A Long Constraint Length VLSI Viterbi Decoder for the DSN

J. Statman, G. Zimmerman, F. Pollara, and O. Collins
Communications Systems Research Section

A new Viterbi decoder, capable of decoding convolutional codes with constraint lengths up to 15, is under development for the DSN. The objective is to complete a prototype of this decoder by late 1990, and demonstrate its performance using the (15, 1/4) encoder in Galileo. The decoder is expected to provide 1 dB to 2 dB improvement in bit SNR, compared to the present (7,1/2) code and existing Maximum-Likelihood Convolutional Decoder (MCD). The new decoder will be fully programmable for any code up to constraint length 15, and code rate 1/2 to 1/6. This article describes the decoder architecture and top-level design.

I. Introduction

The DSN uses concatentated codes to reduce the Bit Error Rate (BER) on the telemetry channel from deep space probes to the DSN complexes. Standard coding, as used for the Voyager mission, consists of an outer (255,223) Reed Solomon (RS) code and an inner convolutional code with constraint length $K = 7$, and code rate 1/2. Decoding is accomplished by a Maximum-Likelihood Convolutional Decoder (MCD), followed by an RS decoder. A typical telemetry chain is shown in Fig. 1. Performance of this coding scheme is well understood [1],[2].

Recently [3], new convolutional codes have been discovered that provide a "2-dB coding gain" over existing codes. The highest gain, 2.11 dB, is achieved by using a (15,1/6) convolutional code, concatenated with a (1023,959) RS code. Using (15,1/6) convolutional codes with a (255,223) RS code results in an estimated coding gain of 1.8 dB. This gain can be realized by building a new Viterbi decoder for the inner code, and using the existing RS decoder. Hence, employing the newly discovered convolutional codes can result in relatively inexpensive improvement in DSN telemetry performance.

To demonstrate the new codes, an encoder for a (15,1/4) convolutional code is being added to the Galileo spacecraft. A rate 1/4 code is used instead of a rate 1/6 code, because of the limited bandwidth available on the Galileo modulator. This encoder, shown in Fig. 2, requires only a small number of parts (20 integrated circuits and 60 discrete components) and thus has a minimal impact on spacecraft complexity. A prototype decoder is being developed, capable of decoding Galileo data, but also of accepting other codes such as DSN standard codes and (15,1/6) convolutional codes. Figure 3 shows the BER versus bit SNR, for various coding schemes, with a predicted coding gain for the Galileo experiment of 1.5 dB.

The complexity of a Viterbi decoder depends on three key parameters: constraint length (i.e., degree of the generating polynomials), code rate (i.e., reciprocal of the number of encoded symbols transmitted for each information bit), and information data rate. The major complexity driver is constraint length, since the amount of hardware is roughly proportional to the number of states, which is $2^{(K-1)}$, where K is the constraint length. Hence a decoder for $K = 15$ is approximately 256 times more complex than a decoder for $K = 7$. Such a

complex decoder can be built with current VLSI technology within reasonable size limitations.

This paper describes the prototype decoder. Section 2 outlines system requirements, and Section 3 describes the top-level design. Section 4 describes in detail the architecture of the processor assembly, the unit performing the actual decoding.

II. Decoder Requirements

Requirements for the decoder can be separated into three categories:

(1) Performance. The decoder will process convolutionally coded data with constraint length up to 15 (programmable) and code rate 1/2 to 1/6 (programmable). Data rate must meet Galileo requirement (134.4 Kbit/sec), with a goal of 1 Mbit/sec. The decoder will utilize a synchronization pattern, if it is present in the uncoded data stream, to support node synch. In addition, an external node synch input will be available.

(2) Interfaces. The decoder will provide DSN interfaces, for testing in CTA 21 and for integration into DSN complexes. At a minimum these include symbol input from the Symbol Synchronizer Assembly (SSA) or the Base-Band Assembly (BBA), decoded information bits to the Frame Synchronization Subsystem (FSS), and interfaces to station monitor and control.

(3) Testability. The decoder will include testing capability for both stand-alone tests and DSN compatibility tests. In the stand-alone test, the decoder will generate a pre-programmed information bit sequence, encode it according to the desired convolutional code, add a programmable amount of white Gaussian noise to the symbols, pass the noisy symbols to the decoder proper (processor assembly), compare the decoded bits to the original sequence, and compute BER, in real time. The decoder will also provide GO or NO GO indication to the operator. For DSN compatibility testing the decoder will receive a symbol stream, and an un-encoded bit stream, decode the symbols, and compute BER.

Additional requirements concerning operating environment, size, power consumption, reliability, fault testing, and maintainability exist, but are not discussed here.[1]

[1] J. Statman, "Draft Task Plan for Large Constraint Length VLSI Viterbi Decoder," JPL IOM 331-87.5-241 (internal document), December 28, 1987.

III. Top-level Design

A functional block diagram of the decoder is shown in Fig. 4. The following is an overview of these blocks:

(1) Processor Assembly. This is the "heart" of the decoder. It consists of 256 identical VLSI chips that perform the maximum-likelihood decoding of the incoming symbol sequence. In addition, this assembly includes path memory, metric normalization circuitry, and the applicable computer, timing, and control interfaces.

(2) Simulator Assembly. The simulator assembly generates a noisy symbol sequence in three steps. First, an information sequence is generated. Next, this sequence is encoded using the appropriate convolutional encoder. Finally, a measured amount of noise is added. In addition, the simulator assembly sends the uncoded information sequence to the comparator assembly, to enable performance evaluation.

(3) Comparator Assembly. This assembly receives "true" information bits from either the simulator assembly or from an external input, and decoded bits from the processor assembly. It aligns the sequences and collects BER data.

(4) Node Synch Assembly. The node synch assembly derives node synch either from the rate of metric increase, from an embedded synch pattern, or from an external source.

(5) Erasure Signal Generator. This is an option under consideration. It is based on an algorithm [4] that compares the incoming symbols to an encoded version of the decoded information bits, to determine probable burst-error locations.

(6) SSA Interface. This module converts the signal coming from the SSA to signals compatible with the decoder. The two key operations are adjustment of voltage levels and removal of additional sign inversions added by some encoders.

(7) FSS Interface. This module sends the decoded bits to the FSS, similar to the existing MCD output.

(8) Other DSN Interfaces. More DSN interfaces are under definition. Options are interface to the Telemetry Processor Assembly (TPA) and interfaces to future DSN data network via Small Computer Standard Interface (SCSI) bus for data transfer, and General Purpose Instrumentation bus (GPIB) for monitor and control.

(9) Computer-Controller-Timing. The computer-controller-timing coordinates the modules described above by providing command, control, and monitor operations during initialization and decoding. In addition, it generates all the required timing signals and allows for extensive stand-alone testing.

The prototype decoder packaging approach is to provide for easy transfer from prototype packaging to a DSN-ready system. Standard DSN packaging techniques are used where possible. The baseline package is in two drawers, mountable in a 19-inch rack. The first drawer is based on a MULTIBUS I card cage and includes all the assemblies and external interfaces, except for the processor assembly. The second drawer includes the processor assembly.

IV. Processor Assembly Architecture

The architecture presented here is for a particular implementation of the Viterbi decoder. We start by reviewing several basic definitions and algorithms that are used elsewhere in this article. It is not intended as a Viterbi decoder tutorial, and the interested reader may read references [1], [2], [5], [6] for further information.

The Viterbi decoder tries to find the best possible match between a stream of received symbols and a path through a state trellis. The processing is sequential, i.e., using the set of symbols corresponding to a single information bit, the decoder progresses from one time-slice through the trellis to the next, while updating its decision on the most likely path and the resulting decoded bits. For a code with constraint length K, the number of states is 2^{K-1}, so in the $K = 15$ decoder there are 16384 states.

We assume here that the code rate is $1/n$. This implies that each state is connected to two preceding states and to two succeeding states, depending on whether the preceding and succeeding information bits are 0 or 1. In fact, it is convenient to organize the states in butterflies (so called because the graph of associated arithmetic resembles a butterfly). Each butterfly contains two states, and has inputs from two other butterflies and outputs to two other butterflies. For $K = 15$, the 16384 states are organized in 8192 butterflies.

The data exchanged between the butterflies are accumulated metrics. These metrics represent the probability of trellis paths, i.e., the lower the accumulated metric, the more likely is the path. There is one accumulated metric per state, or two per butterfly. Accumulated metrics are computed inside the butterfly. For each set of symbols corresponding to an information bit, the butterflies add the existing accumulated metric to the metric associated with the new symbols (so called "branch metric"), resulting in new accumulated metrics. As time passes, accumulated metrics grow, so periodically they are reduced down, or normalized.

A. Basic Trade-Offs

Several implementation choices were made and are documented below. First, the 8192 butterflies can be implemented using serial or parallel architectures, or with a hybrid serial-parallel approach. In a serial architecture, a single physical butterfly processor performs all 8192 butterflies, sequentially. In a parallel architecture, 8192 physical butterflies are used. In a hybrid approach, n physical butterflies are used, each sequencing through $8192/n$ butterflies. The *fully parallel architecture* was chosen.

Next, a choice of arithmetic method is made. The arithmetic operations include addition, subtraction, and comparisons between metrics. The decoder uses integer arithmetic and performs *bit-serial arithmetic*, or bit-by-bit operations. In this approach, the metrics (represented by 8- to 18-bit numbers) are sent serially, on a single wire, LSB to MSB. A separate TDA Progress Report article is under preparation, describing the bit-serial versus parallel arithmetic trade-offs.

Next, the method for decoder graph partitioning is selected. Butterfly interconnection can be represented by a graph with 8192 nodes, where each node corresponds to a butterfly. Each node has inputs from two other nodes and outputs to two other nodes. The partitioning selected will be described in detail in a future progress report. It is a two-level partitioning of the graph, where the first-level subgraphs correspond to printed circuit boards, while secondlevel subgraphs correspond to VLSI chips. Key features of the partitioning are (a) the graph is split among 16 *identical* boards, each with 16 *identical* VLSI chips, leading to easy implementation, (b) any Viterbi decoder of constraint length K can be built by wiring together $2^{(K-7)}$ of these chips or $2^{(K-11)}$ of these boards, and (c) the number of wires between boards and chips is relatively small.

B. Processor Assembly Elements

The processor assembly, shown in Fig. 5, consists of six major functions:

(1) Symbol Conversion. The symbols arriving into the processor assembly are 8-bit 2's complement quantities, arriving at the rate of one symbol per symbol clock. The symbol conversion module buffers the symbols into blocks that correspond to information bits (using the node synch signal), converts the symbols into sign-magnitude values, and rearranges the symbols for bit-serial transmission to the butterflies. It also

computes the sum of the magnitudes of the six symbols and transmits it to butterflies, bit-serially, LSB first.

(2) Butterflies. The butterflies are the core of the decoder. As shown in Fig. 6, each butterfly consists of two main blocks: an Add-Compare-Select (ACS) unit and a Metric Computer. The ACS uses four adders to add branch metrics to accumulated metrics, then compares the sums to select two of them for further transmission. The metric computer uses a set of adders to compute a weighted sum of the received symbols. Both the ACS and the metric computer are mathematically specified below. The complete decoder for $K = 15$ has 8192 butterflies, 32 butterflies per VLSI chip.

(3) Metric Exchange. The metric exchange function is performed by the interconnections between butterflies. Some of the metrics are exchanged inside the VLSI chip, while others are sent via wires between chips and in a backplane. All transmitted metrics must be kept aligned, i.e., the ith bit of transmitted metric is present on all metric exchange wires at the same clock period, regardless of the form of this connection.

(4) Traceback Memory. After each butterfly completes the ACS operation it sends two bits to the traceback memory. These two bits per butterfly (computed once per information bit) represent the results of the two ACS select operations. The traceback memory can be viewed as a matrix where one dimension is the number of states, 16384, and the other dimension corresponds to time, and has $3*7*K$ entries. For $K = 15$, the memory has at least 16384 *3*7*15 bits, or approximately 640 Kbytes.

(5) Traceback Processor. The traceback processor reads and writes the traceback memory to produce decoded bits [4].[2]

(6) Normalization Processor. The normalization processor monitors several accumulated metrics. When any of these metrics exceeds a computer-selected threshold, a normalization command is issued to the butterflies, to be executed during the next information bit time.

C. Butterfly Mathematical Representation and Implementation

The following paragraphs describe the equations of the ACS and the metric computer unit.

[2] F. Pollara and H. Shao, "Memory Management in Traceback Viterbi Decoders," JPL IOM 331-87.2-242 (internal document), February 12, 1987.

1. Add-Compare-Select. A diagram of an ACS is shown in Fig. 7. The accumulated metrics (16-bits) from neighboring states $i0$ and $i1$, which were previously computed in some other ACS unit, are added bit-serially to the branch metrics b_{00}, b_{01}, b_{10}, and b_{11}, provided by the metric computer unit. The operation produces the sums:

$$s_{00} = m_{i0} + b_{00}$$

$$s_{10} = m_{i1} + b_{10}$$

$$s_{01} = m_{i0} + b_{01}$$

$$s_{11} = m_{i1} + b_{11}$$

These sums are shifted into four shift registers and the smaller sum of each pair is selected by the comparators, as follows:

$$\text{if } (s_{00} < s_{10}),\ m_{j0} = s_{00} \text{ and } bit0 = 0,$$

$$\text{otherwise } m_{j0} = s_{10} \text{ and } bit0 = 1$$

$$\text{if } (s_{01} < s_{11}),\ m_{j1} = s_{01} \text{ and } bit1 = 0,$$

$$\text{otherwise } m_{j1} = s_{11} \text{ and } bit1 = 1$$

Here, m_{j0} and m_{j1} are the output accumulated metrics, and $bit0$ and $bit1$ are the results of the decisions, sent to the traceback memory.

2. Branch Metric Computer. The branch metrics are computed in the metric computer (Fig. 8) from the six received symbols, $r_0 \ldots r_5$. The implementation here is slightly different from that found in the literature, resulting in reduction of the dynamic range of the branch metrics by a factor of 2 [4]. Let r_i be represented as sign and magnitude binary numbers, as follows:

$$r_i = (s_i, r_{i6}, r_{i5}, r_{i4}, r_{i3}, r_{i2}, r_{i1}, r_{i0})$$

where s_i are the sign bits. Let $(e_0, e_1, \ldots, e_5)$ be the label assigned to the butterfly at initialization. This label is the output of an encoder making one of the state transitions of the butterfly. Because the generator polynomials of the codes considered have leading and trailing ones, each butterfly has only two possible branch metrics, and they sum to a constant (for a fixed set of symbols). Let

$$c_i = e_i \oplus s_i \qquad i = 0, \ldots, 5$$

then

$$b_{00} = \sum_{i \in A} \hat{r}_i$$

where A is the set of all i's such that $c_i = 1$, and $\hat{r}_i$ are the magnitudes of the r_i's. Also, $b_{10} = x - b_{00}$, where x is

$$x = \sum_{i=0}^{5} \hat{r}_i$$

Finally, for codes with a leading and trailing one in the generator polynomials, $b_{11} = b_{00}$ and $b_{01} = b_{10}$. This condition is met for our codes with $K = 15$. If $K < 15$, we have $b_{11} = b_{10}$ and $b_{01} = b_{00}$.

V. Conclusions

A new DSN Viterbi decoder is under development that benefits from two recent Advanced Systems developments: the successful search for long constraint length codes which yield a "2-dB coding gain," and the VLSI expertise in the Communications Systems Research Section. The top-level design, mathematical characterization, and functional specifications have been completed. The decoder is expected to be ready for testing using a Galileo encoder by late 1990.

References

[1] J. H. Yuen, *Deep Space Communications Systems Engineering*, New York: Plenum Press, 1983.

[2] R. L. Miller, L. J. Deutsch, and S. A. Butman, *On the Error Statistics of Viterbi Decoding and the Performance of Concatenated Codes*, JPL Publication 81-58, Jet Propulsion Laboratory, Pasadena, California, September 1981.

[3] J. H. Yuen and Q. D. Vo, "In Search of a 2-dB Coding Gain," *TDA Progress Report 42-83*, vol. July–September 1985, Jet Propulsion Laboratory, Pasadena, California, pp. 26–33, November 15, 1985.

[4] O. Collins, Ph.D. Thesis, California Institute of Technology, in preparation.

[5] G. C. Clark and J. B. Cain, *Error-Correcting Coding for Digital Communications*, New York: Plenum Press, 1981.

[6] R. J. McEliece, *The Theory of Information and Coding*, Massachusetts: Cambridge Press, 1977.

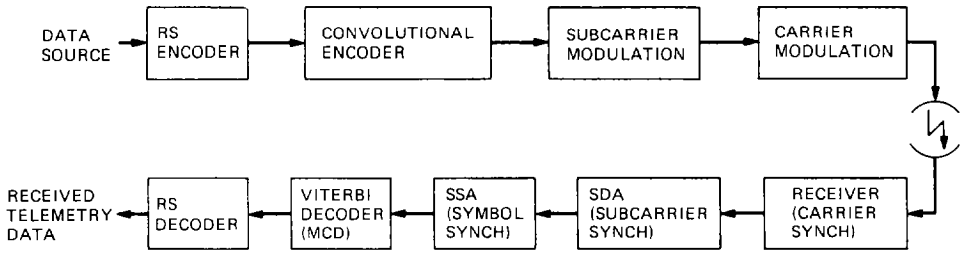

Fig. 1. Typical DSN telemetry chain.

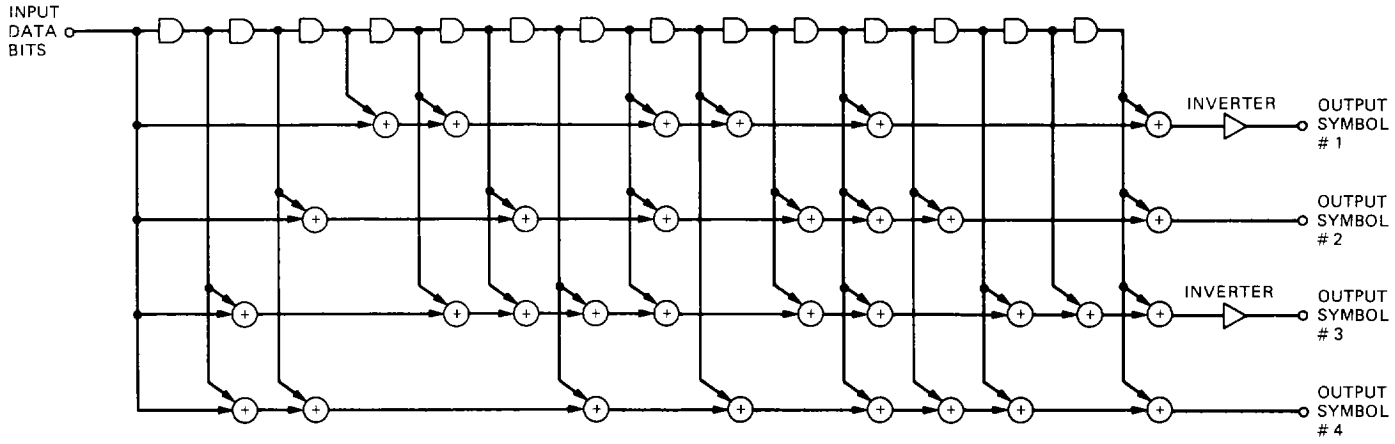

Fig. 2. A (15,¼) convolutional encoder for Galileo.

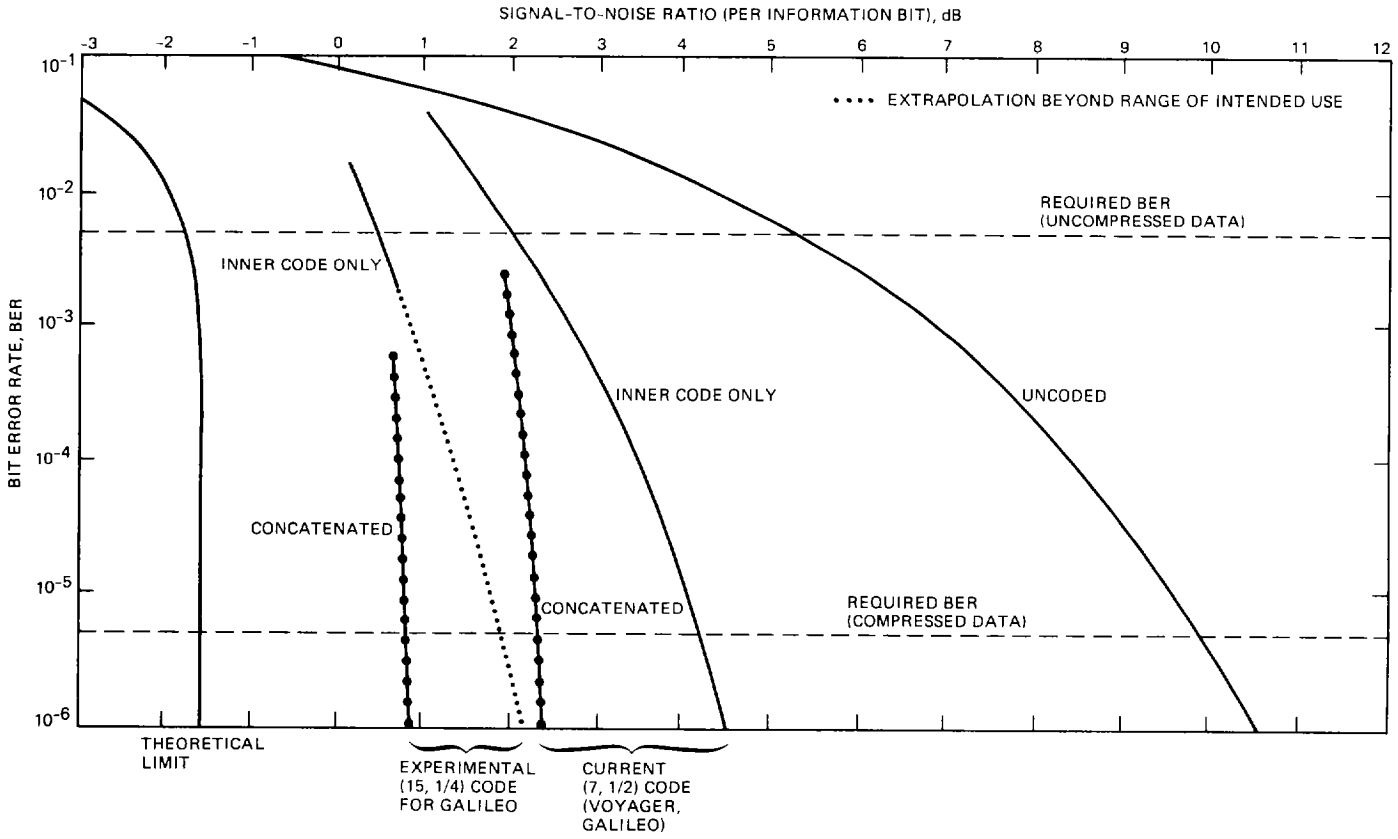

Fig. 3. Code performance.

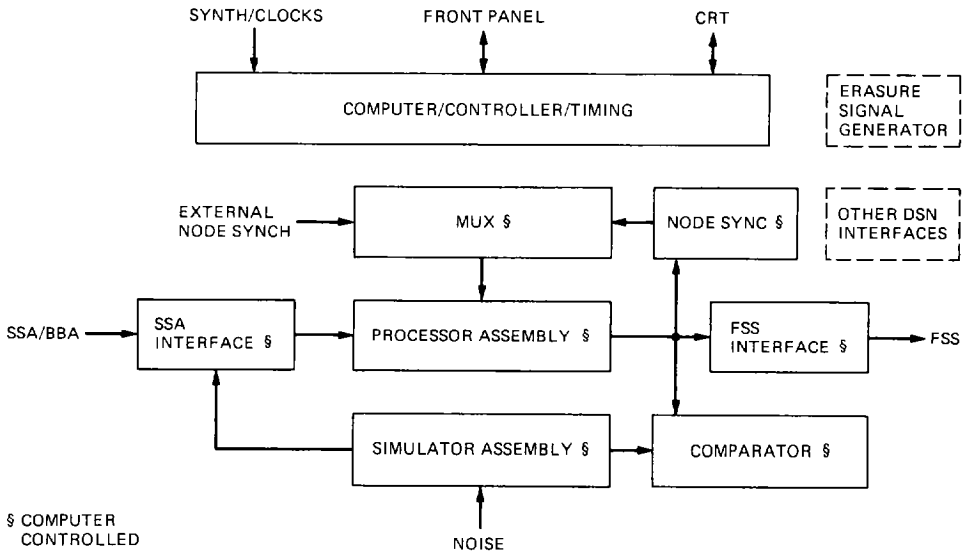

Fig. 4. Decoder functional block diagram.

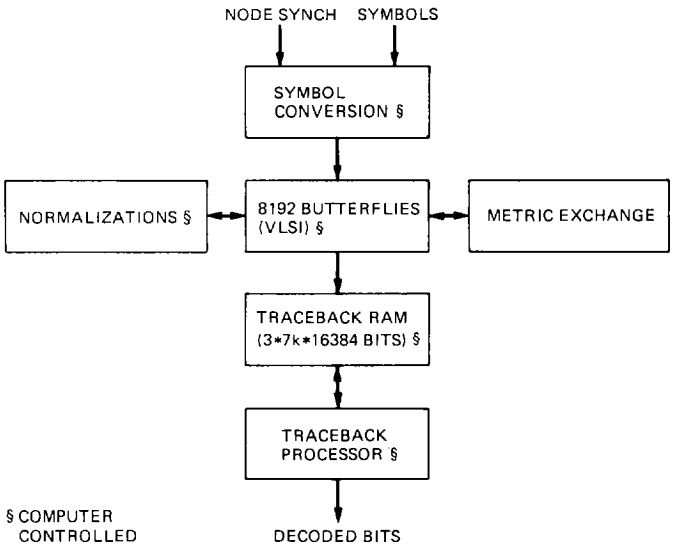

Fig. 5. Processor assembly block diagram.

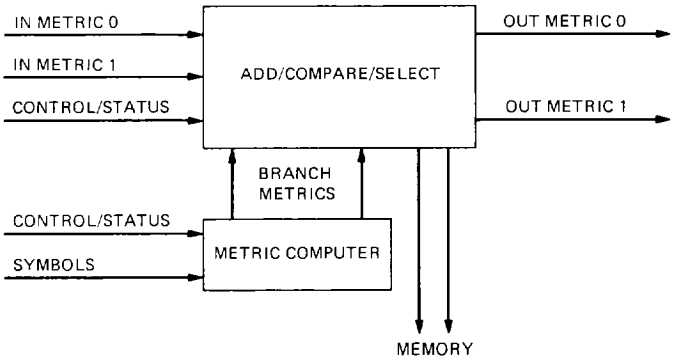

Fig. 6. Block diagram of a single butterfly.

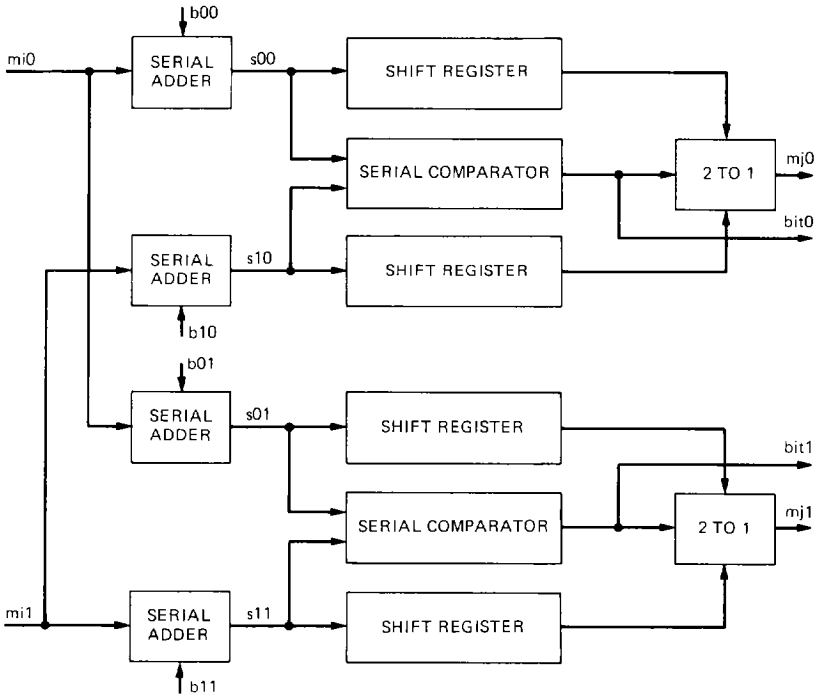

Fig. 7. Add-Compare-Select unit.

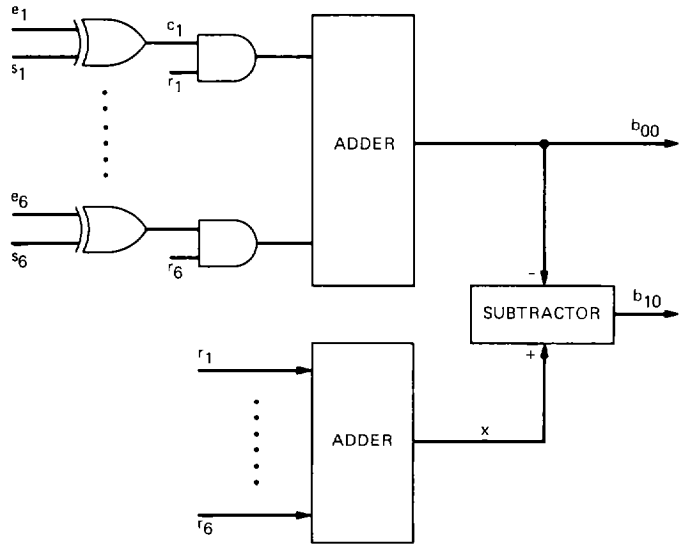

Fig. 8. Branch metric computer.