On Runlength-Limited Coding with DC Control

Ron M. Roth*

Abstract

Constructions are presented of finite-state encoders for certain (d, k)-RLL constraints with DC control. In particular, an example is provided for a rate $8 : 16$ encoder for the $(2, 10)$-RLL constraint that requires no look-ahead in decoding, thus performing favorably compared to the EFMPlus code used in the DVD standard.

Keywords: Runlength-limited coding, Optical recording, DC control, EFM code, EFMPlus code.

*Computer Science Department, Technion, Haifa 32000, Israel. E-mail: ronny@cs.technion.ac.il. This work was done in part while visiting Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA, U.S.A., and in part at Hewlett-Packard Laboratories Israel, Haifa, Israel.

1 Introduction

In optical and magnetic recording systems, the bit stream that is written into the device must satisfy certain constraints. The most common family of such constraints appears to be that of the (d, k)-runlength-limited (RLL) constraints, where the run of 0's between consecutive 1's in the bit stream must have length at least d and no more than k for prescribed parameters d and k. For example, bit streams recorded in the compact disk and the DVD satisfy the $(2, 10)$-RLL constraint [4], [5], [6]. The set of all sequences satisfying a given (d, k)-RLL constraint can be described by reading the labels off of paths in the labeled directed graph as shown in Figure 1.

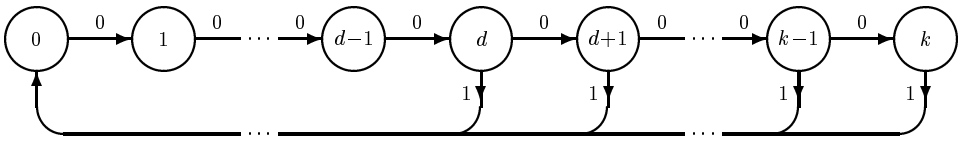

Figure 1: Graph presentation of a (d, k)-RLL constraint.

Given a binary sequence $z_1 z_2 z_3 \cdots$ that satisfies a given (d, k)-RLL constraint, the respective NRZI sequence over the bipolar alphabet $\{+1, -1\}$ is given by $w_1 w_2 w_3 \cdots$ where $w_h = \prod_{i=1}^{h} (-1)^{z_i}$. In addition to satisfying RLL constraints, recording applications typically require also the suppression of the DC component of the recorded NRZI signal; namely, the spectrum (i.e., power spectral density) of the ensemble of NRZI sequences should take small values in the neighborhood of zero frequency [4, Ch. 2], [8], [9], [10]. Another parameter that measures the DC suppression is the running sum variation (in short, RSV), which is defined as the expected value of $(1/\ell) \sum_{h=1}^{\ell} w_h^2$ when $\ell \to \infty$. Clearly, a smaller RSV indicates a better suppression of the DC component.

In order to satisfy a given constraint, a user-provided unconstrained data stream should be encoded; namely, it needs to undergo a uniquely-decodable (or lossless) mapping into a constrained sequence. Encoders for constrained data usually take the form of a finite-state machine. A rate $p : q$ finite-state encoder accepts an input block of p bits and generates a q-bit codeword depending on the input block and the current state of the encoder. The sequences obtained by concatenating the generated q-bit codewords satisfy the constraint.

Each encoder must have a decoder that recovers the input stream from the constrained sequence generated by the encoder. The class of *sliding-block decoders* has the advantage of limiting error propagation. An (m, a)-sliding-block decoder reconstructs an input p-bit block that corresponds to given received q-bit codeword on the basis of the local context of that codeword in the received sequence: the codeword itself, as well as m preceding codewords and a upcoming codewords. The parameter m is called the encoder *memory*

and a is referred to as the *anticipation*. Thus, a single error at the input to a sliding-block decoder can only affect the decoding of at most m+a+1 consecutive codewords.

In addition to having sliding-block decoders, it is desirable that the code have the highest rate possible. The rate of any encoder for a given constraint is bounded from above by the *Shannon capacity* of that constraint. Table 5.4 in [4, p. 91] lists the Shannon capacities of several (d, k)-RLL constraints.

This work presents constructions of finite-state encoders for certain (d, k)-RLL constraints with DC control. Two examples are provided. The first example, presented in Section 2, is a rate 8:16 encoder for the $(2, 10)$-RLL constraint, and the second example, presented in Section 3, is a rate 8:15 encoder for the $(2, 12)$-RLL constraint. Both encoders are sliding-block decodable with zero memory, and the $(2, 10)$-RLL encoder has zero anticipation. DC control is achieved by letting as many input blocks as possible be encoded into two possible codewords; the parity of number of 1's is different in these two codewords and, so, the respective NRZI sequences end with a different polarity, thus reversing the polarity of subsequent codewords. Furthermore, both codewords lead to the same state, thus allowing local replacement of a codeword by its alternate without affecting subsequent codewords. The spectral properties of the new $(2, 10)$-RLL encoder are compared in Section 2 with two existing encoders for the same constraint at the same rate; one of these encoders is the EFMPlus code which is part of the DVD standard. Design considerations are summarized in Section 4.

One architectural feature of the design approach here is having a single compact encoder table that contains the codeword tables of all states in an *overlapping* manner: recognizing that sets of codewords that correspond to different states can intersect, the overlapping encoding table uses the very same entry in the table for each intersecting codeword, regardless of the state from which this codeword is generated. Furthermore, the specific ordering of the codewords in the table provides an easy mechanism of DC control.

2 (2,10)-RLL encoder with DC control

The Shannon capacity of the $(2, 10)$-RLL constraint is approximately 0.5418 [4, p. 91]. We describe here a four-state encoder for this constraint at rate 8:16.

Our encoder consists of a table of 556 codewords, each 16 bits long. Table 4 lists the codewords in hexadecimal form. At each encoding step, the encoder can be in one of the following states: S0, S1, S2-5, or S6-8. Each state is associated with a range of runlengths which is reflected in the state name. E.g., state S6-8 is associated with the runlengths 6, 7, and 8. Encoding is carried out as follows: given an input byte b, a ten-bit address is formed by prefixing b with two bits. This two-bit prefix depends on how the

value of b (as an integer) compares with two thresholds, T1 and T2. These thresholds, in turn, depend on the current state of the encoder. The table of threshold values and the table of prefixes are presented in Table 1 (the notation ϕ stands here for the "don't care" sign).

State	Thresholds (decimal)		Prefixes (binary)		
	T1	T2	$0 \leq b <$ T1	T1 $\leq b <$ T2	T2 $\leq b < 256$
S0	000	001	ϕ	01 or 00	00
S1	004	123	01	01 or 00	00
S2-5	044	050	10 or 01	01	01 or 00
S6-8	044	174	10 or 01	01	01 or 00

Table 1: Thresholds and prefixes for the $(2, 10)$-RLL encoder.

For example, if the current state is S6-8 and the input byte is 049 (decimal), then the address will be $256 + 049 = 305$.

The output codeword is the entry in Table 4 at the computed address and the next encoder state is determined by the last runlength of the generated codeword.

In the example, the output codeword is 0000001001000010 (0242 in hexadecimal notation) and the next encoder state is S1.

There are cases where more than one prefix is possible, resulting in two different codeword candidates. To allow DC control, the encoder table is designed so that—to the largest extent possible—those codeword candidates will have different parity (odd/even) of number of 1's, thereby inducing different parity of sign changes (see [7] for using a similar idea in an unconstrained setting). Furthermore, both codeword candidates lead the encoder to the same state and, therefore, replacement of a codeword with its alternate can be done within an output stream without affecting preceding or following codewords. The decoder can recover the input byte regardless of the specific codeword candidate that was chosen.

For example, if the current state is S1 and the input is 70 (decimal), then the output codeword can be either 0000100000010001 or 0100000010010001. Both codewords lead to state S0.

A schematic diagram of the encoder is shown in Figure 2. Assuming independent and uniformly distributed input, the expected percentage of input bytes within an input sequence for which two codeword candidates exist is 49.7% (this number is computed by first finding the stationary probability of being at each encoder state; see the discussion towards the end of Section 4).

Decoding is carried out as follows. First, an address is found of a table entry that contains the received codeword. The input byte is then obtained by truncating the two

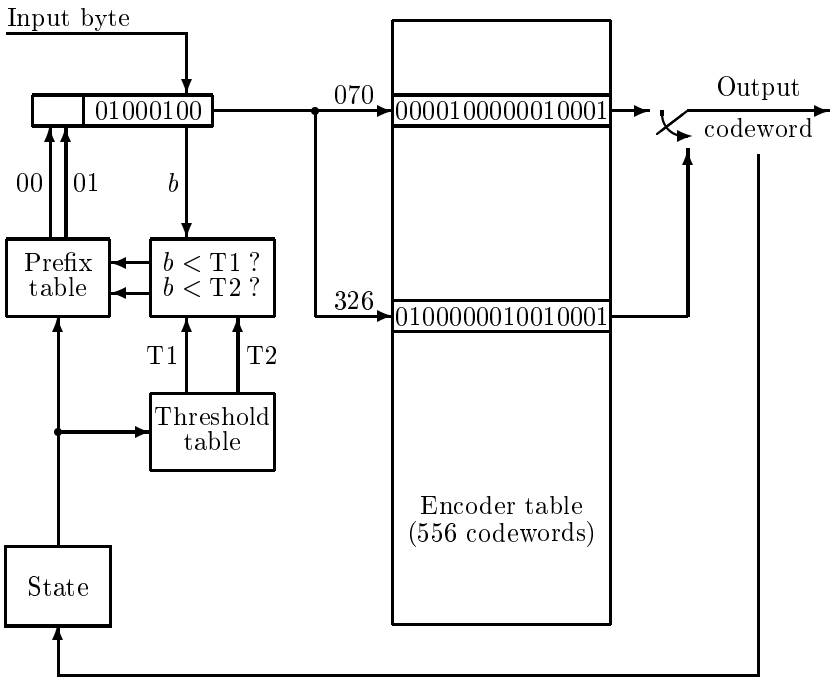

Figure 2: Schematic diagram of the $(2, 10)$-RLL encoder.

most significant bits (m.s.b.'s) of the address where the codeword is located. Recalling the terms defined in Section 1, this is a $(0, 0)$-sliding-block decoder, i.e., only the current codeword is required for the decoding of the current input block.

Next we compare the encoder presented here with two encoders for the $(2, 10)$-RLL constraint at rate 8:16 that are presented in [5]. The first encoder therein, referred to as the modified EFM code, is $(0, 0)$-sliding-block decodable and uses a table of 256 codewords, each 14 bits long. This code differs from the conventional EFM code used in the compact disk in that only two merging bit are inserted between codewords.

The spectra of the encoder presented here and the modified EFM code were obtained by simulation and are shown in Figure 3. In both encoders, the decision between a codeword and its alternate is done on-line by looking ahead at two upcoming input bytes and minimizing the absolute value of the sum of the bipolar values in the NRZI sequence that corresponds to the output binary sequence. The axes in the figure are scaled to match the respective figures in [5]: the length of each input bit is assumed to be two time units, and the NRZI sequence is normalized so that its amplitude is $1/\sqrt{2}$. By comparing the two curves in Figure 3, we conclude that in the low-frequency range, the spectrum of the encoder presented here is 5dB lower than that of the modified EFM code. The improvement of the new encoder over the modified EFM code can be seen also through the RSV values: simulation results show that the RSV of the new encoder—while looking

ahead at two input bytes—is 24.3, whereas the RSV of the modified EFM code (with the same encoding look-ahead) is 36.5.

The second encoder presented in [5] is the EFMPlus code, which is part of the DVD standard. The encoding table of this encoder consists of $1,376$ codewords, each 16 bits long. Unlike the modified EFM code or the new encoder presented here, the EFMPlus code is $(0,1)$-sliding-block decodable, as the decoding of an input byte requires the knowledge of two bits in the upcoming received codeword in addition to knowing the current codeword. On the other hand, when comparing the spectrum of our encoder with the respective curve of the EFMPlus code (Figure 4 in [5]), the two curves look very much alike. In particular, both spectra take the value of approximately -15dB for the normalized frequency value $f = 10^{-3}$.

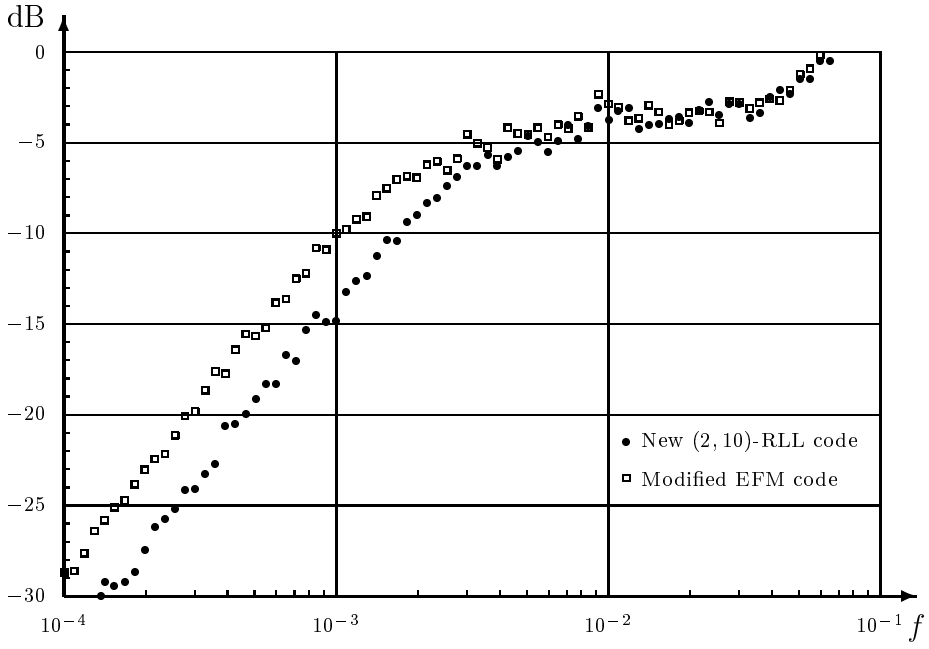

Figure 3: Spectra of the new $(2,10)$-RLL encoder and the modified EFM code with two merging bits. To maintain DC control, each encoder looks ahead at two upcoming input bytes.

3 (2,12)-RLL encoder with DC control

The Shannon capacity of the $(2,12)$-RLL constraint is approximately 0.5471 [4, p. 91]. We next describe an eight-state encoder for this constraint at rate 8:15.

Table 5 lists the encoder table which consists of 551 codewords, each 15 bits long. At each encoding step, the encoder can be in one of the following states: S0, S1, S2a, S3a, S4a, S5-6a, S2-6b, or S7-8. The specific state depends on the last runlength of the previous codeword and the l.s.b. of the previous input byte, as described in Table 2. As

Last runlength of previous codeword	L.s.b. of previous input byte	Current state
0	ϕ	S0
1	ϕ	S1
2	0	S2a
3	0	S3a
4	0	S4a
5 or 6	0	S5-6a
2, 3, 4, 5, or 6	1	S2-6b
7, 8	ϕ	S7-8

Table 2: States in the $(2, 12)$-RLL encoder.

in the $(2, 10)$-RLL encoder of Section 2, encoding is carried out by comparing the input byte b to threshold values and adding suitable two-bit prefixes. The threshold values and the prefixes are specified in Table 3. The output codeword is the entry in Table 5 at the

State	Thresholds (decimal)		Prefixes (binary)		
	T1	T2	$0 \leq b <$ T1	T1 $\leq b <$ T2	T2 $\leq b <$ 256
S0	000	000	ϕ	ϕ	00
S1	002	120	01	01 or 00	00
S2a	005	036	01	01 or 00	00
S3a	009	036	01	01 or 00	00
S4a	015	036	01	01 or 00	00
S5-6a	036	036	01	01 or 00	00
S2-6b	036	039	10	10 or 01	01
S7-8	039	080	10 or 01	01	01 or 00

Table 3: Thresholds and prefixes in the $(2, 12)$-RLL encoder.

computed address. The next encoder state is determined by Table 2.

Like in the previous encoder, DC control is attained by allowing certain input bytes to have two different codeword candidates with different parity of number of 1's (that lead to the same state). Assuming that the input is independent and uniformly distributed, the expected percentage of such input bytes within an input sequence is 12.2%.

The resulting encoder is $(0, 1)$-sliding-block decodable, as demonstrated by the following decoding scheme. First, an address is found of a table entry that contains the received codeword. In case the codeword ends with a runlength of 0, 1, 7, or 8, then

that codeword appears only once in the table. In this case, the input byte is obtained by truncating the two m.s.b.'s of the address where the codeword is located. In case the codeword ends with a runlength of 2, 3, 4, 5, or 6, then it appears twice in the encoder table in two adjacent locations. By truncating the two m.s.b.'s of the address the input byte is determined except for its l.s.b.: the latter bit decodes to 0 if the *next* received codeword is located in the encoder table at address smaller than 292 (this address boundary is marked in Table 5); otherwise, the l.s.b. is 1.

We can redirect into state S5-6 all the codewords that lead to states S2a, S3a, S4a, and S7-8, and delete those four states. By doing this, we obtain a four-state encoder for the $(2, 12)$-RLL constraint; yet, the expected percentage of input bytes that have two codeword candidates reduces to 7.6%.

The spectra of the eight-state and four-state encoders for the $(2, 12)$-RLL constraint were obtained by simulation and are shown in Figure 4, where the encoding look-ahead and the scaling of the axes are the same as in Figure 4. Compared with the encoder in Section 2, the reduction of the DC component here can be observed at significantly smaller frequencies: this is a result of the fact that the percentage of input bytes that have two codeword candidates is much smaller here. The effect of that percentage is very much apparent when comparing the performance of the four-state $(2, 12)$-RLL encoder with the eight-state encoder in the low frequency range: the spectrum of the latter encoder is approximately 6dB lower.

4 Code design

In this section, we outline the principles that guided the design of the coding scheme presented in Section 3. The design does involve some heuristics, but still can be applied to obtain similar coding schemes for other certain (d, k)-RLL constraints; in particular, similar principles guided also the design of the encoder in Section 2.

4.1 State merging and state splitting

Let G denote the graph presentation of the $(2, 12)$-RLL constraint which is obtained from Figure 1 by substituting $d = 2$ and $k = 12$. The adjacency matrix of G, denoted A_G, is a 13×13 matrix whose rows and columns are indexed by the states (vertices) of G and the (u, v)th entry of A_G, denoted $(A_G)_{u,v}$, equals the number of edges from state u to state v in G. The graph G^q is obtained from G in the following manner. The set of states of G^q is the same as that of G, and each edge in G^q corresponds to a path of length q in G, beginning at the initial state of the first edge of the path in G and ending at the terminal state of the last edge of the path in G. The label of an edge in G^q is the word

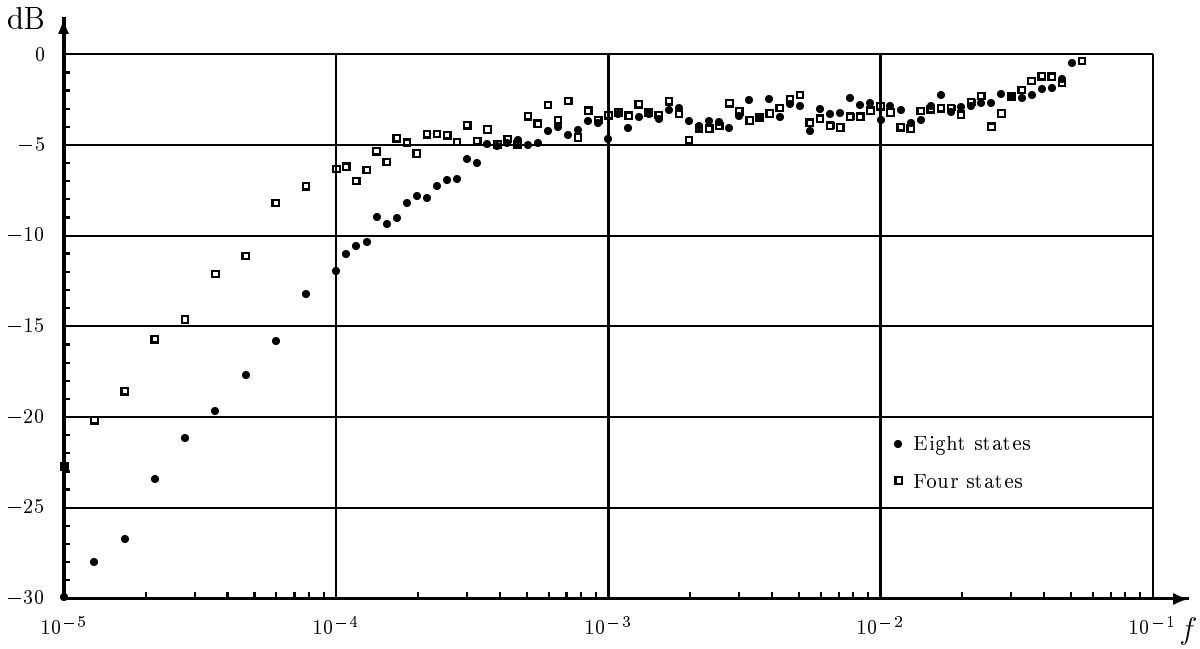

Figure 4: Spectra of the new rate 8:15 eight-state and four-state $(2, 12)$-RLL encoders, with encoding look-ahead of two bytes.

generated by the respective path in G. The adjacency matrix of G^q equals A_G^q.

To obtain a rate 8:15 finite-state encoder, we first invoke the technique of *state splitting* of Adler, Coppersmith, and Hassner [1]; see also a summary in [11] and [12]. We start by computing an $(A_G^{15}, 2^8)$-*approximate eigenvector*, which is a nonnegative nonzero integer vector $\mathbf{x} = [\, x_u \,]_{u=0}^{12}$ such that $A_G^{15}\mathbf{x} \geq 2^8\mathbf{x}$, where the inequality holds component by component. The entry x_u is referred to as the *weight* of state u. An $(A_G^{15}, 2^8)$-approximate eigenvector with components adding up to the smallest sum possible is given by

$$\mathbf{x} \;=\; [\, 1\; 1\; 2\; 2\; 2\; 2\; 2\; 1\; 1\; 0\; 0\; 0\; 0 \,]^T \;. \tag{1}$$

In particular, it can be verified that any $(A_G^{15}, 2^8)$-approximate eigenvector must contain a component which is greater than 1 [11, Section 3.1.4]. Combining this with Proposition 3.34 in [11], it follows that any sliding-block decoder of any finite-state encoder for the $(2, 12)$-RLL constraint at rate 8:15 must have anticipation $\mathsf{a} \geq 1$. The encoder we construct is $(0, 1)$-sliding-block decodable; furthermore, the l.s.b. of the current input byte is the only bit whose decoding may require the knowledge of the next received codeword, in addition to the current codeword. (We remark that in the case of the $(2, 10)$-RLL constraint, there is an approximate eigenvector which is a 0–1 vector. Indeed, for this constraint we presented a $(0, 0)$-sliding-block decoder in Section 2; see also [3].)

States 9 through 12 have zero weight and therefore can be removed from G^{15} with all

their incoming and outgoing edges. States 7 and 8 have the same weight and, in addition, all words that can be generated by paths beginning at state 8 in G^{15} can also be generated starting at state 7. Therefore, state 7 can be *merged* into state 8 by redirecting edges incoming to state 7 so that they terminate in state 8, thereby allowing the deletion of state 7 (see [11], [12]). Similarly, states 2 through 5 can be merged into state 6; yet, to obtain the encoder of Section 3 we merge only state 5 into state 6. That is, *avoiding* state merging in certain cases is the mechanism by which we get more codewords that have alternates and, thus, more DC control. (The same applies to the encoder in Section 2: we could merge states 2 through 7 into state 8 to obtain a $(2, 10)$-RLL encoder with three states; however, to increase the DC control, we only merged states 2 through 4 into state 5 and states 6 and 7 into state 8, resulting in four states altogether).

After merging and deleting states, we obtain a graph H with the following seven states: S0, S1, S2, S3, S4, S5-6, and S7-8. States S0 through S4 correspond to states 0 through 4 in G^{15}, state S5-6 is obtained by merging state 5 into state 6 in G^{15}, and state S7-8 is obtained by merging state 7 into state 8 in G^{15}. Note that the label (codeword) of each edge in H uniquely determines the terminal state of that edge; in fact, the last runlength in that codeword identifies that terminal state. The $(A_G^{15}, 2^8)$-approximate eigenvector $\mathbf{x}$ in (1) reduces to the $(A_H, 2^8)$-approximate eigenvector

$$\mathbf{y} = [\, y_u \,]_u \;=\; [\, 1\ 1\ 2\ 2\ 2\ 2\ 1 \,]^T \,.$$

Next we *split* states in H with weight greater than 1. When a state u is split, two or more descendant states are formed. The incoming edges to u are *duplicated* into each of the descendant states, whereas the outgoing edges from u are *partitioned* among the descendant states. In the graph H, there are four states that need to be split, namely, states S2, S3, S4, and S5-6. It can be verified that each of these states can be split into two states, resulting in descendant states each having at least 2^8 outgoing edges. In fact, after splitting, most states will have more than 2^8 outgoing edges, which will allow having alternate codewords and hence DC control.

We point out that the EFMPlus code in [5] for the $(2, 10)$-RLL constraint can be obtained in a similar manner by one state splitting. So, state splitting can lead to a gain in DC control. On the other hand, our strategy to obtain DC control is *limiting state merging*. We did invoke state splitting in the $(2, 12)$-RLL case not as a means to gain DC control, but simply because it was required to obtain a $(2, 12)$-RLL encoder even if DC were to be ignored. The application of state splitting is the reason for having anticipation greater than 0 in this case. On the other hand, we did not have to (and therefore we did not) apply state splitting to obtain the $(2, 10)$-RLL encoder in Section 2.

4.2 Setting up the table

Straightforward splitting of the four states with weight 2 in H may result in 11 encoder states. To reduce the number of encoder states and to obtain a compact codeword table, we define a certain order on the outgoing edges (or rather, their labels) from each state in H and construct the encoding table based on that order.

Figure 5 presents the structure of the encoding table. The rectangle to the right represents a table of 551 codewords, divided into *runlength intervals*: each runlength interval contains the codewords whose first runlength falls within a given interval of values; e.g., all codewords that start with a runlength between 2 and 4 belong to one runlength interval. Codewords *ending* with a runlength in the range 2–6 are written twice in the table, in two consecutive places; indeed, those codewords correspond to edges that were duplicated due to the splitting of states S2, S3, S4, and S5-6.

The two-sided vertical arrows to the left mark the locations of codewords that can be generated from each state in H. The number of codewords (counting multiplicity) that correspond to each state are written in parentheses under the state name; for every state u, this number is equal to $\sum_v (A_H)_{u,v} y_v$. Note that the runlengths were clustered into runlength intervals in the table in such a way that the interval boundaries separate between segments of the table that correspond to different states in H.

Figure 5 also shows a splitting of states S2, S3, S4, and S5-6 which is marked by the dashed line: In each one of those states, the outgoing edges are partitioned so that edges labeled by codewords that are located at addresses < 292 belong to a descendant state that inherits the name of the parent state with a suffix "a". The rest of the edges belong to the other descendant state that carries the suffix "b". The number 292 was chosen so that state S5-6a will have at least 2^8 outgoing edges. Note that a duplicated codeword in the table corresponds to an incoming edge to a state that was split.

Observe that all codewords that can be generated from a given state form a *contiguous* segment of the table. In fact, it follows from a result by Franaszek [2] that this can always be done for any (d, k)-RLL constraint. Segments of the table that correspond to different states can overlap, thus resulting in a compact encoding table. Also, states S2b, S3b, S4b, and S5-6b are *equivalent* in that the sets of codeword sequences that can be generated from each one of those states are the same. Therefore, we can combine those states into one state which we call S2-6b.

The current table already implies a coding scheme as follows. Encoding is carried out by adding a two-bit prefix to the input byte. This two-bit prefix is chosen so that the resulting address falls within the address range of the (contiguous) segment of the table that corresponds to the current state, as determined by Figure 5; in fact, in many cases more than one prefix is possible. Finding the right prefix can be translated in a straightforward manner into threshold comparison, leading to Table 3.

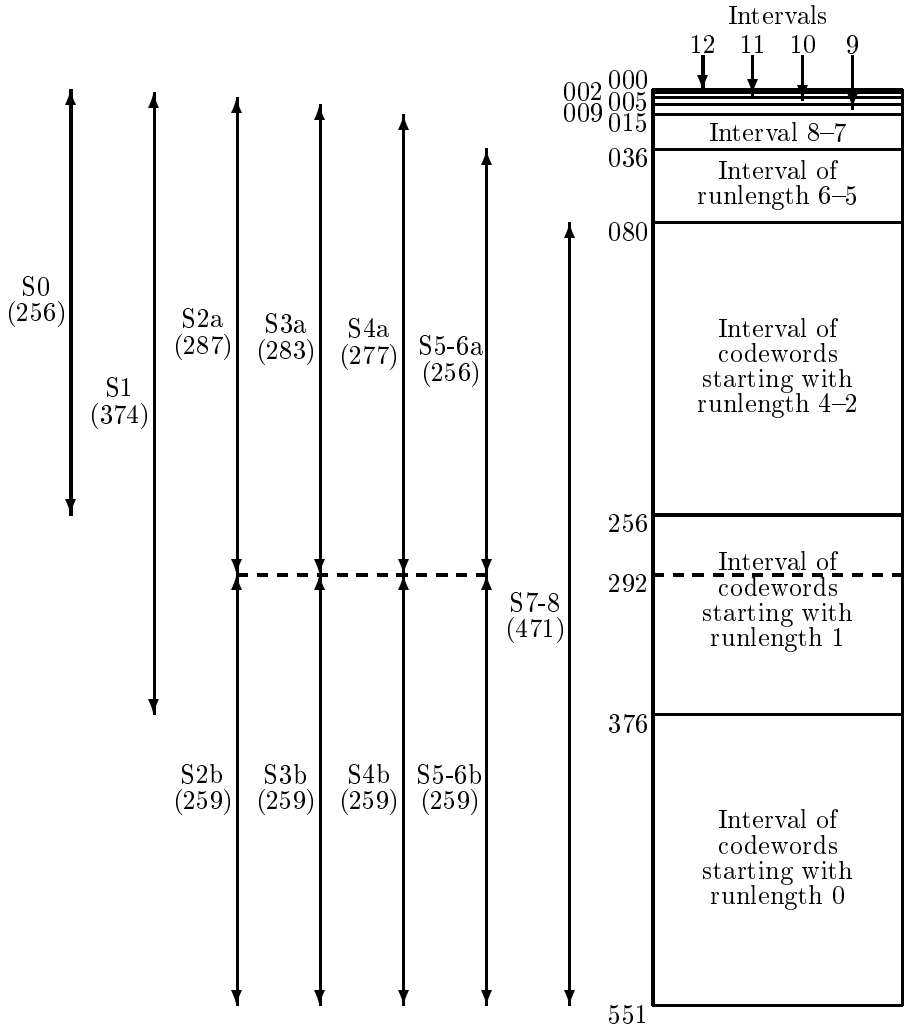

Figure 5: Location of codewords generated from each state in H.

Recall that codewords that appear twice in the table occupy consecutive addresses. By putting the first codeword in each such pair at an even address, the most significant seven bits of the current input byte are determined by the current received codeword.

As a final design stage, we reorder the codewords in our table so that the structure of Figure 5 is maintained, while satisfying additional conditions to allow DC control. More specifically, let a and $a + 2^8$ be two addresses in the table, both belonging to the same table segment corresponding to some state in H. Then, we require that, to the largest extent possible, the following two conditions hold:

(C1) The terminal states of the codewords at addresses a and $a+2^8$ should be the same; this allows to replace a codeword with its alternate without affecting subsequent

encoded codewords.

(C2) The codewords at addresses a and $a + 2^8$ should have different parities (of number of 1's).

We use the following heuristic procedure to reorder the table so that these two conditions are satisfied. Denote by $[x, y)$ a contiguous table portion that starts at address x and ends at address $y-1$ (inclusive). Starting with $s = 000$, we look for the largest ℓ such that for every $i \geq 0$, each of the table portions $[s-2^8 i, \ell-2^8 i)$ is entirely contained in some runlength interval in the table (negative addresses are regarded as 000). We also look for the largest h so that the portion $[s+2^8, h+2^8)$ is entirely contained in a runlength interval. If $\ell \leq h$, we reorder the portion $[s+2^8, h+2^8)$ so that its $(\ell-s)$-prefix matches—according to conditions (C1) and (C2)—the portion $[s, \ell)$, codeword by codeword. This might not always be possible, but in many cases it is, partly because in many cases h is significantly larger than ℓ. If $\ell > h$, we reorder the portions $[s-2^8 i, \ell-2^8 i)$ simultaneously for $i \geq 0$, to match $[s+2^8, h+2^8)$. This process is then re-iterated with $s = \min\{\ell, h\}$, until the whole table is scanned.

Applying this procedure to our table produces a reordering of the table where condition (C1) is fully met, whereas condition (C2) is satisfied for all but 16 pairs of addresses. The procedure can be generalized to any table-based rate $p : q$ encoder for a (d, k)-RLL constraint with $q \geq k$ that has been constructed along the lines of this section, and it will find a full matching if there exists one whenever address 2^p in the table is an interval boundary of one of the runlength ranges. (One can easily adapt this technique to handle also the case where there are two interval boundaries in the table at a distance which is close to 2^p.)

The state diagram of the resulting encoder is a graph $\mathcal{E}$ with eight states, and the adjacency matrix of $\mathcal{E}$ is an 8×8 matrix $A_{\mathcal{E}}$ where all the rows are equal to

$$[\ 59\ \ 40\ \ 28\ \ 19\ \ 13\ \ 15\ \ 75\ \ 7\ ]\ .$$

Rows and columns are indexed by the encoder states, according to the following order: S0, S1, S2a, S3a, S4a, S5-6a, S2-6b, and S7-8 (the entries in $A_{\mathcal{E}}$ do not take into account alternate codewords that are used for DC control). The peculiarity of having equal rows in $A_{\mathcal{E}}$ for all encoder states is a consequence of the fact that the terminal states are the same for codewords that are located in the table at addresses which are at distance 2^8 apart. Assuming independent and uniform distribution on the input bytes, the stationary probability of entering a given encoder state u (see [4, p. 41]) is obtained by dividing $(A_{\mathcal{E}})_{v,u}$ (for an arbitrary v) by 2^8.

The discussion in this section has concentrated on the encoder design of Section 3. Similar design tools can be applied to obtain the encoder of Section 2, as well as encoders for other certain (d, k)-RLL constraints.

Acknowledgment

I would like to thank Josh Hogan from Hewlett-Packard Laboratories for his invaluable contribution to this work.

References

[1] R.L. Adler, D. Coppersmith, M. Hassner, *Algorithms for sliding block codes — an application of symbolic dynamics to information theory, IEEE Trans. Inform. Theory,* 29 (1983), 5–22.

[2] P.A. Franaszek, *Sequence-state methods for run-length-limited coding, IBM J. Res. Develop.,* 14 (1970), 376–383.

[3] J. Gu, T. Fuja, *A new approach to constructing optimal block codes for runlength-limited channels, IEEE Trans. Inform. Theory,* 40 (1994), 774–785.

[4] K.A.S. Immink, *Coding Techniques for Digital Recorders,* Prentice Hall, New York, 1991.

[5] K.A.S. Immink, *EFMPlus: The coding format of the multimedia compact disc, IEEE Trans. Consum. Electron.,* 41 (1995), 491–497.

[6] K.A.S. Immink, H. Ogawa, *Method for encoding binary data,* US patent 4,501,000 (1985).

[7] R.I. Lockey, T.W. Moore, *Digital transmission system using discrepancy line coding,* UK Patent 1,569,076 (1980).

[8] J.C. Mallinson, J.W. Miller, *Optimal codes for digital magnetic recording, Radio and Elec. Eng.,* 47 (1977), 172–176.

[9] J.W. Miller, US patent 4,027,335 (1977).

[10] A.M. Patel, *Zero-modulation encoding in magnetic recording, IBM J. Res. Develop.,* 19 (1975), 366–378.

[11] B.H. Marcus, R.M. Roth, P.H. Siegel, *Constrained systems and coding for recording channels,* in *Handbook of Coding Theory,* V.S. Pless and W.C. Huffman (Editors), Elsevier, Amsterdam, 1998, 1635–1764.

[12] B.H. Marcus, P.H. Siegel, J.K. Wolf, *Finite-state modulation codes for data storage, IEEE J. Sel. Areas Comm.,* 10 (1992), 5–37.

address	Contents of table (hexadecimal)															
(decimal)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
000	0021	0022	0024	0020	0041	0081	0101	0201	0249	0049	0089	0091	0109	0111	0121	0042
016	0082	0102	0202	0092	0112	0122	0044	0048	0084	0088	0090	0124	0224	0244	0248	0040
032	0080	0100	0209	0211	0221	0241	0212	0222	0104	0108	0110	0120	0204	0208	0210	0220
048	0240	0242	0401	0449	0489	0491	0801	0849	0889	0891	0909	0911	0921	1049	1089	1091
064	0409	0411	0421	0441	0481	0809	0811	0821	0841	0881	0901	0402	0492	0802	0892	0912
080	0922	1002	0412	0422	0442	0482	0812	0822	0842	0882	0902	0404	0408	0410	0420	0804
096	0808	0810	0820	0924	0424	0444	0448	0484	0488	0490	0824	0844	0848	0884	0888	0890
112	0904	0908	0910	0920	1024	1044	0440	0480	0840	0880	0900	1109	1111	1121	1209	1211
128	1221	1241	1009	1011	1021	1041	1081	1101	1201	1249	1092	1112	1122	1212	1222	1242
144	1012	1022	1042	1082	1102	1202	1004	1008	1010	1020	1124	1224	1244	1248	1048	1084
160	1088	1090	1104	1108	1110	1120	1204	1208	1210	1220	1040	1080	1100	1240	2049	2089
176	2091	2109	2111	2121	2209	2211	2221	2241	2409	2411	2421	2441	2481	2009	2011	2021
192	2041	2081	2101	2201	2249	2401	2449	2489	2092	2112	2122	2212	2222	2242	2412	2422
208	2442	2482	2012	2022	2042	2082	2102	2202	2402	2492	2004	2008	2010	2020	2124	2224
224	2244	2248	2424	2444	2448	2484	2488	2490	2024	2044	2048	2084	2088	2090	2104	2108
240	2110	2120	2204	2208	2210	2220	2404	2408	2410	2420	2040	2080	2100	2240	2440	2480
256	2491	4802	4804	4824	4101	4081	4041	4021	4849	4809	4409	4811	4209	4411	4821	4402
272	4202	4102	4082	4812	4412	4822	4404	4808	4204	4408	4810	4848	4890	4920	4424	4100
288	4080	4040	4109	4211	4421	4841	4212	4422	4104	4208	4410	4820	4924	4108	4210	4420
304	4840	4842	4011	4449	4891	4491	4009	4489	4889	4249	4921	4911	4909	4201	4401	4801
320	4089	4111	4221	4441	4881	4049	4091	4121	4241	4481	4901	4042	4892	4022	4922	4492
336	4912	4012	4112	4222	4442	4882	4092	4122	4242	4482	4902	4084	4088	4110	4220	4044
352	4048	4090	4120	4024	4448	4490	4844	4888	4910	4488	4248	4224	4884	4444	4908	4484
368	4020	4904	4244	4124	4010	4008	4440	4880	4240	4480	4900	8101	8081	8041	8201	8849
384	8449	8249	8209	8409	8809	9009	9011	8811	9021	8411	8102	8202	8892	8402	9092	8922
400	8212	8412	8812	9012	9022	8822	8084	8104	8204	8404	8808	8804	9010	9008	8224	8824
416	8424	8248	9024	8848	8448	8444	9048	9090	8890	8490	9040	8840	9080	8040	8021	8489
432	8011	8491	8909	9209	8891	8889	8911	9109	9049	8921	9211	9111	9089	8211	8821	9041
448	9249	8421	8841	8109	9081	8441	8881	8111	8082	9112	8912	9122	9222	8492	8802	9242
464	9212	8042	8112	8422	9042	8842	9082	8882	8222	8442	8110	8210	8208	8408	8810	9020
480	9124	9244	9004	8820	8924	8410	9224	9248	8020	8884	8904	8844	8488	8244	8920	8910
496	8908	8124	9120	9044	9084	9104	9088	9110	9220	9108	8440	8880	9100	8080	8100	9240
512	8089	9102	8888	8108	9101	8221	8121	8049	8241	9091	9221	9121	9241	8401	8801	8092
528	8482	8242	8122	9002	8022	8012	9210	9208	9204	8484	8010	8420	8044	8220	8088	8240
544	8480	8900	8091	8901	9201	8481	8902	9202	8024	8120	8048	8090				

Table 4: Table of the $(2, 10)$-RLL encoder.

address	Contents of table (hexadecimal)															
(decimal)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
000	0004	0004	0008	0008	0009	0011	0010	0010	0012	0021	0024	0024	0020	0020	0022	0041
016	0081	0049	0089	0091	0042	0082	0092	0080	0044	0044	0084	0084	0048	0048	0088	0088
032	0090	0090	0040	0040	0101	0102	0202	0201	0249	0109	0111	0121	0209	0211	0221	0241
048	0112	0122	0212	0222	0242	0100	0104	0104	0204	0204	0124	0124	0224	0224	0244	0244
064	0108	0108	0208	0208	0248	0248	0110	0110	0210	0210	0120	0120	0220	0220	0240	0240
080	0401	0449	0489	0491	0409	0411	0421	0441	0481	0809	0402	0492	0412	0422	0442	0482
096	0812	0480	0404	0404	0424	0424	0444	0444	0484	0484	0408	0408	0448	0448	0488	0488
112	0410	0410	0490	0490	0420	0420	0440	0440	0801	0849	0889	0891	0909	0911	0921	1001
128	1049	1089	1091	1109	1111	1121	1209	1211	1221	1241	0811	0821	0841	0881	0901	1009
144	1011	1021	1041	1081	1101	1201	1249	0802	0892	0912	0922	1002	1092	1112	1122	1212
160	1222	1242	0822	0842	0882	0902	1012	1022	1042	1082	1102	1202	0880	0900	1080	1100
176	0804	0804	0924	0924	1004	1004	1124	1124	1224	1224	1244	1244	0824	0824	0844	0844
192	0884	0884	0904	0904	1024	1024	1044	1044	1084	1084	1104	1104	1204	1204	0808	0808
208	1008	1008	1248	1248	0848	0848	0888	0888	0908	0908	1048	1048	1088	1088	1108	1108
224	1208	1208	0810	0810	1010	1010	0890	0890	0910	0910	1090	1090	1110	1110	1210	1210
240	0820	0820	0840	0840	1020	1020	1040	1040	0920	0920	1120	1120	1220	1220	1240	1240
256	2004	2004	2008	2008	2009	2011	2010	2010	2012	2021	2024	2024	2020	2020	2022	2041
272	2081	2001	2049	2089	2042	2082	2002	2080	2044	2044	2084	2084	2048	2048	2088	2088
288	2090	2090	2040	2040	2101	2102	2202	2201	2249	2091	2109	2111	2121	2209	2211	2221
304	2092	2112	2122	2212	2222	2100	2104	2104	2204	2204	2124	2124	2224	2224	2244	2244
320	2108	2108	2208	2208	2248	2248	2110	2110	2210	2210	2120	2120	2220	2220	2240	2240
336	2401	2449	2489	2491	2241	2409	2411	2421	2441	2481	2402	2492	2242	2412	2422	2442
352	2482	2480	2404	2404	2424	2424	2444	2444	2484	2484	2408	2408	2448	2448	2488	2488
368	2410	2410	2490	2490	2420	2420	2440	2440	4009	4011	4021	4041	4081	4101	4201	4249
384	4401	4449	4489	4491	4801	4849	4889	4891	4909	4911	4049	4089	4091	4109	4111	4121
400	4209	4211	4221	4241	4409	4411	4421	4012	4022	4042	4082	4102	4202	4402	4492	4802
416	4892	4912	4002	4092	4112	4122	4212	4222	4242	4412	4422	4442	4480	4880	4900	4080
432	4024	4024	4044	4044	4084	4084	4104	4104	4204	4204	4404	4404	4004	4004	4124	4124
448	4224	4224	4244	4244	4424	4424	4444	4444	4484	4484	4824	4824	4844	4844	4048	4048
464	4088	4088	4108	4108	4008	4008	4248	4248	4448	4448	4488	4488	4848	4848	4888	4888
480	4908	4908	4090	4090	4110	4110	4210	4210	4010	4010	4490	4490	4890	4890	4910	4910
496	4120	4120	4220	4220	4240	4240	4420	4420	4440	4440	4020	4020	4040	4040	4920	4920
512	4804	4804	4208	4208	4441	4481	4410	4410	4482	4809	4884	4884	4820	4820	4812	4811
528	4901	4841	4881	4921	4882	4902	4922	4100	4904	4904	4924	4924	4408	4408	4808	4808
544	4810	4810	4840	4840	4821	4822	4842									

Table 5: Table of the $(2, 12)$-RLL encoder.