S9-60

264314

18.

TDA Progress Report 42-99

November 15, 1989

Memory Management in Traceback Viterbi Decoders

O. Collins
Johns Hopkins University, Maryland

F. Pollara
Communications Systems Research Section

The new Viterbi decoder for long constraint length codes, under development for the DSN, stores path information according to an algorithm called "traceback." The details of a particular implementation of this algorithm, based on three memory buffers, are described. The penalties in increased storage requirement and longer decoding delay are offset by the reduced amount of data that needs to be exchanged between processors, in a parallel architecture decoder.

I. Introduction

A new, long constraint length Viterbi decoder [1] is under development for the DSN, and will be used to decode the constraint length $K = 15$ experimental code adopted by the Galileo mission. This article describes the traceback algorithm that is used in this decoder to store the most likely paths into each node of the decoder's trellis.

The *traceback* (TB) method is one of two known ways to store decisions made by the add-compare-select unit of a Viterbi decoder, and then provide decoded bits as output. The other method is the more traditional *register exchange* (RE) technique. The RE method is suitable for short constraint length decoders or for low-speed decoders due to the large amount of data that needs to be read, modified, and rewritten at each bit time.

The basic difference between these two methods is that the RE method stores the actual hypothesized information sequences (survivors), while the TB method stores the results of comparisons of paths converging into each node of the trellis. The maximum-likelihood path is then found by tracing back through the trellis a path, according to stored decisions. It is worth observing that the bits representing the results of these comparisons actually coincide with the information bits in convolutional codes where the state transitions are governed by a shift register. Therefore, the crucial difference between RE and TB methods lies in the organization of the memory used to store the survivors. The TB method is widely used in practice, but not as widely described in the literature. It is mentioned in [2], and described in [3] without any reference to required storage or resulting decoding delay.

This article deals with the details of one version of the traceback algorithm that is based on three pointers exploring three memory buffers. Details on the hardware implementation of this version of the algorithm may be found in [4]. If L branches are required as the minimum decoding depth ($L \approx 5K$ is a usually accepted rule, but $L \approx 10K$ is more realistic for low E_b/N_0 applications), enough storage must be provided for $3L$ branches in order to perform the necessary buffering for this TB method. This penalty is not important in practice since inexpensive and slow off-chip memory can be used, while the RE method requires fast on-chip registers.

II. Memory Management for Traceback

The memory required for the traceback method is organized in three banks as shown in Fig. 1. Each bank is L bits long and 2^{K-1} bits high (the number of states), which gives a required storage of about 1 Mbyte, for $K = 15$ and $L = 170$ (not including additional storage for 2^{K-1} accumulated metrics). Any bit in this traceback memory can be accessed by an address consisting of the state j ($0 \leq j < 2^{K-1}$) and the bit memory pointer m ($0 \leq m < 3L$).

There are three basic operations going on in the memory banks every bit time:

(1) *Traceback*, which is a "read" operation and traces a path between states on the trellis by computing the next (backward) state address from the presently read memory content.

(2) *Decoding*, which is also a "read" operation and similar to traceback, except that it is performed on "older" data and it produces output information bits corresponding to the path being traced.

(3) *Writing* new data (decisions given by the add-compare-select unit), which moves forward on the trellis. These bits can be written in the locations just freed by the decoding operation.

Every L bit times a new traceback front is started in one memory bank from state 0, and a new decode front is started in a different memory bank at the state where the previous traceback ended. New data is written into the second memory bank as memory locations are freed by the decode operation. The third memory bank remains inactive. After a period of L bit times, the traceback/decode/write operations are switched to a different pair of memory banks, as described in detail below.

Among the three operations, the writing of new data is by far the most time consuming, since 2^{K-1} bits must be written for each information bit time. Read operations only access one bit per information bit time.

Given the amount of required memory, it is cheaper to use commercial RAM chips, rather than to design this memory into custom VLSI circuits.

III. The Traceback Algorithm in Detail

The evolution of memory operations needed for the traceback method is illustrated in Fig. 2, where the vertical axis represents the elapsed time and each box represents the status of memory at a given time. The variables' names are the same later used in the pseudocode description of the algorithm in Fig. 4.

At *time* $= 0$, a new traceback is started from state 0 (*state_tb* $= 0$) at bit memory pointer $m = L - 1$ in the rightmost memory bank (bank $= 0$). The top row of Fig. 2 shows the memory bank number where traceback (*tb*) and decoding (*dec*) operate. This traceback will end in a certain *state_tb* at bit memory pointer $m = 0$.

Simultaneously, a decoding operation starts from state *state_dec* (this is initially an arbitrary value) at bit memory pointer $m = L-1$ and proceeds until $m = 0$, in the leftmost memory bank (bank 1). For each decoded bit, a full column of bits can be overwritten with new data. Notice that all three operations (traceback, decode, and write) evolve from right to left.

At the end of the first block of L bits, a new traceback starts from state 0 (*state_tb* $= 0$) at bit memory pointer $m = 0$, moving from left to right in memory bank 1. At the same time, a decoding operation goes on in the middle bank (bank 2), starting at the state where the previous traceback ended (*state_dec* $=$ *state_tb*) and at bit memory pointer $m = 0$. Also, new data is written in the bank doing decoding. All of these operations evolve from left to right.

After the end of the second block, a traceback and a decoding start at $m = L - 1$, both moving from right to left, in banks 2 and 0, respectively. New data is written in bank 0. After the third block, i.e., during the fourth block, traceback and decoding take place in banks 0 and 1 again, but all operations occur left-to-right, which is opposite to the direction for the first block. Notice that the read/write operations alternate between right-to-left and left-to-right sweeps, which implies that Fig. 2 has a repetition cycle of $3 \times 2 = 6$ blocks.

Only after three full blocks is decoding performed on data that has actually been written, rather than on initialized memory. Therefore the decoding delay is at least $3L$ bits. Since the decoded output is generated in reversed order (last bit first in each block), one must provide a buffer to reverse the output, bringing the decoding delay to $4L$ bits. Finally, the delay due to the encoder memory must be added, which yields a total decoding delay of $4L + K$ bits.

The flow diagram of memory operations is shown in Fig. 3, where the traceback and decoding operations are shown as sequential in time, even though they may happen simultaneously in a specialized hardware.

Figure 4 shows the pseudocode description of the TB algorithm for a convolutionally coded system using a (7,1/2) code and $L = 100$. The C-language version of this algorithm has been used to demonstrate this concept by software simulation and to verify the correctness of the algorithm. The memory is denoted by the three-dimensional array RAM[*state*][*m*][*bank*], M[*state*][.] stores the accumulated metrics, and *d*[.] represents the branch metrics.

IV. Advantages of Traceback for Parallel Processing

In a multiprocessor implementation, the Viterbi algorithm based on register exchange requires that the full survivor sequences be exchanged among processors, together with the accumulated metrics. It has been found [5] that the traceback method drastically reduces the communication bandwidth required between processors by eliminating the need for survivor exchanges. This reduction is achieved at the price of higher decoding delay.

Figure 5 shows a general parallel architecture, where the interconnection network is described in [6]. Since, among the three operations described in Section II, the write operation is the most demanding, it is performed concurrently in each processor and its local memory. Each processor operates sequentially on a certain number of states, and then exchanges the accumulated metrics through the interconnection network. The traceback/decoding operation may use a bus line to transfer information, consisting of traceback memory addresses sent to all memories and single bits coming from a particular memory, corresponding to the memory location referenced by a given address. The new address is computed from the old one and the data bit read from a memory. The address computation does not require parallelism, since it uses only one bit read per information bit. At the end of each block the last address found by the traceback unit is used to initialize the decoding unit. Further details on the hardware implementation may be found in [4].

References

[1] J. Statman, G. Zimmerman, F. Pollara, and O. Collins, "A Long Constraint Length VLSI Viterbi Decoder for the DSN," *TDA Progress Report 42-95*, Jet Propulsion Laboratory, Pasadena, California, pp. 134–142, November 15, 1988.

[2] G. C. Clark and J. B. Cain, *Error Correction Coding for Digital Communications*, Plenum Press, 1981.

[3] C. M. Rader, "Memory Management in a Viterbi Decoder," *IEEE Trans. on Communications*, vol. COM-29, no. 9, pp. 1399–1401, September 1981.

[4] O. Collins, *Coding Beyond the Computational Cutoff Rate*, Ph.D. Thesis, California Institute of Technology, May 1989.

[5] F. Pollara, "Concurrent Viterbi Algorithm with Traceback," *SPIE Proceedings*, vol. 696, p. 204–209, August 1986.

[6] O. Collins, F. Pollara, S. Dolinar, and J. Statman, "Wiring Viterbi Decoders (Splitting deBruijn Graphs)," *TDA Progress Report 42-96*, Jet Propulsion Laboratory, Pasadena, California, pp. 93–103, February 15, 1989.

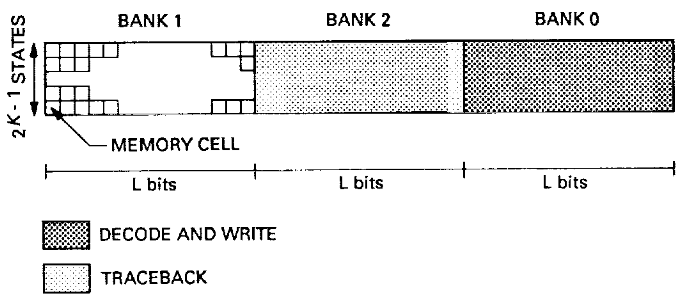

Fig. 1. Memory organization in three banks.

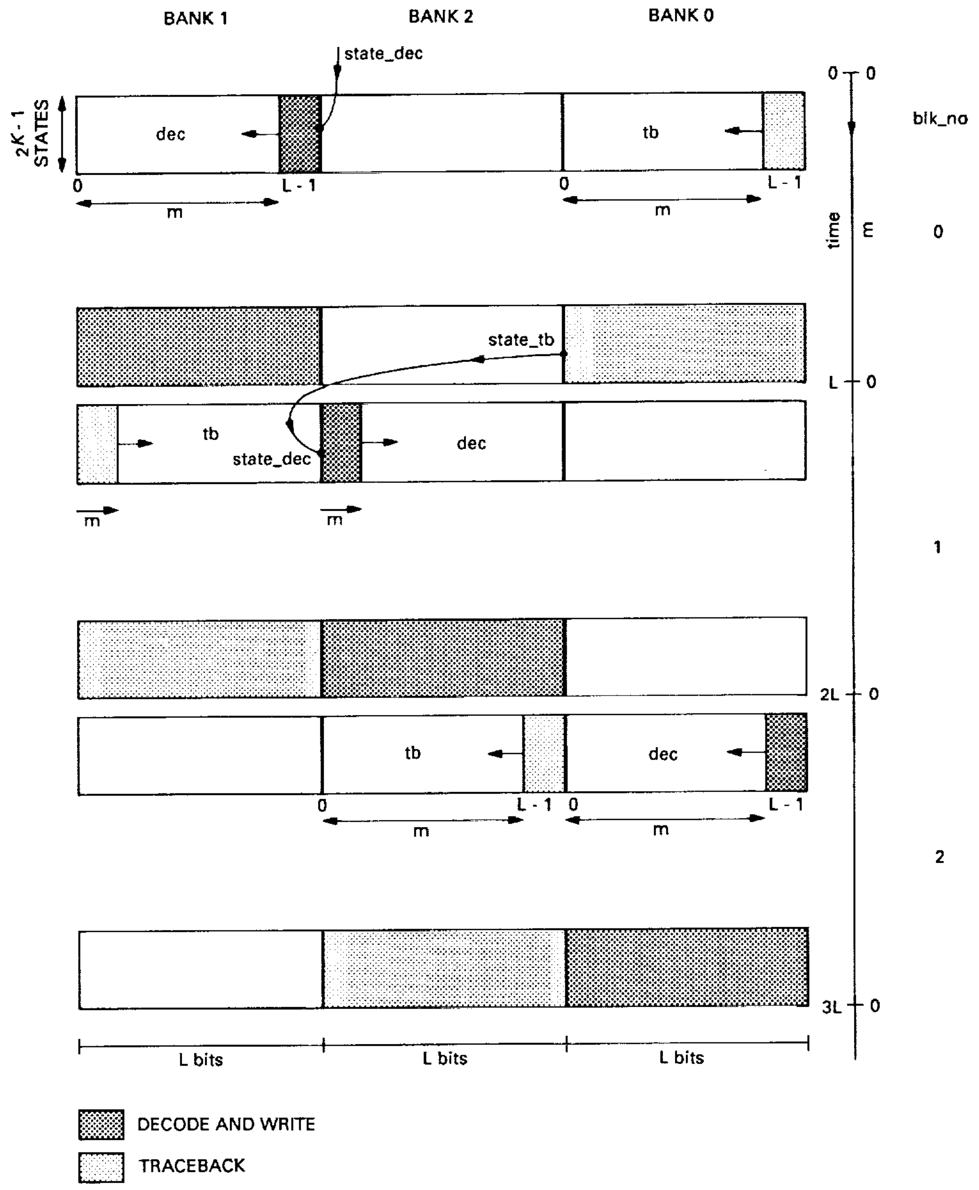

Fig. 2. Evolution of memory operations in the traceback method.

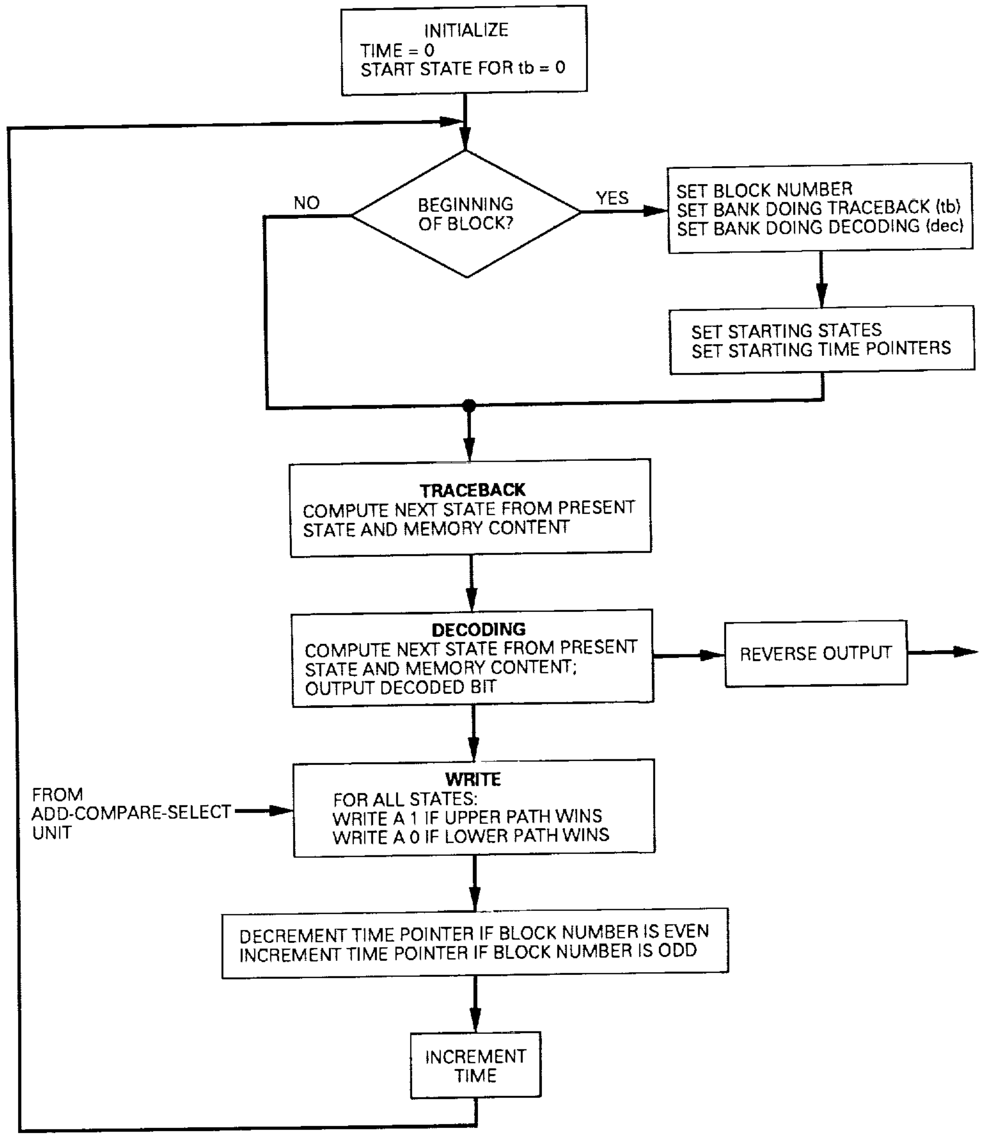
INITIALIZE
TIME = 0
START STATE FOR tb = 0
BEGINNING OF BLOCK?
NO
YES
SET BLOCK NUMBER
SET BANK DOING TRACEBACK (tb)
SET BANK DOING DECODING (dec)
SET STARTING STATES
SET STARTING TIME POINTERS
TRACEBACK
COMPUTE NEXT STATE FROM PRESENT STATE AND MEMORY CONTENT
DECODING
COMPUTE NEXT STATE FROM PRESENT STATE AND MEMORY CONTENT; OUTPUT DECODED BIT
REVERSE OUTPUT
FROM ADD-COMPARE-SELECT UNIT
WRITE
FOR ALL STATES:
WRITE A 1 IF UPPER PATH WINS
WRITE A 0 IF LOWER PATH WINS
DECREMENT TIME POINTER IF BLOCK NUMBER IS EVEN
INCREMENT TIME POINTER IF BLOCK NUMBER IS ODD
INCREMENT TIME

Fig. 3. Flow diagram of memory operations.

ORIGINAL PAGE IS
OF POOR QUALITY

```
K=7                                             "constraint length"
NS=2^(K-1)                                      "number of states"
NS2=NS/2
L=100                                           "truncation length"
state_tb=0
time=0

WHILE time < max
{

m=time MOD L
bit_par=time MOD 2

IF(m==0)                                        "start a new traceback front"
{
blk_no=time/L
blk_par=blk_no MOD 2
tb=blk_no MOD 3                                 "bank doing traceback = 0,1,2"
dec=(tb+1) MOD 3                                "bank doing decoding"
state_dec=state_tb                              "set starting state for decoding"
state_tb=0                                      "set starting state for new traceback"
FOR m FROM 0 TO L-1 {outr[m]=out[m]}            "buffer for order reversal"
}

IF(blk_par==0) {m1=L-m-1}                       "right to left"
ELSE           {m1=m}                           "left to right"

state_dec=(SHIFT RIGHT state_dec OF 1 BIT) OR (NS2*RAM[state_dec][t][dec])
                                                "decoding: address in bank=dec"

state_tb= (SHIFT RIGHT state_tb OF 1 BIT ) OR (NS2*RAM[state_tb ][t][tb ])
                                                 "traceback: address in bank=tb"

out[m]= (SHIFT RIGHT state_dec OF K-2 BITS) AND 1        "output buffer"

FOR j FROM 0 TO NS-1                                     "add, compare, select"
        {
        i=SHIFT RIGHT j OF 1 BIT
        L0 = M[i]      [bit_par XOR 1] + d[K[j]]
        L1 = M[i OR NS2][bit_par XOR 1] + d[3-K[j]]
        IF(L1 < L0)      {                               "write one bit"
                         M[j][bit_par]=L1
                         RAM[j][m1][dec]=1
                         }
        ELSE             {
                         M[j][bit_par]=L0
                         RAM[j][m1][dec]=0
                         }
        }

PRINT outr[L-1-m]                                "print decoded bits in correct order"

time = time+1
}
```

Fig. 4. Pseudocode for traceback algorithm.

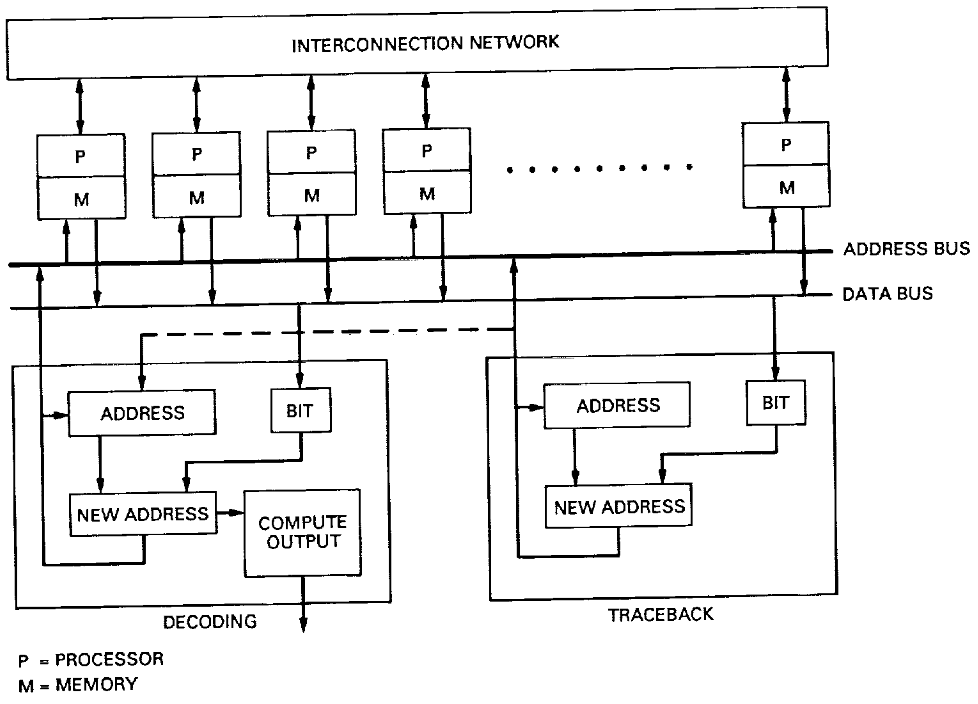

Fig. 5. Parallel traceback architecture.