Based on RPi & Embedded Linux Primer 2nd Edition

Embedded Linux

(초심자 교육 자료)

Date: 2016. 11. 21 ~
Author: Slowboot
(chunghan.yi@gmail.com)
Doc Revision:0.8

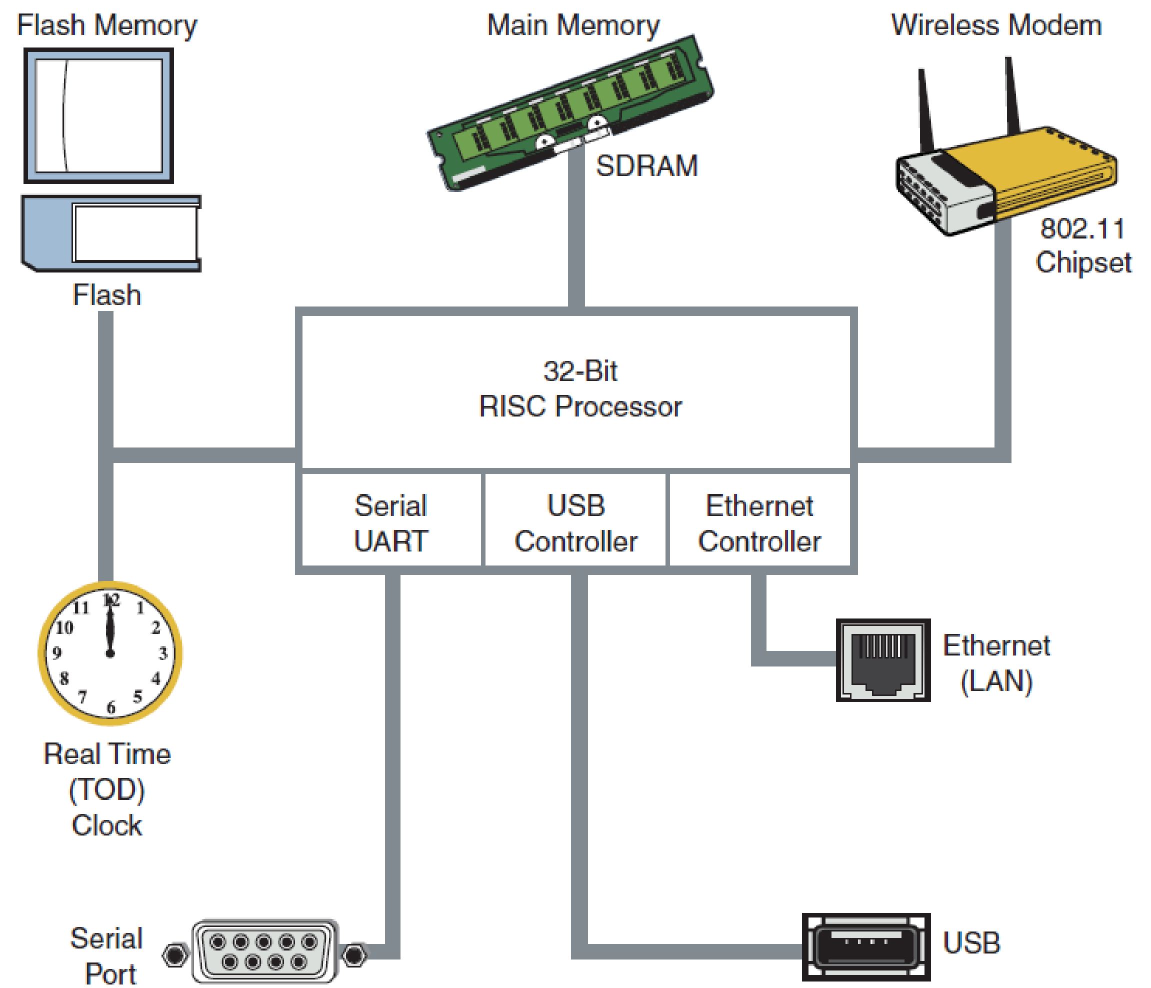
목차

Chapter 1 – 3

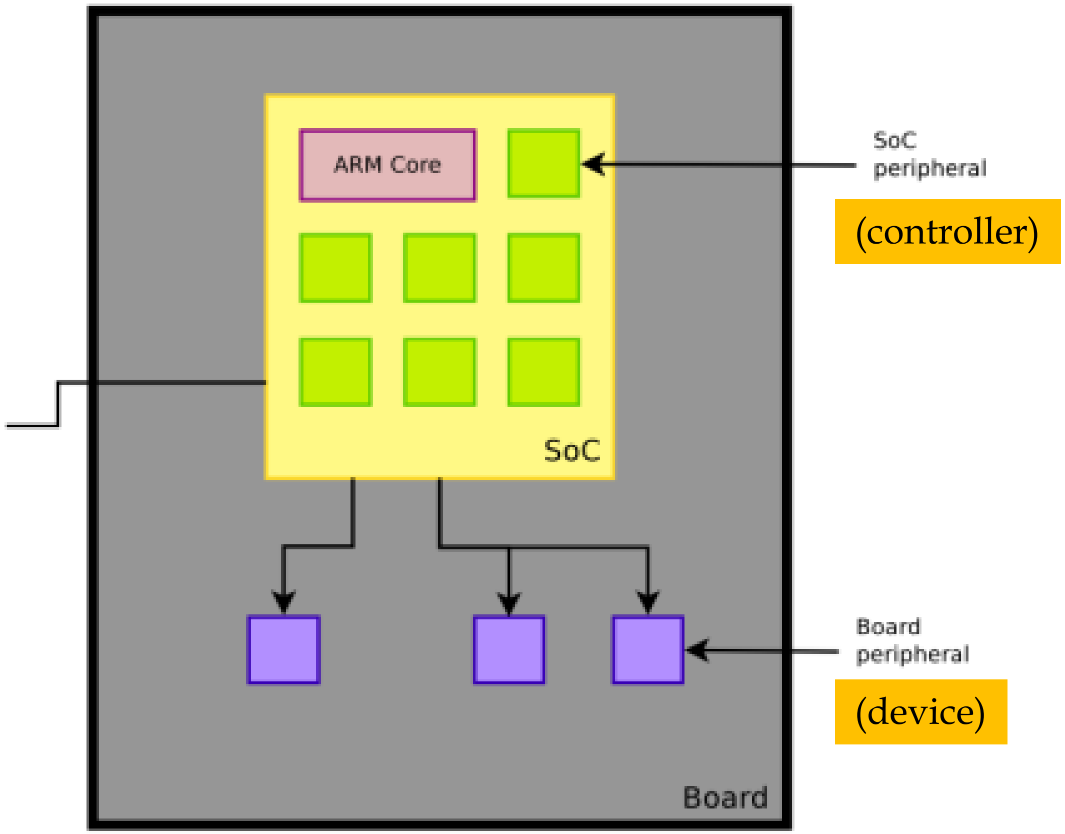
: Embedded System 개요

1. 일반적인 Embedded System Block도(1)

NOR
NAND
eMMC
SD Card
…



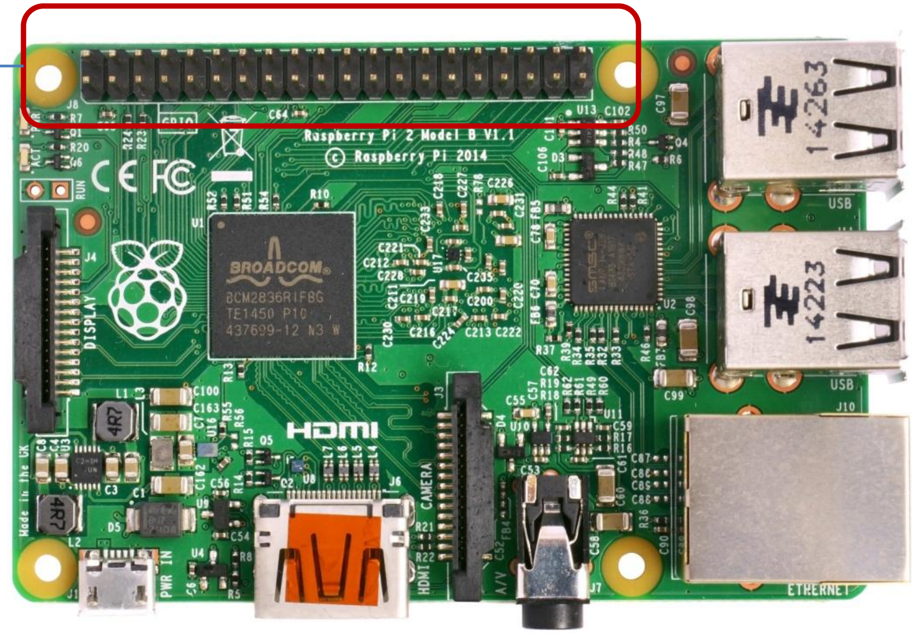
1. 일반적인 Embedded System Block도(2) – ARM SoC & Board



2. Raspberry Pi 2(1) – 보드 & GPIO 헤더

Raspberry Pi2 GPIO Header

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I²C)	DC Power 5v	04
05	GPIO03 (SCL1 , I²C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)	(I²C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40



I2C
SPI
UART
GPIO
DC Power
Ground

1) 900MHz quad-core ARM Cortex-A7 Broadcom BCM2836 CPU

2) Videocore IV GPU

3) 1GB RAM

2. Raspberry Pi 2(2) – 회로도 분석

- 회로도 Review

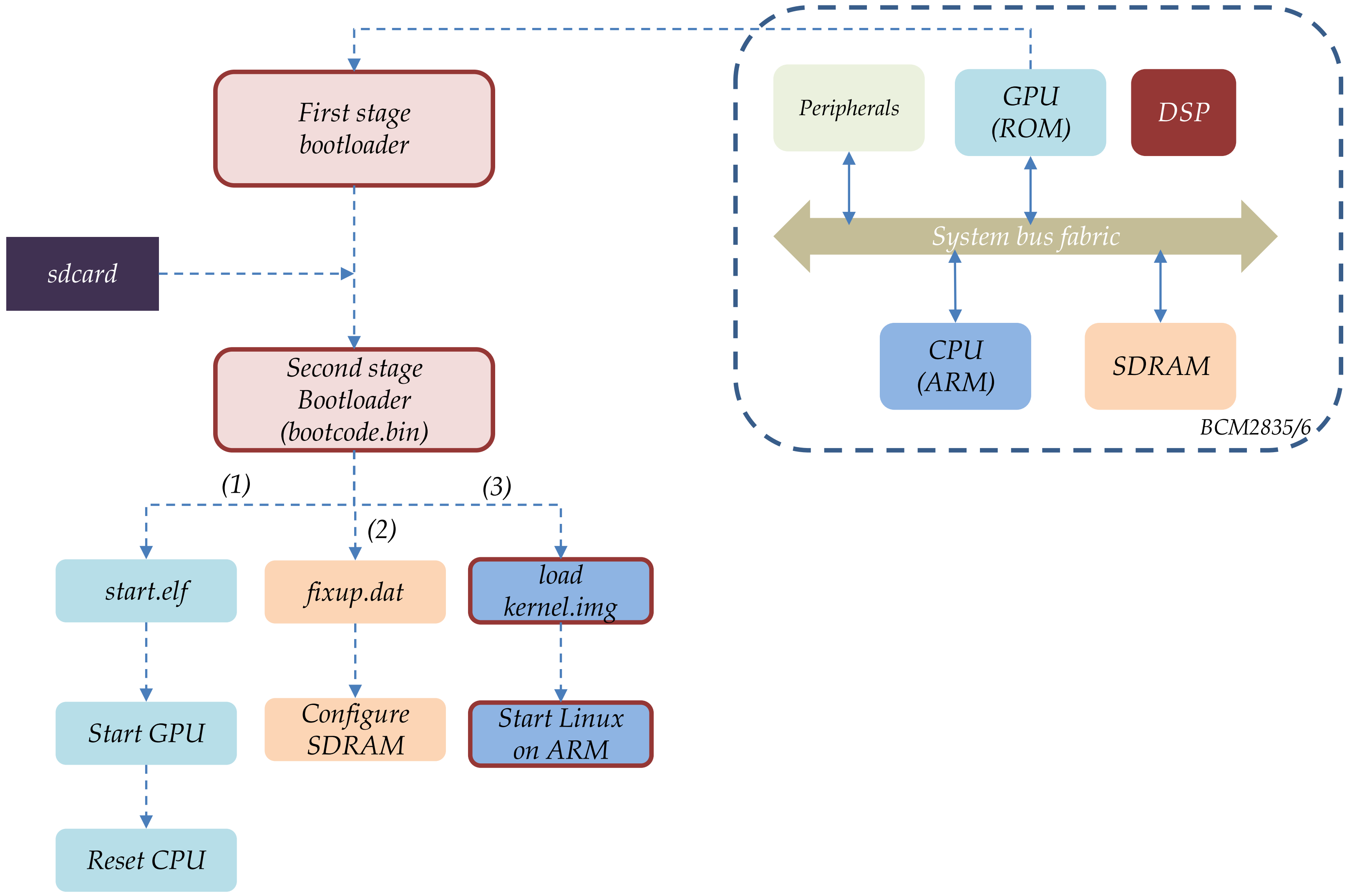
2. Raspberry Pi 2(3) - Buildroot 결과 분석

- output/images/

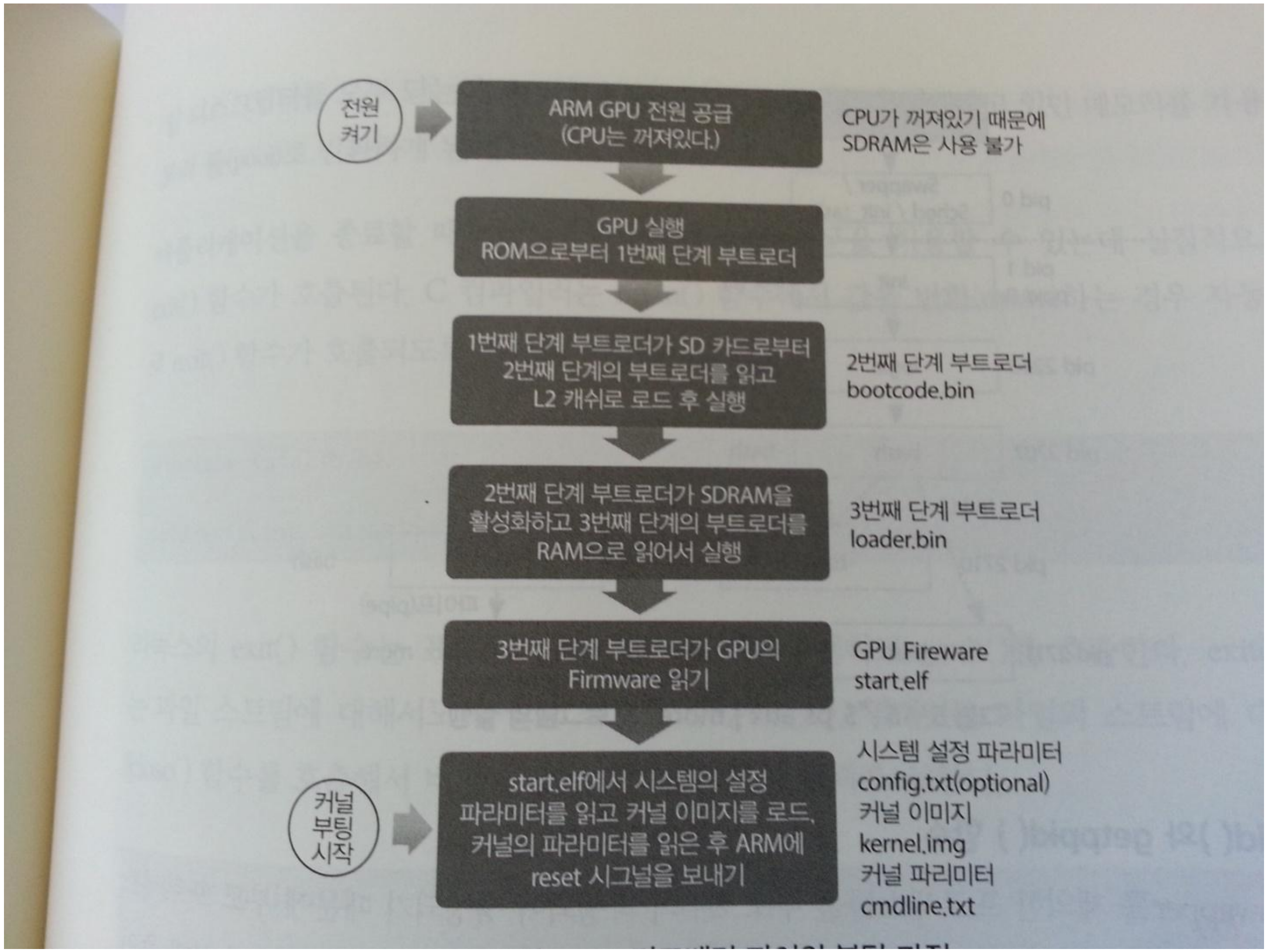
```
+-- rootfs.tar
+-- rpi-firmware/
|   +-- bootcode.bin
|   +-- config.txt
|   +-- fixup.dat
|   +-- start.elf
+-- bcm2708-rpi-b.dtb
+-- bcm2708-rpi-b-plus.dtb
+-- bcm2709-rpi-2-b.dtb
+-- zImage
```

(*) buildroot nconfig 내용 훑어 보자.
➔ 디렉토리 내용도 훑어 보자.
(*) RPi login하여 / 디렉토리 내용 확인해 보자.

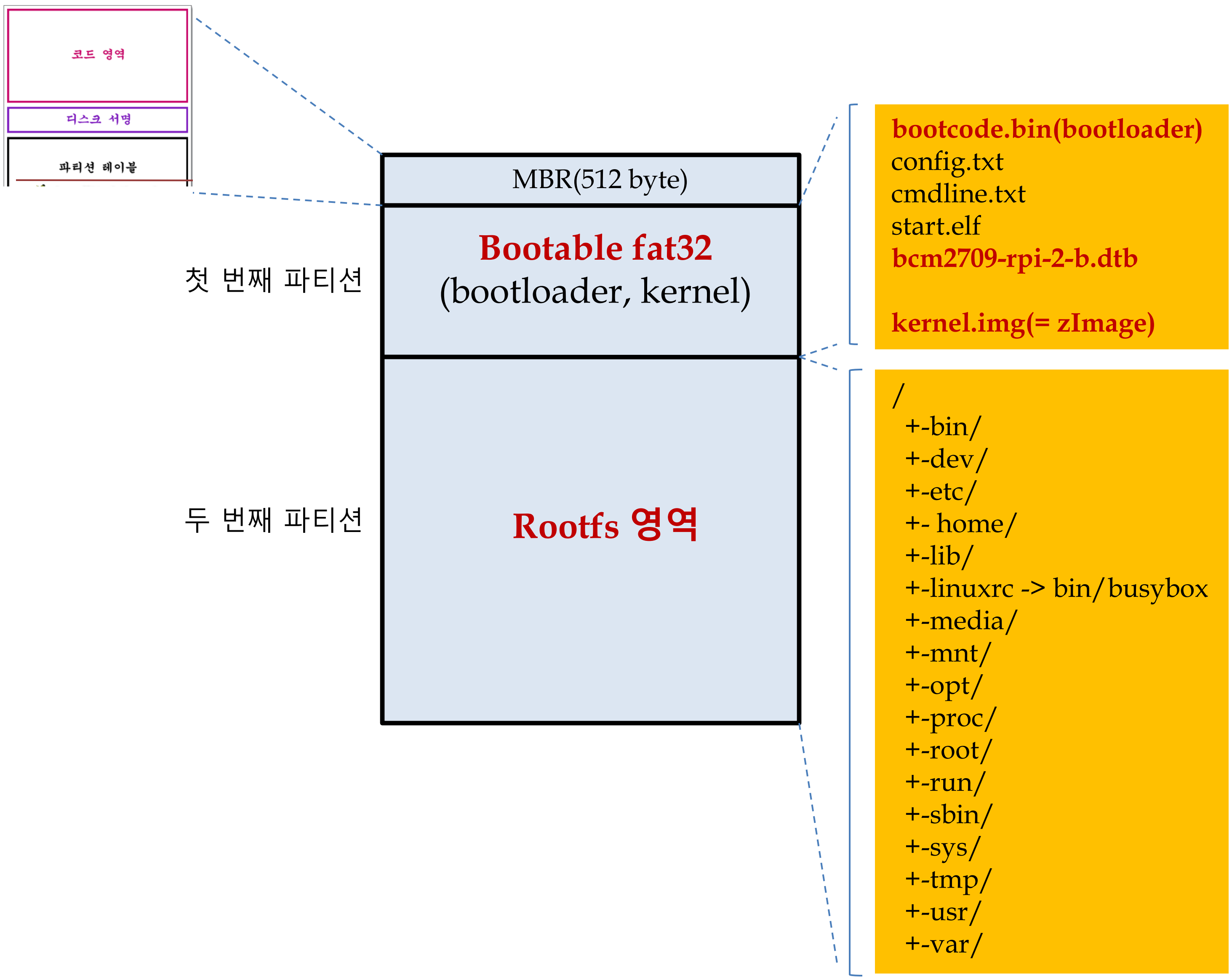
2. Raspberry Pi 2(4) - Booting 순서(1)



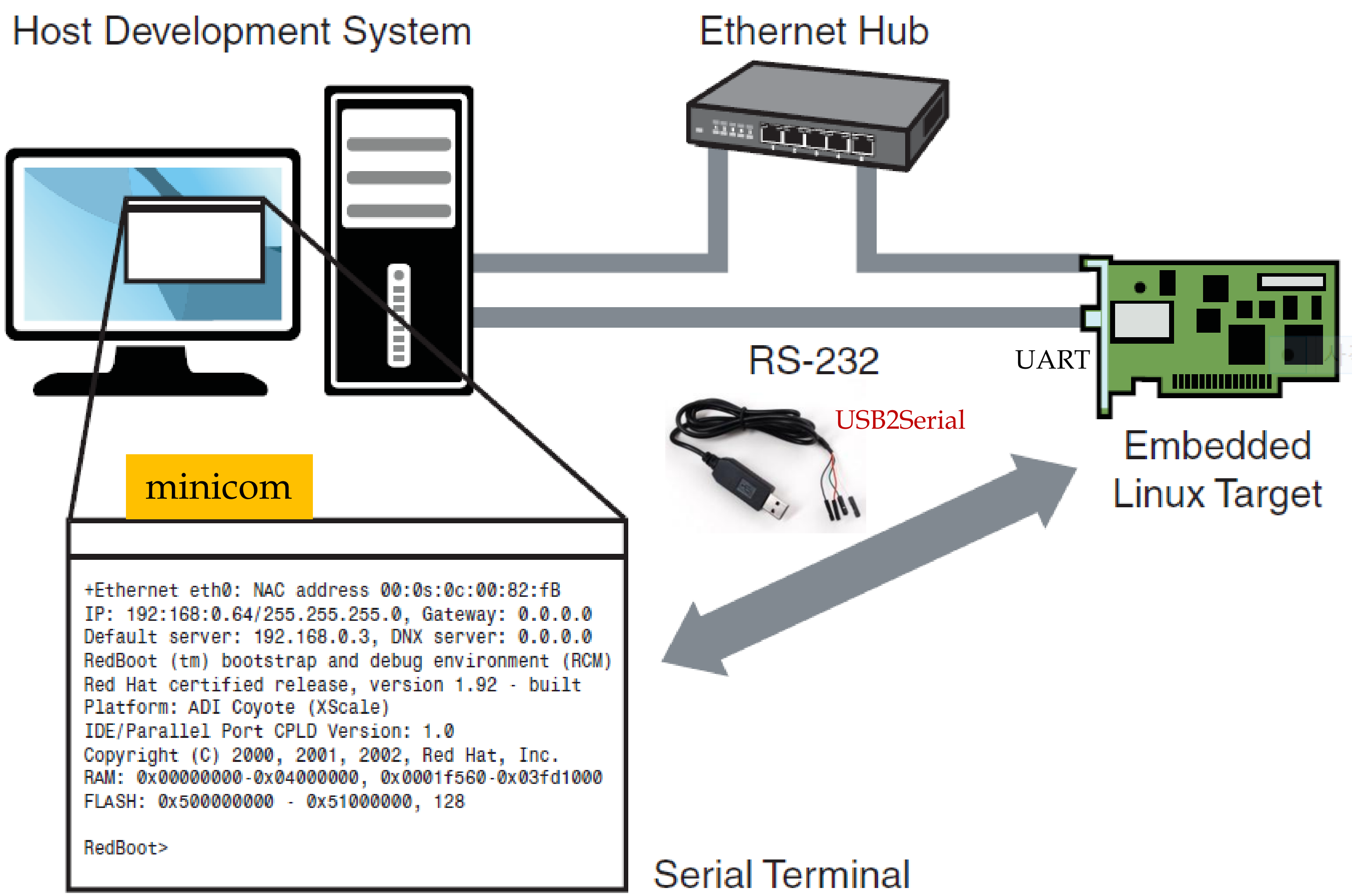
2. Raspberry Pi 2(4) - Booting 순서(2)



2. Raspberry Pi 2(5) - microSD card 구성



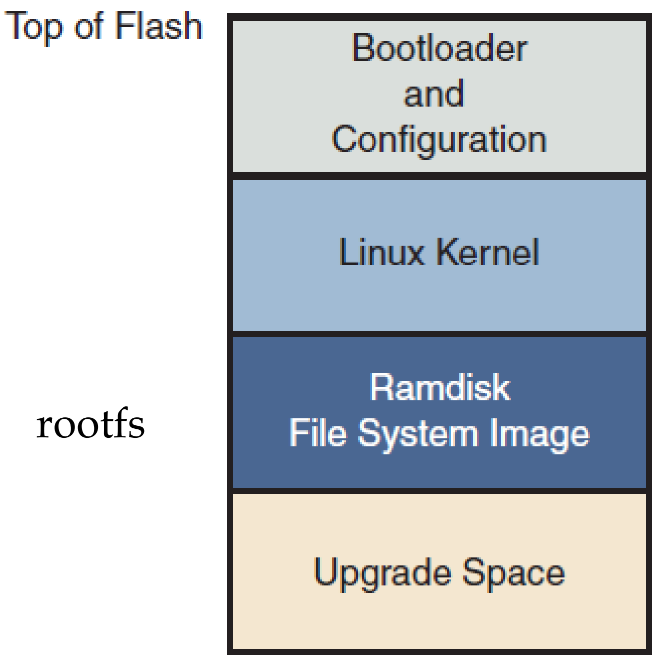
3. Embedded System(1) - Host Computer & Target Board



3. Embedded System(2) – minicom or Teraterm

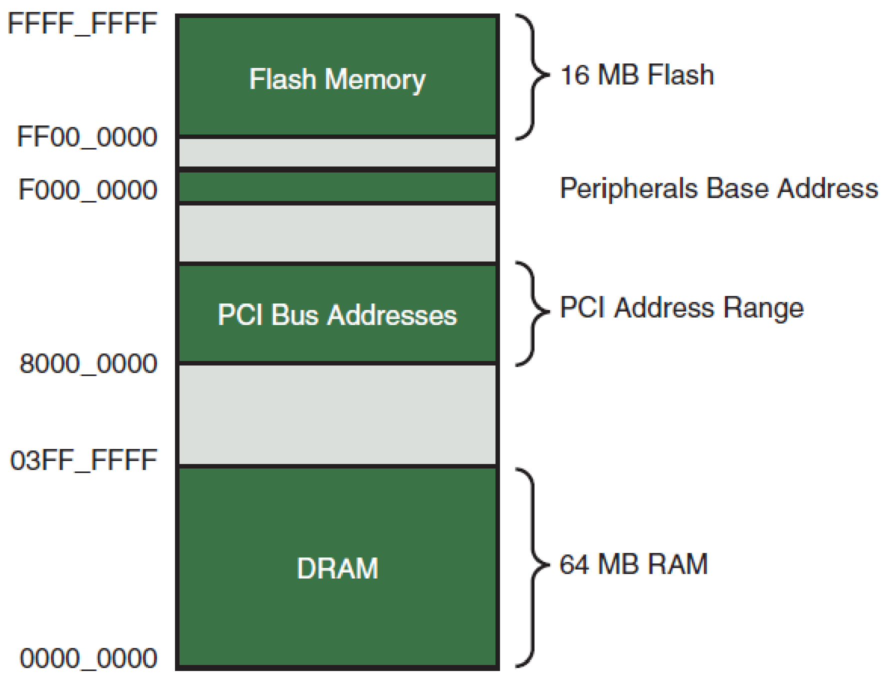
- [사용 방법 소개]

3. Embedded System(3) - Flash Memory & RAM Map



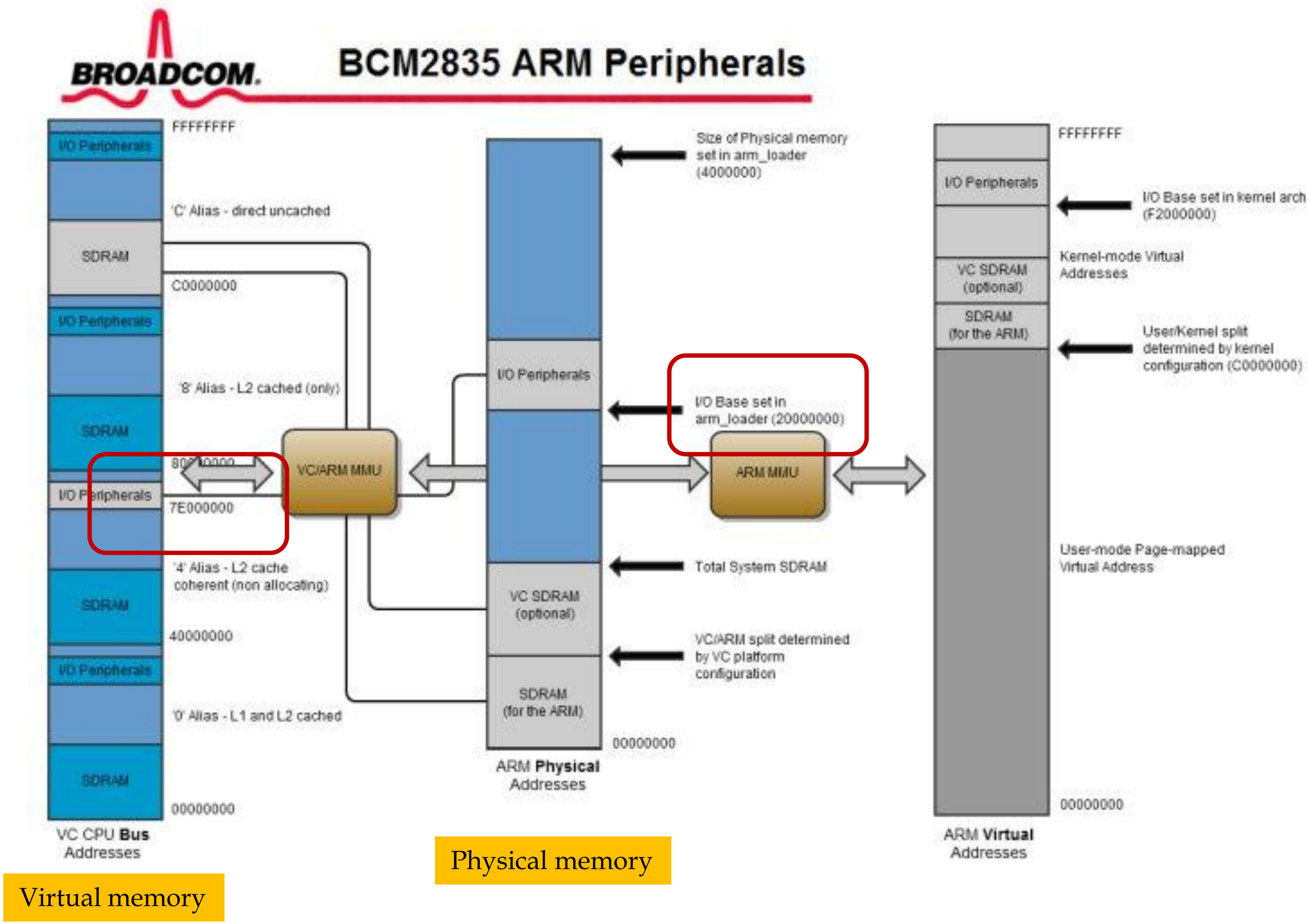
Flash memory layout

Memory mapped 방식

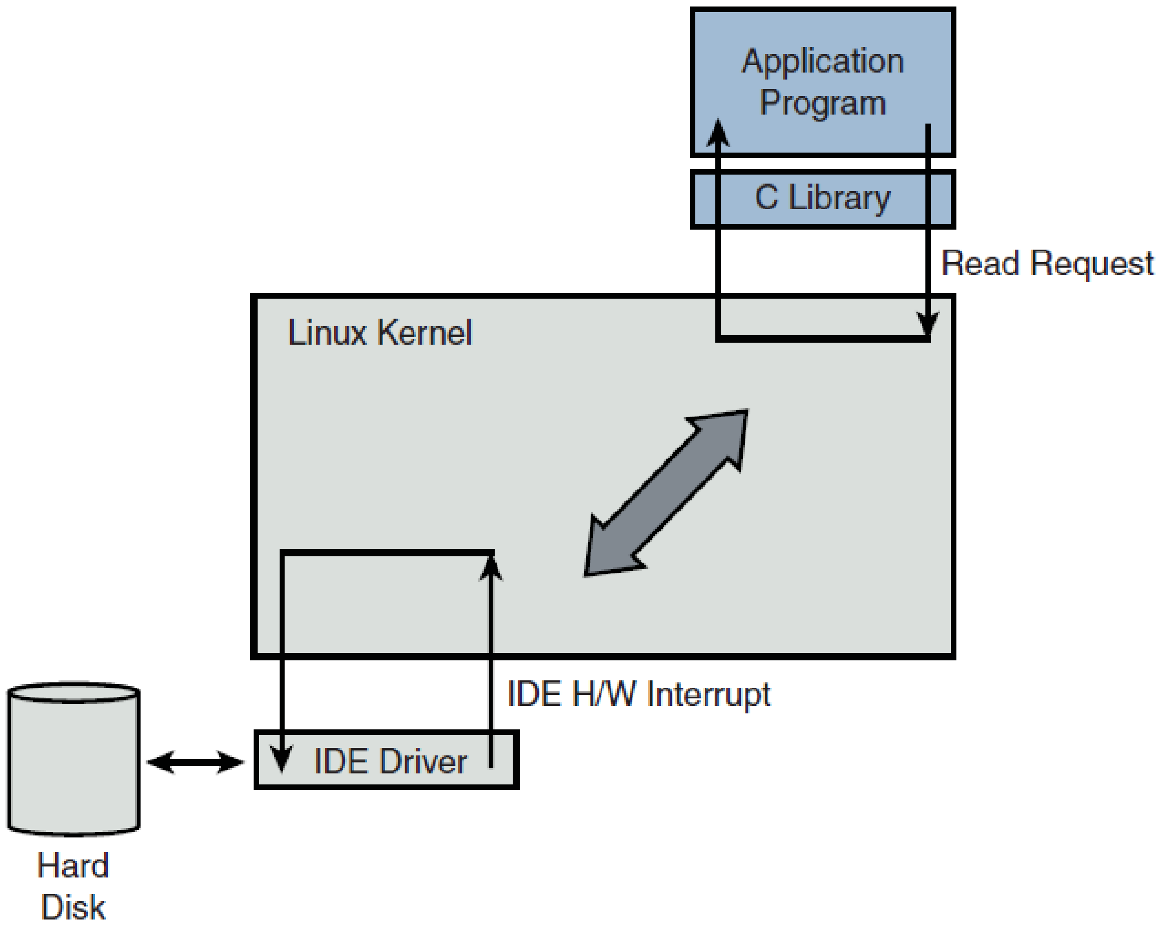


일반적인 Memory Map

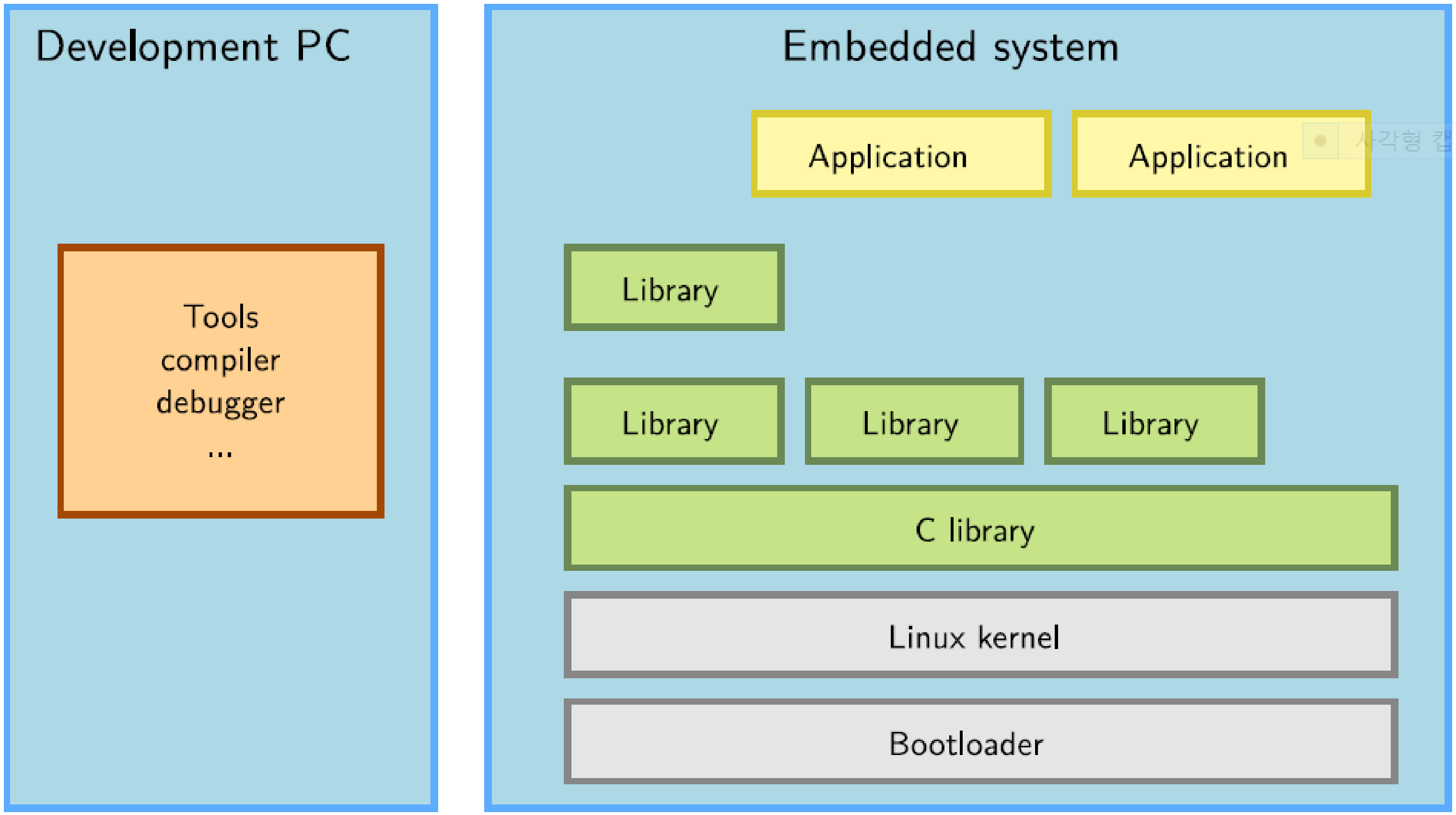
3. Embedded System(4) - RPi Memory Map Example



3. Embedded System(5) – Application & Kernel & Device



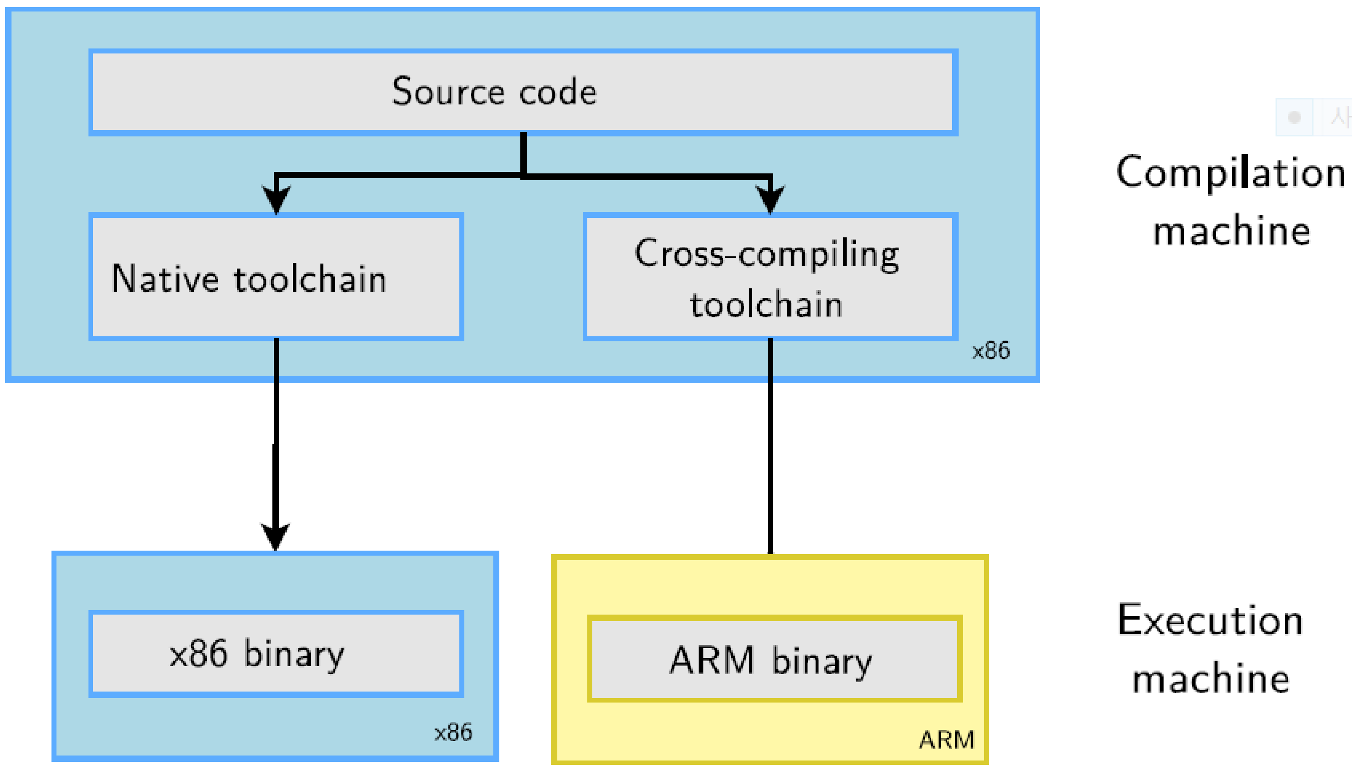
3. Embedded System(6) – S/W 구성 요소(1)



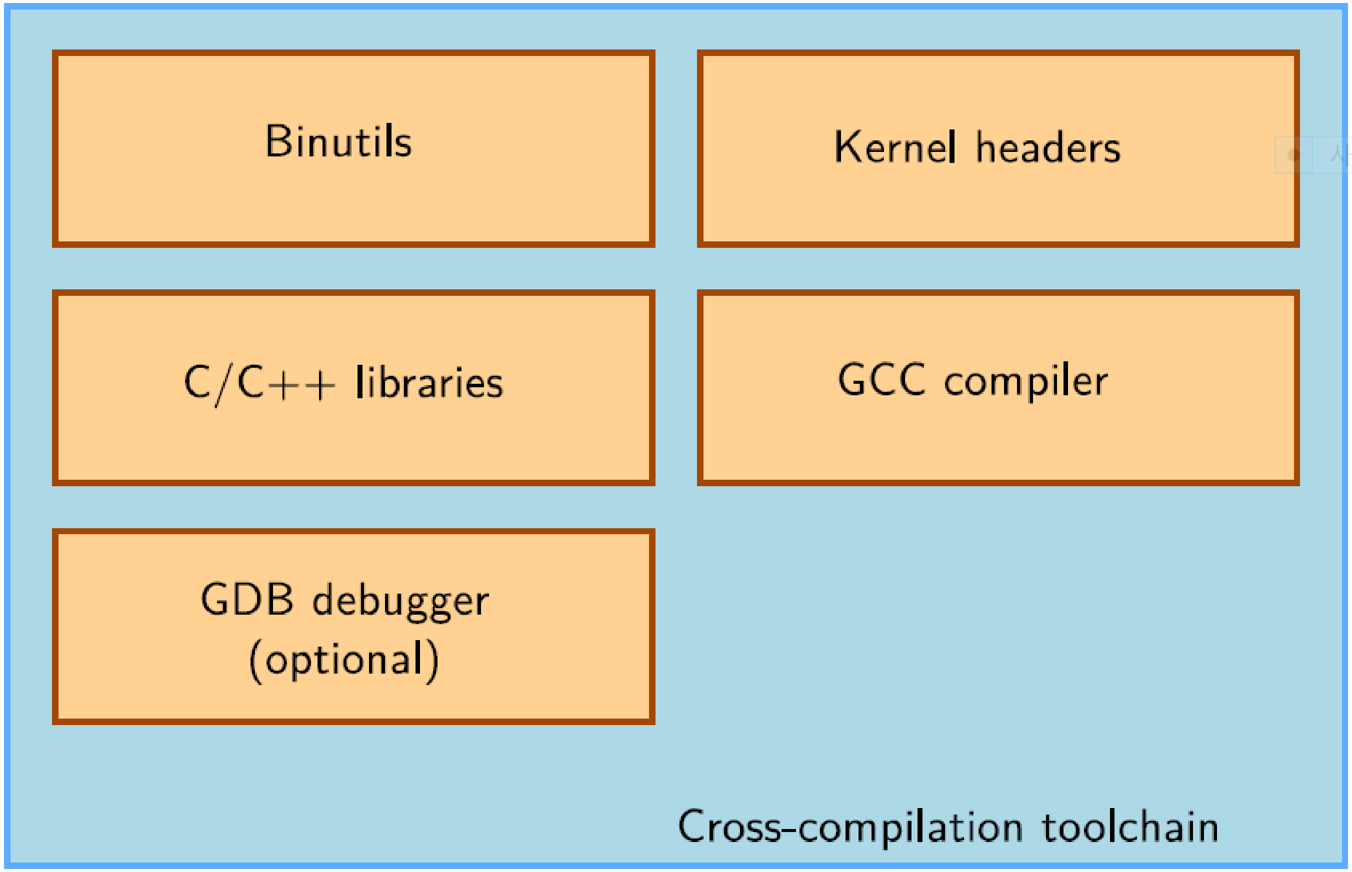
3. Embedded System(6) – S/W 구성 요소(2)

- Cross-compilation toolchain
  - Compiler that runs on the development machine, but generates code for the target
- Bootloader
  - Started by the hardware, responsible for basic initialization, loading and executing the kernel
- Linux Kernel
  - Contains the process and memory management, network stack, device drivers and provides services to user space applications
- C library
  - The interface between the kernel and the user space applications
- Libraries and applications
  - Third-party or in-house

3. Embedded System(7) – Toolchain(1)



3. Embedded System(7) – Toolchain(2)



숙제 1: RPi를 위한 toolchain을 download하고, 간단한 C program을 만들어 RPi 상에서 돌려 볼 것.

3. Embedded System(8) – Example Application Test

- [실습 1] Host(virtualbox)에서 build 후, target board에서 실행시켜 보기

- 실습용 source code: http://www.coopj.com/LPD/
  - LPD_SOLUTIONS.tar.bz2

(*) 참고, buildroot에서 uClibc -> glibc로 바꾼 후, image를 새로 설치 후, 작업

(*) 이게 여유치 않을 경우, 아래 toolchain을 사용하면 됨.
rpi-buildroot/output/host/usr/bin/**arm-buildroot-linux-uclibcgnueabihf-gcc**

```
$ export PATH=$PATH:YOUR_PATH/rpi-buildroot/output/host/usr/bin
$ vi Makefile

CC = arm-buildroot-linux-uclibcgnueabihf-gcc
…
```

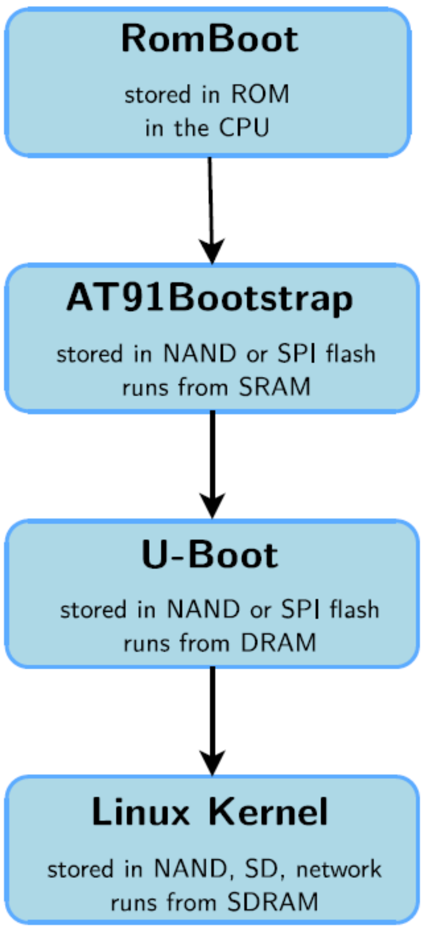
Bootloader
(Chapter 7)

1. U-Boot Bootloader - 가장 유명한 bootloader

U-Boot is a typical free software project

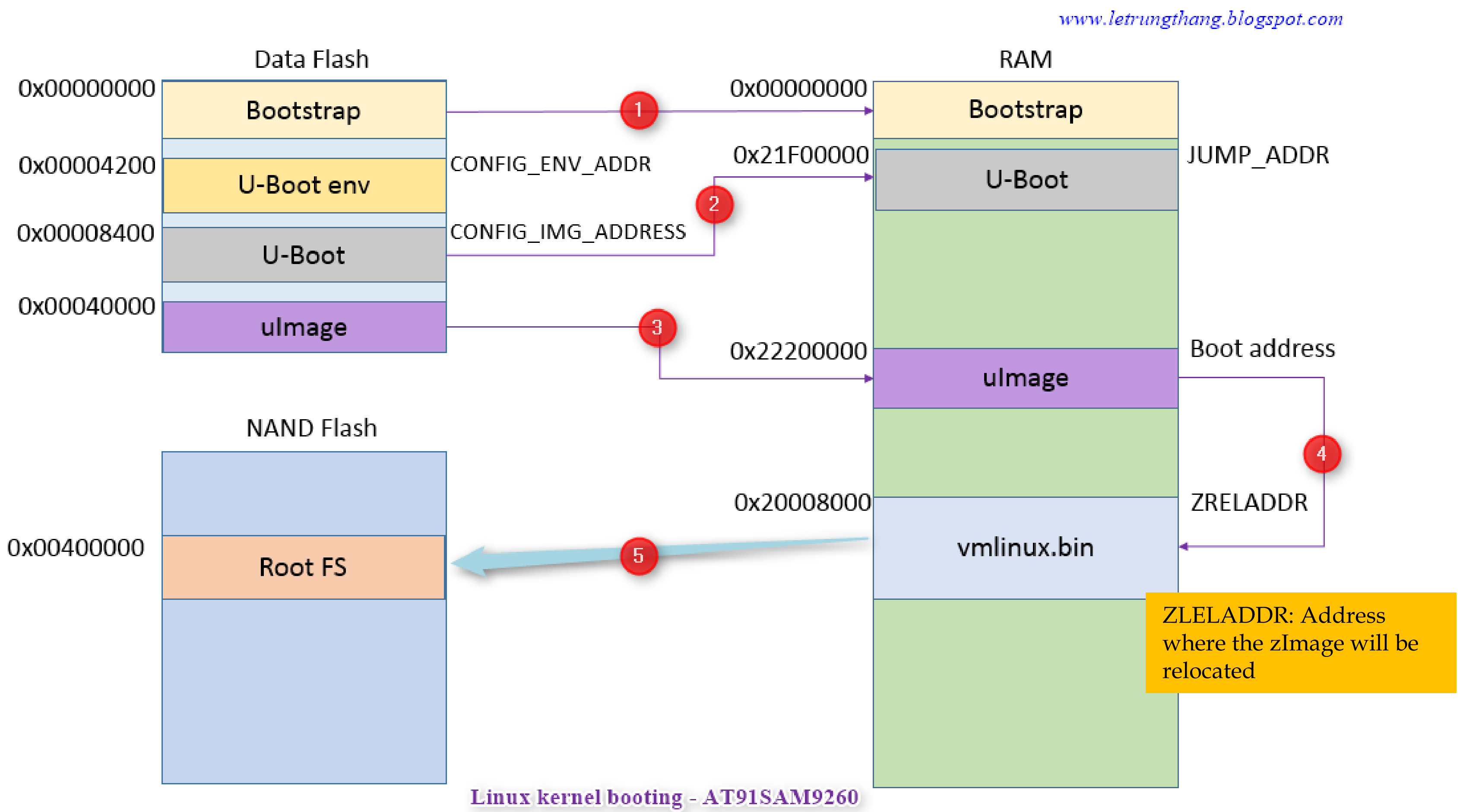
- License: GPLv2 (same as Linux)
- Freely available at http://www.denx.de/wiki/U-Boot
- Documentation available at http://www.denx.de/wiki/U-Boot/Documentation
- The latest development source code is available in a Git repository: http://git.denx.de/?p=u-boot.git;a=summary
- Development and discussions happen around an open mailing-list http://lists.denx.de/pipermail/u-boot/
- Since the end of 2008, it follows a fixed-interval release schedule. Every three months, a new version is released. Versions are named YYYY.MM.

2. Atmel AT91 예(1)



- **RomBoot**: tries to find a valid bootstrap image from various storage sources, and load it into SRAM (DRAM not initialized yet). Size limited to 4 KB. No user interaction possible in standard boot mode.
- **AT91Bootstrap**: runs from SRAM. Initializes the DRAM, the NAND or SPI controller, and loads the secondary bootloader into RAM and starts it. No user interaction possible.
- **U-Boot**: runs from RAM. Initializes some other hardware devices (network, USB, etc.). Loads the kernel image from storage or network to RAM and starts it. Shell with commands provided.
- **Linux Kernel**: runs from RAM. Takes over the system completely (bootloaders no longer exists).

2. Atmel AT91 예(2)



3. 부팅 화면 예(TI OMAP)

```
U-Boot 2013.04 (May 29 2013 - 10:30:21)

OMAP36XX/37XX-GP ES1.2, CPU-OPP2, L3-165MHz, Max CPU Clock 1 Ghz
IGEPv2 + LPDDR/NAND
I2C:   ready
DRAM:  512 MiB
NAND:  512 MiB
MMC:   OMAP SD/MMC: 0

Die ID #255000029ff800000168580212029011
Net:   smc911x-0
U-Boot #
```

u-boot 명령 입력

4. U-Boot 명령 입력 모드

Flash information (NOR and SPI flash)

```
U-Boot> flinfo
DataFlash:AT45DB021
Nb pages: 1024
Page Size: 264
Size= 270336 bytes
Logical address: 0xC0000000
Area 0: C0000000 to C0001FFF (RO) Bootstrap
Area 1: C0002000 to C0003FFF Environment
Area 2: C0004000 to C0041FFF (RO) U-Boot
```

NAND flash information

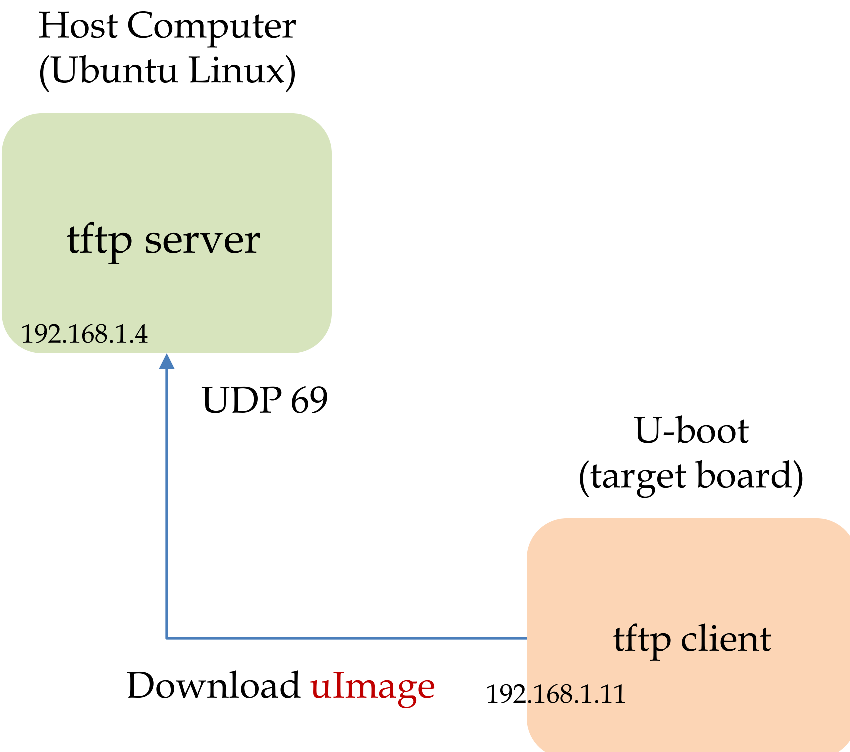
```
U-Boot> nand info
Device 0: nand0, sector size 128 KiB
  Page size       2048 b
  OOB size          64 b
  Erase size    131072 b
```

Version details

```
U-Boot> version
U-Boot 2013.04 (May 29 2013 - 10:30:21)
```

```
u-boot # printenv
baudrate=19200
ethaddr=00:40:95:36:35:33
netmask=255.255.255.0
ipaddr=10.0.0.11
serverip=10.0.0.1
stdin=serial
stdout=serial
stderr=serial
u-boot # printenv serverip
serverip=10.0.0.1
u-boot # setenv serverip 10.0.0.100
u-boot # saveenv
```

5. tftp kernel image loading




```
U-Boot# dhcp
link up on port 0, speed 100, full duplex
BOOTP broadcast 1
DHCP client bound to address 192.168.1.11

--------------------------------------------------------
Next will be configuring the server-ip [our host machine's IP], kernel's
command line and the load address
--------------------------------------------------------

U-Boot# setenv serverip 192.168.1.4
U-Boot# setenv bootfile uImage
U-Boot# setenv bootargs console=ttyO0,115200
root=/dev/mmcblk0p2 rw rootwait ip=dhcp
U-Boot# tftp 0x80200000 uImage
link up on port 0, speed 100, full duplex
Using cpsw device
TFTP from server 192.168.1.4; our IP address is
192.168.1.11
Filename 'uImage'.
Load address: 0x80200000
Loading:
#################################################################
#################################################################T
############
##############################################T
#######################
######################################T ########
146.5 KiB/s
done
Bytes transferred = 3484264 (352a68 hex)
U-Boot# bootm 0x80200000
## Booting kernel from Legacy Image at 80200000 ...
Image Name: Angstrom/3.2.28/beaglebone
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 3484200 Bytes = 3.3 MiB
Load Address: 80008000
Entry Point: 80008000
Verifying Checksum ... OK
Loading Kernel Image ... OK
OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
[ 0.000000] Initializing cgroup subsys cpu
[ 0.000000] Linux version 3.2.28 (koen@Angstrom-F16-vm-
rpm) (gcc version 4.5.4 20120305 (prerelease) (GCC) ) #1
Tue Sep 11 13:08:30 CEST 2012
[ 0.000000] CPU: ARMv7 Processor [413fc082] revision 2
(ARMv7), cr=50c53c7d
[ 0.000000] CPU: PIPT / VIPT nonaliasing data cache,
VIPT aliasing instruction cache
[ 0.000000] Machine: am335xevm
```

숙제 2: ubuntu linux에 tftp server를 추가하시오.
➔ tftp 명령으로 file을 내리고, 올려 보세요.

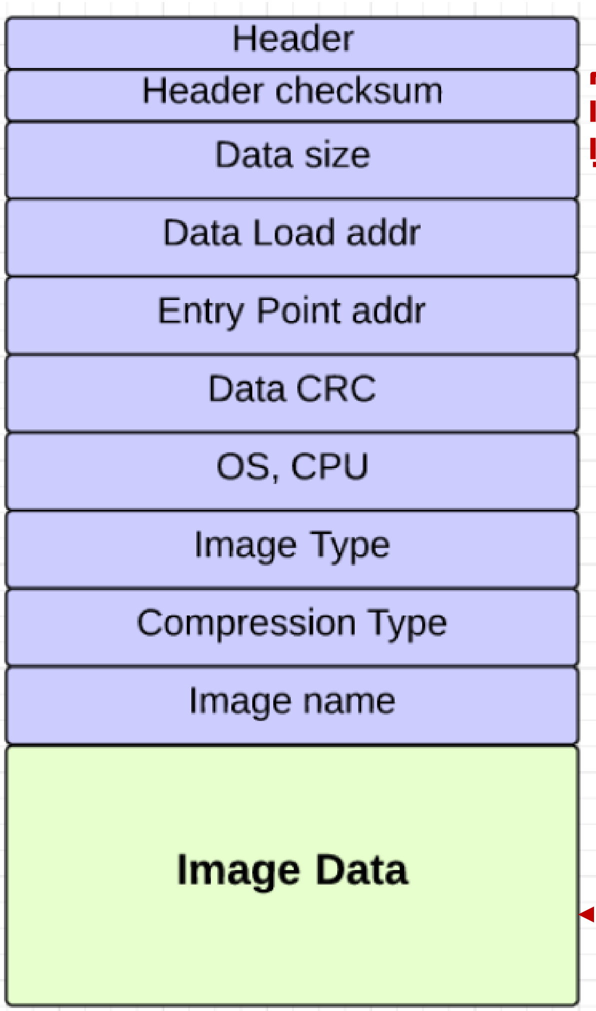
6. Source Download & Build 절차

- git clone git://git.denx.de/u-boot.git

- cd u-boot/
- export CC=`pwd`/${YOUR_PATH}/**arm-linux-gnueabihf-**
- make ARCH=arm CROSS_COMPILE=${CC} distclean
- make ARCH=arm CROSS_COMPILE=${CC} omap3_beagle_defconfig
- make ARCH=arm CROSS_COMPILE=${CC}

숙제 3: U-Boot source code를 download하고, build해 볼 것.

7. uImage – u-boot Kernel Image




```
chyi@earth:~/lterouter/workspace/A20/linux-sunxi$ mkimage -l arch/arm/boot/uImage
Image Name:   Linux-3.4.103-00033-g9a1cd03-dir
Created:      Wed Mar 25 09:55:51 2015
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    3856640 Bytes = 3766.25 kB = 3.68 MB
Load Address: 40008000
Entry Point:  40008000
```

(*) uImage의 loadaddr & entry point는 동일하게 0x4000 8000 임 !
(*) 여기에는 표시되지 않았으나, ZRELADDR 값도 동일하게 설정되어 있음.

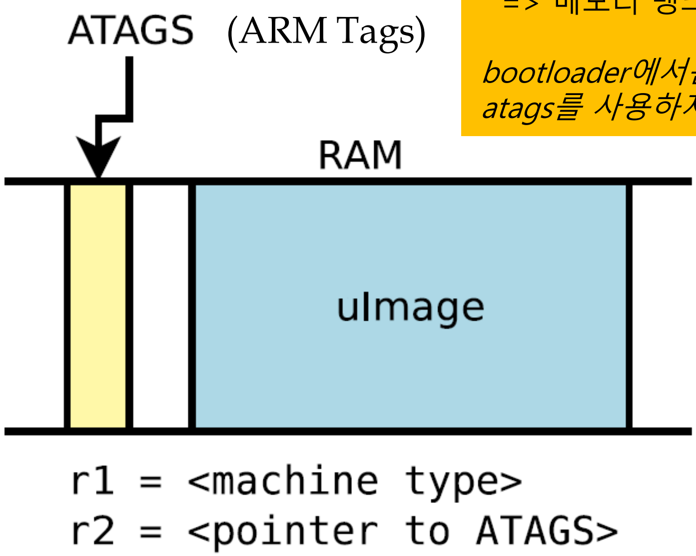
왼쪽과 같은 format으로 uImage가 구성되어 있으며, U-boot이 이를 loading하여, header를 파싱하게 됨.

zImage가 위치함.

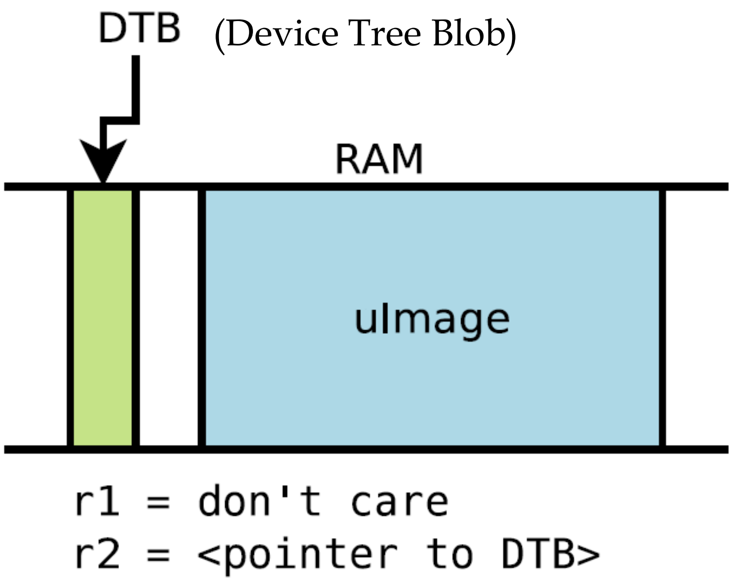
8. ATAGS(OLD) & DTB(NEW)

Atags는 bootloader에서 linux kernel로 필요한 정보를 전달하기 위해 사용함.
=> 메모리 뱅크 정보, 커널 파라미터, 램디스크 위치, 프레임 버퍼 주소 등.

bootloader에서는 atag에 대한 포인터를 kernel 호출인자로 반드시 넘겨줘야 함.
atags를 사용하지 않을 경우 0으로 초기화 함.



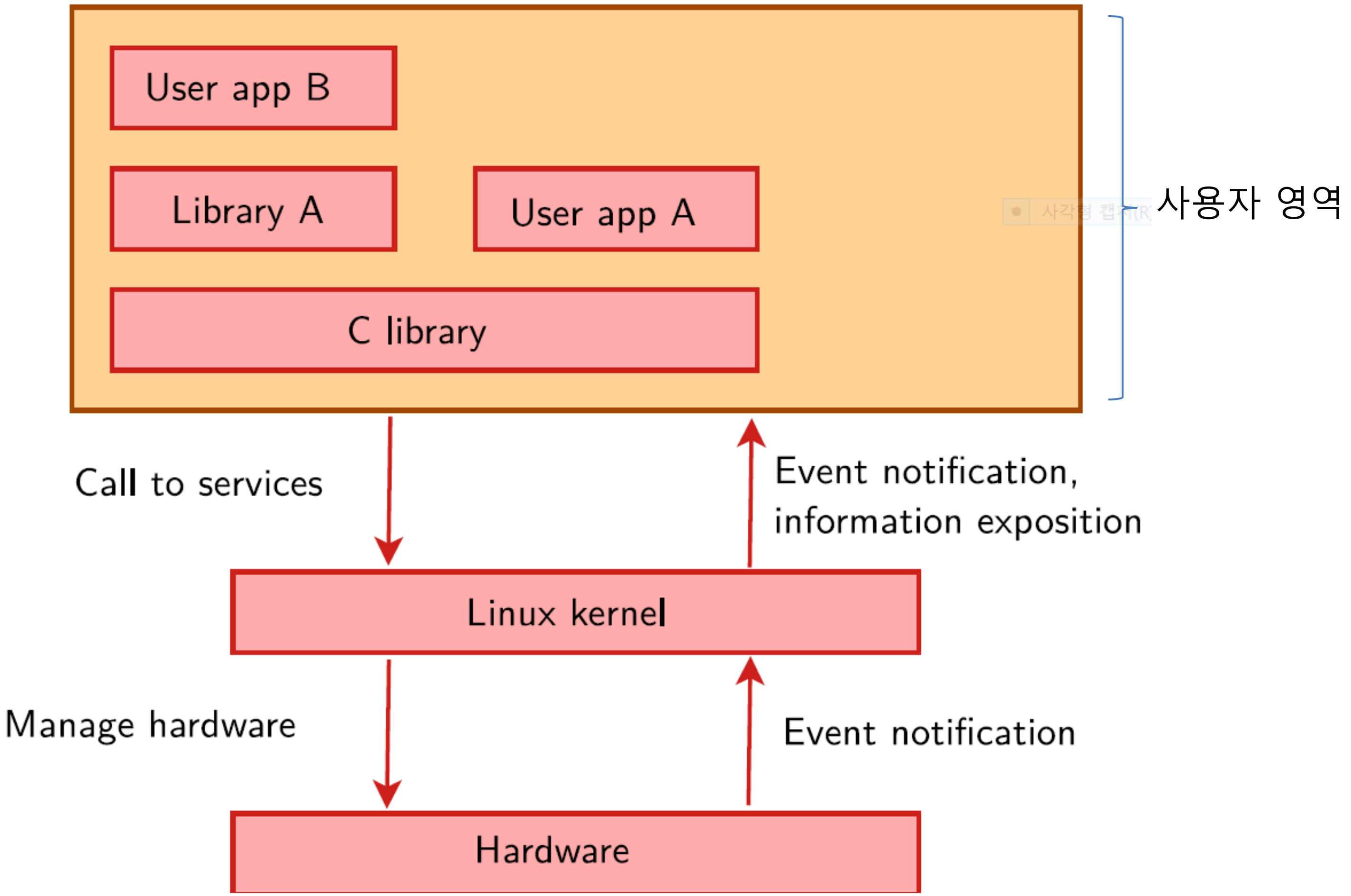
(*) DTB 관련해서는 참고 문헌 [4] 및 [7] 참조



Chapter 4-5.

: kernel build system & kernel 초기화 과정

1. Linux Kernel 개요(1)



1. Linux Kernel 개요(2) – kernel 영역과 사용자 영역(1)

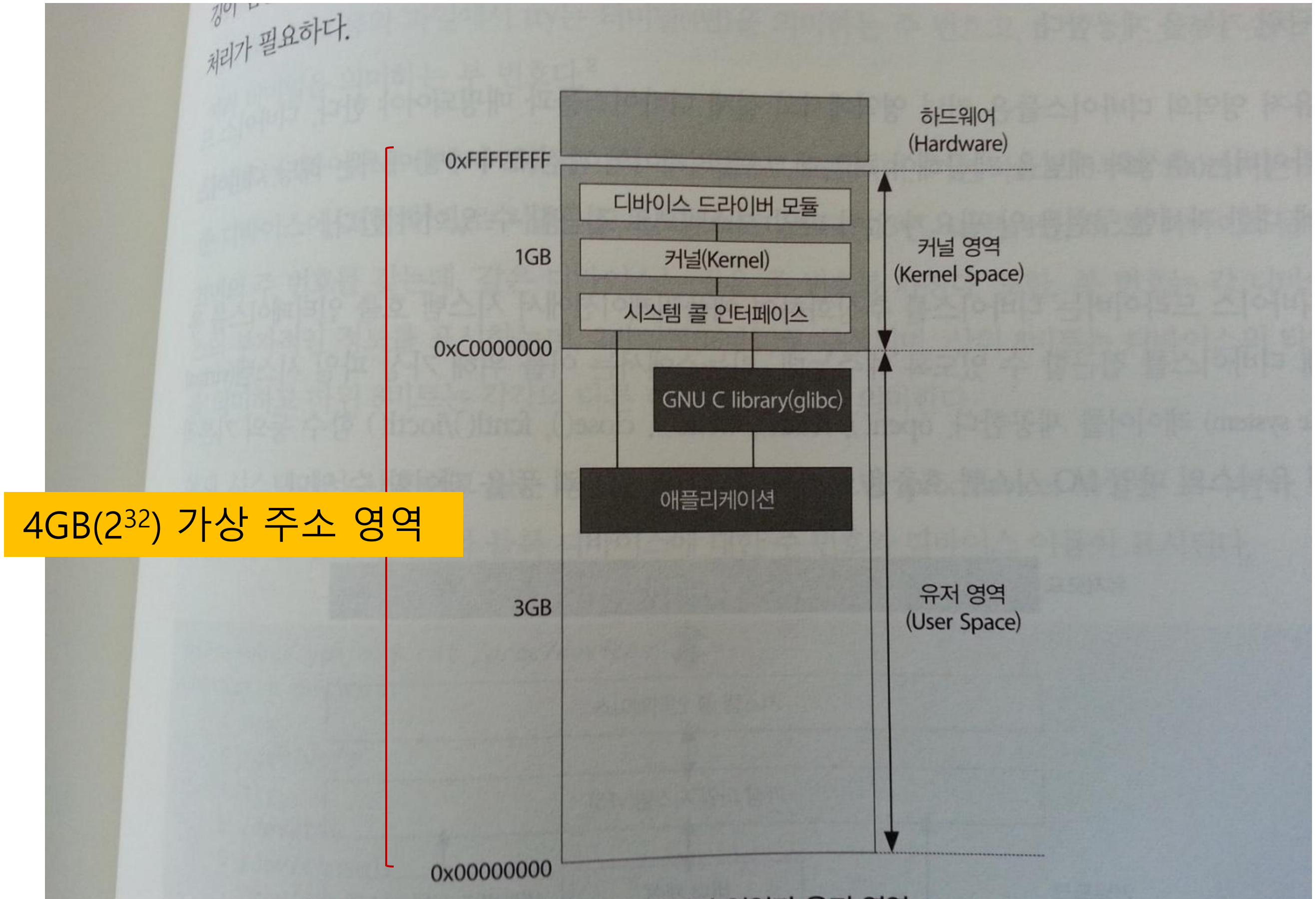


그림 9-4 커널 영역과 유저 영역

1. Linux Kernel 개요(2) – kernel 영역과 사용자 영역(2)

0xC0000000

사용자 영역

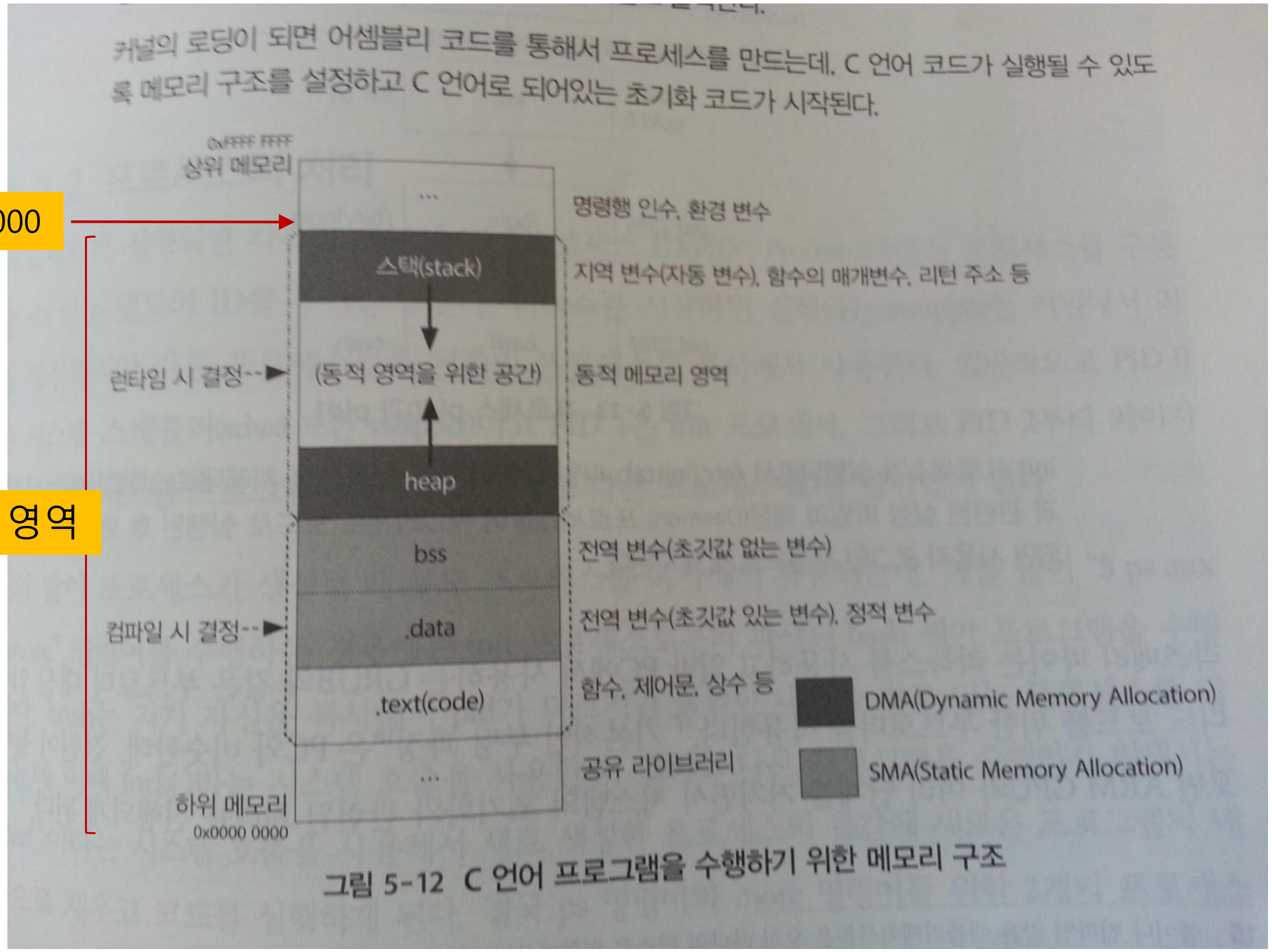

커널의 로딩이 되면 어셈블리 코드를 통해서 프로세스를 만드는데, C 언어 코드가 실행될 수 있도록 메모리 구조를 설정하고 C 언어로 되어있는 초기화 코드가 시작된다.

그림 5-12 C 언어 프로그램을 수행하기 위한 메모리 구조

1. Linux Kernel 개요(2) – kernel 영역과 사용자 영역(3)

인 유닉스의 파일 I/O 시스템 호출을 사용하여 데이터 입출력 등을 제어할 수 있다.

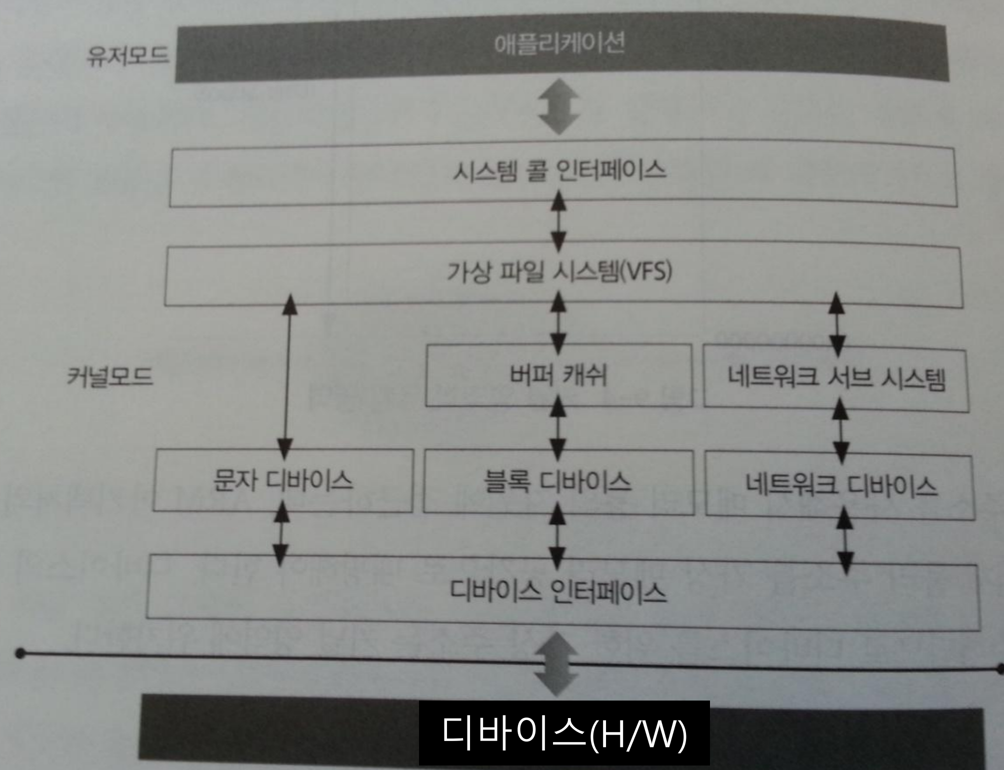
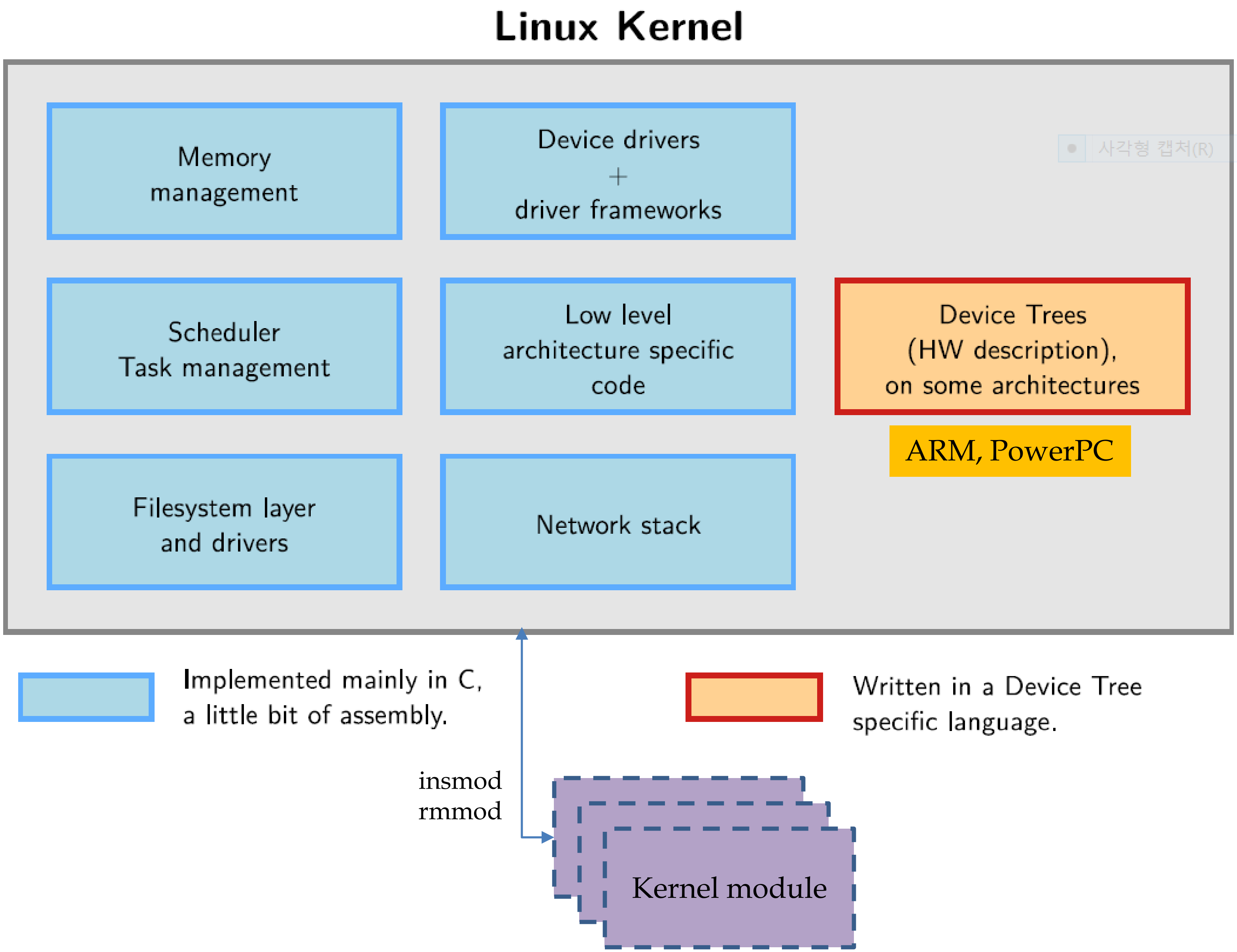


그림 9-5 리눅스 커널과 디바이스 드라이버

1. Linux Kernel 개요(3)

(*) Device Tree 관련해서는 참고 문헌 [4] 참조



1. Linux Kernel 개요(4) – 디렉토리 구성

https://www.kernel.org/

- drivers/: 49.4%
- arch/: 21.9%
- fs/: 6.0%
- include/: 4.7%
- sound/: 4.4%
- Documentation/: 4.0%
- net/: 3.9%
- firmware/: 1.0%
- kernel/: 1.0%
- tools/: 0.9%
- scripts/: 0.5%
- mm/: 0.5%
- crypto/: 0.4%
- security/: 0.4%
- lib/: 0.4%
- block/: 0.2%
- ...

2. Kernel Build System(1) - .config

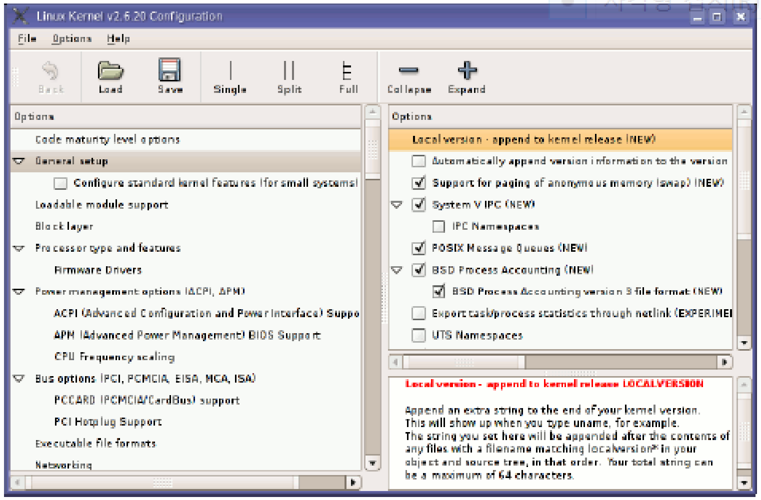
(*) kernel code를 review하자.

```
#
# CD-ROM/DVD Filesystems
#
CONFIG_ISO9660_FS=m
CONFIG_JOLIET=y
CONFIG_ZISOFS=y
CONFIG_UDF_FS=y
CONFIG_UDF_NLS=y

#
# DOS/FAT/NT Filesystems
#
# CONFIG_MSDOS_FS is not set
# CONFIG_VFAT_FS is not set
CONFIG_NTFS_FS=m
# CONFIG_NTFS_DEBUG is not set
CONFIG_NTFS_RW=y
```

2. Kernel Build System(2) – make gconfig

`make gconfig`

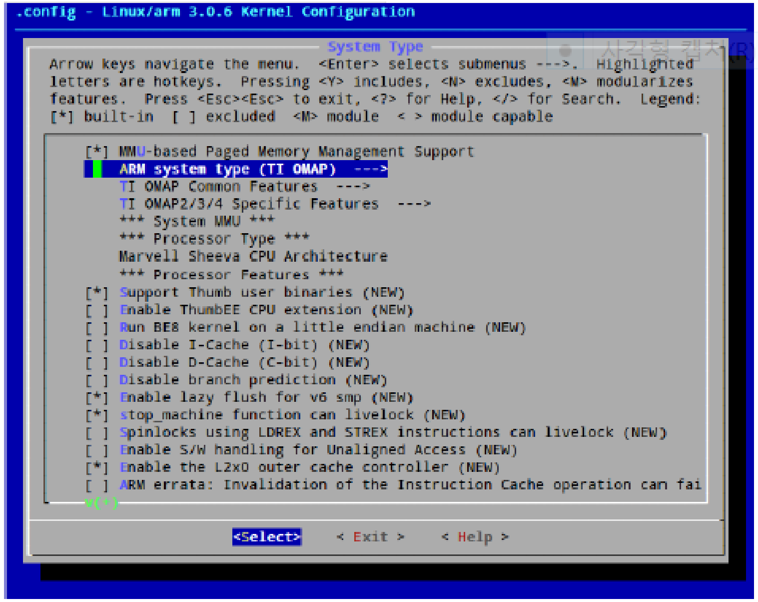
- *GTK* based graphical configuration interface. Functionality similar to that of make `xconfig`.
- Just lacking a search functionality.
- Required Debian packages: `libglade2-dev`



2. Kernel Build System(3) – make menuconfig

`make menuconfig`

- Useful when no graphics are available. Pretty convenient too!
- Same interface found in other tools: BusyBox, Buildroot...
- Required Debian packages: `libncurses-dev`



(*) 내용을 살펴 보자.

2. Kernel Build System(4) – Kconfig & Makefile

LISTING 4-8 Snippet from .../arch/arm/Kconfig

```
source "init/Kconfig"

menu "System Type"

choice
        prompt "ARM system type"
        default ARCH_RPC

config ARCH_CLPS7500
        bool "Cirrus-CL-PS7500FE"

config ARCH_CLPS711X
        bool "CLPS711x/EP721x-based"

...

source "arch/arm/mach-ixp4xx/Kconfig"
```

LISTING 4-11 Makefile from .../arch/arm/mach-ixp4xx Kernel Subdirectory

```
#
# Makefile for the linux kernel.
#

obj-y   += common.o common-pci.o

obj-$(CONFIG_ARCH_IXDP4XX)    += ixdp425-pci.o ixdp425-setup.o
obj-$(CONFIG_MACH_IXDPG425)    += ixdpg425-pci.o coyote-setup.o
obj-$(CONFIG_ARCH_ADI_COYOTE)  += coyote-pci.o coyote-setup.o
obj-$(CONFIG_MACH_GTWX5715)    += gtwx5715-pci.o gtwx5715-setup.o
```

3. Linux Kernel Build(1)

make mrproper	커널 설정 초기화(*항상 수행하는 것 아님*)
make YOURBOARD_defconfig	Target board에 맞는 config 설정 (arch/arm/configs 디렉토리에 있음) => .config 파일 생성됨.
make menuconfig	Kernel configuration 조정 => .config의 내용이 변경됨.
make zImage	압축된 kernel image 생성 => zImage, bzImage, uImage 등 종류마다 다름.
make modules make modules_install	Kernel 모듈 생성 및 설치(rootfs 디렉토리에)
make dtbs	DTB 생성(*ARM, PowerPC 등에서만 필요*)

3. Linux Kernel Build(2)– RPi(1)

- $ git clone --depth=1 https://github.com/raspberrypi/linux

- $ cd linux
- $ KERNEL=kernel7

(RPI buildroot) make linux-menuconfig

- <kernel configuration 지정>
- $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig

- <kernel, module dtb 한번에 build>
- $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs
- or
- <각각 build>
- $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage
- $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- modules
- $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- dtbs

- *(*) See https://www.raspberrypi.org/documentation/linux/kernel/building.md*

숙제 4: kernel source code를 buildroot와 무관하게 download하고, build해 볼 것.
➔ zImage를 새로 build한 것으로 바꾸어, Raspberry Pi에서 돌려 볼 것.

3. Linux Kernel Build(2) – RPi(2)

- 앞 페이지 방법이 여유치 않을 경우, buildroot kernel을 이용해서 직접 build

- **export CROSS_COMPILE**=$(YOUR_PATH)/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/bin/arm-linux-gnueabihf-

- **export PATH**=$(YOUR_PATH)/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian/bin:$PATH

- **export KERNEL_DIR**=$(YOUR_PATH)/rpi-buildroot/output/build/linux-rpi-4.1.y

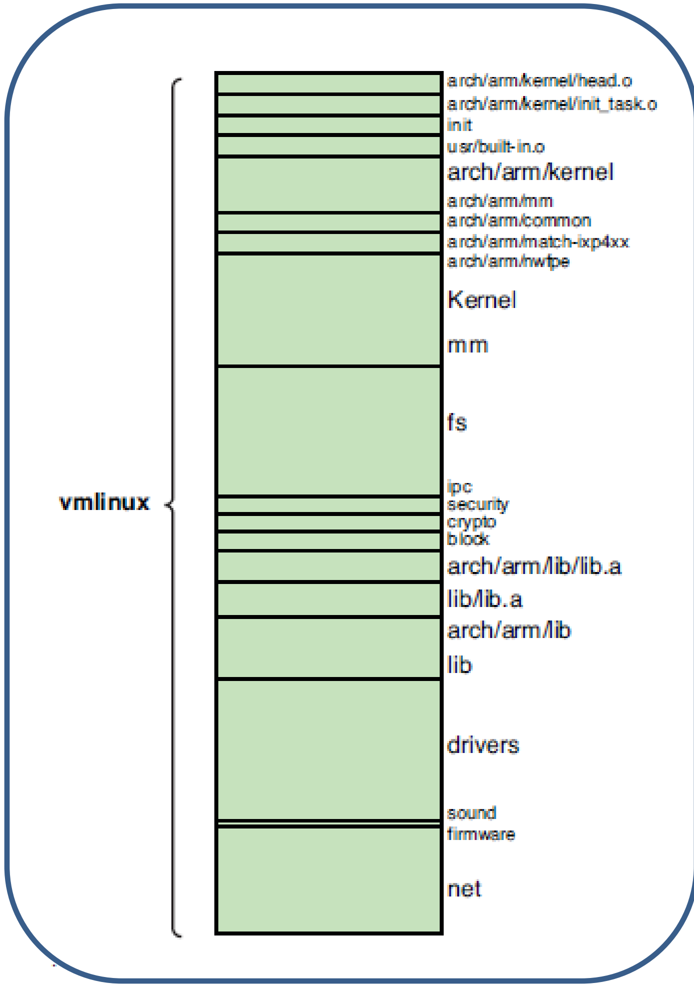
- cd $(YOUR_PATH)/rpi-buildroot/output/build/linux-rpi-4.1.y
- make menuconfig

- make **–j4** **ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-** zImage modules

4. Kernel Build 결과물(1)

```
$ make ARCH=arm CROSS_COMPILE=xscale_be- zImage
  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
```

[7] Executable and Linking Format, a de facto standard format for binary executable files.



일반적인 kernel build 과정

4.2

LISTING 4-1 Continued

```
  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                          [1]
  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
  AS      arch/arm/boot/compressed/head-xscale.o
  AS      arch/arm/boot/compressed/big-endian.o
  LD      arch/arm/boot/compressed/vmlinux             [2]
  OBJCOPY arch/arm/boot/zImage                         [3]
  Kernel: arch/arm/boot/zImage is ready
```

4. Kernel Build 결과물(2) - Kernel Image 구성

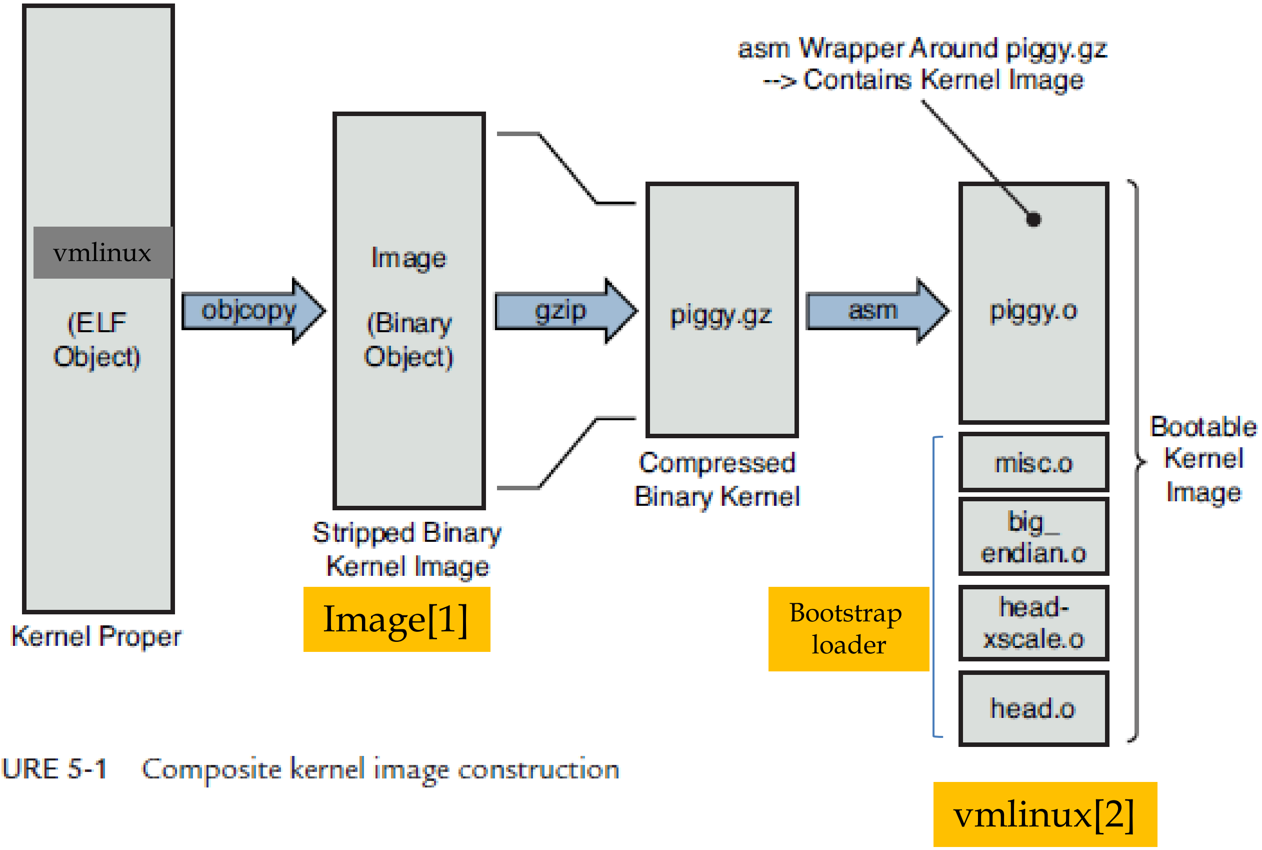
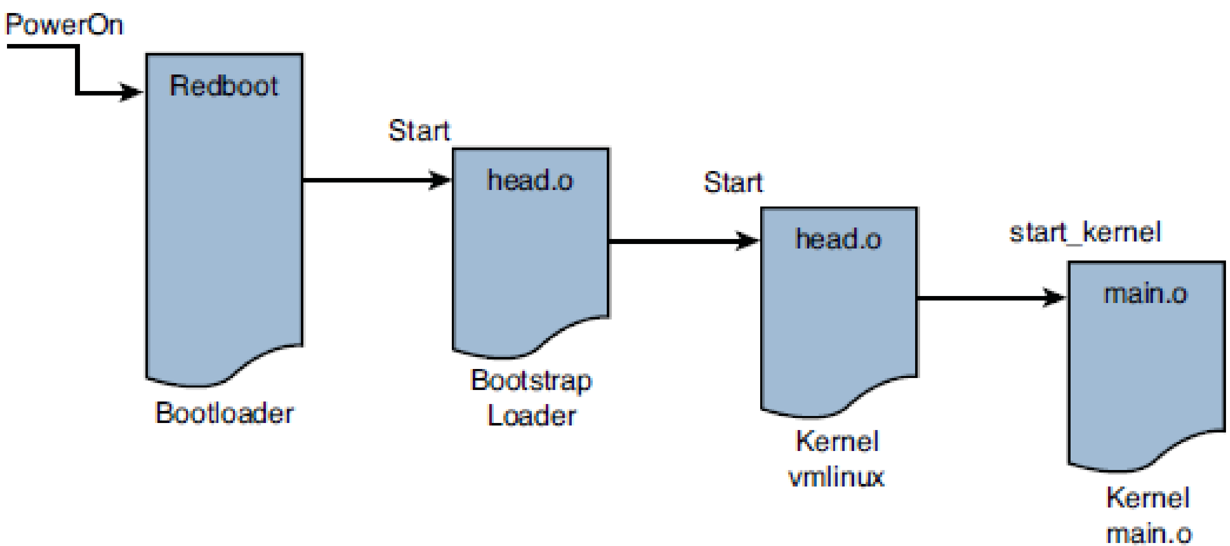


FIGURE 5-1 Composite kernel image construction

4. Kernel Build 결과물(3)- Kernel Boot

LISTING 5-3 Linux Boot Messages on IPX425

```
Using base address 0x01000000 and length 0x001ce114
Uncompressing Linux....... done, booting the kernel.
Linux version 2.6.32-07500-g8bea867 (chris@brutus2) (gcc version 4.2.0
20070126 (prerelease) (MontaVista 4.2.0-3.0.0.0702771 2007-03-10)) #12 Wed Dec 16
23:07:01 EST 2009
CPU: XScale-IXP42x Family [690541c1] revision 1 (ARMv5TE), cr=000039ff
CPU: VIVT data cache, VIVT instruction cache
Machine: ADI Engineering Coyote
Memory policy: ECC disabled, Data cache writeback
```



mage: Piggy and Friends 107

```
tal pages: 16256
=dhcp
es)
s)
```

```
Memory: 61108KB available (3332K code, 199K data, 120K init, 0K highmem)
SLUB: Genslabs=11, HWalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
Hierarchical RCU implementation.
RCU-based detection of stalled CPUs is enabled.
NR_IRQS:64
Calibrating delay loop... 532.48 BogoMIPS (lpj=2662400)
Mount-cache hash table entries: 512
```

5. Kernel 초기화 과정(1)

커널 초기화 과정을 요약해 보면 다음과 같다.

1) bootloader는 bootstrap code를 실행시킨다.

2) Bootstrap 코드는 프로세서와 보드를 초기화하고, 커널의 압축을 풀어 RAM에 적재한 후, start_kernel() 함수를 호출해 준다.

3) 커널은 bootloader로부터 command line option을 복사해 온다.

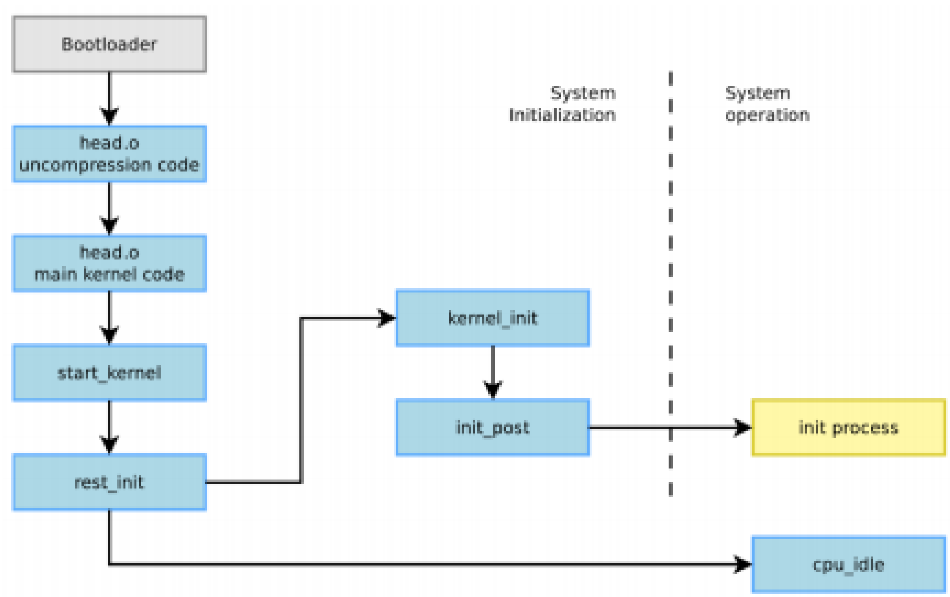
4) 커널은 프로세서와 머신을 초기화시킨다.

5) 콘솔을 초기화한다.

6) 메모리 할당(memory allocation), scheduling, 파일 cache 등 커널 서비스를 초기화 시킨다.

7) 커널 thread(나중에 init process)를 생성하고, idle loop 상태에서 대기한다.

8) 장치를 초기화하고, initcall 매크로를 호출한다.



5. Kernel 초기화 과정(2) – start_kernel

아키텍쳐 특화(architecture-specific)된 초기화 코드

커널이 스스로 압축을 푼 후, 커널의 시작 부분(kernel entry point)로 분기하게 되는데, 이를 담고 있는 파일은 arch/<arch>/kernel/head.S 이며, 주요 임무는 다음과 같다.

1) architecture, 프로세서, 머신 유형 등을 검사한다.

Check the architecture, processor and machine type.

2) MMU를 설정하고, page table 항목을 만든 후, 가상 메모리(virtual memory)를 enable 시킨다.

3) init/main.c 파일에 있는 start_kernel() 함수를 호출한다.

start kernel 함수에 주로 하는 일

1) setup_arch(&command_line) 함수를 호출해 준다.

- arch/<arch>/kernel/setup.c 파일에 정의되어 있는 함수이다.
- Bootloader가 넘긴 command line 값을 복사한다.
- ARM에서는 이 함수는 setup_processor() 함수(CPU 정보가 출력됨)와 setup_machine() 함수(머신을 초기화 시켜줌)를 다시 호출한다

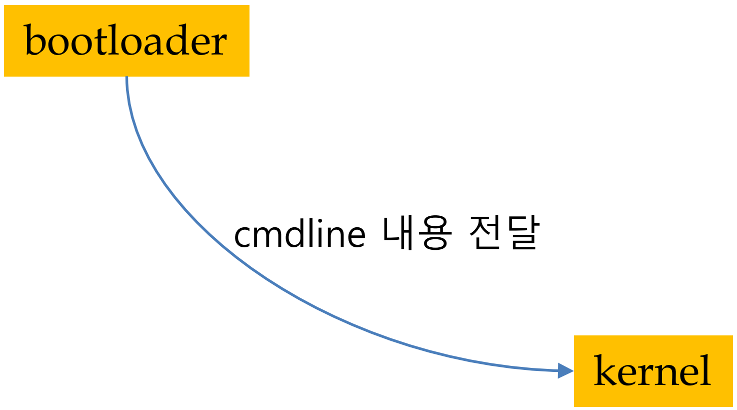
2) 에러 메시지를 출력할 수 있도록, 가능한 한 일찍 콘솔을 초기화해 준다.

3) 다양한 커널 subsystem을 초기화 시켜준다.

4) 최종적으로 rest_init() 함수를 호출한다.

5. Kernel 초기화 과정(3) - Command Line

dwc_otg.lpm_enable=0 **console=ttyAMA0,115200** console=tty1 **root=/dev/mmcblk0p2 rootfstype=ext4** elevator=deadline rootwait



LISTING 5-4 Console Setup Code Snippet

```
/*
 *    Setup a list of consoles. Called from init/main.c
 */
static int __init console_setup(char *str)
{
    char buf[sizeof(console_cmdline[0].name) + 4]; /* 4 for index */
    char *s, *options, *brl_options = NULL;
    int idx;

    ...
    <body omitted for clarity...>
    ...
```


LISTING 5-4 Continued

```
    return 1;
}

__setup("console=", console_setup);
```

5. Kernel 초기화 과정(4) - Subsystem 초기화(1)

(*) chip 제조사 SoC 관련 코드

```
MACHINE_START( …)
  .map_io = ….,
  .reserve = ….,
  .init_irq = ….,
  .handle_irq = ….,
  .init_machine = ….,
  .timer = ….,
  .init_early = ….,
  .restart = ….,
MACHINE_END
```

init/main.c

start_kernel( )

setup_arch( )

```
arm_pm_restart = mdesc->restart;
handle_arch_irq = mdesc->handle_irq;
…
```

<linker script to make ARM linux kernel>

```
SECTIONS
{
   …
   .text: {
            ….
   }
   .init.proc.info : {
            …..
   }
   .init.arch.info : {
            __arch_info_begin = .;
            *(.arch.info.init)
            __arch_info_end = .;
   }
   …
   .data : {
            ….
   }
   …
   .bss : {
            ….
   }
   …
}
```

5. Kernel 초기화 과정(4) - Subsystem 초기화(2)

```
#define INIT_CALLS  \
      VMLINUX_SYMBOL(__initcall_start) = .;  \
       *(.initcallearly.init)  \
       INIT_CALLS_LEVEL(0)  \
       INIT_CALLS_LEVEL(1)  \
       INIT_CALLS_LEVEL(2)  \
       INIT_CALLS_LEVEL(3)  \
       INIT_CALLS_LEVEL(4)  \
       INIT_CALLS_LEVEL(5)  \
       INIT_CALLS_LEVEL(rootfs)  \
       INIT_CALLS_LEVEL(6)  \
       INIT_CALLS_LEVEL(7)  \
      VMLINUX_SYMBOL(__initcall_end) = .;
```

<linker script to make ARM linux kernel>

```
SECTIONS
{
   …
   .init.data : {
         INIT_SETUP(16)
         INIT_CALLS
         CON_INITCALL
         INIT_RAM_FS
   }
   …
}
```

kernel/vmlinux.lds.S

main.c

start_kernel( )

rest_init( )

kernel_init( )

kernel_init_freeable( )

do_basic_setup( )

do_initcalls( )

do_initcall_level( )

do_one_initcall( )

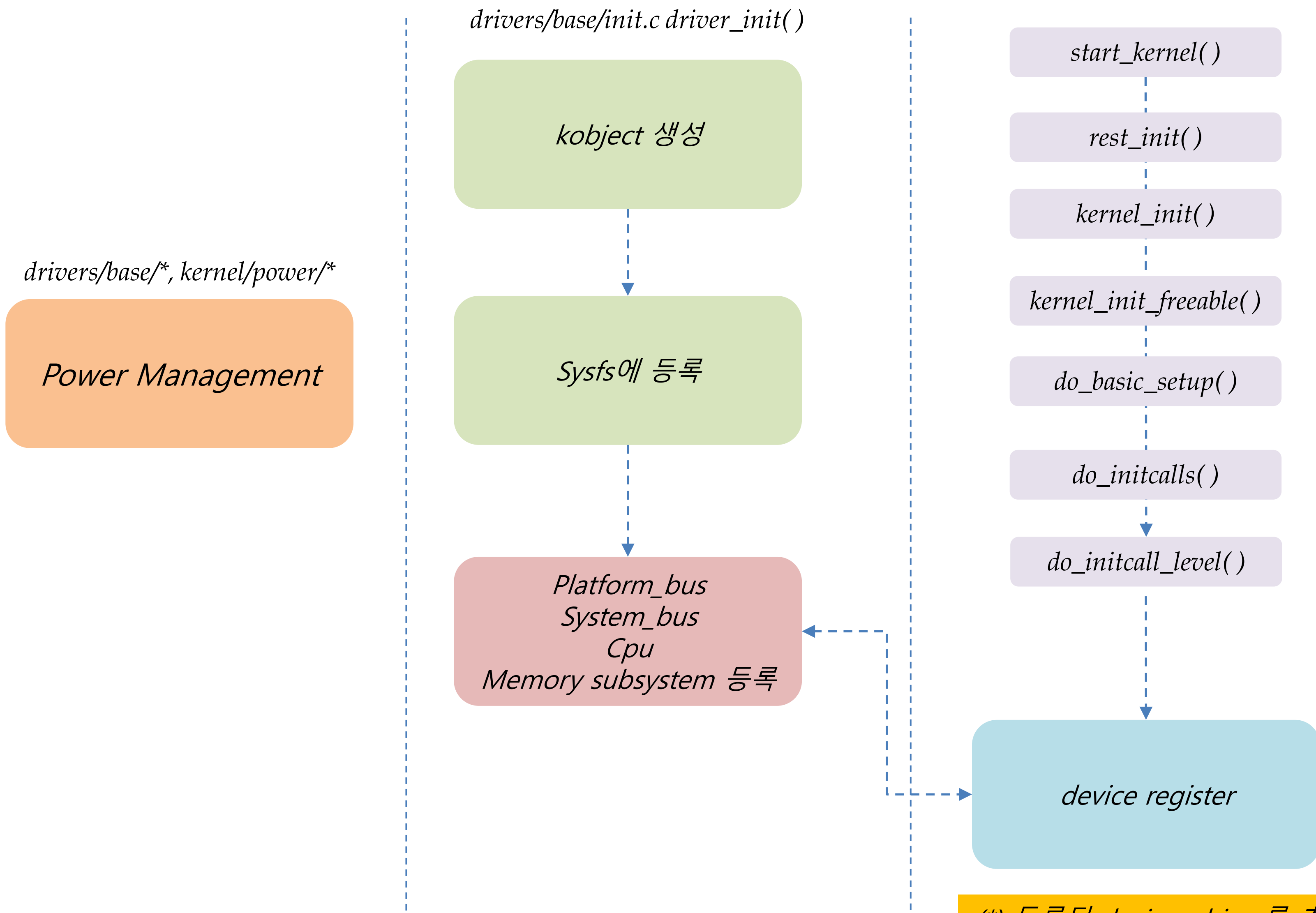
arch_initcall( )

customize_machine( )

machine_desc->init_machine( )

(*) chip dependent codes 초기화
=> 드라이버 혹은 초기화 코드

5. Kernel 초기화 과정(5) - 디바이스 드라이버 초기화



(*) 등록된 device driver를 초기화 !

5. Kernel 초기화 과정(6) - Init process 실행(마지막 step)

LISTING 5-11 Final Kernel Boot Steps from `main.c`

```
static noinline int init_post(void)
    __releases(kernel_lock)
{
<... lines trimmed for clarity ...>
...
if (execute_command) {
      run_init_process(execute_command);
      printk(KERN_WARNING "Failed to execute %s.  Attempting "
                        "defaults...\n", execute_command);
}

run_init_process("/sbin/init");
run_init_process("/etc/init");
run_init_process("/bin/init");
run_init_process("/bin/sh");

panic("No init found.  Try passing init= option to kernel.");
```

(*) ARM linux kernel의 boot flow 관련하여 자세한 정보를 알고 싶으면, 참고 문헌 [5]를 참고하기 바람.

6. Kernel Module 테스트

- [실습 2] Host(virtualbox)에서 build 후, target board에서 실행시켜 보기

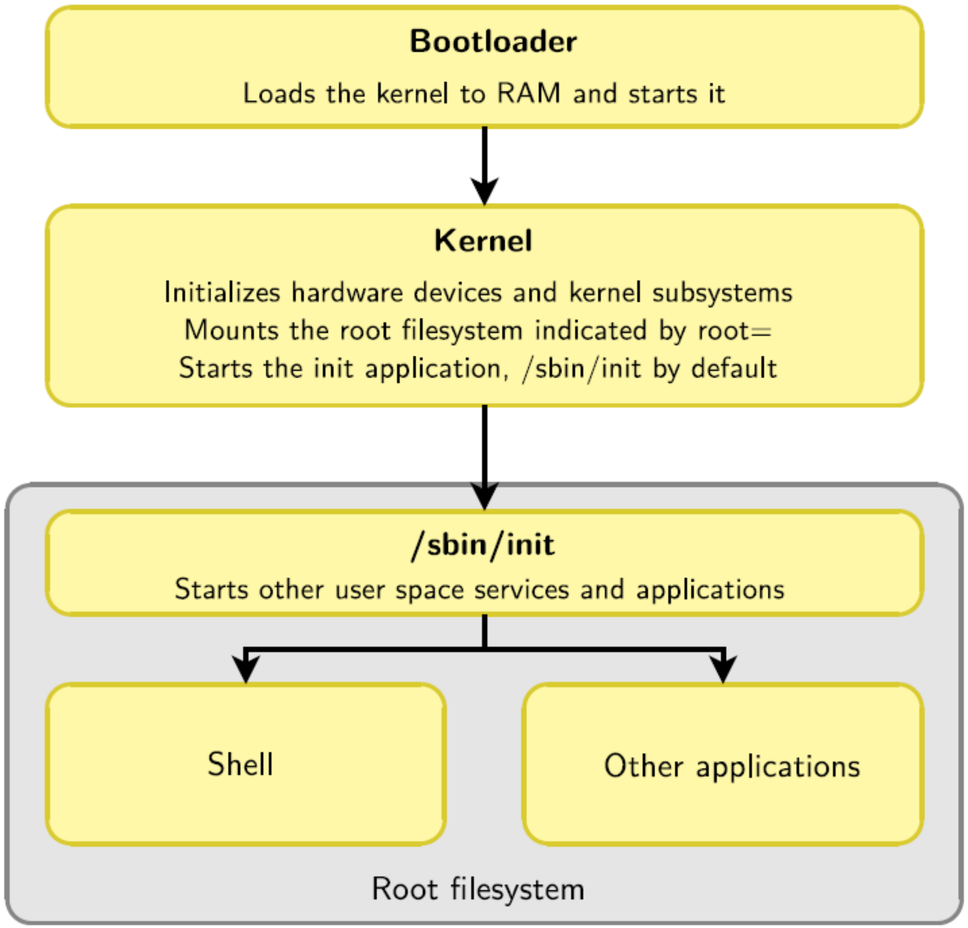
- [실습용 source code] http://www.coopj.com/LDD/
  - LDD_SOLUTIONS.tar.bz2

(*) 간략히 kernel module programming 기법 소개하자.

Chapter 6.

: init process & 사용자 영역 초기화 과정

1. Init process(1)



1. Init process(2)

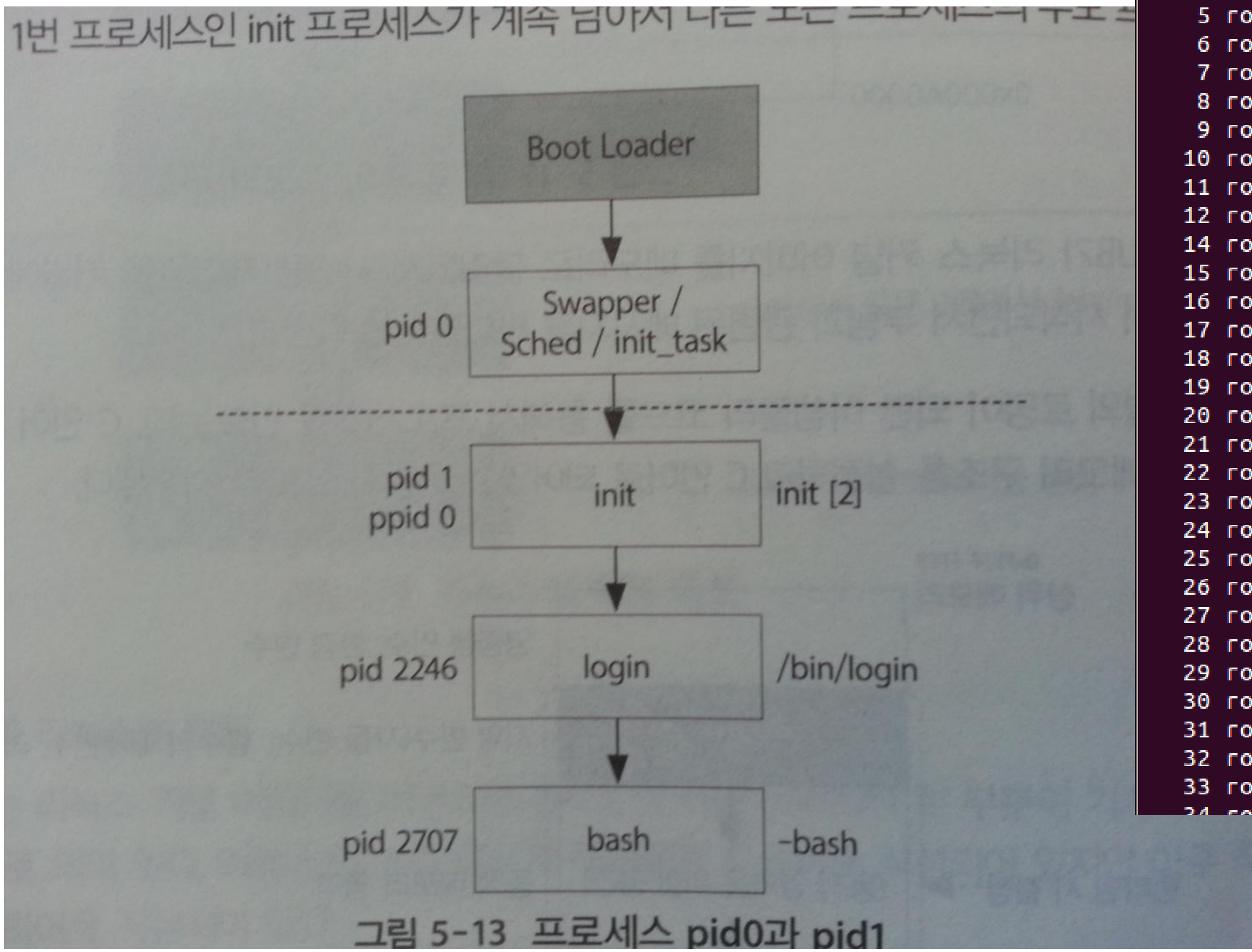


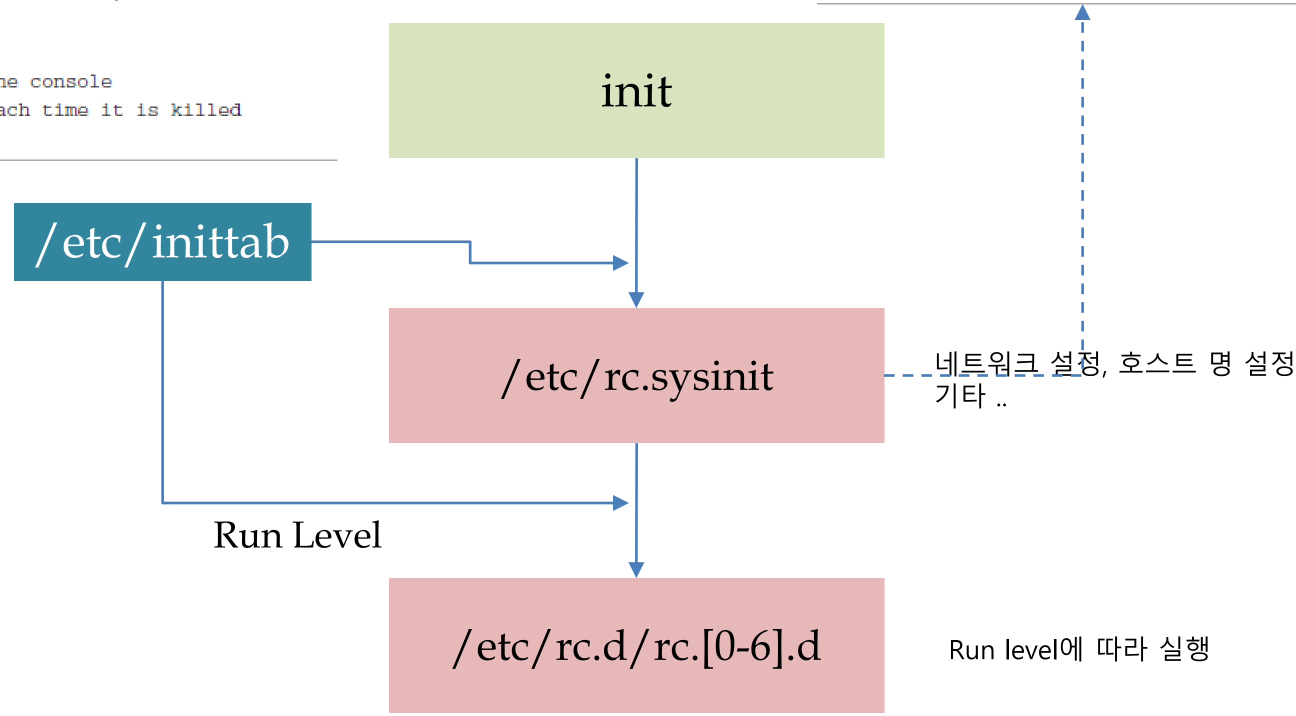
그림 5-13 프로세스 pid0과 pid1

```
# ps
PID   USER       COMMAND
    1 root       init
    2 root       [kthreadd]
    3 root       [ksoftirqd/0]
    4 root       [kworker/0:0]
    5 root       [kworker/0:0H]
    6 root       [kworker/u8:0]
    7 root       [rcu_preempt]
    8 root       [rcu_sched]
    9 root       [rcu_bh]
   10 root       [migration/0]
   11 root       [migration/1]
   12 root       [ksoftirqd/1]
   14 root       [kworker/1:0H]
   15 root       [migration/2]
   16 root       [ksoftirqd/2]
   17 root       [kworker/2:0]
   18 root       [kworker/2:0H]
   19 root       [migration/3]
   20 root       [ksoftirqd/3]
   21 root       [kworker/3:0]
   22 root       [kworker/3:0H]
   23 root       [khelper]
   24 root       [kdevtmpfs]
   25 root       [netns]
   26 root       [perf]
   27 root       [khungtaskd]
   28 root       [writeback]
   29 root       [crypto]
   30 root       [bioset]
   31 root       [kblockd]
   32 root       [kworker/1:1]
   33 root       [rpciod]
```

1. Init process(3) – inittab & runlevel(1)

```
# This is the first process (actually a script) to be run.
si::sysinit:/etc/rc.sysinit

# Execute our shutdown script on entry to runlevel 0
l0:0:wait:/etc/init.d/sys.shutdown

# Execute our normal startup script on entering runlevel 2
l2:2:wait:/etc/init.d/runlvl2.startup

# This line executes a reboot script (runlevel 6)
l6:6:wait:/etc/init.d/sys.reboot

# This entry spawns a login shell on the console
# Respawn means it will be restarted each time it is killed
con:2:respawn:/bin/sh
```

```
#!/bin/sh

echo "This is rc.sysinit"

busybox mount -t proc none /proc

# Load the system loggers
/sbin/syslogd
/sbin/klogd

# Enable legacy PTY support for telnetd
busybox mkdir /dev/pts
busybox mknod /dev/ptmx c 5 2
busybox mount -t devpts devpts /dev/pts
```



1. Init process(3) – inittab & runlevel(2)

TABLE 6-2 Runlevels

Runlevel	Purpose
0	System shutdown (halt)
1	Single-user system configuration for maintenance
2	User-defined
3	General-purpose multiuser configuration
4	User-defined
5	Multiuser with graphical user interface on startup
6	System restart (reboot)

2. Root file system

TABLE 6-1 Top-Level Directories

Directory	Contents
bin	Binary executables, usable by all users on the system[1]
dev	Device nodes (see Chapter 8, "Device Driver Basics")
etc	Local system configuration files
home	User account files
lib	System libraries, such as the standard C library and many others
sbin	Binary executables usually reserved for superuser accounts on the system
tmp	Temporary files
usr	A secondary file system hierarchy for application programs, usually read-
var	Contains variable files, such as system logs and temporary configuration

The very top of the Linux file system hierarchy is referenced by the slash charac itself. For example, to list the contents of the root directory, you would type t

```
$ ls /
```

This produces a listing similar to the following:

```
root@coyote:/# ls /
bin  dev  etc  home  lib  mnt  opt  proc  root  sbin  tmp  usr  var
root@coyote:/#
```

<rootfs를 구성하는 최소 파일>

LISTING 6-1 Contents of a Minimal Root File System

```
.
|-- bin
|   |-- busybox
|   `-- sh -> busybox
|-- dev
|   `-- console
|-- etc
|   `-- init.d
|       `-- rcS
`-- lib
    |-- ld-2.3.2.so
    |-- ld-linux.so.2 -> ld-2.3.2.so
    |-- libc-2.3.2.so
    `-- libc.so.6 -> libc-2.3.2.so

5 directories, 8 files
```

(*) rootfs 디렉토리의 내용을 살펴 보자.

3. Rootfs in Memory(1) - initrd(ramdisk)(1)

LISTING 6-12 Contents of a Sample initrd

```
.
|-- bin
|   |-- busybox
|   |-- echo -> busybox
|   |-- mount -> busybox
|   '-- sh -> busybox
|-- dev
|   |-- console
|   |-- ram0
|   '-- ttyS0
|-- etc
|-- linuxrc
'-- proc

4 directories, 8 files
```

LISTING 6-11 Sample linuxrc File

```
#!/bin/sh

echo 'Greetings: this is 'linuxrc' from Initial Ramdisk'
echo 'Mounting /proc filesystem'
mount -t proc /proc /proc

busybox sh
```

3. Rootfs in Memory(1) - initrd(ramdisk)(2)

```
console=ttyS0,115200 root=/dev/nfs                          \
   nfsroot=192.168.1.9:/home/chris/sandbox/omap-target      \
   initrd=0x10800000,0x14af47
```

<initrd 사용 cmdline>

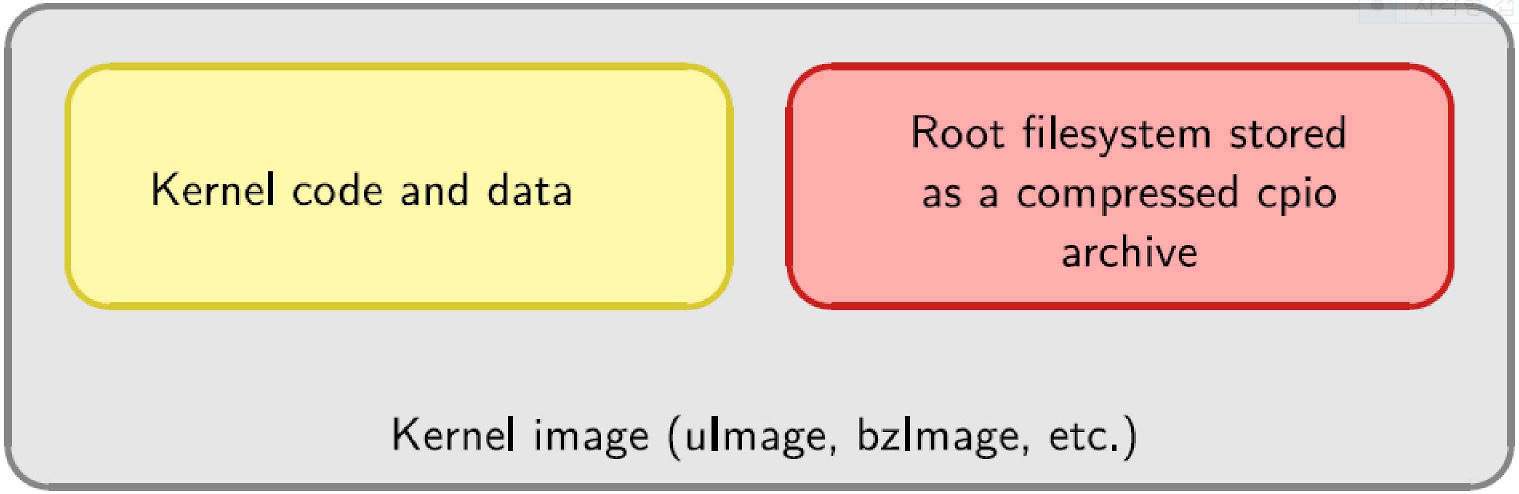
Initrd를 root file system으로 하여, 부팅 !

LISTING 6-10 Booting the Kernel with Ramdisk Support

```
[uboot]> tftp 0x10000000 kernel-uImage
...
Load address: 0x10000000
Loading: ############################ done
Bytes transferred = 1069092 (105024 hex)

[uboot]> tftp 0x10800000 initrd-uboot
...
Load address: 0x10800000
Loading: ########################################### done
Bytes transferred = 282575 (44fcf hex)

[uboot]> bootm 0x10000000 0x10800040
Uncompressing kernel................done.
...
RAMDISK driver initialized: 16 RAM disks of 16384K size 1024 blocksize
...
RAMDISK: Compressed image found at block 0
VFS: Mounted root (ext2 filesystem).
Greetings: this is linuxrc from Initial RAMDisk
Mounting /proc filesystem

BusyBox v1.00 (2005.03.14-16:37+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# (<<<< Busybox command prompt)
```

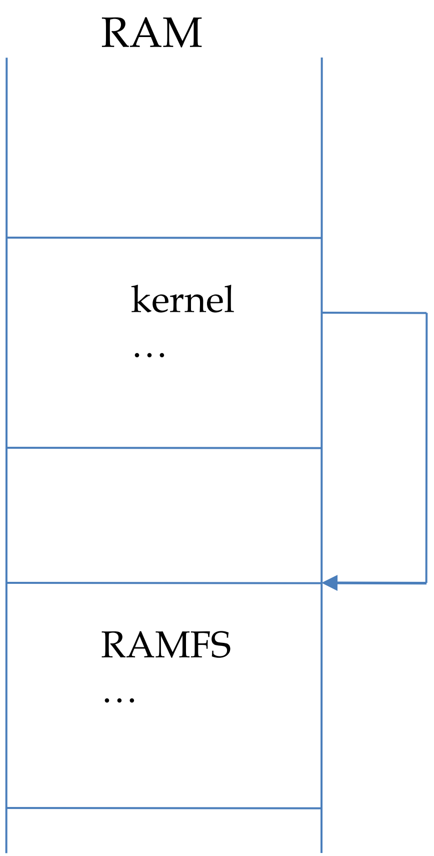
Initrd는 예전 방법이며,
Initramfs가 최신 방법임.

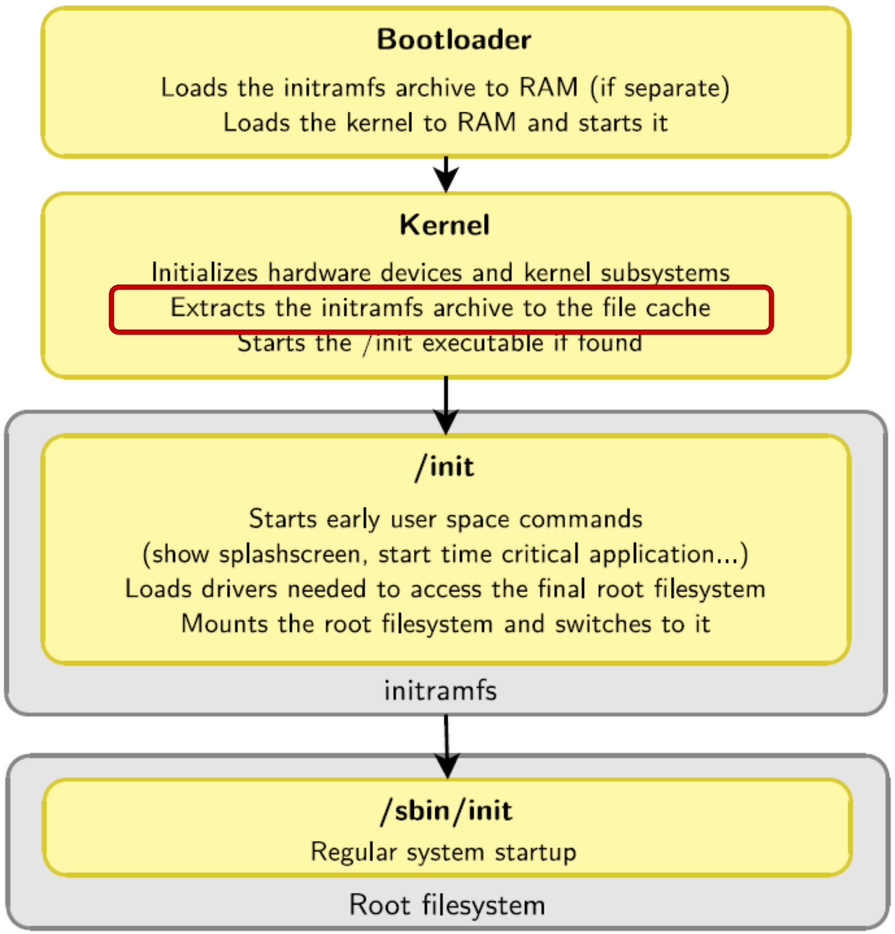
3. Rootfs in Memory(2) – initramfs(1)



uboot> tftp 0x10000000 uImage

3. Rootfs in Memory(2) – initramfs(2)





Chapter 9-11.

: 주요 File systems & Busybox

1. File Systems

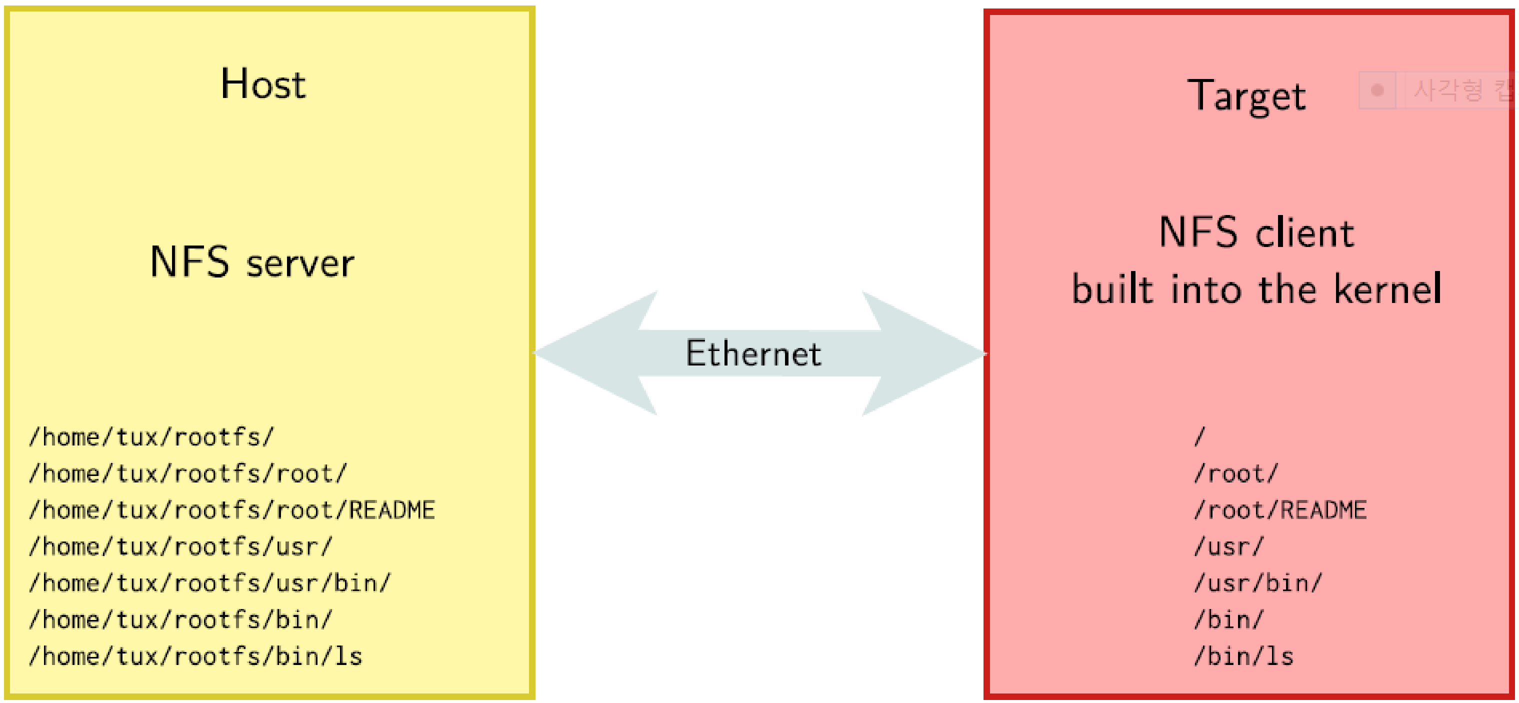
(*) ubifs에 관하여 사용해 보아야 함.

2. Pseudo File System – RAM file system

- /proc
- /sysfs
- /tmpfs

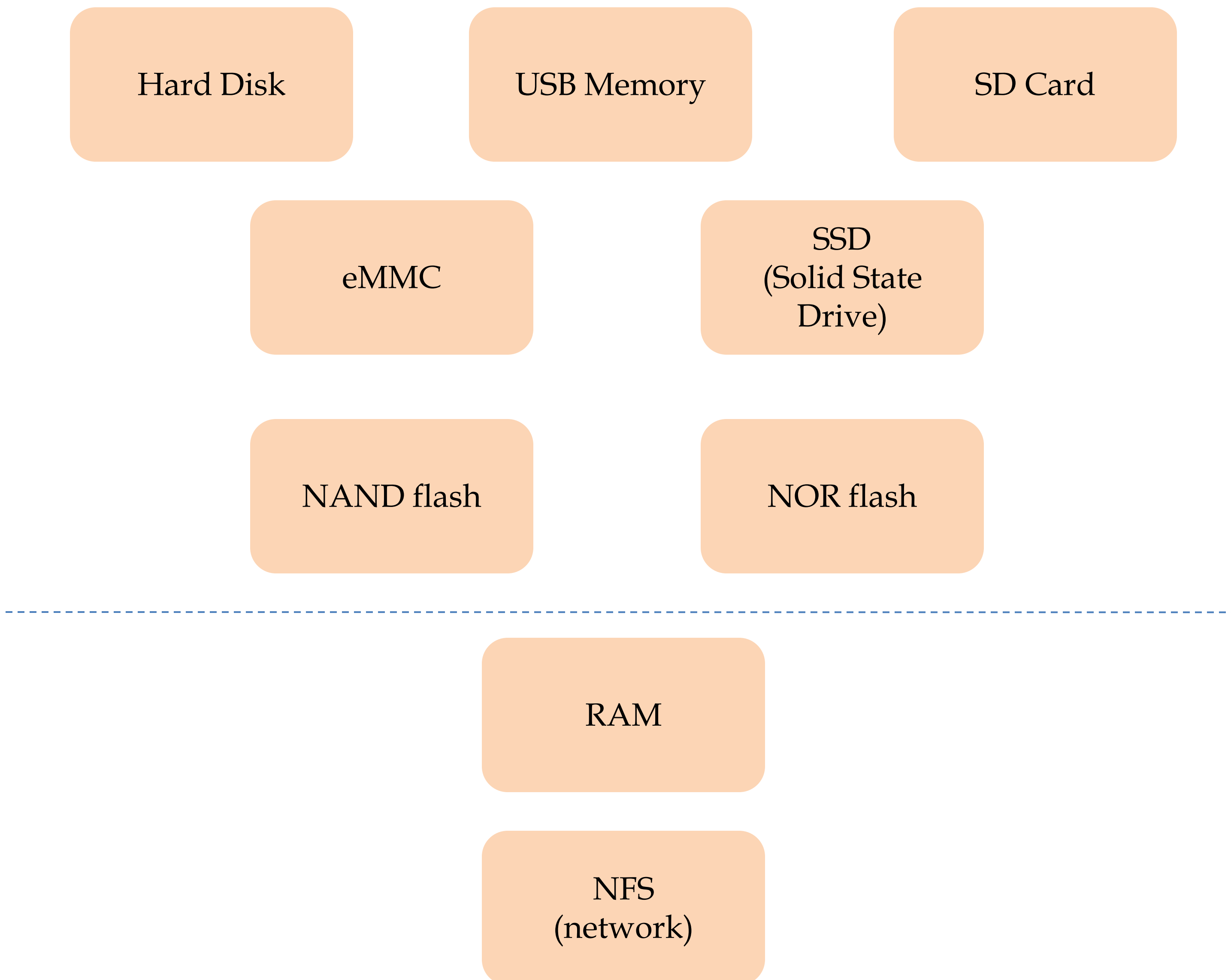
(*) 실제 RPi에 login하여 내용을 보여주자.

3. NFS(Network File System)



숙제 5: nfs server 설정을 해 볼 것.

4. Block vs Flash Devices(1)

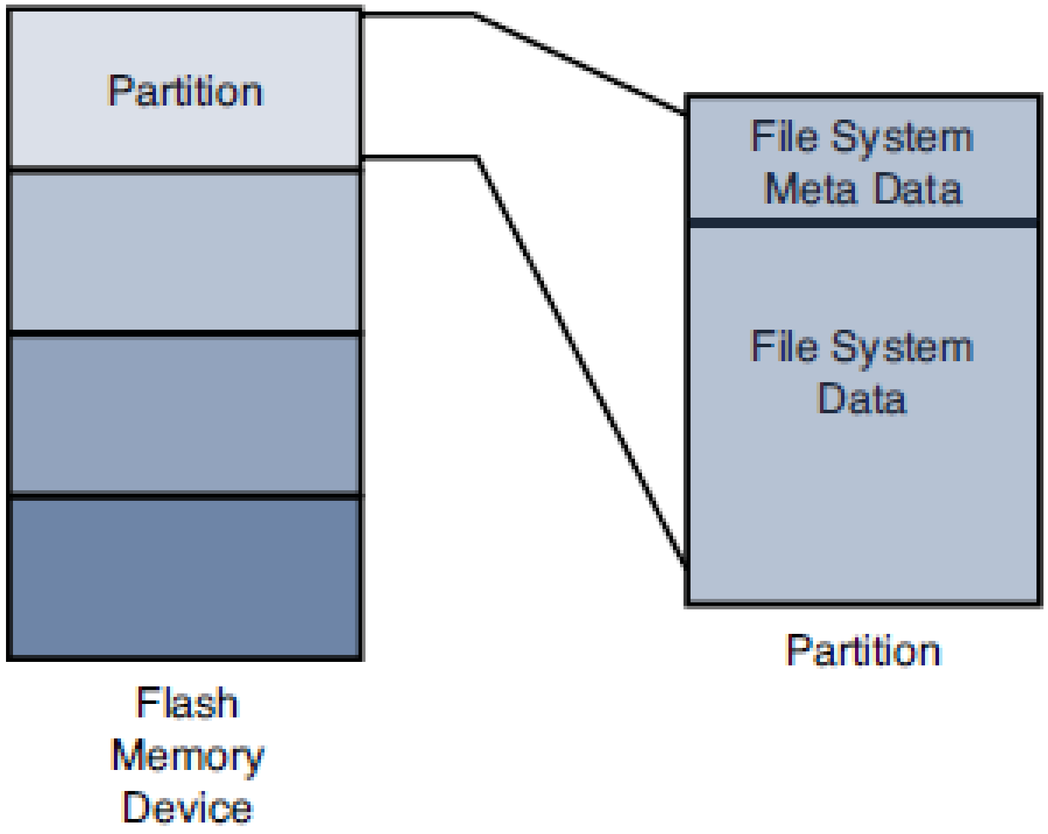


4. Block vs Flash Devices(2)

- Storage devices are classified in two main types: **block devices** and **flash devices**
  - They are handled by different subsystems and different filesystems

- **Block devices** can be read and written to on a per-block basis, without erasing.
  - Hard disks, floppy disks, RAM disks
  - USB keys, Compact Flash, SD card: these are based on flash storage, but have an integrated controller that emulates a block device, managing and erasing flash sectors in a transparent way.
- **Raw flash devices** are driven by a controller on the SoC. They can be read, but writing requires erasing, and often occurs on a larger size than the "block" size.
  - NOR flash, NAND flash

5. Partitions



6. ext2 file system(1) – 파티션 생성

(1) 파티션 생성

LISTING 9-1 Displaying Partition Information Using ***fdisk***

```
# fdisk /dev/sdb

Command (m for help): p

Disk /dev/sdb: 49 MB, 49349120 bytes
4 heads, 32 sectors/track, 753 cylinders
Units = cylinders of 128 * 512 = 65536 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1   *           1         180       11504   83  Linux
/dev/sdb2             181         360       11520   83  Linux
/dev/sdb3             361         540       11520   83  Linux
/dev/sdb4             541         753       13632   83  Linux
```

6. ext2 file system(2) - ext2 file system 생성 및 mount

(2) file system 생성

LISTING 9-2 Formatting a Partition Using **mkfs.ext2**

```
# mkfs.ext2 /dev/sdb1 -L CFlash_Boot_Vol
mke2fs 1.40.8 (13-Mar-2008)
Filesystem label=CFlash_Boot_Vol
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
2880 inodes, 11504 blocks
575 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=11796480
2 block groups
8192 blocks per group, 8192 fragments per group
1440 inodes per group
Superblock backups stored on blocks:
        8193

Writing inode tables: done
Writing superblocks and filesystem accounting i

This filesystem will be automatically checked e
days, whichever comes first.  Use tune2fs -c or
```

(3) file system mount 후, 사용

```
# mount /dev/sdb1 /mnt/flash
```

(4) file 시스템 오류 검사
=> unmount 상태에서 해야 함.

LISTING 9-5 Corrupted File System Check

```
# e2fsck -y /dev/sdb1
e2fsck 1.40.8 (13-Mar-2008)
/dev/sdb1 was not cleanly unmounted, check forced.
Pass 1: Checking inodes, blocks, and sizes
Inode 13, i_blocks is 16, should be 8.  Fix? yes

Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information

/dev/sdb1: ***** FILE SYSTEM WAS MODIFIED *****
/dev/sdb1: 25/2880 files (4.0% non-contiguous), 488/11504 blocks
#
```

(*) ext3, ext4는 ext2에서 확장된 버전임.

7. MTD(Memory Technology Devices)(1)

1) MTD는 raw flash device와의 interface를 제공하는 device driver 계층

2) MTD는 block device는 아님.

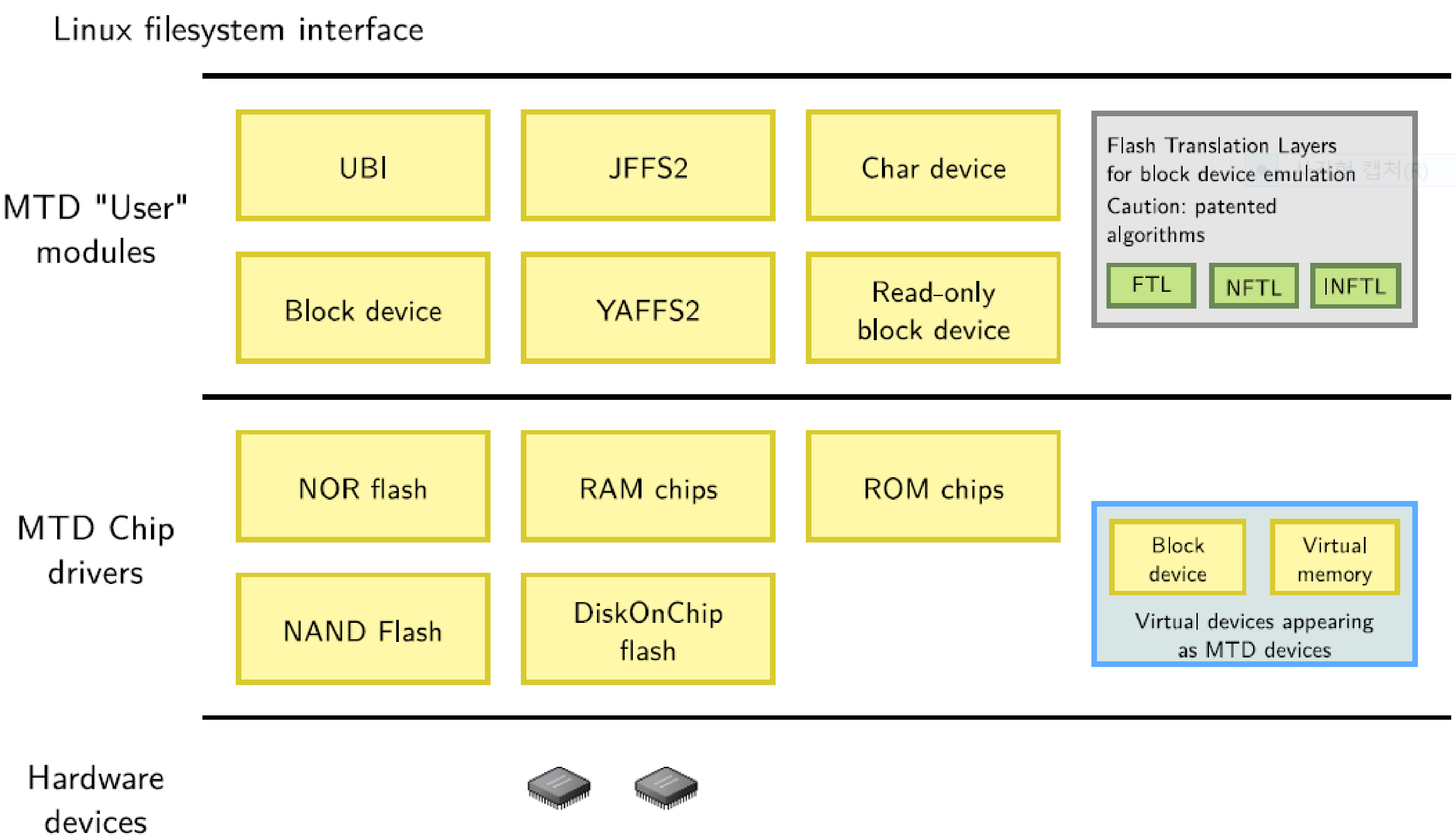
3) MTD는 일정한 크기가 정해지지 않은 erase block 단위로 동작하지만, Block device의 경우는 고정된 크기의 read/write block(섹터라고 함) 단위로 동작함.

4) Block device는 read, write operation만 있으나, MTD는 read, write, **erase** operation이 있음.

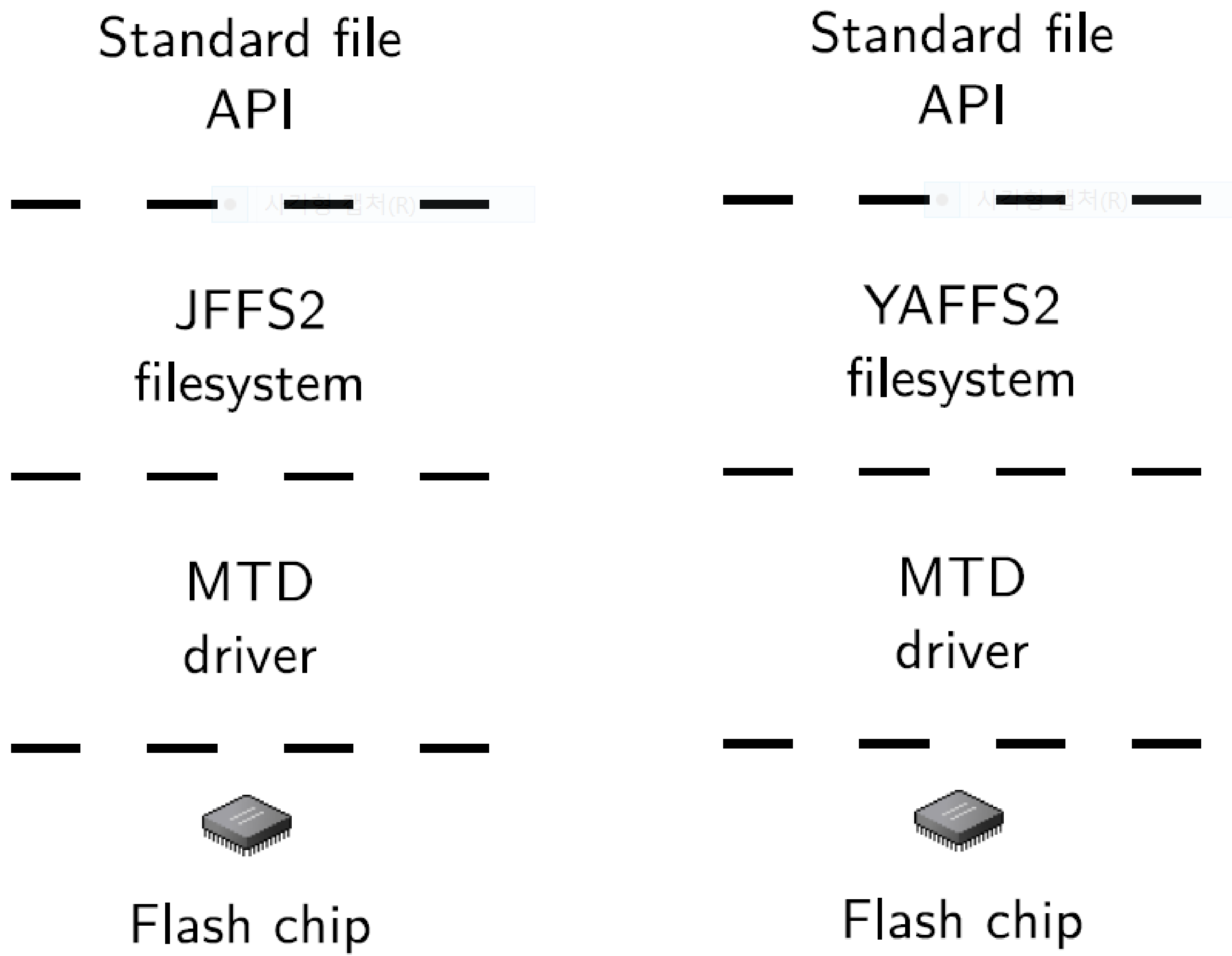
5) 일반적으로 알고 있는 내용과는 달리 SD/MMC cards, CompactFlash cards, USB flash drive 등은 MTD 장치가 아님.

6) 대부분의 linux device driver는 character 아니면 block device 형태이지만, MTD는 이와는 전혀 다른 개념(unique architecture)임.
=> 다만, translation mechanism에 의해, MTD가 character나 block device 처럼 보이기는 함.

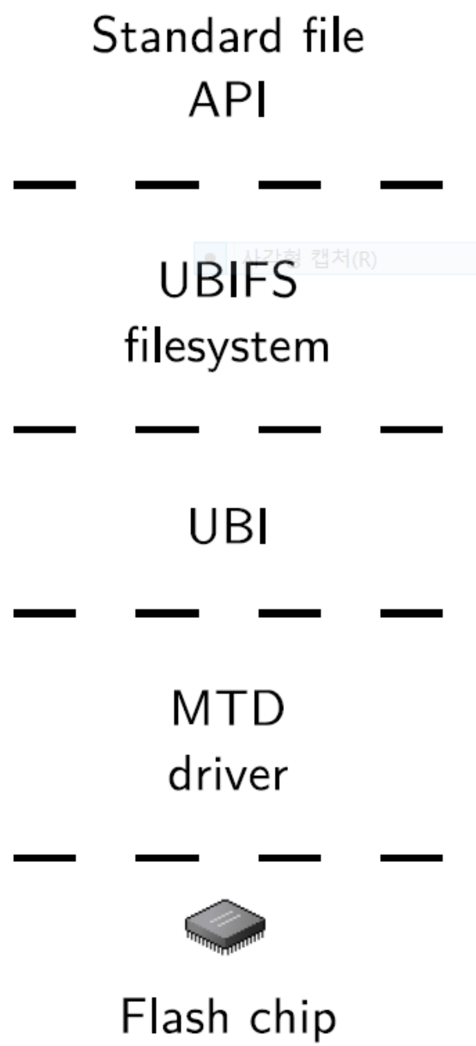
7. MTD(Memory Technology Devices)(2)



7. MTD(Memory Technology Devices)(3) - jffs2, yaffs2



7. MTD(Memory Technology Devices)(4) - UBIFS



(*) 현재 flash file system 중에서 가장 안정적임 !

숙제 6: UBIFS에 관하여 확인해 볼 것.

7. MTD(Memory Technology Devices)(5) – MTD 기본(1)

- <jffs2 이미지 파일 생성>
- mkfs.jffs2 –d ./jffs2-image-dir –o jffs2.bin

- <jffs2 이미지 NAND flash writing & mount>
- dd if=jffs2.bin of=/dev/mtdblock0
- mkdir /mnt/flash
- mount –t jffs2 /dev/mtdblock0 /mnt/flash

- <NAND flash로 부터 jffs2 image 추출하기>
- dd if=/dev/mtdblock0 of=./your-modified-fs-image.bin

7. MTD(Memory Technology Devices)(5) – MTD 기본(2)

LISTING 10-3 Redboot Messages on Power-Up

```
Platform: ADI Coyote (XScale)
IDE/Parallel Port CPLD Version: 1.0
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x04000000, 0x0001f960-0x03fd1000 available
FLASH: 0x50000000 - 0x51000000, 128 blocks of 0x00020000 bytes each.
...
```

This console output tells us that RAM on this board is physically mapped starting at address `0x00000000` and that Flash is mapped at physical address `0x50000000` through `0x51000000`. We can also see that Flash has 128 blocks of `0x00020000` (128KB) each.

Redboot contains a command to create and display partition information stored on Flash. Listing 10-4 is the output of the `fis list` command, part of the Flash Image System family of commands available in the Redboot bootloader.

LISTING 10-4 Redboot Flash Partition List

```
RedBoot> fis list
Name              FLASH addr   Mem addr     Length       Entry point
RedBoot           0x50000000   0x50000000   0x00060000   0x00000000
```

1) Bootloader에서 MTD 파티션을 만든다.

LISTING 10-4 Continued

```
RedBoot config    0x50FC0000   0x50FC0000   0x00001000   0x00000000
FIS directory     0x50FE0000   0x50FE0000   0x00020000   0x00000000
RedBoot>
```

7. MTD(Memory Technology Devices)(5) – MTD 기본(3)

LISTING 10-5 Detecting Redboot Partitions on Linux Boot

```
...
IXP4XX-Flash.0: Found 1 x16 devices at 0x0 in 16-bit bank
 Intel/Sharp Extended Query Table at 0x0031
Using buffer write method
Searching for RedBoot partition table in IXP4XX-Flash.0 at offset 0xfe0000
3 RedBoot partitions found on MTD device IXP4XX-Flash0
Creating 3 MTD partitions on "IXP4XX-Flash.0":
0x00000000-0x00060000 : "RedBoot"
0x00fc0000-0x00fc1000 : "RedBoot config"
0x00fe0000-0x01000000 : "FIS directory"
...
```

2) kernel에서 MTD 파티션을 인식한다.

LISTING 10-6 Kernel MTD Flash Partitions

```
root@coyote:~# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00060000 00020000 "RedBoot"
mtd1: 00001000 00020000 "RedBoot config"
mtd2: 00020000 00020000 "FIS directory"
#
```

7. MTD(Memory Technology Devices)(5) – MTD 기본(4)

LISTING 10-7 Creating a New Redboot Partition

```
RedBoot> load -r -v -b 0x01008000 coyote-40-zImage
Using default protocol (TFTP)
Raw file loaded 0x01008000-0x0114dccb, assumed entry at 0x01008000
RedBoot> fis create -b 0x01008000 -l 0x145cd0 -f 0x50100000 MyKernel
... Erase from 0x50100000-0x50260000: ..........
... Program from 0x01008000-0x0114dcd0 at 0x50100000: ....
... Unlock from 0x50fe0000-0x51000000: .
... Erase from 0x50fe0000-0x51000000: .
... Program from 0x03fdf000-0x03fff000 at 0x50fe0000: .
... Lock from 0x50fe0000-0x51000000: .
```

First, we load the image to be used to create the new partition. We use o
image for the example and load it to memor
new partition using the Redboot `fis create`
the new partition in an area of Flash starting
Redboot first erases this area of Flash and th
sequence, Redboot unlocks its directory area and updates the FIS Directory
new partition information. Listing 10-8 shows the output of `fis list` with
partition. Compare this with the output shown in Listing 10-4.

Bootloader에서 Kernel용 MTD 파티션을 만들고, Kernel image(coyote-40-zImage)를 flash에 Write한다.

LISTING 10-8 New Redboot Partition List

```
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
RedBoot           0x50000000  0x50000000  0x00060000  0x00000000
RedBoot config    0x50FC0000  0x50FC0000  0x00001000  0x00000000
FIS directory     0x50FE0000  0x50FE0000  0x00020000  0x00000000
MyKernel          0x50100000  0x50100000  0x00160000  0x01008000
```

7. MTD(Memory Technology Devices)(5) – MTD 기본(5)

LISTING 10-9 Kernel Command-Line MTD Partition Format

```
mtdparts=<mtddef>[;<mtddef]
 * <mtddef>   := <mtd-id>:<partdef>[,<partdef>]
 * <partdef>  := <size>[@offset][<name>][ro]
 * <mtd-id>   := unique name used in mapping driver/device (mtd->name)
 * <size>     := std linux memsize OR "-" to denote all remaining space
 * <name>     := '(' NAME ')'
```

```
mtdparts=MainFlash:384K(Redboot),4K(config),128K(FIS),-(unused)
```

MTD partition 정보를 command line을 써서, kernel에 전달한다.

7. MTD(Memory Technology Devices)(5) – MTD 기본(6)

kernel codes

LISTING 10-10 PQ2FADS Flash Mapping Driver

```
...
static struct mtd_partition pq2fads_partitions[] = {
        {
#ifdef CONFIG_ADS8272
                .name         = "HRCW",
                .size         = 0x40000,
                .offset         = 0,
                .mask_flags = MTD_WRITEABLE,   /* force read-only */
        }, {
                .name         = "User FS",
                .size         = 0x5c0000,
                .offset         = 0x40000,
#else
```


Kernel 드라이버 코드에서도 MTD partition을 지정해 준다.

LISTING 10-10 Continued

```
                .name         = "User FS",
                .size         = 0x600000,
                .offset         = 0,
#endif
        }, {
                .name         = "uImage",
                .size         = 0x100000,
                .offset         = 0x600000,
                .mask_flags = MTD_WRITEABLE,   /* force read-only */
        }, {
                .name         = "bootloader",
                .size         = 0x40000,
                .offset       = 0x700000,
                .mask_flags = MTD_WRITEABLE,   /* force read-only */
        }, {
                .name         = "bootloader env",
                .size         = 0x40000,
                .offset                = 0x740000,
                .mask_flags = MTD_WRITEABLE,   /* force read-only */
        }
```

7. MTD(Memory Technology Devices)(5) – MTD 기본(7)

- <MTD utils – kernel에서 실행>
- flash_erase /dev/mtd1
- flashcp /workspace/coyote-40-zImage /dev/mtd1
- flashcp /rootfs.ext2 /dev/mtd2

(*) mtd utils를 사용하면, kernel에 nand operation(erase, read, write)이 가능하다.

LISTING 10-16 Booting with JFFS2 as the Root File System

```
RedBoot> load -r -v -b 0x01008000 coyote-zImage
Using default protocol (TFTP)
Raw file loaded 0x01008000-0x0114decb, assumed entry at 0x01008000
RedBoot> exec -c "console=ttyS0,115200 rootfstype=jffs2 root=/dev/mtdblock2"
Using base address 0x01008000 and length 0x00145ecc
Uncompressing Linux...... done, booting the kernel.
...
```

8. Busybox(1)

LISTING 11-3 BusyBox Usage

```
root@coyote # busybox
BusyBox v1.13.2 (2010-02-24 16:04:14 EST) multi-call binary
Copyright (C) 1998-2008 Erik Andersen, Rob Landley, Denys Vlasenko and
others. Licensed under GPLv2.
See source distribution for full notice.

Usage: busybox [function] [arguments]...
   or: function [arguments]...

    BusyBox is a multi-call binary that combines many common Unix
    utilities into a single executable.  Most people will create a link
    to busybox for each function they wish to use and BusyBox will act
    like whatever it was invoked as!

Currently defined functions:
    [, [[, addgroup, adduser, ar, ash, awk, basename, blkid, bunzip2,
    bzcat, cat, chattr, chgrp, chmod, chown, chpasswd, chroot, chvt,
    clear, cmp, cp, cpio, cryptpw, cut, date, dc, dd, deallocvt,
    delgroup, deluser, df, dhcprelay, diff, dirname, dmesg, du,
    dumpkmap, dumpleases, echo, egrep, env, expr, false, fbset,
    fbsplash, fdisk, fgrep, find, free, freeramdisk, fsck, fsck.minix,
    fuser, getopt, getty, grep, gunzip, gzip, halt, head, hexdump,
    hostname, httpd, hwclock, id, ifconfig, ifdown, ifup, init, insmod,
    ip, kill, killall, klogd, last, less, linuxrc, ln, loadfont,
    loadkmap, logger, login, logname, logread, losetup, ls, lsmod,
    makedevs, md5sum, mdev, microcom, mkdir, mkfifo, mkfs.minix, mknod,
    mkswap, mktemp, modprobe, more, mount, mv, nc, netstat, nice,
    nohup, nslookup, od, openvt, passwd, patch, pidof, ping, ping6,
    pivot_root, poweroff, printf, ps, pwd, rdate, rdev, readahead,
    readlink, readprofile, realpath, reboot, renice, reset, rm, rmdir,
    rmmod, route, rtcwake, run-parts, sed, seq, setconsole, setfont,
    sh, showkey, sleep, sort, start-stop-daemon, strings, stty, su,
    sulogin, swapoff, swapon, switch_root, sync, sysctl, syslogd, tail,
    tar, tee, telnet, telnetd, test, tftp, time, top, touch, tr,
    traceroute, true, tty, udhcpc, udhcpd, umount, uname, uniq, unzip,
    uptime, usleep, vi, vlock, watch, wc, wget, which, who, whoami,
    xargs, yes, zcat
```

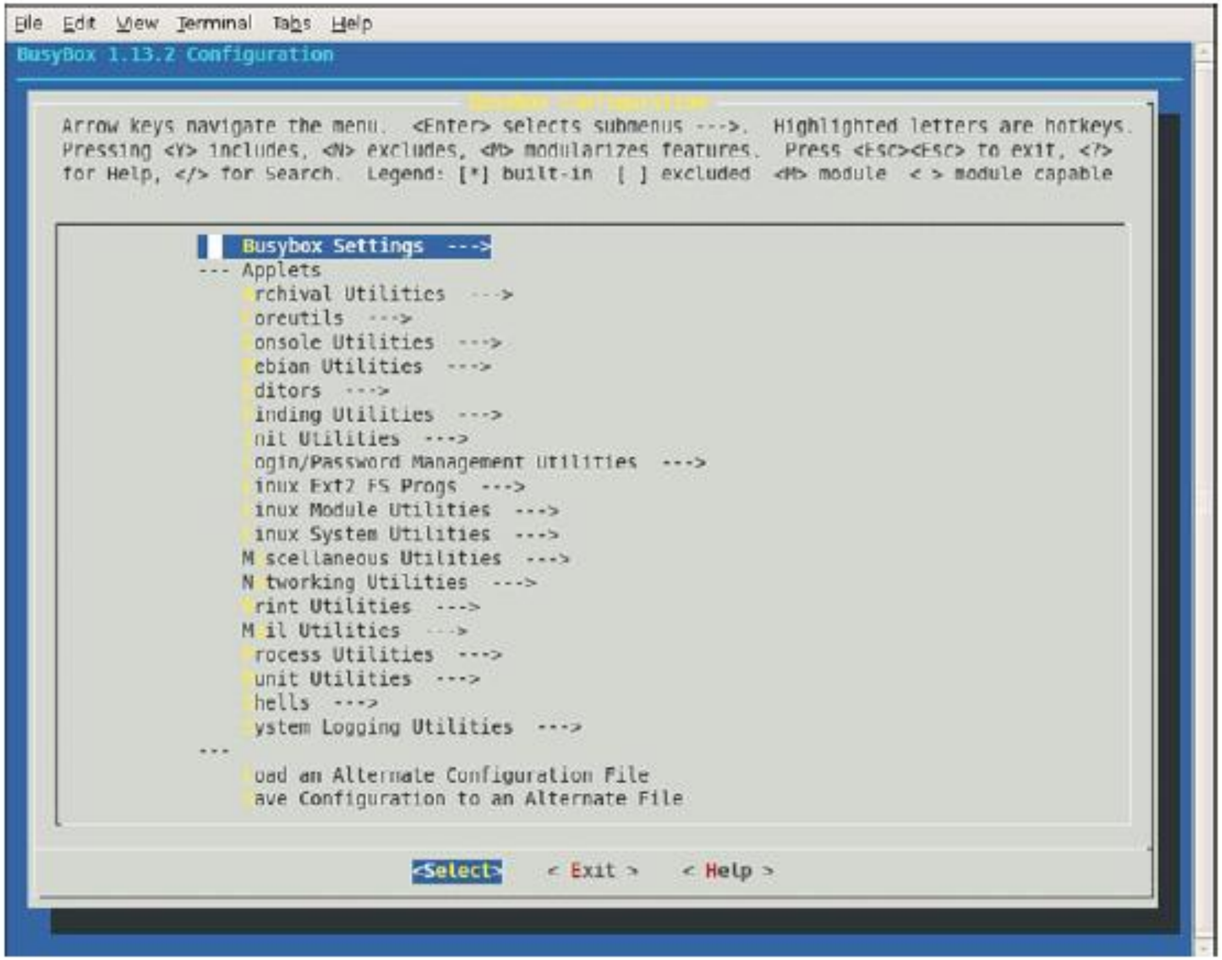
LISTING 11-5 BusyBox Symlink Structure: Tree Detail

```
[root@coyote]$ tree
.
|-- bin
|   |-- addgroup -> busybox
|   |-- busybox
```


LISTING 11-5 Continued

```
|   |-- cat -> busybox
|   |-- cp -> busybox
<...>
|   `-- zcat -> busybox
|-- linuxrc -> bin/busybox
|-- sbin
|   |-- halt -> ../bin/busybox
|   |-- ifconfig -> ../bin/busybox
|   |-- init -> ../bin/busybox
|   |-- klogd -> ../bin/busybox
<...>
|   `-- syslogd -> ../bin/busybox
`-- usr
    |-- bin
    |   |-- [ -> ../../bin/busybox
    |   |-- basename -> ../../bin/busybox
<...>
    |   |-- xargs -> ../../bin/busybox
    |   `-- yes -> ../../bin/busybox
    `-- sbin
        `-- chroot -> ../../bin/busybox
```

8. Busybox(2) - menuconfig



8. Busybox(3) – build 방법

- wget http://busybox.net/downloads/busybox-1.23.2.tar.bz2 tar -xjf busybox-1.23.2.tar.bz2

- cd busybox-1.23.2/
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- defconfig
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- menuconfig

- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- install CONFIG_PREFIX=/home/export/rootfs

숙제 7: busybox source를 download하고, build해 볼 것.

Chapter 12-15.

: Cross-Development Environment

: Debugging Techniques

1. Cross 개발 환경

- 1) Serial console & minicom
- 2) dhcp server
- 3) tftp server
- 4) NFS server

숙제 8: 위의 tool을 한번씩 사용해 보기 !

2. Debugging

- GDB, DDD
- gdbserver
- ctags, cscope(C 개발 환경)
- strace, ltrace
- ps, top
- …

숙제 9: 위의 tool을 한번씩 사용해 보기 !

(*) debugging 기법 관련하여 보다 자세한 사항은 참고 문헌 [7] 참조 !

3. Kernel Panic & Oops

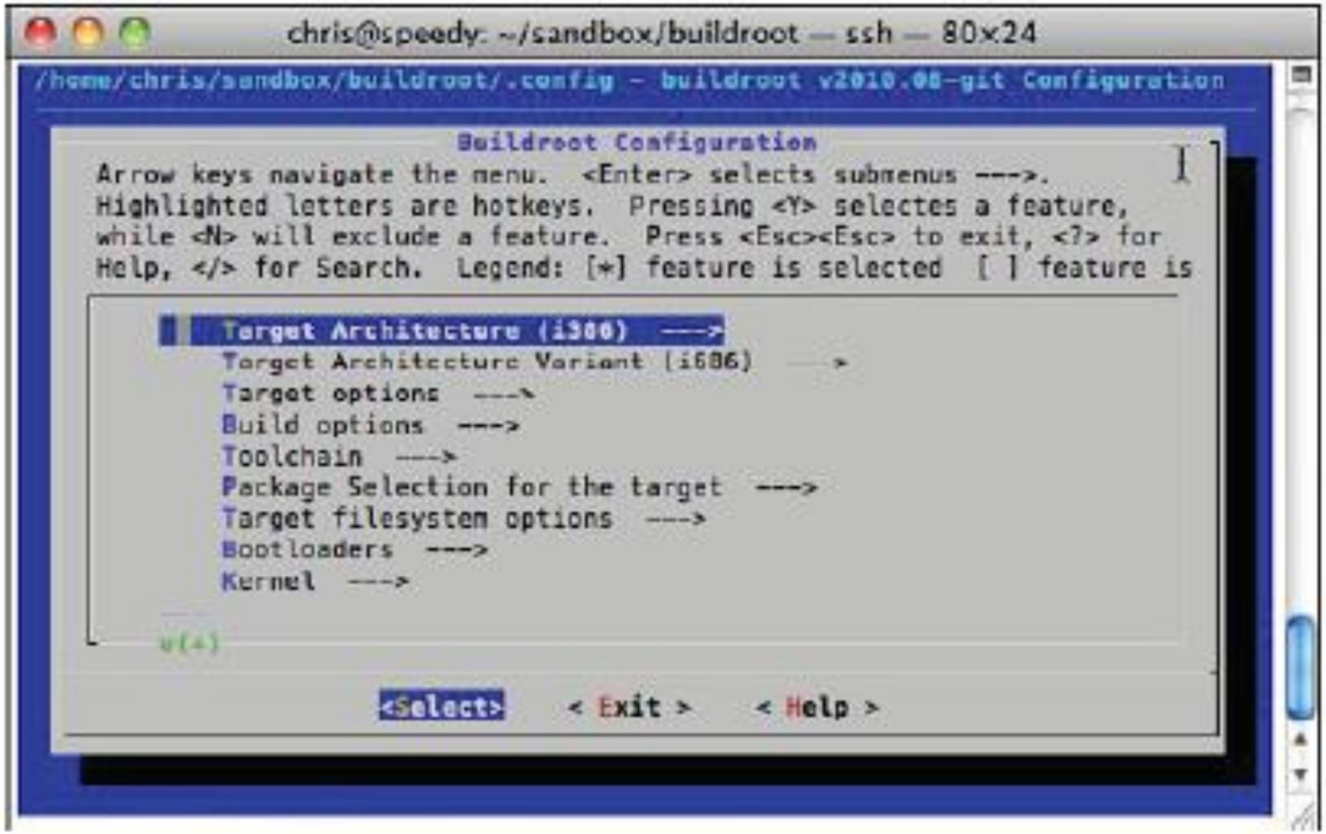
LISTING 13-14 Kernel Oops Display

```
$ modprobe loop
Oops: kernel access of bad area, sig: 11 [#1]
NIP: C000D058 LR: C0085650 SP: C7787E80 REGS: c7787dd0 TRAP: 0300  Not tainted
MSR: 00009032 EE: 1 PR: 0 FP: 0 ME: 1 IR/DR: 11
DAR: 00000000, DSISR: 22000000
TASK = c7d187b0[323] 'modprobe' THREAD: c7786000
Last syscall: 128
GPR00: 0000006C C7787E80 C7D187B0 00000000 C7CD25CC FFFFFFFF 00000000 80808081
GPR08: 00000001 C034AD80 C036D41C C034AD80 C0335AB0 1001E3C0 00000000 00000000
GPR16: 00000000 00000000 00000000 100170D8 100013E0 C9040000 C903DFD8 C9040000
GPR24: 00000000 C9040000 C9040000 00000940 C778A000 C7CD25C0 C7CD25C0 C7CD25CC
NIP [c000d058] strcpy+0x10/0x1c
LR [c0085650] register_disk+0xec/0xf0
Call trace:
 [c00e170c] add_disk+0x58/0x74
 [c90061e0] loop_init+0x1e0/0x430 [loop]
 [c002fc90] sys_init_module+0x1f4/0x2e0
 [c00040a0] ret_from_syscall+0x0/0x44
Segmentation fault
```

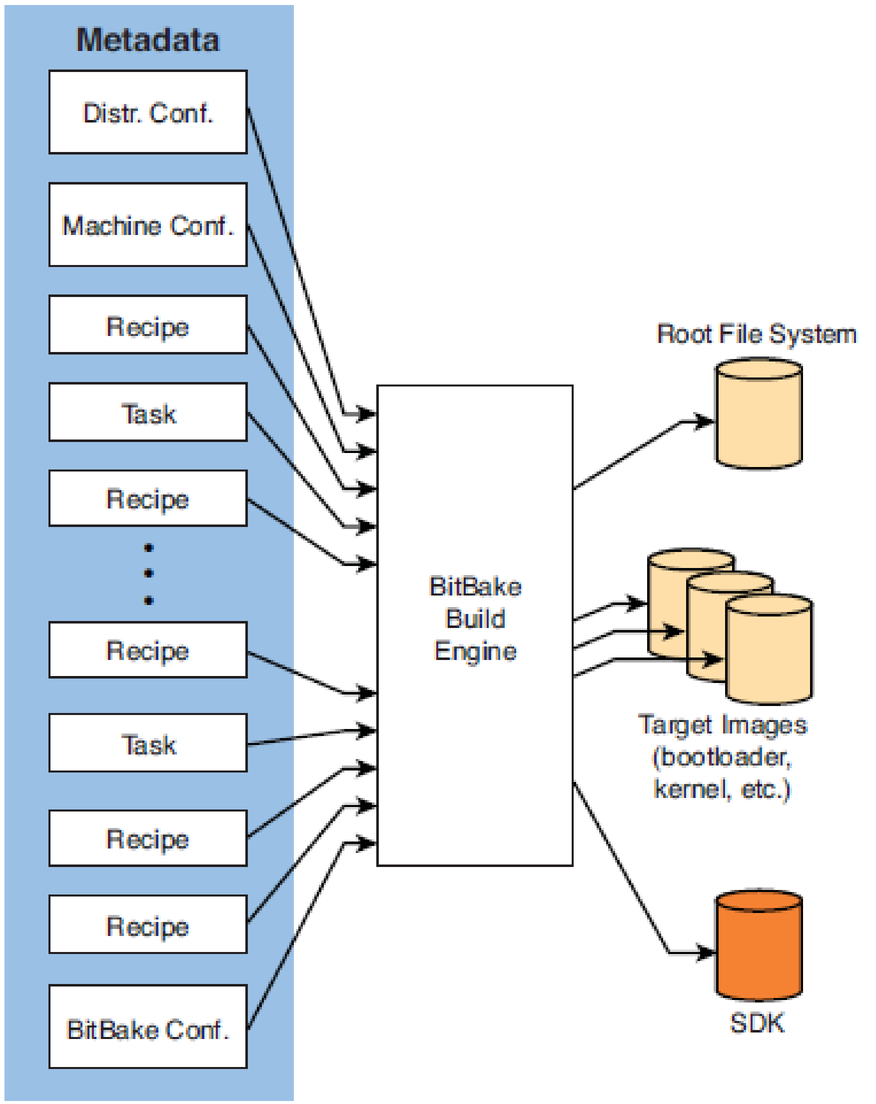
Chapter 16.

: open source build systems

1. Buildroot


chris@speedy: ~/sandbox/buildroot — ssh — 80×24
/home/chris/sandbox/buildroot/.config - buildroot v2010.08-git Configuration
Buildroot Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> selectes a feature,
while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for
Help, </> for Search. Legend: [*] feature is selected [ ] feature is
Target Architecture (i386) --->
Target Architecture Variant (i686) --->
Target options --->
Build options --->
Toolchain --->
Package Selection for the target --->
Target filesystem options --->
Bootloaders --->
Kernel --->
v(+)
<Select> < Exit > < Help >

2. Yocto(Open Embedded)



2. Yocto(Open Embedded) – Recipe Example

LISTING 16-4 Simple OpenEmbedded Recipe: ***hello_1.0.0.bb***

```
DESCRIPTION = "Hello demo project"
PR = "r0"
LICENSE = "GPL"

SRC_URI = "http://localhost/sources/hello-1.0.0.tar.gz"

SRC_URI[md5sum] = "90a8ffd73e4b467b6d4852fb95e493b9"
SRC_URI[sha256sum] = "fd626b829cf1df265abfceac37c2b5629f2ba8fbc3897add29f-
9661caa40fe12"

do_install() {
        install -m 0755 -d ${D}${bindir}
        install -m 0755 ${S}/hello ${D}${bindir}/hello
}
```

2. Yocto(Open Embedded) – BitBake

LISTING 16-5 BitBake Hello Recipe Processing

```
chris@speedy:~/sandbox/build01$ bitbake hello
<...>
NOTE: Executing runqueue
NOTE: Running task 10 of 38 (ID: 5, NOTE: Running task 10 of 38 (ID: 5,
/hello_1.0.0.bb, do_fetch)
NOTE: Running task 11 of 38 (ID: 0, /hello_1.0.0.bb, do_unpack)
NOTE: Running task 15 of 38 (ID: 1, /hello_1.0.0.bb, do_patch)
NOTE: Running task 16 of 38 (ID: 7, /hello_1.0.0.bb, do_configure)
NOTE: Running task 17 of 38 (ID: 8, /hello_1.0.0.bb, do_compile)
NOTE: Running task 18 of 38 (ID: 2, /hello_1.0.0.bb, do_install)
NOTE: Running task 19 of 38 (ID: 10, /hello_1.0.0.bb, do_package)
NOTE: Running task 25 of 38 (ID: 13, /hello_1.0.0.bb, do_package_write_ipk)
NOTE: Running task 26 of 38 (ID: 9, /hello_1.0.0.bb, do_package_write)
NOTE: Running task 29 of 38 (ID: 3, /hello_1.0.0.bb, do_populate_sysroot)
NOTE: Running task 30 of 38 (ID: 12, /hello_1.0.0.bb, do_package_stage)
NOTE: Running task 37 of 38 (ID: 11, /hello_1.0.0.bb, do_package_stage_all)
NOTE: Running task 38 of 38 (ID: 4, /hello_1.0.0.bb, do_build)
NOTE: Tasks Summary: Attempted 38 tasks of which 25 didn't need to be rerun and 0
failed.
```

References

[1] Embedded Linux Primer 2nd Edition, Hallinan, Prentice Hall

[2] 사물인터넷 리눅스 프로그래밍 with 라즈베리파이, 서영진, Jpub

[3] embedded-linux-slides.pdf, free electrons
=> *http://free-electrons.com/doc/training/embedded-linux/*

[4] petazzoni-device-tree-dummies.pdf, free electrons
=> *https://events.linuxfoundation.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf*

[5] 코드로 알아보는 ARM 리눅스 커널, Jpub

[6] http://free-electrons.com/doc/training/linux-kernel/linux-kernel-slides.pdf

[7] AndroidKernelHacks2-7.pdf, http://slowbootkernelhacks.blogspot.kr/

추천 도서

<Linux kernel & device drivers programming>

[1] 리눅스 디바이스 드라이버, 한빛 미디어(저자: 유영창)
  => 오래된 책이나, 유용함.

[2] Linux Device Drivers 3rd Edition, Oreilly
  => 한글 번역판 있음.

[3] Linux Kernel Development 3rd Edition - Addison Wesley 출판사(저자: Rovert Love)
  => 한글 번역판 있음. 이와 유사한 주제를 다루는 기타 교제도 가능함.

[4] 아트멜 스튜디오와 아두이노로 배우는 ATmega328 프로그래밍 - Jpub 출판사(저자: 허경용)
  => ATmega328이 목적이 아니라, Embedded의 기초를 확립하기 위한 내용이 잘 기술되어 있어, 소개함.
  => CPU 및 주변 장치(UART, SPI, I2C, GPIO, Interrupt 등) 개념 이해

<Linux 사용자 영역 programming>

[1] 유닉스 리눅스 프로그래밍 필수 유틸리티 - 한빛미디어 출판사(저자: 백창우)
  => Linux 개발 환경(vim, gcc, ld, make, subversion 등) 숙지 차원 !
  => 단, 여기에 git, bug tracking system, Yocto 등 추가 소개 필요함

[2] Advanced programming in the UNIX environment - Addison Wesley 출판사(저자: Rechard Stevens & Rago)

[3] Linux System Programming - Oreilly 출판사(저자: Rovert Love)
  => Userspace 영역과 관련된 내용으로, 이와 유사한 주제를 다루는 기타 교제도 가능함.