WinCE 5.0 MLC Solution Pre-Programming Guide

WinCE 5.0 MLC NAND Solution (PocketMory) Porting Guide

Copyright © 2006 Samsung Electronics Co, Ltd. All Rights Reserved.

Contact Email : mobilesol.cs@samsung.com

Revision History

Date	Version	Author	Amendment
2007.06.18	0.1	HM.SEO	Master Copy
2007.06.22	0.2	HM.SEO	Master Copy
2007.06.26	0.3	HM.SEO	Master Copy
2007.06.29	0.4	HM.SEO	Master Copy

Contents

1 Overview

This documentation explains below listed articles.

- How to make SMD binary Image for NAND pre-programming.

- The structure of SMD binary.

- How to programming each parts (Block 0, TOC, Eboot, OS, Filesystem, VFL Info Area)

- How to control the bad block.

2 Make SMD Binary Image Using PMSMDMaker

2.1 PMSMDMaker Application

The PMSMDMaker application can make SMD binary for MLC NAND flash Mass-production.

You have to fill all information before make SMD binary.

And press the "Make SMD File" button. Then you can make SMD binary file.
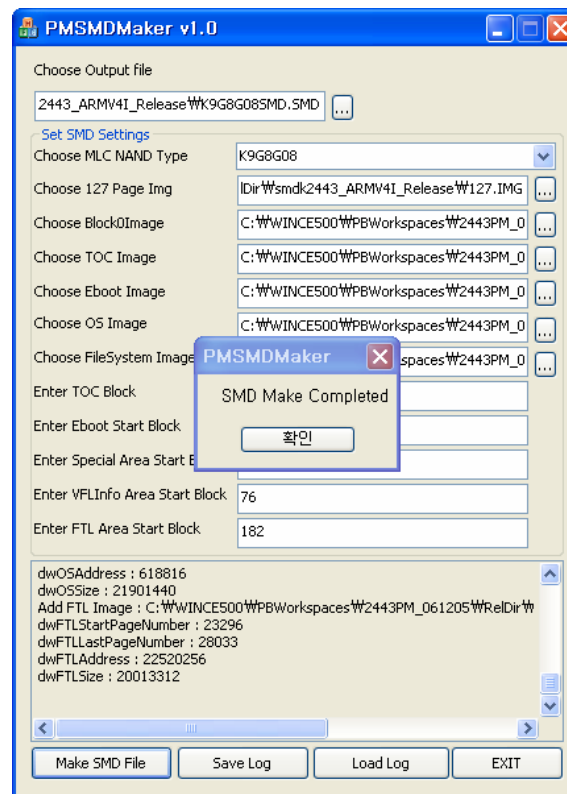
2.2 MLC NAND Type

Choose the NAND flash type what you using.

The PMSMDMaker v1.0 can support listed 6 MLC nand flash types.

- K9G4G08
- K9G8G08
- K9L8G08
- K9LAG08
- K9GAG08
- K9HAG08

2.3   Block 0 Image

Block 0 image is combined nbl1 and nbl2 image.

Nbl1 image will be written on page number 0 and 1 on block 0. This image is copied to stepping stone automatically at boot up time.

Nbl2 image will be written on from page number 6. Don't write page number 2, 3, 4, 5. Because the page number 4 and 5 is paired page of Nbl1.

The Nbl1 have to load the Nbl2 from page number 6.

The block0 page mapping information is as below.

Page Number	
0, 1	Nbl1
6~125	Nbl2
126	Bad information for VFLInfoSection.
127	PocketMory Version Information

2.4   127 Page Image

The Block 0 page 127 has PocketMory Version information.

This image have only 4 byte version information like as below

```
 Page Number [99999999 to exit] = 127

Main Data
================================================= Sector 0
57 30 32 32 ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
================================================= Sector 1
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
================================================= Sector 4
ff 00 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
f9 25 35 2b c0 70 70 00 d7 ec 33 c6 69 53 80 00
d7 ec 33 c6 69 53 80 00 d7 ec 33 c6 69 53 80 00
41 5a 90 f0 36 a4 88 00 41 5a 90 f0 36 a4 88 00
```

2.5   TOC Image

The smdk2443 BSP default TOC dump image is in BSP\Doc directory.

You can make it easy using FIL read function in our Eboot code.

2.6   Eboot Image

Choose Eboot.nb0 image.

2.7   OS and Filesystem Image

Choose OSIMG.IMG file and FTLIMG.IMG file.

It will explain from next page for about how to make those files.

2.8   Enter the block address for each regions

Set the TOC Block, Eboot Block, Special Area Start Block, VFLInfo Block, FTL Start Block.

2.9   Make SMD file button

To make SMD binary image, use this function

2.10   Save Log button

This function is for save log.

2.11   Load Log button

This function is for load previous saved configuration.

3 How to extracts OS image and File System Image from target device?

3.1 DumpImage.exe Application

You need some application to extract OS and File system image.

Sample code is in "SMDK2443\Src\Apps\DumpImage".

You have to change some definition on DumpImage application source code.

At first you have to change the block address for OS area like as below sample.

```
#define PAGESIZE2K                1
#define SPECIAL_AREA_START_L      10
#define SPECIAL_AREA_SIZE_L       50
#define FTL_AREA_START_L          166
#define FTL_AREA_SIZE_L           1882
#define PAGES_PER_SUBLK_L         256

#define SECTORS_PER_PAGE_L        4
#define BYTES_PER_SECTOR_L        512
#define BYTES_PER_SPARE_L         16
#define BYTES_PER_SPARE_PAGE_L    128


#define USE2PLANE_L
```

This sample is for K9LAG08 MLC NAND Flash.

You can get these values with booting message.

If you use our default BSP, then you can get below message through UART debug message.

```
[FIL] ################
[FIL]   MID    = 0xec
[FIL]   DID    = 0xd5
[FIL]   HID[0] = 0x55
[FIL]   HID[1] = 0x25
[FIL]   HID[2] = 0x68
[FIL] ################
[FIL] ############################
[FIL]   FIL Global Information
```

```
[FIL]   BANKS_TOTAL = 2
[FIL]   BLOCKS_PER_BANK = 2048
[FIL]   TWO_PLANE_PROGRAM = 1
[FIL]   SUPPORT_INTERLEAVING = 1
[FIL]   SUBLKS_TOTAL = 2048
[FIL]   PAGES_PER_SUBLK = 256
[FIL]   PAGES_PER_BANK = 262144
[FIL]   SECTORS_PER_PAGE = 4
[FIL]   SECTORS_PER_SUPAGE = 8
[FIL]   SECTORS_PER_SUBLK = 2048
[FIL]   USER_SECTORS_TOTAL = 3799040
[FIL]   ADDRESS_CYCLE = 5
[FIL]   #############################

[INFO] WMR_AREA_SIZE = 10
[INFO] SPECIAL_AREA_START = 10
[INFO] SPECIAL_AREA_SIZE = 50
[INFO] VFL_AREA_START = 60
[INFO] VFL_AREA_SIZE = 106
[INFO] VFL_INFO_SECTION_START = 60
[INFO] VFL_INFO_SECTION_SIZE = 4
[INFO] RESERVED_SECTION_START = 64
[INFO] RESERVED_SECTION_SIZE = 102
[INFO] FTL_INFO_SECTION_START = 166
[INFO] FTL_INFO_SECTION_SIZE = 10
[INFO] LOG_SECTION_SIZE = 7
[INFO] FREE_SECTION_START = 176
[INFO] FREE_SECTION_SIZE = 17
[INFO] FREE_LIST_SIZE = 3
[INFO] DATA_SECTION_START = 193
[INFO] DATA_SECTION_SIZE = 1855
[INFO] FTL_AREA_START = 166
[INFO] FTL_AREA_SIZE = 1882
```

This DumpImage.c file is divided by 3 parts.

At first step, it will create the output file.

The below sample code will make OSIMG.IMG file to "Storage Card" folder.

The "Storage Card" folder can be SD Memory card.

```
// Extract OS Image routine...


RETAILMSG(1,(TEXT("Dump OS Image\r\n")));


pFileOutPtr = fopen("Storage Card/OSIMG.IMG", "wb");
if (!pFileOutPtr)
{
        RETAILMSG(1,(TEXT("Storage Card/OSIMG.IMG file is not opened.!!!\r\n")));
        goto Fail;
}


fseek(pFileOutPtr, 0, SEEK_SET);
```

The second step is scanning all OS area to check last valuable data. It checks all FF area.

```
dwOSStartPage  = SPECIAL_AREA_START_L * PAGES_PER_SUBLK_L;
dwOSEndPage  =  (  SPECIAL_AREA_START_L  +  SPECIAL_AREA_SIZE_L  )  *
PAGES_PER_SUBLK_L - 1;


RETAILMSG(1,(TEXT("OS Area is from %d to %d\r\n"), dwOSStartPage, dwOSEndPage));
RETAILMSG(1,(TEXT("Scanning...\r\n")));
// Scan All FF area
for ( nVpn = dwOSEndPage; nVpn > dwOSStartPage; nVpn-- )
{
        memset(pData, 0xFF, sizeof(pData));
        memset(pSpare, 0xFF, sizeof(pSpare));
        ReadPage( nVpn, pData, pSpare );
        if ( CheckAllFF(pData, sizeof(pData)) == TRUE32 && CheckAllFF(pSpare,
sizeof(pSpare)) ) // TRUE32 means All FF
        {
                continue;
        }
        else
```

```
            {
                    break;
            }
}
dwOSEndPage = nVpn + 1;


RETAILMSG(1,(TEXT("Read Sector from %d to %d\r\n"), dwOSStartPage, dwOSEndPage));
```

And the last step is for dump image using ReadPage function.

```
for( nVpn=dwOSStartPage; nVpn<dwOSEndPage; nVpn++ )
{
        memset(pData, 0xFF, sizeof(pData));
        memset(pSpare, 0xFF, sizeof(pSpare));

        ReadPage( nVpn, pData, pSpare );


#ifdef USE2PLANE_L
        // Write First plane
        fwrite(pData,    sizeof(char),    (SECTORS_PER_PAGE_L*BYTES_PER_SECTOR_L),
pFileOutPtr);
        fwrite(pSpare,   sizeof(char),    (SECTORS_PER_PAGE_L*BYTES_PER_SPARE_L),
pFileOutPtr);

        // Write Second plane
        fwrite(pData+(SECTORS_PER_PAGE_L*BYTES_PER_SECTOR_L),    sizeof(char),
(SECTORS_PER_PAGE_L*BYTES_PER_SECTOR_L), pFileOutPtr);
        fwrite(pSpare+(SECTORS_PER_PAGE_L*BYTES_PER_SPARE_L),    sizeof(char),
(SECTORS_PER_PAGE_L*BYTES_PER_SPARE_L), pFileOutPtr);
#else
        // Write First plane
        fwrite(pData,    sizeof(char),    (SECTORS_PER_PAGE_L*BYTES_PER_SECTOR_L),
pFileOutPtr);
        fwrite(pSpare,   sizeof(char),    (SECTORS_PER_PAGE_L*BYTES_PER_SPARE_L),
pFileOutPtr);
#endif
```

```
}

if (pFileOutPtr) fclose(pFileOutPtr);


RETAILMSG(1,(TEXT("Dump OS Image Finished\r\n")));
```

The ReadPage use Kernel IO control to read each page.

It can not read main data area and spare area at same time.

That's why the Kernel IO control has to be called twice.

```
void ReadPage(DWORD nVpn, unsigned char * pData, unsigned char * pSpare)
{
        Buffer          pBuf;
        VFLPacket       stPacket;
        UINT32          nResult;


#ifdef PAGESIZE4K
        #ifdef USE2PLANE_L
                pBuf.nBitmap = 0xFFFF;
        #else
                pBuf.nBitmap = 0xFF;
        #endif
#else
        #ifdef USE2PLANE_L
                pBuf.nBitmap = 0xFF;
        #else
                pBuf.nBitmap = 0xF;
        #endif
#endif


        pBuf.eStatus = BUF_AUX;
        pBuf.nBank = 0;


        // Read Main Data Area
        pBuf.pData = pData;
        pBuf.pSpare = NULL;
```

```
do {
    /* VFL_Read */
    stPacket.nCtrlCode   = PM_HAL_VFL_READ;
    stPacket.nVbn        = 0;              // Not used
    stPacket.nVpn        = nVpn;
    stPacket.pBuf        = &pBuf;
    stPacket.nSrcVpn     = 0;
    stPacket.nDesVpn     = 0;
    stPacket.bCleanCheck= FALSE32;

    KernelIoControl(IOCTL_POCKETSTOREII_CMD,   /* IO Control Code */
                    &stPacket,              /* Input buffer (Additional Control Code) */
                    sizeof(VFLPacket),    /* Size of Input buffer */
                    NULL,                 /* Output buffer */
                    0,                    /* Size of Output buffer */
                    &nResult);            /* Error Return */

    if (nResult != VFL_SUCCESS)
    {
        RETAILMSG(1,   ((TEXT("[VFLP:ERR]        VFL_Read()    failure.    ERR
Code=%x\r\n"), nResult)));
        break;
    }

} while(0);

// Read Spare Area
pBuf.pData = NULL;
pBuf.pSpare = pSpare;

do {
    /* VFL_Read */
    stPacket.nCtrlCode   = PM_HAL_VFL_READ;
    stPacket.nVbn        = 0;              // Not used
    stPacket.nVpn        = nVpn;
```

```
            stPacket.pBuf        = &pBuf;
            stPacket.nSrcVpn     = 0;
            stPacket.nDesVpn     = 0;
            stPacket.bCleanCheck= FALSE32;


            KernelIoControl(IOCTL_POCKETSTOREII_CMD,   /* IO Control Code */
                            &stPacket,            /* Input buffer (Additional Control Code) */
                            sizeof(VFLPacket),    /* Size of Input buffer */
                            NULL,                 /* Output buffer */
                            0,                    /* Size of Output buffer */
                            &nResult);            /* Error Return */


        if (nResult != VFL_SUCCESS)
        {
            RETAILMSG(1,    ((TEXT("[VFLP:ERR]        VFL_Read()    failure.    ERR
Code=%x\r\n"), nResult)));
            break;
        }

    } while(0);
}
```

Build "DumpImage.C" file and copy the "DumpImage.exe" file to target device and execute it.

Then you can get the OS image and FTL image with below UART output message.

```
Dump OS Image
OS Area is from 2560 to 11007
Scanning...
Read Sector from 2560 to 7873
Dump OS Image Finished
Dump FTL Image
FTL Area is from 38144 to 524287
Scanning...
Read Sector from 38144 to 53470
Dump FTL Image Finished
```

4 Guides for Mass-product

4.1 PocketMory Block Mapping Example

The PocketMory block structure is seems as below table.

Bloc Number (Logical Block)	Image Name
0	Block 0 Image
1~2	TOC block
3~9	Eboot Block
10~59	OS Area (Special Area)
60~63	VFL Info Area
64~165	VFL Reserved Area (for Bad Block remapping)
166~2047	FTL Area

This block mapping example is based on our SMDK2443 BSP with K9LAG08 MLC NAND flash. The block number is logical block. If the MLC NAND flash driver can support 2-plane programming, then the logical 1 block has physical 2 blocks. If the MLC NAND flash can support inter-leaving operation, the logical 1 block has physical 4 blocks.

For understanding about 2-plane programming and Inter-leaving, below description is quoted from MLC NAND flash specification.

Quotation from the K9LAG08U0M_0.8_MLC specification:

K9LAG08U0M is arranged in four 4Gb memory planes. Each plane contains 2,048 blocks and 2112 byte page registers. This allows it to perform simultaneous page program and block erase by selecting one page or block from each plane. The block address map is configured so that two-plane program/erase operations can be executed by dividing the memory array into plane 0~1 or plane 2~3 separately.

For example, two-plane program/erase operation into plane 0 and plane 2 is prohibited. That is to say, two-plane program/erase operation into plane 0 and plane 1 or into plane 2 and plane 3 is allowed
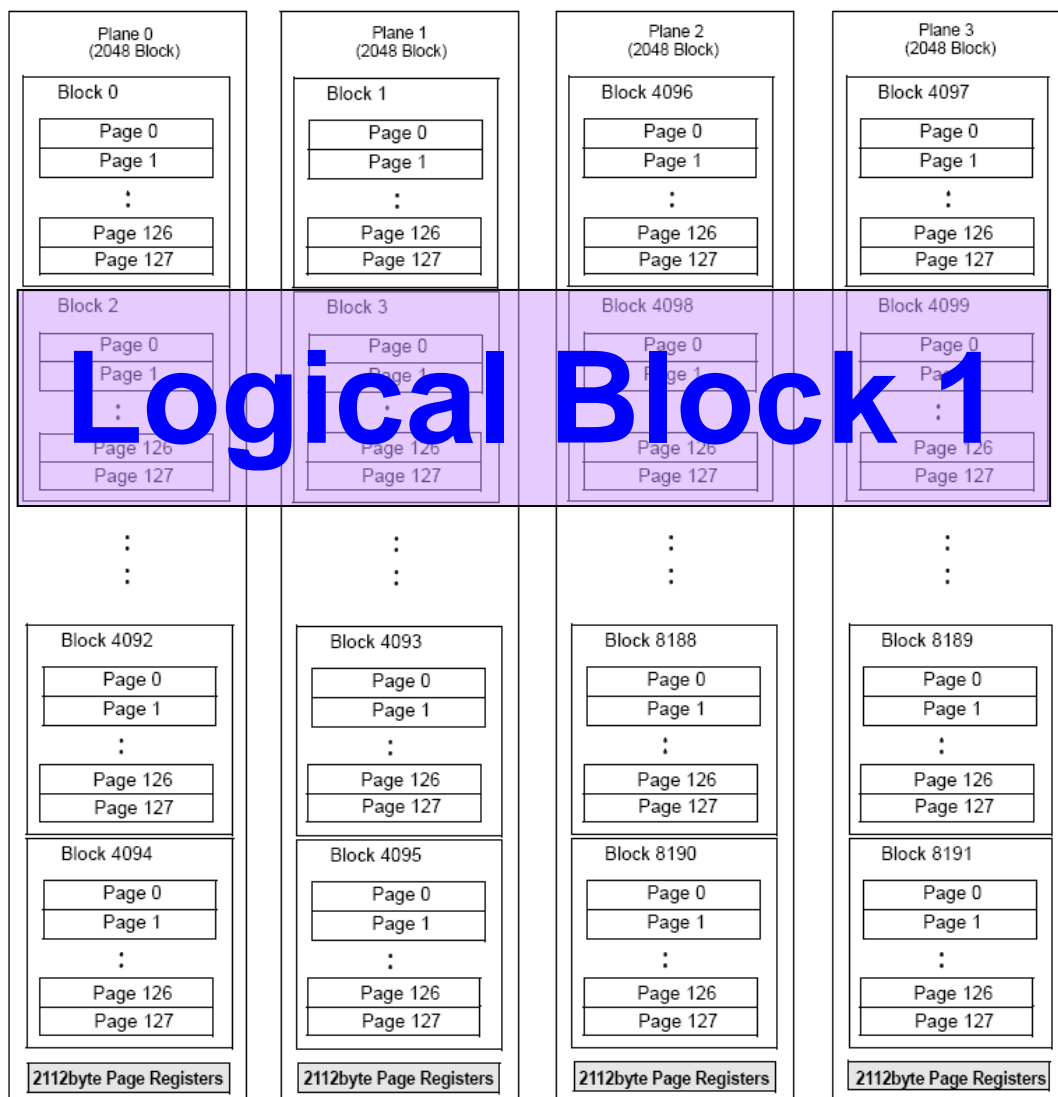
For example, if the OS start block is from logical block 1, the OS image write procedure is as below.

Block 2 Page 0 -> Block 3 Page 0 -> Block 4098 Page 0 -> Block 4099 Page 0 ->

Block 2 Page 1 -> Block 3 Page 1 -> Block 4098 Page 1 -> Block 4099 Page 1 -> …

If the file system start block is logical block N, then the physical block is 2*N, 2*N + 1, (Total Block Number / 2) + 2*N, (Total Block Number / 2) + 2*N + 1.

4.2   The SMD binary format

The SMD binary format is follow the structure of SMDHEADER.

The structure of SMDHEADER is as below.

```
typedef struct _SMDHEADER {
        DWORD dwSig;

        BOOL32 bUse2Plane;
        BOOL32 bUseInterLeaving;
        BOOL32 bUseCEInterLeaving;

        DWORD dwTotalBlockNumber;
        DWORD dwMainPageSize;
        DWORD dwSparePageSize;

        DWORD dwSpecialAreaStartBlock;
        DWORD dwVFLInfoSectionStartBlock;
        DWORD dwReservedSectionStartBlock;
        DWORD dwReservedSectionBlockNumber;
        DWORD dwFTLInfoSectonStartBlock;

        DWORD dw2ndBLStartPageNumber;
        DWORD dw2ndLLastPageNumber;

        DWORD dwTOCPageNumber;

        DWORD dwEBootStartPageNumber;
        DWORD dwEBootLastPageNumber;

        DWORD dwOSStartPageNumber;
        DWORD dwOSLastPageNumber;

        DWORD dwVFLInfoSectionPageNumber;

        DWORD dwFTLStartPageNumber;
        DWORD dwFTLLastPageNumber;
```

```
    DWORD dwStepLDRAddress;
    DWORD dwStepLDRSize;
    DWORD dw2ndBLAddress;
    DWORD dw2ndBLSize;
    DWORD dw127Address;
    DWORD dw127Size;
    DWORD dwTOCAddress;
    DWORD dwTOCSize;
    DWORD dwEbootAddress;
    DWORD dwEbootSize;
    DWORD dwOSAddress;
    DWORD dwOSSize;
    DWORD dwFTLAddress;
    DWORD dwFTLSize;
    DWORD pad[((2048+64)/4)-36];        // Make Total Size to 2048+64
} SMDHEADER, *PSMDHEADER;
```

Each member values are defined as blow.

- dwSig : The signature of SMD binary. (ex : PM10)

- bUse2Plane : Option for using 2 plane or not. Every MLC NAND flash support 2-plane programming.

- bUseInterLeaving : Option for using Inter-leaving or not. The MLC DDP and QDP can support Inter-leaving programming.

- bUseCEInterLeaving : If this value is TRUE, this nand device has 2 CE. This MLC NAND flash will be QDP. (Ex:K9HAG08, K9HBG08)

- dwTotalBlockNumber : The total block number for MLC NAND flash. This block number is physical block number.

- dwMainPageSize : This value indicates the main area size for each page. This value is 2048 or 4096.

- dwSparePageSize : This value indicates the spare area size for each page. This value is 64 or 128.

- dwSpecialAreaStartBlock : The Special Area is used for OS region. This block number is logical block number. If the NAND flash support 2-plane programming, for example, the logical block 10 is physical block 20 and 21. If the NAND flash support 2-plane programming and inter-leaving, the logical block 10 is physical block 20, 21, 4096 + 20 and 4096 + 21. (Assume to dwTotalBlockNumber is 8192)

- dwVFLInfoSectionStartBlock : This value indicates the VFLInfo section block. VFLInfo section has bad block information and mapping table. The VFLInfo structure will described on next chapter. This block number is logical block number.

- dwReservedSectionStartBlock : This block number is reserved area start block for bad block re-mapping. This block number is logical block number.

- dwReservedSectionBlockNumber : The reserved block numbers for bad block re-mapping. This number is depends on NAND flash specification.

- dwFTLInfoSectonStartBlock : This block number indicates the file system area start block. This number is logical block number.

- dw2ndBLStartPageNumber : This number is indicates the start page number of Nbl2 (Second Bootloader). This page number is physical page number. Because the 2nd boot loader is programmed on physical block 0 of plane 0 only. The physical block 0 is only guaranteed by NAND flash manufacture. In normal case this number is 6.

- dw2ndBLLastPageNumber : This page number is last page number for 2nd boot loader. This page number is physical page number.

- dwTOCPageNumber : This number is page number for TOC information. This page number is logical page number. The TOC information is only programmed on plane 0.

- dwEBootStartPageNumber : This number is page number for Eboot area start page. This page number is logical page number. The Eboot information is only programmed on plane 0.

- dwEbootLastPageNumber : This number is size of Eboot regions. This page number is logical page number.

- dwOSStartPageNumber : This page number is start of Special area. This page number is logical number. In case of 2-plane programming, if this page number is N, the physical page number is N*2 and N*2 + 128. In case of inter-leaving and 2-plane programming, if this page number is N, the physical page number is N*2, N*2 + 128, (4096*128) + N*2 and (4096*128) + N*2 + 128.

- dwOSLastPageNumber : This page number is last page number of OS image regions. This page number is logical page number.

- dwVFLInfoSectionPageNumber : This page number is start of VFLInfo section. This page number is logical number.

- dwFTLStartPageNumber : This page number is start of file system area. This page number is logical number.

- dwFTLLastPageNumber : This page number is last page of file system area. This page number is logical number.

- dwStepLDRAddress : This number indicates the start address of step loader image in

SMD binary image.

- dwStepLDRSize : This number is size of step loader.

- dw2ndBLAddress : This number indicates the start address of 2nd boot loader in SMD binary image.

- dw2ndBLSize : This number is size of 2nd boot loader.

- dw127Address : This number indicates the start address of 127 page image in SMD binary image.

- dw127Size : This number is size of 127 page image.

- dwTOCAddress : This number indicates the start address of TOC image in SMD binary image.

- dwTOCSize : This number is size of TOC image.

- dwEbootAddress : This number indicates the start address of Eboot image in SMD binary image.

- dwEbootSize : This number is size of Eboot image.

- dwOSAddress : This number indicates the start address of OS image in SMD binary image.

- dwOSSize : This number is size of OS image.

- dwFTLAddress : This number indicates the start address of file system image in SMD binary image.

- dwFTLSize : This number is size of file system image.

4.3 The VFLCxt Section format

The VFLCxt binary format is follow the structure of VFLCxtLocalUse.

The structure of VFLCxtLocalUse is as below.

```
typedef struct {
        UINT32          nGlobalCxtAge;          /* age for FTL meta information search */
        UINT16          aFTLCxtVbn[64 + 1];     // WMR_NUM_MAPS_MAX = 64
        UINT16          nPadding;

        UINT32  nCxtAge;                        /* context age 0xFFFFFFFF --> 0x0    */
        UINT32  nCxtLocation;                   /* page offset, context is located   */

        /* this data is used for summary                                   */
        UINT16  nNumOfInitBadBlk;               /* the number of initial bad blocks */
        UINT16  nNumOfWriteFail;                /* the number of write fail          */
        UINT16  nNumOfEraseFail;                /* the number of erase fail          */

        /* bad blocks management table & good block pointer */
        UINT16  nBadMapTableMaxIdx;
        UINT16  aBadMapTable[400 * 2];          // 400 : WMR_MAX_RESERVED_SIZE
        UINT8   aBadMark[16384 / 8 / 8];        //  16384  :  WMR_MAX_VB,  last  8  :
BAD_MARK_COMPRESS_SIZE

        /* bad blocks management table within VFL info area */
        UINT8   aBadInfoBlk[4];                 // 4 : VFL_INFO_SECTION_SIZE
} VFLCxtLocalUse;
```

If the MLC NAND flash can support inter-leaving programming, it needs two VFLCxt information. The VFLCxt1 has block information from physical block number 0 to 4095. And the VFLCxt2 has block information from block number 4096 to 8191. (In case of : K9LAG08)

- nGlobalCxtAge : This value is used internally. Set this value to 0 for VFLCxt1, and set to 1 for VFLCxt2.
- aFTLCxtVbn[64 + 1] : This array is filled with dwFTLInfoSectionStartBlock number. This array can be filled with below expression. The number 10 is almost fixed value. Current PocketMory Solution's FTL_INFO_SECTION_SIZE is fixed to 10.

```
for (dwBlockNumber = 0; dwBlockNumber < 10; dwBlockNumber++)     //     10     :
```

```
FTL_INFO_SECTION_SIZE
{
        stVFLCxtL[dwBankCnt].aFTLCxtVbn[dwBlockNumber]
                = stSMDHeaderInfo.dwFTLInfoSectonStartBlock + dwBlockNumber;
}

for (; dwBlockNumber < (64 + 1); dwBlockNumber++) // 64 : WMR_NUM_MAPS_MAX
{
        stVFLCxtL[dwBankCnt].aFTLCxtVbn[dwBlockNumber] = 0xFFFF;
}
```

- nPadding : This value is just for make 4 byte align. This value is set by 0.

- nCxtAge : This value should be 0xFFFFFFFF.

- nCxtLocation : This value's initial value is 3. This value indicates the VFLCxt information location. If the VFLCxt is located on first of VFL Info AREA start block, then this value is 3. Because, in the PocketMory source code, it reference the (nCxtLocation+1)%4 th indexed block. If the nCxtLocation value is 1, then the VFLCxt information is located on 3rd block from VFL Info AREA start block. And it means the 1st and 2nd block of VFL Info Area are bad blocks.

- nNumOfInitBadBlk : This value's initial value is 0. This value is number of Initial bad block. If the total initial bad block number is 5, then this value is 5.

- nNumOfWriteFail : This value should be 0.

- nNumOfEraseFail : This value should be 0.

- nBadMapTableMaxIdx : This value is calculated with below expression.

```
stVFLCxtL[dwBankCnt].nBadMapTableMaxIdx
        = stVFLCxtL[dwBankCnt].nNumOfInitBadBlk/2
        + (stVFLCxtL[dwBankCnt].nNumOfInitBadBlk%2);
```

- aBadMapTable[400 * 2] : This array is bad block mapping table. If initial bad block is found, set the bad block number to this array. For example, if the block number 30 and 44 are bad block and it is first two initial bad block, then the aBadMapTable[0] value is 30 and aBadMapTable[1] value is 44. If the initial bad block is in reserved area, this array indicated value is 0xFFFF. For example, if the reserved area start block number is 100 and if the block number 105 is bad block, then the aBadMapTable[5] value is 0xFFFF. The number 5 is calculated by "105 – 100". At this time, the bad block number is physical block number. For second die, incase of inter-leaving programming, the block number is counted from half of block. For example, if the total block number is 8192 and the 4096+55 block is bad block, and it is first bad block of 2nd die, then the

stVFLCxtL2.nBadMapTable[0] value is 55 not 4151.

- aBadMark[16384 / 8 / 8] : This array's each bit indicates the bad block information. This array is calculated with below expression. This array is initialized with 0xFF and the bit will be changed from 1 to 0.

stVFLCxtL1.aBadMark[dwBlockNumber/8/8]

&= ~((UINT8)(1<<(7-((dwBlockNumber/8)%8))));;

For example, let see the below aBadMark area information. This sample MLC NAND flash has bad block on block number 2933.

stVFLCxtL1.aBadMark[2933/8/8]

&= ~((UINT8)(1<<(7-((2933/8)%8))));

stVFLCxtL1.aBadMark[45]

&= ~((UINT8)(1<<(7-6))); // => 1111101b = 0xfd

Therefore the aBadMark area information looks as below.

- aBadInfoBlk[4] : The bad block info block is 4 blocks. This array is indicates the VFLInfo section is bad block or not. If the VFLInfo section is not Bad block then the value is 0. If it is bad block then the value is 0x01. Finally, this value is written to 126page of block 0.

When write the VFLCxt page, it have to make spare area information. The page lay out is structured as below.

Page Layout for 2Kbytes/Page NAND Flash Device:

0~511	512~1023	1024~1535	1536~2047	2048~2111
Sector 0 512B	Sector 1 512B	Sector 2 512B	Sector 3 512B	Spare Area 64B

Spare Area Layout for 2Kbyte/Page NAND Flash Device:

0(1)	1(1)	2~3(2)	4~15(12)	16~23(8)	24~39(8)	32~39(8)	40~47(8)	48~55(8)	56~63(8)
Bad Mark	Clean Mark	Reserved	Spare Context	Sector0 ECC	Sector1 ECC	Sector2 ECC	Sector3 ECC	Spare ECC	Spare ECC copy

Page Layout for 2Kbytes/Page NAND Flash Device:

0~511	512~1023	1024~1535	1536~2047	2048~2559	2560~3071	3072~3583	3584~4095	4096~4224
Sector 0 512B	Sector 1 512B	Sector 2 512B	Sector 3 512B	Sector 4 512B	Sector 5 512B	Sector 6 512B	Sector 7 512B	Spare Area 128B

Table 4-1. MLC Page Layout 4K page (4096+128 byte)

- Spare Area Layout for 2Kbyte/Page NAND Flash Device:

0 (1)	1 (1)	2~3 (2)	4~23 (20)	24~31 (8)	32~39 (8)	40~47 (8)	48~55 (8)	56~63 (8)	64~71 (8)	72~79 (8)	80~87 (8)	88~95 (8)	96~103 (8)
Bad Mark	Clean Mark	Reserved	Spare Context	Sector0 ECC	Sector1 ECC	Sector2 ECC	Sector3 ECC	Sector4 ECC	Sector5 ECC	Sector6 ECC	Sector7 ECC	Spare ECC	Spare ECC copy

Table 4-2. MLC Spare Area layout 4K page (128 Byte)

For VFLCxt area, those spare area fields have below values.

- Bad Mark : [0xFF]
- Clean Mark : [0x00]
- Reserved : [0xFF][0xFF]
- Spare Context : [0xFF] [0xFF] [0xFF] [0xFF][0x00] [0xFF] [0xFF] [0xFF] [0xFF] [0xFF] [0xFF] [0xFF] – The 5th 0x00 is for confirm mark. Incase of 4 KByte page size, there are 8 more [0xFF].
- Sector0 ECC : This value is calculated by 4bit ECC algorithm. This ECC code is for 512 Bytes of sector 0. The 4 bit ECC algorithm will return the 7 bytes parity code. The 8th byte is 0x0.
- Sector1,2,3,4,5,6,7 ECC : Same with above.
- Spare ECC : This value is calculated by 4bit ECC algorithm. This ECC code is for spare area from 2 to 47. Our 4bit ECC algorithm needs to be inputted the 512bytes data to generate ECC parity code. Therefore the (512-46 = 466) bytes deficient have to be filled up with 0xFF. Incase of 4K Byte page size, the area from 2 to 87 have to be calculated with 4 bit ECC algorithm. Therefore the (512-86 = 426)Bytes have to filled up with 0xFF.
- Spare ECC copy : This value is same with Spare ECC.

4.4   Block 0 Image

The Block 0 Image includes Nbl1 and Nbl2 image.

The Block 0 Image related parameters of SMDHEADER are as below.

- dw2ndBLStartPageNumber

- dw2ndLLastPageNumber

- dw2ndBLAddress

- dw2ndBLSize

The Nbl1 image is written to page 0 and 1, the Nbl2 image is written from page number 6 to size of Nbl2 image. But the Nbl2 images last page is smaller then 126 page. Because the page number 126 and 127 is used for special purpose.

The block 0 image will be programmed like as below picture based on K9LAG08 MLC NAND flash. The plane 0 and 1 is belonging to first die, and plane 2 and 3 is belonging to second die. Each die is base unit for bad block management and inter-leaving programming.
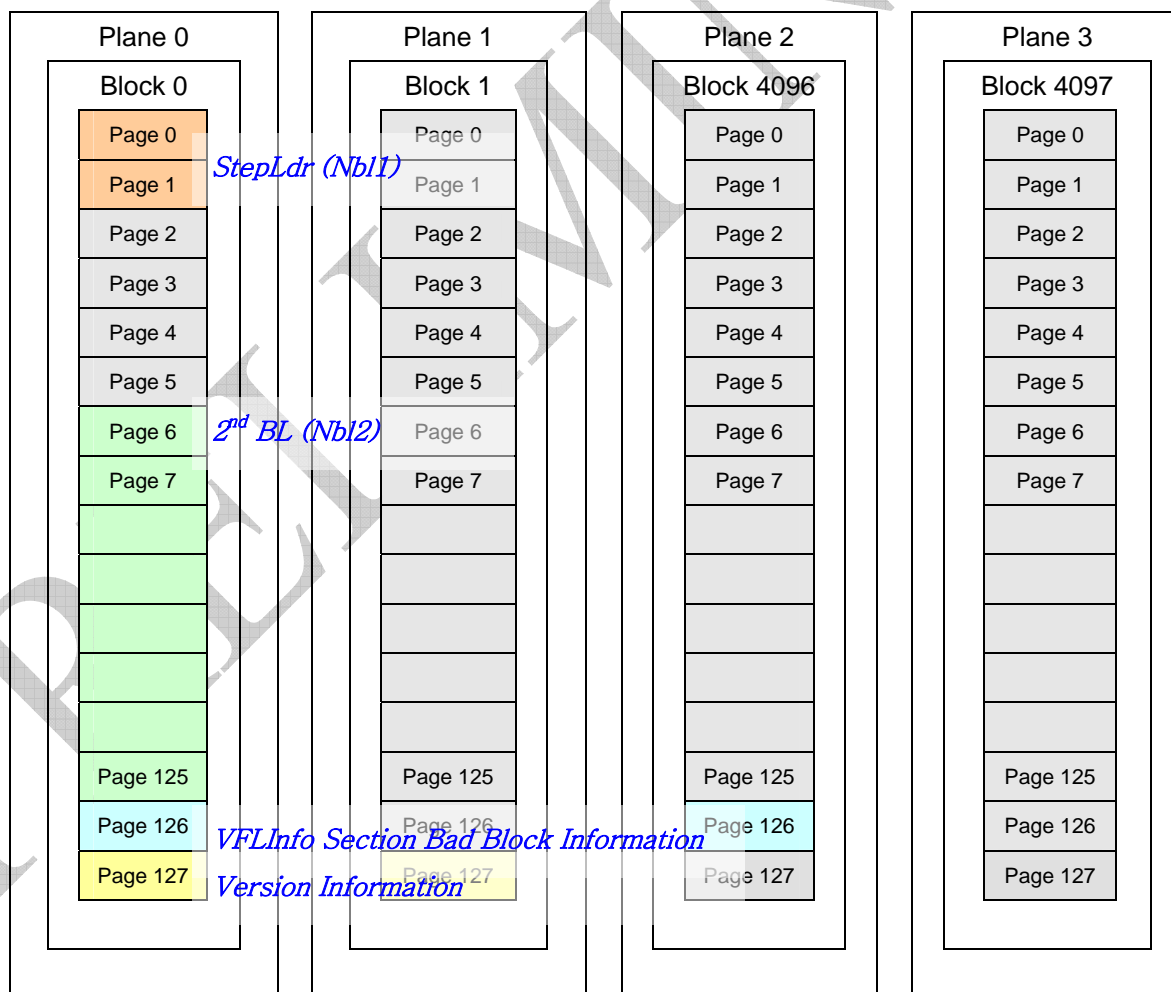
4.4.1 Step loader image

To programming the step loader (Nbl1), the value of dwStepLDRAddress and dwStepLDRSize in SMDHEADER structure have to be referenced. This image has to be programmed on page 0 and 1. The important thing is from the page number 2 to 5 have to be not programmed. It is not allows to write 0xFF.

4.4.2 Second Boot loader image

The second boot loader (Nbl2) will programmed from dw2ndBLStartPageNumber to dw2ndLLastPageNumber. The dw2ndBLStartPageNumber should be page 6 and the dw2ndLLastPageNumber have to smaller then page number 125. The 2nd boot loader image start address of SMD binary is dw2ndBLAddress value, and the 2nd boot loader image size is dw2ndBLSize value.

4.4.3 VFLCxt Bad block info block (126 Page)

The page number 126 has VFLCxt.aBadInfoBlk information. The block 0 page 126 has VFLCxt.aBadInfoBlk for block number from 0 to 4095. The block 4096 page 126 has VFLCxt.aBadInfoBlk for block number from 4096 to 8191. This page contents is depends on initial bad block is in VFLInfo section. Therefore, it needs 4bit ECC algorithm. Most of things for generate spare data is same with VFXCxt spare data generation part.

The first 4 byte data is useful and the remained bytes are filled up with 0xFF.

If the first of VFLCxt Block is bad block, then the array VFLCxt.aBadInfoBlk will be changed from [00][00][00][00] to [01][00][00][00].

```
 Page Number [99999999 to exit] = 126

 Main Data
 ============================================ Sector 0
 00 00 00 00 ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ============================================ Sector 1
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 ============================================ Sector 4
 ff 00 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
 0b 33 33 07 84 f2 f8 00 d7 ec 33 c6 69 53 80 00
 d7 ec 33 c6 69 53 80 00 d7 ec 33 c6 69 53 80 00
 3b f0 3f 49 e4 bc 38 00 3b f0 3f 49 e4 bc 38 00
```

4.4.4 PocketMory Version information (127 Page)

The Page number 127 has PocketMory version information.

This 127 page image related parameters of SMDHEADER are as below.

- dw127Address
- dw127Size

The block 0 page 127 has "220W" data at first 4bytes. The remained data is all 0xFF and the second plane, block 1 page 127, has all 0xFF. The NAND flash fusing machine has to calculate dynamic 4 bit ECC code.

The first 4 byte data is useful and the remained bytes are filled up with 0xFF.

This image is in SMD binary.

```
 Page Number [99999999 to exit] = 127

Main Data
================================================= Sector 0
57 30 32 32 ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
================================================= Sector 1
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

ff ff ff ff                   ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
================================================= Sector 4
ff 00 ff ff ff ff ff ff ff ff ff ff ff ff ff ff
f9 25 35 2b c0 70 70 00 d7 ec 33 c6 69 53 80 00
d7 ec 33 c6 69 53 80 00 d7 ec 33 c6 69 53 80 00
41 5a 90 f0 36 a4 88 00 41 5a 90 f0 36 a4 88 00
```

4.5   TOC Image

The TOC image is used by Eboot.

This TOC image related parameters of SMDHEADER are as below.

- dwTOCPageNumber
- dwTOCAddress
- dwTOCSize

The dwTOCPageNumber is logical page number. If the dwTOCPageNumber is 256 and this MLC NAND flash use 2 plane programming only, then the TOC block number is logically 1 and physically 2. If the physical block number 2 is bad block, then the TOC image has to be written to logical block number 2, physical block number 4.

4.6   Eboot Image

This Eboot image related parameters of SMDHEADER are as below.

- dwEBootStartPageNumber
- dwEBootLastPageNumber
- dwEbootAddress
- dwEbootSize

The dwEBootStartPageNumber is logical page number. If the dwEBootStartPageNumber is 768 and this MLC NAND flash use 2 plane programming only, then the TOC block number is logically 6(768/128 = 6) physically 3(6/2). If the physical block number 6 is bad block, the TOC image has to be written to block number 8(logical block number 4 * 2 plane). It follows bad skip algorithm.
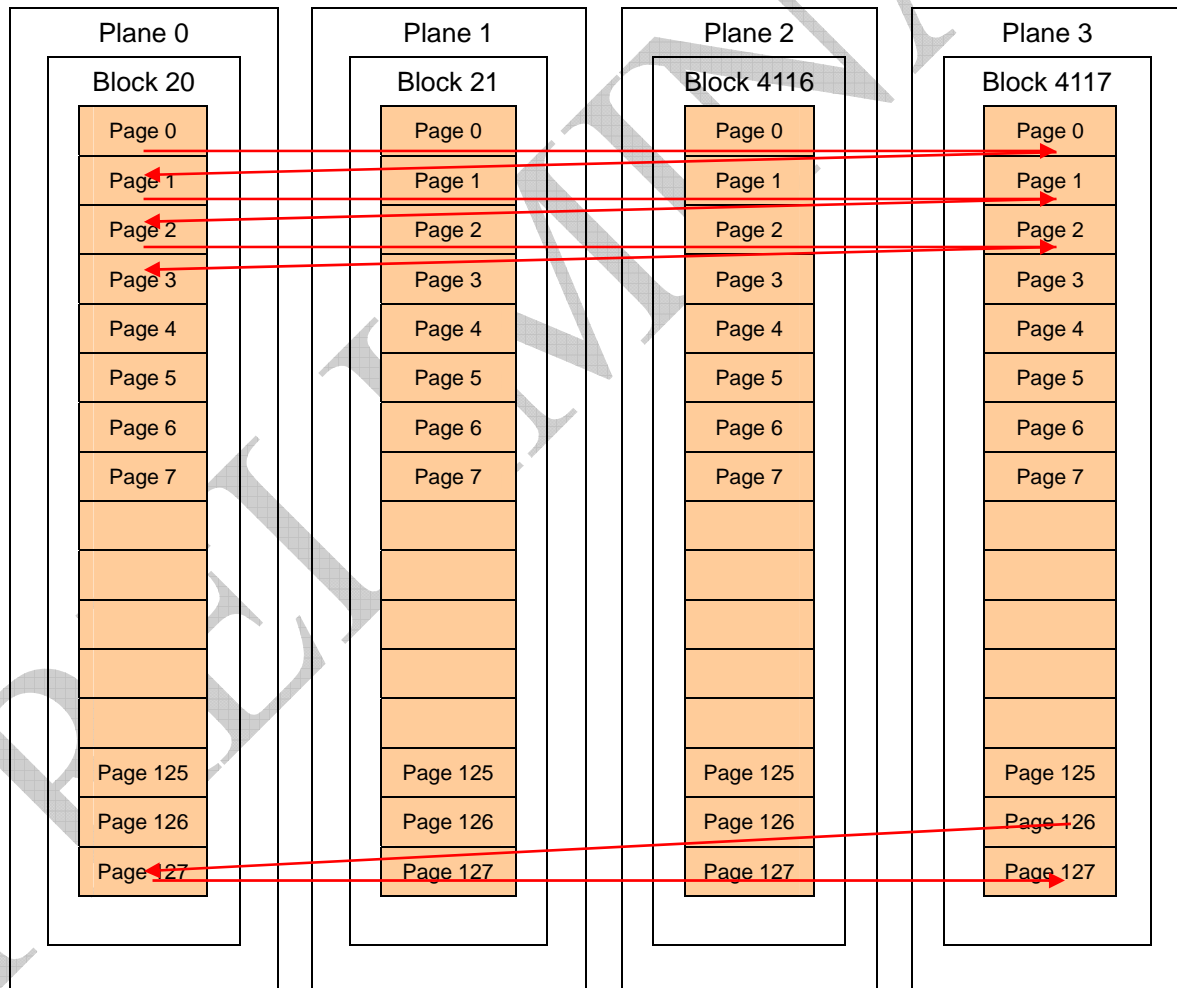
4.7   OS Image

This OS image related parameters of SMDHEADER are as below.

- dwOSStartPageNumber
- dwOSLastPageNumber
- dwOSAddress
- dwOSSize

This page numbers are logical page number. But the different one with Eboot area is the OS area is controlled by VFL function. (The TOC and Eboot area is controlled by FIL function.) Therefore, if the MLC NAND flash can support inter-leaving function, the page numbers in one logical block is 256.

The page write sequence is as below.

In case of use 2-plane programming and inter-leaving:

And it use bad block replacement algorithm. If you met bad block, you should scanning all VFLCxt.aBadMapTable, and if you find the block as marked bad, you have to program the OS

image to indexed block. For example, if you check the block number 60 and this block is bad block, then you have to scan all aBadMapTable array in VFLCxt. And if aBadMapTable[2] value is 60, the bad block replaced block number is dwReservedSectionStartBlock*2 + 2. The reason of multiple by two is the MLC NAND flash can support 2-plane programming.

4.8   File system Image

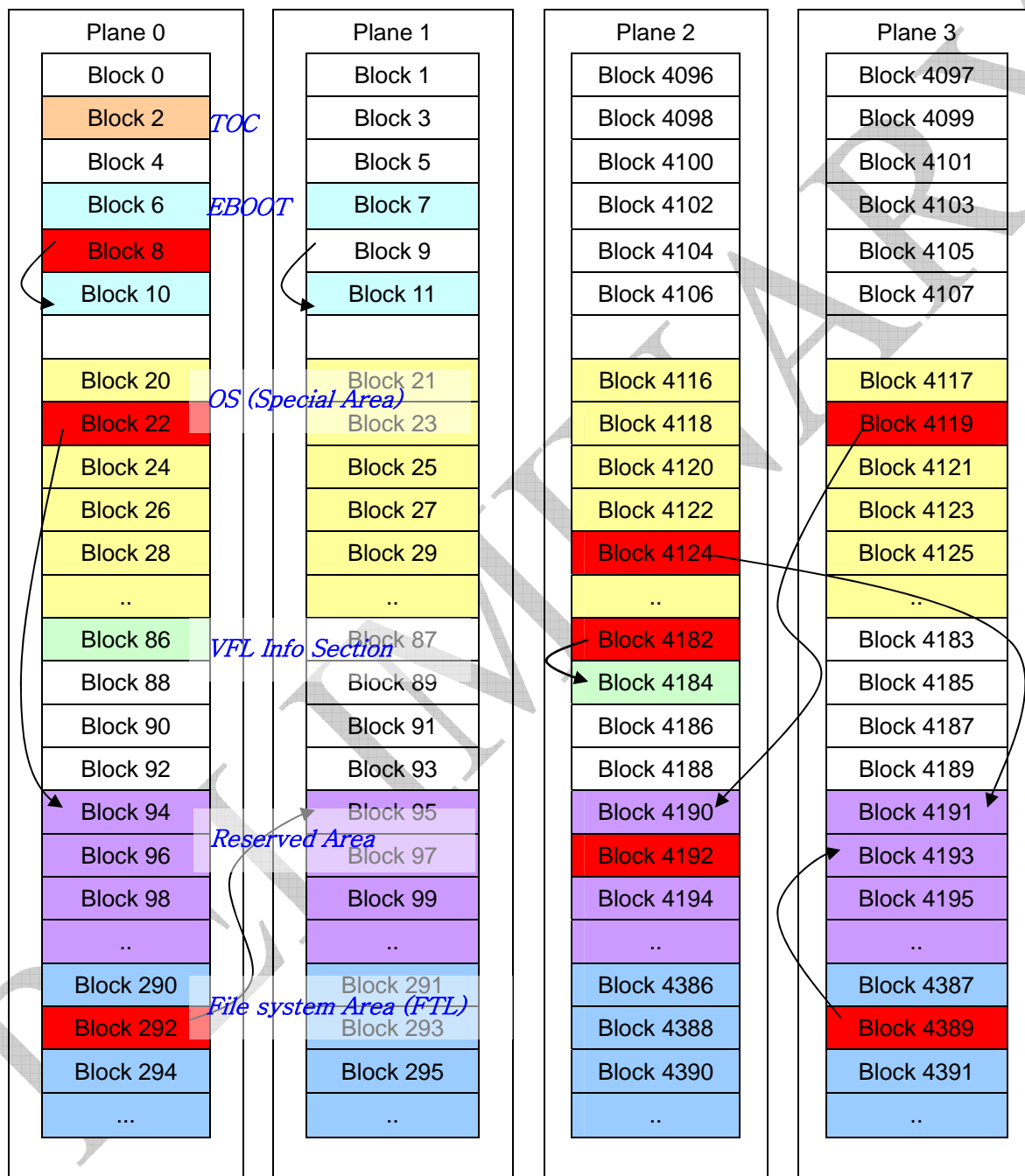This file system image related parameters of SMDHEADER are as below.

- dwFTLInfoSectonStartBlock
- dwFTLStartPageNumber
- dwFTLLastPageNumber
- dwFTLAddress
- dwFTLSize

The programming sequence is same with OS area.

4.9 Total block mapping sample

The below picture shows how to each image can written on each blocks.

The total block mapping seems as below picture:

Plane 0	Plane 1	Plane 2	Plane 3
Block 0	Block 1	Block 4096	Block 4097
Block 2 *TOC*	Block 3	Block 4098	Block 4099
Block 4	Block 5	Block 4100	Block 4101
Block 6 *EBOOT*	Block 7	Block 4102	Block 4103
Block 8	Block 9	Block 4104	Block 4105
Block 10	Block 11	Block 4106	Block 4107
Block 20 *OS (Special Area)*	Block 21	Block 4116	Block 4117
Block 22	Block 23	Block 4118	Block 4119
Block 24	Block 25	Block 4120	Block 4121
Block 26	Block 27	Block 4122	Block 4123
Block 28	Block 29	Block 4124	Block 4125
..	..	..	..
Block 86 *VFL Info Section*	Block 87	Block 4182	Block 4183
Block 88	Block 89	Block 4184	Block 4185
Block 90	Block 91	Block 4186	Block 4187
Block 92	Block 93	Block 4188	Block 4189
Block 94 *Reserved Area*	Block 95	Block 4190	Block 4191
Block 96	Block 97	Block 4192	Block 4193
Block 98	Block 99	Block 4194	Block 4195
..	..	..	..
Block 290 *File system Area (FTL)*	Block 291	Block 4386	Block 4387
Block 292	Block 293	Block 4388	Block 4389
Block 294	Block 295	Block 4390	Block 4391
...	..	..	..

The plane 0 block number 6 and 8 is eboot area. If the block number 8 is bad block, it skip the bad block and write to next block.

The plane 0 block number 22 and 292 is bad block and it is OS area or file system area. This

block replaced to reserved area Block number 94 and 95.

The second die, plane 2 and 3, has 5 bad blocks. It is remapped to block number 4190, 4191, 4193.

If the VFLCxt Block is bad block, it will written to next block in same plane. (4182 -> 4184)

This picture assume the below setting values.

```
BOOL32 bUse2Plane = TRUE32;

BOOL32 bUseInterLeaving = TRUE32;

DWORD dwTotalBlockNumber = 8192;

DWORD dwSpecialAreaStartBlock = 10;

DWORD dwVFLInfoSectionStartBlock = 43;

DWORD dwReservedSectionStartBlock = 47;

DWORD dwReservedSectionBlockNumber = 102;

DWORD dwFTLInfoSectonStartBlock = 149;


DWORD dwTOCPageNumber = 128;


DWORD dwEBootStartPageNumber = 384;

DWORD dwEBootLastPageNumber = ??;     // It depends on Eboot size.


DWORD dwOSStartPageNumber = 2560;     // 10*256

DWORD dwOSLastPageNumber = ??;                    // It depends on OS size.


DWORD dwVFLInfoSectionPageNumber 11008;          // (86/2)*256


DWORD dwFTLStartPageNumber = 37120;              // (290/2)*256

DWORD dwFTLLastPageNumber = ??;       // It depends on Filesystem size
```