Android Build System Overview

: main.mk, Android.mk, bootloader, linux kernel

chunghan.yi@gmail.com, slowboot

Revision	작성자	비고
0.1	이 충 한	최초 작성 *02/14/2012*
0.2	이 충 한	1차 수정 *02/15/2012*
0.3	이 충 한	2차 수정 *03/12/2012*
0.4	이 충 한	*07/18/2012*
0.5	이 충 한	3차 수정 *08/04/2012*

목차

1. Android ICS build 절차(1)

- ***<how to build>***
- *# source build/envsetup.sh*
- *# choosecombo*

  1 *<- release*

  msm8660_surf *<- 숫자를 입력하면 에러 발생함.*

  3 *<- eng(ineer) mode*
- *# make –j4*

- ***<결과 파일>***
- *out/target/product/msm8660_surf/*

  kernel
  boot.img
  persist.img.ext4
  ramdisk-recovery.img
  ramdisk.img
  recovery.img
  system.img.ext4
  userdata.img.ext4

1. Android ICS build 절차(2) – *envsetup.sh & make*

<단계1>

 # . build/envsetup.sh

1) toolchain path 등 개발 환경 설정

2) 기타 아래의 몇 가지 유용한 명령어 제공

m *- build tree의 top에서 부터 make(build) 시작*

➔ *모든 Android.mk 파일을 찾아서 하나로 만든 후, make를 돌림.*

mm *– 현재 directory 아래의 모든 module을 build*

mmm *– 파라미터로 지정한 directory 아래의 모든 module을 build*

➔ *mmm test*

➔ *mmm test snod* ←*test 디렉토리 build 후, system image 생성*

croot *– tree의 top으로 이동*

cgrep *– 모든 C/C++ file에 대해 특정 패턴을 grep하고자 할 때 사용함.*

jgrep *– 모든 java file에 대해 특정 패턴을 grep하고자 할 때 사용함.*

…

<단계2>

choosecombo

1) 바이너리를 upload할 대상을 선택한다(emulator, device 중에서)

2) Build mode(release, debug)를 선택한다.

3) Product model을 선택한다.

4) 구동 mode(user, userdebug, engineer)를 선택한다.

…

<단계3>

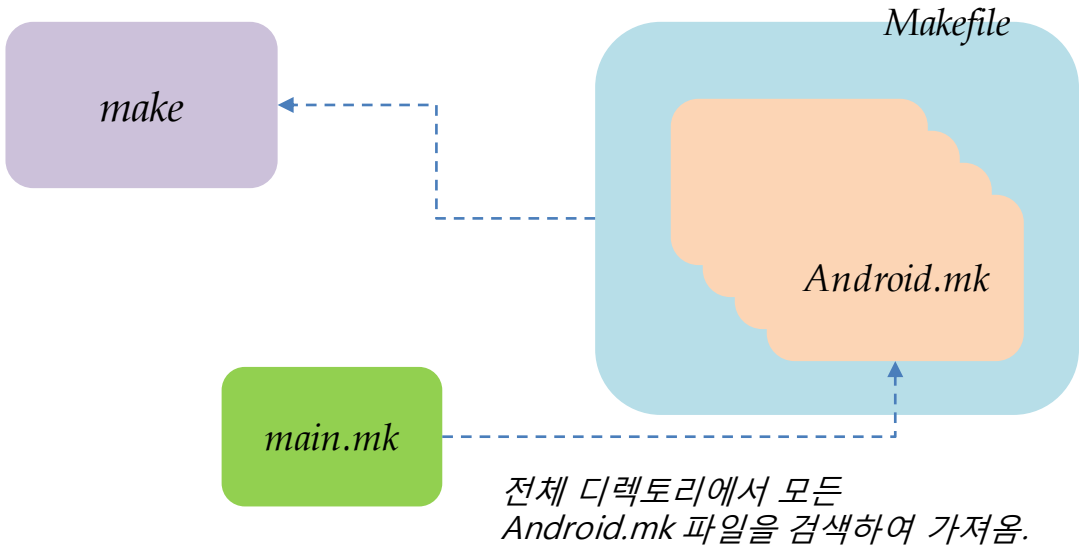
make ←***android/Makefile***

2. Android Build System Overview(1)

(*) android build system은 겉보기와는 달리, 일반적인 GNU make를 이용하여 구축되어 있다. 다만, 시스템 전체를 build하기 위한 관점에서 접근하고 있으며, 각각의 하위 모듈을 build하기 위해서 Android.mk라는 새로운 문법(sub makefile)을 정의하여 사용하고 있다.

<일반적인 GNU build system>

<Android GNU build system>

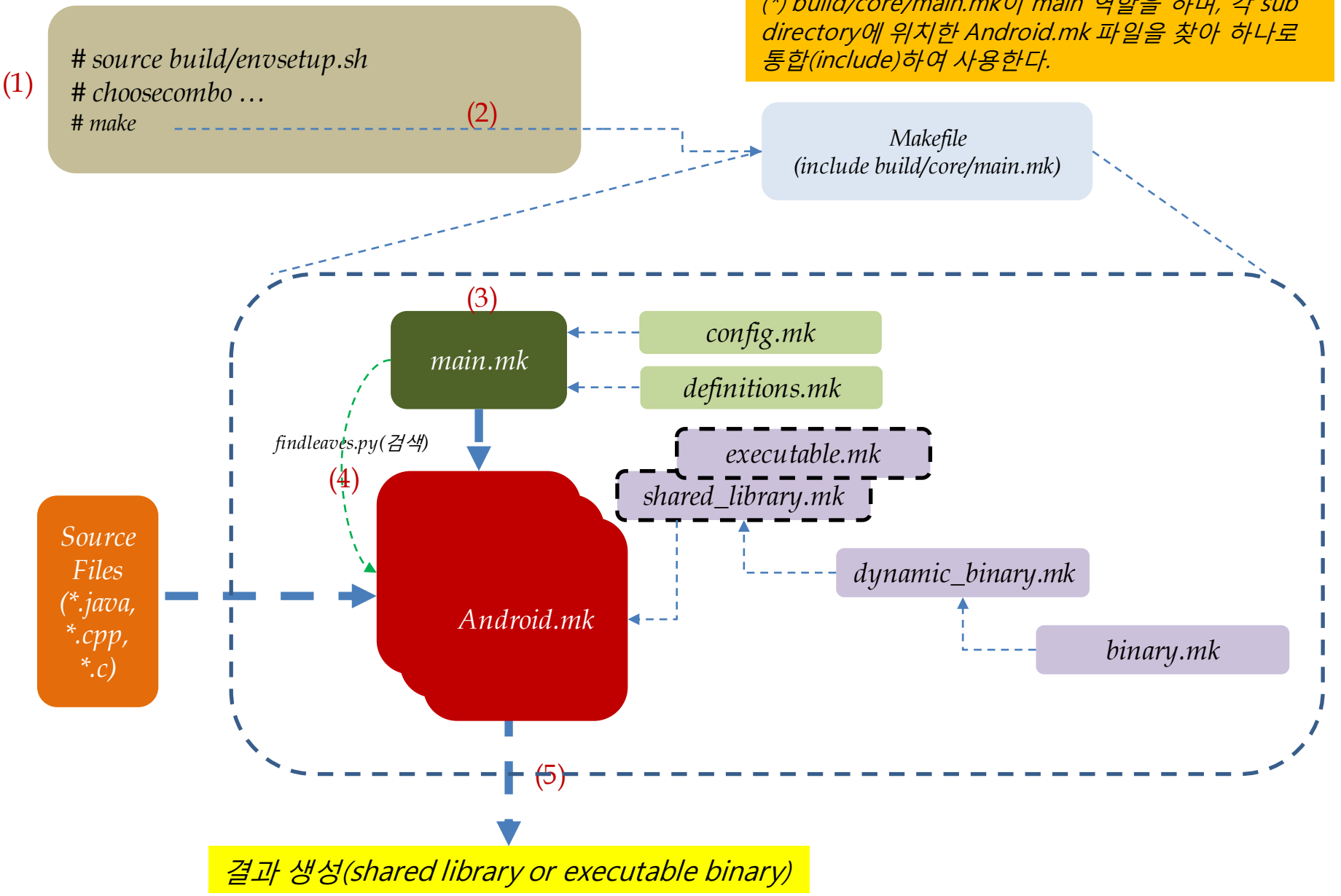


여러 개의 Android.mk 파일을 묶어 한 개의 거대한 Makefile 생성

전체 디렉토리에서 모든 Android.mk 파일을 검색하여 가져옴.

2. Android Build System Overview(2)

(*) Android.mk는 각각의 build 디렉토리에 있음.
(*) 아래에 기술되어 있는 모든 mk file은 build/core 디렉토리에 있음.
(*) build/core/main.mk이 main 역할을 하며, 각 sub directory에 위치한 Android.mk 파일을 찾아 하나로 통합(include)하여 사용한다.



2. Android Build System Overview(3) – *Android.mk 파일 문법*

<Android.mk 파일을 구성하는 field 설명>

➔ ***Android는 build를 위하여 GNU make를 사용하고 있으나, 앞서 기술한 것 처럼 자체 custom makefile을 운용하고 있다(그러나, 내부를 따라가 보면, 결국 기존 Makefile 형태임).***

1) **include $(CLEAR_VARS)**: build 관련 local 변수(아래 내용들)를 모두 clear해 줌.
   ➔ *build/core/clear_vars.mk 파일이 include될 것임.*
   ➔ *이 파일을 보면, Android.mk file에서 사용 가능한 local 변수를 확인할 수 있음.*

2) LOCAL variables

2-1) ***LOCAL_MODULE****: build하려는 module의 이름(결과 파일명)*
2-2) ***LOCAL_SRC_FILES****: build하려는 source 파일들*
2-3) ***LOCAL_STATIC_LIBRARIES****: 이 module(결과물)에 static하게 link하는 libraries*
2-4) ***LOCAL_SHARED_LIBRARIES****: 이 module에 link되는 shared libraries*
2-5) ***LOCAL_C_INCLUDES****: include file을 위한 path 지정(가령: $KERNEL_HEADERS)*
2-6) ***LOCAL_CFLAGS****: compiler에게 전달하는 추가 CFLAGS 지정*
2-7) ***LOCAL_LDFLAGS****: linker에게 전달하는 추가 LDFLAGS 지정*

3) Include BUILD rules

3-1) ***include $(BUILD_EXECUTABLE)***
   ➔ *실행파일을 build하는 rule이 추가됨(여기에서 build함 – 기존 Makefile format)*
   ➔ *build/core/executable.mk 파일이 include될 것임.*
3-2) ***include $(BUILD_SHARED_LIBRARY)****: shared library를 build하는 rule이 추가됨.*
3-3) ***include $(BUILD_STATIC_LIBRARY)****: static library를 build하는 rule이 추가됨.*
3-4) ***include $(BUILD_PREBUILT)****: prebuilt file을 복사하는 rule이 추가됨.*

2. Android Build System Overview(4) – *Android.mk 예(1)*

```
LOCAL_PATH:= $(call my-dir)

include $(CLEAR_VARS)
#Name of target to build
LOCAL_MODULE:= libmylibrary
#Source files to compile
LOCAL_SRC_FILES:= mysrcfile.c mysothersrcfile.c
#The shared libraries to link against
LOCAL_SHARED_LIBRARIES := libcutils
#No special headers needed
LOCAL_C_INCLUDES +=
#Prelink this library, also need to add it to the prelink map
LOCAL_PRELINK_MODULE := true
include $(BUILD_SHARED_LIBRARY)

#Clear variables and build the executable
include $(CLEAR_VARS)
LOCAL_MODULE:= myinfocmd
LOCAL_SRC_FILES:= mycmdsrcfile.c
include $(BUILD_EXECUTABLE)
```

libmylibrary.so를 만드는 예 (shared library)

myinfocmd 실행파일을 만드는 예

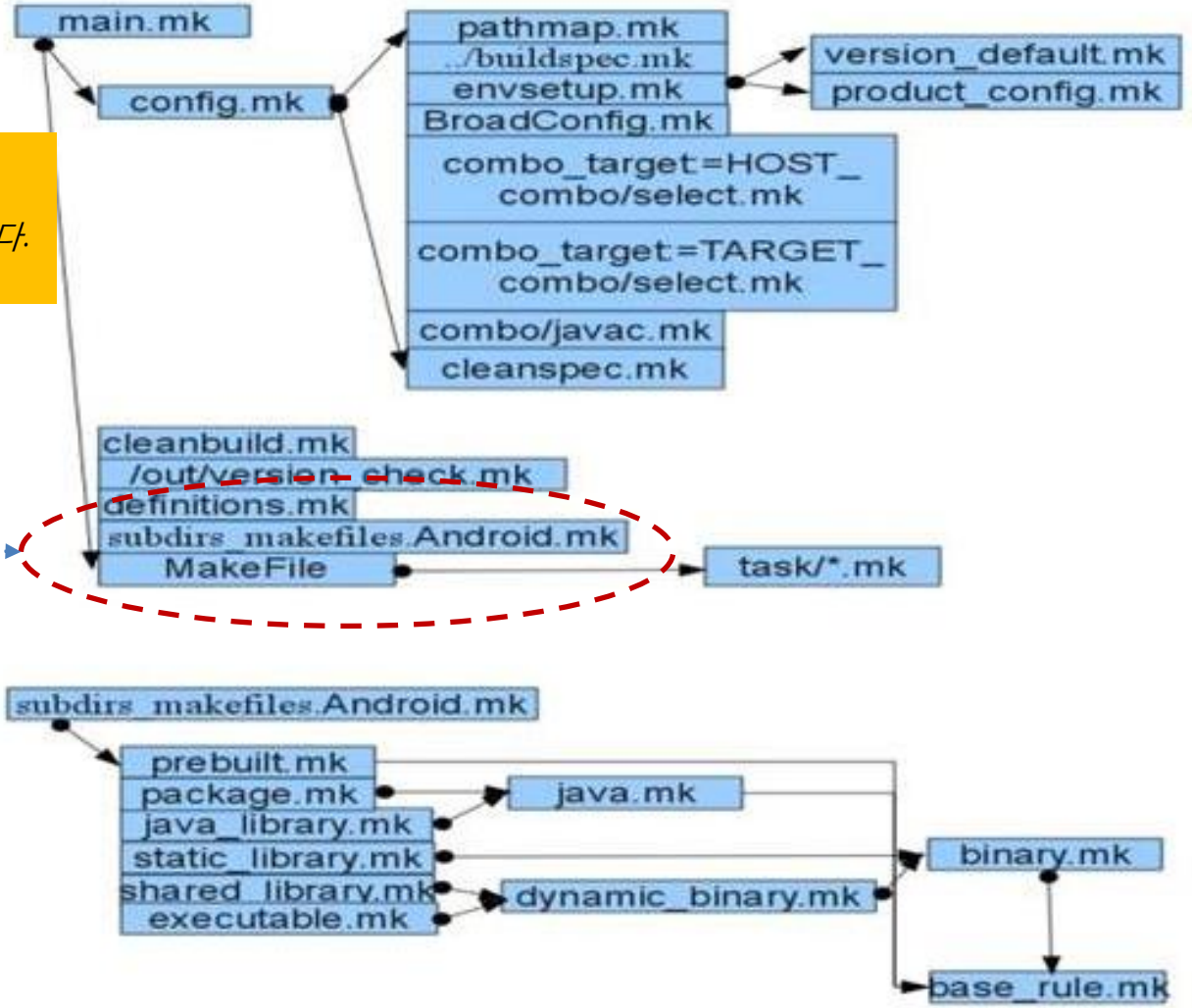
2. Android Build System Overview(4) – *Android.mk 예(2)*

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)

# Build all java files in the java subdirectory
LOCAL_SRC_FILES := $(call all-subdir-java-files)

# Name of the APK to build
LOCAL_PACKAGE_NAME := LocalPackage

LOCAL_CERTIFICATE := vendor/example/certs/app

# Tell it to build an APK
include $(BUILD_PACKAGE)
```

LocalPackage.apk 를 만드는 예

3. build/core/main.mk 파일 분석

build/tools/findleaves.py script를 사용하여 각 폴더의 첫번째 Android.mk를 모두 찾아서 include 시킨다.
*➔ 하나의 거대한 Makefile을 만듦^^*

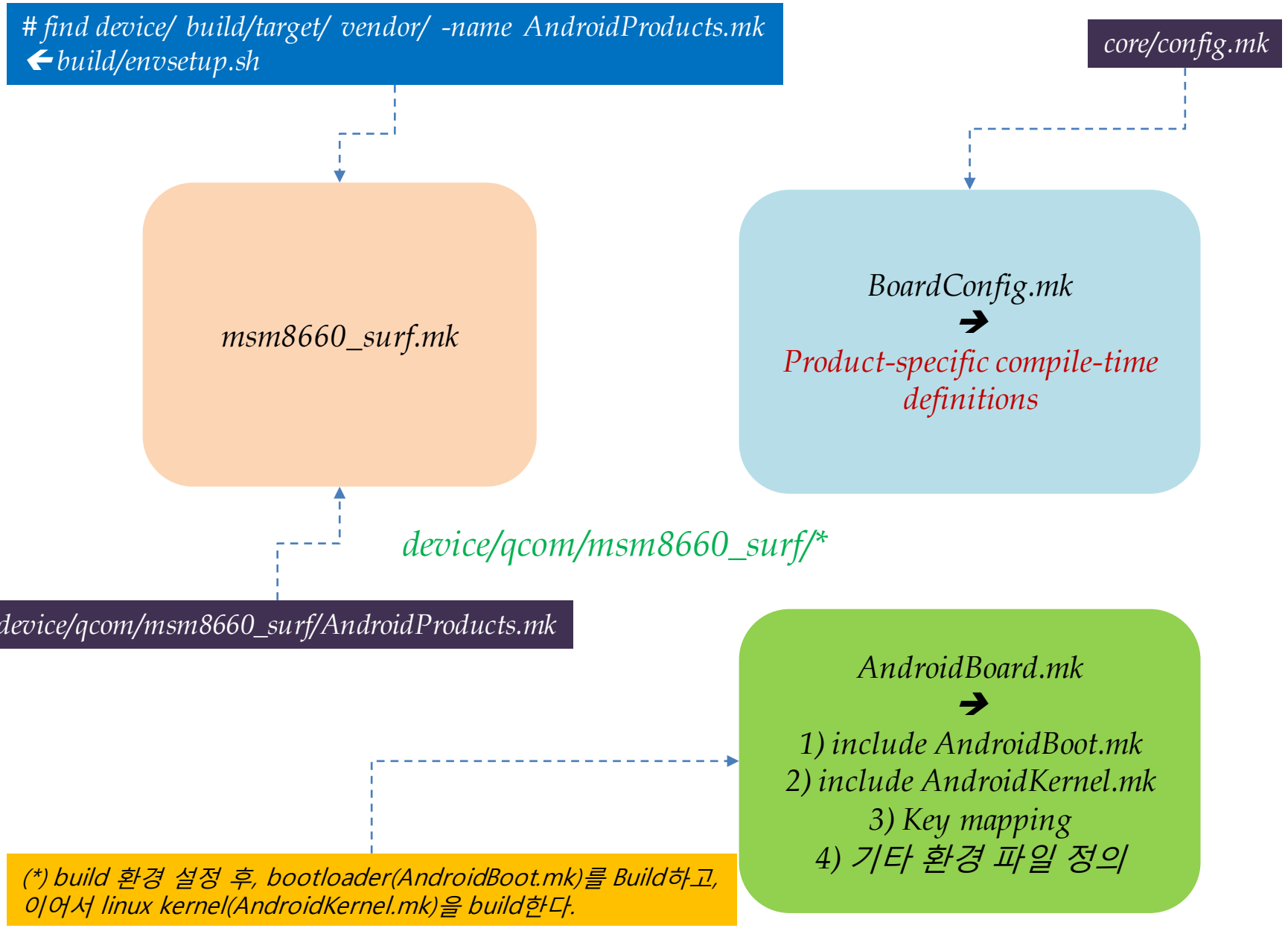


4. build/core/*.mk 주요 파일 정리: ***TODO***

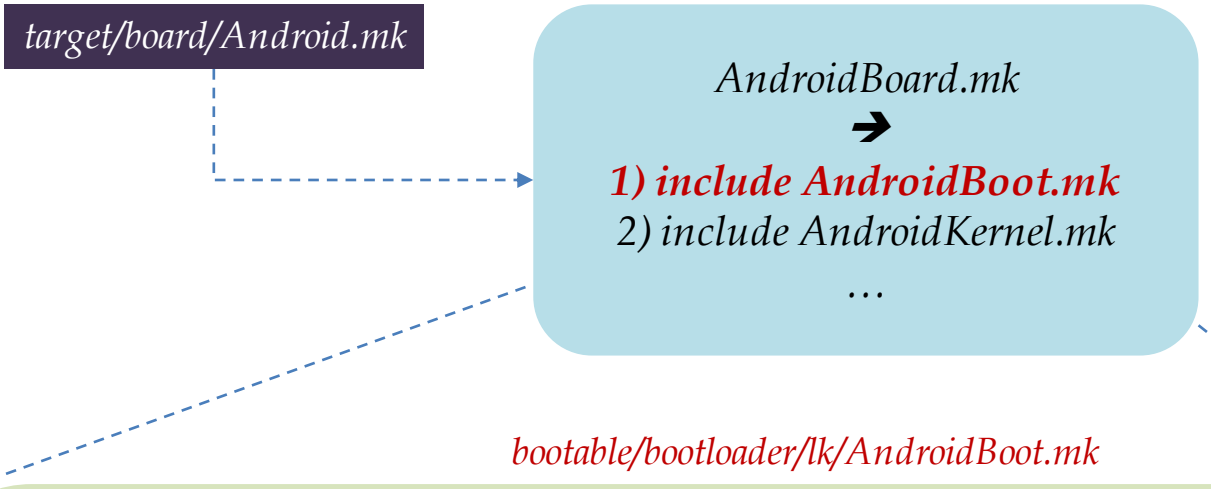
- *1) config.mk*
- *2) definitions.mk*
- *3) envsetup.mk*
- *4) base_rules.mk*
- *…*

5. Target Board 파일 분석

(*) 아래 그림은 Qualcomm chip의 경우에 해당함.



6. Bootloader Makefile 파일 분석(1)



bootable/bootloader/lk/AndroidBoot.mk

➔ bootloader build Makefile
➔ NAND flash 용 output file: appsboot.mbn
➔ eMMC 용 output file: emmc_appsboot.mbn
➔ lk/ 디렉토리(현재 디렉토리) 아래의 makefile 을 이용하여 build 진행함.
➔ makefile 에서는 아래의 파일을 include 하여 사용하고 있음.

```
include project/$(PROJECT).mk
include target/$(TARGET)/rules.mk
include target/$(TARGET)/tools/makefile
include platform/$(PLATFORM)/rules.mk
include arch/$(ARCH)/rules.mk
include platform/rules.mk
include target/rules.mk
include kernel/rules.mk
include dev/rules.mk
include app/rules.mk
include make/module.mk
```

<bootable/bootloader/lk 디렉토리 내용 개략 정리>

1) bootable/bootloader/lk/project/msm8660_surf.mk*
   => project makefile 위치

2) bootable/bootloader/lk/app/aboot*
   => aboot.c : flash/mmc에서 kernel image 읽어 kernel loading 및 start하는 코드
   => fastboot.c: usb cable이용한 fastboot 처리 코드
   => recovery.c: recovery 관련 코드

3) bootable/bootloader/lk/arch*
   => arm/ 디렉토리 아래에 실제 arm 환경에서의 boot 관련 코드 위치함.
   (assembly code 다수 존재함)
   => 아래 (5)의 main(kmain) code를 호출하는 부분 존재함.

4) bootable/bootloader/lk/dev
   => device driver(fbcon, keys, net, pmic, ssbi, usb)

5) bootloader/lk/kernel*
   => boot main(kmain) 이 위치함.
   => kernel의 특성에 해당하는 코드(mutex, thread, timer, event ..)

6) bootloader/lk/lib
   => lk에서 사용하는 library codes

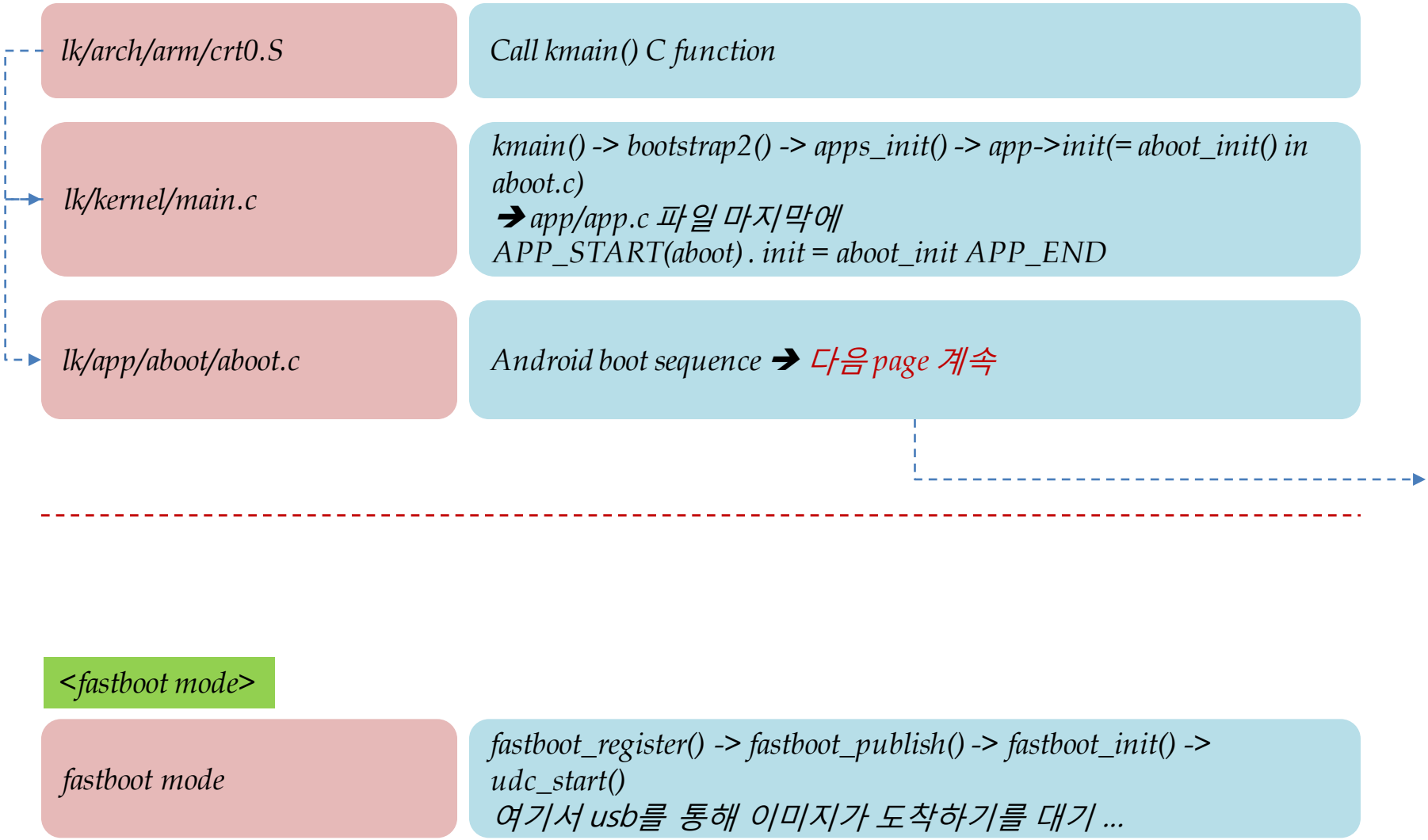
7) bootable/bootloader/lk/make

8) bootable/bootloader/lk/platform/msm8x60*
   => platform specific codes(acpuclock, gpio, hdmi, lcd panel, pmic, pmic battery alarm ...)

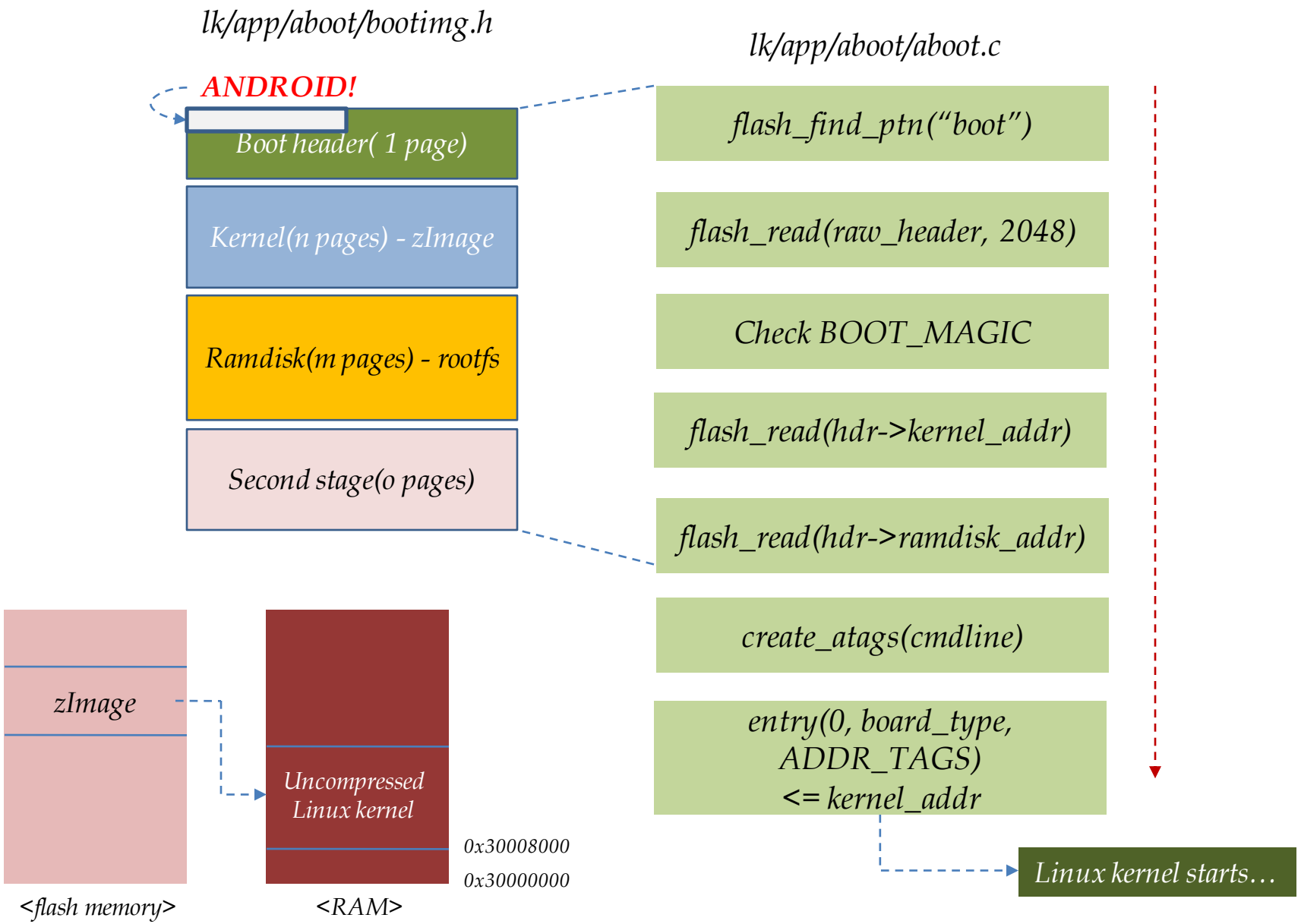
9) bootable/bootloader/lk/scripts
   => 몇가지 script 위치함

10) bootable/bootloader/lk/target*
   => msm8660_surf/ 디렉토리 아래에 target board와 관련한 몇가지 초기화 관련 코드 위치함.
   => target_init( ) 함수는 위의 (5)의 kmain( ) 함수에서 호출함.

6. Bootloader Makefile 파일 분석(2) – ***Boot Sequence***



6. Bootloader Makefile 파일 분석(2) – ***Boot Sequence***



(*) 참고 사항: lk bootloader size 조정하기/1

- How to increase the memory size allocated for the bootloader(Little Kernel) and relocate the base address of SMEM and the kernel accordingly on MSM7x27?
- When is this feature useful?

- - Sometimes it's required to add more feature which bring the increase of code and data size in the application processor's bootloader program, called LK(Little Kernel), but the memory size allocated for both the code and data space of the bootloader program is 1 MByte in default.
- - In this case, the memory size allocated for the bootloader(Little Kernel) needs to be resized to make more space to put the bootloader problem within the available memory.
- - Consequently, needs to relocate SMEM area and the kernel which are allocated contiguous to each other.

- **Solution:**

- - The memory size allocated for the bootloader program can be resized with small amount of code change. but, more changes are required because the SMEM space and the kernel memory are allocated contiguous to each other.
- And the SMEM library which was the blocking part(in changing the base address of it) up to recently is no more dependant on the base address of SMEM. Thus, the base address of SMEM can be relocated by customer by the simple change.
- If you want to know more technical details, refer to the source code file, smem_ext.c in AMSS/products/76XX/services/mproc/smem directory.

- Here below are the changes required.

- [Changes required from Modem code]

- 1. AMSS/products/76XX/build/ms/targtsncjnlym.h (use the file for your target configuration)

- #define SCL_APPS_BOOT_SIZE 0x0300000 // was 0x00100000 (1 MB), increase the memory size the allocated for the bootloader to 3 MB.
- #define SCL_APPS_CODE_BASE 0x00400000 // was 0x00200000, relocate the base address of kernel to 0x00400000
- #define SCL_APPS_TOTAL_SIZE 0x9400000 // was 0x9600000 (150 MB), not effective.
- #define SCL_SHARED_RAM_BASE 0x00300000 // was 0x00100000, relocate the base address of SMEM to 0x00300000

- ** Clean build is required for the change.

- [Changes required from Appl. code]

- 1. LINUX/android/bootable/bootloader/lk/platform/msm7k/include/platform/iomap.h

- #define MSM_SHARED_BASE 0x00300000 // was 0x00100000

- 2. LINUX/android/bootable/bootloader/lk/target/msm7627_surf/rules.mk

(*) 참고 사항: lk bootloader size 조정하기/2

- MEMSIZE := 0x00300000 # was 1MB (0x00100000)

  BASE_ADDR := 0x00400000 # was 0x00200000

  3. LINUX/android/kernel/arch/arm/mach-msm/io.c

  unsigned int msm_shared_ram_phys = 0x00300000; // was 0x00100000

  4. LINUX/android/kernel/arch/arm/mach-msm/Makefile.boot

  # MSM7x27
  zreladdr-$(CONFIG_ARCH_MSM7X27) := 0x00408000 // was 0x00208000
  params_phys-$(CONFIG_ARCH_MSM7X27) := 0x00400100 // was 0x00200100
  initrd_phys-$(CONFIG_ARCH_MSM7X27) := 0x0A000000

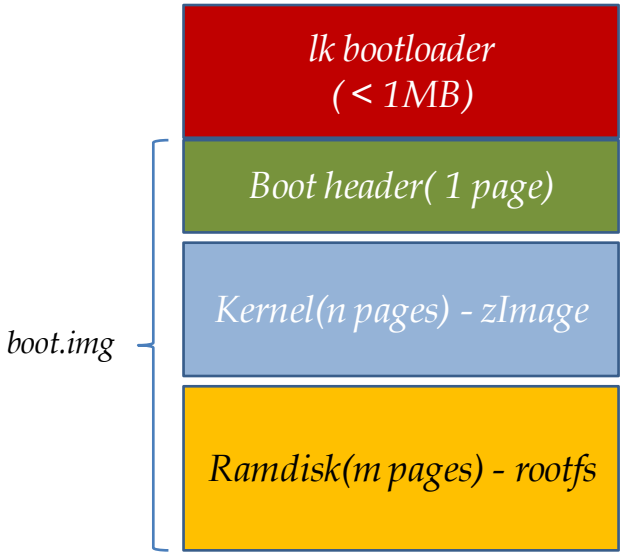
  5. LINUX/android/kernel/arch/arm/configs/msm7627-perf_defconfig

  CONFIG_PHYS_OFFSET=0x00400000 // was 0x00200000

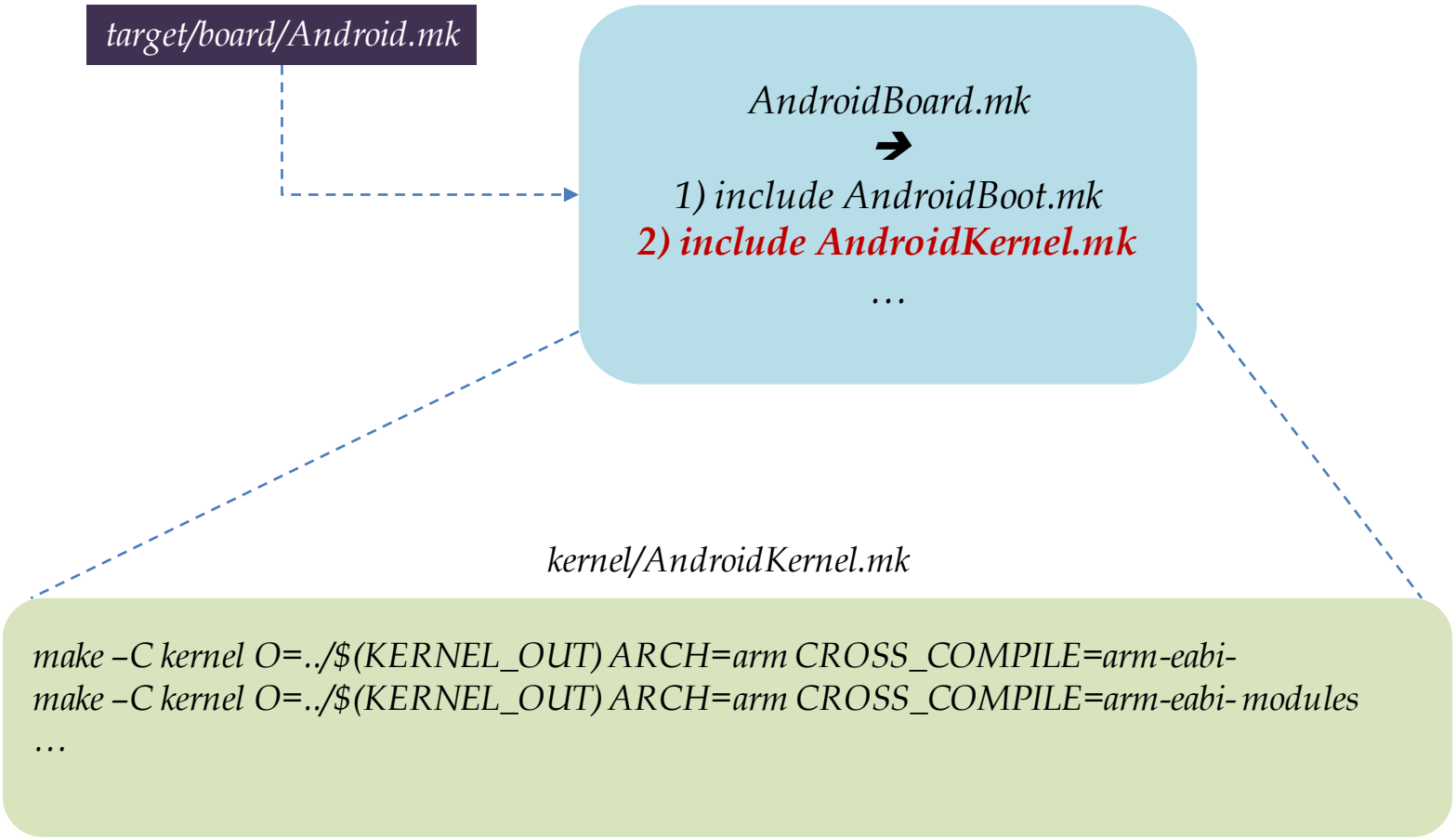
  6. LINUX/android/vendor/qcom/msm7627_surf/BoardConfig.mk

  BOARD_KERNEL_BASE := 0x00400000 // was 0x00200000

  BOARD_KERNEL_CMDLINE := mem=211M console=ttyDCC0 androidboot.hardware=qcom // was mem=213M, because the bootloader



7. Kernel Makefile 파일 분석(1)

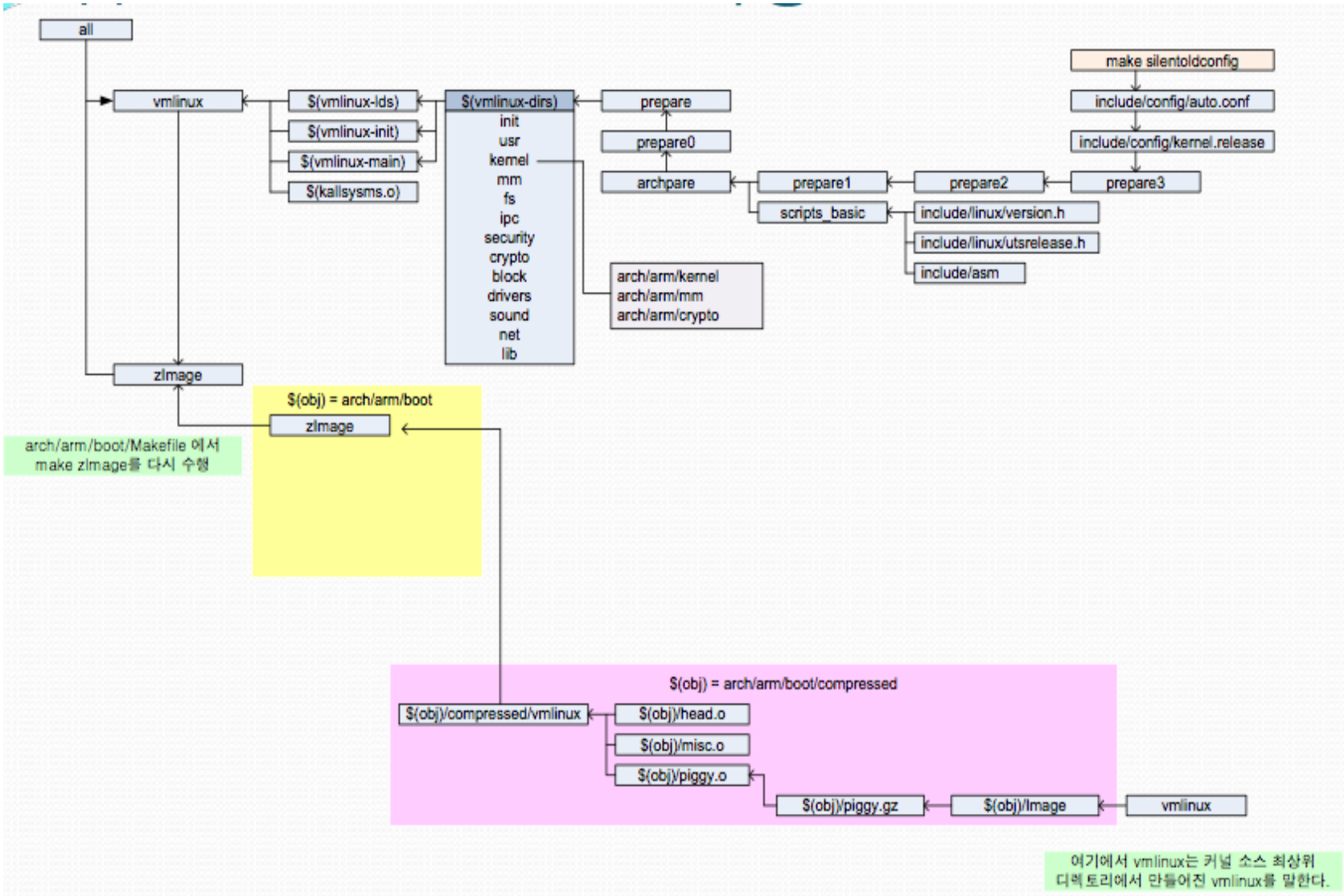


7. Kernel Makefile 파일 분석(2) – *kernel build 과정*

```
CHK     include/linux/version.h
UPD     include/linux/version.h
Generating include/asm-arm/mach-types.h
CHK     include/linux/utsrelease.h
UPD     include/linux/utsrelease.h
SYMLINK include/asm -> include/asm-arm


CC      kernel/bounds.s
GEN     include/linux/bounds.h
CC      arch/arm/kernel/asm-offsets.s
.
. <hundreds of lines of output omitted here>
.
LD      vmlinux
SYSMAP  System.map
SYSMAP  .tmp_System.map
OBJCOPY arch/arm/boot/Image
Kernel: arch/arm/boot/Image is ready
AS      arch/arm/boot/compressed/head.o
GZIP    arch/arm/boot/compressed/piggy.gz
AS      arch/arm/boot/compressed/piggy.o
CC      arch/arm/boot/compressed/misc.o
[redacted]
AS      arch/arm/boot/compressed/big-endian.o
LD      arch/arm/boot/compressed/vmlinux
OBJCOPY arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
```

(*) vmlinux, System.map, Image, bootstrap loader, zImage 등의 파일 생성 과정을 확인할 수 있음.

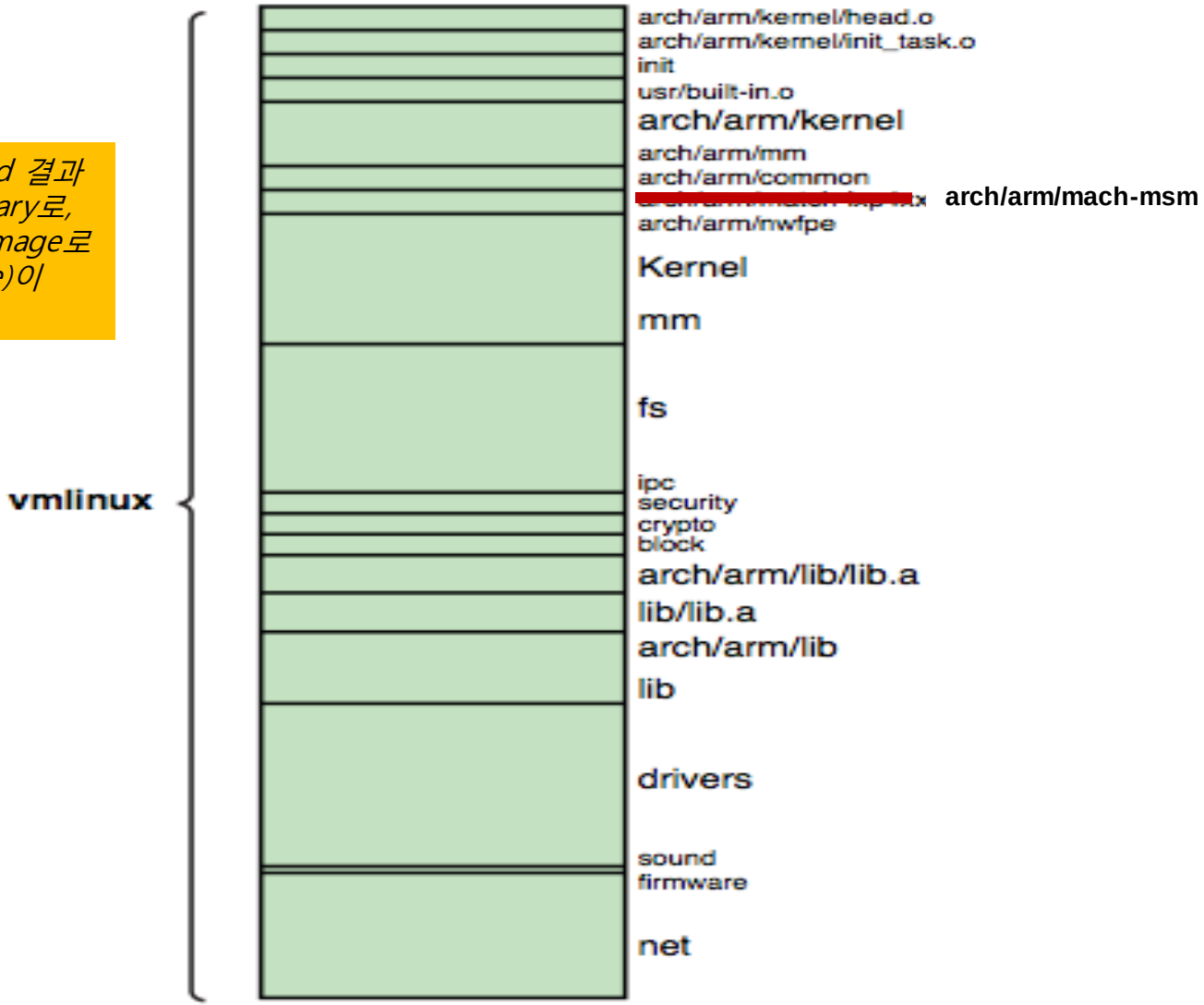
7. Kernel Makefile 파일 분석(3) – *Makefile 계층도*



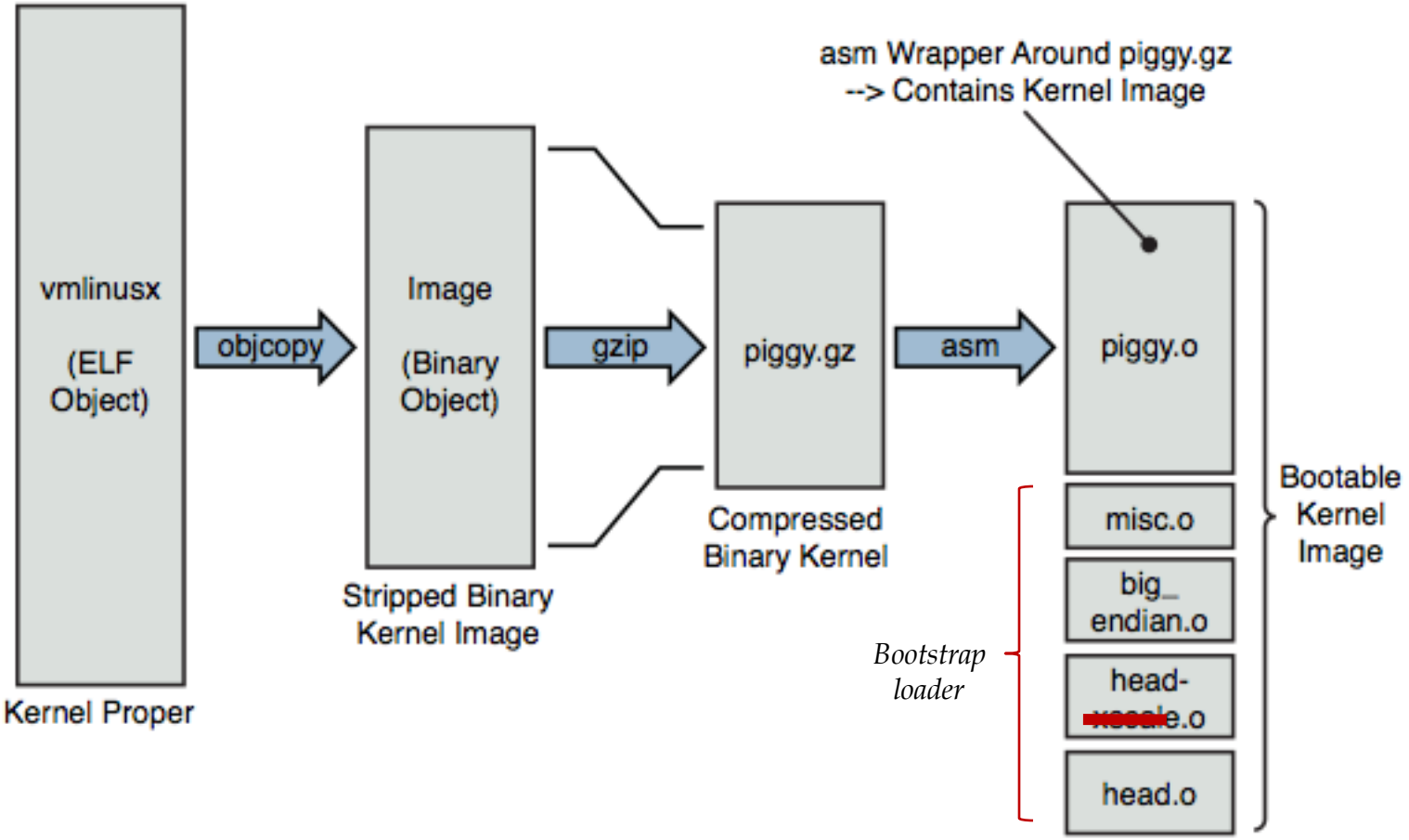
(*) 위 그림은 인터넷에서 복사한 것임[참고 문서 2]

7. Kernel Makefile 파일 분석(4) – *vmlinux 구성*

(*) vmlinux는 kernel build 결과 생성되는 ELF 형식의 binary로, 실제 사용을 위해서는 zImage로 전환하는 과정(다음 page)이 필요하게 된다.

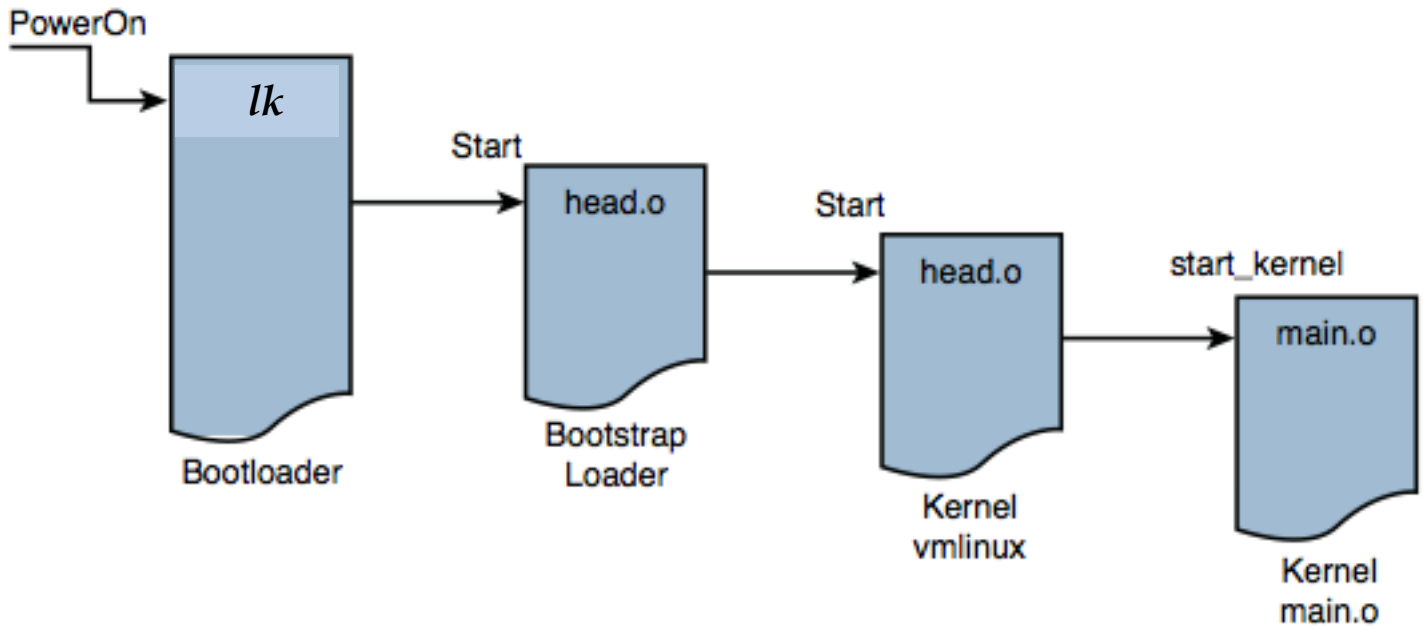


7. Kernel Makefile 파일 분석(5) – *zImage 생성 과정*



(*) objcopy 과정을 거쳐 debugging symbol이 제거되며, 압축(gzip)과 bootstrap loader 통합 과정을 거쳐 최종적으로 부팅 가능하는 zImage 파일이 생성된다.

7. Kernel Makefile 파일 분석(6) – *from bootloader to linux kernel*



(*) bootloader와 bootstrap loader는 전혀 다른 것임^^.
Bootstrap loader는 kernel의 압축을 풀고, memory에 재배치하는 역할을 담당함.

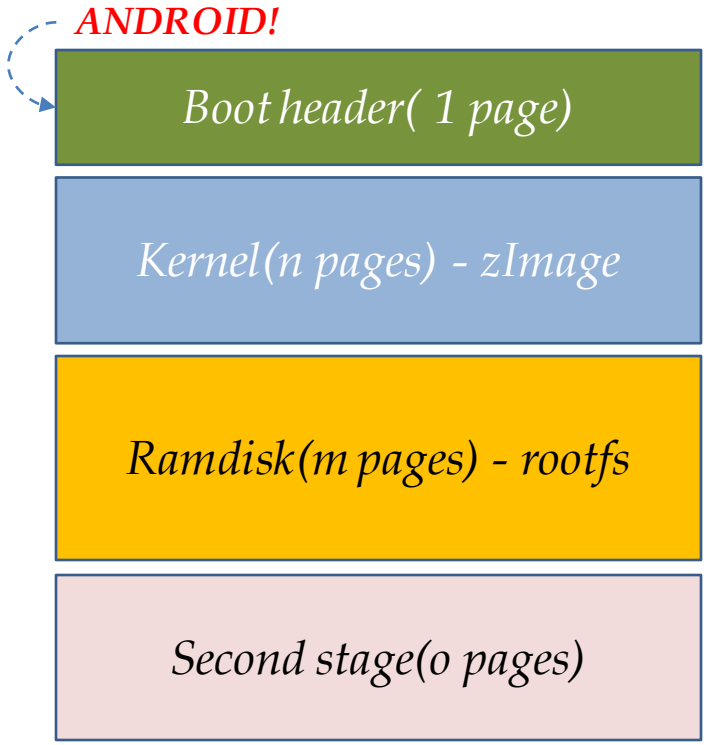
8. Kernel Build Example – ***Qualcomm chip***

(*) 주의 : kernel build 시 "make mrproper" 실행해야 한다는 에러가 발생할 경우에는 out/target/product/msm8660_surf/obj/KERNEL_OBJ 아래의 파일을 모두 삭제 후, 다시 build해 주면 됨.

```
# source build/envsetup.sh
# choosecombo 1 msm8660_surf 3

# make -C kernel O=../out/target/product/msm8660_surf/obj/KERNEL_OBJ ARCH=arm
CROSS_COMPILE=arm-eabi- msm8660_defconfig

# make -C kernel O=../out/target/product/msm8660_surf/obj/KERNEL_OBJ ARCH=arm
CROSS_COMPILE=arm-eabi- menuconfig

# make -C kernel O=../out/target/product/msm8660_surf/obj/KERNEL_OBJ ARCH=arm
CROSS_COMPILE=arm-eabi- zImage

# make -C kernel O=../out/target/product/msm8660_surf/obj/KERNEL_OBJ ARCH=arm
CROSS_COMPILE=arm-eabi- modules
```

8. Kernel Build Example - *boot.img 파일 해부하기(1)*

```
mkbootimg --cmdline "androidboot.hardware=qcom no_console_suspend=1"
--kernel zImage --ramdisk ramdisk.gz --base 0x40200000 -o boot.img
```



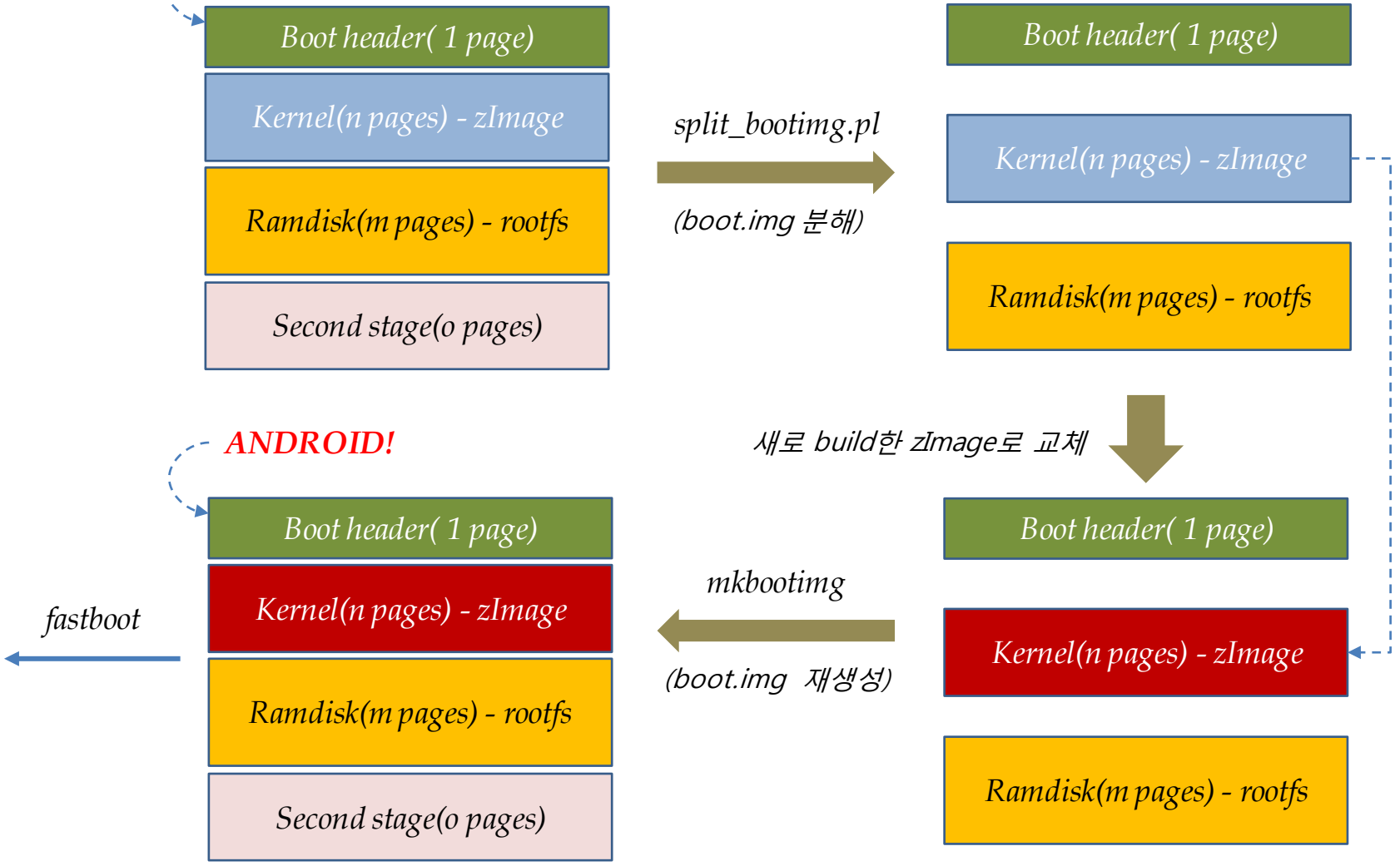
(*) 주의
위의 base 주소(kernel 시작 주소) 값은 device/qcom/msm8660_surf/BoardConfig.mk 파일의 **BOARD_KERNEL_BASE** 정보를 참조 !!!

n = (kernel_size + page_size – 1) / page_size
m = (ramdisk_size + page_size – 1) / page_size
o = (second_size + page_size – 1) / page_size

(Example)
Page size: 2048 (0x00000800)
Kernel size: 4915808 (0x004b0260)
Ramdisk size: 310311 (0x0004bc27)
Second size: 0 (0x00000000)

8. Kernel Build Example - *boot.img 파일 해부하기(2)*

[TIP] split_bootimg.pl 은 인터넷에서 구할 수 있음.
[TIP] 앞서 설명한 mkbootimg와 위에서 설명한 내용을 토대로, 새로 build한 kernel image를 적용한 새로운 boot.img를 만들 수 있게됨(build 시간을 단축하고자 할때)
[TIP] fastboot 명령을 사용하여 새로 만든 boot.img를 flash에 write하면 됨 ^^.



8. Kernel Build Example - *boot.img 파일 해부하기(3)*

- *<boot.img 파일 분해하기>*
- *# ./split_bootimg.pl boot.img*
- *-> boot header*
- *-> boot.img-kernel ← kernel*
- *-> boot.img-ramdisk.gz ← ramdisk root file system*

- *<boot.img 파일 재 생성하기>*
- *# mkbootimg --cmdline " androidboot.hardware=qcom no_console_suspend=1"*
- *--kernel **zImage** --ramdisk **ramdisk.gz** **--base 0x40200000** -o **boot.img***
- *➔ 앞서 설명한 것 처럼, 빨간색 표시 부분은 시스템마다 다르니, 주의 요망(잘못하면, 영영 부팅 안됨^^)*

- *<fastboot으로 boot.img write하기>*
- *# sudo adb reboot-bootloader ← fastboot mode로 전환(혹은 시스템에서 정의한 key 조합 선택하여)*
- *# sudo fastboot flash boot ./boot.img*

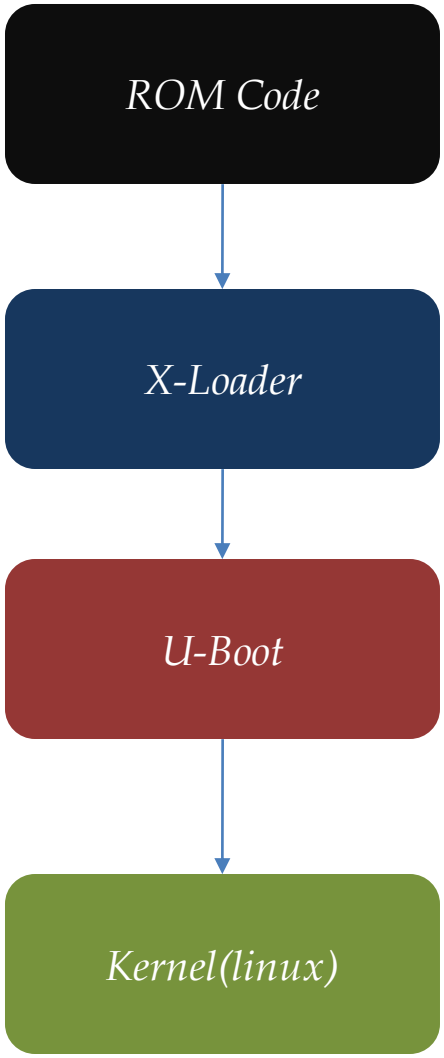
- *-- -- -- -- --*
- *<기타 참고 사항1: ramdisk rootfs 파일 해부하기>*
- *# gzip -d boot.img-ramdisk.gz*
- *# cpio -i < boot.img-ramdisk*
  - *← 현재 디렉토리에 ramdisk file system을 구성하는 파일이 풀리게 됨.*

- *<기타 참고 사항2: 새로운 ramdisk file 만들기 – init.rc, init.qcom.rc등을 수정 후 테스트 시>*
- # find . | cpio –o –H newc | gzip > ../newramdisk.cpio.gz
  - *<= 위에서 cpio로 파일을 풀어둔 디렉토리에서 명령을 실행.*

부록. TI OMAP Boot Sequence

: x-loader, u-boot, kernel

1. Boot Sequence(1)



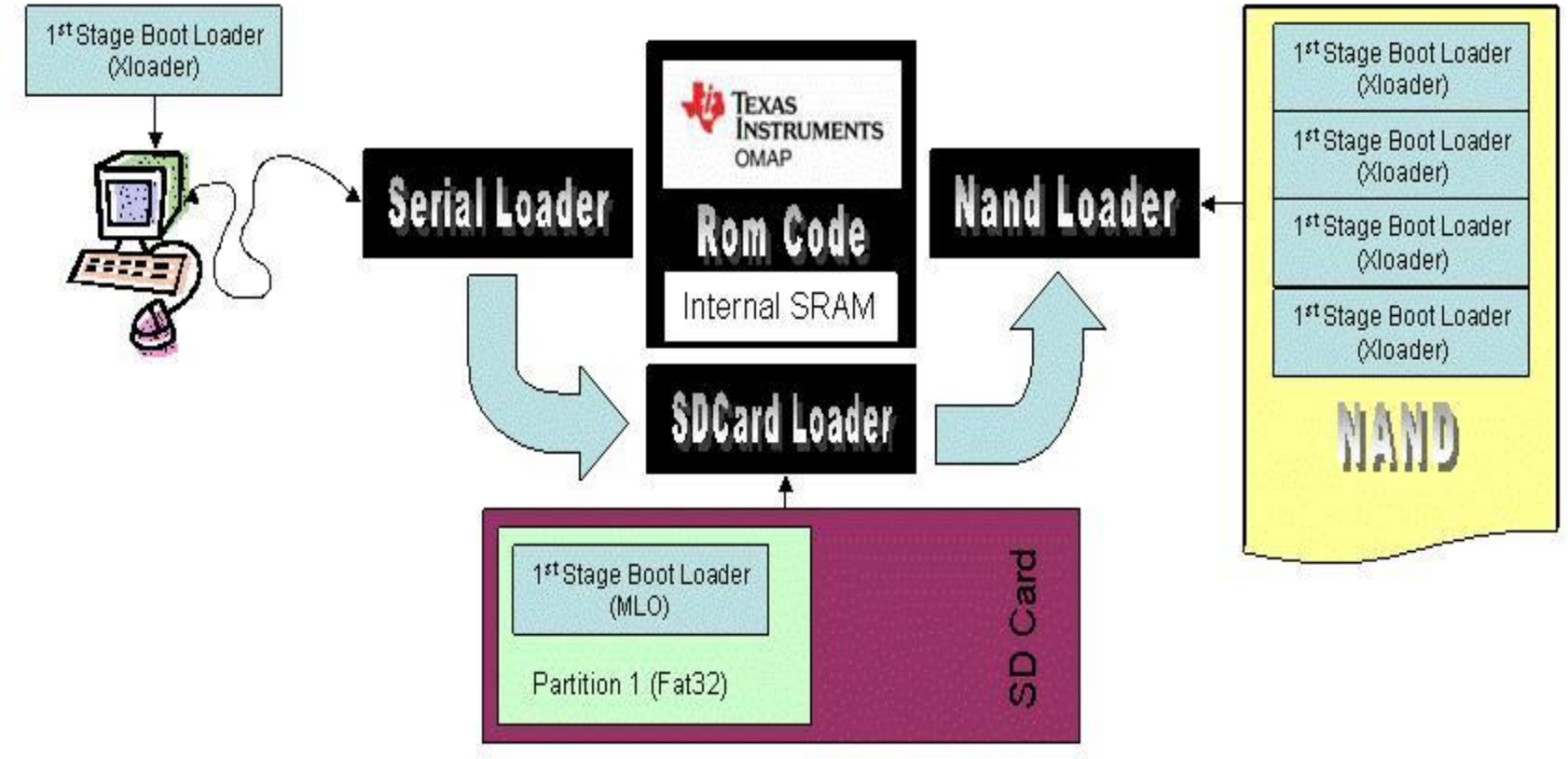
1) Performs minimal clocks, memories and peripheral configuration.
2) Searches the booting devices for a valid booting image.
3) Loads the X-Loader into SRAM and executes it.

1) Sets up the pin muxing.
2) Initializes clocks and memory.
3) Loads the U-Boot into SDRAM and executes it.

1) Performs some additional platform initialization.
2) Sets the boot arguments.
3) Passes control to the kernel image.

1) Decompresses the kernel into SDRAM.
2) Sets up peripherals, such as LCD, HDMI, I2C bus, USB, SD card, Audio, etc.
3) Mounts the Linux file system that contains all userspace libraries/ applications.

1. Boot Sequence(2)



(*) 위의 그림은 omappedia site에서 복사해온 것임.

1. Boot Sequence(3)

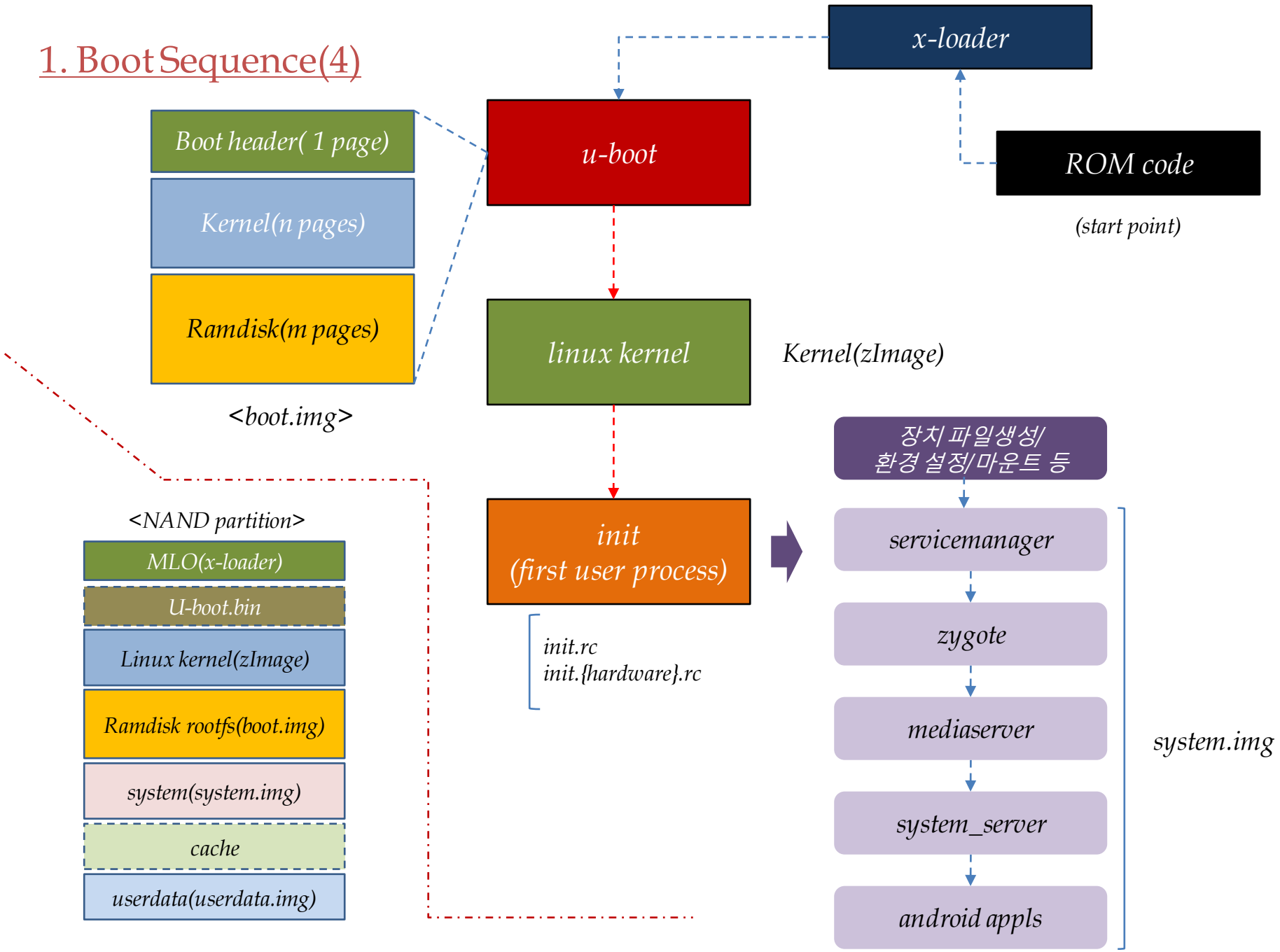
- ***1) ROM code -> X-Loader -> U-Boot -> Kernel***
  - Internal ROM code : Serial, SDCard, eMMC, NAND, USB 중 하나로 부터 부팅 시작
  - 각 장치로 부터 x-loader를 찾는 순서는 gpio configuration pins에 의해 결정됨.
  - 이 gpio configuration pin이 SysBoot임
  - SysBoot
    - 물리주소 0x480022f0
    - 예를 들어, SysBoot pin이 1) seria(UART3) 2) SD card(MMC1) 3) NAND로 설정되어 있다면, ROM code는 1) 2) 3) 순서로 x-loader를 찾으려 할 것임.

- *2) See the http://omappedia.org/wiki/Bootloader_Project for more information about booting sequence.*

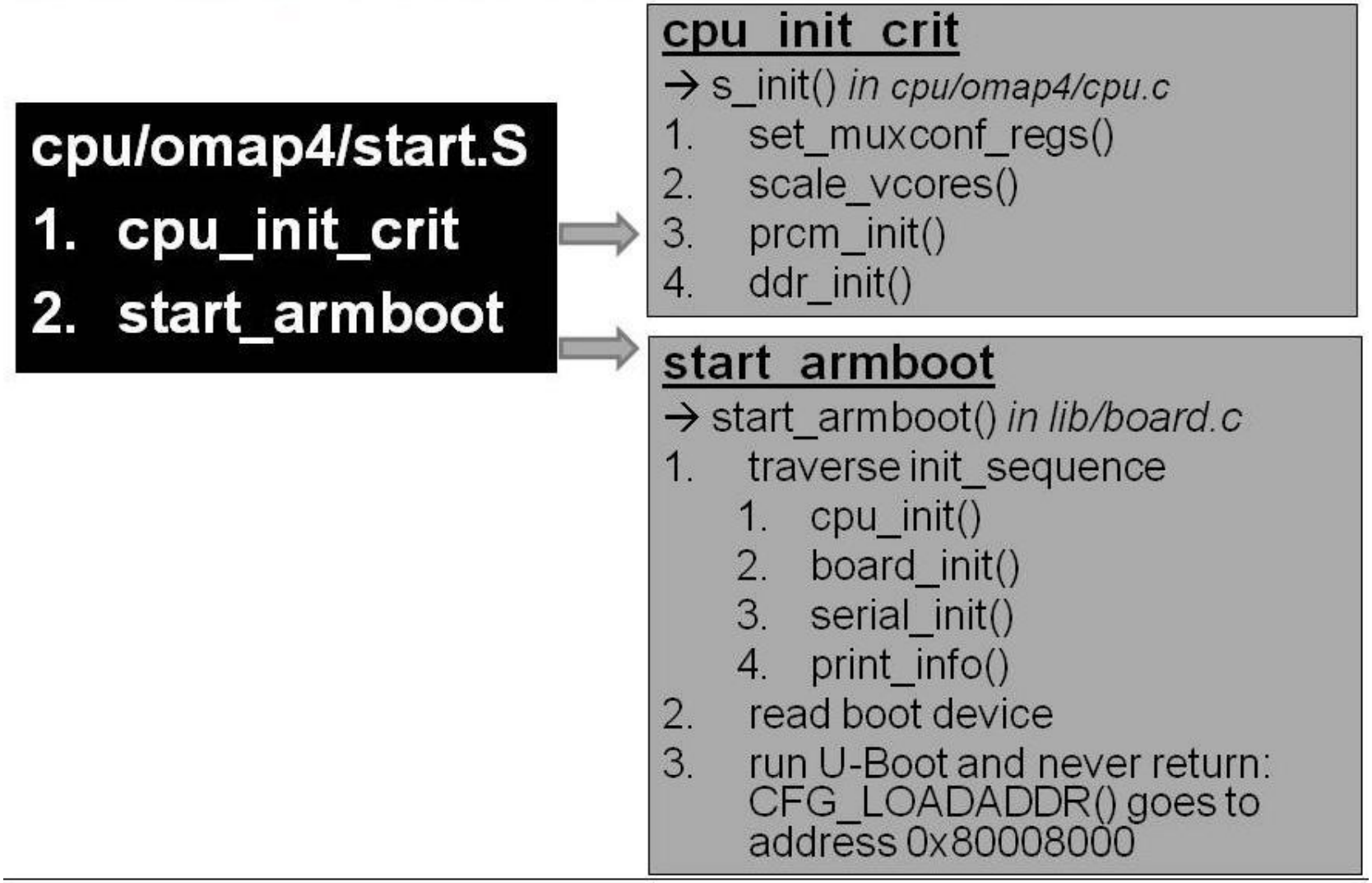
- *3) See the http://omappedia.org/wiki/4AI.1.5_OMAP4_Icecream_Sandwich_Release_Notes for related source codes.*

  ➔ *OMAP Blaze board 용 ICS source download 가능함 !*

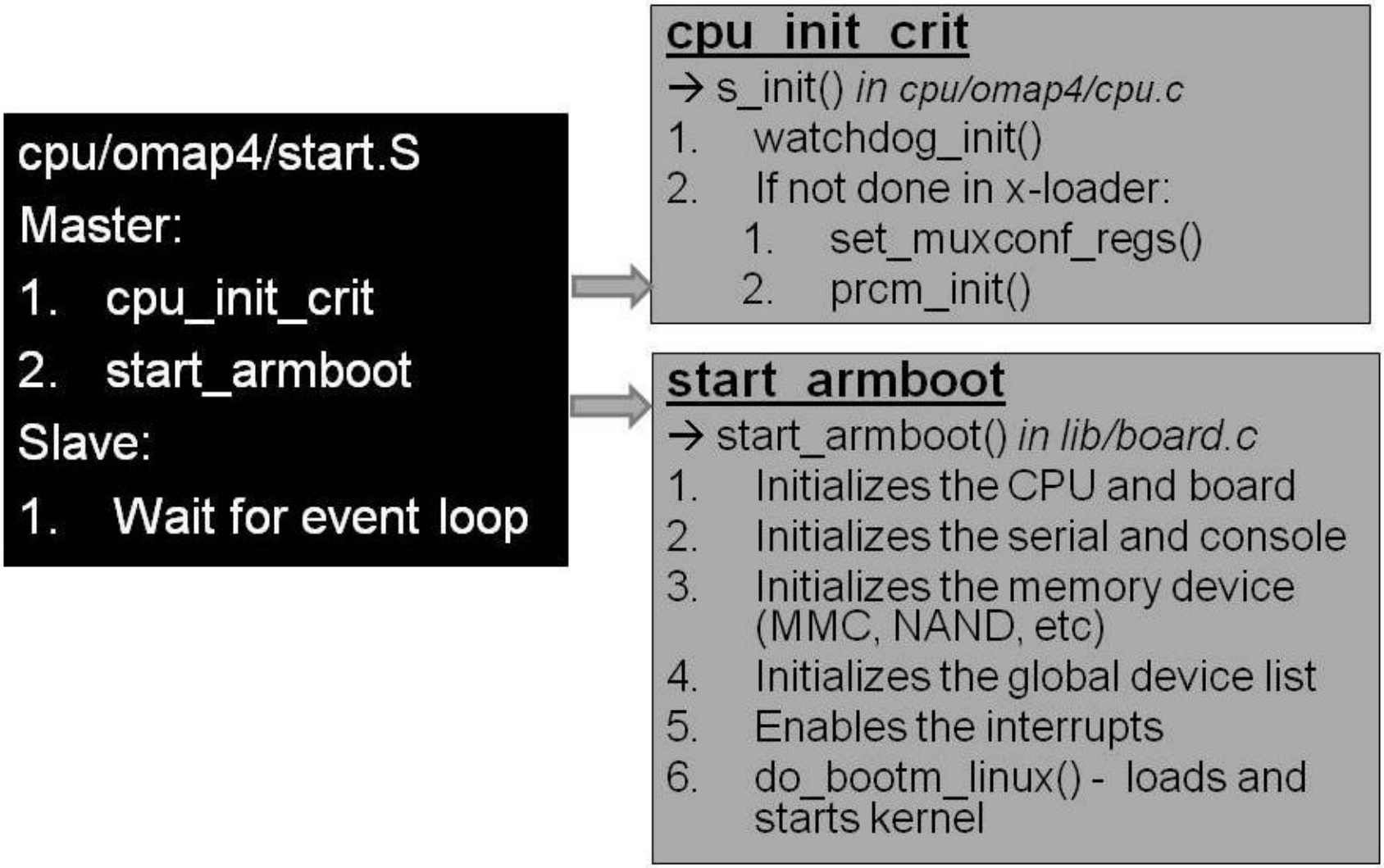
1. Boot Sequence(4)



2. X-Loader Overview



3. U-Boot Overview



References

- *1) Embedded Linux Primer 2nd edition … [Christopher Hallinan]*
- *2) 망고100 보드로 놀아보자 …. [EmbeddedCrazyboys]*
- *3) Some Internet Articles …*

Thanks a lot !

SlowBoot