FACTA UNIVERSITATIS (NIŠ)
Series: Electronics and Energetics vol. 13, No. 3, December 2000, 263-296

EFM++: AN EFFICIENT CODING FORMAT FOR DVD

Bane Vasić, Goran Đorđević and Milorad Tosić

Invited paper

Abstract In Digital Versatile Disc systems [3] the low frequency content of the channel modulation code needs to be minimized in order to reduce interference with the tracking servo system. The modulation code standard for DVD is so called Eight-to-Fourteen Plus (EFM+) code, which is a rate 8/16 runlength limited (d,k)=(2,10) four state code. The task of constructing simple runlength code capable of controlling low frequency content is difficult, and the careful look at the EFM+ code reveals several 'tricks' that made it possible. Certainly the most important of them is the code 'multimodality', which means that the choice of the output word is depends not only on the finite state machine state and the input word, but also of some external quantity (in the EFM+ case it is the running digital sum (RDS)). The DVD specifications [10] define the EFM+ decoder, but leave large freedom in construction of the encoder. More precisely, the encoding rule is considered valid as long as decoder can decode codeword properly. This motivated the question whether it is possible to find better codeword selection rule then the rule described in original EFM+. This paper shows that the answer is positive, and describes an EFM+ like modulation code which provides very efficient low-frequency spectral content suppression in the coded signal. An anticipation is introduced in the encoding algorithm, so that the selection of the codewords is based on permanent control of the recorded sequence RDS. The selection of the optimal codeword in the algorithm is performed by using the trellis searching algorithm for the RDS control and optimal codeword tra-

Manuscript received Oktober. 25, 2000. Part of this work will appear in IEEE Journal on Selected Areas in Communications in Special Issue on Signal Processing for High Density Storage Channels.

B. Vasić was with Bell Laboratories Lucent Technologies. He is now with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, Arizona (e-mail: vasic@ece.arizona.edu).

G. Djordjević is with the School of Electronic Engineering, University of Nis, Beogradska 14, P.O.B.73, 18000 Niš, Serbia.

M. Tosic was with Rutgers Univesity. He is now with Yipee Inc, Rochester New York.

cking. The papers also gives the code analysis, the comparison with theoretically achievable performances and describes in detail hardware implementation of the encoder.

Key words: Coding format EFM++, DVD, encoding algorithm, trellis searching algorithm.

1. Introduction

EFM+ [3] is a modulation code used to transform an arbitrary sequence of input data to the sequence with special runlength properties required for recording on the Digital Versatile Discs (DVD). Specifically EFM+ is an $(d, k) = (2, 10)$ code, which means that runlengths of consecutive like symbols in the precoded stream are in the range $[d + 1, k + 1]$. Parameter d controls the highest transition frequency while the parameter k ensures adequate frequency of transitions for synchronization of the read clock. The EFM+ coding rule is defined by a state finite automaton. The state and the output functions of this automaton are defined so that for some input words there exist an alternative next state and alternative output word. The flexibility in the choice of the mapping the input to the output word (so called the code multimodality) is introduced in order to control the low frequency content of the coded signal. The flexibility in the automaton next and output functions can be viewed as a existence of variety of valid code sequences corresponding to the same input sequence. The term "valid" is used in strict sense here, meaning that the codeword sequence is decodable by the original EFM+ decoder. In other words in all our considerations we assume that the decoding algorithm is fixed and that changes are allowed only in encoder. Suppression of the low frequency spectral components is employed in the DVD system primarily to reduce interference of the low-frequency components with the low frequency signals in the tracking servo systems. According to Immink [3], the servo system is the Achilles' heel of the player as errors correction is totally useless if track or clock loss occurs. Further, the control of the low frequency content of the recorded signal is important for automatic gain control during detection. Finally, if the recorded sequence is dc-free (this notion will be defined more precisely later), then the low-frequency disturbances resulting from fingerprints on the disk can be filtered out without distorting the data signal itself.

Low-frequency content of the recorded signal is closely related to the running digital sum (RDS) of the recorded sequence. Famous RDS theorem [7] establishes the equivalence between spectral zero at zero frequency of some stream and finite range of values that RDS of the stream can take.

The original EFM+ coding rule is such that it controls RDS. In the basic variant of the EFM+ code the choice of the input to output word mapping is such that that the selected output word minimally increases the RDS at the end of the current word. In this way RDS is always in the range $[-RDS_{max}, RDS_{max}]$, where RDS_{max} is several hundreds[1].

It is easy to see that look-ahead algorithm can provide further improvement in RDS control. Look-ahead means that choice of the codewords is based on the RDS after several codewords (at the end of the look-ahead window). Two variants of the EFM+ code with one and two codeword look-ahead have been proposed [3]. As we will see one codeword look-ahead this method is equivalent to the Fano trellis search algorithm [11, pp. 334-344] which traces only two alternative path trough trellis defined by the encoder finite automaton.

Two-word look-ahead is analyzed in [3], and it is shown that it can provide 3dB decrease of the power spectral density (PSD) at low frequencies [3]. No solution for the look-ahead EFM+ encoder has been presented. For all look-ahead methods the minimization function (or metric) is the value of RDS at the end of look-ahead window.

This paper describes an EFM+ like code, named EFM++, that provides an improved way of suppression of the spectral content of the coded signal. The idea motivated the research reported in this paper was to use inherent trellis structure of the code for permanent RDS control by an efficient trellis search algorithm. The modification of the EFM+ code, proposed in this paper, is twofold: Firstly, an anticipation is introduced in the coding algorithm so that selection of the codeword is not based on the local, but on the global optimality criterion of recorded sequence RDS. More precisely the trellis search metric that shows the best performances is based on the control of the second order RDS (defined later). It should be noted that the notion of anticipation of our algorithm differs from Immink's one in sense that our metric takes into account not only the final RDS value at the end of anticipation window, but more complex value capturing the behavior of RDS inside the window. Second modification is that the selection of the optimal codewords in the algorithm is performed by the efficient trellis searching algorithm.

We describe several variants of EFM++ code with different RDS con-

[1]Some parameters of the EFM+ are easy to calculate from the coding table, but since full coding table is not a public information, the authors are forced to avoid giving more details then in [Immink].

trol strength, and give the spectral analysis of these variants. We also propose the hardware realization of the EFM++ encoder. Section II gives the basic definitions related to the spectrum and RDS properties of the codes with low frequency spectral control (usually called dc-free codes). This section also explains the motivation of whole research and expected results. Section III is devoted to the explanation of the variants of EFM+ encoding algorithm. Section IV gives an review of strategies and possible modifications of the original EFM+ coding scheme that can be used to improve low frequency content suppression in the coded signal. In Section V the introduced EFM++ coding format is described. Different variants of the EFM++ codes are discussed and compared in terms of their encoding trees, path metrics and trellis search algorithms. Two search algorithms are considered: with block look-ahead window and with sliding window. The basic hardware structure of the encoder is presented. In Section VI the performance evaluation of EFM++ code is given. The study of auxiliary performance parameters σ_{RDS} and σ_{RDS2} is given. These parameters are used in the process of code design. Power spectral densities of two existing EFM+ encoding methods and EFM++ are calculated and compared at low frequencies. In Section VII the hardware structure and complexity of a window look-ahead EFM++ is analyzed. A comparison is given between so called free-running mode, where there is no limit in the trellis search depth, and the fixed output rate mode where trellis search is limited by speed of whole circuitry. Section VII gives some conclusions.

2. Definitions and Theoretical Boundaries

2.1 Finite automaton model of the encoder

The block diagram of a modulation encoding is given in Fig. 1. The input of the modulation encoder is a sequence $\{x^{(k)}\}$ of the binary source words of length m (k denotes discrete time). As a response to each input word $x^{(k)}$, the modulation encoder, according to its encoding algorithm, generates the binary output codeword $y^{(k)}$ of length n. We reefer to the ratio $R = m/n$ as a code rate. For EFM+ code $m = 8$, $n = 16$, and $R = 1/2$. Note that there might be a delay, l, in the modulation encoder, so that it is more accurate to denote the response to the $x^{(k)}$ by $y^{(k+l)}$, but since the delay does not affect the function of the blocks, we will assume $l = 0$.

Commonly a modulation code rule is described by a finite automaton. The finite automaton is defined by next-state function N, $N : S \times X \rightarrow S$,

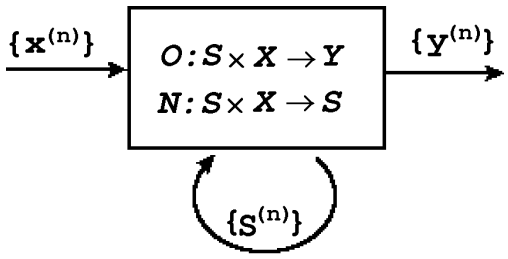


Fig. 1. Finite automaton model of a modulation encoder.

and output function O, $O : S \times X \rightarrow Y$, wherein X, S and Y represents the set of all source words, the set of finite automaton states and the set of codewords, respectively. In optical recording Y is a binary alphabet $\{0, 1\}$. Typically, the codeword sequence $\{y^{(k)}\}$ is precoded so that symbol 1 in the $\{y^{(k)}\}$ causes the transition in the precoded stream. For example if $y^{(k)} = (0001000100000000)$, then precoded version of it can be $z^{(k)} = (-1 - 1 - 1 + 1 + 1 + 1 + 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1)$ or $z^{(k)} = (+1 + 1 + 1 - 1 - 1 - 1 - 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1)$, depending on the last symbol in the $z^{(k-1)}$.

2.2 Statistical properties of the sequences generated by finite automata

Consider bipolar channel sequence $\{a^{(k)}\}_{k>0}$ (bipolar means that $a^{(k)} \in \{-1, +1\}$). Running digital sum (of order 1) of the $\{a^{(k)}\}_{k>0}$ is defined as

$$\rho^{(k-1)} = \rho^{(k-1)} + a^{(k)}, \quad \rho^{(0)} = 0$$

Running digital sum of order N is defined as

$$\rho_N^{(k)} = \rho_N^{(k-1)} + \rho_{N-1}^{(k)}, \quad \rho_n^{(0)} = 0,$$

wherein $\rho_0^{(k)} = a^{(k)}$ and $\rho_1^{(k)} = \rho^{(k)}$.

Running digital sum at the end of the (precoded) codeword $z^{(k)}$, under the assumption that last symbol in $z^{(k-1)}$ was -1, is called codeword disparity, D. For example disparity of $y^{(k)} = (0001000100000000)$ is $D(y^{(k)}) = -9$.

The next important statistics of the stream $\{a^{(k)}\}$ is an autocorrelation function $r^{(k)}$, and it is defined by $r^{(k)} = E\{a^{(n)} \cdot a^{(n+k)}\}$, where $E\{\}$ denotes operator of averaging over the ensemble. The autocorrelation function is an

even and decreasing function ($r^{(-k)} = r^{(k)}$ for all k, and $r^{(k+1)} \leq r^{(k)}$ for all $k > 0$).

When random process $\{a^{(n)}\}$ is generated by finite automaton it can be shown that it belongs to the class of so called cyclostationary processes generated by ergodic Markov source, and it is then possible to define power spectral density (by introducing the random phase τ i.e. by considering $\{a^{(n+\tau)}\}$ (see e.g. [2, pp. 19.]).

The power spectral density (or spectrum) of the stream a(k) generated by the ergodic Markov source, is given by

$$\Phi(f) = \sum_{k \in Z} r^{(k)} \cdot e^{-j2\pi kf}$$

where $j = \sqrt{-1}$, and f is the normalized frequency ($0 \leq f < 1$). Power spectral density function can be written as a sum of two components: continual and discrete spectrum. Discrete spectrum (called also spectral lines) indicates that there is a periodic component in the sequence $\{a^{(k)}\}$. The dc-free requirements means that both discrete and continuous spectral components are zero at zero frequency. The running digital sums play an important role in the estimating the spectral properties of a code. The following facts establish the connection between the running digital sums and the content of the low-frequency end of the spectrum:

–Denote by Θ_1 set of values that sequence $\{\rho_1^{(k)}\}$ can take. Digital sum variation (DSV) is the related to the cardinality of this set (usually Θ_1 is defined as $\Theta_1 = \{-C1, \ldots, C1\}$, and for this symmetric case $DSV = 2C + 1$). If DSV is finite, the spectral density is zero at zero frequency [7], and these codes are called dc-free codes. Low frequency suppression is directly proportional to the value of DSV.

–If the RDS value at the end of the encoded sequence is zero ($\rho^{(+\infty)} = 0$), then the discrete spectrum is zero at zero frequency.

–For dc-free codes, the width of a region of frequencies, close to zero frequency, where the spectral density is low is called notch width and is quantified by a parameter called cutoff frequency, f_0 (defined as $\Phi(f_0) = 1/2$ [5]). The following approximate relationship exists between the sum variance of RDS values, $\sigma_\rho^2 (= \sigma_{RDS}^2)$, and cutoff frequency

$$2\sigma_\rho^2 f_0 \simeq \frac{1}{2\pi}$$

–Denote by Θ_N set of values that sequence $\{\rho_N^{(k)}\}$ can take. If the cardinality of set Θ_1 is finite, then first $2N - 1$ derivatives of the power

spectral density function at $f = 0$ are equal to zero. This means that in the low frequency region $\Phi(f)$ can be written in the form (more elaborate discussion can be found in [6])

$$\Phi(f) = \frac{\Phi^{(2N-1)}(0)}{(2N-1)!} f^{2N-1} + O(f^{2N-1})$$

The "slope" of the curve defined by $\Phi^{(2N-1)}(0)$ depends on the cardinality $|\Theta_N|$. Practically important is symmetric case $\Theta_N = \{-C_N, \ldots, C_N\}$, where $|\Theta_N| = 2C_N + 1$.

An important conclusion that follows from the properties of RDS is that the low frequency content of a coded signal depends on:

- maximal value that RDS takes during encoding ($\max_{n>0}\{\rho^{(n)}\}$) and
- variance of RDS values σ_ρ^2,

and that EFM+ algorithm does not use any of these metrics for codeword selection. EFM+ code uses minimization of RDS. However, since RDS can be negative and therefore does not satisfy triangle inequality it cannot be used as a metric.

2.3 Theoretical limits of low frequency component suppression

A constraint graphs and the best codes

In order to find theoretical limits of the spectral performances of the code we want to construct let us define a sofic system Δ as a set of all bi-infinite sequences generated by walks on a directed graph $G = G(\Delta)$ whose edges are labeled by symbols in a finite alphabet A. The graph $G = (V, E, \pi$ is given by a finite set of vertices (or states) V, a finite set of directed edges E, and a labeling $\pi : E \to A$. So, for a given sequence of edges $\{e^{(k)}\}$ ($e^{(k)} \in E$), we have the output sequence $\{a^{(k)} = \pi(e^{(k)})\}$. Fig. 2 shows the graph $G_{(d,k)}$ of the (d, k) constraint.

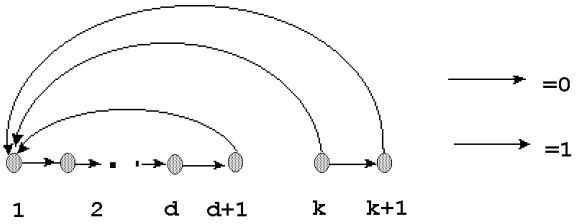


Fig. 2. $G_{(d,k)}$ graph.

A graph G is strongly connected if for every two vertices $u, v \in V$ there exists a path (sequence of edges) from u to v. A graph, G, is deterministic if for each state $v \in V$, the outgoing edges from v, $E(v)$, are distinctly labeled. In practice deterministic and strongly connected representations of a constraint are used.

The adjacency matrix (or vertex transition matrix) $\boldsymbol{D}(G) = \boldsymbol{D} = [D(u,v)]_{u,v \in V}$ of graph G is $|V| \times |V|$ matrix where entry $D(u,v)$ is the number of edges from vertex u to vertex v, and $|V|$ is the number of vertices of the graph G.

The capacity, C, of the sofic system Δ presented by the deterministic graph G is defined as $C = \log_2(\lambda_0)$, where $\lambda_0 = \lambda_0(G)$ is the spectral radius (i.e. the largest of the absolute values of the eigenvalues [8]) of adjacency matrix, $\boldsymbol{D}(G)$.

The graph representation of a given constraint is very important because there is a systematic algorithm for code construction that starts from a directed graph. This algorithm, called ACH algorithm [1], gives the encoder in the form of finite automaton and sliding window decoder with limited error-propagation. Graph capacity, C, gives the maximal possible rate that code developed by ACH algorithm can achieve.

B Composition of (d,k) and RDS constraint

It is straightforward to define graph models for (d,k) constraint, but much more difficult to obtain explicitly graphs for the RDS constraints (The order N RDS constraint is denoted $RDS-N$). Examples of the graphs of the $RDS1$ and $RDS2$ constraints are shown in Fig. 3 and Fig. 4. The axes are labeled by current values of RDSs.

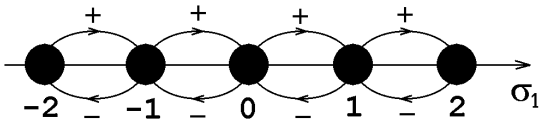


Fig. 3. $RDS1$ Constraint graph ($\Theta_1 = \{-2,-1,0,1,2\}$).

Moreover, if one wants to construct the code that is (d,k) and in the same time $RDS-N$ constrained, the another, much more complicated graph operation over graphs $G_{(d,k)}$ and G_{RDS-N} should be performed. This operation is called fiber product.

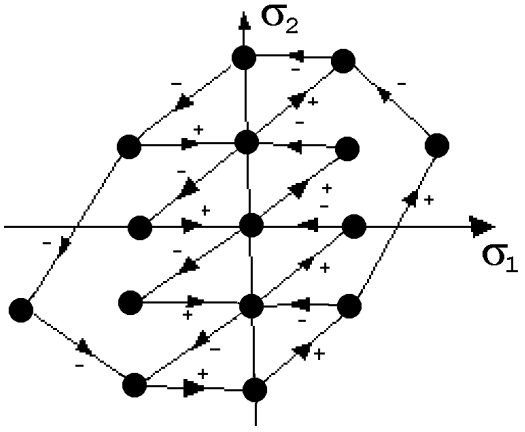


Fig. 4. RDS2 Constraint Graph ($\Theta_2 = \{-2, -1, 0, 1, 2\}$).

The fiber product of the graphs $G_0 = (V_0, E_0, \pi_0)$ and $G_1 = (V_1, E_1, \pi_1)$, $G = G_0 * G_1$, is the Shannon cover of the graph $G = (V, E, \pi)$ for which $V = V_0 \times V_1$, ($\times$ denotes Cartesian product of the sets) and for every edge e_0 from u_0 to v_0 in G_0 and every edge e_1 from u_1 to v_1 in G_1, there exists an edge e ($e = (e_0, e_1)$) in G, emanating from vertex $u = (u_0, u_1) \in V$ and terminating at $v = (v_0, v_1) \in V$ if $\pi_0(e_0) = \pi_1(e_1)$. The label of this edge is $\pi(e) = \pi_0(e_0) = \pi_1(e_1)$. Example of the fiber product is given in Fig. 5. Shaded region denotes the Shannon cover, i.e. irreducible nonzero capacity part.

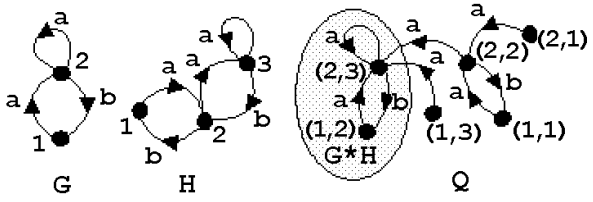


Fig. 5. Illustration of the graph fiber product

It is important to say that codes defined as a composition of $G_{(d,k)}$ and G_{RDS1} or G_{RDS2} constraints are codes with the superior spectral performances for a given code rate (since they produce the sequences with maximal uncertainty, these codes are called maxentropic codes). However, the complexity of their finite automata (obtained by ACH algorithm) grows rapidly with the cardinality of the constituent graphs.

Consider now particular $(d,k) = (2,10)$ constraint combined with $RDS - N$ constraints ($N = 1$, and $N = 2$). If one wants to obtain the code with rate $R = 1/2$, the minimal values of digital sum variations are $C = 7$ for $N = 1$, and $C = 30$ for $N = 2$. Resulting spectra are shown in Fig. 6. It is clear that $RDS2$ constrained sequence has much more suppressed low frequency components.

Since our encoding algorithm is less complex than required by ACH algorithm, it is clear that these theoretical limits cannot be achieved. Our goal is to reduce low frequency content in the recorded signal under constraints given by original EFM+ coding format in DVD specifications.

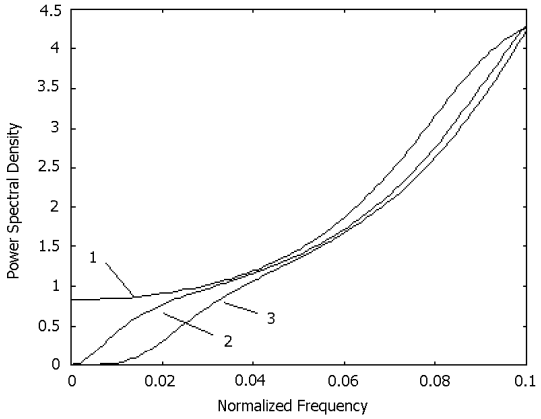


Fig. 6. Power spectral density of different types of (2,10) codes with rates $R \geq 1/2$
1) maxentropic (2,10) sequence;
2) maxentropic (2,10) RDS constrained sequence with $M = 1$, $C = 7$;
3) maxentropic (2,10) RDS constrained sequence with $M = 2$, $C = 30$.

C Multimode codes

As it has been explained in section *B*, even that ACH codes are best codes speaking the information theory language, they are sometimes complicated to implement, so, in some cases it is worth to search for another solution.

Another important result of the research of EFM+ code is the understanding that EFM+ code is not more than the special case of so called multimode codes. The elaboration of multimode codes and their application is beyond scope of this paper, and will be explained only in the extent to provide the reader basis to understand the theoretical setting of trellis search as a method of low frequency control. Multimode codes might have

very important implication on practical code design.

Commonly, there is one-to-one relationship between source words and codewords. These codes are referred to as monomode codes [4]. Multimode code is a code that have more than one next state and more than one output function (but still have a unique sliding-window decoder). Generally, a multimode encoder is defined by a set A of I pairs of next-state and output functions $A = \{(N_i, O_i), 1 \leq i \leq I\}$, and a selection function Ψ. Selection function block is fed by state and output sequence vectors $\{\boldsymbol{s}^{(n)}\}$ and $\{\boldsymbol{y}^{(n)}\}$ and produces a sequence $\{i^{(n)}\}$ of numbers from set $\{1, \ldots, I\}$ that selects one of the coded streams $\{y_1^{(n)}\}, \ldots, \{y_I^{(n)}\}$ as an output stream. In practice memory of the selection block is limited, say to D words, so that formally $\Psi : S^{I \cdot D} \times Y^{I \cdot D} \rightarrow \{1, \ldots, I\}$.

Generally the selection function is a function of the vector sequences $\{\boldsymbol{s}^{(n)}\}$ and $\{\boldsymbol{y}^{(n)}\}$ ($\boldsymbol{s}^{(n)} = (s_1^{(n)}, \ldots, s_I^{(n)})$ and $\boldsymbol{y}^{(n)} = (y_1^{(n)}, \ldots, y_I^{(n)})$).

Let us remind once again that the role of a decoder is to convert the coded stream $\{y^{(n)}\}$ to stream $\{x^{(n)}\}$, and that decoder does it equally well for any O_i and N_i. Typical decoder is a sliding-window decoder. This type of decoder makes a decision on a word $x^{(n)}$ on the basis on m previous received words $y^{(n-m)}, \ldots, y^{(n-1)}$, current word $y^{(n)}$ and a following words $y^{(n+1)}, \ldots, y^{(n+a)}$. Numbers m and a are called decoder memory and decoder anticipation [9].

Crucial problem in multimnode codes is a choice of selection function or and its complexity. Our paper does not give theoretical answer to this problem, but proposes some simple selection function. Selection functions are in the form of accumulating metrics, similarly as in Viterbi detection. Metrics used are heuristic, although they are derived from the theoretical analysis of connections between RDS variance and low frequency content. The rest of the paper is devoted to the discussion on different search algorithms and metrics.

3. The EFM+ Code

In this section the principle of EFM+ coding is explained. We will keep terminology from DVD Specifications (hopefully it will help the most interested readers that are familiar with them) although it is not always quite consistent. The terms from the DVD specifications are capitalized. It is worth noting that EFM+ is not really only a code, it is a coding format since it uses also synchronization (SYNC) words for low frequency control, and it can be also called the encoding algorithm since it uses re-encoding.

3.1 EFM+ coding rule

EFM+ is a $(d, k) = (2, 10)$ modulation code which is used to convert 8-bit input data symbols into 16-bit channel symbols. The channel bit sequence is characterized by two parameters, $d+1 = 3$ and $k+1 = 11$, which specifies the minimum and maximum runlength (i.e., the number of consecutive like symbols), respectively, that may occur in a sequence.

The principle of operation of the EFM+ encoder can be represented by a four state finite automaton. In any given automaton state, by using the Conversion Table associated with the state, the 8-bit input data symbol is converted into 16-bit Code Word, and a new state of the machine is determined. EFM+ finite automaton guaranties that the concatenation of Code Words produces a valid (2,10)-constrained sequence. Under conditions listed below, EFM+ encoder allows one input data to be converted into more than one alternative Code Words:

c1) (Main/Substitution Code Words) The input bytes 0,..,87 (these numbers are byte decimal representations) can be encoded by using either the Main Conversion Table or the Substitution Conversion Table.

c2) ("1-4" exchange) When the next state is 1 or 4, the next input byte may be encoded with either State 1 tables or State 4 tables, if the (2,10)-constraint is satisfied.

c3) (SYNC code selection) Each SYNC word can be encoded by using one of many sequences of length L (given in [10]) obtained by concatenating words that are not valid EFM+ codewords. The selection of SYNC word depends on RDS.

Combined, conditions c1 and c2 may give at least two and at most four alternative representations for the input bytes less than 87, and at most two representations for input bytes greater than or equal to 88.

Alternative encoding represents a crucial property of the EFM+ code that enables low frequency control - always when alternative encoding is allowed, the encoder opts for transmitting that Code Word that minimizes, according to a specific criteria, the low- frequency spectral contents of the encoded sequence.

3.2 Existing EFM+ encoding algorithms

The existing EFM+ encoding algorithms suppress the low-frequency components by controlling the running digital sum (RDS) of the encoded sequence. The RDS is the difference between totals of "+" and "−" sym-

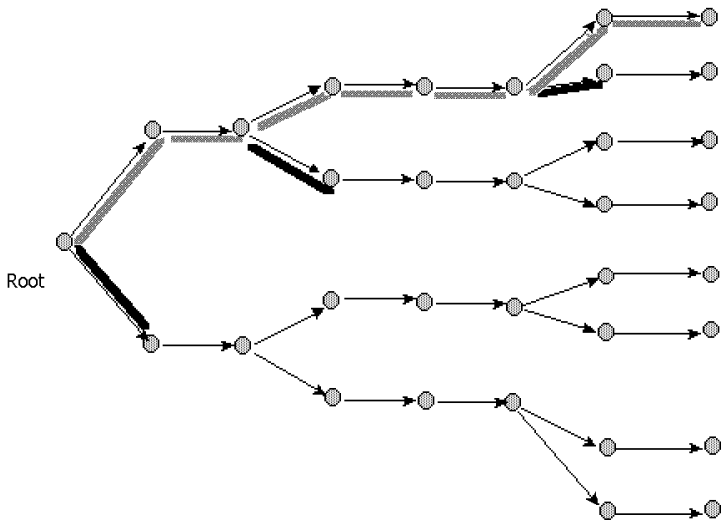
bols accumulated from the beginning of the sequence. The purpose of low frequency control is to produce such an encoded sequence so that the $|RDS|$ is as close to zero as possible for the duration of EFM+ encoding.

The existing EFM+ encoding algorithms use "1-4 alternation" for input bytes which are less than 88, only. Thus, the number of alternative Code Words per input data symbol is at most two.

The following two general approaches can be identified:

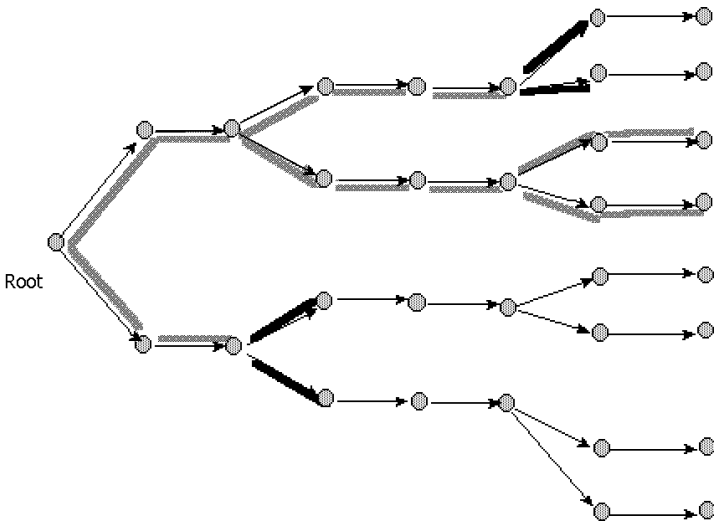
Method 1 (incremental encoding):

If a choice between two Code Words has to be made two RDS values are calculated (one for each choice of Code Word) and subsequently, the Code Word which produce the lower $|RDS|$ is selected. If we draw the sequence of all possible encoder states as a tree, then the Method 1 gives the selected sequence according the following gray bolded path (black branches represent dead paths)



Method 2 (two-path encoding):

Two parallel modulated channel bit streams are generated and buffered. When alternative appears, bit-stream with the lower $|RDS|$ is selected and output, another bit-stream is discarded, and two new bit streams which originate from the selected stream are continued. This method gives the following path through the encoder tree



This way of tree search is know as Fano algorithm.

Note that in the EFM+ coding tables there is no restriction that would prevent that "1-4 alternation" can be applied to any of the input bytes (decoder will work properly), but in order to keep the number of branches from each node less than two (otherwise three buffers would be required) the "artificial" constraint is introduced so that "1-4 alternation" is applicable only to bytes larger than 87. It is also worth noting that in EFM+ algorithm encoding tree has length of one sync frame.

In order to further reduce the $|RDS|$ at the end of each Sync frame, both algorithms, Method 1 and Method 2, can use some kind of smart SYNC word selection, but this will not be discussed in this paper.

4. More Efficient Method of Suppression of Low Frequency Content

Looking at encoding trees for Method 1 and Method 2, natural questions arises: Is there any better way of searching the encoder tree? What is the criterion for complexity comparison between different search strategies? Basically, the idea of this work is to shorten the encoding tree, but to allow to keeping track of more than two alternative paths.

4.1 Analysis of the low frequency control strategies in the existing EFM+ algorithms

Our observations concerning the low frequency control strategies implemented in existing EFM+ encoding algorithms, Method 1 and Method 2, are the following:

- Appalling "1-4 exchange" to the input bytes greater than 87, only, the number of alternative Code Words per input data symbol is limited to at most two. This limitation represents necessity in two-path algorithm. In the incremental encoding, the two-alternative scheme slightly reduces complexity, but on the other hand it may cause an unacceptable performance loss.
- The probability distributions of $|RDS|$ for both algorithms two-path and incremental, are shown in Fig. 7. As can be seen from Fig. 7, the variation of $|RDS|$ values is relatively small. For example, the probability for $|RDS|$ to be greater than 20 is less than 10-5 in both cases.

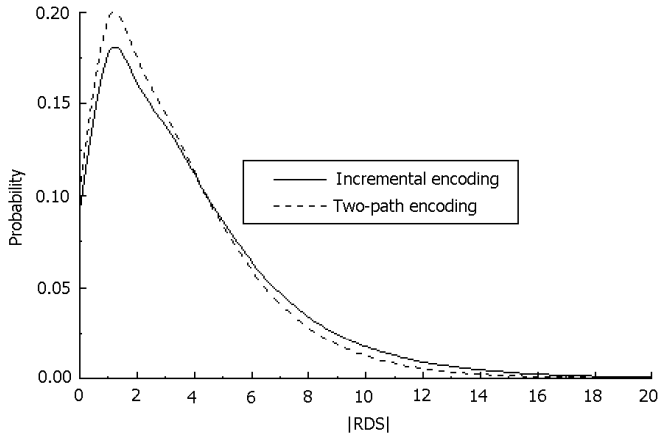


Fig. 7. $|RDS|$ distribution.

- In order to chose the best SYNC word, the complete Sync frame is needed, which means that implementation of this technique requires an output FIFO buffer for holding entire encoded Sync frame. The size of this buffer is 93 words, and it will probably be the dominant component (in terms of the chip occupied area) of the encoder.

- Two-path encoding algorithm posses look-ahead capability, but the look-ahead is limited to two alternative bit-streams, only. Also, this kind of look-ahead requires two buffers for holding two alternative bit-streams. The size of these buffers should be 93 words.
- Two-path encoding algorithm uses the $|RDS|$ at the end of bit-streams as a criterion for bit-stream selection, and does not consider variations of the $|RSD|$ along the bit- streams. As a consequence, bit-stream with larger variations of $|RDS|$ may be selected. An example of such wrong selection is given in Fig. 8. This figure shows the variation of the RDS along two alternative bit-streams. Although, the $|RDS|$ variations of bit-stream_1 is much larger than that of bit-stream_2, two-path encoding algorithm selects bit-stream_1, since it ends with lower $|RDS|$ value.

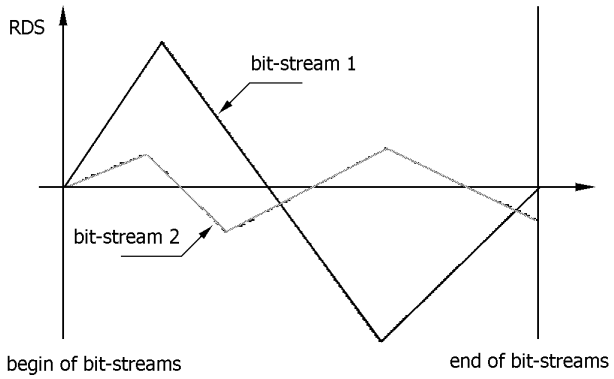


Fig. 8. Two-path selection strategy.

- Both incremental and two-path encoding algorithms control variations of the RDS, and do not consider variations of the $RDS2$. Our simulation study show that under this circumstances, the range of $RDS2$ variations is very large (e.g., within encoded sequence which is 107 bits long, $RDS2$ varies between -105 and 105).

4.2 Strategies for more efficient low frequency suppression

Theoretical results given in Section II and our observations about low frequency control strategies implemented in existing EFM+ encoding algorithms given in Section 3.2 set up the following main directions for low frequency control improvement:

1. Performance gain of "1-4 exchange" extension to all input bytes should be examined.
2. SYNC words are treated in the same way as input bytes less than 88.
3. More aggressive look-ahead strategies than that of two-path encoding algorithm should be examined.
4. Minimization of the $|RDS|$ at the end of the encoded sequence, which is used in the existing EFM+ encoding algorithms, should be replaced by the minimization of $|RDS|$ variation within the encoded sequence.
5. Explicit control $|RDS2|$ variation within the encoded sequence should be examined.

5. The EFM++ Encoders

The global structure of the proposed method for EFM+ encoder with the efficient low frequency suppression (or in short, EFM++ encoder) is given in Fig. 9.

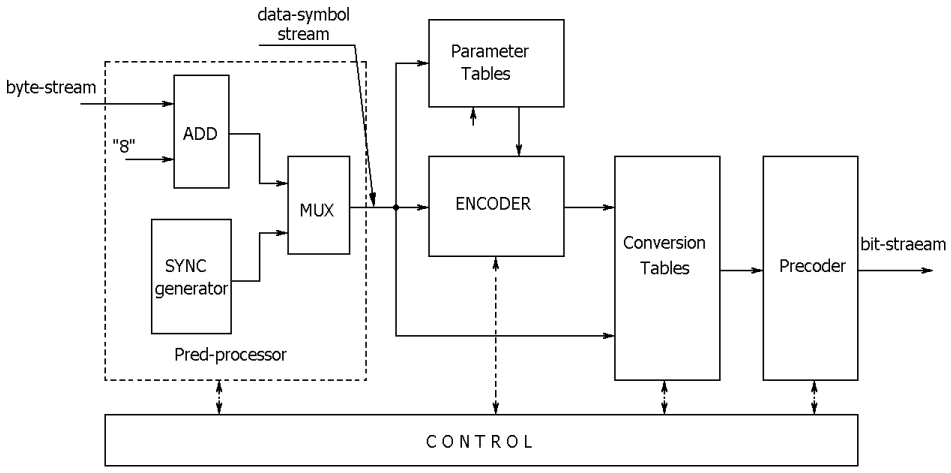


Fig. 9. Global structure of the EFM++ encoder.

The input to the system is the byte-stream, while the output is the modulated channel bit-stream. The Pred-processor blok forms 9-bit data-symbol-stream, by inserting into input byte-stream identifiers of appropriate SYNC words at the beginning of each Sync frame. Data symbol value

$d_{s\in[0,7]}$ identifies one of eight SYNC words, while data symbol value $d_{s\in[8,263]}$ corresponds to input bytes. The Encoder block implements encoding algorithm which task is to associate with each data symbol identifier of one of data-symbol's alternative Code Words. This identifier is than used to select Conversion Table from which the Code Word for the given data symbol is read. The Encoder block uses Parameter Tables to determine next state of the EFM+ state-machine and to calculate the parameters of the encoded bit-stream such as RDS and $RDS2$.

Figure 10 shows the organization of the Parameter and Conversion Tables.

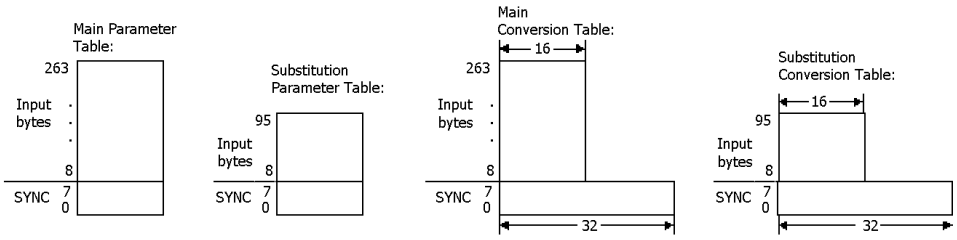


Fig. 10. Organization of the Parameter and Conversion Tables.

For each data symbol, Parameter Table contains "next encoder state" and the following parameters of the corresponding Code Word:

- disparity (for RDS calculation);
- second order disparity (for RDS2 calculation);
- length of the trailing phase;
- length of the leading phase;

The lengths of trailing and leading phases is used for chacking the possibility of "1-4 exchange".

5.1 Incremental EFM++ encoder

When encodes a data symbol, incremental EFM++ encoder examines each alternative Code Word of that symbol and chooses that one that yields the minimal —RDS—. With respect to existing incremental encoder, the incremental EFM++ encoder uses "1-4 exchange" for all input bytes, and does not make any difference between input bytes and SYNC words.

Incremental EFM++ encoder provides better suppression of low-frequency components than existing incremental encoder (see Fig. 12). However, this suppression is less than that of "two-path encoder". The same relationship between incremental EFM++ encoder and two existing encoders can be observed by comparing their $|RDS|$ variances. Namely, the $|RDS|$ variants of the incremental EFM++ encoder is 8.3, which is less than that of basic incremental encoder (10.7) and greater than that of "two-path" encoder (7.8).

5.2 EFM++ encoder with look-ahead

A Encoding tree

All valid encoded sequences that correspond to the given input data-symbol sequence can be represented by an encoding tree. An example of the encoding tree is given in Fig. 11. Each level in the encoding tree corresponds to one data-symbol and each path from the root to a leaf represents one valid encoded sequence. With each node in the tree a label indicating the current encoder state and Conversion Table is associated. For the example in Fig. 11, the number of alternative encoded sequences is 12.

Label MSi indicate that the encoder is in State_i, and the Main Conversion table is used; SSi indicates the use of Substitution Conversion table in State_i.

B Path metrics

For a given data-symbol sequence, the aim of a look-ahead encoding algorithm is to select the path in the encoding tree that minimizes the low-frequency spectral contents of the encoded sequence. The path selection criterion is based on specific *cost function* (or metric) which associates a nonnegative value to each path in the encoding tree. We propose the following three cost functions:

$$M1(\pi) = \sum_{i=1}^{n} C_i \tag{1}$$

$$M2(\pi) = \sum_{i=1}^{n} C_i^2 \tag{2}$$

$$M3(\pi) = \sum_{i=1}^{n} C_i^\star, \quad C_i^\star = \begin{cases} C_i - t, & C_i > t \\ 0, & C_i \leq t \end{cases} \tag{3}$$

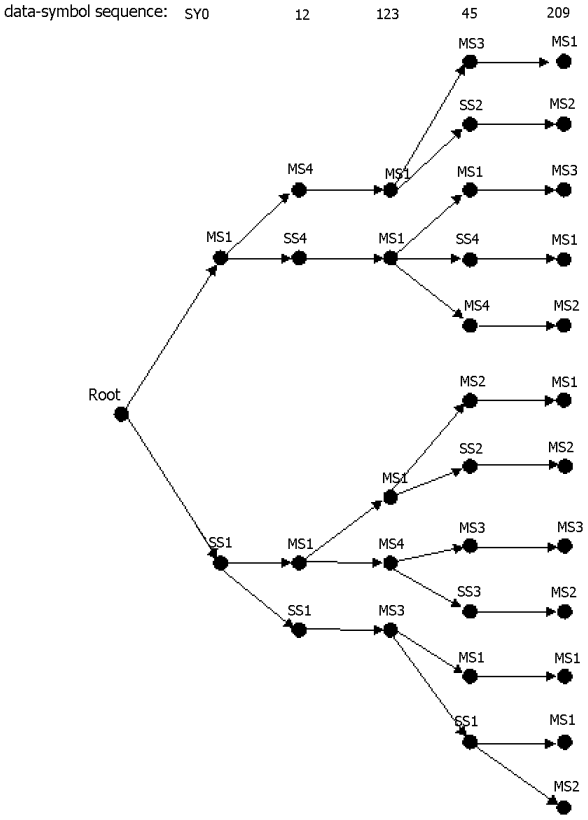


Fig. 11 The encoding tree.

where π is a path in the encoding tree, C_i is the value of $|RDS|$ or $|RDS2|$ at the i-th node in the path π, and t represents some threshold value.

All three cost functions are used to quantify the variation of $|RDS|$ (or $|RDS2|$) within the encoded sequence. Selected encoded sequence should be one that minimizes the adopted cost function.

C Encoding tree search

Our simulation shows that the average number on alternative Code Words per data symbol (i.e. *branching factor* of the encoding tree) is 1.49. This means that the number of alternative encoded sequence for the data symbol sequence of length n is approximately 1.49^n. Thus, the explicit enumeration of all paths in the encoding tree will have the exponential time and space complexity, and obviously represents an unacceptable solution.

In order to reduce the complexity of the encoding tree search we introduce the *n-path search algorithm*. In this algorithm, the number of paths in the enumeration tree that are constructed and examined is limited to n. To represent the operation of the n-path search algorithm we use a graph structure called *Trellis*. The nodes of the trellis are arranged into n rows of, in general, unlimited length. The task of the n-path algorithm is to embeds the encoding tree into trellis. When expands the trellis to a new level, algorithm can use only n nodes from the next column of the trellis. Thus, if the number of the paths that emerges from the current level is greater than n the algorithm selects and continues n paths, only.

Figure 12 shows an embedding of the encoding tree from Fig. 11 in the trellis of height 6. As can be seen, the first three levels of the encoding tree are completely embedded into trellis. The path reduction appears when the third level is expanded to forth level. The nodes in the forth level of the encoding tree is 12, while there are only 6 available nodes in the trellis. Thus, the algorithm continues 6 "best" paths and terminates the remaining 5. The selected paths should be ones with minimal current cost function values.

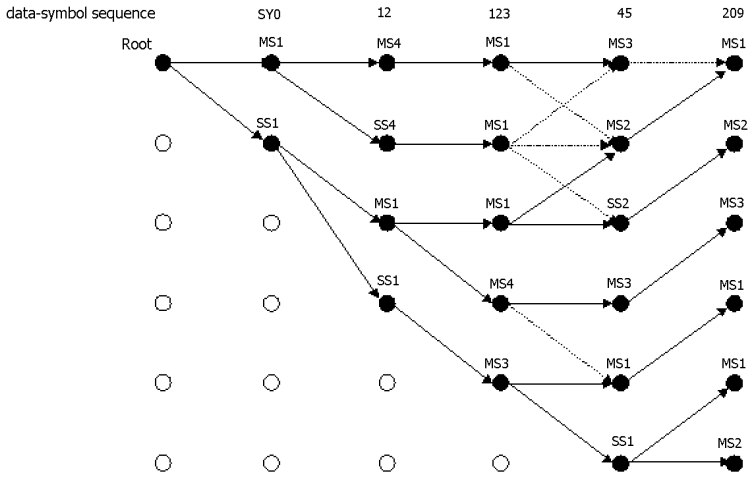


Fig. 12. Embedding the encoding tree from Fig. 7 into the trellis with height 6.

The n-path encoding tree search algorithm can be used for the following two basic look-ahead strategies:

- *Block-look-ahead.* The data-symbol sequence is divided into subsequences of fixed length l. For each subsequence, a new trellis is constructed, then the path in the trellis which ends with minimal value of the cost function is selected, and finally l Code Words of the corresponding encoding sequence is output.

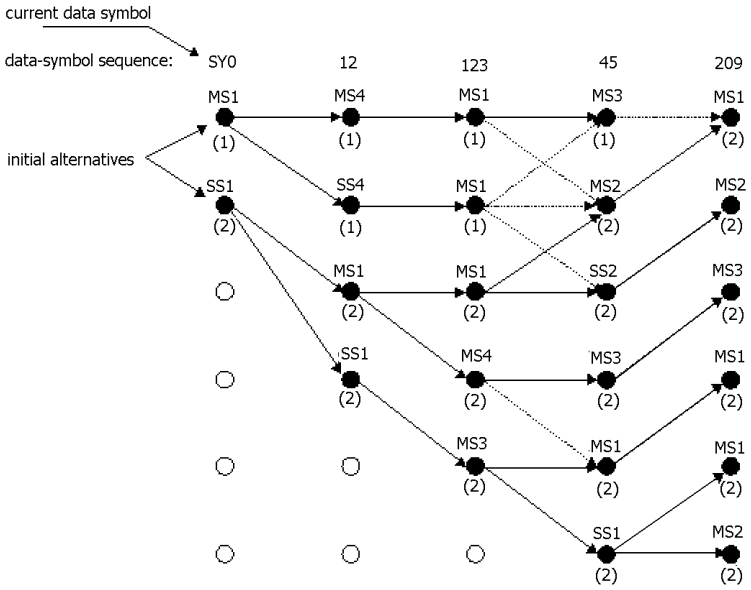


Fig. 13. Operation of window-look-ahead strategy.

- *Window-look-ahead.* The data-symbols are encoded one at time. If for a current data- symbol more than one Code Word exist trellis is constructed to make decision on which alternative Code Words should be output. Principle of operation of the window- look-ahead strategy is illustrated in Fig. 13. Nodes in the first level of the trellis are called initial alternatives. During trellis construction, to each node a label indicating its original initial alternative is associated. When all occupied nodes in a trellis level origin from the same initial alternative, trellis is terminated, and Code Word which correspond to that initial alternative is output. In the given example, the initial alternative (2) is selected, since all survivor paths in 5th path of the trellis start with this alternative. This is what we call the *normal trellis termination.* To ensure

that the trellis will terminate after finite number of constructed levels, a constant l, maximal trellis length, is introduced. When the trellis reaches l-th level, but decision is not yet made, the algorithm finds node with minimal cost function value, and selects the corresponding initial alternative.

6. Performance Evaluation of EFM++ Code

A crucial characteristic of the described look-ahead strategy is that we can easily make a trade-off between the algorithm complexity (in terms of time and space) and the quality of generated encoded sequence by altering parameters of the trellis, i.e. its height and length. In general, by adopting higher trellis, a better results are obtained. However, in the same time, the algorithms spends more time in evaluating larger number of alternative solutions.

The main results of our simulation study are:

- Window-look-ahead shows better performance with respect to block-look-ahead. More specifically, for the same trellis height, the block-look-ahead reaches performance of the window-look-ahead for extremely large trellis lengths, only. Also, block-look-ahead with trellis height n and length l requires n buffers of size l for holding all alternative paths. These buffers are not required for window-look-ahead. (More details about hardware implementation of the window-look-ahead algorithm will be given in Section 7.) Thus, in the sequel we exclude block-look-ahead from consideration.
- Metric $M2$ works slightly more efficiently with respect to metrics $M1$ and $M3$. However, the computational complexity of metric $M2$ is considerably larger than that of $M1$ and $M3$. Thus, for the further investigation we retain metrics $M1$ and $M3$. When the $|RDS|$ is controlled, the best results have been obtained by using metric M3 for threshold value $t = 4$. In the case of $|RDS2|$ control both metrics $M1$ and $M3$ shows approximately the same performances. Thus, in the look-ahead algorithms that control $|RDS|$ (i.e., minimizes the variations of the $|RDS|$) we use metric $M3$ ($t = 4$), while in the look-ahead algorithms that control $|RDS2|$ we use metric $M1$.
- By using the window-look-ahead algorithm for controlling $|RDS|$, the variations of the $|RDS|$ within the encoded sequence can be significantly reduced with respect to the existing algorithms. The level of this reduction depends on trellis height, n, and increases with the in-

creasing of n. The variance of the $|RDS|$, σ^2_{RDS}, as a function of n for window-look-ahead algorithm that uses metric $M3$ ($t = 4$) is shown in Fig. 14. For $n = 1$, window-look-ahead algorithm works exactly the same as incremental VHDS EFM+ encoder. As can be seen from Fig. 14 near-asymptotic performances are reached for $n > 4$. The spectral characteristic of the window-lookahead algorithm for $n = 12$. is given in Fig. 15. From this figure we can see that the more aggressive reduction of $|RDS|$ variations results in better suppression of the low-frequency components.

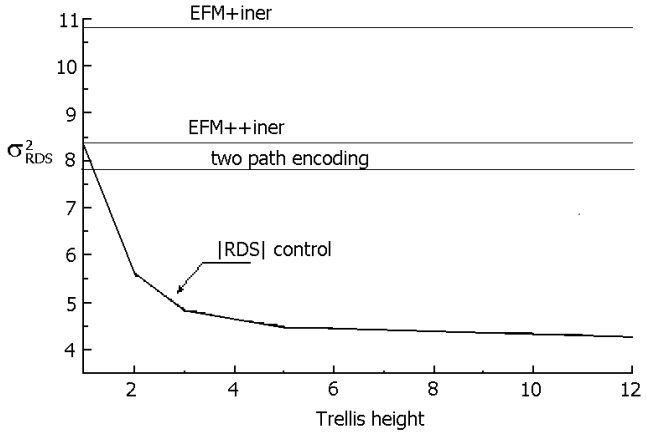


Fig. 14. $|RDS|$ variance in function of trellis height.

- By using the window-look-ahead algorithm that directly controls $|RDS2|$, the range of values that $|RDS2|$ assumes can be reduced. However, at the same time, the variations of $|RDS|$ are increased. Variances of both $|RDS2|$ and $|RDS|$ values within the encoded sequence, obtained by window-look-ahead algorithm that controls the $|RDS2|$ and uses metric $M1$, as a function of trellis height, n, are given in Fig. 15. As can be seen from this figure, for $n < 4$ the variations of $|RDS2|$ are still large and the variations of $|RDS|$ are even larger than that obtained by the incremental encoding algorithms. For $n > 6$ variances reach their near-asymptotic values. In these cases $|RDS|$ variance is comparable with that of window-look-ahead with trellis height 3 that directly controls $|RDS|$. From these results we can conclude that direct control of $|RDS2|$ requires more aggressive look-ahead, with trellis height of at least 6. Spectral characteristic of window-look-ahead with

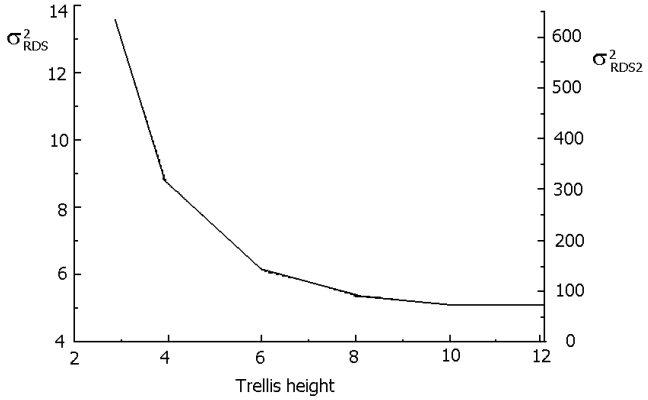


Fig. 15. $|RDS|$ and $|RDS2|$ variance in function of trellis height as obtained by window-look-ahead algorithm that controls $|RDS2|$.

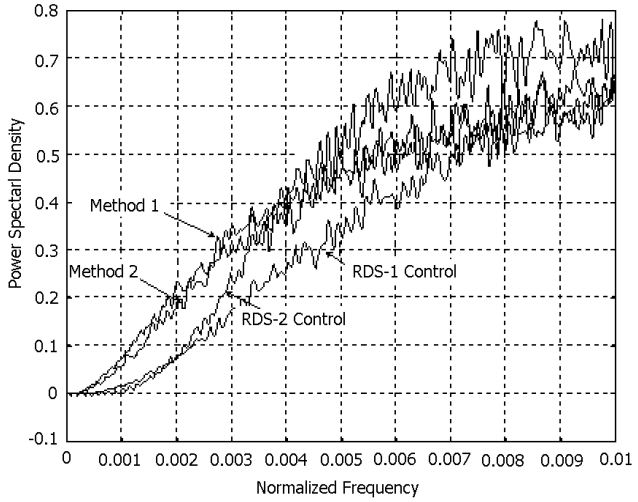


Fig. 16. Spectra of EFM+ and EFM++ codes.

$n = 12$ that controls $|RDS2|$ and use metric $M1$ is shown in Fig. 16. As can be seen, $|RDS2|$ control shows the best low-frequency components suppression characteristics in very-low-frequency range with respect to other analyzed approaches.

Fig. 16 is obtained by computer simulation of the EFM+ and EFM++ encoding process. "Random" input stream is first fed into the modulation

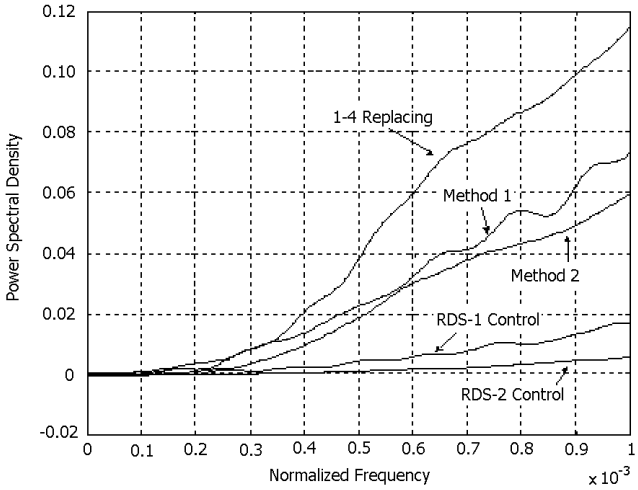


Fig. 17. Spectra of EFM+ and EFM++ codes at very low frequencies.

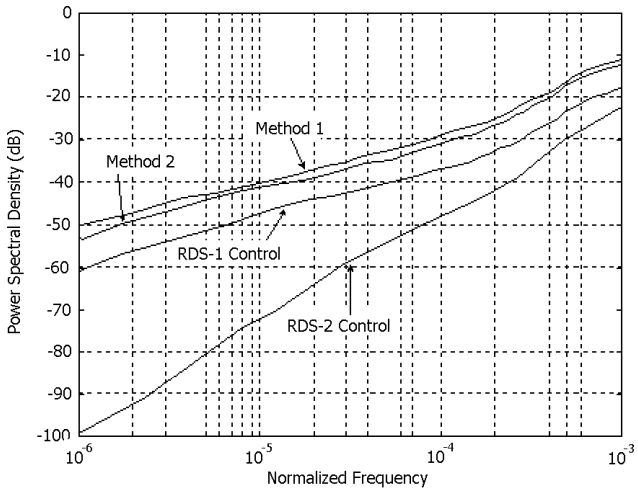


Fig. 18 Slopes of the EFM+ and EFM++ code spectra.

encoder, after which the sync frames are made by adding appropriate sync words according DVD specifications [10]. The plots represent the power spectral density of the channel signal obtained by modulating rectangular pulses by the EFM+ and EFM++ channel sequences. In Fig. 17 the closer look to the spectra at very low frequencies is given. It is easy to see that RDS2 control provides much better suppression of low frequency components (at

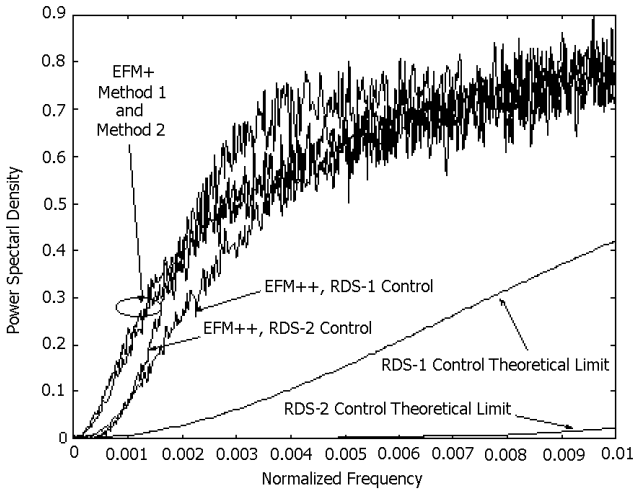


Fig. 19. Comparison of EFM++ spectra with the spectra of maxentropic (2,10) codes with RDS1 and RDS2 control.

least ten times better). As it has been noticed, the method where replacement of Conversion Tables 1 and 4 is allowed for all input bytes shows better performance then Method 1, but worse than Method 2. Figure 18 shows an log-log interpolation of the spectra from Fig. 16. As we expected the slope of the RDS2 controlled spectrum is bigger than slopes of the spectrum of the signals obtained by both original EFM+ methods as well as EFM++ RDS1 control method.

Finally the comparison between multimode coding approach and ACH approach is given in Fig 19. The EFM++ code shows better suppression of the low frequency content compared to the original EFM+ code, but it is still far from the theoretical limit (maxentropic codes).

7. Hardware Structure and Complexity of Window Look-Ahead EFM++ Encoder

The window-look-ahead algorithm constructs the trellis in a level-by-level fashion. When a new level is added to the trellis, only the information from the last trellis level are used. Also, in the window-look-ahead trellis there is no need to memorize the whole paths, but only the identifiers of the initial alternative from which paths start. Thus, it is sufficient to implement in hardware only two levels of trellis (let call these levels current and next). The next cycle of trellis operation starts by initiating of the current level

with the alternative which is selected in previous trellis cycle. By using the content of the current level, the next level is constructed. Then, the termination condition is checked and is not satisfied, the content of the next level is transferred into the current level, and the whole process is repeated. When termination condition is detected, by examining the next level the best initial alternative is determined and output.

The block structure of the widow-look-ahead encoder is given in Fig. 22. The number of Node blocks is equal to the trellis height. Each Node block contains two registers which correspond to one pair of nodes in the current and next trellis level. Each register contains the following information that are required to continue the trellis:

1) RSD value;
2) RDS2 value;
3) Current metric value;
4) The state of the encoder;
5) The length of the trailing phase of the corresponding Code Word;
6) Flag indicating the status of the register, occupied/non-occupied;
7) Identifier of initial alternative.

Control block select occupied current registers in a sequence. For each alternative Code Word of the current data-symbol, Arithmetic Unit calculates the new values of the RDS, RDS2 and metric. These values are written into an unoccupied next registers, if any. Otherwise, if there is no more unoccupied next-registers, the Comparator Block finds the next-register with largest metric-value, and its content is replaced with the new values. End-Detector block check the condition for trellis termination, while Alternative-block buffers the information about initial alternatives.

7.1 Hardware complexity

The hardware complexity of the presented EFM++ encoder is determined by the trellis height. To extend the trellis with the new row, one more Node-block have to be added to the structure in Fig. 20. Each Node block contains about 100-bits of register memory. In comparison with the existing incremental encoder, the EFM++ encoder does not use the output buffer memory for holding entire Sync frame. The amount of logic needed to implement such a buffer is estimated to be sufficient for realization of approximately 8-10 Node blocks. Thus, we can conclude that the hardware complexity of the existing solutions and the proposed EFM++ encoder, are

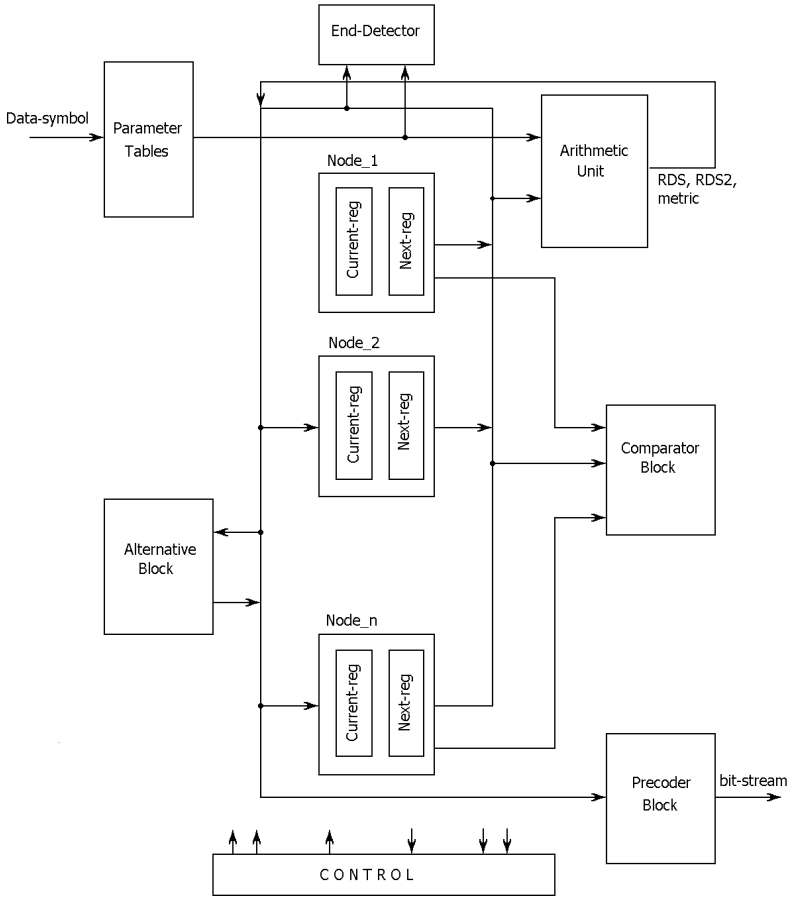


Fig. 20. Block structure of the window-look-ahead encoder.

comparable.

7.2 Time complexity

Having in mind the proposed hardware structure of the EFM++ encoder we identify the following three basic encoder operations:

- *Edge creation.* This operation includes: 1) reading Code-Word's parameters from the Parameter table; 2) calculation of RDS, $RDS2$ and metric values; and 3) writing calculated values into selected next-register.
- *Level initiation.* This operation is performed at the end of the con-

struction of each trellis level to transfer content of the next-registers into current-registers.

- *Trellis termination.* This operation is performed after detecting trellis termination condition to output selected initial alternative and get the new data-symbol.

We assume that each of encoder's basic operations takes one clock cycle to complete. Time complexity of the EFM++ encoder is determined by the number of clock cycles needed by the encoder to process one data-symbol. However, the number of clock cycles per data-symbol is not constant and depends on the size of the trellis that is constructed to select alternative Code Word of the current data-symbol.

We analyze the performance of the EFM++ encoder operating in two different operating modes: *Free-running mode*, and *Fixed-output-rate mode.* In the free-running mode, there does not exist any external limitation on the number of clock cycles that the EFM++ encoder can use to encode data sequence. In order to quantify the time performance of the EFM++ encoder operating in free-running mode we define the *average output rate* of the encoder as the average number of encoded data-symbols per one clock cycle (reciprocal to average number of clock cycles per data-symbol). In the fixed-output-rate mode of operation the EFM++ encoder must ensure the constant rate of the encoded bit-sequence. Under this condition, the number of clock cycles that encoder can spend to encode a data- symbol is limited. As a consequence, situations may occur when the encoder is forced to terminate the trellis construction before the normal-termination condition is encountered. This may degrade the quality of decisions that are made, and the level of this degradation should be examined.

7.3 Free running mode

The results of our simulations concerning the average output rate of the EFM++ encoder with different trellis heights are shown in Fig 21. Curve (1) in Fig. 21 corresponds to the case when there in no limit on trellis length, i.e. the trellis always terminates under normal-termination-condition. Under this condition EFM++ encoder achieves its asymptotic performances in terms of low-frequency components suppression which are presented in Section 4. As can be seen from Fig. 21 the average output rate of the EFM++ encoder rapidly decreases with the increasing of the trellis height. The average output rate can be improved by limiting the maximal trellis length, without sacrificing performance. The curve (2) in Fig. 21 corresponds to

the case when the trellis length (i.e., the number of constructed levels) is limited to 16. With respect to the non- limited-trellis-length case, the average output rate is doubled, on average. On the other hand, the observed degradation in performances, in terms of σ^2_{RDS}, is less then 5%.

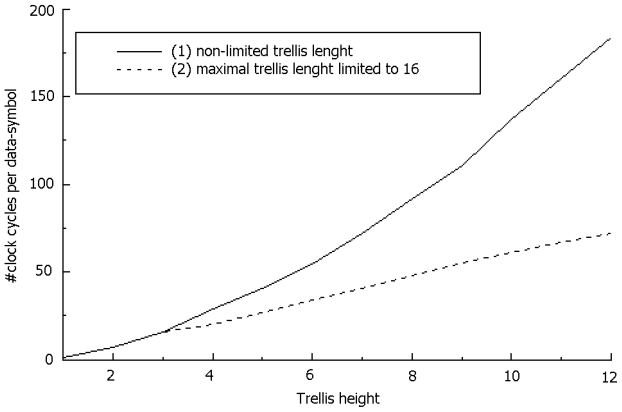


Fig. 21. Average number of clock cycles per one encoded data-symbol.

7.4 Fixed-output-rate mode

In order to examine the performance of the EFM++ encoder under "fixed output bit rate" assumption we have used the simulation model shown in Fig. 22. Data symbol buffer contains the current data-symbol and d_1 subsequent data symbols which are used for trellis construction. The size of this buffer, d_1, determines the maximal trellis length. Output buffer is used to decouple the encoder and precoder, i.e. to adapt the output rate of the encoder to the fixed output rate of the precoder. When the output buffer becomes empty (or near-empty) the encoder is signaled to terminate the trellis construction and immediately selects and outputs the current-data-symbol's alternative. When output buffer becomes full, the encoder is temporary stopped until the room for the new encoded data-symbol become available in the output buffer. The operation frequency of the encoder block is f_t, while the operation frequency of the precoder block (which determines the output bit rate) is $f_b = f_t/k$. Important fact about the system operation is that the precoder produces bits, while encoder processes data-symbols. This means that for $k = 1$ ($f_t = f_b$) encoder has at disposal 16 clock cycles, on average, to encode each data symbol. For $k = 2$, the average number of clock cycles per data symbol is increased to 32, and so on.

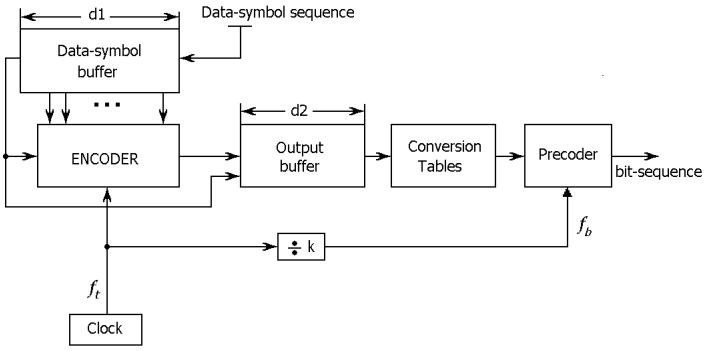


Fig. 22. Simulation model of EFM++ encoder operating in fixed-output-rate mode.

The variance of the $|RDS|$ achieved by EFM++ encoder working in fixed-output-rate mode, in function of trellis height for three different values of parameter k, is given in Fig. 23. In these simulations we have assumed $d_1 = d_2 = 16$. As can be seen form Fig. 23, for each k value there exists optimal trellis height for which the encoder yields minimal variance of the $|RDS|$. For example, for $k = 1$ the best performance is attained with trellis height $n = 3$, giving $\sigma^2_{RDS} = 5.0$ which represents degradation of about 15% with respect to asymptotic EFM++ encoder performance (free-running mode with trellis height $n = 12$). For $k = 2$, the optimal trellis height is $n = 6$, and the performance degradation is less then 8%. For $k = 3$, optimal trellis height is $n = 6$, again, and the performance degradation is reduced below 3%.

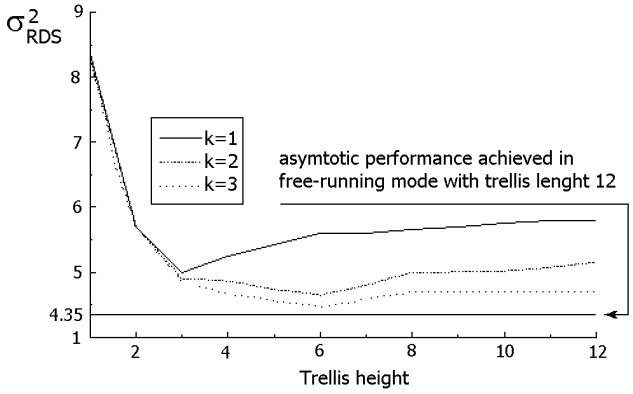


Fig. 23. EFM++ encoder performance in fixed-output-rate operation mode.

Finally, it should be noted that the performance of the EFM++ encoder operating in fixed-output-rate mode can be further improved by applying some of the advanced hardware optimization techniques, such as pipelining. Pipelining will enable partial overlapping of the execution of Edge-creation operations during one trellis level construction. Pipelining requires implementation of an extra hardware, but effectively reduces the total number of clock cycles needed to construct the trellis.

8. Summary

The modification of EFM+ code has been introduced having better suppression of low frequency content than original EFM+ code. Decoders of EFM+ and new EFM++ code are the same, and EFM++ encoder differs from EFM+ encoder in the way of selecting Main and Substitution table codewords. EFM++ encoder applies "1-4 exchange" rule to all input bytes when it is possible. EFM++ encoder also uses trellis search for choosing the best codeword sequence. Optimality criterion for choosing codewords is also different from EFM+ code. Since it has been recognized that RDS was not a metric, a metric based on RDS of second order has been accepted as a metric for EFM++. In fact the best EFM++ algorithm presented combines two metrics. The first one is based on maximal absolute value of the second order RDS, and its role is to suppress spectrum at very low frequencies and provide the cubic decrease of power spectral density. The second metrics is the absolute value of the first order RDS, and it is introduced in order to control the notch width in accordance to the relation between σ_{RDS2} and cut off frequency that holds for dc-free codes.

Trellis search algorithm exploited in EFM++ encoder can work with variety of lengths and heights. Simply speaking, as larger trellis size as better suppression performances. On the other hand trellis size is limited by the limited complexity requirements and the clock of the circuitry on the rest of the chip. The dependence of the first and second order RDS variances on the trellis size has been analyzed. It has been shown that the complexity comparable to the original EFM+ code complexity can be achieved by trellis height of 6 nodes and trellis length of 10 node blocks. Simulation has showed that the power spectrum of this code is more than 10 dB bellow the spectrum of EFM+ code for all frequencies in the servo system bandwidth.

To complete this research it is necessary to explore the servo system sensitivity to the noise produced by the data. According to the literature, low frequency content of the data signal is very significant impairment to the servo system, however we still do not have quantitative understanding of

this phenomenon. Next step in this research would be to simulate the servo system with noise of statistical behavior as low frequency content of the user data signal, and to verify servo behavior in the laboratory by recording data encoded by EFM+ and EFM++ coding rules.

REFERENCES

1. R. L. Adler, D. Coppersmith, and M. Hassner: *Algorithms for sliding block codes; an application of symbolic dynamics to information theory.* IEEE Trans. Inform. Theory, vol. IT-29, January 1983, pp. 5-22.
2. J. W. M. Bergmans: *Digital Baseband Transmission and Recording.* Boston: Kluwer Academic Publishers, 1996.
3. K. A. Schouhamer Immink: *EFMPlus: the coding format of the multimedia compact disc.* IEEE Trans. Consumer Electronics, vol. 41, no. 3, August 1995, pp. 491-497.
4. K. A. Schouhamer Immink and L. Patrovics: *Performance assessment of dc-free multimode codes.* IEEE Trans. Commun. Vol. 45, no. 3, March 1997, pp. 293-299.
5. K. A. Schouhamer Immink: *Spectral null codes.* IEEE Trans. Magnetics, vol. 26, no. 2, March 1990, pp. 130-1135.
6. K. A. Schouhamer Immink, and G. F. M. Beenker: *Binary transmission codes with higher order spectral zeros at zero frequency.* IEEE Trans. Inform. Theory, vol. IT-33, no. 3, May 1987, pp.452-454.
7. C. Monti, and G. Pierobon: *Codes with multiple spectral null at zero frequency.* IEEE Trans. Inform. Theory, vol. 35, no. 2, March 1989, pp. 463-472.
8. C. E. Shannon: *A mathematical theory of communication.* Bell syst. Tech. J., 1948, pp. 372-423 and 623-656.
9. B. Marcus, P. Siegel and J. K. Wolf: *Finite-state modulation codes for data storage.* IEEE J. Select. Areas Commun., vol. 10, no. 1, January 1992, pp. 5-37.
10. DVD Consortium: *DVD specifications for read-only disc, part 1; physical specivisation, version 0.9.* April 1996.
11. S. B. Wicker: *Error Control Systems for digital Communication and Storage.* Englewood Cliffs, Prentice Hall, 1996