Proceedings of the 20th Princeton Conference on Information Sciences and Systems,
Princeton, New Jersey, March 1986

DECODING LONG CONVOLUTIONAL CODES WITH A VLSI PATH GENERATOR CHIP

by
R.M. Goodman and A.J. McAuley

Department of Electrical Engineering
California Institute of Technology
Pasadena, CA 91125

Keywords: error correcting codes, maximum likelihood decoding, convolutional codes

Abstract

In this paper we outline an convolutional code decoding algorithm which is capable of efficiently decoding long convolutional codes. Our particular interest is in using such an algorithm in conjunction with concatonated Reed-Solomon decoding to provide very high coding gains for deep space telemetry applications. In the paper we outline the decoding algorithm and the hardware neccessary to implement a fast megabits/second decoder. The hardware consists of a VLSI path generator chip (PGC) which operates in conjunction with a path search (PS) processor.

1. Introduction

Concatenated coding, with an inner convolutional code and a Reed-Solomon outer code, is known to provide excellent performance for high error rate channels. In order to make best use of the received information, the convolutional decoder should be minimum distance, use soft decision techniques and have a long constraint length. The Viterbi algorithm can satisfy the first two requirements; but the decoder complexity grows exponentially with constraint length, limiting the maximum decoder gain. In this paper we look at the VLSI implementation of a technique which allows much longer constraint lengths with soft decision decoding.

The decoding algorithm utilized in this paper is due to Ng, Goodman and Winfield [Refs 1-8]. The algorithm is maximum likelihood although it can operate in a suboptimum mode to increase throughput speeds. The algorithm uses a sequential decoding approach to avoid an exponential growth in complexity. In addition, the distance and structural properties of convolutional codes are used to considerably reduce the amount of searching needed to find the optimum path when a back-up search is required. The decoder does this in two main ways. First, a small set of paths called permissible paths are utilized to search the whole of a subtree for the better path, instead of actually using all the paths in the given subtree. Second, the decoder identifies which subset of permissible paths should be utilized in a given search and which may be ignored, because they cannot possibly result in a better path. In this way many unnecessary path searches are completely eliminated. Because the decoding effort required by the decoder is low, and the decoding processes simple, the algorithm opens the possibility of building high-speed long constraint length convolutional decoders with optimum performance.

The use of the effort reduction techniques, makes control of the algorithm a fairly complex operation. But, by only implementing the main computational element in VLSI, the control can be left to a microprocessor. The resulting 'path generator chip' (PGC) is capable of decoding at 20 megasegments/second: where one segment is a 3 bit quantization of a received data and parity bit. The chip is also capable of encoding at 20 megabits/second.

The PGC is composed of two serial-parallel Galois field multipliers, a number of registers and a small amount of multiplexers and exclusive-OR gates to control the flow of data. The chip continues to operate at its maximum speed, until an error is encountered. It then interrupts the control processor, so it can decide whether the chip has taken the wrong path. If the processor decides, from the information the PGC provides, that an error is possible, it halts the decoding while it checks to see if the chip needs to be reprogrammed to the 'best path'.

2. The Decoding Algorithm

2.1 Code Properties

Let us introduce some of the distance and structure properties of single-generator convolutional codes that are utilized in the decoding algorithm.

A single-generator convolutional code is one in which each message digit is encoded individually into V code digits, where V is a positive integer, giving a maximum information rate of $1/V$. The V coded digits for each message digit depend on both the present message digit and the $N-1$ previous message digits, where N is the constraint length of the code in segments. The code can be represented by its tree structure, the branches of which can be extended indefinitely from any node. Each branch has one segment of code digits associated with it, and the code digits of the two branches stemming from an arbitrary node are always ones-complements of each other.

The encoding operation is one of selecting a path through the tree in accordance with the message digits. At each node the upper branch is taken if the message digit is a zero, and the lower branch is taken if it is a one.

Consider, for any node in the infinite tree, all the paths that extend b segments forward from that node. The resulting subtree is referred to as a truncated tree, or b-unit, and is divided into two half-trees depending on which branch was chosen at the first node. The initial code tree (S) is the k-unit stemming from the very first node, and is divided into the upper-and lower-half initial code trees (S_0 and S_1, respectively).

We may now summarize several useful properties of these codes.

(a) The code is a group code. That is, if w and w' are two equal-length code paths, belonging to the intial truncated tree S, it implies that there is a path x such that $x = w \oplus w'$ is within S.

(b) If w and w' are paths in *opposite* halves of *any* b-unit, then $x = w \oplus w'$ is a code path in the lower-half *initial* code tree S_1.

(c) The distance between the two half trees of *any* b-unit is defined as the minimum Hamming distance between pairs of paths, one from each half tree.

(d) Combining properties (b) and (c) above, we can state that the minimum distance between half trees of *any* b-unit is equal to the weight of the minimum-weight path in S_1. We can then define a distance function $d(.)$ such that $d(b)$ is the minimum distance between half trees of any b-unit, and depends only on b, and not on the b-unit chosen. The guaranteed error-correcting capability of any b-unit is then $T(b)$, where $T(b)$ is the largest integer such that $T(b) \leq \lfloor d(b) - 1 \rfloor / 2$.

2.2 The basic decoding strategy

Consider the notation:
v the received sequence, which differs from the transmitted sequence due to errors
w the tentatively decoded sequence, a path in the code tree which is the decoders's tentative version of the transmitted sequence
$t = w \oplus v$ the test-error sequence, which has ones in the positions where w and v differ
t_b the sequence consisting of the last b branches of the sequence t.

Our basic decoding strategy is then as follows. We always seek a code path w which is at minimum distance $|t|$ from the received sequence v. In other words, a w is accepted to be the decoded sequence if and only if for all other paths w' in the corresponding truncated tree, w has minimum test-error weight. That is

$$|t| = |w \oplus v| \leq |w' \oplus v| = |t'|$$

We define the basic branch operation (BBO) to be the decoding action of a single branch forward extension which selects the latest segment w_1 of w. Whenever a decoded path w is accepted as being the minimum distance path, the decoder shifts out the earliest segment of w, which is assumed to be a correct representation of the corresponding segment of the transmitted sequence, and shifts in the newly received segment v_1 of v. The BBO then selects w_1, to be the segment closest in distance to v_1.

For the half-rate code, the BBO results in a w_1 that always has a test-error weight $|t_1| = |w_1 \oplus v_1| \leq 1$. Thus $|t_1|$ is either 0 or 1. If we assume that the new segment w_1 results from the extension of a path that has minimum test-error weight, the following are implied. Firstly, if $|t_1|$=0, the new path is guaranteed to have minimum test-error weight, and the decoder returns to the BBO. Alternatively, if $|t_1| = 1$, it is possible that there exists some other path w' with smaller test-error weight $|t'| = |w' \oplus v| < |t|$, and we are faced with a search to find it.

2.3 Searching for the better path

Let us assume that the decoder needs to search the b-unit which spans the last b segments of the code tree, for a w' with smaller test-error weight. We use a procedure based on property (b) of Section 2.2. This states that w' can be directly derived by the modulo-2 operation $w' = w \oplus x$, where x is a truncated path in the lower-half initial code tree. Also,

$$t' = w' \oplus v = w \oplus x \oplus v = t \oplus x$$

and so if w and w' are in opposite halves of a b-unit we can derive the test-error weight of w' by direct modulo-2 addition of t and the b-segment path x. This is still a cumbersome process, however, if all paths in the b-unit have to be used to search for w'. However, we have derived many conditions which the x must satisfy because of the code structure [1-8]. This serves to reduce the x required to search the b-unit to a very small number in most cases of interest. The reduced set of paths needed to search the b-unit are called permissible paths, (PPs), and denoted by P_i. For example, using a particular $\frac{1}{2}$ rate code with good distance properties we find that over six segments there are only three permissible paths which satisfy the conditions on P. These are $P_{(1)} = 31, P_{(2)} = 32201$ and $P_{(3)} = 310101$. It is therefore possible to search the entire 6-unit without back-up, by making only three test-error weight comparisions based on $|t'| = |t \oplus P|$.

Fig. 1. Shows the decoder correcting a 12 segment (24bit) received sequence which contains four errors. The effort is only 12 BBOs and four path map operations.

3. The Soft-Decision Algorithm

Soft-decision decoding gives an asymptotic coding gain of 3db over hard decision, and our algorithm easily extends to cover this. We assume that 8-level quantization is sufficient to recover most of coding gain loss, and henceforth all bit quantities are assumed to be represented by a 3bit soft vector where hard 0 = [000] and hard 1 = [111]. All distance metrics and path weights are now assumed to be soft weights, where the soft distance between a hard 0 and a hard 1 is 7 soft levels.

The decoding algorithm operates in a similar manner to the previous hard version. If the latest segment has a non-zero hard decision test error weight then a search via the stored permissible paths is required to find a path with a better lower soft test error weight, if one exits. A penalty of operating in the soft-decision mode is that more permissible paths must be stored than in the hard-decision case. However, note that this is ROM complexity not RAM. For example, the code used has a hard distance of 9 over 16 segments (32bits), giving an asymptotic soft correction power of 9 bits over this constraint length, and this requires 112Kbits to store the 5000 PPs needed. A length 25 (50bits) decoder gives 11 bit correction, and asymptomatically requires 92Mbits of ROM path storage. These storage figures are not excessive by modern ROM standards. Also, many high weight PPs can be omitted with minimal effect on the decoder output bit error rate, thus avoiding the exponential growth in storage requirement. [6]

The soft-decision decoder would not be feasible if it had to search through all the stored paths every time a non zero $|t_1|$ occurred. By utilizing the distance and structure properties of the code we have found several techniques to significantly reduce the amount of searching the decoder needs to do. Firstly, we can calculate the maximum possible improvement in soft error weight before starting the search. Thus if a particular path mapping achieves this improvement, we can immediately abandon the search. Secondly, not all PPs of each length must be tried in the attempt to find the better path. In fact for a given path length b only PPs of length b below a certain weight need be tried to achieve the seeked for test error weight improvement. The number of these paths at length b depends only on $|t_b|$ the soft weight of the test error sequence over the last b segments. Thus whenever the decoder needs to search for a better path by trying PP mappings from $b = 2$ to

b=DECL (Decoding Constraint Length), the $|t_b|$ is first calculated and this is used to directly identify (address) the subset of paths that have to be tried.

Consider the following example. Assume the all-zero sequence was transmitted and we are now on a path that gives us a soft test error sequence t = 00 04 01 00 01 00 40 40 61 00 40 in octal, containing 5 hard errors and a total soft error weight of 25. The maximum improvement in soft error weight is 1 soft level, because of the weight 4 latest segment. The following table shows the weight of the test error sequence $|t_b|$ over the last b segments, together with the upper bound on PP weight $|P_b|$ and the number of PPs at this weight.

b	$\|t_b\|$	$\|P_b\|$	$\#\|P_b\|$
2	4	-	-
3	11	-	-
4	15	4	1
5	19	5	2
6	19	5	1
7	20	-	-
8	20	-	-
9	21	-	-
10	25	7	3
11	25	7	3

The table shows that only 10 PPs need be searched to find the better path in this case. This is an order of magnitude less than the total number of PPs of length between 2 and 11, which is 197. In fact the 1 level improvement occurs on the third trial mapping, ie the second PP mapping of length 5, and the search terminates.

4. Decoder Hardware

A decoder based on the ideas above has two distinct processes: the 'path generator' (PG) which implements the BBO and the 'path searcher' (PS) which will attempt to find a better path (one with less soft error weight) than the one the PG is currently following.

Figure 2 shows a simplified systems diagram for decoding the contents of a buffer. The PG takes the next three bit (eight level) quantizations of data and parity on every C1 clock cycle, outputing the decoded data value of the data received in the $N+1$'th previous clock cycle. The PG will need to stop when the BBO is unable to follow a path without causing an error in the hard decision values. The PS then tries to find a better path, and if it does it loads this new value into the PG which is then allowed to continue with the next BBO.

The use of effort reduction techniques in the PS, makes control a fairly complex operation. It is therefore best implemented as a dedicated microprocessor capable of doing simple but fast arithmetic and look up operations. However, the PG chip greatly simplifies the PS operation by the inclusion of a comparitor and summer. These enable both the sum of the current segment test error weights, and the test error weight reduction resulting from a permissible path mapping, to be rapidly calculated in a segment serial manner. Both PG and PS are clocked by C2 during this process and interact in the following way. Firstly, the PS accesses the weight of the latest segment of the test error sequence from the PG sum output, in order to determine the maximum weight reduction possible. Secondly, as the search proceeds, at each segment of backup the PG presents the PS with the total weight of the test error sequence of that number of segments. The PS then performs its lookup of the PPs appropriate to that particular search length, based on the total test error reduction. The PS then presents these one at a time, in a segment serial manner, to the PG for testing. If the path is found to reduce the total error weight, but not give the maximum reduction, this PP is tagged by the PS which carries on searching for a better reduction. When the PS is satisfied no better path can be found, the PP giving the best reduction (if one was found) is represented to the PG which performs the mapping. The PS then signals the PG to resume the BBO.

Figure 2 shows a block diagram of the PG chip which is capable of doing the BBO at 20 megasegments/second (clock C1): where each segment is a three bit quantization of the received data and parity bits. The registers, multipliers, summer and comparitor all operate serially. The registers and multipliers are N bits long, where N is the constraint length in segments, and are clocked by either C1 or C2. The error registers are actually two dual shift registers moving data in opposite directions: one clocked by C1 (the main register) and the other by C2 (the test register). The latter is loaded with the contents of the former when LD goes high.

During a BBO, three bit data and parity values are loaded at DI ($d2, d1, d0$) and PI ($p2, p1, p0$) respectively. The most significant bit of the data, i.e the hard decision value d2, is pushed into the data register. Initially we assume this hard data value is correct. Therefore, the data error ($DE = e2, e1, e0$) is the bitwise exclusive-OR sum of the received data, with this hard value. That is: $e2 = d2 \oplus d2 = 0$; $e1 = d2 \oplus d1$, $e0 = d2 \oplus d0$. In order to calculate the parity error ($PE = f2, f1, f0$), assuming the hard data value is correct, we must recalculate the parity based on our current path. To do this we use two Galois Field multipliers (GFMs) operating in GF(2). Both GFMs are based on a fast systolic design with no global data lines and one operand hardwired as the generator polynomial.

The data error register stores a three bit quantization of the data error weight, corresponding to each of the N data values stored in the data register. The most significant bit of these $N-1$ (we always assume that the latest hard data value is correct) values are fed into the lower GFM. The result of this multiplication, together with the result of the top GFM (which recalculates the parity, based on the N last hard data values), are XORed to calculate the expected hard value of parity. The recalculated parity (q) is used to calculate the PE: $f2 = q \oplus p2$; $f1 = q \oplus p1$; $f0 = q \oplus p0$. If the most significant bit of the PE is high ($f2 = 1$), then it is possible there is a better path. Therefore $f2$ is used to stop the PG chip, at the end of the current C1 clock cycle, and start the external PS (via the ERROR signal).

In order to ensure the minimum total test error weight, DE and PE pass through a further XOR gate, where the signals can be conditionally inverted before being stored. If the total error weight (S = s3 ,s2, s1, s0), calculated by the adder A, is greater than seven; then the total weight can be reduced by assuming the hard value of data was in error (instead of the hard value of the parity, which is the default). With CH low the most significant bit of the result (s3), controls the XOR gates to achieve the desired conditional inversion operation.

While the new error weights are being calculated, the corrected data is simultaneously being fed out. This decoded output is evaluated by XORing the $N+1$'th data value with the most significant bit of the $N+1$'th data error value. Also note that, if the EN/DE signal is high, the output is simply the output of the top GFM.

That is, the encoded output of the $d2$ sequence.

If the ERROR line goes high, there is a possibility that there is a better path. Therefore, the PS must assume control of the PG, after the PG has completed its current cycle. The PS first takes LD high to load the main data and parity error registers into their respective test registers, and clear the comparitor and summer. After b clock cycles of the C2 clock, the SUM output will give the accumulated sum of the test error weight over the last b segments. The PS can store these values and use them to tell whether it needs to look for a new path based on stored threshold values. If it does, it will again bring LD high and feed the P-PATH and D-PATH lines with the first possible new path (PP1), one segment at a time. The DIFF signal then outputs the difference between the total test error weight of the current path (t), and that of the new path ($t \oplus PP1$) in a segment by segment manner.

If PS needs to instruct the PG to perform a mapping it restarts C1, sets DIR and CH high and reloads p1, in reverse order with zeros filled in to make it length N, into P-PATH AND D-PATH. Because CH is high, the G logic routes these latter two signals to the XOR gates in front of their respective error registers. Since DIR is also high the other input to the XOR gates will be the output of the error registers. The overall effect is thus to change the error path t to ($t \oplus PP1$).

The decoder as outlined above can operate at Megabit speeds. These are of course average speeds and an input buffer is needed to compensate for differing path search times. However, at output error rates of 10-5 the average number of PPs searched per search is less than one. Also, unlike other sequential decoding schemes the maximum number of paths searched is always finite.

4. Conclusions

In this paper we have outlined our convolutional code decoding algorithm, and the hardware neccessary to build a fast decoder. At present we are developing the Path Generator Chip and decoder techniques to decode a 1/2 rate code over decoding constraint lengths of 50 to 100 bits. This should give coding gains greater than the length7 1/2 rate Viterbi decoder, and operate at Megabit speeds.

References

1. W.H. Ng and R.M.F. Goodman, "An efficient minimum-distance decoding algorithm for convolutional codes," Proc. IEE, Vol. 125, No. 2, Feb., 1978.
2. W.H. Ng and R.M.F. Goodman, "Analysis of the computational and storage requirements for the minimum-distance decoding of convolutional codes," Proc. IEE, Vol. 126, No. 1, Jan., 1979.
3. R.M.F. Goodman and A.F.T. Winfield, "Soft-decision direct mapping decoding of convolutional codes," IEEE International Symposium on Information Theory, Grignano, Italy, June 1979.
4. R.M.F. Goodman, "Towards the maximum likelihood decoding of long convolutional codes," *Algebraic Coding Theory and Applications* Ed. G. Long, Springer-Verlag, 1979.
5. R.M.F. Goodman, "On the design of practical minimum distance convolutional decoders," *Algebraic Coding Theory and Applications* Ed. G. Longo, Springer-Verlag, 1979.
6. R.M.F. Goodman and A.F.T. Winfield, "Soft-decision minimum distance sequential decoding algorithm for convolutional codes," Proc. IEE, Vol. 126, Pt. F., No. 3, June 1981.
7. R.M.F. Goodman and M. Abdul-Wahab, "Multiple-path-tracking sequential decoding of convolutional codes," IEEE International Symposium on Information Theory, St. Jovit, Quebec, Canada, Sept. 1983.
8. R.M.F. Goodman and A.F.T. Winfield, "Asymptotically Maximum Likelihood Sequential Decoders," IEEE International Symposium on Information Theory, Brighton, U.K., June 1985.

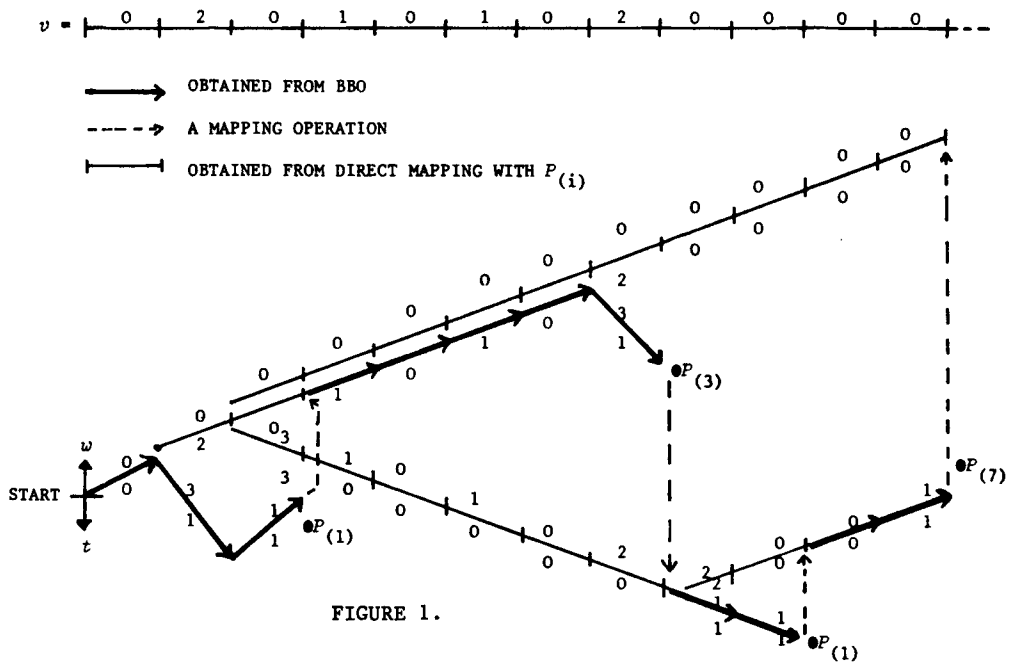


FIGURE 1.

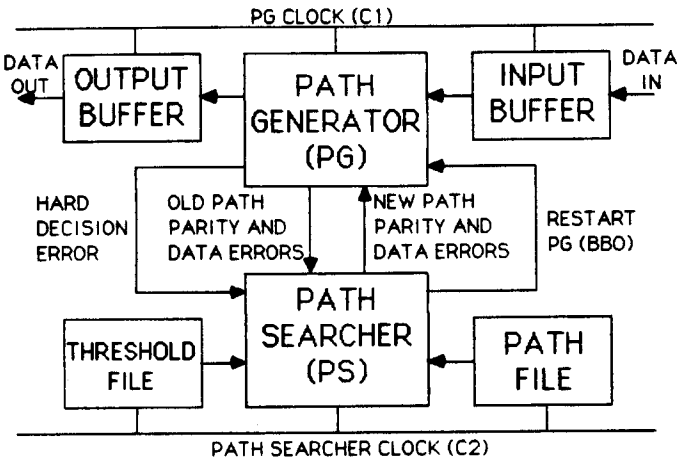
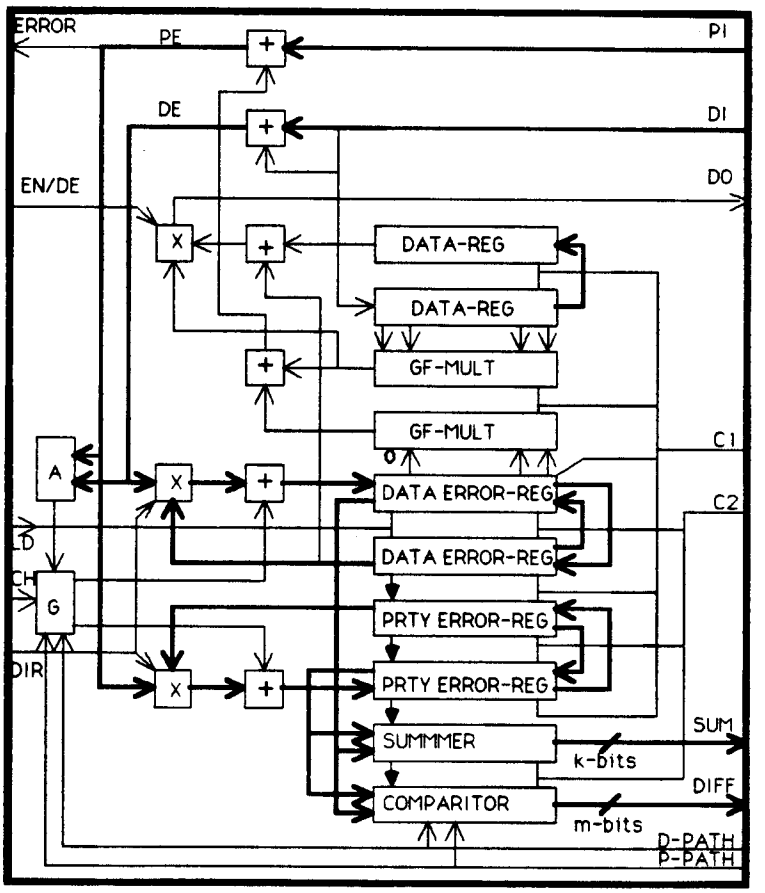


FIGURE 2 DECODER SYSTEM DIAGRAM



+...EXCLUSIVE-OR X...2:1 MULTIPLEXER A...4 BIT ADDER G...CONTROL LOGIC

—— one bit path ——— 3 bit bus (unless specified otherwise)

FIGURE 3 PATH GENERATOR CHIP (PG-CHIP) BLOCK DIAGRAM