(54) Title: MULTI-BANK NON-VOLATILE MEMORY SYSTEM WITH SATELLITE FILE SYSTEM

(57) Abstract: A multi-bank non-volatile memory system is presented. A first of the banks has a main copy of the file system and each of the other banks has a satellite copy of the file system. The back end firmware of the controller executes a thread for each of the banks. After the boot process, during normal memory operations, each of the threads can operate using its own copy of the file system without interrupting the other threads in order to access the file system.
MULTI-BANK NON-VOLATILE MEMORY SYSTEM
WITH SATELLITE FILE SYSTEM

BACKGROUND AND SUMMARY

[0001] This application relates to the operation of re-programmable non-volatile memory systems such as semiconductor flash memory, and, more specifically, to the management of such system that employ a multiple bank structure.

[0002] Solid-state memory capable of nonvolatile storage of charge, particularly in the form of EEPROM and flash EEPROM packaged as a small form factor card, has recently become the storage of choice in a variety of mobile and handheld devices, notably information appliances and consumer electronics products. Unlike RAM (random access memory) that is also solid-state memory, flash memory is non-volatile, and retaining its stored data even after power is turned off. Also, unlike ROM (read only memory), flash memory is rewritable similar to a disk storage device. In spite of the higher cost, flash memory is increasingly being used in mass storage applications. Conventional mass storage, based on rotating magnetic medium such as hard drives and floppy disks, is unsuitable for the mobile and handheld environment. This is because disk drives tend to be bulky, are prone to mechanical failure and have high latency and high power requirements. These undesirable attributes make disk-based storage impractical in most mobile and portable applications. On the other hand, flash memory, both embedded and in the form of a removable card is ideally suited in the mobile and handheld environment because of its small size, low power consumption, high speed and high reliability features.

[0003] Flash EEPROM is similar to EEPROM (electrically erasable and programmable read-only memory) in that it is a non-volatile memory that can be erased and have new data written or "programmed" into their memory cells. Both utilize a floating (unconnected) conductive gate, in a field effect transistor structure, positioned over a channel region in a semiconductor substrate, between source and drain regions. A control gate is then provided over the floating gate. The threshold
voltage characteristic of the transistor is controlled by the amount of charge that is retained on the floating gate. That is, for a given level of charge on the floating gate, there is a corresponding voltage (threshold) that must be applied to the control gate before the transistor is turned "on" to permit conduction between its source and drain regions. In particular, flash memory such as Flash EEPROM allows entire blocks of memory cells to be erased at the same time.

[0004] The floating gate can hold a range of charges and therefore can be programmed to any threshold voltage level within a threshold voltage window. The size of the threshold voltage window is delimited by the minimum and maximum threshold levels of the device, which in turn correspond to the range of the charges that can be programmed onto the floating gate. The threshold window generally depends on the memory device's characteristics, operating conditions and history. Each distinct, resolvable threshold voltage level range within the window may, in principle, be used to designate a definite memory state of the cell.

[0005] The transistor serving as a memory cell is typically programmed to a "programmed" state by one of two mechanisms. In "hot electron injection," a high voltage applied to the drain accelerates electrons across the substrate channel region. At the same time a high voltage applied to the control gate pulls the hot electrons through a thin gate dielectric onto the floating gate. In "tunneling injection," a high voltage is applied to the control gate relative to the substrate. In this way, electrons are pulled from the substrate to the intervening floating gate. While the term "program" has been used historically to describe writing to a memory by injecting electrons to an initially erased charge storage unit of the memory cell so as to alter the memory state, it has now been used interchangeable with more common terms such as "write" or "record."

[0006] The memory device may be erased by a number of mechanisms. For EEPROM, a memory cell is electrically erasable, by applying a high voltage to the substrate relative to the control gate so as to induce electrons in the floating gate to tunnel through a thin oxide to the substrate channel region (i.e., Fowler-Nordheim tunneling.) Typically, the EEPROM is erasable byte by byte. For flash EEPROM, the memory is electrically erasable either all at once or one or more minimum
erasable blocks at a time, where a minimum erasable block may consist of one or more sectors and each sector may store 512 bytes or more of data.

[0007] The memory device typically comprises one or more memory chips that may be mounted on a card. Each memory chip comprises an array of memory cells supported by peripheral circuits such as decoders and erase, write and read circuits. The more sophisticated memory devices also come with a controller that performs intelligent and higher level memory operations and interfacing.

[0008] There are many commercially successful non-volatile solid-state memory devices being used today. These memory devices may be flash EEPROM or may employ other types of nonvolatile memory cells. Examples of flash memory and systems and methods of manufacturing them are given in United States patents nos. 5,070,032, 5,095,344, 5,315,541, 5,343,063, and 5,661,053, 5,313,421 and 6,222,762. In particular, flash memory devices with NAND string structures are described in United States patent nos. 5,570,315, 5,903,495, 6,046,935. Also nonvolatile memory devices are also manufactured from memory cells with a dielectric layer for storing charge. Instead of the conductive floating gate elements described earlier, a dielectric layer is used. Such memory devices utilizing dielectric storage element have been described by Eitan et al., "NROM: A Novel Localized Trapping, 2-Bit Nonvolatile Memory Cell," IEEE Electron Device Letters, vol. 21, no. 11. November 2000, pp. 543-545. An ONO dielectric layer extends across the channel between source and drain diffusions. The charge for one data bit is localized in the dielectric layer adjacent to the drain, and the charge for the other data bit is localized in the dielectric layer adjacent to the source. For example, United States patents nos. 5,768,192 and 6,01 1,725 disclose a nonvolatile memory cell having a trapping dielectric sandwiched between two silicon dioxide layers. Multi-state data storage is implemented by separately reading the binary states of the spatially separated charge storage regions within the dielectric.

[0009] In order to improve read and program performance, multiple charge storage elements or memory transistors in an array are read or programmed in parallel. Thus, a "page" of memory elements are read or programmed together. In existing memory architectures, a row typically contains several interleaved pages or it may constitute
one page. All memory elements of a page will be read or programmed together.

[0010] In flash memory systems, erase operation may take as much as an order of magnitude longer than read and program operations. Thus, it is desirable to have the erase block of substantial size. In this way, the erase time is amortized over a large aggregate of memory cells.

[0011] The nature of flash memory predicates that data must be written to an erased memory location. If data of a certain logical address from a host is to be updated, one way is rewrite the update data in the same physical memory location. That is, the logical to physical address mapping is unchanged. However, this will mean the entire erase block contain that physical location will have to be first erased and then rewritten with the updated data. This method of update is inefficient, as it requires an entire erase block to be erased and rewritten, especially if the data to be updated only occupies a small portion of the erase block. It will also result in a higher frequency of erase recycling of the memory block, which is undesirable in view of the limited endurance of this type of memory device.

[0012] Data communicated through external interfaces of host systems, memory systems and other electronic systems are addressed and mapped into the physical locations of a flash memory system. Typically, addresses of data files generated or received by the system are mapped into distinct ranges of a continuous logical address space established for the system in terms of logical blocks of data (hereinafter the "LBA interface"). The extent of the address space is typically sufficient to cover the full range of addresses that the system is capable of handling. In one example, magnetic disk storage drives communicate with computers or other host systems through such a logical address space. This address space has an extent sufficient to address the entire data storage capacity of the disk drive.

[0013] Flash memory systems are most commonly provided in the form of a memory card or flash drive that is removably connected with a variety of hosts such as a personal computer, a camera or the like, but may also be embedded within such host systems. When writing data to the memory, the host typically assigns unique logical addresses to sectors, clusters or other units of data within a continuous virtual address space of the memory system. Like a disk operating system (DOS), the host writes
data to, and reads data from, addresses within the logical address space of the memory system. A controller within the memory system translates logical addresses received from the host into physical addresses within the memory array, where the data are actually stored, and then keeps track of these address translations. The data storage capacity of the memory system is at least as large as the amount of data that is addressable over the entire logical address space defined for the memory system.

[0014] In current commercial flash memory systems, the size of the erase unit has been increased to a block of enough memory cells to store multiple sectors of data. Indeed, many pages of data are stored in one block, and a page may store multiple sectors of data. Further, two or more blocks are often operated together as metablocks, and the pages of such blocks logically linked together as metapages. A page or metapage of data are written and read together, which can include many sectors of data, thus increasing the parallelism of the operation. Along with such large capacity operating units the challenge is to operate them efficiently.

[0015] For ease of explanation, unless otherwise specified, it is intended that the term "block" as used herein refer to either the block unit of erase or a multiple block "metablock," depending upon whether metablocks are being used in a specific system. Similarly, reference to a "page" herein may refer to a unit of programming within a single block or a "metapage" within a metablock, depending upon the system configuration.

[0016] When the currently prevalent LBA interface to the memory system is used, files generated by a host to which the memory is connected are assigned unique addresses within the logical address space of the interface. The memory system then commonly maps data between the logical address space and pages of the physical blocks of memory. The memory system keeps track of how the logical address space is mapped into the physical memory but the host is unaware of this. The host keeps track of the addresses of its data files within the logical address space but the memory system operates with little or no knowledge of this mapping.

[0017] Another problem with managing flash memory system has to do with system control and directory data. The data is produced and accessed during the course of various memory operations. Thus, its efficient handling and ready access will directly
impact performance. It would be desirable to maintain this type of data in flash memory because flash memory is meant for storage and is nonvolatile. However, with an intervening file management system between the controller and the flash memory, the data can not be accessed as directly. Also, system control and directory data tends to be active and fragmented, which is not conducive to storing in a system with large size block erase. Conventionally, this type of data is set up in the controller RAM, thereby allowing direct access by the controller. After the memory device is powered up, a process of initialization enables the flash memory to be scanned in order to compile the necessary system control and directory information to be placed in the controller RAM. This process takes time and requires controller RAM capacity, all the more so with ever increasing flash memory capacity.

In general, there is continuing search to improve the capacity and performance of non-volatile memory systems. In particular, this can include methods to improve the amount and efficiency of parallelism in memory systems.

**SUMMARY OF THE INVENTION**

According to a general aspect of the invention, a non-volatile memory system having a controller circuit and a non-volatile memory circuit with a plurality of independently operable banks is presented. Each of the banks includes one or more non-volatile memory arrays to store user data and system data. The controller circuit manages the storage of user data on the memory circuit and includes a plurality of bank interfaces, each connected a respective one of the banks to transfer data between the controller and the corresponding bank, and processing circuitry to execute a plurality of threads each independently and concurrently managing a corresponding one of the banks. A first of the banks stores system data including a file system by which the corresponding thread manages the first bank and each of the other banks stores system data including a copy of a portion of the file system by which the corresponding thread manages the bank.

In other aspects, corresponding methods of operating a non-volatile memory system including a non-volatile memory circuit, having a plurality of independently operable banks, and a controller circuit to manage the storage of user on the memory circuit are presented. The method includes: storing in non-volatile memory on a first
of the banks system data including a file system; storing in non-volatile memory on a second of the banks system data including a copy of a portion of the file system; executing on the controller a first thread to manage the first bank according to the file system; and, concurrently with executing the first thread, executing on the controller a second thread to manage the second bank according to the copy of the file system independently of the file system stored in the first bank.

[0021] Various aspects, advantages, features and embodiments of the present invention are included in the following description of exemplary examples thereof, which description should be taken in conjunction with the accompanying drawings. All patents, patent applications, articles, other publications, documents and things referenced herein are hereby incorporated herein by this reference in their entirety for all purposes. To the extent of any inconsistency or conflict in the definition or use of terms between any of the incorporated publications, documents or things and the present application, those of the present application shall prevail.

BRIEF DESCRIPTION OF THE DRAWINGS

[0022] FIG. 1 illustrates schematically the main hardware components of a memory system suitable for implementing the present invention.

[0023] FIG. 2 illustrates schematically a non-volatile memory cell.

[0024] FIG. 3 illustrates the relation between the source-drain current \( I_D \) and the control gate voltage \( V_{CG} \) for four different charges Q1-Q4 that the floating gate may be selectively storing at any one time.

[0025] FIG. 4A illustrates schematically a string of memory cells organized into an NAND string.

[0026] FIG. 4B illustrates an example of an NAND array 210 of memory cells, constituted from NAND strings 50 such as that shown in FIG. 4A.

[0027] FIG. 5 illustrates a page of memory cells, organized for example in the NAND configuration, being sensed or programmed in parallel.

[0028] FIG. 6(0) - 6(2) illustrate an example of programming a population of 4-state
memory cells.

[0029] FIGs. 7A-7E illustrate the programming and reading of the 4-state memory encoded with a given 2-bit code.

[0030] FIG. 8 illustrates the memory being managed by a memory manager with is a software component that resides in the controller.

[0031] FIG. 9 illustrates the software modules of the back-end system.

[0032] FIGs. 10A(i) - 10A(iii) illustrate schematically the mapping between a logical group and a metablock. FIG. 10B illustrates schematically the mapping between logical groups and metablocks.

[0033] FIG. 11 is a block diagram of an example of a multi-bank memory system using a satellite file system.

[0034] FIG. 12 provides some detail of elements stored in the main and satellite file system in the embodiment of FIG. 11.

DETAILED DESCRIPTION

MEMORY SYSTEM

[0035] FIG. 1 to FIG. 7 provide example memory systems in which the various aspects of the present invention may be implemented or illustrated.

[0036] FIG. 8 to FIG. 10 illustrate preferred memory and block architectures for implementing the various aspects of the present invention.

[0037] FIG. 11 and FIG. 12 illustrate the use of a satellite file system in a multi-bank system.

[0038] FIG. 1 illustrates schematically the main hardware components of a memory system suitable for implementing the present invention. The memory system 90 typically operates with a host 80 through a host interface. The memory system is typically in the form of a memory card or an embedded memory system. The memory system 90 includes a memory 200 whose operations are controlled by a
controller 100. The memory 200 comprises one or more array of non-volatile memory cells distributed over one or more integrated circuit chip. The controller 100 includes an interface 110, a processor 120, an optional coprocessor 121, ROM 122 (read-only-memory), RAM 130 (random access memory) and optionally programmable nonvolatile memory 124. The interface 110 has one component interfacing the controller to a host and another component interfacing to the memory 200. Firmware stored in nonvolatile ROM 122 and/or the optional nonvolatile memory 124 provides codes for the processor 120 to implement the functions of the controller 100. Error correction codes may be processed by the processor 120 or the optional coprocessor 121. In an alternative embodiment, the controller 100 is implemented by a state machine (not shown.) In yet another embodiment, the controller 100 is implemented within the host.

Physical Memory Structure

[0039] FIG. 2 illustrates schematically a non-volatile memory cell. The memory cell 10 can be implemented by a field-effect transistor having a charge storage unit 20, such as a floating gate or a dielectric layer. The memory cell 10 also includes a source 14, a drain 16, and a control gate 30.

[0040] There are many commercially successful non-volatile solid-state memory devices being used today. These memory devices may employ different types of memory cells, each type having one or more charge storage element.

[0041] Typical non-volatile memory cells include EEPROM and flash EEPROM. Examples of EEPROM cells and methods of manufacturing them are given in United States patent no. 5,595,924. Examples of flash EEPROM cells, their uses in memory systems and methods of manufacturing them are given in United States patents nos. 5,070,032, 5,095,344, 5,315,541, 5,343,063, 5,661,053, 5,313,421 and 6,222,762. In particular, examples of memory devices with NAND cell structures are described in United States patent nos. 5,570,315, 5,903,495, 6,046,935. Also, examples of memory devices utilizing dielectric storage element have been described by Eitan et al, "NROM: A Novel Localized Trapping, 2-Bit Nonvolatile Memory Cell," IEEE Electron Device Letters, vol. 21, no. 11, November 2000, pp. 543-545, and in United States patents nos. 5,768,192 and 6,01 1,725.
In practice, the memory state of a cell is usually read by sensing the conduction current across the source and drain electrodes of the cell when a reference voltage is applied to the control gate. Thus, for each given charge on the floating gate of a cell, a corresponding conduction current with respect to a fixed reference control gate voltage may be detected. Similarly, the range of charge programmable onto the floating gate defines a corresponding threshold voltage window or a corresponding conduction current window.

Alternatively, instead of detecting the conduction current among a partitioned current window, it is possible to set the threshold voltage for a given memory state under test at the control gate and detect if the conduction current is lower or higher than a threshold current. In one implementation the detection of the conduction current relative to a threshold current is accomplished by examining the rate the conduction current is discharging through the capacitance of the bit line.

**FIG. 3** illustrates the relation between the source-drain current $I_D$ and the control gate voltage $V_{CG}$ for four different charges Q1-Q4. The four solid $I_D$ versus $V_{CG}$ curves represent four possible charge levels that can be programmed on a floating gate of a memory cell, respectively corresponding to four possible memory states. As an example, the threshold voltage window of a population of cells may range from 0.5V to 3.5V. Seven possible memory states "0", "1", "2", "3", "4", "5", "6", respectively representing one erased and six programmed states may be demarcated by partitioning the threshold window into five regions in interval of 0.5V each. For example, if a reference current, $I_{REF}$ of 2 μA is used as shown, then the cell programmed with Q1 may be considered to be in a memory state "1" since its curve intersects with $I_{REF}$ in the region of the threshold window demarcated by $V_{CG} = 0.5V$ and 1.0V. Similarly, Q4 is in a memory state "5".

As can be seen from the description above, the more states a memory cell is made to store, the more finely divided is its threshold window. For example, a memory device may have memory cells having a threshold window that ranges from -1.5V to 5V. This provides a maximum width of 6.5V. If the memory cell is to store 16 states, each state may occupy from 200mV to 300mV in the threshold window.
This will require higher precision in programming and reading operations in order to be able to achieve the required resolution.

[0046] FIG. 4A illustrates schematically a string of memory cells organized into an NAND string. An NAND string 50 comprises of a series of memory transistors M1, M2, … Mn (e.g., n= 4, 8, 16 or higher) daisy-chained by their sources and drains. A pair of select transistors S1, S2 controls the memory transistors chain's connection to the external via the NAND string's source terminal 54 and drain terminal 56 respectively. In a memory array, when the source select transistor S1 is turned on, the source terminal is coupled to a source line (see FIG. 4B). Similarly, when the drain select transistor S2 is turned on, the drain terminal of the NAND string is coupled to a bit line of the memory array. Each memory transistor 10 in the chain acts as a memory cell. It has a charge storage element 20 to store a given amount of charge so as to represent an intended memory state. A control gate 30 of each memory transistor allows control over read and write operations. As will be seen in FIG. 4B, the control gates 30 of corresponding memory transistors of a row of NAND string are all connected to the same word line. Similarly, a control gate 32 of each of the select transistors S1, S2 provides control access to the NAND string via its source terminal 54 and drain terminal 56 respectively. Likewise, the control gates 32 of corresponding select transistors of a row of NAND string are all connected to the same select line.

[0047] When an addressed memory transistor 10 within an NAND string is read or is verified during programming, its control gate 30 is supplied with an appropriate voltage. At the same time, the rest of the non-addressed memory transistors in the NAND string 50 are fully turned on by application of sufficient voltage on their control gates. In this way, a conductive path is effective created from the source of the individual memory transistor to the source terminal 54 of the NAND string and likewise for the drain of the individual memory transistor to the drain terminal 56 of the cell. Memory devices with such NAND string structures are described in United States patent nos. 5,570,315, 5,903,495, 6,046,935.

[0048] FIG. 4B illustrates an example of an NAND array 210 of memory cells, constituted from NAND strings 50 such as that shown in FIG. 4A. Along each
column of NAND strings, a bit line such as bit line 36 is coupled to the drain terminal 56 of each NAND string. Along each bank of NAND strings, a source line such as source line 34 is coupled to the source terminals 54 of each NAND string. Also the control gates along a row of memory cells in a bank of NAND strings are connected to a word line such as word line 42. The control gates along a row of select transistors in a bank of NAND strings are connected to a select line such as select line 44. An entire row of memory cells in a bank of NAND strings can be addressed by appropriate voltages on the word lines and select lines of the bank of NAND strings. When a memory transistor within a NAND string is being read, the remaining memory transistors in the string are turned on hard via their associated word lines so that the current flowing through the string is essentially dependent upon the level of charge stored in the cell being read.

[0049] FIG. 5 illustrates a page of memory cells, organized for example in the NAND configuration, being sensed or programmed in parallel. FIG. 5 essentially shows a bank of NAND strings 50 in the memory array 210 of FIG. 4B, where the detail of each NAND string is shown explicitly as in FIG. 4A. A "page" such as the page 60, is a group of memory cells enabled to be sensed or programmed in parallel. This is accomplished by a corresponding page of sense amplifiers 212. The sensed results are latches in a corresponding set of latches 214. Each sense amplifier can be coupled to a NAND string via a bit line. The page is enabled by the control gates of the cells of the page connected in common to a word line 42 and each cell accessible by a sense amplifier accessible via a bit line 36. As an example, when respectively sensing or programming the page of cells 60, a sensing voltage or a programming voltage is respectively applied to the common word line WL3 together with appropriate voltages on the bit lines.

**Physical Organization of the Memory**

[0050] One important difference between flash memory and of type of memory is that a cell must be programmed from the erased state. That is the floating gate must first be emptied of charge. Programming then adds a desired amount of charge back to the floating gate. It does not support removing a portion of the charge from the floating to go from a more programmed state to a lesser one. This means that update data
cannot overwrite existing one and must be written to a previous unwritten location.

[0051] Furthermore erasing is to empty all the charges from the floating gate and generally takes appreciably time. For that reason, it will be cumbersome and very slow to erase cell by cell or even page by page. In practice, the array of memory cells is divided into a large number of blocks of memory cells. As is common for flash EEPROM systems, the block is the unit of erase. That is, each block contains the minimum number of memory cells that are erased together. While aggregating a large number of cells in a block to be erased in parallel will improve erase performance, a large size block also entails dealing with a larger number of update and obsolete data. Just before the block is erased, a garbage collection is required to salvage the non-obsolete data in the block.

[0052] Each block is typically divided into a number of pages. A page is a unit of programming or reading. In one embodiment, the individual pages may be divided into segments and the segments may contain the fewest number of cells that are written at one time as a basic programming operation. One or more pages of data are typically stored in one row of memory cells. A page can store one or more sectors. A sector includes user data and overhead data. Multiple blocks and pages distributed across multiple arrays can also be operated together as metablocks and metapages. If they are distributed over multiple chips, they can be operated together as megablocks and megapage.

Examples of Multi-level Cell ("MLC") Memory Partitioning

[0053] A nonvolatile memory in which the memory cells each stores multiple bits of data has already been described in connection with FIG. 3. A particular example is a memory formed from an array of field-effect transistors, each having a charge storage layer between its channel region and its control gate. The charge storage layer or unit can store a range of charges, giving rise to a range of threshold voltages for each field-effect transistor. The range of possible threshold voltages spans a threshold window. When the threshold window is partitioned into multiple sub-ranges or zones of threshold voltages, each resolvable zone is used to represent a different memory states for a memory cell. The multiple memory states can be coded by one or more binary bits. For example, a memory cell partitioned into four zones can
support four states which can be coded as 2-bit data. Similarly, a memory cell partitioned into eight zones can support eight memory states which can be coded as 3-bit data, etc.

All-bit, Full-Sequence MLC Programming

[0054] FIG. 6(0) - 6(2) illustrate an example of programming a population of 4-state memory cells. FIG. 6(0) illustrates the population of memory cells programmable into four distinct distributions of threshold voltages respectively representing memory states "0", "1", "2" and "3". FIG. 6(1) illustrates the initial distribution of "erased" threshold voltages for an erased memory. FIG. 6(2) illustrates an example of the memory after many of the memory cells have been programmed. Essentially, a cell initially has an "erased" threshold voltage and programming will move it to a higher value into one of the three zones demarcated by verify levels \( V_{v1} \), \( V_{v2} \) and \( V_{v3} \). In this way, each memory cell can be programmed to one of the three programmed state "1", "2" and "3" or remain un-programmed in the "erased" state. As the memory gets more programming, the initial distribution of the "erased" state as shown in FIG. 6(1) will become narrower and the erased state is represented by the "0" state.

[0055] A 2-bit code having a lower bit and an upper bit can be used to represent each of the four memory states. For example, the "0", "1", "2" and "3" states are respectively represented by "11", "01", "00" and '10". The 2-bit data may be read from the memory by sensing in "full-sequence" mode where the two bits are sensed together by sensing relative to the read demarcation threshold values \( r_{V1} \), \( r_{V2} \) and \( r_{V3} \) in three sub-passes respectively.

Bit-by-Bit MLC Programming and Reading

[0056] FIGs. 7A-7E illustrate the programming and reading of the 4-state memory encoded with a given 2-bit code. FIG. 7A illustrates threshold voltage distributions of the 4-state memory array when each memory cell stores two bits of data using the 2-bit code. Such a 2-bit code has been disclosed in US Patent No. 7,057,939.

[0057] FIG. 7B illustrates the lower page programming (lower bit) in a 2-pass programming scheme using the 2-bit code. The fault-tolerant LM New code
essentially avoids any upper page programming to transit through any intermediate states. Thus, the first pass lower page programming has the logical state (upper bit, lower bit) = (1, 1) transits to some intermediate state (x, 0) as represented by programming the "unprogrammed" memory state "0" to the "intermediate" state designated by (x, 0) with a programmed threshold voltage greater than \( D_A \) but less than \( D_c \).

[0058] FIG. 7C illustrates the upper page programming (upper bit) in the 2-pass programming scheme using the 2-bit code. In the second pass of programming the upper page bit to "0", if the lower page bit is at "1", the logical state (1, 1) transits to (0, 1) as represented by programming the "unprogrammed" memory state "0" to "1".

If the lower page bit is at "0", the logical state (0, 0) is obtained by programming from the "intermediate" state to "3". Similarly, if the upper page is to remain at "1", while the lower page has been programmed to "0", it will require a transition from the "intermediate" state to (1, 0) as represented by programming the "intermediate" state to "2".

[0059] FIG. 7D illustrates the read operation that is required to discern the lower bit of the 4-state memory encoded with the 2-bit code. A readB operation is first performed to determine if the LM flag can be read. If so, the upper page has been programmed and the readB operation will yield the lower page data correctly. On the other hand, if the upper page has not yet been programmed, the lower page data will be read by a readA operation.

[0060] FIG. 7E illustrates the read operation that is required to discern the upper bit of the 4-state memory encoded with the 2-bit code. As is clear from the figure, the upper page read will require a 3-pass read of readA, readB and readC, respectively relative to the demarcation threshold voltages \( D_A, D_B \) and \( D_C \).

[0061] In the bit-by-bit scheme for a 2-bit memory, a physical page of memory cells will store two logical data pages, a lower data page corresponding to the lower bit and an upper data page corresponding to the upper bit.

**Binary and MLC Memory Partitioning**
FIG. 6 and FIG. 7 illustrate examples of a 2-bit (also referred to as "D2") memory. As can be seen, a D2 memory has its threshold range or window partitioned into 4 regions, designating 4 states. Similarly, in D3, each cell stores 3 bits (low, middle and upper bits) and there are 8 regions. In D4, there are 4 bits and 16 regions, etc. As the memory's finite threshold window is partitioned into more regions, the resolution and for programming and reading will necessarily become finer. Two issues arise as the memory cell is configured to store more bits.

First, programming or reading will be slower when the threshold of a cell must be more accurately programmed or read. In fact in practice the sensing time (needed in programming and reading) tends to increase as the square of the number of partitioning levels.

Secondly, flash memory has an endurance problem as it ages with use. When a cell is repeatedly programmed and erased, charges is shuttled in and out of the floating gate 20 (see FIG. 2) by tunneling across a dielectric. Each time some charges may become trapped in the dielectric and will modify the threshold of the cell. In fact over use, the threshold window will progressively narrow. Thus, MLC memory generally is designed with tradeoffs between capacity, performance and reliability.

Conversely, it will be seen for a binary memory, the memory's threshold window is only partitioned into two regions. This will allow a maximum margin of errors. Thus, binary partitioning while diminished in storage capacity will provide maximum performance and reliability.

The multi-pass, bit-by-bit programming and reading technique described in connection with FIG. 7 provides a smooth transition between MLC and binary partitioning. In this case, if the memory is programmed with only the lower bit, it is effectively a binary partitioned memory. While this approach does not fully optimize the range of the threshold window as in the case of a single-level cell ("SLC") memory, it has the advantage of using the same demarcation or sensing level as in the operations of the lower bit of the MLC memory. As will be described later, this approach allows a MLC memory to be "expropriated" for use as a binary memory, or vice versa. How it should be understood that MLC memory tends to have more
stringent specification for usage.

**Binary Memory and Partial Page Programming**

[0067] The charge programmed into the charge storage element of one memory cell produces an electric field that perturbs the electric field of a neighboring memory cell. This will affect the characteristics of the neighboring memory cell which essentially is a field-effect transistor with a charge storage element. In particular, when sensed the memory cell will appear to have a higher threshold level (or more programmed) than when it is less perturbed.

[0068] In general, if a memory cell is program-verified under a first field environment and later is read again under a different field environment due to neighboring cells subsequently being programmed with different charges, the read accuracy may be affected due to coupling between neighboring floating gates in what is referred to as the "Yupin Effect". With ever higher integration in semiconductor memories, the perturbation of the electric field due to the stored charges between memory cells (Yupin effect) becomes increasing appreciable as the inter-cellular spacing shrinks.

[0069] The Bit-by-Bit MLC Programming technique described in connection with FIG. 7 above is designed to minimize program disturb from cells along the same word line. As can be seen from FIG. 7B, in a first of the two programming passes, the thresholds of the cells are moved at most half way up the threshold window. The effect of the first pass is overtaken by the final pass. In the final pass, the thresholds are only moved a quarter of the way. In other words, for D2, the charge difference among neighboring cells is limited to a quarter of its maximum. For D3, with three passes, the final pass will limit the charge difference to one-eighth of its maximum.

[0070] However, the bit-by-bit multi-pass programming technique will be compromised by partial-page programming. A page is a group of memory cells, typically along a row or word line, that is programmed together as a unit. It is possible to program non overlapping portions of a page individually over multiple programming passes. However, owning to not all the cells of the page are programmed in a final pass together, it could create large difference in charges programmed among the cells after the page is done. Thus partial-page programming
would result in more program disturb and would require a larger margin for sensing accuracy.

[0071] In the case the memory is configured as binary memory, the margin of operation is wider than that of MLC. In the preferred embodiment, the binary memory is configured to support partial-page programming in which non-overlapping portions of a page may be programmed individually in one of the multiple programming passes on the page. The programming and reading performance can be improved by operating with a page of large size. However, when the page size is much larger than the host’s unit of write (typically a 512-byte sector), its usage will be inefficient. Operating with finer granularity than a page allows more efficient usage of such a page.

[0072] The example given has been between binary versus MLC. It should be understood that in general the same principles apply between a first memory with a first number of levels and a second memory with a second number of levels more than the first memory.

LOGICAL AND PHYSICAL BLOCK STRUCTURES

[0073] FIG. 8 illustrates the memory being managed by a memory manager with is a software component that resides in the controller. The memory 200 is organized into blocks, each block of cells being a minimum unit of erase. Depending on implementation, the memory system may operate with even large units of erase formed by an aggregate of blocks into "metablocks" and also "megablocks". For convenience the description will refer to a unit of erase as a metablock although it will be understood that some systems operate with even larger unit of erase such as a "megablock" formed by an aggregate of metablocks.

[0074] The host 80 accesses the memory 200 when running an application under a file system or operating system. Typically, the host system addresses data in units of logical sectors where, for example, each sector may contain 512 bytes of data. Also, it is usual for the host to read or write to the memory system in unit of logical clusters, each consisting of one or more logical sectors. In some host systems, an optional host-side memory manager may exist to perform lower level memory management at
the host. In most cases during read or write operations, the host essentially issues a command to the memory system to read or write a segment containing a string of logical sectors of data with contiguous addresses.

[0075] A memory-side memory manager 300 is implemented in the controller 100 of the memory system 90 to manage the storage and retrieval of the data of host logical sectors among metablocks of the flash memory 200. The memory manager comprises a front-end system 310 and a back-end system 320. The front-end system 310 includes a host interface 312. The back-end system 320 includes a number of software modules for managing erase, read and write operations of the metablocks. The memory manager also maintains system control data and directory data associated with its operations among the flash memory 200 and the controller RAM 130.

[0076] FIG. 9 illustrates the software modules of the back-end system. The Back-End System mainly comprises two functional modules: a Media Management Layer 330 and a Dataflow and Sequencing Layer 340.

[0077] The media management layer 330 is responsible for the organization of logical data storage within a flash memory meta-block structure. More details will be provided later in the section on "Media management Layer".

[0078] The dataflow and sequencing layer 340 is responsible for the sequencing and transfer of sectors of data between a front-end system and a flash memory. This layer includes a command sequencer 342, a low-level sequencer 344 and a flash Control layer 346. More details will be provided later in the section on "Low Level System Spec".

[0079] The memory manager 300 is preferably implemented in the controller 100. It translates logical addresses received from the host into physical addresses within the memory array, where the data are actually stored, and then keeps track of these address translations.

[0080] FIGs. 10A(i) - 10A(iii) illustrate schematically the mapping between a logical group and a metablock. The metablock of the physical memory has
physical sectors for storing \( N \) logical sectors of data of a logical group. FIG. 10A(i)
shows the data from a logical group \( LG_i \), where the logical sectors are in contiguous
logical order \( 0, 1, \ldots, N-I \). FIG. 10A(ii) shows the same data being stored in the
metablock in the same logical order. The metablock when stored in this manner is
said to be "sequential." In general, the metablock may have data stored in a different
order, in which case the metablock is said to be "non-sequential" or "chaotic."

[0081] There may be an offset between the lowest address of a logical group and the
lowest address of the metablock to which it is mapped. In this case, logical sector
address wraps round as a loop from bottom back to top of the logical group within the
metablock. For example, in FIG. 10A(iii), the metablock stores in its first location
beginning with the data of logical sector \( k \). When the last logical sector \( N-I \) is
reached, it wraps around to sector 0 and finally storing data associated with logical
sector \( k-I \) in its last physical sector. In the preferred embodiment, a page tag is used
to identify any offset, such as identifying the starting logical sector address of the data
stored in the first physical sector of the metablock. Two blocks will be considered to
have their logical sectors stored in similar order when they only differ by a page tag.

[0082] FIG. 10B illustrates schematically the mapping between logical groups and
metablocks. Each logical group 380 is mapped to a unique metablock 370, except for
a small number of logical groups in which data is currently being updated. After a
logical group has been updated, it may be mapped to a different metablock. The
mapping information is maintained in a set of logical to physical directories, which
will be described in more detail later.

SATELLITE FILE SYSTEM

[0083] This section presents the use of a satellite file system for memory systems
using dual memory banks. To improve performance, the controller executes a back
end thread for each of the banks and communicates with each back through a separate
interface. One of the banks has the main file system for managing the memory, but
copies of some important files are stored in the other bank in a satellite file system.
The satellite file system improves performance by providing both threads access to
copies of the files needed during normal system operations without the threads
needing to rely upon just the main file system. Each of these banks may consist of
one or more chips or be semi-autonomous arrays that can be independently operated on a single chip; and although discussed here for the case of two memory banks, and a corresponding number of bank end threads for the controller’s back end. The discussion below will also be given in the context of a memory card using a NAND-type architecture for memory arrays, but readily extends to architectures and non-card uses, such as embedded systems, SSD, and so on.

[0084] Although the following discussion may be based on various exemplary embodiments to provide concrete examples, the techniques and structures here can be applied fairly generally to memory systems having a controller and multiple banks that can independently operated, where the banks include some amount of non-volatile memory, whether flash or other variety, that can be used to store system data that the controller can use to manage the memory system. In addition to the other referenced cited above, theses can include the various memory systems presented in the following US Patent, patent publication and application numbers: 7,480,766; US-2005-0154819-A1; US-2007-0061581-A1; US-2007-0113030-A1; US-2008-0155178-A1; US-2008-0155228-A1; US-2008-0155176-A1; US-2008-0155177-A1; US-2008-0155227-A1; US-2008-0155175-A1; 12/348,819; 12/348,825; 12/348,891; 12/348,895; 12/348,899; and 61/142,620.

[0085] FIG. 11 is a block diagram illustrating some of the relevant elements of an exemplary two-bank embodiment employing a satellite file system. A host 401 is connected to the memory system 403, which has controller circuitry 411 and memory banks BANK 0 421-0 and BANK 1 421-1 in this example. The controller communicates with the host through host interface 413 and to each of the banks through a corresponding interface BANK INT 421. In the firmware control, the controller 411 executes a front end portion 415 and a back end thread 417 for each bank. In the exemplary embodiment, there is a single host interface 413 and so is the circuitry that transfers the data is also single. The logic in the Front End Thread 413 runs in the controller circuit along with the Backend Firmware Threads (417-0, 417-1) and assigns work for the Backend Firmware Threads. The front end may individually assign tasks to a single bank, or splits tasks between the banks, such as, for example, on a large write command part of the data could be written in one bank and the remaining part in the other bank to increase write performance through parallelism.
Although not explicitly shown, in FIG. 11 the controller will typically have one large contiguous buffer used for data transfers, which is then logically segregated for the different banks. When data is transferred, the data intended for a specific bank is then transferred into its logically segregated buffer. Alternate embodiments could employ physically distinct buffers.

[0086] A number of memory systems have used multiple banks for the memory circuitry to increase parallelism and, consequently, performance. This can be further improved by including the multiple bank interfaces and, further still, by the inclusion of back end thread for each bank, as both shown in FIG. 11. As shown FIG. 11, this is extended by providing bank with a copy of the file system, with the main file system being kept in one bank (taken here as BANK 0 421-0) and satellite copies being kept in other banks, thereby allowing each back end thread (417-0, 417-1) to independently access its own copy (in 421-0, 421-1, respectively) during regular memory operation. (It should be noted that although FIG. 11 refers to firmware control, it will be understood the various functions can implemented in hardware, software, firmware or various combinations of these, as is familiar in the art.)

[0087] Typically, previous arrangements would maintain only a single copy (or single active copy) of the file system. Considering this for the case of a dual bank architecture and that contains two hardware banks, with corresponding software control, the software control could largely execute in parallel. If only a single copy stored all the file system information would be stored on one bank, say Bank 0, and only Bank 0 can (directly) access this information. The file system would store items such as configuration information, firmware microcode and overlay code. Various modules in the system access the file system to retrieve the information they need. Barring overlay code and link table information, which stores the physical link information as described above, all the information that is stored in the file system is read during the card boot up/power on/reset time. During this time, only Bank 0 Thread 417-0 is active and hence there is no issue of contention, performance degradation or deadlock between the two block threads; however, during the subsequent normal course of the card operation, the overlay and link table information needs to be accessed by all the banks based on their individual needs.
[0088] Subsequent to the boot, once all the banks are running, any bank that does not have the file system information will need to interrupt Bank 0 and request Bank 0 to fetch the required information. This arrangement poses several risks to the system. A first of these is that the information sought from the file system is a continuous process and not a one time request. Interrupting Bank 0 on a regular basis would compromise the system performance. Others are the result of a system where some or all host requests are executed by both (or, more generally, all) banks simultaneously. Though tasks can be split between the banks, banks are inter-dependent and wait for tasks to be completed. Bank 0 could be waiting for Bank 1 to finish its task before becoming free; but Bank 1 might need to access file system information and hence interrupt and request Bank 0. This is a deadlock situation as Bank 0 will be unable to process the request since it is waiting for Bank 1 to complete the host operation.

[0089] Consequently, for memory systems whose firmware uses a file system to store non-volatile control information and when multiple channels (or banks) are used to access the firmware, a single copy of this file system becomes an access bottleneck. The introduction of a satellite file system provides a way around this problem by creating a read-only copy of critical portions of the File System that is needed by the secondary banks. Although this increase the amount of non-volatile memory that needs to be set aside for control data, with a corresponding decrease in the amount available for user data, given the current and continuing increase of capacity in such systems, this will be a more than acceptable price for the resultant performance improvements.

[0090] Thus, a primary aspect presented here, is the creation and storage of read only copies of certain files on the second or other additional banks in a dual- or multi-bank system. The satellite file system is a subset of the main file System. In the exemplary embodiment, the satellite file system contains all of the files need in the course of standard, post-boot operations and only the main system has copies of the files used for the just the during the card boot up/power on/reset time, as initially only the one bank is operating at these times. As the threads request files, the file system module can route these calls based on the bank.

[0091] Returning to FIG. 11, maintaining the satellite file system solves these issues
by introducing copies for all of the banks of files needed by the banks during the normal card operation. Consequently, BANK 1 421-1 include the satellite file system in addition to the boot block and user, or host, data. In addition to the boot block and user data, BANK 0 421-0 has the main file system. Copies of some important files that can be stored in a special area in other banks and location information of those copies is stored in the main file system. To improve system reliability, each bank will typically also maintain a back up copy of the main or satellite (as appropriate) file system. Each back end thread 417-0 and 417-1 then performs its functions (read/operations, control operations, various background and low level operations, and so on, as discussed in preceding sections and the various cite references) for the corresponding bank using its respective copy of the file system.

[0092] In a preferred embodiment, during the low level card format, a special satellite file system area is created in bank or banks that do not contain the main copy of the file system. As soon as the main file system is written to, the files that are needed are identified and copied into the satellite file systems. While copying, the location information of the satellite files is stored in the main file system directory. This file system directory is read during the card boot up/power on/reset time and can be accessed by any bank.

[0093] When Bank 1 thread 417-1 requests the file system information, which is contained in one of the special files, the file system modules can recognize the request and fetch the information from the file copy that is stored in the satellite file system in BANK 1 421-1. It does not interrupt BANK 0 421-0, which is not even aware of such a request.

[0094] Like the Main File System, the Satellite File System preferably contains two copies of all the files that it contains, to provide fault tolerance. Since information stored in the satellite file system is fixed and does not change over the lifetime of the card, the satellite file system does not allow any updates to the files that it contains.

[0095] FIG. 12 depicts the elements of the main file system in Bank 0 and the satellite file system in Bank 1 according to the exemplary embodiment. Both banks will have a boot block as well as the user data area and various other control date such as discussed in the preceding sections or cited references. Both the main and satellite
file systems have copies of the files needed during regular operations, such as Link Table files and various overlay files. As Bank 0 is the bank initially active during the boot process, the main file system contains the firmware file, overlay file, flash RISC code file by which the controller interacts with the memory banks, control RISC code file (this is the for the controller that interfaces with the host to transfer data), link table file (specifying how the metablocks in the memory are logically linked) and various configuration files.

[0096] Under this arrangement, the controller is then able to independently and simultaneously execute a plurality of backend firmware threads (software threads), each of which is responsible for managing (execute read/write operations, etc.) that happen on the memory bank to which it is assigned. In the exemplary embodiment, each backend firmware thread issues instructions to its memory bank through a dedicated Flash RISC controller (another controller circuit, one each for each Bank).

[0097] With respect to overlays, during boot up, firmware code is loaded into the RAM area of the controller circuit. As this RAM is generally relatively limited in size, the system often ends up having code that cannot be accommodated completely into this RAM area. To solve this issue, the code is often broken up into two parts: the main code, which needs to be present in the RAM always; and overlay code, which can be loaded in and out of the RAM on demand. The main code is stored in a file called as Firmware File and the overlay code is stored in the Overlay File. The main code, due to its availability in the RAM always, is only loaded once during the boot up; however, sections of overlay code may be needed by any of the firmware threads during the normal execution of the card and the requirements to load such code are thread specific, hence the preference to store the overlay file in both banks.

[0098] The foregoing detailed description of the invention has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. The described embodiments were chosen in order to best explain the principles of the invention and its practical application, to thereby enable others skilled in the art to best utilize the invention in various embodiments and with various modifications as are suited to the particular use.
contemplated. It is intended that the scope of the invention be defined by the claims appended hereto.
IT IS CLAIMED:

1. A non-volatile memory system, comprising:
   a non-volatile memory circuit having a plurality of independently operable banks, each of the banks including one or more non-volatile memory arrays to store user data and system data; and
   a controller circuit to manage the storage of user on the memory circuit, including:
   a plurality of bank interfaces, each connected a respective one of the banks to transfer data between the controller and the corresponding bank;
   processing circuitry to execute a plurality of threads each independently and concurrently managing a corresponding one of the banks, wherein a first of the banks stores system data including a file system by which the corresponding thread manages the first bank and each of the other banks stores system data including a copy of a portion of the file system by which the corresponding thread manages the bank.

2. The non-volatile memory system of claim 1, wherein the processing circuitry includes a dedicated controller for each thread to manage the corresponding bank by issuing instructions thereto through the respective bank interface.

3. The non-volatile memory system of claim 1, wherein the copies of a portion of the file system are read only copies.

4. The non-volatile memory system of claim 1, wherein the non-volatile memory arrays are formed of physical erase blocks and wherein the controller manages the memory using structures of multiple blocks logically linked into composite structures, where the file system and the portion of the file system both include the linking information by which the composite structures are formed.

5. The non-volatile memory system of claim 1, where the file system and the portion of the file system both include one or more overlay files for controlling the corresponding banks.
6. The non-volatile memory system of claim 1, where the file system includes information used during a boot up process that is not include in the portion of the file system.

7. The non-volatile memory system of claim 6, where the information used during a boot up process that is not included in the portion of the file system includes firmware files.

8. The non-volatile memory system of claim 7, where the information used during a boot up process that is not included in the portion of the file system further includes code files for managing of the banks by the threads and for interfacing to a host to which the memory system is attached.

9. The non-volatile memory system of claim 1, where the controller further includes a host interface to transfer data between the memory system and a host to which the non-volatile memory system is connected and wherein the processing circuitry further executes a front end thread managing interactions between the controller and the host.

10. The non-volatile memory system of claim 9, wherein the front end thread assigns tasks for the plurality of threads managing the banks.

11. The non-volatile memory system of claim 1, wherein the first of the banks stores multiple copies of the file system and each of the other banks stores multiple copies of the portion of the file system.

12 A method of operation a non-volatile memory system, the memory system including a non-volatile memory circuit having a plurality of independently operable banks and a controller circuit to manage the storage of user on the memory circuit, the method comprising:

storing in non-volatile memory on a first of the banks system data including a file system;
storing in non-volatile memory on a second of the banks system data including a copy of a portion of the file system;
executing on the controller a first thread to manage the first bank according to the file system; and
concurrently with executing the first thread, executing on the controller a second thread to manage the second bank according to the copy of the file system independently of the file system stored in the first bank.

13. The method of claim 12, wherein each of the first and second threads is executed on a respective dedicated controller and respectively communicates with the first and second banks through a corresponding bank interface.

14. The method of claim 12, wherein the copy of a portion of the file system is a read only copy.

15. The method of claim 12, wherein the non-volatile memory arrays are formed of physical erase blocks and wherein the controller manages the memory using structures of multiple blocks logically linked into composite structures, where the file system and the portion of the file system both include the linking information by which the composite structures are formed.

16. The method of claim 12, where the file system and the portion of the file system both include one or more overlay files for controlling the corresponding banks.

17. The method of claim 12, further comprising:
prior to executing the second thread on the controller, performing a boot up process executed by the first thread using information in the files system stored on the first bank that is not included in the portion of the file system.

18. The method of claim 17, where the information used during the boot up process that is not included in the portion of the file system includes firmware files.
19. The method of claim 17, where the information used during a boot up process that is not included in the portion of the file system further includes code files for managing of the banks by the threads and for interfacing to a host to which the memory system is attached.

20. The method of claim 12, where the controller further includes a host interface to transfer data between the memory system and a host to which the non-volatile memory system is connected, the method further comprising:

executing on the controller a front end thread managing interactions between the controller and the host.

21. The method of claim 20, wherein the front end thread assigns tasks for the plurality of threads managing the banks.
FIG. 1
FIG. 4A
Programming into Four States Represented by a 2-bit Code

**FIG. 6**
Multistate Memory

**FIG. 7A**

Lower Page Programming (2-bit Code)

**FIG. 7B**

Upper Page Programming (2-bit Code)

**FIG. 7C**
Lower Page Read (2-bit Code)

**FIG. 7D**

Upper Page Read (2-bit Code)

**FIG. 7E**
FIG. 9
Fig. 10A

Logical Group
(i) $L_{G_j}$

Physical Group
(Metablock)
(ii) $M_{B_j}$

(iii) $M_{B_j}$

Page Tag

Fig. 10B

Logical Group

Physical Group
(Metablock)

Logical to Physical Directories

380

$LG_0$

$LG_1$

$LG_2$

...

$LG_x$

...

370

$MB_0$

$MB_1$

$MB_2$

...

$MB_i$

...

FIG. 12
**INTERNATIONAL SEARCH REPORT**

**International application No**
PCT/US2010/038014

**A. CLASSIFICATION OF SUBJECT MATTER**

INV. G06F3/06 G06F17/30 G11C29/00

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

**B. FIELDS SEARCHED**

Minimum documentation searched (classification system followed by classification symbols)

G06F GIIC

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

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, WPI Data

**C. DOCUMENTS CONSIDERED TO BE RELEVANT**

<table>
<thead>
<tr>
<th>Category</th>
<th>Citation of document with indication, where appropriate, of the relevant passages</th>
<th>Relevant to claim No</th>
</tr>
</thead>
</table>

**D**

Further documents are listed in the continuation of Box C

See patent family annex

**Date of the actual completion of the international search**

20 August 2010

**Date of mailing of the international search report**

27/08/2010

**Name and mailing address of the ISA/ European Patent Office, P B 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel (+31-70) 340-2040, Fax (+31-70) 340-3016**

Authorized officer

De Ceul aer, Bart
<table>
<thead>
<tr>
<th>Patent document cited in search report</th>
<th>Publication date</th>
<th>Patent family member(s)</th>
<th>Publication date</th>
</tr>
</thead>
<tbody>
<tr>
<td>WO 2008070173 AI</td>
<td>12-06-2008</td>
<td>CA 2672035 AI</td>
<td>12-06-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CA 2672100 AI</td>
<td>12-06-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN 101689130 A</td>
<td>31-03-2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN 101715575 A</td>
<td>26-05-2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN 101681282 A</td>
<td>24-03-2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN 101689131 A</td>
<td>31-03-2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CN 101690068 A</td>
<td>31-03-2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 2126709 A2</td>
<td>02-12-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 2108143 A2</td>
<td>14-10-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 2100214 AI</td>
<td>16-09-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 2115563 A2</td>
<td>11-11-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 2126679 A2</td>
<td>02-12-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 2109822 AI</td>
<td>21-10-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 2109812 A2</td>
<td>21-10-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 2126680 A2</td>
<td>02-12-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EP 2126698 A2</td>
<td>02-12-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JP 2010512568 T</td>
<td>22-04-2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JP 2010515116 T</td>
<td>06-05-2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JP 2010512584 T</td>
<td>22-04-2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JP 2010512586 T</td>
<td>22-04-2010</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KR 20090087119 A</td>
<td>14-08-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KR 20090097906 A</td>
<td>16-09-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KR 20090087498 A</td>
<td>17-08-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KR 20090102788 A</td>
<td>30-09-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KR 20090095641 A</td>
<td>09-09-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KR 20090102789 A</td>
<td>30-09-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2008313312 AI</td>
<td>18-12-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2008141043 AI</td>
<td>12-06-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2008229079 AI</td>
<td>18-09-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2008140724 AI</td>
<td>12-06-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2008183953 AI</td>
<td>31-07-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2008140932 AI</td>
<td>12-06-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2008140909 AI</td>
<td>12-06-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2008140910 AI</td>
<td>12-06-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2008168304 AI</td>
<td>10-07-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2008256183 AI</td>
<td>16-10-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2008256292 AI</td>
<td>16-10-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2008183882 AI</td>
<td>31-07-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2009132760 AI</td>
<td>21-05-2009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>US 2008313364 AI</td>
<td>18-12-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2008070172 A2</td>
<td>12-06-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2008070174 A2</td>
<td>12-06-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2008070191 A2</td>
<td>12-06-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2008070796 A2</td>
<td>12-06-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2008070798 AI</td>
<td>12-06-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2008070799 A2</td>
<td>12-06-2008</td>
</tr>
<tr>
<td>WO 2008070173 AI</td>
<td>12-06-2008</td>
<td>WO 2008070800 AI</td>
<td>12-06-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2008070802 A2</td>
<td>12-06-2008</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WO 2008070803 AI</td>
<td>12-06-2008</td>
</tr>
<tr>
<td>US 6006307 A</td>
<td>21-12-1999</td>
<td>NONE</td>
<td></td>
</tr>
<tr>
<td>US 4818932 A</td>
<td>04-04-1989</td>
<td>NONE</td>
<td></td>
</tr>
</tbody>
</table>

Form PCT/ISA/210 (patent family annex) (April 2005)