FTL ALGORITHMS FOR NAND-type FLASH MEMORY

Introduction:

This paper explains about NAND-type flash memory and also defines the role of an intermediate software layer called FTL and various FTL algorithms as a solution to the hardware issues.FTL Algorithms are classified into basic and advanced mapping depending on the complexity of the mapping method. The basic mapping algorithms are those that simply map logical addresses to physical addresses in the unit of a sector or block. The advanced mapping algorithms can be further classified to performance and durability enhancing algorithm. Performance enhancing algorithms are mostly concerned with erase-before write issue. Performance enhancing algorithms are classified into three major schemes: log sector based, log –block based and state based depending upon the methods of utilizing spare blocks of flash memory. The durability enhancement is related to the degradation of the physical block. The durability enhancement algorithms are further classified into wear-leveling and cleaning policy depending on the level of monitoring. This paper aims at properly surveying the important features of flash memories and FTL algorithms with a new level of classifications. Till now, clear demarcation of the work on the flash memories is not available. This work will be greatly beneficial to future studies in this area of overall developments in the field of flash memories.

NAND-type flash memory:

NAND-type flash memory is a new type of flash memory which is characterised by high density data storage, small chip size, low cost per bit memory. It requires an intermediate software layer called FTL for managing and controlling data by which the overall performance of flash memory can be enhanced.

Sector based mapping algorithm:

In sector based mapping, every logical sector is mapped to its corresponding physical sector. If there are n logical sectors seen by the file system,then the row size of the logical to physical mapping table is n.

Drawback:
Requires large amount of RAM.
Suitable for small embedded systems.

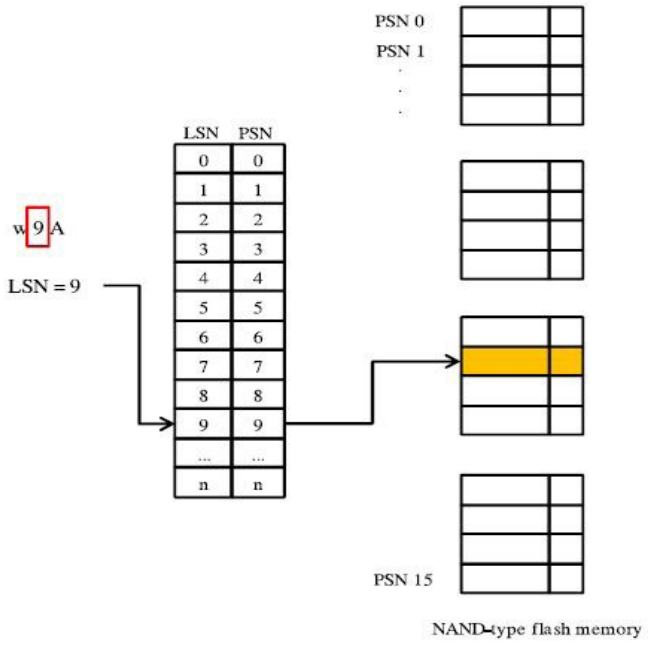

Fig 1: Sector mapping algorithm

For example, if the file system issues a command "w 9 A: write data A to logical sector number 9",the FTL writes data to physical sector number 9 according to mapping table. When another write operation occurs in PSN 9,an erase operation is inevitable due to erase beforew write.Since its already occupied by data, FTL determines the location of an empty sector,writes data and adjust the mapping table.If an empty sector does not exist, FTL will select a victim block from flash memory,copy the data to a spare free block and update the mapping table.

Algorithm:

Input : Logical Sector Number(LSN), data
Issue command to write data to LSN ie "W LSN Data"
FTL finds the corresponding PSN and writes data to it.
File system issues another write command
If Logical Sector Number is empty
write data to it
else
find an empty sector and write data
else
search a victim block and copy data to the spare free block

Block Mapping Algorithm:

In the block mapping algorithm, the row size of the logical to physical mapping table corresponds to the number of blocks in the flash memory. When the file system issues the command "W 9 A write data A to logical sector number 9", the LBN logical block number is first calculated by dividing the LSN via the number of sectors per block.

Drawback:
If the file system issues many write command with identical LSNs, this will lead to severe performance degradation due to write and erase operations.

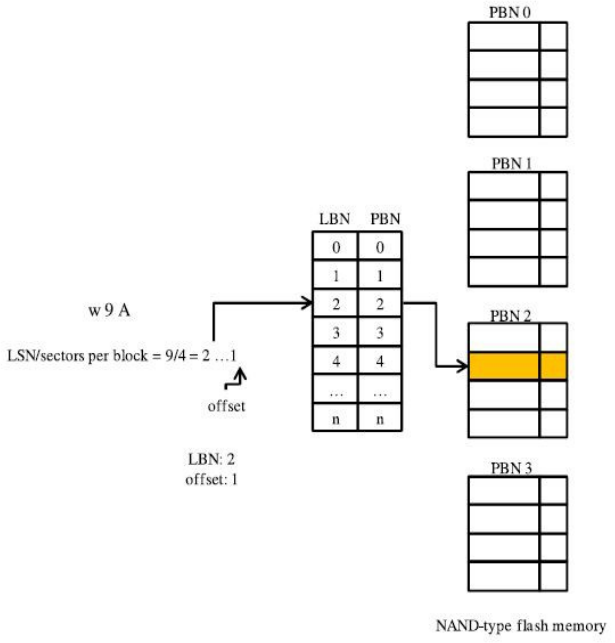

Fig 2: Block mapping algorithm

Algorithm:

Input : Logical Sector Number(LSN), data
Issue command to write data to Logical Sector Number
Calculate Logical Block Number ie LBN = LSN/no of sectors per block
Calculate the offset(modulus of LBN)
Write data to corresponding block obtained with the offset calculated.
Issue another write command with the same LSN
if Logical Block Number is empty
write data
else
erase and write.
Repeat the above steps

Hybrid Mapping:

It is introduced due to the disadvantages of both sector and block mapping algorithms. The hybrid mapping uses both block mapping and sector mapping algorithm. First it uses block mapping to achieve the corresponding physical block and it then uses a sector mapping to locate an available empty sector within the physical block.
When the file system issues the command, “w 9 A”, FTL calculates the LBN 2(9/4) and retrieves PBN 2 from the block mapping table. Next, an empty sector is searched from the corresponding PBN’s sector mapping table. In the example, since the first sector of PBN 2 is empty, the data is written to the first sector. Since the data is written in the location where the logical and physical sector offsets (e.g., 1 and 0, respectively) are different, the corresponding logical sector number of data, LSN 9, is indicated to the first sector spare area of PBN.

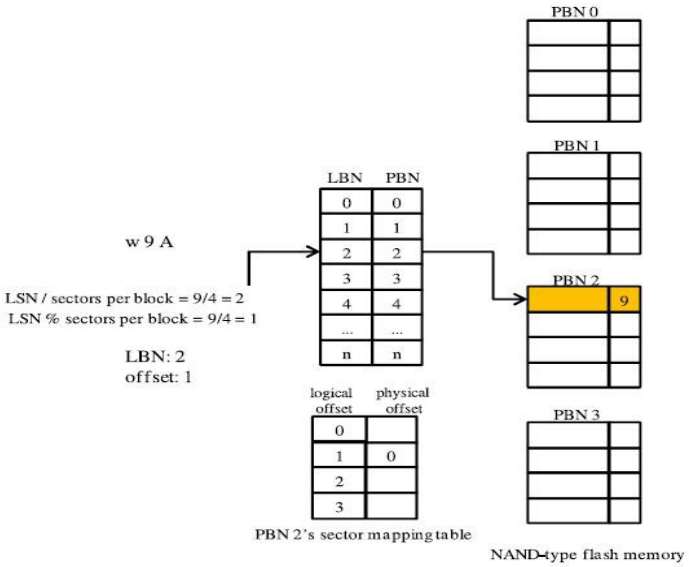

Fig 3: Hybrid Mapping Algorithm

Algorithm:

Input : Logical Sector Number(LSN), data
Issue the command to write data to Logical Sector Number(LSN)
Calculate LBN and retrieves the corresponding PBN from block mapping table.
Search for an empty sector and writes data to it.
Update the first sector of spare area.

Performance Evaluation:

$$Cread = \chi Tr$$

$$Cwrite = piTw + po(\chi Tr + Tw) + pe(Te + Tw + Tc)$$

Cread and Cwrite denote the costs of read and write commands issued from the file system layer
The variable χ represents the number of read operations.
Tr , Tw and Te are the costs of read, write, and erase operations processed in the flash memory layer.
Tc is the cost of copies needed for transferring the valid data to another free block before erasing.
pi and po are the probability for in-place and out-of-place technique,
pe is the probability that a write command incurs any erase operations.

Performance Enhancing algorithm:

Mitsubishi Algorithm:

Mitsubishi algorithm allocates log sectors within the same block from the original data. It is based on the block mapping algorithm, in order to have benefits in small SRAM requirement and small overhead in mapping table construction. Mitsubishi proposed “space sectors” as another term for log sectors. The space sectors are empty sectors that are allocated within a physical block to be used as a buffer for the rest of the sectors. Therefore, one physical block in Mitsubishi is composed of a general sector area and a space sector area.

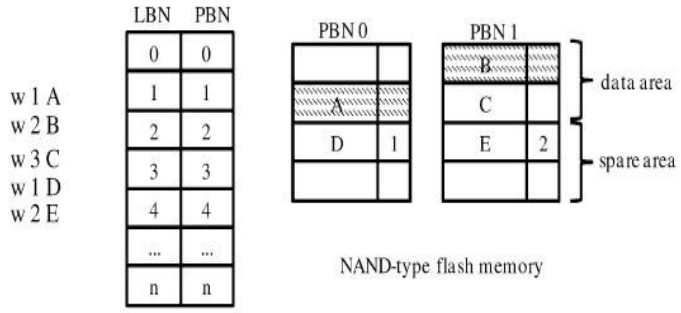

Fig 4: Mitsubishi Algorithm

Figure shows that a physical block is composed of two general sectors and two space ssectors. . When the file system issues the first write request, “w 1 A”, LBN 0 (1/2) and the logical sector offset 1 (1%2) are calculated and the corresponding PBN is obtained from the block mapping table. As thephysical sector offset 1 of PBN 0 is initially empty, the data is written to the second sector of PBN 0. Additional write operations are performed in the same way until “w 1 D”. When the corresponding physical sector is already occupied with data, Mitsubishi searches an empty sector starting from the first sector of the space sector area and writes data in that sector.Thus, data “D” is written in the first sector of the space sectors, and “1” is indicated in the spare area for re-calling its logical sector number. When a read operation, “r LSN: read data from LSN”, occurs, the most up-to-date can be found by scanning the spare areas from the

M-Systems:

M-Systems propose FTL algorithms known as ANAND and FMAX. These algorithms are examples of log-sector scheme which allocate log-sectors within a different block from the original data. Their techniques are based on the block mapping algorithm ie one logical block is mapped to two physical blocks giving the ratio of 1:2. The write operation is similarly performed to the block mapping technique, but when an overwrite is inevitable due to the occurrence of identical logical sector numbers, the data is written to “replacement block”.

When a write operation, “w 1 A”, is issued, LBN and logical sector offset are calculated and PBN is retrieved from the block mapping table. However, when the targeted sector is already filled with data, the write operation is performed differently in ANAND and FMAX. In ANAND, the overwrite is written in the replacement block where the logical and physical sector offsets are identical (in-place).

In Fig. “w 1 D” is written in the replacement block, PBN 100, since the physical sector offset 1 of PBN 0 is already filled with data “A”. When the third identical logical sector number (“w 1 F”) occurs, valid data of both data block and replacement block are merged into a free block. The free block then becomes a new data block, and old data block and the replacement block are erased. Gathering valid data from more than two blocks into one block is called “merge operation”.

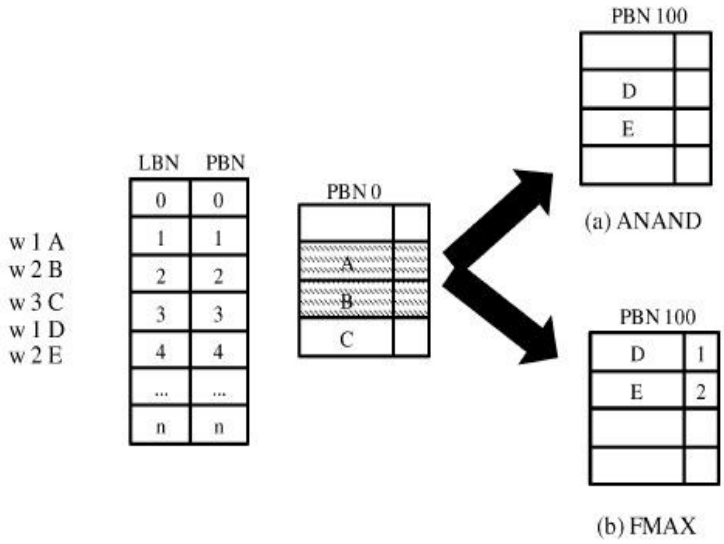

Fig 4: ANAND and FMAX Algorithm

Basic Algorithm calling sequence:

```
if(!(space_available))
{
   Erase_block (block_to_erase);
}
Program (beginning_address, data, number_bytes_to_program);
Read();
}

Program (beginning_address, data, number_bytes_to_program)
{
Flash_address = logical_address;
```

```
while (number_of_bytes_to_program)
   {
   Send_command (Program_command, Flash_address, data_byte);
   while (Flash_status = = not_done)
    {
     wait;
     }
   number_of_bytes_to_program--;

   }
}
```