Extending The Error Correction Capability Of Linear Codes

Ashok D. Ingle
Daniel P. Siewiorek

Carnegie-Mellon University
Pittsburgh, Pa.

March, 1973

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0107) and is monitored by the Air Force Office of Scientific Research.

Extending The Error Correction Capability Of Linear Codes

ABSTRACT

Linear transmission codes were developed under the assumption that the bit errors are independent and random. When these codes are applied to digital circuits, this assumption no longer holds. Using the past history of the bit failures, a linear transmission code that can detect k bit failures can be made to tolerate and correct upto (k-1) bit failures. Thus if the classical error correction bounds are assumed , a linear transmission code used in digital circuitry is under-utilized. For example, the single-error-correction, double-error-detection Hamming code could be used to correct up to two bit failures with some additional error correction circuitry. A simple algorithm for correcting these extra errors in linear codes is presented.

INTRODUCTION

Several studies of linear transmission codes have been made over the last twenty years. These codes are designed for error detection (to detect d errors the minimum code weight must be d+1), for error correction (minimum weight 2t+1 to correct t errors), and for error detection/correction (minimum weight t+d+1 to correct t errors and detect d errors). [PeteW72]. Most transmission codes are based on the assumption that in transmission, each symbol is affected independently by noise and therefore the probability of a given error pattern depends only on the number of errors.

Transmission codes, such as the Hamming code, have been applied to digital circuitry such as memories. In such a case, specific bits of the word are directly associated with a given memory bit, driver, bus line, etc. Once a bit has taken an incorrect value, there is a much higher probability that the particular bit will be in error again due to a permanent or transient failure in its associated circuitry. An error correction algorithm can use the past history of bit failures to increase the error correction capability beyond the traditionally accepted limits for transmission codes.

LINEAR CODES

A bit failure will be considered to be any permanent or transient failure in the logic circuitry associated with that bit. Under this assumption, it will be shown that a minimum weight t code can detect as well as correct t-2 failures. Only in the case of noise (which can affect the various bits of information independently) will the classical bounds of t-1 error detection, $\lfloor (t-1)/2 \rfloor$ error correction be applicable. Thus transmission codes, when applied to digital circuits, are under-utilized when the classical bounds are assumed.

Classically, a linear code is a code space V generated by a generator matrix G. It is also the null space of the parity-check matrix H. i.e.

$$VH' = 0$$

$$\text{or,} \quad \forall\, v \in V, \quad vH' = 0$$

For the treatment that follows assume that the minimum weight of the code is t [*]. A single error may be specified by a vector

---

* Definitions [PeteW72] :
Hamming weight of a vector in a linear code is the number of nonzero symbols in the vector.
The ***minimum weight of a linear code*** is the minimum of the Hamming weights of all the nonzero vectors.

which has a single bit set to one. If we denote a vector whose leading bit is "1" and all other bits are "0" by $\hat{e}$, then the set of all possible single errors is the permutation ring generated by $\hat{e}$. Mathematically, for any $e_i \in E$, the weight of e_i is 1 and for all $i \neq j$, $e_i \neq e_j$. In other words E contains all distinct vectors of weight 1. A vector with a single error is thus characterised by $\tilde{v} = v + e_i$. Therefore,

$$\tilde{v}H^T = (v + e_i)H^T$$

$$= vH^T + e_iH^T$$

$$= e_iH^T$$

e_iH^T is called the syndrome, denoted by s and is nonzero if $t > 1$.

Now, assume that there are m errors, where $m<t$. Then :

$$\tilde{v} = v + \sum_{i=1}^{m} e_i \qquad \text{for some } v \in V$$

Therefore,

$$s = \tilde{v} H^T$$

$$= (v + \sum_{i=1}^{m} e_i) H^T$$

$$= v H^T + (\sum_{i=1}^{m} e_i) H^T$$

$$= (\sum_{i=1}^{m} e_i) H^T$$

Since $W(e_i) = 1$, $\forall$ i, and since all e_i's are distinct, $W(\sum_{i=1}^{m} e_i) = m$. And because $m<t$, $(\sum_{i=1}^{m} e_i) \notin V$. Hence, $s \neq 0$. This conclusively shows that we detect as many as (t-1) errors.

ERROR CORRECTION

In this section, we intend to show that by using the past history of bits in failure, one can extend the error correcting capability far beyond that predicted by the classical bounds. We must modify the existing fault-model. We will now assume that the bit failures occur independently and one at a time. But since we no longer restrict our attention to arbitrary failure patterns in hitherto unfailed hardware, failure history is of importance. We will assume that all failures are of the stuck-at type (This assumption can easily be waived, as we will show later). Therefore, once a bit has failed, it remains failed. A failed bit may yet be correct if the data happens to match with the stuck-at value of the bit. A bit will be said to be in error if it takes on the wrong value. Thus, at any particular point in time, we will presume the knowledge of the bits in failure and assume that the bit failure pattern may cover at most one bit outside these known failed bits. We now proceed to prove our claims starting with an example.

Consider the Hamming (8,4) code. The minimum weight, t=4. Under the assumption that no two (new) bit failures occur simultaneously, the first single bit failure (characterized by, say, e_i) is uniquely specified by the syndrome, $s_i = e_i H^T$. At the next check point, as the syndrome s is formed, three cases

are possible :

i) s indicates no error (i.e. s=0). The failed bit(s) agree with their expected values.

ii) $s = s_1$. There is a single error, characterized by e_1, namely, the same as before. It can be corrected.

iii) $s \neq 0$, $s \neq s_1$. Let $s_2 = s$. Let e_2 be the vector that chacterizes the error indicated by s_2.

a) Treat it as a single bit failure indicated by s_2. Correct it and re-form s. If s = 0 then it was a single bit failure, and is now corrected, otherwise it was a double error.

b) For double error, one of the bit failures must be at s_1. Correct it, re-form s, and correct the bit indicated by s now.

Successful correction of upto 2 (= t-2) errors is thus achieved [*].

By generalization of this procedure, correction of upto (t-2) bit failures for codes with the minimum weight t>4 is

---

* Note that it is possible to simplify this algorithm, by using the peculiarity of the Hamming (8,4) code, namely, that one bit of the syndrome merely is overall parity, which enables one to distinguish between even and odd errors. We chose to ignore this, however, in order to maintain the generality of the discussion.

possible. Consider the following algorithm, which is a direct generalization of the one described above. We will now change i bits from the known bit failures at a time, and form the syndrome. This syndrome will indicate a particular bit. We will "correct" that bit, and re-form the syndrome. If the syndrome is zero then the correction is over, and we will update the known bit failure pattern. Otherwise the "correction" was erroneous, we restore the bit value and proceed to check the next set of i bits. If we have exhausted all the patterns of i bits, we will increment i, and start again. The process continues until we either correct the errors or exhaust all the known bit failures. We may restate the algorithm as follows :

Assume that m bits were known to have failed, where m≤(t-2).

a) Set i = 0.

b) Change i of the known failed bits at a time and form the syndrome s. If m = (t-2) then go to step d.

c) Change the bit indicated by s and re-form s.

d) If s=0 then done, update the knowledge of failed bits, if necessary, and EXIT.

e) If all sets of i bits have been tried then go to step f else go to step b.

f) Set i = i+1 .

g) If i ≤ (t-2) then go to step b, otherwise the fault exceeds our correction capability.

The algorithm terminates in a finite number of steps under our assumptions. However, as a safety measure, one may check for i exceeding (m+1) at step f, which may detect particular instances of multiple errors beyond the known failures. To prove that the algorithm can correct (t-2) failures :

Let m = number of bits known to have failed; m ≤ (t-2).

Let k = number of bits in error in the current word; by virtue of the assumed failure model k ≤ (m+1).

Since the algorithm is exhaustive, it will certainly attain the combination of bits in error at some point. Therefore, one only needs to prove that it does not yield a zero syndrome for any combination of bits other than those in error.

We will consider the two cases m ≤ (t-3) and m = (t-2), separately. First let m ≤ (t-3). Consider any trial combination of i bits. Let the number of bits that are changed erringly (as the i bits are changed) be w. Thus the number of correct changes is (i-w), and there are (i-w) bits common between k and i. Hence the total number of errors in the word, after the syndrome is formed and "correction" made, equals (k+w+1) - (i-w) or k+2w+1-i. To ensure that no erroneous "correction" terminates the algorithm, the following inequality

must hold.

$$(k + 2 w + 1 - i) \leq (t - 1) \quad \ldots (1)$$

By the definition of w, $w \leq i$.

Therefore, $2w - i \leq w$.

or, $k + 2w - i + 1 \leq k + w + 1$.

Thus, (1) holds if

$$k + w + 1 \leq (t - 1)$$

Again, by definition, k and w occupy at most (m+1) bits and they span disjoint sets of bits. (See Figure 1).

Hence, $k + w \leq (m + 1)$

Therefore, (1) holds if

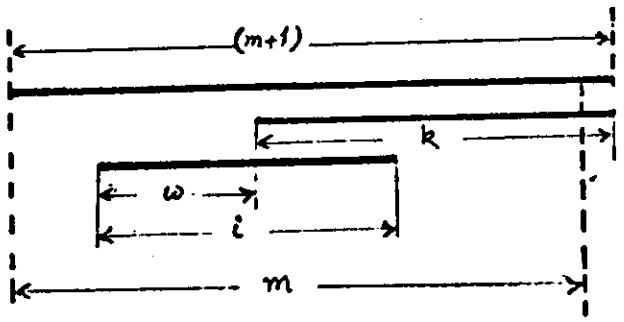
$$(m + 1) + 1 \leq (t - 1)$$

or, if $m \leq (t - 3)$

But this is true by assumption. Therefore, (1) holds.

Now, let us take the case of m = (t-2). Since we do not attempt "correction", we need not consider the addition of an error. In other words, we know that we have reached the bound on correction of additional errors and restrict ourselves to correcting errors

Figure 1 : Conceptual mapping of bits in failure.



among the known (t-2) failed bits. Now, $k \leq m = (t-2)$. Hence, the inequality (I) may be rewritten as :

$$(k + 2w - i) \leq (t - 1) \quad \ldots\ldots\ldots\ldots\ldots (II)$$

Again, $2w - i \leq w$.

or, $k + 2w - i \leq k + w$.

Therefore, (II) holds if

$$k + w \leq (t - 1)$$

But now k and w span disjoit bits among m (= t-2) bits.

Hence, $k + w \leq (t - 2)$

Therefore, (II) holds.

Thus it is conclusively proved that the algorithm terminates in and only in the required correction for $m \leq (t-3)$; and hence we can correct upto (t-2) errors.

DISCUSSION

The algorithm, if implemented in its present form, could be extremely time consuming. In the worst case ($k = m+1 = t-2$), one has to go through

$$\sum_{i=0}^{t-2} \binom{t-2}{i} = 2^{t-2}$$

different trials before arriving at the proper correction.

Some improvement may be achieved by use of table look-up. Figure 2 may exemplify such methods. The syndrome and the known bit failures may be used to index into a table and retrieve the

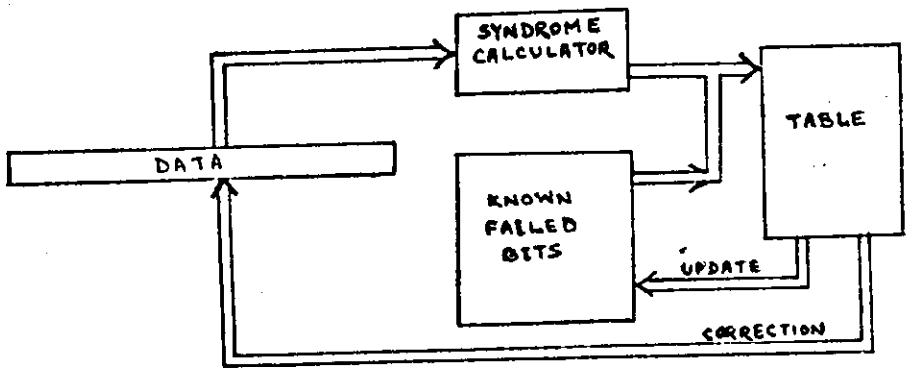
required corrections. Depending on the versatility of the table, which may be function of the storage space available, the indexing may involve from a single reference to an organized search of the table.

The algorithm presented here is very simple-minded, and serves only as a tool to prove our claims. In practice, it should be improved upon. Clever codes may be designed in order to simplify and minimize the overheads involved (In Hamming (8,4) code, for example, it is possible to reduce a step, as the overall parity bit can indicate even or odd error).

The assumption of stuck-at faults may be relaxed, if along with the bit failures, one also stores the value the bit is stuck at. Every fault, then, can be treated as a stuck-at fault, and stored as such. If any bit differs from its stuck-at value, the fault was a transient one, and hence can be deleted from the store.

The improved correction capability may be employed in

Figure 2 : One implementation of table look-up scheme.



several ways. Consider a Hamming single-error-correction, double-error-correction code applied to a bit-sliced memory. When an error is detected, a spare bit-slice may be switched in. This new bit-slice will contain erroneous information until every bit on the slice (i.e. every word in the memory block) is written into at least once. Since this "update" time may be substantial - and, in the worst case, infinite - the correction capability of the system may be seriously affected, even though additional spare bit-slices may be available. The second failure causes the memory to halt. By extending the correction capability, as outlined above, a bit failure can be corrected, even in the presence of a newly switched in bit-slice that may continue to yield erroneous information. Extended correction capability may also provide the basis for toleration of greater number of bit errors and may be a feasible alternative to dynamic switching of bit-slices.

Since the theory employed here is that of the classical linear codes, all the extentions of the theory, such as the ones to byte (or symbol) correction, may be similarly derived from the work presented here.

The effects of our new assumption should be studied in connection with other theoretical work, especially other codes (e.g. arithmatic codes), and this is the direction of our

future work.

REFERENCES

PeteW72 Peterson, W. W., Weldon, E. J. Jr, Error-Correcting Codes, II Ed., The MIT Press, Cambridge, Massachusetts, 1972.