RC 18855 (82405) 4/23/93
Telecommunications 39 pages

Research Report

The ANSI Fibre Channel Transmission Code

Albert X. Widmer

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598

The ANSI Fibre Channel Transmission Code

Albert X. Widmer

IBM Research
T. J. Watson Research Center
Yorktown Heights, NY 10598

Abstract: The Fibre Channel Standard transmission code is described here in a form suitable for implementers and link architects. A design method for similar codes and evaluation criteria are presented. Useful features of the code are described, and a complete design is shown, which is based on CMOS logic elements and meets all current FCS requirements.

CONTENTS

Introduction

The Fibre Channel Standard (FCS) is being developed under the auspices of the American National Standards Institute (ANSI) by a Task Group X3T9.3 of the Technical Committee [1992] on Device Level Interfaces. The standard describes the point-to-point physical interface, transmission protocol, and signaling protocol of a high performance serial link for support of the higher level protocols associated with High-Performance Parallel Interface (HIPPI), Intelligent Peripheral Interface (IPI), Small Computer System Interface (SCSI), and others, as summarized by Sachs [1992]. A brief and simple description of the transmission code adopted for FCS has been published in Electronics Letters [Widmer and Franaszek, 1983], and a more elaborate one in the IBM Journal of Research and Development. [Widmer and Franaszek, 1983].

This paper intends to give a justification for the transmission code adopted. Also, it presents the design method that led to this code, and describes some guidelines for the logic implementation in an IC. The viewpoint is that of an IC logic designer, rather than a transmission code theoretician [Franaszek, 1989][Cideciyan, 1993] , for example, the trellis diagram is used to visualize the steps of choosing certain coded bit patterns to represent data patterns. A complete description of an implementation is included.

We first describe the application environment and define some concepts which are useful in constructing codes. We then describe the underlying principle of a certain class of transmission codes, and a method to develop and evaluate several variants. Then the coded bit patterns of the partitioned FCS code are defined in the form of trellis diagrams. The transmission parameters of the code are evaluated. A non-partitioned 8B/10B code is also constructed for purposes of comparison, and a 10B/12B code is presented to further illustrate the design and evaluation technique. Next, data points and special characters are assigned to each coded vector of the FCS 8B/10B code and enumerated in tables. Logic equations are developed for the translation to and from the coded domain and for error checking, and an example of CMOS circuits for a coder, a decoder, and for error checking is given. We conclude with application notes on byte, word, or frame insertion and removal, the Idle sequence, the comma, direct bit manipulation, the Manchester subset of the code, observations on error control with the code, and a description of the low frequency requirements for the coded serial bit stream. An "Appendix" on page 37 explains why a receiver should change the byte alignment only on a command from the next higher protocol level.

Code environment

The Fibre Channel operates at a serial coded bit rate of 1.0625 Gb/s, or at rates reduced by a factor of 2, 4, or 8. The transmission media for the higher rates and longer distances is optical fiber, but coaxial cable or shielded twisted pair cables may be substituted where they can meet distance and bit rate requirements. The use of a single two level code for all these variations is generally not disputed. However, some alternate codes were considered.

The approach of scrambling and adding a modest number of control bits, often used by telecommunications common carriers, is not attractive for the following reasons:

- For the comparatively short computer links, the maximum exploitation of the available bandwidth is subordinated to transceiver cost considerations. Scrambling,

with the associated occasional long intervals between transitions and DC wander, increases the cost of the transceiver and makes the production of a compact, inexpensive, and totally monolithic design more elusive.

- The specified error rate for FCS links is better than 10^{-12}, and some users expect error rates of less than 10^{-15}. Common carriers are usually satisfied with substantially higher error rates. It is harder to achieve this quality level with no effective hard limit on the run length, which is the maximum number of contiguous identical symbols. Also, for common carrier links the statistics are more random to begin with, since the traffic is typically composed of interleaved short segments from many sources. A bad pattern is unlikely to be repeated on retransmission. In contrast, FCS traffic consists of frames with hundreds of bytes up to a limit of 2148 bytes.
- State of the art high performance electronic package technology is generally not affordable for computer links. Receiver circuits, including the clock recovery circuit, are very sensitive to switching noise. The combination of low cost components with extreme error performance requirements is much more difficult to meet and test with poorly constrained bit patterns. It is simpler and less costly to verify adequate performance with a reduced set of constrained bit patterns, as represented by the FCS 8B/10B transmission code.
- FCS frames vary in length and are organized around 4 byte words, with special, non-data markers with byte synchronizing capability at the start and end of a frame. The network may also include dynamic switches that may require resynchronization at both the bit and word level after each switching event on the affected outbound links. Rapid alignment of the receivers with the bit and word boundaries, and recognition of the start and end of frame markers, are requirements. Also, an out-of-synchronization condition must be quickly recognized and corrected. Statistical and counting methods commonly used for this purpose for scrambled data are too cumbersome and slow.

Another code seriously considered by the task force was the 4B/5B code of FDDI [Ross, 1989] because of the existing use. Its main disadvantage is the lack of DC balance; only the rate at which imbalance accrues, is constrained. The importance of well behaved low frequency characteristics has increased over the last decade because of the trend toward totally monolithic designs. Poor component tolerances and the availability of only very small capacitances in monolithic designs make it difficult to avoid reductions in the noise margins with otherwise comparable, unbalanced codes. Today's unexpected reality is that for many applications metallic media still have a cost advantage. For metallic media, DC balance is more important. DC balance and a reduced low frequency content bring many benefits that are appreciated by experienced circuit designers:

- Level settings of the laser driver and the receiver threshold can be based on the average signal level, which is simpler and more precise than using level restoring circuits.
- Thermal cycling of lasers or LED's is eliminated.
- Capacitive or transformer coupling is possible, without complications, to accommodate various package, grounding, and power supply configurations at the transmitter or receiver, or for isolation from metallic transmission lines.

- Capacitive differential coupling at the front end of fiber optic receivers with small integrated capacitors is more easily accomplished and provides better noise margins.
- At the lower data rates, designers may still include DC restoration circuits with a balanced code to reduce the capacitance values to a range compatible with integration in monolithic circuits. Such restoration circuits require less precision and complexity for a balanced code.
- Receivers at the end of computer links generally require a large dynamic range, which is more readily achieved with a balanced code.
- On metallic lines, a balanced code can often operate without equalization, which otherwise would be needed to correct the different attenuation of high and low frequency signal components.
- It is desirable to set the low frequency cutoff of receivers as high as possible to filter several noise sources, such as: power supply noise, low frequency modal noise arising from movements of multimode fibers, or 1/f noise of front end devices, especially GaAs devices not optimized for low noise analog operation.
- Design, performance testing, and trouble shooting is simplified for a well constrained code. Numerous comparisons between codes have been based on random data. A paper by Guo et al. [1990] concludes that the FDDI code has about the same error rate as the FCS code for random data at a certain design point, if the low frequency cutoff-off is reduced by a factor of 10 for the FDDI receiver. Such comparisons systematically belittle and camouflage the disadvantages of poorly constrained codes. One purpose of a good code is to provide maximum freedom from the vagaries of data statistics. For an analysis of FCS traffic with its long contiguous frames, the assumption of random data must be used with great care and reservation.

Another disadvantage of the FDDI code is the NRZI (Inversion on ones) feature, which usually requires Exclusive-OR circuits operating at the serial rate. This reduces the maximum rate that can be achieved with a given logic technology. Alternately, an NRZI conversion in the parallel domain entails additional logic complexity.

A third code proposal with sizable support for FCS involved a table look-up 8B/10B code with an additional 2 bits per byte redundancy for forward error correction. The stated goal was to achieve an error rate of better than 10^{-17} on laser links with noise floors resulting from laser reflections and mode partitioning. Also, some FCS users find the round trip delays for retransmission on links of several miles unacceptable. We plan to publish a separate paper showing how most single error bursts of various length in a FCS frame can be corrected, using the code presented here, with modest bandwidth overhead, inserted after the end of the frame.

Several other coding proposals were made. They were generally characterized by: very long run lengths, e.g. 20, or a soft limit of 45 in another case; no, or poor control of DC; long error propagation; and the lack of a comma.

Concepts and definitions

Some authors have used the following terms with slightly different meaning, especially with related applications, such as magnetic recording. They are here defined as follows:

A *vector* is a particular bit pattern of a defined length. A *source vector* is the uncoded data or control character from an information source. Source vectors

below contain 3, 5, 8, or 10 bits. A *coded vector* is the bit pattern resulting from coding. Coded vectors below contain 4, 6, 10, or 12 bits.

The *run length RL* is the maximum number of contiguous one or zero bits, either inside bytes or across byte boundaries, as seen on the transmission media, i.e. after the NRZI operation, if applicable. NRZI stands for Non Return to Zero Invert on ones: a technique where a polarity or level transition represents a one, and the absence of a transition represents a zero.

A *code point* is an uncoded source vector together with the associated single vector or pair of vectors in the coded domain.

The disparity of a vector, or the *block disparity DB* is the difference between the number of one and zero bits in a defined block of bits.

The *running disparity RD* takes on a new value after every transmitted bit. It is incremented by one for a one bit and decremented by one for a zero bit. For balanced codes it is usually referenced to a steady state average value of zero.

The *digital sum variation DSV* is the difference between the maximum and minimum value of the running disparity between any two points in the bit stream (not just between byte boundaries). For a balanced code, the DSV is finite over an arbitrarily long string of bits.

The *normalized offset* of a block of bits is closely related to the low frequency spectrum content that the bit pattern can generate. It is defined as the sum of the RD values after each transmitted bit, divided by the number of bits in the block, e.g. a coded byte. The normalized offset is a new parameter for the easy and more accurate measurement of low frequency differences than is possible with the DSV alone, which does not penalize codes that remain near the disparity extremes for a longer time. It has been validated by simulations as described under "Low frequency wander (Table II)" on page 8.

A *comma* [Stiffler, 1971] indicates the proper byte boundaries and can be used for the instantaneous acquisition or verification of the character and word boundaries. To be useful, a comma must occur with 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. A *comma character* is a coded byte completely containing a comma.

A *Special Character* is a valid transmission character which does not translate into one of the 256 valid data bytes. Special characters are sometimes also called control characters because of the way they are used in a system.

A FCS *word* is a string of four contiguous bytes occurring on boundaries that are modulo 4 from a specified reference, e.g. the start of a frame.

An *ordered set* is a word composed of a Special Character in its first position followed by three data characters.

The construction of low disparity binary transmission codes

Basic coding principle (Fig. 1 and Table I)

The transmission code adopted for FCS is a low disparity partitioned 8B/10B block code, as initially published by Widmer and Franaszek [1983], with only a few additional application-specific rules. The basic coding principle had been applied previously to the case of 5B/6B and 3B/4B codes by Griffiths [1969]. The concept used to remove the DC content from the coded sequences can be thought of as an extension of the widely-used ternary bipolar or alternate mark inversion code, for which a zero bit is transmitted as a zero level, and one bits are transmitted alternately as positive and negative pulses, to maintain overall DC balance. In the family of mB/nB codes, the zero level and the pulses are replaced by a group of n bits, where n is even. A one bit is transmitted at the up level and a zero bit as a down level. An equal number of one and zero bits in a group of n bits is analogous to the zero level of the bipolar code. A group with a majority of ones takes the place of a positive pulse, and the complement of that group represents a negative pulse. A positive majority and its negative complement are synonyms in terms of information content in the same way as in the bipolar code, where a positive or a negative pulse are both interpreted as a one bit. This principle is illustrated in "Fig. 1. Extension of bipolar code" on page 31 for the case of n=4. The shaded areas represent the two-bit positive or negative disparity of unbalanced blocks.

Before selecting any particular code group or vectors, it is useful to take an orientation tour through a table of binomial coefficients shown in "Table I. No. of vectors of disparity DB for nB bits" on page 26, which plots the number of vectors of block disparity DB for groups of nB binary digits. The trivial case, 1 bit, is either a one with DB = +1, or a zero with DB = −1. The 2B column represents the various Manchester type codes, showing two balanced vectors (01 and 10) and complementary vectors with DB = ±2, which are used on the token ring LAN [Bux, 1983] as delimiters. An intrinsically balanced code would have enough coded vectors of zero disparity to translate all data patterns into coded vectors. Examination of the 10B column shows, that it is not possible to construct an 8B/10B code with balanced vectors alone, because 252 is less than 256. Actually we require at least one extra code point for control functions. Table I also reveals that it is possible to construct an 8B/10B code with local parity, for which any single error in a byte can be detected and located to a specific byte, by using a combination of balanced vectors and complementary vectors with DB = ±4, as proposed by Martin [1985]. Such codes exhibit more low frequency content and the implementation is more costly. In the 4B column we have 6 balanced vectors and 4 pairs of vectors with DB = ±2, enough to code 10 different data vectors. For the coding of 3 bits we need at least 8 code points, but prefer 9 to code control information. Similarly the 6B column contains more useful variants than the 32 needed for the coding of 5 data bits. Note that non-zero disparity 4B vectors may be balanced by either 4B or 6B vectors of complementary polarity, and vice versa.

Table I also shows why it is difficult to construct a balanced 4B/5B code. There are no balanced vectors in the 5B column or any other odd numbered column. Even if we use all the higher order disparities, there are only 16 pairs available, just enough to code 4 data bits.

If we look beyond Table I, we find that 2 bytes can be coded into balanced 20 bit blocks, because there are 20!/(10!•10!) = 184,756 balanced vectors, many more than

needed to code 16 bits (2^{16} = 65,536). A code with DSV=6 and perhaps RL=4 appears feasible. Such a code has two inherent advantages:

- The coding of any two-byte word does not depend on the coding of any other word; in coding theory such codes are said to be 'memoryless'. A memoryless code has a significant advantage in applications which require extensive insertion and removal of information blocks in the coded byte stream, as is the case for time division multiplexing and perhaps for ATM (Asynchronous Transfer Mode).
- The code can detect any errors in a two-byte word that do not maintain the balance between ones and zeros. The error can be located to a particular word and thus error correction schemes based on simple parity can easily be implemented.

The disadvantages are a more complex implementation and error spreading up to 16 bits. A code in this class has been suggested by McMahon et al. [1992], in order to improve the low frequency behavior over that of the FCS code. The proposal is apparently based on the fallacy that a balanced code automatically results in minimal low frequency wander. The trace of a signal pattern through the trellis diagram pretty much represents the low frequency wander and it is obvious that even for a code which uses only balanced words, the trace can remain at or above zero RD for an indefinite time and then stay at the opposite polarity also for an indefinite time. This means that arbitrary low frequencies can be generated, although the amplitude of such power spectrum components remains low. Using methods described under "Low frequency wander (Table II)" on page 8, we have simulated the eye closure resulting from a high pass filter in the signal path of a 16B/20B code with the above parameters, and a run length of at most 2 at the start and end of the 20 bit blocks. The bit patterns simulated are n x "1101101010'1010100100" with a normalized offset of +2 followed by n x "0010010101'0101011011" with a normalized offset of −2. In the range of interest for low frequency cut-off points of 4% or less of the bit rate and for eye closure penalties of less than 7 dB, we found almost identical eye closures as for the partitioned FCS code.

Construction of the partitioned 8B/10B code (Fig. 2)

To construct an 8B/10B code, we can either do it in a single step, as done over the years in many versions [Stevens, 1990], [McMahon et al., 1992], [Zurfluh, 1992], [Cideciyan, 1992], or compose the code from compatible, but separate 5B/6B and 3B/4B codes. We show in "Comparison of partitioned and non-partitioned 8B/10B codes" on page 11 that both approaches result in comparable transmission parameters, but the partitioned approach has a significant advantage in implementation, less error spread, and better error locating capability, which is important for error correction.

The first step in code construction is the selection of the vectors to be used in the coded domain. The 5B/6B part of the partitioned code needs a minimum of thirty-two 6B (6 bit) code points, to code 5 source bits. A source vector may be translated into a balanced vector or a pair of complementary vectors assigned to a single source vector. The 3B/4B part requires at least 8 code points. Because there are a few more coded vectors available than needed, we can improve the transmission parameters by striking out or restricting the use of some of the vectors with the worst properties. A trellis diagram as shown in "Fig. 2. Trellis for partitioned 8B/10B code" on page 31 and "Fig. 3. Trellis for non-partitioned 8B/10B code" on page 32 can be a big help in visualizing the effects of choosing various code options and for weeding out unsuitable vectors that would violate run length or disparity constraints, or vectors which would destroy the uniqueness of the chosen comma. The trellis diagrams shown here plot the running

disparity (vertical axis) as a function of time or digit intervals for various coded vectors. The symbols ±, +, and − are abbreviations for RD=0, RD=+1, and RD=−1, respectively. All vectors shown in the trellis diagrams start and end with RD = +1, or RD = −1. The relative position of each node on the vertical axis represents the RD at the end of the bit sequence to the left. Each one bit is marked by a rising line segment extending over one digit interval; conversely a zero bit is represented by a falling line. Trellis diagrams are commonly used as state diagrams. For this particular application, another way of looking at the trellis diagram is as the output of a perfect integrator for transmitted square pulses. This perspective helps to understand the low frequency characteristics of a code. As a low pass filter, the integrator sorts out the low frequency components. Each valid bit sequence generates one of the possible traces through the diagram. The numbers to the right of the nodes indicate the number of different paths from the starting node to that location. These numbers are easily generated from node to node, starting with the left side entry-node. In this way the number of vectors meeting the established constraints can readily be obtained without mathematics and pattern searches. The letters 'abcdei' and 'fghj' refer to the individual bits of 6B and 4B vectors, respectively. This notation is explained in detail under "Code assignment" on page 14.

As mentioned previously, it is not possible to balance an 8B/10B code within one byte, so we do the next best thing and require that the running disparity RD at the start and end of a byte, and for this particular code also after the 6th coded bit, be constrained to ±1. The polarity of RD must be passed along during coding from byte to byte. Fig. 2 shows at the top vectors referred to as Group A, which are balanced, and can be entered regardless of the starting RD. The vectors shown in Group B are also balanced, but can be entered only when the starting RD is −1. If RD = +1, the complement of the vectors must be used. Without this restriction, a RL of 6 and a DSV of 8 would result. Group C contains vectors with a DB of +2. Again, they must be complemented if the starting RD is +1. Group A, B, and C together account for thirty-two 6B code points and seven 3B code points. The 8th 3B code point is illustrated in Group D and consists of 4 synonymous vectors. The normal vector '0001' at the top left side or its complement '1110' at the top right side may be used only if the 6B vector does not pass through node X or Y, because it would generate a RL of 5. If a 6B vector passes through X or Y, the alternate 4B vectors '1000' or '0111' at the bottom must be used. This alternate vector can generate a run of 5 across the trailing byte boundary that is always preceded by a run of just one and thus is distinct from the comma, which consists of a run of 2 or more followed by a run of 5.

The thirty-two 6B and eight 4B code points defined so far are used to code the 256 code points of a data byte. Note that 6B and 4B coding is dependent on the starting RD, but 4B coding is sometimes dependent on the trailing 2 bits of 6B as well. However, the decoding of 6B or 4B is state independent; it can be accomplished based solely on the 6 or 4 bits and there is no knowledge required of the running disparity, except for error checking.

Group E defines a set of 8 special characters. They can be recognized as such by the presence of the bold line. The trailing 4 bits of 6B are either all one or all zero.

Group E includes 3 comma characters. A comma is a singular sequence of bits which can occur in the serial bit stream in only one position relative to the byte boundaries. For practical reasons the comma should be limited to one byte and be present in a special character, rather than a data byte. Implementation cost and speed penalties are minimized, if the comma is short and near the front end of the comma character.

The selected comma sequence for this code is a RL of 5 preceded by a RL of 2 or more, i.e. '0011111' or its complement '1100000'. Comma characters with many transitions are preferred. The receiver phase locked loop normally acquires bit synchronization on a repetitive Idle Sequence. This task is easier with lots of transitions. FCS includes a comma in the Idle Sequence to establish verified byte and word alignments before the start of a frame.

As an exception applicable to Group E only, a positive starting RD causes all ten bits of a byte to be complemented, even if the 4B part is balanced. This special treatment preserves the comma sequence regardless of the starting RD.

Group F shows another set of 4 special bytes. They are recognizable as such by the pattern '101000' (or the complement) in the last 6 bits of the byte. Their distinction from all other vectors of the code can readily be verified with the trellis diagram, which shows all Group F vectors going through node F; as shown in Group D, no data traces ending in '1000' go through node F.

The Envelope shows the superposition of Group A through F for both starting RD polarities. In the trellis diagram for the envelope, not all traces are valid. Invalid traces include a RL of 4 starting with the first bit 'a' of a byte, a RL of 5 starting with the 5th bit 'e', and others as defined under "Error checking equations" on page 19.

Transmission properties of the partitioned 8B/10B code

The parameters of most interest to the link and receiver designers are as follows:

Transition density

A high transition density and a short maximum RL are helpful in containing the cost of the receiver clock, especially monolithic designs. A low transition density to reduce crosstalk noise might be preferred for transmission on unshielded metallic cables.

The maximum RL is five. Contiguous runs of 5 can only be generated by the special character '0011111000' or '1100000111' followed by certain other characters. FCS does not currently use this character and the FCS specifications rule out future enhancements which would generate contiguous runs of five[1] . Except for this character, the minimum transition density is 3 per byte. The FCS Idle Sequence, which includes a comma, has 32 transitions per 40 coded bits. For random data the chance for a RL = 5 across a byte boundary is 0.0011.

Low frequency wander (Table II)

The code is DC balanced with a maximum DSV of 6. The DSV is a very coarse measure of the low frequency energy content. A supplemental measure is the offset, which amounts to 1.9 for this code. The offset can be further explained as the area in the trellis envelope diagram between zero disparity and the outermost contour a coded byte pattern may follow, normalized to the average area per bit time. This measure is related to the amplitude of low frequency components. The frequency of the low end spectral components is related to the maximum time between crossings through zero

---

[1] This character, as used by the IBM ESCON (IBM Trade Mark) channel, can generate 2 contiguous runs of 5 and a false comma across the trailing end of the special character by the sequence '0011111000'001... or its complement. This requires receiver circuitry that prevents the start of a comma search between the leading bits of the 2 comma versions; once a comma is detected, the detection circuitry must be disabled for more than 5 bit intervals.

RD by a trace in an extended trellis diagram representing many bytes. For a FCS frame structure such crossings cannot be farther apart than the start and end of a frame. For such situations, spectral components cannot be lower than about 1/2F Hz, where F represents the frame length in units of time. However, the farther apart the zero crossings are, the lower are the amplitudes of the associated low frequencies.

We have simulated the effects of the low frequency wander on the eye diagram closure using worst case data patterns. The square waves from the signal source are first passed through a low pass filter. The filter chosen is the one recommended for measurements on FCS links, namely a fourth-order Bessel-Thompson filter with the 3 dB cutoff set at 0.75 of the bit rate. The signal is then passed through a single series capacitor and the eye closure penalty in decibels is computed for various 3 dB cut-off points of this RC high pass filter, expressed as a fraction of the coded bit rate f_0. The bit pattern applied follows the upper contour of the trellis envelope (1101001010) for several bytes, then moves to the lower contour by the pattern '1101000100', then it follows for an equal number of bytes the lower contour (0010110101), and finally returns to the upper contour via the pattern '0010111011'. "Table II. Eye closure penalty in dB" on page 26 shows the result for a total pattern length of 4, 12, 50, 100, or 200 bytes. From the progression of the eye closure with increasing pattern length, one can extrapolate that the longest possible patterns have not significantly more eye closure in the range of interest than the 50 byte pattern. The simulation results show that for a penalty of less than 0.25 dB, the high pass cut-off point should be no higher than 0.1% of the bit frequency. This is about a factor of ten lower than for Manchester code. Note, that the penalty doubles in the lowest frequency range for each doubling of the low frequency cut-off point.

Framing information

The partitioned 8B/10B code includes a comma, which is complete after the seventh bit of a byte and requires the monitoring of a minimum of six bits (0x11111 or 1x00000). Small values for these two parameters reduce the number of circuits in critical high speed portions of a deserializer. Three different comma characters are available; one of them, 00111110101 or the complement, has 4 transitions, not counting possible transitions at the character boundaries. Another comma character (0011111000 or complement) with only 2 inside transitions has a Huffman distance of 2 from data characters, i.e. it takes at least 2 bit errors to generate this comma character from data, and vice versa. This is true only with correct byte alignment.

Decodability and error extension

Decoding of data can be done by the examination of just 6 and 4 bits at a time. A single error in data will not create a burst longer than 5 or 3 bits in the decoded domain, unless a special character was generated from data and vice versa. In FCS, the single defined special character must be part of a valid ordered set, and a faulty special character will be exposed as spurious in most cases right away. Alternately, an error in the special character could convert it to data, which might change any bits of the coded byte. Again, such a conversion on a FCS link is spotted no later than in the next word, which will be inconsistent with the defined frame structure.

Error detection

A combination of any odd number of bit errors is detected as a violation of coding rules. An even number of bit errors is often detected as well. A pair of complementary adjacent or closely spaced errors has the best chance to escape detection by the decoder. Detectable errors are immediately recognized as invalid characters or as disparity violations no later than the end of the frame. For random data, the chance that a detectable error remains undetected within the first eight bytes from and including the error event is about one in 20,000.

A non-partitioned 8B/10B code (Fig. 3)

This is another example for the code design method using trellis diagrams. It shows what kind of parameters can be obtained for a 8B/10B code if implementation aspects are disregarded. We keep the maximum DSV at 6, but reduce the maximum RL to 4. We also should have an idea of how to generate a good comma. To find a suitable comma, we rely on intuition and trial and error. Commas are generally selected from extreme bit patterns; otherwise too many good vectors must be discarded to obtain a unique relationship of the comma position relative to the byte boundaries. Vectors which comprise the maximum RL and a rapid maximum change in the running digital sum can be considered as extreme, and have fewer aliases which must be eliminated. Since we restrict ourselves to coded bytes with a disparity of zero or ± 2, the comma can include at most 6 ones or 6 zeros. The bit patterns '1101111' and '1111011' or their complements are candidates for commas meeting our general criteria. We have chosen '1101111' and its complement for the code below. It also requires the monitoring of only six bits (11x1111). For the construction of a similar code, Stevens [1990] proposed a comma with 2 contiguous RL's of 4, namely '1000011110' or its complement. This entails a more complex deserializer, because we have to monitor 7 bits (000x1111) and the comma is not complete until after the 9th bit of the byte. Also, there are only 3 transitions inside this comma character. This is a disadvantage if the application calls for a comma character in the repetitive idle sequence.

In Fig. 3 all the vectors of a non-partitioned code are shown. Note that the RL at the start and end of the bytes is limited to 2 or less; therefore, the RL across the byte boundaries can be no more than 4. Any vectors which could generate a false comma across byte boundaries are also eliminated.

Group 1 contains 162 balanced disparity-independent vectors. Vectors through node Z with a positive starting RL are not allowed, because they would extend the maximum DSV of the envelope to 8.

Group 2 is a set of 4 balanced vectors with disparity constraints. If the starting RD is positive, the complements of the respective vectors must be entered. Vectors through the nodes X could generate false commas and are eliminated.

The vectors of Group 3 have all a DB of +2. They can only be entered if the starting RD is −1; otherwise their complement must be entered. If we draw a trellis through all the nodes shown for Group 3A, we find that it accommodates 135 vectors. Using other trellis diagrams which are subsets of this all-inclusive trellis, 30 vectors can be identified which either exceed the RL = 4 constraint, or generate false commas; we are left with 105 usable vectors with a DB of +2. Fig. 3 shows only the usable vectors. We also have to make sure that there are no duplicates, which could be done by an explicit vector listing and a programmed search. We gain more insight with the following procedure using the trellis diagrams: Group

3A presents a first set of 27 vectors going thorough node A. Unrestricted use of the other labelled nodes (B to H) generates violations. Group 3B is distinct from Group 3A, because all vectors go through node B, which is not part of Group 3A. Group 3BC is different from Group 3A and 3B, because the vectors go through nodes B and C. Group 3A vectors go through node C, but not B. Similarly Group 3D, 3E, 3F, 3G, and 3H are different from all preceding groups, respectively, because all vectors in each of the groups pass through a node which is not part of the previous groups.

A vector through node X of Group 3BC would generate the chosen comma pattern within a byte and it could be used instead of the comma included in Group 3E, which occurs one bit earlier and is therefore preferred. Because of the different position relative to the byte boundaries, one of the commas must be eliminated from the set of valid vectors.

The non-partitioned 8B/10B code of Fig. 3 includes 271 code points with the specified constraints, enough to code 256 data and 15 special points. The Envelope represents the sum of the vectors with positive or negative starting RD. We can see, that the maximum DSV is in fact 6. From the envelope we determine an offset of 2.3, which is 21% more than for the partitioned code.

Variations of non-partitioned codes

Many applications need only one or a very few special characters. We can define a code with only six special characters, and four or more transitions per byte, by eliminating 9 selected code points from Fig. 3:

Group 1: 1000111100 1100001110 1100011100 and complements

Group 3B: 1100011110 and complement

Group 3E: 1100111100 and complement

Group 3H: 0011110011 and complement

Some link architects value a Huffman distance of two between data and the comma. This can be obtained by eliminating 10 vectors from Group 1:

0010111100 0100111100 0110011100 0110101100 0110110100 and complements

The Huffman distance of 2 is only valid for correct byte alignment, not across byte boundaries.

If the disparity restrictions are removed from Group 2, we gain 4 code points for each of the above cases. The DSV increases from 6 to 8, but the offset remains at a value of 2.3. Simulations show that the compensation of the higher DSV with the offset remaining at 2.3 requires a 20% lower cut-off frequency for any high pass filter.

Comparison of partitioned and non-partitioned 8B/10B codes

The non-partitioned code has a shorter RL and can be further constrained to increase the transition density. The DSV is the same, but the offset is larger by 21% (2.3/1.9) for the non-partitioned codes. Simulations show that the low frequency cut-off point must be reduced by about 8% to maintain an equal eye opening. Error spread is generally increased to 1 byte for the non-partitioned code. The error locating capabilities of the partitioned code are far better, because an unbalanced error which is not immediately identified as a disparity violation or invalid character is more than twice as likely to be detected as a disparity violation in each succeeding byte. This feature is important for error correction and will be described in a forthcoming paper.

The number of special characters beyond a bare minimum is not a significant asset for many applications. FCS so far uses only one special character, a comma character. Additional compatible special characters could be added to the partitioned FCS code, if needed, with moderate growth in hardware complexity.

Similarly, the reduction of the number of spurious commas by a Huffman distance of 2 for the comma does result in only small performance improvements for FCS, if the rules of the explicitly-enabled byte alignment mode are followed. The reasons why the byte alignment circuitry must be disabled once alignment has been achieved are explained in an appendix at the end of this report. Under prevailing error rates, a synchronization attempt by the deserializer on a spurious comma is extremely rare. If it happens, the attached communications adapter will not recognize new synchronization unless it has seen 3 ordered sets that all start with a comma, with no errors in any of the sets or in any of the data between the sets. Otherwise immediate resynchronization is initiated. Once the verified synchronized state has been achieved, it will not be abandoned unless a significant number of errors suggest that synchronization may have been lost. This provides much stronger protection against false word alignments than any 10 bit pattern can provide. In this scheme, even a large reduction of spurious commas has a negligible effect and spurious commas are not much different from ordinary errors.

A Huffman distance of 2 for the comma is desirable, if error correction is included on links with frames of variable length. It is then more important to reliably recognize the start and end of a frame by simple means.

To summarize, there are no great differences in the important performance parameters of the two types of codes described above. There are different trade-offs, and one or the other code might be preferred for a specific application. The FCS code has good overall characteristics and no serious defects. As a partitioned code it is significantly easier to implement, which is still relevant today.

A detailed examination of the trellis diagrams leads to the conclusion that the codes presented above represent about the best results for the currently relevant transmission parameters that can be achieved for 8B/10B codes.

A partitioned 10B/12B code

Some specialized instrumentation generates 10 bit data units. A compatible partitioned 10B/12B transmission code can be constructed from concatenated pairs of 6B vectors from Fig. 2. Such a code can easily be evaluated with the help of trellis diagrams. It is well behaved with a DSV of 6, an offset of 2.17, a maximum RL of 5 with at most two contiguous runs of 5. The minimum transition density is 3 per 12 bit byte.

The comma sequence is '11111011' or '00000100' for the alignment to 12 bit boundaries. With negative starting disparity, a comma character can be generated by coding K28.D13 (00111 1.10110 0), which results in −001111+101100+. (For an explanation of the terminology 'K28.D13' refer to "Code assignment" on page 14). This is sufficient if the frame architecture is rigged so frames always start with negative disparity, analogous to FCS. If K28.D13 is coded with a positive starting disparity, we get +110000−101100−, which does not include the comma sequence. To get a comma with a positive starting disparity, we could code K28.D18, which translates to +110000−010011−. To generate a comma character regardless of the starting disparity, we have to use a trick analogous to the FCS code, i.e. if the first 6B vector of a 12B character is K28 ending with negative RD, all balanced 6B vectors in the second half

are inverted. With such rules, K28.D13 with positive starting RD translates to +110000–010011–, and the comma is preserved for either polarity. Because the run length is limited to five, one can limit the comma search to 7 bits as follows: 'xx11111x11xx' or the complement. Besides the comma character, it is possible to define 28 other special characters. The character K28 may be used only in the leading half of 12B vectors.

Using symmetries between the two 6 bit halves of the 12B code, we can define a second set of 30 special characters. For the set of 6B vectors applicable to the trailing half of the 12B characters, the K28 vector is replaced by K15. The coder input for K15 is 11110 1, and the coded output is –111100+ or the complement. This second set of special character includes another comma (11011111 or the complement) with identical alignment relative to the byte boundaries. This second comma can be embedded in a comma character D12.K15 (001100.11110 1) which is coded as –001101.111100+. To preserve the comma for positive starting RD, coding rules symmetrical to the ones above can be defined. The code point K15 must not be used in the first half of the 12B character.

Coding Map for the Partitioned 8B/10B Code

Up to now we have considered just the existence of enough vectors within the desired constraints. We now go to the assignment of 256 data and some control characters to specific coded vectors for the partitioned 8B/10B code only. Easy implementation is the prime consideration for the assignments chosen. The guiding principle is to change as few bits as possible in the translation to the coded domain, and to classify the vectors so a minimum set of rules applies to as many vectors as possible.

Code assignment

5B/6B and 3B/4B coding (Table III and IV)

"Table III. 5B/6B Coding" on page 27 and "Table IV. 3B/4B Coding" on page 28 show the coding assignments chosen. The capital letters 'ABCDE FGH' represent an uncoded byte. An additional input K signals a special character, if set to a one. The lower case letters 'abcdei fghj' represent the coded byte. The column 'Name' is used for easy reference to a line in the table; the numeric value in the name is equal to the decimal value of the uncoded bits, assuming the left bit is the least significant bit (This naming convention must not be taken as an inference to the location of the actual low order bit for a specific application!). The letter D in a name suggests a data point, the letter K a special character, and D/K means that the vector is coded the same regardless of the type of byte. The character may be recognizable as special by a pattern in the other part of the byte or by a pattern crossing the 6B/4B boundary. For serial transmission, bit 'a' is transmitted first and 'j' last, usually in non-return to zero mode.

The logic function classification L13 indicates that there are 1 one and 3 zeros in ABCD; L04 means there are 4 zeros. Analogous definitions hold for L40, L31, and L22. A single quotation mark (') following a symbol is used to represent negation; a dot (•) and a plus sign (+) stand for the logical AND and OR functions, respectively.

The primary coded bits labelled with lower case letters are obtained by first copying the corresponding uncoded uppercase bits and appending the extra bits 'i' and 'j' with a value of zero following the 'e' and 'h' bits, respectively. However, if the logic functions defined under 'Bit coding' hold, then the bold ones or zeros in the primary coded column represent exceptions to the above rule, e.g. if L04 is true, as happens for D0 and D16, bits 'b' and 'c' are forced to one. The second expression (L04•E) for D16 and D31 applies for the coding of the 'i' bit only.

In Table III, the bit coding classification L31•D•E' identifies three vectors D11, D13, and D14 that affect the coding of the 4B part, if the RD in front of the byte is +1 and FGH = 111. Similarly, L13•D'•E refers to D17, D18, and D20 which modify the coding of FGH = 111 if the front-end RD is −1. The function S is true, if any of these conditions exists. In Table IV the primary coded vector for FGH = 111 is listed on line Dx.P7. The alternate vector of line Dx.A7 or Kx.A7 is used if S = 1 or K = 1.

The disparity classifications determine the RD requirements at the front end of a coded vector, as indicated in the column DR. A DR value of + or − indicates that the RD must be +1 or −1, respectively, otherwise the alternate, complementary vector to the right must be entered. A DR entry of ± is assumed, unless there is an entry under disparity classifications. The same logic terms also classify the coded vector according to the block disparity as shown by entries in column DB. The DB entry is zero to indi-

cate a balanced coded vector; this includes all lines which do not have a logic expression in the Disparity column plus D7 in Table III and Dx.2 in Table IV. A DB value of + indicates that there are 2 more ones than zeros. The second disparity classification entry (F•G•H) for Dx.P7, Dx.A7, and Ky.A7 of Table IV applies to the DB column only.

The coding Tables III through VII are functionally identical to those originally published [Widmer and Franaszek, 1983]. There is a minor semantic difference: The S function was formerly defined in terms of the coded bits 'ei' and the RD at the end of the 6B block, to better illustrate the reasons for using the alternate Dx.A7 and Ky.A7 code points. It is now expressed as a function of the RD at the start of the byte, and of the uncoded bits ABCDE, to give better guidance for implementation.

Special Characters (Table V)

The 12 special characters are listed explicitly in "Table V. Special Characters (K = 1)" on page 28. Three of them contain the comma sequence, as indicated with bold type. To maintain the comma sequence for K28.1 and K28.5 regardless of the RD, it was necessary to deviate from the normal coding procedure for balanced 4B data vectors: For K = 1 all 4B vectors are disparity-dependent. The restrictions of Note 6 prevent the generation of false commas across the trailing end of K28.7, and eliminate any contiguous runs of five. These restrictions do not apply to applications of the code in IBM ESCON[2] channels [IBM Corp., 1992]. FCS currently uses only the K28.5 special character. At the time the code was developed, it was believed that the additional K28.7 comma character would be useful. It takes at least two bit errors to generate it from data, if byte synchronization is maintained. This feature makes the K28.7 attractive for use in Start and End of Frame delimiters, but not for the Idle Sequence because K28.7 has fewer transitions. Otherwise, a more convenient definition of K28.7 is –0011110001 or +1100001110, which does not contain a comma and has no sequence restrictions, but is currently not included in the valid alphabet.

Decoding (Table VI and VII)

"Table VI. 6B/5B Decoding" on page 29 and "Table VII. 4B/3B Decoding, K Function" on page 30 use similar notation as explained above for coding. The classifications P22, P13, and P31 are defined the same way as L22, L13, and L31, respectively. The bit K is assumed as zero unless Note 7 is valid. The disparity parameters are needed for error checks only. The entries in column DR of Table VII for K28.1, K28.2, K28.5, and K28.6 are intentionally left blank, since they are not needed for error checking; otherwise they would be negative.

Note that correct decoding of the balanced 6B and 4B blocks is dependent on signal polarity; the signal polarity between encoder and decoder cannot be arbitrarily changed.

---

[2] Trademark or registered trademark of the International Business Machines Corporation.

Circuit Implementation (Fig. 4)

"Fig. 4. Top level schematic for partitioned 8B/10B coder and decoder" on page 33 shows all the required inputs and outputs for a coder, decoder and error checking circuits. They are explained in detail below. For the coder, the signal +PDES4 represents the running disparity at the end of a byte as a reference for the coding of the next byte. For the decoder, an analogous reference signal +PDER4 from one byte to the next is required for error checking, but it is contained within Fig. 7 and does not show up in the top diagram.

The partitioned 8B/10B coder and decoder can be designed using only combinatorial logic and a few polarity hold latches to keep track of the RD. A single coder and decoder built with 0.8 μm CMOS gates, as shown below, can readily achieve rates well over 100 Mbyte/s. The circuits use logic gates with a maximum fan-in of 4. For the designs shown, preference has been given to NAND gates which use stacked n-type devices, over NOR gates which use stacked p-type devices. Because the n-type devices are faster, NAND gates tend to be faster. Questions have been raised about implementation difficulties for multi-Gb/s links [Zurfluh, 1992], but it should be obvious that the logic speed requirements are noticeably less demanding for this coder and decoder than for other link level circuits, such as the serializer and deserializer. There are also numerous options available to build the coder and decoder in much slower technology with an increase in circuit count. Parallel coders for 1.0625 Gb/s links using much slower technology have also been built, and some designs have been realized using logic synthesis programs. The operation of parallel coders has been described in concpt by Cideciyan [1993]. Timing and circuit efficiencies can be realized, if the coder and decoder design is not planned in isolation, but with its interface with the serializer and deserializer in mind, e.g. if it should become necessary to partition the coding process further into several steps, or to operate several coders in parallel, some of the steps, e.g. complementation, could be done as part of the conversion from a 4 byte format to the one byte format in the serialization process, or vice versa at the receiver end to save storage latches and to reduce latency. Users who prefer table look-up techniques can resort to either large 8B/10B tables with simple timing requirements, or exploit the partitioned structure with smaller 5B/6B and 3B/4B tables at the cost of more complex control. The 5B/6B table would comprise information about DR, DB, and the S function in addition to the bit translations.

Coder equations and circuit example

Conceptually, coding is done in two steps. We first make the translation to the primary vectors, and then determine whether the alternate, complemented vectors must be used to meet the disparity rules.

Generation of Primary Vectors

The logic equations necessary for the translation to the primary vectors can be read directly from Table III and IV:

$a = A$

$b = B \cdot L40' + L04$

$c = C + L04 + L13 \cdot D \cdot E$

$d = D \cdot L40'$

$i = L22 \cdot (E' + K) + E \cdot (L04 + L40 + L13 \cdot D')$

$f = F \cdot [F \cdot G \cdot H \cdot (S + K)]'$

$g = G + F' \cdot G' \cdot H'$

$h = H$

$e = E•(L13•D•E)' + L13•E'$ $\qquad$ $j = (F \neq G)•H' + F•G•H•(S+K)$

Disparity Control

For the description of disparity control via complementation, we introduce some new notation:

NDF and PDF reflect the Negative or Positive RD at the Front of a block.

NDB and PDB reflect the Negative or Positive Disparity of a coded block as indicated in the column DB of the Tables III-VII.

S6 stands for the 'abcdei' bits at the Sender, R6 for the same bits at the Receiver.

S4 stands for the 'fghj' bits at the Sender, R4 for the same bits at the Receiver.

NDFS6 and PDFS6 indicate negative or positive RD in front of a S6 block. NDFS4 and PDFS4 indicate negative or positive RD in front of a S4 block.

NDRS6 and PDRS6 represent all vectors with a minus or a plus sign (but not both) in the DR column of Table III, respectively. NDRS4 and PDRS4 are defined in the same way for Table IV.

NDBS6 and PDBS6 comprise all code points with a Negative or Positive sign, respectively, in the DB column of Table III. NDBS4 and PDBS4 are defined in the same way using Table IV.

BALS6 and BALS4 comprise all code points with a zero entry in the DB column of Table III and IV, respectively.

NDES6, PDES6, NDES4, and PDES4 refer to negative or positive polarity of the RD at the end of a S6 or S4 block, respectively.

If CMPLS6 or CMPLS4 is true, the alternate, complemented vector must be entered.

For the coding of a specific S6 or S4 vector, the polarity indicated in the column DR of Table III or IV, respectively, must match NDF or PDF, otherwise the alternate, complemented vector must be used.

From Table III and IV we can extract the following Boolean equations, using the logic expressions for disparity classification:

$NDBS6 = L22'•L31'•E' + L13•D•E$ $\qquad$ $NDBS4 = F'•G'$

$PDBS6 = L22'•L13'•E + K$ $\qquad$ $PDBS4 = F•G•H$

$BALS6 = NDBS6'•PDBS6'$ $\qquad$ $BALS4 = NDBS4'•PDBS4'$

$PDFS6(n+1) = PDES4(n)$ $\qquad$ $PDFS4 = PDES6$

$PDFS6 \equiv NDFS6'$ $\qquad$ $NDES4 \equiv PDES4'$

$PDRS6 = NDBS6$ $\qquad$ $PDRS4 = NDBS4 + (F \neq G)•K$

$NDRS6 = PDBS6 + L31•D'•E'$ $\qquad$ $NDRS4 = F•G$

$CMPLS6 = NDFS6•PDRS6 + PDFS6•NDRS6$

$CMPLS4 = NDFS4•PDRS4 + PDFS4•NDRS4$

$PDES6 = PDFS6•BALS6 + NDFS6•BALS6'$ $\qquad$ $NDES6 = PDES6'$

$PDES4 = PDFS4•BALS4 + NDFS4•BALS4'$ $\qquad$ $NDES4 = PDES4'$

All possible receiver input patterns are accounted for as either valid code points or as code violations. To achieve continuity from byte to byte, the ending RD PDES4

of byte n is stored in a latch, and then the starting RD of byte n+1 is made equal to it [PDFS6(n+1) = PDES4(n)].

Special Fibre Channel Requirements

FCS defines Ordered Sets which are used as Frame Delimiters, Primitive Signals, and Primitive Sequences. All these sequences start with the special character K28.5 and are followed by 3 data characters. The standard requires that many of the Ordered Sets start with a negative disparity. To force a negative starting disparity, the attached communications adaptor must assert a control line M together with the control line K. Similarly, the End of Frame Delimiter must always end with negative disparity, which is accomplished by manipulation of the 4B part of the second byte of the ending delimiter, which is a data byte. The data point Dx.4 is used in this position to go from positive to negative disparity, and Dx.5 is used to maintain the existing negative disparity. To get the desired result, the adapter will always present Dx.4 in this position and assert the M line, but not the K line. The circuitry of Fig. 5 will then force bit F to one if NDFS6 is true. This procedure converts Dx.4 to Dx.5. We can also control the conversion of Dx.0 into Dx.1 with the same result in disparity, and Dx.6 into Dx.7, with complementary original and modified ending disparity. The circuitry assumes a balanced S6 block, so NDFS6 = NDFS4 is true.

Coder circuit example (Fig. 5)

A complete coding circuit is shown in "Fig. 5. Partitioned 8B/10B coder" on page 34. The signal names follow closely the logical expressions and acronyms defined above, but the AND operator (*) is usually omitted and the OR operator is represented by the letter v. The logical function implied by a signal name preceded by a plus (+) sign or a minus (−) sign is true, if the line is at the upper or lower level, respectively. The boxes labelled AO are AND-OR gates, and the partitioning between the two or more AND functions is marked by extra spacing between the input ports. The latches shown are level sensitive polarity hold latches.

The timing of the coder is as follows: We assume a timing resolution of 10 intervals per byte time, indicated by clock pulses CD0, CD1, ..., CD9. We also assume that the coder feeds a commutator type Serializer, for which it is more fitting, if the leading and trailing bit groups are updated at separate times to avoid timing hazards. At the input to the coder, new data ABCDEKM must be loaded at time CD0, and FGH at time CD6. The value of the function S, the bit K, and a logical function derived from the signals NDFS6, K, and M, are stored in 3 latches at time CD6 for later use. The value of PDES6 is stored in a latch at time CD9, and PDES4 in another latch at time CD2 for use as the starting disparity PDFS6 of the next byte. The coded bits abcdei can be read out at time CD9, and fghj at time CD5.

Decoder and error checking equations, circuit example

The decoder consists of circuits to restore the original byte ABCDEFGH K, and circuits to indicate all transmission errors to the extent that they are detectable by the transmission code.

Decoder equations (Table VIII)

The necessary bit translations for decoding can be extracted from Table VI and VII and are listed in condensed form in "Table VIII. 10B/8B Decoder Equations." on page 30. The value of the decoded bits 'ABCDE FGH' is equal to the coded bits 'abcde fgh', respectively, except for the cases where there is a F(alse) entry in the table. The bit K is normally assumed to be zero.

Error checking equations

The semantics are similar to those listed for the coder under 'Disparity Control', but some of the terms also include invalid vectors not found in the tables, e.g. PDBR6 and NDBR6 also include P40 and P04, respectively, which are invalid. The FCS rules require, that the ending disparity of R6 = 000111 and R4 = 0011 be considered positive regardless of any bit counts, and the ending disparity of R6 = 111000 and R4 = 1100 are considered negative. This requires a few more gates for implementation, but is meant to give a more accurate error count. To accommodate it, we introduce a term DU which refers to an assumed block disparity. PDB is a subset of PDU. Note that PDU is not equal to NDU' or PDE.

For decoding, if the DR value in Table VI or VII for a received vector R6 or R4 does not match NDF or PDF, a disparity violation exists because of an error. The term 'DV' refers to a disparity violation: DVR6 refers to the combinations of NDFR6 and PDRR6, or PDFR6 and NDRR6; DVR4 refers to the combinations NDFR4 and PDRR4, or PDFR4 and NDRR4. The term 'INV' refers to an inherently invalid code block. Disparity violations limited to and recognizable from a single byte, such as a byte with PDUR6 followed by a vector with NDRR4, are recorded as invalid bytes (INVBY) and not included in DVR4. The output line +VIOL is at the up level if the current byte is either invalid (INVBY) or indicates a disparity violation, possibly resulting from an error in a previous byte. The following definitions hold:

PDBR6 = P31•(e+i) + P22•e•i + P40	PDBR4 = f•g•(h+j) + (f+g)•h•j
PDUR6 = PDBR6 + d•e•i	PDUR4 = PDBR4 + h•j
NDRR6 = PDBR6 + a•b•c	NDRR4 = PDBR4 + f•g
NDBR6 = P13•(e'+i')+P22•e'•i' + P04	NDBR4 = f'•g'•(h'+j') + (f'+g')•h'•j'
NDUR6 = NDBR6 + d'•e'•i'	NDUR4 = NDBR4 + h'•j'
PDRR6 = NDBR6 + a'•b'•c'	PDRR4 = NDBR4 + f'•g'
PDFR6(n+1) = PDER4(n)	NDFR6 ≡ PDFR6'
PDER6 = PDFR6•NDUR6'+PDUR6	PDER4 = PDER6•NDUR4'+PDUR4
NDER6 = NDFR6•PDUR6'+NDUR6	NDER4 = NDER6•PDUR4'+NDUR4

DVR6 = PDFR6•NDRR6 + NDFR6•PDRR6

INVR6 = P40 + P04 + P31•e•i+ P13•e' •i'

DVR4 = PDFR6•NDUR6'•NDRR4 +NDFR6•PDUR6'•PDRR4

K28 = (c=d=e=i) Kx.7 = (e≠i)•(i=g=h=j)

INVBY = INVR6+(f=g=h=j)+(e=i=f=g=h)+K28'•(i≠g=h=j)+K28•(f=g=h)
+Kx.7•PDBR6'•NDBR6'+PDUR6•NDRR4+NDUR6•PDRR6

Decoder and error checking, circuit example (Fig. 6 and 7)

An example of a CMOS implementation for the decoder and the error checking circuits is shown in "Fig. 6. 10B/8B decoder (6B/5B)" on page 35 and "Fig. 7. 10B/8B decoder (4B/3B, error checks)" on page 36. Less than half the circuits are dedicated to decoding, the rest performs error checking. This must be taken into consideration in any comparison of complexity with less constrained codes that have weaker detection capabilities. There is considerable flexibility allowed for the timing of these circuits. To reduce switching noise, especially on external interfaces, designers generally choose to reduce simultaneous switching of circuits. A workable timing grid conforming with such rules for the decoder is as follows: new bits 'abcdei' and 'fghj' are presented at the input at time CD0 and CD1, respectively; the decoded bits 'FGH' are sampled at the output at time CD7, bits 'ABC' at time CD8, bits 'DEK' and line VIOL at time CD9; the signal PDER4 is stored in a first latch at time CD9, and transferred to a second latch for use in the next byte as PDFR6 at time CD1. Additional skew between individual lines can be generated by circuit delays as appropriate.

The delay of some critical paths of both the coder and decoder can be reduced by the addition of a few extra gates in non-critical paths for a reduction of the load and of the fan-in of critical gates. If necessary, one could also reduce some clocking margins. Our implementation of a 106.25 Mbyte/s version, using 0.8 μm CMOS and only gates with the lowest power, has plenty of margin and does not require any such trade-offs.

Byte, Word, and Frame Insertion or Removal

If not properly planned for, insertions or deletions can cause problems with this and other similar codes. To code a byte properly, the RD at the front of the insertion must be known. If an insertion or deletion changes the polarity of the RD, all subsequent disparity dependent coded bytes have to be modified or recoded. For FCS, all ordered sets between frames, and the frames themselves are defined with negative starting and ending RD. Each of these sets can thus be coded with an assumed negative starting disparity, and insertion or removal has no effect on the succeeding bytes.

For the general case, it is useful to know that balanced, disparity-independent 10B vectors, as shown in Fig. 2, Group A, can always be inserted or removed without regard to RD considerations. There are 18x4 = 72 such bytes, or the equivalent of a little more than 6 information bits per byte.

If the starting RD is known, one can include all 134 balanced data bytes, and the 7 balanced special characters, enough to code the equivalent of 7 data bits into a 10 bit byte. Alternatively, one might just repeat the byte, inserting 2 bytes for each byte of information. Any pair of repeated source bytes is always balanced in the coded domain. The extra constraints and redundancy provide additional error protection, which may be welcome for such isolated pieces of information.

Idle Sequence

For FCS, the time between frames is filled with 4 byte Idles (K28.5, D21.4, D21.5, D21.5). The sequence starts and ends with negative RD. The aim is to maintain all the settings in the transmitter and receiver, including the bit and word clock alignments, at their normal level. Idles may be added or deleted at network nodes, but rules exist so there are never less than 2 Idles between frames. Since receivers usually acquire initial bit and word synchronization while Idles are being received, an Idle sequence with many transitions is preferred. The FCS Idle has 32 transitions, an average of 8 per byte, which is significantly more than the average of 60.5 transitions for 100 bytes of coded random data.

To evaluate the suitability of the Idle sequence for low cost clock recovery, the signal power spectrum as typically presented after processing to the clock recovery circuits has been computed for many Idle sequences with similar structure as the specified FCS sequence. For this signal, a strong harmonic frequency at the bit rate and much lower components in the immediate neighborhood are desirable. Since the Idle sequence is repeated every 40 bits, the frequency spectrum components appear at 2.5% intervals from the reference frequency, which is equal to the coded bit rate. The power of each of the 8 harmonic frequency components within $\pm 10\%$ of the reference is down by a factor of at least 5.5 for the FCS Idle sequence. This Idle, when repeated, includes a RL of 3, followed by a RL of 5 (the comma), and 8 bits later another RL of 3. Other potential Idles with a RL of 2 followed by an RL of 5, and three RL's of 2 spread around the trailing bytes improve the above ratio from 5.5 to 7.5, and so are less likely to result in false phase lock during bit sync acquisition. A typical phase locked loop has been shown to synchronize about 10% faster with the alternate Idles, which was judged insufficient to warrant a change in the FCS specification.

Comma

The comma has two main functions: When synchronization is lost, it provides a marker which can be used by the deserializer circuitry for a quick synchronization; this avoids multiple attempts at synchronization except in the case of a spurious comma. Although false initial synchronization is rare, proper synchronization for most applications should always be verified. FCS provides rules for verification and detection of asynchronism. A second function of the comma is to generate a guaranteed error for any misalignment of the receiver with the word boundaries. Not all coded data bytes will generate errors on misalignment.

For FCS, the comma is present in the first of 4 bytes of all Ordered Sets. Ordered Sets are defined for the Start and End of Frame delimiters, for the Idle word, and for other repetitive primitive sequences. These signals all start with negative disparity, except the End of Frame delimiter which may start with either polarity. Therefore, the synchronization circuitry may be simplified by searching for just one polarity of the comma, i.e. the polarity starting with negative RL (0011111). If the word alignment is lost in one frame, and if we fail to synchronize on the End of Frame delimiter, there are still a minimum of three other ordered sets left to achieve verified synchronization by the start of the next frame. This assumes that the byte synchronization circuitry of the serial to parallel converter can present the first comma character, which is used to establish tentative synchronization, uncorrupted at the output to the next level of control circuitry.

We also note that the second bit of the comma can be ignored for establishing a tentative alignment, thus the circuitry must just look for a match of the '0x11111' bit pattern. As was pointed out previously, a more extensive pattern match such as 0011111010 is just more costly and does not noticeably improve performance. This is especially so, because for spurious runs of 5 one bits, the leading and trailing neighborhood is more likely to contain zeros than ones, so on average an extra bit reduces the likelihood of false sync by less than the expected factor of two.

Bit Manipulation

Some link level architectures, such as the Token Ring, require easy and quick on-the-fly access to the coded bit level to recognize and change a token indicator, a reservation bit, a frame copied bit, and others, or to read and increment hop counts. For network performance reasons, the node delay must be minimized. For an mB/nB block code, the value of an individual bit generally cannot be determined or changed without reference to the entire block. One would thus expect, that block codes on token rings would lead to node delays at least as long as the blocks. Some authors [Keller et al., 1983] and committees [Moulton, 1983] have erroneously concluded that for this reason, block codes as presented here are not well suited for such applications.

For the FCS code and some other codes with similar properties, the problem can be overcome by restricting the coded set in the critical header and trailer fields to balanced disparity-independent vectors [Widmer, 1987] as represented by Group A of Fig. 2. An inspection of Table III and Table IV reveals that for these code points A = a, B = b, ... , H = h always holds. For a vector of Group A, the 'i' bit provides odd parity for the 6B field, and the 'j' bit even parity for the 4B field. For this restricted code set any pair of complementary bits confined to a 6B or 4B subblock can be inverted without affecting the meaning of any other bits, if no Group B vector is generated.

The bit patterns of the Manchester code are a subset of Group A and can never generate group B vectors. We can thus code a 6B or a 4B field in Manchester code, and remain totally compatible with the FCS code. These fields can be either read and manipulated directly in the coded domain, or through the 8B/10B coder/decoder, whichever is more convenient. The reduction in coding efficiency for a single 6B or 4B field is not an issue if it has been planned for in the frame architecture.

Error Protection

FCS protects all frames with a sturdy 4 byte cyclic redundancy check. The additional monitoring of transmission code violations serves 3 purposes:

1. Quick detection of the out-of-synchronization condition while transmitting data, and verification of correct byte alignment
2. Monitoring of the link error rate
3. Enhancement of the frame validity check to support the goal of never accepting an invalid frame

The four-byte start and end of frame delimiters, which both include a comma, help to reliably mark the start and end of a frame. It is well known that a CRC is less effective, if start or end are not detected properly, or if many widely separated errors exist. On the other hand, the transmission code is weakest in detecting closely spaced complementary pairs of errors, and strongest for widely separated errors and large error bursts. Therefore, it ideally complements the detection capability of the CRC.

For some applications it is desirable to give some extra protection to a small field in the frame, either because it is for some reason not included in the CRC check, or one wants to verify a segment, e.g. an address, without having to wait for the CRC at the end of the frame. If the 4B vector preceding the protected bytes, and the 6B vector following it, are restricted to those other than Group A of Fig. 2, any odd number of errors in the protected field will show up either as an invalid character, or a disparity violation, no later than the succeeding 6B vector. The leading 4B vector prevents a preceding error to show up as a disparity violation in the protected field.

Alternatively, if we pack 5 Manchester coded bits into a 10 bit byte, as described previously, and let the 5 bits represent 4 data bits and 1 parity bit, we can accomplish single bit error correction. These basic methods can be tailored to specific applications.

Low frequency wander

A single pole low frequency cut-off point at 0.1% of the coded bit rate produces a worst case eye closure of 0.25 dB; for a cut-off point at 1% the penalty is 2.21 dB. For further details refer to Table II and the associated explanations under "Low frequency wander (Table II)" on page 8.

Conclusion

For the development of the simple 8B/10B codes, a trellis diagram gives considerably more insight and guidance than large bit tables usually obtained by computer searches. A comparison of the partitioned FCS transmission code with the best non-partitioned 8B/10B code, shows the latter with an advantage in run length and transition density, which help the receiver clock designer. The partitioned FCS code has better low frequency behavior, which favors the preamplifier design. Also, it is significantly less costly to realize in hardware, as demonstrated by circuit examples using standard books from current production CMOS technology, which readily operate under worst case conditions at well over 100 Mbyte/s. The FCS code is more flexible for special adaptations and has several useful secondary features.

Acknowledgments

I am indebted to J.D. Crow and R.J.S. Bates for encouragement and guidance. The eye closure penalties have been calculated with a tool developed by R.J.S. Bates. The Idle sequence has been analyzed with tools from J.F. Ewen. R.P. Rizzo modelled the pull-in time for various Idle sequences. I thank B.D. Parker for the simulation and verification of the coder and decoder circuits, and K.R. Wrenner for helpful suggestions.

References

X3T9 Technical Committee. *Fibre Channel, Physical and Signaling Interface (FC-PH), Rev. 3.0. FC-P/92-001R3.0.* American National Standards Institute, 1430 Broadway, New York, NY 10018, 1992. Available from: Global Engineering, 2805 McGaw St., Irvine, CA 92714

Martin W. Sachs. Fiber Channel Standard. IBM Research Report, RC 18365, Yorktown Heights, NY. September 1992.

A. X. Widmer and P. A. Franaszek. Transmission Code for High-Speed Fibre-Optic Data Networks. *Electronics Letters*, 19(6):202-203, March 1983.

A. X. Widmer and P. A. Franaszek. A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code. *IBM Journal of Research and Development*, 27(5):440-451, September 1983.

P. A. Franaszek. Coding for constrained channels: A comparison of two approaches. *IBM Journal of Research and Development*, 33(6):602-608, November 1989.

R. D. Cideciyan. Attractive Line Codes for High-Speed LANs and MANs and their Realization. Proc. IEEE Int'l Conf. on Communications "ICC 93", 5413, Geneva, Switzerland. May 1993.

F. E. Ross. An Overview of FDDI: The Fiber Distributed Data Interface. *IEEE Journal on Selected Areas in Communications*, 7(7):1043-1051, September 1989.

B. Guo, E. Gershon, G. Arakaki and J. Kubinec. Error contribution of DC imbalance for the 4B/5B and 8B/10B codes. *Communication Systems: Towards Global Integration, Singapore ICCS'90 Conference Proceedings*, 1(P1.5.1-1.5.5):21-25, Elsevier, Amsterdam, Netherlands, 1990.

J.J. Stiffler. *Theory of Synchronous Communications.* Prentice Hall Inc., Englewood Cliffs, NJ, 1971.

J. M. Griffiths. Binary Code Suitable for Line Transmission. *Electronics Letters*, 5(4):79-81, IEE, February 1969.

W. Bux, et al. Architecture and Design of a reliable Token-Ring Network. *IEEE Journal on Selected Areas in Communications*, SAC-1(5):561-765, Nov. 1983.

G.N.N. Martin. A Rate 8/10 DC Balanced Code with Local Parity. *IBM Technical Disclosure Bulletin*, 27(9):5272-5279, February 1985.

D. H. McMahon, A. A. Kirby, B. A. Schofield and K. Springer. U.S. Pat. No. 5,144,304. *Data and Forward Error Control Coding Techniques for Digital Signals.* Sep. 1, 1992.

R. W. Stevens. U.S. Pat. No. 5,025,256. *Data Transmission Code.* June 18, 1990.

E. A. Zurfluh, et al. The IBM Zurich Research Laboratory's 1.13 Gb/s LAN. IBM Research Report, RZ 2278, Zurich. February 1992.

R. D. Cideciyan. High-Speed Realization of Modulation Codes for Optical Fiber Transmission. IBM Research Report, RZ 2295, Zurich. March 1992.

IBM Corp., Enterprise Systems Architecture/390, ESCON I/O Interface, number SA22-7202-02, August 1992.

H. J. Keller, H. Meyr and H. R. Mueller. Transmission Design Criteria for a Synchronous Token Ring. *IEEE Journal on Selected Areas in Communications*, SAC-1(5):721-733, Nov. 1983.

R. Kirk Moulton. ANSI-FDDI Committee, Summary of ANSI Working Meeting 9/26/83, 1983.

A. X. Widmer. U.S. Pat. No. 4,665,517. *Method of Coding to minimize Delay at Communication Node.* May 12, 1987.

Tables

Table I. No. of vectors of disparity DB for nB bits

DB	1B	2B	3B	4B	5B	6B	7B	8B	9B	10B
+10	-	-	-	-	-	-	-	-	-	1
+9	-	-	-	-	-	-	-	-	1	0
+8	-	-	-	-	-	-	-	1	0	10
+7	-	-	-	-	-	-	1	0	9	0
+6	-	-	-	-	-	1	0	8	0	45
+5	-	-	-	-	1	0	7	0	36	0
+4	-	-	-	1	0	6	0	28	0	120
+3	-	-	1	0	5	0	21	0	84	0
+2	-	1	0	(4)	0	(15)	0	56	0	(210)
+1	1	0	3	0	10	0	35	0	126	0
+0	0	2	0	(6)	0	(20)	0	70	0	(252)
-1	1	0	3	0	10	0	35	0	126	0
-2	-	1	0	(4)	0	(15)	0	56	0	(210)
-3	-	-	1	0	5	0	21	0	84	0
-4	-	-	-	1	0	6	0	28	0	120
-5	-	-	-	-	1	0	7	0	36	0
-6	-	-	-	-	-	1	0	8	0	45
-7	-	-	-	-	-	-	1	0	9	0
-8	-	-	-	-	-	-	-	1	0	10
-9	-	-	-	-	-	-	-	-	1	0
-10	-	-	-	-	-	-	-	-	-	1

Table II. Eye closure penalty in dB

High ~~Low~~ pass cut-off f/f0	Pattern lengths (bytes)				
	4	12	50	100	200
0.01%	0.01	0.01	0.01	0.02	0.02
0.02%	0.03	0.03	0.03	0.03	0.04
0.05%	0.07	0.07	0.09	0.10	0.12
0.10%	0.14	0.16	0.20	0.23	0.24
0.25%	0.37	0.44	0.58	0.59	0.59
0.50%	0.80	1.02	1.16	1.16	1.16
1.00%	1.79	2.20	2.21	2.21	2.21
2.00%	3.77	4.09	4.08	4.08	
4.00%	6.90	6.96	6.96	6.96	

Table III. 5B/6B Coding

Name	ABCDE K	Classifications		DR abcdei	DB abcdei
		Bit coding	Disparity	Primary	Alternate
D0	00000 0	L04	L22'•L31'•E'	+ 011000	− 100111
D1	10000 0	L13•E'	L22'•L31'•E'	+ 100010	− 011101
D2	01000 0	L13•E'	L22'•L31'•E'	+ 010010	− 101101
D3	11000 0	L22•E'		± 110001	0
D4	00100 0	L13•E'	L22'•L31'•E'	+ 001010	− 110101
D5	10100 0	L22•E'		± 101001	0
D6	01100 0	L22•E'		± 011001	0
D7	11100 0		L31•D'•E'	− 111000	0 000111
D8	00010 0	L13•E'	L22'•L31'•E'	+ 000110	− 111001
D9	10010 0	L22•E'		± 100101	0
D10	01010 0	L22•E'		± 010101	0
D11[1]	11010 0	L31•D•E'		± 110100	0
D12	00110 0	L22•E'		± 001101	0
D13[1]	10110 0	L31•D•E'		± 101100	0
D14[1]	01110 0	L31•D•E'		± 011100	0
D15	11110 0	L40	L22'•L31'•E'	+ 101000	− 010111
D16	00001 0	L04, L04•E	L22'•L13'•E	− 011011	+ 100100
D17[2]	10001 0	L13•D'•E		± 100011	0
D18[2]	01001 0	L13•D'•E		± 010011	0
D19	11001 0			± 110010	0
D20[2]	00101 0	L13•D'•E		± 001011	0
D21	10101 0			± 101010	0
D22	01101 0			± 011010	0
D/K23	11101 x		L22'•L13'•E	− 111010	+ 000101
D24	00011 0	L13•D•E	L13•D•E	+ 001100	− 110011
D25	10011 0			± 100110	0
D26	01011 0			± 010110	0
D/K27	11011 x		L22'•L13'•E	− 110110	+ 001001
D28	00111 0			± 001110	0
K28	00111 1	L22•K	K	− 001111	+ 110000
D/K29	10111 x		L22'•L13'•E	− 101110	+ 010001
D/K30	01111 x		L22'•L13'•E	− 011110	+ 100001
D31	11111 0	L40, L40•E	L22'•L13'•E	− 101011	+ 010100

[1] S = 1 for PDFS6 (RD = +1)

[2] S = 1 for NDFS6 (RD = −1)

Table IV. 3B/4B Coding

NAME	FGH	K	Classifications		DR	fghj	DB	fghj
			Bit coding	Disparity	Primary			Alternate
D/Kx.0[3]	000	x	F'•G'•H'	F'•G'	+	0100	−	1011
Dx.1	100	0	(F≠G)•H'		±	1001	0	
K28.1	100	1	(F≠G)•H'	(F≠G)•K	+	1001	0	0110
Dx.2	010	0	(F≠G)•H'		±	0101	0	
K28.2	010	1	(F≠G)•H'	(F≠G)•K	+	0101	0	1010
D/Kx.3[3]	110	x		F•G	−	1100	0	0011
D/Kx.4[3]	001	x		F'•G'	+	0010	−	1101
Dx.5	101	0			±	1010	0	
K28.5	101	1		(F≠G)•K	+	1010	0	0101
Dx.6	011	0			±	0110	0	
K28.6	011	1		(F≠G)•K	+	0110	0	1001
Dx.P7	111	0		F•G, F•G•H	−	1110	+	0001
Dx.A7[5]	111	0	F•G•H•S	F•G, F•G•H	−	0111	+	1000
Ky.A7[4]	111	1	F•G•H•K	F•G, F•G•H	−	0111	+	1000

[3] Kx is restricted to K28.

[4] Ky is restricted to K23, K27, K28, K29, K30.

[5] S = L31•D•E'•PDFS6 + L13•D'•E•NDFS6.

Table V. Special Characters (K = 1)

NAME	ABCDE	FGH	K	DR	abcdei	fghj	DB	DR	abcdei	fghj	DB
				Primary				Alternate			
K28.0	00111	000	1	−	001111	0100	0	+	110000	1011	0
K28.1*	00111	100	1	−	001111	1001	+	+	110000	0110	−
K28.2	00111	010	1	−	001111	0101	+	+	110000	1010	−
K28.3	00111	110	1	−	001111	0011	+	+	110000	1100	−
K28.4	00111	001	1	−	001111	0010	0	+	110000	1101	0
K28.5*	00111	101	1	−	001111	1010	+	+	110000	0101	−
K28.6	00111	011	1	−	001111	0110	+	+	110000	1001	−
K28.7*[6]	00111	111	1	−	001111	1000	0	+	110000	0111	0
K23.7	11101	111	1	−	111010	1000	0	+	000101	0111	0
K27.7	11011	111	1	−	110110	1000	0	+	001001	0111	0
K29.7	10111	111	1	−	101110	1000	0	+	010001	0111	0
K30.7	01111	111	1	−	011110	1000	0	+	100001	0111	0

* Singular comma character for byte synchronization.

[6] K28.7 must not be followed by D3, D11, D12, D19, D20, D/K28.

Table VI. 6B/5B Decoding

NAME	abcdei	Decoding class	Disparity	ABCDE	K^7	DR	DB
D0	011000	P22•b•c•(e=i)	P22•e'•i'	00000	0	+	−
D0	100111	P22•b'•c'•(e=i)	P22•e•i	00000	0	−	+
D1	100010	P13•i'	P13•i'	10000	0	+	−
D1	011101	P31•i	P31•i	10000	0	−	+
D2	010010	P13•i'	P13•i'	01000	0	+	−
D2	101101	P31•i	P31•i	01000	0	−	+
D3	110001			11000	0	±	0
D4	001010	P13•i'	P13•i'	00100	0	+	−
D4	110101	P31•i	P31•i	00100	0	−	+
D5	101001			10100	0	±	0
D6	011001			01100	0	±	0
D7	111000		P31•d'•e'•i'	11100	0	−	0
D7	000111	P13•d•e•i	P13•d•e•i	11100	0	+	0
D8	000110	P13•i'	P13•i'	00010	0	+	−
D8	111001	P31•i	P31•i	00010	0	−	+
D9	100101			10010	0	±	0
D10	010101			01010	0	±	0
D11	110100			11010	0	±	0
D12	001101			00110	0	±	0
D13	101100			10110	0	±	0
D14	011100			01110	0	±	0
D15	101000	P22•a•c•(e=i)	P22•e'•i'	11110	0	+	−
D15	010111	P22•a'•c'•(e=i)	P22•e•i	11110	0	−	+
D16	011011	P22•b•c•(e=i)	P22•e•i	00001	0	−	+
D16	100100	P22•b'•c'•(e=i)	P22•e'•i'	00001	0	+	−
D17	100011			10001	0	±	0
D18	010011			01001	0	±	0
D19	110010			11001	0	±	0
D20	001011			00101	0	±	0
D21	101010			10101	0	±	0
D22	011010			01101	0	±	0
D/K23	111010		P31•e	11101	x	−	+
D/K23	000101	P13•e'	P13•e'	11101	x	+	−
D24	001100	a'•b'•e'•i'	P22•e'•i'	00011	0	+	−
D24	110011	a•b•e•i	P22•e•i	00011	0	−	+
D25	100110			10011	0	±	0
D26	010110			01011	0	±	0
D/K27	110110		P31•e	11011	x	−	+
D/K27	001001	P13•e'	P13•e'	11011	x	+	−
D28	001110			00111	0	±	0
K28	001111	c•d•e•i	P22•e•i	00111	1	−	+
K28	110000	c'•d'•e'•i'	P22•e'•i'	00111	1	+	−
D/K29	101110		P31•e	10111	x	−	+
D/K29	010001	P13•e'	P13•e'	10111	x	+	−
D/K30	011110		P31•e	01111	x	−	+
D/K30	100001	P13•e'	P13•e'	01111	x	+	−
D31	101011	P22•a•c•(e=i)	P22•e•i	11111	0	−	+
D31	010100	P22•a'•c'•(e=i)	P22•e'•i'	11111	0	+	−

Table VII. 4B/3B Decoding, K Function

Name	fghj	Decoding class	Disparity	FGH	K[7]	DR	DB
D/Kx.0	0100	f'•h'•j'	f'•h'•j'	000	x	+	−
D/Kx.0	1011	f•h•j	f•h•j	000	x	−	+
D/Kx.1	1001			100	x	±	0
K28.1	0110	c'•d'•e'•i'•(h≠j)		100	1		0
D/Kx.2	0101			010	x	±	0
K28.2	1010	c'•d'•e'•i'•(h≠j)		010	1		0
D/Kx.3	1100		f•g•h'•j'	110	x	−	0
D/Kx.3	0011	f'•g'•h•j	f'•g'•h•j	110	x	+	0
D/Kx.4	0010		f'•g'•j'	001	x	+	−
D/Kx.4	1101	f•g•j	f•g•j	001	x	−	+
D/Kx.5	1010			101	x	±	0
K28.5	0101	c'•d'•e'•i'•(h≠j)		101	1		0
D/Kx.6	0110			011	x	±	0
K28.6	1001	c'•d'•e'•i'•(h≠j)		011	1		0
Dx.7	1110		f•g•h	111	0	−	+
Dx.7	0001	f'•g'•h'	f'•g'•h'	111	0	+	−
D/Kx.7	0111	g•h•j	g•h•j	111	x	−	+
D/Kx.7	1000	g'•h'•j'	g'•h'•j'	111	x	+	−

[7] K = K28 + Kx.7 = (c=d=e=i) + (e≠i)•(i=g=h=j)

Table VIII. 10B/8B Decoder Equations.

Classifications	A = a	B = b	C = c	D = d	E = e	F = f	G = g	H = h	K = 0
P22•b•c•(e=i)	-	F	F	-	-				
P22•b'•c'•(e=i)	F	-	-	F	F				
P13•i'	-	-	-	-	F				
P31•i	F	F	F	F	-				
P22•a•c•(e=i)	-	F	-	F	-				
P22•a'•c'•(e=i)	F	-	F	-	F				
a'•b'•e'•i'	-	-	F	-	F				
a•b•e•i	F	F	-	F	-				
P13•(d•i+e')+c'•d'•e'•i'	F	F	F	F	F				
f'•h'•j'						-	F	-	
f•h•j						F	-	F	
g•h•j						F	-	-	
g'•h'•j'						-	F	F	
f'•g'•h•j+f•g•j+f'•g'•h'+u[8]						F	F	F	
K[7]									F

[8] u = c'•d'•e'•i'•(h≠j) = K28•i'•(h≠j)

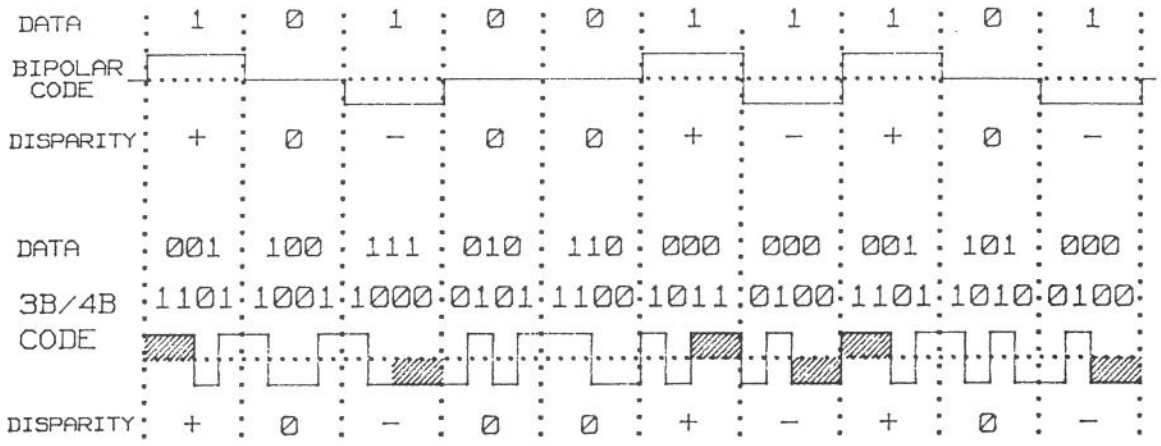

Fig. 1. Extension of bipolar code

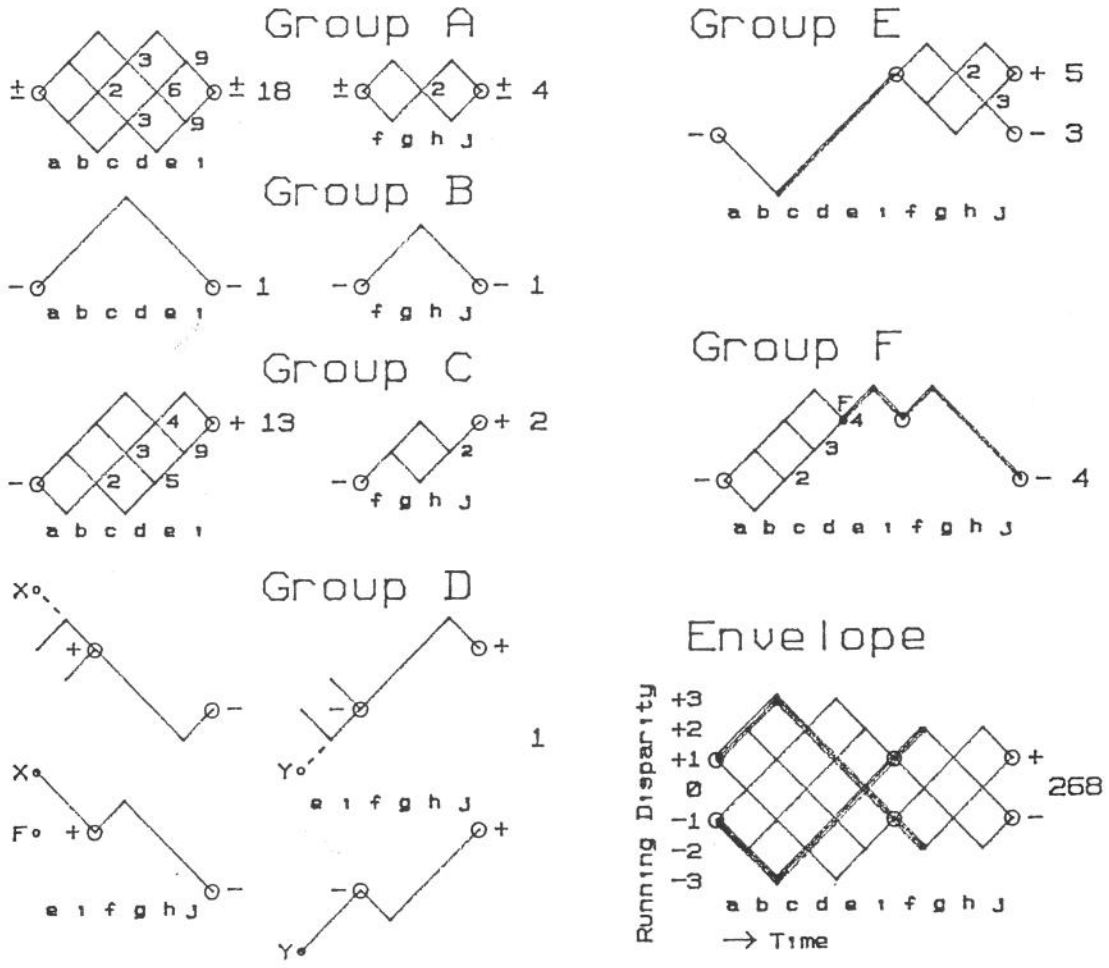

Fig. 2. Trellis for partitioned 8B/10B code

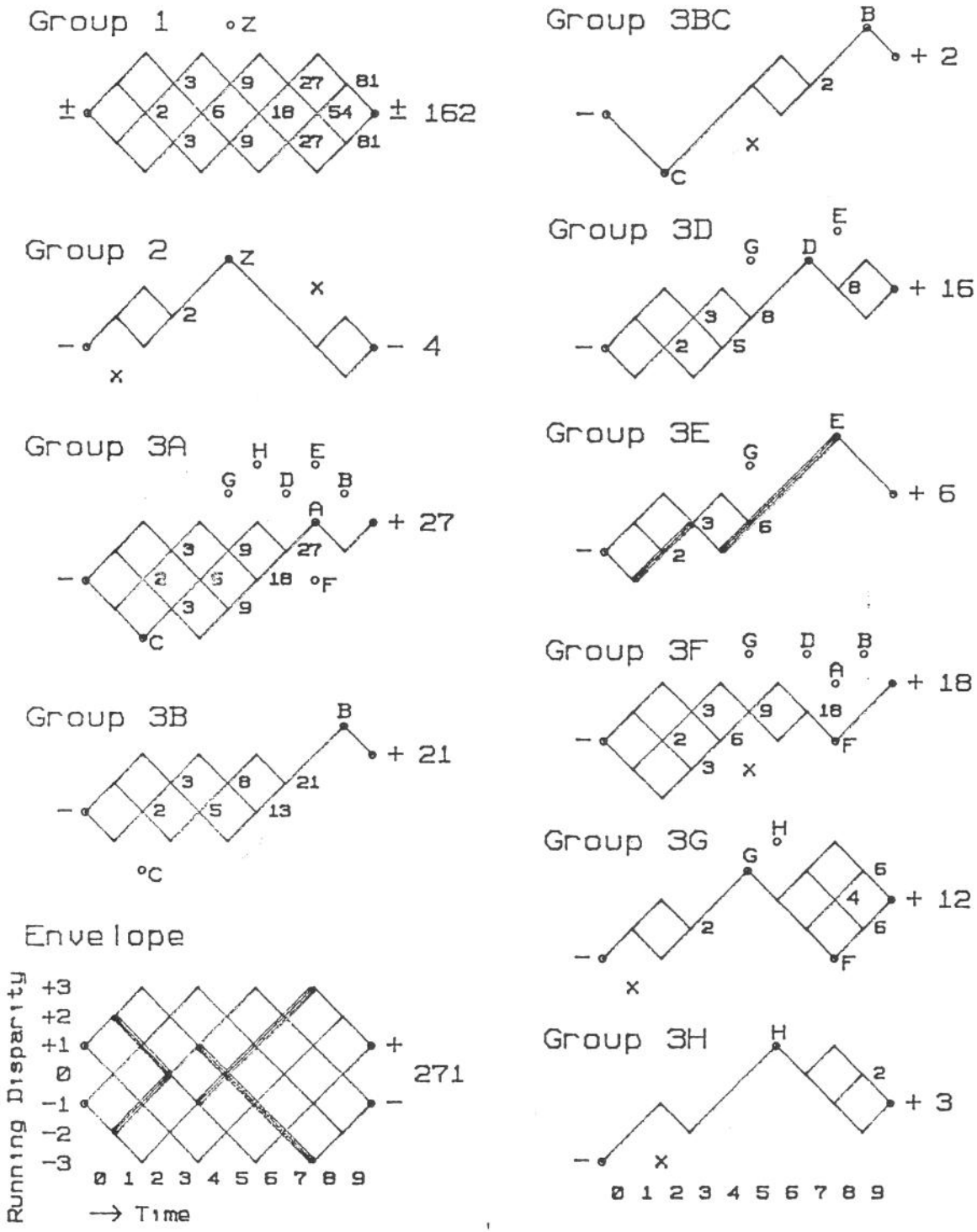

Fig. 3. Trellis for non-partitioned 8B/10B code

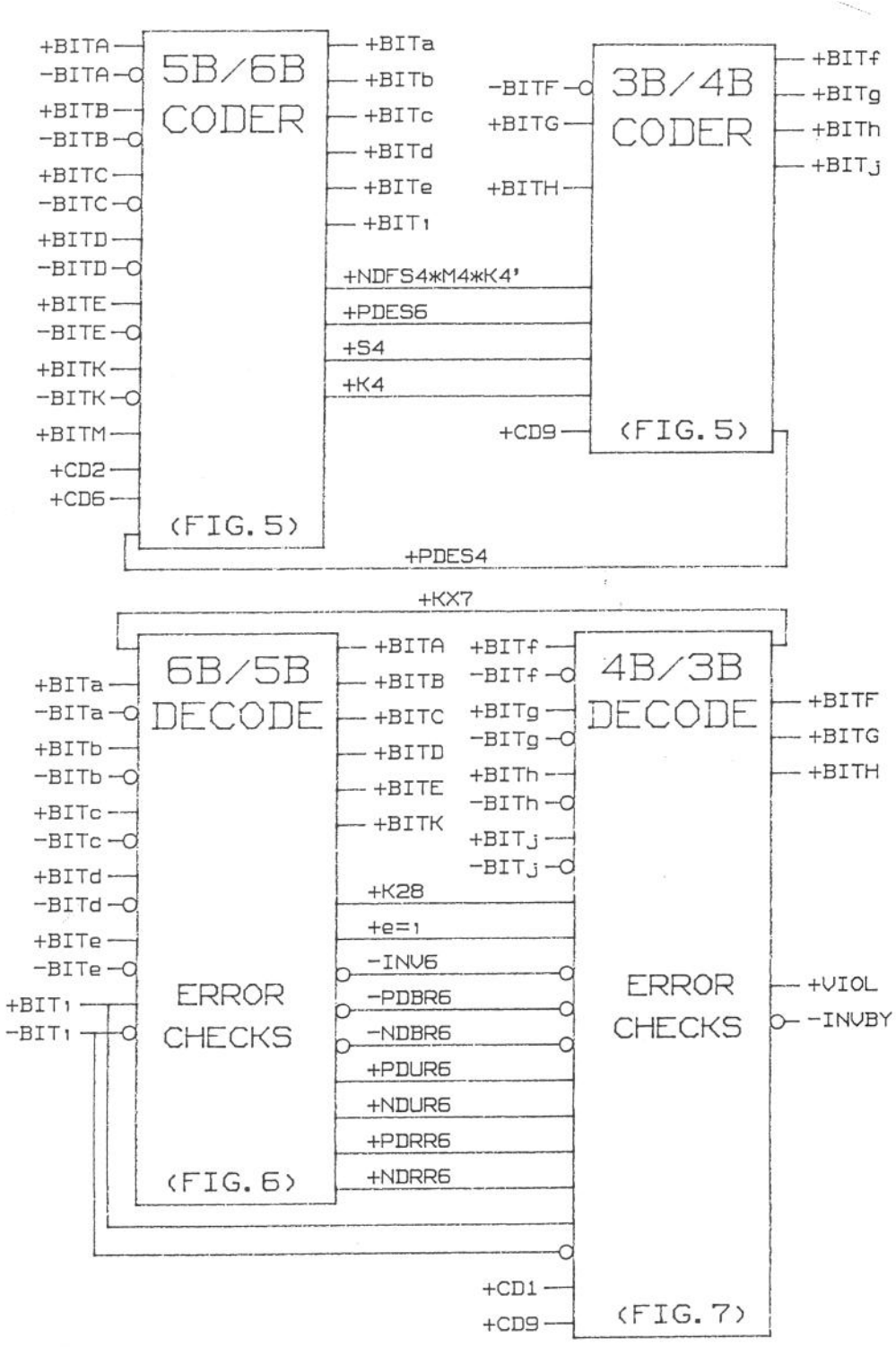

Fig. 4. Top level schematic for partitioned 8B/10B coder and decoder

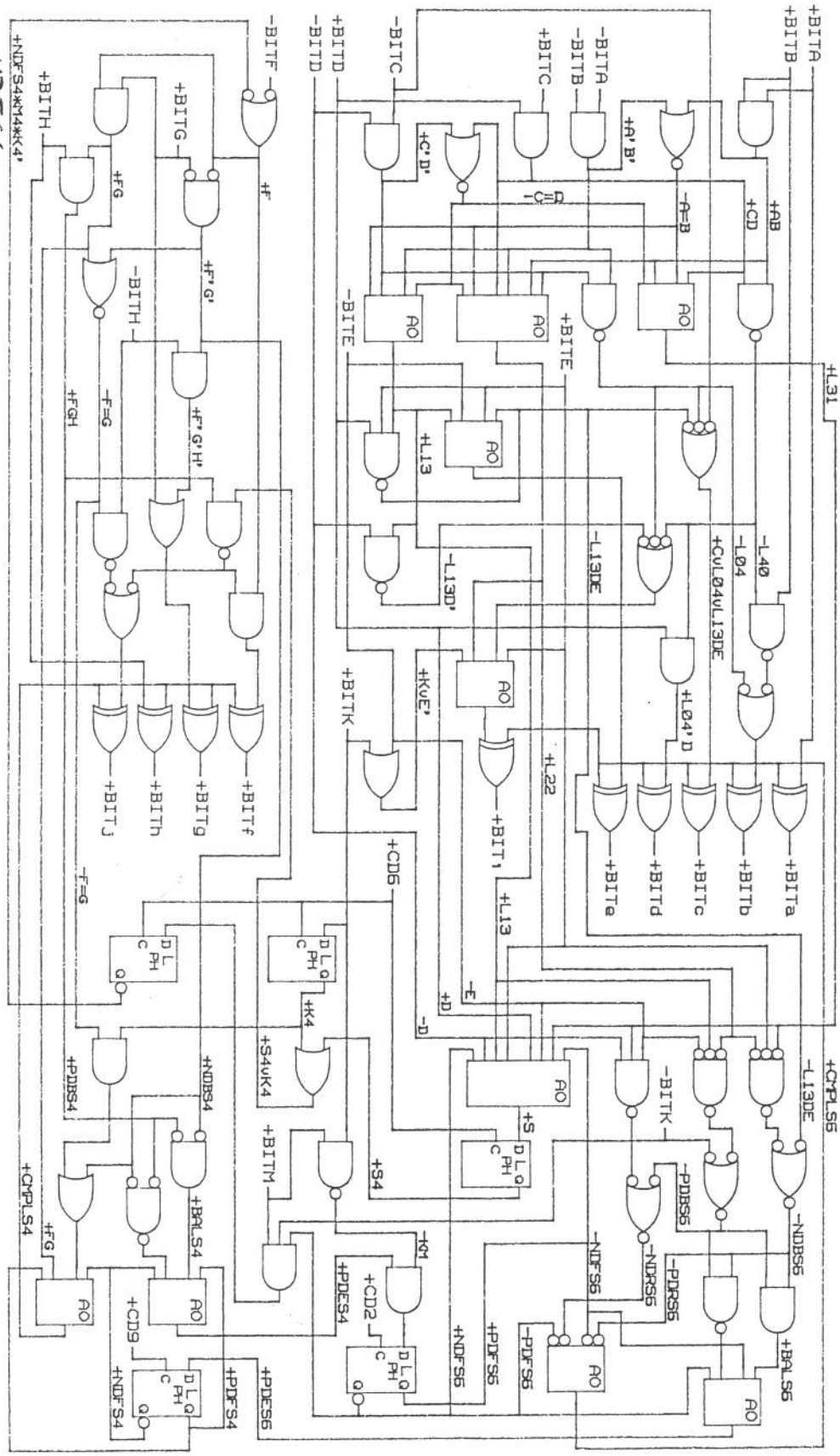

Fig. 5. Partitioned 8B/10B coder

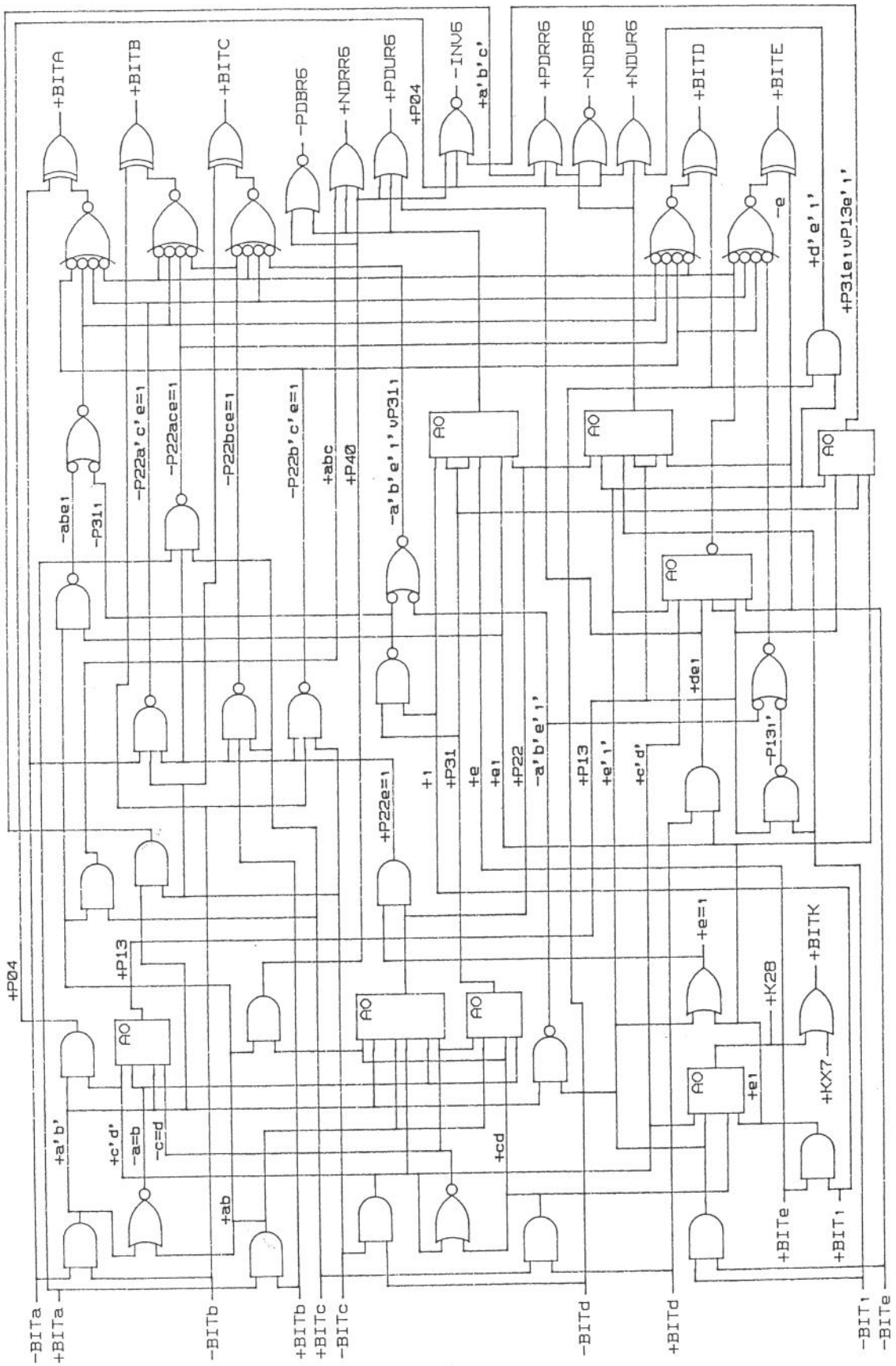

Fig. 6. 10B/8B decoder (6B/5B)

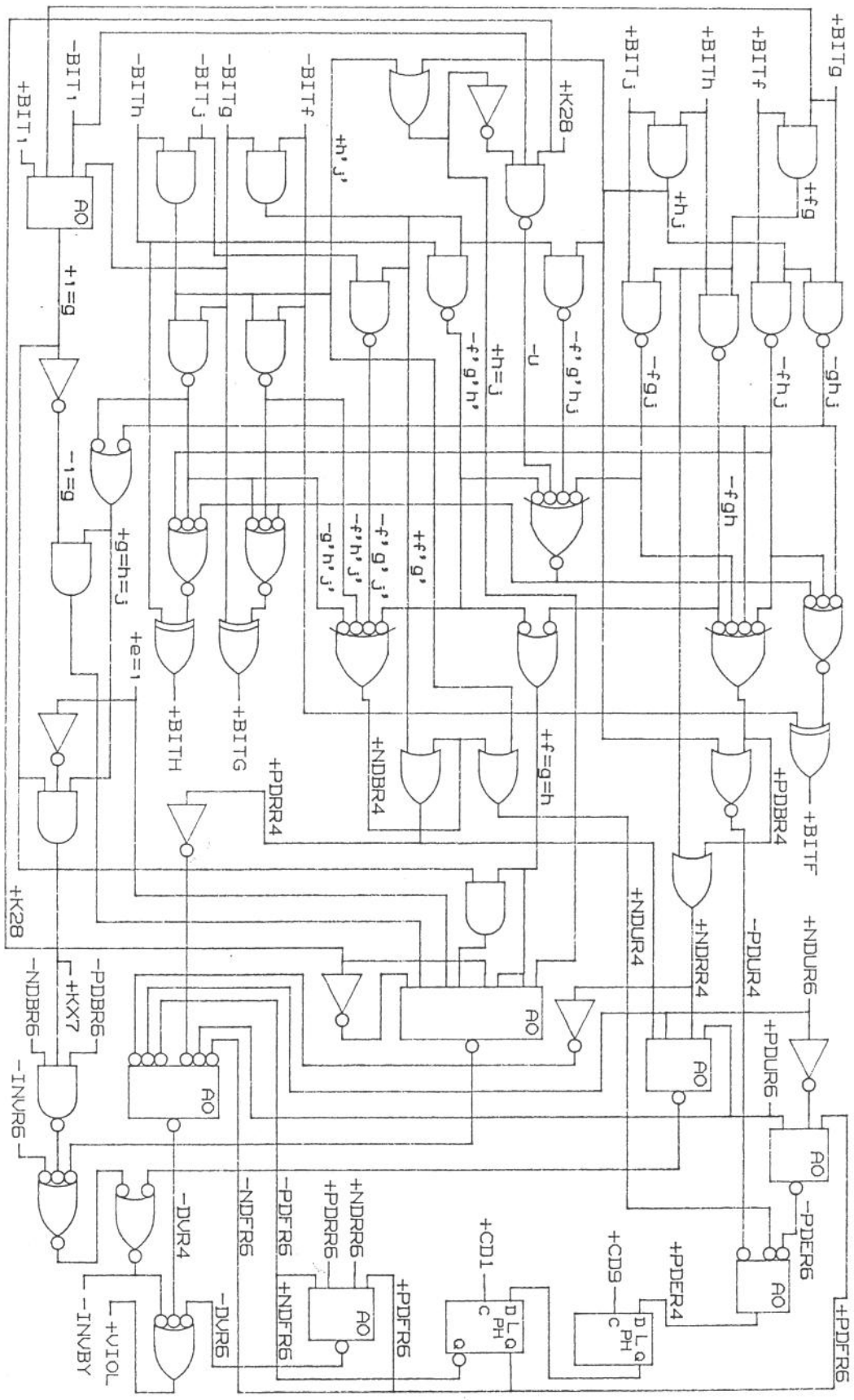

Fig. 7. 10B/8B decoder (4B/3B, error checks)

The Case for explicitly-enabled Byte Alignment Mode

In explicitly-enabled byte alignment mode, a receiver will not change the alignment unless an explicit request from a higher protocol level commands it to realign word synchronization on the next comma. In contrast, the continuously-enabled byte alignment mode, also referred to as 'Hot Sync', will realign on any comma in the bit stream.

1. In the FCS protocol it is the FC-1 level which determines, based on several code violation events, that synchronization has been lost and needs to be reestablished. Because any individual comma may be spurious, the FC-1 protocol goes through a synchronization verification process before starting data transfers to the higher protocol levels. If FC-1 has seen 3 error free ordered sets without intervening errors, the conclusion is made that the deserializer has in fact achieved correct synchronization. Each ordered set includes a comma and a total of 40 bits. 'Hot Sync' clearly counteracts and defeats the purpose of these procedures; in fact, the data presented to FC-2 may differ from the data received at the front end of the deserializer, without any code violations ever showing up, if synchronism is not maintained. There is no way to be sure that two or three ordered sets have been received without errors in between, if 'Hot Sync' is active.
2. A properly designed transmission link and receiver clock will not lose synchronization unless some rather drastic event has occurred, such as a temporary break in the link, a power interruption, or electromagnetic interference above the specified values. On the other hand, a spurious comma of one byte or less can easily be generated from data by one or more errors. Therefore, once a link is operational, a misaligned comma within a frame is most likely an error rather than an out of synchronization condition. In dealing with uncertainty, a communications receiver should always make the most likely assumptions. The continuously-enabled byte alignment mode or 'Hot Sync' violates this principle.
3. A long established goal is to never accept a false frame as valid on any link over the life of the equipment. The efficacy of cyclic redundancy checks is very much dependent on the exact demarcation of the start and end of a frame. The frame boundaries can be established with better reliability if clock and byte alignments are preserved. Given a reliable receiver clock and a 32 bit CRC, one can then be reasonably confident to meet the above goal, but much less so if any kind of 'Hot Sync' is allowed to interfere. Similar receivers for applications other than FCS may operate with only a 16 bit CRC, in which case the effects of 'Hot Sync' are severely detrimental.
4. There may be uncompressed image, voice, or other real time packets transported on the links that can or must tolerate errors without retransmission. In such situations 'Hot Sync' will generate large defects from barely noticeable imperfections.
5. It is more difficult to obtain a good error count with 'Hot Sync' active.
6. 'Hot Sync' introduces complexities into systems diagnosis and test.

7. A receiver design for continuously-enabled byte alignment tends to be more complex and more costly.

8. For explicitly-enabled byte alignment mode it is quite sufficient to monitor only for the bare minimum 6 or 7 bits that define the comma (e.g. '0x11111'). This saves circuits and power in the critical high speed area of the receiver and helps in achieving the highest possible data rate for a given technology. 'Hot Sync' advocates tend to compensate the exposure to false commas by monitoring in the receiver for longer sequences, e.g. 10 bits. However, this does not improve an inherently bad situation sufficiently, especially if one considers how false commas can be generated from data. It can readily be shown that the bits following a spurious comma are not random. For instance the bits following a false 0011111 comma sequence are more likely to be zeros than ones, so monitoring for the full K28.5 character (0011111010) will reduce the incidence of false commas by a factor of substantially less than eight that one would expect for random data. But even with a factor of eight, the incidence of spurious commas disturbing the existing byte alignment is still far too high to ignore. Also, if the receiver synchronizes on a full 10 bit comma character, compatibility with future expansions of the architecture, that may include the other two comma characters, is compromised.

9. In summary, 'Hot Sync' has identifiable adverse operational and cost consequences and so far no advantages for the FCS application have been identified, except compatibility with some existing hardware.