Recurrent Codes: Easily Mechanized, Burst-Correcting, Binary Codes

By D. W. HAGELBARGER

(Manuscript received April 14, 1959)

A class of codes capable of correcting multiple errors is described. Some of these codes can be implemented with considerably less hardware than was needed for previous multiple error-correcting codes. A general method is shown for constructing a code of redundancy 1/b that will correct error bursts of Kb or fewer digits (K and b integers). The logical design of the encoder and decoder, as well as the guard space requirement of good digits between bursts of errors, is described.

I. INTRODUCTION

In adapting the existing telephone network to high-speed digital data transmission, an error control problem arises. Most of the circuits were designed primarily for voice-type signals and considerable attention was given to control of thermal noise. Because of the high redundancy of speech, impulse noise on the lines is usually not even noticed by the telephone users, and hence has not been a serious problem. On the other hand, high-speed digital data (especially numerical data) contain little redundancy, and the noise pulses may resemble the signal pulses and thus cause errors.

The deliberate introduction of redundancy to detect and correct transmission errors has been used for some time. Early systems[1] used repetition of characters and duplication of channels. There were two schemes which sent pictures of the characters using raster scans. By the late 1930's a radio telegraph system[2] using a 3-out-of-7 code for error detection had been patented and telephone apparatus using 2-out-of-5 codes[3] was being designed.

Most, if not all, of the recent work on error-detecting or error-correcting codes stems from Hamming's Systematic Parity Check codes.[4] These codes will correct a single error per block of digits. Since then, much work has been done on codes for multiple errors (see Refs. 5 through 16). The assumption has usually been made that the errors are statistically independent. On many communication channels, however, the errors are not independent but tend to come in groups. For example, a lightning stroke may knock out several adjacent telegraph pulses. These groups of errors are called "bursts". Codes for detecting and correcting bursts have been proposed by Abramson,[17] Gilbert,[18] Hamming[13] and Meyer.[13] The class of codes described here differs in that the block structure has been minimized; the resulting symmetry allows very simple mechanization and also simplifies the synchronization problem. Since the block size is small, the codes also fit very naturally into systems where the data must be accepted and delivered continuously, rather than in batches.

II. EXAMPLE

Before describing the general recurrent code we will give a particularly simple example. Assume that we wish to correct bursts of length six or less. The simple code has every other digit a check digit, giving a redundancy of one-half. The encoder is illustrated in Fig. 1. It consists of a shift register of length seven. The data digits enter from the left (Position 1) and are shifted through the register before being transmitted. For each shift we generate a check digit so that the parity (number of 1's) of the check digit and the data digits in the first and fourth positions of the shift register is even (zero or two). This check digit is transmitted before the data digit in the seventh position. Then a shift is made; the data digit which was in Position 7 is transmitted, and a new check digit is calculated. This process is illustrated in Fig. 2; the successive lines are one shift time apart. During this time interval one new data digit is accepted by the encoder and two digits, one data, one check, are transmitted.

The decoder is shown in Fig. 3. The received code enters the switch where the alternating data and check digits are separated, the check digits going to the lower shift register and the data digits to the upper

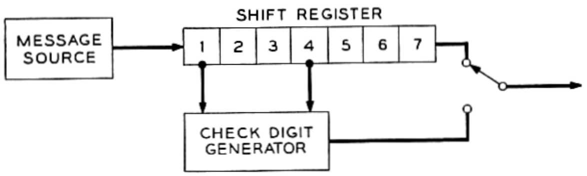


Fig. 1 — Encoder.

TIME	DATA BITS	ENCODER SHIFT REG $D_1 + D_4 \equiv C$ (MOD 2)								TRANSMITTED CODE
		1	2	3	4	5	6	7	C	
0	0 1 1 0	0	1	1	0	1	0	0	0	
1	0 1 1	0	0	1	1	0	1	0	1	0 0
2	0 1	1	0	0	1	1	0	1	0	0 1 0 0
3	0	1	1	0	0	1	1	0	1	1 0 0 1 0 0
4		0	1	1	0	0	1	1	0	0 1 1 0 0 1 0 0

Fig. 2 — Timing chart of digits moving through the encoder.

one. There are two copies of the parity circuits, R and S, each one checking the parity relation imposed by the encoder. The decoding rule is:

> Whenever both R and S fail, change the data digit in Position 4 ($0 \rightarrow 1$, $1 \rightarrow 0$) while shifting it to Position 5. If only one parity check fails, make no change.

The corrected data digits are available at Position 5 of the data shift register. In a burst of length six or less, there can be at most three data digits and three check digits wrong. The parity relation used in encoding involves digits which are spread far enough apart so that no burst of six or less will affect more than a single digit in any one parity group.

After any burst of length six or less, a 20-digit errorless message is enough to completely refill the decoder shift registers. Hence, the next burst can be corrected without interference from a previous burst, if there is a "clean" message 19 or more digits long between bursts.

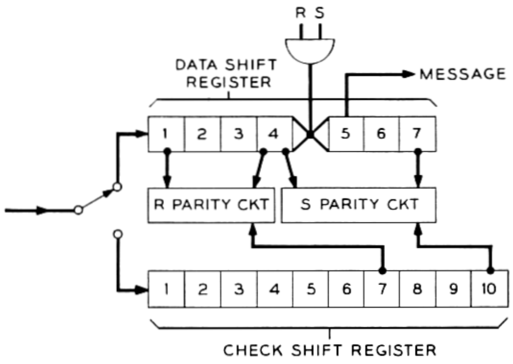


Fig. 3 — Decoder.

TABLE I

Burst Length	Data Shift Register Length	Check Shift Register Length	Guard Space Between Bursts
4	5	7	13
6	7	10	19
8	9	13	25
10	11	16	31

If the message is to be retransmitted the check digits can be corrected at Position 10 of the check digit shift register (Fig. 3). The rule is:

Whenever parity check R holds and parity check S fails, change the digit in Position 10 of the check digit register.

A data digit error cannot cause parity R to hold and parity S to fail, because this would require an error in Position 7 of the data register, and, since all data digit errors are corrected in going from Position 4 to Position 5, this cannot happen.

This particular coding scheme fails first for a burst of length seven, consisting of a check digit and another check digit six digits later. When just these two check digits are wrong, the decoder assumes that a data digit is wrong and changes it.

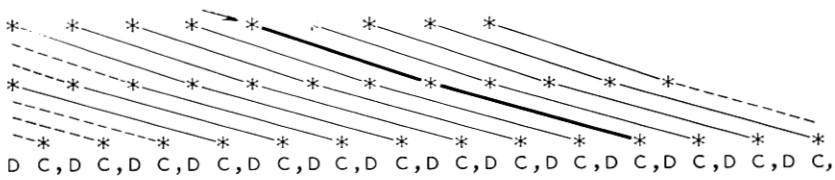
A demonstration device using this code has been built. It consists of a punched tape reader, an encoder, a transmission line, a decoder and a tape printer. The circuitry uses relays and can be operated fast, slow or one step at a time. Digits in the encoder, transmission line and decoder are displayed on lamps. The transmission line has switches for inserting errors in the encoded message. Other lamps are used to indicate parity check failures. An auxiliary circuit can be used for detecting overlong bursts, flashing a yellow lamp whenever the decoder has a burst and locking up a red lamp whenever a detectable overlong burst occurs.

There is an extension of this code to correct bursts of any even length. We merely spread out the parity check so that the burst can only effect one term in any parity group. To correct bursts of length $2K$ or less requires an encoding shift register of length $2K + 1$. The decoding data digit register has the same length, $2K + 1$, and the check digit register must be $3K + 1$. The parity checks involve digits K apart in the registers. A clean message of length $6K + 1$ will always be sufficient separation between bursts. Table I shows typical values.

III. GENERAL RECURRENT CODE

The general (binary) recurrent code is constructed as follows:

We are given a *message* to be transmitted consisting of *data* digits.


D C, D C, D C, D C, D C, D C, D C, D C, D C, D C, D C, D C, D C, D C, D C,

$P_1 = P_D$	1	0	0	1	0	0	0
$P_2 = P_C$	0	0	0	0	0	0	1

Fig. 4 — Portion of encoded message.

We will add to this *check* digits to form the *encoded message*. The encoded message is divided into blocks of length b. One position in the block is assigned to be a check digit† and the rest are data digits. Since the block must have at least one data digit, the shortest block length is $b = 2$. (This is the value used in the example of Section II.) The data digits are loaded into the data digit positions in the order received. The check digit is determined by a parity relation applied once for each block. This parity relation extends over a selected set of the digits in p consecutive blocks. (To be useful against bursts, p must be at least 2 and usually will be larger.) Fig. 4 shows a portion of the encoded message from the example of Section II. The data and check bits are indicated by D's and C's; the blocks are marked off with commas and the parity relation is shown by lines having *'s over the digits in a given *parity group*. Note that p for this example is 7; that is, any one of the parity groups extends over seven blocks. Every parity group has three digits in it, two data and one check; thus, each data digit is in two parity groups and each check digit is in only one parity group.

We will denote which digits enter into the parity relation by b binary words of p digits each. We call these the *parity words* and label them P_1, P_2, $\cdots$, P_b. Consider p consecutive blocks. We form P_1 by observing the first position of each block; if the digit is in this parity group we write 1, otherwise 0. Then P_2 depends on the second positions of these p blocks, and so on. This is illustrated in Fig. 4. We have used the parity group indicated by the arrow and written the parity words so that the digits fall under the corresponding blocks. Thus P_1 has 1's in the first and fourth blocks and P_2 has a 1 only in the seventh block. (The numbering of digits and blocks here and in Fig. 4 is from right to left. Fig. 4 can be considered a "snapshot" of the encoded message on the trans-

† Codes with more than one check digit per block are possible, but it can be shown that, for a given efficiency and burst-correcting capability, they always require more complicated encoding and decoding equipment than would the equivalent code with one check digit per block.

mission line. This is different from the numbering method used below, where the digits are considered to be flowing past a fixed point, and are labeled with numbers which increase with time.) With this notation, there is a simple correspondence between the 1's in the parity words and the connections to the shift registers of the encoder and decoder.

Another way to describe these codes is to think of the digits leaving the encoder as being numbered serially; if X_i is the ith digit, then the parity relation is given by a recurrent equation of the form:

$$X_{r+kb} \oplus X_{s+kb} \oplus \cdots \oplus X_{w+kb} = \text{constant},$$

where the constant is 0 or 1, $\oplus$ means sum modulo 2, k takes successive integral values, b is the block length, and $r, s, \cdots, w$ denote which digits enter into the parity group.

If we order the terms so that $r < s < \cdots < w$, then

$$p - 2 < \frac{w - r}{b} < p.$$

The requirements that every position in the block must be represented at least once implies that each of the integers $0, 1, \cdots, (b - 1)$ must occur at least once as a remainder upon dividing $r, s, \cdots, w$ by b.

For the example of Section II we have:

$$X_{1+2k} \oplus X_{8+2k} \oplus X_{14+2k} = 0.$$

IV. BURSTS

Consider an encoded message flowing through a communication channel. A burst occurs and some of the digits of the message are changed. We will describe the burst pattern by a binary word having a 1 for each changed digit and a 0 for each correct digit. We require that the first and last digits of the word both be 1's, since there is no point in including correct digits which are outside of the bursts. The length of the burst is the number of digits in the word. There are 2^{l-2} different burst patterns of length l and 2^{l-1} different burst patterns of length l or less. The latter are the odd binary numbers having l or fewer digits. For example, the eight burst patterns of length 4 or less are: 1, 11, 101, 111, 1001, 1011, 1101 and 1111.

The effect of a burst on the encoded message depends on the phase of the burst pattern with respect to the block structure; hence, we will have to consider $b2^{l-1}$ different possible bursts of length l or less if the code has a block length b. We will indicate particular bursts either by showing a portion of the encoded message with the erroneous digits marked with *'s or by listing the erroneous digits with superscripts to indicate which block a particular digit occupies. For example,

$$\cdots DC, \overset{*}{D}\overset{*}{C}, D\overset{*}{C}, \overset{*}{D}C, \cdots$$

is the same as $D^2C^2C^1D^0$.

V. SYNDROMES

At the decoder we will always have a circuit for checking the parity relation imposed by the encoder. This circuit gives an output once for each block received. If the parity check fails, the output is a 1; otherwise, it is a 0. In a practical transmission system, there are usually no errors and the check circuit has an output of all 0's. When a burst of errors does occur, the check circuit will give a pattern of 1's and 0's. This pattern will be used to identify the burst, and hence we shall call it the *syndrome*. Since the syndromes occur immersed in a string of 0's, only binary words having 1's on both ends (odd numbers) can be used as syndromes and, as above, there are 2^{k-1} possible different syndromes with k or fewer digits.

Our first problem is to choose the parity relation in such a fashion that each burst of length l or less has a distinct syndrome.† A further problem is to choose the parity relation so that, given a distinct syndrome, the correction of the burst which caused it is easily mechanized. That is, we want a systematic scheme for correcting a burst, given the corresponding syndrome, which is much better than having a table of all possible syndromes with the corresponding bursts.

The procedure for calculating the syndrome corresponding to a given burst is as follows:

$$\cdots D\ C, D\ C, D_3\overset{*}{C}, \overset{*}{D}_2C, \overset{*}{D}_1\overset{*}{C}, \overset{*}{D}_0C, D\ C, D\ C, \cdots$$

$$\begin{array}{lcccccccccc}
P_D{}^0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & & & \\
P_C{}^1 & & 0 & 0 & 0 & 0 & 0 & 0 & 1 & & \\
P_D{}^1 & & 1 & 0 & 0 & 1 & 0 & 0 & 0 & & \\
P_D{}^2 & & & 1 & 0 & 0 & 1 & 0 & 0 & 0 & \\
P_C{}^3 & & & & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
\text{Syndrome:} & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1
\end{array}$$

† If our code has a block length b which is a power of 2, then it is possible for $b2^{l-1} = 2^{k-1}$. A *close-packed* recurrent code is one which has all possible syndromes of k or fewer digits in a one-to-one correspondence with all possible bursts of length l or less. Since this is of more mathematical than practical interest, examples are deferred to Appendix I.

Number the blocks, starting with 0 for the first block containing an error, and continue the numbering far enough to include all blocks having errors in the burst under consideration. (The direction of numbering should be such that increasing numbers represent digits received at later times.) Write the parity word for the error in block number 0. Under this write the parity words for any other errors in this block. Now write the parity words for the other errors, shifting each one (in the direction of increasing time) the number of places equal to the block number in which the error occurs. The syndrome is the sum modulo 2 of these parity words. In the sample above we use superscripts on the parity words to indicate in which block each error occurred. This shows the syndrome for the indicated burst of length 6, using the code of Section II. Note that there is never more than a single 1 in any column. By spreading out the parity words with 0's sufficiently to prevent any interaction of errors (in a burst not larger than the design maximum), we allow the use of a simple circuit for recognizing the parity words and correcting the errors one at a time. In other words, the decoder for our example of Section II is simple because we have a single circuit for correcting a data-digit error, which is time-shared by every data digit. Each data digit is compared with four other digits, all far enough away from each other so that not more than one of these five digits can be in error due to an allowable burst. Under these circumstances, it is an easy matter to decide if the particular data digit needs correcting.

VI. LOWER REDUNDANCY CODES

We can apply the same technique, spreading out the parity words with 0's so as to avoid interactions between errors in a burst, to make the redundancy as low as we wish and still have a relatively simple correcting mechanism. For instance, if we desire a code with a redundancy of one-quarter good for bursts of length 4, we choose the following parity words (the digits are spaced to emphasize the method of forming):

P_A	111	000	000	0
P_B	000	101	000	0
P_C	000	000	110	0
P_D	000	000	000	1

Note that placing the code 100 to the extreme right allowed us to shorten the parity words by dropping the last two columns, which were all 0's. In assigning the parity words, the groups of 1's should be arranged to go from upper left to lower right as shown above. That is, the order of the groups of 1's in the parity words should agree with the order of the digits in the block structure. If this is not done extra columns of 0's must be inserted in the array of parity words, which would mean more shift register stages in the encoder and decoder. The difficulty is illustrated as follows:

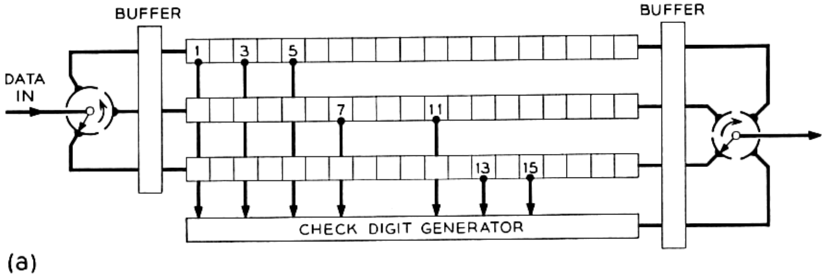
		Code I	*Code II*
	P_A	111 000 000 0	000 111 000 0
	P_B	000 101 000 0	101 000 000 0
Burst:		$\cdots A\overset{*}{B}CD\overset{*}{A}B \cdots$	
	$P_A{}^0$	1110000000	0001110000
	$P_B{}^1$	0001010000	1010000000
Syndrome:		1110101	10011
		[O.K.]	[N.G.]

(The burst $\cdots \overset{*}{A}BCDA\overset{*}{B} \cdots$ is not allowed, since the above codes are for bursts of length 4 or less.) If we wish to make a code of the same redundancy good for bursts of length 8 or less, we form the parity words by inserting 0's between each of the digits of the above code, giving:

P_A	10101	000000	000000	00
P_B	00000	010001	000000	00
P_C	00000	000000	010100	00
P_D	00000	000000	000000	01

Fig. 5(a) shows an encoder for this code. The data digits enter from the left side and are cyclically switched to the three shift registers by the input commutator. The buffers allow the shift registers to be stepped together. A check digit, calculated from the parity of the indicated positions of the shift registers is transmitted with the digits from the last positions of the registers by the output commutator. Note the correspondence between the parity words and the shift registers. The top register comes from P_A, the second from P_B, and so on. Each digit of a parity word becomes a stage of shift register. The stages representing 1's are connected to the parity circuit; those representing 0's are not. If P_D, which has a single 1 at the right end, is assigned to the check digit, no shift register is required at the encoder for this parity word. (However, one is needed in the decoder.)

At the decoder, Fig. 5(b), the incoming digits are commutated to the four shift registers (synchronization must be maintained so that the digits get in the proper registers.) As each block arrives at the decoder, the parity relation is checked; if it fails, a 1 is put in the syndrome register at the bottom of the figure. Suppose that a digit in the top register is wrong; it will cause parity failures as it goes through Positions 1, 3 and 5. When it is in Position 5 the syndrome register will have 1's in Positions R, S and T. This will enable the AND circuit, which will correct the error as it shifts from Position 5 to Position 6. In a similar manner, an error in the second register is corrected between Positions 11 and 12, and an error in the third register between Positions 17 and 18. Whenever a 1 reaches Position T of the syndrome register, any 1's in Positions R and S are cleared on the next shift. (If for some reason it should be desirable to correct the check digits rather than discard them, this can be done by adding the extension to the bottom register shown dotted.) Because of the way the taps for the parity circuit are spaced,



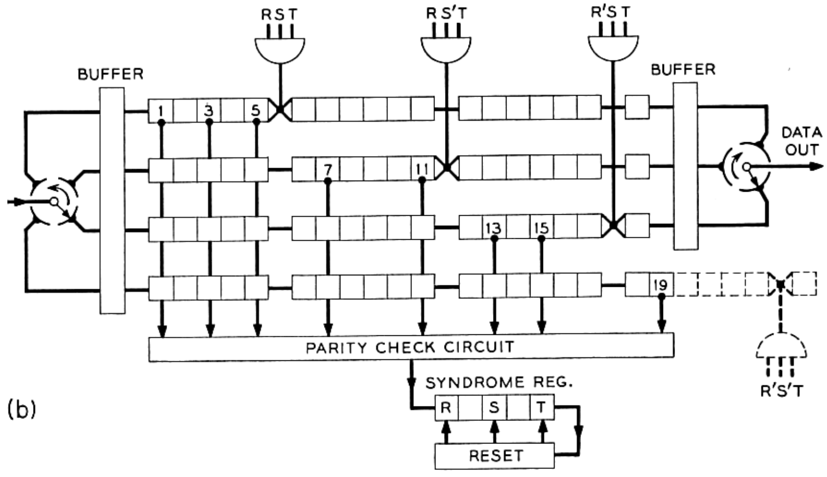


Fig. 5 — (a) Encoder; (b) decoder.

any register can correct two adjacent errors, and the system is good against bursts of length 8 or less.

It takes 92 digits entering the decoder to completely refill all the registers (including the syndrome register). Thus, a guard space of 91 good digits between bursts is sufficient to assure that there is no interaction between bursts.

In general, a procedure for constructing a code of redundancy $1/b$ (block length b) is as follows:

> Take the first b binary numbers and let L be the number of digits in the largest one. Form each of these numbers to a L-digit word by adding zeros to the *right* end. Now form a square array with b rows and b columns. The entries in the array are L-digit words. Put the above-formed words along the main diagonal, with the word having a single 1 going in the lower right corner. The order of the other words on the diagonal is arbitrary. Fill in all remaining words with zeros. Now replace the right-hand column of L-digit words with single-digit words; 1 in the bottom row, 0 elsewhere. (Strike out the $(L - 1)$-digit columns from the right.)

The rows of this array are the parity words of the desired code. The order from top to bottom is the ("snapshot") order of occurrence of the corresponding digits in the block structure of the message.* This code will correct all bursts of length b or less. To make a code good for burst of length Kb or less, add $K - 1$ zeros between each adjacent pair of digits of the above parity words.

If the odd binary numbers were not increased to L digits as above, certain otherwise allowable bursts could cause syndromes which would be incorrectly interpreted by the decoder. For instance, 001 and 110 might add to form 001110. The procedure given prevents this type of difficulty.

VII. SHIFT REGISTER AND GUARD SPACE REQUIREMENTS

If we design codes by the method indicated in the previous section, the method is regular enough to allow us to give formulas for the shift register stages and guard space.†

Definitions:

$$b,\ l,\ k,\ L(b) \text{ are positive integers.}$$

* The bottom row is the parity word for the digit that is transmitted first in any block. See the direction of rotation of the output commutator in Fig. 5(a).

† If the block length is not a power of 2 it may be possible to save a little from these calculations by taking advantage of the fact that one or more of the odd numbers is not used. For example, the burst-length 3, redundancy one-third decoder shift registers can be reduced from 24 to 21 stages.

The block length is b with one check digit; hence, the redundancy is $1/b$. The code is to be usable against bursts of length l or less, where $l = kb$. $L(b)$ is the smallest integer such that

$$L(b) \geqq 1 + \log_2 b.$$

Thus,

b	2	3	4	5	6	7	8	9	10
$L(b)$	2	3	3	4	4	4	4	5	5

The encoder will have $b - 1$ registers, each of length

$$\left(\frac{l}{b}\right) L(b)(b - 1) + 1$$

or

$$(b - 1)^2 \left(\frac{l}{b}\right) L(b) + b - 1$$

stages. The decoder will have

$$l(b - 1)L(b) + 2b + \left(\frac{l}{b}\right) L(b) - l$$

stages.

The guard space can be shown to be

$$blL(b) + b - l - 1.$$

Table II shows some typical values. (Efficiency is 1-redundancy.)

VIII. DETECTION OF OVERLONG BURSTS

Any error-correcting system fails when the errors get beyond its correcting capabilities. In some cases, it is desirable to detect that a burst has occurred which the system cannot correct. In Section II we mentioned that the code for that example failed on a burst consisting of two check-digit errors exactly six apart. The effect of this burst is to cause the decoder to change the data digit which is common to the parity groups containing these check digits. This is a fundamental difficulty of the particular code and cannot be avoided, since such a burst converts our encoded message to what looks like a different encoded message with a single data-digit error. In general, we cannot correct or even detect bursts which convert one encoded message to another encoded message, or to another encoded message modified by an allowable burst (assuming we still wish to correct allowable bursts). For the example of Section II there are:

four bursts of length 9,
two bursts of length 8,
one burst of length 7,

which convert one encoded message to another encoded message modified by an allowable burst. These seven bursts are not detectable, but all other bursts of length 9 or less are detectable and, of course, all bursts of length 6 or less are correctable. In connection with the code demonstrator described in Section II, a circuit which detects overlong bursts by monitoring the sequence of failures of the two parity circuits in Fig. 3 has been built. All 512 bursts of length 9 or less have been tried on it, and it rings the alarm on all the uncorrected bursts except the seven listed above.

In the code of Section II the syndrome for a single data error is the same as the syndrome for a pair of check-digit errors six apart. To improve the error-detection capabilities of our code, we can always change the parity words so that the first occurrence of two bursts giving the same syndrome involves bursts longer than the correctable one.

As an example, consider the code given by the parity words

$$P_D = 1001001000000,$$

$$P_C = 0000000001001.$$

This particular code permits a very simple overlong-burst-detection scheme. It corrects or detects all bursts of length 13 or less and corrects all bursts of length 6 or less. The encoder and decoder are shown in

TABLE II

Block	Efficiency (in per cent)	Burst Length	Shift Register Stages		Guard Space
			Encoder	Decoder†	
2	50	2	3	8	7
		4	5	12	13
		6	7	16	19
		8	9	20	25
		10	11	24	31
3	67	3	14	24	26
		6	26	42	50
		9	38	60	74
4	75	4	30	43	47
		8	57	78	91
5	80	5	68	89	99
		10	132	168	194

† The decoder here is different from the one for Table I; this one has a syndrome register.

Fig. 6. Note that, in the process of correcting an allowable burst, the three parity-check circuits R, S and T cannot have either of the failure patterns 010 or 101. It happens that one or the other of these patterns occurs for every burst of length 13 or less that is not corrected by the decoder.

The decoder here shows an alternative arrangement compared to Fig. 5. Instead of the syndrome shift register, we have three copies of the parity check circuit shifted so that Position 7 of the data register is the only one common to all three. If we were to make a circuit similar to Fig. 5, we would save the S and T parity circuits, the last five stages of the data register and the last six stages of the check register in exchange for a seven-stage syndrome register with its reset circuit.

IX. SYNCHRONIZATION

In general, there are two synchronization problems with any binary code, bit synchronization and block synchronization. For purposes of

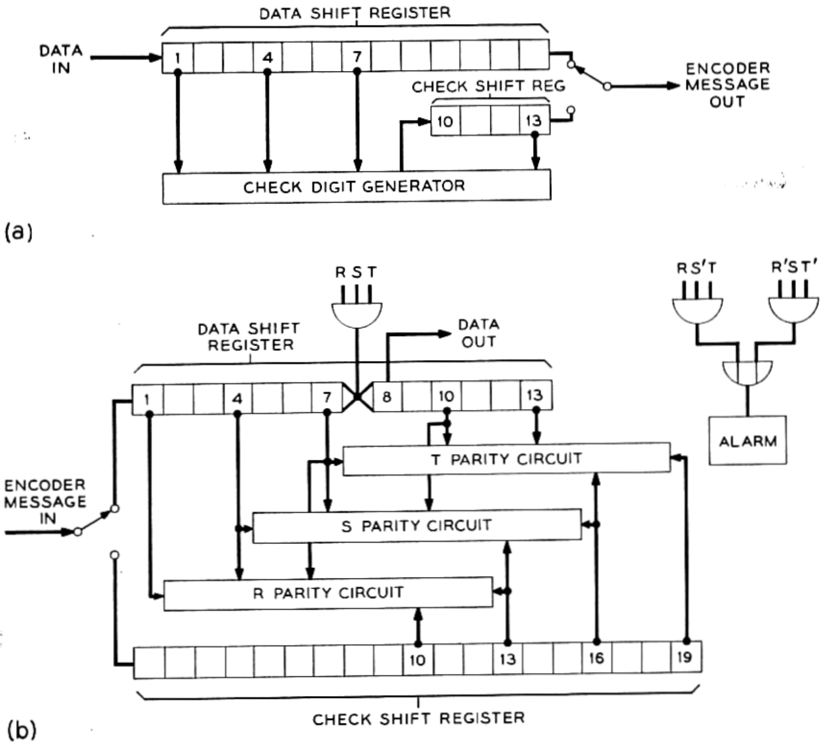


Fig. 6 — (a) Encoder; (b) decoder.

TABLE III — IMPOSSIBLE CODES

		Number of Terms	
		ODD	EVEN
Number of Ones	ODD	$00000\cdots$	$0000\cdots$, $11111\cdots$
	EVEN	$11111\cdots$	

bit synchronization, it is desirable to have transitions from 0 to 1 or 1 to 0 at some minimum rate. By proper choice of the number of terms in the parity relation and also whether the parity is odd or even we can prevent either of the codes $00000\cdots$ or $11111\cdots$ from occurring in the encoded message, regardless of what the input to the encoder may be. It is probably also desirable to exclude these codes as possible encoded messages since they are the most likely messages to be put out by an encoder with something stuck on or off. Table III shows which codes cannot occur.

TABLE IV

Redundancy	Burst	Syndrome
$\frac{1}{2}$	A^0	111
	B^0	001
	A^0B^0	110
	B^1A^0	101
$\frac{1}{4}$	A^0	1100
	B^0	0111
	C^0	1101
	D^0	1001
	A^0B^0	1011
	B^0C^0	1010
	C^0D^0	0100
	D^1A^0	1111
$\frac{1}{8}$	A^0	01100
	B^0	10111
	C^0	11100
	D^0	01101
	E^0	10010
	F^0	11101
	G^0	11001
	H^0	10011
	A^0B^0	11011
	B^0C^0	01011
	C^0D^0	10001
	D^0E^0	11111
	E^0F^0	01111
	F^0G^0	00100
	G^0H^0	01010
	H^1A^0	10101

With the redundancy one-half code, the block synchronization problem is minimized, since there are only two phases that the decoder can have. One possibility is to make a decoder with two equal shift registers and two copies of the error-correcting circuit, one wired in each of the possible phases. The fact that the wrong parity circuits fail half of the time can be used to tell which phase to use.

X. ACKNOWLEDGMENT

I wish to thank E. F. Moore for many helpful suggestions. R. V. Anderson built the demonstration device described in Section II.

APPENDIX

Examples of Close-Packed Codes

All of the examples known to date are for burst length two. It is not too hard to show that there is no close-packed code of redundancy one-half good for burst length three (see Table IV).

REFERENCES

1. Storch, P., Evolution of Long Distance Type-Printing Traffic by Wire and Radio, Elek. Z., **55,** 1934, pp. 109; 141.
2. Moore, J. B. and Mathes, R. E., U. S. Patent 2,183,147.
3. Holbrook, B. D., U. S. Patent 2,317,191.
4. Hamming, R. W., Error Detecting and Error Correcting Codes, B.S.T.J., **29,** 1950, p. 147.
5. Plotkin, M., Binary Codes with Specified Minimum Distance, Research Div. Rep. 51-20, Moore School of Electrical Engineering, Univ. of Pennsylvania, 1951.
6. Golay, M. J. E., Binary Coding, I.R.E. Trans., **PGIT-4,** 1954, p. 23.
7. Elias, P., Error-Free Coding, I.R.E. Trans., **PGIT-4,** 1954, p. 29; Coding for Noisy Channels, I.R.E. Conv. Rec., 1955, Part 4, p. 37.
8. Muller, D. E., Application of Boolean Algebra to Switching Circuit Design and to Error Detection, I.R.E. Trans., **EC-3,** 1954, p. 6.
9. Reed, I. S., A Class of Multiple-Error-Correcting Codes and the Decoding Scheme, I.R.E. Trans., **PGIT-4,** 1954, p. 38.
10. Slepian, D., A Class of Binary Signalling Alphabets, B.S.T.J., **35,** 1956, p. 203.
11. Lloyd, S. P., Binary Block Coding, B.S.T.J., **36,** 1957, p. 517.
12. Ulrich, W., Non-Binary Error Correction Codes, B.S.T.J., **36,** 1957, p. 1341.
13. Brown, A. B. and Meyers, S. T., Evaluation of Some Error Correction Methods Applicable to Digital Data Transmission, I.R.E. Conv. Rec., 1958, Part 4, p. 37.
14. Green, J. H., Jr., and SanSoucie, R. L., An Error-Correcting Encoder and Decoder of High Efficiency, Proc. I.R.E., **46,** 1958, p. 1741.
15. Kautz, W. H., A Class of Multiple-Error-Correcting Codes, Stanford Research Institute, 1958.
16. Sacks, G. E., Multiple Error Correction by Means of Parity Checks, I.R.E. Trans., **IT-4,** 1958, p. 145.
17. Abramson, N. M., A Class of Systematic Codes for Nonindependent Errors, Stanford Electronics Labs., Tech. Rep. No. 51, 1958.
18. Gilbert, E. N., A Problem in Binary Coding, Symposium on Combinatorial Designs and Analysis, Amer. Math. Soc., 1958 (to be published).