SEC- RFS1.2.1- PRE

RFS v1.2.1

Pre-programming Guide

07-APR -2006 , Version 1.5

SAMSUNG ELECTRONICS

Copyright notice

Trademarks

RFS is a trademark of Samsung Electronics Co., Ltd in Korea and other countries.

Restrictions on Use and Transfer

All software and documents of **RFS** are commercial. Therefore, you must install, use, redistribute, modify and purchase only in accordance with the terms of the license agreement you entered into with Samsung Electronics Co., Ltd.

All advertising materials mentioning features or use of this software must display the following acknowledgement:
"This product includes **RFS** written by Samsung Electronics Co., Ltd."

Contact Information

Flash Software Group
Memory Division
Samsung Electronics Co., Ltd

Address: BanWol-Dong, Hwasung-City
Gyeonggi-Do, Korea, 445-701

Preface

Purpose

This document is a guide for the pre-programming into NAND/OneNAND device using XSR. The purpose of this document is to help programmers of ROM writer manufacturer and PDA (or Phone) manufacturer.

Scope

This document describes the pre-programming and handles GBBM2.2 scheme. In particular, this document focuses on the initial bad block management.

Definitions and Acronyms

BMF	Block Map Field
BMI	Block Map Information
BMS	Block Map Sector
BMSG	Block Map Sector Group
ECC	Error Correction Code
GBBM2.2	Global Bad Block Management 2.2
LBMSG	Locked Block Map Sector Group
UBMSG	Unlocked Block Map Sector Group
XSR	eXtended Sector Remapper
STL	Sector Translation Layer
BML	Block Management Layer
LPCA	Locked reserved block Pool Control Area
LPCB	Locked reserved block Pool Control Block
UPCA	Unlocked reserved block Pool Control Area
UPCB	Unlocked reserved block Pool Control Block
PIA	Partition Information Area
LSN	Logical Sector Number
Page	NAND/OneNAND flash memory is partitioned into fixed-sized pages. A page is (2048 + 64) bytes of Large block NAND/ONENAND.
Sector	The file system performs read/write operations in a 512-byte unit called sector.
Main Area	NAND/OneNAND flash memory location for user data. 2048-byte area (Large block NAND/OneNAND) in a page.
Spare Area	NAND/ONENAND flash memory location for meta data. 64-byte area (Large block NAND/OneNAND) in a page.

Related Documents

SEC Semiconductor division, *NAND Flash Memory & SmartMedia Data Book*, Samsung Electronics Co., Ltd 2002

History

Version	Date	Comment	Author	Approved by
0.1	18-DEC-2003	Initial draft	Songho Yoon	
1.0	09-FEB-2004	fixed miss information of table 3	Songho Yoon	
1.1	21-SEP-2004	Spare Area Assignment	Janghwan Kim	
1.2	24-JUN-2005	Fixed miss information about value of age field in BMS	Se Wook Na	Song Ho Yoon
1.3	16-NOV-2005	Reorganization for GBBM2.1	Younwon Park	
1.4	09-DEC-2005	Reviewed	Thomas	
1.5	07-APR-2006	Reorganization for GBBM2.2	Byoung Young Ahn	

Contents

Figures

Tables

1. Introduction

PDA (or Phone) manufacturer needs ROM writer to write ROM-image to NAND/OneNAND flash memory. Samsung Electronics recommends “pre-programming” to PDA (or Phone) manufacturer and ROM writer manufacturer. The pre-programming is a method to write ROM-image to NAND/OneNAND flash memory of Samsung Electronics by using GBBM2.2 scheme.

Figure 1 illustrates the relationship among companies; NAND flash memory supplier (Samsung Electronics), PDA (or Phone) manufacturer and ROM writer manufacturer.

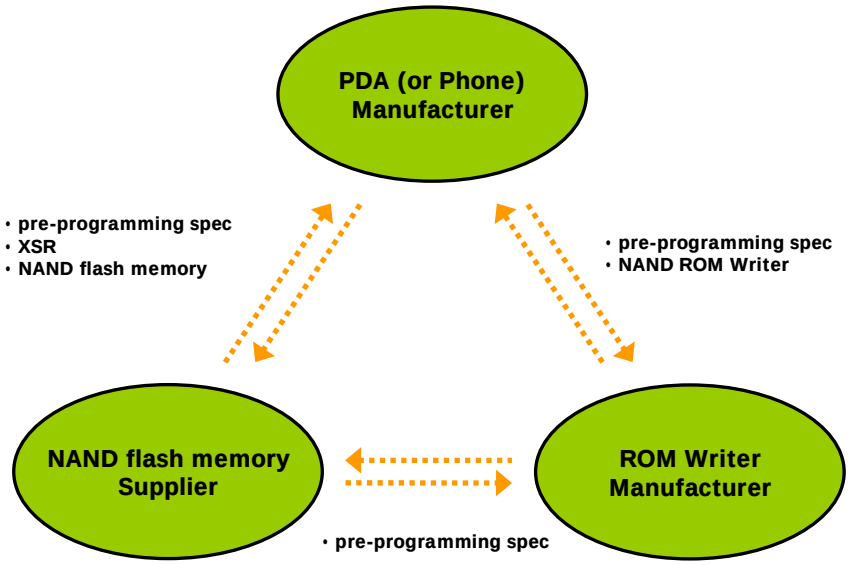


Figure 1. Relationship among companies

2. XSR

This chapter describes XSR features and architecture.

2.1. XSR

XSR is an ultimate software solution for using Samsung NAND/OneNAND flash memory. XSR is the acronym for eXtended Sector Remapper. Samsung Electronics provides XSR to PDA (or Phone) manufacturer.
XSR has STL (Sector Translation Layer) and BML (Block Management Layer) as shown in Figure 2. In XSR, BML supports GBBM2.2 scheme.GBBM2.2 scheme will be explained in next chapter. ROM writer writes ROM Image into NAND/OneNAND device by using GBBM2.2 scheme. BML mounts this ROM Image which is pre-programmed by ROM writer.

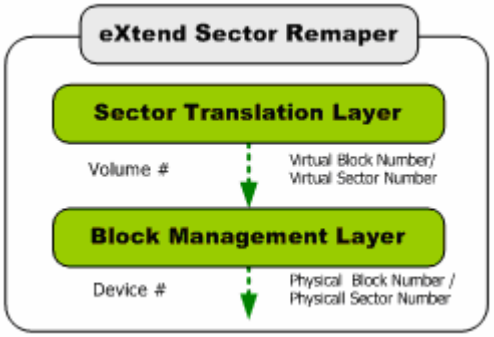


Figure 2. XSR Architecture

BML receives virtual flash address and volume number as an input parameter.

Figure 3 shows the mapping information from virtual flash address to physical flash address. A volume is a single virtual device that consists of same multiple devices. Each volume can have the different type of NAND/OneNAND devices. BML maps virtual flash address to physical flash address in order to treat multiple NAND/OneNAND devices as a single virtual device. ROM writer manufacturer should know the relationship between virtual flash address and physical flash address because ROM writer should do what BML does.

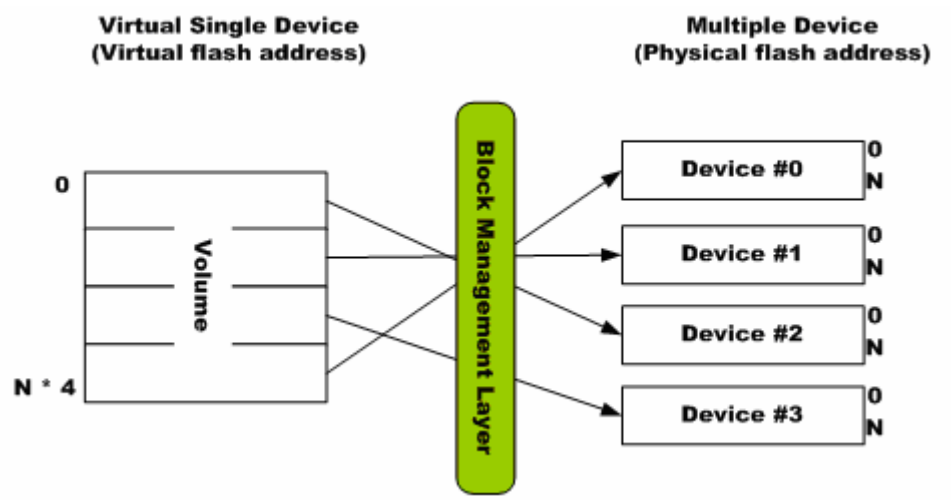


Figure 3. Virtual flash address to physical flash address

Figure 4 shows the mapping information from virtual flash address to physical flash address when the volume has 1 NAND/OneNAND device. Volume can have 1 NAND/OneNAND device or 2 NAND/OneNAND devices or 3 NAND/OneNAND devices or 4 NAND/OneNAND devices.

Note

This document assumes that volume has 1 NAND/OneNAND device.

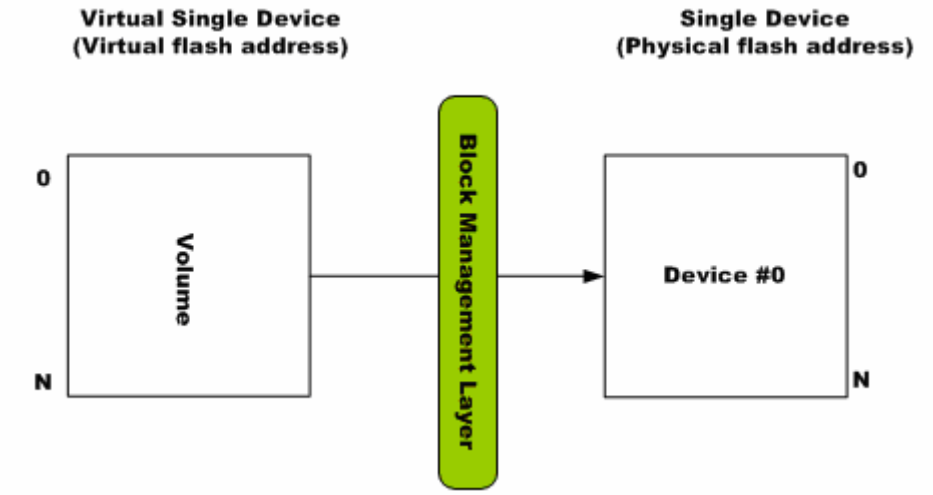


Figure 4. Virtual flash address to physical flash address in case that volume has 1 NAND/OneNAND device

```
Virtual Sector #        = Virtual address / 512
Virtual Block #         = Virtual Sector # /
                          # of Sectors in block
Virtual Sector offset # = Virtual Sector # %
                          # of Sectors in block
Virtual Sector #        = Virtual Block # *
                          # of Sectors in block +
                          Virtual Sector offset #
```

Figure 5. Formula for virtual sector # and virtual block #

3. GBBM2.2 Scheme

GBBM2.2 is the scheme for managing bad blocks in a volume. BML uses GBBM2.2 scheme for managing bad blocks in a volume.

Internally, BML gets the physical device number using virtual flash address and the number of devices in a volume. Then, BML calculates semi-physical flash address of the physical device and translates semi-physical flash address to physical flash address. Because this document assumes that volume has 1 NAND/OneNAND device, virtual flash address is same as semi-physical flash address.

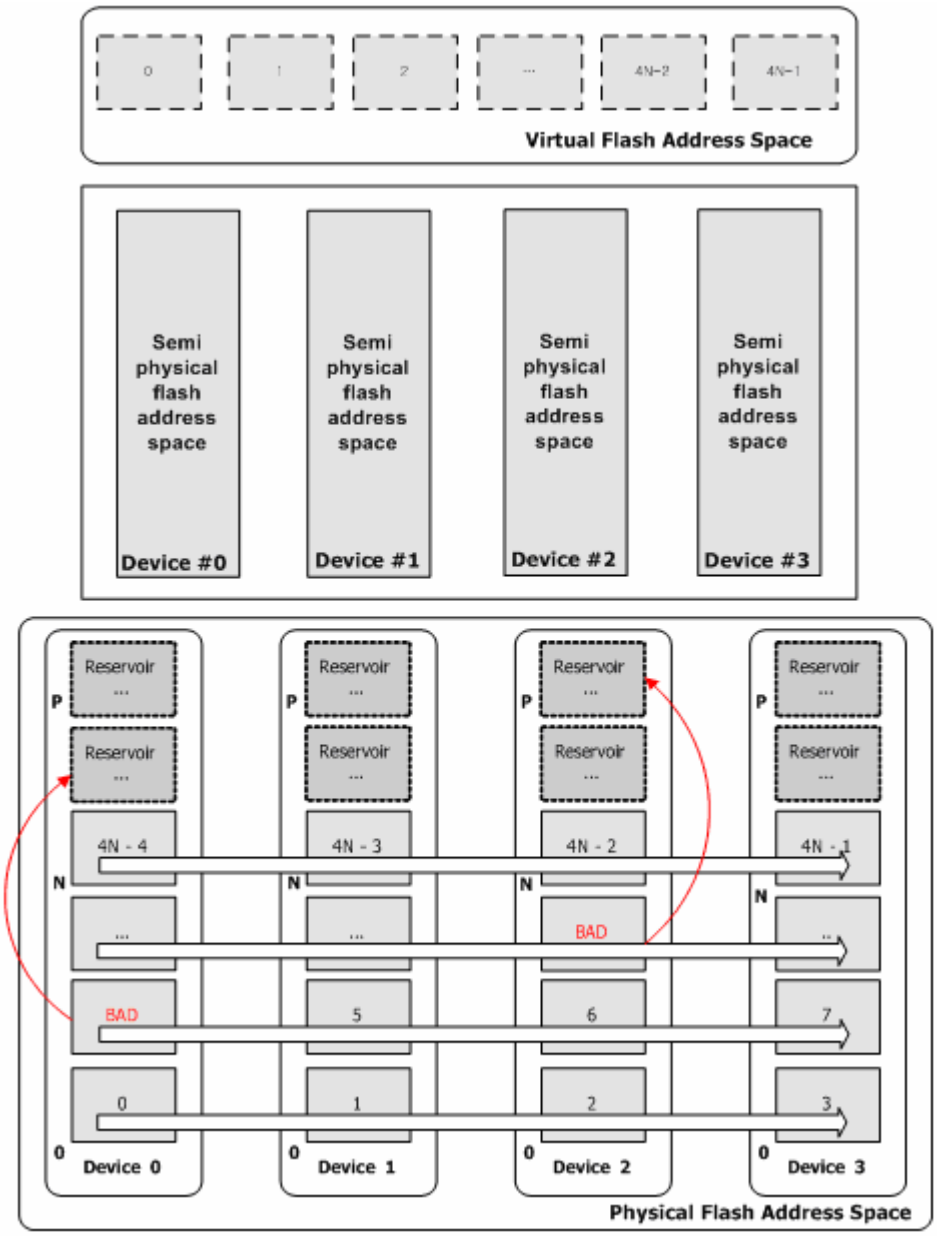


Figure 6. Address translation

If there are no bad blocks, semi-physical block number is equal to physical block number. However, if the accessed semi-physical block is a bad block, semi-physical block number will not be same as physical block number because BML remaps bad block to valid block.

Note

Byte Order of each field in meta information should follow the little endian format.

Note

ROM writer must generate ECC value using Samsung Standard ECC algorithm basically. However, if ECC algorithm which is applied by NAND controller or OneNAND is not compatible with Samsung Standard ECC algorithm, ROM writer must use ECC algorithm of NAND controller to generate ECC value. Then, ROM writer must write ECC with data at the Samsung Standard ECC location in pre-programming.
Refer to Samsung NAND Flash Spare Area Assignment Standard document for more detail information about ECC algorithm and ECC location.

3.1. Bad Blocks

Global Bad Block Management 2.2 (GBBM2.2) scheme manages two kinds of bad blocks of NAND/OneNAND flash memory.

Initial bad block
: An initial bad block is included when first shipped. This block must not be erased. For information about method to check whether a block is initial bad block or not, refer to NAND/OneNAND flash memory specification of each device.

Run-time bad block
: If a program-error(or erase error) is detected during writing (or erasing) NAND/OneNAND flash memory, the block to be written (or erased) becomes the run-time bad block.

Note

The pre-programming uses GBBM2.2 scheme for managing ONLY an initial bad block. However, the run-time bad block is not managed by the pre-programming.

3.2. Reservoir

The whole area of NAND/OneNAND device is divided into two parts by GBBM2.2 scheme; Reservoir and Partitions like Figure 7. Reservoir is a special area for managing bad blocks.

The whole area of NAND/OneNAND device is also divided into two parts by Write Protection Scheme; Unlocked Area and Locked Area as shown in Figure 7. Write Protection Scheme provides the method to protect data in to NAND/OneNAND device by NAND/OneNAND device or NAND controller. Locked Area is the area that must be protected from system crash or hacking after booting up. Locked Area can not be written and erased after being booted up. Unlocked Area is the area that can be written and erased after being booted up.

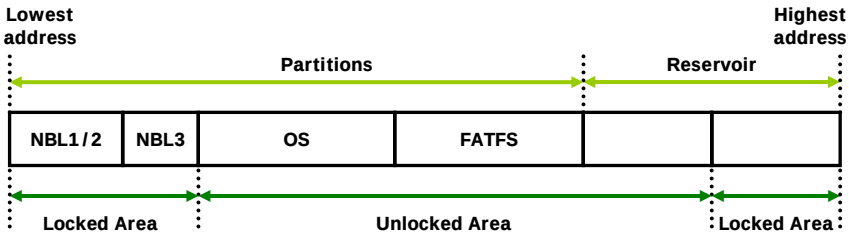


Figure 7. Organization of NAND/OneNAND device by GBBM2.2 Scheme and Lock Scheme

Reservoir contains 6 meta blocks and several reserved blocks, which is Reserved Block Pool. 6 meta blocks consist of ERL block, REF block, LPCBs and UPCBs. Reserved Block Pool is the reserved area to replace the bad blocks in Partitions. Organization of Reservoir is shown at Figure 8.

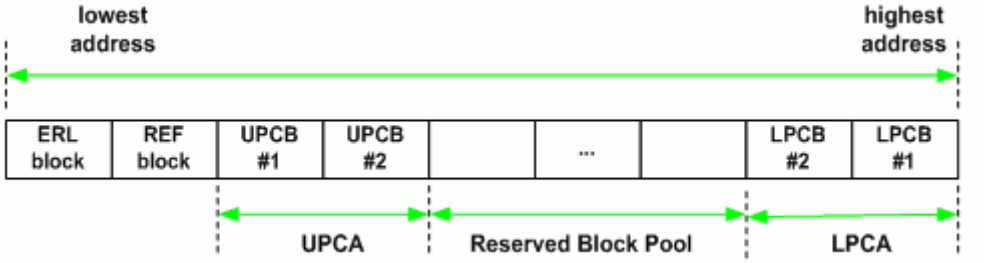


Figure 8. Organization of Reservoir

Components of Reservoir are shown in Table 1.

Table 1. Components of Reservoir

Components of Reservoir	Description
Reserved Block Pool	Reserved Block Pool is the pool of reserved blocks. The bad block in Partitions is replaced by the block in Reserved Block Pool.The number of reserved block is specified in NAND/OneNAND flash memory specification. (the number of maximum valid blocks - the number of minimum valid blocks).
ERL block	ERL is the acronym for Erase-Refreshing List. ERL block is used for Erase-Refreshing scheme. It should be located first block in reservoir and should be empty.
REF block	REF block means refreshing block. It is used for Erase-Refreshing scheme. It should be located second block in reservoir and should be empty.
UPCB	UPCB is the acronym for Unlocked reserved block Pool Control Block. It has PCH and BMSG. UPCB #1 should be written in pre-programming
UPCA	UPCA is the acronym for Unlocked Pool Control Area. To avoid the loss of meta information from sudden power-off, two copies of UPCB can be maintained in UPCA (UPCB #1 and UPCB #2).
LPCB	LPCB is the acronym for Locked reserved block Pool Control Block. It has PCH, PIA and BMSG. LPCB #1

	should be written in pre-programming.
LPCA	LPCA is the acronym for Locked Pool Control Area. To avoid the loss of meta information from sudden power-off during repartition, two copies of LPCB can be maintained in LPCA(LPCB #1 and LPCB #2)

The structure of LPCB is same as the structure of UPCB basically. Figure 9 shows the structure of PCB. PCB contains PCH (reserved block Pool Control Header), PIA (Partition Information Area) and BMSG (Block Map Sector Group). All components of PCB has the its mirror data in order to confirm the valid meta information. PCH' is the mirror of PCH. PIA' is the mirror of PIA. BMS #q' is the mirror of BMS #q.

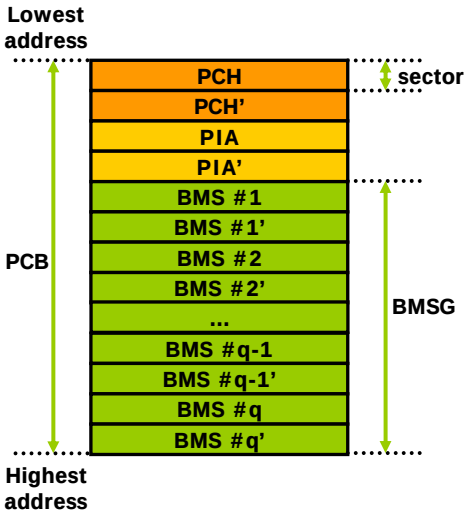


Figure 9. PCB structure

Components of PCB are shown at Table 2.

Table 2. Components of PCB

Components of Reservoir	Description
PCH	PCH is the acronym for reserved block Pool Control Header. It contains meta information for managing PCB. PCH should be written during pre-programming.
PIA	PIA is the acronym for Partition Information Area and contains partition information about Partitions and partition information extension. PIA of LPCB has partition information. And PIA of UPCB has partition information extension. PIA of LPCB should be written in pre-programming
BMS	BMS is the acronym for Block Map Sector. BMS contains the mapping information from bad block of Partitions to reserved block of Reserved Block Pool. BMS #1, BMS #1' ,BMS #2 and BMS #2' should be written in pre-programming
BMSG	BMSG is the acronym for Block Map Sector Group. BMSG is the group of several BMSs (Block Map Sector).

Reservoir can be divided into two parts by Write Protection Scheme: 1st Reservoir and 2nd Reservoir.

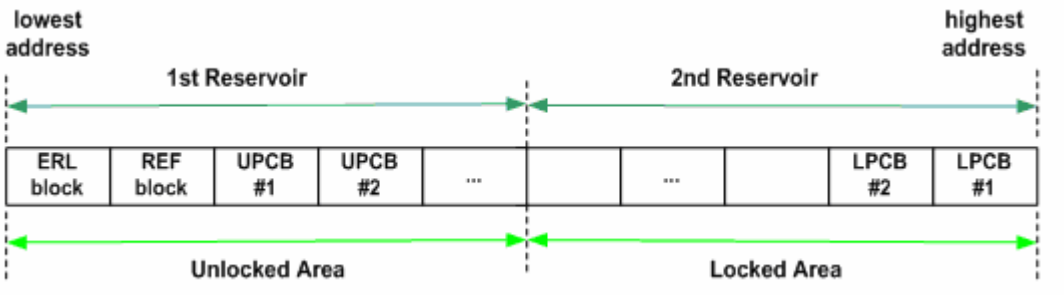


Figure 10. 1st Reservoir and 2nd Reservoir

1st Reservoir is the area for managing bad blocks of Unlocked Area in Partitions. It has ERL block, REF block, UPCA and the blocks that replace bad blocks in Unlocked Area of Partitions. Two copies of UPCB are maintained in UPCA: UPCB #1 and UPCB #2.

2nd Reservoir is the area for managing bad blocks of Locked Area in Partitions. It has the LPCA and the blocks that replace bad blocks in Locked Area of Partitions. . Two copies of LPCB is maintained in LPCA: LPCB #1 and LPCB #2.

3.2.1. PCH

PCH is used to manage the PCB. PCH is located in 1st sector and 2nd sector of PCB. 2nd sector is the mirror of 1st sector. And data of 1st sector is same as data of 2nd sector. The following table shows the fields of PCH. In pre-programming, PCH of LPCB #1 and UPCB #1 should be written.

Table 3. The fields of PCH

Field name	Offset in page (bytes)	Size (bytes)	Description
aSig	0x000	8	It is the signature of PCB. It contains 'LOCKPCHD' in LPCB. And It contains 'ULOCKPCH' in UPCB.
nAge	0x008	2	It is the age of PCB. In pre-programming, it should be set to 1.
nAltPCB	0x00A	2	It is the block number of the alternative PCB. It contains the block number of LPCB. And it contains the block number of the alternative PCB in UPCB.
nEraseSig	0x00C	8	It is the erase signature. In pre-programming. It should contain all zero.
Reserved	0x014	492	reserved field

PCH has confirmation mark in spare area. In pre-programming, the value of PCH confirmation mark is `0xFE`. The size of PCH confirmation mark is 1 byte. The position of PCH confirmation mark is as follow.

Note

Position of PCH confirmation mark = start of LSN position in spare area + 3

Refer to Appendix of this paper for more detailed information about LSN position in spare area.

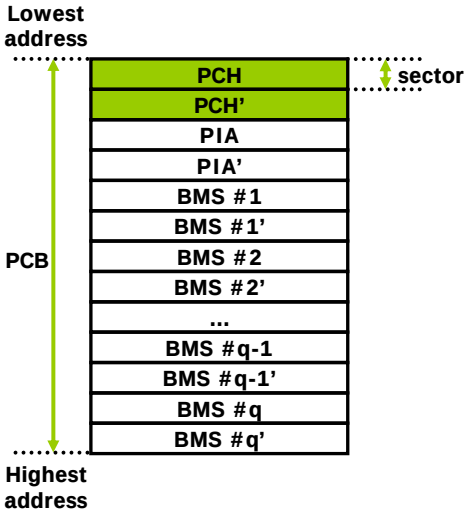


Figure 11. PCH location in PCB

PCH location in PCB is shown at Figure 11.

3.2.2. PIA

PIA is the acronym for Partition Information Area. PIA contains Partition Information or Partition Information Extension. PIA of LPCB contains Partition Information and PIA of UPCB contains Partition Information Extension. In pre-programming, PIA of LPCB should be written. PIA of UPCB will be written run-time by BML API. The fields of Partition Information are shown in Table 4.

Table 4. Fields of Partition Information

Field name	Offset in page (bytes)	Size (bytes)	Description
aSig	0x000	8	It is the signature of Partition Information. It contains 'XSRPARTI'.
nVer	0x008	4	It is the version of Partition Information. It contains 0x00011000.
nNumOfPartEntry	0x00C	4	It is the number of partition entry. The maximum number of nNumOfPartEntry is 31.
stPEntry	0x010 + 16 * nth Partition Entry #	16	It is the data structure for partition entry

The fields of Partition Entry are shown at Table 5.

Table 5. Fields of Partition Entry

Field Name	Offset in page (bytes)	Size (bytes)	Description
nID	0x000	4	It is Partition ID
nAttr	0x004	4	It is the attribute of Partition
n1stVbn	0x008	4	It is the first virtual block number of Partition
nNumOfBlks	0x00C	4	It is the number of blocks in Partitions

XSR has the pre-defined Partition ID. Table 6 shows the pre-defined ID.

Table 6. Pre-defined value of Partition ID

Pre-defined ID	Value	Description
PARTITION_ID_NBL1	0x00000000	NAND boot loader stage 1
PARTITION_ID_NBL2	0x00000001	NAND boot loader stage 2
PARTITION_ID_NBL3	0x00000002	NAND boot loader stage 3
PARTITION_ID_COPIEDOS	0x00000003	OS image copied from NAND/OneNAND device to RAM
PARTITION_ID_DEMANDONOS	0x00000004	OS image loaded on demand
PARTITION_ID_FILESYSTEM	0x00000008	file system
PARTITION_ID_FILESYSTEM1	0x00000009	file system

Partition ID from `0x00000000` to `0x0FFFFFFF` is reserved in XSR. And a user can define the own Partition ID from `0x10000000`.

Partition also has the 3 kinds of attribute: FROZEN_RO, RO and RW. Table 7 shows Partition Attributes.

Table 7. Partition Attributes

Attribute ID	Value	Description
FROZEN_RO	0x00000022	It is Read-Only attribute basically. And this partition can be lock-tightened area by hardware (NAND/OneNAND device or NAND controller). Thus, this attribute can't be changed in normal mode.
RO	0x00000002	It is Read-Only attribute. This attribute can be changed by software (BML API).
RW	0x00000001	It is Read-Write attribute. This attribute can be changed by software (BML API).

Note

A contiguous area from block #0 only can be the Frozen-RO partition. And only FROZEN_RO can be allocated as a LOCKED area.

Example of Partition and its attribute is shown at Figure 12. In example, NBL partition has the Frozen-RO attribute. OS partition has RO attribute. FATFS partition has RW attribute.
If volume has one device, the Frozen-RO partition becomes Locked Area and the remaining partitions become Unlocked Area.

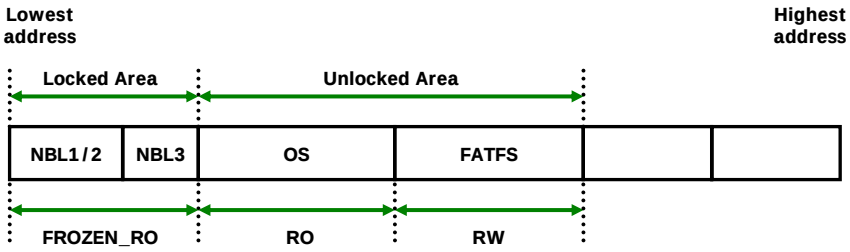


Figure 12. Example of Partitions and it's Attributes

PIA location in PCB is shown in Figure 13. PIA is located in 3rd sector and 4th sector of PCB. 4th sector is the mirror of 3rd sector. And data of 3rd sector is equal to data of 4th sector.

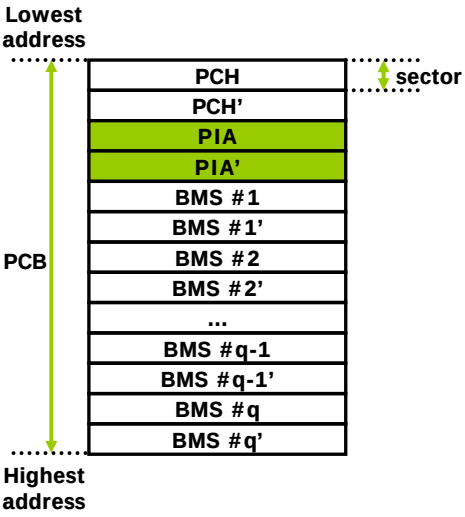


Figure 13. PIA location in PCB

PIA has confirmation mark in spare area. In pre-programming, the value of PIA confirmation mark is `0xFE`. The size of PIA confirmation mark is 1 byte. The position of PIA confirmation mark is as follow.

Note

Position of PIA confirmation mark = start of LSN position in spare area + 3

Refer to Appendix section for more detail information about LSN position in spare area.

3.2.3. BMS

BMS is the acronym for Block Map Sector. BMS contains the mapping information from bad block of Partitions to reserved block of Reserved Block Pool. ***Two copies of BMS are used to store the mapping information and each of the BMS has its mirror data.***

A BMS can be either LBMS or UBMS. LBMS is located in LPCA. UBMS is located in UPCA. The structure of LBMS is same as that of UBMS basically.

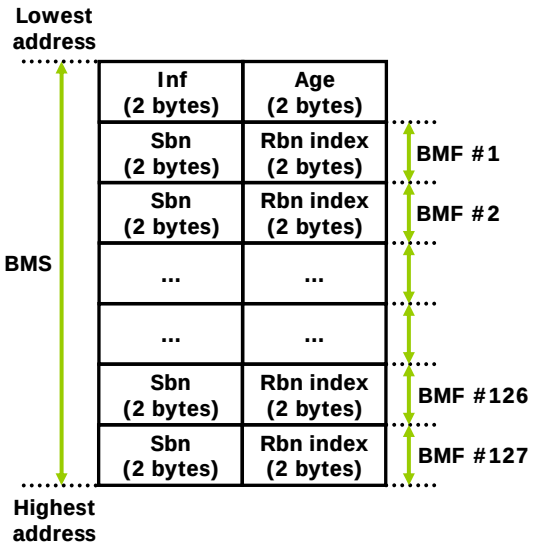


Figure 14. BMS Structure

BMS structure is shown in Figure 14. BMS can be divided into 3 parts: BMS Information Field, BMS Age and Block Map Fields. BMS can have 127 Block Map Fields. Block Map Field represents the mapping information from bad block number in Partitions to replaced block index.

Sbn (Semi-physical block number) is the bad block number in Partitions. ***Rbn*** is the replaced block index and is an offset from 1st block number in Reservoir.

Note

Replaced Block Index = Reserved Block # - 1st Block # of Reservoir

Inf field shows a cause as to why BMS is created. ***Inf*** field of BMS should be set to ***0xFCFE*** in pre-programming. And Age field of BMS should be set to ***0x0000*** in pre-programming.

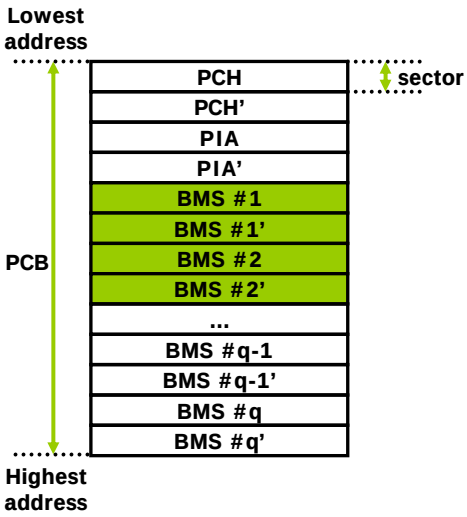


Figure 15. BMS location in PCB

BMS position in PCB is shown in Figure 15. In pre-programming, BMS #1, BMS #1', BMS #2 and

BMS #2' should be written. BMS is allocated from 5th sector to the end of sector in block.

BMS has confirmation mark in spare area. In pre-programming, the value of BMS confirmation mark is `0xFE`. The size of BMS confirmation mark is 1 byte. The position of BMS confirmation mark is as follow.

Note

Position of BMS confirmation mark = start of LSN position in spare area + 3

Refer to Appendix section for more detail information about LSN position in spare area.

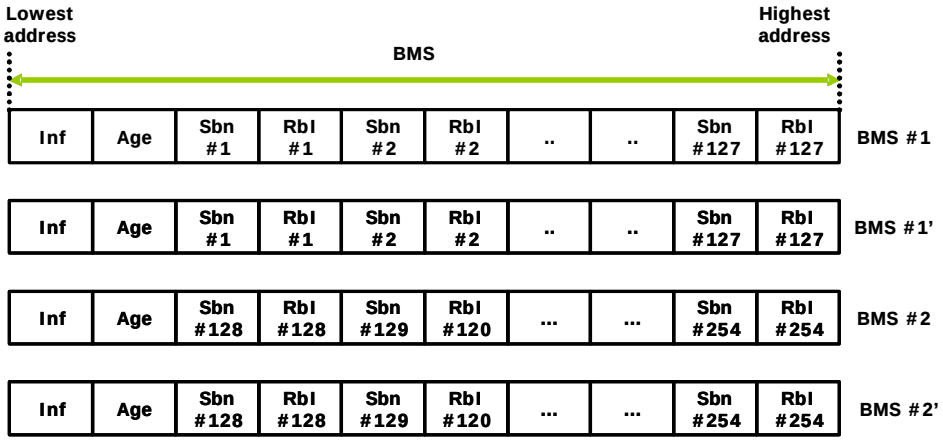


Figure 16. Numbering order of BMF in each BMS

In order to store mapping information, two copies of BMS is used. Thus, total number of mapping information in Reservoir is `762`. And each BMS has its mirror. Thus, In pre-programming, four copies (or eight or twelve copies) of BMS should be written according to the number of initial bad blocks. If the number of initial bad blocks is more than 254, eight copies of BMS should be written. And if the number of initial bad blocks is more than 508, twelve copies of BMS should be written.

3.2.4. UPCA

UPCA has two copies of UPCB. The description of UPCB #1 and UPCB #2 is shown in Table 8. First two blocks are reserved for Erase-refreshing scheme. After the two blocks, by scanning Reservoir from the lowest address to the highest address, UPCB #1 is allocated in the first valid block (not initial bad block) and UPCB #2 is allocated in the second valid block (not initial bad block). If two valid blocks are found, the scanning will be finished.

Table 8. UPCB #1 and UPCB #2

UPCA	Description
UPCB #1	BMSG of this block contains the mapping information for both initial

	bad blocks of Unlocked Area in Partitions and reserved blocks in 1st Reservoir. PIA of this block contains the partition information extension. Partition information extension is not used in pre-programming. UPCB #1 is used to manage initial bad blocks in pre-programming.
UPCB #2	UPCB #2 is not used in pre-programming. But, UPCB #2 will be used in XSR.

UPCA Position is shown in Figure 17.

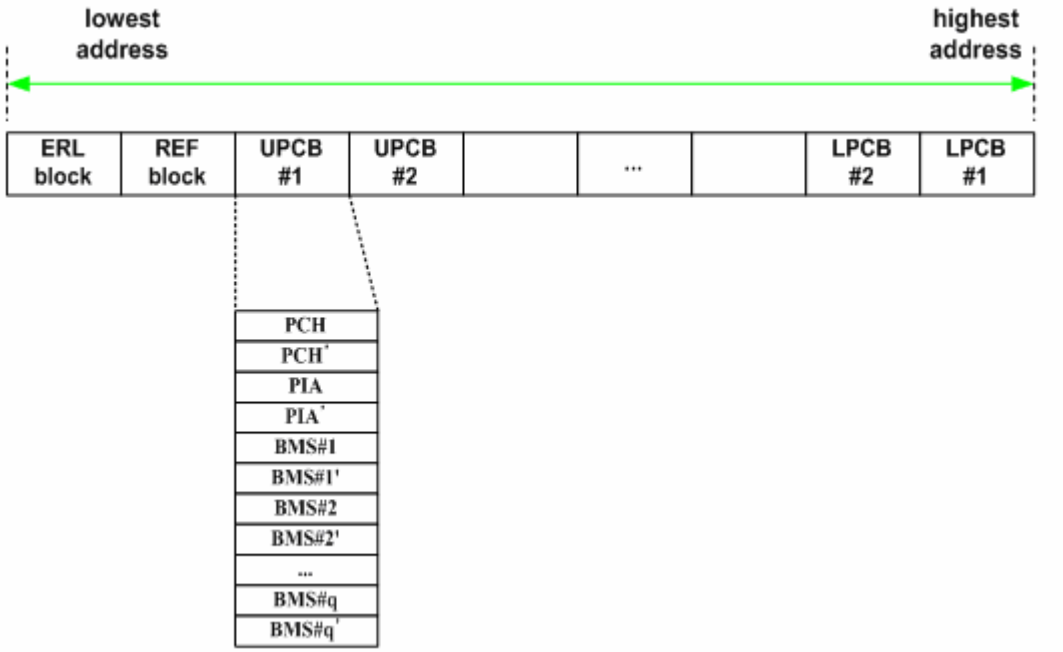


Figure 17. UPCA location

3.2.5. LPCA

LPCA has two copies of LPCB.. By scanning Reservoir from the highest address to the lowest address, LPCB #1 is allocated in the first valid block (not initial bad block) and LPCB #2 is allocated in the second valid block (not initial bad block).. If valid block is found, the scanning will be finished.

LPCB #1 is an area to contain the mapping information for both bad blocks of Locked Area in Partitions and reserved blocks in 2nd Reservoir. LPCB #1 is used to manage initial bad blocks in pre-programming.

Table 9. LPCB #1 and LPCB #2

LPCA	Description
LPCB #1	BMSG of this block contains the mapping information for both initial bad blocks of Locked Area in Partitions and reserved blocks in 2nd Reservoir. PIA of this block contains the partition information. Partition information should be written in pre-programming.

	LPCB #1 is used to manage initial bad blocks in pre-programming.
LPCB #2	LPCB #2 is not used in pre-programming. But, LPCB #2 will be used in XSR.

LPCA position is shown in Figure 18.

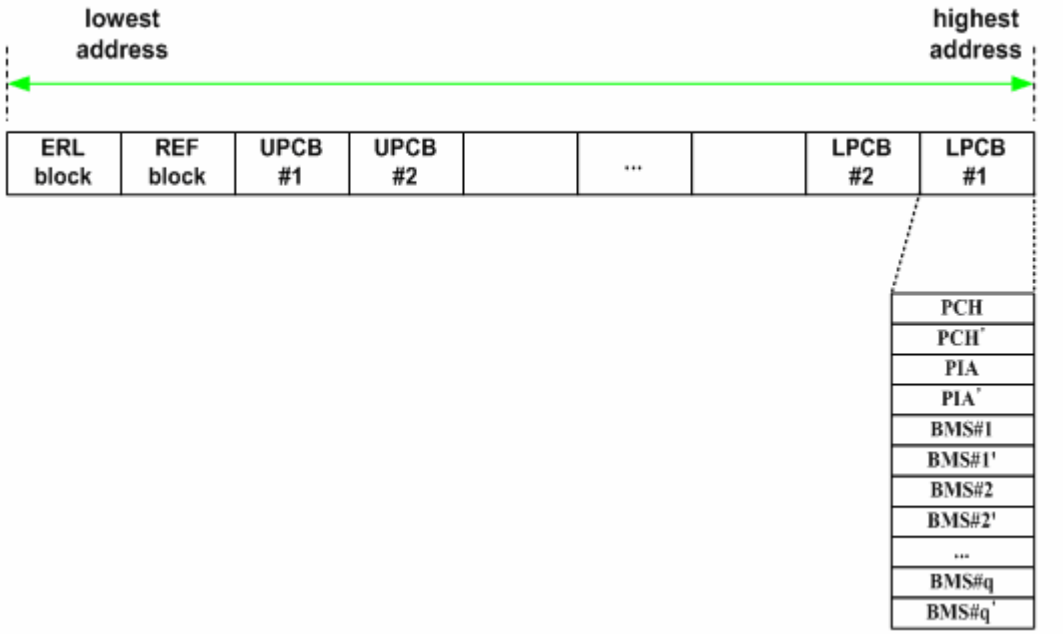


Figure 18. LPCA position

3.3. Format Reservoir

Formatting Reservoir is to make meta information for managing Reservoir and to write meta information into each PCB. Formatting Reservoir has the following steps.

Reservoir initialization
Make-up BMS
Make-up PCH
Write PCH, PI and BMS into PCB

3.3.1. Reservoir Initialization

Reservoir Initialization is to search for the block numbers, which are allocated for UPCB #1 and LPCB #1, LPCB #2 in Reservoir. UPCB #1, UPCB #2 and LPCB #1, LPCB #2 should be allocated to valid block (not initial bad block).
Following pseudo-code is an example for Reservoir Initialization. In pseudo-code, ***nNumOfMetaBlkInReservoir*** is the fixed value 6.

```
Uint16 nUPCB[2];
Uint16 nLPCB[2];

Uint16 nAllocLPtr;
Uint16 nAllocUPtr;
Uint16 n1stPbnOfReservoir;
```

```
ReservoirInitialization(
   Uint16 nNumOfTotalBlk,
   Uint16 nNumOfBlkInReservedBlkPool,
   Uint16 nNumOfMetaBlkInReservoir)
{
    n1stPbnOfReservoir = nNumOfTotalBlk – \
                         nNumOfBlkInReservedBlkPool - \
                         nNumOfMetaBlkInReservoir;

    nAllocLPtr = nNumOfTotalBlk;
    nAllocUPtr = n1stPbnOfReservoir – 1 + 2;

    nUPCB[0] = GetValidBlk(scan from lowest address to highest);

    nUPCB[1] = GetValidBlk(scan from lowest address to highest);

    nLPCB[0] = GetValidBlk(scan from highest address to lowest);

    nLPCB[1] = GetValidBlk(scan from highest address to lowest);


    if ((nUPCB[0] == “not valid block”) ||
        (nUPCB[1] == “not valid block”) ||
        (nLPCB[0] == “not valid block”) ||
        (nLPCB[1] == “not valid block”))
    {
       reservoir initialization is failed
    }
}

uint16 GetValidBlk(uint16 nDir)
{
    uint16 nBlkNum;

    if (nDir == scan from lowest to highest address)
    {
      while (1)
      {
        nBlkNum = nAllocLPtr - 1;
        if (nBlkNum <= nAllocUPtr)
        {
            break;
        }
        if (nBlkNum != initial bad block)
        {
            nAllocLPtr--;
            return nBlkNum;
        }
      }
    }
    else
    {
      while (1)
      {
        nBlkNum = nAllocUPtr + 1;
```

```
        if (nBlkNum >= nAllocLPtr)
        {
            break;
        }
        if (nBlkNum != initial bad block)
        {
            nAllocUPtr++;
            return nBlkNum;
        }
      }
    }

    return “not valid block”
}
```

3.3.2. Make-up BMS

After Reservoir Initialization, BMS should be created by GBBM2.2 scheme. In order to make BMS, ROM writer should search for initial bad blocks in Partitions. ROM writer scans Locked Area and Unlocked Area in Partitions from block #0 to the end block in Partitions to search for initial bad blocks.

If an initial bad block is detected during scanning Locked Area in Partitions, ROM writer performs following steps:

It searches for a free reserved block in Reserved Block Pool from the highest address to the lowest address by top-down fashion.
It searches for a free BMF entry in LBMS (Locked BMS) of Locked Area from entry #1 to entry #762.
It stores the initial bad block number into `nSbn` of the free BMF.
It stores the free reserved block number – *1st Pbn of Reservoir* into `nRbI` of the free BMF.

If an initial bad block is detected during scanning Unlocked Area in Partitions, ROM writer performs following steps:

It searches for a free reserved block in Reserved Block Pool from lowest address to highest address by bottom-up fashion.
It searches for a free BMF in UBMS (Unlocked BMS) Unlocked Area from entry #1 to entry #762.
It stores the initial bad block number into `nSbn` of the free BMF.
It stores the free reserved block number – *1st Pbn of Reservoir* into `nRbI` of the free BMF.

Following pseudo code is an example of making-up BMS.

```
#define  BMF_PER_BMS   127
#define  EMPTY_BLK     0xFFFF

Typedef struct {
    Uint16 nSbn;
    Uint16 nRbI;
} BMF;

typedef struct {
    uint16 nInf;
    uint16 nAge;
    BMF    stBMF[MAX_BMF];
```

```
} BMS;

BMS stLBMS[6];
BMS stUBMS[6];

Void MakeUpLBMS()
{
   uint16 nBMFIdx;
   uint16 nBMSIdx;
   uint16 nBlkNum;
   uint16 nRsvPbn;

   memset(&stLBMS[0], 0xFF, sizeof(BMS));
   memset(&stLBMS[1], 0xFF, sizeof(BMS));
   memset(&stLBMS[2], 0xFF, sizeof(BMS));
   memset(&stLBMS[3], 0xFF, sizeof(BMS));
   memset(&stLBMS[4], 0xFF, sizeof(BMS));
   memset(&stLBMS[5], 0xFF, sizeof(BMS));

   stLBMS[0].nInf = 0xFCFE;
   stLBMS[0].nAge = 0x0001;
   stLBMS[1].nInf = 0xFCFE;
   stLBMS[1].nAge = 0x0001;

   nBMSIdx = 0;
   nBMFIdx = 0;

   for (nBlkNum = 0; nBlkNum <= number of blocks in Locked Area;
        nBlkNum ++)
   {
      if (nBlkNum != initial bad block)
      {
         continue;
      }

      // Get a free reserved block in 2nd Reservoir
      nRsvPbn = GetVaidBlk(scan from highest address to lowest);

      stLBMS[nBMSIdx].stBMF[nBMFIdx].nSbn = nBlkNum;
      stLBMS[nBMSIdx].stBMF[nBMFIdx].nRbI = \
                     nRsvPbn - n1stPbnOfReservoir;

      if (++nBMFIdx >= BMF_PER_BMS)
      {
         nBMFIdx = 0;
         nBMSIdx++;
         if (nBMSIdx == 2)
         {
            stLBMS[2].nInf = 0xFCFE;
            StLBMS[2].nAge = 0x0001;
            stLBMS[3].nInf = 0xFCFE;
            stLBMS[3].nAge = 0x0001;
         }
         If (nBMSIdx == 4)
         {
            stLBMS[4].nInf = 0xFCFE;
```

```
            stLBMS[4].nAge = 0x0001;
            stLBMS[5].nInf = 0xFCFE;
            stLBMS[5].nAge = 0x0001;
        }
    }
  }
}

Void MakeUpUBMS()
{
   uint16 nBMFIdx;
   uint16 nBMSIdx;
   uint16 nBlkNum;
   uint16 nRsvPbn;

   memset(&stUBMS[0], 0xFF, sizeof(BMS));
   memset(&stUBMS[1], 0xFF, sizeof(BMS));
   memset(&stUBMS[2], 0xFF, sizeof(BMS));
   memset(&stUBMS[3], 0xFF, sizeof(BMS));
   memset(&stUBMS[4], 0xFF, sizeof(BMS));
   memset(&stUBMS[5], 0xFF, sizeof(BMS));

   stUBMS[0].nInf = 0xFCFE;
   stUBMS[0].nAge = 0x0001;
   stUBMS[1].nInf = 0xFCFE;
   stUBMS[1].nAge = 0x0001;

   nBMSIdx = 0;
   nBMFIdx = 0;

   for (nBlkNum = first Pbn of Unlocked Area;
        nBlkNum <= last Pbn of Unlocked Area;
        nBlkNum ++)
   {
      if (nBlkNum != initial bad block)
      {
         continue;
      }

      // Get a free reserved block in 2nd Reservoir
      nRsvPbn = GetValidBlk(scan from lowest address to highest);

      stLBMS[nBMSIdx].stBMF[nBMFIdx].nSbn = nBlkNum;
      stLBMS[nBMSIdx].stBMF[nBMFIdx].nRbI = \
                     nRsvPbn - n1stPbnOfReservoir;

      if (++nBMFIdx >= BMF_PER_BMS)
      {
         nBMFIdx = 0;
         nBMSIdx++;
      }
      If (nBMSIdx == 2)
      {
         stUBMS[2].nInf = 0xFCFE;
         stUBMS[2].nAge = 0x0001;
         stUBMS[3].nInf = 0xFCFE;
```

```
            stUBMS[3].nAge = 0x0001;
        }
        If (nBMSIdx == 4)
        {
            stUBMS[4].nInf = 0xFCFE;
            stUBMS[4].nAge = 0x0001;
            stUBMS[5].nInf = 0xFCFE;
            stUMBS[5].nAge = 0x0001;
        }
    }
}

Void MakeUpBMS()
{
    MakeUpLBMS();
    MakeUpUBMS();
}
```

3.3.3. Make-up PCH

After creating BMS, PCH should be made by GBBM2.2 scheme. There are 2 copies of UPCH and LPCH in Reservoir. Following pseudo code is an example of making-up PCH.

```
Typedef struct {
   Uint8  aSig[8];
   Uint16 nAge;
   Uint16 nAltPcb;
   Uint32 nEraseSig1;
   Uint32 nEraseSig2;
   Uint8  aPad[492];
} PoolCtlHdr;

PoolCtlHdr stLPCH;
PoolCtlHdr stUPCH;

Void MakeupPCH()
{
    memset(&stUPCH, 0xFF, sizeof(PoolCtlHdr));
    memset(&stLPCH, 0xFF, sizeof(PoolCtlHdr));

    memcpy(&(stUPCH.aSig), “ULOCKPCH”, 8);
    stUPCH.nAge    = 0x0001;
    stUPCH.nAltPcb = nUPCB[1];
    stUPCH.nEraseSig1 = 0x00000000;
    stUPCH.nEraseSig2 = 0x00000000;

    memcpy(&(stLPCH.aSig), “LOCKPCHD”, 8);
    stLPCH.nAge    = 0x0001;
    stLPCH.nAltPcb = nLPCB[1];
    stLPCH.nEraseSig1 = 0x00000000;
    stLPCH.nEraseSig2 = 0x00000000;
}
```

3.3.4. Write PCH, PI and BMS into PCB

After completing the creation of BMS and PCH, ROM writer should perform following steps:

Erase whole block of Reservoir except initial bad blocks.
Write ***stLPCH*** into 1st sector and 2nd sector of block #***nLPCB[0]*** with ECC and confirmation mark.
Write ***stUPCH*** into 1st sector and 2nd sector of block #***nUPCB[0]*** with ECC and confirmation mark.
Write Partition Information into 3rd sector and 4th sector of block #***nLPCB[0]*** with ECC and confirmation mark.
Write ***stLBMS[0]*** into 5th sector and 6th sector of block #***nLPCB[0]*** with ECC and confirmation mark.
Write ***stLBMS[1]*** into 7th sector and 8th sector of block #***nLPCB[0]*** with ECC and confirmation mark.
Write ***stUBMS[0]*** into 5th sector and 6th sector of block #***nUPCB[0]*** with ECC and confirmation mark.
Write ***stUBMS[1]*** into 7th sector and 8th sector of block #***nUPCB[0]*** with ECC and confirmation mark.
Additionally, others could be written according to the number of initial bad blocks.
(stLBMS[2],stLBMS[3], stUBMS[2], stUBMS[3])

3.4. Flash Operation using BMFs

After formatting Reservoir, ROM writer can perform the flash operation (read/write/erase) to Partitions using BMFs of BMS.
When ROM writer accesses any address in Partitions, this address becomes semi-physical flash address. The semi-physical flash address is divided into the block number of semi-physical flash address and offset in this block. Then, ROM writer checks whether this block number is registered among `nSbn` in BMFs of BMS. If this block number is registered in BMFs, ROM writer should access remapped address (`nRbI + n1stPbnOfReservoir` + offset). Otherwise, GBBM scheme should access the address, which is `nRbn` + offset.

Following pseudo code is an example of Flash Operation through BMFs.

```
Uint16
GetPhysicalBlkNum(uint16 nSemiPhysicalBlk)
{
    uint16 nIdx;
    uint16 nBMSIdx;
    BMF    *pBMF;

    if (nSemiPhysicalBlk is included in Locked Area)
    {
        for (nBMSIdx = 0; nBMSIdx < 6; nBMSIdx++)
        {
           pBMF = stLBMS[nBMSIdx].stBMF;

           for (nIdx = 0; nIdx < BMF_PER_BMS; nIdx++)
```

```
            {
               if (pBMF[nIdx].nSbn == nSemiPhysicalBlk)
               {
                  return pBMF[nIdx].nRbI + n1stPbnOfReservoir;
               }
               else if (pBMF[nCnt].nSbn == EMPTY_BLK)
               {
                  return nSemiPhysicalBlk;
               }
            }
         }
      }
      else if (nSemiPhysicalBlk is included in Unlocked Area)
      {
         for (nBMSIdx = 0; nBMSIdx < 6; nBMSIdx++)
         {
            pBMF = stUBMS[nBMSIdx].stBMF;
            for (nIdx = 0; nIdx < BMF_PER_BMS; nIdx++)
            {
               if (pBMF[nIdx].nSbn == nSemiPhysicalBlk)
               {
                  return pBMF[nIdx].nRbI + n1stPbnOfReservoir;
               }
               else if (pBMF[nIdx].nSbn == EMPTY_BLK)
               {
                  return nSemiPhysicalBlk;
               }
            }
         }
      }

      return nSemiPhysicalBlk;
   }

   FlashOperation(uint16 nSemiPhysicalAddr, uint8 *pBuf)
   {
      uint16 nPhysicalBlkNum;
      uint16 nPhysicalAddr;
      uint16 nSemiPhysicalBlkNum;
      uint16 nOffsetInBlk;

      nSemiPhysicalBlkNum = nSemiPhysicalAddr /
                            the size of one block;
      nOffsetInBlk = nSemiPhysicalAddr %
                            the size of one block;

      nPhysicalBlkNum = GetPhysicalBlkNum(nSemiPhysicalBlkNum);

      nPhysicalAddr = nPhysicalBlkNum * the size of one block
                    + nOffsetInBlk;

      FlashOperation into nPhysicalAddr with pBuf
   }
```

4. Pre-programming Procedure

The pre-programming is method to write ROM image into NAND/OneNAND device by GBBM2.2 scheme.

4.1. Precondition

4.1.1. Supporting NAND/OneNAND flash memory

This pre-programming guide supports Samsung large block SLC NAND devices and OneNAND devices that satisfy the following pre-conditions

The size of main array in page should be equal to 1Kbyte or 2Kbyte.
The size of spare array in page should be equal to 32 bytes or 64 bytes.
SLC type large block NAND and OneNAND devices

Note

GBBM2.2 based on XSRv1.5 supports only Samsung large block SLC NAND devices and Samsung OneNAND devices.

4.1.2. Input Parameter for ROM writer

In order to write ROM image file, the input parameter is offered to ROM writer.
The input parameters are as follows.

1. The number of blocks in whole area of one single chip
2. The number of blocks in Reservoir in one single chip
3. ROM image file
4. Partition Information

4.1.2.1. Number of Blocks

The followings are needed to get information for number of blocks.

Number of blocks in whole area of one single chip
Number of blocks in whole area of one single chip is different based on capacity of each device. Please refer to the specification of each chip for more information about the number of blocks in one single chip.
Number of blocks in Reservoir in one single chip
The number of blocks in Reservoir in one single chip is “the number of Reserved Block Pool + the number of meta blocks”. The number of Reserved Block Pool is the gap between the maximum and minimum values of valid blocks. This gap means the number of maximum available bad blocks. Please refer to the specification of each chip for more information about this gap between the maximum and minimum values of valid blocks.

Table 10 shows the sample information about the number of blocks for Reservoir per each device’s capacity of OneNAND.
For more information, refer to the Specification of each device.

Table 10. The sample information about the number of blocks for Reservoir for OneNAND

OneNAND Density	The number of blocks in one single chip	The number of blocks for Reservoir (Component Type)
128Mbit	256	5 + 6
256Mbit	512	10 + 6
512Mbit	512	10 + 6
1Gbit	1024	20 + 6
2Gbit	2048	40 + 6

4.1.2.2. ROM image file

ROM image file can contain NAND/OneNAND Boot Loader, OS image. ROM image is stored into NAND/OneNAND Boot loader area or OS Partition. ROM writer manufacturer should get ROM image file from PDA (or Phone) manufacturer to burn into the NAND/OneNAND device. Normally, real size of each image is less than the size of each partition. When making ROM image file, dummy contents will be added into ROM image file like Figure 19.

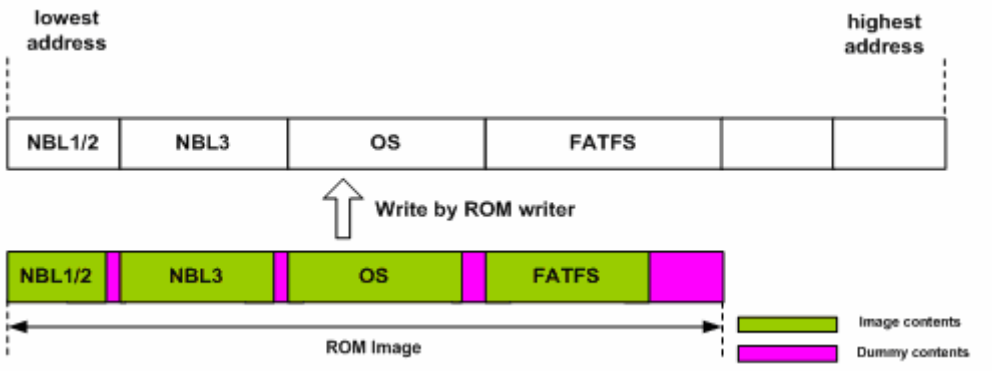


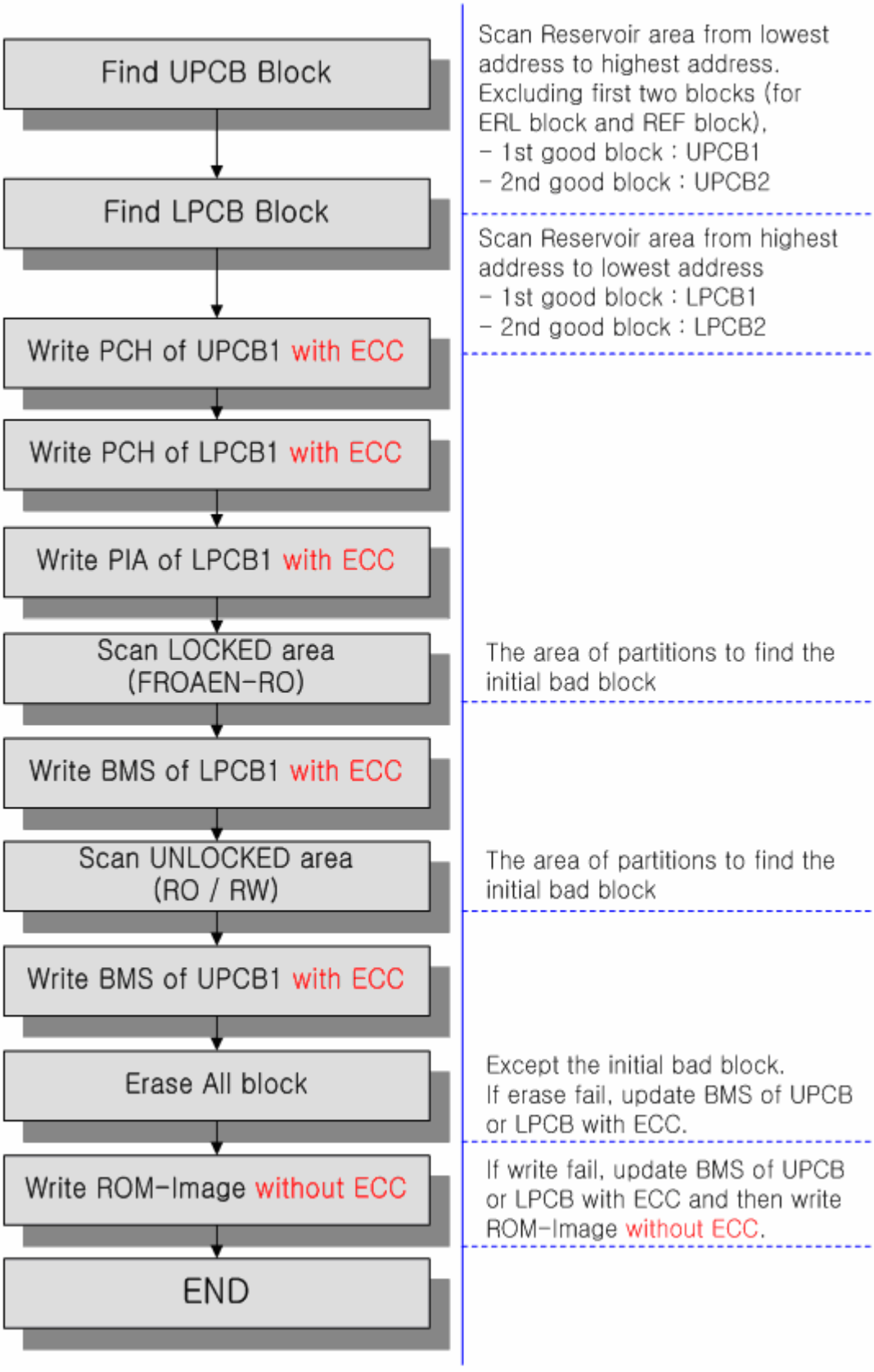
Figure 19. ROM image file

4.1.2.3. Partition Information

For detailed information about Partition Information, refer to Chapter Partition Information Area 3.2.2 of this document. ROM writer manufacturer should get Partition Information from PDA (or Phone) manufacturer.

4.2. Write ROM image file

When PDA (or Phone) manufacturer gives the input parameter of ROM writer (number of blocks in whole area of one single chip, number of blocks in Reservoir in one single chip, ROM image file, and Partition Information) into ROM writer, ROM writer should perform the following steps:


Find UPCB Block
Find LPCB Block
Write PCH of UPCB1 with ECC
Write PCH of LPCB1 with ECC
Write PIA of LPCB1 with ECC
Scan LOCKED area (FROAEN-RO)
Write BMS of LPCB1 with ECC
Scan UNLOCKED area (RO / RW)
Write BMS of UPCB1 with ECC
Erase All block
Write ROM-Image without ECC
END
Scan Reservoir area from lowest address to highest address. Excluding first two blocks (for ERL block and REF block),
- 1st good block : UPCB1
- 2nd good block : UPCB2
Scan Reservoir area from highest address to lowest address
- 1st good block : LPCB1
- 2nd good block : LPCB2
The area of partitions to find the initial bad block
The area of partitions to find the initial bad block
Except the initial bad block. If erase fail, update BMS of UPCB or LPCB with ECC.
If write fail, update BMS of UPCB or LPCB with ECC and then write ROM-Image without ECC.

Refer to chapter 3 GBBM2.2 scheme for more information of each step.

In order to speed up pre-programming, first ROM writer had better check whether written data is all 0xFF or not, and then, it does not need to write data if the data is all 0xFF.

The initial structure of LPCB and UPCB is shown in Figure 20

Note

ROM image can contains main area and spare area data of NAND/OneNAND flash.

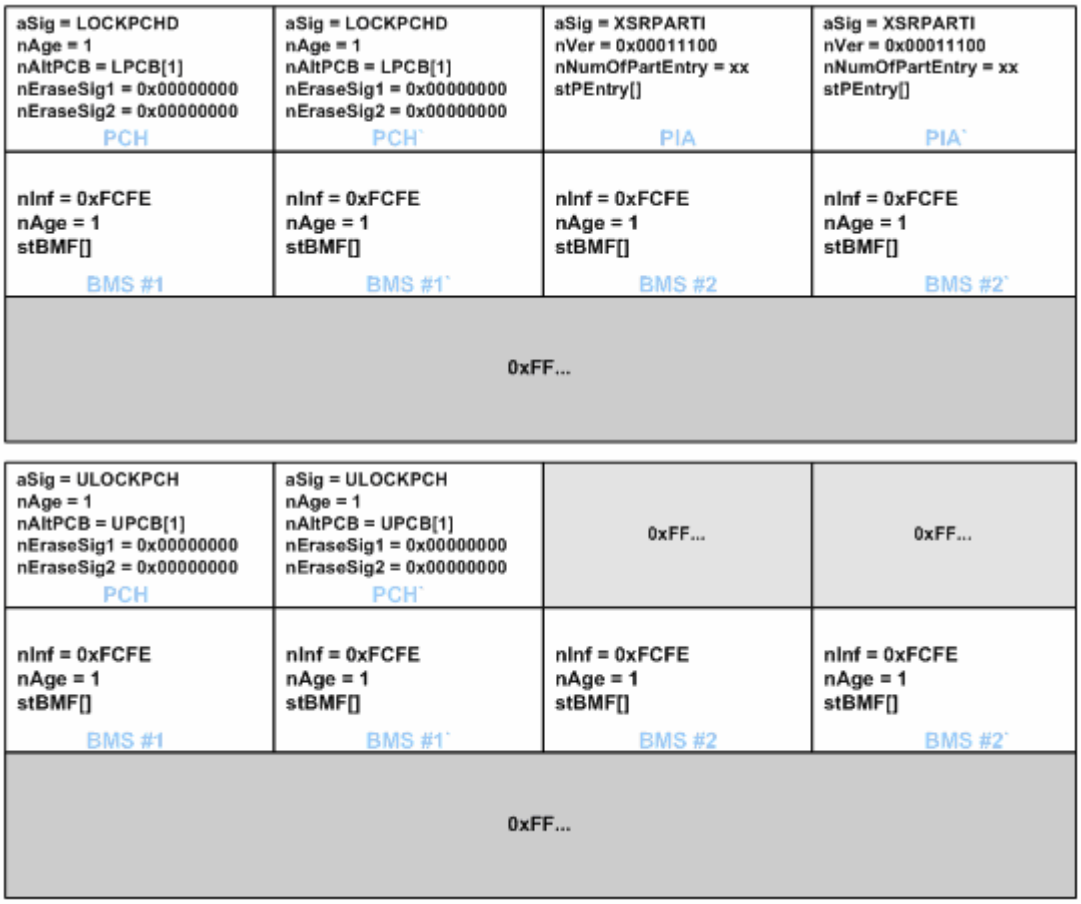


Figure 20. The initial structure of LPCB #1 and UPCB #1 after Pre-programming is completed

The format of meta data in XSR is shown in Figure 21.

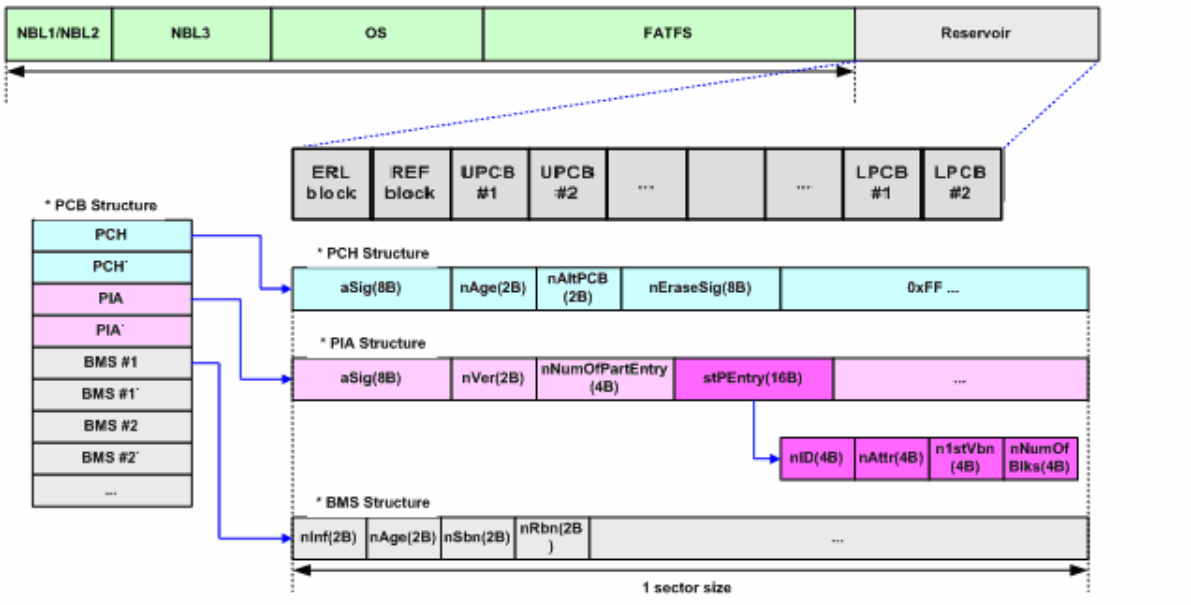


Figure 21. Format of meta data in XSR

Appendix

I. Spare Area Assignment

BML XSR uses some meta data of spare area at BML format and open phase. As mentioned above, meta data of reservoir uses confirmation mark of spare area. However, Spare area organization is variable for every NAND/OneNAND flash type. So BML uses LSN position as a basement position. BML always set the position of conformation mark as LSN + 3. The following figure shows how to assign spare area.

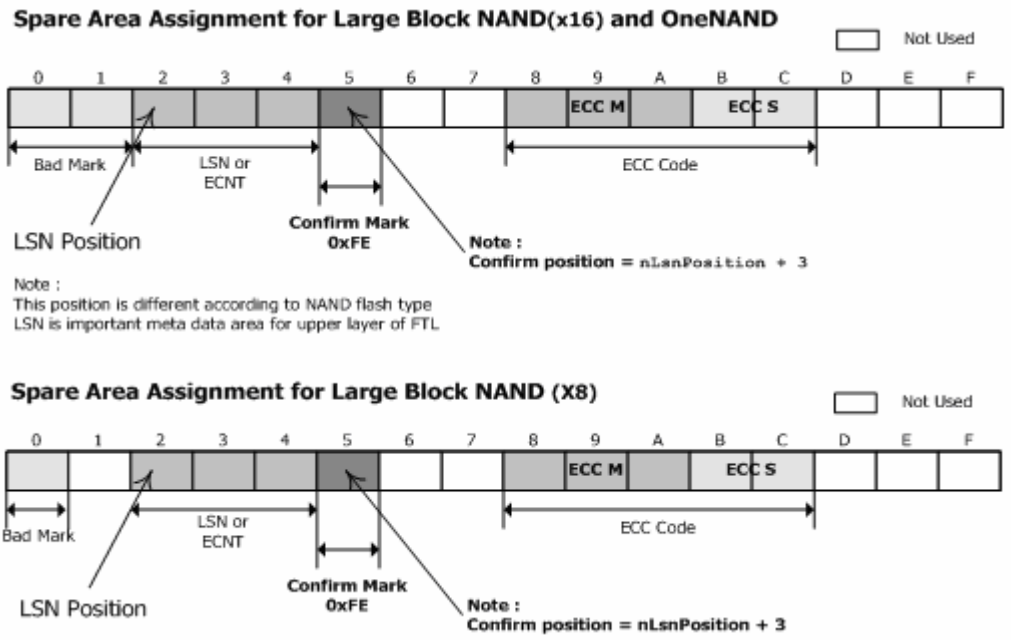


Figure 22. Spare Area Assignment for Large block NAND and OneNAND devices

Figure 22 is describes the sample of Spare assignment of large block NAND flash and OneNAND)