

ERROR CORRECTION SCHEMES
FOR BROADCAST TELETEXT
SYSTEMS

Brian Mortimer

Michael Moore

**Department of
Mathematics and Statistics**



P
91
C655
M678
1984

LKC
P
91
.C655
M678
1984
c.2

IC

**Carleton University
Ottawa, Canada**

P
91
.CESS
M678
1984
S-Gen

ERROR CORRECTION SCHEMES
FOR BROADCAST TELETEXT
SYSTEMS

Brian Mortimer

Michael Moore

Prepared under DSS Contract No. OST83-00078

for the

Department of Communications

Ottawa, Canada

March, 1984

Principal Investigator:
Dr. Brian Mortimer
NSERC University Research Fellow
Department of Mathematics and Statistics
Carleton University
Ottawa, Canada

Scientific Authority:
Dr. Mike Sablatash
Communications Research Centre
Department of Communications
Ottawa, Canada

Industry Canada
Library - Queen

NOV 21 2013

Industrie Canada
Bibliothèque - Queen

Table of Contents

Statement of Results

Acknowledgements

1. Introduction and Results

1.1 Error Correction Schemes: an Outline	1-1
1.2 Our Proposed System	1-2
1.3 Correction Capabilities of the Double Bundle Code	1-8
1.4 Other Proposed Teletext Codes	1-10
1.5 Performance Results with Independent Errors	1-12
1.6 Decoding the Double Bundle	1-12

2. Performance of the Double Bundle Code with Independent Errors

2.1 Results	2-1
2.2 Methods	2-5
Appendix: the Software	2-14

3. An Assessment of the Proposed Japanese Teletext Code

3.1 The Japanese Proposal and the NABTS	3-1
3.2 The Comparison of Various Packet Coding Systems	3-3
3.3 Method of Calculation	3-6

4. Performance in a More Real World

4.1 After all Errors May Not be Independent	4-1
4.2 Analyzing Field Data for Code Performance: a beginning	4-2
4.3 An Example: E184R-2	4-4
Appendix: the Software	4-6
(a) FLDANALB	4-11
(b) DATA CRUNCH	4-13
(c) QUICK PASS	

5. Support for the Patent Applications

5.1 Activities	5-1
5.2 A Hardware Encoder for Carleton Code.	5-1

References

CONCATENATED ERROR CORRECTION SCHEMES FOR BROADCAST TELETEX SYSTEMS

DSS Contract No. OST83-00078

Principal Researcher: Dr. Brian Mortimer

Principal Results

1. The Double Bundle error correcting code was defined and studied on channels with independent errors. It was shown that this code gives a significant improvement in performance compared to the Single Bundle code at bit error rates less than 10^{-3} .
2. Repeated decoding of the Double Bundle code was studied and shown to give a minor improvement over single decoding for bit error rates at most 10^{-3} and independent errors. Decoding more than twice is only rarely useful.
3. Decoding of the Double Bundle code was studied. It was shown to be possible to decode an h packet bundle on a 6809 microprocessor at 2 MHz. in atmost 3.5h milliseconds.
4. Various proposed coding options were compared with the Japanese proposal, called code 'Best', on channels with independent errors. The Japanese code out-performs our code on this channel at a cost of low information rate and of violating the NABTS.
5. A framework was devised and software written to analyze the performance

of various codes on the field data collected by the Communications Research Centre. One site was analyzed as an example only.

6. A hardware encoder was designed for Carleton Code.

J. B. Montimer
March 22, 1984

ACKNOWLEDGMENTS

It is a pleasure to acknowledge the assistance we have received from Linton Hutchison in dealing with the field data and from Susan Jameson, Beverley Hall and Suzanne Drahotsky in preparing the reports .

1. INTRODUCTION AND RESULTS

1.1 Error Correction Schemes: an Outline

This report is concerned with proposals for error correction schemes appropriate for broadcast teletext systems. Error correction can greatly improve the performance and extend the range of such systems. Since a broadcast system will be used in both rural and urban settings a variety of error types are to be expected and an error correction system must be able to deal with independent and a spectrum of burst error events. Moreover, code should place a minimum demand on the teletext decoder in terms of cost, memory, hardware. The code should be quickly decodable to permit on-line decoding or minimum delay off-line processing. Furthermore, the code rate must be kept high to satisfy the efficiency demands of the system operators.

An additional constraint which we have insisted upon is that the coding system consists of a nested sequence of compatible codes of increasing power. Thus a teletext decoder which is only capable of decoding the simplest code will be able to decode data encoded with a more powerful code to the extent that the decoder is able. Conversely, a teletext decoder which is equipped to make use of the most powerful code can still decode the simpler codes by ignoring part of its own abilities.

The background organization of the data to be encoded is described in the North American Broadcast Teletext Standard (draft) (NABTS)[1]. The coding scheme proposed here is compatible with this standard. (The codes discussed in Chapter 3 which have been proposed by Japanese do not appear to be compatible with the NABTS so our comparison with them is only approximate.) The basic unit for data transmissions is the data packet or 'packet'. Each packet consists of 33 bytes (of 8 bits each). All bytes have odd parity. (The packet is preceded by synchronization signals to form a data line. This will not concern us, though we will feel the effects of these signals.)

1.2 Our Proposed System

We will now describe our proposal for an error correction system for a Broadcast Teletext System. The early parts of the system are already included in the NABTS [1] and the remainder can easily be included.

The first 5 bytes form the prefix and are encoded with an odd-parity Hamming [8,4] code. Thus the prefix is encoded with a rate $\frac{1}{2}$ code. One of the 5 prefix bytes, called the packet structure byte, is used to indicate the number of bytes taken from the remaining 28,

which form the data block, for use in error correction. The number taken is 0, 1, 2 or 28. Let us denote this number by S .

We take the values of S in turn:

- $S = 0$: In this case the only error protection comes from the fact that the 28 bytes of the data block have odd parity. Thus an odd number of errors in any byte is detected but no correction is performed.
- $S = 1$: One of the bytes (the last) of the data block is redundant. We will take this byte to be the exclusive-or (mod 2 sum) of the other 27 bytes. This forms a 'two-way' parity check code capable of single bit-error correction and double bit-error detection.
- $S = 2$: The two check bytes are defined in such a way as to make the data block a codeword of a particular (algebraically defined) Reed-Solomon code, which we call code C. This code can be decoded byte-wise and will correct any single byte-error or any double byte-errors when both erroneous bytes have a parity failure. Implicit in this statement is the code's ability to correct any double bit-error pattern.

S = 28: In this case all 28 bytes of the data block are used as check bytes for "vertical" codewords. The vertical codewords are defined as elements of the code C discussed above and hence require two check bytes per codeword. Using vertical as well as horizontal coding results in what we have called a Bundle Code. This scheme is described below.

The Bundle Coding system encodes a set of h data packets into a bundle. Each packet has a Hamming encoded prefix of 5 bytes and data block encoded with a horizontal code (one of the cases $S = 0, 1, 2$ above). We have considered two ways to complete the system which we call the Single Bundle and Double Bundle systems. The Single Bundle system was introduced in [2] and [3]. It uses 13 vertical codewords (14 if the horizontal code is type $S = 0$ or 1) from code C. Each such codeword takes two bytes from each data block of the bundle. The two bytes from the last **block** are the check bytes of the codeword. For best performance the two bytes are taken as byte i and $i + 13$ in each data block.

The Double Bundle system was briefly mentioned in [3] and is discussed at length in this report. The final 2 data blocks are used to hold the vertical check bytes. Once again code C is used

vertically and now we take the i^{th} byte from each block and write the two check bytes in the i^{th} bytes of the fixed two blocks. The Double Bundle System is somewhat better than the Single Bundle for independent errors and is very much more powerful for bursts.

The proposed system would work as follows. The data packets would be encoded with one of the horizontal (or data block) codes:

- (i) odd parity bytes
- (ii) two-way parity check
- (iii) code C .

A bundle might also be formed if the information transmitter wishes. The vertical codewords are built up as the data packets are sent down the channel. When the pre-defined bundle length has been reached the check bytes can be calculated and two (or one for single-) packets of 28 check bytes, encoded with the horizontal code, are sent.

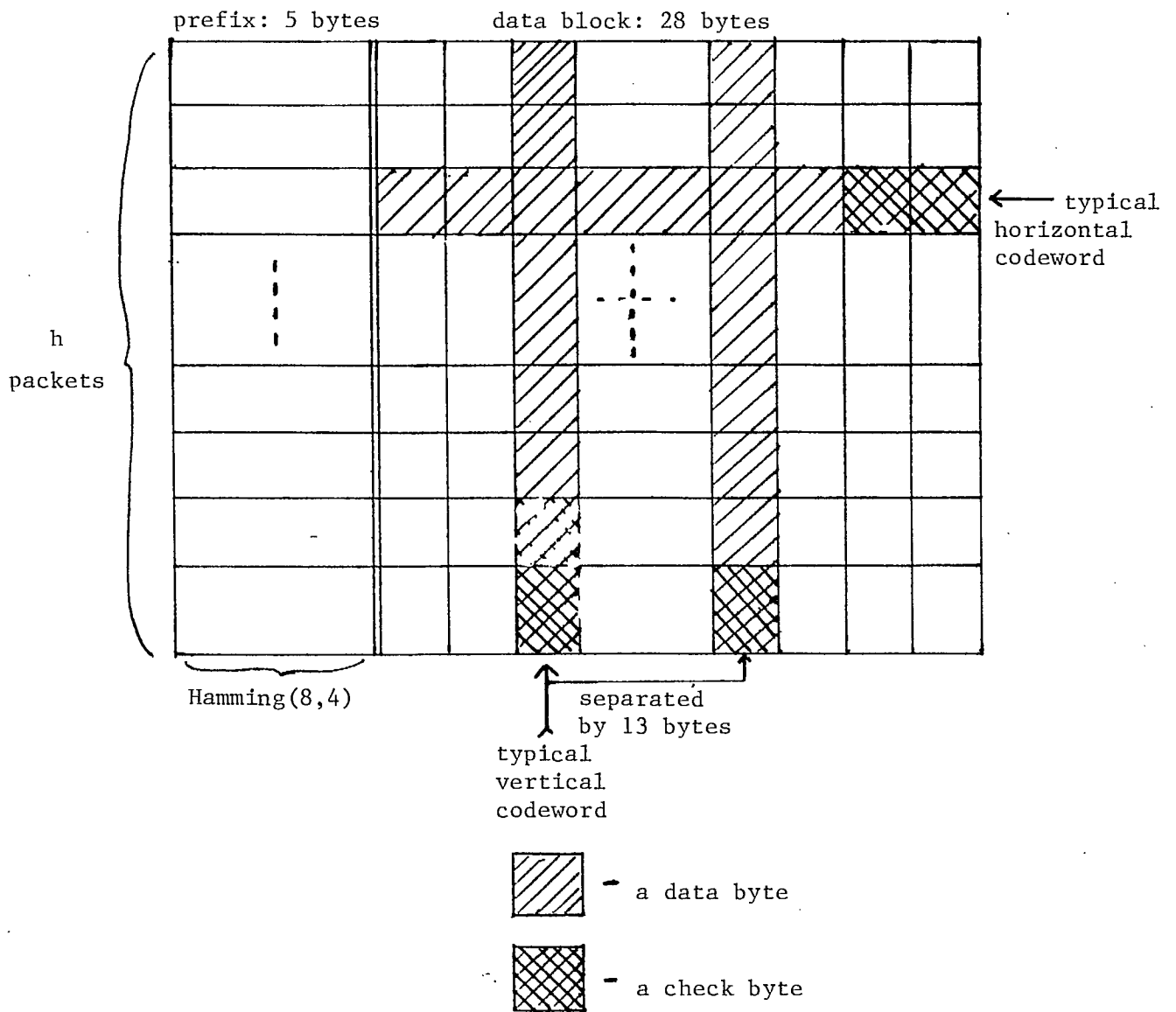


Figure 1 The arrangement of the Single Bundle Code

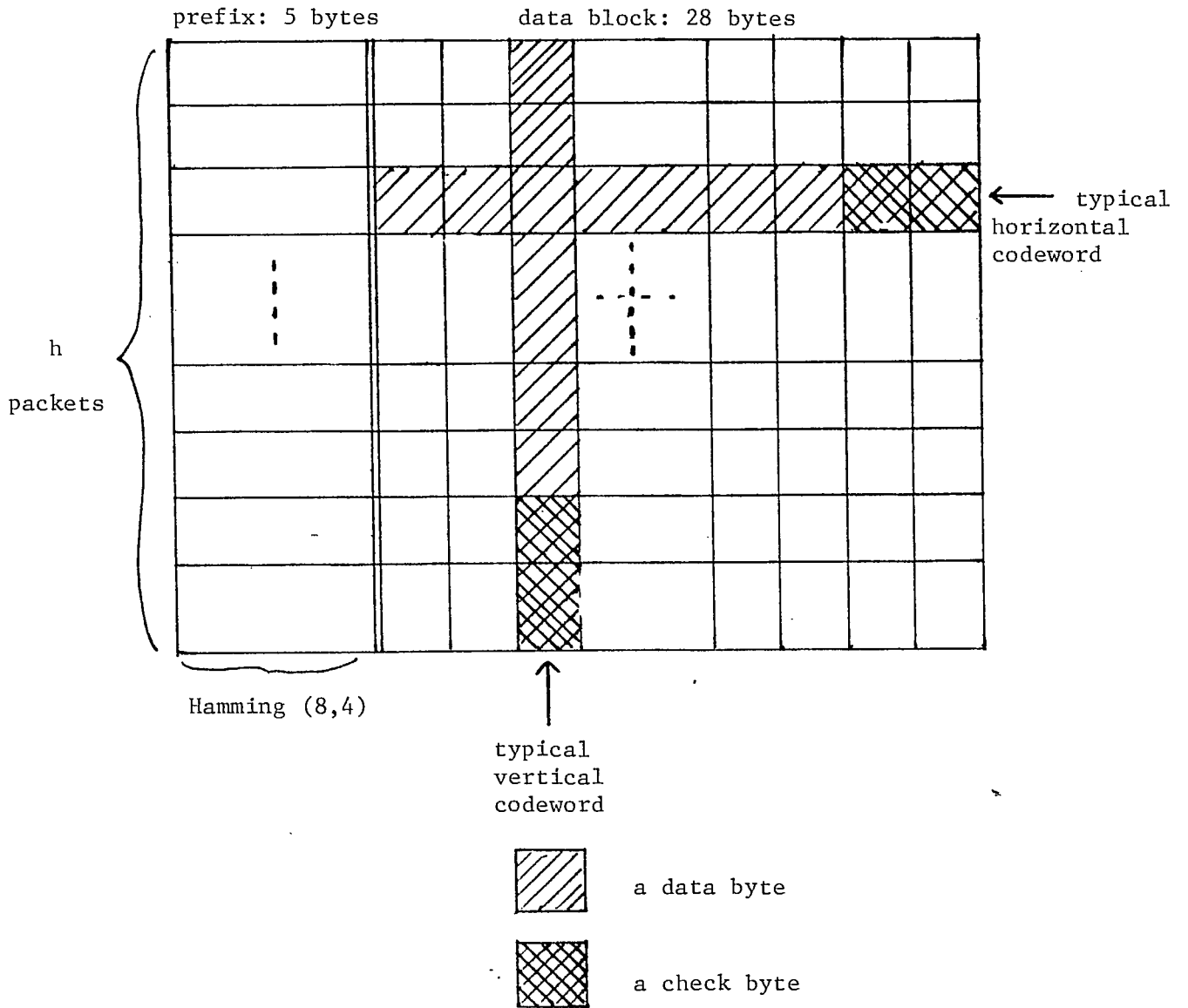


Figure 2 The Arrangement of the Double Bundle Code

1.3 Correction Capabilities of the Double Bundle Code

The Hamming codes in the prefix can correct a single bit error in each of the five bytes and detect any double error. Essentially all undetected errors will result in a mis-identification of the packet and the loss of a data block. Whole packets may be lost by synchronization failures.

The horizontal code is one of three choices: Parity-Only, Parity Product, code C. The first is an error detecting code while the second is a single-error correcting double-error detecting code (SEC-DED). The code C is single byte-error correcting and also corrects a double byte-error if the two erroneous bytes each show a parity failure. (This second feature is used in the vertical codewords below.)

The Bundle Code uses 26 vertical codewords from code C, storing the check bytes in two data blocks, and insists that all data blocks of the bundle, including the check blocks, are encoded with code C. To decode the bundle, the packets are received and decoded using the prefix and horizontal codes. The decoded data blocks are stored in a buffer. Any missing data blocks are written into this buffer as a string of bytes with a parity failure i.e., in the odd parity NABTS context they can be written in as a string of zeroes.

The vertical codewords are now decoded bitwise. Each vertical codeword contains one byte from each data block. Many sets of errors will be decoded in this place. For example:

- any burst of length at most 33 bytes will be corrected if the horizontal codewords have cleaned out any other errors,
- one or two missing data blocks will be replaced if the horizontal codewords have cleaned out any other errors,
- any pattern less than 6 errors anywhere in the bundle will be corrected,
- a scattering of short bursts has a good chance of being corrected.

This ends the "Single Decoding" of the Double Bundle Code.

At this point one could go back to the horizontal code and correct any correctable patterns left by the Single Decoding. Then one could go on to the vertical codewords. In effect, we run the full Bundle Decoder twice. We call this Double Decoding. Double Decoding provides some benefits, but there is only an insignificant improvement in repeating a third time.

So far we have been describing the full power of the Bundle System obtained by using horizontal and vertical codewords from Code C. A reduced version of the system might be of use in particular environments. One might use Parity-Product codewords (SEC) on the

horizontal data blocks and also for the vertical codewords. The decoder could recognize this since one of the prefix bytes will indicate the horizontal code and a single check packet could be used as a flag that vertical codewords are from the Parity-Product code.

In the report [3] we studied the Single Bundle System in depth. The benefit from using a Double Bundle System is an improvement in performance with both independent errors and with bursts. Moreover, the increased redundancy of the Double Bundle System can be offset by using longer bundles without degrading performance significantly.

1.4 Other Proposals for Teletext Codes

In this section we will describe some of the previous literature by other authors relevant to our subject.

The Parity-Product, or row-column, code has often been studied. It was suggested as a teletext code by Sablatash and Storey [7], [12]. In their paper they discuss the performance of this code with independent errors.

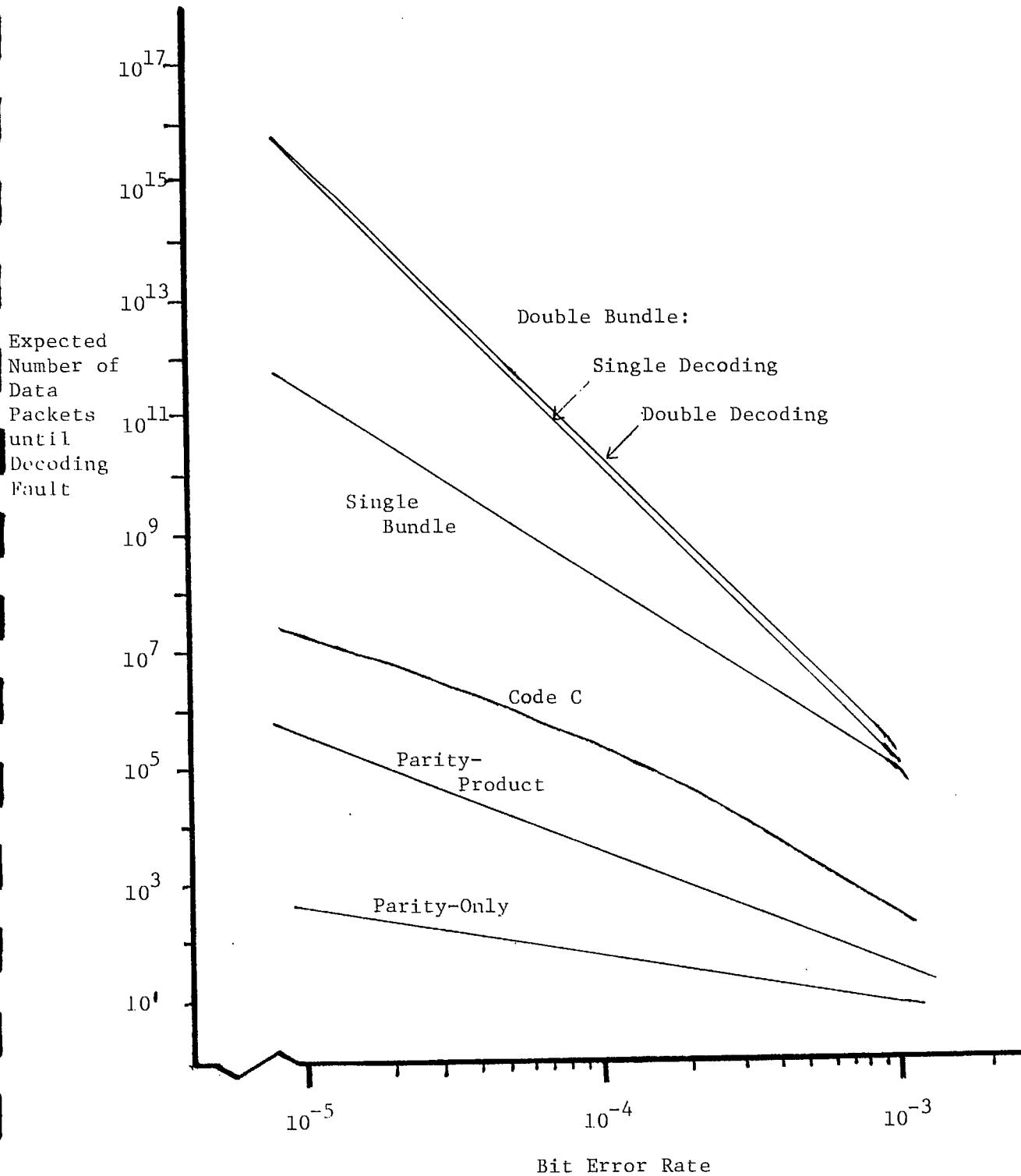
The Japanese have proposed a different packet code which we discuss in Chapter 3.

Bhargava, Allard and Seguin [4], [5] proposed a novel data packet code using one byte of parity checks (the same as the Parity-Product). This code is defined in such a way that all single

errors are correctable and a double error affecting two different bytes is correctable. The code is defined by finding an appropriate set of 8×8 binary matrices. We showed in [14] that this code essentially meets the bound for a probability of decoding error in a one-byte data block code. Decoding is feasible but not outstandingly quick. Upwards extension to a two byte code has not been examined.

In a report [6] of February, 1983, Hari, Seguin, Bhargava and Allard examined the performance of two Reed-Solomon codes defined using bytes as symbols, with the Broadcast Teletext System in the back of their minds. In fact they study a code of length 27 bytes with 2 check bytes and a code of 28 bytes with 3 check bytes (hence not exactly on target for NABTS). The first of these codes is essentially Code C, shortened by one byte and using the α and α^3 in place of 1 and α in the defining rule for the code ($C(x)$ is a codeword (polynomial) if and only if $C(1) = C(\alpha) = 0$). Therefore their code should have roughly the same performance as Code C with independent errors. In fact they do get the same results; compare our Figure 1.3 with [6, Fig. 5.5, page 78]. Their choice of generator polynomial (i.e. $C(x)$ is a codeword if and only if $C(\alpha) = C(\alpha^3) = 0$) means that their code does not extend the Parity-Product code.

Figure 1.1 Comparison of Several Codes with
Independent Errors



1.5 Performance Results with Independent Errors

So far we have only compared codes on the assumption of independent errors. We use as a measure the expected number of data packets until decoding fails, i.e. an error which is not correctable. Our results are displayed in Figure 1.1. Note the small benefit from using Double Decoding (cycling the Bundle Decoder twice) and the small benefit from Double Bundle over Single Bundle at higher rates. In fact, the reason for going to a Double Bundle is to give better performance with burst errors. As discussed in [3] the benefit of either Bundle code with independent errors is to overcome decoding failures in the prefix code by replacing one or two data blocks.

The comparison with the Japanese code BEST is contained in Chapter 3.

1.6 Decoding the Double Bundle

We have carried through an implementation of a Double Bundle Decoder in MC6809 software. This is a look up table decoder which uses *at most*:

- 3.5h milliseconds,
- 364 bytes of program,
- 256 bytes of look-up table.

The decoding is greatly simplified by using the same code for horizontal and vertical codewords and by taking only 1 byte from each data block for each vertical codeword (as in Double Bundle).

2. PERFORMANCE OF THE DOUBLE BUNDLE CODE WITH INDEPENDENT ERRORS

2.1 Results

The most common theoretical benchmark used for assessing error correcting codes is their performance on channels with independent errors. Such channels do arise at least occasionally in practice and a channel with burst errors may well have a background of independent errors.

We suppose that each bit is received in error with probability p with p in the range 10^{-3} down to 10^{-5} . At each bit error rate a particular code has a fixed probability P_{CD} of correctly decoding a packet. We use the expected number of correct packets before an undecodable one arrives. We approximate this by $1 / (1 - P_{CD})$.

We compare six codes, (four of which have already been compared in [3]). The codes are

- Parity-Only
- Parity-Product
- Code C
- Single Bundle ($h = 9$)
- Double Bundle ($h = 13$): single decoding
- Double Bundle ($h = 13$): double decoding.

The length of the Bundle Codes were chosen to give them comparable rates. The results are in Table 2.1; they are portrayed graphically in Figure 1.3 .

Table 2.1 Expected Number of Packets until Incorrect Decoding:
various codes

Code / BER →	10^{-3}	4×10^{-4}	10^{-4}	10^{-5}	Rate
Parity-Only	1.0e1	1.7e1	5.4e1	5.0e2	.82
Parity-Product	5.4e1	2.9e2	4.2e3	4.2e5	.79
Code C	5.9e2	8.1e3	5.4e5	1.9e7	.77
Single Bundle	1.1e5	2.3e6	1.7e8	1.4e11	.69
Double Bundle - S.D.	2.1e5	2.6e7	2.7e10	2.7e15	.66
Double Bundle - D.D.	3.3e5	2.9e7	2.8e10	2.7e15	.66

The effect of varying the Bundle Length is revealed in Tables 2.2 and 2.3 .

Table 2.2 Expected Number of Packets until Incorrect Decoding:
various lengths, single decoding

Bundle Length (h)	Bit Error Rate			
	10^{-3}	3×10^{-4}	10^{-4}	10^{-5}
5	6.0e5	2.4e8	5.8e10	5.5e15
6	5.2e5	2.2e8	5.2e10	5.1e15
7	4.5e5	1.9e8	4.6e10	4.6e15
8	3.9e5	1.7e8	4.2e10	4.1e15
9	3.4e5	1.6e8	3.8e10	3.7e15
10	3.0e5	1.4e8	3.5e10	3.4e15
11	2.7e5	1.3e8	3.2e10	3.2e15
12	2.4e5	1.2e8	2.9e10	2.9e15
13	2.1e5	1.1e8	2.7e10	2.7e15
14	1.9e5	1.0e8	2.6e10	2.5e15
15	1.6e5	9.8e7	2.4e10	2.4e15

Table 2.3 Expected Number of Packets until Incorrect Decoding:
various lengths, double decoding

Bundle Length (h)	Bit Error Rate			
	10^{-3}	3×10^{-4}	10^{-4}	10^{-5}
5	6.9e5	2.5e8	5.9e10	5.5e15
6	6.1e5	2.2e8	5.2e10	5.1e15
7	5.5e5	2.0e8	4.7e10	4.6e15
8	4.9e5	1.8e8	4.2e10	4.1e15
9	4.5e5	1.6e8	3.8e10	3.7e15
10	4.1e5	1.5e8	3.5e10	3.4e15
11	3.8e5	1.4e8	3.2e10	3.2e15
12	3.5e5	1.3e8	3.0e10	2.9e15
13	3.3e5	1.2e8	2.8e10	2.7e15
14	3.1e5	1.1e8	2.6e10	2.5e15
15	2.9e5	1.0e8	2.4e10	2.4e15

What we see is that bundle length is not very crucial over this range and that performance levels are all acceptable. Double decoding is beneficial if the BER is high enough but is not usually very helpful. It may be very useful for the range 10^{-3} to 10^{-2} .

2.2 Methods

We are studying the performance of the Double Bundle code in an environment of independent errors. We will therefore assume that errors occur in the data bits with a fixed probability p (BER) and for convenience we set $q = 1 - p$ and $x = p/q$. (This x is essentially the same size as p .) We assume that the message bits do not influence the pattern of errors so that we can and generally do assume that a message of all zeroes has been sent.

We would like to calculate the probability of a correct decoding of the Double Bundle in the two cases of Single and Double Decoding. We will deal with Single Decoding first. There are three possible scenarios in this case depending on the number (0, 1 or 2) of prefix codes that fail. The probability that a prefix code is correctly decoded is

$$\text{PrefixCor} = (q^8 + 8 \cdot pq^7)^5 = (q^8(1+8x))^5 .$$

We will also need the probability of a correct decoding of a horizontal codeword and of the Bundle Code which we denote respectively HorCor , Bundle Cor .

If the bundle includes h packets in all (including check packets) we can write down formulae for the probability that a correct decoding is achieved after 0, 1 or 2 prefix failures:

$$\text{All Prefix Cor} = \text{PrefixCor}^h \text{ Bundle Cor}$$

$$\text{One Failed Prefix} = h(1 - \text{PrefixCor})(\text{PrefixCor} * \text{HorCor})^{h-1}$$

$$\text{Two Failed Prefixes} = \binom{h}{2}(1 - \text{PrefixCor})^2(\text{PrefixCor} * \text{HorCor})^{h-2}.$$

We are left with the calculation of HorCor and BundleCor .

The horizontal codewords are assumed to be corrected by the bitwise decoder. Thus any error in a single byte is corrected and any error in two bytes which changes the parity of each is correctable.

Since there are 28 bytes in each codeword the probability of an error

pattern corrupting one byte is $28 \sum_{t=1}^8 \binom{8}{t} p^t q^{224-t}$ which we can

write as

$$q^{224} 28((1+X)^8 - 1).$$

The probability of the second type of error is

$$q^{224} \binom{28}{2} \sum_{t=2}^8 \sum_{s=\max(t-3,1)}^{\min(t,4)} \binom{8}{2s-1} \binom{8}{2t-2s+1} X^{2t}.$$

Here we are summing over the patterns of $2t$ errors of which $2s-1$

fall in one byte and $2(t-s)+1$ fall in the other. Of course

$1 \leq 2s-1 \leq 7$ and $2s-1 < 2t$ and $2(t-s)+1 < 2t$ and

$1 \leq 2(t-s)+1 \leq 7$. This all boils down to $\max(t-3, 1) \leq s \leq \min(t, 4)$.

Finally, we include the expression for a correct reception and obtain

$$\text{HorCor} = q^{224} \left[1 + 28((1+X)^8 - 1) + \sum_{t=1}^{28} \sum_{s=\max(t-3,1)}^8 \binom{8}{2s-1} \binom{8}{2t-2s+1} X^{2t} \right].$$

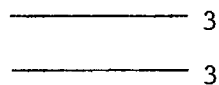
In the 1983 Final Report [3] we used $q^{224} [1 + 224X + \binom{224}{2} X^2]$ which gives nearly the same result being the probability of 0, 1 or 2 bit errors.

We come to the probability of an error pattern correctable by Single Decoding of the bundle itself. We write

$$\text{BundleCor} = 1 - q^{224} \left[\sum_{k=1}^{\infty} I_k X^k \right]$$

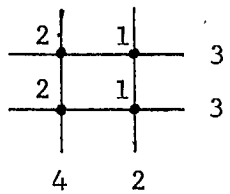
where I_k is the number of uncorrectable error patterns of k errors. Note that I_k is a non-linear function of h . In fact $I_k = 0$ for $k = 1, 2, 3, 4$ and 5 . The point is that the horizontal code can correct double errors so it will clear out all errors from packets with only 1 or 2. Moreover, if only one horizontal codeword is uncorrectable then the vertical codewords correct it. Thus the minimal pattern of uncorrectable errors is two packets with 3 errors in each.

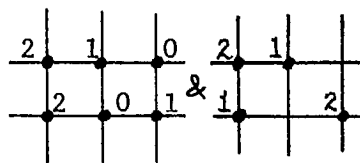
If six errors are uncorrectable by the double bundle code then they occur as three in each of two codewords. We denote this by drawing two labelled horizontal lines:

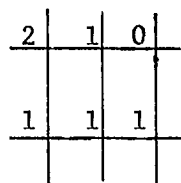


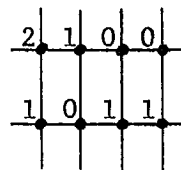
This is a type 3-3 error pattern. Not all of these are uncorrectable.

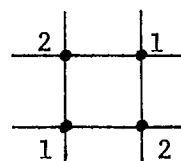
We draw in vertical lines to represent vertical codewords and use numbers at the intersections to denote the number of errors in the byte where the vertical and horizontal codewords meet. The following arrangements are uncorrectable:

6(a)  $\binom{h}{2} \binom{v}{2} 2 \binom{8}{2}^2 8^2$

6(b)  $\binom{h}{2} \binom{v}{3} 15 \binom{8}{2}^2 8^2$

6(c)  $\binom{h}{2} \binom{v}{3} 6 (2) \binom{8}{2} 8^4$

6(d)  $\binom{h}{2} \binom{v}{4} 4 (2) 3 \binom{8}{2} 8^4$

6(e)  $\binom{h}{2} \binom{v}{2} 2 \binom{8}{2}^2 8^2$

Here we denote by h , the number of horizontal codewords and by v the number of vertical codewords (i.e. $v = 28$). The formulae at

the right enumerate the number of different error patterns of the given form. Note that the type 6(e) patterns are the only ones which are not correctable on a second pass; count them as I_6^* .

This gives

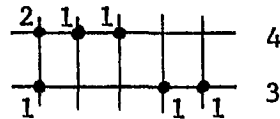
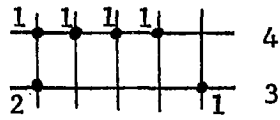
$$\begin{aligned} I_6 &= \binom{h}{2} \left\{ \binom{v}{2} 4 \binom{8}{2}^2 8^2 + \binom{v}{3} [15 \binom{8}{2}^2 8^2 + 6 \cdot 2 \cdot \binom{8}{2} 8^4] + \binom{v}{4} 4 \cdot 2 \cdot 3 \binom{8}{2} 8^4 \right\} \\ &= \binom{h}{2} \binom{v}{2} (200704) + \binom{h}{2} \binom{v}{3} (2128756) + \binom{h}{2} \binom{v}{4} (2752512) \\ I_6^* &= \binom{h}{2} \binom{v}{2} 2 \binom{8}{2}^2 8^2 = \binom{h}{2} \binom{v}{2} (100352) . \end{aligned}$$

Moving to patterns of 7 errors we see that many uncorrectable patterns are in fact an uncorrectable pattern of 6 errors with one more error in any of the $(h-2)224$ remaining bits. We denote these by types 7(a) through 7(e) and they count for $I_6(h-2)224$ patterns of uncorrectable errors.

The remaining uncorrectable patterns of seven errors are of type 3-4. The packet with 3 errors has 2 in one byte and 1 in another; the packet with 4 errors has errors in at least 3 bytes or else 3 in one and 1 in another. We list the possibilities with their enumerations.

7(f)

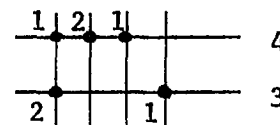
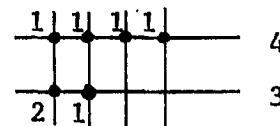
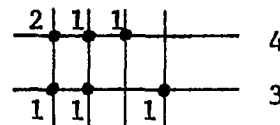
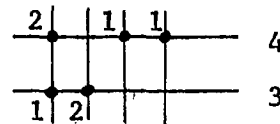
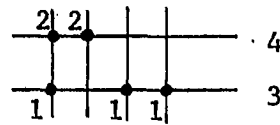
(2-1-1-1-1)



$$\binom{h}{2} \binom{v}{5} 100 \cdot 8^5 \binom{8}{2}$$

7(g)

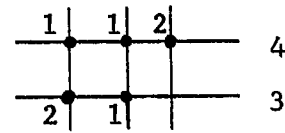
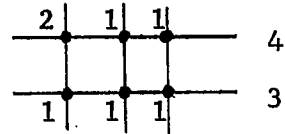
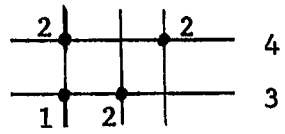
(3-2-1-1)



$$\binom{h}{2} \binom{v}{4} 2 \cdot 4 \cdot 3 \cdot 8 \binom{8}{2} \left(\binom{8}{2} 8^2 \cdot 2 + 2 \cdot 8^4 + 8^4 + \binom{8}{2} 2 \cdot 8^2 \right)$$

7(h)

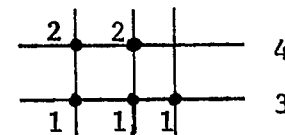
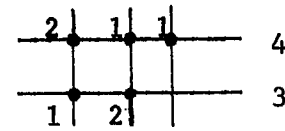
(3-2-2)



$$\binom{h}{2} \binom{v}{3} 2 \cdot 3 \cdot \binom{8}{2} 8 \left(2 \binom{8}{2}^2 + 8^4 + 2 \binom{8}{2} 8^2 \right)$$

7(i)

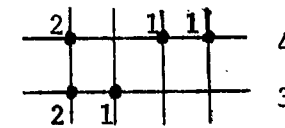
(3-3-1)



$$\binom{h}{2} \binom{v}{3} 2 \cdot 3 \cdot 8^3 \binom{8}{2}^2 (2+1)$$

7(j)

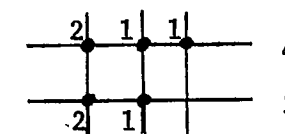
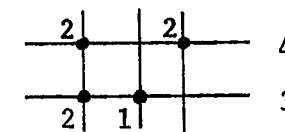
(4-1-1-1)



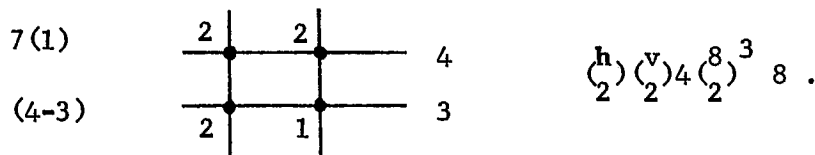
$$\binom{h}{2} \binom{v}{4} 2 \cdot 4 \cdot \binom{8}{2}^2 \cdot 3 \cdot 8^3$$

7(k)

(4-2-1)



$$\binom{h}{2} \binom{v}{3} 2 \cdot 6 \cdot \binom{8}{2}^2 \left(\binom{8}{2} 8 + 8^3 \right)$$



The result is an expression for I_7 :

$$I_7 = I_6 (h-2)224 + \binom{h}{2} \left[\binom{v}{5} (91750400) + \binom{v}{4} \overset{1.5276E8}{\cancel{452700}} \right. \\ \left. + \binom{v}{3} (26578944) + \binom{v}{2} (702464) \right].$$

The weight 8 uncorrectable error patterns are distributed in horizontal codewords in one of the following ways: 3-3-1-1, 3-3-2, 4-3-1, 4-4, 5-3. The first three types can be counted using I_6 and I_7 ; they account for essentially all of the uncorrectable error patterns. We count them by noting that for example a type 3-3-1-1 uncorrectable error consists of an uncorrectable pattern of six errors (type 3-3) and one more error in a third packet. (Note that the weight 7 type 4-3 uncorrectable error patterns are enumerated as $I_7 - I_6 (h-2)224$.)

<u>type</u>	<u>number</u>
3-3-1-1	$I_6 \binom{h-2}{2} 224^2$
3-3-2	$I_6 (h-2) \binom{224}{2}$
4-3-1	$[I_7 - I_6 (h-2)224] (h-2)224$

We take $I_8 \approx I_6 \binom{h-2}{2} 224^2 + I_6 (h-2) \binom{224}{2} + [I_7 - I_6 (h-2)224] (h-2)224$.

For the case of nine errors we have many types, but most of them are covered by adding one additional error to a weight 8 error in one of $(h-2)$ packets. Similarly, for 10 errors:

$$I_9 \approx I_8 (h-2)224$$

$$I_{10} \approx I_9 (h-2)224 .$$

(Using these estimates over counts some types and ignores others.)


```

PROGRAM Independent;
{Performance of the double bundle with independent errors}
{March 26, 1984}

```

```

VAR P:real;
    h,flag:integer;

```

```

FUNCTION Power(X:real;M:integer):real;
VAR I:integer;
    POW:real;
BEGIN
    POW:=1;
    IF M>0 THEN FOR I:=1 TO M DO POW:=POW*X;
    IF M<0 THEN FOR I:=-1 DOWNT0 M DO POW:=-POW/X;
    POWER:=POW
END;{POWER}

```

```

FUNCTION Power2(X:real;M,K,SI:integer):real;
VAR SUM,TERM:real;
    I:integer;
BEGIN
    Sum:=1;Term:=1;
    For I:=1 to K do
    BEGIN
        Term:=Term*X*SI*(M-I+1)/I;
        Sum:=Sum + Term
    END;
    POWER2:=Sum
END;{Power2}

```

```

FUNCTION inf(a,b:integer):integer;
begin
    inf:=a;
    if b<a then inf:=b
end;

```

```

FUNCTION sup(a,b:integer):integer;
begin
    sup:=a;
    if b>a then sup:=b
end;

```

```

PROCEDURE LENGTH(h:integer;P:real);

```

```

VAR Q,X,H2,H3:real;
    i,s,t:integer;
    term,sum2,sum,V,V1,V2,V3,V4,V5:REAL;
    I6,I7,I8,I9,I10,ExpDecFault,ProbCor:REAL;
    HorCor2,PrefixCor,ByteCor,HorCor,BundleCorrectable:real;
    AllPrefixesCorrect,OneFailedPrefix,TwoFailedPrefixes:real;
    C,D: array [0..8] of real;

```

```

BEGIN
Q:=1-P;X:=P/Q;
V:=28;
H2:=H*(H-1)/2;H3:=H2*(H-2)/3;
V2:=V*(V-1)/2;V3:=V2*(V-2)/3;V4:=V3*(V-3)/4;V5:=V4*(V-4)/5;

D[0]:=1;for I:=1 to 7 do D[I]:=D[I-1]*(224-I+1)/I;

{I6:=H2*V2*1.00352e5;}
I6:=H2*V2*2.007E5 + H2*V3*2.1289E6 + H2*V4*2.753E6;
{I7:=H3*V2*6.7436E7 + H2*V3*6.02112E5 + H2*V2*5.268E5;}
I7:=I6*((h-2)/3)*24*v + H2*V5*9.175E7 + H2*V4*1.528E8 +
    H2*V3*2.658E7 + H2*V2*7.025E5;
I8:=I7*(224*(H-2));
I9:=I8*(224*(H-2));
I10:=I9*(224*(h-2));

C[0]:=1;for I:=1 to 8 do c[i]:=c[i-1]*(8-i+1)/i;

sum:=0;
for t:=2 to 8 do
begin
    term:=0;
    for s:=sup(t-3,1) to inf(t,4) do term:=term + c[2*s-1]*c[2*t-2*s+1];
    sum:=sum+term*power(X,2*t);
end;

sum2:=0;for t:=3 to 8 do sum2:=sum2+c[t]*power(X,t);

PrefixCor:=Power(power(Q,8)*(1+8*X),5);

HorCor:=Power2(P,224,10,-1)*(1+28*(8*X + 28*X*X +sum2)+378*(64*X*X + sum));

BundleCorrectable:=1 - Power2(P,224*h,10,-1)*(I6*Power(X,6) + I7*Power(X,7)
    + I8*Power(X,8) + I9*Power(X,9) + I10*Power(X,10));

AllPrefixesCorrect:=Power(PrefixCor,h) * BundleCorrectable;

OneFailedPrefix:=h*(1-PrefixCor)*Power(PrefixCor*HorCor,h-1);

TwoFailedPrefixes:=H2*Power((1-PrefixCor),2)*Power(PrefixCor*HorCor,h-2);

ProbCor:=AllPrefixesCor+OneFailedPrefix+TwoFailedPrefixes;

ExpDecFault:=1/(1-ProbCor);
write((ExpDecFault*(h-2)):5, ' ');
END;

```

```
BEGIN {The program proper....}
  writeln('Hardcopy? 0/1 ');readln(flag);
  if FLAG=1 then writeln(chr(27),'[10;75s',chr(27),'[5;60r');
  writeln('Expected number of PACKETS until decoding fault - Double Bundle'
  writeln('                                using Single decoding');
  writeln;writeln('*****');writeln;writeln;
  write('Bit error rate-----> ');
  writeln(' 1e-3      4e-4      3e-4      1e-4      1e-5');
  writeln('Bundle length');
  FOR h:=5 TO 15 DO
    begin
      write(h:2,' ');length(h,1e-3);
      length(h,4e-4);length(h,3e-4);
      length(h,1e-4);length(h,1e-5);writeln
    end;
END.
```

3. AN ASSESSMENT OF THE PROPOSED JAPANESE TELETEXT CODE

3.1 The Japanese Proposal and the NABTS

The Japanese have proposed [2] that a particular majority-logic decodable cyclic code, which they have called "BEST", be used as error protection for teletext data packets. They have described this code as covering all bits of the packet after the bit and byte synchronization bytes. Thus the control bits for addressing, continuity count and packet structure would be encoded along with any data bits. They do not make use of or refer to using bytes of odd parity or of a set of five Hamming encoded prefix bytes.

The code "BEST" is capable of correcting any pattern of 8 bit errors that corrupt a single data packet. The field data reported in [8] suggest that this could take care of about 99.3% of erroneous packets and deliver the quality of service that the Japanese desire. In addition, the choice of a majority logic decodable code will allow an LSI hardware implementation. This would in turn allow this powerful code to be decoded on the fly as the packets arrive. Their idea is to perform error correction at the "signal level" ([11], page 3, Table I, Item 3.9).

To achieve its correction power this code "BEST" must use 82 check bits. The proposal [9] is based on a 34 byte data packet whereas

the NABTS (draft) specifies 33 bytes ([1], Sec. 3.1, page 9). Moreover, the NABTS (draft) assumes that all bytes have odd parity ([1], Sec. 3.3, page 10) and that there is a five byte prefix encoded with a Hamming [8,4] code ([1], Sec. 3.2, page 9). The Japanese assessment of their code does not refer to any of this structure. On a 33 byte packet BEST would have $264 - 82 = 182$ message bits, some of which are control bits (20 in the NABTS scheme). So the information rate is about 69%. However, if we follow the NABTS prescriptions then we are committed to $20 + 28$ further bits of parity checks. This would mean a total of $48 + 82 = 130$ check bits for $264 - 130 = 134$ message bits; an information rate of 51%. Thus the "BEST" code is only practical if all other parity checking and error correction coding is removed.

The document [9] does not make any comment on implementation. How will the code be organized with respect to the byte structure? Of course 8 does not divide 82 so there needs to be some comment on where the check bits will sit in the packet.

More information on the code itself, which is obtained from a perfect, cyclic difference set, can be obtained in [10], page 134.

3.2 The Comparison of Various Packet Coding Systems

The document [9] compares the code "BEST" with several other packet codes (of a variety of rates) in two ways. In the first study the probability of a page with an uncorrectable error is calculated for each code in the context of independent errors. The results were presented as a graph over the range of bit error rates from 10^{-5} to 10^{-2} ([9], fig. 2). The codes were also compared in a field trial. We have attempted to reproduce the theoretical results and extend them to include the code C and the Bundle coding system which we have proposed in [3] as a teletext coding scheme.

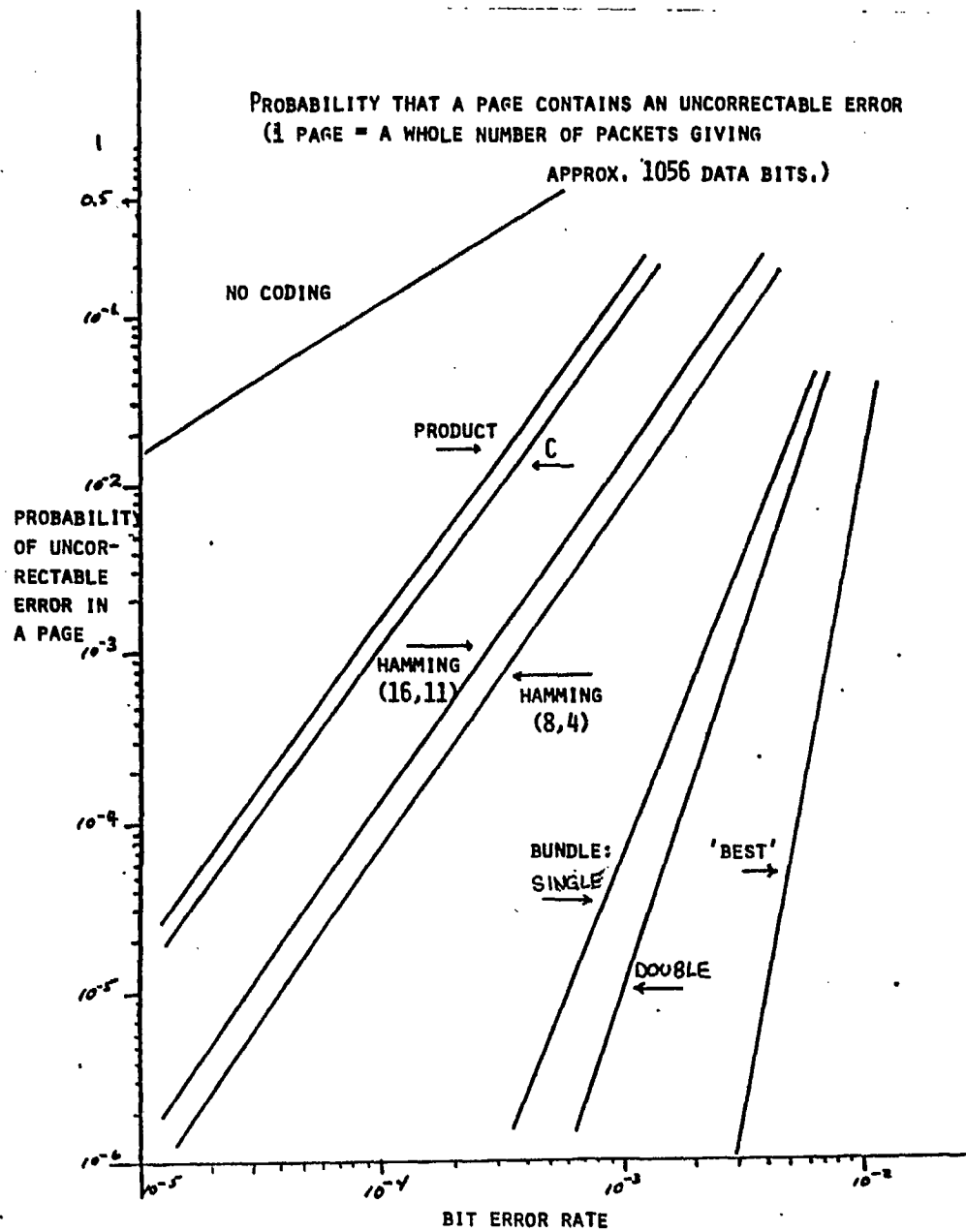
The document [9] is not clear on the size of a page. This is important since the codes cover a wide range of rates. They quote a page length of 120 characters but do not relate this to bits or bytes. We found that by charging "BEST" with 20 control bits and taking a page as about 1056 information bits we could come reasonably close to their results. In our study we have used a whole number of packets for each page. The number of packets depends on the rate of the code but was chosen to come as close to 1056 information bits per page as possible. Our results are presented in Table 3.1 and Figure 3.1.

Table 3.1 Parameters Used in Comparing the Codes

Code	Information rate	Number of packets/page	Number of data bits/page
No coding	1.00	5	1056
Prefix with Hor. & Ver. Parity	0.79	6	1134
Prefix with Code C	0.77	6	1092
Code BEST	0.69	6	972
Bundle System (code C)	0.67	7	1092
Prefix with 14 Hamming [16,11]	0.66	7	1078
Prefix with 28 Hamming [8,4]	0.50	9	1008

We observe that the Bundle coding system based on code C actually achieves the target of a page error rate below 5×10^{-2} at a bit error rate of 5×10^{-3} which the Japanese have set in [8].

In addition to the theoretical studies, the codes were compared in [9] in a number of field trials. It was correctly observed that the channels were not introducing independent errors but the exact nature of the bursts encountered can not be deduced from the data presented. A comparison of the observed page error rates [8], Fig. 2, with the theoretical curves [9], Fig. 2 show that in fact the performance was usually but not always worse than that predicted from



independent errors. Without more detailed information about the burst patterns we cannot predict the performance of the Bundle coding system on real channels. We note though that the Bundle code should work better on a bursty channel than code "BEST" since the latter code is restricted to bursts of length 8 bits per packet. The Japanese strategy appears to be to select a code which performs very much better than their target on the theoretical independent error channel so that even when there are bursts it doesn't degrade below the target.

3.3 Method of Calculation

The results in Section 3.2 were calculated in a reasonably straightforward way. A single bit-error correcting code of length t used on a channel with independent errors that arise with probability of correct decoding given by $q^t + tpq^{t-1}$ where $q = 1 - p$. So we obtain some of the required formulas for probability of correct decoding of a packet:

No Code	: q^n	
Hor. & Ver. Parity	: $q^n + npq^{n-1}$	= $q^n(1 + nX)$
34 bytes of Hamming [8,4]	: $[q^8 + 8pq^7]^{34}$	= $q^n(1 + 8X)^{34}$
17 bytes of Hamming [16,11]	: $[q^{16} + 16pq^{15}]^{17}$	= $q^n(1 + 16X)^{17}$

where $n = 272$ and $X = p/q$.

The formulas for Code C and the Double Bundle Codes have been developed in Chapter 2. The Single Bundle Code is dealt with in detail in [3], Table 1.3, page 15.

The code "BEST" is an 8 error correcting code, so the probability of a correct decoding is

$$PC = \sum_{i=0}^8 \binom{n}{i} p^i q^{n-i} .$$

This number is generally very close to one so we in fact calculate the complement,

$$1 - PC = \sum_{i=9}^n \binom{n}{i} p^i q^{n-i} .$$

Chapter 4. Performance in a More Real World

4.1 Afterall Errors May Not be Independent

The assumption that errors are independent events is convenient for analysis but is not necessarily what happens in any real channel. There are several ways to try to estimate the performance of an error correction system in a real or more realistic channel. In previous reports [14], [15] we have used model channels to assess codes. A variety of stochastic processes were hypothesized and expected performance was calculated for a set of codes. This is only relevant to the real expected performance if the models describe the real channels. We have not found in the literature, any reports of stochastic models fitted to channels relevant to Broadcast Teletext System at the high bit transmission rate used by the System (5.727272Mbits/Sec). It is hard to see how to modify a description of a channel with a low bit transmission rate to describe what would happen if the bit rate was increased greatly.

Better than modelling is to measure real channels. Field data became available to us in February, 1984 and we have worked on using this data. Ultimately we should be able to get a very good idea of the level of coding required. We may eventually be able to describe one or several models of the channels on which the data was collected.

4.2 Analyzing Field Data for Code Performance

The field data in our analysis system exists as a computer file consisting of a sequence of integers. The data looks like one of the following:

```
x(20),
-1 y n1 e1 n2 e2 . . . . ny ey,
-2.
```

The x means x packets in a row received correctly. The -1 flags a packet with y byte errors. The integers n_i, e_i say that byte n_i has error e_i . The error is the decimal representation of the error byte. Eg. 11, 64 means error 01000000 in byte 11. The flag -2 means that a packet is missing (through synchronization failure or prefix code failures). Since we might have a very large number of correct packets in a row (on a good day), we may write several numbers into the file for that one series. Eg. we might have:

```
17 - 1 1 4 3 200 19 -2 -1 2 3 8 21 17 113 . . . .
```

meaning 17 correct packets
a packet with error (00000011) in byte 4
200 + 19 correct packets
a missing packet
a packet with error (0000 1000) in byte 3 and
error (0001 0001) in byte 21
113 correct packets
:
:
:

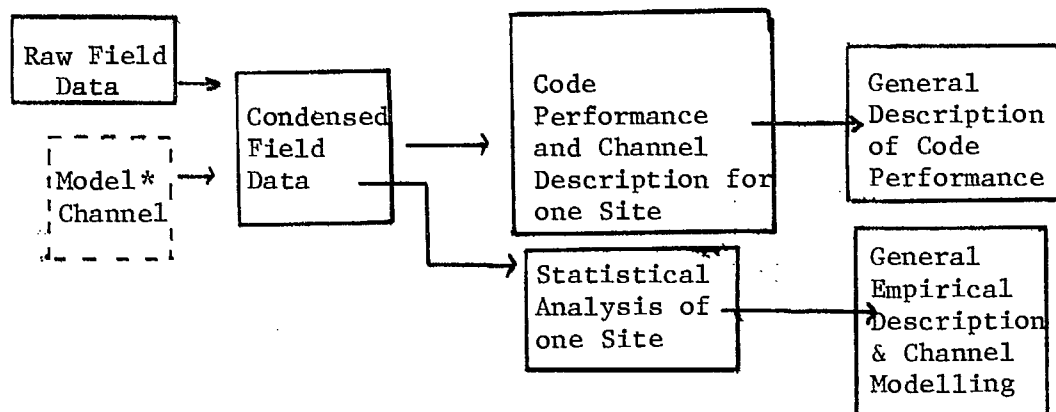
The data does not arrive from CRC in this form, but in a similar but expanded form. We have written the program DATACRUNCH of the Appendix to convert the field data to our format.

We have also written a program to analyze the condensed field data for code performance. This is the program FLDANALB. This program looks at the data in the context of a run-time supplied bundle length. It reports generally on the channel and then on the bundle code. The parameters calculated are:

- i) number of packets
- ii) number of packets with errors
- iii) number of missing packets
- iv) number of bit errors; BER
- v) the number of packets with i bit errors for $i = 1$ to 10.
- vi) the number of packets with at least i errors or missing for $i = 1$ to 3.
- vii) BUNDLE REPORT; for length h .
 the number of correctly received bundles
 the number of correct or correctable bundles
 the number of uncorrectable bundles.

Note that item vi) tells us how to assess the performance of the Parity-Product code and Code C.

This is of course just the first stage in analyzing the field data. The whole picture looks something like this.



Before the field data became available we had planned to use a model channel (*) to produce files of Condensed Field Data. This is however now irrelevant.

In conjunction with this analysis strategy we have also written programs such as QUICK PASS that do part of the analysis in a different way and hence provide a check on the program correctness. Ensuring internal consistency is a crucial part of a large programming project.

4.3 An Example: E184R-2

We illustrate our analysis with one particular site, namely E184R-2. This site used a Norpak Mark IV teletext decoder. The output of FLDANALD is given in Figure 4.1.

Figure 4-1: Output of FLDANALB

SITE 184R-2, MARK IV Decoder

1146 of 7489 packets were in error or missing
rate = 1.526E-1

0 of these were missing

The Bit Error Rate was
i.e. 1333 bit errors in 1.797E6 bits (\approx .3 sec)

i	#of packets with i errors
1	985
2	136
3	24
4	1
5	0
.	.
.	.
.	.
.	.
11	0

BUNDLE REPORT	length = 10 packets
750 bundles	
139 correct	
750 correctable	
0 uncorrectable (rate = 0)	

Note that the this site, during the test

		<u>Avg. #packets between failures</u>
Parity-Only failed with frequency	15.3%	6.6
Parity-Product	2.1%	46.6
Code C	.33%	300.3
Double Bundle (h=10)	0%	-

Since this is only 1 second at one site on one day we cannot conclude anything further.

Appendix: Software

Included here are three PASCAL programs as follows:

- (a) FLDANALB - a program which analyzes the performance of the Double Bundle Code on a "standard field-data" file.
- (b) DATACRUNCH - a program which produces the "standard field-data" file from a raw data file.
- (c) QUICKPASS - a program which counts the number of packets and the number of bad packets in a "standard field-data" file. This program is a check on FLDANALB which performs a more intricate calculation.

(a) PROGRAM FLDANALB;

{March 14, 1984

Analysis of bundle code performance on a file of Condensed
Field data}

```
var h,x,y,R,U,W,leftovers,n,e,i:integer;
    CBunLen, CBadPackets:integer;
    BitError,UnFix,CorBundle,Fix,TotalBundles:real;
    TotalPackets,BadPackets,Missing:real;
    flag,title,INFILE,OUTFILE:string;
    INDATA,OUTDATA:text;
    ErrDis:array[1..11] of integer;
    B:array [1..15,1..28] of integer;
```

```
{*****
*****}
```

```
PROCEDURE CLEARB;
```



```
var i,j:integer;
```

```
begin
```

```
for i:= 1 to 15 do for j:=1 to 28 do B[i,j]:=0
end;
```

```
{*****
*****}
```

```
PROCEDURE ErroneousBundle;
```

```
{This routine deals with a bundle which contains an error}
```

```
var odd,even,i,j,k:integer;
    cor:boolean;
```

```
begin
```

```
for i:=1 to CBadPackets do
```

```
begin
```

```
for j:=1 to 28 do write(B[i,j]);
```

```
writeln
```

```
end;
```

```
writeln('*****');
```

```
cor:=true;
```

```
{Horizontal Decoding}
```

```
for j:=1 to CBadPackets do
```

```
begin
```

```
odd:=0;even:=0;
```

```
for k:=1 to 28 do if (B[j,k] mod 2)=1 then odd:=odd+1
else if B[j,k]>0 then even
```

```
n:=even+1;
```

```
if ((even=0) and (odd<=2)) or ((even=1) and (odd=0)) then,
```

```
for k:=1 to 28 do B[j,k]:=0
```

```
end;
```

```
{Vertical Decoding}
```

```
for j:=1 to 26 do
```

```
begin
```

```
odd:=0;even:=0;
```

```
for k:=1 to CBadPackets do if (B[k,j] mod 2)=1 then odd:=odd+1
else if B[k,j]>0 then even:=e
```

```
ven+1;
```

```
if not(((even=0) and (odd<=2)) or ((even=1) and (odd=0))) then
```

```
cor:=false
```

```
end;
```

```
if cor=true then Fix:=Fix+1 else UnFix:=UnFix+1;
```

```
CBadPackets:=0;
```

```
CBunlen:=0;
```

```
ClearB;
```

```
end;[ erroneous bundle ]
```

```
{*****  
*****}
```

```
FUNCTION Weight(X:integer):integer;
```

```
{Returns the weight of X when it is converted to a binary string}
```

```
var s,w,t:integer;
```

```
begin
```

```
t:=128;w:=0;
```

```
while t>=1 do
```

```
begin
```

```
s:=x div t;
```

```
x:=x-s*t;
```

```
t:=t div 2;
```

```
w:=w+s
```

```
end;
```

```
Weight:=w
```

```
end;[ Weight function]
```

```
{*****  
*****}
```

```
{Main Program}
```

```
Begin
```

```
write('Name of the input file, please: ');readln(INFILE);
```

```
reset(INDATA,INFILE);
```

```
writeln('Hardcopy <Y/N>');readln(flag);
```

```
if (flag='y') or (flag='Y') then OUTFILE:='printer:' else OUTFILE:  
='console';
```

```
rewrite(OUTDATA,OUTFILE);
```

```
leftovers:=0;CBunLen:=0;CBadPackets:=0;BitError:=0;Missing:=0;CorB  
undle:=0;
```

```
BadPackets:=0;TotalPackets:=0;Fix:=0;UnFix:=0;
```

```
ClearB;R:=500;
```

```
for i:=1 to 11 do ErrDis[i]:=0;
```

```
readln(INDATA,title);writeln(OUTDATA,title);writeln(OUTDATA);  
writeln('Bundle length, please ');readln(h);
```

```
while not(eof(INDATA)) do
```

```
begin
```

```
readln(INDATA,x);
```

```
if x>0 then
```

```
begin
```

```
TotalPackets:=TotalPackets + x;
```

```
x:=x+leftovers;leftovers:=0;
```

```
if x >= (h-CBunLen) then
```

```

begin
x := x-h+CBunLen;
CorBundle := CorBundle + (x div h);
Leftovers := x mod h;
if CBadPackets = 0 then
begin
CorBundle := CorBundle + 1;
CBunLen := 0
end
else ErroneousBundle
end
else CBunLen := CBunLen + x
end;

if x=-1 then
begin
TotalPackets:=TotalPackets + 1;
BadPackets:= BadPackets + 1;
CBadPackets:= CBadPackets + 1;
CBunLen := CBunLen +1;

readln(INDATA,y);U:=0;
for i:=1 to y do
begin
readln(INDATA,n,e);W:=weight(e);
BitErrors:=BitErrors + W;
BICBadPackets,n] := W;
U := U + W
end;

if U<11 then ErrDis[U] := ErrDis[U] + 1
else ErrDis[11]:= ErrDis[11]+1;
if CBunLen = h then ErroneousBundle

end;

if x=-2 then
begin
TotalPackets := TotalPackets + 1;
BadPackets := BadPackets +1;
Missing := Missing + 1;
CBunLen := CBunLen + 1;
CBadPackets := CBadPackets + 1;

for i:=1 to 28 do BICBadPackets,i] := 7;

if CBunLen = h then ErroneousBundle
end;

end;[ data reading ]

{Parameter Calculations}

TotalBundles := CorBundle + Fix + UnFix;

writeln(OUTDATA,trunc(BadPackets),' of ',trunc(TotalPackets),

```

```

' packets were in error or missing. Rate=',
BadPackets/TotalPackets);
writeln(OUTDATA);writeln(OUTDATA,trunc(Missing),' of these were mi
ssing');
writeln(OUTDATA);

writeln(OUTDATA,'The Bit Error Rate was ',BitError/(264*TotalPacke
ts));
writeln(OUTDATA,'i.e. ',trunc(BitError),
' bit errors in ',trunc(264*TotalPackets),' bits.'
);

writeln(OUTDATA,'-----');
writeln(OUTDATA,' i # of Packets with i errors');
writeln(OUTDATA,'-----');
for i:= 1 to 11 do writeln(OUTDATA,' ',i:2,' ',ErrDis[i
]:5);
writeln(OUTDATA,'-----');

writeln(OUTDATA,'BUNDLE REPORT: length= ',h,' packets');
writeln(OUTDATA);writeln(OUTDATA,trunc(TotalBundles));
writeln(OUTDATA,trunc(CorBundle),' correct bundles,');
writeln(OUTDATA,trunc(Fix+CorBundle),' were correctable and');
writeln(OUTDATA,trunc(UnFix),
' were uncorrectable. Rate ',UnFix/TotalBundle
s)

end.{PROGRAM!}

```

(b)

```

PROGRAM DATA CRUNCH;
{FIELD DATA RENDERED DOWN TO SIZE}

var INFILE,OUTFILE,TITLE,S:string;
    i,a,b,j,x,max,count,U,PAGE,LINE:integer;
    w:char;
    INDATA:text;
    OUTDATA:text;

function trans(t:string):integer;
var s,i:integer;

begin
    s:=0;
    for i:=1 to 3 do s:=8*s+ord(t[i])-ord('0');
    trans:=s
end;

PROCEDURE finish;
begin
    writeln(OUTDATA,count);
    close(INDATA);
    close(OUTDATA,lock);
    EXIT(PROGRAM)
end;

PROCEDURE GetLine;

begin
    if eof(INDATA) then finish;
    readln(INDATA,s);u:=12;
    while length(s)<55 do
        begin
            writeln(s,' ---->',count);
            if eof(INDATA) then finish;
            readln(INDATA,s);
            u:=33
        end;
    end;

FUNCTION Next:integer;

begin
    if eof(INDATA) then finish;
    if U<61 then U:=U+7 else GetLine;
    Next:=trans(copy(s,U,3))
end;

{THE MAIN PROGRAM <-----*****}

begin
    write('Data File Name: ');readln(INFILE);
    reset(indata,infile);

```

```

writeln;writeln;write('Results to File: ');readln(OUTFILE);
rewrite(OUTDATA,OUTFILE);
readln(TITLE);writeln(OUTDATA,TITLE);

for i:= 1 to 7 do readln(INDATA,s);
U:=26;
MAX:=27720;
COUNT:=0;
READLN(INDATA,S);

while eof(INDATA)=false do
begin

X:=next;

if x=0 then
begin
count:=count+1;
if count=max then
begin
writeln(OUTDATA,count);
writeln('--> ',count);
count:=0
end;
end
else if x<33 then
begin
writeln(OUTDATA,count);count:=0;
writeln(OUTDATA,-1);writeln(OUTDATA,x);
write('--> ',-1,' ',x);
for i:=1 to x do
begin
a:=next;b:=trans(copy(s,U-3,3));
write(' == ',b,' ',a);
writeln(OUTDATA,b,' ',a);
end;
writeln;
end
else if x=255 then repeat getline until (copy(s,1,6)='000000')
else writeln(OUTDATA,-2)
end;

finish

end.

```

(C) PROGRAM QUICKPASS;

{Quick run through the data}

```

var title,infile,outfile:string;
    x,y,i,pack,badpack:integer;
    OUTDATA,INDATA:text;
    flag:string;

begin
write('Input file name please: ');readln(infile);
reset(INDATA,infile);
write('Hardcopy? <y/n>');readln(flag);
if (flag='y') or (flag='Y') then outfile:='printer:' else outfile:
='console: ';
rewrite(OUTDATA,outfile);

readln(INDATA,title);writeln(OUTDATA,title);writeln;
pack:=0;badpack:=0;

while not(eof(INDATA)) do
begin
readln(INDATA,x);
if x>0 then pack:=pack + x;
if x<0 then
begin
pack:=pack+1;
badpack:=badpack + 1;
if x=-1 then
begin
readln(INDATA,y);
for i:=1 to y do readln(INDATA)
end;
end;
if (pack mod 1000)=0 then write(OUTDATA,pack,' ',badpack)
end;

write(OUTDATA,pack,' ',badpack)
end.

```

Chapter 5. Support for the Patent Applications

5.1 Activities

During the course of this contract, the Department of Communications decided to attempt to patent 1) the Carleton Code and 2) the Bundle Code. After acceptance by the Canadian Patent Development Limited these files were passed to the firm of Barragar and Oyen for preparation of the patent applications.

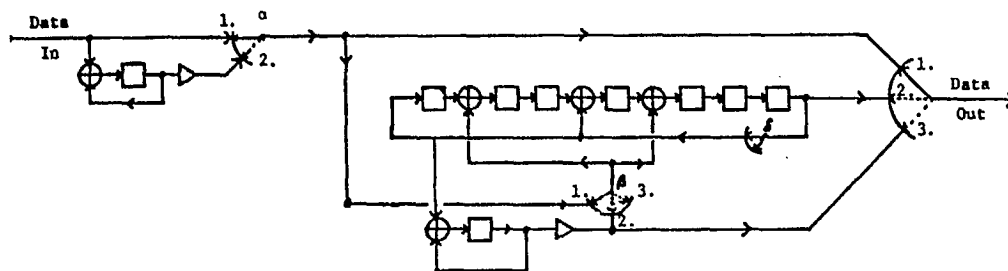
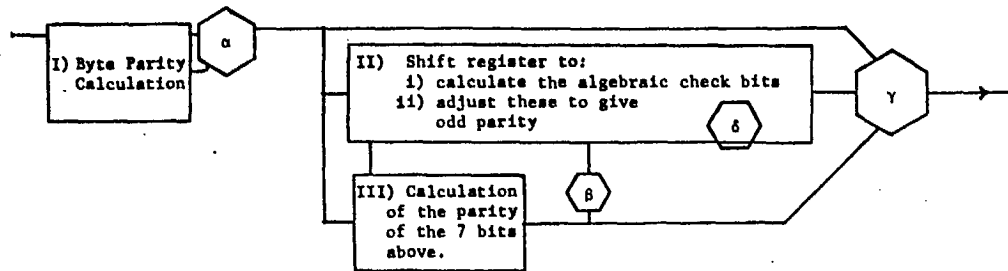
During this year we have undertaken the following activities in support of the patent applications:

- (i) participated in two meetings with John Morissey and Lyn Cassan of Barragar and Oyen to discuss and explain the coding systems,
- (ii) held numerous conversations with Lyn Cassan as she prepared the patent applications,
- (iii) provided an annotated diary of all occasions when we have presented any of our results over the preceding contracts to an audience, and supplied copies of all materials used in such lectures,
- (iv) read and commented on various versions of the patent applications and,
- (v) designed a hardware encoder for Carleton Code.

5.2 A hardware Encoder for Carleton Code.

The Carleton Code, which is in a sense one of the constituents of Code C, was defined in [14] and [15]. A hardware encoder for this code was designed to support the attempt to patent application for Carleton Code. We present this encoder below.

CARLETON CODE ENCODER



Clocking and reset circuits are not included in the diagram.

CARLETON CODE ENCODER

Switch Specifications

α :	8 bit cycle ,	at 1 for 7 bits, at 2 for 1 bit.
β :	224 bit cycle ,	at 1 for 27 bytes, at 2 for bit 2 of byte 28, at 3 for bits 1, 3, 4, 5, 6, 7, 8 of byte 28.
γ :	224 bit cycle ,	at 1 for bytes 1 to 27, at 2 for bits 2, ... , 8 of byte 28, at 3 for bit 1 of byte 28.
δ :	224 bit cycle ,	closed for 27 bytes open for byte 28

Operations

- I) This circuit calculates the cumulative parity of the incoming data bits. After 7 bits, switch α changes and the inverted contents of the parity flip-flop pass down the channel. Not shown is a circuit to reset the flip-flop to zero at this point.
- II) This sequence of mod 2 adders (EOR gates) and flip-flops perform two functions. For the first 27 bytes it calculates the sum

$$\sum_{i=8}^{224} c_i \alpha^i$$

expressing the value as the seven coefficients of a sum

$$\sum_{j=0}^6 d_j \alpha^j .$$

The circuit III then contains the parity of these 7 bits. If this parity is even then we must add (1 0 0 1 0 0 0 1) to the byte (d_0 d_1 ... d_6 0) . If the parity is odd we send (d_0 d_1 ... d_6 0) unaltered.

For the 28th byte, the circuit does not cycle on bit 1 but the inverted parity from circuit III is sent out as d_7 . Hence switch γ is in position 3. On bit 2, the register resumes cycling and the inverted parity is added to d_0 and d_3 (i.e. switch β is at position 2) as these shift up the register. This shift pushes d_6 down the channel. The shift register then continues to shift d_5 , d_4 ... d_0 out of the register with no feedback (switch δ open) and no additions (switch β at position 3).

E.g. Each data bit arrives as if it were in the α^8 position. So since $\alpha^8 = \alpha + \alpha^4$ we add the bit to stages 2 and 5. The feedback then multiplies this bit by α once for each successive bit which is input.

Suppose 0 1 1 1 0 0 0 0 \rightarrow is fed in. We will have

<u>input</u>	<u>register</u>	
	0 0 0 0 0 0 0	
0	0 0 0 0 0 0 0	
0	0 0 0 0 0 0 0	
0	0 0 0 0 0 0 0	
0	0 0 0 0 0 0 0	
0	0 0 0 0 0 0 0	
1	0 1 0 0 1 0 0	
1	0 1 1 0 1 1 0	
1	0 1 1 1 1 1 1	
0	1 0 1 0 1 1 1	$= \alpha^9 + \alpha^{10} + \alpha^{11}$

- III) Circuit III calculates the parity of the seven bits in the register II above. This parity changes whenever a one is fed back from the end to the beginning of register II. Incoming data bits do not change the parity since they are added to two bits of II.

REFERENCES

- [1] North American Broadcast Teletext Specification - Draft, June, 1983.
- [2] B.C. Mortimer and M.J. Moore, "Two-byte Data Block and Bundle Codes for the Broadcast Telidon System", Progress Report, Department of Communications, DSS Contract No. OSU82-00164, November, 1982.
- [3] Brian Mortimer and Michael Moore, "More Powerful Error-Correction Scheme ofr the Broadcast Telidon System", Final Report, Department of Communications, DSS Contract No. OSU82-00164, March, 1983.
- [4] P. Allard, V.K. Bhargava and G.E. Seguin, "Realization, Economic and Performance Analysis of Error-correcting Codes and ARQ Systems for Broadcast Telidon and other Videotex Transmission", Department of Communications, Ottawa, DSS Contract No. OSU80-00133, Final Report, June, 1981.
- [5] G.E. Seguin, P. Allard and V.K. Bhargava, "A Class of High Rate Codes for Byte-oriented Information Systems", I.E.E.E. Trans. Comm. COM-31, 1983.
- [6] K. Hari, G.E. Seguin, V.K. Bhargava and P.E. Allard, "Further Results on High Rate Codes for Byte-oriented Information Systems", Research Report, Department of Electrical Engineering, Concordia University, Montreal, Quebec, February, 1983.
- [7] M. Sablatash and J.R. Storey, "Determination of Through puts, Efficiencies and Optimal Block Lengths for an Error Correcting Scheme for the Canadian Broadcast Telidon System", Can. Elec. Eng. J., 5 (1979), pp. 25-39.
- [8] "Simmulation of Error Correction for Japanese Teletext", CCIR (1982-1986), Study Groups Document 11/29-E (Japan), 6 May, 1983.
- [9] "Error Correction for Japanese Teletext", CCIR (1982-1986), Study Groups Document 11/129-E (Japan), 29 August, 1983.
- [10] George C. Clark and J. Bib Cain, Error-Correction Coding for Digital Communications, Plenum, New York, 1981.

- [11] "Japanese Teletext Utilizing an Alpha - DRCS - Photographic Coding Scheme", CCIR (1982-1986), Study Groups Document 11/27 (Japan), 2 May, 1983.
- [12] B.C. Mortimer, "A Correction to a Recent Analysis of a Product Code", Can. Elec. J., 7 (1982) 40.
- [13] Jack K. Wolf, Arnold M. Michelson and Allen H. Levesque, "On the Probability of Undetected Error for Linear Block Codes", I.E.E.E. Trans. Comm. Vol. COM-30, 1980, pp. 317-324.
- [14] B. Leroux, M. Moore, B.C. Mortimer, L. Oattes and T. Ritchford, "A Study of the Use of Error Correcting Codes in the Canadian Broadcast Telidon System", Department of Communications, Ottawa, Progress Report, DSS Contract No. OSU81-00095, August, 1981.
- [15] Brian Mortimer, "A Description of the Carleton Code", DSS Contract No. OSU81-00095, Progress Report, Department of Communications, Canada, September, 1981.

