Concluding Remarks

A class of codes, having check matrices which are the tensor products of the check matrices of nonbinary and binary codes has been described. In particular, it has been shown in detail how the error-correction, error-detection and error-location capabilities of such codes depend on the component codes.

Many other variations of such codes exist in addition to those discussed in this paper. In some applications it may be desirable to have subblocks of various sizes. One of the most *easily* implemented means to this end would be to choose the code C' to yield the maximum desired subblock size and then delete columns of the check matrix $\mathbf{H}'$ corresponding to the desired "shorter" subblocks. (If C' is a cyclic code, shortened cyclic codes would be used for the shorter subblocks). Alternatively, C' can be replaced by a sequence of binary codes (each code for a separate subblock) all of which have ρ check digits but possibly have different lengths and error-control capabilities. Moreover, independent of the choice of block length other classes of error patterns in addition to random errors and single burst errors can be utilized for $\mathcal{E}_i'$ and $\mathcal{E}_i''$. For example, the process can be iterated to obtain codes for the correction of "bursts of bursts $\cdots$ of bursts." A channel model in which errors occur in "bursts of bursts $\cdots$ of bursts" has been proposed by Mandelbrot [20].

References

[1] Reed, I. S., A class of multiple-error-correcting codes and the decoding scheme, *IRE Trans. on Information Theory*, vol IT-4, Sep 1954, pp 38–49.

[2] Bose, R. C., and D. K. Ray-Chaudhuri, A class of error-correcting binary group codes, *Information and Control*, vol 3, Mar 1960, pp 68–79.

[3] Hocquenghem, A., Codes correcteurs derreurs, *Chiffres*, vol 2, Sep 1959, pp 147–156.

[4] Reed, I. S., and G. Solomon, Polynomial codes over certain finite fields, *J. Siam*, vol 8, Jul 1960, pp 16–21.

[5] Abramson, N. M., A class of systematic codes for non-independent errors, *IRE Trans. on Information Theory*, vol IT-5, Dec 1959, pp 150–157.

[6] Fire, P., A class of multiple-error-correcting binary codes for non-independent errors, Rept RSL-E-2, Sylvania Electric Products, Inc., Mt. View, Calif., Mar 1959.

[7] Elspas, B., and R. A. Short, A note on optimum burst-error-correcting codes, *IRE Trans. on Information Theory*, vol IT-8, Jan 1962, pp 39–42.

[8] Gross, A. J., A note on some binary group codes which correct errors in bursts of four or less, *IRE Trans. on Information Theory (Correspondence)*, vol IT-8, Oct 1962, p 384.

[9] Peterson, W. W., *Error Correcting Codes*, Cambridge, Mass.: M.I.T. Press, 1961.

[10] Stone, J. J., Multiple burst error correction, *Information and Control*, vol 4, Dec 1961, pp 324–331.

[11] Corr, F., Multiple burst detection, *Proc. IRE (Correspondence)*, vol 49, Aug 1961, p 1337.

[12] Calabi, L., Additions and multiplications of codes, Tech Memo No 11, Park Math. Labs., Carlisle, Mass., Contract AF19(604)-3471, Jun 1959.

[13] Slepian, D., Some further theory of group codes, *Bell Sys. Tech. J.*, vol 39, 1960, pp 1219–1252.

[14] Calabi, L., and H. G. Haefeli, A class of binary systematic codes correcting errors occurring at random and in bursts, *IRE Trans. on Information Theory*, vol IT-5, May 1959, pp 79–94.

[15] Calabi, L., and R. Darst, Three operations on binary systematic codes, Final Rept 3471, Park Math. Labs., Carlisle, Mass., 1961.

[16] ——, A study of the sum and the product of two codes, Scientific Rept No 3, Contract AF19(604)-7493, Park Math. Labs., Carlisle, Mass., Aug 1961.

[17] Schmandt, F. D., Single burst-error-correction capabilities of binary cyclic codes, RADC-TDR-63-301, RADC, Griffiss AFB, N. Y., Aug 1963.

[18] Wolf, J. K., and B. Elspas, Error-locating codes—a new concept in error control, *IEEE Trans. on Information Theory*, vol IT-9, Apr 1963, pp 113–117.

[19] Wolf, J. K., On an extended class of error-locating codes, (accepted for publication in *Information and Control*).

[20] Mandebrot, B., Electromagnetic turbulence in communication systems, Internat'l Conf. on Microwaves Circuit Theory and Information Theory, Tokyo, Japan, Sep 1964.

Fibonacci Codes for Synchronization Control

WILLIAM H. KAUTZ, MEMBER, IEEE

Abstract—**A new family of codes is described for representing serial binary data, subject to constraints on the maximum separation between successive changes in value ($0 \rightarrow 1$, $1 \rightarrow 0$, or both), or between successive like digits (0's, 1's, or both). These codes have application to the recording or transmission of digital data without an accompanying clock. In such cases, the clock must be regenerated during reading (receiving, decoding), and its accuracy controlled directly from the data itself.**

The codes developed for this type of synchronization are shown to be optimal, and to require a very small amount of redundancy. Their encoders and decoders are not unreasonably complex, and they can be easily extended to include simple error detection or correction for almost the same additional cost as is required for arbitrary data.

Manuscript received October 1, 1964.

The author is with the Stanford Research Institute, Menlo Park, Calif.

I. Introduction

WHENEVER a sequence of binary digits is recorded on a continuous recording medium, some means must be provided for regenerating during reading the timing signals which separate and distinguish successive digits. Several methods are known for accomplishing this synchronization, but they all are relatively costly in terms of the amount of redundancy which they devote to establish proper timing. Namely,

1) A separate clock channel may be used to synchronize one or more parallel data channels. (Equivalently, the extra channel may be an odd-parity-check channel.)

2) Three-level signals may be employed, to distinguish a 1 (+ level) and 0 (− level) from no signal (zero level). Thus, each binary digit is self-timed, at the cost of using a ternary number representation system.

3) Reading may be performed at an approximately uniform rate, so that a fixed-frequency local clock may be used. In this case, a dummy block of one or more synchronizing digits is usually placed at the end of each block of data digits, so that the local clock rate can be servoed periodically to agree with the reading rate.

In this paper we propose the use of an arrangement such as 3), except that the synchronizing information is now to be distributed throughout the data block, with the aid of some special codes for representing the data. We show that this form of synchronization requires much less redundancy than the usual "block synchronization" method mentioned in 3). For the ranges of word lengths and synchronization intervals likely to be encountered in practice, the complexity of the encoding and decoding circuitry is not unreasonable, even when the codes are augmented to include a small amount of error checking.

The codes proposed here are related, though not equivalent, to some prefix codes devised by Gilbert [1] for another type of synchronization problem.

II. Derivation of the Coding Problem

Whether the redundant information is lumped in a block at the end of the code word or is distributed throughout the code word, synchronization control is based on repeated measurements of the times at which the read signal changes from 0 to 1 or from 1 to 0. When reading a sequence of binary digits, therefore, we must require that all successive transitions in binary-signal value within each allowable code word be separated by no more than some prescribed number m of digit positions. In terms of what we will call a *string*—an unlengthenable sequence of consecutive like digits within a code word—this condition reads:

$$\left.\begin{array}{l}\text{Every } n\text{-digit code word contains no 0 strings or}\\ \text{1 strings longer than } m.\end{array}\right\} \quad (1)$$

We seek for arbitrary given m and n (where $m \leq n$) a code $C_1(m, n)$, which is a list of n-digit code words satisfying this condition.

It will be convenient to first convert this coding problem into another equivalent form. Corresponding to each n-digit code word $a = (a_n a_{n-1} \cdots a_2 a_1)$ of $C_1(m, n)$, we may form an $(n - 1)$-digit companion word $b = (b_{n-1} b_{n-2} \cdots b_2 b_1)$, defined by

$$b_j = a_j \oplus a_{j+1}, \qquad j = 1, 2, \cdots n - 1$$

where $\oplus$ designates exclusive-OR (modulo-2) addition. That is, each b-word is a kind of Boolean "derivative" of the corresponding a-word. By this reduction, each string of 0's or 1's in a is converted into a string of 0's (but shorter by one) in b. Moreover, the reduction is reversible, except for the choice of value of a_1—a reflection of the fact that if the code word a is in code C_1, then $\bar{a} = (\bar{a}_n \bar{a}_{n-1} \cdots \bar{a}_2 \bar{a}_1)$ is also in C_1. Thus, to each pair $(a, \bar{a})$ of words in $C_1(m, n)$, there corresponds a unique word b in another code $C_2(m - 1, n - 1)$, all of whose code words satisfy the condition that every $(n - 1)$-digit code word contains no 0 string longer than $m - 1$. Equivalently, then, we may seek a code $C_2(m, n)$, all of whose code words satisfy the condition:

$$\left.\begin{array}{l}\text{Every } n\text{-digit code word contains}\\ \text{no 0 strings longer than } m.\end{array}\right\} \quad (2)$$

Rather trivially, we may also speak of a third code $C_3(m, n)$, all of whose code words $c = (c_n, c_{n-1} \cdots c_2 c_1)$ are the complements of those in $C_2(m, n)$ (that is, $c = \bar{b}$), and therefore satisfy the condition:

$$\left.\begin{array}{l}\text{Every } n\text{-digit code word contains}\\ \text{no 1 strings longer than } m.\end{array}\right\} \quad (3)$$

If we designate by $N_i(m, n)$ the total number of code words in a code $C_i(m, n)$ $(i = 1, 2, 3)$, then we have immediately for the codes derived as above

$$N_1(m, n) = 2N_2(m - 1, n - 1)$$

$$N_2(m, n) = N_3(m, n)$$

The unique relations between code words in these three coding problems also guarantee that if any one of the three codes can be shown to be *maximal*—that is, if it contains the maximum possible number of code words consistent with its defining condition 1), 2), or 3)—then the other two are also maximal.

Note incidently at this point that these codes (if they can be found) would also solve directly the problem in which it is the *level* or *gain* of the reading mechanism, rather than its timing, which must be controlled by frequent readjustment in a closed-loop control device. In this case, we require that each pair of successive 0's [code $C_3(m, n)$], or each pair of successive 1's [code $C_2(m, n)$] or each pair of consecutive like digits—0's and 1's—[code $C_1(m, n)$] be separated by no more than m intervening digits of opposite value, in each n-digit code word. The indicated codes would not only provide acceptable solutions to these three problems, but if maximal in the sense defined above, they would provide minimum redundancy solutions as well.

III. Code Construction

A code family $C_3(m, n)$ for all positive integral m and n (but $m \leq n$) may be constructed as follows. Recall first that a conventional binary number constitutes a maximal k-digit code for representing any integer x between 0 and

TABLE I

FIBONACCI WEIGHTS $w_j^{(s)}$

j =	1	2	3	4	5	6	7	8	9	10	11	12
s = 1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	3	5	8	13	21	34	55	89	144	233
3	1	2	4	7	13	24	44	81	149	274	504	927
4	1	2	4	8	15	29	56	108	208	401	773	1490
5	1	2	4	8	16	31	61	120	236	464	912	1793
6	1	2	4	8	16	32	63	125	248	492	976	1936
7	1	2	4	8	16	32	64	127	253	504	1004	2000
8	·	·	·	·	·	·	·	128	255	509	1016	2028
9	·	·	·	·	·	·	·	·	256	511	1021	2040
10									·	512	1023	2045
11									·	·	1024	2047
12	1	2	4	8	16	32	64	128	256	512	1034	2048

$N_1(m, n) = 2w_n^{(m)}$ $\quad N_2(m, n) = N_3(m, n) = w_{n+1}^{(m+1)}$

$2^k - 1$; namely,

$$x = \sum_{i=1}^{k} d_i 2^{i-1}$$

in which the binary coefficients d_i are the digits of the number $x = (d_k d_{k-1} \cdots d_2 d_1)$ to the number base 2. The *weights* 2^{i-1} in this representation are simply powers of this number base ($j = 1, 2, \cdots k$).

Consider now the use of a set of diminished weights, $w_1, w_2, \cdots w_n$ for an n-digit representation $x = (c_n c_{n-1} \cdots c_2 c_1)$ in the same form; namely:

$$x = \sum_{i=1}^{n} c_i w_i. \tag{4}$$

For a given positive integer *order* s (where $s \leq n$), we will select the weights $w_j^{(s)} = w_j$ to satisfy[1]

$$\left.\begin{aligned} w_j &= 2^{j-1} && 1 \leq j \leq s \\ w_j &= w_{j-1} + w_{j-2} + \cdots + w_{j-s} && s < j \end{aligned}\right\}. \tag{5}$$

That is, the first s weights are the same as in the conventional binary case, but each weight after the sth weight is the sum of the previous s weights.[2]

[1] The superscript on $w_j^{(s)}{}_j$ will be dropped whenever it is clear which value of the order s is intended.

[2] We mention at this point the possible use of a number system with non-integral base B, hence weights $w_j = B^{j-1}$, which also satisfy the recurrent part of (5) for all integral j; hence, $B^s = B^{s-1} + B^{s-2} + \cdots + B + 1$. However, number representations in this system are unnecessarily lengthy (e.g., 9 = 10010.0101 for $s = 2$, $B = 1.618$), and the relationship between these code words and the power-of-two-weighted code words is less direct than that derived for the codes developed in this section.

For $s = 2$, for example, we get the weight sequence

$$w_1 = 1, w_2 = 2, w_3 = 3,$$
$$w_4 = 5, w_5 = 8, w_6 = 13, \text{ etc.}$$

This is the sequence of well-known Fibonacci numbers [2]. For arbitrary $s > 0$, we may call the sequence of weights defined by (5) *(generalized) Fibonacci weights* of order s.

Table I lists the values of these weights for a range of values of j and s. Some aids to the calculation of these weights and of others outside of the range of the Table are presented in the Appendix. Observe for the moment only that the terms w_j form an increasing sequence

$$w_j > w_{j-1} \tag{6}$$

for any fixed s.

We now show by a constructive encoding process that, *for fixed s, and for every integer x between 0 and $w_{n+1} - 1$, there exists a unique n-digit, Fibonacci-weighted binary representation $x = (c_n c_{n-1} \cdots c_2 c_1)$ which satisfies Condition 3) for a code $C_3(s - 1, n)$.*

This construction is inductive, i.e., iterative, and generates the representation of x, most significant digit first, as follows. Let $y_n = x$, and then form successively the numbers $y_{n-1}, y_{n-2}, \cdots, y_j, \cdots, y_1$ according to:

$$y_{j-1} = y_j - c_j w_j$$

where

$$c_j = 1 \quad \text{iff} \quad w_j \leq y_j < w_{j+1}. \tag{7}$$

That is, the number x is successively diminished by whichever weights of the sequence $w_n, w_{n-1}, \cdots w_1$, taken in

this order, do not produce a negative result. If the weight w_i is actually substracted from the running difference, then $c_i = 1$; if not, then $c_i = 0$. For example, if $s = 2$, $n = 6$, and $x = 19$, then this construction yields from the weight sequence 13, 8, 5, 3, 2, 1 (listed above) the representation $19 = (101001)$, as follows:

$$\begin{aligned} 19 - 13 &= 6 & c_6 &= 1 \\ 6 - 8 &< 0 & c_5 &= 0 \\ 6 - 5 &= 1 & c_4 &= 1 \\ 1 - 3 &< 0 & c_3 &= 0 \\ 1 - 2 &< 0 & c_2 &= 0 \\ 1 - 1 &= 0 & c_1 &= 1. \end{aligned}$$

Note that this same construction is the one frequently used for expressing a decimal number x in power-or-two-weighted binary form. For this same example, for which $19 = (10011)$

$$\begin{aligned} 19 - 16 &= 3 & d_5 &= 1 \\ 3 - 8 &< 0 & d_4 &= 0 \\ 3 - 4 &< 0 & d_3 &= 0 \\ 3 - 2 &= 1 & d_2 &= 1 \\ 1 - 1 &= 0 & d_1 &= 1. \end{aligned}$$

The uniqueness of the representation obtained from the construction (7) follows directly from (4), if only it can be shown that the construction can always be carried to completion with a zero remainder. Assume this to be the case for every x in the range $0 \leq x < w_j$. (It is obviously so for $j = 1$.) Then for any x in the range $w_j \leq x < w_{j+1}$, the construction (7) yields $c_n = c_{n-1} = \cdots = c_{j+1} = 0$ and $c_j = 1$, since all of the differences $x - w_n$, $x - w_{n-1}$, $\cdots$ $x - w_{j+1}$, $x - w_j$ are negative except the last one, by the inequality (6). Consider now the residue $x - w_j$. The defining equation (5) for the weights w_j may be expressed in the form (by subtracting (5) from itself, with j replaced by $j + 1$):

$$\begin{aligned} w_{j+1} &= 2w_j & 1 \leq j \leq s \\ w_{j+1} &= 2w_j - w_{j-s} & s < j \end{aligned} \tag{8}$$

so that $w_{j+1} \leq 2w_j$; thus, this residue $x - w_j$ is bounded according to

$$x - w_j < w_{j+1} - w_j \leq w_j$$

By the inductive hypothesis, the construction (7) may therefore be carried to completion with a zero remainder, starting with this residue.

The induction is valid up to $j = n$, so we have shown that for every integer x in range $0 \leq x < w_{n+1}$ there exists a unique representation $x = (c_n c_{n-1} \cdots c_2 c_1)$.

It remains to show that no more than $s - 1$ consecutive c_i can have the value 1. By (4), any such sequence of s c-values, $c_i = c_{i-1} = \cdots = c_{i-s+1} = 1$ $(s \leq i \leq n)$, would contribute to the weighted sum for x an amount

$$w_i + w_{i-1} + \cdots + w_{i-s+1}$$

which by (5) equals the next larger weight w_{i+1}. Take the leftmost such string. If $i < n$, then $c_{i+1} = 0$, and the string

$$\cdots \overbrace{011 \cdots 1}^{s} \cdots$$
$$\uparrow$$
$$c_i$$

could be replaced by

$$\cdots \overbrace{100 \cdots 0}^{s} \cdots$$
$$\uparrow$$
$$c_i$$

without changing the value of the sum (4). The same substitution can be made for any remaining strings of s 1's to the right. However, since the construction (7) prefers a 1 over a 0 in each digit position, working from the left, the former representation containing s consecutive 1's cannot arise. If $i = n$, then we would have $x \geq w_{n+1}$, which also could not arise, since x was assumed to fall in the range $0 \leq x < w_{n+1}$. The particular Fibonacci-weighted representation which is generated by construction (7) therefore satisfies condition 3) for a $C_3(m, n)$ code, with $m = s - 1$, and the assertion of this section has been proved.

IV. Maximality of Fibonacci-Weighted Codes

The uniqueness of the code representations of each of the w_{n+1} possible integral values of x in the range $0 \leq x < w_{n+1}$, for the $C_3(m, n)$ code just described, guarantees that the code contains at least

$$N_3(m, n) = w_{n+1}^{(m+1)}$$

code words. We now want to show that no $C_3(m, n)$ code can contain more than this number of code words, so that this Fibonacci-weighted code is maximal.

By the equivalences developed in section II, the related $C_2(m, n)$ and $C_1(m, n)$ codes derivable from this code are maximal if, and only if, $C_3(m, n)$ is itself maximal. It will be easier to derive this upper bound in terms of the first code $C_1(m, n)$. We then need to show that, for given m and n, the number $N_1(m, n)$ of possible n-digit binary words containing no 0 strings or 1 strings longer than m cannot exceed $2w_n^{(m)}$. To this end, observe that any complementary pair of words, such as

$$\left.\begin{matrix} 0100101000 \\ 1011010111 \end{matrix}\right\}$$

can be represented uniquely and unambiguously by the ordered additive partition of n into its string lengths:

$$10 = 1 + 1 + 2 + 1 + 1 + 1 + 3.$$

Riordan [3] has shown that the number of such ordered additive partitions of n (which he calls *compositions of n*) having no constituent integer greater than m is equal to the number which we have defined in (5) as $w_n^{(m)}$. That is,

$$N_1(m, n) \leq 2w_n^{(m)}.$$

Therefore, all three codes are maximal.

Henceforth, we will use the designations $C_1(m, n)$, $C_2(m, n)$, and $C_3(m, n)$ for these Fibonacci codes which are generated from construction (7) of the Section III.

V. Encoding and Decoding

The central encoding and decoding processes may be discussed in terms of the code $C_3(m, n)$, since the conversion between its code words and those of $C_2(m, n)$ and $C_1(m, n)$ is quite simple, both in concept and implementation.

The tasks of encoding and decoding are essentially the same, the basic process being one of converting a binary number representation from one set of weights to another:

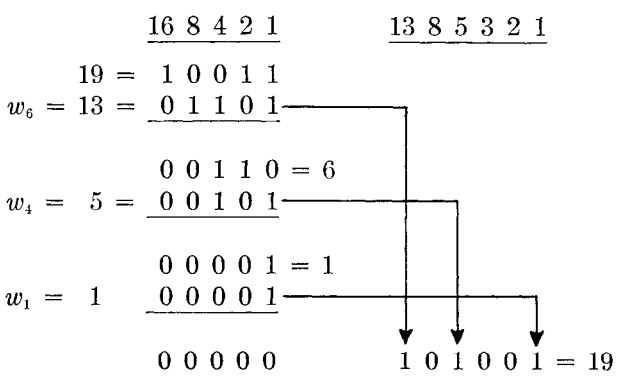
$$x = (d_k d_{k-1} \cdots d_2 d_1) \rightleftarrows x = (c_n c_{n-1} \cdots c_2 c_1)$$

power-of-two weights Fibonacci weights.

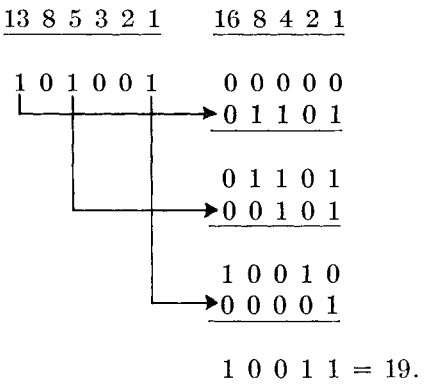
Unfortunately, however, such conversions require some sort of arithmetic computation, which might be carried out more easily in one of the number systems than the other. Consequently, the preferred encoder and decoder might employ entirely different conversion algorithms.

If the arithmetic is to be performed in the familiar power-of-two-weighted number system, then one can perform encoding according to the construction process expressed in (7), and decoding by (4). In each case, it is only necessary to express, that is, to have available in the encoder and decoder, each Fibonacci weight w_i as a k-digit binary number. For $x = 19$ and $s = 2$ (our earlier example), these processes take the form

Encoding:



Decoding:



Alternatively, the arithmetic might be performed in the Fibonacci-weighted system, in which case the use of these two equations should be interchanged: (4) for encoding, and (7) for decoding. However, addition of Fibonacci-weighted binary numbers is not so simple. In the power-of-two-weighted system, an overflow in digit position j (whose weight equals 2^{j-1}) contributes an amount $2 \cdot 2^{j-1} = 2^j$ to the sum; this overflow is added in by passing a 1 as a carry digit to the *left*, to digit position $j + 1$ (whose weight equals 2^j). In the Fibonacci-weighted system, the identity (8), which may be expressed as

$$\left.\begin{aligned} 2w_j &= w_{j+1} && 1 \leq j \leq s \\ 2w_j &= w_{j+1} + w_{j-s} && s < j \end{aligned}\right\}$$

indicates that the amount $2w_j$ contributed by an overflow in digit position j must now be passed as a carry not only to the *left* to digit position $(j + 1)$, but also the *right* (if such position falls within the representation) to digit position $(j - s)$.

This double-carry seriously complicates the circuitry of a parallel adder, and its bidirectionality renders the conversion of the adder from a parallel to a simple serial form practically impossible. Moreover, the result of such Fibonacci addition must in general be further corrected to clear out to the left all 1 strings longer than $s - 1$. Consequently, power-of-two-based arithmetic, as first described, is much preferred.

It is pertinent to inquire whether or not there exists a conversion procedure for either encoding or decoding, or both, in which the coded digits may be generated one at a time, in synchronism with the reception of successive source digits. Inspection of the power-of-two-weighted equivalents of the Fibonacci weights and the Fibonacci-weighted equivalents of the power-of-two weights reveals that, in general, neither the most significant nor the least significant digit of a resultant word can be known until *all* digits of the source word have been received and appropriately added. Thus, the circuit which performs conversion must contain signal propagation paths which

are both left-going and right-going within the entire length of the source word. Reduction of the coder to a purely serial and instantaneous mode of operation (as in a conventional binary adder) is, therefore, basically impossible.

Both of the code conversion procedures described above require that the power-of-two-weighted representations of the n Fibonacci weights be available during the conversion process. Thus, these weights must either be stored in an auxiliary memory, or be generated by local circuitry. The recurrence relations (5) and (8) indicate that each of these weights may be generated easily from previous values, requiring the storage of only s consecutive weights, regardless of how large n might be.

Regarding the relation between the parameters k and n, it is a simple matter to choose the parameter n just large enough so that $2^k \leq N_3(m, n) = w_{n+1}^{(m+1)}$; that is, to choose n so that

$$w_n < 2^k \leq w_{n+1}.$$

Inspection of Table I reveals that, except for rather small values of $m = s - 1$, even very long codes require a number $n - k$ of redundant digits equal to only one or two. (The dotted line, whose position is derived in the Appendix, encloses the region of Table I within which $n - k = 1$.)

This situation may be compared with the block synchronization method discussed in Section I. To avoid long 1 strings under Condition (3), for example, a redundant 0 must be inserted after every $(m - 1)$th digit of an arbitrary k-digit data word. Thus, a total of about $k/(m - 1)$ redundant digits are required for block synchronization. This number is generally much larger than the value of $n - k$ required for Fibonacci codes.

Code words of code $C_2(m, n)$ may now be formed by merely complementing corresponding code words of code $C_3(m, n)$. Code words of code $C_1(m, n)$ may be formed by "integrating" the corresponding code words of $C_2(m - 1, n - 1)$ (the inverse of the "derivative" process described in Section II), with $a_1 = d_1$. Alternatively, they can be generated by using a set of weights which (except for the first, which must equal unity) are the doubles of those employed for $C_2(m-1, n-1)$ and $C_3(m-1, n-1)$:

$$1, 2w_1^{(m)}, 2w_2^{(m)}, \cdots 2w_{n-1}^{(m)}.$$

In this case the construction (7) must be modified somewhat to use a more complex rule of preference for 1's and 0's, instead of a simple preference for 1's.

VI. Augmentation of Fibonacci Codes to Include Error Checking

The effects of binary errors on the code words is twofold. First, these errors can cause the peak string lengths to be increased beyond the design value m specified in the noise-free case. For example, an error in the central 0 between two 1 strings such as occur in $\cdots \underbrace{011 \cdots}_{m} \underbrace{1011 \cdots}_{m} 10 \cdots$ can increase the maximum length of 1 strings in a code word from m to $2m - 1$. Second, these errors misrepresent the data, so that error checking must be employed if accuracy is to be retained. The fact that the errors may be correctible does not in any way compensate for the first effect, since the guaranteed separations of 0's, etc., are needed during reading or demodulation, not after the code words have been read, checked, and corrected.

We will assume here that the possible presence of errors has already been taken into account in the original specification of the required value of m. Admittedly, some economy might be achieved by using a code which has a smaller value of the maximum separation m, but which satisfies such additional conditions on the *minimum* separation that the effective value of m is made independent of these errors. The potential savings in redundancy and in encoding and decoding equipment which might result from this alternative are felt to be small. This would appear to be particularly the case when the ultimate statistical performance criteria, rather than the criteria of completely error-free behavior up to a certain noise level, are applied to the system.

With this assumption, we may therefore neglect the effect of the noise in increasing the effective value of m, and concern ourselves only with the error-checking process itself.

In general, the error checking need not be applied over the Fibonacci code words themselves, but could be applied to portions of these code words, or to blocks of several such code words at a time. Aside from the necessity of interspersing buffer digits between code words in the latter case (so that strings at the ends of successive code words cannot combine to form strings of excessive length), these other alternatives contribute no new encoding problems. Therefore, we consider here only the most elementary case—namely, that checking redundancy in the form of ρ extra digits is to be added directly to the n-digit Fibonacci code words themselves, to yield $(n + \rho)$-digit, error-checked code words.

The number of possible combinations of error types, number of errors, and separation conditions is so large that we will limit the present investigation to a brief consideration of a few particular and exemplary ways in which error checking can be applied to Fibonacci code words for code $C_3(m, n)$. The extension to the other Fibonacci codes introduces no fundamental difficulties.

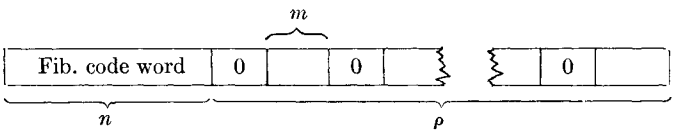
1) The simplest method of error checking is to treat the Fibonacci code words as binary data in the usual sense, neglecting for the moment the digit-separation properties of these code words. Any linear error-correcting code [4] may be used to generate a redundant subword, which is to be affixed to the end of the Fibonacci word. This redundant subword must then be modified to satisfy

the separation condition; for example, buffer 0's may be inserted at the beginning of the subword, and after every mth position in the subword, in order to break up 1 strings longer than m. For single error correction, for example, the Hamming code requires exactly $R = 1 + [\log_2 [1 + n + \log_2 n]]$ redundant digits, (where the brackets [ ] denote the integral part of the quantity within). After buffer 0's are inserted, this number increases from R to:

$$\rho = R + 1 + \left[\frac{R - 1}{m - 1}\right]$$

Normally, $\rho \ll n$ and $m \ll n$, so that the required number of buffer 0's is relatively small.

For this construction, the error-checked code word takes the form:



The extra 0's could normally be inserted and removed with only a small amount of additional circuitry in the encoder and decoder, respectively. Thus, one may employ cyclic error-correcting codes for error checking, and retain their considerable advantages over other types of codes: simple encoders and decoders, a minimum or near-minimum number of redundant digits for a prescribed degree of error checking, and considerable versatility in the types and patterns of errors which can be handled by the same code: isolated or burst, detection or correction [4].

2) All $0 \rightarrow 1$ errors, in any pattern and number, may be detected with a modification of the *Berger* error-detection code [5]. In the Berger code, each n-digit code word has affixed to it a $(1 + [\log_2 n])$-digit check word, which is the power-of-two-weighted binary representation of the total number of 0's in the code word. For example, code word 1011001 becomes 1011001011. This amount of redundancy is set by the fact that the total number of 0's may range from 0 to n.

Actually, one may use for the check word *any* positive-weighted binary representation of the total number of 0's, such as the Fibonacci-weighted representation already employed for the code word itself. Moreover, for code $C_3(m, n)$, the total number of 0's now ranges from $[n/(m + 1)]$ to n.

Allowing for a buffer 0 between the code and check words, the required number ρ of redundant digits may be readily determined to be the smallest integer which satisfies

$$w_\rho^{(m+1)} \geq n + 1 - \left[\frac{n}{m + 1}\right].$$

The check word is simply the corresponding Fibonacci-encoded representation of the total number of 0's in the code word in excess of the minimum possible number of 0's, $[n/(m + 1)]$.

This code provides a relatively economical form of detection of all $0 \rightarrow 1$ errors. It is easily modified to handle $1 \rightarrow 0$ errors instead, should this case be desired.

3) In what is called by Peterson [6] an *"AN" code*, originally developed by Brown [7] and Diamond [8], an n-digit binary number (code word) y is encoded by multiplying it by a constant integer γ, to obtain a new binary number γy having $n + \rho$ digits. Decoding is performed by dividing the (possibly erroneous) number γy by γ; this division will yield the number y, with a zero remainder if there has been no error, or possibly with some other remainder if an error has appeared in any position of the $(n + \rho)$ digit code word. In fact, a $0 \rightarrow 1$ or $1 \rightarrow 0$ error in digit position j will add or subtract, respectively, to γy an amount equal to the jth weight: 2^{j-1} in a power-of-two-weighted number system, and w_j in a Fibonacci-weighted system. For single error correction, therefore, we require that the number γ be selected large enough so that all of the weights used in the binary representation of γy, as well as the negatives of these weights, have distinct and nonzero remainders when divided by γ. If only $0 \rightarrow 1$ errors are to be corrected, then the negative weights need not be considered. If only detection is needed, then the remainders must be nonzero, but not necessarily distinct.

This encoding and decoding principle is valid whether the weights are powers of two or are Fibonacci weights, but the multiplication and division by γ are awkward in the Fibonacci case. To circumvent this difficulty, the order of conversion with the Fibonacci and "AN" codes may be reversed. That is, encoding may be carried out by first multiplying the original k-digit binary word by γ, and then converting this word to Fibonacci weights; decoding is performed by reconversion from Fibonacci to power-of-two weights before dividing by γ. The final code word should now have a number $n + \rho$ of digits just large enough so that

$$w_{n+\rho+1}^{(m+1)} > \gamma(2^k - 1).$$

The constant γ must still be selected so that all $n + \rho$ weights and their negatives have distinct nonzero remainders when divided by γ. The determination of a suitable γ may involve considerable calculation, but it need be done only once for each pair of values of m and k. For example, for $m = 2$, $k = 8$, and the correction of single $0 \rightarrow 1$ errors only, we may use $\gamma = 26$, $n + \rho = 15$, since

$$w_{16}^{(3)} = 10609 > 26 \cdot (2^8 - 1)$$

and since all of the weights $w_1^{(3)}, w_2^{(3)}, \cdots w_{15}^{(3)}$ have distinct nonzero remainders on division by 26. A lesser fractional redundancy may be expected for larger values of k.

Note that the data and check digits in these codes are not separately identifiable. However, no buffer digit is required in the middle of the code word.

The "AN codes" were originally developed for protection against errors arising during arithmetic operations. Even with Fibonacci weights, they still provide protection of this type, although the difficulties of Fibonacci addition noted previously make such application unlikely.

VII. Conclusions

We have shown with a new family of codes how the efficiency of binary data transmission or recording without an accompanying clock can be increased appreciably, in comparison with the systems in current use. This is done by arranging each valid code word to have a prescribable density of signal changes, so that the phase of the local clock (strobe) at the receiver or reading device can be controlled as accurately as necessary. The increase in efficiency is measured in terms of the number of redundant digits required. However, it is shown that the encoding and decoding apparatus is not unreasonably expensive, even when some common types of error detection or correction are included.

The existence of these codes therefore enlarges the available repertory of different channels and noise types against which error protection may be provided by coding means.

These codes should most likely find application in the recording of single-track digital data on magnetic, photographic, or similar continuous media, where the reading rate can be controlled in open-loop fashion only within a few per cent. A value of m between 5 and 20 should then be adequate to servo the reading clock to within a small fraction of one clock period. The block length n is most likely determined by the form of the source data to be recorded, and secondarily by the error rate.

Extensions of these codes which might well be investigated are 1) more efficient Fibonacci type codes with error correction, particularly burst error correction, 2) codes in which the minimum as well as the maximum string length is limited, to avoid the need for the assumption made at the beginning of Section VI and 3) codes with simpler encoders and decoders, even though the codes are somewhat more redundant. These last codes might be generated by attempting to modify or simplify the process of addition of Fibonacci-weighted code words.

Acknowledgment

For time and facilities in the preparation of this paper, the author is indebted to the Technical University of Denmark, Copenhagen, to Stanford Research Institute, Menlo Park, Calif., and particularly to Compagnie des Machines Bull, Paris, at whose premises the major part of the work was conducted. He is especially grateful to H. Azuelos and M. Nadler of this organization for their direct collaboration and other valuable assistance.

Appendix

Calculation of Fibonacci Weights

While one may use the recurrence relations (5) or (8) to calculate recursively any desired weight $w_j^{(s)}$, it would be helpful to be able to estimate directly a particular weight without calculating all previous weights of the same order. To this end we derive an approximate expression for $w_j^{(s)}$, which incidentally indicates the asymptotic behavior of the Fibonacci code parameters.

Our development parallels somewhat that of Gilbert [1]. Riordan [3] provides a generating function for $w_j^{(s)}$, namely

$$W^{(s)}(t) = \sum_{j=0}^{\infty} W_j^{(s)} t^j = \frac{t(1 - t^s)}{1 - t(2 - t^s)}.$$

That is, the jth weight is given by the coefficient of t^j in the power-series expansion of $W^{(s)}(t)$. This rational function may be expanded in partial fractions to give

$$W^{(s)}(t) = \frac{A^{(s)}(t)}{B^{(s)}(t)} = \sum_{i=1}^{s} \frac{R_i^{(s)}}{t - t_i^{(s)}}$$

in which the residue $R_i^{(s)}$ is

$$R_i^{(s)} = \left. \frac{A^{(s)}(t)}{\frac{\partial}{\partial t} B^{(s)}(t)} \right|_{t = t_i^{(s)}}$$

and the $t_i^{(s)}$ are the s roots of the denominator polynomial

$$B^{(s)}(t) = 1 - t(2 - t^s) = 0$$

(The root at $t = 1$ does not contribute a term to $W^{(s)}(t)$, since $A^{(s)}(1) = 0$ also).

This polynomial equation may be expressed in the form

$$t = \frac{1}{2 - t^s} \tag{9}$$

from which it is apparent that one root $t = t_1^{(s)} = t_1$, say, is positive, real, and somewhat less than unity. This root therefore contributes to $W^{(s)}(t)$ a summand whose power-series expansion in t has terms which *increase* with increasing j

$$\frac{R_1^{(s)}}{t - t_1} = \sum_{j=0}^{\infty} \frac{-R_1^{(s)}}{t_1} \left(\frac{t}{t_1}\right)^j.$$

All other roots are distinct and fall outside of the circle $|t| = 1$; they therefore contribute to $W^{(s)}(t)$ summands whose power-series expansions in t have terms which *decrease* with increasing j. Thus, for large j, we have:

$$W^{(s)}(t) \approx \frac{R_1^{(s)}}{t - t_1}$$

or

$$w_j^{(s)} \approx \frac{-R_1^{(s)}}{t_1^{j+1}}.$$

The residue may be evaluated:

$$R_1^{(s)} = \frac{A^{(s)}(t_1)}{B^{(s)\prime}(t_1)} = \frac{t_1(1 - t_1^s)}{(s+1)t_1^s - 2} = \frac{t_1(1 - t_1)}{2st_1 - s - 1}$$

giving the weight value

$$w_i^{(s)} \approx \frac{1 - t_1}{(s + 1 - 2st_1)t_1^i}. \tag{10}$$

But from (9) above, the root t_1 may be developed in the form of a rapidly converging series

$$t_1 = \frac{1}{2 - t_1^s} = \frac{1}{2 - \left\{\frac{1}{2 - t_1^s}\right\}} = \cdots$$

$$t_1 = \frac{1}{2}\left\{1 + \frac{1}{2^{s+1}} + \frac{s+1}{2^{2s+1}} + \cdots\right\}. \tag{11}$$

Equations (10) and (11) allow rapid calculation of the approximate value of any weight $w_i^{(s)}$. For $j = 10$, $s = 4$, for example,

$$t_1 = \tfrac{1}{2}(1 + 2^{-s} + 5\cdot 2^{-9} + \cdots) \approx 0.5206$$

$$w_{10}^{(4)} \approx \frac{0.4794}{(0.8352)(0.5206)^{10}} = 393$$

in comparison with the exact value

$$w_{10}^{(4)} = 401.$$

It is of interest to determine that portion of Table I over which a single redundant digit is required for the Fibonacci-weighted code: $n - k = 1$. Since an n-digit code has $w_{n+1}^{(s)}$ code words, this region of the table is defined by the range

$$2^k = 2^{n-1} \leq w_{n+1}^{(s)} < 2^n$$

The upper bound $w_i^{(s)} < 2^{i-1}$ is set by the first diagonal above the main diagonal, $j = s + 1$. The lower bound, $w_i^{(s)} \geq 2^{i-2}$, may be estimated as follows. For s not too small, and $j \gg s$, we may write, from (11)

$$t_1 \approx \tfrac{1}{2}(1 + 2^{-s-1})$$

so that

$$w_i^{(s)} \approx \frac{(1 - 2^{-s-1})2^{j-1}}{(1 - s2^{-s-1})(1 - 2^{-s-1})^j}.$$

Thus, the lower bound becomes

$$2^{j-2} \lesssim 2^{j-1}\{1 - (j + 1 - s)2^{-s-1} + \cdots\}$$

or $(j + 1 - s) \lesssim 2^s$. The range of values of j and s over which one redundant digit is required is outlined by a dotted line in Table I. For large s and j, it is expressed by

$$s + 1 \leq j \lesssim 2^s + s - 1.$$

This range therefore includes most of the useful part of the table.

References

[1] Gilbert, E. N., Synchronization of binary messages, *IRE Trans. on Information Theory*, vol IT-6, Sep 1960, pp 470–477.
[2] Riordan, J., *An Introduction to Combinatorial Analysis*, New York: Wiley, 1958, p 14.
[3] *Ibid.*, pp 124–125, 154–155.
[4] Peterson, W. W., *Error-Correcting Codes*, New York: Wiley, 1961, chaps 8–9.
[5] Berger, J. M., A note on error detection codes for asymmetric channels, *Information and Control*, vol 4, Mar 1961, pp 68–73.
[6] Peterson, *op. cit.*, chap 13.
[7] Brown, D. T., Error detecting and correcting binary codes for arithmetic operations, *IRE Trans. on Electronic Computers*, vol EC-9, Sep 1960, pp 333–337.
[8] Diamond, J. M., Checking Codes for Digital Computers, *Proc. IRE (Correspondence)*, vol 43, Apr 1955, pp 487–488.