HEWLETT PACKARD

# On Runlength-Limited Coding with DC Control

Ron M. Roth
HP Laboratories Israel *
HPL-96-52 (R.1)
February, 1998

runlength-limited coding, optical recording, magnetic recording, DC control

This work describes a lossless coding scheme that maps unconstrained binary sequences into sequences that obey the $(d,k)$-RLL constraint. The input sequence is divided into nonoverlapping blocks, and a number of input blocks can be encoded into two different codewords which have different parity of 1's, thus allowing for DC control. The proposed scheme uses a compact table of codewords that serves all states, as well as a simple mechanism to locate the codeword, or codewords, that are associated with each input block. Decoding is carried out in a state-independent manner.

Two examples are provided. The first example is a four-state (2,10)-RLL encoder at rate 8:16, using a table of 546 codewords. The percentage of bytes that allow for DC control is 49.7%, on the average. Decoding is carried out by recovering the input byte from the current 16-bit codeword.

The second example is a (2,12)-RLL encoder at rate 8:15, using a table of 551 codewords. The percentage of bytes that allow for DC control ranges between 7.6% and 12.2%, depending on the number of states of the encoder which, in turn, can range between four and eight. Decoding is carried out by recovering the input byte from the current, and possibly the next 15-bit codeword.

*Technion City, Haifa 32000, Israel

# 1 Introduction

## 1.1 $(d, k)$-RLL constraints and charge constraints

In optical and magnetic recording systems, the bit stream that is written into the device must satisfy certain constraints. Apparently, the most common family of such constraints are 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, the compact disk uses a code with the constraint $(d, k) = (2, 10)$ [Imm91], [IO85]. An example of a sequence satisfying this constraint is

$$\cdots 000100000000001001000001 00 \cdots ,$$

in which the first four runlengths are 3, 10, 2, and 5. Magnetic recording standards include the $(1, 7)$-RLL constraint [AHM82] and the $(1, 3)$-RLL constraint [Mill63]. The parameter $k$ is imposed to guarantee sufficient sign changes in the recorded waveform which are required for clock synchronization during read-back. The parameter $d$ is required to prevent inter-symbol interference.

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.

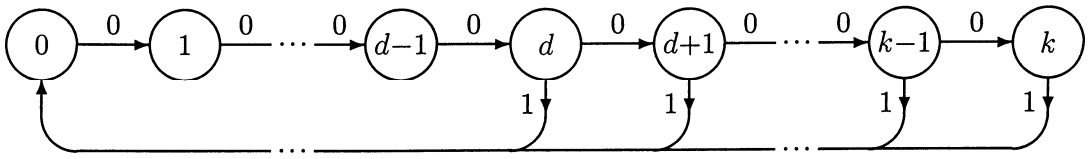


Figure 1: Graph presentation of a $(d, k)$-RLL constraint.

Another type of constraint requires that the input stream will not contain a DC component. This can be achieved by requiring the existence of a positive integer $B$ such any recorded sequence $w_1 w_2 \cdots w_\ell$, now regarded over the symbol alphabet $\{+1, -1\}$, will satisfy the inequality

$$\Big| \sum_{h=i}^{j} w_h \Big| \leq B$$

for every $1 \leq i \leq j \leq \ell$. Sequences that obey these conditions are said to satisfy the $B$-charge constraint. The larger the value of $B$, the less reduction there will be in the DC component.

However, in certain applications, the charge constraint can be relaxed, thus allowing to obtain higher coding rates. In such applications, the DC control is achieved by using a

coding scheme that allows in every prescribed number of symbols (on the average) to reverse the polarity of subsequent symbols.

DC control and $(d, k)$-RLL constraints can be combined. In such schemes, the constraint consists of binary sequences $z_1 z_2 z_3 \cdots$ that satisfy the $(d, k)$-RLL constraint, such that the respective NRZI sequences

$$(-1)^{z_1} \ (-1)^{z_1+z_2} \ (-1)^{z_1+z_2+z_3} \ \cdots$$

have a controlled DC component.

DC control is used in optical recording to avoid interference with the servo system and to allow filtering of noise resulting from finger-prints [Imm91, Ch. 2]. Charge constraint sequences have also been used in many magnetic-tape recording systems employing rotary-type recording heads. See [Patel75], [MM77], [Mill77].

## 1.2 Finite-state encoders

An encoder is a uniquely-decodable (or lossless) mapping of an unconstrained data stream 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. In optical storage devices, the $p$-bit input block is typically taken to be a byte so that it matches the unit size used in the error-correction scheme.

Each encoder must have a decoder that recovers the input stream out of the constrained sequence generated by the encoder. A state-dependent decoder accepts, as input, a $q$-bit codeword and reconstructs the respective input $p$-bit block depending on the state, as well as a prescribed number of upcoming $q$-bit codewords. Such a decoder retraces the encoding steps of the encoder, thus reconstructing the input bit stream. However, in the presence of errors, a state-dependent decoder is susceptible to (possibly unbounded) error propagation, since it might lose track of the state sequence of the encoder.

Error propagation can be limited by the use of sliding-block decoders. A sliding-block decoder makes a decision on a 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 a fixed number $\mathsf{m}$ of preceding codewords and a fixed number $\mathsf{a}$ of later codewords. Thus, a single error at the input to a sliding-block decoder can only affect the decoding of at most $\mathsf{m}+\mathsf{a}+1$ 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 [Imm91, p. 91] lists the Shannon capacities of several $(d, k)$-RLL constraints.

## 1.3 New coding scheme

This work presents a coding method for certain $(d, k)$-RLL constraints with DC control that is suitable for optical storage devices.

The coding scheme has the following properties:

1. The encoder is a finite-state machine that maps input blocks (typically bytes) to codewords. The first example provided is a rate 8:16 four-state encoder for the $(2, 10)$-RLL constraint, and the second example is a rate 8:15 encoder for the $(2, 12)$-RLL constraint. The number of states of the latter encoder ranges between four and eight, depending on the the required DC control.

2. The main building block of the encoder is a table of codewords that serves all states. In the provided examples, the sizes of the tables are 546 and 551 codewords, respectively. Encoding is carried out by prefixing the input block with a fixed number of bits (two bits in the provided examples) which depend on the input block as well as the current encoder state. The result is an address to the table from which the current encoded codeword is taken. Assuming random input, the probability of being at any given state is independent on the previous state.

3. The encoder features a DC control by allowing for a number of input blocks to have two possible encoded codewords: The parity of the number of 1's is different in these two codewords and, so, the respective NRZI sequences end with a different polarity, thus allowing to reverse the polarity of subsequent codewords. In those cases where DC control is possible, the address of the alternate codeword is obtained by adding a fixed number ($2^8$ in the provided examples) to the computed address. Furthermore, both codewords lead to the same state, thus allowing for local replacement of a codeword without affecting subsequent codewords.

4. The encoder has a sliding-block decoder that can be efficiently implemented using an associative memory which contains the encoder table. The current input block is recovered from the current codeword and, possibly, a prescribed number of subsequent codewords. In the first example, the input byte can be fully recovered from the address in the table where the current received codeword is located. In the second example, the most significant seven bits of the current input byte can be obtained in a similar manner. As for the least significant bit (l.s.b.), it may either be computed from that address as well, or — depending on the current codeword — by determining which half of the table the next codeword is located in.

The novelty in the design approach here is using 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 allows for an easy mechanism of DC control, with the additional property that the probability of being at a given state is independent of the previous state.

The first example of the $(2, 10)$-RLL encoder is presented in Section 2 (see also the appendix), and the second example of the $(2, 12)$-RLL encoder is presented in Section 3. The design of the latter encoder is summarized in Section 4. Similar design tools have led to the $(2, 10)$-RLL encoder as well, and they can be used to obtain encoders for other certain RLL constraints.

# 2 Coding scheme for the (2,10)-RLL constraint

## 2.1 Encoding

The $(2, 10)$-RLL encoder consists of a table of 546 codewords, each having 16 bits. Table 11 lists the codewords in hexadecimal form.

At each encoding step, the encoder can be in one out of four 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, as shown in Figure 2. The two-bit prefix depends on how the value

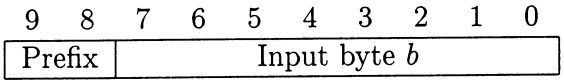


Figure 2: Format of address to table.

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 $\phi$ stands here for the "don't care" sign).

For example, if the current state is S6-8 and the input byte is 038 (decimal), then the address will be $256 + 038 = 294$.

The output codeword is the entry in Table 11 at the computed address. The next encoder state is the unique state with an associated runlength range that includes the last runlength of the generated codeword.

In the example, the output codeword is 010000100010010 (4212 in hexadecimal notation) and the next encoder state is S1.

| State | Thresholds (decimal) | | Prefixes (binary) | | |
|---|---|---|---|---|---|
| | T1 | T2 | $0 \leq b < \mathrm{T1}$ | $\mathrm{T1} \leq b < \mathrm{T2}$ | $\mathrm{T2} \leq b < 256$ |
| S0 | 000 | 001 | $\phi$ | 01 or 00 | 00 |
| S1 | 004 | 123 | 01 | 01 or 00 | 00 |
| S2-5 | 034 | 050 | 10 or 01 | 01 | 01 or 00 |
| S6-8 | 034 | 174 | 10 or 01 | 01 | 01 or 00 |

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

There are cases where more than one prefix is possible, resulting in two different codeword candidates. The encoder table is designed so that those codeword candidates will have different parity of sign changes (namely, different parity of number of 1's), thus allowing for DC control. Furthermore, both choices 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 0100100000010001. Both codewords lead to state S0.

A schematic diagram of the encoder is shown in Figure 3.

Assuming random input, the percentage of input bytes within an input sequence for which two codeword candidates exist is 49.7% on the average.

**Remark 1.** The encoder table does not contain the (hexadecimal) words 8111, 8121, 8421, 8821, 8812, 8822, 9124, 9244, 8408, 8810, as well as none of the ten 16-bit words in the $(2, 10)$-RLL constraint that end with a runlength of 9 or 10. Therefore, these words can be used for synchronization. By reducing the threshold 034 in Table 1, more codewords at the end of the encoder table can be reserved for special use.

The appendix contains an alternate construction of a $(2, 10)$-RLL encoder in which the encoded data never contains the binary pattern 00010001000100010001000100010001, starting at any bit phase. This, in turn, allows having synchronizing words that consist of this pattern, followed by auxiliary information which may be required during synchronization. The percentage of bytes that allow for DC control in the resulting encoder is smaller by approximately 1% compared to the encoder presented in this section.

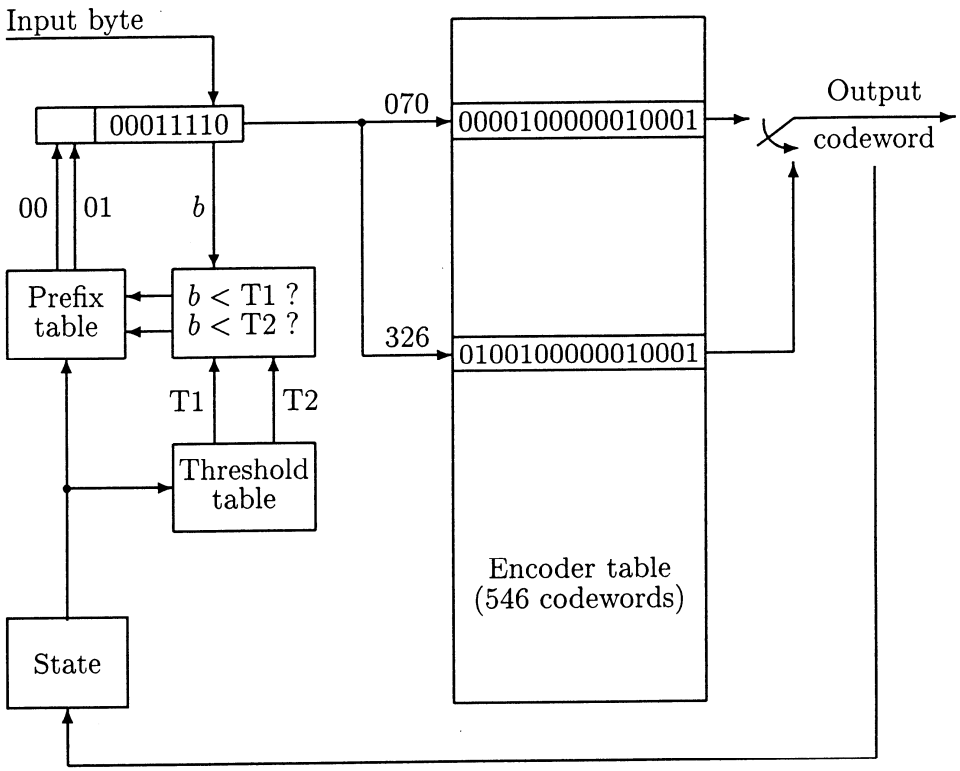


Figure 3: Schematic diagram of the (2, 10)-RLL encoder.

## 2.2 Decoding

Decoding is carried out in a state-independent manner, using the encoder table which is assumed to reside in an associative read-only memory. First, an address is found of a table entry that contains the received codeword. The input byte is then obtained by truncating the two most significant bits (m.s.b.'s) of the address where the codeword is located.

# 3 Coding scheme for the (2,12)-RLL constraint

## 3.1 Levels of encoding

The coding scheme presented next is for the (2, 12)-RLL constraint. The encoding algorithm has several encoding levels, depending on the acceptable number of states and the desirable DC control. In the basic level (referred to as Level-4), the encoder has four states and DC control (i.e., codeword replacement) is possible in 7.6% of the input bytes, on the average. Extended levels are obtained by adding more states, reaching eight states in the Level-8

encoder. The latter level allows for DC control in 12.2% of the input bytes, on the average.

The encoder structure for Level-4 is described in Section 3.2. The decoding algorithm is discussed in Section 3.3, and the extended levels of encoding are presented in Section 3.4.

## 3.2 Basic level of encoding — Level-4

Table 12 lists the encoder table which consists of a table of 551 codewords, each of length 15 bits. The table is divided into two *sections*: The first section contains the first 292 codewords, and the second contains the remaining 259 codewords.

At each encoding step, the encoder can be in one out of four states: S0, S1, S2-6a, or S2-6b. 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.

| Last runlength of previous codeword | L.s.b. of previous input byte | Current state |
|---|---|---|
| 0 | $\phi$ | S0 |
| 1 | $\phi$ | S1 |
| 2, 3, 4, 5, or 6 | 0 | S2-6a |
| 2, 3, 4, 5, or 6 | 1 | S2-6b |
| 7 or 8 | $\phi$ | S2-6b |

Table 2: States in Level-4.

As in the $(2, 10)$-RLL encoder that was described in Section 3, 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.

| State | Thresholds (decimal) | | Prefixes (binary) | | |
|---|---|---|---|---|---|
| | T1 | T2 | $0 \leq b <$ T1 | T1 $\leq b <$ T2 | T2 $\leq b < 256$ |
| S0 | 000 | 000 | $\phi$ | $\phi$ | 00 |
| S1 | 002 | 120 | 01 | 01 or 00 | 00 |
| S2-6a | 036 | 036 | 01 | 01 or 00 | 00 |
| S2-6b | 036 | 039 | 10 | 10 or 01 | 01 |

Table 3: Thresholds and prefixes in Level-4.

The output codeword is the entry in Table 12 at the computed address. The next encoder state is determined by Table 2.

DC control is attained by allowing certain input bytes to have two different codeword candidates. Assuming random input, the percentage of such input bytes within an input sequence is 7.6% on the average.

A threshold T1 (respectively, T2) for a state S will be denoted hereafter by T1(S) (respectively, T2(S)).

**Remark 2.** If T2(S2-6b) is changed to 036, then the codewords at addresses 548, 549, and 550 will never be used. In addition, the encoder table does not contain any of the ten 15-bit words in the $(2, 12)$-RLL constraint that end with a runlength of 9 or more. This makes all those words available for synchronization.

Alternately, T2(S2-6a) and T1(S2-6b) can be increased to any (same) value between 036 and T2(S2-6b).

## 3.3 Decoding

As with the $(2, 10)$-RLL encoder of Section 2, decoding is carried out in a state-independent manner, using the encoder table. 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, the first of which having an even address (therefore, the respective two addresses differ in their l.s.b. only). By truncating the two m.s.b.'s of the address, all the bits of the input byte, with the exception of its l.s.b., are determined. In order to recover the l.s.b. of the input byte, the *next* received codeword is found in the encoder table. If it is located in the first section of Table 12 (i.e., at an address smaller than 292), then the l.s.b. of the input byte is 0. Otherwise, it is 1.

**Remark 3.** When the received codeword ends with a runlength of 2, 3, 4, 5, or 6, the criterion for determining the l.s.b. of the input byte can be modified provided that the following changes are made: (a) increase T2(S2-6a) and T1(S2-6b) to 038, and (b) switch between the table contents at the following address pairs: 036 $\leftrightarrow$ 041, 037 $\leftrightarrow$ 048, 292 $\leftrightarrow$ 297, and 293 $\leftrightarrow$ 304. With those changes, the l.s.b. of the current input byte can be determined by the first and second runlengths of the next codeword according to Table 4, whenever the current codeword ends with a runlength of 2 through 6.

| First runlength in next codeword | Second runlength in next codeword | Decoded bit |
|---|---|---|
| 2 or more | $\phi$ | 0 |
| 1 | 5 or more | 0 |
| 1 | 2, 3, or 4 | 1 |
| 0 | $\phi$ | 1 |

Table 4: Alternate method for decoding the l.s.b. of the input byte.

## 3.4 Extended levels of encoding and decoding

The advantage of using extended levels of encoding is having more input bytes, on the average, where DC control is possible, at the expense of increasing the number of states of the encoder. Assuming random input, the percentage of input bytes which have two codeword candidates reaches 12.2% on the average, compared to 7.6% in Level-4.

Level-8 is an extended encoding level in which the decoder has the largest number of states, namely, eight. Each state is determined by the last runlength of the previous codeword and the l.s.b. of the previous input byte, according to Table 5.

| Last runlength of previous codeword | L.s.b. of previous input byte | Current state |
|---|---|---|
| 0 | $\phi$ | S0 |
| 1 | $\phi$ | 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 | $\phi$ | S7-8 |

Table 5: States in Level-8.

Encoding is carried out by comparing the input byte $b$ to threshold values and adding suitable prefixes, according to Table 6.

The last three codewords in the encoder table can be used for synchronization if T2(S2-6a) and T1(S7-8) are changed to 036.

There are 215 input bytes at state S7-8 for which two output codewords are possible. In each such pair, both codewords lead to the same encoder state, and all but 16 pairs allow for DC control (namely, there are 16 pairs where the two codewords have the same parity of number of 1's). In all other states, all replacement codewords allow for DC control.

| State | Thresholds (decimal) | | Prefixes (binary) | | |
|---|---|---|---|---|---|
| | T1 | T2 | $0 \leq b < \text{T1}$ | $\text{T1} \leq b < \text{T2}$ | $\text{T2} \leq b < 256$ |
| S0 | 000 | 000 | $\phi$ | $\phi$ | 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 6: Thresholds and prefixes in Level-8.

Some states in Level-8 can be deleted, thus generating lower encoding levels. When a state is deleted, the codewords that led the encoder to that state need to be redirected into another state. The states that can be deleted and the redirection required are summarized in Table 7.

| State deleted | Redirection required to state |
|---|---|
| S2a | S3a, S4a, or S5-6a |
| S3a | S4a or S5-6a |
| S4a | S5-6a |
| S7-8 | S2-6b |

Table 7: Redirection of codewords when states are deleted.

In particular, the Level-4 encoder of Section 3.2 is obtained by deleting states S2a, S3a, S4a, and S7-8, in which case state S5-6a becomes state S2-6a.

Table 8 summarizes the percentage of random input bytes, on the average, for which DC control is possible, for certain configurations of deletion of states.

The decoder described in Section 3.3 can be used *as is* for extended encoding levels as well. Remark 3 still holds, except that now there are several thresholds T2 equaling 036 in Table 6 (as well as T1(S2-6b)) that need to be changed to 038.

# 4 Code design

In this section, we outline the principles that guided the design of the coding scheme presented in Section 3. Those principles also guided the design of the encoder in Section 2,

| Encoding level | States deleted | Percentage of bytes with DC control |
|---|---|---|
| Level-8 | None | 12.2% |
| Level-7 | S4a | 11.8% |
| Level-6 | S3a and S4a | 11.0% |
| Level-5 | S2a, S3a, and S4a | 9.7% |
| Level-4 | S2a, S3a, S4a, and S7-8 | 7.6% |

Table 8: Some configurations of extended encoding levels.

and can in fact be applied to design similar coding schemes for other certain $(d, k)$-RLL constraints.

## 4.1 Preliminaries

Let $G$ denote the graph presentation of the $(2, 12)$-RLL constraint which is shown in Figure 4. The adjacency matrix of $G$, denoted $A_G$, is a $13 \times 13$ matrix whose rows and columns are

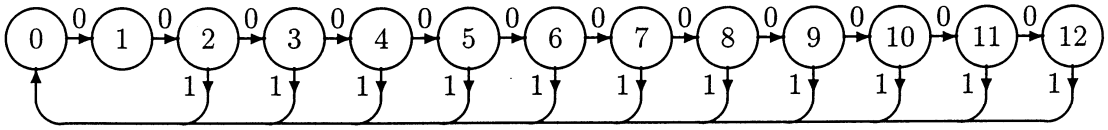


Figure 4: $(2, 12)$-RLL constraint.

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 $G$. That is, for $0 \leq u, v \leq 12$,

$$(A_G)_{u,v} = \begin{cases} 1 & \text{if } v = u + 1 \\ 1 & \text{if } 2 \leq u \leq 12 \text{ and } v = 0 \\ 0 & \text{otherwise} \end{cases}.$$

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 generated by the respective path in $G$. The adjacency matrix of $G^q$ equals $A_G^q$. For $q = 15$ we get the matrix in Figure 5.

## 4.2 State merging and state splitting

To obtain a rate 8:15 finite-state encoder, we first invoke the technique of *state splitting* of Adler, Coppersmith, and Hassner [ACH83]; see also a summary in [MSW92] and [MRS95].

$$A_G^{15} = \begin{bmatrix} 59 & 40 & 28 & 19 & 13 & 9 & 6 & 4 & 3 & 2 & 1 & 1 & 1 \\ 87 & 59 & 40 & 28 & 19 & 13 & 9 & 6 & 4 & 3 & 2 & 1 & 1 \\ 126 & 87 & 59 & 40 & 28 & 19 & 13 & 9 & 6 & 4 & 3 & 2 & 1 \\ 125 & 86 & 59 & 40 & 27 & 19 & 13 & 9 & 6 & 4 & 3 & 2 & 1 \\ 124 & 85 & 58 & 40 & 27 & 18 & 13 & 9 & 6 & 4 & 3 & 2 & 1 \\ 122 & 84 & 57 & 39 & 27 & 18 & 12 & 9 & 6 & 4 & 3 & 2 & 1 \\ 119 & 82 & 56 & 38 & 26 & 18 & 12 & 8 & 6 & 4 & 3 & 2 & 1 \\ 115 & 79 & 54 & 37 & 25 & 17 & 12 & 8 & 5 & 4 & 3 & 2 & 1 \\ 109 & 75 & 51 & 35 & 24 & 16 & 11 & 8 & 5 & 3 & 3 & 2 & 1 \\ 100 & 69 & 47 & 32 & 22 & 15 & 10 & 7 & 5 & 3 & 2 & 2 & 1 \\ 87 & 60 & 41 & 28 & 19 & 13 & 9 & 6 & 4 & 3 & 2 & 1 & 1 \\ 68 & 47 & 32 & 22 & 15 & 10 & 7 & 5 & 3 & 2 & 2 & 1 & 0 \\ 40 & 28 & 19 & 13 & 9 & 6 & 4 & 3 & 2 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Figure 5: Adjacency matrix of $G^{15}$.

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

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

In particular, any $(A_G^{15}, 2^8)$-approximate eigenvector must contain a component which is greater than 1. Combining this with Proposition 3.34 in [MRS95], it follows that there is no finite-state encoder for the $(2, 12)$-RLL constraint at rate 8:15 that has a sliding-block decoder with $\mathsf{m} = \mathsf{a} = 0$ (refer to Section 1.2 and see also [GuF94]). The encoder we construct has a sliding-block decoder with $\mathsf{m} = 0$ and $\mathsf{a} = 1$. Furthermore, seven bits of the current input byte can be determined from the current received codeword alone. The l.s.b. of the current input byte is the only bit which may require the next received codeword for decoding. (We remark that in the case of the $(2, 10)$-RLL constraint, there is an approximate eigenvector which is a 0–1 vector. Indeed, in this case, we have managed to obtain in Section 2 a sliding-block decoder with $\mathsf{m} = \mathsf{a} = 0$. Such encoders are called block decodable; see also [GuF94].)

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 [MSW92], [MRS95]). Similarly, states 2 through 5 can be merged into state 6. Indeed, when doing so, we will end up with the Level-5 encoder which is defined in Table 8. To obtain the Level-8 encoder, we merge only state 5 into state 6. (The same applies with respect 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 gain more DC control, we 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 (word) of each edge in $H$ uniquely determines the terminal state of that edge; in fact, the last runlength in that word identifies that terminal state. Such graphs $H$ are said to have memory 1.

The adjacency matrix of $H$ is given by

$$A_H = \begin{bmatrix} 59 & 40 & 28 & 19 & 13 & 15 & 7 \\ 87 & 59 & 40 & 28 & 19 & 13 & 10 \\ 126 & 87 & 59 & 40 & 28 & 32 & 15 \\ 125 & 86 & 59 & 40 & 27 & 32 & 15 \\ 124 & 85 & 58 & 40 & 27 & 31 & 15 \\ 119 & 82 & 56 & 38 & 26 & 30 & 14 \\ 109 & 75 & 51 & 35 & 24 & 27 & 13 \end{bmatrix},$$

with an $(A_H, 2^8)$-approximate eigenvector

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

States with weight greater than 1 need to be *split*. When a state $u$ is split, two or more descendent states are formed. The incoming edges to $u$ are *duplicated* into each of the descendent states, whereas the outgoing edges from $u$ are *partitioned* among the descendent states. In addition, the weight of $u$ is shared among the descendent states so that, after the splitting, the following holds: The weights of the descendent states are positive integers that sum up to the weight of their parent state, and the weights of the terminal states of the edges outgoing from each descendent state $v$ sum up to at least $2^8$ times the weight of $v$. An encoder is obtained after several rounds of splitting, when all weights become 1. 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 descendent 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 encoder in [Imm95] for the $(2, 10)$-RLL constraint can be obtained in a similar manner by one state splitting. On the other hand, we did not have to apply state splitting in order to obtain the encoder in Section 2.

## 4.3 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

turn to the next design step of defining a certain order on the outgoing edges (or rather, their labels) from each state in $H$. The specific order we choose will allow for simple decoding rules.

To this end, we write down all words of length 15 that satisfy the $(2, 12)$-RLL constraint, according to descending order of their first runlength. We omit words that end with a runlength of 9 or more, since the respective edges were deleted when $H$ was formed. On the other hand, each word ending with a runlength of 2, 3, 4, 5 or 6 will be written twice, in two consecutive places. Indeed, those words correspond to edges that were duplicated due to the splitting of states S2, S3, S4, and S5-6. The resulting list consists of 551 words and will serve as the codeword table of the encoder.

If we count edges with their multiplicity according to the weight of their terminal state, we find that there are $\sum_v (A_H)_{\mathrm{S0},v} x_v = 2^8$ outgoing edges from state S0 in $H$, and their labels start with runlengths between 2 and 12. Those labels appear as the first $2^8$ codewords in the table. Similarly, the first runlength of the labels of the edges outgoing from state S1 ranges between 1 and 11. Those labels appear in the table at $\sum_v (A_H)_{\mathrm{S1},v} x_v = 374$ consecutive entries, starting at address 002 (the third entry).

Figure 6 presents a schematic diagram that shows the location of codewords in the table that can be generated from each state in $H$. The rectangle to the right represents the table of 551 codewords, divided into *runlength intervals* according to their first runlength. Some runlengths have been combined in the figure; e.g., all codewords that start with a runlength between 2 and 4 are marked as one runlength interval.

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 $\sum_v (A_H)_{u,v} x_v$. 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 6 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 descendent state that inherits the name of the parent state with a suffix "a". The rest of the edges belong to the other descendent state that inherits the name of the parent state with the suffix "b". The number 292 was chosen so that state S5-6a will have at least $2^8$ outgoing edges. In order to have a valid splitting, we make sure that the entries at addresses 291 and 292 do not contain two copies of the same codeword. Each incoming edge to a state that was split is duplicated to enter both descendents of that state, and both new edges inherit the same label of the parent edge. A duplicated codeword in the table corresponds to such a duplicated edge. We will follow a convention whereby the first copy in the table of such a codeword corresponds

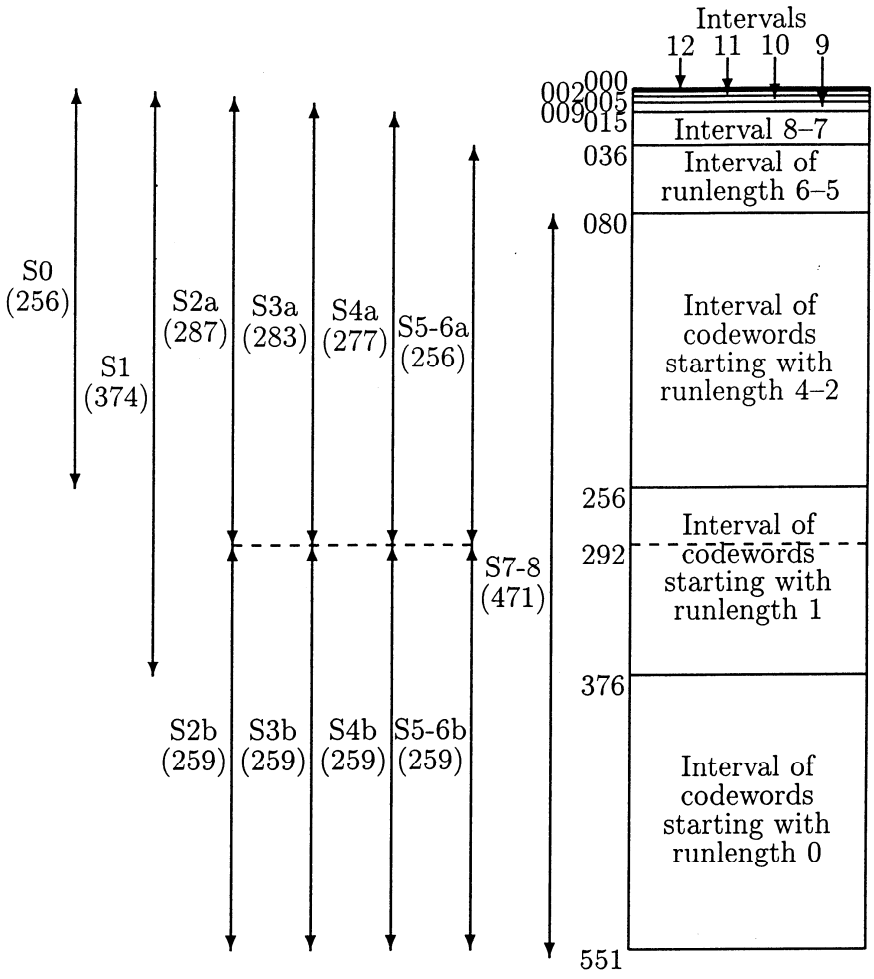


Figure 6: Location of words generated from each state in $H$.

to an edge entering state "a", whereas the second copy enters state "b".

The following observations can be made:

- 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 [Fra70] 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 table that serves all states.
- States S2b, S3b, S4b, and S5-6b are *equivalent* in the sense 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.

- All states, with the exception of S0 and S5-6a, have more than $2^8$ outgoing edges. States S1 and S7-8 have significantly more edges.

The current table already allows for a coding scheme as follows: Encoding is carried out by adding a two-bit prefix to the input byte as shown in Figure 2. The 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 6. Since the table segment of each state consists of at least $2^8$ codewords, such a prefix can always be found. In fact, in many cases more than one prefix is possible. Finding the right prefix can be translated into threshold comparison.

We demonstrate this on state S2a. The table segment that corresponds to this state occupies addresses $a$ in the range $005 \leq a < 292$. If the input byte is a number $b$ in the range $005 \leq b < 256$, then $b$ can serve as the address to the table from which the codeword is to be taken. Otherwise, if $b < 005$, then we obtain the address by adding $2^8$ (or prefixing '01') to $b$. Notice that prefixing '01' yields a valid address also when $b < 036$. Therefore, we set the thresholds T1 and T2 to 005 and 036, respectively, and encoding will proceed as follows: if $b <$ T1, then the prefix is '01'; otherwise, if $b <$ T2, then the prefix can be either '01' or '00'; otherwise, the prefix is '00'. These thresholds and prefixes coincide with those in Table 6.

Decoding is carried out as follows: If the received codeword appears once in the table, then the input byte is uniquely determined by the address of that codeword. Note that this holds regardless of the state from which this codeword was generated during encoding. If the codeword appears twice, then we need to verify whether the next encoder state was an "a" descendent of a state or rather state S2-6b. Codewords that can be generated from "a" states appear in the table at addresses smaller than 292, whereas codewords that can be generated from state S2-6b appear at addresses $\geq 292$. Therefore, by determining which section of the table the next codeword is located in, the decoder can fully recover the input byte.

Recall that codewords that appear twice in the table occupy consecutive addresses. If, in addition, we manage to put the first codeword in each such pair at an even address, then the most significant seven bits of the current input byte will be determined by the current received codeword. Such an assignment of addresses can be easily obtained in our case.

## 4.4 Caring for the DC control

So far, we have shown how a table can be constructed so that encoding and decoding are efficient and error propagation is limited. We will now reorder the codewords in our table so that the structure of Figure 6 is maintained, while satisfying additional conditions to allow for 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 will hold for every such pair of addresses:

(C1) The terminal states of the codewords at addresses $a$ and $a + 2^8$ should be the same. If this condition is satisfied, then we can interchangeably encode any one of those two codewords, without affecting subsequent encoded codewords.

(C2) The codewords at addresses $a$ and $a + 2^8$ should have different parities (of number of 1's). The option to choose between these two codewords during encoding will yield the desired effect of DC control.

In our case, for all $002 \leq a < 551 - 2^8 = 295$, there is a state in $H$ such that both $a$ and $a+2^8$ belong to the segment corresponding to that state in the table. For the sake of simplicity, we will apply conditions (C1) and (C2) also to the remaining addresses 000 and 001. This way, we obtain a "semi-periodic" table where the codewords at every pair of addresses at distance $2^8$ apart have the same terminal state and different parities (to the largest extent possible).

We reorder the table using the procedure of Figure 7. Table portions that start at address $x$ and end at address $y-1$ (inclusive) are denoted in the procedure by $[x, y)$. The basic idea of the procedure is scanning the table, starting at address 000, and identifying table portions $[s, \ell)$ such that for every $i = 0, 1, \ldots, \lfloor s/2^8 \rfloor$, each of the portions $[s-2^8 i, \ell-2^8 i)$ is entirely contained in some runlength interval in the table. We then look at a new portion $[s+2^8, h+2^8)$, which is also entirely contained in some runlength interval. If $\ell \leq h$, then we reorder the portion $[s+2^8, h+2^8)$ so that its $(\ell-s)$-prefix matches the portion $[s, \ell)$, codeword by codeword, in terms of conditions (C1) and (C2). This might not always be possible, but in many cases it is, partly because in many cases $h$ is significantly larger than $\ell$. If $\ell > h$, then we reorder the portions $[s-2^8 i, \ell-2^8 i)$ simultaneously for $i = 0, 1, \ldots, \lfloor s/2^8 \rfloor$, to match $[s+2^8, h+2^8)$.

Applying the procedure to our table, we obtain 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 would have found a full matching had there existed one.

The procedure can be generalized to obtain a semi-periodic table for 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 Sections 4.2 and 4.3 (namely, the encoder is obtained by state splitting of a graph presentation $H$ of a subset of the $(d, k)$-RLL constraint, and then constructing an encoder table in which codewords are ordered according to their first runlength and duplicated according to the weights of their terminal states in $H$). The procedure 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. The procedure can be easily adapted to handle also the case where there are two interval boundaries in the table at a distance which is close to $2^p$.

```
n = 2^8;  M = 551;
s ← 0;
while (s+n < M) {
    ℓ ← min( t | t > s and t−ni is a runlength interval boundary for
                                                some i = 0, 1, ..., ⌊s/n⌋ );
    h ← min( t | t > s and t+n is a runlength interval boundary );
    if (ℓ ≤ h) {
        Reorder [s+n, h+n) so that [s+n, ℓ+n) matches [s, ℓ);
        s ← ℓ; }
    else {
        Reorder [s−ni, ℓ−ni) simultaneously for i = 0, 1, ..., ⌊s/n⌋ so
                                    that [s−ni, h−ni) match [s+n, h+n);
        s ← h; } }
```

Figure 7: Procedure for reordering the table to allow for DC control.

The state diagram of the resulting encoder is a graph $\mathcal{E}$ with eight states, and the adjacency matrix of $\mathcal{E}$ is an $8 \times 8$ 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. These, in turn, are counted separately in the following matrix $D_{\mathcal{E}}$:

$$D_{\mathcal{E}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 28 & 19 & 12 & 9 & 6 & 7 & 34 & 3 \\ 7 & 5 & 3 & 2 & 2 & 2 & 9 & 1 \\ 6 & 4 & 3 & 2 & 1 & 2 & 8 & 1 \\ 5 & 3 & 2 & 2 & 1 & 1 & 6 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 48 & 35 & 22 & 14 & 9 & 11 & 56 & 4 \end{bmatrix}$$

(16 additional codewords that can be generated from state S7-8 have been omitted since they do not satisfy condition (C2)).

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. Therefore, the distribution of the terminal states of the first $2^8$ outgoing edges from a given encoder state is the same for all encoder states.

The stationary probability of being at a given encoder state $u$ is obtained by dividing $(A_{\mathcal{E}})_{v,u}$ (for an arbitrary $v$) by $2^8$. These probabilities are summarized in the second column of

| State | Stationary probability | DC control |
|---|---|---|
| S0 | 0.2305 | 0 |
| S1 | 0.1563 | 0.4609 |
| S2a | 0.1094 | 0.1211 |
| S3a | 0.0742 | 0.1055 |
| S4a | 0.0508 | 0.0820 |
| S5-6a | 0.0586 | 0 |
| S2-6b | 0.2930 | 0.0117 |
| S7-8 | 0.0273 | 0.7773 |

Table 9: Stationary probabilities of encoder states.

Table 9. The entries in the third column of Table 9 are obtained by dividing each row-sum in $D_{\mathcal{E}}$ by $2^8$. These numbers measure for each encoder state the fraction of input bytes that allow for DC control from that state. The expectation of those numbers, with respect to the stationary probabilities, yields the average fraction of input bytes in a random sequence that allow for DC control. This average is equal to 0.1219.

It is worthwhile pointing out that the Shannon capacity of the $(2, 12)$-RLL constraint is 0.547 and, therefore, the encoder operates at a rate, 8:15, which is just 2.5% below capacity. Furthermore, the effective Shannon capacity of a $(2, 12)$-RLL constraint having a DC control in 12.19% of the input bytes is 0.536, which means that the Level-8 encoder operates at a rate which is only 0.5% below the effective capacity.

The discussion in this section has concentrated on the design of the Level-8 encoder. Reduction in the number of states (at the expense of a more limited DC control) can be obtained by merging and deleting states according to Table 7. The incoming edges to a deleted state $u$ are redirected to one of the states $v$ with the property that every codeword sequence that can be generated from $v$ can also be generated from $u$ (and, in fact, it can be generated from $u$ for the same sequence of input bytes).

Similar design tools can be applied to obtain the encoder of Section 2, as well as encoders for other certain $(d, k)$-RLL constraints.

# 5 Acknowledgment

I would like to thank Josh Hogan from the Storage Technology Department at HP Labs for his invaluable contribution to this work.

# 6 Appendix: Alternate (2,10)-RLL encoder

This appendix contains an alternate construction of a $(2, 10)$-RLL encoder in which the encoded data never contains the bit pattern 00010001000100010001000100010001, starting at any bit phase. As mentioned in Section 2, this property allows having synchronizing words that consist of this pattern, followed by auxiliary information which may be required during synchronization.

The encoder has four states, S0, S1, S2-5, and S6-8, which are similar to those of the encoder in Section 2. The encoder table consists of 547 codewords, each having 16 bits. Table 13 lists the codewords in hexadecimal form. The threshold values are summarized in Table 10.

| State | Thresholds (decimal) | | Prefixes (binary) | | |
|---|---|---|---|---|---|
| | T1 | T2 | $0 \leq b <$ T1 | T1 $\leq b <$ T2 | T2 $\leq b < 256$ |
| S0 | 000 | 000 | $\phi$ | $\phi$ | 00 |
| S1 | 008 | 121 | 01 | 01 or 00 | 00 |
| S2-5 | 035 | 054 | 10 or 01 | 01 | 01 or 00 |
| S6-8 | 035 | 177 | 10 or 01 | 01 | 01 or 00 |

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

The exclusion of the binary pattern 00010001000100010001000100010001 from the encoded stream is achieved in the following manner:

(a) The (hexadecimal) words 1111, 4444, and 8888 do not appear in the table. Thus, the hexadecimal patterns 1111111, 4444444, or 8888888 can never appear in the encoded stream when the latter is represented as a hexadecimal sequence in which codeword boundaries are aligned with the hexadecimal symbols.

(b) The codewords 2220, 2221, 2222, and 2224 appear at the beginning of the table, and, due to the choice of T1(S1), those codewords cannot be generated from state S1. Since any codeword of the form $xxx2$ terminates in state S1, no two consecutive codewords in the encoded stream can have the form $xxx2$ $222x$. (The removal of 2222 from the table might seem to be a simpler solution; however, the words 1111 and 2222 cannot both be removed from the table, or else there would not be 256 codewords that could be generated from state S0.)

The percentage of bytes that allow for DC control in the resulting encoder is 48.6%, which is less by about 1% than the DC control attained by the encoder of Section 2.

# 7 References


[ACH83] 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.

[AHM82] R.L. ADLER, M. HASSNER, J. MOUSSOURIS, *Method and apparatus for generating a noiseless sliding block code for a* $(1, 7)$ *channel with rate 2/3,* US patent 4,413,251 (1982).

[Fra70] P.A. FRANASZEK, *Sequence-state methods for run-length-limited coding, IBM J. Res. Develop.,* 14 (1970), 376–383.

[GuF94] J. GU, T. FUJA, *A new approach to constructing optimal block codes for runlength-limited channels, IEEE Trans. Inform. Theory,* 40 (1994), 774–785.

[Imm91] K.A.S. IMMINK, *Coding Techniques for Digital Recorders,* Prentice Hall, New York, 1991.

[Imm95] K.A.S. IMMINK, *EFMPlus: The coding format of the multimedia compact disc, Proc. 16th Symp. Inform. Theory in the Benelux,* Nieuwerkerk a/d Yssel, May 1995.

[IO85] K.A.S. IMMINK, H. OGAWA, *Method for encoding binary data,* US patent 4,501,000 (1985).

[MM77] J.C. MALLINSON, J.W. MILLER, *Optimal codes for digital magnetic recording, Radio and Elec. Eng.,* 47 (1977), 172–176.

[Mill63] A. MILLER, *Transmission system,* US patent 3,108,261 (1963).

[Mill77] J.W. MILLER, US patent 4,027,335 (1977).

[Patel75] A.M. PATEL, *Zero-modulation encoding in magnetic recording, IBM J. Res. Develop.,* 19 (1975), 366–378.

[MRS95] B.H. MARCUS, R.M. ROTH, P.H. SIEGEL, *Constrained systems and coding for recording channels,* to appear in *Handbook of Coding Theory,* R.A. Brualdi, W.C. Huffman, V. Pless (Eds.), Elsevier, Amsterdam. Also appeared as TR 839, Computer Science Department, Technion, Haifa, Israel, December 1994.

[MSW92] 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 (decimal) | Contents of table (hexadecimal) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 0021 | 0022 | 0024 | 0020 | 0041 | 0081 | 0101 | 0201 | 0249 | 0049 |
| 10 | 0089 | 0091 | 0109 | 0111 | 0121 | 0042 | 0082 | 0102 | 0202 | 0092 |
| 20 | 0112 | 0122 | 0044 | 0048 | 0084 | 0088 | 0090 | 0124 | 0224 | 0244 |
| 30 | 0248 | 0040 | 0080 | 0100 | 0209 | 0211 | 0221 | 0241 | 0212 | 0222 |
| 40 | 0104 | 0108 | 0110 | 0120 | 0204 | 0208 | 0210 | 0220 | 0240 | 0242 |
| 50 | 0401 | 0449 | 0489 | 0491 | 0801 | 0849 | 0889 | 0891 | 0909 | 0911 |
| 60 | 0921 | 1049 | 1089 | 1091 | 0409 | 0411 | 0421 | 0441 | 0481 | 0809 |
| 70 | 0811 | 0821 | 0841 | 0881 | 0901 | 0402 | 0492 | 0802 | 0892 | 0912 |
| 80 | 0922 | 1002 | 0412 | 0422 | 0442 | 0482 | 0812 | 0822 | 0842 | 0882 |
| 90 | 0902 | 0404 | 0408 | 0410 | 0420 | 0804 | 0808 | 0810 | 0820 | 0924 |
| 100 | 0424 | 0444 | 0448 | 0484 | 0488 | 0490 | 0824 | 0844 | 0848 | 0884 |
| 110 | 0888 | 0890 | 0904 | 0908 | 0910 | 0920 | 1024 | 1044 | 0440 | 0480 |
| 120 | 0840 | 0880 | 0900 | 1109 | 1111 | 1121 | 1209 | 1211 | 1221 | 1241 |
| 130 | 1009 | 1011 | 1021 | 1041 | 1081 | 1101 | 1201 | 1249 | 1092 | 1112 |
| 140 | 1122 | 1212 | 1222 | 1242 | 1012 | 1022 | 1042 | 1082 | 1102 | 1202 |
| 150 | 1004 | 1008 | 1010 | 1020 | 1124 | 1224 | 1244 | 1248 | 1048 | 1084 |
| 160 | 1088 | 1090 | 1104 | 1108 | 1110 | 1120 | 1204 | 1208 | 1210 | 1220 |
| 170 | 1040 | 1080 | 1100 | 1240 | 2049 | 2089 | 2091 | 2109 | 2111 | 2121 |
| 180 | 2209 | 2211 | 2221 | 2241 | 2409 | 2411 | 2421 | 2441 | 2481 | 2009 |
| 190 | 2011 | 2021 | 2041 | 2081 | 2101 | 2201 | 2249 | 2401 | 2449 | 2489 |
| 200 | 2092 | 2112 | 2122 | 2212 | 2222 | 2242 | 2412 | 2422 | 2442 | 2482 |
| 210 | 2012 | 2022 | 2042 | 2082 | 2102 | 2202 | 2402 | 2492 | 2004 | 2008 |
| 220 | 2010 | 2020 | 2124 | 2224 | 2244 | 2248 | 2424 | 2444 | 2448 | 2484 |
| 230 | 2488 | 2490 | 2024 | 2044 | 2048 | 2084 | 2088 | 2090 | 2104 | 2108 |
| 240 | 2110 | 2120 | 2204 | 2208 | 2210 | 2220 | 2404 | 2408 | 2410 | 2420 |
| 250 | 2040 | 2080 | 2100 | 2240 | 2440 | 2480 | 2491 | 4022 | 4024 | 4020 |
| 260 | 4041 | 4081 | 4101 | 4201 | 4249 | 4049 | 4089 | 4091 | 4109 | 4111 |
| 270 | 4121 | 4042 | 4082 | 4102 | 4202 | 4092 | 4112 | 4122 | 4044 | 4048 |
| 280 | 4084 | 4088 | 4090 | 4124 | 4224 | 4244 | 4248 | 4040 | 4080 | 4100 |
| 290 | 4209 | 4211 | 4221 | 4241 | 4212 | 4222 | 4104 | 4108 | 4110 | 4120 |
| 300 | 4204 | 4208 | 4210 | 4220 | 4240 | 4242 | 4401 | 4449 | 4489 | 4491 |
| 310 | 4801 | 4849 | 4889 | 4891 | 4909 | 4911 | 4921 | 4009 | 4021 | 4011 |
| 320 | 4409 | 4411 | 4421 | 4441 | 4481 | 4809 | 4811 | 4821 | 4841 | 4881 |
| 330 | 4901 | 4402 | 4492 | 4802 | 4892 | 4912 | 4922 | 4012 | 4412 | 4422 |
| 340 | 4442 | 4482 | 4812 | 4822 | 4842 | 4882 | 4902 | 4404 | 4408 | 4410 |
| 350 | 4420 | 4804 | 4808 | 4810 | 4820 | 4924 | 4424 | 4444 | 4448 | 4484 |
| 360 | 4488 | 4490 | 4824 | 4844 | 4848 | 4884 | 4888 | 4890 | 4904 | 4908 |
| 370 | 4910 | 4920 | 4010 | 4008 | 4440 | 4480 | 4840 | 4880 | 4900 | 9109 |
| 380 | 9111 | 9121 | 9209 | 9211 | 9221 | 9241 | 9009 | 9011 | 9021 | 9041 |
| 390 | 9081 | 9101 | 9201 | 9249 | 9092 | 9112 | 9122 | 9212 | 9222 | 9242 |
| 400 | 9012 | 9022 | 9042 | 9082 | 9102 | 9202 | 9004 | 9008 | 9010 | 9020 |
| 410 | 8108 | 9224 | 8090 | 9248 | 9048 | 9084 | 9088 | 9090 | 9104 | 9108 |
| 420 | 9110 | 9120 | 9204 | 9208 | 9210 | 9220 | 9040 | 9080 | 9100 | 9240 |
| 430 | 8449 | 8489 | 8491 | 8909 | 8911 | 8921 | 8201 | 8011 | 8021 | 8041 |
| 440 | 8401 | 9091 | 8801 | 9049 | 8081 | 8409 | 8411 | 8221 | 8241 | 8481 |
| 450 | 8901 | 8209 | 8049 | 8441 | 8841 | 8089 | 8492 | 8912 | 8922 | 8012 |
| 460 | 8022 | 8042 | 8402 | 9002 | 8802 | 8082 | 8412 | 8422 | 8442 | 8482 |
| 470 | 8902 | 8212 | 8842 | 8092 | 8204 | 8208 | 8110 | 8120 | 8924 | 8220 |
| 480 | 8044 | 8048 | 8420 | 8404 | 8210 | 8084 | 8088 | 8410 | 8824 | 8844 |
| 490 | 8848 | 8884 | 8888 | 8890 | 8904 | 8908 | 8910 | 8920 | 8224 | 8248 |
| 500 | 8010 | 8020 | 8424 | 8448 | 8490 | 8124 | 8840 | 8880 | 8900 | 8040 |
| 510 | 8100 | 8080 | 8091 | 8122 | 9024 | 8024 | 8811 | 8881 | 8109 | 8211 |
| 520 | 8809 | 8849 | 9089 | 8891 | 8101 | 8889 | 8249 | 8242 | 8882 | 8112 |
| 530 | 8222 | 8892 | 8102 | 8202 | 8444 | 9044 | 8484 | 8488 | 8244 | 8104 |
| 540 | 8820 | 8804 | 8808 | 8240 | 8480 | 8440 | | | | |

Table 11: Table of the $(2, 10)$-RLL encoder.

| address | Contents of table (hexadecimal) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| (decimal) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 0004 | 0004 | 0008 | 0008 | 0009 | 0011 | 0010 | 0010 | 0012 | 0021 |
| 10 | 0024 | 0024 | 0020 | 0020 | 0022 | 0041 | 0081 | 0049 | 0089 | 0091 |
| 20 | 0042 | 0082 | 0092 | 0080 | 0044 | 0044 | 0084 | 0084 | 0048 | 0048 |
| 30 | 0088 | 0088 | 0090 | 0090 | 0040 | 0040 | 0101 | 0102 | 0202 | 0201 |
| 40 | 0249 | 0109 | 0111 | 0121 | 0209 | 0211 | 0221 | 0241 | 0112 | 0122 |
| 50 | 0212 | 0222 | 0242 | 0100 | 0104 | 0104 | 0204 | 0204 | 0124 | 0124 |
| 60 | 0224 | 0224 | 0244 | 0244 | 0108 | 0108 | 0208 | 0208 | 0248 | 0248 |
| 70 | 0110 | 0110 | 0210 | 0210 | 0120 | 0120 | 0220 | 0220 | 0240 | 0240 |
| 80 | 0401 | 0449 | 0489 | 0491 | 0409 | 0411 | 0421 | 0441 | 0481 | 0809 |
| 90 | 0402 | 0492 | 0412 | 0422 | 0442 | 0482 | 0812 | 0480 | 0404 | 0404 |
| 100 | 0424 | 0424 | 0444 | 0444 | 0484 | 0484 | 0408 | 0408 | 0448 | 0448 |
| 110 | 0488 | 0488 | 0410 | 0410 | 0490 | 0490 | 0420 | 0420 | 0440 | 0440 |
| 120 | 0801 | 0849 | 0889 | 0891 | 0909 | 0911 | 0921 | 1001 | 1049 | 1089 |
| 130 | 1091 | 1109 | 1111 | 1121 | 1209 | 1211 | 1221 | 1241 | 0811 | 0821 |
| 140 | 0841 | 0881 | 0901 | 1009 | 1011 | 1021 | 1041 | 1081 | 1101 | 1201 |
| 150 | 1249 | 0802 | 0892 | 0912 | 0922 | 1002 | 1092 | 1112 | 1122 | 1212 |
| 160 | 1222 | 1242 | 0822 | 0842 | 0882 | 0902 | 1012 | 1022 | 1042 | 1082 |
| 170 | 1102 | 1202 | 0880 | 0900 | 1080 | 1100 | 0804 | 0804 | 0924 | 0924 |
| 180 | 1004 | 1004 | 1124 | 1124 | 1224 | 1224 | 1244 | 1244 | 0824 | 0824 |
| 190 | 0844 | 0844 | 0884 | 0884 | 0904 | 0904 | 1024 | 1024 | 1044 | 1044 |
| 200 | 1084 | 1084 | 1104 | 1104 | 1204 | 1204 | 0808 | 0808 | 1008 | 1008 |
| 210 | 1248 | 1248 | 0848 | 0848 | 0888 | 0888 | 0908 | 0908 | 1048 | 1048 |
| 220 | 1088 | 1088 | 1108 | 1108 | 1208 | 1208 | 0810 | 0810 | 1010 | 1010 |
| 230 | 0890 | 0890 | 0910 | 0910 | 1090 | 1090 | 1110 | 1110 | 1210 | 1210 |
| 240 | 0820 | 0820 | 0840 | 0840 | 1020 | 1020 | 1040 | 1040 | 0920 | 0920 |
| 250 | 1120 | 1120 | 1220 | 1220 | 1240 | 1240 | 2004 | 2004 | 2008 | 2008 |
| 260 | 2009 | 2011 | 2010 | 2010 | 2012 | 2021 | 2024 | 2024 | 2020 | 2020 |
| 270 | 2022 | 2041 | 2081 | 2001 | 2049 | 2089 | 2042 | 2082 | 2002 | 2080 |
| 280 | 2044 | 2044 | 2084 | 2084 | 2048 | 2048 | 2088 | 2088 | 2090 | 2090 |
| 290 | 2040 | 2040 | 2101 | 2102 | 2202 | 2201 | 2249 | 2091 | 2109 | 2111 |
| 300 | 2121 | 2209 | 2211 | 2221 | 2092 | 2112 | 2122 | 2212 | 2222 | 2100 |
| 310 | 2104 | 2104 | 2204 | 2204 | 2124 | 2124 | 2224 | 2224 | 2244 | 2244 |
| 320 | 2108 | 2108 | 2208 | 2208 | 2248 | 2248 | 2110 | 2110 | 2210 | 2210 |
| 330 | 2120 | 2120 | 2220 | 2220 | 2240 | 2240 | 2401 | 2449 | 2489 | 2491 |
| 340 | 2241 | 2409 | 2411 | 2421 | 2441 | 2481 | 2402 | 2492 | 2242 | 2412 |
| 350 | 2422 | 2442 | 2482 | 2480 | 2404 | 2404 | 2424 | 2424 | 2444 | 2444 |
| 360 | 2484 | 2484 | 2408 | 2408 | 2448 | 2448 | 2488 | 2488 | 2410 | 2410 |
| 370 | 2490 | 2490 | 2420 | 2420 | 2440 | 2440 | 4009 | 4011 | 4021 | 4041 |
| 380 | 4081 | 4101 | 4201 | 4249 | 4401 | 4449 | 4489 | 4491 | 4801 | 4849 |
| 390 | 4889 | 4891 | 4909 | 4911 | 4049 | 4089 | 4091 | 4109 | 4111 | 4121 |
| 400 | 4209 | 4211 | 4221 | 4241 | 4409 | 4411 | 4421 | 4012 | 4022 | 4042 |
| 410 | 4082 | 4102 | 4202 | 4402 | 4492 | 4802 | 4892 | 4912 | 4002 | 4092 |
| 420 | 4112 | 4122 | 4212 | 4222 | 4242 | 4412 | 4422 | 4442 | 4480 | 4880 |
| 430 | 4900 | 4080 | 4024 | 4024 | 4044 | 4044 | 4084 | 4084 | 4104 | 4104 |
| 440 | 4204 | 4204 | 4404 | 4404 | 4004 | 4004 | 4124 | 4124 | 4224 | 4224 |
| 450 | 4244 | 4244 | 4424 | 4424 | 4444 | 4444 | 4484 | 4484 | 4824 | 4824 |
| 460 | 4844 | 4844 | 4048 | 4048 | 4088 | 4088 | 4108 | 4108 | 4008 | 4008 |
| 470 | 4248 | 4248 | 4448 | 4448 | 4488 | 4488 | 4848 | 4848 | 4888 | 4888 |
| 480 | 4908 | 4908 | 4090 | 4090 | 4110 | 4110 | 4210 | 4210 | 4010 | 4010 |
| 490 | 4490 | 4490 | 4890 | 4890 | 4910 | 4910 | 4120 | 4120 | 4220 | 4220 |
| 500 | 4240 | 4240 | 4420 | 4420 | 4440 | 4440 | 4020 | 4020 | 4040 | 4040 |
| 510 | 4920 | 4920 | 4804 | 4804 | 4208 | 4208 | 4441 | 4481 | 4410 | 4410 |
| 520 | 4482 | 4809 | 4884 | 4884 | 4820 | 4820 | 4812 | 4811 | 4901 | 4841 |
| 530 | 4881 | 4921 | 4882 | 4902 | 4922 | 4100 | 4904 | 4904 | 4924 | 4924 |
| 540 | 4408 | 4408 | 4808 | 4808 | 4810 | 4810 | 4840 | 4840 | 4821 | 4822 |
| 550 | 4842 | | | | | | | | | |

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

| address (decimal) | Contents of table (hexadecimal) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 0021 | 0022 | 0024 | 0020 | 2221 | 2222 | 2224 | 2220 | 0041 | 0081 |
| 10 | 0101 | 0201 | 0249 | 0049 | 0089 | 0091 | 0109 | 0111 | 0121 | 0042 |
| 20 | 0082 | 0102 | 0202 | 0092 | 0112 | 0122 | 0044 | 0048 | 0124 | 0224 |
| 30 | 0244 | 0248 | 0040 | 0080 | 0100 | 0209 | 0211 | 0212 | 0084 | 0088 |
| 40 | 0090 | 0104 | 0108 | 0110 | 0120 | 0204 | 0208 | 0210 | 0220 | 0221 |
| 50 | 0241 | 0222 | 0242 | 0240 | 0401 | 0449 | 0489 | 0491 | 0801 | 0849 |
| 60 | 0889 | 0891 | 0909 | 0911 | 0921 | 1049 | 0409 | 0411 | 0421 | 0441 |
| 70 | 0481 | 0809 | 0811 | 0821 | 0841 | 0881 | 0901 | 0402 | 0492 | 0802 |
| 80 | 0892 | 0912 | 0922 | 0412 | 0422 | 0442 | 0482 | 0812 | 0822 | 0842 |
| 90 | 0882 | 0902 | 0404 | 0408 | 0410 | 0420 | 0804 | 0808 | 0810 | 0820 |
| 100 | 0424 | 0444 | 0448 | 0484 | 0488 | 0490 | 0824 | 0844 | 0848 | 0884 |
| 110 | 0888 | 0890 | 0904 | 0908 | 0910 | 0920 | 0440 | 0480 | 0840 | 0880 |
| 120 | 0900 | 1089 | 1091 | 1109 | 1241 | 1121 | 1209 | 1211 | 1221 | 1009 |
| 130 | 1011 | 1021 | 1041 | 1081 | 1101 | 1201 | 1249 | 1002 | 1092 | 1112 |
| 140 | 1122 | 1212 | 1222 | 1242 | 1012 | 1022 | 1042 | 1082 | 1102 | 1202 |
| 150 | 0924 | 1004 | 1008 | 1010 | 1020 | 1124 | 1224 | 1244 | 1248 | 1024 |
| 160 | 1044 | 1048 | 1084 | 1088 | 1090 | 1104 | 1108 | 1110 | 1120 | 1204 |
| 170 | 1208 | 1210 | 1220 | 1040 | 1080 | 1100 | 1240 | 2089 | 2091 | 2109 |
| 180 | 2111 | 2121 | 2209 | 2211 | 2049 | 2241 | 2409 | 2411 | 2421 | 2441 |
| 190 | 2481 | 2009 | 2011 | 2021 | 2041 | 2081 | 2101 | 2201 | 2249 | 2401 |
| 200 | 2449 | 2489 | 2491 | 2112 | 2122 | 2212 | 2092 | 2242 | 2412 | 2422 |
| 210 | 2442 | 2482 | 2012 | 2022 | 2042 | 2082 | 2102 | 2202 | 2402 | 2492 |
| 220 | 2008 | 2010 | 2020 | 2124 | 2004 | 2244 | 2248 | 2424 | 2444 | 2448 |
| 230 | 2484 | 2488 | 2490 | 2044 | 2048 | 2084 | 2088 | 2090 | 2104 | 2108 |
| 240 | 2110 | 2120 | 2204 | 2208 | 2210 | 2024 | 2404 | 2408 | 2410 | 2420 |
| 250 | 2040 | 2080 | 2100 | 2240 | 2440 | 2480 | 4021 | 4022 | 4024 | 4020 |
| 260 | 4009 | 4012 | 4044 | 4124 | 4041 | 4081 | 4101 | 4201 | 4249 | 4049 |
| 270 | 4089 | 4091 | 4109 | 4111 | 4121 | 4042 | 4082 | 4102 | 4202 | 4092 |
| 280 | 4112 | 4122 | 4048 | 4088 | 4224 | 4244 | 4248 | 4008 | 4040 | 4080 |
| 290 | 4100 | 4209 | 4211 | 4212 | 4084 | 4090 | 4110 | 4104 | 4108 | 4120 |
| 300 | 4220 | 4204 | 4208 | 4210 | 4420 | 4221 | 4241 | 4222 | 4242 | 4240 |
| 310 | 4401 | 4449 | 4489 | 4491 | 4801 | 4849 | 4889 | 4891 | 4909 | 4911 |
| 320 | 4921 | 4011 | 4409 | 4411 | 4421 | 4441 | 4481 | 4809 | 4811 | 4821 |
| 330 | 4841 | 4881 | 4901 | 4402 | 4492 | 4802 | 4892 | 4912 | 4922 | 4412 |
| 340 | 4422 | 4442 | 4482 | 4812 | 4822 | 4842 | 4882 | 4902 | 4404 | 4408 |
| 350 | 4410 | 4820 | 4804 | 4808 | 4810 | 4924 | 4424 | 4010 | 4448 | 4484 |
| 360 | 4488 | 4490 | 4824 | 4844 | 4848 | 4884 | 4888 | 4890 | 4904 | 4908 |
| 370 | 4910 | 4920 | 4440 | 4480 | 4840 | 4880 | 4900 | 9089 | 9091 | 9109 |
| 380 | 9111 | 9121 | 9209 | 9211 | 9221 | 9009 | 9011 | 9021 | 9041 | 9081 |
| 390 | 9101 | 9201 | 9249 | 9002 | 9092 | 9112 | 9122 | 9212 | 9222 | 9242 |
| 400 | 9012 | 9022 | 9042 | 9082 | 9102 | 9202 | 8924 | 9004 | 9008 | 9010 |
| 410 | 9020 | 9124 | 9224 | 9244 | 9248 | 9024 | 9044 | 9048 | 9084 | 9088 |
| 420 | 9090 | 9104 | 9108 | 9110 | 9120 | 9204 | 9208 | 9210 | 9220 | 9040 |
| 430 | 9080 | 9100 | 9240 | 8489 | 8491 | 8909 | 8911 | 8921 | 8201 | 8011 |
| 440 | 8021 | 8041 | 8449 | 8401 | 8801 | 8849 | 8081 | 8409 | 8411 | 8221 |
| 450 | 8241 | 8481 | 8901 | 8209 | 8049 | 8441 | 8841 | 8089 | 8091 | 8912 |
| 460 | 8922 | 8012 | 8022 | 8042 | 8492 | 8402 | 8802 | 8082 | 8412 | 8422 |
| 470 | 8442 | 8482 | 8902 | 8212 | 8842 | 8092 | 8208 | 8210 | 8120 | 8024 |
| 480 | 8220 | 8204 | 8048 | 8420 | 8404 | 8408 | 8084 | 8088 | 8410 | 8244 |
| 490 | 8848 | 8884 | 8844 | 8010 | 8904 | 8908 | 8910 | 8920 | 8224 | 8248 |
| 500 | 8020 | 8124 | 8424 | 8448 | 8490 | 8484 | 8840 | 8880 | 8900 | 8040 |
| 510 | 8100 | 8080 | 8821 | 8122 | 8824 | 8820 | 8809 | 8112 | 8444 | 8104 |
| 520 | 8811 | 8881 | 8109 | 8211 | 8121 | 8249 | 8889 | 8891 | 8101 | 9049 |
| 530 | 9241 | 8242 | 8882 | 8822 | 8222 | 8892 | 8102 | 8202 | 8488 | 8890 |
| 540 | 8804 | 8044 | 8808 | 8108 | 8240 | 8480 | 8440 | | | |

Table 13: Table of the alternate $(2, 10)$-RLL encoder.