Nested Block Decodable Run Length Limited Codes

Joshua Hogan, Ron M. Roth, Gitit Ruckenstein
HP Laboratories Palo Alto
HPL-2000-91
July 10th, 2000*

block decodable encoders, deterministic encoders, nested encoders, run length-limited constraints

Consider a (d_1, k_1)-RLL constraint that is contained in a (d_2, k_2)-RLL constraint, where $k_1 \geq 2d_1$ and $d_2 > 0$, and fix a codeword length $q > k_2$. It is shown that whenever there exist block decodable encoders with codeword length q for those two constraints, there exist such encoders where one is a subgraph of the other; furthermore, both encoders can be decoded by essentially the same decoder. Specifically, a (d_1, k_1)-RLL constrained word is decoded by first using a block decoder of the (d_2, k_2)-RLL encoder, and then applying a certain function to the output of that decoder.

* Internal Accession Date Only
Approved for External Publication

Nested Block Decodable Runlength Limited Codes

Josh Hogan* Ron M. Roth† Gitit Ruckenstein‡

July 10, 2000

Abstract

Consider a (d_1, k_1)-RLL constraint that is contained in a (d_2, k_2)-RLL constraint, where $k_1 \geq 2d_1$ and $d_2 > 0$, and fix a codeword length $q > k_2$. It is shown that whenever there exist block decodable encoders with codeword length q for those two constraints, there exist such encoders where one is a subgraph of the other; furthermore, both encoders can be decoded by essentially the same decoder. Specifically, a (d_1, k_1)-RLL constrained word is decoded by first using a block decoder of the (d_2, k_2)-RLL encoder, and then applying a certain function to the output of that decoder.

Keywords: Block decodable encoders; Deterministic encoders; Nested encoders; Runlength-limited constraints.

1 Introduction

In secondary storage systems, it is usually required that the recorded binary sequences satisfy certain constraints. In the most common types of constraints, lower and upper bounds are set on the runs of '0's in binary words. Hereafter, by a 'runlength' in a binary

*Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304, USA.

†Computer Science Department, Technion, Haifa 32000, Israel. Work done in part while on sabbatical leave from Technion at Hewlett-Packard Laboratories, Palo Alto, CA 94304, USA, and in part while at Hewlett-Packard Laboratories—Israel, Haifa, Israel.

‡Computer Science Department, Technion, Haifa 32000, Israel. Work done as a summer student of Hewlett-Packard Laboratories—Israel and Hewlett-Packard Laboratories, Palo Alto, CA 94304, USA.

word we mean the length of a run of '0's which is delimited either by '1's or by the beginning or end of the word. For instance, the runlengths in the word

$$10000100010000000000010 \tag{1}$$

are 0, 4, 3, 10, and 1. The (d, k)-runlength limited (RLL) constraint consists of all binary words in which each runlength is at most k and—with the exception of the first and last runlengths—at least d. For example, the current compact disk and DVD standards use the constraint $(d, k) = (2, 10)$ (the word in (1) satisfies this constraint). The set of all finite binary words that satisfy the (d, k)-RLL constraint will be denoted by $\mathcal{S}_{(d,k)}$.

Arbitrary input sequences need to be encoded into sequences that satisfy the constraint. An encoding model that is commonly used for this matter is that of a finite-state encoder at a fixed rate $p : q$, where the input binary sequence is divided into blocks of a fixed length p, and each such block is mapped, in a state-dependent manner, into a codeword of length q. The sequence of generated codewords forms a word that satisfies the constraint. A primary requirement from encoders is that we should be able to decode (reconstruct) the input binary sequence from the output constrained sequence.

Of particular interest are *block decodable encoders*. Such encoders can be decoded by a *block decoder*, which maps every codeword of length q into the respective p-block, independently of the context of that codeword within the output sequence of generated codewords. Whether a block decodable encoder exists depends on the specific constraint—i.e., on d and k—and on the parameters p and q. Block decodable encoders are preferable due to their simple decoding structure and their immunity against error propagation.

In the current emerging technology and development of erasable and writable dense optical disks, we may face the scenario where recorders will differ in their writing capabilities, thereby requiring different constraints. For example, home recorders might use an encoder $\mathcal{E}_1$ at rate $p_1 : q_1$ for a (d_1, k_1)-RLL constraint; on the other hand, manufacturers of optical disks may be able to record data using an encoder $\mathcal{E}_2$ at a higher rate $p_2 : q_2$ for a (d_2, k_2)-RLL constraint where $d_2 \leq d_1$ and $k_2 \geq k_1$.

In spite of the different encoders, we would still like a disk player to have the capability of decoding both encoding schemes. As an alternative approach to having on board a separate decoder for each encoder, the authors have recently suggested in [5] that the encoders $\mathcal{E}_1$ and $\mathcal{E}_2$ be designed so that their decoders can be combined to a great extent. To this end, we assume that $q_1 = q_2$ (and so $p_1 \leq p_2$). A decoder $\mathcal{D}_2$ of $\mathcal{E}_2$ will decode—as before—sequences of the (d_2, k_2)-RLL constraint by dividing each sequence of output symbols into non-overlapping words of length q, and mapping each such word into an input binary p_2-block. A decoder $\mathcal{D}_1$ of $\mathcal{E}_1$ will be obtained by first applying $\mathcal{D}_2$ to the sequence of the (d_1, k_1)-RLL constraint to produce a sequence of binary p_2-blocks; then,

a combinational circuit (function) ψ will map each decoded input p_2-block into an input p_1-block (see Figure 1). If such a combined decoding scheme exists, we will say that the encoder $\mathcal{E}_1$ is *(block) observable* from $\mathcal{E}_2$ and that $(\mathcal{E}_1, \mathcal{E}_2)$ is an *observable pair.*

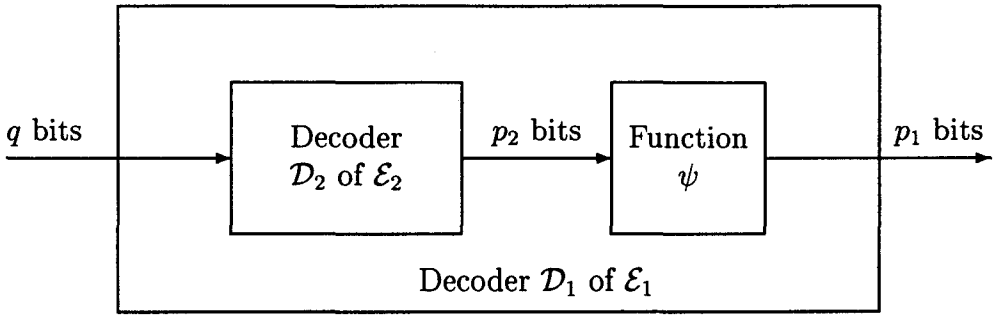

Figure 1: Decoding of an observable pair $(\mathcal{E}_1, \mathcal{E}_2)$.

The main result of this work is presented in Sections 3 and 4, where we consider any two (d_i, k_i)-RLL constraints that satisfy

$$k_2 \geq k_1 \geq 2d_1 \geq 2d_2 > 0 \, .$$

Given two respective rates $p_i : q$ where $q > k_2$, we show that whenever there exist block decodable encoders for those constraints at the given rates, one can construct such encoders that form an observable pair. Furthermore, the encoders are *nested*: the underlying graph presentation of one encoder is contained in the other. Our result relies to a great extent on the characterization of Gu and Fuja in [4] for the range of parameters for which there exist (individual) block decodable RLL encoders.

An example for the case $(d_1, k_1) = (3, 10)$ and $(d_2, k_2) = (2, 10)$, which may have practical applications, is presented in the appendix.

The next section contains the necessary background material.

2 Background

The definitions we provide next are taken from [5] and [8] and are tailored for the special case of RLL constraints.

2.1 Finite-state encoders

A *(finite-state) encoder for* $\mathcal{S}_{(d,k)}$ *at rate* $p : q$ is a finite labeled directed graph $\mathcal{E} = (V, E, L)$ with a nonempty finite set V of states, a set E of edges, and an edge labeling $L : E \to \{0,1\}^q$ such that the following three conditions hold: (i) there are 2^p outgoing edges from each state in $\mathcal{E}$; (ii) the concatenation of labels of paths in $\mathcal{E}$ are all words in $\mathcal{S}_{(d,k)}$; and (iii) $\mathcal{E}$ is lossless, namely, any two distinct paths with the same initial state and terminal state generate different words.

A label of an edge will be referred to as a *codeword.* The set of all codewords that actually label edges of $\mathcal{E}$ will be denoted by $\Sigma(\mathcal{E})$.

An *encoder for* $\mathcal{S}_{(d,k)}$ *at rate* $p : q$ exists if and only if

$$p/q \leq \lim_{\ell \to \infty} \frac{1}{\ell} \log_2 |\mathcal{S}_{(d,k)} \cap \{0,1\}^\ell| \,,$$

the right-hand side being the *capacity* of $\mathcal{S}_{(d,k)}$ [1], [10]. It is known that the capacity of $\mathcal{S}_{(d,k)}$ equals the logarithm to base 2 of the largest positive root of the equation

$$z^{k+1} - \sum_{j=0}^{k-d} z^j = 0 \,.$$

Table 5.4 in [6, p. 91] lists the capacity values of several (d,k)-RLL constraints.

A *tagged* encoder for $\mathcal{S}_{(d,k)}$ at rate $p : q$ is an encoder $\mathcal{E} = (V, E, L)$ where the outgoing edges from each state in $\mathcal{E}$ are assigned distinct *input tags* from $\{0,1\}^p$. For tagged encoders, we will extend the definition of the mapping $L : E \to \Sigma(\mathcal{E})$ to $L : V \times \{0,1\}^p \to \Sigma(\mathcal{E})$, where $L(u, \boldsymbol{s})$ is the label of the outgoing edge from state u that is tagged by $\boldsymbol{s}$.

Encoding with a tagged encoder $\mathcal{E}$ is carried out as follows. Given a positive integer ℓ, an unconstrained input binary word of length $p\ell$ is regarded as an input tag sequence $\boldsymbol{s} = \boldsymbol{s}_1 \boldsymbol{s}_2 \ldots \boldsymbol{s}_\ell$ of length ℓ over $\{0,1\}^p$. The sequence $\boldsymbol{s}$ defines a path of length ℓ in $\mathcal{E}$ starting at some prescribed initial state u_0, and the image (encoding) of $\boldsymbol{s}$ is the sequence of codewords $\boldsymbol{w} = \boldsymbol{w}_1 \boldsymbol{w}_2 \ldots \boldsymbol{w}_\ell$ that label the edges along that path. The sequence $\boldsymbol{w}$ is a word of length $q\ell$ in $\mathcal{S}_{(d,k)}$, and the lossless condition is needed in order to able to decode the sequence $\boldsymbol{s}$ from the word $\boldsymbol{w}$.

An encoder $\mathcal{E}$ is *deterministic* if the outgoing edges from any given state are labeled distinctly. An encoder $\mathcal{E}$ is *irreducible* if every state is accessible from any other state in $\mathcal{E}$.

Let $\mathcal{E}_1 = (V_1, E_1, L_1)$ and $\mathcal{E}_2 = (V_2, E_2, L_2)$ be encoders (for possibly different constraints at possibly different rates). We say that $\mathcal{E}_1$ is *nested* in $\mathcal{E}_2$—or that $(\mathcal{E}_1, \mathcal{E}_2)$ is a *nested pair*—if $V_1 \subseteq V_2$, $E_1 \subseteq E_2$, and L_1 is the restriction of L_2 to E_1.

Note that encoders are deterministic or nested according to whether their underlying *untagged* graphs are.

A tagged encoder is *block decodable* if edges labeled by the same codeword are tagged by the same input tag. A block decodable encoder $\mathcal{E}$ at rate $p : q$ can be decoded through a *block decoder* which is a function $\mathcal{D} : \Sigma(\mathcal{E}) \to \{0,1\}^p$ that maps a codeword $\boldsymbol{w}$ to the input tag assigned to any edge labeled $\boldsymbol{w}$. A block decodable encoder is necessarily deterministic (with respect to the *labels* of the edges).

2.2 Observable encoders

Let $\mathcal{E}_1$ be a block decodable encoder for $\mathcal{S}_{(d_1,k_1)}$ at rate $p_1 : q$ and let $\mathcal{E}_2$ be a block decodable encoder for $\mathcal{S}_{(d_2,k_2)}$ at rate $p_2 : q$ such that $\mathcal{S}_{(d_1,k_1)} \subseteq \mathcal{S}_{(d_2,k_2)}$. Except for degenerate cases we will always have $p_1 < p_2$. We say that $\mathcal{E}_1$ is *(block) observable* from $\mathcal{E}_2$ if the following two conditions hold:

1. every sequence of codewords that is generated by a path in $\mathcal{E}_1$ can also be generated in $\mathcal{E}_2$, and —
2. a block decoder $\mathcal{D}_1 : \Sigma(\mathcal{E}_1) \to \{0,1\}^{p_1}$ of $\mathcal{E}_1$ and a block decoder $\mathcal{D}_2 : \Sigma(\mathcal{E}_2) \to \{0,1\}^{p_2}$ of $\mathcal{E}_2$ are related for some function $\psi : \{0,1\}^{p_2} \to \{0,1\}^{p_1}$ by

$$\mathcal{D}_1(\boldsymbol{w}) = \psi(\mathcal{D}_2(\boldsymbol{w})) \quad \text{for every } \boldsymbol{w} \in \Sigma(\mathcal{E}_1).$$

We say that $\mathcal{E}_1$ is *weakly-observable* from $\mathcal{E}_2$ if condition 1 is relaxed to require only that $\Sigma(\mathcal{E}_1) \subseteq \Sigma(\mathcal{E}_2)$.

We mention that condition 1 (respectively, its relaxed form) allows to assume that when no errors are present, the decoder $\mathcal{D}_2$ is fed by sequences of codewords (respectively, by individual codewords) that can be generated by $\mathcal{E}_2$, even when $\mathcal{D}_2$ is applied to sequences generated by $\mathcal{E}_1$. This provides the possibility of incorporating an error detection mechanism into $\mathcal{D}_2$: such a mechanism will track sequences (respectively, codewords) that cannot be generated by $\mathcal{E}_2$.

The following example is essentially Example 3.1 in [5] and is repeated here for the purpose of demonstrating the definitions introduced in this section.

Example 2.1 The capacity of the $(2,3)$-RLL constraint is approximately 0.2878. An encoder for $\mathcal{S}_{(2,3)}$ at rate $1:4$ is shown in Figure 2, where the notation s/w on an edge indicates the input tag s and the codeword (label) w that are associated with that edge. This encoder, denoted $\mathcal{E}_1$, is a block decodable encoder for $\mathcal{S}_{(2,3)}$ at rate $1:4$, with the following block decoder $\mathcal{D}_1 : \Sigma(\mathcal{E}_1) \to \{0,1\}$:

$$\mathcal{D}_1(0001) = \mathcal{D}_1(0100) = 0 \quad \text{and} \quad \mathcal{D}_1(0010) = \mathcal{D}_1(1001) = 1 \, .$$

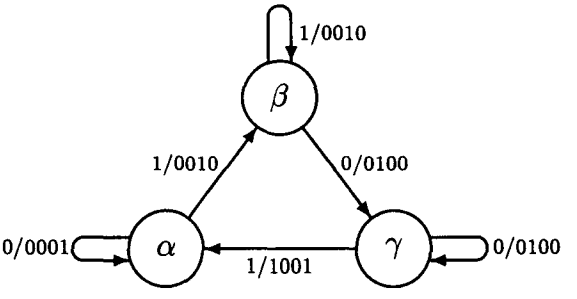

Figure 2: Rate $1:4$ finite-state encoder for $\mathcal{S}_{(2,3)}$.

The capacity of the $(1,3)$-RLL constraint is approximately 0.5515, and an encoder for $\mathcal{S}_{(1,3)}$ at rate $2:4$ is shown in Figure 3. This encoder, denoted $\mathcal{E}_2$, is a block decodable encoder for $\mathcal{S}_{(1,3)}$ at rate $2:4$, with a block decoder $\mathcal{D}_2 : \Sigma(\mathcal{E}_2) \to \{00,01,10,11\}$ defined by

$$\mathcal{D}_2(0001) = \mathcal{D}_2(1010) = 00, \; \mathcal{D}_2(0010) = \mathcal{D}_2(1001) = 01,$$
$$\mathcal{D}_2(0100) = 10, \text{ and } \mathcal{D}_2(0101) = 11.$$

It is easy to see that $\mathcal{E}_1$ is nested in $\mathcal{E}_2$. Furthermore, $\mathcal{E}_1$ is observable from $\mathcal{E}_2$. Indeed, let $\psi : \{00,01,10\} \to \{0,1\}$ be given by

$$\psi(00) = \psi(10) = 0 \quad \text{and} \quad \psi(01) = 1 \, .$$

Then, $\mathcal{D}_1(w) = \psi(\mathcal{D}_2(w))$ for every $w \in \{0001, 0010, 0100, 1001\}$.

We can transform $\mathcal{E}_2$ into another encoder for $\mathcal{S}_{(1,3)}$ by eliminating state β in $\mathcal{E}_2$ and redirecting all its incoming edges (excluding self-loops) into state γ. This yields a two-state encoder $\mathcal{E}_2'$ for $\mathcal{S}_{(1,3)}$ at rate $2:4$. Since $\Sigma(\mathcal{E}_2') = \Sigma(\mathcal{E}_2)$, the encoder $\mathcal{E}_2'$ can be

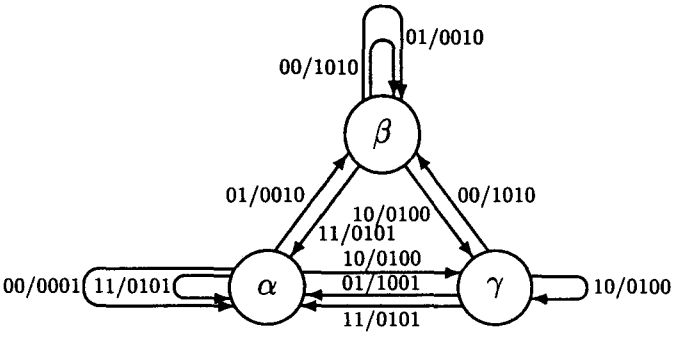

Figure 3: Rate 2 : 4 finite-state encoder for $\mathcal{S}_{(1,3)}$.

decoded by the same block decoder $\mathcal{D}_2$, and $\mathcal{E}_1$ is weakly-observable from $\mathcal{E}_2'$. On the other hand, $\mathcal{E}_1$ is not (fully) observable from $\mathcal{E}_2'$: the codeword sequence '0010 0010' can be generated in $\mathcal{E}_1$ but not in $\mathcal{E}_2'$. □

We refer the reader to [5] for a second example (Example 3.3 therein) of a block decodable encoder for $\mathcal{S}_{(3,7)}$ at rate 5 : 14, which is observable from a block decodable encoder for $\mathcal{S}_{(2,13)}$ at rate 7 : 14. □

In the appendix, we present a block decodable encoder for $\mathcal{S}_{(3,10)}$ at rate 6 : 16, which is weakly-observable from a block decodable encoder for $\mathcal{S}_{(2,10)}$ at rate 8 : 16. In addition to producing sequences that satisfy the respective constraints, these encoders also possess certain properties that allow for DC control (see [6, Section 2.5], [7], [9]).

It was shown by Franaszek in [3] that there exists a deterministic encoder for $\mathcal{S}_{(d,k)}$ at rate $p : q$ if and only if there exists a block decodable encoder for the same constraint at the same rate. Gu and Fuja obtained in [4] an almost-full characterization of the parameters p, q, d, and k for which there exist deterministic—and hence block decodable—encoders for $\mathcal{S}_{(d,k)}$ at rate $p : q$ (see Section 3 below).

In [5], a wider notion of observability was defined which applies to encoders that are not necessarily block decodable. It was then shown in [5] that for irreducible deterministic encoders, the requirement of having nested encoders is equivalent to having observable encoders (for the respective constraints at the same rates). On the other hand, it was demonstrated in [5, Example 3.2] that there are cases where there exist nested pairs of block decodable encoders, yet there are no observable pairs of block decodable encoders.

3 Observable encoders for $d_2 > 1$

Denote by $\mathbb{N}$ the set of all nonnegative integers. Given a (d, k)-RLL constraint, a positive integer q, and two sets $R, R' \subseteq \mathbb{N}$, let $\mathcal{L}(q, d, k, R, R')$ be the set of all words of length q in $\mathcal{S}_{(d,k)}$ in which the first runlength (of '0's) is in R and last runlength is in R'. For $r \in \mathbb{N}$, we will use the notation "$\geq r$" to denote the set $\{x \in \mathbb{N} : x \geq r\}$. Similar notations such as "$<r$" or "$=r$" will have their obvious meanings.

Next we present a construction of a block decodable encoder $\mathcal{E}_1$ for $\mathcal{S}_{(d_1,k_1)}$ at rate $p_1 : q$ that is observable from a block decodable encoder $\mathcal{E}_2$ for $\mathcal{S}_{(d_2,k_2)}$ at rate $p_2 : q$. Our construction assumes that the parameters d_i, k_i, and q satisfy the chain of inequalities

$$q > k_2 \geq k_1 \geq 2d_1 \geq 2d_2 > 0\,; \tag{2}$$

note that the second and fourth inequalities are necessary for having $\mathcal{S}_{(d_1,k_1)} \subseteq \mathcal{S}_{(d_2,k_2)}$. In addition, by [4] it follows that for $i = 1, 2$ it is required that

$$p_i \leq \log_2 |\mathcal{L}(q, d_i, k_i, \geq d_i, <k_i)|$$

in order to have block decodable encoders at rates $p_i : q$ for $\mathcal{S}_{(d_i,k_i)}$ (regardless of observability). To avoid degenerate cases, we will further assume that a strict inequality holds in (2) in either the second or the fourth inequality (or both), and that $p_1 < p_2$.

(The condition (2) is *sufficient* for our construction to work, and it may as well be the case that this condition can be relaxed, although we do not discuss such a relaxation here. Also observe that the first and third inequalities in (2) appear also in the Beenker-Immink construction [2], which can be viewed as a predecessor of the Gu-Fuja construction [4].)

We will assume in this section that $d_2 > 1$, deferring the treatment of the case $d_2 = 1$ to Section 4.

For $i = 1, 2$, we define $n_i = 2^{p_i}$, $\Upsilon_i = \{0, 1, \ldots, n_i - 1\}$, and $\Delta_i = k_i - d_i + 1$; so, $n_2 \geq 2n_1$ and $\Delta_2 > \Delta_1$. We adopt the convention that bit locations in a binary word are indexed starting with 0. The first runlength in a binary word $\boldsymbol{w}$ will be denoted by $\ell(\boldsymbol{w})$.

3.1 Encoding tables

Our encoders, $\mathcal{E}_1$ and $\mathcal{E}_2$, are defined through two tables, $\mathcal{T}$ and $\mathcal{A}$, each consisting of n_2 distinct words of length q. Only the first n_1 entries in each table will be used by $\mathcal{E}_1$. The tables are described next, followed by the definition of the encoders in Section 3.2.

For $i \in \{1,2\}$ and a word $\boldsymbol{w} \in \mathcal{L}(q, d_2, k_2, >\Delta_i, <k_2)$, let $\varphi_i(\boldsymbol{w})$ be the word obtained from $\boldsymbol{w}$ by inverting the '0' at location $\ell(\boldsymbol{w}) - \Delta_i$. The entries in $\mathcal{T} = \big(\mathcal{T}(j)\big)_{j \in \Upsilon_2}$ are distinct elements of $\mathcal{L}(q, d_2, k_2, \geq d_2, <k_2)$ such that the following two conditions are satisfied for $j \in \Upsilon_1$:

1. $\mathcal{T}(j) \in \mathcal{L}(q, d_1, k_1, \geq d_1, <k_1)$.
2. If $\mathcal{T}(j) \in \mathcal{L}(q, d_1, k_1, \geq \Delta_1 + d_2, <k_1)$ then $\mathcal{T}(j + n_1) = \varphi_1(\mathcal{T}(j))$.

Hereafter, we will use the short-hand notation $\ell(j)$ for $\ell(\mathcal{T}(j))$. We will also let $\rho(\boldsymbol{w})$ denote the word obtained from a binary word $\boldsymbol{w}$ by inverting the bit at location 0 in $\boldsymbol{w}$.

Clearly, there is much freedom left in setting the entries in $\mathcal{T}$. For example, we may fill the first n_1 entries in $\mathcal{T}$ consecutively by words from $\mathcal{L}(q, d_1, k_1, =r, <k_1)$ for descending values of r, starting with $r = k_1 - 1$ and ending with $r = d_1$. Then, we continue filling the table consecutively with elements from $\mathcal{L}(q, d_1, k_1, =r, <k_1)$ for $r = d_1 - 1, d_1 - 2, \ldots, d_2$ so that $\mathcal{T}(j + n_1) = \varphi_1(\mathcal{T}(j))$, until we reach the largest index $j \in \Upsilon_1$ that satisfies $\mathcal{T}(j) \in \mathcal{L}(q, d_1, k_1, \geq \Delta_1 + d_2, <k_1)$. The remaining entries in $\mathcal{T}$ are words from $\mathcal{L}(q, d_2, k_2, \geq d_2, <k_2)$ that haven't been inserted so far.

The entries in $\mathcal{A} = \big(\mathcal{A}(j)\big)_{j \in \Upsilon_2}$ are distinct elements of $\mathcal{L}(q, d_2, k_2, \leq 1, <k_2)$ that satisfy the following conditions:

3. $\mathcal{A}(j) \in \mathcal{L}(q, d_1, k_1, \leq 1, <k_1)$ for $j \in \Upsilon_1$.
4. If $\ell(j) = \Delta_i + 1$ for an index $j \in \Upsilon_i$ then $\mathcal{A}(j) = \varphi_i(\mathcal{T}(j))$.

For example, for every $j \in \Upsilon_2$, we can let $\mathcal{A}(j)$ be $\rho(\mathcal{T}(j))$ except when $\ell(j) \in \{d_2, \Delta_2 + 1\}$ or when $j \in \Upsilon_1$ and $\ell(j) \in \{d_1, \Delta_1 + 1\}$. For the remaining undetermined entries (i.e., when $\ell(j) = d_2$ or when $j \in \Upsilon_1$ and $\ell(j) = d_1$), the relationship between the entries $\mathcal{A}(j)$ and $\mathcal{T}(j)$ might be somewhat more involved.

Next we show (Proposition 3.1 below) that tables $\mathcal{T}$ and $\mathcal{A}$ that satisfy conditions 1–4 can indeed be constructed. We first point out that if we deleted condition 4, then the existence of $\mathcal{T}$ and $\mathcal{A}$ would follow from the Gu-Fuja construction [4]. Indeed, Lemma A5 in [4] states that

$$|\mathcal{L}(q, d, k, \geq d, <k)| \leq |\mathcal{L}(q, d, k, \leq 1, <k)|$$

whenever $1 \leq d < k$ and $q \geq d$. Recalling that n_1 and n_2 are required to satisfy

$$n_i \leq |\mathcal{L}(q, d_i, k_i, \geq d_i, <k_i)| \quad \text{for } i = 1, 2\,, \tag{3}$$

there are sufficiently many elements in $\mathcal{L}(q, d_1, k_1, \geq d_1, <k_1)$ and $\mathcal{L}(q, d_1, k_1, \leq 1, <k_1)$ which can be assigned to the first n_1 entries in $\mathcal{T}$ and $\mathcal{A}$, respectively, while satisfying conditions 1 and 3. Also, there are enough remaining elements in $\mathcal{L}(q, d_2, k_2 \geq d_2, <k_2)$ and $\mathcal{L}(q, d_2, k_2, \leq 1, <k_2)$ that can be used for the remaining entries of $\mathcal{T}$ and $\mathcal{A}$; note that to this end, condition 2 poses no impediment (recall that $n_2 \geq 2n_1$).

It is condition 4 where the Gu-Fuja result needs to be refined, since this condition introduces cases where certain elements of $\mathcal{L}(q, d_1, k_1, \leq 1, <k_1)$ are forced to be assigned to entries $\mathcal{A}(j)$ while the respective entries $\mathcal{T}(j)$ are not in $\mathcal{L}(q, d_1, k_1, \geq d_1, <k_1)$ (hence, the inequality $|\mathcal{L}(q, d_1, k_1, \geq d_1, <k_1)| \leq |\mathcal{L}(q, d_1, k_1, \leq 1, <k_1)|$ does not guarantee that we have enough available elements in $\mathcal{L}(q, d_1, k_1, \leq 1, <k_1)$ that can be assigned to the first n_1 entries in $\mathcal{A}$). By condition 4, such a situation occurs when (and only when) $\mathcal{T}(j)$ belongs to the set

$$\mathcal{X} = \left\{ \begin{array}{r} \boldsymbol{w} \in \mathcal{L}(q, d_2, k_2, =\Delta_2 + 1, <k_2) \setminus \mathcal{L}(q, d_1, k_1, \geq d_1, <k_1) \ : \\ \varphi_2(\boldsymbol{w}) \in \mathcal{L}(q, d_1, k_1, =1, <k_1) \end{array} \right\} .$$

Since the image under φ_2 of each word $\boldsymbol{w} \in \mathcal{X}$ is in $\mathcal{L}(q, d_1, k_1, =1, <k_1)$, it follows that $\boldsymbol{w}$ violates the (d_1, k_1)-RLL constraint only in its first runlength, which may be equal to either $k_1 + 1$ or $k_1 + 2$. This implies that $\mathcal{X}$ is nonempty only if $\Delta_2 \in \{k_1, k_1+1\}$ and

$$|\mathcal{X}| = \left\{ \begin{array}{ll} |\mathcal{L}(q-k_1-2, d_1, k_1, \geq d_1, <k_1)| & \text{if } \Delta_2 = k_1 \\ |\mathcal{L}(q-k_1-3, d_1, k_1, \geq d_1, <k_1)| & \text{if } \Delta_2 = k_1 + 1 \end{array} \right. .$$

Since $|\mathcal{L}(t, d_1, k_1, \geq d_1, <k_1)|$ is nondecreasing with t, we thus have

$$|\mathcal{X}| \leq |\mathcal{L}(q-k_1-2, d_1, k_1, \geq d_1, <k_1)| . \tag{4}$$

Proposition 3.1 *Suppose that*

$$q > k_2 \geq k_1 \geq 2d_1 \geq 2d_2 > 0$$

and that (3) is satisfied by n_1 and n_2 where $n_2 \geq 2n_1$. There exist tables

$$\mathcal{T} \in (\mathcal{L}(q, d_2, k_2, \geq d_2, <k_2))^{n_2} \quad \text{and} \quad \mathcal{A} \in (\mathcal{L}(q, d_2, k_2, \leq 1, <k_2))^{n_2} ,$$

each consisting of distinct entries, such that conditions 1–4 hold.

Proof. In view of the foregoing discussion, it suffices to show that

$$|\mathcal{L}(q, d_1, k_1, \geq d_1, <k_1)| \leq |\mathcal{L}(q, d_1, k_1, \leq 1, <k_1)| - |\mathcal{X}| .$$

Letting $\mu(q)$ stand for $|\mathcal{L}(q, d_1, k_1, \geq d_1, <k_1)|$, we first note that

$$\mu(q) = \sum_{i=d_1+1}^{k_1+1} \mu(q-i) \; . \tag{5}$$

Substituting $q-1$ for q, we get

$$\mu(q-1) = \sum_{i=d_1+1}^{k_1+1} \mu(q-1-i) = \sum_{i=d_1+2}^{k_1+2} \mu(q-i) \; . \tag{6}$$

We now subtract respective sides of (5) and (6) to yield

$$\mu(q) - \mu(q-1) = \mu(q-d_1-1) - \mu(q-k_1-2) \leq \mu(q-2) - \mu(q-k_1-2) \; , \tag{7}$$

with the inequality holding since $t \mapsto \mu(t)$ is nondecreasing. Noting that $\mu(q-i) = |\mathcal{L}(q, d_1, k_1, =i-1, <k_1)|$ for $i = 1, 2$, we thus obtain from (7),

$$\begin{aligned} \mu(q) &\leq \mu(q-1) + \mu(q-2) - \mu(q-k_1-2) \\ &= |\mathcal{L}(q, d_1, k_1, =0, <k_1)| + |\mathcal{L}(q, d_1, k_1, =1, <k_1)| - \mu(q-k_1-2) \\ &\leq |\mathcal{L}(q, d_1, k_1, \leq 1, <k_1)| - |\mathcal{X}| \; , \end{aligned}$$

where the second equality follows from (4). This leads to the desired result. □

Following arguments similar to those in [4], it can be shown that through a proper ordering of the entries in $\mathcal{T}$ and $\mathcal{A}$, enumerative coding can be applied to compute efficiently the values $\mathcal{T}(j)$ and $\mathcal{A}(j)$ for any given index j [6, p. 117]. In practice, the tables are generated only once and then hard-wired into the encoders.

3.2 Encoder graphs

For $i = 1, 2$, the tagged encoder $\mathcal{E}_i = (V_i, E_i, L_i)$ consists of a set of k_i states,

$$V_i = \{0, 1, \ldots, k_i-1\} \; .$$

We regard the input tags as integers $j \in \Upsilon_i$ and define the labeling $L_i : V_i \times \Upsilon_i \to \mathcal{L}(q, d_i, k_i, \geq 0, <k_i)$ for every $u \in V_i$ and $j \in \Upsilon_i$ as follows:

$$L_i(u, j) = \begin{cases} \mathcal{T}(j) & \text{if } k_i - \ell(j) \geq u \;\text{ and }\; u < d_1 \\ \varphi_i(\mathcal{T}(j)) & \text{if } k_i - \ell(j) < u < d_i \\ \mathcal{A}(j) & \text{if } \left\{k_i - \ell(j) < u \text{ and } u \geq d_i\right\} \;\text{ or }\; u \geq d_1 \end{cases} \; .$$

The terminal state of an edge labeled $\boldsymbol{w}$ is v, where v equals the last runlength in $\boldsymbol{w}$. Note that $L_i(u,j)$ is well-defined when $k_i - \ell(j) < u < d_i$: in this case we always have $\ell(j) > \Delta_i$ and, so, $\mathcal{T}(j)$ belongs to the domain of φ_i.

One can readily verify that each sequence of codewords that is generated by $\mathcal{E}_i$ forms a binary word that belongs to $\mathcal{S}_{(d_i,k_i)}$.

Lemma 3.2 *For $i = 1, 2$, the set $\Sigma(\mathcal{E}_i)$ can be partitioned into $\Sigma(\mathcal{E}_i) = T_i \cup \Phi_i \cup A_i$, where*

$$T_i = \left\{\mathcal{T}(j)\right\}_{j\in\Upsilon_i}, \quad \Phi_i = \left\{\varphi_i(\mathcal{T}(j))\right\}_{j\in\Upsilon_i\,:\,\ell(j)>\Delta_i+1}, \quad \text{and} \quad A_i = \left\{\mathcal{A}(j)\right\}_{j\in\Upsilon_i}.$$

In particular, for every $\boldsymbol{w} \in \Sigma(\mathcal{E}_i)$,

$$\boldsymbol{w} \in \begin{cases} T_i & \text{if } \ell(\boldsymbol{w}) \geq d_i \\ \Phi_i & \text{if } 1 < \ell(\boldsymbol{w}) < d_i \\ A_i & \text{if } \ell(\boldsymbol{w}) \leq 1 \end{cases}.$$

Proof. By the definition of the labeling $L_i : V_i \times \Upsilon_i \to \mathcal{L}(q, d_i, k_i, \geq 0, <k_i)$ it follows that $\Sigma(\mathcal{E}_i) \subseteq T_i \cup \Phi_i' \cup A_i$, where

$$\Phi_i' = \left\{\varphi_i(\mathcal{T}(j))\right\}_{j\in\Upsilon_i\,:\,\ell(j)>\Delta_i}.$$

Furthermore, by the way $\mathcal{A}$ is constructed (condition 4) we have

$$\Phi_i' \setminus \Phi_i = \left\{\varphi_i(\mathcal{T}(j))\right\}_{j\in\Upsilon_i\,:\,\ell(j)=\Delta_i+1} \subseteq A_i.$$

Therefore, $\Sigma(\mathcal{E}_i) \subseteq T_i \cup \Phi_i \cup A_i$.

Conversely, we have $L_i(0,j) = \mathcal{T}(j)$ and $L_i(d_1,j) = \mathcal{A}(j)$ for every $j \in \Upsilon_i$, and $L_i(k_i - \ell(j) + 1, j) = \varphi_i(\mathcal{T}(j))$ when $\ell(j) > \Delta_i$. Therefore, $T_i \cup \Phi_i' \cup A_i \subseteq \Sigma(\mathcal{E}_i)$.

The sets T_i, Φ_i, and A_i are disjoint, thereby forming a partition: from the definition of $\mathcal{T}$, φ_i, and $\mathcal{A}$, the first runlength of any given word $\boldsymbol{w} \in \Sigma(\mathcal{E}_i)$ determines the partition element to which $\boldsymbol{w}$ belongs. □

When $u > v \geq d_1$, states u and v in $\mathcal{E}_i$ are in fact identical: in these states, the outgoing edges that are tagged by j have the same label, $\mathcal{A}(j)$, and therefore also the same terminal state (which is determined by the last runlength of $\mathcal{A}(j)$). Therefore, we can *merge* the states $d_1, d_1+1, \ldots, k_i-1$ by deleting them and re-directing their incoming edges into one new state, $[d_1, k_i-1]$, whose outgoing edges—with their tagging, labeling,

and terminal states—are the same as those in $u = d_1$. This turns each encoder $\mathcal{E}_i$ into an encoder with only $d_1 + 1$ states. (Furthermore, if we ignored observability and were interested only in constructing a block decodable encoder for $\mathcal{S}_{(d_2,k_2)}$, then $\mathcal{E}_2$ could be simplified to have at most $d_2 + 1$ states.) Yet, for clarity, we will maintain hereafter the definition of $\mathcal{E}_i$ in its unmerged form, namely, having k_i states.

(In the typical case where n_i is strictly smaller than $|\mathcal{L}(q, d_i, k_i, \geq d_i, <k_i)|$ we have more flexibility in selecting the codewords when constructing the encoders. For example, the codewords can sometimes be restricted to be from a set $\mathcal{L}(q, d_i, k_i, \geq 0, <r)$ where r is now smaller than k_i. In fact, this is the case in the Beenker-Immink construction [2], where we have $r = \Delta_i$, and this will be done in the appendix, where $r = 9$ while $k_i = 10$. The additional flexibility in selecting the codewords may allow more merging opportunities, thereby reducing the number of resulting states. The reader is referred to [8, Section 4.6.1] for more general settings in which merging can be applied.)

3.3 Decoding

For $i = 1, 2$ let

$$\mathcal{D}_i : \Sigma(\mathcal{E}_i) \to \Upsilon_i$$

be defined as follows. For a codeword $\boldsymbol{w} \in \Sigma(\mathcal{E}_i)$, let $\boldsymbol{w}'$ be obtained from $\boldsymbol{w}$ by inverting the (first) '1' at location $\ell(\boldsymbol{w})$. Then,

$$\mathcal{D}_i(\boldsymbol{w}) = \text{index } j \text{ for which } \begin{cases} \mathcal{T}(j) = \boldsymbol{w} & \text{if } \ell(\boldsymbol{w}) \geq d_i \\ \mathcal{T}(j) = \boldsymbol{w}' & \text{if } 1 < \ell(\boldsymbol{w}) < d_i \\ \mathcal{A}(j) = \boldsymbol{w} & \text{if } \ell(\boldsymbol{w}) \leq 1 \end{cases} . \tag{8}$$

Lemma 3.3 *$\mathcal{D}_i$ is a block decoder of $\mathcal{E}_i$.*

Proof. This follows directly from the partition in Lemma 3.2. □

Let

$$\mathcal{D} : \Sigma(\mathcal{E}_1) \cup \Sigma(\mathcal{E}_2) \to \Upsilon_2$$

be the following extension of $\mathcal{D}_2$ to the domain $\Sigma(\mathcal{E}_1) \cup \Sigma(\mathcal{E}_2)$:

$$\mathcal{D}(\boldsymbol{w}) = \text{index } j \text{ for which } \begin{cases} \mathcal{T}(j) = \boldsymbol{w} & \text{if } \ell(\boldsymbol{w}) \geq d_2 \\ \mathcal{T}(j) = \boldsymbol{w}' & \text{if } 1 < \ell(\boldsymbol{w}) < d_2 \\ \mathcal{A}(j) = \boldsymbol{w} & \text{if } \ell(\boldsymbol{w}) \leq 1 \end{cases} . \tag{9}$$

Obviously, $\mathcal{D}_2(\boldsymbol{w}) = \mathcal{D}(\boldsymbol{w})$ for every $\boldsymbol{w} \in \Sigma(\mathcal{E}_2)$. Note that the domains of $\mathcal{D}$ and $\mathcal{D}_2$ might differ, since $\Sigma(\mathcal{E}_1)$ is not necessarily a subset of $\Sigma(\mathcal{E}_2)$. This means that $(\mathcal{E}_1, \mathcal{E}_2)$ is not necessarily a weakly-observable pair.

(Indeed, suppose that $d_2 > 2$ and let $j \in \Upsilon_1$ be such that $\Delta_1 + 1 < \ell(\mathcal{T}(j)) < \Delta_1 + d_2$. Consider the word $\boldsymbol{w} = \varphi_1(\mathcal{T}(j))$. On the one hand, Lemma 3.2 implies that $\boldsymbol{w} \in \Sigma(\mathcal{E}_1)$. On the other hand, since $1 < \ell(\boldsymbol{w}) < d_2$, the word $\boldsymbol{w}$ is neither an entry in $\mathcal{A}$ nor in $\mathcal{T}$; furthermore, the second runlength in $\boldsymbol{w}$ is $\Delta_1 - 1$ and, as such, $\boldsymbol{w}$ cannot be in the range of φ_2. Hence, it follows from Lemma 3.2 that $\boldsymbol{w} \notin \Sigma(\mathcal{E}_2)$.)

Yet, the next lemma will show that we are nearly done in obtaining a weakly-observable pair.

Let $\psi_{\text{mod}} : \Upsilon_2 \to \Upsilon_1$ map each integer in Υ_2 to its remainder in Υ_1 when divided by n_1. That is, recalling that $n_i = 2^{p_i}$, the function ψ_{mod} chops off the $p_2 - p_1$ most-significant bits of the binary representation of its argument.

Lemma 3.4 *$\mathcal{D}_1(\boldsymbol{w}) = \psi_{\text{mod}}(\mathcal{D}(\boldsymbol{w}))$ for every $\boldsymbol{w} \in \Sigma(\mathcal{E}_1)$.*

Proof. Let $\boldsymbol{w}$ be a word in $\Sigma(\mathcal{E}_1)$. Clearly, $\mathcal{D}(\boldsymbol{w}) = \mathcal{D}_1(\boldsymbol{w})$ whenever $\ell(\boldsymbol{w}) \geq d_1$ or $\ell(\boldsymbol{w}) < d_2$. Assume now that $1 < d_2 \leq \ell(\boldsymbol{w}) < d_1$ and let $j \in \Upsilon_1$ be the value of $\mathcal{D}_1(\boldsymbol{w})$. In this case we have $\boldsymbol{w} = \varphi_1(\mathcal{T}(j))$ where $\mathcal{T}(j) \in \mathcal{L}(q, d_1, k_1, \geq\Delta_1 + d_2, <k_1)$. By the way $\mathcal{T}$ is constructed it follows that $\mathcal{T}(j + n_1) = \varphi_1(\mathcal{T}(j)) = \boldsymbol{w}$ and, so, $j = \mathcal{D}(\boldsymbol{w}) - n_1 = \psi_{\text{mod}}(\mathcal{D}(\boldsymbol{w}))$. □

We may interpret Lemma 3.4 as stating that $\mathcal{E}_1$ is 'almost observable' from $\mathcal{E}_2$: indeed, $\mathcal{D}$, when restricted to the domain $\Sigma(\mathcal{E}_2)$, is a decoder of $\mathcal{E}_2$.

3.4 Incorporating full observability

We next modify the encoder $\mathcal{E}_2$ to obtain (full) observability. Specifically, we construct a tagged encoder $\mathcal{E}_2' = (V_2', E_2', L_2')$ with $V_2' = V_2$, where each state has n_2 outgoing edges that are tagged by Υ_2. The labeling $L_2' : V_2 \times \Upsilon_2 \to \mathcal{L}(q, d_2, k_2, \geq 0, <k_2)$ is defined for every $(u, j) \in V_2 \times \Upsilon_2$ as follows:

$$L_2'(u,j) = \begin{cases} \varphi_1(\mathcal{T}(j)) & \text{if } j \in \Upsilon_1 \text{ and } d_1 - d_2 \leq k_1 - \ell(j) < u < d_1 \\ L_2(u,j) & \text{otherwise} \end{cases} .$$

The terminal state of an edge labeled $\boldsymbol{w}$ in $\mathcal{E}_2'$ is a state v, where v equals the last runlength in $\boldsymbol{w}$.

We first need to establish that $\mathcal{E}_2'$ is an encoder for $\mathcal{S}_{(d_2,k_2)}$. By construction, each state in $\mathcal{E}_2'$ has n_2 outgoing edges, and it is easy to see that $\mathcal{E}_2'$ generates only sequences in $\mathcal{S}_{(d_2,k_2)}$. The next lemma implies the losslessness of $\mathcal{E}_2'$.

Lemma 3.5 *(The untagged version of) $\mathcal{E}_2$ is deterministic.*

Proof. Suppose to the contrary that there is a state $u \in V_1$ and two input tags, $j \in \Upsilon_1$ and $j' \in \Upsilon_2 \setminus \Upsilon_1$, such that $d_1 - d_2 \leq k_1 - \ell(j) < u < d_1$ and

$$\varphi_1(\mathcal{T}(j)) = L_2'(u,j) = L_2'(u,j') = L_2(u,j') \ .$$

Using the notations of Lemma 3.2, this implies that $\varphi_1(\mathcal{T}(j)) \in \Phi_1 \cap (T_2 \cup \Phi_2 \cup A_2)$. Since the first runlength of each word in Φ_1 is greater than 1 and the second runlength equals $\Delta_1 - 1$, the set Φ_1 intersects with neither A_2 nor Φ_2. It follows that $\varphi_1(\mathcal{T}(j)) \in \Phi_1 \cap T_2$, which occurs only when $j' = j + n_1$ and $\varphi_1(\mathcal{T}(j)) = \mathcal{T}(j')$. This, in turn, implies

$$\ell(j) = \ell(\varphi_1(\mathcal{T}(j))) + \Delta_1 = \ell(j') + \Delta_1 \geq d_2 + \Delta_1 = k_1 - d_1 + d_2 + 1 \ .$$

Hence, $k_1 - \ell(j) < d_1 - d_2$, thus contradicting our assumption on j. □

Lemma 3.6 *(The untagged version of) $\mathcal{E}_1$ is nested in $\mathcal{E}_2'$.*

Proof. Given $(u,j) \in V_1 \times \Upsilon_1$, we show that there exists $j' \in \Upsilon_2$ such that $L_2'(u,j') = L_1(u,j)$. To this end, we distinguish between several cases.

Case 1: $k_1 - \ell(j) \geq u$ and $u < d_1$. In this case we have $L_2'(u,j) = L_2(u,j) = \mathcal{T}(j) = L_1(u,j)$.

Case 2: $d_1 - d_2 \leq k_1 - \ell(j) < u < d_1$. Here we have $L_2'(u,j) = \varphi_1(\mathcal{T}(j)) = L_1(u,j)$.

Case 3: $k_1 - \ell(j) < d_1 - d_2$ and $k_1 - \ell(j) < u < d_1$. In this range,

$$\ell(j) \geq k_1 - (d_1 - d_2) + 1 = \Delta_1 + d_2 \ ,$$

which, in turn, implies that $\mathcal{T}(j + n_1) = \varphi_1(\mathcal{T}(j))$ and that

$$k_2 - \ell(j + n_1) = k_2 - \ell(j) + \Delta_1 \geq \Delta_1 > d_1 > u \ .$$

Hence,

$$L_2'(u, j + n_1) = L_2(u, j + n_1) = \mathcal{T}(j + n_1) = \varphi_1(\mathcal{T}(j)) = L_1(u,j) \ ,$$

where in the last equality we make use of the condition $k_1 - \ell(j) < u < d_1$.

Case 4: $u \geq d_1$. Here we have, $L'_2(u,j) = L_2(u,j) = \mathcal{A}(j) = L_1(u,j)$.

It follows from all cases that every codeword that can be generated from state u in $\mathcal{E}_1$ can also be generated from state u in $\mathcal{E}'_2$, while terminating in the same state. This implies that $\mathcal{E}_1$ is nested in $\mathcal{E}'_2$. □

We point out that even though $\mathcal{E}_1$ is nested in $\mathcal{E}'_2$, the assignment of input tags to codewords may differ in the two encoders. In fact, from the proof of Lemma 3.6 we see that such a difference occurs in (and only in) case 3, where we have $L_1(u,j) = L'_2(u, j+n_1)$; that is, edges labeled by $L_1(u,j)$ are assigned the input tag j in $\mathcal{E}_1$ and the input tag $j' = j + n_1$ in $\mathcal{E}'_2$. It follows that if $\mathcal{D}'_2 : \Sigma(\mathcal{E}'_2) \rightarrow \Upsilon_2$ is a block decoder of $\mathcal{E}'_2$ then the block decoder of $\mathcal{E}_1$ satisfies $\mathcal{D}_1(\boldsymbol{w}) = \psi_{\text{mod}}(\mathcal{D}'_2(\boldsymbol{w}))$ for every $\boldsymbol{w} \in \Sigma(\mathcal{E}_1)$. One can readily check that a block decoder of $\mathcal{E}'_2$ is obtained by restricting the mapping $\mathcal{D}$ in (9) to the domain $\Sigma(\mathcal{E}'_2)$. Hence, we reach the following conclusion.

Proposition 3.7 *$(\mathcal{E}_1, \mathcal{E}'_2)$ is a nested and observable pair.*

4 The case $d_2 = 1$

Our construction for the case $d_2 = 1$ follows the framework presented in Section 3, with a modification in the way the tables are constructed. The case $d_1 = 1$ is treated in Section 4.1, while the construction for larger values of d_1 is presented in Section 4.2.

4.1 $d_1 = d_2 = 1$

In this case we have $\Delta_i = k_i$; so, conditions 2 and 4 on $\mathcal{T}$ and $\mathcal{A}$ in Section 3.1 become vacuous. On the other hand, both $\mathcal{T}$ and $\mathcal{A}$ may now contain entries whose first runlength is 1; so, we will require that the two tables coincide on such entries. Specifically, we require that each of the tables $\mathcal{T}$ and $\mathcal{A}$ consist of distinct entries such that the following conditions hold for $i = 1, 2$ and every $j \in \Upsilon_i$:

1. $\mathcal{T}(j) \in \mathcal{L}(q, 1, k_i, \geq 1, < k_i)$.
2. If $\ell(j) = 1$ then $\mathcal{A}(j) = \mathcal{T}(j)$.
3. If $\ell(j) > 1$ then $\mathcal{A}(j) \in \mathcal{L}(q, 1, k_i, = 0, < k_i)$.

For example, we can let $\mathcal{A}(j) = \rho(\mathcal{T}(j))$ whenever $\ell(j) > 1$.

Given such tables, the encoders are defined as in Section 3.2. For $d_i = 1$, the labeling $L_i : V_i \times \Upsilon_i \to \Sigma(\mathcal{E}_i)$ reduces to

$$L_i(u, j) = \begin{cases} \mathcal{T}(j) & \text{if } u = 0 \\ \mathcal{A}(j) & \text{if } u > 0 \end{cases} .$$

It follows that encoder $\mathcal{E}_1$ is nested in $\mathcal{E}_2$; furthermore, edges in the two encoders that are labeled by the same codewords have the same input tags. Note that the states of each encoder $\mathcal{E}_i$ can be merged into two states, 0 and $[1, k_i - 1]$.

A block decoder $\mathcal{D}_2 : \Sigma(\mathcal{E}_2) \to \Upsilon_2$ of $\mathcal{E}_2$ is given by

$$\mathcal{D}_2(\boldsymbol{w}) = \text{index } j \text{ for which } \begin{cases} \mathcal{T}(j) = \boldsymbol{w} & \text{if } \ell(\boldsymbol{w}) \geq 1 \\ \mathcal{A}(j) = \boldsymbol{w} & \text{if } \ell(\boldsymbol{w}) = 0 \end{cases} , \tag{10}$$

and a block decoder of $\mathcal{E}_1$ is obtained by restricting the domain of $\mathcal{D}_2$ to $\Sigma(\mathcal{E}_1)$.

4.2 $d_1 > 1$ and $d_2 = 1$

In this case, the table $\mathcal{A}$ is not shared by the two encoders. Specifically, we define three tables, $\mathcal{T} = \big(\mathcal{T}(j)\big)_{j \in \Upsilon_2}$, $\mathcal{A} = \big(\mathcal{A}(j)\big)_{j \in \Upsilon_2}$, and $\mathcal{A}_1 = \big(\mathcal{A}_1(j)\big)_{j \in \Upsilon_1}$: while $\mathcal{T}$ and $\mathcal{A}$ are still accessed by $\mathcal{E}_2$ and the first n_1 entries in $\mathcal{T}$ are accessed by $\mathcal{E}_1$, the latter encoder will now access $\mathcal{A}_1$ instead of the first n_1 entries in $\mathcal{A}$.

The entries in $\mathcal{T}$ are distinct elements of $\mathcal{L}(q, 1, k_2, \geq 1, < k_2)$, and the entries in $\mathcal{A}_1$ are distinct elements of $\mathcal{L}(q, d_1, k_1, \leq 1, < k_1)$ such that the following conditions hold for every $j \in \Upsilon_1$:

1'. $\mathcal{T}(j) \in \mathcal{L}(q, d_1, k_1, \geq d_1, < k_1)$.

2'. If $\mathcal{T}(j) \in \mathcal{L}(q, d_1, k_1, \geq \Delta_1 + 1, < k_1)$ then $\mathcal{T}(j + n_1) = \varphi_1(\mathcal{T}(j))$.

3'. If $\ell(j) \notin \{d_1, \Delta_1 + 1\}$ then $\mathcal{A}_1(j) = \rho(\mathcal{T}(j))$.

4'. If $\ell(j) = \Delta_1 + 1$ then $\mathcal{A}_1(j) = \varphi_1(\mathcal{T}(j))$.

5'. If $\mathcal{A}_1(j) \in \mathcal{L}(q, d_1, k_1, = 1, < k_1)$ then $\mathcal{T}(j + n_1) = \mathcal{A}_1(j)$.

Conditions 1' through 4' have their counterparts in Section 3.1 (with $\mathcal{A}_1$ now replacing $\mathcal{A}$), except that condition 3' is now stronger. Note that since $\ell(j)$ can never take the value $\Delta_2 + 1 = k_2 + 1$, condition 4' is simpler than condition 4 in Section 3.1.

Condition 5' is new and may affect the value of $\mathcal{T}(j+n_1)$ only when $\ell(j) \in \{d_1, \Delta_1+1\}$.

The inequality $n_2 \leq |\mathcal{L}(q, 1, k_2, \geq 1, <k_2)|$ guarantees that there are sufficiently many distinct elements in $\mathcal{L}(q, 1, k_2, \geq 1, <k_2)$ which can be inserted in $\mathcal{T}$, and the Gu-Fuja construction allows to fill in the tables $\mathcal{T}$ and $\mathcal{A}_1$ so that conditions 1'–5' hold.

The table $\mathcal{A}$ consists of distinct elements of $\mathcal{L}(q, 1, k_2, \leq 1, <k_2)$ that satisfy the following conditions for every $j \in \Upsilon_2$:

6'. If $\mathcal{A}_1(j) \in \mathcal{L}(q, d_1, k_1, =0, <k_1)$ for $j \in \Upsilon_1$ then $\mathcal{A}(j) = \mathcal{A}_1(j)$.

7'. If $\ell(j) = 1$ then $\mathcal{A}(j) = \mathcal{T}(j)$.

8'. If $\ell(j) > 1$ then $\mathcal{A}(j) \in \mathcal{L}(q, 1, k_2, =0, <k_2)$.

For example, when $\ell(j) > 1$, we can satisfy condition 8' by letting $\mathcal{A}(j) = \rho(\mathcal{T}(j))$, unless $\mathcal{A}(j') = \mathcal{A}_1(j') = \rho(\mathcal{T}(j))$ for some $j' \in \Upsilon_1 \setminus \{j\}$. By conditions 3' and 4', the excluded case can occur only when $\ell(j') = d_1$, where instead we can let $\mathcal{A}(j)$ be $\rho(\mathcal{T}(j'))$. Note that when $j \in \Upsilon_1$ and $\ell(j) \notin \{d_1, \Delta_1 + 1\}$, conditions 3', 6', and 8' become

$$\mathcal{A}_1(j) = \mathcal{A}(j) = \rho(\mathcal{T}(j)) \,,$$

and when $j \in \Upsilon_1$ and $\ell(j) = \Delta_1 + 1$, conditions 2', 4', 5' and 7' become

$$\mathcal{A}_1(j) = \mathcal{A}(j + n_1) = \mathcal{T}(j + n_1) = \varphi_1(\mathcal{T}(j)) \,.$$

Irrespective of the value of $\ell(j)$, we have

$$\mathcal{A}_1(j) \in \big\{\mathcal{A}(j), \mathcal{A}(j + n_1)\big\} \quad \text{for every } j \in \Upsilon_1 \,. \tag{11}$$

The encoders $\mathcal{E}_i = (V_i, E_i, L_i)$ are now defined as in Section 3.2, except that $\mathcal{E}_1$ now accesses $\mathcal{A}_1$ instead of $\mathcal{A}$. That is, the labeling $L_1 : V_1 \times \Upsilon_1 \to \Sigma(\mathcal{E}_1)$ is given by

$$L_1(u, j) = \begin{cases} \mathcal{T}(j) & \text{if } k_1 - \ell(j) \geq u \quad \text{and} \quad u < d_1 \\ \varphi_1(\mathcal{T}(j)) & \text{if } k_1 - \ell(j) < u < d_1 \\ \mathcal{A}_1(j) & \text{if } u \geq d_1 \end{cases} \,,$$

and $L_2 : V_2 \times \Upsilon_1 \to \Sigma(\mathcal{E}_2)$ is given by

$$L_2(u, j) = \begin{cases} \mathcal{T}(j) & \text{if } k_2 - \ell(j) \geq u \quad \text{and} \quad u < d_1 \\ \mathcal{A}(j) & \text{if } k_2 - \ell(j) < u \quad \text{or} \quad u \geq d_1 \end{cases} \,.$$

A block decoder $\mathcal{D}_1$ of $\mathcal{E}_1$ is obtained by substituting $\mathcal{A}_1$ for $\mathcal{A}$ in (8), and a block decoder $\mathcal{D}_2$ of $\mathcal{E}_2$ is given by (10).

Lemma 4.1 *$(\mathcal{E}_1, \mathcal{E}_2)$ is a nested pair.*

Proof. As was the case in the proof of Lemma 3.6, we show that for every $(u, j) \in V_1 \times \Upsilon_1$ there is $j' \in \Upsilon_2$ such that $L_2(u, j') = L_1(u, j)$.

Case 1: $k_1 - \ell(j) \geq u$ and $u < d_1$. In this case we have $L_1(u, j) = L_2(u, j) = \mathcal{T}(j)$.

Case 2: $k_1 - \ell(j) < u < d_1$. Here we have

$$L_1(u, j) = \varphi_1(\mathcal{T}(j)) = \mathcal{T}(j + n_1) = L_2(u, j + n_1) \ ,$$

where the last equality follows from

$$k_2 - \ell(j + n_1) = k_2 - \ell(j) + \Delta_1 \geq \Delta_1 > d_1 > u \ .$$

Case 3: $u \geq d_1$. By (11) we have,

$$L_1(u, j) = \mathcal{A}_1(j) \in \Big\{\mathcal{A}(j), \mathcal{A}(j + n_1)\Big\} \ .$$

Hence, $L_1(u, j) \in \Big\{L_2(u, j), L_2(u, j + n_1)\Big\}$. □

Lemma 4.2 *$\mathcal{D}_1(\boldsymbol{w}) = \psi_{\text{mod}}(\mathcal{D}_2(\boldsymbol{w}))$ for every $\boldsymbol{w} \in \Sigma(\mathcal{E}_1)$.*

Proof. Let $j = \mathcal{D}_1(\boldsymbol{w})$ for a word $\boldsymbol{w} \in \Sigma(\mathcal{E}_1)$. Then,

$$\boldsymbol{w} = \begin{cases} \mathcal{T}(j) & & & \text{if } \ell(\boldsymbol{w}) \geq d_1 \\ \varphi_1(\mathcal{T}(j)) & = & \mathcal{T}(j + n_1) & \text{if } 1 < \ell(\boldsymbol{w}) < d_1 \\ \mathcal{A}_1(j) & = & \mathcal{T}(j + n_1) & \text{if } \ell(\boldsymbol{w}) = 1 \\ \mathcal{A}_1(j) & = & \mathcal{A}(j) & \text{if } \ell(\boldsymbol{w}) = 0 \end{cases} .$$

Therefore, $\mathcal{D}_2(\boldsymbol{w}) \in \{j, j + n_1\}$. □

The following result combines Lemmas 4.1 and 4.2.

Proposition 4.3 *$(\mathcal{E}_1, \mathcal{E}_2)$ is a nested and observable pair.*

Appendix: $(3, 10)$-RLL and $(2, 10)$-RLL encoders

We describe here a block decodable $(3, 10)$-RLL encoder $\mathcal{E}_{(3,10)}$ at rate $6 : 16$ which is weakly-observable from a block decodable $(2, 10)$-RLL encoder $\mathcal{E}_{(2,10)}$ at rate $8 : 16$ (the respective capacities of the constraints are approximately 0.4460 and 0.5418). Depending on the encoder state, certain input tags can map to two codewords which differ in their parity (odd/even) number of 1's. This freedom allows to control the DC level of the recorded signal [6]. Such a provision is present also in the compact disk and DVD coding schemes [7] (yet, the encoding rate in the compact disk is $8 : 17$, and the encoder in the DVD standard is not block decodable).

The encoders herein have been obtained by combining the method presented in this paper with the one in [9] (in fact, the encoder $\mathcal{E}_{(2,10)}$ is very similar to one of the encoders in [9]). This, in turn, has required some deviations from the model presented in Section 3.

The main building block of the two encoders is one encoding table, which consists of 547 distinct codewords, each of length 16 bits. The first 256 entries in the table form the table $\mathcal{T}$ of Section 3.1, and the remaining 291 entries form essentially the table $\mathcal{A}$; the larger number of entries in $\mathcal{A}$ results from having more than one codeword mapped to certain input tags. The encoding table, which will be denoted by $\mathcal{T}||\mathcal{A}$, is shown in Table 4, and Table 1 shows a partition of the address range of $\mathcal{T}||\mathcal{A}$ according to the runlength properties of its entries. While $\mathcal{E}_{(2,10)}$ accesses the whole table, the encoder $\mathcal{E}_{(3,10)}$ accesses the entries whose addresses have the form $3 + 4t$ for $t = 0, 1, \ldots, 116$ (the boldface entries in Table 4). Note that this deviates from our convention in Section 4, according to which $\mathcal{E}_{(3,10)}$ would access the first 64 entries in $\mathcal{T}$ and $\mathcal{A}$; this modification, however, allows to make use of the simple encoding scheme as presented in [9].

	Address range	Contents of entries taken from	
		address $\equiv 3 \pmod 4$	address $\not\equiv 3 \pmod 4$
$\mathcal{T}$	$[000, 010)$	$\mathcal{L}(16, 3, 10, =9, \leq 8)$	$\mathcal{L}(16, 2, 10, \{2, 10\}, \leq 8)$
	$[010, 054)$	$\mathcal{L}(16, 3, 10, \{6, 7, 8\}, \leq 8)$	$\mathcal{L}(16, 2, 10, \{6, 7, 8, 9\}, \leq 8)$
	$[054, 177)$	$\mathcal{L}(16, 3, 10, \{3, 4, 5\}, \leq 8)$	$\mathcal{L}(16, 2, 10, \{3, 4, 5\}, \leq 8)$
	$[177, 256)$	$\mathcal{L}(16, 3, 10, =2, \leq 8)$	$\mathcal{L}(16, 2, 10, =2, \leq 8)$
$\mathcal{A}$	$[256, 377)$	$\mathcal{L}(16, 3, 10, =1, \leq 8)$	$\mathcal{L}(16, 2, 10, =1, \leq 8)$
	$[377, 468)$	$\mathcal{L}(16, 3, 10, =0, \leq 8)$	$\mathcal{L}(16, 2, 10, =0, \leq 8)$
	$[468, 547)$	$\mathcal{L}(16, 2, 10, =0, \leq 8)$	

Table 1: Skeleton of encoding table.

None of the three elements in $\mathcal{L}(16, 3, 10, =10, \leq 9)$ can be generated by $\mathcal{E}_{(3,10)}$, thereby making condition 2 in Section 3.1 vacuous. Furthermore, although the construction in

Section 3.1 allows to include codewords whose last runlength is 9, the six elements in $\mathcal{L}(16, 2, 10, \geq 0, =9)$ have been excluded from $\mathcal{T}||\mathcal{A}$, thus resulting in fewer encoder states (see below). In addition, when $\ell(j) = \Delta_i + 1$ for an index $j \in \Upsilon_i$, we have allowed $\mathcal{A}(j)$ to take any value from $\mathcal{L}(16, d_i, k_i, =1, \leq 8)$ and re-defined $\varphi_i(\mathcal{T}(j))$ to be equal to $\mathcal{A}(j)$. This has made condition 4 vacuous and introduced more flexibility in setting up the table entries so as to accommodate the technique in [9].

Input tags are 8-bit bytes in $\{0,1\}^8$, where in the case of $\mathcal{E}_{(3,10)}$ the two least-significant bits of the bytes are fixed to be '11'. (Therefore, to bridge the difference between the order of entries in the tables here and that in Section 3.1, we can associate each input tag $\boldsymbol{s} = s_0 s_1 \ldots s_7 \in \{0,1\}^8$ with an index $j = j(\boldsymbol{s}) \in \Upsilon_2 = \{0, 1, \ldots, 255\}$, where, say, $j(\boldsymbol{s}) = 255 - \sum_{i=0}^{7} s_i 2^{7-i}$.)

The encoder $\mathcal{E}_{(3,10)}$ has four states, 1, 2, [3, 5], and [6, 8] (state notation follows the one introduced in Section 3.2, namely, $[r, r']$ is the terminal state of all edges labeled by codewords whose last runlength lies between r and r'). Again, this deviates from Section 3.2, according to which we would expect $\mathcal{E}_{(3,10)}$ to have—after merging—the set of states $\{0, 1, 2, [3, 9]\}$. As explained in [9], we can gain DC control by *not* applying the merging to its full extent; therefore, instead of having a state [3, 9], we have ended up with more states, namely, [3, 5], [6, 8], and 9. On the other hand, it turns out that at a rate 6 : 16, we can spare all codewords whose last runlength is either 0 or 9, thereby deleting states 0 and 9.

The encoder $\mathcal{E}_{(2,10)}$ has also four states, 0, 1, [2, 5], and [6, 8] (note again the difference from Section 3.2 and that state 9 has been deleted also from $\mathcal{E}_{(2,10)}$, since there are no entries in $\mathcal{T}||\mathcal{A}$ whose last runlength is 9). Certain elements in $\mathcal{L}(16, 2, 10, =2, \leq 8)$ have been placed among the first ten entries in $\mathcal{T}||\mathcal{A}$ so that they are inaccessible from state 1; this prevents the 28-bit pattern '0001000100010001000100010001' from appearing anywhere in the coded bit stream, thus making such a pattern suitable for synchronization.

Encoding is carried out as follows: given an input byte $\boldsymbol{s}$, a ten-bit address is formed by prefixing $\boldsymbol{s}$ with two bits. This two-bit prefix depends on how the value, $|\boldsymbol{s}|$, of $\boldsymbol{s}$ as an integer compares with two thresholds, T_1 and T_2. These thresholds, in turn, depend on the current state of the encoder. The thresholds and prefixes of each encoder are summarized in Tables 2 and 3. The second column in those tables shows the address range of the entries in $\mathcal{T}||\mathcal{A}$ that can be accessed from any given state.

There are cases where more than one prefix is possible, resulting in two different codeword candidates which have different parity of number of 1's; such codeword candidates are located in $\mathcal{T}||\mathcal{A}$ at addresses that are 256 apart. Furthermore, both codeword candidates label edges that terminate in the same state and, therefore, replacement of a codeword with its alternate can be done locally within a generated sequence of codewords

State	Address range	Thresholds (decimal)		Prefixes (binary)		
		T_1	T_2	$0 \le \lvert s \rvert < T_1$	$T_1 \le \lvert s \rvert < T_2$	$T_2 \le \lvert s \rvert < 256$
1	[000, 256)	000	000	—	—	00
2	[010, 377)	010	121	01	01 or 00	00
[3, 5]	[054, 468)	054	212	01	01 or 00	00
[6, 8]	[177, 468)	177	212	01	01 or 00	00

Table 2: Thresholds and prefixes for $\mathcal{E}_{(3,10)}$. The two least-significant bits of s are fixed to be '11'.

State	Address range	Thresholds (decimal)		Prefixes (binary)		
		T_1	T_2	$0 \le \lvert s \rvert < T_1$	$T_1 \le \lvert s \rvert < T_2$	$T_2 \le \lvert s \rvert < 256$
0	[000, 256)	000	000	—	—	00
1	[010, 377)	010	121	01	01 or 00	00
[2, 5]	[054, 547)	035	054	10 or 01	01	01 or 00
[6, 8]	[177, 547)	035	177	10 or 01	01	01 or 00

Table 3: Thresholds and prefixes for $\mathcal{E}_{(2,10)}$.

without affecting preceding or following codewords. This simple encoding mechanism follows from the fact that codewords generated from any given state are located in a *contiguous* segment of $\mathcal{T}||\mathcal{A}$. This applies also to $\mathcal{E}_{(3,10)}$ if we regard only entries that are located at addresses of the form $3 + 4t$.

In order to obtain DC control, we need to be able to generate more than 64 codewords from certain states in $\mathcal{E}_{(3,10)}$, and more than 256 codewords in $\mathcal{E}_{(2,10)}$. Consider for example the codewords that can be generated from states $u \ge d_1$. While in Section 3.2 we have restricted the generated codeword to be taken only from $\mathcal{A}$, here we allow the codeword to be also $\mathcal{T}(j)$ as long as $k_i - \ell(j) \ge u$. Also observe that in all instances where a codeword $\varphi_i(\mathcal{T}(j))$ can be generated we necessarily have $\ell(\mathcal{T}(j)) = \Delta_i + 1$ and, so, $\varphi_i(\mathcal{T}(j)) = \mathcal{A}(j)$.

Yet, on the other hand, we require that two codeword candidates for the same input tag have different parity, label edges that terminate in the same state, and be located in $\mathcal{T}||\mathcal{A}$ at addresses 256 apart. Due to those conditions, only 53 entries in $\mathcal{A}$ are accessible by $\mathcal{E}_{(3,10)}$, compared to 64 entries in Section 3.2.

A block decoder $\boldsymbol{w} \mapsto \mathcal{D}_{(2,10)}(\boldsymbol{w})$ of $\mathcal{E}_{(2,10)}$ is obtained by deleting the two most-

significant bits of the 10-bit address of the entry in $\mathcal{T}||\mathcal{A}$ that contains the codeword $\boldsymbol{w}$. When restricted to the domain $\Sigma(\mathcal{E}_{(3,10)})$, this is also a block decoder of $\mathcal{E}_{(3,10)}$, with the range consisting of bytes having least-significant bits '11'.

The encoder $\mathcal{E}_{(3,10)}$ is weakly-observable from $\mathcal{E}_{(2,10)}$. Nesting and full observability can be attained if we do not exclude the 28-bit pattern '00010001...0001 from appearing in the bit stream; we then need to slightly modify $\mathcal{T}||\mathcal{A}$ and unmerge state $[2,5]$ in $\mathcal{E}_{(2,10)}$ into states 2 and $[3,5]$.

The power spectral densities of the two encoders are shown in Figure 4. We have used the same scaling of the axes as in [9] and applied the same local optimization (through encoding look-ahead) when selecting the generated codeword between two codeword candidates. The power spectral density of $\mathcal{E}_{(2,10)}$ is virtually the same as that of the $(2,10)$-RLL encoder in [9].

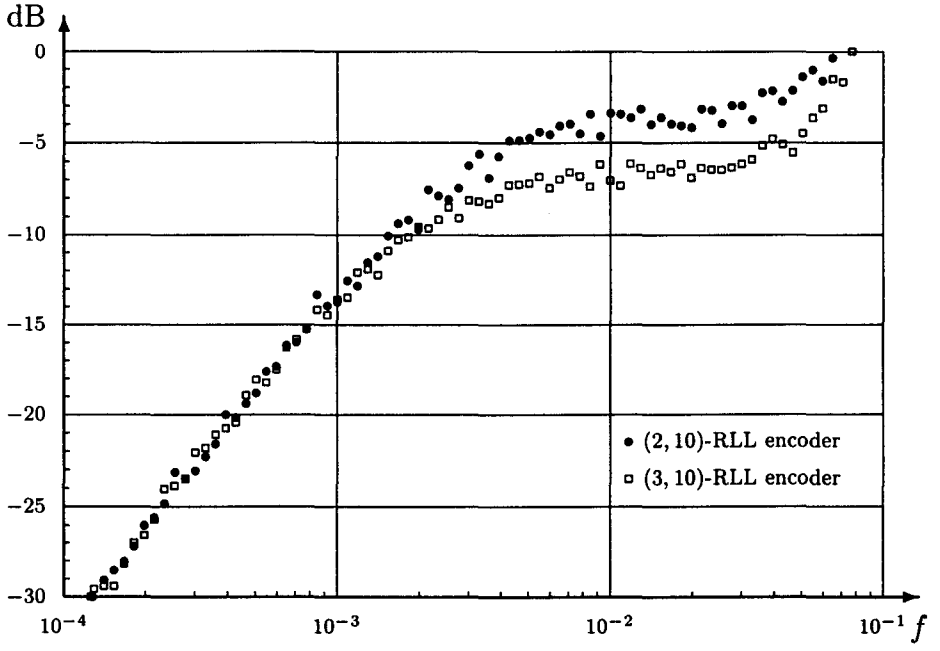

Figure 4: Power spectral densities of $\mathcal{E}_{(2,10)}$ and $\mathcal{E}_{(3,10)}$, with encoding look-ahead of two bytes.

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] G.F.M. BEENKER, K.A.S. IMMINK, *A generalized method for encoding and decoding run-length-limited binary sequences, IEEE Trans. Inform. Theory,* 29 (1983), 751–754.

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

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

[5] J. HOGAN, R.M. ROTH, G. RUCKENSTEIN, *Nested input-constrained codes, IEEE Transactions on Information Theory,* to appear.

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

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

[8] 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.

[9] R.M. ROTH, *On runlength-limited coding with DC control, IEEE Trans. Commun.,* 48 (2000), 351–358.

[10] C.E. SHANNON, *The mathematical theory of communication, Bell Sys. Tech. J.,* 27 (1948), 379–423.

addr. (dec.)	Contents of table (hexadecimal)															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0024	0021	0022	**0042**	2222	0020	2221	**0040**	2220	2224	0048	**0084**	0041	0049	0081	**0102**
16	0201	0249	0089	**0202**	0109	0111	0121	**0082**	0101	0091	0092	**0080**	0122	0124	0224	**0100**
32	0248	0112	0244	**0104**	0211	0221	0241	**0208**	0222	0242	0044	**0088**	0090	0209	0108	**0110**
48	0120	0204	0212	**0210**	0220	0240	0401	**0422**	0489	0491	0801	**0442**	0889	0891	0909	**0842**
64	0921	1049	0409	**0404**	0421	0441	0481	**0804**	0811	0821	0841	**0402**	0901	0881	0492	**0802**
80	0892	0912	0922	**1004**	0449	0849	0482	**0440**	0822	0911	0882	**0410**	0411	0412	0902	**0420**
96	0809	0484	0844	**0820**	0424	0890	0448	**0808**	0488	0490	0824	**0810**	0848	0884	0812	**0888**
112	0904	0908	0910	**1088**	0880	0480	0840	**1110**	0900	1089	1091	**1002**	1241	1121	1209	**1102**
128	1221	1009	1011	**1022**	1041	1081	1101	**1082**	1249	1109	1092	**1044**	1122	1212	1222	**1104**
144	1012	1021	1042	**1084**	1211	1202	0924	**0408**	1124	1024	0444	**1008**	1224	1244	1248	**1010**
160	1112	1048	1201	**1020**	1090	1242	0920	**1108**	1120	1204	1208	**1040**	1220	1210	1080	**1100**
176	1240	2089	2091	**2102**	2111	2121	2209	**2022**	2049	2241	2409	**2202**	2421	2441	2481	**2004**
192	2011	2021	2041	**2008**	2101	2201	2249	**2010**	2449	2489	2491	**2020**	2122	2212	2092	**2088**
208	2412	2422	2442	**2100**	2012	2211	2492	**2082**	2109	2411	2402	**2042**	2081	2401	2112	**2110**
224	2009	2244	2248	**2210**	2444	2448	2484	**2044**	2490	2488	2048	**2084**	2242	2090	2408	**2108**
240	2124	2120	2204	**2208**	2424	2024	2404	**2104**	2410	2420	2480	**2080**	2482	2240	2440	**2040**
256	4804	4201	4892	**4402**	4492	4890	4041	**4100**	4124	4810	4808	**4404**	4101	4809	4081	**4102**
272	4249	4891	4811	**4082**	4409	4821	4411	**4202**	4489	4209	4812	**4080**	4412	4824	4848	**4040**
288	4888	4822	4448	**4204**	4211	4841	4421	**4408**	4212	4842	4924	**4410**	4820	4109	4120	**4420**
304	4048	4090	4112	**4108**	4024	4840	4011	**4442**	4921	4491	4009	**4222**	4889	4909	4849	**4422**
320	4449	4911	4089	**4104**	4221	4881	4441	**4084**	4111	4121	4901	**4022**	4481	4091	4922	**4042**
336	4912	4012	4802	**4044**	4401	4801	4882	**4440**	4122	4021	4482	**4210**	4241	4902	4242	**4110**
352	4049	4490	4844	**4220**	4424	4224	4910	**4088**	4488	4884	4908	**4208**	4248	4904	4092	**4020**
368	4484	4244	4920	**4008**	4880	4480	4900	**4010**	4240	9109	9209	**8022**	8489	8041	9121	**8822**
384	8491	9041	9021	**8842**	9011	8809	8409	**8442**	8811	9111	9222	**8844**	9112	8402	9242	**8444**
400	9082	9249	9022	**8884**	9221	8412	9008	**8408**	8404	9208	9048	**8410**	8804	9010	9004	**8210**
416	9122	8908	8821	**8420**	8224	8922	8910	**8010**	8248	9120	9090	**8880**	8448	9110	9040	**8440**
432	8040	8909	8021	**8222**	8249	8011	8921	**8882**	8081	8449	8201	**8422**	8911	9211	8889	**8204**
448	8411	8209	9081	**8220**	8841	8211	8109	**8088**	8089	8441	8881	**8208**	8492	8202	8912	**8020**
464	8802	8892	8102	**8840**	8112	8849	9042	8212	8101	8891	9102	8122	9101	8221	8082	8488
480	8241	9124	8808	9210	9244	9020	8820	9104	8104	9248	8124	9108	9212	8904	8848	8424
496	9224	8484	9044	8890	8108	8244	8920	9220	9088	8490	8080	9100	8042	8100	9240	8900
512	8824	8901	8242	9202	8902	8084	8091	8480	8044	9084	9204	9024	8481	9049	9201	8482
528	8121	8049	9241	8092	9091	8401	8801	9012	9009	9089	9002	8240	8012	8048	8024	9080
544	8120	9092	8924													

Table 4: Encoding table $\mathcal{T}||\mathcal{A}$. The boldface entries indicate codewords accessed by $\mathcal{E}_{(3,10)}$.