Exynos4212 iROM

Booting Guide

Revision 1.0

August 2011

Application Note

SAMSUNG ELECTRONICS

SAMSUNG

Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. Samsung assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained herein.

Samsung reserves the right to make changes in its products or product specifications with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

This publication does not convey to a purchaser of semiconductor devices described herein any license under the patent rights of Samsung or others.

Samsung makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Samsung assume any liability arising out of the application or use of any product or circuit and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

"Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by the customer's technical experts.

Samsung products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, for other applications intended to support or sustain life, or for any other application in which the failure of the Samsung product could create a situation where personal injury or death may occur.

Should the Buyer purchase or use a Samsung product for any such unintended or unauthorized application, the Buyer shall indemnify and hold Samsung and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that Samsung was negligent regarding the design or manufacture of said product.

Samsung Electronics Co., Ltd.
San #24 Nongseo-Dong, Giheung-Gu
Yongin-City, Gyeonggi-Do, Korea 446-711

Contact Us: *cheongwoo.1.lee@samsung.com*
TEL: (82)-(31)-209-4210
FAX: (82)-(31)-209-0837

Home Page: *http://www.samsungsemi.com*

Warning

This document is intended only for the recipients designated by Samsung Electronics Co. Ltd. (“Samsung”). As it contains the trade secrets and confidential information of Samsung which are protected by Competition Law, Trade Secrets Protection Act and other related laws, this document may not be, in part or in whole, directly or indirectly publicized, distributed, photocopied or used (including in a posting on the Internet where unspecified individuals may access it) by any unauthorized third party. Samsung reserves its right to take legal measures and claim damages against any party that misappropriates Samsung’s trade secrets or confidential information.

警 告

本文件仅向经韩国三星电子株式会社授权的人员提供，其内容含有商业秘密保护相关法规规定并受其保护的三星电子株式会社商业秘密，任何直接或间接非法向第三人披露、传播、复制或允许第三人使用该文件全部或部分内容的行为（包括在互联网等公开媒介刊登该商业秘密而可能导致不特定第三人获取相关信息的行为）皆为法律严格禁止。此等违法行为一经发现，三星电子株式会社有权根据相关法规对其采取法律措施，包括但不限于提出损害赔偿请求。

Revision History

Revision No.	Date	Description	Refer to	Author(s)
0.00	Aug. 11, 2010	• EVT0 Secure booting		CW Lee

Table of Contents

List of Figures

List of Tables

1 OVERVIEW

This application note explains the way to build the secure BL1(1st Bootloader) and BL2(2nd Bootloader) images in the booting environment of Exynos4212. iROM code(iROM Bootloader) of Exynos4212 confirms to download the BL1 image with checksum, verifies the integrity of the secure BL1 image, decrypts the secure BL1 image, and then iROM goes to BL1. In the BL1, the integrity of the secure BL2 is verified. If the secure BL2 image is verified successfully, BL1 will go to BL2. In order to verify the integrity of the secure image on each stage, iROM code provides the secure library functions to reuse in BL1 and BL2.

2 BOOT CODE

2.1 IROM CODE

Figure 2-1 shows the booting sequence in iROM. First, iROM provides the basic environments for executing the arm codes. Second, the secure BL1 is downloaded from the booting devices: SD/MMC, eMMC4.3, eMMC4.4, and NAND. Next step, iROM checks the integrity of the downloaded BL1.

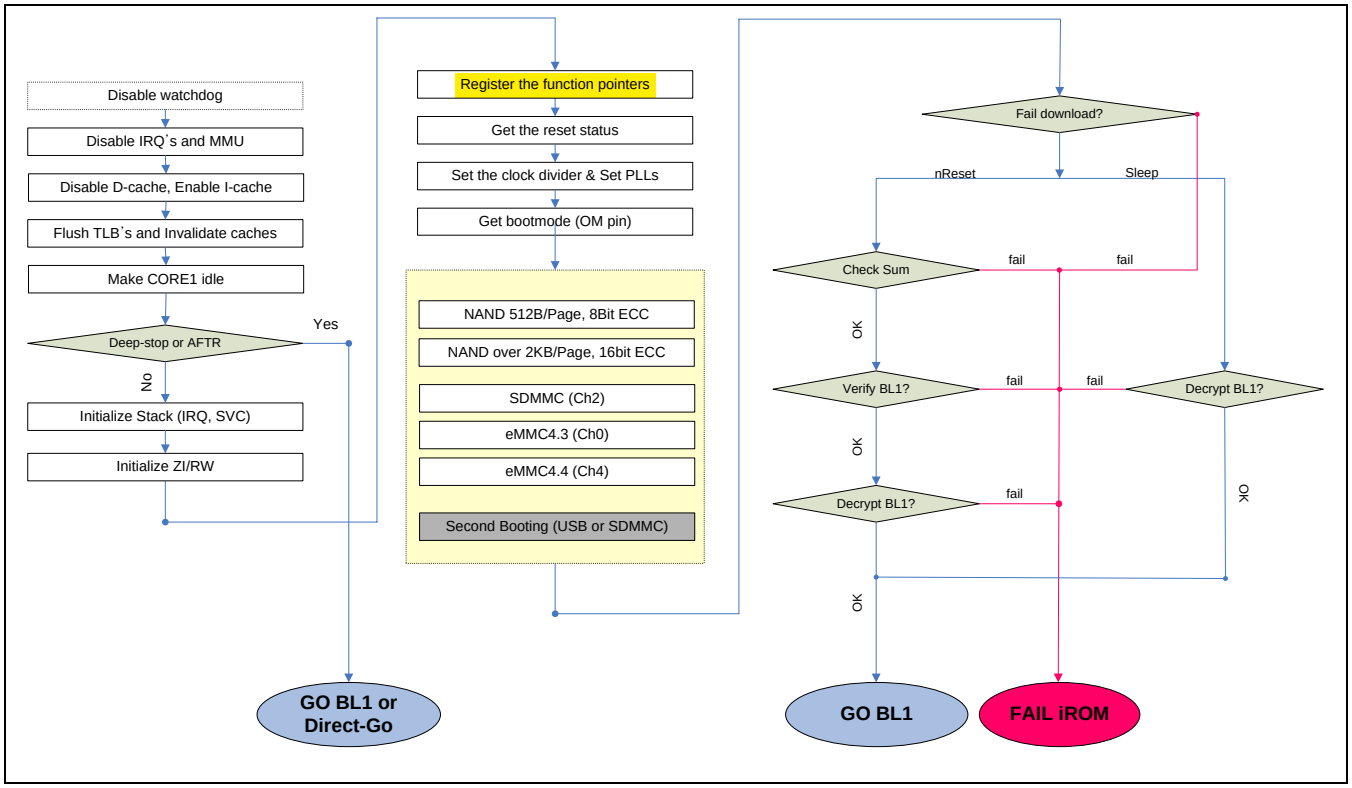


Figure 2-1 iROM Booting Sequence

The function of "Direct-Go" is provided when waking up from AFTR or Deep-stop. If the flag of Direct-Go is given at the address of 0x0202_0020 and the address of Direct-Go is given at the address of 0x0202_0024, then the next program counter will be the address of Direct-Go, not the address of BL1 reset vector. The flag of Direct-Go to be enabled is "0xFCBA_0D10".

The booting device can be selected by OM pins. Table1 shows the OM configuration for selecting the booting device.

Table 2-1 OM configuration for selecting the booting device

OM[5:1]	1st Device	2nd Device
2 (b'00010)	SDMMC_CH2	USB
3 (b'00011)	eMMC43_CH0	USB
4 (b'00100)	eMMC44_CH4	USB
8 (b'01000)	NAND_512_8ECC	USB
9 (b'01001)	NAND_2KB_OVER	USB
19 (b'10011)	eMMC43_CH0	SDMMC_CH2
20 (b'10100)	eMMC44_CH4	SDMMC_CH2
24 (b'11000)	NAND_512_8ECC	SDMMC_CH2
25 (b'11001)	NAND_2KB_OVER	SDMMC_CH2

NOTE:

OM[6] should be fixed to zero.

Just 512B of main data plus 26B of ECC data are written to the main area of each page of NAND. The remainder of each page is 'don't-care'. The main purpose is to support the various kinds of NAND devices (The size of one page and the size of one block is various. For example, 512B per page, 2048B per page, 4096B per page, and 8192B per page can be supported).
The seed of randomizer in each page is fixed to '0x59A9'.
The OM configurations of OM[5:1]=0, 1, 5~7, 10~18, 21~23, and 26~31 are reserved.

2.2 BL1 AND BL2 CODE

The guide for Exynos4212 secure booting is to use the secure boot chain such as BL1 and BL2. The purpose of the separation of BL1 and BL2 is to separate chip-dependant parts from platform-dependent parts. Chip-dependent parts contain the BL1 functions for downloading the BL2 code to internal RAM regardless of platform types. However the platform configuration should be easy to be changed by set makers such as operation frequency and memory type. And, so as to get secure context of BL1, the set makers should supply chip maker with their BL2 code public key generated by CodeSigner Client. This separation makes the set makers use their own boot image without any co-work or permission of the chip maker, once the set makers get the signed BL1image from the chip maker.

2.2.1 SECURE BL1 BOOT SEQUENCE

BL1 code copies the BL2 image to internal RAM. BL1 code checks the integrity of the BL2 image. BL1 code should be independent of external platform configuration. The role of BL1 code is to do stepping stone for BL2 code which is generated by set makers. The secure context data should be attached to the BL1 image and it contains public key for BL2 from set maker. Secure context is generated by CodeSigner Server managed by chip maker. The address of secure context is predefined in the iROM. In Chapter 3, internal memory configuration shows the detail information for BL1 memory configuration. Figure 2-2 shows the booting sequence of BL1 code.

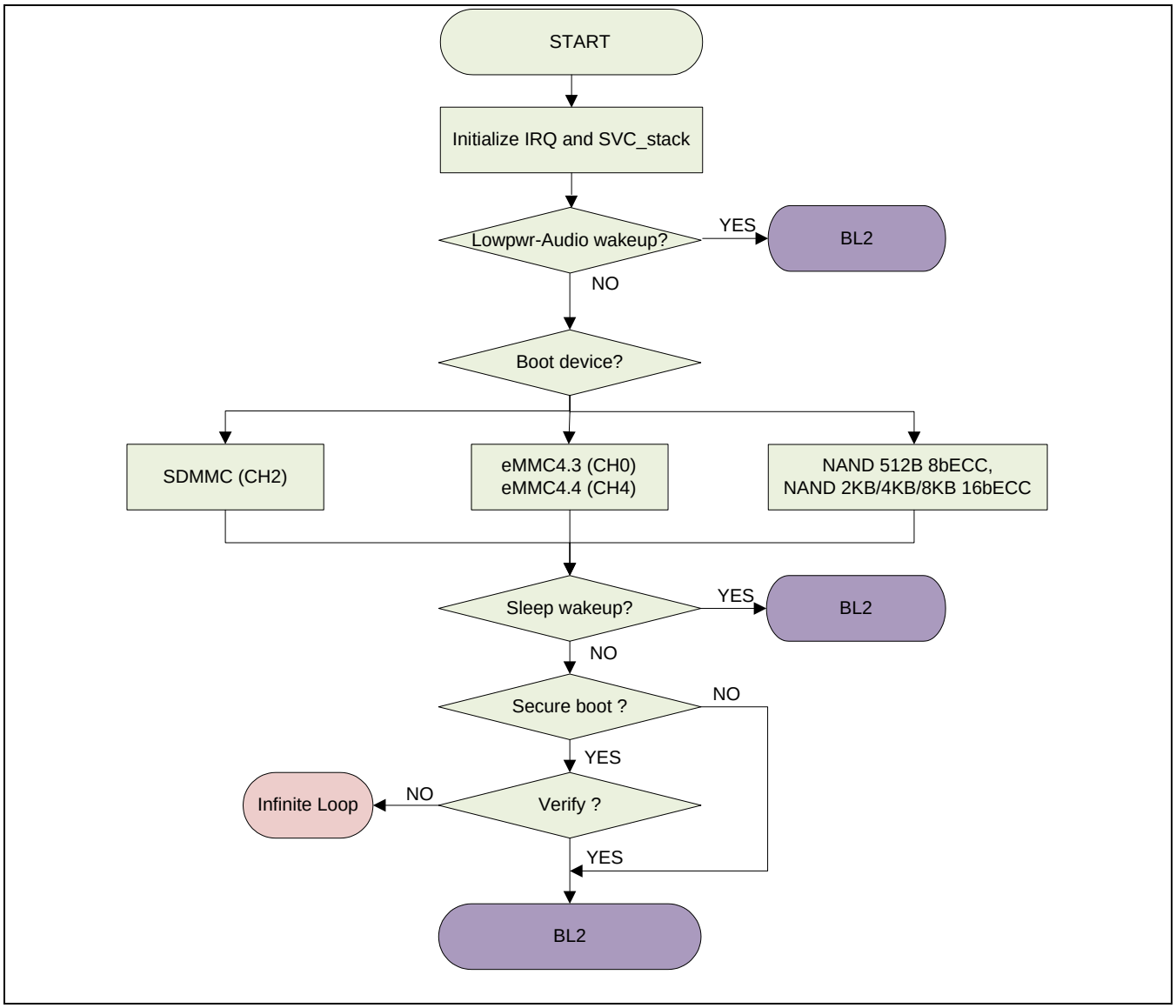


Figure 2-2 Secure BL1 Booting Sequence

2.2.2 SECURE BL2 BOOT SEQUENCE

BL2 code copies the OS image(BL3) to external DRAM area and checks the integrity of OS code. BL2 code configures the operating frequency and DRAM initialization. If there is necessary to configure additional setting to system, the set makers can configure it in the BL2 code. BL2 code is independent of BL1 code. But the address of BL2 signature is fixed in BL1 and the size of BL2 image cannot exceed the BL1 secure context area. In Chapter 3, internal memory configuration shows the detail information for BL2 memory configuration. Figure 2-3 shows the booting sequence of BL2 code.

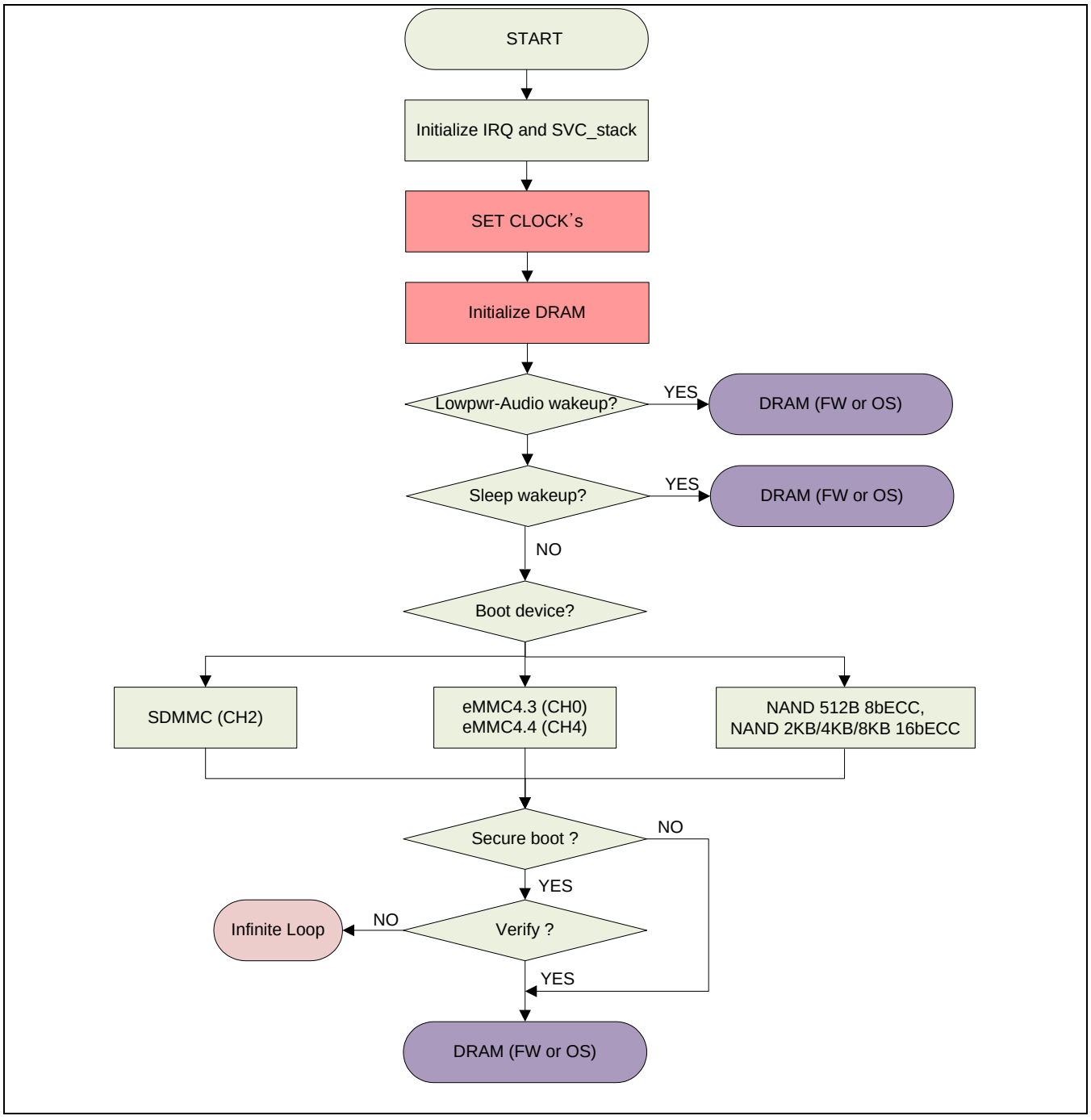


Figure 2-3 Secure BL2 Booting Sequence

2.2.3 DIRECT-GO

This is the option to skip processing of codes on BL1 and BL2 after the system wakes up from AFTR, DEEP-STOP, and LPA mode. If the specific registers are configured for Direct-go before entering AFTR (or DEEP-STOP or LPA), iROM codes will continue to dram codes without processing of BL1 and BL2. The registers for Direct-Go are as followings.

Table 2-2 Direct-Go Registers

Register Name	Address
SFR for Direct-Go flag	0x0202_0020
SFR for Direct-Go address	0x0202_0024

If the value of Direct-Go flag is equal to 0xFCBA_0D10, then next program counter after finishing iROM codes will be the dram address designated at Direct-Go address.

2.2.4 BOOTING TIME (EXAMPLES)

The running time of iROM and BL1 can be dependent on the booting device.

Table 2-3 shows an example of the running time of iROM and BL1.

The 'wakeup' means the wakeup from SLEEP mode.

Table 2-3 Booting time examples

Boot Mode	iROM -reset-	BL1 -reset-	iROM -wakeup-	BL1 -wakeup-	Example Device
SDMMC_CH2	656 ms	128 ms	279 ms	23 ms	SanDisk miniSD 8GB
eMMC43_CH0	460 ms	128 ms	83 ms	23 ms	KLMAG4FEJA-A
eMMC44_CH4	432 ms	131 ms	49 ms	26 ms	KLMAG4FEJA-A
NAND_16bECC	112 ms	192 ms	30 ms	85 ms	K9F2G08U0A

3 INTERNAL MEMORY MAP

Internal memory of Exynos4212 has been configured as shown in Figure 3-1. The size of the secure BL1 is 8192B. In order to execute iROM properly, 5KB should be reserved at the start of internal memory. The secure context for BL1 code should be located at 0x0202_3000 of internal memory. The size of BL2 code can be user defined and depends on BL1 code. However, in S.LSI's reference code of BL1, the valid size of BL2 code would be less than 14332B 14KB-4B, 4B is the checksum) and if the size of BL2 code is less than 14332B, the rest area up to 14332B should be filled with zeros. The signature for BL2 should be located 0x0202_6C00 of internal memory and the checksum for BL2 should be at 0x0202_6BFC in S.LSI's reference code.

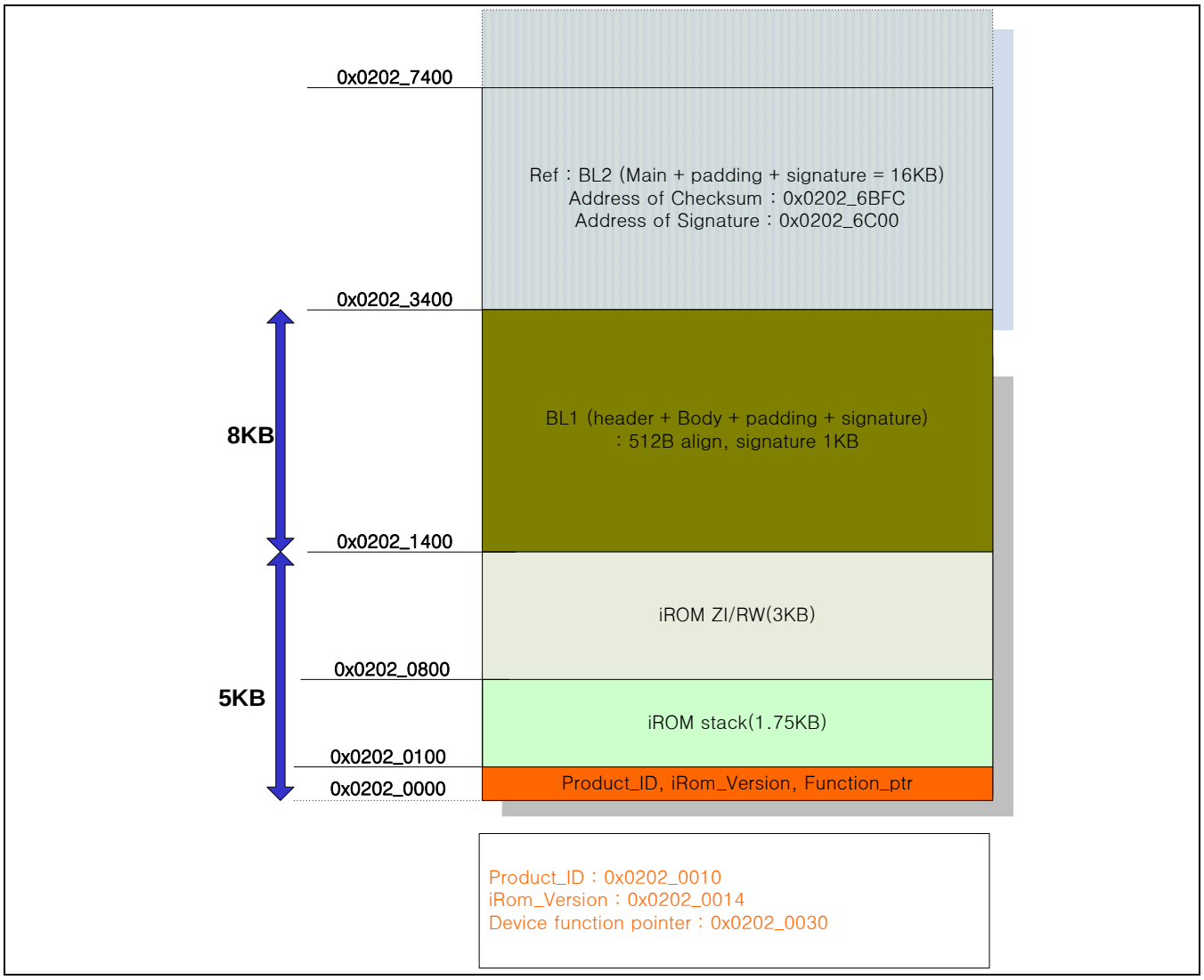


Figure 3-1 Internal Memory Map

In the internal memory map, the significant information is located on the start of internal memory. The address for device copy functions is stored from 0x0202_0030 to 0x0202_0070. The detail explanation for device copy functions is presented in the chapter 4.

4 GENERATION OF BL1 AND BL2 CODES

4.1 SECURE BOOT CONTEXT

```
#define     SB20_MAX_EFUSE_DATA_LEN          20

#define     SB20_MAX_RSA_KEY          (2048/8)
#define     SB20_MAX_SIGN_LEN         SB20_MAX_RSA_KEY

#define     SB20_HMAC_SHA1_LEN        20

//-----------------------------------------
typedef struct
{
    int             rsa_n_Len;
    unsigned char   rsa_n[SB20_MAX_RSA_KEY];
    int             rsa_e_Len;
    unsigned char   rsa_e[4];
} SB20_RSAPubKey;

typedef struct
{
    int             rsa_n_Len;
    unsigned char   rsa_n[SB20_MAX_RSA_KEY];
    int             rsa_d_Len;
    unsigned char   rsa_d[SB20_MAX_RSA_KEY];
} SB20_RSAPrivKey;
```

```
//-----------------------------------------
typedef struct
{
    SB20_RSAPubKey rsaPubKey;
    unsigned char  signedData[SB20_HMAC_SHA1_LEN];
} SB20_PubKeyInfo;

//-----------------------------------------
typedef struct
{
    SB20_RSAPubKey stage2PubKey;              // Stage 2 public key for secure booting.
    int            code_SignedDataLen;    // RSA Signature Length
    unsigned char  code_SignedData[SB20_MAX_SIGN_LEN]; // RSA Signature Value
SB20_PubKeyInfo    pubKeyInfo;                // Public Key and it's HMAC
    unsigned char  func_ptr_BaseAddr[128];    // Function pointer of iROM's
// secure boot function
    unsigned char  reservedData[80];
} SB20_CONTEXT;
```

If customers want to use one key-pair, stage2PubKey and pubKeyInfo.rsaPubkey will be same.

4.2 SECURE BL1 AND BL2 CODES

In accordance with the secure boot chain, BL1 code verifies the integrity of BL2 and BL2 code verifies the OS integrity. At that time, the files listed below are required to use the secure boot library function. These files are also used to make secure BL2 code in order to check OS integrity.

BL1_SB20_C220.c

BL1_SB20_C220.h

The table below is the lists of library functions used by BL1 and BL2 codes in order to verify the integrity of BL2

and OS image.

Table 4-1 Function Description of Check_Signature

Prototype	int Check_Signature(SB20_CONTEXT *SB20_CONTEXT_ADDRESS, unsigned char *BL2_ADDRESS, int BL2Len, unsigned char *BL2_SIGNDATA_ADDRESS, int SB20_MAX_SIGN_LEN)	
Description	Verify the image integrity	
Parameters	*SB20_CONTEXT_ADDRESS	Secure Context Base Address (=0x0202_3000)
	*BL2_ADDRESS	BL2 or OS Image Base Address
	BL2Len	BL2 or OS Image Size except Image Signature Size, Byte Count
	*BL2_SIGNDATA_ADDRESS	BL2 or OS Image Signature Base Address
	SB20_MAX_SIGN_LEN	BL2 or OS Image Signature Size(=256 Byte)
Return Value	SB_OK	BL2 or OS Image Integrity OK.(return 0x0)
	Others	BL2 or OS Image Integrity Fail.
Remarks		

Example)

```
result = Check_Signature ((SB20_CONTEXT *)0x02023000, \
(unsigned char*)0x02023400,                                    \
int(0x3800),                                                   \
(unsigned char*)(0x02026C00),          \
(int)256 );    // the size of signature 256byte

if(result == SB_OK)
        ((void (*)(void))(0x02023400))();
```

4.3 SECURE MACRO FUNCTIONS

4.3.1 GENERATION OF FUNCTION POINT ADDRESS

To reduce BL1 and BL2 code size, these codes uses secure boot functions in iROM of SoC. The followings are secure boot function used in BL1and BL2. In Exynos4212, the addresses of the secure boot functions are as below.

Secure boot functions	Description
Verify_PSS_RSASignatrue2	RSA PSS Signature verification function

When BL1 is signed using the CodeSigner, Secure Boot Context's func_ptr_Base field is stored with 0x00. After the verification of BL1's signature in iROM, the Secure Boot Context's func_ptr_BaseAddr field is filled with secure boot function address by iROM operation.

Example: In Exynos4212, Secure Boot Context's func_ptr_BaseAddr are changed as below.

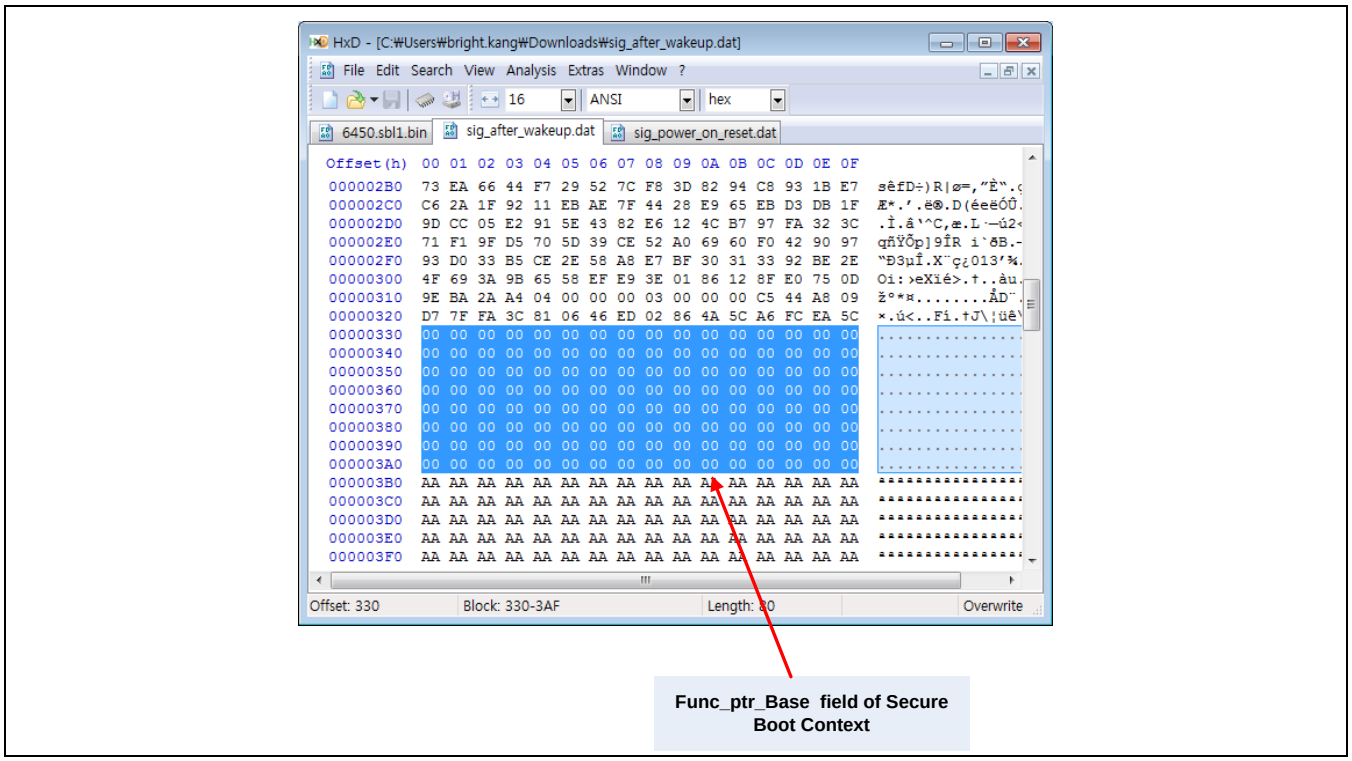


Figure 4-1 Secure Boot Context's Func_ptr_Base field before checking integrity of BL1 in iROM

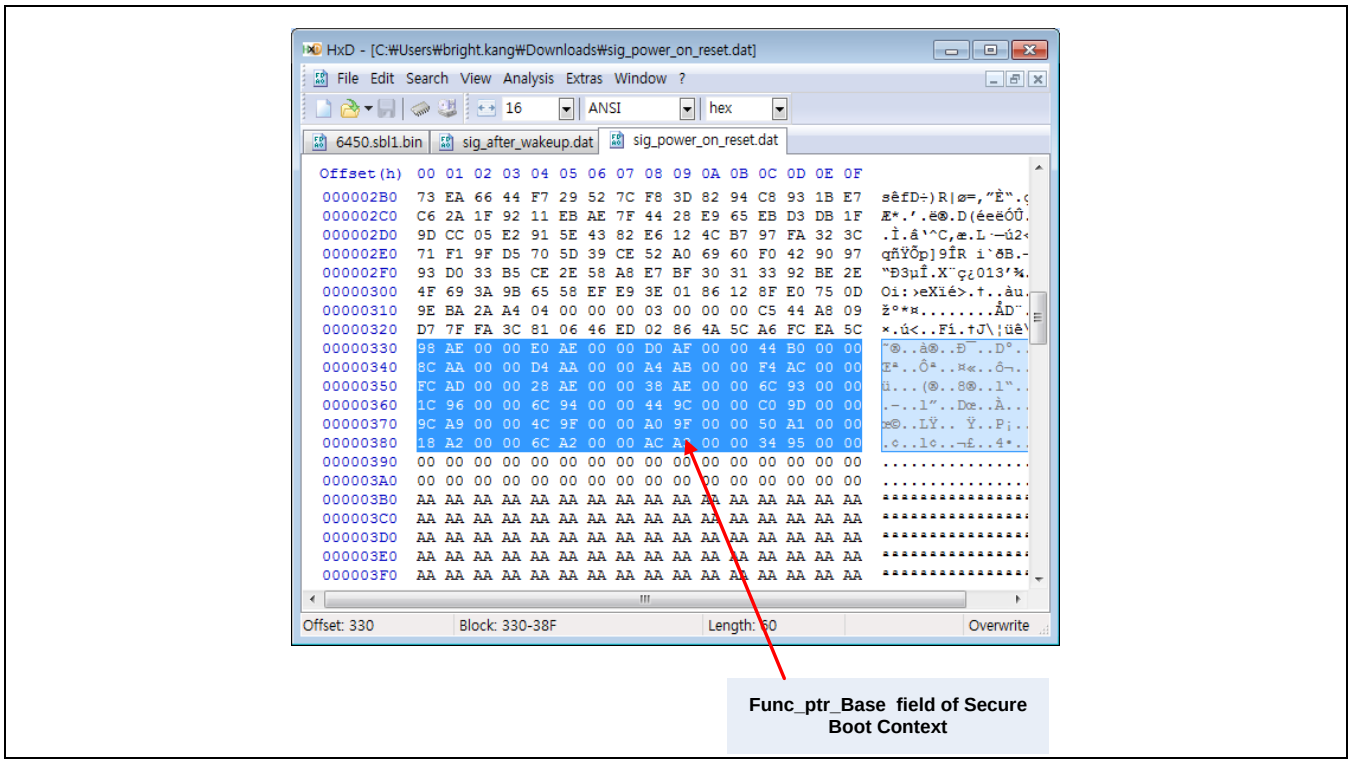


Figure 4-2 Secure Boot Context's Func_ptr_Base field after checking integrity of BL1 in iROM

To call secure boot functions in BL1, there is defined macros for secure boot functions. When the function of Verify_PSSRSASignature2 is necessary in BL1, the Macro of "macro_Verify_PSS_RSASignature2" is provided.

4.3.2 MACRO FUNCTION DESCRIPTION

Table 4-2 Macro Verify PSS RSASignature2

Prototype	#define macro_Verify_PSS_RSASignature2(BASE_FUNC_PTR,a,b,c,d,e,f) \ (((int(*)(unsigned char *, int, unsigned char *, int, unsigned char *, int)) \ (*((unsigned int *)(BASE_FUNC_PTR + 48)))) \ ((a),(b),(c),(d),(e),(f)))	
Description	macro for Verify_PSS_RSASignature2 function call in iROM.	
Parameters	BASE_FUNC_PTR	the address to store function pointer in iROM(Actually, Secure Boot Context's func_ptr_BaseAddr field)
	a	RSA public key data pointer
	b	RSA public key data length
	c	Input message pointer(i.e BL2's Address)
	d	Input message length
	e	Signature pointer

	f	Signature length
Return Value	SB_OK(0)	RSA Signature Verification is successful.
	Others	RSA Signature Verification is fail.
Remarks		

4.4 DEVICE COPY FUNCTIONS

The Exynos4212 iROM supports the block copy functions for the booting device. These internal functions can copy any data from the booting device to internal SRAM.

Table 4-3 Function Pointers for the Booting Device

Address	Name	Description
0x02020030	SDMMC_ReadBlocks	This function copies the data of SD and MMC type device to destination : Return type (True=1/False=0), Arguments (u32 SrcBlock, u32 NumofSrcBlock, void * DstByte)
0x0202003C	LoadBL2FromEmmc43Ch0	This function copies BL2 of the boot area data of eMMC 4.3 to internal RAM : Return type (True=1/False=0), Arguments (u32 SrcBlock, u32* DstByte)
0x02020040	Emmc43_EndBootOp_eMMC	This Function is ending operation for eMMC4.3 boot mode : Return type (void), Arguments (void).
0x02020044	MSH_ReadFromFIFO_eMMC	This function copies the boot area data of eMMC 4.4 to destination : Return type (True=1/False=0), Arguments (u32 SrcBlock, void * DstByte).
0x02020048	MSH_EndBootOp_eMMC	This Function is ending operation for eMMC4.4 boot mode : Return type (void), Arguments (void)
0x02020070	LoadImageFromUsb	This function copies the data through USB. If the enumeration is passed in iROM, this function could be available : Return type (True=1/False=0), Arguments (void)

Warning: The frequency of clocks supplied to SDMMC and eMMC are 20Mhz at the Booting time. MPLL is the source of these clocks.

Warning: If SDMMC or eMMC is chosen as the booting device, the copy functions for SDMMC or eMMC would be available in BL1 and BL2. If you use the copy function, please do not change the clocks for SDMMC or eMMC. Additionally do not change the configuration of PLL related to SDMMC or eMMC. Proper booting operations could not be guaranteed under illegal clock changes.

- eMMC4.4 Copy Function Address

```
void MSH_ReadFromFIFO_eMMC(u32 uNumofBlocks, void * uDstAddr)

* This Function copies eMMC4.4 Card Data to memory.
* @param uNumofBlocks : Number of Blocks for transfer operation.
* @param uDstAddr : It is indicated Destination Address (System Memory)
```

- MSH_EndBootOp_eMMC

```
void MSH_EndBootOp_eMMC(void)

* This Function is ending operation for eMMC4.4 boot mode. When end of booting mode in eMMC4.4,
you call this function. This function used for wait about end of boot operation.
```

- eMMC4.3 Copy Function Address

```
bool SDMMC_ReadOperation_eMMC(u32 uNumofBlocks, void * uDstAddr)

* This Function copies eMMC4.3 Card Data to memory.
* @param uNumofBlocks : Number of Blocks for transfer operation.
* @param uDstAddr : It is indicated Destination Address (System Memory)
```

- Emmc43_EndBootOp_eMMC

```
void Emmc43_EndBootOp_eMMC(void)

* This Function is ending operation for eMMC4.3 boot mode. When end of booting mode in eMMC4.3,
you call this function. This function used for wait about end of boot operation.
```

- SDMMC_ReadBlocks

```
void SDMMC_ReadBlocks(u32 uStBlock, u32 uNumofBlocks, void * uDstAddr)

* This Function copies SD/MMC Card Data to memory.
* @param uStBlock : Start Block Number. It is indicated the start block's location.
```

```
* @param uNumofBlocks : Number of Blocks for transfer operation.
* @param uDstAddr : It is indicated Destination Address (System Memory)
```

- LoadImageFromUsb

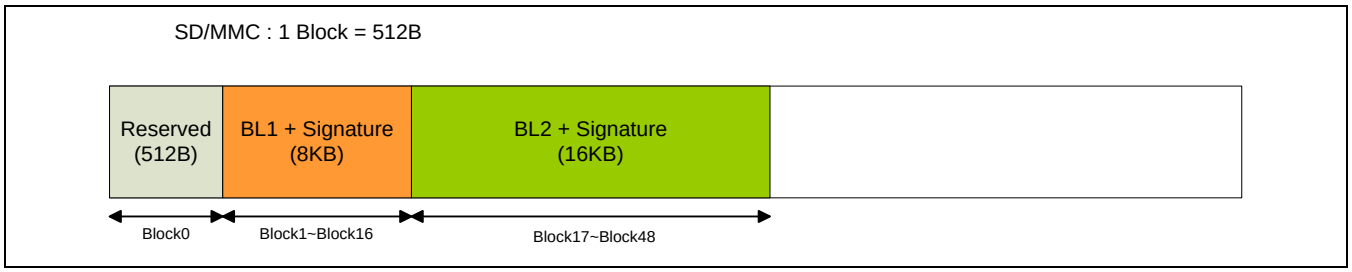
```
bool LoadImageFromUsb(void)

* This Function copies the Booting Data through.
* @void
```

4.5 ASSIGNMENT GUIDE FOR THE BOOTING BLOCKS

iROM will copy the 8KB of data from the booting device regardless of the secure and the non-secure.

4.5.1 SD/MMC/MOVINAND

BL1(1st Boot loader) should be located at the offset of 512B. iROM only loads 8KB BL1 code to internal memory.

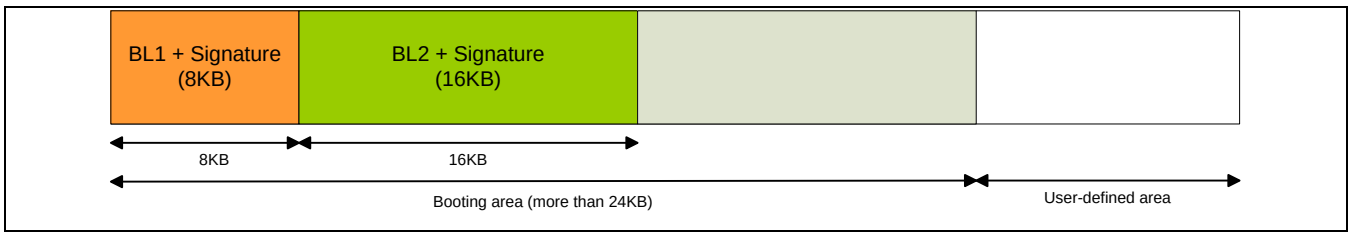


4.5.2 EMMC4.3 AND EMMC4.4

The eMMC4.3 device has the separated boot area in the boot operation mode. The size of boot area is determined by Extended CSD register.

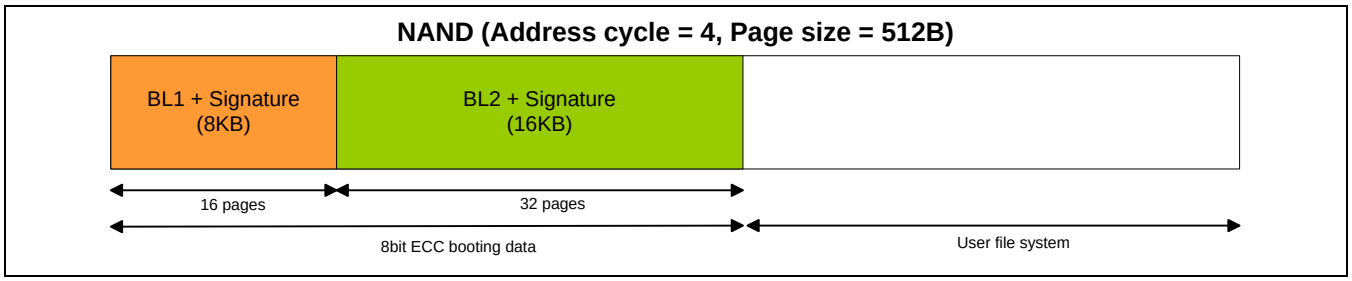
This guide is a sample, but there are 2 mandatory rules.

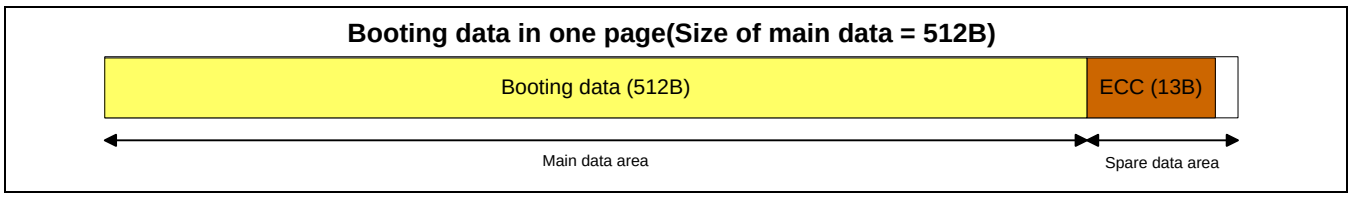
- BL1(1st Boot loader) should be located at block 0 of the booting block.
- BL2(2nd Boot loader)'s location should be the consecutive position of BL1.



4.5.3 NAND (ADDRESS CYCLE 4, PAGE SIZE = 512B)

BL1(1st Boot loader) should be located at block 0 and page 0.
The data written to Block0 should be encoded by 8bit ECC.



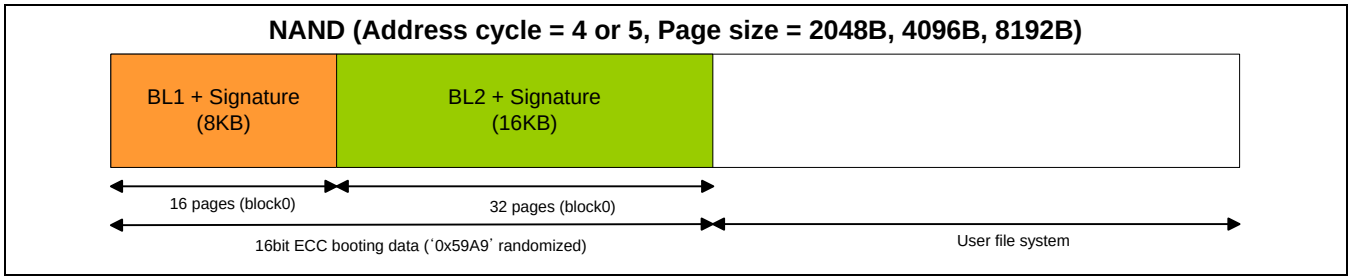


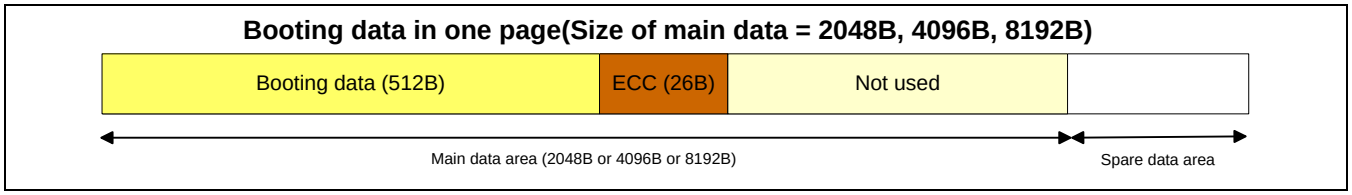
4.5.4 NAND (ADDRESS CYCLE 4 OR 5, PAGE SIZE = 2048B, 4096B, 8192B)

BL1(1st Boot loader) should be located at block 0 and page 0.

BL1 is encoded by 16bit ECC.

Each page has 512B message data and 26B ECC data in main area and spare area is not used.





5 CORE #1 BOOT REGISTER

In iROM, the core #0 is used for the booting procedure and the core #1 is in the idle state at the beginning. If a programmer wants the core #1 to escape from the idle state, the next program counter of the core #1 should be written to the address of 0x0202_0000(Core#1 boot register) by core#0. Next step, the core#1 will start to run after setting event to core#1 from core#0.

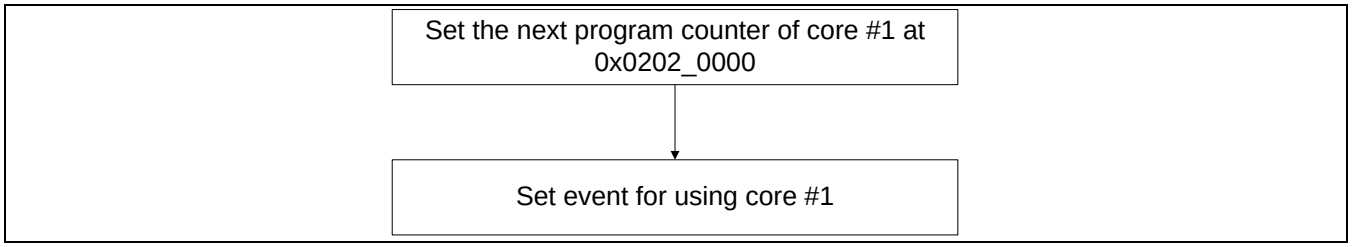


Figure 5-1 Core #1 Escaping From the Idle State

6 EMMC GUIDE

6.1 EMMC POWER CONTROL

For iROM to support eMMC 4.4 Boot mode and reset mode, the Power cycling circuit should be adapted very carefully. The Power cycling circuit and iROM Boot code perform to keep level of VCCM and VCCQ of eMMC4.4 device low below 0.5V for a few periods. By controlling voltage level of VCCM and VCCQ, eMMC4.4 status returns to the pre-idle state. So IROM is back to boot mode and can receive boot data(BL1, BL2) from eMMC4.4 slave.

The example eMMC power cycling circuit is as follows

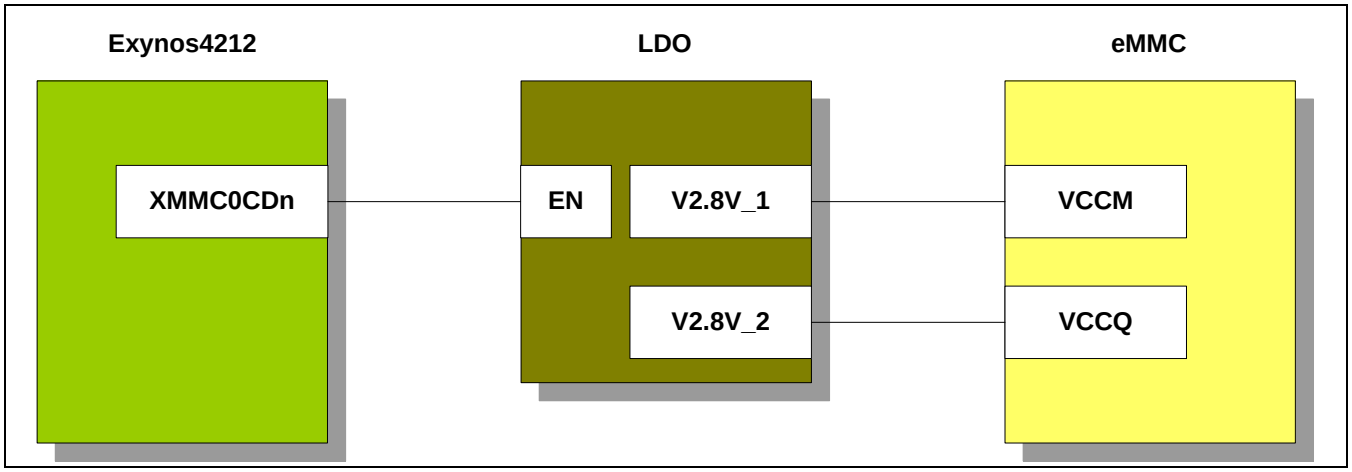


Figure 6-1 eMMC Power Control Concept

In the power circuit two things must be considered.

First, The LDO enable input signal is connected with a GPIO control pin. In the above example, The XMMC0CDn pin controls LDO power. When a Exynos4212 system boots or reset happens, XMMC0CDn pin goes to low status"0" for some periods. So LDO output is disabled for some periods. When XMMC0CDn turns to high status"1", LDO enable is on and then V2.8V_1, V2.8V_2 outputs can supply eMMC VCCM, VCCQ with 2.8V voltage.

This action performs that eMMC status returns to pre-idle state.

How long the GPIO pin should keep low status is a very important problem. By eMMC 4.4 spec, the VCCM or VCCQ of eMMC should be below 0.5V longer than 1ms for slave to return to the pre-idle state. But when using MoviNAND, Both VCCM and VCCQ should keep low status more than 1ms to prevent eMMC booting fail.

Second, because LDO discharge time may be various, The period in which VCCM and VCCQ are below 0.5V may be considered very carefully.

If a LDO discharge time is very long, XMMC0CDn can't control LDO output voltage level correctly.

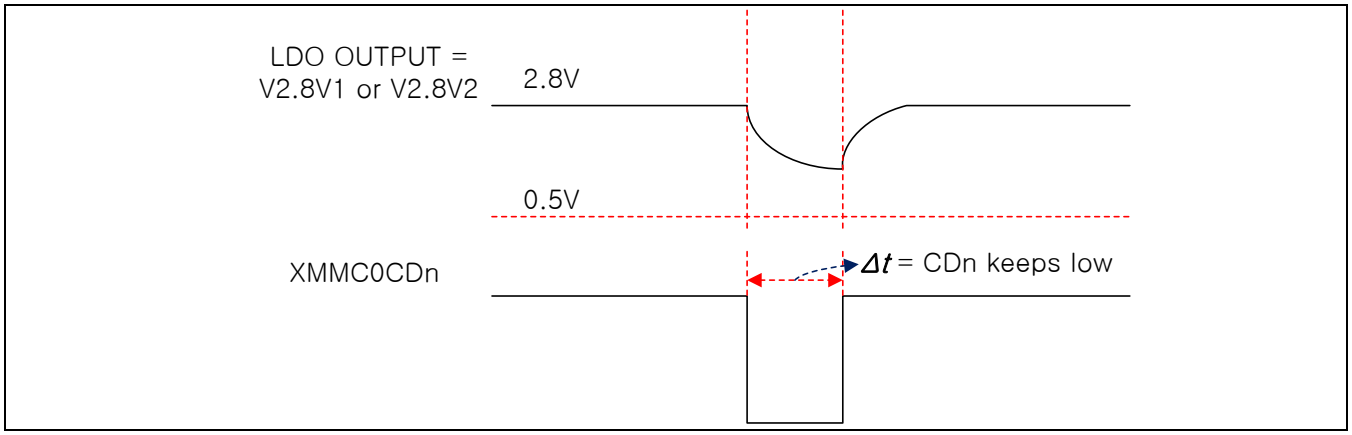


Figure 6-2 eMMC Power Control Concept

If LDO discharge time is long, LDO output can't reach voltage level which is lower than 0.5V or keep low level for 1ms. So ∆t should be long enough for LDO output voltage to reach voltage level than 0.5V.

Exynos4212 IROM can support various ∆t period. A customer who want to change ∆t period can modify the value after reset booting.