A Table-Driven (Feedback) Decoder

Donald L. Bitzer and Mladen A.Vouk

North Carolina State University, Department of Computer Science, Box 8206
Raleigh, NC 27695-8206

Abstract

We present a table-driven technique for decoding of the convolutionally encoded data. The approach can be used in either the feedback or the direct mode. A data-independent syndrome vector is generated on the transmitted bits. If it is different from zero it is used to address pre-computed tables of corrections for the encoded bits. The decoding is performed as a separate step. When the feedback mode is used the error propagation is minimized by appropriate choice of the codes, the table construction and the decoding algorithms. The approach allows hardware simplicity comparable to majority-logic decoding, but has error correcting capabilities that can be made as close to the optimal as desired through the adjustment of the syndrome vector length and the correction table size. This makes the method very attractive for high speed satellite and network applications. The performance of the method can be further enhanced through soft detection.

1. Introduction

Error control techniques are increasingly being applied to digital communication links to enable significant performance improvements. These techniques enable, for fixed transmit (or receive) power levels and for allowable error probabilities, the transfer of more information per unit time. For example, a major feature of the advanced satellite communications technology is the use of dynamic rain fade compensation. The current error compensation specifications often call for a symbol rate reduction, and half-rate convolutional coding which results in an overall reduction in the data rate. Since it is envisioned that in the future satellite data transmission rates should be in excess of several hundred Mbps, the complexity and the speed of the error correcting hardware will play an important role. Similarly, fiber-based gigabit networks would normally be expected to operate under very low noise conditions. However, even very occasional errors may require relatively complex protocols and re-transmission of considerable amounts of data. While re-transmission cannot be ruled out completely it may be advantageous to use forward error recovery to lower the already low error rates even further and thus greatly simplify the communication protocols.

The hardware complexity required for the currently used error control techniques may be a problem in high speed applications. In this paper we discuss a possible alternative to the commonly employed Viterbi and sequential algorithms. We describe a simple and fast table look-up based feedback decoder[1] which allows hardware simplicity comparable to that of majority-logic decoding, but provides error correcting capabilities comparable to the Viterbi and sequential decoding approaches.

Section 2 briefly reviews the related work. Section 3 describes the table-driven approach, and section 4 provides a summary.

2. Summary of Related Work

Two more widely used encoding techniques are the block coding and the convolutional coding [e.g. 1 and references therein]. Convolutional codes are generally conceded to be operationally better than the block codes, particularly with respect to ease of implementation, equipment complexity, power consumption, and flexibility [e.g. 2].

Various algorithms are available for decoding convolutional codes. The Viterbi algorithm has received considerable attention [2 - 5, 13]. This algorithm is maximum-likelihood and optimum for the decoding of convolutional codes. A difficulty with the Viterbi decoding is the fixed amount of computation always required per decoder information block for a given code constraint length, and that this effort grows exponentially with the code constraint length. Under low noise conditions a more flexible (adaptive) algorithm may be desirable.

Sequential decoding represents an alternative procedure

[1] U.S. Patent applied for.

CH2859-5/91/0000/0385$01.00 © 1991 IEEE

[e.g. 1, 6 - 8]. The performance of sequential coding is slightly less than optimal, but the decoding effort is basically independent of the code constraint length, so large constraint lengths can be used, and very low error probabilities can be achieved. Usually, the code constraint length is 20 bits or more. The number of computations needed to decode a frame of data is a random variable. A typical load figure is 1 to 2 computations per decoded bit, however, noisy blocks may make sequential decoding impractical by requiring excessive computations to retrieve transmitted data.

An algebraic approach called majority-logic or threshold decoding can also be applied to convolutional codes [1, 9]. Majority-logic decoding differs from Viterbi and sequential decoding in the fact that the error detection process is data-independent, and that the final decision in an information block is based on only one constraint length of the received blocks rather than on the entire received sequence. Because of the latter, majority-logic decoding usually results in inferior performance when compared with Viterbi or sequential decoding where the correction decisions are made based on at least five constraint lengths. On the other hand, the implementation of a threshold decoder is much simpler, and it typically needs only one computation (cycle) per bit.

In all three approaches some of the more important design parameters are the code constraint length which heavily determines the ability to detect errors, the coding rate, i.e. the applied information redundancy, and the number of receiver quantization levels which can provide further enhancement such as weighting of each bit change by the signal to noise ratio for that bit, i.e. soft detection [e.g. 10].

An ideal decoder would have performance approaching maximum-likelihood, but would have hardware complexity and speed comparable to a majority-logic decoder. We now discuss an approach which may provide such a combination.

3. Table-Driven Decoding

We will explain the table-driven decoder using examples based on half-rate non-systematic codes. However, the technique can be used with any coding rate and with both systematic and non-systematic codes, although the performance is superior when non-systematic codes are used. For illustration a 2/3 coding example will be given without detailed explanations.

3.1 One-to-One Mapping

The table-driven decoding method is based on the existence of a one-to-one mapping of a set of encoded bits and a set of data bits. Even though the encoded bits in half-rate coding are generated at twice the information data rate, it is possible to find a relationship between the encoded bits and the data which involves the same number of bits.

Iteration	No. of Data Bits	No. of Encoded bits
0	L	2
1	L+1	4
2	L+2	6
	•••	
n	L+n	2(n+1)

Figure 1. On-to-one mapping in half-rate encoding with constraint length L.

Consider production of the encoded bits during half-rate encoding with constraint length (or code length) L (Figure 1). The first L data bits produce two encoded bits. Each additional data bit produces two more encoded bits. A **necessary** condition for the one-to-one mapping is that the number of data bits and encoded bits be the same (i.e. same number of elements is needed in each set). By requiring the number of the data and the encoded bits in the n^{th} iteration to be equal, and solving for n we have

$$L+n = 2(n+1)$$

$$n = L-2.$$

Therefore, $L+L-2 = 2L-2 = 2(L - 1)$ data and encoded bits are needed for uniquely resolving a message encoded with code of length L, and 2(L-1) bits must be taken at a time for it to be possible to have a one-to-one mapping. Since each bit can be in one of the two states, the number of elements in each mapping set is $2^{2(L-1)}$. However, a reduced set of 2(L-1) independent basis elements can be used to obtain an element of the complete mapping set by combining the basis elements (vectors) through the exclusive-OR operations.

To illustrate the ideas we use as an example code of length L = 3. Let the two code words for half-rate coding be $C_1 = 011$ and $C_2 = 111$. The number of bits necessary for one-to-one mapping is 2(L-1) = 4. Hence, in this simple example there are $2^{2(L-1)} = 16$ elements in the complete mapping table, or four elements in the reduced table. It is easy to determine the table relationships if one starts with each of the 4 bit combinations of the data and applies the two code words to obtain the related 4 encoded bits. The process is illustrated in Figure 2 where P_1 and P_2 denote the encoded bits produced by half-rate encoding.

The resulting full table is shown in Figure 3, and it is

easy to see that it can be constructed by using the exclusive-OR operation to combine one, or more, of the four basis data vectors (0001, 0010, 0100, 1000) and their corresponding encoded bit vectors.

Parity Bits		First Pair	Second Pair
Data In:		101101	101101...
C_1 mask	AND	011	011
XOR 3 bits		001 = 1 = P_1	011 = 0 = P_1
Data In:		101101	101101...
C_2 mask	AND	111	111
XOR 3 bits		101 = 0 = P_2	011 = 0 = P_2
Parity Out:		1000 ...	

Figure 2. The process of half-rate encoding

Full Encoding Table

	Encoded Bits	Data
0	0000	0000
1	0011	0001
2	1111	0010
3	1100	0011
4	1101	0100
5	1110	0101
6	0010	0110
7	0001	0111
8	0100	1000
9	0111	1001
10	1011	1010
11	1000	1011
12	1001	1100
13	1010	1101
14	0110	1110
15	0101	1111

Reduced Encoding Table

	Data	Encoded Bits
1	0001	0011
2	0010	1111
4	0100	1101
8	1000	0100

Figure 3. Full and reduced encoding tables.

For example, the output (data = 1011..., encoded bits = 1000) could have been obtained by exclusive-OR (XOR) of the "data" basis elements of 1000, 0010 and 0001, i.e.

Data: 1000 *XOR* 0010 *XOR* 0001 = 1011
Encoded Bits: 0100 *XOR* 1111 *XOR* 0011 = 1000

Converting the data bits into the encoded bits is relatively easy, and can be implemented in very simple ways. However, converting the encoded bits back into the data bits (with error correction) is normally a more difficult operation. In our case, this can be easily accomplished conceptually using a reduced table consisting of the basis elements of the encoded bits instead of the data bits[2].

[2] Reduced decoding table – Encoded bits: 0001, 0010, 0100, 1000; corresponding Data: 0111, 0110, 1000, 1011.

The **sufficient** condition for unique table-based conversion of encoded bits to data is that the encoded bits matrix of the reduced encoding table for codes C_1 and C_2 can be transformed[3] to an identity matrix of order 2(L-1). When the same transformation operations are also applied to the data matrix of the reduced table we obtain a complete decoding table. If, in this process, one can obtain a single digit in each basis encoded bit position the mapping data-encoded-data is one-to-one. If it is impossible to get the unique reverse mapping, then there is no two-way one-to-one mapping, and these code words should not be used.

In general, mapping for a R=p/q rate code (p uncoded bits results in q encoded bits, p<q) requires q(L-p)/(q-p) wide tables. For instance, a 2/3 encoding with 4 bit non-systematic codes requires 3(L-2) = 6 bit wide tables[4].

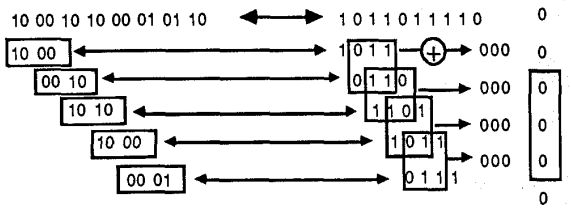

Figure 4 Table-driven decoding of uncorrupted encoded bit stream.

3.2 Error Syndrome Generation

To illustrate the essence of the above decoding procedure we refer to Figure 4 which shows the table-driven conversion of a stream of uncorrupted encoded bits into data. Each set of four encoded bits gives its corresponding four data bits. By shifting over two encoded bits at a time we shift the data bits one bit at a time. In this example, because the encoded bits remain uncorrupted, the corresponding (overlapping) data bits obtained from different sets of encoded bits yield the same value (boxed sets on the right/data side of the figure). Consequently, the three overlapping data bits obtained each time an encoded bit is shifted over by two produce a difference of zero. This difference from successive table look-ups will always be zero if there are no errors in the encoded bits and, in fact, it can be shown that the syndrome generating process is data independent.

[3] For example, by back-substitution based on the exclusive-OR operations on the matrix rows.

[4] A 2/3 rate coding example – Codes: 1110, 1001, 1011; Data: 000001, 000010, 000100, 001000, 001000, 010000, 100000; Encoded bits: 000011, 000101, 011100, 101111, 100000, 111000;

When a single bit error occurs in the encoded stream as shown in Figure 5, then the difference between the successive overlapping bits may not be zero. It can be shown that it will either be zero, or a constant which depends only on the two code words chosen. Since the effect of the multiple errors is to produce a differential sequence which is, in fact, the exclusive-OR of the differences obtained for the errors taken independently, the resulting differences for the multiple errors will also be zero, or the same constant. It can be also shown that the patterns of zero, and this constant N, become the sequence which is the reverse sequence of the opposite code word used to generate the encoded bit in error. One can represent these differences of zero or N as either zero or one. We call this sequence "error syndrome sequence". The syndrome sequence (s-bits or parity-check sum) can be generated readily by a shift register (as shown in Figure 6). Note that this syndrome generation process is essentially the same as the one that would be used for non-systematic half-rate code in majority-logic decoding [1]. However, there is a difference between the way majority-logic decoding treats this syndrome vector, and the way it is treated in the table-driven approach.

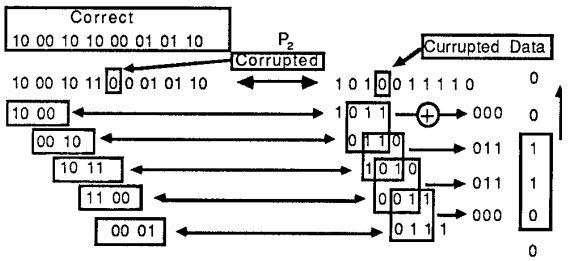

Figure 5 Encoded stream with a corrupted encoded bit P_2 and the corresponding corrupted data stream.

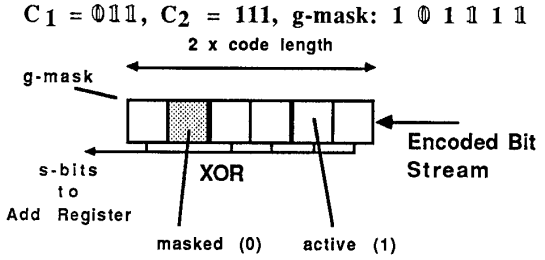

Figure 6. Syndrome bit generation

To understand the difference consider the production of the syndrome bits for half-rate encoding with constraint length L (Figure 7). The first 2L encoded bits produce one syndrome bit. Each additional pair of encoded bits produces one more syndrome bit. From Figure 5 and Table 3 we see that a two-bit block with an encoded bit in error can account for at most three (in general, L for half-rate codes) differences (s-bits) before it moves out of the width of the mapping tables. Therefore, setting the number of syndrome bits in the n^{th} iteration equal to L and solving for n yields

$$n = L-1$$

Thus, 2L+2(L-1) = 4L-2 encoded bits are needed for the complete generation of a syndrome vector of length L. This means that for the correct decoding and error correction based on the syndrome vector, we must consider 4L-2 encoded bits because they all influence a syndrome vector of length L. The classical majority-logic decoding theory [e.g. 1], although it generates the syndrome vector correctly, tends to truncate the extra 2(L-1) bits. This can degrade performance of a typical majority-logic decoder[5].

Iteration No.	of Encoded Bits	No. of Syndrome bits
0	2L	1
1	2L+2	2
2	2L+4	3
	•••	
n	2L+2n	n+1

Figure 7. Syndrome bit generation for half-rate coding with constraint length L.

In addition to being used for error correction, the syndrome vector is most of the time even a better tool for error detection. However, it can be shown that there are special error patterns which can masquerade as correct uncorrupted encoded representing data and produce a syndrome of zero. This is a basic property of any convolutionally encoded data stream and applies to all decoding approaches. For half-rate coding the basic "zero" is the reverse of the g-mask (see Figure 6). Other "zero" patterns can be formed by appropriately shifting this basic pattern, and by combining it with itself through exclusive-OR operations. The longer the constraint length L, the less likely it is that such a "zero" vector will go undetected.

---

[5] In general, p/q rate encoding results in $w_s=[(L-p)/(q-p) + 1]$ s-bits over the mapping table width, g-mask is qw_s long, and $\{q[2(L-p)/(q-p) + 1]\}$ encoded bits are required to generate these s-bits. For 2/3 coding there are 3(L-2)/3 + 1 = L-1 differences over the table width of 3(L-2) bits, and therefore 6L-9 encoded bits are responsible for the L-1 s-bits. In our 2/3 rate coding example L=4, 3(L-2) is 6, L-1 is 3, and the g-mask is 3(L-1) = 9 bits long and equal to 011001111. The corresponding reduced decoding table – Encoded bits: 000001, 000010, 000100, 001000, 010000, 100000; decoded Data: 110110, 110111, 110100, 101101, 011101, 010000.

3.3 Correction of Encoded Bits Through Table Look-Up

The process of correcting the encoded bit errors employs table look-up. This table is generated by producing the syndrome pattern that will result from a combination of errors over a range of at least (4L-2) encoded bits. The encoded bit range is usually centered around the encoded stream window (encoded bit locations) to which single bit corrections will be applied. Since the syndrome bits are data independent, in our combination patterns we only need include bits which are in error. The syndrome bit patterns are generated starting with the most likely combination of errors. At low error rates this is usually single bit errors (Figure 8), followed by two bit errors, etc. At higher error rates the sequence may start with some other error combination. We then use the syndrome bit sequence as an address (s-address) into a table. At an s-address the content of the table is a zero if the pattern does not have an error in the bit location for which the table is constructed, and a one if it does (P_A and P_B in the vertical transparent box in the encoded stream of Figure 8). There are many more zeros in such a table than ones. If there were no limits on the size of the table, i.e. the number of syndrome bits used could be as long as one wished, then the error correction could be made more and more perfect.

$C_1 = 011,\ C_2 = 111$

#	encoded bits	s-bits	decimal	
0	00 00 00 00 00	00000	0	
1	00 00 00 00 01	11100	4	
2	00 00 00 00 10	11000	0	
3	00 00 00 01 00	01110	6	
4	00 00 00 10 00	01100	4	Conflict
5	00 00 01 00 00	00111	7	
6	00 00 10 00 00	00110	6	
7	00 01 00 00 00	00011	3	
8	00 10 00 00 00	00011	3	
9	01 00 00 00 00	00001	1	
10	10 00 00 00 00	00001	1	

d	s-address	P_B	P_A
0	00000	0	0
1	00001	0	0
2	00010	x	x
3	00011	0	0
4	00100	0	0
5	00101	x	x
6	00110	1/0	0
7	00111	0	1
8	01000	-	-
9	01001	-	-
10	01010	-	-

Figure 8. Encoded stream with single bit errors, and the corresponding correction table based on the 3 bit s-patterns.

The top part of Figure 8 shows the encoded stream (4L-2 = 10 bits wide) with the corresponding s-bits generated on the basis of these 10 bits, and the resulting 3-bit correction table address, d (unshaded bits). Note the right-to-left bottom-to-top travel of the s-bit patterns across the s-stream in synchronization with the diagonal left-to-right bottom-to-top encoded bit error movement. The lower part of the figure shows the correction table based on the content of the two central bits in the encoded stream. Some of the entries in this table will be empty (x) because one bit errors do not generate these addresses. Those that have zero in the sampling window will have a 0 in the table, and those that have a one will have 1 in the table. In one case (table address 6) we have a double entry 1/0 for P_B indicating a conflict. We discuss this below.

In practice, only a finite number of syndrome bits can be used. Consequently, different sets or different combinations of errors can produce the same combination of the syndrome bits over a finite range and thus lead to an ambiguous determination of the correctness of the received bit stream. We consider the example table for one-bit errors. The addresses are three bits wide, and there is address degeneracy. For example, if two different error patterns produce the same syndrome address, but the bit in question in the error pattern is the same, then no error is created by the table (e.g., the encoded stream entries #1 and #4 in Figure 8 both produce the same table address, 4, but neither requires a change in the encoded bit). However, if the two bits are different at that location in the error pattern, there is a contradiction (e.g., the P_B entries for encoded streams #3 and #6 in Figure 8 are different but produce the same correction table address, 6). This may lead to an erroneous correction, or result in no correction of a encoded bit error. If this problem is not resolved in some way the entry in the table should be set to zero to reduce the probability of miscorrection.

Once the correction tables have been built, correction of the encoded bits occurs by forming the syndrome bit based address, accessing the table, and determining if the particular encoded bit in the correction window position should be changed. The correction may be applied immediately, or with a delay pending confirmation of the change in one of the future correction cycle. When feedback is employed, then once the correction has been applied, the s-vectors are re-computed (or adjusted) to take the correction into account.

3.4 Decoding With Disambiguation

There are several options in the situation where a conflict occurs in the correction tables because of the overlap of the s-addresses from two different encoded bit error patterns. One option is to extend the syndrome address, which is normally based on 4L-2 encoded bits, to encompass encoded bits further out. Consider Figure 8. If the encoded bits range on which s-addresses are generated is extended by

4 bits on the right side of the original 10 bits, or alternatively symmetrically by 2 bits on each side of the original 4L-2 range, then the generated s-addresses will have two bits more. These additional two s-bits are shown in the shaded vertical strip in Figure 8 assuming the 4 bits, the one-sided extension in the encoded stream covers, are zeros. We see that by using the 5 bit s-addresses instead of the 3 bit addresses we resolve the conflict between the #3 and the #6 encoded stream entries since in the correction table the addresses for these two error patterns would now become decimal 14 and 6, respectively, instead of 6 and 6. This table now guarantees correction of all single encoded bit error patterns within an effective constraint length of 14 encoded bits (4L-2+extension = 14). In general, by s-address extension we can guarantee complete disambiguation for any number of error patterns. For example, using half-rate coding and 8-bit non-symmetric codes with one-sided 24-bit extension, i.e. a 20-bit address (1 Mbit by two bit) correction table , we can guarantee complete disambiguation of all addresses for one, two, three and four bit errors in the encoded stream. The problem is that the correction table size grows exponentially with the size of the s-vector, so direct s-vector extension may very quickly become too expensive and other methods have to be used.

Another option (usually combined with the first one) is to resolve the ambiguity through a second or third look-up in the same table. For example, the additional look-ups may involve computation of another syndrome address now positioned around a specific, and possibly uncorrected, encoded bit that normally would have been associated with one of the error patterns that resulted in the address overlap. For instance, when the 3-bit s-address for the encoded stream entry #3 is re-computed at the position of the two encoded bits on the right of the transparent strip it is 7. For this address the correction table indicates that the encoded bit P_A is wrong (one). On the other hand, the second look-up generated around the same bits for the stream image #6 gives address 3 in the correction table which indicates that there is no error in P_A (zero). This differentiates between the #3 and the #6 patterns. In many cases it is sufficient to use only one additional look-up to resolve the overlap problem and guarantee correct decision. However, since the position of the disambiguation pair can be different for every conflict, a problem with this technique is the extra storage needed in the correction tables to indicate which of the bit pairs should be used.

There are other options such as the use of several different tables. This allows, for example, tuning of the correction process to specific noise profiles (e.g. burst errors). In this paper we will not discuss these options, but it is important to note that the use of the shifted s-vectors, or different sets of s-bits and different tables, in effect increases the s-vector length without exponentially increasing the table size.

3.5 Decoding and Soft Detection

If all encoded bits are corrected in the above process, conversion of the encoded bits to the data bits could proceed in a very simple way as illustrated by one of the relationships in Figure 9. However, even when some encoded bits are questionable, it is possible to remove several of them from the decoding scheme and still recover the original data.

To illustrate this we refer to the reduced decoding table in Figure 9. Remembering that shifting two encoded bits shifts over one data bit at the time, we can see that any given data bit can be derived from 2(L-1) sets of encoded bits. Given a set of encoded bits one can go to the table and obtain a set of p-bits to use in the translation which have not been marked as potentially problematic. The marking of these bits can either be obtained from the knowledge of the uncertainty in the table look-up (e.g. encoded stream entries #3 and #6 in Figure 8), and/or from the signal to noise detection (soft detection) for any given bit.

For example, assume that to start with the encoded bits are P_1 through P_4 as given in the first row below the reduced table in Figure 9. These four encoded bits can be combined by exclusive-OR (XOR) operations to produce D_1 through D_4 data bits. In particular, from the rightmost column of the **Data** side of the reduced table we see that data bit D_4 will be influenced only by the state of P_1 and P_4 bits, i.e. P_1 XOR P_4. Shifting in two new encoded bits, P_5 and P_6, and one data bit D_5 provides D_4 mappings given in the second row of the **Mapping** column, etc. Relationships for other data bits can be obtained in a similar way.

Reduced Decoding Table

Encoded Bits	Data	Mapping
1 0 0 0	1 0 1 1	
0 1 0 0	1 0 0 0	
0 0 1 0	0 1 1 0	
0 0 0 1	0 1 1 1	
$P_1P_2P_3P_4$	$D_1D_2D_3D_4$	$D_4 = P_1+P_4$
$P_3P_4P_5P_6$	$D_2D_3D_4D_5$	$D_4 = P_3+P_5+P_6$
$P_5P_6P_7P_8$	$D_3D_4D_5D_6$	$D_4 = P_7+P_8$
$P_7P_8P_9P_{10}$	$D_4D_5D_6D_7$	$D_4 = P_7+P_8$

Figure 9. Redundancy available for decoding of individual data bits.

During the analysis of the encoded bits we may suspect the quality of say P_1 due to a conflict in the correction table,

or because of the signal-to-noise ratio. Then the decoding of D_4 based on P_1 would also be suspect. However, if bits P_3, P_5 and P_6 are not problematic, then one can derive D4 from this second (or some other) relationship without miscorrection. This process has been successfully used to limit the error propagation from the corrupted or the miscorrected encoded bits to the decoded data bits. A detailed analysis of the process is in progress.

For illustration consider the conflict situation shown in the Figure 8 correction table. Suppose that to start with P_1 corresponds to P_A and P_2 to P_B in the encoded stream entry #6. The correction table shows that there is no change for P_1 and therefore it is marked as being correct. However, there is an s-address conflict for bit P_2. Therefore we mark P_2 as problematic. To be on the safe side the correction is not applied, and the (potential) error in the encoded stream is allowed to remain. Next, we shift in two more encoded bits, P_3 and P_4. Now the encoded stream image is that given by the #4 entry. For both bits the correction table address 4 indicates no change (as it should) and both bits are marked as correct. The next two bits that are scrutinized are P_5 and P_6. The appropriate encoded stream image is that of the #2 entry which reduces to the s-address of 0. Again no change is indicated by the correction table, and both bits are marked as correct. We now have a sufficient number of correct encoded bits to correctly decode bit D_4 using the second decoding relationship. The encoded bit error that went uncorrected because of the table conflict was corrected by the translation process of the encoded bits to data, and the encoded bit error did not propagate into the decoded data stream.

In some situations (e.g. burst errors, high error rates) it is possible that all relationships for a decoding of a data bit contain at least one problematic encoded bit. In that case an option which gives reasonable results is a table-based minimization of the number of changes that need to be applied to (problematic) encoded bits in order to make all relationships for that data bit agree.

3.6 Performance

For the errors for which a table entry is unambiguous we can guarantee correction. For the other errors, its is possible to compute and measure the probability that the correction table entry fails to indicate the appropriate course of action. This suggests the following approximate model for the performance of a simple single look-up table-driven feedback decoder on a binary symmetric channel given a convolutional code with coding ratio R=p/q, and constraint length of L. The bit error probability after correction of the first block of q encoded bits, $P_{p1}(E)$ is

$$P_{p1}(E) \le \frac{1}{q} \sum_{i=t+1}^{n_e} B_i \binom{n_e}{i} p^i (1-p)^{n_e-i}$$

where t is the number of encoded bits in error that are guaranteed correction through the table look-up, $n_e=q[2(L-p)/(q-p)+1]+$(extension bits) is the effective constraint length, B_i is the probability that table fails for an i-bit error within n_e, and p is the channel transition probability. When feedback is applied each correction subtracts the error from the encoded stream and also adjusts the syndrome vector. If the there is no error propagation then the bit error probability for any block, $P_p(E)$, is equal to $P_{p1}(E)$. This is the "Process Model 2" in Figure 10. In practice, however, error propagation will occur, so the performance of the basic error correction mechanism will be degraded. The error propagation in the encoded stream may be limited by choosing the codes with good resynchronization properties, or by some other mechanism such as temporary reduction of the feedback once an error burst is detected. The error propagation into the decoded information blocks may be limited through selective use of correct encoded bits (see section 3.5).

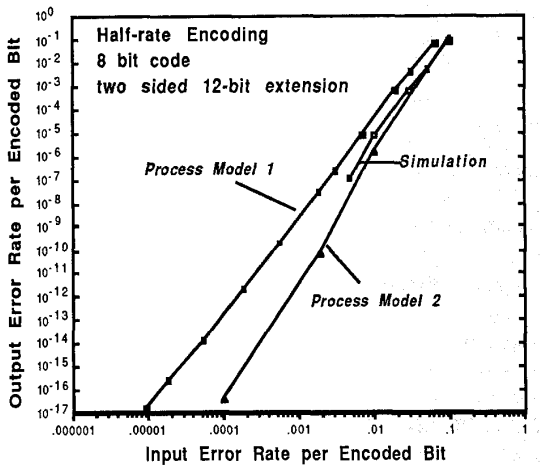

Figure 10. Illustration of the performance of the table-driven error correction approach for half-rate encoding based on a simulation and two probabilistic models.

We have built simulators for the outlined approach. The preliminary results are very encouraging. Example of a half-rate coding simulation is shown in Figure 10. The obtained output error rate per encoded bit is plotted against the channel error rate per encoded bit. Also shown are two process models. The Process Model 1 is a pessimistic model based on an unstructured (random) table, while the

Process Model 2 was described in the preceding paragraph. We used 8 bit codes with two-sided 12 bit extension (20 bit s-addresses). Results show that a simple look-up can guarantee correction of all 1, 2, and 3 bit errors (t=3) within an effective constraint length window of n_e=54 encoded bits. The corresponding probability of not correcting 4 bit errors within the same window is about 0.000003 (B_4), for the codes and the variants of the technique we have examined so far. The probability of not correcting 5 and 6 bit errors is about 0.0007 (B_5=B_6), and the probability of not correcting 7,8,9 and 10 bit errors is 0.33. One-sided extensions provides far better performance but requires more sophisticated error propagation control.

4. Summary

We have described a novel data-independent error correction and decoding approach for convolutionally encoded bit streams. The codes have to satisfy certain criteria for the approach to work. One criterion is that they must provide an orthogonal set of transformation vectors which allows a two-way one-to-one mapping between the uncoded data and the encoded stream. The error correction and decoding process can be reduced to a set look-ups in pre-computed tables.

A systematic mapping of single-bit, two-bit, three-bit, etc., encoded stream errors into error syndrome bit patterns gives correction tables which can then be used at run-time to map observed syndrome patterns into the information on which of the encoded bits is likely to be corrupted. To take full advantage of the error correcting capabilities of the syndrome vectors, decoding has to be augmented with disambiguation indicator bits and error propagation limiting process. The tables have to be built for the most likely error patterns. The table size depends on the number of bits in error that we wish to correct at run-time. Theoretical models and simulations show that the method is fast, and can provide decoding that competes with the commonly employed Viterbi decoding. The speed is the result of the intrinsically low complexity of the table look-up process, and of the fact that the existence of an error can be very quickly determined from the syndrome bits while the actual look-up and error correction needs to be done only on the average. The performance stems from the fact that the table-driven approach bases correction decisions on at least three to five code constraint lengths. Soft-detection enhancements are possible.

A number of research issues related to the table-driven approach remain open. For example, improvement of the correction and the decoding table efficiency and reduction of its size. We plan on building prototype decoding units to demonstrate the approach. Two routes will be taken. In one we will use standard ECL logic chips to provide XOR operations and look-ups. In the other approach we will attempt to utilize the BLITZEN[6] chip [12] to exploit parallelism inherent in the proposed decoding method.

Acknowledgments

We are grateful to Mr. Christopher Alix (U. of Illinois, Urbana-Champaign) for his invaluable help and many contributions in the first stages of the project. Our thanks also go to Mr. Wang LiFeng and Ms. Elena Gonzalez of NCSU for their continued involvement in the project.

Bibliography

[1] S. Lin and D.J. Costello, Jr., *Error Control Coding – Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, N.J. 07632, 1983.

[2] G.D. Forney, Jr., R, G, Gallager, G.R. Lang, F.M. Longstaff, and S.U. Qureshi, "Efficient Modulation for Band-Limited Channels," IEEE J. on Selected Areas in Communications, Vol. SAC-2, No. 5, pp. 632-647, September 1984.

[3] A.J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," IEEE Trans. Inf. Theory, IT-13, pp 260-269, April 1967.

[4] G.D. Forney, Jr., "Maximum Likelihood Sequence Estimation of Digital Sequences in the Presence of Intersymbol Interference," IEEE Trans. Inf. Theory, IT-18, pp 363-378, May 1972.

[5] G.D. Forney, Jr., "The Viterbi Algorithms," Proc IEEE, 61, pp. 268-278, March 1973.

[6] J.M. Wozencraft and B. Reiffen, *Sequential Decoding*, MIT Press, Cambridge, Mass., 1961.

[7] R.M. Fano, "A Heuristic Discussion of Probabilistic Decoding," IEEE Trans. Inf. Theory, IT-9, pp. 64-74, April 1963.

[8] F. Jelinek, "A Fast Sequential Decoding Algorithm Using a Stack,: IBM J. Res. and Dev., 13, pp. 675-685, November 1969.

[9] J.L. Massey, *Threshold Decoding*, MIT Press, Cambridge, Mass., 1963.

[10] G. Ungerboeck, "Trellis-Coded Modulation with Redundant Signal Sets Parts I & II:" IEEE Communications Magazine, Vol. 25, No. 2, pp. 5-21, February 1987.

[12] D.W. Blevins, E.W. Davis, R.A. Heaton, and J.H. Reif, "BLITZEN: A Highly Integrated Massively Parallel Machine," J. of Parallel and Distributed Computing, 8, pp. 150-160, 1990.

[13] G.D. Forney, Jr., "Convolutional Codes II: Maximum Likelihood Decoding," Inf. Control, 25, pp. 222-266, July 1974.

---

[6] The BLITZEN project was in part sponsored by NASA GSFC. Production of BLITZEN chips for use in spaceborn equipment is under consideration.