

# Efficient dc-Free RLL Codes for Optical Recording

Kees A. Schouhamer Immink, *Fellow, IEEE*, Jin-Yong Kim, Sang-Woon Suh, and Seong Keun Ahn

**Abstract**—We will report on new dc-free runlength-limited codes (DCRLL) intended for the next generation of DVD. The efficiency of the newly developed DCRLL schemes is extremely close to the theoretical maximum, and as a result, significant density gains can be obtained with respect to prior art coding schemes. With a newly developed DCRLL ( $d = 2$ ) code we can achieve a 9% higher overall rate than that of DVD's EFMPlus.

**Index Terms**—Channel capacity, constrained code, dc-free code,  $(d, k)$  sequence, optical recording, runlength-limited (RLL) sequence.

## I. INTRODUCTION

THE design of codes for optical recording is essentially the design of combined *dc-free* and *runlength limited* (DCRLL) codes [1]. Eight to Fourteen Modulation (EFM), invented by Immink and Ogawa in the early Eighties [2], and EFMPlus [3] were adopted as the recording code for the compact disc (CD) and DVD, respectively.

Binary sequences generated by a  $(d, k)$  RLL encoder have at least  $d$  and at most  $k$ ,  $k > d$ , “zero’s” between successive “one’s”. The series of encoded bits is converted, via a modulo-2 integration operation, called *precoding*, to a corresponding modulated signal formed by bit cells having a high or low signal value, a “one” being represented in the modulated signal by a change from a high to a low signal value or vice versa. A “zero” is represented by the lack of change of the modulated signal. Specifically, codes with minimum runlength parameter  $d = 2$  have been widely employed in optical recording, while codes with  $d = 1$  have been proposed for future systems [4]. For that reason will focus our design efforts on efficient RLL codes with minimum runlength parameter  $d = 1$  and  $d = 2$ . Thereafter we will discuss the development of a new DCRLL coding arrangement that employs a highly efficient RLL inner code which is extended by a second coding mechanism, such as, for example, Guided Scrambling [5], used for spectral shaping (and other) purposes. We start with the development of the new RLL codes.

## II. VERY EFFICIENT RLL CODING SCHEMES

Let the integers  $m$  and  $n$  denote the information word length and codeword length, respectively. The maximum rate,  $R = m/n$ , of an RLL code, given values of  $d$  and  $k$ , is called the

Paper approved by V. K. Bhargava, the Editor for Coding and Communication Theory of the IEEE Communications Society. Manuscript received May 2001; revised January 2, 2002. This paper was presented in part at the International Symposium on Information Theory (ISIT), Washington, DC, June 2001.

K. A. S. Immink is with Turing Machines Inc., 3016 DK Rotterdam, The Netherlands (e-mail: immink@turing-machines.com).

J.-Y. Kim, S.-W. Suh, and S. K. Ahn are with the DCT Team, Multi-Media Labs, LG Electronics Inc., Seocho-Gu, Seoul 137-724, Korea.

Digital Object Identifier 10.1109/TCOMM.2003.809752

TABLE I  
CAPACITY  $C(1, k)$  AND  $C(2, k)$  AS A FUNCTION OF  $k$

$k$	$C(1, k)$	$C(2, k)$
7	0.6793	0.5174
8	0.6853	0.5293
9	0.6888	0.5369
10	0.6909	0.5418
11	0.6922	0.5450
$\infty$	0.6942	0.5515

Shannon capacity, and it is denoted by  $C(d, k)$ . Table I tabulates  $C(d = 1, k)$  and  $C(d = 2, k)$  for relevant values of  $k$ . The efficiency of an RLL code is usually measured by a quantity called *code efficiency*,  $\eta$ , defined by

$$\eta = \frac{R}{C}(d, k). \quad (1)$$

For ease of presentation we will first focus on the design of RLL codes with  $d = 1$ . Later we will extend the ideas to the design of codes with  $d = 2$ .

Up till now, small  $(1, k)$  codes with a rate exceeding two-thirds have not been published. There are only two approaches for constructing a  $(1, k)$  RLL code, whose rate is larger than two-thirds. Firstly, we may relax the maximum runlength  $k$  to a value larger than 7. Note that a  $(1, 7)$  code was first put to practical use in the early seventies, and that since the advent of hard-disk drives (HDDs), significant improvements in signal processing for timing recovery circuits have made it possible to employ codes with a much larger maximum runlength  $k$ . Secondly, on top of that we may endeavor to design a more efficient code. The efficiency of the rate  $2/3$ ,  $(1, 7)$  code is  $0.6667/0.6793 = 0.981$ , which reveals that we can gain at most 1.9% in rate by an alternative, more efficient, code redesign. If we fully relax the  $k$  constraint, i.e. set  $k = \infty$ , we can at most gain 3.97% in code rate. In other words, a viable improvement in code rate of a  $(d = 1)$  encoder ranges from 1.9 to 3.97%.

In the sequel of this paper we will show how to create a  $(1, 14)$  code, whose rate is 3.85% better than the traditional rate  $2/3$ ,  $(1, 7)$  code. We start, in the next subsection, with a simple problem, namely finding integers  $m$  and  $n$  that improve the rate,  $2/3$ , of the industry standard code.

### A. Suitable Integers $m$ and $n$ for $d = 1$

We will start with a simple exercise, namely a search for pairs of integers  $m$  and  $n$  that are suitable candidates for a coding rate exceeding  $2/3$ . Obviously, the “best” code is a code with a rate,  $m/n$ , that exactly equals the capacity  $C(d, k)$  for desired values of  $d$  and  $k$ . One is tempted to ask if it is possible to choose the integers  $m$  and  $n$  such that  $m/n = C(d, k)$ . The answer—a sounding no—was given by Ashley and Siegel [6], who

TABLE II  
 INTEGERS  $m$  AND  $n$  SUCH THAT  $2/3 < R = m/n < C(1, \infty)$ . THE  
 QUANTITY  $\eta = R/C(1, \infty)$  EXPRESSES THE CODE EFFICIENCY

$m$	$n$	$1 - \eta$ %
34	49	0.0525
9	13	0.2786
11	16	0.9711
13	19	1.4449
15	22	1.7895
17	25	2.0514

showed that, besides a very few trivial exceptions, the capacity  $C(d, k)$  is an irrational number. Thus, as the rate of a code  $m/n$ , where  $m$  and  $n$  are integers, is rational, the capacity can only be approached.

In order to obtain some feeling if there are many “practical” pairs of such integers  $m$  and  $n$ , we wrote a one-line computer program for searching integers  $m$  and  $n$  that satisfy the inequalities  $2/3 < m/n < C(1, \infty)$ , where for reasons of implementation we set  $n < 50$ . All pairs of integers found are shown in Table II. Surprisingly there are just six  $m$  and  $n$  pairs whose quotient is larger than  $2/3$ .<sup>1</sup> Perusal of the table reveals that the code rate  $m/n = 9/13$  is highly attractive as it is just 0.28% below the Shannon capacity  $C(1, \infty)$ . The next better code of rate  $34/49$  is far less attractive as it is much more complex and adds a minute 0.2% to the density gain with respect to a rate  $9/13$  code. We therefore concentrated our attention on a rate  $9/13$  code. The fact that the rate  $9/13$  is less than capacity does not mean that a code with that rate can be *practically* constructed. In the next subsection, we will show how a rate  $9/13$ , (1,14) code can be created using a new design technique.

### B. Encoder Description

In this section, we will describe a finite-state encoder that generates sequences satisfying the  $d = 1$  constraint (the  $k$  constraint is ignored for a while for ease of presentation). We start with a few definitions. A codeword is a binary string of length  $n$  that satisfies the  $d = 1$  constraint. The set of codewords,  $E$ , is divided into four subsets  $E_{00}$ ,  $E_{01}$ ,  $E_{10}$ , and  $E_{11}$ . The four subsets are characterized as follows. Codewords in  $E_{00}$  start and end with a “0”, codewords in  $E_{01}$  start with a “0” and end with a “1”, etc. The encoder has  $r$  states, which are divided into two state subsets of a first and second type. The encoder has  $r_1$  states of the first type and  $r_2 (= r - r_1)$  states of the second type. The two types of coding states are characterized by the fact that all codewords in the  $r_1$  states of the first type must start with a “0”, while codewords in the  $r_2$  states of the second type are free to start with a “1” or a “0”.

The encoder state-transition rules are now easily described. Codewords that end with a “0”, i.e., codewords in subsets  $E_{00}$  and  $E_{10}$  may enter any of the  $r = r_1 + r_2$  encoder states. Codewords that end with a “1” may only enter the  $r_1$  states of the first type only (and not the states of the second kind). Note that, by definition, the codewords in states of the first type start with a “0”, and codewords in states of the second type may start with a

<sup>1</sup>We omitted trivial pairs, such as 18 and 26, etc., that are multiples of given smaller pairs. This, by the way, does not mean the omitted pairs are irrelevant for a specific code design.

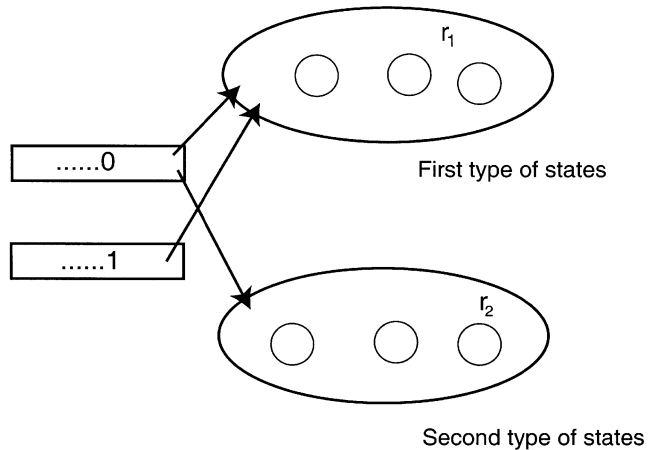


Fig. 1. Codewords that end with a “0” may be followed by codewords in the  $r_1$  states of the first type and the  $r_2$  states of the second type, while words that end with a “1” may only be followed by codewords in the  $r_1$  states of the first type.

“1”, which prohibits that a codeword ending with “1” may enter states of the second type. The encoder concept is schematically represented in Fig. 1. It is essential that the sets of codewords that belong to a given state (of any type) do not have codewords in common (i.e., sets of codewords associated with coding states are disjoint). This attribute implies that any codeword can unambiguously be identified to the state from which it emerged. Then, as we will show, it is possible to assign the same codeword to more than one information word (the miraculous multiplication of codewords). The sliding-block decoder can, by observing both the current and the next codeword—for identifying the next state—, uniquely decide which of the information words was actually transmitted. Codewords in subsets  $E_{10}$  and  $E_{00}$  can, as codewords in these two subsets end with a “0”, be followed by codewords in any of  $r = r_1 + r_2$  states, and can, thus, be assigned  $r = r_1 + r_2$  times to different information words. Similarly, codewords in  $E_{11}$  and  $E_{01}$  can only be followed by the  $r_1$  states of the 1st kind, and can therefore be assigned  $r_1$  times to different information words. Given the above encoder model, we can write down two necessary conditions of such a rate  $m/n$ ,  $d = 1$  code.

Let  $|E_{xy}|$  denote the size of  $E_{xy}$ . Then, following the above arguments, there are at maximum  $r|E_{00}| + r_1|E_{01}|$  codewords leaving the  $r_1$  states of the first type. For a rate  $m/n$  code, there should be at least  $r_1 2^m$  codewords leaving the  $r_1$  states of the first type. Thus we can write down the first condition

$$r|E_{00}| + r_1|E_{01}| \geq r_1 2^m. \quad (2)$$

Similarly, the second condition follows from the fact that there should be a sufficient amount of codewords leaving the  $r$  states. We find

$$r(|E_{00}| + |E_{10}|) + r_1(|E_{01}| + |E_{11}|) \geq r 2^m. \quad (3)$$

Note that the inequalities (2) and (3) are equal to the approximate eigenvector equation, which plays an essential role in a variety of code constructions, such as the state-splitting method [7]. There is, however, quite a difference as the two inequalities above imply a very specific encoder structure (including size), while, in general, the approximate eigenvector merely gives a

TABLE III  
VALUES OF  $r_1$  AND  $r_2$  THAT SATISFY CONDITIONS (2) AND (3)

$r_1$	$r_2$	$r = r_1 + r_2$
3	2	5
5	3	8
6	4	10
7	5	12
8	5	13

TABLE IV  
DISTRIBUTION OF THE VARIOUS SUBSETS AND STATES

group/state	1	2	3	4	5
$E_{00}$	72	72	87	0	2
$E_{01}$	52	52	27	0	13
$E_{10}$	0	0	0	72	72
$E_{11}$	0	0	0	52	37

loose upper bound to the encoder size of the code found by the state-splitting method.

With a small computer we can, given  $m$  and  $n$ , easily find integers  $r$  and  $r_1$  that satisfy the above two conditions. An example of a rate 9/13 code will show the effectiveness of the new construction.

### C. Rate 9/13, ( $d = 1, k$ ) Codes

Assume the construction of a rate 9/13 encoder. Then  $|E_{00}| = 233$ ,  $|E_{10}| = |E_{01}| = 144$ , and  $|E_{11}| = 89$ . Table III shows values of  $r_1$ ,  $r_2$ , and  $r = r_1 + r_2$  that satisfy Conditions (2) and (3). After finding suitable values of  $r_1$  and  $r_2$ , the next step in the code construction is the distribution of the various codewords among the various states. In order to find such a distribution, a trial and error approach has been used. Table IV shows, for  $r_1 = 3$  and  $r_2 = 2$  as an example (Note that the distribution given is not unique, there are many other ways for allocating the codewords to the states), how the codewords in the various subsets can be allocated to the various states. From Table IV, we discern that the subset  $E_{00}$  of size 233 has 72 words in States 1 and 2, 87 words in State 3, and 2 words in State 5. Thus in total:  $72 + 72 + 87 + 2 = 233$ . Similarly, it can be verified that the four row sums equal the number of codewords in each of the four subsets. Codewords that end with a "0", i.e., codewords in  $E_{10}$  and  $E_{00}$ , can be assigned  $r = 5$  times to different information words, while codewords that end with a "1", i.e. codeword in  $E_{11}$  and  $E_{01}$ , can be assigned  $r_1 = 3$  times to different information words. Thus, the total number of information words that can be assigned to the codewords in State 1 is  $5 \times 72 + 3 \times 52 = 516$ . Similarly, it can be verified that from any of the  $r = 5$  encoder states there at least 516 information words that can be assigned to codewords, which shows that the code can accommodate 9-bit information words. An enumeration table such as Table IV suffices to construct a code by assigning codewords to the coding states and source words.

It can be verified with the procedure outlined above that a 13-state encoder of (code) size 520 can be created. The maximum size of any 13-bit  $(1, \infty)$  code equals  $\lfloor 2^{13C(1, \infty)} \rfloor = 521$ , and we therefore conclude that the above code is quite efficient

TABLE V  
INTEGERS  $m$  AND  $n$  SUCH THAT  $8/15 < R = m/n < C(2, \infty)$ . THE QUANTITY  $\eta = R/C(2, \infty)$  EXPRESSES THE CODE EFFICIENCY

$m$	$n$	$1 - \eta$ %
11	20	0.2720
17	31	0.5644
6	11	1.0962
19	35	1.5672
13	24	1.7830
20	37	1.9872
7	13	2.3642
15	28	2.8623
8	15	3.2940

( $\eta = 0.9996$ ) particularly considering that the encoder has a relatively small number, 13, of states. For  $(n = 12)$ -bit codewords, we find that a 13-state encoder achieves the maximum code size, 321, possible. These codes are supposedly the most efficient in existence in terms of relative performance. Such extremely efficient codes could up till now only be constructed with "large" codewords, but as shown here also selected "small" codes can have a rate which is very close to the channel capacity.

As the above code can accommodate more than the required 512 words, surplus 'worst-case' codewords can be deleted for minimizing the  $k$  constraint. After a judicious process of deleting codewords that end or start with "long" runs of '0's, we constructed a 5-state (1,18) code, and a 13-state (1,14) code. Note, in Table I, that the smallest possible  $k$  for a rate 9/13 code equals 12.

A few words are in order about the decoder. A decoder must observe both the current and the upcoming codeword to uniquely decode the encoded sequence of codewords into a sequence of information words. Single channel bit errors can thus lead to at most two decoded  $m$ -bit symbols. The decoder comprises two look-up tables: the next-state look-up table and the data look-up table. The next-state look-up table has the next codeword as an input, and the state to which this word belongs as an output. The data look-up table has the output of the next-state look-up table and the current codeword as an input, and the output of the data look-up table is the decoded information word.

## III. EFFICIENT $d = 2$ CODES

Up till now we have concentrated on the design of efficient  $d = 1$  codes, and as both code parameters,  $d = 1$  and  $d = 2$ , are of great practical interest for optical recording, we will now repeat the exercise for the case  $d = 2$ .

### A. Suitable Integers $m$ and $n$ for $d = 2$

RLL codes with minimum runlength parameter  $d = 2$  have been widely published. Table I tabulates  $C(2, k)$  as a function of  $k$ , and from this table the reader can easily discern the head room available for the design of a code of rate  $R = m/n > 8/15$ . The rate 8/15 is, see Table I, 3.3% below channel capacity  $C(2, \infty)$ . Table V shows values of  $m$  and  $n$ , where  $8/15 \leq m/n < C(2, \infty)$  and  $n < 50$ . The ' $m$  and  $n$ ' pairs are ordered according to their quotient  $m/n$ . Clearly, the quotients 11/20, 6/11, and 7/13 are suitable candidate rates for the creation of

small ( $d = 2$ ) codes. Efficiency-wise speaking the code of rate 17/31 is also attractive, but the code is far too complex for current implementation. Kim [8] has been granted a U.S. Patent on an embodiment of a rate 7/13, (2,25) code, which operates with a single merging bit (3PM principle [9]). In the next subsection, we will describe in detail how the very efficient ( $d = 2$ ) codes with the above mentioned rates can be constructed.

*B. Encoder Description*

In this section we will describe a finite-state encoder that generates sequences that satisfy the  $d = 2$  constraint (note that the  $k$  constraint will be ignored for a while). We start with a few definitions. The encoder is assumed to have  $r$  states, which are divided into three state subsets of states of a first, second, and third type. The state subsets are of size  $r_1, r_2,$  and  $r_3 (= r - r_1 - r_2)$ , respectively. A codeword is a binary string of length  $n$  that satisfies the  $d = 2$  constraint. The set of codewords is divided into nine subsets denoted by  $E_{0000}, E_{0001}, E_{0010}, E_{0011}, E_{0100}$  etc, where the two first symbols of the subset subscript denote the first two symbols of the codeword, and the last two symbols of the subset subscript denote the last two symbols of the codeword. Thus, codewords in  $E_{0000}$  start and end with ‘00’; codewords in  $E_{0001}$  start with ‘00’ and end with a ‘01’, etc. The codewords in the various subsets are distributed over the various states of the three types such that

- codewords in states of the first type start with “00”;
- codewords in states of the second type start with “01” or “00”;
- codewords in states of the third type start with “10,” “01,” or “00”.

The state-transition rules are now easily described. Codewords that end with the string “00,” i.e., codewords in subsets  $E_{0000}, E_{0100},$  and  $E_{1000}$  may enter any of the  $r$  encoder states. Codewords that end with a “10” may not be followed by codewords in a state of the third type. Similarly, codewords that end with a “1” may only be followed by codewords belonging to states of the first type. The state sets of codewords from which a selection is to be made do not have codewords in common. As a result, it is possible to assign the same codeword to different information words. For example, codewords that end with “00,” i.e., codewords in subsets  $E_{0000}, E_{0100},$  and  $E_{1000},$  may enter any state so that these codewords can be assigned  $r = r_1 + r_2 + r_3$  times to different information words. Codewords that end with “10”, i.e. words in subsets  $E_{0010}, E_{0110},$  and  $E_{1010}$  may enter states of the first and second type so that these codewords can be assigned  $(r_1 + r_2)$  times to different information words. Similarly, codewords that end with a “1”, i.e. words in the remaining subsets  $E_{0001}, E_{0101},$  and  $E_{1001}$  can be assigned  $r_1$  times. Given the above encoder model, it is straightforward to write down three conditions for the existence of such a rate  $m/n$  code. Define

$$A_1 = r|E_{0000}| + (r_1 + r_2)|E_{0010}| + r_1|E_{0001}|,$$

$$A_2 = r|E_{0100}| + (r_1 + r_2)|E_{0110}| + r_1|E_{0101}|,$$

and

$$A_3 = r|E_{1000}| + (r_1 + r_2)|E_{1010}| + r_1|E_{1001}|.$$

TABLE VI  
EXAMPLE OF THE DISTRIBUTION OF THE VARIOUS SUBSETS AND STATES OF A RATE 6/11, (2, k) CODE

group/state	1	2	3	4	5	6	7	8	9
$E_{0000}$	6	4	4	4					
$E_{0010}$	1	2	2	4					
$E_{0001}$	1	4	4	1			1		
$E_{1000}$					4	4	5		
$E_{1010}$					2	2	2		
$E_{1001}$					4	4	1		
$E_{0100}$								4	5
$E_{0110}$								2	2
$E_{0101}$								4	2

TABLE VII  
SURVEY OF NEWLY DEVELOPED CODES

$m$	$n$	$d$	$k$	states	$\eta = R/C(d, k)$
11	20	2	23	9	0.9975
7	13	2	11	9	0.9880
6	11	2	15	9	0.9915
9	13	1	14	13	0.9979
9	13	1	18	5	0.9973
11	16	1	10	13	0.9951

Then the conditions are

$$A_1 \geq r_1 2^m, \tag{4}$$

$$A_1 + A_2 \geq (r_1 + r_2) 2^m, \tag{5}$$

$$A_1 + A_2 + A_3 \geq (r_1 + r_2 + r_3) 2^m. \tag{6}$$

In a similar vein as with the ( $d = 1$ ) codes discussed previously, we have experimented with the selection of suitable values of  $m, n, r_1, r_2,$  and  $r_3$ . Many good codes have been found. As a typical example, which is amenable for a hand check, we will show results of a 9-state, (2, k) code of rate 6/11. Given the choice of the code rate, we use a small computer program to find suitable values of  $r_1, r_2,$  and  $r_3$  that satisfy conditions (4)–(6). A possible distribution of the various codeword sets, where we opted for  $r_1 = 4, r_2 = 2,$  and  $r_3 = 3,$  is shown in Table VI. Such a distribution table suffices to construct the code. After judiciously barring worst-case codewords from the coding table, we were able to construct a rate 6/11, (2,15) code. Note, see Table I, that  $k = 11$  is the smallest value possible for the given rate 6/11. Using the above construction methods, we built a 9-state rate 11/20, (2,23) code, whose efficiency is 0.25% less than unity. In addition, we constructed a rate 7/13, (2,11) code, whose efficiency is 1.1% less than unity.

Table VII summarizes the new RLL codes,  $d = 1$  and  $d = 2,$  we have found. As we can see, the efficiency of the majority of the new codes is just a few tenths of a percent below capacity.

The efficiency of the new construction technique can be further exemplified by a second example, where the code size,  $M,$  is not equal to a power of two. The ‘spare’ codewords can be used as alternative channel representations for suppressing the If components. The codeword length equals  $n = 16$ . Table VIII shows the efficiency,  $\eta = R/C(2, \infty),$  as a function of the number of encoder states,  $r$ . It shows that the construction technique yields fine results as the codes obtained reach efficiencies that are only a tenth of a percent below capacity. Note that

TABLE VIII  
CODE SIZE,  $M$ , FOR  $d = 2$ ,  $n = 16$ , AND SELECTED  
VALUES OF THE NUMBER OF ENCODER STATES  $r$

$r_1$	$r_2$	$r_3$	$r$	$M$	$\eta = R/C(2, \infty)$
1	1	1	3	426	0.98995
2	1	2	5	430	0.99148
3	1	3	7	431	0.99186
4	2	3	9	447	0.99782
7	3	5	15	450	0.99891
13	6	9	28	452	0.99963

the maximum size of a code with codewords of length  $n = 16$  equals 453.

At this junction, we have completed the description of the new RLL codes, and we are in the position to describe how we can turn the newly developed RLL codes into DCRLL codes.

#### IV. GUIDED SCRAMBLING

In Guided Scrambling (GS), each information word can be represented by a member of a selection set consisting of  $L = 2^p$ ,  $p \geq 1$ , codewords. The encoder generates the selection set, and the “best” (according to a predefined penalty function) codeword in the selection set is selected for transmission. The penalty function weighs each element of the selection set according to its spectral and other properties such as maximum runlength and so on. The maximum runlength constraint,  $k$ , imposed by the GS penalty function can be made smaller than that of the inner RLL code. Naturally, the GS method cannot fully guarantee the  $k$  constraints, but the probability of occurrence of such vexatious subsequences can be made extremely small. Other (runlength) constraints, such as MTR, can be added to the penalty function if required.

In the preferred GS format,  $m_1$  user bits are multiplexed with  $p$  redundant bits, which are a part of the input of the channel encoder. The  $p$  redundant bits are used to generate a selection set of size  $L = 2^p$ . In the proposed coding format, the channel encoder input comprises  $p$  redundant bits plus  $m_1$  user bits that from a *super* block. The  $p+m_1 = Km$ -bit super block is scrambled using a self-synchronizing (feedback register) scrambler (see for more details [1, Chapter 13]). Then, under the rules of the RLL code, the  $p + m_1 = Km$ -bit scrambled super block is translated into  $Kn$  channel bits. The above scrambling/encoding step is repeated  $2^p$  times for all possible combinations of the  $p$  redundant bits. The encoder transmits the sequence that best matches the channel constraints such as lf content,  $k$ -constraint, etc., discussed above.

The integers  $p$  and  $m_1$  are integers chosen such that

$$Km = p + m_1 \quad (7)$$

where  $K$  is an integer that denotes the number of  $m$ -bit information words in a super block. In a practical environment of a byte-oriented system,  $m_1$  is a multiple of eight, i.e.  $m_1 \bmod 8 = 0$ . Thus the overall rate  $R_o$  of the code is

$$R_o = \frac{m_1}{Kn} = \frac{Km - p}{Kn}. \quad (8)$$

In the next subsection, we will select values of  $p$  and  $m_1$ , and show results of computer simulations.

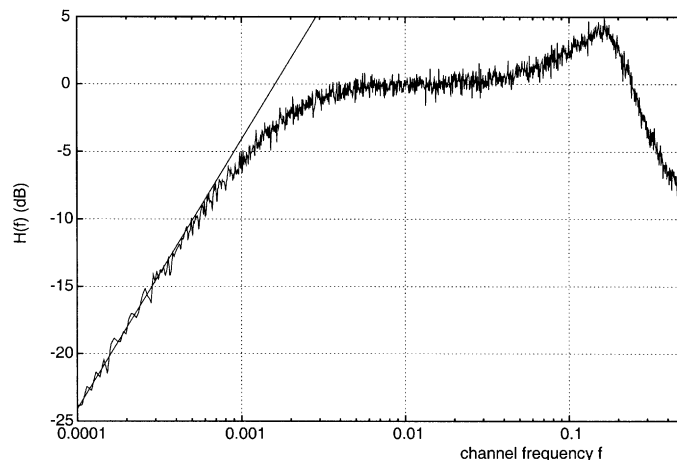


Fig. 2. Simulation results of a PSD function of a ( $d = 1$ ,  $k = 10$ ) code of overall rate  $R_o = 0.68376$ . The spectrum was computed on the basis of 10 million channel bits. The straight line is a “best fit” estimate of the low-frequency part of the spectrum. We simply discern that  $H(f = 10^{-4}) = -24.3$  dB.

#### A. Results and Comparison with Prior Art Methods

We have written a computer program to simulate the performance of GS in conjunction with the newly developed RLL codes. The power spectral density (PSD),  $H(f)$ , and other relevant characteristics can easily be computed.

As a typical example, we will show results obtained with the rate 9/13, (1,14) RLL code. Fig. 2 shows the spectrum,  $H(f)$ , versus (channel) frequency,  $f$ , for  $p = 5$ ,  $k = 10$ , and  $K = 45$ . The overall code rate is  $R_o = (Km - p)/Kn = (45 \times 9 - 5)/(45 \times 13) = 0.68376$ . Note that the overall code is byte oriented as  $Km - p = 400$  is a multiple of eight. The scrambler polynomial used in our simulations is  $1 + x + x^5$ . In the runlength penalty function, we set the maximum “zero” runlength to  $k = 10$ , which means that the code essentially behaves as a ( $d = 1$ ,  $k = 10$ ) code. The spectrum,  $H(f)$ , versus frequency  $f$  has a parabolic shape in the low-frequency range, which shows as a straight line as a result of the logarithmic frequency axis used. Using a computer simulation of the encoding process, we compute the PSD of an encoded sequence. Then using a best-fit (LMS) estimation technique involving the low-frequency components of the spectrum, we derive an estimate of the low-frequency performance. For example, in Fig. 2, we estimate that  $H(10^{-4}) \approx -24.3$  dB. In similar vein, we estimated  $H(10^{-4})$  as a function of the overall rate,  $R_o$ . Results are shown in Fig. 3 for  $p = 5$  and  $p = 8$ . The maximum runlength  $k$  in the GS penalty function was set to  $k = 10$ . Similar curves can be plotted for other values of  $k$ , and, obviously, the more relaxed the  $k$  constraint, the more lf suppression. In order to compare our results with the maximum theoretical performance of DCRLL codes, we invoked the algorithms derived by Braun and Janssen [10], which compute the *maxentropic* performance of ( $d, k$ ) codes. The maxentropic performance sets a theoretical limit to the performance of any implemented DCRLL code. Fig. 3 shows that the implemented codes operate very close to the best theoretical performance. For  $p = 5$  the implemented codes are 2–3 dB, (for  $p = 8$ , 1–2 dB) below the theoretical ceiling. As a further

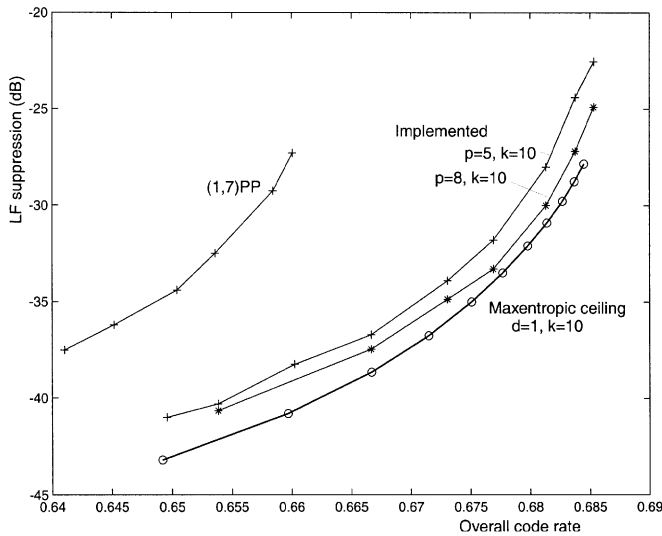


Fig. 3. The two upper curves show the lf suppression,  $H(10^{-4})$ , as a function of the overall code rate  $R_o$ . The upper curve shows results for  $p = 5$ , and the lower curve for  $p = 8$ . The maximum imposed runlength for both cases is  $k = 10$ . As a comparison we plotted the theoretical ceiling,  $H_{\min}(10^{-4})$ , of maxentropic ( $d = 1, k = 10$ ) sequences [1]. The curve denoted by (1,7)PP gives results of a prior art code [4].

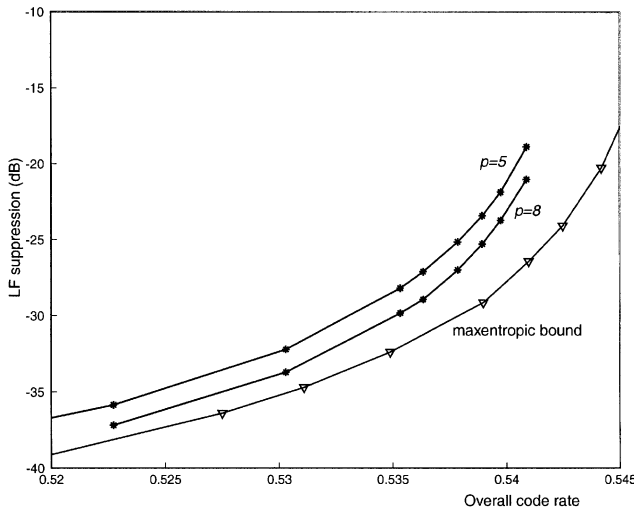


Fig. 4. The two upper curves show the lf suppression,  $H(10^{-4})$ , as a function of the overall rate  $R_o$ . The upper curve is for  $p = 5$ , and the lower curve is for  $p = 8$ . The maximum imposed runlength for both cases is  $k = 12$ . As a comparison we plotted the theoretical ceiling,  $H_{\min}(10^{-4})$ , of maxentropic ( $d = 2, k = 12$ ) sequences.

comparison we plotted the performance of a prior art rate 2/3, (1,7) code [4], which is extended with dc-control bits on data sequence level.

We proceed with a second example. Fig. 4 shows the lf spectral performance of the rate 6/11, (2,15) code in conjunction with Guided Scrambling. Results are given for  $p = 5$  and  $p = 8$ . As reported in the above  $d = 1$  case, the combination of an effi-

cient RLL code and GS works quite satisfactorily as only 2–3 dB can be gained with respect to the theoretical ceiling.

### V. CONCLUSIONS

We have studied the construction of extremely efficient RLL codes. We have shown that there is a very limited number of pairs of integers  $m$  and  $n$ , whose quotient  $m/n$  form a suitable coding rate for ( $d = 1$ ) and ( $d = 2$ ) RLL codes that are more efficient than prior art codes. Suitable values for the rate of a ( $d = 1$ ) code are 9/13 and 11/16, while for ( $d = 2$ ) codes we have 11/20, 7/13, and 6/11.

We have disclosed a novel technique for designing very efficient RLL codes, whose rate is only a few tenths below capacity. For example, we have constructed a 13-state rate 9/13, (1,14) RLL code, whose rate is only 0.2% below channel capacity  $C(1, 14)$ . In addition, we have constructed a new rate 6/11, (2,15) code, a rate 11/20, (2,23) code, and a rate 7/13, (2,11) code.

Results of computer simulations have shown that the arrangement of the newly developed RLL codes in conjunction with Guided Scrambling (GS) is extremely efficient in terms of overall rate and spectral performance as we have shown that only a few dB in spectral performance can be gained with respect to the theoretical ceiling. With a newly developed  $d = 2$  code we achieved a 9% higher overall rate than that of DVD's EFMPlus.

### REFERENCES

- [1] K. A. S. Immink, *Codes for Mass Data Storage Systems*. Geldrop, The Netherlands: Shannon Foundation, 1999.
- [2] K. A. S. Immink and H. Ogawa, "Method for Encoding Binary Data," U.S. Patent 4 501 000, Feb. 19, 1985.
- [3] K. A. S. Immink, "EFMPlus: the coding format of the multimedia compact disc," *IEEE Trans. Consumer Electron.*, vol. 41, pp. 491–497, Aug. 1995.
- [4] T. Narahara, S. Kobayashi, M. Hattori, Y. Shimpuku, G. van den Enden, J. A. Kahlman, M. van Dijk, and R. Woudenberg, "Optical disc system for digital video recording," in *Proc. Joint Int. Symp. on Optical Memory and Optical Data Storage*, Hawaii, July 11–15, 1999.
- [5] I. J. Fair, W. D. Gover, W. A. Krzymien, and R. I. MacDonald, "Guided scrambling: a new line coding technique for high bit rate fiber optic transmission systems," *IEEE Trans. Commun.*, vol. 39, pp. 289–297, Feb. 1991.
- [6] J. J. Ashley and P. H. Siegel, "A note on the shannon capacity of run-length-limited codes," *IEEE Trans. Inform. Theory*, vol. IT-33, pp. 601–605, July 1987.
- [7] R. L. Adler, D. Coppersmith, and M. Hassner, "Algorithms for sliding block codes. An application of symbolic dynamics to information theory," *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 5–22, Jan. 1983.
- [8] M. J. Kim, "7/13 Channel coding and decoding method using RLL(2,25) code," U.S. Patent 6 188 336, Feb. 13, 2001.
- [9] G. V. Jacoby, "Method and apparatus for encoding and recovering binary digital data," U.S. Patent 4 323 931, Apr. 6, 1982.
- [10] V. Braun and A. J. E. M. Janssen, "On the low-frequency suppression performance of DC-free runlength-limited modulation codes," *IEEE Trans. Consumer Electron.*, vol. 42, no. 4, pp. 939–945, Nov. 1996.