convolutional codes: sequential decoding, threshold decoding, and Viterbi decoding. Practical applications of convolutional encoding became possible when Viterbi proposed a maximum-likelihood method for decoding convolutional codes in 1967 (see Reference 4). The Viterbi method is fast enough to allow real-time decoding for short constraint length (k) codes with high-speed processors (made possible by recent advances in VLSI technology). Long constraint length codes require so much path memory that it is not practical to use the Viterbi algorithm when decoding. This maximum-likelihood method is equivalent to a dynamic programming solution to the problem of finding the shortest path through a trellis (see Reference 1).

The Motorola digital signal processor (DSP56001) has the perfect architecture for communication channel modulators and demodulators. For example, the complexity of these components for high data rate modems has forced designers to find a digital signal processing chip solution to what once was an analog problem. In terms of design simplicity, the possibility of performing error correction and modem functions on the same chip has become very important. The DSP56001 has many features that make it possible to perform Viterbi decoding quickly and efficiently, allowing not only a single-chip solution to the high-speed modem application but also a single-chip solution for a higher speed Viterbi decoder. The dual memory architecture allows parallel moves concurrent with arithmetic operation, and the instruction set provides flexibility to easily program using this capability (see Reference 5). This feature of the DSP56001 is fully exploited in the included code.

Because the Viterbi algorithm is a dynamic programming model, a trellis has been chosen as the example for the report. Using this trellis, each step in designing the code is explained, giving enough description to duplicate or modify the existing code for a different trellis. This application note specifically addresses the trellis for the CCITT V.32 modem standard, which uses TCM consisting of quadrature amplitude modulation (QAM) combined with a differential and convolutional encoder (see Reference 3). The techniques described can be applied to any trellis, and the included DSP56001 code can be modified to work with any trellis, not only the V.32. ■

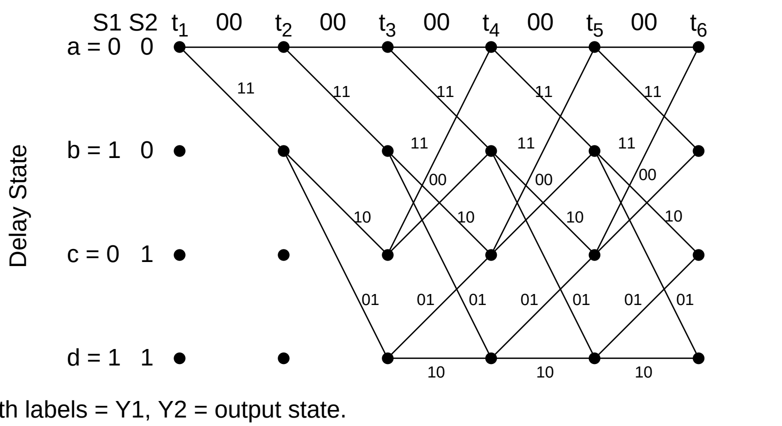


Figure 1-2 Trellis Representation

SECTION 2

Trellis (Convolutional) Encoding

2.1 Theory

"The input to the encoder is the parallel data stream $Q1_n$, $Q2_n$, $Q3_n$, and $Q4_n$."

A convolutional encoder is a function of the number of input bits (N) for the number of output bits (M) and the constraint length (K). N,M,K and the given generator function can completely describe a convolutional encoder. The generator function of the encoder is the impulse response of the encoder—that is, the output of the encoder when the input sequence is a one followed by zeros. Thus, the encoder equation can then be expressed as:

$$v = u * g \qquad \text{Eqn. 2-1}$$

where: * = convolution operation
v = output
u = input
g = generator polynomial (see Reference 1).

As shown, a convolutional encoder adds redundant bits to a signal data stream. Adding more bits generally increases the bit error rate (BER) since there are more encoder output bits per input bits to transmit with the same average power. When this happens, a reduction in the signal-to-noise ratio (SNR) occurs since there is less signal power per bit and, therefore, an increase in the BER. However, when a convolutional encoder adds a redundant bit to a data stream, it is done in such a way that the SNR is increased by allowing only certain transitions to occur. The coding gain associated with convolutional encoding is given by:

$$\mathrm{CodingGain} = 10\log_{10}\left[\left(d^2_{min}/P_{av}\right)\mathrm{encoded}\left(d^2_{min}/P_{av}\right)\mathrm{unencoded}\right]$$

Eqn. 2-2

where: d^2_{min} = the minimum distance between possible transitions

P_{av} = the average power of the signal for the encoded and unencoded cases, respectively (see Reference 6).

The minimum distance must now be examined. Figure 2-1 shows a 16QAM signal constellation, where four bits are required to represent one point on the constellation (see Reference 7). Figure 2-2 shows a 32 QAM constellation where five bits represent one point on the constellation (see Reference 7). For the 16QAM case, the minimum distance from one point to the next is 2.

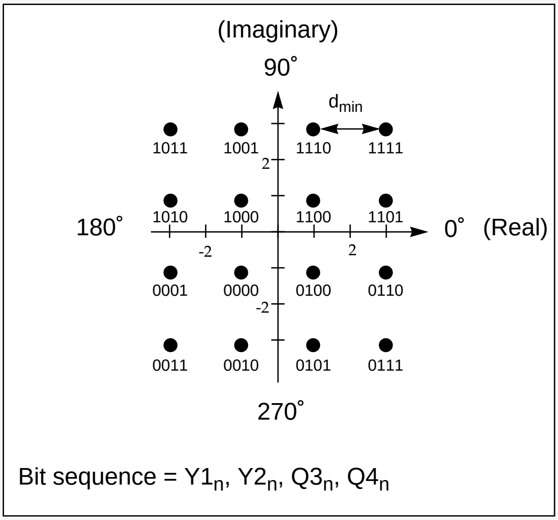


Figure 2-1 16QAM Constellation

For the 32QAM case in which the extra bit is a redundant bit obtained from a convolutional encoder so that the actual data rate is the same as the 16QAM case, the points are located such that the minimum distance is now $\sqrt{10}$. Note that transitions between nearest neighbors on an encoded constellation are not allowed in contrast to those on an unencoded constellation. This is how a larger d^2_{min} and, consequently, coding gain is realized with convolutional encoding. Substituting these values into Eqn. 2-2 results in a coding gain of approximately 4dB. For the 32QAM case using TCM, this would result in a decrease in the BER over the unencoded 16QAM case for the same average signal power.

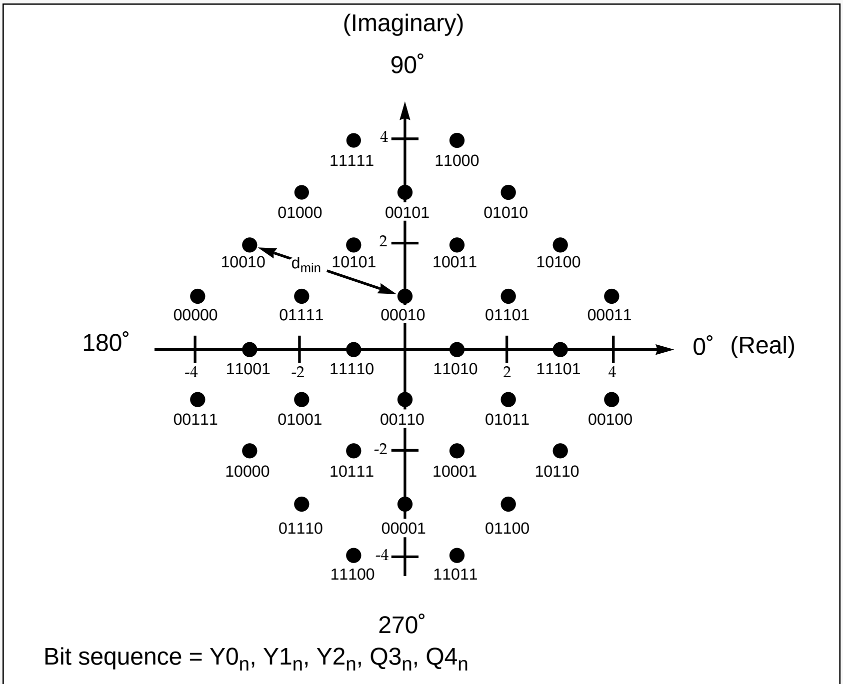


Figure 2-2 32QAM Constellation

Figure 2-3 shows the nonlinear convolutional and differential encoder for the V.32 standard (see Reference 7). The input to the encoder is the parallel data stream $Q1_n$, $Q2_n$, $Q3_n$, and $Q4_n$. The first function performed is differential encoding (see Table 2-1), which provides 90 degrees of phase invariance (see Reference 6). This means that $Q3_n$ and $Q4_n$ are the same for points on the 32QAM constellation that are 90 degrees from each other. The convolutional encoder has eight states resulting from the three delays: S1, S2, and S3. State 1 (S1) is the rightmost state, with state 2 (S2) and state 3 (S3) to the left, respectively. The output of the encoder is $Y0_n$, $Y1_n$, $Y2_n$, $Q3_n$, and $Q4_n$ where $Y0_n$, $Y1_n$, $Y2_n$ are now considered a path when referring to the trellis. The trellis for the convolutional encoder is shown in Figure 2-4 (see Reference 8).

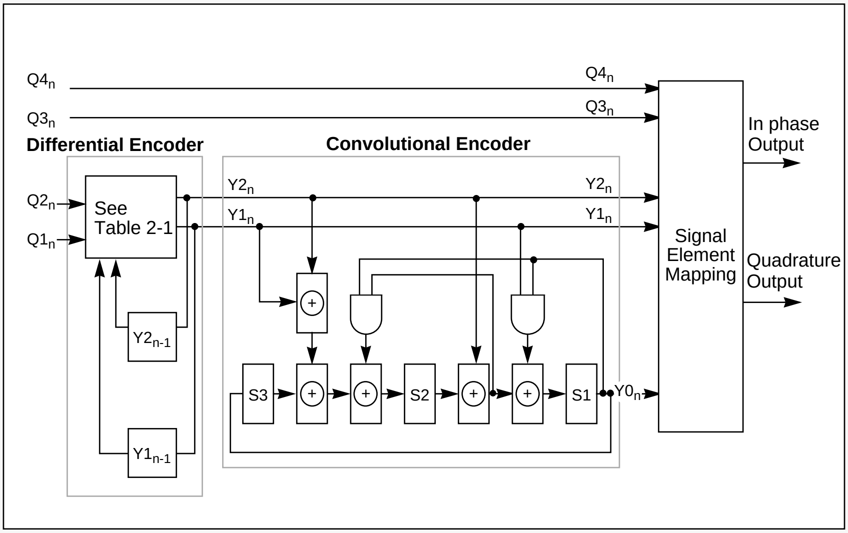


Figure 2-3 V.32 Encoding Diagram

Table 2-1: Differential Encoder

Input Bits		Past Output Bits		Output Bits	
$Q1_n$	$Q2_n$	$Y1_{n-1}$	$Y2_{n-1}$	$Y1_n$	$Y2_n$
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

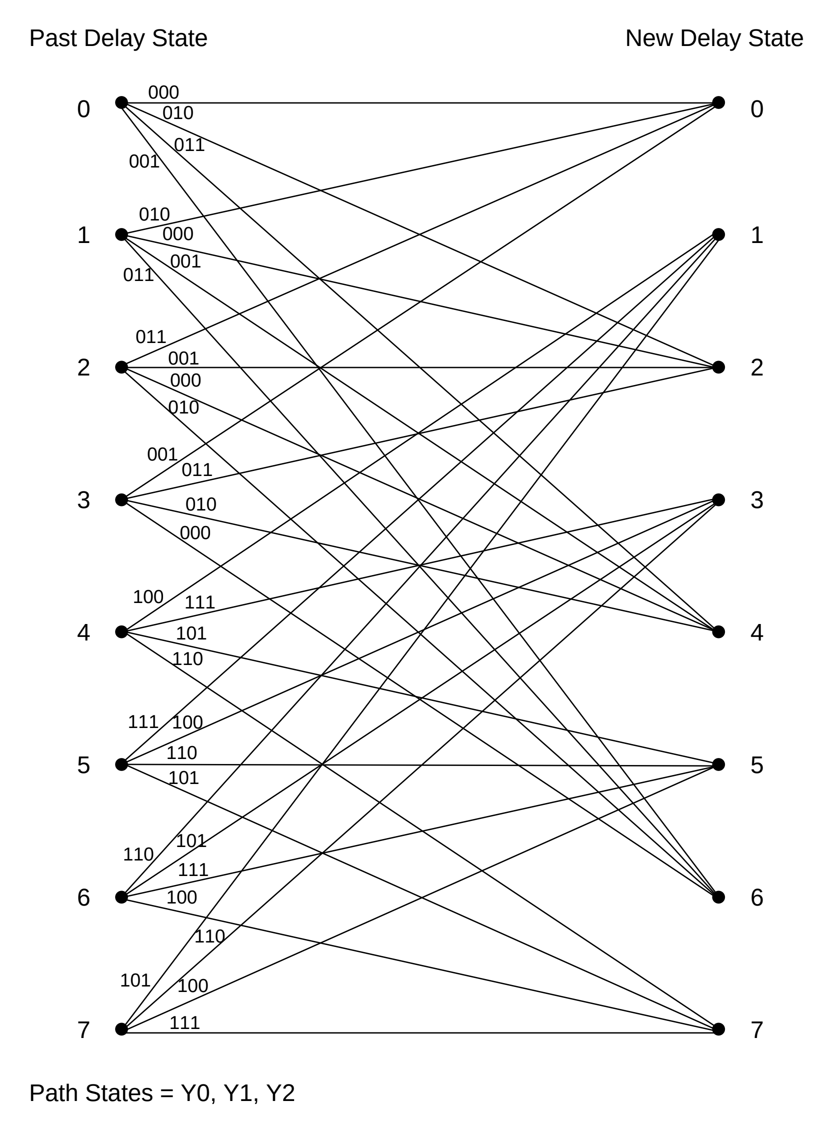


Figure 2-4 V.32 Trellis Diagram

2.2 Implementation

Implementing the encoder of Figure 2-3 on the DSP56001 is relatively simple. The differential encoder is implemented by storing the previous outputs of the differential encoder and then performing the appropriate exclusive OR ($\underline{\vee}$) and AND (Λ) functions defined by:

$$Y1_n = Q1_n \ \underline{\vee}\ Y1_{n-1} \qquad \text{Eqn. 2-1}$$

$$Y2_n = (Q1_n \ \Lambda\ Y1_{n-1}) \ \underline{\vee}\ Y2_{n-1} \ \underline{\vee} \qquad \text{Eqn. 2-2}$$

The convolutional encoder is implemented in much the same way. There are three delays (S1, S2, and S3); each requiring separate memory locations. The information from each delay is used at each input and then updated, based on the configuration of Figure 2-3. The output ($Y0_n$) at each time period is the value of delay 1 (S1) before it is updated. Figure 2-5 shows a flowchart of the encoding process. S1, S2, and S3 are referred to as the delay state of the encoder and the decoder; $Y0_n$, $Y1_n$, $Y2_n$ are referred to as the path state of the encoder and the decoder. ■

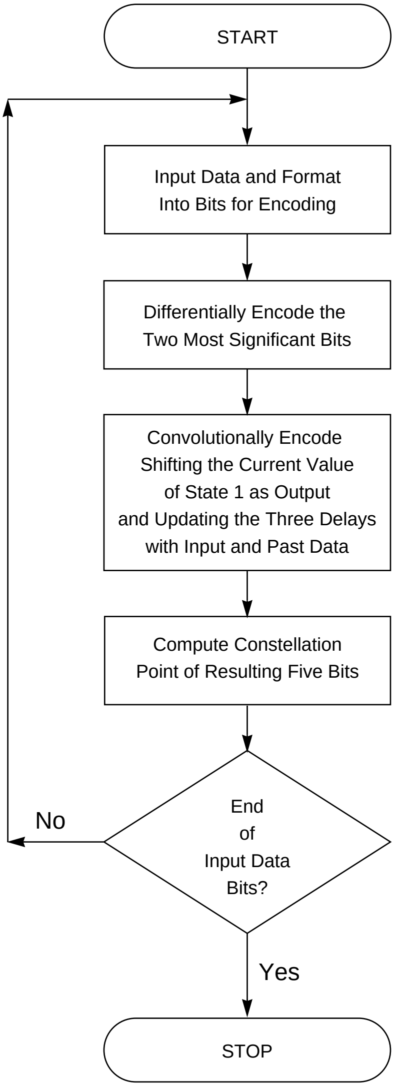


Figure 2-5 Encoding Flowchart

SECTION 3

Viterbi Decoding

3.1 Theory

"Decoding must be done by performing each decoder function in the reverse order in which it was encoded."

The Viterbi algorithm for decoding uses the structure of the trellis (i.e., the allowed transitions) and the input data to determine the most likely path through the trellis. The output for time (t_0) reflects a decision made by the decoder on data received up to N time periods in the future. This means that the output for time (t_0) is necessarily delayed by N time periods or that the latency of the decoder is N time periods. N is determined by the constraint length of the code and, for near-optimum decoding, is four or five times the constraint length (see Reference 9).

The most likely path through the trellis is one that is a minimum-distance path for the input data or the path closest to the received data in Euclidean distance. In other words, the Viterbi algorithm minimizes the distance (see Reference 1):

$$d(r,v) = \sum_{i=0}^{N-1} d(r_i,v_i) \qquad \text{Eqn. 3-1}$$

where: r_i and v_i are the received and the decoded signal sequence, respectively

At each time period, every delay state in the trellis can have several paths (defined by each trellis) going into it, but only one will be the minimum distance for that delay state. Thus, the beginning point at each time period is the delay state with the smallest accumulated distance to trace the minimum-distance path through the past N-1 time periods of the trellis. Next, the algorithm determines the minimum-distance paths to the next delay state by evaluating the input to determine which point on the constellation in each path is closest; determining the Euclidean distance to each of those points; and then, based on the trellis structure and the minimum-distance paths, determining the minimum distance to each delay state. After defining the trellis, the steps taken to decode the data are as follows (see Reference 1):

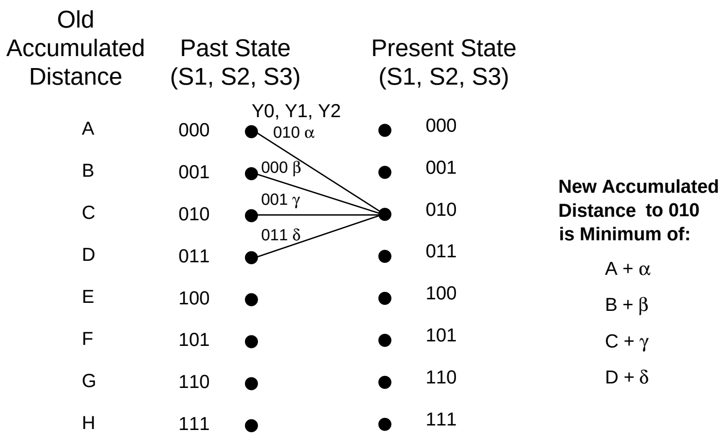
1. Compute the minimum-distance path states at each input and the corresponding Euclidean distances and store them for each path state.

2. Compute the accumulated distance to each delay state by adding the distance for each path state going into a delay state to the distance of the delay state where the path state originated, keeping the smallest of these distances and storing the path state and the delay state from which it came. Eliminate all other path states going into that delay state.

3. Find the delay state with the smallest accumulated distance and trace it back N times to read the path state, which is the output of the decoder for that time period.

Figure 3-1 shows the possible paths to delay state 010 for the V.32 trellis and how the minimum distance to 010 is chosen from the possible paths.

When the minimum-distance path is found at each delay state, the path state taken to get there from the last delay state must also be stored (i.e., 001 in Figure 3-1 assuming C+γ was the minimum) so that, in N time periods, the output can be determined from the endpoint of the minimum-distance path at time t_0 + N. The most likely path can be traced by storing the minimum-distance path state ($Y0_n$, $Y1_n$, $Y2_n$) to each delay state as well as the path originating the delay state (S1, S2, S3).



NOTE: α, β, γ, δ are path distances.

Figure 3-1 Possible Paths to State 010
These paths are calculated and used to determine the minimum-distance path.

This tracing is done by starting at the minimum accumulated distance delay state, back-tracking to the delay state it came from, and repeating this process N-1 times. That is, the minimum accumulated distance for all eight states identifies the state to be used as the starting point from which to trace back N time periods.

Once the state for t_0 is found, the path taken to get to that state becomes the output of the decoder for the time period t_0. For instance, in Figure 3-1, if at t_0, the end point of the minimum-distance path was 010, then the output of the Viterbi decoder would be 001 if C + γ was the minimum-distance path.

At every time period, the accumulated distance to each delay state is calculated and updated, and the minimum-distance path state ($Y0_n$, $Y1_n$, Y2n) to each delay state is stored as well as the delay state it came from (S1, S2, S3). Storing this data creates a history, making it possible to trace back to get the correct output of the decoder.

A block diagram of the V.32 decoder showing inputs and outputs is illustrated in Figure 3-2. It can be compared to the block diagram of the encoder shown in Figure 2-3 to keep track of the input and output bit order. Decoding must be done by performing each decoder function in the reverse order in which it was encoded. In this case, the trellis decoding is performed before the differential decoding.

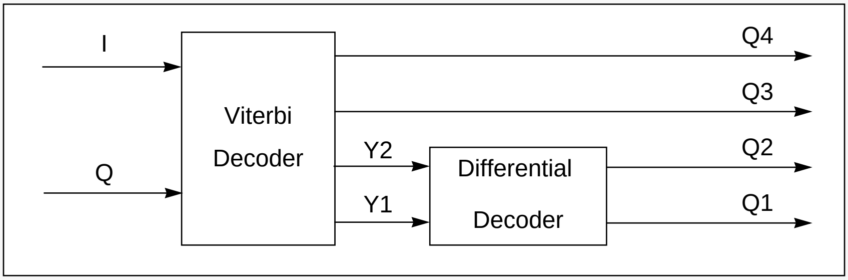


Figure 3-2 V.32 Decoder Block Diagram

3.2 Implementation

Implementing the Viterbi decoder on the DSP56001 involves:

1. Segregating memory locations properly
2. Recognizing boundary conditions of the trellis
3. Utilizing the modulo addressing capability of the DSP56001

First, a brief description of the functions for the decoder must be analyzed to realize the importance of the three previously mentioned ideas. At initialization, the x and y components of the points on the constellation are stored in memory for distance computations. For every input, the Euclidean distance to the closest point in each path state is computed.

After computing this distance, the minimum accumulated distance to each delay state can be computed. Then, the minimum-distance delay state is used as the starting point to trace back to find the output of the previous 16 time periods. Recall that the number of time periods needed should be four or five times the constraint length (in this case, K=4). Since four times the constraint length in this case is 16(4 x K), this makes modulo addressing easier than using 20(5 x K) because 16 is a power of two. Once the output delay state is found, the closest point in the path state to the original input at that time period is computed and is the output of the decoding process. Figure 3-3 shows the decoder flowchart. Each of these functions is discussed separately in the following sections. The decoder code is included in **APPENDIX B DSP56001 Decoding Program Listing**.

3.2.1 Initialization

During initialization, the x and y components of the constellation points are stored in internal memory since they are accessed frequently. The modulo settings for other parts of the code are also set here. All distance tables must be initialized as well. Since all paths should begin at the 000 delay state, this accumulated distance location should contain the value zero, assuming that the initial conditions of the delays in the encoder are 000. In practical implementation, it is important to assume that the path started from state 000. Setting all other accumulated distances to a large value will ensure that the path starts at 000. Figure 3-4 shows a memory map for the decoder.

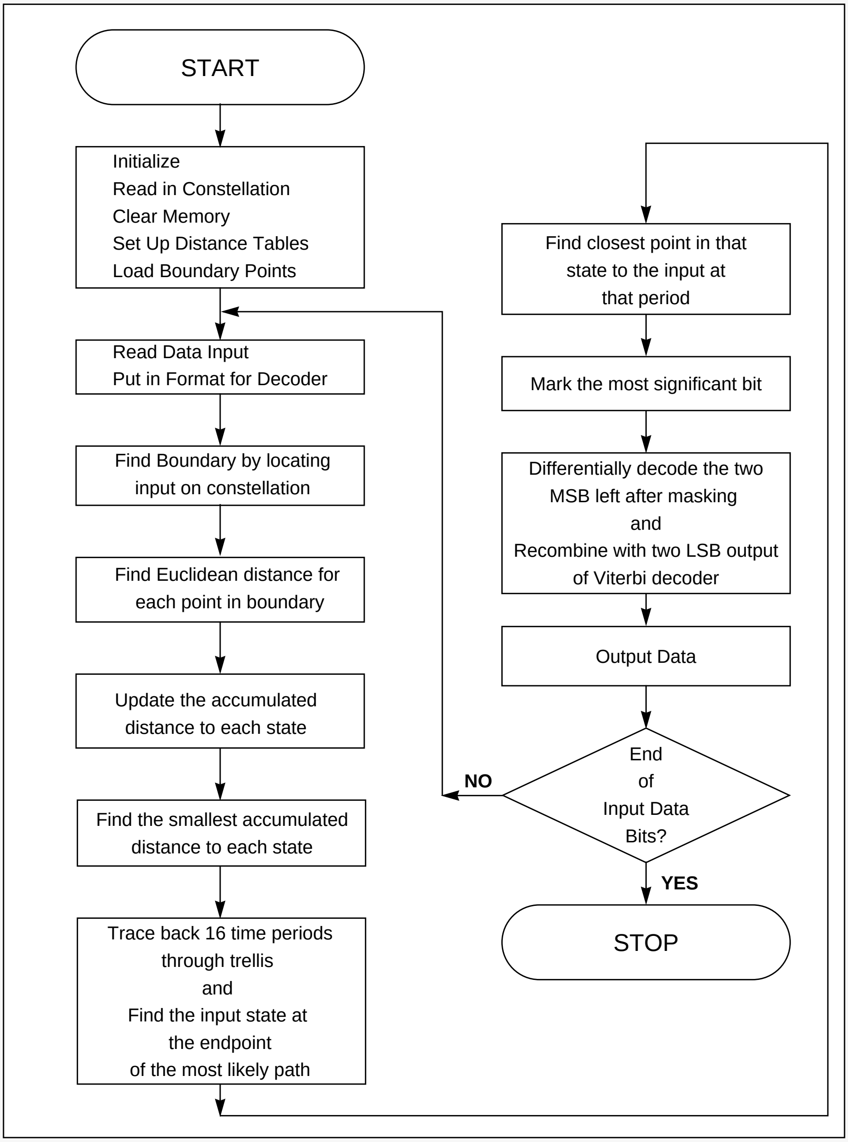


Figure 3-3 Decoder Flowchart

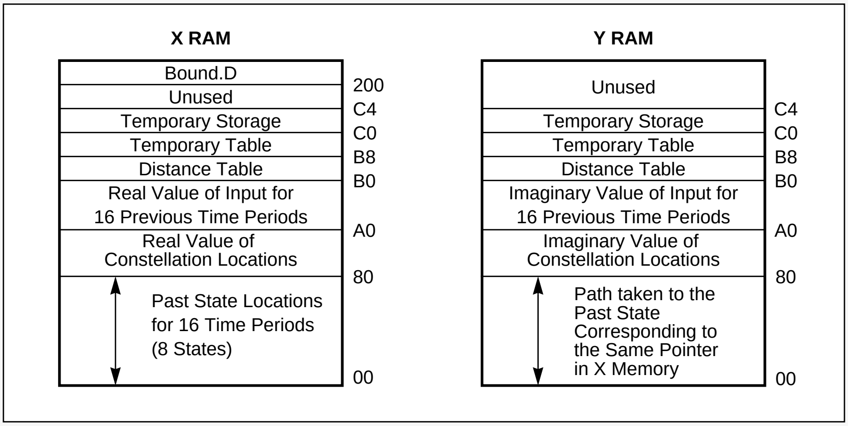


Figure 3-4 Decoder Memory Map used by the DSP56001 program listing in Appendix B

3.2.2 Finding the Minimum Distance

This routine analyzes the input data point and determines the Euclidean distance to the closest point in each of the eight path states. The Euclidean distance is defined to be:

$$d = \sqrt{(x_c - x_i)^2 + (y_c - y_i)^2} \qquad \text{Eqn. 3-2}$$

where: x_c and y_c are the x and y coordinates of the point on the constellation and

x_i and y_i are the coordinates of the input data.

This calculation can be done by computing the Euclidean distance to each point in each path state and only keeping the smallest computed distance for that path state; however, there is a more efficient way to calculate this distance (see Reference 8). Looking at Figure 3-5 for the location of points for path state ($Y0_n$, $Y1_n$, $Y2_n$) 110, the boundaries that equally separate the four points are shown as dashed lines.

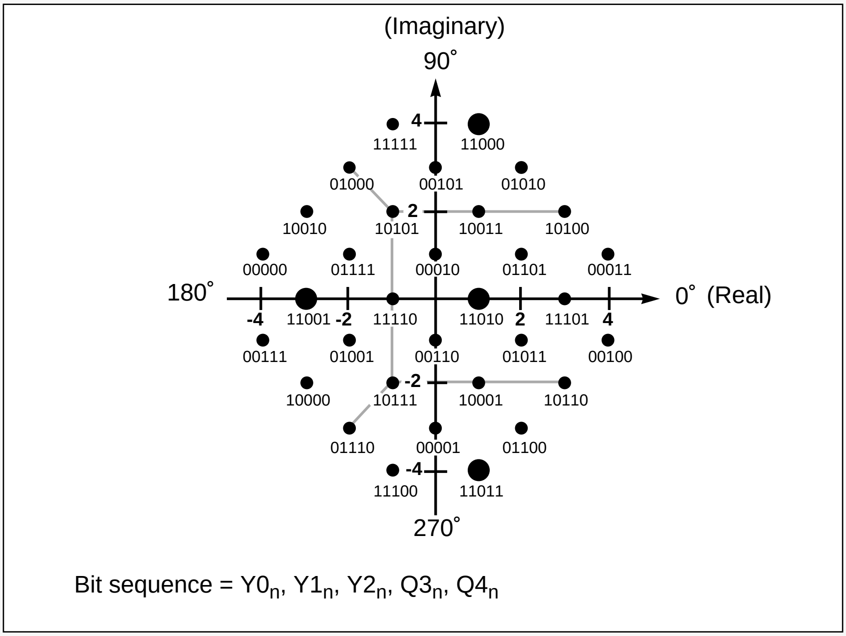
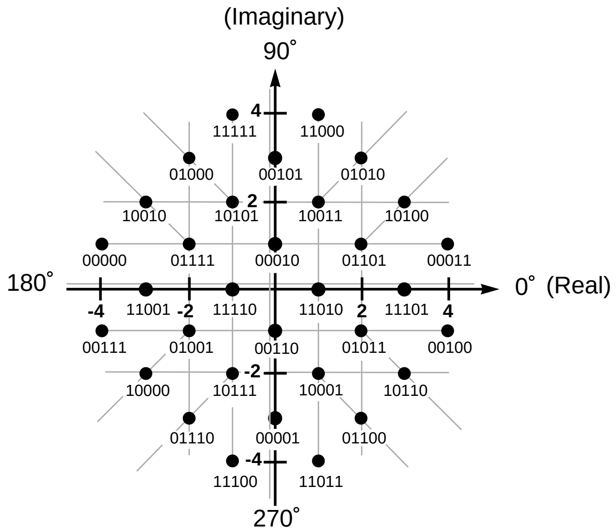


Figure 3-5 Boundary for State 010

Figure 3-5 shows which point for this path state is closest to the input using these boundaries. Superimposing the boundaries for all eight path states creates the boundaries shown in Figure 3-6. There are 52 bounded areas on the constellation. For every bonded area, there is a unique set of eight points corresponding to the closest point for every path state should the input fall in the bounded area. For example, assuming that the input fell into region 6 in Figure 3-6, then the eight closest points would be 00010, 00101, 01010, 01101, 10011, 10101, 11000, and 11111.

Once the bounded area of the input is determined, then the smallest Euclidean distance for every path is just the Euclidean distance to that point for each path state determined by the boundary condition, which reduces the execution time considerably.

These distances are then stored for use in the next part of the code. The detailed discussion for their storage order is presented in **SECTION 3.2.3 Finding the Accumulated Distance to Each State**. The eight points for each of the 52 bounded areas are loaded into memory at the beginning of the decoding process by loading the bound.d file (see **APPENDIX C DSP56001 Bound.D Data File**). Since there are eight path states in each of the 52 areas, this requires 416 words of data memory. The larger memory is not needed if the much slower direct approach is used — namely, finding the distance to all 32 points in the constellation. A decision must be made to optimize either speed or memory usage. The included code uses the fast version, which requires more memory.



Bit sequence = $Y0_n$, $Y1_n$, $Y2_n$, $Q3_n$, $Q4_n$

Figure 3-6 Superimposed Boundaries for All States

3.2.3 Finding the Accumulated Distance to Each State

Close analysis of the V.32 trellis in Figure 2-4 reveals that there are a limited number of path states (four) to each new delay state from the previous time period. Table 3-1 identifies the combination of previous delay states and path states to reach each delay state for the current time period. If this table is viewed as memory, it shows that, by arranging the data as illustrated, the code needed to update the appropriate delay state can be minimized. Even delay states are only reached by the previous delay states 000, 001, 010, 011, and odd delay states are only reached by the previous delay states 100, 101, 110, and 111.

Table 3-1 Minimum Path Table

New State S1, S2, S3	Old State S1, S2, S3	Path Y0, Y1, Y2	New State S1, S2, S3	Old Start S1, S2, S3	Path Y0, Y1, Y2
000	000	000	001	100	100
000	001	010	001	101	111
000	010	011	001	110	110
000	011	001	001	111	101
010	000	010	011	100	111
010	001	000	011	101	100
010	010	001	011	110	101
010	011	011	011	111	110
100	000	011	101	100	101
100	001	001	101	101	110
100	010	000	101	110	111
100	011	010	101	111	100
110	000	001	111	100	110
110	001	011	111	101	101
110	010	010	111	110	100
110	011	000	111	111	111

Similarly, the path state to each new even delay state can only be 000, 001, 010, 011, and the path state for each new odd delay can be only be 100, 101, 110, 111. In the even case, if the path states are put in the order 000, 010, 011, 001 by incrementing in two cases (for states 0 and 4) and decrementing in two cases (for states 2 and 6), they are easily stepped through when computing the minimum accumulated distance to each delay state. For delay state 2 (010), the pointer is initialized to the second location (path state 010) prior to decrementing; for delay state 4, the pointer is initialized to the third location (path state 011) prior to incrementing; and for delay state 6, the pointer is initialized to the fourth location (path state 001) prior to decrementing.

In practice, it is impossible to continue to accumulate these distances without running into an overflow problem. Thus, an alternate way to obtain the accumulated distance measurement is a weighted accumulation method, which can be expressed as (see Reference 10):

$$d_{new} = \beta d_{old} + (1 - \beta) d_{path} \qquad \text{Eqn. 3-3}$$

where: 0<<ß<1 denotes the smoothing parameter

This method (essentially a low-pass filter) ensures that the newly accumulated distance is a bounded arithmetic value. This method has been shown to be unbiased estimates (see Reference 10.) Although

Eqn. 3-3 uses all past values to compute a current accumulated distance, the value of β is directly related to the time constant, τ, which gives the number of recent past values to estimate the accumulated distance as:

$$\tau \approx \frac{2}{1-\beta} \qquad \text{Eqn. 3-4}$$

Using this equation, 85% of d_{new} comes from the points in the time constant, τ, and the remaining 15% is contributed by points previous to τ. The value of $\beta = 0.9$ was used for this code and simulation; however, it should be considered a performance parameter that can be modified for different applications and performance specifications.

At several points in the code, two paths can have the same minimum distance. In these situations, an arbitrary choice is made to keep one or the other path.

In the included code, the arbitrary choice has been made to keep the last path found; however, it could be changed to pick the first path encountered or the first path in even cases and the last path in odd cases. The performance of the included code is not affected by changing this arbitrary choice (see Reference 1).

Once the minimum distance for each state is accumulated, the path taken to get there and the previous state are stored in memory to be used when tracing back through the trellis to determine an output. Since the memory is 16 time periods deep and there are eight states, the memory is set up to be 128 words in each memory. The previous state information is stored in a circular buffer in x memory, and the path information is stored in a circular buffer in y memory. In this manner, the new states and paths will overwrite the oldest information in the buffer each time period.

3.2.4 Traceback

Tracing back through 16 time periods along the most likely path is now done by taking advantage of the fact that the memory is set up to be circular around 128 points. Starting at the most current location in which the latest paths for each state were updated as well as the state from which the path came, the memory is decremented by eight to the previous time period. When this is done, the pointer is then updated (a number between zero and seven) to correspond with the previous state from which the most likely path came. The information stored at this pointer location is the state information for the next time period. This procedure is done until the last time period in which the path is read, instead of the state from which it came. This path determines the output of the decoder.

3.2.5 Data Out

By taking the path found at the end of traceback, the output of the Viterbi decoder can be determined. The path corresponds to one of eight paths

($Y0_n$, $Y1_n$, $Y2_n$), and the least significant bits correspond to the unencoded bits ($Q3_n$ and $Q4_n$). The closest of the four points from that path to the input at that time period is the output of the decoder. The most significant bit ($Y0_n$) is stripped off at this point (the redundant bit added in the coding process).

3.2.6 Differential Coding

The decoding of the differential encoding is now performed by taking the two most significant bits of the Viterbi decoder output to perform the following:

$$Q1_n = Y1_n \veebar Y1_{n-1} \qquad \text{Eqn. 3-5}$$

$$Q2_n = (Q1_n \wedge Y1_{n-1}) \veebar Y2_{2n-1} \veebar Y2_n \qquad \text{Eqn. 3-6}$$

where:
- $Y1_n$ and $Y2_n$ are the most significant bit of the Viterbi decoder output.
- $Q1_n$ and $Q2_n$ are now combined with the two least significant bits of the Viterbi output to complete the decoding process.

SECTION 4

Summary

"Modifying the code... to work with any trellis diagram should be fairly straightforward."

The included decoder code takes approximately 700 instruction cycles for every four input bits. For the V.32 case, this is about 15% of the processor. Modifying the code in ***APPENDIX A DSP56001 Encoding Program Listing*** and ***APPENDIX B DSP56001 Decoding Program Listing*** to work with any trellis diagram should be fairly straightforward. Modification is accomplished by going through each step explained; defining the memory, order of storage, and modulo settings for the new trellis. Boundary conditions obviously change, causing a need for ***APPENDIX C DSP56001 Bound.D Data File*** to be redone for any new constellation.

Hopefully, this application note gives enough explanation and description on how to implement a convolutional encoder with a Viterbi decoder on the DSP56001 for any trellis. Textbooks and fundamental papers dealing with coding theory are listed in References 11,12, and 13 for convenience. ■

APPENDIX: A

DSP56001 Encoding Program Listing

```
;This is a convolutional encoder for the V.32 which takes it's input from
;a file and and tests the output for all states as well as well as inputs.
locate equ  $ee

       statement   equ      60
       output      equ      50
       input       equ      40
       start       equ      $40
                   org      p:start
                   move     #statemem,r3
                   do       #104,code
                   move     #input+3,r2
                   move     #locate,r6
                   move     #output,r5
                   move     y:(r6),a
                   move     #>$1,x0
                   do       #4,loop
                   and      x0,a a,x1
                   move     a1,x:(r2)-
                   move     x1,a
                   asr      a
       loop
                   jsr      encode
                   move     #locate+1,r6
                   move     #input,r2
                   clr      b
                   clr      a    y:(r4),b0
                   addl     b,a
                   do       #4,loop2
                   move     x:(r2)+,b0
                   addl     b,a
       loop2
                   move     a0,y:(r6)
       code
       encode
                   move     #input,r0
                   move     #output,r4
                   move     #statemem,r1
                   move     x:(r0)+,x1
                   move     x:(r1)+,a
                   move     a,y:(r4)
                   and      x1,a x:(r0),x0
                   move     x:(r1)-,b
                   eor      x0,b a,y0
                   eor      y0,b b,y1
                   move     b,x:(r1)+y:(r4),b
```

Figure A-1 DSP56001 Encoding Program Listing (sheet 1 of 2)

```
        and     y1,b x0,a
        move    (r1)+
        eor     x1,a x:(r1),x0
        eor     x0,a y:(r4),y1
        move    b,y0
        eor     y0,a y1,x:(r1)-
        move    a,x:(r1)+
        rts
end
```

Figure A-1 DSP56001 Encoding Program Listing (sheet 2 of 2)

APPENDIX: B

DSP56001 Decoding Program Listing

```
;This program is a Viterbi Decoder for V.32.  There is a 16
;time period delay which will approach the maximum possible
;gain for this type of encoder.

                   page 132,66,3,3,0
                   opt  cex
                   org  l:$0000

       period      dsm  128
       location    dsm  32
       input       dsm  16
       tables      dsm  8
       temp        dsm  8

       endlong     equ  *
                   org  x:endlong
       storr6      ds   1
                   org  x:512
       boundry1    ds   32
       boundry2    ds   32
       boundry3    ds   32
       boundry4    ds   32
       boundry5    ds   32
       boundry6    ds   32
       boundry7    ds   32
       boundry8    ds   32
       boundry9    ds   32
       boundry10   ds   32
       boundry11   ds   32
       boundry12   ds   32
       boundry13   ds   32

       start       equ  $40
       four        equ  $200000
       three       equ  $180000
       two         equ  $100000
       one         equ  $080000
       zero        equ  $000000
       mone        equ  $f80000
       mtwo        equ  $f00000
       mthree      equ  $e80000
       mfour       equ  $e00000

       large       equ  .9
       small       equ  .1
       offset      equ  $010000
```

Figure B-1 DSP56001 Decoding Program Listing (sheet 1 of 11)

```
                        org  p:start
                        jsr  initialize
                        do   #115,endrun
                        jsr  readdata
                        jsr  findmindist
                        jsr  accumdist
                        jsr  traceback
                        jsr  outputdata
       endrun

;this initialization routine initializes register and modifiers
;as well as clearing the memeory.
;the constellation is also loaded into memory here.
;the accumulated distance array is set so that state zero starts out
;at a value of zero and all others start out larger, forcing the paths
;to merge at the zero states.

       initialize
                   move     #127,m1
                   move     #127,m5
                   move     #15,m6
                   move     #0,r1
                   clr      b       #$0,r0
                   clr      a    r0,r5
                   do       #256,clrmem
                   move     a,x:(r0)+b,y:(r5)+
       clr  mem
                   move     #tables+1,r7
                   move     #$400000,a1
                   rep      #7
                   move     a1,x:(r7)+

                   move     #input,b1
                   move     b1,x:storr6


;Now load full scale values of the constellationin the table location.


                   move     #location,r0
                   move     r0,r4
                   move     #mfour,a
                   move     #one,b
                   move     a,x:(r0)+b,y:(r4)+
                   move     #zero,a
                   move     #mthree,b
                   move     a,x:(r0)+b,y:(r4)+
                   move     #one,b
                   move     a,x:(r0)+b,y:(r4)+
                   move     #four,a
                   move     a,x:(r0)+b,y:(r4)+
                   move     #mone,b
                   move     a,x:(r0)+b,y:(r4)+
                   move     #zero,a
                   move     #three,b
                   move     a,x:(r0)+b,y:(r4)+
```

Figure B-1 DSP56001 Decoding Program Listing (sheet 2 of 11)

```
        move    #mone,b
        move    a,x:(r0)+       b,y:(r4)+
        move    #mfour,a
        move    a,x:(r0)+       b,y:(r4)+
        move    #mtwo,a
        move    #three,b
        move    #mone,y1
        move    a,x:(r0)+       b,y:(r4)+
        move    a,x:(r0)+       y1,y:(r4)+
        move    #two,a
        move    a,x:(r0)+       b,y:(r4)+
        move    a,x:(r0)+       y1,y:(r4)+
        move    #one,b
        move    #mthree,y1
        move    a,x:(r0)+       y1,y:(r4)+
        move    a,x:(r0)+       b,y:(r4)+
        move    #mtwo,a
        move    a,x:(r0)+       y1,y:(r4)+
        move    a,x:(r0)+       b,y:(r4)+
        move    #one,a
        move    a,x0
        move    #mthree,a
        move    #two,b
        move    b,y0
        move    #mtwo,b
        move    a,x:(r0)+       b,y:(r4)+
        move    x0,x:(r0)+      b,y:(r4)+
        move    a,x:(r0)+       y0,y:(r4)+
        move    x0,x:(r0)+      y0,y:(r4)+
        move    #three,a
        move    a,x0
        move    #mone,a
        move    x0,x:(r0)+      y0,y:(r4)+
        move    a,x:(r0)+       y0,y:(r4)+
        move    x0,x:(r0)+      b,y:(r4)+
        move    a,x:(r0)+       b,y:(r4)+
        move    #one,a
        move    #zero,b
        move    b,y0
        move    #four,b
        move    a,x:(r0)+       b,y:(r4)+
        move    #mthree,x0
        move    x0,x:(r0)+      y0,y:(r4)+
        move    a,x:(r0)+       y0,y:(r4)+
        move    #mfour,b
        move    a,x:(r0)+       b,y:(r4)+
        move    #mone,a
        move    a,x:(r0)+       b,y:(r4)+
        move    #three,x0
        move    x0,x:(r0)+      y0,y:(r4)+
        move    a,x:(r0)+       y0,y:(r4)+
        move    #four,b
        move    a,x:(r0)+       b,y:(r4)+
        rts
```

Figure B-1 DSP56001 Decoding Program Listing (sheet 3 of 11)

```
;readdata reads in the data from a simulator file.  Since it is read in as
;a point on the constellation, it must be converted to real and imaginary
;componenets by indexing into a table.
;it is also offfset by a value "offset" so it is not considered to be perfect
;data.

       readdata
            move                             y:$efe,a
            move            a,n2
            move            #location,r2
            move            x:storr6,r6
            lua    (r2)+n2,r4
            move            #>offset,x0
            move            x:(r4),a
            add    x0,a                      y:(r4),b
            add    x0,b     a,x:(r6)
            move                             b,y:(r6)+
            move            r6,x:storr6
            rts

;the minimum distance is found to the closest point in every state and stored.
;the values are stored so that indexing is made easier, state 0,2,3,1,4,7,6,5.
;this will greatly reduce the number of cycles needed later.
;a smoothing function is used to accumulate distances in the accumulated
;table so this minimum distance is multiplied by .1.

       findmindist

            move            x:-(r6),a
            move            #one,x0
            cmpm   x0,a                      y:(r6),b
;x>1
            jgt    <bigone
            cmpm   x0,b     #boundry1,r2

;x<1,y<1, load r2 with boundry 1 and continue

            jlt    <continue
            move            #two,x1
            cmpm   x1,b     #boundry4,r2

;x<1,y>1andy<2, load r2 with boundry4, go on

            jlt    <continue
            move            #boundry6,r2

;x<1,y>2, load r2 with boundry6 and continue

            jmp    <continue
bigone      move            #two,x1
            cmpm   x1,a

;x>2, jmp to that case
            jgt    <bigtwo
            cmpm   x0,b     #boundry2,r2

;x>1 ans x<2, y<1 load boundry2 and continue
```

Figure B-1 DSP56001 Decoding Program Listing (sheet 4 of 11)

```
            jlt    continue
            cmpm   x1,b      #boundry5,r2


;x>1,y<2 load boundry 5 and continue

            jlt    <continue

bigtwo      cmpm   x0,b      #boundry3,r2

;x>2 and y<1 so load boundry3 and continue

            jlt    <continue

            abs    a         #two,y0
            abs    b         a,x1
            sub    y0,a      b,y1
            sub    x0,b
            cmpm   a,b       y1,b
            jgt    <greatery1
            cmp    y0,b      #boundry7,r2

            jlt    <continue
            move             #boundry12,r2
            jmp    <continue

greatery1   sub    y0,b      x1,a
            sub    x0,a
            cmpm   a,b       x1,a
            jgt    <greatery2
            cmp    y0,a      #boundry10,r2
            jlt    <continue
            move   y1,b
            cmp    y0,b      #boundry11,r2
            jlt    <continue
            move             #boundry9,r2
            jmp    <continue

greatery2   cmp    y0,a      #boundry8,r2
            jlt    <continue
            move             #boundry13,r2

continue    clr    a         x:(r6),x1
            cmp    x1,a      y:(r6),y1
            jgt    <negx
            cmp    y1,a      #24,n2
            jgt    <posxnegy
posxposy    jmp    <findist
posxnegy    move             x:(r2)+n2,x0    ;update r2 by 24
            jmp    <findist
negx        cmp    y1,a      #8,n2
            jgt    <negxnegy
negxposy    move             x:(r2)+n2,x0    ;update r2 by 8
            jmp    <findist
negxnegy    move             x:(r2)+n2,x0    ;update r2 by 16
            move             x:(r2)+n2,x0
```

Figure B-1 DSP56001 Decoding Program Listing (sheet 5 of 11)

```
findist
          move            x:(r2)+,r0
          move            #tables,r4
          move            x:(r0),a
          sub    x1,a                        y:(r0),b
          sub    y1,b     a,x0
          mpy    x0,x0,a  b,y0
          mac    y0,y0,a  x:(r2)+,r0
          move            #small,x0a,y0
          mpy    x0,y0,a
          move            x:(r0),a           a,y:(r4)+
          sub    x1,a                        y:(r0),b
          sub    y1,b     a,x0               y:(r4)+,y0
          mpy    x0,x0,a  b,y0
          mac    y0,y0,a  x:(r2)+,r0
          move            #small,x0a,y0
          mpy    x0,y0,a                     y:(r4)+,b
          move            x:(r0),a           a,y:(r4)-
          sub    x1,a                        y:(r0),b
          sub    y1,b     a,x0               y:(r4)-,y0
          mpy    x0,x0,a  b,y0
          mac    y0,y0,a  x:(r2)+,r0
          move            #small,x0a,y0
          mpy    x0,y0,a
          move        x:(r0),a               a,y:(r4)+
          sub    x1,a                        y:(r0),b
          sub    y1,b     a,x0
          mpy    x0,x0,a  b,y0
          mac    y0,y0,a  x:(r2)+,r0
          move            #small,x0a,y0
          mpy    x0,y0,a
          move            x:(r0),a           a,y:(r4)+
          sub    x1,a                        y:(r0),b
          sub    y1,b     a,x0               y:(r4)+,y0
          mpy    x0,x0,a  b,y0
          mac    y0,y0,a  x:(r2)+,r0
          move            #small,x0a,y0
          mpy    x0,y0,a
          move            x:(r0),aa,         y:(r4)+
          sub    x1,a                        y:(r0),b
          sub    y1,b     a,x0               y:(r4)+,y0
          mpy    x0,x0,a  b,y0
          mac    y0,y0,a  x:(r2)+,r0
          move            #small,x0a,y0
          mpy    x0,y0,a                     y:(r4)+,b
          move            x:(r0),a           a,y:(r4)-
          sub    x1,a                        y:(r0),b
          sub    y1,b     a,x0
          mpy    x0,x0,a  b,y0
          mac    y0,y0,a  x:(r2)+,r0
          move            #small,x0a,y0
          mpy    x0,y0,a
          move            x:(r0),a           a,y:(r4)-
          sub    x1,a                        y:(r0),b
          sub    y1,b     a,x0
          mpy    x0,x0,a  b,y0
          mac    y0,y0,a  x:(r2)+,r0
```

Figure B-1 DSP56001 Decoding Program Listing (sheet 6 of 11)

```
            move        #small,x0 a,y0
            mpy     x0,y0,a
            move                  a,y:(r4)
            rts

;the accumulted distance routine  adds the smallest distance from the
;previously computed table for all pathes going into a state and
;does this for all eight states.

accumdist
            clr     a         #tables,r0
            move              #$7fffff,a1
            move              r0,r4
            move              #temp,r2
            move              #3,m0
            move              m0,m4
            move              #2,n1
            move              n1,n5
            move              r1,r5

;find minimum distance to state zero
            do      #4,statezero
            move              x:(r0),x0       y:(r4),b
            add     x0,b
            cmp     b,a
            tge     b,a       r0,r3
            tge     b,a       r4,r7
            move              x:(r0)+,x0      y:(r4)+,b
statezero
            move              r3,x:(r1)+n1
            move              a,x:(r2)+       y:(r4)+,b
            clr     a                         r7,y:(r5)+n5
            move              #$7fffff,a1

;find minimum distance to state two
            do      #4,statetwo
            move              x:(r0),x0       y:(r4),b
            add     x0,b
            cmp     b,a
            tge     b,a       r0,r3
            tge     b,a       r4,r7
            move              x:(r0)+,x0y:(r4)-,b
statetwo
            move              r3,x:(r1)+n1
            move              a,x:(r2)+       y:(r4)+,b
            clr     a                         r7,y:(r5)+n5
            move    #$7fffff,a1

;find minimum distance to state four
            do      #4,statefour
            move              x:(r0),x0       y:(r4),b
            add     x0,b
            cmp     b,a
            tge     b,a       r0,r3
            tge     b,a       r4,r7
            move              x:(r0)+,x0      y:(r4)+,b
```

Figure B-1 DSP56001 Decoding Program Listing (sheet 7 of 11)

```
statefour
            move            r3,x:(r1)+n1
            move            a,x:(r2)+       y:(r4)+,b
            clr     a                       r7,y:(r5)+n5
            move            #$7fffff,a1

;find minimum distance to state six
            do      #4,statezsix
            move            x:(r0),x0       y:(r4),b
            add     x0,b
            cmp     b,a
            tge     b,a     r0,r3
            tge     b,a     r4,r7
            move            x:(r0)+,x0      y:(r4)-,b
statezsix
            move            r3,x:(r1)-n1
            move            a,x:(r2)+
            move            r7,y:(r5)
            move            #tables+4,r4
            move            r4,r0
            move            x:(r1)-n1,a
            clr     a       x:(r1)-,b
            move            #$7fffff,a1
            move            r1,r5

;find minimum distance to state one
            do      #4,stateone
            move            x:(r0),x0y:(r4),b
            add     x0,b
            cmp     b,a
            tge     b,a     r0,r3
            tge     b,a     r4,r7
            move            x:(r0)+,x0      y:(r4)+,b
stateone
            move            r3,x:(r1)+n1
            move            a,x:(r2)+y:(r4)+,b
            clr     a       r7,y:(r5)+n5
            move            #$7fffff,a1

;find minimum distance to state three
            do      #4,statethree
            move            x:(r0),x0       y:(r4),b
            add     x0,b
            cmp     b,a
            tge     b,a     r0,r3
            tge     b,a     r4,r7
            move            x:(r0)+,x0      y:(r4)-,b
statethree
            move            r3,x:(r1)+n1
            move            a,x:(r2)+y:(r4)+,b
            clr     a       r7,y:(r5)+n5
            move            #$7fffff,a1
            move            (r4)+
```

Figure B-1 DSP56001 Decoding Program Listing (sheet 8 of 11)

```
;find minimum distance to state five
            do     #4,statefive
            move            x:(r0),x0y:(r4),b
            add    x0,b
            cmp    b,a
            tge    b,a      r0,r3
            tge    b,a      r4,r7
            move            x:(r0)+,x0      y:(r4)-,b
statefive
            move            r3,x:(r1)+n1
            move            a,x:(r2)+       y:(r4)-,b
            clr    a        r7,y:(r5)+n5
            move            #$7fffff,a1

;find minimum distance to state seven
            do     #4,stateseven
            move            x:(r0),x0       y:(r4),b
            add    x0,b
            cmp    b,a
            tge    b,a      r0,r3
            tge    b,a      r4,r7
            move            x:(r0)+,x0      y:(r4)+,b
stateseven
            move            r3,x:(r1)+
            move            a,x:(r2)+       y:(r4)+,b
            clr    b        r7,y:(r5)+
            move            #$7fffff,b1

;now move new accumulated distances into the  accumulated distance
;table from the temporary table
;also find the min distance state and store in r4 which is no longer used
            move            #$ffff,m0
            move            #$ffff,m4
            move            #temp,r3
            move            #tables,r0
            move            #large,x1
            move            #2,n0
            do     #4,endtable
            move            x:(r3)+,x0
            mpy    x1,x0,a
            cmp    a,b  a,  x:(r0)+n0
            tge    a,b      r0,r4
endtable
            move            #tables+1,r0
            do     #4,endtablex
            move            x:(r3)+,x0
            mpy    x1,x0,a
            cmp    a,b  a,  x:(r0)+n0
            tge    a,b      r0,r4
endtablex
;store in r0 instead of r4

            move            r4,r0
            move            #8,n1
            move            (r0)-n0
            rts
```

Figure B-1 DSP56001 Decoding Program Listing (sheet 9 of 11)

```
;the traceback routine now goes back through every time period starting
;with the current time period and finds the state from which the path
;came from one time period previous.  At the end of this search, the
;last state found will also point to the path at that state, which is the
;output  of the trellis.

traceback

;find the displacement from the pointer to table and store value in n4
            move            #tables,n0
            move            (r1)-n1
            lua    (r0)-n0,n5
            move            r1,r5
            do     #15,endtrace
            move            (r1)-n1
            move            x:(r5+n5),r0
            move            r1,r5
            lua    (r0)-n0,n5
endtrace
            move            #location,r0
            move            y:(r5+n5),a
            rts

;the output data routine unscrambles the path order and finds one
;of the four points on the constellation coresponding to the output state
;which is closest to the original input at that time period.

outputdata
            move            a,b
            move            #>$b1,x0
            cmp    x0,a     #>$b2,y0
            teq    y0,b
            cmp    y0,a     #>$b3,x0
            teq    x0,b
            cmp    x0,a     #>$b1,y0
            teq    y0,b
            move            #>$b5,x0
            cmp    x0,a     #>$b7,y0
            teq    y0,b
            cmp    y0,a
            teq    x0,b
            move            b,r2
            move            #tables,n2
            move            x:storr6,r6
            lua    (r2)-n2,n3
            move            n3,a
            asl    a
            asl    a
            move   a,       n0
            move            r6,r3
            lua    (r0)+n0,r4
            move            #>$7fffff,x1
            move            r4,r0
            do     #4,endout
            move            x:(r3),a        y:(r6),b
            move            x:(r0)+,x0      y:(r4)+,y0
```

Figure B-1 DSP56001 Decoding Program Listing (sheet 10 of 11)

```
                sub     x0,a
                sub     y0,b        a,x0
                mpy     x0,x0,a     b,y0
                mac     y0,y0,a
                tfr     a,b         x1,a
                cmp     x1,b
                tlt     b,a         r0,r7
                move    a,x1
endout
                clr     a           (r7)-
                move                #location,n0
                move                r7,r0
                move                #$f,a1
                lua     (r0)-n0,r7
                move                r7,x0
                and                 x0,a
                move                a1,y:$eff
                rts
```

Figure B-1 DSP56001 Decoding Program Listing (sheet 11of 11)

APPENDIX: C

DSP56001 Bound.D Data File

```
_DATA XE 200 82
 86 8b 8d 93 95 9a 9e 82
 86 89 8f 93 95 9a 9e 82
 86 89 8f 91 97 9a 9e 82
 86 8b 8d 91 97 9a 9e 82
 86 8b 8d 93 94 9a 9d 82
 86 89 8f 92 95 99 9e 82
 86 89 8f 90 97 99 9e 82
 86 8b 8d 91 96 9a 9d 83
 84 8b 8d 93 94 9a 9d 80
 87 89 8f 92 95 99 9e 80
 87 89 8f 90 97 99 9e 83
 84 8b 8d 91 96 9a 9d 82
 85 8a 8d 93 95 9a 9e 82
 85 88 8f 93 95 9a 9e 81
 86 89 8e 91 97 9a 9e 81
 86 8b 8c 91 97 9a 9e 82
 85 8a 8d 93 94 9a 9d 82
 85 88 8f 92 95 99 9e 81
 86 89 8e 90 97 99 9e 81
 86 8b 8c 91 96 9a 9d 82
 85 8a 8d 93 95 98 9f 82
 85 88 8f 93 95 98 9f 81
 86 89 8e 91 97 9b 9c 81
 86 8b 8c 91 97 9b 9c 83
 84 8a 8d 93 94 9a 9d 80
 87 88 8f 92 95 99 9e 80
 87 89 8e 90 97 99 9e 83
 84 8b 8c 91 96 9a 9d 82
 85 8a 8d 93 94 98 9f 82
 85 88 8f 92 95 98 9f 81
 86 89 8e 90 97 9b 9c 81
 86 8b 8c 91 96 9b 9c 83
 85 8a 8d 93 94 98 9d 80
 85 88 8f 92 95 99 9f 81
 87 89 8e 90 97 99 9c 81
 84 8b 8c 91 96 9b 9d 82
 85 8a 8d 93 94 98 9d 82
 85 88 8f 92 95 99 9f 81
 86 89 8e 90 97 99 9c 81
 86 8b 8c 91 96 9b 9d 83
 85 8a 8d 93 94 9a 9d 80
 85 88 8f 92 95 99 9e 81
 87 89 8e 90 97 99 9e 81
 84 8b 8c 91 96 9a 9d 83
```

Figure C-1 DSP56001 Bound.D Data File

REFERENCES

1. Lin, S. And D. Costello, Error Control Coding: fundamentals and Applications, Englewood Cliff, NJ: Prentice Hall, 1983.
2. Sklar, B., Digital Communications Fundamentals and Applications, Englewood Cliffs, NJ: Prentice Hall, 1988.
3. Ungerboeck, G., "Trellis Coded Modulation with Redundant Signal Sets Part 1: Introduction," IEEE Communications Magazine, vol. 25, no. 2, February 1987, pp. 5-11.
4. A.J. Viterbi, "Error Bounds for Convolutional Codes and An Asymptotically Optimum Decoding Algorithm," IEEE Trans. Inf. Theory, vol. IT-13, April 1967, pp. 260-269.
5. DSP56001/DSP56001 Digital Signal Processor User's Manual, (DSP56000UM/AD), Motorola Inc., 1989.
6. Wei, L.F., "Rotationally Invariant Convolutional Channel coding with Expanded Signal Space — Part 1:180," EEE Journal on Selected Areas in Communications, vol. SAC-2, no. 5, September 1984, p.661.
7. The International Telegraph and Telephone Consultative Committee, CCITT Red Book, "Data Communication over the Telephone Network," vol. 8, 1985, pp. 221-238.
8. Fagen, A., et al., "Single DSP Implementation of a High Speed Echo Cancelling Modem Employing Trellis Coding," Proc. of the Intnl. ESA Workshop on DSP Techniques Applied to Space Communications, November 1988.

9. Heller, J.A. and I.W. Jacobs, “Viterbi Decoding for Satellite and Space Communication,” IEEE Trans. Commun. Tech., vol. COM-19, no. 5, October 1971, pp. 835-848.

10. Magotra, N., et al., “A Comparison of Two Parametric Estimation Schemes,” Proc. IEEE, vol. 74, no. 5, May 1986, pp. 760-761. ■