TWO-BYTE DATA BLOCK AND BUNDLE CODES FOR THE BROADCAST TELIDON SYSTEM

Brian Mortimer

Mike Moore

with the assistance of

James Currie

Paula Gray

Andrew Dobrawalski

Lee Oattes

Brian Leroux

A Progress Report prepared for the Department of Communications, Ottawa under DSS Contract No. 0SU82-00164

November, 1982

CONTENTS

0. Introduction 1.

1. A Particular Two-Byte Code

1.1 The Code $\mathcal{C}$ 1.

1.2 Relationship to the Product One-Byte Code 3.

1.3 Decoding the Code $\mathcal{C}$ 3.

1.4 Code Dependent and Independent Factors 4.

1.5 The Code $\mathcal{C}$ Defined 6.

1.6 Performance under Independent Errors 10.

Figure 1.1 Expected Number of Blocks Until Decoding Failure and Error 12.

2. Coding A Bundle of Data Blocks

2.1 The Bundle Codes 13.

Figure 2.1 Arrangement of the 14 Interleaved Codewords of the Bundle Code 15.

2.2 Performance of the System with Independent Errors 16.

Figure 2.2 Expected Number of Bundles Until Decoding Fault 17.

2.3 Implementations 18.

Bibliography 21.

0. INTRODUCTION

This report describes some of the results obtained by our research group in our work on error correcting codes for the Broadcast Telidon System. We begin by describing a particular code which could be used as a two-byte data block code. This code extends the product code and corrects all double errors and any error confined to a single byte. It is in fact a single symbol correcting Reed-Solomon code and is easily decoded.

The second topic considered is the idea of collecting a number of data blocks together and encoding them as a unit. At the receiving end, the decoding would be done off line after all the packets from that unit have arrived. We show that this approach can give very good performance with negligible delays to the user.

We have actually pursued many other avenues of inquiry in our research and some of these will be outlined in our Final Report. The topics covered here though are perhaps the most important.

1. A Particular Two-Byte Code

1.1 The Code $\mathcal{C}$

We will present in this section a particular error correcting code. We will call it $\mathcal{C}$. This code is an instance of a Reed-Solomon code. We consider it in the context of a teletext system taking as its symbols bytes of

8 bits. The code $\mathcal{C}$ uses two bytes of redundancy and can be extended in a natural way to cover 127 bytes in total.

This code is capable of two types of correction: error correcting and erasure correction. An 'error' in this context means an erroneous symbol whose location is unknown while an 'erasure' denotes an erroneous symbol whose location is known. The distinction is unimportant if the symbols are binary since to know the location of a binary symbol is to know how to correct it. In our case there are 256 possible errors in each symbol so locating a faulty byte is only half the problem.

The code $\mathcal{C}$ can correct any single byte-error or any double byte-erasure. Thus if any error corrupts a single byte it can be corrected. In addition, if an error corrupts two bytes in a way that makes the bytes locatable, say by leaving them with the wrong parity, this error can also be corrected. In any one codeword we do one type of correction or the other but not both. Clearly, this code will correct any single bit error. If two bit errors occur then they either fall in the same byte or fall in two bytes and change the parity of each. Thus double bit errors are always correctable. If three bit errors fall two in one byte and one in a second or in three different bytes then these errors are not correctable. (In fact a two byte code cannot possibly correct all triple errors).

In the context of a teletext system using known parity data bytes the decoding can procede as follows:

. Count the number N of bytes with a parity failure,

. If N=0 or 1 then attempt a single byte (error) correction,

. If N=2 then attempt a double byte (erasure) correction,

. If N > 2 then declare a failure.

1.2 Relationship to the Product One-Byte Code

The code $\mathcal{C}$ uses two bytes of check symbols. It is defined in such a way that the exclusive-or sum of all the bytes is zero. Thus each codeword of $\mathcal{C}$ is also a codeword the Product code specified in the provisional version of BS-14 [1]. Either of the two bytes can be used as the Product code check byte. When interpreted as a Product codeword we just have to remember that the penultimate byte is not data and should not be passed to the picture generating unit.

1.3 Decoding the Code

The code $\mathcal{C}$ can be quickly decoding using a software decoder based on look-up tables. We have an implementation which uses a Motorolla 6809 processor running at 1.29 MHz. The decoder takes about a millisecond to decode one codeword using 512 bytes of look-up tables and a program 181 bytes long.

Hardware decoding would certainly be feasible. It would be a straightforward exercise in implementing algebraic equations in logic circuits. Some commercial enterprises, such as Berlekamp and Golomb's Cycotomics Ltd., are currently marketing hardware Reed-Solomon decoders of much greater complexity than would be required for our code.

1.4 <u>Code Dependent and Independent Factors</u>

The code $\mathcal{C}$ has many virtues:

. it extends the Product code,

. it corrects all double bit errors,

. it has a straightforward algebraic definition,

. it can be used both as a data block code and to form an interleaved code on a collection of data blocks (see § 2 below).

We must deal though with the question of optimality. Is there a better code? The criterion to be optimized is performance; that is, the number of errors allowed to reach the user of the teletext system. We take decoding complexity as a secondary factor to be considered and this acts as a constraint on the optimization process.

The decoding of a received message may have one of two outcomes: a correct decoding or an incorrect decoding. We thus can calculate (at least in principle) for a given channel error model the probability P_C of a correct decoding and hence the probability $P_I = 1-P_C$ of an incorrect decoding. This depends only on the patterns of errors which are corrected (single bit-error, double bit-error, single byte-error etc.). So if we are correcting double bit-errors then every double-bit-error-correcting code which is decoded as such has the same P_C and P_I .

The code itself becomes important when we look in more detail at P_I. There are two forms of 'incorrect decoding' which can occur. The decoder may notice an uncorrectable error and set a flag even though it can't correct the fault. This is a decoding failure and occurs with probability P_F. The other possibility is that the errors are uncorrectable but fool the decoder into believing that they can be corrected. This outcome is called a decoding error and occurs with probability P_E. The result of a decoding failure is that the system must wait for a rebroadcast; the effect of decoding error is to pass rubbish on to the user as genuine data. It also follows that $P_I = P_F + P_E$. In general decoding errors are rare compared to decoding failures.

The optimization problem then splits into two parts:

. maximize P_C by choosing the right decoding strategy for your error patterns and system,

. minimize P_E, given the decoding strategy, by choosing a code which has few codewords which resemble the commonest error types.

These remarks apply equally to the data block codes and to the bundle codes outlined in § 2 below. We have attempted to use the bundle code to make P_C large. Thus the expected number of bundles until a decoding error or failure is made large enough that the importance of minimizing P_E is greatly reduced. Moreover we have tried to use the same two-byte code for both the data block and bundle codes to eliminate the need for a second decoder.

Given these assumptions and constraints we are left only with the possibility of finding another code with smaller P_E which fits our conditions. This code would then give better service in the case of either decoders which do not deal with bundle codes or databases which are not bundle coded. This is a hard problem. We will give some results in 1.6 on the situation when errors arise independently.

1.5 <u>The Code $\mathcal{C}$ Defined</u>

The code $\mathcal{C}$ is a one-symbol-error-correcting Reed Solomon code with symbols taken from the field $\mathbf{F}_{128}$ of 123 elements [3], [4]. We define using a primitive element of the field. Thus the 127 powers β^0, β^1, ..., β^{126} are distinct and are a list of the non-zero field elements. The codewords are polynomials in one variable with coefficients in $\mathbf{F}_{128}$. These polynomials

$C(X)$ must satisfy two conditions to belong to $\mathcal{C}$ (for a constant $n < 128$):

(i) $\deg(C(X)) < n$

(ii) $C(\beta^0) = C(\beta^1) = 0$.

The first condition implies that each code polynomial represents a unique vector of length n over F_{128}:

$$C(X) = C_0 + C_1X + \ldots C_{n-1}X^{n-1} \equiv (C_0, C_1, \ldots, C_{n-1}) .$$

This constant n is the code length (in symbols) and must be less than 128.

The second condition allows us to perform corrections. Suppose that a codeword $(C_0, \ldots, C_{n-1})$ is sent and an error occurs in a single symbol C_j. Then for some e_j the vector received is actually

$$(C_0, C_1, \ldots, C_j+e_j, \ldots, c_{n-1}).$$

Writing this as polynomials $R(X) = C(X) + E(X)$ is received where $E(X) = e_jX^j$. Now we evaluate $R(X)$ at $\beta^0 = 1$ and β^1 noting that $C(\beta^0) = C(\beta^1) = 0$. We obtain,

$$\begin{aligned} R(\beta^0) &= C(\beta^0) + E(\beta^0) \\ &= E(\beta^0) \\ &= e_j\beta^{0\cdot j} = e_j \end{aligned}$$

which is the error value, and

$$\begin{aligned} R(\beta) &= C(\beta) + E(\beta) \\ &= E(\beta) \\ &= e_j\,\beta^j . \end{aligned}$$

Thus we write $R(\beta)/R(1)$ as a power β^j of β to obtain j, the index of the faulty byte. Then $R(1)$ is the quantity to be subtracted to correct the error. This proves that $\mathcal{B}$ is single symbol correcting.

Now suppose that the error polynomial has the form $E(X) = e_j X^j + e_k X^k$ so that errors occurred in the j-th and k-th symbols. Suppose that the integers j and k are known. This is the erasure situation. Again we evaluate at $\beta^0 = 1$ and $\beta^1 = \beta$. Then

$$e_j + e_k = R(1)$$
$$e_j \beta^j + e_k \beta^k = R(\beta)$$

are known as are the coefficients β^j and β^k. Thus we can solve for e_j and e_k (since $j \neq k$) and correct the errors. Therefore two symbols whose locations are known can be corrected.

We must now relate the symbols to bytes. Let α be another primitive element of F_{128}. In fact we will take $\alpha^8 = \beta$ but this is only important later. The first 7 powers of α namely $1, \alpha, \alpha^2, \ldots, \alpha^6$ are linearly independent over $F_2 = \{0,1\}$ so each field element μ corresponds to a unique 7-tuple of bits by

$\mu \leftrightarrow (u_0, u_1, \ldots, u_6)$ iff $\mu = u_0\alpha^0 + \ldots u_6 \alpha^6$

with each $u_i = 0$ or 1. The set $\{\alpha^0, \alpha^1, \ldots, \alpha^7\}$ is linearly dependent since α satisfies a polynomial of degree 7. We will take α as a root of $X^7 + X^3 + 1$ for example. Now each field element has two representations as

$$\mu = u_0\alpha^0 + \ldots u_7\alpha^7$$

with $u_i = 0$ or 1. In one representation the parity of the byte $(u_0, u_1, \ldots, u_7)$ is even and in the other it is odd. We change from one representation to the other by adding $0 = 1 + \alpha^3 + \alpha^7$. Therefore if we use bytes of one fixed parity, we have a unique byte for each field element.

Now the bytes to be coded $B_0, B_1, \ldots, B_{n-3}$ represent field elements and we are to determine B_{n-2}, B_{n-1} so that

$$B(1) = B_0 + B_1 + \ldots + B_{n-1} = 0$$

$$B(\beta) = B_0 + B_1\beta + \ldots + B_{n-1}\beta^{n-1} = 0 .$$

But $B_0 + B_1 + \ldots + B_{n-1}$ is just the exclusive-or of the corresponding bytes since we are working in a field of characteristic 2. Therefore a codeword of $\mathcal{C}$ is a Product codeword. The second condition translates into the statement that if the bytes $B_0, B_1 \ldots, B_{n-1}$ are written as a bit string $m_0 m_1, \ldots, m_{8n-1}$ then we have

$$\sum_{i=0}^{8n-1} m_i\alpha^i = 0 .$$

This follows from the assumption $\beta = \alpha^8$ and the use of the quasi-basis $\{\alpha^0, \alpha^1, \ldots, \alpha^7\}$ for F_{128}. This means that the codewords of $\mathcal{C}$ are also codewords of Carleton code as it was defined in [2].

1.6 Performance under Independent Errors

Once we have decided to use a double-bit-error correcting code on the data blocks this determines the probability of a correct decoding. Further refinement of the code can only change the probability of decoding failure and decoding error. All of these probabilities depend crucially on the frequency and correlations of the errors in the channel. We deal only with the case of independent errors at a time invariant bit error rate p.

When errors are independent the probability of decoding error P_E can be calculated from the weight distribution of the code; that is, from the numbers A_i of codewords with exactly i non-zero entries for $i = 0,1,\ldots,n$. For a double bit-error correcting code of length n we have $A_0 = 1$, $A_1 = A_2 = A_3 = A_4 = 0$ and

$$P_E = q^n \left\{ \sum_{k=4}^{n} \left[\binom{n-k+2}{2} A_{k-2} + (n-k+1) A_{k-1} + A_k + (k+1) A_{k+1} + \binom{k+2}{2} A_{k+2} \right] x^k \right\}$$

where $q = 1-p$ and $x = p/q$. We see that P_E is reduced greatly if we can take $A_5 = 0$ and then minimize A_6. The low weight distribution of code $\mathcal{C}$, which has no odd weight codewords, is

i	0	1 2 3 4 5	6	7	8	9	10
A_i	1	0	8,600	0	1.21E6	0	1.75E10 .

Can we find another code with no odd weight codewords and with A_6 smaller than 8,600 , perhaps even with $A_6=0$? The answer to the first qustion is unknown but we can prove that A_6 must be positive.

What we can show is that any 28 byte long code with two check bytes, one of which is the check byte of Product code, with data bytes of even parity must have $A_6 > 1000$. This implies that such a code cannot correct all triple errors. The number of weight 6 codewords in $\mathcal{C}$ might be reduced by using an α which is a root of a different polynomial or by taking β to be different from α^8. We have not tried this as yet. Moreover the lower bound on A_6 is not by any means tight.

We should emphasize that the data block code is part of a coding system. Ultimately the only performance that matters is the performance of the system as a whole. The most important calculations are therefore of the performance of the prefix code, data block code and any 'bundle' code acting in concert. We present the results of such calculations in the next section.

<u>FIGURE 2.1</u> Performance of Code $\mathcal{C}$ Against Independent Errors

BER	CODE	Expected Number of Data Blocks until Decoding Failure	/ Decoding Error
10^{-3}	Product	47	2.9E4
	$\mathcal{C}$	638	9.7E6
10^{-4}	Product	4068	2.5E7
	$\mathcal{C}$	5.5E5	7.9E10
10^{-5}	Product	4.0E5	2.3E10
	$\mathcal{C}$	5.4E8	7.8E14

2. Coding a Bundle of Data Blocks

2.1 The Bundle Codes

At a higher level than the individual data packets we might choose to encode 'bundles' or groups of data blocks together with a more powerful code to clean out any errors left by the packet level codes. In this section we will describe various ways of doing this.

For purposes of discussion we will fix the size of the bundle at 8 data blocks with perhaps a 9th data block whose 28 bytes are the check bytes of the bundle code. We defer considerations of other sizes to our Final Report. We have available a number of strategies for coding the bundle. For any given strategy we ask, 'what is the probability that after decoding a particular bundle of 8 data packets will be completely correct?' We reformulate this question and calculate an equivalent parameter: the mean number of bundles before an incorrect bundle decoding. This mean number depends on the error patterns and their frequencies in the channel and on the decoding strategy but does not depend on the particular code used to implement that strategy.

The coding of a bundle comes in 3 layers. The first is the prefix code which consists of 5 bytes each encoded with the Hamming (8,4) code. Due to the continuity count byte (the 4th byte) and the interpretation of the packet structure byte (the 5th byte) it will be a rare event that a decoding error in the prefix actually goes unnoticed by the system. The most common error effect is that a packet is lost or appears to be lost from the broadcast stream. This might also occur if there are synchronization problems when the line arrives.

The second layer is the data block code. We consider three options:

Byte-Parity (Parity)

The only coding is the parity bit in each data byte.

Single Bit Error Correction (SEC)

The 28th byte of the block is a check byte which allows the correction of a single bit error. With only one check byte it is not possible to correct all double errors.

Double Bit Error Correction (DEC)

The 27th and 28th bytes of the block are check bytes. Any double bit error can be corrected.

The third layer of coding covers all of the data blocks together (but excludes the prefix bytes). We consider four options. The first is to add no further coding. The other three are simply alternative decoding strategies for the same encoding. For these we interleave a number of vertical byte-correcting codes across the set of 8 data blocks adding 28 check bytes as a ninth data block (the layout of the interleaving is given in Figure 2.1). We assume that the basic code which is then interleaved is capable of correcting any error pattern confined to a single byte or any two byte-errors when the bytes are known (erasure decoding). We could use code $\mathcal{E}$, for example. We summarize the four strategies for coding the bundle as follows.

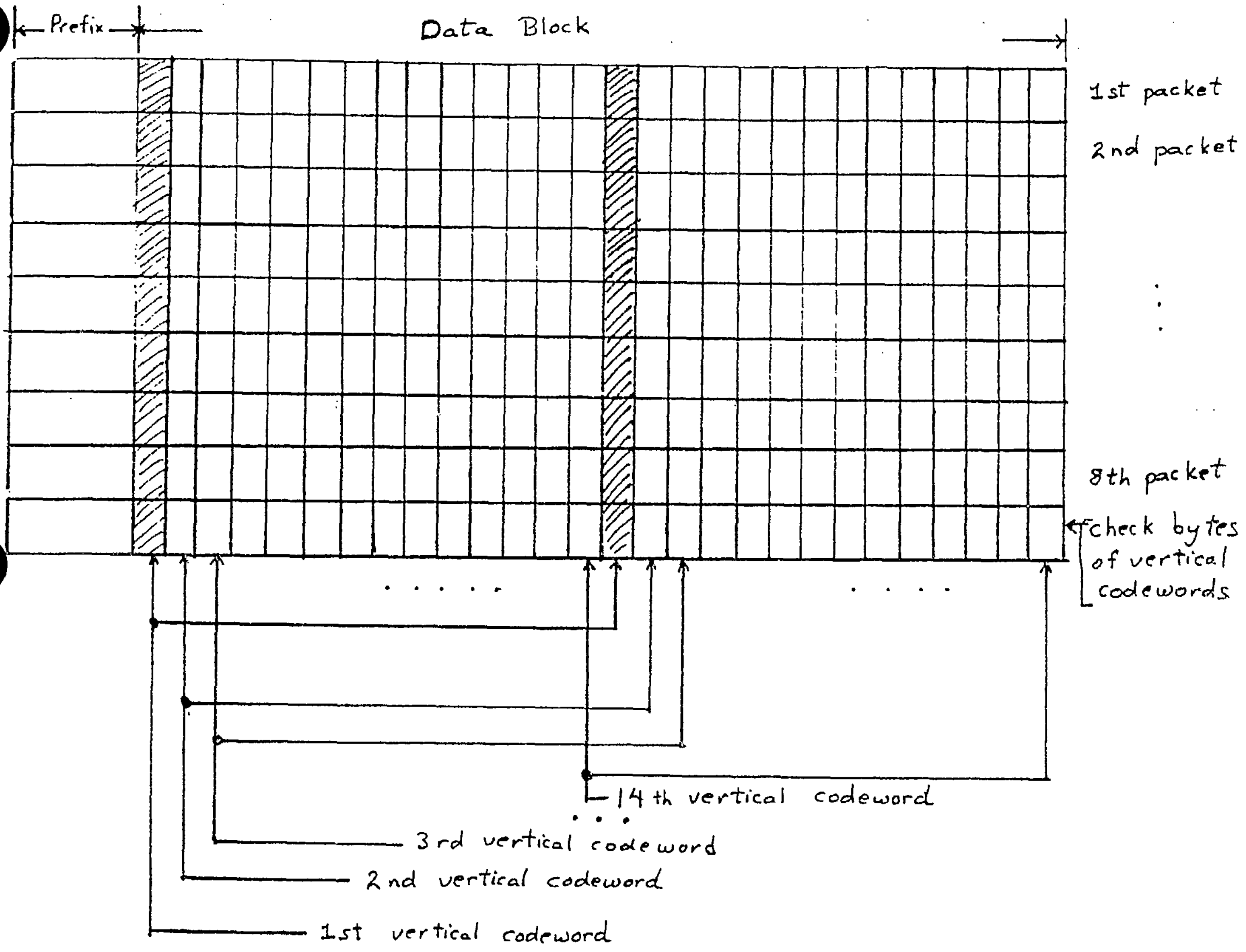

Figure 2.1 Arrangement of the 14 Vertical Codewords of the Bundle Code

Nocode:

The eight data blocks are sent as they are with no further protection.

Erasure correction:

The bundle code is used to replace a lost data block by erasure correction. One data block can be retrieved and no other corrections are attempted on the bundle. This technique can overcome a failure in a single prefix or a single synchronization error.

14 X 1-Byte-error correction:

A single byte error in each of the 14 interleaved codewords is corrected. If there are two faulty bytes in any one codeword then the errors are not correctable.

Erasures and Errors:

In each of the interleaved codewords we correct either a single erroneous byte or two bytes with parity failures or two bytes which are absent entirely. In this scheme a single missing data block is replaced if there are no other errors.If all the data blocks are present we correct either one byte error or two byte erasures in each of the 14 vertical codewords.

2.2 Performance of the System with Independent Errors

In order to calculate the mean number of pages before a decoding fault (not a correct decoding) we must first choose a coding strategy for the second and third levels and then make an assumption about the patterns and frequency of the errors. As to the latter we will assume at this stage that the errors arise independently and defer the question of burst errors to later consideration. We are then able to present (Figure 2.2) for each combination of coding

FIGURE 2.2 Expected Number of Bundles Until Decoding Fault

Coding Stategy		Bit Error Rate						
Bundle	Data Block	10^{-3}	$8x10^{-4}$	$6x10^{-4}$	$4x10^{-4}$	$2x10^{-4}$	10^{-4}	10^{-5}
No code	Parity	1.2	1.3	1.5	2.0	3.3	6.1	56
	SEC	6.2	9.2	16	33	129	505	4.9E4
	DEC	74	137	304	940	6280	3.9E4	1.6E6
Erasure correction of 1 data block	Parity	1.2	1.3	1.4	1.8	3.0	5.9	50
	SEC	65	145	421	1969	2.9E4	4.5E5	4.4E9
	DEC	9615	3.3E4	1.6E5	1.6E6	7.0E7	2.7E9	1.3E14
Single byte correction in 14 v.c.	Parity	8.4	13	22	47	185	730	7.2E4
	SEC	71	127	252	621	2630	1.0E4	1.0E6
	DEC	352	615	1241	3247	1.6E4	7.0E4	7.9E6
1-byte error 2-byte erase. in 14 v.c.	Parity	140	263	596	1920	1.5E4	1.1E5	2.8E7
	SEC	*	*	*	*	*	*	*
	DEC	1.5E4	3.2E4	8.5E4	3.2E5	2.7E6	2.2E7	2.0E10
Expected number of bit errors in the data blocks		2.02	1.61	1.21	0.81	0.40	0.20	0.02

Notes: 1)A Bundle contains eight data blocks and except for the case of 'No code' above there is a ninth data block consisting of check bytes.
2)The abbreviation v.c. above stands for vertical codeword.
3)The entries marked * have not been calculated at time of printing.

strategies and a range of bit error rates the expected number of bundles until an incorrect decoding.

The data of Figure 2.2 show clearly that performance can be greatly improved if a bundle code is used ,at least, to correct erased data blocks and then to correct more if it can.This is especially true if a double bit error correcting code is used on the packets first. The importance of the double bit error corrections is made clear by calculating the expected number of bit errors in the bundle. We have included these numbers in Figure 2.2 as well. At a bit error rate of 10^{-3} we expect slightly more than 2 errors. The probability that two bit errors are in the same block is just 1/9. Thus a single bit error correcting code has a lot of trouble. Also without erasure decoding we cannot escape the failures from the prefix so these act as a limiting factor. We can easily calculate for example that the expected number of bundles until at least one prefix fails is 800 at a BER of 10^{-3} but we expect 1.4E6 bundles before at least two prefixes fail at the same BER.

2.3 <u>Implementations</u>

We can interpret 'implementations' in at least three relevant ways:

. How will we tell a teletext decoder that a bundle code is in use?

. How should we encode and decode a particular code in hardware or software and how long does it take?

. In a bundle code, is the check line encoded with a data block code? Which one takes priority?

It is the second two questions that we address in this section. We assume that the data blocks code is either the Product code or code $\mathcal{C}$ and the vertical codewords of the 14 bundle code are also from code $\mathcal{C}$. We remarked that decoding a code $\mathcal{C}$ codeword takes about 1 millisecond in M6809 software at a clock speed of 1.29 MHz. Decoding the whole bundle would then require about 23 milliseconds including both horizontal and vertial codewords. This would represent a negligible delay on the part of a user. Of course we have ignored all problems of overhead in the programming but we won't delay the user by more than a twinkle of an eye at the outside. The details are deferred to the Final Report.

The third question is more subtle. Suppose the data blocks have a two byte code and we use only 13 vertical codewords in the interleaving. Thus we do not encode the data block check bytes in the bundle code. Then the ninth line in the bundle code has 26 bytes of bundle checks and the last two can be encoded for the horizontal data block code. This is a reasonable solution to the problem since having used the data block code we no longer care whether there are errors in its check bytes or not.

If for some reason the data block code is a single byte Product code and we then superimpose a bundle code then the preceding paragraph does not apply. Now however, a happy event comes to pass. Encode the 8 data blocks with Product code, then add the ninth line of 28 check bytes of 14 vertical codewords. It happens that this ninth line is in fact automatically a codeword of the Product code.

Therefore no matter which data block code is used we can arrange to have the bundle code check block also a codeword of the data block code, albeit in a non-uniform way.

Bibliography

[1] "Television Broadcast Videotex",Broadcast Specification No. 14, Issue 1, Provisional, Telecommunication Regulatory Service,Department of Communications, Canada, June 19,1982

[2] Mortimer, B.C., "A Study of the Use of Error-Correcting Codes in Broadcast Telidon",Final Report,DSS Contract No.OSU81-00095,Deparment of Communications, Canada, February,1982

[3] MacWilliams,F.J. and N.J.A.Sloane,"The Theory of Error-Correcting Codes",North Holland Mathematical Library Vol.16, North Holland, Amsterdam, 1977

[4] Peterson, W.W. and E.J.Weldon,"Error-Correcting Codes", 2nd Edition,MIT press, 1972