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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 April 2009 (30.04.2009)

PCT

(10) International Publication Number
WO 2009/053961 A2

(51) International Patent Classification:
G06F 7/74 (2006.01)

(21) International Application Number:
PCT/IL2008/001228

(22) International Filing Date:
17 September 2008 (17.09.2008)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/996,027	25 October 2007 (25.10.2007)	US
60/996,782	5 December 2007 (05.12.2007)	US
61/006,805	31 January 2008 (31.01.2008)	US
61/064,853	31 March 2008 (31.03.2008)	US
61/071,465	30 April 2008 (30.04.2008)	US
61/071,466	30 April 2008 (30.04.2008)	US
61/129,608	8 July 2008 (08.07.2008)	US

(71) Applicant *(for all designated States except US)*: **DENSBITS TECHNOLOGIES LTD.** [IL/IL]; Building 30, Flr 3, Matam Industrial Park, P.O.Box 15111, 31015 Haifa (IL).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **WEINGARTEN, Hanan** [IL/IL]; Hayarden 2, 46377 Herzliya (IL). **KATZ, Michael** [IL/IL]; 75 Moriah Ave., 34616 Haifa (IL).

(74) Agent: **PEARL COHEN ZEDEK LATZER**; 5 Shenkar Street, P.O. Box 12704, Herzlia 46733 (IL).

(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, SK, SL, SM, ST, 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:
— *without international search report and to be republished upon receipt of that report*

(54) **Title:** SYSTEMS AND METHODS FOR MULTIPLE CODING RATES IN FLASH DEVICES

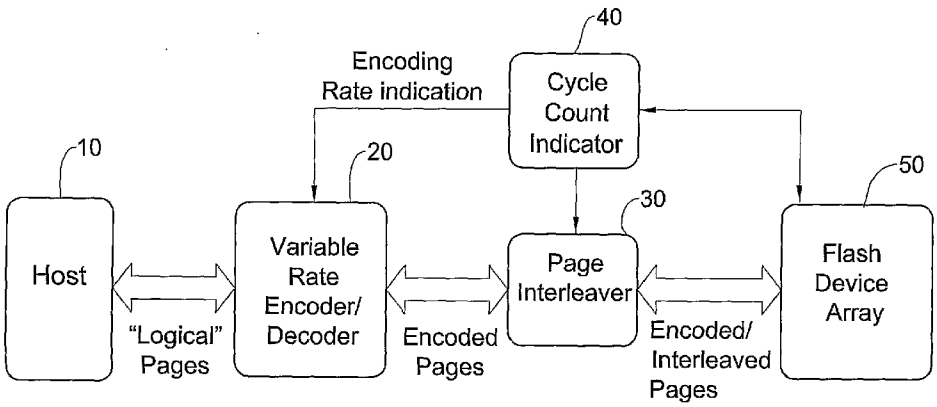


Figure 1

(57) **Abstract:** A system and method for encoding information arriving from a host in order to store the coded information in flash memory, the method comprising encoding information arriving from a host for storage at a flash memory location including generating a number of redundancy bytes, the encoding proceeding at an encoding rate which is a function of the number of redundancy bytes generated, the encoding including determining an effective error rate, including an anticipated rate of expected reading errors, for the flash memory location; and selecting the encoding rate as a function of the effective error rate such that the number of redundancy bytes is sufficient to overcome the anticipated rate of expected reading errors with a predetermined degree of certainty.

WO 2009/053961 A2

SYSTEMS AND METHODS FOR MULTIPLE CODING RATES IN FLASH DEVICES

REFERENCE TO CO-PENDING APPLICATIONS

Priority is claimed from the following co-pending applications: US Provisional Application No. 60/996,027, filed October 25, 2007 and entitled "Systems and Methods for Coping with Variable Bit Error Rates in Flash Devices", US Provisional Application No. 61/071,466, filed April 30, 2008 and entitled "Systems and Methods for Multiple Coding Rates in Flash Devices", US Provisional Application No. 60/996,782, filed December 5, 2007 and entitled "Systems and Methods for Using a Training Sequence in Flash Memory", US Provisional Application No. 61/064,853, filed March 31, 2008 and entitled "Flash Memory Device with Physical Cell Value Deterioration Accommodation and Methods Useful in Conjunction Therewith", US Provisional Application No. 61/006,805, filed January 31, 2008 and entitled "A Method for Extending the Life of Flash Devices", US Provisional Application No. 61/071,465, filed April 30, 2008 and entitled "Systems and Methods for Temporarily Retiring Memory Portions" and US Provisional Application No. 61/129,608, filed July 8, 2008 and entitled "A Method for Acquiring and Tracking Detection Thresholds in Flash Devices".

Other co-pending applications include: US Provisional Application No. 60/960,207, filed September 20, 2007 and entitled "Systems and Methods for Coupling Detection in Flash Memory", US Provisional Application No. 61/071,467, filed April 30, 2008 and entitled "Improved Systems and Methods for Determining Logical Values of Coupled Flash Memory Cells", US Provisional Application No. 60/960,943, filed October 22, 2007 and entitled "Systems and methods to reduce errors in Solid State Disks and Large Flash Devices" and US Provisional Application No. 61/071,469, filed April 30, 2008 and entitled "Systems and Methods for Averaging Error Rates in Non-Volatile Devices and Storage Systems", US Provisional Application No. 61/006,120, filed

December 19, 2007 and entitled "Systems and Methods for Coping with Multi Stage Decoding in Flash Devices", US Provisional Application No. 61/071,464, filed April 30, 2008 and entitled "A Decoder Operative to Effect A Plurality of Decoding Stages Upon Flash Memory Data and Methods Useful in Conjunction Therewith", US Provisional Application No. 61/006,385, filed January 10, 2008 and entitled "A System for Error Correction Encoder and Decoder Using the Lee Metric and Adapted to Work on Multi-Level Physical Media", US Provisional Application No. 61/064,995, filed April 8, 2008 and entitled "Systems and Methods for Error Correction and Decoding on Multi-Level Physical Media", US Provisional Application No. 60/996,948, filed December 12, 2007 and entitled "Low Power BCH/RS Decoding: a Low Power Chien-Search Implementation", US Provisional Application No. 61/071,487, filed May 1, 2008 and entitled "Chien-Search System Employing a Clock-Gating Scheme to Save Power for Error Correction Decoder and other Applications", US Provisional Application No. 61/071,468, filed April 30, 2008 and entitled "A Low Power Chien-Search Based BCH/RS Recoding System for Flash Memory, Mobile Communications Devices and Other Applications", US Provisional Application No. 61/006,806, filed January 31, 2008 and entitled "Systems and Methods for using a Erasure Coding in Flash memory", US Provisional Application No. 61/071,486, filed May 1, 2008 and entitled "Systems and Methods for Handling Immediate Data Errors in Flash Memory", US Provisional Application No. 61/006,078, filed December 18, 2007 and entitled "Systems and Methods for Multi Rate Coding in Multi Level Flash Devices", US Provisional Application No. 61/064,923, filed April 30, 2008 and entitled "Apparatus For Coding At A Plurality Of Rates In Multi-Level Flash Memory Systems, And Methods Useful In Conjunction Therewith", US Provisional Application No. 61/064,760, filed March 25, 2008 and entitled "Hardware efficient implementation of rounding in fixed-point arithmetic", US Provisional Application No. 61/071,404, filed April 28, 2008 and entitled "Apparatus and Methods for Hardware-Efficient Unbiased Rounding", US Provisional Application No. 61/136,234, filed August 20, 2008 and entitled "A Method Of Reprogramming A Non-Volatile Memory Device Without Performing An Erase Operation", US Provisional Application No. 61/129,414, filed June 25, 2008 and entitled

"Improved Programming Speed in Flash Devices Using Adaptive Programming", and several other co-pending patent applications being filed concurrently (same day).

FIELD OF THE INVENTION

The present invention relates generally to error correction coding and more particularly to error correction coding of data to be stored in memory such as flash memory.

BACKGROUND OF THE INVENTION

The state of the art is believed to be represented by the following publications inter alia:

[1] "Interleaving policies for flash memory", United States Patent 20070168625

[2] "Minimization of FG-FG coupling in flash memory", United States Patent 6996004

[3] Construction of Rate (n-1)/n Punctured Convolutional Code with Minimum Required SNR Criterion, Pil J. Lee, IEEE Trans. On Comm. Vol. 36, NO. 10, Oct. 1988

[4] "Introduction to Coding Theory", Ron M. Roth, Cambridge University Press, 2006

5. United States Patent 5,077,737; 6,781,910; 6,873,543; 6,891,768; 6,914,809; 6,961,890 to Smith; 6,990,012; 7,079,436; 7,149,950; and 7,191,379; Published US Applications 2004015771; 2005172179; 2007226592; and 2007171730; and Published PCT Application No. WO2006013529.

[5] Coded modulation to increase storage capacity of multilevel memories, Hui-Ling Lou; Sundberg, C.-E. Global Telecommunications Conference, 1998. GLOBECOM 98. The Bridge to Global Integration. IEEE Volume 6, Issue, 1998 Page(s):3379 - 3384 vol.6

[6] On-chip error correcting techniques for new-generation flash memories, Gregori, S.; Cabrini, A.; Khouri, O.; Torelli, G., Proceedings of the IEEE, Volume 91, Issue 4, April 2003 Page(s): 602 – 616

[7] Multi-level memory systems using error control codes, Hsie-Chia Chang; Chien-Ching Lin; Tien-Yuan Hsiao; Jieh-Tsorng Wu; Ta-Hui Wang, Circuits and Systems, 2004. ISCAS apos;04. Proceedings of the 2004 International Symposium on Volume 2, Issue, 23-26 May 2004 Page(s): II - 393-6 Vol.2.

[8] Paulo Cappelletti, Clara Golla, Piero Olivo, Enrico Zanoni, "Flash Memories", Kluwer Academic Publishers, 1999

[9] G. Campardo, R. Micheloni, D. Novosel, "CLSI-Design of Non-Volatile Memories", Springer Berlin Heidelberg New York, 2005.

The disclosures of all publications and patent documents mentioned in the specification, and of the publications and patent documents cited therein directly or indirectly, are hereby incorporated by reference.

SUMMARY OF THE INVENTION

The following terms may be construed either in accordance with any definition thereof appearing in the prior art literature or in accordance with the specification, or as follows:

Bank = Device Back= memory Bank = Flash bank = Several Flash memory chips connected to the same controller and jointly providing a large amount of storage space.

Bit error rate (BER) = a parameter that a flash memory device manufacturer commits to vis a vis its customers, expressing the maximum proportion of wrongly read bits (wrongly read bits/ total number of bits) that users of the flash memory device need to expect at any time during the stipulated lifetime of the flash memory device e.g. 10 years.

Block = a set of flash memory device cells which must, due to physical limitations of the flash memory device, be erased together. Also termed erase sector, erase block.

Cell: A component of flash memory that stores one bit of information (in single-level cell devices) or n bits of information (in a multi-level device having 2 exp n levels). Typically, each cell comprises a floating-gate transistor. n may or may not be an integer. "Multi-level" means that the physical levels in the cell are, to an acceptable level of certainty, statistically partitionable into multiple distinguishable regions, plus a region corresponding to zero, such that digital values each comprising multiple bits can be represented by the cell. In contrast, in single-level cells, the physical levels in the cell are assumed to be statistically partitionable into only two regions, one corresponding to zero and one other, non-zero region, such that only one bit can be represented by a single-level cell.

Charge level: the measured voltage of a cell which reflects its electric charge.

Coding Rate = Number of information bits / (Number of information bits + Redundancy bits).

Cycling: Repeatedly writing new data into flash memory cells and repeatedly erasing the cells between each two writing operations.

Decision regions: Regions extending between adjacent decision levels, e.g. if decision levels are 0, 2 and 4 volts respectively, the decision regions are under 0 V, 0 V – 2 V, 2V – 4 V, and over 4 V.

Demapping: basic cell-level reading function in which a digital n-tuple originally received from an outside application is derived from a physical value representing a physical state in the cell having a predetermined correspondence to the digital n-tuple.

Digital value or "logical value": n-tuple of bits represented by a cell in flash memory capable of generating 2 exp n distinguishable levels of a typically continuous physical value such as charge, where n may or may not be an integer.

Erase cycle: The relatively slow process of erasing a block of cells (erase sector), each block typically comprising more than one page, or, in certain non-flash memory devices, of erasing a single cell or the duration of so doing. An advantage of erasing cells collectively in blocks as in flash memory, rather than individually, is enhanced programming speed: Many cells and typically even many pages of cells are erased in a single erase cycle.

Erase-write cycle: The process of erasing a block of cells (erase sector), each block typically comprising a plurality of pages, and subsequently writing new data into at least some of them. The terms "program" and "write" are used herein generally interchangeably.

Flash memory: Non-volatile computer memory including cells that are erased block by block, each block typically comprising more than one page, but are written into and read from, page by page. Includes NOR-type flash memory, NAND-type flash memory, and PRAM, e.g. Samsung PRAM, inter alia, and flash memory devices with any suitable number of levels per cell, such as but not limited to 2, 4, or 8.

Low cycle-count flash device: A flash device in which space S is reserved for redundancy bytes sufficient in number to support decoding, at a predetermined level of

accuracy, of original data stored at locations within the flash device which have been subjected to no more than a low number, c, of cycle counts. The number of redundancy bytes fitting into the space S is not sufficient to support decoding, at the predetermined level of accuracy, of original data stored at locations within the flash device which have been subjected to more than c cycle counts.

Mapping: basic cell-level writing function in which incoming digital n-tuple is mapped to a program level by inducing a program level in the cell, having a predetermined correspondence to the incoming logical value.

Page = A portion, typically 512 or 2048 or 4096 bytes in size, of a flash memory e.g. a NAND or NOR flash memory device. Writing can be performed page by page, as opposed to erasing which can be performed only erase sector by erase sector. A few bytes, typically 16 – 32 for every 512 data bytes are associated with each page (typically 16, 64 or 128 per page), for storage of error correction information. A typical block may include 32 512-byte pages or 64 2048-byte pages.

Precise read, soft read: Cell threshold voltages are read at a precision (number of bits) greater than the number of Mapping levels (2^n). The terms precise read or soft read are interchangeable. In contrast, in "hard read", cell threshold voltages are read at a precision (number of bits) smaller than the number of Mapping levels (2^n where n = number of bits per cell).

Present level, Charge level: The amount of charge in the cell. The amount of charge currently existing in a cell, at the present time, as opposed to "program level", the amount of charge originally induced in the cell (i.e. at the end of programming).

Program: same as "write".

Program level (programmed level, programming level): amount of charge originally induced in a cell to represent a given logical value, as opposed to "present level".

Reprogrammability (Np): An aspect of flash memory quality. This is typically operationalized by a reprogrammability parameter, also termed herein "Np", denoting the number of times that a flash memory can be re-programmed (number of erase-write

cycles that the device can withstand) before the level of errors is so high as to make an unacceptably high proportion of those errors irrecoverable given a predetermined amount of memory devoted to redundancy. Typically recoverability is investigated following a conventional aging simulation process which simulates or approximates the data degradation effect that a predetermined time period e.g. a 10 year period has on the flash memory device, in an attempt to accommodate for a period of up to 10 years between writing of data in flash memory and reading of the data therefrom.

Resolution: Number of levels in each cell, which in turn determines the number of bits the cell can store; typically a cell with 2^n levels stores n bits. Low resolution (partitioning the window, W, of physical values a cell can assume into a small rather than large number of levels per cell) provides high reliability.

Retention: of original physical levels induced in the cells; retention is typically below 100% resulting in deterioration of original physical levels into present levels.

Retention time: The amount of time that data has been stored in a flash device, typically without, or substantially without, voltage having been supplied to the flash device i.e. the time which elapses between programming of a page and reading of the same page.

Symbol: Logical value

Threshold level: the voltage (e.g.) against which the charge level of a cell is measured. For example, a cell may be said to store a particular digital n-tuple D if the charge level or other physical level of the cell falls between two threshold values T.

Certain embodiments of the present invention seek to provide use of variable rate code, depending on the number of program/erase cycles, enabling compact storage when Flash device performs well.

Certain embodiments of the present invention seek to provide use of an additional bank of Flash devices to enable efficient encoding with variable code rates, beyond the allotted redundancy of the device.

Certain embodiments of the present invention seek to provide using a weighted cycle count, depending also on the time passage from the last burst of cycles, as determined by the number of errors counted during the last read.

Certain embodiments of the present invention seek to provide a cycle count indicator stored along with the data on a Flash e.g. with a single encoding/decoding block.

There is thus provided, in accordance with at least one embodiment of the present invention, a method for encoding information arriving from a host in order to store the coded information in flash memory, the method comprising encoding information arriving from a host for storage at a flash memory location including generating a number of redundancy bytes, the encoding proceeding at an encoding rate which is a function of the number of redundancy bytes generated, the encoding including determining an effective error rate, including an anticipated rate of expected reading errors, for the flash memory location; and selecting the encoding rate as a function of the effective error rate such that the number of redundancy bytes is sufficient to overcome the anticipated rate of expected reading errors with a predetermined degree of certainty.

Further in accordance with at least one embodiment of the present invention, the flash memory location comprises a page.

Still further in accordance with at least one embodiment of the present invention, determining comprises determining a rate of expected reading errors assuming reading occurs after a time period T, based on known data on frequency of errors when information is read from a flash memory which has undergone a simulation of effects of passage of the time period T.

Additionally in accordance with at least one embodiment of the present invention, determining comprises reading a stored effective error rate, identifying a time t at which the effective error rate was stored, and updating the effective error rate as a function of the stored effective error rate and the time, to take into account spontaneous recovery of the flash memory location, over time, from effects of some of the program cycles to which the flash memory location has been subjected.

Still further in accordance with at least one embodiment of the present invention, the updating comprises finding, for at least one stored number of program cycles to which

the flash memory location was subjected at time t, a number of program cycles to which the flash memory location is subjected currently, after which the rate of errors currently read from the flash memory location equals the rate of errors currently read from the flash memory location which was subjected to the stored number of program cycles at time t.

Further in accordance with at least one embodiment of the present invention, the flash memory device comprises an NROM flash memory device.

Also provided, in accordance with at least one embodiment of the present invention, is a computer having an operating system, a storage unit comprising flash memory, and an operating system/storage unit interface providing interface between the storage unit and the operating system, the operating system being operative to query the storage unit to ascertain locations of bad flash memory portions, the interface including a bad location indicator operative to provide the operating system with an indication of the locations of bad flash memory portions, a flash memory controller operative to receive at least one of read commands, write commands, and status commands from a user, each command associated with a logical address, to translate each logical address into a physical address within the flash memory, and to translate the commands into at least one of read, erase and program instructions for the flash memory, and wherein the flash memory controller is operative to determine an encoding rate having fluctuations and accordingly and responsively to the querying, to declare as bad, at least one portion of the flash memory, which was previously devoted to original data and which, due to the fluctuations, is now devoted to redundancy bytes.

Further in accordance with at least one embodiment of the present invention, identifying a time t at which the effective error rate was stored, comprises associating a timer with the flash memory location, capable of generating a time stamp each time an operation is performed upon the flash memory location and storing at least one time stamp.

Further in accordance with at least one embodiment of the present invention, the operation comprises a flash memory location reading operation.

Still further in accordance with at least one embodiment of the present invention, identifying a time t at which the effective error rate was stored comprises reading the flash memory location including quantifying and correcting errors encountered, and

estimating the time of last programming as a function of the quantification of errors encountered.

Additionally in accordance with at least one embodiment of the present invention, the effective error rate for an individual flash memory location is stored adjacent the individual flash memory location.

Further in accordance with at least one embodiment of the present invention, the flash memory location comprises a page and the effective error rate for the page is stored within the page.

Further in accordance with at least one embodiment of the present invention, the effective error rate is encoded at a fixed coding rate corresponding to a fixed number of redundancy bytes which is high enough to overcome expected corruptions of the effective error rate.

Still further in accordance with at least one embodiment of the present invention, the encoding, determining and selecting are performed for a plurality of flash memory locations.

Also provided, in accordance with at least one embodiment of the present invention, is high cycle count flash memory apparatus having a cycle count C, the apparatus comprising a set of flash memory devices including at least one low cycle-count flash device having a cycle count $c < C$ storing original data elements, at least one additional flash device sufficient in size to store redundancy bytes sufficient in number to ensure that if information is encoded into the set of flash devices, with redundancy, thereby to generate a set of redundancy bytes, the additional flash device is sufficiently large to store a set of redundancy bytes sufficiently large to enable the information to be decoded at at least a predetermined level of accuracy; and apparatus for reading at least one original data element from the low cycle flash memory device, and, in parallel, for reading at least one redundancy byte generated in the course of encoding the original data element, from the additional flash device.

Further in accordance with at least one embodiment of the present invention, the set of flash memory devices includes more than one flash device and the at least one additional flash device stores redundancy bytes for the more than one flash device.

Any suitable processor, display and input means may be used to process, display, store and accept information, including computer programs, in accordance with some or all of the teachings of the present invention, such as but not limited to a conventional personal computer processor, workstation or other programmable device or computer or electronic computing device, either general-purpose or specifically constructed, for processing; a display screen and/or printer and/or speaker for displaying; machine-readable memory such as optical disks, CDROMs, magnetic-optical discs or other discs; RAMs, ROMs, EPROMs, EEPROMs, magnetic or optical or other cards, for storing, and keyboard or mouse for accepting. The term "process" as used above is intended to include any type of computation or manipulation or transformation of data represented as physical, e.g. electronic, phenomena which may occur or reside e.g. within registers and /or memories of a computer.

The above devices may communicate via any conventional wired or wireless digital communication means, e.g. via a wired or cellular telephone network or a computer network such as the Internet.

The apparatus of the present invention may include, according to certain embodiments of the invention, machine readable memory containing or otherwise storing a program of instructions which, when executed by the machine, implements some or all of the apparatus, methods, features and functionalities of the invention shown and described herein. Alternatively or in addition, the apparatus of the present invention may include, according to certain embodiments of the invention, a program as above which may be written in any conventional programming language, and optionally a machine for executing the program such as but not limited to a general purpose computer which may optionally be configured or activated in accordance with the teachings of the present invention.

A detailed description of the embodiments referred to above, and other embodiments, follows.

Any trademark occurring in the text or drawings is the property of its owner and occurs herein merely to explain or illustrate one example of how an embodiment of the invention may be implemented.

Unless specifically stated otherwise, as apparent from the following discussions, it is appreciated that throughout the specification discussions, utilizing terms such as, "processing", "computing", "estimating", "selecting", "ranking", "grading", "calculating", "determining", "generating", "reassessing", "classifying", "generating", "producing", "stereo-matching", "registering", "detecting", "associating", "superimposing", "obtaining" or the like, refer to the action and/or processes of a computer or computing system, or processor or similar electronic computing device, that manipulate and/or transform data represented as physical, such as electronic, quantities within the computing system's registers and/or memories, into other data similarly represented as physical quantities within the computing system's memories, registers or other such information storage, transmission or display devices.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a simplified functional block diagram illustration of a system which varies the encoding rate, e.g. the amount of redundancy added, according to the number of cycles in an erase block, all constructed and operative in accordance with certain embodiments of the present invention;

Fig. 2 is a simplified functional block diagram illustration of the encoder/decoder of Fig. 1, constructed and operative in accordance with certain embodiments of the present invention;

Fig. 3A is a diagram of the Flash device array of Fig. 1, constructed and operative in accordance with a first embodiment of the present invention;

Fig. 3B is a diagram of the Flash device array of Fig. 1, constructed and operative in accordance with a second embodiment of the present invention;

Fig. 4 is a diagram of a page comprising both a back pointer and a cycle count indicator which may be stored as part of the redundancy of the page, along with error

correction information, all in accordance with certain embodiments of the present invention;

Fig. 5A is a diagram of a page mapping scheme which is effected subsequent to an encoding process which uses a low rate, i.e. with large redundancy blocks, in accordance with certain embodiments of the present invention;

Fig. 5B is a diagram of a page mapping scheme which is effected subsequent to an encoding process which uses a high rate, i.e. with small redundancy blocks, in accordance with certain embodiments of the present invention;

Fig. 6 is a diagram of the Flash device array of Fig. 1, constructed and operative in accordance with certain embodiments of the present invention;

Fig. 7 is a diagram of one possible page mapping scheme which is suitable for the embodiment of Fig. 8;

Fig. 8 is a simplified functional block diagram illustration of a variable rate system including a single NAND Flash device 1200 in accordance with certain embodiments of the present invention;

Fig. 9 is a simplified functional block diagram illustration of the uController of Fig. 8, in accordance with certain embodiments of the present invention;

Fig. 10 is a bit error rate vs. cycle count graph useful in accordance with certain embodiments of the present invention;

Fig. 11 is an example of a Physical Block Usage table, filled with example data, which may be used by the memory management unit of Fig. 9 in accordance with certain embodiments of the present invention;

Fig. 12 is an example of a logical page lookup table, filled with example data, which may be used by the memory management unit of Fig. 9 in accordance with certain embodiments of the present invention;

Figs. 13 and 14 are simplified flowchart illustrations of methods for locating a logical page, in accordance with certain embodiments of the present invention;

Fig. 15 is a simplified flowchart illustration of a "read logical page" method which may be performed by the uController 1100 of Fig. 8, in accordance with certain embodiments of the present invention;

Figs. 16A-16B, taken together, form a simplified flowchart illustration of a block copy and modify method, operative in accordance with certain embodiments of the present invention;

Fig. 16C is a simplified flowchart illustration of a method for performing step 1780 of Fig. 16B, in accordance with certain embodiments of the present invention;

Fig. 16D is a simplified flowchart illustration of a method for performing step 1784 of Fig. 16B, in accordance with certain embodiments of the present invention; and

Fig. 17 is a simplified flowchart illustration of a page writing method, operative in accordance with certain embodiments of the present invention.

DETAILED DESCRIPTION OF CERTAIN EMBODIMENTS

Today's Flash memory devices store information with a high density of Flash cells with ever smaller dimensions. In addition, Multi-Layer Cells (MLC) store several bits per cell by setting the amount of charge in the cell. The amount of charge is then measured by a detector, as measured by a threshold voltage of the transistor gate. Due to inaccuracies during the programming procedure and charge loss due to time and temperature (also known as retention), the measured levels during a Read operation suffer from detection errors. The small dimensions of the Flash cells result in cells that can store very small amounts of charge, enhancing the effects of inaccuracies due to programming and retention. Thus, new single layer cells (SLC) and multi-level cell devices have significantly increased bit error rate (BER), decreasing the reliability of the device.

Flash devices are organized into (physical) pages. Each page comprises a section allocated for data (512bytes-4Kbytes) and a small amount of bytes (16-32 bytes for every 512 data bytes) comprising redundancy and back pointers. The redundancy bytes are used to store error correcting information, for correcting errors which may have occurred e.g. during the page Read. Each Read and Program operation is performed on an entire page. A number of pages are grouped together to form an Erase Block (EB). A page cannot be erased unless the entire erase block which comprises it is erased.

An important measure of a Flash device quality is the number of times (Np) it may be reprogrammed before irrecoverable errors occur. The higher the number of program-erase cycles, the higher the bit error rate. Thus, today's multi-level cell devices can perform around Np=1000 cycles or less before the allocation of 16-32 bytes of redundancy per 512 bytes of data bytes becomes insufficient to correct errors. Single-level cell devices usually perform better but obtain a much lower density, and hence their prices are much higher. Note that following Np program-erase cycles the device is still operational but the bit error rate is higher. Furthermore, in many devices (e.g. NROM Flash devices), this behavior is predictable and it can be shown that the number of redundancy bytes required to correct these errors does not jump rapidly.

One application of Flash devices is solid state disks (SSD) where an array of Flash devices is used as the storage media of a computer hard drive, thus enjoying the fast

Read and Access times of Flash chips. In a solid state disk (SSD), several Flash chips are programmed and read simultaneously to increase the Read and Program speeds. For this purpose, solid state disks are arranged into multiple "Banks" that allow parallel read/write operations, each Bank typically comprising a number of Flash devices. The Read/Program performance may be multiplied by the number of Banks, compared with a single Flash chip, as is evident from Reference [1].

Certain embodiments of the present invention provide for encoding with variable redundancy length, depending on the number of Program/Erase cycles (Np) which have occurred in an erase block. For this purpose, an additional Bank of Flash devices may be added which contains additional space for redundancy and data, beyond that allotted in the Flash pages. Thus, the additional redundancy may facilitate correction of a greater number of errors than is conventional, thereby to provide for a significantly higher number of Program/Erase cycles than is conventional and/or than was originally designed. As the additional Bank is read and programmed in parallel with the other banks, the performance hit is small. Furthermore, as only a small proportion of memory banks is typically added to the existing number of banks, such as for example 1 bank per every 4 banks, the added cost may be much smaller than the cost of replacing multi-level cell with single-level cell devices to increase reliability. In applications or environments where Np is small and no additional redundancy is required, the added bank may be used for data.

Fig. 1 illustrates a system which varies the encoding rate, e.g. the amount of redundancy added, according to the number of cycles in an erase block. If a program operation is initiated by a host 10, the processing of the data may be as follows:

a. The Host 10 sends "logical" pages to be programmed on the Flash array 50. The host may comprise a computer such as a personal computer, or an SD card-employing application. b. A Variable Rate Encoder/Decoder 20 encodes the logical pages and adds redundancy e.g. as per the example described in detail below. The amount of redundancy added may for example depend on the rate indication obtained by a Cycle count indicator 40.

c. A Page Interleaver 30 receives a sequence of, say, L encoded pages and simultaneously programs them into the Flash array.

d. The Cycle Count Indicator 40 reads the number of cycles that the erase block has gone through from the flash array and determines, e.g. as per the example described in detail below, an appropriate encoding rate. The cycle count may be stored in the erase block on some of the redundancy bytes of some of the pages in the erase block. Indicator 40 typically also knows how to program the cycle count following erasure of the block and programming of the block's first page.

A read operation may be performed analogously, with the operations reversed.

Variable Rate Encoder/Decoder 20 performs variable rate encoding and decoding operations according to the rate indication provided by the Cycle Count Indicator 40 . Any suitable method may be employed for varying rate of the code, such as but not limited to the following:

1. Conventional codes for Flash devices include algebraic codes such as Reed Solomon or BCH e.g. as described in Reference [4]. A suitable method for changing the rate of such a code is to change the length of the data being encoded/decoded while maintaining constant redundancy.

For example, consider a binary BCH code designed for a 2Kbyte page with a redundancy of 128 bytes. Such a code may be used on some of today's devices with standard redundancy. To code with twice the redundancy, code two 1Kbyte-batches and add to each a redundancy of 128bytes. Thus, overall, redundancy of 256 bytes is provided for every 2Kbytes of data. This approximately doubles the possible magnitude of the input bit error rate. Similarly, the redundancy may be quadrupled by coding over batches of 512 bytes.

2. It may sometimes be more efficient, although less compact, to add an inner convolutional code with several puncturing options and use an Algebraic code as an outer code. The algebraic code may be a standard one such as RS or BCH codes. The inner code may be a convolutional code whose puncturing rate may be a parameter of the code e.g. as described in Reference [3]. Fig. 2 is an example of this type of encoder/decoder. If the initial rate (i.e. the rate derived by the redundancy inherently allotted in each Flash device per page) is used, switches A and B are set to bypass the convolutional encoder/decoder and only the algebraic code may be operative. If lower rates are

employed, the parameter f is set according to the cycle count e.g. as described below, and the convolutional code may be used.

For example, consider an outer encoder using a binary BCH code designed for a 2Kbyte page with a redundancy of 128 bytes. By setting f=4, the convolutional encoder may yield a rate of 4/5 and the overall redundancy may be 640 bytes. 128 bytes may be stored with the data on the same page and the other 512 bytes may be stored separately. By setting f=8, a rate of 8/9 is obtained and the overall redundancy is 384 bytes.

3. An additional option is to build a set of encoders/decoder, each designed for a specific rate. This option may be least compact but may assure the most effective use of the redundancy bytes.

Page Interleaver 30 receives a sequence of encoded pages and programs them simultaneously onto the Flash array 50. For example, consider a flash array with 5 Banks of Flash devices and assume that the variable code rate is set to 4/5. The host may simultaneously send 4 logical pages which may be encoded at a rate of 4/5 and stored simultaneously over the 5 memory banks, one page per bank. Allocation of banks to data or redundancy may be done in one of several ways such as but not limited to the following:

a. The data + redundancy may be stored sequentially over the banks. For example, Fig. 3A is a diagram of a Flash array with 5 banks. In each bank a single page may be used. Following encoding at a rate of 4/5, the first bank contains part of the first encoded page. The second bank contains the remainder of the first page and the beginning of the second encoded page, and so on; the remaining encoded pages are distributed over the banks as shown.

b. The data of each logical page is associated with its own bank and the additional redundancy of all encoded pages is stored in a designated bank. For example, Fig. 3B is a diagram of a Flash array with 5 banks. Banks 1 through 4 store the data and some partial redundancy of the respective encoded pages. Bank 5 contains the rest of the redundancies of each of the encoded pages.

The interleaver 30 of Fig. 1 may additionally be operative to determine the correct page address, bank, and device within a bank, when a read or program operation is performed.

If a constant rate is used, one-to-one addressing is able to translate each "logical" address to a "physical" address. However, if variable rate codes are used, it may be desired to utilize unused memory locations when the redundancy is small. Fig. 5A illustrates a page mapping which is effected subsequent to an encoding process which uses a low rate, i.e. with large redundancy blocks. In contrast, Fig. 5B illustrates a page mapping which is effected subsequent to an encoding process which uses a high rate, i.e. with small redundancy blocks. The compacting of the pages in Fig. 5B creates a page mapping problem between the logical address and its actual address. To overcome the mapping problem, a back pointer may be associated with each page or set of pages (e.g. erase block), which maps the page or set to its logical address. This back pointer may be stored during a program operation as part of the redundancy. Then, during a read or program operation, the interleaver 30 may scan the array 50 to determine the actual address.

Alternatively, the interleaver 30 may read the entire array 50 and fill a table mapping the logical addresses to physical addresses during the initialization of the system. Fig. 4 is a diagram of a page comprising both a back pointer 330 and a cycle count indicator 320 which may be stored as part of the redundancy of the page, along with the actual error correction information 340. The cycle count indicator 320 (Np) may optionally be used to determine the encoder rate during the next programming cycle and the rate during a decoding procedure, in which case, the cycle count may be encoded separately, as described in detail below.

According to certain embodiments of the present invention, as the cycles occur at the erase block level rather than at the individual page level. it may be more efficient to store the cycle counter only once per erase block instead of once per page. If so, a specific location within the erase block may be allocated to store the cycle count (Np) for the entire block. Similarly, as the cycle count affects the encoding rate of the entire block, only one back pointer may be stored for an entire block. For example, a back pointer to the first page in the block may be stored, and no back pointers for any of the remaining pages may be stored; instead, the other pages may be counted out sequentially starting from the back pointer of the first page.

Another result of allowing high rate pages to be compacted over the banks as shown in Fig. 5B may be that as the Flash array is cycled, the encoding may use lower rates and hence, more space may be used, such that the space available in the flash array 50 for actual data may become smaller in time. This may result in addressing problems. For example, at the beginning of life two (say) logical pages may be programmed to the array, first written to the lowest logical address available and the last to the highest logical address. As time progresses, the pages in between the 2 logical addresses are cycled and are programmed with lower rates, hence employing more space, such that it may no longer be possible to program all the addresses that were available at the beginning. To accommodate this possibility, some logical addresses in the array 50 may be marked as "bad blocks" or "unavailable blocks", to alert the host 10, typically via a suitable controller, that these addresses are not to be used. Alternatively, some pages may be falsely marked as occupied although in fact they contain no data.

For compactness and simplicity, an address space which is believed to be typical at end of life may be pre-designated.

The Cycle Count Indicator 40 reads the cycle count field of an individual Erase Block or page and determines the encoding/decoding rate for the encoder 20 and for the interleaver 30. Cycle Count Indicator 40 also determines the number of cycles to be programmed next.

Any suitable scheme may be employed to store the cycles count indicator (Np), such as but not limited to the following:

a. The cycle count indicator 40 may not be encoded along with the data and instead may be encoded separately typically using its own code e.g. a short BCH code, thereby to allow the decoding rate to be determined before reading the page.

b. The cycle count indicator 40 may be encoded with the data using the same encoder/decoder, during the read operation. A suitable encoding technique for this purpose is, for example, the selectably-puncturable inner convolutional code/outer algebraic coding technique described above. The outer, algebraic code may use a short redundancy which fits the data in one page in the Flash device e.g. as per the example described in detail below. The remaining redundancy, which is due to the inner convolution code, may be stored separately. For high rate coding only the outer

(algebraic) code may be used whereas for lower rates coding, both the inner and outer codes may be used. The decoding may be performed first based solely on the algebraic code. If the decoding is unsuccessful, the read operation may be repeated with reliance on the inner decoder as well. Whereas this technique may be time consuming for high cycle counts (Np), no additional special encoder may be needed for the cycle count indicator.

Normally, the cycle count for each time an erase block is programmed, is simply an increment of the previous count. However, Flash memory devices, with time, tend to "forget" the number of cycles and improve their bit error rate performance following retention. For example, a Flash which underwent 1000 program/erase cycles within a period of one week may suffer a noticeable deterioration in bit error rate performance following retention. However, a Flash device that underwent 999 program/Erase cycles and only a year later underwent its 1000th cycle may exhibit better bit error rate performance, following additional retention, as compared to the previous flash device which underwent all 1000 cycles in the first week. The time passage from the previous erase block erasure can be estimated on a basis of the cycle counts and the number of errors which occurred during the latest read operation of the oldest page in the erase block. The cycle count (Np) can therefore be set to a lower value if the number of errors is not large, e.g. as per the example described below, thus enabling a higher rate encoding and more compact storage. A cycle counter so programmed is termed below a "weighted" cycle count or "weighted" cycle count indicator.

Weighted cycle count is a parameter characterizing a memory portion such as an erase block which is typically an increasing function of the number of erase-write cycles that memory portion has undergone, and an increasing function of the effective duration of time the memory portion had available to recover from erase-write cycles it has undergone, where the effective duration of time is a function of the length of the time period the memory portion had available to recover from erase-write cycles it has undergone (also termed retirement) and the temperature of the memory portion during that time period.

Computation of the weighted cycle count following a certain period of retirement may or may not be based upon a physical model of the Flash memory device. In [1] N. Mielke, H. Belgal, I. Kalastrisky, P. Kalavade, A. Kurtz, Q. Meng, N. Righos, and J. Wu,

"Flash EEPROM Threshold Instabilities due to Charge Trapping During Program/Erase Cycling", IEEE Trans. On Device and Materials Reliability, Vol. 4, No. 3, Sep. 2004, the authors propose a model where the fraction of traps which disappear at room temperature during time Tr is given by $\propto \cdot log(T_r)$ for some constant trap dissipation rate which depends on the Flash technology being used and the temperature. Therefore, according to some embodiments of the present invention, as the traps are the main contributors for endurance degradation with cycling and retention (at technologies of <100nm), the disappearance of traps may be represented by effectively decreasing the weighted cycle count.

In the same paper the authors show that the number of traps grows approximately as the square root of the number of cycles. Therefore, according to some embodiments of the present invention, an effective number of cycles may be approximated by: $C_S \cdot \left(1-\propto \cdot log(T_r)\right)^2$ where Cs is the previous weighted cycle count.

Flash array 50, as shown in Fig. 6, typically comprises L+1 banks of Flash memory devices of which L banks are allocated for data and Bank L+1 is allocated for redundancy. Each Bank comprises M Flash memory devices. Each memory device comprises K Erase Blocks and each Erase Block comprises T pages. At each bank, the erase blocks are numbered serially from 1 to MK. In each erase block, the pages are numbered from 1 to T, resetting at the beginning of each erase block.

An example of how the usage of code may be modified to accommodate different rates and hence, handling different bit error rates (BERs), is now described. An algebraic BCH code is designed over the field GF(213), as described in Reference [4], to correct 24 errors. Such a code typically employs 39 bytes of redundancy. This code may be employed to encode data with lengths (L) between 1 bytes and 984 bytes. The rate is determined by the ratio r=L/(L+39). The shorter the data length, the higher the bit error rate that the code can handle. In this example, the length of the logical page is 2048 bytes. Three strategies for using the above code are now described:

1. Encode 2048 bytes of data _+ 11 bytes provided for additional information such as cycle count, and add 39*3=117 bytes. To do this, divide the 2048+11 bytes into 3 consecutive sections of 686,686 and 687 bytes. Encode each section separately, thereby to generate 39 bytes of redundancy per section, for a total of 2048+128=2176 bytes. This

code can handle bit error rate of 6.5E-4 and at its output produces a bit error rate of 1E-15.

2. Encode 2048 bytes of data _+ 11 bytes and add 20*39 bytes. This may be effected by dividing the 2048+11 bytes into 20 consecutive sections, the first 19 having 103 bytes and the last one having 102 bytes. Encode each section separately, thereby to generate 39 bytes of redundancy, for a total of 2839 bytes. This code can handle a bit error rate of 3E-3 and at its output produces a bit error rate of 1E-15.

3. Encode 2048 bytes of data _+ 11 bytes and add 58*39 bytes. This may be effected by dividing the 2048+11 bytes into 58 consecutive sections, the even sections having 35 bytes and the odd sections having 36 bytes. Encode each section separately, thereby to generate 39 bytes of redundancy, for a total of 4321 bytes. This code can handle bit error rate of 5.5E-3 and at its output produces a bit error rate of 1E-15.

Fig. 10 shows an example of how the bit error rates may change as a function of the program erase cycles in a NAND flash device employing 3 bits/cell. It can be seen from the graph of Fig. 10 that for 3 bits/cell specifically, the first strategy may be employed up to Np =3000, followed by the second strategy until Np = 5000 and above Np=5000 the third strategy may be employed. An alternative is to switch between the 4 bit/cell configuration and the 3 bit/cell configuration depending on the cycle count.Fig. 8 illustrates a variable rate system including a single NAND Flash device 1200 in accordance with certain embodiments of the present invention. In this system, logical pages may accommodate more than a single physical page as shown in Fig. 7. The system of Fig. 8 typically comprises a uController (1100), a NAND Flash device (1200) and a Host (1000). The system may realize an application such as a USB drive or an SD card; the Host 1000 would then be a PC or an application using SD cards such as a digital camera or an MP3. The Host 1000 sends Read/Write commands to the uController 1100 which translates these read/Write commands to read/program/erase commands on the Flash device 1200. The communication between the host 1000 and uController 1100 may occur through standard protocols such as the SD standard or the USB protocol.

Fig. 9 is a simplified functional block diagram illustration of the uController 1100 of Fig. 8, in accordance with certain embodiments of the present invention. As shown, the uController 1100 comprises an Encoder 1120 and Decoder 1130. Typically, each of

these can encode at various selectable rates, e.g. as described above, to produce different amounts of redundancy. A host interface 1110 receives the commands from the host 1000 and responds thereto. The host interface 1110 typically comprises a buffer that stores a logical page to be programmed.

The pages sent by the host 1000 may be smaller than the data length used by the encoder 1120 in which case the host interface 1110 translates the data sizes to those used by encoder 1120. For example, the encoder 1120 may use a constant size of 2048 bytes of data whereas the interface with the host 1000 may occur in chunks of 512 bytes. The interface 1110 may serialize 4 chunks of 512 bytes to obtain the 2048 bytes. If this is not possible, e.g. if not enough chunks have arrived, then the interface 1110 may append to the available chunks, a sequence of zeroes, until a full page of 2048 bytes is obtained.

The NAND interface 1150 connects to the NAND flash 1200 and is operative to send suitable program/erase and read commands to the NAND flash 1200. The NAND interface 1150 typically comprises a program buffer (not shown) the size of a page (say 2048+128 bytes). Each time the program buffer is full, a program command is initiated to a physical page location, previously sent to the NAND interface 1150.

A payload logical page buffer 1170 is typically provided to temporarily hold the data to be programmed. A memory management unit 1140 interfaces with the host interface 1110, the variable rate decoder 1130 and, optionally, with a clock 1160, all as described in detail below. The optional clock (1160) stores time stamps and is provided in those embodiments which utilize effective cycle counts rather than raw cycle counts.

Memory management unit 1140 is operative to translate between the logical page addresses as perceived by the host 1000, to physical page addresses stored on the NAND flash 1200. Memory management unit 1140 handles the different rates used by the system, typically using two tables, examples of which are illustrated in Figs. 11 and 12 respectively. These tables are now described:

The Physical Block Usage table (e.g. as in Fig. 11) is a lookup table that associates some or all of the following information items (columns in Fig. 11) with each erase block (row in Fig. 11) in the flash device 1200:

a cycle count (Np), a number of logical pages that the erase block can store. This parameter, which is also termed herein "#PagesPerBlock" typically depends on the Cycle

count Np and determines rates at which the encoder operates in order to create an encoded page, and

a number of logical pages which are actually allocated to this erase block. Optionally, we may also store a time stamp indicating when the block was last programmed.

The logical page lookup table, e.g. as in Fig. 12, is a lookup table that stores, for each set of, say, 32 logical addresses, some or all of the following information items:

Block#: an erase block in the flash array 1200 which has been allocated to store these addresses

FirstPage#: a logical page within the block on which the first address in this set is stored, and

OccupyVec: a vector of, say, 32 bits indicating which pages have already been programmed and which pages have not. In the illustrated example, 1= programmed, 0=empty.

The size of the set of logical addresses may be the largest common divider of possible values of #PagesPerBlock stored in the physical block usage table of Fig. 11. For example, if the table of Fig. 11 indicates that there are blocks storing 128, 96 and 64 logical pages then the set of logical addresses may include 32 logical addresses.

The illustrated embodiment and example data are based on the 3-rate algebraic code described above and assume that the Flash array 1200 includes Erase blocks, each comprising 128 physical pages, each including 2048+128 bytes. It is also assumed that the data payload comprises 2048 bytes received from the host 1000 and 11 bytes for cycle counts and back pointers as described above. The rate selected for the encoder 1120 and decoder 1130 depends on the cycles count as described above. Thus, a block may comprise 128, 96 or 64 logical pages, using rate 1, 2 or 3.

If a Block comprises 96 logical pages, each set of 32 logical pages is assumed, in the illustrated example, to be programmed into a set of 42 physical pages such that logical pages 0-31 are mapped to physical pages 0-41, logical pages 32-63 are mapped to physical pages 42-83 and logical pages 64-95 are mapped to physical pages 84-125. If there are 64 logical pages per block, each logical page is mapped to two different physical pages.

Each entry in the Logical page lookup table of Fig. 12 comprises a reference to a set of 32 logical pages. Thus, in order to locate the location of a logical page, the methods of Figs. 13 and 14 may be followed. Fig. 15 is, then, a suitable "read logical page" operation which may be performed by the uController 1100 of Fig. 8. Regarding a "write logical page" operation, if a logical page was previously programmed, it cannot be replaced without first erasing the physical block which contains it. Also, logical pages are now encoded over several physical pages. Hence, even if a target logical page has not been programmed, an adjacent logical page may have been programmed, in which case it may not be possible to program the target page because it may be partially located on the same physical page allocated to another programmed page. To overcome this problem a block copy and modify operation, e.g. as shown in Figs. 16A – 16D, may be performed, followed by a write operation e.g. as shown in Fig. 17.

It is appreciated that each of the methods shown in the various simplified flowchart illustrations provided herein may include some or all of the steps actually shown, suitably ordered e.g. as shown.

In the illustrated embodiment, logical pages have a uniform, predetermined length. However, due to the variable code rate, each erase block is able to store a different number of logical pages. In the illustrated embodiment, 3 example strategies (not intended to be limiting) are employed, as described above, and the number of logical pages which each erase block is to store is determined as a function of the strategy. In the example described herein, the code rates are such that 128, 96 and 64 logical pages are stored per erase block. Therefore, for example, the number of bytes used to store each logical page when strategy 2 is used, is 2839, as may be seen, e.g. in step 1350 of Fig. 13. The numbers of bytes used to store each logical page when strategies 1 and 3 are used are each of course different. In strategy 1, each physical page stores exactly one logical page in the illustrated example. In strategy 3, each logical page is stored over exactly two physical pages as is evident e.g. from step 1385 in Fig. 13.

Specifically, Fig. 14 describes one method for receiving (step 1410) an indication of a logical page, e.g. from a host, and returning the physical page mapped to that logical page. In step 1420, the table of Fig. 12 is accessed, at the record corresponding to the logical page address or the floor value thereof. For example, if the logical page address

obtained in step 1410 is 319, the record for which logical page address = 288 is accessed. In this record, block#=3 and Firstpagenumber = 32. It is appreciated that each record in the table of Fig. 12 refers to a "segment" including a plurality of logical pages (32 logical pages in the illustrated example). So, the record accessed above indicates that the segment to which the current logical page belongs, is stored in erase block 3 starting from logical page 32 within that erase block. The Occupyvec field of the above record indicates whether or not each logical page in the segment to which the record corresponds, is occupied; "1" indicating that the logical page is occupied (also termed herein "allocated") and "0" indicating that the logical page is unoccupied or unallocated. For example, in the segment in question, the first, third, fourth, fifth and sixth logical pages are occupied, and the second logical page is not (as evident from the fact that the first 6 bits in the OccupyVec field of the record corresponding to logicalPageAdress = 288, are 101111 respectively.

If the logical page is not allocated i.e. does not contain data, an "unallocated" indication is returned (step 1440); if it is allocated, the physical address of the logical page in question, within the erase block whose serial number is Block#, is computed (step 1450). One suitable method of performing step 1450 is illustrated in Fig. 13. The output of the method of Fig. 14 includes (step 1460) the block in which the logical page whose address was obtained in step 1410 is stored or is to be stored, the number of the physical page within that block at which the logical page is stored or is to be stored, the number of the byte within that physical page at which the logical page is stored (since the beginning of each logical page generally does not correspond to the beginning of the physical page) and the status of the logical page: either programmed, free or blocked; or unallocated (step 1440).

Fig. 13 is a possible implementation of Fig. 14, step 1450 and comprises a method for finding a physical address of a known logical page on which to write or from which to read, within a known block, all as a function of a previously effected mapping between physical and logical pages. As described above, the block number (Block#) is known. The variables to be obtained in step 1310 may be known e.g. from step 1420 of Fig. 14. In step 1315, the table of Fig. 11 is used to determine how many logical pages are stored in the block (block No. 3) in which the logical page in question resides. The answer, in

the illustrated embodiment, may be either 64, 96 or 128; and in fact happens to be 96 as shown in the fourth record of Fig. 11 (which corresponds to block No. 3).

Step 1320 computes remainders after dividing logical addresses by 32. The remainder indicates the position of the logical page within its 32-page section (e.g. logical page 319 is 31st within its section). Therefore, the remainder stipulates the bit within the OccupyVec vector which pertains to the logical page in question. Step 1320 collects information as to whether the logical page, and the pages just before and after it, are or are not programmed.

The method now typically proceeds according to the number of pages per block which in the illustrated embodiment, may be either 64, 96 or 128. In each case, the byte within the physical page starting from which the logical page is or is to be stored, is computed, and an indication is provided of whether the page is free i.e. both it and its neighbors are unoccupied or unallocated, blocked (i.e. free but with at least one occupied neighbor) or allocated.

Page status may be used in reading. For example, Fig. 15 is a simplified flowchart illustration of a "read logical page" method which may be performed by the uController 1100 of Fig. 8, in accordance with certain embodiments of the present invention. As shown, after a logical page address is obtained e.g. from a host (step 1510) and the physical address of that logical page is located e..g using the method of Fig. 14 (step 1520), the method proceeds depending on whether or not the logical page is found to be programmed. If not, the logical page buffer 1170 in step 1540 is reset (e.g. with zeros). If the logical page is programmed, it is decoded (step 1590), using one of the three (in the illustrated example) code rates, as determined (step 1560) by the number of pages in the block in which the logical page is stored.

Page status may also be used in writing. For example, Fig. 17 is a simplified flowchart illustration of a page writing method, operative in accordance with certain embodiments of the present invention. According to the method of Fig. 17, a copy modify process is used to write on a particular logical page (step 2070), if the physical page storing the logical page and/or at least one of the neighbors of that physical page is found to be programmed (step 2030). In the copy modify process, which may follow the method of Figs. 16A – 16B, the erase block which stores the logical page in question is copied to

another location and during the copying procedure the logical page in question is modified to store the new data. The original block is then erased (step 2080). The PagesPerBlock field is then updated depending on the cycle count Np since in the illustrated embodiment, the cycle count determines the code rate, using cut-off points of (say) 3000 and 5000.

Otherwise, i.e. if the physical page storing the logical page and both of its neighbors are unoccupied, the code rate is set (step 2040) depending on the value of the PagesPerBlock field in the table of Fig. 11 and the data is encoded at that rate (step 2050). The encoded data is then programmed into the flash (step 2060), at the appropriate physical address e.g. as determined by the method of Fig. 14.

Figs. 16A-16B, taken together, form a simplified flowchart illustration of a block copy and modify method, operative in accordance with certain embodiments of the present invention and suitable for implementing the copy-modify step 2070 of Fig. 17. Fig. 16C is a simplified flowchart illustration of a method for performing step 1780 of Fig. 16B, in accordance with certain embodiments of the present invention. Fig. 16D is a simplified flowchart illustration of a method for performing step 1784 of Fig. 16B, in accordance with certain embodiments of the present invention.

It is appreciated that software components of the present invention including programs and data may, if desired, be implemented in ROM (read only memory) form including CD-ROMs, EPROMs and EEPROMs, or may be stored in any other suitable computer-readable medium such as but not limited to disks of various kinds, cards of various kinds and RAMs. Components described herein as software may, alternatively, be implemented wholly or partly in hardware, if desired, using conventional techniques.

Included in the scope of the present invention, inter alia, are electromagnetic signals carrying computer-readable instructions for performing any or all of the steps of any of the methods shown and described herein, in any suitable order; machine-readable instructions for performing any or all of the steps of any of the methods shown and described herein, in any suitable order; program storage devices readable by machine, tangibly embodying a program of instructions executable by the machine to perform any or all of the steps of any of the methods shown and described herein, in any suitable order; a computer program product comprising a computer useable medium having

computer readable program code having embodied therein, and/or including computer readable program code for performing, any or all of the steps of any of the methods shown and described herein, in any suitable order; any technical effects brought about by any or all of the steps of any of the methods shown and described herein, when performed in any suitable order; any suitable apparatus or device or combination of such, programmed to perform, alone or in combination, any or all of the steps of any of the methods shown and described herein, in any suitable order; information storage devices or physical records, such as disks or hard drives, causing a computer or other device to be configured so as to carry out any or all of the steps of any of the methods shown and described herein, in any suitable order; a program pre-stored e.g. in memory or on an information network such as the Internet, before or after being downloaded, which embodies any or all of the steps of any of the methods shown and described herein, in any suitable order, and the method of uploading or downloading such, and a system including server/s and/or client/s for using such; and hardware which performs any or all of the steps of any of the methods shown and described herein, in any suitable order, either alone or in conjunction with software.

Certain operations are described herein as occurring in the microcontroller internal to a flash memory device. Such description is intended to include operations which may be performed by hardware which may be associated with the microcontroller such as peripheral hardware on a chip on which the microcontroller may reside. It is also appreciated that some or all of these operations, in any embodiment, may alternatively be performed by the external, host-flash memory device interface controller including operations which may be performed by hardware which may be associated with the interface controller such as peripheral hardware on a chip on which the interface controller may reside. Finally it is appreciated that the internal and external controllers may each physically reside on a single hardware device, or alternatively on several operatively associated hardware devices.

Any data described as being stored at a specific location in memory may alternatively be stored elsewhere, in conjunction with an indication of the location in memory with which the data is associated. For example, instead of storing page- or erase-

sector-specific information within a specific page or erase sector, the same may be stored within the flash memory device's internal microcontroller or within a microcontroller interfacing between the flash memory device and the host, and an indication may be stored of the specific page or erase sector associated with the cells.

It is appreciated that the teachings of the present invention can, for example, be implemented by suitably modifying, or interfacing externally with, flash controlling apparatus. The flash controlling apparatus controls a flash memory array and may comprise either a controller external to the flash array or a microcontroller on-board the flash array or otherwise incorporated therewithin. Examples of flash memory arrays include Samsung's K9XXG08UXM series, Hynix' HY27UK08BGFM Series, Micron's MT29F64G08TAAWP or other arrays such as but not limited to NOR or phase change memory. Examples of controllers which are external to the flash array they control include STMicroelectrocincs's ST7265x microcontroller family, STMicroelectrocincs's ST72681 microcontroller, and SMSC's USB97C242, Traspan Technologies' TS-4811, Chipsbank CBM2090/CBM1190. Examples of commercial IP software for Flash file systems are: Denali's Spectra™ NAND Flash File System, Aarsan's NAND Flash Controller IP Core and Arasan's NAND Flash File System. It is appreciated that the flash controller apparatus need not be NAND-type and can alternatively, for example, be NOR-type or phase change memory-type.

A Flash controlling apparatus, whether external or internal to the controlled flash array, typically includes the following components: a Memory Management/File system, a NAND interface (or other flash memory array interface), a Host Interface (USB, SD or other), error correction circuitry (ECC) typically comprising an Encoder and matching decoder, and a control system managing all of the above.

The present invention may for example interface with or modify, as per any of the embodiments described herein, one, some or all of the above components and particularly the ECC and memory management components. The memory management component, or a functional unit interacting therewith, is, according to certain embodiments of the present invention, able to handle different size s of encoded pages over more than one physical page.

Features of the present invention which are described in the context of separate embodiments may also be provided in combination in a single embodiment. Conversely, features of the invention, including method steps, which are described for brevity in the context of a single embodiment or in a certain order may be provided separately or in any suitable subcombination or in a different order. "e.g." is used herein in the sense of a specific example which is not intended to be limiting.

CLAIMS

1. A method for encoding information arriving from a host in order to store said coded information in flash memory, the method comprising:

encoding information arriving from a host for storage at a flash memory location including generating a number of redundancy bytes, the encoding proceeding at an encoding rate which is a function of the number of redundancy bytes to be generated, said encoding including:

determining an effective error rate, including an anticipated rate of expected reading errors, for the flash memory location; and

selecting the encoding rate as a function of said effective error rate such that said number of redundancy bytes is sufficient to overcome said anticipated rate of expected reading errors with a predetermined degree of certainty.

2. A method according to claim 1 wherein said flash memory location comprises a page.

3. A method according to claim 1 wherein said determining comprises determining a rate of expected reading errors assuming reading occurs after a time period T, based on known data on frequency of errors when information is read from a flash memory which has undergone a simulation of effects of passage of said time period T.

4. A method according to claim 1 wherein said determining comprises reading a stored effective error rate, identifying a time t at which said effective error rate was stored, and updating said effective error rate as a function of said stored effective error rate and said time, to take into account spontaneous recovery of the flash memory location, over time, from effects of some of the program cycles to which the flash memory location has been subjected.

5. A method according to claim 4 wherein said updating comprises finding, for at least one stored number of program cycles to which the flash memory location was subjected at time t, a number of program cycles to which the flash memory location is

subjected currently, after which the rate of errors currently read from said flash memory location equals the rate of errors currently read from said flash memory location which was subjected to said stored number of program cycles at time t.

6. A method according to claim 1 wherein said flash memory device comprises an NROM flash memory device.

7. In a computer having an operating system, a storage unit comprising flash memory, and an operating system/storage unit interface providing interface between the storage unit and the operating system, the operating system being operative to query the storage unit to ascertain locations of bad flash memory portions, the interface including a bad location indicator operative to provide the operating system with an indication of said locations of bad flash memory portions,

a flash memory controller operative to receive at least one of read commands, write commands, and status commands from a user each said command associated with a logical address, to translate each logical address into a physical address within said flash memory, and to translate said commands into at least one of read, erase and program instructions for the flash memory, and wherein the flash memory controller is operative to determine an encoding rate having fluctuations and accordingly and responsively to said querying, to declare as bad, at least one portion of said flash memory, which was previously devoted to original data and which due to said fluctuations is now devoted to redundancy bytes.

8. A method according to claim 4 wherein said identifying a time t at which said effective error rate was stored, comprises associating a timer with said flash memory location, capable of generating a time stamp each time an operation is performed upon said flash memory location and storing at least one said time stamp.

9. A method according to claim 8 wherein said operation comprises a flash memory location reading operation.

10. A method according to claim 4 wherein said identifying a time t at which said effective error rate was stored comprises reading said flash memory location including quantifying and correcting errors encountered, and estimating the time of last programming as a function of said quantification of errors encountered.

11. A method according to claim 1 wherein said effective error rate for an individual flash memory location is stored adjacent said individual flash memory location.

12. A method according to claim 11 wherein said flash memory location comprises a page and said effective error rate for said page is stored within said page.

13. A method according to claim 12 wherein said effective error rate is encoded at a fixed coding rate corresponding to a fixed number of redundancy bytes which is high enough to overcome expected corruptions of said effective error rate.

14. A method according to claim 1 wherein said encoding, determining and selecting are performed for a plurality of flash memory locations.

15. High cycle count flash memory apparatus having a cycle count C, the apparatus comprising:

a set of flash memory devices including at least one low cycle-count flash device having a cycle count $c < C$ storing original data elements;

at least one additional flash device sufficient in size to store redundancy bytes sufficient in number to ensure that if information is encoded into said set of flash devices, with redundancy, thereby to generate a set of redundancy bytes, said additional flash device is sufficiently large to store a set of redundancy bytes sufficiently large to enable said information to be decoded at at least a predetermined level of accuracy; and

apparatus for reading at least one original data element from said low cycle flash memory device, and, in parallel, for reading at least one redundancy byte generated in the course of encoding said original data element, from said additional flash device.

16. Apparatus according to claim 15 wherein said set of flash memory devices includes more than one flash device and wherein said at least one additional flash device stores redundancy bytes for said more than one flash device.

17. A system for encoding information arriving from a host in order to store said coded information in flash memory, the system comprising:

a variable rate encoder operative to encode information arriving from a host for storage at a flash memory location including generating a number of redundancy bytes, the encoding proceeding at an encoding rate which is a function of the number of redundancy bytes generated, said variable rate encoder including:

an error rate definer operative to determine an effective error rate, including an anticipated rate of expected reading errors, for the flash memory location; and

an encoding rate definer selecting the encoding rate as a function of said effective error rate such that said number of redundancy bytes is sufficient to overcome said anticipated rate of expected reading errors with a predetermined degree of certainty.

18. A method for reading in high cycle count flash memory apparatus having a cycle count C, the method comprising:

providing a set of flash memory devices including at least one low cycle-count flash device having a cycle count $c < C$ storing original data elements;

providing at least one additional flash device sufficient in size to store redundancy bytes sufficient in number to ensure that if information is encoded into said set of flash devices, with redundancy, thereby to generate a set of redundancy bytes, said additional flash device is sufficiently large to store a set of redundancy bytes sufficiently large to enable said information to be decoded at at least a predetermined level of accuracy; and

reading at least one original data element from said low cycle flash memory device, and, in parallel, reading at least one redundancy byte generated in the course of encoding said original data element, from said additional flash device.

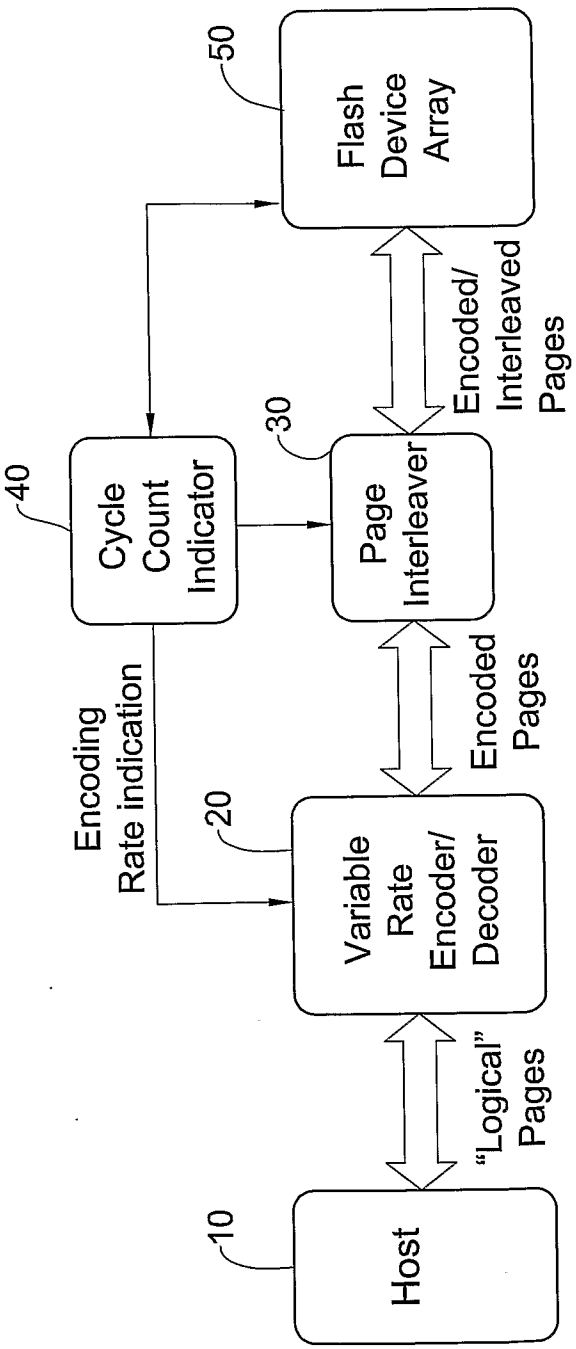


Figure 1

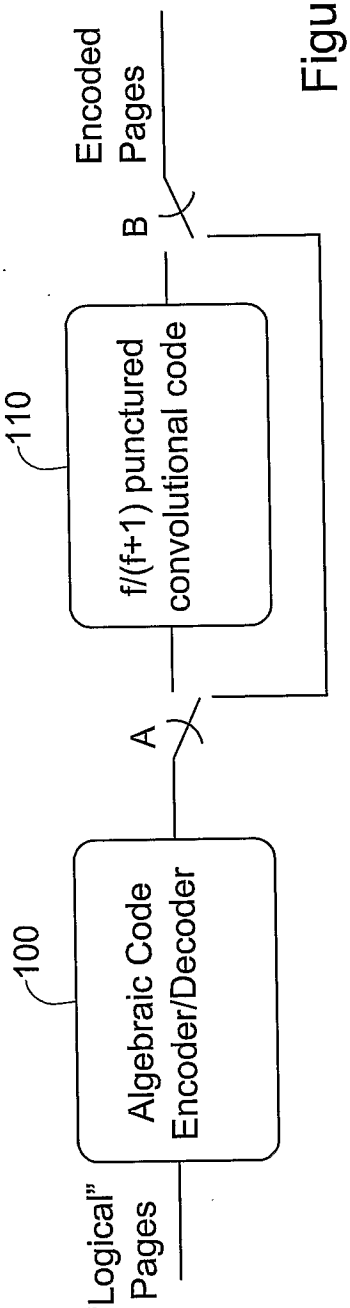


Figure 2

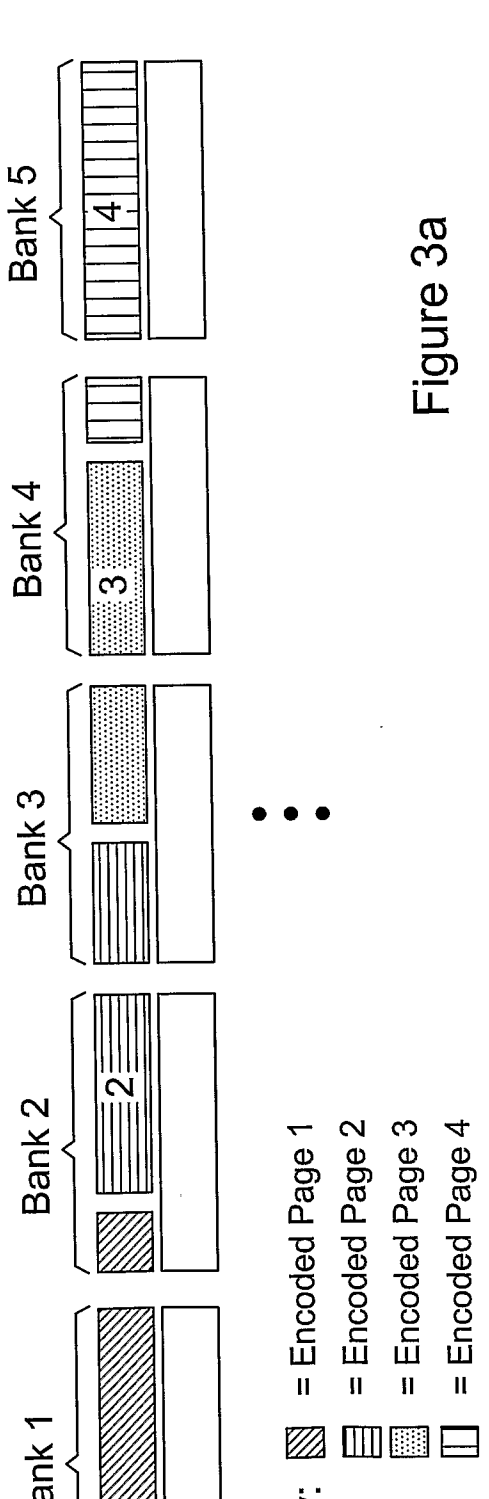


Figure 3a

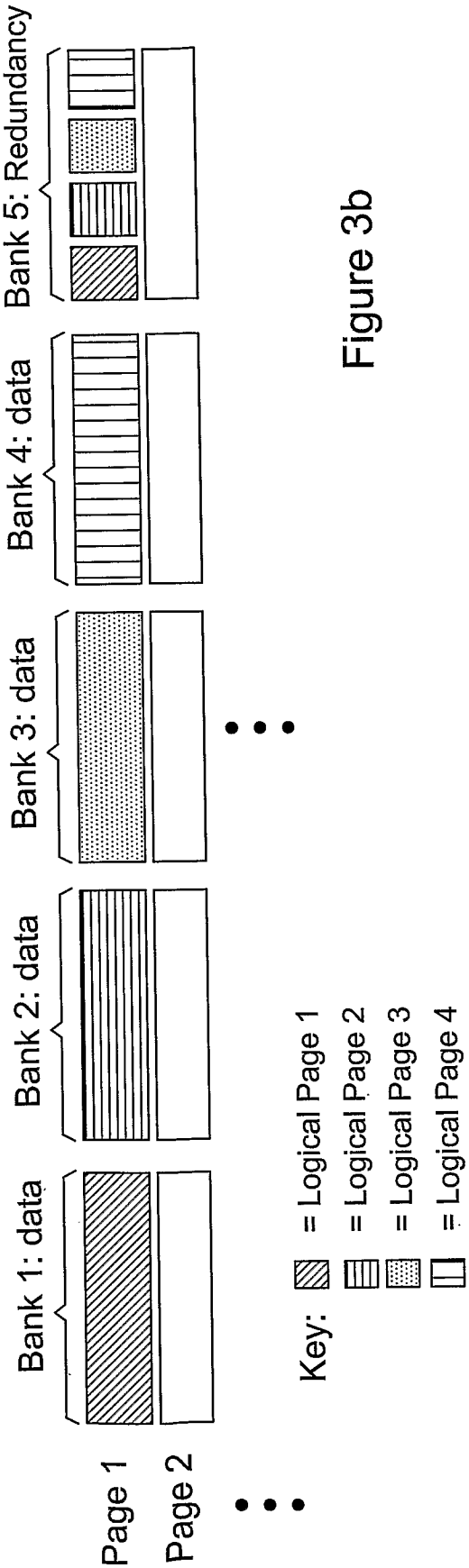


Figure 3b

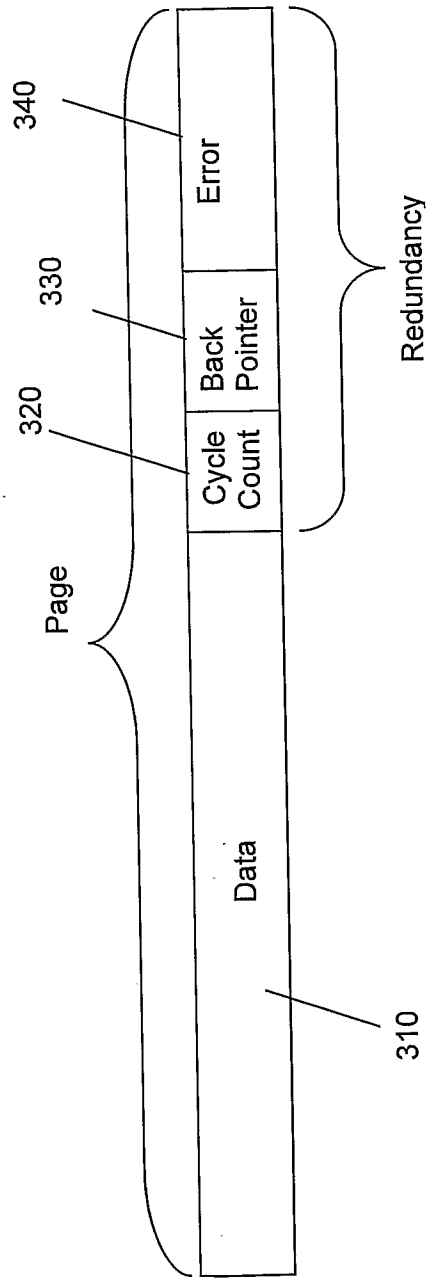

340
330
320
Error
Back Pointer
Cycle Count
Redundancy
Page
Data
310

Figure 4

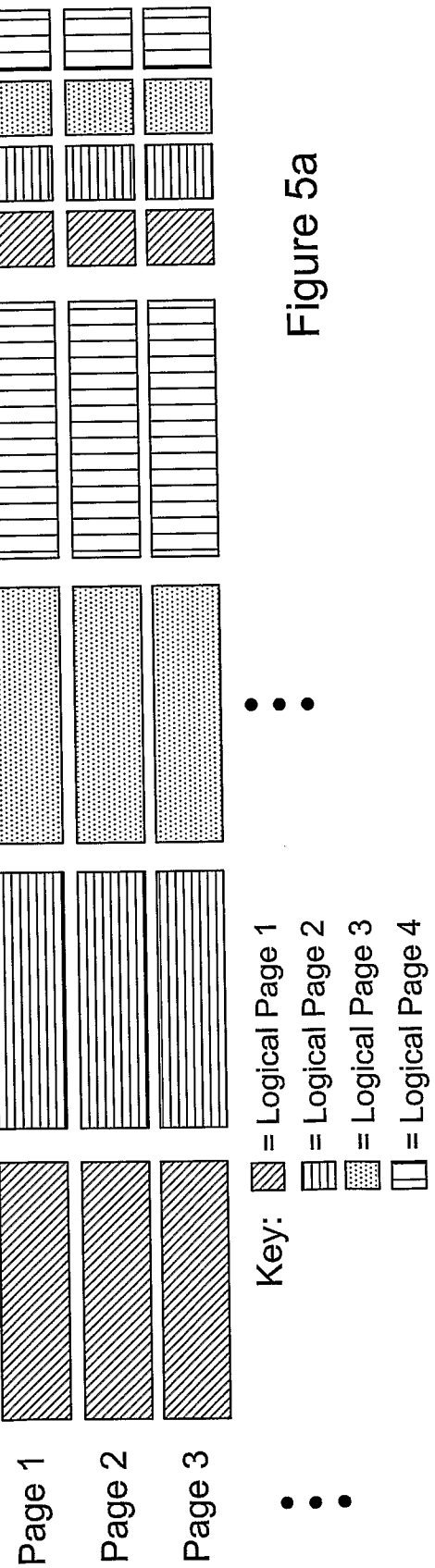


Figure 5a

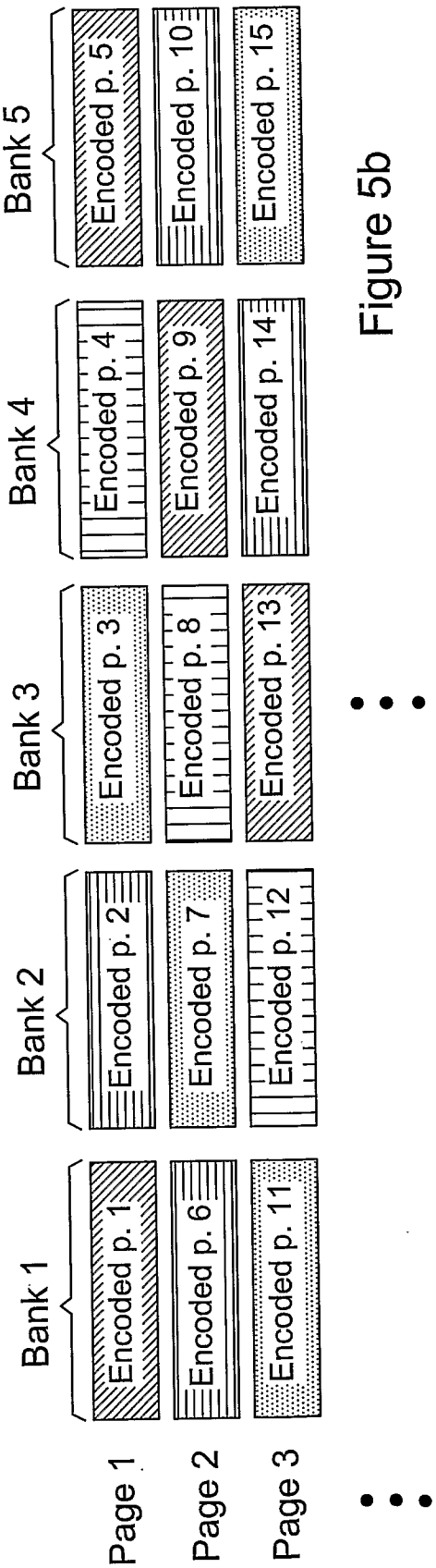


Figure 5b

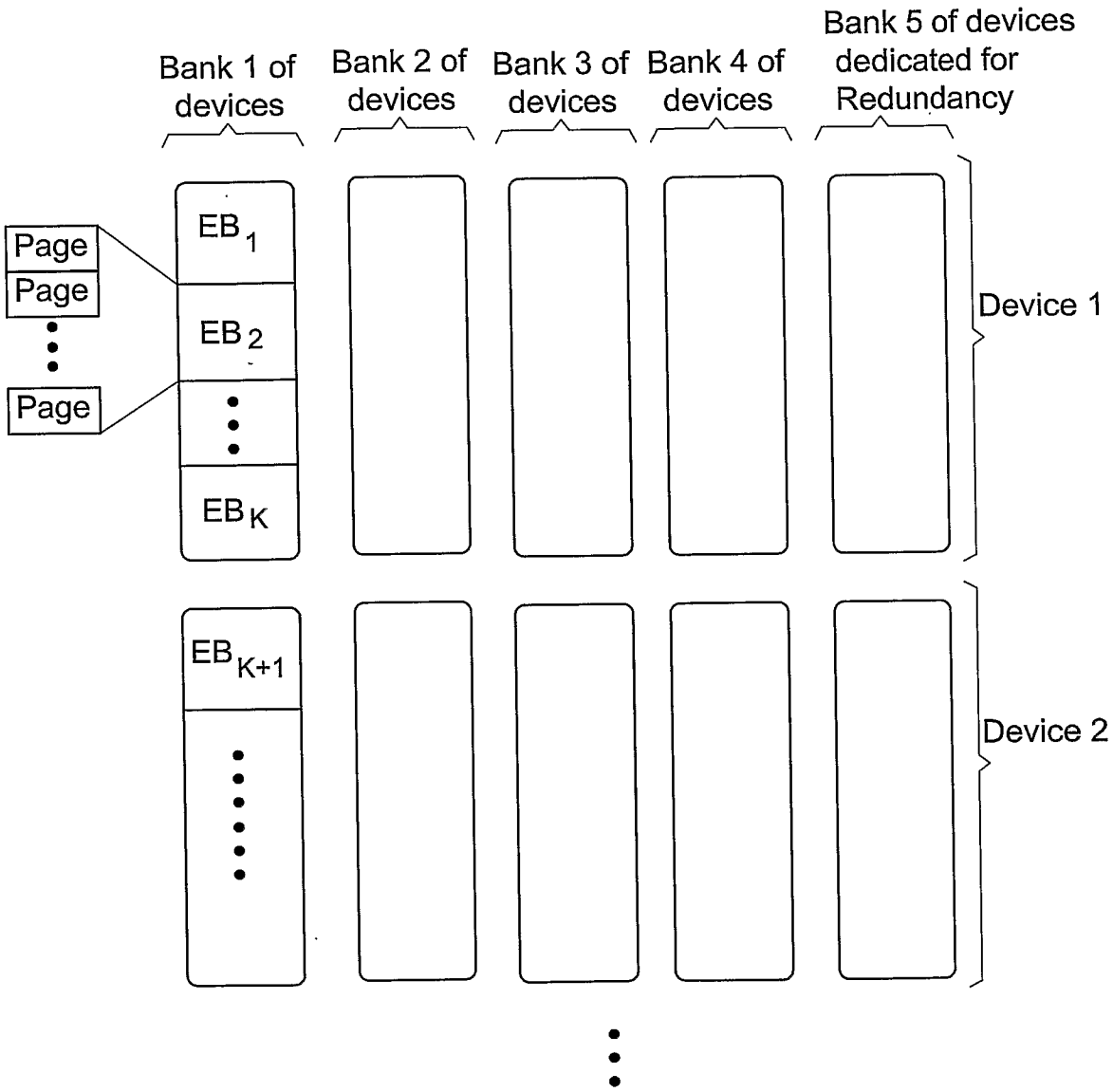

Bank 1 of devices
Bank 2 of devices
Bank 3 of devices
Bank 4 of devices
Bank 5 of devices dedicated for Redundancy
Page
Page
Page
EB 1
EB 2
EB K
EB K+1
Device 1
Device 2

Figure 6

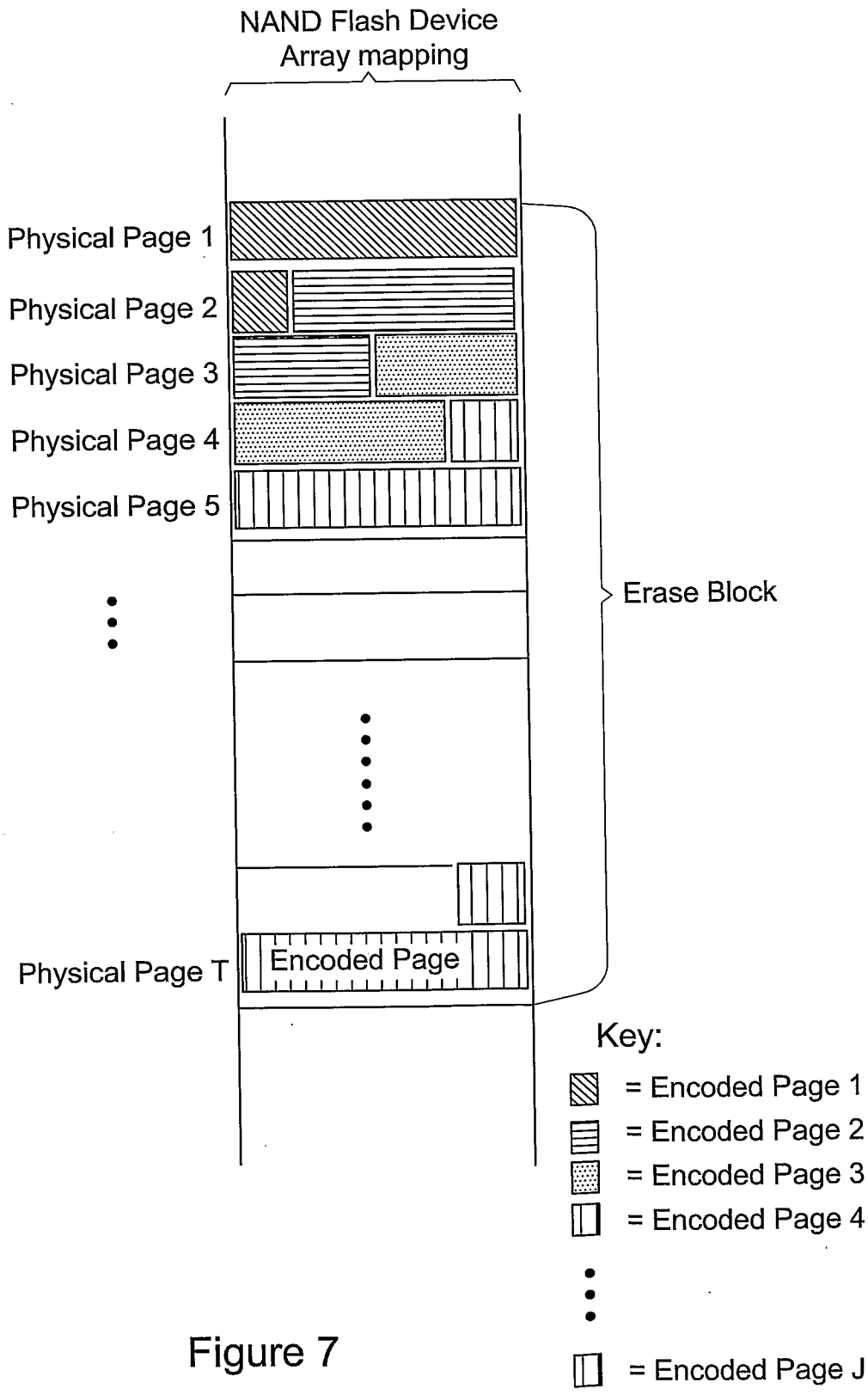

NAND Flash Device
Array mapping
Physical Page 1
Physical Page 2
Physical Page 3
Physical Page 4
Physical Page 5
Erase Block
Physical Page T
Encoded Page
Key:
= Encoded Page 1
= Encoded Page 2
= Encoded Page 3
= Encoded Page 4
= Encoded Page J

Figure 7

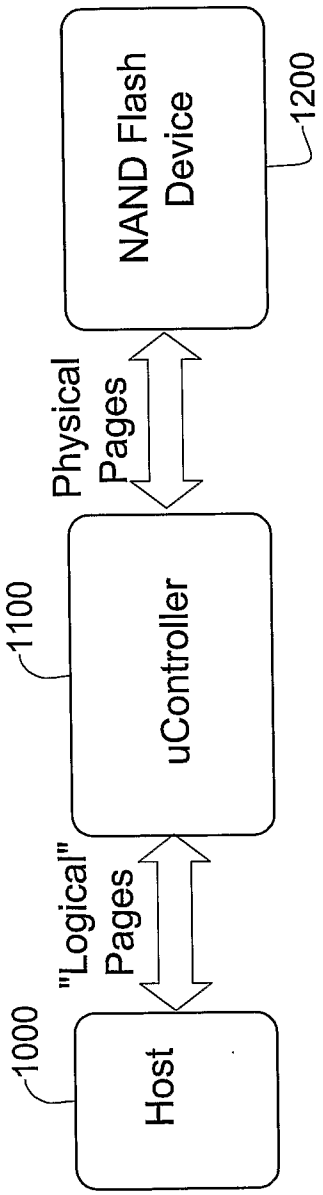

NAND Flash Device
1200
Physical Pages
uController
1100
"Logical" Pages
Host
1000

Figure 8

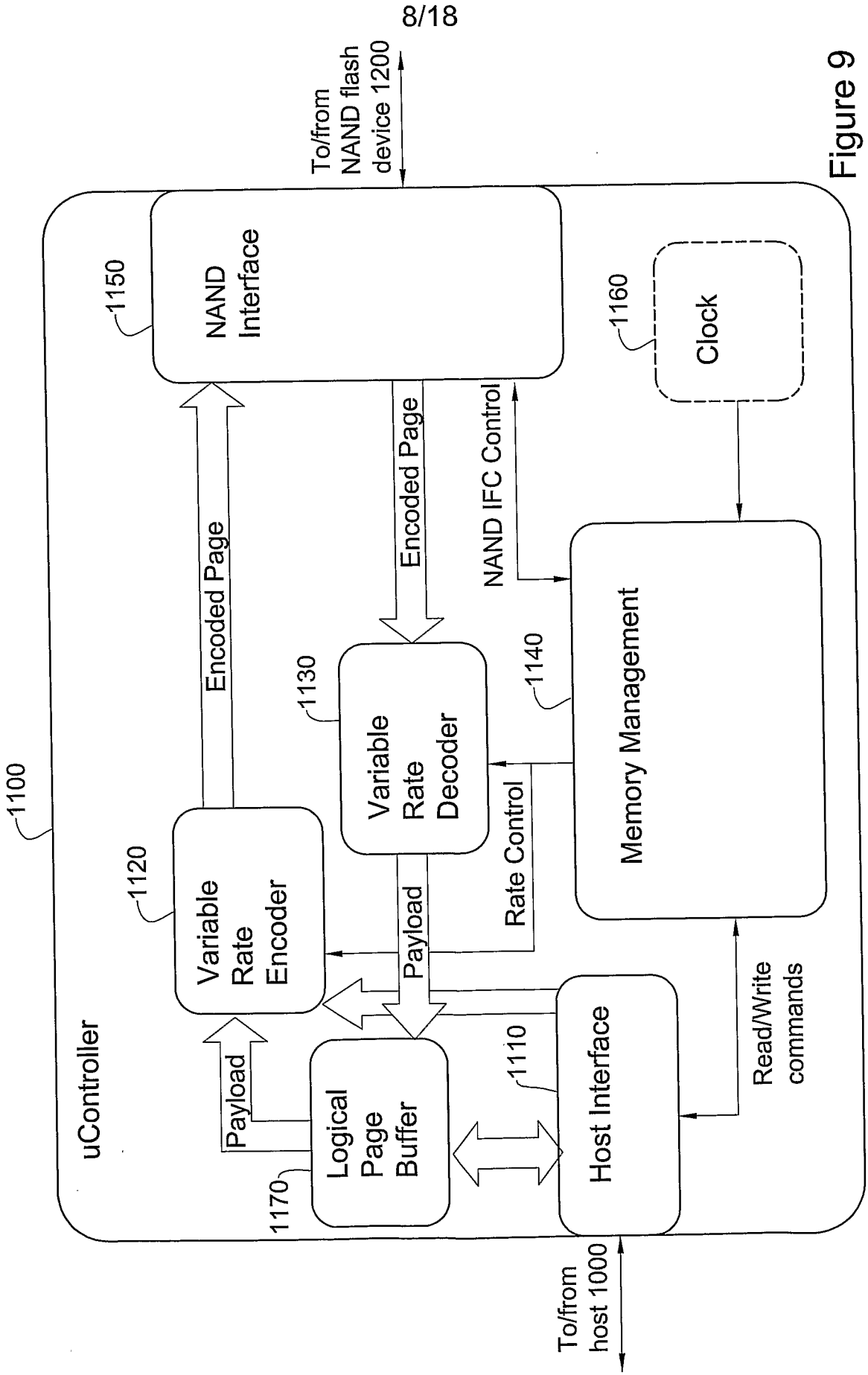

To/from NAND flash device 1200
1100
uController
1150
NAND Interface
1160
Clock
Encoded Page
Encoded Page
NAND IFC Control
1130
Variable Rate Decoder
1140
Memory Management
1120
Variable Rate Encoder
Payload
Rate Control
Payload
1110
Logical Page Buffer
Host Interface
Read/Write commands
1170
To/from host 1000

Figure 9

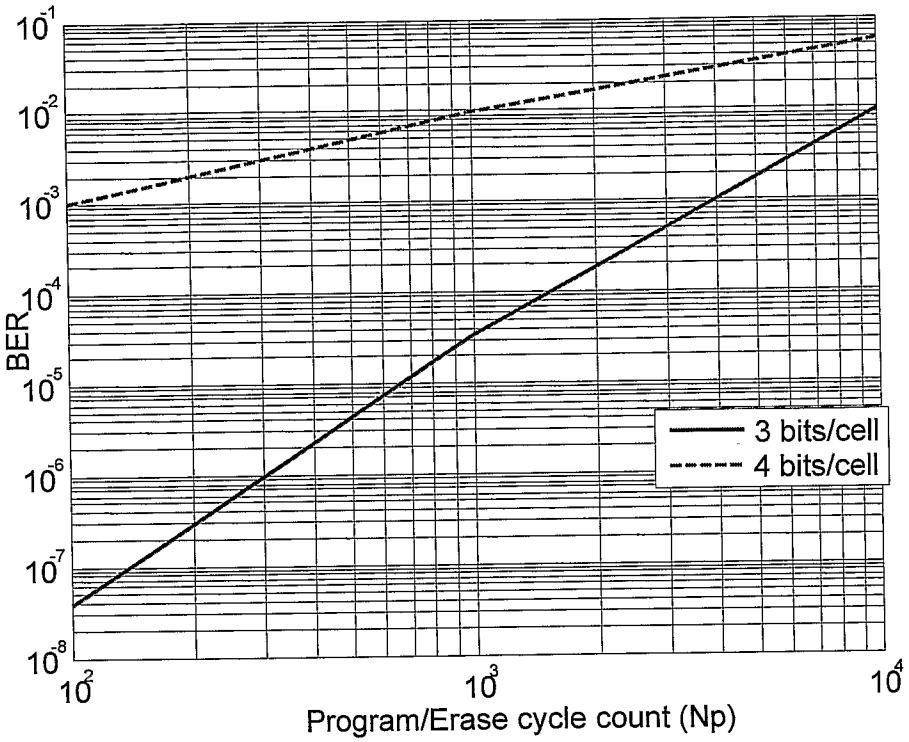

BER
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
10^-7
10^-8
10^2
10^3
10^4
Program/Erase cycle count (Np)
3 bits/cell
4 bits/cell

Figure 10

Physical Block #	Cycle Count (Np)	# of "logical" pages in Block (#PagesPerBlock)	Allocated Logical Pages	Time Stamp
0	10	128	128	1 Year ago
1	7951	64	64	1 minute ago
2	917	128	128	10 Hours ago
3	3715	96	96	9 Hours ago
4	4000	96	0	5 Hours ago

Figure 11

Logical page Address	Block #	Encoded Page # of first logical page (FirstPage#)	Logical Pages Programmed (OccupyVec) (32 bits)
0	0	0	011011...
32	0	32	101111...
64	0	64	11111...
96	0	96	11111...
128	2	0	011011...
160	2	32	101111...
192	2	64	11111...
224	2	96	11111...
256	3	0	011011...
288	3	32	101111...
320	3	64	11111...
352	4	0	11111...
384	4	32	11111...
416	unallocated		

Figure 12

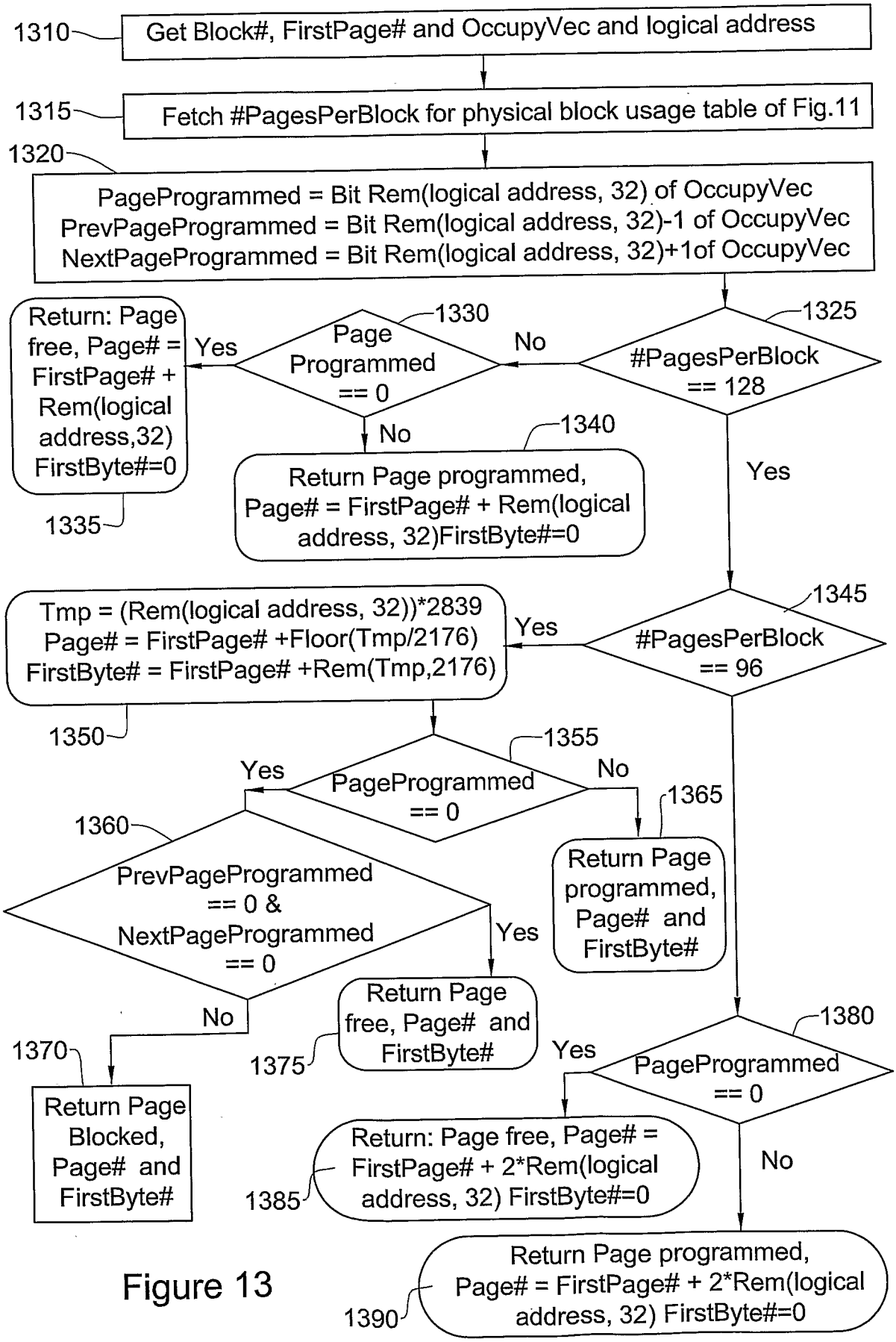

1310
Get Block#, FirstPage# and OccupyVec and logical address
1315
Fetch #PagesPerBlock for physical block usage table of Fig.11
1320
PageProgrammed = Bit Rem(logical address, 32) of OccupyVec
PrevPageProgrammed = Bit Rem(logical address, 32)-1 of OccupyVec
NextPageProgrammed = Bit Rem(logical address, 32)+1of OccupyVec
1325
#PagesPerBlock == 128
1330
Page Programmed == 0
Yes
No
Return: Page free, Page# = FirstPage# + Rem(logical address,32) FirstByte#=0
1335
No
1340
Return Page programmed, Page# = FirstPage# + Rem(logical address, 32)FirstByte#=0
Yes
1345
#PagesPerBlock == 96
Yes
Tmp = (Rem(logical address, 32))*2839
Page# = FirstPage# +Floor(Tmp/2176)
FirstByte# = FirstPage# +Rem(Tmp,2176)
1350
1355
PageProgrammed == 0
Yes
No
1360
PrevPageProgrammed == 0 & NextPageProgrammed == 0
Yes
No
1365
Return Page programmed, Page# and FirstByte#
Return Page free, Page# and FirstByte#
1375
1370
Return Page Blocked, Page# and FirstByte#
1380
PageProgrammed == 0
Yes
No
Return: Page free, Page# = FirstPage# + 2*Rem(logical address, 32) FirstByte#=0
1385
Return Page programmed, Page# = FirstPage# + 2*Rem(logical address, 32) FirstByte#=0
1390

Figure 13

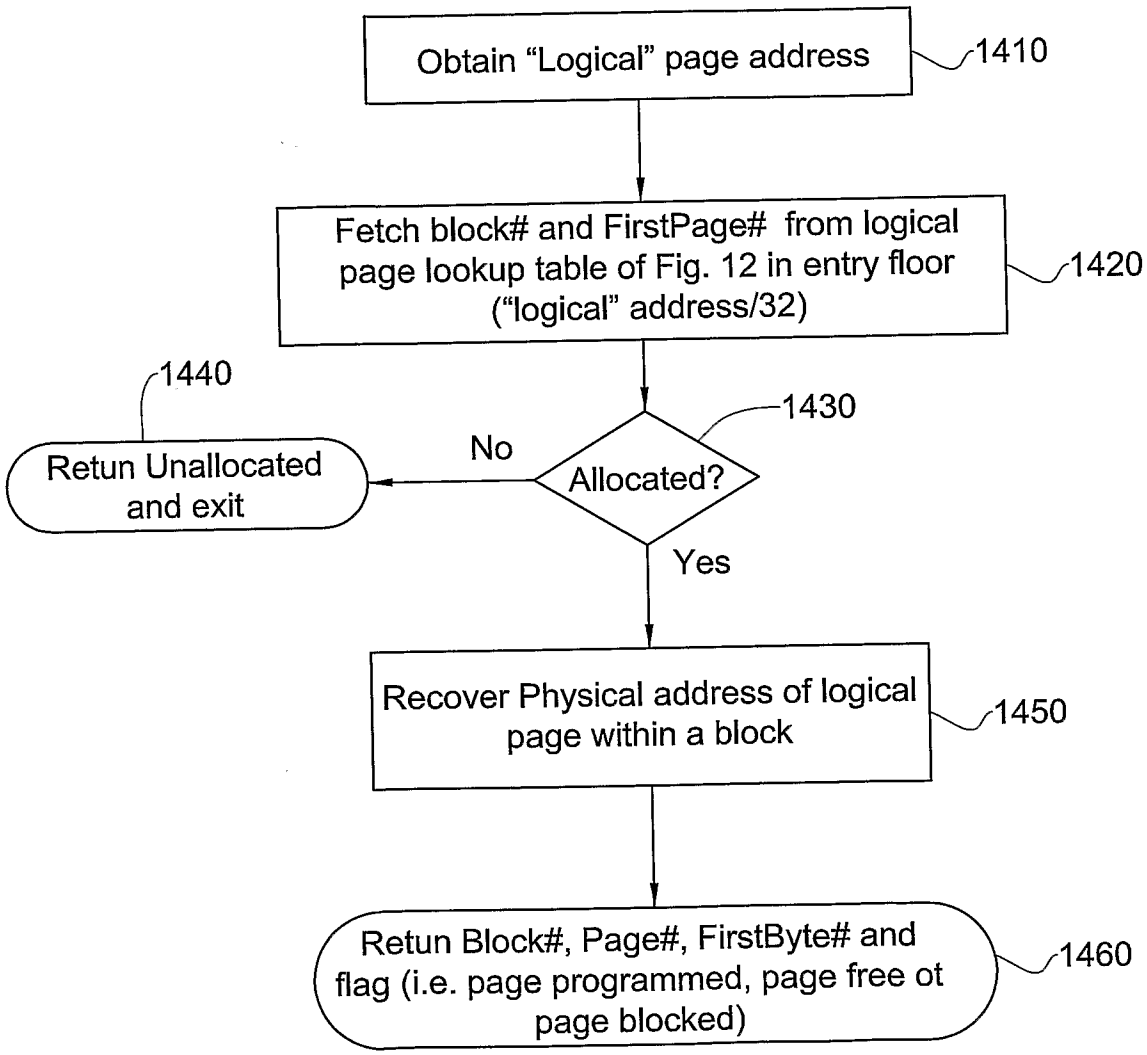

Obtain "Logical" page address
1410
Fetch block# and FirstPage# from logical page lookup table of Fig. 12 in entry floor ("logical" address/32)
1420
1430
Allocated?
No
1440
Retun Unallocated and exit
Yes
Recover Physical address of logical page within a block
1450
Retun Block#, Page#, FirstByte# and flag (i.e. page programmed, page free ot page blocked)
1460

Figure 14

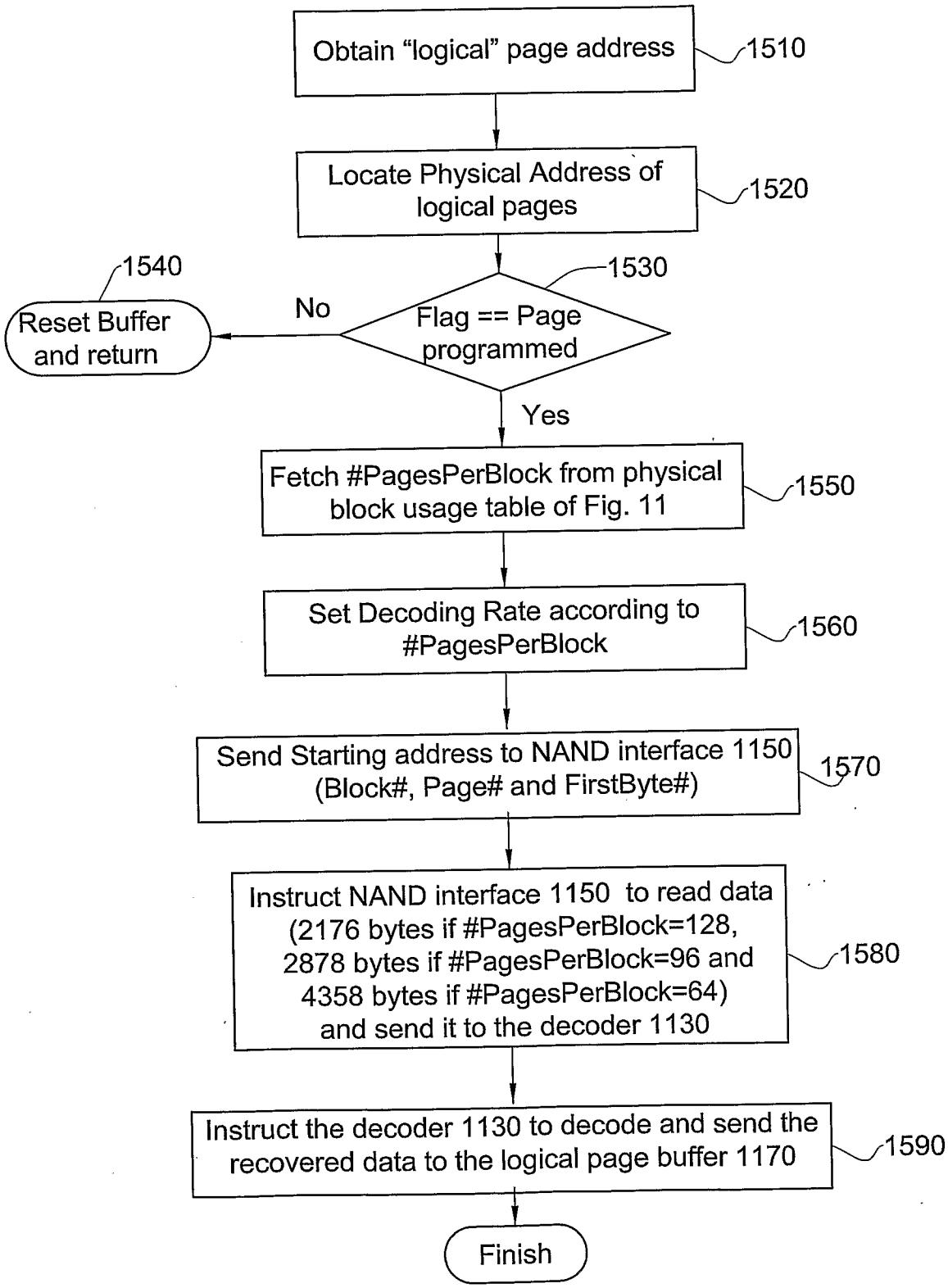

Obtain "logical" page address
1510
Locate Physical Address of logical pages
1520
1530
Flag == Page programmed
No
1540
Reset Buffer and return
Yes
Fetch #PagesPerBlock from physical block usage table of Fig. 11
1550
Set Decoding Rate according to #PagesPerBlock
1560
Send Starting address to NAND interface 1150 (Block#, Page# and FirstByte#)
1570
Instruct NAND interface 1150 to read data (2176 bytes if #PagesPerBlock=128, 2878 bytes if #PagesPerBlock=96 and 4358 bytes if #PagesPerBlock=64) and send it to the decoder 1130
1580
Instruct the decoder 1130 to decode and send the recovered data to the logical page buffer 1170
1590
Finish

Figure 15

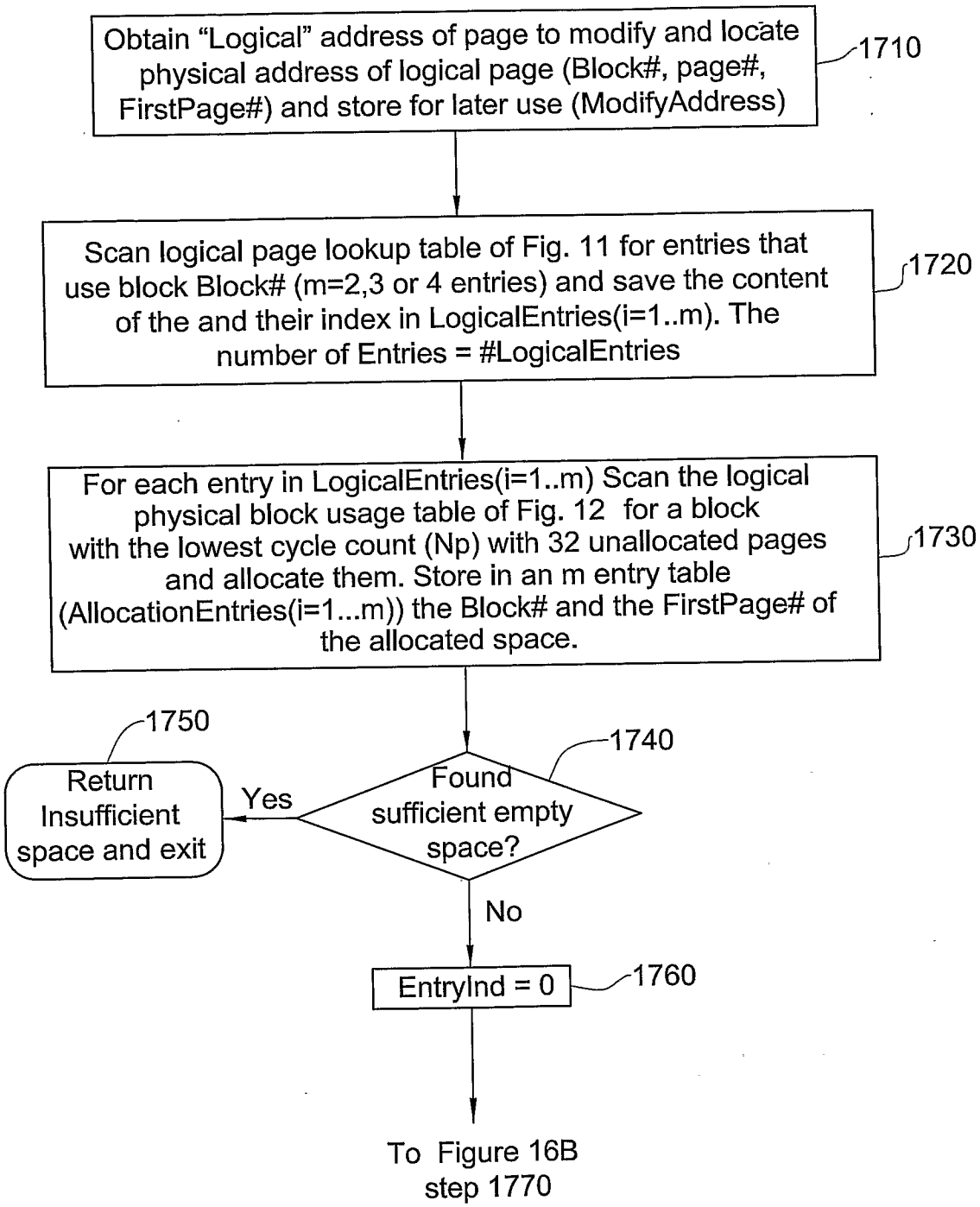

Obtain "Logical" address of page to modify and locate physical address of logical page (Block#, page#, FirstPage#) and store for later use (ModifyAddress)
1710
Scan logical page lookup table of Fig. 11 for entries that use block Block# (m=2,3 or 4 entries) and save the content of the and their index in LogicalEntries(i=1..m). The number of Entries = #LogicalEntries
1720
For each entry in LogicalEntries(i=1..m) Scan the logical physical block usage table of Fig. 12 for a block with the lowest cycle count (Np) with 32 unallocated pages and allocate them. Store in an m entry table (AllocationEntries(i=1...m)) the Block# and the FirstPage# of the allocated space.
1730
1750
Return Insufficient space and exit
Yes
1740
Found sufficient empty space?
No
EntryInd = 0
1760
To Figure 16B step 1770

Figure 16A

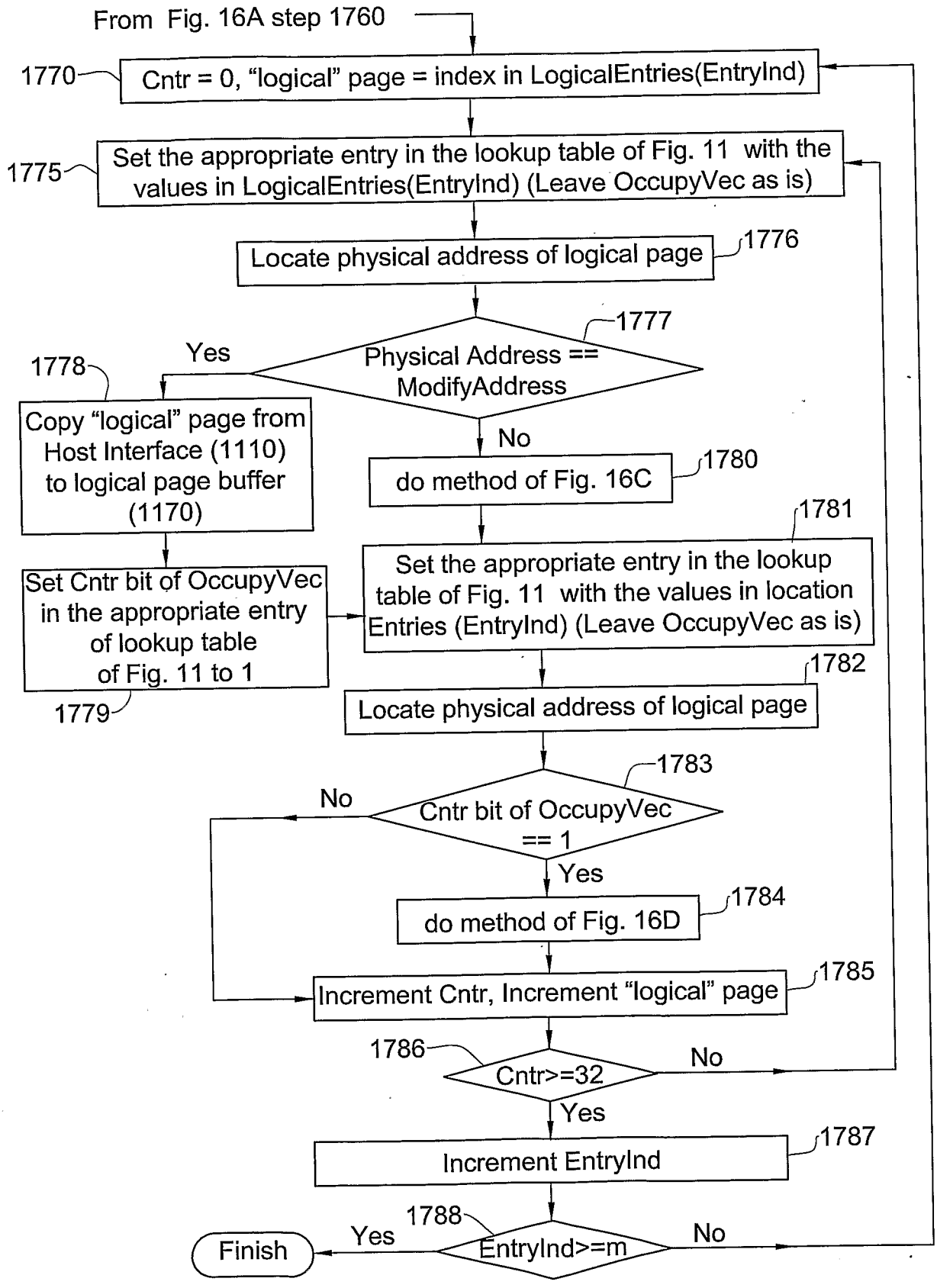

From Fig. 16A step 1760
1770
Cntr = 0, "logical" page = index in LogicalEntries(EntryInd)
1775
Set the appropriate entry in the lookup table of Fig. 11 with the values in LogicalEntries(EntryInd) (Leave OccupyVec as is)
Locate physical address of logical page
1776
1777
Physical Address == ModifyAddress
Yes
1778
Copy "logical" page from Host Interface (1110) to logical page buffer (1170)
No
do method of Fig. 16C
1780
1781
Set the appropriate entry in the lookup table of Fig. 11 with the values in location Entries (EntryInd) (Leave OccupyVec as is)
Set Cntr bit of OccupyVec in the appropriate entry of lookup table of Fig. 11 to 1
1779
1782
Locate physical address of logical page
1783
Cntr bit of OccupyVec == 1
No
Yes
do method of Fig. 16D
1784
Increment Cntr, Increment "logical" page
1785
1786
Cntr>=32
No
Yes
Increment EntryInd
1787
1788
EntryInd>=m
Yes
No
Finish

Figure 16B

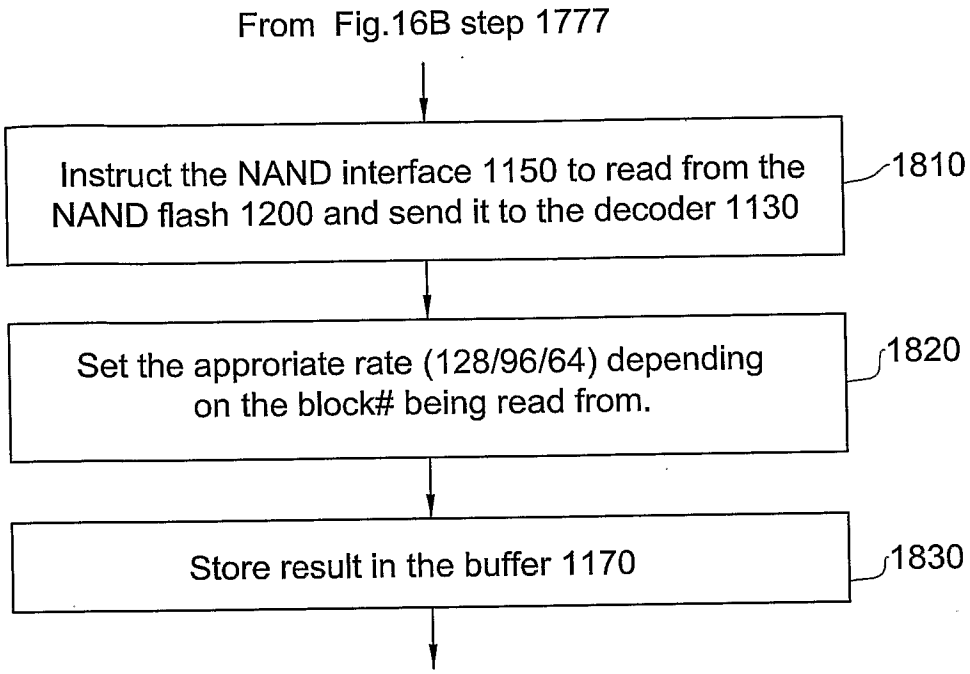


Figure 16C

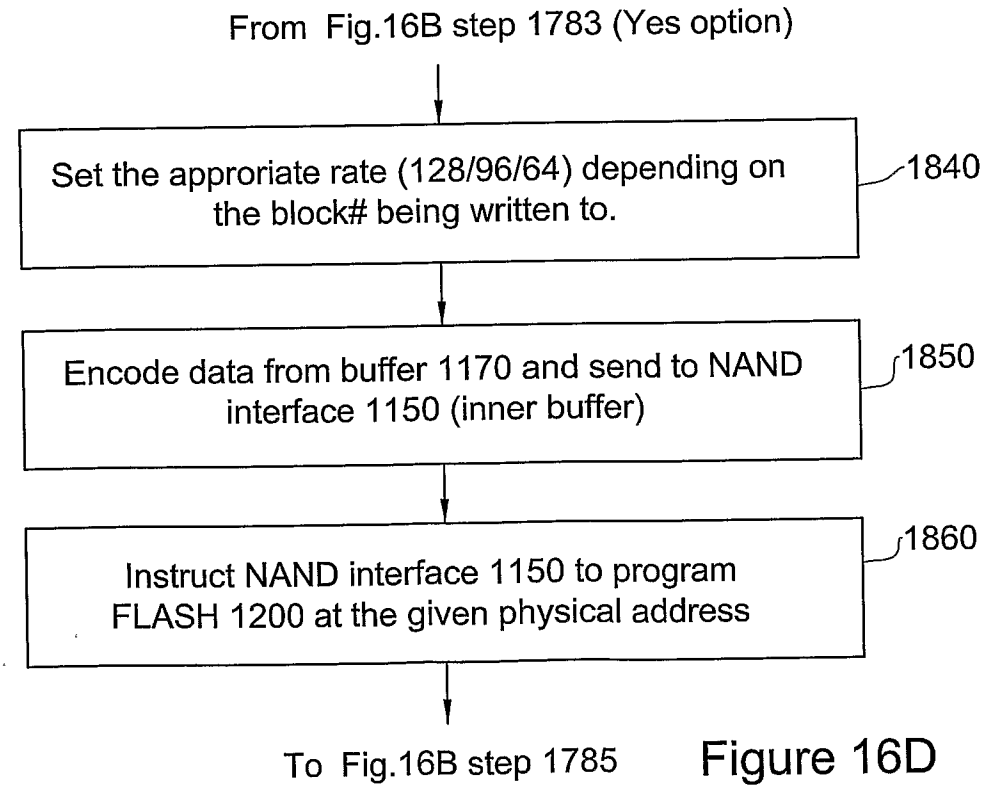


Figure 16D

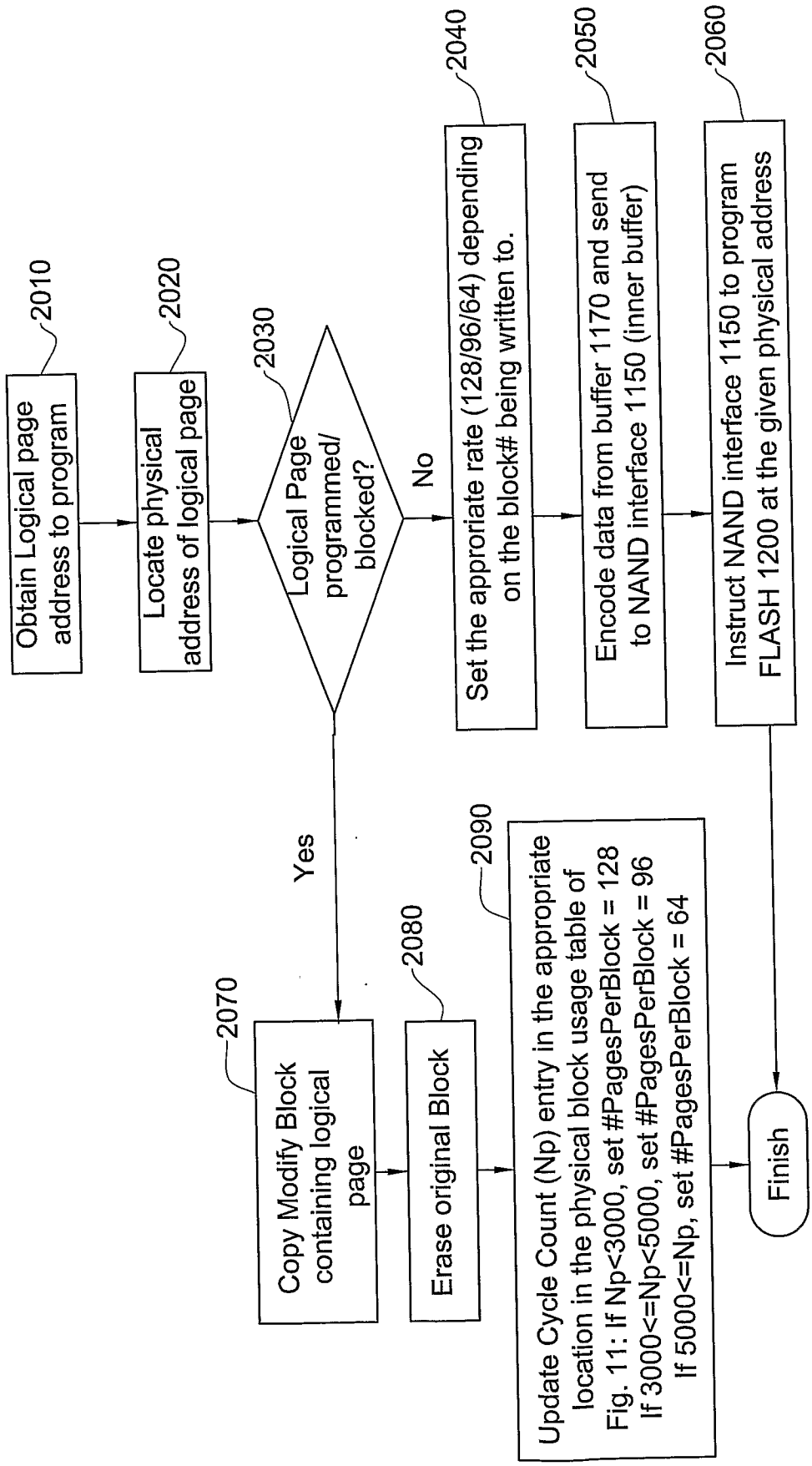

Obtain Logical page address to program
2010
Locate physical address of logical page
2020
Logical Page programmed/ blocked?
2030
No
Yes
Set the approriate rate (128/96/64) depending on the block# being written to.
2040
Encode data from buffer 1170 and send to NAND interface 1150 (inner buffer)
2050
Instruct NAND interface 1150 to program FLASH 1200 at the given physical address
2060
Copy Modify Block containing logical page
2070
Erase original Block
2080
Update Cycle Count (Np) entry in the appropriate location in the physical block usage table of Fig. 11: If Np<3000, set #PagesPerBlock = 128 If 3000<=Np<5000, set #PagesPerBlock = 96 If 5000<=Np, set #PagesPerBlock = 64
2090
Finish

Figure 17