Design of a Rate 6/7 Maximum Transition Run Code

Barrett Brickner and Jaekyun Moon
Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455

***Abstract*—Maximum transition run (MTR) codes provide significant minimum distance gains when used with sequence detectors operating at high linear densities. A method for reducing the RLL *k* constraint associated with MTR block codes is presented. A block decodable, rate 4/5 MTR code with *k*=4 illustrates the technique. This reduction of *k* is combined with sliding-block decoding to develop a 97.8% efficient rate 6/7 MTR code with *k*=8.**

I. Introduction

MODULATION codes are used to eliminate troublesome patterns or to introduce certain desirable characteristics in the recorded sequences [1]. An important case of the former is suppression of minimum distance error events, which dominate the performance of sequence detectors. At low densities, the minimum distance error is a single bit error, which does not allow for an improvement via the elimination of input patterns. However, as linear densities in the magnetic recording channel approach 2.5 bits per PW_{50}, the event changes. Assuming the NRZ input symbols (write current levels) are $\{-1,+1\}$, the error pattern $e_k = \pm(2,-2,2)$ dominates. Examination of the input patterns reveals that at least one pattern in each error generating pair contains three or more transitions. Therefore, a code that eliminates these error prone patterns by limiting the maximum number of consecutive transitions to two can improve the performance of near-optimal detectors.

The class of codes known as maximum transition run (MTR) codes limits the number of consecutive transitions to *j*=2, and yields a coding gain [2][3]. The usual RLL *k* constraint is retained for timing recovery, leading to an encapsulation of the code parameters as MTR(*j*;*k*), where *j* is the maximum number of consecutive transitions. Prior to write precompensation, transitions allowed by the MTR code have a constant phase relative to the write clock. This is in contrast to the ternary 3PM code in which a pair of closely spaced transitions are used as a third symbol by shifting the pair half of the bit window to place the zero crossing at the same position in the bit window as the peak of an isolated pulse [4]. The codes developed in this paper are all MTR(2;*k*) types. Although the MTR code provides a concise set of constraints to yield the desired coding gain, similar distance gains may be obtained by other constraints such as those developed for E^2PRML [5].

II. Rate 4/5 Code Design For Reduced *k* Constraint

One design methodology for MTR block codes is to maximize the code rate and then minimize the RLL *k* constraint. In doing so, however, it is possible to arrive at codes with reasonable rates but large values for *k*. To reduce *k*, a simple method for designing a two-state encoder for use with block decoding is proposed. With a block code, long periods of nontransitions generally occur when a codeword that ends with a string of zeros is concatenated with one that beings with a string of zeros. These long runs can be broken by inserting ones at the boundary. The key is to use codewords with a pair of ones at the boundary when the adjacent codeword has a zero next to the pair of ones. Because these words are not allowed by the block encoder, they can be decoded unambiguously.

To illustrate, this method is applied to the rate 4/5MTR(2;8) block code described by the mapping in Table I. If the encoder state *z* is defined as the trailing bit of the previous codeword, then the following conditional mappings can be employed:

$$y|_{x=0001} = \begin{cases} 11001, & \text{if } z=0 \\ 00001, & \text{if } z=1 \end{cases} \tag{1}$$

and

$$y|_{x=0010} = \begin{cases} 11010, & \text{if } z=0 \\ 00010, & \text{if } z=1\ . \end{cases} \tag{2}$$

The vector $\boldsymbol{x} = (x_1 x_2 \ldots x_m)$ represents the *m*-bit data word, and $\boldsymbol{y} = (y_1 y_2 \ldots y_n)$ is the *n*-bit codeword. Equations (1) and (2) implement the substitution rule

$$\ldots.0,000.. \rightarrow \ldots.0,110.. \tag{3}$$

where the comma indicates the boundary between two codewords. The results is a rate 4/5MTR(2;6) code that requires a two state encoder and block decoder. By looking forward to the next data word, the encoder can determine if the following codeword will begin with a zero. In that case, the substitution

$$..000,0\ldots. \rightarrow ..011,0\ldots. \tag{4}$$

is applied to further reduce *k*. As shown in Fig. 1, the encoder is a two-state encoder that uses the present data word $\boldsymbol{x}$ and the next data word $\boldsymbol{w}$ to produce block decodable 4/5MTR(2;4) codewords.

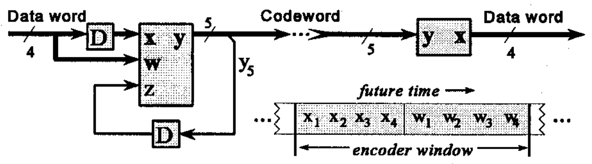

Fig. 1. Block diagram for the 4/5 MTR(2;4) encoder/decoder.

The encoder block is described by the Boolean logic functions

$$\begin{aligned}
k &= w_1 w_2 w_3 + w_3 w_4 + \overline{w_1}\,\overline{w_2} \\
y_4 &= \overline{x_2}\,\overline{x_4}\,\overline{k} + \overline{x_2} x_3 \overline{x_4} + x_1 \overline{x_2} x_3 + \overline{x_1} x_2 x_3 \\
y_1 &= \overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4} + \overline{x_1}\,\overline{x_2}\,\overline{z} + x_3 \overline{y_4} + x_3 x_4 \\
y_2 &= x_3 \overline{x_4} y_1 y_4 + \overline{x_3} x_4 y_1 + x_1 \overline{y_1} \\
y_3 &= x_2; \qquad y_5 = x_4 \overline{y_4} + \overline{x_3} y_4
\end{aligned} \tag{5}$$

and the corresponding decoder by

$$\begin{aligned}
x_2 &= y_3 \\
x_3 &= y_1 \overline{y_2}\,\overline{y_4} y_5 + y_4 \overline{y_5} + y_1 y_3 \\
x_4 &= y_1 \overline{y_2} y_4 x_3 + \overline{y_4} y_5 \\
x_1 &= \overline{y_3}\,\overline{y_5} x_4 + y_3 \overline{y_4} x_3 + \overline{y_1} y_2\ .
\end{aligned} \tag{6}$$

Manuscript received January 31, 1997.
This work was supported in part by Seagate Technology.

Table I. Data to codeword mapping for a rate 4/5MTR(2;8) code.

0000↔10000	0100↔00100	1000↔01000	1100↔01100
0001↔00001	0101↔00101	1001↔01001	1101↔01101
0010↔00010	0110↔00110	1010↔01010	1110↔10100
0011↔10001	0111↔10110	1011↔10010	1111↔10101

Multiplication in the logic equations is equivalent to a Boolean *AND*, "+" denotes the *OR* operation, and an overbar indicates inversion. Logic equations given in this paper are minimized with respect to their complexity rather than timing requirements. As a result, some output variables are functions of other outputs as in (4) where x_1 depends on x_4. Evaluation of the functions from top to bottom in the given ordering is sufficient to guarantee that the necessary variables are valid at any given point. Again, note that although the encoder has two states (determined by the last codebit), the decoder can be realized as a block decoder because the substituted codewords are uniquely mapped to data words.

III. Rate 6/7 Code Design

A rate 6/7 code is possible in block form, if the input/output block sizes are quadrupled to 24/28; however, the large block sizes would result in an undesirably large logic circuit. Instead, a 6/7 state-dependent code requiring a sliding block decoder with a modest window size is developed. The procedure described here is specific to MTR code design and is not appropriate for general code design. By focusing solely on the MTR parameters, this method can exploit codeword properties intrinsic to the MTR constraints.

A. Codeword Selection

To begin, the set of n=7 bit codewords that would be valid for an MTR j=2 block code are determined. (At this point, the RLL k constraint is ignored). For n=7, there are 57 valid words, including the all-zeros word; these are referred to as the *basic set*. For m=6, $2^m = 64$ codewords are required. To satisfy this requirement, an *extended set* is formed by combining "110" with the set of length n=4 MTR block codewords. In hexadecimal form, the extended set is {60, 61, 62, 64, 65, 66, 68, 69, 6A}. These additional words bring the codeword count to 66, 2 more than are required. The extended set will not violate the MTR constraint if the preceding codeword ends in a zero. However, a method for preventing a violation when a one precedes an extended codeword is required; otherwise, three consecutive ones could occur. By converting the trailing three bits of the preceding word to "011," the presence of a substitution is indicated. For the problem at hand, two substitutions are needed. These substitutions are referred to as Type I and Type II and are described by

$$\text{....001,110....} \rightarrow \text{....011,001....} \tag{7}$$

and

$$\text{....101,110....} \rightarrow \text{....011,010....} \tag{8}$$

respectively. These substitutions allow the decoder to determine the type by looking forward three bits into the next codeword.

B. Bounding and reduction of the k constraint.

From the basic and extended sets of codewords, a rate 6/7 MTR(2;∞) code could be constructed. Because there are two extra codewords, the all zeros codeword is discarded, which bounds the maximum run of zeros to k=12 (generated by the concatenation of 1000000,0000001). Reduction of the k constraint is accomplished by employing the method of section II. For the 6/7 code, a single rule is required. This Type III substitution is described by

$$\text{....000,000....} \rightarrow \text{....011,000....} \tag{9}$$

Because the "011" pattern is followed by three consecutive zeros, it cannot be confused with the Type I and II substitutions. The resulting worst case zero-runs occur with the pattern pairs "1000000,001...." and "....100,0000001." Thus, the Type III substitution reduces k to give a rate 6/7 MTR(2;8) code.

C. Codeword mapping

In a ROM based lookup table, any mapping of data words to codewords would be acceptable; however, the complexity of the corresponding Boolean logic equations will depend on the chosen mapping. The approach here is to generate a *reasonable* mapping; to find an optimal solution would require definition of the criterion for optimality, which could be area, speed, or some other metric. A divide-and-conquer approach in which the codeword set is partitioned into disjoint subsets containing a number of elements equal to a power of two makes the problem more tractable. Eight sets of eight codewords were used in the mapping chosen for this paper. These partitions are illustrated in Fig. 2 where a dot represents a valid codeword with a hexadecimal value equal to the sum of the row and column labels. The partitions are labeled alphabetically in the order in which they will be used. Partitions A, B, C, E, F and G were chosen so that the three most significant bits in the codeword map to the three most significant bits in the data word. Within these partitions, the codewords are ordered by the four least significant bits as {8,1,2,9,4,5,6,A} so that the bits in seven of the codewords can be mapped directly to the data word. Similarly, the codewords in partitions D and H were ordered in an ad hoc manner that by inspection would yield a reasonable mapping. The basic code mapping (without substitutions) is provided in Table II.

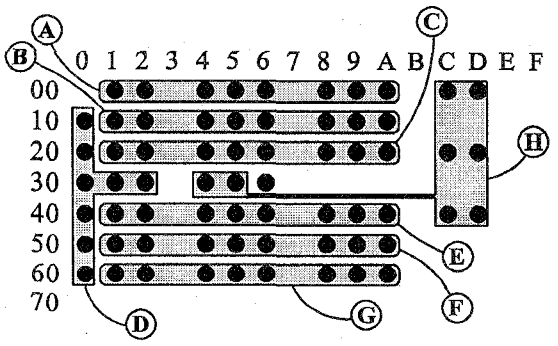

Fig. 2. Partitioning of the n=7 codeword set assuming no substitutions.

Table II. Data to codeword map for a rate 6/7 code without substitutions.

000000↔0001000	010000↔0101000	100000↔1001000	100000↔1101000
000001↔0000001	010001↔0100001	100001↔1000001	100001↔1100001
000010↔0000010	010010↔0100010	100010↔1000010	100010↔1100010
000011↔0001001	010011↔0101001	100011↔1001001	100011↔1101001
000100↔0000100	010100↔0100100	100100↔1000100	100100↔1100100
000101↔0000101	010101↔0100101	100101↔1000101	100101↔1100101
000110↔0000110	010110↔0100110	100110↔1000110	100110↔1100110
000111↔0001010	010111↔0101010	100111↔1001010	100111↔1101010
001000↔0011000	011000↔0110001	101000↔1011000	111000↔0001100
001001↔0010001	011001↔0010000	101001↔1010001	111001↔0001101
001010↔0010010	011010↔0100000	101010↔1010010	111010↔0101100
001011↔0011001	011011↔0110000	101011↔1011001	111011↔0101101
001100↔0010100	011100↔1000000	101100↔1010100	111100↔1001100
001101↔0010101	011101↔1010000	101101↔1010101	111101↔1001101
001110↔0010110	011110↔1100000	101110↔1010110	111110↔0110100
001111↔0011010	011111↔0110010	101111↔1011010	111111↔0110101

Based on the mapping in Table II, the encoder can be implemented as a finite state machine using the present and future data word as input. The corresponding decoder uses a 13-bit wide window that encompasses the present codeword and the three adjacent bits from both the preceding and next codewords. Representative block diagrams for these two components are shown in Fig. 3. On the encoder side, $\boldsymbol{x}$ and $\boldsymbol{w}$ are the present and future data words, respectively. The state variables z and s are delayed versions of the trailing bit from the codeword and the auxiliary variable u. The present codeword at the decoder is denoted $\boldsymbol{y}$ with $\boldsymbol{z}$ and $\boldsymbol{v}$ being the past and present codewords, respectively.

The encoder block implements the set of Boolean logic equations described by

$$
\begin{aligned}
k &= w_1 w_2 w_3 \overline{w_4}\,\overline{w_5} + \overline{w_1}\,\overline{w_2}\,\overline{w_3} \\
m &= \overline{w_1} w_2 w_3 w_4 w_5 \overline{w_6} + w_1 w_2 \overline{w_3} \\
p_1 &= \overline{x_3} + \overline{x_2} \\
p_2 &= \overline{x_4}\,\overline{x_5} + x_6 \\
p_3 &= p_1 + \overline{x_1} \\
p_4 &= \overline{x_5} + \overline{x_4} \\
y_3 &= x_1 x_2 \overline{x_3} z s + x_3 z s \overline{p_4} + x_1 x_3 \overline{p_4} + \overline{x_1} x_3 p_2 + \overline{x_2} x_3 \\
p_5 &= x_3 \overline{y_3} \\
u &= x_6 m \overline{p_3} + x_4 x_6 m \\
y_1 &= x_4 \overline{z} p_5 + x_4 \overline{x_5}\,\overline{p_1} + x_1 \overline{x_3}\,\overline{z} + x_1 \overline{x_2} \\
y_2 &= \overline{x_4}\,\overline{x_6} y_3 \overline{p_1} + x_5 x_6 \overline{p_1} + x_2 \overline{x_3}\,\overline{y_3} + x_5 p_5 + y_3 \overline{p_3} \\
y_4 &= \overline{x_6} p_1 p_2 + x_5 p_1 p_2 + x_1 p_5 \\
y_5 &= x_4 \overline{y_4}\,\overline{u} p_1 + \overline{u}\,\overline{p_3} \\
y_7 &= k p_2 p_4 p_3 + \overline{x_1} k \overline{p_1}\,\overline{p_2} + \overline{x_1}\,\overline{x_6}\,\overline{p_1} p_2 + x_6 p_1 p_4 + x_6 \overline{p_3} \\
y_6 &= \overline{x_1} x_6 y_7 \overline{p_1} + \overline{y_7} p_2 \overline{p_4} + x_5 p_1 \overline{p_2} + \overline{x_6} y_7 p_1 + y_7 \overline{p_2} + m y_7
\end{aligned}
\qquad (10)
$$

and the decoder implements

$$
\begin{aligned}
m_1 &= \overline{z_5} z_6 z_7 (y_2 + y_3) \\
m_2 &= \overline{y_5} y_6 y_7 (v_2 + v_3) \\
q_1 &= m_2 + \overline{y_6} \\
q_2 &= y_6 \overline{y_7} \\
q_3 &= y_7 m_2 \overline{v_2} + y_5 \\
q_4 &= \overline{y_4}\,\overline{y_5}\,\overline{y_6}\,\overline{y_7} + y_4 q_3 \\
x_2 &= \overline{y_4} y_7 \overline{q_1} + q_4 + m_1 + y_2 \\
q_5 &= \overline{x_2} + m_1 + \overline{y_3} \\
x_6 &= y_7 q_5 q_1 + y_6 \overline{v_2}\,\overline{q_5} + \overline{y_5}\,\overline{y_7}\,\overline{q_5} + y_4 q_2 + \overline{q_5}\,\overline{q_1} + y_5 y_7 \\
x_3 &= \overline{y_4} y_7 \overline{x_6} + y_3 \overline{m_1} + q_4 \\
q_6 &= \overline{x_3} + \overline{x_2} \\
x_1 &= y_7 m_2 x_6 \overline{q_6} + y_5 \overline{q_6} + y_1 q_6 + m_1 \overline{x_3} \\
x_4 &= x_6 q_2 + \overline{y_4} q_3 + y_1 \overline{q_6} + m_1 x_3 \\
x_5 &= y_2 q_5 \overline{q_6} + y_2 x_6 \overline{q_6} + y_4 x_6 q_6 + m_1 \overline{q_6} + y_5 \overline{q_5} + q_2 \,.
\end{aligned}
\qquad (11)
$$

D. Efficiency and Performance

Although the rate of codes employing the MTR j=2 constraint is limited to the capacity C=0.8791, the RLL k=8 constraint reduces the capacity to C=0.8760. Thus, the efficiency of the rate 6/7 code described here is $Eff = rate/capacity = 98.6\%$. By comparison, the rate 4/5MTR(2;4) code is 95.5% efficient. A rate of 7/8 is permitted under the MTR constraint; however, such a code would have to be 99.5% efficient even before incorporating the k constraint. Therefore, a rate 6/7 code is likely to be the highest rate, practical

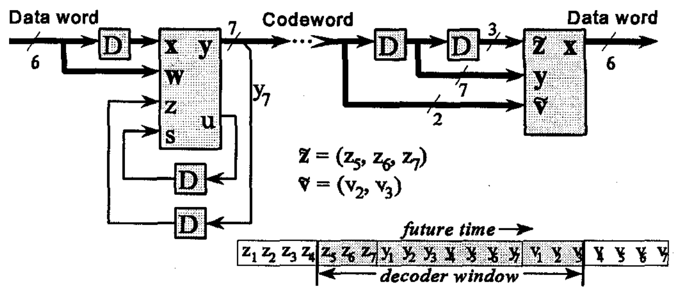

Fig. 3. Block diagram for the 6/7 MTR(2;8) encoder/decoder.

MTR code. This does not imply that a code with less complexity or a smaller RLL k constraint could not be constructed. Although not shown here, a 96% efficient rate 5/6MTR(2;6) code was also constructed using the method presented in this section.

Simulations were performed to quantify the performance gain of the 6/7 code over a rate 16/17 RLL(0,6/6) code [6]. SNR in Fig. 4 is $10\log_{10}(1/\sigma_n^2)$ where the amplitude of the Lorentzian isolated pulse has been normalized to 1 and σ_n^2 is the variance of the additive white Gaussian noise. At a user bit density of 2.5 bits/PW50, the MTR coded E^{2}PRML and FDTS/DF τ=2 detectors show 1dB improvement over the same systems using an RLL code.

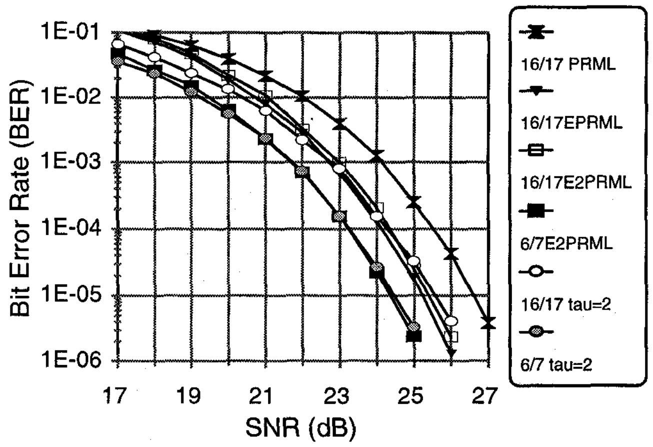

Fig. 4. Performance with a Lorentzian pulse at 2.5 user bits/PW50.

IV. Conclusion

A state-dependent encoding method for reducing the RLL k constraint in systems employing block decoding was presented. This technique was used to aid in the development of a rate 6/7 MTR(2;8) code. With an efficiency of 97.8%, this code is close to the maximum theoretical code rate but can be implemented with reasonable complexity.

References

[1] P. H. Siegel and J. K. Wolf, "Modulation coding for information storage," *IEEE Commun. Mag.*, vol. 29, no. 12, pp. 68-86, Dec. 1991.

[2] J. Moon and B. Brickner, "Maximum transition run codes for data storage systems," *IEEE Trans. Magn.*, vol. 32, no. 5, pp. 3992-3994, Sept. 1996.

[3] J. Moon and B. Brickner, "Method and apparatus for implementing maximum transition run codes," *U.S. Patent Pending No. 60/014,954*, filed April 5, 1996.

[4] G. V. Jacoby, "Ternary 3PM magnetic recording code and system," *IEEE Trans. Magn.*, vol. 17, no. 6, pp. 3326-3328, Nov. 1981.

[5] R. Karabed and P. H. Siegel, "Coding for higher order partial response channels," *SPIE* vol. 2605, pp. 115-126, Oct. 1995.

[6] K. P. Tsang, "Method and apparatus for implementing run length limited codes in partial response channels," *U.S. Patent 5,537,112*, July 16,1996.