(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
20 August 2009 (20.08.2009)

PCT

(10) International Publication Number
WO 2009/102304 A1

(51) International Patent Classification:
G06F 9/00 (2006.01)

(21) International Application Number:
PCT/US2008/002836

(22) International Filing Date:
3 March 2008 (03.03.2008)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
12/070,074 14 February 2008 (14.02.2008) US

(71) Applicant *(for all designated States except US)*: **HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P.** [US/US]; 11445 Compaq Center Drive West, Houston, TX 77070 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **ROTH, Ron, M.** [IL/IL]; 33 Ruth Street, 34404 Haifa (IL). **VONTOBEL, Pascal, O.** [CH/US]; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US).

(74) Agent: **LEHMANN, Eileen**; Hewlett-Packard Company, Intellectual Property Administration, P.O. Box 272400, M/S 35, Fort Collins, Colorado 80527-2400 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG,

[Continued on next page]

(54) Title: METHOD AND SYSTEM FOR DETECTION AND CORRECTION OF PHASED-BURST ERRORS, ERASURES, SYMBOL ERRORS, AND BIT ERRORS IN A RECEIVED SYMBOL STRING

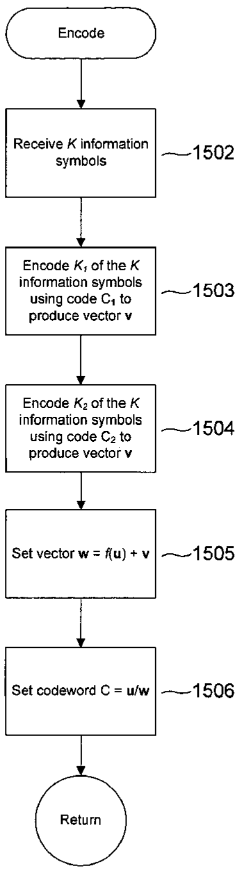

Encode
Receive *K* information symbols
1502
Encode K_1 of the K information symbols using code C_1 to produce vector v
1503
Encode K_2 of the K information symbols using code C_2 to produce vector v
1504
Set vector **w** = f(**u**) + **v**
1505
Set codeword C = **u**/**w**
1506
Return

Figure 15A

(57) Abstract: Embodiments of the present invention include ECC-based encoding-and-decoding schemes that are well suited for correcting phased bursts of errors or erasures as well as additional symbol errors and bit errors. Each encoding-and-decoding scheme that represents an embodiment of the present invention is constructed from two or more component error-correcting codes and a mapping function $f(\cdot)$. The composite error-correcting codes that represent embodiments of the present invention can correct longer phased bursts or a greater number of erasures in addition to single-bit errors and symbol errors, respectively, than either of the component codes alone, and are more efficient than previously developed ECC-based encoding-and-decoding schemes for correcting phased bursts of symbol errors and erasures combined with additional bit errors and symbol errors.

WO 2009/102304 A1

SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, NO, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

METHOD AND SYSTEM FOR DETECTION AND CORRECTION OF PHASED-BURST ERRORS, ERASURES, SYMBOL ERRORS, AND BIT ERRORS IN A RECEIVED SYMBOL STRING

TECHNICAL FIELD

The present invention is related to correction of errors or erasures that occur in symbol strings passed through an error-and-erasure-introducing channel, including electronic transmission of the symbol string or storage of the symbol strings in, and retrieval of the symbol strings from, an electronic memory.

BACKGROUND OF THE INVENTION

The field of error-correcting codes ("ECCs") has been well studied and researched for over 50 years. Many different types of encoding-and-decoding schemes based on error-correcting codes have been developed for application to many different problem domains. ECC-based encoding-and-decoding schemes generally involve introduction of redundant information into an encoded information stream to allow various types of errors subsequently introduced in the information stream to be detected and corrected. As with most computational techniques, there are a variety of advantages, disadvantages, efficiencies, and inefficiencies associated with any particular encoding-and-decoding scheme applied to any particular problem domain. For example, as the amount of redundant information added to an information stream increases, the quantities and types of errors that can be detected and corrected within the information stream generally increases, but the information, or space, efficiency of transmission of the information stream decreases due to the increasing overhead of the redundant information. Space inefficiencies can also result from the need to create and maintain large amounts of data needed for encoding or decoding, such as decoding tables, discussed below. As another example, a symbol efficient code may involve complex computation, and may therefore be computationally, or time, inefficient. The overall efficiency of a code is related to the sum of the space and time efficiencies of the code, but space efficiency is often obtained at the expense of time efficiency, and vice versa. Certain types of ECC-based encoding-and-decoding schemes are better suited to detecting and correcting certain types of errors, and may be less well suited for detecting and correcting other types of errors. As new problem domains are recognized, or as new problem domains emerge as a result of the development of new types of technologies, continued development of new ECCs and ECC-based encoding-and-decoding schemes well suited for the newly recognized problem domains or newly developed technologies are needed in order to provide for efficient and accurate error detection and correction.

SUMMARY OF THE INVENTION

Embodiments of the present invention include ECC-based encoding-and-decoding schemes that are well suited for correcting phased bursts of errors or erasures as well as additional symbol errors and bit errors. Each encoding-and-decoding scheme that represents an embodiment of the present invention is constructed from two or more component error-correcting codes and a mapping function $f(\bullet)$. The composite error-correcting codes that represent embodiments of the present invention can correct longer phased bursts or a greater number of erasures in addition to single-bit errors and symbol errors, respectively, than either of the component codes alone, and are more efficient than previously developed ECC-based encoding-and-decoding schemes for correcting phased bursts of symbol errors and erasures combined with additional bit errors and symbol errors.

According to one embodiment of the present invention, encoding of information into a composite-code codeword is carried out by receiving K_1 information symbols and encoding the K_1 information symbols by a first component code C_1 encoder to produce a C_1 codeword **u** of length N_1 symbols. Then, K_2 information symbols are encoded by a second component code C_2 encoder to produce a codeword **v** of length N_2. A vector **w** of length N_2 symbols is obtained by adding a non-identity mapping of **u**, $f(\mathbf{u})$, to **v**. Finally, a composite-code-C codeword is generated by concatenating **u** and **w** together.

According to one embodiment of the present invention, decoding of a composite-code codeword is carried out by decoding component-code codewords. A component-code-C_1 codeword **u** of length N_1 containing K_1 information symbols and

a modified component-code-C_2 codeword of length N_2, $\mathbf{w} = \mathbf{v} + f(\mathbf{u})$, generated, during encoding, from a component-code-C_2 codeword $\mathbf{v}$ containing K_2 information symbols, where $K = K_1 + K_2$ and $N = N_1 + N_2$, and a non-identity mapping function, $f(\bullet)$, are extracted from a composite-code-C codeword. An estimated component-code-C_2 codeword $\hat{\mathbf{v}}$ and an estimated error word $\hat{\mathbf{e}}$ are then generated from the modified component-code-C_2 codeword by applying a C_2 decoder to the modified component-code-C_2 codeword. Which of a number of types of expected errors that may occur subsequent to encoding of the composite-code-C codeword is determined from the error word $\hat{\mathbf{e}}$. When more than a first threshold number of erasures and erasures have occurred, but less than a second threshold number of errors have occurred, the determined errors are assigned to either the component-code-C_1 codeword or to the modified component-code-C_2 codeword, and when assigned to the component-code-C_1 codeword, are corrected. Other error and erasure occurrences are marked. An estimated component-code-C_1 codeword $\hat{\mathbf{u}}$ is obtained by applying a C_1 decoder to the estimated component-code-C_1 codeword $\hat{\mathbf{u}}$. Finally, K_1 information symbols are extracted from the estimated component-code-C_1 codeword $\hat{\mathbf{u}}$ and K_2 information symbols are extracted from the estimated component-code-C_2 codeword $\hat{\mathbf{v}}$ to produce K extracted information symbols.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a basic problem to which ECC-based encoding-and-decoding schemes are applied.

Figure 2 illustrates various different views of a digitally encoded information stream.

Figure 3A illustrates the vector space V of all possible codewords produced by a systematic linear block code that encodes information into codewords of length n.

Figure 3B shows an exemplary code, or vector subspace, of the vector space V shown in Figure 3A.

Figure 4 shows the distance between any two codewords $\mathbf{v}$ and $\mathbf{w}$, $D(\mathbf{v},\mathbf{w})$.

Figure 5 illustrates encoding and transmission of a vector **u** of k information bits by a systematic linear block code.

Figure 6 illustrates encoding of the information-bit vector **u** to produce codeword **v**, as discussed with reference to Figure 5.

Figures 7A-B show an exemplary systematic generator matrix **G** and an exemplary systematic parity-check matrix **H** for a systematic linear block code.

Figure 8 shows a property of the transpose of the parity-check matrix, $\mathbf{H}^T$.

Figure 9 illustrates a portion of the decoding process for a systematic linear block code.

Figure 10 illustrates a decoding table that can be constructed for any systematic linear block code over GF(2).

Figure 11 shows a portion of the table of elements for GF(2^8).

Figure 12 illustrates the basic characteristics of a composite code that represents one embodiment of the present invention.

Figure 13 illustrates the characteristics of the symbol-to-symbol mapping function $f(\bullet)$ used in embodiments of the present invention.

Figure 14 shows two different implementations of the symbol-to-symbol mapping function $f(\bullet)$.

Figure 15A provides a high-level control-flow diagram for encoding of information bits into a composite-code codeword according to one embodiment of the present invention.

Figure 15B illustrates construction of the composite code C[72,66,5] that represents one embodiment of the present invention.

Figure 16 illustrates a method of encoding a composite-code codeword that can be carried out repeatedly on an input stream of information symbols to produce an output stream of composite-code codewords.

Figure 17A illustrates the notion of a sub-block within a codeword of the composite code that represents one embodiment of the present invention.

Figure 17B illustrates the various different types of errors that the composite code that represents one embodiment of the present invention is designed to detect and correct.

Figure 18 provides a high-level control-flow diagram for decoding of a composite code that represents one embodiment of the present invention.

Figures 19-20 provide a control-flow diagram that illustrates one embodiment of the decoding process for composite codes that represent embodiments of the present invention.

Figure 21 illustrates the information received for each step of a decoding method for the composite code that represents one embodiment of the present invention.

Figure 22 shows a block diagram of a physical memory device in which embodiments of the present invention may be employed.

Figure 23 illustrates mapping between codeword symbols and DRAM units in a bank of DRAM units that together comprise the electronic data storage component of the physical memory device illustrated in Figure 22.

DETAILED DESCRIPTION OF THE INVENTION

The present invention is directed to error-correcting codes ("ECCs") and ECC-based encoding-and-decoding schemes well suited for detecting and correcting phased bursts of symbol errors and/or erasures and additional single-bit errors and symbol errors, respectively, in a symbol string passed through an erasure-and-error-introducing channel. The present invention is discussed, below, in three subsections. In a first subsection, an overview of one family of error-correcting codes is provided. These error-correcting codes are examples of component ECCs that may be used to construct the composite ECCs that represent embodiments of the present invention, although many additional types of ECCs may be used as components for the composite ECCs. In a following subsection, a brief summary of groups and fields is provided. Finally, in a third subsection, composite codes and composite-code-based encoding-and-decoding schemes to which the present invention is directed are described, with detailed descriptions of encoding and decoding methods for one disclosed composite code.

Systematic Linear Block Codes

Figure 1 illustrates a basic problem to which ECC-based encoding-and-decoding schemes are applied. In Figure 1, a binary-encoded information stream 102 is input to a memory, communications system, or other electronic device, subsystem, or system 104 that exhibits characteristics of an error-introducing channel 105. Subsequently, the digitally encoded information stream is extracted 106 from the memory, communications system, or other electronic device, subsystem, or system 104. It is desirable, and generally necessary, that the extracted information stream 106 be identical to the originally input information stream 102. In order to achieve error-free recovery of information input to the memory, communications system, or other electronic device, subsystem, or system 104, an encoder 108 can be used to introduce redundant information into the information stream and decoder 110 can be used to employ the redundant information to detect and correct any errors introduced by the error-introducing-channel characteristics of the memory, communications system, or other electronic device, subsystem, or system 104. In Figure 1, the binary-encoded information stream is represented in a left-to-right direction 102 when input and in a right-to-left direction when extracted 106. However, in general discussions of ECC-based encoding-and-decoding schemes, an encoded information stream is generally represented in left-to-right order, regardless of whether the information stream represents an input information stream or a received information stream, with the understanding that encoded information is generally transmitted sequentially, bit-by-bit, or byte-by-byte, and then reassembled on reception.

Error-introducing-channel characteristics may be exhibited by an electronic communications medium, such as a fiber-optic cable with a transmitting port on one end and a receiving port at the other end, an Ethernet link with Ethernet ports and controllers included in computing devices that are connected by the Ethernet link, and in other familiar electronic-communications media. Alternatively, error-introducing-channel characteristics may be exhibited by an information-storage device or component, including different types of electric memories, a mass-storage device, or a physical data-storage medium, such as a DVD or CD. In the case of a communications medium, an information stream initially input to a transmission port may be subsequently received as a corrupted information stream by a receiving port, with errors introduced into the information stream by port-processing components, noise in the transmission medium, and other such error-introducing phenomena. In the case of a storage medium, an initial information stream input to the storage medium may be subsequently retrieved from the storage medium in a corrupted form, with errors introduced into the information stream by storage-component controllers and other processing components, by noise and transmission media, and by electronic, magnetic, and/or optical instabilities in the storage media.

There are various types of errors that may corrupt an encoded information stream. Random bit or symbol errors may result in alteration of the bit or symbol values of certain bits and symbols in the information stream, with the bits or symbols in the information stream having a known or estimable probability of corruption. Burst errors result in corruption in runs of adjacent bits and/or symbols. Many different types of systematic errors, in addition to burst errors, may also occur.

Figure 2 illustrates various different views of a digitally encoded information stream. A digitally encoded information stream can be viewed as an ordered sequence of bit values, or, in other words, the information stream comprises a long, linear array of bit values. Alternatively, the same encoded information stream can be viewed as the ordered sequence of symbols, each symbol comprising a fixed number of bit values. For example, in Figure 2, the binary encoded information stream 202 can be alternately viewed as an ordered sequence of four-bit symbols 204. The value "9" shown in Figure 2 for the second symbol 206 in the ordered sequence of symbols corresponds to the ordered set of bit values 208-211 in the bit-value representation of the encoded information stream 202. In yet another view, the encoded information stream may be viewed as an ordered sequence of blocks 212, each block including an ordered sequence of a fixed number of symbols. Finally, an information stream may be encoded, by a systematic linear block code, to include redundant information to allow for errors to be subsequently detected and corrected. The encoded information stream 214 comprises an ordered sequence of blocks, or codewords, each codeword corresponding to a block in the information stream. For example, the codeword 216 of the encoded information stream corresponds to the block of symbols 218 in the block-view of the information stream 212. Each codeword includes an additional symbol 220-222, represented in Figure 2 by the characters *R′*, *R″*, and *R‴*. This extra symbol represents the redundant information included in the information stream by one type of systematic linear block code. In alternative types of linear block codes, each codeword may comprise a first, selected number of information symbols as well as a second selected number of additional symbols representing added redundant information, with the ratio of redundant information symbols to information symbols generally correlated with the number of errors or erasures that may be detected and the number of errors or erasures that may be corrected.

One commonly used type of ECC is a systematic linear block code over a finite field GF(q), where q represents the number of symbols in a field over which the code is defined. When q is a power of 2, 2^m, the symbols of the field are represented as m-tuples. When m is equal to 8, symbols are conveniently represented as bytes. The notation "GF(2)" stands for the binary Galois field with two elements, or symbols, "0" and "1." Given a fixed number of bits in each encoded block, or codeword, produced by a systematic linear block code over GF(2), all of the possible codewords together comprise a vector space. A vector space has certain algebraic properties, including being commutative under addition, closure under scalar multiplication, and is distributive and associative with respect to vector addition and scalar multiplication of vectors. Figure 3A illustrates the vector space V of all possible bit vectors of length n over GF(2). A particular systematic linear block code C that produces codewords of length n is a k-dimensional vector subspace of V, the vector subspace having all of the properties of a vector space. Figure 3B shows an exemplary code, or vector subspace, of the vector space V shown in Figure 3A. Each k-dimensional vector in the vector subspace represents k bits of information from an information stream. The k bits of information are supplemented, by the systematic linear block code, with $r = n\text{-}k$ additional bits to produce a codeword. There is one particular pattern of r additional bits, or parity bits, for each different possible k-dimensional vector of information bits. Thus, a systematic linear block code comprises 2^k different n-bit vectors of the vector space V that constitute a vector subspace. For a systematic linear block code over GF(q), rather than bits, each vector containing n symbols, of which k symbols are information symbols and n - k symbols are redundant information used for detecting and correcting errors. The vector subspace comprising the codewords of the systematic linear block code over GF(q) contains q^k vectors.

An important characteristic of an ECC is the minimal distance d between any two codewords of the code. Figure 4 shows the distance between any two codewords **v** and **w**, D(**v**,**w**), of an ECC over GF(2). The vector **v** is a 12-bit codeword 402 and **w** is a second 12-bit codeword 404. Subtracting **w** from **v** by modulo 2 subtraction, equivalent to a bit-by-bit XOR operation, produces the difference between **v** and **w**, **v** - **w**, 406. In the case of an ECC over GF(2), the number of bits with bit value "1" in the vector **v** - **w** 406 is equal to the distance between **v** and **w**, D(**v**,**w**). In the general case of an ECC over GF(q), the number of non-zero positions in the difference vector **v** - **w** is the distance between the two codewords **v** and **w**. The weight of any particular codeword **v**, W(**v**), is the number of non-zero positions in the codeword. Thus, D(**v**,**w**) = W(**v** - **w**) = 3 in the example shown in Figure 4.

Figure 5 illustrates encoding and transmission of a vector **u** of k information symbols by a q-ary systematic linear block code. The k information symbols are considered to be a k-dimensional vector **u** 502. A systematic linear block code encodes the k information symbols, represented by the vector **u**, as a vector **v** of length $k + r = n$ 504. A systematic linear block code places r check symbols, or parity symbols, together in a subvector of vector **v** having length r, generally either at the beginning or the end of vector **v**. In the example shown in Figure 5, and continued in subsequent figures, the parity symbols $p_0, p_1, \ldots, p_{r-1}$ 506 are shown in the initial part of vector **v**, and the k information symbols 508 follow. The codeword **v** is then transmitted through a communications medium or stored to, and retrieved from, a storage medium to produce the corresponding received word **x** 510. When no errors occur in transmission or storage, **x** = **v**. However, when random transmission or storage errors occur, **x** $\neq$ **v**. In many cases, the recipient of the vector **x** cannot compare **x** with the initial, corresponding vector **v** in order to ascertain whether errors have or have not occurred. Therefore, the recipient of vector **x** assumes that each symbol, or bit, in **x** may have been corrupted with some probability of corruption. Therefore, the symbols in **x** are primed, in Figure 5, to indicate that the symbols may have been corrupted with a known or estimable probability of corruption. Thus, symbol p_0 512 in codeword **v** corresponds to symbol p_0' 514 in the received word **x**.

Figure 6 illustrates encoding of the information-bit vector **u** to produce codeword **v**, as discussed with reference to Figure 5. A k x n matrix **G** 602 can be found, for a given systematic linear block code, to generate a unique codeword **v** corresponding to each possible information-symbol vector **u**. As shown in Figure 6, **u** 604 is multiplied by **G** 606 to produce the codeword **v** 608 corresponding to **u**. The matrix **G** is called a generator matrix for the systematic linear block code. The matrix **G** consists of k linearly independent codewords of the systematic linear block code C. Thus, codewords for systematic linear block codes are easily and mechanically generated from corresponding blocks of information symbols by matrix multiplication. In fact, each matrix **G** defines a systematic linear block code.

Figures 7A-B show an exemplary systematic generator matrix **G** and an exemplary systematic parity-check matrix **H** for a systematic linear block code. The generator matrix **G** 702, as shown in Figure 7A, can be spatially partitioned into a parity-bit matrix **P** 704 of dimension k x r, and a k x k identity matrix $\mathbf{I}_k$ 706. The parity-bit matrix **P**, during matrix multiplication of **u** x **G**, generates the r parity symbols of **v**, and the identity matrix $\mathbf{I}_k$ 706 generates the k information symbols of **u** within the codeword **v**.

For each systematic linear block code, there is a parity-check matrix **H** corresponding to the generator matrix **G**. Figure 7B illustrates the form of the parity-check matrix **H**. As can be seen in Figure 7B, the parity-check matrix is an r x n matrix that can be spatially partitioned into an r x r identity matrix $-\mathbf{I}_r$ 710 and the transpose of the parity-check matrix $\mathbf{P}^T$ 712. Any particular systematic linear block code is completely specified either by a generator matrix **G** or by the parity-check matrix **H** corresponding to the generator matrix **G**. The parity-check matrix **H** is itself a generator for a linear code, with each codeword including *r* information symbols. The linear code generated by the parity-check matrix is the dual code of the systematic linear block code C generated by the generator matrix **G**. Figure 8 shows a property of the transpose of the parity-check matrix, $\mathbf{H}^T$. As shown in Figure 8, the transpose of the parity-check matrix, $\mathbf{H}^T$ 802, when used to multiply a codeword **v** of the systematic linear block code C, always generates the all-zero vector, $\mathbf{0}$, of dimension *r* 806. In other words, for each codeword **v** of systematic linear block code C:

$$\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$$

Figure 9 illustrates a portion of the decoding process for a systematic linear block code. As discussed above, the received word **x** 902 may contain errors with respect to the corresponding, initially transmitted or stored codeword **v** 904. As discussed above, subtracting **v** from **x**, in the case that both **v** and **x** are known, produces a resultant vector 906 in which a non-additive-identity symbol ("1" in the case of GF(2)) appears at every position at which vectors **x** and **v** differ. Thus, **x** - **v** = **e**, where **e** is referred to as the "error vector," essentially a map of occurred errors Of course, in general, only **x** is known. Thus, **x** equals **v** + **e**, where both **v** and **e** are generally unknown. Multiplication of the received word **x** 908 by the transpose of the parity-check matrix, $\mathbf{H}^T$, 910, produces an *r*-dimension vector **s** 912 referred to as the "syndrome" of **x**. The syndrome of **x** is equal to $\mathbf{e} \cdot \mathbf{H}^T$. Thus:

$$\mathbf{s} = \mathbf{e} \cdot \mathbf{H}^T = \mathbf{x}\mathbf{H}^T$$

Figure 10 illustrates a decoding table that can be constructed for any systematic linear block code over GF(*q*). As shown in Figure 10, a q^r x q^k table, called the "standard array," 1002 can be constructed for any systematic linear block code. The first row 1004 of the standard array is an ordered sequence of the codewords $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{q^k-1}$. The codeword $\mathbf{v}_0$ is the all-zero-symbol code vector (0, 0,…, 0). Each column *i* of the standard array can be considered to contain all possible received words $\mathbf{x}_j$ corresponding to the codeword $\mathbf{v}_i$ in the first element of the column. In other words, the set of all possible received words **V** has q^n elements, and is partitioned into q^k partitions, each partition corresponding to a codeword of the systematic linear block code C, with any received word **x** considered to correspond to the codeword associated with the partition of all possible codewords to which **x** belongs. For example, all of the elements of the first column 1006 of the standard array $\{e_1, e_2, \ldots, e_{2^r-1}\}$ correspond to all possible error vectors that, when added to the all-zero codeword $\mathbf{v}_0$, produce received words that are decoded to the all-zero codeword $\mathbf{v}_0$.

As discussed with reference to Figure 9, multiplication of a received word **x** by the transpose of the parity-check matrix $\mathbf{H}^T$ produces a syndrome vector **s** equal to $\mathbf{e}\cdot\mathbf{H}^T$. The syndromes computed for all of the elements in each row of the standard array are therefore identical, depending only on **e** and $\mathbf{H}^T$. Therefore, information contained in the standard array, for decoding purposes, can be compressed into a decoding table 1008 that shows the association between each recognized error pattern $\mathbf{e}_i$ and the syndrome corresponding to that error pattern $\mathbf{e}_i\mathbf{H}^T$. Decoding of codewords of a systematic linear block code is, like encoding, carried out by a relatively conceptually simple process:

$$\left.\begin{array}{l}\mathbf{s}_\mathbf{x} = \mathbf{x}\mathbf{H}^T \\ \hat{\mathbf{e}} = \mathbf{e} \text{ in decoding table associated with } \mathbf{s}_\mathbf{x} \\ \hat{\mathbf{v}} = \mathbf{x} + \hat{\mathbf{e}}\end{array}\right\}$$

However, although conceptually simple, designing codes that can be efficiently decoded is a decidedly non-trivial task. Decoding tables, for example, are impractical for codes with medium and large q, r and/or n parameters, since the size of the decoding table 1008 is proportional to $2(q^r)(n)$. Thus, great effort is generally undertaken to design codes with properties that allow for decoding algorithms that are both space and time efficient.

As can be seen in the standard array shown in Figure 10, by increasing the number of parity symbols included in each codeword, a larger number of different error patterns may be recognized. However, as the ratio $\frac{r}{n}$ increases, the space efficiency of encoding decreases. In general, the error patterns recognized by a systematic linear code are chosen to be the most probable error patterns. For random-bit errors, the error vectors with least weight are generally the most probable error patterns. For other types of errors, different sets of error patterns may be more probable.

While systematic linear block codes over GF(2) have been discussed, above, systematic linear block codes, including Reed-Solomon codes, can be analogously constructed over any field GF(q). Often, it is convenient to construct systematic linear block codes over extension fields of GF(2), generally specified as GF(2^m), where m is an integer greater than 1.

Groups and Fields

In this subsection, an overview of groups and fields is provided. A group is a set of elements, over which a binary operation * is defined. The group is closed under the binary operation *. In other words, for any two elements of the group a_1 and a_2, $a_1 * a_2 = a_i$, where a_i is also an element of the group. The binary operation * is associative, so that:

$$(a_1 * a_2) * a_3 = a_1 * (a_2 * a_3)$$

A group has a unique identity element e such that, for every element a_i in the group, there is an inverse element a_i^{-1}:

$$a_i * a_i^{-1} = a_i^{-1} * a_i = e$$

A group is commutative, or Abelian, when, for any pair of elements a_i and a_j:

$$a_i * a_j = a_j * a_i$$

A field is a commutative group with respect to two different binary operations. One operation may be denoted "+," with the identity element for the operation +, e_+, equal to 0, and the other operation may be denoted "*," with e_*, the identity element for the operation *, equal to 1. Furthermore, the operation * is distributive:

$$a * (b + c) = a * b + a * c$$

GF(2) is a binary field, with the + operation equivalent to modulo-2 addition, or the binary XOR operation, and the * operation equivalent to modulo-2 multiplication, or the Boolean AND operation. GF(q) is a field over the elements

$\{0,1, \ldots, q\text{-}1\}$ where q is a prime number. The field GF(q^m) is an extension field of GF(q), where the elements are defined as polynomials with coefficients in GF(q). GF(2^m) is an extension field of GF(2) where elements are polynomials with coefficients in GF(2).

A polynomial $p(\xi)$ of degree m is primitive when the smallest positive integer n for which $p(\xi)$ divides $\xi^n + 1$ is equal to $n = 2^m$-1. The extension field GF(2^m) can be represented as a field F of polynomial elements, as follows:

$$\mathrm{GF}\left(2^m\right)=\mathrm{F}=\left\{0,1,\alpha,\alpha^2,\ldots,\alpha^{2^m-2}\right\}$$

where α is a third symbol, in addition to 1 and 0;

$p(\alpha)=0$;

$\alpha^{2^m-1}=1$; and

$\alpha^{2^m-1}+1=0$.

For the operation * in F:

$$e_* = 1$$
$$\left(\alpha^i\right)^{-1} = \alpha^{2^m-i-1}$$

For the operation + in F:

$$e_+ = 0$$
$$-\alpha^i = \alpha^i$$

In addition to representing the elements of F as powers of α, each element in F can also be represented as a polynomial with binary coefficients:

$$\alpha^i = a_{i,0} + a_{i,1}\alpha + a_{i,2}\alpha^2 + \cdots + a_{i,m-1}\alpha^{m-1}$$

Addition of elements of F is easily carried out by polynomial addition, and multiplication of elements of F is easily carried out by adding exponents of the elements expressed as powers of α.

For an extension field, such as GF(2^8), a table can be constructed for each element in GF(2^8), each entry of which shows the powers representation of the element, the polynomial representation of the element, and a tuple of binary values comprising the coefficients of the polynomial representation of the element. Figure 11 shows a portion of the table of elements for GF(2^8). The first column 1102 of the table 1100 shows the powers representation of the elements of GF(2^8), the middle column 1103 provides the polynomial representation for the elements, and the final column 1104 shows the 8-bit binary-coefficient-tuple representation of each element. Additional tables can be constructed for multiplication and addition operations. Thus, the field GF(2^8) can be expressed as a set of 256 elements, each element an 8-bit tuple, with multiplication, addition, and subtraction operations specified by tables based on operations performed on the underlying polynomials. It is important to note that the multiplication, subtraction, and addition operations for the 8-bit element of GF(2^8) are not equivalent to familiar binary arithmetic operations supported by electronic computers. As one example, in binary arithmetic:

$$00100000 + 10111000 = 11011000$$

but in GF(2^8) addition:

$$\begin{aligned} \alpha^2 &= 00100000 = \alpha^2 \\ \alpha^8 &= 10111000 = 1 + \alpha^2 + \alpha^3 + \alpha^4 \\ \alpha^2 + \alpha^8 &= \alpha^{193} = 1 + \alpha^3 + \alpha^4 = 10011000 \end{aligned}$$

The example of GF(2^8) is provided, because, in one disclosed embodiment of the present invention, a composite code over GF(2^8) is constructed from two component codes over GF(2^8). Each symbol in a codeword can be viewed as an 8-bit tuple that represents an element of GF(2^8). Note that there are 256 elements in GF(2^8). Thus, every possible 8-bit tuple is an element of GF(2^8). In general, for encoding and decoding purposes, information bytes are considered to be symbols in GF(2^8), but prior to encoding and following decoding, the information bytes are viewed as standard binary-encoded bytes. In the following discussion of the present invention, example codes over GF(2^8) are discussed, but, the methods of the present invention can be applied to creation of composite codes over any field GF(q). It turns out, for computing efficiency, composite codes over GF(2^m) are desirable, for efficiency in symbol representation and efficiency in computational operations.

Embodiments of the Present Invention

The present invention is directed to a family of composite error-correcting codes that are constructed using at least two component codes and a function $f(\bullet)$, described below, that maps symbols of a field over which the composite code is defined to other symbols of the field. In the following discussion, one particular composite code from the family of composite codes that represent embodiments of the present invention is discussed. The discussed composite code is a code over 8-bit symbols of the extension field GF(2^8). However, composite codes can be analogously constructed for symbols of an arbitrary field GF(q) or GF(q^m), using component codes constructed for symbols of the arbitrary field.

Figure 12 illustrates the basic characteristics of a composite code that represents one embodiment of the present invention. The composite code is constructed over GF(2^8) and produces codewords of length $N = 72$, where N is the length, in 8-bit symbols. An exemplary codeword 1202 is shown in Figure 12. The codeword contains $K = 66$ information symbols and $R = 6$ parity-check symbols. The minimum distance between codewords is $D = 5$ symbols. The composite code can also be viewed as a code over GF(2). An exemplary codeword of the composite code over GF(2) 1210 is also shown in Figure 12. When viewed as a code over GF(2), each codeword has $n = 576$ bits of which $k = 528$ bits 1212 are information bits and $r = 48$ bits 1214 are parity-check bits. The minimum distance between codewords is in the range $5 \leq d \leq 40$, depending on the nature of the particular component codes used to construct the code. A linear block code having the characteristics $N = 72$, $K = 66$, and $D = 5$ would be expected to be able to detect and correct $(D - 1) / 2 = 2$ symbol errors or 4 symbol erasures. However, the composite code that represents an embodiment of the present invention can correct a larger number of symbol errors when they occur in bursts, a larger number of erasures, and a number of symbol errors and bit errors in addition to error bursts and erasures.

Coding and decoding methods for the composite code that represents one embodiment of the present invention relies on a symbol-to-symbol mapping function $f(\bullet)$. Figure 13 illustrates the characteristics of the symbol-to-symbol mapping function $f(\bullet)$ used in embodiments of the present invention. In Figure 13, a sequence of 256 8-bit symbols representing the 256 elements of GF(2^8) 1302 is partially displayed. The second through ninth symbols of GF(2^8), referred to as the set "M," 1304 include those symbols with 8-bit-tuple representations that each includes only a single bit with bit value "1." These 8-bit vectors in the set M correspond to GF(2^8) elements $\{1, \alpha^1, \alpha^2, \ldots, \alpha^7\}$ in the representation of GF(2^8) shown in Figure 11. Any function $f(\bullet)$ that maps symbols of GF(2^8) to other symbols of GF(2^8) can be employed for coding and decoding of the composite code that represents an embodiment of the present invention, providing that the function $f(\bullet)$ is linear, has a strict inverse function $f(\bullet)^{-1}$, and maps any symbol of the set M to a symbol of GF(2^8) that is not in the set M:

$$f(u \in M) \rightarrow u' \notin M$$
$$\exists f^{-1}(u) : f^{-1}(f(u)) = u$$

Figure 14 shows two different implementations of the symbol-to-symbol mapping function $f(\bullet)$. In one implementation, $f(u)$ may be implemented as multiplication of a bit-vector representation of symbol u by an m x m matrix 1402, where m is the m of the binary extension field GF(2^m) over which the code is constructed, in the current case, 8. In an alternative embodiment, a lookup table 1404 can be prepared to provide $f(u)$ values for each possible symbol u. In the case of GF(2^8) symbols, the symbol represented by the bit-vector **u** can be used as a numeric byte value to index the lookup table.

In alternative embodiments, the mapping function $f(\bullet)$ may be a different function. In general, the purpose of $f(\bullet)$ is to map certain types of error-word symbols to alternative symbol values, to allow the occurrence of errors of that type to be assigned either to an estimated C_2 codeword or to a C_1 codeword extracted from a composite-code codeword during decoding. All embodiments of the present invention employ a non-identity mapping function $f(\bullet)$.

The function $f(\bullet)$ may be applied to symbols, as discussed above, or may be applied to a vector of symbols. For example, the function $f(\bullet)$ may be applied to an entire codeword **u** to produce a modified codeword $f(\mathbf{u})$, with the symbol function $f(\bullet)$ applied to each symbol of the codeword to generate each corresponding symbol of the modified codeword.

Figure 15A provides a high-level control-flow diagram for encoding of information bits into a composite-code codeword according to one embodiment of the present invention. In step 1502, K_1 information symbols are received. In step 1503, K_1 information symbols are encoded by a first component code C_1 encoder to produce a C_1 codeword **u** of length N_1 symbols. In step 1504, K_2 information symbols are encoded by a second component code C_2 encoder to produce a codeword **v** of length N_2. In step 1505, a vector **w** of length N_2 symbols is obtained by adding a non-identity mapping of **u**, $f(\mathbf{u})$, to **v**. Finally, in step 1506, a composite-code-C codeword is generated by concatenating **u** and **w** together, the composite-code-C codeword having a length $N = N_1 + N_2$ and containing $K = K_1 + K_2$ information symbols.

Figure 15B illustrates construction of the composite code C[72,66,5] that represents one embodiment of the present invention. As discussed above, the composite code relies on two component codes. The component codes may be Reed-Solomon codes, systematic linear-block codes defined over GF(q), binary systematic linear block codes, or other types of codes. In the disclosed embodiment, the first component code C_1 produces codewords with $N_1 = 36$, $K_1 = 34$, and $D_1 = 3$ and the second component C_2 has the characteristics $N_2 = 36$, $K_2 = 32$, and $D_2 = 5$. It is assumed that C_1 can detect and correct $s1$ symbol erasures and $t1$ symbol errors, where $s1 + 2t1 < D_1$, and that C_2 can detect and correct $s2$ symbol erasures and $t2$ symbol errors, where $s2 + 2t2 < D_2$. In fact, such codes are well known.

As shown in Figure 15B, C_1 encodes $K_1 = 34$ information symbols 1512 to produce a 36-symbol C_1 codeword **u** 1516 and C_2 encodes $K_2 = 32$ information symbols 1514 to produce a 36-symbol C_2 codeword **v** 1518. These codewords are combined to create a codeword of the composite code C [72,66,5] that represents one embodiment of the present invention. Thus, $K_1 + K_2 = 32 + 34 = 66$ information symbols are encoded into each 72-symbol codeword of the composite code that represents one embodiment of the present invention. Both C_1 codeword **u** and C_2 codeword **v** have $N = 36$ symbols. The function $f(\bullet)$ is applied successively to each symbol in **u** to produce a vector $f(\mathbf{u})$ 1520. The vector $f(\mathbf{u})$ is then added to the C_2 codeword **v** 1522 to produce the vector $\mathbf{w} = f(\mathbf{u}) + \mathbf{v}$ 1524. Then, the codeword **u** 1516 is concatenated with $\mathbf{w} = f(\mathbf{u}) + \mathbf{v}$ to produce an $N = 72$ codeword 1526 of the composite code that represents one embodiment of the present invention. When the symbols of this codeword are transmitted or stored, the symbols from **u** and **w** alternate in the transmitted symbols, as shown in the sequence of transmitted symbols 1528, with symbol $\mathbf{u}_0$ 1530 first transmitted and symbol $\mathbf{w}_{\frac{n}{2}-1}$ 1532 last transmitted. Figure 16 illustrates a method of encoding a composite-code codeword that can be carried out repeatedly on an input stream of information symbols to produce an output stream of composite-code codewords. In step 1602, $K_1 + K_2$ information symbols are received for encoding. In step 1604, the first K_1 information symbols are encoded by a C_1 encoder to produce a C_1 codeword **u**. In step 1606, the next K_2 information symbols are encoded by a C_2 encoder to produce a C_2 codeword **v**. In step 1608, the vector $\mathbf{w} = f(\mathbf{u}) + \mathbf{v}$ is generated from **u** and **v** using the symbol-to-symbol mapping function $f(\bullet)$. Finally, in step 1610, **u** and **w** are concatenated together to produce a composite-code codeword. The encoding of a composite-code codeword by the method illustrated in Figure 16 can be carried out repeatedly on an input stream of information symbols to produce an output stream of composite-code codewords.

The above method of computing vector **w** generates a non-systematic code C. A systematic code C can be obtained by precoding. Precoding is carried out by extracting a prefix of length K_2 from $f(\mathbf{u})$, $\mathit{prefix}(f(\mathbf{u}))$, and creating a vector **a** comprising the next K_2 information symbols from the input stream. A word **v'** is then produced as: $\mathbf{v}' = \mathbf{a} - \mathit{prefix}(f(\mathbf{u}))$. Finally, **v'** is used as the K_2 information symbols that are encoded into a C_2 codeword **v**", and **v**" is then used to compute vector **w** by: $\mathbf{w} = f(\mathbf{u}) + \mathbf{v}''$.

In alternative embodiments of the present invention, composite-code codewords can be produced by other methods. The order of encoding using component codes may differ, the component codes may differ, and different symbol-to-symbol mapping functions may be employed. Alternative composite codes within the family of composite codes that represent embodiments of the present invention may have different characteristics *N*, *K*, and *D*, depending on the underlying code characteristic of the component codes C_1 and C_2. In alternative embodiments of the present invention, each component code may itself be generated from two or more underlying component codes.

Figure 17A illustrates the notion of a sub-block within a codeword of the composite code that represents one embodiment of the present invention. As shown in Figure 17, a composite-code codeword 1702 can be viewed as containing 8-bit symbols, such as symbol 1704 alternatively shown expanded into an 8-bit symbol vector 1706. Each pair of symbols, such as the pair of symbols 1708-1709, can be together viewed as a sub-block 1710. Thus, a composite-code codeword can be viewed alternatively as an ordered sequence of bits, an ordered sequence of 8-bit symbols, or as ordered sequence of sub-blocks.

Figure 17B illustrates the various different types of errors that the composite code that represents one embodiment of the present invention is designed to detect and correct. An important additional parameter of the composite code is the parameter *L*, a largest integer less than *D*/2. For various types of alternative composite codes, the value *L* may be fixed within the range of integers $1 < L < \frac{D}{2}$. A first type of error is referred to as a "phased-burst" error. A phased-burst error is illustrated in the first word 1712 shown in Figure 17B. A phased-burst error is any number of corrupted symbols within a block of adjacent symbols comprising *L* sub-blocks. As shown in the word 1712 in Figure 17B, four symbols, shown with cross-hatching 1714-1717 are corrupted, and all four symbols fall within a block comprising sub-blocks 4 and 5. It is assumed that a codeword containing a phased-burst error does not contain any sub-block erasures. In the case of the phased-burst error, when all four symbols within a block are corrupted, there is a small probability that the composite code may not be able to correct the errors. However, this small probability is smaller than the probability that a Reed-Solomon code with equivalent redundancy cannot correct the errors, and the composite codes of the current invention are more time efficient than Reed-Solomon codes with equivalent redundancy. When less than four symbols within the block are corrupted, all of the corrupted symbols can be corrected.

A tS error type is illustrated in the second codeword 1730 shown in Figure 17B. The tS error type includes up to L - t sub-block erasures and t corrupted symbols. In the example shown in Figure 17B, there is a single sub-block erasure 1722 and a single additional corrupted symbol 1724, so that t = 1 and L – t = 1 erased sub-block. Alternatively, there may be two erased sub-blocks and no additional corrupted symbols or two corrupted symbols and no additional erased sub-blocks. A third type of error condition to which the composite codes of the present invention are directed are 1R errors in which up to L sub-blocks are erased and one additional 1-bit error has occurred. The third codeword 1736 in Figure 17B illustrates a 1R error in which two sub-blocks 1738-1739 are erased and a single-bit error 1740 occurs in symbol 1742.

One motivation for development of the composite codes that represent embodiments of the present invention is for error correction of a newly developed type of electronic memory. Because of the construction of this memory, the majority of expected errors include phased-burst errors, tS-type errors, and 1R-type errors. Error correction is carried out in hardware in these electronic-memory systems, and therefore the error correction component represents a significant design and manufacturing overhead. For this reason, designers and manufacturers wish to use as efficient a code as possible for detecting and correcting the expected phased-burst, tS, and 1R errors. The composite codes that represent embodiments of the present invention successfully detect and correct these expected error types using less parity-check symbols than would be needed by a conventional Reed-Solomon code for an equal number of information symbols.

Figure 18 provides a high-level control-flow diagram for decoding of a composite code that represents one embodiment of the present invention. In step 1802, a composite-code-C codeword of length N, containing K information symbols, is received. In steps 1804-1806, a component-code-C_1 codeword $\mathbf{u}$ of length N_1 containing K_1 information symbols and a modified component-code-C_2 codeword of length N_2, $\mathbf{w} = \mathbf{v} + f(\mathbf{u})$, generated, during encoding, from a component-code-C_2 codeword $\mathbf{v}$ containing K_2 information symbols, where $K = K_1 + K_2$ and $N = N_1 + N_2$, and a non-identity mapping function, $f(\bullet)$, are extracted from the composite-code-C codeword and an estimated component-code-C_2 codeword $\hat{\mathbf{v}}$ and an estimated error word $\hat{\mathbf{e}}$ are generated from the modified component-code-C_2 codeword by applying a C_2 decoder to the modified component-code-C_2 codeword. In step 1808, which of a number of types of expected errors occurred subsequent to encoding of the composite-code-C codeword is determined from the error word $\hat{\mathbf{e}}$. When more than a first threshold number of erasures and erasures have occurred, but less than a second threshold number of errors have occurred, as determined in step 1816, the determined errors are assigned to either the component-code-C_1 codeword or to the modified component-code-C_2 codeword, and when assigned to the component-code-C_1 codeword, are corrected in steps 1818 and 1820. Other error and erasure occurrences are noted, in steps 1810, 1812, and 1814. In step 1822, an estimated component-code-C_2 codeword $\hat{\mathbf{u}}$ is obtained by applying a C_1 decoder to the estimated component-code-C_1 codeword $\hat{\mathbf{u}}$. Finally, in step 1824, K_1 information symbols are extracted from the estimated component-code-C_2 codeword $\hat{\mathbf{u}}$ and K_2 information symbols are extracted from the estimated component-code-C_2 codeword $\hat{\mathbf{v}}$ to produce K extracted information symbols.

Next, decoding of a received composite-code codeword is discussed. Figures 19-20 provide a control-flow diagram that illustrates one embodiment of the decoding process for composite codes that represent embodiments of the present invention. First, in step 1902, a composite-code C codeword is received. The received word can be viewed as two parts:

$$[\mathbf{u}_r \mid \mathbf{w}_r]$$

where $\mathbf{u}_r$ is the received $\mathbf{u}$, or $\mathbf{u} + e_1$

$\mathbf{w}_r$ is the received $\mathbf{w}$, or $f(\mathbf{u}) + v + e_2$

Next, in step 1903, the addition of $f(\mathbf{u})$ to $\mathbf{v}$ or $\mathbf{v}''$ during encoding is reversed by:

$$\mathbf{v}_r = -f(\mathbf{u}_r) + \mathbf{w}_r$$

Next, in step 1904, the computed word $\mathbf{v}_r$ is decoded using a C_2 decoder to produce estimated codeword $\hat{\mathbf{v}}$ and estimated error word $\hat{\mathbf{e}}$:

$$C_2^{-1}(\mathbf{v}_r) \rightarrow \hat{\mathbf{v}} \text{ and } \hat{\mathbf{e}}$$
$$\text{where } \hat{\mathbf{e}} = -f(e_1) + e_2$$

where the function-like symbol $C_2^{-1}(\bullet)$ represents decoding by a decoder for component code C_2.

If the C_2 decoding of $\mathbf{v}_r$ fails, as determined in step 1905, decoding of the composite-code codeword fails. Next, in a series of conditional steps, Boolean flags representing phased-burst ("PB"), tS, and 1R errors are set to indicate whether or not these types of errors appear to have occurred within the received word. Note that the notation "_1R" is used for the 1R flag, below, to be consistent with later-discussed pseudocode. It should be noted that the presence of erased sub-blocks is generally indicated by a separate, out-of-band erasure indication that is not part of the received word. When no erasures have occurred and when all symbol errors have occurred within *L* adjacent sub-blocks lined with a block boundary, as determined in step 1906 and as discussed above with reference to Figure 17B, then the flag PB is set TRUE in step 1908. Otherwise the flag PB is set to FALSE, in step 1910. When the flag PB contains the value FALSE and when the number of erased sub-blocks and the number of any additional non-zero symbols in the estimated error vector $\hat{\mathbf{e}}$ sum to a value less than or equal to *L*, as determined in step 1912, then the flag tS is set TRUE in step 1914. Otherwise the flag tS is set to be FALSE in step 1916. When both PB and tS contain the Boolean value FALSE, and when the number of erased sub-blocks is less than or equal to *L* and at most only one additional 1-bit symbol error has been found in the error vector $\hat{\mathbf{e}}$, as determined in step 1917B, then the flag _1R is set TRUE in step 1919. Otherwise the flag _1R is set to be FALSE in step 1920. A 1-bit error is detected when a non-zero symbol *s* in the estimated error vector $\hat{\mathbf{e}}$ is either an element of the set M or $-s$ is mapped to the set M by the symbol-to-symbol function $f^{-1}(\ )$, alternatively expressed as:

$$s \in \mathrm{M} \text{ or } f^{-1}(-s) \in M$$

Thus, $f(\bullet)$ maps a single-bit error that occurs in $\mathbf{u}_r$ to a symbol with more than two bits with bit value "1," so that a single-bit error in $\mathbf{u}_r$ can be distinguished from a single-bit error in $\mathbf{v}_r$. Coding resumes in the flow-control diagram of Figure 20. If none of the three Boolean flags PB, tS, and _1R are set to TRUE, as determined in step 2002, then the decoder returns a FALSE value in step 2004. Otherwise, vector $\mathbf{u}'$ is set to the first half of the received C codeword $\mathbf{u}_r$ in step 2006. If the flag _1R is set to TRUE, as determined in step 2008, then if a single non-zero symbol s_γ is found in the estimated error vector $\hat{\mathbf{e}}$ at position γ and s_γ is not an element of the set M, as determined in step 2010, the symbol at the same position γ in $\mathbf{u}'$ is replaced with the original symbol from which the inversely mapped negative error symbol is subtracted by GF(2^8) subtraction in step 2012. Steps 2008, 2010, and 2012 allow for detection of a single-bit error in addition to L sub-block erasures. When the non-zero $\hat{\mathbf{e}}$ symbol s_γ is an element of M, then the single-bit error occurred in the latter half of the received word, or, in other words, in $\mathbf{v}_r$. However, when s_γ can be mapped to M by $F^{-1}(-s_\gamma)$, the single-bit error occurred in the first portion of the C codeword. In that case, the error is corrected in step 2012. Next, if the Boolean flag PB contains the value TRUE, as determined in step 2014, and if there are non-zero symbols in $\hat{\mathbf{e}}$, as determined in step 2016, then the symbols in the block containing the errors are marked as erased in step 2018. If the flag tS contains the Boolean value TRUE, as determined in step 2020, and if there are any non-zero symbols in $\hat{\mathbf{e}}$ outside of any detected erasures, as determined in step 2022, then those additional symbol errors are marked as erasures in step 2024. In step 2026, a C_1 decoder is applied to $\mathbf{u}'$ to produce the estimated original vector $\hat{\mathbf{u}}$. If the C_1 decoder fails, as determined in step 2027, composite-code decoding fails. Otherwise, in step 2028, K_1 symbols are extracted from $\hat{\mathbf{u}}$ and K_2 symbols are extracted from $\hat{\mathbf{v}}$ that together form a sequence of K decoded information symbols that are returned in step 2030. As in the case of step 1904, should the C_1 decoder fail, in step 2026, then decoding fails.

Next, a C++-like pseudocode implementation of a decoding method for decoding the above-described composite code that represents one embodiment of the present invention is provided. Figure 21 illustrates the information received for each step of a decoding method for the composite code that represents one embodiment of the present invention. Received information includes an erasure map 2102 with a single bit for each symbol in the codeword indicating whether or not the symbol has been erased. The received information includes an erasure map 2102 that includes a bit flag for each symbol of a received word indicating whether or not the symbol has been erased, and a received word 2104 that, as discussed above, includes a first portion 2106 $\mathbf{u}_r$ which equals $\mathbf{u} + e_1$, although $\mathbf{u}$ and e_1 are not known, and a second part 2108 $\mathbf{v}_r$ which equals $F(\mathbf{u}) + \mathbf{v} + e_2$, although $\mathbf{u}$, $\mathbf{v}$, and e_2 are not known.

The pseudocode implementation first includes a number of constant integer declarations:

```
const int C1K = 34;
const int C2K = 32;
const int CK = C1K + C2K;
const int C1R = 2;
const int C2R = 4;
const int CR = C1R + C2R;
const int C1D = 3;
const int C2D = 5;
const int CD = 2 * C1D > C2D ? C2D : 2 * C1D;
const int N = CR + CK;
const int L = floor ((CD-1) / 2);
const int symPSubBlk = 2;
const int Nsub = N / symPSubBlk;
const int N2 = N / 2;
const int blkPlus = N2 / symPSubBlk;
const int b = 8;
```

These constants include the basic parameters for composite code C and component codes C1 and C2, discussed above, including: (1) *C1K*, *C2K*, and *CK*, the number of information symbols in the codewords of C1, C2, and C, respectively; (2) *C1R*, *C2R*, and *CR*, the number of parity-check symbols in codewords of C1, C2, and C, respectively; (3) the minimum distance between codewords *C1D*, *C2D*, and *CD* for codewords of C1, C2, and C, respectively; (4) a constant *N*, the number of symbols in a codeword of the composite code *C*; (5) the number *L*, equal to the largest integer less than *(CD-1)*/2 in the disclosed implementation, as discussed above; (6) *N2*, the number of symbols in codewords of component code C1 and C2, where *N2* = *N*/2; *symPSubBlk*. the number of symbols per sub-block; (7) blkPlus, that, when added to the sub-block index of a block in a first portion of a composite codeword, generates the sub-block index of the corresponding sub-block of a second portion of the composite codeword; and (8) a constant *b*, the number of bits in a symbol, or, equivalently, a number equal to *m* in the expression GF(2^m) for the field over which the composite code C is constructed.

Next, type definitions are provided: (1) for a codeword symbol; (2) C, C1, and C2 codewords; and (3) erasure maps for C, C1, and C2 codewords:

```
1 typedef unsigned char symbol; // b <= 8 only
2 typedef symbol C_WORD[N];
3 typedef symbol C1_WORD[N2];
4 typedef symbol C2_WORD[N2];
5 typedef bool C_ERASURE_WORD[N];
6 typedef bool C1_ERASURE_WORD[N2];
7 typedef bool C2_ERASURE_WORD[N2];
```

It should be noted that the C++ type "unsigned char" can only be used to represent a symbol when the constant *b* is less than or equal to 8. When *b* = 8, the unsigned-char data type, also referred to as a "byte," is exactly the size needed to represent each symbol expressed as a tuple of binary coefficients, and thus GF(2^8) is a most convenient field over which to construct a code, for computational efficiency.

Next, a declaration is provided for the set *M* which includes all symbols with 8-bit-tuple representations that include only a single bit with the bit value "1." This declaration employs the fact that the tuples in set *M* correspond to bytes, in normal binary byte-value representations, to powers of two:

```
1 const symbol M[b] = {1, 2, 4, 8, 16, 32, 64, 128}; // elements of GF(2^b) with a
                                                    // single-bit tuple representation
```

Next, declarations for five functions are provided:

```
1 bool C1(C1_WORD c1Word, C1_ERASURE_WORD erasures,
2         C1_WORD decodedC1Word, C1_WORD errors);
3 bool C2(C2_WORD c2Word, C2_ERASURE_WORD erasures,
4         C2_WORD decodedC2Word, C2_WORD errors);
5 symbol f(symbol a);
6 symbol fInverse(symbol a);
7 symbol GF2bSubtraction(symbol y, symbol z);
```

The first two functions are decoders for component codes C1 and C2. These two functions receive a codeword and erasure map and return a decoded codeword and an error word, as described above. The function $f(\bullet)$ and the function $f^{-1}(\bullet)$, discussed above, are declared on lines 5 and 6 of the above code block. Finally, on line 7, a GF(2^8) subtraction function for subtracting a GF(2^8) symbol z from a GF(2^8) symbol y is provided. As discussed above, it is assumed that component codes *C1* and *C2* exist, and that encoders and decoders are available for these component codes. No implementations are provided for the above five functions, as the decoder implementations depend on the particular component codes selected for use in constructing a composite code, because the functions $f(\bullet)$ and $f^{-1}(\bullet)$ are straightforwardly implemented, the implementations depending on the field over which the composite code is defined, and because GF(2^b) subtraction is well known.

Next, a number of class declarations are provided. First, three classes that represent an input symbol stream, an input erasure stream, and an output symbol stream are provided:

```
class symbolStream
{
    public:
            bool start();
            bool getNext(int num, symbol* buffer);
};
```

```
class erasureStream
{
    public:
            bool start();
            bool getNext(int num, bool* buffer);
};
```

```
class outputStream
{
    public:
            bool start();
            void outputNext(int num, symbol* buffer);
            void finish();
};
```

The various streams can be started and then accessed in order to input or output specified numbers of symbols. Implementations are not provided for these classes, since stream input and output is both well known, operating-system dependent, and possibly hardware-platform dependent.

Next, a class declaration for a class "C-decoder" is provided:

```
1 class C_decoder
2 {
3     private:
4             symbolStream s;
5             erasureStream Er;
6             outputStream out;
7
8             void deInterleave(C_WORD c, C_ERASURE_WORD er);
9             bool decodeNextBlock(C_WORD c, C_ERASURE_WORD er,
10                                   symbol* buffer);
11
12    public:
13            bool decode();
14 };
```

The class "C_decoder" includes three private data members *s*, *Er*, and *out* that represent instances of the symbol stream, erasure stream, and output stream classes, respectively. The class "C_decoder" includes two private function members, declared on lines 8-10. The first private function member, "deInterleave," transforms *n* symbols received from an input stream into a C codeword by deinterleaving the symbols that are interleaved, as discussed with reference to Figure 15 (specifically 1518 in Figure 15). The private function member "decodeNextBlock" receives a C codeword and a corresponding erasure map and outputs *K* decoded information symbols to an output stream. The single public function member "decode," declared on line 13, continuously decodes symbols from an input stream and outputs corresponding decoded information symbols to an output stream.

Implementation of the function member "decode" is next provided:

```
1 bool C_decoder::decode()
2 {
3     s.start();
4     Er.start();
5     out.start();
6     C_WORD c;
7     C_ERASURE_WORD er;
8     symbol buffer[CK];
9
10    while (s.getNext(N, c) && Er.getNext(N, er))
11    {
```

```
12              deInterleave (c, er);
13              if (!decodeNextBlock(c, er, buffer)) return false;
14              out.outputNext(CK, buffer);
15      }
16      return (true);
17 };
```

In the *while*-loop of lines 10-15, the function member "decode" extracts a next codeword and corresponding erasure map from the input streams *c* and *Er*, deinterleaves the input symbols on line 12, decodes the codeword on line 13, and outputs corresponding decoded information symbols on line 14. This loop continues until either decoding fails, on line 13, or until there are no additional coded symbols available from the information stream, as determined on line 10.

Next, an implementation of the function member "decodeNextBlock" is provided:

```
bool C_decoder::decodeNextBlock(C_WORD c, C_ERASURE_WORD er,
                                symbol* buffer)
{
    symbol* ur = &(c[0]);
    symbol* wr = &(c[N2]);

    bool* er1 = &(er[0]);
    bool* er2 = &(er[N2]);

    C1_WORD uHat, uPrime, e1Hat;
    C2_WORD vHat, vr, e2Hat;

    bool PB, tS, _1R, erased;

    symbol gamma;

    int i, j, blkIndex, gammaIndex;
    int erasures[L];
    int numErasures = 0;
    int nonZeroSymbols[L];
    int numNonZeroSymbols = 0;

    for (i = 0; i < N2; i++)
            vr[i] = -f(ur[i]) + wr[i]; //vr = v + -f(e1) + e2

    if (!C2(vr, er2, vHat, e2Hat)) return false;

    for (i = 0; i < N; i++)
    {
            if (er[i])
```

```
        {
                blkIndex = (i / symPSubBlk) * symPSubBlk;
                if ((i != blkIndex) && er[blkIndex]) continue;
                if (numErasures == L) return false;
                else erasures[numErasures++] = blkIndex;
        }
}

for (i = 0; i < N2; i++)
{
 erased = false;
 blkIndex = (i / symPSubBlk) * symPSubBlk;
 for (j = 0; j < numErasures; j++)
 {
   if (erasures[j] == blkIndex || erasures[j] == blkIndex + blkPlus)
        erased = true; break;
 }
 if (erased)
        i = ((blkIndex + 1) * symPSubBlk) - 1;
 else
        {
                if (e2Hat[i] != 0)
                {
                        if (numNonZeroSymbols == L) return false;
                        else nonZeroSymbols[numNonZeroSymbols++] = i;
                }
        }
}

if (numErasures == 0)
{
   if (numNonZeroSymbols == 0) PB = true;
        else if (nonZeroSymbols[numNonZeroSymbols - 1] -
                nonZeroSymbols[0] <= L)
           PB = true;
        else PB = false;
}

if (!PB && numNonZeroSymbols + numErasures < L) tS = true;
else tS = false;

if (!PB && !tS && numNonZeroSymbols == 1)
{
        gammaIndex = nonZeroSymbols[0];
        gamma = 0;
        if (e2Hat[gammaIndex] == 0) _1R = true;
        else
        {
                _1R = false;
                for (i = 0; i < b; i++)
                {
                        if ((e2Hat[gammaIndex] == M[i]))
 83                     {
 84                             _1R = true;
 85                             break;
 86                     }
 87                     gamma = fInverse(-e2Hat[gammaIndex]);
 88                     if (gamma = M[i])
 89                     {
 90                             _1R = true;
 91                             break;
 92                     }
 93                     gamma = 0;
 94             }
 95         }
 96 }
 97
 98 if (!PB && !tS && !_1R) return false;
 99
100 for (i = 0; i < N2; i++) uPrime[i] = ur[i];
101
102 if (PB)
103         for (i = 0; i < numErasures; i++)
104                 for (j = 0; j < symPSubBlk; j++)
105                         er1[i + j] = true;
106
107 if (tS)
108         for (i = 0; i < numNonZeroSymbols; i++)
109                 er1[nonZeroSymbols[i]] = true;
110
111 uPrime[i] = GF2bSubtraction(uPrime[i], gamma);
112
113 if (!C1(uPrime, er1, uHat, e1Hat)) return false;
114
115 for (i = 0; i < C1K; i++) *buffer++ = uPrime[i];
116 for (i = 0; i < C2K; i++) *buffer++ = vHat[i];
117 return true;
118 }
```

The function member "decodeNextBlock" receives a composite-code codeword *c*, corresponding erasure map *er*, and a symbol buffer in which to place the decoded information symbols corresponding to received word *c*. On lines 4-5 symbol pointers *ur* and *wr* are declared to point to the first and second halves of the received word C. These symbol pointers *ur* and *wr* correspond to $\mathbf{u}_r$ and $\mathbf{w}_r$ in Figure 21. Similarly, on lines 7-8, erasure-map pointers *er1* and *er2* are declared to point to the portions of the received erasure word *er* corresponding to the first half and the second half of the received word C, respectively.

On lines 10-21, a number of local variables are declared. These local variables have names corresponding to the notation used in the above discussions of the composite code, component codes, composite-code encoding, and composite-code decoding. For example, the variable vHat, declared on line 11, represent the estimated decoded codeword $\hat{v}$ discussed above. The arrays "erasures" and "nonZeroSymbols," declared on lines 18 and 20, respectively, contain the indices of erased sub-blocks and the indices of additional errors detected in the erasure map and the estimated error vector *e2Hat*, respectively. The use of the local variables is clarified by their use, described below, in the function member "decodeNextBlock."

On lines 23-24, the vector "vr" is computed as $vr = wr - f(ur)$. On line 26, *vr* is decoded to produce $\hat{v}$ and $\hat{e}$, referred in the code as "vHat" and "e2Hat," respectively. In the *for*-loop of lines 28-37, indices of all erased sub-blocks are determined and stored in the array "erasures." Note that if the number of sub-block erasures is greater than *L*, the decode routine fails, since only up to *L* erasures can be detected and corrected by the composite code implemented in the pseudocode. Note also that if the C2 decoder, invoked on line 26, fails, then decoding fails. Next, in the *for*-loop of lines 39-58, any errors in $\hat{e}$, represented by non-zero symbols, apart from any detected erased sub-blocks are noted, and the indices of the non-zero symbols corresponding to the errors are stored in the array "nonZeroSymbols."

On lines 60-67, the Boolean flag *PB* is set to TRUE or FALSE, depending on whether or not a phased-burst error is detected in the codeword. *PB* is set to TRUE when there are no erasures and when either there are no additional error symbols or all of the error symbols occur within a single block composed of *L* adjacent sub-blocks. Recall that the function member "decodeNextBlock" will have already failed if there are more than *L* erased sub-blocks. Next, on lines 69-70, the Boolean flag *tS* is set to TRUE or FALSE, depending on whether or not a *tS*-type error is detected in the received word. The flag *tS* is set to TRUE when PB is FALSE and the number of erased sub-blocks added to the number of additional error symbols produces a sum less than or equal to *L*.

Next, on lines 72-96, the Boolean flag _1R is set to TRUE or FALSE. The Boolean flag _1R is set to TRUE when there is a single additional 1-bit error, or no additional errors, along with up to L erased sub-blocks. Note that decoding has already failed, on line 34, if more than L erased sub-blocks were detected. An error symbol represents a 1-bit error when either the error symbol is a member of the set M, as determined on line 81, or an inverse mapping by $f^{-1}(\bullet)$ of the GF(2^b)-additive inverse of the symbol value of the error symbol maps to M, as determined on line 88.

When all of the Boolean flags are FALSE, as determined on line 98, then decoding fails. Otherwise, *uPrime* is set to the first portion of the received word *c* on line 100. When PB is TRUE, all symbols of all sub-blocks containing errors are marked as erasures, on lines 102-105. When *tS* is TRUE, then all additional error symbols are marked as erasures, on lines 107-109. When a 1-bit additional error is detected in the first portion of the codeword, on line 88, then, on line 109, *uPrime* is altered to correct the area by subtraction of the inversely mapped inverse symbol value from *uPrime*, on line 111. Finally, on line 113, *uPrime* is decoded by the C1 decoder. If the C1 decoder fails, then decoding fails. Otherwise, the information symbols in *uPrime* and *vHat* are placed in the buffer for return to the member function "decode."

As mentioned above, composite codes that represent embodiments of the present invention may be constructed to efficiently detect and correct specific types of error and erasure patterns and occurrences. For example, suppose that it is desired to detect and correct up to L erased sub-blocks and t additional random single-bit errors in a symbol. When $L < D1$ and $L + 2t < D2$, and when a linear code C' over GF(2) exists with $C'N$ equal to $2*b$, dimension $K' = b$, and minimum codeword distance $D' \geq 2 * t + 1$, where C' is defined by a parity check matrix $\mathbf{H}' = [\mathbf{I}_b \mid -\mathbf{A}]$, and where $-\mathbf{A}$ is invertible, then the above described composite code can be used to detect up to $L = 2$ erased sub-blocks and $t = 2$ additional random single-bit errors, in the case of the above-discussed composite code. Note that $-\mathbf{A}$ is a b x b matrix over GF(2). In this case, the symbol-to-symbol mapping function $f(\bullet)$ is defined to be: $f(\bullet) = \mathbf{u}_i \cdot \mathbf{A}^T$ where $\mathbf{u}_i$ is a symbol of GF(2^8), in the case of the above-discussed composite code. The condition "$L < D1$ and $L + 2t < D2$" ensures that the $C2$ decoder can successfully decode $\mathbf{v}'_r$. The condition related to linear code C' ensures that $f(\bullet)$ will successfully map a symbol $\mathbf{u}_i$ with one or two random-bit errors to a different symbol distinguishable from a one-or-two-random-bit-error corrupted symbol, so that the composite-code decoder can determine in which of the two halves of the codeword that a one-or-two-random-bit corruption occurred. In the current case, a non-zero symbol $\hat{\mathbf{e}}_i = -f(\hat{\mathbf{e}}_{i,1}) + \hat{\mathbf{e}}_{i,2}$ in the C2 error word $\hat{\mathbf{e}}$ can be used to generate a syndrome for C', $\mathbf{s} = \hat{\mathbf{e}}_i \mathbf{H}'^T$, and the syndrome s can then be used to select a corresponding C' error word $\mathbf{e}'$ that comprises the concatenation of $\hat{\mathbf{e}}_{i,2}$ and $-f(\hat{\mathbf{e}}_{i,1})$. Thus, a random-two-bit-error corrupted symbol in $\hat{\mathbf{e}}$ can be attributed to the first half or the second half of a composite-code codeword.

Figure 22 shows a block diagram of a physical memory device in which embodiments of the present invention may be employed. The memory 2202 includes a bank of individual DRAM component memories 2204-2208, a bus controller and logic for receiving and transmitting data 2210, an encoder 2212 for applying a composite code to data values prior to storage in the memory, and a decoder 2214 for decoding encoded values retrieved from the memory. Memory operations include storing a block of data words 2216 identified by an address and size, in words, 2218 into the memory and retrieving a block of words 2220 identified by an address and size, in words, 2222 from the memory.

Figure 23 illustrates mapping between codeword symbols and DRAM units in a bank of DRAM units that together comprise the electronic data storage component of the physical memory device illustrated in Figure 22. In one embodiment, each word received for storage in the memory is encoded to a composite-code codeword 2303 by the encoder component of the memory (2212 in Figure 22), which can be viewed as an array of blocks, such as block 2304, each block comprising a number of sub-blocks, such as sub-block 2306. Each block is mapped into a corresponding DRAM, as indicated by double-headed arrows 2308-2313 in Figure 23. Thus, for example, a DRAM failure would result in a phased-burst error spanning a block of the codeword. The composite codes of the present invention are designed to correct the most probable failure modes of the memory. For example, the above-discussed composite code can correct for a single DRAM failure, several subseveral sub-block and symbol failures in several DRAMS. By using the composite code, the probability of a memory error, already quite low due to the low probability of a memory-component error, is substantially lowered by correcting for any of the most probable component errors.`

Although the present invention has been described in terms of particular embodiments, it is not intended that the invention be limited to these embodiments. Modifications within the spirit of the invention will be apparent to those skilled in the art. For example, as discussed above, any number of different component codes may be combined to create a composite code, providing that suitable symbol-to-symbol mapping functions $f(\bullet)$ can be found to map certain errors to corresponding symbols that pass through component-code encodings. The encoding and decoding methods for composite codes may be implemented in software, firmware, hardware, or a combination of two or more of software, firmware, and hardware. Software implementations may employ any of a variety of different programming languages, modular organizations, control structures, data structure, and may vary by any of many other such programming parameters. Composite codes of the present invention may be devised for efficient detection and correction of many different types of error and erasure patterns and occurrences. In alternative embodiments of the present invention, different symbol-to-symbol mapping functions may be employed to determine the location of certain types of errors in a composite-code codeword. In still alternative embodiments of the present invention, the mapping function $f(\bullet)$ may map pairs of symbols to other pairs of symbols, or may map other portions of a codeword to different values.

The foregoing description, for purposes of explanation, used specific nomenclature to provide a thorough understanding of the invention. However, it will be apparent to one skilled in the art that the specific details are not required in order to practice the invention. The foregoing descriptions of specific embodiments of the present invention are presented for purpose of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations are possible in view of the above teachings. The embodiments are shown and described in order to best explain the principles of the invention and its practical applications, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalents:

CLAIMS

1. A method for encoding K information symbols, the method comprising:

using a first component code C_1 to encode K_1 information symbols in a C_1 codeword **u** of length N_1 symbols;

using a second component code C_2 to encode K_2 information symbols in a C_2 codeword **v** of length N_2;

generating a vector **w** of length N_2 symbols by adding a non-identity mapping of **u**, $f(\mathbf{u})$, to v; and

generating a composite-code-C codeword by concatenating **u** and **w** together, the composite codeword of length $N = N_1 + N_2$ containing $K= K_1 + K_2$ information symbols.

2. The method of claim 1 wherein component codes C_1 and C_2 and composite code C are linear block codes over GF(q) containing symbols that each comprises an element of GF(q).

3. The method of claim 1 wherein component codes C_1 and C_2 and composite code C are linear block codes over GF(2^8) containing symbols that each comprises an 8-bit element of GF(2^8).

4. The method of claim 3 wherein:

N_1 equals 36 symbols;

K_1 equals 34 symbols;

N_2 equals 36 symbols;

K_2 equals 32 symbols;

$N = 72$ symbols; and

$K = 66$ symbols.

5. The method of claim 1 wherein the non-identity mapping $f(\bullet)$ is applied, symbol-by-symbol, to each symbol $\mathbf{u}_i$ in vector **u** and maps each symbol value equal to a particular type of expected error-word symbol value to a different symbol value that can be used to

subsequently identify whether or not an error of the expected type, detected on decoding of the C codeword, has occurred in **u** or in **w.**

6. The method of claim 1 wherein there is an inverse function $f^{-1}(\bullet)$ such that $f^{-1}(f(\mathbf{u})) = \mathbf{u}$.

7. The method of claim 1 wherein the non-identity mapping $f(\bullet)$ maps a symbol $\mathbf{u}_i$, a bit-tuple representation of which includes only a single bit having binary value "1," to a different symbol $f(\mathbf{u}_i)$, a bit-tuple representation of which includes at least two bits having binary value "1."

8. A memory device that includes an encoder that encodes words received for storage in the memory by the method of claim 1.

9. Computer instructions encoded in a computer-readable medium for encoding K information symbols by the method of claim 1.

10. A method for decoding a composite-code-C codeword of length N, containing K information symbols, to extract the K information symbols, the method comprising:

extracting, from the composite-code-C codeword, a component-code-C_1 codeword of length N_1 containing K_1 information symbols and a modified component-code-C_2 codeword of length N_2 generated, during encoding, from a component-code-C_2 codeword containing K_2 information symbols, where $K = K_1 + K_2$ and $N = N_1 + N_2$, and a non-identity mapping function, $f(\bullet)$;

generating an estimated component-code-C_2 codeword $\hat{\mathbf{v}}$ and an estimated error word $\hat{\mathbf{e}}$ from the modified component-code-C_2 codeword by applying a C_2 decoder to the modified component-code-C_2 codeword;

determining, from the error word $\hat{\mathbf{e}}$, which of a number of types of expected errors occurred subsequent to encoding of the composite-code-C codeword;

when more than a first threshold number of erasures and erasures have occurred, but less than a second threshold number of errors have occurred, assigning determined errors to either the component-code-C_1 codeword or to the modified component-code-C_2 codeword;

correcting any of the determined errors in the component-code-C_1 codeword that can be corrected based on the estimated component-code-C_2 codeword $\hat{\mathbf{v}}$ and an estimated error word $\hat{\mathbf{e}}$;

generating an estimated component-code-C_2 codeword $\hat{\mathbf{u}}$ by applying a C_1 decoder to the component-code-C_1 codeword; and

extracting K_1 information symbols from the estimated component-code-C_2 codeword $\hat{\mathbf{u}}$ and K_2 information symbols from the estimated component-code-C_2 codeword $\hat{\mathbf{v}}$ to produce K extracted information symbols.

11. The method of claim 10 wherein component codes C_1 and C_2 and composite code C are linear block codes over GF(q) containing symbols that each comprises an element of GF(q).

12. The method of claim 10 wherein component codes C_1 and C_2 and composite code C are linear block codes over GF(2^8) containing symbols that each comprises an 8-bit element of GF(2^8).

13. The method of claim 12 wherein:

N_1 equals 36 symbols;

K_1 equals 34 symbols;

N_2 equals 36 symbols;

K_2 equals 32 symbols;

$N = 72$ symbols; and

$K = 66$ symbols.

14. The method of claim 10 wherein determining, from the error word $\hat{\mathbf{e}}$, which of a number of types of expected errors occurred subsequent to encoding of the composite-code-C codeword further comprises:

considering an additionally received indication of erased symbols in the composite-code-C codeword to determine whether any of a number of different types of errors occurred in the composite-code-C codeword following encoding of the composite-code-C codeword.

15. The method of claim 10 wherein the number of different types of errors include:

a phased burst error, comprising only erroneous symbols within a threshold number of adjacent sub-blocks of symbols within the composite-code-C codeword;

a type tS error, comprising up to a threshold number of erased sub-blocks of symbols and additional erroneous symbols; and

a type 1R error, comprising up to a first threshold number of erased sub-blocks and up to a second hreshold number of additional single-bit errors.

16. The method of claim 15 wherein assigning determined errors to either the component-code-C_1 codeword or to the component-code-C_2 codeword further includes:

for each non-zero symbol in the error word $\hat{\mathbf{e}}$ indicative of an expected type of error, assigning an error of the expected type of error to the estimated component-code-C_2 codeword $\hat{\mathbf{v}}$; and

for each non-zero symbols in the error word $\hat{\mathbf{e}}$, $\hat{\mathbf{e}}_i$, that can each be mapped by an inverse of a non-identity mapping $f(\bullet)$, $f^{-1}(\bullet)$, to a symbol indicative of an expected type of error, assigning an error of the expected type of error to the component-code-C_1 codeword.

17. The method of claim 15 wherein the non-identity mapping $f(\bullet)$ maps a symbol $\hat{\mathbf{e}}_i$, a bit-tuple representation of which includes only a single bit having binary value "1," to a symbol $f(\hat{\mathbf{e}}_i)$, a bit-tuple representation of which includes at least two bits having binary value "1."

18. The method of claim 15 wherein correcting any of the determined errors in the component-code-C_1 codeword that can be corrected based on the estimated component-code-C_2 codeword $\hat{\mathbf{v}}$ and an estimated error word $\hat{\mathbf{e}}$ further includes:

marking sub-blocks containing erasures as being erased in the component-code-C_1 codeword; and

correcting any symbols in the component-code-C_1 codeword corresponding to error-word symbols that can be mapped by an inverse of a non-identity mapping $f^{-1}(\bullet)$ to a symbol indicative of an expected type of error.

19. A memory device that includes a decoder that decodes words retrieved from a memory component of the memory device by the method of claim 10.

20. Compute instructions encoded in a computer-readable medium for encoding K information symbols by the method of claim 10.

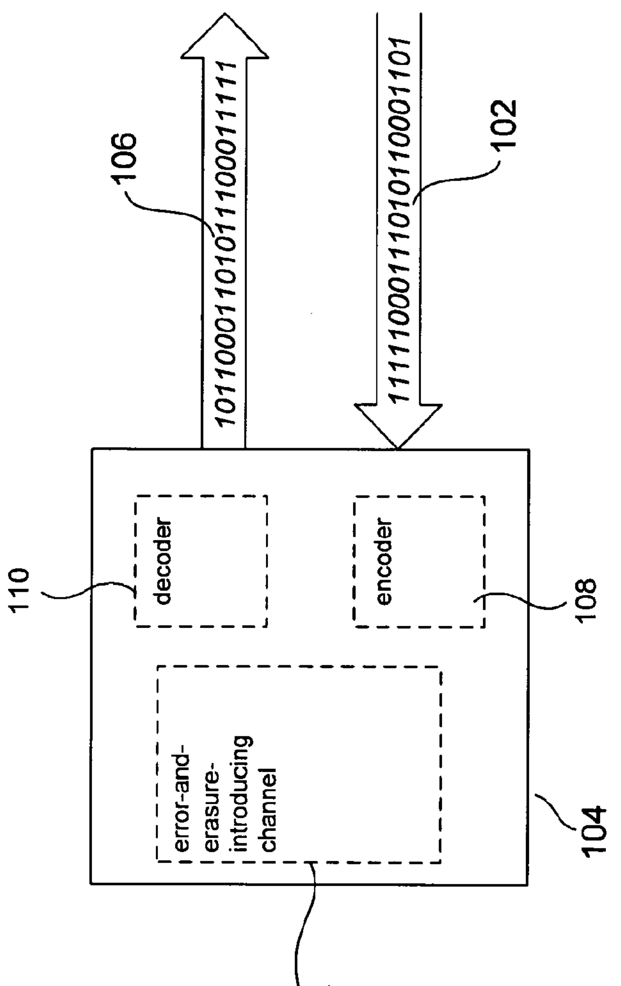

106
1011000110101011100011111
102
1111100011101010110001101
110
decoder
encoder
108
error-and-erasure-introducing channel
104
105

Figure 1

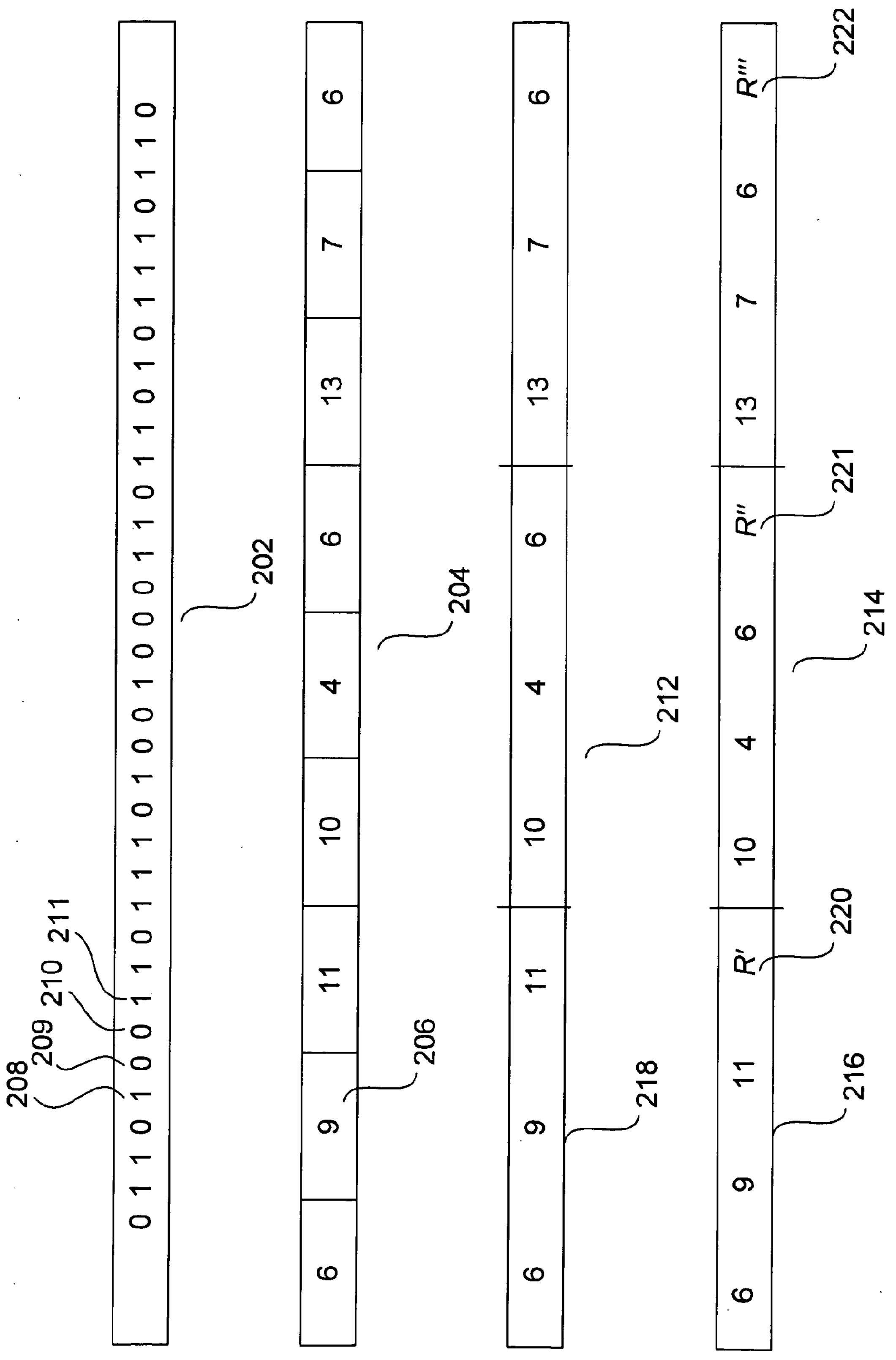

0 1 1 0 1 0 0 1 1 0 1 1 1 0 1 0 0 1 0 0 0 1 1 0 1 1 0 1 0 1 1 1 0 1 1 0
202
208
209
210
211
6 9 11 10 4 6 13 7 6
204
206
6 9 11 10 4 6 13 7 6
212
218
6 9 11 R' 10 4 6 R'' 13 7 6 R'''
214
216
220
221
222

Figure 2

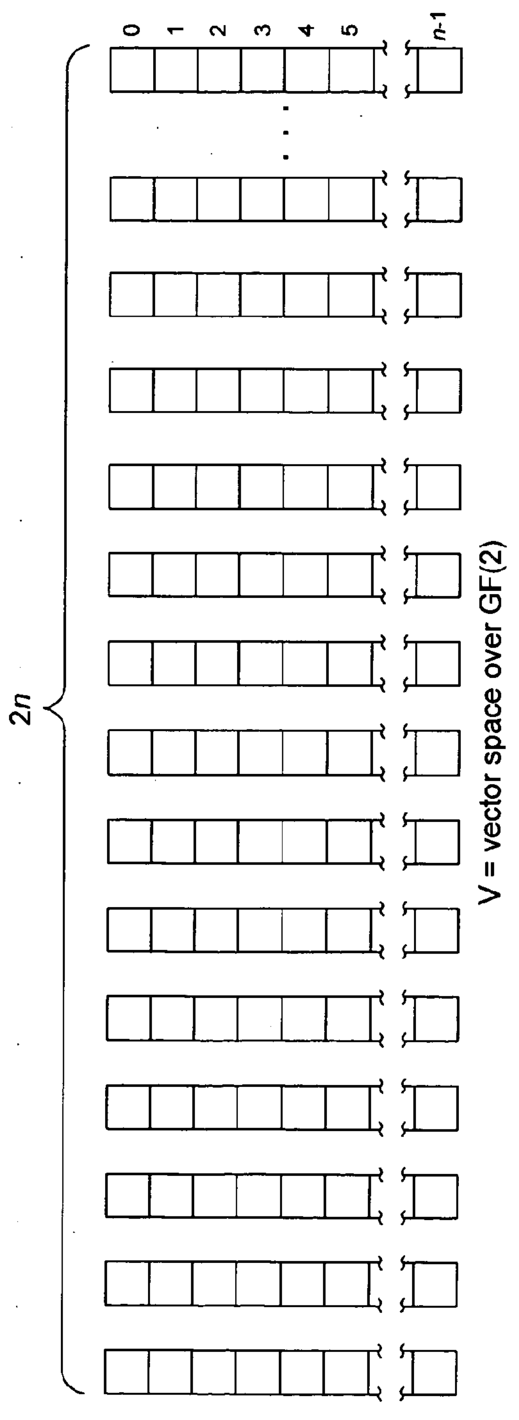


Figure 3A

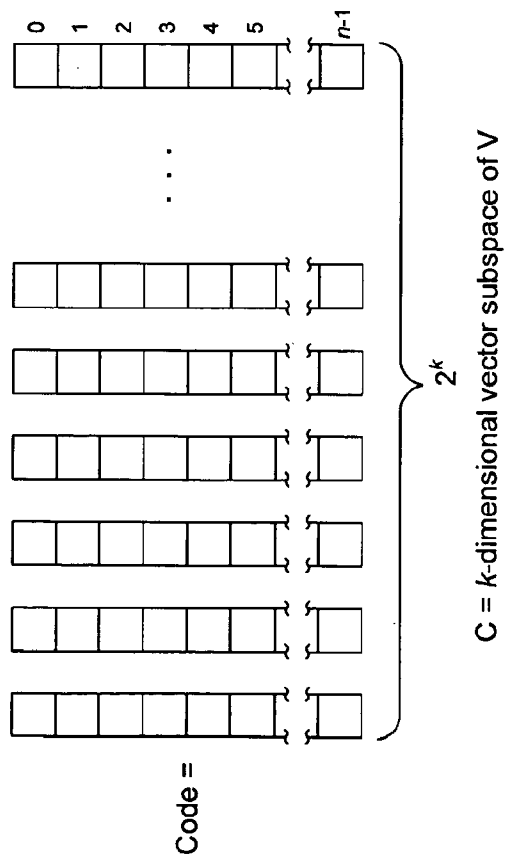


Figure 3B

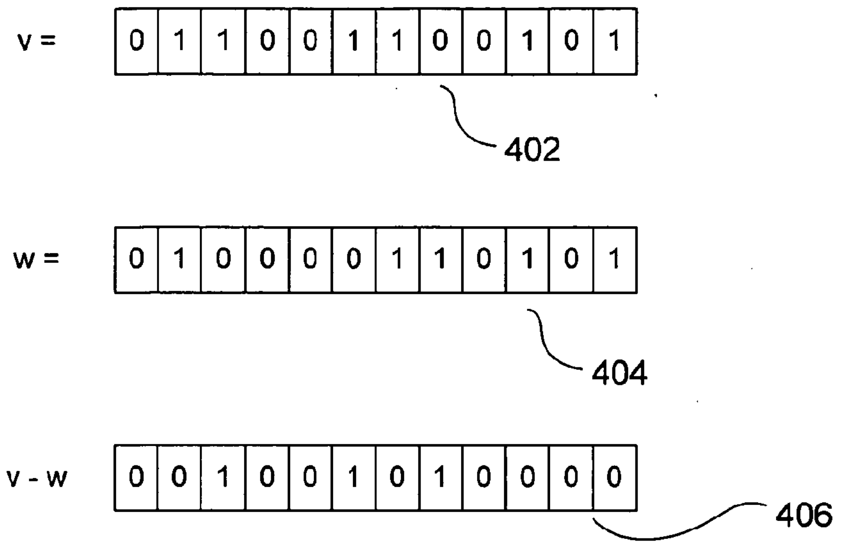


Figure 4

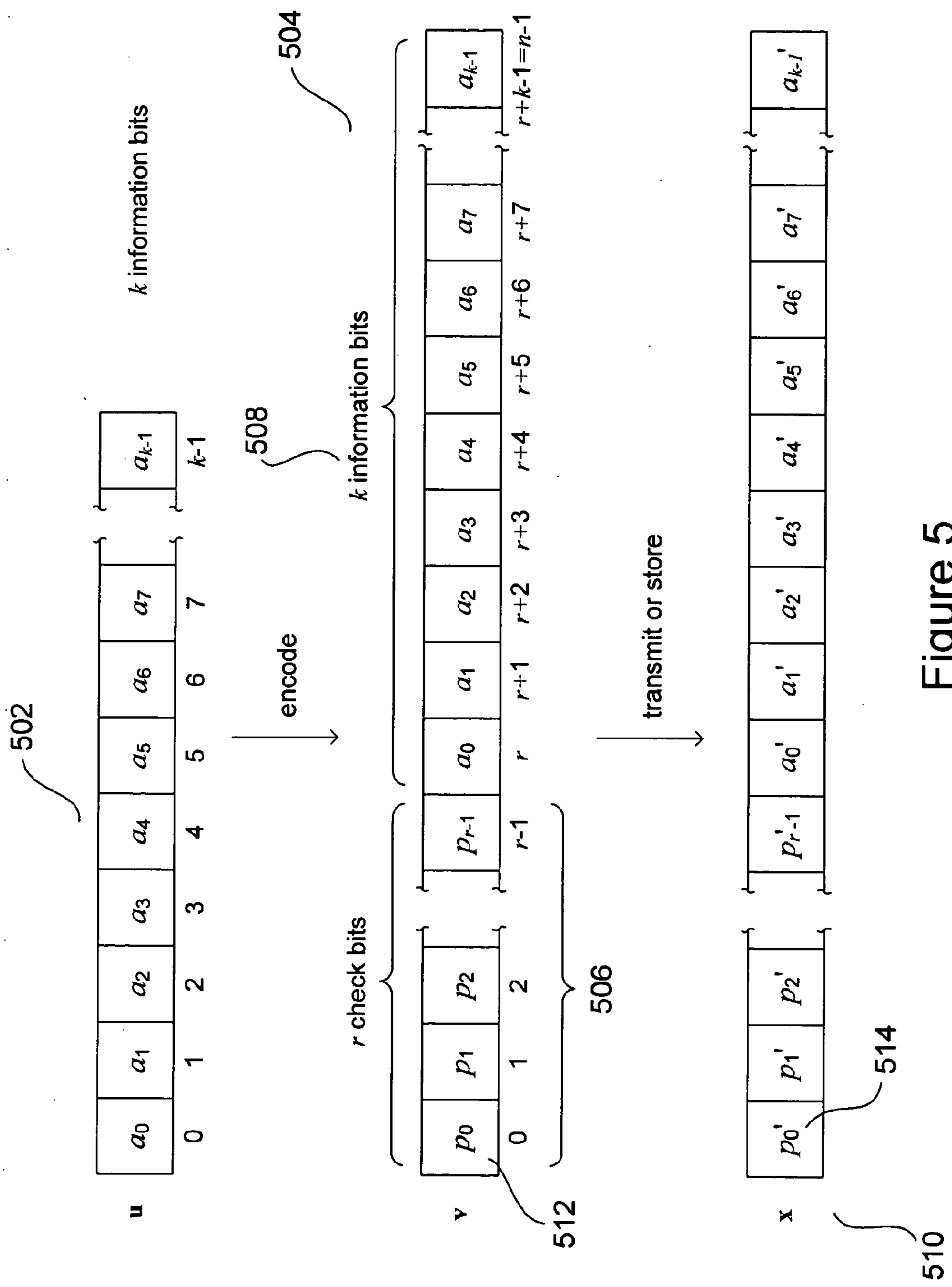
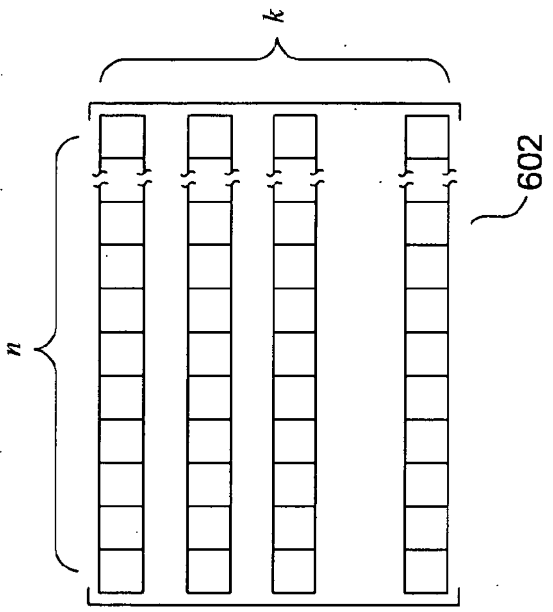

502
u
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_{k-1}
0 1 2 3 4 5 6 7 k-1
k information bits
encode
504
508
v
p_0 p_1 p_2 p_{r-1} a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_{k-1}
0 1 2 r-1 r r+1 r+2 r+3 r+4 r+5 r+6 r+7 r+k-1=n-1
r check bits
k information bits
506
512
transmit or store
x
p_0' p_1' p_2' p_{r-1}' a_0' a_1' a_2' a_3' a_4' a_5' a_6' a_7' a_{k-1}'
514
510

Figure 5



608 604 606

$$\underbrace{\begin{bmatrix} v_0 & v_1 & v_2 & \cdots & v_{n-3} & v_{n-2} & v_{n-1} \end{bmatrix}}_{\mathbf{v}} \quad \underbrace{\begin{bmatrix} u_0 & u_1 & u_2 & \cdots & u_{k-1} \end{bmatrix}}_{\mathbf{u}} \cdot \begin{bmatrix} g_{0,0} & g_{0,1} & g_{0,2} & g_{0,3} & \cdots & g_{0,n-1} \\ g_{1,0} & & & & & \\ g_{2,0} & & & & & \\ g_{3,0} & & & \ddots & & \\ \vdots & & & & & \\ g_{k-1,0} & & & & & g_{k-1,n-1} \end{bmatrix}$$

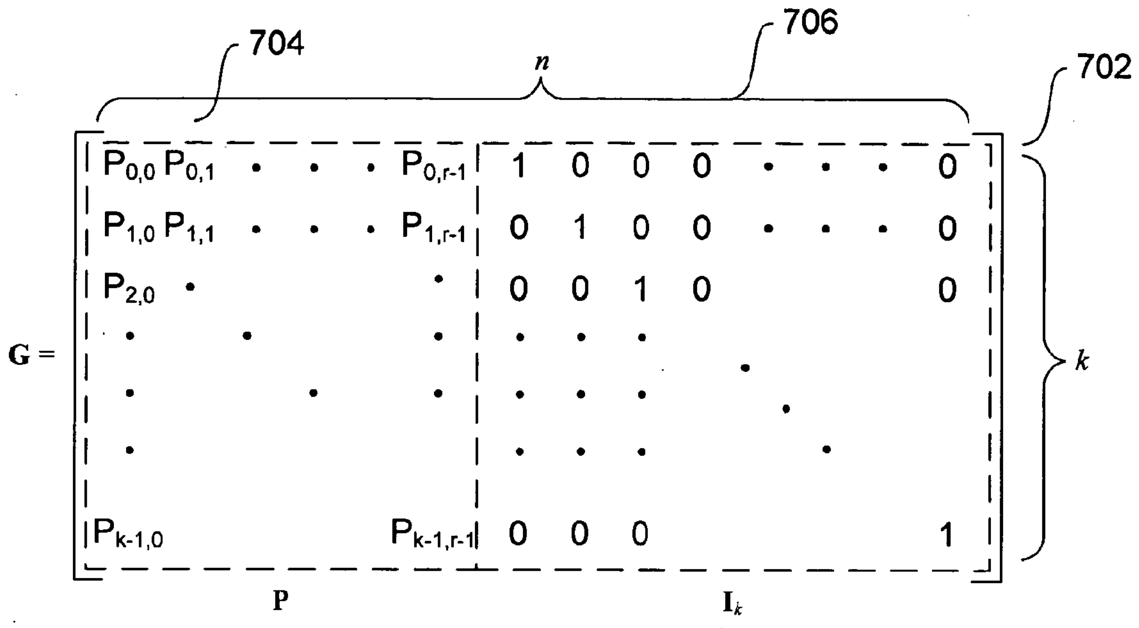

704
706
702
n
k
G =
P
I_k
$P_{0,0}$ $P_{0,1}$ $P_{0,r-1}$
$P_{1,0}$ $P_{1,1}$ $P_{1,r-1}$
$P_{2,0}$
$P_{k-1,0}$ $P_{k-1,r-1}$

Figure 7A

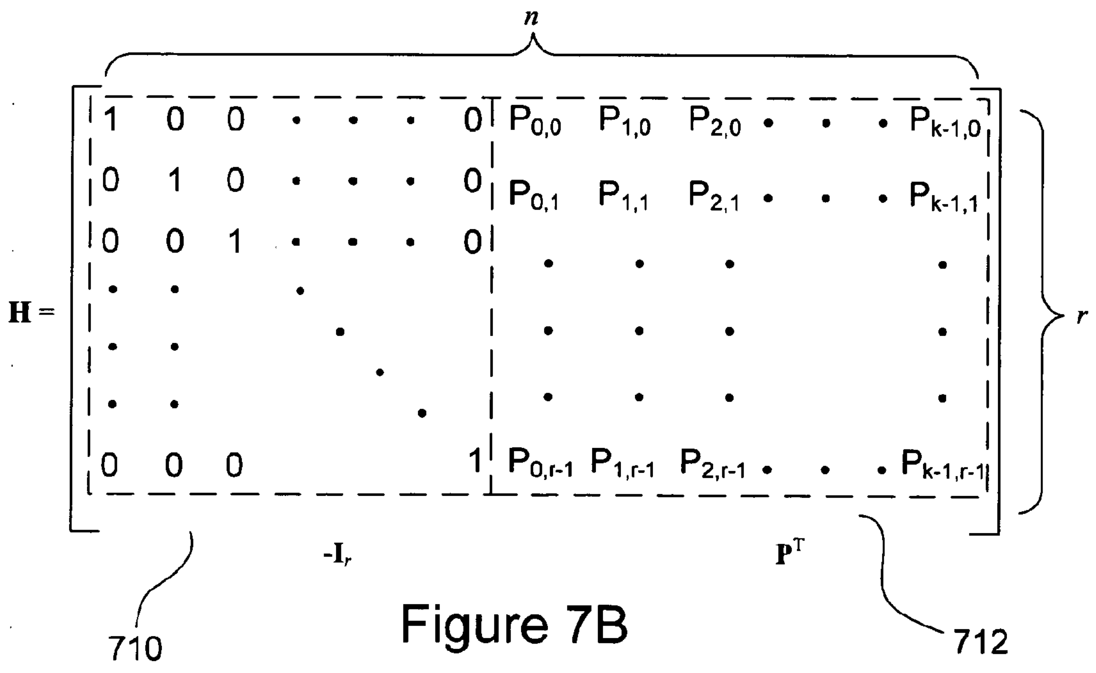

n
r
H =
-I_r
P^T
710
712
$P_{0,0}$ $P_{1,0}$ $P_{2,0}$ $P_{k-1,0}$
$P_{0,1}$ $P_{1,1}$ $P_{2,1}$ $P_{k-1,1}$
$P_{0,r-1}$ $P_{1,r-1}$ $P_{2,r-1}$ $P_{k-1,r-1}$

Figure 7B

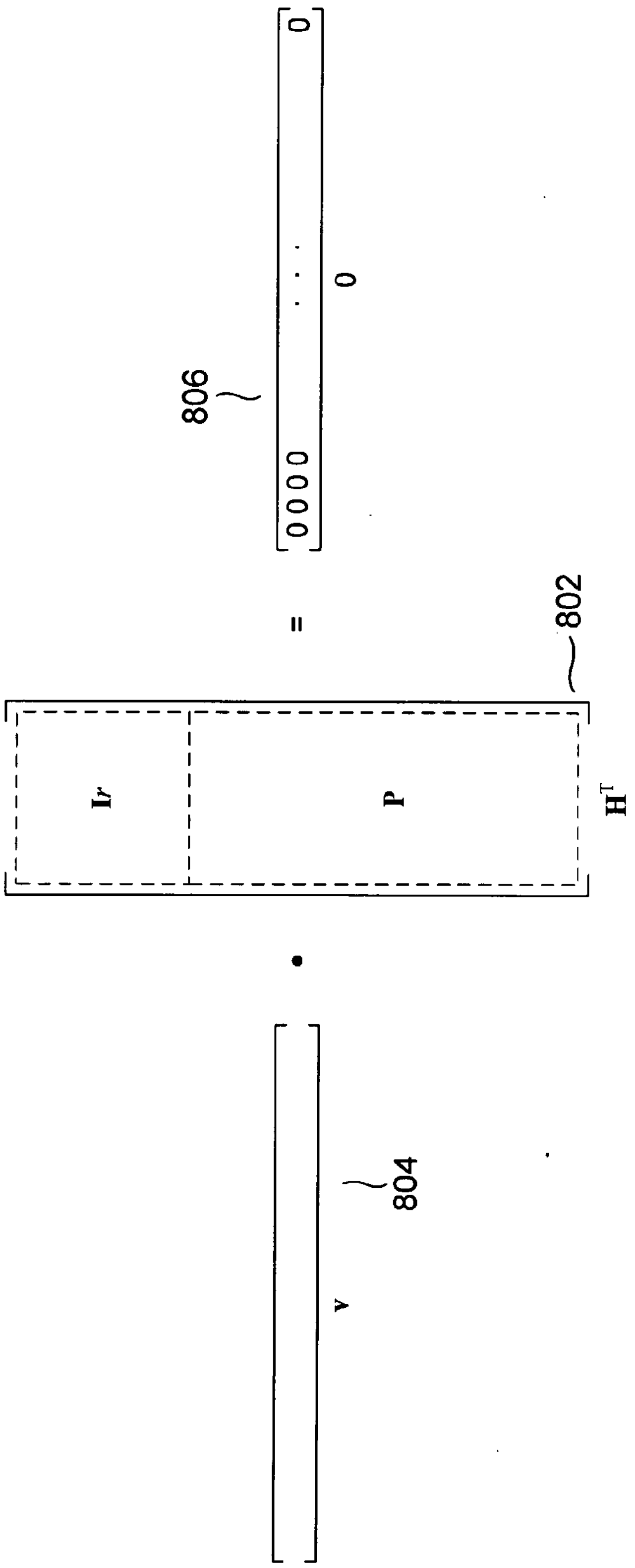


Figure 8

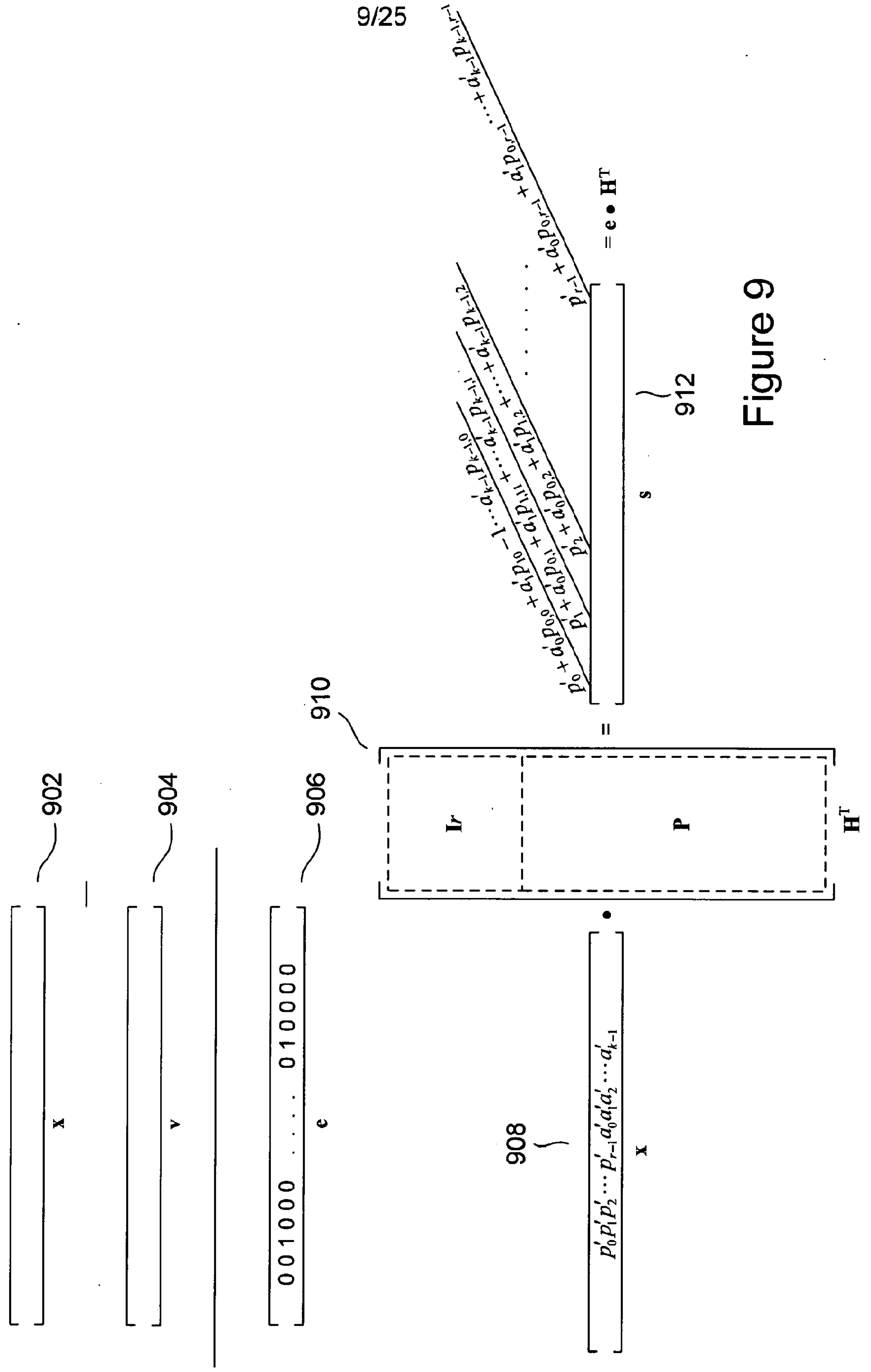
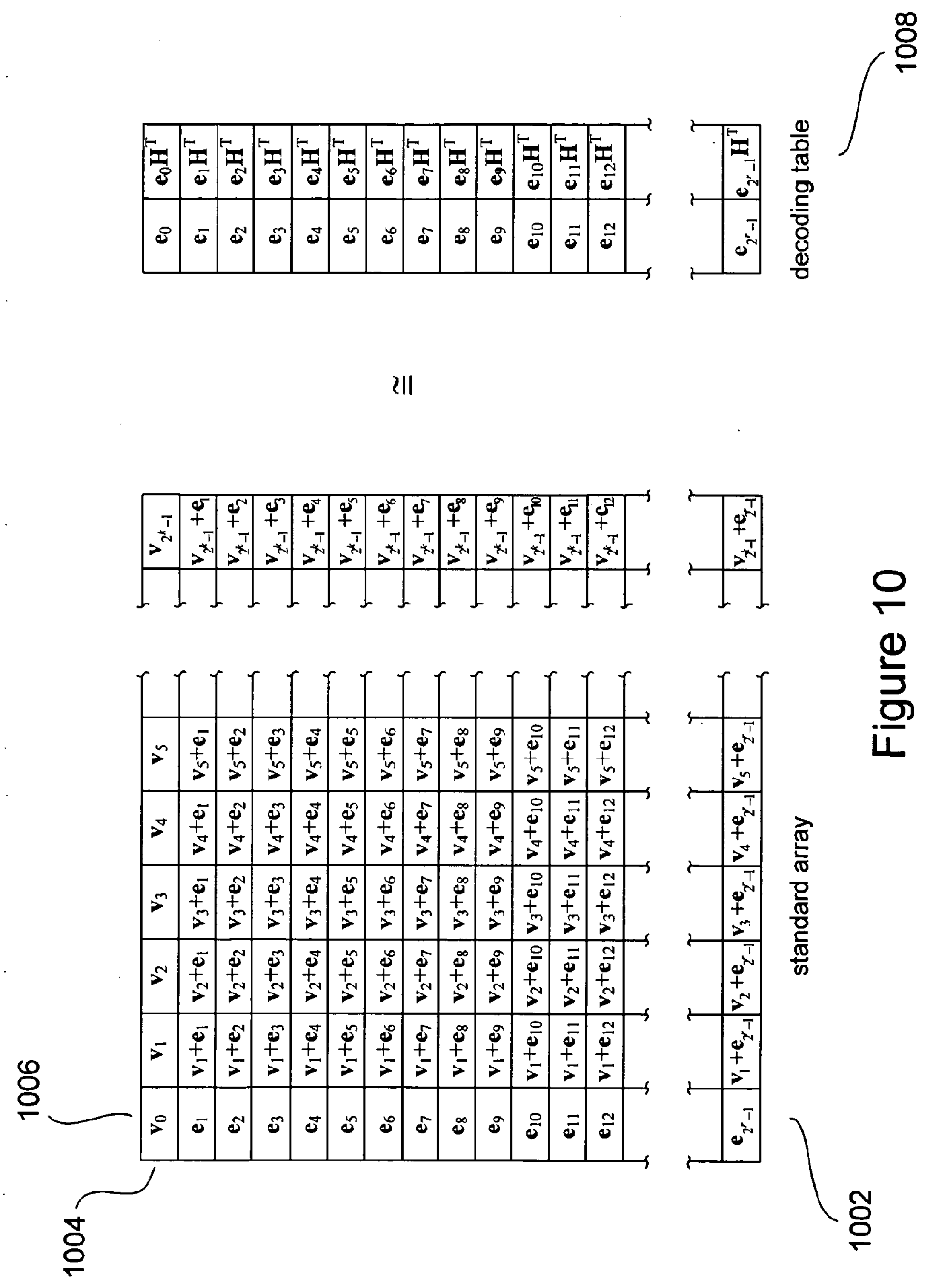

902
x
904
v
906
0 0 1 0 0 0 . . . . 0 1 0 0 0 0
e
908
$p'_0 p'_1 p'_2 \cdots p'_{r-1} a'_0 a'_1 a'_2 \cdots a'_{k-1}$
x
910
$\mathbf{I}_r$
$\mathbf{P}$
$\mathbf{H}^T$
$p'_0 + a'_0 p_{0,0} + a'_1 p_{1,0} - 1 \cdots a'_{k-1} p_{k-1,0}$
$p'_1 + a'_0 p_{0,1} + a'_1 p_{1,1} + \cdots a'_{k-1} p_{k-1,1}$
$p'_2 + a'_0 p_{0,2} + a'_1 p_{1,2} + \cdots + a'_{k-1} p_{k-1,2}$
$p'_{r-1} + a'_0 p_{0,r-1} + a'_1 p_{0,r-1} \cdots + a'_{k-1} p_{k-1,r-1}$
$= \mathbf{e} \bullet \mathbf{H}^T$
s
912

Figure 9



v_0	v_1	v_2	v_3	v_4	v_5	…	v_{2^k-1}
e_1	v_1+e_1	v_2+e_1	v_3+e_1	v_4+e_1	v_5+e_1	…	$v_{2^k-1}+e_1$
e_2	v_1+e_2	v_2+e_2	v_3+e_2	v_4+e_2	v_5+e_2	…	$v_{2^k-1}+e_2$
e_3	v_1+e_3	v_2+e_3	v_3+e_3	v_4+e_3	v_5+e_3	…	$v_{2^k-1}+e_3$
e_4	v_1+e_4	v_2+e_4	v_3+e_4	v_4+e_4	v_5+e_4	…	$v_{2^k-1}+e_4$
e_5	v_1+e_5	v_2+e_5	v_3+e_5	v_4+e_5	v_5+e_5	…	$v_{2^k-1}+e_5$
e_6	v_1+e_6	v_2+e_6	v_3+e_6	v_4+e_6	v_5+e_6	…	$v_{2^k-1}+e_6$
e_7	v_1+e_7	v_2+e_7	v_3+e_7	v_4+e_7	v_5+e_7	…	$v_{2^k-1}+e_7$
e_8	v_1+e_8	v_2+e_8	v_3+e_8	v_4+e_8	v_5+e_8	…	$v_{2^k-1}+e_8$
e_9	v_1+e_9	v_2+e_9	v_3+e_9	v_4+e_9	v_5+e_9	…	$v_{2^k-1}+e_9$
e_{10}	v_1+e_{10}	v_2+e_{10}	v_3+e_{10}	v_4+e_{10}	v_5+e_{10}	…	$v_{2^k-1}+e_{10}$
e_{11}	v_1+e_{11}	v_2+e_{11}	v_3+e_{11}	v_4+e_{11}	v_5+e_{11}	…	$v_{2^k-1}+e_{11}$
e_{12}	v_1+e_{12}	v_2+e_{12}	v_3+e_{12}	v_4+e_{12}	v_5+e_{12}	…	$v_{2^k-1}+e_{12}$
$e_{2^\gamma-1}$	$v_1+e_{2^\gamma-1}$	$v_2+e_{2^\gamma-1}$	$v_3+e_{2^\gamma-1}$	$v_4+e_{2^\gamma-1}$	$v_5+e_{2^\gamma-1}$	…	$v_{2^k-1}+e_{2^\gamma-1}$

standard array

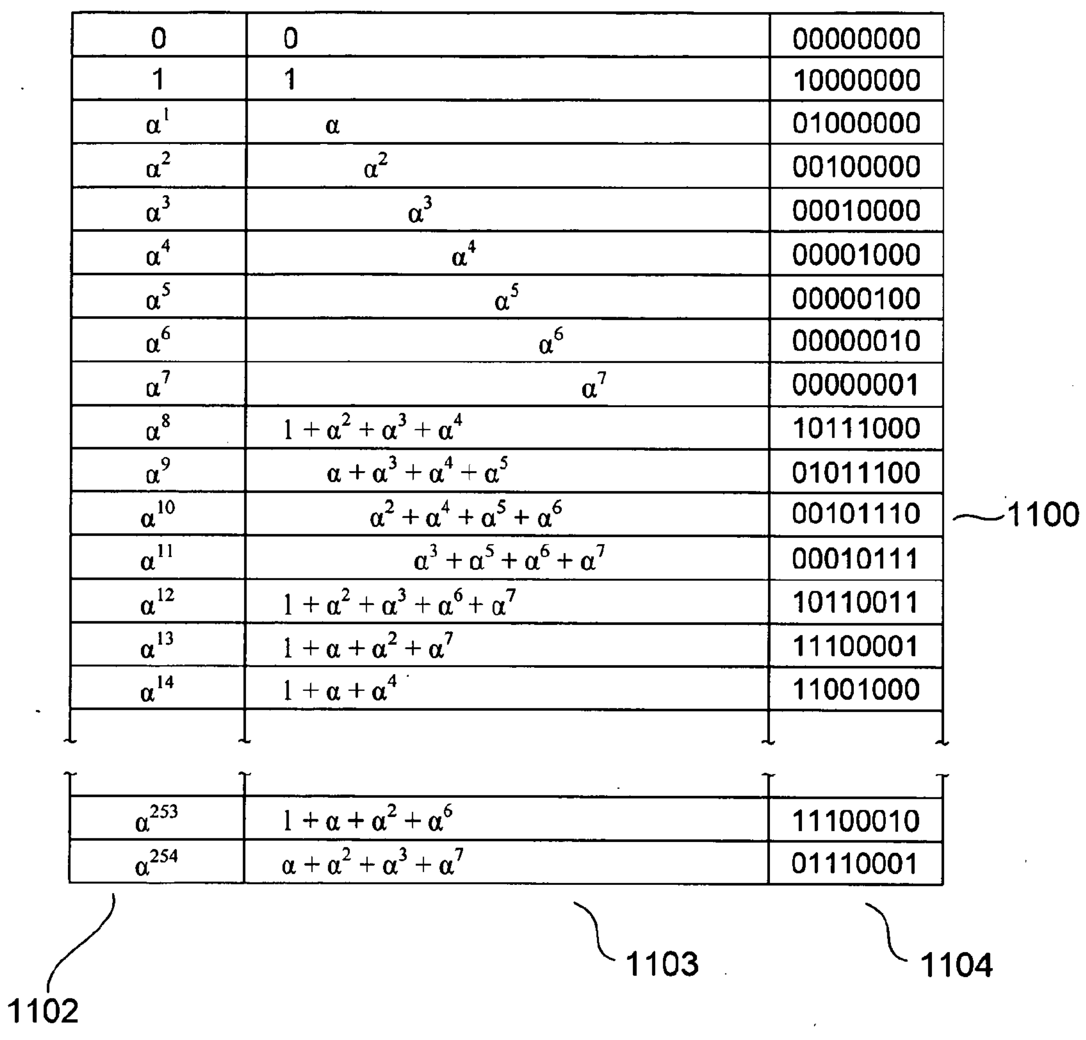
$\cong$

e_0	e_0H^T
e_1	e_1H^T
e_2	e_2H^T
e_3	e_3H^T
e_4	e_4H^T
e_5	e_5H^T
e_6	e_6H^T
e_7	e_7H^T
e_8	e_8H^T
e_9	e_9H^T
e_{10}	$e_{10}H^T$
e_{11}	$e_{11}H^T$
e_{12}	$e_{12}H^T$
$e_{2^\gamma-1}$	$e_{2^\gamma-1}H^T$

decoding table

Figure 10

GF(2^8)


1100
1102
1103
1104

0	0	00000000
1	1	10000000
α^1	α	01000000
α^2	α^2	00100000
α^3	α^3	00010000
α^4	α^4	00001000
α^5	α^5	00000100
α^6	α^6	00000010
α^7	α^7	00000001
α^8	$1 + \alpha^2 + \alpha^3 + \alpha^4$	10111000
α^9	$\alpha + \alpha^3 + \alpha^4 + \alpha^5$	01011100
α^{10}	$\alpha^2 + \alpha^4 + \alpha^5 + \alpha^6$	00101110
α^{11}	$\alpha^3 + \alpha^5 + \alpha^6 + \alpha^7$	00010111
α^{12}	$1 + \alpha^2 + \alpha^3 + \alpha^6 + \alpha^7$	10110011
α^{13}	$1 + \alpha + \alpha^2 + \alpha^7$	11100001
α^{14}	$1 + \alpha + \alpha^4$	11001000
α^{253}	$1 + \alpha + \alpha^2 + \alpha^6$	11100010
α^{254}	$\alpha + \alpha^2 + \alpha^3 + \alpha^7$	01110001

Figure 11

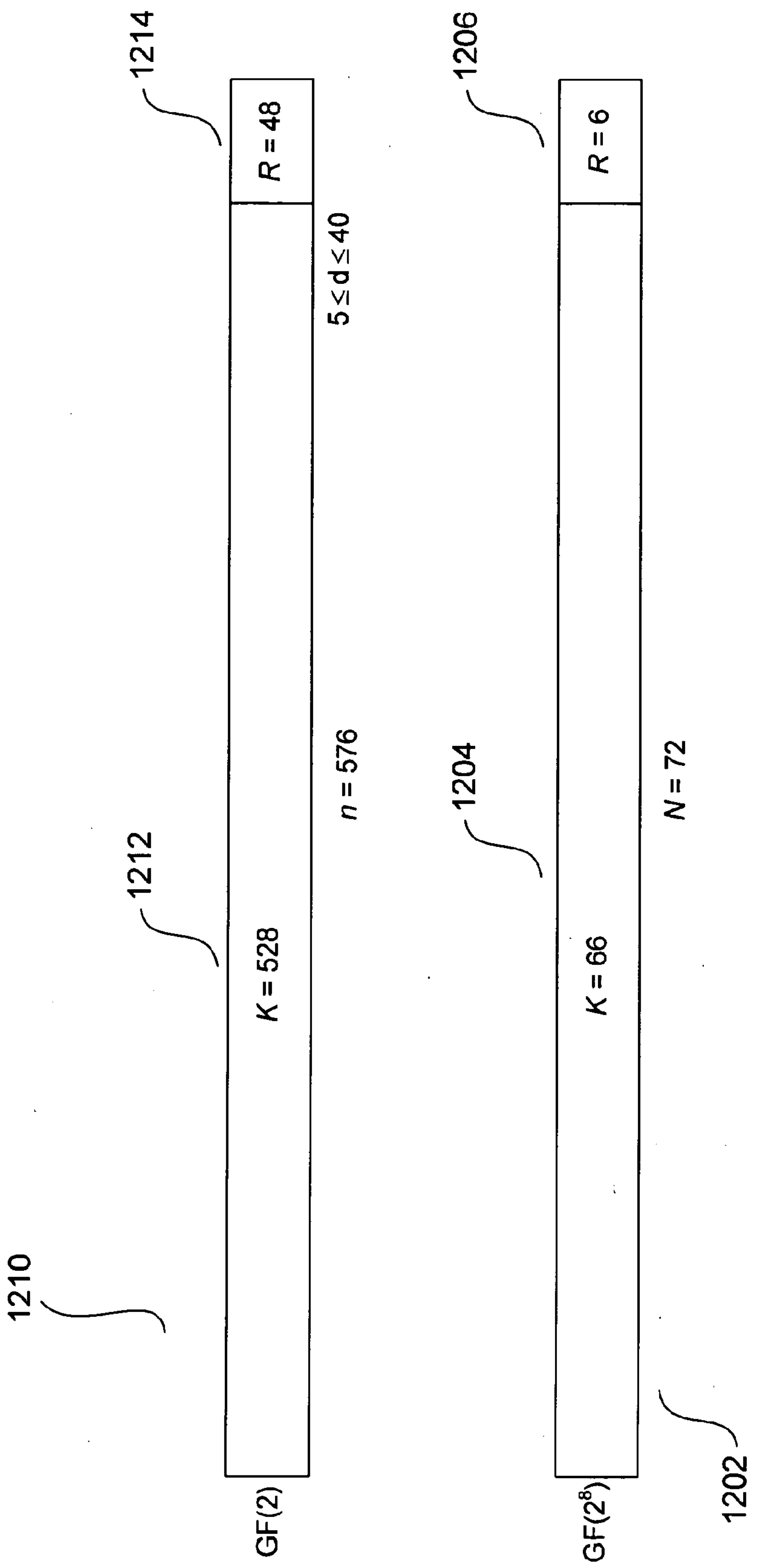
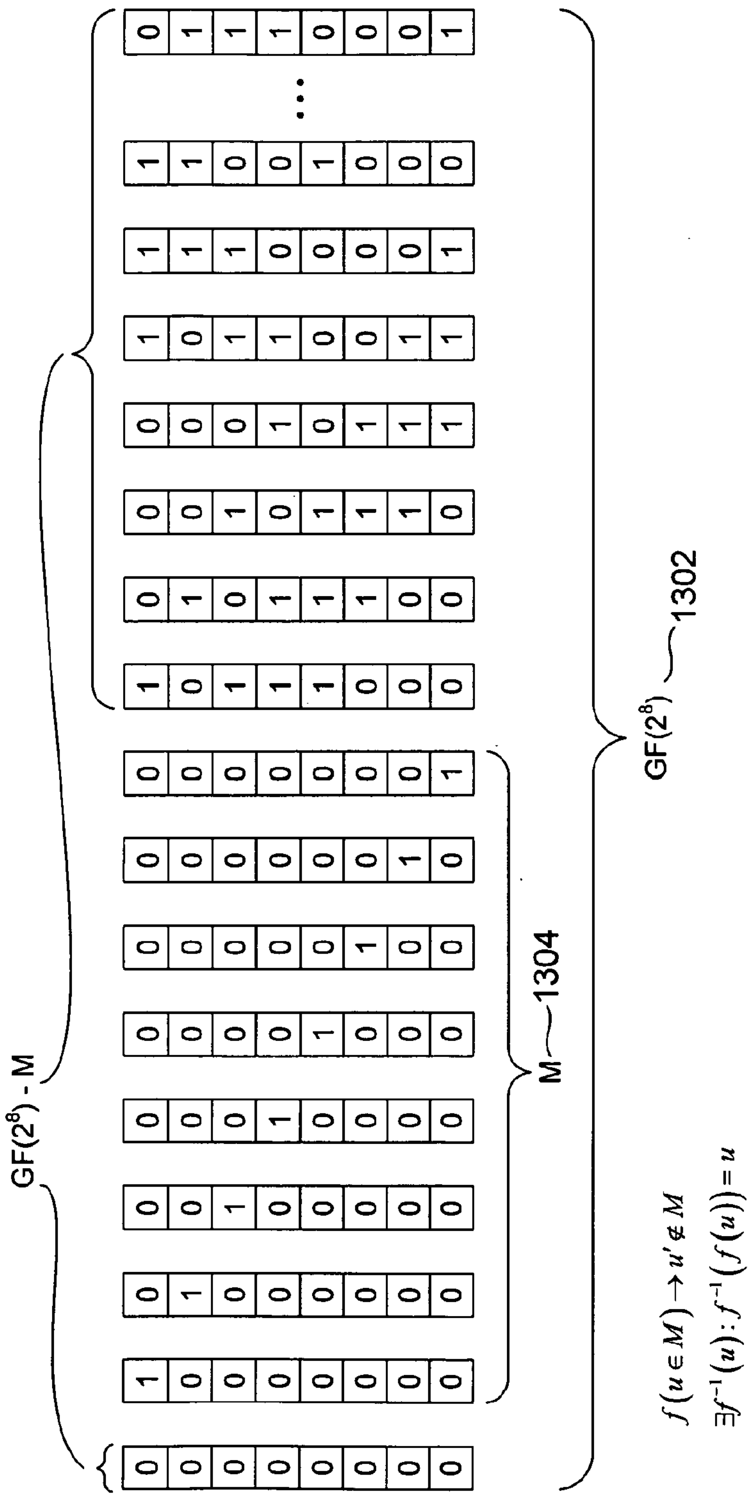

1214
R = 48
5 ≤ d ≤ 40
n = 576
1212
K = 528
1210
GF(2)
1206
R = 6
1204
N = 72
K = 66
1202
GF(2^8)

Figure 12



$f(u \in M) \to u' \notin M$

$\exists f^{-1}(u): f^{-1}(f(u)) = u$

Figure 13

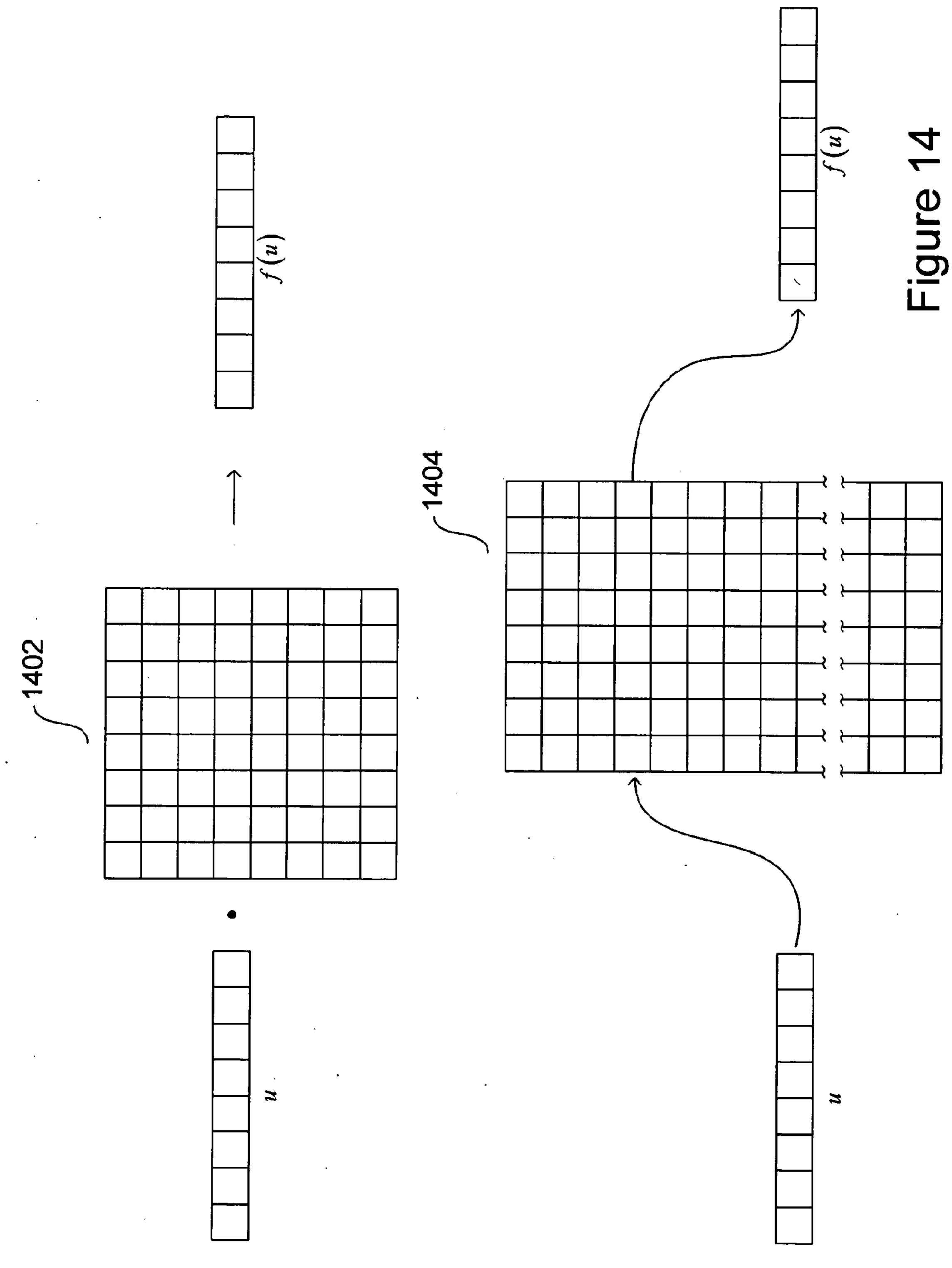


Figure 14

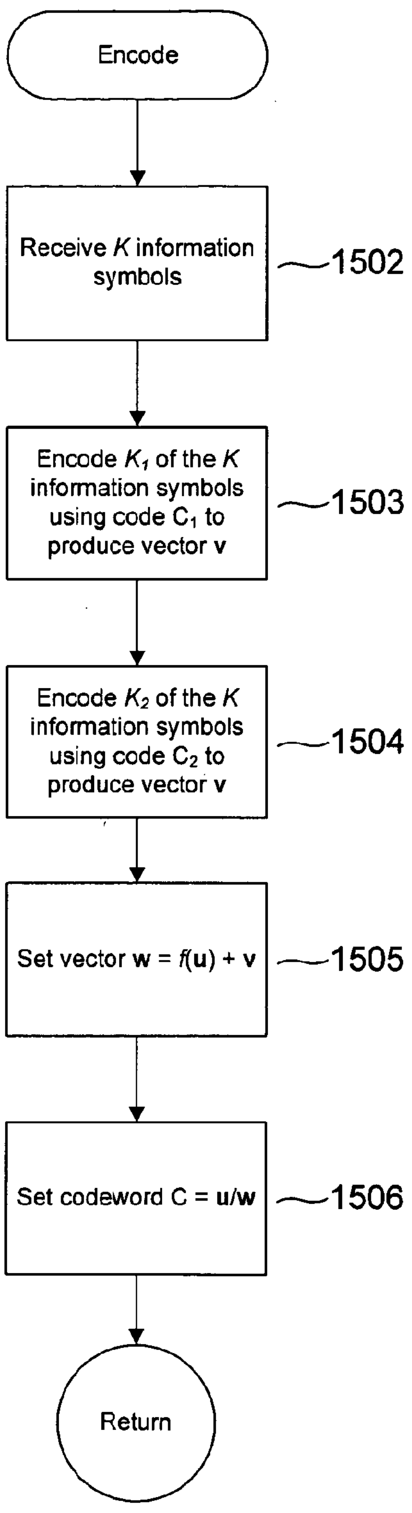

Encode
Receive K information symbols
1502
Encode K_1 of the K information symbols using code C_1 to produce vector **v**
1503
Encode K_2 of the K information symbols using code C_2 to produce vector **v**
1504
Set vector **w** = f(**u**) + **v**
1505
Set codeword C = **u**/**w**
1506
Return

Figure 15A

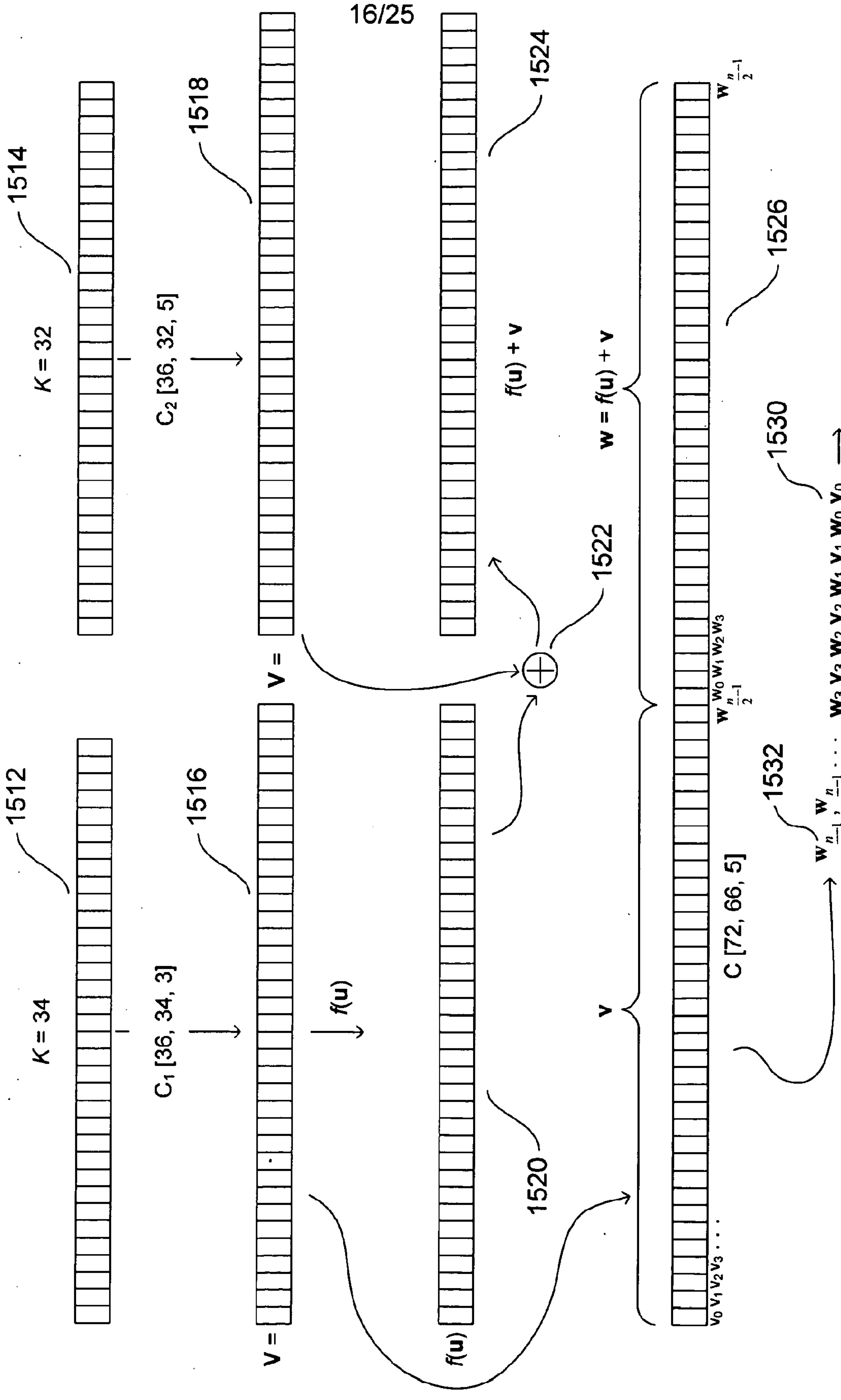

1512
1514
K = 34
K = 32
C_1 [36, 34, 3]
C_2 [36, 32, 5]
1516
1518
v =
v =
f(u)
f(u)
1520
1522
f(u) + v
1524
v
w = f(u) + v
C [72, 66, 5]
1526
v_0 v_1 v_2 v_3 . . .
w_{n/2-1} w_0 w_1 w_2 w_3
w_{n/2-1}
1532
w_{n/2-1}, w_{n/2-1} . . .
w_3 v_3 w_2 v_2 w_1 v_1 w_0 v_0
1530
1528

Figure 15B

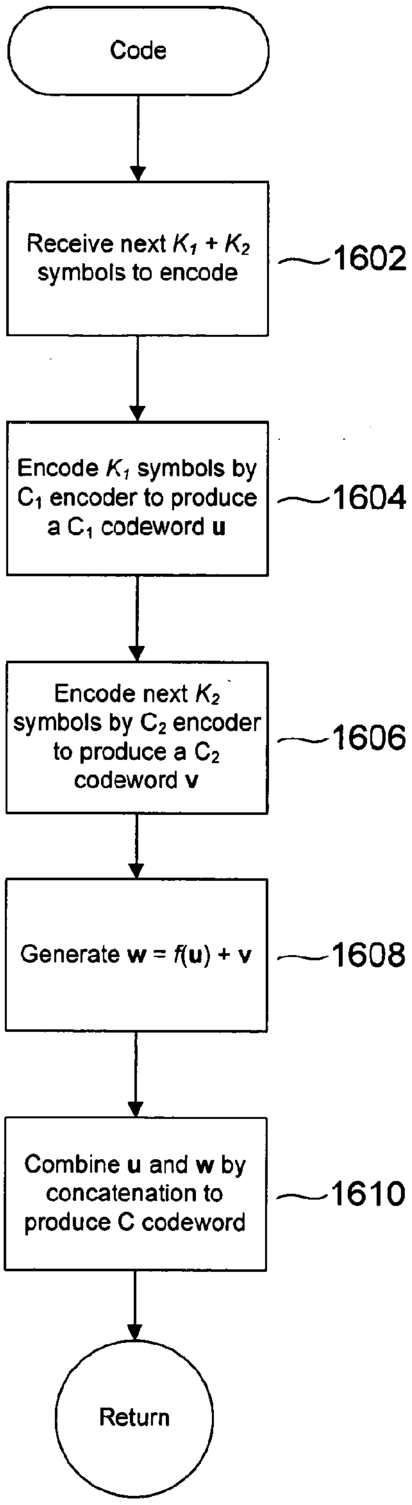

Code
Receive next $K_1 + K_2$ symbols to encode
1602
Encode K_1 symbols by C_1 encoder to produce a C_1 codeword **u**
1604
Encode next K_2 symbols by C_2 encoder to produce a C_2 codeword **v**
1606
Generate **w** = f(**u**) + **v**
1608
Combine **u** and **w** by concatenation to produce C codeword
1610
Return

Figure 16

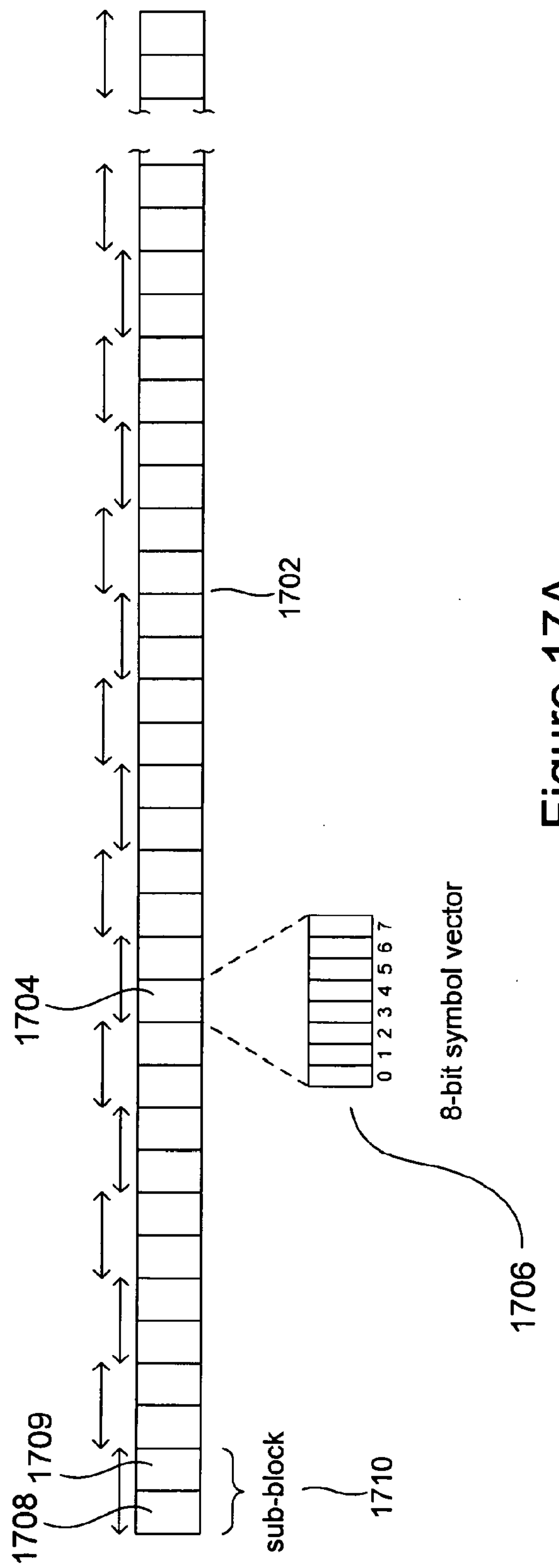

1702
1704
1706
1708
1709
1710
sub-block
8-bit symbol vector
0 1 2 3 4 5 6 7

Figure 17A

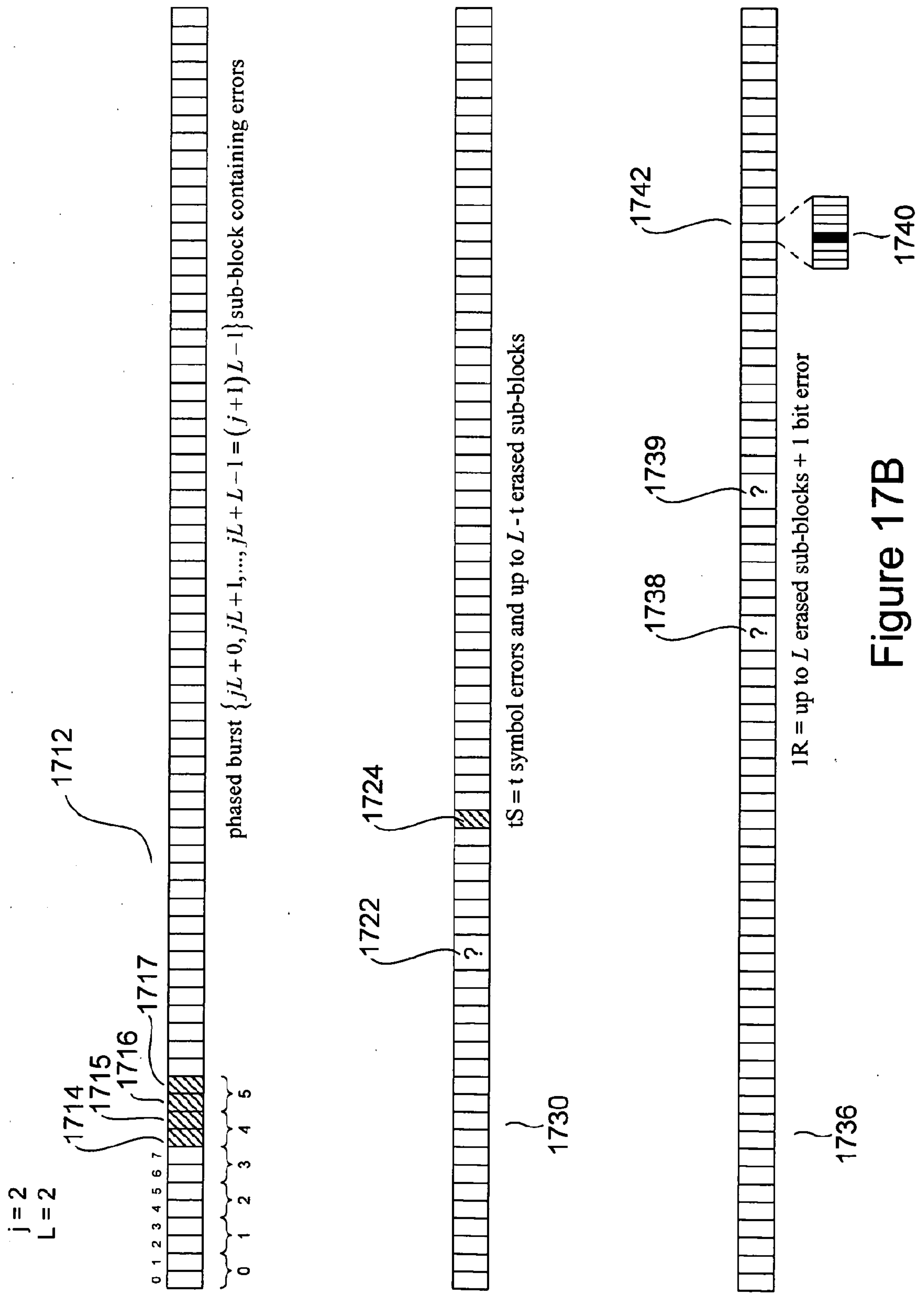

j = 2
L = 2
0 1 2 3 4 5 6 7
0 1 2 3 4 5
1714
1715
1716
1717
1712
phased burst $\{jL+0, jL+1, \ldots, jL+L-1 = (j+1)L-1\}$ sub-block containing errors
?
1722
1724
1730
tS = t symbol errors and up to L - t erased sub-blocks
?
?
1738
1739
1742
1740
1736
1R = up to L erased sub-blocks + 1 bit error

Figure 17B

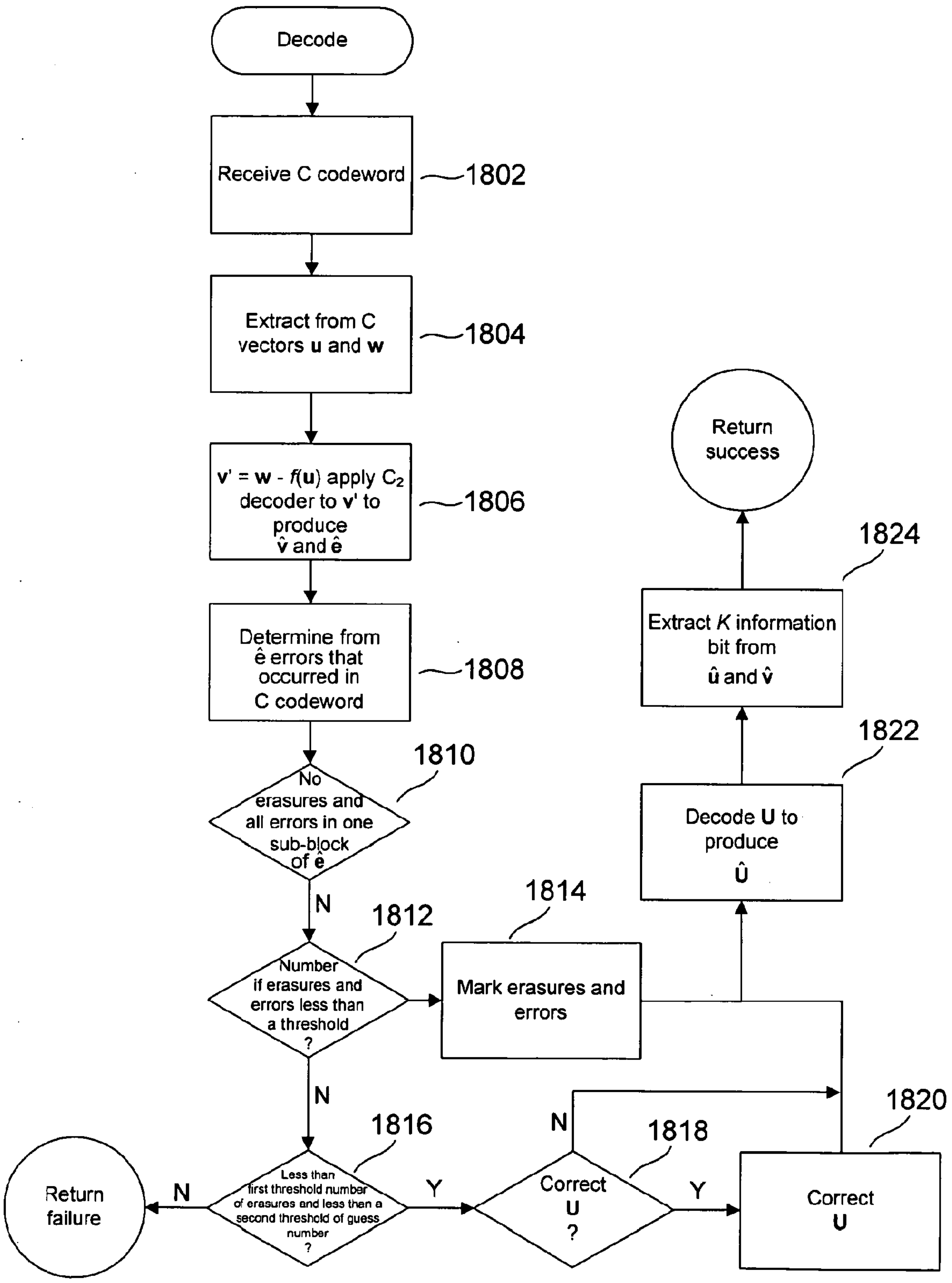

Decode
Receive C codeword
1802
Extract from C vectors **u** and **w**
1804
v' = **w** - *f*(**u**) apply C_2 decoder to **v**' to produce $\hat{v}$ and $\hat{e}$
1806
Determine from $\hat{e}$ errors that occurred in C codeword
1808
No erasures and all errors in one sub-block of $\hat{e}$
1810
N
Number if erasures and errors less than a threshold ?
1812
Mark erasures and errors
1814
N
Less than first threshold number of erasures and less than a second threshold of guess number ?
1816
N
Return failure
Y
Correct **U** ?
1818
N
Y
Correct **U**
1820
Decode **U** to produce $\hat{U}$
1822
Extract *K* information bit from $\hat{u}$ and $\hat{v}$
1824
Return success

Figure 18

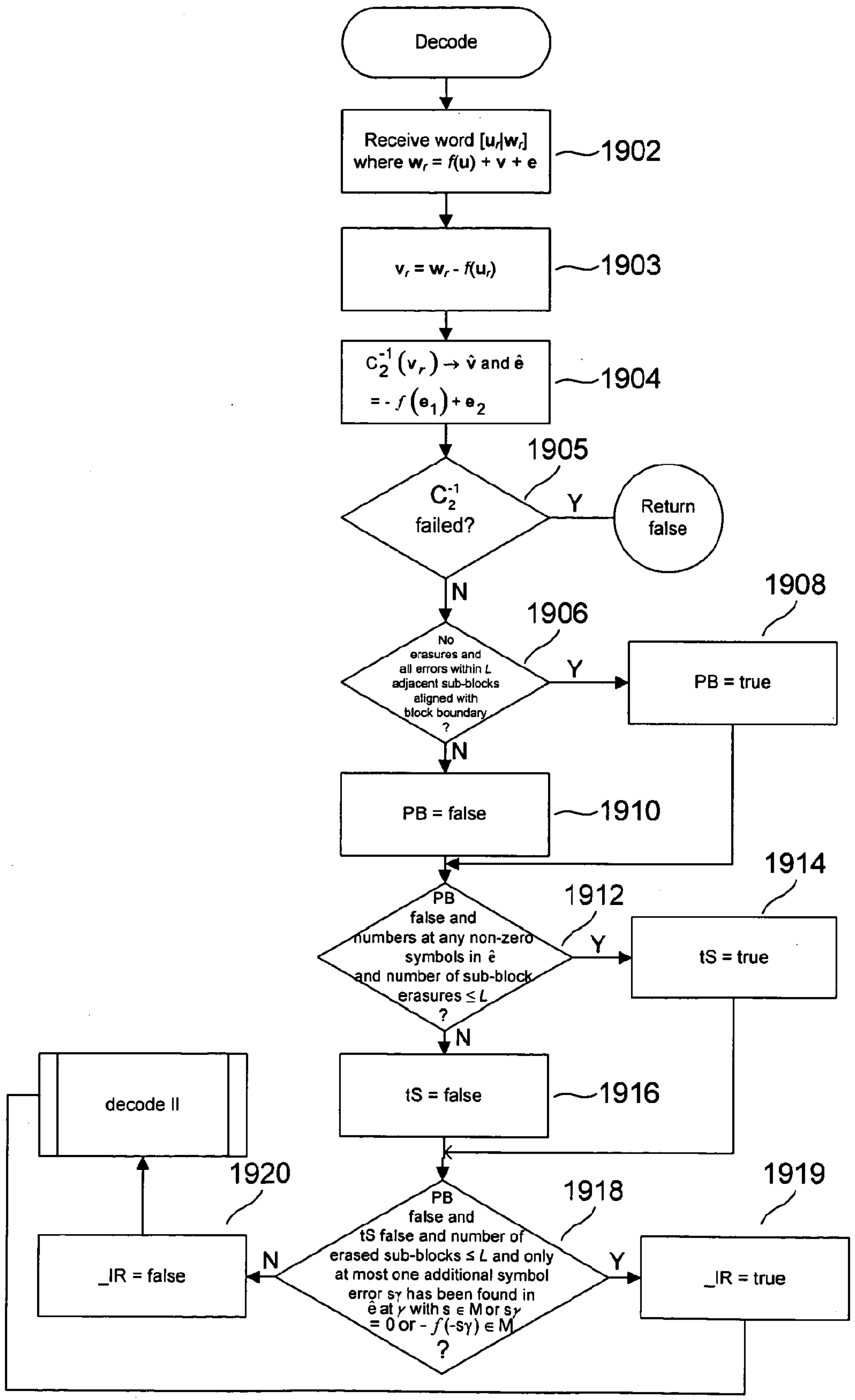

Decode
Receive word $[\mathbf{u}_r|\mathbf{w}_r]$ where $\mathbf{w}_r = f(\mathbf{u}) + \mathbf{v} + \mathbf{e}$
1902
$\mathbf{v}_r = \mathbf{w}_r - f(\mathbf{u}_r)$
1903
$C_2^{-1}(v_r) \rightarrow \hat{v}$ and $\hat{e} = -f(e_1) + e_2$
1904
1905
C_2^{-1} failed?
Y
Return false
N
1906
No erasures and all errors within L adjacent sub-blocks aligned with block boundary ?
Y
1908
PB = true
N
PB = false
1910
1912
PB false and numbers at any non-zero symbols in $\hat{e}$ and number of sub-block erasures $\leq L$?
Y
1914
tS = true
N
tS = false
1916
decode II
1920
1918
PB false and tS false and number of erased sub-blocks $\leq L$ and only at most one additional symbol error $s\gamma$ has been found in $\hat{e}$ at γ with $s \in M$ or $s\gamma = 0$ or $-f(-s\gamma) \in M$?
N
_IR = false
Y
1919
_IR = true

Figure 19

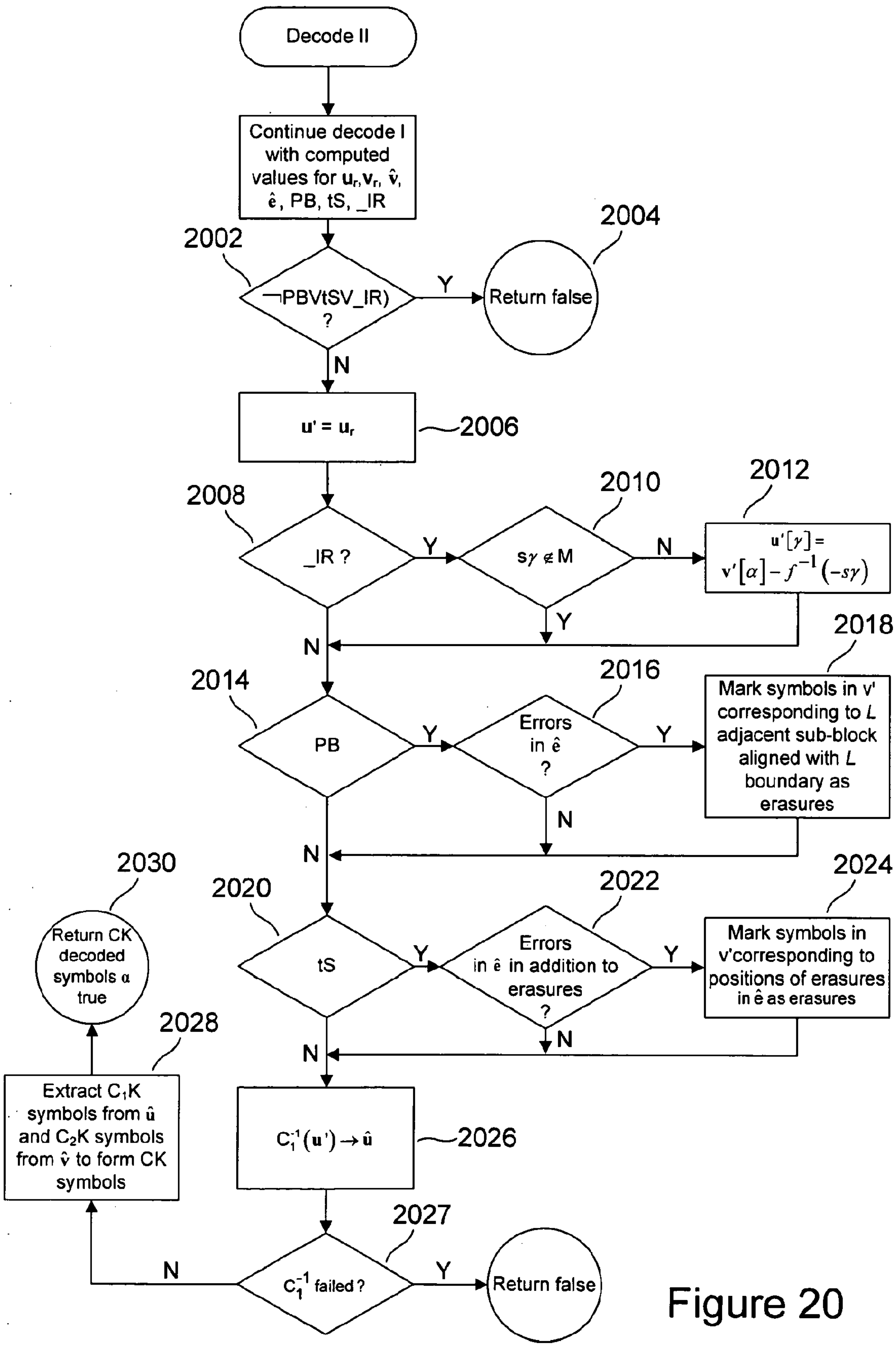

Decode II
Continue decode I with computed values for $u_r, v_r, \hat{v}$, $\hat{e}$, PB, tS, _IR
2002
$\neg$PBVtSV_IR) ?
Y
N
2004
Return false
$u' = u_r$
2006
2008
_IR ?
Y
N
2010
$s\gamma \notin M$
N
Y
2012
$u'[\gamma] = v'[\alpha] - f^{-1}(-s\gamma)$
2014
PB
Y
N
2016
Errors in $\hat{e}$?
Y
N
2018
Mark symbols in v' corresponding to L adjacent sub-block aligned with L boundary as erasures
2020
tS
Y
N
2022
Errors in $\hat{e}$ in addition to erasures ?
Y
N
2024
Mark symbols in v'corresponding to positions of erasures in $\hat{e}$ as erasures
$C_1^{-1}(u') \rightarrow \hat{u}$
2026
2027
C_1^{-1} failed ?
Y
N
Return false
2028
Extract C_1K symbols from $\hat{u}$ and C_2K symbols from $\hat{v}$ to form CK symbols
2030
Return CK decoded symbols α true

Figure 20

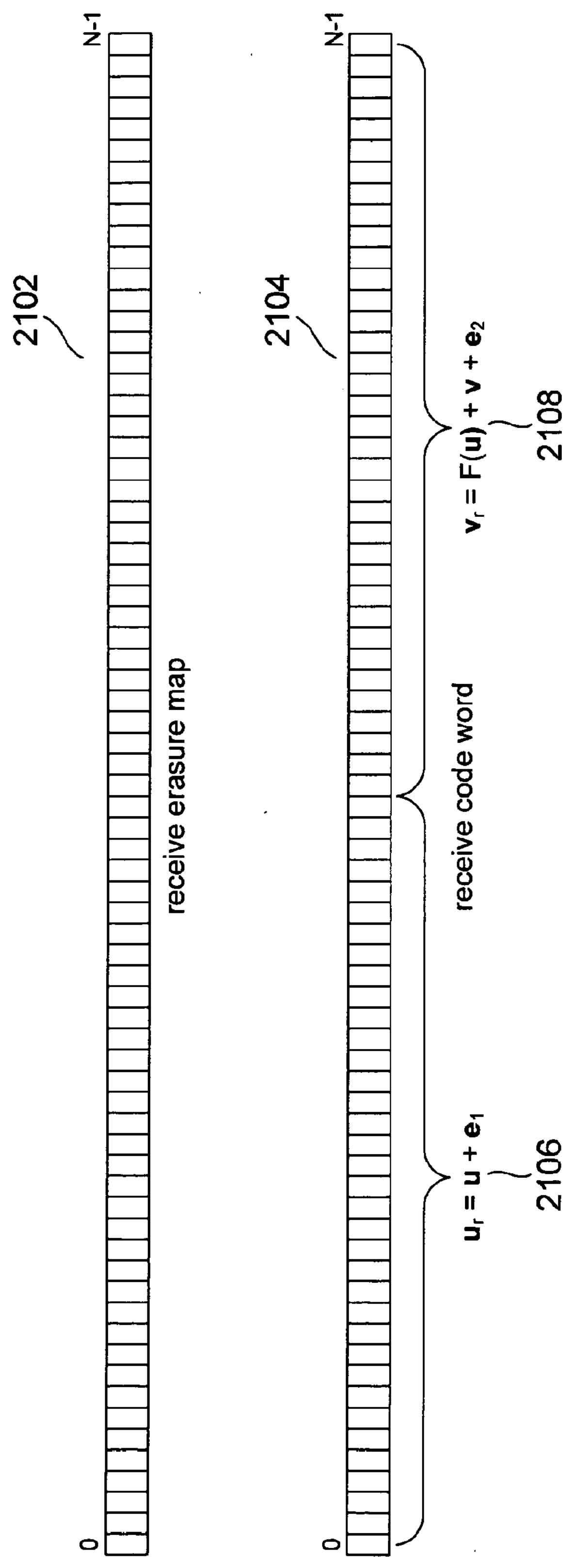

2102
N-1
0
receive erasure map
2104
N-1
0
receive code word
$u_r = u + e_1$
2106
$v_r = F(u) + v + e_2$
2108

Figure 21

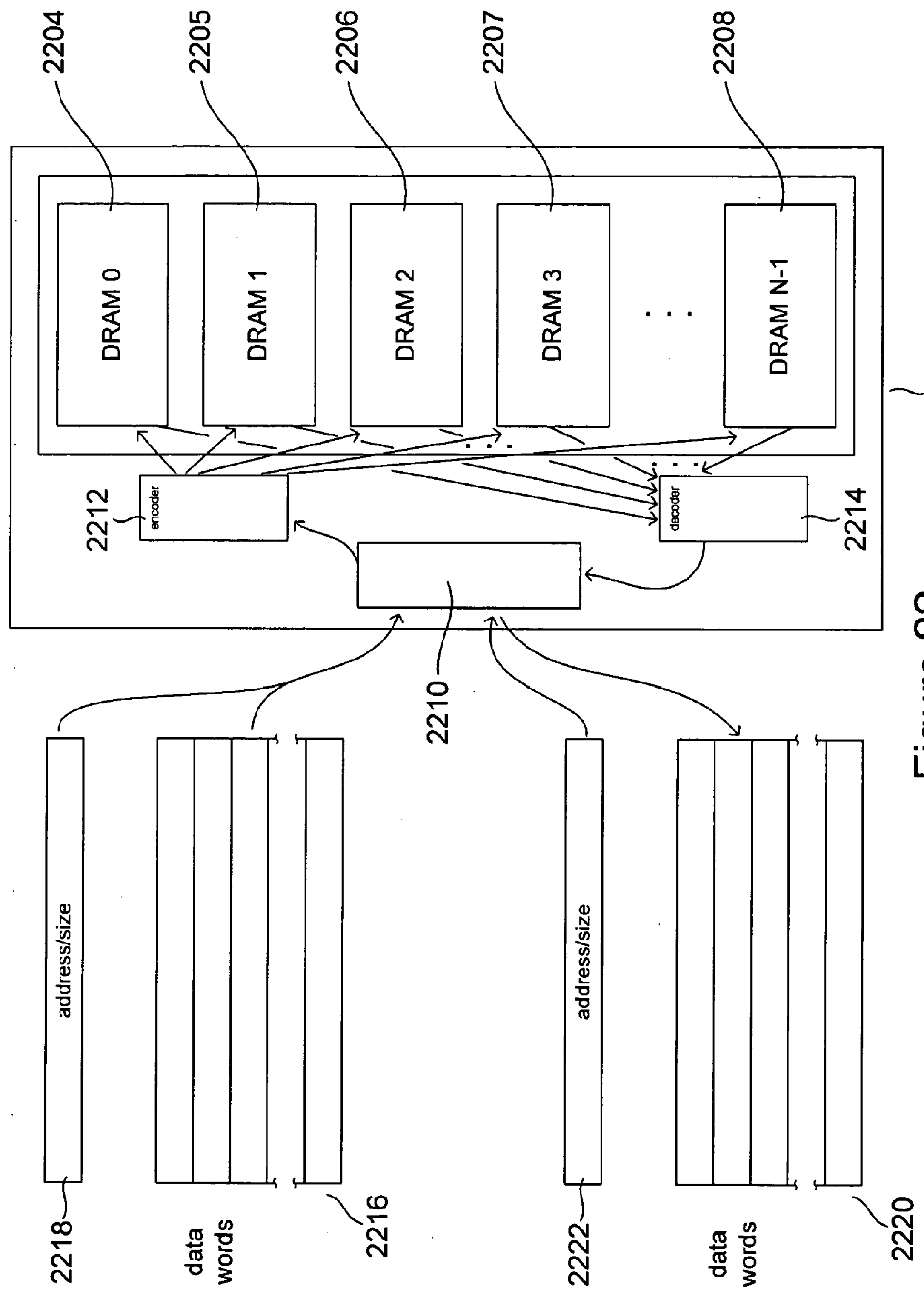

2204
2205
2206
2207
2208
DRAM 0
DRAM 1
DRAM 2
DRAM 3
. . .
DRAM N-1
2202
2212
encoder
decoder
2214
2210
address/size
data words
2218
2216
address/size
data words
2222
2220

Figure 22

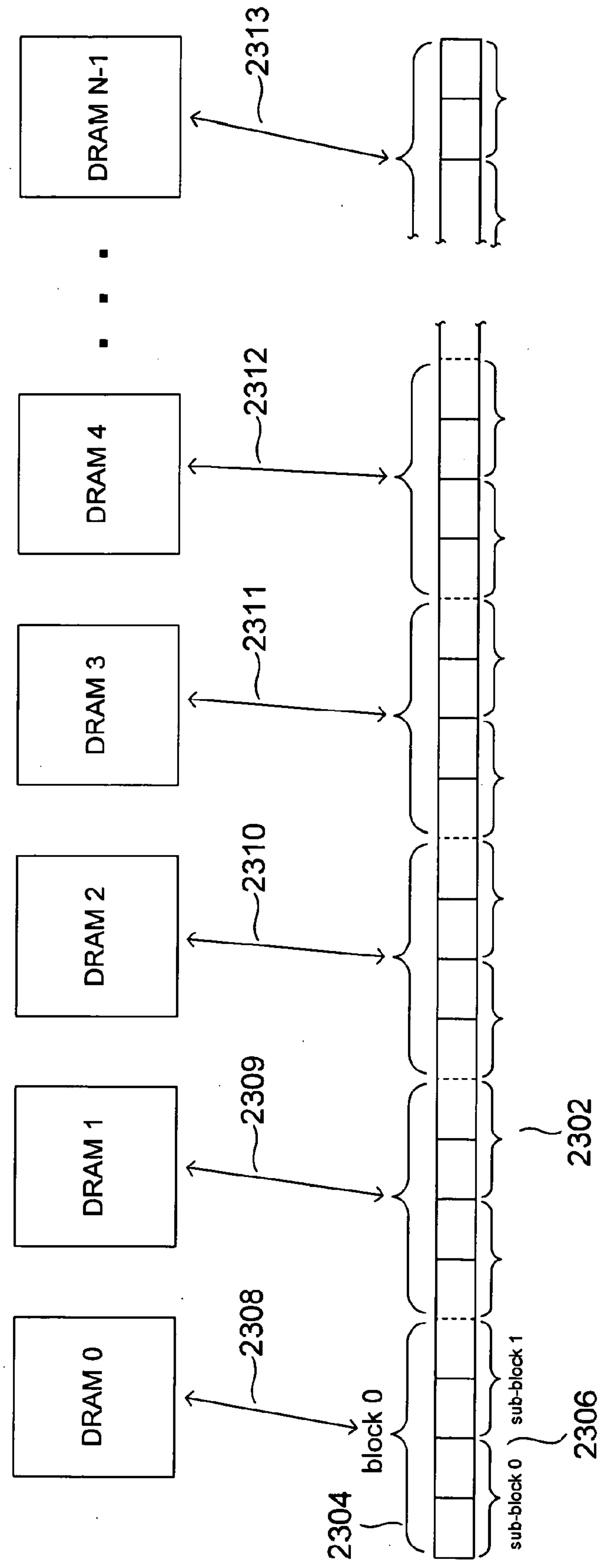

DRAM 0
DRAM 1
DRAM 2
DRAM 3
DRAM 4
DRAM N-1
2308
2309
2310
2311
2312
2313
2302
2304
2306
block 0
sub-block 0
sub-block 1

Figure 23

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2008/002836

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/00(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 8, G06F 11/00, G06F 11/10, H03M 13/00

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
WPI, e-Korean Intellectual Property Office Patent Search System.
Keywords : encode, information, symbol, length, and composite

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2002/0099996 A1 (MASAYUKI DEMURA, HIRONOBU NAGURA, TETSUYA TAMURA, KEISUKE TANAKA(JP)) Jul.25,2002 Abstract The paragraphs [0030]-[0071] in the detailed description Fig.1 - fig.10 Claims 1,2,5,12	1 - 20
A	US 2007/0011598 A1 (Hitachi Global Storage Technologies Netherlands B.V.(NL)) Jan.11,2007 Abstract The paragraphs [0027]-[0045] in the detailed description Fig.1A - fig.4	1 - 20
A	EP 0 793 174 B1 (SUN MICROSYSTEMS, INC.(US)) May.07,2003 The paragraphs [0028]-[0031] in the detailed description Fig.1 and fig.2	1 - 20

☐ Further documents are listed in the continuation of Box C. ☒ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents,such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search 23 JUNE 2008 (23.06.2008)	Date of mailing of the international search report **23 JUNE 2008 (23.06.2008)**
Name and mailing address of the ISA/KR Korean Intellectual Property Office Government Complex-Daejeon, 139 Seonsa-ro, Seo-gu, Daejeon 302-701, Republic of Korea Facsimile No. 82-42-472-7140	Authorized officer PARK, Sung Ho Telephone No. 82-42-481-5743

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/US2008/002836

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2002-0099996 A1	25.07.2002	CN 1225491 A	11.08.1999
		JP 11-274941 A2	08.10.1999
		JP 3165099 B2	14.05.2001
		KR 10-1999-0072241	27.09.1999
		TW 451185 A	21.08.2001
		US 6553533 BB	22.04.2003
US 2007-011598 A1	11.01.2007	CN 1881477 A	20.12.2006
EP 0793174 B1	07.05.2003	JP 10-031628 A2	03.02.1998
		SG 76501 A1	21.11.2000
		US 5781568 A	14.07.1998