MICROPROCESSOR-CONTROLLED PERMUTATION DECODING OF BLOCK ERROR-CORRECTING CODES.

R.M.F. Goodman* B.Sc., Ph.D.

A.D. Green* B.Sc.

Summary

Microprocessors can be used to simplify the hardware required to build the decoder of a block error-control system, but only at the expense of the low data throughput rates characteristic of software implementation. This paper proposes a new soft-decision permutation decoding algorithm for cyclic block codes, and deals with the implementation of both hard and soft-decision permutation decoding on an Intel 8080A microprocessor system. In particular, coding gain and data throughput trade-offs are investigated for both total software and software plus specialised hardware implementations, under Gaussian noise conditions. Finally, possible system developments for achieving higher throughput speeds are discussed.

1. Introduction

Block error-correcting codes are used to provide effective error-control in many digital communications systems. In general, the main problem in implementing such a scheme is that of the decoder hardware complexity required to efficiently decode a particular code, and thus realise a certain degree of error-control or coding gain. In particular, the use of demodulator 'confidence' information in soft-decision decoding, can raise the hardware complexity to prohibitive levels. The advent of microprocessors, however, has greatly simplified the implementation of both hard and soft-decision decoding schemes in terms of providing both powerful and flexible bit-handling and computation facilities with a minimum hardware package count. Unfortunately, the penalty to be paid by transferring complexity from hardware to software is that of low data throughput rates, and in the case of a real time decoding system this raises the question of buffer overflow. These problems may be overcome to some extent by the use of specialised hardware modules which operate under microprocessor control, and perform some of the decoding operations which would otherwise be particularly time consuming if executed in software.

In this paper we investigate the coding and data-throughput performance for a variety of mixed hardware/software decoder implementations, all based on an Intel 8080A microprocessor system. We restrict our attention to two codes, the (23,12) triple-error correcting Golay code, and the (23,11) expurgated Golay code; although the proposed soft-decision algorithm can be used on more powerful codes provided they are cyclic.

In the case of hard-decision decoding, the demodulator only outputs a 'hard' 0/1 decision so that 23 bits per block are transferred into the microprocessor and therefore relatively fast operation is possible. The coding gain obtainable is approximately 2dB at a user output bit error rate of 10^{-5}. In the case of soft-decision decoding, the demodulator assigns a 'confidence' value to each output digit in addition to the 'hard' 0/1 decision. In practice this involves replacing the one bit

* Department of Electronic Engineering, University of Hull

hard-decision device at the demodulator output with a J bit A to D converter. Each incoming data bit is therefore effectively quantised to $Q = 2^J$ levels, and the microprocessor must now handle 23 J-bit bytes per block. This, together with the fact that the soft-decision algorithm is more complex and time consuming than the hard-decision algorithm, means that operation is inevitably slower than hard-decision. The advantage of soft-decision however is that coding gains of approximately 3.8dB are achievable, thus effectively doubling the power of the codes. The actual increase in coding gain (over that achievable with hard-decision) that can be expected from soft-decision depends on both the number and spacing of the quantisation levels, and on the channel characteristics. It has, however, been shown (ref. 1) that for the additive white Gaussian channel with optimum-spacing infinite-level quantisation the maximum coding gain is lower-bounded by about 2dB. Fortunately, the degradation involved in using the much more practical equal-spacing 8-level quantisation is only about 0.25 (ref. 2), and for an 8-bit microprocessor either 8 or 16 level quantisation is suitable. In addition, it has been shown (ref. 3) that for both the Gaussian and Rayleigh-fading channels, the performance of a soft-decision decoder approaches that of the optimum maximum-likelihood decoder. This implies a 3dB improvement over hard-decision for the Gaussian channel, at high signal-to-noise ratios.

This paper develops in the following way. Firstly, the basic concepts of hard and soft-decision decoding are outlined. Next, permutation decoding is introduced, and the hard and soft-decision algorithms are developed from this. Finally, the microprocessor implementation of the algorithm is described, and system performance results under Gaussian noise conditions are presented and discussed.

2. Hard and Soft-Decision Decoding

Given an (n,k) t-error correcting block code the optimum method of decoding is maximum-likelihood decoding, which for the hard-decision binary symmetric channel is equivalent to minimum-distance decoding. A minimum-distance decoder attempts to find the codeword nearest in terms of Hamming distance to the received block v(x). That is, the codeword which satisfies

$$\min \left\{ \sum_{i=1}^{n} (v_i(x) \oplus u_i(x)) \right\} = \min \left\{ \sum_{i=1}^{n} e_i(x) \right\}.$$

Alternatively, this is equivalent to finding the minimum weight error pattern e(x) which will turn the received block v(x) into a valid codeword u(x).

In the case of soft-decision decoding each digit in the block is quantised to Q levels, so that the demodulator estimate of a particular digit can be expressed as a J bit byte $[v_i(x)]$, where the square brackets are used to denote a soft-decision quantity. Therefore for 8-level quantisation $[v_i(x)]$ can be [000], [001],..., [110], [111]. In a similar way we may form a soft-decision estimate of the error digit $[e_i(x)]$ which has been added (modulo-2) to a given received digit $[v_i(x)]$ by the channel. Thus, given that $[u_i(x)]$ (= [000] or [111]) was transmitted, and $[v_i(x)]$ was received, then $[e_i(x)] = [v_i(x)] \oplus [u_i(x)]$. The value of the soft-decision error digit in <u>levels</u> can therefore lie between 0 and (Q-1), and a value of $\geq(Q/2)$ constitutes an 'error' in the hard decision sense.

We may now define a soft-decision minimum distance decoder as a decoder that attempts to find the codeword at minimum <u>soft-decision</u> distance (minimum number of level errors) from the received block v(x). That is,

$$\min \{ \sum_{i=1}^{n} ([v_i(x)]\oplus[u_i(x)])\} = \min \{ \sum_{i=1}^{n} ([e_i(x)])\}.$$

We may now estimate the error-correcting capability of an (n,k) block code in the soft-decision sense. If the hard minimum distance of the code is d_h then its bounded-distance hard correcting power is the largest integer $t_h \leq \{(d_h-1)/2\}$. In the soft-decision sense codewords are $\geq d_s = (Q-1)d_h$ soft-decision levels apart, and therefore the bounded-distance guaranteed soft-decision error-correction power in levels is the largest integer $t_s \leq \{(d_s-1)/2\}$. In order to see the improvement obtained from soft-decision decoding, consider the (23,12) $d_h = 7$ Golay code. In this case $t_h = \{(7-1)/2\} = 3$ errors per block. For soft-decision, $d_s = (8-1) \times 7 = 49$ levels, and therefore correction can be guaranteed for $t_s = \{(49-1)/2\} = 24$ level errors per block. The smallest number of level errors that constitutes an error in the 'hard' sense is $Q/2 = 4$; the most probable pattern of 6 hard-decision errors therefore has a soft-decision weight of $6 \times 4 = 24$ levels, which is within the soft-decision bounded-distance of the code, and can therefore be corrected. Asymptotically, at high signal-to-noise ratios, soft-decision decoding therefore effectively doubles the 'hard' correcting power of a code.

In practice, the algorithms used in this paper attempt to choose the minimum weight (hard or soft) error pattern from amongst a number of alternatives. In this way the predicted coding gains can be achieved but only at the expense of computation time, and therefore low data throughput rates.

3. Permutation Decoding

Given a t-error correcting (n,k) cyclic code with generator polynomial g(x), the syndrome s(x) of a received block v(x) can be written s(x) = remainder {v(x)/g(x)}; and depends only on the error pattern e(x), and not on the actual codeword u(x) transmitted. We may thus write e(x) = q(x)g(x) + s(x). If the errors in e(x) are all confined to the n-k parity check positions of v(x), that is $x^{n-k-1}, \ldots, x, 1$, then q(x)=0 and e(x) = s(x). That is, the error pattern is identical to the syndrome. If the errors are not confined to the n-k parity check positions of v(x), then it may be possible to move the errors into the parity check positions by rearranging bit positions within the block according to some code preserving permutation (ref. 4). The errors then appear in the parity check section of some other block v'(x) which is a permutation of the original received block v(x). The syndrome s'(x) is then identical to the error pattern e'(x), which is a permutation of the original error pattern e(x). Once the error pattern is known it can be reverse-permed and modulo-2 added to the data bits of received block to give the corrected data. A hard-decision permutation decoder for the Golay code would therefore operate by sucessively applying code preserving permutations to the received block, calculating the corresponding syndrome, and testing the syndrome for a weight of $\leq t_h = 3$. Once the syndrome weight is 3 or less the error pattern is assumed to be 'trapped', and reverse-permutation gives the original error pattern. In the case of the (23,11) $d_h = 8$ expurgated Golay code, the syndrome weight may never go down to ≤ 3, in which case the decoder chooses the minimum weight syndrome encountered after exhausting all possible permutations.

It is important that the permutations used are code-preserving permutations. This ensures that the original code and the permed code have the same generator polynomial g(x), and therefore the same syndrome calculation hardware can be used to decode both the original block and the permed block.

There are two main code-preserving permutations which are applicable to all cyclic codes, and which are used in both the hard and soft-decision decoding algorithms. These are the cyclic permutation, and the binary permutation. The cyclic permutation states that any end-around cyclic shift of a codeword results in another codeword. That is, the bits in the block are permed according to: $x^i \rightarrow x^i + 1$ modulo x^n, where i indicates the ith bit in the block. The binary permutation ensures that the code is preserved if:

$$x^i \rightarrow x^{i.2^j} \text{ modulo } x^n, \text{ for } 0 \leq j \leq p \qquad \ldots\ldots.(1)$$

The number of distinct perms (neglecting no permutation) is p, where p is such that $n.c = 2^{p+1}-1$, c being the smallest possible integer. In the case of the (23,12) Golay code there are a total of 10 distinct binary permutations, and 23 cyclic permutations for each binary permutation.

4. The Hard and Soft-Decision Permutation Decoding Algorithms

The hard decision algorithm is as follows:

1. The syndrome s(x) of v(x) is calculated. This can either be done by software or hardware. Figure 4 shows a general hardware syndrome generator for the (23,12) Golay code. The syndrome is loaded either serially (n-k shifts) or in parallel. Once the generator is loaded with the first n-k bits of v(x), feedback is allowed, and after k further shifts the syndrome is in the register. If the weight of s(x) is ≤ 3 the error pattern is found and can be modulo-2 added to v(x). If the weight is zero, no errors are assumed to have occurred, and v(x) is correct.
2. If the weight of s(x) > 3 a cyclic permutation is tried. The syndrome s'(x) of an i-place cyclic shift of v(x) is easily computed by simply shifting the syndrome generator, with feedback applied, i times. Thus the syndrome generator is shifted up to 23 times, and a check on syndrome weight is made after each shift. If the weight of s'(x) is 3 or less, the error pattern is found, and can be added to v(x) after cyclic reverse-shifting.
3. If after 23 shifts the weight of s'(x) has not gone down to 3 or less, a binary perm is tried. In this case v(x) is permed according to equation 1, and applied to the syndrome generator. If the weight of the permed syndrome is 3 or less the error pattern is found. If not, then cyclic perms are applied by simply executing further shifts of the syndrome generator as in step 2 above. Once the weight of the syndrome goes down to 3 or less the error pattern is found, and must be cyclicly reverse-permed and then binary reverse-permed before being modulo-2 added to v(x).
4. Operation continues in a like manner with binary perms followed by n cyclic perms, until all distinct binary perms are exhausted. In the case of the (23,12) Golay code all error patterns of 3 or less can be 'trapped' by using only 3 of the 10 binary perms (plus no perm). For the (23,11) code only 2 binary perms are needed. In this latter case it is possible that the syndrome weight never goes down to 3. This indicates that an error pattern of weight > 3 has been detected, and the decoder can either refuse to decode or choose the smallest weight syndrome encountered.

The soft-decision algorithm proceeds in a manner similar to the hard-decision algorithm in that the sequence of syndrome computation, cyclic perms, binary plus cyclic perms, is the same. However, in the hard-decision case we simply searched for a syndrome weight ≤ 3; in the soft-decision case a more complex (and therefore more time-consuming) test has to be applied after each syndrome computation. In essence, we are searching for an error-pattern whose soft-decision weight is $< t_s$, if one cannot be found we choose the smallest weight encountered in the whole decoding process. Assuming that the tentative error pattern at some stage in the decoding process is $e'(x) = [000\ldots0_k s'(x)]$, the soft-decision weight of that error pattern in levels is given by:

$$\sum_{i=1}^{n} \{ \ [v_i'(x)] \oplus [h_i'(x)] \oplus [e_i'(x)] \ \} \qquad \ldots\ldots.(2)$$

where $[h_i'(x)]$ is the hard decision estimate of $[v_i'(x)]$, and equals [000] or [111] as $v'(x) = 0$ or 1. The speed with which this computation can be performed has a great effect on the overall speed of the algorithm, and although equation 2 implied that n additions of J bit soft-decision numbers are required, significantly fewer are in fact needed (ref. 5). Also, other techniques can be used to omit unnecessary soft weight calculations.

The main factor which makes the soft-decision algorithm slower than the hard-decision algorithm is that more binary perms must be performed before a decoding decision can be made. This is because that with soft-decision, error patterns of weight up to 6 are correctable, and all 10 binary perms are needed if the majority of patterns of weight ≤ 6 are to be 'trappable'. It is therefore important that a particular error pattern is 'trapped' in as few binary perms as possible. We have found that if the binary perms are used consecutively, that is, $j = 0, 1, 2, 3\ldots10$, this is not the case. In fact, the optimum perm order is $j = 0, 4, 1, 8, 5, 2, 7, 9, 3, 6, 10$ for the (23,11) code, and $j = 0, 1, 2, 4, 3, 6, 10, 8, 9, 7, 5$ for the (23,12) code, and not perming in the optimum order has a significant effect on the average effort required to decode a block, and hence on the average data throughput rate.

It is possible to restrict the number of binary perms attempted in order to increase the decoding speed. However, as the number of perms tried is decreased from 10, more and more of the higher weight error patterns become 'untrappable' and thus the coding performance is degraded. The degradation is, however, gradual as the highest weight error patterns are also the least probable, and therefore they do not affect the output error rate as much as the lower weight patterns. For example, for the (23,11) code, if all 10 perms are used then all patterns of weight 5 and less are trappable, and 82% of weight 6 patterns are trappable. If only 5 perms are used then all patterns of weight 3 or less are trappable, with 99% of weight 4, 85% of weight 5, and 65% of weight 6, also being trappable. In this way a trade-off of coding gain versus data throughput speed can be established.

5. Microprocessor Implementation

5.1 Hardware Structure

The development decoding system is based on an Intel 8080A microprocessor and uses the standard SDK-80 system development kit, with additional RAM and two specialised hardware modules which perform syndrome calculation and hard-decision bit permutation. The development system is shown in figure 1, and as well as executing the actual decoding schemes, allows for the input and editing of programs, and the output of performance statistics.

The syndrome generator is shown in figure 2, and is of the type shown in figure 4, with the added refinement that syndrome weights are calculated.

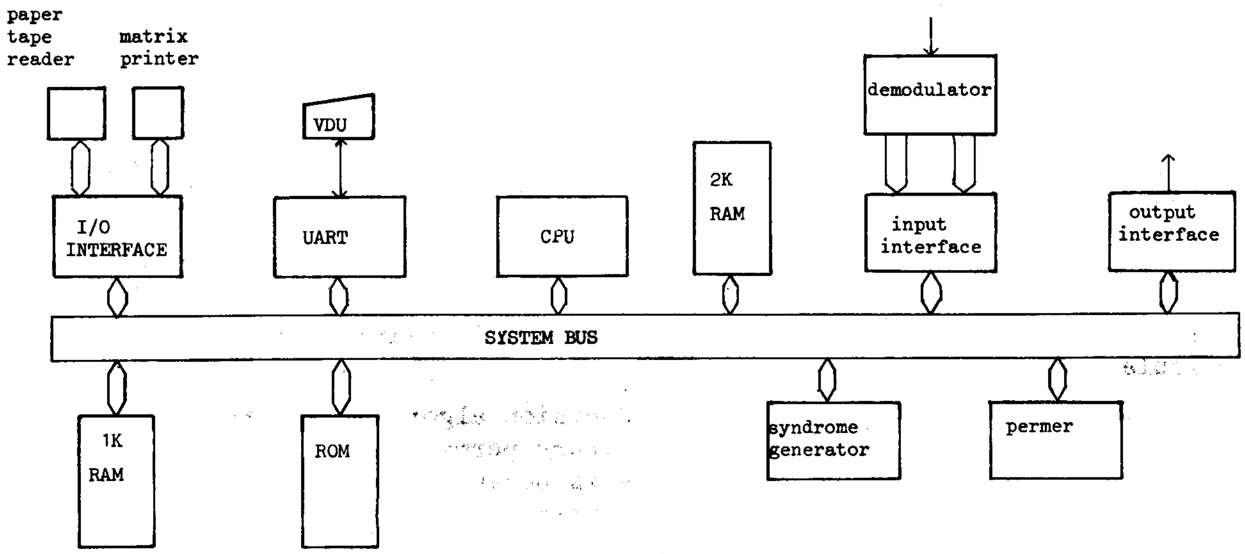

Fig.1. DEVELOPMENT SYSTEM

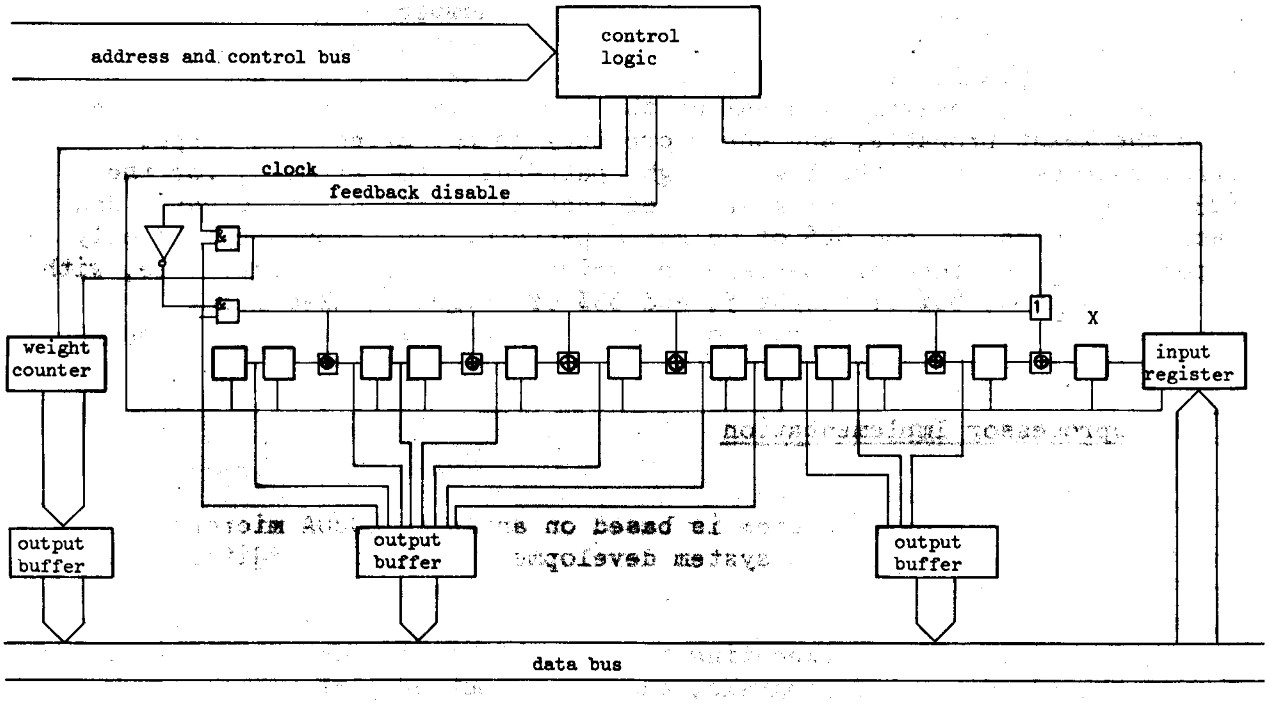

Fig.2. SYNDROME GENERATOR MODULE

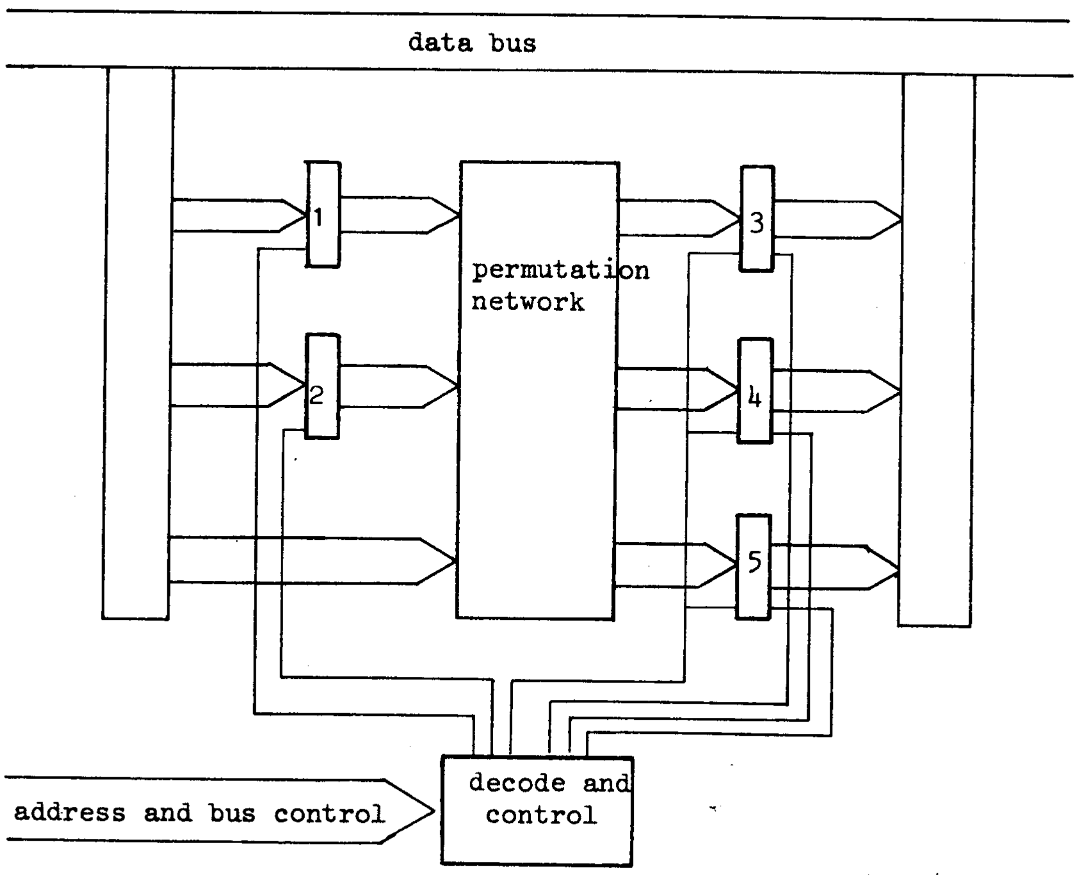

Fig.3. PERMUTATION MODULE

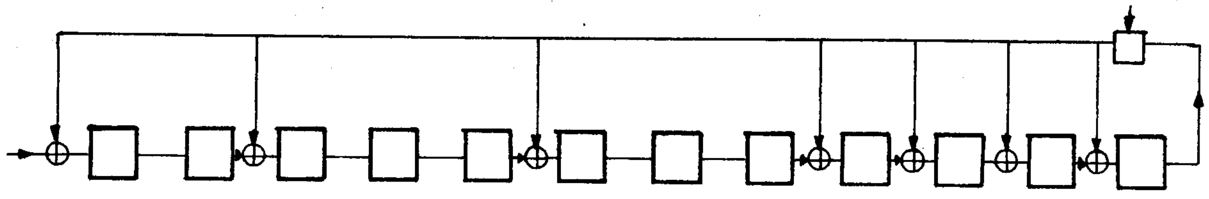

Fig.4. SYNDROME GENERATOR (23,11)

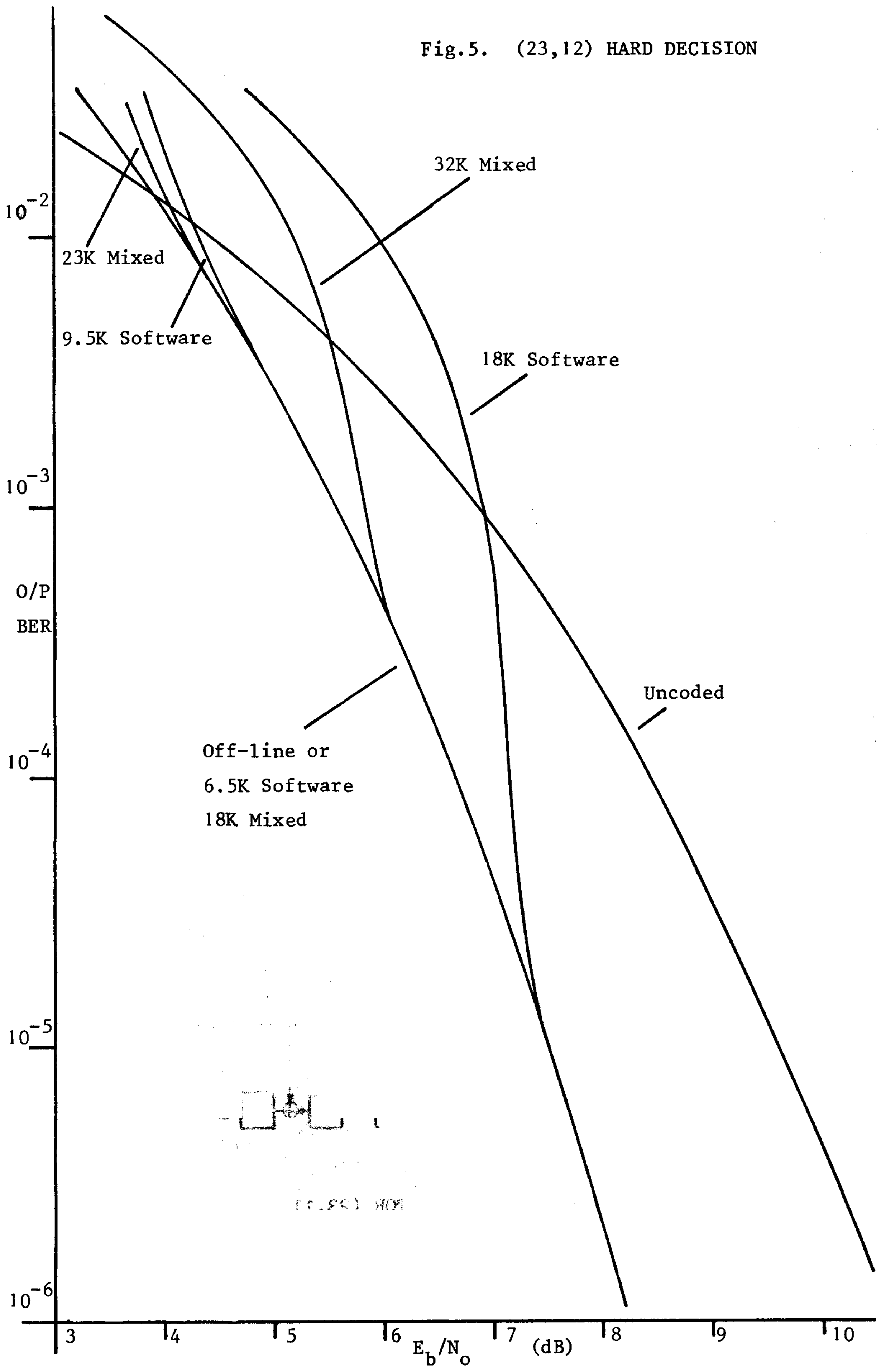

Fig.5. (23,12) HARD DECISION
32K Mixed
23K Mixed
9.5K Software
18K Software
Uncoded
Off-line or
6.5K Software
18K Mixed
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
O/P
BER
3
4
5
6
7
8
9
10
E_b/N_o (dB)

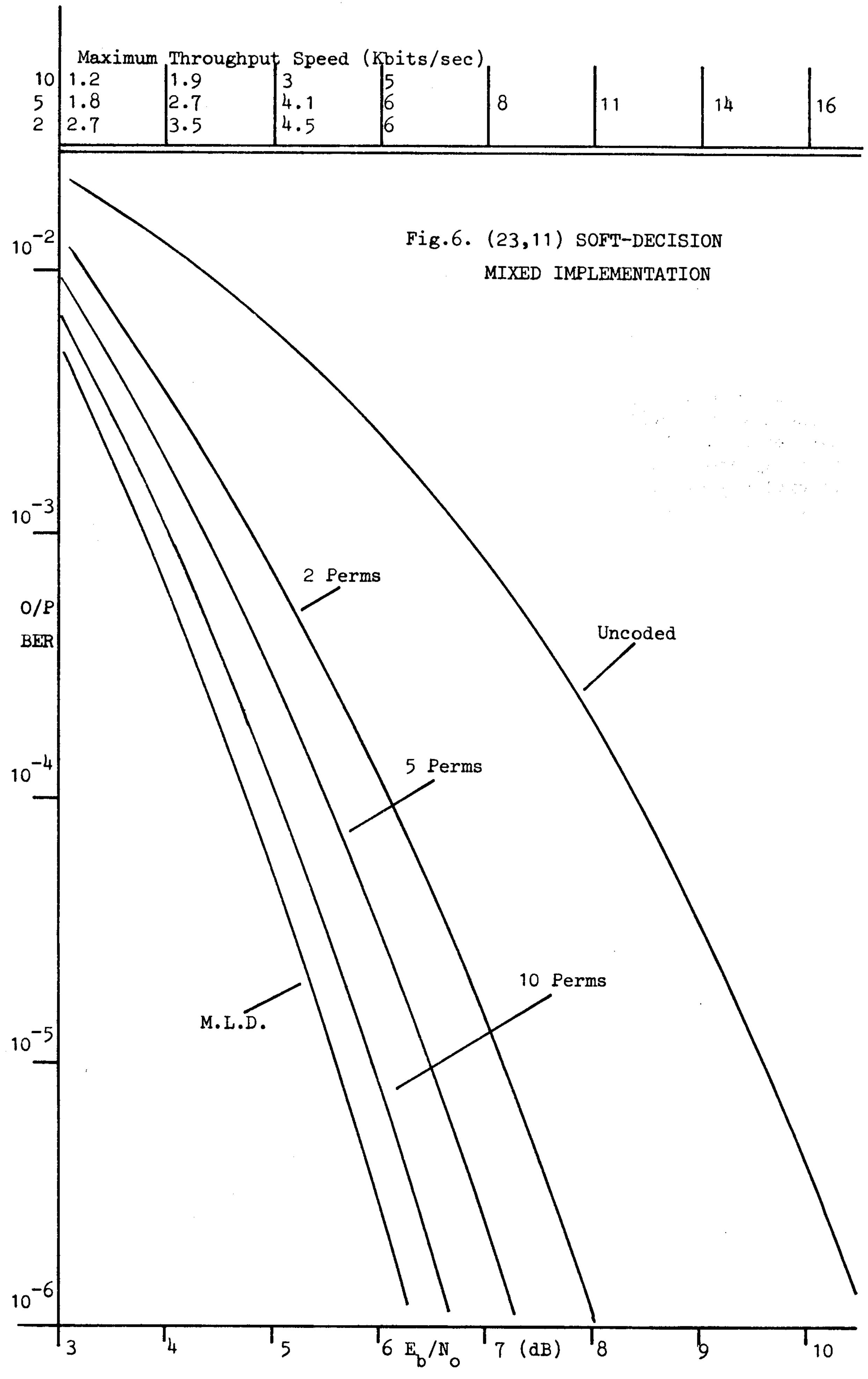

Fig.6. (23,11) SOFT-DECISION MIXED IMPLEMENTATION

The syndrome generator has its input/output organised as locations in memory, thus enabling the processor to read or write directly to it. Operation of the generator under program control is as follows. The syndrome generator is first loaded with the 23 bit received block by writing 3 bytes of 8, 8, and 7 bits to the 8-bit input register. After each of the bytes has been loaded, 8 clock pulses are applied to the registers, thus shifting the information into the Galois Field (2) divider. The bistable X ensures that the eighth (unused) bit of the third byte is not shifted into the divider. The syndrome is now available in the divider register. The feedback on the divider is then disabled, and a further 11 bit cyclic shift of the divider register is made, to accumulate the weight of the register's contents into the weight counter, and replace the syndrome in the register. Syndrome and syndrome weight outputs are then directly available onto the data bus by reading the divider and weight registers, which are memory-mapped. The syndrome generator also has the ability to compute the syndrome of a single place cyclic shift of the received block, by performing a single place shift of the current contents of the divider register, with feedback enabled. To do this the divider register is clocked by the processor write signal when the control address of the syndrome generator is selected. Again, this is followed by an 11 bit cyclic shift to calculate the weight of the new syndrome. The syndrome generator is clocked by the processor clock at approximately 2MHz. This ensures that the actual time taken to calculate syndrome and syndrome weight is equivalent to a few machine instruction times, which means that as far as the software is concerned these are available 'instantly'.

Bit permutation of the hard-decision received blocks is performed by the circuit shown in figure 3, which consists of five 8-bit latches, the three output latches being tri-state. The operation of the bit permer is as follows. The block requiring permutation is fed to the permer in three bytes, the first to latch 1, the second to latch 2. As the third byte is written to the permer, latches 3, 4, and 5 are enabled and the current information on the data bus, as well as the outputs from latches 1 and 2, are loaded into these, via the permutation network. As the outputs of latches 3, 4, and 5 are read by the processor, the information on the data bus is latched into the corresponding input latch. In this way the action of reading the output of the permer automatically forms the next perm in the latch outputs, thus making the execution of multiple permutations a simple matter of repeated read commands. Note, however, that the permutations are executed in numerical order, so that extra software control is required to run through perms in the optimum order.

The input interface accepts data from the demodulator in the form of a parallel 4-bit quantised estimate of each channel digit, which can therefore take values between 0000 and 1111. The interface separates the 4-bit estimate into a hard-decision estimate, and three bits of soft-decision information. The hard-decision bits are clocked into an 8 bit register, to form a byte of hard-decision information, while the soft-decision information is treated as a 'nibble' of 4-bits, and therefore packed two to a byte. Thus for every 8 channel bits there are 5 bytes of data to be input when using the soft-decision algorithm, or one byte when using hard decision.

5.2 Software Structure

The input of data from the input interface takes place under interrupt control, there being 4 interrupts required per 8 channel digits, to input the 5 bytes of information. For hard-decision decoding, only one interrupt per 8 channel digits is required.

The storage area allocated to the input data depends on the amount of RAM available, which can be easily expanded to 64K, and is used in a cyclic order to provide an automatic buffer capability. In the case of hard-decision the location of the buffer input and output is tracked by two pointers which are incremented modulo the size of the buffer area. In the case of soft-decision the hard and soft data are organised in separate streams in such a way that the address of both sets of information bear a constant relation to each other, so that again only two pointers are needed.

Blocks are then processed according to the algorithms described in section 4. In addition as processing carries on a check is kept on the state of the buffer. Also, input interrupts occur which must as a matter of priority be serviced. Corrected message bits are then output as they become available, although this could proceed in synchronism with input interrupts if output buffering is used.

The storage requirements for implementing the algorithms described are between one-half and 1K of ROM for program, and 256 bytes of RAM for hard-decision buffer storage, and program variables. Soft-decision requires a correspondingly larger buffer area. As far as programming is concerned, the need for high speed processing has led to the programs being written very much 'in-line'. That is, subrouting and looping have been removed as much as possible from the time-critical sections of the programs, in order to keep overheads to a minimum.

6. Performance Results

The performance of the hard and soft-decision permutation algorithms has been investigated on the microprocessor system under conditions of Gaussian noise. Binary antipodal signalling with equal-spacing 16-level quantisation and matched filter detection is assumed. The bit probability of error for the Gaussian channel is given by the Q-function as: $p = Q\ (\sqrt{2E_b/N_oR})$, where N_o is the single-sided noise power density, E_b is the energy per information bit, and R is the code rate. Note that all performance curves are corrected for rate, that is, plotted versus E_b/N_o, to ensure a valid comparison between coded and uncoded transmission. Figure 5 shows the performance of the (23,12) Golay code using hard-decision decoding with both software only, and mixed software/hardware implementations. Under offline decoding conditions, it can be seen that the full hard-decision performance of the code is obtained. When a buffer limit of 32 blocks is introduced this same performance can be obtained at data throughput rates of 6.5 bits/sec in the case of a software only implementation, and 18k bits/sec in the case of a mixed software/hardware-modules implementation. The effect of increasing data throughput, which is to degrade the coding performance, is also shown. The degradation occurs because of the decoder action taken when the number of blocks in the buffer exceeds 32. Under these conditions the decoder simply outputs data blocks without decoding at all, until the number of blocks is below the limit. The degradation is not significant for rates below 9.5K bits/sec (software), and 23K bits/sec (mixed). The degradation gets progressively worse with increasing data throughput, and is quite severe at 18K bits/sec (software) and 32K bits/sec mixed, for low signal-to-noise ratios. Note that the actual size of the buffer is 64 blocks, and the performance degradation could be reduced if a bigger buffer was used, with a more complex buffering algorithm.

Figure 6 shows the performance of the (23,11) expurgated Golay code using soft-decision mixed implementation decoding. When the full 10 perms are used it can be seen that the code's performance is only about 0.4dB worse than the optimum maximum-likelihood decoder (M.L.D.). Furthermore, restricting the maximum number of perms to 5, and 2 (as is used in hard-decision) degrades the code's M.L.D. performance by 0.9dB and 1.5dB respectively, at a user error rate of 10^{-5}. Also shown are the maximum throughput speeds that can be achieved at various signal-to-noise ratios, without any performance degradation. Operation above these speeds will involve a rapid degradation resulting in a performance curve similar to those in figure 5. That is, a curve which degrades from the point at which the operating speed exceeds the maximum speed indicated. However, degradation could be made more gradual by resorting to hard-decision decoding, rather than simply outputting undecoded blocks, when the buffer limit is exceeded.

7. Discussion

It has been shown that, even at poor signal-to-noise ratios, the 8080A microprocessor decoding system as built can achieve the theoretically predicted coding gains at speeds of the order of 20K bits/sec for hard-decision, and 1K bit/sec for soft-decision, by using permutation decoding on the (23,12) and (23,11) Golay codes. At high signal-to-noise ratios much faster speeds can be achieved. Given the present system, soft-decision throughput speeds could be increased by reducing the number of quantising levels to 8, thus reducing the number of bytes per channel bit transferred into the processor; or by operating the input under D.M.A. control, thus freeing the processor of input duties at the expense of more hardware.

The use of latest generation processors, however, together with program optimisation could be expected to increase throughput speeds and/or reduce implementation package count further. In particular, the 9080 is available in a version approximately twice as fast as the 8080A used. Alternatively, the 8085 makes possible the implementation of the hard-decision software-only program with a total of 5 packages, and would be 1.3 times as fast as the present system. In general, the maximum throughput rates obtainable with the present generation of general purpose 8 bit microprocessors would not be much more than twice those obtained with the existing system. The use of 16 bit processors, however, in particular the Plessey MIPROC would give speed increases of about ten times, and would be necessary if it were required to decode longer and more powerful codes. The only way to achieve even further speed increases would be to base a system on a bipolar bit-slice microprogrammable microprocessor, such as the 2900 series. Such a system would enable structure to be specifically tailored to the decoding task and would probably realise speed increases of one or two orders of magnitude, but at the expense of greatly increased complexity (30-40 chip implementation) and development time.

8. References

1. Wozencraft, J. M., and Jacobs, I. M. : Principles of Communication Engineering, Wiley New York, 1965.

2. Heller, J. A., and Jacobs, I.M. : 'Viterbi Decoding for Satellite and Space Communication', Linkabit Corporation, 1971.

3. Chase, D. : 'A Class of Algorithms for Decoding Block Codes with Channel Measurement Information', IEEE Trans., IT-18, 1972.

4. MacWilliams, F. J. : 'Permutation decoding of systematic codes', B.S.T.J., 1964, 43.

5. Goodman, R. M. F., and Green, A. D. : 'Microprocessor-Controlled Soft-Decision Decoding of Error-Correcting Block Codes', IERE Conference Proceedings, No. 37, Loughborough, 1977.