

NORTHEAST RADIO OBSERVATORY CORPORATION

HAYSTACK OBSERVATORY

WESTFORD, MASSACHUSETTS 01886

5 October 1984

Area Code 617  
692-4764

TO: VLBA Acquisition Group  
FROM: S. Wilson  
SUBJECT: Error Correciton for VLBI

February 18, 1984

To: R. Escoffier, L. D'Addario

From: S. Wilson

Re: Error Control Coding for VLBA Recording  
Part 1: Extended Hamming Codes

If Hamming codes are "extended" by adding an overall even-parity bit on the conventional Hamming codewords, we form a code capable of correcting all single error patterns and flagging all patterns with an even number of errors. Since single-error correction and double-error detection are the most likely events, these codes are sometimes known as SEC-DED, and are widely used in computer memories after a further shortening. (The PDP 11 series uses a (22,16) SEC-DED code on internal memory words to correct single errors within a memory location, and to interrupt the machine if a double error is detected.) These codes are easily implemented with short shift registers and a small amount of miscellaneous gates, and though not as powerful as more complicated codes, they give interesting performance as described below.

In this note I shall present the performance of these codes for a number of blocklengths (and correspondingly several redundancies) as a function of raw channel error probability,  $p$ . The first four extended Hamming codes have parameters tabulated below:

$n$	$k$	$r=k/n$	overhead, %
32	26	0.81	19
64	57	0.89	11
128	120	0.94	6
256	247	0.965	3.5

Encoding is performed with an  $(n-k)$ -bit shift register with feedback, and in all cases above, this amounts to at most a few MSI chips. Decoding begins with computation of the syndrome, again done with an  $(n-k)$ -bit shift register virtually identical to the encoder. Based on the syndrome pattern ( $n-k$  bits) the decoder attempts to determine the error pattern and correct it, or to flag the codeword as containing an uncorrectable error. For the codes above, this is most easily done in ROM. The ROM stores the location of the single error if the syndrome corresponds to a correctable pattern or an error flag otherwise. Thus the ROM size is  $2^{(n-k)}$  by  $\log_2 n$ . In the worst-case above this is only 512 by 8. Of the 512 locations, 256 would contain error locations for the single error (at least we think they're single errors) while the remainder would store "FLAG".

To summarize, if zero or one error occur among  $n$  bits, the decoder releases a correct block of  $k$  bits. If an even number (non-zero) of errors occur, the word is correctly flagged. If three, five, etc. errors occur, then a decoding failure (DF) occurs. The decoder believes a single error occurred and attempts to correct it, introducing additional error into the decoded block. An expression for probability of decoding failure is then

$$P[DF] = \sum_{i=3,5,7,\dots}^n \binom{n}{i} p^i (1-p)^{n-i}$$

For reasonably small  $p$ , only the first term in the sum is significant.

When the decoder releases an incorrect block, only a few of the  $k$  bits will typically be incorrect. We can estimate the decoded bit error probability by arguing that the triple error patterns are the dominant cause of failure, and the decoder thinks a single error somewhere else actually occurred. Thus typically four of the  $k$  bits are wrong, and probability of decoded error is roughly  $4/k$  times  $P[DF]$ . This is plotted in Figure 1. To interpret, we see that if  $p=10^{-3}$  then the output error probability for the  $n=128$  code is  $10^{-5}$ . This gain is with an overhead of only 6%.

The decoder "erases" blocks occasionally, and it is important to know how often this occurs. All even-weight error patterns produce such dismissal; again for small  $p$  only the double-error events are significant.

$$P[\text{dismissal}] = \sum_{i=2,4,6,\dots}^n \binom{n}{i} p^i (1-p)^{n-i}$$

Figure 2 provides a plot of this probability for the extended Hamming codes, and we note that if a throw-away rate of a few percent is tolerable, as it probably is for the VLBA, then channel error rates as high as  $2 \times 10^{-3}$  can be accepted.

In summary, the (64,57) and (128,120) extended Hamming codes appear attractive for the VLBA application. If the raw error rate is below  $10^{-3}$ , then the decoded bit error rate is below  $10^{-5}$ , and the dismissal percentage is less than 1%. Even if  $p$  becomes five times higher, the performance is perhaps still tolerable. Of course, if the error probability is even better, the decoder performance improves dramatically. The throughput loss due to code redundancy is 11% and 6% respectively, and both are easily implemented. The length 256 code would not seem to be preferable as it produces slightly higher decoded error probability, with only marginal decrease in redundancy. Block diagrams of generic encoders, syndrome formers, and a non-ROM implementation of decoding are attached.

There are more powerful codes (mostly BCH codes) which I will soon report upon. An example is the (128,112) BCH code, capable of correcting single and double-error patterns. However, triple errors, at least some of them, cause decoder failure, and it's not clear that performance is much superior to the extended Hamming codes. Of course, the latter achieve a low  $P[DF]$  partly by dismissing double-error blocks. For these remaining codes to outperform the extended Hamming codes, they will probably need to be longer, and have more complicated decoders. A competitor to the (64,57) code having the same throughput might be a (255,223) quadruple-error correcting BCH code.

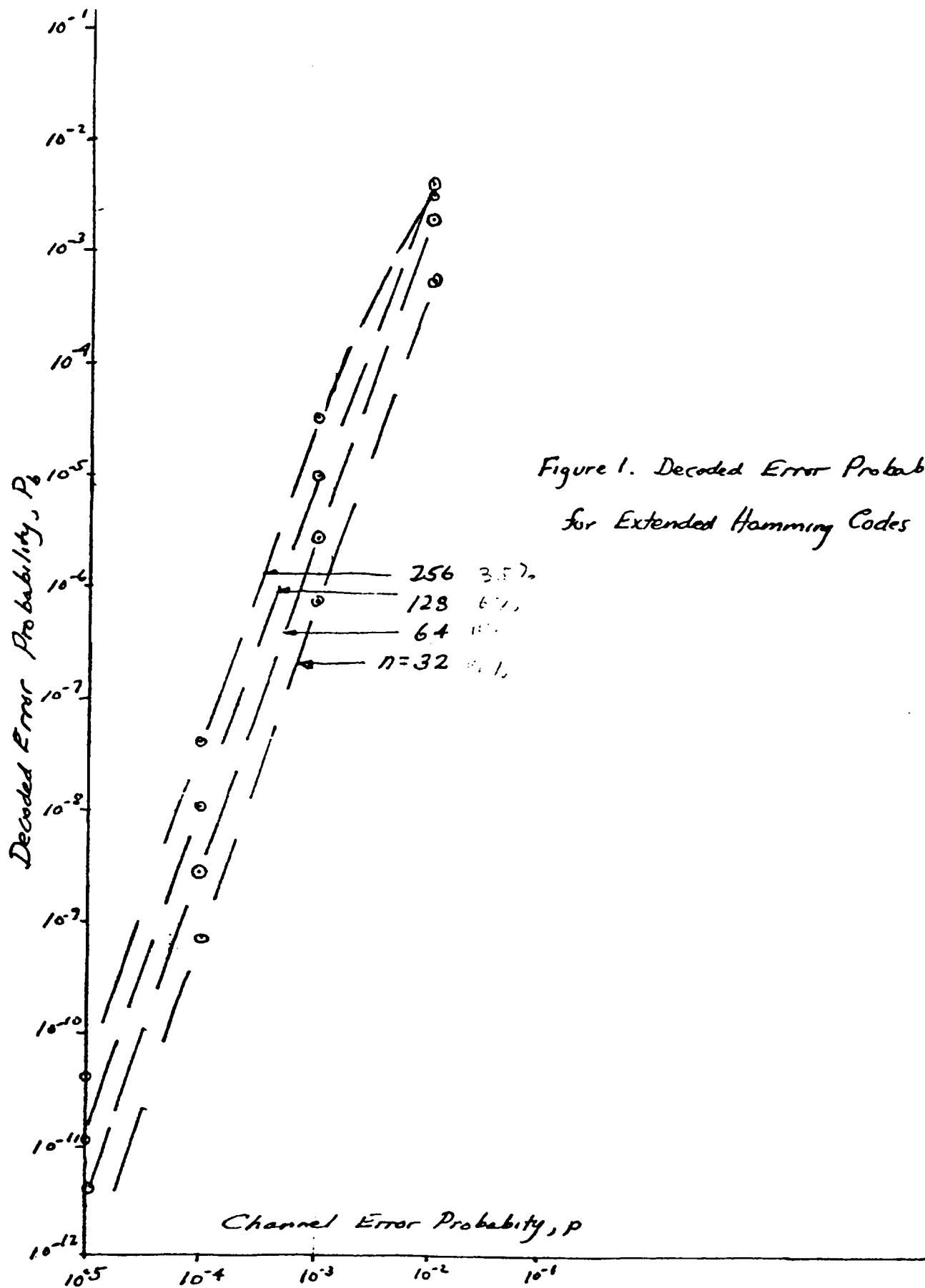


Figure 1. Decoded Error Probability  
for Extended Hamming Codes

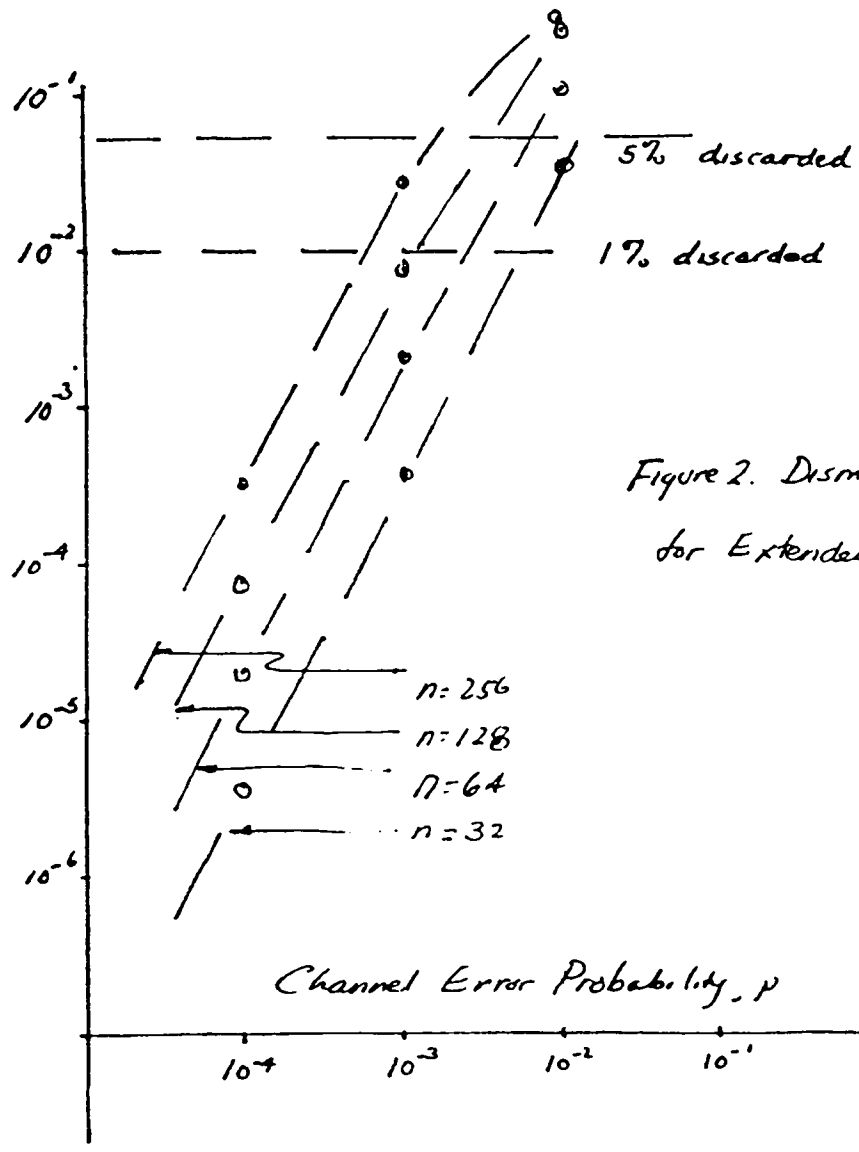


Figure 2. Dismissal Probability  
for Extended Hamming Codes

Error Control Coding

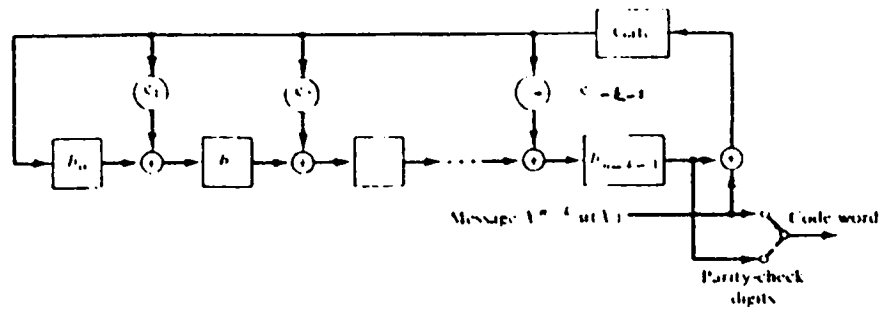


Figure 4.1 Encoding circuit for an  $(n, k)$  cyclic code with generator polynomial  $g(X) = 1 + r_1X + r_2X^2 + \dots + r_{n-k}X^{n-k} + X^n$ .

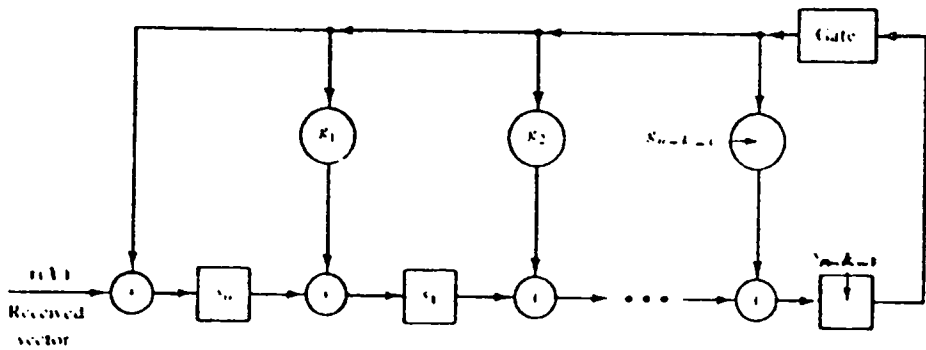


Figure 4.5 An  $(n - k)$ -stage syndrome circuit with input from the left end.

Ref: Lin & Cost

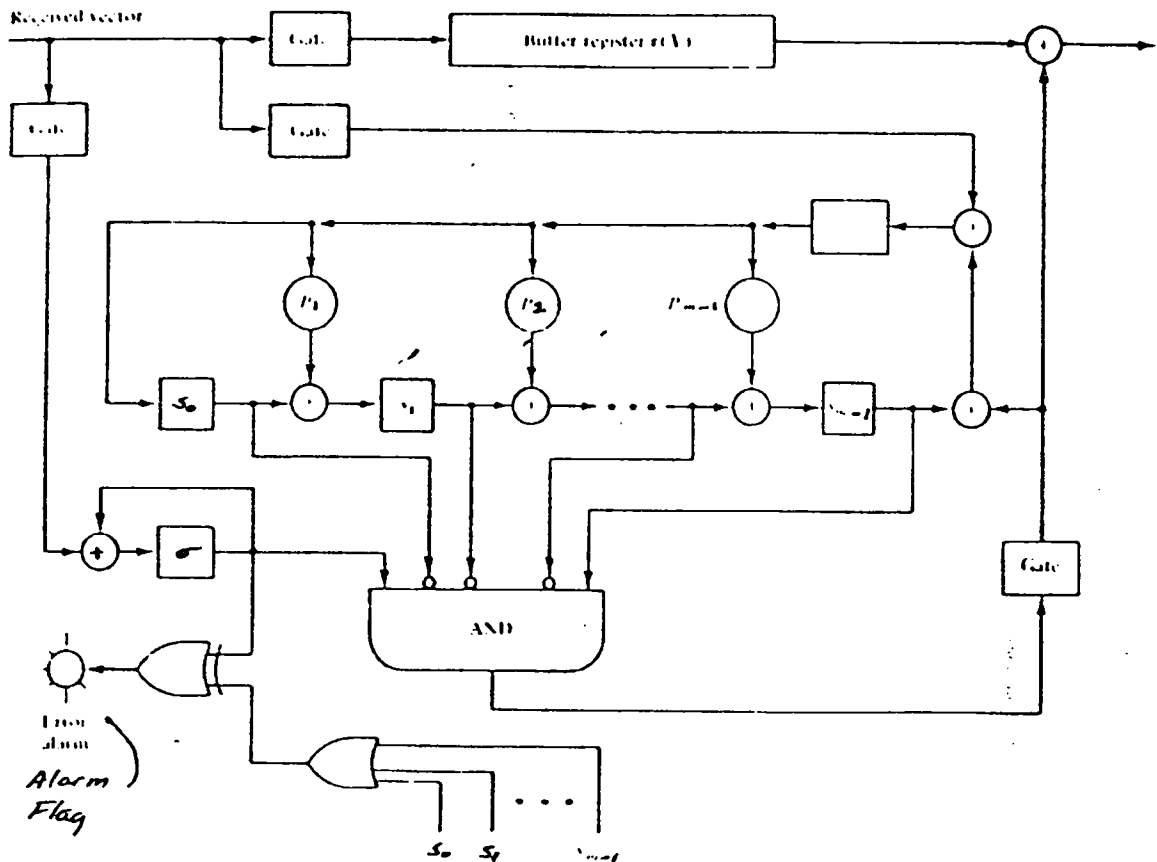


Figure 4.14 Decoding circuit for a single-error-correcting and double error-detecting code. (Assume  $c$ )

April 5, 1985<sup>4</sup>

To: R. Escoffier, L. D'Addario

From: S. G. Wilson

Re: Error Control Coding for VLBA Recording  
Part 2: Multiple Error Correcting/Detecting Codes

In the previous memo I described the SEC-DED Hamming codes which are simple to instrument, and give rather good performance, e.g. a (128,120) code has an overhead of 6%, but the decoded error probability is about  $10^{-5}$  for a raw channel error rate of  $10^{-3}$ , and the block dismissal rate is about 1% for the same condition.

This memo describes several slightly more powerful BCH codes having multiple-error correcting capability. Performance improves somewhat in exchange for more complicated decoders, though I regard them as quite feasible.

#### CODES:

The codes under study are listed below:

Code	Overhead, %	Corrects/Detects
(64,51)	20	2/3
→ (128,112)	12	2/3
(256,239)	6.6	2/3
(256,231)	10	3/4

All codes are capable of being encoded with a feedback shift register having at most 24 bits. Decoding proceeds first by computing the syndrome of the error message with a circuit virtually the same as the encoder. At this point, the decoding must proceed differently for the codes above. For the shorter codes, either table look-up of the error positions or the Meggitt decoding procedure is used. The latter recirculates the syndrome  $n$  times, looking for a pattern corresponding to a correctable error in the right-most buffer position. Implementing this error-recognizer can be performed nicely with gate array technology. The longer codes, especially the last one above, would use an algebraic type of decoding.

#### PERFORMANCE

##### (64,51) CODE:

This code has minimum distance of 6, meaning it can correct up to two errors and detect all triple errors. At least some quadruple error events cause decoding errors, and an upper bound on probability of decoding failure is obtained by assuming all higher-order error patterns lead to decoding error.

$$P[\text{decoding error}] < \binom{64}{4} p^4 (1-p)^{60} + \binom{65}{5} p^5 (1-p)^{59}$$

$$\approx 6.35 \times 10^5 p^4 (1-p)^{60}$$

We also estimate when a decoding error occurs at most five of the 51 information bits are incorrectly output. Thus,

$$P[\text{bit error}] \approx 6.2 \times 10^4 p^4 (1-p)^{60}$$

The probability of a block dismissal can be approximated as the probability of having exactly three errors in 64 bits (actually many 4, 5, etc. error events are detected as well, but for small  $p$  these are not significant).

$$P[\text{block dismissal}] \approx \binom{64}{3} p^3 (1-p)^{61} = 4.2 \times 10^4 p^3 (1-p)^{61}$$

(128, 112) CODE:

This code also is double-error correcting, triple-error detecting. By the same argument as above,

$$P[\text{bit error}] = \left(\frac{5}{112}\right) P[\text{decoding error}]$$

$$= \left(\frac{5}{112}\right) \binom{128}{4} p^4 (1-p)^{124} = 4.8 \times 10^5 p^4 (1-p)^{124}$$

$$\text{and } P[\text{block dismissal}] = \binom{128}{3} p^3 (1-p)^{125} = 3.4 \times 10^5 p^3 (1-p)^{125}$$

(256, 239) CODE:

Again the code is two-error correcting, triple-error detecting, and

$$P[\text{bit error}] = \left(\frac{5}{239}\right) \binom{256}{4} p^4 (1-p)^{252}$$

$$\approx 3.7 \times 10^6 p^4 (1-p)^{252}$$

Also,

$$P[\text{block dismissal}] \approx \binom{256}{3} p^3 (1-p)^{253} = 2.76 \times 10^6 p^3 (1-p)^{253}$$

(256, 231) CODE:

This code is capable of correcting triple errors and detecting quadruple errors since minimum distance between codewords is eight. Approximate performance is given by

$$P[\text{bit error}] = \left(\frac{6}{231}\right) \binom{256}{5} p^5 (1-p)^{251} = 2.3 \times 10^8 p^5 (1-p)^{251}$$



$$\text{and } P[\text{block dismissal}] \approx \binom{256}{4} p^4 (1-p)^{252}$$

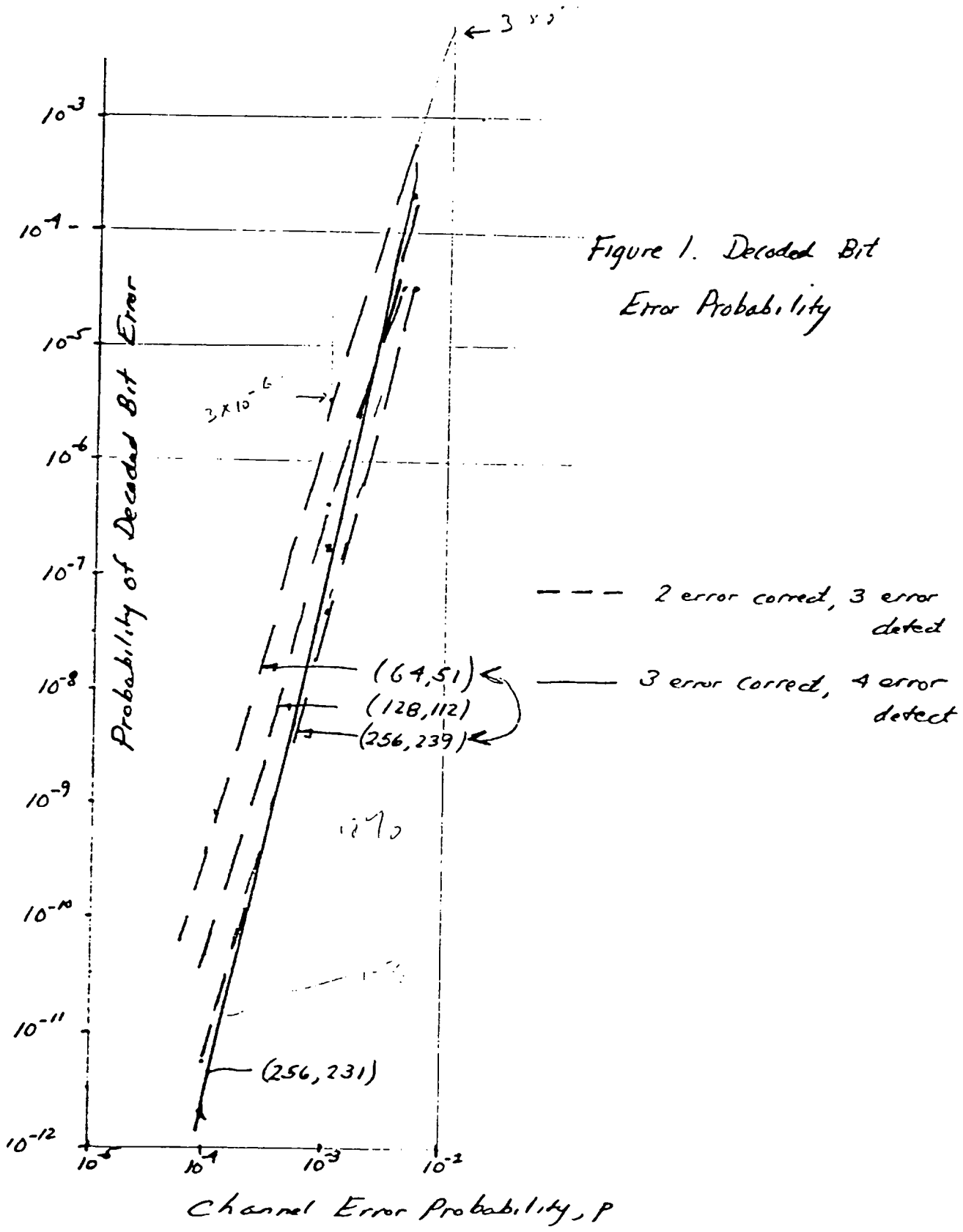
These results are plotted in Figures 1 and 2 (compare the same figures in the previous memo for the SEC-DED codes). From Figure 1 we see that at  $p=10^{-3}$ , all codes give at least two orders of magnitude improvement in decoded error probability. The most fair comparison with earlier results is obtained with codes of roughly equal overhead. A SEC-DED (64,57) code and the (256,231) code both have about 11% overhead. With  $p=10^{-3}$ , the former gives  $P[\text{bit error}] = 3 \times 10^{-6}$  while the latter gives  $1.8 \times 10^{-7}$ . As the channel improves, the longer code improves more rapidly than the SEC-DED code as it has a larger exponent on  $p$ .

Figure 2 shows that dismissal rates are acceptable, i.e. less than 1%, for all codes provided  $p < 10^{-3}$ ;  $p$  being five times worse still leaves reasonable throughput.

My evaluation is that the codes of most interest are the (128,112) code and the (256,239) code. Their decoders are reasonably simple, using either ROM-based or Meggitt decoding. The (64,51) code probably has too large an overhead burden, and the (256,231) code, though performing the best of all studied thus far, would need a relatively complicated algebraic decoder, at least with my current understanding of the problem.

The remaining question is how to weigh the complexity/performance question between these candidates and the SEC-DED codes. My current opinion is that complexity is easily manageable, especially if gate array technology is used, and that the (128,112) code represents a reasonable compromise. It happens that this same code is used in INTELSAT V time-division multiple access at information rates in the vicinity of 60 Mbps.

A final comment on implementation is that independent errors have been assumed throughout; since errors are likely to occur in clusters on a tape system, interleavers are required. For a (128,120) code a block interleaver of 128 by 16 bit size would probably be adequate. The requirement for the interleaver/deinterleaver is another issue mitigating against longer codes. Also, to decode at 10 Mbps rates, a double-buffer strategy will be necessary unless a ROM look-up of error positions can be accomplished in one shift time (100 nanoseconds). The Meggitt decoder would be cycling one syndrome and the received message, while another buffer was being loaded. During the next word, the buffers switch roles.



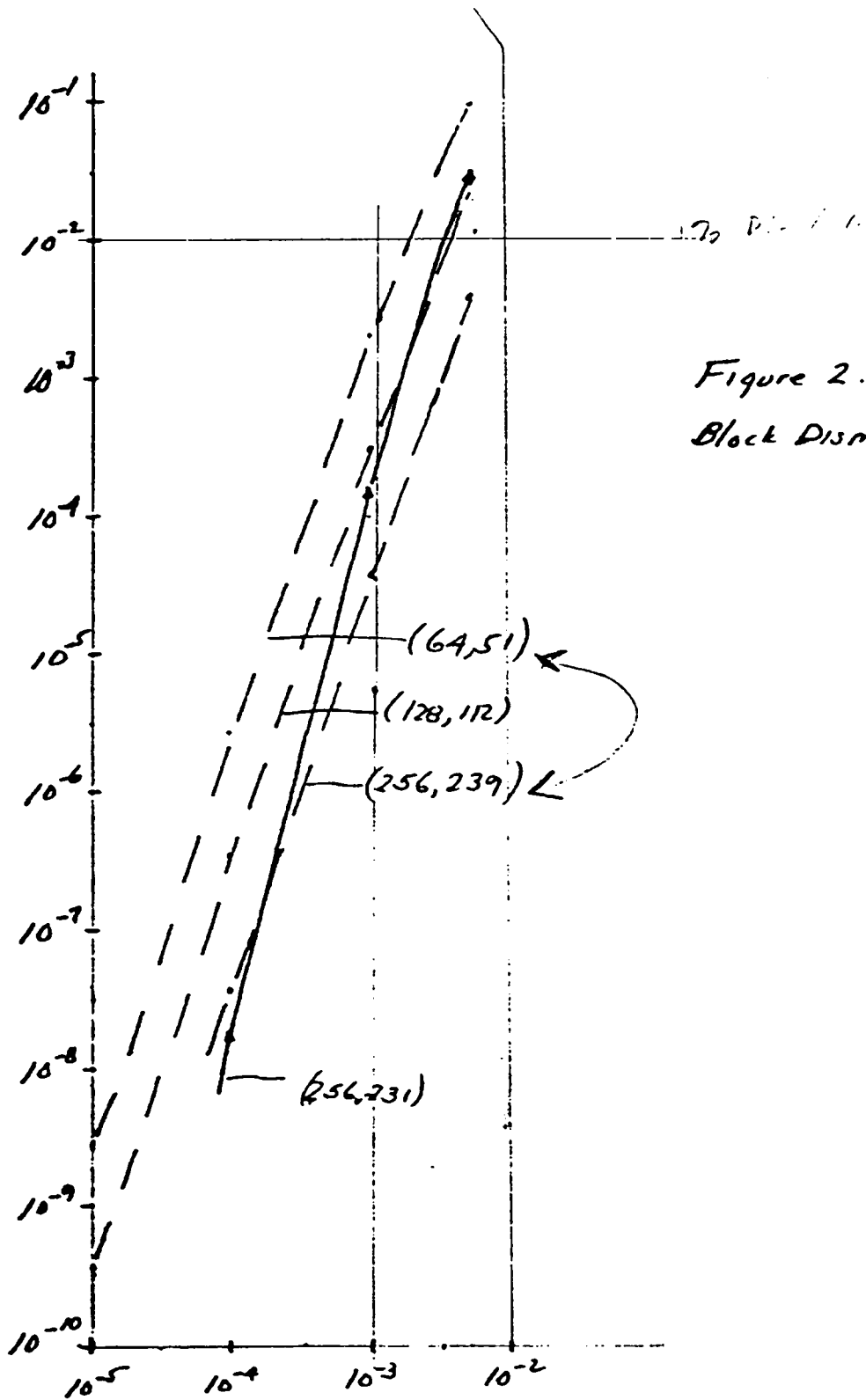


Figure 2.  
 Block Dismissal Probability

# USE OF REED-SOLOMON CODES FOR VLB RECORDING

Stephen G. Wilson

August 20, 1984

## ABSTRACT

A high-speed Reed-Solomon coding scheme is described which is capable of correcting single byte errors and detecting double byte errors in a codeword written across 32 tracks on a recording system, with a 3-track redundancy required. On a random error binary channel with  $p = 10^{-3}$ , the decoded error probability is estimated to be  $2 \times 10^{-5}$ , and the block dismissal probability is .01. The code is capable of correcting indefinitely long error bursts along any one recorder channel.

Coding and decoding algorithms are provided, and a quick implementation study indicates that standard low-power Schottky logic can provide both functions with roughly 125 IC's.

## USE OF REED-SOLOMON CODES FOR VLB RECORDING

### I. INTRODUCTION

This memo describes the use of non-binary Reed-Solomon codes for high-rate recording of data for the VLBI project. The technique is applicable to either multi-channel longitudinal recording, or to the case where many less-expensive, single-channel VCR's operate in parallel.

The application is illustrated in Figure 1, where we visualize up to 32 simultaneous bit streams produced by the sensors, each having a bit rate of 4 Mbps. The "recorder" can be viewed as either a single longitudinal recorder with parallel tracks, or a number of single-channel VCR's. In either case, the error control coding is done "across-tape", so that this decoder is capable of correcting errors due to single-channel electronics failure. Furthermore, the coding is done in terms of symbols from a non-binary alphabet. Although binary codes can be employed across tape as suggested, the overhead, or redundancy, required to correct one error out of, say, 32 tracks is higher than for non-binary codes. Figure 2 illustrates the manner in which code symbols are formed and written on tape, or the effective "tape" of several parallel VCR's. Note  $b$  bits represent a symbol in this coding scheme, and the code is defined on the field  $GF(2^b)$ .

In the following discussion, code design considerations will be summarized, tentative recommendations made, a performance analysis given, and implementation requirements sketched.

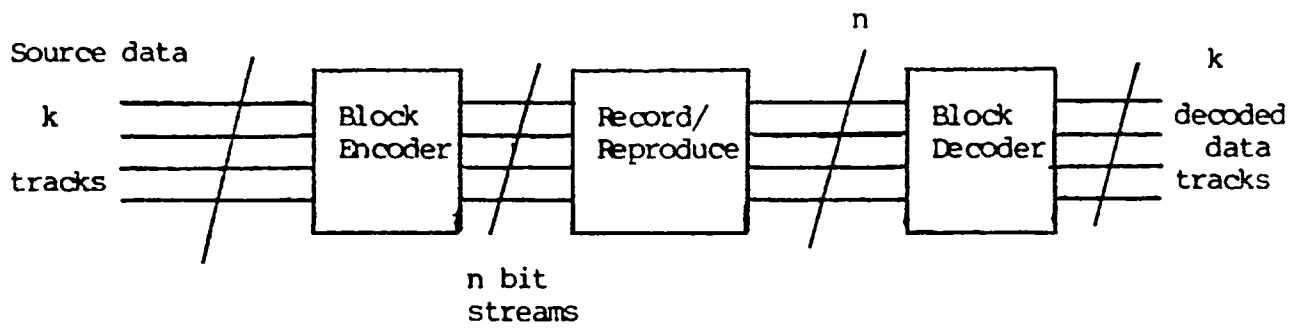


Figure 1: General Configuration

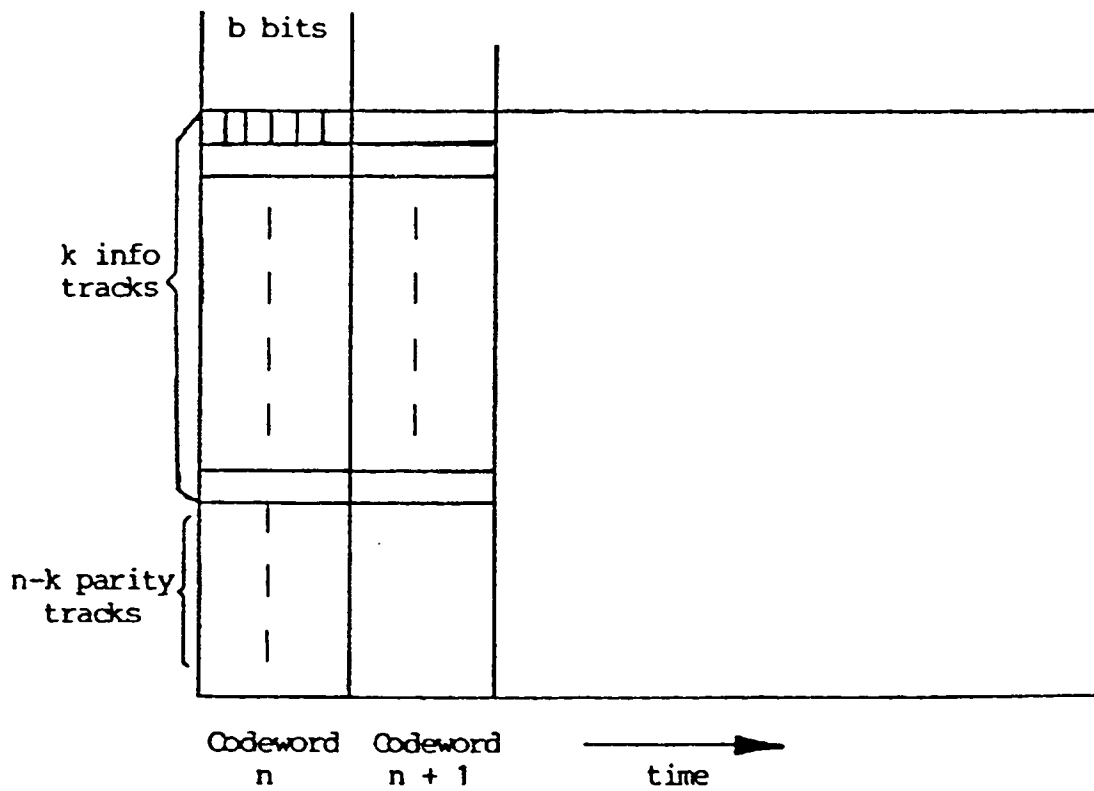


Figure 2: Arrangement of Coded Data

## II. CODE SELECTION

We are interested in non-binary codes of high rate, or low redundancy, capable of correcting a single symbol (byte) error, and perhaps detecting and flagging double errors as well. We refer to these properties as SBEC or SBEC/DBED respectively. (A byte here is  $b$  bits, not the more usual 8). Choice of the byte size does not affect the encoder complexity substantially, as long as it's reasonably small, i.e.  $< 8$ . However, the decoder complexity is strongly influenced by byte size; in particular the size of the table used to locate error positions is  $2^{2b}$  words. Again we're motivated to keep  $b$  small. On the other hand, the byte-size is related to the codeword length for Reed-Solomon codes in that

$$n < 2^b$$

(There are other codes which avoid this restriction, but much less is known about them). Keeping in mind a block length of about 32, we find the byte size ought to be at least  $b = 5$ .

The natural Reed-Solomon codes have

$$n = 2^b - 1$$

$$n - k = d_{\min} - 1$$

where  $d_{\min}$  is the minimum distance of the code. For  $b=5$ , the natural codes are (31, 29) which is SBEC and (31, 28) which is SBEC/DBED.

It may be more convenient to have a block length of 32, which can be obtained by lengthening the Reed-Solomon codes to (32, 30) or (32, 29) respectively. This may be done without changing  $d_{\min}$ , and thus the error correcting capability.

Another possibility is that slightly longer codes are convenient, say to accommodate 32 information tracks. The construction then would use  $b = 6$ , giving  $n = 63$  as the natural length. These codes may be shortened to  $(34, 32)$  or  $(35, 32)$ , depending on whether SBEC or SBEC/DBED is desired.

The redundancy, or overhead, of these codes is about 6.6% and 9.4% for SBEC and SBEC/DBED respectively. The redundancy penalty is felt in this application as a need for extra tracks, not a lowering of throughput per unit time.

For the subsequent discussion, I shall consider only the  $(32, 30)$  SBEC code and the  $(32, 29)$  SBEC/DBEC code. Accommodating the alternatives above in either analysis or design is a minor change on the baseline case.

### III. PERFORMANCE ANALYSIS

#### $(32, 30)$ Code over GF $(32)$

The lengthened code has  $d_{\min} = 3$ . There are  $32 \cdot 31$  single-error patterns and one zero-error pattern, totalling 993 patterns. The number of syndromes is  $32^2$  since there are 2 syndrome symbols; thus the number of syndromes only slightly exceeds the required number of correctable patterns. This says that very few double-errors could be corrected (or detected as well) and a good approximation is to assume all double and higher error patterns cause decoding error.

The symbol error probability  $P_s$  is approximately  $bp = 5p$  assuming  $p$  reasonably small. The probability the decoder commits a decoding error is:



$$\begin{aligned}
P_{DE} &= \sum_{j=2}^{32} \binom{32}{j} P_S^j (1-P_S)^{32-j} \cong \binom{32}{2} P_S^2 (1-P_S)^{30} \\
&= 496 (25) p^2 (1-5p)^{30} \\
&\cong 12400 p^2 (1-150p)
\end{aligned}$$

For  $p = 10^{-3}$ , a baseline choice,  $P_{DE} = .0105$ . Thus, the block error rate is rather large. This is not too surprising given the effective binary block-length of  $32 \cdot 5 = 160$  and the channel error rate of  $10^{-3}$ . The probability of two bit errors in 160 is about .01, and with high probability, these are located in two distinct bytes.

The decoded bit error probability,  $P_b$ , is estimated by assuming a most-likely error event of two byte errors, each having a single bit error. The decoder thinks a single byte error somewhere else occurs, and this decoded byte is basically garbage. This expected number of bit error per block error is then  $(b/2) + 1 + 1$ , and since there are  $30 \cdot b$  bits released per block.

$$P_b = \frac{(b/2 + 2)}{30 \cdot b} [12400 p^2 (1-150p)]$$

Again taking  $p = 10^{-3}$ ,

$$P_b \cong 3 \times 10^{-4}$$

If  $p$  improves to  $10^{-5}$ , then  $P_b \cong 3.5 \times 10^{-8}$ . Thus as the raw channel improves, the end-to-end performance improves twice-as-fast in order of magnitude.

(32, 29) Code over GF (32)

This code has  $d_{\min} = 4$  and is SBEC/DBED. We can reduce the decoded error rate substantially if we're willing to dismiss a small fraction of the data.

The number of distinct syndromes is now  $32^3 = 32768$ . We note this is plenty to cover the zero and single error cases. There are  $\binom{32}{2} \cdot (31)^2$  distinct double errors, which is 476,656. We can of course, not correct all double-errors since we run out of distinct syndromes, but it happens that these double-error patterns are all confined to cosets other than those of the zero or single-error patterns. These syndromes can be easily recognized as described later, and the blocks flagged. Hence it requires at least three byte errors to force a decoder error. It is likely that many (probably most) of the three error (and higher) error patterns are flagged; however as I do not yet have a handle on the actual number, I make the pessimistic assumption that all 3-error patterns cause a decoder error. Thus,

$$\begin{aligned} P_{DE} &= \sum_{j=3}^{32} \binom{32}{j} P_S^j (1-P_S)^{32-j} \cong \binom{32}{3} P_S^3 (1-P_S)^{29} \\ &= 4960 (125 p^3) (1-145p) \\ &= 6.2 \times 10^5 p^3 (1-145p) \end{aligned}$$

For  $p = 10^{-3}$ ,  $P_{DE} = 5.3 \times 10^{-4}$ , much lower than before. The decoded error probability is estimated by assuming three symbol errors, each with a single bit error, cause a fourth symbol error elsewhere in the block to be released.

$$P_D = \frac{(b/2 + 3)}{29 \cdot b} [6.2 \times 10^5 p^3 (1-145p)]$$

With  $p = 10^{-3}$ ,  $P_D = 2.0 \times 10^{-5}$ . Also for  $p = 10^{-5}$ ,  $P_D = 2.1 \times 10^{-11}$ .

With this code, we dismiss code blocks as bad whenever we think a double-byte error occurs. The probability this happens is

$$P \text{ [dismissal]} \cong \sum_2^{32} P_S^2 (1-P_S)^{30}$$
$$\cong 12400p^2 (1-150p)$$

When  $p = 10^{-3}$ ,  $P \text{ [dismissal]} \cong .01$ , or 1% of the data is discarded. With the SBEC code, it was these cases which caused erroneous data to be released.

Comments:

1. The performance analysis for the SBEC/DBED code is the least accurate, but is pessimistic in any case.
2. The channel is likely to have a bursty nature. This can be caused by electronics problems associated with a single track. These types of errors are corrected by the across-tape coding patterns here. Other types of bursty error phenomena, such as oxide imperfections are probably spatially isotropic. Bursts in the direction "across tape" are not well handled by the schemes here. A big interleaving arrangement could help, but at the speeds contemplated here, this is probably prohibitive. A "helical" patterning on the tape can ensure no codeword occupies a position on more than one channel, and no more than one column along the tape.
3. Another decoding alternative is "errors-and-erasures" decoding. If it is known that a certain code position is untrustworthy, it's reasonable that a smart decoder can do better than if it didn't use this information.

This information might be available from observation of electronics over a period of time, or the decoder can "learn" the bad tracks itself by counting errors per track. The (32, 29) code, which without side information is able to correct one error and detect two, can correct one erasure and another randomly placed error.

The decoding for "errors and erasures" is only slightly more difficult than for errors-only; the bigger question is whether this information can be obtained easily and reliably during decoding.

#### IV. CODING AND DECODING FOR (31, 28) SBEC/DBED CODE

Let  $\bar{u} = (u_{k-1}, \dots, u_0)$  denote the information k-tuple, with  $u_i \in GF(2^b)$ . The code will be systematic with the k highest-order (or left-most) positions having these symbols. Letting  $\bar{c} = (c_{n-1}, c_{n-2}, \dots, c_0)$ , we have  $c_{n-1} = u_{k-1}, \dots, c_3 = u_0$ . The remaining three parity symbols are formed as described next.

Let  $u(x)$  be the polynomial representation of the information vector. We define

$$c_2 = \text{rem} \frac{x u(x)}{x + 1}$$

which is a constant contained in  $GF(32)$ . Similarly,

$$c_1 = \text{rem} \frac{x u(x)}{x + \alpha}$$

$$c_0 = \text{rem} \frac{u u(x)}{x + \alpha^{-1}}$$

where  $\alpha$  is a primitive element in  $GF(32)$  and  $\alpha^{-1}$  is its multiplicative inverse. We note the polynomials

$$v_2(x) = x u(x) + c_2, \quad v_1(x) = x u(x) + c_1 \text{ and } v_0(x) = x u(x) + c_0$$

are all of degree at most  $k$  and have  $(x + 1)$ ,  $(x + \alpha)$ ; and  $(x + \alpha^{-1})$  as factors respectively. Alternatively  $\alpha^0 = 1$ ,  $\alpha^1$ , and  $\alpha^{-1}$  are roots of  $v_2(x)$ ,  $v_1(x)$ , and  $v_0(x)$  respectively. In terms of the code symbols,  $v_2(x) = 0$  implies

$$u_{k-1} + u_{k-2} + \dots + u_0 + c_2 = 0$$

which says  $c_2$  is a symbol chosen equal to the sum of the information symbols.

Likewise  $v_1(\alpha) = 0$  implies

$$\alpha (u_{k-1} \alpha^{k-1} + u_{k-2} \alpha^{k-2} + \dots + u_0) + c_1 = 0$$

or

$$u_{k-1} \alpha^k + u_{k-2} \alpha^{k-1} + \dots + u_0 \alpha + c_1 = 0$$

Finally,  $v_2(\alpha^{-1}) = 0$  implies

$$\alpha^{-1} (u_{k-1} \alpha^{-k+1} + u_{k-2} \alpha^{-k+2} + \dots + u_1 \alpha^{-1} + u_0) + c_0 = 0$$

These three equations are the parity check equations for the code. The three symbols may be generated by the block diagram circuits in Figure 3, more detail of which is given in the Appendix.

Now suppose we receive a codeword  $\bar{r} = (\hat{u}, \hat{c}_2, \hat{c}_1, \hat{c}_0)$ . We define syndrome symbols in keeping with the parity equations of the code:

Noting that  $\hat{u}_j = u_j + e_j$  and  $\hat{c}_m = c_m + e_m$ , we have

$$\begin{aligned}
 S_2 &= e_{n-1} + e_{n-2} + \dots + e_3 + e_2 \\
 S_1 &= e_{n-1} \alpha^k + e_{n-2} \alpha^{k-1} + \dots + e_3 \alpha + e_1 \\
 S_0 &= e_{n-1} \alpha^{-k} + e_{n-2} \alpha^{-k+1} + \dots + e_3 \alpha^{-1} + e_0
 \end{aligned}$$

These symbols can be computed using circuits nearly identical to those of Figure 3.

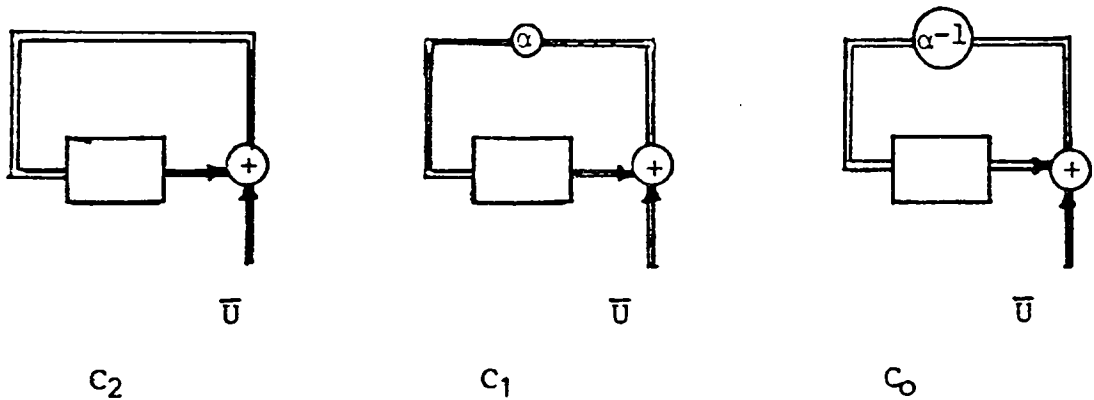





Figure 3: Circuits for Calculating Parity Symbols for SBEC/DBED Code

-  denotes a b-bit storage cell
-  denotes a GF ( $2^b$ ) multiplier
-  denotes a GF ( $2^b$ ) adder  
(b-bit ex-or)

Now consider several cases of error patterns:

Case I: Zero Errors

The syndrome symbols are all zero. If we obtain this result we must say the codeword is valid; it may be that a three-error pattern fools the decoder.

Case II: Single Error of Value  $\beta$  in  $j^{\text{th}}$  position,  $n-1 < j < 3$ .

This is the case of a single error in an information position

$$S_2 = \beta$$

$$S_1 = \beta \alpha^{j-3}$$

$$S_0 = \beta \alpha^{-j+3}$$

Thus  $S_2$  is the value of the error and  $S_2/S_1 = \alpha^{3-j}$ . This equation may be solved for  $j$ , the error position. Solution of the equation may be by trial-and-error (test all 28 values for  $j$ ) or by table look-up.

Case III: Single error of Value  $\beta$  in Position  $j$ ,  $2 > j > 0$ .

$S_j$  will be non-zero, and the other two syndromes will be zero. When this event is detected, we release the information, as it is deemed error free.

Case IV: Double-Byte Error

IVa: Error  $\beta_0$  and  $\beta_1$  in positions 0 and 1

$$S_2 = 0$$

$$S_1 = \beta_1$$

$$S_0 = \beta_0$$

Case IV: Double-Byte Error (Continued)

IVb: Errors  $\beta_0$  and  $\beta_2$  in positions 0 and 2

$$S_2 = \beta_2$$

$$S_1 = 0$$

$$S_0 = \beta_0$$

IVc: Error  $\beta_2$  and  $\beta_1$  in positions 2 and 1

$$S_2 = \beta_2$$

$$S_1 = \beta_1$$

$$S_0 = 0$$

All of these cases give syndrome patterns differing from the single error or zero error syndromes and hence can be detected.

IVd: Errors of type  $\beta$  in positions  $j, k$   $31 > j, k > 3$ , i.e., both errors of same type (an improbable event).

$$S_2 = \beta + \beta = 0$$

$$S_1 = \beta (\alpha^{j-3} + \alpha^{k-3}) \neq 0$$

$$S_0 = \beta (\alpha^{-j+3} + \alpha^{-k+3}) \neq 0$$

Again this is distinct from the pattern of single error or zero-error syndromes and is detectable.

IVe: Errors of type  $\beta$ , one in information set, one in parity  $c_2$ .

$$S_2 = \beta + \beta = 0$$

$$S_1 = \beta (\alpha^{j-3}) \neq 0$$

$$S_0 = \beta (\alpha^{-j+3}) \neq 0$$



Case IV: Double-Byte Error (Continued)

IVf: Error of type  $\beta$ , one in information set, one in parity  $c_1$

$$S_2 = \beta$$

$$S_1 = \beta (\alpha^{j-3}) + \beta = \beta (\alpha^{j-3} + 1)$$

$$S_0 = \beta (\alpha^{-j+3})$$

IVg: Error of type  $\beta$ , one in information set, one in parity  $c_0$

$$S_2 = \beta$$

$$S_1 = \beta \alpha^{j-3}$$

$$S_0 = \beta \alpha^{-j+3} + \beta$$

IVh: Two errors of value  $\beta_j$  and  $\beta_k$  in positions  $j, k$

$$S_2 = \beta_j + \beta_k \neq 0$$

$$S_1 = \beta_j \alpha^{j-3} + \beta_k \alpha^{k-3}$$

$$S_0 = \beta_j \alpha^{-j+3} + \beta_k \alpha^{-k+3}$$

Cases IVf, IVg, and IVh have the apparent potential of producing three non-zero syndromes, which would at first glance correspond to a single error (Case II). However, the syndromes in Case II have a unique relationship; namely

$$S_1/S_2 = \alpha^{j-3} \text{ and } S_0/S_2 = \alpha^{3-j} \quad \text{for the same } j$$

It may be shown that Cases IVf, IVg and IVh produce syndromes for which this is not true, and these events are hence detectable as neither zero-error nor single-error.

The decoding rules are summarized in the following table:

$S_0$	$S_1$	$S_2$	<u>Action</u>
0	0	0	accept information
NZ	NZ	NZ	test if $S_1/S_2$ and $S_0/S_2$ give same $j$ ; if so, have single error of value $S_2$ in $j^{\text{th}}$ position; if not dismiss block
0	NZ	NZ	dismiss
NZ	0	NZ	dismiss
NZ	NZ	0	dismiss
0	0	NZ	accept (single parity error)
0	NZ	0	accept (single parity error)
NZ	0	0	accept (single parity error)

Extension to (32, 29) SBEC/DBED Code:

We can lengthen the code adding an extra information symbol, and keeping three parity symbols. The codeword is again systematic so that  $c_{32} = u_{29}$ ,  $c_{31} = u_{28} \dots c_3 = u_0$ . The symbol  $c_2$  is again chosen so that

$$\sum_{i=2}^{32} c_i = 0$$

i.e. it forms an overall check on the symbols;  $c_1$  is such that

$$\alpha \sum_{i=3}^{32} c_i \alpha^{i-3} + c_1 = 0$$

and  $c_0$  is such that

$$\alpha^{-1} \sum_{i=3}^{32} c_i \alpha^{3-i} + c_0 = 0$$

If no errors occur in 32 positions, all syndromes will be zero. If a single error occurs in an information position,  $S_2$  is the error value, and  $S_1/S_2 = \alpha^{j-3}$  and  $S_0/S_2 = \alpha^{3-j}$  for the same  $j$ . Syndrome rules are followed as before for the other cases. The circuits for calculating the parity symbols and syndrome are exactly as before, but are simply clocked one extra time.

(Note: the lengthening can actually be done once more without lessening performance, to say (33, 30), but these values are perhaps not so convenient).

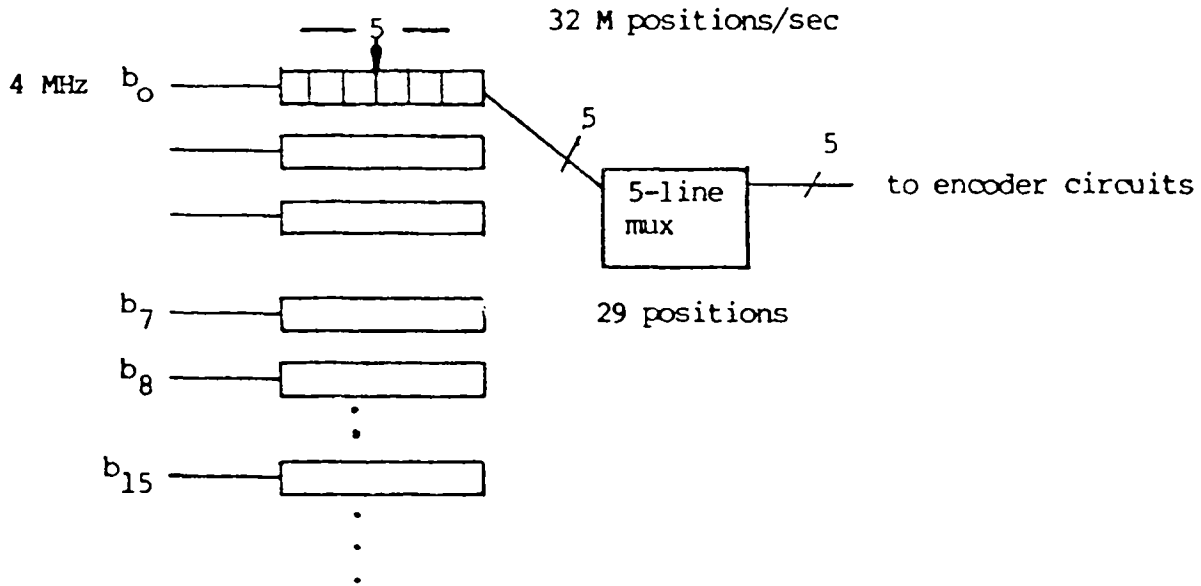
## V. IMPLEMENTATION CONSIDERATIONS

We assume use of the (32, 29) code over GF (32), with 29 parallel tracks being presented at 4 Mbps. Building the encoder/decoder is complicated by the fact that coding is across-tape, whereas the symbols are arriving bit-serial, byte parallel.

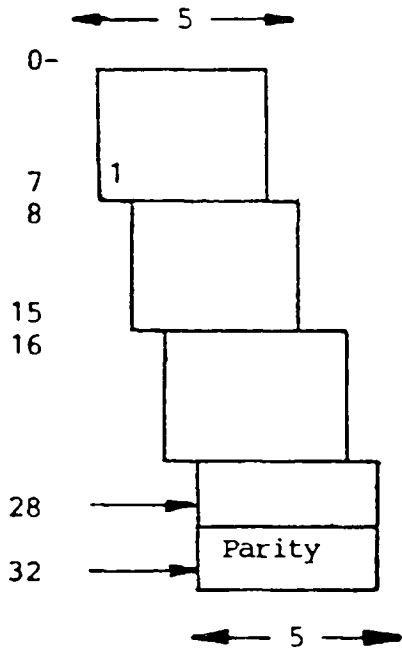
An implementation which has high internal speed requirement is to load 29 5-bit registers in parallel, then have a multiplexer sequence these bytes through the parity calculating circuits shown earlier. The clock rate of this multiplexer and parity circuits must be  $29 (4 \times 10^6) = 116 \text{ MHz (!)}$ , given that we wish to complete all the computation within one shift time. Recording then amounts to clocking out the 5-bit registers along with the three 5-bit parity registers, to the 32-track recorder.

A way to lower the internal logic speed is to allow time-skewing of the codeword, and pipeline the processing. If 4 Mbps is the bit rate per track, then the byte rate is 800 kHz per track. To avoid missing data, we need to sample each track  $8 \times 10^5$  times per second. A convenient method is to adopt

a multiplexer that scans 8 tracks (or symbols) every bit time,  $1 / (4 \times 10^6)$  second. Then it scans 8 more bytes, etc., until it has processed the required 29 tracks. The multiplexer rate and internal computation rate would be 32 MHz.



The bits that join to form a codeword are boxed in the diagram below:



The codeword would also physically appear with this skew on the tape; if desired, this "one bit per 8 track" skew could be eliminated by adding off-setting delay bits prior to recording.

The multiplexer actually needs to sweep only 29 tracks every  $1 / (8 \times 10^5)$  seconds, or have a stepping rate of 23.2 MHz. If it has a 32 MHz step rate, the clocking is synchronous and there will be a short delay before the next cycle begins. (We need to wait until a new 5 bits have entered the register).

Decoding can utilize the same arrangement, paying attention to deskewing, so that the correct bits are grouped together. At the completion of a multiplexer scan, we have the three syndromes, and within one shift time (250 nsec) we must find the error location, and load a register with the error value, for subsequent mod-2 addition of the error value with the desired track. Error correction will require a mod-2 adder on each track which is selected according to the error locations.

For a projected throughput on the order of 128 Mbps, it appears that low-power Schottky logic is sufficiently fast to perform encoding and decoding. A rough projection of parts count (see below) gives 125 standard IC's. Gate array technology could reduce this some, but a substantial number of IC's are shift register buffer elements.

Estimated Package Count:

Encoder

1. Input Registers and 3 Parity Registers:	32	29 8-bit SR's and 3 8-bit latches
2. Parity Calculation		6 quad ex-ors
3. Multiplexer	(10?)	could use two (5-1 muxes for each) bit of 5 bits
4. Miscellaneous dividers & counters	< 5	
		<hr/>
		50 IC's (MSI)

Decoder

Same as for encoder,	50
plus 2 1K x 5 ROM's, 5-bit ex-or,	+ 25
location register, 8 quad ex-ors	
for error correction, 8 quad two-input	
gates for gate selecting, 5-bit to	
32 line decoder	
	<hr/>
	75 IC's

APPENDIX 1: Arithmetic in GF (32)

We view the elements of GF (32) as binary polynomials of degree 4 or less, and perform field addition by adding polynomial coefficients mod-2, and field multiplication by performing polynomial multiplication modulo  $p(x)$ , a primitive polynomial of degree 5. We take  $p(x) = x^5 + x^2 + 1$ . Doing so gives the field of Table 2, with elements listed as 5-tuples by ascending powers of  $\alpha$ , where  $\alpha$  is primitive,  $\alpha = (01000)$ .

GF(2<sup>5</sup>) generated by  $p(X) = 1 + X^2 + X^5$

0	00000	15	11111
1	10000	16	11011
2	01000	17	11001
3	00100	18	11000
4	00010	19	01100
5	00001	20	00110
6	10100	21	00011
7	01010	22	10101
8	00101	23	11110
9	10110	24	01111
10	01011	25	10011
11	10001	26	11101
12	11100	27	11010
13	01110	28	01101
14	00111	29	10010
		30	01001

Table 2

For example, if we wish to add  $\alpha^2 + \alpha^3$  we have  $(00100) + (00010) = (00110) = \alpha^{20}$ , whereas  $\alpha^2 \cdot \alpha^3 = \alpha^5$ . (Exponents add as usual, reduced mod 31.)

We note any field element  $\beta$  can be expressed as

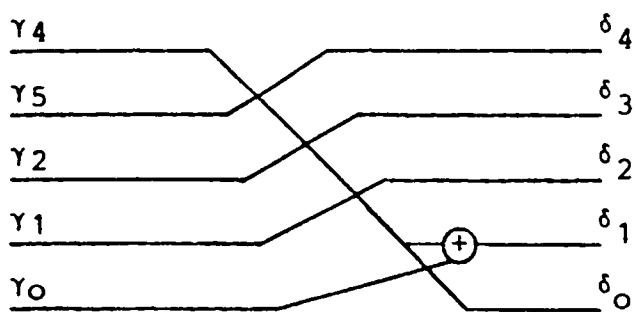
$$\beta = b_0 + b_1 \alpha + b_2 \alpha^2 + b_3 \alpha^3 + b_4 \alpha^4$$

where  $b_i$  are binary coefficients. If we wish to add two elements  $\beta$  and  $\gamma$ , we simply mod-2 add their respective bits, so GF (32) addition is a trivial extension of GF (32) addition.

Suppose we wish to multiply  $\gamma = \gamma_0 + \gamma_1 \alpha + \gamma_2 \alpha^2 + \gamma_3 \alpha^3 + \gamma_4 \alpha^4$  by a fixed value  $\alpha$ , as in the encoder/decoder circuits.

$$\begin{aligned} \alpha\gamma &= \gamma_0 \alpha + \gamma_1 \alpha^2 + \gamma_2 \alpha^3 + \gamma_3 \alpha^4 + \gamma_4 \alpha^5 \\ &= \gamma_0 \alpha + \gamma_1 \alpha^2 + \gamma_2 \alpha^3 + \gamma_3 \alpha^4 + \gamma_4 (\alpha + 1) \\ &= \gamma_4 + (\gamma_0 + \gamma_4) \alpha + \gamma_1 \alpha^2 + \gamma_2 \alpha^3 + \gamma_3 \alpha^4 \end{aligned}$$

This circuit is implemented by:



Now suppose we wish to form

$$\theta = \alpha \gamma + \beta$$

$$\text{Then } \theta_4 = \gamma_3 + \beta_4$$

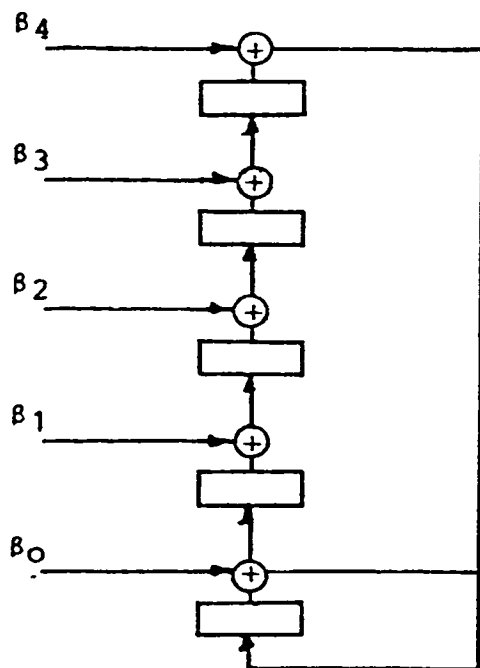
$$\theta_3 = \gamma_2 + \beta_3$$

$$\theta_2 = \gamma_1 + \beta_2$$

$$\theta_1 = \gamma_0 + \gamma_4 + \beta_1$$

$$\theta_0 = \gamma_4 + \beta_0$$

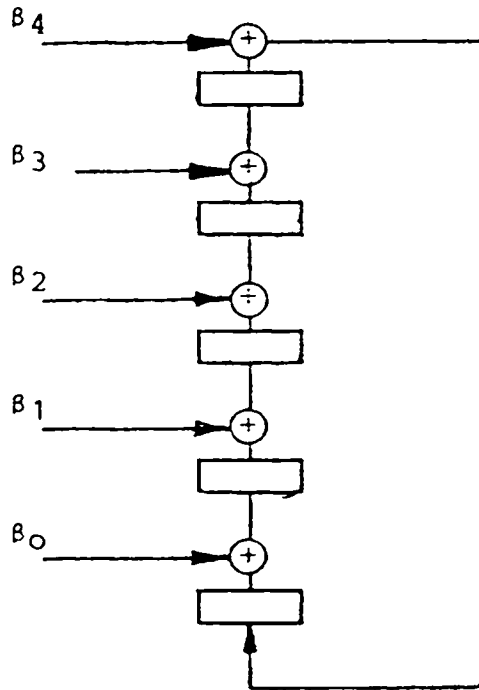
which is implemented as



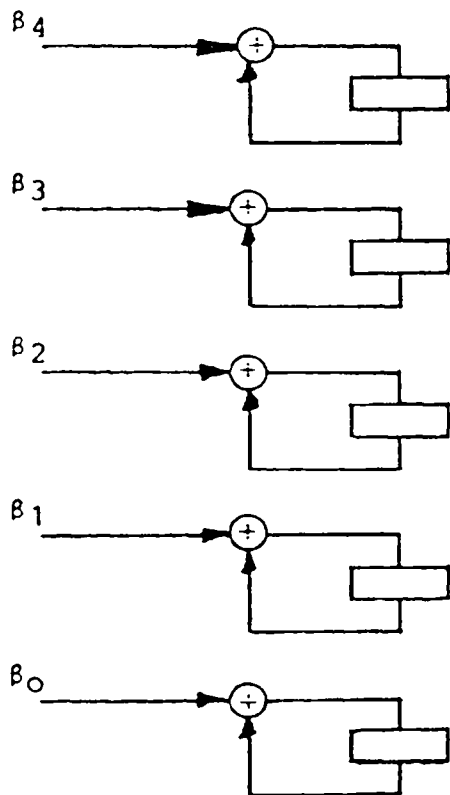
All cells and adders are binary



By similar reasoning, multiplication by  $\alpha^{-1}$  and adding to  $\beta$  is built as



Finally, multiplication by 1 and adding  $\beta$  is



These three circuits provide  $c_1$ ,  $c_0$ , and  $c_2$  respectively when encoding, with  $\bar{\beta}$  representing the information symbols. We zero the register initially and clock each 29 times. After this, the register contents contains the desired symbol.

Likewise for decoding we zero the registers, clock in the 29 received symbols to each, then add with  $\hat{c}_2$ ,  $\hat{c}_1$ , or  $\hat{c}_0$  to obtain  $S_2$ ,  $S_1$ , and  $S_0$  respectively.

## APPENDIX 2: Table-Look-Up From Syndromes

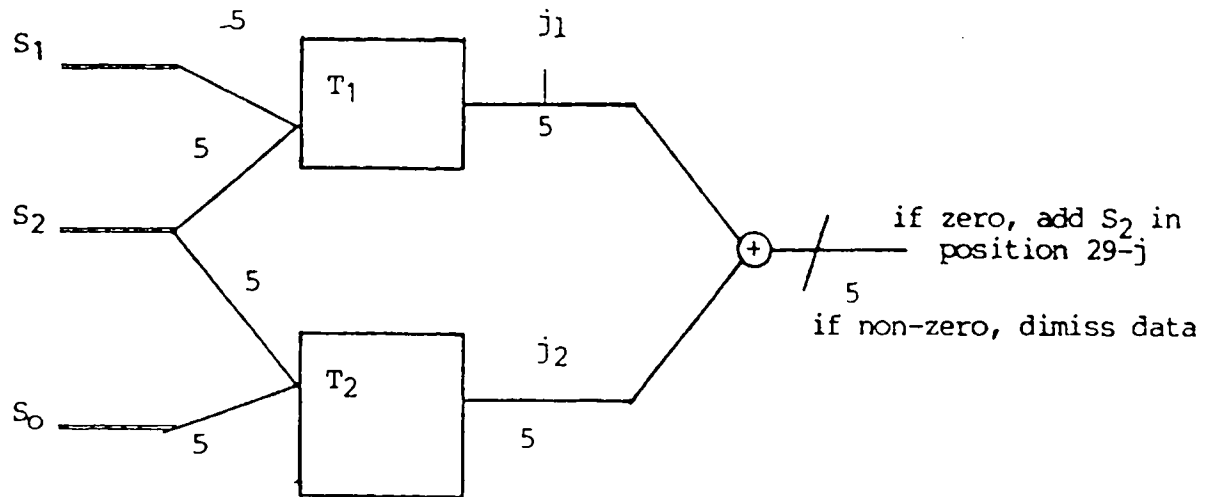
Given  $S_2$ ,  $S_1$ , and  $S_0$  as described earlier, we implement the decoding rule: accept if  $S_2 = S_1 = S_0 = 0$  or  $S_0 = S_1 = 0$ ,  $S_2 \neq 0$ , etc. The Boolean variable A is

$$A = S_2 \cdot \bar{S}_1 \cdot \bar{S}_0 + \bar{S}_2 \cdot \bar{S}_1 \cdot S_0 + \bar{S}_2 \cdot S_1 \cdot \bar{S}_0 + S_2 \cdot \bar{S}_1 \cdot \bar{S}_0$$

If A is true, we take no further action, and release data as is.

If  $S_2$ ,  $S_1$ , and  $S_0$  are all non-zero, we test whether the syndrome corresponds to a correctable single error. We must compute  $S_1/S_2 = \alpha^{j-3}$  and  $S_0/S_2 = \alpha^{3-j}$ , checking to see if the same  $j$  results. This can be done best by having two tables of 1024 words by 5 bits. Table 1 stores the solution to  $S_1/S_2 = \alpha^{j-3}$ , when addressed by  $S_1$  and  $S_2$ , while table 2 stores the solution to  $S_0/S_2 = \alpha^{3-j}$ , when addressed by  $S_0$  and  $S_2$ . If both solutions agree then the  $29-j^{\text{th}}$  symbol of the information patterns should have  $S_2$  added to it. If they don't agree, we dismiss the data, thinking a double-byte error occurred.

Schematically we have the system on the following page:



$T_1$  and  $T_2$  are 1024 words by 5 bits each

A "bad-data" bit is set if exactly one syndrome is zero, or if the above sum is non-zero. In actual operation, we probably would configure the decoder to always release its best guess, i.e. always decode, but to merely have a single flag bit set whenever the data is suspect. The user of the data has the final say in how to utilize this. If we ignore the flag and always decode, the error rate goes up by roughly a factor of 20, but we're not throwing away data.

(Alternate at press-time: We could use a table with  $2^{15} = 32k$  words x 6 bits. The first five bits store the error location if it's to be corrected, and bit 6 would be a flag bit, when the solutions for  $j$  do not equate. This would save ~~some~~ on package count.)