Low Overhead Block Coding for Multi-Gb/s Links

Alistair Coles, David Cunningham
Extended Enterprise Laboratory
HPL-98-168
October, 1998

block coding,
16B18B,
fibre optic

An 8B10B block coding scheme is currently employed in several fibre optic applications operating at approximately 1Gb/s. This code results in a symbol rate that is 25% greater than the data rate (i.e. 25% overhead). We describe an alternative block coding scheme which may be used to implement lower overhead coding for fiber optic multi-Gb/s links. The coding scheme is extremely simple to implement, generates a d.c. balanced signal and provides non-data codewords for control signaling. As an example, a 16B18B code (overhead 12.5%) is described.

Internal Accession Date Only

1 Introduction

A block code is often used to condition signals before transmission on fibre optic links. For example, the Fibrechannel [2] and Gigabit Ethernet [3] standards both require data to be coded according to an 8B10B code described in [1]. This code ensures that the signals transmitted are d.c. balanced and contain a sufficient number of transitions to enable easy clock recovery. In addition the code described in [1] is relatively simple to implement, requiring only several hundred logic gates. Although this code has very favourable properties, it does require the rate of transmission to be 25% greater than the data rate. We shall refer to this rate increase as the coding overhead:

$$\text{Overhead} = \frac{\text{transmission rate} - \text{data rate}}{\text{data rate}} \times 100\% \quad (1)$$

In this paper we describe a coding technique that may be used to achieve lower overhead while still addressing the requirements for links operating at multi-Gb/s over optical fiber channels. The requirements for these links are summarized below:

- minimum coding overhead: As data rates on fibre optic links progress beyond 1 Gb/s to for example 2.5 Gb/s, CMOS technology will be stretched to the limit of its performance. In order to maximize data throughput, as little increase in speed as possible should be incurred by coding the data. The primary objective of this coding proposal is therefore to reduce the coding overhead, and other requirements have been traded off to meet this.
- byte alignment: the input to the coder should align naturally with the data path width.
- control space: the code should provide some redundant space for control signaling.
- error detection: the code should not interfere with the error detection properties of common cyclic redundancy checks (e.g. CRC32). Some degree of error detection within the code itself is desirable.
- d.c. balance: often this is the primary objective of coding. In this case it is assumed that a relatively large disparity, and relatively long runlengths are tolerable (cf. runlengths > 70 are known to occur in SONET systems).
- simplicity: gate count should be minimized.

2 Code operation

The coding scheme may be used in any configuration where the number of bits input to the coder, X, is even. The code operates by attaching two extra bits to each block of X input bits. We shall refer to the X input bits as the payload and the two

extra bits as the payload label. The payload may be either data or control, as distinguished by the label. The coding procedure is illustrated in figure 1.

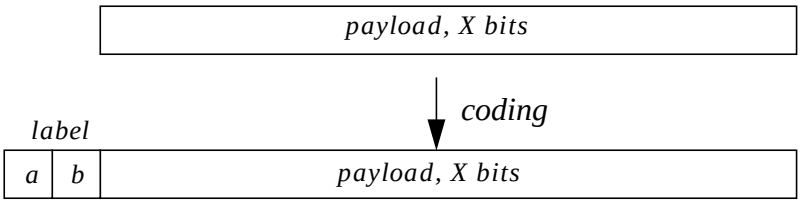

Figure 1 Coding procedure.

The code maintains dc balance (i.e. an equal number of ones and zeros transmitted) by inverting the payload when necessary, and indicating that inversion has occurred through the label. The value of the two label bits is determined as follows. The weight of any payload (i.e. the difference between the number of ones and the number of zeros) is examined for each payload and the following actions taken:

- If the weight is non-zero and of equal sign to the running disparity at the end of the last coded block, then the payload is inverted and the label bits are set to 00. The weight of the output codeword (payload plus label) is either zero or has opposite sign to the running disparity.
- If the weight is non-zero and of opposite sign to the running disparity at the end of the last coded block, then the payload is not inverted and the label bits are set to 11. The weight of the output codeword (payload plus label) is either zero or has opposite sign to the running disparity.
- If the weight of a data payload is zero (i.e. equal number of ones and zeros) then the payload is not inverted and the label bits are set to 01 to indicate that the payload is weight zero data. Note that in this case the weight of the output codeword (payload plus label) is also zero.

These rules guarantee that the running disparity is bounded.

Control information may be passed as zero weight payloads. The label bits are set to 10 which distinguishes the control information from data, and ensures that the weight of the output codeword (payload plus label) is also zero.

To summarize, the label bits are used as shown in table 1.

Table 1: Coding rules

Payload	Wt(payload)	rds	a	b	Invert?
Data	>0	<0	1	1	N
Data	>0	>0	0	0	Y
Data	<0	<0	0	0	Y
Data	<0	>0	1	1	N
Data	0	-	0	1	N
Control	0 by design	-	1	0	N

Note 1: The cases when the rds is zero are avoided by initializing the rds to either +1 or -1, remembering that all coded blocks have even length and therefore even weight, so that the rds never becomes zero at a coded block boundary.

Note 2: In the case when the data payload is weight -2 (i.e. two less ones than zeros) and the rds is <=0, the payload is inverted resulting in a weight of +2. However, the label bits are set to 00 resulting in the weight of the coded block becoming zero. This example illustrates that payload inversion does not always reduce the absolute value of the rds, but prevents the absolute value of the rds from increasing.

3 Code properties

3.1 Overhead:

The code overhead is 2 bits for each X bits i.e. 2/X x 100%.

3.2 Byte alignment:

This is easily achieved by picking X to be a multiple of eight.

3.3 Control space:

The size of the control space is equal to the number of weight zero payloads that may be formed with length X. This is equal to:

$$C = \frac{X!}{\left(\left(\frac{X}{2}\right)!\right)^2} \qquad (2)$$

which grows rapidly with X. e.g. for X=8, C=70 and for X=16, C=12870.

In practice a simple (albeit inefficient) scheme which generates control payloads with zero weight is to pick any X/2 bit symbol Y to represent a control state, and set the payload equal to the concatenation of Y and $\overline{Y}$. For X=16 this provides 256 control states.

3.4 Error detection:

Error events may be classified according to the number of bits errored within either the payload or the label.

3.4.1 Data payload errors

A single error in a payload results in a single error in the decoded data. Furthermore, any number of errors within the payload will result in the same number of errors in the decoded data. This is a useful property as it means that the errors are not multiplied. The error detection properties of any CRC that is used will remain unchanged for errors within the payload.

3.4.2 Control payload errors

Any odd number of errors within a control payload field will always be detected as they will result in a non-zero control payload weight, which is disallowed by design.

3.4.3 Label errors

A single error in the label field will always be detected as it will result in a label associated with a weight zero payload being created for a non-weight zero payload or vice versa.

3.4.4 Combined label and payload errors

A single error in the label field combined with a single error in the associated data payload field can result in either a single error in the decoded data (label 01 changed to label 11, payload error changes payload weight to non-zero or vice versa) or can result in all X decoded data bits being complemented (label 01 changed to label 00, payload error changes payload weight to non-zero or vice versa).

A single error in the label field combined with a single error in the associated control payload field can result in the control payload being mistakenly decoded as data (label 10 changed to label 11, payload error changes payload weight to non-zero or vice versa).

Two errors in the label field associated with a data payload will result in all X decoded data bits being complemented (label field 11 changed to 00 or vice versa), or will result in the data payload being mistaken for a control payload (label field 01 changed to 10).

Two errors in the label field associated with a control payload will result in the control being misinterpreted as data (label field 10 changed to 01).

3.4.5 Summary of error detection properties

In summary, a single error anywhere in the coded block is either detected or results in a single error in the decoded data.

Two errors anywhere in the coded block can result in either a single decoded data error, or a complemented decoded data payload, or a data payload being mistaken for control, or vice versa.

3.5 d.c. balance:

The coding rules cause the running digital sum to be bounded. The running disparity is constrained to +/- (1.5X+2). The running digital sum at codeword boundaries is constrained to +/-(X+1).

The maximum run length (i.e. maximum run of identical symbols) that may occur within a sequence of codewords is 2.5X + 2. An example of how this runlength occurs is shown in Table 2.

Table 2: Maximum runlength example

rds	-(X+1)	-1	X+1
payload weight	X-2	X	-2
label	11	11	11
coded block	$11\text{-}01^{X-1}$	$11\text{-}1^{X}$	$11\text{-}1^{(x/2)-1}0^{(x/2)+1}$

Note 1: 1^{X} denotes a run of x ones.

For the case when X=16, we have a maximum runlength of 42. Clearly this is relatively large, but is the price paid for low coding overhead and simplicity.

3.6 Simplicity:

The scheme is extremely simple to implement. No coding logic or lookup table is required other than to calculate the weight of each payload and invert when necessary.

3.7 Summary of properties

The properties of the code are summarized quantitatively in Table 3.

Table 3: Properties summary

	X		
	8	16	32
Overhead (%)	25	12.5	6.25
Max. control space	70	12870	very large
Simply implemented control space (payload = {Y,$\overline{Y}$})	16	256	65536
rds bounds	+/- 14	+/-26	+/-50
max. runlength (bits)	22	42	82
latency (bits)	8	16	32

4 A 16B18B code

To achieve lower overhead than an 8B10B code, X must be greater than 8. When X=16, the input to the code is byte aligned, and the overhead is reduced to 12.5%. The 16B18B code provides a large control space (256 control states may be encoded using the simple method described above).

The maximum runlength of the 16B18B code (42) is long compared to the IBM 8B10B code. However, clock recovery circuits developed for telecommunications applications should be capable of maintaining lock with this code. In SONET applications it is possible for runlengths of greater than 70 to occur.

A 16B18B code operating according to the rules described in this paper is therefore a good alternative to 8B10B coding, requiring reduced coding overhead.

4.1 Code block synchronization

There are many potential ways to achieve code block synchronization (i.e. correct alignment in the receiver of the label and payload fields). One technique would be to transmit a comma sequence during a start-up or training period. As an example consider a 16B18B code using the simple control payload scheme. A training control state of 1111 1111 would be transmitted as:

[10 1111 1111 0000 0000][10 1111 1111 0000 0000][10 1111 1111 0000 0000]

The receiver will search for the label field 10. By inspection, the 10 field is only found in two locations: the correct location, and at the end of the run of eight 1's.

However, if the label field is (wrongly) assumed to be at the end of the run of 1's, the following 16 bits are not a legal control payload (first 8 bits are not the complement of the second 8 bits). The receiver will therefore detect the mistake. The correct codeword boundary may therefore be discovered by searching for a 10 pattern followed by a valid control payload. Note that this sequence also contains a regular 010 pattern to aid in PLL clock acquisition.

4.1.1 Error detection with CRC-32 for a 16B18B code

We have shown that each single error occurring in the data payload results in a single error in the decoded data. Errors confined to the data payload are therefore not multiplied when the data is decoded, and the error detection properties of a CRC are not affected by the block coding.

Errors occurring in the label may result in error multiplication. Two errors in a coded block may result in a burst of errors up to the length of the payload in the decoded data. For the 16B18B code the implication is that a burst of up to 16 errors in the decoded data may occur if two or more errors occur in a coded block. However, such an error burst will be detected by a CRC provided that the length of the CRC is greater than or equal to 16 (since in general a cyclic redundancy check of length *y* will detect an error burst having length up to and equal to *y*). Since many data communications applications employ a 32 bit CRC, we conclude that all combinations of two errors within a 16B18B coded block will be detected by the CRC.

5 Summary

A coding technique has been described that may be used to construct block codes of the form XB(X+2)B, where X is even. The motivation of this technique is to achieve low coding overhead, and as X increases the coding overhead decreases. The codes are d.c. balanced and extremely simple to implement.

In particular, a 16B18B code may be constructed that has a coding overhead of 12.5%. The maximum runlength of this code is 42, which is less than the runlengths that may be experienced in SONET systems.

6 References

[1] A.X. Widmer and P.A. Franaszek, "A DC-balanced, partitioned-block, 8B/ 10B transmission code", IBM J. Res. Develop., vol. 27, 5, Sept. 1983, pp.440-451.

[2] ANSI X3.230-1994, "Fibre Channel - Physical and Signaling Interface (FC-PH)".

[3] IEEE 802.3z, "Media Access Control (MAC) Parameters, Physical Layer, Repeater and Management Parameters for 1000 Mb/s Operation".