PRACTICAL IMPLEMENTATION OF A FLEXIBLE VITERBI DECODER

Jean L. Bérubé

Canadian Marconi Company
2442 Trenton, P.O. Box 30B
Montréal, Québec, Canada H3P 1Y9

Jean Conan

Department of Electrical Engineering
Ecole Polytechnique of Montreal
P.O. Box 6079, Station "A"
Montréal, Québec, Canada H3C 3A7

RESUME

Dans cet article, nous présentons les éléments d'implantation d'un décodeur de Viterbi câblé, capable de fonctionner pour des taux de codage et des longueurs de contrainte variables. Une des caractéristiques essentielles de ce décodeur résulte de l'utilisation du parallélisme lors de l'extension des sous-treillis du code. Les conséquences en sont une plus grande souplesse au niveau de l'implantation et une vitesse considérablement accrue par rapport à un décodeur opérant de façon séquentielle. Une autre caractéristique de cette machine tient au fait qu'il permet de décoder des codes perforés de taux de codage élevé. D'autre part, le couplage avec un système d'acquisition de données en temps réel utilisant un ordinateur IBM-PC permet, dans un temps très court, de réaliser des simulations significatives très longues dont les statistiques sont calculables instantanément.

SUMMARY

This paper presents the design of a Viterbi decoder with selectable constraint length and encoding rate. The main features of this decoder are its use of parallelism in performing the extensions on the code sub-trellisses, which enables higher throughput and easier configuration of the decoder than is possible with recursive extension of each sub-treillis. Furthermore, another feature of this decoder is its ability to decode high rate punctured convolutional codes. Added flexibility is obtained through the use of an IBM-PC based control program which allows for easy changes of parameters as well as mass storage recordings of decoded data suitable for off-line statistical analysis.

I. INTRODUCTION

With the advent of digital communications by satellite, the use of convolutional encoding with the Viterbi decoding is becoming more popular because of the extremely good error correcting capabilities of this technique. However, computer simulations of the Viterbi algorithm require a large amount of computer time and it was felt possible that a fast, yet flexible, decoder could be built for extensive simulation purposes. This decoder uses the property that the trellis of a code can be decomposed into sub-trellisses [1,2] and that it is possible to apply the basic Add-Compare-Select (ACS) operations on each sub-treillis simultaneously. The parallel use of commercial dedicated MSI circuits which perform the basic ACS operations results in greater decoding speed than possible with a multi- microprocessor system [2], while still retaining sufficient flexibility in the choice of the coding rate and the code parameters used.

II. CONVOLUTIONAL CODES AND VITERBI DECODING

A binary convolutional code of rate R = 1/v and constraint length K is defined as the output range of some 1-input/v-outputs feedforward Linear Modular Circuit over GF(2) as illustrated in Figure 1. Assuming the K-stages shift register initially loaded with 0's, the codewords are computed by feeding an input binary symbol u_t into the shift register at each time unit t and computing a corresponding output binary v-tuple

$$\underline{x}_t \triangleq (x_t^1, x^2{}_t, \ldots, x^v{}_t),$$

called an output branch, according to the input u_t, the current content

$$\underline{s}_t \triangleq (u_{t-1}, \ldots, u_{t-K+1}) \text{ of}$$

the shift register, referred to as the state at time t, and the particular connections to the v output modulo 2 adders which characterize the encoder.

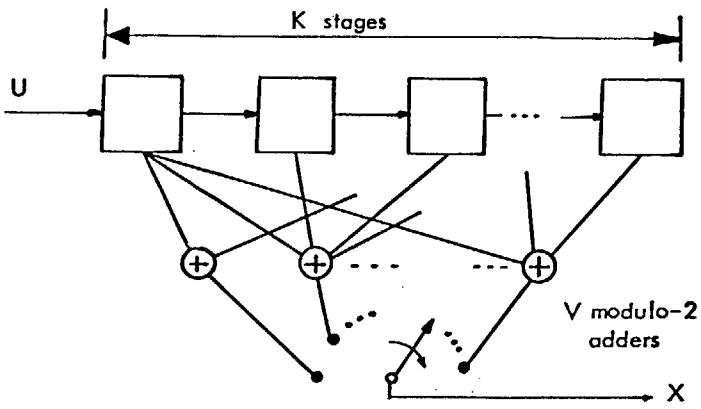

Fig. 1. General rate 1/v convolutional encoder of constraint length K.

Formally, letting $m \triangleq K-1$ be the memory order of the encoder, and defining the K v-tuples g^l $l = 0, 1, \ldots, m$ of 0's and 1's as specifying the connection array of the encoder through the relations

$$g_i^l = \begin{cases} 1 & \text{if the } l^{th} \text{ cell of the shift register is connected to the } i^{th} \text{ modulo 2 adder, } i = 1, \ldots, v. \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

and the related m x v matrix

$$G^* = \begin{bmatrix} g^1 \\ g^2 \\ . \\ . \\ g^m \end{bmatrix}, \quad (2)$$

the input/output map associated with the bit by bit encoding process can be represented through the map

$$E: U \times \underline{S} \rightarrow \underline{X}: (u_t, \underline{s}_t) \rightarrow \underline{x}_t = u_t g^0 + \underline{s}_t G^*, \quad (3)$$

where U, $\underline{S}$, $\underline{X}$ are respectively the input, state and output spaces. Furthermore, the dynamical behaviour of the encoder can be described by means of the next state transition map

$$Q: U \times \underline{S} \rightarrow \underline{S}: (u_t, \underline{s}_t) \rightarrow \underline{s}_{t+1} \triangleq (u_t, u_{t-1}, \ldots, u_{t-m+1}) \quad (4)$$

The graphical representation of the map Q together with the input/output E described by (3) is known as the Good-De-Bruijn graph $G_{q,m}$ of the encoder and is contructed as follows. To each element $\underline{s} \in \underline{S}$ is associated a vertex in $G_{q,m}$. Each vertex is connected to two successors as computed by Q and the arcs corresponding to the transitions $\underline{s}_1 \rightarrow \underline{s}_2$ are labeled by the pair $(u, \underline{x})$, where u is the input causing the transition and $\underline{x}$ is the corresponding output branch. For illustrative purposes, such a graph is represented in Figure 2 for the rate 1/2, m=2 code with connection array

$$g^0 = (1,1),\ g^1 = (1,0),\ g^2 = (1,1).$$

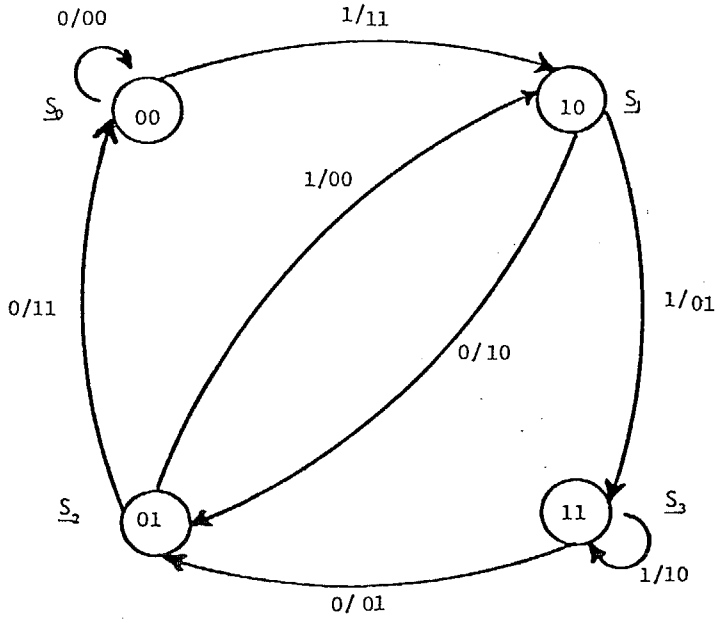

Fig. 2. Good-De-Bruijn transition graph for K=3.

For the purpose of decoding, a maximum likelihood sequence decoder must compare all possible encoded sequences with the noisy received sequence and choose the one with maximum a posteriori probability. The trellis diagram introduced by Forney [3] is a highly structured representation of all the input/output sequences of the code which constitutes the key to the Viterbi decoding procedure. Such a trellis can be constructed as follows. At time t = 0, the encoder is assumed to be in the null state. Ordering all the elements in the state space $\underline{S}$ in a fixed and predefined manner, we link all the possible state values at time t to all the possible state values at time t+1 by the corresponding connecting arcs in $G_{q,m}$ for t = 0,1,2,.... The trellis can then be terminated at some time L+m by appending m trailing 0's to the L information bits for resynchronisation purposes (sequence decoder), or unterminated by letting L go to infinity (bit decoder).

If the input sequence is U, the operation of a convolutional encoder can be described by a unique path in the trellis diagram resulting in the encoded output sequence X^U. Let Y be the received sequence after transmission of X^U over a Discrete Memoryless Channel (i.e., a transmission channel in which errors occur independently from one transmitted symbol to the other). We can characterize uniquely such a channel by the probability $P\{y|x\}$ that the symbol y will be received, given that x has been transmitted. For a DMC, we can write the "metric" (minus the log likelihood function) of the path U extending over n branches as

$$\Gamma_n^U \triangleq -LOG[P(Y|U)] = -\log[P(Y|X^U)] = \sum_{i=0}^{n-1} \gamma_I^U, \quad (5)$$

where the so called branch "metric" γ_i^U is defined as

$$\gamma_i^U = -\log[P(\underline{y}_i|\underline{x}_i^U)] = -\sum_{j=1}^{v} \log[P(y_i^j|x_i^{j,U})] \quad (6)$$

and y_i^j and $x_i^{j,U}$ are respectively the j^{th} symbol on the i^{th} received branch of Y and the j^{th} symbol on the i^{th} coded branch of the hypothesized path U.

Under these conditions, the decoding process which minimizes the probability of a sequence error P_E for a terminated sequence of L information bits is then tantamount to finding the path U in the trellis diagram whose sequence of branches X^U is "closest" to the received sequence Y in the sense that the corresponding metric Γ_{L+m}^U is minimized.

The Maximum Likelihood Viterbi Decoding (MLVD) algorithm is then equivalent [4] to the forward dynamic program which solves on the trellis diagram the foregoing optimization problem and satisfies the following terminal constraints on the state.

$$\underline{s}|_{t=0} = \underline{s}|_{t=L+m} = \text{null state.}$$

Such an algorithm is most easily described by the following recursive procedure.

MLVD Algorithm

Step 1: Extend in the trellis all the distinct 2^m paths diverging from the null state over m branches and compute their metrics. Call these paths the survivors.

$l \leftarrow m$.

Step 2: (basic Add-Compare - Select (ACS) operation)

Extend all the 2^m paths surviving at depth l and compute their new metrics by adding the metric relative to the l^{th} received branch to the present value. Such an operation results in two paths reconverging at each one of the trellis vertices at depth l+1. Select as survivor the one with the smallest metric. (In case of tie pick one at random).
$l \leftarrow l + 1$.

If $l = L+m$, stop. (One of the MLD paths is the survivor at the null state).

A suboptimal variant of the MLVD algorithm which is suitable for real time applications and will be referred to as the Real Time Viterbi Decoder (RTVD) can be outlined as follows. Its mode of operation is identical to that of the MLVD except that step 2 is repeated indefinitely. The operation of bit decoding is however performed as follows. Given $\Delta > m$, an integer called the decoding lag, the decoded bit at time ℓ, provided $\ell \geq \Delta$, is the bit at depth $\ell-\Delta$ on the surviving path which is the most likely among all the survivors at step Δ. It turns out that a majority decision on the first bit of the truncated surviving paths constitutes also an effective, yet simpler, real time decoding rule.

The same algorithm can be applied to high rate punctured codes obtained by removing channel symbols periodically from the transmitted sequences. If T (T a multiple of v) is the period of the puncturing process, a binary T-tuple $\underline{P} = (p_0, p_1, \ldots, p_{T-1})$ $p_i \in \{0,1\}$ (the puncturing pattern) specifies whether or not the i^{th} symbol in the window of period T is transmitted ($p_i=1$) or discarded ($p_i=0$). If ν is the number of 0's in $\underline{P}$, the effective rate of transmission is

$$R = \frac{1}{v} \times \frac{1}{1 - \frac{\nu}{T}} .$$

III. DECODER ARCHITECTURE

From a general standpoint, the speed of the Viterbi decoding algorithm can be optimized by applying the ACS operations on all 2^{K-1} states of the trellis simultaneously, However, as the constraint length gets larger, the amount of circuits needed grows exponentially. The decoder hereby presented uses a trellis decomposition technique in order to perform the ACS operations in a parallel and semi-parallel fashion so as to minimize the required amount of circuits for a given decoding speed. The following technique optimizes, by reducing them to a minimum, the interconnections between the accumulated metric and path registers necessary at each state.

Let us represent the state $\underline{S}$ of the encoder by:

$$\underline{S} = \underline{M}.d_{-} , \qquad (7)$$

where $\underline{M}$ is the memory of the encoder composed of the last K-2 input bits and d_{-} is the oldest bit of the state.

A state $\underline{S}_i$, called a predecessor, after an input bit d_{-}, will change to a state $\underline{S}_{i+1}$, called a successor, in the following manner:

$$\underline{M}.d_{-} \rightarrow d_{-}.\underline{M}$$

For any rate 1/n, a predecessor will have only 2 successors differing by their first bit. Furthermore, any successor will have 2 predecessors differing only by their last bit d_{-}. It is then possible to decompose the trellis into sub trellisses in the manner illustrated by the example of Figure 3.

00	00	00	00
10	10	01	10
01	01	10	01
11	11	11	11

Fig. 3 Trellis decompositioin into sub-trellisses for the case K=3.

A complete parallel implementation of the Viterbi decoder requires an ACS unit for each sub-trellis. Since an ACS unit will always have the same source and destination states, we can use ping-pong memories for each state and alternate the read/write cycles while using hardwire transfers of the output of those memories to the input memories of the appropriate ACS units.

It is also possible to use the sub-trellis approach in a semi-parallel fashion so that each ACS unit will operate on more than one sub-trellis. Consider the full parallel implementation for a constraint length K' code; where each of the states accumulated metric and path registers are connected to their own ACS units. By letting the above registers be memory modules with a depth of $2^{K-K'}$, where K>K', it is possible to implement a decoder for a constraint length K code without having to modify the present interconnections. In order ot achieve this, each memory location in the memory modules will contain the required information for a particular state and the state location in the memory module will be determined in the following manner.

If $\underline{S}$ is the state stored in a particular module in the full parallel mode, a particular address $\underline{X}$, in the module, points to state $\underline{X}.\underline{S}$ for $\underline{X}=0,1,\ldots,2^{K-K'}-1$.

Every state in a memory module will find its successors and predecessors through the hardwired process used in the parallel case. Therefore the principle of decomposition into sub-trellisses still apply and one needs only to sequence the memory addresses in the following manner to achieve complete semi-parallel operations:

```
┌ read address X;
|
└ X = 0,1,...,2^(K-K'-1).

┌ write address Y
|
| Y = M/2 M even,
| Y = (M-1)/2 + 2^(K-K'-1) M odd;
|
└ for M = 0,1,...,2^(K-K'-1).
```

IV CONFIGURATION

The decoder is composed of three sections. The first one serves to input the parameters of the convolutional code used and also to select the decoding parameters. To use the decoder, the operator can choose the following coding parameters through the data acquisition computer;

1. The constraint length K from 3 to 7;
2. The encoding rate to be selected among the values 1/2, 1/3, 1/4;
3. The code generating vectors;
4. The value of Eb/No, representing the signal upon noise ratio of the transmitting channel (assumed to be the Additive White Gaussian Noise (AWGN) channel);
5. Select the punctured code mode [6], in which case the puncturing sequence to be used must be provided. The allowed punctured rates are of the form (N-1)/N with:

   2<N<18 for rate 1/2 codes,
   2<N<12 for rate 1/3 codes N odd,
   2<N<10 for rate 1/4 codes.

After these parameters have been entered, the computer sets the configuration of the decoder and initializes the noise generator and data acquisition modules. The control program will start by calculating the output symbols for each state of the encoder as derived from the constraint length, rate as well as the code generating vectors used. These symbols are then stored in the appropriate input memory locations of the decoder. Upon reception of input symbols by the decoder, these stored symbols will be compared to the received ones and their Hamming distances will then be used as transition metrics by the decoder. If the punctured code option has been selected, the control program will also load the periodic puncturing sequence into a dummy symbol padding circuit found in the second section of the decoder.

The operator also selects the following decoding parameters:

1. The number of decision levels on each symbol (2, 4 or 8).
2. The length of the path memory (1 to 48).
3. The rule of selection of the decoded bit which can be either a variable majority decision on all the paths or a decision on a particular path.

When all the above parameters have been transfered to the decoder, the control program will choose the proper microprogram of the decoder in order to control the source and destination of the accumulated state metric and memory paths of each ACS unit. The choice of the microprogram is based on the constraint length used, which determines the amount of states present in the decoder. After this initialization phase, the control program relinquishes all control of the decoder and is left in a dormant state waiting for requests from the acquisition module to transfer the data relative to the decoded symbols.

The second section of the decoder comprises all the circuits which perform the ACS operations, as well as the storage registers for the accumulated metric and path at each state. The decoder is designed to operate on two sub-trellisses simultaneously, enabling full parallel decoding operation for codes of constraint length equal to 3. By using memory modules, as outlined in the previous section, for the accumuluted metric and path registers, semi-parallel operations becomes possible on codes of constraint lengths between 4 and 7. As noted above, such a semi-parallel operation is possible without any added circuitry or interconnections between the memory modules.

The maximum path length truncation of the decoder is 48 and then sufficient long to have negligible effect on the performance of rate 1/n codes of the longest allowed constraint length value K=7 [5].

Included in the second section of the decoder is a dummy symbol padding device utilized, whenever a punctured code is used, to insert the symbols that will have no bearing on the decoded bit decision. This enables the decoding of rate (N-1)/N punctured codes using the same procedure as for the original 1/n codes [6]. Symbol insertion is performed with a speed factor sufficient to quarantee effective insertion of the maximum number of punctured symbols between two received symbols without having to slow down the input symbol rate. This section also contains the circuitry which periodically normalizes all the state metrics in the trellis by subtracting a given value from all the metrics when they exceed the threshold. Such a normalization is required to avoid overflow on the metric values resulting usually in decoding failure.

The third and final section of the decoder contains the input and output interfaces, a digital noise generator as well as a data acquisition module. The digital noise generator is based on a pseudo-random sequence generator which compares random bit strings to a threshold value fixed by the control program. The probability of crossing this threshold is the same as the probability of error for an AWGN channel operating at the given value of Eb/No and code rate. The periodicity of the random sequence generator is over 11,000 years at the maximum symbol rate allowed, creating, for all practical purposes, effective random events for all the simulation run lengths. The digital noise generator output consists of 1, 2 or 3 bits corresponding to 2, 4 or 8 soft quantized decision levels.

The data acquisition module compresses the input and output symbols from the

decoder and stores them in one of two buffers. When one buffer is full, the software starts filling the other and requests a transfer to the mass storage control program for the purpose of providing off-line statistical analysis of the simulation results.

V CONCLUDING REMARKS.

The decoder has been constructed using TTL circuits of the Fast Schottky family. A decoding speed of 2.5 Mbit/s is achieveable under the full parallel configuration. For semi-parallel operations with codes of constraint lengths K between 4 and 7, this speed decrease exponentially by a factor of 2^{K-3}. As a matter of comparison, we can consider a software implementation on an IBM-PC running at 4.77 MHz. To execute 4 subtractions, 8 additions, 4 comparisons and 4 selections, (the basic ACS operations), the PC requires a minimum of 84 µsec while the decoder on hand needs only 400 nsec. The software running time is calculated for an assembly language program in which one of the two variables is in an internal microprocessor register while the other is in memory. If we consider all the other functions performed simultaneously by the decoder such as the transition metric computations, address sequencing for predecessors and successors, detection of a renormalization condition, choice of the decoded bit and more; a program realizing the same operations on a PC would be at least 1000 times slower. For K=7, a simulation run of 10^8 decoded bits, necessary to obtain meaningful statistics for an error rate of 10^{-5}, is realized in about 11 minutes with the decoder presented. The same simulation on an IBM-PC would take a minimum of 7 and a half days. As such, this machine provides an effective self-teaching tool for those interested in getting acquainted with error correcting techniques based on the use of convolutional codes combined with Viterbi decoding.

REFERENCES

[1] J. Conan, "**Implementation of Microprocessor-Based Viterbi Type Decoder**", Proc. Mini、Microprocessors Symp., Zurich, June 1978, pp. 87-93.

[2] J. Conan, "**An F8 Microprocessor-Based Breadboard for the Simulation of Communication Links Using Rate 1/2 Convolutional Codes and Viterbi Decoding**", IEEE Transactions on Communications, V.C-31, No 2, Feb. 1983.

[3] G.D. Forney, Jr., "**Coding system design for advanced solar transmissions**", Codex Corp. Watertown, MA Final Rep., Contr. NAS2.3637, Dec. 1967.

[4] J.K. Omura, "**On the Viterbi decoding algorithm.**" IEEE Trans. Inform. Theory, vol. IT-15, Jan. 1969.

[5] J.A. Heller and I.M. Jacobs, "**Viterbi decoding for Satellite and Space Communications**", IEEE Trans. on Communications Twechnology, Vol. COM-19, October 1971, pp. 835-838.

[6] J.B. Cain, G.L. Clark, Jr. J.M. Geist, "**Punctured Convolutional Codes of Rate (n-1)/n and Simplified Maximum Decoding**", IEEE Trans. on Inform. Theory, Vol. IT-25, PP. 97-100, Jan.1979.