A Simple Implementation of the Viterbi Algorithm on the Motorola DSP56001

Dion D. Messer and Sangil Park

Motorola Inc.
Digital Signal Processing Operations
Austin, TX 78735

ABSTRACT

As system designers design communication systems with digital instead of analog components to reduce noise and increase channel capacity, they must have the ability to perform traditional communication algorithms digitally. The use of Trellis Coded Modulation as well as the extensive use of convolutional encoding for error detection and correction requires an efficient digital implementation of the Viterbi Algorithm for real time demodulation and decoding. Digital Signal Processors are now fast enough to implement Viterbi decoding in conjunction with the normal receiver/ transmitter functions for lower speed channels on a single chip as well as performing fast decoding for higher speed channels, if the algorithm is implemented efficiently. The purpose of this paper is to identify a good way to implement the Viterbi Algorithm (VA) on the Motorola DSP56001, balancing performance considerations with speed and memory efficiency.

1.0 Introduction

The DSP56001 is a digital signal processing chip which is well suited for communication applications and is also adaptable for Viterbi decoding because of it's dual data memory structure, zero overhead modulo addressing, and hardware do loop capability. The key obstacles to implementing the VA on the 56001 are: overflow in the "accumulated distance to each state" calculation, and finite memory availability for path storage. This paper discusses novel solutions to these obstacles in implementing the VA on the 56001 as well as an evaluation of the performance of the decoder using this implementation. An example code is used to help explain the concepts in this paper. The example code is the V.32 trellis, which is shown in Figure 1. Figure 2 shows the corresponding constellation and Figure 3 is a block diagram of the encoder.

This 8 state trellis is used as an example because the short constraint length is a less complicated structure to use for explanation than the K=7 codes which are popular on satellite channels. However, the performance of K=7 codes will be discussed in section 3.

2.0 Background

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 4 or 5 times the constraint length [9].

The most likely path through the trellis is determined to be that one which 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 [1]:

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

where r_i and v_i are the received and the decoded signal sequence respectively.

Looking at Figure 3, there are 3 delays, (Sl, S2, and S3) and the data they contain at any given time period is called the delay state in this discussion. The output (YOn, Yln, and Y2n) are referred to as the path state because they refer to the state of the path.

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 delay state with the smallest accumulated distance is the beginning point, at that time period, to trace the minimum distance path through the past N-l time periods of the trellis. The minimum distance paths to the next delay state are then determined by evaluating the input to determine which point on the constellation in each path it is closest to, determining the Euclidean distance to each of those points, then, based on the trellis structure, and the minimum distance paths, determine the minimum accumulated distance to each delay state. So, after defining the trellis, the steps taken to decode the data are given below [1].

1) At each input compute the minimum distance path states 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 4 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 4 assuming $C+\gamma$ 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. By storing the minimum distance path state (YOn, Yln, Y2n) to each delay state, as well as the delay state (Sl, S2, S3) the path originated from, the most likely path can be traced. This is done by starting at the minimum accumulated distance state, going to the state it came from, and repeating this process N-l 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 4, if at t_0, the end point of the minimum distance path turned out to be 010 then the output of the Viterbi decoder would be 001.

In summary, at every time period, the accumulated distance to each delay state is calculated and updated and the minimum distance path state (YOn, Yln, Y2n) to each delay state is stored, as well as the delay state it came from (Sl, S2, S3). This creates a history so that it is possible to trace back in time to get the correct output of the decoder.

A block diagram of the V.32 decoder showing inputs and outputs is shown in Figure 5. It can be compared to the block diagram of the encoder shown in Figure 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 done first and then the differential decoding is done.

3.0 Performance Parameters

The three basic parameters which affect the performance of the Viterbi algorithm are discussed in this section.

3.1 The Accumulated Distance Calculation

At every input, the accumulated distance to each state must be recomputed by adding previous accumulated distances to current path distances. Since the DSP56001 is a fixed point processor, this cannot occur continuously without resulting in an overflow problem. Thus, an alternate way to obtain the accumulated distance measurement is a weighted accumulation method which can be expressed as [10]:

$$d_{new} = \beta d_{old} + (1 - \beta) d_{path} \qquad (2)$$

where $O << \beta < 1$ denotes the smoothing parameter. This method (essentially a low pass filter) ensures that the new accumulated distance is a bounded arithmetic value. It has also been shown that this method gives unbiased estimates [10]. Although (2) 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:

$$\tau \approx \frac{2}{1-\beta} \qquad (3)$$

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. In testing this implementation, values of β which fall in the range,$.9 < \beta < .99$ provided very good results in that there was no change in bit error rate (BER) with blocks of 10^4 data bits. Comprehensive tests using larger blocks of data are planned to chart the BER as β is varied over the same range. It is expected that this will produce an optimum value of β for different constraint length codes.

3.2 Path Memory Length

As stated previously, the number of time periods for near-optimum decoding is 4 or 5 times the constraint length K. The objective is to determine a path memory length which gives an optimum BER, decodes at an acceptable speed, and which conserves memory. Because of the looping capability of the DSP56001 and the modulo addressing scheme, each time period of path memory only requires 4 instruction cycles to trace. Therefore, in the case of V.32, time periods of 16 and 20 only take 60 and 80 instruction cycles respectively to determine the output when tracing through the trellis. The difference of 20 instruction cycles has a minimal affect on the total instruction cycle count needed for the decoding process. Since each time period requires only 4 instruction cycles, the extra processing time is not an issue in determining the path length.

Each additional time period does require extra memory locations for each state in the trellis. When the constraint length is short, the number of states are fewer and fewer extra memory locations are needed. As the constraint length increases to K=7, there are 2^{K-1} or 64 states. this means for each extra time period there needs to be 64 extra memory locations. In the K=7 case, the path length at 4 times K is 28 time periods and at 5 times K it is 35 time periods. For the difference of 7 time periods there would have to be 448 additional memory locations. When memory is scarce, this could be the decision factor in determining the path length used with this implementation.

Testing path memory lengths of 4 and 5 times the constraint length revealed no difference in BER performance. Again, blocks of 10^4 data bits were used for this testing. Additional testing of larger blocks of data is expected to reveal that the longer the path memory the better the performance of the decoder.

3.3 Maximum Data Rate

The Motorola DSP56001 currently operates at 27 MHz with an instruction cycle of 75 ns. 700 instruction cycles are all that is needed to decode on input symbol (4 output bits) for the V.32 case. This is only 15% of the processor capability, allowing time to perform the modem transmit and receive functions if desired. If the processor is used only as a stand alone decoder for this code, a data rate of 76Kbs can be achieved using 100% of the processor!

In the case of the constraint length K=7 codes often used for satellite channels, there are 64 delay states which pushes the processing for each input to 1300 instruction cycles. Using 100% of the processor, a data rate of 10Kbs can be achieved with present processor speeds. Ad-

vances in VLSI technology will push the processor speeds faster in the near future, allowing an even higher data rate for decoding on the DSP56001.

4.0 Summary

As shown, the Motorola DSP56001 offers a flexible solution to the Viterbi decoding task in a communications channel. Since it is programmable, it is possible to decode any number of different codes with varying constraint lengths by boot loading the software for the desired code. Using the DSP56001 can help in making designs compact by eliminating special purpose chips for decoding, echo cancellation, PLL, timing recovery, equalization, modulation and demodulation. All of these tasks can be performed on the DSP56001 for low data rate channels. As DSP's are used more and more in communication systems, Viterbi decoding as a software solution will become a necessity for efficient system designs.

The software for the example given is available on the Motorola DSP bulletin board (512-891-3771), or by contacting the authors at the address given.

REFERENCES

[1] S. Lin and D. Costello, Error Control Coding: Fundamentals and Applications, Prentice Hall, 1983.

[2] B. Sklar, Digital Communications Fundamentals and Applications, Prentice Hall, 1988, p. 319.

[3] G. Ungerboeck, "Trellis Coded Modulation with Redundant Signal Sets Part I:Introduction," IEEE Communications Magazine 25(2) (February 1987).

[4] A. J. Viterbi, "Error Bounds for Convolutional Codes and An Asymptotically Optimum Decoding Algorithm," IEEE Trans. Inf. Theory, vol IT13, April 1967, pp. 260-269.

[5] DSP56000 Digital Signal Processor User's Manual, Motorola Inc., 1989.

[6] L. -f. Wei, "Rotationally Invariant Convolutional Channel Coding with Expanded Signal Space - Part 1:180," IEEE Journal on Selected Areas in Communications SAC-2(5) p. 661 (September 1984)

[7] CCITT, The International Telegraph and Telephone Consultative Committee, Red Book. Volume 8. 1985. p.222.

[8] A. Fagen, 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, Noordwijk, November 1988.

[9] J.A. Heller, and I.W. Jacobs, "Viterbi Decoding for Satellite and Space Communication," IEEE Trans. Commun. Technol., vol. COM19, no 5, October 1971, pp. 835-848.

[10] N. Magotra, et Al., "A Comparison of Two Parametric Estimation Schemes," Proc. IEEE, vol. 74, No. 5, pp.760-761, May 1986.

[11] E.A. Lee and D.G. Messershmitt, Digital Communication, Kluwer Academic Publishers, 1988.

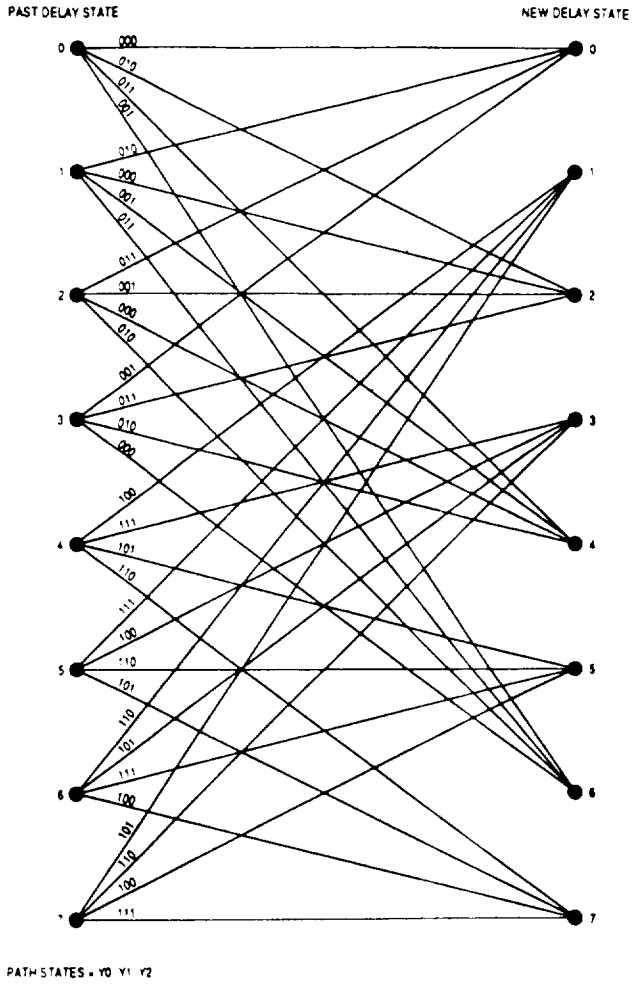

Figure 1. Trellis Diagram

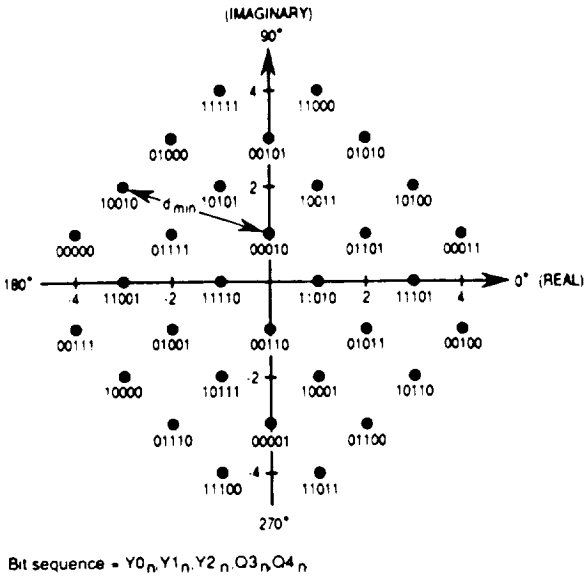

Figure 2. V.32 Constellation

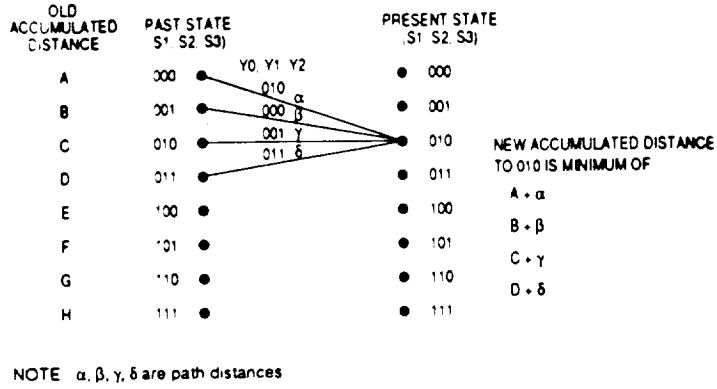

Figure 4. Possible Paths to State 010

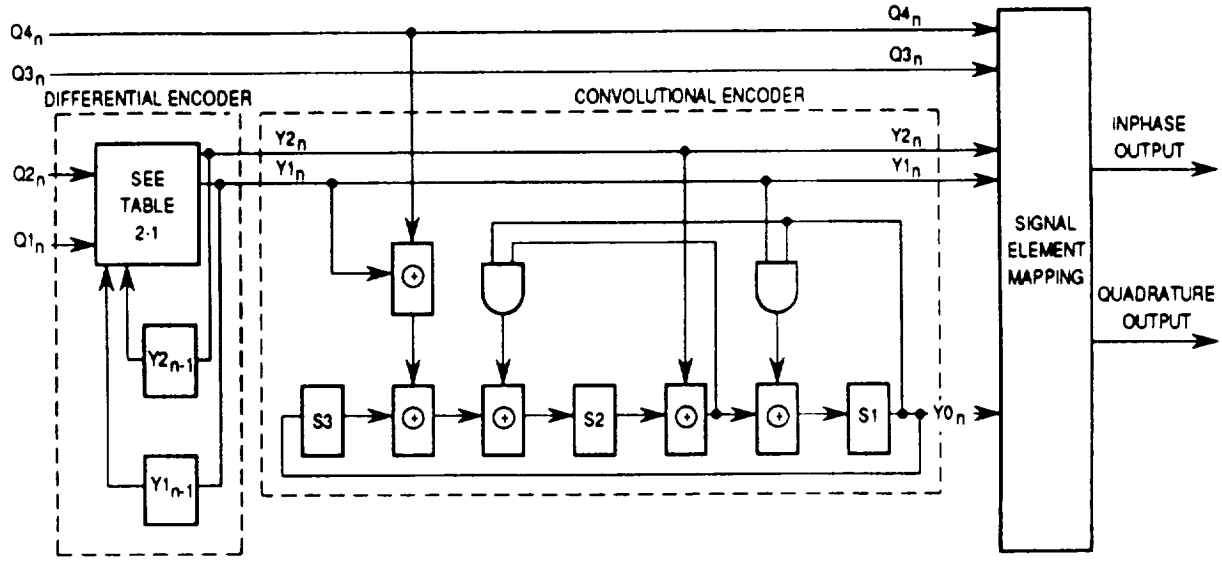

Figure 3. V.32 Encoding Diagram

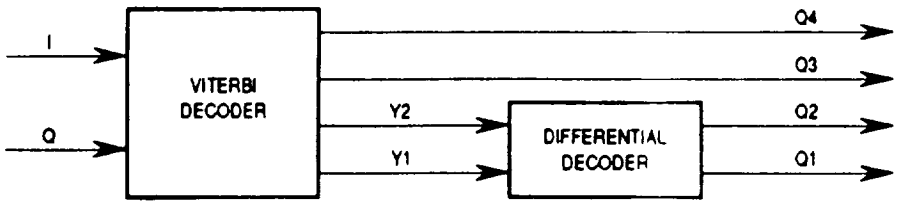

Figure 5. V.32 Decoder Block Diagram