

# IoT Platform 6<sup>th</sup> Week

## - Interfacing to RPi I<sup>2</sup>C and SPI -

Jaeseok Yun

Soonchunhyang University

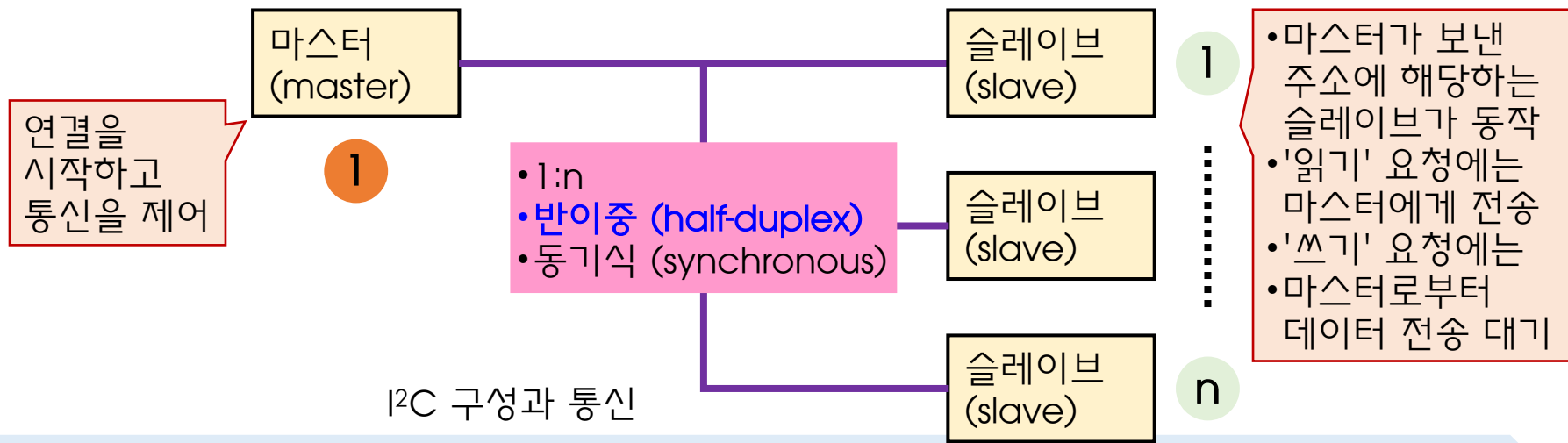
# I<sup>2</sup>C 통신 (리뷰)

임베디드 시스템 (아두이노)

- 필립스 반도체 (현재, NXT 반도체) 에서 개발한 **산업 (de facto) 표준**<sup>3</sup>
- 인텔이 개발한 SMBus 등 변형 표준도 있음
- 장치마다 정확한 사용을 위해 **데이터 시트 확인!**
- 아이-스퀘어드-씨 (I-squared-C), 아이-투-씨

■ Inter-Integrated Circuit<sup>1,2</sup>

- 마스터 (master) 장치와 하나 이상의 슬레이브 (slave) 장치로 구성
  - ✓ 1 (마스터) : n (슬레이브) 통신이 가능
- **송신과 수신**이 동시에 진행될 수 없는 반이중 (half-duplex) 통신 표준
- 소프트웨어로 슬레이브의 주소를 지정해서 1:n 통신 구현 (SPI와 비교)
- 마이크로프로세서와 주변장치 (peripheral) 간 '저속, 근거리' 통신 (버스, bus<sup>4</sup>)에서 많이 활용
  - ✓ 예, 다양한 센서, LCD 디스플레이 등

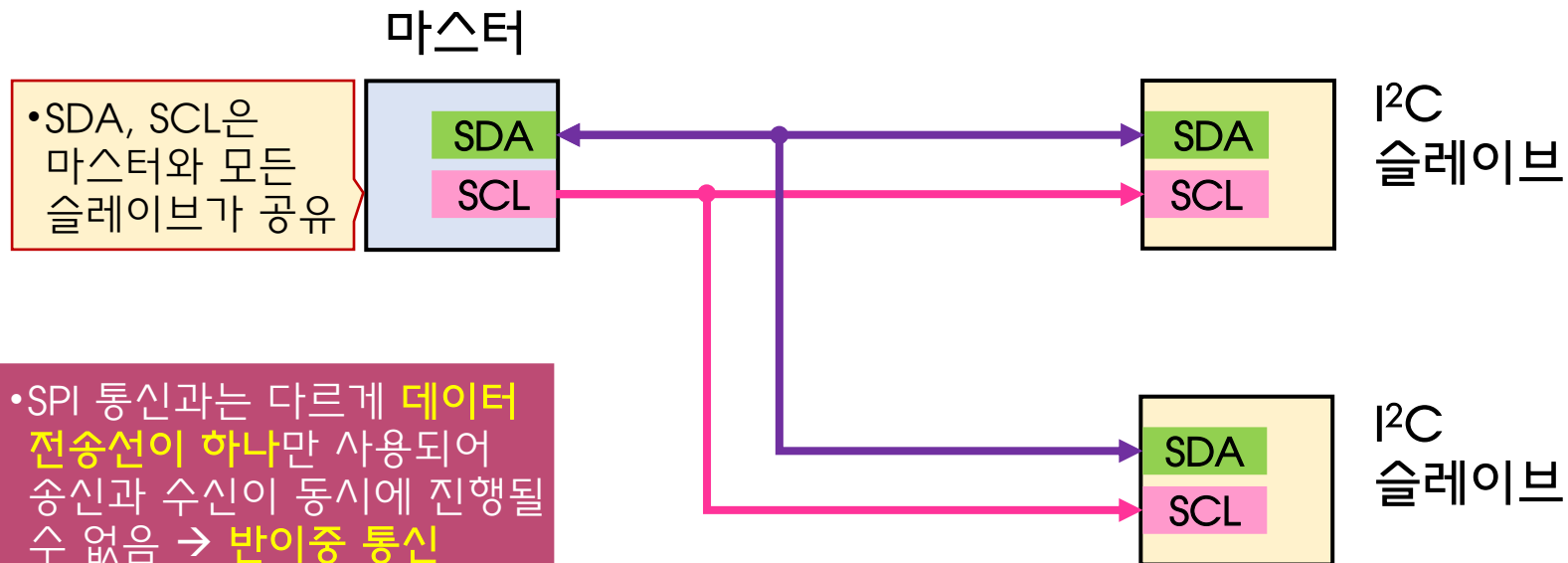


<sup>1</sup> <https://en.wikipedia.org/wiki/I%C2%B2C>  
<sup>2</sup> <https://learn.sparkfun.com/tutorials/i2c/all>  
<sup>3</sup> de facto 표준이란 대중성과 시장 장악력을 바탕으로 어느 회사나 단체의 특정 방식 (예, 제조, 통신 등)이 표준 처럼 인정되는 것, 예로서 HDMI 포트  
<sup>4</sup> 컴퓨터 구조에서 '버스'란 컴퓨터 내부에서 컴포넌트 사이에 또는 컴퓨터 사이에 데이터를 주고 받는 통신 시스템을 통칭하는 말

# I<sup>2</sup>C Inter-Integrated Circuit

## ■ I<sup>2</sup>C 연결선 (wire)과 연결 방법

- SCL (Serial CLock): 마스터-슬레이브 장치 간 **동기화를 위한 클럭** 전송
- SDA (Serial DAta): 마스터-슬레이브 장치 간 **주고 받는 데이터** 전송



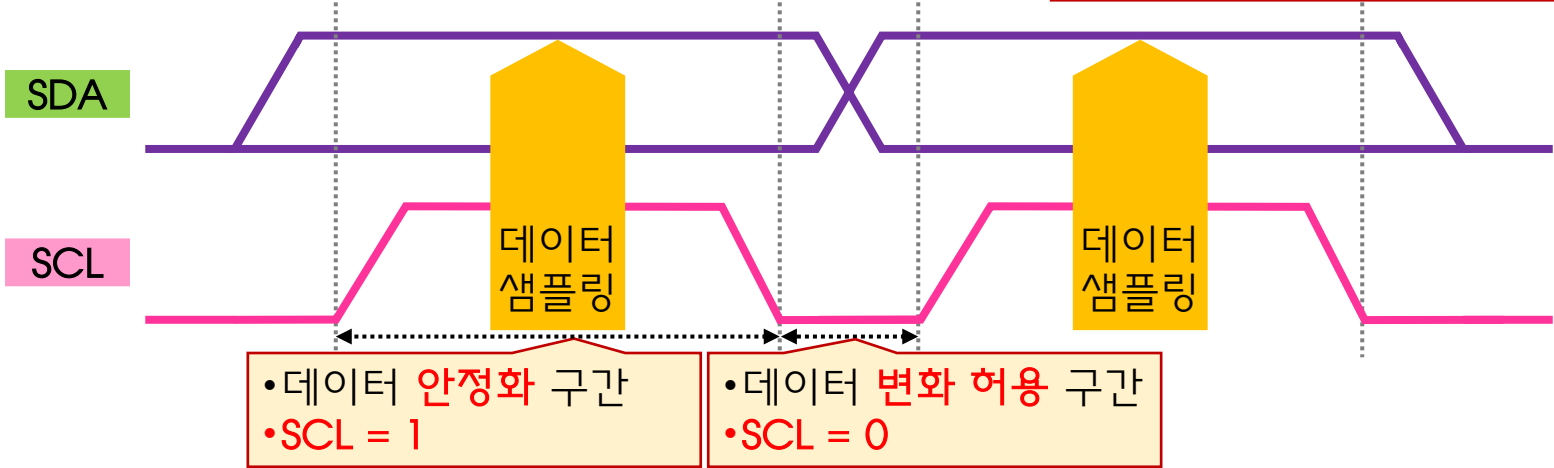
- SPI 통신과는 다르게 **데이터 전송선이 하나만** 사용되어 송신과 수신이 동시에 진행될 수 없음 → **반이중 통신**
- SPI 통신의 **SS**를 대신하여 **슬레이브를 지정**할 수 있는 방법이 필요 → **주소**

I<sup>2</sup>C 통신 연결 방법 (1:2)

# I<sup>2</sup>C 데이터 전송 방식<sup>1</sup>

## ■ 데이터 샘플링과 변화 허용

- SCL = HIGH, 수신 데이터 샘플링
- SCL = LOW, 데이터 변화



## ■ 데이터 전송의 시작과 종료

- SCL = HIGH, SDA 변화하는 경우 데이터 전송의 시작과 종료

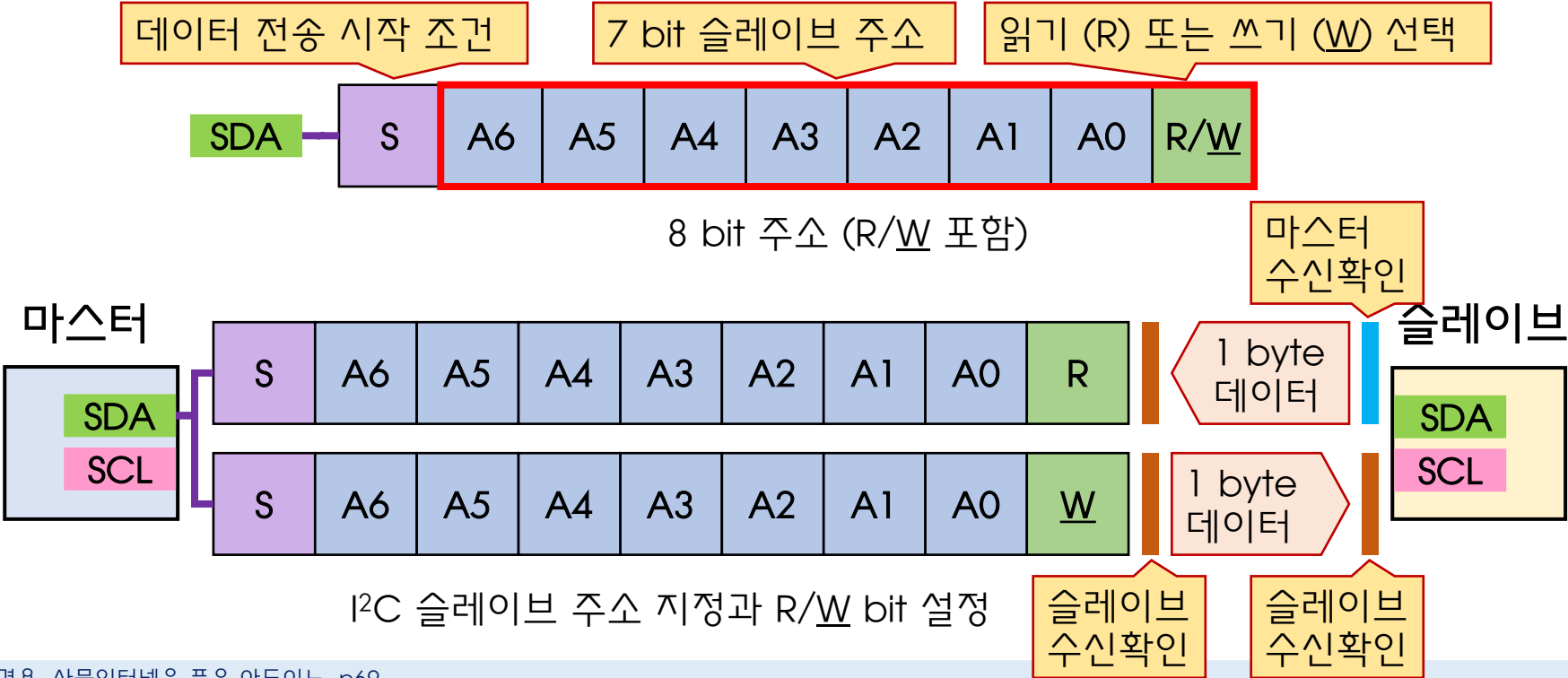


<sup>1</sup> 허경용, 사물인터넷을 품은 아두이노, p69

# I<sup>2</sup>C 데이터 전송 방식<sup>1</sup>

## ■ 슬레이브 주소 지정

- 슬레이브 주소 (A6-A0, 7 bit) + 읽고/쓰기 (R/W, 1 bit)
  - ✓ 슬레이브 주소 + **R**: '슬레이브에게 데이터 전송을 '요청 (requesting data)'
    - 슬레이브는 1 byte 데이터를 마스터로 전송
  - ✓ 슬레이브 주소 + **W**: '슬레이브에게 데이터 전송을 '알림 (sending data)'
    - 슬레이브는 1 byte 데이터를 마스터로부터 수신할 준비

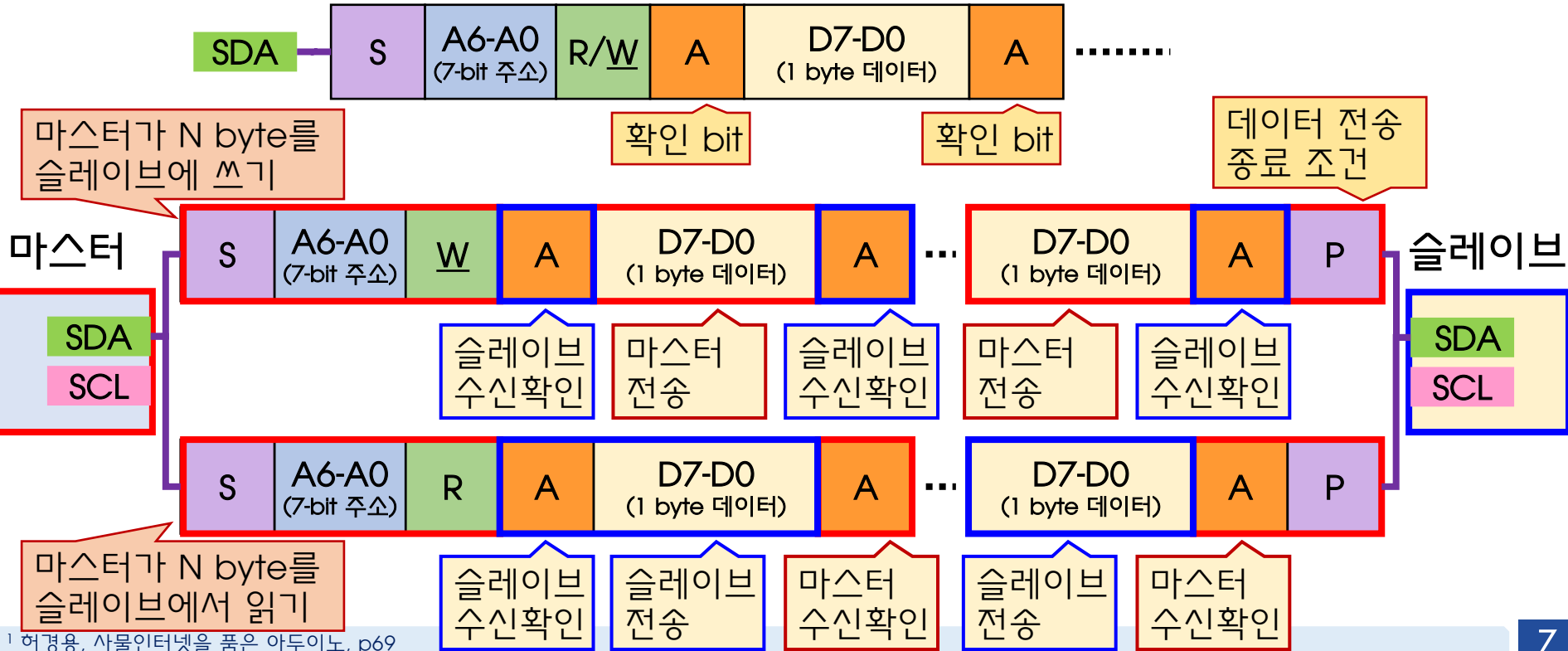


<sup>1</sup> 허경용, 사물인터넷을 품은 아두이노, p69

# I<sup>2</sup>C 데이터 전송 방식<sup>1</sup>

## ■ 데이터 수신 여부 확인 (ACK/NACK)

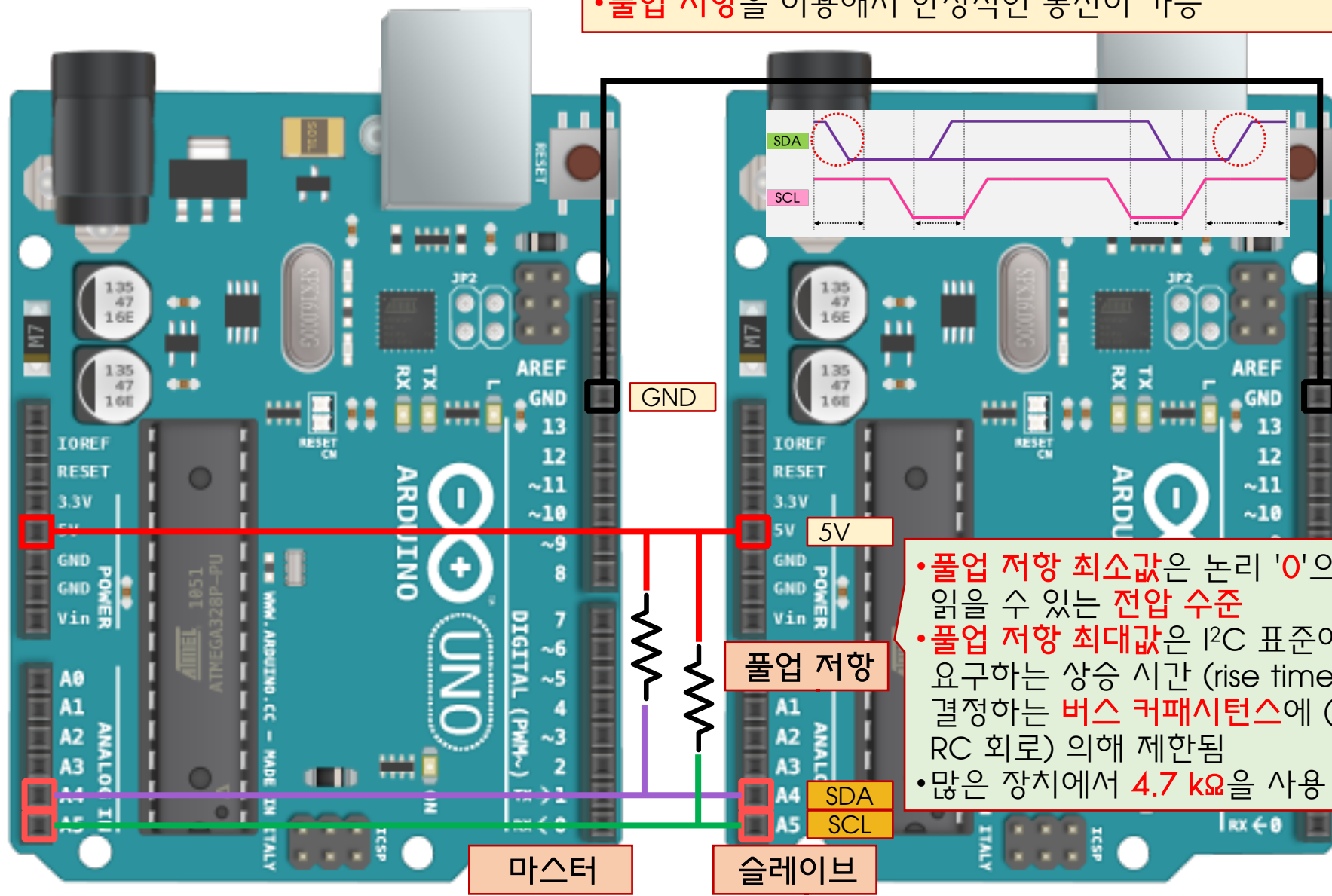
- 수신 측은 데이터를 잘 받았음을 송신 측에 알리기 위해 1 bit를 전송
  - ✓ 잘 받았을 때 '오류 없음 (acknowledgement)'을 의미하는 **ACK (LOW)** 전송
  - ✓ 수신 과정에서 문제가 있었을 때 '오류 있음 (negative acknowledgement)'을 의미하는 **NACK (HIGH)** 전송
- 주소 (1 byte)와 데이터 (1 byte) 전송 때 모두 수신측은 확인 bit 전송



<sup>1</sup> 허경용, 사물인터넷을 품은 아두이노, p69

# 아두이노 우노 I<sup>2</sup>C 예약 핀 번호

- SCL과 SDA 은 사용되지 않는 경우 HIGH를 유지
- I<sup>2</sup>C는 출력단의 내부가 '오픈 드레인' 구조 (미완성 상태) 이므로 다른 회로와 연결되지 않은 경우 **플로팅** 상태
- **풀업 저항**을 이용해서 안정적인 통신이 가능



- **풀업 저항 최소값**은 논리 '0'으로 읽을 수 있는 **전압 수준**
- **풀업 저항 최대값**은 I<sup>2</sup>C 표준에서 요구하는 상승 시간 (rise time)을 결정하는 **버스 커패시턴스**에 (즉, RC 회로) 의해 제한됨
- 많은 장치에서 **4.7 kΩ**을 사용

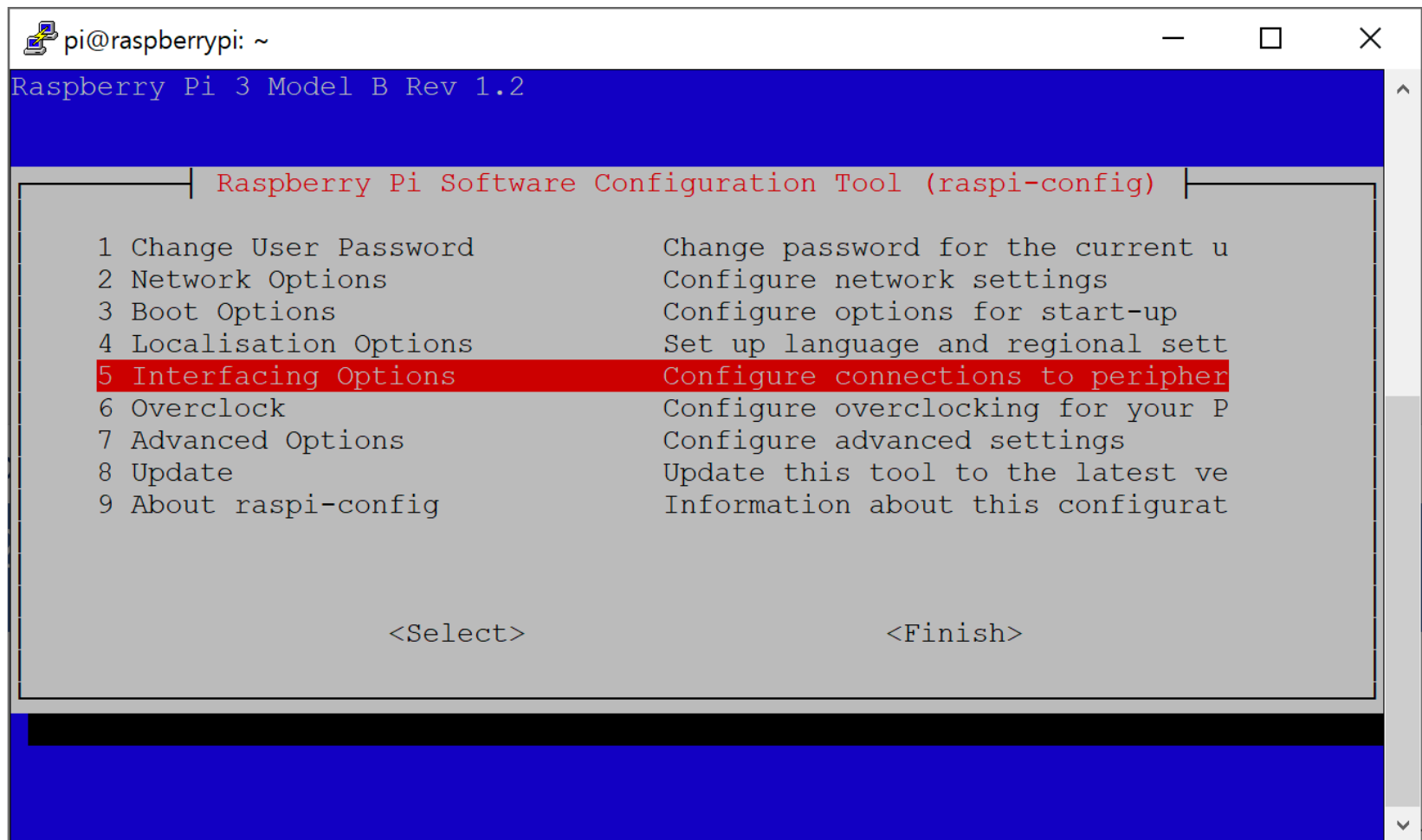


# RPi I<sup>2</sup>C 설정

raspi-config

# RPI I<sup>2</sup>C 활성화<sup>1</sup>

## ■ sudo raspi-config

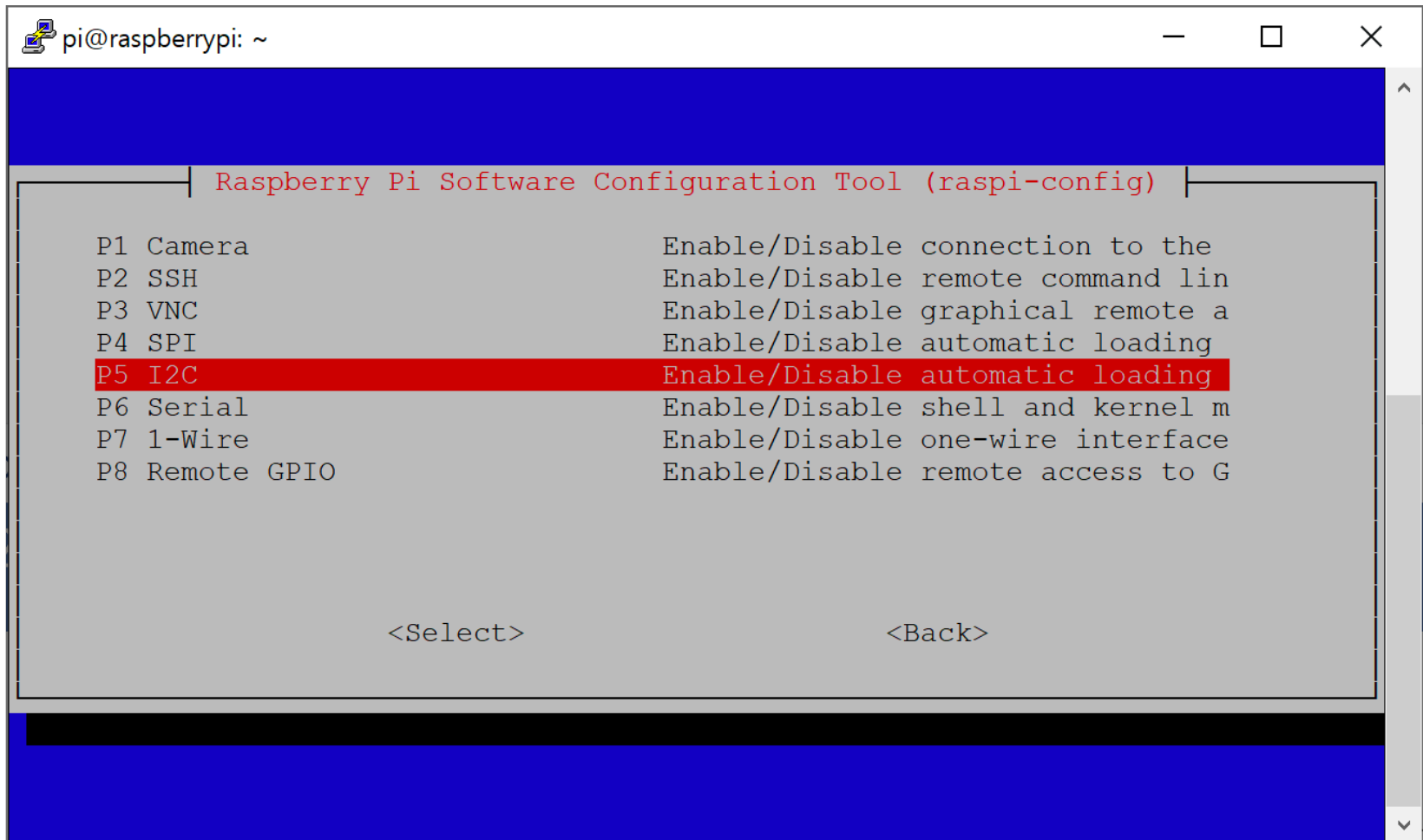


```
pi@raspberrypi: ~  
Raspberry Pi 3 Model B Rev 1.2  
  
Raspberry Pi Software Configuration Tool (raspi-config)  
  
1 Change User Password      Change password for the current u  
2 Network Options           Configure network settings  
3 Boot Options              Configure options for start-up  
4 Localisation Options      Set up language and regional sett  
5 Interfacing Options        Configure connections to peripher  
6 Overclock                 Configure overclocking for your P  
7 Advanced Options          Configure advanced settings  
8 Update                   Update this tool to the latest ve  
9 About raspi-config        Information about this configurat  
  
      <Select>                      <Finish>
```

<sup>1</sup> 익스플로링 라즈베리 파이, p335

# RPI I<sup>2</sup>C 활성화<sup>1</sup>

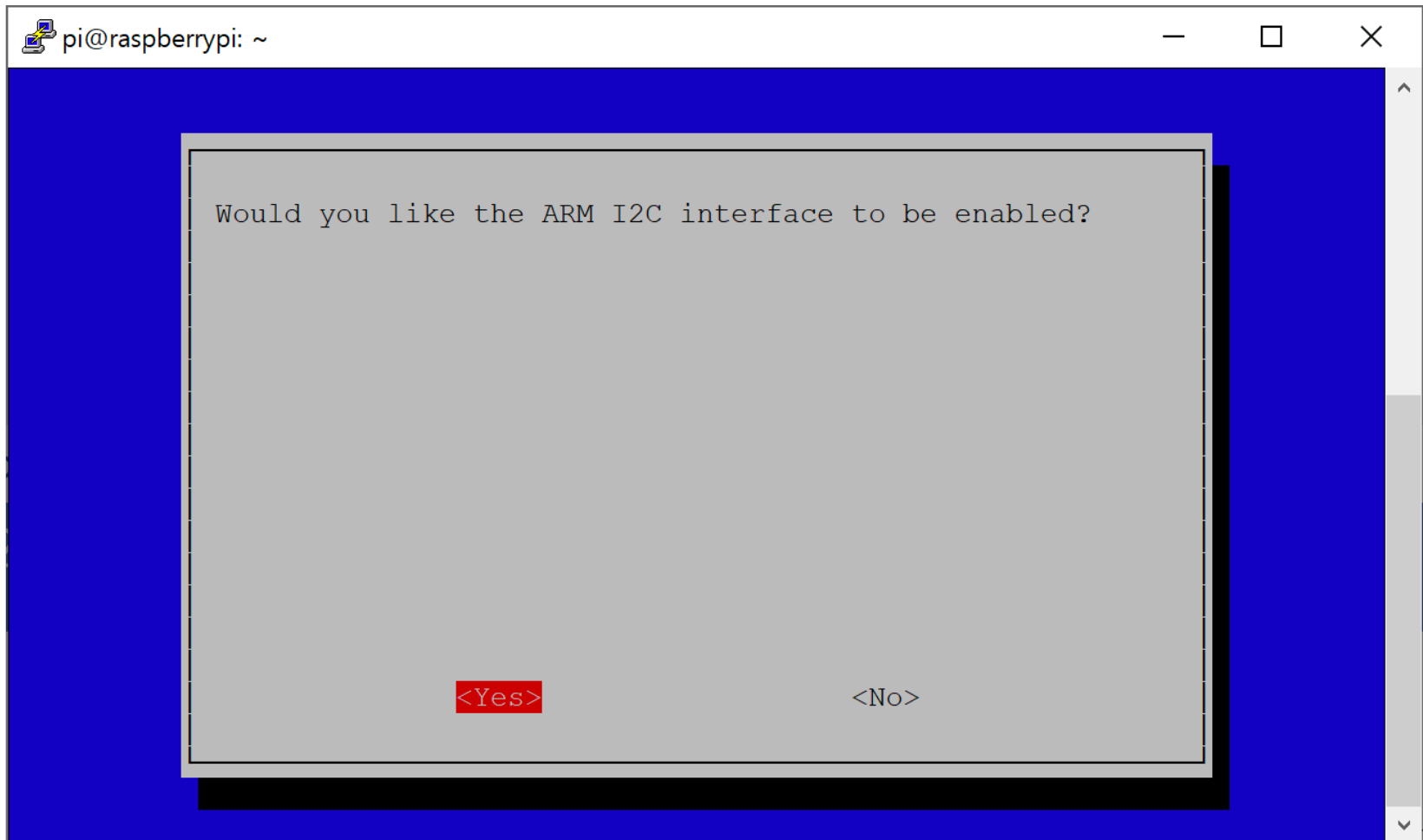
## ■ sudo raspi-config



<sup>1</sup> 익스플로링 라즈베리 파이, p335

# RPI I<sup>2</sup>C 활성화<sup>1</sup>

## ■ sudo raspi-config



<sup>1</sup> 익스플로링 라즈베리 파이, p335

# RPI I<sup>2</sup>C 활성화<sup>1</sup>

## ■ sudo raspi-config



<sup>1</sup> 익스플로링 라즈베리 파이, p335

# RPI I<sup>2</sup>C 활성화<sup>1</sup>

## ■ I<sup>2</sup>C 모듈과 인터페이스 사용 가능 확인

```
pi@raspberrypi: ~  
pi@raspberrypi:~ $ sudo raspi-config  
pi@raspberrypi:~ $ more /boot/config.txt | grep i2c_arm  
dtparam=i2c_arm=on  
pi@raspberrypi:~ $
```

- I<sup>2</sup>C 활성화 확인
- /boot/config.txt<sup>2</sup> 직접 수정 가능

<sup>1</sup> 익스플로링 라즈베리 파이, p335

<sup>2</sup> RPi가 부팅할 때 참고하는 설정 파일로서 일반 PC의 BIOS 역할을 대신함

# RPI I<sup>2</sup>C 활성화<sup>1</sup>

## ■ sudo nano /boot/config.txt

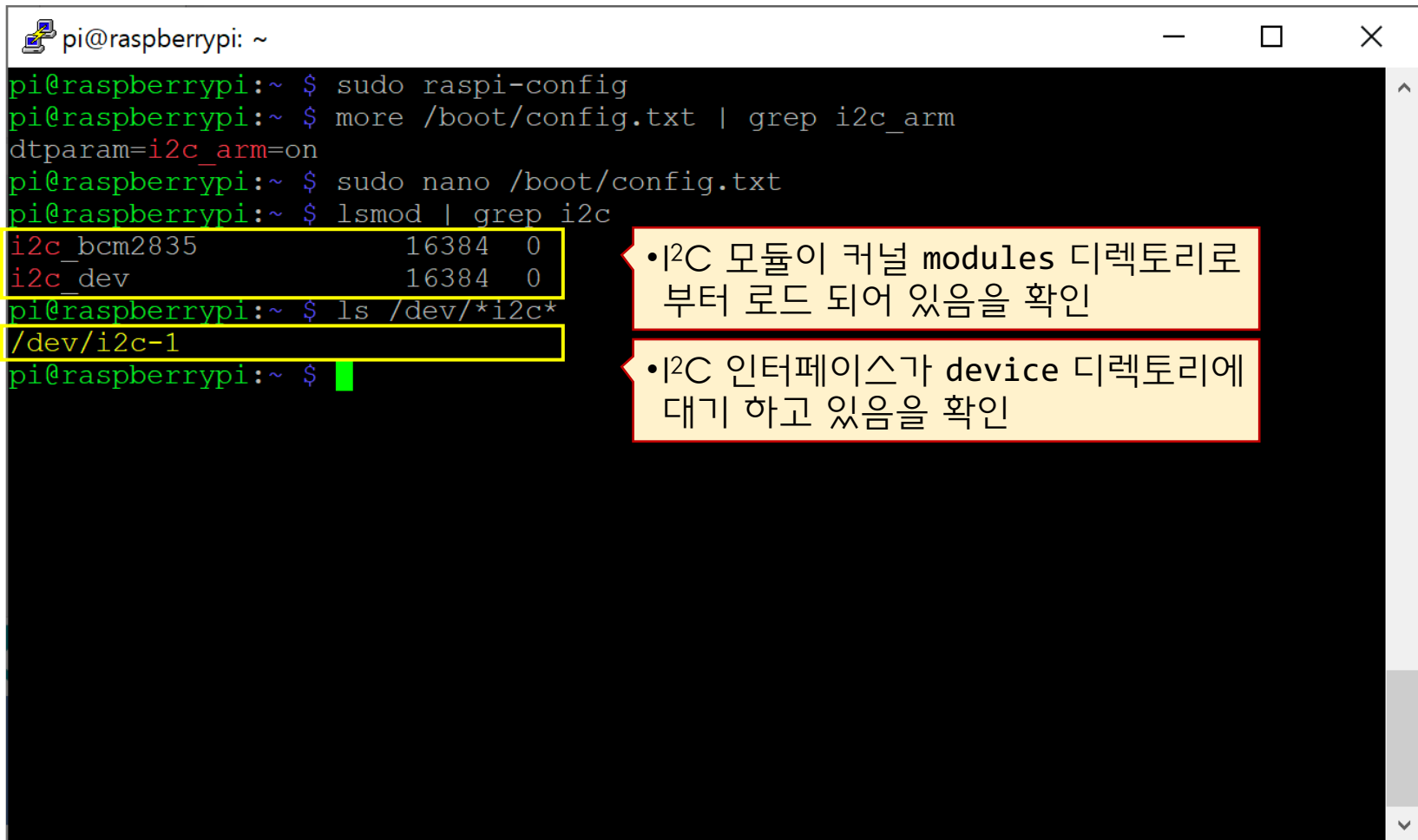
```
pi@raspberrypi: ~  
GNU nano 2.7.4 File: /boot/config.txt  
# uncomment to increase signal to HDMI, if you have interference, blanking, or  
# no display  
#config_hdmi_boost=4  
  
# uncomment for composite PAL  
#sdtv_mode=2  
  
#uncomment to overclock the arm. 700 MHz is the default.  
#arm_freq=800  
  
# Uncomment some or all optional hardware interfaces  
dtparam=i2c arm=on  
#dtparam=i2s=on  
#dtparam=spi=on  
  
# Uncomment this to enable the lirc-rpi module  
#dtoverlay=lirc-rpi  
  
# Additional overlays and parameters are documented /boot/overlays/README  
[ File '/boot/config.txt' is unwritable ]  
^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 ^_ Go To Line
```

<sup>1</sup> 익스플로링 라즈베리 파이, p335

<sup>2</sup> RPI가 부팅할 때 참고하는 설정 파일로서 일반 PC의 BIOS 역할을 대신함

# RPI I<sup>2</sup>C 활성화<sup>1</sup>

## ■ I<sup>2</sup>C 모듈과 인터페이스 사용 가능 확인



```
pi@raspberrypi: ~  
pi@raspberrypi:~ $ sudo raspi-config  
pi@raspberrypi:~ $ more /boot/config.txt | grep i2c_arm  
dtparam=i2c_arm=on  
pi@raspberrypi:~ $ sudo nano /boot/config.txt  
pi@raspberrypi:~ $ lsmod | grep i2c  
i2c_bcm2835      16384  0  
i2c_dev         16384  0  
pi@raspberrypi:~ $ ls /dev/*i2c*  
/dev/i2c-1  
pi@raspberrypi:~ $
```

- I<sup>2</sup>C 모듈이 커널 modules 디렉토리로 부터 로드 되어 있음을 확인
- I<sup>2</sup>C 인터페이스가 device 디렉토리에 대기 하고 있음을 확인

<sup>1</sup> 익스플로링 라즈베리 파이, p335



# RPi GPIO 핀 헤더<sup>1,3</sup>

I<sup>2</sup>C0과  
I<sup>2</sup>C1  
비교표<sup>2</sup>

하드웨어 버스	SW 장치	SDA 핀	SCL 핀	설명
I2C1	/dev/i2c-1	3	5	일반적인 I2C 버스
I2C0	/dev/i2c-0	27	28	HAT 관리를 위해 예약된 I2C 버스



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

<sup>1</sup> <http://www.raspberrypi-geek.com/howto/GPIO-Pinout-Rasp-Pi-1-Model-B-Rasp-Pi-2-Model-B>  
<sup>2</sup> 익스플로링 라즈베리 파이, p337  
<sup>3</sup> <https://pinout.xyz/pinout/>

RPI I<sup>2</sup>C 확인 도구<sup>1</sup>

## Practice

```
pi@raspberrypi: ~  
pi@raspberrypi:~ $ apt-cache policy i2c-tools  
i2c-tools:  
  Installed: 3.1.2-3  
  Candidate: 3.1.2-3  
  Version table:  
*** 3.1.2-3 500  
    500 http://raspbian.raspberrypi.org/raspbian stretch/main armhf Packages  
    100 /var/lib/dpkg/status  
pi@raspberrypi:~ $  
pi@raspberrypi:~ $  
pi@raspberrypi:~ $  
pi@raspberrypi:~ $  
pi@raspberrypi:~ $  
pi@raspberrypi:~ $  
pi@raspberrypi:~ $  
pi@raspberrypi:~ $ i2cdetect -l  
i2c-1    i2c                bcm2835 I2C adapter          I2C adapter  
pi@raspberrypi:~ $
```

• 설치 버전 확인!

• 리눅스에서 I<sup>2</sup>C 버스 장치와 인터페이스 테스트 툴 'i2c-tools' 설치 확인  
• 만약 설치되어 있지 않다면 (none)  
\$ sudo apt install i2c-tools

• 활성화 되어 있는 I<sup>2</sup>C 버스 확인  
• 일반 I<sup>2</sup>C 버스인 I2C1 활성화 확인

# 가속도 센서

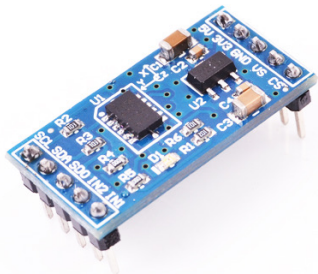
## ADXL345

# 가속도 센서 동작 원리

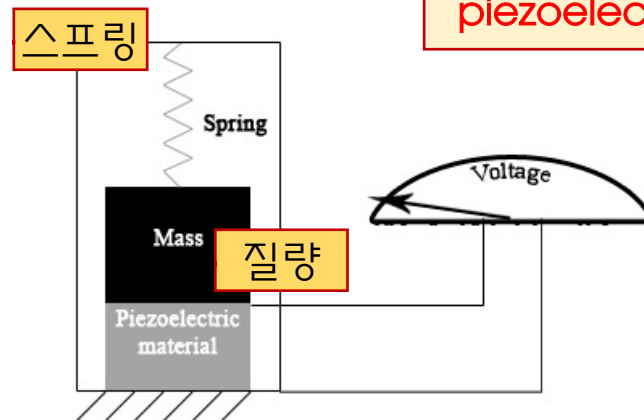
## ■ ADXL345<sup>1</sup>

- 3축 가속도 센서: 고정 10-bit 측정 해상도 ( $\pm 16$  g일 때 13-bit 까지 가능)
- 디지털 인터페이스: I<sup>2</sup>C, SPI
- 감지 범위:  $\pm 2/\pm 4/\pm 8/\pm 16$  g (중력가속도), 조정 가능
- 동작 전압: 2-3.6 V

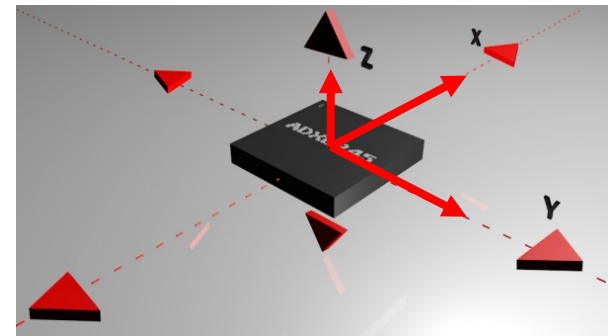
• 가속도 센서는 기계 동작을 전기 신호로 변환하여 정적 가속도 (예, 중력)와 동적 가속도를 측정  
 • 가속도는 3축으로 표현, 3축 센서가 많이 사용  
 • 기계 동작 (즉 힘)을 전기 신호로 바꾸는 물질:  
**piezoelectric**, piezoresistive, capacitive



ADXL345 모듈<sup>2</sup>



가속도 센서 동작 원리<sup>3</sup>



3축 가속도 센서 측정<sup>3</sup>

<sup>1</sup> <https://www.analog.com/en/products/adxl345.html#product-overview>

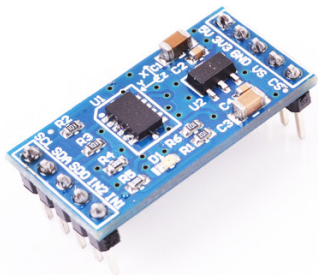
<sup>2</sup> <http://www.eleparts.co.kr/goods/view?no=3039995>

<sup>3</sup> <https://learn.sparkfun.com/tutorials/accelerometer-basics/all>

# 가속도 센서 회로 구성

## ■ ADXL345<sup>1</sup>

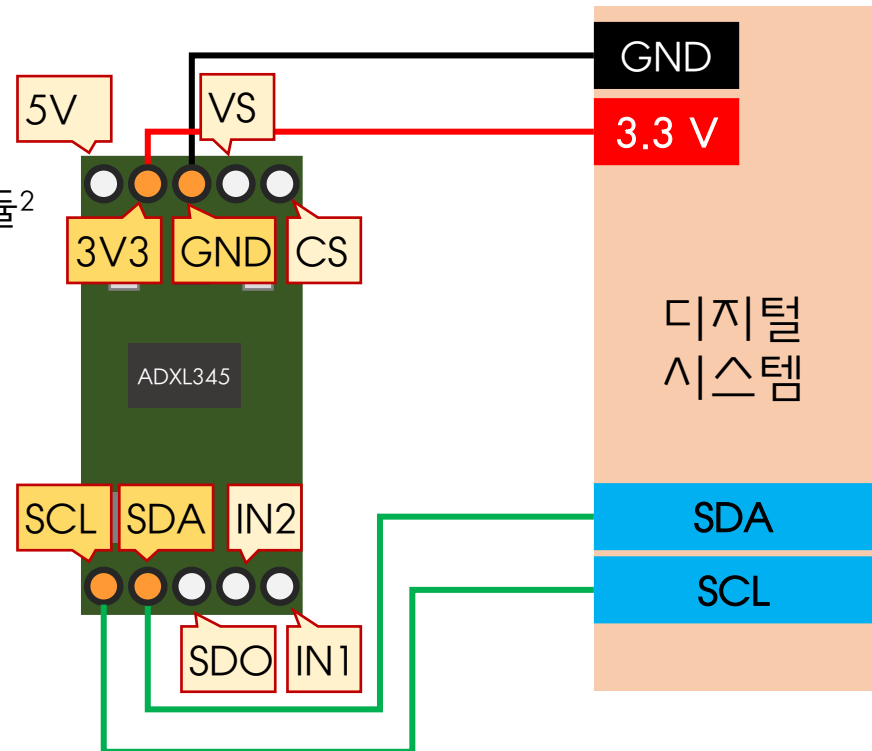
- 3축 가속도 센서: 고정 10-bit 측정 해상도 ( $\pm 16$  g일 때 13-bit 까지 가능)
- 디지털 인터페이스: I<sup>2</sup>C, SPI
- 감지 범위:  $\pm 2/\pm 4/\pm 8/\pm 16$  g (중력가속도), 조정 가능
- 동작 전압: 2-3.6 V



ADXL345 모듈<sup>2</sup>

ADXL345 모듈<sup>2</sup>

- Analog Devices 는 ADXL345를 생산
- 여러 회사가 (예, SparkFun<sup>3</sup>) ADXL345 모듈을 생산
- 회로 구성은 해당 모듈 매뉴얼 참고



<sup>1</sup> <https://www.analog.com/en/products/adxl345.html#product-overview>

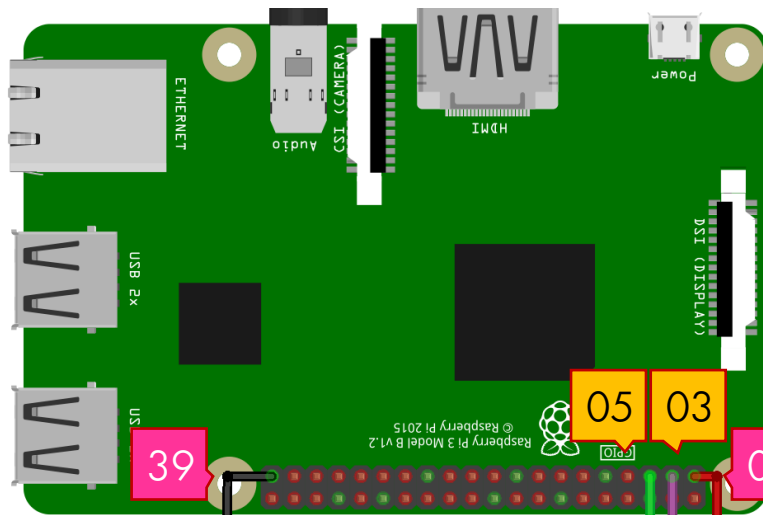
<sup>2</sup> <http://www.eleparts.co.kr/goods/view?no=3039995>

<sup>3</sup> <https://learn.sparkfun.com/tutorials/accelerometer-basics/all>

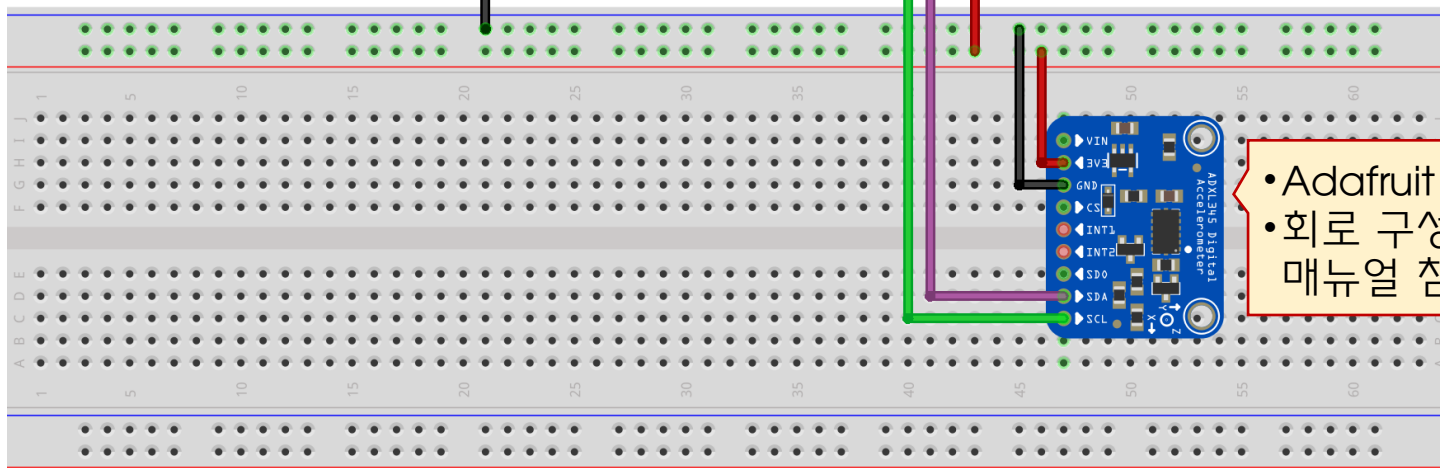
## 가속도 센서 회로 구성

## Practice

## ■ 가속도 센서와 RPi 연결



- (4핀) 점퍼 케이블을 이용하여 그림과 같이 연결
- 정면에서 보았을 때 (**데이터시트 확인!**),  
GND 핀, RPi GND 연결  
Vcc 핀, RPi 3.3V 연결  
SDA 핀, **RPi P03** (GPIO02) 연결  
SCL 핀, **RPi P05** (GPIO03) 연결
- **RPi 1번** (VCC), **RPi 39번** (GND) 구성



- Adafruit ADXL345 모듈
- 회로 구성은 해당 모듈 매뉴얼 참고

# 가속도 센서 테스트<sup>1</sup>

Practice

```
pi@raspberrypi: ~
pi@raspberrypi:~ $ i2cdetect -y 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --

pi@raspberrypi:~ $ i2cdump -y 1 0x53 b
    0  1  2  3  4  5  6  7  8  9  a  b
00: e5 00 00 00 00 00 00 00 00 00 00 00
10: 82 00 00 00 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00
30: 82 00 0d 01 00 00 15 00 00 00 00 00
40: e5 00 00 00 00 00 00 00 00 00 00 00
50: 00 00 00 00 00 00 00 00 00 00 00 00
60: 0a 08 00 00 00 00 00 00 00 00 00 00
70: 00 00 00 00 00 00 00 00 00 00 00 00
80: 00 00 00 4a 00 00 00 00 00 00 00 00
90: 00 00 00 00 00 00 00 00 00 00 00 00
a0: 00 00 00 00 00 00 00 00 00 00 00 00
b0: 00 00 00 00 00 00 00 00 00 00 00 00
c0: 00 4a 00 00 00 00 00 00 00 00 00 00
d0: 00 00 00 00 00 00 00 00 00 00 00 00
e0: 00 00 00 00 00 00 00 00 00 00 0a 08
f0: 82 00 0b 01 00 00 15 00 00 00 00 00

pi@raspberrypi:~ $
```

- I<sup>2</sup>C 버스에 연결된 장치를 감지하여 **7-bit 주소** (0x00-0x7F)를 출력
- -y: 경고 무시 (인터랙티브 모드 비활성화)

- I<sup>2</sup>C 버스에 감지된 장치의 주소
- **ADXL345**는 제조사에서 주소를 **0x53**으로 고정

- I<sup>2</sup>C 버스에 연결된 특정 주소 (예, 0x53) **장치의 레지스터 값을 읽어서** 16진수 블록 형식으로 보임
- 종속 장치에 기록할 수 있으므로 주의

센서 이동 후 가속도 값 변화 확인

- 데이터 포맷 (데이터 표현 방법)
- ADXL345 데이터 시트 참조<sup>2</sup>
- i2cset 설정, i2cget 확인 가능<sup>1</sup>

**Register 0x31—DATA\_FORMAT (Read/Write)**

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	



# 가속도 센서 리눅스 프로그래밍 (i2c.h, i2c-dev.h)

Practice

```
#include <iostream>
#include <stdio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/i2c.h>
#include <linux/i2c-dev.h>
#include <iomanip>
#include <unistd.h>
using namespace std;
```

// Small macro to display value in hexadecimal with 2 places

```
#define HEX(x) setw(2) << setfill('0') << hex << (int)(x)
```

// The ADXL345 Resisters required for this example

```
#define DEVID 0x00
#define POWER_CTL 0x2D
```

```
#define DATA_FORMAT 0x31
```

```
#define DATA_X0 0x32
#define DATA_X1 0x33
#define DATA_Y0 0x34
#define DATA_Y1 0x35
#define DATA_Z0 0x36
#define DATA_Z1 0x37
#define BUFFER_SIZE 0x40
unsigned char dataBuffer[BUFFER_SIZE];
```

- 리눅스에서 I<sup>2</sup>C 버스 드라이버로 디바이스 인터페이스를 하기 위해 GNU (general public license) 라이선스로 배포되는 무료 소프트웨어<sup>1,2</sup>
- 많은 리눅스 배포판에 기본으로 설치

- POWER\_CTL 레지스터
- 측정 모드로 설정 (0x08)

**Register 0x2D—POWER\_CTL (Read/Write)**

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

**Table 21. g Range Setting**

Setting		g Range
D1	D0	
0	0	±2 g
0	1	±4 g
1	0	±8 g
1	1	±16 g

**Register 0x31—DATA\_FORMAT (Read/Write)**

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

**Register 0x32 to Register 0x37—DATA\_X0, DATA\_X1, DATA\_Y0, DATA\_Y1, DATA\_Z0, DATA\_Z1 (Read Only)**

- DATA\_FORMAT 레지스터
- ±2 g 로 설정 (0x00)

- DATA[xyz][0] 레지스터
- x/y/z축 하위/상위 바이트 데이터

<sup>1</sup> <https://github.com/torvalds/linux/blob/master/include/linux/i2c.h>  
<sup>2</sup> <https://github.com/torvalds/linux/blob/master/include/uapi/linux/i2c-dev.h>



# 가속도 센서 리눅스 프로그래밍 (i2c.h, i2c-dev.h)

## Practice

```
int writeRegister(int file, unsigned char address, char value) {
    unsigned char buffer[2];
    buffer[0] = address;
    buffer[1] = value;
    if (write(file, buffer, 2) != 2) {
        cout << "Failed write to the device" << endl; return 1;
    }
    return 0;
}
```

- 리눅스 파일 **file**을 이용한 ADXL345 레지스터 쓰기
- 2 바이트 ([0]: 주소, [1]: 값) 쓰기

```
int readRegisters(int file){
    writeRegister(file, 0x00, 0x00);
```

- General call 주소
- 다음 읽기 요청에 데이터를 보내야 할 주소 알림 효과

```
    if (read(file, dataBuffer, BUFFER_SIZE) != BUFFER_SIZE){
        cout << "Failed to read in the full buffer." << endl;
        return 1;
    }
    if (dataBuffer[DEVID] != 0xE5){
        cout << "Problem detected! Device ID is wrong" << endl;
        return 1;
    }
    return 0;
}
```

- 리눅스 파일 **file**을 이용해 ADXL 레지스터에서 40 (BUFFER\_SIZE) 바이트 데이터를 버퍼로 읽기

- 데이터 송신자 주소 (0xE5) 확인 검증 작업

```
short combineValues(unsigned char msb, unsigned char lsb){
    //shift the msb right by 8 bits and OR with lsb
    return ((short)msb << 8) | (short)lsb;
}
```

- ADXL X, Y, Z 데이터 레지스터 상위/하위 바이트 값 합치기

# 가속도 센서

```
#define POWER_CTL 0x2D
#define DATA_FORMAT 0x31
```

Register 0x2D—POWER_CTL (Read/Write)							
D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

Register 0x31—DATA_FORMAT (Read/Write)							
D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

Table 21. g Range Setting		
Setting		
D1	D0	g Range
0	0	±2 g
0	1	±4 g
1	0	±8 g
1	1	±16 g

## Practice

```
int main(){
    cout << "Starting the ADXL345 sensor application" << endl;
    if ((file = open("/dev/i2c-1", O_RDWR)) < 0) {
        cout << "failed to open the bus" << endl; return 1;
    }
    if (ioctl(file, I2C_SLAVE, 0x53) < 0) {
        cout << "Failed to connect to the sensor" << endl; return 1;
    }
    writeRegister(file, POWER_CTL, 0x08);
    writeRegister(file, DATA_FORMAT, 0x00);
    readRegisters(file);
    cout << "The Device ID is: " << HEX(dataBuffer[DEVID]) << endl;
    cout << "The POWER_CTL mode is: " << HEX(dataBuffer[POWER_CTL]) << endl;
    cout << "The DATA_FORMAT is: " << HEX(dataBuffer[DATA_FORMAT]) << endl;
    cout << dec << endl; //reset back to decimal

    int count = 0;
    while (count < 60) {
        short x = combineValues(dataBuffer[DATA1], dataBuffer[DATA0]);
        short y = combineValues(dataBuffer[DATA5], dataBuffer[DATA4]);
        short z = combineValues(dataBuffer[DATA3], dataBuffer[DATA2]);

        cout << "X=" << x << "Y=" << y << " Z=" << z << " sample=" << count << " \r" << flush;
        usleep(1000000);
        readRegisters(file); count++;
    }
    close(file);
    return 0;
}
```

- I<sup>2</sup>C1을 접근하기 위한 리눅스 파일 시스템을 사용
- '/dev/i2c-1'를 file에 연결
- file에 0x53 주소를 가진 I<sup>2</sup>C 기기를 슬레이브로 설정

- ADXL345 전원관리를 '측정모드'로 설정
- ADXL345 측정범위를 '2 ±g'로 설정
- ADXL345 레지스터 40 바이트 읽기

• x, y, z 축 가속도 값 변환

- 1초마다 ADXL345 레지스터 읽기
- 60번 반복

## 가속도 센서 프로그램 실행

Practice

```
pi@raspberrypi: ~/exploringrpi/chp08/i2c/test
pi@raspberrypi:~/exploringrpi/chp08/i2c/test $ ./ADXL345
Starting the ADXL345 sensor application
The Device ID is: e5
The POWER_CTL mode is: 08
The DATA_FORMAT is: 00
X=243 Y=-82 Z=-84 sample=7
```

- x, y, z 축 가속도 측정값 확인
- 실제 가속도 값으로 변환 방법 개인 실습
- 예로서, x=243 은 x축 방향으로 몇 m/sec<sup>2</sup> 인가?

- wPi 라이브러리도 I<sup>2</sup>C 관련 라이브러리 제공
- 이 경우 `#include <wiringPiI2C.h>` 필요
- 교재 p350과 wPi 웹페이지를 참고하여 개인 실습
- wPi 기반 프로그램은 RPi에서만 동작함을 주의!

# SPI 통신 (리뷰)

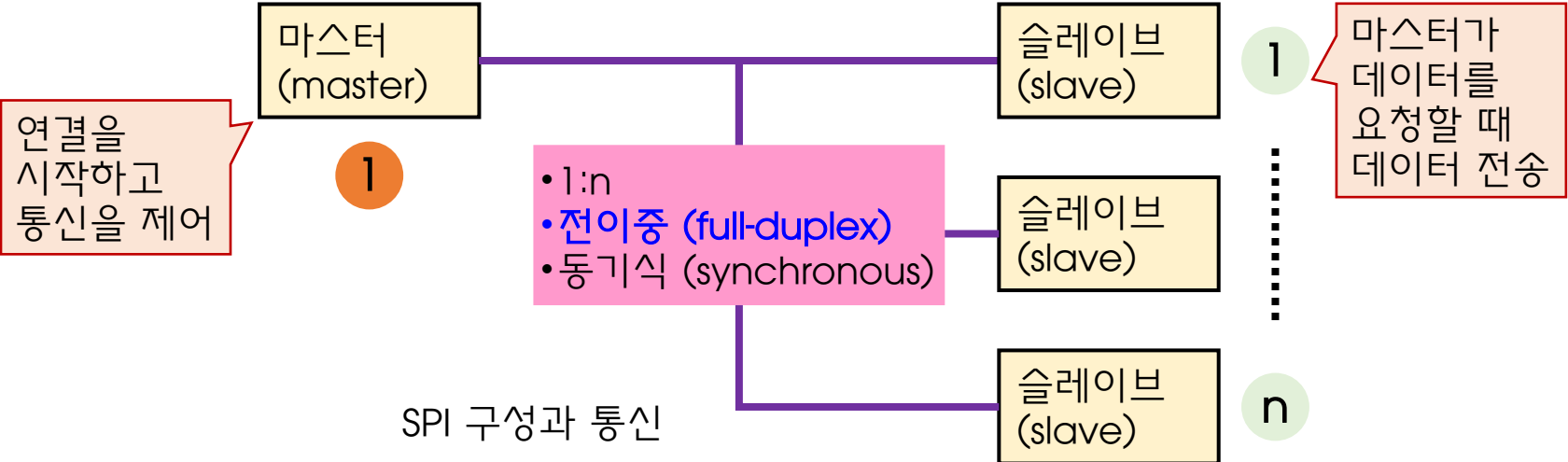
임베디드 시스템 (아두이노)

# SPI Serial Peripheral Interface

- 모토롤라에서 개발한 **산업 (de facto) 표준**<sup>3</sup>
- SPI 프로토콜은 공식적인 표준을 따르지 않기 때문에 장치마다 약간씩 다르게 동작
- 장치마다 정확한 사용을 위해 **데이터 시트 확인!**

## ■ Serial Peripheral Interface<sup>1,2</sup>

- 마스터 (master) 장치와 하나 이상의 슬레이브 (slave) 장치로 구성
  - ✓ 1 (마스터) : n (슬레이브) 통신이 가능
- 데이터 **송신과 수신**이 **동시에 진행**될 수 있는 전이중 (full-duplex) 통신 표준
- 특정 슬레이브 선택 (Slave Select) 선을 이용해 1:n 통신 구현 (I<sup>2</sup>C와 비교)
- 마이크로프로세서와 주변장치 (peripheral) 간 '고속, 근거리' 통신 (버스, bus<sup>4</sup>)에서 많이 활용
  - ✓ 예, 센서, SD 카드, 레지스터 등



<sup>1</sup> [https://en.wikipedia.org/wiki/Serial\\_Peripheral\\_Interface](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface)  
<sup>2</sup> <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all>  
<sup>3</sup> de facto 표준이란 대중성과 시장 장악력을 바탕으로 어느 회사나 단체의 특정 방식 (예, 제조, 통신 등)이 표준 처럼 인정되는 것, 예로서 HDMI 포트  
<sup>4</sup> 컴퓨터 구조에서 '버스'란 컴퓨터 내부에서 컴포넌트 사이에 또는 컴퓨터 사이에 데이터를 주고 받는 통신 시스템을 통칭하는 말

# SPI Serial Peripheral Interface

## ■ SPI 연결선 (wire)과 연결 방법

- MOSI (Master Out Slave In): **마스터에서 슬레이브** 장치로 **데이터 전송**
- MISO (Master In Slave Out): **슬레이브에서 마스터** 장치로 **데이터 전송**
- SCLK (Serial CLock): 마스터-슬레이브 장치 간 **동기화를 위한 클럭 전송**
- SS (Slave Select): 마스터가 데이터를 주고 받을 **슬레이브 장치를 선택**



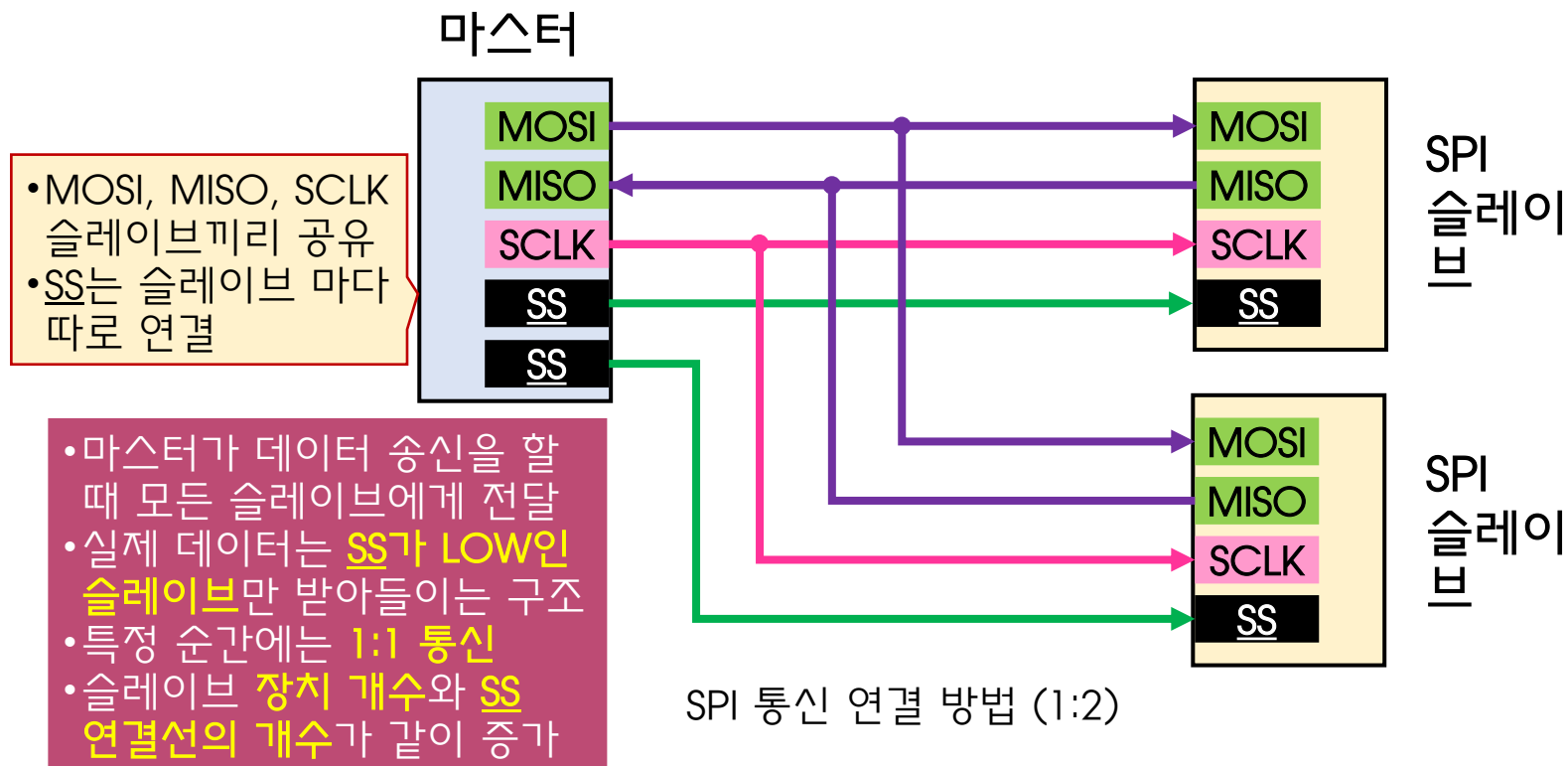
- SS 연결선의 ' ' 는 부논리 동작
- 평상시에는 (선택 받지 않았을 때는) HIGH를 유지, 특정 슬레이브를 선택할 때 LOW

SPI 통신 연결 방법 (1:1)

# SPI Serial Peripheral Interface

## ■ SPI 연결선 (wire)과 연결 방법

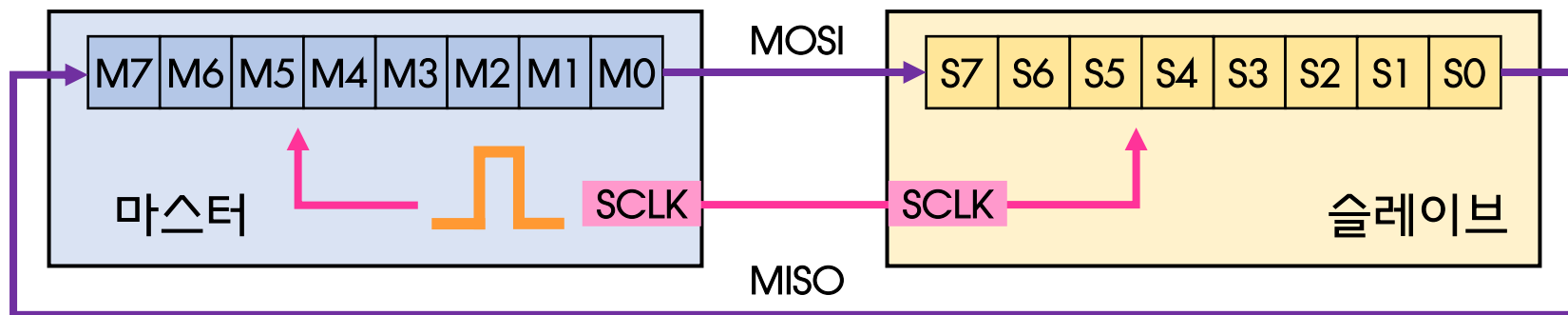
- MOSI (Master Out Slave In): **마스터에서 슬레이브 장치로 데이터 전송**
- MISO (Master In Slave Out): **슬레이브에서 마스터 장치로 데이터 전송**
- SCLK (Serial CLock): 마스터-슬레이브 장치 간 **동기화를 위한 클럭 전송**
- SS (Slave Select): 마스터가 데이터를 주고 받을 **슬레이브 장치를 선택**



# SPI 데이터 전송 방식<sup>1</sup>

## ■ 마스터-슬레이브 간 동시 송수신

- 항상 송신과 수신이 '**동시에**' 이루어짐 (UART는 송수신이 별개로 동작)
- SPI는 마스터에서 전달하는 '**시리얼 클럭**'을 기준으로 데이터를 전송
  - ✓ 마스터에서 슬레이브로 전달할 때 마스터 클럭 기준으로 진행
  - ✓ 슬레이브로 마스터에서 전달할 때 마스터 클럭 기준으로 진행
- 마스터와 슬레이브의 데이터 버퍼는 '**원형 큐 (circular queue)**'를 이룸
  - ✓ 마스터와 슬레이브가 데이터를 주고 받을 때 동시에 이루어짐



SPI 마스터-슬레이브 간 데이터 전송 구조

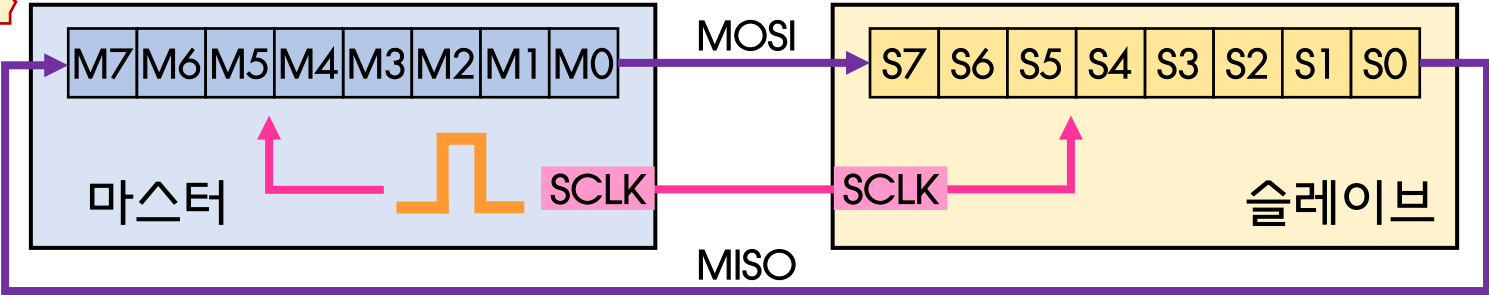
<sup>1</sup> 허경용, 사물인터넷을 품은 아두이노, p54



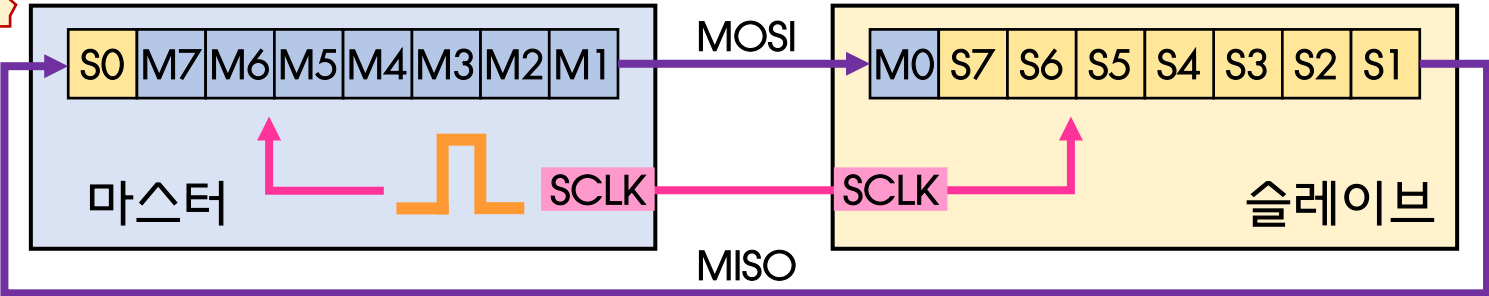
# SPI 데이터 전송 방식<sup>1</sup>

- 시리얼 클럭이 하나 발생하면 마스터 데이터 버퍼의 1 bit 데이터 M0가 슬레이브로 전달되며 동시에,
- 슬레이브 데이터 버퍼의 1 bit 데이터 S0가 마스터로 전달
- 즉 두 버퍼 간 원형 큐 데이터 이동과 동일한 결과
- 8번의 시리얼 클럭으로 1 byte 데이터 '교환' 가능

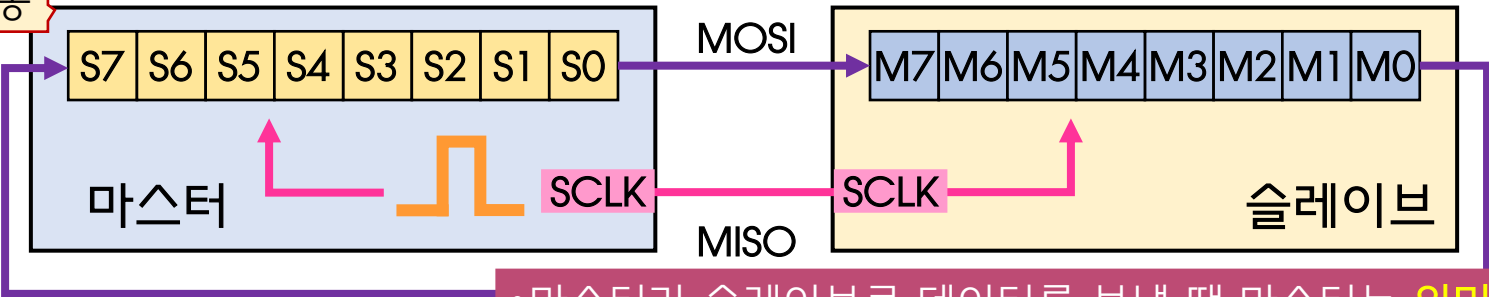
•대기상태



•1bit 전송



•1byte 전송



- 마스터가 슬레이브로 데이터를 보낼 때 마스터는 의미없는 데이터를 슬레이브로부터 받음, 반대의 경우도 동일한 결과

<sup>1</sup> 허경용, 사물인터넷을 품은 아두이노, p54

# SPI 데이터 전송 방식<sup>1</sup> 데이터 전송 다이어그램

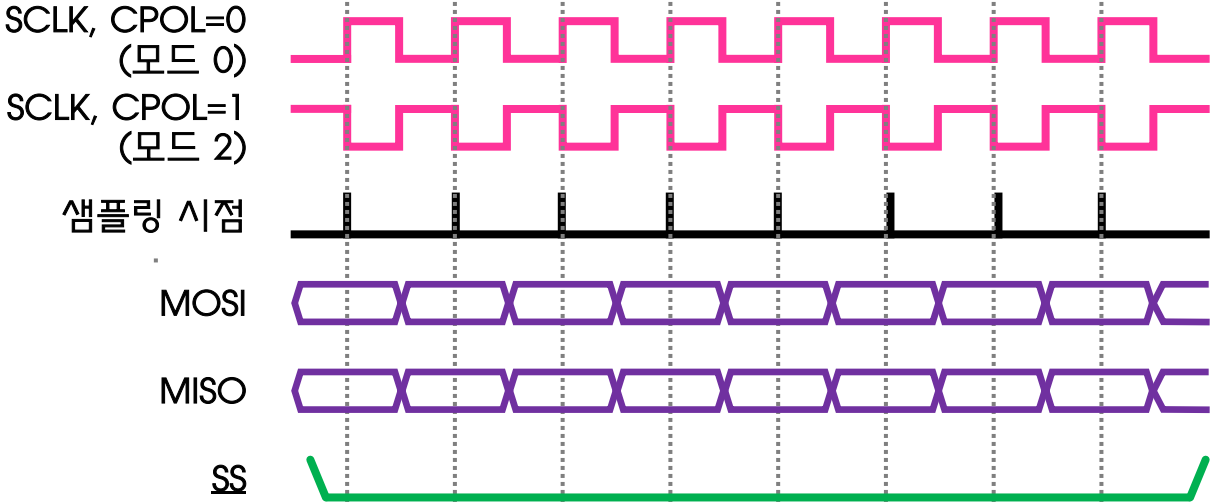
- 모드 0: CPOL=0, CPHA=0
- 모드 1: CPOL=0, CPHA=1
- 모드 2: CPOL=1, CPHA=0
- 모드 3: CPOL=1, CPHA=1

- CPOL = 0  
비활성일 때 SCLK = LOW  
활성일 때 SCLK = HIGH
- CPOL = 1  
비활성일 때 SCLK = HIGH  
활성일 때 SCLK = LOW

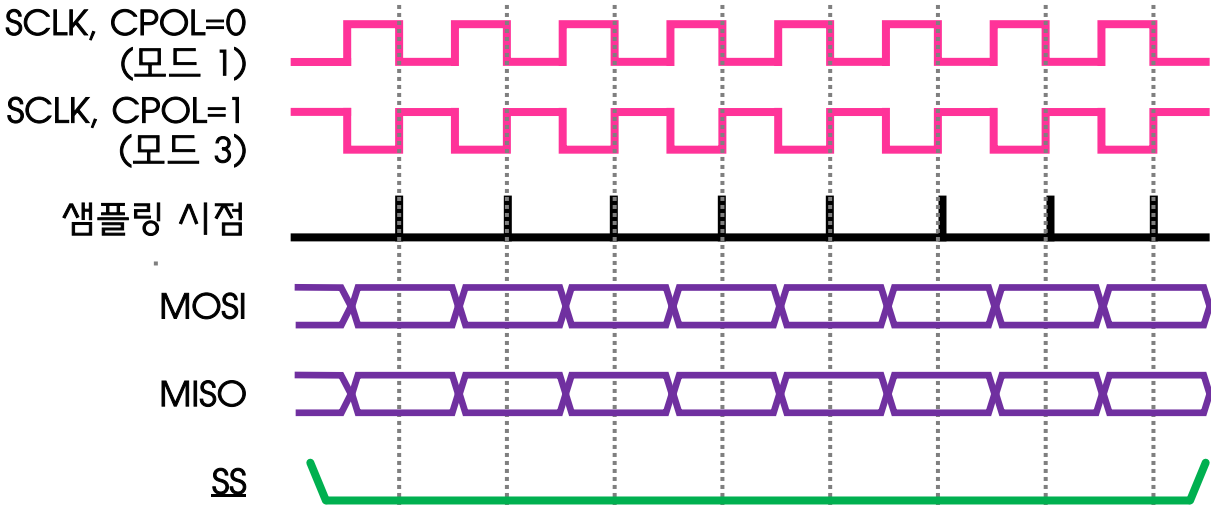
- CPHA = 0
- 비활성→활성으로 바뀌는 에지에서 샘플링

- CPOL: SPI 버스가 유힤 상태일 때 클럭 값 결정 (Clock POLarity)
- CPHA: 데이터 샘플링 시점을 결정 (Clock PHAse)

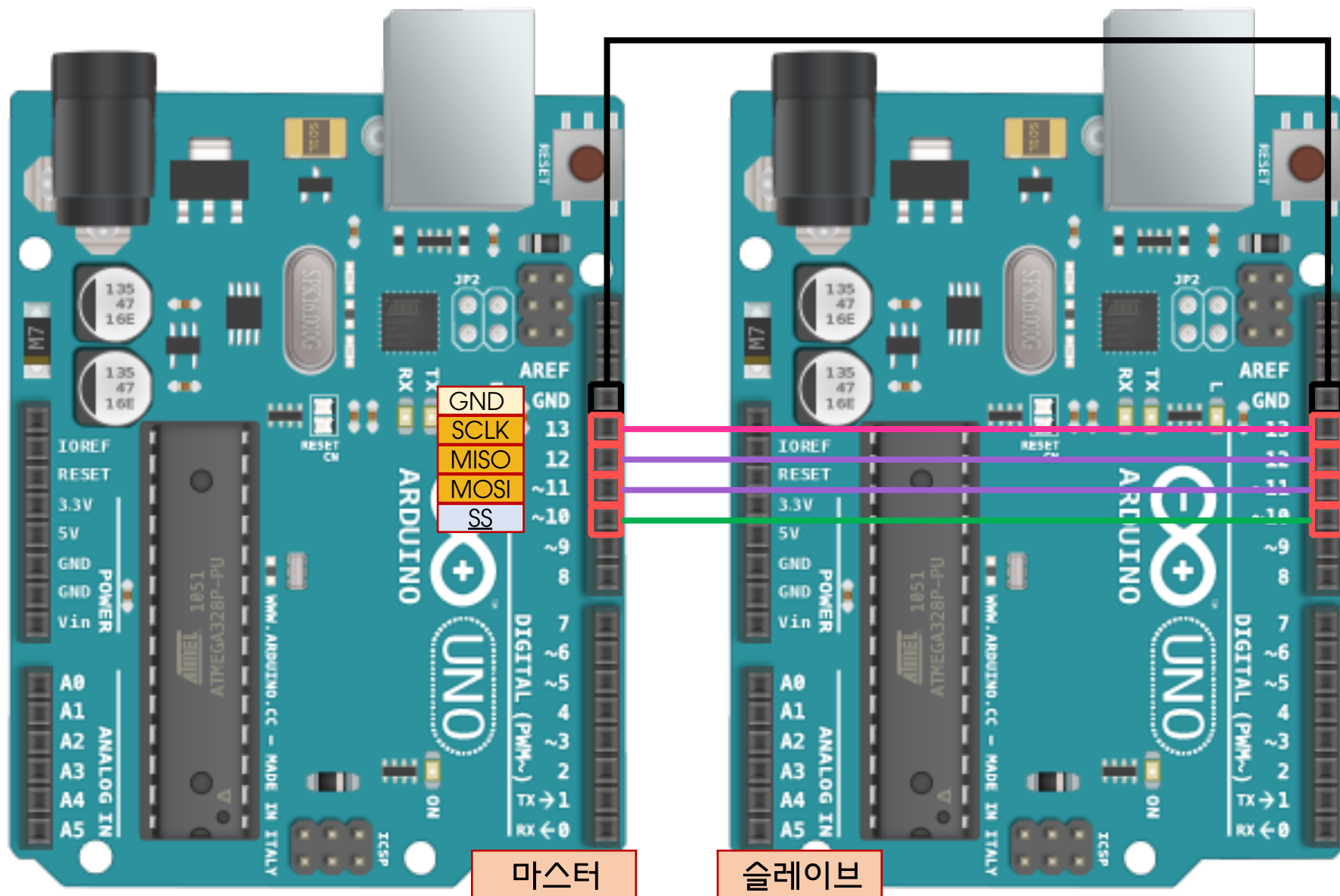
- CPHA = 1
- 활성→비활성으로 바뀌는 에지에서 샘플링



• 데이터 샘플링 동안 SS=LOW 유지



# 아두이노 우노 SPI 예약 핀 번호



마스터

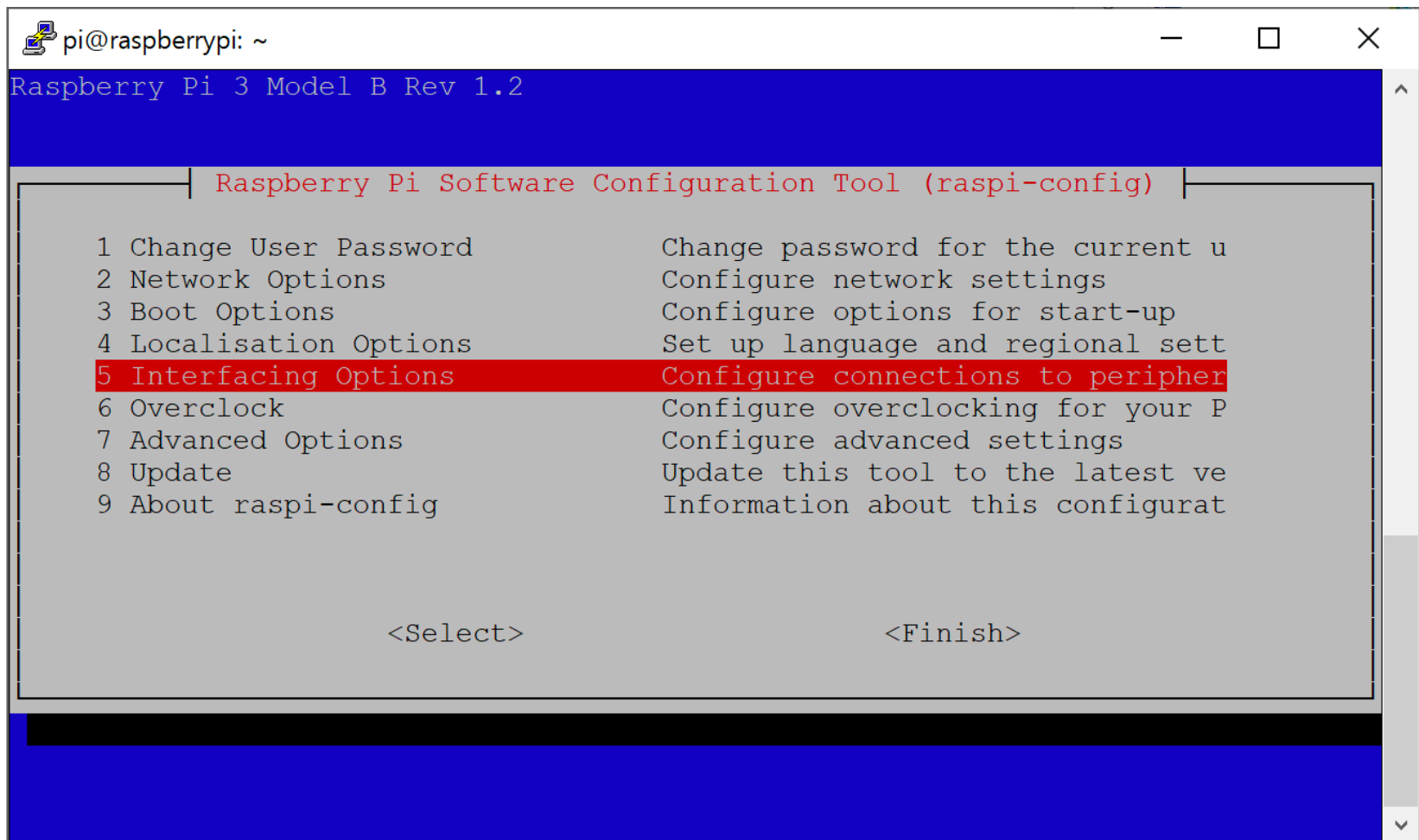
슬레이브

# RPi SPI 설정

raspi-config

# RPI SPI 활성화<sup>1</sup>

## ■ sudo raspi-config



```
pi@raspberrypi: ~
Raspberry Pi 3 Model B Rev 1.2

Raspberry Pi Software Configuration Tool (raspi-config)

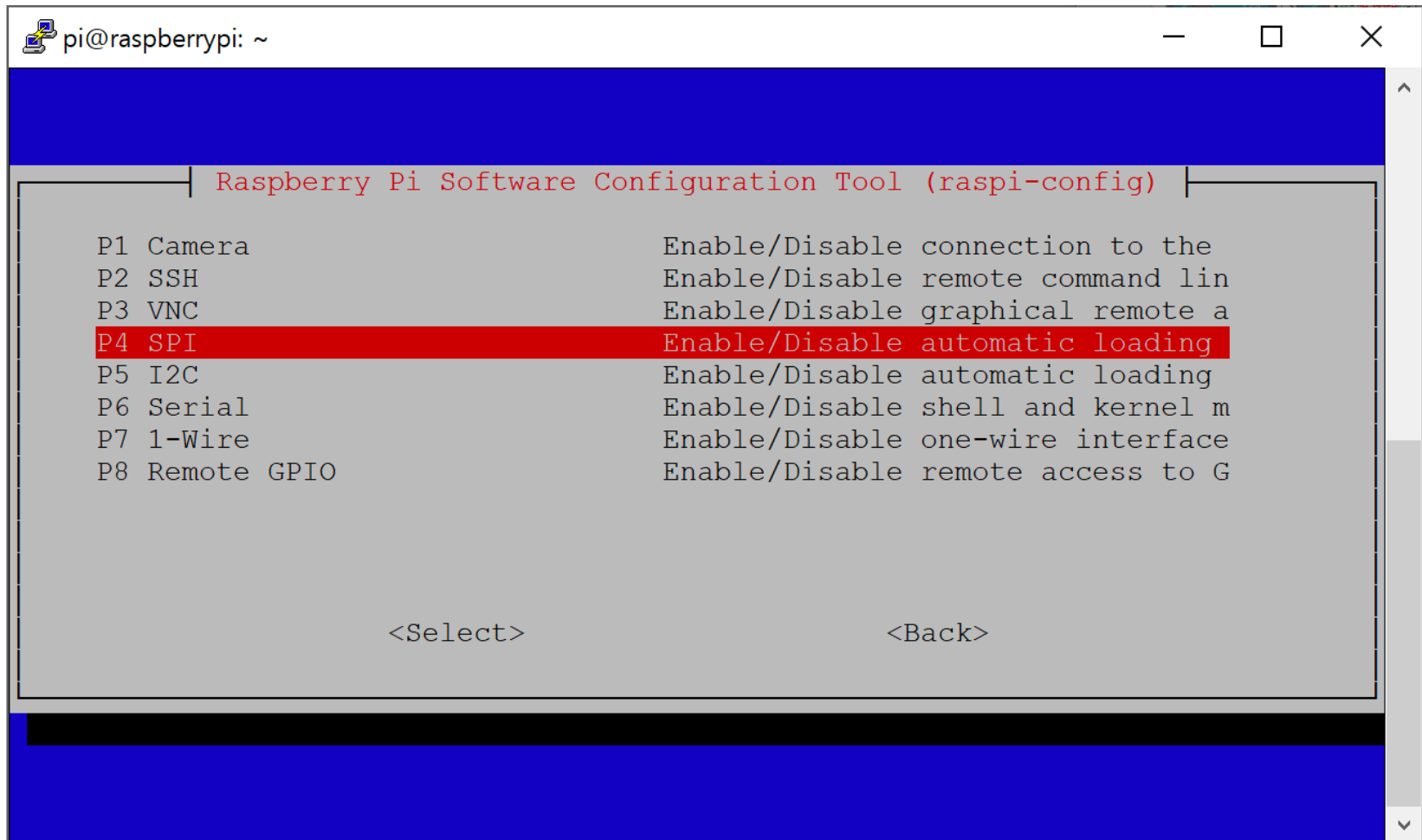
1 Change User Password      Change password for the current u
2 Network Options           Configure network settings
3 Boot Options              Configure options for start-up
4 Localisation Options      Set up language and regional sett
5 Interfacing Options        Configure connections to peripher
6 Overclock                 Configure overclocking for your P
7 Advanced Options          Configure advanced settings
8 Update                    Update this tool to the latest ve
9 About raspi-config        Information about this configurat

<Select>                    <Finish>
```

<sup>1</sup> 익스플로링 라즈베리 파이, p356

# RPI SPI 활성화<sup>1</sup>

## ■ sudo raspi-config



<sup>1</sup> 익스플로링 라즈베리 파이, p356

# RPI SPI 활성화<sup>1</sup>

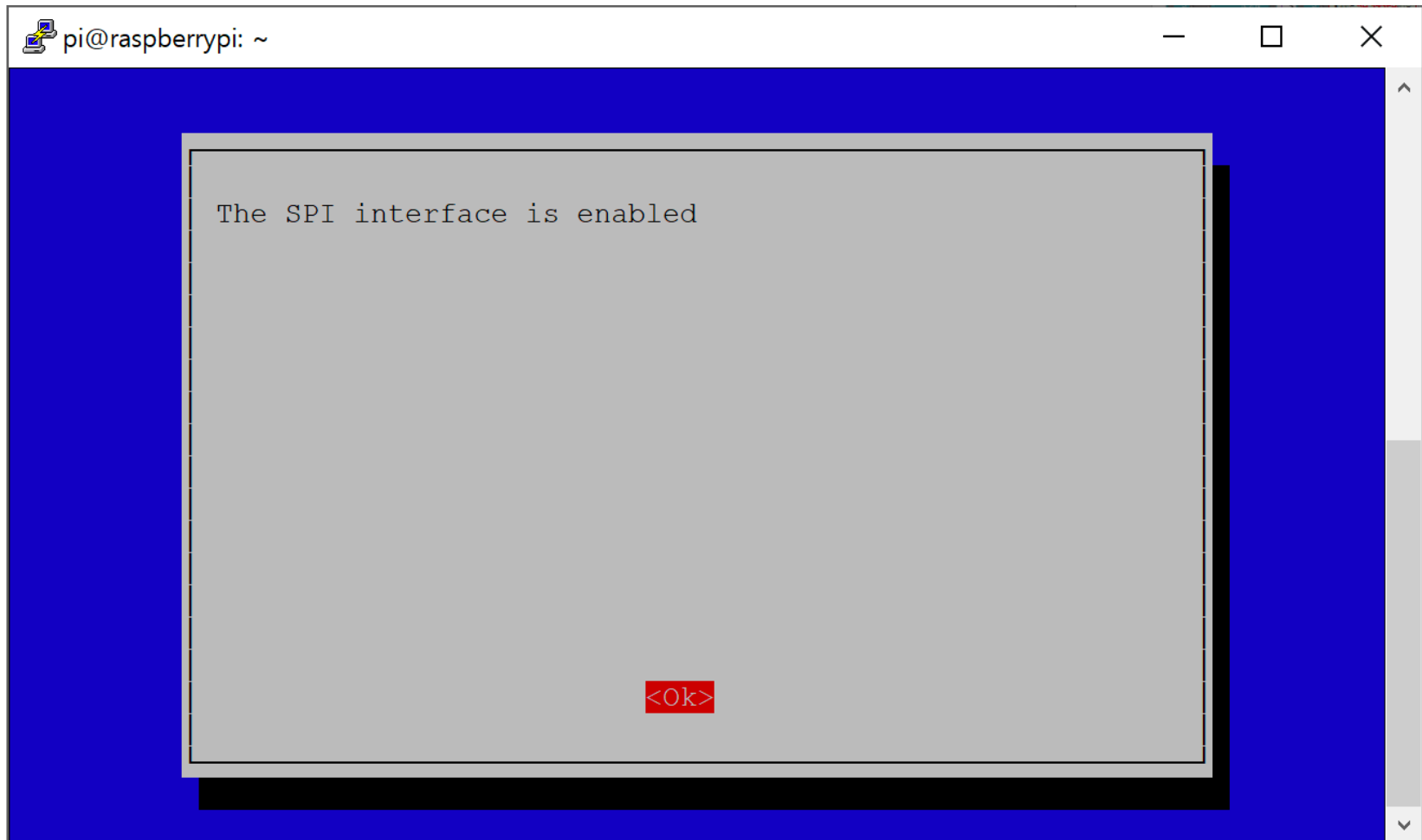
## ■ sudo raspi-config



<sup>1</sup> 익스플로링 라즈베리 파이, p356

# RPI SPI 활성화<sup>1</sup>

## ■ sudo raspi-config



<sup>1</sup> 익스플로링 라즈베리 파이, p356



# RPI SPI 활성화<sup>1</sup>

## ■ SPI 모듈과 인터페이스 사용 가능 확인

```
pi@raspberrypi: ~  
pi@raspberrypi:~ $ sudo raspi-config  
pi@raspberrypi:~ $ more /boot/config.txt | grep spi  
dtparam=spi=on  
pi@raspberrypi:~ $
```

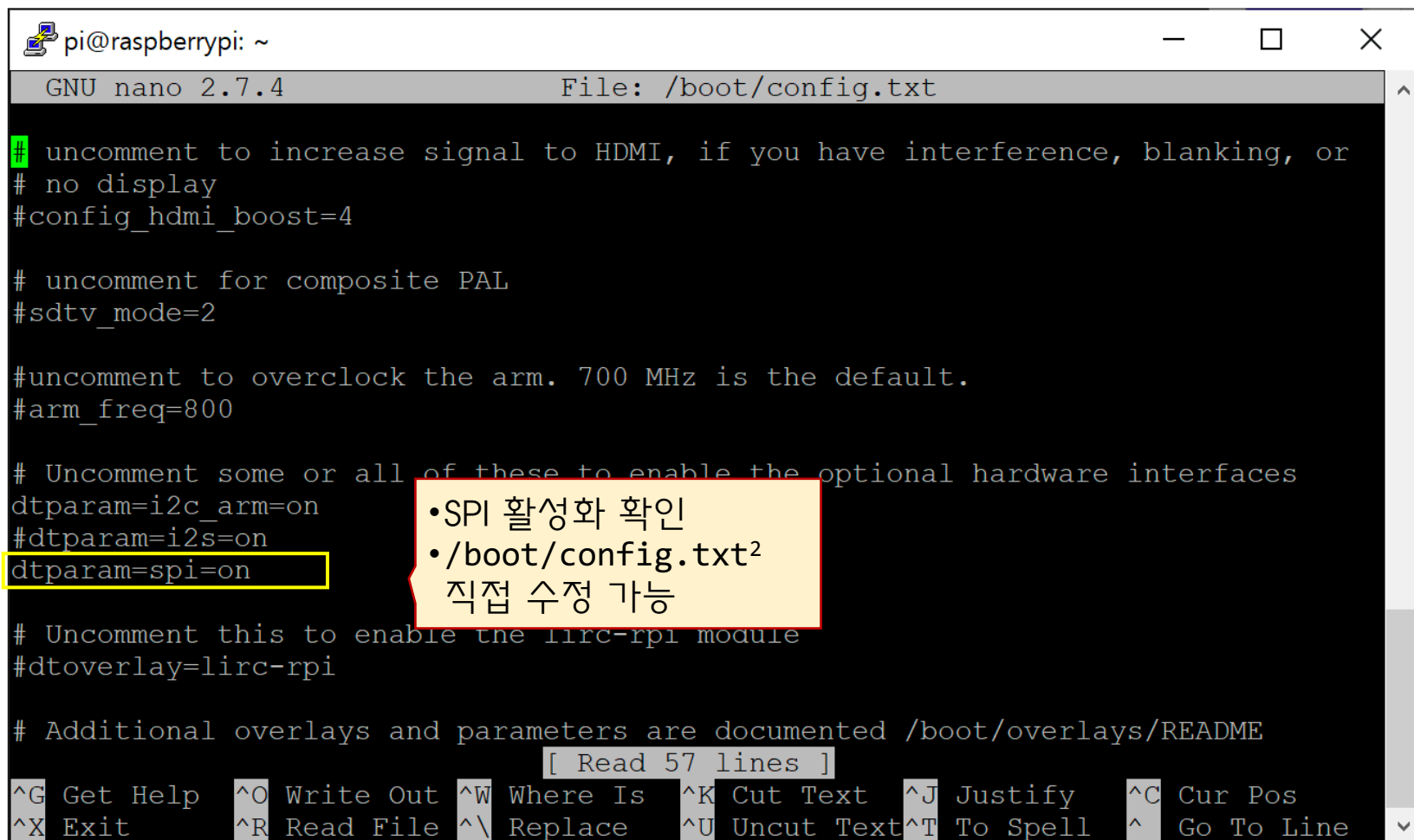
SPI 활성화 확인  
• /boot/config.txt<sup>2</sup>  
직접 수정 가능

<sup>1</sup> 익스플로링 라즈베리 파이, p356

<sup>2</sup> RPi가 부팅할 때 참고하는 설정 파일로서 일반 PC의 BIOS 역할을 대신함

# RPI SPI 활성화<sup>1</sup>

## ■ sudo nano /boot/config.txt



```
pi@raspberrypi: ~
GNU nano 2.7.4 File: /boot/config.txt

# uncomment to increase signal to HDMI, if you have interference, blanking, or
# no display
#config_hdmi_boost=4

# uncomment for composite PAL
#sdtv_mode=2

#uncomment to overclock the arm. 700 MHz is the default.
#arm_freq=800

# Uncomment some or all of these to enable the optional hardware interfaces
dtparam=i2c_arm=on
#dtparam=i2s=on
dtparam=spi=on

# Uncomment this to enable the lirc-rpi module
#dtoverlay=lirc-rpi

# Additional overlays and parameters are documented /boot/overlays/README
[ Read 57 lines ]

^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 ^_ Go To Line
```

- SPI 활성화 확인
- /boot/config.txt<sup>2</sup>
- 직접 수정 가능

<sup>1</sup> 익스플로링 라즈베리 파이, p356

<sup>2</sup> RPI가 부팅할 때 참고하는 설정 파일로서 일반 PC의 BIOS 역할을 대신함

# RPI SPI 활성화<sup>1</sup>

## ■ SPI 모듈과 인터페이스 사용 가능 확인

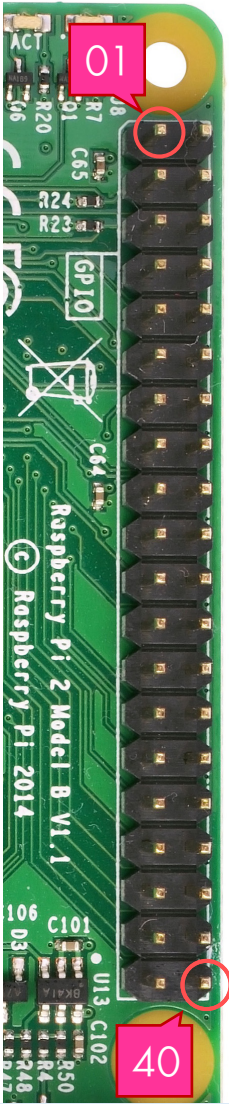
```
pi@raspberrypi: ~  
pi@raspberrypi:~ $ sudo raspi-config  
pi@raspberrypi:~ $ more /boot/config.txt | grep spi  
dtparam=spi=on  
pi@raspberrypi:~ $ sudo nano /boot/config.txt  
pi@raspberrypi:~ $ lsmod | grep spi  
spidev                16384  0  
spi_bcm2835           16384  0  
pi@raspberrypi:~ $ ls /dev/*spi*  
/dev/spidev0.0 /dev/spidev0.1  
pi@raspberrypi:~ $
```

- SPI 모듈이 커널 modules 디렉토리로 부터 로드 되어 있음을 확인
- SPI 인터페이스가 device 디렉토리에 대기 하고 있음을 확인
- 두개의 활성화 모드 (0, 1)가 있는 SPI 장치인 spidev0만 존재<sup>2</sup>
  - The SPI driver exposes two device files (/dev/spidev-0.0 and /dev/spidev-0.1), one for each cable select line (CE0 and CE1 on RPi)

<sup>1</sup> 익스플로링 라즈베리 파이, p356

<sup>2</sup> <https://www.bootc.net/projects/raspberry-pi-kernel/>

# RPI GPIO 핀 헤더<sup>1,2,3</sup>



Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I2C)	DC Power 5v	04
05	GPIO03 (SCL1 , I2C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO) SPI0	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)	(I2C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

<sup>1</sup> <http://www.raspberry-pi-geek.com/howto/GPIO-Pinout-Rasp-Pi-1-Model-B-Rasp-Pi-2-Model-B>  
<sup>2</sup> 익스플로링 라즈베리 파이, p233  
<sup>3</sup> <https://pinout.xyz/pinout/>

# Analog-Digital 변환기

MCP3008

# 아날로그 디지털 변환기 회로 구성

• RPi는 아두이노와 달리 아날로그 입력이 없음을 기억!

## ■ MCP3008<sup>1</sup>

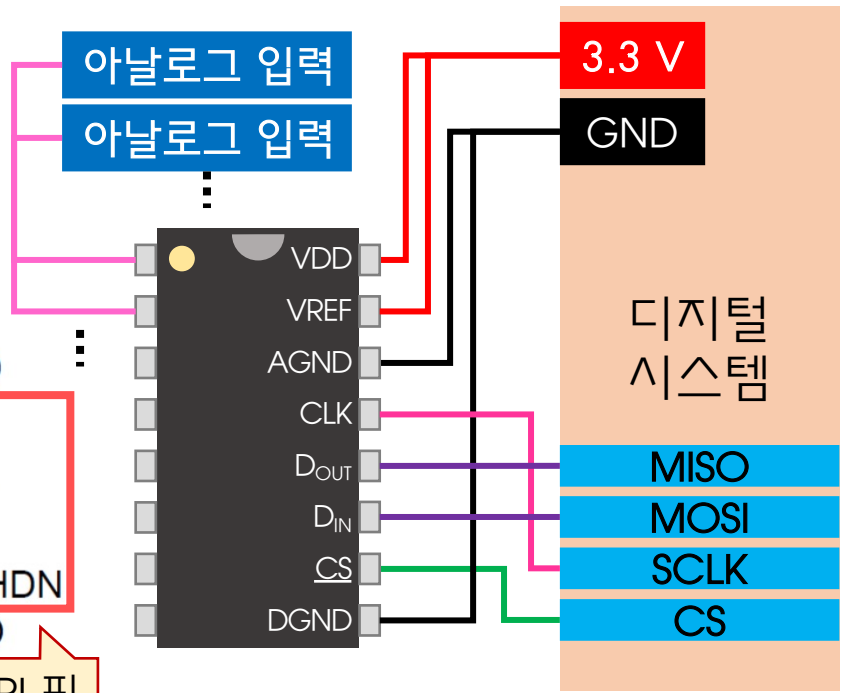
- 10-bit 아날로그 디지털 변환기 (analog-to-digital convertor)
- 디지털 인터페이스: SPI
- 입력 채널: 8 채널 (CH0-CH7), MCP3004는 4채널 지원을 제외하고 동일
- 동작 전압: 2.7-5.5 V



MCP3008<sup>2</sup>



MCP3008 핀 구성<sup>1</sup>



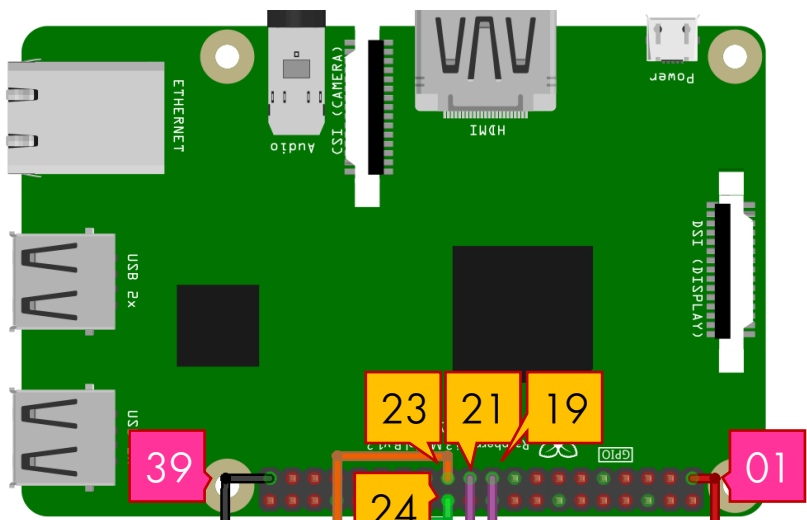
MCP3008 회로 연결<sup>3</sup>

<sup>1</sup> <https://www.microchip.com/wwwproducts/en/en010530>  
<sup>2</sup> <https://www.eleparts.co.kr/EPXHYARU>  
<sup>3</sup> <https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters/mcp3008>

# 아날로그 디지털 변환기 회로 구성

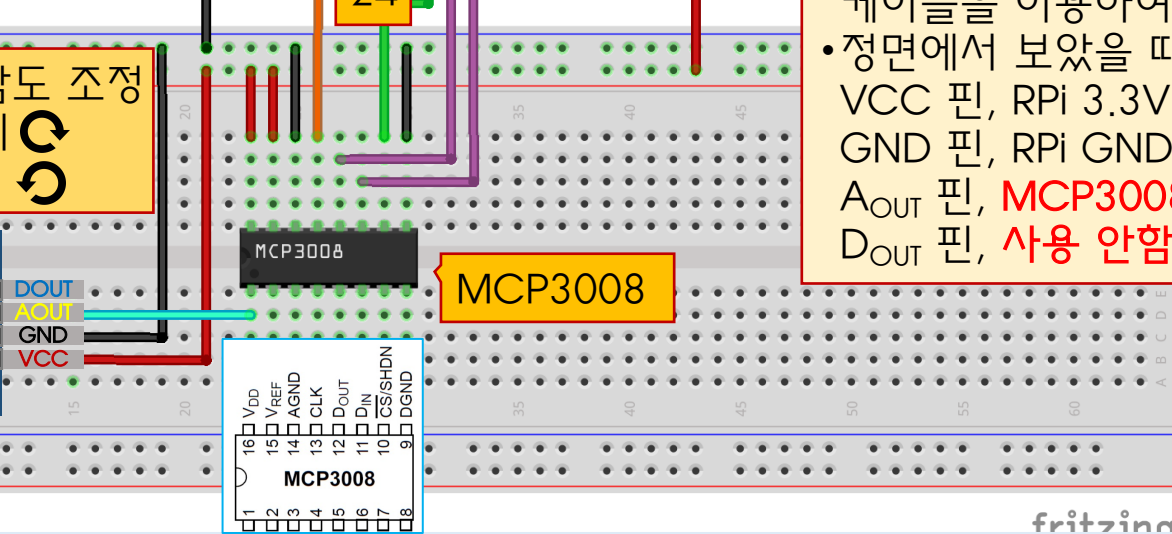
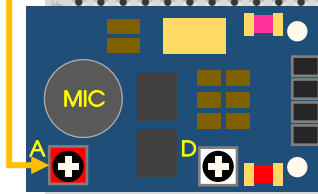
Practice

## ■ RPi, MCP3008, 소리 센서<sup>1</sup> 연결



- RPi-MCP3008을 (4핀) 점퍼 케이블을 이용하여 그림과 같이 연결
- 정면에서 보았을 때 (데이터시트 확인!), V<sub>DD</sub>, V<sub>REF</sub> 핀, RPi 3.3V 연결
- AGND, DGND 핀, RPi GND 연결
- CLK 핀, RPi P23 (GPIO11) 연결
- D<sub>OUT</sub> 핀, RPi P21 (GPIO09) 연결
- D<sub>IN</sub> 핀, RPi P19 (GPIO10) 연결
- CS 핀, RPi P24 (GPIO 08) 연결
- RPi 1번 (VCC), RPi 39번 (GND) 구성

- 아날로그 출력 감도 조정
- 작은 소리도 감지
- 큰 소리만 감지



- 소리 센서 MCP3008를 (3핀) 점퍼 케이블을 이용하여 그림과 같이 연결
- 정면에서 보았을 때 (데이터시트 확인!), VCC 핀, RPi 3.3V 연결
- GND 핀, RPi GND 연결
- A<sub>OUT</sub> 핀, MCP3008 CH0 핀 연결
- D<sub>OUT</sub> 핀, 사용 안함

<sup>1</sup> <https://www.waveshare.com/sound-sensor.htm>

## 아날로그 디지털 변환기 wPi 프로그래밍 (SPI)

## Practice

```
#include <stdio.h>
#include <wiringPi.h>
#include <mcp3004.h>
```

```
#define BASE 200
#define SPI_CHAN 0
```

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

```
    mcp3004Setup(BASE, SPI_CHAN);
```

```
    int x;
```

```
    while (1) {
        x = analogRead(BASE);
```

```
        if (x < 500)
            printf("%d\n", x);
        delay(100);
```

```
    }
```

```
    return 0;
```

```
}
```

- WiringPi 라이브러리를 이용한 SPI 통신 프로그램 개발을 위해 헤더 파일 포함 (**wiringPi.h, mcp3004.h**)
- WiringPi 라이브러리를 사용하지 않고 개발 가능  
→ RPi가 아닌 **일반 임베디드 소프트웨어 개발**에 유리

- BASE는 wPi에서 MCP3008의 채널 입력 (CH0-CH7) 핀 번호를 위해 정의한 **가상의 시작 번호**
- 이 경우 CH0은 200번, CH1은 201번, ..., CH7은 207번
- SPI\_CHAN을 0으로 설정한 것은 RPi의 SPI0 사용을 의미

- MCP3008의 CH0 핀 번호를 200으로 설정
- MCP3008과 RPi SPI0 간 SPI 버스 통신 설정

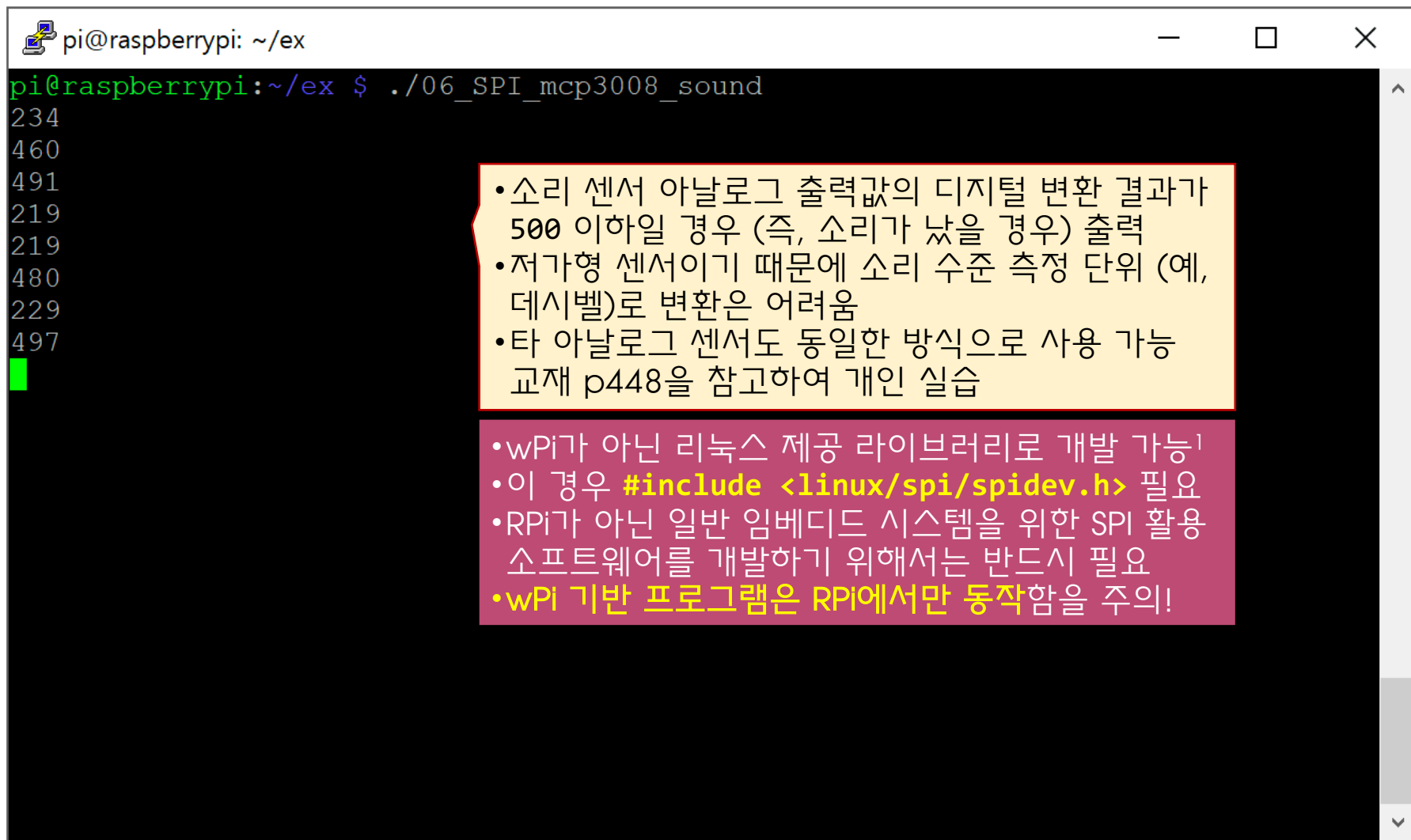
- MCP3008의 CH0 핀 번호 아날로그 값 (0-3.3V)을 디지털 값 (0-1024= $2^{10}$ )으로 변환

- WaveShare 소리 센서는 '조용하다' → '소리가 나면',
- D<sub>OUT</sub>은 HIGH → LOW 디지털 출력 후 → HIGH 복귀
- A<sub>OUT</sub>은 1.6V (약 512) → 소리 크기에 비례하여 낮은 전압 출력 후 → 1.6V 복귀
- 소리가 나서 A<sub>OUT</sub> DAC 변환 결과가 '500' 이하로 내려가면 변환값과 함께 출력



## 주변 소리 센서 프로그램 실행

Practice



```
pi@raspberrypi: ~/ex
pi@raspberrypi:~/ex $ ./06_SPI_mcp3008_sound
234
460
491
219
219
480
229
497
```

- 소리 센서 아날로그 출력값의 디지털 변환 결과가 500 이하일 경우 (즉, 소리가 낮을 경우) 출력
- 저가형 센서이기 때문에 소리 수준 측정 단위 (예, 데시벨)로 변환은 어려움
- 타 아날로그 센서도 동일한 방식으로 사용 가능  
교재 p448을 참고하여 개인 실습

- wPi가 아닌 리눅스 제공 라이브러리로 개발 가능<sup>1</sup>
- 이 경우 **#include <linux/spi/spidev.h>** 필요
- RPi가 아닌 일반 임베디드 시스템을 위한 SPI 활용 소프트웨어를 개발하기 위해서는 반드시 필요
- wPi 기반 프로그램은 RPi에서만 동작함을 주의!

<sup>1</sup> <http://www.hertaville.com/interfacing-an-spi-adc-mcp3008-chip-to-the-raspberry-pi-using-c.html>

# Summary

- I<sup>2</sup>C bus in RPi
- Accelerometer ADXL345
- SPI bus in RPi
- ADC MCP3008

# Thank you

Questions?

Contact: [eclass.sch.ac.kr](http://eclass.sch.ac.kr)  
(순천향대학교 학습플랫폼 LMS)