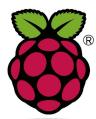


Ver_2019_09_03



Raspberry Pi 기반

임베디드시스템응용



머릿말 1

머릿말

Ver_2019_09_03

이 책은 라즈베리파이보드 3B+ 기반 임베디드 시스템 응용에 관한 것이다. 소스는 주로 C 언어로 구현하였으며, 어떤 부분은 지나치게 세세하게, 어떤 부분은 거칠게, 어떤 부분은 장황하게 서술되었을 수 있다. 미진한 부분은 차차 보완토록 하겠다.

앞서 고심하여 익히고 이를 웹상에 게시한 많은 이들의 자료를 참고하였으며, 각 장의 끝에 그 내역을 덧 붙였다. 그들에게 진심 감사의 말을 지면으로나마 전한다. 아울러 이 자료는 누구든 맘껏 참조하여 조금이나마 도움이 될 수 있으면 좋겠다.

이와 관련된 소스 등의 자료는 다음의 사이트 [IFC415]임베디드응용및실습 꼭지를 참고바란다.

<https://cms3.koreatech.ac.kr/sites/joo/home.html>

2 라즈베리파이기반 임베디드시스템 응용

목차

제1장 임베디드 시스템 소개	1
1.1 IoT HW 플랫폼	1
1.2 라즈베리파이 모델	5
제2장 개발 환경 구축 I	9
2.1 싱글보드 컴퓨터	9
2.2 Wi-Fi 망 구축	15
2.3 개발 환경 툴	32
2.3.1 PuTTY 원격 접속	35
2.3.2 NFS 서비스	37
2.3.3 SFTP 서비스	40
2.3.4 Samba 서비스	44
2.3.5 mstsc 원격 접속	47
2.3.6 DD for Windows	50
2.4 리눅스 명령	52
제3장 개발 환경 구축 II	63
3.1 VMware Player	63
3.2 VM(Ubuntu) 설치	66
3.3 VMware Tools	84
3.4 툴 체인	90
3.5 한글 표시 및 입력기 설치	95



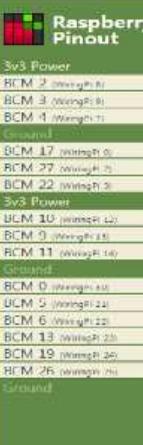
2 라즈베리파이기반 임베디드시스템응용

제4장 입출력 디바이스 제어 I	103
4.1 wiringPi 라이브러리	103
4.2 LED 제어	110
4.3 시그널 처리	114
4.4 LED PWM 제어	117
4.5 BTN 제어	122
4.6 인터럽트 처리	127
 제5장 입출력 디바이스 제어 II	133
5.1 터치스위치 제어	133
5.2 자석스위치 제어	135
5.3 움직임감지센서 제어	137
5.4 피에조 부저 모듈	139
5.5 LED 어레이 제어	144
5.6 3색 LED 제어	146
5.7 릴레이 모듈 제어	149
5.8 초음파센서 제어	153
 제6장 모터 제어	161
6.1 DC 모터 제어	161
6.2 스텝핑 모터 제어	170
6.3 서보 모터 제어	175
 제7장 도트매트릭스 및 키패드	183
7.1 8x8 도트매트릭스	183
7.2 4x4 키패드	195
7.3 CLCD	202

제8장 시리얼 통신	213
8.1 시리얼 통신	213
8.2 루프백 시리얼통신	222
8.3 Windows PC와의 시리얼통신	224
8.4 가상머신과의 시리얼통신	230
8.5 Arduino 보드와의 시리얼통신	233
제9장 1-wire 인터페이스	241
9.1 1WI(1-wire interface)	241
9.2 온습도 센서 모듈	246
제10장 I2C 인터페이스	253
10.1 I2C 인터페이스	253
10.2 ADC/DAC 모듈	257
10.3 OLED 제어	269
제11장 SPI 인터페이스	275
11.1 SPI 인터페이스	275
11.2 MCP3208 ADC	280
11.3 아날로그 입력 센서 제어	284
11.3.1 가변저항	285
11.3.2 화염 센서	288
11.3.3 조도센서	289
11.3.4 수분센서	291
11.3.5 온도센서 및 습도센서 제어	292
11.3.6 적외선거리 센서	294
제12장 U-Boot 부트로더 및 GPIO	307
12.1 U-Boot 부트로더	307
12.2 GPIO	322

4 라즈베리파이기반 임베디드시스템응용

제13장 커널 컴파일 및 커널 모듈	333
13.1 커널 컴파일	333
13.2 가상주소에 의한 디바이스 제어	342
13.3 커널 모듈	348
13.4 커널 모듈의 동적 링크	361
13.5 커널 모듈을 커널에 포함하기	380
제14장 소켓프로그램에 의한 원격 제어	391
14.1 TCP/IP 프로토콜	391
14.2 TCP 소켓프로그래밍	398
14.3 UDP 소켓프로그래밍	416
제15장 웹서비스에 의한 원격 제어	429
15.1 Apache 웹서버	429
15.2 CGI 프로그래밍	441
15.3 CGI에 의한 LED 제어	448
제16장 FPGA 디바이스 제어	459
16.1 카메라 모듈	459
16.2 FPGA 디바이스 제어	464
16.3 GUI 통한 디바이스 제어	484
부록	497
부록1. 라즈베리파이3 핀 레이아웃	497
부록2. 네트워크 레이아웃	498
부록3. wiringPi 라이브러리	499



제1장 임베디드 시스템 소개 1

제1장 임베디드 시스템 소개

아두이노 보드, 라즈베리파이 보드 등의 임베디드 시스템에 대해 간략히 살펴본다.

1.1 IoT HW 플랫폼

IoT 환경에서 사용되는 H/W 플랫폼의 대표적인 것들이 아두이노와 라즈베리파이다. 아두이노와 라즈베리파이는 약간의 프로그래밍을 통해 각종 센서와 액추에이터를 제어할 수 있는 소형 컴퓨터이다

주요 보드인 아두이노와 라즈베리파이 보드의 간략한 사용은 다음과 같다.

구분	아두이노	라즈베리파이
모양		
Spec	CPU : ATMega328 CPU Speed : 16Mhz RAM : 2K	CPU : ARM Cortex-A53 CPU Speed : 1.2Ghz RAM : 1G
OS	없음	라즈베리안(리눅스 계열), 리눅스, 윈도우
개발 언어	c언어	파이썬, 웹프로그래밍, C언어 등등
개발 방법	IDE(통합개발환경)에서 코딩 후 아두이노 보드로 업로드	os가 설치되어 있으므로 개발툴을 이용하여 개발 후 컴파일된 실행파일 수행

1.1.1 Arduino

2005년 이탈리아 Massimo Banzi 교수에 의해 개발된 보드이다. 아두이노는 자체

2 라즈베리파이기반 임베디드시스템응용

운영체제나 펌웨어가 없으며, sketch(Arduino IDE)를 통해 C언어 형태로 프로그래밍하고 그 머신코드를 보드에 업로드하여 센서들을 제어한다. 센서 제어에 특화된 마이크로컨트롤러이다.



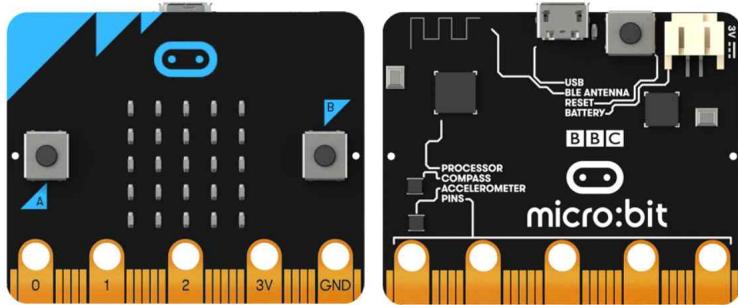
1.1.2 Raspberry Pi

자체적인 운영체제를 설치가능하고, 내부에서 프로그래밍가능하다. 아두이노와 달리 모니터, 키보드, 마우스 등을 연결하여 개인용 컴퓨터로 활용이 가능하며, 이더넷, HDMI, USB를 지원한다.



1.1.3 micro:bit

BBC micro:bit는 4*5cm 크기의 코딩 교육용 미니 보드다. 영국 국영방송국인 BBC 주도하에 삼성전자, 마이크로소프트, ARM, 랭커스터 대학교 등의 산학공동으로 개발하였다. 5x5(25개)의 LED를 통해서 정보를 표시가능하고, 온도 센서, 방위 센서, 가속도(모션) 센서 등을 기본으로 탑재하고 있다. 여러 단자를 통해 다양한 센서를 추가적으로 연결해서 사용할 수도 있다. 또한, 블루투스를 이용해서 스마트 폰이나 BBC micro:bit 간에 통신을 지원한다.



マイクロビット는 파이썬(Python)과 자바스크립트 블록 에디터(JavaScript Block Editor)라는 개발환경을 지원하고 있는데, 자바스크립트 블록 에디터는 MIT의 스크래치와 같은 블록과 자바스크립트를 함께 이용해 작업할 수 있는 환경을 제공한다.

라즈베리파이처럼 프로세서 속도가 빠른 미니컴퓨터이기도 하며, 아두이노처럼 외부센서도 제어가능하다. 국내에서 보드를 구하기 쉽지 않고 개발 예제들이 많지 않다. 국내에서 최근 EBS 코딩교육에 활용하고 있다.

4 라즈베리파이기반 임베디드시스템응용

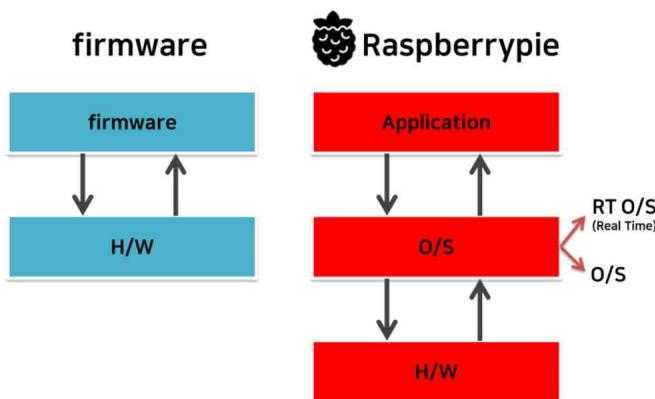
구분	micro:bit	ARDUINO	RASPBERRY PI
CPU	ARM Cortex-M0	Atmega 328	ARM Cortex-A5
clock	16Mhz	16Mhz	1.2GHz
처리	32bit	8bit	32bit
RAM	16KB	2KB	1GB
저장공간	256KB	32KB	micro SD
내장센서	5x5 ledmatrix, 푸쉬버튼,온도,방위,가속도센서	-	-
통신	블루투스	블루투스	블루투스, WiFi
인터페이스	3V 건전지홀더용	5V microUSB	5V microUSB HDMI, Ethenet, USBx4, Audio, SD card
운영체제	-	-	Linux 등(micro SD)
프로그래밍 언어	블록기반언어	텍스트기반언어 (C)	텍스트기반언어 (C, Phyton 등)
주 개발환경	자바스크립트블록에디터	Sketch	고급언어개발환경
용도	마이크로컨트롤러	마이크로컨트롤러	싱글보드컴퓨터 (SBC)
가격	20\$ 대	\$30 대	\$35 대
특징	입출력센서 제어에 특화	입출력센서 제어에 특화	입출력센서 제어 및 데이터 처리에 특화
주용도	코딩 입문자 교육	전공자의 교육 및 개발	전공자의 교육 및 개발

1.1.4 IoT 디바이스 제어 방식

아두이노 보드와 라즈베리파이 보드간의 가장 큰 차이점은 운영체제의 포함 여부이다.

아두이노 보드나 마이크로 비트 보드의 경우 일반 펌웨어가 직접 하드웨어를 제어하는 반면, 라즈베리파이 보드에서는 응용 프로그램이 운영체제에 요청을 하면, 운영체제가 하드웨어 디바이스를 제어하는 방식이다. 라즈베리파이보드의 경우 운영체제가 하드웨어 제어를 관여하면서 시간 정확성은 조금 떨어질 수 있으며, RT 운

영체제가 아닌 경우, 센서의 제어에 있어 완벽한 정확도를 요구하기는 어려울 수 있다.



1.2 라즈베리파이 모델

라즈베리파이는 2012년도 Model 1 B를 시작하여 2018년도 3월 모델 3 B+까지 출시되었다. 라즈베리파이 3 모델은 Wi-Fi가 내장되어 있으므로 별도의 무선랜카드나 동글을 사용할 필요 없이 인터넷 연결이 가능하다.

다음 표는 라즈베리파이 모델들에 대한 사양을 간략히 정리한 것이다.

6 라즈베리파이기반 임베디드시스템응용

Name	Arduino	Raspberry Pi Model A+	Raspberry Pi Model B +	Raspberry Pi 2 Model B	Beaglebone Black
Price	\$29.95	\$20	\$35	\$35	\$45
Processor	ATMega 328	ARM 11	ARM Cortex A7	ARM Cortex A8	
Clock Speed	16Mhz	700 Mhz	900 Mhz	1 Ghz	
RAM	2KB	256MB	512MB	1GB	512MB
Multi Core	Single	Single	Quad	Single	
Flash	32KB	Micro SD Card		SD Card	
GPIO	26	40		92	
Ethernet	N/A	10/100			
USB	N/A	USB 2.0 x 1	USB 2.0 x 4	USB 2.0 x 1	
Video Out	N/A	HDMI, Composite		N/A	
Audio Out	N/A	HDMI, Analog		Analog	

3B 모델과 3B+ 모델간 차이점

원주율 3.14를 기념하는 Pi Day인 2018년 3월 14일에 라즈베리파이 3B의 업그레이드 버전인 라즈베리파이 3B+가 공개되었다.

다음 내용은 라즈베리 파이 공식 웹사이트의 내용을 간략히 정리한 것이다. 라즈베리파이 3B+와 라즈베리파이 3B는 동일한 가격대이며, 주된 차이점은 다음과 같다

- 1.4Ghz의 Quad Core A53
- Wi-Fi는 802.11AC를 지원 (2.4Ghz, 5Ghz)
- Bluetooth 4.2 지원
- 더 빠른 속도의 Ethernet 포트를 장착 (USB 2.0을 이용한 기가비트 이더넷)
- PoE 지원 (별도의 HAT이 필요)
- 향상된 PXE 네트워크 / USB 스토리지를 통한 부팅
- 향상된 열 관리

참고자료

[1] IoT HW 플랫폼 비교

<http://blog.naver.com/PostView.nhn?blogId=kt5709&logNo=220956687930>

[2] IoT HW 플랫폼 비교

<http://blog.naver.com/PostView.nhn?blogId=simjk98&logNo=221065531547>

[3] IoT HW 플랫폼 비교

<http://blog.naver.com/PostView.nhn?blogId=playcodingacademy&logNo=221084610825>

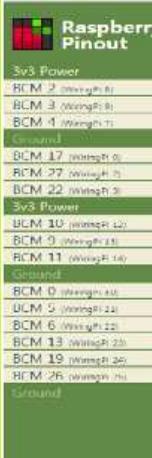
[4] micro:bit <http://leocom.tistory.com/12618>

[5] “사물인터넷을 위한 BBC micro:bit 프로그래밍”, 서영진, 제이펍, 2017

[6] “신나는 라랄라 라즈베리파이 배우기”, 정종화, 메카솔루션, 2018

[7] 라즈베리파이 3B+ 개선점 정리, <https://ung-d.tistory.com/47>

8 라즈베리파이기반 임베디드시스템 응용



제2장 개발 환경 구축 I 9

제2장 개발 환경 구축 I

라즈베리파이 보드를 이용하여 하나의 독자적인 컴퓨터 시스템을 구축하는 싱글보드 컴퓨터(single board computer) 구성, 무선공유기를 이용하여 Wi-Fi 망을 구축하는 과정을 살펴본다. 또한 응용프로그램 개발과정에서 Windows 시스템과 데이터를 공유하기 위한 유용한 툴들의 설치 및 사용에 대해 살펴본다.

2.1 싱글보드 컴퓨터

라즈비안 이미지가 기록된 micro SD 카드를 라즈베리파이 보드의 뒷면에 삽입하고, 모니터케이블, 마우스, 키보드를 연결함으로써 라즈베리파이 보드로 하나의 온전한 컴퓨터 시스템을 구축할 수 있다. 이러한 이유로 라즈베리파이 보드를 SBC(single board computer)라 부르기도 한다.

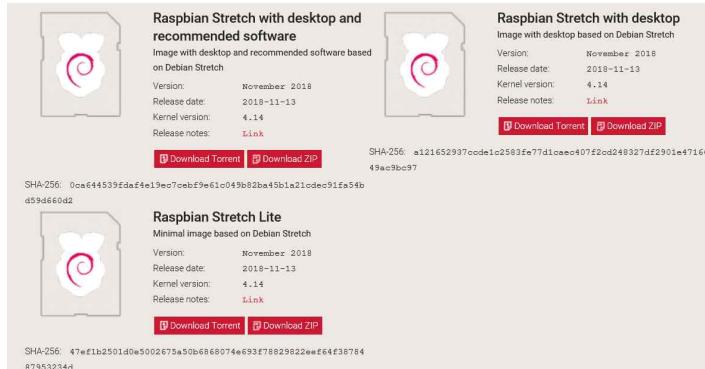
2.1.1 라즈비안 이미지 기록

라즈비안(Raspbian, Linux debian) 이미지를 micro SD 카드에 기록하여야 한다. 우선 라즈비안 이미지파일을 얻기 위해 다음의 사이트를 방문하여 이미지 파일을 다운로드한다.

<https://www.raspberrypi.org/downloads/raspbian/>

라즈베리파이 사이트에 접속하면 다음과 같은 화면이 나오고, RASPBIAN STRETCH LITE에서 이미지 파일을 다운로드한다.

10 라즈베리파이기반 임베디드시스템용



다운로드 한 2018-11-18-raspbian-stretch-lite.zip 파일의 압축을 풀고, 다음의 이미지 파일을 micro SD 카드에 이미지 기록 툴을 이용하여 기록한다.

2018-11-18-raspbian-stretch-lite.img

micro SD에 이미지를 기록할 기록 툴은 여러 가지가 있으며, 이 중 Etcher 이미지 기록 툴을 사용하는 것을 살펴보도록 하겠다.

Etcher 기록 툴은 다음 사이트에서 다운로드 받을 수 있다.

<https://etcher.io/>

Etcher는 Mac OS, Linux 그리고 Windows상에서 실행할 수 있는 그래픽형태의 SD 카드 기록 툴로 손쉽게 사용할 수 있다. 또한 zip 파일을 풀지 않고도 이미지를 직접 기록하는 것이 가능하다.

Etcher를 사용하여 이미지를 기록하려면 우선 툴을 다운로드하여 설치한다. SD 카드를 삽입한 SD 카드 리더기를 컴퓨터와 연결한다. Etcher를 실행하고 기록할 라즈비안 이미지 파일이나 zip 파일을 선택한다. 그런 후 이미지를 기록할 SD 카드를 선택한다. 선택된 사항들이 맞는지 확인하고, 이미지 기록을 위해 Flash! 버튼을 클릭한다. 기록하고 검증하는데 20여분 정도 소요된다.

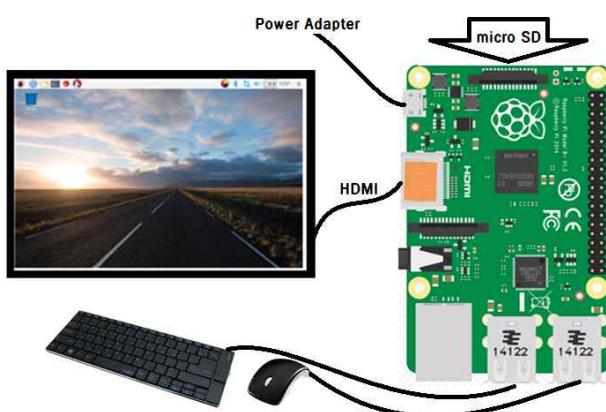


추가로 micro SD 기록 툴로 흔히 win32diskimager 도 사용되며, 다음 사이트에서 다운로드 할 수 있으며, 사용법은 웹 서핑을 통해 확보할 수 있다.

<https://sourceforge.net/projects/win32diskimager/>

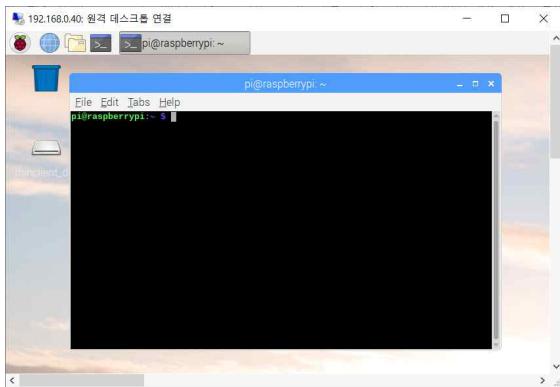
2.1.2 하드웨어 연결

다음 그림과 같이 라즈비안 이미지가 기록된 micro SD 카드를 라즈베리파이 보드의 뒷면 삽입구에 삽입한다. HDMI 케이블로 모니터와 연결하고, USB 키보드와 마우스를 USB 포트에 연결한다. 그리고 라즈베리파이 보드에 5V, 2.5mA 전원을 인가한다.



12 라즈베리파이기반 임베디드시스템용

최초의 부팅에는 시스템 환경구축과 관련한 얼마간의 시간이 소요되며, 기본 계정인 pi 계정으로 자동 로그인된다. 다음과 같은 화면이 나타나며, 터미널 창을 열어 리눅스 명령들을 사용할 수 있다.



로그아웃할 때는 산딸기(시작) 아이콘 - shutdown을 클릭하여 종료하거나, 터미널 창에서 다음과 같이 logout 명령을 사용한다.

```
$ logout
```

기본 로그인 계정명은 pi이고 암호는 raspberry이다. 원격에서 라즈베리파이에 접속하는 경우 계정명과 암호를 요구하므로 내정된 암호를 잘 기억해 둘 필요가 있다.

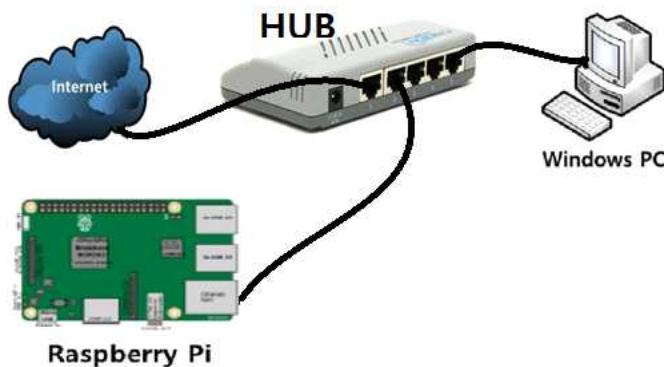
2.1.3 유선망 연결

Windows 시스템과 동일한 네트워크 주소를 갖는 라즈베리파이 보드에 설정할 추가의 고정 IP 주소를 준비한다. 라즈베리파이에 설정할 고정 IP 주소는 172.18.14.xx라 가정한다.

망구성

Widows 시스템은 HUB를 통하여 인터넷 상에 이미 맞물려 있다고 가정하고, 아래

그림과 같이 라즈베리파이 보드의 이더넷 포트에 LAN 케이블을 연결하고 HUB에 연결한다. 실습실 환경에서는 기존 PC에 연결된 LAN 케이블을 빼서 라즈베리파이 보드에 연결하면 동일 효과를 볼 수 있다.



고정 IP 주소 설정

IP 주소를 설정할 경우 라즈베리파이 모델 3부터는 /etc/network/interfaces 파일이 아닌 /etc/dhcpcd.conf 파일을 수정하여 설정해야 한다.

\$ sudo nano /etc/dhcpcd.conf

```

File Edit Tabs Help
GNU nano 2.7.4          File: /etc/dhcpcd.conf          Modified
#static routers=192.168.1.1
#static domain_name_servers=192.168.1.1

# fallback to static profile on eth0
#interface eth0
#fallback static_eth0

interface eth0
    static ip_address=172.18.14.189
    static netmask=255.255.255.0
    static routers=172.18.14.254
    static domain_name_servers=168.126.63.1

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^L Go To Line
  
```

14 라즈베리파이기반 임베디드시스템용

설정할 고정 IP 주소가 172.18.14.xx라고 가정할 경우, 파일 끝부분에 다음의 내용을 추가하고 저장한다.

```
interface eth0
    static ip_address=172.18.14.xx
    static netmask=255.255.255.0
    static routers=172.18.14.254
    static domain_name_servers=168.126.63.1
```

다음의 명령으로 재부팅하여 변경된 설정 정보를 반영할 수 있게 한다.

```
$ sudo reboot
```

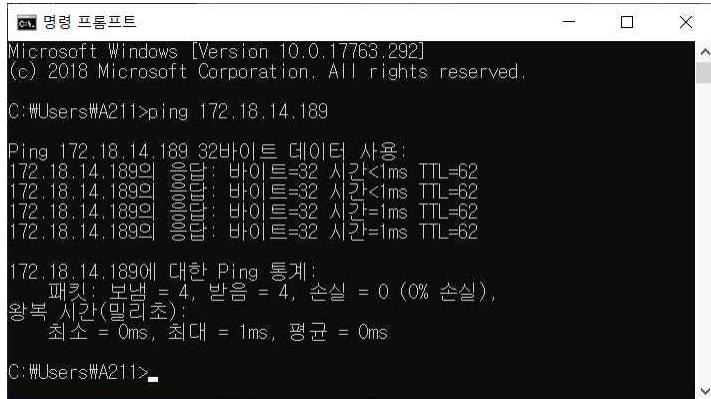
이제 다음의 ifconfig 명령을 사용하여, 고정 IP 주소의 설정 여부를 확인할 수 있다.

```
$ ifconfig eth0
```

```
pi@raspberrypi:~ $ ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
      inet 172.18.14.189  netmask 255.255.0.0  broadcast 172.18.255.255
          inet6 fe80::92bc:56cd:aa19:eefb  prefixlen 64  scopeid 0x20<link>
            ether b8:27:eb:06:fb:58  txqueuelen 1000  (Ethernet)
              RX packets 18990  bytes 1936942 (1.8 MiB)
              RX errors 0  dropped 0  overruns 0  frame 0
              TX packets 13242  bytes 17092085 (16.3 MiB)
              TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
pi@raspberrypi:~ $
```

다음과 같이 PC에서 명령프롬프트 창을 열어 라즈베리파이와의 통신이 가능한지 ping하여 통신가부를 확인한다.

```
C:\> ping 172.18.14.xx
```



```

C:\Users\A211>ping 172.18.14.189

Ping 172.18.14.189 32바이트 데이터 사용:
172.18.14.189의 응답: 바이트=32 시간<1ms TTL=62
172.18.14.189의 응답: 바이트=32 시간<1ms TTL=62
172.18.14.189의 응답: 바이트=32 시간=1ms TTL=62
172.18.14.189의 응답: 바이트=32 시간=1ms TTL=62

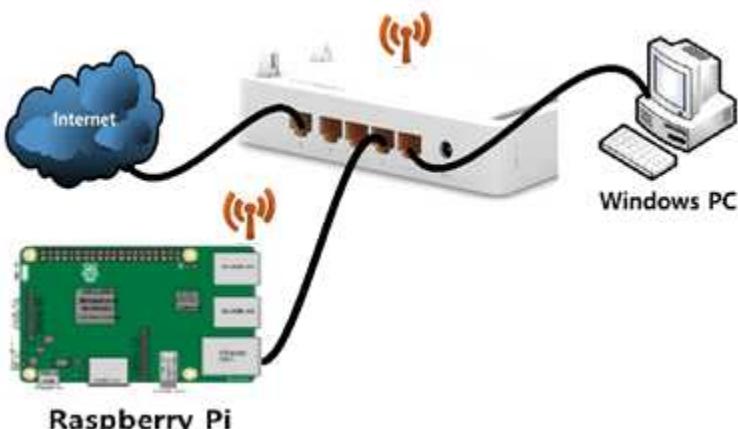
172.18.14.189에 대한 Ping 통계:
파킷: 보냄 = 4, 받음 = 4, 손실 = 0 (0% 손실),
왕복 시간(밀리초):
    최소 = 0ms, 최대 = 1ms, 평균 = 0ms

C:\Users\A211>

```

2.2 Wi-Fi 망 구축

무선공유기를 사용하여 Wi-Fi 망을 구축하는 것은 학교, 공공기관과 같이 고정 IP 주소를 사용하기보다 가정과 같이 ISP 업자로부터 IP 주소를 동적으로 할당 받아 통신하는 상황에서 보다 일반적으로 활용될 수 있는 방법이다.



Wi-Fi 망에 접속하기 위해서는 라즈베리파이 환경설정에서 Wi-Fi Country 항목이 필히 GB Britain (UK) 으로 설정되어 있어야한다. Change Wi-Fi Country 항

16 라즈베리파이기반 임베디드시스템용

목은 디폴트로 GB Britain (UK) 로 설정되어 있으며, 이 항목을 다른 것으로 변경하게 되면 Wi-Fi를 사용할 수 없게 되므로 유의할 필요가 있다.

2.2.1 초기 환경설정

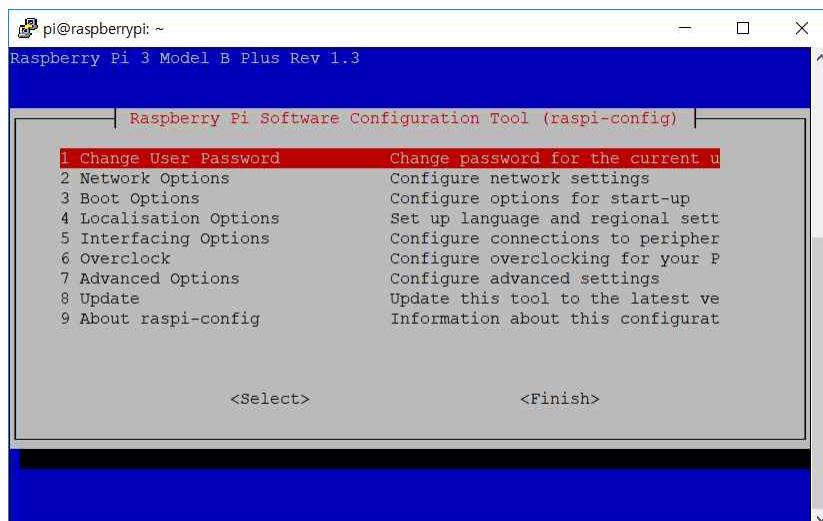
라즈베리파이를 활용하기에 앞서 pi계정의 패스워드 변경, 앞서 언급한 Wi-Fi 접속가능위한 조치 등의 몇가지 필요한 초기의 환경설정에 대해 살펴본다.

pi 계정의 암호 변경

따로 라즈베리파이 아이디와 패스워드를 설정하지 않았다면 기본 계정명은 ‘pi’, 암호는 ‘raspberry’로 설정돼 있다. pi 계정의 암호를 변경하기 위해서는 터미널 창을 열어 다음의 명령을 실행한다.

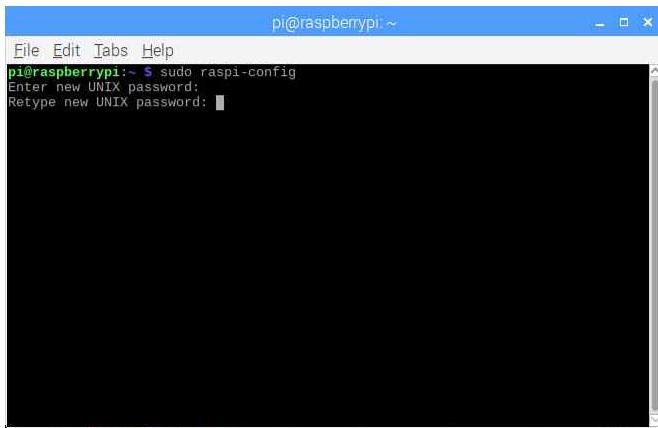
```
$ sudo raspi-config
```

항목중 1 Change User Password를 선택한다.



다음과 같이 암호 입력 창이 나타나면, pi 계정의 새 암호를 입력하고 확인차원에

서 다시 입력하면 암호의 변경이 완료된다. Terminal 창에서 암호를 입력할 때 글자가 보이지 않으므로 유의하여 동일하게 입력한다. 본 실습을 위해 pi 계정의 암호는 embedded 로 통일하자.



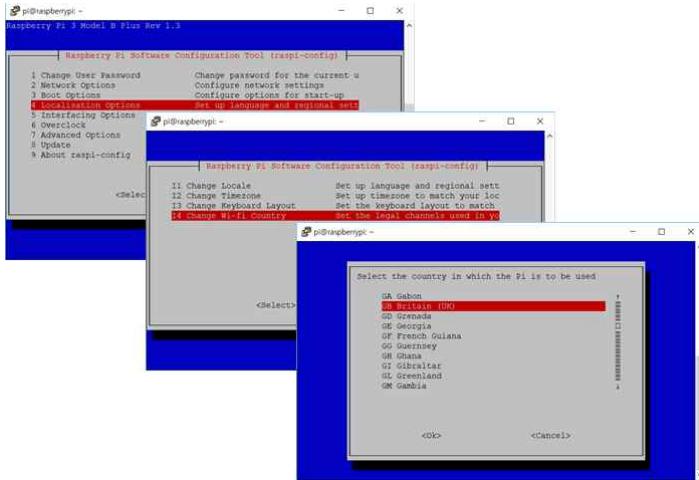
pi 계정의 암호가 변경되었으면 raspi-config를 종료하고 재부팅한다. 이후부터 라즈베리파이에 로그인할 때 계정명은 pi, 암호는 방금 설정한 것을 입력하면 된다.

Wi-Fi 접속을 위한 조치

Wi-Fi 망에 접속하기 위해서는 라즈베리파이에서 Wi-Fi Country 항목이 필히 GB Britain (UK) 으로 설정되어 있어야한다고 앞서 언급하였다. 다음의 명령을 사용하여 Localization - Wi-Fi Country 항목의 설정 정보가 GB Britain (UK) 으로 되도록 설정한다.

```
$ sudo raspi-config
```

18 라즈베리파이기반 임베디드시스템용



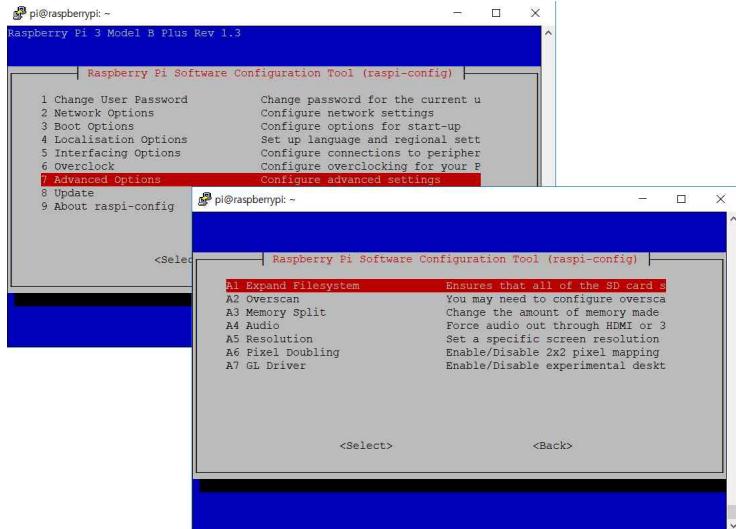
SD 메모리 최대 사용

라즈베리파이는 기본적으로 SD 카드 메모리 4GB를 사용하도록 되어 있다. 이보다 큰 사이즈의 SD 카드를 가지고 있는 경우 SD 메모리 최대 사용을 설정함으로써 보유하고 있는 SD 카드 용량 전체를 사용할 수 있다.

현재의 SD 카드 용량을 확인하려면 다음의 명령을 사용하면 된다.

\$ lsblk

SD 카드 메모리 최대 사용을 위해서는 항목중 7 Advance Options를 선택하고, 서브 항목에서 A1 Expand Filesystem을 선택한다.



micro SD 카드 용량이 16GB인 경우로 최대화한 후, micro SD 카드 용량 확인을 위해 `lsblk` 명령을 사용한 결과는 다음 화면과 같다.

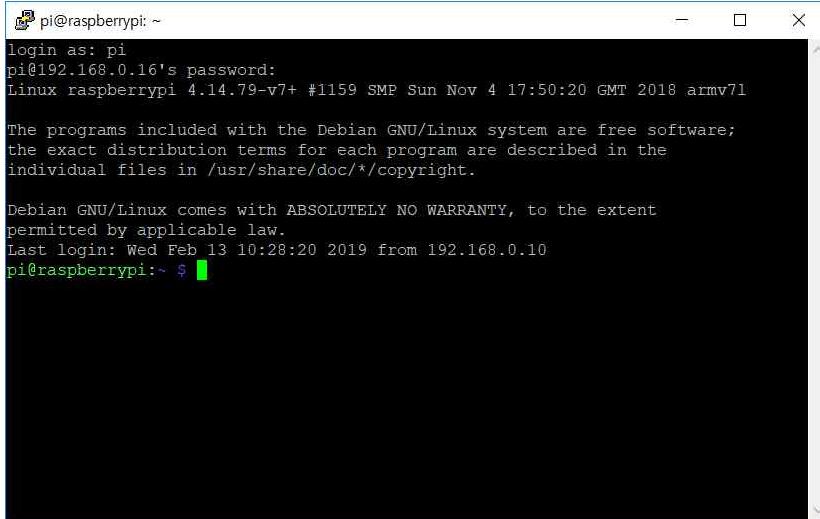
```
pi@raspberrypi: ~
pi@raspberrypi: ~ $ lsblk
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
mmcblk0     179:0   0 14.9G  0 disk
`-mmcblk0p1 179:1   0 43.9M  0 part /boot
`-mmcblk0p2 179:2   0 14.8G  0 part /
pi@raspberrypi: ~ $
```

pi 계정의 암호 변경, SD 메모리카드 최대 사용 등의 설정이 완료되었으므로 `raspi-config`를 종료하고 다음과 같이 재부팅한다.

```
$ sudo reboot
```

이후 로그인은 변경된 패스워드를 사용하여 로그인한다.

20 라즈베리파이기반 임베디드시스템용



```
pi@raspberrypi: ~
login as: pi
pi@192.168.0.16's password:
Linux raspberrypi 4.14.79-v7+ #1159 SMP Sun Nov 4 17:50:20 GMT 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Feb 13 10:28:20 2019 from 192.168.0.10
pi@raspberrypi:~ $
```

2.2.2 무선공유기 설정

ipTIME의 특정 모델인 ipTIME A304 무선공유기를 사용하여 Wi-Fi 망을 구축하는 과정에 대해 살펴본다. 내부 망의 IP 대역은 192.168.0.xx을 사용하기로 한다.

앞에서 살펴본 Wi-Fi 망 그림과 같이 무선공유기와 함께 LAN 케이블을 연결한다. 그리고 Windows PC의 IP 주소를 동일 대역의 IP 주소 192.168.0.10으로 설정한다.

무선공유기 접속

Win 웹브라우저 주소창에 다음의 주소를 입력한다. 이 주소는 무선공유기의 기본 IP 주소이다.

<http://192.168.0.1/>

정상적으로 접속된 후에는 망 관리자 로그인 화면이 나타난다. 이 망관리자 로그인 화면에서 초기에 설정된 기본 망관리자 계정명은 admin이다. 따라서 지시하는 대로 admin/admin(초기암호)/이미지문자를 입력하여 로그인한다.

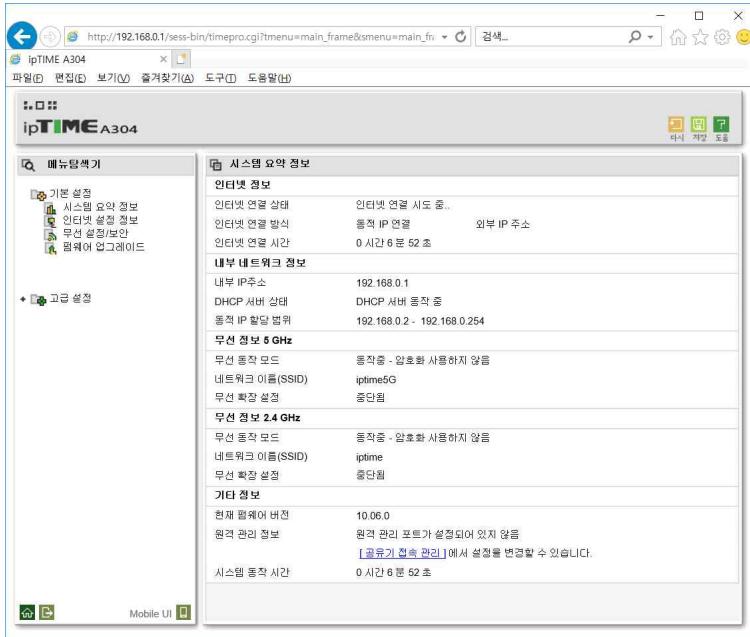


로그인한 후 다음과 같은 접속 초기 화면이 나타나면, 관리도구 아이콘을 클릭한다.



다음과 같이 무선공유기의 시스템 요약 정보의 화면이 나타난다. 초기에는 인터넷 연결상태 항목에 ‘인터넷 연결시도중...’ 메시지가 나타나면 아래에서 설명하는 내용들의 설정이 필요한 경우이다.

22 라즈베리파이기반 임베디드시스템용

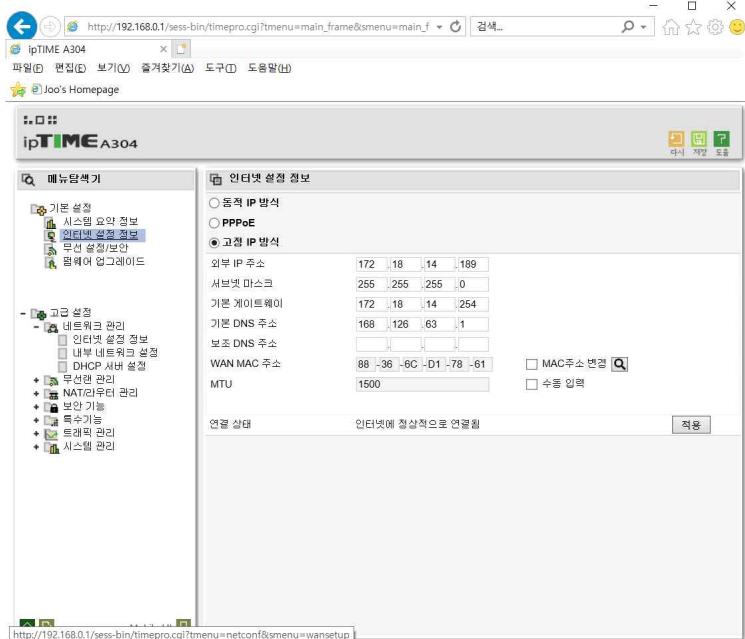


이 화면은 메뉴탐색기의 기본설정-시스템 요약 정보를 클릭한 경우와 동일한 내용을 보여준다.

무선공유기 설정

기본설정-인터넷 설정 정보

메뉴탐색기에서 기본설정-인터넷 설정 정보를 클릭하면 다음의 화면이 나타난다.

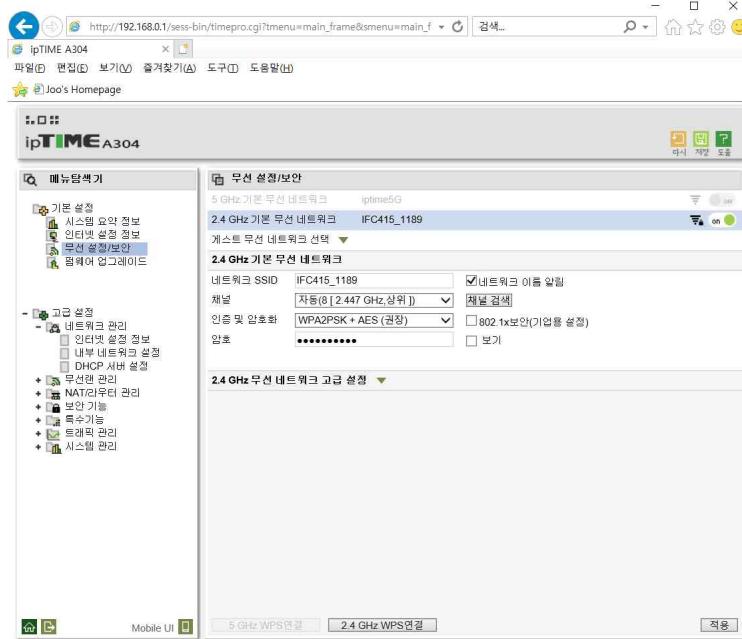


고정 IP 방식 항목을 체크하고, 외부 IP 주소 등의 정보 확인 및 설정 후 적용 버튼을 클릭한다. 여기서 외부 IP 주소는 기존 Windows 시스템에서 사용하던 고정 IP 주소임을 유의한다.

기본설정-무선 설정/보안

라즈베리파이는 2.4GHz의 Wi-Fi를 지원하므로, 5GHz 기본 무선 네트워크는 Off로 설정하고, 2.4GHz 기본 무선 네트워크를 On으로 한다.

24 라즈베리파이기반 임베디드시스템용



위의 화면에서 다음 정보를 설정한다. 네트워크 SSID(service set identifier)와 암호는 Wi-Fi 망에 접속할 때 필요한 정보로, 네트워크 SSID와 암호는 각자 정의해 되, 실습을 위해 아래의 형식을 따라 정의할 것을 권고한다.

- 네트워크 SSID : IFC415_xxxx
- 채널 : 자동
- 인증 및 암호화 : (권장)항목 선택
- 암호 : 000000xxxx

-네트워크 이름 알림 체크

입력이 완료되었으면 ‘적용’ 버튼 클릭하여 반영한다.

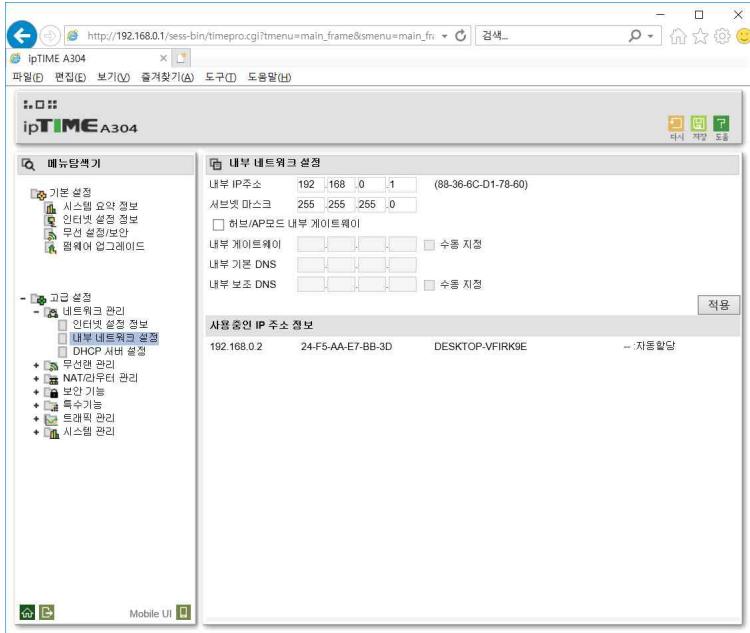
고급설정-네트워크관리-인터넷 설정 정보

고급설정-네트워크관리-인터넷 설정 정보는 기본설정-인터넷 설정 정보와 동일하다.

고급설정-네트워크관리-내부 네트워크 설정

다음 화면처럼 고급설정-네트워크관리-내부 네트워크 설정에서는 사용 중인 IP 주

소 정보들을 확인할 수 있다.

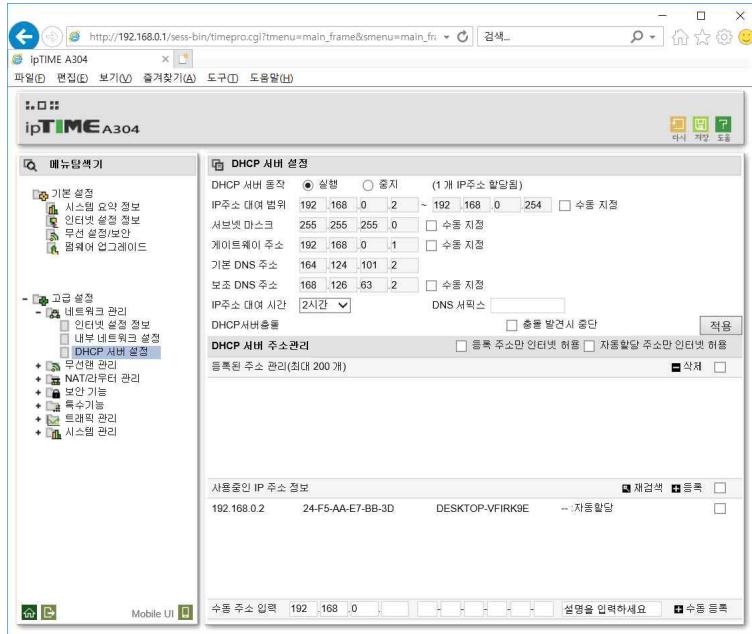


스마트폰 등이 Wi-Fi 망에 접속할 때, 무선공유기에 의해 내부 IP 주소가 동적으로 할당되며, 이론상 최대 200여대를 접속할 수 있으나 다수 접속의 경우 통신 속도가 느려질 수 있다.

고급설정-네트워크관리-DHCP 서버 설정

메뉴탐색기에서 고급설정-네트워크관리-DHCP 서버 설정을 클릭하면 다음 화면과 같이 나타난다.

26 라즈베리파이기반 임베디드시스템용

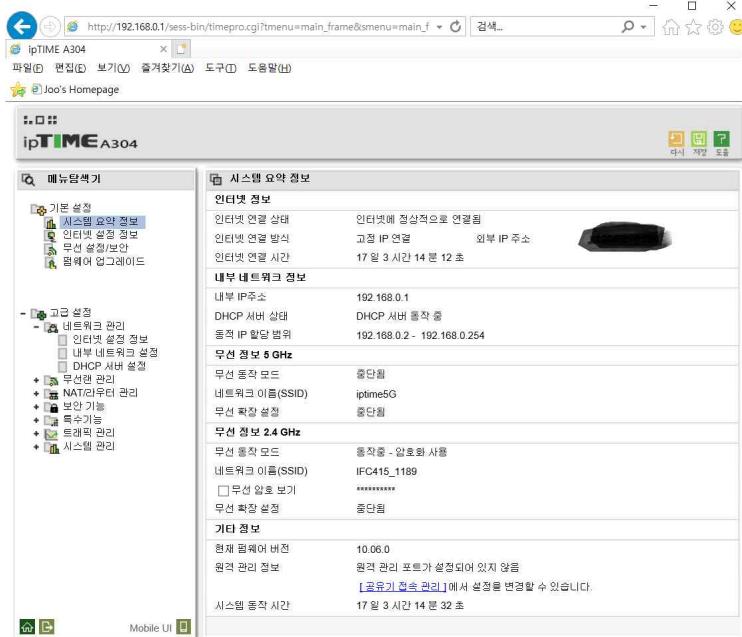


DHCP 서버 동작 항목에서 ‘실행’ 항목을 체크하고, 그 다음의 내용들을 확인한다. 특히, 게이트웨이 주소 항목의 IP 주소는 이 무선공유기를 경유하여 인터넷을 사용하는 시스템들에서 IP 주소를 설정할 때 gateway 주소로 사용되므로 잘 기억하여 두도록 한다.

이제 필요한 설정들이 완료되었으므로, 메뉴탐색기에서 기본설정-시스템 요약 정보를 클릭하여 인터넷연결 상태를 확인한다.

기본설정-시스템 요약 정보

앞에서 살펴보았듯이 인터넷 연결상태 항목에 ‘인터넷 연결시도중...’ 메시지 대신 ‘인터넷 정상적으로 연결됨’을 확인할 수 있다.



이로써 무선공유기의 환경설정이 완료되었다.

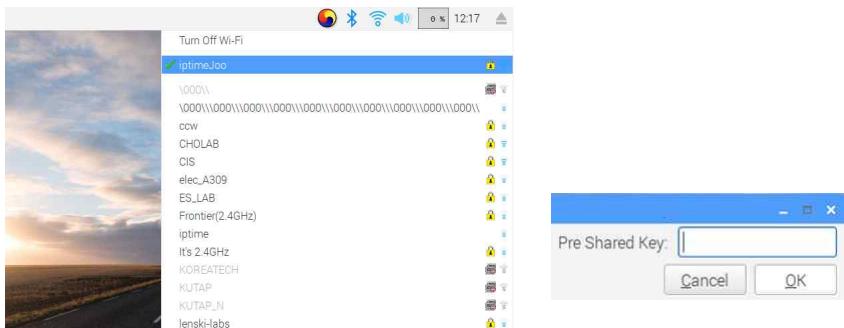
2.2.3 Wi-Fi 망 접속

라즈베리파이보드로 SBC를 구성한 후, 라즈베리파이에서 다음 그림에서 보듯이 우상단의 아이콘들 중 화살표 모양의 아이콘을 선택하여, 무선 공유기 목록이 표시되면 접속할 무선네트워크를 선택한다.

28 라즈베리파이기반 임베디드시스템용



목록중 접속할 네트워크 이름의 무선공유기를 선택하면 다음과 같이 암호를 입력 화면이 나타나고 네트워크에 접속할 암호를 입력하면 Wi-Fi 망 접속이 된다. Wi-Fi 망에 접속이 되면 화면 우상단의 화살표 아이콘이 Wi-Fi 아이콘으로 변경된다. 이로서 Wi-Fi 망에 접속이 완료된다.



터미널 창을 열어 다음의 명령을 통해 무선공유기에 의해 동적으로 할당된 내부 IP 주소를 확인할 수 있으며, 192.168.0.15로 할당되었음을 확인할 수 있다.

```
$ ifconfig wlan0
```

```
pi@raspberrypi:~ $ ifconfig wlan0
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.0.15 netmask 255.255.255.0 broadcast 192.168.0.255
        inet6 fe80::f4c:7e3b:8a7:f73 prefixlen 64 scopeid 0x20<link>
          ether b8:27:eb:4d:9f:65 txqueuelen 1000  (Ethernet)
            RX packets 300200 bytes 31851241 (30.3 MiB)
            RX errors 0 dropped 1 overruns 0 frame 0
            TX packets 333506 bytes 268040284 (255.6 MiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

다음과 같이 PC에서 명령프롬프트 창을 열어 라즈베리파이와의 통신이 가능한지 ping하여 통신가부를 확인한다.

```
C:\> ping 192.168.0.15
```

2.2.4 IP 주소 변경

실습을 위해 Windows 시스템과 라즈베리파이의 IP 주소를 변경하는 과정에 대해 살펴본다.

Windows의 IP 주소 설정

초기 무선공유기 설정한 상태에서는 Windows 시스템에서 사용하던 IP 주소는 Wi-Fi 망의 외부 IP 주소로 사용되고, Windows 시스템은 무선공유기의 DHCP 서버에 의해 동적 IP 주소가 설정되어 사용 중인 상태이다.

확인을 위해 다음 화면과 같이 Windows 환경에서 명령프롬프트 창을 열어 ipconfig 명령을 사용하여 자동 설정된 내부 IP 주소를 확인할 수 있다. 현재 자동 할당된 내부 IP 주소는 192.168.0.16으로 확인된다. Windows 시스템에 설정되었던 기존의 고정 IP 주소는 이후부터 무선공유기의 외부 IP 주소로 사용된다.

```
C:\> ipconfig
```

30 라즈베리파이기반 임베디드시스템용

```
C:\Users\Administrator>ipconfig

Windows IP 구성

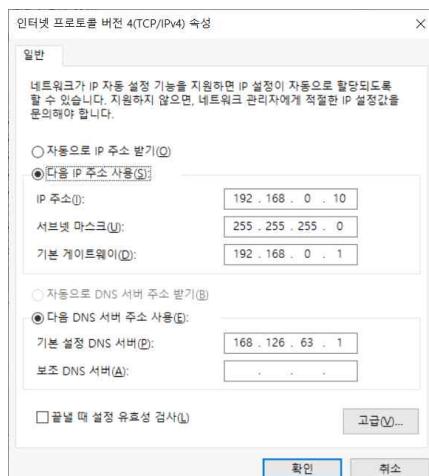
이더넷 어댑터 로컬 영역 연결:
  연결별 DNS 접미사. . . . . : 
  링크-로컬 IPv6 주소 . . . . . : fe80::301e:e7a3:af5:a8fa%11
  IPv4 주소 . . . . . : 192.168.0.16
  서브넷 마스크 . . . . . : 255.255.255.0
  기본 게이트웨이 . . . . . : 192.168.0.1

이더넷 어댑터 VMware Network Adapter VMnet1:
  연결별 DNS 접미사. . . . . : 
  링크-로컬 IPv6 주소 . . . . . : fe80::2d54:78e0:5703:6f7a%14
  IPv4 주소 . . . . . : 192.168.197.1
  서브넷 마스크 . . . . . : 255.255.255.0
  기본 게이트웨이 . . . . . : 

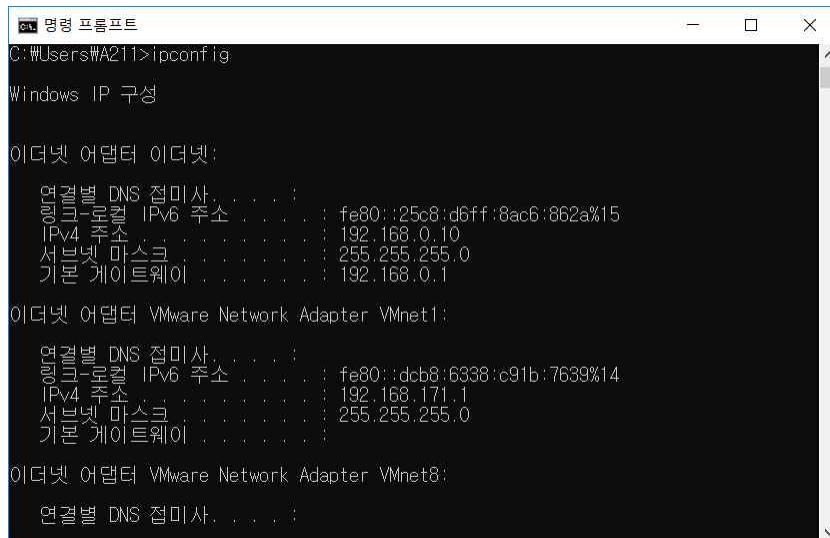
이더넷 어댑터 VMware Network Adapter VMnet8:
  연결별 DNS 접미사. . . . . :
```

실습을 위해 Windows 시스템의 IP 주소를 변경하는 과정은 다음과 같다. 설정 IP 주소는 192.168.0.10 이라고 가정한다.

이더넷-어댑터 옵션 변경에서 다음 화면과 같이 설정하고자하는 IP 주소, 서브넷마스크를 입력하고, 기본 게이트웨이 항목에는 앞서 무선공유기 설정에서 기억하라고 하였던 기본 게이트웨이 주소를 입력하고, DNS 서버주소 사용을 선택하고, 기본설정 DNS 서버의 주소로 168.126.63.1을 입력한다.



입력이 완료되면, Windows 시스템의 다시 시작을 클릭하여 재부팅한다. 새로이 설정된 IP 주소는 다음 화면과 같이 명령프롬프트 창을 열어 ipconfig 명령으로 확인할 수 있다.



```
C:\Users\#A211>ipconfig

Windows IP 구성

이더넷 어댑터 이더넷:
  연결별 DNS 접미사 . . . . . : fe80::25c8:d6ff:8acd:862a%15
  링크-로컬 IPv6 주소 . . . . . : 192.168.0.10
  IPv4 주소 . . . . . : 192.168.0.10
  서브넷 마스크 . . . . . : 255.255.255.0
  기본 게이트웨이 . . . . . : 192.168.0.1

이더넷 어댑터 VMware Network Adapter VMnet1:
  연결별 DNS 접미사 . . . . . : fe80::dcbb:6338:c91b:7639%14
  링크-로컬 IPv6 주소 . . . . . : 192.168.171.1
  IPv4 주소 . . . . . : 192.168.171.1
  서브넷 마스크 . . . . . : 255.255.255.0
  기본 게이트웨이 . . . . . :

이더넷 어댑터 VMware Network Adapter VMnet8:
  연결별 DNS 접미사. . . . . :
```

네트워킹 가능 여부는 ping 명령을 사용하거나, 웹브라우저를 실행하여 특정사이트 접속시도로 확인할 수 있다.

라즈베리파이 IP 주소 변경

라즈베리파이의 IP 주소를 변경하는 과정에 대해 살펴본다. 라즈베리파이에 현재 설정된 IP 주소를 확인하기 위해 다음의 명령을 사용한다.

\$ ifconfig

```
eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
      inet 172.18.14.189 netmask 255.255.255.0 broadcast 172.18.14.255
          inet6 fe80::feaf:3dddf:d3dc:c250 prefixlen 64 scopeid 0x20<link>
          .......
```

```
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.0.15 netmask 255.255.255.0 broadcast 192.168.0.255
```

32 라즈베리파이기반 임베디드시스템용

```
inet6 fe80::feaf:3ddf:d3dc:c250  prefixlen 64  scopeid 0x20<link>
```

.....

eth0 항목은 유선망의 IP 설정 정보를, wlan0 항목은 무선망의 IP 설정 정보를 나타낸다.

실습환경에 맞추어 유선 및 무선의 IP 주소를 각각 192.168.0.30과 192.168.0.40으로 설정하기로 가정한다. 이를 위해서는 /etc/dhcpcd.conf 파일을 편집하여야 한다. DNS 서버 주소는 168.126.63.1로 설정한다.

```
$ sudo nano /etc/dhcpcd.conf
interface eth0
    static ip_address=192.168.0.30
    static netmask=255.255.255.0
    static routers=192.168.0.1
    static domain_name_servers=168.126.63.1
interface wlan0
    static ip_address=192.168.0.40
    static netmask=255.255.255.0
    static routers=192.168.0.1
    static domain_name_servers=168.126.63.1
```

저장후 설정된 IP 주소를 반영하기 위해 다음의 명령을 사용하여 재부팅한다.

```
$ sudo reboot
```

재부팅후 ifconfig 명령을 사용하여 설정된 IP 주소를 확인한다.

2.3 개발 환경 툴

라즈베리파이는 HDMI 케이블을 통해 기존의 모니터와 연결하고, 키보드와 마우스를 USB 포트에 연결하면 하나의 완전한 컴퓨터로 사용할 수 있다. 그러나 전용의

모니터, 키보드, 마우스가 없더라도 또 다른 컴퓨터에서 원격으로 접속하여 운영할 수 있다. 특히 소스 코딩 등의 개발 환경에서는 원격으로 접속하여 운영하는 것이 보다 효율적일 수 있다.

원격 접속을 통한 개발 환경을 구축하는데 유용한 툴들로 다음의 것들이 있다. 이들을 설치하고 운영하는 방법을 살펴보도록 한다.

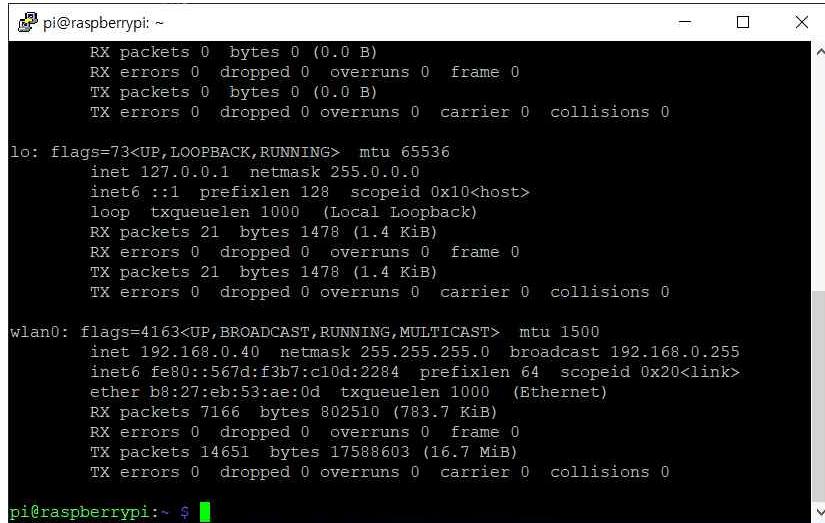
- PuTTY
- NFS 서비스
- FTP(FileZilla) 서비스
- samba 서비스
- mstsc(Microsoft Terminal Services Client) 원격 접속
- DD for Windows

IP 주소 확인

원격 접속하려면 우선 라즈베리파이의 IP 주소를 알고 있어야 하며, ifconfig 명령으로 라즈베리파이의 IP 주소를 확인할 수 있다. 라즈베리파이에 유선 랜을 통해 원격 접속하려면 ‘eth0’ 항의, Wi-Fi 무선 랜으로 원격 접속하려면 ‘wlan0’ 항의 IP 주소를 사용한다.

```
$ ifconfig
```

34 라즈베리파이기반 임베디드시스템용



```
pi@raspberrypi: ~
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        RX packets 21 bytes 1478 (1.4 Kib)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 21 bytes 1478 (1.4 Kib)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.40 netmask 255.255.255.0 broadcast 192.168.0.255
    inet6 fe80::567d:f3b7:c10d:2284 prefixlen 64 scopeid 0x20<link>
        ether b8:27:eb:53:ae:0d txqueuelen 1000 (Ethernet)
        RX packets 7166 bytes 802510 (783.7 Kib)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 14651 bytes 17588603 (16.7 MiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

pi@raspberrypi: ~ $
```

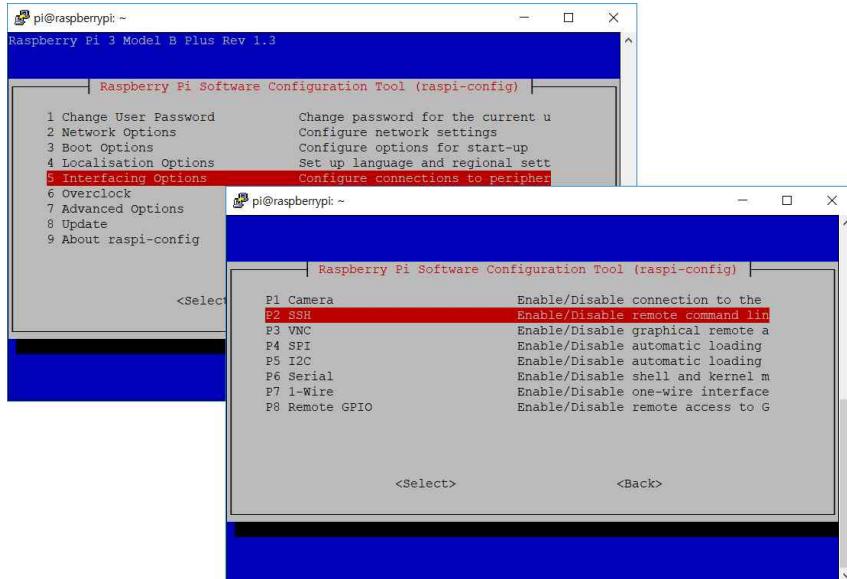
SSH 활성화

SSH(Secure Shell)은 네트워크 상의 다른 컴퓨터에 로그인하거나 원격 시스템에서 명령을 실행하고 다른 시스템으로 파일을 복사할 수 있도록 해 주는 응용 프로그램 또는 그 프로토콜을 가리킨다. 기존의 rsh, rlogin, 텔넷 등을 대체하기 위해 설계되었으며, 강력한 인증 방법 및 안전하지 못한 네트워크에서 안전하게 통신을 할 수 있는 기능을 제공한다. 기본적으로는 22번 포트를 사용한다. SSH는 통신할 때 암호화 기법을 사용하기 때문에 보다 안전하다.

어떠한 툴을 사용하든 원격 접속을 하려면 우선 다음의 명령을 사용하여 라즈베리파이의 raspi-config에서 SSH 서비스를 활성화하여야 한다.

```
$ sudo raspi-config
```

Interfacing Options 항목을 선택하여 그 서브 화면의 SSH 항목을 활성화한다.



이상으로 라즈베리파이에서의 설정은 완료되었다. 그리고 재부팅한다.

2.3.1 PuTTY 원격 접속

puTTY를 이용한 라즈베리파이의 접속을 알아보자. PuTTY는 서버가 물리적으로 떨어져 있어도 원격으로 접속해 작업을 지원하는 가상 터미널 프로그램이다. PuTTY는 SSH, 텔넷, rlogin, raw TCP를 위한 클라이언트로 동작하는 프리 및 오픈 소스 단말 애플리케이션 프로그램이다. PuTTY라는 이름에는 특별한 뜻이 없으나 tty는 유닉스 전통의 터미널의 이름을 가리키며 teletype를 짧게 줄인 것이다.

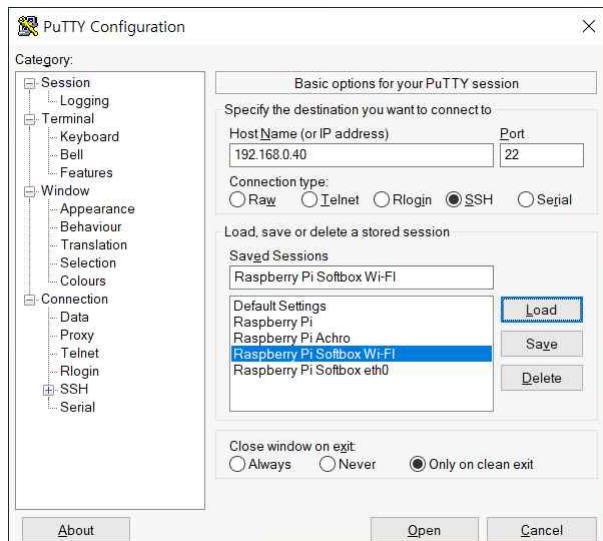
PuTTY 실행

웹브라우저에서 다음 그림과 같이 PuTTY 다운로드 사이트 <https://www.chiark.greenend.org.uk/~sgtatham/putty/>에 접속하여 다운로드 한다.

36 라즈베리파이기반 임베디드시스템용



PutTY를 실행하면 다음과 같은 초기 화면이 나타난다. 좌측 'Session'을 클릭하고 우측의 'Host Name'에 접속하고자하는 라즈베리파이의 IP 주소를 입력한다. 그리고 'Connection type'에 SSH가 제대로 체크됐는지 확인 후 'Open'을 클릭한다. 필요에 따라 현재 접속하는 세션 정보를 저장하였다가 나중에 필요시 로드하여 간편히 재접속하는 것이 가능하다.



PutTY를 통해 원격접속하면 다음과 같은 화면이 나타나며, 접속할 계정의 ID와

패스워드를 입력하면 된다. PuTTY를 통한 원격접속은 텍스트 모드의 CLI 환경을 제공한다.

```

pi@raspberrypi: ~
login as: pi
pi@192.168.0.40's password:
Linux raspberrypi 4.14.98-v7+ #1200 SMP Tue Feb 12 20:27:48 GMT 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Feb 25 12:25:28 2019 from 192.168.0.10
pi@raspberrypi:~ $
```

2.3.2 NFS 서비스

NFS 서비스는 네트워크 상의 다른 시스템의 파일시스템을 공유하여 파일 등을 공유하기 위한 서비스이다. 특히 가상머신인 Ubuntu와 라즈베리파이 사이에 파일을 공유할 때 유용하게 사용되는 서비스이다.

가상머신의 파일시스템 내 특정 디렉터리를 라즈베리가 공유하도록 하는 과정에 대해 살펴본다. 즉, NFS 서버를 가상머신 상에 설치하고, 가상머신상의 /nfs 디렉터리를 라즈베리가 공유할 수 있도록 설정하는 과정에 대해 살펴본다.

NFS 서버 구축

다음의 명령들로 새로운 패키지를 설치하기에 앞서 기존 설치된 패키지를 최신버전으로 업데이트 및 업그레이드 한다. 첫 명령은 기존 설치된 패키지를 최신 버전으로 업데이트 즉, 다운로드하는 명령이고, 그 다음의 명령은 다운로드된 최신 버전의 패키지를 설치하는 명령이다.

38 리즈베리파이기반 임베디드시스템용

```
root@ubuntu:~# apt-get update  
root@ubuntu:~# apt-get upgrade
```

이제 가상머신에서 다음 명령을 사용하여 NFS 서버 패키지를 설치한다.

```
root@ubuntu:~# apt-get install nfs-kernel-server  
Reading package lists... Done  
.....
```

다음으로 외부에서 공유할 디렉터리 /nfs를 다음의 명령을 사용하여 생성한다.

```
root@ubuntu:~# mkdir /nfs
```

다음으로 NFS 서비스를 위한 환경 설정을 위해 다음의 명령을 사용하여 /etc(exports 파일을 편집한다.

```
root@ubuntu:~# gedit /etc/exports  
/nfs *(rw,sync,no_root_squash,no_subtree_check)
```

/nfs는 NFS 서비스할 디렉터리로 공유할 파일 등을 위치시킬 디렉터리이며, *는 동일 네트워크 주소를 사용하는 모든 IP 주소가 공유할 수 있음을 나타낸다. 특정 IP 주소의 시스템만 공유할 수 있도록 하려면 *대신 그 특정 IP 주소를 기입하면 된다. ()속의 내용은 공유 디렉터리에 대한 접근권한을 명시하는 것이다.

작성이 완료되면 저장하고 다음의 명령들을 사용하여 NFS 서비스를 시작하거나 재부팅한다.

```
root@ubuntu:~# service nfs-kernel-server start  
root@ubuntu:~# service rpcbind restart
```

혹은, root@ubuntu:~# reboot

NFS 서비스

우선 가상머신의 서버 측에서는 다음 명령을 사용하여 공유할 파일들을 NFS 서비스위한 디렉터리로 복사하고, 복사여부를 확인한다. 현 작업디렉터리에 h_hello와 t_hello라는 실행파일이 있다고 가정한다.

```
root@ubuntu:~# cp *_hello /nfs
root@ubuntu:~# ls /nfs
h_hello      t_hello
```

이 과정을 통하여 NFS 서버 측에서는 NFS 서버가 실행 중에 있고, 공유할 파일들도 /nfs에 복사되어 있는 상태이다.

이제 라즈베리파이 보드에서 NFS 서버 측의 공유 디렉터리를 적절한 곳에 마운트하여 파일들을 공유할 수 있다. 우선 라즈베리파이에 puTTY로 접속한 후, 터미널창을 열어 마운트 포인트로 사용할 디렉터리 /share를 다음의 명령을 사용하여 생성한다. 파일시스템의 / 바로 밑에 생성하는 것이 편리할 것이다.

```
$ sudo mkdir /share
```

이제 서버 측의 공유디렉터리인 /nfs를 라즈베리파이의 /share에 마운트하기 위해 다음의 명령을 사용한다.

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

마운트가 정상적으로 되면, 다음의 명령을 사용하여 공유 디렉터리의 내용을 확인할 수 있다. 이제 NFS 서버 측의 /nfs의 내용은 라즈베리파이의 /share 디렉터리를 통하여 공유하므로, 파일들을 접근하여 편집하거나 실행 등을 할 수 있다.

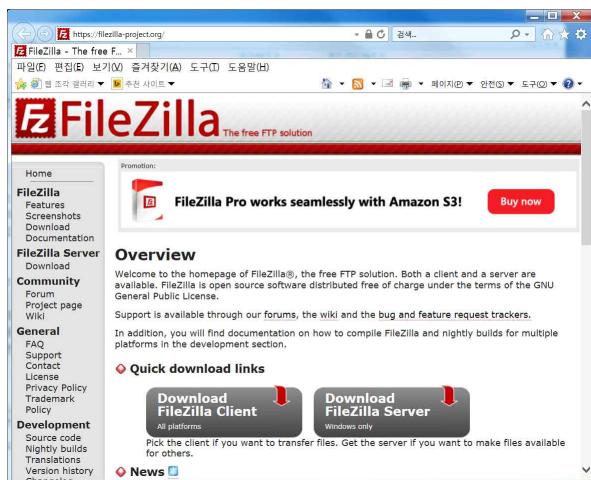
```
$ sudo ls /share
h_hello      t_hello
```

40 라즈베리파이기반 임베디드시스템용

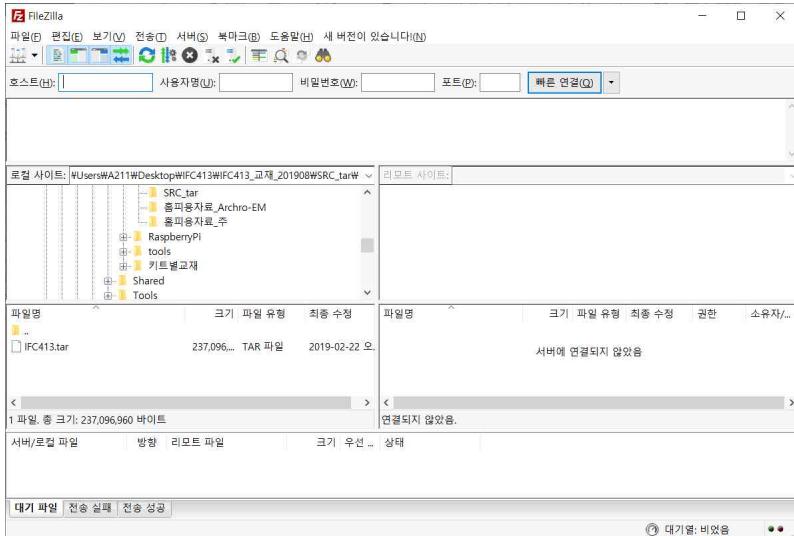
2.3.3 SFTP 서비스

FileZilla FTP client

FileZilla는 FTP 서비스를 받기 위한 FTP 클라이언트 프로그램이다. 웹브라우저에서 다음과 같이 FileZilla의 사이트 <https://filezilla-project.org/>에 접속하고, FileZilla Client를 클릭하여 FileZilla FTP Client를 다운로드한다.



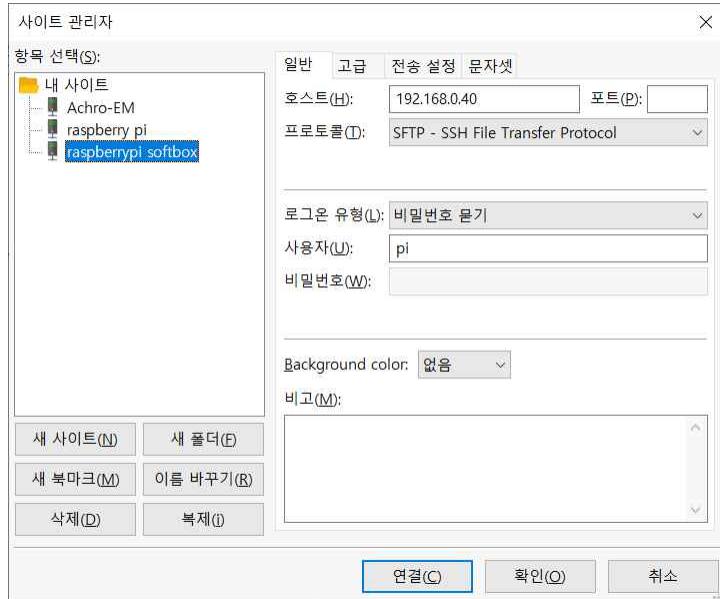
다운로드한 FileZilla를 실행하여 설치한다. 설치가 종료되면 바탕화면의 FileZilla 아이콘을 클릭하여 FTP 클라이언트를 실행시킨다. FileZilla의 실행 초기화면은 다음과 같다.



이 화면에서 좌측은 클라이언트 시스템 측의 파일시스템이, 우측은 접속한 서버 측의 파일 시스템의 용도로 사용되며, 상단 화면에서는 취한 동작의 내역을 확인할 수 있다.

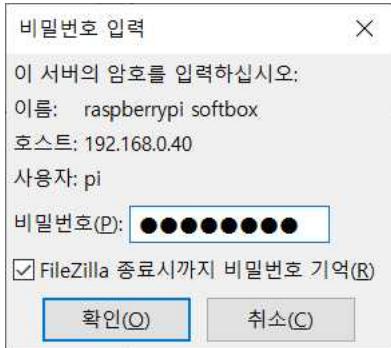
이제 FTP 서비스를 위해 서버에 접속하려면, 화면의 좌상단의 ‘사이트 관리자 열기’ 아이콘을 클릭하거나, 메뉴의 ‘파일 – 사이트관리자’ 항목을 선택한다. 그러면 다음과 같은 사이트 관리자 창이 나타난다.

42 라즈베리파이기반 임베디드시스템용

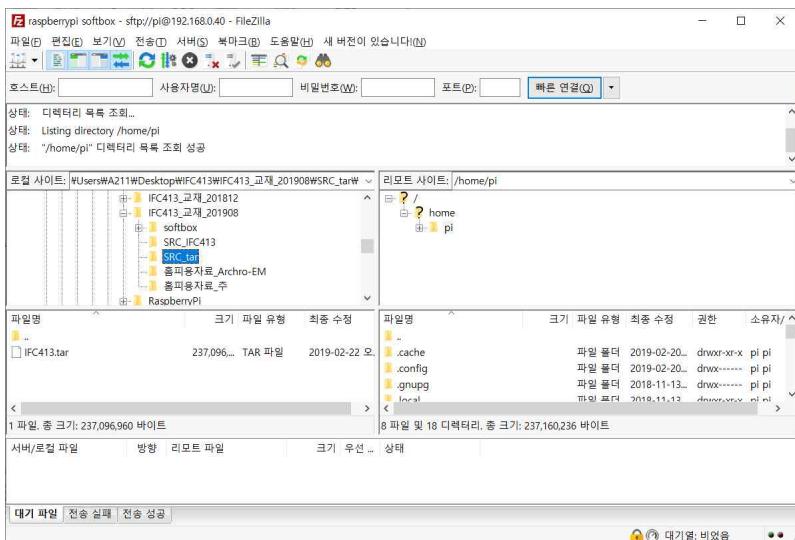


좌측화면의 새 사이트 버튼을 클릭하여 접속할 사이트에 대한 정보를 입력하면 된다. 필요에 따라 연결 정보를 등록 관리하여 활용할 수 있다. 우측화면의 일반 탭의 호스트 항목 입력란에 접속할 시스템의 IP 주소를, 프로토콜 선택 란에서 SFTP(SSH FTP)를, 로그온 유형란에서 비밀번호 묻기, 로그온의 사용자 입력란에 pi를 입력하고, 연결 버튼을 클릭한다. 라즈베리파이에 접속하기 위해서는 필히 프로토콜을 SFTP로 선택한다. 다음과 같이 패스워드 입력 화면에서 패스워드를 입력하여 접속한다.

추후 관련정보를 활용한 접속을 위해 좌측 항목선택에 새이름을 부여하여 등록하여 재활용 할 수 있다.



정상적으로 접속하게 되면, 다음 그림의 우측과 같이 로그인 계정의 홈 디렉터리의 파일시스템 구조가 나타난다.



파일 전송을 위해서는 화면 우측의 서버 파일시스템의 적절한 디렉터리를 선택한다. 초기 접속의 경우 로그인 계정의 홈 디렉터리이므로 기본적으로 로그인 계정의 홈 디렉터리 내로만 파일 전송이 가능하다. 또한 로그인 홈 디렉터리는 디폴트로 로그인 계정에게 기록권한이 부여되어 있으므로 저장이 가능하다. 이제 좌측의 클라이언트 쪽에서 파일 혹은 폴더를 선택하고 더블클릭하면 서버 측의 해당 위치로 전송되고 우측화면에서 확인 가능하다. 서버 측의 파일을 삭제하려면 서버 측 화면에서 파일을 선택하고 마우스 우측버튼 눌러 삭제 항목을 선택하면 된다.

2.3.4 Samba 서비스

인터넷 상에서 파일을 주고받기 위해서는 보통 FTP 서버를 구축해서 사용하는데, 또 다른 방식으로는 삼바 서비스가 있다. FTP가 웹브라우저나 전용 FTP 클라이언트 프로그램을 사용하는 것에 비해, 삼바 서비스는 윈도우즈에서 네트워크 드라이브로 등록하면 윈도우즈의 폴더를 다루듯이 사용할 수 있는 서비스이다. FTP 서비스가 주로 외부 망에서 사용된다면 삼바 서비스는 내부 망 내의 컴퓨터간 파일을 공유할 때 주로 사용된다.

라즈베리파이에서 삼바 서버를 구축하고 사용하는 방법을 살펴보자.

삼바 서버 패키지 설치

삼바 서버 패키지를 설치하는 과정으로 일단 기존 설치된 패키지들을 업데이트 및 업그레이드한다.

```
$ sudo apt-get update  
$ sudo apt-get upgrade
```

다음으로 samba와 samba-common-bin 패키지를 다음의 명령들을 사용하여 설치한다.

```
$ sudo apt-get install samba  
$ sudo apt-get install samba-common-bin
```

삼바서버 환경 설정

삼바 서버의 환경 설정파일 /etc/samba/smb.conf 파일을 편집하기 위해 다음의 명령을 사용한다. 환경설정 파일의 끝에 다음의 내용을 추가한 후 저장한다.

```
$ sudo nano /etc/samba/smb.conf
```

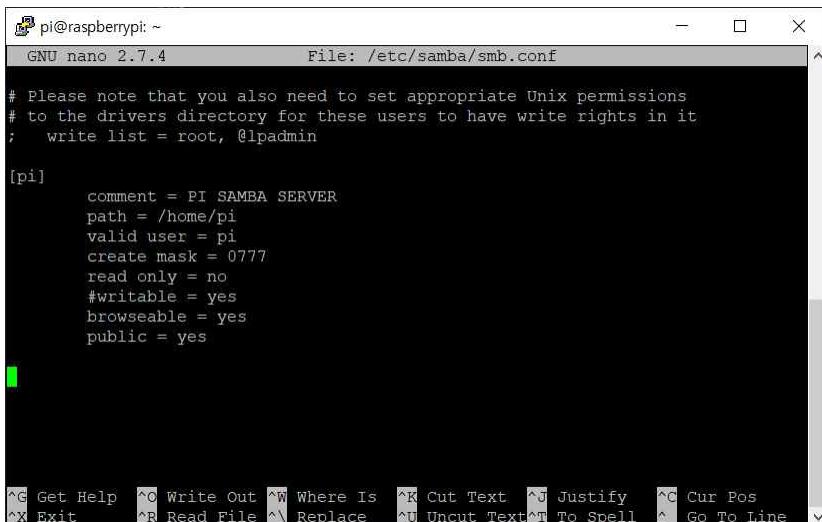
.....
[pi]

```

comment = PI SAMBA SERVER
path = /home/pi/
valid user = pi
create mask = 0777
read only = no           # 혹은, writable = yes
browseable = yes
public = yes

```

path 항목에는 삼바 서버 측에서 공유의 목적으로 제공할 디렉터리의 경로를 설정한다. 그리고 유효 사용자, 접근 권한 등을 설정하는 내용을 포함하고 있다.



```

pi@raspberrypi: ~
GNU nano 2.7.4          File: /etc/samba/smb.conf

# Please note that you also need to set appropriate Unix permissions
# to the drivers directory for these users to have write rights in it
;   write list = root, @lpadmin

[pi]
comment = PI SAMBA SERVER
path = /home/pi
valid user = pi
create mask = 0777
read only = no
#writable = yes
browseable = yes
public = yes

^G Get Help  ^C Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos
^X Exit      ^R Read File  ^Y Replace  ^U Uncut Text  ^T To Spell  ^L Go To Line

```

삼바 사용자의 비밀번호 등록

FTP 서버는 라즈베리파이 운영체제인 라즈비안에서 사용하는 아이디와 패스워드를 그대로 사용하지만, 삼바 서버의 경우 삼바 사용자의 패스워드를 따로 설정해야한다. 삼바 사용자의 패스워드를 변경하는 명령은 다음과 같은 smbpasswd이며, 사용자 계정 pi의 삼바용 패스워드를 설정하기 위해 다음의 명령을 사용한다.

```
$ sudo smbpasswd -a pi
```

46 라즈베리파이 기반 임베디드 시스템 활용

```
pi@raspberrypi: ~  
pi@raspberrypi:~ $ sudo smbpasswd -a pi  
New SMB password:  
Retype new SMB password:  
pi@raspberrypi:~ $
```

이렇게 하면 삼바 서버에 접속할 수 있는 계정과 그 패스워드를 추가 등록하게 된다. 패스워드는 embedded로 통일하자.

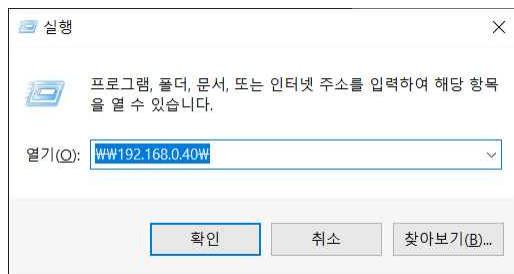
삼바 서버 재실행

환경 설정이 변경되었으므로 재부팅하거나, 다음의 명령을 사용하여 삼바 서버를 재실행하여 변경된 환경설정 정보가 반영되어 실행될 수 있게 한다.

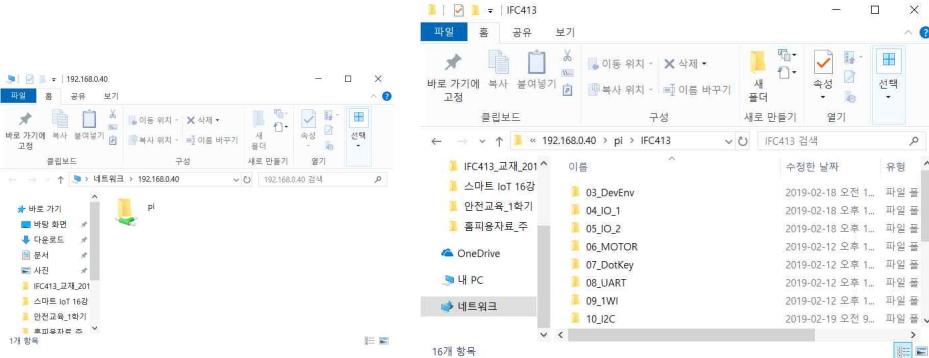
```
$ sudo /etc/init.d/samba restart
```

윈도우즈에서 네트워크 드라이브 연결

삼바 서버의 공유영역을 접속하려면 웹브라우저 주소창, 혹은 프로그램 및 파일검색 창에서 삼바 서버 시스템의 IP 주소를 \\192.168.0.40\와 같이 입력한다.



그러면 삼바 서버에서 공유하기로 한 디렉터리가 다음 그림과 같이 원도우즈의 폴더 형태로 나타나고 이후 원도우즈의 폴더에서 파일을 접근하듯이 사용하면 된다.

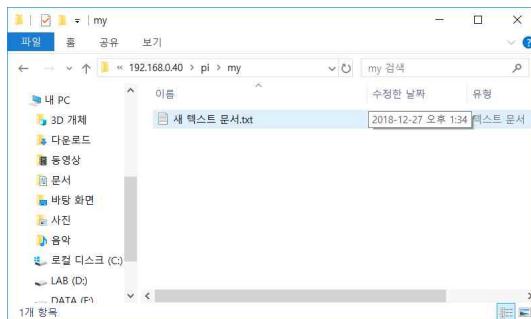


디렉터리 접근권한 변경

필요에 따라 공유할 디렉터리의 접근 권한을 변경할 필요가 있을 때는 다음과 같이 chmod 명령을 사용할 수 있다. 다음의 명령은 현 작업디렉터리의 ./my 디렉터리 및 그 하부에 대해 접근권한을 777로 설정한다.

```
$ sudo cd /home/pi
$ sudo mkdir my
$ sudo chmod -R 777 ./my
```

Windows 시스템에서 라즈베리파이의 /home/pi/my 디렉터리 하부에 대해 파일 생성, 삭제 등등이 가능하게 된다.



2.3.5 mstsc 원격 접속

48 라즈베리파이 기반 임베디드 시스템 활용

앞서의 PuTTY 접속과 마찬가지로 원격 데스크톱 연결 또한 SSH 서비스가 활성화되어 있어야 하며, 그 절차는 앞에서 살펴보았다.

RDP(Remote Desktop Protocol)는 마이크로소프트에서 개발한 GUI 형식으로 원격 접속 프로토콜이다. 이것을 본떠서 만든 것이 오픈진영의 XRDP다. SSH가 쉘 중심의 CUI(character based user interface) 혹은 CLI(command line interface)라고 한다면, XRDP는 GUI(graphical user interface)를 위해서 만들어진 프로그램이다.

xrdp 패키지 설치

새로운 패키지 설치에 앞서 우선 다음 명령들을 사용하여 기존 설치된 패키지들을 업데이트 및 업그레이드한다.

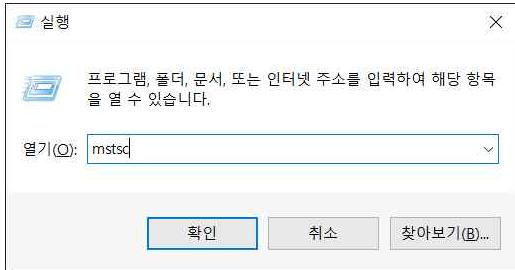
```
$ sudo apt-get update  
$ sudo apt-get upgrade
```

다음의 명령으로 xrdp 패키지를 설치한다. 설치 중간에 다음 메시지 나오면 y를 입력하여 진행한다.

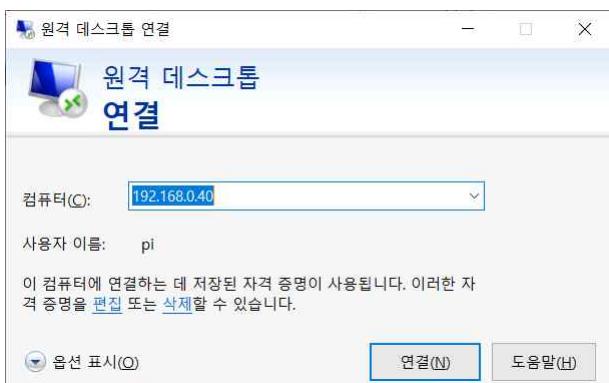
```
$ sudo apt-get install xrdp  
.....  
Do you want to continue? [Y/n] // y 입력  
.....
```

윈도우즈에서의 원격 접속

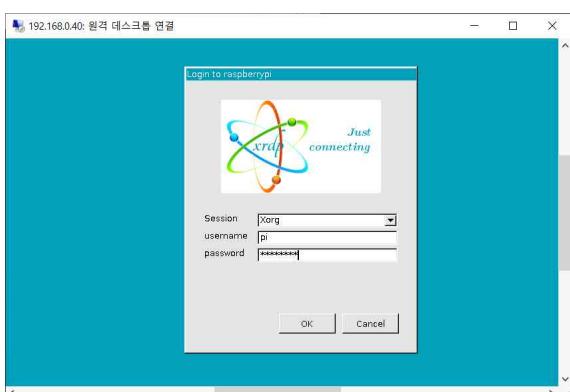
라즈베리파이에 원격 접속하려면 윈도우에서 시작 아이콘을 클릭 후 다음 그림과 같이 ‘실행’ 입력 창에서 mstsc(Microsoft Terminal Services Client)를 입력하여 실행한다.



그런 후 다음과 같은 원격 데스크톱 연결 화면에서 라즈베리파이의 IP 주소를 입력 후 연결을 클릭한다.

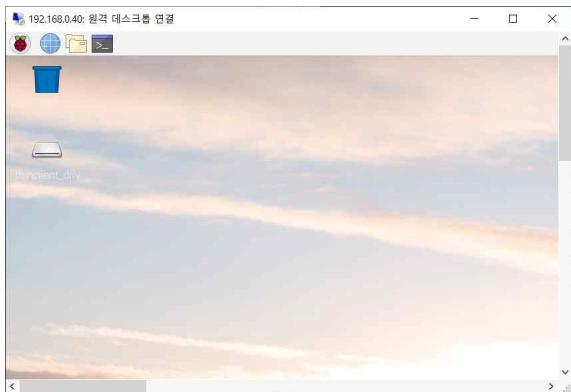


원격 접속이 되면 다음과 같은 로그인 화면이 출력되고 계정 아이디와 패스워드를 입력하여 로그인한다.



50 라즈베리파이기반 임베디드시스템용

이제 원격의 컴퓨터인 윈도우즈의 원격 데스크톱 연결 창에 다음 그림과 같이 라즈비안의 초기실행 화면이 나타나고, 이후부터 GUI 환경의 라즈베리파이를 사용할 수 있다.



2.3.6 DD for Windows

Raspbian OS, 패키지, 응용 프로그램 등이 Micro SD 카드에 적재되어 있기 때문에 잘못된 사용에 의한 파손이 일어나는 경우, 리눅스 시스템에서는 Linux 명령인 dd 명령을 이용하여 시스템을 준비할 수 있다. 하지만 Windows 시스템을 사용하는 경우, 유사기능의 Windows용 disk dup 프로그램인 DD for windows를 이용하여 SD 카드의 내용을 백업하였다가 복구하여 사용할 수 있다.

윈도우용 dd는 실행파일 형태로 압축이 되어있으므로 해당 압축파일을 원하는 곳에 복사하여 푼 다음, 실행파일인 DDWin.exe를 실행하면 된다. 이 실행파일에 대해 바탕화면에 단축아이콘을 만들어두면 편리하게 사용할 수 있다.

DD for windows를 실행하면 아래와 같은 화면이 나타난다. 실행시 DDWin.exe에서 마우스 우측버튼을 클릭하여 관리자권한으로 실행하여야 Micro SD 카드를 인식할 수 있으므로 유의한다. 단순 실행하는 경우 포맷해야 한다는 메시지가 나타난다.



SD 카드의 백업

라즈베리파이 보드상의 Micro SD 카드를 제거하고, 카드리더기에 삽입한 후, Windows 시스템에 연결한다. DD for Windows를 실행한 후, ‘Choose Disk’를 눌러서 연결된 카드리더기를 선택한다. 다음으로 ‘Choose File’을 눌러서 백업할 적당한 드라이브 및 폴더를 선택하고, 저장할 파일 이름을 입력한다. 백업 파일의 확장자는 .ddi이다. 끝으로 Micro SD 카드의 내용을 백업하기 위해 ‘Backup’ 버튼을 클릭한다.



SD 카드로 복원

새로 준비한 Micro SD 카드를 카드리더기에 삽입하고 Windows 시스템에 카드리더기를 연결한 후, DD for Windows를 실행한다. ‘Choose Disk’를 눌러서 연결된 카드리더기를 선택한 후, ‘Choose file’을 눌러 기존에 백업했던 ddi 파일을 선택한다. 이제 ‘Restore’ 버튼을 클릭하여 해당 백업 파일을 Micro SD 카드에 기록 한다.



2.4 리눅스 명령

라즈비안에서 기본적인 리눅스 명령어는 다음의 사이트의 내용을 참조하며, 보다 많은 명령들에 대해서는 도서 혹은 웹사이트를 참조하기 바란다.

<https://www.raspberrypi.org/documentation/linux/>

히스토리 기능과 자동완성 기능

히스토리 기능은 기존에 사용하였던 명령을 유지하였다가 다시 불러서 사용하는 기능으로, 자판의 상하키 이용하여 선택하여 실행할 수 있다. 또한 자동완성 기능은 디렉터리 이름이나 파일이름을 모두 타이핑할 필요없이 선두 문자를 입력하고 <TAB> 키를 누르면 자동 완성해주는 기능이다.

파이프 기능

파이프(pipe, |)기능은 여러 명령의 연결통로를 형성하여 한 프로세스의 표준출력을 다른 프로세스의 표준입력으로 전달하는 기능이다. 각 명령의 파이프 형성을 위해 |를 사용한다.

```
$ ls -al /usr/bin | more          // /usr/bin의 파일목록을 페이지단위 출력  
$ ls /usr/bin | sort | less       // 파일목록을 소팅하여 페이지단위 출력
```

절대 경로 및 상대 경로

경로지정은 절대경로와 상대경로로 지정하는 방법이 있다. 절대경로는 파일시스템의 뿌리인 / 디렉터리부터 경로를 지정하는 방법이며, 상대 경로는 현재 디렉터리

(.) 혹은 부모 디렉터리(..)부터 상대적인 위치를 지정하는 방법이다.

명령어 도움말

명령어의 세부 사용방법에 대한 도움말을 얻고자 할 때 man 명령을 사용한다. 다음은 ls 명령의 사용법에 대한 도움말을 보여준다. 응용 툴의 경우 툴명으로 도움말을 얻을 수 있다.

```
$ man ls           // 기본 리눅스 명령의 도움말
$ apt             // apt 툴의 도움말
```

2.4.1 파일시스템관련 명령

우선 파일 및 디렉터리의 속성에 대해 살펴본다.

파일 및 디렉터리 속성

파일 및 디렉터리의 속성은 ls -l 명령으로 확인할 수 있다.

```
pi@raspberrypi:~ $ ls -l
drwxr-xr-x  2 pi pi  4096 Nov 13 2018 Desktop
```

각 라인의 선두에 있는 밑줄 부분이 속성을 나타낸다. 첫 문자는 파일 유형을 나타내는 것으로, -는 일반파일을, b는 블록장치를, c는 문자장치를, d는 디렉터리를, l은 심볼릭 링크파일을, p는 파이프를, s는 소켓장치를 나타낸다. 이후의 3문자씩은 각각, user, group, others 계정의 사용자가 해당 파일 혹은 디렉터리의 접근권한 속성을 나타내는 것으로, r은 readable, w는 writable을, x는 executable을 의미하며, -표시되어 있으면 해당 접근권한을 금지한 것이다.

다음과 같이 pi 사용자가 touch 명령으로 빈 파일을 생성하면, 일반 사용자의 경우 기본 접근권한 속성으로 rw-r-r-로 설정된 것을 관찰할 수 있다. 또한 아래의 밑줄 부분은 파일을 생성한 사용자, 즉 owner와 해당 사용자가 속한 group을 나타낸다.

```
pi@raspberrypi:~ $ touch test
```

54 라즈베리파이기반 임베디드시스템용

```
pi@raspberrypi:~ $ ls -l test  
-rw-r--r-- 1 pi pi 0 Aug 30 14:10 test
```

파일시스템 관련 명령으로 디렉터리를 만들거나 삭제, 이동 및 파일을 생성, 복사, 삭제, 내용보기 등과 관련된 명령을 살펴본다. 이들 명령들은 옵션을 지정할 때 -를 앞세운다.

ls 명령

ls(list) 명령은 파일 혹은 디렉터리내의 목록을 나열하는 명령으로 다음과 같이 사용될 수 있다. 와일드카드문자로 ?, *로 특정 목록을 지정할 수 있다.

```
$ ls                      // 현재 디렉터리의 파일목록  
$ ls -a                   // 모든파일(히든파일포함)  
$ ls -l                   // 속성까지 상세하게  
$ ls -al                  // 모든 파일(히든파일포함)을 자세하게  
$ ls /usr/bin              // /usr/bin/ 디렉터리의 파일목록  
$ ls *.txt                // 특정파일만(파일명이 .txt로 끝나는 모든 파일)  
$ ls -l /usr/bin/a*
```

pwd 명령

pwd(present working directory) 명령은 현 작업디렉터리의 절대경로를 출력함으로써 현 작업디렉터리의 위치가 파일시스템의 어디에 위치하는지 알려준다.

```
pi@raspberrypi:~ $ pwd  
/home/pi
```

mkdir 명령

mkdir(make directory) 명령은 디렉터리를 생성하는 명령으로 다음과 같이 사용될 수 있다.

```
$ mkdir mydir  
$ mkdir -p mydir/subdir/tmp          // parents, 서브디렉토리까지 생성
```

cd 명령

cd(change directory) 명령은 특정 디렉터리로 진입하는 이동 명령으로 다음과 같이 사용될 수 있다.

```
$ cd                      // 사용자의 홈디렉터리로 이동
$ cd ..                  // ..은 현 디렉터리의 부모 디렉터리를 의미
$ cd mydir               // 현 디렉터리내의 서브 디렉터리 mydir로
                         // $ cd ./mydir과 동일
$ cd /usr/bin             // /usr/bin로 이동(절대경로)
$ cd ../../usr/bin        // 홈 디렉터리에서 위 명령과 동일 효과(상대경로)
```

rmdir 명령

rmdir(remove directory) 명령은 디렉터리를 대상으로 삭제하는 명령으로, 대상 디렉터리 내에는 어떠한 파일도 없어야 한다.

```
$ rmdir mydir
$ rmdir -p mydir/subdir // 여러 계층의 디렉터리 삭제, 비어있어야 함
```

일반적으로 비어있지 않아도 강제 삭제하는 옵션을 준 rm 명령을 사용하기도 한다.

touch 명령

touch 명령은 내용이 없는 파일을 생성하는 명령이나, 일반적으로 기존 생성된 파일의 생성일시를 통일하는데 사용된다.

```
$ touch test1 test2      // 빈 파일들을 생성
$ touch *                 // 모든 파일의 생성일시를 현재일시로 변경
```

nano 명령

파일을 생성하기 위한 편집기를 실행하는 명령으로 라즈비안에서 제공되는 단순한 형태의 파일 편집기다. 가상머신에서는 이 편집기외에 gedit 편집기 등을 사용할 수 있다.

```
$ nano test.c
```

cat 명령

56 리즈베리파이기반 임베디드시스템용

cat 명령은 텍스트 파일의 내용을 화면에 출력하는 명령이다.

```
$ cat test.c // 파일내용 보기
```

more/less 명령

이들 명령은 텍스트 파일을 화면 단위로 출력하는 명령으로, 출력된 후 다음 화면을 보려면 <space> 키를, 이전 화면을 보려면 키를 누른다.

```
$ more test // more 명령과 동일기능  
$ less test
```

head 명령

이 명령은 파일의 머리, 즉 첫 라인부터 지정한 라인 수만큼 출력하는 명령으로 기본 설정 라인수는 10이다.

```
$ head test // 기본 10라인 보기  
$ head -20 test // 20라인 보기
```

tail 명령

이 명령은 파일의 끝 라인부터 지정한 라인 수만큼 출력하는 명령으로 기본 설정 라인수는 10이다.

```
$ tail test // 기본 10라인 보기  
$ tail -2 test // 끝의 2라인보기
```

rm 명령

rm(remove)은 기본 적으로 파일을 삭제하는 명령이나 옵션을 활용하여 비어있지 않은 디렉터리까지 삭제할 수 있는 명령이다. 이들 옵션에는 i(interactive), r(recursive), f(force) 등이 있다.

```
$ rm test // test 파일 삭제  
$ rm -i test // 삭제 확인 질의후  
$ rm -r mydir // 디렉터리 삭제시  
$ rm -rf mydir // mydir와 그 하부를 삭제
```

cp 명령

cp(copy) 명령은 파일이나 디렉터리를 복사하는 명령으로 다음과 같이 사용될 수 있다.

```
$ cp test testcopy          // 파일 복사
$ cp -r mydir yourdir      // 디렉터리 복사
$ cp -a /home/sample/tests . // 파일의 속성 유지한 채 복사, archive
```

mv 명령

mv(move) 명령은 cp 명령과 달리 다른 곳으로 이동 혹은 이름을 재명명할 때 사용하는 명령이다.

```
$ mv aaa bbb ./mydir        // ./mydir은 디렉터리로 두파일 이동
$ mv /etc/* ./mydir         // /etc/하부 모든파일을 ./mydir로 이동
$ mv test mytest            // test를 mytest로 변경, 파일혹은 디렉터리
$ mv mydir yourdir          // mydir 디렉터리명을 yourdir로 변경
```

chmod 명령

chmod(change mode) 명령은 파일 혹은 디렉터리의 접근권한 속성을 변경하는 명령이다.

```
$ chmod go-w filedir        // filedir에 group, others의 기록권한 삭제
$ chmod a+x filedir          // filedir에 모두에게 실행권한 설정
$ chmod 755 filedir          // filedir에 octal 표현으로 -rwxr-xr-x 설정
$ chmod -R 777 ./filedir       // 하부 디렉터리까지 설정
```

chown/chgrp 명령

이들 명령은 파일 혹은 디렉터리에 대한 소유권 및 그룹을 설정하거나 변경할 때 사용하는 명령이다.

```
$ chown root /home/pi/test      // owner를 root로
$ chgrp root /home/pi/test       // group을 root로
$ chown root.root /home/pi/test   // 소유자 및 그룹 함께 변경
```

58 리즈베리파이기반 임베디드시스템용

ln 명령

ln(link) 명령은 Windows에서 바로가기 기능과 유사한 기능을 하는 명령으로 복잡한 파일명, 긴 경로명을 간단히 취급할 때 유용한 명령으로 하드(hard) 링크와 심볼릭(symbolic) 링크 형태가 있다.

하드 링크는 다음과 같이 사용할 수 있으며, 어느 파일을 수정하든 두 파일은 동일 내용을 가지나 각기 독자적인 저장공간에 위치한다.

```
$ ln aaa.txt atext          // 하드링크
```

심볼릭 링크는 -s 옵션을 사용하여 다음과 같이 사용할 수 있으며, ls 명령으로 보면 이 심볼릭링크 파일은 "링크파일->원본파일"로 표시된다. 윈도우즈에서 바로가기기능과 흡사하다 할 수 있다.

```
$ ln -s ./mydir/subdir/test ltest
$ cat ltest                  // 다른 경로에 있는 파일을 쉽게 접근
```

```
$ ln -s /mnt/share/ share   // 디렉터리 지정
$ ls share                   // 파일목록 보기
$ cd share                   // /mnt/share로 이동
```

unzip 명령

압축된 파일을 풀 때 사용하는 명령으로 다음과 같이 사용된다.

```
$ unzip tools.zip
```

tar 명령

tar(tape archive) 명령은 여러 개의 파일 혹은 디렉터리를 하나의 파일로 묶거나 풀어낼 때 사용하는 명령으로, 통상 .tar 파일을 취급한다. 옵션 표기시 -를 접두하지 않고, tar 파일 생성시 cvf를, 풀때는 xvf를 사용한다. 묶기와 gzip 압축을 병행할 때는 z 옵션을 추가한다.

```
$ tar cvf my.tar ./my        // ./my의 모든 파일을 my.tar파일로 묶음
```

```
$ tar tvf my.tar           // 풀지않고 포함 목록만 확인
$ tar xvf my.tar          // 풀기
$ tar cvfz my.tar.gz ./my // 뮤기와 gzip 동시에, .gz 확장자
$ tar xvzf my.tar.gz      // 압축복원과 풀기 동시에
```

2.4.2 검색 명령

grep 명령

이 명령은 표준입력으로부터 자료를 필터링한 후 출력하는 명령이다.

```
$ grep hello abc.txt        // 파일내에 hello포함 라인 검색
$ ls -al | grep test        // 파일목록중에 test 포함된 것 출력
```

whereis 명령

현 탐색경로가 아닌 표준 프로그램 위치에서 검색하여 경로명 포함 출력하는 명령이다.

```
pi@raspberrypi:~ $ whereis ifconfig
ifconfig: /sbin/ifconfig /usr/share/man/man8/ifconfig.8.gz
```

find 명령

find 명령은 탐색경로명 및 패턴으로 특정 파일의 위치를 검색하는 명령으로 다음과 같이 사용할 수 있다. 찾을 패턴을 지정하기 위해 -name 옵션을 사용한다.

```
pi@raspberrypi:~ $ find /usr -name wiringPi*
/usr/include/wiringPi2C.h
/usr/include/wiringPi.h
/usr/include/wiringPiSPI.h
```

2.4.3 네트워크 및 기타 관련 명령

60 라즈베리파이 기반 임베디드 시스템 활용

ifconfig 명령

이 명령은 네트워크 장치에 설정된 IP 주소들의 설정 정보를 확인하거나 변경하는 명령이다.

```
$ ifconfig // 현재 NIC의 설정 정보 보기  
$ ifconfig eth0 172.18.14.xx // IP 주소 변경
```

ping 명령

이 명령은 해당 시스템까지 패킷을 송수신하여 네트워킹 상태를 파악하는 명령이다.

```
$ ping 192.168.0.40
```

hostname 명령

이 명령은 호스트명을 출력한다.

```
$ hostname  
raspberrypi
```

uname 명령

uname 명령은 프로세서, 커널 버전 등 운영체제에 대한 정보를 확인할 때 사용하는 명령이다.

```
$ uname -a // 시스템의 모든 정보 보기  
$ uname -p // 프로세서 확인  
$ uname -r // 커널 버전 확인
```

clear 명령

출력 화면을 초기화하는 명령이다.

```
$ clear
```

참고자료

[1] 라즈비안 다운로드 사이트

<https://www.raspberrypi.org/downloads/raspbian/>

[2] SD 카드 기록 툴 Etcher 다운로드 사이트

<https://etcher.io/>

[3] SD카드 기록 툴 win32diskimager 다운로드 사이트

<https://sourceforge.net/projects/win32diskimager/>

[4] iptime 설치도우미 다운로드 사이트

http://iptime.com/iptime/?page_id=67&uid=14729&mod=document

[5] 와이파이 연결 <http://maker1st.tistory.com/2>

[6] 고정 IP주소 설정 <http://erider.co.kr/142>

[7] PuTTY 다운로드 사이트

<https://www.chiark.greenend.org.uk/~sgtatham/putty/>

[8] 원격 데스크톱 연결 <http://maker1st.tistory.com/3>

[9] 파일질라 클라이언트 <http://chokyuhwan.tistory.com/28>

[10] FileZilla 다운로드 사이트 <https://filezilla-project.org/>

[11] 삼바서버 구축 <http://withcoding.com/48>

[12] “라즈베리파이를 이용한 Embedded Linux 설계 및 응용”, (주)휴인스, 2018

제3장 개발 환경 구축 II

라즈비안 커널을 컴파일하는 경우 라즈베리보드에서 컴파일 시간이 수십 시간에 달한다. 반면 Windows의 가상머신에서는 수시간 소요된다. 따라서 컴파일시간을 단축하기 위해서는 Windows 환경에서 개발하는 것이 바람직하다. Windows 환경에서 VMware player 설치, 가상머신 설치, 크로스 컴파일러 설치 등의 과정에 대해 살펴본다. 또한 가상머신 상이든 라즈베리파이 보드에서든 한글 표시 및 입력을 가능도록 하는 과정에 대해서도 살펴본다.

본 장에서 언급하는 내용들은 개발 환경이 라즈베리파이 보드 상에서만 이루어지는 경우 생략가능하다.

3.1 VMware Player

VMware Player는 서로 다른 운영체제를 특정 시스템에서 사용 가능하도록 하는 툴이다. 이를 위해서는 VMware Player 내에 다른 운영체제인 가상머신을 설치하여 사용하게 된다.

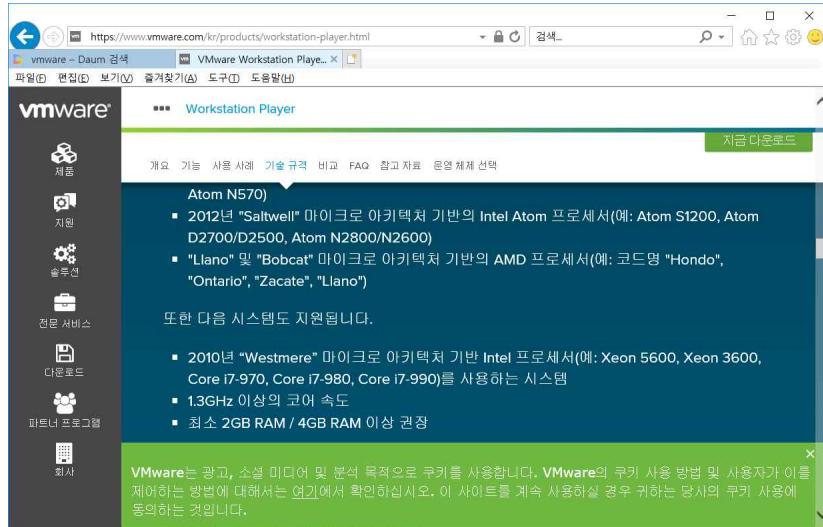
VMware Player 다운로드

VMware Player를 설치할 Windows가 32비트인지 64비트 운영체제인지를 확인할 필요가 있고, 그에 따른 버전의 VMware Player를 다운로드한다. VMware Player 다운로드 사이트는 다음과 같다.

<https://www.vmware.com/kr/products/workstation-player.html>

해당 사이트에 접속하면 다음의 화면이 나타난다.

64 라즈베리파이기반 임베디드시스템용



혹은, 바탕화면 [IFC415] 폴더내에 64비트용으로 다운로드 받은 것을 사용하여도 된다.

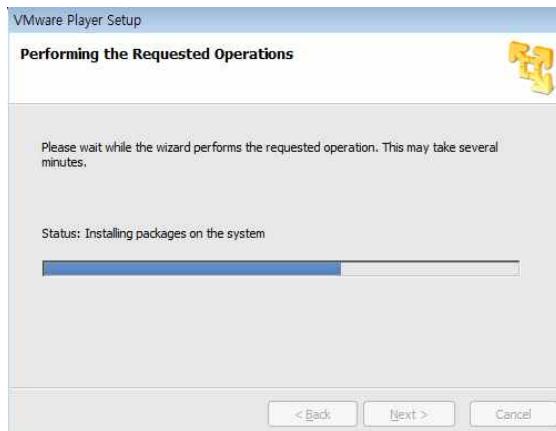
VMware-player-14.1.3-9474260.exe

VMware Player 설치

다운로드 받은 VMware Player의 실행파일을 클릭하여 설치를 시작하며 다음과 같은 초기화면이 나타나고 next 버튼을 클릭 한다.



다음과 같은 화면의 패키지 설치과정이 나타난다.



끝으로, 다음과 같은 라이센서 동의 화면이 나타나면, I accept 항목을 체크하고 next 버튼을 클릭한다.



설치 과정중에 추가의 메시지 창이 나타날 수 있으며, 적절히 선택하여 진행하면 된다. 설치가 완료되면 바탕화면에 VMware player 아이콘이 생성된다.

3.2 VM(Ubuntu) 설치

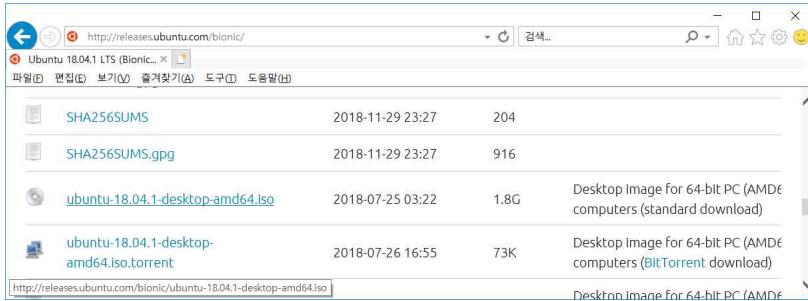
가상머신(virtual machine)으로 리눅스 계열의 Ubuntu를 사용한다. Ubuntu 설치 과정에 대해 살펴본다.

3.2.1 Ubuntu 이미지 다운로드

가상머신으로 사용할 Ubuntu 이미지를 다음 사이트에 접속하여 다운로드한다.

<http://releases.ubuntu.com/bionic/>

위의 사이트에 접속하면 다음과 같은 화면이 나타난다.

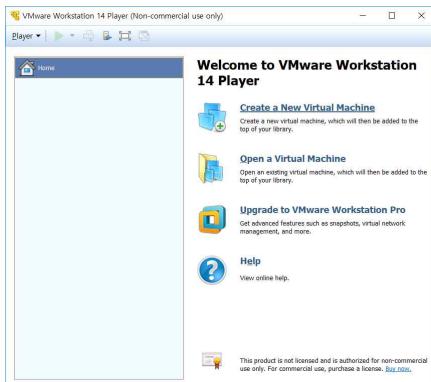


가능하면 ubuntu-18.04 이후의 버전을 다운로드 받길 권장한다. 실습을 위해 다운로드 받은 버전은 Ubuntu 18.04.1 LTS이며, 이미지 파일명은 다음과 같다.

ubuntu-18.04.1-desktop-amd64.iso

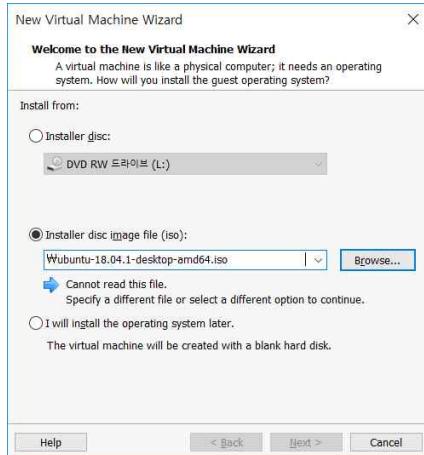
3.2.2 가상머신 설치

VMware Player 실행하면 다음과 같은 초기 화면이 나타난다.

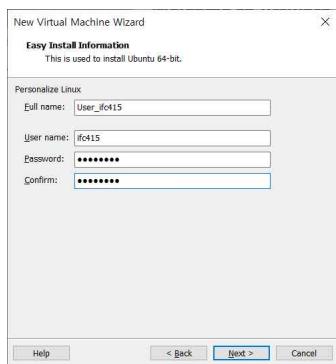


초기화면에서 Create a New VM 항목을 클릭하면, 다음과 같은 화면이 나타난다. 이 화면에서 Installer disk image file 항목을 체크하고, browse 버튼을 눌러 다운로드 받은 이미지파일을 지정한 후, next 버튼을 누른다.

68 리즈베리파이기반 임베디드시스템용



다음과 같은 Easy install Information 화면이 나타난다. 이 화면에서 가상머신의 호스트명, 기본 계정명과 암호를 지정하고 next 버튼을 클릭한다.



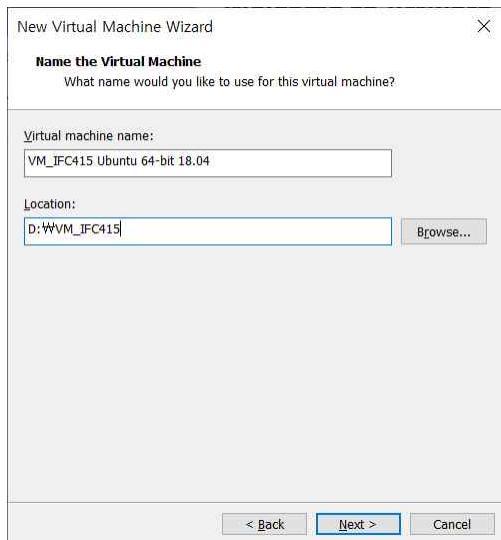
실습을 위해 각 항목은 다음과 같이 통일하도록 한다.

Full name : User_ifc415
User name : ifc415
Password : embedded
Confirm : embedded

// user full name
// 가상머신 기본 계정명

다음과 같은 가상머신의 이름 및 위치 지정 화면이 나타난다. 가상머신의 이름은 VMware player의 초기화면의 좌측창에 나타나는 이름이며, Location은 가상머신

의 이미지가 설치될 경로를 나타내는 것으로 관리의 편의를 위해 적절히 지정하고, next 버튼을 클릭한다.



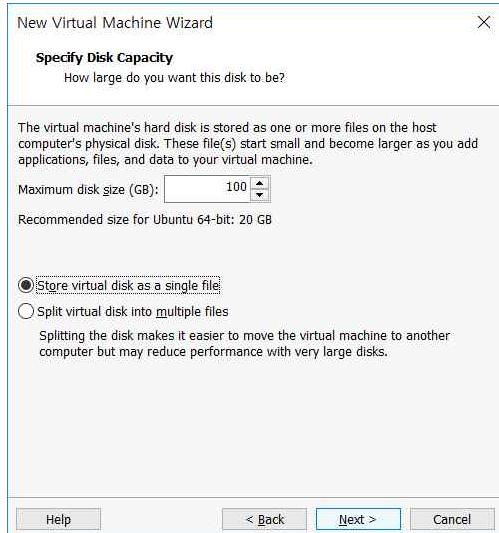
실습을 위해 다음과 같이 통일할 것을 권한다.

VM name : VM_IFC415 Ubuntu 64-bit 18.04 // 라이브러리 이름

Location : D:\VM_IFC415 // 필히 D: 드라이브에

다음의 디스크 용량 지정 화면이 나타나면 가상머신이 사용할 수 있는 하드디스크의 최대공간을 지정하고, NTFS 형식의 파일시스템의 경우 가상머신을 하나의 파일로 저장 가능하므로 이 항목을 선택할 수 있다. 설정이 완료되었으면 next 버튼을 클릭한다.

70 라즈베리파이기반 임베디드시스템용

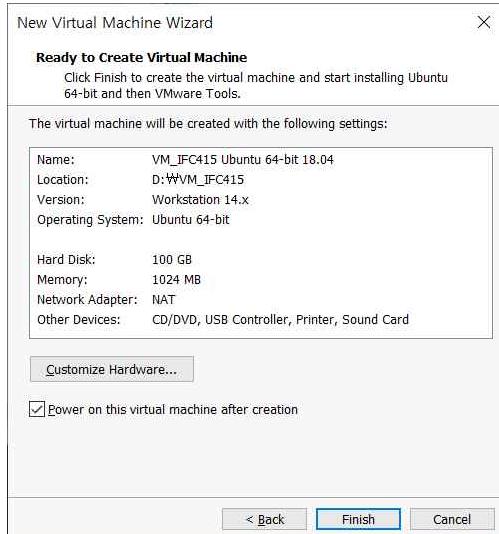


최대 디스크 사이즈 등은 다음과 같이 설정할 것을 권고한다.

Maximum disk size : 100G

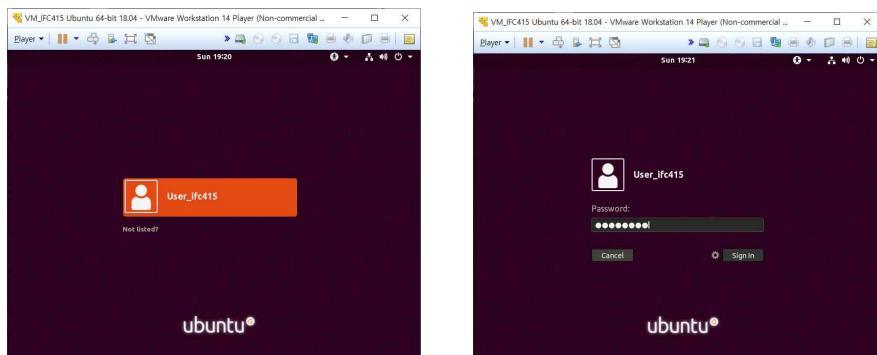
Store VM as a single file 체크

다음과 같은 가상머신의 기본 설정정보 보기 화면이 나타는데, 지금까지 설정한 가상머신의 요약 정보를 보여준다. Customize H/W 버튼은 가상머신을 위한 프로세서 코어수, 메모리, 랜카드, 시리얼포트 등의 하드웨어 관련 설정을 할 때 사용할 수 있는데, 이와 관련된 내용은 추후 살펴보도록 한다.

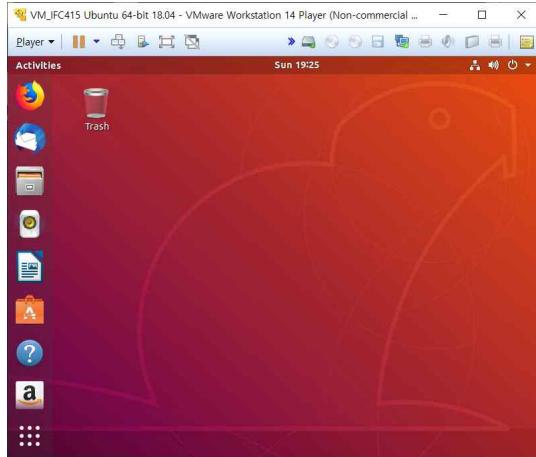


Finish 버튼을 클릭하면 가상머신의 설치를 개시하며, 약 20여분 소요되어 설치완료된 후, 가상머신으로 부팅까지 진행한다.

가상머신의 로그인 화면이 다음과 같이 나타나면 User_ifc415 항목을 클릭한다. 로그인을 위해 기본 계정인 ifc415 계정의 암호를 요구하는데, 설정한 암호를 입력하면 로그인된다.



정상적으로 로그인 되면 다음과 같은 가상머신의 초기 화면이 나타난다. 이제 Windows 시스템에서 VMware Player를 통해 리눅스 시스템을 사용할 수 있는 가상머신을 구축하게 된 것이다.

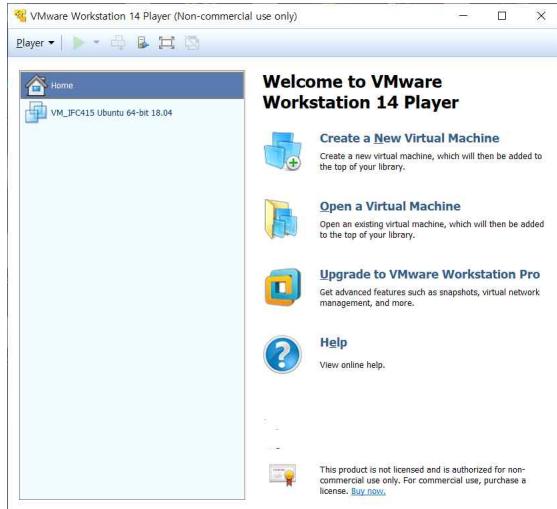


가상머신의 종료를 위해 로그아웃을 하려면, VMware Player의 Player - exit 항목을 클릭한 후, power off를 선택한다.

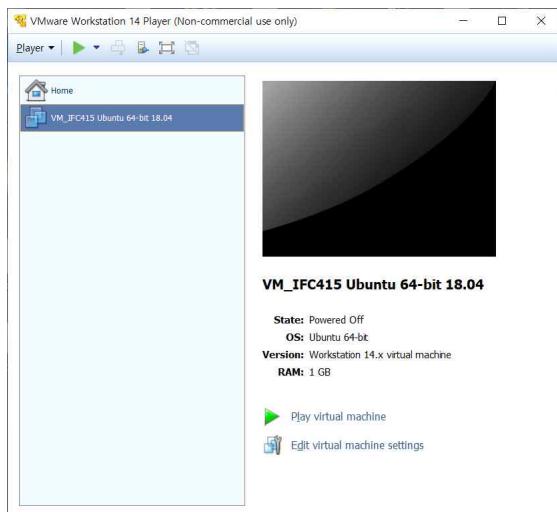
3.2.3 가상머신 환경 설정

앞에서 살펴본 가상머신 설치과정 중 끝 부분에서 언급된 설정된 요약정보 보기 화면의 Customize H/W 버튼을 눌러 설정할 수 있는 것에 대한 내용과 동일한 내용을 다룬다.

우선 VMware Player 재실행하면 다음과 같은 초기 화면이 나타난다. 화면의 좌측 창에서 설치된 가상머신의 이름의 항목을 확인할 수 있다. 이와 같이 서로 다른 운영체제를 VMware player에 생성하여 다양한 운영체제를 다루어 볼 수 있다.



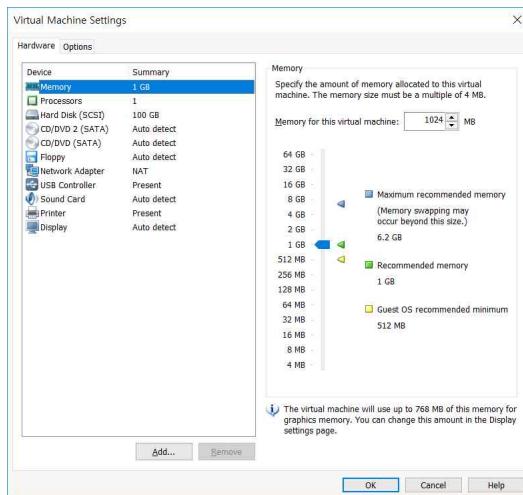
위의 화면에서 좌측 창에서 사용할 가상머신 VM_IFC415 Ubuntu 64-bit... 를 선택한다. 선택하면 다음과 같은 화면이 나타난다.



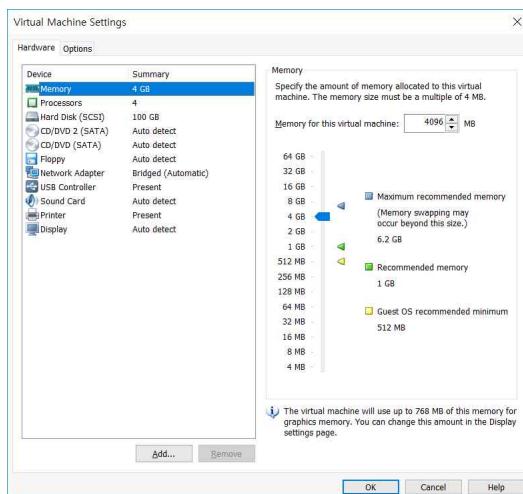
우측 창의 Play virtual machine 항목을 클릭하면 가상머신으로 부팅하는 것이다. 가상머신에 대한 하드웨어 설정을 해야 하므로 Edit virtual machine settings 항목을 클릭한다.

7.4 라즈베리파이기반 임베디드시스템용

다음과 같은 화면이 나타나고, Hardware 탭은 가상머신의 하드웨어 환경설정을 위한 탭이다.

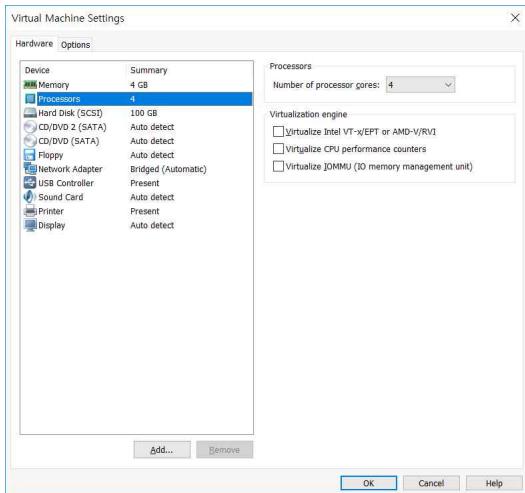


Memory 항목에서는 가상머신을 위해 사용할 메모리 공간을 지정하는 것으로, Windows 시스템에 설치된 전체 메모리 사이즈의 1/2정도로 지정한다. 시스템에 설치된 메모리 사이즈가 8GB라면 4GB로 설정한다.



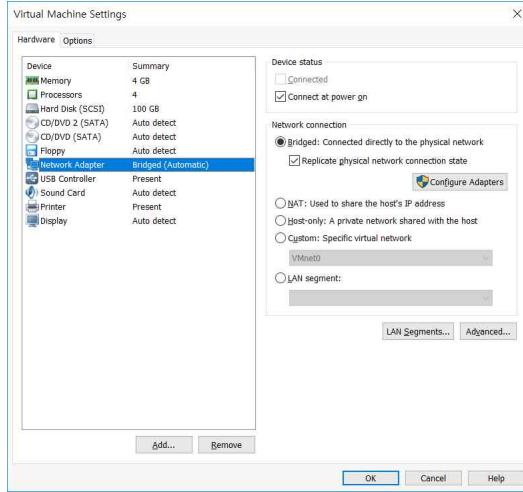
Processors 항목에서는 시스템 CPU의 코어수를 설정한다. CPU의 코어수는

Windows의 실행창에서 msinfo32 명령을 사용하여 확인할 수 있다. 코어수가 4개라면 4로 설정한다.



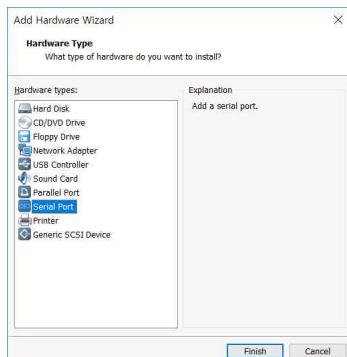
다음으로 Network Adapter 항목을 선택하여 적절히 설정한다. 우측 창의 Network connection 부분을 자신의 개발환경에 따라 설정한다. 가상머신에서 Windows 환경의 IP 주소와 동일한 IP 주소를 사용할 경우는 NAT 항목을 선택한다. 즉, Windows에 설정된 IP 주소와 동일한 IP 주소로 가상머신에서 웹서핑 등을 하겠다는 의미이다. 반면, 가상머신에서 Windows 환경의 IP 주소와 다른 IP 주소를 사용할 경우는 Bridged 항목을 선택하고, Replicate항목도 체크한다. 임베디드 시스템 개발 환경과 같은 개발을 용도인 경우 각기 다른 IP 주소를 사용할 수 있도록 후자의 방법을 취하는 것이 바람직하다.

76 라즈베리파이기반 임베디드시스템용

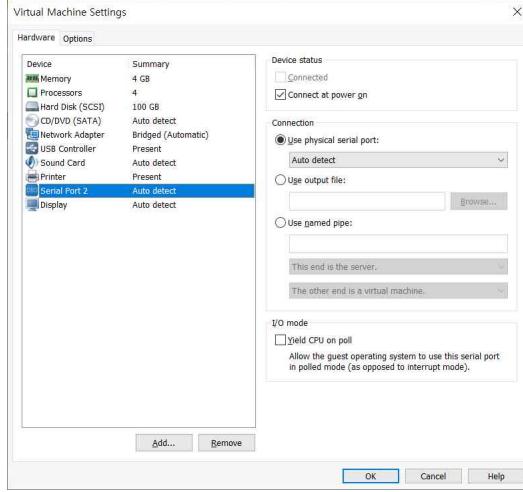


따라서 실습을 위해 Bridged 항목을 선택하고, Replicate.... 항목을 체크하도록 한다.

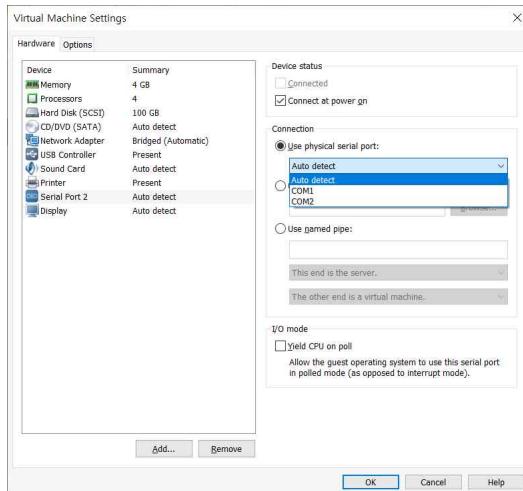
좌측 창의 하드웨어 목록에 없는 경우는 좌측 창 하단의 Add.. 버튼을 클릭하고 다음과 같은 창이 나타나면 추가하여 구성할 수 있다. 예로 시리얼 포트를 추가하기 위해 Add.. 버튼을 클릭하면 다음과 같은 하드웨어 목록이 나타나고 Serial port를 선택하고 Finish 버튼을 누르면 된다.



시리얼 포트의 하드웨어가 추가되면 다음과 같이 Hardware 탭에 Serial port 항목이 추가된다.



시리얼 포트의 설정은 다음 화면과 같이 Use physical serial port 항목을 체크한다. 시스템에 물리적으로 하나의 시리얼 포트가 제공되는 경우, Windows 환경에서 시리얼 포트명이 COM1이면, 가상머신에서는 serial port 2를 의미하는 /dev/ttyS1을 디바이스 명으로 사용해야 함을 유의한다.

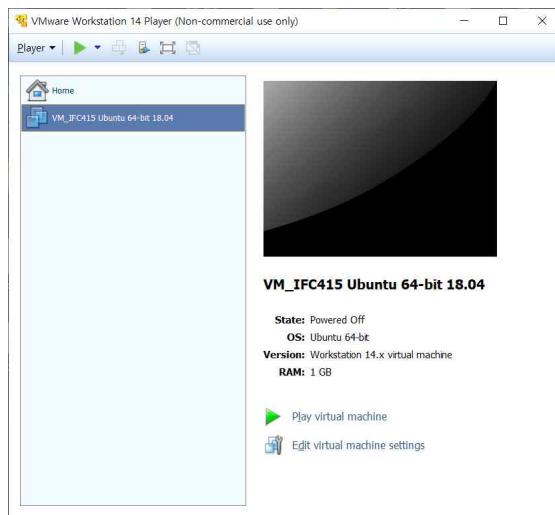


이로써 가상머신을 위한 필요한 하드웨어 환경설정이 완료된다.

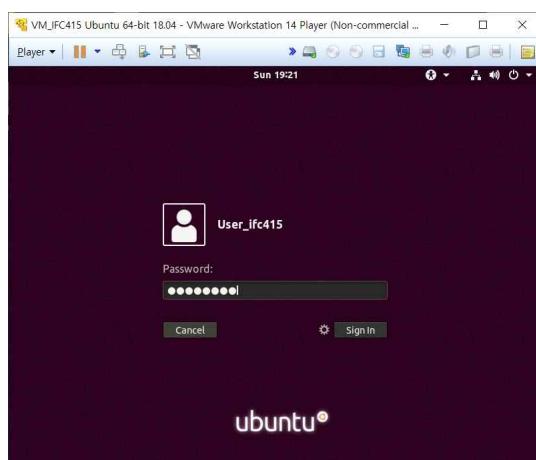
78 리즈베리파이기반 임베디드시스템용

3.2.4 가상머신 활용

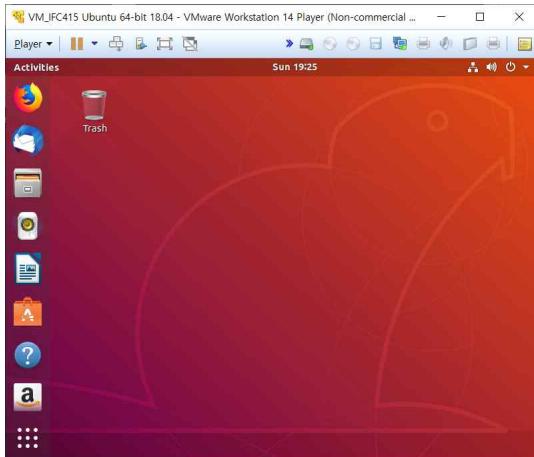
VMware Player 아이콘을 클릭하여 실행한 후, 다음과 같은 화면의 좌측 창에서 가상머신 이름을 선택 후, 우측 창에서 Play virtual machine 항목을 클릭한다.



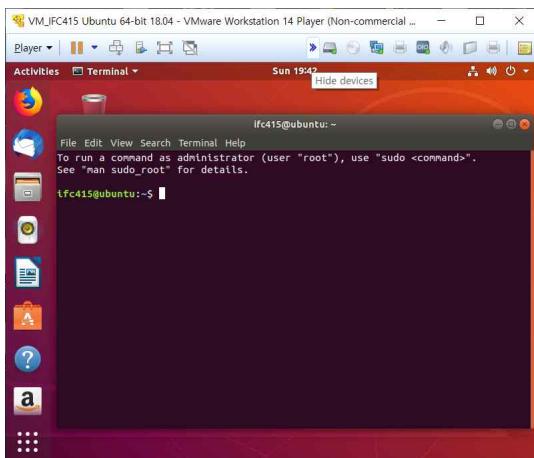
다음과 같은 로그인 화면에서 ifc415 계정의 패스워드를 입력하여 로그인한다.



가상머신의 초기 실행화면은 다음과 같다.



리눅스 명령을 사용하여 어떠한 작업을 하고자 할 때 Windows 환경에서의 명령 프롬프트 창과 유사한 터미널 창을 연다. 터미널 창은 바탕화면에서 마우스를 우클릭하여 open terminal 항목을 선택하거나, 단축키로 Ctrl-Alt-T를 입력하면 된다. 터미널 창은 열면 다음과 같은 화면이 나타난다.



이제 이 터미널 창에서 Linux 명령어를 사용하여 일처리를 진행할 수 있다.

슈퍼유저 계정과 일반유저 계정간 전환

80 리눅스 명령어 기반 임베디드 시스템 활용

개발 과정에서 슈퍼유저로 명령을 실행해야 하는 것이 편리할 수도 있다. 이 경우 슈퍼유저로 전환하기 위해 다음의 절차를 진행할 수 있다.

우선 ifc415 계정이 슈퍼유저로 전환할 때 슈퍼유저의 암호로 사용될 암호를 설정한다. 이때 다음의 명령을 사용한다. 편의를 위해 ifc415 유저와 동일한 암호(embedded)로 설정하도록 한다.

```
ifc415@ubuntu:~$ sudo passwd root  
[sudo] password for ifc415: embedded  
Enter new UNIX password: embedded  
Retype new UNIX password: embedded
```

이제 ifc415 계정이 슈퍼유저로 최초 전환할 때는 다음의 명령을 사용하고, 암호를 입력한다.

```
ifc415@ubuntu:~$ sudo su  
[sudo] password for ifc415: embedded
```

슈퍼유저로 전환되면 명령행의 프롬프트가 \$에서 #으로 다음과 같이 변경되어 나타난다. 물론 계정명도 슈퍼유저의 계정명인 root로 변경된다.

```
root@ubuntu:/home/ifc415# // su 로그인 상태
```

슈퍼유저의 홈 디렉터리로 이동하기 위해서는 다음의 명령을 사용하면 된다.

```
root@ubuntu:/home/ifc415# cd
```

아울러 현재의 디렉터리 경로를 알고 싶으면 다음의 명령을 사용하여 확인할 수 있다.

```
root@ubuntu:~# pwd  
/root/
```

앞 장에서 리눅스 명령을 사용할 때 슈퍼유저 권한으로 실행하기 위해 각 명령 앞

에 sudo를 사용한 것과 달리, 슈퍼유저로 전환한 현재의 경우 리눅스의 명령 앞에 sudo를 붙이지 않아도 슈퍼유저 권한으로 실행된다.

슈퍼유저가 다음의 exit 명령을 사용하여 로그아웃하게 되면 일반 유저 ifc415 계정의 로그인 상태로 전환된다.

```
root@ubuntu:~# exit          // 슈퍼유저 logout
ifc415@ubuntu:~$           // 일반유저 ifc415 로그인 상태
```

이후부터 다음과 같이 슈퍼유저로의 전환시에는 암호를 묻지 않고 슈퍼유저로 전환된다.

```
ifc415@ubuntu:~$ sudo su
root@ubuntu:/home/ifc415#          // su 로그인 상태
root@ubuntu:/home/ifc415# cd        // su의 홈 이동
```

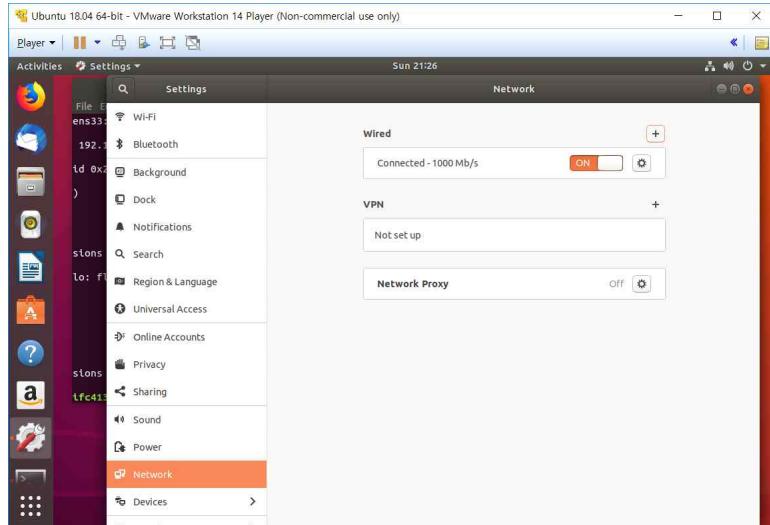
3.2.5 가상머신의 IP 주소 설정

Windows 환경의 IP 주소와 다른 IP 주소를 가상머신에서 사용할 것이므로, VMware Player의 network adapter 설정에서 다음과 같이 설정되어 있어야 한다. 가상 머신에 설정할 IP 주소는 192.168.0.20라고 가정한다.

Bridged 선택
 Replicate...로 체크

가상머신의 초기화면의 좌측에 나열된 아이콘중 Show Application 아이콘을 클릭하여 Settings - Network 까지 이동한다. 다음과 같은 화면이 나타나면, Wired 항의  항을 클릭한다.

82 리즈베리파이기반 임베디드시스템용



Wired 항의 항을 클릭하여 다음의 화면이 나타나면 IPv4 탭은 선택한다. IPv4 Method 항목에서 manual을 선택하고, Address 항목과 DNS 항목에 아래 화면과 같이 IP 주소 정보를 설정하고 Apply 버튼을 클릭한다. DNS는 168.126.63.1로 설정한다.



재부팅한 후, Details 탭을 선택하면 다음 화면과 같이 설정된 정보들을 확인할 수 있다.

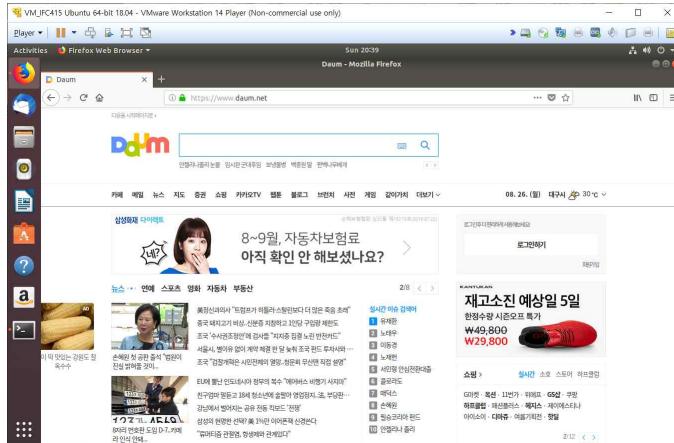


혹은, 가상머신 설정된 IP 주소를 확인하기 위해 터미널 창에서 ifconfig 명령을 실행하면 된다. 혹시 ifconfig 명령어가 없다는 메시지가 나오면 이를 위한 패키지 net-tools를 먼저 설치한 후 ifconfig 명령을 실행하면 된다. 가상머신에서 랜카드의 디바이스명이 ens33임을 유의한다.

```
root@ubuntu:~# apt install net-tools // 미설치시
root@ubuntu:~# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.0.20 netmask 255.255.255.0 broadcast 192.168.0.255
            inet6 fe80::2c4:b745:7608:3d0a prefixlen 64 scopeid 0x20<link>
              ether 00:0c:29:af:fc:38 tx.....
```

가상머신의 네트워킹 여부는 ping 명령을 사용하거나, 다음과 같이 웹 브라우저 통해 특정 사이트에 접속하는 것으로 확인 가능하다.

84 라즈베리파이기반 임베디드시스템용



3.3 VMware Tools

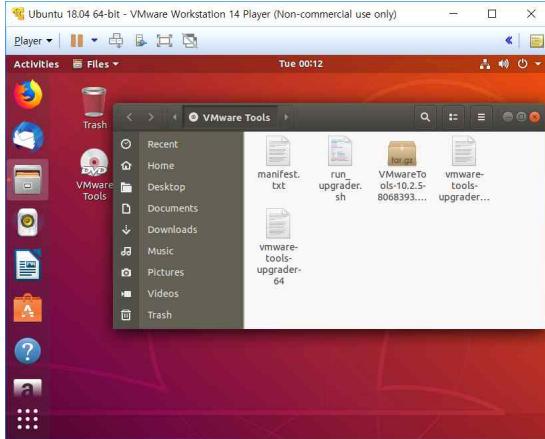
가상머신에 VMware Tools을 설치하면 많은 편의기능을 확보할 수 있어, 개발시 큰 도움이 되므로 그 설치과정에 대해서 살펴본다.

VMware Tools의 편의 기능은 다음과 같다.

- 마우스 이동으로 제어권 전환(OS 전환시 Ctrl-Alt 비사용)
- 디스플레이 옵션 확장(Windows환경의 디스플레이와 동일)
- 드래그앤파울 파일 이동(작은 크기의 파일 권고)
- 클립보드 공유
- OS간 공유폴더 사용

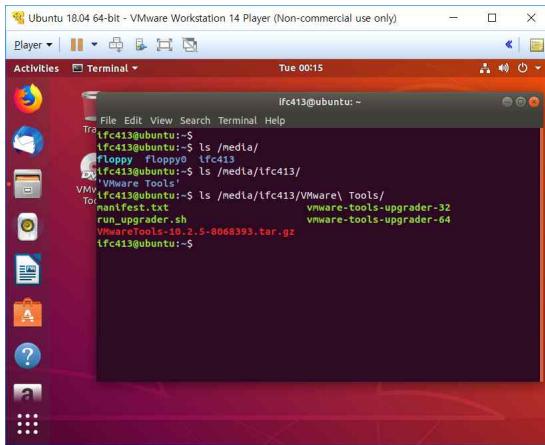
3.3.1 VMware Tools 다운로드

가상머신에 일반 유저 ifc415 계정으로 로그인한 상태에서, VMware Player의 Player > Manage > (Re)Install VMware Tools 클릭하면 VMware Tools이 설치된 후, 다음과 같은 화면이 나타난다.



다운로드된 VMware Tools은 가상머신 파일시스템의 /media/ifc415/VMware Tools/ 경로에 위치한다. 이 경로중의 ifc415는 로그인 계정의 ID를 나타낸다. 다음의 명령을 사용하여 파일들을 확인할 수 있다.

```
$ ls /media/ifc415/VMware\ Tools/
```



파일 목록중 VMwareTools-10.2.5-8068393.tar.gz 파일이 설치할 VMware Tools이다.

3.3.2 VMware Tools 설치

다음의 명령들을 사용하여 슈퍼유저로 전환한 후, VMware Tools의 tar 파일을 슈퍼유저의 홈 디렉터리로 복사한다.

```
ifc415@ubuntu:~$ sudo su
[sudo] password for ifc415: embedded
root@ubuntu:/home/ifc415# cd
root@ubuntu:~# ls
root@ubuntu:~# ls /media/ifc415/VMware\ Tools/
manifest.txt
VMwareTools-10.2.5-8068393.tar.gz vmware-tools-upgrader-64
run_upgrader.sh vmware-tools-upgrader-32

root@ubuntu:~# cp
/media/ifc415/VMware\ Tools/VMwareTools-10.2.5-8068393.tar.gz ./
root@ubuntu:~# ls
VMwareTools-10.2.5-8068393.tar.gz
```

다음의 명령으로 tar 파일을 풀면, vmware-tools-distrib 디렉터리가 보인다.

```
root@ubuntu:~# tar xvfz VMwareTools-10.2.5-8068393.tar.gz
root@ubuntu:~# ls
VMwareTools-10.2.5-8068393.tar.gz vmware-tools-distrib
```

다음의 명령들로 vmware-tools-distrib 디렉터리로 이동한 후, 설치를 진행한다.

```
root@ubuntu:~# cd vmware-tools-distrib/
root@ubuntu:~/vmware-tools-distrib# ls
bin    caf    doc    etc    FILES    INSTALL    installer    lib    vgauth
vmware-install.pl
root@ubuntu:~/vmware-tools-distrib# ./vmware-install.pl
```

설치하는 과정동안 yes/no의 답을 경우 y, 경로 묻는 경우 enter 키 입력하여

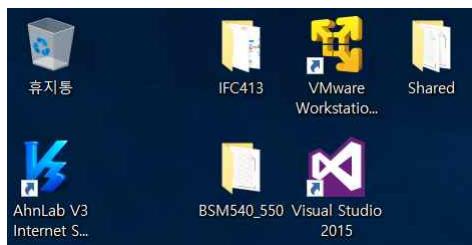
default 설정된 경로를 사용하도록 답한다. 이로서 VMware Tools이 설치가 완료되었으며, 다음의 명령으로 재부팅한다.

```
root@ubuntu:~/vmware-tools-distrib# reboot
```

3.3.3 공유 폴더 활용

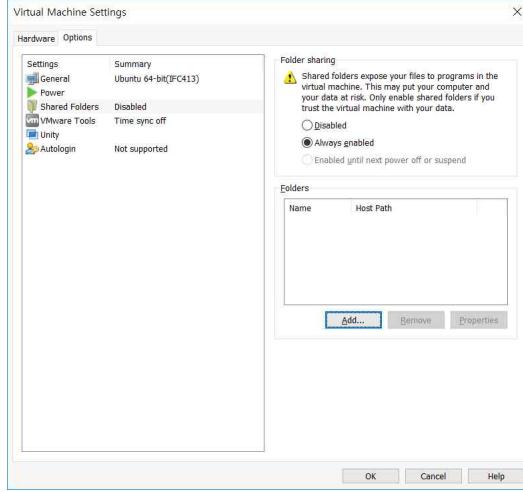
VMware Tools의 여러 기능 중에 공유폴더 기능을 사용하면, Windows와 가상머신간의 자료 공유가 가능하다. 공유 폴더를 사용하는 과정에 대해 살펴본다.

우선, Windows의 바탕화면에 가상머신과 공유할 폴더를 생성한다. 편의를 위해 폴더명은 Shared라고 가정하며, 다음 그림은 공유폴더가 생성된 후의 바탕화면의 일부를 보여준다.

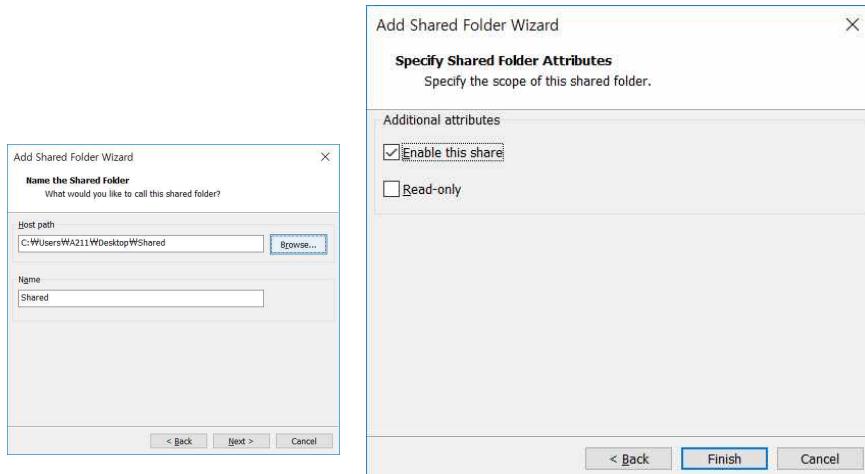


가상머신에서 Windows 환경의 공유폴더를 사용하기 위해서는 앞에서 가상머신의 하드웨어 설정에서 살펴본 다음과 같은 Virtual Machine settings 화면으로 이동하면 된다. 이 화면에서 Options 탭을 클릭하고 좌측 창에서 Shared Folders 항목을 클릭한 후 우측 창에서 Always enabled 항을 체크한다.

88 라즈베리파이기반 임베디드시스템용



이제 공유폴더의 경로를 지정하기 위해, 위 화면의 우측창에서 Add.. 버튼을 클릭하면 다음의 화면이 나타난다. 이 화면에서 Windows 환경에서 공유하려 생성한 폴더의 경로를 지정하여 주고 Finish 버튼을 클릭한다.

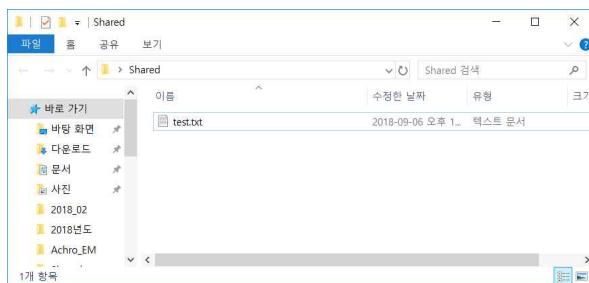


이로써 공유폴더를 사용하는데 필요한 조치는 완료되었다. 이렇게 되면 이후부터 Windows의 공유 폴더 Shared는 가상머신의 파일시스템에서 /mnt/hgfs/Shared 디렉터리로 마운트된다.

즉, Windows 환경에서 가상머신으로 전달할 자료는 Shared 폴더에 옮겨놓고, 가

상머신에서는 /mnt/hgfs/Shared 경로에서 그 자료를 접근할 수 있다. 역으로 가상머신에서 Windows로 전달할 자료가 있으면 /mnt/hgfs/Shared 디렉터리로 옮겨놓으면, Windows에서는 Shared 폴더내에서 그 자료를 접근할 수 있다.

공유 폴더의 사용을 확인하기 위해 Windows의 Shared 폴더에 test.txt 파일을 다음과 같이 생성한다.



메모장을 열어 test.txt 파일의 내용을 다음과 같이 편집한다.

Hello.....

가상머신 상에서 다음의 명령을 사용하여 Windows 환경에서 작성된 파일을 접근할 수 있다.

```
ifc415@ubuntu:~$  
ifc415@ubuntu:~$ sudo su // su(root 계정)로 전환  
[sudo] password for ifc415: embedded  
root@ubuntu:/home/ifc415# cd // root의 홈 디렉터리 이동  
root@ubuntu:~#  
  
root@ubuntu:~# ls /mnt/hgfs/  
Shared  
root@ubuntu:~# cd /mnt/hgfs/Shared/ // 공유 디렉터리로 이동  
root@ubuntu:/mnt/hgfs/Shared# ls  
test.txt  
root@ubuntu:/mnt/hgfs/Shared# cat test.txt // 파일내용 보기
```

Hello.....

위의 역 과정으로 가상머신 상에서 다음의 명령을 사용하여 test.txt 파일을 수정한 후 저장하면 Windows 환경에서 수정된 내용을 확인할 수 있다.

```
root@ubuntu:/mnt/hgfs/Shared# gedit test.txt  
Hello.....Raspberry pi.....
```

3.4 툴 체인

가상 머신에서 라즈베리파이 상에서 실행될 응용프로그램 등을 작성할 때는 라즈베리파이용 컴파일러를 설치하여야 한다. 이렇게 타깃 시스템과 다른 환경에서 타깃용의 실행파일을 생성하기 위한 컴파일러를 툴 체인(tool-chain) 혹은 크로스 컴파일러라고 한다.

3.4.1 필수 패키지 설치

개발과 관련된 필수 패키지들을 다음의 절차에 따라 설치한다. 32비트 호환 라이브러리, ncurses, 개발을 위한 기본 패키지들을 설치한다.

```
root@ubuntu:~# dpkg --add-architecture i386  
  
root@ubuntu:~# apt-get update  
root@ubuntu:~# apt-get upgrade  
root@ubuntu:~# apt-get install lib32z1 // 32bit 호환  
  
root@ubuntu:~# apt-get install lib32ncurses5 // ncurses  
root@ubuntu:~# apt-get install lib32ncurses5-dev  
  
root@ubuntu:~# apt-get install build-essential // 개발
```

3.4.2 크로스 컴파일러

크로스 컴파일러 설치

다음의 명령을 사용하여 라즈베리파이용의 크로스 컴파일러를 설치한다. 컴파일러는 /usr/ 디렉터리에 설치된다.

```
root@ubuntu:~# apt-get install gcc-arm-linux-gnueabihf
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer
required:
libqpdf13
.......
```

컴파일러의 버전 정보를 출력하는 다음 명령을 사용하여 컴파일러 설치를 확인할 수 있다.

```
root@ubuntu:~# arm-linux-gnueabihf-gcc --version
arm-linux-gnueabihf-gcc (Ubuntu/Linaro 7.3.0-27ubuntu1~18.04) 7.3.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is
NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
```

크로스 컴파일러 경로 설정

크로스 컴파일러의 경로를 컴파일 때마다 명령행에서 지정하는 것은 번거롭다. 따라서 이의 해결을 위해 다음의 절차를 진행한다.

슈퍼유저의 홈 디렉터리에 있는 히든 파일 .bashrc 파일을 다음 명령을 사용하여 편집한다.

```
root@ubuntu:~# gedit .bashrc
# 파일의 끝에 다음을 추가
# Cross Compiler - raspberrypi
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
```

수정된 .bashrc 파일의 내용을 반영하기 위해 다음의 명령을 실행한다.

```
root@ubuntu:~# source .bashrc
```

3.4.3 소스 컴파일

C 소스 작성

간단한 메시지를 출력하는 C 소스 파일을 다음 명령을 사용하여 작성한다.

```
root@ubuntu:~# gedit hello.c
#include <stdio.h>

int main(void) {
    printf("Hello....\n\n");
    return 0;
}
```

타깃용 크로스 컴파일

위의 소스 파일을 타깃용의 실행파일로 생성하는 크로스 컴파일을 다음 명령과 같이 실행한다.

```
root@ubuntu:~# arm-linux-gnueabihf-gcc -o t_hello hello.c
root@ubuntu:~# ls
t_hello  hello.c
```

실행파일인 t_hello를 다음과 같이 실행하면 가상머신에서는 실행할 수 없다는 메시지가 출력된다.

```
root@ubuntu:~# ./t_hello
bash: ./hello: cannot execute binary file: Exec format error
```

다음의 명령을 통해 t_hello 실행파일이 ARM용, 즉 라즈베리파이용임을 확인할 수 있다.

```
root@ubuntu:~# file ./t_hello
./hello: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.32,
BuildID[sha1]=605ab13e2935cea088c3bc9d91fd43718e8d02a2, not stripped
```

호스트용 컴파일

소스 파일을 호스트, 즉, 가상머신용의 실행파일로 생성하는 컴파일을 위해 다음 명령을 사용한다.

```
root@ubuntu:~# gcc -o h_hello hello.c
root@ubuntu:~# ls
h_hello  t_hello  hello.c
```

호스트용의 실행파일을 다음과 같이 실행하면, 잘 실행되고 실행 결과를 확인할 수 있다.

```
root@ubuntu:~# ./h_hello
Hello....
```

다음의 명령을 통해 h_hello 실행파일이 x86-64용, 즉 가상머신용임을 확인할 수 있다.

```
root@ubuntu:~# file ./h_hello
```

```
./h_hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),  
dynamically linked (uses shared libs), for GNU/Linux 2.6.24,  
BuildID[sha1]=6f55ff519eceb3a255ebcb3eb29620dad454ece9, not stripped
```

3.4.4 라즈베리파이에서 실행

우선 생성된 파일들을 NFS 서비스위한 디렉터리로 복사한다. 다음의 명령으로 _hello로 끝나는 모든 파일들을 /nfs 디렉터리로 복사하고, 복사여부를 확인한다.

```
root@ubuntu:~# cp *_hello /nfs  
root@ubuntu:~# ls /nfs  
h_hello t_hello
```

이 과정을 통하여 NFS 서버측에서는 NFS 서버가 실행 중에 있고, 공유할 파일들도 /nfs에 복사되어 있는 상태이다.

이제 라즈베리파이에서 NFS 서버측의 공유 디렉터리를 적절한 곳에 마운트하여 파일들을 공유할 수 있다.

라즈베리파이에 puTTY로 접속한 후, 터미널 창을 열어 다음의 명령들을 사용하여 마운트 하고 공유되었는지 확인한다.

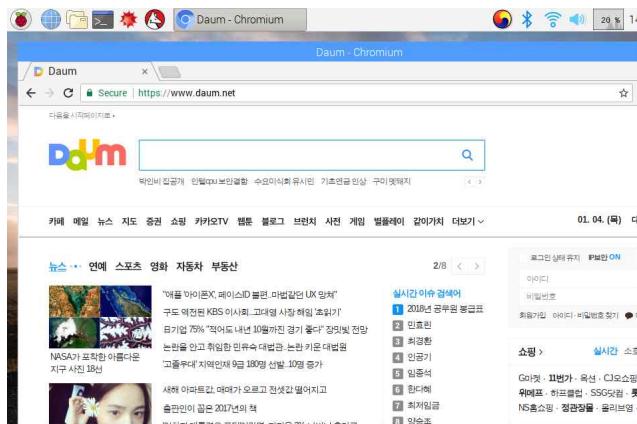
```
$ sudo mount -t nfs 192.168.0.20:/nfs /share  
$ sudo ls /share  
h_hello t_hello
```

이제 다음 명령들을 사용하여 /share 디렉터리로 이동하여 바로 실행하여 본다.

```
$ sudo cd /share  
$ sudo ./t_hello // 정상적으로 실행  
Hello....  
$ sudo ./h_hello // 실행 불가
```

3.5 한글 표시 및 입력기 설치

라즈베리파이 보드나 가상 머신에서 왼쪽 상단에 Menu 옆의 지구본 문양의 웹 브라우저 아이콘을 선택하여 실행하고, 웹 브라우저 주소창에서 접속할 사이트 주소를 입력하여 인터넷 접속을 할 수 있다. 실제로는 다음 그림과 같이 한글이 제대로 표시되지 않을 수 있다. 이의 해결을 위해서는 한글 표시기 및 입력기의 설치과정을 거쳐야 한다.



한글이 표시될 때 깨지는 현상과 한글의 입력을 가능하게 하려면 관련 패키지를 설치하여야 한다.

3.5.1 한글 폰트 패키지 설치

터미널 창의 프롬프트에서 다음 명령들을 사용하여 기존 패키지를 업데이트 및 업그레이드 한다.

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

이제 본격적인 한글 관련 패키지들의 설치를 진행하도록 하자. 현재 상태로는 웹브

96 리눅스 배리피아이기반 임베디드 시스템 활용

라우저에서 한글이 깨져 표시되는데 이를 제대로 보이게 하려면 다음의 명령을 통해 fonts-unfonts-core 패키지를 설치하여야 한다.

```
$ sudo apt-get install fonts-unfonts-core
```

```
.....
```

```
.....
```

```
Do you want to continue? [Y/n]
```

```
.....
```

패키지들을 설치하는 동안에 위와 같은 진행여부를 묻는 질의가 나타나면 'y'를 입력하여 설치 진행을 계속한다. 이후 패키지 설치할 때 위와 같은 질의가 나타나면 동일한 방식으로 진행한다.

3.5.2 한글 입력기 패키지 설치

때로, 웹 브라우저, 터미널, 프로그램 소스 등에서 한글을 입력할 필요 있을 수 있다. 이 경우를 위해서는 한글 입력기 관련 패키지를 설치하여야 한다. 즉, 다음의 명령을 통해 ibus-hangul 패키지를 설치하여야 한다.

```
$ sudo apt-get install ibus-hangul
```

```
.....
```

```
.....
```

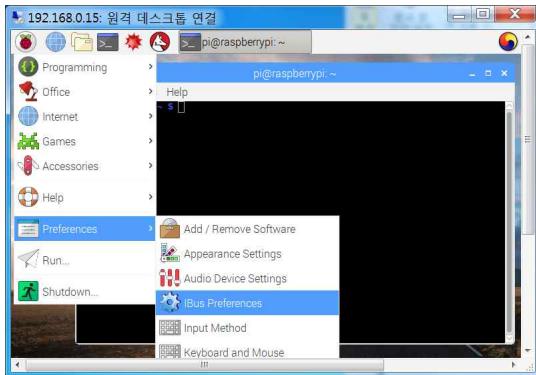
```
Do you want to continue? [Y/n]
```

```
.....
```

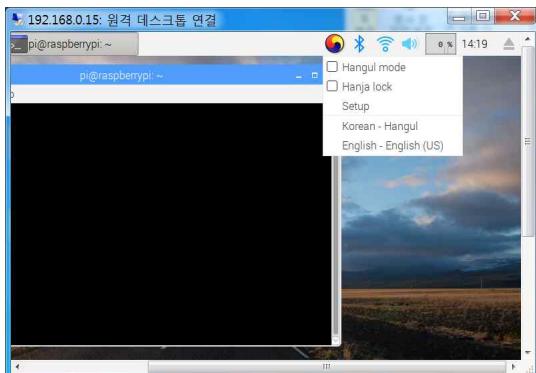
이제 한글 관련 패키지들의 설치가 완료되었다. 설치된 패키지를 반영하도록 재부팅한다. 재부팅은 다음의 명령을 사용하거나, 산딸기(시작) - Shutdown - Reboot 항목을 선택하여 진행할 수 있다.

```
$ sudo reboot
```

재부팅 후, 다음과 같이 산딸기(시작) - Preferences - iBUS preferences 항목이 추가된 것을 확인할 수 있다.



또한, 화면 우측 상단에 위와 같이 삼태극 모양의 새로운 아이콘이 생겼다면 한글 입력기 패키지의 설치가 완료된 것이다.



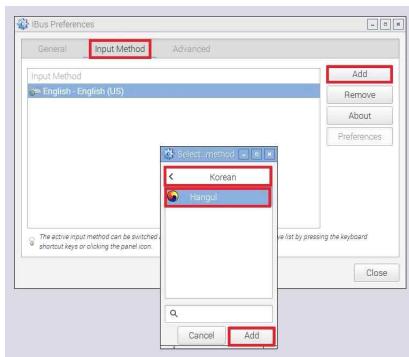
한글 폰트 패키지가 설치되었으므로, 웹 브라우저 등에서 한글의 깨짐이 더 이상 나타나지 않게 된다. 이제 한글 입력기를 시스템에 등록하고 자판 및 한영 전환기 등을 설정할 차례이다.

3.5.3 한글 입력기 등록, 한글자판 및 한영 전환기 등의 설정

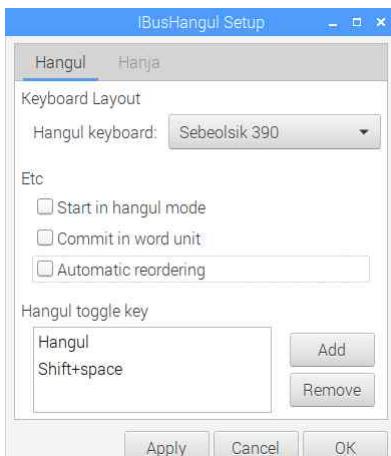
한글 입력기를 등록하려면 창의 우상단에 있는 삼태극 문양의 아이콘을 마우스로 우클릭하여 Preferences를 실행하거나, 산딸기(시작) - Preferences - iBUS preferences 항목을 선택한다. 그런 후 다음과 같이 나타난 창의 Input Method

98 리즈베리파이기반 임베디드시스템용

탭에서 Add 버튼을 클릭하여 “Korean - Hangul” 항목을 선택하여 추가한다.



화면의 우상단의 삼태극 문양의 아이콘을 클릭하여 한글 입력기 관련내역을 확인할 수 있으며, 다음과 같은 setup 항목에서 한글자판을 변경하거나, 한/영 전환키를 설정할 수 있다. 내정된 한영 전환기는 Shift+Space이다.



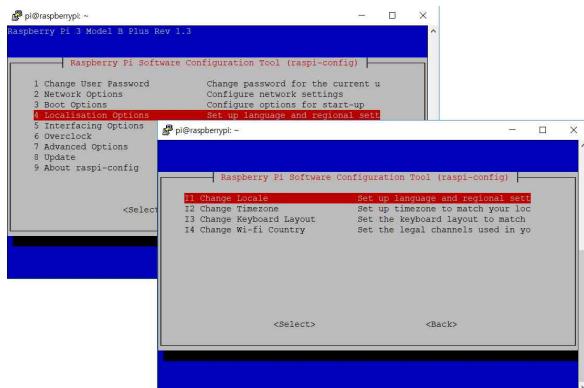
이제 웹브라우저에서 국내 포털 사이트 등을 접속하여 한글이 깨짐 없이 보이는지와 한글 입력 등이 가능한지를 확인할 필요가 있다. 혹은 터미널 창에서 한글 입력과 표시가 가능한지 확인할 수 있다.

3.5.4 시스템 환경에서 한글 표시

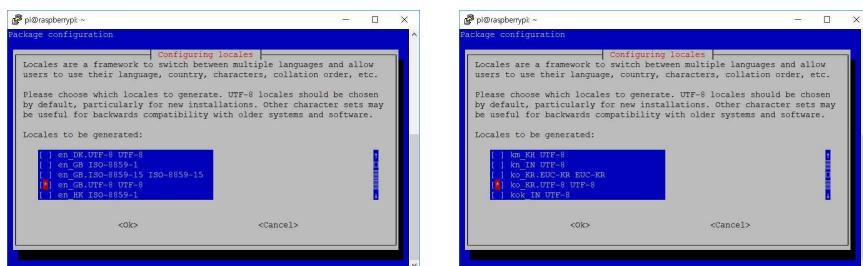
메뉴항목 등에서 한글로 표시하려면 시스템 환경에서 한글이 사용될 수 있도록 해야 한다. 이를 위해서는 다음 명령과 같이 raspi-config를 실행하여 환경 설정 한다.

```
$ sudo raspi-config
```

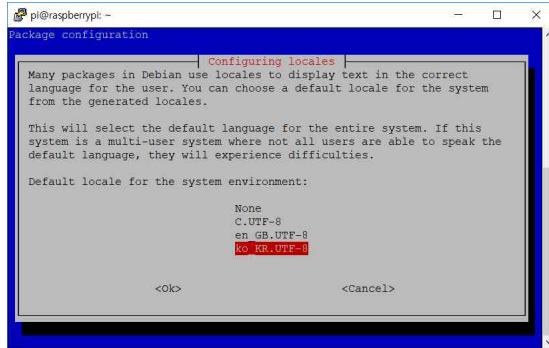
명령실행 후 나타난 다음과 같은 화면에서 localisation Options 항목을 선택한다.



서브화면에서 Change Locale 항목을 선택하고, 다음과 같은 화면이 나타나면 “ko_KR.UTF-8 UTF-8”을 찾아 스페이스바를 눌러 선택하고, Ok를 선택한다. 영어를 사용하도록 en_GB.UTF-8 UTF-8 혹은 en_US.UTF-8 UTF-8는 기본 선택되어 있다.



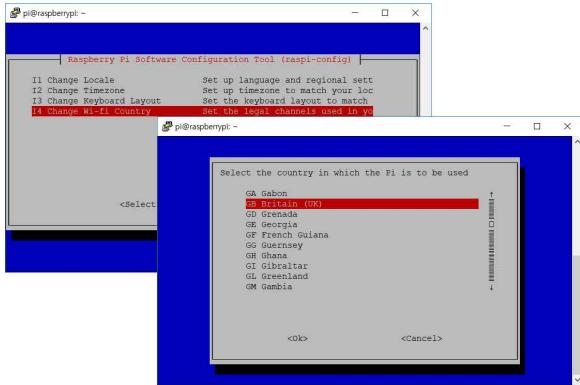
다음 그림과 같이 등록된 언어들 중 시스템 환경에서 사용할 언어를 선택하고, Ok 를 선택한다. 한글은 사용하도록 할 것이므로 kr_KR.UTF-8을 선택한다.



raspi-config를 종료하고, 재부팅하여 메뉴항목을 보면 다음과 같이 메뉴 등의 시스템 환경에서 한글로 표시되는 것 확인할 수 있다.

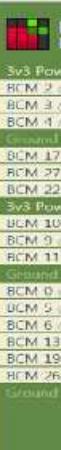


참고로, Change Timezone에서는 Area에 Asia를, Location에 Seoul을 선택하여 시간존을 대한민국으로 설정할 수 있으며, Change Keyboard Layout 항목에서는 자판 설정을 할 수 있다. Change Wi-Fi Country 항목은 디폴트로 GB Britain (UK)로 설정되어 있으며, 이 항목을 다른 것으로 변경하게 되면 Wi-Fi를 사용할 수 없게 되므로 유의할 필요가 있다.



참고자료

- [1] 한글 입력기 설치 <http://maker1st.tistory.com/4>
- [2] 한글 표시 설정 <http://nimtemdo.tistory.com/44>



제4장 입출력 디바이스 제어 I

라즈베리파이에 연결된 각종 입출력 디바이스 혹은 센서를 제어하기 위해서 매우 유용한 wiringPi 라이브러리가 제공된다. 이 라이브러리는 전적으로 라즈베리파이를 위한 것으로, 이를 활용한 입출력 디바이스 제어에 대해 살펴본다.

4.1 wiringPi 라이브러리

라즈비안을 micro SD 카드에 기록한 경우 기본적으로 wiringPi 라이브러리가 설치되어 있다. wiringPi 라이브러리 설치 여부는 다음 명령을 사용하여 버전관련 정보 등이 나타나면 이미 설치된 것이다.

```
pi@raspberrypi:~/IFC415/04_IO_1 $ gpio -v
gpio version: 2.50
Copyright (c) 2012-2018 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty
.....
```

4.1.1 라즈베리파이에 wiringPi 설치

라즈베리파이 보드에 wiringPi 라이브러리를 직접 설치하는 경우, 소스의 다운로드 및 설치 과정에 대해서는 다음의 사이트 자료를 참고한다.

<http://wiringpi.com/download-and-install/>

위의 사이트에서 wiringPi 라이브러리 소스인 wiringPi-8d188fa.tar.gz 파일을 다운로드 받은 상태에서 설치하는 과정은 다음과 같다. 해당 파일은 Windows 환경에서 다운로드 받았고, 이 tar 파일을 공유 폴더에 위치시켰다고 가정한다.

다음의 명령들을 사용하여 라이브러리 소스파일을 복사하고 풀기하면 디렉터리가 생성된다. 생성된 디렉터리 내의 build 스크립트를 실행하여 설치한다. 설치과정에서 현재 시스템의 C 컴파일러로 컴파일된 후 설치함을 유의한다.

```
$ cp /mnt/hgfs/Shared/wiringPi-8d188fa.tar.gz ./
$ tar xvfz wiringPi-8d188fa.tar.gz ./
$ cd wiringPi-8d188fa
$ ./build
```

설치확인은 다음의 명령을 사용하여 버전관련 정보 등이 나타나면 정상적으로 설치된 것이다.

```
pi@raspberrypi:~/IFC415/04_IO_1/wiringPi-8d188fa $ gpio -v
gpio version: 2.46
Copyright (c) 2012-2018 Gordon Henderson
This is free software.....
```

wiringPi 라이브러리의 헤더 파일 및 목적코드들은 각각 /usr/include와 /usr/lib 디렉터리 아래에 설치되며, 다음의 명령을 통하여 확인가능하다.

```
pi@raspberrypi:~ $ ls /usr/include/wi*
/usr/include/wiringPi.h      /usr/include/wiringSerial.h
/usr/include/wiringPiI2C.h   /usr/include/wiringShift.h
/usr/include/wiringPiSPI.h

pi@raspberrypi:~ $ ls /usr/lib/libwi*
/usr/lib/libwiringPiDev.so    /usr/lib/libwiringPi.so
/usr/lib/libwiringPiDev.so.2.50 /usr/lib/libwiringPi.so.2.50
```

또 다른 설치 방법으로 git 툴을 사용하여 가장 최근의 소스를 온라인에서 다운로드하여 설치하는 경우는 다음의 절차로 진행한다. 이를 위해서는 git 툴 패키지를 설치해야 하므로, 다음의 과정을 진행한다.

```
$ apt-get update
$ apt-get upgrade
```

```
$ apt-get install git-core // git 툴 설치
```

이제, git 툴을 이용하여 다운로드하면 wiringPi 디렉터리가 생성되면서 그 하부에 소스가 위치한다. 생성된 디렉터리 내의 build 스크립트를 실행하여 설치한다.

```
$ git clone git://git.drogon.net/wiringPi // 다운로드
$ cd wiringPi
$ ./build
```

이제 wiringPi 라이브러리를 사용하는 응용 소스를 컴파일할 때는 다음과 같이 사용한다. 응용 소스는 led_01.c라 가정한다.

```
$ gcc -o led_01 led_01.c -lwiringPi
```

혹은, 다음의 Makefile을 작성하여 make 유ти리티를 활용할 수 있다.

```
$ cat Makefile
=====
# Makefile
=====
CC = gcc # for raspberrypi
#CC = arm-linux-gnueabihf-gcc # for virtual machine

CFLAGS = -c
LDFLAGS = -lwiringPi
LDLIBS =

%: %.c
    $(CC) -o $@ $< $(LDFLAGS)

default:
    @echo "Type \"make filename without .c\" to compile..."

clean:
    rm -f *.o
```

make 유ти리티를 사용하여 응용 소스를 컴파일할 때는 다음과 같이 소스 파일명을 확장자 없이 사용한다.

```
# make led_01
```

4.1.2 가상머신에 wiringPi 설치

wiringPi 라이브러리는 전적으로 라즈베리파이 보드용으로 설계된 것이다. 가상머신에 wiringPi 라이브러리를 설치하는 경우는 build 스크립트가 실행되면서 시스템의 C 컴파일러로 컴파일한 후 설치하므로 파일포맷이 다르다. 따라서 앞서의 라즈베리파이보드에 설치하는 과정과는 달리해야 한다.

라즈베리파이 보드에 이미 설치된 wiringPi 관련 파일들을 크로스 컴파일러를 위한 라이브러리 디렉터리들로 복사하는 방법을 생각할 수 있다. 즉, 라즈베리파이 보드의 /usr/include/wi* 파일들을 가상머신상의 크로스 컴파일러가 있는 디렉터리인 /usr/arm-linux-gnueabihf/include 디렉터리로 복사하고, 라즈베리파이 보드의 /usr/lib/libwi* 파일들을 가상머신상의 크로스 컴파일러가 있는 디렉터리인 /usr/arm-linux-gnueabihf/lib 디렉터리로 복사하면 된다.

가상머신에는 크로스컴파일러가 설치되어 있고, NFS 서비스 또한 사용할 수 있는 상황이라고 가정한다. 즉 가상머신의 / nfs 디렉터리를 라즈베리파이보드의 /share 디렉터리에 마운트된 상태라고 가정한다.

우선 라즈베리파이 보드에서 다음 명령들을 사용하여 /share 디렉터리로 wiringPi 관련 파일들을 복사한다.

```
pi@raspberrypi:~ $ sudo mkdir /share/wPi/include -p  
pi@raspberrypi:~ $ sudo mkdir /share/wPi/lib -p  
pi@raspberrypi:~ $ ls /share/wPi/ // 디렉터리 생성 확인  
include lib  
pi@raspberrypi:~ $ sudo cp /usr/include/wi* /share/wPi/include/  
pi@raspberrypi:~ $ sudo cp /usr/lib/libwi* /share/wPi/lib/
```

이제 가상머신에서 /nfs 디렉터리에 있는 관련 파일들을 크로스 컴파일러의 관련 디렉터리들로 다음의 명령들을 사용하여 복사한다.

```
root@ubuntu:~# ls /nfs/wPi/
include  lib
root@ubuntu:~# cp /nfs/wPi/include/* /usr/arm-linux-gnueabihf/include/
root@ubuntu:~# cp /nfs/wPi/lib/* /usr/arm-linux-gnueabihf/lib/
```

다음의 명령으로 크로스 컴파일러의 관련 디렉터리들로 잘 복사되었는지 확인한다.

```
root@ubuntu:~# ls /usr/arm-linux-gnueabihf/include/wi*
root@ubuntu:~# ls /usr/arm-linux-gnueabihf/lib/libwi*
```

이로서 가상머신에 wiringPi 라이브러리의 설치가 완료된다.

이제 wiringPi 라이브러리를 사용하는 응용 소스를 크로스 컴파일할 때는 다음과 같이 컴파일러로 크로스 컴파일러를 명시한다.

```
# arm-linux-gnueabihf-gcc -o led_01 led_01.c -lwiringPi
```

혹은, Makefile을 다음과 같이 수정하여 크로스 컴파일러를 지정하여 사용한다.

```
# cat Makefile
=====
# Makefile
=====
#CC = gcc # for raspberrypi
CC = arm-linux-gnueabihf-gcc # for virtual machine

CFLAGS = -c
LDFLAGS = -lwiringPi
LDLIBS =

%: %.c
$(CC) -o $@ $< $(LDFLAGS)
```


위의 화면에서 보듯이 각 GPIO 핀에 대한 BCM 번호체계와 wiringPi 번호체계가 각기 다를 것을 확인할 수 있다.

GPIO 핀을 통해 디바이스 제어 실습을 진행하려면 다음 그림과 같이 브래드보드에 T-Cobbler를 장착하고 40P 플랫케이블을 사용하여 라즈베리파이보드와 연결한다. 디바이스 회로는 브래드 보드에 구성한다.



4.1.4 wiringPi.h 헤더 파일

wiringPi.h 헤더파일은 라즈베리파이의 GPIO 핀을 제어하기 위한 필수 라이브러리 함수들을 선언하고 있다. 이들 중 주요 함수들의 기능을 살펴보면 다음과 같다.

```
wiringPiSetup();                                // wiringPi pin # 사용
GPIO 핀의 번호체계를 wiringPi 번호체계를 사용하도록 설정하는 함수

wiringPiSetupGpio();                            // BCM pin # 사용
GPIO 핀의 번호체계를 BCM GPIO 번호체계를 사용하도록 설정하는 함수

pinMode(P_LED, OUTPUT);                         // output 용
pinMode(P_BTN, INPUT);                          // input 용
해당 핀의 입출력 용도를 입력용 혹은 출력용인지 설정하는 함수
```

```

digitalWrite(P_LED, LOW);           // write digital signal LOW, 0, 0v
digitalWrite(P_LED, HIGH);          // write digital signal HIGH, 1, 5v
해당 핀에 low, high 신호를 출력하는 함수

```

```

digitalRead(P_BTN);                // read digital signal
해당 핀의 신호를 읽어 들이는 함수

```

4.2 LED 제어

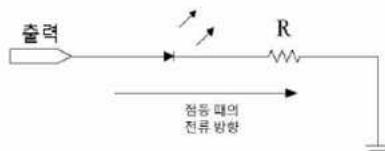
LED 출력 디바이스 제어에 대해 살펴본다.

4.2.1 LED 회로

LED(light emitted diode)는 전류가 흐르면 점등되는 다이오드이다. 입출력 장치로 LED를 구동할 때 두 가지 회로로 구성할 수 있다.

정논리의 LED 회로

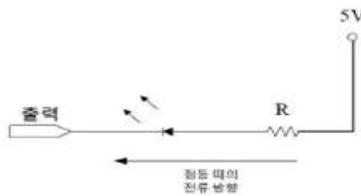
정논리의 LED 회로는 다음 회로와 같이 GPIO 핀을 전류의 소스(Source)로 사용하는 것이다.



GPIO 핀(출력)에 논리 0을 출력하면 GPIO 핀의 전압은 0V이어서 전류가 흐르지 않고 LED가 꺼진다. 반면, 논리 1을 출력하면 GPIO 핀의 전압은 5V가 되어 순방향 바이어스가 걸려 LED가 켜진다. 이와 같이 GPIO 핀에 논리 1을 인가하였을 때 LED가 ON되고, 논리 0을 인가하였을 때 LED가 OFF되며, 이 같이 동작하는 것을 정논리로 동작한다고 한다.

부논리의 LED 회로

부논리의 LED 회로는 다음 회로와 같이 GPIO 핀을 전류의 싱크(Sink)로 사용하는 것이다.

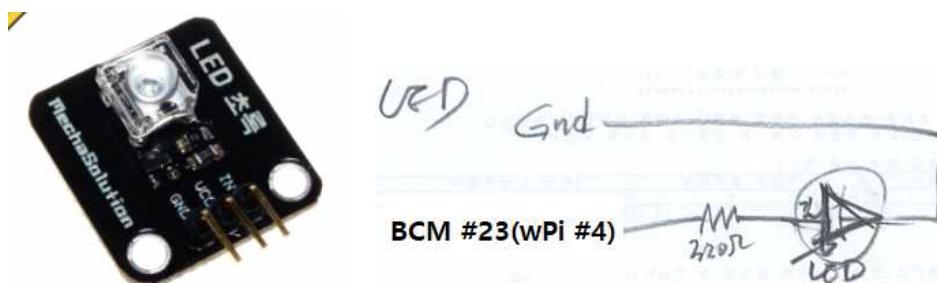


GPIO 핀(출력)에 논리 0을 출력하면 출력 단의 전압은 0V이고 따라서 다이오드에 순방향 바이어스가 걸리고 전류가 흘러 LED가 켜진다. 반면 논리 1을 출력하면 출력 단의 전압은 5V가 되어 전류가 흐르지 않아 LED가 꺼진다. 이 같은 방식으로 동작하는 것을 부논리로 동작한다고 한다.

또한, 회로내의 저항은 전류의 세기를 결정함으로써 LED 밝기를 결정하는 소자이다. 저항이 크면 전류의 세기는 작아져 LED의 밝기는 어두워진다. 필요에 따라 저항은 제거하여도 된다.

LED 모듈

다음과 같은 LED 모듈을 사용하거나, 회로도와 같이 브래드 보드에 정논리의 LED 회로를 구성한다. GPIO 핀은 BCM_GPIO #23을 사용한다.



4.2.2 실습과제

[실습1] LED 제어 I (wiringPi GPIO 핀번호 체계)

다음의 소스는 LED를 OFF, ON 하기를 반복하는 프로그램으로, LED는 Low 신호에서 OFF, High 신호에서 ON된다. GPIO 핀 번호는 wiringPi 번호 체계를 사용하고, wiringPi.h 라이브러리 함수를 활용한 것이다.

```
$ sudo nano led_01.c
//=====
// led_01.c
//
//=====
#include <stdio.h>
#include <wiringPi.h>

#define P_LED    4           // *1) BCM_GPIO #23

int main(void) {
    printf("[LED testing....]\n");

    if(wiringPiSetup() == -1)          // *2) wiringPi pin# scheme
        return 1;

    pinMode(P_LED, OUTPUT);          // output

    while(1) {
        printf("OFF.....\n");
        digitalWrite(P_LED, LOW);      // 0
        delay(1000);

        printf("ON.....\n");
        digitalWrite(P_LED, HIGH);     // 1
        delay(1000);
    }

    return 0;
}

$ gcc -o led_01 led_01.c -lwiringPi
```

```
$ sudo ./led_01
```

```
pi@raspberrypi: ~
root@raspberrypi:~/IFC413/05_IO_1/led# ./led_01
[LED testing....]
OFF.....  
ON.....  
OFF.....  
ON.....  
OFF.....  
ON.....
```

가상머신에서 이 응용 소스를 컴파일할 때는 다음의 명령을 사용한다. 생성된 파일의 포맷은 ARM용으로 확인되며, 라즈베리파이 보드에 전달하여 실행한다.

```
# arm-linux-gnueabihf-gcc -o led_01 led_01.c -lwiringPi
# file led_01
led_01: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV),
dynamically linked, interpreter /lib/ld-, for GNU/Linux 3.2.0,
BuildID[sha1]=a237df6a6e0e944917905043b627ef8d26f272c5, not stripped
```

[실습2] LED 제어 II (BCM GPIO 핀번호 체계)

[실습1]의 소스에서 GPIO 번호 체계를 wiringPi 번호 체계대신 BCM 번호 체계를 사용하도록 재 구현하여 본다.

GPIO 번호 체계를 BCM 번호 체계로 사용하려면 위 소스에서 *)로 표시된 부분을 아래와 같이 대체하면 된다.

*1) 부분을

```
#define P_LED 23 // BCM_GPIO #
```

*2) 부분을

```
if(wiringPiSetupGpio() == -1) // BCM GPIO pin# scheme
    return 1;
```

4.3 시그널 처리

보통 디바이스 제어 프로그램을 종료할 경우 Ctrl-C 키를 눌러 강제 종료한다. 이 때 GPIO 신호는 프로세스 종료 시점의 상태를 그대로 유지하게 된다. 즉, 전원을 켜ن 상태나 전등이 ON인 상태에서 프로그램을 종료해도 전원이 공급되고, 전등이 ON 상태 그대로 남아있게 된다. 이는 디바이스 유형에 따라 심각한 문제를 야기할 수 있다. 이를 방지하는 방법에 대해 알아보자.

프로세스가 예외적인 상황으로 인해 종료될 때 발생될 수 있는 다양한 유형의 시그널을 제공하며, 이를 적절히 처리함으로써 문제 발생의 소지를 줄일 수 있다. 이러한 프로세서 시그널을 처리하기 위한 라이브러리 함수를 위해 signal.h 헤더파일이 기본적으로 C 언어에서 제공된다.

signal.h 라이브러리

시그널 핸들러 함수의 프로토타입은 다음과 같으며, 함수 명은 사용자가 적절히 명명할 수 있다.

```
// signal handler func.  
void INTHandler(int sig);
```

다음 함수는 특정 시그널에 대한 핸들러 함수를 등록하는 함수로, 첫 인자는 시그널 유형이며, 두 번째 인자는 서비스 해줄 시그널 핸들러 함수이다.

```
signal(SIGINT, INTHandler);
```

프로세스 시그널 유형

컴퓨터 시스템 상에서 발생할 수 있는 여러가지 상황이 있을 수 있고, 이는 OS에 따라 다르게 설정된다. 라즈베리 파이에서 지원하는 시그널은 kill 명령을 통해 알아볼 수 있다.

```
pi@raspberrypi:~/softbox/softboxtest $ kill -l
```

```
pi@raspberrypi:~/IFC413/05_IO_1/led $ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT      4) SIGILL      5) SIGTRAP
 6) SIGABRT     7) SIGBUS      8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTOU      23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS     34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
pi@raspberrypi:~/IFC413/05_IO_1/led $
```

보통 32번까지의 시그널은 대부분의 UNIX 계열에서 비슷한 값을 보여준다. 33번부터 64번까지는 OS마다 약간의 차이를 보이기도 한다. 33번부터 62번까지는 리얼타임과 관련해서 예약된 시그널들이다. 각각의 시그널은 int 형의 숫자로 표시되며, 시그널 번호에 따라, 각기 다른 상황을 표시한다. 몇 가지 중요한 시그널만을 정리하면 아래 표와 같다.

시그널 이름	설명
SIGHUP	터미널을 잃어버렸을 때
SIGABRT	프로그램의 비정상 종료시
SIGINT	Ctrl-C나 DELETE 키를 입력했을 때
SIGIO	비동기적인 입출력이 발생했을 때
SIGKILL	프로세스를 강제 종료할 때
SIGPIPE	단절된 파일에 write할 경우 발생
SIGSEGV	잘못된 메모리 참조를 시도하였을 때
SIGSTOP	프로세스를 일시중단할 때 (Ctrl+z)
SIGUSR1	사용자를 위해 정의된 시그널

프로세스를 강제로 종료하기 위해 Ctrl-c를 누를 때 SIGINT 시그널이 발생된다. 이 시그널이 발생되었을 때 문제가 발생할 수 있는 상황을 방지할 수 있도록 적절히 처리한 후, 종료할 수 있도록 하는 것이 바람직하다.

[실습3] LED 제어 II (시그널 처리)

LED를 OFF, ON하기를 반복하는 프로그램으로, LED는 Low 신호에서 OFF,

High 신호에서 ON된다. 추가적으로 프로그램 수행시 Ctrl-c 키 입력에 대해 시그널 핸들러 함수를 활용하도록 구현한 것이다.

```
$ sudo nano led_02.c
//=====
// led_02.c
//      using signal handler function
//=====

#include <stdio.h>
#include <wiringPi.h>
#include <signal.h>      // signal
#include <stdlib.h>      // exit()

#define P_LED    4          // BCM_GPIO #23

void sig_Handler(int sig);
void init(void);

int main(void) {
    printf("[LED testing....]\n");

    if(wiringPiSetup() == -1)           // wiringPi pin #
        return 1;

    signal(SIGINT, sig_Handler);      // register signal handler func

    pinMode(P_LED, OUTPUT);

    while(1) {
        printf("OFF.....\n");
        digitalWrite(P_LED, LOW);
        delay(1000);

        printf("ON.....\n");
        digitalWrite(P_LED, HIGH);
        delay(1000);
    }

    return 0;
}
```

```
// signal handler function
void sig_Handler(int sig) {
    printf("\n\nEnding....\n\n");
    init();           // call init() func.
    exit(0);          //
}

void init(void) {
    digitalWrite(P_LED, LOW);           // LED OFF
}
```

\$ gcc -o led_02 led_02.c -lwiringPi
\$ sudo ./led_02

실행중 LED가 ON인 상태에서 Ctrl-c를 누르더라도, 시그널 처리에 의해 LED가 OFF 상태로 종료되는 것을 관찰할 수 있다. 이 같이 프로세스가 예기치 않은 상황으로 인해 종료될 경우에 초기 상태가 되게 조치할 수 있도록 시그널 핸들러 함수를 사용하여 구현할 것을 권고한다.

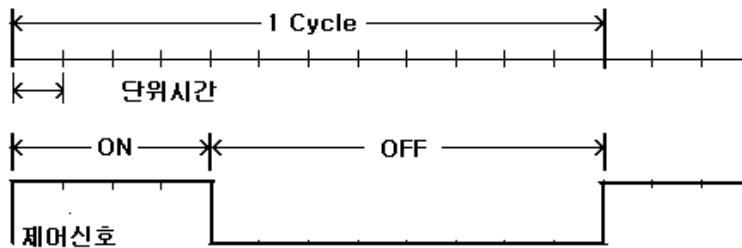
4.4 LED PWM 제어

LED 디바이스에 인가되는 신호 크기를 소프트웨어적으로 조절함으로써 LED의 밝기를 미세하게 조절하는 것이 가능하다. 이를 위한 방법이 PWM 제어이다.

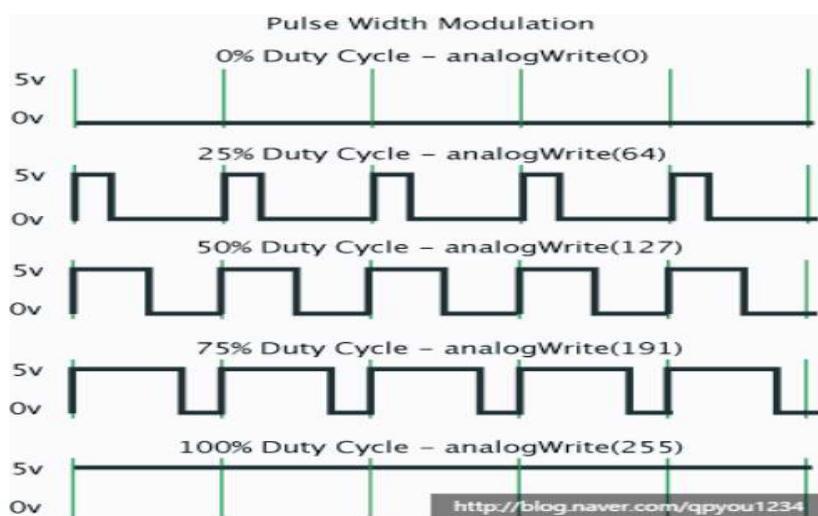
4.4.1 PWM(pulse width modulation) 제어

PWM 제어는 일정한 시간의 한 주기 내에서 제어신호의 ON, OFF시간을 조절함으로써 달성할 수 있다. 아래 그림과 같이 한 사이클이 12개의 단위시간으로 구성되고 이러한 사이클이 반복된다고 가정한다. 여기서 단위 시간은 일정시간을 소요하는 시간 지연 루틴으로 구성할 수 있다. 그림에서 한 사이클을 이루고 있는 제어신호가 단위 시간 4회의 ON과 8회의 OFF로 구성된 예를 보이고 있다. 한 사이클

을 구성하는 단위 시간들에 대하여 ON의 신호가 인가된 단위 시간의 비를 둘티비(duty ratio)라 한다. 이 경우 둘티비는 $4/12 = 1/3$ 이다.



이와 같이 한 사이클을 구성하는 ON과 OFF의 단위 시간 수를 조절함으로써 GPIO 핀에 연결된 디바이스에 그에 따른 수준의 신호를 인가할 수 있다. 다음 그림은 여러 사이클에 대한 둘티비에 따른 신호를 보여준다.



4.4.2 wiringPi.h 라이브러리

wiringPi.h 라이브러리에서 PWM 관련 함수들만 살펴보면 다음과 같다.

```
pinMode(P_LED, PWM_OUTPUT);
pinMode(P_LED, SOFT_PWM_OUTPUT);
해당 핀을 PWM 신호 출력용으로 설정하는 함수
```

```
pwmWrite(P_LED, 0);           // PWM 범위가 디폴트인 경우, min, 0~1023
pwmWrite(P_LED, 1023);        // PWM 범위가 디폴트인 경우, max, 0~1023
PWM 범위가 디폴트인 경우, 해당 핀에 신호를 출력하는 함수
```

4.4.3 softPwm.h 라이브러리

softPwm.h 라이브러리는 보조적으로 사용할 수 있는 라이브러리로 보다 섬세한 PWM 제어를 위한 함수들을 제공한다. 아래에서 이들 함수들의 기능을 살펴본다.

```
softPwmCreate(P_LED, 0, 100);      // PWM range
두 번째 인자는 초기치, 세 번째 인자는 PWM 범위를 나타내며, 이 함수를 사용하지 않으면 PWM 범위는 디폴트로 1024로 설정되어 있다.
```

```
softPwmWrite(P_LED, 0);           // min
softPwmWrite(P_LED, 50);          // 드티비 50%인 시그널 출력
softPwmWrite(P_LED, 100);         // max
해당 핀에 PWM 범위에 대한 %비로 신호를 출력하는 함수
```

[실습4] LED PWM 제어 I

LED를 OFF, ON 및 LED 밝기를 점차 어둡게 하기를 반복하는 프로그램으로, GPIO 핀 번호는 wiringPi 번호 체계를 사용하고, wiringPi.h와 softPwm.h 라이브러리 함수들을 활용한 것이다.

```
$ sudo nano led_03.c
//=====
// led_03.c
//     using softPwm funcs
//=====
#include <stdio.h>
```

```
#include <wiringPi.h>
#include <softPwm.h>           // PWM

#define P_LED    4                // BCM_GPIO #23

int main(void) {
    printf("[LED testing....]\n");

    if(wiringPiSetup() == -1)      // wiringPi pin# scheme
        return 1;

    //pinMode(P_LED, OUTPUT);       // output
    //pinMode(P_LED, PWM_OUTPUT);   // output
    pinMode(P_LED, SOFT_PWM_OUTPUT); // output

    softPwmCreate(P_LED, 0, 100); // *) set PWM range

    while(1) {
        printf("OFF.....\n");
        softPwmWrite(P_LED, LOW); // 0%
        delay(1000);

        printf("ON.....\n");
        softPwmWrite(P_LED, 100); // 100%
        delay(1000);

        printf("Dimming...\n");
        for(int i=100; i>=0; i--) {
            softPwmWrite(P_LED, i);
            delay(30);
        }
    }

    return 0;
}
```

```
$ gcc -o led_03 led_03.c -lwiringPi
$ sudo ./led_03
```

4.4.4 PWM 출력 전용 GPIO 핀 활용

softPwm.h 라이브러리를 사용않고 PWM 방식으로 제어 가능한 PWM 출력 전용의 핀은 BCM 번호체계의 #12, #13, #18, #19의 4개가 제공된다. 또한 이 핀을 사용하여 PWM 방식의 제어를 위해서는 /dev/gpiomem을 사용하므로 실행할 때 반드시 슈퍼유저의 권한으로 실행해야 함을 유의한다.

[실습5] LED PWM 제어 II (PWM 신호 전용 GPIO 핀 사용)

다음의 프로그램은 softPwm.h 파일을 사용하지 않고, PWM 방식으로 제어하기 위해서는 PWM 신호를 출력하는 GPIO 핀을 사용해야한다. PWM 신호 출력용의 4개 핀중 BCM_GPIO #18(wiringPi #1)을 사용한 소스이므로 LED 회로를 재연결한다.

```
$ sudo nano led_04.c
//=====
// led_04.c
//      controlled by PWM without softPwm.h
//=====

#include <stdio.h>
#include <wiringPi.h>

#define P_LED    1          // BCM_GPIO #18

int main(void) {
    int i;

    printf("[LED testing....]\n");

    if(wiringPiSetup() == -1)      // wiringPi pin# scheme
        return 1;

    pinMode(P_LED, PWM_OUTPUT);   // output

    for(i=0; i<=1024; i++) {
        printf("%d.....\n", i);
        pwmWrite(P_LED, i);
        delay(10);
    }
}
```

```
}

for(i=1023; i>=0; i--) {
    printf("%d.....\n", i);
    pwmWrite(P_LED, i);
    delay(10);
}

printf("OFF.....\n");
pwmWrite(P_LED, 0);
delay(1000);

return 0;
}
```

```
$ gcc -o led_04 led_04.c -lwiringPi
$ sudo ./led_04 // 슈퍼유저 권한으로 실행
```

다음과 같이 일반유저 권한으로 실행하는 경우 디바이스 파일 접근에 제한이 있음을 참고한다.

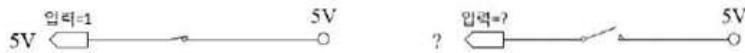
```
pi@raspberrypi:~/IFC415/04_IO_1/led $ ./led_04
[LED testing....]
pinMode PWM: Unable to do this when using /dev/gpiomem. Try sudo?
```

4.5 BTN 제어

4.5.1 푸쉬버튼 스위치

푸쉬버튼 스위치 회로의 입력단에 연결된 GPIO 핀은 입력 핀에 0V가 걸리면 논리 0을 읽고, 입력 핀에 5V가 걸리면 논리 1을 읽는다.

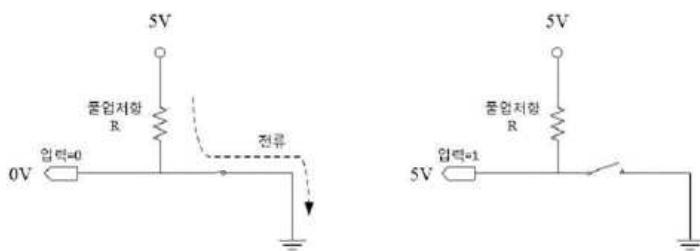
다음의 버튼 스위치 회로를 고려해보자.



좌측과 같이 스위치가 닫히면 입력 단에 5V가 걸려 논리 1이 입력된다. 반면, 우측과 같이 스위치가 열리면 입력 단의 전압은 0V가 아니라, 플로팅(floating) 상태가 된다. 이 때 입력 단자의 신호는 정의되지 않아서 논리를 결정할 수 없다. 따라서, 위와 같은 회로로는 정확한 스위치 상태를 파악할 수 없으므로 풀업 저항(pull-up resistor), 또는 풀다운 저항(pull-down resistor)을 사용해서 스위치 회로를 구성한다.

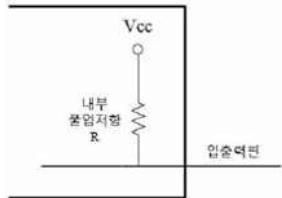
풀업 저항

다음 회로는 풀업 저항을 사용한 버튼 스위치 회로이다.



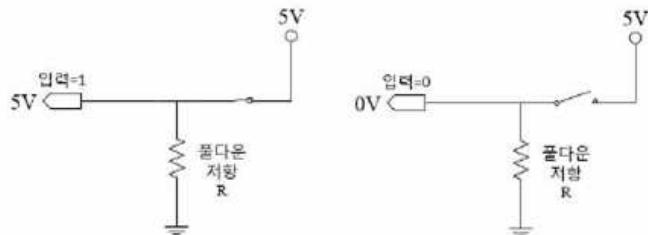
위 회로의 좌측과 같이 스위치가 닫히면 전류가 접지로 흐른다. 입력 단은 접지와 연결되어 있으므로 0V가 되고 입력단은 논리 0을 읽는다. 우측과 같이 스위치가 열리면 전류가 흐르지 않는다. 따라서 풀업 저항 사이의 전압 강하가 발생하지 않아 입력 단에는 5V가 걸린다. 따라서 입력단과 연결된 GPIO 핀은 논리 1을 읽는다. 이렇게 풀업 저항을 연결한 경우, 플로팅 상태는 발생하지 않는다.

컨트롤러에 따라서는 다음과 같이 내부 풀업저항을 제공하는 경우가 있으며, 이 경우 외부 풀업저항을 사용할 필요가 없다.



풀다운 저항

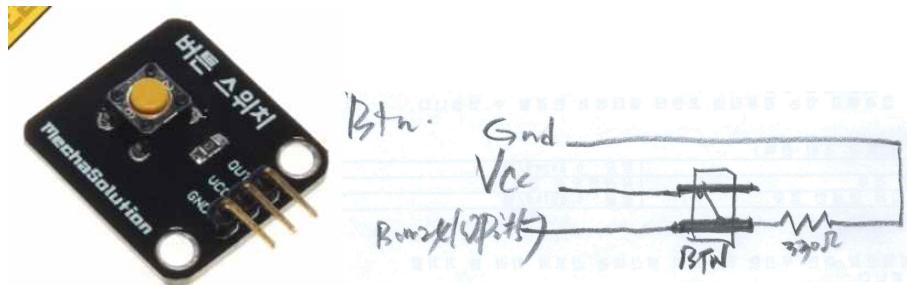
다음 회로는 풀다운 저항을 사용한 스위치 회로이다.



스위치가 닫히면 전류가 통하여 입력 단에는 5V가 걸린고, 입력단에 연결된 GPIO 핀으로부터 논리 1을 읽는다. 스위치가 열리면 입력 단은 저항을 통해 접지와 연결되어 있으므로 전류가 GND로 흐르게 되어 0V가 되고, 입력단의 GPIO 핀은 논리 0을 읽는다. 이와 같은 풀다운 회로도 플로팅 상태는 발생되지 않는다.

회로구성

다음과 같은 BTN 모듈을 사용하거나, 회로도를 참조하여 브레드 보드에 BTN 회로를 구성한다.



[실습6] LED/BTN 제어 I

다음 소스는 BTN 누르고 있는 동안 LED를 ON 하는 프로그램이다.

```
$ sudo nano ledBtn_01.c
//=====
// ledBtn_01.c
//
//=====
#include <stdio.h>
#include <wiringPi.h>

#define P_LED 1           // BCM_GPIO #18
#define P_BTN 5           // BCM_GPIO #24

int main(void) {
    int btn;

    printf("[LED/BTN testing....]\n");

    if(wiringPiSetup() == -1)      // wiringPi pin #
        return 1;

    pinMode(P_LED, OUTPUT);
    pinMode(P_BTN, INPUT);

    printf("LED ON while BTN press.....\n");

    btn = 0;
    while(1) {
```

```

btn = digitalRead(P_BTN);                                // read
if(btn==1) {
    digitalWrite(P_LED, HIGH);
    printf("ON.....\n");
}
else {
    digitalWrite(P_LED, LOW);
    printf("OFF.....\n");
}
delay(10);
}
return 0;
}

```

다음과 같이 make 유ти리티를 사용하여 컴파일한다.

```

$ make ledBtn_01
$ ./ledBtn_01

```

[실습7] LED/BTN 제어 II

BTN 누름에 따라 LED를 ON, OFF를 토글하는 프로그램이다.

```

$ sudo nano ledBtn_02.c
//=====
// ledBTN_02.c
//
//=====
#include <stdio.h>
#include <wiringPi.h>

#define P_LED 1          // BCM_GPIO #18
#define P_BTN 5          // BCM_GPIO #24

int main(void) {
    int led = 0;
    int in, inOld = 0;

```

```

if(wiringPiSetup() == -1)
    return 1;

pinMode(P_LED, OUTPUT);
pinMode(P_BTN, INPUT);

while(1) {
    in = digitalRead(P_BTN);
    if(inOld != in && in == 1) {
        //printf("detect!!\n");
        if(led == 0) {
            digitalWrite(P_LED, 1);
            led = 1;
            printf("ON.....\n");
        }
        else {
            digitalWrite(P_LED, 0);
            led = 0;
            printf("OFF.....\n");
        }
    }
    inOld = in;
    delay(10);
}
return 0;
}

```

```

$ make ledBtn_02
$ ./ledBtn_02

```

4.6 인터럽트 처리

4.6.1 이벤트 처리 방식

디바이스에서의 신호 변경에 대한 사건을 처리하는 방식은 풀링 방식과 인터럽트 방식으로 나누어 볼 수 있다.

풀링(polling) 방식

지금까지의 소스들은 신호변경 사건에 대해 질의하는 방식의 풀링 방식의 제어이다. 즉 아래 소스에서와 같이 디바이스로부터의 신호레벨 값을 프로그램에서 읽어 그 값에 따라 적절한 조치를 취하는 방식으로 처리하는 것을 말한다.

```
btn = digitalRead(P_BTN);           // read, Polling Method
if(btn==1) {
    digitalWrite(P_LED, HIGH);
    printf("ON.....\n");
}
else {
    digitalWrite(P_LED, LOW);
    printf("OFF.....\n");
}
```

인터럽트(Interrupt) 방식

인터럽트 방식은 특정 디바이스의 신호레벨을 읽어 처리하는 것이 아니다. 인터럽트 방식의 처리는 다음 소스와 같이 특정 디바이스에 사건이 발생되었을 때 실행 할 함수를 정의하고, 이 함수를 특정 디바이스의 특정 사건에서 실행되도록 등록하는 과정으로 구현한다.

```
// Interrupt Service Routine
void btnISR(void) {
    if(led == 0) {
        digitalWrite(P_LED, 1);
        led = 1;
        printf("ON.....\n");
    }
    else {
        digitalWrite(P_LED, 0);
        led = 0;
        printf("OFF.....\n");
    }
}
```

```
// main() function
.....
if( wiringPiISR(P_BTN, INT_EDGE_RISING, btnISR) < 0 ) {
    fprintf(stderr, "Unable to setup ISR: %sn", strerror(errno));
    return 1;
}
.....
```

4.6.2 인터럽트 취급관련 상수 및 함수

wiringPi.h 헤더 파일에 선언된 인터럽트 처리와 관련된 매크로상수 및 함수들은 다음과 같다.

```
// Interrupt levels(mode)
#define INT_EDGE_SETUP          0
#define INT_EDGE_FALLING        1
#define INT_EDGE_RISING          2
#define INT_EDGE_BOTH            3

// Interrupts
//      (Also Pi hardware specific)
extern int waitForInterrupt(int pin, int mS) ;
extern int wiringPiISR(int pin, int mode, void (*function)(void)) ;
특정 핀에 인터럽트 서비스를 등록하는 함수로, mode는 사건의 시점을 나타낸다.
```

4.6.3 실습과제

[실습8] LED/BTN 제어 III

BTN 누름에 따라 LED ON, OFF를 토글하는 프로그램. 버튼으로부터 인터럽트 요청에 의해 동작한다.

```
$ sudo nano ledBtnInt_01.c
```

```
//=====
// ledBtnInt_01.c
//     using Interrupt
//=====

#include <stdio.h>
#include <wiringPi.h>

#define P_LED 1           // BCM_GPIO #18
#define P_BTN 5           // BCM_GPIO #24

int led = 0;

// Interrupt Service Routine
void btnISR(void) {
    if(led == 0) {
        digitalWrite(P_LED, 1);
        led = 1;
        printf("ON.....\n");
    }
    else {
        digitalWrite(P_LED, 0);
        led = 0;
        printf("OFF.....\n");
    }
}

int main(void) {
    if(wiringPiSetup() == -1)
        return 1;

    pinMode(P_LED, OUTPUT);
    pinMode(P_BTN, INPUT);

    wiringPiISR(P_BTN, INT_EDGE_RISING, btnISR); //

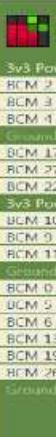
    while(1) {
        // Normal
    }

    return 0;
}
```

```
$ make ledBtnInt_01  
$ ./ledBtnInt_01
```

참고자료

- [1] 크로스컴파일환경에서 wiringPi 라이브러리 복사
<http://blog.naver.com/PostView.nhn?blogId=lkj900&logNo=221033794958>
- [2] “아두이노를 활용한 공학기초 다지기”, 김한종저, 영출판, 2017



제5장 입출력 디바이스 제어 II

본 장에서는 기타의 입출력 디바이스들을 제어하는 것에 대해 살펴본다.

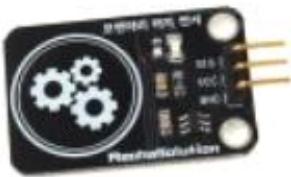
5.1 터치스위치 제어

5.1.1 IM120710023 터치스위치

터치스위치는 토클형과 비토클형이 있으며, 토클형은 터치할 때마다 이전상태의 반전된 상태의 신호를 출력하며, 비토클형은 터치가 없을 때는 HIGH 상태를, 터치하면 LOW 상태의 디지털 신호를 출력한다.



Toggle Touch Sensor



Direct Touch Sensor

이 모듈의 S단자는 디지털 출력 신호단자이며, V, G는 Vcc, Gnd 단자이다.

5.1.2 실습예제

[실습1] 비토클형 터치스위치를 이용하여 LED 점멸하기

터치스위치를 터치하고 있는 동안 LED를 ON하고, 터치 않는 동안 LED를 OFF하는 프로그램이다.

```
$ sudo nano touch_01.c
//=====
```

```
// touch_01.c
//      direct touch sensor
//=====
#include <stdio.h>
#include <wiringPi.h>

#define P_TCH 1                  // BCM_GPIO 18
#define P_LED 4                  // BCM_GPIO 23

int main(void) {
    printf("[P_TCH testing....]\n");

    if(wiringPiSetup() == -1)
        return 1;

    pinMode(P_TCH, INPUT);      // input
    pinMode(P_LED, OUTPUT);     // output

    while(1) {
        if(digitalRead(P_TCH)==HIGH) {           // not touch
            digitalWrite(P_LED, LOW);
            printf(".");
        }
        else {                                // touch
            digitalWrite(P_LED, HIGH);
            printf("\ntouched !!");
        }
    }
    return 0;
}

$ make touch_01
$ ./touch_01
```

5.2 자석스위치 제어

5.2.1 자석스위치

다음 그림과 같은 DC용 자석(리드)스위치는 주로 문이나 창문 등에 설치되어 개폐 여부를 확인하는데 흔히 사용한다. 자석스위치 유형은 붙었을 때 스위치가 OFF이고 떨어졌을 때 ON이 되는 형태와, 붙어 있을 때가 스위치 ON이고 떨어졌을 때가 스위치 OFF가 되는 형태의 2가지 유형이 있다.



실습에서는 붙어 있으면 ON이고 떨어져 있으면 OFF가 되는 자석스위치를 이용한다. 즉 두 쪽이 붙어 있으면 연결단자가 연결되어 ON 상태를, 떨어져 있으면 두 개의 연결 단자가 분리되어 OFF 상태를 가진다.

5.2.2 실습과제

[실습2] 자석스위치를 이용하여 LED 점멸하기

마그네틱스위치를 이용하여 LED를 점멸하는 프로그램이다. 마그네틱 스위치가 붙어 있으면 LED를 OFF하고, 떨어져 있으면 LED를 ON한다.

```
$ sudo nano magnetic_01.c
//=====
// magnetic_01.c
//      LED ON while magnetic ON
//=====
```

```
#include <stdio.h>
#include <wiringPi.h>

#define P_LED    1          // BCM_GPIO #18
#define P_MAG    5          // BCM_GPIO #24

int main(void) {
    int status;

    printf("[LED/MAG testing....]\n");

    if(wiringPiSetup() == -1)
        return 1;

    pinMode(P_LED, OUTPUT);
    pinMode(P_MAG, INPUT);

    printf("P_LED ON while P_MAG ON.....\n");

    status = 0;
    while(1) {
        status = digitalRead(P_MAG);
        if(status==1) {
            digitalWrite(P_LED, HIGH);
            printf("ON.....\n");
        }
        else {
            digitalWrite(P_LED, LOW);
            printf("OFF.....\n");
        }
        delay(10);
    }

    return 0;
}

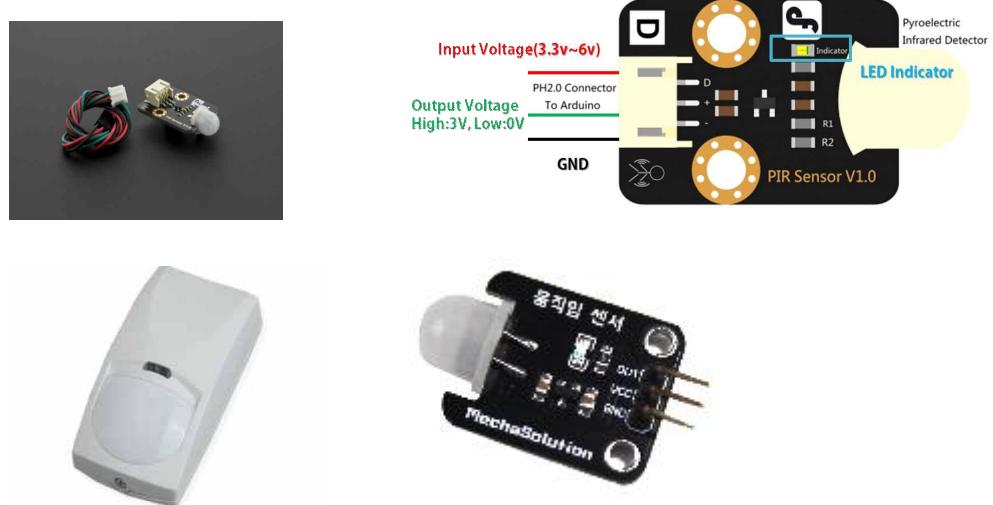
$ make magnetic_01
$ ./magnetic_01
```

5.3 움직임감지센서 제어

움직임 감지 센서는 일반적인 모션감지 센서로 사람이나 생물체의 움직임을 감지하는 센서이다. 본 절에서는 두가지 유형의 움직임 감지 센서를 활용하여 보도록 하겠다.

5.3.1 SEN0171 PIR 센서

SEN0171 PIR(Passive Infrared) 센서는 적외선을 이용하여 움직임을 감지하는 센서이다. 다음과 같이 다양한 형태가 제공된다.



SEN0171 사양은 다음과 같으며, 감지 거리는 7m에 이른다.

- Input Voltage: 3.3 ~ 5V, 6V Maximum
- Working Current: 15uA
- Working Temperature: -20 ~ 85 °C
- Output Voltage: High 3V, low 0V
- Output Delay Time(High Level): About 2.3 to 3 Seconds

- Detection angle: 100 °
- Detection distance: 7 meters
- Output Indicator LED(When output HIGH, it will be ON)
- Pin limit current: 100mA
- Connection Interface: PH2.0-3
- Module size: 30mm × 22mm(1.18"×0.87")

PIR 센서에는 3개의 핀이 있으며, 빨강 선은 Vcc, 검정선은 GND이며, 녹색 선이 움직임을 감지하였을 때 출력되는 신호선이다. 또한 모듈에 있는 LED는 움직임이 감지되었을 때 점멸한다.

회로연결

5V 전압이 나오는 4번 핀과, GND인 6번 핀을 PIR센서 1, 3번에 연결하고, 출력 신호선은 적절한 GPIO 핀에 연결한다.

[실습3] 움직임 감지 센서 I

움직임이 감지되면 LED를 ON하는 프로그램을 작성한다.

```
$ sudo nano motion_01.c
//=====
// motion_01.c
//      LED ON while detecting motion
//=====

#include <stdio.h>
#include <wiringPi.h>

#define P_LED    1          // BCM_GPIO #18
#define P_MOT    5          // BCM_GPIO #24

int main(void) {
    int status;

    printf("[LED/MOT testing....]\n");
    if(wiringPiSetup() == -1)
```

```
return 1;

pinMode(P_LED, OUTPUT);
pinMode(P_MOT, INPUT);

printf("P_LED ON while P_MOT ON.....\n");

status = 0;
while(1) {
    status = digitalRead(P_MOT);
    if(status==1) {
        digitalWrite(P_LED, HIGH);
        printf("ON.....\n");
    }
    else {
        digitalWrite(P_LED, LOW);
        printf("OFF.....\n");
    }
    delay(10);
}

return 0;
}

$ make motion_01
$ ./motion_01
```

5.4 피에조 부저 모듈

5.4.1 피에조 부저 모듈

피에조 부저(SM-1212C)



피에조란 외부의 압력 변화에 전압이 발생하고, 반대로 전압이 걸리면 물리변형이 발생하는 것을 말한다. 전자를 압전 효과, 후자를 역압전 효과라고 하며, 이런 성질을 활용하여 피에조 부저(Piezo Buzzer)는 전기신호를 받아 음파(진동)을 발생시켜 소리를 내도록 하는 디바이스를 말한다. 이 부저 모듈은 인가된 주파수 신호에 따라 출력되는 소리가 달라진다.

주파수 신호를 만들어 내려면, 주파수와 주기간의 개념을 이해하고 신호를 발생시켜야 하며 주파수와 주기와의 관계는 다음과 같다

$$T = 1/f$$

위 식에서 T는 주기(시간)이면 f는 주파수. 예를 들어 위의 표에서 4옥타브 솔 음은 392Hz 이다. 이 주파수는 $T=1/392\text{Hz}=2551.02[\mu\text{sec}]$ 로, 2551.02 μs 의 주기를 가지며 이 주기 마다 파형을 한 번씩 출력하여야만 4옥타브 솔 음을 피에조 부저를 통해 낼 수 있다.

다음의 표는 계이름에 따른 주파수를 정리한 것이다.

계이름	주파수(Hz)	계이름	주파수(Hz)	계이름	주파수(Hz)
도	262	도	523	도	1047
도#	277	도#	554	도#	1109
레	294	레	587	레	1175
레#	311	레#	622	레#	1245
미	330	미	659	미	1319
파	349	파	693	파	1397
파#	370	파#	740	파#	1480
솔	392	솔	784	솔	1568
솔#	415	솔#	831	솔#	1661
라	440	라	880	라	1760
라#	466	라#	932	라#	1864
시	494	시	988	시	1975
				도	2093

5.4.2 softTone.h 라이브러리

피에조 부저 모듈을 제어하는데 유용한 라이브러리 함수는 softTone.h에 선언되어 있다. 이들 함수들의 기능은 다음과 같다.

`extern int softToneCreate(int pin);`

주파수 신호를 출력하도록 핀을 설정하는 함수이다.

`extern void softToneWrite(int pin, int freq);`

특정 주파수의 신호를 해당 핀에 출력하는 함수이다.

`extern void softToneStop(int pin);`

주파수 신호의 출력을 정지하는 함수이다.

5.4.3 실습과제

[실습4] 부저 모듈 제어 I

부저 모듈에 주파수를 변경하면서 출력하여 소리를 출력하는 프로그램이다. 피에조 부저 회로는 소스를 참조하여 구성한다.

```
$ sudo nano buzer_01.c
//=====
// buzer_01.c
//
//=====
#include <stdio.h>
#include <wiringPi.h>
#include <softTone.h>          // *
#define P_BUZ      1           // BCM_GPIO #13

int main(void) {
    int freq;
```

```
if(wiringPiSetup() == -1)
    return 1;

pinMode(P_BUZ, OUTPUT);

softToneCreate(P_BUZ);

freq = 262;                      // Do
while(1) {
    softToneWrite(P_BUZ, freq);   // 주파수 발생
    delay(1000);

    freq += 20;
}

softToneStop(P_BUZ);             // 음 정지
delay(1000);

return 0;
}
```

```
$ make buzer_01
$ ./buzzer_01
```

[응용1] 부저 모듈 제어 III

다음 소스는 동요 ‘학교종’을 연주하는 프로그램이다.

```
$ sudo nano buzer_03.c
//=====
// buzer_03.c
//     deangdeangdeang...
//=====
#include <stdio.h>
#include <wiringPi.h>
#include <softTone.h>

#define P_BUZ 1      // BCM_GPIO #13
```

```

#define Do      262
#define Re      294
#define Mi      330
#define Pa      349
#define Sol     392
#define Ra      440
#define Si      494
#define Do1     523

// 학교종이땡땡땡 음계 및 지속시간
int MusicData[] = {Sol, Sol, Ra, Ra, Sol, Sol, Mi,
                    Sol, Sol, Mi, Mi, Re,
                    Sol, Sol, Ra, Ra, Sol, Sol, Mi,
                    Sol, Mi, Re, Mi, Do};

int DelayTime[] = {400, 400, 400, 400, 400, 400, 800,
                   400, 400, 400, 400, 800,
                   400, 400, 400, 400, 400, 400, 800,
                   400, 400, 400, 400, 800};

int main(void) {
    int i;

    printf("[Music Play...]\n");

    if(wiringPiSetup() == -1)
        return 1;

    pinMode(P_BUZ, OUTPUT);

    softToneCreate(P_BUZ);

    for(i=0; i<24; i++) {
        softToneWrite(P_BUZ, MusicData[i]);
        delay(DelayTime[i]);
    }
    softToneStop(P_BUZ);           // 음 정지
    delay(1000);

    return 0;
}

```

```
$ make buzer_03
$ ./buzer_03
```

5.5 LED 어레이 제어

LED 4개 1조를 제어하는 것에 대해 살펴보자. LED들을 아래의 내용에 따라 회로를 구성한다.

```
#define P_LED_D0      26      // BCM_GPIO 12
#define P_LED_D1      27      // BCM_GPIO 16
#define P_LED_D2      28      // BCM_GPIO 20
#define P_LED_D3      29      // BCM_GPIO 21
```

[실습5] LED 어레이 제어

4개의 LED를 제어하는 프로그램으로, 0부터 15까지의 값을 4개의 LED에 표시하기를 반복한다. 비트필드와 구조체를 활용하여 구현한 것이다.

```
$ sudo nano ledAry_01.c
//=====
// ledAry_01.c
//     LED Array(4 LED) control
//=====

#include <stdio.h>
#include <stdint.h>
#include <wiringPi.h>

#define P_LED_D0      26      // BCM_GPIO #12, LSB
#define P_LED_D1      27      // BCM_GPIO #16
#define P_LED_D2      28      // BCM_GPIO #20
#define P_LED_D3      29      // BCM_GPIO #21, MSB

struct leds {
```

```

        unsigned int    led0      : 1;    // LSB
        unsigned int    led1      : 1;
        unsigned int    led2      : 1;
        unsigned int    led3      : 1;    // MSB
        unsigned int    higher     : 4;    // higher nibble
};

union ucode {
    uint8_t val;
    struct leds pin;           // bit-field
};

void setup(void) {
    pinMode(P_LED_D0, OUTPUT);
    pinMode(P_LED_D1, OUTPUT);
    pinMode(P_LED_D2, OUTPUT);
    pinMode(P_LED_D3, OUTPUT);
}

void out_data(union ucode acode) {
    digitalWrite(P_LED_D0, acode.pin.led0);
    digitalWrite(P_LED_D1, acode.pin.led1);
    digitalWrite(P_LED_D2, acode.pin.led2);
    digitalWrite(P_LED_D3, acode.pin.led3);
}

int main(void) {
    union ucode acode;
    int i;

    printf("[LED Array testing....]\n");

    if(wiringPiSetup() == -1)
        return 1;

    setup();

    while(1) {
        for(i=0; i<16; i++) {
            acode.val = i;
            out_data(acode);
            printf("..... HEX : %0X, DEC : %2d \n",

```

```

        acode.val, acode.val);
delay(1000);
}
}

return 0;
}

```

```

$ gcc -o ledAry_01 ledAry_01.c -lwiringPi
$ ./ledAry_01

```

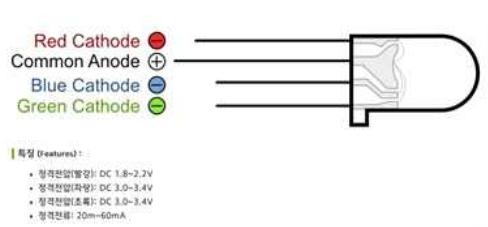
5.6 3색 LED 제어

5.6.1 140C05 RGB LED 모듈

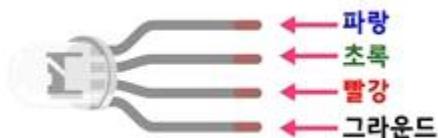


3색 LED는 극성에 따라 커먼 애노드와 커먼 캐소드의 두 가지 유형이 있다.

Common Anode 유형은 핀 배열은 다음과 같으며, 다리가 긴 것이 +이다.



Common Cathode 유형은 다음과 같다.



[실습6] 3색 LED 제어

3색 LED를 각각 ON/OFF하여 몇가지 색상을 표시하는 프로그램이다. 각 LED를 ON하려면 Low 신호를 출력하여야 함을 유의한다.

```
$ sudo nano rgbLed_01.c
//=====
// rgbLed_01.c
//     led ON (Low signal)
//=====

#include <stdio.h>
#include <wiringPi.h>

#define P_LED_R      0          // BCM_GPIO #17
#define P_LED_G      2          // BCM_GPIO #27
#define P_LED_B      3          // BCM_GPIO #22

int main(void) {
    printf("[rgbLED testing....]\n");

    if(wiringPiSetup() == -1)
        return 1;

    pinMode(P_LED_R, OUTPUT);
    pinMode(P_LED_G, OUTPUT);
    pinMode(P_LED_B, OUTPUT);

    // all OFF
    digitalWrite(P_LED_R, HIGH);
```

```
digitalWrite(P_LED_G, HIGH);
digitalWrite(P_LED_B, HIGH);
delay(3000);

while(1) {
    printf("rLED ON.....\n");
    digitalWrite(P_LED_R, LOW);
    digitalWrite(P_LED_G, HIGH);
    digitalWrite(P_LED_B, HIGH);
    delay(2000);

    printf("gLED ON.....\n");
    digitalWrite(P_LED_R, HIGH);
    digitalWrite(P_LED_G, LOW);
    digitalWrite(P_LED_B, HIGH);
    delay(2000);

    printf("bLED ON.....\n");
    digitalWrite(P_LED_R, HIGH);
    digitalWrite(P_LED_G, HIGH);
    digitalWrite(P_LED_B, LOW);
    delay(2000);

    printf("(r+g)LED ON.....\n");
    digitalWrite(P_LED_R, LOW);
    digitalWrite(P_LED_G, LOW);
    digitalWrite(P_LED_B, HIGH);
    delay(2000);

    printf("(r+g+b)LED ON.....\n");
    digitalWrite(P_LED_R, LOW);
    digitalWrite(P_LED_G, LOW);
    digitalWrite(P_LED_B, LOW);
    delay(2000);

    printf("all LED OFF.....\n");
    digitalWrite(P_LED_R, HIGH);
    digitalWrite(P_LED_G, HIGH);
    digitalWrite(P_LED_B, HIGH);
    delay(4000);
}
```

```

        return 0;
}

```

```

$ gcc -o rgbLed_01 rgbLed_01.c -lwiringPi
혹은, $ make rgbLed_01
$ ./rgbLed_01

```

5.7 릴레이 모듈 제어

릴레이(Relay)는 전자석의 원리로 전류가 흐르면 자기장을 형성해 자기력으로 자석을 끌어 당겼다가 전류가 흐르지 않으면 자석을 놓는 원리를 활용한 것으로 스위치 역할로써 사용이 가능하다. 낮은 전압과 전류를 이용하여 더 높은 전압과 전류를 제어하는 곳에 많이 사용된다.

5.7.1 DFR0017 디지털 릴레이 모듈

릴레이 모듈은 구동 전압에 따라, 릴레이 개수에 따라, 구동가능 고압회로에 따라 다양한 유형이 있을 수 있다. 다음의 그림은 1채널과 4채널의 릴레이 모듈의 의양을 보이고 있다.



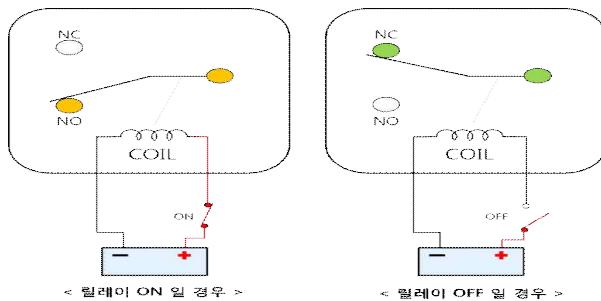
1채널 릴레이 모듈



4채널 릴레이 모듈

릴레이 모듈에는 제어신호 인가와 관련한 3개 단자(5V, GND, DAT)와 고압회로를 연결하기 위한 3개 단자(NC : Normal Closed, COM : common, NO : Normal Open)가 있다.

고압회로 연결방식으로는 Normally Open 방식과 Normally Closed 방식의 2가지가 있다.

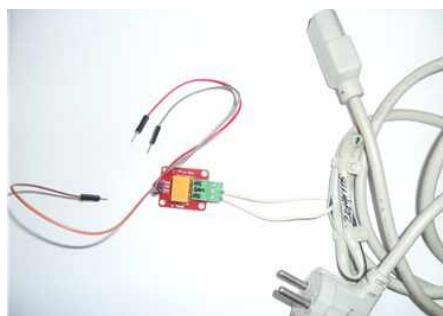


Normally Open 방식은 릴레이의 NO단자와 COM 단자에 고압회로 단자를 연결하는 방식으로, 보통 상태에서 전류가 흐르지 않는 상태이다. 이 경우 릴레이 제어단자에 High 신호를 인가하면 고압회로에 전류가 흐르는 상태가 된다.

Normally Closed 방식은 릴레이의 NC단자와 COM 단자에 고압회로 단자를 연결하는 방식으로, 보통 상태에서 고압회로에 전류가 흐르는 상태이다. 이 경우 릴레이 제어단자에 Low 신호를 인가하면 고압회로에 전류가 흐르지 않는 상태가 된다.

5.7.2 실습과제

다음 그림은 COM 단자와 NO 단자에 고압회로 단자를 연결한 Normally Open 방식을 적용한 예로 실습에 활용한다.



[실습7] 릴레이 모듈 제어 I

릴레이 모듈에 제어신호 HIGH를 인가하였을 때 릴레이를 ON하는 프로그램이다. 제어할 고압회로는 COM과 NO 단자에 연결하여 Normally Open 상태로 구현한다.

```
$ sudo nano relay_01.c
//=====
// relay_01.c
//      Normally Open (COM, NO) ..... ON at High signal
//=====

#include <stdio.h>
#include <wiringPi.h>

#define P_RLY    29           // BCM_GPIO #21

// funciton definition
void run(void) {
    digitalWrite(P_RLY, HIGH);
    printf("ON.....\n");
    delay(5000);

    digitalWrite(P_RLY, LOW);
    printf("OFF.....\n");
    delay(5000);
}

int main(void) {
    printf("[Relay testing....Normally Open]\n");

    if(wiringPiSetup() == -1)
        return 1;

    pinMode(P_RLY, OUTPUT);

    while(1) {
        run();
    }
}
```

```
        return 0;  
    }
```

```
$ make relay_01  
$ ./relay_01
```

[응용2] 릴레이 모듈 제어 II

릴레이 모듈에 제어신호 Low 신호를 인가하였을 때 릴레이를 ON하는 프로그램을 구현한다. 즉, Normally Closed 상태로 배선하는 경우이다. 즉, 제어할 고압회로를 COM과 NC 단자에 연결한다.

```
$ sudo nano relay_02.c  
=====  
// relay_02.c  
//      Normally Closed (NC, COM) ..... ON at LOW signal  
=====  
#include <stdio.h>  
#include <wiringPi.h>  
  
#define P_RLY 29           // BCM_GPIO #21  
  
// funciton definition  
void run(void) {  
    digitalWrite(P_RLY, LOW);  
    printf("ON.....\n");  
    delay(5000);  
  
    digitalWrite(P_RLY, HIGH);  
    printf("OFF.....\n");  
    delay(5000);  
}  
  
int main(void) {  
    printf("[Relay testing....Normally Closed]\n");  
  
    if(wiringPiSetup() == -1)
```

```

        return 1;

pinMode(P_RLY, OUTPUT);

while(1) {
    run();
}

return 0;
}

```

```

$ make relay_02
$ ./relay_02

```

5.8 초음파센서 제어

5.8.1 HC-SR04 초음파 센서 모듈

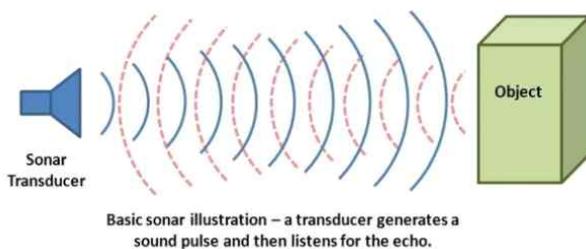
초음파 거리측정(HC-SR04) 모듈은 2cm~400cm의 비접촉 거리측정이 가능하며, 정확도는 3mm까지이다. 이 모듈은 초음파 송신기, 수신기 및 제어 회로로 구성되며, 동작 주파수는 40Hz, 측정각은 15도까지 가능하다. 대상 물체의 면적은 0.5 평방 미터 이상으로, 물체의 면은 가능한 한 매끄러운 것을 권고한다. 외양은 다음 그림과 같다.



연결 단자는 전원을 위한 Vcc, Gnd와 트리거 펄스용 입력(Trig) 단자, 수신기가 수신한 반향 신호를 출력(Echo)하는 단자가 있다.

기본적인 동작 원리는 다음과 같다.

- IO 트리거 신호로 최소한 10Us 동안의 High 신호를 사용한다.
- 모듈은 자동으로 8개의 40 kHz 신호를 전송하고, 되돌아오는 펄스 신호가 있는지를 검출한다.
- 되돌아 온 신호가 있으면, High 레벨 신호의 지속시간은 초음파를 보내고 나서 돌아오는 데까지 걸리는 시간이다. 따라서 대상까지의 거리는 (High 지속시간 × 음속(340M/S)) / 2로 얻을 수 있다.

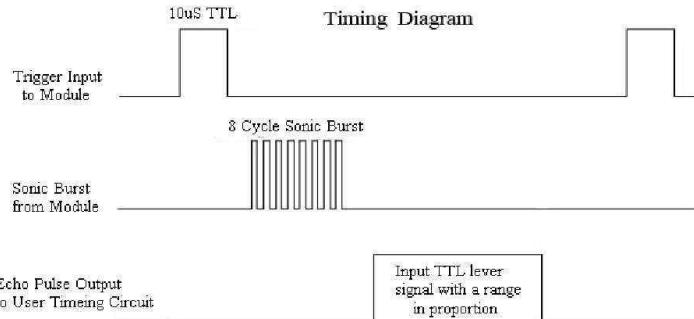


5.8.2 초음파 거리센서 제어방법

초음파 거리 센서의 제어방법은 아래의 타이밍 도를 참조한다.

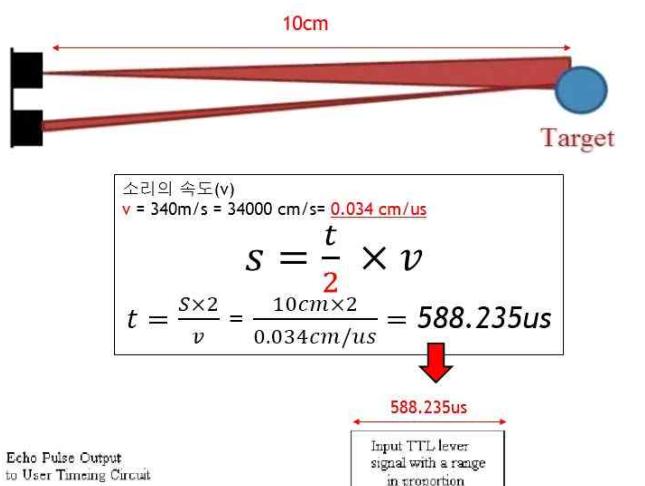
거리측정을 시작하려면 초음파 거리 센서의 트리거 입력에 10uS 펄스를 인가하면 된다. 그런 후에, 모듈은 40kHz 8주기의 초음파 버스트 신호를 보내고 반향되는 에코 신호를 수신하여 전달한다. 에코 신호는 대상 객체와의 거리에 비례하는 펄스 폭을 가진다. 트리거 신호 전송과 에코 신호 수신 사이의 시간 간격을 통해 거리를 계산할 수 있다. 거리(cm) = uS/58, 거리(inch) = uS/148, 또는 거리(m) = High 레벨 시간 * 속도 (340M/S) / 2 등의 계산식을 활용하여 거리를 계산할 수 있다.

주의할 것은 에코 신호를 트리거 신호로 오인하는 것을 방지하기 위해 60ms 이상의 측정 주기를 사용할 것을 권고한다.



초음파 거리센서를 이용한 거리 계산은 거리 = 속도 x 시간 이므로, 거리 = 소리의 속도(0.034 cm/us) x 에코 펄스시간이 된다.

아래 그림은 역으로 에코펄스의 시간을 구해보았다. 만약 장애물이 10cm라 가정했을 때, 에코 펄스는 588.235us가 된다는 것을 알 수 있다. 소리의 속도는 340m/s = 0.034 cm/s 즉, 1cm가는데 걸리는 시간은 29us인데, 왕복이므로 58us가 된다. 10cm이므로 580us이 소요된다.



5.8.3 wiringPi.h 라이브러리

초음파 센서 제어에서 유용한 시간지연 및 시간단위 반환 함수들이 wiringPi.h 헤더파일에 선언되어 있다. 이들 함수들의 기능은 다음과 같다.

시간지연함수

```
void delay(unsigned int howLong);           // millisec 단위  
void delayMicroseconds(unsigned int howLong); // microsec 단위
```

단위시간 반환함수

```
unsigned int millis(void);      // 현 시각을 millisec 단위로  
unsigned int micros(void);     // 현 시각을 microsec 단위로
```

5.8.4 실습과제

[실습8] 초음파센서 제어

초음파센서 모듈을 이용하여 거리를 측정하는 프로그램이다. trig, echo 단자를 적절히 연결한다.

```
$ sudo nano usonic_01.c  
=====  
// usonic_01.c  
//      HC-SR04 module  
=====  
#include <stdio.h>  
#include <wiringPi.h>  
  
#define P_USO_TRIGGER      5      // Trigger Pulse, BCM_GPIO #24  
#define P_USO_ECHO          4      // Echo Pulse, BCM_GPIO #23  
  
int main(void) {  
    long start, end;  
    int duration;  
    float distance;  
  
    if(wiringPiSetup() == -1)
```

```
return 1;

pinMode(P_USO_TRIG, OUTPUT);
pinMode(P_USO_ECHO, INPUT);

printf("[UltraSonic testing....]\n");
while(1) {
    // sending 2us signal
    digitalWrite(P_USO_TRIG, LOW);
    delayMicroseconds(2);

    // sending 10microSec high signal
    digitalWrite(P_USO_TRIG, HIGH);
    delayMicroseconds(10);           // 10us
    digitalWrite(P_USO_TRIG, LOW);

    // wait for burst signal, 8x40kHz=8x25us=200us
    delayMicroseconds(200);

    // receiving echo signal
    while(digitalRead(P_USO_ECHO)==LOW) // wait until 1
        ;
    start = micros();                // us unit

    while(digitalRead(P_USO_ECHO)==HIGH) // wait until 0
        ;
    end = micros();

    // get High level duration(us)
    duration = end - start;

    // calculate the distance(cm)
    distance = duration / 58.;

    if(distance > 400)              // invalid distance
        printf("");
    else {
        printf("%d,%d duration:%d ... ",
               start, end, duration);
        printf("Distance : %.2f cm\n", distance);
    }
}
```

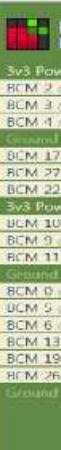
158 라즈베리파이 기본 임베디드시스템 응용

```
// delay for measurement cycle, > 60milliSec  
delay(100);  
}  
}
```

```
$ make usonic_01  
$ ./usonic_01
```

참고자료

- [1] “아두이노를 활용한 공학기초 다지기”, 김한종저, 영출판, 2017
- [2] 3색 LED 제어
<http://cafe.daum.net/aduino/R8N2/4?q=%BE%C6%B5%CE%C0%CC%B3%EB%203%BB%F6%20LED>
- [3] 움직임 감지 센서 <http://emaru.tistory.com/18>
- [4] 마이크로웨이브 움직임 감지 센서
https://www.dfrobot.com/wiki/index.php/MicroWave_Sensor_SKU:_SEN019
- [5] 릴레이 <http://oogundam.tistory.com/438>



제6장 모터 제어

본 장에서는 DC 모터, 스텝핑 모터, 서보 모터의 제어에 대해 살펴본다.

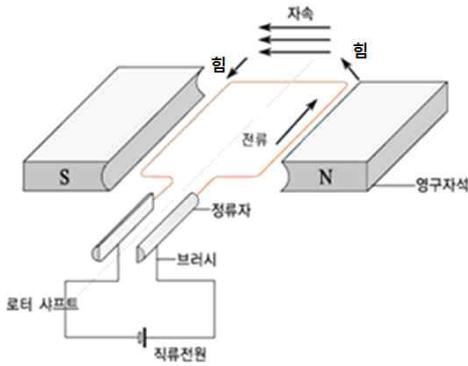
6.1 DC 모터 제어

Half Bridge 회로를 사용하여 DC 모터를 제어하는 방법에 대해 알아보자.

6.1.2 DC 모터

DC 모터는 직류 전동기라고 불리며 건전지와 같은 직류 전원에서 동작하는 모터이다. 가해주는 전압에 따라서 속도가 달라지는 특징을 가지고 있는 이 모터는 작은 크기와 간단한 사용법으로 인하여 휴대용 선풍기나 전기면도기, 드라이기 등의 일상생활에서 널리 사용된다. 모터는 자석이 같은 극끼리 서로 밀어내고, 다른 극끼리는 끌어당기는 성질을 이용한 것으로 원형으로 자석을 배열하고 그 안에 전기를 흘려주어 전자석을 만들면 자석의 힘을 회전 운동으로 바꿀 수 있다.

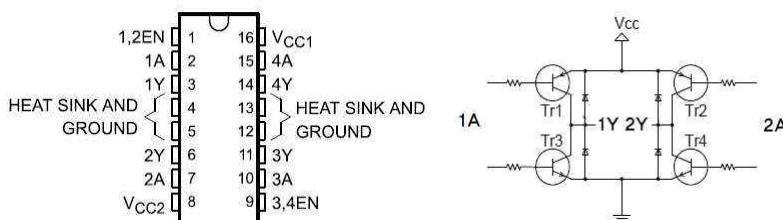
DC모터의 구동 원리는 자기장 속에 전류가 흐르면 전류가 흐르는 도선은 힘을 받아 움직인다는 사실을 이용한 것으로 자기장의 방향은 N극에서 S극으로 향하며, 전류의 방향은 +극에서 -극으로 향한다. 자기장의 방향과 전류의 방향에 따라 플레밍의 원손 법칙에 의해 도선이 받게 되는 힘의 방향이 결정된다. 다음 그림에서 보듯이 자석의 S극 근처에 있는 도선에 전류와 자기장의 방향을 플레밍의 원손 법칙을 적용해 보면 그 도선은 아래쪽으로 힘을 받게 되고 N극 근처의 도선에도 적용해보면 위쪽으로 힘을 받게 되어 도선은 반 시계 방향으로 회전 운동을 하게 된다. 전류가 반대방향으로 흐른다면 도선은 시계방향으로 회전한다.



6.1.2 SN754410 Half Bridge 드라이버

앞서의 모터 구동 원리에서 살펴보았듯이 전류의 방향을 바꾸면 모터의 회전방향을 바꿀 수 있다. 이를 위한 회로가 Half bridge 회로이며, IC 형태로 제공된다. 이를 위한 IC로 SN754410가 있으며, L293D IC도 호환된다.

SN754410 Half bridge IC에는 4개의 반 Half bridge 회로가 내장되어 있으며, 2개의 반 Half bridge 회로를 사용하여 한 채널의 Half bridge회로를 구성할 수 있다. 따라서, 하나의 IC로 2채널의 Half bridge 회로를 구성할 수 있다. SN754410 IC의 핀 배열과 핀 용도는 다음 그림과 같다.

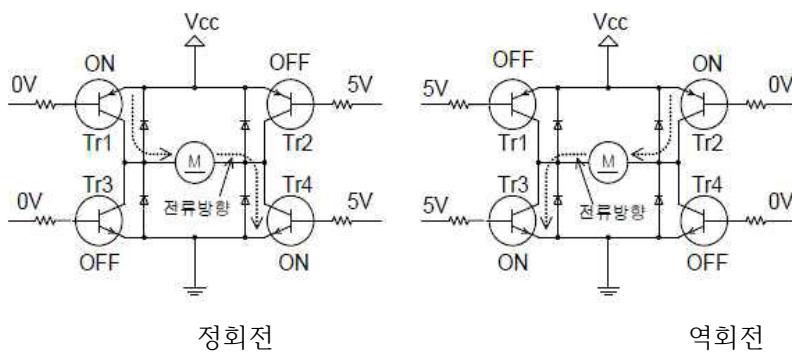


IC의 좌측 핀들을 활용한 한 채널의 Half bridge 회로를 구성하는 것에 대해 살펴보면 다음과 같다. 1A와 2A핀은 트랜지스터의 베이스에 연결되어 모터의 회전 방향을 결정하는 핀으로, GPIO 핀과 연결한다. 1Y와 2Y핀은 브리지회로의 출력단자로 DC 모터에 연결한다. 1,2EN핀은 한 채널의 Half bridge 회로의 인에이블 단자로, High 신호가 인가될 경우 Half bridge 회로가 동작하나, Low 신호가 인가되

는 경우 동작 하지 않는다. 아울러 이 인에이블 단자에 PWM 신호를 인하하면 모터의 회전 속도를 조절하는 것이 가능하다. Vcc1핀은 IC 로직의 전원 공급용으로 5V를 인가한다. 4, 5, 12, 13의 4개 핀은 GND에 연결한다. Vcc2핀은 모터를 구동하기 위한 전원을 공급하는 핀으로, 모터구동에 필요한 전압을 인가하면 되며 36V까지 인가가능하다. 나머지 우측의 핀들은 두 번째 채널의 Half bridge 회로를 구성하는데 사용될 수 있으며, 그 기능은 앞에서 살펴본 바와 같다.

DC 모터의 정회전과 역회전

다음 그림과 같이 pnp, npn 트랜지스터 각 2개로 H-브리지를 구성하면 단일 전원으로 DC 모터를 정회전/역회전시킬 수 있다.



위의 두 트랜지스터는 pnp형이고 아래 두 트랜지스터는 npn형이다. 그림의 좌측은 정회전할 때 신호로, Tr1은 pnp형이므로 베이스에 0V를 가해 Tr1을 켜고, Tr4는 npn형이므로 베이스에 5V를 가해 Tr4를 켜고 Tr2와 Tr3는 꺼진다. 따라서 전류는 그림에서 보인 방향으로 흐르게 되어 모터가 정회전한다. 반면, 우측은 역회전할 때 신호로 Tr1과 Tr4를 끄고 Tr2와 Tr3를 켜게 되고, 전류는 정회전과는 반대로 흐르게 되어 모터는 역회전을하게 된다.

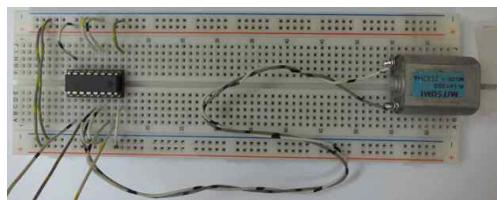
Half bridge 회로의 1A, 2A, 1,2EN 핀에 인가되는 신호와 동작 상태는 다음 표와 같다.

164 라즈베리파이기반 임베디드시스템응용

IA	2A	EN	동작
HIGH	LOW	HIGH	정회전
LOW	HIGH	HIGH	역회전
LOW	LOW	HIGH	브레이크
X	X	LOW	정지

회로연결

실습을 위해 Half bridge 회로와 DC 모터로 구성한 회로의 그림은 다음과 같다.



6.1.3 실습과제

dcmotor.h 사용자 라이브러리

DC 모터 제어를 위한 사용자 라이브러리를 다음과 같이 구현한다.

```
$ sudo nano dcmotor.h
//=====
// dcmotor.h
//      DC motor + Hbridge(SN754410)
//=====
#ifndef __DCMOTOR_H__
#define __DCMOTOR_H__

#include <stdio.h>
#include <wiringPi.h>

#define P_BRG_EN12      27      // BCM_GPIO #16
#define P_BRG_1A         28      // BCM_GPIO #20
#define P_BRG_2A         29      // BCM_GPIO #21

void setPinMode(void);
```

```
void forward(void);
void backward(void);
void brake(void);
void stop(void);

void setPinMode(void) {
    pinMode(P_BRG_EN12, OUTPUT);
    pinMode(P_BRG_1A, OUTPUT);
    pinMode(P_BRG_2A, OUTPUT);
}

void forward(void) {
    printf("forward....\n");

    digitalWrite(P_BRG_EN12, LOW);
    digitalWrite(P_BRG_1A, HIGH);
    digitalWrite(P_BRG_2A, LOW);

    digitalWrite(P_BRG_EN12, HIGH);      // run
}

void backward(void) {
    printf("backward....\n");

    digitalWrite(P_BRG_EN12, LOW);
    digitalWrite(P_BRG_1A, LOW);
    digitalWrite(P_BRG_2A, HIGH);

    digitalWrite(P_BRG_EN12, HIGH);      // run
}

void brake(void) {
    printf("brake....\n");

    digitalWrite(P_BRG_EN12, LOW);
    digitalWrite(P_BRG_1A, LOW);
    digitalWrite(P_BRG_2A, LOW);

    digitalWrite(P_BRG_EN12, HIGH);      // run
}

void stop(void) {
```

```
    printf("stop....\n");

    digitalWrite(P_BRG_EN12, LOW);
}

#endif
```

[실습1] DC 모터 제어

DC 모터의 회전방향을 변경하는 프로그램으로, 정회전하다 역회전하기를 3회 반복하는 프로그램이다.

```
$ sudo nano dcMotor_01.c
//=====
// dcMotor_01.c
//      DC motor + Hbridge(SN754410)
//=====

#include <stdio.h>
#include <wiringPi.h>

#include "dcmotor.h" // user library

int main(void) {
    int i;

    if(wiringPiSetup() == -1)
        return 1;

    setPinMode();

    for(i=0; i<3; i++) {
        forward();
        delay(3000);

        brake();
        delay(1000);

        backward();
        delay(3000);
```

```

        brake();
        delay(1000);
    }

    stop();
    delay(3000);

    return 0;
}

```

\$ make dcMotor_01
\$./dcMotor_01

[실습2] DC 모터의 속도 제어

Half bridge회로의 Enable 단자에 PWM 신호를 인가하여 속도를 가변하는 프로그램으로, 점점 속도 높이다가 최대속도에 이르러서 역으로 속도를 줄여가는 프로그램이다.

```

$ sudo nano dcMotor_02.c
//=====
// dcMotor_02.c
//      DC motor + Hbridge(SN754410)
//=====

#include <stdio.h>
#include <wiringPi.h>
#include <softPwm.h>

#include "dcmotor.h"

void forwardPWM(int i) {
    printf("forward....\n");
    digitalWrite(P_BRG_EN12, LOW);
    digitalWrite(P_BRG_1A, HIGH);
    digitalWrite(P_BRG_2A, LOW);

    softPwmWrite(P_BRG_EN12, i);           // run
}

```

```
void stopPWM(void) {
    printf("stop....\n");
    softPwmWrite(P_BRG_EN12, LOW);
}

int main(void) {
    int i;

    if(wiringPiSetup() == -1)
        return 1;

    setPinMode();
    softPwmCreate(P_BRG_EN12, 0, 100);

    for(i=0; i<=100; i+=5) {
        printf("%d ..... \n", i);
        forwardPWM(i);           //
        delay(3000);
    }

    for(i=100; i>=0; i-=5) {
        printf("%d ..... \n", i);
        forwardPWM(i);           //
        delay(3000);
    }

    stopPWM();
    delay(3000);

    return 0;
}
```

```
$ make dcMotor_02
$ ./dcMotor_02
```

[실습3] DC 모터 제어

버튼 스위치의 입력에 따라 DC 모터의 회전방향을 바꾸는 프로그램이다.

```
$ sudo nano dcMotor_03.c
//=====
// dcMotor_03.c
//      DC motor + Hbridge(SN754410) + BTN
//=====

#include <stdio.h>
#include <wiringPi.h>

#include "dcmotor.h"           // user library

#define P_BTN      5           // BCM_GPIO #24

int main(void) {
    int i;

    if(wiringPiSetup() == -1)
        return 1;

    pinMode(P_BTN, INPUT);
    setPinMode();

    while(1) {
        if(digitalRead(P_BTN)==0)
            forward();
        else
            backward();
    }

    return 0;
}

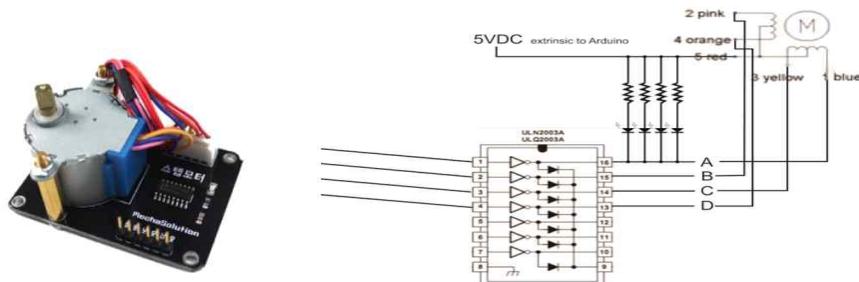
$ make dcMotor_03
$ ./dcMotor_03
```

6.2 스텝핑 모터 제어

스테핑 모터는 그 이름에서도 암시하는 것처럼, 한 스텝만큼의 각도 단위로 회전량을 제어할 수 있는 특성을 가진 모터를 말한다. 이용자가 모터의 회전각도와 회전 속도, 회전방향을 제어할 수 있기 때문에 다양한 기기에 응용되고 있다.

9.2.1 28BYJ-48 스텝핑 모터

28BYJ-48 스텝핑 모터는 감속장치가 내장되어 있으며, 4 페이즈(phase)로 제어할 수 있다. 한 회전은 약 2040스텝으로 구성되며, 각 스텝의 각도는 $5.625/64$ 도이며, 최대속도는 14rpm이다. 스텝핑 모터 모듈의 회로도는 다음과 같다.



6.2.2 ULN2003A 드라이버

ULN2003A 드라이버의 각 phase에 대한 4개 핀에 GPIO 핀을 하나씩 연결한다, 각 phase는 A, B, C, D라 할 때 스텝핑 모터를 여러 방식으로 제어할 수 있다. 4 phase의 스텝핑 모터 구동 방법으로 4 phase 4 step으로 할 경우 다음의 방법들이 있을 수 있다.

구분	A	B	C	D
step 1	1	0	0	0
step 2	0	1	0	0
step 3	0	0	1	0
step 4	0	0	0	1

구분	A	B	C	D
step 1	1	1	0	0
step 2	0	1	1	0
step 3	0	0	1	1
step 4	1	0	0	1

4 phase 8 step으로 다음과 같이 제어할 수도 있다.

구분	A	B	C	D
step 1	1	1	0	0
step 2	0	1	0	0
step 3	0	1	1	0
step 4	0	0	1	0
step 5	1	0	1	1
step 6	0	0	0	1
step 7	1	0	0	1
step 8	1	0	0	0

6.2.3 실습과제

[실습4] 스텝모터 제어 I

회전방향은 시계방향(정회전)으로 하되, 다양한 모드로 구동하는 것을 테스트 한다.

```
$ sudo nano stepMotor_01.c
=====
// stepMotor_01.c
//      28BYJ-48 Step Motor + ULN2003A Driver
//=====

#include <stdio.h>
#include <wiringPi.h>

#define P_ULN_IN1 26    // BCM_GPIO #12
#define P_ULN_IN2 27    // BCM_GPIO #16
#define P_ULN_IN3 28    // BCM_GPIO #20
#define P_ULN_IN4 29    // BCM_GPIO #21

void setPinMode(void) {
    pinMode(P_ULN_IN1, OUTPUT);
    pinMode(P_ULN_IN2, OUTPUT);
    pinMode(P_ULN_IN3, OUTPUT);
    pinMode(P_ULN_IN4, OUTPUT);
}
```

```
void run(int in1, int in2, int in3, int in4) {
    digitalWrite(P_ULN_IN1, in1);
    digitalWrite(P_ULN_IN2, in2);
    digitalWrite(P_ULN_IN3, in3);
    digitalWrite(P_ULN_IN4, in4);
    delay(10); // speed
}

int main(void) {
    if(wiringPiSetup() == -1)
        return 1;

    setPinMode();

    while(1) {

        /*
         * // 2phase 4step .... X
         * run(0, 0, 1, 0);
         * run(0, 1, 1, 0);
         * run(0, 1, 0, 0);
         * run(0, 0, 0, 0);
        */

        /*
         * // 4phase 4step I .... O
         * run(1, 0, 0, 0);
         * run(0, 1, 0, 0);
         * run(0, 0, 1, 0);
         * run(0, 0, 0, 1);
        */

        /*
         * // 4phase 4step II ... O
         * run(1, 1, 0, 0);
         * run(0, 1, 1, 0);
         * run(0, 0, 1, 1);
         * run(1, 0, 0, 1);
        */
    }
}
```

```

    // 4phase 8step ... O
    run(1, 1, 0, 0);
    run(0, 1, 0, 0);
    run(0, 1, 1, 0);
    run(0, 0, 1, 0);
    run(0, 0, 1, 1);
    run(0, 0, 0, 1);
    run(1, 0, 0, 1);
    run(1, 0, 0, 0);

}

return 0;
}

```

```

$ make stepMotor_01
$ ./stepMotor_01

```

[실습5] 스텝 모터 제어 II

4phase-4step 방식을 취하되, 회전방향을 시계방향-반시계방향을 반복하는 프로그램이다.

```

$ sudo nano stepMotor_02.c
//=====
// stepMotor_02.c
//      28BYJ-48 Step Motor + ULN2003A Driver
//      CW-CCW-CW-..., using 4phase-4step
//=====
#include <stdio.h>
#include <wiringPi.h>

#define P_ULN_IN1 26    // BCM_GPIO #12
#define P_ULN_IN2 27    // BCM_GPIO #16
#define P_ULN_IN3 28    // BCM_GPIO #20
#define P_ULN_IN4 29    // BCM_GPIO #21

void setPinMode(void) {

```

```
pinMode(P_ULN_IN1, OUTPUT);
pinMode(P_ULN_IN2, OUTPUT);
pinMode(P_ULN_IN3, OUTPUT);
pinMode(P_ULN_IN4, OUTPUT);
}

void run(int in1, int in2, int in3, int in4) {
    digitalWrite(P_ULN_IN1, in1);
    digitalWrite(P_ULN_IN2, in2);
    digitalWrite(P_ULN_IN3, in3);
    digitalWrite(P_ULN_IN4, in4);
    delay(10); // speed
}

int main(void) {
    int min=0;
    int max=200;
    int i;

    if(wiringPiSetup() == -1)
        return 1;

    setPinMode();

    while(1) {
        for(i=0; i<max; i++) { // CW
            run(1, 1, 0, 0);
            run(0, 1, 1, 0);
            run(0, 0, 1, 1);
            run(1, 0, 0, 1);
            printf("CW : %d\n", i);
        }

        for( ; i>=min; i--) { // CCW
            run(0, 0, 1, 1);
            run(0, 1, 1, 0);
            run(1, 1, 0, 0);
            run(1, 0, 0, 1);
            printf("CCW : %d\n", i);
        }
    }
}
```

```

        return 0;
}

```

```

$ make stepMotor_02
$ ./stepMotor_02

```

6.3 서보 모터 제어

6.3.1 Micro servo SG90 모듈

Micro servo SG90 모듈은 소형 RC 기기에 사용가능한 미니 경량 마이크로 서보 모터이다. 오렌지색 선에 PWM 신호를 인가하여 구동하게 되며, 구동 가능한 각도는 180도 정도이다.

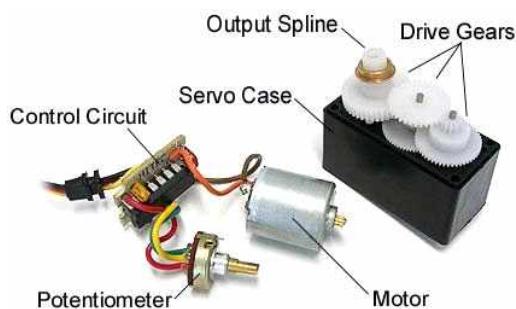
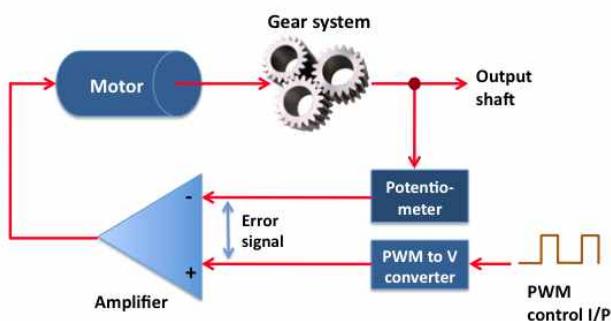


서보 모터는 DC 모터의 일종으로 기어, 모터의 방향 및 위치를 제어하기 위한 회로가 장착되어 있는 모터이다. 서보 모터는 정교한 각 위치를 제어가 가능하기 때문에 로봇이나 RC 카 등 모터의 위치를 제어하기 위한 분야에서 사용된다. 일반적인 서보 모터는 회전범위가 180도이지만, 어떤 것들은 360도 혹은 그 이상을 회전하기도 한다.

서보 모터의 움직임 제어를 위해서 출력 샤프트의 위치 정보는 트랜스듀서를 사용하여 즉각적으로 제어회로에 공급되는데, 이를 위한 가장 간단한 방법은 가변저항

을 출력 셔프트나 기어트레인에 붙이는 것이다. 제어 회로는 가변저항으로부터 오는 피드백 신호(샤프트의 현재 위치정보)를 제어 입력신호(샤프트가 회전할 위치 정보)와 비교하여 차이(에러신호)가 있다면 DC 모터를 회전하여 그 차이를 줄여준다. 원하는 위치로 모터가 정확하게 회전하였을 경우 에러 신호는 0이 된다.

다음 그림은 일반적인 서보 모터의 동작 원리를 표현한 것이다.

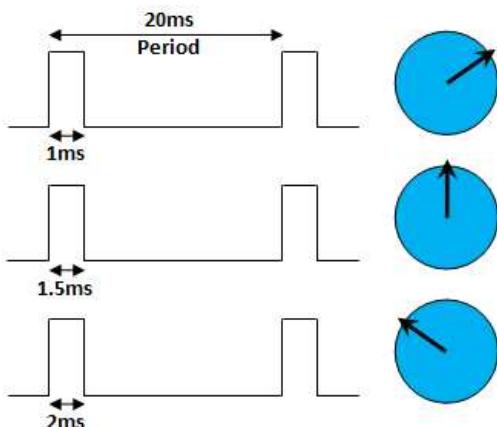


서보 모터로의 제어 입력신호는 PWM 신호로 일반적으로 50Hz를 사용한다. 이것은 펄스가 매 20ms마다 반복되어야 한다는 것을 의미한다. 펄스의 너비는 출력 셔프트의 각 위치를 결정한다. 서보 모터 내에 있는 전자회로는 PWM 신호를 출력 전압신호로 변환하여 준다. 이 출력 전압신호는 가변저항으로부터 받은 피드백 전압과 비교되어, 만약 두개의 값에 차이가 있다면 차이가 0이 될 때까지 적당한 위치로 모터를 회전시킨다.

서보 모터는 전원공급용 Vcc와 Gnd 단자와 PWM 제어신호 입력용의 제어신호 단자 하나가 제공된다. 5V에서 동작하며 180도의 회전이 가능하다.

서보 모터의 제어

일반적인 펄스폭은 1.0에서 2.0ms 사이이다. 표준 서보 모터에 있어 1.0ms에서 1.5ms의 펄스폭은 시계방향으로 모터를 회전시키며, 1.5에서 2.0ms는 반시계방향으로 회전시킨다. 1.5ms 펄스폭은 서보 모터를 중앙으로 회전시킨다. 이러한 펄스폭은 모델에 따라 다를 수 있다. 다음 그림은 펄스폭에 따른 회전위치를 보여준다.



다음 표는 사용하는 서보 모터의 각 위치에 따른 펄스폭을 표현한 것이다. 여기서 펄스의 반복주기는 50Hz(20ms)임을 유의한다.

Pulse width (ms)	Angular position (° CCW)
0.7	0 (min)
1.1	45
1.5	90
1.9	135
2.3	180 (max)

6.3.2 PWM 신호 생성에 의한 제어

사용하는 서보 모터에 적합한 PWM 신호를 생성하고, 이를 인가하여 서보 모터를 구동하는 방법에 대해 살펴본다.

PWM 신호생성과 관련한 함수는 다음과 같다.

pwmSetMode (int mode);

PWM_MODE_BAL(balance), PWM_MODE_MS(mark:space)

pwmSetRange (unsigned int range);

pwmSetClock (int divisor);

WiringPi는 PWM 신호 생성시 기본적으로 Balanced 모드이며, Mark:Space 모드로 설정해야한다. 즉 PWM_MODE_MS로 모드 설정한다.

PWM 주파수는 다음 식으로 계산할 수 있고, 이것이 서보 모터 주기와 일치하도록 clock과 range를 설정한다.

$$P W M F R E Q U E N C y = 19.2 [MHz] / C L O C K / R a N g e$$

19.2MHz이란 Raspberry Pi의 PWM이 있는 베이스 클럭 주파수이며, 베이스 클럭을 clock으로 나눈 값 ($= 19.2 [MHz] / C L O C K$)은 PWM 카운터의 주기를 나타내며 이 주기에서 PWM 카운터가 1 씩 증가된다. range는 PWM 분해능에 영향을 주는 값이다.

clock과 range를 SG90 서보 모터의 사양에 맞게 결정하기 위해,
 $CLOCK = 400$, $R a N g e = 1024$ 를 대입하면 다음과 같다.

PWM 카운터 주기

$$: 19.2 [MHz] / C L O C K = 19.2 * 10^6 / 400 = 48 [KHz] \approx 0.0208 [MS]$$

PWM 주파수

$$\begin{aligned}
 & : 19.2 [MHz] / C L O C K / R a N g e \\
 & = 19.2 * 10^6 / 400 / 1024 \\
 & = 46.875 [Hz] \approx 21.3 [MS]
 \end{aligned}$$

SG90 서보 모터의 PWM 신호 주기는 50Hz, 즉 20ms이다. 생성된 PWM 주파수는 대략 비슷하다.

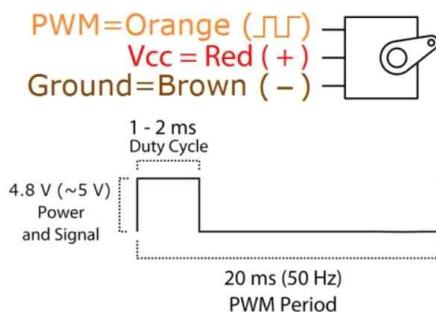
다음 서보 모터를 끝에서 끝까지 회전 값을 계산해 보면 다음과 같다.

SG90의 사양에서 구동가능한 각에 대한 펄스 폭은 0.5 ~ 2.4ms이므로,

$$48 [KHz] * 0.5 [MS] = 48000 [Hz] * 0.0005 [s] = 24$$

$$48 [KHz] * 2.4 [MS] = 48000 [Hz] * 0.0024 [s] = 115.2 \approx 115$$

이 값을 pwmWrite() 함수 호출시 인자로 넘겨주면 서보 모터의 좌우로 한계까지 회전가능하다.



Position "0" (1.5 ms pulse) is middle, "90" (~2 ms pulse) is all the way to the right, "-90" (~1 ms pulse) is all the way to the left.

6.3.3 실습과제

[실습6] 서보 모터 제어 I

PWM 신호의 브리티비를 변경하면서 서보 모터를 구동하는 프로그램이다.

```
$ sudo nano servo_01.c
```

```

//=====
// servo_01.c
//      Microservo SG90 module
//
//=====

#include <stdio.h>
#include <stdlib.h>
#include <wiringPi.h>
#include <softPwm.h>

#define P_SVM    26           // BCM_GPIO #12

int main(void) {
    int i;

    if(wiringPiSetup() == -1)
        exit(1);

    pinMode(P_SVM, SOFT_PWM_OUTPUT);

    softPwmCreate(P_SVM, 0, 200);

    printf("[SG90 Servo Motor testing.....]\n");
    while(1) {

        for(i=0; i<=100; i+=2) {
            printf("..... pos : %d\n", i);
            softPwmWrite(P_SVM, i);
            delay(1000) ;
        }
    }

    return 0;
}

$ make servo_01
$ sudo ./servo_01

```

실행 결과를 관찰한 결과, 6이 극동이면, 30은 극서. 6보다 작은 경우 여러 바퀴

회전, 기타의 범위에서는 무반응, 대체로 8은 좌회전 한계, 30은 우회전 한계로 관찰되었다.

참고자료

[1] DC 모터

<http://blog.naver.com/PostView.nhn?blogId=qpyou1234&logNo=221022966226>

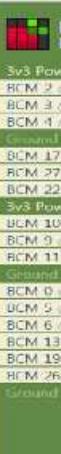
[2] 하프 브리지 회로(L293D)

<http://blog.naver.com/PostView.nhn?blogId=k4264639&logNo=221093300394>

[3] 스테핑 모터 <http://micropilot.tistory.com/2914>

[4] 서보 모터

<http://blog.naver.com/PostView.nhn?blogId=ubicomputing&logNo=150157476790>



제7장 도트매트릭스 및 키패드

본 장에서는 8x8 DotMatrix 모듈, 4x4 Keypad 모듈을 구동하기 위한 동적 제어 방법에 대해 살펴본다. 또한 CLCD 모듈의 사용에 대해 살펴본다.

7.1 8x8 도트매트릭스

7.1.1 정적 제어 및 동적 제어

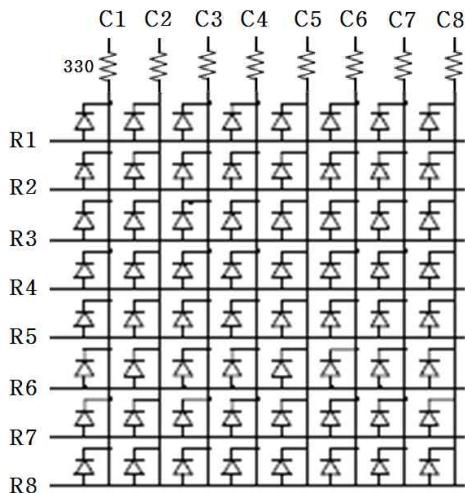
다수의 기본 소자들이 행과 열로 배치되어 있는 입출력 모듈을 제어하는 방법은 정적 제어 방법과 동적 제어 방법으로 나누어 볼 수 있다.

정적 제어(static control) 방법은 지금까지 입출력 모듈들을 제어한 방법이기도 하다. 즉, 버튼 모듈, LED 모듈 등에서 각 버튼, LED 당 하나씩 MCU의 입출력 핀을 할당하고, 각 입출력 핀에 1비트의 데이터를 출력 혹은 입력함으로써 디바이스를 제어하는 방법이다. 이러한 제어 방법은 디바이스의 제어가 단순하다는 장점은 있으나, 키보드나 LED 전광판과 같이 제어해야 할 소자가 다수인 경우 해당 소자 개수만큼의 입출력 GPIO 핀이 요구되는데, 이는 한정된 GPIO 핀을 제공하는 현실에서 제약요인으로 작용할 수 있다.

반면, 동적 제어(dynamic control) 방법은 다수의 기본 디바이스로 구성된 입출력 장치를 보다 적은 입출력 핀으로도 제어할 수 있는 방법이다. 동적 제어 방법은 입출력 디바이스를 구성하고 있는 기본 소자 모두를 동시에 제어하는 것이 아니라 특정 시점에서 소자의 일부만을 제어하되, 짧은 시간차를 두어 또 다른 일부의 소자를 제어하는 방식으로 일정 시간 후 모든 소자를 제어하는 방법이다. 예로 출력 장치의 경우 인간 시각 체계의 잔상 효과를 이용하는 방법이기도 하다. 이러한 동적 제어 방법은 해당 디바이스를 제어하는데 요구되는 입출력 핀의 수를 줄일 수 있으므로 한정된 입출력 핀만을 제공하는 환경에서 장점이 될 수 있으나, 소프트웨어적으로 제어하는데 신경을 써야 한다.

7.1.2 DotMatrix 모듈

DotMatrix 모듈은 다음 그림과 같이 LED가 2차원 평면으로 배열된 모듈이다. 이 DotMatrix 모듈을 온전히 제어하기 위해서는 행과 열을 위해 각각 8개의 GPIO 핀이 필요하며, 동적 제어방식으로 구동하여야 한다.



DotMatrix 모듈의 동적 제어

DotMatrix 모듈을 제어하기 위해서는 열 Cn중 활성화하고자하는 하나의 열에 대해서만 Low 신호를 인가하고, Rn에 표시할 패턴 데이터를 출력하는 식으로 제어한다. 실질적으로 한 시점에서 특정 열에 대한 LED 패턴만 출력하지만, 이와 같은 출력 과정을 열을 바꾸어 가며 빠르게 패턴 데이터를 출력하면, DotMatrix 모듈 전체의 LED에 원하는 패턴을 출력하는 효과를 얻을 수 있다. 이러한 제어 방식을 DotMatrix의 동적 제어라 한다.

다음의 코드는 이러한 동적 제어 과정을 프로그램 코드로 표현한 것으로, 각 열에 Low 신호를 인가해 가면서 행에 FFH 패턴 데이터를 출력한다. 단, 8x8 DotMatrix 모듈을 제어하기 위한 8비트의 COLS라는 포트가 추가로 제공된다고 가정한 상황이다. 처리 결과는 DotMatrix 모듈 내의 모든 LED를 ON하는 결과를 얻게 된다.

```

unsigned char colData = 0x01;
unsighed char i;

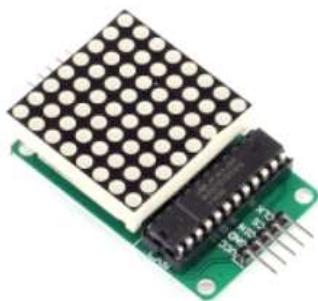
COLS = colData;
i = 0;
while(1) {
    delay(10); // short delay
    if(++i == 8)
        i = 0;

    COLS = ~(colData << i); // low signal to a Col
    P1 = 0xFF; // all bits ON
}

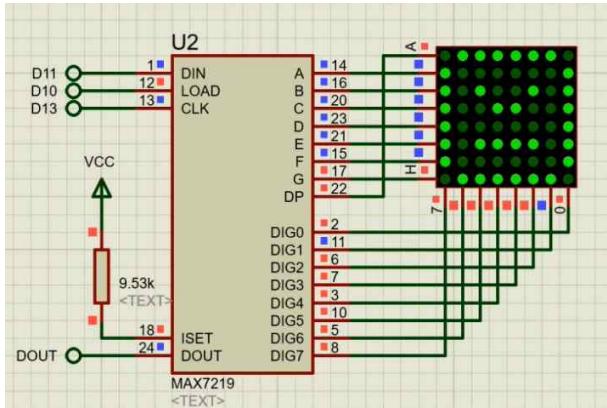
```

7.1.2 8x8 DotMatrix 모듈

위와 같은 동적 제어 방법은 요구하는 GPIO 핀이 많이 소요되고, 소프트웨적인 동적 제어방법에 어려움이 있을 수 있다. 이 같은 동적 제어 방식을 하드웨어적으로 할 수 있는 IC가 제공되고 있으며, MAX7219가 그 기능을 한다. 근래의 모듈들은 이러한 MAX7219 드라이버가 장착된 DotMatrix 모듈이 일반적이다.



이러한 DotMatrix 모듈의 회로도는 다음 그림과 같으며, 3개의 GPIO 핀을 연결하여 제어할 수 있다.



이 모듈내의 MAX7219 드라이버는 출력하는 데이터도 쉬프트 방식으로 1비트씩 인가하므로 구동에 필요한 GPIO 핀의 수를 획기적으로 줄일 수 있으며, 동적 제어 방식에 대해 신경쓸 필요가 없다. 다만 MAX7219 드라이버를 다룰 수 있어야 한다.

7.1.3 wiringShift.h 라이브러리

우선 데이터를 비트단위 쉬프트 방식으로 취급하는 라이브러리 함수에 대해 살펴본다. 데이터를 쉬프트 방식으로 취급하는 함수들을 위한 라이브러리가 wiringShift.h 헤더파일로 제공된다.

비트단위 쉬프트에 있어 LSB 비트부터 혹은 MSB 비트부터 입력 혹은 출력할지를 선택할 수 있는 2가지 상수가 다음과 같이 정의되어 있다.

```
#define LSBFIRST      0      // order constant
#define MSBFIRST      1      // order constant
```

다음의 함수는 데이터를 비트 단위로 쉬프트하며 입력받는 함수이다. dPin은 데이터 인가 GPIO 핀이며, cPin은 Clk을 위한 GPIO 핀이다.

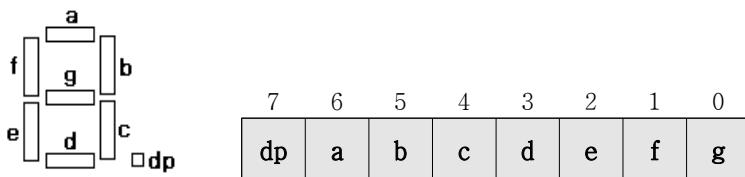
```
extern uint8_t shiftIn(uint8_t dPin, uint8_t cPin, uint8_t order);
```

데이터를 비트단위로 쉬프트하면서 출력하는 함수로 다음의 함수가 정의되어 있다. dPin은 데이터 인가 GPIO 핀이며, cPin은 Clk을 위한 GPIO 핀이다. order는 어느 비트부터 우선 출력할 것인가를 선택하며, val은 출력할 8비트 데이터이다.

```
extern void shiftOut(uint8_t dPin, uint8_t cPin, uint8_t order, uint8_t val);
```

7.1.4 MAX7219 드라이버

MAX7219는 여러 자리의 FND나 도트매트릭스 등을 제어할 때 유용하게 사용된다. 참고로 하나의 FND에 데이터를 표현할 때의 데이터 포맷은 다음과 같음을 유의하고, 이는 도트매트릭스 모듈과의 회로연결에 고려되어야 한다.

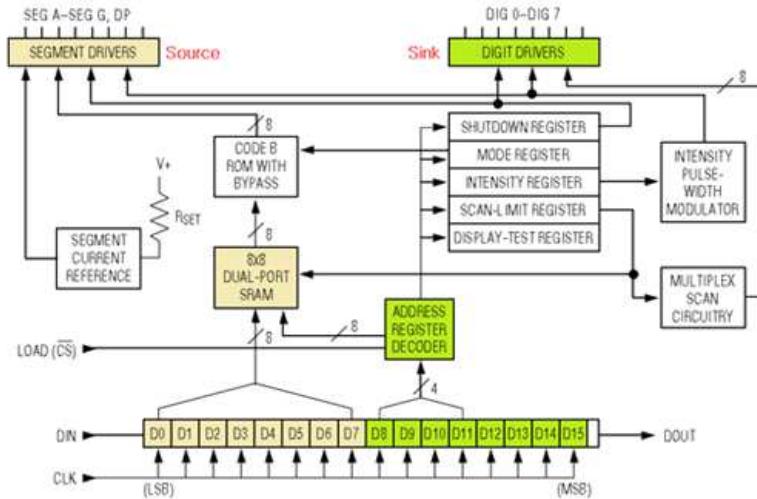


도트매트릭스는 8x8크기의 Common Cathode 방식으로 우리가 표현하고자 하는 이미지(Segment)를 LED의 Anode로 보내주면 8개의 LED가 공통으로 묶인 Cathode에서 전류를 Sink하는 방식이다.

MAX7219에는 전류를 제한하는 방식으로 IC가 동작하므로 LED에 별도의 저항을 연결할 필요는 없다. 제어를 위한 몇가지 명령을 활용하여 밝기, 인가할 데이터비트 수 등을 제어할 수 있다.

내부 구조는 아래 그림과 같이 구성되어 있다. 라즈베리파이에서 레지스터 주소 1바이트와 데이터 1 바이트 즉 총 2바이트를 보내주는데 이 2바이트 중 상위 바이트의 8비트중 하위 4비트는 Address Register Decoder로 들어가고 하위 8비트는 MAX7219에 있는 메모리로 저장된다. 여기서 저장된 데이터는 다음에 값이 들어올 때까지 일정한 시간 간격으로 Segment Drivers로 보내지게 된다. 즉, 자체적으로 동적 제어 방식에 의해 연결된 디바이스를 구동하는 것이다.

188 라즈베리파이기반 임베디드시스템응용



아래 그림은 제어에 사용되는 16개의 레지스터에 대한 주소를 확인할 수 있다. No-Op 레지스터는 2개 이상의 MAX7219를 사용할 경우에 사용하며, 특정 주소의 MAX7219에 명령을 전달하기 위해서 사용하는 것이므로 하나만 사용하는 경우 사용할 필요가 없는 레지스터이다. Digit 0부터 Digit 7까지의 레지스터는 각 Digit 위치에 보낼 데이터의 주소를 지정할 때 사용한다. 여기서 Digit 0 레지스터의 주소는 0xX1임을 유의한다.

REGISTER	ADDRESS					HEX CODE
	D15-D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Digit 0	X	0	0	0	1	0xX1
Digit 1	X	0	0	1	0	0xX2
Digit 2	X	0	0	1	1	0xX3
Digit 3	X	0	1	0	0	0xX4
Digit 4	X	0	1	0	1	0xX5
Digit 5	X	0	1	1	0	0xX6
Digit 6	X	0	1	1	1	0xX7
Digit 7	X	1	0	0	0	0xX8
Decode Mode	X	1	0	0	1	0xX9
Intensity	X	1	0	1	0	0xXA
Scan Limit	X	1	0	1	1	0xXB
Shutdown	X	1	1	0	0	0xXC
Display Test	X	1	1	1	1	0xXF

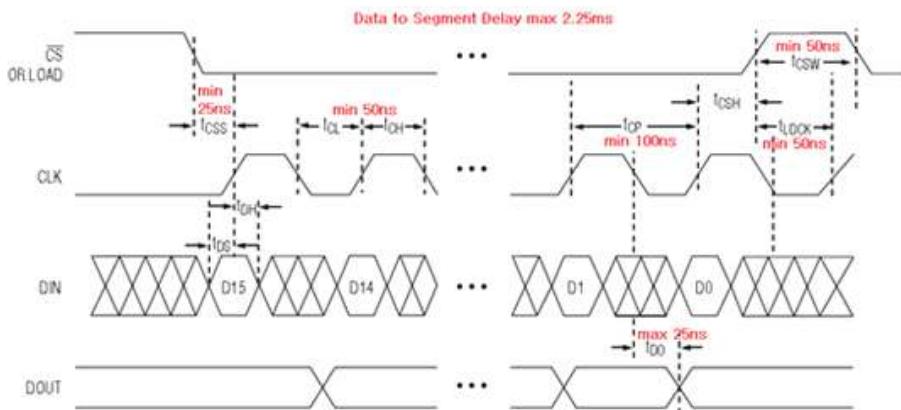
하단의 5개의 레지스터는 MAX7219를 초기화할 때 사용하는 레지스터들이다.

Decode mode 레지스터는 Digit n 레지스터의 데이터를 BCD 코드 폰트로 디코딩(FND 사용시)할지 여부를 설정하는 것으로 도트매트릭스의 경우 모두 BCD 코드로 디코딩 않는 0x00로 설정한다. Intensity 레지스터는 밝기 정도를 설정하는 레지스터로 0x00~0x0f의 16단계 설정 가능하다. Scan limit 레지스터는 Digit 레지스터를 몇 개까지 사용할 것인지 설정하는 것으로 Digit n 레지스터까지 사용할 경우 n을 설정한다. 도트매트릭스는 8x8로 Digit 7 레지스터까지 사용해야 하므로 0x07로 설정한다. shutdown 레지스터는 shutdown 시 0x00으로, 노말 상태 경우 0x01로 설정한다.

초기화와 관련된 레지스터들을 활용하여 8x8 도트매트릭스 모듈의 MAX7219를 초기화하는 함수를 구현하면 다음과 같다.

```
void MAX7219_INIT(void) {
    MAX7219_WRITE(0x09, 0x00);
    // Decode Mode : no decode all digits(0x00)
    MAX7219_WRITE(0x0A, 0x0F); // Intensity : max(0x0F)
    MAX7219_WRITE(0x0B, 0x07);
    // Scan Limit : Digit 0 ~ Digit 7, (0x07)
    MAX7219_WRITE(0x0C, 0x01); // Shutdown : Normal(0x01)
}
```

다음의 타이밍 도는 프로그램을 코딩할 때 신호를 정상적으로 인식할 수 있는 최소한의 시간과 신호의 순서를 알려주는 것으로 CS, CLK, DIN의 타이밍이 맞아야 제대로 동작이 가능하다.

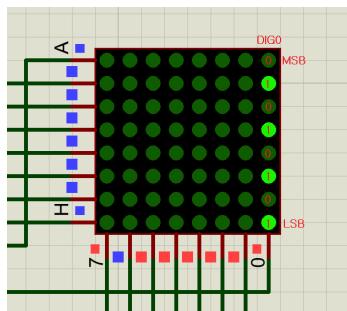


이를 함수로 구현하면 다음과 같다.

```
void MAX7219_WRITE(uint8_t address, uint8_t data) {
    digitalWrite(CS, LOW);
    shiftOut(DIN, CLK, MSBFIRST, address);
    shiftOut(DIN, CLK, MSBFIRST, data);
    digitalWrite(CS, HIGH);
}
```

초기화 된 후에, 다음의 문장을 실행하면 Digit 0에 0x55(0b01010101)를 전송하여 아래 그림과 같이 출력된다. 오른쪽 끝이 Digit 0 열이 되고 전송된 데이터는 MSB가 위쪽에 자리를 잡게 된다.

```
MAX7219_WRITE(1, 0x55); //
```



MAX7219는 최대 10MHz까지 신호를 수신할 수 있으므로 SPI 인터페이스 방식으로도 제어가 가능하다. 이 방식으로의 제어는 SPI 인터페이스 장에서 살펴본다.

7.1.5 실습예제

[실습1] 도트매트릭스 제어 I

글자 마를 구성하는 패턴중 한 열씩을 차례로 표시하여 글자를 완성하는 프로그램이다. `shiftOut()` 함수를 활용한다.

```
$ nano dotMtx_01.c
```

```

//=====
// dotMtx_01.c
//     using shift functions
//=====

#include <stdio.h>
#include <wiringPi.h>
#include <wiringShift.h>           // shift funciton

#define P_MAX_CLK      0      // BCM_GPIO #17
#define P_MAX_CS       2      // BCM_GPIO #27
#define P_MAX_DIN      3      // BCM_GPIO #22

const uint8_t image[] = {          // 5x8 font
    0x1E, 0x12, 0x1E, 0x00, 0x04, 0x07, 0x04, 0x04, // ㅁ
};

const uint8_t allOff[] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

uint8_t buf[8];

void MAX7219_WRITE(uint8_t address, uint8_t data) {
    digitalWrite(P_MAX_CS, LOW);
    shiftOut(P_MAX_DIN, P_MAX_CLK, MSBFIRST, address);
    shiftOut(P_MAX_DIN, P_MAX_CLK, MSBFIRST, data);
    digitalWrite(P_MAX_CS, HIGH);
}

void MAX7219_INIT(void) {
    MAX7219_WRITE(0x09, 0x00);
        // Decode Mode : no decode all digits(0x00)
    MAX7219_WRITE(0x0A, 0xF);   // Intensity : max(0x0F)
    MAX7219_WRITE(0x0B, 0x07);
        // Scan Limit : Digit 0 ~ Digit 7, (0x07)
    MAX7219_WRITE(0x0C, 0x01); // Shutdown : Normal(0x01)
}

void setup(void) {
    pinMode(P_MAX_DIN, OUTPUT);
    pinMode(P_MAX_CS, OUTPUT);
    pinMode(P_MAX_CLK, OUTPUT);
}

```

```
digitalWrite(P_MAX_CS, HIGH);

MAX7219_INIT();
}

int main(void) {
    int i, j;
    int k;

    printf("[Dot Matrix testing....]\n");

    if(wiringPiSetup() == -1)
        return 1;

    setup();

    for(i=0; i<8; i++){
        MAX7219_WRITE(i+1, image[i]);
        delay(1000);
    }

    for(i=0; i<8; i++){
        MAX7219_WRITE(i+1, allOff[i]);
    }
    delay(1000);

    return 0;
}
```

```
$ make dotMtx_01
$ ./dotMtx_01
```

[실습2] 도트매트릭스 제어 II

문자열 마이크로프로세서를 한 글자 패턴씩을 차례로 표시하는 프로그램이다.

```
$ nano dotMtx_02.c
//=====
```

```

// dotMtx_02.c
//      using shift functions
//=====
#include <stdio.h>
#include <wiringPi.h>
#include <wiringShift.h>      // shift funciton

#define P_MAX_CLK      0      //BCMGPIO #17
#define P_MAX_CS       2      //BCMGPIO #27
#define P_MAX_DIN      3      //BCMGPIO #22

const uint8_t image[] = {      // 5x8 font
    0x1E, 0x12, 0x1E, 0x00, 0x04, 0x07, 0x04, 0x04, //나
    0x09, 0x15, 0x09, 0x01, 0x01, 0x01, 0x01, 0x01, //이
    0x1F, 0x01, 0x07, 0x01, 0x00, 0x1F, 0x00, 0x00, //크
    0x1F, 0x01, 0x1F, 0x10, 0x1F, 0x00, 0x04, 0x1F, //로
    0x1F, 0x0A, 0x0A, 0x1F, 0x00, 0x1F, 0x00, 0x00, //프
    0x1F, 0x01, 0x1F, 0x10, 0x1F, 0x00, 0x04, 0x1F, //로
    0x04, 0x0A, 0x11, 0x00, 0x05, 0x1D, 0x05, 0x05, //세
    0x04, 0x0A, 0x11, 0x00, 0x01, 0x1F, 0x01, 0x01 //서
};

const uint8_t allOff[] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

uint8_t buf[8];

void MAX7219_WRITE(uint8_t address, uint8_t data) {
    digitalWrite(P_MAX_CS, LOW);
    shiftOut(P_MAX_DIN, P_MAX_CLK, MSBFIRST, address);
    shiftOut(P_MAX_DIN, P_MAX_CLK, MSBFIRST, data);
    digitalWrite(P_MAX_CS, HIGH);
}

void MAX7219_INIT(void) {
    MAX7219_WRITE(0x09, 0x00);
    // Decode Mode : no decode all digits(0x00)
    MAX7219_WRITE(0x0A, 0x0F); // Intensity : max(0x0F)
    MAX7219_WRITE(0x0B, 0x07);
    // Scan Limit : Digit 0 ~ Digit 7, (0x07)
    MAX7219_WRITE(0x0C, 0x01); // Shutdown : Normal(0x01)
}

```

```
}

void setup(void) {
    pinMode(P_MAX_DIN, OUTPUT);
    pinMode(P_MAX_CS, OUTPUT);
    pinMode(P_MAX_CLK, OUTPUT);
    digitalWrite(P_MAX_CS, HIGH);

    MAX7219_INIT();
}

int main(void) {
    int i, j;
    int k;

    printf("[Dot Matrix testing....]\n");

    if(wiringPiSetup() == -1)
        return 1;

    setup();

    for(j=0; j<8; j++){
        for(i=0; i<8; i++){
            MAX7219_WRITE(i+1, *(image+j*8+i));
        }
        delay(1000);
    }

    for(i=0; i<8; i++){
        MAX7219_WRITE(i+1, *(allOff + i));
    }
    delay(1000);

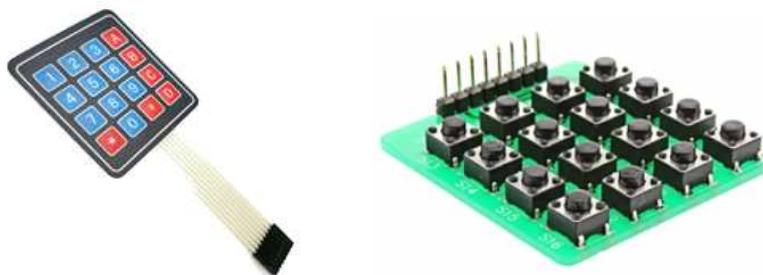
    return 0;
}

$ make dotMtx_02
$ ./dotMtx_02
```

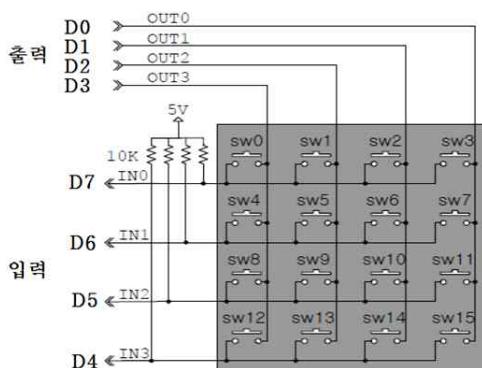
7.2 4x4 키패드

7.2.1 4x4 matrix 키패드 모듈

4x4 키패드 모듈은 다음의 외양을 갖는다. 8개의 연결 핀중 1, 2, 3, 4번 핀은 행을, 5, 6, 7, 8번 핀은 열을 나타낸다.



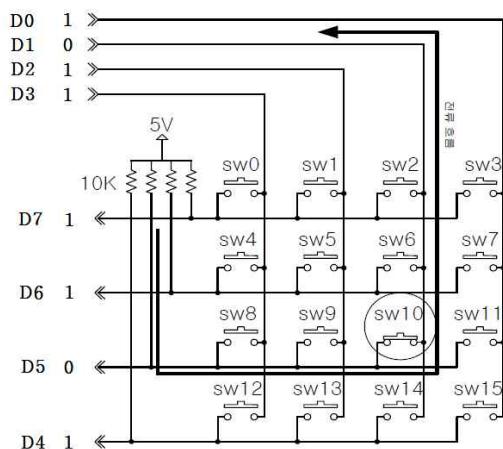
회로는 회로도와 같이 키패드 모듈의 D7, ..., D4단에 풀업저항을 연결한다.



7.2.2 키패드 동작 제어

196 라즈베리파이기반 임베디드시스템응용

Keypad 모듈의 키 눌림 감지를 설명하기 위해 다음 그림에서 보듯이 sw10의 버튼 스위치가 눌렸다고 가정한다. 이 sw10이 있는 열에 즉, 포트의 하위 니블을 통해 Low 신호를 인가하였을 때, 포트의 상위 니블을 통해 판독한 값에는 Low 신호를 갖는 비트가 존재한다. 이 때 하위 니블에 인가한 신호와 상위 니블을 통해 판독한 신호를 조합하면, 해당 키에 대한 코드 값을 결정할 수 있다. 만일 이러한 과정에서 어떠한 버튼도 눌리지 않으면, 판독한 값에는 Low 신호가 있을 수 없다.



다음은 키패드 모듈을 동적 제어 방법에 의해 제어하기 위한 코드를 구현한 것이다.

```
out = 0x01; // 특정 열 선택
for(i=0; i<4; i++) {
    P1 = ~out; // 특정 열에 Low 신호 인가
    in = (~P1) & 0xF0; // 상위 니블 판독
    if(in) {
        in |= out; // 코드 생성
        break;
    }
    out <= 1; // 다음 열
}
```

다음 표는 이러한 조합에 의해 결정된 각 키에 대한 코드 값을 정리한 것이다. 여기서, 각 키의 코드 값은 하위 니블에 인가된 신호에 비트 반전하고, 상위 니블을

통해 판독된 신호를 비트 반전하여 논리합의 형태로 결정한다.

키	상위니블	하위니블	코드값	키	상위니블	하위니블	코드값
SW0	~0x80	~0x08	0x88	SW8	~0x20	~0x08	0x28
SW1	~0x80	~0x04	0x84	SW9	~0x20	~0x04	0x24
SW2	~0x80	~0x02	0x82	SW10	~0x20	~0x02	0x22
SW3	~0x80	~0x01	0x81	SW11	~0x20	~0x01	0x21
SW4	~0x40	~0x08	0x28	SW12	~0x10	~0x08	0x18
SW5	~0x40	~0x04	0x24	SW13	~0x10	~0x04	0x14
SW6	~0x40	~0x02	0x22	SW14	~0x10	~0x02	0x12
SW7	~0x40	~0x01	0x21	SW15	~0x10	~0x01	0x11

7.2.3 실습과제

keypad.h 사용자 라이브러리

키패드 모듈을 동적제어하기 위한 사용자 라이브러리를 다음과 같이 keypad.h 파일로 구현한다.

```
$ nano keypad.h
//=====
// keypad.h
//=====
#ifndef __KEYPAD_H__
#define __KEYPAD_H__

#include <stdint.h>

#define P_KPD_D0      21          // BCM_GPIO #05
#define P_KPD_D1      22          // BCM_GPIO #06
#define P_KPD_D2      23          // BCM_GPIO #13
#define P_KPD_D3      24          // BCM_GPIO #19
#define P_KPD_D4      25          // BCM_GPIO #26
#define P_KPD_D5      26          // BCM_GPIO #12
#define P_KPD_D6      27          // BCM_GPIO #16
#define P_KPD_D7      28          // BCM_GPIO #20

#define NO_KEY        0x00
```

```

struct pins {
    unsigned int pin0      : 1;      // LSB
    unsigned int pin1      : 1;
    unsigned int pin2      : 1;
    unsigned int pin3      : 1;
    unsigned int pin4      : 1;
    unsigned int pin5      : 1;
    unsigned int pin6      : 1;
    unsigned int pin7      : 1;      // MSB
};

union ucode {
    uint8_t val;
    struct pins pin;           // bit-field
};

uint8_t p_in = NO_KEY;
union ucode acode;

void setup(void) {
    pinMode(P_KPD_D0, OUTPUT);   // output
    pinMode(P_KPD_D1, OUTPUT);   // output
    pinMode(P_KPD_D2, OUTPUT);   // output
    pinMode(P_KPD_D3, OUTPUT);   // output

    pinMode(P_KPD_D4, INPUT);    // input
    pinMode(P_KPD_D5, INPUT);    // input
    pinMode(P_KPD_D6, INPUT);    // input
    pinMode(P_KPD_D7, INPUT);    // input
}

void out_data(union ucode acode) {
    digitalWrite(P_KPD_D0, acode.pin.pin0);
    digitalWrite(P_KPD_D0, acode.pin.pin1);
    digitalWrite(P_KPD_D0, acode.pin.pin2);
    digitalWrite(P_KPD_D0, acode.pin.pin3);
}

uint8_t in_data(void) {
    union ucode acode;

```

```
acode.val = 0x00;

acode.pin.pin4 = digitalRead(P_KPD_D4);
acode.pin.pin5 = digitalRead(P_KPD_D5);
acode.pin.pin6 = digitalRead(P_KPD_D6);
acode.pin.pin7 = digitalRead(P_KPD_D7);

return acode.val;
}

static uint8_t keyScan(void) {
    uint8_t out, i, in, tmp;
    union ucode acode;

    out = 0x01;
    for(i=0; i<4; i++) {
        acode.val = ~out;
        out_data(acode);

        tmp = in_data();
        in = (~tmp) & 0xF0;

        if(in) {
            in |= out;
            break;
        }
        out <<= 1;
    }

    return in;
}

uint8_t keyInput(void) {
    uint8_t in, in1;

    in = keyScan();

    while(1) {
        delay(10);
        in1 = keyScan();
        if(in == in1)
            break;
    }
}
```

```
        in = in1;
    }

    if(!(in & 0xF0)) {          // 상위니를 검사, 어떤키도 눌리지 않았다면
        p_in = NO_KEY;
        return NO_KEY;
    }
    if(p_in==in)               // 이전키와 같다면, 눌리지 않은 것으로 간주
        return NO_KEY;

    p_in = in;
    return in;
}

#endif
```

[실습1] 키패드 제어 I

사용자 라이브러리 keypad.h를 활용하여 키패드 모듈로부터 눌린 키의 정보를 출력하는 프로그램을 구현한다.

```
$ nano keypad_01.c
//=====
// keypad_01.c
//      4x4 Keypad module, dynamic control
//      keypad software module : keypad.h
//
//=====
#include <stdio.h>
#include <stdint.h>
#include <wiringPi.h>

#include "keypad.h"

int main(void) {
    uint8_t key;
    char ch;

    printf("[4x4 Keypad testing....]\n");
```

```
if(wiringPiSetup() == -1)
    return 1;

setup();

while(1) {
    key = keyInput();
    if(key==NO_KEY)
        continue;

    ch = ' ';
    switch(key) {
        case 0x88 :
            ch = '1'; break;
        case 0x84 :
            ch = '2'; break;
        case 0x82 :
            ch = '3'; break;
        case 0x81 :
            ch = 'A'; break;
        case 0x48 :
            ch = '4'; break;
        case 0x44 :
            ch = '5'; break;
        case 0x42 :
            ch = '6'; break;
        case 0x41 :
            ch = 'B'; break;
        case 0x28 :
            ch = '7'; break;
        case 0x24 :
            ch = '8'; break;
        case 0x22 :
            ch = '9'; break;
        case 0x21 :
            ch = 'C'; break;
        case 0x18 :
            ch = '*'; break;
        case 0x14 :
            ch = '0'; break;
        case 0x12 :
```

```

        ch = '#'; break;
    case 0x11 :
        ch = 'D'; break;
    }
    printf("key : %x, symbol : %c\n", key, ch);
}

return 0;
}

```

```

$ make keypad_01
$ ./keypad_01

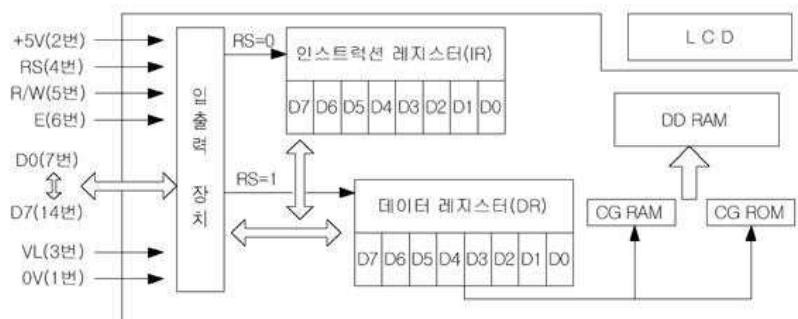
```

키를 눌러 그 코드값과 키 이름의 출력결과를 관찰한다.

7.3 CLCD

7.3.1 CLCD 모듈

CLCD 모듈의 내부 구조는 다음 그림과 같다.



특히 5V로 구동되는 CLCD를 사용할 경우, 라즈베리파이는 3.3V로 동작되므로 5V 신호가 GPIO핀으로 인가되는 경우 고장의 원인이 될 수 있다. 따라서 라즈베리파

이와 동작을 실습할 때 R/W 핀은 Gnd에 연결하여, 라즈베리파이로 5V 신호를 기록하지 않도록 해야 함을 유의한다.

7.3.2 CLCD 라이브러리

CLCD는 4비트 버스 모드와 8비트 버스 모드로 제어할 수 있다. CLCD를 제어하기 위한 함수들을 위한 헤더파일은 /usr/include/lcd.h로 제공된다. 이들 함수들의 기능을 살펴본다.

다음의 함수는 LCD를 초기화하는 함수로, 표시 행과 열의 수, 버스 제어 모드, 사용 GPIO 핀 등을 설정하는 함수이다. 여기서 사용키로한 GPIO 핀은 핀 모드가 용도에 맞게 자체에서 설정됨을 유의하며, 다른 어떠한 LCD 관련 함수보다 우선하여 호출되어야 한다.

```
extern int lcdInit (const int rows, const int cols, const int bits,
                    const int rs, const int strb,
                    const int d0, const int d1, const int d2, const int d3,
                    const int d4, const int d5, const int d6, const int d7) ;
```

여기서 rows, cols은 LCD 유형에 따른 표시 행과 열을 의미하며, bits는 버스제어 모드에 따라 4 혹은 8로 설정한다. rs, strb는 RS 단자와 E 단자의 GPIO 핀 번호를, d0 ~ d3은 4비트 버스 모드를 위한 GPIO 핀 번호를, d0 ~ d7은 8비트 버스 모드를 위한 GPIO 핀 번호를 설정한다. 여기서의 핀 번호체계는 GPIO 핀 번호 체계사용 함수에 의존한다. return 값은 이후의 제어에서 핸들로 사용되며, 적절히 초기화가 불가능할 경우 -1을 반환한다.

다음은 4비트 버스모드와 8비트 버스모드에서의 초기화 함수 호출 사례를 보여준다.

```
fd = lcdInit (2, 16, 4, 11, 10, 0, 1, 2, 3, 0, 0, 0, 0);      //4비트 버스모드
fd = lcdInit (2, 16, 8, 11, 10, 0, 1, 2, 3, 4, 5, 6, 7);      //8비트 버스 모드
```

커서를 화면으로 이동하거나 화면 소거 등등의 다음과 같은 함수들이 정의되어 있으

며, 함수명을 통해 그 기능을 추정할 수 있다.

```
extern void lcdHome(const int fd) ;  
extern void lcdClear(const int fd) ;  
extern void lcdDisplay(const int fd, int state) ;  
extern void lcdCursor(const int fd, int state) ;  
extern void lcdCursorBlink(const int fd, int state) ;
```

다음 함수는 데이터를 출력할 커서의 위치를 지정하는 함수로, x는 디스플레이의 라인번호(0, 1), y는 칼럼번호(0,)를 나타낸다.

```
extern void lcdPosition(const int fd, int x, int y) ;
```

CLCD에 데이터를 출력하는 함수로, 다음과 같이 한 문자 출력 함수, 문자열 출력 함수, 서식지정된 문자열 출력 함수가 정의되어 있다.

```
extern void lcdPutchar(const int fd, unsigned char data) ;  
extern void lcdPuts(const int fd, const char *string) ;  
extern void lcdPrintf(const int fd, const char *message, ...) ;
```

다음의 함수는 LCD 명령을 CLCD에 기록하는 함수이다.

```
extern void lcdSendCommand(const int fd, unsigned char command) ;
```

사용자 문자 패턴을 LCD에 등록하는 다음의 함수가 정의되어 있다. index는 CGRAM의 주소, data[]는 한 문자의 패턴 데이터들의 배열을 의미한다.

```
extern void lcdCharDef(const int fd, int index, unsigned char data[8]) ;
```

Makefile 수정

lcd.h는 wiringPI 라이브러리의 기본구성이 아니므로 컴파일시 -lwiringPiDev를 덧붙여서 컴파일한다. 따라서 make 유ти리티를 사용하여 컴파일하는 경우라면 다음과 같이 수정한다.

```

=====
# Makefile for CLCD
=====
CC = gcc
CFLAGS = -c
LDFLAGS = -lwiringPi -lm -lwiringPiDev
LDLIBS =

%: %.c
    $(CC) -o $@ $< $(LDFLAGS)

default:
    @echo "Type \"make filename\" to compile..."

clean:
    rm -f *.o

```

7.3.3 실습과제

[실습1] CLCD 제어 I

lcd.h를 활용하여 CLCD에 문자열 및 문자를 표시하는 프로그램을 구현한다. 4비트 버스모드를 사용한다.

```

$ nano clcd_01.c
//=====
// clcd_01.c
//      2x16 CLCD, 4bit bus mode
//      lcd library : lcd..h
//
//=====
#include <stdio.h>
#include <wiringPi.h>
#include <lcd.h>           // in /usr/include

```

```
#define CLCD_RS      0      // BCM_GPIO #17
#define CLCD_RW      1      // BCM_GPIO #18

#define CLCD_D0      2      // BCM_GPIO #27
#define CLCD_D1      3      // BCM_GPIO #22
#define CLCD_D2      4      // BCM_GPIO #23
#define CLCD_D3      5      // BCM_GPIO #24

int main(void) {
    int fd;           // CLCD handle

    printf("2x16 CLCD testing....4bit bus]\n");

    if(wiringPiSetup() == -1)
        return 1;

    // 4bit bus mode
    fd = lcdInit(2, 16, 4, CLCD_RS, CLCD_RW,
                 CLCD_D0, CLCD_D1, CLCD_D2, CLCD_D3, 0, 0, 0, 0);
    lcdPosition(fd, 0, 0);

    // display a string
    lcdPrintf(fd, "CLCD Testing..");
    delay(1000);

    // display a character
    lcdPosition(fd, 1, 0);           //
    lcdPutchar(fd,'A');

    delay(10000);

    lcdClear(fd);

    return 0;
}

$ make clcd_01
$ ./clcd_01
```

[실습2] CLCD 제어 II

lcd.h를 활용하여 CLCD에 사용자 패턴을 등록하고 이를 표시하는 프로그램을 구현한다. 4비트 버스모드를 사용한다.

```
$ nano clcd_02.c
//=====
// clcd_02.c
//      2x16 CLCD, 4bit bus mode
//      lcd library : lcd..h
//      register & display User patterns
//=====

#include <stdio.h>
#include <wiringPi.h>
#include <lcd.h>                                // in /usr/include

#define CLCD_RS      0      // BCM_GPIO #17
#define CLCD_RW      1      // BCM_GPIO #18

#define CLCD_D0      2      // BCM_GPIO #27
#define CLCD_D1      3      // BCM_GPIO #22
#define CLCD_D2      4      // BCM_GPIO #23
#define CLCD_D3      5      // BCM_GPIO #24

unsigned char image[] = {      // 5x8 font
    0x1E, 0x12, 0x1E, 0x00, 0x04, 0x07, 0x04, 0x04, // 마
    0x09, 0x15, 0x09, 0x01, 0x01, 0x01, 0x01, 0x01, // 이
    0x1F, 0x01, 0x07, 0x01, 0x00, 0x1F, 0x00, 0x00, // 크
    0x1F, 0x01, 0x1F, 0x10, 0x1F, 0x00, 0x04, 0x1F, // 로
    0x1F, 0x0A, 0x0A, 0x1F, 0x00, 0x1F, 0x00, 0x00, // 프
    0x1F, 0x01, 0x1F, 0x10, 0x1F, 0x00, 0x04, 0x1F, // 로
    0x04, 0x0A, 0x11, 0x00, 0x05, 0x1D, 0x05, 0x05, // 세
    0x04, 0x0A, 0x11, 0x00, 0x01, 0x1F, 0x01, 0x01  // 서
};

int main(void) {
    int fd;          // CLCD handle
    int i;

    printf("2x16 CLCD testing....4bit bus]\n");

    if(wiringPiSetup() == -1)
        return 1;
```

```

// 4bit bus mode
fd = lcdInit(2, 16, 4, CLCD_RS, CLCD_RW,
              CLCD_D0, CLCD_D1, CLCD_D2, CLCD_D3, 0, 0, 0, 0);
lcdPosition(fd, 0, 0);

lcdPrintf(fd, "CLCD Testing..");
delay(1000);

// register user patterns
for(i=0; i<8; i++)
    lcdCharDef(fd, i, &image[i*8]);

// display user patterns
lcdPosition(fd, 1, 0);
for(i=0; i<8; i++) {
    lcdPutchar(fd, i);
    delay(1000);
}
delay(10000);

lcdClear(fd);

return 0;
}

```

```

$ make clcd_02
$ ./clcd_02

```

[응용1] CLCD 제어 III

lcd.h를 활용하여 CLCD에 사용자 패턴을 등록하고 이를 표시하는 프로그램을 구현한다. 단, 8비트 버스모드를 사용한다.

```

$ nano clcd_03.c
//=====================================================================
// clcd_03.c
//      2x16 CLCD, 8bit bus mode
//      lcd library : lcd..h

```

```

//      register & display User patterns
//=====
#include <stdio.h>
#include <wiringPi.h>
#include <lcd.h>           // in /usr/include

#define CLCD_RS      0      // BCM_GPIO #17
#define CLCD_RW      1      // BCM_GPIO #18

#define CLCD_D0      2      // BCM_GPIO #27
#define CLCD_D1      3      // BCM_GPIO #22
#define CLCD_D2      4      // BCM_GPIO #23
#define CLCD_D3      5      // BCM_GPIO #24

#define CLCD_D4      6      // BCM_GPIO #25
#define CLCD_D5      7      // BCM_GPIO #04
#define CLCD_D6      8      // BCM_GPIO #02
#define CLCD_D7      9      // BCM_GPIO #03

unsigned char image[] = {      // 5x8 font
    0x1E, 0x12, 0x1E, 0x00, 0x04, 0x07, 0x04, 0x04, // 마
    0x09, 0x15, 0x09, 0x01, 0x01, 0x01, 0x01, 0x01, // 이
    0x1F, 0x01, 0x07, 0x01, 0x00, 0x1F, 0x00, 0x00, // 크
    0x1F, 0x01, 0x1F, 0x10, 0x1F, 0x00, 0x04, 0x1F, // 로
    0x1F, 0x0A, 0x0A, 0x1F, 0x00, 0x1F, 0x00, 0x00, // 프
    0x1F, 0x01, 0x1F, 0x10, 0x1F, 0x00, 0x04, 0x1F, // 로
    0x04, 0x0A, 0x11, 0x00, 0x05, 0x1D, 0x05, 0x05, // 세
    0x04, 0x0A, 0x11, 0x00, 0x01, 0x1F, 0x01, 0x01   // 서
};

int main(void) {
    int fd;          // CLCD handle
    int i;

    printf("2x16 CLCD testing....4bit bus]\n");

    if(wiringPiSetup() == -1)
        return 1;

    // 8bit bus mode
    fd = lcdInit(2, 16, 8, CLCD_RS, CLCD_RW,
                 CLCD_D0, CLCD_D1, CLCD_D2, CLCD_D3,

```

```
    CLCD_D4, CLCD_D5, CLCD_D6, CLCD_D7);
lcdPosition(fd, 0, 0);

lcdPrintf(fd, "CLCD Testing..");
delay(1000);

// register user patterns
for(i=0; i<8; i++)
    lcdCharDef(fd, i, &image[i*8]);

// display user patterns
lcdPosition(fd, 1, 0);
for(i=0; i<8; i++) {
    lcdPutchar(fd, i);
    delay(1000);
}
delay(10000);

lcdClear(fd);

return 0;
}
```

```
$ make clcd_03
$ ./clcd_03
```

참고자료

[1] MAX7219 드라이버

<http://blog.naver.com/PostView.nhn?blogId=kiatwins&logNo=221014079210>

[2] 도트매트릭스

<http://blog.naver.com/PostView.nhn?blogId=kiatwins&logNo=221014081177&parentCategoryNo=&categoryNo=&viewDate=&isShowPopularPosts=false&from=postView>

[3] 도트매트릭스

<http://blog.naver.com/PostView.nhn?blogId=kiatwins&logNo=221014084555&parentCategoryNo=&categoryNo=&viewDate=&isShowPopularPosts=false&from=postView>

[4] 키패드

<http://blog.naver.com/PostView.nhn?blogId=microfun&logNo=220581794180>

[5] 키패드

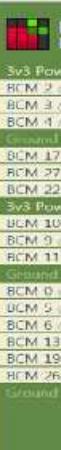
<http://me2.do/5YubgGIX>

[6] CLCD 제어

<http://blog.naver.com/PostView.nhn?blogId=nkkh159&logNo=220788205550>

[7] CLCD

https://cms3.koreatech.ac.kr/sites/joo/IFC181/IFC181_11.pdf



제8장 시리얼 통신

본 장에서는 시리얼 통신에 대해 살펴본다. 좀 더 세부적으로는 루프백 시리얼 통신부터 라즈베리파이 보드와 Win PC간, 라즈베리파이 보드와 아두이노 보드간, 라즈베리파이 보드와 8051 보드간 시리얼 통신에 대해 살펴보고, 각 보드에 있는 입출력 디바이스를 시리얼 통신으로 원격의 시스템에서 제어하는 것에 대해 살펴본다.

8.1 시리얼 통신

시리얼 통신은 UART(Universal Asynchronous Receiver/Transmitter) 통신이라고도 한다. 라즈베리파이 보드에서 RS232 시리얼 통신을 위해 송신용 단자인 TxD는 BCM_GPIO #14핀을, 수신용 단자인 RxD는 BCM_GPIO #15핀을 사용하도록 제공하고 있다. 또한, wiringPi 라이브러리로 시리얼 통신을 위한 라이브러리 제공된다.

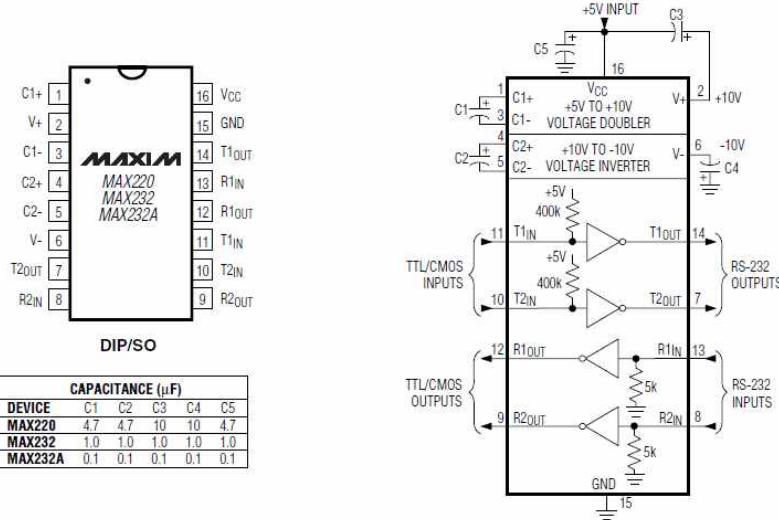
8.1.1 MAX232CPE

PC와 라즈베리파이 보드는 사용되는 신호 기준이 틀리기 때문에 바로 연결할 수가 없다. 라즈베리파이 보드는 0V~5.0V의 디지털 신호이고, PC는 -12V ~ +12V 디지털 신호를 사용하기 때문이다. 신호가 서로 다른 이 두 기종의 컴퓨터를 연결하기 위해서는 MAX232 IC를 이용하여 신호 레벨을 변환할 수 있도록 하여야 한다.

MAX232CPE 핀 레이아웃

회로 연결시 MAX232 계열의 IC 모델에 따라 회로 연결에 사용되는 캐패시터의 규격이 다름을 유의한다.

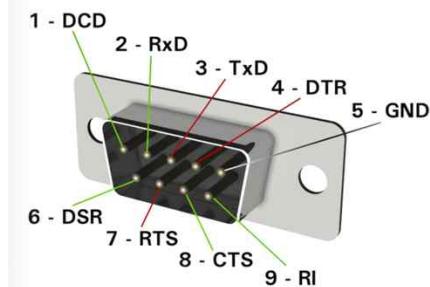
214 라즈베리파이기반 임베디드시스템응용



RS232 커넥터(D-sub 9) 핀 레이아웃

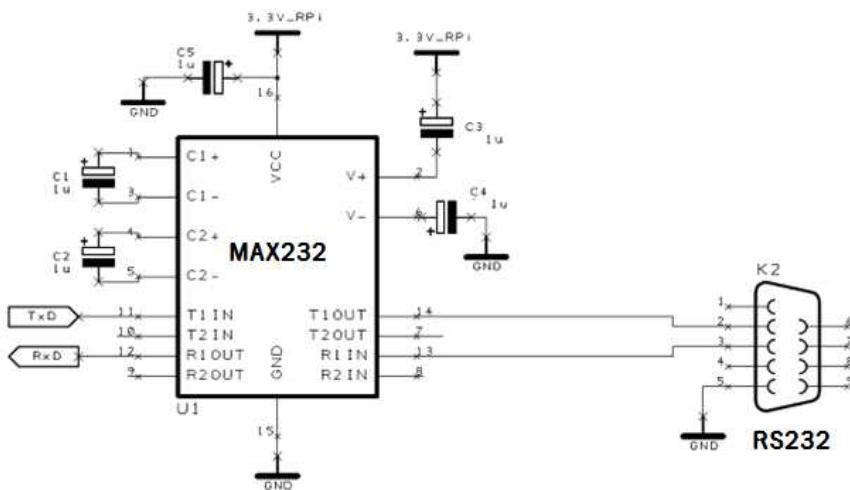
RS232 커넥터의 여러 핀 중 단순한 시리얼 통신을 위해 4개의 핀만을 사용한다. 이중 최소로 사용할 핀은 #2(RxD) , #3(TxD), #5(GND)이며, #5(GND) 핀은 사용하지 않아도 된다.

number	name	description	level	Dir.	Etc.
1	DCD	Data Carrier Detect	RS232	Input	Mandatory
2	RXD	Receive Data	RS232	Input	Mandatory
3	TXD	Transmit Data	RS232	Output	Mandatory
4	DTR	Data Terminal Ready	RS232	Output	Optional
5	GND	Ground	Ground	-	Mandatory
6	DSR	Data Set Ready	RS232	Input	Optional
7	RTS	Request To Send	RS232	Output	Optional
8	CTS	Clear To Send	RS232	Input	Optional
9	RI	Ring Indicator	RS232	Input	Optional



회로구성

PC와 라즈베리파이 보드간 시리얼 통신을 위한 전압 레벨의 호환을 위해 MAX232 IC를 활용하여 회로를 구성한다. MAX232에 1uP 캐패시터 5개를 사용하여 다음 그림과 같이 회로를 구성하고, MAX232 IC와 RS232 커넥터간의 TxD, RxD 단자는 서로 교차로 연결해 주어야 한다. 즉 MAX232의 T1OUT 단자와 RS232 커넥터의 RxD 핀(#2)을 연결하고, MAX232의 R1IN 단자와 RS232 커넥터의 TxD 핀(#3)을 연결한다. 아래의 그림과 같이 구성하면 교차 연결된 것이다.



이와 동일한 기능을 하는 기성 제품으로는 다음과 같은 RS232-TTL 변환 모듈이 있다. PC 측과의 거리가 떨어진 경우, PC에서 RS-232C 포트 혹은 USB 포트의 사용에 따라 연장케이블을 추가 사용할 수 있다.



참고로, 최신의 컴퓨터 또는 노트북에는 시리얼 포트가 제공되지 않는 경우가 있을 수 있다. 이러한 경우 일반적으로 USB 포트가 제공되므로 다음과 같은 USB to

216 라즈베리파이기반 임베디드시스템응용

Serial 변환 케이블이나 PL-2303 모듈을 이용할 수 있다. 이들을 사용할 경우 필요에 따라 드라이버를 설치할 필요가 있을 수 있다. 좌측 USB to Serial 변환 케이블의 몸체에는 우측과 같은 모듈이 들어있다.



이들 기성 모듈을 사용할 때에도 라즈베리파이 보드의 TxD, RxD 단자는 기성모듈의 RxD, TxD 단자로 각각 연결해 주어야 한다.

8.1.2 시리얼 통신 활성화

시리얼 통신을 위해서는 시리얼 통신 디바이스 파일 /dev/ttys0이 존재해야 한다. 라즈베리파이 보드에서 시리얼 통신과 관련한 초기 상태의 디바이스 파일 정보를 확인하기 위해 다음의 명령을 사용한다.

```
$ ls /dev/tty*
```

```
pi@raspberrypi:~/IFC413/serial $ ls /dev/tty*
/dev/tty   /dev/tty19  /dev/tty3   /dev/tty40  /dev/tty51  /dev/tty62
/dev/tty0  /dev/tty2   /dev/tty30  /dev/tty41  /dev/tty52  /dev/tty63
/dev/tty1  /dev/tty20  /dev/tty31  /dev/tty42  /dev/tty53  /dev/tty7
/dev/tty10 /dev/tty21  /dev/tty32  /dev/tty43  /dev/tty54  /dev/tty8
/dev/tty11 /dev/tty22  /dev/tty33  /dev/tty44  /dev/tty55  /dev/tty9
/dev/tty12 /dev/tty23  /dev/tty34  /dev/tty45  /dev/tty56  /dev/ttyAMA0
/dev/tty13 /dev/tty24  /dev/tty35  /dev/tty46  /dev/tty57  /dev/ttyprintk
/dev/tty14 /dev/tty25  /dev/tty36  /dev/tty47  /dev/tty58
/dev/tty15 /dev/tty26  /dev/tty37  /dev/tty48  /dev/tty59
/dev/tty16 /dev/tty27  /dev/tty38  /dev/tty49  /dev/tty6
/dev/tty17 /dev/tty28  /dev/tty39  /dev/tty5  /dev/tty60
/dev/tty18 /dev/tty29  /dev/tty4  /dev/tty50  /dev/tty61
pi@raspberrypi:~/IFC413/serial $
```

출력된 결과에서는 /dev/ttys0 디바이스 파일이 보이지 않는다. 이러한 시리얼 통신이 가능한지와 관련한 정보들은 다음의 명령들을 사용하여 확인 가능하다.

```
$ cat /boot/config.txt
```

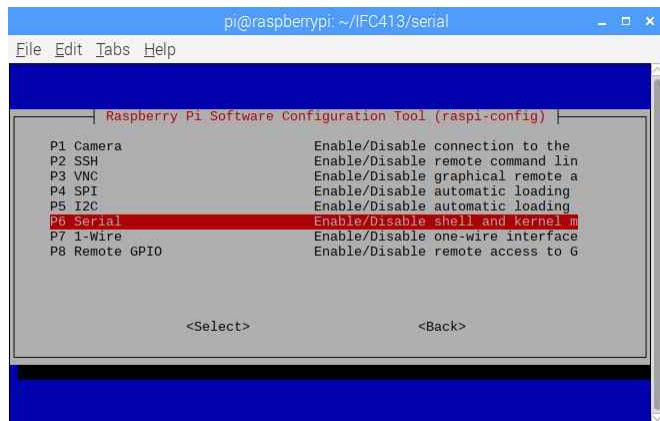
혹은, \$ cat /boot/cmdline.txt

혹은, \$ dmesg | grep tty

시리얼 통신을 위한 원하는 디바이스 파일이 존재하지 않는 것은 라즈베리파이의 환경 설정에서 시리얼 통신을 활성화하지 않았기 때문이다. 따라서 라즈베리파이 환경 설정을 위한 명령인 raspi-config을 사용하여 시리얼 통신이 가능하도록 활성화해야 한다.

다음 명령을 실행하면, 나타나는 창에서 Interfacing Options 항목을 선택한 후, Serial 항목을 선택하여 활성화한다. 이 항목을 활성화하면 시리얼 통신을 위한 디바이스 파일을 자동 생성하고 이후 부팅부터 적용되도록 설정한다. 재부팅 여부의 질의에 대한 응답으로든 재부팅 명령을 사용하든 재부팅한다.

\$ sudo raspi-config



재부팅한 후, 시리얼 통신을 위한 디바이스 파일의 생성 여부 확인을 위해 다음의 명령을 사용한다.

\$ 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/ttyprintk
pi@raspberrypi:~ $
```

218 라즈베리파이 기본 임베디드 시스템 활용

시리얼 통신을 위한 디바이스 파일 /dev/ttyS0 이 추가생성된 것을 확인할 수 있다. 이 파일이 시리얼 통신에 사용할 디바이스 파일이며, 이 파일을 열어 시리얼 통신을 할 수 있다. 특히 기본적으로 이 디바이스 파일의 접근권한은 슈퍼유저에게 있으므로, 이 디바이스 파일을 열어 시리얼 통신하는 응용 프로그램은 슈퍼유저 권한으로 실행해야 함을 유의한다.

다음 명령을 통하여 라즈베리파이 보드의 내정된 시리얼 통신 속도를 확인할 수 있으며, 특별한 경우가 아니면 시리얼 통신을 위한 baudrate를 이 115200로 설정 한다.

```
$ 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 $ █
```

다음 명령을 사용하여 시리얼 통신을 위한 디바이스 파일이 활성화되었는지 확인 할 수 있다.

```
$ 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 $ █
```

이렇게 시리얼 통신이 활성화되면, 시리얼 통신을 통한 로그인과 시리얼 통신 포트를 사용할 수 있는 상태가 된 것이다. 앞에서 살펴본 기타의 명령들을 사용하여 시리얼 통신의 활성 여부를 파악할 수 있다.

8.1.3 시리얼 모니터링

PC를 라즈베리파이 보드의 시리얼 단말장치처럼 사용하는 방법에 대해 살펴본다. 여기서는 USB-TTL Serial Cable을 이용하여 모니터나 LAN 케이블 없이 PC와 라즈베리파이 보드를 연결하고, 시리얼 통신으로 라즈베리파이 보드를 모니터링한다.

다음 그림과 같이 USB-TTL Serial Cable의 흰색 핀(RxD)을 라즈베리파이의 TxD 핀에, 초록 핀(TxD)을 라즈베리파이의 RxD 핀에 연결하고, 빨강 핀(Vcc)을 5V와 검정 핀(Gnd)을 Gnd 핀에 연결한다. 라즈베리파이 보드에 전원 공급기가 있는 경우 빨강 핀(Vcc)을 5V 단자와 연결하지 않아도 된다. 만일 라즈베리파이 보드에 외부 전원 공급기를 사용하지 않는 경우라면 빨강 핀(Vcc)을 5V 단자와 연결하면 USB를 통한 전원공급 효과를 얻을 수 있다. 이제 USB-TTL Serial 케이블의 USB 포트를 Windows PC에 연결한다.



이제 Windows 장치 관리자에서 드라이버를 온라인 업데이트하여 설치하거나, 다음의 사이트에서 PL2303 드라이버를 다운로드하여 설치를 진행한다.

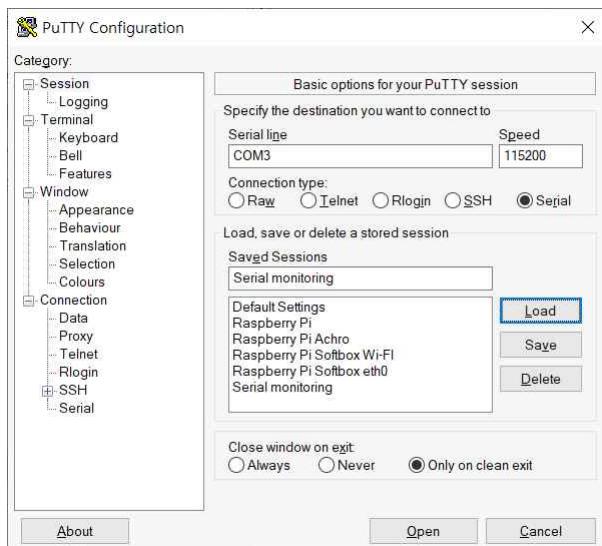
<https://learn.adafruit.com/adafruits-raspberry-pi-lesson-5-using-a-console-cable/software-installation-windows>

드라이버 설치 후 Windows 장치 관리자에서 COM 포트 번호를 다음과 같이 확인할 수 있다.

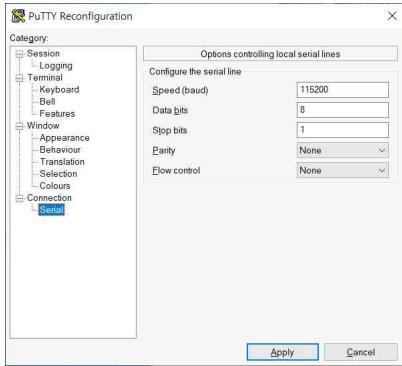
220 라즈베리파이기반 임베디드시스템응용



이제 시리얼 통신으로 접속하기 위해 PuTTY를 실행하여 그림과 같이 Connection Type은 'Serial'로 하고, Speed(Baudrate)는 115200으로 환경 설정 한다.



또한 좌측 Connection - Serial을 클릭하여 나타난 다음과 같은 화면에서 Flow control 항목을 none으로 반드시 선택한다.



이제 Open을 클릭하여 접속을 시도한다. 혹여 아무것도 나타나지 않는 경우는 라즈베리파이보드의 전원을 차단하였다가 재 인가하면 다음과 같은 화면을 볼 수 있다. 계정명과 패스워드를 입력하여 로그인하여 명령 등을 사용할 수 있다. 이는 LAN을 통해 SSH 연결할 때처럼 새로운 터미널을 여는 것이 아님을 참고한다.

```

bcm2835-aux-uart 3f215040.serial: could not get clk: -517
raspbian GNU/Linux 9 raspberrypi ttyS0
raspberripi login: pi
Password:
Last login: Tue May 28 01:17:09 UTC 2019 on ttym1
Linux raspberrypi 4.9.41-v7+ #1023 SMP Tue Aug 8 16:00:15 BST 2017 armv7l
The programs included with the Debian GNU/Linux system are free software.
SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

pi@raspberrypi:~$ ls
Desktop  Downloads  Pictures  python_games  Videos
Documents  Music    Public    Templates
pi@raspberrypi:~$ 

```

8.1.4 wiringSerial.h 라이브러리

wiringPi 라이브러리로 시리얼 통신을 위한 wiringSerial.h 라이브러리가 제공된다. 이 라이브러리는 시리얼 통신을 위한 다음과 같은 함수들이 정의되어 있다.

시리얼 통신을 위한 디바이스 파일과 보레이트 설정하여 여는 함수 외에 열기, 닫기, 버퍼에 있는 데이터를 버리는 다음과 같은 함수들이 있다.

```
extern int serialOpen(const char *device, const int baud) ;  
extern void serialClose(const int fd) ;  
extern void serialFlush(const int fd) ;
```

또한 시리얼 전송을 하는 함수로, 한 문자, 문자열, 서식지정된 문자열을 전송할 수 있는 다음의 함수들이 제공된다.

```
extern void serialPutchar(const int fd, const unsigned char c) ;  
extern void serialPuts(const int fd, const char *s) ;  
extern void serialPrintf(const int fd, const char *message, ...) ;
```

데이터를 시리얼 수신할 때, 수신할 데이터가 가용한지를 파악하기 위한 함수와 한 문자를 수신하는 함수가 다음과 같이 제공된다.

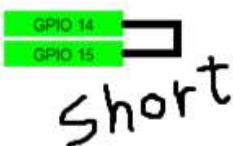
```
extern int serialDataAvail(const int fd) ;  
extern int serialGetchar(const int fd) ;
```

8.2 루프백 시리얼통신

루프백 시리얼 통신은 자신의 시스템에서 보낸 데이터를 자신이 수신하는 상황의 시리얼 통신이다.

[실습1] 루프백 시리얼 통신

루프백 시리얼 통신을 위해서 회로의 연결은 TxD 핀(BCM_GPIO #14)과 RxD 핀(BCM_GPIO #15)을 직접 연결하도록 구성한다.



```
$ nano uart_01.c
//=====
// uart_01.c
//      loopback
//      RxD(BCM_GPIO #14), TxD(BCM_GPIO #15)
//=====

#include <stdio.h>
#include <string.h>
#include <wiringPi.h>           // delay()
#include <wiringSerial.h>

#define BAUD    115200

int main(void) {
    int fd;
    unsigned char asc, rec;

    if((fd = serialOpen("/dev/ttyS0", BAUD)) < 0) {
        printf("Device file open error!! use sudo ...\\n");
        return 1;
    }

    printf("[UART testing..... loopback]\\n");

    asc = 65;
    while(1) {
        printf("Transmitting ... %d ", asc);
        serialPutchar(fd, asc);

        delay(1);

        if(serialDataAvail(fd)) {
            rec = serialGetchar(fd);
            printf("==> Received : %d  %c\\n", rec, rec);
            serialFlush(fd);
        }

        delay(300);
        asc++;
    }
}
```

```
        return 0;  
    }
```

```
$ make uart_01  
$ sudo ./uart_01 // 필히 sudo로
```

The screenshot shows a terminal window titled 'pi@raspberrypi: ~/IFC413/08_UART'. The window displays a series of transmitted and received characters. The transmitted characters are in uppercase (A-T) and the received characters are in lowercase (a-t). The output is as follows:

```
pi@raspberrypi:~/IFC413/08_UART $ sudo ./uart_01  
[UART testing..... loopback]  
Transmitting ... 65 ===> Received : 65 A  
Transmitting ... 66 ===> Received : 66 B  
Transmitting ... 67 ===> Received : 67 C  
Transmitting ... 68 ===> Received : 68 D  
Transmitting ... 69 ===> Received : 69 E  
Transmitting ... 70 ===> Received : 70 F  
Transmitting ... 71 ===> Received : 71 G  
Transmitting ... 72 ===> Received : 72 H  
Transmitting ... 73 ===> Received : 73 I  
Transmitting ... 74 ===> Received : 74 J  
Transmitting ... 75 ===> Received : 75 K  
Transmitting ... 76 ===> Received : 76 L  
Transmitting ... 77 ===> Received : 77 M  
Transmitting ... 78 ===> Received : 78 N  
Transmitting ... 79 ===> Received : 79 O  
Transmitting ... 80 ===> Received : 80 P  
Transmitting ... 81 ===> Received : 81 Q  
Transmitting ... 82 ===> Received : 82 R  
Transmitting ... 83 ===> Received : 83 S  
Transmitting ... 84 ===> Received : 84 T  
^C  
pi@raspberrypi:~/IFC413/08_UART $
```

주의) 루프백 방식으로 회로 연결한 채 재부팅하지 말 것!, 부팅문제. 2019.7.10.
여러번 수행시도할 것, 깔끔한 실행이 이루어지지 않음.

8.3 Windows PC와의 시리얼통신

Windows PC와 라즈베리파이 보드를 시리얼 케이블로 연결하여, Windows PC와 라즈베리파이 보드간 시리얼 통신에 대해 살펴보자.

8.3.1 Windows PC 환경설정

Windows PC에서 USB 시리얼 포트 활성화 및 환경 설정이 필요하며, 그동안 사용된 PuTTY를 시리얼 통신 프로그램으로 사용하는 것에 대해 살펴보자.

USB Serial Port 설정

Windows의 장치관리자 화면에서 USB Serial 케이블을 연결하면 다음과 같이 포트가 추가된다. 다음 그림에서 보듯이 현재 시스템에서 시리얼 포트는 COM3으로 배정된 것을 확인할 수 있다. 만일 시리얼 포트가 추가되지 않는다면 관련 드라이버를 설치해야 한다.



통신 포트가 인식되었으면 해당 항목을 클릭하여 다음과 같은 속성 창에서 시리얼 통신에 필요한 정보를 설정한다. 실습을 위해 보 레이트 19200, 데이터비트 8, 패리티 없음, 스탶비트 1, 흐름제어 없음으로 설정한다.

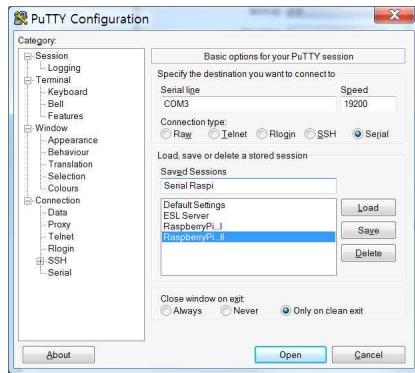


PuTTY 통신 프로그램

이제 시리얼 통신 프로그램으로 PuTTY를 사용하자. 다음 그림과 같은 PuTTY 실

226 라즈베리파이기반 임베디드시스템응용

행 첫 화면에서 선택 사항들 중 연결 유형에서 Serial을 체크하고, 통신포트 및 보레이트를 설정한다. 필요에 따라 세션에 대한 설정 정보를 저장하여 운영할 수 있으며 추후 사용을 위해 적절한 이름으로 세션을 저장한다.



추후 시리얼 통신을 위한 통신 프로그램으로 사용할 때는 저장된 세션 이름을 선택하고, open 버튼을 클릭하면 된다.

8.3.2 실습과제

[실습2] PC와 시리얼통신

Windows PC와에서 송신하는 데이터를 라즈베리파이에서 수신하고 수신한 데이터를 PC로 되 전송하는 시리얼 통신에 대해 살펴보자. baudrate는 19200으로 설정하기로 한다.

```
$ nano uart_02.c
//=====
// uart_02.c
//      with Windows PC
//      retransmit a received char from Win PC
//=====

#include <stdio.h>
#include <string.h>

#include <wiringSerial.h>
```

```
#define BAUD      19200          // 115200

int main(void) {
    int fd;
    int rec;

    if((fd = serialOpen("/dev/ttyS0", BAUD)) < 0) {
        printf("Device file open error!! use sudo ...\\n");
        return 1;
    }

    printf("[UART test with Win PC...]\\n");
    serialPuts(fd, "[UART test with Win PC...]\\n");

    while(1) {
        if(serialDataAvail(fd)) {
            rec = serialGetchar(fd);
            printf("Received from Win PC : %d %c \\n",
                   rec, (char)rec);

            // re-transmit..
            serialPutchar(fd, rec);
            serialFlush(fd);
        }
    }

    return 0;
}
```

\$ make uart_02

Windows PC에서 PuTTY 시리얼통신 프로그램을 실행한 후, 라즈베리파이에서 다음과 같이 실행한다.

\$./uart_02

접속 메시지가 PuTTY 화면에 나타나면, PuTTY 화면에서 문자를 입력한다. 입력한 문자는 라즈베리파이 보드에 전달되어 수신 문자에 대한 정보를 표시하고, 수

228 라즈베리파이기반 임베디드시스템응용

신한 문자를 되 반송함으로써 PuTTY 화면에 입력한 문자가 표시되는 것을 관찰할 수 있다.



[실습3] 시리얼통신에 의한 LED 제어

위의 실습예제를 바탕으로 PC에서 전송하는 문자가 1 혹은 0이나에 따라 LED를 ON 혹은 OFF하는 프로그램을 구현한다. 아래 소스의 볼드체의 부분이 라즈베리파이 보드에 있는 LED를 제어하기 위해 추가된 부분이다.

```
$ nano uart_03.c
//=====
// uart_03.c
//      with Windows PC
//      control LED on Raspberry Pi board
//=====

#include <stdio.h>
#include <string.h>
#include <wiringPi.h>
#include <wiringSerial.h>

#define P_LED    1          // BCM_GPIO #18
#define BAUD    19200        // 115200

int main(void) {
    int fd;
    int rec;
```

```
if((fd = serialOpen("/dev/ttyS0", BAUD)) < 0) {
    printf("Device file open error!! use sudo ...\\n");
    return 1;
}

if(wiringPiSetup() == -1)
    return 1;

pinMode(P_LED, OUTPUT);

printf("[UART test with Win PC + LED control]\\n");
serialPuts(fd, "[UART test with Win PC + LED control]\\n");

while(1) {
    if(serialDataAvail(fd)) {
        rec = serialGetchar(fd);
        printf("Received from Win PC : %d %c  ",
               rec, (char)rec);

        // re-transmit..
        serialPutchar(fd, rec);
        serialFlush(fd);

        //// LED control....
        if(rec == '0') {
            digitalWrite(P_LED, LOW);
            printf("==> Led OFF.....\\n");
        }
        else if(rec == '1') {
            digitalWrite(P_LED, HIGH);
            printf("==> Led ON.....\\n");
        }
        else
            printf("==> No control data.....\\n");
    }
}

return 0;
}
```

실행 결과는 다음과 같으며, 0과 1의 문자를 전송하는 경우, 라즈베리파이 보드의 LED를 OFF, ON하는 것을 관찰할 수 있다.



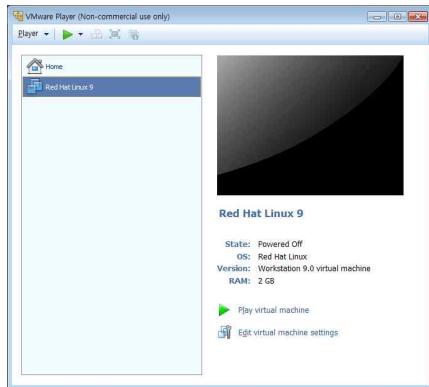
```
pi@raspberrypi:~/IFC413/serial $ ./uart_03
[UART test with PC + LED control]
Received from PC : 97 a ==> No control data.....
Received from PC : 115 s ==> No control data.....
Received from PC : 100 d ==> No control data.....
Received from PC : 49 1 ==> Led ON.....
Received from PC : 97 a ==> No control data.....
Received from PC : 100 d ==> No control data.....
Received from PC : 48 0 ==> Led OFF.....
Received from PC : 65 A ==> No control data.....
Received from PC : 68 D ==> No control data.....
```

8.4 가상머신과의 시리얼통신

8.4.1 가상 머신에 시리얼 포트 추가

우선 가상 머신이 설치된 Windows 환경에서 장치관리자를 통해 시리얼 포트가 무엇으로 설정되어 있는지 확인한다. 기본적으로 시리얼 통신 포트는 COM1일 것이나, USB-Serial Adaptor를 사용하는 경우는 시리얼 통신 포트가 다른 형태의 COMx일 것이다.

이제 가상 머신에 시리얼 통신 포트를 추가하는 과정을 진행한다. 우선 VM player를 실행하고 다음 그림의 좌측 화면에서 사용할 가상 머신을 선택하고, 우측 화면의 settings 항목을 클릭하거나, 메뉴의 “Player-Manage-VM settings”을 클릭한다.



나타난 창의 "Hardware탭"에서 "Add" 클릭한다. 그리고 Serial Port를 추가하고, Windows 환경에서 살펴본 통신 포트 명(COM1)을 설정한다. 이로써 가상머신에 시리얼 포트를 추가하는 것이 완료된다.

8.4.2 리눅스용 시리얼 통신 프로그램

리눅스에서 제공하는 시리얼 통신 프로그램으로 minicom 패키지가 있다. 가상 머신에서 라즈베리파이 보드와의 시리얼 통신을 위해 minicom 패키지를 이용한다. minicom은 텍스트 방식의 터미널 애플리케이션 통신 프로그램으로, Windows의 hyper terminal과 유사한 기능을 한다.

우선 다음의 명령을 사용하여 minicom 패키지를 설치한다.

```
root@ubuntu:~# apt-get update
root@ubuntu:~# apt-get upgrade
root@ubuntu:~# apt-get install minicom
```

minicom 환경설정

minicom을 사용할 때 시리얼 통신포트, 시리얼 통신속도 등의 환경 설정을 위해 다음 명령과 같이 -s 옵션을 사용한다. 그러면 다음의 화면이 나타난다.

```
root@ubuntu:~# minicom -s
```



위의 화면에서 'Serial Port Setup' 항목을 선택하고, 그 하부 화면에서 다음과 같이 각 항목을 설정한다. Serial Device 항목에는 가상 머신에 시리얼 포트를 추가할 때 설정된 정보가 시리얼포트1이면 /dev/ttyS0로, 시리얼포트2면 /dev/ttyS1으로 설정한다. Bps/Par/Bits 항목은 통신 속도 및 패리티 비트 등 설정하는 부분이고, Hardware Flow Control 항목은 No로 설정한다.

A-Serial Device	: <u>/dev/ttyS1</u>
B-Lockfile Location	: <u>/var/lock</u>
E-Bps/Par/Bits	: <u>115,200bps 8N1</u>
F-Hardware Flow Control	: <u>No</u>

설정된 시리얼 통신 포트의 환경설정 정보를 저장하기 위해 'Save setup as dfl' 항목을 선택한다. 이후 minicom을 실행하게 되면 저장된 정보를 사용하여 자동 환경 설정하여 실행되게 한다. 환경 설정을 마치기 위해 'Exit from minicom' 항목을 선택한다.

이후부터 다음과 같이 시리얼 통신 프로그램을 실행하면 앞서 설정되어 저장된 환경설정 정보 파일을 이용하여 환경 설정하여 실행된다.

```
root@ubuntu:~# minicom
```

minicom 내부 명령어가 몇 가지 제공되는데, 'ctrl+a'한 후 z를 누르면 다음과 같은 화면을 볼 수 있으며, 가용한 명령어의 도움말을 얻을 수 있다.

```

root
ash:
root          Minicom Command Summary
root
root          Commands can be called by CTRL-A <key>
root
root          Main Functions           Other Functions
root
ash: Dialing directory..D  run script (Go)....G | Clear Screen.....C
root Send files.....S  Receive files.....R | cOnfigure Minicom..O
root comm Parameters....P Add linefeed.....A | Suspend minicom....J
in Capture on/off.....L Hangup.....H | eXit and reset....X
onf send break.....F initialize Modem...M | Quit with no reset.Q
root Terminal settings..T run Kermit.....K | Cursor key mode....I
root lineWrap on/off....W local Echo on/off..E | Help screen.....Z
root
root          Select function or press Enter for none.■
onf

```

8.4.3 실습과제

[실습4] 시리얼통신에 의한 LED 제어

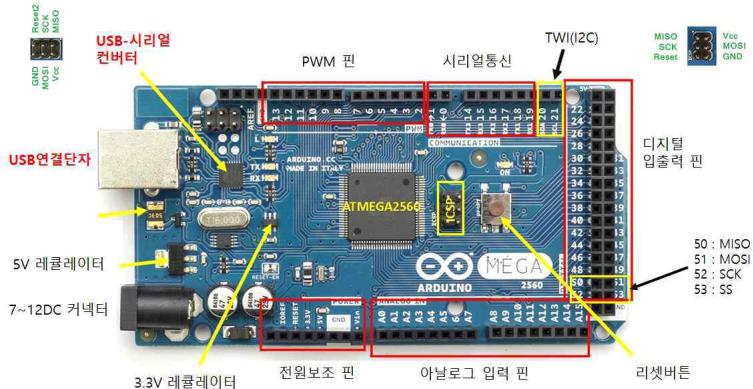
가상 머신에서 전송하는 문자가 1 혹은 0이냐에 따라 라즈베리파이 보드의 LED를 ON 혹은 OFF하는 프로그램을 구현한다. 앞의 [실습3]의 소스를 그대로 사용하면 된다.

8.5 Arduino 보드와의 시리얼통신

8.5.1 Arduino Mega ADK 보드

아두이노 메가 ADK 보드에는 다음 그림에서 보듯이 USB-시리얼 컨버터가 내장되어 있다. 스케치에서 아두이노 소스를 작성하고 컴파일하여 목적 코드를 업로드한 것은 이러한 USB-시리얼 통신을 이용한 것이다.

234 라즈베리파이기반 임베디드시스템응용

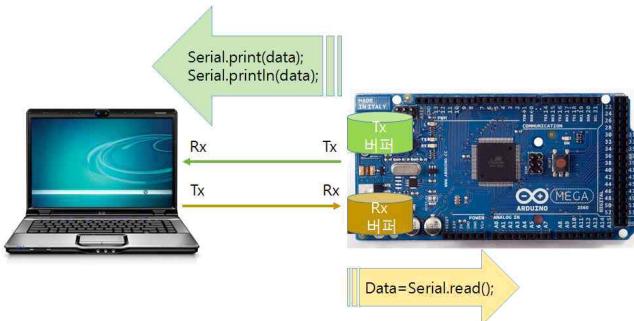


아두이노 메가 ADK 보드에는 아래 표와 같이 총 4개의 시리얼 포트를 제공하며, 컴퓨터 USB와 아두이노 시리얼 포트간 통신에 사용되는 시리얼 포트는 포트 번호 0이 사용된다.

시리얼변수명	포트번호	Tx 핀	Rx 핀	
Serial	0	1	0	Serial0가 아님
Serial1	1	18	19	
Serial2	2	16	17	
Serial3	3	14	15	

아두이노 시리얼 통신을 사용하여 외부 컴퓨터 또는 다른 통신 장치와 연결하기 위해서는 송수신 선을 연결해주고 통신 속도를 맞추어 주어야 한다. 송수신 선의 연결은 아두이노 시리얼 포트의 Tx 핀은 다른 장치의 Rx 핀을 연결하고, 시리얼 포트의 Rx 핀은 다른 장치의 Tx 핀과 연결하는 식으로 교차 연결해야 한다. 또한 시리얼 통신 속도는 Serial.begin() 함수를 사용하여 설정할 수 있다.

시리얼 통신 동작 원리는 시리얼 통신을 하는 송수신기 간에 데이터 전송속도나 처리속도 시간차를 해결하기 위하여 다음 그림과 같이 데이터를 쌓아두는 버퍼를 사용한다. 시리얼 통신으로 송신할 때는 Serial.print() 함수를 사용하며, 이 경우 데이터는 전송 버퍼에 들어가게 되며, 수신시는 Serial.read() 함수를 사용하여 수신버퍼에서 읽어 오게 된다.



아울러, 아누이도 IDE 창의 우측 상단에 있는 시리얼모니터 아이콘을 클릭하면 시리얼 모니터 화면이 나타나, 시리얼 데이터의 송수신과정을 모니터링할 수 있다.

8.5.2 Arduino 보드용 소스 (LED 제어)

아두이노 보드에 위치한 LED를 시리얼 통신에 의해 원격의 시스템에서 제어하는 것을 수행하려 한다.

회로 구성

이를 위해 아두이노 보드의 디지털 핀 번호 3에 LED 회로를 다음과 같이 연결한다.



Arduino 보드용 소스

원격의 시스템으로부터 시리얼 통신 포트로 문자를 수신하고, 이 문자가 '0'이면 LED를 OFF하고, '1'이면 LED를 ON하는 기능의 프로그램으로, 데이터의 수신은 serialEvent() callback 함수를 사용한 것이다. 통신 속도는 19200으로 설정한 것이다.

236 라즈베리파이기반 임베디드시스템응용

스케치에서 다음의 소스를 작성한 후, 컴파일한다.

```
//=====
// uart_04_arduino.ino
//     for Arduino board
//     LED controlled by remote RaspberryPi
//=====
const int ledPin = 3;           // LED에 연결된 핀 번호

char data;                     // 수신 문자를 저장하기 위한 변수

void setup() {
    pinMode(ledPin, OUTPUT);   // LED를 출력으로 설정
    Serial.begin(19200);       // 시리얼 개방, 통신속도 설정
}

void loop() {
    switch(data) {
        case '0' :
            digitalWrite(ledPin, LOW);      // LED OFF
            break;
        case '1' :
            digitalWrite(ledPin, HIGH);     // LED ON
            break;

        default :
            break;
    }
}

// 수신버퍼에 데이터가 있을 때 마다 호출되는 콜백 함수
void serialEvent() {
    data = Serial.read();
    Serial.println(data);
}
```

컴파일 후 목적코드를 아두이노 보드의 플래쉬 메모리에 기록하는 업로드 작업을 진행한다. 목적코드를 업로딩하려면 다음의 2가지 작업이 선행되어야 한다. 메뉴의 도구 항목중 보드와 시리얼포트 항목에서 아두이노 보드의 모델 설정과 ISP를

위한 시리얼 포트를 설정해 주어야 한다. 이 후 업로드 아이콘을 클릭하면 된다.

다음으로 아두이노 소스를 개발한 PC로부터 아두이노 보드 및 USB 케이블을 제거한다. 현재 아두이노 보드의 플래쉬 메모리에는 직전에 업로드한 목적코드가 기록되어 있으며, 전원이 인가되면 언제든 해당 목적코드가 실행될 수 있는 상태이다.

8.5.3 실습과제

[실습5] 아두이노 보드의 LED 제어

시리얼 통신에 의해 아두이노보드 상의 LED를 라즈베리파이 보드에서 제어하는 프로그램이다.

라즈베리 파이 보드용 소스

```
$ nano uart_04.c
//=====
// uart_04.c
//      Arduino LED Control
//      for Raspberry Pi board
//      pair file : uart_04_arduino.ino
//=====

#include <stdio.h>
#include <string.h>

#include <wiringSerial.h>

#define BAUD 19200

int main(void) {
    int fd;
    char con:

    if((fd = serialOpen("/dev/ttyS0", BAUD)) < 0) {
        printf("Device file open error!! use sudo ...\\n");
        return 1;
    }
```

```

printf("[UART test with Arduino board]\n");
serialPuts(fd, "[UART test with Arduino board]\n");

while(1) {
    printf("\nInput a char: ");
    con = getchar();
    getchar();           // trash enter key
    //fflush(stdin);

    //serialPutchar(serialGetchar(fd));
    // transmit
    serialPutchar(fd, con);
    serialFlush(fd);
}

return 0;
}

```

\$ make uart_04

라즈베리파이 보드와 아두이노 보드간 시리얼 통신을 위해 두 보드를 연결한다. 이 때 두 보드간 RxD, TxD 단자를 교차하여 연결한다. 이로써 두 보드간 하드웨어 연결은 완료되었다. 그런 후 아두이노 보드에 전원을 인가하기 위해 아두이노 보드에 연결된 USB 잭을 라즈베리파이 보드의 USB 잭에 연결한다. 전원이 인가된 아두이노 보드에서는 시리얼 통신에 의해 자신의 LED를 제어하는 프로그램이 실행 중인 상태이다.

이제, 라즈베리 파이 환경에서 다음과 같이 아두이노 보드와 시리얼 통신하는 프로그램을 실행한다. 실행 화면에서 아두이노 보드의 LED를 제어하기 위한 0, 혹은 1의 문자를 입력한다. 입력 문자가 1인 경우 아두이노 보드의 LED가 ON, 0인 경우 OFF되는 것을 관찰할 수 있다.

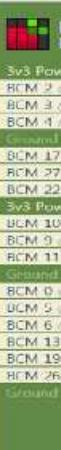
\$./uart_04

```
pi@raspberrypi:~/IFC413/serial $ ./uart_04
[UART test with Arduino board]

Input a char: a
Input a char: 1
Input a char: 0
Input a char: █
```

참고자료

- [1] USB-TTL 케이블활용한 시리얼통신 <https://tibyte.kr/258>
- [2] USB-TTL 케이블활용한 시리얼통신 <https://iotea.tistory.com/1>
- [3] 루프백 시리얼 통신 <http://embejied.tistory.com/40>
- [4] 시리얼 통신으로 모니터없이 접속하기 <https://kamang-it.tistory.com/226>
- [5] PL-2303이용한 PC와의 시리얼 통신 <http://cccding.tistory.com/93>
- [6] 시리얼 통신
<http://blog.naver.com/PostView.nhn?blogId=leesoo9297&logNo=221111871607>
- [7] 시리얼 통신 <http://creamp.tistory.com/7>
- [8] <http://hongci.tistory.com/8>
- [9] 시리얼 통신
<http://blog.naver.com/PostView.nhn?blogId=erroring&logNo=221009126246>
- [10] 아두이노교재, 13장 시리얼 통신, 김한종
- [11] KUT15 보드 시리얼 통신
https://cms3.koreatech.ac.kr/sites/joo/IFC181/IFC181_14.pdf



제9장 1-wire 인터페이스

디바이스간 하나의 신호선을 사용하여 신호를 송수신하는 1-wire 인터페이스에 대해 살펴본다.

9.1 1WI(1-wire interface)

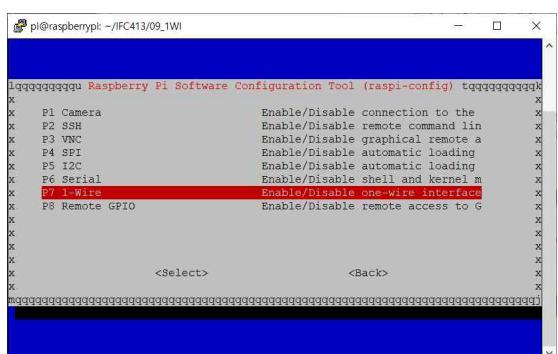
하나의 신호선을 사용하여 신호를 입력 및 출력하기 위한 방법으로 1-wire 인터페이스가 사용된다. 특히 디지털 온습도 모듈인 DHT11 모듈은 1-wire 인터페이스로 제어한다.

9.1.1 1-Wire 활성화

1-wire interface 방법을 사용하려면 다음의 raspi-config 명령으로 1-Wire 인터페이스가 활성화되도록 환경 설정해야 한다.

```
$ sudo raspi-config
```

아래와 같은 화면에서 Interfacing Options 항목을 선택하고, 그 서브 화면에서 1-Wire 항을 선택하여 1-wire 인터페이스가 활성화되도록 설정한다.



이를 반영하기 위해 다음의 명령을 사용하여 재부팅한다.

```
$ sudo reboot
```

부팅될 때 활성화되도록 설정되었는지 확인을 위해 다음의 명령을 사용하여, dtoverlay=w1-gpio 라인이 추가되어 있는지 살펴본다.

```
$ sudo cat /boot/config.txt
```

9.1.2 1-Wire 인터페이스

1-wire 인터페이스는 하나의 데이터 버스로 송/수신이 가능한 통신 방식이라 Single-Wire Two-Way 통신 방식이라고도 한다.

데이터 포맷

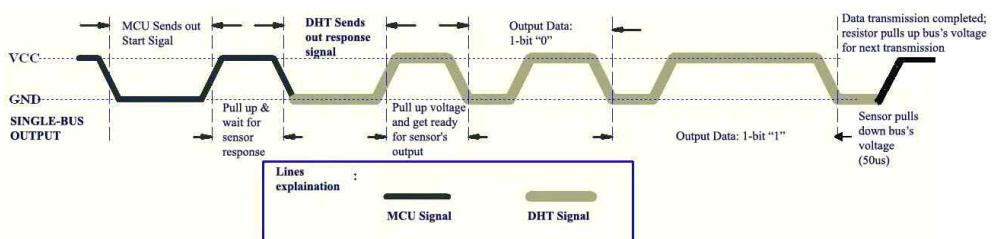
하나의 데이터 버스를 통한 I-W 터페이스 통신에서 데이터 형식은 MCU와 DHT11 센서 간의 통신 및 동기화에 사용된다. 통신 개시부터 데이터 전송까지 완료하는 하나의 통신 과정에는 약 4ms가 소요된다.

완전한 데이터 형식은 다음 그림과 같이 5바이트(40 비트)로 구성되며, DHT 센서는 상위 데이터 비트부터 먼저 전송한다. 온습도 데이터는 각 습도 데이터 2바이트와 온도 데이터 2바이트이며, 이들은 각각 1바이트씩의 정수 부분과 소수이하 부분으로 구성된다. 그리고 이들 4바이트에 대해 바이트단위로 합산한 결과인 checksum 1바이트가 추가된다. 이 checksum은 합산 결과의 하위 바이트이며, 신호의 송수신 오류검증에 사용된다.

byte 4	byte 3	byte 2	byte 1	byte 0
RH integral	RH decimal	T integral	T decimal	checksum

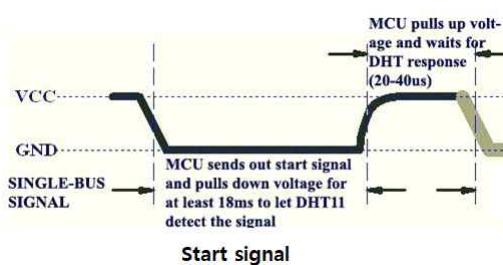
전체적인 통신 절차

전체적인 1WI의 통신절차는 다음 그림과 같다. MCU가 통신 개시 신호를 보내게 되면 DHT11은 저전력 소비 모드에서 실행 모드로 변경되어 MCU가 개시 신호를 완료하기를 기다린다. 그런 후, DHT는 통신 개시 신호의 응답 신호를 보낸다. 이후 DHT11은 상태 습도 및 온도 정보를 포함하는 40비트의 데이터를 MCU에 보내고, MCU로부터 다시 통신 개시 신호를 수신할 때까지 저전력 소비 모드에 있게 된다.



1) 통신 개시 신호 전송

하나의 데이터 버스의 초기 idle 상태는 High 레벨 신호를 갖는다. MCU가 1WI 통신을 시작하려면 데이터 버스에 Low 레벨 신호를 최소 18ms동안 유지하여야 한다. 이는 DHT11 센서가 통신을 개시하려는 신호임을 탐지할 수 있도록 보장한다. 그런 후 MCU는 데이터 버스에 High 레벨 신호를 인가하고, DHT11으로부터 응답을 기다린다. DHT가 응답하는데 소요되는 시간은 20~40us정도이다.



MCU가 DHT로 통신 개시 신호를 전송하는 과정을 코드로 구현하면 다음과 같다.

```
// 1) send start signal
pinMode(P_DHT_DAT, OUTPUT);
```

```

// 1wire init- to HIGH
digitalWrite(P_DHT_DAT, HIGH);
delay(1);

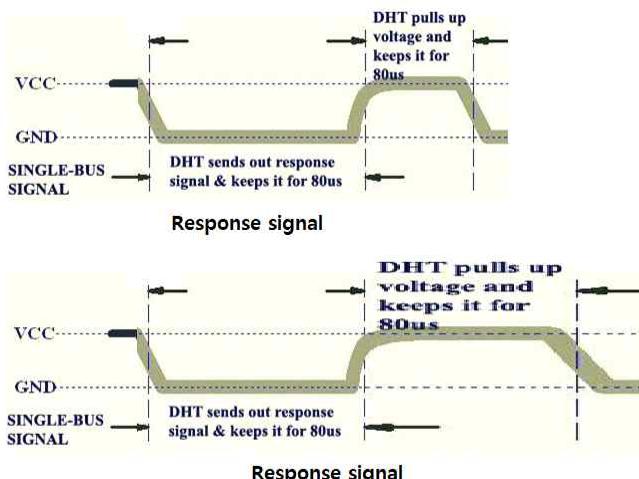
// MCU sends out start signal & low signal for at least 18ms
// to let DHT11 detect the signal
digitalWrite(P_DHT_DAT, LOW);
delay(18);

// MCU pulls up voltage and waits for DHT response(20~40us)
digitalWrite(P_DHT_DAT, HIGH);

```

2) 통신 개시 신호에 대한 응답 신호

DHT가 통신 개시 신호를 감지하면, 그 응답 신호로 DHT는 데이터 버스에 Low 레벨 신호를 보내고 이를 80us동안 유지한다. 그런 후 다시 DHT는 데이터 버스에 High 레벨 신호를 보내고 이를 80us동안 유지한다. 이 동안 DHT는 온습도 데이터를 전송하기 위한 준비를 한다.



MCU의 통신 개시 신호에 대한 DHT의 응답 신호 수신과정을 코드로 구현하면 다음과 같다.

```
pinMode(P_DHT_DAT, INPUT);
```

```

// check : high -> low
while(digitalRead(P_DHT_DAT) == HIGH)           // wait
;

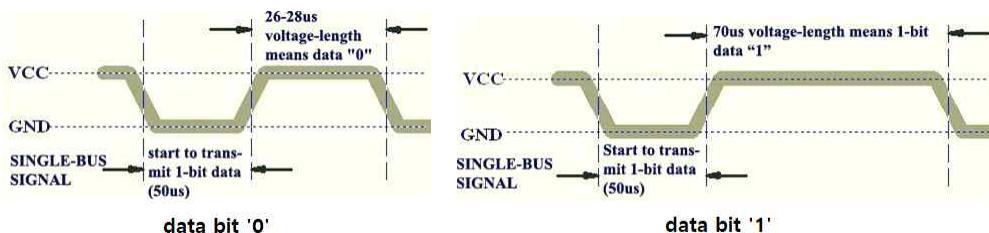
// 2) DHT sends out response signal & keeps it for 80us
// check : low -> high
while(digitalRead(P_DHT_DAT) == LOW)            // wait
;

// DHT pulls up voltage & keeps it for 80us
// check : high -> low
while(digitalRead(P_DHT_DAT) == HIGH)
;

```

3) 데이터 비트 전송 신호

DHT는 온습도 데이터를 비트 단위로 데이터 버스를 통해 전송한다. 이 때 각 데이터 비트는 50us동안의 Low 레벨 신호를 앞 세운다. 그런 후 아래 그림과 같이 전송할 데이터 비트에 따라, 비트 0인 경우 26~28us 동안의 High 레벨 신호를, 비트 1인 경우는 70us 동안의 High 레벨 신호를 전송한다. 이 과정은 전체 데이터가 40비트이므로, 40회 반복 된다.



데이터 비트를 전송하는 과정을 코드로 구현하면 다음과 같다. 데이터 비트 0과 1의 구분은 High 레벨 신호의 시간(us)에 대해 임계치를 적용하여 구분도록 구현한 것이다.

```

for(bit=0; bit<MAX_BIT; bit++) {
    // Start to transmit 1-bit data (50us)
    while(digitalRead(P_DHT_DAT) == LOW)
;

```

```

// 26-28us voltage-length means data "0"
// 70us voltage-length means 1-bit data "1"
st = micros();
while(digitalRead(P_DHT_DAT) == HIGH)
    delayMicroseconds(1);
en = micros();

// get a bit data
dht11Res[bit/8] <= 1;           // bit shift
duration = en - st;
//printf(" %d \n", duration);
if(duration > THRESHOLD)        // us, threshold
    dht11Res[bit/8] |= 1;         // bit value 1
else
    dht11Res[bit/8] |= 0;         // bit value 0
}

```

참고로 DHT의 응답 신호가 항상 High 레벨 신호인 경우, DHT가 제대로 응답하지 않음을 의미하므로 연결을 확인할 필요가 있다.

마지막 데이터 비트가 전송된 후에 DHT11은 Low 레벨의 신호를 50us 동안 유지함으로써 데이터 전송을 완료하였음을 표시한다. 그런 후 그 다음의 통신을 위한 idle 상태를 위해 자체 저항에 의해 데이터 버스에는 High 레벨 신호가 부여된다.

9.2 온습도 센서 모듈

9.2.1 DHT11 온습도 센서 모듈

다음 그림은 DHT11 모듈을 보여준다.



DHT11(Digital Humidity an Temperature Sensor) 온습도 모듈은 정전식 습도 센서와 온도 센서를 사용하여 대기 중의 습도와 온도를 측정하고 디지털 신호를 출력한다. 이 모듈은 3.3V-5V에서 동작되며, 습도 범위는 20% ~ 95%이고 오차범위는 +/- 5%이내이며, 온도 측정 범위는 0°C ~ 50°C이고 오차범위는 +/- 2°C이다. 1-wire 인터페이스를 사용하므로 그 타이밍에 유의하여 제어해야 한다. DHT11은 새로운 값을 얻는데 최소 2초의 시간이 요구된다. DHT22 센서에 비해 정확도는 떨어지나 저가이다.

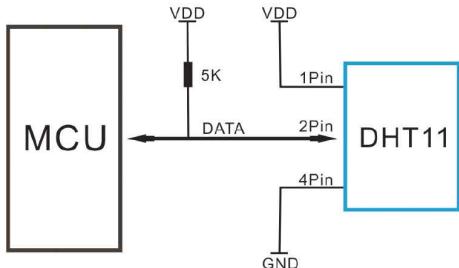
다음의 사이트에서 DHT 모듈을 위한 라이브러리가 제공된다.

<https://github.com/adafruit/DHT-sensor-library>

한 단자를 통해 신호를 출력 및 입력해야 하며, 1-Wire 인터페이스 방식으로 제어 한다.

회로구성

회로 구성을 위해 GPIO 핀은 아무거나 사용하여 다음과 같이 연결한다. 소스에서 는 #7(BCM_GPIO #04)을 사용한다.



9.2.2 실습과제

[실습1] DHT11 모듈

DHT11 온습도 모듈을 1-wire 인터페이스로 제어하여 상대 온습도를 출력하는 프로그램이다. GPIO 편은 어느 것을 사용하여도 좋다.

```

$ nano dht11_01.c
//=====
// dht11_01.c
//      DHT11 module by one-wire interface
//=====

#include <stdio.h>
#include <stdlib.h>
#include <wiringPi.h>

#define P_DHT_DAT      7      // BCM_GPIO #04, one-wire interface

#define MAX_BIT         40     // DHT11 data -> 40bit
#define THRESHOLD       45     // bit 0 : 20us+, bit 1 : 70us+
#define PASS            0
#define ERROR           1     // checksum error

// DHT11 functions
void startSignalToDHT(void);
void responseSignalFromDHT(void);
int readDHT11(void);

// Response data
// : integral RH, decimal RH,  integral T, decimal T, check sum

```

```

unsigned char dht11Res[5];

int main(void) {
    int cnt, ret;
    int i;

    printf("[DHT11 testing..... by 1-Wire interface]\n");

    if(wiringPiSetup() == -1)
        return 1;

    cnt = 0;
    while(1) {
        startSignalToDHT();           // start signal
        responseSignalFromDHT();     // response signal
        ret = readDHT11();           // read DHT data

        printf("[%d] ", cnt);
        for(i=0; i<5; i++)
            printf("%d, ", dht11Res[i]);
        printf("..... ");
        if(ret == PASS)
            printf("RH : %d.%d%%, Temp : %d.%d'C\n",
                   dht11Res[0], dht11Res[1], dht11Res[2], dht11Res[3]);
        else
            printf("Data Error!!!\n");

        delay(2000);                 // 2sec for DHT11 sensor
        cnt++;
    }
    return 0;
}

// MCU Sends out Start Signal to DHT
void startSignalToDHT(void) {

    // 1) send start signal
    pinMode(P_DHT_DAT, OUTPUT);

    // 1wire init- to HIGH
    digitalWrite(P_DHT_DAT, HIGH);
    delay(1);
}

```

```
// MCU sends out start signal & low signal for at least 18ms
// to let DHT11 detect the signal
digitalWrite(P_DHT_DAT, LOW);
delay(18);

// MCU pulls up voltage and waits for DHT response(20~40us)
digitalWrite(P_DHT_DAT, HIGH);
}

// DHT Sends out Response Signal to MCU
void responseSignalFromDHT(void) {

    pinMode(P_DHT_DAT, INPUT);

    // check : high -> low
    while(digitalRead(P_DHT_DAT) == HIGH)           // wait
        ;

    // 2) DHT sends out response signal & keeps it for 80us
    // check : low -> high
    while(digitalRead(P_DHT_DAT) == LOW)           // wait
        ;

    // DHT pulls up voltage & keeps it for 80us
    // check : high -> low
    while(digitalRead(P_DHT_DAT) == HIGH)
        ;
}

// read DHT datas
int readDHT11(void) {
    unsigned char counter;
    unsigned char bit, checksum;
    unsigned int st, en, duration;
    int i;

    // init
    for(i=0; i<5; i++)
        dht11Res[i] = 0;

    // Start data transmission
```

```

for(bit=0; bit<MAX_BIT; bit++) {
    // Start to transmit 1-bit data (50us)
    while(digitalRead(P_DHT_DAT) == LOW)
        ;
    // 26-28us voltage-length means data "0"
    // 70us voltage-length means 1-bit data "1"
    st = micros();
    while(digitalRead(P_DHT_DAT) == HIGH)
        delayMicroseconds(1);
    en = micros();

    // get a bit data
    dht11Res[bit/8] <=> 1;           // bit shift
    duration = en - st;
    printf(" %d \n", duration);
    if(duration > THRESHOLD)         // us, threshold
        dht11Res[bit/8] |= 1;          // bit value 1
    else
        dht11Res[bit/8] |= 0;          // bit value 0
}

// checksum
checksum = 0;
for(i=0; i<4; i++)
    checksum += dht11Res[i];

if(checksum == dht11Res[4])
    return PASS;
else
    return ERROR;
}

$ make dht11_01
$ ./dht11_01

```

실행중에 입김을 불어 온습도 변화되는 것을 관찰한다.

252 라즈베리파이기반 임베디드시스템응용

참고자료

[1] DHT11 제어

<http://server-engineer.tistory.com/admin/entry/post/?id=390>

3v3 Pow
BCM 2
BCM 3
BCM 4
Ground
BCM 17
BCM 27
BCM 22
3v3 Pow
BCM 10
BCM 5
BCM 11
Ground
BCM 0
BCM 5
BCM 6
BCM 13
BCM 19
BCM 26
Ground

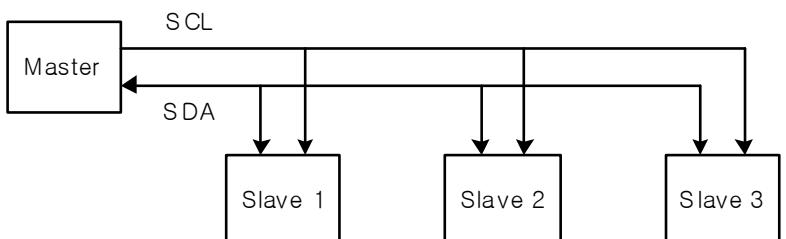
제10장 I²C 인터페이스

라즈베리파이의 GPIO는 아두이노 보드와는 다르게 디지털 신호의 입출력만 가능하도록 되어있다. 또한 라즈베리파이는 아날로그 신호원의 디바이스를 취급하기 위한 ADC(Analog Digital Converter)가 내장되어있지 않다. 따라서 아날로그 신호의 디바이스를 취급하기 위해서는 별도의 AD 변환기를 사용해야 한다. 그리고 일반적으로 AD 변환기는 I²C 인터페이스 통신을 통해 데이터를 확보할 수 있다.

10.1 I²C 인터페이스

I²C(Integrated circuit) 인터페이스는 IC들 간의 데이터 통신을 목적으로 하는 통신방식으로 2개 신호선을 사용하여 통신톤록 정의하고 있다. 이들 신호선은 SDA(Serial data)와 SCL(Serial Clock)이다. 이 통신 방식은 회로가 간단하고, 비용이 저렴하며, 통신과정에서 잡음이 적은 등의 장점을 가진다.

I²C 통신을 위해 다음 그림과 같이 송/수신측은 하나의 마스터(master)와 다수의 슬레이브(slave)로 구성될 수 있다. 마스터가 어느 슬레이브와 통신을 할 것인지를 위해 각 슬레이브에는 주소가 부여된다.



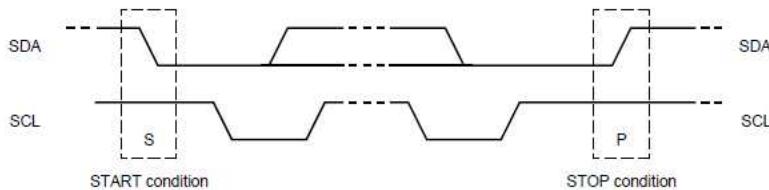
10.1.1 시작 조건과 종료 조건

버스가 사용되지 않을 때 SDA와 SCL 라인엔 High 신호가 유지된다. 데이터 전송

254 라즈베리파이기반 임베디드시스템응용

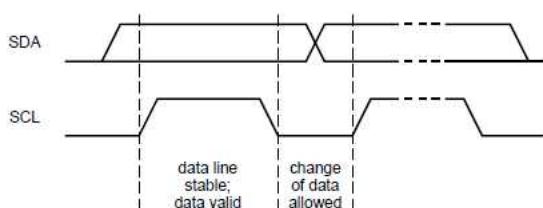
은 버스가 사용되지 않을 때만 개시될 수 있다. I2C 통신을 위해서는 항상 시작 조건으로 개시되어야 하고 종료 조건으로 끝나야 하며, 데이터는 이 두 조건의 사이에 전송된다.

다음 그림에서 보듯이 시작(Start) 조건은 마스터가 슬레이브에게 I2C 통신을 개시하겠다는 의미의 상태를 나타내는 것으로, SCL이 High인 동안 SDA에서의 High 레벨에서 Low 레벨로의 천이 신호로 정의된다. 반면 종료(Stop) 조건은 마스터가 슬레이브에게 통신을 종료하겠다는 의미를 나타내는 상태이며, SCL이 High인 동안 SDA에서의 Low 레벨에서 High 레벨로의 천이 신호로 정의된다.



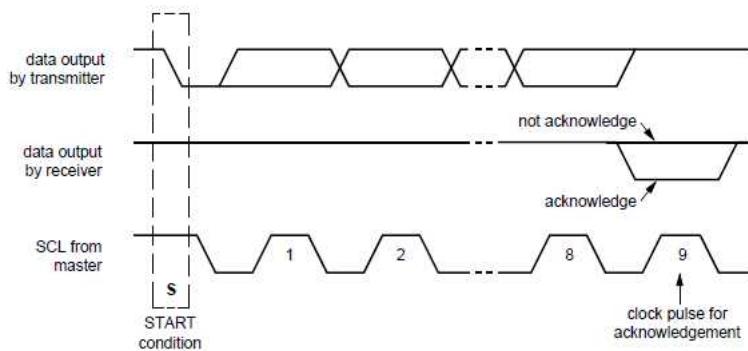
10.1.2 데이터 비트의 전송

시작 조건과 종료 조건의 사이에서 데이터들이 전송되며, 전송되는 데이터 바이트의 수는 무제한이다. 하나의 데이터 비트는 매 클럭 펄스동안 전송될 수 있다. 다음 그림과 같이 SDA 상의 데이터 비트는 SCL이 High 레벨 신호인 동안 안정적으로 유지되어야 한다. 만일 SCL이 High 레벨인 동안 SDA에서의 신호 변화는 제어 신호로 해석될 수 있다.



10.1.3 ACK 신호

송신측에서 수신측으로 전송되는 데이터의 바이트마다 수신측으로부터 ACK 신호를 받아야 한다. 즉, 수신측 디바이스는 매 1바이트의 데이터를 수신한 후에는 ACK 신호를 생성해 송신측으로 전송해야 한다. 다음 그림과 같이 ACK 신호를 전송하는 디바이스는 SCL이 High 신호인 동안 안정적으로 SDA에 Low 레벨의 신호를 주어야 하는 것이다.



마스터는 슬레이브로부터의 마지막 바이트에 대한 ACK 신호를 생성하지 않음에 의해 전송측에 데이터의 끝임을 신호해야 한다. 이러한 상황에서 마스터가 종료 조건을 생성할 수 있도록 전송측은 SDA에 High 신호를 주어야 한다.

10.1.4 I2C 버스 프로토콜

시작 조건이 전송된 후에는 통신할 슬레이브의 주소 정보가 첫 데이터 바이트로 전송되어어야 한다.

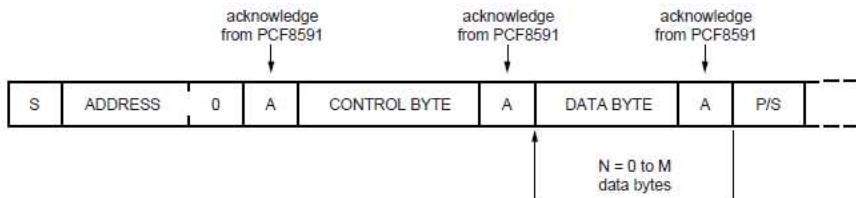
PCF8591 디바이스의 슬레이브 주소지정 포맷은 다음과 같다. 주소 비트로 3비트가 사용되므로 8개까지의 슬레이브 디바이스를 연결할 수 있다. 또한 슬레이브 주소 바이트의 LSB 비트는 R/W용으로 0이면 기록 모드, 1이면 판독 모드를 나타낸다.

256 라즈베리파이기반 임베디드시스템응용

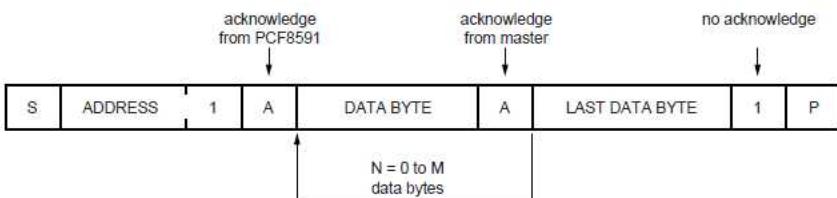
7	6	5	4	3	2	1	0
1	0	0	1	A2	A1	A0	R/W

슬레이브 디바이스의 주소지정은 입력 편 A2, A1, A0를 Vcc 혹은 Gnd에 연결함으로써 정의할 수 있으며, 동일한 I2C 버스 상에서 8개까지의 PCF8591를 구별할 수 있도록 주소지정할 수 있다.

기록 모드를 위한 통신 포맷은 다음과 같다.



판독 모드를 위한 통신 포맷은 다음과 같다. 슬레이브는 마지막 바이트 전송 후에는 ACK 신호를 받지 않고 마스터가 종료 조건을 진행할 수 있도록 한다.



10.1.5 wiringPiI2C.h 헤더파일

wiringPi는 I2C 인터페이스 통신을 위한 라이브러리를 제공한다.

I2C 통신을 위한 슬레이브 주소를 지정하여 초기화하는 함수로 다음의 함수들이 있다. devID는 i2cdetect 명령을 통해 얻은 슬레이브 디바이스의 I2C 주소이며, device가 명시된 경우는 /dev 디렉터리 내의 디바이스파일을 개방하는 반면, 명시

되지 않은 경우 라즈베리파이 버전을 확인하여 적절한 디바이스파일을 개방한다.

```
extern int wiringPi2CSetupInterface(const char *device, int devId) ;
extern int wiringPi2CSetup(const int devId) ;
```

슬레이브 디바이스에 제어 바이트 등을 전송하는 함수로 다음과 같은 함수들이 제공된다. 일부 디바이스는 레지스터를 지정할 필요 없이 단순히 데이터를 전송하는 것이 가능한데, 이 경우는 첫 번째 함수를 사용한다. 나머지 함수는 레지스터를 지정하여 1바이트 혹은 2바이트를 전송할 때 사용된다.

```
extern int wiringPi2CWrite(int fd, int data) ;
extern int wiringPi2CWriteReg8(int fd, int reg, int data) ;
extern int wiringPi2CWriteReg16(int fd, int reg, int data) ;
```

슬레이브 디바이스로부터 데이터를 수신 받는 함수로 다음과 같은 함수들이 제공된다. 일부 디바이스는 레지스터를 지정할 필요 없이 단순히 데이터를 수신하는 것이 가능한데, 이 경우는 첫 번째 함수를 사용한다. 나머지 함수는 레지스터를 지정하여 1바이트 혹은 2바이트를 읽어 들일 때 사용된다.

```
extern int wiringPi2CRead(int fd) ;
extern int wiringPi2CReadReg8(int fd, int reg) ;
extern int wiringPi2CReadReg16(int fd, int reg) ;
```

10.2 ADC/DAC 모듈

10.2.1 ADC/DAC(YL-40) 모듈

ADC/DAC(YL-40) 모듈의 외양은 다음과 같으며, ADC/DAC 기능의 핵심소자인 PCF8591 IC가 장착되어 있다. 이 모듈은 4개의 아날로그 디바이스를 연결할 수 있으며, 하나의 아날로그 신호 출력 단자가 제공된다.



ADC/DAC(YL-40) 모듈상의 2개 LED 중 빨강 LED는 전원투입 여부를, 초록 LED는 출력전압에 비례하는 밝기를 나타낸다. 또한 대표적인 아날로그 신호를 발생시키는 소자인 조도센서(AIN0), 온도센서(AIN1), 10K옴의 가변저항(AIN3)이 장착되어 있으며, 이들 소자의 연결을 위해 3개의 점퍼선 P5(AIN0), P4(AIN1), P6(AIN3)이 구비되어 있다. 즉, 조도 센서의 아날로그 신호에 대한 ADC를 위해서는 점퍼선 AIN0를 통해 연결하고, 이 때의 신호 채널은 AIN0가 된다. 장착된 아날로그 디바이스들에 대한 내정된 채널과 점퍼선을 정리하면 다음 표와 같다.

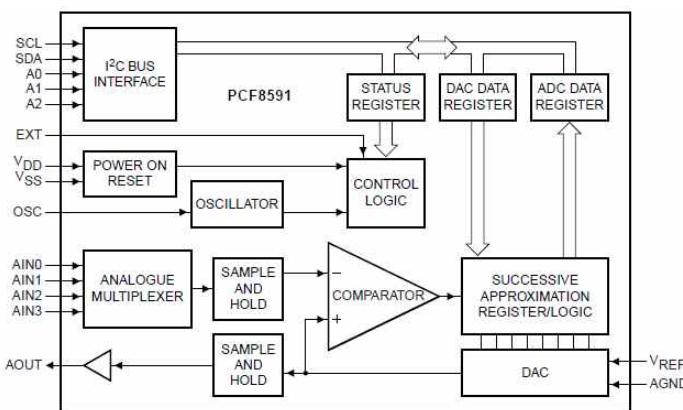
채널	내정디바이스	점퍼선	비고
AIN0	조도센서	P5	
AIN1	온도센서	P4	
AIN2	-	-	
AIN3	가변저항	P6	

그리고 모듈의 좌측에는 하나의 아날로그 출력 채널 AOUT과 4개의 아날로그 입력 채널을 연결할 수 있는 핀 AIN0, ..., AIN3이 있다. 다른 아날로그 신호의 디바이스는 이 핀에 연결하여 시험할 수 있다. 만일 모듈에 장착된 디바이스와 중복되는 채널을 사용하여 외부에 디바이스를 연결할 경우 해당 점퍼선을 제거하여야 한다. 모듈의 우측에는 I2C 통신을 위한 SDA, SCKL 단자와 전원용 단자가 구비되어 있다.

10.2.2 PCF8591 IC

PCF8591의 블록도는 다음과 같다. 상태 레지스터에는 전송된 제어 바이트가 위치하며, ADC 변환이 이루어질 때 상태 레지스터의 설정에 따라 제어 로직에 의해

선택된 아날로그 채널의 신호에 대한 AD 변환 결과는 ADC 데이터 레지스터에 위치한다. DA 변환할 때는 변환할 디지털 바이트는 3번째 바이트로 전송되어 DAC 데이터 레지스터에 위치하고, 이는 DAC를 통해 아날로그 신호로 변환되어 AOUT으로 출력된다.

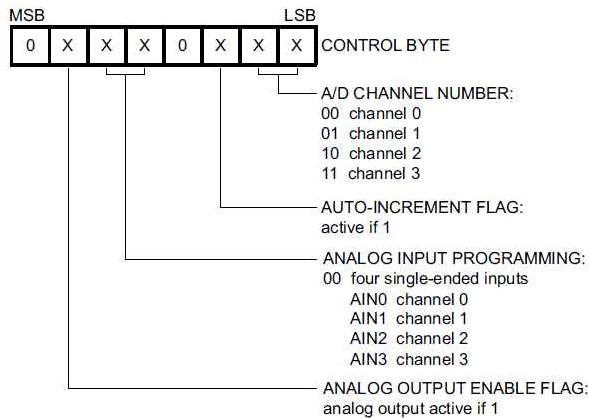


I2C 통신에서 유효한 주소를 각 PCF8591 장치에 전송함으로써 활성화된다. 어드레스는 고정 부분 및 프로그램 가능한 부분으로 구성된다. 프로그램 가능한 부분은 주소 핀 A2, A1 및 A0에 따라 설정해야 한다. 이 모듈에서 슬레이브 주소 설정을 위한 A2, A1, A0은 Gnd(Vss)에 연결되어 있다. 따라서 이 모듈의 슬레이브 주소는 000이다.

7	6	5	4	3	2	1	0
0	1	0	0	1	A2	A1	A0

이 슬레이브 주소 정보는 I2C 버스 프로토콜의 시작 조건 이후에 항상 첫 번째 바이트로 전송된다. 실제 전송에서는 슬레이브 주소가 1비트 좌쉬프트되고, LSB 비트에는 데이터 전송 방향을 설정하는 읽기/쓰기 비트가 위치한다.

PCF8591 장치에 전송된 두 번째 바이트는 제어 바이트로 상태 레지스터에 저장되며, PCF8591의 기능을 제어하는 데 사용된다. 제어 바이트의 각 비트에 대한 용도는 다음과 같으며, 보다 세부적인 것은 데이터 시트를 참조한다.



제어 바이트의 상위 니블은 아날로그 출력을 활성화하며, 아날로그 입력을 단일 엔드 혹은 차동 입력인지 설정하는데 사용된다. 하위 니블은 상위 니블에 의해 정의된 아날로그 입력 채널 중 하나를 선택하는데 사용된다. 두 니블의 MSB 비트는 미래의 사용을 위해 예약되었으며, 논리 0으로 설정해야 한다.

자동 증가 플래그가 설정되면 각 A/D 변환 후 채널 번호가 자동으로 증가한다. 내부 오실레이터가 사용되는 응용에서 자동 증가 모드가 필요한 경우는 제어 바이트의 비트 6의 아날로그 출력 인에이블 플래그를 1로 설정해야 한다.

전원이 투입된 초기 상태에 제어 레지스터의 모든 비트는 로직 0으로 설정된다. D/A 컨버터 및 발진기는 전력 절약을 위해 비활성화되고, 아날로그 출력은 하이 임피던스 상태에 있게 된다.

PCF8591 장치에 전송되는 세 번째 바이트는 DAC 데이터 레지스터에 저장되고, D/A 컨버터를 통해 아날로그 전압으로 변환된다.

끝으로 판독 사이클에서 전송되는 첫 번째 바이트 데이터는 이전 판독 사이클의 변환 결과 코드를 포함함을 유의한다. 즉, 슬레이브로부터 수신한 첫 바이트는 이전 변환 결과이므로 폐기하고, 그 다음 바이트부터가 유효한 변환 결과인 것이다.

10.2.2 I2C 주소 확인

회로 구성

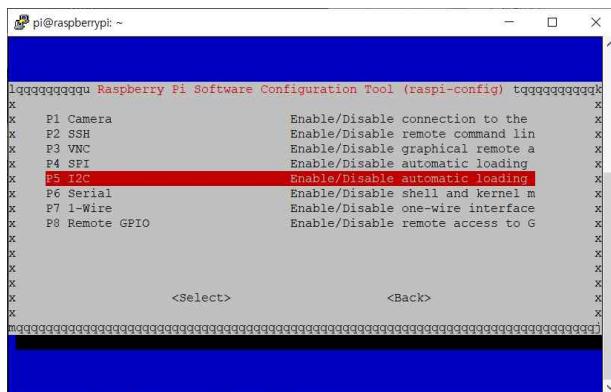
I2C 통신을 위해 라즈베리파이는 GPIO 핀으로 SDA1과 SCL1을 제공하고 있다. ADC 모듈의 SDA 단자는 SDA1(wiringPi #8, BCM_GPIO #2)에, SCL 단자는 SCL1(wiringPi #9, BCM_GPIO #3)에 연결한다. 그리고 전원 단자를 연결한다.

I2C 활성화

I2C 인터페이스를 통한 통신을 위해서는 라즈베리파이의 환경 설정에서 이 인터페이스를 활성화하여야 한다. 이를 위해 다음의 명령을 사용한다.

```
$ sudo raspi-config
```

실행 결과로 열린 창에서 Interfacing Options 항목을 선택하면 다음과 같은 화면이 나타나고 여기서 I2C 항목을 선택하여 활성화한다.



다음으로 다음 명령을 사용하여 재부팅한다.

```
$ sudo reboot
```

I2C 인터페이스의 활성화와 관련하여 부팅할 때 활성화 내역을 보려면 다음의 명령을 사용한다. `dtparam=i2c_arm=on` 라인이 있어야 한다.

```
$ cat /boot/config.txt
```

dtparam=i2c_arm=on

또한, 부팅할 때 I2C 통신을 위해 적재할 커널 모듈을 확인하려면 다음의 명령을 사용한다. 라즈베리파이 버전에 따라 i2c-dev 혹은 유사한 형태가 지정되어 있으면 된다.

```
$ cat /etc/modules  
i2c-dev
```

I2C 디바이스 주소 확인

모듈을 활용하여 회로를 구성하여 전원을 인가 한 후, 다음의 명령을 통해 I2C 통신 디바이스인 YL-40 모듈의 주소를 확인할 수 있다.

```
$ i2cdetect -y 1
```

```
pi@raspberrypi:~/IFC415/10_I2C/adc $ i2cdetect -y 1  
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  
00: --- --- --- --- --- --- --- --- --- --- ---  
10: --- --- --- --- --- --- --- --- --- --- ---  
20: --- --- --- --- --- --- --- --- --- --- ---  
30: --- --- --- --- --- --- --- --- --- --- ---  
40: --- --- --- --- --- 48 --- --- --- --- ---  
50: --- --- --- --- --- --- --- --- --- --- ---  
60: --- --- --- --- --- --- --- --- --- --- ---  
70: --- --- --- --- --- --- --- --- --- --- ---  
pi@raspberrypi:~/IFC415/10_I2C/adc $
```

이 명령의 결과로 나온 주소는 0x48이며, 이 주소의 하위 3비트의 000이 해당 모듈의 실 주소인 것이다.

이 주소는 wiringPiI2CSetup() 함수에서 다음과 같이 설정하여 사용한다.

```
if( (fd = wiringPiI2CSetup(0x48) ) < 0) {
```

제어 레지스터를 위한 제어 바이트는 싱글 엔드 방식의 아날로그 입력에 대해 처리할 것이므로, 특정 채널 선택을 위해 다음과 같이 기록 함수를 사용한다.

0 0 0 0 0 x x

```
wiringPiI2CWrite(fd, 0x00 | a2dChannel); // 채널선택
```

각 채널을 자동 증가 방식으로 선택하게 하려면 제어 바이트를 다음과 같이 설정하여 전송한다. 즉, 자동 증가 비트와 아날로그 출력 인에이블 비트를 1로 설정한다.

0 1 0 0 0 1 x x

```
wiringPiI2CWrite(fd, 0x44); // 채널 자동증가모드
```

10.2.3 실습과제

[실습1] 가변저항을 활용한 전압을 아날로그 신호원으로 디지털 변환

가변저항을 조정하여 그에 따른 전압(0~5V)을 아날로그 신호원으로 하여 디지털 변환하려면, 모듈에서 P6 단자를 점퍼선으로 연결하여야 한다. P6 단자를 연결하는 것은 AIN3 채널을 선택하는 것이다.

```
$ sudo nano adc_01.c
//=====
// adc_01.c
//      using YL-40 module(PCF8591 ADC/DAC)
//      AIN4 : voltage by VR, P6 Jumper
//=====
#include <stdio.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>

int main (void) {
    int fd;
    int i, cnt;
    int a2dChannel = 3; // analog channel AIN3, VR
    int prev, a2dVal;
    float a2dVol;
    float Vref = 5.0;

    printf("[ADC/DAC(YL-40) Module testing.....]\n");
```

```

if((fd = wiringPiI2CSetup(0x48))<0) {
    printf("wiringPiI2CSetup failed:\n");
}

cnt = 0;
while(1) {
    wiringPiI2CWrite(fd, 0x00 | a2dChannel);           // 0000_0011

    prev = wiringPiI2CRead(fd);           // Previously byte, garvage
    printf("[%d] previous = %d, ", cnt, prev);

    a2dVal = wiringPiI2CRead(fd);
    printf("2nd a2dVal = %d, ", a2dVal);

    a2dVol = 5.0 - (a2dVal * Vref / 255);
    printf("a2dVol = %f[V]\n", a2dVol);

    delay(1000);
    cnt++;
}

```

\$ make adc_01

\$./adc_01

```

pi@raspberrypi:~/IFC415/I0_I2C/adc $ ./adc_01
[ADC/DAC (YL-40) Module testing.....]
[0] previous = 96, 2nd a2dVal = 96, a2dVol = 3.117647[V]
[1] previous = 96, 2nd a2dVal = 96, a2dVol = 3.117647[V]
[2] previous = 96, 2nd a2dVal = 96, a2dVol = 3.117647[V]
[3] previous = 96, 2nd a2dVal = 96, a2dVol = 3.117647[V]
[4] previous = 96, 2nd a2dVal = 96, a2dVol = 3.117647[V]
[5] previous = 96, 2nd a2dVal = 96, a2dVol = 3.117647[V]
[6] previous = 96, 2nd a2dVal = 96, a2dVol = 3.117647[V]
[7] previous = 96, 2nd a2dVal = 201, a2dVol = 1.058824[V]
[8] previous = 201, 2nd a2dVal = 255, a2dVol = 0.000000[V]
[9] previous = 255, 2nd a2dVal = 255, a2dVol = 0.000000[V]
[10] previous = 255, 2nd a2dVal = 175, a2dVol = 1.568627[V]
[11] previous = 176, 2nd a2dVal = 91, a2dVol = 3.215696[V]
[12] previous = 91, 2nd a2dVal = 0, a2dVol = 5.000000[V]
[13] previous = 0, 2nd a2dVal = 0, a2dVol = 5.000000[V]
[14] previous = 0, 2nd a2dVal = 0, a2dVol = 5.000000[V]
[15] previous = 0, 2nd a2dVal = 0, a2dVol = 5.000000[V]
[16] previous = 0, 2nd a2dVal = 0, a2dVol = 5.000000[V]
[17] previous = 0, 2nd a2dVal = 0, a2dVol = 5.000000[V]
[18] previous = 0, 2nd a2dVal = 0, a2dVol = 5.000000[V]
[19] previous = 0, 2nd a2dVal = 0, a2dVol = 5.000000[V]
[20] previous = 0, 2nd a2dVal = 0, a2dVol = 5.000000[V]
^C

```

위의 결과는 가변저항을 변경하면서 얻은 결과이다. 아울러 읽어 들인 첫 번째 바이트(prev)는 이전 단계에서 읽은 변환결과와 동일함을 확인할 수 있다. 따라서 두

첫 번째 읽은 값은 버린다.

[실습2] 조도센서의 출력을 아날로그 신호원으로 디지털 변환

모듈에 장착된 조도 센서의 출력을 아날로그 신호원으로 하여 디지털 변환하려면, 모듈에서 P5 단자를 점퍼선으로 연결한다. P5 단자를 연결하는 것은 AIN0 채널을 선택하는 것이다.

```
$ sudo nano adc_02.c
//=====
// adc_02.c
//      using YL-40 module(PCF8591 ADC/DAC)
//      AIN0 : CDS sensor, P5 Jumper
//=====

#include <stdio.h>
#include <wiringPi.h>
#include <wiringPi2C.h>

int main (void) {
    int fd;
    int i, cnt;
    int a2dChannel = 0;      // analog channel AIN0, CDS sensor
    int prev, a2dVal;
    int threshold = 180;

    printf("[ADC/DAC(YL-40) Module testing.....]\n");

    if((fd = wiringPi2CSetup(0x48))<0) {
        printf("wiringPi2CSetup failed:\n");
    }

    cnt = 0;
    while(1) {
        wiringPi2CWrite(fd, 0x00 | a2dChannel);      // 0000_0000

        prev = wiringPi2CRead(fd);      // Previously byte, garvage
        a2dVal = wiringPi2CRead(fd);
        printf("[%d] prev = %d, ", cnt, prev);
        printf("a2dVal = %d, ", a2dVal);
    }
}
```

```

        if(a2dVal < threshold)
            printf("Bright!!\n");
        else
            printf("Dark!!\n");

        delay(1000);
        cnt++;
    }
}

```

```

$ make adc_02
$ ./adc_02

```

```

pi@raspberrypi:~/IFC415/I0_I2C/adc$ ./adc_02
[ADC/DAC (YL-40) Module testing.....]
[0] prev = 0, a2dVal = 152, Bright!!
[1] prev = 153, a2dVal = 155, Bright!!
[2] prev = 156, a2dVal = 156, Bright!!
[3] prev = 156, a2dVal = 156, Bright!!
[4] prev = 155, a2dVal = 155, Bright!!
[5] prev = 154, a2dVal = 163, Bright!!
[6] prev = 163, a2dVal = 195, Dark!!
[7] prev = 195, a2dVal = 188, Dark!!
[8] prev = 188, a2dVal = 187, Dark!!
[9] prev = 187, a2dVal = 188, Dark!!
[10] prev = 188, a2dVal = 187, Dark!!
[11] prev = 188, a2dVal = 188, Dark!!
[12] prev = 188, a2dVal = 160, Bright!!
[13] prev = 160, a2dVal = 156, Bright!!
[14] prev = 156, a2dVal = 156, Bright!!
[15] prev = 156, a2dVal = 155, Bright!!
[16] prev = 155, a2dVal = 155, Bright!!
[17] prev = 154, a2dVal = 153, Bright!!
[18] prev = 153, a2dVal = 152, Bright!!
[19] prev = 152, a2dVal = 152, Bright!!
^C
pi@raspberrypi:~/IFC415/I0_I2C/adc$ 

```

위의 결과는 CDS 센서의 출력 값에 대한 디지털 변환 결과를 보이며, 임계치를 180으로 하여 밝고 어둡기를 표시한 것이다.

[실습3] 온도센서의 출력을 아날로그 신호원으로 디지털 변환

모듈에 장착된 온도센서의 출력을 아날로그 신호원으로 하여 디지털 변환하려면, 모듈에서 P4 단자를 점퍼선으로 연결한다. P4 단자를 연결하는 것은 AIN1 채널을 선택하는 것이다.

```

$ sudo nano adc_03.c
//=====
// adc_03.c
//      using YL-40 module(PCF8591 ADC/DAC)

```

```

//      AIN1 : thermistor sensor, P4 Jumper
//=====
#include <stdio.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>

int main (void) {
    int fd;
    int i, cnt;
    int a2dChannel = 1; // analog channel AIN1, thermistor sensor
    int prev, a2dVal;
    int threshold = 100;

    printf("[ADC/DAC(YL-40) Module testing.....]\n");

    if((fd = wiringPiI2CSetup(0x48))<0) {
        printf("wiringPiI2CSetup failed:\n");
    }

    cnt = 0;
    while(1) {
        wiringPiI2CWrite(fd, 0x00 | a2dChannel); // 0000_0001

        prev = wiringPiI2CRead(fd); // Previously byte, garvage
        a2dVal = wiringPiI2CRead(fd);
        printf("[%d] prev = %d, ", cnt, prev);
        printf("a2dVal = %d, ", a2dVal);

        if(a2dVal > threshold)
            printf("Hot!!\n");
        else
            printf("Normal!!\n");

        delay(1000);
        cnt++;
    }
}

```

```

$ make adc_03
$ ./adc_03

```

[실습4] 디지털 변환(자동 증가 모드)

자동 증가 비트를 1로 설정하여 4개의 아날로그 신호를 디지털 신호로 변환하여 그 값을 출력하는 프로그램이다. 모듈에서 모든 점퍼선은 연결 상태로 한다. 소스의 다음 코드부분을 주목한다.

```
wiringPi2CWrite(fd, 0x44);      // 0100_01xx, Auto-Inc

$ sudo nano adc_04.c
=====
// adc_04.c
//      using YL-40 module(PCF8591 ADC/DAC)
//      Auto-Inc mode
=====
#include <stdio.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>

int main (void) {
    int fd;
    int i, cnt;
    int prev, ain[4];
    float a2dVol;
    float Vref = 5.0;

    printf("[ADC/DAC(YL-40) Module testing.....Auto Inc Mode]\n");

    if((fd = wiringPi2CSetup(0x48))<0) {
        printf("wiringPi2CSetup failed:\n");
    }

    cnt = 0;
    while(1) {
        wiringPi2CWrite(fd, 0x44);      // 0100_01xx, Auto-Inc

        prev = wiringPi2CRead(fd);      // Previously byte, garvage
        for(i=0; i<4; i++)           // for 4 channels
            ain[i] = wiringPi2CRead(fd);
```

```
    printf("[%d] ", cnt);
    for(i=0; i<4; i++)
        printf("ain%d = %d, ", i, ain[i]);
    printf("\n");

    delay(1000);
    cnt++;
}

}

$ make adc_04
$ ./adc_04
```

```
pi@raspberrypi:~/rFc415/10_i2cadc$ ./adc_04  
[ADC/DAC (MLY-40) Module testing.....Auto Inc Mode]  
[0] ain1 = 151, ainl = 255, ain2 = 109, ain3 = 0  
[1] ain1 = 153, ainl = 255, ain2 = 109, ain3 = 0  
[2] ain1 = 152, ainl = 255, ain2 = 108, ain3 = 0  
[3] ain0 = 152, ainl = 255, ain2 = 109, ain3 = 0  
[4] ain1 = 161, ainl = 255, ain2 = 109, ain3 = 0  
[5] ain0 = 167, ainl = 255, ain2 = 110, ain3 = 0  
[6] ain1 = 169, ainl = 255, ain2 = 110, ain3 = 0  
[7] ain1 = 172, ainl = 255, ain2 = 110, ain3 = 166,  
[8] ain1 = 171, ainl = 255, ain2 = 110, ain3 = 255,  
[9] ain1 = 171, ainl = 255, ain2 = 110, ain3 = 172,  
[10] ain0 = 170, ainl = 255, ain2 = 110, ain3 = 0,  
[11] ain0 = 156, ainl = 255, ain2 = 109, ain3 = 0,  
[12] ain0 = 153, ainl = 255, ain2 = 108, ain3 = 0,  
[13] ain0 = 152, ainl = 255, ain2 = 108, ain3 = 0,  
[14] ain0 = 151, ainl = 255, ain2 = 109, ain3 = 0,  
[15] ain0 = 149, ainl = 255, ain2 = 109, ain3 = 0,  
[16] ain0 = 149, ainl = 255, ain2 = 109, ain3 = 0,  
^C  
pi@raspberrypi:~/rFc415/10_i2cadc$  
pi@raspberrypi:~/rFc415/10_i2cadc$  
pi@raspberrypi:~/rFc415/10_i2cadc$  
pi@raspberrypi:~/rFc415/10_i2cadc$
```

4개 채널의 변환결과를 출력하되, 가변저항(ain3)만을 변경하면서 살펴본 결과이다.

10.3 OLED 제어

10.3.1 SSD1306 OLED 모듈



10.3.2 라이브러리 다운로드

OLED 모듈을 위한 라이브러리와 데모용 소스를 다운로드하기 위해 다음의 명령을 사용한다.

```
$ git clone https://github.com/iliapenev/ssd1306_i2c.git
```

다음 명령으로 현재 작업 디렉터리에 다운로드되어 생성된 ssd1306_i2c 디렉터리로 이동하여 파일들의 목록을 살펴본다.

```
$ cd ssd1306_i2c/  
$ ls  
README demo.c oled_fonts.h ssd1306_i2c.c ssd1306_i2c.h
```

ssd1306_i2c 이름의 파일들이 I2C 인터페이스를 통한 OLED 용 라이브러리 파일이며, oled_font 파일은 표시할 폰트 데이터들을 정의하고 있다. demo.c는 이 라이브러리를 활용한 데모 소스 파일이다.

ssd1306_i2c.h 헤더 파일을 살펴보면, 이 라이브러리는 다음의 함수들을 정의하고 있으며, 그 기능은 함수명으로 추정가능하다.

```
void ssd1306_begin(unsigned int switchvcc, unsigned int i2caddr);  
void ssd1306_command(unsigned int c);
```

```

void ssd1306_clearDisplay(void);
void ssd1306_invertDisplay(unsigned int i);
void ssd1306_display();

void ssd1306_startscrollright(unsigned int start, unsigned int stop);
void ssd1306_startscrollleft(unsigned int start, unsigned int stop);
void ssd1306_startscrolldiagright(unsigned int start, unsigned int stop);
void ssd1306_startscrolldiagleft(unsigned int start, unsigned int stop);
void ssd1306_stopscroll(void);

void ssd1306_dim(unsigned int dim);

void ssd1306_drawPixel(int x, int y, unsigned int color);

void ssd1306_drawFastVLine(int x, int y, int h, unsigned int color);
void ssd1306_drawFastHLine(int x, int y, int w, unsigned int color);

void ssd1306_fillRect(int x, int y, int w, int h, int fillcolor);

void ssd1306_setTextSize(int s);
void ssd1306_drawString(char *str);
void ssd1306_drawChar(int x, int y, unsigned char c, int color, int size);

```

10.3.3 실습예제

OLED 모듈의 동작을 살펴보려면, 우선 회로를 연결하고, I2C 인터페이스를 활성화한다. 그런 후, OLED 모듈의 주소를 확인하기 위해 다음의 명령을 사용한다.

```
$ i2cdetect -y 1
```

확인한 결과 OLED의 주소는 0x3C이다. 이 주소가 잘 설정되어 있는지 다음 명령을 사용하여 해당 부분을 확인한다.

```
$ nano ssd1306_i2c.h
.....
#define SSD1306_I2C_ADDRESS 0x3C           // 확인
.....
```

[실습5] OLED 모듈 데모

라이브러리에 포함된 데모 파일로 OLED에 텍스트 및 다각형을 출력하는 프로그램이다.

```
$ nano demo.c
#include "ssd1306_i2c.h"

void main() {

    ssd1306_begin(SSD1306_SWITCHCAPVCC,
                   SSD1306_I2C_ADDRESS);

    ssd1306_display();           // Adafruit logo is visible
    ssd1306_clearDisplay();
    delay(5000);

    char* text = "This is demo for SSD1306 i2c driver
                  for Raspberry Pi";
    ssd1306_drawString(text);
    ssd1306_display();
    delay(5000);

    ssd1306_dim(1);
    ssd1306_startscrollright(00, 0xFF);
    delay(5000);

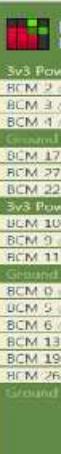
    ssd1306_clearDisplay();
    ssd1306_fillRect(10,10, 50, 20, WHITE);
    ssd1306_fillRect(80, 10, 130, 50, WHITE);
    ssd1306_display();
}
```

다음과 같이 컴파일하고 실행하여 그 결과를 살펴본다.

```
$ gcc -o demo demo.c ssd1306_i2c.c -lwiringPi  
$ ./demo
```

참고자료

- [1] I2C 문서
- [2] I2C 자료 https://cms3.koreatech.ac.kr/sites/joo/IFC224/IFC224_09.pdf
- [3] ADC/DAC PCF8591
<http://blog.naver.com/PostView.nhn?blogId=ubicomputing&logNo=220900474245>
- [4] PCF8591 문서, <https://www.nxp.com/docs/en/data-sheet/PCF8591.pdf>
- [5] ADC/DAC PCF8591T
<http://blog.naver.com/PostView.nhn?blogId=roboholic84&logNo=220497674350>
- [6] OLED 제어
<http://blog.naver.com/PostView.nhn?blogId=makepluscode&logNo=221375105015>



제11장 SPI 인터페이스

라즈베리파이는 범용적인 목적으로 입출력을 담당하는 GPIO 핀을 가지고 있으며, UART 통신, I2C 통신, SPI 통신을 가지고 있지만, ADC(Analog Digital Converter) 기능이 없어 아날로그 센서를 직접 활용하지 못한다. 그래서 아날로그 센서를 취급하려면 ADC 칩인 MCP3208을 이용하여 ADC 기능을 갖추도록 해야하는데, MCP3208 ADC는 SPI 인터페이스를 제공한다.

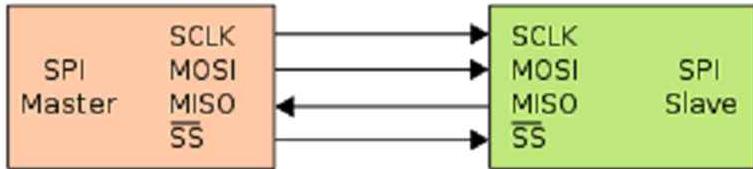
본 장에서는 이러한 상황에서의 통신방식인 SPI 인터페이스에 대해 살펴보고, 이를 이용해서 MCP3208 ADC와 통신하여 ADC에 연결된 아날로그 디바이스로부터 데이터를 얻는 방법에 대해 살펴본다.

11.1 SPI 인터페이스

SPI(serial peripheral interface)는 이종 컴퓨터간의 데이터 통신을 위해 사용되는 통신 방식으로, 다른 MCU, ADC, 센서와의 통신에 많이 사용된다. SPI는 Motorola가 1980년대에 프로토콜을 제안하면서 시작되었다. I2C와는 다르게 하나의 클럭 시간에 양방향으로 데이터를 송수신할 수 있는 전이중(Full-duplex) 통신 방식이다.

11.1.1 장치간 연결

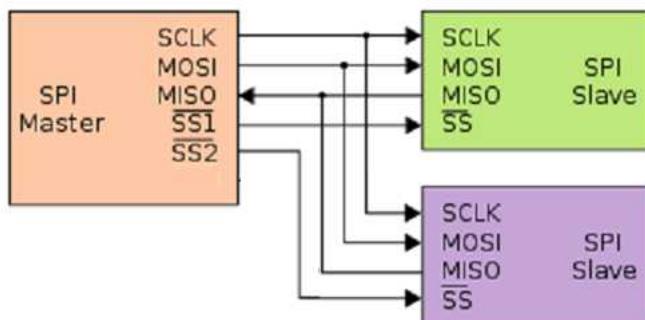
SPI에서도 I2C와 마찬가지로 마스터(master)와 슬레이브(slave)로 장치를 구분한다. 다음의 그림은 마스터와 슬레이브간 기본적인 연결 상태를 보여준다. 그림에서의 SS(slave select)는 라즈베리파이보드에서 CE0, CE1을 의미한다.



라즈베리파이 보드에서 SPI 인터페이스를 위한 신호선의 GPIO 핀 번호와 그 기능을 정리하면 다음 표와 같다.

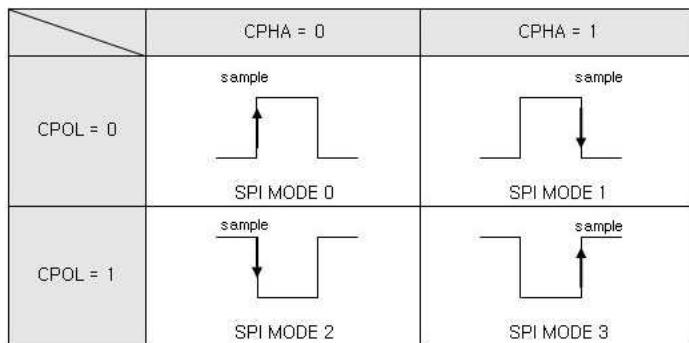
신호	wPi	BCM_GPIO	기 능
SCLK	#14	#11	Serial Clock(output from master)
MOSI	#12	#10	Master Out Slave In(data output from master)
MISO	#13	#09	Master In Slave Out(data output from slave)
CE0	#10	#08	Slave Select(often <u>active low</u> output from master)
CE1	#11	#07	Slave Select(often <u>active low</u> output from master)

라즈베리파이3 보드는 슬레이브 칩 선택단자가 CE0, CE1로 2개가 제공된다. 따라서 다음 그림과 같이 단순 직접 연결하는 경우 슬레이브 칩을 두 개까지 연결하여 두 개의 채널을 제공할 수 있다. 2x4 디코더를 사용하여 4개까지의 슬레이브를 연결할 수 있다.



11.1.2 샘플링 모드

SPI의 동작 모드는 다음 그림과 같이 SCLK 신호의 CPOL(극성)과 CPHA(위상)의 상태에 따라 샘플링 시점을 나타내는 4가지의 동작 모드가 있다.

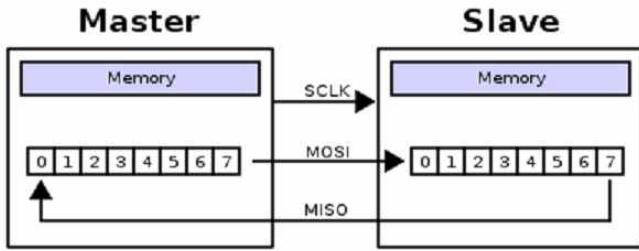


이를 정리하면 다음 표와 같다.

SPI Mode	Clock Polarity (CPOL)	Clock Phase (CPHA)
00	0	0
01	0	1
10	1	0
11	1	1

11.1.3 통신과정

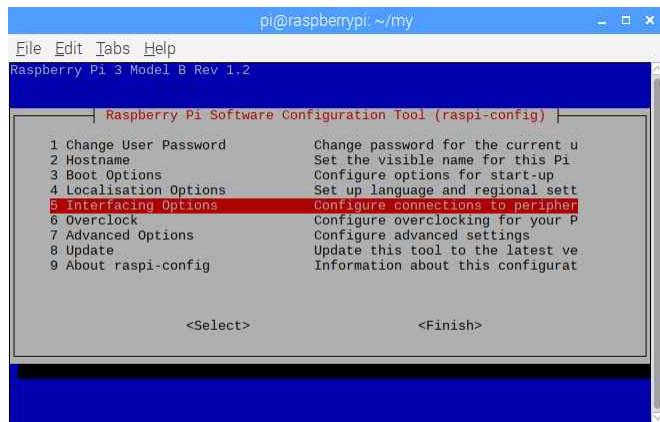
SPI 통신에서 데이터 기록 동작은 마스터가 SCLK에 클럭 신호를 출력하면서 MOSI 단자로 데이터를 출력한다. 마스터의 데이터 판독 동작은 읽기 명령 후 SCLK를 계속 보내는 동안 MISO 단자를 통해 데이터를 수신한다. 이는 마치 마스터와 슬레이브의 데이터 레지스터가 이웃해 있고 SCLK 클럭 신호에 동기되어 비트단위 로테이션하는 방식으로 데이터의 통신이 이루어지는 것이다.



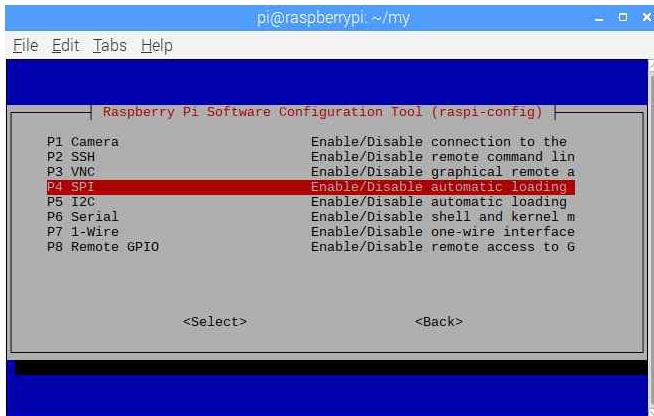
11.1.4 SPI 활성화

raspi-config 명령을 사용하면 다음과 같은 화면이 나타난다.

```
$ sudo raspi-config
```



Interfacing Options 항목을 선택하고, 나타난 다음과 같은 서브화면에서 SPI 항목을 선택하여 활성화한다.



그리고 다음의 명령을 사용하여 시스템을 재부팅한다.

```
$ sudo reboot
```

SPI 인터페이스의 활성화와 관련하여 부팅할 때 활성화 내역을 보려면 다음의 명령을 사용한다. `dtparam=spi=on` 라인이 있어야 한다.

```
$ cat /boot/config.txt
dtparam=spi=on
```

두 채널에 대한 디바이스 파일을 확인하려면 다음의 명령을 사용한다. `spidev0.0`는 채널 0에 대한, `spidev0.1`은 채널 1에 대한 디바이스 파일이다.

```
$ ls /dev/spi*
/dev/spidev0.0  /dev/spidev0.1
```

11.1.5 wiringPiSPI.h 라이브러리

wiringPi 라이브러리에 `wiringPiSPI.h`로 SPI 인터페이스 통신을 위한 함수들이 정의되어 있다.

하나의 SPI 통신 채널을 초기화하는 함수로 다음의 함수가 제공된다. 라즈베리파이는 채널 선택단자가 2개 제공된다. 따라서 channel은 채널번호로 0(CE0) 혹은 1(CE1)을 설정한다. speed는 500,000 ~ 32,000,000 사이의 정수로, SPI 클럭 속도를 Hz 단위로 표시하며, 반환 값은 파일 기술자이며, 초기화 오류시 -1을 반환한다. 내정된 모드 0로 연다.

```
int wiringPiSPISetup(int channel, int speed);
```

SPI 모드 설정을 함께할 수 있는 다음과 같은 함수가 제공된다. 위의 함수는 이 함수에서 mode 값을 0으로 하여, 내정된 모드로 초기화하는 함수이다.

```
int wiringPiSPISetupMode(int channel, int speed, int mode);
```

다음의 함수는 선택된 SPI 채널에 쓰기/읽기 동작을 동시에 수행하는 함수이다. data는 송수신 데이터를 저장할 여러 바이트를 의미하며, len은 그 길이를 바이트 단위로 지정한다.

```
int wiringPiSPIDataRW(int channel, unsigned char *data, int len);
```

기타의 함수로 다음과 같은 SPI 채널에 대한 파일기술자를 얻는 함수와,

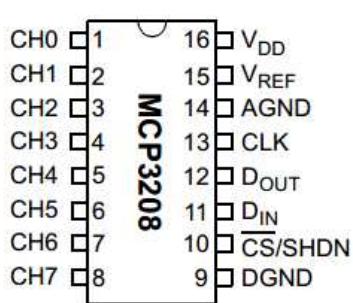
```
int wiringPiSPIGetFd(int channel);
```

11.2 MCP3208 ADC

MCP3208은 8-Channel 12-Bit A/D 변환기다. 반면 MCP3204는 4개의 채널을 가진다. 즉, 센서에서 출력되는 아날로그 신호를 12비트의 디지털 신호로 변환하는 ADC 칩이다. 예를 들어 0v ~ 3.3v 의 센서 출력값을 0 ~ 4095 (12bit) 의 디지털 값으로 변환해주는 기능을 해준다.

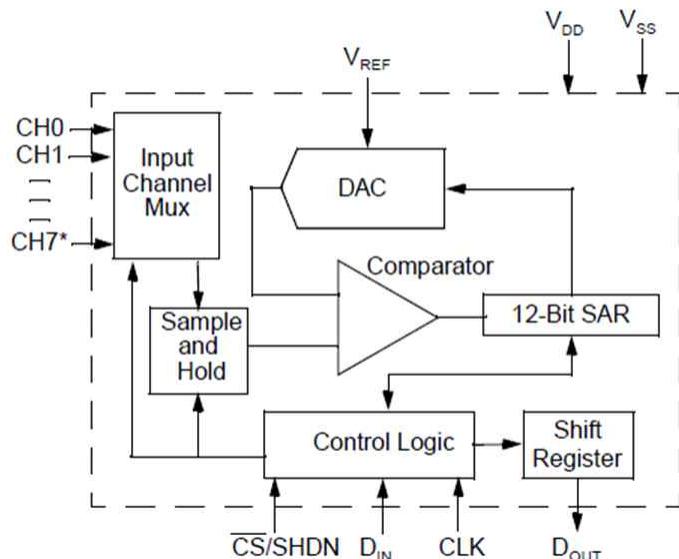
11.2.1 MCP3208의 핀 레이아웃 및 블록도

MCP3208 ADC의 핀 구조 및 그 기능은 다음과 같다.



Name	Function
V _{DD}	+2.7V to 5.5V Power Supply
DGND	Digital Ground
AGND	Analog Ground
CH0-CH7	Analog Inputs
CLK	Serial Clock
D _{IN}	Serial Data In
D _{OUT}	Serial Data Out
CS/SHDN	Chip Select/Shutdown Input
V _{REF}	Reference Voltage Input

MCP3208의 블록도는 다음과 같다. 총 8개의 ADC 채널(CH0, ..., CH7)을 가지고 있다.



11.2.2 MCP3208 전송 포맷

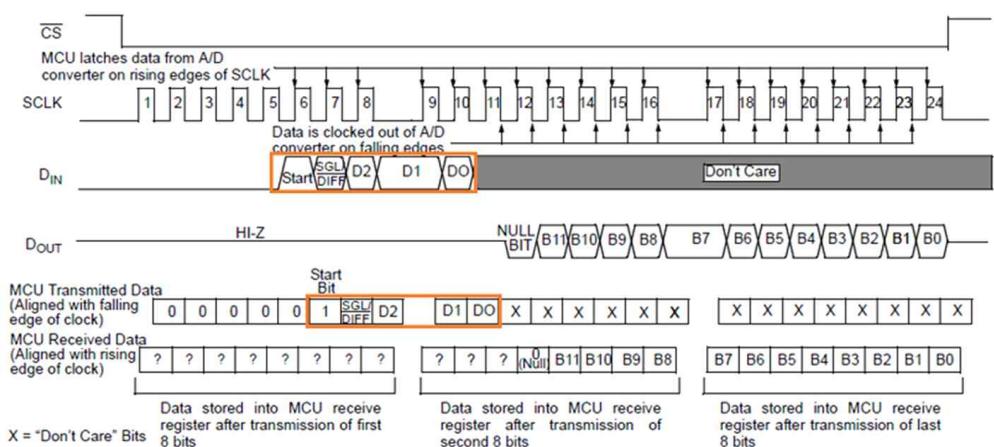
채널 설정

싱글 엔디드 입력 방식에서의 채널을 선택할 때의 제어 비트는 다음과 같다. SIN/DIF 비트는 싱글엔디드냐 차동 방식이냐를 위한 것으로 싱글엔디드 경우 1이며, D2, D1, D0 비트로 채널 번호를 설정한다. 차동 방식의 입력 채널의 경우는 데이터 시트를 참조한다.

CONTROL BIT SELECTIONS				INPUT CONFIGURATION	CHANNEL SELECTION
SINGLE/DIFF	D2	D1	D0		
1	0	0	0	single ended	CH0
1	0	0	1	single ended	CH1
1	0	1	0	single ended	CH2
1	0	1	1	single ended	CH3
1	1	0	0	single ended	CH4
1	1	0	1	single ended	CH5
1	1	1	0	single ended	CH6
1	1	1	1	single ended	CH7

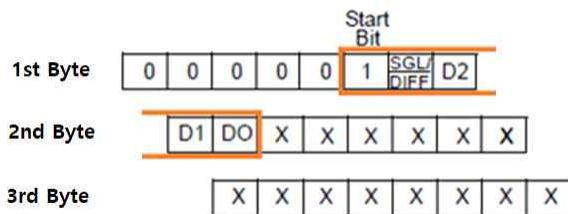
전송 포맷

전체적인 송수신과정의 타이밍 도는 다음과 같다. MCU가 데이터를 전송할 때는 SCLK의 상승에지에 동기화되고, MCU가 AD 변환된 결과 데이터를 수신할 때는 SCLK의 하강에지에 동기화된다.



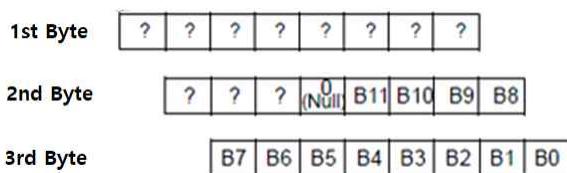
MCP3208의 전송 포맷은 3바이트로 구성된다. 전송 데이터의 포맷은 다음과 같이

첫 바이트와 두 번째 바이트의 일부 비트를 사용하여 채널 선택 제어비트를 설정한다. 두 번째 바이트의 일부와 3번째 바이트는 수신할 데이터를 위한 비트들로 구성되며, 이는 아직 다 수신되지 않은 ADC 데이터를 수신하기 위해서는 SCLK가 계속 출력되어야 함이다.



송수신의 바이트 단위로 일어난다. 즉, 전송 데이터의 첫 바이트가 전송되면서 데이터를 수신한다. 이때 수신하는 데이터는 의미없다. 두 번째 바이트가 전송되면서 두 번째 바이트를 수신하며, 이중 하위 니블의 값은 AD 변환 결과의 최상위 유효비트 정보를 수신하게 된다.

AD 변환 결과의 수신이 완료되면 데이터 포맷의 결과는 다음과 같다. 두 번째 바이트의 하위 니블에 변환결과의 최상위 유효비트들이, 세 번째 바이트에 하위 비트들의 데이터가 위치하게 된다.



MCP3208 동작속도

MCP3208 동작 속도는 인가되는 전압에 따라 다르다. 전원 전압은 2.7V ~ 5.5V를 사용할 수 있는데, 2.7V에서는 50ksps로 초당 50k의 샘플링이 이루어지는 반면 5V에서는 100ksps로 초당 100k의 샘플링이 이루어진다.

SPI 통신으로 데이터를 송수신할 때 하나의 데이터를 받아오는데 총 20개의 SCLK 을 사용하므로, 100ksps로 샘플링할 때 인가해 주어야 하는 SPI CLK 주파수는 2MHz이다.

11.2.3 MCP3208 제어 주요 코드

다음 코드는 통신을 위한 클럭 SCLK를 위한 주파수를 정의한다.

```
#define SPI_FREQ      1000000 // 1MHz int
```

MCP3208 IC와의 SPI 송수신 데이터 포맷은 다음과 같이 3바이트로 구성된다. 첫 번째 바이트는 시작비트, 싱글 엔디드 입력 선택비트, 3비트 채널 주소의 최상위 비트로 구성되며, 두 번째 바이트는 채널주소의 하위 2비트를 최상위 비트 쪽에 설정하며, 이하 비트와 3번째 바이트는 돈케어 조건이므로 0으로 설정한 코드이다.

```
uint8_t buf[3];  
  
buf[0] = 0x06 | ((aChAddr & 0x07) >> 2); // 0000 0 start single D2  
buf[1] = ((aChAddr & 0x07) << 6); // D1 D0 00_0000  
buf[2] = 0x00; // 0000_0000
```

데이터의 송수신을 위해 CS0단자에 Low 신호를 보낸 후, 위의 3바이트 데이터를 전송하면서 수신하는 함수를 호출한다. 그 다음 CS0 단자에 High 신호를 보내 통신을 종료한다.

```
digitalWrite(P_MCP_CS0, LOW); // chip select  
wiringPiSPIDataRW(P_MCP_DIN, buf, 3);  
digitalWrite(P_MCP_CS0, HIGH); // shutdown
```

데이터의 전송 및 수신 함수의 결과로 데이터 포맷의 두 번째 바이트의 하위 니블과 3번째 바이트인 12비트는 선택된 아날로그 채널의 AD 변환결과이며, 이로 다음과 같이 조합하여 값을 얻는다.

```
buf[1] = 0x0F & buf[1]; // masking lower nibble  
val = (buf[1] << 8) | buf[2]; // make 12 bits by concatenation
```

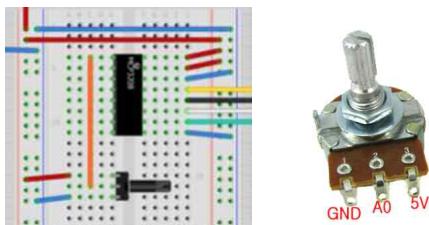
11.3 아날로그 입력 센서 제어

MCP2308 ADC를 사용하여 SPI 통신으로 가변저항, 화염센서, 조도센서, 온습도센서, 등의 아날로그 디바이스로부터 데이터를 얻는 과제를 살펴본다.

11.3.1 가변저항

[실습1] 가변저항을 활용한 신호의 ADC 변환

가변저항으로부터의 출력 신호를 AD 변환하여 그 결과를 살펴본다. 가변저항 회로는 아래 그림과 같이, MCP2308의 CH0에 가변저항의 출력단을 연결한다.



```
$ sudo nano spiMCP3208_01.c
//=====
// spiMCP3208_01.c
//      MCP3208 + sensors
//      CH0 : VR
//      CH1 : Flame sensor
//      CH2 : CDS sensor
//      CH3 : moisture sensor
//      CH4 : temperature sensor
//      CH5 : humidity sensor
//      CH6 : Infrared distance sensor
//=====

#include <stdio.h>
#include <stdint.h>
#include <wiringPi.h>
#include <wiringPiSPI.h>          // SPI interface

#define P_MCP_CLK      14      // BCM_GPIO #11, SCLK, 연결
#define P_MCP_DIN      12      // BCM_GPIO #10, MOSI, 연결
#define P_MCP_DOUT     13      // BCM_GPIO #09, MISO, 연결
```

```

#define P_MCP_CS0      10      // BCM_GPIO #08, CE0, 연결
#define SPI_CHAN_0      0       // slave channel select 0

#define P_MCP_CS1      11      // BCM_GPIO #07, CE1
#define SPI_CHAN_1      1       // slave channel select 1

#define SPI_SPEED      1000000 // 1MHz int

// analog sensor list
enum analogSensor {VR=0, FLAME, CDS, MOIST, TEMP, HUMI, IRD};

// function prototype
int readMCP3208(int fd, uint8_t analogCh);

int main(void) {
    enum analogSensor analogCh;
    int fd;
    int val;

    printf("[SPI MCP3208 + sensors testing...]\n");

    if(wiringPiSetup() == -1) {
        return 1;
    }

    if((fd = wiringPiSPISetup(SPI_CHAN_0, SPI_SPEED)) == -1) {
        return 1;
    }

    analogCh = VR;           // VR
    //analogCh = FLAME;      // flame sensor
    //analogCh = CDS;        // CDS sensor
    //analogCh = MOIST;      // moisture sensor
    //analogCh = TEMP;       // temperature sensor
    //analogCh = HUMI;       // humidity sensor
    //analogCh = IRD;        // infrared distance sensor

    while(1) {
        val = readMCP3208(fd, analogCh);

        printf("analogCh_%d Val = %d\n", analogCh, val);
        delay(1000);
    }
}

```

```

    }

    return 0;
}

int readMCP3208(int fd, uint8_t analogCh) {
    uint8_t buf[3];
    int val;

    buf[0] = 0x06 | ((analogCh & 0x07) >> 2);           // 0000 0 start single D2
    buf[1] = ((analogCh & 0x07) << 6);                  // D1 D0 00 0000
    buf[2] = 0x00;                                         // 0000 0000

    wiringPiSPIDataRW(fd, buf, 3);
    //wiringPiSPIDataRW(P_MCP_DIN, buf, 3);

    buf[1] = 0x0F & buf[1];                                // masking lower nibble
    val = (buf[1] << 8) | buf[2];                         // make 12 bits by concatnation

    return val;
}

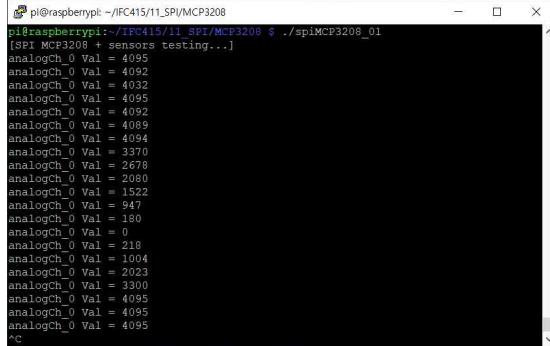
```

```

$ make spiMCP3208_01
$ ./spiMCP3208_01

```

다음의 결과는 가변저항을 점차 크게 하다가(시계반대방향) 다시 줄여가면서(시계방향) 얻어진 결과이다.

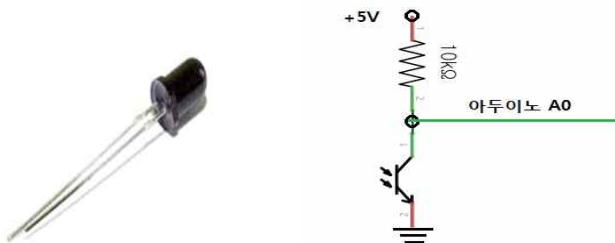


```
p@raspberrypi:~/FC415/11_SPI/MCP3208$ ./spiMCP3208_01
[SPI MCP3208 + sensors testing...]
analogCh_0 Val = 4095
analogCh_0 Val = 4095
analogCh_0 Val = 4033
analogCh_0 Val = 4095
analogCh_0 Val = 4095
analogCh_0 Val = 4089
analogCh_0 Val = 1004
analogCh_0 Val = 3370
analogCh_0 Val = 2678
analogCh_0 Val = 2080
analogCh_0 Val = 1522
analogCh_0 Val = 947
analogCh_0 Val = 180
analogCh_0 Val = 0
analogCh_0 Val = 218
analogCh_0 Val = 1004
analogCh_0 Val = 2023
analogCh_0 Val = 3300
analogCh_0 Val = 4095
analogCh_0 Val = 4095
analogCh_0 Val = 4095
^C
```

11.3.2 화염 센서

YG1006 화염 센서

광 트랜지스터는 특정 범위의 빛의 파장을 감지하는 특성을 이용하여 화염 감지 센서로 사용된다. 화염 감지 센서(flame sensor, or YG1006 phototransistor) 회로 구성은 그림과 같이 $10k\Omega$ 저항을 사용하여 광 트랜지스터에 연결한다. 사용된 광 트랜지스터는 다리가 긴 쪽을 GND에 연결한다. MCP2308의 CH1에 화염센서의 출력단을 연결한다.



[실습2] 화염센서 신호의 AD 변환

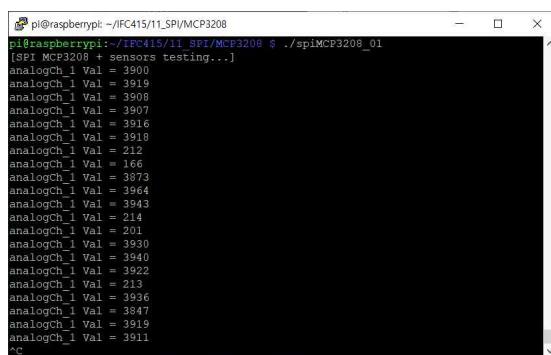
화염센서로부터 얻은 신호를 AD 변환하여 화염발생 여부에 따른 결과값을 확인한다.

```
$ sudo nano spiMCP3208_01.c
```

[실습1]의 소스에서 다음과 같이 수정하여 저장한다.

```
.....
//analogCh = VR;           // VR
analogCh = FLAME;         // flame sensor
//analogCh = CDS;          // CDS sensor
//analogCh = MOIST;         // moisture sensor
.....
$ make spiMCP320_01
$ ./spiMCP3208_01
```

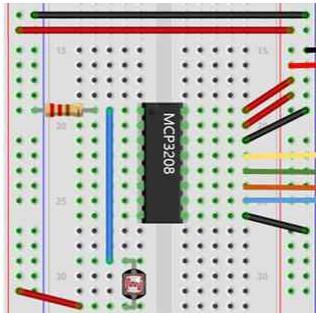
보통 상태에서 시작해서 3차례 정도 라이터를 켰을 때의 결과이며, 라이터를 켰을 때 그 값이 급격히 낮아지는 것으로 관찰된다.



```
pi@raspberrypi: ~/FC415/11_SPI/MCP3208 $ ./spiMCP3208_01
[SPi MCP3208 + sensors testing...]
analogCh_1 Val = 3900
analogCh_1 Val = 3919
analogCh_1 Val = 3909
analogCh_1 Val = 3907
analogCh_1 Val = 3916
analogCh_1 Val = 3910
analogCh_1 Val = 212
analogCh_1 Val = 166
analogCh_1 Val = 3873
analogCh_1 Val = 3964
analogCh_1 Val = 3943
analogCh_1 Val = 214
analogCh_1 Val = 201
analogCh_1 Val = 3930
analogCh_1 Val = 3940
analogCh_1 Val = 3922
analogCh_1 Val = 213
analogCh_1 Val = 3916
analogCh_1 Val = 3847
analogCh_1 Val = 3919
analogCh_1 Val = 3911
^C
```

11.3.3 조도센서

조도 센서회로를 다음 회로와 같이 구성하고, MCP2308의 CH2에 조도센서의 출력 단을 연결한다.



[실습3] 조도센서 신호의 AD 변환

조도센서로부터 얻은 신호를 AD 변환하여 밝기에 따른 결과값을 확인한다.

```
$ sudo nano spiMCP3208_01.c
```

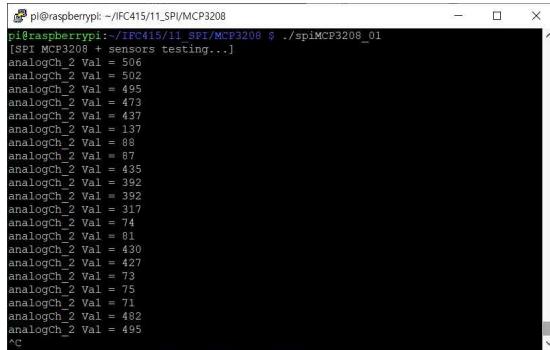
[실습1]의 소스에서 다음과 같이 수정하여 저장한다.

```
.....
//analogCh = VR;           // VR
analogCh = FLAME;         // flame sensor
//analogCh = CDS;          // CDS sensor
//analogCh = MOIST;        // moisture sensor
.....
```

```
$ make spiMCP320_01
```

```
$ ./spiMCP3208_01
```

다음의 결과는 실내에서 측정한 결과이며, 세차례 조도센서의 빛을 차단하여 얻은 결과이다. 차단하는 경우 급격히 값이 줄어드는 것으로 관찰된다.

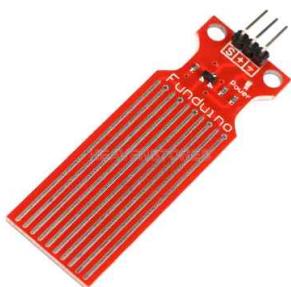


```
pi@raspberrypi:~/FC415/11_SPI/MCP3208$ ./spiMCP3208_01
[SPI MCP3208 + sensors testing...]
analogCh_2 Val = 506
analogCh_2 Val = 502
analogCh_2 Val = 495
analogCh_2 Val = 473
analogCh_2 Val = 437
analogCh_2 Val = 137
analogCh_2 Val = 86
analogCh_2 Val = 77
analogCh_2 Val = 435
analogCh_2 Val = 392
analogCh_2 Val = 392
analogCh_2 Val = 317
analogCh_2 Val = 74
analogCh_2 Val = 81
analogCh_2 Val = 430
analogCh_2 Val = 427
analogCh_2 Val = 73
analogCh_2 Val = 75
analogCh_2 Val = 71
analogCh_2 Val = 482
analogCh_2 Val = 495
^C
```

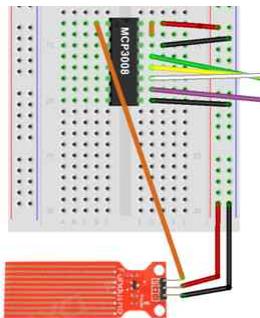
11.3.4 수분센서

수분 센서

수분 센서(moisture sensor)는 토양에서의 수분정도에 따라 건조여부를 판단하는데 이용될 수 있다. 일반적으로 AD 변환후의 결과로 건조한 토양의 경우 0~300, 습한 토양의 경우 300~700의 값을 얻을 수 있다.



수분센서 회로를 다음 그림과 같이 MCP2308의 CH3에 수분센서의 출력단을 연결 한다.



[실습4] 수분센서 제어

수분센서를 통하여 수분의 정도에 따른 AD 변환 결과를 측정하는 프로그램이다.

```
$ sudo nano spiMCP3208_01.c
```

[실습1]의 소스에서 다음과 같이 수정하여 저장한다.

```
.....
//analogCh = VR;           // VR
//analogCh = FLAME;        // flame sensor
//analogCh = CDS;          // CDS sensor
analogCh = MOIST;         // moisture sensor
.....
```

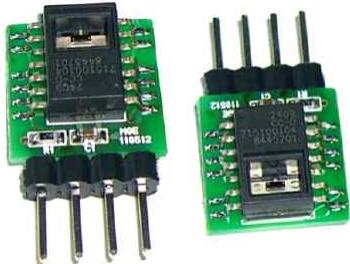
```
$ make spiMCP320_01
```

```
$ ./spiMCP3208_01
```

11.3.5 온도센서 및 습도센서 제어

온습도센서(CHIPCAP-L/Module)

온습도 센서 모듈의 외양은 다음 그림과 같다. 4개의 핀이 있으며, 1번 핀은 VCC, 2번 핀은 GND이며, 3번 핀은 습도 센서의 출력단이며, 4번 핀은 온도 센서의 출력단이다.



회로연결은 MCP2308의 CH4에 온도센서의 출력단인 4번 핀을, CH5에는 습도센서의 출력단인 3번 핀을 연결한다.

[실습5] 온도 및 습도 센서

온도 및 습도 센서의 신호를 AD 변환하여 출력하는 프로그램이다.

```
$ nano spiMCP3208_01.c
```

main() 함수의 다음 부분을 아래와 같이 수정한다.

```
//aChAddr = VR;           // VR
//aChAddr = FLAME;        // flame sensor
//aChAddr = CDS;          // CDS sensor
//aChAddr = MOIST;         // moisture sensor
aChAddr = TEMP;          // temperature sensor
//aChAddr = HUMI;          // humidity sensor
```

```
$ make spiMCP320_01
```

```
$ ./spiMCP3208_01
```

습도 센서의 결과 값은 소스를 다음과 같이 수정하여 실행하면, 관찰할 수 있다.

```
.....
//aChAddr = TEMP;          // temperature sensor
aChAddr = HUMI;            // humidity sensor
....
```

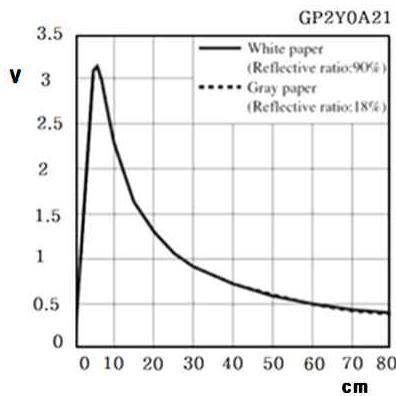
11.3.6 적외선거리 센서

SHARP 2Y0A21 적외선 거리 센서

SHARP 2Y0A21 적외선 거리 센서는 적외선을 쏘아서 반사되어온 값을 이용해 거리를 측정하는 적외선 거리측정 센서이다. 초음파 거리 센서보다는 보다 안정적인 거리측정이 가능하다는 장점이 있으나, 거리측정의 범위가 다소 근거리로 한정되는 단점을 가지기도 한다.



다음 그래프에서 보듯이 적외선 거리 센서의 측정거리는 약 10cm에서 최대 80cm 까지의 거리를 측정할 수 있으며, 10cm보다 적은 거리는 측정이 불가하다. 일반적으로 10cm~30cm의 근거리의 거리 측정에 적합하다.



적외선 거리 센서의 회로는 Vcc와 Gnd에 전원선을 연결하고, A0 출력단을 MCP3208의 CH6에 연결하여 구성한다.

[실습6] 적외선거리 센서

적외선 거리센서를 MCP3208의 CH6에 연결하여 대상체와의 거리에 따른 AD 변환결과를 출력하는 프로그램이다.

```
$ nano spiMCP3208_01.c
```

main() 함수의 다음 부분을 아래와 같이 수정한다.

```
//aChAddr = VR;           // VR
//aChAddr = FLAME;        // flame sensor
//aChAddr = CDS;          // CDS sensor
//aChAddr = MOIST;         // moisture sensor
//aChAddr = TEMP;          // temperature sensor
//aChAddr = HUMI;          // humidity sensor
aChAddr = IRD;           // infrared distance sensor
```

```
$ make spiMCP320_01
```

```
$ ./spiMCP3208_01
```

참고로 거리변환식을 활용하여 cm로 표현하려면 거리변환식을 파악해야 한다. 또한 거리변환식에서 math.h 라이브러리의 pow() 함수를 사용하므로, 컴파일할 때 math.h 라이브러리를 포함하도록 -lm 옵션을 추가하여 다음과 같이 컴파일해야 함을 유의한다.

```
$ gcc -o spiMCP3208_01 spiMCP3208_01.c -lwiringPi -lm
```

11.4 도트매트릭스 제어

앞의 제7장에서 wiringPiShift 라이브러리를 사용하여 MAX7219가 장착된 도트매트릭스 모듈에 패턴 데이터를 출력하는 것에 대해 살펴보았다. MAX7219의 내부 구조 및 제어 방법은 앞서의 내용을 참조한다.

11.4.1 SPI에 의한 도트매트릭스 제어

MAX7219 드라이버가 장착된 도트매트릭스 모듈의 제어는 적은 핀을 사용하면서 도트매트릭스를 동적으로 제어할 수 있다. MAX7219는 최대 10MHz까지 신호를 수신할 수 있으므로 SPI 인터페이스 방식으로도 제어가 가능하다.

MAX7291를 내장한 도트매트릭스 모듈의 회로 구성은 SPI 인터페이스용 핀중 SCLK, MOSI, CE0만 다음의 소스에서 보듯이 연결한다.

```
#define P_MAX_CLK      14      // BCM_GPIO #11, SCLK, 연결  
#define P_MAX_DIN      12      // BCM_GPIO #10, MOSI, 연결  
#define P_MAX_CS0       10      // BCM_GPIO #8, CE0, 연결
```

SPI 통신으로 도트매트릭스를 제어하기 위한 핵심 코드는 다음과 같다. MAX7291를 내장한 도트매트릭스를 위해서는 2바이트의 데이터 전송 포맷이 필요하며, 첫 바이트는 레지스터 주소, 두 번째 바이트는 패턴 데이터를 실어 전송한다.

```
void MAX7219_WRITE(uint8_t address, uint8_t data) {  
    uint8_t buf[2];  
  
    buf[0] = address;  
    buf[1] = data;  
  
    digitalWrite(P_MAX_CE0, LOW);  
    wiringPiSPIDataRW(P_MAX_MOSI, buf, 2);  
    digitalWrite(P_MAX_CE0, HIGH);  
}
```

MAX7219의 초기화 함수는 다음 코드와 같으며, 이와 관련된 레지스터들에 대한 내용은 앞서의 장에서 언급된 내용을 참조한다.

```
void MAX7219_INIT(void) {  
    MAX7219_WRITE(0x09, 0x00);  
        // Decode Mode : no decode all digits(0x00)  
    MAX7219_WRITE(0x0A, 0x0F);    // Intensity : max(0x0F)  
    MAX7219_WRITE(0x0B, 0x07);  
        // Scan Limit : Digit 0 ~ Digit 7, (0x07)  
    MAX7219_WRITE(0x0C, 0x01);    // Shutdown : Normal(0x01)  
}
```

11.4.2 실습예제

[실습7] 도트매트릭스 제어 I

SPI 통신을 이용하여 도트매트릭스에 문자패턴을 표시하는 프로그램이다.

```
$ nano spiDotMtx_01.c
//=====
// spiDotMtx_01.c
//      using SPI interface
//      MAX7219 driver
//=====

#include <stdio.h>
#include <stdint.h>
#include <wiringPi.h>
#include <wiringPiSPI.h>          // SPI interface

#define P_MAX_CLK      14      // BCM_GPIO #11, SCLK, 연결
#define P_MAX_DIN      12      // BCM_GPIO #10, MOSI, 연결
#define P_MAX_DOUT     13      // BCM_GPIO #09, MISO
#define P_MAX_CS0      10      // BCM_GPIO #08, CE0, 연결
#define SPI_CHAN_0      0

#define P_MAX_CS1      11      // BCM_GPIO #07, CE1

#define SPI_SPEED      1000000 // 1MHz int

const uint8_t image[] = {      // 5x8 font
    0x1E, 0x12, 0x1E, 0x00, 0x04, 0x07, 0x04, 0x04, //마
    0x09, 0x15, 0x09, 0x01, 0x01, 0x01, 0x01, 0x01, //이
    0x1F, 0x01, 0x07, 0x01, 0x00, 0x1F, 0x00, 0x00, //크
    0x1F, 0x01, 0x1F, 0x10, 0x1F, 0x00, 0x04, 0x1F, //로
    0x1F, 0x0A, 0x0A, 0x1F, 0x00, 0x1F, 0x00, 0x00, //프
    0x1F, 0x01, 0x1F, 0x10, 0x1F, 0x00, 0x04, 0x1F, //로
    0x04, 0x0A, 0x11, 0x00, 0x05, 0x1D, 0x05, 0x05, //세
    0x04, 0x0A, 0x11, 0x00, 0x01, 0x1F, 0x01, 0x01  //서
};

const uint8_t allOff[] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

```
};

// function prototype
void setup(void);
void MAX7219_INIT(void);
void MAX7219_WRITE(uint8_t address, uint8_t data);

int main(void) {
    int i, j;

    printf("[Dot Matrix testing....]\n");

    if(wiringPiSetup() == -1)
        return 1;

    if(wiringPiSPISetup(SPI_CHAN_0, SPI_SPEED) == -1) {
        return 1;
    }

    setup();

    for(j=0; j<8; j++) {
        for(i=0; i<8; i++) {
            MAX7219_WRITE(i+1, *(image+j*8+i));
        }
        delay(1000);
    }

    for(i=0; i<8; i++) {
        MAX7219_WRITE(i+1, *(allOff + i));
    }
    delay(1000);

    return 0;
}

void setup(void) {
    pinMode(P_MAX_CS0, OUTPUT);

    MAX7219_INIT();
}
```

```

void MAX7219_INIT(void) {
    MAX7219_WRITE(0x09, 0x00);
        // Decode Mode : no decode mode all digits(0x00)
    MAX7219_WRITE(0x0A, 0x0F);
        // Intensity(0x00~0x0F) : max(0x0F)
    MAX7219_WRITE(0x0B, 0x07);
        // Scan Limit : Digit 0 ~ Digit 7,(0x07)
    MAX7219_WRITE(0x0C, 0x01);      // Shutdown : Normal(0x01)
}

void MAX7219_WRITE(uint8_t address, uint8_t data) {
    uint8_t buf[2];

    buf[0] = address;           // MAX7219 register address
    buf[1] = data;              // pattern data

    digitalWrite(P_MAX_CS0, LOW);
    wiringPiSPIDataRW(P_MAX_DIN, buf, 2);
    digitalWrite(P_MAX_CS0, HIGH);
}

```

```

$ make spiDotMtx_01
$ ./spiDotMtx_01

```

[실습8] 도트매트릭스 제어 II

패턴 데이터를 좌우로 스크롤하는 방법은 8바이트의 패턴 데이터를 임시버퍼에 복사하고 바이트 단위로 로테이션한다. 다음 코드의 함수는 패턴을 좌로 스크롤하는 것이다.

```

void shift_left(uint8_t *array) {
    uint8_t temp;
    int i;

    temp = array[0];
    for(i=0; i<7; i++) {
        array[i] = array[i+1];
    }
}

```

```

    }
    array[7] = temp;
}

```

패턴을 상하로 스크롤하는 방법은 다음 코드와 같이 패턴데이터를 바이트단위로 이동을 하지 않고 비트단위로 쉬프트해야 한다. 즉 비트 7의 데이터를 임시버퍼에 저장하고 하위 7개 비트를 좌 쉬프트한 후, 임시버퍼에 저장된 비트를 비트0에 위치시킨다.

```

void shift_up(uint8_t *array) {
    uint8_t temp;
    int i;

    for(i=0; i<8; i++) {
        temp = array[i] & 0x80;
        array[i] <<= 1;
        array[i] |= (temp >> 7);
    }
}

```

다음 소스는 SPI 통신을 이용하여, 도트매트릭스에 상하좌우 스크롤링 효과를 주면서 패턴을 표시하는 프로그램이다.

```

$ nano spiDotMtx_02.c
//=====
// spiDotMtx_02.c
//      using SPI interface
//      MAX7219 driver, scrolling
//=====

#include <stdio.h>
#include <stdint.h>
#include <wiringPi.h>
#include <wiringPiSPI.h>      // SPI interface

#define P_MAX_CLK      14      // BCM_GPIO #11, SCLK, 연결
#define P_MAX_DIN      12      // BCM_GPIO #10, MOSI, 연결
#define P_MAX_DOUT     13      // BCM_GPIO #9, MISO
#define P_MAX_CS0      10      // BCM_GPIO #8, CE0, 연결
#define SPI_CHAN_0      0

```

```

#define P_MAX_CS1      11      // BCM_GPIO #7, CE1
#define SPI_SPEED      1000000 // 1MHz int

const uint8_t image[] = {      // 5x8 font
    0x1E, 0x12, 0x1E, 0x00, 0x04, 0x07, 0x04, 0x04, // 마
    0x09, 0x15, 0x09, 0x01, 0x01, 0x01, 0x01, 0x01, // 이
    0x1F, 0x01, 0x07, 0x01, 0x00, 0x1F, 0x00, 0x00, // 크
    0x1F, 0x01, 0x1F, 0x10, 0x1F, 0x00, 0x04, 0x1F, // 로
/*
    0x1F, 0x0A, 0x0A, 0x1F, 0x00, 0x1F, 0x00, 0x00, // 프
    0x1F, 0x01, 0x1F, 0x10, 0x1F, 0x00, 0x04, 0x1F, // 로
    0x04, 0x0A, 0x11, 0x00, 0x05, 0x1D, 0x05, 0x05, // 세
    0x04, 0x0A, 0x11, 0x00, 0x01, 0x1F, 0x01, 0x01 // 서
};
0x00,0x18,0x3C,0x7E,0x3C,0x3C,0x3C,0x00, // left
0x00,0x3C,0x3C,0x3C,0x7E,0x3C,0x18,0x00, // right
0x00,0x10,0x3E,0x7E,0x7E,0x3E,0x10,0x00, // up
0x00,0x08,0x7C,0x7E,0x7E,0x7C,0x08,0x00 // down
};

const uint8_t allOff[] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

uint8_t buf[8];

// function prototype
void setup(void);
void MAX7219_INIT(void);
void MAX7219_WRITE(uint8_t address, uint8_t data);
void shift_left(uint8_t *array);
void shift_right(uint8_t *array);
void shift_up(uint8_t *array);
void shift_down(uint8_t *array);

int main(void) {
    int i, j;
    int k = 17;

    printf("[Dot Matrix testing....]\n");
}

```

```
if(wiringPiSetup() == -1)
    return 1;

if(wiringPiSPISetup(SPI_CHAN_0, SPI_SPEED) == -1) {
    return 1;
}

setup();

for(j=0; j<8; j++) {
    for(i=0; i<8; i++) {
        MAX7219_WRITE(i+1, *(image+j*8+i));
    }
    delay(1000);
}

for(i=0; i<8; i++) {
    MAX7219_WRITE(i+1, *(allOff + i));
}
delay(1000);

// scrolling
for(i=0; i<8; i++)           // memcpy()
    buf[i] = image[i];

while(k--) {
    for(i=0; i<8; i++) {
        MAX7219_WRITE(i+1, *(buf+i));
    }
    delay(100);
    shift_left(buf);          //
}
delay(1000);

return 0;
}

void setup(void) {
    pinMode(P_MAX_CS0, OUTPUT);

    MAX7219_INIT();
}
```

```

void MAX7219_INIT(void) {
    MAX7219_WRITE(0x09, 0x00);
        // Decode Mode : no decode mode all digits(0x00)
    MAX7219_WRITE(0x0A, 0x0F);
        // Intensity(0x00~0x0F) : max(0x0F)
    MAX7219_WRITE(0x0B, 0x07);
        // Scan Limit : Digit 0 ~ Digit 7,(0x07)
    MAX7219_WRITE(0x0C, 0x01);      // Shutdown : Normal(0x01)
}

void MAX7219_WRITE(uint8_t address, uint8_t data) {
    uint8_t buf[2];

    buf[0] = address;           // MAX7219 register address
    buf[1] = data;              // pattern data

    digitalWrite(P_MAX_CS0, LOW);
    wiringPiSPIDataRW(P_MAX_DIN, buf, 2);
    digitalWrite(P_MAX_CS0, HIGH);
}

void shift_left(uint8_t *array) {
    uint8_t temp;
    int i;

    temp = array[0];
    for(i=0; i<7; i++) {
        array[i] = array[i+1];
    }
    array[7] = temp;
}

void shift_right(uint8_t *array) {
    uint8_t temp;
    int i;

    temp = array[7];
    for(i=7; i>0; i--) {
        array[i] = array[i-1];
    }
    array[0] = temp;
}

```

304 라즈베리파이 기본 임베디드 시스템 응용

```
}

void shift_up(uint8_t *array) {
    uint8_t temp;
    int i;

    for(i=0; i<8; i++) {
        temp = array[i] & 0x80;
        array[i] <= 1;
        array[i] |= (temp >> 7);
    }
}

void shift_down(uint8_t *array) {
    uint8_t temp;
    int i;

    for(i=0; i<8; i++) {
        temp = array[i] & 0x01;
        array[i] >= 1;
        array[i] |= (temp << 7);
    }
}

$ make spiDotMtx_02
$ ./spiDotMtx_02
```

참고자료

[1] SPI 개요

<http://blog.naver.com/PostView.nhn?blogId=phj790122&logNo=221065026667&parentCategoryNo=&categoryNo=&viewDate=&isShowPopularPosts=false&from=postView>

[2] MCP3208 ADC

<http://blog.naver.com/PostView.nhn?blogId=phj790122&logNo=221064374287&parentCategoryNo=&categoryNo=&viewDate=&isShowPopularPosts=false&from=postView>

[3] SPI 테스트 <http://icbanq.tistory.com/469>

[4] 온습도 센서, <http://icbanq.tistory.com/1232>

[5] 조도 센서, <http://fishpoint.tistory.com/2050>

[6] 화염 센서

<http://blog.naver.com/PostView.nhn?blogId=m.yang59&logNo=220022308279>

[7] 수분센서

<http://blog.naver.com/PostView.nhn?blogId=roboholic84&logNo=220416082742>

[8] 센서보드 <http://icbanq.tistory.com/876>

[9] 적외선거리센서

<http://blog.naver.com/PostView.nhn?blogId=boilmint7&logNo=220927816896>

[10] 적외선거리센서 <http://blog.daum.net/ejleep1/458>

[11] SPI에 의한 도트매트릭스 제어

<http://blog.naver.com/PostView.nhn?blogId=kiatwins&logNo=220792432677&parentCategoryNo=&categoryNo=&viewDate=&isShowPopularPosts=false&from=postView>

[12] SPI에 의한 도트매트릭스 제어(스크롤링)

<http://blog.naver.com/PostView.nhn?blogId=kiatwins&logNo=221014087180&parentCategoryNo=&categoryNo=&viewDate=&isShowPopularPosts=false&from=postView>

제12장 U-Boot 부트로더 및 GPIO

본 장에서는 라즈베리파이에서 부트로더의 동작방식을 살펴보고, U-boot 부트로더를 포팅한다. 또한 GPIO 핀을 직접적으로 제어하는 것에 관해 살펴본다.

12.1 U-Boot 부트로더

12.1.1 부트로더의 이해

부트로더는 시스템을 초기화하고 운영체제를 적재하거나 실행하기 위해 시스템 초기화코드, 하드웨어 제어 프로그램, 네트워크, USB 등의 프로토콜과 일부 파일 시스템을 관리한다.

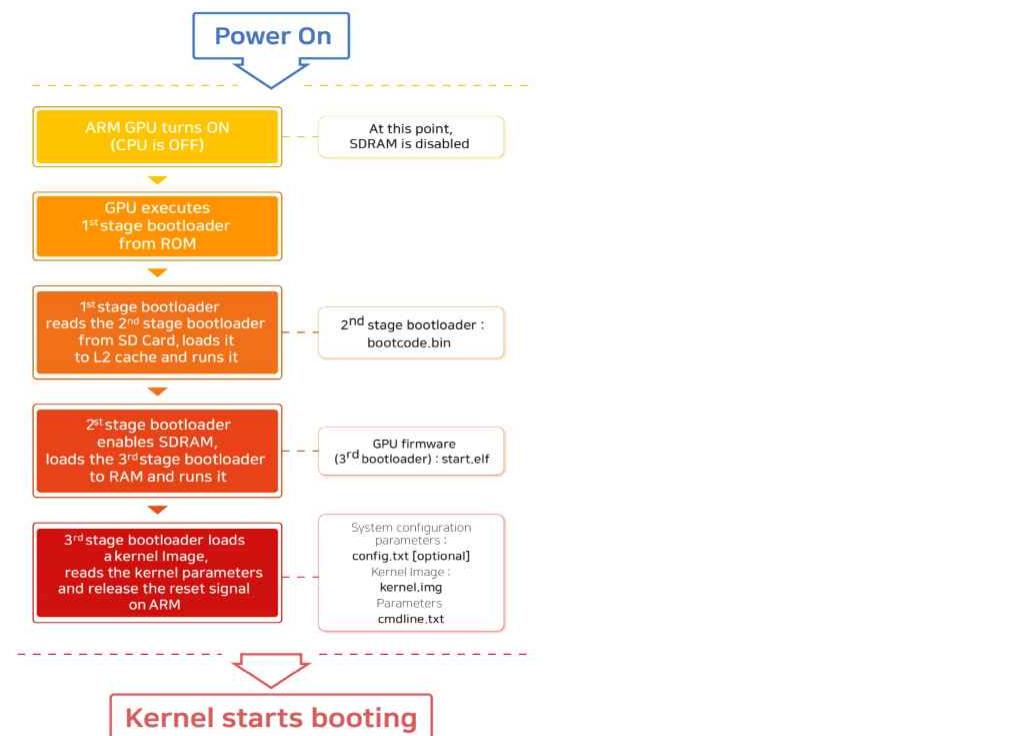
부트로더의 기능을 살펴보면 다음과 같다. 전원이 입력되면 시스템 클록 설정, 메모리 관리 설정 등 하드웨어 설정 및 소프트웨어 관련 환경 설정을 통해 시스템을 초기화하는 시스템 초기화 기능을 한다. 또한 일반 시스템에서 BIOS와 CMOS 설정과 유사하게 부팅 방법, 네트워크 관련한 IP 주소 설정 등의 시스템 동작 환경 기능을 가진다. 그리고 임베디드 시스템에서 초기에 플래시 메모리에 저장되어 있는 커널 이미지를 DRAM에 적재하고 실행함으로서 커널 부팅의 기능을 가진다. 기타 커널 이미지와 부트로더 이미지가 기록되어 있는 플래시 메모리를 관리하거나, 시스템의 동작 상태 감시, 하드웨어 정상 동작 여부 검사, 메모리 검사와 POST(Power-On Self Test) 등의 모니터링 기능을 수행한다.

부트로더의 특징으로는 하드웨어 의존성이 강하며, 일반적으로 부트 섹터에 위치하게 되므로 크기가 작아야한다. 또한 부트로더를 작성하려면 부트로더의 시작 코드는 해당 프로세서의 어셈블리어로 작성되기 때문에 그 어셈블리어를 알고 있어야 하며, 프로세서의 구조와 특징들에 대해서도 파악하고 있어야 한다.

현재 가용한 부트로더 종류로는 LILO, GRUB, Loadlin, EtherBoot, Blob, PMON, RedBoot, U-Boot 등이 있다.

ARM CPU의 부트로더

라즈베리파이 보드에 있는 ARM CPU의 부트로더의 동작 절차를 살펴보면 다음 그림과 같다.



전원이 투입되면 ARM GPU가 활성화되고 ROM으로부터 1st 부트로더를 실행한다. 이 때 CPU는 비활성상태이며 SDRAM은 사용할 수 있는 상태가 아니다. 1st 부트로더는 SD 카드로부터 bootcode.bin 파일인 2nd 부트로더를 L2 캐쉬메모리에 적재하고 실행한다. 2nd 부트로더는 fixedup.dat 파일을 참조하여 SDRAM을 파티션 설정 후 SDRAM을 활성화하고 start.elf 파일인 3rd 부트로더를 RAM으로 적재하고 실행한다. 다음으로 3rd 부트로더는 커널 이미지를 적재하고 커널 패러메터를 읽고 ARM에 reset 신호를 인가함으로써 커널 부팅을 시작한다. 이때 관련된 파일로 kernel.img, config.txt, cmdline.txt, bcm2710-rpi-3-b-plus.dtb와 overlays 디렉터리가 있다. 3rd 부트로더는 커널 이미지 파일 대신 이진파일 형태의 U-Boot 와 같은 또 다른 부트로더 혹은 응용 프로그램을 적재하여 실행할 수 있다.

부팅과 관련된 파일들은 /boot 디렉터리에 위치하며 이들 파일중 주요한 파일들을 살펴보면 다음과 같다.

```
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
-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
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
```

12.1.2 U-Boot 부트로더

커널이미지 대신 U-Boot 부트로더로 부팅하려면, 부트로더의 이진파일을 /boot 디렉터리에 위치시켜야 하고, config.txt 파일을 일부 수정해야한다.

U-boot 부트로더를 컴파일하고, 부팅하여 사용하는 과정에 대해서 아래에서 살펴보도록 한다. 작업은 가상머신에서 진행한다. 가상머신에 로그인한 후 다음 명령을 사용하여 슈퍼유저로 전환한다.

```
ifc413@ubuntu:~$ sudo su
root@ubuntu:/home/ifc413# cd
root@ubuntu:~#
```

U-Boot 소스 다운로드

다운로드하는데 사용할 툴인 git 패키지를 다음의 절차에 따라 설치한다.

```
root@ubuntu:~# apt-get update
root@ubuntu:~# apt-get upgrade
```

3.10 라즈베리파이기반 임베디드시스템용

```
root@ubuntu:~# apt-get install git // git 패키지 설치
```

이제 다음의 명령을 사용하여 B-Boot 부트로더 소스를 다운로드한다.

```
root@ubuntu:~# git clone git://git.denx.de/u-boot.git
```

현 작업 디렉터리 내에 ./u-boot라는 디렉터리 명으로 부트로더 소스가 다운로드 된다. 다음의 명령을 통해 다운로드되어 생성된 디렉터리를 확인할 수 있다.

```
root@ubuntu:~# ls  
u-boot
```

혹은, 이미 다운로드 받은 압축 파일(/u-boot/u-boot.tar.gz, 2018.9 버전)이 있는 경우, 다음의 명령들을 사용하여 적절한 위치로 복사하여 풀어 놓는다.

```
root@ubuntu:~# ls /mnt/hgfs/Shared/  
root@ubuntu:~# cp /mnt/hgfs/Shared/Achro-EM/u-boot.tar.gz ./  
root@ubuntu:~# tar xvf u-boot.tar.gz  
root@ubuntu:~# ls  
u-boot
```

U-Boot 소스 구성

다음의 명령을 사용하여 U-Boot 부트로더의 홈 디렉터리로 이동하여 그 내용들을 살펴본다.

```
root@ubuntu:~# cd u-boot  
root@ubuntu:~/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#
```

U-Boot의 홈 디렉터리에는 컴파일을 위한 Makefile이 있으며, 여러 하위 디렉터리를 가지고 있다.

arch : 프로세서 아키텍처별로 각 아키텍처에 따른 소스를 구성하는 디렉터리
 board : 각 제조사별 보드 관련 소스를 구현한 디렉터리
 common : U-Boot에서 공통적으로 사용되는 소스를 구현한 디렉터리
 disk : disk를 관리와 관련한 소스를 구현한 디렉터리
 doc : U-Boot 관련 문서를 가지고 있는 디렉터리
 drivers: U-Boot에서 지원되는 다양한 장치들에 대한 드라이버 소스를 구현한 디렉터리
 examples : U-Boot에서 독립적으로 프로그램을 실행시키는 샘플코드를 포함한 디렉토리로 api와 standalon용 예제
 fs : U-Boot에서 지원하는 다양한 파일시스템에 대한 소스를 구현한 디렉터리
 include : U-Boot에서 사용되는 헤더 파일을 소스를 구현한 디렉터리
 configs : 보드별 동작에 필요한 환경 설정 파일이 위치한 디렉터리
 lib : U-Boot에서 사용되는 각종 라이브러리를 구현한 디렉터리
 net : U-Boot에서 지원되는 UDP, IP, TFTP 등 네트워크 프로토콜 관련한 소스를 구현한 디렉터리
 post : POST와 관련한 소스를 구현한 디렉터리
 tools: U-Boot 사용에 필요한 각종 호스트 유ти리티를 구현한 디렉터리

U-Boot 부트로더를 특정 보드에 맞게 컴파일할 경우 주목해야 할 디렉터리는 특정 보드에 맞게 정의된 환경설정파일이 있는 configs와, 특정 아키텍춰를 위한 arch이며, 응용 테스트를 위한 examples 디렉터리이다.

U-Boot 부트로더 컴파일

타깃 보드에 맞는 U-Boot 환경설정파일들이 ./configs 디렉터리에 제공되며 다음의 명령을 사용하여 확인할 수 있다.

```
root@ubuntu:~/u-boot# ls ./configs/rpi*
```

3.12 라즈베리파이 기본 임베디드 시스템 활용

```
pi@raspberrypi:~/IFC415/12_GPIO/UBoot/u-boot/configs $ ls rp* -l
-rw-r--r-- 1 pi pi 1004 Oct 22 2018 rpi_0_w_defconfig
-rw-r--r-- 1 pi pi 999 Oct 22 2018 rpi_2_defconfig
-rw-r--r-- 1 pi pi 1077 Oct 22 2018 rpi_3_32b_defconfig
-rw-r--r-- 1 pi pi 1073 Oct 22 2018 rpi_3_defconfig
-rw-r--r-- 1 pi pi 995 Oct 22 2018 rpi_defconfig
pi@raspberrypi:~/IFC415/12_GPIO/UBoot/u-boot/configs $
```

rpi_3_32b_defconfig 파일이 라즈베리파이 3B+ 보드용의 부트로더 환경설정 파일이다. 따라서 이 파일을 활용하여 U-Boot 환경설정하기 위해 다음의 명령을 사용한다.

```
root@ubuntu:~/u-boot# make rpi_3_32b_defconfig
```

실행하면 오류가 발생되는데 이는 두 개의 패키지가 설치되지 않아 발생하는 것으로, 다음의 명령을 사용하여 요구되는 bison, flex 패키지를 설치한다.

```
root@ubuntu:~/u-boot# apt install bison
root@ubuntu:~/u-boot# apt install flex
```

다시 환경 설정을 시도하면 오류없이 환경설정이 완료된다.

```
root@ubuntu:~/u-boot# make rpi_3_32b_defconfig // 오류없음
```

현 작업 디렉터리에 환경설정내용이 .config 히든 파일로 저장되고, 컴파일시 이 히든 파일에 환경설정된 내용을 토대로 컴파일이 진행된다.

이제 부트로더 소스를 컴파일하기 위해 다음의 명령을 사용한다. 옵션 -j4는 프로세서 코어의 개수를 의미한다. 현 작업 디렉터리의 .config 히든 파일의 내용을 토대로 소스를 컴파일하여 부트로더의 이진파일 u-boot.bin을 생성한다.

```
root@ubuntu:~/u-boot# make -j4
```

다음의 명령으로 컴파일 결과 생성된 파일들을 살펴볼 수 있다.

```
root@ubuntu:~/u-boot# ls
```

```
root@ubuntu:~/u-boot# ls u-boot* -al
-rwxr-xr-x 1 root root 3511472 Dec 17 22:43 u-boot
-rw-r--r-- 1 root root 431244 Dec 17 22:43 u-boot.bin
-rw-r--r-- 1 root root 8743 Dec 17 22:43 u-boot.cfg
-rw-r--r-- 1 root root 5298 Dec 17 22:43 u-boot.cfg.configs
-rw-r--r-- 1 root root 1719 Dec 17 22:43 u-boot.lds
-rw-r--r-- 1 root root 522157 Dec 17 22:43 u-boot.map
-rw-r--r-- 1 root root 431244 Dec 17 22:43 u-boot-nodtb.bin
-rw-r--r-- 1 root root 1239948 Dec 17 22:43 u-boot.srec
-rw-r--r-- 1 root root 137552 Dec 17 22:43 u-boot.sym
root@ubuntu:~/u-boot#
```

u-boot.bin 등등의 생성 파일 관찰할 수 있으며, 여기서 U-Boot 부트로더의 이진 파일인 u-boot.bin 파일을 타깃보드의 BOOT 파티션인 /boot 디렉터리에 복사하고, 커널이미지 대신 사용될 수 있도록 조치하면 부트로더로 부팅할 수 있게 된다.

U-Boot 부트로더 포팅

이제 부트로더 이진파일을 라즈베리파이보드의 /boot 디렉터리로 위치시키고, 부팅할 때 커널이미지 대신 부트로더 이진파일을 실행할 수 있도록 해야 한다.

부트로더 이진파일을 라즈베리파이 보드의 /boot 디렉터리로 위치하기 위해 NFS 서비스를 통하여 공유할 수 있다. 물론 가상머신에서 SD 카드를 마운트하여 직접 기록하는 것도 가능하다.

NFS 서비스를 통해 부트로더 이진파일을 복사하는 과정은 다음과 같다. 우선 가상 머신의 서버 측에서는 공유 디렉터리에 해당 파일을 위치시킨다.

```
root@ubuntu:~/u-boot# cp u-boot.bin /nfs
root@ubuntu:~/u-boot# ls /nfs
h_hello t_hello u-boot.bin
root@ubuntu:~/u-boot#
```

NFS 서비스의 클라이언트 측, 즉 target 보드로 puTTY를 이용하여 접속한 후, 슈퍼유저 계정으로 전환한다.

```
pi@raspberrypi:~ $ sudo su
root@raspberrypi:/home/pi# cd
root@raspberrypi:~# ls
```

3.14 라즈베리파이 기본 임베디드 시스템 활용

해당 이진 파일을 다음 과정을 통하여 /boot 파티션으로 복사한다.

```
root@raspberrypi:~# mount -t nfs 192.168.0.20:/nfs /share
root@raspberrypi:~# ls /share
h_hello  t_hello  u-boot.bin
root@raspberrypi:~# cd /boot
root@raspberrypi:/boot# cp /share/u-boot.bin /boot
root@raspberrypi:/boot# ls *.bin
bootcode.bin  u-boot.bin
```

config.txt 파일 편집

부트로더 관련 파일들이 확보되면 부팅할 때 커널이미지 대신 U-Boot 부트로더를 실행하도록 다음의 명령을 사용하여 /boot/config.txt 파일을 편집하여 저장하고, 새 부팅한다.

```
root@raspberrypi:/boot# nano config.txt
.....
# B-boot loader
kernel=u-boot.bin
```

```
root@raspberrypi:/boot# reboot
```

U-boot 부트로더로 부팅하는 경우, 가능하면 모니터와 키보드를 연결하여 SBC 환경을 갖출 것을 권고한다. 부팅이 시작되자마자 임의의 키를 눌러 부트로더의 명령모드로 진입할 수 있도록 한다. 부트로더의 명령모드로 진입하게 되면 다음과 같은 부트로더 프롬프트가 나타난다.

U-Boot>

```
Net: No ethernet found.
starting USB...
USB0: scanning bus 0 for devices... 5 USB Device(s) found
      scanning usb for storage devices... 0 Storage Device(s) found
Hit any key to stop autoboot: 0
U-Boot>■
```

이제 부트로더가 제공하는 내부 명령들을 사용할 수 있는 상태이다.

12.1.3 U-Boot 부트로더 내부명령

부트로더의 내부명령에 대한 도움을 얻으려면 다음과 같이 help 명령을 사용한다.

```
U-Boot> help
```

부트로더의 내부명령 중 환경변수 설정, GPIO 핀 제어, 메모리 덤프와 관련된 주요한 명령들을 살펴보면 다음과 같다.

printenv 명령은 설정된 환경변수들을 모두 보여준다. 특정 환경변수의 설정정보를 보려면 다음과 같이 사용할 수 있다.

```
U-Boot> printenv ipaddr
```

참고로, 환경변수의 설정내용을 부트로더의 내부명령에서 인용하여 사용할 때는 다음과 같은 형식을 갖춘다.

```
${kernel_addr_r} // kernel_addr_r에 설정된 값을 인용
```

setenv 명령은 환경변수에 특정 값을 설정하는 명령이다. 통신서비스를 위해 타깃 보드의 IP 주소를 설정하거나 서버측 IP 주소를 설정할 때 다음과 같이 사용할 수 있다.

```
U-Boot> setenv ipaddr 192.168.0.40 // ip address 설정
U-Boot> setenv serverip 192.168.0.20 // server address 설정
```

saveenv 명령은 setenv 명령을 사용하여 환경변수의 설정값이 변경되었을 때 이를 저장하는 명령으로, 설정된 환경변수들을 FAT에 uboot.env 파일로 저장한다.

gpio 명령은 GPIO 핀에 High 혹은 Low 신호 출력 등을 하여 GPIO 핀을 제어하

3.16 라즈베리파이기반 임베디드시스템용

는 명령이다. 이 명령의 형식은 다음과 같다.

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

```
U-Boot> gpio set 18 // BCM GPIO #18에 High 신호 출력
```

md 명령은 memoy dump 명령으로 주소지정된 곳으로부터의 메모리내용을 보여주는 명령으로 다음과 같이 사용될 수 있다.

```
U-Boot> md 0x0C100000
```

tftpboot 명령은 TFTP 서비스를 받기 위한 클라이언트 명령으로 다음과 같은 형식으로 사용된다.

```
tftpboot [적재할 위치] [적재할 파일]
```

다음은 standalone application인 임의 파일을 메모리의 특정위치에 파일 전송하는 예이다.

```
U-boot> tftpboot 0x0C100000 hello_world.bin
```

go 명령은 메모리에 저장된 standalone application을 실행시키는 명령이다.

```
B-boot> go 0x0C100000
```

12.1.4 TFTP 서비스

U-Boot 부트로더의 동작을 실행해 보려면 다른 시스템에서 커널 이미지 파일이나 응용 프로그램의 이진파일 등을 부트로더가 실행중인 상태에서 확보할 필요가 있으며, 이 과정에서 TFTP 서비스를 활용할 수 있다.

TFTP는 시스템간 파일을 전송할 수 있는 통신 프로토콜인 FTP 서비스를 약식으로 구현한 서비스이다. 우선 가상머신에 TFTP 서비스를 구축한다.

가상 머신 환경에서 tftp 서버 구축

보안상의 이유로 Ubuntu에서 기본적으로 TFTP를 더 이상 지원하지 않는다. 따라서 다음 명령을 사용하여 해당 패키지를 다운로드 한다.

```
root@ubuntu:~# apt-get install tftpd-hpa tftp-hpa
```

다음 명령을 사용하여 TFTP를 위한 환경설정 파일 /etc/default/tftp-hpa을 생성 한다.

```
root@ubuntu:~# gedit /etc/default/tftp-hpa
# /etc/default/tftpd-hpa
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/tftpboot"
TFTP_ADDRESS="0.0.0.0:69"
TFTP_OPTIONS="--secure"
```

TFTP 서버에서 공유할 파일이 위치하는 디렉터리의 경로를 "/tftpboot"로 하며, 포트 69번을 사용한다는 의미이다. 따라서 공유할 파일이 위치할 디렉터리를 다음과 같이 만들고, 소유자와 접근권한을 설정한다.

```
root@ubuntu:~# mkdir /tftpboot
root@ubuntu:~# chown nobody /tftpboot
root@ubuntu:~# chmod -R 777 /tftpboot
```

이제 공유할 파일을 /tftpboot 디렉터리에 위치시키면 된다. 부트로더에서 사용할 부트로더 이진파일이나 커널 등의 이미지 파일을 위치시킨다.

TFTP 서비스를 개시하려면 다음과 같은 명령을 사용하며, 환경설정파일의 내용이 변경되는 경우 항상 이 명령을 사용하여 재가동해야 한다.

```
root@ubuntu:~# service tftpd-hpa restart
```

3.18 라즈베리파이기반 임베디드시스템용

서비스 여부를 다음의 명령으로 확인할 수 있다.

```
root@ubuntu:~# servcie tftpd-hpa status
```

실제 접속하여 서비스 되는지 확인하려면, 서비스 디렉터리에 임의 파일을 위치시키고 다음의 명령을 통하여 다운로드되는지 확인하여 정상적으로 다운로드되면 TFTP 서버가 잘 서비스하고 있는 상태이다.

```
root@ubuntu:~# tftp localhost
> get file
> quit
root@ubuntu:~# ls
```

U-Boot에서 TFTP 서비스 받기

우선 환경변수를 사용하여 TFTP 클라이언트(개발보드)의 IP 주소와 TFTP 서버의 IP 주소를 다음 명령을 사용하여 설정한다.

```
U-boot> setenv ipaddr 192.168.0.40 // 개발보드의 IP 주소 설정
U-boot> setenv serverip 192.168.0.20 // 서버(가상머신)의 IP 주소 설정
```

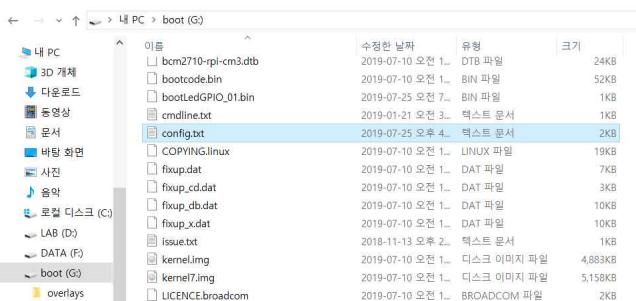
U-Boot 부트로더는 TFTP 클라이언트 역할의 명령으로 tftpboot 명령을 제공한다. 따라서 다음과 같이 사용하여 파일전송 서비스를 받는다.

```
U-boot> tftpboot 0x0C100000 hello_world.bin
```

12.1.5 원상 복구

더 이상 U-Boot 부트로더를 사용하지 않고, 커널 이미지로 부팅하려면 /boot/config.txt 파일에서 추가된 부분을 제거하면 된다. 다양한 방법이 있을 수 있으나, 다음과 같이 Windows 환경에서 SD 카드를 연결하여 원상 복구하는 방법을 소개한다.

라즈베리파이 보드에서 SD 메모리 카드를 제거하고, SD 카드 리더기에 삽입한다. 이 카드 리드기를 Windows 시스템의 USB 포트에 연결한다. 다음과 같이 파일탐색기에서 SD 카드내의 boot 드라이브 선택 후, config.txt 파일을 열어 편집한다.



파일의 끝에 있는 kernel=u-boot.bin 부분을 삭제, 혹은 주석 처리하여 저장한다. 이후, SD 카드를 빼서 라즈베리파이 보드에 삽입하고 전원을 인가하면 커널 이미지로 부팅한다.

12.1.6 실습예제

[실습1] B-boot 부트로더에서 GPIO 제어

B-boot 부트로더가 실행가능한 상태에서 LED 회로를 BCM_GPIO #18에 연결한다. 부트로더 내부 명령을 사용하여 LED를 ON/OFF한다. 모니터와 키보드를 연결하여 SBC 형태에서 작업할 것을 권고한다.

```

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

```

[실습2] Hello_world.bin 실행

U-Boot 부트로더의 내부명령을 활용하여 ./exmples/standalone에 있는 hello_world.bin 이진파일을 실행도록 한다.

우선 standalone application이 메모리에 적재될 주소를 확인할 필요가 있으며, 이는 다음 명령을 통해 확인 가능하다. 강조된 라인의 주소를 잘 기억한다. 이는 부트로더를 통해 메모리 공간에 저장될 시작주소임을 의미한다.

```
$ 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

.......
```

아울러 다음의 두 파일의 내용을 확인한다.

```
./examples/standalone/Makefile
./examples/standalone/hello_world.c
```

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

extra-y      := hello_world
extra-$(CONFIG_SMC91111)          += smc91111_eeprom
extra-$(CONFIG_SMC911X)           += smc911x_eeprom
```

```

extra-$(CONFIG_SPI_FLASH_ATMEL)      += atmel_df_pow2
extra-$(CONFIG_PPC)                  += sched

#
# Some versions of make do not handle tr.....
.....

```

다음의 hello_world.c 소스는 부트로더가 동작되는 방식으로 실행되는 프로그램의 예이다. 일반 응용프로그램과 달리 주 함수명이 main()이 아님을 유의한다.

```

$ cat 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[]) {
    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");

    printf ("argc = %d\n", argc);

    for (i=0; i<=argc; ++i) {
        printf ("argv[%d] = \"%s\"\n",
               i,
               argv[i] ? argv[i] : "<NULL>");
    }

    printf ("Hit any key to exit ... ");

```

```
while (!tstc())
    ;
/* consume input */
(void) getc();
printf ("\n\n");
return (0);
}
```

위 소스 컴파일은 부트로더 소스와 함께 이미 컴파일 되어 있으므로, ./examples/standalone/hello_world.bin을 TFTP 서비스를 받을 수 있도록 TFTP 서비스 디렉터리로 이동시킨다.

라즈베리파이 보드의 부트로더로 U-Boot 부트로더가 실행중인 상태에서 다음의 tftpboot 명령으로 메모리에 저장하고 go 명령으로 실행한다.

```
B-boot> tftpboot 0x0C100000 hello_world.bin
B-boot> md 0x0C100000
B-boot> go 0x0C100000
```

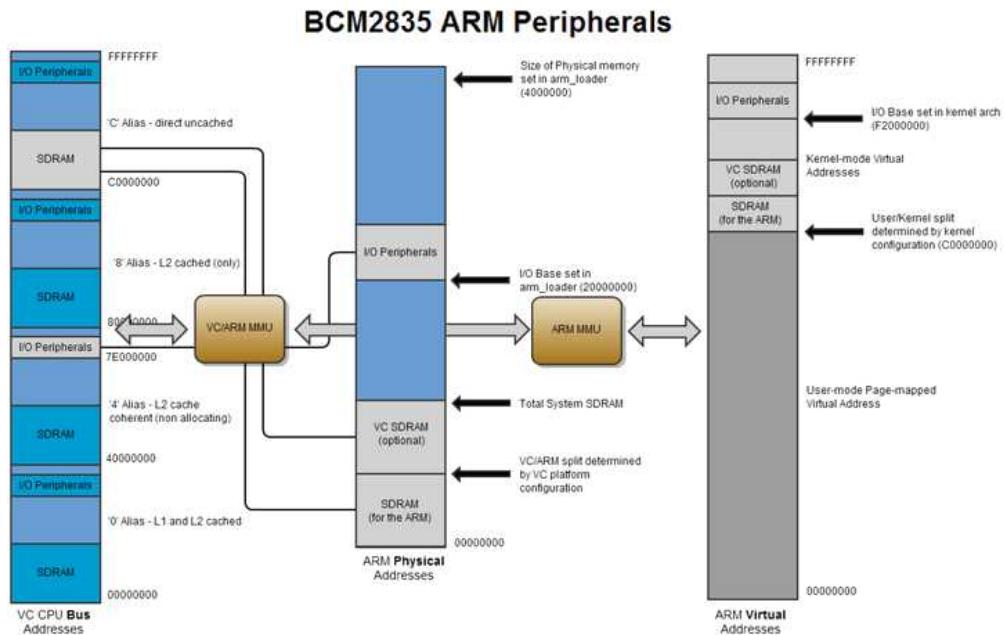
12.2 GPIO

GPIO(general-purpose input output)는 입력이나 출력을 포함한 동작이 런타임 시에 사용자에 의해 제어될 수 있는 디지털 신호 핀을 의미한다. 라즈베리파이 보드에서의 GPIO를 이해하려면 BCM 계열의 칩 데이터시트를 이해할 필요가 있다.

라즈베리파이 B+보드의 핵심 칩은 BCM2835이며, 라즈베리파이 2의 경우 BCM2836, 라즈베리파이 3의 경우 BCM2837이 사용된다. 본 절에서는 BCM2835 자료를 바탕으로 언급하되 라즈베리파이 3의 BCM2837에서 변경된 내용은 따로 강조한다.

12.2.1 BCM2835 메모리 맵

우선 GPIO 핀을 제어하려면 BCM2835에서의 입출력장치에 대한 메모리 맵을 알아야 하며, 메모리 맵은 다음과 같다.

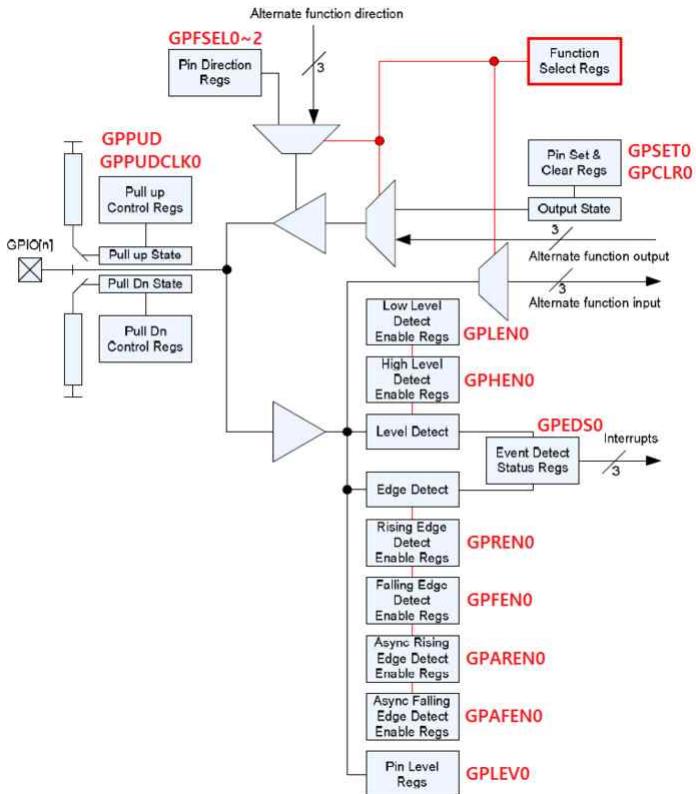


입출력 주변장치를 접근할 때 데이터시트에서는 "VC CPU BUS Address"를 기준으로 레지스터 주소를 표현하고 있으나, 실제로 코드를 작성할 때에는 "ARM Physical Address"를 사용한다. 위 그림과 같이 라즈베리파이 B+에 장착된 BCM2835는 입출력 장치를 위한 베이스 주소(Io Peripheral Base)로 "0x2000_0000" 번지를 사용한다. 하지만 BCM2836이 장착된 라즈베리파이 2, 혹은 BCM2837이 장착된 라즈베리파이 3의 경우 입출력 장치를 위한 베이스 주소(Io Peripheral Base)는 "0x3F00_0000" 번지임을 유의한다.

12.2.2 GPIO 블록도

다음 그림은 라즈베리파이의 GPIO 핀 블록도 및 기능별 관련 레지스터를 보여준

다. 라즈베리파이에서는 54개의 GPIO 핀을 제공하며, 핀의 입출력 방향을 등을 제어하는 기능 선택 레지스터, 핀에 데이터를 입력 혹은 출력하는 것과 관련된 여러 레지스터들이 제공되고 있다.



GPIO 핀 블록도에 보듯이 각 기능별로 약 13가지의 레지스터 유형이 제공되고 있다.

12.2.3 GPIO 관련 레지스터

GPIO 제어와 관련된 주요한 몇 가지 레지스터들에 대해 살펴본다. 기타의 레지스터들의 기능은 데이터시트를 참조한다.

라즈베리파이에서는 GPIO 핀을 총 54개를 제공하나, 실제적으로 사용할 수 있는 GPIO 핀은 보드의 40핀 GPIO 헤더를 통해 활용할 수 있는 28개뿐이다. GPIO 관련한 레지스터의 종류는 아래와 같이 13종이 있으며, 각 레지스터의 물리주소를 확인 할 수 있다. 코드 작성시 BCM2837이 장착된 라즈베리파이 3의 경우 입출력 장치를 위한 베이스 주소(IO Peripherals Base)는 "0x3F00_0000" 번지이며, GPIO 관련 레지스터의 베이스 주소는 입출력장치의 베이스 주소에 0x20_0000을 더한 0x3F20_0000임을 유의한다.

PhysicalAddr	FieldName	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 -	-	-	-
0x3F20_0040	GPEDS0	GPIO Pin Event Detect Status 0	32	R/W
0x3F20_0044	GPEDS1	GPIO Pin Event Detect Status 1	32	R/W
0x3F20_0048	- Reserved -	-	-	-
0x3F20_004C	GPREN0	GPIO Pin Rising Edge Detect Enable 0	32	R/W
0x3F20_0050	GPREN1	GPIO Pin Rising Edge Detect Enable 1	32	R/W
0x3F20_0054	- Reserved -	-	-	-
0x3F20_0058	GPFEN0	GPIO Pin Falling Edge Detect Enable 0	32	R/W
0x3F20_005C	GPFEN1	GPIO Pin Falling Edge Detect Enable 1	32	R/W
0x3F20_0060	- Reserved -	-	-	-
0x3F20_0064	GPHEN0	GPIO Pin High Detect Enable 0	32	R/W
0x3F20_0068	GPHEN1	GPIO Pin High Detect Enable 1	32	R/W
0x3F20_006C	- Reserved -	-	-	-
0x3F20_0070	GPLEN0	GPIO Pin Low Detect Enable 0	32	R/W

```
0x3F20_0074 GPLEN1 GPIO Pin Low Detect Enable 1 32 R/W
0x3F20_0078 - Reserved - -
0x3F20_007C GPAREN0 GPIO Pin Async. Rising Edge Detect 0 32 R/W
0x3F20_0080 GPAREN1 GPIO Pin Async. Rising Edge Detect 1 32 R/W
0x3F20_0084 - Reserved - -
0x3F20_0088 GPAFEN0 GPIO Pin Async. Falling Edge Detect 0 32 R/W
0x3F20_008C GPAFEN1 GPIO Pin Async. Falling Edge Detect 1 32 R/W
0x3F20_0090 - Reserved - -
0x3F20_0094 GPPUD GPIO Pin Pull-up/down Enable 32 R/W
0x3F20_0098 GPPUDCLK0 GPIO Pin Pull-up/down Enable Clock 0 32 R/W
0x3F20_009C GPPUDCLK1 GPIO Pin Pull-up/down Enable Clock 1 32 R/W
0x3F20_00A0 - Reserved - -
0x3F20_00B0 - Test 4 R/W
```

특정 GPIO 핀을 LED 등과 같은 출력 디바이스를 연결하는 경우 관련 레지스터는 GPFSELn, GPSETn, GPCLRn이며, 버튼 스위치와 같은 입력 디바이스를 연결하는 경우 관련 레지스터는 GPFSELn, GPLEVn 등이 최소로 사용된다.

GPFSELn(GPIO Function Select Registers)

GPFSELn은 입력용이냐, 출력용이냐 등의 기능을 선택하는데 사용하는 레지스터이다. 특정 GPIO 핀의 기능 선택을 위해 3비트씩 사용되며, BCM2835는 총 54개의 GPIO를 지원하므로 총 162비트가 필요하다. 하지만 사용가능한 레지스터의 사이즈는 32비트이다. 따라서 32비트 길이의 레지스터에는 10개 GPIO 핀에 대한 기능 설정이 가능하고 2비트가 남는다. 즉, 32 비트 길이의 GPFSEL 레지스터 하나에 10개의 GPIO 핀에 대한 기능 설정이 가능하므로 54개 GPIO 핀을 위해서는 총 6개의 GPFSEL이 필요하다. 그래서 GPFSELn(n=0,...,5)의 6개 레지스터 사용된다.

예로 다음 그림은 GPFSELn의 비트 구조를 보인다. GPFSEL0 레지스터의 최하위 비트부터 3비트씩 각 GPIO 핀의 기능을 설정하는 것이며, 최하위쪽은 BCM_GPIO #0을 위한 것으로 10개의 GPIO 핀에 대한 설정을 할 수 있으며, 최상위 2비트는 사용되지 않는다. 54개 GPIO 핀을 제공하므로 GPFSEL5 레지스터의 하위 12비트는 사용되나 나머지 상위 비트들은 사용되지 않는다.

		GPIO_9				GPIO_8			...	GPIO_2			GPIO_1			GPIO_0			
		31	30	29	28	27	26	25	24		8	7	6	5	4	3	2	1	0
GPFSEL0	-	-																	
GPFSEL1	-	-																	
GPFSEL2	-	-																	
GPFSEL3	-	-																	
GPFSEL4	-	-																	
GPFSEL5	-	-	-	-	-	-	-	-	-										

기능의 선택은 해당 핀을 입력용으로 설정하는 경우는 000으로, 출력용으로 설정하는 경우 001을 설정하며, 나머지 6가지 경우는 GPIO 핀을 다른 기능으로 사용하도록 설정하는 것이며 데이터시트를 참조한다.

GPSETn(GPIO Pin Output Set Registers, n=1,2)

GPSETn은 32비트의 길이의 레지스터로 각 GPIO 핀당 1비트씩 할당되며, 총 32개의 GPIO 핀을 제어할 수 있으며, 해당 GPIO 핀에 HIGH 신호를 출력하고자 할 때 해당 비트를 1로 설정한다.

	31	30	29	28	27	26	25	24	23	22	21	...	6	5	4	3	2	1	0
GPSET0																			
GPSET1	-	-	-	-	-	-	-	-	-	-	-								

GPSET0는 BCM_GPIO #0부터 BCM_GPIO #31까지, GPSET1은 BCM_GPIO #32부터 BCM_GPIO #53까지를 위해 사용된다.

GPCLRn(GPIO Pin Output Clear Registers, n=1,2)

GPCLRn 레지스터는 32비트의 길이의 레지스터이며 각 GPIO 핀당 1비트씩 할당되며, 총 32개의 GPIO 핀을 제어할 수 있으며, 해당 GPIO 핀에 LOW 신호를 출력하고자 할 때 해당 비트를 1로 설정한다.

	31	30	29	28	27	26	25	24	23	22	21	...	6	5	4	3	2	1	0
GPCLR0																			
GPCLR1	-	-	-	-	-	-	-	-	-	-	-								

GPCLR0는 BCM_GPIO #0부터 BCM_GPIO #31까지, GPCLR1은 BCM_GPIO #32

부터 BCM_GPIO #53까지를 위해 사용된다.

GPLEVn(GPIO Pin Level Registers, n=1,2)

GPLEVn는 32비트의 길이를 가지며, 각 GPIO 핀당 1비트씩 할당되며, 총 32개의 GPIO 핀을 제어할 수 있으며, 해당 GPIO 핀에 인가된 신호가 반영되는 레지스터이다. 즉, 이 레지스터의 해당 GPIO 핀에 인가된 신호를 읽어들이면 해당 GPIO 핀으로부터 신호를 입력받는 것이다.

	31	30	29	28	27	26	25	24	23	22	21	...	6	5	4	3	2	1	0
GPLEV0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
GPLEV1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

12.2.4 실습예제

[실습3] LED 제어 (부트로더 실행방식)

U-Boot 부트로더가 실행되는 방식으로 응용프로그램의 실행하되 GPIO 핀에 연결된 LED를 ON/OFF하기를 반복하는 프로그램을 작성한다. 이때 LED 회로가 연결된 GPIO 핀을 접근하려면 해당 GPIO 핀의 물리 주소를 사용해야 함을 유의한다. 또한 BCM_GPIO #18에 연결하며, 이 핀 번호는 BCM 주소체계 임을 유의한다.

다음 명령으로 작업 디렉터리로 이동하여, Makefile을 수정하고, 소스를 작성한다.

```
$ cd ./examples/standalone
$ nano Makefile
.....
extra-y      := bootLedGPIO_01          // 작성할 파일명으로
.....
```

```
$ nano bootLedGPIO_01.c
//=====
// bootLedGPIO_01.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 bootLedGPIO_01(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] &= ~(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(100);

        //          29     20 19     10 9      0
        // GPCLR0 = 0b00 0000000000 0100000000 0000000000;
        gpio[GPCLR0 / 4] |= (1 << 18);      // LED OFF
        printf("LED OFF....\n");
        mdelay(100);
    }

    return 0;
}

```

```
$ cd ../../  
$ sudo make clean  
$ sudo make -j4
```

```
pi@raspberrypi:~/IFC415/12_GPIO/UBoot/u-boot $ ls examples/standalone/  
Makefile          bootLedGPIO_01.o      nds32.lds      sparc.lds  
Makefile_org       bootLedGPIO_01.srec    ppc_longjmp.S   stubs.c  
README.smc9111_eeprom bootLedGPIO_01.su      ppc_setjmp.S   stubs.o  
atmel_df_pow2.c    hello_world.c     riscv.lds      stubs.su  
bootLedGPIO_01     libstubs.o        sched.c  
bootLedGPIO_01.bin  mips.lds         smc9111_eeprom.c  
bootLedGPIO_01.c    mips64.lds       smc911x_eeprom.c  
pi@raspberrypi:~/IFC415/12_GPIO/UBoot/u-boot $
```

부트로더의 내부명령을 사용하여 bootLedGPIO_01.bin을 메모리에 적재하고 실행하여 결과를 살펴본다.

```
B-boot> tftpboot 0x0C100000 bootLedGPIO_01.bin  
B-boot> go 0x0C100000
```

이를 통해 운영체제없이 부트로더가 실행되는 방식으로 물리주소를 이용하여 GPIO 핀에 연결된 디바이스를 제어할 수 있음을 알 수 있다.

[실습4] 커널 이미지로 부팅 (미실습)

U-Boot 부트로더의 내부명령을 사용하여 커널 이미지를 사용하여 부팅하는 과정을 살펴본다.

참고로 U-Boot 부트로더의 내부명령을 사용하여 커널 이미지를 적재하고 부팅하는 과정은 다음과 같다. 라즈베리파이보드를 위해 커널 이미지가 위치할 주소 kernel_addr_r는 0x0008_0000로, fdt_addr_r는 0x0260_0000으로 설정되어 있다. bootz 명령에서 인자가 여럿일 경우 그 구분을 위해 -를 사용한다.

```
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}
```

[실습5] LED 제어용 부트로더 (미구현)

부트로더 실행 방식으로 실행되는 응용프로그램을 통해 주변입출력 장치를 제어한다. 운영체제없이 오직 해당 기능을 하는 임베디드시스템화라고 볼 수 있다.

자료출처

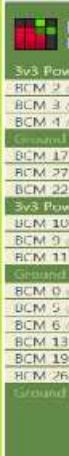
[1] SoC BCM2835

<http://blog.naver.com/PostView.nhn?blogId=kiatwins&logNo=220773150389&parentCategoryNo=&categoryNo=&viewDate=&isShowPopularPosts=false&from=postView>

[2] U-boot <https://dalgong2.tistory.com/2>

[3] U-boot와 부트로더 <https://jhhkim3624.tistory.com/83>

[4] U-boot와 TFTP <https://dalgong2.tistory.com/3>



제13장 커널 컴파일 및 커널 모듈

커널 컴파일과 커널 모듈을 구현하여 커널에 동적으로 링크하여 디바이스를 제어하는 방법에 대해 살펴본다.

13.1 커널 컴파일

전체 과정은 커널 소스를 다운로드하여 설치하고 커널 환경설정, 커널 컴파일, 새로운 커널 이미지로 부팅으로 구성된다.

우선, 다음 명령을 통해 현재의 라즈베리파이 보드의 커널 버전은 4.19.42 버전으로 확인된다.

```
pi@raspberrypi:~ $ uname -r  
4.19.42-v7+
```

또한 현재의 가상머신의 커널 버전은 4.15.0으로 확인된다.

```
root@ubuntu:~# uname -r  
4.15.0-43-generic
```

2019년도 8월 현재 일반적인 컴퓨터 환경에서 커널 컴파일에 소요되는 시간은 가상머신에서 10여분, 라즈베리파이보드에서 90여분정도 소요됨을 참고한다. 따라서 여기서는 가상 머신에서 작업하는 것을 기준하여 설명한다.

13.1.1 커널 소스 설치

커널 소스의 다운로드

커널 소스는 여러 방법으로 다운로드 할 수 있다. Windows 환경에서 웹 서비스를

334 라즈베리파이 기본 임베디드 시스템 활용

통해 다운로드하거나, Lunux 환경에서 git 툴 혹은 wget 툴 등을 통하여 다운로드할 수 있다.

Windows 환경에서 다운로드하려면 웹브라우저를 통해 다음의 사이트에 접속하여 2018년 2월 현재 최신 버전의 커널 소스 압축파일인 rpi-4-9.y.tar.gz을 다운로드 한다.

<https://github.com/raspberrypi/linux/archive/rpi-4.9.y.tar.gz>

이 파일을 삼바서비스 등을 이용하여 가상머신이나 라즈베리파이 보드의 적당한 디렉터리로 복사한다. 위 커널 소스파일은 학습자료에 포함되어 있음을 참고한다.

가상머신이나 라즈베리파이 보드의 Linux 환경에서 커널 소스를 다운로드 할 때는 wget 툴 혹은 git 툴을 사용할 수 있다. wget 툴의 경우 압축된 파일을 다운로드 하는 반면, git 툴은 압축되지 않은 커널 소스를 다운로드한다. 따라서 wget 툴을 사용하는 것이 상대적으로 좀 더 빠르다.

wget 툴을 사용할 경우 다음의 명령들을 사용한다. 현재의 작업 디렉터리에 해당 커널소스의 tar 파일이 다운로드 된다.

```
# apt-get install wget // wget 패키지 설치  
# wget https://github.com/raspberrypi/linux/archive/rpi-4.9.y.tar.gz  
# ls  
rpi-4-9.y.tar.gz
```

git 툴을 사용할 경우 다음의 명령들을 사용한다. 최신 버전의 커널 소스가 현재의 작업 디렉터리에 linux/ 라는 디렉터리를 생성하여 그 하부에 다운로드 된다. 2019년 7월 현재 다운로드한 커널 소스의 버전은 4.19.60으로 확인된다.

```
# apt-get install git  
# git clone --depth=1 https://github.com/raspberrypi/linux.git  
# ls  
linux
```

커널 소스 설치

우선 다운로드 받은 압축파일을 다음의 tar 명령을 통해 풀어 놓는다. git 툴로 다운로드한 경우 이 과정이 불필요하다.

```
# tar xvfz rpi-4.9.y.tar.gz
```

현 작업디렉터리에 커널 소스가 있는 linux-rpi-4.9.y/ 디렉터리가 생성된다.

```
# ls  
linux-rpi-4.9.y
```

Linux 환경에서 커널 소스는 관례적으로 /usr/src 디렉터리 하부에 위치한다. 따라서 다음의 명령을 사용하여 생성된 커널 소스 linux-rpi-4.9.y 디렉터리 이하를 /usr/src/ 디렉터리로 옮겨놓는다.

```
# mv linux-rpi-4.9.y /usr/src/
```

컴파일시 편의를 위해 다음의 명령을 사용하여 커널 소스 디렉터리를 linux라는 링크파일명으로 심볼릭 링크한다.

```
# cd /usr/src  
# ln -s ./linux-rpi-4.9.y/ linux  
# ls -al  
lrwxrwxrwx 1 root root 16 Jul 30 21:15 linux -> linux-rpi-4.99.7/  
drwxrwxr-x 24 root root 4096 Jan 31 2018 linux-rpi-4.9.y
```

이후 linux-rpi-4.9.y 커널 소스의 홈디렉터리로 경로명은 /usr/src/linux가 된다. 즉 다음의 명령을 사용하면 linux-rpi-4.9.y 커널 소스의 홈디렉터리가 현 작업디렉터리가 된다.

```
# cd linux
```

참고로, 심볼릭 링크방법은 여러 버전의 커널 소스를 사용할 때 유용한 방법일 수

있다. 즉 또 다른 버전의 커널 소스가 linux-rpi-4.19.60로 존재할 때 이를 다음 명령으로 심볼릭 링크함으로써 경로 /usr/src/linux는 linux-rpi-4.19.60 디렉터리를 의미한다.

```
# ln -s ./linux-rpi-4.19.60/ linux
# ls -al
lrwxrwxrwx 1 root root 16 Jul 30 21:15 linux -> linux-rpi-4.19.60/
drwxr-xr-x 27 root root 4096 Jul 30 19:47 linux-rpi-4.19.60
drwxrwxr-x 24 root root 4096 Jan 31 2018 linux-rpi-4.9.y
```

커널 소스의 버전 확인

커널 소스의 버전을 확인하려면 다음의 명령을 사용하여 Makefile의 앞부분을 살펴본다. 커널 버전은 4.9.79로 확인된다.

```
# head ./linux-rpi-4.9.y/Makefile
VERSION = 4
PATCHLEVEL = 9
SUBLEVEL = 79
EXTRAVERSION =
NAME = Roaring Lionus
.....
```

13.1.2 커널 환경 설정

다음의 명령을 사용하여 커널소스의 홈 디렉터리로 이동한다. 다음의 명령은 심볼릭 링크가 어느 버전의 커널소스로 되어 있느냐에 따라 커널소스의 버전이 다를 수 있음을 유의한다.

```
# cd /usr/src/linux
```

커널 소스에는 커널이 실행될 시스템에 맞게 잘 정의된 환경 설정된 파일들이 제공되며, 이들 환경설정 파일들은 ./arch 디렉터리로 제공된다. 라즈베리파이보드의 경우 ./arch/arm/configs/ 디렉터리에 존재한다.

다음의 명령으로 라즈베리파이 보드를 위해 정의된 환경설정 파일들을 볼 수 있다. 라즈베리파이 1의 경우 bcmrpi_defconfig 파일을, 라즈베리파이 2, 3의 경우 bcm2709_defconfig 파일을, 라즈베리파이 제로의 경우 bcm2835_defconfig 파일을 사용한다.

```
# ls -l ./arch/arm/configs/bcm*
-rw-r--r-- 1 root root 34666 Jul 30 19:46 bcm2709_defconfig
-rw-r--r-- 1 root root 3973 Jul 30 19:46 bcm2835_defconfig
-rw-r--r-- 1 root root 35002 Jul 30 19:46 bcmrpi_defconfig
```

실습환경이 라즈베리파이3이므로 다음의 명령으로 bcm2709_defconfig 환경파일을 이용하여 커널 환경설정을 한다.

```
# make bcm2709_defconfig
HOSTCC scripts/basic/fixdep
.....
# ls -al .con*
-rw-r--r-- 1 root root 145823 Aug  1 11:54 .config
-rw-r--r-- 1 root root 145823 Aug  1 11:54 .config.old
```

커널 컴파일할 때 환경설정을 위한 참조 파일은 .config 헤든 파일이다. 위의 make 명령은 .config 헤든 파일의 내용을 bcm2709_defconfig의 내용으로 대체하는 것과 다르지 않다.

다음으로 커널의 환경 설정 정보를 항목별로 확인하거나 변경하려면 다음의 명령을 사용한다. 설정 내용을 변경하여 저장하고 종료한 경우, 변경된 정보는 .config 헤든 파일에 반영된다.

```
# make menuconfig
HOSTCC scripts/kconfig/mconf.o
In file included from scripts/kconfig/mconf.c:23:0:
scripts/kconfig/lxdialog/dialog.h:38:20: fatal error: curses.h: No such file
or directory
```

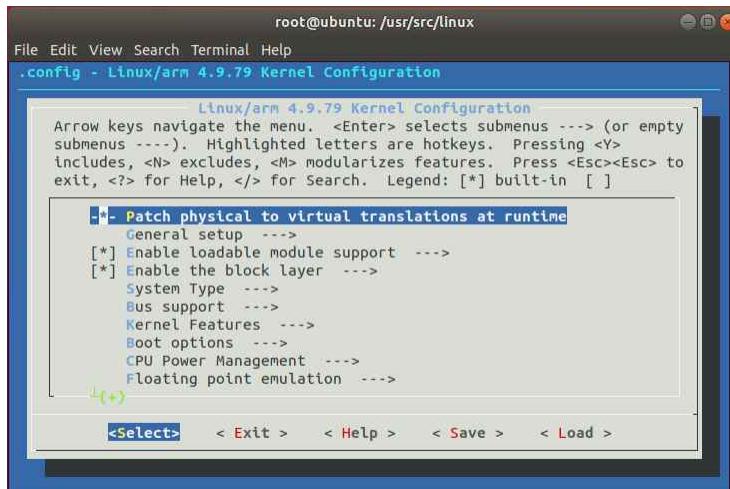
```
#include CURSES_LOC  
^  
compilation terminated.
```

위와 같이 위 명령을 실행할 때 문제가 발생할 경우는 ncurses 패키지가 설치되지 않아 발생하는 것이므로, 다음 명령으로 libncurses5-dev 패키지를 설치한 후 다시 시도한다.

```
# apt-get install libncurses5-dev
```

다음 명령이 문제 없이 실행된 경우 아래 그림과 같은 환경설정 정보를 확인할 수 있는 화면을 볼 수 있으며, 해당 내용은 .config 하든 파일에 설정된 내용과 동일하다.

```
# make menuconfig
```



13.1.3 커널 컴파일

커널 소스를 컴파일하는 것은 타깃보드에서 실행될 커널 이미지를 생성하는 과정이므로 가상머신에서 작업중이라면 크로스컴파일러로 컴파일되어야 한다. 따라서 크로스 컴파일러 설치 여부와 경로지정이 제대로 되었는지 다시 한번 확인하다.

ARCH와 CROSS_COMPILE 환경변수는 커널 컴파일할 때 Makefile에서 크로스컴파일을 위한 환경변수로 사용된다.

```
# cd
# ls .bashrc
# cross compiler .... raspberrypi
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
```

또한 컴파일과정에서 커널이름을 나타내는 KERNEL 환경변수를 참조한다. 라즈베리파이 1의 경우 커널이름은 kernel이며, 라즈베리파이 2, 3의 경우 kernel7이다. 이는 부팅할 때 커널 이미지 파일의 파일명과 동일해야 한다. 실습환경에 맞게 다음과 같이 KERNEL 환경변수를 설정한다.

```
# KERNEL=kernel7
```

다음으로 커널 컴파일중에 시상수 관련하여 계산 과정에서 bc 계산기가 사용되므로 다음의 명령으로 bc 패키지를 설치한다. 설치되어 있지 않은 경우 커널 컴파일 과정에서 bc 패키지가 없다는 오류가 발생한다.

```
# apt-get install bc
```

커널 소스의 컴파일은 다음과 같은 명령을 사용한다. -j4 옵션은 시스템 의존적인 옵션으로 CPU가 4개 코어를 지원하는 경우 4개 코어를 사용하여 컴파일하겠다는 의미이다.

```
# make -j4 zImage modules dtbs
```

정상적으로 컴파일되면 컴파일된 이미지 파일은 ./arch/arm/boot/ 디렉터리에 zImage 파일로 생성되며 부가적인 파일들 또한 이 디렉터리 하부에 생성된다.

```
pi@raspberrypi:/usr/src/linux/arch/arm/boot $ ls -al
total 14396
drwxr-xr-x  5 root root    4096 Feb  7 17:55 .
drwxr-xr-x  99 root root   4096 Feb  7 13:37 ..
drwxr-xr-x  2 root root   4096 Feb  7 13:37 bootp
drwxr-xr-x  2 root root   4096 Feb  7 17:55 compressed
drwxr-xr-x  4 root root  69632 Feb  7 17:55 dtbs
-rw-r--r--  1 root root     47 Feb  7 13:37 .gitignore
-rw xr-xr-x  1 root root 13053952 Feb  7 17:55 Image
-rw-r--r--  1 root root    89 Feb  7 17:55 .Image.cmd
-rw-r--r--  1 root root   1648 Feb  7 13:37 install.sh
-rw-r--r--  1 root root   2816 Feb  7 13:37 Makefile
-rw xr-xr-x  1 root root  4574304 Feb  7 17:55 zImage
-rw-r--r--  1 root root    116 Feb  7 17:55 .zImage.cmd
pi@raspberrypi:/usr/src/linux/arch/arm/boot $
```

다음 명령으로 zImage 파일의 파일유형을 살펴보면, 타깃보드인 ARM 보드용의 부팅가능한 실행파일로 크로스컴파일이 잘 이루어진 것을 확인할 수 있다.

```
# file ./arch/arm/boot/zImage
./arch/arm/boot/zImage: Linux kernel ARM boot executable zImage
(little-endian)
```

생성된 커널 이미지가 제대로 실행되기 위해서는 동일 버전의 커널 모듈 라이브러리들이 필요하다. 다음의 명령은 해당 버전의 커널 모듈 라이브러리 디렉터리를 /lib/modules/에 생성한다. 이 디렉터리는 커널이미지가 실행될 때나, 커널 모듈을 컴파일할 때 활용되는 라이브러리 디렉터리로써의 역할을 하게 된다. 커널 이미지 파일과 이 라이브러리 디렉터리는 한 몸으로 움직여야 한다.

```
# make modules_install
```

위 명령은 /lib/modules/ 디렉터리에 커널버전명의 디렉터리를 4.9.79-v7/를 생성하여 커널 모듈들을 복사한다. 또한 커널 소스가 위치하는 /usr/src/linux-rpi-4.9.y를 심볼릭 링크하는 build, source 라는 파일명의 2개 링크파일을 해당 디렉터리 내에 생성하며, 커널 모듈 컴파일시 유용하게 사용된다.

```
# ls /lib/modules
4.9.79-v7
```

만일 커널 소스를 다시 컴파일할 경우에는 다음의 명령으로 기존 컴파일된 목적코드 등을 우선 제거하고 컴파일할 수 있다.

```
# make clean
```

13.1.4 새 커널 이미지 적용

다음과 같이 ./arch/arm/boot 디렉터리에 생성된 zImage는 타깃보드의 /boot/ 디렉터리에 파일명 kernel7.img로 복사함으로써 새로운 커널 이미지로 사용할 수 있다. 이때 동일 버전의 커널 모듈 라이브러리들도 함께 타깃 보드에 설치되어야 한다.

새로운 버전의 커널 이미지 파일과 커널 모듈 라이브러리를 NFS 서비스를 사용하여 라즈베리파이 보드로 전달한다.

우선 다음의 명령들을 사용하여 가상머신의 NFS 서비스를 위한 /nfs 디렉터리로 관련 파일들을 복사한다.

```
# cd ./arch/arm/boot
# cp zImage /nfs
# cp -rf /lib/modules/4.9.79-v7 /nfs
```

참고로, 새로운 커널로 부팅하는데 문제가 있을 경우 다음의 파일들을 라즈베리파이 보드에 복사할 필요가 있을 수 있다.

```
cp ./arch/arm/boot/dts/*.dtb /boot/
cp ./arch/arm/boot/dts/overlays/*.dtbo /boot/overlays/
cp ./arch/arm/boot/dts/overlays/README /boot/overlays/
```

라즈베리파이 보드에서

다음의 명령을 사용하여 기존 커널 이미지 파일을 적절한 이름으로 퇴피시켜 놓는다. 이는 나중에 복구하기 위한 것이다.

```
$ cd /boot
```

```
$ sudo mv kernel7.img kernel7_org.img
```

라즈베리파이 보드에서 다음과 같이 가상머신을 /nfs 디렉터리를 라즈베리파이 보드의 /share에 마운트한다.

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

다음 명령으로 새로운 커널 이미지 파일 zImage를 kernel7.img 파일로 /boot/ 디렉터리로 복사하고, 커널 모듈들을 /lib/modules 디렉터리로 복사한다.

```
$ sudo cp /share/zImage kernel7.img  
$ sudo cp -rf /share/4.9.79-v7 /lib/modules
```

새로운 커널 이미지로 부팅할 준비가 완료되었으므로 다음의 명령으로 재부팅하고 커널 버전을 확인한다.

```
$ sudo reboot  
$ uname -r  
4.9.79-v7
```

13.2 가상주소에 의한 디바이스 제어

13.2.1 GPIO 핀의 가상주소

커널이 부팅된 후에 GPIO 핀에 연결된 디바이스를 제어하려면, 메모리 디바이스 파일 /dev/mem을 개방한 후, mmap() 함수를 이용하여 GPIO 핀의 물리 주소에 대한 가상 주소로 접근해야 한다.

다음 명령으로 메모리 디바이스 파일인 /dev/mem을 살펴보면 문자 디바이스이며, 슈퍼유저 소유이다.

```
$ ls -l /dev/mem
crw-r----- 1 root kmem 1, 1 Aug 2 13:03 /dev/mem
```

GPIO 핀에 연결된 LED 디바이스를 가상 주소를 통해 접근하는 경우 전체적인 절차를 코드로 간략히 표현하면 다음과 같다.

```
fd = open("/dev/mem", O_RDWR);
addr = (char *)mmap(NULL, 4096,
                     PROT_READ | PROT_WRITE, MAP_SHARED,
                     fd, GPIO_BASE);
*addr = 0xFF;
munmap(addr, 1);
close(fd);
```

우선 open() 함수를 사용하여 메모리 디바이스 파일 /dev/mem를 개방하고, 파일 기술자를 반환받는다. 물리 주소를 활용한 mmap() 함수를 사용하여 가상 주소를 포인터로 얻는다. 이 포인터를 사용하여 제어할 디바이스에 데이터를 입출력한다. 다음으로 가상 주소를 위한 포인터의 메모리 공간을 munmap() 함수를 사용하여 해제하고, 개방된 메모리 디바이스 파일을 close() 함수를 사용하여 닫는다.

다음의 mmap() 함수는 물리 주소를 사용하여 가상 주소를 확보하는 함수이다.

```
void * mmap(void *start, size_t length, int prot, int flags,
           int fd, off_t offset);
```

start는 특정 메모리주소를 요청하기 위해 사용하며 통상 NULL로 설정하며, length는 메모리 세그먼트의 길이를 나타낸다, prot는 메모리 세그먼트에 접근할 때 접근허용 권한을 설정하는 항목으로, PROT_READ, PROT_WRITE, PROT_EXEC, PROT_NONE의 상수가 정의되어 있으며, 각각 읽기 허용, 쓰기 허용, 실행 허용, 접근 불허를 의미한다. flags는 페이지에 대한 변경이 다른 곳에 반영되는 방법을 설정하는 것으로 MAP_PRIVATE, MAP_SHARED, MAP_FIXED 상수가 정의 되어 있으며, 각각 다른 프로세스와 대응영역을 비공유, 공유, 지정된 주소로 고정을 의미한다. fd는 메모리 디바이스 파일의 파일기술자이며, offset은 디바이스의 물리 주소를 의미한다.

다음의 munmap() 함수는 mmap() 함수로 매핑된 메모리를 반환하는 기능을 한다.

```
int munmap(void *start, size_t length);
```

13.2.2 실습예제

GPIO 핀의 가상주소를 통해 해당 GPIO 핀에 연결된 입출력 디바이스를 구동하기 위한 실습을 진행한다. 사용된 입출력 디바이스는 LED와 BTN이다.

[실습1] GPIO LED 제어

다음 소스는 BCM_GPIO #18에 연결된 LED를 ON/OFF하기를 반복하는 프로그램이다. LED 회로의 연결은 BCM 주소체계 임을 유의한다.

```
$ nano ledGPIO_01.c
//=====
// ledGPIO_01.c
//      BCM_GPIO #18, LED
//      using mmap()
//=====

#include <stdio.h>
#include <unistd.h>           // sleep(), close()
#include <fcntl.h>
#include <sys/mman.h>

#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) {
    int fd;
    char * gpio_mmap;
```

```

volatile unsigned int * gpio;
int i;

printf("[mmapGPIO testing.....LED]\n");

if((fd = open("/dev/mem", O_RDWR | O_SYNC)) < 0) {
    printf("/dev/mem open error!!\n");
    return 1;
}

gpio_mmap = (char *)mmap(NULL, 4096,
    PROT_READ | PROT_WRITE,
    MAP_SHARED, fd, GPIO_BASE);
if(gpio_mmap == MAP_FAILED) {
    printf("mmap() error!!\n");
    return 1;
}

gpio = (volatile unsigned int *)gpio_mmap;

//          19   18           11   10
// GPFSEL1 = 0bxx xxx 001 xxx xxx xxx xxx xxx xxx xxx xxx xxx;
gpio[GPFSEL1 / 4] &= ~(6 << 24);      // #18, output

while(1) {
    //          29     20 19     10 9      0
    // GPSET0 = 0b00 0000000000 0100000000 0000000000;
    gpio[GPSET0 / 4] |= (1 << 18);      // LED ON
    printf("LED ON.....\n");
    sleep(1);

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

munmap(gpio_mmap, 4096);
close(fd);

return 0;

```

```
}
```

```
$ gcc -o ledGPIO_01 ledGPIO_01.c
혹은, $ make ledGPIO_01
```

응용 프로그램을 실행할 때는 /dev/mem 디바이스 파일을 개방해야 하므로, 필히 슈퍼유저 권한으로 다음과 같이 실행한다.

```
$ sudo ./ledGPIO_01
```

[실습2] GPIO LED 및 BTN 제어

다음 소스는 BCM_GPIO #17에 연결된 버튼 스위치를 누르는 동안 LED를 ON하는 프로그램이다. LED는 BCM_GPIO #18에 연결한다.

```
$ nano ledGPIO_02.c
//=====
// ledGPIO_02.c
//      BCM_GPIO #18, LED
//      BCM_GPIO #17, BTN
//      using mmap()
//=====

#include <stdio.h>
#include <unistd.h>           // sleep(), close()
#include <fcntl.h>
#include <sys/mman.h>

#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

int main(void) {
    int fd;
```

```

char * gpio_mmap;
volatile unsigned int * gpio;
int i, ret;

printf("[mmapGPIO testing.....LED/BTN]\n");

if((fd = open("/dev/mem", O_RDWR | O_SYNC)) < 0) {
    printf("/dev/mem open error!!\n");
    return 1;
}

gpio_mmap = (char *)mmap(NULL, 4096,
                        PROT_READ | PROT_WRITE,
                        MAP_SHARED, fd, GPIO_BASE);
if(gpio_mmap == MAP_FAILED) {
    printf("mmap() error!!\n");
    return 1;
}

gpio = (volatile unsigned int *)gpio_mmap;

gpio[GPFSEL1 / 4] &= ~(7 << 21);           // #17, input
gpio[GPFSEL1 / 4] &= ~(6 << 24);           // #18, output

while(1) {
    ret = (gpio[GPLEV0 /4] & (1 << 17));   // press?
    if(ret) {
        printf("Pressed, then LED ON.....\n");
        gpio[GPSET0 / 4] |= (1 << 18); // LED ON
    }
    else {
        printf("Released, then LED OFF....\n");
        gpio[GPCLR0 / 4] |= (1 << 18); // LED OFF
    }
}

munmap(gpio_mmap, 4096);
close(fd);

return 0;
}

```

```
$ gcc -o ledGPIO_02 ledGPIO_02.c
```

```
혹은, $ make ledGPIO_02
```

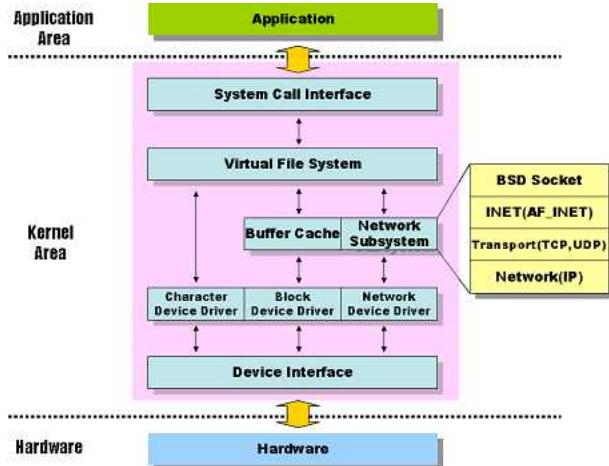
```
$ sudo ./ledGPIO_02
```

13.3 커널 모듈

입출력 디바이스를 제어하기 위한 커널 모듈은 디바이스 드라이버라고 칭하기도 한다. 본 절에서는 커널의 구조, 커널 모듈의 구성 및 커널 모듈을 커널에 동적으로 링크하여 활용하기 위한 관련 명령들을 살펴본다.

13.3.1 Linux 커널 구조

리눅스 커널의 구조는 다음 그림과 같다. 리눅스 커널은 하드웨어와 사용자 응용프로그램 사이의 인터페이스를 제공한다. 특히 각각의 입출력 장치를 제어하기 위한 디바이스 드라이버는 디바이스 유형에 따라 문자형, 블록형, 네트워크형 디바이스로 분류되며, 이들 장치를 제어할 때는 가상 파일시스템상의 디바이스 파일 형태로 제공된다. 사용자 응용 프로그램은 디바이스 드라이버와 사상되는 디바이스 파일을 개방하여 각 입출력 장치를 제어한다.



디바이스 유형

입출력 디바이스는 각 디바이스의 성격에 따라 3가지 유형으로 분류할 수 있다.

문자(character) 디바이스는 해당 디바이스와의 데이터 입출력시 데이터를 문자단위로 순차적인 전송특성을 지니며 입출력 버퍼가 필요없는 디바이스를 의미한다. 이러한 종류의 디바이스로는 키보드, 마우스, 프린터, 단말장치 등이 있다.

문자형 디바이스 파일들은 다음 명령에 의한 결과와 같이 디바이스 파일 유형이 c로 표현되는 디바이스를 말한다.

```
$ ls -l /dev/tty?
crw--w---- 1 root tty 4, 0 Aug  6 14:10 /dev/tty0
crw--w---- 1 root tty 4, 1 Aug  6 14:10 /dev/tty1
.....
```

블록(block) 디바이스는 데이터를 블록단위로 버퍼를 사용하여 임의 접근이 가능한 디바이스를 말한다. 리눅스에서 이러한 블록 디바이스는 512B, 혹은 1024B 크기의 블록단위로 데이터를 입출력하며 파일시스템을 구축할 수 있는 디바이스이기도 하다. 대표적인 블록 디바이스로는 CD-ROM, 하드디스크, 램디스크 등이 있다.

이들 블록 디바이스는 디바이스 파일들은 다음 명령에 의한 결과와 같이 디바이스 파일 유형이 b로 표현되는 디바이스를 말한다.

```
$ ls -l /dev/ram?
brw-rw---- 1 root disk 1, 0 Aug  6 14:10 /dev/ram0
brw-rw---- 1 root disk 1, 1 Aug  6 14:10 /dev/ram1
.......
```

끝으로 네트워크(network) 디바이스는 네트워크 통신을 통해 네트워크 패킷을 송수신할 수 있는 디바이스를 말하며, 파일시스템에 디바이스 파일로 존재하지 않는다. 네트워크 디바이스로는 Ethernet, PPP, Slip, ATM, ISDN NIC 등이 있다.

문자 디바이스나 블록 디바이스를 제어하기 위해 필요에 따라 커널에 동적 혹은 정적으로 링크하여 사용하는 디바이스 드라이버는 커널 모듈로써 구현된다.

13.3.2 커널 모듈

커널 모듈은 디바이스를 구동하기 위한 구동 프로그램으로 디바이스 드라이버라고 칭하기도 한다. 이들 커널 모듈은 물리적인 디바이스와 가상 파일시스템간에 데이터전송 인터페이스를 제공한다. 또한 커널에 정적 혹은 동적으로 링크되어 사용된다.

커널 모듈 소스는 해당 디바이스의 동작 특성에 따른 고유한 기능을 구현한 함수들로 구성된다. 디바이스가 xxx라고 가정할 때, 해당 디바이스를 제어하기 위한 커널 모듈 소스의 기본 골격은 다음과 같다.

```
//=====
// xxx device driver module
//=====

#include <linux/kernel.h>           //1. Header Files
#include <linux/module.h>
#include <linux/init.h>

....

#define MOD_MAJOR    200          // major no.
#define MOD_NAME     "xxx"        // module name
```

```

int xxx_open(...){...}                                //2. Functions
int xxx_release(...){...}
ssize_t xxx_write(...){...}
ssize_t xxx_read(...){...}
....
```

```

static struct file_operations xxx_fops = {           //3. File Operations
    .owner = THIS_MODULE,
    .open = xxx_open,                                // 멤버 및 초기화 병행
    .release = xxx_release,
    .read = xxx_read,
    .write = xxx_write,
    .llseek = NULL
};
```

```

static int xxx_init() {...}                         //4. 모듈링크시 초기화 수행
static void xxx_exit() {...}                        //5. 모듈제거시 수행
```

```

module_init(xxx_init);                            // 매크로, insmod시 수행
module_exit(xxx_exit);                           // 매크로, rmmod시 수행
```

file_operations 구조체는 함수들의 포인터 구조체이며, 커널 소스의 흄 디렉터리 하부 ./include/linux/fs.h 에서 다음과 같이 정의하고 있다.

```

static const struct file_operations _fops = {
    .owner = THIS_MODULE,
    .open = _fops ## _open,
    .release = simple_attr_release,
    .read = simple_attr_read,
    .write = simple_attr_write,
    .llseek = generic_file_llseek,
}
```

라즈베리파이 커널 소스에서는 6개 필드를 제공하며 이 구조체의 각 필드에는 관련함수를 정의하는 경우 그 함수를 가리킬 수 있도록 초기화해야 하며, 정의하지 않는 경우 NULL로 설정한다. 관련 함수들의 정의는 이 구조체보다 앞서 정의되어야 한다. .owner 필드는 이 구조체의 소유자를 지정하는 필드로, 소유자정보를 담고 있는 struct module 구조체를 포인팅한다. 일반적으로 module.h 헤더파일에

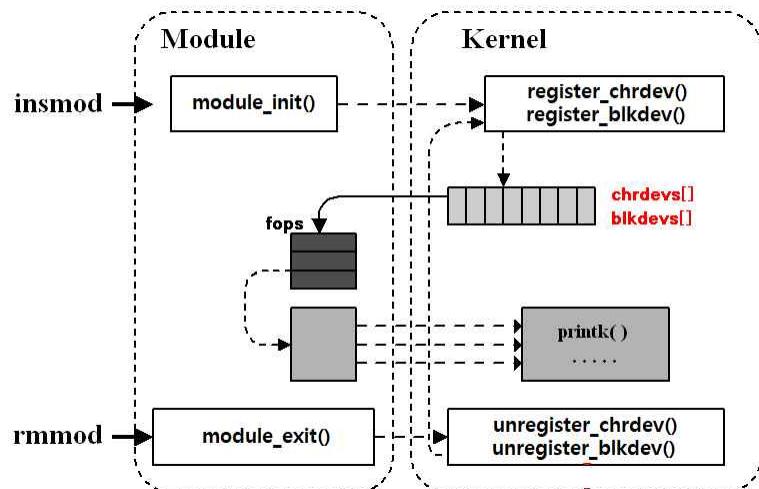
정의된 매크로인 THIS_MODULE로 초기화한다. .llseek 필드는 이 모듈의 파일포인터 위치를 강제 이동시키는 함수를 설정하는 필드이다. 이기타의 함수들에 대해서는 나중에 보다 세세히 살펴보도록 한다.

module_init()는 모듈 자신을 커널에 등록하여 링크시키는 xxx_init() 함수를 호출하는 매크로 함수로, 해당 커널 모듈을 커널에 링크시키는 insmod 명령에 의해 실행된다. module_exit()는 커널에 등록된 모듈 자신의 등록을 해지하여 링크를 해제하는 xxx_exit() 함수를 호출하는 매크로 함수로, 해당 커널 모듈을 커널에서 제거하는 rmmod 명령에 의해 실행된다.

이외에 디바이스의 주 번호, 디바이스 이름 등의 정보를 포함한다.

13.3.3 커널 모듈관련 명령

커널이 커널에 링크되는 과정의 개념도는 다음 그림과 같다.



module_init() 매크로는 xxx_init()를 실행하고, xxx_init()에서는 디바이스 유형에 따라 register_chrdev(), 혹은 register_blkdev() 시스템 함수를 호출한다. 이를 함수는 디바이스 유형에 따라 커널내의 링크를 위한 자료구조인 chrdevs[]와

blkdevs[]에 모듈의 주번호와 부번호를 인덱스로 한 위치에 모듈 이름 및 모듈의 파일구조체의 주소를 등록함으로써 커널에 모듈을 링크한다.

module_exit() 매크로는 xxx_exit()를 실행하고, xxx_exit()에서는 디바이스 유형에 따라 unregister_chrdev(), 혹은 unregister_blkdev() 시스템 함수를 호출하여 chrdevs[] 혹은 blkdevs[]에 등록된 모듈 정보를 제거함으로써 모듈을 커널로부터 링크 해제한다.

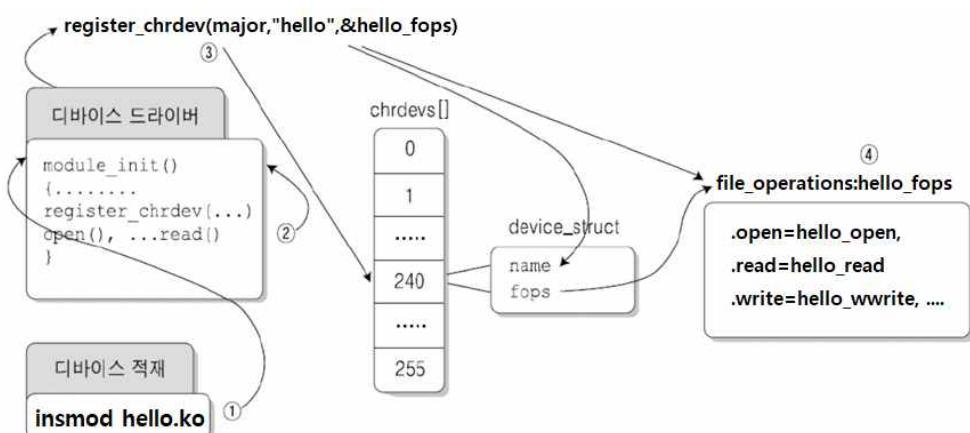
커널 모듈을 링크하거나 링크 해제하는 명령을 포함하여 커널 모듈과 관련된 명령에 대해 살펴보면 다음과 같다. 커널 모듈의 파일명은 hello.ko라고 가정한다.

insmod

insmod 명령은 커널 모듈을 커널에 링크하는 명령이다. 커널 모듈파일의 확장자까지 포함한 풀 네임을 사용한다.

```
$ sudo insmod hello.ko
```

이 명령이 실행되면 커널 모듈 소스에서 module_init() 함수가 호출되고 디바이스 파일의 주번호를 활용하여 모듈이름 및 file_operations 구조체를 등록함으로써 커널에 모듈을 링크시킨다. 아래 그림은 모듈이름이 hello이고, 주번호가 240인 경우이다.



rmmod

rmmod 명령은 커널 모듈을 커널로부터 링크 해제하는 명령이다. 이 명령의 경우 모듈 파일명을 확장자없이 커널 모듈이름만 쓰거나 풀네임을 쓸 수 있다.

```
$ sudo rmmod hello 혹은 hello.ko
```

이 명령이 실행되면 커널 모듈 소스에서 module_exit() 함수가 호출되고 커널 모듈을 커널로부터 등록 해제한다.

modinfo

modinfo 명령은 커널 모듈이 어느 커널 버전으로 컴파일되었는지의 정보를 알려주며, 모듈이 정상적으로 커널에 링크되려면 해당 커널 버전과 일치하여야 한다.

```
$ sudo modinfo hello.ko
```

lsmod

lsmod 명령은 현재 커널에 링크된 커널 모듈의 목록을 보여주는 명령으로 통산 grep 명령과 함께 사용된다.

```
$ lsmod | grep hello
```

또한 파일시스템에 커널 모듈에 대한 디바이스 파일을 생성하는 명령으로 mknod 가 있다.

mknod

mknod 명령은 커널 모듈에 대한 디바이스 파일 명을 생성하는 명령이다. 디바이스 파일은 /dev/ 디렉터리에 위치해야한다. 이 명령은 디바이스 유형, 주번호, 부번호 등의 정보가 필요하다. 디바이스 유형을 표현할 때 문자 디바이스는 c를, 블록디바이스는 b를 사용하며 주번호와 부번호는 1바이트 크기로 주어진다. 부번호는 주번호가 동일한 디바이스가 여럿일 경우 그 식별을 위한 번호로 사용된다.

```
$ sudo mknod /dev/hello c major# minor#
```

참고로 dmesg 명령으로 최근에 커널에서 출력한 메시지 즉, printk()를 통해 출력

한 메시지를 확인할 수 있다.

```
$ dmesg
```

13.3.4 실습예제

커널 모듈을 구현하고 테스트하기 위한 예제들에 대해 살펴본다. 우선 커널 모듈을 컴파일할 때 실행할 커널 버전과 일치해야 하므로, 다음의 명령을 사용하여 커널 모듈 버전을 확인한다.

```
$ uname -r  
4.9.79-v7
```

[실습1] hello 모듈 테스트

가상의 디바이스 hello를 가정하여 최소한의 모듈 구성 요소를 갖추어 모듈 컴파일, 모듈 등록 및 제거과정을 테스트해 본다.

Makefile

아래의 밑줄 부분은 자신의 커널소스 버전과 일치시킨다.
`/lib/modules/4.9.79-v7/build` 디렉터리는 커널 소스가 설치된 `/usr/src/linux/` 디렉터리다. 또한 `obj-m`에 설정하는 모듈의 목적코드명도 유의한다.

```
$ nano Makefile
=====
# Makefile for device module
=====
CC      := /usr/bin/gcc
KDIR   := /lib/modules/4.9.79-v7/build

obj-m  := hello.o

build:
```

```
make -C $(KDIR) SUBDIRS=`pwd` modules  
clean:  
rm -rf *.o *.ko *.mod.c *.cmd *.order *.symvers .tmp*
```

hello 모듈

```
$ nano hello.c  
=====  
// hello.c  
//      Hello MODULE test  
=====  
#include <linux/module.h>  
#include <linux/kernel.h>  
#include <linux/init.h>  
  
MODULE_LICENSE("GPL");  
  
int hello_init(void) {  
    printk("HELLO MODULE is loaded.\n");  
    return 0;  
}  
  
void hello_exit(void) {  
    printk("HELLO MODULE is unloaded.\n");  
}  
  
module_init(hello_init);  
module_exit(hello_exit);
```

모듈 컴파일

커널 모듈의 컴파일은 다음과 같이 슈퍼유저 권한으로 진행한다. 컴파일 과정에서 커널 소스 디렉터리와 모듈 라이브러리 디렉터리를 참조하는 것을 관찰할 수 있다.

```
$ sudo make  
make -C /lib/modules/4.9.79-v7/build SUBDIRS=`pwd` modules
```

```
make[1]: Entering directory '/usr/src/linux-rpi-4.9.y'
  CC [M]  /home/pi/IFC415/13_KERNEL/moduleTest/hello/hello.o
Building modules, stage 2.
MODPOST 1 modules
  CC      /home/pi/IFC415/13_KERNEL/moduleTest/hello/hello.mod.o
  LD [M]  /home/pi/IFC415/13_KERNEL/moduleTest/hello/hello.ko
make[1]: Leaving directory '/usr/src/linux-rpi-4.9.y'
```

정상적으로 컴파일이 된 후, 다음과 같이 생성된 hello.ko 파일이 hello 커널 모듈이다.

```
$ ls
Makefile      hello.c  hello.mod.c  hello.o      modules.order
Module.symvers hello.ko  hello.mod.o
```

모듈 등록 및 제거

다음 명령으로 컴파일된 커널 모듈이 실행가능한 커널버전은 4.9.79-v7임을 확인할 수 있다.

```
$ modinfo hello.ko
filename:      /home/pi/IFC415/13_KERNEL/moduleTest/hello/hello.ko
license:       GPL
srcversion:    502289B12C7BD57EC69249A
depends:
vermagic:     4.9.79-v7 SMP mod_unload modversions ARMv7 p2v8
```

다음의 명령을 사용하여 현재 등록된 모듈 내역을 확인하거나 특정 모듈의 등록여부를 확인한다.

```
$ lsmod
$ lsmod | grep hello
```

다음 명령으로 hello 모듈을 등록하며, 모듈 이름은 확장자까지 포함한 전체 이름을 사용한다. 참고로 Invalid module format Error가 발생하게 되는 경우는 커널 버전과 모듈 버전의 불일치로 인한 것이므로 모듈과 동일한 버전의 커널 이미지로

재부팅하여 다시 시도한다. dmesg 명령으로 최근에 커널에서 출력한 메시지 즉, printk()를 통해 출력한 메시지를 확인할 수 있다.

```
$ sudo insmod hello.ko
$ dmesg
$ lsmod | grep hello
hello           1084  0
```

다음의 명령들을 사용하여 모듈의 링크를 해제하고 해제여부를 확인할 수 있다. 모듈 제거시 확장자 포함한 모듈이름을 사용하거나 확장자 제외한 모듈이름만 사용해도 된다.

```
$ sudo rmmod hello
$ dmesg
$ lsmod | grep hello
```

13.3.3 커널 모듈에서 디바이스 제어

앞서의 hello 모듈을 활용하여 간단히 디바이스를 제어하는 과정에 대해 살펴본다. 디바이스를 제어하려면 커널의 가상주소공간에서 해당 디바이스에 대한 가상 주소를 확보해야한다. 즉, 커널의 가상주소공간에서 입출력 디바이스의 물리주소를 활용하여 가상주소를 얻을 때 linux/io.h 혹은 asm/io.h 라이브러리를 사용한다. 이 라이브러리에서는 ioremap() 함수과 iounmap() 함수 등을 제공한다.

ioremap() 함수

ioremap() 함수는 디바이스의 물리주소를 사용하여 커널의 가상 메모리주소 공간에서 해당 디바이스의 가상주소를 얻는 함수로, 다음의 함수들이 정의되어 있다. offset은 디바이스의 물리주소이다.

```
void *ioremap(unsigned long offset, unsigned long size);
void *ioremap_nocache(unsigned long offset, unsigned long size);
```

iounmap() 함수

iounmap() 함수는 ioremap() 함수에 의해 디바이스의 가상메모리 주소를 위해 할당된 공간을 해제하는 함수이다. addr을 가상주소를 위한 포인터를 준다.

```
void *iounmap(void *addr);
```

[실습2] ioremap()에 의한 디바이스 제어

앞의 hello 모듈 소스를 바탕으로 LED 디바이스를 5회 점멸하는 모듈을 구현한다. LED 회로는 BCM_GPIO #18에 연결하며, 소스는 ./helloLed 디렉터리를 참조한다.

```
$ nano helloLed.c
//=====
// helloLed.c
//      hello + LED module
//=====

#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/delay.h>
#include <linux/io.h>           // ioremap(), iounmap()

#define GPIO_BASE      0x3F200000    // for BCM2836, BCM2837
#define BLOCK_SIZE     4096

// address offset of registers for BCM_GPIO #18
#define GPIO_LED       18           // BCD_GPIO #18
#define GPFSEL1        (0x04/4)     // int *
#define GPSET0         (0x1C/4)
#define GPCLR0         (0x28/4)

volatile unsigned int * gpio_addr;

// LED blinking
static void blinking(void) {
    int i;

    for(i=0; i<5 ; i++){
        // 1C h : 28 dec = 4 * 7
```

```
        *(gpio_addr+GPSET0) |= 1 <<(GPIO_LED); // ON
        mdelay(1000);

        // 28 h : 40 dec = 4 * 10
        *(gpio_addr+GPCLR0) |= 1 <<(GPIO_LED); // OFF
        mdelay(1000);
    }

static int hello_init(void) {
    printk("insmod : Hello, GPIO LED Module Test.....\n");

    gpio_addr = ioremap(GPIO_BASE, BLOCK_SIZE);

    // GPFSEL1 = gpio_addr + 1
    *(gpio_addr+GPFSEL1) &= ~(6 << (((GPIO_LED)%10)*3));

    blinking();           // LED ON/OFF

    return 0;
}

static void hello_exit(void) {
    iounmap(gpio_addr);
    printk("rmmod : Hello, GPIO LED Module Test.....\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

앞서의 Makefile을 현재의 커널버전 정보를 얻어 컴파일하도록 다음과 같이 작성 한다.

```
$ nano Makefile
=====
# Makefile for device module
=====
KERVER := $(shell uname -r)
PWD := $(shell pwd)
```

```

CC      := /usr/bin/gcc
KDIR   := /lib/modules/$(KERVER)/build

obj-m := helloLed.o

build:
    make -C $(KDIR) SUBDIRS=$(PWD) modules

clean:
    rm -rf *.o *.ko *.mod.c *.cmd *.order *.symvers .tmp*

```

다음 명령으로 이 모듈 소스를 컴파일한다.

\$ make

다음과 같이 커널모듈을 커널에 링크하는 시점에서 LED가 5회 점멸하는 것을 확인할 수 있다.

\$ sudo insmod hello.ko

결과 확인이 완료되면 다음 명령을 사용하여 모듈을 제거한다.

\$ sudo rmmod hello.ko

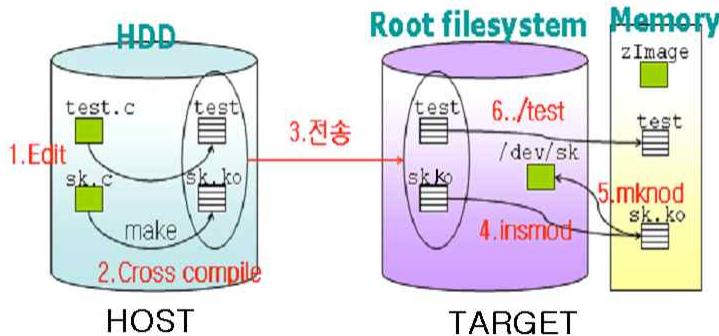
13.4 커널 모듈의 동적 링크

앞서의 커널 모듈 테스트에서의 방법처럼 커널 이미지와 독립적으로 커널 모듈을 구현하고 필요에 따라 동적으로 링크하여 사용하는 예를 살펴본다.

13.4.1 커널 모듈에 의한 디바이스 제어

커널 모듈을 사용하여 특정 디바이스를 제어하기 위한 전체적인 흐름은 다음 그림과 같다. 커널 모듈의 이름은 sk이고, 이 모듈을 이용한 응용프로그램은 test인 상

황이다.



커널 모듈내 함수 구현

제어할 디바이스가 입력용 혹은 출력용에 따라 커널 모듈에서 관련 함수들을 정의해야한다. 이들 함수내에서 구현해야할 사항들에 대해서 살펴본다.

xxx_init() 함수는 커널 모듈을 커널에 링크시켜 등록하는 기능의 함수로, 함수 원형은 다음과 같다. 즉 insmod 명령이 실행될 때 수행되는 함수이다. 이 함수에서는 디바이스 유형에 따라 커널에 모듈을 등록하는 register_chrdev() 혹은 register_blkdev() 시스템 함수를 호출하여야 한다. 커널에 모듈이 등록될 때 처리할 기타의 작업들이 포함되어야 한다. 이러한 작업들에는 인터럽트 처리를 위한 초기화 등이 있을 수 있다.

```
int xxx_init(void)
```

xxx_exit() 함수는 커널에 링크된 모듈을 링크 해제하는 기능의 함수로, 함수 원형은 다음과 같다. 즉 rmmod 명령이 실행될 때 수행되는 함수이다. 이 함수에서는 디바이스 유형에 따라 커널로부터 모듈의 링크를 해제하는 unregister_chrdev() 혹은 unregister_blkdev() 시스템 함수를 호출하여야 한다. 기타 커널에 모듈이 등록될 때 처리하였던 작업들을 해제하는 작업들이 포함되어야 한다.

```
void xxx_exit(void)
```

`xxx_open()` 함수는 응용 프로그램에서 디바이스 파일을 개방하는 `open()` 함수 호출시 실행되는 함수다. 또한 `xxx_release()` 함수는 응용 프로그램에서 디바이스 파일을 닫는 `close()` 함수 호출시 실행되는 함수다. 이들 함수들의 원형은 다음과 같다.

```
static int xxx_open(struct inode *inode, struct file *filp)
static int xxx_release(struct inode *inode, struct file *filp)
```

`xxx_read()` 함수는 디바이스로부터 데이터를 읽어 들이는 기능의 함수로, 함수 원형은 다음과 같다. `filp`에 지정된 디바이스로부터 `count` 수만큼의 데이터를 `buf`로 읽어 들인다. 응용 프로그램에서 `read()` 함수 호출시 실행되는 함수다.

```
static ssize_t xxx_read(struct file *filp, char *buf, size_t count, loff_t *)
```

`xxx_write()` 함수는 디바이스로 데이터를 출력하는 기능의 함수로, 함수 원형은 다음과 같다. `inode`에 지정된 디바이스로 `gdata`의 데이터를 `length`만큼 출력한다. 응용 프로그램에서 `write()` 함수 호출시 실행되는 함수다.

```
static ssize_t xxx_write(struct file *inode, const char *gdata, size_t length, loff_t *off_what)
```

커널 모듈내 함수들과 응용프로그램에서의 호출 함수들의 관계와 호출 예를 정리하면 다음 표와 같다.

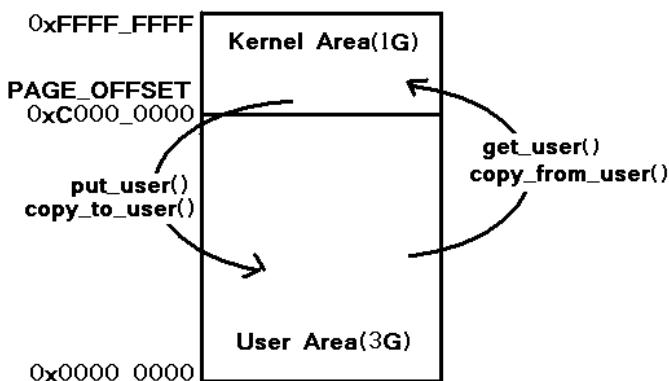
커널 모듈(xxx)	응용프로그램	응용프로그램에서 함수 호출 예
<code>xxx_open()</code>	<code>open()</code>	<code>fd=open("/dev/xxx", O_WRONLY);</code>
<code>xxx_release()</code>	<code>close()</code>	<code>close(fd);</code>
<code>xxx_read()</code>	<code>read()</code>	<code>read(fd, buf, len);</code>
<code>xxx_write()</code>	<code>write()</code>	<code>write(fd, &buf, len);</code>

커널 영역과 사용자 영역간 데이터 전송 함수

특정 디바이스의 커널 모듈은 커널 영역에서 실행되는 모듈이며, 사용자 응용 프로그램에서 디바이스로 데이터를 전송하려면 사용자 영역과 커널 영역간에 데이터전

송이 가능해야 한다.

하지만 다음 그림과 같이 리눅스는 커널과 커널 모듈이 실행되는 커널 영역과 사용자의 응용프로그램이 실행되는 사용자 영역을 구분하고 있으며, 이 가상 메모리 공간에서의 두 영역간 교차 접근을 제한하는 보호 정책을 채택하고 있다. BCM2835 데이터시트를 살펴보면 그 경계 주소는 0xC000_0000이다.



커널 영역에서 실행되는 입출력디바이스의 커널 모듈과 사용자 응용 프로그램간의 데이터 전송을 위한 시스템 함수를 제공하고 있다. 이들 함수들의 이름은 커널 관점에서 결정되었으며 `asm/uaccess.h` 파일로 제공된다.

사용자 영역으로부터 커널 영역으로 데이터를 전달하는 함수로 다음의 함수들이 제공되며, 이들 함수들은 일반적으로 커널 모듈의 `xxx_write()` 함수 내에서 호출한다. `get_user()` 함수는 기본 자료형의 데이터에 대해 사용자 영역으로부터 커널 영역으로 전달하는 함수이며, `copy_from_user()` 함수는 사용자 영역으로부터 커널 영역으로 다수의 데이터인 블록 데이터를 전달하는 함수이다.

```
get_user(void *x, const void *addr)
```

```
// 사용자 영역 *addr의 값을 커널 영역인 x로 sizeof(addr)만큼 복사
```

```
copy_from_user(void *to, void *from, unsigned long size)
```

```
// 사용자 영역의 주소 from으로부터 size만큼 커널 영역 to로 복사
```

커널 영역으로부터 사용자 영역으로 데이터를 전달하는 함수로 다음의 함수들이 제공되며, 이들 함수들은 일반적으로 커널 모듈의 xxx_read() 함수 내에서 호출한다. put_user() 함수는 기본 자료형 데이터를, copy_to_user() 함수는 블록 데이터를 커널 영역으로부터 사용자 영역으로 전달하는 함수이다.

```
put_user(void *x, const void *addr)
    // *x의 값을 user 영역인 addr로 sizeof(addr) 만큼 복사

copy_to_user(void *to, void *from, unsigned long size)
    // 커널영역의 주소 from으로부터 size만큼 사용자 영역 to로 복사
```

13.4.2 rawGPIO 제어에 의한 LED 제어

GPIO LED 디바이스를 제어하기 위한 디바이스 드라이버 모듈을 작성하고, 이를 이용하여 LED 디바이스 제어하는 것에 대해 살펴본다.

[실습3] rawGPIO 제어에 의한 LED 제어

GPIO LED 제어를 위한 디바이스 드라이버 모듈의 소스는 다음과 같다. 모듈의 메이저 번호는 201, 모듈이름은 gpioled로 하고, LED 회로는 BCM_GPIO #18에 연결한다. 소스는 ./gpioLed 디렉터리에 제공된다.

```
$ nano gpioled.c
//=====
// gpioled.c
//      LED Device Driver : Direct Access
//=====

#include <asm/uaccess.h>          // copy_to_user(), copy_from_user()
#include <linux/ioport.h>
#include <linux/module.h>
#include <linux/fs.h>              // open(), close(), read(), write()
#include <linux/cdev.h>
#include <linux/io.h>              // ioremap(), iounmap()

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

```

#define GPIO_RANGE      1024
// address offset of registers for BCM_GPIO #18
#define GPFSEL0         (0x00/4)        // int *
#define GPSET0          (0x1C/4)
#define GPCLR0          (0x28/4)

// #define MOD_MAJOR     0 // automatic allocation
#define MOD_MAJOR       201
#define MOD_NAME        "gpioled"

#define GPIO_LED        18           // BCM_GPIO #18

volatile unsigned int * gpio_addr;

int gpioled_open(struct inode *minode, struct file *mfile) {
    printk("Kernel Module Open(): %s\n", MOD_NAME);
    return 0;
}

int gpioled_release(struct inode *minode, struct file *mfile) {
    printk("Kernel Module close(): %s\n", MOD_NAME);
    return 0;
}

ssize_t gpioled_write(struct file *inode, const char *gdata, size_t length,
loff_t *off_what) {
    unsigned char c;

    get_user(c, gdata);      // if c==1 then ON, else OFF

    if (c == 1)
        *(gpio_addr+GPSET0) |= 1 << (GPIO_LED);      // ON
    else
        *(gpio_addr+GPCLR0) |= 1 << (GPIO_LED);      // OFF

    return length;
}

static struct file_operations gpioled_fops = {
    .write       = gpioled_write,
    .open        = gpioled_open,
    .release     = gpioled_release,
}

```

```

};

int gpioled_init(void) {
    int result;

    result = register_chrdev(MOD_MAJOR, MOD_NAME, &gpioled_fops);
    if(result < 0) {
        printk("Can't get any major\n");
        return result;
    }

    printk("Init Module: Major number %d\n", MOD_MAJOR);
    printk("Init Module: Major number %d\n", result);

    gpio_addr = ioremap(GPIO_BASE, GPIO_RANGE);

    // GPFSELn, #18, out
    *(gpio_addr + (GPIO_LED/10)) &= ~(6 << (((GPIO_LED)%10)*3));

    return 0;
}

void gpioled_exit(void) {
    unregister_chrdev(MOD_MAJOR, MOD_NAME);
    printk("%s DRIVER CLEANUP\n", MOD_NAME);
}

module_init(gpioled_init);
module_exit(gpioled_exit);

```

위의 gpioled 커널 모듈을 활용한 LED 응용 프로그램의 소스는 다음과 같으며, 명령행에서 on 혹은 off 문자열을 인자로 전달받아 LED를 제어하도록 구현한다.

```

$ nano gpioled_test.c
//=====
// gpioled_test.c
//
//=====
#include <stdio.h>

```

```

#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(int argc, char **argv) {
    int fd;
    char buf;

    if(argc <= 1) {
        printf("Usage : ./gpioled_test on or off\n");
        return -1;
    }

    fd = open("/dev/gpioled", O_WRONLY);
    if(fd == -1) {
        printf( "Device Open ERROR!\n");
        exit(-1);
    }

    if(argv[1][0] == 'o' && argv[1][1] == 'n')
        buf = 1;
    else if(argv[1][0] == 'o' && argv[1][1] == 'f' && argv[1][2] == 'f')
        buf = 0;

    write(fd, &buf, 1);           // LED control

    close(fd);

    return 0;
}

```

컴파일에 사용되는 Makefile을 커널 모듈뿐만 아니라 이를 제어하기 위한 응용 프로그램도 컴파일하도록 다음과 같이 작성한다.

```

$ nano Makefile
=====
# Makefile for device module

```

```

=====
KERVER := $(shell uname -r)
PWD    := $(shell pwd)

CC      := /usr/bin/gcc
KDIR   := /lib/modules/$(KERVER)/build

obj-m := gpioled.o

TEST_TARGET = gpioled_test
TEST_OBJS = gpioled_test.o
TEST_SRCS = gpioled_test.c

build: $(TEST_TARGET)
       make -C $(KDIR) SUBDIRS=$(PWD) modules

$(TEST_TARGET) : $(TEST_OBJS)
               $(CC) -o $@ $(TEST_OBJS)

clean:
       rm -rf *.o *.ko *.mod.c *.cmd *.order *.symvers .tmp*
       rm -f $(TEST_TARGET)

```

다음의 명령으로 커널 모듈과 응용 프로그램을 함께 컴파일한다. LED 디바이스를 제어하기 위한 커널 모듈 gpioled.ko와 응용 프로그램인 gpioled_test 파일이 생성된다.

```

$ sudo make
$ ls
Makefile      gpioled.ko      gpioled.o      gpioled_test.c
Module.symvers gpioled.mod.c  gpioled_test.o
gpioled.c     gpioled.mod.o  gpioled_test  modules.order

```

재 컴파일 등의 이유로 생성된 목적코드 등을 제거하기 위해서는 다음 명령을 사용하여 커널 모듈과 응용 프로그램 컴파일의 결과물을 제거할 수 있다.

```
$ sudo make clean
```

```
$ ls  
Makefile gpioled.c gpioled_test.c
```

커널 모듈과 응용 프로그램의 컴파일이 완료되었으면, 다음 명령에 따라 /dev/디렉터리에 디바이스 파일 즉, 노드를 생성하고 접근 속성을 변경한다.

```
$ sudo mknod /dev/gpioled c 201 0  
$ sudo chmod 777 /dev/gpioled  
$ ls -l /dev/gpio*  
crw-rw---- 1 root gpio 254, 0 Aug 6 14:10 /dev/gpiochip0  
crwxrwxrwx 1 root root 201, 0 Aug 7 12:38 /dev/gpioled  
crw-rw---- 1 root gpio 246, 0 Aug 6 14:10 /dev/gpiomem
```

다음과 같이 커널 모듈을 등록하고, 응용 프로그램을 실행하여 그 결과를 관찰한다.

```
$ sudo insmod gpioled.ko  
$ lsmod | grep gpioled  
gpioled           2052  0
```

이제 해당 모듈의 디바이스 파일을 개방하여 제어하는 응용 프로그램을 실행하여 결과를 관찰한다. 응용 프로그램은 명령행에서 문자열을 전달받도록 구현되어 있으므로, 명령행에서 on 혹은 off 문자열을 전달하여 실행한다.

```
$ ./gpioled_test on  
$ ./gpioled_test off
```

응용 프로그램의 실행 결과를 확인하였으며, 다음의 명령을 통하여 모듈 제거, 노드 삭제 등의 과정을 진행하여 커널 모듈의 링크를 해제한다.

```
$ sudo rmmod gpioled  
$ lsmod | grep gpioled  
  
$ sudo rm /dev/gpioled
```

```
$ ls /dev/gpio*
```

13.4.3 gpio.h 라이브러리를 이용한 LED 제어

linux/gpio.h 라이브러리의 함수들을 활용하여 GPIO 핀을 제어할 수 있다. 이 라이브러리의 헤더파일은 해당 커널 소스의 홈 디렉터리 하부에 있는 ./include/linux/gpio.h 이다.

linux/gpio.h 라이브러리 함수

이 라이브러리가 제공하는 함수 중 주요한 함수들에 대해 살펴본다.

다음의 gpio_request() 함수는 GPIO 핀으로 요청하는 함수이다. gpio는 GPIO 핀 번호이며, label은 핀에 대한 이름을 나타낸다. 정상적인 경우 반환값은 0이며, 오류 발생 시 음수로 에러 번호를 반환한다.

```
int gpio_request(unsigned int gpio, const char *label);
```

위의 함수를 통해 요청된 GPIO 핀은 다음 gpio_free() 함수를 통해 할당을 해제할 수 있다.

```
void gpio_free(unsigned int gpio);
```

또한 GPIO 핀의 입출력 방향을 지정하는 함수로 다음의 함수들이 있다. gpio_direction_input() 함수는 해당 핀을 입력용으로 설정하며, gpio_direction_output() 함수는 출력용으로 설정한다. value는 초기 출력 값으로 0 혹은 1로 초기화되어야 한다. 정상적으로 설정되면 0을, 아니면 음수로 에러 번호를 반환한다.

```
int gpio_direction_input(unsigned int gpio);
int gpio_direction_output(unsigned int gpio, int value);
```

다음의 함수는 GPIO 핀과의 데이터 입출력을 위한 함수이다.

```
int gpio_get_value(unsigned int gpio);
void gpio_set_value(unsigned int gpio, int value);
```

다음 함수는 인터럽트 처리를 위해 GPIO 핀의 IRQ 번호를 반환하는 함수이다. 해당 GPIO 핀은 gpio_request()로 꼭 얻어야 하고, 해당 GPIO 핀은 일단 입력 모드로 설정된다. 반환된 irq 번호를 사용하여 request_irq() 함수를 통해 인터럽트 핸들러 함수를 등록할 수 있다.

```
int gpio_to_irq(unsigned int gpio);
```

[실습4] gpio.h 함수를 활용한 LED 제어

linux/gpio.h 라이브러리 함수를 이용하여 구현한 디바이스 드라이버 모듈을 통하여 LED 디바이스를 제어하는 프로그램을 구현한다. 소스는 ./gpioLed2 디렉터리로 제공된다.

```
$ nano gpioled.c
//=====
// gpioled.c
//      LED Device Driver
//      using /linux/gpio.h
//=====

#include <asm/uaccess.h>      // copy_to_user(), copy_from_user()
#include <linux/ioport.h>
#include <linux/module.h>
#include <linux/fs.h>          // open(), close(), read(), write()
#include <linux/cdev.h>
#include <linux/io.h>           // ioremap(), iounmap()
#include <linux/gpio.h>         // *)!!!!

#define MOD_MAJOR      0      // automatic allocation
#define MOD_NAME       "gpioled"
#define GPIO_LED       18     // BCM_GPIO #18

int gpioled_open(struct inode *minode, struct file *mfile) {
```

```

        printk("Kernel Module Open(): %s\n", MOD_NAME);
        return 0;
    }

int gpioled_release(struct inode *minode, struct file *mfile) {
    printk("Kernel Module close(): %s\n", MOD_NAME);
    return 0;
}

ssize_t gpioled_write(struct file *inode, const char *gdata, size_t length,
loff_t *off_what) {
    unsigned char c;

    get_user(c, gdata);

    gpio_set_value(GPIO_LED, ((c == 0) ? 0 : 1));

    return length;
}

static struct file_operations gpioled_fops = {
    .write      = gpioled_write,
    .open       = gpioled_open,
    .release    = gpioled_release,
};

int gpioled_init(void) {
    int result;

    result = register_chrdev(MOD_MAJOR, MOD_NAME, &gpioled_fops);
    if(result < 0) {
        printk("Can't get any major\n");
        return result;
    }

    printk("Init Module: Major number %d\n", MOD_MAJOR);

    gpio_request(GPIO_LED, "LED");
    gpio_direction_output(GPIO_LED, 0);

    return 0;
}

```

```
void gpioled_exit(void) {
    unregister_chrdev(MOD_MAJOR, MOD_NAME);
    printk("%s DRIVER CLEANUP\n", MOD_NAME);

    gpio_free(GPIO_LED);
}

module_init(gpioled_init);
module_exit(gpioled_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("melee");
MODULE_DESCRIPTION("Raspberry Pi 3 GPIO LED Device Driver Module");
```

테스트를 위한 응용 프로그램은 앞의 예제와 동일한 소스를 사용한다.

13.4.4 GPIO BTN 제어

gpio.h 함수를 사용하되, 버튼 스위치의 입력에 대해 인터럽트 방식으로 처리하는 방법에 대해 살펴본다.

인터럽트 처리

인터럽트 방식으로 처리하기 위해서는 ./include/linux/interrupt.h 헤더파일을 소스에 포함시킨다. 이와 관련된 함수들에 대해 살펴본다.

인터럽트 핸들러 함수의 원형은 다음과 같으며, 등록된 디바이스로부터 인터럽트가 요청되었을 때 수행되는 함수로, 인터럽트 요청에 대한 서비스를 포함하도록 작성한다.

```
static irqreturn_t btn_interrupt(int irq, void *dev_id) {
    .....
}
```

`request_irq()` 함수는 인터럽트 핸들러 함수를 등록하는 함수로 아래와 같은 함수 원형으로 제공된다. 이 함수에서 `irq`는 GPIO 핀에 대한 인터럽트 번호를 설정해야 하며, 이는 `gpio_to_irq()` 함수에 의해 확보할 수 있다. `handler`에는 인터럽트 핸들러 함수를 지정하며, `flags`는 인터럽트 공유 등의 처리 속성을 설정한다. `name`에는 인터럽트 소유자를 지정하며, `dev`는 일반적으로 NULL로 설정한다. 정상적으로 수행되면 0을 리턴하며, 그렇지 않은 경우 여러 상황에 맞는 음수 값을 반환한다.

```
static inline int __must_check
request_irq(unsigned int irq, irq_handler_t handler, unsigned long flags,
           const char *name, void *dev)
```

`free_irq()` 함수는 인터럽트 핸들러 함수의 등록을 해제하는 함수로 함수 원형은 다음과 같다.

```
void free_irq(unsigned int irq, void *dev);
```

기타 인터럽트 요청을 활성화 혹은 불활성화하는 함수들로, `enable_irq()` 함수, `disable_irq()` 함수가 제공되고 있다.

인터럽트 처리의 주요 코드

인터럽트 처리를 위한 커널 모듈의 주요 코드부분을 발췌하면 다음과 같다.

```
static int btn_irq;

// interrupt handler
static irqreturn_t btn_interrupt(int irq, void *dev_id) {
    .....
    // service
    return 0;
}

int gpiobtn_init(void) {
    int result, irq;
    .....
    gpio_request(GPIO_BTN, "SWITCH");
```

```
gpio_direction_input(GPIO_BTN);

btn_irq = gpio_to_irq(GPIO_BTN);
irq = request_irq(btn_irq, &btn_interrupt, IRQF_TRIGGER_FALLING,
                   "SWITCH", NULL);
.....
}

void gpiobtn_exit(void){
.....
free_irq(btn_irq, NULL);
.....
}
```

[실습5] 인터럽트에 의한 BTN 제어

BTN 디바이스로부터의 입력에 대해 인터럽트 방식으로 처리하는 디바이스 드라이버를 구현하고 이를 테스트하는 프로그램을 구현한다. 소스는 ./gpioBtn2 디렉터리를 참조한다.

```
$ nano gpiobtn.c
//=====
// gpiobtn.c
//      GPIO Input Device Driver
//      using Interrupt method & gpio.h
//=====

#include <linux/interrupt.h>          // *)!!
#include <linux/cdev.h>
#include <linux/module.h>
#include <linux/io.h>
#include <linux/gpio.h>                // *)!!
#include <asm/uaccess.h>

#define MOD_MAJOR    202
#define MOD_NAME     "gpiobtn"

#define GPIO_BTN      17      // BCM_GPIO #17
```

```
static int btn_irq;
static char msg[40] = {0};

// interrupt handler
static irqreturn_t btn_interrupt(int irq, void *dev_id) {
    char temp[40] = "GPIO Switch was Pushed!";

    strcpy(msg, temp);

    return 0;
}

static ssize_t gpiobtn_read(struct file *filp, char *buf, size_t count, loff_t
*l) {
    int result;

    result = copy_to_user(buf, &msg, sizeof(msg));

    memset(msg, 0, 40);

    return result;
}

static int gpiobtn_open(struct inode *inode, struct file *filp) {
    printk("Kernel Module Open(): %s\n", MOD_NAME);
    return 0;
}

static int gpiobtn_release(struct inode *inode, struct file *filp) {
    printk("Kernel Module close(): %s\n", MOD_NAME);

    disable_irq(GPIO_BTN);

    return 0;
}

static struct file_operations gpiobtn_fops = {
    .read        = gpiobtn_read,
    .open        = gpiobtn_open,
    .release     = gpiobtn_release,
};
```

```
int gpiobtn_init(void) {
    int result, irq;

    result = register_chrdev(MOD_MAJOR, "GPIO_INTERRUPT",
                            &gpiobtn_fops);
    if(result < 0) {
        printk(KERN_WARNING"Can't get major %d\n",
               MOD_MAJOR);
    }

    gpio_request(GPIO_BTN, "SWITCH");
    gpio_direction_input(GPIO_BTN);

    btn_irq = gpio_to_irq(GPIO_BTN);
    irq = request_irq(btn_irq, &btn_interrupt, IRQF_TRIGGER_FALLING,
                      "SWITCH", NULL);
    if(irq < 0)
        printk(KERN_ERR "%s: Request for IRQ %d failed\n",
               __FUNCTION__, GPIO_BTN);

    printk("init module, GPIO major number : %d\n", MOD_MAJOR);

    return 0;
}

void gpiobtn_exit(void){
    unregister_chrdev(MOD_MAJOR, "GPIO_INTERRUPT");

    free_irq(btn_irq, NULL);
    gpio_free(GPIO_BTN);

    printk("%s DRIVER CLEANUP\n", MOD_NAME);

    return;
}

module_init(gpiobtn_init);
module_exit(gpiobtn_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("melee");
```

```
MODULE_DESCRIPTION("Raspberry Pi 3 GPIO Switch Device Driver Module");
```

```
$ nano gpiobtn_test.c
//=====================================================================
// gpiobtn_test.c
//
//=====================================================================
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>

int main(void) {
    int fd;
    char buf[40];

    fd = open("/dev/gpiobtn", O_RDWR);
    if(fd < 0) {
        printf( "Device Open ERROR!\n");
        exit(1);
    }

    printf("Please push the GPIO button!\n");

    read(fd, buf, 40);           // read
    printf("%s\n", buf);

    close(fd);

    return 0;
}
```

13.5 커널 모듈을 커널에 포함하기

커널 모듈을 커널 이미지에 포함하여 디바이스를 제어하는 방법에 대해서 살펴본다. 즉, 커널 소스의 일부로 디바이스 드라이버 모듈을 포함시켜 커널 환경 설정한 후 커널 컴파일을 하여 생성된 커널 이미지로 부팅하여 활용하는 방법이다.

13.5.1 커널 모듈을 커널 소스에 등록

커널 모듈이 문자 디바이스의 경우는 /usr/src/linux/drivers/char/로, 블록 디바이스의 경우는 /usr/src/linux/drivers/block/ 디렉터리로 커널 소스 파일을 위치시킨다. 또한 해당 디렉터리 내의 Kconfig에 메뉴항목을 추가해 주고, 커널 컴파일할 때 커널 모듈도 컴파일될 수 있도록 관련 Makefile을 수정해야 한다. 여기서는 문자 디바이스의 커널 모듈을 기준으로 진행하며, LED 제어용 커널 소스 gpioled.c를 활용한다.

Kconfig의 메뉴 항목 포맷

우선 Kconfig 파일에 등록하는 메뉴 항목의 포맷에 대해 살펴본다. Kconfig 파일에서 메뉴 항목의 포맷은 다음과 같이 구성된다. 다음은 메뉴 심볼 명이 DEVMEM인 경우로 memuconfig 화면에서 하나의 메뉴항목으로 표현된다.

```
config DEVMEM
    bool "/dev/mem virtual device support"
    default y
    help
        Say Y here if you want to support the /dev/mem device.
        The /dev/mem device is used to access areas of physical
        memory.
        When in doubt, say "Y".
```

메뉴 속성항목 중 다음과 같은 라인은 메뉴 유형을 설정하는 항목이다. 메뉴유형 뒤의 문자열은 하나의 메뉴 항목으로 표시되는 프롬프트 문자열이다.

```
bool "/dev/mem virtual device support"
```

메뉴 유형에는 bool, tristate, string, hex, int가 있다. 위와 같이 bool 유형은 두 가지 상태 y/n를 선택할 수 있는 메뉴 유형이며, tristate 유형은 세 가지 상태 y/n/m(module)를 선택할 수 있는 메뉴 유형이다. 이외의 string, hex, int 메뉴 유형은 데이터를 입력받을 수 있는 유형으로, 각각 문자열, 16진수, 10진수 데이터를 입력받아 설정할 수 있다.

메뉴 속성 default는 메뉴 유형에 따라 기본 설정 상태를 지정하는 속성으로, y는 커널에 built-in, n은 no butlt-in, m은 module로 설정함을 나타낸다. 메뉴 속성 help는 이후의 문자열들은 해당 메뉴설정의 도움말임을 나타낸다. 이외의 메뉴 속성으로 다른 항목과의 의존성을 나타내는 depends on 속성이 있다.

커널 모듈 소스를 커널 소스내로 복사

이제 LED 제어용 커널 모듈을 등록하는 과정에 대해 살펴본다. LED 제어용 커널 모듈은 문자 디바이스 유형이다. 따라서 우선 커널 소스의 문자 디바이스에 대한 커널 모듈 소스 파일을 위치시키는 /usr/src/linux/drivers/char/ 디렉터리에 커널 모듈을 다음과 같이 복사한다. gpioled.c는 앞에서 살펴본 LED 제어용 커널 모듈의 소스이다.

```
# cd /usr/src/linux
# cp ???/gpioled.c ./drivers/char/
```

Kconfig 편집하여 메뉴 항목 추가

커널 컴파일시 커널 환경설정하는 menuconfig에서 커널 모듈에 대한 메뉴 항목이 나타나게 하려면 해당 디렉터리의 Kconfig 파일에 등록해야 한다.

```
# cd ./drivers/char/
```

메뉴 항목의 등록을 위해 Kconfig 파일의 적절한 위치에서 다음과 같이 bool 메뉴 유형의 메뉴 항목을 추가한다.

```
# nano ./drivers/char/Kconfig
```

```
.....  
source "drivers/char/xillybus/Kconfig"  
  
config GPIO_LED_TEST  
    bool "GPIO LED Module Test Program"  
    default y  
    help  
        GPIO LED Testing.....  
  
endmenu
```

만일 tristate 메뉴 유형으로 추가하는 경우 위의 메뉴 항목부분에서 다음과 같이 작성한다.

```
tristate "GPIO LED Module Test Program"
```

위와 같이 메뉴항목을 추가하면 해당 메뉴의 심볼명 GPIO_LED_TEST는 컴파일 등에서 CONFIG_가 접두한 CONFIG_GPIO_LED_TEST라는 환경변수로 접근 가능하다.

Makefile 수정

커널 컴파일할 때 커널 모듈이 컴파일될 수 있도록 현 디렉터리의 Makefile에 메뉴의 심볼명을 활용한 다음과 같은 라인을 추가한다. CONFIG_GPIO_LED_TEST는 Kconfig에서 등록한 모듈 항목에 대한 심볼명에 대한 환경변수임을 유의하고, 목적 코드명을 설정한다.

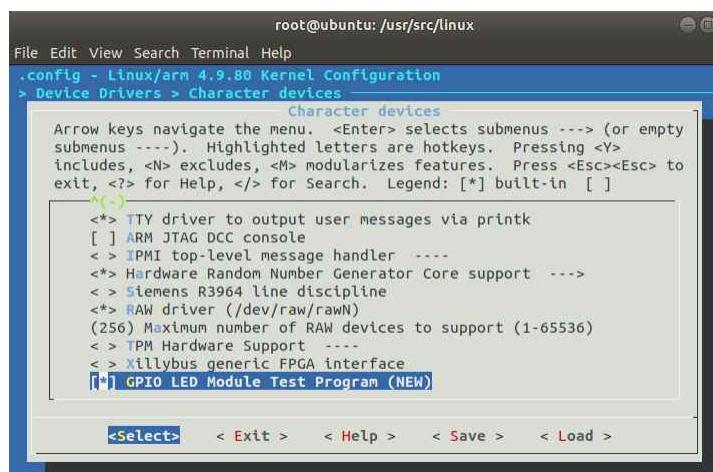
```
# nano ./drivers/char/Makefile  
.....  
obj-$(CONFIG_BRCM_CHAR_DRIVERS) += brcm/  
  
obj-$(CONFIG_GPIO_LED_TEST) += gpioled.o
```

커널 환경설정

```
# KERNEL=kernel7
# make bcm2709_defconfig
# make menuconfig
```

bool 유형의 경우, 메뉴에서 다음과 같이 이동하면 Kconfig 파일에서 등록한 항목을 관찰할 수 있다.

```
Device Drivers ->
    Character devices ->
        [GPIO LED Module Test Program]
```



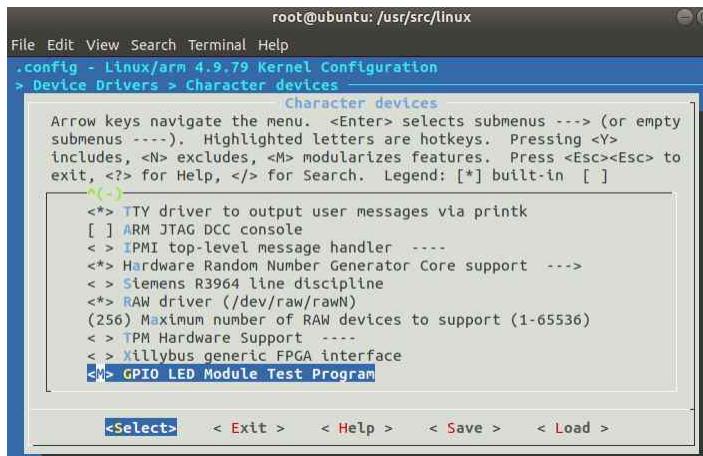
메뉴 유형이 bool인 경우 []이, tristate인 경우 <>로 메뉴항목 앞에 표시된다. *가 포함되면 현 선택상태는 y이며, m은 module을, 비어있으면 n인 상태를 나타낸다. 메뉴 유형이 bool 혹은 tristate에 따라 스페이스바를 눌러 적절히 선택하고 저장 한다. 설정된 환경설정 내용은 히든파일 .config에 저장된다. y를 선택하는 경우 커널 컴파일후 커널 이미지에 해당 커널 모듈이 포함되며, m이 설정되면 커널 모듈은 커널 모듈 라이브러리에 포함된다.

tristate 유형의 경우, 메뉴에서 다음과 같이 이동하면 Kconfig 파일에서 등록한 항목을 관찰할 수 있다.

```
Device Drivers ->
```

Character devices ->

< >GPIO LED Module Test Program



다음의 명령을 통해서 커널 모듈에 대한 심볼명이 .config 헤든 파일에 포함되었는지와 그 설정 상태를 확인할 수 있다. bool 유형으로 y를 선택한 경우는 다음과 같다.

```
# cat .config | grep CONFIG_GPIO_LED*
CONFIG_GPIO_LED_TEST=y
```

tristate 유형으로 m을 선택한 경우는 다음과 같다.

```
# cat .config | grep CONFIG_GPIO_LED*
CONFIG_GPIO_LED_TEST=m
```

13.5.2 bool 메뉴 유형

Kconfig에 등록할 때 메뉴 유형이 bool 혹은 tristate 인 경우, 그 설정에 따라 커널 모듈을 사용하는 절차가 다를 수 있다. 우선 다음의 명령을 통해 커널 컴파일을 위한 커널 소스의 홈 디렉터리로 이동한다.

```
# cd /usr/src/linux
```

bool 메뉴 유형이며 y를 선택한 경우를 가정한다. 이 경우 커널 컴파일하면 커널 모듈은 커널 이미지 파일에 포함된다. 따라서 다음과 같이 커널 이미지 파일만 생성하도록 컴파일할 수 있다.

```
# make -j4 zImage
```

이제 NFS 서비스를 통해 타깃 보드로 커널 이미지 파일을 전달하여 /boot 디렉터리에 kernel7.img 파일로 위치시킨다.

```
# cp ./arch/arm/boot/zImage /nfs
```

타깃 보드에서 NFS 서비스를 통해 커널 이미지 파일을 /boot/ 디렉터리로 복사한다. 필요에 따라 기존 커널 이미지 파일을 다른 이름으로 토피시켜 둘 것을 권고한다.

```
$ sudo mount -t nfs 192.168.0.20:/nfs /share
$ cd /boot/
$ cp /share/zImage ./kernel7.img
```

커널 모듈이 포함된 새로운 커널 이미지로 재부팅한다.

```
$ sudo reboot
```

```
$ dmesg | grep Init*
[    0.308241] Init Module: Major number 201
[    0.775732] Init: Port Power? op_state=1
[    0.781612] Init: Power Port (0)
[    0.932344] Initializing XFRM netlink socket
[    5.490183] gpiomem-bcm2835 3f200000.gpiomem: Initialised: Registers
at 0x3f200000
```

부팅 후 모듈 등록 확인은 lsmod 명령으로는 확인불가하나, 다음의 명령과 같이

/proc/devices 파일을 확인하여 등록 여부를 확인할 수 있다.

```
$ cat /proc/devices | grep gpio*
201 gpioled
246 bcm2835-gpiomem
254 gpiochip
```

이제 다음과 같이 디바이스 파일을 생성하고 응용프로그램을 실행하여 동작을 확인할 수 있다.

```
$ sudo mknod /dev/gpioled c 201 0
$ sudo chmod 777 /dev/gpioled
$ ./gpioled_test on
혹은 $ ./gpioled_test off
```

13.5.3 tristate 메뉴 유형

tristate 메뉴 유형이며 m를 선택한 경우를 가정한다. 이 경우 커널 컴파일하면 LED 제어용 커널 모듈은 커널 모듈 라이브러리에 포함된다. 따라서 커널 이미지뿐만 아니라 커널 모듈 컴파일도 병행하도록 다음과 같은 명령을 사용한다. 추가한 커널 모듈 소스를 컴파일하는 과정을 관찰할 수 있다.

```
# make -j4 zImage modules dtbs
scripts/kconfig/conf --silentoldconfig Kconfig
.....
UPD      kernel/config_data.h
CC [M]   kernel/configs.o
CC [M]   drivers/char/gpioled.o
Building modules, stage 2.
Kernel: arch/arm/boot/Image is ready
Kernel: arch/arm/boot/zImage is ready
MODPOST 1573 modules
CC      drivers/char/gpioled.mod.o
CC      kernel/configs.mod.o
LD [M]  drivers/char/gpioled.ko
```

LD [M] kernel/configs.ko

그리고 다음의 명령을 사용하여 커널 모듈 라이브러리도 설치한다.

```
# make modules_install
```

위 명령으로 인해 다음과 같이 LED 제어용 커널 모듈인 gpioled.ko 파일은 /lib/modules/4.9.79-v7/kernel/drivers/char/ 디렉터리에 위치한다.

```
# cd /lib/modules/4.9.79-v7/
# ls kernel/drivers/char/
broadcom gpioled.ko
```

이제 NFS 서비스를 위해 다음 명령을 사용하여 /nfs 디렉터리로 커널 이미지 파일뿐만 아니라 커널 모듈 라이브러리까지 복사한다.

```
# cp ./arch/arm/boot/zImage /nfs
# cp -rf /lib/modules/4.9.79-v7/ /nfs
```

타깃 보드에서 NFS 서비스를 위한 마운트 후 커널 이미지 파일을 /boot/ 디렉터리로 복사하고 커널 모듈 라이브러리는 /lib/modules/ 디렉터리 하부에 복사한다. 필요에 따라 기존 커널 이미지 파일을 다른 이름으로 퇴피시켜 둘 것을 권고한다.

```
$ sudo mount -t nfs 192.168.0.20:/nfs /share
$ sudo cd /boot/
$ sudo cp /share/zImage ./kernel7.img
$ sudo cp -rf /share/4.9.79-v7/ /lib/modules/
```

m을 선택한 경우이므로 해당 커널 모듈을 부팅시에 적재하도록 하려면 다음과 같이 /etc/modules 파일에 "gpioled"를 추가해준다. 이 파일의 내용은 부팅시에 적재할 커널 모듈의 이름을 등록하는 파일이다. y를 선택하여 커널 이미지 파일에 커널 모듈을 포함시킨 경우 이 과정은 불필요하다. 만일 부팅시에 커널 모듈을 적재하도록 /etc/modules 파일에 등록하지 않은 경우 insmod, rmmod 명령을 사용하여 동적으로 링크 및 링크 해제하여 사용할 수 있다.

```
$ sudo nano /etc/modules  
gpioled
```

새로운 커널 이미지로 재부팅한다.

```
$ sudo reboot
```

재부팅후 다음의 명령들의 하나로 커널 모듈의 등록여부를 확인할 수 있다.

```
$ lsmod  
$ cat /proc/devices | grep gpio*  
201 gpioled  
246 bcm2835-gpiomem  
254 gpiochip
```

```
$ dmesg | grep Init*  
[ 0.308241] Init Module: Major number 201  
[ 0.775732] Init: Port Power? op_state=1  
[ 0.781612] Init: Power Port (0)  
[ 0.932344] Initializing XFRM netlink socket  
[ 5.490183] gpiomem-bcm2835 3f200000.gpiomem: Initialised: Registers  
at 0x3f200000
```

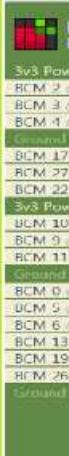
이제 다음과 같이 디바이스 파일을 생성한 후, 응용프로그램을 실행하여 결과를 확인할 수 있다.

```
$ sudo mknod /dev/gpioled c 201 0
```

```
$ sudo ./gpioled_test on  
혹은 $ sudo ./gpioled_test off
```

참고자료

- [1] 커널 컴파일(크로스컴파일포함) <https://phj790122.blog.me/221055176981>
- [2] 커널 크로스컴파일 <http://webnautes.tistory.com/547>
- [3] file_operations 구조체 <https://chardoc.tistory.com/8>
- [4] 디바이스드라이버
<http://blog.naver.com/PostView.nhn?blogId=jhjclever&logNo=220897769436&parentCategoryNo=&categoryNo=&viewDate=&isShowPopularPosts=false&from=postView>
- [5] 디바이스 드라이버 모듈 테스트, 한기대 임베디드응용및실습 교과자료
- [6] 디바이스 드라이버에 의한 GPIO LED 제어
<https://phj790122.blog.me/221069905163>
- [7] LED 디바이스드라이버
<http://newind2000.tistory.com/341?category=600663>
- [8] 디바이스 드라이버에 의한 LED 및 버튼스위치 제어
<https://phj790122.blog.me/221069905163>
- [9] Kconfig 명령 <http://mungi.kr/292>
- [10] 커널에 모듈 추가 <http://kkapsfam.tistory.com/4>



제14장 소켓프로그래밍에 의한 원격 제어

TCP 소켓 및 UDP 소켓을 활용한 소켓프로그래밍에 대해 살펴보고, 소켓 프로그램을 통해 서버 측의 디바이스를 원격에서 제어하는 것에 대해 살펴본다.

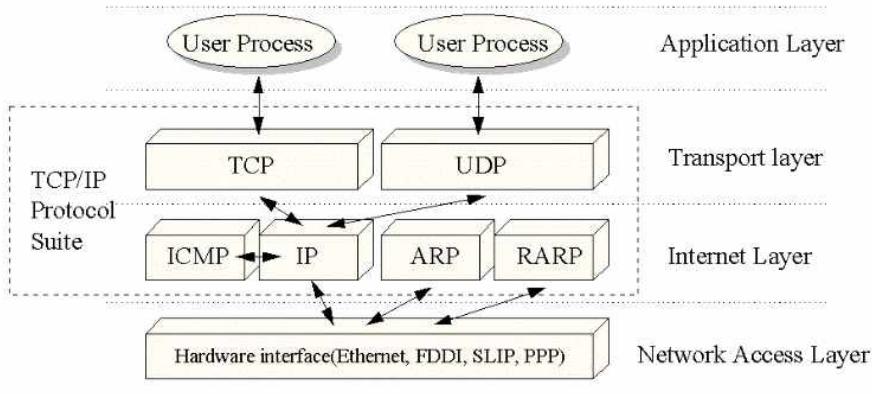
14.1 TCP/IP 프로토콜

14.1.1 TCP/IP 프로토콜

통신 프로토콜이란 한 컴퓨터에서 다른 컴퓨터로 데이터를 송수신하는 과정에서 필요한 사항을 규정한 약속이며, 오늘날 인터넷의 통신 프로토콜의 표준은 TCP/IP 스택이다.

오늘날의 인터넷은 1960년대 미국 국방부가 대학, 연구소 등의 연구기관들을 네트워크로 연결하는 ARPANet을 구축하고, 방위 관련 연구자들끼리 자유롭게 연구정보를 교환하기 위한 네트워크로부터 출발한다. 이 망에서의 통신 프로토콜이 TCP/IP 프로토콜이다. 점차 이 망에 또 다른 망들이 합류하고, 초기의 TCP/IP를 개선하고 표준화하면서 오늘날 전 세계적인 규모의 인터넷으로 성장하게 되었다.

TCP/IP 프로토콜 계층 구조는 다음 그림과 같다. TCP/IP는 각 계층은 인접계층간의 인터페이스를 제공하며, 통신과정에서 문제가 있을 때 특정 계층에만 집중하여 보완할 수 있도록 통신을 위한 규약들을 4개 계층으로 구분하고 있다.



애플리케이션 계층은 사용자가 사용하는 특정한 목적을 위한 소프트웨어의 실행을 위한 프로토콜을 의미하며, 예로 Telnet, FTP, SMTP, HTTP 등의 프로토콜들이 있다.

트랜스포트 계층(transport layer)은 전송 데이터를 일정 단위(패킷)로 나누고 포장하는 것에 관한 규약으로, TCP 프로토콜과 UDP 프로토콜이 대표적이다. TCP(Transmission Control Protocol) 프로토콜은 송수신측 간에 데이터를 신뢰성 있게 전달하기 위한 프로토콜로, 데이터를 전송하기 전에 데이터 전송을 위한 연결을 확보하고 데이터를 송수신하는 연결지향 프로토콜이다. 또한 데이터를 패킷으로 나누어 전송 시 전달되는 과정에서 패킷이 손망실되면 그 오류를 찾아 재전송을 요청하거나, 여러 개로 나뉜 패킷이 서로 다른 경로를 통해 목적지에 도달될 때 도착 순서는 전송 순서와 다를 수 있어 그 순서를 재조합하는 등의 데이터 송수신에서 신뢰성을 보장하는 프로토콜이다. 반면 신뢰성 보장을 위해 계층을 거칠 때마다 패킷에 헤더정보를 추가함으로써 통신과정에서 부담을 주는 점도 있다. 일 반적으로 TCP 프로토콜은 전송처리 속도는 감수하더라도 신뢰성 있는 데이터 전송이 필요한 경우에 사용하는 프로토콜이라 할 수 있다.

UDP(User Datagram protocol) 프로토콜은 데이터 전달을 위한 연결을 미리 설정하지 않고, IP 주소 및 포트 번호를 매 송수신 때마다 패킷에 실어 네트워크에 송출하는 방식으로 통신한다. 계층을 거치면서 신뢰성보장을 위한 헤더 정보가 포함되지 않으므로 통신 과정에서 부담이 적으며 고속 처리가 가능한 프로토콜이기도 하다. UDP 프로토콜은 데이터 전송의 신뢰성을 보장하지 않더라도 소량의 데

이터 전송이나 고속성이 요구되는 멀티미디어 응용 등에서 유용한 프로토콜이다.

인터넷 계층(internet layer)의 대표적인 프로토콜은 IP(internet protocol)이다. IP 프로토콜은 패킷에 목적지 IP 주소를 할당하고, 네트워크를 통해 목적지 시스템으로 도달하게 하는 프로토콜이다. 패킷의 전달경로는 서로 다를 수 있으며 패킷 전송의 신뢰성과 연결성은 제공하지 못한다.

매체접근 계층(media access layer, 링크 계층)은 네트워크에 접속하는 하드웨어 와의 물리적 인터페이스를 제공한다.

TCP/IP 프로토콜의 IP 계층에서 망을 통해 패킷을 목적지 시스템으로 전달하기 위한 식별 수단으로 IP 주소가 사용된다. 또한 전달 계층에서는 시스템에 도착된 패킷을 응용 계층의 어느 응용 소프트웨어로 전달할지를 식별하기 위한 수단으로 포트 번호가 사용된다. 가령 패킷이 수신되었는데 FTP 응용 소프트웨어로 혹은 HTTP 응용 소프트웨어로 보내야 할지를 식별할 때 포트 번호가 사용된다. 참고로 물리 계층에서 해당 시스템을 식별할 수 있는 수단으로 NIC 혹은 MAC 주소가 사용된다.

14.1.2 소켓 프로그래밍

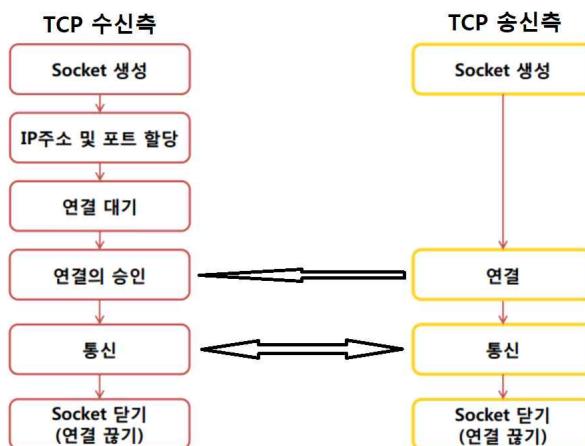
운영체제는 네트워크를 통해 송신측과 수신측 간의 데이터를 전송하기 위한 수단으로 소켓을 제공한다. 따라서 송수신측간 데이터 전송을 위한 소프트웨어 구현과정을 소켓 프로그래밍이라 한다. 여기서 소켓은 전기제품을 사용하기 위해 전력망에 전원 코드를 소켓에 삽입하는 것과 유사한 의미에서 유래되었으며, 데이터 전송을 위해 네트워크에 접속하는 수단이 소켓인 것이다.

TCP 방식의 통신 절차

TCP 프로토콜은 데이터 전송에 앞서 송신측과 수신측간에 연결을 미리 확보하고, 패킷에서의 오류 검출 및 재조합 등을 통해 신뢰성 있는 데이터 전송을 가능케 하는 프로토콜이다.

TCP 프로토콜을 사용한 송수신측간의 통신 절차는 아래 그림과 같다. 수신측은 소

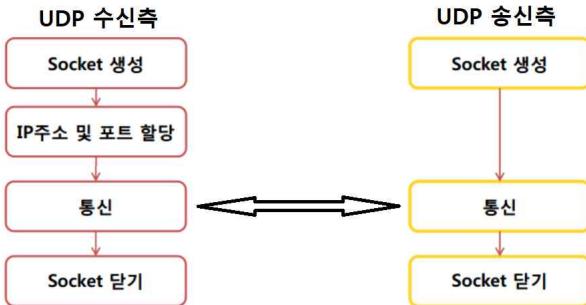
켓을 생성하고 자신의 IP 주소 및 포트 번호를 설정하고 클라이언트로부터의 연결 요청을 대기한다. 송신측도 소켓을 생성하고 자신의 IP 주소 등을 설정한 후 서버로의 연결 요청을 한다. 수신측은 송신측의 연결 요청을 수락하여 연결을 설정한다. 이후 송수신측은 서로 데이터를 송수신하며 어느 한쪽에서 소켓을 닫음으로써 연결은 해제된다.



UDP 방식의 통신 절차

UDP 프로토콜은 데이터 전송에 앞서 송신측과 수신측간에 연결을 미리 확보하지 않고, 패킷을 네트워크에 송출하는 비연결성의 그리고 패킷의 오류에 대한 조치가 없으므로 데이터의 신뢰성을 보장하지 못하는 프로토콜로, 고속성이 요구되는 상황에서 가용하다.

UDP 프로토콜을 사용한 송수신측간의 통신 절차는 아래 그림과 같다. 수신측은 소켓을 생성하고 자신의 IP 주소 및 포트 번호를 설정하고 클라이언트로부터의 패킷 수신을 대기한다. 송신측은 소켓을 생성하고 자신의 IP 주소 등을 설정한다. 또한 수신측 IP 주소 등의 정보와 자신의 IP 주소 등을 패킷에 실어 망에 송출한다. 수신측은 송신측의 패킷을 수신한다. 송신측으로 데이터를 전송할 때는 수신한 패킷으로부터 수신측 IP 주소 등의 정보를 패킷과 함께 망에 송출한다. 이와 같은 방식으로 데이터 송수신이 이루어진다. UDP에서의 소켓 닫음은 TCP와는 달리 생성된 소켓을 해제할 뿐이다.



소켓 생성 함수 : `socket()`

소켓의 유형에는 TCP 프로토콜에서 사용하는 Stream 유형과 UDP 프로토콜에서 사용되는 Datagram 유형이 있다.

소켓 프로그래밍에서 IP 주소를 표현하거나 소켓을 생성하여 데이터 송수신을 위해서는 다음의 헤더파일을 포함시킨다.

```
#include <sys/socket.h>
#include <sys/types.h>
```

소켓 프로그램에서 소켓을 생성하는 함수는 다음과 같으며, 설정에 따라 스트림 소켓 혹은 데이터그램 소켓을 생성한다.

```
int socket(int domain, int type, int protocol);
```

domain에는 IP 주소체계를 지정하는 것으로 AF_INET(address family) 혹은 PF_INET(protocol family)로 설정한다. type은 소켓 유형을 설정하는 것으로 스트림 유형의 경우 SOCK_STREAM을, 데이터그램 유형의 경우 SOCK_DGRAM으로 설정한다. protocol은 프로토콜을 설정하는 것으로 0으로 설정하거나, TCP 혹은 UDP에 따라 각각 IPPROTO_TCP 혹은 IPPROTO_UDP로 설정할 수 있다.

TCP 프로토콜을 위한 스트림 유형의 소켓을 생성하거나, UDP 프로토콜을 위한 데이터그램 유형의 소켓을 생성하려면 다음과 같이 호출한다.

```
sock = socket(AF_INET, SOCK_STREAM, 0);      // 스트림 소켓
sock = socket(AF_INET, SOCK_DGRAM, 0);        // 데이터그램 소켓
```

소켓 주소 구조체 : sockaddr

패킷 송수신을 위해 송수신측 시스템을 식별하는 등을 위해서 IP 주소체계, IP 주소 및 포트 번호의 정보가 필요하다. 이러한 정보를 담을 수 있는 구조체들이 사용된다.

다음의 sockaddr 구조체는 소켓 주소를 표현하는 범용의 구조체이다. 즉, TCP/IP 소켓을 포함한 다양한 소켓의 주소체계를 위한 범용의 구조체이다. sa_family 필드는 소켓의 프로토콜 주소 체계를 지정하는 필드이며, sa_data는 IP 주소를 설정하는 필드로 주소 체계에 따라 실제 소요 바이트 수는 다를 수 있다.

```
struct sockaddr {
    sa_family_t sa_family; // 소켓 패밀리의 주소체계
    char sa_data[14];     // 해당 주소체계에서 사용하는 주소 정보
}
```

다음의 sockaddr_in 구조체는 TCP/IP 프로토콜의 IPv4 주소체계에서 사용하는 소켓 주소를 표현하는 구조체이다. sin_family는 소켓 주소 패밀리 체계를 지정하는 것으로 IPv4인 경우 AF_INET으로 설정한다. IPv6인 경우 AF_INET6이다. sin_port는 16비트의 포트 번호를, sin_addr은 또 다른 구조체로 IPv4 주소체계의 경우 32비트의 IP 주소를 설정한다. 포트 번호와 IP 주소는 빅엔디안 (big-endian) 방식으로 표현되어야 한다. sin_zero는 범용 구조체 sockaddr와 크기를 맞추기 위한 필드이다.

```
struct sockaddr_in {
    sin_family_t sin_family; // IPv4 주소체계 경우, AF_INET 설정
    uint16_t sin_port;      // 포트 번호, 16bit
    struct in_addr sin_addr; // IP 주소, 32bit
    char sin_zero[8];       // sockaddr 구조체와 크기 일치위해, 항상 0
}

struct in_addr {
    uint32_t s_addr;        // IPv4 주소체계의 IP 주소, 32bit
}
```

소켓 프로그램에서 sockaddr_in 구조체는 제반 정보를 설정하고, 앞서의 범용 소켓 주소 구조체인 sockaddr로 형변환하여 사용된다.

IPv4 주소체계를 사용하는 서버 측과 클라이언트 측에서 이들 구조체를 초기화하는 예를 살펴보면 다음과 같다.

서버 측의 초기화는 다음과 같다. sin_family에는 AF_INET, 포트 번호와 IP 주소는 빅 엔디안방식으로 설정되어야 한다. htons() 함수는 값을 네트워크상의 표현 방식인 빅 엔디안 방식으로 변환한다. INADDR_ANY는 서버가 실행되는 시스템에 설정된 IP 주소를 빅 엔디안방식으로 정의된 매크로다

```
#define PORT 0x5005
struct sockaddr_in server_address;
memset(&server_address, 0, sizeof(server_address));
server_address.sin_family = AF_INET;
server_address.sin_port = htons(PORT);
server_address.sin_addr.s_addr = INADDR_ANY;
```

클라이언트 측의 초기화는 다음과 같다. inet_addr() 함수는 점십진표기로 표현된 IP 주소에 대한 문자열을 빅엔디안 방식의 수치로 변환하는 함수이다.

```
struct sockaddr_in client_address;
memset(&client_address, 0, sizeof(client_address));
client_address.sin_family = AF_INET;
client_address.sin_port = htons(PORT);
client_address.sin_addr.s_addr = inet_addr("192.168.0.40"); // 서버IP
```

이들 초기화된 구조체는 소켓 프로그램에서 다음과 같이 범용 구조체 sockaddr로 형변환되어 사용된다.

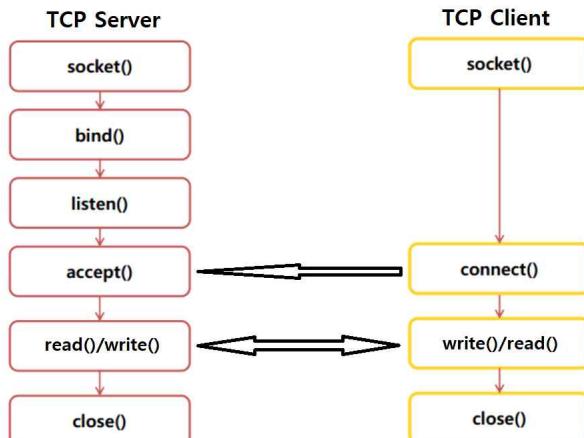
```
(struct sockaddr *)&server_address // 서버측, 범용구조체로 형변환
(struct sockaddr *)&client_address // 클라이언트측, 범용구조체로 형변환
```

14.2 TCP 소켓프로그래밍

TCP 방식에 의해 서버와 클라이언트 간 메시지를 송수신하는 서버 프로그램과 클라이언트 프로그램간의 함수 호출 절차, 관련 함수를 살펴본다. 또한 서버/클라이언트 프로그램을 구현하여 메시지 송수신을 살펴보고, 디바이스의 원격 제어도 확인해 본다.

14.2.1 TCP 서버/클라이언트간 함수 호출 절차

TCP 방식의 통신 절차를 소켓 관련 함수의 호출과정으로 표현하면 다음 그림과 같다.



TCP 서버는 `socket()` 함수로 소켓을 생성하고, 자신의 IP 주소 및 포트 번호를 `bind()` 함수를 통해 설정한다. `listen()` 함수로 연결 수를 지정하고 `accept()` 함수를 호출하여 클라이언트의 연결 요청까지 블록킹 상태에 있게 된다. 반면 클라이언트는 소켓을 생성하고 접속할 서버의 IP 주소 및 포트 번호를 설정하여 `connect()` 함수를 호출함으로써 서버에 연결 요청한다. 서버는 연결이 가능하면 `accept()` 함수의 블록킹 상태에서 벗어남으로써 연결 요청을 수락한다. 이후 서버와 클라이언트는 `read()`, `write()` 함수를 사용하여 데이터를 송수신한다. 서버 측이나 클라이언트 측에서 `close()` 함수를 호출하면 해당 쪽의 소켓이 종료된다.

트 측 어느 측에서나 close() 함수를 호출하여 소켓을 해제하여 연결을 종료할 수 있다.

14.2.2 TCP 소켓 관련 함수

이들 함수들은 서버 측에서 혹은 클라이언트 측에서 사용되는 함수들로 구분될 수 있다.

bind() 함수

bind() 함수는 통신을 위해 소켓 기술자와 소켓 주소 구조체를 연결하는 함수이며, 원형은 다음과 같다. sockfd는 socket() 함수를 통해 생성된 소켓 기술자를, myaddr은 소켓 주소 구조체를, addrlen은 소켓 주소 구조체의 크기를 설정하며, 정상적으로 이루어지면 반환값은 0, 오류 발생시는 -1을 반환한다.

```
int bind(int sockfd, struct sockaddr *myaddr, int addrlen);
```

다음의 사용 예에서 보듯이 소켓 주소 구조체는 범용구조체로 형변환하여 사용한다.

```
bind(sockfd, (struct sockaddr *)&server_address, sizeof(server_address));
```

listen() 함수

listen() 함수는 클라이언트로부터의 가용한 연결요청 대기 큐의 크기를 설정하는 함수로 원형은 다음과 같다. n은 서버가 수용할 수 있는 클라이언트 요청의 대기 큐의 크기를 설정한다. 클라이언트의 연결 요청은 대기 큐에 머물다 가용할 때 요청 순서에 따라 서비스된다.

```
int listen(int sockfd, int n)
```

다음의 사용 예는 서버가 최대 2개의 connect() 요청을 대기시킬 수 있으며, 세 번째 이후의 connect() 요청은 거절하여 클라이언트가 알게 한다.

```
listen(sockfd, 2);
```

accept() 함수

accept() 함수는 실행중일 때 클라이언트의 연결요청이 있을 때까지 블록킹 상태에 있게 되며, 클라이언트의 연결 요청이 있을 때 통신 채널을 생성함으로써 연결요청을 수락하고 클라이언트의 주소 구조체를 포인터로 반환한다. addr 및 addrlen은 연결 수락된 클라이언트의 주소 구조체와 그 크기를 반환받기 위한 포인터이다. 그리고 클라이언트의 소켓 기술자를 반환한다. 이후 클라이언트 측과의 송수신은 소켓 기술자를 이용한다.

```
int accept(int sockfd, struct sockaddr *addr, int *addrlen)
```

이 함수가 실행중일 때 클라이언트의 연결요청이 있을 때까지 블록킹 상태에 있게 됨을 유의한다.

다음의 사용 예와 같이 호출한다.

```
cli_fd = accept(serv_fd, (struct sockaddr *)&cli_addr, &clilen))
```

connect() 함수

connect() 함수는 클라이언트가 서버로의 연결 요청을 하는 함수로 원형은 다음과 같다. sockfd는 클라이언트의 소켓 기술자를, serv_addr 및 addrlen은 서버의 주소 정보를 담고 있는 주소 구조체와 그 크기를 설정한다.

```
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

다음의 사용 예와 같이 함수 호출 전에 연결한 서버에 대한 주소 구조체를 초기화하여야 한다.

```
struct sockaddr_in serv_addr;
```

serv_addr.sin_family	= AF_INET;
serv_addr.sin_addr.s_addr	= inet_addr("192.168.0.40");
serv_addr.sin_port	= htons(0x5005);

```
connect(sock_fd, (struct sockaddr *)&serv_addr, sizeof(serv_addr))
```

read() / write() 함수

read() / write() 함수는 TCP 방식에서의 패킷 송수신에 사용되는 함수로, 함수 원형은 다음과 같다. sockfd는 상대 측의 소켓 기술자를 설정하는데 서버 경우 accept() 함수가 반환한 소켓기술자를, 클라이언트의 경우 자신의 소켓 기술자를 설정한다. buf에는 송수신 데이터 버퍼를, bufsize는 버퍼 크기를 지정하며 정상적인 송수신시 반환값은 송수신한 바이트수다.

```
int write(sockfd, char *buf, int bufsize)
int read(sockfd, char *buf, int bufsize)
```

UDP 방식에서는 이들 함수 대신 sendto(), recvfrom() 함수를 사용한다.

close() 함수

close() 함수는 통신을 위한 소켓을 닫음으로써 통신을 종료하는 함수로 원형은 다음과 같다. TCP 방식에서는 미처리된 패킷들을 모두 처리한 후에 소켓을 닫음으로써 연결을 종료한다. 예외적으로 미처리 패킷을 폐기하거나, 지정 시간동안 처리하도록 할 경우 setsockopt()를 부가적으로 사용할 수 있다. 반면 UDP 방식에서는 단순히 소켓을 닫는 작업만 한다.

```
close(sockfd);
```

14.2.3 TCP 서버/클라이언트

TCP 방식에 의해 서버와 클라이언트 간 메시지를 송수신하는 서버 프로그램과 클라이언트 프로그램을 구현한다. 서버의 포트번호로 0x5005를 사용하기로 한다.

Linux용 TCP server 소스

```
$ nano tcpServer.c
//=====
// tcpSever.c (Linux)
//      for Chatting
```

```
//=====
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <netinet/in.h>

#define PORTNUM      0x5005          // port#
#define BUFFSIZE     256

int main(void) {
    struct sockaddr_in serv_addr, cli_addr;
    int serv_fd, cli_fd, clilen;
    char buffer[BUFFSIZE];

    printf("[TCP server for chatting...]\n");

    // 1) create server socket
    if((serv_fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("ERROR opening socket");
        exit(1);
    }

    // 2) setting server socket structure
    memset((char *) &serv_addr, 0x00, sizeof(serv_addr));
    serv_addr.sin_family      = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port        = htons(PORTNUM);

    // 3) bind()
    if(bind(serv_fd, (struct sockaddr *)&serv_addr,
            sizeof(serv_addr)) == -1) {
        perror("ERROR on binding");
        exit(1);
    }

    // 4) listen()
    listen(serv_fd, 5);

loop:
    // 5) accept(), blocking...
```

```
clilen = sizeof(cli_addr);
if((cli_fd = accept(serv_fd, (struct sockaddr *)&cli_addr,
&clilen)) == -1) {
    perror("ERROR on accept");
    exit(1);
}

// 6) read/write, write(), sending...
write(cli_fd, "Welcome to Chat Server.....", BUFFSIZE);

while(1) {
    memset(buffer, 0x00, sizeof(buffer));

    // 6) read/write, read(), receiving...
    if((read(cli_fd, buffer, BUFFSIZE)) == -1) {
        perror("ERROR reading from socket");
        exit(1);
    }
    printf("[Guest] %s\n", buffer);

    if(buffer[0] == 'q') {
        close(cli_fd);
        goto loop;
    }

    memset(buffer, 0x00, sizeof(buffer));
    printf("[Server] ");
    fgets(buffer, BUFFSIZE, stdin);

    // 6) read/write, write(), sending...
    write(cli_fd, buffer, BUFFSIZE);
    if(buffer[0] == 'q')
        break;
}

// 7) close(), close server socket, disconnection
close(serv_fd);

return 0;
}
```

Linux용 TCP client 소스

자신의 실습환경에 맞게 서버의 IP 주소를 다음의 소스에서 SERVERIP 매크로에 지정하여 저장함을 유의한다.

```
$ make tcpClient.c
//=====
//  tcpClient.c (Linux)
//      for Chatting
//=====

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>

#define SERVERIP    "192.168.0.40"          // taget !!
#define PORTNUM     0x5005                  // port
#define BUFFSIZE    256

int main(void) {
    int sock_fd;
    struct sockaddr_in serv_addr;
    char buffer[BUFFSIZE];

    printf("[TCP server for chatting...]\n");

    // 1) create client socket
    if((sock_fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("ERROR opening socket");
        exit(1);
    }

    // 2) setting server socket structure
    serv_addr.sin_family      = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(SERVERIP);
    serv_addr.sin_port        = htons(PORTNUM);

    // 3) connect
    if(connect(sock_fd, (struct sockaddr *)&serv_addr,
```

```
        sizeof(serv_addr)) == -1){
    perror("ERROR connecting");
    exit(1);
}

// 4) read/write, read(), receiving...
memset(buffer, 0x00, sizeof(buffer));
if(read(sock_fd, buffer, BUFFSIZE)== -1){
    perror("ERROR reading from socket");
    exit(1);
}
printf("[Server] %s\n", buffer);

while(1) {
    memset(buffer, 0x00, sizeof(buffer));

    printf("[Guest] ");
    fgets(buffer, BUFFSIZE, stdin);

    // 4) read/write, write(), sending...
    if(write(sock_fd, buffer, strlen(buffer)) == -1) {
        perror("ERROR writing to socket");
        exit(1);
    }
    if(buffer[0] == 'q')
        break;

    // 4) read/write, read(), receiving...
    memset(buffer, 0x00, sizeof(buffer));
    if(read(sock_fd, buffer, BUFFSIZE) == -1) {
        perror("ERROR reading from socket");
        exit(1);
    }

    printf("[Server] %s\n", buffer);
    if(buffer[0] == 'q')
        break;
}

// 5) close(), close client socket, disconnection
close(sock_fd);
```

406 라즈베리파이기반 임베디드시스템응용

```
        return 0;  
}
```

[실습1] TCP 방식의 서버/클라이언트 I

Linux 시스템간 위 소스를 활용하여 TCP 서버와 클라이언트간 메시지 송수신을 확인한다.

TCP 방식의 서버 소스와 클라이언트 소스를 다음 명령으로 컴파일한다.

```
$ make tcpServer  
$ make tcpClient
```

다음과 같이 서버 측 프로그램 tcpServer는 라즈베리파이 보드에서 실행하여 대기하고, 클라이언트 측 프로그램 tcpClient는 가상머신으로 이동하여 실행한다.

라즈베리파이보드 : Linux 서버 측 \$./tcpServer
가상머신 : Linux 클라이언트 측 \$./tcpClient

서로 번갈아가며 메시지를 전달하여 그 결과를 관찰한다. 연결 종료를 위해서는 메시지의 첫 문자를 q로 시작하면 된다.

Windows용 TCP client 소스

Windows 환경에서의 소켓 프로그래밍 즉, 원속 프로그래밍을 할 때 winsock2.h 헤더 파일을 아래와 같이 포함하여야 한다.

```
#include <winsock2.h>
```

윈도우 소켓의 버전과 해당 버전을 지원하는 라이브러리를 초기화하는 과정이 필수적이며 다음의 코드를 포함해야한다. MAKEWORD 매크로 함수를 사용하여 주변호 2, 부변호 2의 소켓 버전을 사용하고, 해당버전의 원속 라이브러리를 사용하겠다고 초기화하는 것이다.

```
WSADATA wsaData;
```

```

if(WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
    perror("ERROR WSAStartup()");
    exit(1);
}

```

초기화된 라이브러리의 해제를 위해서는 다음의 함수를 호출하면 되며, 통상 프로그램의 종료 직전에 위치시킨다.

```
WSACleanup();
```

또한, 소켓 관련 함수들은 소켓을 닫을 때 close() 대신 closesocket() 함수를 사용하는 것을 제외하고는 리눅스에서의 소켓 프로그래밍과 동일하다.

다음의 Windows용 클라이언트 소스는 범용성을 위해 실행할 때 서버의 포트 번호를 명령행에 인자로 전달할 수 있도록 다음과 같이 구현한다. 포트 번호는 0x5005로 정한다.

```

//=====
// tcpClientWin.c (Windows)
//      for Chatting
//=====

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <winsock2.h>           // *)

#define SERVERIP      "192.168.0.40"      // taget !!
#define PORTNUM       0x5005                // port
#define BUFFSIZE       256

int main(int argc, char *argv[]) {
    SOCKET sock_fd;
    SOCKADDR_IN serv_addr;
    WSADATA wsaData;
    char buffer[BUFFSIZE];

    if(argc != 2) {
        printf("usage : %s [port]\n", argv[0]);
    }
}
```

```
        exit(0);
    }

    printf("[TCP client for chatting...\n");

    if(WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        perror("ERROR WSAStartup()");
        exit(1);
    }

    // 1) create client socket
    if((sock_fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("ERROR opening socket");
        exit(1);
    }

    // 2) setting server socket structure
    serv_addr.sin_family      = AF_INET;
    serv_addr.sin_addr.s_addr  = inet_addr(argv[1]); // SERVERIP
    serv_addr.sin_port         = htons(PORTNUM);

    // 3) connect
    if(connect(sock_fd, (SOCKADDR *)&serv_addr,
               sizeof(serv_addr)) == -1) {
        perror("ERROR connecting");
        exit(1);
    }

    // 4) read/write, read(), receiving...
    memset(buffer, 0x00, sizeof(buffer));
    if(recv(sock_fd, buffer, BUFFSIZE, 0) == -1) {
        perror("ERROR reading from socket");
        exit(1);
    }
    printf("[Server] %s\n", buffer);

    while(1) {
        memset(buffer, 0x00, sizeof(buffer));
        printf("[Guest] ");
        fgets(buffer, BUFFSIZE, stdin);

        // 4) read/write, write(), sending...
```

```

        if(send(sock_fd, buffer, strlen(buffer), 0) == -1) {
            perror("ERROR writing to socket");
            exit(1);
        }
        if(buffer[0] == 'q')
            break;

        // 4) read/write, read(), receiving...
        memset(buffer, 0x00, sizeof(buffer));
        if(recv(sock_fd, buffer, BUFFSIZE, 0) == -1) {

            perror("ERROR reading from socket");
            exit(1);
        }

        printf("[Server] %s\n", buffer);
        if(buffer[0] == 'q')
            break;
    }

    // 5) close(), close client socket, disconnection
    closesocket(sock_fd);

    WSACleanup();

    return 0;
}

```

[실습2] TCP 방식의 서버/클라이언트 II

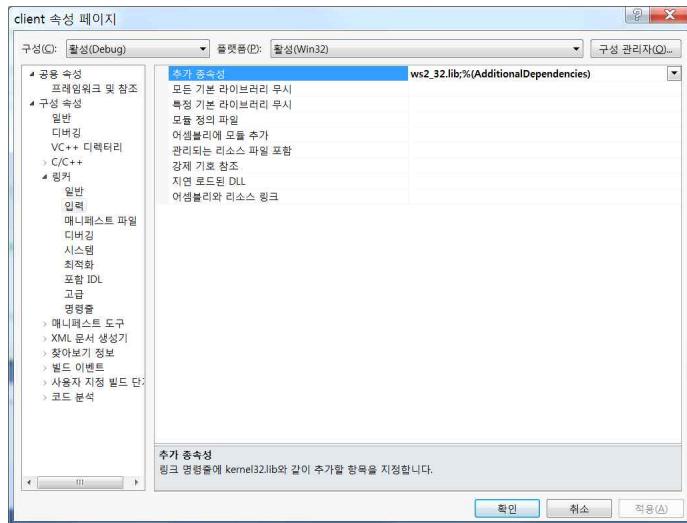
Linux TCP 서버와 Windows TCP 클라이언트간 메시지 송수신을 확인한다.

우선 Visul Studio를 실행하고 프로젝트 이름을 tcpClientWin으로 지정하고, 위의 소스 파일을 편집창에 복사한다.

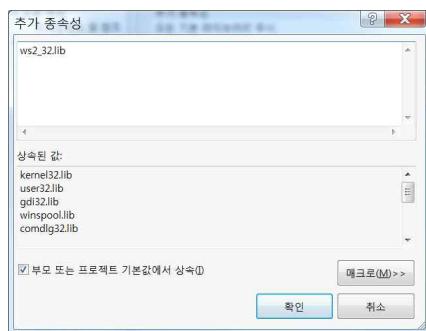
다음으로 원속 프로그램을 컴파일하려면 컴파일에 앞서 다음과 같은 과정으로 Windows 소켓 라이브러리 ws2_32.lib를 함께 링크할 수 있도록 환경 설정해야 한다.

4.10 라즈베리파이기반 임베디드시스템응용

단축키 Alt-F7 키를 눌러 다음과 같은 프로젝트의 속성 페이지 창을 열고, 좌측화면의 ‘구성속성 - 링커 - 입력’을 클릭하면 우측과 같은 화면이 나온다. 우측 화면의 ‘추가 종속성’ 항목을 클릭한다.



그리고 우측 끝의 역삼각형의 버튼을 클릭하고 편집항목을 선택하면 다음의 화면이 나타나고 여기서 ws2_32.lib를 입력하여 확인 버튼을 누른다.



이로써 원속 라이브러리를 링킹할 수 있도록 준비가 완료된다.

빌드를 위한 단축키 F7 키를 이용하여 컴파일하면 프로젝트 이름의 실행파일 tcpClientWin.exe가 생성된다.

동작확인을 위해 다음과 같이 서버 측 프로그램 tcpServer는 라즈베리파이 보드에서 실행하여 대기하고, 클라이언트 측 프로그램 tcpClientWin은 Windows 환경에서 실행한다. 클라이언트 프로그램을 실행할 때 서버의 IP 주소를 명령행에 전달해 준다.

라즈베리파이보드 : Linux 서버 측 \$./tcpServer

Windows : 클라이언트 측 c:> tcpClientWin 192.168.0.40

서로 번갈아가며 메시지를 전달하여 그 결과를 관찰한다. 연결 종료를 위해서는 메시지의 첫 문자를 q로 시작하면 된다.

다음 그림은 TCP 서버와 클라이언트 간에 교대로 메시지를 송수신하는 것을 보여 준다. 전달하는 메시지의 첫 글자가 q이면 접속을 종료한다.

```

pi@raspberrypi:~/IFC413/socket/tcp $ ./tcpServer
[TCP server for chatting...]
[Guest] hello....
[Server] hi.....
[Guest] quit

```



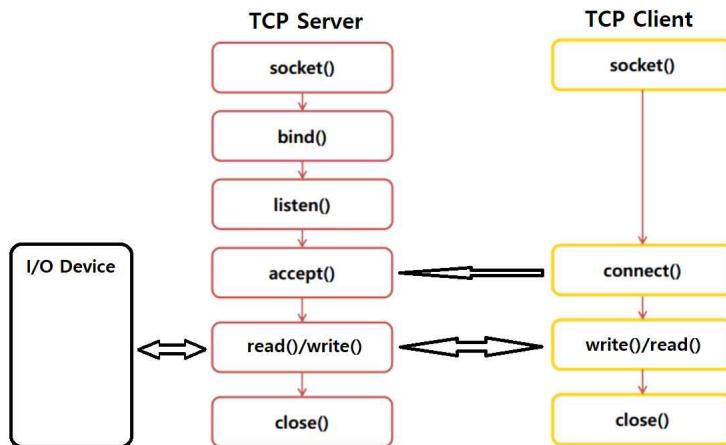
```

E:\MS_C\RA\SP\client\Debug> tcpClient 220.68.70.199
[TCP client for chatting...]
[Server] Welcome to Chat Server.....
[Guest] hello....
[Server] hi.....
[Guest] quit
E:\MS_C\RA\SP\client\Debug>

```

14.2.4 TCP 서버/클라이언트에 의한 디바이스 제어

TCP 서버/클라이언트 소스를 활용하여 서버 측에 있는 입출력 디바이스를 원격의 클라이언트가 제어하는 흐름은 다음 그림과 같이 표현할 수 있다. 패킷에 디바이스 제어 정보를 포함하여 전달함으로써 서버 측의 디바이스를 제어할 수 있다. 물론 서버 측에서는 입출력 디바이스를 접근할 수 있는 코드가 포함되어야 한다. 디바이스의 접근은 mmap(), ioremap(), 커널 모듈 등의 여러 방법으로 가능하다.



앞에서 살펴본 메시지를 주고받는 TCP 방식의 서버와 클라이언트 소스를 바탕으로, 디바이스를 제어하는 프로그램을 구현해본다. 서버 측 시스템에 연결된 디바이스를 제어하는 것이므로 서버 측 소스만 디바이스를 제어할 수 있도록 수정한다.

Linux TCP Server 소스

아래 소스에서 LED는 wiringPi 핀 번호체계 #1로 회로를 구성한 경우이다. 다음 소스의 볼드체 부분이 서버 시스템에 있는 LED 디바이스를 제어하기 위해 추가된 부분이다. 클라이언트 측으로부터 메시지의 첫 문자가 0 혹은 1에 따라 LED를 제어한다.

```

$ nano tcpServer_01.c
//=====
//  tcpSever_01.c (Linux)
//      Chatting + controlling LED device
//=====

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>

#include <wiringPi.h>           //

```

```

#define PORTNUM      0x5005      // port#
#define BUFFSIZE     256

#define P_LED        1           // BCM_GPIO #18

int main(void) {
    struct sockaddr_in serv_addr, cli_addr;
    int serv_fd, cli_fd, clilen;
    char buffer[BUFFSIZE];

    printf("[TCP server for chatting and controlling LED...]\n");

    // 1) create server socket
    if((serv_fd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("ERROR opening socket");
        exit(1);
    }

    // 2) setting server socket structure
    memset((char *) &serv_addr, 0x00, sizeof(serv_addr));
    serv_addr.sin_family      = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port        = htons(PORTNUM);

    // 3) bind()
    if(bind(serv_fd, (struct sockaddr *)&serv_addr,
            sizeof(serv_addr)) == -1) {
        perror("ERROR on binding");
        exit(1);
    }

    // 4) listen()
    listen(serv_fd, 5);

loop:
    // 5) accept(), blocking...
    clilen = sizeof(cli_addr);
    if((cli_fd = accept(serv_fd, (struct sockaddr *)&cli_addr,
                        &clilen)) == -1) {
        perror("ERROR on accept");
    }
}

```

```
        exit(1);
    }

    // 6) read/write, write(), sending...
    write(cli_fd, "Welcome to Chat Server.....LED control..", BUFFSIZE);

    if(wiringPiSetup() == -1)           // wiringPi pin #
        return 1;

    pinMode(P_LED, OUTPUT);          //

    while(1) {
        // 6) read/write, read(), receiving...
        memset(buffer, 0x00, sizeof(buffer));
        if((read(cli_fd, buffer, BUFFSIZE)) == -1){
            perror("ERROR reading from socket");
            exit(1);
        }
        printf("[Guest] %s\n", buffer);

        if(buffer[0] == 'q') {
            close(cli_fd);
            goto loop;
        }
        else if(buffer[0] == '0')           // LED control....
            digitalWrite(P_LED, LOW);
        else if(buffer[0] == '1')
            digitalWrite(P_LED, HIGH);

        // 6) read/write, write(), sending...
        memset(buffer, 0x00, sizeof(buffer));
        printf("[Server] ");
        fgets(buffer, BUFFSIZE, stdin);
        write(cli_fd, buffer, BUFFSIZE);
        if(buffer[0] == 'q')
            break;
    }

    // 7) close(), close server socket, disconnection
    close(serv_fd);

    return 0;
```

```
}
```

[실습3] TCP 서버/클라이언트에 의한 디바이스 제어

TCP 서버/클라이언트 메시지 송수신으로 서버 측에 연결된 LED 디바이스를 클라이언트에서 제어하는 것을 확인한다.

다음과 같이 서버 측 소스 파일을 컴파일한다.

```
$ make tcpServer_01.c
```

클라이언트 소스는 변경이 불필요하므로, 실행환경에 따라 앞에서 사용한 Linux용 혹은 Windows용 클라이언트 프로그램을 사용한다.

다음 그림과 같이 클라이언트 측에서 전송하는 문자열 중 첫 문자를 0 혹은 1로 시작하면, 이는 서버 측에 있는 LED 디바이스를 각각 OFF 혹은 ON으로 제어하므로 이를 확인한다.

```
pi@raspberrypi:~/IFC413/socket/tcp $ ./tcpServer_01
[TCP server for chatting and controlling LED...]
[Guest] hello.

[Server] hi.....
[Guest] 1

[Server] led...on.....
[Guest] 0

[Server] Led7|OFF.....
[Guest] ok

[Server] quit
pi@raspberrypi:~/IFC413/socket/tcp $
```

```
E:\WP_MS_C\WRASP\client\Debug> tcpClient 220.68.70.199
[TCP client for chatting...]
[Server] Welcome to Chat Server.....LED control..
[Guest] hello.
[Server] hi.....
[Guest] 1
[Server] led...on.....
[Guest] 0
[Server] Led OFF.....
[Guest] ok
[Server] quit

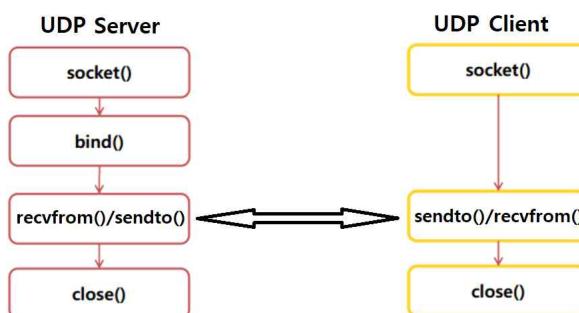
E:\WP_MS_C\WRASP\client\Debug>
```

14.3 UDP 소켓프로그래밍

UDP 방식에 의해 서버와 클라이언트 간 메시지를 송수신하는 서버 프로그램과 클라이언트 프로그램간의 함수 호출 절차, 관련 함수를 살펴본다. 또한 서버/클라이언트 프로그램을 구현하여 메시지 송수신을 살펴보고, 디바이스의 원격 제어도 확인해 본다.

14.3.1 UDP 서버/클라이언트간 함수 호출 절차

UDP 방식의 통신 절차를 소켓 관련 함수의 호출과정으로 표현하면 다음 그림과 같다.



UDP 서버는 `socket()` 함수로 소켓을 생성하고, 자신의 IP 주소 및 포트 번호를 `bind()` 함수를 통해 설정하고 `recvfrom()` 함수를 실행하여 클라이언트로부터의 패킷 수신을 대기한다. 반면 클라이언트는 소켓을 생성하고 `sendto()` 함수를 호출하여 자신의 IP 주소 및 포트번호와 접속할 서버의 IP 주소 및 포트 번호를 패킷과 함께 망에 송출한다. 서버에 패킷이 도착하면 그 패킷으로부터 클라이언트의 IP 주소 및 포트 번호와 자신의 IP 주소 및 포트 번호를 패킷과 함께 `sendto()` 함수 호출로 망에 송출한다. 이와 같은 방식으로 송수신이 이루어지며 송수신되는 패킷에는 목적지 IP 주소와 포트 번호가 항상 포함된다. 서버나 클라이언트의 `close()` 함수 호출은 생성된 소켓을 단순히 닫는다.

14.3.2 UDP 소켓 관련 함수

TCP 방식에서 데이터 송수신을 위해 `read()`와 `write()` 함수가 사용되었으나, UDP 방식에서는 이들 함수 대신 `sendto()`, `recvfrom()` 함수를 호출하여 사용한다.

`sendto()` / `recvfrom()` 함수

이들 함수들의 원형은 다음과 같다. 공통적으로 `s`는 해당 함수를 호출하는 자신의 소켓 기술자를, `buf`는 송수신 데이터 버퍼, `length`는 버퍼의 크기, 그리고 `flags`는 통상 0으로 설정한다. `sendto()` 함수에서 `to`와 `tolen`은 패킷 수신 측의 소켓 주소 구조체와 그 크기를 나타내며, `recvfrom()` 함수에서 `from`과 `fromlen`은 패킷 송신 측의 소켓 주소 구조체와 그 크기를 나타낸다.

```
int sendto(int s, char *buf, int length, int flags,
           sockaddr* to, int tolen);
int recvfrom(int s, char* buf, int length, int flags,
             sockaddr* from, socklen_t *fromlen);
```

위 함수들의 서버측에서의 사용 예는 다음과 같다.

```
sendto(serv_fd, buffer, BUFFSIZE, 0,
       (struct sockaddr *)&cli_addr, sizeof(cli_addr));

recvfrom(serv_fd, buffer, BUFFSIZE, 0,
         (struct sockaddr *)&cli_addr, &clilen);
```

14.3.3 UDP 서버/클라이언트

UDP 방식에 의해 서버와 클라이언트 간 메시지를 송수신하는 서버 프로그램과 클라이언트 프로그램을 구현한다.

UDP server 소스

```
//=====
```

```
// udpSever.c (Linux)
//      for Chatting
//=====
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>

#define PORTNUM      0x5005
#define BUFFSIZE     256

int main(void) {
    struct sockaddr_in serv_addr, cli_addr;
    int serv_fd, cli_fd, clilen;
    char buffer[BUFFSIZE];

    printf("[UDP Server for Chatting...\n");

    // 1) create server socket
    if((serv_fd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("ERROR opening socket");
        exit(1);
    }

    // 2) setting server socket structure
    memset((char *) &serv_addr, 0x00, sizeof(serv_addr));
    serv_addr.sin_family      = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port        = htons(PORTNUM);

    // 3) bind()
    if(bind(serv_fd, (struct sockaddr *) &serv_addr,
           sizeof(serv_addr)) == -1) {
        perror("ERROR on binding");
        exit(1);
    }

    while(1) {
        // 4) recvfrom/sendto,
        memset(buffer, 0x00, sizeof(buffer));
        clilen = sizeof(cli_addr);
```

```

        recvfrom(serv_fd, buffer, BUFFSIZE, 0,
                  (struct sockaddr *)&cli_addr, &clilen);
        printf("[Guest] %s\n", buffer);
        if(buffer[0] == 'q') {
            close(cli_fd);
            break;
        }

        // 4) recvfrom/sendto, sending...
        printf("[Server] ");
        memset(buffer, 0x00, sizeof(buffer));
        fgets(buffer, BUFFSIZE, stdin);
        sendto(serv_fd, buffer, BUFFSIZE, 0,
               (struct sockaddr *)&cli_addr, sizeof(cli_addr));
        if(buffer[0] == 'q')
            break;
    }
    // 5) close(),
    close(serv_fd);

    return 0;
}

```

Linux용 UDP client 소스

```

//=====
// udpClient.c (Linux)
//      for Chatting
//=====

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>

#define SERVERIP      "192.168.0.40"
#define PORTNUM       0x5005
#define BUFFSIZE       256

int main(void) {

```

```
struct sockaddr_in serv_addr;
int cli_fd;
int servlen;
char buffer[BUFFSIZE];

printf("[UDP Client for Chatting...\n");

// 1) create socket
if((cli_fd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    perror("ERROR opening socket");
    exit(1);
}

serv_addr.sin_family      = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr(SERVERIP);
serv_addr.sin_port        = htons(PORTNUM);

while(1) {
    memset(buffer, 0x00, sizeof(buffer));
    printf("[Guest] ");
    fgets(buffer, BUFFSIZE, stdin);

    // 2) recvfrom/sendto, sending....
    sendto(cli_fd, buffer, BUFFSIZE, 0,
           (struct sockaddr *)&serv_addr, sizeof(serv_addr));
    if(buffer[0] == 'q')
        break;

    // 2) recvfrom/sendto, receiving....
    memset(buffer, 0x00, sizeof(buffer));
    servlen = sizeof(serv_addr);
    recvfrom(cli_fd, buffer, BUFFSIZE, 0,
             (struct sockaddr *)&serv_addr, &servlen);
    printf("[Server] %s\n", buffer);
    if(buffer[0] == 'q')
        break;
}

// 3) close(), disconnection
close(cli_fd);

return 0;
```

```
}
```

[실습4] UDP 방식의 서버/클라이언트 I

위 소스를 활용하여 Linux 시스템 상에서 UDP 서버와 클라이언트간 메시지 송수신을 확인한다.

UDP 방식의 서버 소스와 클라이언트 소스를 다음 명령으로 컴파일한다.

```
$ make udpServer
$ make udpClient
```

다음과 같이 서버 측 프로그램 udpServer는 라즈베리파이 보드에서 실행하여 대기하고, 클라이언트 측 프로그램 udpClient는 가상머신으로 이동하여 실행한다.

라즈베리파이보드 : Linux 서버 측 \$./udpServer
 가상머신 : Linux 클라이언트 측 \$./udpClient

서로 번갈아가며 메시지를 전달하여 그 결과를 관찰한다. 소켓을 닫으려면 메시지의 첫 문자를 q로 시작하면 된다.

Windows용 UDP client 소스

```
=====//
// udpClientWin.c (Windows)
//      for Chatting
=====
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <winsock2.h>

#define SERVERIP    "220.68.70.199"
#define PORTNUM     0x5005
#define BUFFSIZE    256

int main(int argc, char *argv[]) {
```

```
SOCKADDR_IN serv_addr;
WSADATA wsaData;
SOCKET cli_fd;
int servlen;
char buffer[BUFFSIZE];

if(argc != 2) {
    printf("usage : %s [port]\n", argv[0]);
    exit(0);
}

printf("[TCP client for chatting...]\n");

if(WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
    perror("ERROR WSAStartup()");
    exit(1);
}

// 1) create socket
if((cli_fd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    perror("ERROR opening socket");
    exit(1);
}

serv_addr.sin_family      = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr(argv[1]); // SERVERIP
serv_addr.sin_port        = htons(PORTNUM);

while(1) {
    memset(buffer, 0x00, sizeof(buffer));
    printf("[Guest] ");
    fgets(buffer, BUFFSIZE, stdin);

    // 2) recvfrom/sendto, sending....
    sendto(cli_fd, buffer, BUFFSIZE, 0,
           (struct sockaddr *)&serv_addr, sizeof(serv_addr));
    if(buffer[0] == 'q')
        break;

    // 2) recvfrom/sendto, receiving....
    memset(buffer, 0x00, sizeof(buffer));
    servlen = sizeof(serv_addr);
```

```

recvfrom(cli_fd, buffer, BUFFSIZE, 0,
          (struct sockaddr *)&serv_addr, &servlen);
printf("[Server] %s\n", buffer);
if(buffer[0] == 'q')
    break;
}

// 3) close(), disconnection
closesocket(cli_fd);

return 0;
}

```

[실습5] UDP 방식의 서버/클라이언트 II

Linux UDP 서버와 Windows UDP 클라이언트간 메시지 송수신을 확인한다.

Visual studio에서 프로젝트 이름을 udpClientWin로 지정하고 위의 소스를 편집 창에 복사하고, 컴파일하면 udpClientWin.exe이 생성된다.

동작확인을 위해 다음과 같이 서버 측 프로그램 udpServer는 라즈베리파이 보드에서 실행하여 대기하고, 클라이언트 측 프로그램 udpClientWin은 Windows 환경에서 실행한다. 클라이언트 프로그램을 실행할 때 서버의 IP 주소를 명령행에 전달해준다.

라즈베리파이보드 : Linux 서버 측 \$./udpServer

Windows : 클라이언트 측 c:> udpClientWin 192.168.0.40

다음 그림은 UDP 서버와 클라이언트 간에 교대로 메시지를 송수신하는 것을 보여 준다. 전달하는 메시지의 첫 글자가 q이면 소켓을 닫는다.

```

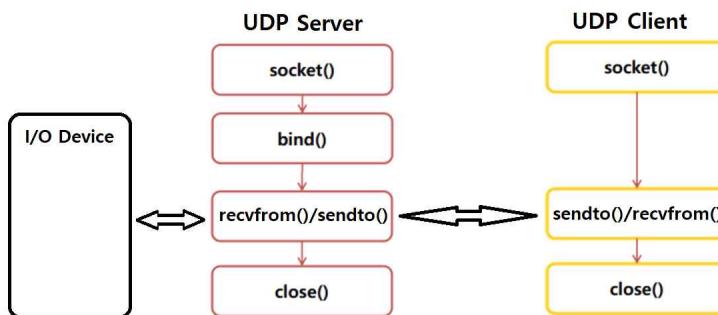
pi@raspberrypi:~/IFC413/socket/udp $ ./udpServer
[UDP Server for Chatting...
[Guest] hi
[Server] hello.....
[Guest] bye,..
[Server] bye, bye.....
[Guest] quit
pi@raspberrypi:~/IFC413/socket/udp $ 

```

```
E:\WP_MS_CWRASP\client\Debug> udpClient 220.68.70.199
[TCP client for chatting...]
[Guest] hi
[Server] hello.....
[Guest] bye...
[Server] bye, bye.....
[Guest] quit
E:\WP_MS_CWRASP\client\Debug>
```

14.3.4 UDP 서버/클라이언트에 의한 디바이스 제어

UDP 서버/클라이언트 소스를 활용하여 서버 측에 있는 입출력 디바이스를 원격의 클라이언트가 제어하는 흐름은 다음 그림과 같이 표현할 수 있다. 패킷에 디바이스 제어 정보를 포함하여 전달함으로써 서버 측의 디바이스를 제어할 수 있다. 물론 서버 측에서는 입출력 디바이스를 접근할 수 있는 코드가 포함되어야 한다. 디바이스의 접근은 mmap(), ioremap(), 커널 모듈 등의 여러 방법으로 가능하다.



앞에서 살펴본 메시지를 주고받는 UDP 방식의 서버와 클라이언트 소스를 바탕으로, 디바이스를 제어하는 프로그램을 구현해본다. 서버 측 시스템에 연결된 디바이스를 제어하는 것이므로 서버 측 소스만 디바이스를 제어할 수 있도록 수정한다.

Linux UDP Server 소스

아래 소스에서 LED는 wiringPi 핀 번호체계 #1로 회로를 구성한 경우이다. 다음 소스의 볼드체 부분이 서버 시스템에 있는 LED 디바이스를 제어하기 위해 추가된 부분이다. 클라이언트 측으로부터 메시지의 첫 문자가 0 혹은 1에 따라 LED를 제

어한다.

```
//=====
// udpSever_01.c (Linux)
//      for Chatting + controlling LED
//=====

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>

#include <wiringPi.h>          //



#define PORTNUM      0x5005
#define BUFFSIZE     256

#define P_LED 1           // BCM_GPIO #18

int main(void) {
    struct sockaddr_in serv_addr, cli_addr;
    int serv_fd, cli_fd, clilen;
    char buffer[BUFFSIZE];

    printf("[UDP Server for Chatting and controlling LED...\n");

    // 1) create server socket
    if((serv_fd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("ERROR opening socket");
        exit(1);
    }

    // 2) setting server socket structure
    memset((char *) &serv_addr, 0x00, sizeof(serv_addr));
    serv_addr.sin_family      = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port        = htons(PORTNUM);

    // 3) bind()
    if(bind(serv_fd, (struct sockaddr *) &serv_addr,
           sizeof(serv_addr)) == -1) {
```

```
    perror("ERROR on binding");
    exit(1);
}

if(wiringPiSetup() == -1)          // wiringPi pin #
    return 1;

pinMode(P_LED, OUTPUT);           //

while(1) {
    // 4) recvfrom/sendto,
    memset(buffer, 0x00, sizeof(buffer));
    clilen = sizeof(cli_addr);
    recvfrom(serv_fd, buffer, BUFFSIZE, 0,
              (struct sockaddr *)&cli_addr, &clilen);
    printf("[Guest] %s\n", buffer);
    if(buffer[0] == 'q') {
        close(cli_fd);
        break;
    }
    else if(buffer[0] == '0')          // LED control....
        digitalWrite(P_LED, LOW);
    else if(buffer[0] == '1')
        digitalWrite(P_LED, HIGH);

    // 4) recvfrom/sendto, sending...
    printf("[Server] ");
    memset(buffer, 0x00, sizeof(buffer));
    fgets(buffer, BUFFSIZE, stdin);
    sendto(serv_fd, buffer, BUFFSIZE, 0,
           (struct sockaddr *)&cli_addr, sizeof(cli_addr));
    if(buffer[0] == 'q')
        break;
}
// 5) close(),
close(serv_fd);

return 0;
}
```

[실습6] UDP 서버/클라이언트에 의한 디바이스 제어

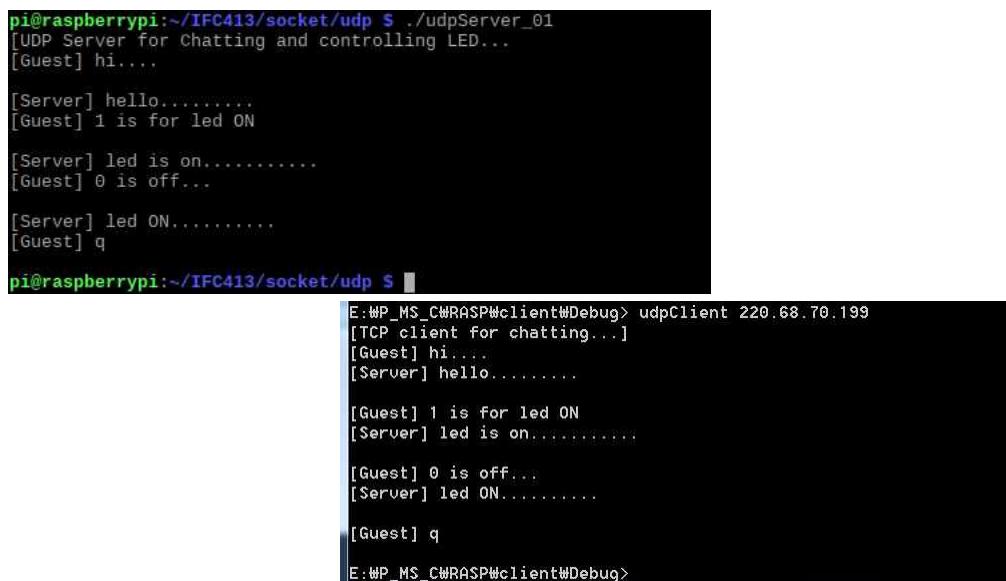
UDP 서버/클라이언트 메시지 송수신으로 서버 측에 연결된 LED 디바이스를 클라이언트에서 제어하는 것을 확인한다.

다음과 같이 서버 측 소스 파일을 컴파일한다.

```
$ make udpServer_01.c
```

클라이언트 소스는 변경이 불필요하므로, 실행환경에 따라 앞에서 사용한 Linux용 혹은 Windows용 클라이언트 프로그램을 사용한다.

다음 그림과 같이 클라이언트 측에서 전송하는 문자열 중 첫 문자를 0 혹은 1로 시작하면, 이는 서버 측에 있는 LED 디바이스를 각각 OFF 혹은 ON으로 제어하므로 이를 확인한다.



```

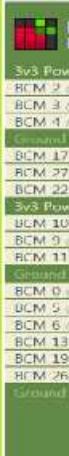
pi@raspberrypi:~/IFC413/socket/udp $ ./udpServer_01
[UDP Server for Chatting and controlling LED...
[Guest] hi....
[Server] hello.....
[Guest] 1 is for led ON
[Server] led is on.....
[Guest] 0 is off...
[Server] led ON.....
[Guest] q

pi@raspberrypi:~/IFC413/socket/udp $ E:\WP_MS_C\WRASP\client\Debug> udpClient 220.68.70.199
[TCP client for chatting...]
[Guest] hi....
[Server] hello.....
[Guest] 1 is for led ON
[Server] led is on.....
[Guest] 0 is off...
[Server] led ON.....
[Guest] q
E:\WP_MS_C\WRASP\client\Debug>

```

참고자료

- [1] TCP/IP 및 소켓프로그래밍 <https://itmining.tistory.com/127>
- [2] TCP 서버/클라이언트 <http://luckyyowu.tistory.com/71>
- [3] 소켓 프로그래밍 <http://luckyyowu.tistory.com/71?category=755949>
- [4] 윤성우의 열혈 TCP/IP 소켓프로그래밍, 윤성우, 오렌지출판사, 2012
- [5] 임베디드응용및실습, 소켓프로그래밍에 의한 디바이스제어
https://cms3.koreatech.ac.kr/sites/joo/IFC412/IFC412_13.pdf



제15장 웹서비스에 의한 원격 제어

APM이라 함은 Apache 웹서버, 프로그래밍 언어 PHP, 데이터베이스인 mySql을 의미하며, 이들은 많은 응용에서 서로 긴밀히 사용된다. APM을 설치할 때는 다음의 순서로 설치한다.

```
$ sudo apt-get install apache2          // apache2 웹 서버 설치  
$ sudo apt-get install mysql-server     // mysql 서버 설치  
$ sudo apt-get install mysql-client      // mysql 클라이언트 설치  
$ sudo apt-get install phpmyadmin        // phpmyadmin 설치
```

물론 웹 서비스만 할 경우라면 Apache 웹서버만 설치한다. 본 장의 실습을 위해 웹서버만 설치하여 진행한다.

15.1 Apache 웹서버

15.1.1 아파치 웹서버 설치

다음 명령과 같이 패키지 관리도구인 apt-get을 사용하여 라즈베리파이에 아파치 웹서버를 설치한다. 아파치 웹서버 패키지는 /etc/apache2에 설치된다.

```
$ sudo apt-get install apache2
```

다음 명령을 사용하여 아파치 웹서버 패키지 설치되어 실행중인지를 확인할 수 있다.

```
$ ps -ef | grep apache  
root      578      1  0 Aug12 ?          00:00:14 /usr/sbin/apache2 -k start  
www-data 15793    578  0 06:25 ?          00:00:00 /usr/sbin/apache2 -k start  
www-data 15794    578  0 06:25 ?          00:00:00 /usr/sbin/apache2 -k start  
www-data 15795    578  0 06:25 ?          00:00:00 /usr/sbin/apache2 -k start
```

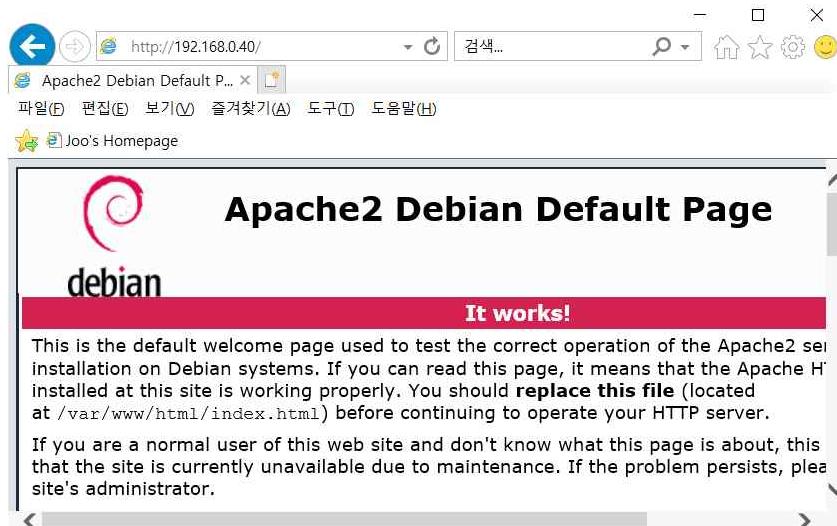
430 라즈베리파이기반 임베디드시스템용

```
www-data 15796  578  0 06:25 ?          00:00:00 /usr/sbin/apache2 -k start
www-data 15797  578  0 06:25 ?          00:00:00 /usr/sbin/apache2 -k start
pi       16191 16169  0 08:12 pts/0    00:00:00 grep --color=auto apache
```

설치후 내정된 웹 서버의 홈 디렉터리는 /var/www/html/이며, 첫 페이지의 HTML 파일 이름은 index.html이다.

이제 웹 브라우저의 주소창에 다음의 IP 주소로 접속하여 아래와 같이 기본 웹페이지가 나타나면 아파치 웹서버가 정상 작동하는 것이다. 이는 웹 서버의 홈 디렉터리 /var/www/html/에 있는 index.html의 내용이 웹 브라우저에 보이는 것이다.

<http://192.168.0.40/>
혹은 <http://192.168.0.40/index.html>



참고로, /var/log/apache2/ 디렉터리에는 각종 로그 정보가 기록되는데, 웹서버에 문제가 발생했을 때는 다음과 같이 로그 파일을 확인해 볼 수 있다. 장기간의 구동으로 로그 정보가 누적되어 메모리 공간 부족 현상이 나타나 동작하지 않을 수 있다. 이 경우 로그 파일을 삭제하여 공간을 확보할 수 있다. /var/log/apache2 디렉터리는 존재해야 웹서버가 동작하는데 문제가 없음을 유의한다.

```
$ tail /var/log/apache2/access.log          // 접속 로그 파일
$ tail /var/log/apache2/error.log           // 오류 로그 파일
```

15.1.2 CGI 활성화

다음의 명령을 사용하면 Apache2 설치 디렉터리의 구조를 확인할 수 있다. apache2.conf 환경설정 파일은 웹서버의 주 환경설정 파일이다. -available 접미의 디렉터리는 환경설정 파일이나 모듈들을 미리 준비하여 둔 곳이며, -enabled 접미의 디렉터리는 서버가 실행될 때 실제로 반영되는 환경설정 파일이나 모듈들에 대해 심볼릭 링크된 파일이 위치하는 곳이다.

```
$ cd /etc/apache2
$ tree ./
./
├── apache2.conf
├── conf-available
│   ├── charset.conf
│   ├── javascript-common.conf
│   ├── .....
│   └── serve-cgi-bin.conf
└── conf-enabled
    ├── charset.conf -> ../conf-available/charset.conf
    ├── .....
    └── serve-cgi-bin.conf -> ../conf-available/serve-cgi-bin.conf
├── envvars
├── magic
└── mods-available
    ├── access_compat.load
    ├── .....
    ├── cgi.load           // *) CGI 프로그램을 위한 모듈
    ├── .....
    └── xml2enc.load
└── mods-enabled
    ├── access_compat.load -> ../mods-available/access_compat.load
    ├── .....
    └── status.load -> ../mods-available/status.load
```

432 라즈베리파이기반 임베디드시스템용

```
└── ports.conf
└── sites-available
|   ├── 000-default.conf
|   └── default-ssl.conf
└── sites-enabled
    └── 000-default.conf -> ../sites-available/000-default.conf
```

CGI 프로그램을 실행하도록 하려면 환경설정 파일의 수정과 cgi 모듈을 활성화하는 작업을 진행해야 한다.

CGI 홈 디렉터리 변경을 위한 환경설정 파일 편집

./conf-available/ 디렉터리는 준비된 환경설정 파일들이 있으며, ./conf-enabled/ 디렉터리는 활성화될 환경설정 파일에 대한 심볼릭 링크된 파일들이 위치한다.

./conf-available/ 디렉터리에 있는 serve-cgi-bin.conf 파일의 <IfDefine ENABLE_USR_LIB_CGI_BIN> 부분을 다음 명령을 사용하여 살펴보자. 기본적으로 CGI를 위한 홈 디렉터리는 /usr/lib/cgi-bin/으로 설정되어 있다.

```
$ cd /etc/apache2/conf-available/
$ sudo nano serve-cgi-bin.conf

.....
<IfDefine ENABLE_USR_LIB_CGI_BIN>
    ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
    <Directory "/usr/lib/cgi-bin">
        AllowOverride None
        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Require all granted
    </Directory>
....
```

CGI를 위한 디렉터리를 /var/www/html/cgi-bin/으로 변경하려면 밑줄부분을 CGI 프로그램이 있을 경로로 다음과 같이 수정한다. ScriptAlias 줄은 반드시 있어야 한다.

```
ScriptAlias /cgi-bin/ /var/www/html/cgi-bin/
<Directory "/var/www/html/cgi-bin/">
```

위에서 환경 설정하였듯이 CGI 프로그램들이 위치할 디렉터리를 다음의 명령으로 생성한다.

```
$ sudo mkdir /var/www/htm/cgi-bin
```

cgi 모듈 활성화

다음으로 CGI 모듈을 활성화해야 한다. a2enmod 명령은 apache2 모듈 활성화 명령으로, ./mods-available/ 디렉터리에 준비된 모듈에 대한 심볼릭 링크를 ./mods-enabled/ 디렉터리에 생성하여 해당 모듈을 활성화한다.

cgi 프로그램을 위한 준비된 모듈은 ./mods-available/cgi.load이다. 다음 명령으로 cgi 관련 준비된 모듈을 확인할 수 있다.

```
$ ls ./mods-available/cgi*
./mods-available/cgi.load  ./mods-available/cgid.load
./mods-available/cgid.conf
```

다음 명령으로 cgi 관련 모듈이 활성화되도록 설정되었는지를 확인할 수 있다. 해당 모듈 파일에 대한 심볼릭 링크가 설정되어 있지 않음을 관찰할 수 있다.

```
$ ls ./mods-enabled/cgi*
ls: cannot access './mods-enabled/cgi*': No such file or directory
```

다음의 명령을 실행하여 ./mods-available/cgi.load 모듈을 ./mods-enabled/에 심볼릭 링크하여 cgi 모듈을 활성화하도록 등록한다.

```
$ sudo a2enmod cgi
```

다음 명령으로 활성화 모듈 목록의 디렉터리를 확인해 보면 심볼릭 링크 파일이 생성된 것을 확인할 수 있다.

```
$ ls ./mods-enabled/cgi* -l
lrwxrwxrwx 1 root root 26 Aug 16 08:58 ./mods-enabled/cgi.load ->
..../mods-available/cgi.load
```

이제 CGI 프로그램을 위한 환경설정이 완료되었으므로 이를 반영하도록 다음의 명령을 사용하여 apache2의 서비스 재시작하거나 재부팅한다.

```
$ sudo systemctl restart apache2
혹은, $ sudo service apache2 restart
혹은, $ sudo reboot
```

[실습1] hello CGI 테스트

간단한 메시지를 출력하는 hello CGI 프로그램을 구현하여 테스트한다. 이를 위해 hello.c 소스와 helloTest.htm 파일을 아래와 같이 작성한다.

```
$ cat hello.c
//=====
// hello.c
//=====
#include <stdio.h>

int main(void) {

    printf("Content-type: text/html\n\n");
    printf("<H1>Hello, World.....\n\n");

    return 0;
}
```

```
$ cat helloTest.html
<h1>CGI test ..... GET method ...</h1>
<hr><p>
<FORM method=GET action=".//cgi-bin/hello.cgi">
<INPUT type="text" name="value" maxlength="10" size="10">
```

```
<INPUT type="submit" name="button" value="submit">
</FORM>
```

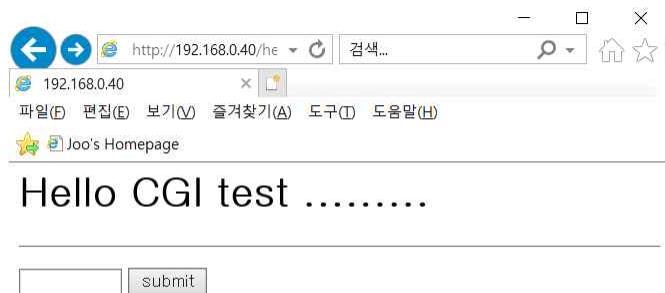
다음 명령으로 C 소스를 컴파일한다. 생성파일은 확장자가 .cgi가 붙도록 컴파일한다.

```
$ sudo gcc -o hello.cgi hello.c
```

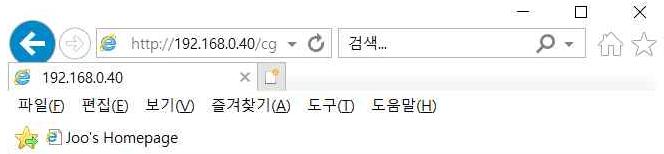
이제 다음 명령을 사용하여 관련 파일들을 웹서비스를 위한 디렉터리로 복사한다.

```
$ sudo cp hello.cgi /var/www/html/cgi-bin/
$ sudo cp helloTest.html /var/www/html/
```

웹브라우저의 주소창에 `http://192.168.0.40/helloTest.html`을 입력하여 접속하면 다음의 화면이 나타난다.



HTML 파일에서 작성한 태그들이 표시된 것뿐이다. submit 버튼을 클릭하면 다음과 같이 CGI 프로그램이 실행된 결과가 나타난다.



15.1.3 HTML FORM 태그

HTML의 FORM 태그를 활용한 최소한의 구성은 다음과 같다. method 속성은 데이터 전송 방식을 규정하는 것으로 GET, POST 등이 있으며, action 속성에는 넘겨받은 데이터를 처리할 CGI 프로그램을 설정하는 부분이다. 최소한 type 속성이 submit인 input 태그를 내부에 두도록 구성해야 CGI 프로그램으로 데이터를 전달 할 수 있다.

```
<FORM method=GET action=".//cgi-bin/formtag.cgi">  
.....  
<input type=submit name=transmit value=transmit>  
</FORM>
```

HTML의 FORM 태그 내에서 사용가능한 태그로 input 태그가 대표적이며, type 속성에 설정하는 값에 따라 다음과 같이 다양한 유형들이 제공된다.

텍스트

```
<label>text</label>  
<input type=text name=name maxlength=10>  
<p>
```

암호

```
<label>password</label>  
<input type=password name=pw maxlength=10>  
<p>
```

숫자

수치만 입력 받는다.

```
<label>number</label>
<input type=number name=num maxlength=2>
<p>
```

라디오버튼

동일한 name 속성을 갖는 항목들 중에서 오직 하나의 항목만을 선택할 수 있다. 기본 선택항목은 checked 속성을 사용한다.

```
<label>radio</label>
<input type=radio name=led value=1 checked> ON
<input type=radio name=led value=0> OFF
<p>
```

체크박스

동일한 name 속성을 갖는 항목들 중에서 여러 항목을 선택할 수 있다. 배열의 형태로 전달된다.

```
<label>checkbox</label>
<input type=checkbox name=ledArray value=8 checked> LED3
<input type=checkbox name=ledArray value=4> LED2
<input type=checkbox name=ledArray value=2> LED1
<input type=checkbox name=ledArray value=0 checked> LED0
<p>
```

리스트

select 태그 내에서 option 태그로 항목을 나열한다. 디폴트 선택항목의 표시는 selected 속성으로 표시한다.

```
<label>list</label>
<select name=year>
    <option value=1> 2011 </option>
    <option value=2> 2012 </option>
    <option value=3 selected> 2013 </option>
```

```
<option value=4> 2014 </option>
<option value=5> 2015 </option>
</select>
<p>
```

버튼

type 속성의 값이 button이면 일반 버튼을, submit이면 FORM 태그내의 내용을 전송하는 버튼, reset이면 FORM 태그 내에서 설정된 내용을 초기화한다.

```
<label>button</label>
<input type=button name=ok value=ok>
<p>

<label>submit</label>
<input type=submit name=transmit value=transmit>
<p>

<label>reset</label>
<input type=reset name=cancel value=cancel>
<p>
```

텍스트영역

추가로 여러 라인의 텍스트를 입력할 수 있는 다음과 같은 textarea 태그가 제공된다.

```
<label>textarea</label>
<textarea name=memo rows=10 cols=40> memo... </textarea>
<p>
```

[실습2] FORM 태그 테스트

앞에서 살펴본 태그들을 시험해 보기위한 것이다. 일단 이 소스에서 method 속성과 action 속성에 설정된 값은 크게 의미를 두지말자.

```
$ nano formTag.html
```

```
<FORM method=GET action=".//cgi-bin/formtag.cgi">
```

```
<label>text</label>
<input type=text name=name maxlength=10>
<p>

<label>password</label>
<input type=password name=pw maxlength=10>
<p>

<label>number</label>
<input type=number name=num maxlength=2>
<p>

<label>radio</label>
<input type=radio name=led value=1 checked> ON
<input type=radio name=led value=0> OFF
<p>

<label>checkbox</label>
<input type=checkbox name=ledArray value=8 checked> LED3
<input type=checkbox name=ledArray value=4> LED2
<input type=checkbox name=ledArray value=2> LED1
<input type=checkbox name=ledArray value=0 checked> LED0
<p>

<label>list</label>
<select name=year>
    <option value=1> 2011 </option>
    <option value=2> 2012 </option>
    <option value=3 selected> 2013 </option>
    <option value=4> 2014 </option>
    <option value=5> 2015 </option>
</select>
<p>

<label>textarea</label>
<textarea name=memo rows=10 cols=40> memo... </textarea>
<p>

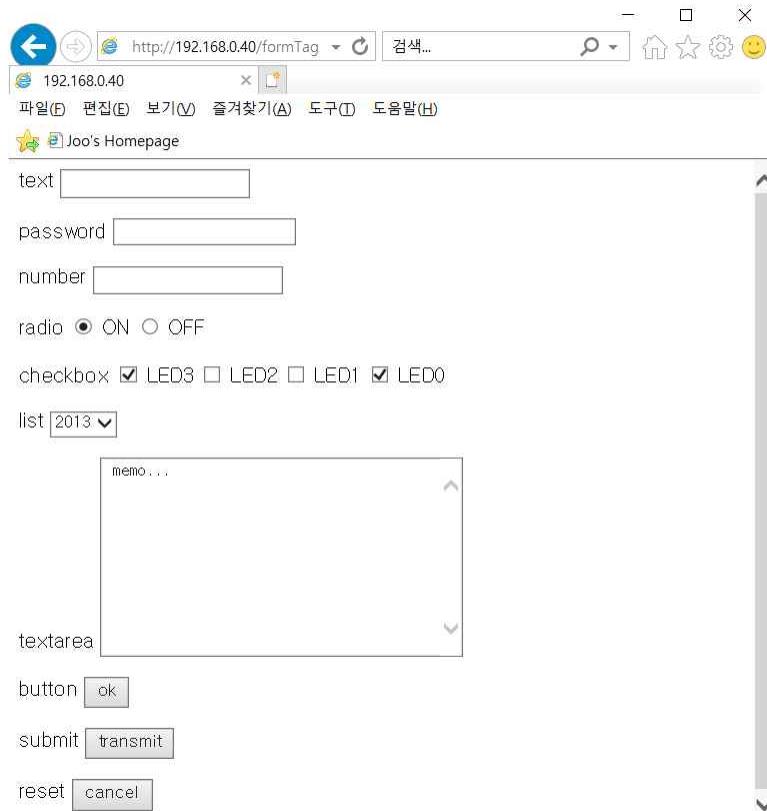
<label>button</label>
<input type=button name=ok value=ok>
<p>
```

```
<label>submit</label>
<input type=submit name=transmit value=transmit>
<p>

<label>reset</label>
<input type=reset name=cancel value=cancel>
<p>

</FORM>
```

웹 브라우저의 주소창에서 <http://192.168.0.40/formTag.html>를 입력하여 접속하면 다음의 화면이 나타난다.



POST 방식의 경우, 위 소스의 첫 부분인 FORM 태그 내의 method 속성을 POST로 다음과 같이 대체한다.

```
<FORM method=POST action="./cgi-bin/formtag.cgi">
```

15.2 CGI 프로그래밍

C 언어를 사용하여 CGI 프로그램을 구현하는 것을 살펴본다. CGI 프로그램에 데이터를 전달하는 방식은 GET 방식과 POST 방식의 두 가지가 있다.

GET 방식은 전송 가능한 정보의 길이가 한정되며, 전송 정보는 웹 브라우저의 주소창에 URL과 함께 인자 형태로 전송됨으로 인해 전송 정보가 드러나므로 보안에 취약하다 할 수 있다. 인자는 이름=값의 쌍으로 이루어지고 각 인자간은 &로 구분되어 전송된다. 반면, POST 방식은 전송 가능한 정보의 길이에 제한이 없으며, 전송 정보는 웹 브라우저의 주소창에 나타나지 않으므로 GET 방식보다 보안상 우위라 할 수 있으며, 전송정보는 헤더 내에 담겨 전송된다.

15.2.1 GET 방식의 CGI

CGI 관련하여 환경변수들은 정리하면 다음 표와 같다.

환경변수	기능
REMOTE_ADDR	접속자의 IP 주소
REQUEST_METHOD	GET, POST 등 요청 방법
QUERY_STRING	FORM 태그 요청 문자열
CONTENT_LENGTH	FORM 태그 요청 문자열의 길이
HTTP_USER_AGENT	접속자의 웹브라우저 정보
SERVER_SOFTWARE	웹서버 정보

또한 시스템의 환경변수의 값을 얻는 함수로 getenv() 함수가 제공된다. 이를 사용

하여 접속자의 IP 주소와 요청 문자열을 얻는 코드는 다음과 같다.

```
printf("Your IP Address : %s<br>\n", getenv("REMOTE_ADDR"));
printf("QUERY_STRING : %s<br>\n", getenv("QUERY_STRING"));
```

QUERY_STRING으로 전달된 내용은 INPUT 태그의 유형에 따른 개별 값을 추출하기 위해서는 토큰으로 파싱할 필요가 있다.

[실습3] GET 방식의 CGI

GET 전달방식으로의 CGI 프로그램을 구현하고 이를 테스트 한다.

```
$ nano testGet.html
<h1>CGI Test, by GET method</h1>
<hr><p>

<FORM method=GET action=".//cgi-bin/testGet.cgi">
<label>text</label>
<input type=text name=name maxlength=10>
<p>

<label>number</label>
<input type=number name=num maxlength=2>
<p>

<label>radio</label>
<input type=radio name=led value=1 checked> ON
<input type=radio name=led value=0> OFF
<p>

<label>checkbox</label>
<input type=checkbox name=ledArray value=8 checked> LED3
<input type=checkbox name=ledArray value=4> LED2
<input type=checkbox name=ledArray value=2> LED1
<input type=checkbox name=ledArray value=0 checked> LED0
<p>

<input type=submit name=transmit value=transmit>
<p>
```

</FORM>

```
$ nano testGet.c
//=====
// testGet.c
//      CGI Test, by GET method
//=====

#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("Content-type: text/html\n\n");

    printf("<html>\n\n");

    printf("<head>\n");
    printf("<title>CGI Test, by GET</title>\n");
    printf("</head>\n\n");

    printf("<body>\n");
    printf("<h1>CGI Test, by GET</h1>\n");
    printf("<hr><p>\n");

    printf("Your IP Address : %s<br>\n", getenv("REMOTE_ADDR"));
    printf("QUERY_STRING : %s<br>\n", getenv("QUERY_STRING"));

    printf("</body>\n\n");
    printf("</html>\n");

    return 0;
}
```

다음 명령과 같이 소스 프로그램을 컴파일하여 cgi 프로그램을 생성한다.

```
$ gcc -o testGet.cgi testGet.c
```

다음과 같이 생성된 파일들을 서비스를 위해 웹서버의 홈 디렉터리 내에 복사한다.

444 라즈베리파이기반 임베디드시스템응용

```
$ cp testGet.html /var/www/html/  
$ cp testGet.cgi /var/www/html/cgi-bin/
```

웹브라우저의 주소창에 <http://192.168.0.40/testGet.html> 을 입력하여 접속하면 다음의 화면이 나오고, 각 항목을 적절히 입력 및 선택하고 transmit 버튼을 클릭 한다.



CGI 프로그램의 실행 결과가 다음과 같이 웹 브라우저에 표시된다.



15.2.2 POST 방식의 CGI

GET 방식과 달리 FORM 태그 요청 문자열 즉, QUERY_STRING으로 전달된 내용

을 읽기 위해서는 다음과 같은 read() 함수를 사용한다.

```
read(0, buf, 1024)
```

위 함수를 통해 읽은 내용은 buf에 위치하고, INPUT 태그의 유형에 따른 개별 값을 추출하기 위해서는 토큰으로 파싱할 필요가 있다.

[실습4] POST 방식의 CGI

POST 전달방식으로의 CGI 프로그램을 구현하고 이를 테스트 한다.

```
$ nano testPost.html
<h1>CGI Test, by POST method</h1>
<hr><p>

<FORM method=POST action=".//cgi-bin/testPost.cgi">
<label>text</label>
<input type=text name=name maxlength=10>
<p>

<label>number</label>
<input type=number name=num maxlength=2>
<p>

<label>radio</label>
<input type=radio name=led value=1 checked> ON
<input type=radio name=led value=0> OFF
<p>

<label>checkbox</label>
<input type=checkbox name=ledArray value=8 checked> LED3
<input type=checkbox name=ledArray value=4> LED2
<input type=checkbox name=ledArray value=2> LED1
<input type=checkbox name=ledArray value=0 checked> LED0
<p>

<input type=submit name=transmit value=transmit>
<p>
</FORM>
```

```
$ nano testPost.c
//=====
// testPost.c
//      CGI Test, by POST method
//=====

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    char buf[1024];

    printf("Content-type: text/html\n\n");

    printf("<html>\n\n");
    printf("<head>\n");
    printf("<title>CGI Test, by POST</title>\n");
    printf("</head>\n\n");

    printf("<body>\n");
    printf("<h1>CGI Test, by POST</h1>\n");
    printf("<hr><p>\n");

    printf("Your IP Address : %s<br>\n", getenv("REMOTE_ADDR"));

    memset(buf, 0x00, 1024);
    while(read(0, buf, 1024) > 0) {
        printf("POST STRING : %s<p>\n", buf);
    }

    printf("</body>\n\n");
    printf("</html>\n");

    return 0;
}
```

다음 명령과 같이 소스 프로그램을 컴파일고, 관련파일들을 웹서버의 홈 디렉터리

로 복사한다.

```
$ gcc -o testPost.cgi testPost.c
$ cp testPost.html /var/www/html/
$ cp testPost.cgi /var/www/html/cgi-bin/
```

웹브라우저의 주소창에 <http://192.168.0.40/testPost.html> 을 입력하여 접속하면 다음의 화면이 나오고, 각 항목을 적절히 입력 및 선택하고 transmit 버튼을 클릭 한다.



CGI 프로그램의 실행 결과가 다음과 같이 웹 브라우저에 표시된다.



15.3 CGI에 의한 LED 제어

라즈베리파이의 디바이스를 제어하기 위해서는 WiringPi 라이브러리가 접근하는 디바이스 파일 /dev/gpiomem을 웹 클라이언트가 접속하여 접근할 수 있도록 설정해야 한다.

현재의 접근 설정을 살펴보면 다음과 같다.

```
$ sudo ls -l /dev/gpiomem
crw-rw---- 1 root gpio 243, 0 Jan 15 10:44 gpiomem
```

따라서 다음과 같이 접근 속성을 웹을 통해 누구나 접근할 수 있도록 667 접근 속성으로 변경한다. 아래와 같이 디바이스 파일의 접근속성을 변경하였더라도 재부팅하는 경우 원래의 접근속성으로 되돌아가므로 이를 유의한다.

```
$ sudo chmod 667 /dev/gpiomem
crw-rw-rwx 1 root gpio 243, 0 Jan 15 10:44 gpiomem
```

아래에서 대표적인 출력 디바이스인 LED에 대해서 GET 혹은 POST 방식의 CGI를 통해 제어해 본다.

15.3.1 GET 방식의 CGI에 의한 LED 제어

QUERY_STRING으로 전달된 내용을 INPUT 태그의 유형에 따른 개별 값을 추출하기 위한 구조체 및 함수를 다음과 같이 정의한다.

```
typedef struct {
    char name[32];
    char val[32];
} entry;

void getToken(char *word, char *qStr, char deli);
```

```

// get token
void getToken(char *word, char *str, char deli) {
    int i, new_i;

    // extract token
    for(i=0; ((str[i]) && (str[i] != deli)); i++)
        word[i] = str[i];

    word[i] = '\0';

    if(str[i])           // skip delimiter character
        i++;

    // make a string of remaining str
    new_i = 0;
    while(str[new_i++] = str[i++])
        ;
}

```

위 함수를 다음과 같이 구분자 =, &를 활용하여 호출하여 원하는 토큰을 추출한다.

```

char *qStr;
entry item;

qStr = (char *)getenv("QUERY_STRING");
getToken(item.name, qStr, '=');
getToken(item.val, qStr, '&');

```

[실습5] GET 방식에 의한 LED 제어

LED를 제어할 수 있는 GET 전달 방식으로의 CGI 프로그램을 구현하고 이를 테스트 한다.

```

$ nano ledGet.html
<h1>LED control..... GET ...</h1>
<hr><p>
<FORM method=GET action=".//cgi-bin/ledGet.cgi">

```

```
<label>LED : </label>
<input type=text name=name maxlength=10>
<p>

<input type=submit name=transmit value=transmit>
<p>
</FORM>
```

다음 소스의 볼드체 부분은 LED 디바이스의 제어와 관련된 부분이다. getToken() 함수는 문자열에서 구분자를 이용하여 토큰을 추출하는 함수이다.

```
$ nano ledGet.c
//=====
// ledGet.c CGI
//      LED control, by GET method
//=====

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <wiringPi.h>      /////
#define P_LED 1           ///// BCM_GPIO #18

typedef struct {
    char name[32];
    char val[32];
} entry;

void getToken(char *word, char *qStr, char deli);

int main(void) {
    char *qStr;
    entry item;

    if(wiringPiSetup() == -1)          ///// wiringPi pin #
        return 1;
```

```

pinMode(P_LED, OUTPUT);           /////

printf("Content-type: text/html\n\n");
printf("<html>\n\n");
printf("<head>\n");
printf("<title>LED CGI by GET...</title>\n");
printf("</head>\n\n");
printf("<body>\n");
printf("<h1>LED CGI by GET...</h1>\n");
printf("<hr><p>\n");

printf("Your IP Address : %s<br>\n", getenv("REMOTE_ADDR"));

printf("QUERY_STRING : %s<br><p>\n", getenv("QUERY_STRING"));
qStr = (char *)getenv("QUERY_STRING");
getToken(item.name, qStr, '=');
getToken(item.val, qStr, '&');

printf("Extracted token : %s = %s<br>", item.name, item.val);

//// LED control...
if(item.val[0] == '0') {
    digitalWrite(P_LED, LOW);
    printf("==> Led OFF.....<br>");
}
else if(item.val[0] == '1') {
    digitalWrite(P_LED, HIGH);
    printf("==> Led ON.....<br>");
}
else
    printf("==> Wrong data.....<br>");

printf("</body>\n\n");
printf("</html>\n");

return 0;
}

// get token
void getToken(char *word, char *str, char deli) {
    int i, new_i;

```

452 라즈베리파이기반 임베디드시스템응용

```
// extract token
for(i=0; ((str[i]) && (str[i] != deli)); i++)
    word[i] = str[i];

word[i] = '\0';

if(str[i])           // skip delimiter character
    i++;

// make a string of remaining str
new_i = 0;
while(str[new_i++] = str[i++])
    ;
}
```

다음의 명령과 같이 -lwiringPi를 포함하여 컴파일한다.

```
$ gcc -o ledGet.cgi ledGet.c -lwiringPi
```

준비된 HTML 문서와 CGI 프로그램을 합당한 디렉터리로 복사한다.

웹 브라우저에서 <http://192.168.0.40/ledGet.html>로 접속하면 다음의 화면이 나온다. 입력 문자열중 첫 문자는 LED 디바이스를 제어하는 문자이므로 ON하고자 하는 경우 '1'을 OFF하고자하는 경우 '0'문자로 시작하는 문자열을 입력하고 transmit 버튼을 클릭한다.



오류 메시지가 나오는 경우는 /dev/gpiomem의 접근 속성을 변경하여 주고 다시 시도한다. LED를 ON 하는 경우의 실행 결과 화면은 다음과 같으며, 라즈베리 보드 상의 LED는 ON 된 것을 확인할 수 있다.



15.3.2 POST 방식의 CGI에 의한 LED 제어

[실습6] POST 방식에 의한 LED 제어

LED를 제어할 수 있는 POST 전달 방식으로의 CGI 프로그램을 구현하고 이를 테스트 한다.

```
$ nano ledPost.html
<h1>LED Control, by POST...</h1>
<hr><p>

<FORM method=POST action=".//cgi-bin/ledPost.cgi">

<label>LED : </label>
<input type=text name=name maxlength=10>
<p>

<input type=submit name=transmit value=transmit>
<p>

</FORM>
```

454 라즈베리파이 기본 임베디드 시스템 활용

```
$ nano ledPost.c
//=====
// ledPost.c CGI
//      LED control, by POST method
//=====

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <wiringPi.h>           /////
#define P_LED 1                  ///// BCM_GPIO #18

typedef struct {
    char name[32];
    char val[32];
} entry;

void getToken(char *word, char *qStr, char deli);

int main(void) {
    char buf[1024];
    char *qStr;
    entry item;

    if(wiringPiSetup() == -1)          ///// wiringPi pin #
        return 1;

    pinMode(P_LED, OUTPUT);          /////

    printf("Content-type: text/html\n\n");
    printf("<html>\n\n");
    printf("<head>\n");
    printf("<title>LED CGI by POST...</title>\n");
    printf("</head>\n\n");
    printf("<body>\n");
    printf("<h1>LED CGI by POST...</h1>\n");
    printf("<hr><p>\n");

    printf("Your IP Address : %s<br>\n", getenv("REMOTE_ADDR"));
```

```

        memset(buf, 0x00, 1024);
        while(read(0, buf, 1024) > 0) {
            printf("POST STRING : %s<p>\n", buf);
        }

        qStr = buf;
        getToken(item.name, qStr, '=');
        getToken(item.val, qStr, '&');
        printf("Extracted token : %s = %s<br>", item.name, item.val);

        //// LED control....
        if(item.val[0] == '0') {
            digitalWrite(P_LED, LOW);
            printf("==> Led OFF.....<br>");
        }
        else if(item.val[0] == '1') {
            digitalWrite(P_LED, HIGH);
            printf("==> Led ON.....<br>");
        }
        else
            printf("==> Wrong data.....<br>");

        printf("</body>\n\n");
        printf("</html>\n");

        return 0;
    }

// get token
void getToken(char *word, char *str, char deli) {
    int i, new_i;

    // extract token
    for(i=0; ((str[i]) && (str[i] != deli)); i++)
        word[i] = str[i];
    word[i] = '\0';

    if(str[i])           // skip delimiter character
        i++;

    // make a string of remaining str
    new_i = 0;
}

```

456 라즈베리파이기반 임베디드시스템응용

```
while(str[new_i++] = str[i++])  
;  
}
```

다음의 명령으로 컴파일한다.

```
$ gcc -o ledPost.cgi ledPost.c -lwiringPi
```

준비된 HTML 문서와 CGI 프로그램을 합당한 디렉터리로 복사한다.

웹 브라우저에서 <http://192.168.0.40/ledPost.html>로 접속하면 다음의 화면이 나온다. 입력 문자열중 첫 문자는 LED 디바이스를 제어하는 문자이므로 ON 하고자 하는 경우 '1'을, OFF 하고자 하는 경우 '0' 문자로 시작하는 문자열을 입력한다.



LED를 OFF 하는 경우의 실행 결과 화면은 다음과 같으며, 라즈베리 보드 상의 LED는 OFF 된 것을 확인할 수 있다.



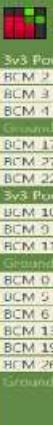
LED CGI by POST...

Your IP Address: 192.168.0.10
POST STRING : name=0....&transmit=transmit

Extracted token : name = 0....
=> Led OFF.....

참고자료

- [1] APM 설치 <http://fishpoint.tistory.com/2230>
- [2] 웹서버 홈 디렉터리 변경 <http://karl27.tistory.com/8>
- [3] CGI 언어 비교 <http://jokergt.tistory.com/165>
- [4] CGI 언어 비교
<http://tip.daum.net/question/328839?q=%EC%95%84%ED%8C%8C%EC%B9%98+%EC%9B%B9%EC%84%9C%EB%B2%84+CGI>
- [5] HTML의 FORM 태그 <http://kimjungkwon.com/91>
- [6] HTML의 FORM 태그 <http://kimhyun2017.tistory.com/92>
- [7] CGI 설정 <http://tribal1012.tistory.com/140>
- [8] CGI 설정
<http://blog.naver.com/PostView.nhn?blogId=icharley&logNo=221090197788>
- [9] CGI 설정
<https://tasdikrahman.me/2015/09/30/Running-CGI-Scripts-on-Apache2-Ubuntu/>
- [10] GET과 POST 방식의 차이 <http://devbox.tistory.com/146>
- [11] 아파치 CGI 문서
<http://httpd.apache.org/docs/2.4/en/howto/cgi.html>
- [12] 임베디드응용 및 실습
https://cms3.koreatech.ac.kr/sites/joo/IFC412/IFC412_14.pdf



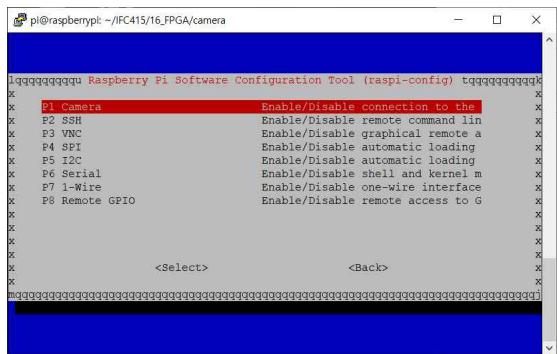
제16장 FPGA 디바이스 제어

Achro-EM 키트에서 카메라 모듈을 사용하여 영상을 확보하거나, 동영상 스트리밍 서비스에 대해 살펴본다. 그리고 자체적인 주소 체계를 정의하고 있는 FPGA 입출력 디바이스를 제어하는 방법에 대해 살펴본다. 또한 QT를 활용하여 GUI에서 FPGA 입출력 디바이스를 제어하는 방법에 대해 살펴본다.

16.1 카메라 모듈

카메라 모듈을 시험하려면 라즈베리파이 환경설정 파일에서 카메라를 활성화하여야 한다. 다음의 명령을 통해 카메라를 활성화한다.

```
$ sudo raspi-config
```



16.1.1 정지영상 및 동영상

라즈베리파이에서는 기본적으로 정지영상과 동영상을 캡춰하기 위한 명령을 제공하고 있다.

카메라 영상 캡쳐

460 라즈베리파이 기본 임베디드 시스템 활용

정지 영상을 캡춰하기 위해서는 raspistill 명령을, 동영상을 캡춰하기 위해서는 raspivid 명령을 사용한다. 다음 명령과 같이 사용하면 관련 옵션들에 대한 도움말을 확인할 수 있다.

```
$ raspistill | less  
$ raspivid | less
```

이들 명령에서 사용되는 기본적인 옵션들을 정리하면 다음 표와 같다.

옵션	기능
-rot	화면 회전각 설정
-t	ms단위 후 캡춰
-k	키보드의 x키나 enter키의 입력때 캡춰
-o	저장할 파일 이름, 확장자 포함
-q	화질 선택, 0~100중 설정
-n	미리보기 화면 OFF, 기본적으로 미리보기 5초 설정됨
-w	영상의 가로 폭, 픽셀단위
-h	영상의 세로 폭, 픽셀단위
-md	화면길이와 해상도 선택모드 지정, 0~7
-e	인코더 변경, jpg, bmp, gif, png, 기본 jpg임
-ss	셔터 스피드 설정, 최대 6000ms

jpg 포맷으로 이미지 저장할 때 다음의 명령을 사용하면, 약 5초 동안 동영상을 디스플레이하고 마지막 순간의 정지영상을 해당 파일로 저장한다.

```
$ cd ./cam_test  
$ raspistill -v -o image.jpg  
$ ls  
image.jpg
```

라즈베리파이에서 기본 동영상의 포맷은 h264이다. 동영상을 일정시간동안 캡춰하는 경우는 다음의 명령을 사용하면 초당 30프레임으로 5초(5000msec)동안의 동영상을 video.h264 파일로 저장한다.

```
$ raspivid -o video.h264 -fps 30 -t 5000
```

동영상 포맷 변경

라즈베리파이 카메라 프로그램은 H264 포맷을 사용하고 있기에, 경우에 따라서 일 반적인 동영상파일 포맷으로 변환할 필요가 있을 수 있다. 생성된 h264 포맷의 동 영상에 대한 포맷을 변경하는 툴 혹은 플레이어를 설치하려면 다음과 같이 관련 패키지를 설치하여야 한다.

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install mplayer netcat ffmpeg gpac
```

다음의 명령은 ffmpeg을 이용하여 H264 포맷을 mkv포맷으로 변환시키는 예다.

```
$ ffmpeg -r 30 -i video.h264 -vcodec copy video.mkv
```

다음의 명령은 mp4 파일 포맷으로 변환하는 명령이다.

```
$ MP4Box -add video.h264 video.mp4
```

mplayer를 사용해 변환된 파일을 보고자 할 때는 다음과 같은 명령을 사용할 수 있다.

```
$ mplayer -zoom -x 200 -y 200 video.mp4
```

16.1.2 동영상 스트리밍

라즈베리파이 카메라모듈을 이용한 웹스트리밍 서비스를 위해 mjpg-streamer가 최적화되어있다. 속도가 느린 편이지만 현장 상황을 웹을 이용하여 모니터링하기에 는 무리가 없다.

우선 mjpg-streamer 컴파일을 위해 다음 명령을 사용하여 영상관련 라이브러리 및 cmake 패키지를 설치한다.

```
$ sudo apt-get update  
$ sudo apt-get upgrade  
$ sudo apt-get install git cmake libjpeg8-dev imagemagick -y
```

이제 mjpg-streamer 소스를 다음 명령을 사용하여 다운로드한다.

```
$ pwd  
/home/pi/IFC415/16_FPGA/camera  
$ git clone https://github.com/liamfraser/mjpg-streamer
```

위의 명령은 현 작업디렉터리에 ./mjpg-streamer/ 디렉터리로 mjpg-streamer 소스를 다운로드 한다. 다음 명령으로 소스 내역을 살펴볼 수 있다. www 디렉터리는 웹브라우저를 통해 접속할 때의 페이지들이 위치한다.

```
$ cd ./mjpg-streamer/mjpg-streamer-experimental  
$ ls  
CHANGELOG  mjpg_streamer.c  README          start.sh  utils.h  
LICENSE    mjpg_streamer.h  README-UNSTABLE  TODO      www  
Makefile   plugins        scripts         utils.c
```

이제 다운로드 받은 mjpg-streamer를 컴파일한다.

```
$ make clean  
$ make  
$ ls  
CHANGELOG          mjpg_streamer  output_http.so  start.sh  www  
input_raspicam.so  mjpg_streamer.c  plugins       TODO  
input_uvc.so       mjpg_streamer.h  README        utils.c  
LICENSE           mjpg_streamer.o  README-UNSTABLE  utils.h  
Makefile          output_file.so  scripts      utils.o
```

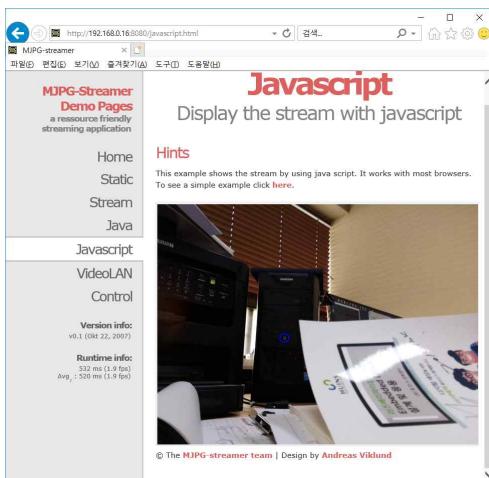
앞의 과정들을 통해 mjpg-streamer를 실행할 수 있는 상황이 되었다. 마지막으로 스트리밍 서비스를 할 수 있도록 스크립트 파일 mjpg.sh을 다음과 같이 작성한다. 아래에서 끝의 두 라인은 하나의 라인으로 작성한다.

```
$ pwd
/home/pi/IFC415/16_FPGA/camera
$ sudo nano mjpg.sh
export STREAMER_PATH=$HOME/IFC415/16_FPGA/camera/mjpg-streamer/mjpg-streamer-experimental
export LD_LIBRARY_PATH=$STREAMER_PATH
$STREAMER_PATH/mjpg_streamer -i "input_raspicam.so -d 200" -o "output_http.so
-w $STREAMER_PATH/www"
```

카메라 스트리밍 서비스를 위해 다음 명령으로 스크립트 파일을 실행한다. 이로써 카메라 스트리밍 서비스가 시작된다. 물론 웹서버는 실행 중에 있어야 한다.

```
$ sh mjpg.sh
```

카메라 스트리밍 서비스를 받으려면 Windows 환경의 웹브라우저 주소창에 <http://192.168.0.40:8080>를 입력하여 접속한다. mjpg-streamer는 기본적으로 포트번호 8080을 사용한다. 다음 화면과 같이 나타나면 좌측 메뉴항목에서 JavaScript를 선택하여 카메라 영상의 웹스트리밍을 확인할 수 있다.



웹 페이지에서 동영상 스트리밍 서비스의 확인이 완료되면 웹 브라우저를 닫고, 터미널 창에서 Ctrl-c를 눌러 동영상 스트리밍을 종료한다.

16.2 FPGA 디바이스 제어

(주)휴인스에서 제작한 Achro-EM 임베디드 키트 상의 디바이스를 제어하는 것에 관해 살펴본다.

16.2.1 FPGA 디바이스

Achro-EM 키트의 FPGA 디바이스의 배치 외관은 다음과 같다.



키트의 하단에 SW7 딥 스위치가 있으며 기본적으로 하향으로 위치시킨다. 이 경우 독자적으로 디바이스들을 구동하도록 설계되어 있다.

제시된 소스를 사용하여 각 디바이스를 제어하는 실습을 진행할 때는 SW7 스위치 #1(좌측)을 ON(상향)으로 설정하여 사용자 모드로 전환해야 한다. 참고로 프로그램에서 사용자 모드로 전환할 때는 Demo Register에 High 신호를 출력하면 된다.

이 키트에서 FPGA 디바이스들은 자체의 주소체계를 가지고 있다. 각 디바이스에 대한 어드레스 맵과 제시된 소스에서 각 디바이스 드라이버의 주번호를 정리하면 다음과 같다.

순	장치	어드레스	Node	Major
1	LED	0x016	/dev/fpga_led	260
2	FND	0x004	/dev/fpga_fnd	261
3	Dot Matrix	0x210	/dev/fpga_dot	262
4	Text LCD	0x090	/dev/fpga_text_lcd	263
5	Buzzer	0x070	/dev/fpga_buzzer	264
6	Push Switch	0x050	/dev/fpga_push_switch	265
7	Dip Switch	0x000	/dev/fpga_dip_switch	266
8	Step Motor	0x00C	/dev/fpga_step_motor	267
EN	Demo Register	0x300	N/A	N/A

16.2.2 FPGA 인터페이스 모듈

우선 제시된 소스들은 커널 모듈과 응용프로그램으로 구성되어 있다. 따라서 각 디바이스의 모듈을 컴파일하려면 실행될 커널 버전과 동일한 커널 소스가 있어야 하며, 커널 소스로의 경로지정 또한 바르게 되어 있어야 한다.

linux-rpi-4.9.y 커널 버전은 linux-rpi-4.9.y 라고 가정하고 진행한다. 이러한 상황에서 linux-rpi-4.9.y 버전의 소스 디렉터리와 커널 모듈 라이브러리 디렉터리가 잘 설치되어 있어야 한다.

```
$ ls /usr/src -l
total 12
lrwxrwxrwx 1 root root 18 Aug 1 11:51 linux -> ./linux-rpi-4.9.y/
drwxr-xr-x 24 pi pi 4096 Oct 22 2018 linux-rpi-4.13.34
drwxr-xr-x 25 pi pi 4096 Aug 2 12:17 linux-rpi-4.9.y
drwxr-xr-x 3 root root 4096 Nov 13 2018 sense-hat
```

위와 같은 경우 커널 소스는 linux-rpi-4.9.y이고, 커널 소스의 홈디렉터리 경로는 /usr/src/linux이다. 따라서 제시된 각 소스의 Makefile의 KDIR 환경변수에 다음과 같이 설정하여야 제대로 컴파일 됨을 유의한다.

KDIR := /usr/src/linux/

아래의 실습 진행은 가상머신이 아닌 라즈베리파이 보드에서 진행한다.

FPGA 인터페이스 모듈

FPGA 모듈과 라즈베리파이보드의 GPIO 핀간 매핑을 위한 디바이스 드라이버 소스이다. 각 FPGA 디바이스를 제어하기 위해서 우선적으로 이 디바이스 드라이버가 커널에 링크되어 있어야 한다.

```
$ cat fpga_interface_driver.c
/* FPGA LED Ioremap Control
FILE : fpga_fpga_itf_driver.c*/

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/delay.h>
#include <linux/interrupt.h>
#include <linux/gpio.h>
#include <asm-generic/bitsperlong.h>

#define CTRL_nWE      0
#define CTRL_nOE      1
#define CTRL_nCS      2

static struct gpio iom_fpga_address[] = { // A1 ~ A11, A0=LOW
    /*{ 10, GPIOF_OUT_INIT_LOW, "ADDRESS 00" },*/
    { 11, GPIOF_OUT_INIT_LOW, "ADDRESS 01" },
    { 12, GPIOF_OUT_INIT_LOW, "ADDRESS 02" },
    { 13, GPIOF_OUT_INIT_LOW, "ADDRESS 03" },
    { 14, GPIOF_OUT_INIT_LOW, "ADDRESS 04" },
    { 15, GPIOF_OUT_INIT_LOW, "ADDRESS 05" },
    { 16, GPIOF_OUT_INIT_LOW, "ADDRESS 06" },
    { 17, GPIOF_OUT_INIT_LOW, "ADDRESS 07" },
    { 18, GPIOF_OUT_INIT_LOW, "ADDRESS 08" },
    { 19, GPIOF_OUT_INIT_LOW, "ADDRESS 09" },
    { 20, GPIOF_OUT_INIT_LOW, "ADDRESS 10" },
    { 21, GPIOF_OUT_INIT_LOW, "ADDRESS 11" },
};

static struct gpio iom_fpga_data[] = {
    { 2, GPIOF_OUT_INIT_LOW, "DATA 0" },
    { 3, GPIOF_OUT_INIT_LOW, "DATA 1" },
    { 4, GPIOF_OUT_INIT_LOW, "DATA 2" },
```

```

{ 5, GPIOF_OUT_INIT_LOW, "DATA 3" },
{ 6, GPIOF_OUT_INIT_LOW, "DATA 4" },
{ 7, GPIOF_OUT_INIT_LOW, "DATA 5" },
{ 8, GPIOF_OUT_INIT_LOW, "DATA 6" },
{ 9, GPIOF_OUT_INIT_LOW, "DATA 7" },
};

static struct gpio iom_fpga_control[] = {
{ 22, GPIOF_OUT_INIT_LOW, "nWE" },
{ 23, GPIOF_OUT_INIT_LOW, "nOE" },
{ 25, GPIOF_OUT_INIT_LOW, "nCS" },
};

static void iom_fpga_itf_set_default(void) {
    int i = 0;

    gpio_set_value(10, 0); // A0: always set to LOW

    for (i=0; i<ARRAY_SIZE(iom_fpga_address); i++) {
        gpio_set_value(iom_fpga_address[i].gpio, 0);
    }

    for (i=0; i<ARRAY_SIZE(iom_fpga_data); i++) {
        gpio_set_value(iom_fpga_data[i].gpio, 0);
    }

    for (i=0; i<ARRAY_SIZE(iom_fpga_control); i++) {
        gpio_set_value(iom_fpga_control[i].gpio, 1);
    }
}

static int iom_fpga_itf_open(void) {
    int ret = 0;

    ret = gpio_request_array(iom_fpga_address,
                           ARRAY_SIZE(iom_fpga_address));
    if (ret) {
        printk(KERN_ERR "Unable to request address GPIOs: %d\n",
              ret);
        return ret;
    }
}

```

```

ret = gpio_request_array(iom_fpga_data,
                        ARRAY_SIZE(iom_fpga_data));
if (ret) {
    printk(KERN_ERR "Unable to request data GPIOs: %d\n", ret);
    return ret;
}

ret = gpio_request_array(iom_fpga_control,
                        ARRAY_SIZE(iom_fpga_control));
if (ret) {
    printk(KERN_ERR "Unable to request control GPIOs: %d\n",
           ret);
    return ret;
}

iom_fpga_itf_set_default();
return ret;
}

static int iom_fpga_itf_release(void) {
    iom_fpga_itf_set_default();

    gpio_free_array(iom_fpga_address,
                   ARRAY_SIZE(iom_fpga_address));
    gpio_free_array(iom_fpga_data, ARRAY_SIZE(iom_fpga_data));
    gpio_free_array(iom_fpga_control, ARRAY_SIZE(iom_fpga_control));

    return 0;
}

ssize_t iom_fpga_itf_write(unsigned int addr, unsigned char value) {
    size_t length = 1;
    int i = 0;

    printk("FPGA WRITE: address = 0x%x, data = 0x%x \n", addr,
          value);

    for (i=0; i<ARRAY_SIZE(iom_fpga_address); i++) {
        gpio_set_value(iom_fpga_address[i].gpio, (addr >> i) & 0x1);
    }

    for (i=0; i<ARRAY_SIZE(iom_fpga_data); i++) {

```

```

        gpio_set_value(iom_fpga_data[i].gpio, (value >> i) & 0x1);

    }

    gpio_set_value(iom_fpga_control[CTRL_nCS].gpio, 0); udelay(1);
    gpio_set_value(iom_fpga_control[CTRL_nWE].gpio, 0); udelay(5);
    //printf("CS:%d, ", i,
    gpio_get_value(iom_fpga_control[CTRL_nCS].gpio));
    //printf("WE:%d, ", i,
    gpio_get_value(iom_fpga_control[CTRL_nWE].gpio));
    //printf("\n");
    gpio_set_value(iom_fpga_control[CTRL_nWE].gpio, 1);
    gpio_set_value(iom_fpga_control[CTRL_nCS].gpio, 1);

    /*
    // Debugging...
    for (i=0; i<ARRAY_SIZE(iom_fpga_address); i++) {
        printf("Address(%d):%d, ", i,
        gpio_get_value(iom_fpga_address[i].gpio));
    }

    printf("\n");
    for (i=0; i<ARRAY_SIZE(iom_fpga_data); i++) {
        printf("Data(%d):%d, ", i, gpio_get_value(iom_fpga_data[i].gpio));
    }

    printf("\n");
    printf("CS:%d, ", gpio_get_value(iom_fpga_control[CTRL_nCS].gpio));
    printf("WE:%d, ", i,
    gpio_get_value(iom_fpga_control[CTRL_nWE].gpio));
    printf("\n");
    */

    return length;
}
EXPORT_SYMBOL(iom_fpga_itf_write);

unsigned char iom_fpga_itf_read(unsigned int addr) {
    unsigned char value = 0;
    int i = 0;

    for (i=0; i<ARRAY_SIZE(iom_fpga_address); i++) {
        gpio_set_value(iom_fpga_address[i].gpio, (addr >> i) & 0x1);

```

```

    }

    gpio_set_value(iom_fpga_control[CTRL_nCS].gpio, 0); udelay(1);
    gpio_set_value(iom_fpga_control[CTRL_nOE].gpio, 0); udelay(1);

    for (i=0; i<ARRAY_SIZE(iom_fpga_data); i++) {
        value += gpio_get_value(iom_fpga_data[i].gpio) << i;
    }

    gpio_set_value(iom_fpga_control[CTRL_nCS].gpio, 1);
    gpio_set_value(iom_fpga_control[CTRL_nOE].gpio, 1);

    printk("FPGA READ: address = 0x%x, data = 0x%x \n",
           addr, value);

    return value;
}
EXPORT_SYMBOL(iom_fpga_itf_read);

int __init iom_fpga_itf_init(void) {
    printk("init module: %s\n", __func__);
    iom_fpga_itf_open();
    return 0;
}

void __exit iom_fpga_itf_exit(void) {
    printk("exit module: %s\n", __func__);
    iom_fpga_itf_release();
}

module_init(iom_fpga_itf_init);
module_exit(iom_fpga_itf_exit);

MODULE_LICENSE("GPL");

```

컴파일을 위한 Makefile에서 실습상황에 맞게 일부 수정할 필요가 있다. 특히, KDIR 환경변수에 커널 소스 디렉터리를 바르게 지정해주어야 한다.

```

$ cat Makefile
#Makefile for a basic kernel module

```

```

obj-m := fpga_interface_driver.o
KDIR :=/usr/src/linux/
PWD :=$(shell pwd)

all: driver

driver:
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules

install_nfs:
    cp -a fpga_interface_driver.ko /nfs

install_scp:
    scp fpga_interface_driver.ko pi@192.168.0.40:/home/pi/

clean:
    rm -rf *.ko
    rm -rf *.mod.*
    rm -rf *.o
    rm -rf Module.symvers
    rm -rf modules.order
    rm -rf .interface*
    rm -rf .tmp*

```

다음과 같이 컴파일하면 밑줄처럼 커널 모듈이 생성된다.

```

$ make
$ ls
Makefile          fpga_interface_driver.ko      fpga_interface_driver.o
Module.symvers    fpga_interface_driver.mod.c  modules.order
fpga_interface_driver.c  fpga_interface_driver.mod.o

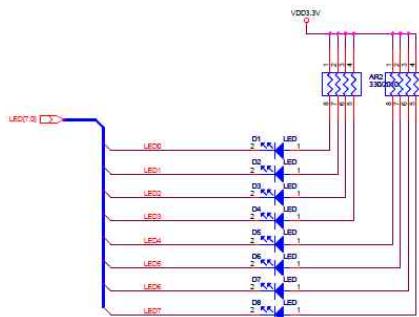
```

이제, 이 모듈을 다음의 명령을 사용하여 커널에 링크시킨다. 각각의 FPGA 디바이스를 제어하려면 반드시 이 모듈이 커널에 링크되어야 한다.

```
$ sudo insmod fpga_interface_driver.ko
```

16.2.3 FPGA LED 제어

키트에서 LED 디바이스의 회로도는 다음과 같다. 이 회로도에서는 LED에 Low 신호가 인가하면 해당 LED는 ON 된다. 하지만 해당 키트는 라즈베리파이에서 출력된 신호의 반전된 신호를 이 회로에 인가하므로, 라즈베리파이가 High 신호를 출력하면 해당 LED는 ON 됨을 유의한다. LED 디바이스의 FPGA 주소는 0x016이며, 이 주소에 1바이트의 데이터를 출력함으로써 8개의 LED를 제어할 수 있다. 키트상의 D8은 데이터의 LSB 비트에 대응함을 유의한다.



LED 디바이스 드라이버 소스

LED 디바이스의 FPGA 주소는 0x016이며, 소스에서 모듈의 주변호는 260, 모듈이름은 fpga_led로 설정되어 있다.

```
$ cat fpga_led_driver.c
/*
 * FPGA LED Ioremap Control
 */
FILE : fpga_led_driver.c
AUTH : largest@huins.com */

#include <linux/module.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/platform_device.h>
#include <linux/delay.h>
```

```

#include <asm/io.h>
#include <asm/uaccess.h>
#include <linux/kernel.h>
#include <linux/ioport.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/version.h>

#define IOM_LED_MAJOR 260           // major number
#define IOM_LED_NAME "fpga_led"    // ioboard led device name

#define IOM_LED_ADDRESS 0x016       // FPGA physical address

//Global variable
static int ledport_usage = 0;

// define functions...
ssize_t iom_led_write(struct file *inode, const char *gdata, size_t length,
loff_t *off_what);
ssize_t iom_led_read(struct file *inode, char *gdata, size_t length, loff_t
*off_what);
int iom_led_open(struct inode *minode, struct file *mfile);
int iom_led_release(struct inode *minode, struct file *mfile);

// define file_operations structure
struct file_operations iom_led_fops = {
    .owner        = THIS_MODULE,
    .open         = iom_led_open,
    .write        = iom_led_write,
    .read         = iom_led_read,
    .release      = iom_led_release,
};

// when led device open ,call this function
int iom_led_open(struct inode *minode, struct file *mfile) {
    if(ledport_usage != 0) return -EBUSY;
    ledport_usage = 1;
    return 0;
}

// when led device close ,call this function

```

```
int iom_led_release(struct inode *minode, struct file *mfile) {
    ledport_usage = 0;
    return 0;
}

// when write to led device ,call this function
ssize_t iom_led_write(struct file *inode, const char *gdata, size_t length,
loff_t *off_what) {
    unsigned char value;
    const char *tmp = gdata;
    if (copy_from_user(&value, tmp, 1))
        return -EFAULT;

    iom_fpga_itf_write((unsigned int)IOM_LED_ADDRESS,value);
    return length;
}

// when read to led device ,call this function
ssize_t iom_led_read(struct file *inode, char *gdata, size_t length, loff_t
*off_what) {
    unsigned char value = 0;
    char *tmp = gdata;

    value = iom_fpga_itf_read((unsigned int)IOM_LED_ADDRESS);
    if (copy_to_user(tmp, &value, 1))
        return -EFAULT;
    return length;
}

int __init iom_led_init(void) {
    int result;
    result = register_chrdev(IOM_LED_MAJOR,
                            IOM_LED_NAME, &iom_led_fops);
    if(result < 0) {
        printk(KERN_WARNING"Can't get any major\n");
        return result;
    }
    printk("init module %s, major number %d, minor nuber %d\n",
          IOM_LED_NAME, IOM_LED_MAJOR, result);
    return 0;
}
```

```

void __exit iom_led_exit(void) {
    unregister_chrdev(IOM_LED_MAJOR, IOM_LED_NAME);
}

module_init(iom_led_init);
module_exit(iom_led_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Huins");

```

LED 제어용 응용 프로그램 소스

명령행에서 LED에 출력한 1바이트의 데이터를 정수형으로 입력 받도록 구현된 소스다.

```

$ cat fpga_test_led.c
/* FPGA LED Test Application
File : fpga_test_led.c */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define LED_DEVICE "/dev/fpga_led"

int main(int argc, char **argv) {
    int dev;
    int data;
    int retval;

    if(argc!=2) {
        printf("please input the parameter! \n");
        printf("ex)./test_led 7 (0~255)\n");
        return -1;
    }

```

476 라즈베리파이 기본 임베디드 시스템 활용

```
data = atoi(argv[1]);
if((data<0)|| (data>0xff)) {
    printf("Invalid range!\n");
    exit(1);
}

dev = open(LED_DEVICE, O_RDWR);
if (dev<0) {
    printf("Device open error : %s\n",LED_DEVICE);
    exit(1);
}

retval=write(dev, &data, 1);
if(retval<0) {
    printf("Write Error!\n");
    return -1;
}

sleep(1);

data=0;
retval=read(dev, &data, 1);
if(retval<0) {
    printf("Read Error!\n");
    return -1;
}
printf("Current LED Value : %d\n",data);

printf("\n");

close(dev);

return(0);
}
```

컴파일을 위한 Makefile을 일부 수정한다.

```
$ cat Makefile
#Makefile for a basic kernel module
```

```

obj-m := fpga_led_driver.o
KDIR := /usr/src/linux/
PWD := $(shell pwd)

all: driver app
driver:
$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules

app:
    arm-linux-gnueabihf-gcc -static -o fpga_test_led fpga_test_led.c

install_nfs:
    cp -a fpga_led_driver.ko /nfsroot
    cp -a fpga_test_led /nfsroot

Install_scp:
    scp fpga_led_driver.ko fpga_test_led
pi@192.168.1.xxx:/home/pi/Modules

clean:
    rm -rf *.ko
    rm -rf *.mod.*
    rm -rf *.o
    rm -rf fpga_test_led
    rm -rf Module.symvers
    rm -rf modules.order
    rm -rf .led*

```

소스 컴파일 및 실행

```

$ cd fpga_led
$ make

$ sudo insmod fpga_led_driver.ko
$ sudo mknod /dev/fpga_led c 260 0

$ sudo ./fpga_test_led 1          // D8 LED ON

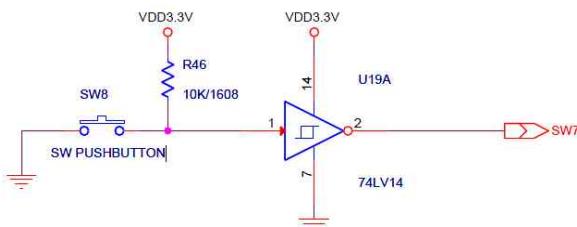
```



혹은, \$ sudo ./fpga_test_led 255 // 모든 LED를 ON

16.2.4 FPGA BTN 제어

하나의 BTN 스위치 회로도는 다음과 같으며, 9개의 스위치 회로가 3x3의 형태로 배열되어 있다. 이 회로는 스위치를 눌렀을 때 High 신호를 출력한다.



BTN 디바이스 모듈 소스

이 회로의 FPGA 물리 주소는 0x050이고, 소스에서 주번호는 265, 모듈 이름은 fpga_push_switch로 설정되어 있다.

```
$ cat fpga_push_switch_driver.c
/*
 * FPGA PUSH SWITCH IOREMAP Control
 * FILE : fpga_push_switch_driver.c
 * AUTH : largest@huins.com*/
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/init.h>
```

```

#include <linux/slab.h>
#include <linux/platform_device.h>
#include <linux/delay.h>

#include <asm/io.h>
#include <asm/uaccess.h>
#include <linux/kernel.h>
#include <linux/ioport.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/version.h>

#define MAX_BUTTON 9

#define IOM_FPGA_PUSH_SWITCH_MAJOR 265           // major#
#define IOM_FPGA_PUSH_SWITCH_NAME "fpga_push_switch" // name
#define IOM_FPGA_PUSH_SWITCH_ADDRESS 0x050        // addr

extern unsigned char iom_fpga_itf_read(unsigned int addr);
extern ssize_t iom_fpga_itf_write(unsigned int addr, unsigned char value);

//Global variable
static int fpga_push_switch_port_usage = 0;
static unsigned char *iom_fpga_push_switch_addr;
static unsigned char *iom_demo_addr;

// define functions...
ssize_t iom_fpga_push_switch_read(struct file *inode, char *gdata, size_t length, loff_t *off_what);
int iom_fpga_push_switch_open(struct inode *minode, struct file *mfile);
int iom_fpga_push_switch_release(struct inode *minode, struct file *mfile);

// define file_operations structure
struct file_operations iom_fpga_push_switch_fops = {
    owner:      THIS_MODULE,
    open:       iom_fpga_push_switch_open,
    read:       iom_fpga_push_switch_read,
    release:   iom_fpga_push_switch_release,
};

// when fpga_push_switch device open ,call this function

```

```
int iom_fpga_push_switch_open(struct inode *minode, struct file *mfile) {
    if(fpga_push_switch_port_usage != 0) return -EBUSY;
    fpga_push_switch_port_usage = 1;
    return 0;
}

// when fpga_push_switch device close ,call this function
int iom_fpga_push_switch_release(struct inode *minode, struct file *mfile)
{
    fpga_push_switch_port_usage = 0;
    return 0;
}

// when read from fpga_push_switch device ,call this function
ssize_t iom_fpga_push_switch_read(struct file *inode, char *gdata, size_t
length, loff_t *off_what) {
    int i;
    unsigned char push_sw_value[MAX_BUTTON];
    unsigned char value;

    for(i=0;i<=length;i++) {
        value = iom_fpga_itf_read((unsigned
int)IOM_FPGA_PUSH_SWITCH_ADDRESS+i);
        push_sw_value[i] = value &0xFF;
    }

    if (copy_to_user(gdata, push_sw_value, length))
        return -EFAULT;

    return length;
}

int __init iom_fpga_push_switch_init(void) {
    int result;

    result = register_chrdev(IOM_FPGA_PUSH_SWITCH_MAJOR,
                           IOM_FPGA_PUSH_SWITCH_NAME,
                           &iom_fpga_push_switch_fops);
    if(result < 0) {
        printk(KERN_WARNING"Can't get any major\n");
        return result;
    }
}
```

```

    }

    printk("init module, %s major number : %d\n",
           IOM_FPGA_PUSH_SWITCH_NAME,
           IOM_FPGA_PUSH_SWITCH_MAJOR);

    return 0;
}

void __exit iom_fpga_push_switch_exit(void) {
    unregister_chrdev(IOM_FPGA_PUSH_SWITCH_MAJOR,
                      IOM_FPGA_PUSH_SWITCH_NAME);
}

module_init(iom_fpga_push_switch_init);
module_exit(iom_fpga_push_switch_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Huins");

```

BTN 제어용 응용 프로그램 소스

9개 BTN의 상태를 주기적으로 읽어 출력하는 프로그램 소스이다. 키보드에서 ctrl-c의 입력에 대한 시그널 처리로 프로그램을 종료하도록 구현되어 있다.

```

$ cat fpga_test_push.c
/* FPGA Push Switch Test Application
File : fpga_test_push.c
Auth : largest@huins.com */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <signal.h>

```

482 리즈베리파이기반 임베디드시스템응용

```
#define MAX_BUTTON 9

unsigned char quit = 0;

void user_signal1(int sig) {
    quit = 1;
}

int main(void) {
    int i;
    int dev;
    int buff_size;

    unsigned char push_sw_buff[MAX_BUTTON];

    dev = open("/dev/fpga_push_switch", O_RDWR);
    if (dev<0){
        printf("Device Open Error\n");
        close(dev);
        return -1;
    }

    (void)signal(SIGINT, user_signal1);

    buff_size=sizeof(push_sw_buff);
    printf("Press <ctrl+c> to quit. \n");
    while(!quit){
        usleep(400000);
        read(dev, &push_sw_buff, buff_size);

        for(i=0;i<MAX_BUTTON;i++) {
            printf("[%d] ",push_sw_buff[i]);
        }
        printf("\n");
    }
    close(dev);
}
```

Makefile

```

$ cat Makefile
#Makefile for a basic kernel module
obj-m := fpga_push_switch_driver.o
KDIR   := /usr/src/linux/
PWD    := $(shell pwd)

all: driver app

driver:
        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules

app:
        arm-linux-gnueabihf-gcc -static -o fpga_test_push_switch \
fpga_test_push_switch.c

install_nfs:
        cp fpga_push_switch_driver.ko /nfsroot
        cp fpga_test_push_switch /nfsroot

install_scp:
        scp fpga_push_switch_driver.ko pi@192.168.1.243:/ho/me/pi/Modules
clean:
        rm -rf *.ko
        rm -rf *.mod.*
        rm -rf *.o
        rm -rf fpga_test_push_switch
        rm -rf Module.symvers
        rm -rf modules.order
        rm -rf .push_switch*
        rm -rf .tmp*

```

소스 컴파일 및 실행

```

$ cd fpga_push_switch
$ make

$ sudo insmod fpga_push_switch_driver.ko
$ sudo mknod /dev/fpga_push_switch c 265 0

```

```
$ sudo ./fpga_test_push_switch
Press <ctrl+c> to quit.
[0] [0] [0] [0] [0] [0] [0] [0] [0]
[0] [0] [0] [0] [0] [0] [0] [0] [0]
[0] [0] [0] [0] [1] [0] [0] [0] [0]
[0] [0] [0] [0] [1] [0] [0] [0] [0]
[0] [0] [0] [0] [0] [0] [0] [0] [0]
[0] [0] [0] [1] [0] [1] [0] [0] [0]
[0] [0] [0] [1] [0] [1] [0] [0] [0]
[0] [0] [0] [0] [0] [0] [0] [0] [0]
[0] [0] [0] [0] [0] [0] [0] [0] [0]
```

실행중에 버튼 스위치를 눌러 그 결과를 확인한다. 눌린 스위치는 1로 표시된다.

16.3 GUI 통한 디바이스 제어

GUI(Graphical user interface)는 Windows 환경과 같이 화면위에 그래픽 요소들을 배치하고, 마우스 등의 기기로 간편히 프로그램을 실행하거나 파일을 생성하는 등의 사용자에게 시스템 사용에 있어서의 편의를 제공할 수 있는 수단이다.

GUI Widget을 통해서 앞에서 살펴본 FPGA 디바이스들을 GUI 환경에서 제어하는 것에 대해 살펴본다.

16.3.1 Qt Creator 설치

GUI 환경을 제공하기 위한 개발 툴로 Qt가 있다. 즉, Qt는 GUI를 더욱 편리하게 개발할 수 있게 해주는 크로스 플랫폼 프레임워크이다. Qt에서 주로 사용되는 언어는 C++이지만, Python이나 C, 펠 등 다양한 언어 환경에서 사용 할 수 있다. Qt는 일반 프로그램 구현처럼 코딩하여 사용할 수 있지만, 보다 편리한 환경을 제공하기 위한 Qt Creator, Qt Designer 등의 툴도 있다.

Qt를 통해 개발하는 방법은 가상머신 상에서 크로스컴파일을 통해 개발하는 방법과 타깃 보드 상에서 개발하는 경우로 나누어 볼 수 있다. 크로스컴파일을 통해 개발하는 것은 보드마다 환경 설정을 다르게 해줘야 하고, 얼마간의 준비 과정을 거쳐야 한다. 따라서 여기서는 Qt Creator를 사용하여 프로그램을 구현하는 것에 대해 살펴본다.

C 소스를 컴파일할 수 있는 환경이 이미 구축된 상황인 경우 끝의 두 패키지를 설치하면 된다.

```
$ sudo apt-get update
$ sudo apt-get install build-essential
$ sudo apt-get install cmake
$ sudo apt-get install qtcreator
$ sudo apt-get install qt5-default
```

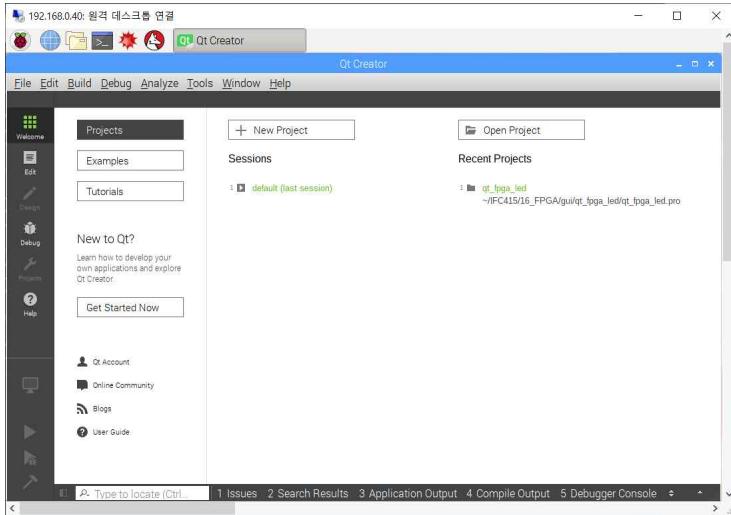
또한 Achro-EM 상의 디스플레이 모듈의 기본 해상도는 800x480으로 화면이 너무 작아 Qtcreator를 사용하기에는 무리가 있으므로, Windows 환경에서 mstsc로 라즈베리파이 보드에 접속한 후, 다음의 작업을 진행한다.

16.3.2 qt_test 프로젝트

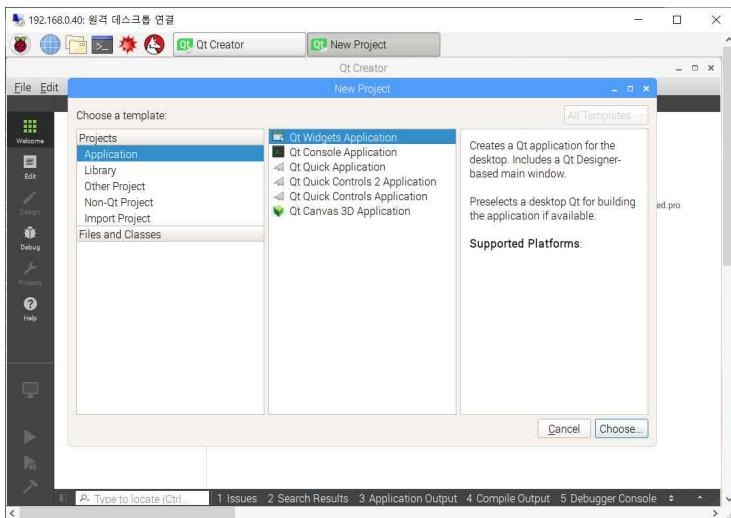
단순히 윈도우를 화면에 띄우는 프로그램을 작성하면서 Qt Creator의 환경설정 등 의 과정을 살펴본다.

Qt Creator의 설치가 끝나고, 시작 - Programming 메뉴에 Qt Creator 아이콘 이 보일 것이며, 이를 클릭하여 실행한다. 초기 실행화면은 다음과 같다.

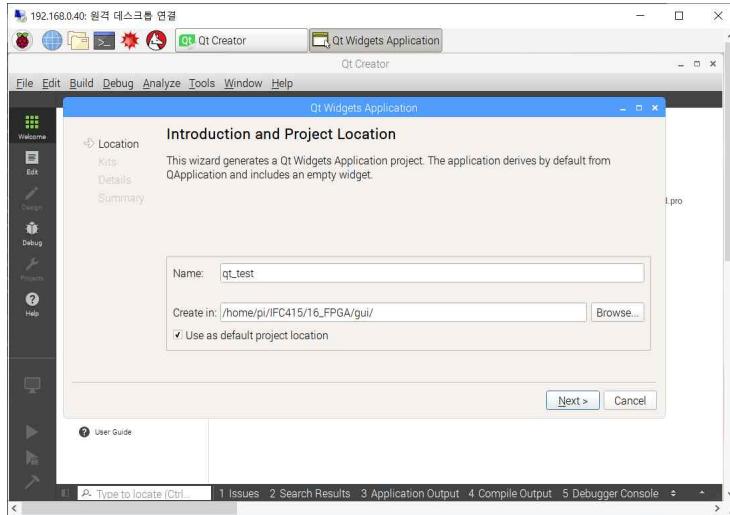
486 라즈베리파이기반 임베디드시스템응용



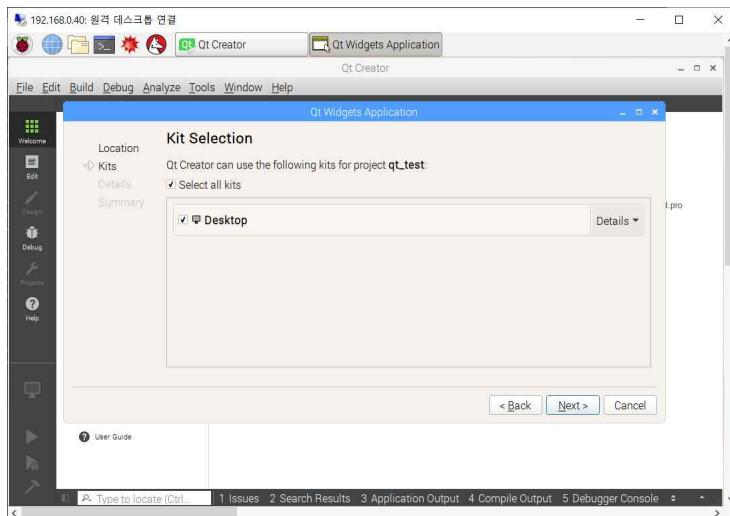
새로운 프로젝트를 생성할 것이므로, New Project 항목을 클릭하면 다음과 같은 서브 화면이 나타난다.



이 화면에서 Application - Qt Widgets Application 설정을 확인하고, Choose 버튼을 클릭하면 다음의 화면이 보인다.



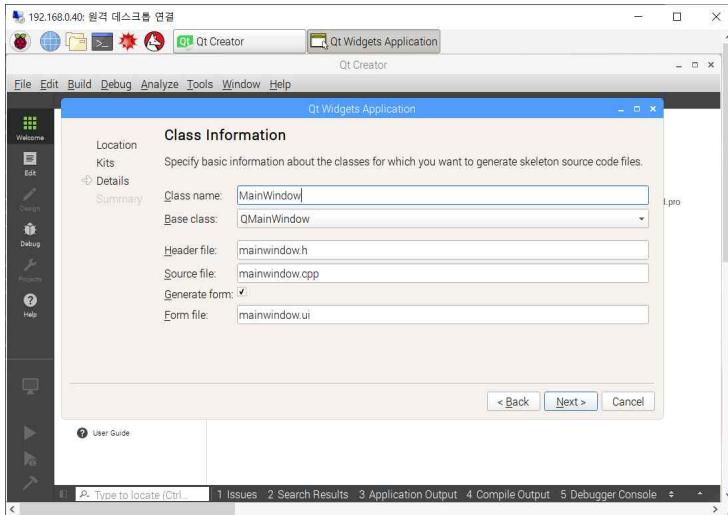
이 화면의 Name에 프로젝트 명을, Create in은 프로젝트 디렉터리가 위치할 경로를 설정한다. 이 경로 아래에 프로젝트 명의 디렉터리가 생성되고 관련 소스들이 위치한다. 또한 컴파일하게 되면 컴파일 결과 실행파일 등은 이 경로에 build-프로젝트명/Desktop-Debug 디렉터리로 위치하게 된다. 프로젝트 명은 qt_test로 가정 한다. Next 버튼을 클릭하면 다음의 화면이 나타난다.



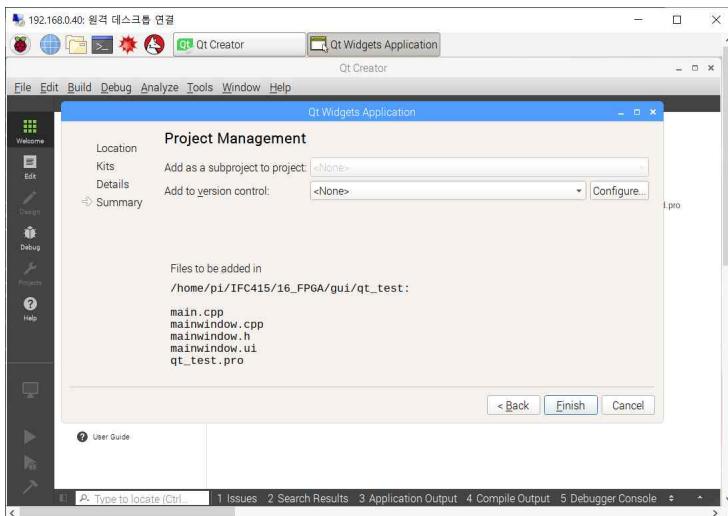
이 Kit Selection 화면에서 Desktop 선택을 확인하고, Next 버튼을 클릭하면 다음

488 라즈베리파이기반 임베디드시스템용

의 화면이 나타난다.

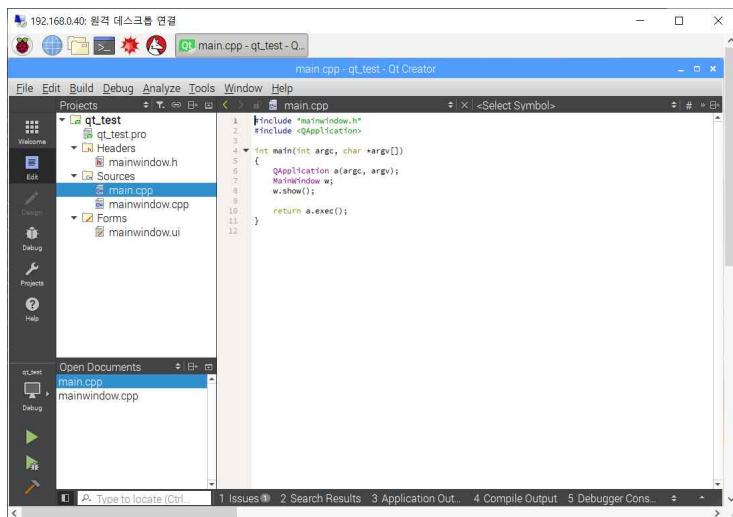


이 Class information 화면에서는 프로젝트관련 클래스 명 및 관련 파일이름들을 설정하는 화면이다. 변경없이 Next 버튼을 클릭하면 다음의 화면과 같이 새로운 프로젝트관련 요약정보가 나타난다. .



이 화면의 내용을 확인 후 Finish 버튼을 클릭하면, 다음과 같이 생성된 프로젝트

관련 화면이 나타난다. 이는 지금까지의 과정을 통하여 새로운 프로젝트 qt_test가 생성되고, 관련 소스 파일의 기본적인 포맷도 완성된다. qt_test 프로젝트와 관련 소스 파일들을 선택하여 편집하여 기능을 추가하는 등의 작업을 할 수 있는 상태가 된 것이다.



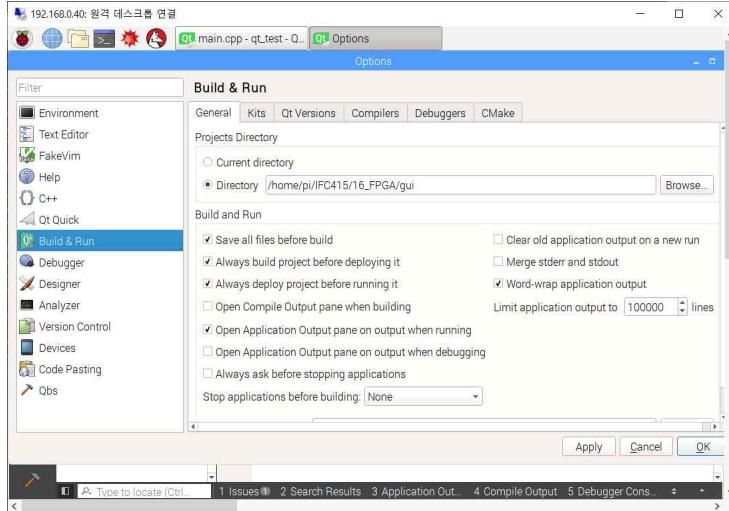
이제 컴파일을 위한 환경 설정관련 내용을 살펴보도록 한다. 이 과정은 컴파일러를 등록하는 등의 작업인 것이다.

환경설정관련

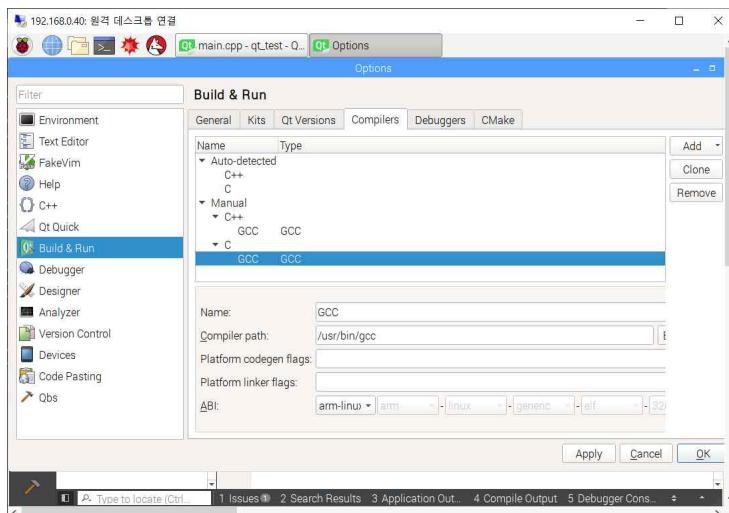
우선 메뉴의 Tools - Options를 클릭하여 다음의 화면이 나타나면, 좌측의 항목 중 Build & Run을 선택하면 화면 우측과 같이 관련 탭들의 창이 나타난다. 이들 탭들에서 삼각형 모양의 경고 항목이 없어야 컴파일하는데 문제가 없다.

우선 General 탭에서는 프로젝트를 생성할 때 설정하였던 프로젝트 디렉터리와 일 반적인 관련 옵션을 확인할 수 있다.

490 라즈베리파이기반 임베디드시스템응용

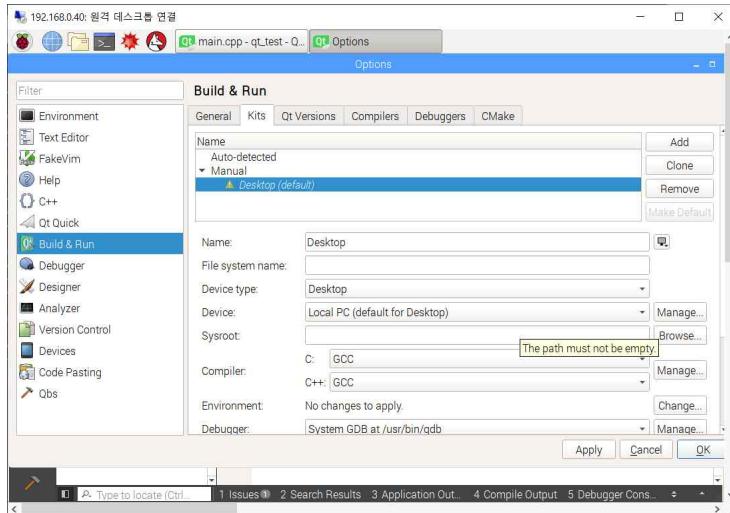


다음과 같은 Compilers 탭 화면에서 컴파일러의 경로를 등록지정한다. Add - GCC - C를 선택하고, Compiler path항에 /usr/bin/gcc를 설정하고 Apply 버튼을 클릭한다. 또한 Add - GCC- C++을 선택하고, Compiler path항에 /usr/bin/gcc를 설정하고 Apply 버튼을 클릭한다.



다음으로 Kits 탭의 Desktop 항목을 선택하여 나타난 다음과 같은 화면에서 Compiler C:와 Compiler C++: 항에 앞서 설정한 GCC를 선택하고 Apply 버튼을

클릭한다.

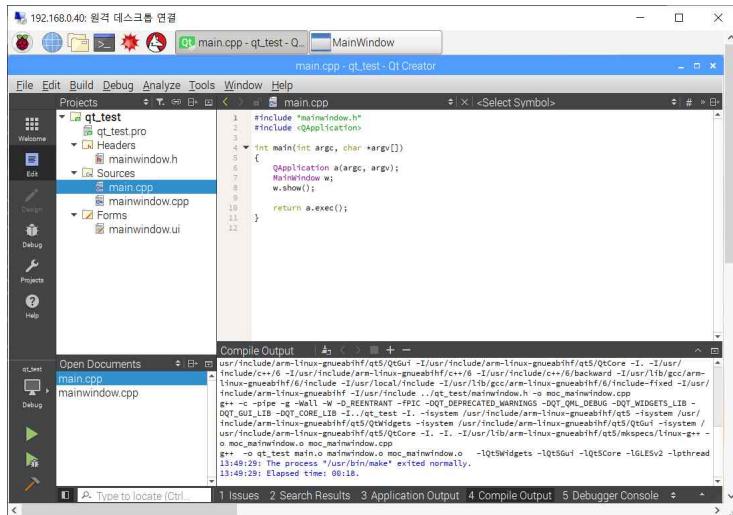


기타 Debugger, Qt version, CMake 탭 등의 설정내용을 확인하고, OK 버튼을 클릭하여 환경설정 관련 화면을 닫는다.

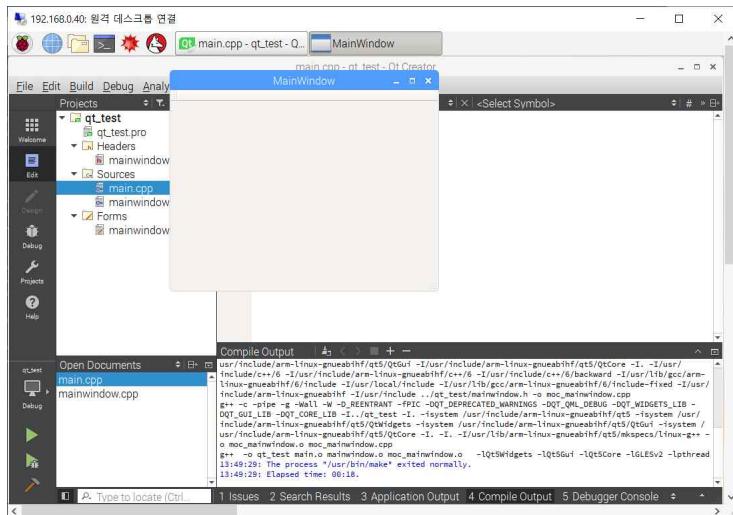
일단 환경설정이 제대로 되어 컴파일이 가능한 상태가 되면 다른 프로젝트를 열어도 이 상태가 유지되므로 컴파일하는데 문제가 없게 된다.

컴파일 및 실행

컴파일 및 실행을 위해서는 메뉴에서 Build를 선택하거나 Qt Creator의 좌하단측의 삼각형 모양을 클릭하면 된다. 컴파일 결과는 이 화면 하단의 항목 중 4 Compile Output을 클릭하여 확인할 수 있다.



다음 화면은 qt_test 프로젝트의 실행 결과로 생성된 실행화면을 보여주고 있다.



이제 터미널 창에서 `qt_test` 프로젝트 관련 내용들을 살펴보자. 다음의 소스 파일들은 기본 포맷으로 자동 생성된 파일들이다.

```
$ pwd
```

/home/pi/IFC415/16_FPGA/gui

```
$ ls qt_test/
main.cpp     mainwindow.h    qt_test.pro
mainwindow.cpp mainwindow.ui  qt_test.pro.user
```

컴파일 후, 생성된 파일들은 다음과 같이 build_프로젝트명_Desktop-Debug 디렉터리에 위치하는 것을 관찰할 수 있다. qt_test 파일이 실행 파일이다.

```
$ ls build-qt_test-Desktop-Debug/
main.o          Makefile        moc_mainwindow.o ui_mainwindow.h
mainwindow.o   moc_mainwindow.cpp qt_test
```

다음 명령으로 실행을 위해 실행파일을 사용자 계정의 ~/Desktop/으로 복사한다. 이는 앞서 환경설정 화면에서 Kits 탭의 Device type이 Desktop으로 되어있어서이다.

```
$ cp build-qt_test-Desktop-Debug/qt_test ~/Desktop/
```

바탕화면을 살펴보면 qt_test 아이콘이 보이고, 실행하려면 이 아이콘을 더블클릭하면 된다.

16.3.3 GUI 환경에서의 FPGA LED 제어

GUI 환경에서 LED를 제어하는 소스는 qt_fpga_led 디렉터리로 제시된다. 이미 소스가 작성된 상태이므로 Qt Creator의 초기화면에서 Open Project 항목을 선택하여 경로 지정하여 프로젝트를 연다. 그런 후 컴파일하여 실행하면 된다.

```
$ pwd
/home/pi/IFC415/16_FPGA/gui
$ ls build-qt_fpga_led-Desktop-Debug/
main.o          Makefile        moc_mainwindow.o ui_mainwindow.h
mainwindow.o   moc_mainwindow.cpp qt_fpga_led
```

다음 명령으로 실행을 위해 실행파일을 사용자 계정의 ~/Desktop/으로 복사한다.

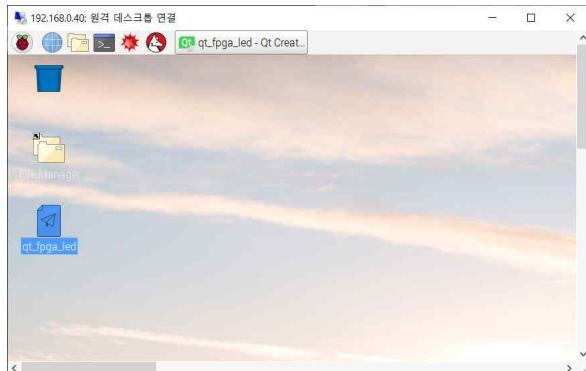
494 라즈베리파이기반 임베디드시스템용

```
$ cp build-qt_fpga_led-Desktop-Debug/qt_fpga_led ~/Desktop/
```

다음과 같이 사용자 계정의 Desktop 디렉터리에 있는 파일들은 사용자 계정으로 로그하였을 때 바탕화면에 나타나는 목록들이다. 잘 복사된 것을 확인할 수 있다.

```
$ cd  
$ ls ./Desktop/  
pcmanfm.desktop  qt_fpga_led
```

바탕화면을 살펴보면 다음과 같이 qt_fpga_led의 아이콘이 생성된 것을 관찰할 수 있다. 이 실행파일은 FPGA LED 디바이스를 GUI 환경에서 제어하는 응용 프로그램일 뿐이다.

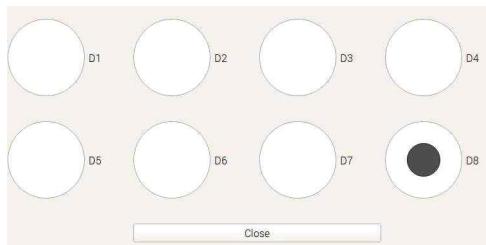


GUI 환경의 LED 제어 응용프로그램을 실행하기 위해서는 다음의 과정을 진행해야 한다. 우선 앞 절에서 살펴본 FPAG 인터페이스 모듈과 LED 제어용 모듈을 다음의 명령으로 커널과 링크한다.

```
$ sudo insmod fpga_interface_driver.ko
```

```
$ sudo insmod fpga_led_driver.ko  
$ sudo mknod /dev/fpga_led c 260 0
```

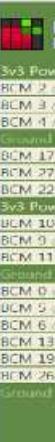
이제 바탕화면의 해당 아이콘을 더블 클릭하면 다음과 같은 실행화면이 나타난다.



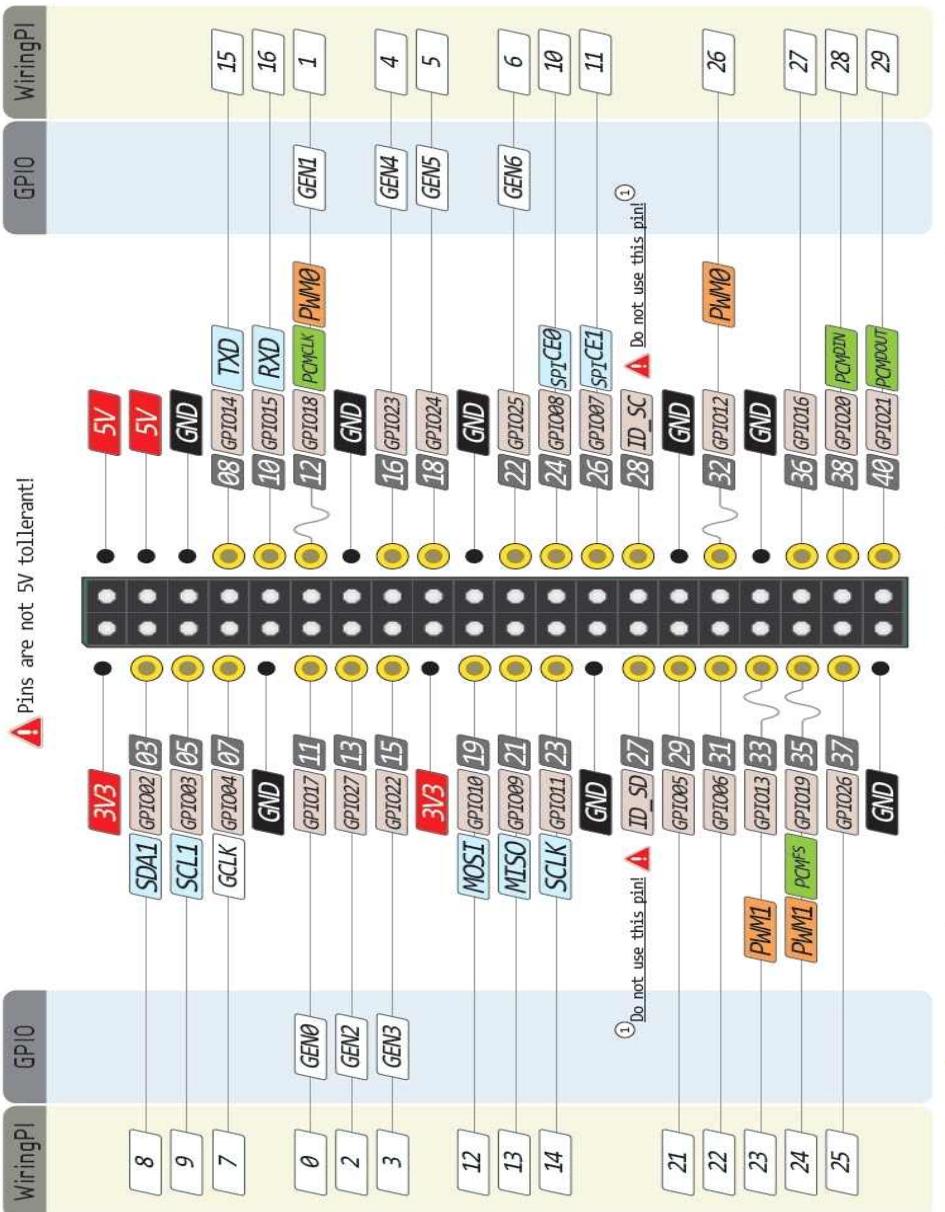
마우스로 적절한 위치를 클릭함으로써 FPGA LED를 제어할 수 있고, 그 결과를 관찰할 수 있다.

참고자료

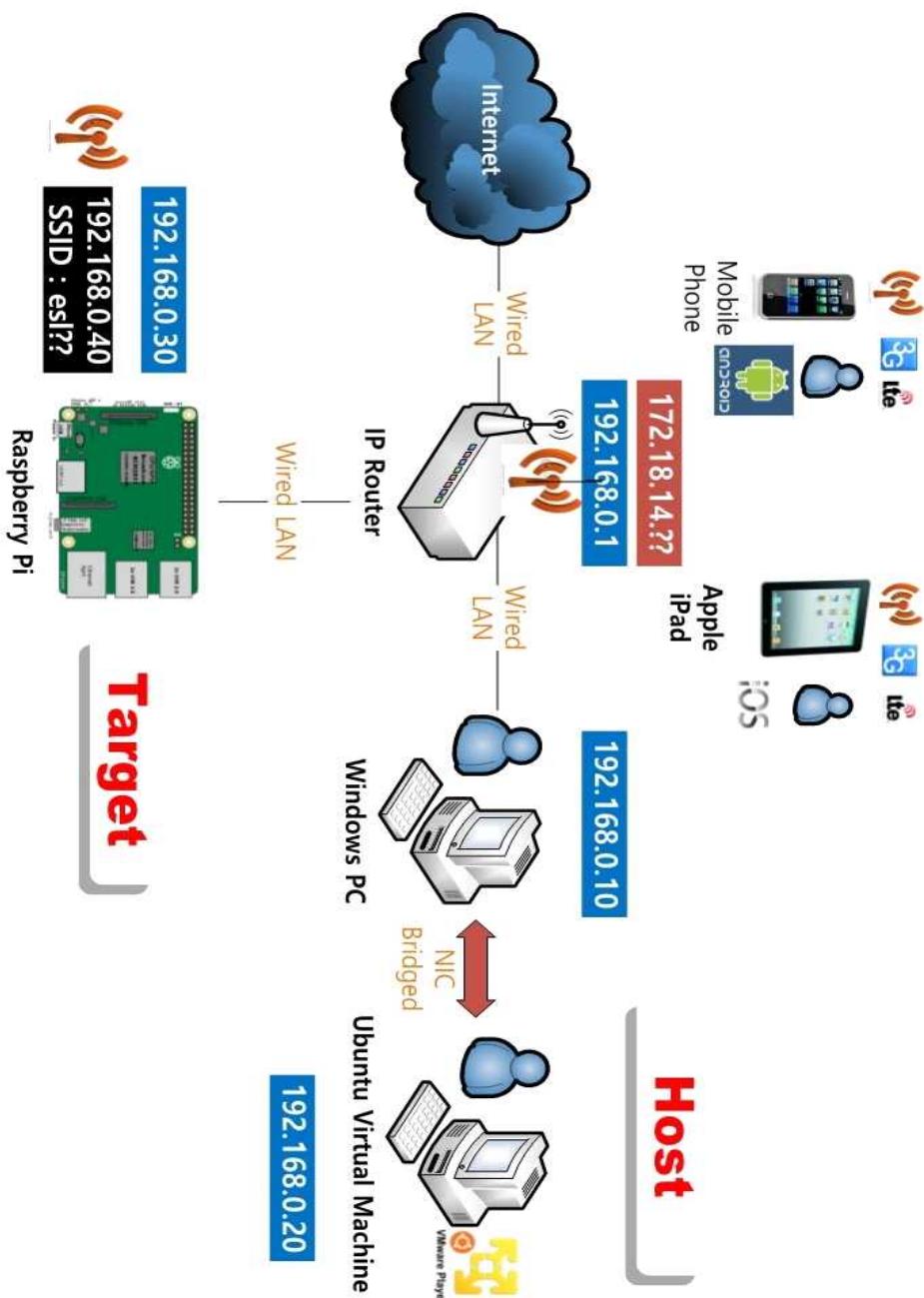
- [1] (주)휴인스, Achro EM-Kit Linux 설계 및 응용



부록1. 라즈베리파이3 핀 레이아웃



부록2. 네트워크 레이아웃



부록3. wiringPi 라이브러리

/usr/include/wiringPi.h

```
pi@raspberrypi:~/my $ cat /usr/include/wiringPi.h
/*
 * wiringPi.h:
 *      Arduino like Wiring library for the Raspberry Pi.
 *      Copyright (c) 2012-2017 Gordon Henderson
 ****
 * This file is part of wiringPi:
 *      https://projects.drogon.net/raspberry-pi/wiringpi/
 *
 *      wiringPi is free software: you can redistribute it and/or modify
 *      it under the terms of the GNU Lesser General Public License as
 *      published by
 *          the Free Software Foundation, either version 3 of the License, or
 *          (at your option) any later version.
 *
 *      wiringPi is distributed in the hope that it will be useful,
 *      but WITHOUT ANY WARRANTY; without even the implied warranty
 *      of
 *          MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the
 *      GNU Lesser General Public License for more details.
 *
 *      You should have received a copy of the GNU Lesser General Public
 *      License along with wiringPi. If not, see <http://www.gnu.org/licenses/>.
 ****
 */
#ifndef __WIRING_PI_H__
#define __WIRING_PI_H__
// C doesn't have true/false by default and I can never remember which
//      way round they are, so ...
//      (and yes, I know about stdbool.h but I like capitals for these and
I'm old)
#ifndef TRUE
#define TRUE    (1==1)
```

```

# define      FALSE  (!TRUE)
#endif
// GCC warning suppressor
#define UNU __attribute__((unused))
// Mask for the bottom 64 pins which belong to the Raspberry Pi
// The others are available for the other devices
#define PI_GPIO_MASK (0xFFFFFC0)
// Handy defines
// wiringPi modes
#define WPI_MODE_PINS          0
#define WPI_MODE_GPIO           1
#define WPI_MODE_GPIO_SYS        2
#define WPI_MODE_PHYS            3
#define WPI_MODE_PIFACE          4
#define WPI_MODE_UNINITIALISED   -1
// Pin modes
#define INPUT                  0
#define OUTPUT                 1
#define PWM_OUTPUT              2
#define GPIO_CLOCK              3
#define SOFT_PWM_OUTPUT          4
#define SOFT_TONE_OUTPUT         5
#define PWM_TONE_OUTPUT          6
#define LOW                     0
#define HIGH                    1
// Pull up/down/none
#define PUD_OFF                 0
#define PUD_DOWN                1
#define PUD_UP                  2
// PWM
#define PWM_MODE_MS              0
#define PWM_MODE_BAL              1
// Interrupt levels
#define INT_EDGE_SETUP             0
#define INT_EDGE_FALLING            1
#define INT_EDGE_RISING             2
#define INT_EDGE_BOTH               3
// Pi model types and version numbers
// Intended for the GPIO program Use at your own risk.
#define PI_MODEL_A                 0
#define PI_MODEL_B                 1
#define PI_MODEL_AP                2

```

```
#define PI_MODEL_BP            3
#define PI_MODEL_2              4
#define PI_ALPHA                 5
#define PI_MODEL_CM              6
#define PI_MODEL_07              7
#define PI_MODEL_3               8
#define PI_MODEL_ZERO             9
#define PI_MODEL_CM3             10
#define PI_MODEL_ZERO_W           12
#define PI_VERSION_1              0
#define PI_VERSION_1_1             1
#define PI_VERSION_1_2             2
#define PI_VERSION_2              3
#define PI MAKER_SONY             0
#define PI MAKER_EGOMAN            1
#define PI MAKER_EMBEST            2
#define PI MAKER_UNKNOWN            3
extern const char *piModelNames [16] ;
extern const char *piRevisionNames [16] ;
extern const char *piMakerNames [16] ;
extern const int piMemorySize [ 8 ] ;
//      Intended for the GPIO program Use at your own risk.
// Threads
#define PI_THREAD(X) void *X (UNU void *dummy)
// Failure modes
#define WPI_FATAL     (1==1)
#define WPI_ALMOST    (1==2)
// wiringPiNodeStruct:
//      This describes additional device nodes in the extended wiringPi
//      2.0 scheme of things.
//      It's a simple linked list for now, but will hopefully migrate to
//      a binary tree for efficiency reasons - but then again, the
//      chances
//      of more than 1 or 2 devices being added are fairly slim, so who
//      knows....
struct wiringPiNodeStruct
{ int      pinBase ;
  int      pinMax ;
  int          fd ;      // Node specific
  unsigned int data0 ; // ditto
  unsigned int data1 ; // ditto
  unsigned int data2 ; // ditto
```

```

unsigned int data3 ; // ditto
void (*pinMode) (struct wiringPiNodeStruct *node,
int pin, int mode) ;
void (*pullUpDnControl) (struct wiringPiNodeStruct *node,
int pin, int mode) ;
int (*digitalRead) (struct wiringPiNodeStruct *node, int
pin) ;
//unsigned int (*digitalRead8) (struct wiringPiNodeStruct *node, int
pin) ;
void (*digitalWrite) (struct wiringPiNodeStruct *node, int
pin, int value) ;
// void (*digitalWrite8) (struct wiringPiNodeStruct *node, int
pin, int value) ;
void (*pwmWrite) (struct wiringPiNodeStruct *node,
int pin, int value) ;
int (*analogRead) (struct wiringPiNodeStruct *node,
int pin) ;
void (*analogWrite) (struct wiringPiNodeStruct *node,
int pin, int value) ;
struct wiringPiNodeStruct *next ;
} ;
extern struct wiringPiNodeStruct *wiringPiNodes ;
// Function prototypes
// c++ wrappers thanks to a comment by Nick Lott
// (and others on the Raspberry Pi forums)
#ifndef __cplusplus
extern "C" {
#endif
// Data
// Internal
extern int wiringPiFailure (int fatal, const char *message, ...) ;
// Core wiringPi functions
extern struct wiringPiNodeStruct *wiringPiFindNode (int pin) ;
extern struct wiringPiNodeStruct *wiringPiNewNode (int pinBase, int
numPins) ;
extern void wiringPiVersion (int *major, int *minor) ;
extern int wiringPiSetup (void) ;
extern int wiringPiSetupSys (void) ;
extern int wiringPiSetupGpio (void) ;
extern int wiringPiSetupPhys (void) ;
extern void pinModeAlt (int pin, int mode) ;
extern void pinMode (int pin, int mode) ;

```

```

extern      void pullUpDnControl    (int pin, int pud) ;
extern      int  digitalRead       (int pin) ;
extern      void  digitalWrite     (int pin, int value) ;
extern unsigned int  digitalRead8   (int pin) ;
extern      void  digitalWrite8    (int pin, int value) ;
extern      void  pwmWrite        (int pin, int value) ;
extern      int  analogRead      (int pin) ;
extern      void  analogWrite     (int pin, int value) ;

// PiFace specifics
//      (Deprecated)
extern int  wiringPiSetupPiFace (void) ;
extern int  wiringPiSetupPiFaceForGpioProg (void) ; // Don't use this
- for gpio program only
// On-Board Raspberry Pi hardware specific stuff
extern      int  piGpioLayout     (void) ;
extern      int  piBoardRev       (void) ;           // Deprecated
extern      void  piBoardId       (int *model, int *rev, int *mem,
int *maker, int *overVolted) ;
extern      int  wpiPinToGpio    (int wpiPin) ;
extern      int  physPinToGpio   (int physPin) ;
extern      void  setPadDrive    (int group, int value) ;
extern      int  getAlt          (int pin) ;
extern      void  pwmToneWrite   (int pin, int freq) ;
extern      void  pwmSetMode     (int mode) ;
extern      void  pwmSetRange    (unsigned int range) ;
extern      void  pwmSetClock    (int divisor) ;
extern      void  gpioClockSet   (int pin, int freq) ;
extern unsigned int  digitalReadByte (void) ;
extern unsigned int  digitalReadByte2 (void) ;
extern      void  digitalWriteByte (int value) ;
extern      void  digitalWriteByte2 (int value) ;

// Interrupts
//      (Also Pi hardware specific)
extern int waitForInterrupt (int pin, int mS) ;
extern int  wiringPiISR      (int pin, int mode, void (*function)(void)) ;

// Threads
extern int  piThreadCreate    (void *(*fn)(void *)) ;
extern void piLock            (int key) ;
extern void piUnlock          (int key) ;

// Schedulling priority
extern int piHiPri (const int pri) ;
// Extras from arduino land

```

```
extern void delay (unsigned int howLong) ;
extern void delayMicroseconds (unsigned int howLong) ;
extern unsigned int millis (void) ;
extern unsigned int micros (void) ;
#ifndef __cplusplus
#endif
#endif
```

/usr/include/wiringPiI2C.h

```
pi@raspberrypi:~/my $ cat /usr/include/wiringPiI2C.h
/*
 * wiringPiI2C.h:
 *      Simplified I2C access routines
 *      Copyright (c) 2013 Gordon Henderson
 ****
 * This file is part of wiringPi:
 *      https://projects.drogon.net/raspberry-pi/wiringpi/
 *
 * wiringPi is free software: you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 *
 * wiringPi is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty
 * of
 *      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the
 *      GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with wiringPi.
 * If not, see <http://www.gnu.org/licenses/>.
 ****
 */
#ifndef __cplusplus
extern "C" {
#endif
```

```

extern int wiringPi2CRead           (int fd) ;
extern int wiringPi2CReadReg8      (int fd, int reg) ;
extern int wiringPi2CReadReg16     (int fd, int reg) ;
extern int wiringPi2CWrite         (int fd, int data) ;
extern int wiringPi2CWriteReg8     (int fd, int reg, int data) ;
extern int wiringPi2CWriteReg16    (int fd, int reg, int data) ;
extern int wiringPi2CSetupInterface (const char *device, int devId) ;
extern int wiringPi2CSetup        (const int devId) ;
#endif
#endif

```

/usr/include/wiringPiSPI.h

```

pi@raspberrypi:~/my $ cat /usr/include/wiringPiSPI.h
/*
 * wiringPiSPI.h:
 *   Simplified SPI access routines
 *   Copyright (c) 2012-2015 Gordon Henderson
 ****
 * This file is part of wiringPi:
 *   https://projects.drogon.net/raspberry-pi/wiringpi/
 *
 * wiringPi is free software: you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 *
 * wiringPi is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty
 * of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the
 *   GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with wiringPi.
 * If not, see <http://www.gnu.org/licenses/>.
 ****
 */

```

```
#ifdef __cplusplus
extern "C" {
#endif
int wiringPiSPISetupFd      (int channel) ;
int wiringPiSPIDataRW     (int channel, unsigned char *data, int len) ;
int wiringPiSPISetupMode   (int channel, int speed, int mode) ;
int wiringPiSPISetup      (int channel, int speed) ;
#ifndef __cplusplus
#endif
```

/usr/include/wiringSerial.h

```
pi@raspberrypi:~/my $ cat /usr/include/wiringSerial.h
/*
 * wiringSerial.h:
 *      Handle a serial port
 ****
 * This file is part of wiringPi:
 *      https://projects.drogon.net/raspberry-pi/wiringpi/
 *
 *      wiringPi is free software: you can redistribute it and/or modify
 *      it under the terms of the GNU Lesser General Public License as
 *      published by
 *          the Free Software Foundation, either version 3 of the License, or
 *          (at your option) any later version.
 *
 *      wiringPi is distributed in the hope that it will be useful,
 *      but WITHOUT ANY WARRANTY; without even the implied warranty
 *      of
 *          MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the
 *      GNU Lesser General Public License for more details.
 *
 *      You should have received a copy of the GNU Lesser General Public
 *      License along with wiringPi. If not, see <http://www.gnu.org/licenses/>.
 ****
 */
#ifndef __cplusplus
```

```

extern "C" {
#endif
extern int serialOpen    (const char *device, const int baud) ;
extern void serialClose (const int fd) ;
extern void serialFlush (const int fd) ;
extern void serialPutchar (const int fd, const unsigned char c) ;
extern void serialPuts   (const int fd, const char *s) ;
extern void serialPrintf (const int fd, const char *message, ...) ;
extern int serialDataAvail (const int fd) ;
extern int serialGetchar (const int fd) ;
#ifndef __cplusplus
#endif
}

```

/usr/include/wiringShift.h

```

/*
 * wiringShift.h:
 *     Emulate some of the Arduino wiring functionality.
 *
 * Copyright (c) 2009-2012 Gordon Henderson.
 ****
 * This file is part of wiringPi:
 *     https://projects.drogon.net/raspberry-pi/wiringpi/
 *
 * wiringPi is free software: you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as
 * published by
 *     the Free Software Foundation, either version 3 of the License, or
 *     (at your option) any later version.
 *
 * wiringPi is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY: without even the implied warranty
 * of
 *     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the
 *     GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
License

```

```
*      along with wiringPi. If not, see <http://www.gnu.org/licenses/>.  
*****  
*/  
#define LSBFIRST      0  
#define MSBFIRST      1  
#ifndef _STDINT_H  
# include <stdint.h>  
#endif  
#ifdef __cplusplus  
extern "C" {  
#endif  
extern uint8_t shiftIn     (uint8_t dPin, uint8_t cPin, uint8_t order) ;  
extern void      shiftOut    (uint8_t dPin, uint8_t cPin, uint8_t order,  
                           uint8_t val) ;  
#ifdef __cplusplus  
}#endif
```

/usr/include/softTone.h

```
pi@raspberrypi:~/my $ cat /usr/include/softTone.h  
/*  
 * softTone.c:  
 *      For that authentic retro sound...  
 *      Er... A little experiment to produce tones out of a Pi using  
 *      one (or 2) GPIO pins and a piezeo "speaker" element.  
 *      (Or a high impedance speaker, but don'y blame me if you  
 blow-up  
 *      the GPIO pins!)  
 *      Copyright (c) 2012 Gordon Henderson  
*****  
* This file is part of wiringPi:  
*      https://projects.drogon.net/raspberry-pi/wiringpi/  
*  
*      wiringPi is free software: you can redistribute it and/or modify  
*      it under the terms of the GNU Lesser General Public License as  
*      published by the Free Software Foundation, either version 3 of the
```

```
*      License, or (at your option) any later version.  
*  
*      wiringPi is distributed in the hope that it will be useful,  
*      but WITHOUT ANY WARRANTY; without even the implied warranty  
of  
*          MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

See the

```
*      GNU Lesser General Public License for more details.  
*  
*      You should have received a copy of the GNU Lesser General Public  
*      License along with wiringPi.  
*      If not, see <http://www.gnu.org/licenses/>.
```

```
*****  
*/  
#ifdef __cplusplus  
extern "C" {  
#endif  
extern int softToneCreate (int pin) ;  
extern void softToneStop (int pin) ;  
extern void softToneWrite (int pin, int freq) ;  
#ifdef __cplusplus  
}#endif
```

