

제12강

U-Boot + GPIO 제어

시리얼 모니터링
부트로더
U-Boot 부트로더

GPIO 제어(LED, BTN)

준비물) USB-TTL 시리얼케이블, T-코블러, 브레드보드, LED 모듈, LAN 케이블

참조) 교재 제8장 전반부, 제12장

GPIO핀 확장보드

* 확장보드 구성

: T-Cobbler, 브레드보드, 40P 플랫케이블 활용

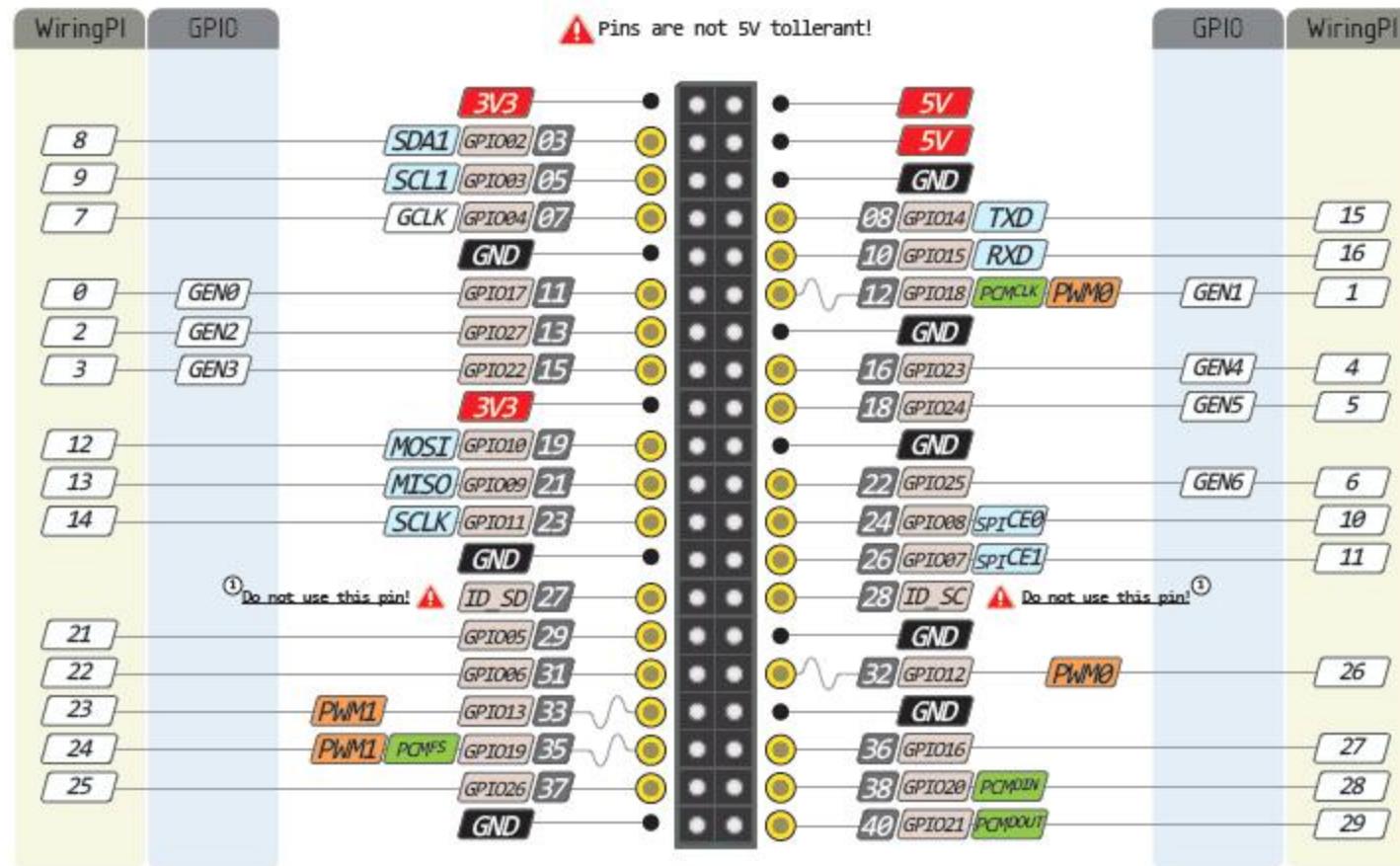


: 점퍼선 준비할 것!!(A210실에서, 30cm 20가닥 활용하여)

- 양끝 피복제거(30cmx10ea, 15cmx10ea, 10cmx15ea)

GPIO핀 확장보드(계속)

* 핀 레이아웃 (부록 1 참조)

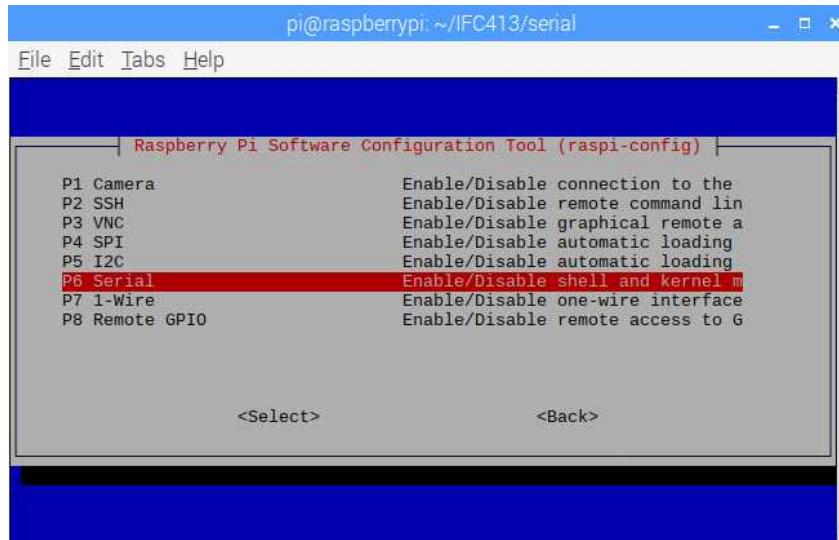


시리얼 모니터링

* 라즈베리파이 환경설정

\$ sudo raspi-config

: Interface Options – Serial 활성화



=> 시리얼통신을 위한 장치 파일 /dev/ttys0 생성

\$ sudo reboot // 재부팅

시리얼 모니터링(계속)

* 몇가지 확인사항

\$ ls /dev/tty* -al

```
crw--w---- 1 root tty      4,   8 Jul 11 10:48 /dev/tty8
crw--w---- 1 root tty      4,   9 Jul 11 10:48 /dev/tty9
crw-rw---- 1 root dialout 204, 64 Jul 11 10:48 /dev/ttymA0
crw--w---- 1 root tty      4,  64 Jul 11 10:48 /dev/ttys0
crw----- 1 root root     5,   3 Jul 11 10:48 /dev/ttypunkt
pi@raspberrypi:~ $
```

: 디바이스 파일 **/dev/ttys0** 이 추가 생성된 것을 확인

: 이 장치 파일의 내정된 접근권한 자는 슈퍼유저임을 유의

\$ cat /boot/config.txt

```
# Enable audio (loads snd_bcm2835)
dtparam=audio=on
start_x=1
gpu_mem=128
dtoverlay=w1-gpio
enable_uart=1
```

: **enable_uart=1**의 라인이 추가된 것을 확인

시리얼 모니터링(계속)

- * 몇가지 확인사항(계속)

\$ cat /boot/cmdline.txt

```
pi@raspberrypi:~/IFC413/08_UART $ cat /boot/cmdline.txt
dwc_otg.lpm_enable=0 console=serial0,115200 console=tty1 root=PARTUUID=5605427a-
02 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait
pi@raspberrypi:~/IFC413/08_UART $
```

: 내정된 baudrate는 115200 임을 확인

\$ dmesg | grep tty

```
pi@raspberrypi:~/IFC413/08_UART $ dmesg | grep tty
[    0.000000] Kernel command line: coherent_pool=1M 8250.nr_uarts=1 bcm2708_fb.
fbwidth=1280 bcm2708_fb.fbheight=1024 bcm2708_fb.fbswap=1 vc_mem.mem_base=0x3ec0
0000 vc_mem.mem_size=0x40000000 dwc_otg.lpm_enable=0 console=ttyS0,115200 conso
le=tty1 root=PARTUUID=5605427a-02 rootfstype=ext4 elevator=deadline fsck.repair=
yes rootwait
[    0.000908] console [tty1] enabled
[    1.029007] 3f201000.serial: ttyAMA0 at MMIO 0x3f201000 (irq = 81, base_baud
= 0) is a PL011 rev2
[    1.037691] console [ttyS0] disabled
[    1.044730] 3f215040.serial: ttyS0 at MMIO 0x0 (irq = 53, base_baud = 3125000
0) is a 16550
[    2.074783] console [ttyS0] enabled
pi@raspberrypi:~/IFC413/08_UART $
```

: ttyS0 활성화 확인 가능

시리얼 모니터링(계속)

* 시리얼 케이블 연결(USB-TTL 시리얼 케이블 활용)

- : 흰색(RxD) 핀을 rpi의 TxD(BCM_GPIO #14)와 연결
- : 녹색(TxD) 핀을 rpi의 RxD(BCM_GPIO #15)와 연결
- : 검정(Gnd) 핀을 Gnd와 연결
- : 빨강(Vcc) 핀을 5V와 연결(전용 전원공급기연결시 비연결)

* 드라이버 다운로드 및 설치 (선택사항)

<https://learn.adafruit.com/adafruits-raspberry-pi-lesson-5-using-a-console-cable/software-installation-windows>

* 시리얼포트 확인 (Windows 장치관리자에서)

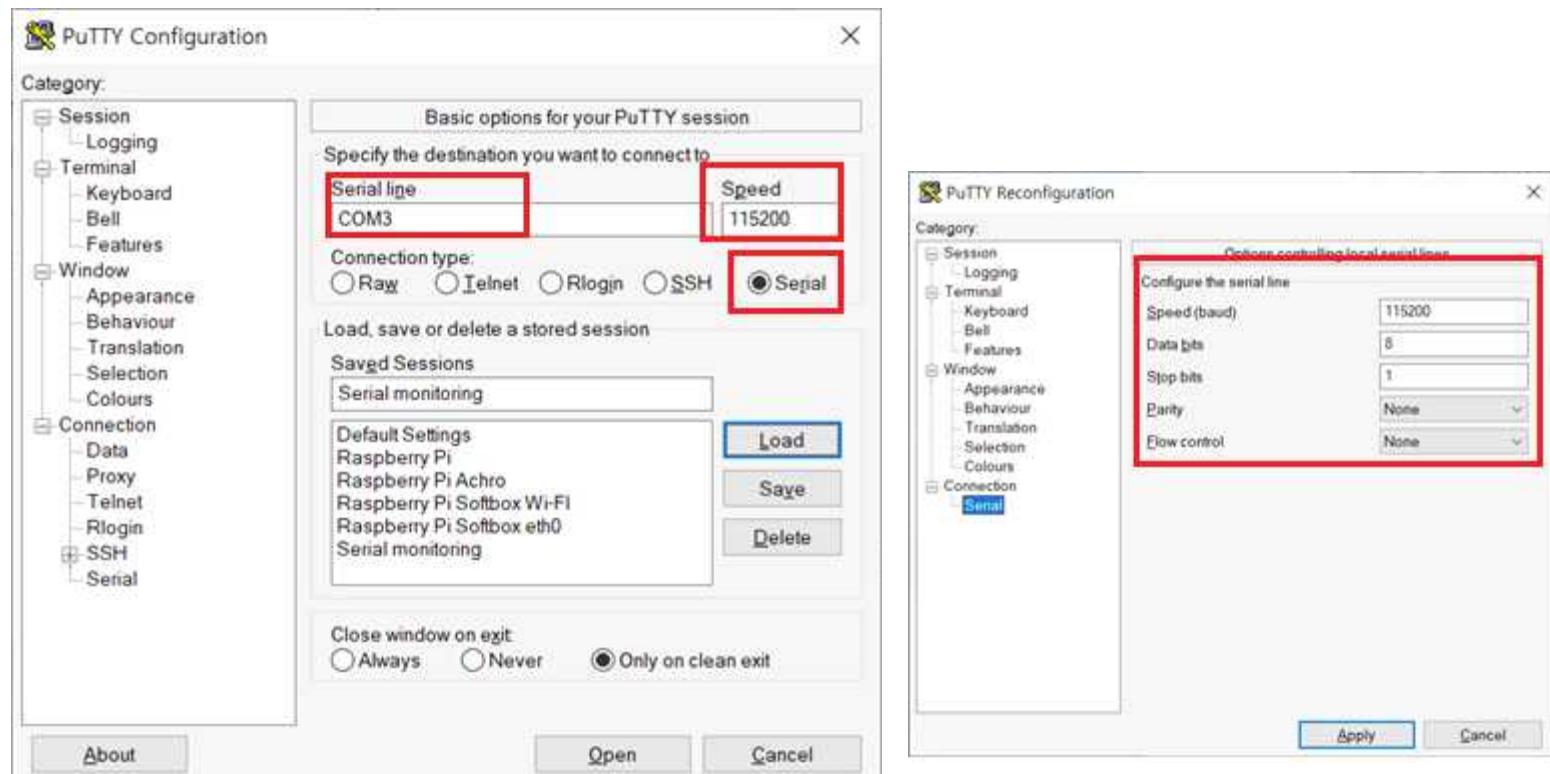


시리얼 모니터링(계속)

* PuTTY로 접속

: 포트번호, speed, 연결유형을 Serial로 선택

: 우측창 Flow control 항목을 필히 none으로 설정후, Open

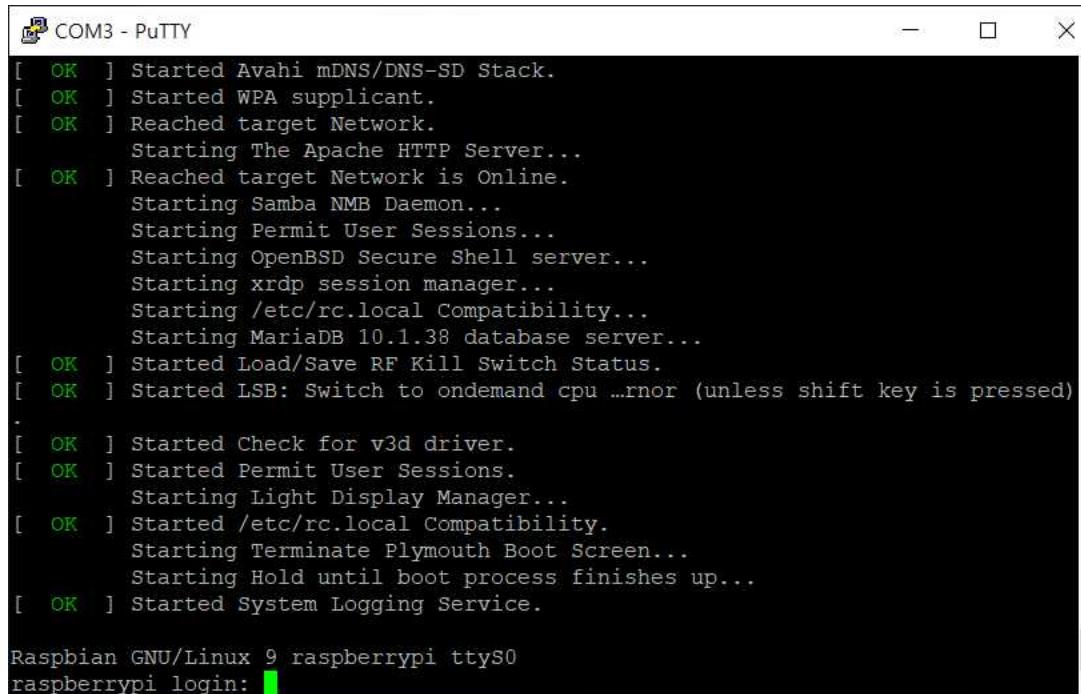


시리얼 모니터링(계속)

* 모니터링

: 라즈베리파이보드 전원 재투입, 혹은 임의 키 눌러봄

참고) SSH를 통해 새 터미널을 여는 것이 아님!!!



```
[ OK ] Started Avahi mDNS/DNS-SD Stack.  
[ OK ] Started WPA supplicant.  
[ OK ] Reached target Network.  
      Starting The Apache HTTP Server...  
[ OK ] Reached target Network is Online.  
      Starting Samba NMB Daemon...  
      Starting Permit User Sessions...  
      Starting OpenBSD Secure Shell server...  
      Starting xrdp session manager...  
      Starting /etc/rc.local Compatibility...  
      Starting MariaDB 10.1.38 database server...  
[ OK ] Started Load/Save RF Kill Switch Status.  
[ OK ] Started LSB: Switch to ondemand cpu ...rnor (unless shift key is pressed)  
. [ OK ] Started Check for v3d driver.  
[ OK ] Started Permit User Sessions.  
      Starting Light Display Manager...  
[ OK ] Started /etc/rc.local Compatibility.  
      Starting Terminate Plymouth Boot Screen...  
      Starting Hold until boot process finishes up...  
[ OK ] Started System Logging Service.  
  
Raspbian GNU/Linux 9 raspberrypi ttyS0  
raspberrypi login:
```

: 로그인후, 리눅스 명령 사용 가능

시리얼 모니터링(계속)

- * USB–시리얼케이블 인식불능시 조치

- : 현재 시리얼케이블 인식 불안정

- 1) 타깃보드의 전원 OFF

- 2) USB–시리얼케이블을 PC에 재연결후 인식확인

- 3) PuTTY로 시리얼접속

- 4) 타깃보드에 전원 인가

참고) Windows와 VM간 USB 시리얼 전환시

- : VM 플레이어 우상단 USB 아이콘 및 시리얼포트 아이콘 활용

- : 두 OS간 동시 사용 불가

부트로더

* boot loader

: 시스템 초기화후 OS로 부팅

: 부트로더의 기능

- 시스템초기화
- 시스템 동작에 필요한 환경 설정
- 운영체제 부팅
- 메모리 관리
- 시스템 모니터링 등

* boot loader 종류

: Lilo, GRUB, Loadlin, EtherBoot, Blob, PMON,
RedBoot, U-Boot 등

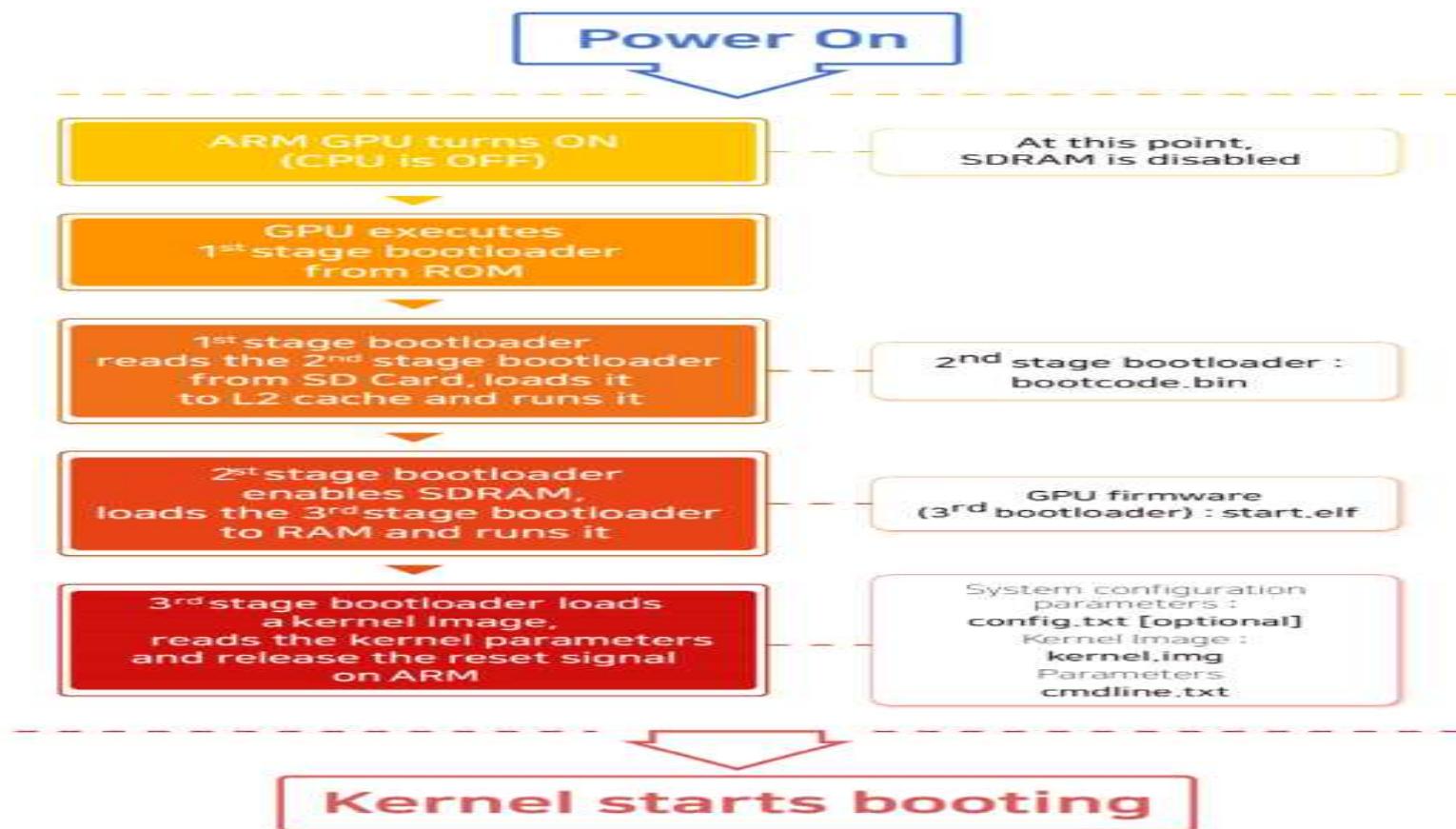
부트로더(계속)

- * /boot 디렉터리의 주요 파일
 - : 부팅과 관련된 파일들이 위치

```
$ cd /boot  
pi@raspberrypi:/boot $ ls -al  
-rwxr-xr-x 1 root root 26108 Jul 10 09:15 bcm2710-rpi-3-b-plus.dtb  
-rwxr-xr-x 1 root root 52296 Jul 10 09:15 bootcode.bin (2nd bootloader)  
-rwxr-xr-x 1 root root 142 Jan 21 2019 cmdline.txt  
-rwxr-xr-x 1 root root 1727 Jul 27 02:03 config.txt  
-rwxr-xr-x 1 root root 6701 Jul 10 09:15 fixup.dat  
-rwxr-xr-x 1 root root 5000184 Jul 10 09:15 kernel.img  
-rwxr-xr-x 1 root root 5281136 Jul 10 09:15 kernel7.img (kernel image)  
drwxr-xr-x 2 root root 14336 Jul 10 09:15 overlays  
-rwxr-xr-x 1 root root 2873444 Jul 10 09:14 start.elf (3rd bootloader)
```

부트로더(계속)

* 라즈베리파이(ARM)의 부트로더 동작 흐름도



부트로더(계속)

* 부트로더 동작 순서

1) 1st stage bootloader(ROM에 위치)

: 제조과정에서 SoC에 프로그래밍(사용자에 의해 재 프로그래밍 불가)

: SD카드의 FAT32 부트 파티션 마운트후, 2nd 부트로더 실행

2) 2nd stage bootloader (**bootcode.bin**)

: fixedup.dat 참조하여 SDRAM 파티션 및 활성화

: 3rd 부트로더(**start.elf**)를 SDRAM에 적재후 실행

3) 3rd stage bootloader (**start.elf**)

: User code를 메모리에 적재하고 실행

*) User code(**리눅스커널, U-boot, 응용프로그램 등**)

: 이진형태의 파일

: 통상 kernel.img라 명명된 커널 이미지 및 부가 파일들

: 또 다른 부트로더(예, U-boot), 혹은 응용프로그램일 수 있음

U-Boot 부트로더

- * U-Boot를 실행할 때 할 일

- 1) 부트로더 소스 컴파일후 생성된 u-boot.bin을
/boot 디렉터리에 위치
- 2) u-boot.bin가 실행되도록 /boot/config.txt 편집
kernel=u-boot.bin 추가

- * 가상머신에서

: 로그인후 슈퍼유저로 전환

```
ifc415@ubuntu:~$ sudo su
root@ubuntu:/home/ifc415# cd
root@ubuntu:~#
```

U-Boot 부트로더(계속)

- * U-Boot 소스 다운로드 (방법1, 참고)

: git 툴을 활용하여 다운로드

```
# apt-get update
```

```
# apt-get upgrade
```

```
# apt-get install git // git 패키지 설치
```

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

// 현 작업디렉터리에 **./u-boot**로 다운로드 됨

```
# ls
```

```
./u-boot
```

U-Boot 부트로더(계속)

* 학습자료 활용 ([12_Uboot.tar](#))

: 다운로드하여 공유폴더 Shared에 위치시킨 후,

```
# mkdir IFC415                                // 교과 작업 디렉터리 생성
# cd IFC415
# ls /mnt/hgfs/Shared/
12_Uboot.tar .....
# cp /mnt/hgfs/Shared/12_Uboot.tar ./
# tar xvf 12_Uboot.tar                         // 풀기
# cd 12_Uboot
# ls
bareMetal  bootGpioLed.c  bootGpioLedBtn.c  u-boot.tar.gz
# tar xvfz u-boot.tar.gz                      // U-Boot 로더 소스 풀기
# ls
./u-boot
```

U-Boot 부트로더(계속)

* 소스 구성

:

```
# cd u-boot  
# ls
```

```
root@ubuntu:~/u-boot# ls  
api    common   doc        env      Kbuild   MAINTAINERS README  
arch   config.mk Documentation examples Kconfig  Makefile   scripts  
board  configs  drivers     fs       lib      net        test  
cmd    disk     dts        include  Licenses post      tools  
root@ubuntu:~/u-boot#
```

- ./configs** : 타깃보드용의 환경설정 파일들 위치
- ./arch** : 특정 아키텍쳐관련 소스
- ./examples** : 테스트용 예제

U-Boot 부트로더(계속)

- * U-Boot 환경설정파일

- : ./configs에 타깃보드용 환경설정파일들 위치

- : 라즈베리파이 3B+ 보드용은 **rpi_3_32b_defconfig**

```
# ls ./configs/rpi*
```

```
root@ubuntu:~/u-boot# ls ./configs/rpi*
./configs/rpi_0_w_defconfig  ./configs/rpi_3_32b_defconfig  ./configs/rpi_defconfig
./configs/rpi_2_defconfig    ./configs/rpi_3_defconfig
root@ubuntu:~/u-boot# █
```

U-Boot 부트로더(계속)

- * 라즈베리파이용 U-Boot 환경설정

: U-Boot 홈 디렉터리에서

```
# make rpi_3_32b_defconfig // 오류발생
```

참고) 오류발생시 조치 => bison, flex 패키지 설치

```
# apt install bison
```

```
# apt install flex
```

```
# make rpi_3_32b_defconfig // 오류없음
```

: U-Boot의 환경설정내용이 .config 히든파일로 저장

U-Boot 부트로더(계속)

- * U-Boot 컴파일

make -j4

: -j4 옵션은 코어의 개수 지정

ls u-boot*

u-boot u-boot.cfg u-boot.lds u-boot-nodtb.bin
u-boot.sym u-boot.bin u-boot.cfg.configs u-boot.map
u-boot.srec

=> u-boot.bin이 U-Boot의 이진파일임

이 파일을 타깃보드의 /boot 디렉터리로 복사

U-Boot 부트로더(계속)

- * 부트로더 포팅(방법1.. NFS 서비스를 통해)
: NFS 서버측

```
# cp u-boot.bin /nfs      // NFS 서비스 디렉터리로 복사  
# ls /nfs  
h_hello t_hello u-boot.bin
```

U-Boot 부트로더(계속)

- * 부트로더 포팅(방법1.. NFS 서비스를 통해)

: 라즈베리파이보드로 PuTTY 접속하여 로그인후,

: 타깃보드의 /boot 파티션으로 복사

```
$ sudo mount -t nfs 192.168.0.20:/nfs /share
```

```
$ ls /share
```

h_hello t_hello u-boot.bin

```
$ cd /boot
```

```
$ sudo cp /share/u-boot.bin ./ // 복사
```

```
$ ls *.bin
```

bootcode.bin u-boot.bin

U-Boot 부트로더(계속)

- * config.txt 파일 편집 (끝에 추가)

: 커널 이미지 파일 대신 U-Boot 이진파일을 실행하도록 설정

\$ sudo nano config.txt

.....

```
# U-boot loader  
kernel=u-boot.bin
```

- * 재부팅

\$ sudo reboot // 재부팅

U-Boot 부트로더(계속)

- * 부트로더 포팅(방법2...SD카드 boot 파티션에 기록)

: 가상머신에서 SD 카드를 USB 통해 연결후,

lsblk // SD 카드의 boot 파티션은 /dev/sdc1임을 유의!!

```
sda      8:0    0  100G  0 disk
└─sda1   8:1    0  100G  0 part /
sdc      8:32   1  14.5G  0 disk
└─sdc1   8:33   1  41.8M  0 part /media/ifc415/boot
└─sdc2   8:34   1  14.4G  0 part /media/ifc415/b4ea8e46-fe87-4ddd-9e94-506c37005
```

```
# mkdir ./tmp                      // 마운트포인트 생성
# mount /dev/sdc1 ./tmp             // boot 파티션을 마운트
# ls ./tmp                          // 마운트 확인
# cp u-boot.bin ./tmp              // U-Boot 부트로더 복사
# nano ./tmp/config.txt            // 환경파일 편집
```

kernel=u-boot.bin

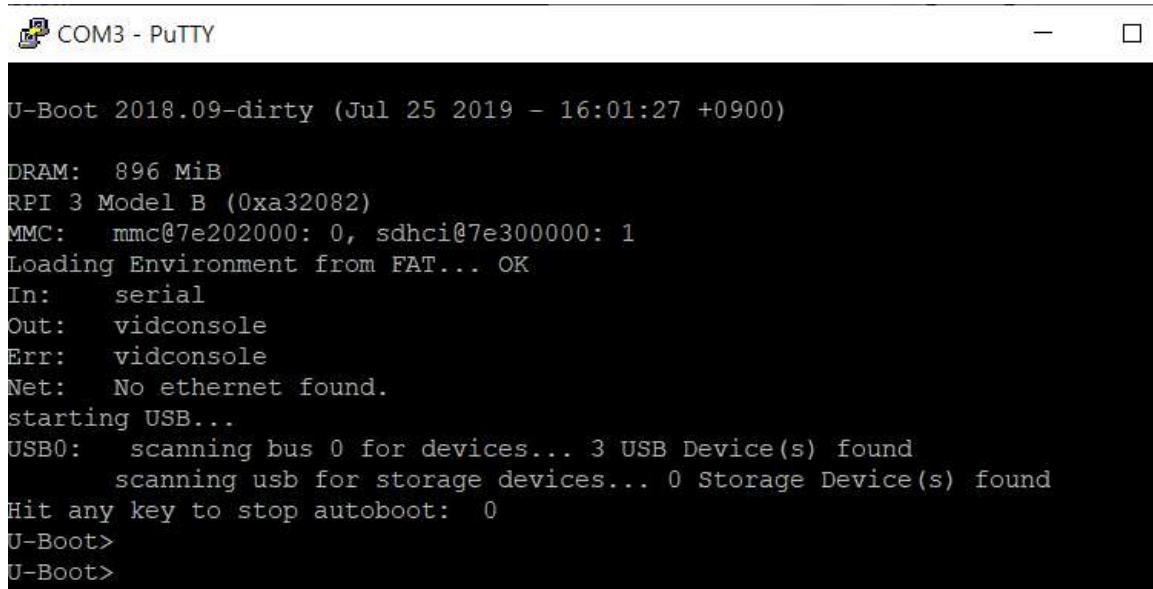
```
# sync
# umount ./tmp                    // 마운트 해제
```

: SD 카드를 라즈베리파이 보드에 장착후 부팅

U-Boot 부트로더(계속)

* U-Boot 부트로더 실행

- : 라즈베리파이보드에 전원 인가후,
Hit any key to stop.... 메시지 나타나면 즉시 키 누름
- : U-Boot 부트로더 프롬프트 **U-Boot>** 가 나타남



```
U-Boot 2018.09-dirty (Jul 25 2019 - 16:01:27 +0900)

DRAM: 896 MiB
RPI 3 Model B (0xa32082)
MMC: mmc@7e202000: 0, sdhci@7e300000: 1
Loading Environment from FAT... OK
In:   serial
Out:  vidconsole
Err:  vidconsole
Net:  No ethernet found.
starting USB...
USB0: scanning bus 0 for devices... 3 USB Device(s) found
      scanning usb for storage devices... 0 Storage Device(s) found
Hit any key to stop autoboot: 0
U-Boot>
U-Boot>
```

– U-boot 부트로더의 내부 명령 활용

U-Boot 부트로더(계속)

* U-Boot 내부명령

help .. 내부 명령에 대한 도움말

printenv ... 환경변수의 설정내역 출력

> **printenv ipaddr**

setenv ... 환경변수에 값 설정 (실습용 망관련 IP 적용)

> **setenv ipaddr 192.168.0.30** (ip address 설정)

> **setenv serverip 192.168.0.20** (server address 설정)

saveenv ... 환경변수 설정 내역 저장

> **saveenv** // /boot/uboot.env 파일로

환경변수 인용시 사용형식(예, kernel_addr_r, ...)

> **echo \${kernel_addr_r}** // 설정내용 출력

U-Boot 부트로더(계속)

* U-Boot 내부명령(계속)

gpio ... GPIO 제어 명령

> gpio set 18 // BCM_GPIO #18에 High신호 출력

ls ... 파일목록 보기

> ls mmc 0:1

nfs ... NFS 클라이언트 명령, /nfsroot/내 파일 전송

> nfs 0x0c100000 192.168.0.20:hello_world.bin

tftpboot ... TFTP 클라이언트 명령

> tftpboot 0x0c100000 hello_world.bin

fatload ... /boot 파티션에 있는 hello_world.bin 파일 적재

> fatload mmc 0:1 0x0c100000 hello_world.bin

go .. 메모리내 특정주소에 있는 응용프로그램 실행

> go 0x0c100000

U-Boot 부트로더(계속)

* TFTP 서버 구축 (가상머신에서)

```
# apt-get install tftpd-hpa tftp-hpa          // 패키지 설치
# gedit /etc/default/tftpd-hpa                // 환경설정 파일 생성
# /etc/default/tftpd-hpa
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/tftpboot"                  // 서비스 디렉터리 변경
TFTP_ADDRESS="0.0.0.0:69"
TFTP_OPTIONS="--secure"

# mkdir /tftpboot                             // 서비스 디렉터리 생성
# chown nobody /tftpboot                     // owner 변경
# chmod -R 777 /tftpboot                     // 접근권한 변경

# service tftpd-hpa restart                 // 서비스 재개
```

U-Boot 부트로더(계속)

* TFTP 서버 구동 확인

```
# service tftpd-hpa status // 서비스여부 확인위해
```

: 혹은, 파일전송 테스트

-서비스 디렉터리(/tftpboot)에 임의 파일(test.txt) 위치후

```
# tftp localhost
```

```
> get test.txt // 다운로드(파일전송)
```

```
> quit // tftp 클라이언트 종료
```

```
# ls
```

test.txt

U-Boot 부트로더(계속)

* U-Boot에서 TFTP 서비스 받기

- 1) LAN 케이블로 공유기와 라즈베리파이보드간 유선연결 후,
- 2) 양측의 IP 주소 설정 후, tftpboot 명령 사용

U-Boot> **setenv ipaddr 192.168.0.30**

// 타깃보드의 IP 주소 설정, 동일대역의 IP 주소 가능

U-Boot> **setenv serverip 192.168.0.20**

// 서버(가상머신)의 IP 주소 설정

U-Boot> **ping 192.168.0.20**

host 192.168.0.20 is alive

U-Boot> **tftpboot 0x0C100000 hello_world.bin**

// *hello_world.bin*을 0x0c10_0000주소에 기록, [실습2]

U-Boot 부트로더(계속)

[실습1] U-Boot에서 GPIO 제어

회로구성) LED 모듈을 BCM_GPIO #18에 연결

: Vcc, Gnd 연결, in을 BCM_GPIO #18과 연결

U-Boot> **help** // 내부 명령 목록

U-Boot> **gpio** // gpio 제어 명령

gpio <input|set|clear|toggle> <pin>

U-Boot> **gpio set 18** // LED ON

U-Boot> **gpio clear 18** // LED OFF

U-Boot> **gpio toggle 18** // LED toggle

U-Boot 부트로더(계속)

[실습2] hello_world.bin 실행

: U-Boot 소스에서 제공하는 테스트용 standalone app.

: U-Boot 소스컴파일과 동시에 이미 컴파일 됨

: 관련 주요 파일 참조

./arch/arm/armconfig.mk

// standalone 응용의 적재될 메모리 주소 확인

./examples/standalone/Makefile

./examples/standalone/hello_world.c

```
# cat ./arch/arm/config.mk
# SPDX-License-Identifier: GPL-2.0+
#
# (C) Copyright 2000–2002
# Wolfgang Denk, DENX Software Engineering, wd@denx.de.

ifndef CONFIG_STANDALONE_LOAD_ADDR
ifneq ($(CONFIG_ARCH_OMAP2PLUS),)
CONFIG_STANDALONE_LOAD_ADDR = 0x80300000
else
CONFIG_STANDALONE_LOAD_ADDR = 0x0c100000      #*)
endif
endif
.....
.....
.....
.....



# cat ./examples/standalone/Makefile
# SPDX-License-Identifier: GPL-2.0+
#
# (C) Copyright 2000–2006
# Wolfgang Denk, DENX Software Engineering, wd@denx.de.

extra-y      := hello_world          // 소스파일명
extra-$($(_CONFIG_SMC91111))      += smc91111_eeprom
.....
```

```
# cat ./examples/standalone/hello_world.c
// SPDX-License-Identifier: GPL-2.0+
/*
 * (C) Copyright 2000
 * Wolfgang Denk, DENX Software Engineering, wd@denx.de.
 */

#include <common.h>
#include <exports.h>

int hello_world (int argc, char * const argv[])          // main() 아님
{
    int i;

    /* Print the ABI version */
    app_startup(argv);
    printf ("Example expects ABI version %d\n", XF_VERSION);
    printf ("Actual U-Boot ABI version %d\n", (int)get_version());

    printf ("Hello World\n");
    .....
}
```

==> 함수명 hello_world를 main으로 해도 무방(2019.09.30)

* TFTP 서비스를 통한 적재 (방법1)

: hello_world.bin을 TFTP 서비스디렉터리로 복사후,

cp ./examples/standalone/hello_world.bin /tftpboot

U-Boot> **setenv ipaddr 192.168.0.30**

U-Boot> **setenv serverip 192.168.0.20**

U-Boot> **saveenv** // uboot.env 파일로 설정정보들 저장

U-Boot> **tftpboot 0x0C100000 hello_world.bin**

U-Boot> **md 0x0C100000** // memory dump

U-Boot> **go 0x0C100000** // 응용프로그램 실행

```
0c1000e0: e599c070 e59c1010 e599c070 e59c1014      p.....p.....
0c1000f0: e599c070 e59cf018 e599c070 e59cf01c      p.....p.....
U-Boot> go 0x0c100000
## Starting application at 0x0C100000 ...
Example expects ABI version 9
Actual U-Boot ABI version 9
Hello World
argc = 1
argv[0] = "0x0c100000"
argv[1] = "<NULL>"
Hit any key to exit ... █
```

* TFTP 서비스를 사용하지 않고 적재(방법2)

: hello_world.bin을 타깃보드의 /boot에 복사후,

U-Boot> **fatload mmc 0:1 0x0C100000 hello_world.bin**

// 타깃보드의 /boot에 있는 이진파일을 메모리에 적재

// mmc 0:1(maj:min)은 SD의 /boot 파티션

// 참고) \$ lsblk

// mmcblk0p1 ... SD의 /boot 파티션

// mmcblk0p2 ... SD의 / 파티션

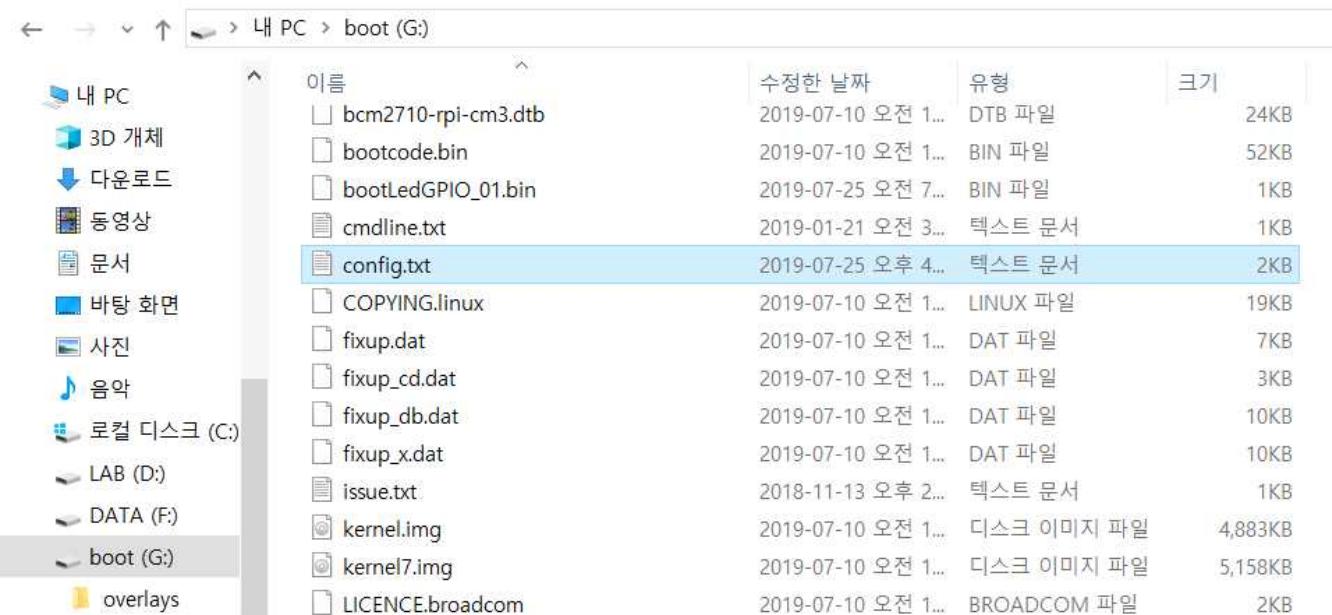
U-Boot> **md 0x0C100000** // memory dump

U-Boot> **go 0x0C100000** // 응용프로그램 실행

U-Boot 부트로더(계속)

* 원상 복구방법

- : Windows 환경에서
- : SD카드의 boot 드라이브에서의 config.txt 편집
- : kernel=u-boot.bin을 주석처리, 저장후 재부팅



이름	수정한 날짜	유형	크기
bcm2710-rpi-cm3.dtb	2019-07-10 오전 1...	DTB 파일	24KB
bootcode.bin	2019-07-10 오전 1...	BIN 파일	52KB
bootLedGPIO_01.bin	2019-07-25 오전 7...	BIN 파일	1KB
cmdline.txt	2019-01-21 오전 3...	텍스트 문서	1KB
config.txt	2019-07-25 오후 4...	텍스트 문서	2KB
COPYING.linux	2019-07-10 오전 1...	LINUX 파일	19KB
fixup.dat	2019-07-10 오전 1...	DAT 파일	7KB
fixup_cd.dat	2019-07-10 오전 1...	DAT 파일	3KB
fixup_db.dat	2019-07-10 오전 1...	DAT 파일	10KB
fixup_x.dat	2019-07-10 오전 1...	DAT 파일	10KB
issue.txt	2018-11-13 오후 2...	텍스트 문서	1KB
kernel.img	2019-07-10 오전 1...	디스크 이미지 파일	4,883KB
kernel7.img	2019-07-10 오전 1...	디스크 이미지 파일	5,158KB
LICENCE.broadcom	2019-07-10 오전 1...	BROADCOM 파일	2KB

GPIO 제어

* GPIO(General purpose input output)

- : 입력이나 출력을 포함한 동작이 사용자에 의해 제어되는 디지털 신호핀
- : 하나의 입출력 핀에 여러 기능 부여로 복잡한 구조
- : 기본적으로 입력과 출력이 가능하며, 하나의 핀은 GPIO로 혹은 다른 기능으로도 사용 가능

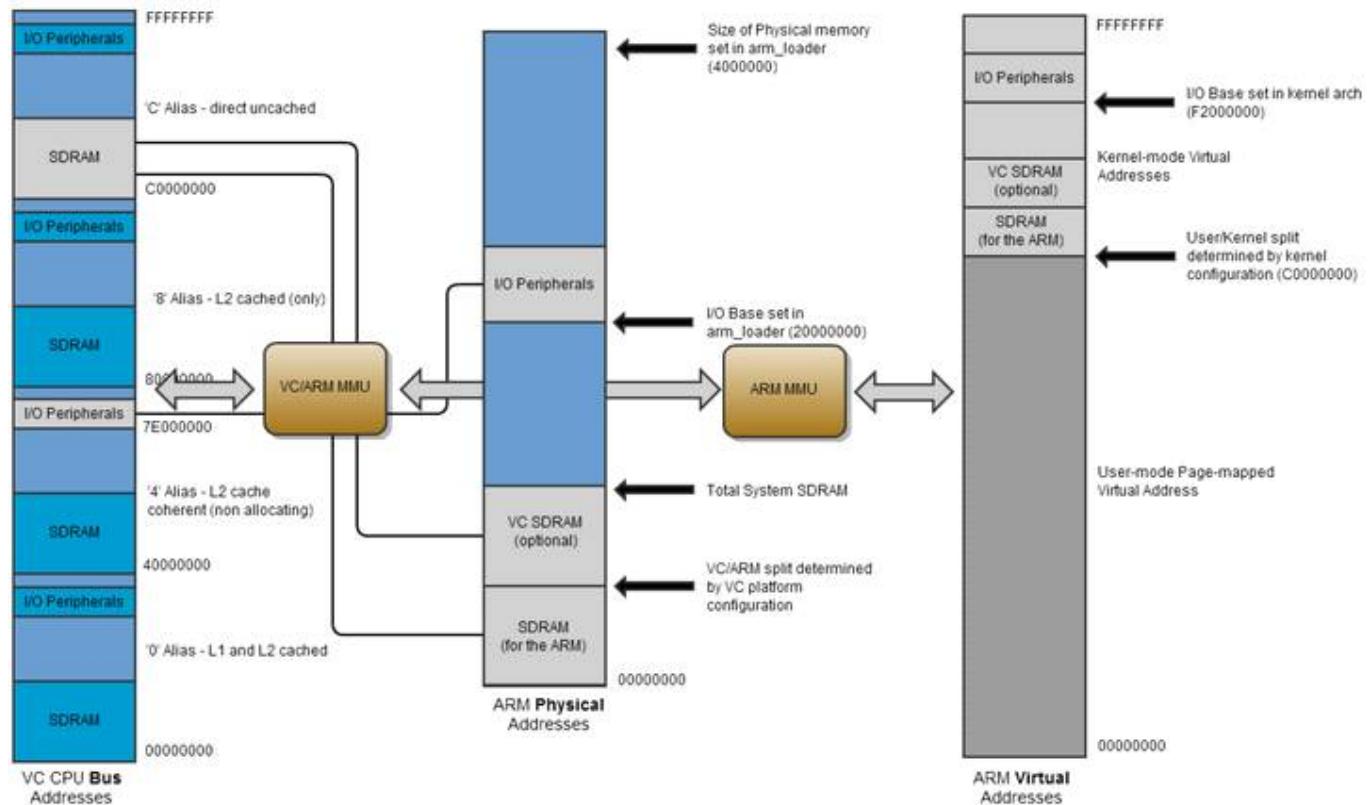
- : 타깃보드에서 총 54개의 GPIO 핀 제공

GPIO 제어(계속)

* BCM2835의 메모리 맵

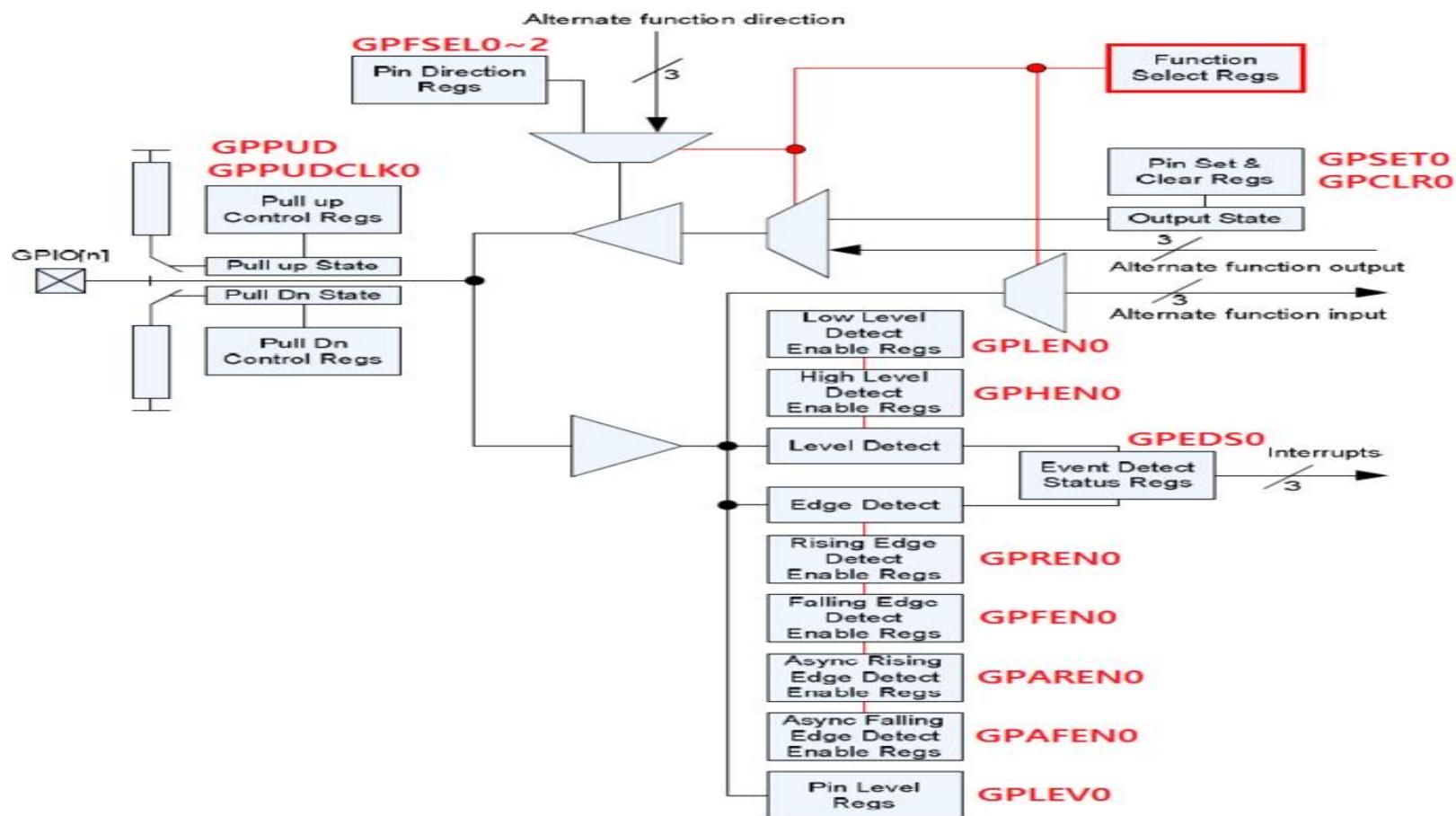
: BCM2837의 IO Peripherals Base 주소 ... **0x3F00_0000**

BCM2835 ARM Peripherals



GPIO 제어(계속)

* GPIO 핀 블록도 및 관련 레지스터



GPIO 제어(계속)

- * GPIO 관련 레지스터들의 베이스 주소
 - : IO Peripherals Base(**0x3F00_0000**) + **0x0020_0000**
 - : 즉, **0x3F20_0000**

GPIO 제어(계속)

- * GPIO 관련 레지스터(대략 13 종)
 - : 주요 레지스터의 물리주소 및 기능

PhysicalAddr	Field Name	Description	Size	Read/Write
0x3F20_0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x3F20_0004	GPFSEL1	GPIO Function Select 1	32	R/W
0x3F20_0008	GPFSEL2	GPIO Function Select 2	32	R/W
0x3F20_000C	GPFSEL3	GPIO Function Select 3	32	R/W
0x3F20_0010	GPFSEL4	GPIO Function Select 4	32	R/W
0x3F20_0014	GPFSEL5	GPIO Function Select 5	32	R/W
0x3F20_0018	- Reserved	-	-	-
0x3F20_001C	GPSET0	GPIO Pin Output Set 0	32	W
0x3F20_0020	GPSET1	GPIO Pin Output Set 1	32	W
0x3F20_0024	- Reserved	-	-	-
0x3F20_0028	GPCLR0	GPIO Pin Output Clear 0	32	W
0x3F20_002C	GPCLR1	GPIO Pin Output Clear 1	32	W
0x3F20_0030	- Reserved	-	-	-
0x3F20_0034	GPLEV0	GPIO Pin Level 0	32	R
0x3F20_0038	GPLEV1	GPIO Pin Level 1	32	R
0x3F20_003C	- Reserved	-	-	-
.....				

GPIO 제어(계속)

- * **GPFSELn**(GPIO Function Select Registers)
 - : GPIO 핀의 입력용/출력용, 다른 기능 설정
 입력용 000, 출력용 001, 다른기능(데이터시트 참조)
 - : 각 핀당 3비트씩 소요, 각 레지스터의 상위2비트 비사용
 - : 1개 레지스터에 LSB부터 할당하여 10개 GPIO핀 설정
 - : 예) **BCM_GPIO #18 출력용, BCM_GPIO #1 입력용**

GPIO 제어(계속)

* GPIO 핀관련 물리주소

: 출력용 디바이스(예, LED .. BCM_GPIO#18)

: 입력용 디바이스(예, BTN .. BCM_GPIO#17)

```
#define GPIO_BASE      0x3F200000 // for BCM2836, BCM2837

// address offset of registers for GPIO
#define GPFSEL1      0x04
#define GPSET0      0x1C
#define GPCLR0      0x28
#define GPLEV0      0x34

char * gpio_mmap;                                // 메모리맵으로 설명!!
volatile unsigned int * gpio;

gpio_mmap = (char *)GPIO_BASE;                    // 물리 주소
gpio = (volatile unsigned int *)gpio_mmap;
```

GPIO 제어(계속)

* 출력용 디바이스(예, LED .. BCM_GPIO#18)

: 출력용(001) ...GPFSEL1 레지스터

```
//          19 18          11 10
// GPFSEL1 = 0bxx xxx 001 xxx xxx xxx xxx xxx xxx xxx xxx;
gpio[GPFSEL1 / 4] |= (7 << 24); // set 111(7) corresponding bits
gpio[GPFSEL1 / 4] &= ~(6 << 24); // #18, output, ~(001)=110(6)
```

: High 출력할 때 ... GPSET0 레지스터

```
//          29          20 19          10 9          0
// GPSET0 = 0b00 00000000000 01000000000 00000000000;
gpio[GPSET0 / 4] |= (1 << 18); // LED ON
```

: Low 출력할 때 ... GPCLR0 레지스터

```
//          29          20 19          10 9          0
// GPCLR0 = 0b00 00000000000 01000000000 00000000000;
gpio[GPCLR0 / 4] |= (1 << 18); // LED OFF
```

GPIO 제어(계속)

* 입력용 디바이스(예, BTN .. BCM_GPIO#17)

: 입력용(000) ...GPFSEL1 레지스터

```
//          19 18 17           11 10
// GPFSEL1 = 0bxx xxx xxx 000 xxx xxx xxx xxx xxx xxx xxx xxx;
// gpio[GPFSEL1 / 4] |= (7 << 21); // set 111(7) corresponding bits
// gpio[GPFSEL1 / 4] &= ~(7 << 21); // #17, input, ~(000)=111(7)
```

: 신호읽기 ... GPLEVO 레지스터

```
//          29      20 19      10 9      0
// GPLEVO = 0bXX XXXXXXXXXXXX XXXXXXXXXXXX XXXXXXXXXXXX;
// AND    0b00 0000000000 0010000000 00000000000;
ret = (gpio[GPLEVO / 4] & (1 << 17)) >> 17; // press?
// ret는 0, 혹은 1
```

: 기타 레지스터가 사용될 수 있음

GPIO 제어(계속)

[실습3] LED 제어 (부트로더 실행방식)

: LED ON/OFF 하기를 반복하는 standalone app.

: GPIO 핀의 물리주소로 접근하여 제어

```
# cd ./examples/standalone
# nano Makefile

.....
extra-y := bootGpioLed // 작성할 파일명으로
.....



# nano bootGpioLed.c ( 학습자료 파일 복사하여 활용 가 )
//=====
// bootGpioLed.c
//    BCM_GPIO #18, LED
//    to execute like U-boot bootloader
//=====
#include <common.h>           // U-boot
```

```
#include <exports.h>           // U-boot

#define GPIO_BASE      0x3F200000 // for BCM2836, BCM2837

// address offset of registers for BCM_GPIO #18
#define GPFSEL1        0x04
#define GPSET0         0x1C
#define GPCLR0         0x28

int main(void) {
    char * gpio_mmap;
    volatile unsigned int * gpio;

    printf("[rawGPIO testing.....LED like U-boot]\n");

    gpio_mmap = (char *)GPIO_BASE; // 물리 주소

    gpio = (volatile unsigned int *)gpio_mmap;

    //          19  18          11  10
    // GPFSEL1 = 0bxx xxx 001 xxx xxx xxx xxx xxx xxx xxx xxx;
    gpio[GPFSEL1/4] |= (7 << 24); // set 111(7) corres. bits
    gpio[GPFSEL1/4] &= ~(6 << 24); // #18,output,~(001)=110(6)

    while(1) {
        //          29      20 19      10 9      0
        // GPSET0 = 0b00 0000000000 0100000000 0000000000;
        gpio[GPSET0 / 4] |= (1 << 18); // LED ON
```

```
    printf("LED ON.....\n");
    mdelay(1000);

    //          29      20 19      10 9      0
    // GPCLR0 = 0b00 0000000000 0100000000 0000000000;
    gpio[GPCLR0 / 4] |= (1 << 18);      // LED OFF
    printf("LED OFF.....\n");
    mdelay(1000);
}

return 0;
}
```

* **컴파일**

```
# cd ../../
# make clean           // 앞서 컴파일로 생성된 목적코드 등을 제거
# make -j4
# ls ./examples/standalone/
bootGpioLed.bin       .....
```

- * TFTP 서비스 디렉터리로 bootGpioLed.bin 파일 복사후,

```
# cp ./examples/standalone/bootGpioLed.bin /tftpboot
```

- * U-Boot 환경에서

```
U-Boot> gpio toggle 18 // gpio 테스트
```

```
U-Boot> gpio toggle 18
```

```
U-Boot> tftpboot 0x0C100000 bootGpioLed.bin
```

```
U-Boot> go 0x0C100000
```

**==> 운영체제없이 부트로더가 실행되는 방식으로
GPIO 핀에 연결된 디바이스를 제어가능!!!!
(블리주소를 이용하여)**

응용과제

[응용1] U-Boot 부트로더(타깃보드에서 컴파일)

: 앞의 U-Boot 부트로더 소스 및 응용프로그램의 컴파일은
타깃보드에서도 가능하므로 이를 실습

[응용2] GPIO LED 제어(핀 변경)

: U-Boot 부트로더에서 standalone app. 실행방식으로
: 각자 임의 GPIO 핀을 선택하여 LED 회로를 연결
(#20보다 큰 것 선택)

응용과제(계속)

[응용3] GPIO LED 및 BTN 제어

- : U-Boot 부트로더에서 standalone app. 실행방식으로
(bootGpioLedBtn.c 참조)
- : LED와 BTN 회로 활용
- : 수행동작은 각자 정의할 것

참고) BTN 모듈의 회로연결은 교재4장 후반 참조

- Pull-up 저항 모드 (모듈기판 명시대로 연결)
(press : Low signal, release : High signal)
- Pull-down 저항 모드 (Vcc와 Gnd를 교차 연결)
(press : High signal, release : Low signal)

응용과제(계속)

[응용4] Bare-Metal GPIO LED 제어

- : Bare-metal(bare machine) ... 운영 체제를 포함하여 어떤 소프트웨어도 설치되어 있지 않은 하드웨어를 의미
- : **./bareMetal** 소스 참조
 - (GPIO LED 제어, BCM_GPIO #21)

* 절차

- 1) 새 SD 메모리카드를 fat32 형식으로 포맷후,
- 2) /boot 디렉터리의 다음 3개 파일 복사
 - : **bootcode.bin, fixup.dat, start.elf**
- 3) **./bareMetal** 소스 컴파일후, **kernel7.img** 복사(LED 제어)
 - # **make clean**
 - # **make**
- 4) 타깃보드에 SD 메모리카드 삽입후 파워 인가

응용과제(계속)

[응용5] Bare-Metal GPIO LED 및 BTN 제어

- : bare-metal 환경에서 GPIO LED 및 BTN 제어
- : ./bareMetal 소스 참조
- : 회로연결은 각자 정의

GPIO 제어(계속)

[응용 6] U-Boot 부트로더 (... 추후 시도)

- : U-Boot 부트로더에서 커널 이미지를 부팅
- : TFTP 서비스 디렉터리에 관련 파일(아래 밑줄) 위치후,

```
U-Boot> setenv ipaddr 192.168.0.40  
U-Boot> setenv serverip 192.168.0.20  
U-Boot> tftpboot ${kernel_addr_r} kernel7.img  
U-Boot> tftpboot ${fdt_addr_r} bcm2710-rpi-3-b-plus.dtb  
U-Boot> bootz ${kernel_addr_r} - ${fdt_addr_r}
```

// - : initrd 없이

(참고)

```
U-Boot> echo ${kernel_addr_r}  
0x00080000  
U-Boot> echo ${fdt_addr_r}  
0x02600000
```

: 아래 과정은 SD의 /boot 내 파일을 적재하는 과정임

U-Boot> **fatload mmc 0:1 \${kernel_addr_r} kernel7.img**

5281136 bytes read in 248 ms (20.3 MiB/s)

U-Boot> **fatload mmc 0:1 \${fdt_addr_r} bcm2710-rpi-3-b-plus.dtb**

25489 bytes read in 3 ms (8.1 MiB/s)

U-Boot> **bootz \${kernel_addr_r} - \${fdt_addr_r}**

Kernel image @ 0x080000 [0x0000000 - 0x509570]

Flattened Device Tree blob at 02600000

Booting using the fdt blob at 0x2600000

reserving fdt memory region: addr=0 size=1000

Using Device Tree in place at 02600000, end 02609390

Starting kernel ...