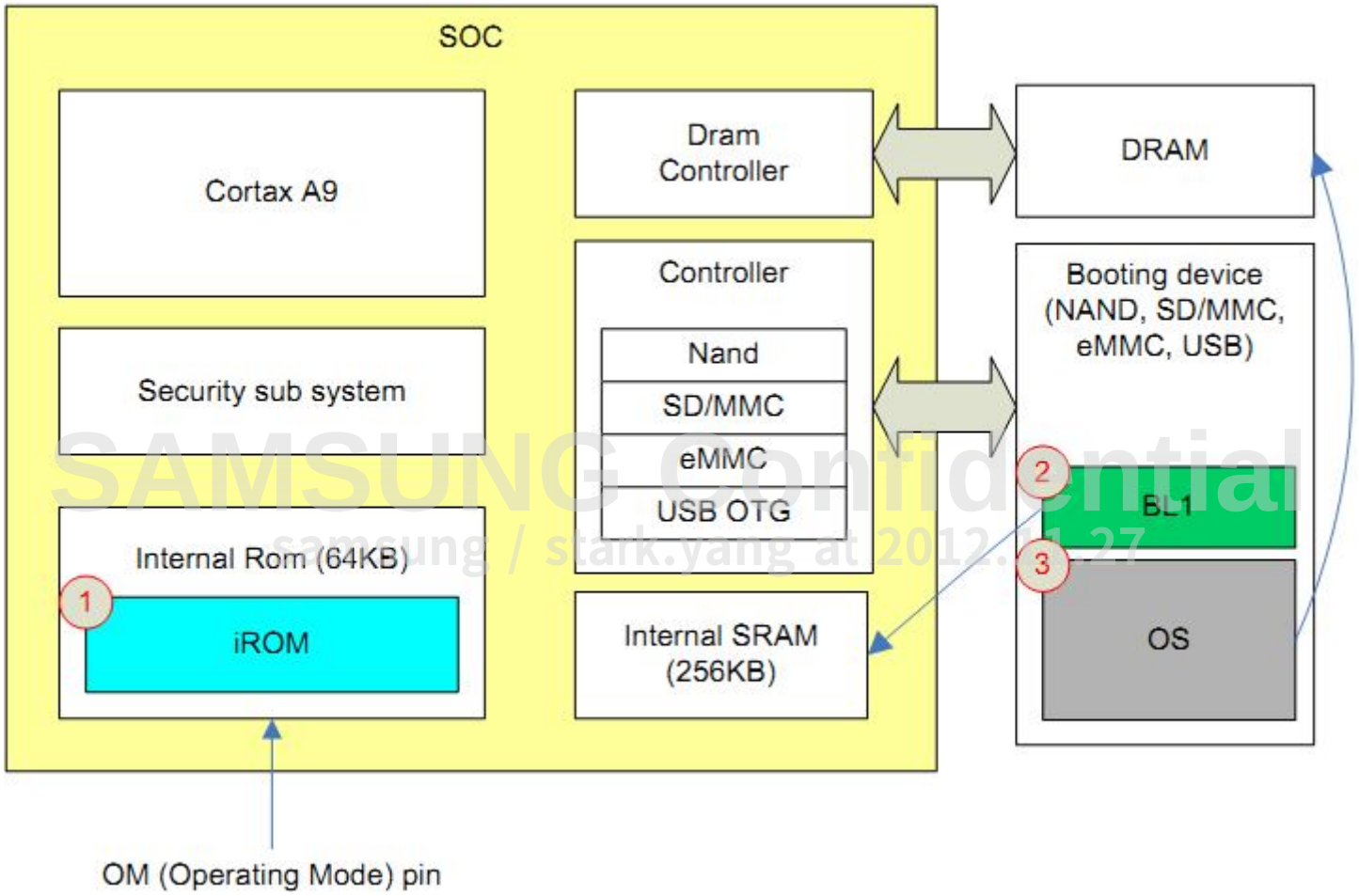
u-boot mr1 structure

Training Document

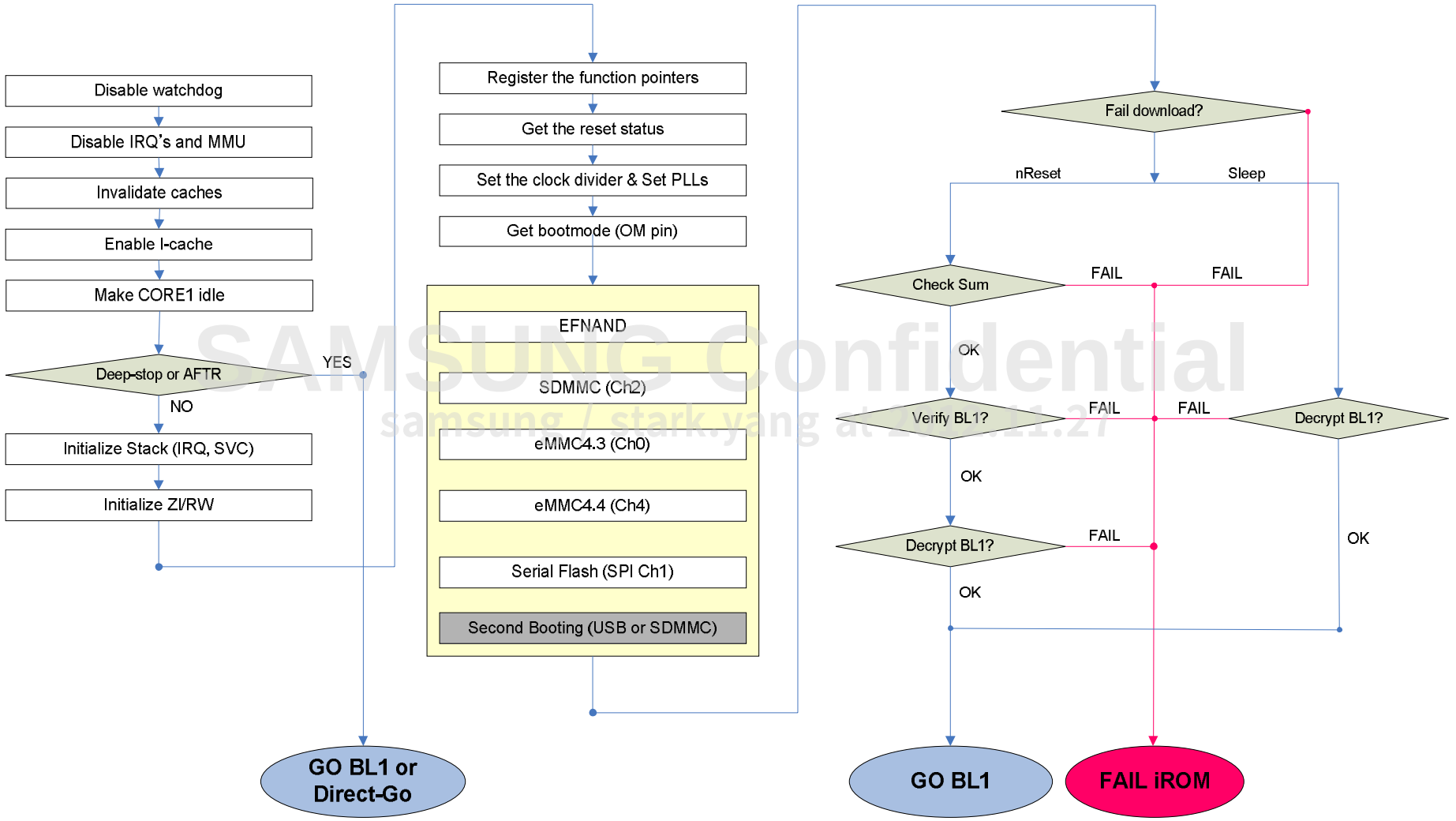
Index

- Overview
- Booting Flow Chart
- Functions of ISP
- Feature List
- Architecture of ISP Solution
- Android HAL
- H/W Design Guide
- Process of Technical Support

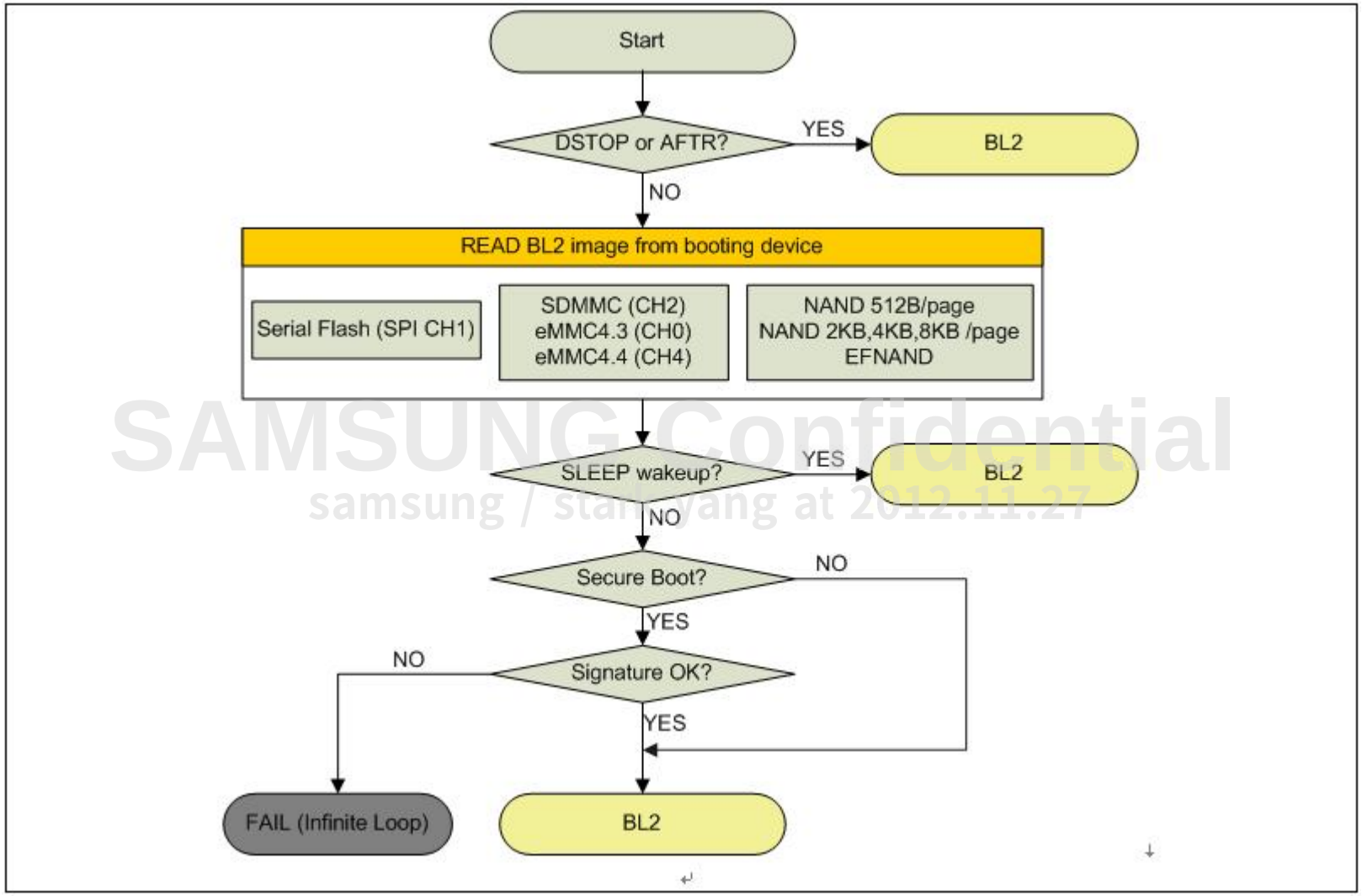
Overview



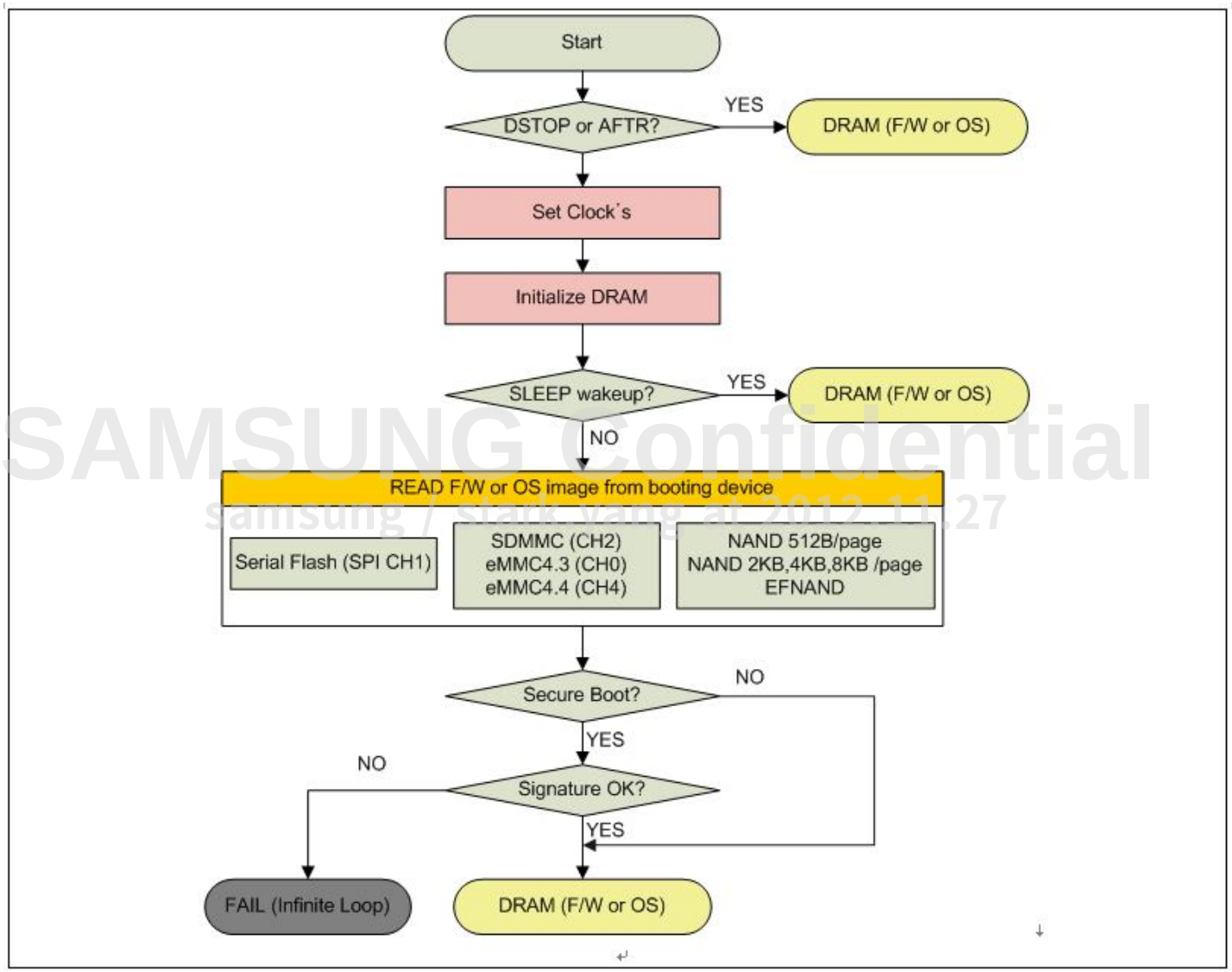
iROM booting flow chart



BL1 booting flow chart



BL2 booting flow chart

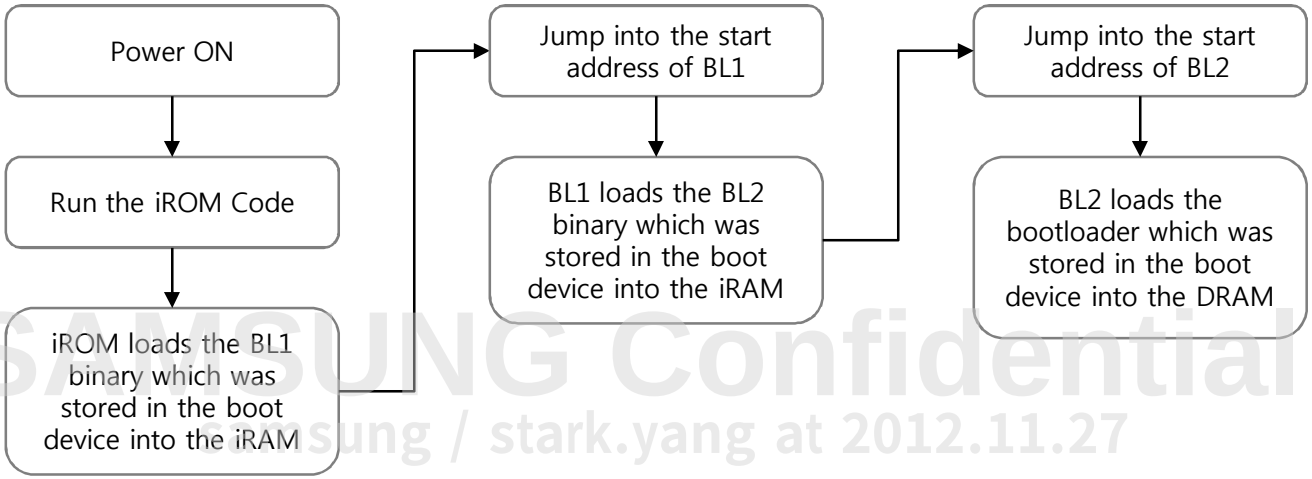


Definition

- Secure Boot?
  - Purpose
    - Prevent the bootloader/kernel/android image from being changed with an image which can be used for malicious purpose
  - How
    - Encrypt/Decrypt the image using public/private key
      - RSA (Rivest-Shamir-Adleman) Public-Key Encryption Algorithm
      - AES (Advanced Encryption Standard) Block Encryption Algorithm

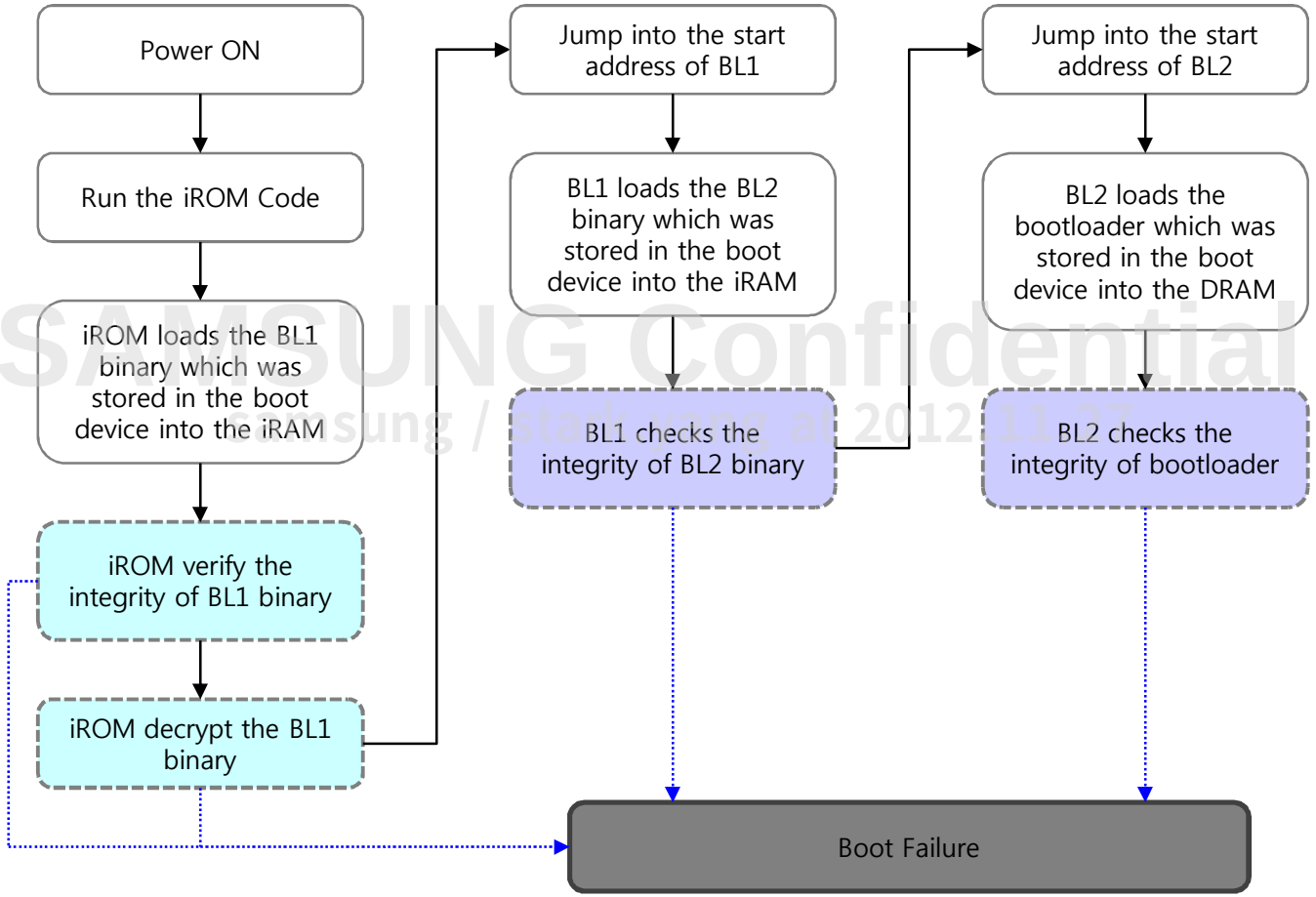
Boot Sequence : Normal

- Normal Boot Sequence

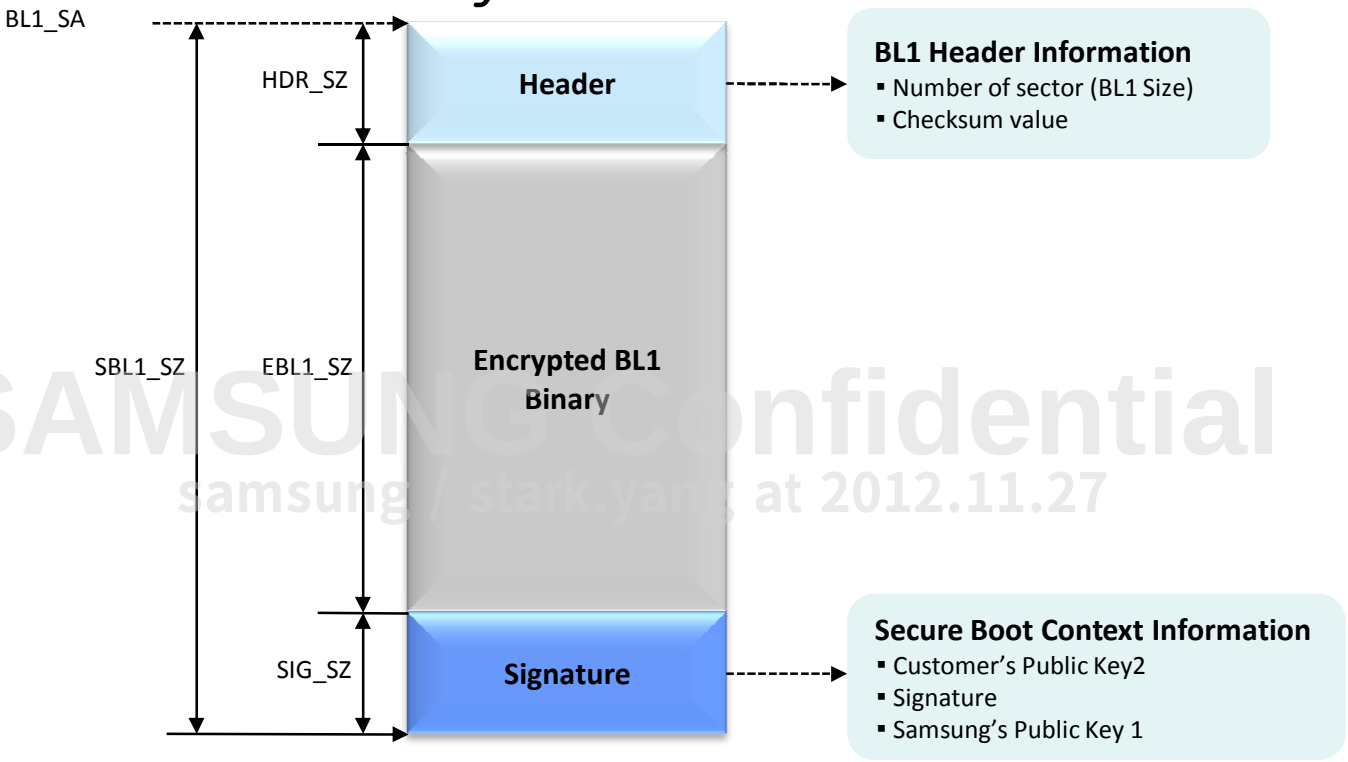


Boot Sequence : Security Featured

- Security Featured Boot Sequence

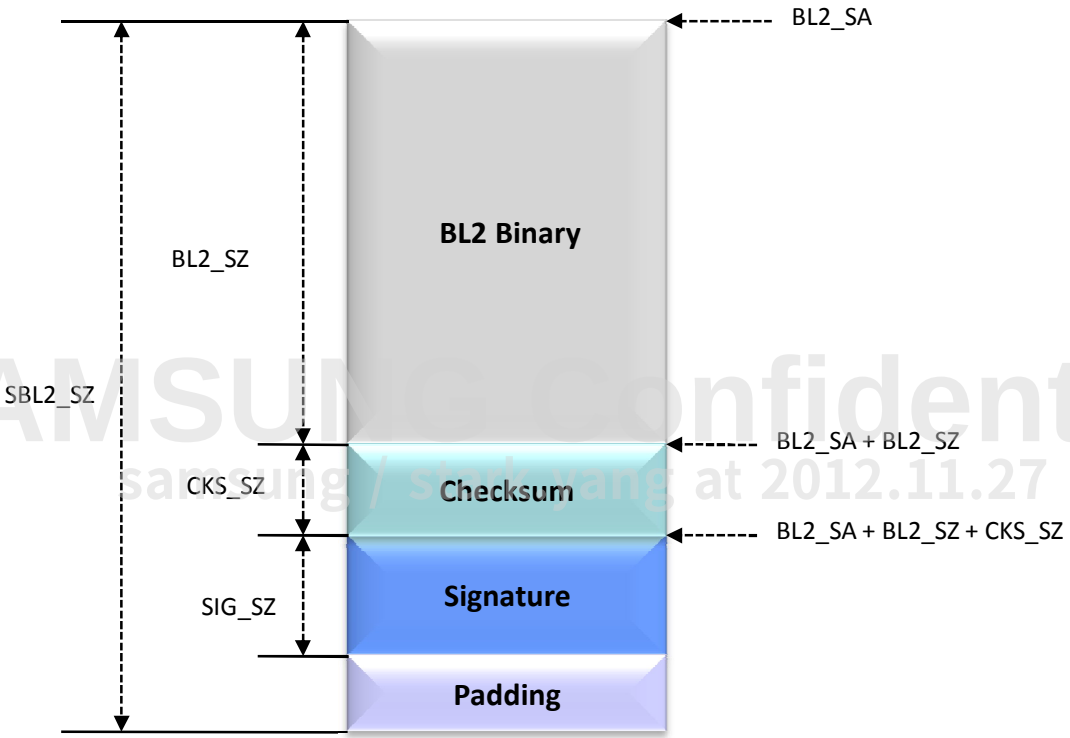


- Signed BL1 Binary Structure



	BL1 Start Address (BL1_SA)	Header Size (HDR_SZ)	Encrypted BL1 Size (EBL1_SZ)	Signature Size (SIG_SZ)	Signed BL1 Size (SBL1_SZ)
uboot	02021400h	16Byte	7152Byte	1024Byte	8KB
uboot_mr1	02021400h	16Byte	14336Byte	1024Byte	15KB

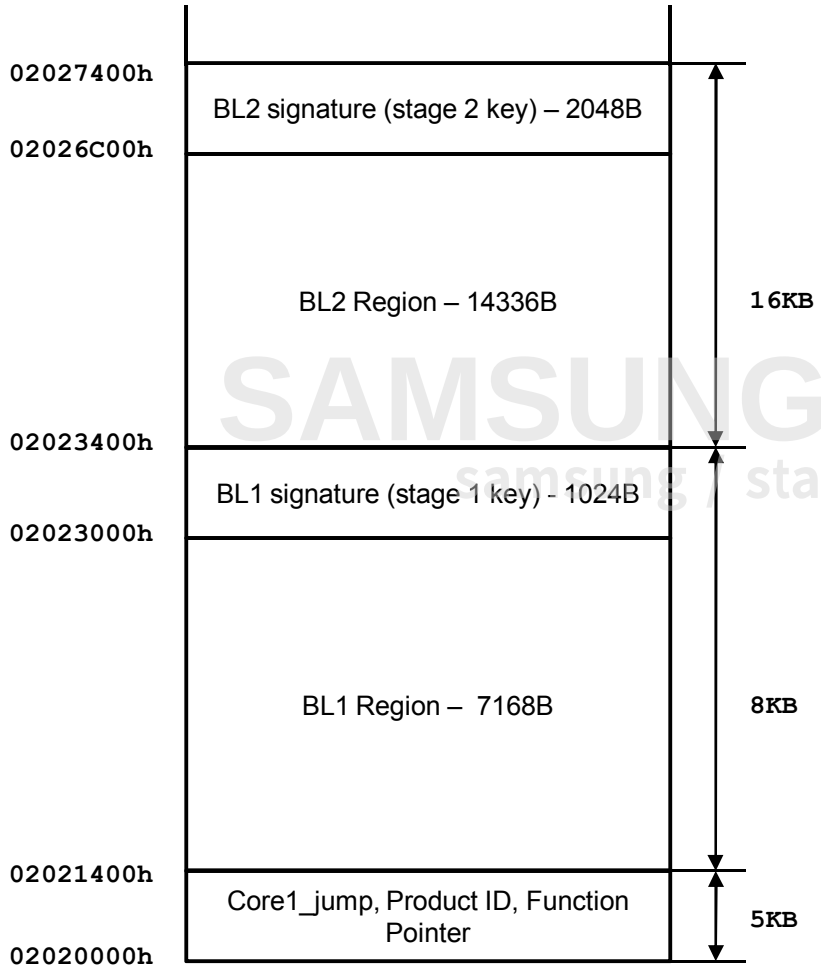
- Signed BL2 Binary Structure



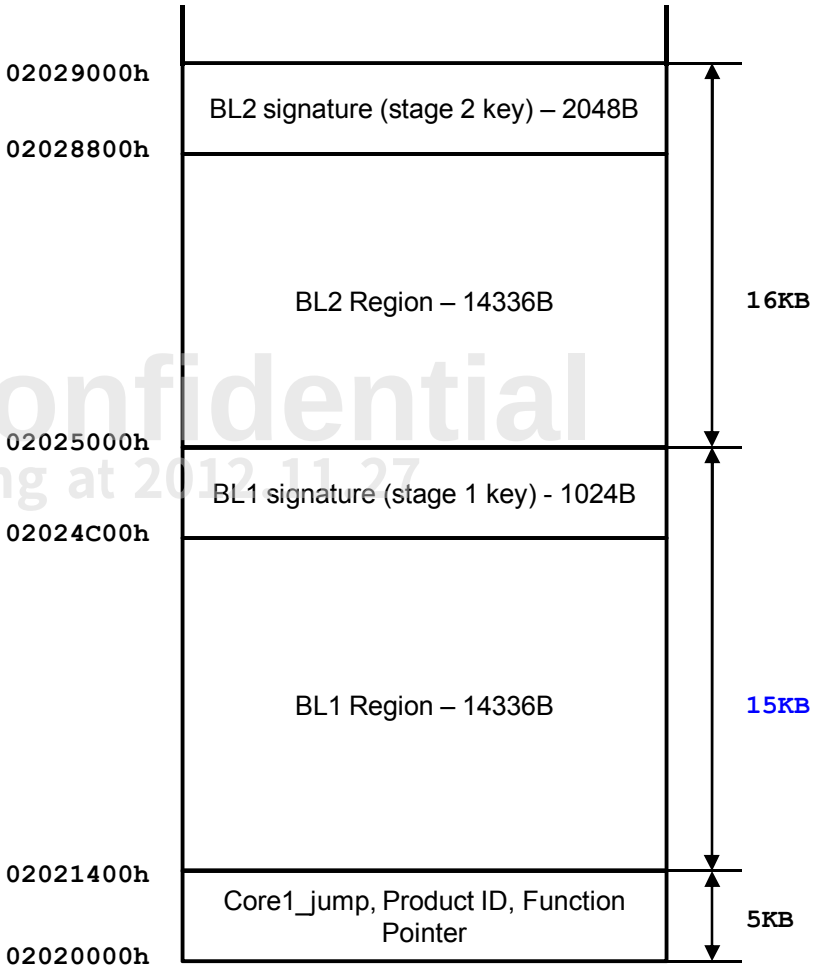
	BL2 Start Address (BL1_SA)	BL2 Size (HDR_SZ)	CheckSum Size (CKS_SZ)	Signature Size (SIG_SZ)	Signed BL2 Size (SBL2_SZ)
uboot	02023400h	14332Byte	4Byte	256Byte	16KB
uboot_mr1	02025000h	14332Byte	4Byte	256Byte	16KB

Internal Memory Map

- Exynos4412



- Exynos4412 Prime



Bootloader Conclusion

[1] CodeSigner_V21 -v2.1 Exynos4412 -STAGE2_KEYGEN
->Exynos4412_V21.prv + Exynos4412_V21.spk

[2] BL1: Realse binary code by HQ
[sbl1]public key + raw BL1 binary -> encrypted binary file
8K → 15K

[3] BL2:

1) split -b 14336 u-boot.bin bl2 (14336 = 14K)
2) 1-word Checksum : $(tc4_uboot)/ sdfuse_q/chksum.c
3) Signature: use Exynos4412_V21.prv (+256B)
4) Add padding to 16384 = 16K

Bootloader Conclusion

[4] uboot.bin

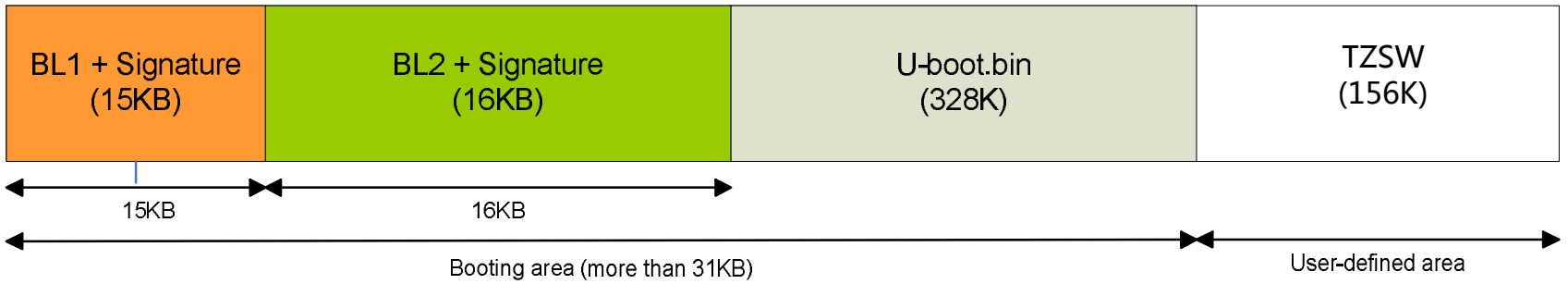
Add padding

uboot.bin 335872 = 328K

[5] TrustZone = 92K → 156K

BL1(15K) + BL2(16K) + uboot.bin(328K) + TZ(156K)

= u-boot-mr-exynos4412-evt2-efused-tz.bin(515K)

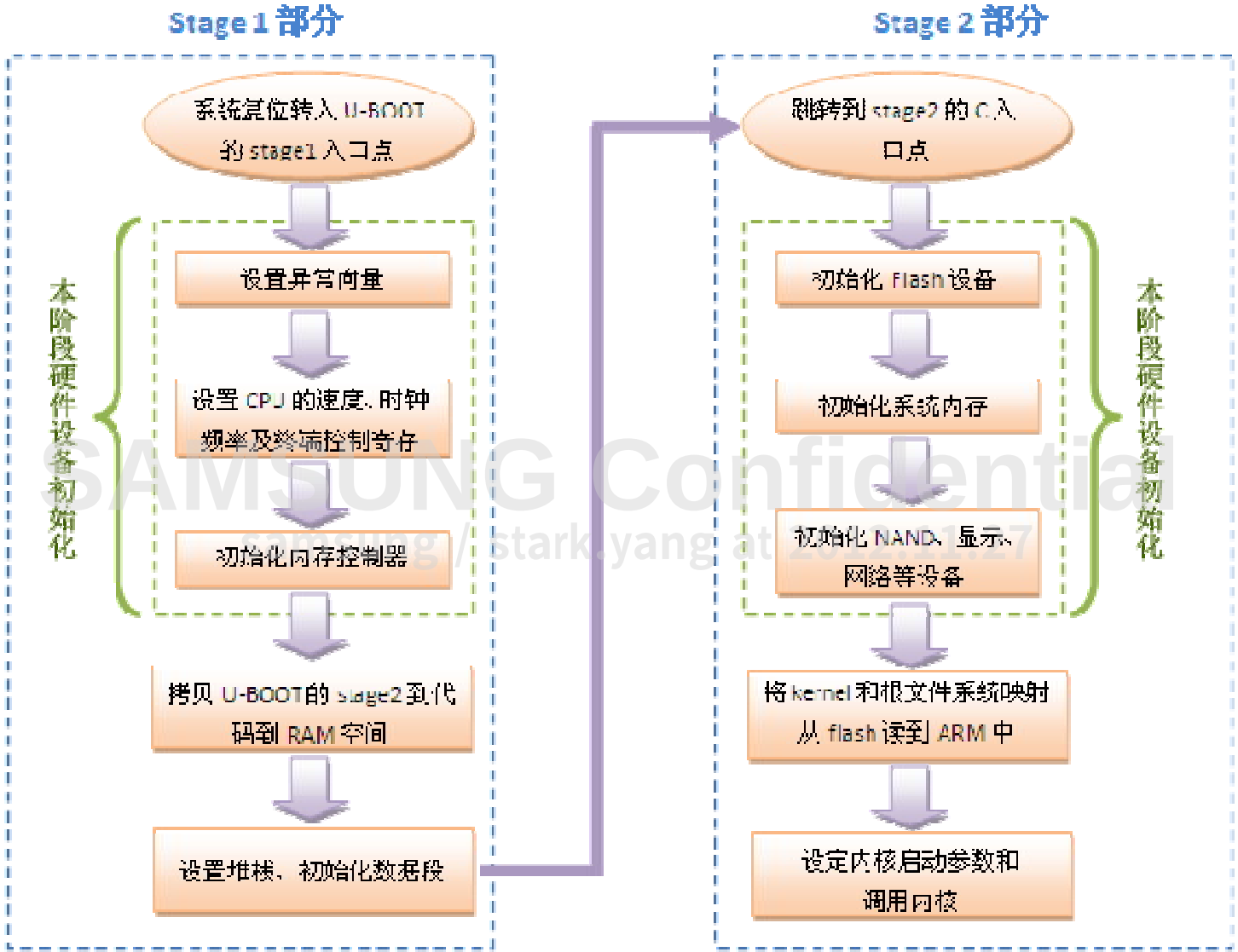


New SBL1 & TZSW Feature

- 15K SBL1
  - ➢ fix the LPA bug (HQ, SSCR没有重现过)
  - ➢ support 156K TZSW

- 156K TZSW
  - ➢ support 2G DRAM
  - ➢ 寄存器的访问策略 → Kernel

u-boot Boot Sequence



U-BOOT 启动流程图

uboot_mr1

- u-boot version 2010.03 → 2010.12

➢代码结构不变

➢(函数的名称，接口)

uboot_mr1

- 文件目录结构上的主要改变: 把处理器架构体系相关的内容合并,移到arch目录下.

➢ lib_xxx/ → arch/xxx/lib/

➢ cpu/xxx/ → arch/xxx/cpu/

➢ include/asm-xxx/ → arch/xxx/include/

➢ lib_[arch]通用 → lib/

- 移植工作主要在 arch/xxx 和board 目录 (以前是cpu/xxx/和board)

Main Procedures

- Uboot/cpu/arm_cortexa9/start.S
- Uboot/board/samsung/smdkc210/lowlevel_init.S

- Uboot_mr/arch/arm/cpu/armv7/start.S
- Uboot_mr/board/samsung/smdk4212/
  lowlevel_init.S
  clock_init_smdk4212.S
  mem_init_smdk4212.S

- Uboot/lib_arm/board.c

- Uboot_mr/arch/arm/lib/board.c

- ./common/main.c

u-boot Command

- ./common/cmd_**.c
  - U_BOOT_CMD(name,maxargs,rep,cmd,usage,help)

U_BOOT_CMD

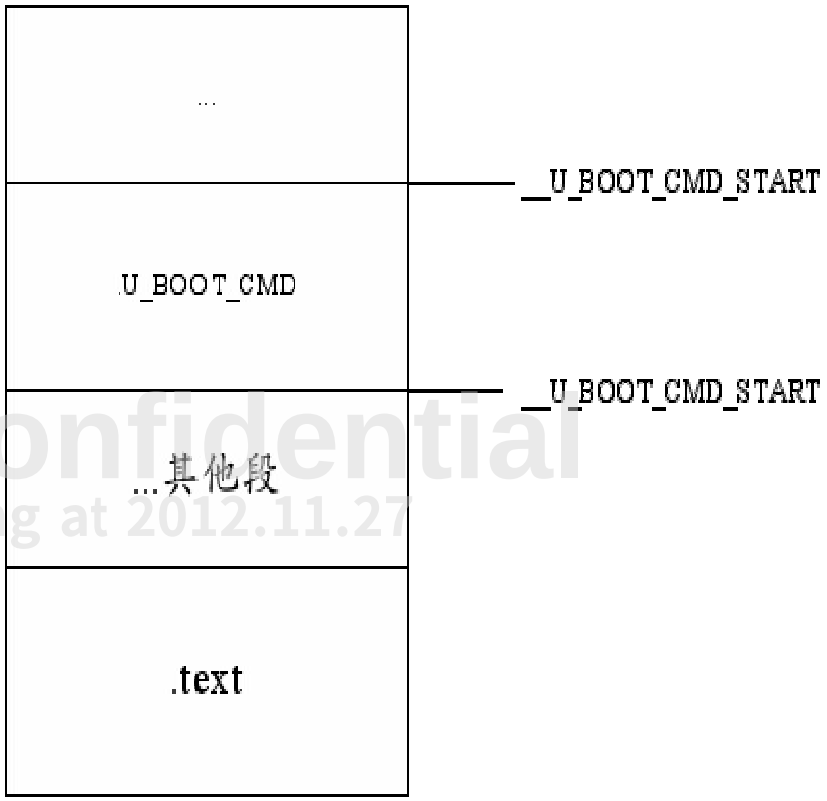
UBoot是利用U_BOOT_CMD保存用户名和相对应要处理的函数（include₩command.h)

```
struct cmd_tbl_s {
          char      *name; /* Command Name                              */
          int       maxargs;  /* maximum number of arguments           */
          int       repeatable;          /* autorepeat allowed?                    */
                              /* Implementation function    */
          int        (*cmd)(struct cmd_tbl_s *, int, int, char *[]);
          char      *usage;   /* Usage message    (short)    */
#ifdef    CONFIG_SYS_LONGHELP
          char      *help;    /* Help  message    (long)     */
#endif
#ifdef CONFIG_AUTO_COMPLETE
          /* do auto completion on the arguments */
          int        (*complete)(int argc, char *argv[], char last_char, int maxv, cha
r *cmdv[]);
#endif
};

#define Struct_Section  __attribute__ ((unused,section (".u_boot_cmd")))

#define U_BOOT_CMD(name,maxargs,rep,cmd,usage,help) \
cmd_tbl_t __u_boot_cmd_##name Struct_Section = {#name, maxargs, rep, cmd, usage, help}
```

u-boot Command

查看lds链接脚本发现的.u_boot_cmd放在了一起。而起还定义了两个常量__u_boot_cmd_start和__u_boot_cmd_end还表示所有命令的起始位置和结束位置。

所以只需对该段进行遍历就可以得到所有的命令了。



Recovery

- 进入main_loop之前判断
- ./arch/arm/cpu/armv7/exynos/recovery.c

- 寄存器状态 →

1. Factory data reset

➢erase userdate & cache

➢format fat_partition

➢(预装应用)

Recovery

- 开机时,使用(电源键+音量键)的组合

- 检查按键状态 →

2. Update from sd/mmc

类似于fastboot flash, 源(usb → sdcard)不同

./common/cmd_fastboot.c sdfuse命令

./common/cmd_movi.c movi命令

3. Recovery mode

Kernel + Recovery-ramdisk

bootm 参数 → BOOTCMD

Logo Display

- 在Kernel起来之前，uboot第二阶段显示Logo

0. Logo data r/w.
   fastboot / movi / partition~
1. Display Controller / MIPI-DSI driver
2. LCD driver
3. PMIC - LCD Power
4. Display funtion

- ./arch/arm/cpu/armv7/exynos/fimd.c

Add Partitions

- Need more partitions
- Already have 4 primary partitions

➢ 改动最少 → P + P + P + E 方案

➢ ./common/cmd_mmc_fdisk.c　分区创建,fdisk命

EMBR的写和读

➢ ./common/cmd_fat.c & cmd_ext2.c

分区的格式化, fatformat & ext3format命令

Logical Partitoion的start_block和block_cnt

➢ ./common/cmd_fastboot.c　Image写入, fastboot命令

分区与sd/mmc地址的印射

Add Partitions

- 扩展分区表中相对扇区地址

Add Partitions


```
    for(i=0; i<=CFG_MORE_PARTITIONS_NUM; i++)     {
          block_start = 0x3F;
          block_offset = calc_unit(Logical_Part_Size[i], sdInfo) - 0x3F;
          partInfo[0].bootable          = 0x00;
          partInfo[0].partitionId       = 0x83;
          make_partitionInfo(block_start, block_offset, sdInfo, &partInfo[0]);
////////////////////////////////////////////////////////////
          block_remain -= (block_offset + 0x3F);
          ////////////////////////////////////////////////////////////
          block_offset = block_remain;
          block_start = partInfoEx.block_count - block_remain;
          partInfo[1].bootable          = 0x00;
          partInfo[1].partitionId       = 0x05;
          make_partitionInfo(block_start, block_offset, sdInfo, &partInfo[1]);
////////////////////////////////////////////////////////////
          memset(embr, 0x00, 512);
          embr[510] = 0x55; embr[511] = 0xAA;
          encode_partitionInfo(partInfo[0], &embr[0x1BE]);
          encode_partitionInfo(partInfo[1], &embr[0x1CE]);
////////////////////////////////////////////////////////////
          embr += 512;
    }
```

Q&A