DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
SECTION COMMUNICATION, INFORMATIQUE

Technical Report EP84-R-11
Classification: Library of Congress no...

HARDWARE REALIZATION OF SEQUENTIAL DECODERS

by

Pierre Desmarteau Ing.
Canadian Marconi Company

and

David Haccoun Ing., Ph.D.
Department of Electrical Engineering

April 1984

Ecole Polytechnique de Montréal

Campus de l'Université
de Montréal
Case postale 6079
Succursale 'A'
Montréal, Québec
H3C 3A7

HARDWARE REALIZATION OF SEQUENTIAL DECODERS *

by

Pierre Desmarteau Ing.

Canadian Marconi Company

and

David Haccoun Ing., Ph.D.
Department of Electrical Engineering
École Polytechnique de Montréal

* This research was supported by Canadian Marconi Company, Montréal.

HARDWARE REALIZATION OF SEQUENTIAL DECODERS

Pierre Desmarteau and David Haccoun.

FOREWORD

This is a report on the first phase of the research project No 49242-035 supported by Canadian Marconi Company as of December 12, 1983, concerning the analysis and design methodology for the implementation of sequential decoders. The present research project consists of the following two phases, the first one ending on March 31 1984, and the second one on March 31 1985.

PHASE I

The first part of the project is the preliminary study which will define the design methodology approach.

The quantized Z - J algorithm of sequential decoding is to be broken down and analyzed to extract all required functions in terms of different states of the decoder.

A general block diagram of the proposed decoder is to be given. Design methodology will be extracted and developed. Software requirements will also be evaluated using data flow diagrams, data structured diagrams and data dictionary.

Finally guide lines are to be given for the final realization which will be reported in part II of this study. It is assumed that emulation of the proposed decoder will be performed on a 8088 microcomputer system (IBM-PC).

PHASE II

Using the design approach determined in part I, this phase of the study will analyze possible trade-offs in order to fine-tune the performance versus complexity of the final design.

A working model will be developed on an IBM-PC system and the decoder will be emulated to measure its performance.

Guide lines will finally be given in terms of scaling factors to arrive at a faster stand-alone sequential codec

This second part is not presented in this technical report. A second report shall be presented on March 31 1985.

CONTENT OF THE REPORT

After introducing the general problem of forward error correction in modern digital communication, sequential decoding is presented in section 2, in particular the so-called Zigangirov-Jelinek stack algorithm and its quantized version. The objectives, specifications, decoding process analysis and organization of the sequential decoder are presented in section 3 together with the ensuing hardware and software requirements. Finally the design approach and the breakdown of all principal elements of the decoder are given in section 4. In order to help the readers that way not be familiar with sequential decoding, a comprehensive bibliography is also provided.

1.0 INTRODUCTION

With the ever increasing use of modern digital communication the problem of providing suitable error control for these systems becomes of prime importance, and Forward Error Correction (FEC) is more and more an essential component in the system.

A very substantial research effort has been devoted over the last twenty five years on theoretical investigation and analysis of coding / decoding techniques, and a huge literature exists on the subject [1] [2]. In FEC, the main difficulties reside at the decoder, and one of the problems in the widespread use of FEC techniques is the material realization of decoders that are powerful, give low error probability, and yet are practical and not to complex to implement.

In memoryless channels where the noise is essentially white, systems using convolutional coding with probabilistic decoding are among the most powerful while being readily implementable. For convolutional codes the two techniques of probabilistic decoding are Viterbi decoding and sequential decoding [3]. These decoding techniques yield a probability of error that is exponentially decreasing with the constraint length K of the code. However the average decoding effort and the complexity of a Viterbi decoder both increase exponentially with K, whereas they are practically independent of K for a sequential decoder. As a consequence Viterbi decoding is alway used with short constraint length codes ($K < 7$), that is where only a modest performance and coding gain are required. On the other hand whenever high error performance and large coding gains are required, sequential decoding becomes an excellent choice, using whithout difficulty very long constraint length codes ($K > 40$).

In this report we present preliminary results concerning design methodology and trade off analysis of a sequential decoder realization.

2.0 SEQUENTIAL DECODING

Sequential decoding is a suboptimal decoding procedure for convolutional codes; it is essentially a tree searching procedure that attempts to find the most likely transmitted sequence or transmitted path in the encoded tree [3].

A sequential decoder has a replica of the encoder used at the transmitting end, so that all possible transmitted coded sequences can be locally reproduced at the receiving end. The search for the most likely path is performed one branch at a time along the most promising path among all those examined. For each tree branch j examined a likelihood value or "metric" γ_j is computed. These branch metrics are essentially the log likelihood function $\log P(\underline{Y}_j/\underline{X}_j^{(\underline{U})})$ between the coded symbols $\underline{X}_j^{(\underline{U})}$ corresponding to a possible data input U_j and the received sequence $\underline{Y}_j$.

Assuming a discrete memoryless channel (DMC) the branch metric along the paths are cumulative so that over a tree of length L branches, just like Viterbi decoding the objective of a sequential decoder is to find the path $\underline{U}$ for which the toal accumulated metric

$$\Gamma_L^{(\underline{U})} = \sum_{i=1}^{L} \gamma_i$$

is maximum over all possible tree paths. However unlike Viterbi decoding which uses the trellis structure of the code and examine all distinct paths, a sequential decoder uses the tree structure of the code and follows only that path in the tree that appears to be the most likely. As a consequence the computational effort is constant but large for Viterbi decoding whereas it is on the average typically very small but, unfortunately, highly variable for sequential decoding. This computational variability is one of the principal drawbacks of sequential decoding and several methods have been devised to reduce it [4] [5] [6].

There are two principal sequential decoding algorithms. The Fano algorithms [7] and the Zigangirov-Jelinek (Z-J) stack algorithm [8]. Only the stack algorithm is considered in this report.

2.1 The Z-J Stack Algorithm

The Z-J is a very simple algorithm using a list of the already searched paths ordered in decreasing order of their accumulated metric values. This list is called a STACK.

The top of stack is the path with the largest accumulated metric among the paths in the stack. The top of stack node is the one which is always searched further (i.e. extended). After each extension the stack is reordered so that a path whose metric is ever increasing will continue to be searched further. Should its metric decrease and drop from the top position that path will be properly stored in the stack and the new top node will be extended. The decoding algorithm is then:

1- Compute the metrics of all successors of the top node and enter the new paths in their proper place in the stack.

2- Remove from the stack the nodes whose successors were just inserted.

3- Find the new top node. If the new top node is the final node, stop. Otherwise go to step 1.

2.1.1 Quantized Jelinek Stack Algorithm

A physical problem with the Z-J stack algorithm is to keep the stack exactly ordered after each extension. To overcome this difficulty a

modified Jelinek algorithm was developed using SUBSTACKS. In each substack (Q) are stored all those nodes whose metric value (Γ_n) lies within a certain range according to following quantizing rule:

$$Q\,H \leq \Gamma_n < (Q + 1)\,H$$

where H is the substack quantizing step in metric values. This modified Z-J algorithm is sometimes called the JELINEK algorithm.

In this quantized version, the search for the top node is reduced to the search of the highest nonempty substack. Therefore the algorithm now becomes.

1- Compute the metrics of the successors of any node (see note 1) from the highest nonempty substack and enter them in their proper substack.

2- Remove from the stack the node whose successors were just inserted.

3- Get a node from the highest nonempty substack. If this node is the final node, stop. Otherwise go to step 1.

This algorithm which is now quite practical will be used to implement the proposed sequential decoder.

Note 1: For practical purposes the easiest way is to take the last node entered in the highest substack. (LIFO or last in, first out).

3.0 PROPOSED SEQUENTIAL DECODER

A sequential decoder using quantized Z-J stack algorithm is to be implemented. A general block diagram is shown in figure 3.

3.1 Objectives

The following objectives are to be observed:

1- Design a working prototype model which could be used as a reference for sequential decoders implementations.

2- Decoding speed is not a principal parameter, but the prototype target is 600 bits per second or better. However, depending upon the amount of statistics that may have to be collected, this target speed may be substantially reduced.

3- Maximum flexibility in terms of programmability is required. Programmable parameters requirements are:

Constrain length	: $0 < 32$ bits
Message length	: $0 < L < 512$ bits
Rate	: 1/2, 2/3 and 3/4
Code	: Convolutional with down loadable connecting vectors matrix

4- Prototype has to be easily transportable. A 16 bits microcomputer system (IBM-PC or equivalent) is a good approach to flexibility and transportability.

5- This prototype is not necessarily the end product, but could rather be considered as a development tool enabling sequential decoders emulation. In this case, the study would give guide lines and scaling procedure to follow in order to design smaller and/or faster stand-alone sequential decoders. It is worth while to mention that these design parameters may be changed and properly modified in accordance with results and experiences obtained from the performance testing of the hardware sequential decoder [9], and/or other software packages of computer simulation of sequential decoding that exist at École Polytechnique.

3.2 Specifications

3.2.1 Encoder

a) Code

Type	: Convolutional
Coding rate	: Programmable, R = 1/2, 2/3 or 3/4
Constraint length	: Programmable, $0 < K < 32$
Message length	: Programmable, $o < L < 512$ bits

b) Input (from user)

Type	: Serial binary synchronous
Format	: L message bits, following (K-1) zero bits (TAIL)
User rate R_s	: 600, 800 or 900 bits/sec respectively for code rate of 1/2, 2/3 or 3/4. A constant decoding speed is assumed for any coding rate.
Timing	: Synchronisation is external (user)

c) Output (to DMC channel simulator)

Type : Serial, binary synchronous

Encoder rate : 1200 symbols/sec. This is assuming message length of 512 bits and a constraint length of 32. Otherwise, encoder output rate is:

$$\text{OUT RATE} = R_c = Rs/R$$

where $R = n/v = 1/2,\ 2/3$ or $3/4$

L = message length

K = constraint length

$K-1$ = tail length

3.2.2 Decoder

a) Code

Same as encoder (3.2.1 a)

b) Input (from DMC channel simulator)

Type : Serial

Quantization : Programmable

2 levels (hard quantization)
4 or 8 levels (soft quantization)

Rate : 1200 symbols/sec

Timing : External synchronization

c) Output (to user)

Type	: Serial binary synchronous
Format	: L message bits then (K-1) zero bits (TAIL)
Rate	: 600, 800 or 900 bits/sec respectively for code rates of 1/2, 2/3 or 3/4
Timing	: Synchronization is external (user)

d) Characteristics

Type	: Sequential
Algorithm	: Quantized Z-J stack algorithm
Speed	: 1200 calculations per second minimum
Branch metrics	: Range from - 128 to + 127
Accumulated metric	: Range from -4096 to + 4095
Substack quantizing	: Steps of 8 metric units
Message length	: Programmable, $0 < L < 512$ bits

3.3 Decoding Process Analysis

The decoding process consists formally of the following sub-processor or elementary decoding operations.

1- Read the received symbol from the input buffer.

2- Decode the incoming symbol.

2.1- Get top of stack node.

2.2- Perform branch metric calculation for each branch. This is to extend all branches of top of stack node (FATHER NODE).

2.3- Calculate accumulated metric for each successor node. Also generate data elements to be stored in stack.

2.4- Perform SUBSTACK entry (insertions in proper substack).

2.5- Build records for all successors nodes.

2.6- Perform STACK operation.

2.7- Erase father node from stack and substack (see note 2).

3- Get new symbol then go to step 1, unless this is the end of block, in which case proceed with step 4.

4- Recover decoded path and load decoded sequence in output buffer.

5- Reset substack and stack, then go to step 1.

In this process, data source and sink are respectively the input and output buffers. The data storage are the stack, the substack and branch metric look-up table.

It is assumed that the input buffer is large enough to handle incoming received symbols while the decoder is "back searching".

A data flow diagram of the decoding process is given in figure 1.

A structural chart is also given in figure 2 where Yourdon's structured programming technique [10] has been used.

Note 2: The father node is not totally erased from the stack, but it cannot be extended anew.

3.4 Memory Organization

In addition to the memory (RAM) requirements for the microcomputer operating system, the decoder requirements for data storage are as follows. They are given in ISPS (Instruction Set Processor Specification).

1- Stack (82 K bytes)

```
STACK_DATA [0:4095] <0:159> = NODE [0:4095] [0:9] <0:15 >

ACCUMULATED_METRIC <0:12> = NODE [0] <0:12>
            DEPTH  <0:8 > = NODE [1] <0:8>
        NEXT_NODE  <0:11> = NODE [2] <0:11>
           FATHER  <0:11> = NODE [3] <0:11>
            STATE  <0:95> = NODE [4:9] <0:15>
            SPARE  <0:17> = NODE [0] <13:15>
                            NODE [1] < 9:15>
                            NODE [2] <12:15>
                            NODE [3] <12:15>
```

2- Substack (2K bytes)

```
SUB_STACK_DATA [0:1023] <0:15> = POINTER [0:1023] <0:15>
```

3- Input buffer (20K bytes)

```
IN_BUFFER [0:10000] <0:11> = BRANCH [0:10000] <0:3> <0:23>*
```

4- Output buffer (5K bytes)

```
OUT_BUFFER [0:10000] <0:2>
```

* In soft quantization, up to 3 bits are used (8 levels)

3.5 Hardware Requirements

In this first phase, no hardware will be described other than standard IBM-PC microcomputer with the following standard equipment:

1- IBM-PC computer with monochrome display

2- 512 K bytes of random access memory on board

3- COMBO communication (RS-232) card

4- Two 5 1/4 inches double sided/double density disk drives

Any other special hardware equipment and/or interface, if required, will be described in the second part of the study (MARCH 1985).

3.6 Software Requirements

The IBM-PC computer used utilizes standard IBM disk operating system with IBM basic interpreter.

It is beyond this first technical report to decide in which language the software will be developed but one of the following options will be used:

1- PASCAL; because of its structured characteristics and high transportability. A Pascal compiler (down to machine language) would be required.

2- BASIC ; in this case, this program would be developed in basic then compiled. A basic compiler would then be required.

In either cases, some subroutines will have to be designed in machine language (assembler) for efficiency (speed) purposes.

The software will be designed using structured programming techniques.

4.0 DESIGN APPROACH

A high speed sequential decoder had been developed recently at École Polytechnique de Montréal. A complete technical description [9] and a technical report of this decoder are available [11].

A hardware microsequencer capable of parallel processing was used to achieve the required decoding speed in excess of 1 mega-symbols per second. Fifteen micro instructions were implemented to realize the decoding algorithm.

The proposed design approach is retaining the same decoder algorithm, but the hardware implementation is changed to suit the microprocessor sequential processing requirements. System block diagrams for the convolutional encoder and the sequential decoder are shown respectively in figures 4 and 5 and are described in the following sections.

4.1 Convolutional Encoder

A typical convolutional encoder configuration is shown in figure 4.

At this stage, this encoder is presumably not to be implemented as this study is concentrating on the sequential decoder. All sorts of available off line sources could be used to test the performance of the decoder.

Basically the encoder output data would come from a disk file originally recorded using one of the following sources:

1- Data base available from program on high speed sequential decoder recently developed at École Polytechnique.

2- The software module used as a "local encoder" (refer to section 4.2) could be used to encode any serial binary source available. The tail (zeros) bits would be stuffed at proper positions to form the required block structure of (L + K - 1) bits.

The noise sequence is also to come from a disk file. Some typical noise sequences would be stored on disk for repetitive testing.

The output of the DMC channel simulator is also to come from a disk file. This file is to be generated by adding (exclusive OR bit by bit) noise sequences (from noise file) to data sequences (from data file).

4.1.1 User Source

It is assumed at this stage that the user source is stuffing the (K-1) zero bits for the TAIL, every (L) bits of message. In practice insertion of the tail should be transparent for the user, but it is beyond this work to decide at which level this insertion will be done.

User bit rate is tentatively specified to be 600, 800 or 900 bits per second respectively for coding rate of 1/2, 2/3 or 3/4. This is including the tail or last (K-1) bits of each block from the source.

This leads to a worst-case effective data rate (message bit rate, excluding the tail) of 300, 400 or 450 bits per second, as is derived from the following assumptions on the L and K parameters.

$$1 \leq K \leq 32$$
$$1 \leq L \leq 512$$

$$1 \leq \frac{(L + K - 1)}{L} \leq 2 \qquad \text{for } L \geq K$$

4.1.2 Rate Converter (1: n bits burst)

This rate converter, is used as an input buffer to get the (n) bits to be shifted forward into the encoder memory elements. It provides bursts of (n) pulses repetitive every (n/CLK) second to the shifting clock of the (n * K) shift register.

4.1.3 Encoder

The convolutional encoding is done formally by the following modules.

1- SHIFT REGISTER (n * K bits)

2- CONNECTING VECTORS TABLE (v vectors of K bits)

3- "AND" LOGICAL FUNCTION (v * K bits out)

4- EXCLUSIVE "OR" FUNCTION (v bits out)

5- OUTPUT MULTIPLEXER (v: 1)

This is a classical convolutional encoder. Connecting vectors are used in conjunction with the AND function to MASK the shift register bits in order to enable them to either or not participate to the final parity check calculations (EXCLUSIVE OR). These functions are easily implemented on a microcomputer. The output multiplexer gives back a serial output to the DMC channel simulator.

An equivalent electrical schematic diagram is given in figure 6. In this figure, a dot represents as usual an electrical connection. Figure 6 is also the representation of the output of the AND function of one bit of some connecting vector and the corresponding bit of the (n * K) shift register.

4.1.4 Configuration Set-Up Table

This table is storing all programmable parameters which define the encoder configuration. Proper interface is provided for the operator in order to program the encoder configuration. The same table will also be used to preset decoder configuration according to desired requirements.

4.2 Sequential Decoder

The general system block diagram is given in figure 5.

The system concept is close to the one used in the original high speed decoder. This structure is expanded to enable higher coding rates (2/3 and 3/4).

This block diagram is a subjective representation of the decoder structure, at the system level. It may not be the final data flow breakdown implementation. Nevertheless it is showing the main modules to be developed, as well as how these modules are connected to one another.

4.2.1 Multilevel Input Line Receiver

Input signal from the DMC is buffered using an analog demultiplexer (1: v) symbols and (v) sample-and-hold circuits. This is to store and preserve the absolute value of the input signal level for soft quantization.

Quantization is achieved using (v) analog to digital converter. Each converter is programmed by the configuration set-up table to have (q) quantizing bits, that is 2^q levels ($1 \leq q \leq 3$ for 2, 4 or 8 levels).

4.2.2 Input Buffer

While receiving noisy sequences, the sequential decoder will have to back-up in the decoded tree (BACK SEARCH) whenever an INCORRECT PATH is recognized. In order not to lose any input information, an input buffer must be used. This buffer has to be large enough to store all input symbols while encountering the worst case estimated "time lost" to the back search. An initial size of 10 kilo entries of (v * q) bits is contemplated. This value could be readjusted at a later date and upon results from tests. Synchronization of incoming symbols writing, and decoder reading of input buffer is anticipated to be one of the implementation problem that must be solved. To that effect a special timing module is planned.

4.2.3 Branch Metric Calculator

Branch metric calculation is done by comparing the "distances" between each received (v * q) symbols and the best path (top of stack) node extended branches. Extention of the top of the stack node is done using a "LOCAL CONVOLUTIONAL ENCODER". This encoder is identical to the one described in section 4.1 and figure 4. Each of the $2^{(v-1)}$ branch sequences is estimated by the local encoder and compared to the received sequence.

The output of the distance comparator is translated in branch metric value through a BRANCH METRIC LOOK-UP TABLE. This table is initially loaded from disk. Its content is function of the code rate and the channel and is readily calculated before hand.

4.2.4 Accumulated Metric Calculator

Each extended branch metric is added to the previous accumulated metric in order to get the new total metrics. Each new accumulated metric is also divided by 8 to find the substack in which the corresponding survivor node is to be stored. The metric calculator module also stores the decoder state at each decoding step. Accumulated metric calculation is repeated for each (2^{v-1}) branches at each extention.

4.2.5 Data Formatter

The data formatter module is collecting the required data from other modules in order to built node data elements to be stored into the stack.

4.2.6 Output buffer

After a block of (L + K - 1) bits is successfully decoded the decoded sequence is retrieved from the stack by the "received sequence recovery" module. This consists of stepping backward in the decoded tree and extract the right path information sequence. This sequence is then loaded into the output buffer to be shifted out back to the user.

REFERENCES

[1] I.M., JACOBS, "Practical Applications of Coding", IEEE Trans. Inf. Theory, IT-20, pp. 305-310, May 1974.

[2] E.R., BERLEKAMP, "The Technology of Error-Correcting Codes", Proc. IEEE, Vol. 68, pp. 564-593, May 1980.

[3] V.K., BHARGAVA, D., HACCOUN, R., MATYAS, P., NUSPL, "Digital Communications by Satellite", J. Wiley, N.Y., 1981.

[4] D., HACCOUN, M.J., FERGUSON, "Generalized STack Algorithms for Decoding Convolutional Codes", IEEE Trans. on Inf. Theory, Vol. IT-21, No. 6, Nov. 1975.

[5] D., HACCOUN, A., JANELLE, "Adaptive Sequential Decoding by Stack Algorithms", 22nd Midewest Symposium on Circuits and Systems, Philadelphia, pp. 171-175, June 1979.

[6] P., CHEVILLAT and D. COSTELLO, " A Multiple-Stack Algorithm for Erasurefree Decoding of Convolutional Codes", IEEE Trans. on Comm., COM-25, Dec. 1977.

[7] R.M., FANO, "A Heuristic Discussion of Probabilistic Decoding", IEEE Trans. on Inform. Theory, Vol. IT-9, April 1962.

[8] F. JELINEK, "A Fast Sequential Decoding Algorithm Using a Stack", IBM Journal of R. and D., Nov. 1969.

[9] C., MORIN, "Conception d'un décodeur séquentiel rapide", Thèse M.Sc.A., École Polytechnique de Montréal, Nov. 1980.

[10] E., YOURDON, "Managing the Structured Techniques", Yourdon Press, N.Y. 1979.

[11] D., HACCOUN, "High Speed Sequential Decoder", Research Report for Canadian Marconi Company, March 1984.

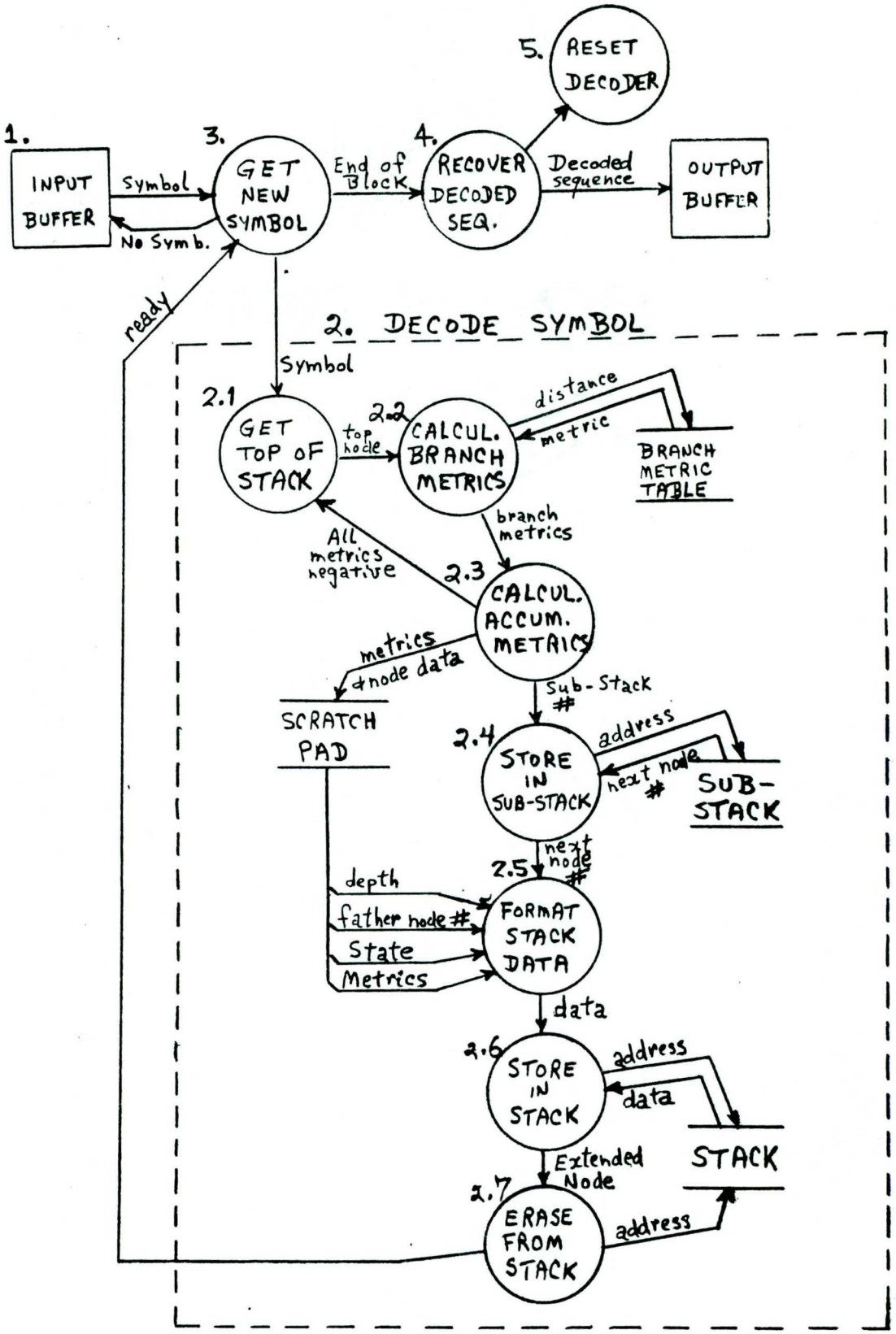


FIGURE 1

DECODING PROCESS DATA FLOW DIAGRAM

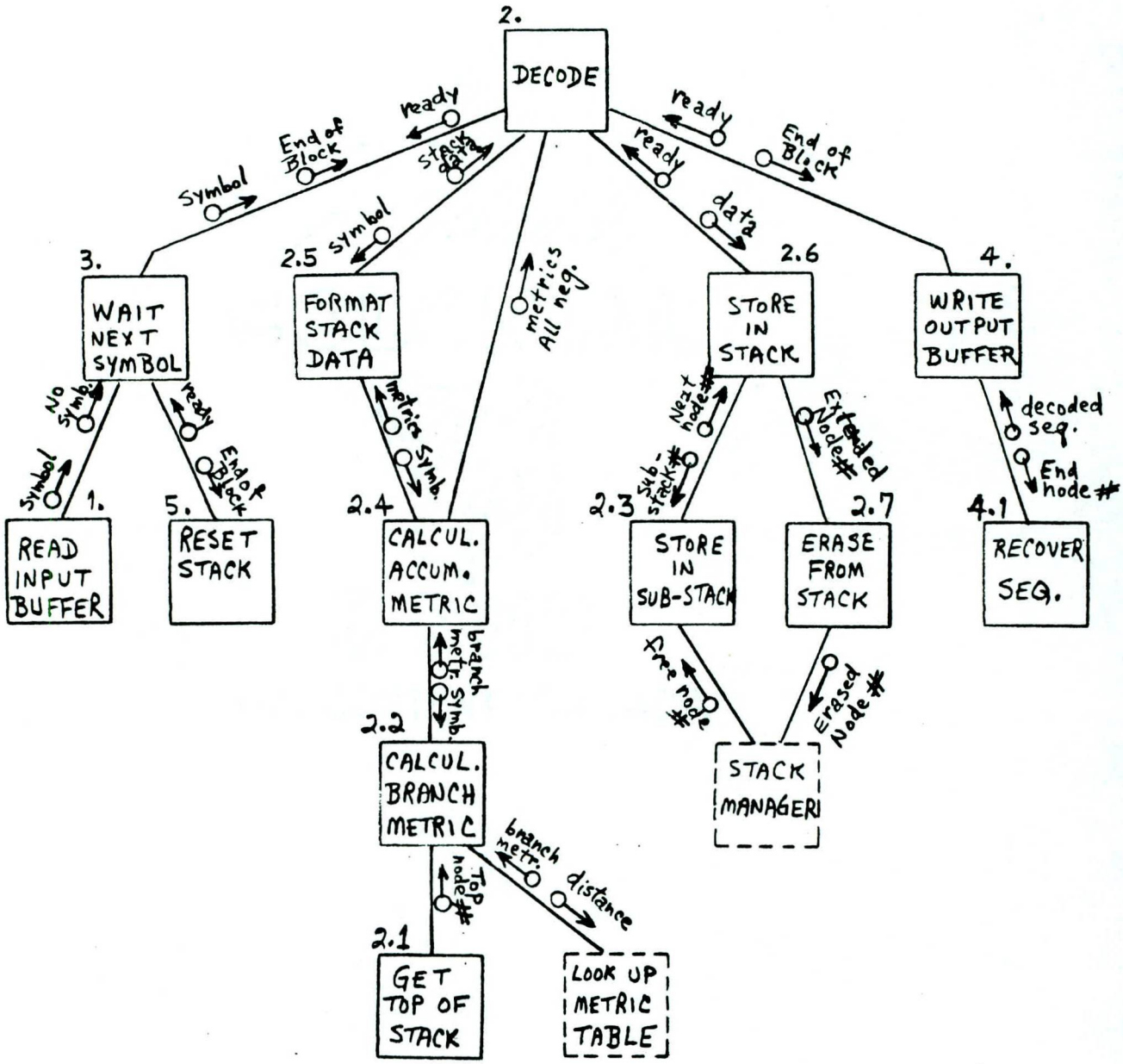
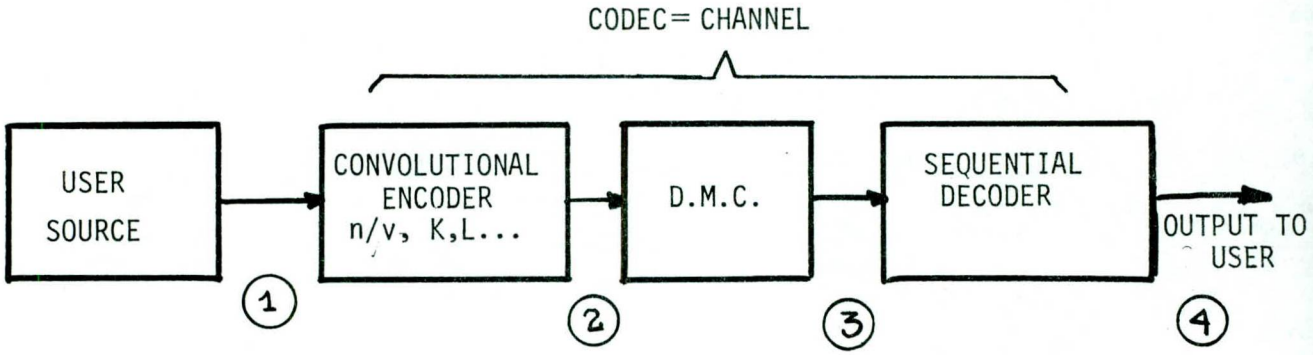


FIGURE 2

DECODING PROCESS STRUCTURED CHART

(TOPDOWN APPROACH)



(1) INPUT TO ENCODER: User bit rate R_s = 600, 800 or 900 bits/sec respectively for rate R = 1/2, 2/3 or 3/4. Format is L bits of message followed by (K-1) bits for the TAIL.

(2) OUTPUT OF ENCODER: Binary, channel rate $R_c = R_s/R$ = 1200 symb/s.

(3) INPUT TO DECODER: OUTPUT OF ENCODER with ADDED GAUSSIAN NOISE. PROPERLY QUANTIZED.

(4) OUTPUT OF DECODER: same as (1)

FIGURE 3

CODEC GENERAL BLOCK DIAGRAM

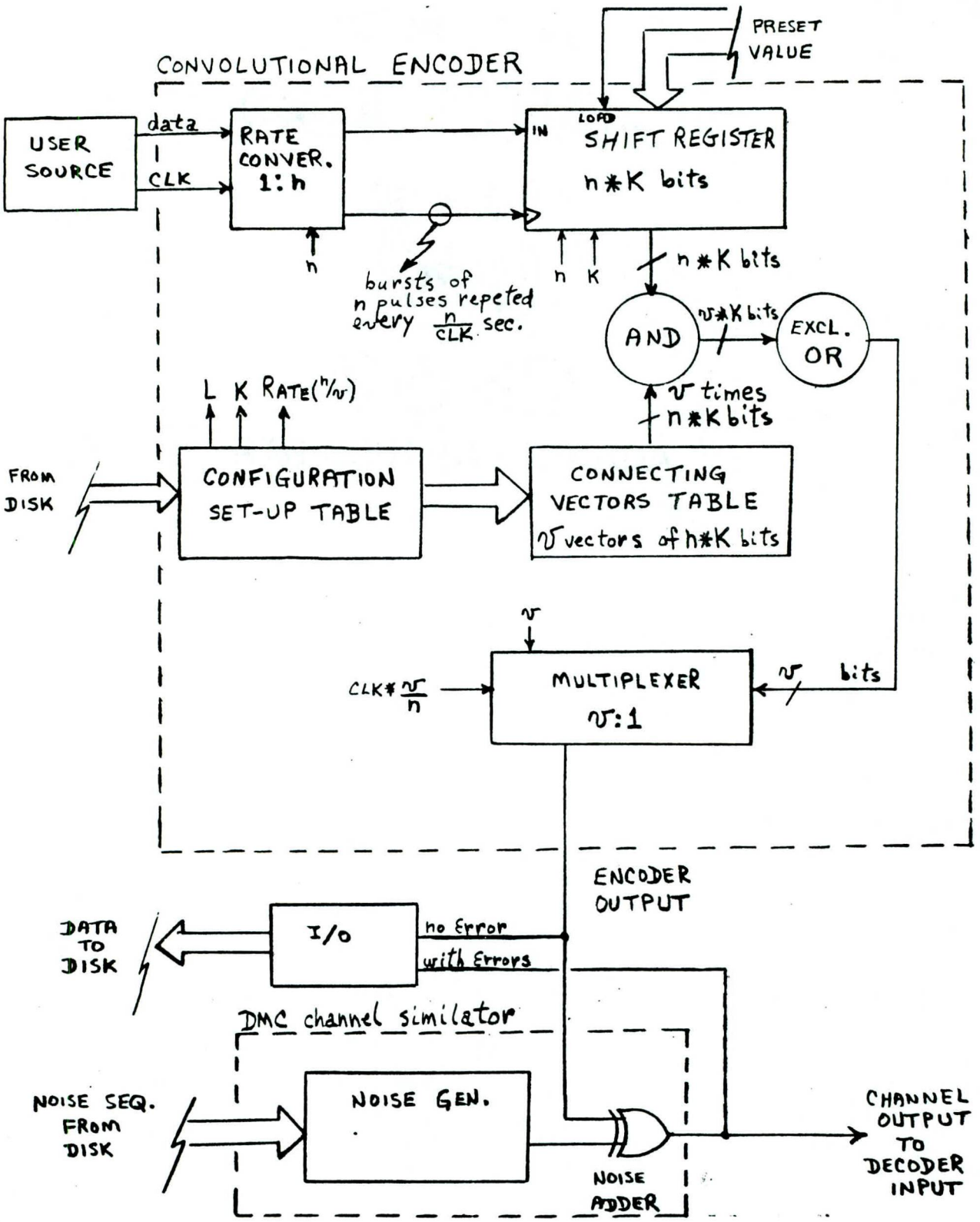


FIGURE 4

ENCODER AND DMC SIMULATOR BLOCK DIAGRAMS.

$K \leq 32$; $L \leq (512 - K)$; $R = n/v = 1/2;\ 2/3,\ 3/4$.

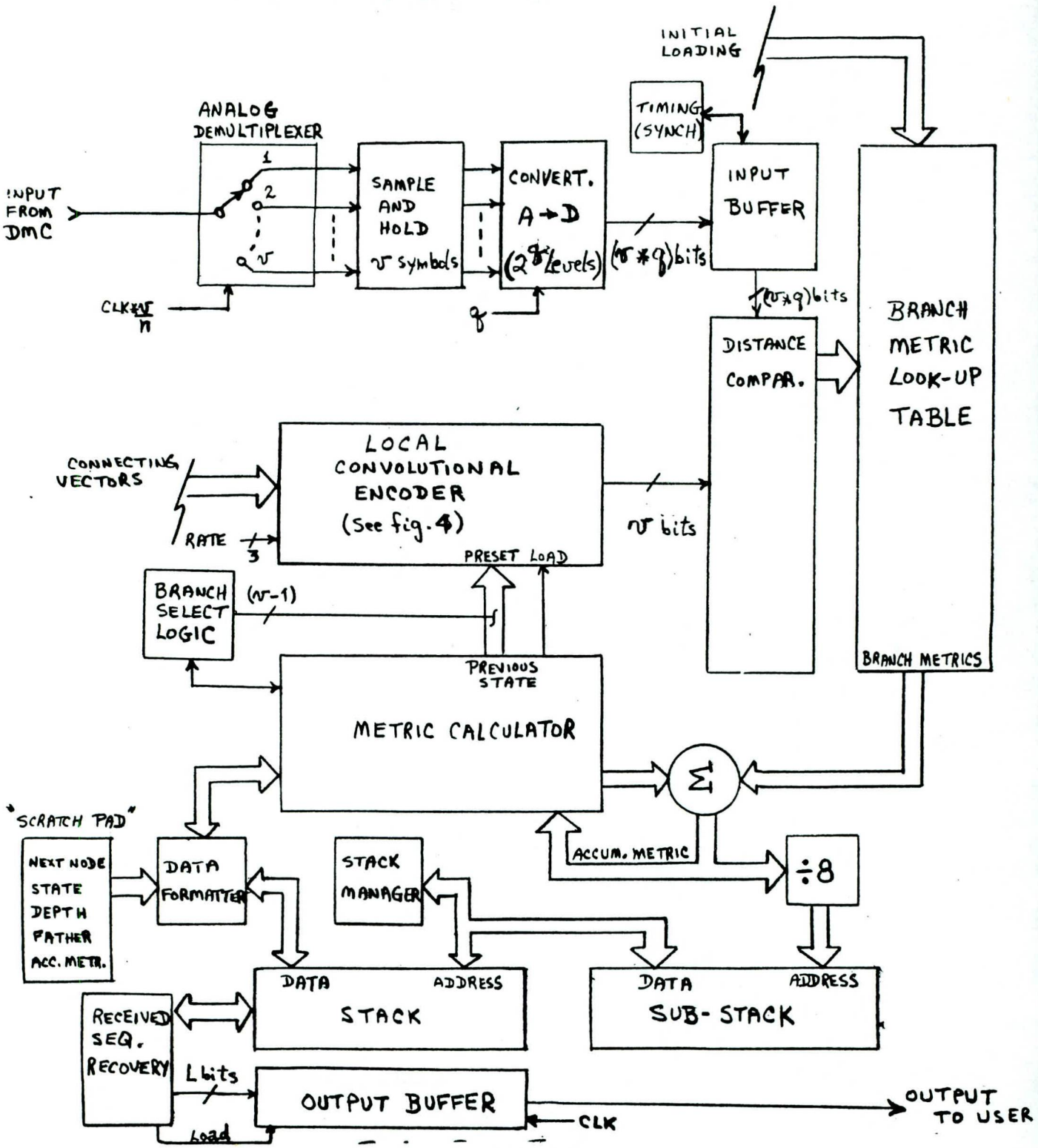


FIGURE 5

SEQUENTIAL DECODER BLOCK DIAGRAM

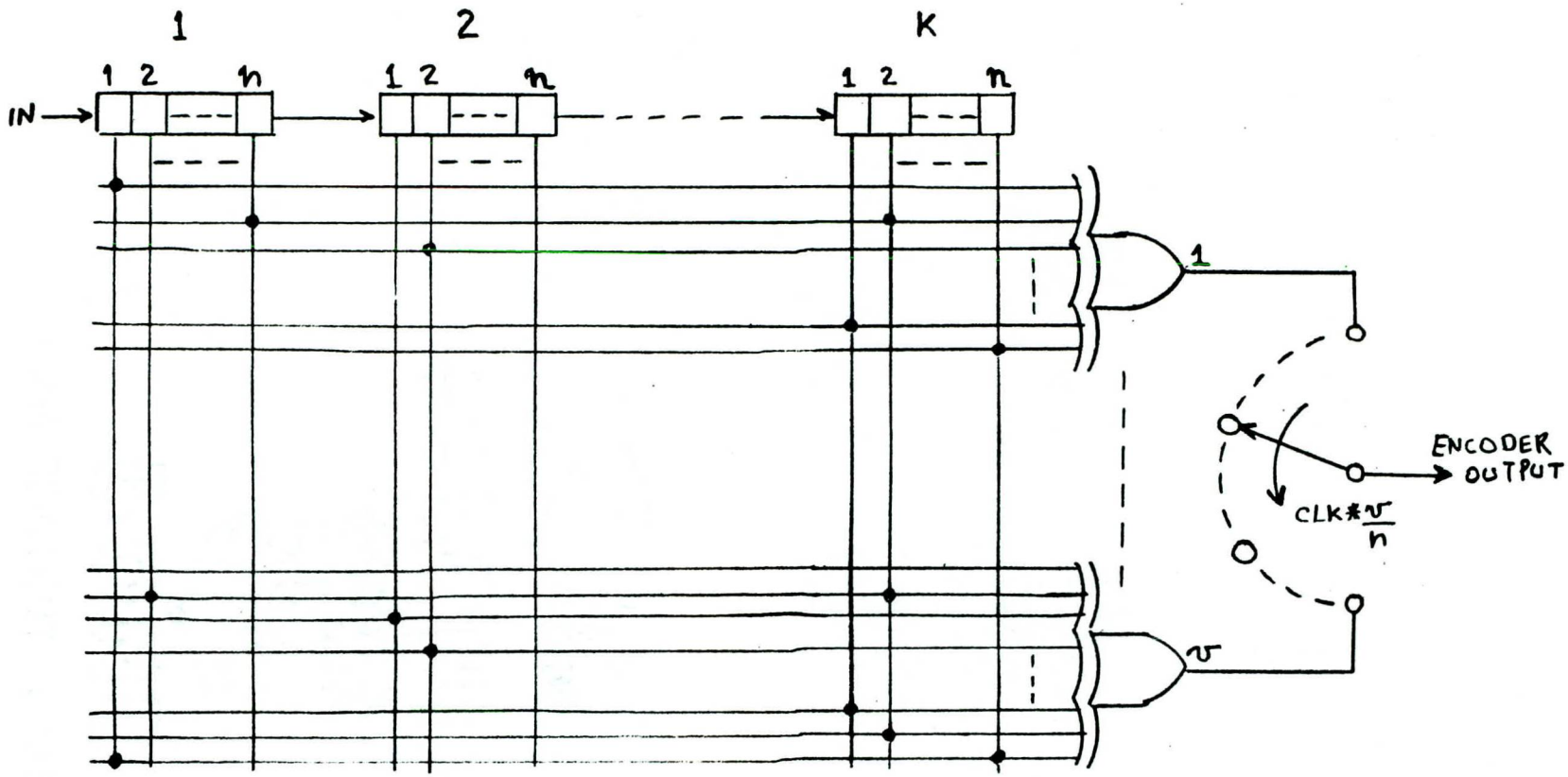


FIGURE 6

CONVOLUTIONAL ENCODER CONNECTION DIAGRAM

Each dot represents both an electrical connection and the AND function performed between a memory element and a connecting code vector.