

Error Detection and Correction in Point-to-Point Links

Eric Hazen, *Boston University*

October 27, 1999

Abstract

As point-to-point links become more widely used in large systems, the subject of error detection and corrections becomes important. The error rates expected in commercial links is high enough that single-bit error must be corrected and multiple-bit errors detected to ensure correct system operation.

This note discusses expected bit error rates in the context of a specific (but typical) high-energy physics application and suggests detection and correction methods. Some background on error detection and correction is presented.

1 Motivation

For our application we are considering the use of simple serial links using LVDS (low-voltage differential signalling), marketed as “Flat-Links” or “Channel-Links” by various manufacturers. One manufacturer quotes single-bit error rates of 10^{-12} as “easily achievable”. For the purposes of this note this error rate is assumed.

For example, the DØ silicon track trigger system has 6 VME crates with about 30 point-to-point links per crate. Each link carries 32 bit data words at a rate of 26MHz. This corresponds to a total of about $1.5 * 10^{11}$ bits/sec transmitted, and a single-bit error rate of 1 error per 7 seconds. Errors at this rate must not be permitted to corrupt physics data or cause synchronization loss in the trigger or data acquisition systems.

Single-bit errors are largely the result of jitter in the electrical signal, compounded by dispersion as a result of cable capacitance. Multi-bit errors may also occur, but will likely be the result of external noise coupled onto the signal cables. In the DØ system, our approach will be to correct all single-bit errors on-the-fly using an approach described here, and to detect multiple-bit errors using a CRC (cyclic redundancy check).

2 Error Correction

Single-bit errors may be automatically corrected by means of additional check bits transmitted with the data bits. The number of additional bits required to correct single-bit errors in words of various lengths is given in the table.

<i>Data bits</i>	<i>Check bits</i>
1	2
2-4	3
5-11	4
12-26	5
27-57	6

The formula

$$(m + r + 1) \leq 2^r \quad (1)$$

must be satisfied, where m is the number of data bits and r is the number of check bits.

If the check bits are placed at bit positions 1, 2, 4... in the resulting word, then the algorithm for error correction is quite simple. Take for example the eight-bit data word 10010110. We intersperse parity bits as described above, which leaves the data bits at positions 3, 5, 6, 7, 9, 10, 11, 12.

Bit	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>	<i>11</i>	<i>12</i>
Type	$\underline{P_1}$	$\underline{P_2}$	D_1	$\underline{P_3}$	D_2	D_3	D_4	$\underline{P_4}$	D_5	D_6	D_7	D_8
Value	1	0	1	1	0	0	1	0	0	1	1	0

Each parity bit P_n is located at a bit position which is a power of two, representing the even parity sum of all data bits containing a binary '1' in the corresponding bit of its bit position. (Bit positions which contain parity bits are summed as zero)

$$\begin{aligned}
 P_1 &= \text{bits}(1 + 3 + 5 + 7 + 9 + 11) \\
 &= (P_1 + D_1 + D_2 + D_4 + D_5 + D_7) \\
 &= (0 + 1 + 0 + 1 + 0 + 1) = 1 \\
 \hline
 P_2 &= \text{bits}(2 + 3 + 6 + 7 + 10 + 11) \\
 &= (P_2 + D_1 + D_3 + D_4 + D_6 + D_7) \\
 &= (0 + 1 + 0 + 1 + 1 + 1) = 0 \\
 \hline
 P_3 &= \text{bits}(4 + 5 + 6 + 7 + 12) \\
 &= (P_3 + D_2 + D_3 + D_4 + D_8) \\
 &= (0 + 0 + 0 + 1 + 0) = 1 \\
 \hline
 P_4 &= \text{bits}(8 + 9 + 10 + 11 + 12) \\
 &= (P_4 + D_5 + D_6 + D_7 + D_8) \\
 &= (0 + 0 + 1 + 1 + 0) = 0
 \end{aligned}$$

So, the word with error-correcting bits added is 101100100110. To check for an error, one must perform the parity calculations as described above (with the parity bits included in the sum). A binary word is formed by concatenating the four resulting calculated parity

bits. If the value of this word is not zero, then its value specifies the bit position of the incorrect data bit.

For example, suppose we receive 101100100100, which contains one error in the next-to-last bit location. If we calculate the parity, we get:

$$P_1 = (1 + 1 + 0 + 1 + 0 + 0) = 1$$

$$P_2 = (0 + 1 + 0 + 1 + 1 + 0) = 1$$

$$P_3 = (1 + 0 + 0 + 1 + 0) = 0$$

$$P_4 = (0 + 0 + 1 + 0 + 0) = 1$$

The resulting binary word is 1011 = 11₁₀. Since it is non-zero, we know there was an error. The error is at bit position 11, or the second to the last bit. To perform the correction, we merely flip that bit.