An Asymmetric Encoding Scheme for Word Stuffing

By M. M. BUCHNER, JR.

(Manuscript received November 6, 1969)

The effectiveness of word stuffing for synchronization depends upon our ability to distinguish the stuff words from the data words at the destination and, thus, delete correctly the stuff words. If all input sequences are permitted, the data words must be encoded before stuffing occurs so that the stuff word can be distinct from the data words. In this paper, we give the code that, for a given redundancy, maximizes the minimum distance between the stuff word and any data word. This helps to prevent the loss of character synchronization because of transitions between data and stuff words due to transmission errors. In contrast to the perfect distance symmetries between the code words of the group codes normally encountered in error-control work, the primary virtue of the present code is its highly asymmetric distance structure.

I. INTRODUCTION

When a digital communication network is used for data transmission, it may be necessary to adjust transmission rates within the network to achieve synchronization. Word stuffing[1,2] is a technique that can be used for this purpose. The basic idea is to group the transmitted bits into words which we call data words. The data words are formed for transmission and are not related to any word structure that may exist in the customer's data stream. Stuff words, which are distinguishable from the data words, are inserted into the stream of data words at the transmitter. Thus, transmission rates can be adjusted within the network by inserting or deleting stuff words. At the destination, the stuff words are deleted whereas the data words are delivered to the customer. The effectiveness of word stuffing depends upon our ability to distinguish the stuff words from the data words at the destination and, thus, delete correctly the stuff words. If the stuff words are incorrectly deleted, bits will be inserted into or deleted from the customer's data

stream. As a result, the received bits will be incorrectly formatted and incorrectly interpreted. When this occurs, we say that character synchronization is lost. Character synchronization is important in data transmission because once it is lost, subsequent bits are erroneously interpreted even if transmitted correctly.

Two problems arise. First, we require that all input sequences are allowed. Thus, redundancy must be added to the data words by an encoder so that it is possible to choose a stuff word that is distinct from the data words. Second, when transmission errors occur, it is possible for:

(*i*) a stuff word to be transformed into a data word,
(*ii*) a data word to be transformed into a stuff word, or
(*iii*) a data word to be transformed into another data word.

In most cases, (*i*) and (*ii*) are more serious than (*iii*) because of the resulting loss of character synchronization. The prevention of type (*iii*) errors is generally performed by the customer's terminal, when required, and is not considered in this paper.

Previously, Mattesich and Richters[1] proposed a format for the data words and the stuff word. The format results in the stuff word being distance one* from a data word. Therefore, a single transmission error can change a data word into the stuff word or vice versa with a corresponding loss of character synchronization.

We give an alternative encoding scheme that, for a given redundancy, maximizes the minimum distance between the stuff word and any data word. This helps to prevent the loss of character synchronization because of transitions between data and stuff words due to transmission errors. An implementation is given for arbitrary word size and redundancy. For a redundancy of one bit, a particularly simple encoding-decoding technique is described.

Some other methods of achieving synchronization by means of stuffing have been proposed. A word stuffing technique, proposed by Butman[2], achieves a distance d between the stuff word and any data word by inserting deliberate errors in certain data words at the transmitter to keep the data words at least distance d from the stuff word. Butman's technique requires some knowledge of the statistics of the transmitted data words for selection of the stuff word and a relatively large word size so that the deliberate insertion of errors is infrequent. It results in deliberate errors in the customer's data and prohibits the reception of

---

* The distance, frequently called the Hamming distance, between two binary words X and Y is the number of positions in which X and Y differ.

certain data words. The latter point is troublesome if the data words used in transmission are identical to the data words of the customer.

Pulse stuffing, rather than word stuffing, can be used to achieve synchronization. Individual pulses are inserted at the transmitter and a separate data link is used to signal the locations of the stuff pulses. References 3 through 6 are representative of the work in pulse stuffing.

II. PRELIMINARIES

A model of the processing before transmission is shown in Fig. 1. The input binary data stream is segmented into k-bit data words denoted by A, where

$$A = (a_k , a_{k-1} , \cdots , a_1).$$

We assume that all of the 2^k possible sequences for A are allowed. In order for the stuff word to be distinguishable from the data words, the alphabet is enlarged by adding redundancy to A. Thus, the encoder generates from A an n-tuple B where $n > k$ and

$$B = (b_n , b_{n-1} , \cdots , b_1).$$

The sequence B is transmitted. The dimensions of such a code are denoted by (n, k).

There are 2^n possible sequences for B of which 2^k are used to transmit data. Thus, there are $2^n - 2^k$ values of B that are not used for data but that can be used for other purposes including the stuff word. For $n > k$,

$$2^n - 2^k \geqq 2^k. \tag{1}$$

The reader should appreciate the tremendous flexibility available in the design of the coding scheme because, by (1), never more than half of the possible B sequences are used as data words.

The processing after transmission is indicated in Fig. 2. As shown later, it is possible to construct codes that have a minimum distance greater than one between the stuff word and any data word for practical values of n and k. For these codes, the first step at the destination is to delete the stuff word and all other received words "sufficiently close"

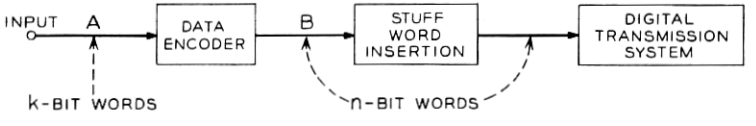

Fig. 1 — Processing before transmission.

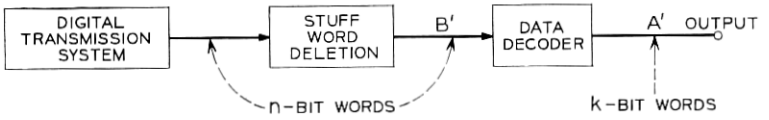

Fig. 2—Processing after transmission.

to the stuff word. In saying "sufficiently close," we mean that the received word is more likely the result of a stuff word corrupted by errors than a data word corrupted by errors. Let B' denote the n-tuples that are not deleted where

$$B' = (b_n' , b_{n-1}' , \cdots , b_1').$$

The decoder operates on B' to form the k-bit word A' where

$$A' = (a_k' , a_{k-1}' , \cdots , a_1').$$

The system functions properly if

$$A' = A.$$

At this point, we describe for reference the encoding and decoding schemes presented in Ref. 1. An (8, 7) code is used, that is, $n = 8$, $k = 7$. The encoder generates B from A by the relation

$$b_8 = 1$$

$$b_i = a_i \quad \text{for} \quad 1 \leqq i \leqq 7.$$

Thus,

$$B = (1, a_7 , a_6 , \cdots , a_1).$$

The stuff word is (00000001). At the receiver, only the word (00000001) is deleted as the stuff word; all other received words are decoded as data words. Thus, the stuff word is interpreted as a data word if one or more transmission errors occur. Alternatively, because the data word (10000001) is distance one from the stuff word, a single error can convert this data word into the stuff word. Any other data word requires at least two errors for conversion into the stuff word.

III. DESCRIPTION OF THE CODE

We construct a code that has one stuff word and 2^k data words such that the minimum distance between the stuff word and any data word is maximized. The problem is to divide the 2^n n-tuples into three sets; namely,

(*i*) the stuff word denoted by S,
(*ii*) the set D consisting of the 2^k data words, and
(*iii*) the set U consisting of the $2^n - 2^k - 1$ unused words,

such that, for given values of n and k, the minimum distance between S and any element of D is maximized.

Choose one of the n-tuples to be the stuff word S. For given n, k and S, let d_m denote the maximum possible minimum distance between S and any element of D. We determine d_m by observing that if all elements of D are to be at least distance d_m from S, then U must contain all n-tuples that are distance $d_m - 1$ or less from S. Thus, the set U contains

$$\begin{aligned} &\text{the } \binom{n}{1} \quad n\text{-tuples distance 1 from } S, \\ &\text{the } \binom{n}{2} \quad n\text{-tuples distance 2 from } S, \\ &\vdots \\ &\text{the } \binom{n}{d_m - 1} \quad n\text{-tuples distance } d_m - 1 \text{ from } S, \end{aligned} \tag{2a}$$

and δ n-tuples distance greater than $d_m - 1$ from S such that

$$\sum_{l=1}^{d_m - 1} \binom{n}{l} \leqq 2^n - 2^k - 1 < \sum_{l=1}^{d_m} \binom{n}{l} \tag{2b}$$

and

$$\delta = 2^n - 2^k - 1 - \sum_{l=1}^{d_m - 1} \binom{n}{l}. \tag{2c}$$

The data words are the remaining 2^k words.

The value of d_m is determined by (2b). From (2b), it follows that d_m is independent of S and is determined entirely by n and k. Also, it is easy to show that changing a code by adding* a constant n-tuple to all words simply rotates the code with no change in the distance properties, including d_m.

The code is specified by (2) up to the choice of the δ unused words that are at distance greater than $d_m - 1$ from S. While the choice of these δ words does not alter d_m, the probability that a data word is transposed into the stuff word is minimized if the δ words are all chosen

* The addition is component-by-component modulo two addition and is denoted by $\oplus$.

to be distance d_m from S. In practice, the choice of the δ words does not change this probability substantially and it appears preferable to assign the δ words to simplify encoder-decoder design.

The case $n = k + 1$ is of interest because of the low redundancy. In Appendix A, it is shown that for $n = k + 1$,

$$d_m = \left[\frac{k}{2}\right] + 1$$

where $[x]$ denotes the largest integer less than or equal to x. Also, for $n = k + 1$ and k even, it is shown in Appendix A that $\delta = 0$. A plot of d_m versus n for $n = k + 1$ and $n = k + 2$ is given in Fig. 3.

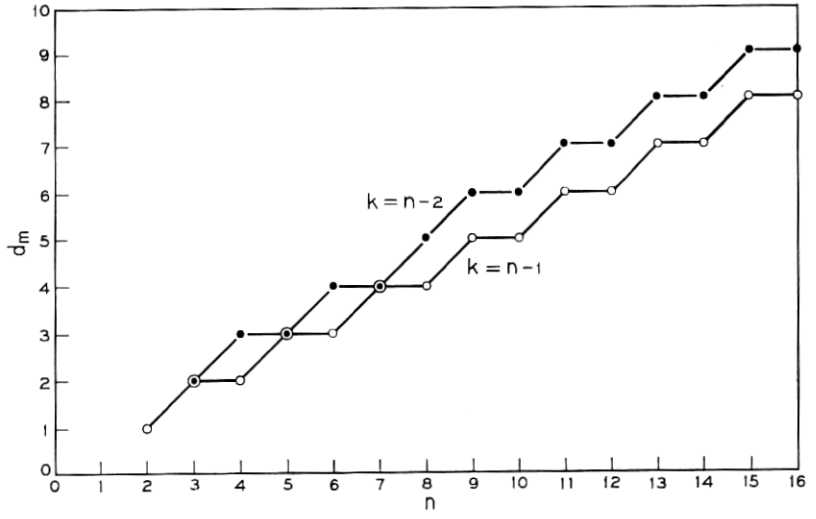

Fig. 3—Maximum minimum distance d_m for various n.

Transmission of the all-zero word can be avoided by choosing S so that the all-zero word is one of the unused words. However, it may be convenient, particularly when detecting stuff words, for S to be the all-zero word because then distance from S is equivalent to weight* and can be computed by counting ones. These two objectives can be simultaneously satisfied as follows. Design the encoder and stuff word using a code C in which S is the all-zero word. Let S' be one of the unused words in C. Add S' to each word immediately before transmission and

* The weight of a binary word X is the number of nonzero components in X and is denoted by $w(X)$.

add S' to each word immediately after transmission. The double addition of S' permits the suppression of the all-zero word for transmission but is transparent for the encoding, stuff word detection, and decoding operations.

Because the data words and the stuff word form a subset of the set of all possible n-tuples, we may, on the average, transmit an unequal number of zeros and ones. By varying $w(S)$ from 0 to n, the relative number of zeros and ones in the data words can be varied from mostly ones to mostly zeros.

IV. ENCODING, DECODING AND STUFF WORD DETECTION: GENERAL CASE

In this section, we specify an encoder that achieves d_m for arbitrary n and k. Let the stuff word be the all-one n-tuple. A necessary and sufficient condition for the encoder to achieve d_m is that each A must be encoded into a unique B such that the maximum weight of any B is $n - d_m$.

We begin by regarding each A as the k-bit natural binary representation of some integer α, $0 \leqq \alpha \leqq 2^k - 1$. Thus,

$$A = B_k(\alpha)$$

where

$$\alpha = \sum_{i=1}^{k} 2^{i-1} a_i \ .$$

We can construct a table that, for each α, gives the corresponding A and B sequences. The entries are in the order of increasing α. For illustration, consider the (9, 7) code where $d_m = 6$. The first 24 entries for the (9, 7) code are shown in Table 1. The A column corresponds to counting in binary from the all-zero k-tuple to the all-one k-tuple. The B column is also formed by counting in binary except that all n-tuples with weight greater than $n - d_m$ are omitted from the count. It follows that the maximum weight of any B is $n - d_m$. The arrows in Table 1 indicate where 9-tuples with weight greater than three have been omitted in the B column.

Let us examine the counting in the B column of Table 1 in greater detail. Consider counting to the B sequence for $\alpha = 22$, that is,

$$(000011000). \tag{3}$$

position 5 — position 4

First, count to (000010000). At this point, all 4-tuples of weight not

TABLE I—RELATIONSHIP BETWEEN A AND B FOR (9, 7) CODE; ONLY THE FIRST 24 VALUES OF A ARE SHOWN.

α	$A = B_7(\alpha)$	B
0	0000000	000000000
1	0000001	000000001
2	0000010	000000010
3	0000011	000000011
4	0000100	000000100
5	0000101	000000101
6	0000110	000000110
7	0000111	000000111
8	0001000	000001000
9	0001001	000001001
10	0001010	000001010
11	0001011	000001011
12	0001100	000001100
13	0001101	000001101
14	0001110	000001110 ←
15	0001111	000010000
16	0010000	000010001
17	0010001	000010010
18	0010010	000010011
19	0010011	000010100
20	0010100	000010101
21	0010101	000010110 ←
22	0010110	000011000
23	0010111	000011001

greater than three have been used in positions one through four. Then, count to the sequence in (3) by using all 3-tuples of weight not greater than two in positions one through three while keeping a one in position five. As shown in Fig. 4, the value of α associated with the sequence in (3) has two components. The first component, denoted by $\alpha_1(5)$, is the number of B sequences used in counting to obtain the one in position five. Similarly, the second component, denoted by $\alpha_2(4)$, is the number of B sequences used in counting to obtain the one in position four. Accordingly,

$$\alpha = \alpha_1(5) + \alpha_2(4)$$

where

$$\alpha_1(5) = \sum_{i=0}^{3} \binom{4}{i} = 15,$$

$$\alpha_2(4) = \sum_{i=0}^{2} \binom{3}{i} = 7.$$

The above ideas can be formalized so that, for an arbitrary B (de-

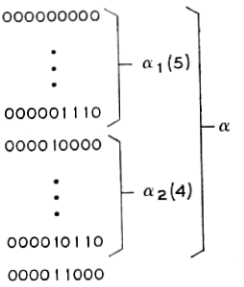

Fig. 4—Counting to determine α.

noted by B_0), it is possible to determine the corresponding value of α (denoted by α_0). Let $\omega = w(B_0)$, $\omega \leqq n - d_m$. Let $\beta_1, \beta_2, \cdots, \beta_\omega$ denote the position numbers of the ω nonzero components of B_0 where

$$\beta_1 > \beta_2 > \cdots > \beta_\omega .$$

For example, if $B_0 = (000011000)$, $\omega = 2$ and $\beta_1 = 5$, $\beta_2 = 4$. Let $\alpha_i(\beta_i)$ denote the contribution of the one in position β_i to α_0, that is,

$$\alpha_0 = \sum_{i=1}^{\omega} \alpha_i(\beta_i).$$

Observe that $\alpha_i(\beta_i)$ is the number of sequences in the B column from the sequence whose nonzero components are in positions $\beta_1, \beta_2, \cdots, \beta_{i-1}$ to the sequence whose nonzero components are in positions $\beta_1, \beta_2, \cdots, \beta_i$. Thus, $\alpha_i(\beta_i)$ is the number of $(\beta_i - 1)$-tuples of weight not greater than $n - d_m - i + 1$ and is given by

$$\alpha_i(\beta_i) = \sum_{m=0}^{\mathrm{Min}(n-d_m-i+1,\beta_i-1)} \binom{\beta_i - 1}{m}. \tag{4}$$

It follows that

$$\alpha_0 = \sum_{i=1}^{\omega} \sum_{m=0}^{\mathrm{Min}(n-d_m-i+1,\beta_i-1)} \binom{\beta_i - 1}{m}.$$

Equation (4) can be used to find B_0 when α_0 is given, that is, to design an encoder.* For each value of i, $1 \leqq i \leqq n - d_m$, construct an array. In the ith array, list j and $\alpha_i(j)$ as j runs from one to $n - i + 1$. To encode α_0, find in the first array the largest j (denoted by j_1) such that

$$\alpha_1(j_1) \leqq \alpha_0 .$$

* The ideas in this paragraph are illustrated by a numerical example in Appendix B.

Next, find in the second array the largest j (denoted by j_2) such that

$$\alpha_2(j_2) \leqq \alpha_0 - \alpha_1(j_1).$$

In the ith array, find the largest j such that

$$\alpha_i(j_i) \leqq \alpha_0 - \sum_{m=1}^{i-1} \alpha_m(j_m).$$

The process continues until

$$\sum_{m=1}^{\omega'} \alpha_m(j_m) = \alpha_0$$

for some $\omega' \leqq n - d_m$. It follows that B_0 has ones in positions $j_1, j_2, \cdots, j_{\omega'}$ and zeros in all remaining positions. Also, $\omega' = \omega$, the weight of B_0 .

It is possible to obtain a recurrence relation for the elements of the arrays. It is shown in Appendix C that

$$\alpha_i(j) + \alpha_{i+1}(j) = \alpha_i(j+1) \tag{5}$$

for $1 \leqq j \leqq n - i$, $1 \leqq i \leqq n - d_m - 1$. By knowing that $\alpha_i(1) = 1$ for $1 \leqq i \leqq n - d_m$ and that $\alpha_{n-d_m}(j) = j$ for $1 \leqq j \leqq d_m + 1$, equation (5) can be used to generate elements of the arrays.

Also, equation (5) can be used to construct an encoder directly. First, we specify the subtraction and storage device shown in Fig. 5. Let R denote the integer stored in the device. The output is equal to the integer stored in the device, that is, R. Let the input be an integer R', $0 \leqq R' \leqq R$. When the device is activated, the number stored in the device becomes $R - R'$ and, thus, the output also takes the value $R - R'$.

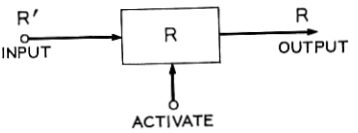

Fig. 5—Subtraction and storage device.

The encoder operates as shown in Fig. 6. The storage devices are preset so that $R_i = \alpha_i\,(n - i + 1)$, $1 \leqq i \leqq n - d_m$, and $T = \alpha$, the integer representation of the A sequence to be encoded.* Position the

* For the (9, 7) code, the preset values are $R_1 = 93$, $R_2 = 29$ and $R_3 = 7$.

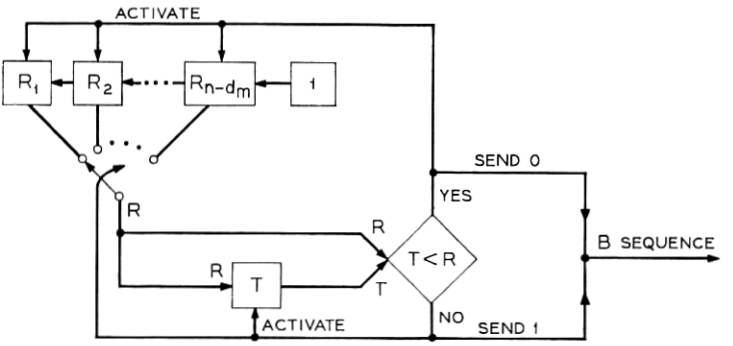

Fig. 6—Encoder for arbitrary n and k.
Preset Values: $T = \alpha$
$R_1 = \alpha_1(n)$
$R_2 = \alpha_2(n - 1)$
.
.
.
$R_{n-d_m} = \alpha_{n-d_m}(d_m + 1)$

switch so $R = R_1$. If $T < R_1$, the $n - d_m$ storage devices are activated and their contents reduced. Also, a 0 is transmitted, that is, $b_n = 0$. However, if $T \geqq R_1$, a 1 is sent, R_1 is subtracted from T, and the switch is shifted so $R = R_2$. The process continues until the entire B sequence is generated. Notice that the B sequence is generated and, thus, transmitted with b_n first and b_1 last.

The decoder is shown in Fig. 7. The storage devices are preset so that $R_i = \alpha_i\ (n - i + 1)$, $1 \leqq i \leqq n - d_m$, and the accumulator is set equal to zero. Position the switch so $R = R_1$. Prior to the decoder, if the weight of the received n-tuple is greater than $n - d_m$ because errors have occurred, the weight is reduced to $n - d_m$ by arbitrarily converting sufficient ones to zeros. Thus, we assume that $w(B') \leqq n - d_m$ and that B' arrives with b'_n first and b'_1 last. The decoder first considers b'_n. If $b'_n = 0$, the $n - d_m$ storage devices are activated and their contents reduced. If $b'_n = 1$, the accumulator is increased by R_1 and the switch is shifted so $R = R_2$. The process continues until b'_1 has been used. After b'_1 has been used, the accumulator contains the integer representation of A'.

For the stuff word detector, S and all other received words less than a specified distance from S are deleted. Thus, the detector counts the zeros in each word and, if the count is less than the specified distance, deletes the word.

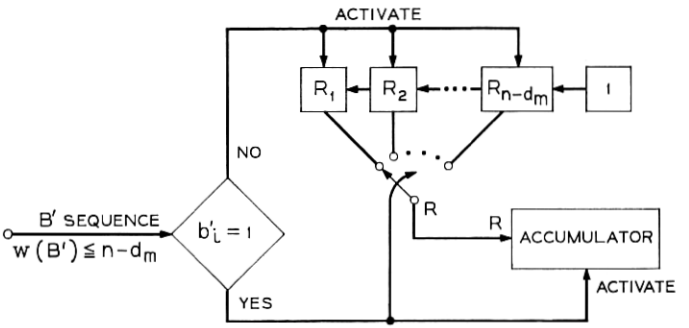

Fig. 7—Decoder for arbitrary n and k.
Preset Values: $R_1 = \alpha_1(n)$
$R_2 = \alpha_2(n-1)$
.
.
.
$R_{n-d_m} = \alpha_{n-d_m}(d_m + 1)$
Accumulator = 0

V. ENCODING, DECODING, AND STUFF WORD DETECTION: $n = k + 1$

In this section, we give a possible implementation for the encoder, decoder, and stuff word detector for $n = k + 1$.

5.1 *Case 1*

Let the stuff word S be the all-zero $(k + 1)$-tuple. The encoder is specified in (6) and shown in Fig. 8. In Fig. 8, the A sequence is assumed to arrive with a_k first and a_1 last. Similarly, b_{k+1} is transmitted first with b_1 last.

$$\text{If} \quad w(A) \geqq \left[\frac{k}{2}\right] + 1, \qquad b_{k+1} = 0.$$

$$\text{If} \quad w(A) \leqq \left[\frac{k}{2}\right], \qquad b_{k+1} = 1. \tag{6}$$

$$\text{Then} \quad b_i = b_{k+1} \oplus a_i \quad \text{for} \quad 1 \leqq i \leqq k.$$

Decoding is also straightforward. The operation necessary for decoding is

$$a'_i = b'_{k+1} \oplus b'_i \quad \text{for} \quad 1 \leqq i \leqq k. \tag{7}$$

Figure 9 shows the decoder in equation (7). In Fig. 9, it is assumed that b'_{k+1} is received first and b'_1 is received last. The stuff word detector is the same as in Section IV except ones are counted instead of zeros.

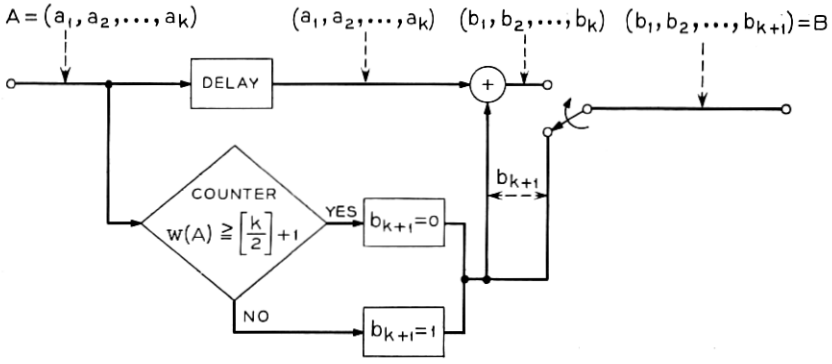

Fig. 8—Encoder for Case 1.

The encoding-decoding technique in (6) results in some error multiplication. Suppose that position i, $1 \leqq i \leqq k$, is in error and that position $k + 1$ is correct. Then, from equations (6) and (7),

$$a_i' = b_{k+1} \oplus 1 \oplus b_i = 1 \oplus a_i .$$

The error in position i is delivered to the customer. However, if position $k + 1$ is in error, from equations (6) and (7),

$$a_i' = 1 \oplus b_{k+1} \oplus b_i = 1 \oplus a_i \quad \text{for} \quad 1 \leqq i \leqq k.$$

The point is that now all k data positions are in error. However, character synchronization is maintained because the correct number of bits are delivered to the destination.

As noted in Section III, it is possible to design a code for which the stuff word is the all-zero word and then, for transmission, suppress the all-zero word by adding an unused word to each word before trans-

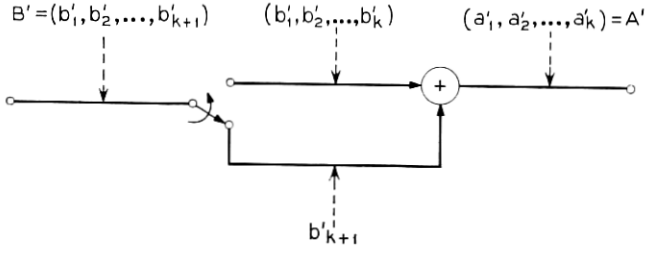

Fig. 9—Decoder for Case 1.

mission. However, it is easy to combine the encoding and addition operations. We give two examples.

5.2 *Case 2*

Let the stuff word be

$$S = (1 \underbrace{0 \cdots 0}_{k \text{ positions}}).$$

position $k + 1$

The encoder is given in (8).

$$\begin{aligned} &\text{If} \quad w(A) \geqq \left[\frac{k}{2}\right] + 1, \qquad b_{k+1} = 1. \\ &\text{If} \quad w(A) \leqq \left[\frac{k}{2}\right], \qquad b_{k+1} = 0. \end{aligned} \tag{8}$$

$$\text{Then } b_i = 1 \oplus b_{k+1} \oplus a_i \quad \text{for} \quad 1 \leqq i \leqq k.$$

The decoder performs the operations in (9).

$$a'_i = 1 \oplus b'_{k+1} \oplus b'_i \quad \text{for} \quad 1 \leqq i \leqq k. \tag{9}$$

5.3 *Case 3*

The stuff word is

$$S = (\underset{\text{position } k+1}{0} \; 0 \cdots 0 \underbrace{1 \cdots 1}_{k_1 \text{ positions}}).$$

The encoder is specified in (10).

$$\begin{aligned} &\text{If} \quad w(A) \geqq \left[\frac{k}{2}\right] + 1, \qquad b_{k+1} = 0. \\ &\text{If} \quad w(A) \leqq \left[\frac{k}{2}\right], \qquad b_{k+1} = 1. \end{aligned} \tag{10}$$

$$\begin{aligned} \text{Then } b_i &= 1 \oplus b_{k+1} \oplus a_i \quad \text{for} \quad 1 \leqq i \leqq k_1, \\ b_i &= b_{k+1} \oplus a_i \quad \text{for} \quad k_1 + 1 \leqq i \leqq k. \end{aligned}$$

The decoder performs the operations in (11).

$$a_i' = 1 \oplus b_{k+1}' \oplus b_i' \quad \text{for} \quad 1 \leqq i \leqq k_1 ,$$
$$a_i' = b_{k+1}' \oplus b_i' \quad \text{for} \quad k_1 + 1 \leqq i \leqq k. \tag{11}$$

Notice that if $k_1 = k$, the decoder for Case 3 is identical to the decoder for Case 2.

VI. THE (8, 7) CODE

Because Bell System PCM channels use a basic 8-bit word, the (8, 7) code is of interest. From Fig. 3, it is possible to design an (8, 7) code with $d_m = 4$. Let R_s and R_d denote the stuff rate and the rate of occurrence of the data words that are distance four from S, respectively. When the stuff rate is low ($R_s < R_d$), the decision rule at the receiver is biased in favor of the data words by deleting as stuff words all received words distance one from S and decoding as data words all remaining received words. Conversely, for high stuff rates ($R_s > R_d$), the decision rule is biased in favor of the stuff words by deleting as stuff words all received words distance two or less from S and decoding as data words all remaining received words. When the rates are approximately equal ($R_s \simeq R_d$), the two decision rules give comparable performance.

It is possible to modify the (8, 7) code by relaxing the minimum distance requirement so that all data words are merely required to be at least distance three from S. Received words distance one from S are deleted as stuff words; the remaining received words are decoded as data words. This balanced code gives roughly the same performance as either of the biased codes when $R_s \simeq R_d$. However, the reduction in minimum distance provides for less error multiplication (see Section V).

Let $\bar{T}_{d|s}$ denote the mean time between erroneous conversions of a stuff word into a data word and let $\bar{T}_{s|d}$ denote the mean time between erroneous conversions of a data word into a stuff word. We use the following assumptions:

(*i*) Transmission errors are independent of the transmitted bits, independent of each other, and occur with probability $p = 10^{-7}$.

(*ii*) The transmission rate is 64×10^3 bits per second or, because $n = 8$, 8000 words per second.

(*iii*) All 7-bit input words are equally likely.

(*iv*) The stuff rate is R_s stuff words per second.*

* The value of R_s will vary depending upon the application. For fine adjustment of clock rates, R_s typically would be less than 20 words per second. If word stuffing is used for speed padding as well as adjusting clock rates (for example, to send 50 kilobit service over a 64-kilobit line), R_s could be 850 words per second or larger.

The balanced (8, 7) code is applicable except for extremely high or low stuff rates. Let N_3 denote the number of data words distance three from S (the exact value of N_3 depends upon the encoder). Then

$$\bar{T}_{d|s} \cong \frac{1}{3600R_s\left[\binom{8}{2}p^2\right]} = \frac{9.93 \times 10^8}{R_s} \text{ hours}$$

and

$$\bar{T}_{s|d} \cong \frac{1}{3600(8000 - R_s)\left\{\frac{\binom{3}{2}N_3}{2^7}p^2\right\}} = \frac{1.18 \times 10^{12}}{N_3(8000 - R_s)} \text{ hours.}$$

By a modification of the encoder-decoder in Case 1 of Section V, it is possible to reduce the error multiplication in the decoder. The stuff word is

$$S = (00000000).$$

The encoder is given in (12).

$$\begin{aligned} &\text{If } w(A) \geqq 3, \quad b_8 = 0. \\ &\text{If } w(A) \leqq 2, \quad b_8 = 1. \\ &\text{Then } b_i = b_8 \oplus a_i \text{ for } 1 \leqq i \leqq 4, \\ &\qquad b_i = a_i \qquad \text{for } 5 \leqq i \leqq 7. \end{aligned} \tag{12}$$

The decoder performs the operations in (13).

$$\begin{aligned} a_i' &= b_8' \oplus b_i' \quad \text{for } 1 \leqq i \leqq 4, \\ a_i' &= b_i' \qquad\quad \text{for } 5 \leqq i \leqq 7. \end{aligned} \tag{13}$$

Notice that an error in position eight now results in four rather than seven errors for the customer.

VII. COMPARISON WITH GROUP CODES

In the binary group codes normally encountered in error-control work, the code words are a set of 2^k n-tuples selected so that the code words form a group under component-by-component modulo two addition. Because of the resulting perfect distance symmetries between the code words,* the probability that a transmitted word is decoded in

* Let X and Y be code words. For any $\epsilon(1 \leqq \epsilon \leqq n)$, the number of code words distance ϵ from X is equal to the number of code words distance ϵ from Y for all X and Y.

error does not depend upon the transmitted word (provided the transmission errors are independent of the transmitted bits).

The code presented herein has $2^k + 1$ code words (the 2^k elements of D plus the stuff word S). The code words do not form a group and exhibit highly asymmetric distance properties. It is the asymmetric distance properties that enable us to use the available redundancy to protect against the loss of character synchronization due to transmission errors.

We note that it is possible to design other asymmetric codes that are, in a sense, generalizations of the code in (2). Instead of a single stuff word, there are now several special words with unique distance properties with respect to each other and the set D. Such codes might be used in a data transmission system where, for example, one wishes to provide more protection for control characters than for data words. A wide range of capabilities is possible and future work in the design of these codes should prove profitable.

VIII. CONCLUSIONS

For a given redundancy, we give the code that maximizes the minimum distance between the stuff word and any data word. An encoder and decoder are given for arbitrary n and k. For $n = k + 1$, a particularly simple encoding-decoding technique is described. Certain properties of the (8, 7) code are considered in detail.

IX. ACKNOWLEDGMENT

The author wishes to thank E. J. Hronik for pointing out the importance of this problem and for a number of useful discussions during the course of the work.

APPENDIX A

Derivation of d_m for $n = k + 1$

Let $n = k + 1$. From (2b), d_m is chosen so that

$$\sum_{l=1}^{d_m-1} \binom{k+1}{l} \leqq 2^k - 1 < \sum_{l=1}^{d_m} \binom{k+1}{l}. \tag{14}$$

However,

$$2^{k+1} - 2 = \sum_{l=1}^{k} \binom{k+1}{l}.$$

Therefore, for k even,

$$2^k - 1 = \sum_{l=1}^{k/2} \binom{k+1}{l}$$

which, from (14), implies that

$$d_m = \frac{k}{2} + 1$$

and, from (2c), that $\delta = 0$. For k odd,

$$2^k - 1 = \sum_{l=1}^{[k/2]} \binom{k+1}{l} + \frac{1}{2}\binom{k+1}{\left[\frac{k}{2}\right]+1} \tag{15}$$

where $[k/2]$ denotes the largest integer $\leqq k/2$. Thus, from (14) and (15),

$$d_m = \left[\frac{k}{2}\right] + 1.$$

APPENDIX B

Encoder for (9, 7) *Code*

For the (9, 7) code, $d_m = 6$. Thus, construct the three arrays shown in Table II. Consider encoding $\alpha_0 = 22$. In the first array, $\alpha_1(5) = 15$ is the largest $\alpha_1(j)$ not greater than 22. Therefore, $j_1 = 5$. Next, in the second array, we find that the largest $\alpha_2(j)$ not greater than

$$\alpha_0 - \alpha_1(5) = 22 - 15 = 7$$

is $\alpha_2(4) = 7$. Thus, $j_2 = 7$. However,

$$\sum_{m=1}^{2} \alpha_m(j_m) = \alpha_1(5) + \alpha_2(4) = 22 = \alpha_0$$

TABLE II—ARRAYS FOR ENCODING FOR THE (9, 7) CODE.

Array 1		Array 2		Array 3	
j	$\alpha_1(j)$	j	$\alpha_2(j)$	j	$\alpha_3(j)$
1	1	1	1	1	1
2	2	2	2	2	2
3	4	3	4	3	3
4	8	4	7	4	4
5	15	5	11	5	5
6	26	6	16	6	6
7	42	7	22	7	7
8	64	8	29		
9	93				

so the process terminates. Therefore, B_0 has ones in positions $j_1 = 5$ and $j_2 = 4$, that is,

$$B_0 = (000011000).$$

position 5 ↑↑ position 4

APPENDIX C

Proof of Equation (5)

After substituting (4) into (5), we must show that

$$\sum_{m=0}^{\mathrm{Min}(n-d_m-i+1,\,j-1)} \binom{j-1}{m} + \sum_{m=0}^{\mathrm{Min}(n-d_m-i,\,j-1)} \binom{j-1}{m} = \sum_{m=0}^{\mathrm{Min}(n-d_m-i+1,\,j)} \binom{j}{m} \qquad (16)$$

for $1 \leqq j \leqq n - i$, $1 \leqq i \leqq n - d_m - 1$. Choose an i and consider j as j increases from 1 to $n - i$. Suppose that $1 \leqq j \leqq n - d_m - i + 1$. Then (16) reduces to

$$\sum_{m=0}^{j-1} \binom{j-1}{m} + \sum_{m=0}^{j-1} \binom{j-1}{m} = \sum_{m=0}^{j} \binom{j}{m}$$

or

$$2^{j-1} + 2^{j-1} = 2^j.$$

Now, suppose that $n - d_m - i + 1 < j \leqq n - i$. Then (16) becomes

$$\sum_{m=0}^{n-d_m-i+1} \binom{j-1}{m} + \sum_{m=0}^{n-d_m-i} \binom{j-1}{m} = \sum_{m=0}^{n-d_m-i+1} \binom{j}{m}.$$

However,

$$\begin{aligned} 1 + \sum_{m=1}^{n-d_m-i+1} \binom{j-1}{m} + \sum_{m=0}^{n-d_m-i} \binom{j-1}{m} &= 1 + \sum_{m=0}^{n-d_m-i} \left[\binom{j-1}{m+1} + \binom{j-1}{m}\right] \\ &= 1 + \sum_{m=0}^{n-d_m-i} \binom{j}{m+1} \\ &= \sum_{m=0}^{n-d_m-i+1} \binom{j}{m}. \end{aligned}$$

The argument is valid for each i, $1 \leqq i \leqq n - d_m - 1$.

REFERENCES

1. Mattesich, R. R., and Richters, J. S., unpublished work, Bell Telephone Laboratories, December 13, 1968.
2. Butman, S., "Synchronization of PCM Channels by the Method of Word Stuffing," IEEE Trans. Comm. Tech., *COM-16,* No. 2 (April 1968), pp. 252–254.
3. Johannes, V. I., and McCullough, R. H., "Multiplexing of Asynchronous Digital Signals Using Pulse Stuffing with Added-Bit Signaling," IEEE Trans. Comm. Tech., *COM-14,* No. 5 (October 1966), pp. 562–568.
4. Mayo, J. S., "An Approach to Digital System Networks," IEEE Trans. Comm. Tech., *COM-15,* No. 2 (April 1967), pp. 307–310.
5. Mayo, J. S., "Experimental 224 Mb/s PCM Terminals," B.S.T.J., *44,* No. 9 (November 1965), pp. 1813–1841.
6. Witt, F. J., "An Experimental 224 Mb/s Digital Multiplexer-Demultiplexer Using Pulse Stuffing Synchronization," B.S.T.J., *44,* No. 9 (November, 1965), pp. 1843–1885.