A. X. Widmer
P. A. Franaszek

A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code

This paper describes a byte-oriented binary transmission code and its implementation. This code is particularly well suited for high-speed local area networks and similar data links, where the information format consists of packets, variable in length, from about a dozen up to several hundred 8-bit bytes. The proposed transmission code translates each source byte into a constrained 10-bit binary sequence which has excellent performance parameters near the theoretical limits for 8B/10B codes. The maximum run length is 5 and the maximum digital sum variation is 6. A single error in the encoded bits can, at most, generate an error burst of length 5 in the decoded domain. A very simple implementation of the code has been accomplished by partitioning the coder into 5B/6B and 3B/4B subordinate coders.

1. Introduction

This paper presents a transmission code that is well suited for high-speed local area networks and computer links. Such links require relatively simple and reliable transceivers at low cost, and a good code choice can significantly contribute to this goal. The rapidly evolving fiber optic technology is expected to penetrate this application. Except for some special situations, digital fiber optic links generally operate in binary on/off, rather than ternary mode (because of a better optical signal-to-noise margin and much simpler receiver circuits). For this reason we confine ourselves to binary codes.

A code that is free of dc, or one that has a constant dc component regardless of data patterns [1], provides many advantages for fiber optic and electromagnetic wire links. High-gain fiber optic receivers need an ac coupling stage near the front end. Also, control of transmitter level, receiver gain, and equalization are simplified, and the precision of control is improved if these can be based on the average signal power (especially at the higher data rates). DC restoration circuits are an alternate solution to the stated control problems, but the circuits tend to lose precision with increasing data rates.

Additional redundancy is required for reliable clock recovery and for the coding of special control characters, e.g., to delimit the start and end of information packets. At high data rates it is desirable to maintain a constant byte rate and to reduce the number of logic circuits operating at the signaling rate, which is sometimes set near the technology limits. These objectives can be met most readily if redundancy is added at a constant rate to each byte (in contrast to selective, pattern-dependent bit stuffing). A constant expansion factor also benefits other areas of a transmission system, such as error control, buffer design in gateways, address expansion or substitution, and clock design.

Modern communications architectures transmit information in the form of packets with a defined field structure for addresses, information, and communication and error control. The number of bits in each of these fields and in the entire packet is generally a multiple of eight bits. Buffers and relevant interfaces are also byte oriented. In such systems a byte code has the advantage of a natural affinity with the packet boundaries and the byte rate clocks and is readily implemented with lower-speed logic on the parallel side of the system. Otherwise attractive binary codes, such as the

© **Copyright** 1983 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

5B/6B code [2], do not readily mesh with a byte-oriented structure. The required adaptations in the clocking area, at interfaces, and in the packet structure are awkward and increase the high-speed gate count.

With these considerations in mind, and looking for a fairly efficient code, one would first tend to examine the possibility of an 8B/9B code. However, to realize such a code, implementation and performance parameters (other than efficiency) have to be compromised to a degree which is inordinate in comparison with the results which can be obtained with 8B/10B codes. Pursuing overall objectives similar to ours, Kiwimagi [3] proposed a 4B/5B code with fairly relaxed performance constraints. Our approach consolidates a 5B/6B and a 3B/4B code into a compound 8B/10B code. A 5B/6B and a 3B/4B coder operating separately have been described by Griffiths [2]. We modified the code tables to improve the performance parameters and to facilitate implementation; we also defined a set of synchronizing, or comma, characters, and other special non-data characters which we describe later. Finally, we built and operated a practical and remarkably simple implementation of the entire coder and decoder, and demonstrated that it can be integrated in a relatively slow technology with straightforward interfaces. This paper is an elaboration of a previous short article on the identical code [4].

Section 2 of this paper provides a description of general coding constraints, as well as a discussion of coding alternatives which led to the code structure proposed here. Section 3 defines the proposed 8B/10B code in detail. Section 4 describes the performance parameters and features of the code. Finally, Section 5 treats the implementation aspects.

2. Channel constraints and coding alternatives

• *Channel constraints*

As noted previously, the stream of signals transmitted down the channel must be constrained so that enough transitions for timing recovery and little or no dc spectral components are present.

A measure of the energy at and near dc is the *digital sum variation* or DSV [5], which is obtained as follows: Each channel symbol (corresponding to a possible signal waveform during a unit interval) is assigned an algebraic value corresponding to its dc component. The DSV is defined as the variation in the running sum of the encoded data stream, i.e., the maximum minus the minimum value. For a binary, or two-level, code, the *1* and *0* bits are generally assigned values of $+1$ and -1, respectively. In the following, the maximum DSV is denoted by the symbol v. Note that the number of levels in the running sum is $v + 1$.

Table 1 Examples of channel capacities C.

d	k	v	C
0	1	5	0.655
0	2	5	0.803
0	3	5	0.842
0	4	5	0.850

The *run length* is defined as the number of identical contiguous symbols which appear in the signal stream. For a binary code, the run length is the number of contiguous *1*s or *0*s after encoding. What is of interest is the shortest (X) and the longest (Y) run lengths that appear. These two parameters are often given in the form (d, k) where $d = X - 1$ and $k = Y - 1$. The (d, k) representation gives the minimum (d) and maximum (k) number of symbols between unequal symbols. For a (0, 3) code, for example, any symbol can be followed by no more than three contiguous identical symbols, for a maximum run length of 4. Codes designed for digital transmission usually have a parameter d of 0. The preferred codes for magnetic recording, on the other hand, usually have a parameter value d of 1 or greater; i.e., the minimum spacing between transitions is longer than a symbol interval.

• *Theoretical limits*

Coding with a (d, k) run length limit with a bound of v for the DSV is an example of coding for an input-restricted channel. Three criteria of consequence here are (1) the channel capacity C (which represents the maximum coding rate of bits per channel symbol), (2) the coder complexity, and (3) the amount of error propagation. Table 1 lists channel capacities C for a sample of (d, k, v) constraints.

A variety of formal methods exist for the construction of such codes. References [5–13] may be used as an entry into the literature. It is known that a code may always be obtained as long as the desired coding rate does not exceed the channel capacity. Moreover, even for the class of constraints discussed here, where the decoder must be state-independent [5] to avoid infinite error propagation, it has recently been proven [10] that a mapping with this property is always attainable. Thus, theoretically, it is possible to obtain a (0, 2, 5)-rate 4/5 code. However, a coder and decoder with these parameters may not be practical because coding complexity and error propagation must be considered, as well as additional requirements, such as special signaling sequences.

One way of classifying the complexity of a code is by the number of information (source) bits that must be examined

Table 2 Code complexity m for various code constraints.

d	k	v	s	w	m
0	2	5	4	5	3
0	2	5	8	10	≥ 2
0	3	5	8	10	1
0	4	5	4	5	2

Table 3 5B/6B Encoding.

Name	*ABCDEK*	*Classifications*		*D-1*	*abcdei*	*D0*	*abcdei*
		Bit encoding	*Disparity*				*Alternate*
D.0	00000 0	L04	L22′•L31′•E′	+	011000	−	100111
D.1	10000 0	L13•E′	L22′•L31′•E′	+	100010	−	011101
D.2	01000 0	L13•E′	L22′•L31′•E′	+	010010	−	101101
D.3	11000 0	L22•E′		x	110001	0	
D.4	00100 0	L13•E′	L22′•L31′•E′	+	001010	−	110101
D.5	10100 0	L22•E′		x	101001	0	
D.6	01100 0	L22•E′		x	011001	0	
D.7	11100 0		L31•D′•E′	−	111000	0	000111
D.8	00010 0	L13•E′	L22′•L31′•E′	+	000110	−	111001
D.9	10010 0	L22•E′		x	100101	0	
D.10	01010 0	L22•E′		x	010101	0	
D.11	11010 0			x	110100	0	
D.12	00110 0	L22•E′		x	001101	0	
D.13	10110 0			x	101100	0	
D.14	01110 0			x	011100	0	
D.15	11110 0	L40	L22′•L31′•E′	+	101000	−	010111
D.16	00001 0	L04, L04•E	L22′•L13′•E	−	011011	+	100100
D.17	10001 0	L13•D′•E		x	100011	0	
D.18	01001 0	L13•D′•E		x	010011	0	
D.19	11001 0			x	110010	0	
D.20	00101 0	L13•D′•E		x	001011	0	
D.21	10101 0			x	101010	0	
D.22	01101 0			x	011010	0	
D/K.23	11101 x		L22′•L13′•E	−	111010	+	000101
D.24	00011 0	L13•D•E	L13•D•E	+	001100	−	110011
D.25	10011 0			x	100110	0	
D.26	01011 0			x	010110	0	
D/K.27	11011 x		L22′•L13′•E	−	110110	+	001001
D.28	00111 0			x	001110	0	
K.28	00111 1	L22•K	K	−	001111	+	110000
D/K.29	10111 x		L22′•L13′•E	−	101110	+	010001
D/K.30	01111 x		L22′•L13′•E	−	011110	+	100001
D.31	11111 0	L40, L40•E	L22′•L13′•E	−	101011	+	010100

by the coder in choosing a word when using a bounded delay code [8, 9]. Suppose, for example, that the rate is s/w, where s bits at a time are encoded into words of length w. A parameter m may be used to represent the number of s-bit groups that need to be examined in the coding process when choosing a word for transmission. Consider the case of $m = 2$, $s = 4$, and $w = 5$. The value of m indicates that the coding process requires at each step the examination of two 4-bit groups of data. Table 2 gives the minimum values of m for various constraints and rate 4/5. Note for example that at least three 4-bit groups must be examined at each step for a (0, 2, 5) code.

The code described here lies outside the framework discussed in References [5–12] since it is composed of subblocks whose length and coding rate are not uniform. However, these variations are periodic, so the new code may be viewed as a special case of a fixed-length code with $s = 8$ and $w = 10$ for $d = 0$, $k = 4$, and $v = 6$ constraints. Construction of such a fixed-length code may be done by first obtaining a set of *principal states*, described as follows: Given a finite state machine description of the channel constraints, a principal state set for block encoding is one where, from each member of the set, there are sufficient code words terminating in states within the set. An algorithm for finding such sets and associated code words is described in [5].

A corresponding algorithm may be formulated for the partitioned-block code discussed here. The difference is that words used in the 5B/6B code must terminate in coding states for the 3B/4B code, and vice versa. Partitioned-block codes may thus be handled by a fairly straightforward extension of the theory.

The coding technique discussed above may be viewed as frameworks which deal with the structure and existence of various mappings between unconstrained data and coder output. However, the design of a code for a given application involves a number of engineering tradeoffs which are perhaps best understood by examining specific examples. To clarify the reason for the design choice, some of the alternatives that were considered are subsequently described.

- *Coding alternatives*

1. The partitioned-block code that was chosen has parameters $d = 0$, $k = 4$, and $v = 6$, with a rate of 4/5 obtained by combining a rate 5/6 and a rate 3/4 code. Error propagation is limited to five bits. This means that an isolated additive error in detection results in at most five erroneous bits in the decoded data stream. Encoding requires the examination of at most five source bits at a time.
2. Another possibility is a standard block code with no look-ahead, with parameters $d = 0$, $k = 3$, $v = 5$, $s = 8$, $w = 10$, and $m = 1$. Here eight source bits must be examined in choosing a code word. Error propagation is in general eight bits. The modest improvement in run length and DSV over Option 1 were not considered sufficient to justify the increase in error propagation and the substantially greater complexity of the coder and decoder.
3. A third possibility that was investigated is the construction of codes with words of length five. A bounded-delay or variable-length code [9] with $k = 4$, $v = 5$, $s = 4$, and

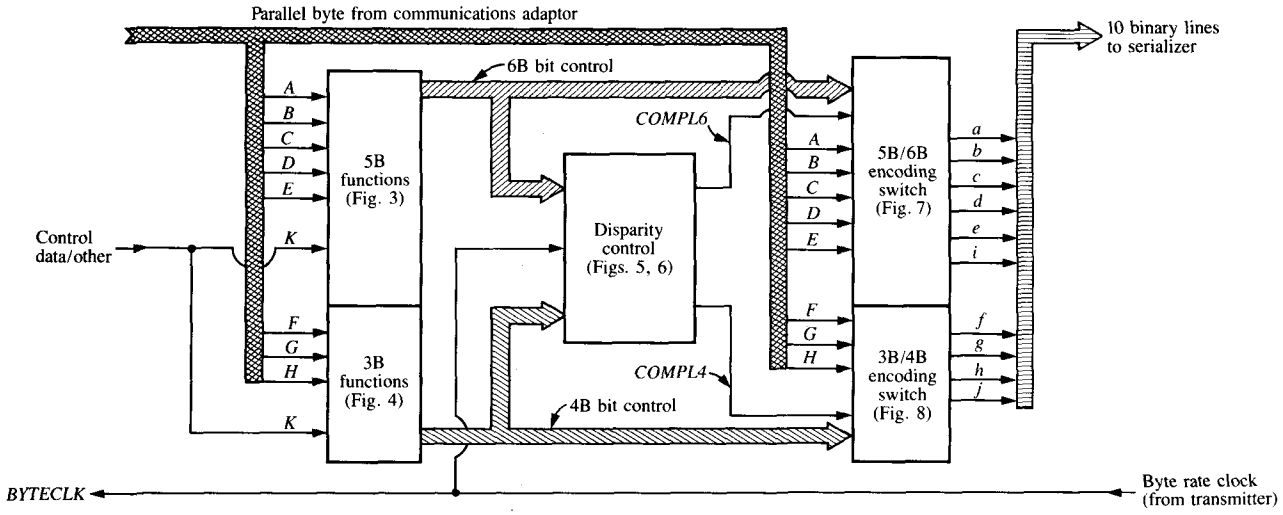

Figure 1 Block diagram of the 8B/10B encoder.

$m = 2$ turned out to be the best compromise among values of k, v, error propagation, and complexity. The code may be designed so that the maximum number of bits examined in coding is limited to 5. Error propagation is five bits (as in the code actually chosen). The amount of circuitry required for a coder and decoder is also comparable. However, this code has several disadvantages arising from the fact that there is little suitable redundancy available for the mapping of special characters. Another disadvantage arises from the variable-length look-ahead format, which results in a requirement for padding at the end of data packets.

3. The 8B/10B coding map

• *General overview*

Figure 1 shows a communications adapter interface consisting of the eight data lines *ABCDEFGH* (note the uppercase notation), a control line *K*, and a clock line *BYTECLK* operating at the byte rate. The control line *K* indicates whether the lines *A* through *H* represent data or control information.

For encoding purposes, each incoming byte is *partitioned* into two subblocks. The five binary lines *ABCDE* are encoded into the six binary lines *abcdei* (note the lowercase notation), following the directions of the 5B/6B logic functions and the disparity control. Similarly, the three bits *FGH* are encoded into *fghj*.

The *disparity* of a block of data is the difference between the number of *1*s and *0*s in the block; positive and negative disparity numbers refer to an excess of *1*s and *0*s, respectively. For both the 6B = *abcdei* and 4B = *fghj* subblocks the permitted disparity is either 0, +2, or −2. The coding rules require that the polarity of nonzero disparity blocks alternates. For this purpose, no distinction is made between 6B and 4B subblocks; i.e., a surplus of two *1*s in a 6B block can be compensated by two excess *0*s in either a 6B or a 4B block and vice versa.

Nonzero disparity code points are assigned in complementary pairs to a single source data point. The encoding functions generate one of them; if it violates the alternating polarity rule, the complete subblock is inverted in the encoding switch. Determination of disparity and polarity in the 6B encoder is followed by the corresponding operations of the 4B encoder, then the running disparity parameter is passed along for encoding of the next byte. The majority of the coded subblocks are of *zero disparity* and are, with some exceptions, independent of the running disparity; i.e., they do not have a complement.

The ten encoded lines *abcdeifghj* normally interface with the serializer; the *a*-bit must be transmitted first and *j* last.

• *Code definition*

The 8B/10B encoding is accomplished by encoding the bits *ABCDE* of the input byte into the line digits *abcdei* in a 5B/6B encoder following the coding plan and rules of Table 3, and the bits *FGH* in a 3B/4B encoder into the line digits *fghj* as shown in Table 4.

5B/6B encoding The first column in Table 3, headed by "Name," gives the 32 decimal equivalents for the input lines *ABCDE*, assuming *A* is the low-order bit and *E* the high-order bit. For regular data (D.x) the line *K* must be held at 0; a few code points can be part of special characters which are recognizable as other than data; such code points are named

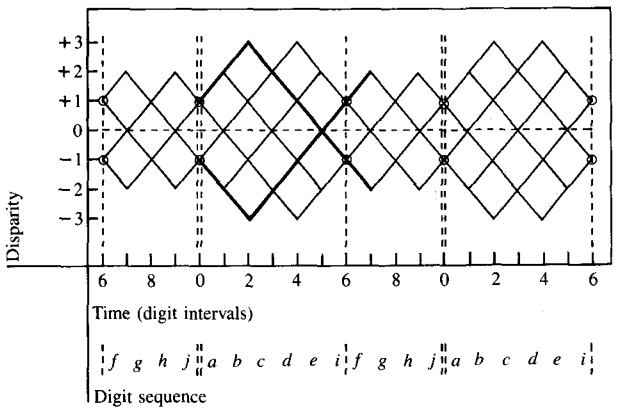

Figure 2 Disparity vs time plot.

Table 4 3B/4B Encoding.

Name	*FGH K*	*Classifications*		*D-1*	*fghj*	*D0*	*fghj*
		Bit encoding	*Disparity*				*Alternate*
D/K.x.0[a]	000 x	F′•G′•H′	F′•G′	+	0100	−	1011
D.x.1	100 0	(F≠G)•H′		x	1001	0	
D.x.2	010 0	(F≠G)•H′		x	0101	0	
D/K.x.3[a]	110 x		F•G	−	1100	0	0011
D/K.x.4[a]	001 x		F′•G′	+	0010	−	1101
D.x.5	101 0			x	1010	0	
D.x.6	011 0			x	0110	0	
D.x.P7	111 0		F•G, F•G•H	−	1110	+	0001
D/K.y.A7[b,c]	111 x	F•G•H•(S+K)	F•G, F•G•H	−	0111	+	1000
•K.28.1	100 1	(F≠G)•H′	(F≠G)•K	+	1001	0	0110
K.28.2	010 1	(F≠G)•H′	(F≠G)•K	+	0101	0	1010
K.28.5	101 1		(F≠G)•K	+	1010	0	0101
K.28.6	011 1		(F≠G)•K	+	0110	0	1001

[a] K.x is restricted to K.28.
[b] K.y is restricted to K.23, K.27, K.28, K.29, K.30.
[c] S = {e•i•(D−1=−)} OR {e′•i′•(D−1=+)}.

D/K.x or K.x and have an *x* or *1* in column *K*. To encode special characters, the *K* line must be *1*.

In the "classification" columns, *L04* means that there are no *1*s but four *0*s in *ABCD; L13* means that there are one *1* and three *0*s in *ABCD*, etc. The letter "L" indicates that this logic function or classification is part of the 5B/6B encoder. Analogous functions labeled "P" are defined for decoding (Table 6). An accent to the right of a symbol is used to represent complementation; *E′* means the complement of *E*; a dot (•) stands for the logical AND function.

In the column under the left "*abcdei*" heading there are listed all the code points which are generated directly by the 5B/6B logic functions from the *ABCDE* inputs. The coding table was designed so that a minimal number of bits must be changed on passing through the encoder, and so that the changes which are required can be classified into a few groups applicable to several code points. All the bits in Table 3 which require action by the 5B/6B logic functions (other than complementation of the complete subblock) are in bold type, assuming that the extra digit "*i*" is added with a normal value of *0*.

When the inputs meet the logical conditions listed on the left side under "bit encoding," then the bold type bits are changed to the values shown in the left "*abcdei*" column; e.g., if *L04* holds, the *b* and *c* digits are forced to *1*s, as shown for D.0 and D.16. The second entry in the "bit encoding" column for D.16 (*L04* • *E*) and D.31 (*L40* • *E*) applies to the *i*-digit. For lines with no classification entry, the *ABCDE* bits translate unchanged into *abcde* and the added *i*-bit is a *0*.

The "alternate *abcdei*" column to the right of Table 3 shows the complement for those *ABCDE* inputs which have alternate code points. Individual 6B (and 4B) subblocks are complemented in conformity with the disparity rules. At all subblock boundaries the running disparity is either +1 or −1 and never 0 (see Fig. 2).

The column "*D* − 1" indicates the required running disparity for entry of the adjacent subblock to the right. An *x* in the "*D* − 1" column means that (*D* − 1) can be + or −. In this code the polarity of the running disparity at subblock boundaries is identical to the polarity of the most recent nonzero disparity block.

As an example for encoding of the first line D.0 of Table 3: If the running disparity matches (*D* − 1) = +, the output of the encoder will be *011000*; otherwise the entire subblock is complemented to *100111*.

The "D0" column indicates the disparity of the encoded subblock to the left, which is either 0, +2, or −2. The disparities for the alternate code points on the right side of Tables 3 and 4 are exact complements of those to their left, and are not shown.

As in bit encoding, the encoder hardware determines directly from the *ABCDE* and *K* inputs the disparity of a subblock. The respective logic functions for classification of code words in terms of disparity requirements are shown in a separate column in Table 3.

In Table 3, line D.7, a pair of zero-disparity 6B subblocks (*111000* and *000111*) is assigned to a single data point with an entry disparity constraint similar to those applicable to nonzero-disparity subblocks. This coding feature reduces the

maximum digital sum variation from eight to six and, combined with an analogous rule in the 3B/4B encoder for D/K.x.3 of Table 4 (*1100* and *0011*), eliminates all sequences of run length 6 and most of those with run length 5.

The technique of assigning a pair of complementary zero-disparity subblocks to a single code point is also used uniformly for all 4B subblocks which are part of a special character, as shown in Tables 4 and 5 for K.28.1, K.28.2, K.28.3, K.28.5, and K.28.6.

3B/4B encoding Table 4 follows the conventions and notations of Table 3. In Table 4 some lines have two entries in the column for the classification of disparity; the left classification refers to the entry disparity $D - 1$, and the right one to *D0*.

The encoding of D.x.P7 (*primary 7*) and D/K.y.A7 (*alternate 7*) requires an explanation. The D/K.y.A7 code point was introduced to eliminate the run length 5 sequence in digits *eifgh*. The A7 code replaces the P7 encoding whenever

$[(e = i = 1) \cdot (D - 1 = -)]$ OR

$[(e = i = 0) \cdot (D - 1 = +)]$ OR

$(K = 1)$.

Note that whenever $K = 1$, $FGH = 111$ is always translated into $fghj = 0111$ or its complement.

The D/K.y.A7 encoding can generate a run length (RL) 5 sequence across the trailing character boundary in the *ghjab* bits; however, this sequence is preceded by a run length of only 1 in digit *f*, with one exception. If the leading character is the special character K.28.7, a RL 5 sequence across the trailing character boundary is preceded by another RL 5 sequence in *cdeif*. For the significance of these distinctions, see the section on special characters.

The zero-disparity 4B subblocks of K.28.1, K.28.2, K.28.5, and K.28.6 are handled similarly to D/K.x.3 with respect to complementation in order to generate some special characters with byte synchronizing or comma properties.

Special characters Special characters are defined here as extra code points beyond the 256 needed to encode a byte of data. They are generally used to establish byte synchronization, to mark the start and end of packets, and sometimes to signal control functions such as ABORT, RESET, SHUTOFF, IDLE, and link diagnostics. The set of twelve special characters shown in Table 5 can be generated by the coding rules defined in Tables 3 and 4. They all comply with the general coding constraints of a maximum run length of 5 and a maximum digital sum variation of 6.

Table 5 Special characters ($K = 1$).

Name	*ABCDEFGH K*	*D-1*	*abcdei*	*fghj*	*D0*	*D-1*	*abcdei*	*fghj*	*D0*
							Alternate		
K.28.0	00111 000	1	– 001111	0100	0	+ 110000	1011		0
K.28.1*	00111 100	1	– **001111**	1001	+	+ **110000**	0110		–
K.28.2	00111 010	1	– 001111	0101	+	+ 110000	1010		–
K.28.3	00111 110	1	– 001111	0011	+	+ 110000	1100		–
K.28.4	00111 001	1	– 001111	0010	0	+ 110000	1101		0
K.28.5*	*00111 101*	1	– **001111**	1010	+	+ **110000**	0101		–
K.28.6	00111 011	1	– 001111	0110	+	+ 110000	1001		–
K.28.7*[d]	00111 111	1	– **001111**	1000	0	+ **110000**	0111		0
K.23.7	11101 111	1	– 111010	1000	0	+ 000101	0111		0
K.27.7	11011 111	1	– 110110	1000	0	+ 001001	0111		0
K.29.7	10111 111	1	– 101110	1000	0	+ 010001	0111		0
K.30.7	01111 111	1	– 011110	1000	0	+ 100001	0111		0

* Singular comma (for byte synchronization).

[d] K.28.7 must not be contiguous to another K.28.7.

Table 6 6B/5B Decoding.

Name	*abcdei*	*Decoding class*	*Disparity class*	*ABCDEK*	*D-1*	*D0*
D.0	011000	P22•b•c•(e=i)	P22•e′•i′	00000	0	+ –
D.0	100111	P22•b′•c′•(e=i)	P22•e•i	**00000**	0	– +
D.1	100010	P13•i′	P13•i′	10000	0	+ –
D.1	011101	P31•i	P31•i	***10000***	0	– +
D.2	010010	P13•i′	P13•i′	01000	0	+ –
D.2	*101101*	P31•i	P31•i	**01000**	0	– +
D.3	110001			11000	0	x 0
D.4	001010	P13•i′	P13•i′	00100	0	+ –
D.4	110101	P31•i	P31•i	**00100**	0	– +
D.5	101001			10100	0	x 0
D.6	011001			01100	0	x 0
D.7	111000		P31•d′•e′•i′	11100	0	– 0
D.7	000111	P13•d•e•i	P13•d•e•i	**11100**	0	+ 0
D.8	000110	P13•i′	P13•i′	00010	0	+ –
D.8	111001	P31•i	P31•i	***00010***	0	– +
D.9	100101			10010	0	x 0
D.10	*010101*			01010	0	x 0
D.11	110100			11010	0	x 0
D.12	001101			00110	0	x 0
D.13	101100			10110	0	x 0
D.14	011100			01110	0	x 0
D.15	101000	P22•a•c•(e=i)	P22•e′•i′	11110	0	+ –
D.15	010111	P22•a′•c′•(e=i)	P22•e•i	**11110**	0	– +
D.16	011011	P22•b•c•(e=i)	P22•e•i	***00001***	0	– +
D.16	100100	P22•b′•c′•(e=i)	P22•e′•i′	**00001**	0	+ –
D.17	*100011*			10001	0	x 0
D.18	010011			01001	0	x 0
D.19	110010			11001	0	x 0
D.20	001011			00101	0	x 0
D.21	101010			10101	0	x 0
D.22	011010			01101	0	x 0
D/K.23	111010		P31•e	11101	x	– +
D/K.23	000101	P13•e′	P13•e′	***11101***	x	+ –
D.24	001100	a′•b′•e′•i′	P22•e′•i′	00011	0	+ –
D.24	110011	a•b•e•i	P22•e•i	**00011**	0	– +
D.25	100110			10011	0	x 0
D.26	*010110*			01011	0	x 0
D/K.27	110110		P31•e	11011	x	– +
D/K.27	001001	P13•e′	P13•e′	**11011**	x	+ –
D.28	001110			00111	0	x 0
K.28	001111	c•d•e•i	P22•e•i	00111	1	– +
K.28	110000	c′•d′•e′•i′	P22•e′•i′	**00111**	1	+ –
D/K.29	101110		P31•e	10111	x	– +
D/K.29	010001	P13•e′	P13•e′	***10111***	x	+ –
D/K.30	011110		P31•e	01111	x	– +
D/K.30	*100001*	P13•e′	P13•e′	**01111**	x	+ –
D.31	101011	P22•a•c•(e=i)	P22•e•i	11111	0	– +
D.31	010100	P22•a′•c′•(e=i)	P22•e′•i′	11111	0	+ –

The first group of eight special characters K.28.x, Table 5, can be recognized as other than data by observing that *abcdei* = *001111* or *110000*. In data we never have $c = d = e = i$.

The second group of four special characters K.x.7, Table 5, is characterized by *eifghj* = *101000* or *010111*. The distinction from data is the encoding of *FGH* into *0111* or *1000*, where *1110* or *0001* would be used for data.

Commas and packet delimiters A *comma* [14] indicates the proper byte boundaries and can be used for instantaneous acquisition or verification of byte synchronization. To be useful, the comma sequence must be singular and must occur with a uniform alignment relative to the byte boundaries. In the absence of errors, the comma must not occur in any other bit positions, neither within characters nor through overlap between characters. Three characters in this code (K.28.1, K.28.5, K.28.7) have comma properties. They are marked with an asterisk in Table 5, and the singular sequence is printed in bold type. These three characters are also most suitable delimiters to mark the start and end of an information packet.

The *singular comma* in this code is a sequence of run length (RL) 2 or more ending with digit *b* followed by a RL 5 sequence in digits *cdeif*, where this second sequence is not permitted to be the RL ≥ 2 sequence of another comma. In other words, if two or three RL ≥ 2/RL 5 sequences overlap, only the first and third are recognized as commas. This rule is necessary because in some situations the K.28.7 comma is followed by another RL 5 sequence in digits *ghjab*.

A sequence of contiguous K.28.7 characters would generate alternating RL 5 sequences of *1*s and *0*s, which is not useful for character synchronization and is poor for bit-clock synchronization. For this reason, no adjacent K.28.7 characters are allowed. Despite this restriction, the K.28.7 comma is often preferred over the other two because in the synchronized state *no single error can generate a valid K.28.7 from data.*

Idle sequence We must distinguish between communications links which maintain byte synchronization from packet to packet and those which do not. It is desirable to maintain a high transition density during the idle state to ease the acquisition of bit synchronism by the receiver clock. Therefore, when byte synchronism is not maintained between packets, a sequence of alternating *1*s and *0*s is suitable and can be generated by the encoder with either a D.21.5 or a D.10.2 input.

For an exact balance between the number of *1*s and *0*s, the ending disparity must be carried forward and the new packet should start only after any even-numbered encoded digit. However, a start on an odd-numbered digit will degrade the noise margin by a negligible amount if the packets are several bytes long, and if the running disparity of the idle sequence is constrained to -1, 0, or $+1$, by starting the sequence with a *0* or a *1*, depending on whether the previous packet has ended with positive or negative disparity, respectively (see Fig. 2).

If byte synchronism is carried forward from packet to packet, it is useful to fill the packet gap with characters other than data. A steady stream of K.23.7, K.27.7 or K.29.7 characters generates sixty transitions for every 100-digit interval. Each character is recognizable as other than data, but byte synchronism cannot be acquired from this idle sequence unless a comma character is inserted every now and then. As an alternative, a sequence of contiguous K.28.5 commas generates a transition for 50% of the digit intervals.

4. Code evaluation

• *Digital sum variation and disparity*

The maximum DSV between arbitrary points in this code is 6. Sometimes the DSV is quoted with reference to specific bit positions, such as the end of a character, and a lower figure usually results. The maximum DSV between any two *i/f* or *j/a* bit boundaries is 2.

As mentioned earlier, the term *disparity* designates the difference between the number of *1* and *0* bits in a defined block of digits, or the instantaneous deviation from the long-term average value of the running digital sum. All 6B and 4B subblocks individually, and the complete 10-bit characters, have a disparity of either 0 or ± 2; i.e., each valid character in the 10B alphabet either has five *1*s and five *0*s or six *1*s and four *0*s or four *1*s and six *0*s.

It is instructive to plot the disparity as a function of time or digit intervals as is done in Fig. 2, where each *1* bit is marked by a line segment extending over one digit interval and rising at a 45° angle; conversely, a *0* bit is represented by a falling line. As an example, starting at the circled $+1$ disparity value on the leftmost *j/a* bit boundary, a *110100* digit pattern would lead along the upper contour to disparity $= +1$ at the *i/f* bit boundary. All data characters and special characters of Tables 3, 4, and 5 are represented in Fig. 2. The bold lines depict the comma sequence in its two complementary manifestations.

From the diagram it is immediately evident that the *maximum DSV* between arbitrary points is 6. Since the disparity is bounded, the code is free of any dc component.

• *Run length*

An examination of Fig. 2 shows that RL 5 sequences are only possible starting with bit positions *c*, *e*, *g*, and *j*. However, inspection of Table 3 shows that the 6B alphabet does not include any code points for which $a = b = c = d$, and that $c = d = e = i$ is confined to the K.28.x special characters. These added constraints render impossible any RL 5

sequence starting with j and limit those starting with c to the comma sequences listed in Table 5. The RL 5 data sequence starting with e is eliminated through the alternate code point D.x.A7 of Table 4, which in turn is the sole generator of the RL 5 sequence starting with g. Note, however, that this sequence is always preceded by RL 1 except when overlapping with the K.28.7 comma.

Beyond the extremes in the run length, the clustering of sequences of maximum or near-maximum run length is also of interest. The data sequence with the lowest transition density, which can be maintained indefinitely, has an average of 30 transitions per 100-digit interval.

• *Error detection*

The first aspect to be considered is the exploitation of the code redundancy for error checking. The second area to be explored is how errors in the line code interact and affect error detection by cyclic redundancy checks applied to the unencoded digits.

Error checking using redundancy of 8B/10B code We are not concerned here with the correctness of individual characters, only with the validity of entire packets. Each packet starts and ends with a delimiter. For packets defined in this way, the start and end characters each contain at least one nonzero-disparity subblock, which prevents disparity violations arising from errors to be carried forward into another packet. The difference between the number of *1*s and *0*s in a valid packet is 0, +2, or −2. In general, unless a delimiter has been complemented exactly by noise, any error pattern in a packet for which the number of erroneous *1*s is not equal to the number of erroneous *0*s can be detected. Beyond this general rule, many other errors are detected as well, because of the alternating disparity rules, especially if the error digits are far apart. Still other errors do not change the number of *1*s or *0*s but generate illegal characters outside the defined alphabet and thus are detected.

To implement the error detection scheme, five checks are required: (1) all 6B and 4B subblocks of a packet have to have either 0, +2, or −2 disparity, (2) nonzero disparity blocks must alternate in polarity, (3) D.7 and D/K.x.3 must follow the disparity rules, (4) a valid packet must have the specified delimiters at both ends, and (5) in addition, the following conditions are violations of coding rules and can be attributed to errors:

$a = b = c = d$,

$P13 \cdot e' \cdot i'$,

$P31 \cdot e \cdot i$,

$f = g = h = j$,

$e = i = f = g = h$,

$i \neq e = g = h = j$,

$(e = i \neq g = h = j) \cdot (c = d = e)'$,

$P31' \cdot e \cdot i' \cdot g' \cdot h' \cdot j'$,

$P13' \cdot e' \cdot i \cdot g \cdot h \cdot j$.

Note that undefined 6B and 4B subblocks give a fairly precise location of an error. For violations of the disparity rules, the location of an error can be bounded within a range which depends on the data pattern. Locating errors may be useful in support of forward error correction schemes.

In the paragraph on commas we claimed that with the receiver in character synchronism, no single digit error can generate the K.28.7 delimiter (*0011111000* or complement) from encoded data. To verify this claim, consider that no data code points have $c = d = e = i$; therefore, at least one digit error is needed to generate *abcdei* = *001111*, e.g., from *001110* or *001101*. Moreover, *fghj* = *1000* in data requires that $e = i = 0$; however, none of those 6B subblocks which can be changed by a single error into *001111* meet this requirement; therefore, a second error is needed in the 4B subblock, e.g., a change from *fghj* = *1010* to *1000*. The converse is also true: no single error can transform K.28.7 into a data code point.

The simplest error patterns which may escape detection by the code are a single erroneous *1* complemented by a single erroneous *0*. Such complementary errors, when confined to a single subblock, may simply change it into another valid code point. Single errors in subblocks change the disparity. Thus it is possible that a complementary pair of digit errors can change the disparity of two subblocks in conformance with the alternating polarity rule, such that the errors are not detectable by the code.

Cyclic redundancy checks (CRC) with 8B/10B code Single errors (or short error bursts) in the encoded line digits of a block code can generate a longer error burst in the decoded message. For the 8B/10B code proposed here, the effects of line digit errors are always confined to the 6B or 4B subblocks in which they occur and generate error bursts no longer than 5 or 3 respectively (from a single line digit error). This derives from the fact that each 6B or 4B subblock is uniquely decodable on the basis of just the digit values belonging to that subblock and without any reference to disparity or other extraneous parameters. The only exceptions are the special characters K.28.1, K.28.2, K28.5, K.28.6, for which the decoding of the *fghj* bits is dependent on the *abcdei* bits. However, adverse effects from this are limited because special characters usually appear only at specified slots with respect to the packet boundaries and usually are not covered by the CRC.

From the preceding paragraph one can conclude that the CRC used with this code should detect at least any combina-

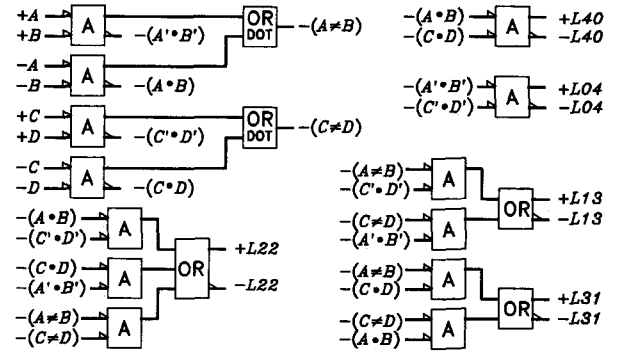

Figure 3 Encoder: 5B/6B classification, L functions.

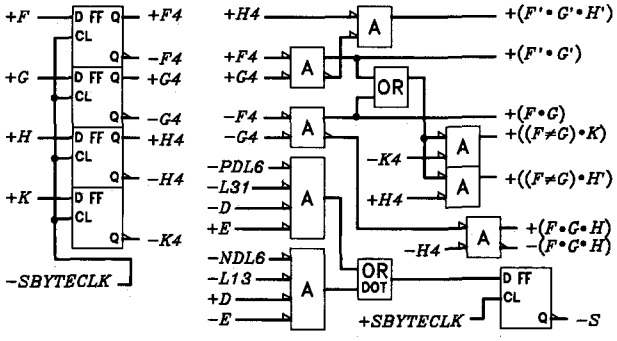

Figure 4 Encoder: 3B/4B classification, S function.

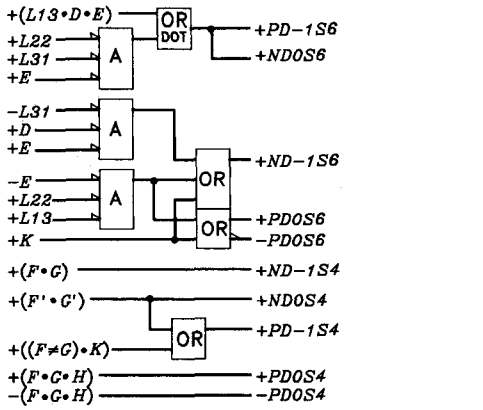

Figure 5 Encoder: disparity classifications.

tion of double errors in the line digits if it is to make a significant impact on the combined guaranteed level of error detection. A double error in the line digits generates, in the worst case, two error bursts of 5 each, after decoding. Fire codes are well suited for this application. Peterson and Brown [15, Theorem 8] show how to specify generator polynomials for cyclic codes with the capability of detecting two error bursts. With 16 check bits, two bursts of combined length 10 or less can be detected in packets as long as 142 bytes; 24 check bits can accomplish the same thing for packets as long as 36 862 bytes.

A CRC can generally detect any single error burst of a length which does not exceed the number of check bits. With the 8B/10B code described here, any single error burst of length 15 or less in the encoded digits cannot grow to more than 16 bits after decoding. Similarly, error bursts of length 25 and 35 in the encoded bits translate into error bursts no longer than 24 and 32 bits, respectively, after decoding.

In summary, for all packet lengths of practical interest, one can conclude that a 24-bit or longer Fire code combined with the inherent error detection capability of 8B/10B mapping can detect any combination of up to three additive errors in the line digits, provided that the beginning and end of the packets have been correctly established.

In practice, however, it is difficult to provide guaranteed error protection. This is because the distance properties of error detection codes do not apply to such conditions as incorrect detection of packet boundaries or the insertion of spurious channel bits. Here, a CRC may be viewed as providing for the detection of errors with a given probability. An n-bit CRC, if properly designed, reduces the probability of undetected errors by roughly a factor of 2^{-n}. For $n = 16$, this is in the order of 1.5×10^{-5}. The 8B/10B code detects substantially more than half of the random error patterns as noted above, so that the resulting overall probability of undetected errors becomes less than 10^{-5} for a 16-bit CRC. This may be quite adequate for fiber optic transmission, which is expected to have very good noise properties.

5. Implementation aspects

An 8B/10B encoder and a matching serializer, deserializer, and decoder were built and operated as part of an experimental fiber optic link operating at a signaling rate of 200 MHz. The encoder and decoder were implemented with ®MECL 10 000 series circuits [16]; and although tested at only 20 Mbytes/s, a maximum speed almost twice as fast can be projected with allowance for worst-case gate delays. All flip-flops (FF) shown are of the positive-edge-triggered type.

- *Logic circuits for the encoder*

The encoder is clocked by a byte rate clock (*+SBYTECLK*) derived from the transmitter and serializer clocks. The data source buffer (not shown) responds after each positive transition of *+SBYTECLK* with a new byte, while at the same time the encoded bits *abcdei* in the output register (Fig. 7, shown on the facing page) are being updated. A 0.6-byte cycle later, the source bits *FGHK* are read into FFs (Fig. 4)

on the positive transitions of $-SBYTECLK$, and the encoded bits *fghj* (Fig. 8) are updated. This staggered transfer is made possible because of the partitioned structure of the code; it simplifies the design of the serializer and the deserializer.

Figure 3 shows the generation of some of the basic classifications of Table 3 from the encoder inputs. For an explanation of the letter notations refer to Section 3.

The circuits of Fig. 4 generate the classifications of Table 4. The *F*, *G*, *H*, and *K* inputs are buffered to allow a serializer interface with the staggered timing mentioned earlier. As illustrated in Fig. 4, $-PDL6$ or $-NDL6$ is at the down level if the running disparity following the *i*-bit is positive or negative, respectively. Signals *PDL6* and *NDL6* are generated in Fig. 6.

Figure 5 implements the disparity classifications of both Tables 3 and 4. All inputs are from Figs. 3 and 4, or the data source, except for $(L13 \cdot D \cdot E)$, which can be found in Fig. 7. The mnemonics for the outputs are as follows: P for "positive," N for "negative," S for "sender" (as distinct from similar decoder functions on the receiving end), "$D - 1$" and "*D0*" refer to the respective columns in Tables 3 and 4, the number "6" associates a function with 5B/6B encoding and Table 3, the number "4" with 3B/4B encoding and Table 4. As examples, $+PD{-}1S6$ is at the up level for any input which has a plus sign in the "$D - 1$" column of Table 3, $+ND0S4$ is at the up level for any input with a minus sign in the "*D0*" column of Table 4.

The upper FF in Fig. 6 keeps track of the running disparity at the end of the *i*-bit, and the lower FF does the same for the *j*-bit. The gates to the right make the decision whether the alternate, complemented code points of Tables 3 and 4 apply (on the basis of the running disparity and the "$D - 1$" entry disparity classifications of Fig. 5). The gates to the left determine how to update the FFs, taking into account the *D0* disparity of the subblock being encoded, complementation, and the running disparity at the end of the previous subblock. As seen on the transmission link (L), $+PDL4$ is at the up level for a positive running disparity at the end of the *j*-bit.

Figure 7 shows the actual transformation of the five input bits *ABCDE* into the six *abcdei* output bits according to Table 3. The gates to the left of the XOR (exclusive-or) gates bring about all the bold type bit changes of Table 3. Figure 8 shows the 3B/4B encoding according to Table 4.

• *Logic circuits for the decoder*

The logical functions for the decoder and their classifications are defined in Tables 6 and 7. The implementation of these

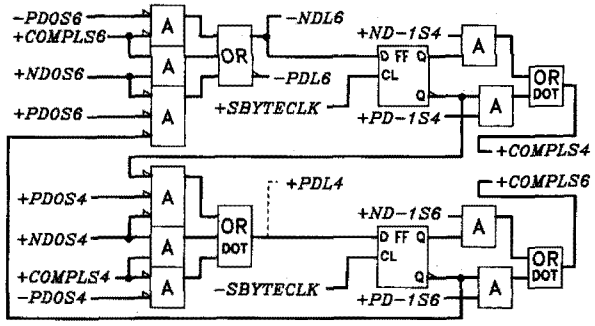

Figure 6 Encoder: control of complementation.

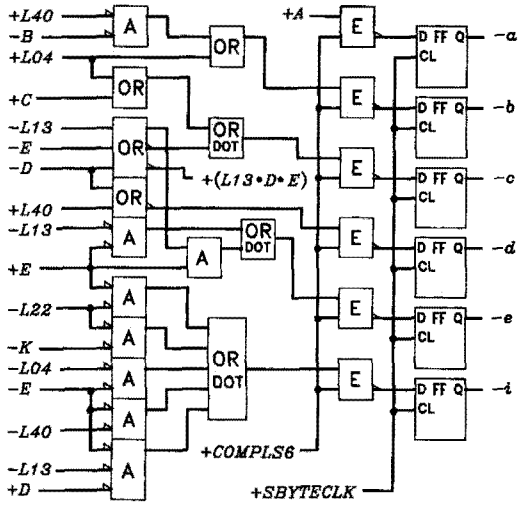

Figure 7 5B/6B encoding. Note: E = XOR.

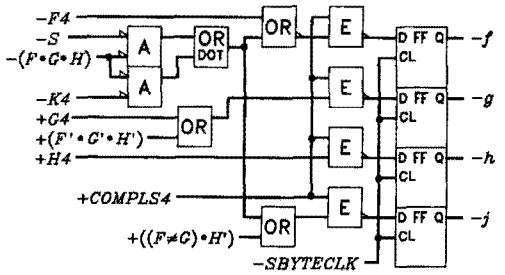

Figure 8 3B/4B encoding. Note: E = XOR.

tables in logic circuitry is very similar to encoding, but simpler, because it does not depend on the disparity. For decoding, the *i*- and *j*-bits are dropped and some of the remaining bits are complemented as indicated by bold **0** and

Table 7 4B/3B Decoding, K function.

Name	*f g h j*	*Decoding class*	*Disparity class*	*FGH*	*K*[c]	*D-1*	*D0*
D/K.x.0	0100	$f' \cdot h' \cdot j'$	$f' \cdot h' \cdot j'$	000	x	+	−
D/K.x.0	1011	$f \cdot h \cdot j$	$f \cdot h \cdot j$	000	x	−	+
D/K.x.1	1001			100	x	x	0
K.28.1	0110	$c' \cdot d' \cdot e' \cdot i' \cdot (h \neq j)$		100	1		0
D/K.x.2	0101			010	x	x	0
K.28.2	1010	$c' \cdot d' \cdot e' \cdot i' \cdot (h \neq j)$		010	1		0
D/K.x.3	1100		$f \cdot g \cdot h' \cdot j'$	110	x	−	0
D/K.x.3	0011	$f' \cdot g' \cdot h \cdot j$	$f' \cdot g' \cdot h \cdot j$	110	x	+	0
D/K.x.4	0010		$f' \cdot g' \cdot j'$	001	x	+	−
D/K.x.4	1101	$f \cdot g \cdot j$	$f \cdot g \cdot j$	001	x	−	+
D/K.x.5	1010			101	x	x	0
K.28.5	0101	$c' \cdot d' \cdot e' \cdot i' \cdot (h \neq j)$		101	1		0
D/K.x.6	0110			011	x	x	0
K.28.6	1001	$c' \cdot d' \cdot e' \cdot i' \cdot (h \neq j)$		011	1		0
D.x.7	1110		$f \cdot g \cdot h$	111	0	−	+
D.x.7	0001	$f' \cdot g' \cdot h'$	$f' \cdot g' \cdot h'$	111	0	+	−
D/K.x.7	0111	$g \cdot h \cdot j$	$g \cdot h \cdot j$	111	x	−	+
D/K.x.7	1000	$g' \cdot h' \cdot j'$	$g' \cdot h' \cdot j'$	111	x	+	−

[c] $K = (c=d=e=i)$ OR $(P13 \cdot e' \cdot i \cdot g \cdot h \cdot j)$ OR $(P31 \cdot e \cdot i' \cdot g' \cdot h' \cdot j')$.

1 entries in the tables. The disparity classifications of Tables 6 and 7 are needed only for error checking. Note that correct decoding of zero-disparity subblocks is dependent on signal polarity; i.e., the signal polarity between encoder and decoder cannot be arbitrarily changed.

• *Logic circuit performance requirements*

All the encoder and decoder circuits operate at the byte rate. To estimate the maximum tolerable gate delay, we look at a critical delay path. We use two gate delays for the XOR function, two gate delays from FF clock input to output, and one gate delay for the FF set-up time. It is assumed that valid input appears at the encoder input some two gate delays after the rising transition in *+SBYTECLK*. It takes another three gate delays to generate *L13* in Fig. 3. Following the longest path for *L13* in Fig. 7 to the data input of the *e*-FF together with the required set-up time adds another five delays. The total amounts to ten gate delays, which must be less than one byte interval.

There are other paths of equal length. The critical delay paths can be reduced to nine or eight gate delays with a moderate increase in circuit count; e.g., *L13* of Fig. 3 can be generated from *ABCD* with two, rather than three, logic gate delays using five 4-way gates. As an approximate rule one can state that the *gate delays* for the encoder and decoder should *not exceed one digit interval*; as an example, 10-ns technology can be used for data rates up to 10 Mbytes/s attached to a serial link operating at a digit rate of 100 MHz.

• *Circuit count*

For the circuit count, each XOR gate is counted as three gates. The gate count for the encoder (Figs. 3–8) is 89 gates and 17 FFs; the decoder required 79 gates and 9 FFs; error checking takes 44 gates and 2 FFs. Counting each FF as six gates results in a total of 380 gates for the encoder, decoder, and error checking. At low and medium data rates, where the gate delays are much less than a digit interval, some trivial design changes can significantly reduce the circuit count. For one, if the total encoding and decoding time delays are only a fraction of a byte, ten buffer FFs at the output of the decoder and the encoder buffers of Fig. 7 or Fig. 8, or both, can be eliminated. Second, the remaining FFs can perhaps be implemented as 3-gate latches after some modifications in the clocks. Therefore, at low byte rates, an encoder and decoder without error checking can be implemented with 168 gates and seven latches for a gate count total of 189 gates. With a complete set of error-checking circuits, the lower limit on the gate count is about 244.

The circuit count for the serializer, deserializer, and associated clocks for our experimental 200-MHz link amounts to 30 gates and 34 edge-triggered FFs for an equivalent total gate count of 234 high-speed gates.

6. Conclusion

For packet transmission over local area networks or computer links, the 8B/10B coding format described yields a near-optimum combination of relevant properties such as coding efficiency, complexity, digital sum variation, run length, error propagation, and suitability for ring or point-to-point topologies. These desirable attributes are, to a large extent, a consequence of the partitioned code structure. It is a binary code, the preferred transmission mode for fiber optic links for this kind of application, and it is compatible with the byte structure, which is ubiquitous in computer networks.

An implementation of a coder and decoder with emitter-coupled logic required a total of only 380 logic gates with gate delays up to one-tenth of a byte-clock interval.

Although the code was developed with high-speed fiber optic links in mind, it should also benefit lower-speed links over copper wire, which quite often run with 1B/2B codes currently. If such links are crosstalk-limited, the lower signaling rate of the 8B/10B code provides significant improvements in the signal-to-noise margin, because of lower attenuation and higher crosstalk impedance at the reduced rate. Finally, the difference in speed requirements in the serial and parallel sections of a communications adapter using this code is reduced, which in turn makes single-technology, single-chip implementations possible more often than with other codes.

Acknowledgments

We are indebted to J. D. Crow for project initiation and guidance; to A. E. Ferdinand and R. F. Steen for project funding and valuable discussions; and we thank T. Tsung for building and programming the automated link test system.

References

1. Y. Takasaki, M. Tanaka, N. Maeda, K. Yamashita, and K. Nagano, "Optical Pulse Formats for Fiber Optic Digital Communications," *IEEE Trans. Commun.* **COM-24,** 404–413 (1976).
2. J. M. Griffiths, "Binary Code Suitable for Line Transmission," *Electron. Lett.* **5,** 79–81 (1969).
3. R. G. Kiwimagi, "Encoding/Decoding for Magnetic Record Storage Apparatus," *IBM Tech. Disclosure Bull.* **18,** 3147–3149 (1976).
4. A. X. Widmer and P. A. Franaszek, "Transmission Code for High-Speed Fibre-Optic Data Networks," *Electron. Lett.* **19,** 202–203 (1983).
5. P. A. Franaszek, "Sequence-State Coding for Digital Transmission," *Bell Syst. Tech. J.* **47,** 143–157 (1968).
6. P. A. Franaszek, "Sequence-State Methods for Run-Length-Limited Coding," *IBM J. Res. Develop.* **14,** 376–383 (1970).
7. A. M. Patel, "Zero-Modulation Encoding in Magnetic Recording," *IBM J. Res. Develop.* **19,** 366–378 (1975).
8. Peter A. Franaszek, "A General Method for Channel Coding," *IBM J. Res. Develop.* **24,** 638–641 (1980).
9. P. A. Franaszek, "Construction of Bounded Delay Codes for Discrete Noiseless Channels," *IBM J. Res. Develop.* **26,** 506–514 (1982).
10. B. Marcus, "Sofic Systems and Encoding Data on Magnetic Tape," Preliminary Report, *Notices, Amer. Math. Soc.* **29,** 43 (1982).
11. R. L. Adler, D. Coppersmith, and M. Hassner, "Algorithms for Sliding Block Codes," *IEEE Trans. Info. Theory* **IT-29,** 5–22 (1983).
12. G. Nigel N. Martin, Glen G. Langdon, Jr., and Stephen J. P. Todd, "Arithmetic Codes for Constrained Channels," *IBM J. Res. Develop.* **27,** 94–106 (1983).
13. Ta-Mu Chien, "Upper Bound on the Efficiency of DC-Constrained Codes," *Bell Syst. Tech. J.* **49,** 2267–2287 (1970).
14. J. J. Stiffler, "Theory of Synchronous Communications," Prentice-Hall, Inc., Englewood Cliffs, NJ, 1971, pp. 368–372.
15. W. W. Peterson and D. T. Brown, "Cyclic Codes for Error Detection," *Proc. IRE* **49,** 228–235 (1961).
16. *Motorola ®MECL Integrated Circuits,* Series C, Motorola Semiconductor Products, Inc. (subsidiary of Motorola, Inc.), Phoenix, AZ.

Received November 1, 1982; revised May 6, 1983

Peter A. Franaszek *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Franaszek is a member of the Computer Sciences Department at the Thomas J. Watson Research Center. His interests include analytical problems associated with storage hierarchies, computer organization, magnetic recording, and digital communications. He received his B.Sc. degree from Brown University, Providence, Rhode Island, in 1962, and the M.A. and Ph.D. degrees from Princeton University in 1964 and 1965. During the academic year 1973–1974, he was on sabbatical leave at Stanford University as a Consulting Associate Professor of Electrical Engineering and Computer Science. Prior to joining IBM in 1968, he was a member of the technical staff at Bell Telephone Laboratories. Dr. Franaszek is a member of the American Association for the Advancement of Science, Sigma Xi, and Tau Beta Pi.

Albert X. Widmer *IBM Communication Products Division, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Widmer is a senior engineer with the advanced technology group, which is dedicated to complementing selected Research Division projects for efficient technology transfer to a product division. Recently he worked closely with research groups active in fiber optic technology and I/O architectures. He received an IBM Outstanding Innovation Award in 1981 for his contribution to the design of an optical receiver. Mr. Widmer graduated in 1956 from the Swiss Federal Institute of Technology (ETH), Zurich, with the Dipl. Ing. degree in electrical engineering. After holding a position as a teaching assistant with the Institute for Telecommunications, ETH, he joined the IBM Advanced Systems Development Division in 1959. He worked on the hardware aspects of a variety of special engineering projects, many of them related to communications, in IBM laboratories in Poughkeepsie, Mohansic, and Fishkill NY. His current interests are image processing and signal processor applications. Mr. Widmer is a member of the Institute of Electrical and Electronics Engineers.