Design a NAND FTL

2023.08.25
쥬용지에
arashio1111@dankook.ac.kr

1. Design

Struct & function

#define TOTAL_BLOCKS 256
#define PAGE_PER_BLOCK 16
#define PAGE_SIZE 4096

Page
Char data[PAGE_SIZE]

Block
Page pages[PAGE_PER_BLOCK]
Int valid[PAGE_PER_BLOCK]

Flash : Block flash[TOTAL_BLOCKS]

Int allowcateBlock()

Void eraseBlock(int blockNumber)

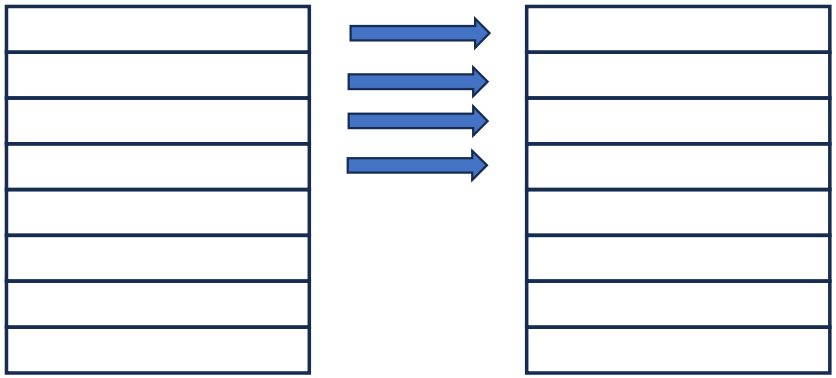
GC

Int InvalidPageInBlock(int blockNumber)

Void readPage(int LBN, int PN, char *buffer)

Void writePage(int LBN, int PN, char *buffer)

Allocate & mapping



int L2P[TOTAL_BLOCKS]

int P2L[TOTAL_BLOCKS]

```
#define TOTAL_BLOCKS 256
#define PAGE_PER_BLOCK 16
#define PAGE_SIZE 4096
```

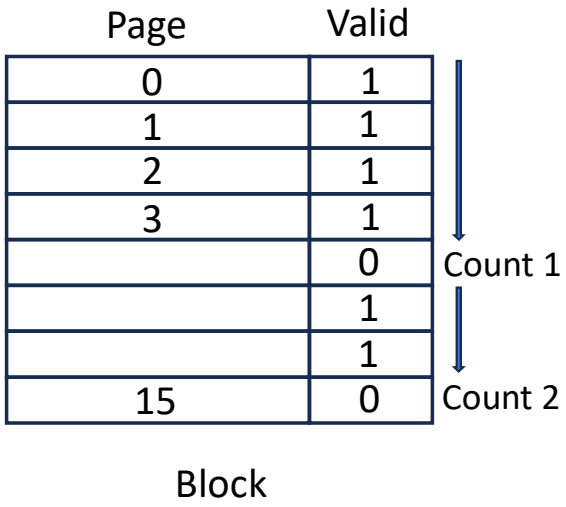
```
Block flash[TOTAL_BLOCKS];
int L2P[TOTAL_BLOCKS];
int P2L[TOTAL_BLOCKS];

void initFTL() {
    for (int i = 0; i < TOTAL_BLOCKS; i++) {
        L2P[i] = -1;
        P2L[i] = -1;
        for (int j = 0; j < PAGES_PER_BLOCK; j++) {
            flash[i].valid[j] = 0;
        }
    }
}

int allocateBlock() {
    for (int i = 0; i < TOTAL_BLOCKS; i++) {
        if (L2P[i] == -1 && P2L[i] == -1) {
            return i;
        }
    }
    return -1;
}
```

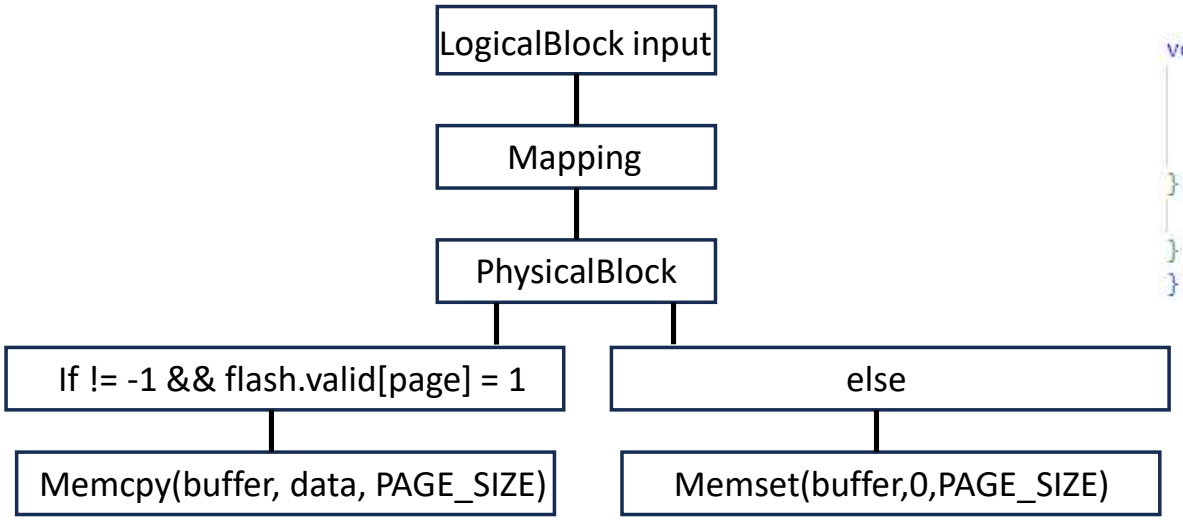
InvalidPageInBlock

```
#define TOTAL_BLOCKS 256
#define PAGE_PER_BLOCK 16
#define PAGE_SIZE 4096
```



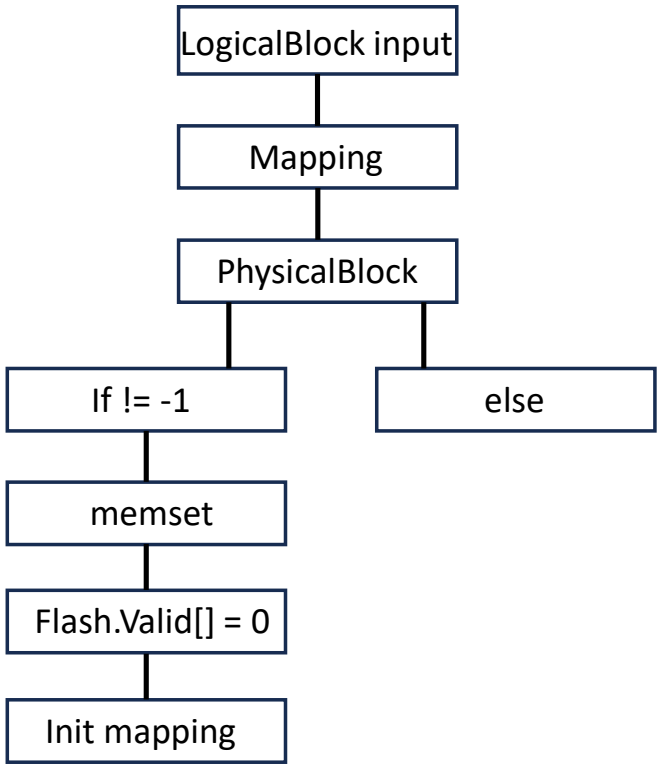

```c
int invalidPagesInBlock(int block) {
    int count = 0;
    for (int j = 0; j < PAGES_PER_BLOCK; j++) {
        if (flash[block].valid[j] == 0) {
            count++;
        }
    }
    return count;
}
```

ReadPage



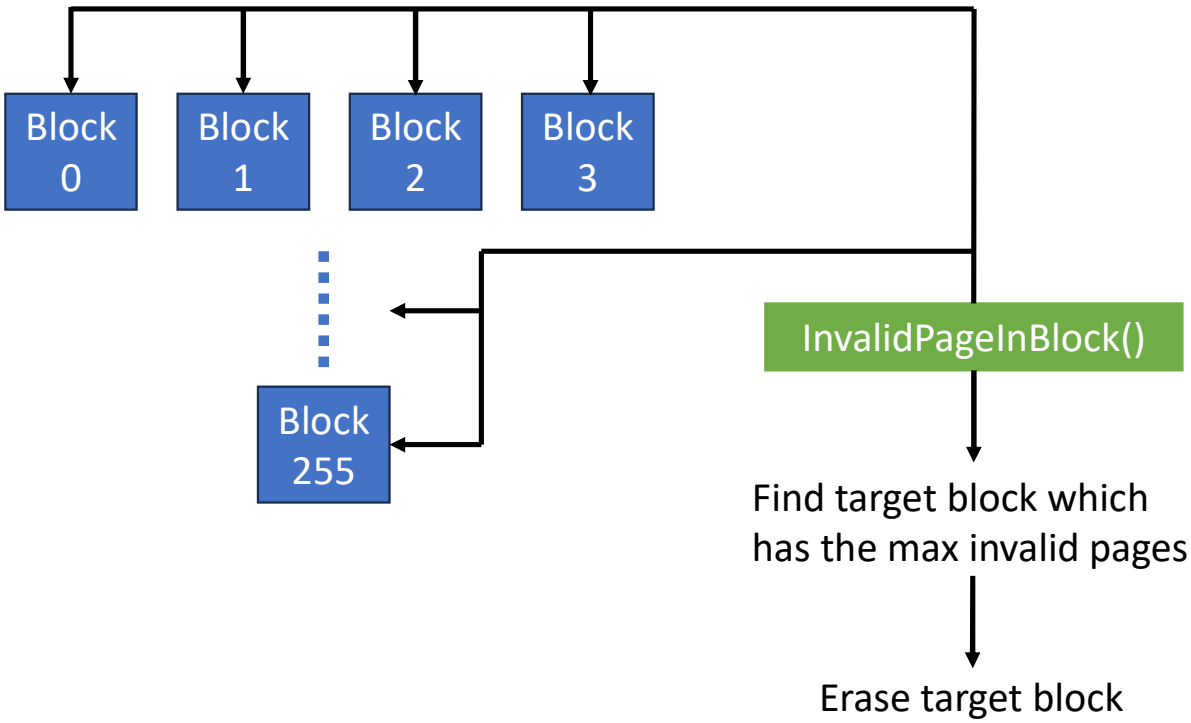

```
void readPage(int logicalBlock, int page, char *buffer) {
    int physicalBlock = L2P[logicalBlock];
    if (physicalBlock != -1 && flash[physicalBlock].valid[page]) {
    memcpy(buffer, flash[physicalBlock].pages[page].data, PAGE_SIZE);
} else {
    memset(buffer, 0, PAGE_SIZE);
}
}
```

EraseBlock




```
void eraseBlock(int logicalBlock) {
    int physicalBlock = L2P[logicalBlock];
    if (physicalBlock != -1) {
        for (int j = 0; j < PAGES_PER_BLOCK; j++) {
            memset(flash[physicalBlock].pages[j].data, 0, PAGE_SIZE);
            flash[physicalBlock].valid[j] = 0;
        }
        L2P[logicalBlock] = -1;
        P2L[physicalBlock] = -1;
    }
}
```

GC



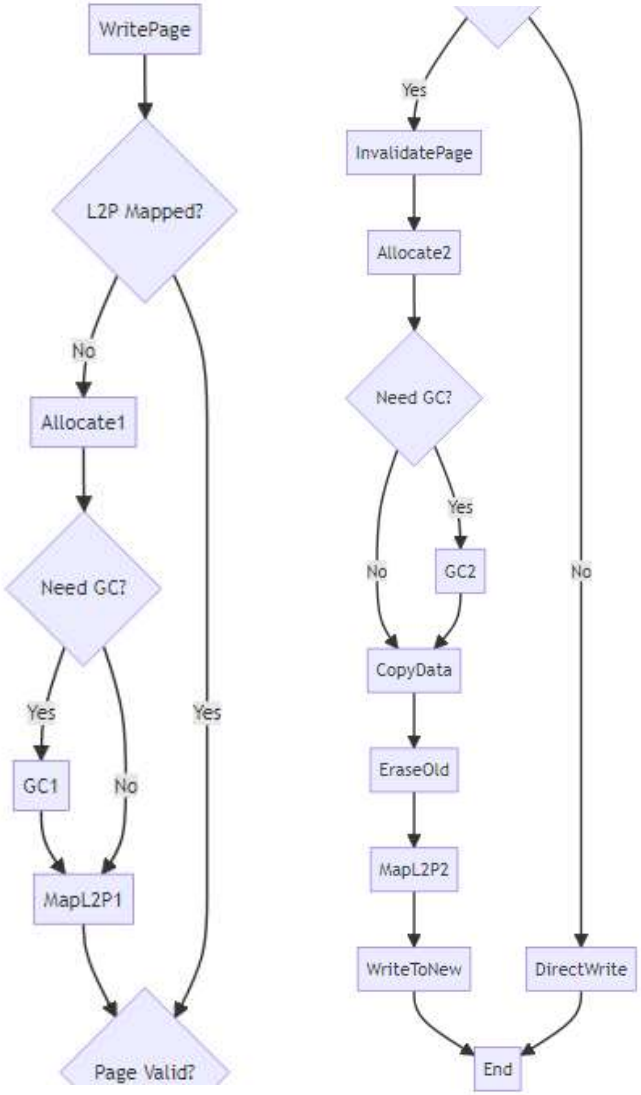

```
void garbageCollector() {
    int targetBlock = -1;
    int maxInvalidPages = -1;

    for (int i = 0; i < TOTAL_BLOCKS; i++) {
        int currentInvalidPages = invalidPagesInBlock(i);
        if (currentInvalidPages > maxInvalidPages) {
            targetBlock = i;
            maxInvalidPages = currentInvalidPages;
        }
    }

    if (targetBlock != -1) {
        eraseBlock(P2L[targetBlock]);
    }
}
```

Copy valid pages operation in write page.

WritePage




```c
void writePage(int logicalBlock, int page, char *buffer) {
    int physicalBlock = L2P[logicalBlock];
    if (physicalBlock == -1) {
        physicalBlock = allocateBlock();
        if (physicalBlock == -1) {
            garbageCollector();
            physicalBlock = allocateBlock();
        }
        L2P[logicalBlock] = physicalBlock;
        P2L[physicalBlock] = logicalBlock;
    }

    if (flash[physicalBlock].valid[page] == 0) {
        memcpy(flash[physicalBlock].pages[page].data, buffer, PAGE_SIZE);
        flash[physicalBlock].valid[page] = 1;
    } else {
        flash[physicalBlock].valid[page] = 0;
        int newPhysicalBlock = allocateBlock();
        if (newPhysicalBlock == -1) {
            garbageCollector();
            newPhysicalBlock = allocateBlock();
        }
        for (int j = 0; j < PAGES_PER_BLOCK; j++) {
            if (flash[physicalBlock].valid[j]) {
                memcpy(flash[newPhysicalBlock].pages[j].data, flash[physicalBlock].pages[j].data, PAGE_SIZE);
                flash[newPhysicalBlock].valid[j] = 1;
            }
        }
        eraseBlock(logicalBlock);
        L2P[logicalBlock] = newPhysicalBlock;
        P2L[newPhysicalBlock] = logicalBlock;

        memcpy(flash[newPhysicalBlock].pages[page].data, buffer, PAGE_SIZE);
        flash[newPhysicalBlock].valid[page] = 1;
    }
}
```

Test

writePage & readPage test

GC test

eraseBlock test

```c
int main() {
    initFTL();

    char buffer[PAGE_SIZE] = "Hello";
    char readBuffer[PAGE_SIZE];

    writePage(10, 5, buffer);
    readPage(10, 5, readBuffer);
    printf("Read data: %s\n", readBuffer);

    strcpy(buffer, "World");
    writePage(10, 5, buffer);
    readPage(10, 5, readBuffer);
    printf("Read data: %s\n", readBuffer);

    eraseBlock(10);
    readPage(10, 5, readBuffer);
    printf("Read data: %s\n", readBuffer);

    return 0;
}
```

Test result :

```
sslab@sslab-System-Product-Name:~/文档/output$ ./"common"
Read data: Hello
Read data: World
Read data:
```

Design a NAND FTL

Thank you!
Q & A ?

2023.08.25
쥬용지에
arashio1111@dankook.ac.kr