

IoT Platform 7th Week

- Interfacing to RPi UART -

Jaeseok Yun

Soonchunhyang University

UART 통신 (리뷰)

임베디드 시스템 (아두이노)

Serial vs. Parallel 직렬 (시리얼) vs. 병렬 통신

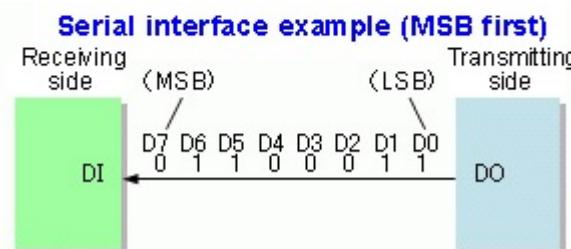
■ 장치 간 통신 방법: 시리얼 vs. 패럴렐^{1,2}

시리얼 통신

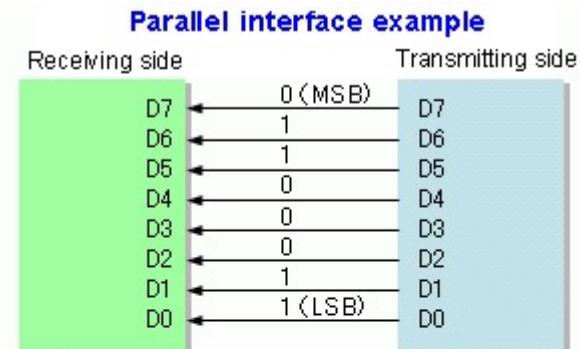
- 통신 채널이나 컴퓨터 버스를 통해 데이터를 '**한 번에 한 비트씩**' '**순서대로** (sequentially)' 보내는 방식
- 시리얼 통신은 직렬화/역직렬화 (serialize / deserialize)³ 작업 추가가 필요하지만 병렬 방식에 비해 **원거리 통신에 유리**
- 현대 컴퓨터에 와서는 신호 집적도와 통신 속도 발전으로 근거리 통신에서도 많이 활용

패럴렐 통신

- 통신 채널이나 컴퓨터 버스를 통해 데이터를 '**한 번에 여러 비트씩**' '**동시에** (simultaneously)' 보내는 방식
- 구조가 간단하고 **한 번에 많은 데이터**를 보낼 수 있으나 통신 거리가 길어질 수록 케이블 가격과 동기화 문제가 발생
- 집적 회로, 페리페럴 버스, 메모리 기기에서 많이 활용



직렬 통신과 병렬 통신¹



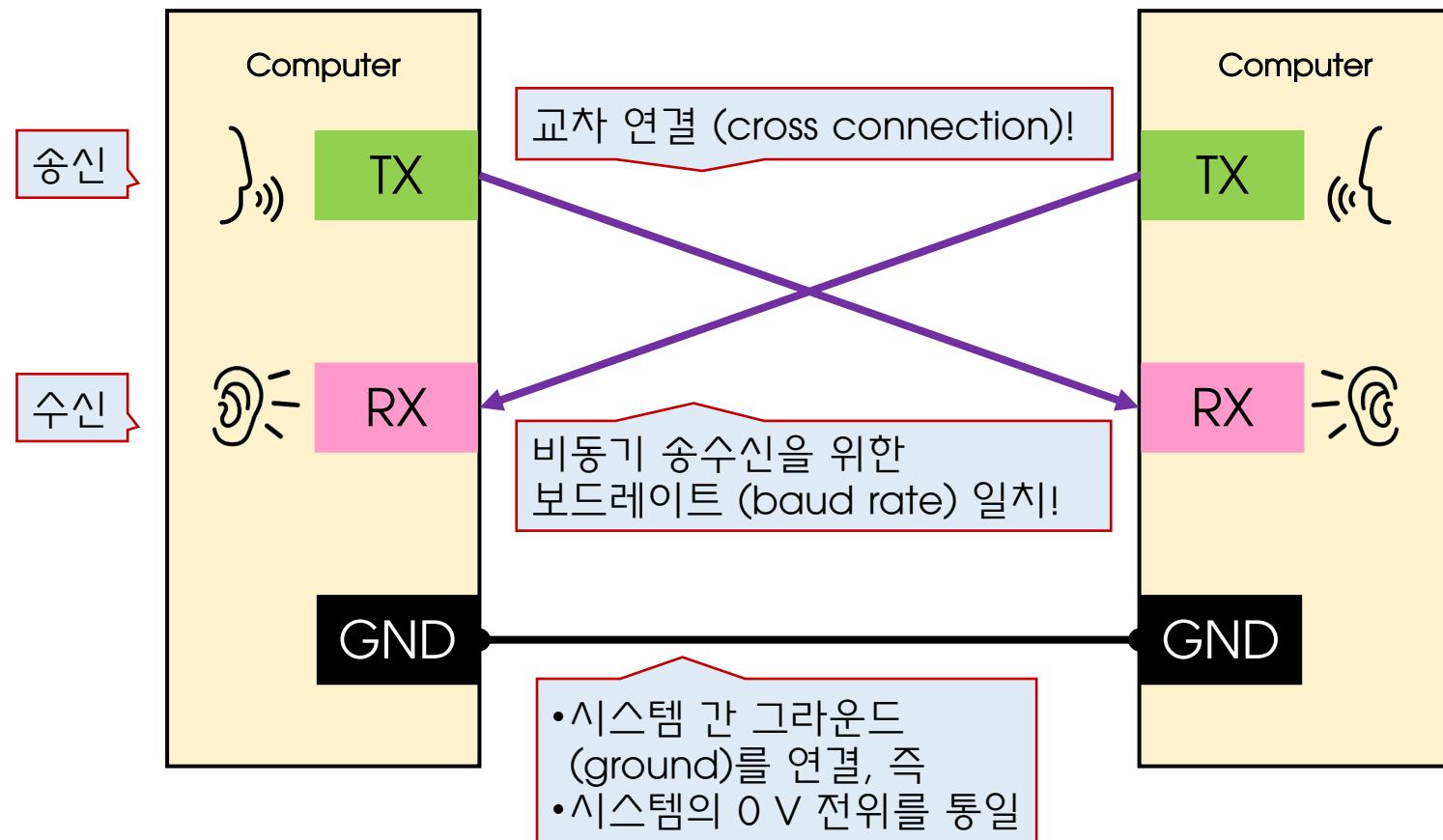
¹ https://en.wikipedia.org/wiki/Serial_communication

² <https://learn.sparkfun.com/tutorials/serial-communication/all>

³ 컴퓨터에서 데이터 처리 방식은 대부분 병렬이므로 직렬 통신을 위해서는 데이터를 한 비트씩 나열하는 직렬화가 필요하고 반대 쪽에서는 역직렬화 작업이 필요

UART

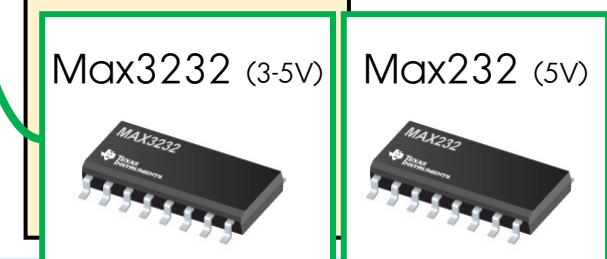
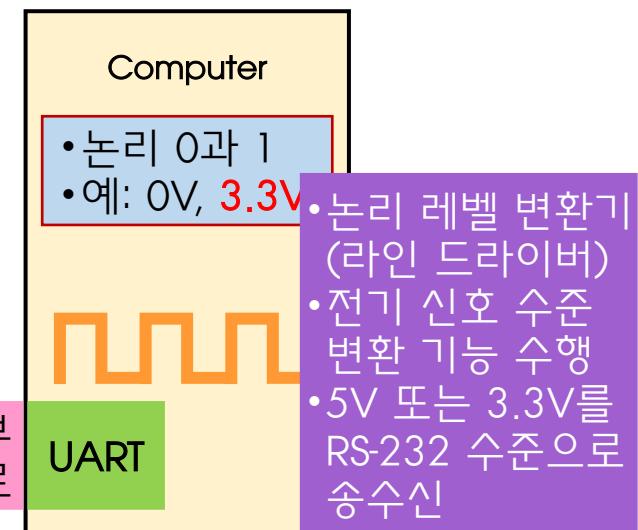
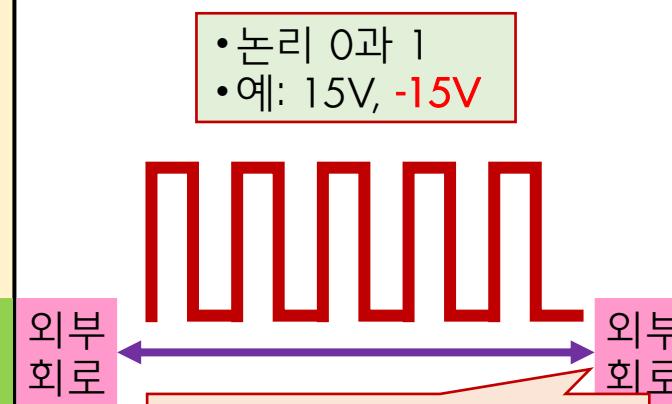
■ UART (universal asynchronous receiver-transmitter)



Universal ART: 범용

■ 범용シリ얼 통신이 필요한 이유?

- 기기 간 데이터 전송 과정에서 거리가 멀어질 수록 손실이 일어나서 잡음과 간섭에 의해 오류 증가로 **높은 전위를 갖는 전기 신호**를 사용할 필요가 있음
- 기기 간 데이터 전송과정에서 논리 0과 1을 나타내는 **전기 신호 수준**이 다를 수 있음 (예, $5V \rightarrow 12V \rightarrow 3.3V$)



- 마이크로 컨트롤러에서 통신을 위한 하드웨어 회로
- 송신할 병렬 데이터를 직렬로 바꿔주고 (**직렬화**), 수신된 직렬 데이터를 병렬로 바꿈 (**역직렬화**) 수행

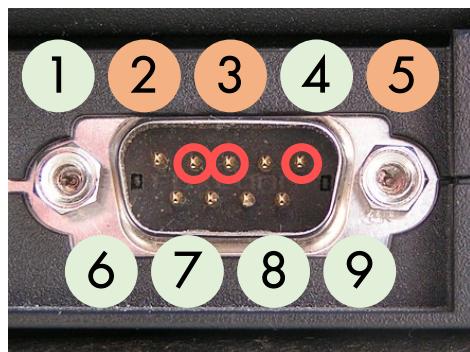
¹ transistor-transistor logic을 의미하며 저렴한 가격을 장점으로 IC 초창기 활발하게 사용되었으며 요즈음 대부분 CMOS로 대체되었다.

Universal ART: 범용

- 현대 컴퓨터에서는 시리얼 포트는 USB로 대체
- RS-232 통신을 하기 위해서는 추가 모듈³ 설치 필요

■ RS-232 통신과 컴퓨터

- (과거) 컴퓨터는 시리얼 통신으로 RS-232 제공
- 아두이노는 시리얼 통신으로 UART 제공

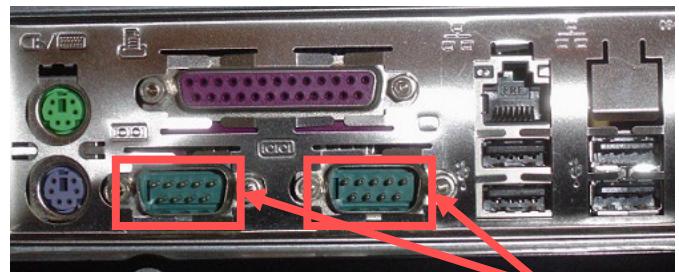


D-Sub 타입 직렬
포트 (DB-9)¹

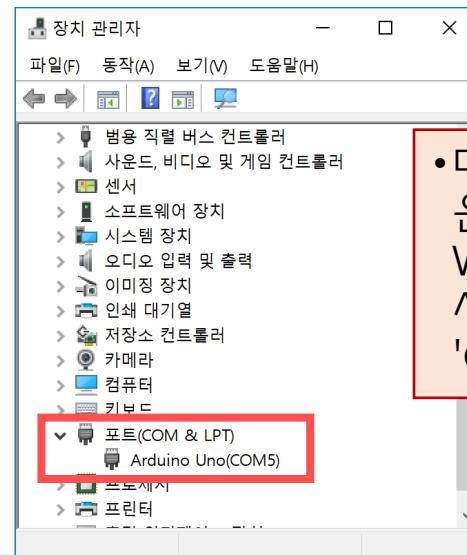
• 마이크로
컨트롤러의 UART
인터페이스와
통신하기 위해 2,
3, 5번 핀만 사용

핀번호	줄임말	전체말
1	DCD	Data Carrier Detect
2	RXD	Receive Data
3	TXD	Transmit Data
4	DTR	Data Terminal Ready
5	GND	Ground
6	DSR	Data Set Ready
7	RTS	Ready To Send
8	CTS	Clear To Send
9	RI	Ring Indicator

예전에 사용된 컴퓨터 인터페이스²



시리얼 포트
(COM1, COM2)



• 마이크로소프트 계열
운영체제 (DOS,
Windows)에서는
시리얼 포트를
'COM_no'라고 명명

¹ https://en.wikipedia.org/wiki/Serial_port

² <https://www.gizbot.com/how-to/how-to-transfer-files-using-com1-port.html>

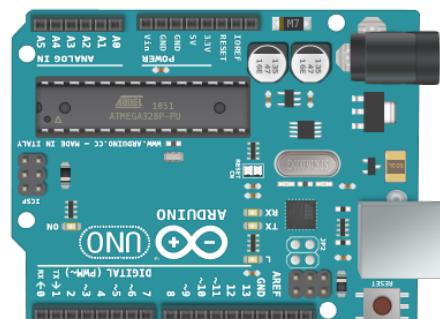
³ <https://www.amazon.com/RS-232-Serial-Adapter-Windows-64bit/dp/B004WL7MIQ>

Universal Asynchronous RT: 비동기

■ 비동기 시리얼 통신이 필요한 이유?

- 두 기기 간 데이터 통신을 할 때 동작 속도가 다름 (예, 16 MHz vs. 4.2 GHz 클럭)
- 시리얼 통신을 위한 송수신 '**속도 약속**' 필요!
- 데이터 전송 속도를 두 기기가 알아야 하며, '**보드레이트**' 설정

아두이노 Uno



16 MHz ATmega328

컴퓨터



4.2 GHz Intel Core i7

- **보드레이트**는 일반적으로 **bps** (bit per second, 초당 송수신 하는 비트 수)로 나타냄
- 예로서, 9600 bps는 초당 9600 비트를 송수신
- UART 통신에서 대표적으로 사용되는 보드레이트는 **9600, 19200, 38400, 57600, 115200** 등

9600 보드레이트
19200 보드레이트
38400 보드레이트
57600 보드레이트
74880 보드레이트
115200 보드레이트
230400 보드레이트
250000 보드레이트

아두이노 시리얼 모니터
보드레이트 설정 창

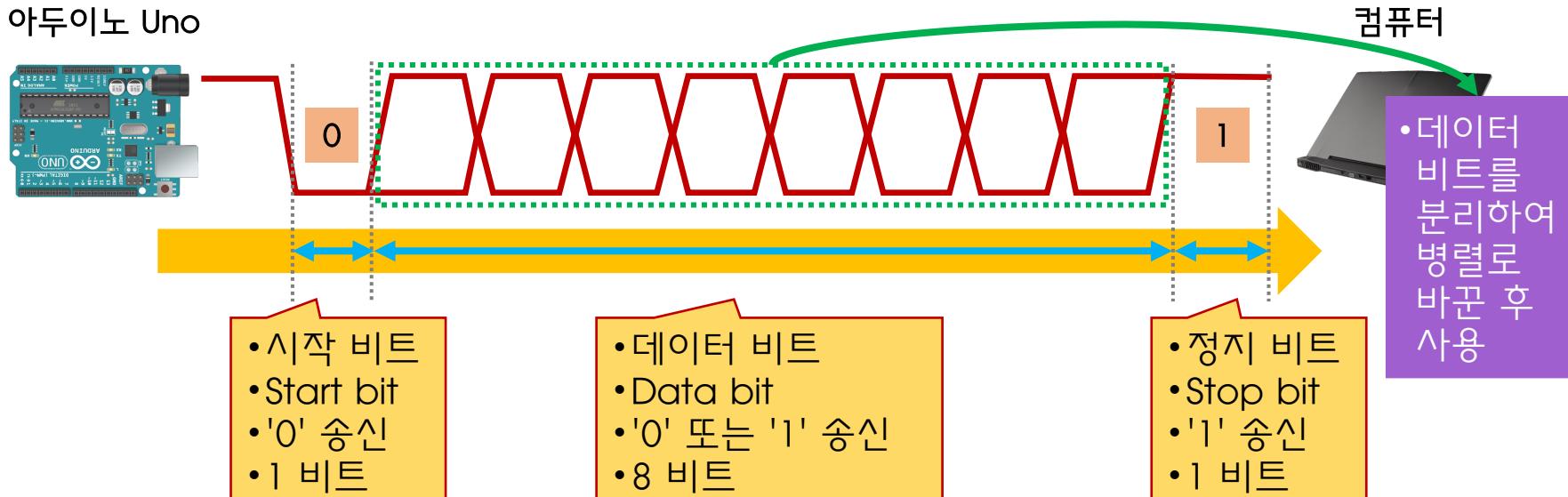
Universal Asynchronous RT: 비동기

- 클럭을 사용하는 USART¹ 방식도 있음
- 클럭이 있기 때문에 시작/정지 비트 없이 빠른 속도로 송수신이 가능하나 아두이노는 없음

■ 비동기 통신 방법

- 두 기기 간 데이터 통신을 할 때 동작 속도가 다름 (예, 16 MHz vs. 4.2 GHz 클럭)
- シリ얼 통신을 위한 송수신 '속도 약속' 필요!
- 데이터 전송 속도를 두 기기가 알아야 하며, '속도 약속'을 위해 '보드레이트' 설정

아두이노 Uno



- 시작 비트
- Start bit
- '0' 송신
- 1 비트

- 데이터 비트
- Data bit
- '0' 또는 '1' 송신
- 8 비트

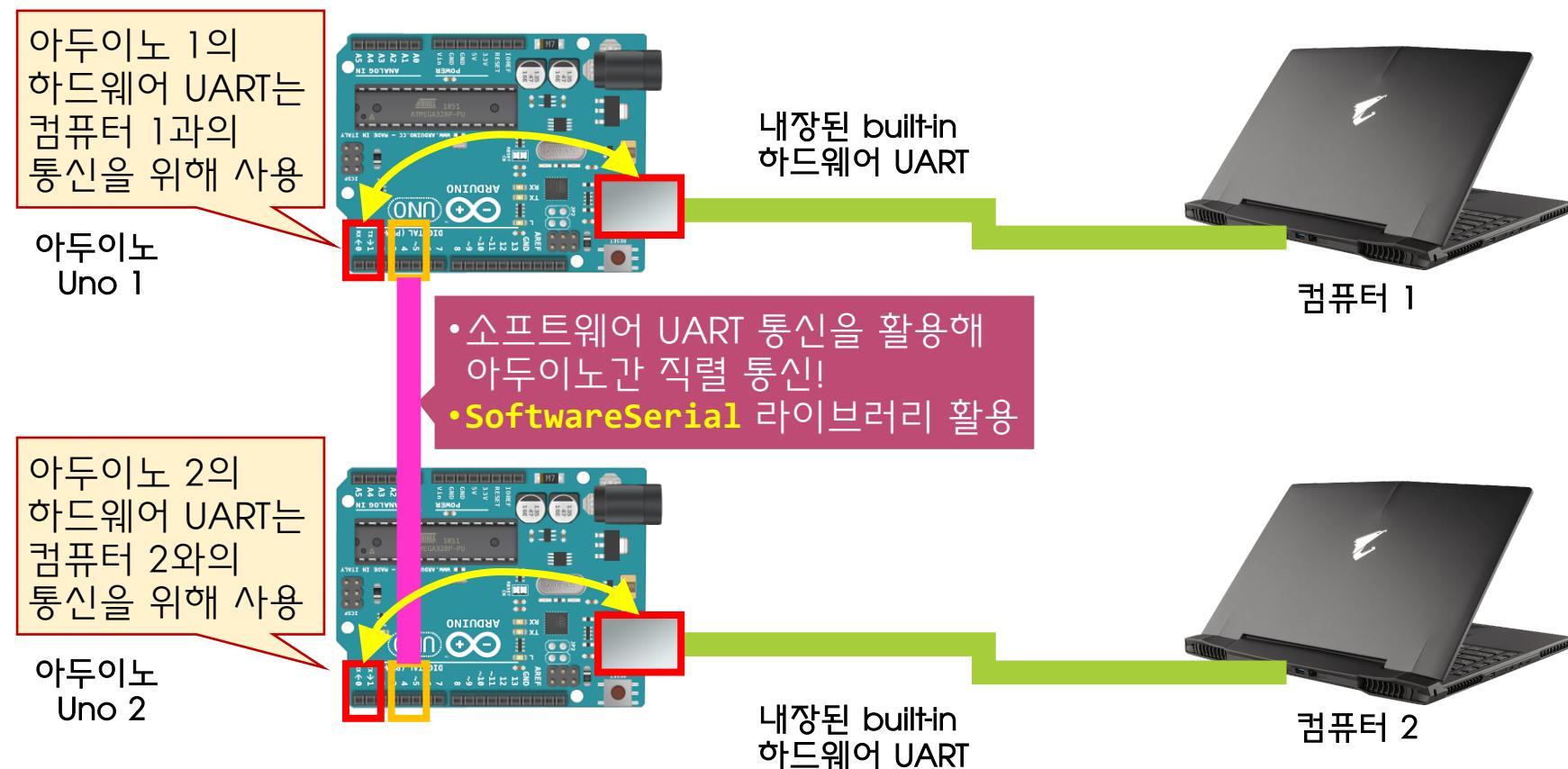
- 정지 비트
- Stop bit
- '1' 송신
- 1 비트

- UART는 비동기식 통신으로 동기화를 위한 클럭을 사용하지 않음
- 데이터가 전송되지 않을 때는 항상 '1'의 상태
- 데이터의 시작과 끝을 알기 위해 시작 비트 (start bit, 0)과 정지 비트 (stop bit, 1)를 사용

소프트웨어 UART 통신

■ 소프트웨어적으로 UART 통신 지원

- 내장된 UART 통신 (핀 0과 1)을 사용하고 있을 때 (예, USB-컴퓨터 연결)
- 아두이노의 다른 디지털 핀을 활용하여 UART 통신 기능을 모사 (replicate the functionality using software)



RPi UART 이해

RPi GPIO 핀 헤더^{1,2,3}



Pin#	NAME
01	3.3v DC Power
03	GPIO02 (SDA1 , I2C)
05	GPIO03 (SCL1 , I2C)
07	GPIO04 (GPIO_GCLK)
09	Ground
11	GPIO17 (GPIO_GEN0)
13	GPIO27 (GPIO_GEN2)
15	GPIO22 (GPIO_GEN3)
17	3.3v DC Power
19	GPIO10 (SPI_MOSI)
21	GPIO09 (SPI_MISO)
23	GPIO11 (SPI_CLK)
25	Ground
27	ID_SD (I2C ID EEPROM)
29	GPIO05
31	GPIO06
33	GPIO13
35	GPIO19
37	GPIO26
39	Ground

풀업

풀다운

풀다운

풀업

풀다운

NAME	Pin#
(TXD0) GPIO14	08
(RXD0) GPIO15	10
(GPIO_GEN1) GPIO18	12
Ground	14
(GPIO_GEN4) GPIO23	16
(GPIO_GEN5) GPIO24	18
Ground	20
(GPIO_GEN6) GPIO25	22
(SPI_CE0_N) GPIO08	24
(SPI_CE1_N) GPIO07	26
(I2C ID EEPROM) ID_SC	28
Ground	30
GPIO12	32
Ground	34
GPIO16	36
GPIO20	38
GPIO21	40

풀다운

풀다운

풀다운

풀업

풀다운

풀다운

- RPi는 GPIO14, 15에서 시리얼 포트를 제공

¹ <http://www.raspberry-pi-geek.com/howto/GPIO-Pinout-Rasp-Pi-1-Model-B-Rasp-Pi-2-Model-B>

² 익스플로링 라즈베리 파이, p233

³ <https://pinout.xyz/pinout/>

RPi 시리얼 포트 리눅스와 시리얼 포트^{1,2,3,4}

- RPi는 GPIO14, 15에서 시리얼 포트를 제공
- 리눅스 파일 시스템에서는 **/dev/ttyAMA0**
- 리눅스에서 시리얼 포트의 사용 용도
 - '네트워크 연결 (인터넷 등)'이 없을 때 컴퓨터에 연결하여 터미널 에뮬레이터를 실행하고 로그인 해 컴퓨터를 제어하는 수단으로 사용 → **'콘솔 (console)'** **로그인**이라고 부름
 - RPi에서는 시리얼 포트를 **'콘솔'** **로그인으로 사용**하도록 **기본 설정**되어 있으며 'getty' 소프트웨어 서비스를 이용해 사용
 - 그러나 우리가 원하는 시리얼 포트의 용도는 '**다른 기기** (예, 먼지 센서, 아두이노 등)**로부터 데이터를 수신** (또는 송신)하는 것'이 목적
 - 따라서, 우리가 추가로 해야 하는 것은,
 - ✓ RPi에 기본으로 설정되어 있는 **'콘솔'** **로그인을 비활성화 (disable)**로 만들고
 - ✓ **시리얼 포트에 대한 제어권**을 가져야 함

¹ <https://spellfoundry.com/2016/05/29/configuring-gpio-serial-port-raspbian-jessie-including-pi-3/>

² <https://www.raspberrypi.org/documentation/configuration/uart.md>

³ <https://wikidocs.net/7974>

⁴ 익스플로링 라즈베리 파이, p374

RPi 시리얼 포트 RPi 3과 이전 RPi 차이^{1,2,3,4}

■ RPi 3 이후부터 블루투스 기능이 포함

- **블루투스** 송수신 기능을 **/dev/ttyAMA0**에 연결해 놓은 상태
- 블루투스가 사용되면 **/dev/ttyAMA0**을 (시리얼 통신으로) 사용할 수 없음
- **/dev/ttyAMA0**은 **하드웨어 시리얼 포트 (즉, UART)**이며 **높은 성능**을 가짐
- 다른 시리얼 포트 (**/dev/ttys0**)는 부분적으로 **소프트웨어 시리얼 포트** (즉, **mini UART**)이며 **성능이 상대적으로 떨어짐**
 - ✓ CPU 코어 주파수에 의존하며 CPU가 높은 연산을 할 때 시리얼 포트에 안 좋은 영향

■ 정리하면, RPi 3에서 시리얼 포트는

- **/dev/ttyAMA0** → '블루투스'로 연결
- **/dev/ttys0** → 'GPIO 시리얼 포트 (GPIO14, 15)'로 연결

■ RPi 3 이전 모델과 호환성 문제 발생!

¹ <https://spellfoundry.com/2016/05/29/configuring-gpio-serial-port-raspbian-jessie-including-pi-3/>

² <https://www.raspberrypi.org/documentation/configuration/uart.md>

³ <https://wikidocs.net/7974>

⁴ 익스플로링 라즈베리 파이, p374

RPi 시리얼 포트 RPi 3과 이전 RPi 차이^{1,2,3,4}

■ 정리하면, RPi 3에서 시리얼 포트는

- `/dev/ttyAMA0` → '블루투스'로 연결
- `/dev/ttys0` → 'GPIO 시리얼 포트 (GPIO14, 15)'로 연결

■ RPi 3 이전 모델과 호환성 문제 발생

- RPi 3은 GPIO 시리얼 포트가 `/dev/ttys0` 이지만,
- 이전 RPi 들은 `/dev/ttyAMA0` 이기 때문에 코드 실행 문제 발생
- 이전 코드들을 RPi 3에서 사용하려면 코드 내 모든 **ttyAMA0을 ttys0로 대체**가 필요 → 하지만 이전 RPi 2와는 호환성이 없어짐

■ 문제 해결을 위해 '시리얼 포트 alias (별칭)' 등장

- RPi 3에서는 **serial0**과 **serial1** 별칭 사용
- RPi 커널이 탑재 RPi 모델에 따라 자동으로 별칭을 실제 시리얼 포트에 연결
 - ✓ RPi 3 **serial0**은 GPIO14, 15를 가리키고, **/dev/ttys0**로 알려진 mini UART 링크
 - **serial1**은 '블루투스'가 사용하는 **/dev/ttyAMA0**로 알려진 하드웨어 UART 링크
- ✓ 이전 RPi 모델은 **serial0**이 `/dev/ttyAMA0`로 알려진 하드웨어 UART를 가리킴

¹ <https://spellfoundry.com/2016/05/29/configuring-gpio-serial-port-raspbian-jessie-including-pi-3/>

² <https://www.raspberrypi.org/documentation/configuration/uart.md>

³ <https://wikidocs.net/7974>

⁴ 익스플로링 라즈베리 파일, p374

RPi UART 활성화

■ /dev/ttyAMA0과 alias serial1 확인

```
pi@raspberrypi: ~
pi@raspberrypi: ~ $ ls /dev/tty*
/dev/tty      /dev/tty19   /dev/tty3     /dev/tty40   /dev/tt
/dev/tty0     /dev/tty2    /dev/tty30    /dev/tty41   /dev/tt
/dev/tty1     /dev/tty20   /dev/tty31    /dev/tty42   /dev/tt
/dev/tty10    /dev/tty21   /dev/tty32    /dev/tty43   /dev/tt
/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: ~ $ ls -l /dev/serial*
lrwxrwxrwx 1 root root 7 Mar 11 04:01 /dev/serial1 -> ttyAMA0
pi@raspberrypi: ~ $
```

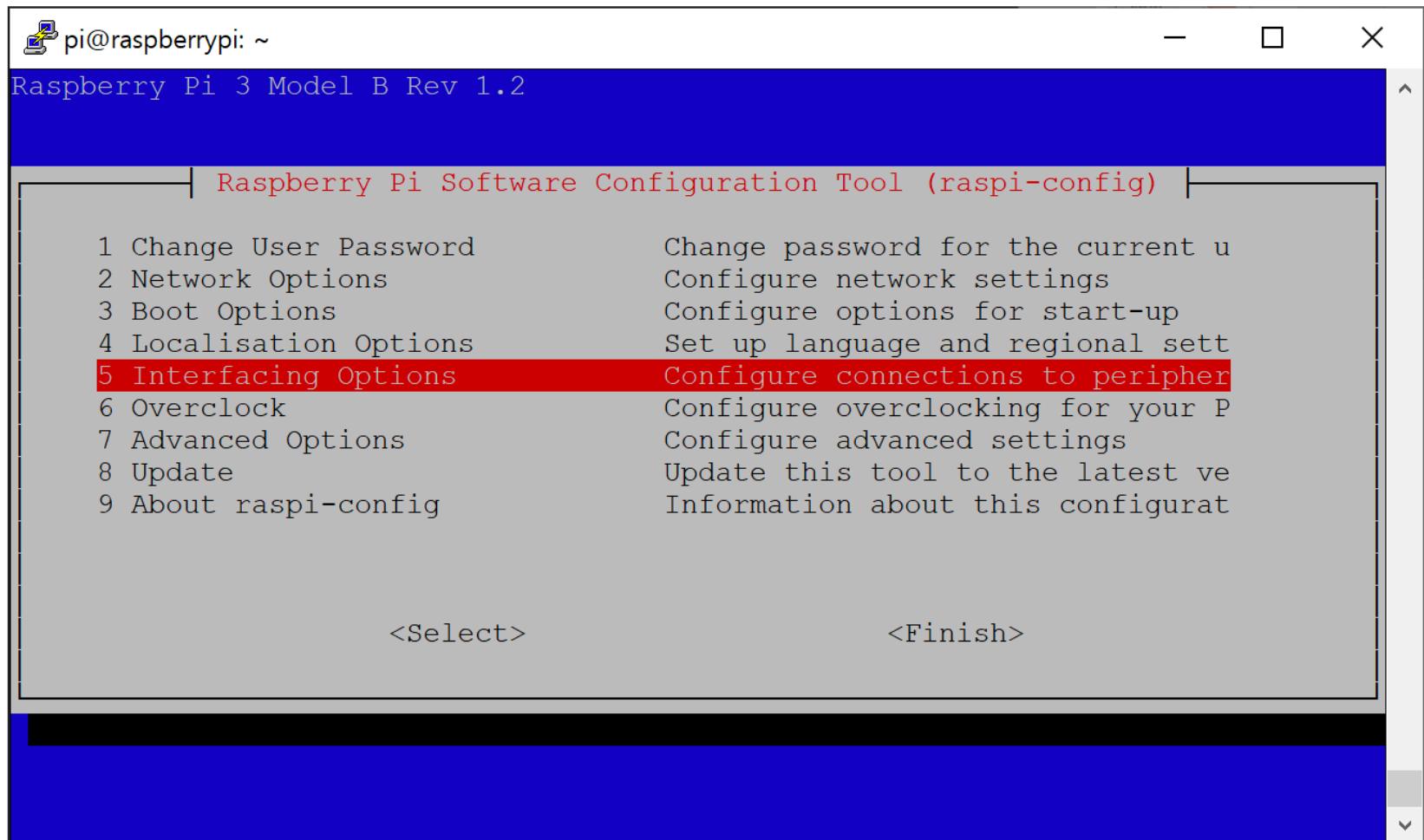
- '블루투스'로 연결된 /dev/ttyAMA0
- 하드웨어 UART 사용

- serial1 alias를 사용하여 ttyAMA0으로 연결되어 있는 상태
- 만약 RPi 3 이전 모델에서 실행하면 /dev/serial0이 연결되어 있음

- RPi 3에서는 **소프트웨어 UART (ttyS0, serial0)**이 초기에는 비 활성화되어 있으므로 **활성화**가 필요

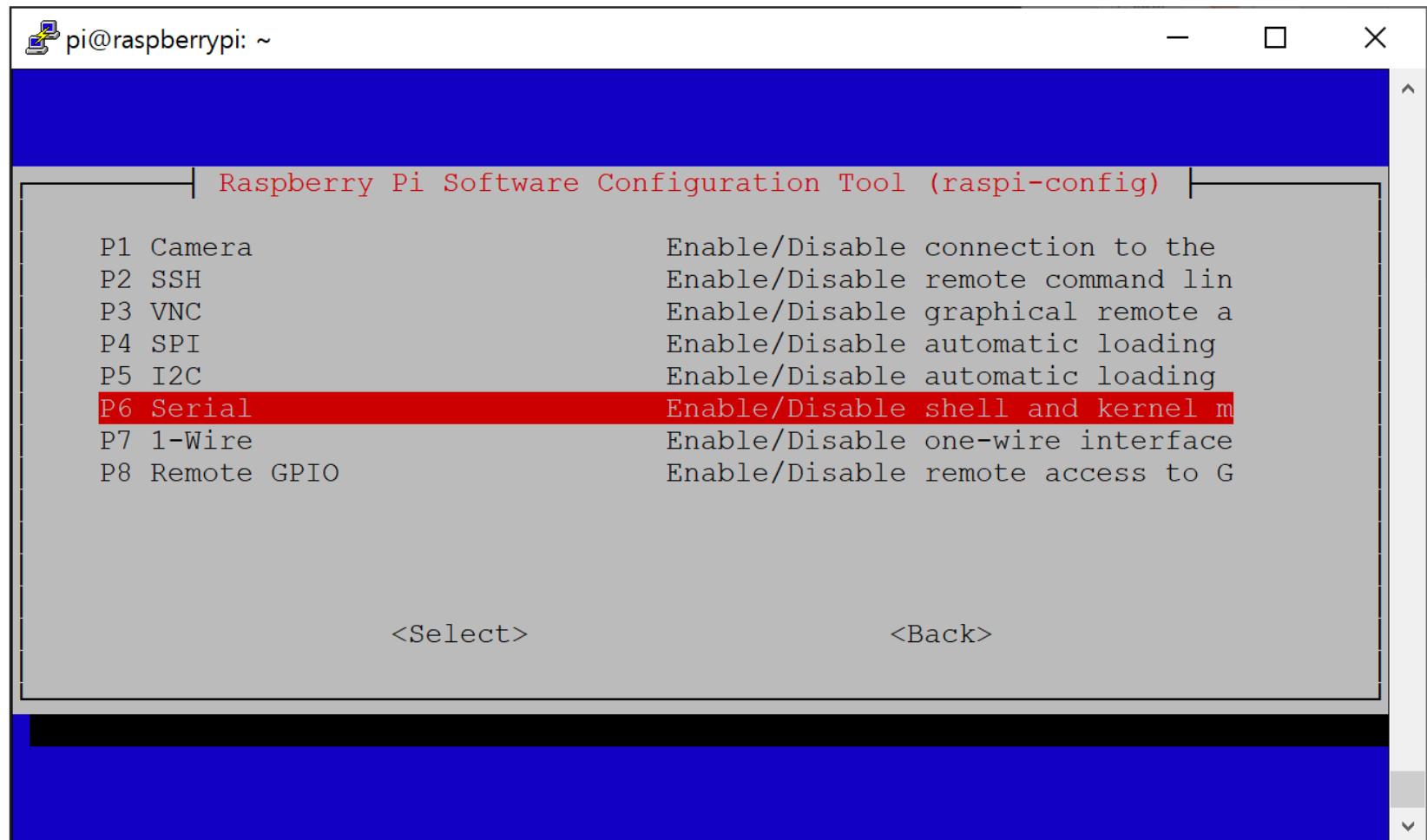
RPi UART 활성화

■ sudo raspi-config



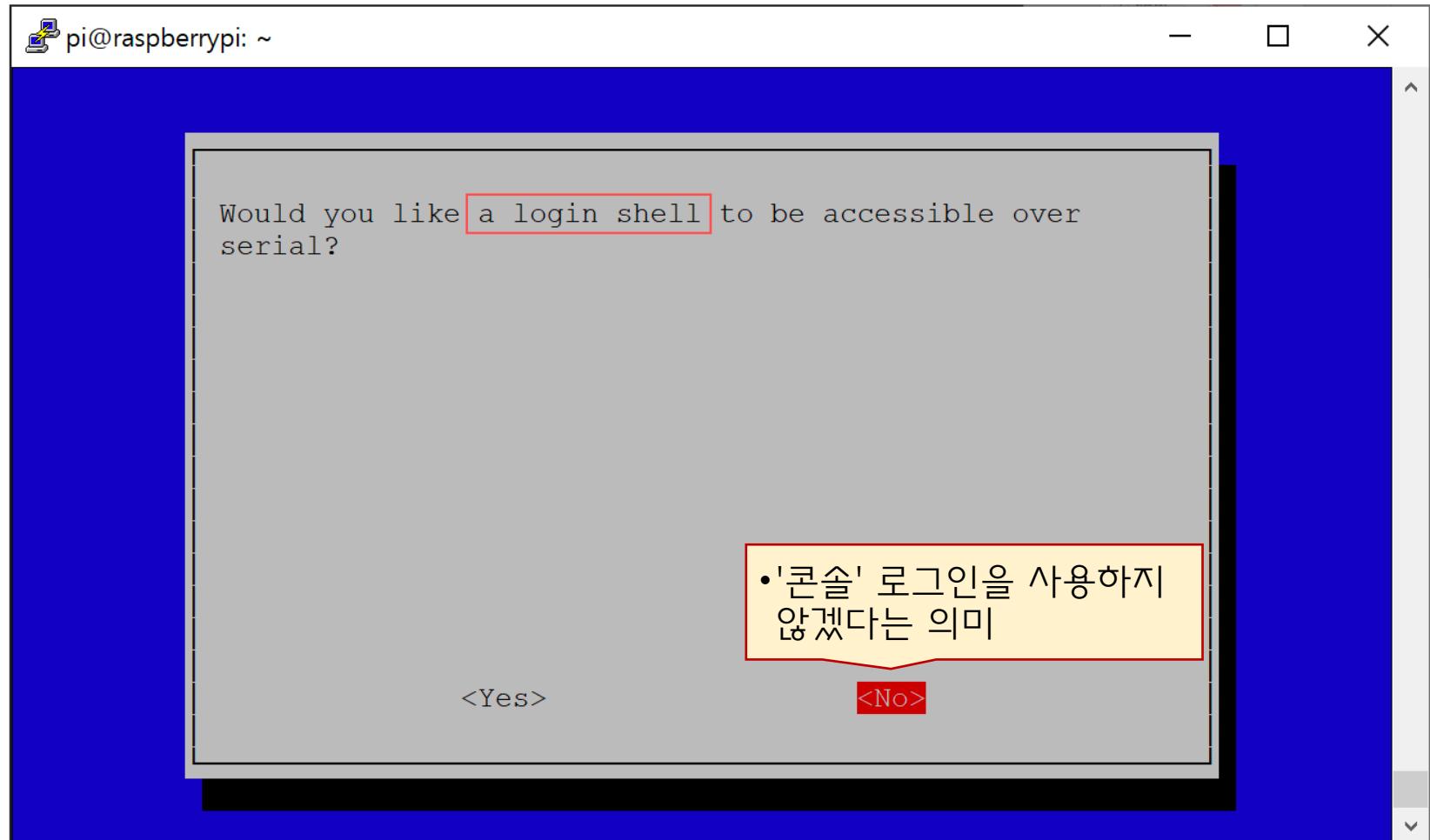
RPi UART 활성화

■ sudo raspi-config



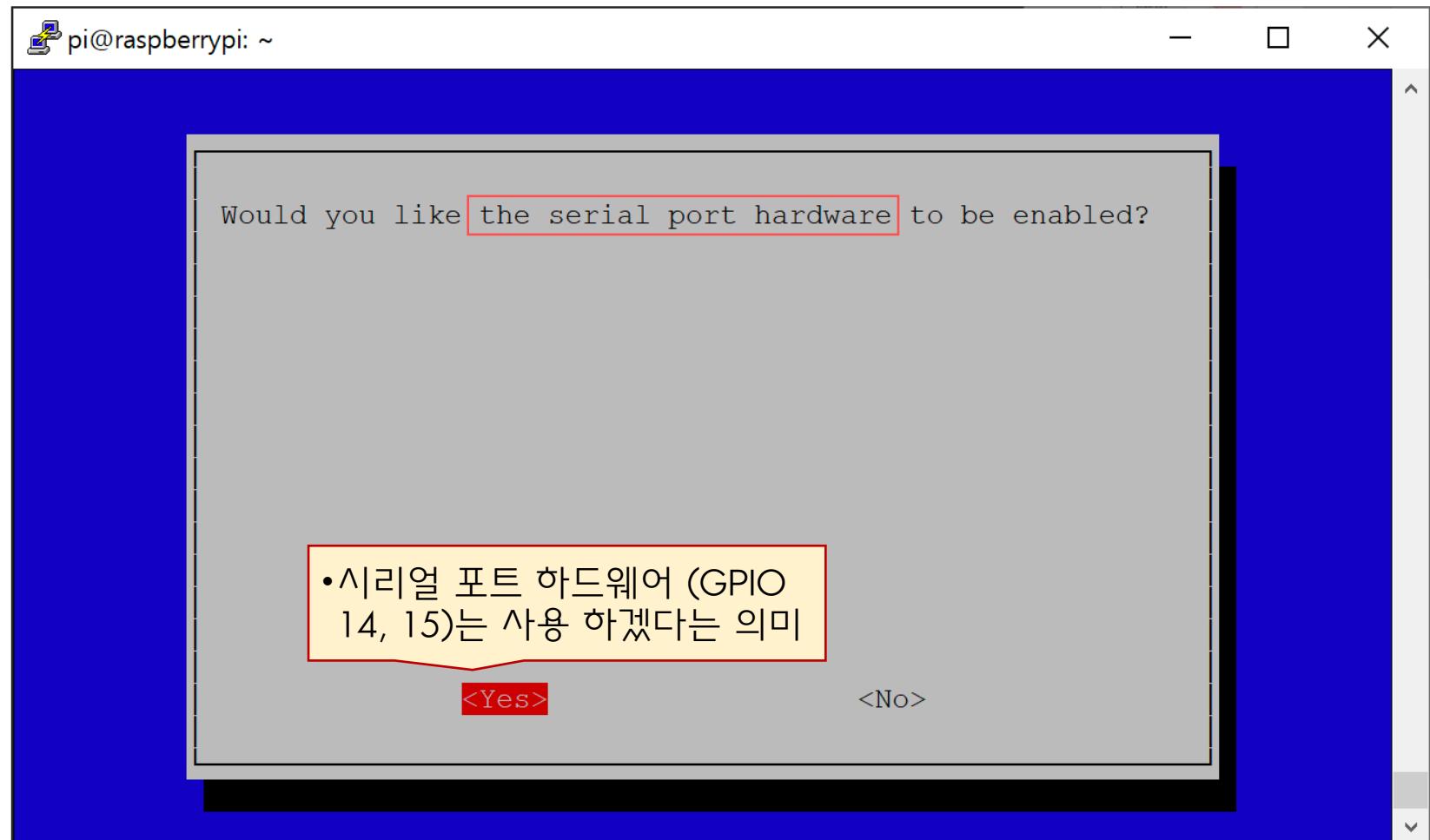
RPi UART 활성화

■ sudo raspi-config



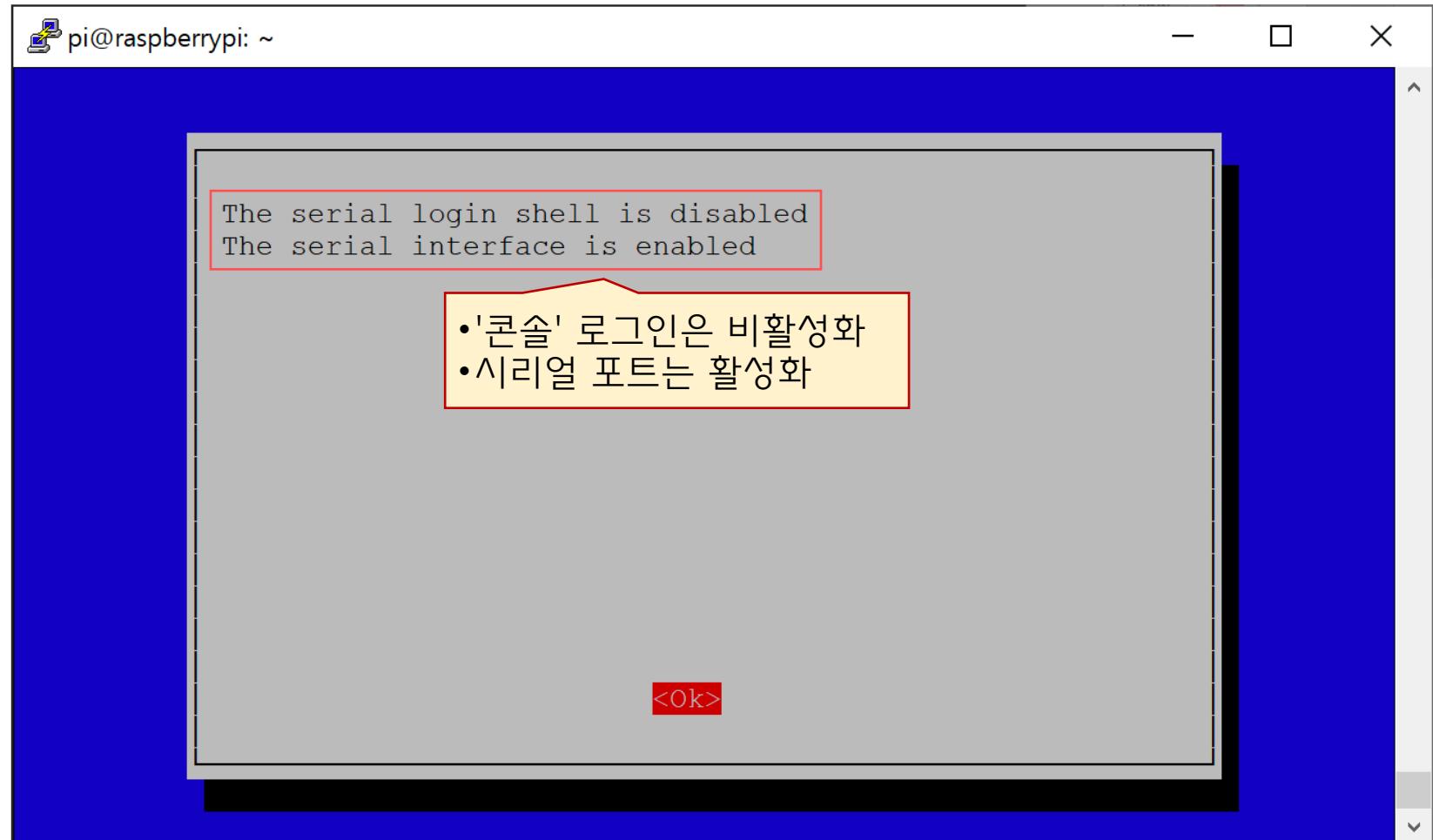
RPi UART 활성화

■ sudo raspi-config



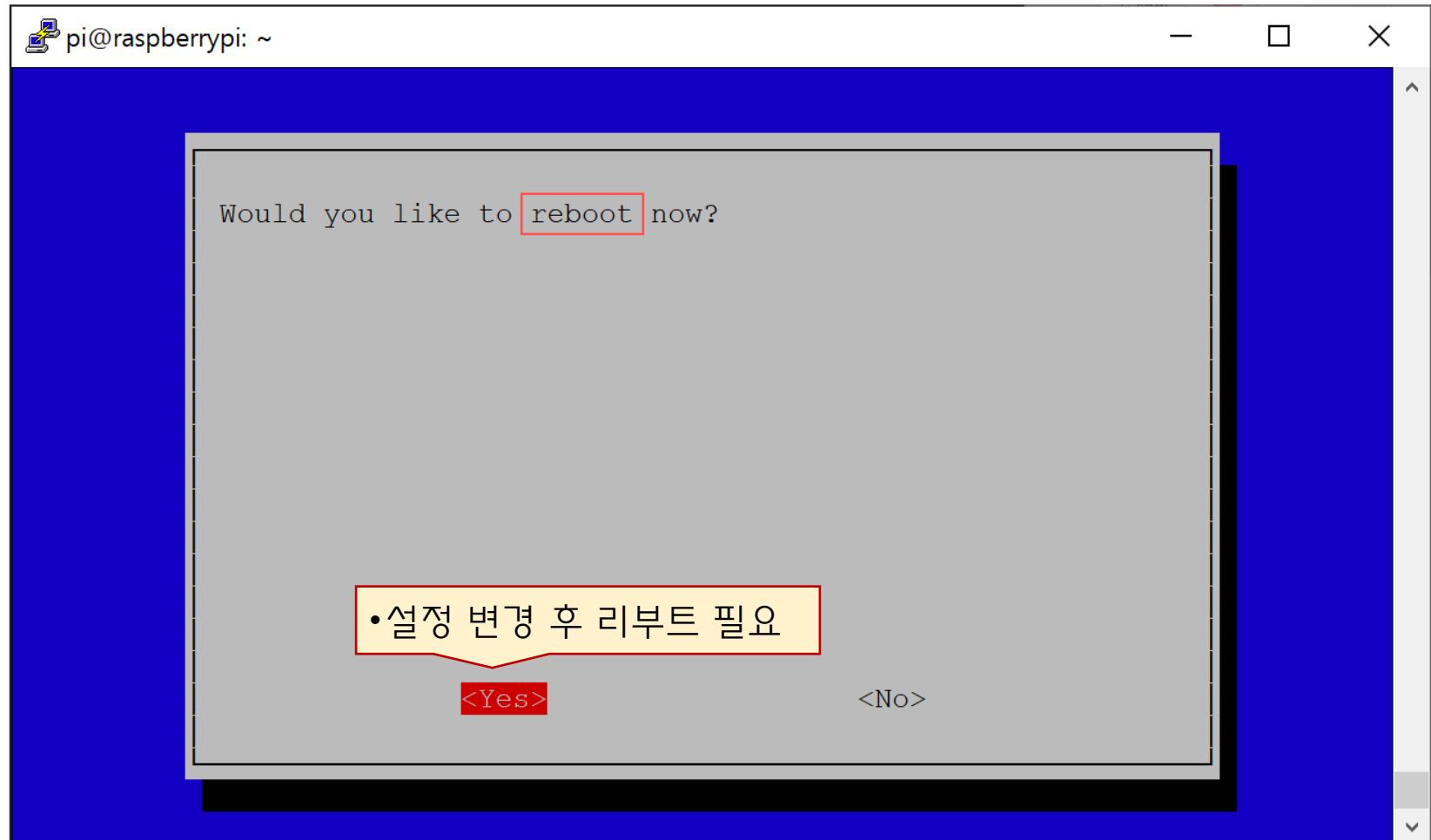
RPi UART 활성화

■ sudo raspi-config



RPi UART 활성화

■ sudo raspi-config



RPi UART 활성화

■ cat /boot/config.txt

```
pi@raspberrypi: ~
# 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
#dtoparam=i2c_arm=on
#dtoparam=i2s=on
#dtoparam=spi=on

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

# Additional overlays and parameters are documented /boot/overlays/README

# Enable audio (loads snd_bcm2835)
dtoparam=audio=on
lcd_rotate=2
enable_uart=1
pi@raspberrypi: ~
```

• 설정 파일 /boot/config.txt¹에
UART 활성화 명령 추가 확인

¹ RPi가 부팅할 때 참고하는 설정 파일로서 일반 PC의 BIOS 역할을 대신함

RPi UART 활성화

■ /dev/ttyS0과 alias serial0 확인

```
pi@raspberrypi: ~ 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/ttv45   /dev/ttv56   /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/ttyS0
/dev/tty15    /dev/tty26   /dev/tty37     /dev/tty48   /dev/tty59
/dev/tty16    /dev/tty27   /dev/tty38     /dev/tty49   /dev/tty60
/dev/tty17    /dev/tty28   /dev/tty39     /dev/tty50   /dev/tty61
/dev/tty18    /dev/tty29   /dev/tty4      /dev/tty51   /dev/tty62
pi@raspberrypi: ~ ls -l /dev/serial*
lrwxrwxrwx 1 root root 5 Mar 11 17:12 /dev/serial0 -> ttyS0
lrwxrwxrwx 1 root root 7 Mar 11 17:12 /dev/serial1 -> ttyAMA0
pi@raspberrypi: ~
```

• 'GPIO14, 15'로 연결된 /dev/ttyS0
• mini UART 사용

• serial0 alias를 사용하여 ttyS0으로 연결되어 있는 상태
• RPi 3 이전 모델에서 실행하면 serial0에 ttyAMA0이 연결 (블루투스가 없기 때문)

RPi UART 활성화 RPi 3 블루투스 교환/제거^{1,2,3,4}

■ RPi 3 기본 설정은

- `/dev/ttyAMA0` → '블루투스'로 연결
- `/dev/ttys0` → 'GPIO 시리얼 포트 (GPIO14, 15)'로 연결

■ RPi 3 시리얼 포트를 교환 하고 싶을 경우

- 즉, 높은 성능의 `/dev/ttyAMA0`을 다시 GPIO14, 15에 연결하고 싶다면,
- 디바이스 오버레이 (`dtoverlay`) **`pi3-miniuart-bt`** 수행
 - ✓ 즉 mini UART (`/dev/ttys0`)을 블루투스에 연결
- 또는 아예 블루투스를 사용하고 싶지 않다면, **`pi3-disable-bt`** 수행

■ 정리하면,

- `sudo nano /boot/config.txt`
- `dtoverlay=pi3-miniuart-bt` (또는 `pi3-disable-bt`)추가
- 저장 후 리부트 하면
 - ✓ **`serial0 -> ttyAMA0`** 확인 (코드를 `serial0`로 작성해 놓으면 관계 없이 동작)
 - ✓ **`serial1 -> ttys0`**

¹ <https://spellfoundry.com/2016/05/29/configuring-gpio-serial-port-raspbian-jessie-including-pi-3/>

² <https://www.raspberrypi.org/documentation/configuration/uart.md>

³ <https://wikidocs.net/7974>

⁴ 익스플로링 라즈베리 파일, p374

RPi PC 간 UART 통신

USB UART 변환기

RPi PC 간 UART 통신 USB UART 변환기²

- RPi UART 와 타 기기シリ얼 포트는 논리 레벨에 차이
 - RPi UART (3.3V, 논리 HIGH)
 - PCシリ얼 포트, 아두이노 Uno (5V, 논리 HIGH)
 - 해결 방법은 **논리 레벨 변환 회로**¹ (즉, 3.3 V <-> 5 V 변환)를 사용
- 특히 PCシリ얼 포트는 USB가 대체하는 추세
 - 3.3V (또는 5V) UART 신호를 USB (5V) 신호로 변환하는 방법이 필요
 - 해결방법은 **USB-to-TTL (Serial UART) 변환기** 사용
 - 이는 USB 인터페이스를 제공하는 기기 (예, PC, RPi)에 편리하게 **다수의 UART 장치들을 연결할 수 있음**



WaveShare PL2303 USB UART 변환기³

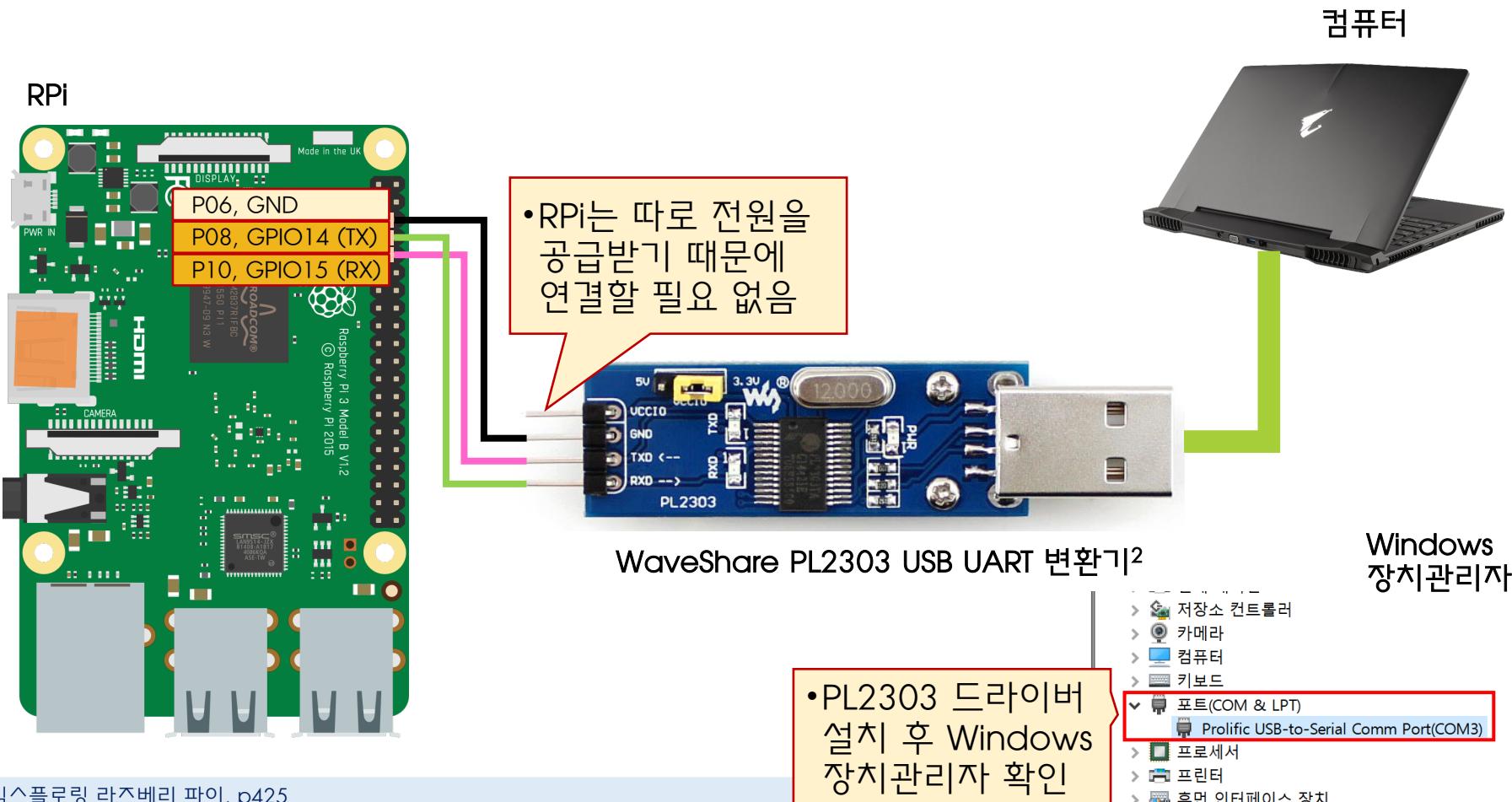
¹ 익스플로링 라즈베리 파이, p385

² 익스플로링 라즈베리 파이, p425

³ [https://www.waveshare.com/wiki/PL2303_USB_UART_Board_\(type_A\)](https://www.waveshare.com/wiki/PL2303_USB_UART_Board_(type_A))

RPi PC 간 UART 통신 USB UART 변환기

- USB UART 변환기를 이용하여 PC와 RPi 연결
 - Windows 용 PL2303 드라이버 설치³



¹ 익스플로링 라즈베리 파이, p425

² [https://www.waveshare.com/wiki/PL2303_USB_UART_Board_\(type_A\)](https://www.waveshare.com/wiki/PL2303_USB_UART_Board_(type_A))

³ https://www.waveshare.com/wiki/File:PL2303_Windows_Driver.7z

RPi PC 간 UART 통신 RPI → PC



■ PC와シリ얼통신 테스트

pi@raspberrypi: ~

```
pi@raspberrypi: ~ $ stty -F /dev/ttys0
speed 9600 baud; line = 0;
-brkint -imaxbel
pi@raspberrypi: ~ $ stty -F /dev/serial0
speed 9600 baud; line = 0;
-brkint -imaxbel
pi@raspberrypi: ~ $ stty -F /dev/serial0 57600
pi@raspberrypi: ~ $ stty -F /dev/serial0
speed 57600 baud; line = 0;
-brkint -imaxbel
pi@raspberrypi: ~ $ stty -F /dev/serial0 9600
pi@raspberrypi: ~ $ stty -F /dev/serial0
speed 9600 baud; line = 0;
-brkint -imaxbel
pi@raspberrypi: ~ $ date > /dev/serial0
pi@raspberrypi: ~ $ echo "Hello, world!" > /dev/serial0
pi@raspberrypi: ~ $
```

- stty 명령은 리눅스 터미널 설정 확인 변경^{1,2,3}
- stty [-F Device]
- 해당 터미널 설정 속도 (즉 9600) 확인 가능

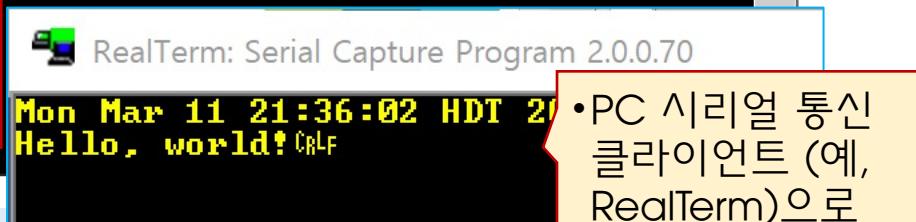
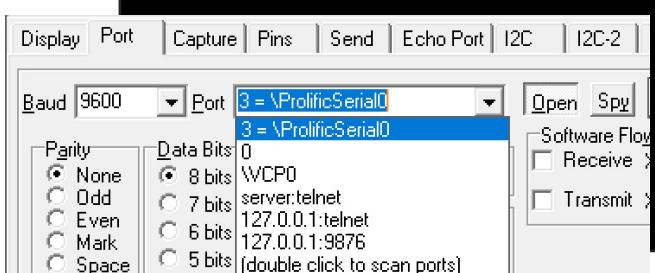
- 실제 이름 ttys0과 별칭 serial0 모두 사용 가능

- serial0 속도를 57600 으로 설정 후 확인

- serial0 속도를 9600 으로 설정 후 확인

- シリ얼포트 파일 시스템에
재지정자 (redirection) '>'를
이용하여 직접 쓰기를 통해
シリ얼통신으로 전달

•シリ얼포트,
보드레이트
설정 필요



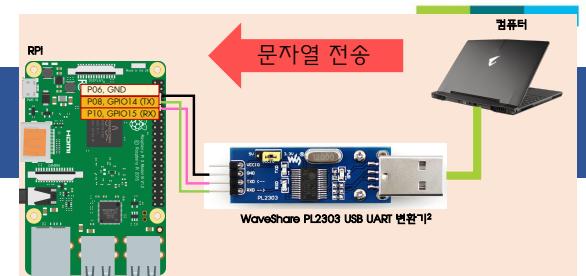
- PCシリアル통신
클라이언트 (예,
RealTerm)으로
메시지 수신 확인

¹ <https://www.computerhope.com/unix/sttym.htm>

² <https://linux.101hacks.com/unix/stty/>

³ http://www.armadeus.org/wiki/index.php?title=Serial_ports_usage_on_Linux

RPi PC 간 UART 통신 PC → RPi

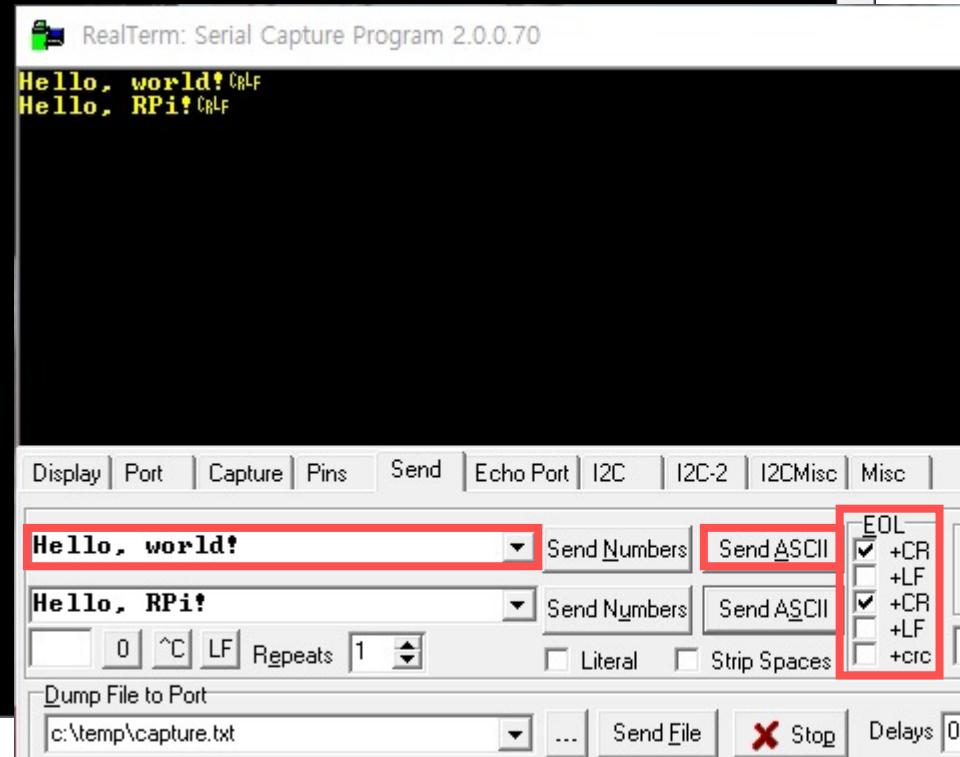


- RPi 확인 방법: cat < \dev\serial0

```
pi@raspberrypi: ~
pi@raspberrypi: ~ $ cat < /dev/serial0
Hello, world!
Hello, RPi!
```

•/dev/serial0 으로 들어오는 메시지 출력

- 처음에는 빈 화면 유지
- PC에서 시리얼 통신 클라이언트 (예, RealTerm)을 통해 메시지를 보내면 수신



- PC 시리얼 통신 클라이언트 화면
- PC에서 메시지를 전송하면 RPi 화면 (또는 SSH 원격 접속 화면)에서 확인 가능

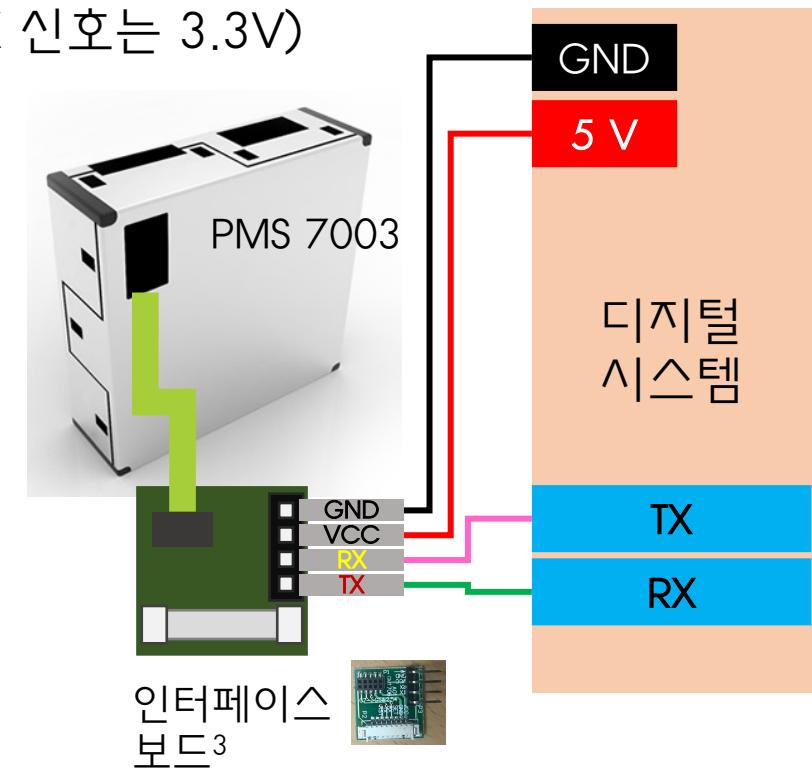
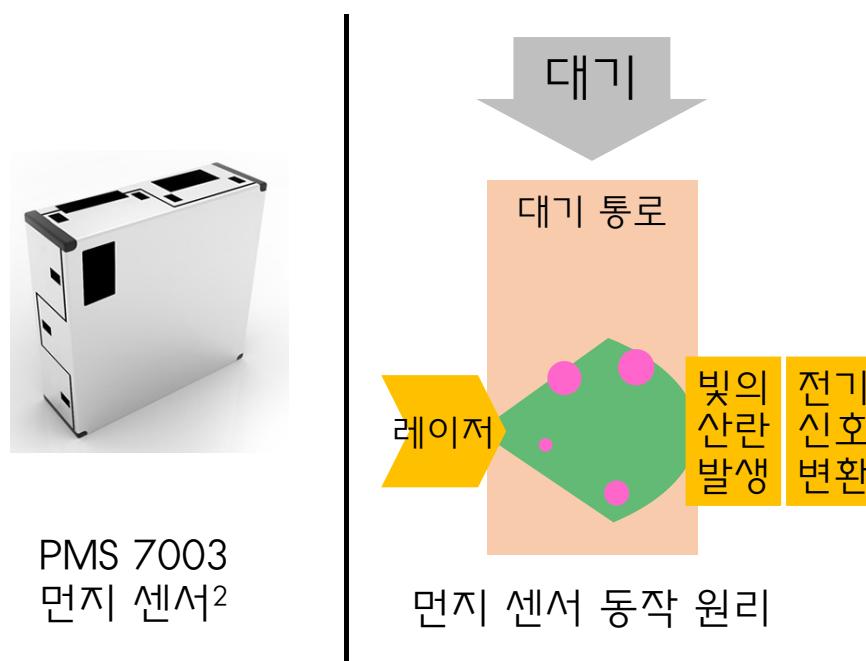
먼지 | 센서

PMS 7003

먼지 센서 동작 원리

■ PMS 7003¹

- 대기 중 부유하는 입자 (즉, 먼지) 집중도 측정
- 디지털 인터페이스: UART
- 감지 입자 크기: 지름이 최소 $0.3 \mu\text{m}$ 입자 까지 측정
- 동작 전압: (팬 사용으로) 5V (TX/RX 신호는 3.3V)



¹ <http://www.plantower.com/en/content/?110.html>

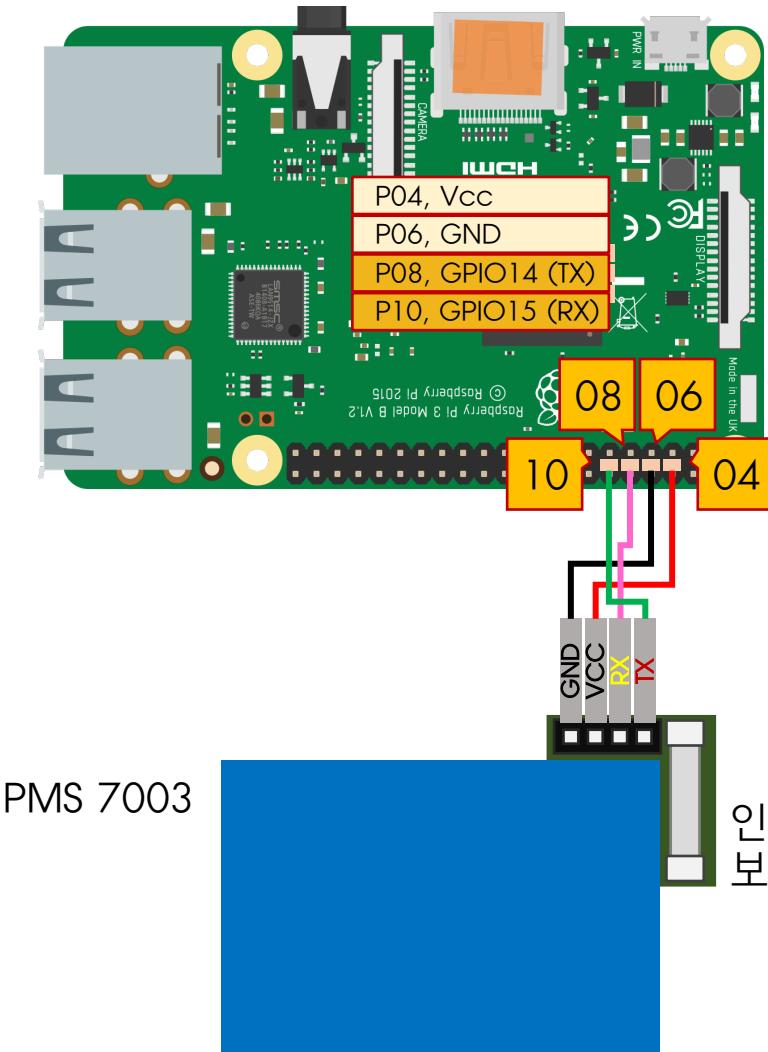
² <https://www.eleparts.co.kr/goods/view?no=4208690>

³ <https://www.eleparts.co.kr/goods/view?no=6060637>

먼지 센서 회로 구성 (USB 지원 않는 기기)

Practice

■ 먼지 센서와 RPi 연결



- (4핀) 점퍼 케이블을 이용하여 그림과 같이 연결
- 정면에서 보았을 때 (데이터시트 확인!), GND 핀, RPi GND 연결
VCC 핀, **RPi 5V** 연결
RX 핀, **RPi P08** (GPIO14) 연결
TX 핀, **RPi P10** (GPIO15) 연결

센서와 인터페이스 보드 연결



먼지 센서 테스트

Practice

■ RPi 확인 방법: od -A x -t x1z -v /dev/serial0

 pi@raspberrypi: ~

```
pi@raspberrypi: ~ $ od -A x -t x1z -v /dev/serial0
```

```
000000 42 4d 00 1c 00 01 00 03 00 03 00 02 00 04 00 04
000010 00 50 00 00 00 00 00 00 00 00 00 00 00 00 01 0c
000020 42 4d 00 1c 00 03 00 03 00 03 00 04 00 04 00 04
000030 00 64 00 00 00 00 00 00 00 00 00 00 00 00 01 24
000040 42 4d 00 1c 00 03 00 03 00 03 00 04 00 04 00 04
000050 00 64 00 00 00 00 00 00 00 00 00 00 00 00 01 24
000060 42 4d 00 1c 00 03 00 03 00 04 00 04 00 04 00 06
000070 00 78 00 10 00 00 00 03 00 03 00 03 00 00 01 54
000080 42 4d 00 1c 00 01 00 03 00 04 00 02 00 04 00 06
000090 00 78 00 10 00 00 00 03 00 03 00 03 00 00 01 50
0000a0 42 4d 00 1c 00 01 00 03 00 04 00 02 00 04 00 06
0000b0 00 8c 00 10 00 00 00 03 00 03 00 03 00 00 01 64
0000c0 42 4d 00 1c 00 02 00 04 00 07 00 04 00 06 00 0a
```

- 시리얼 포트로 들어오는 데이터를 16 진수로 확인

- 데이터 시트를 통해 전송 프로토콜 바이트 별 정보 분석

- '42 4d'로 시작되는 데이터가 보이지 않는 경우,
- Linux 'screen' 툴 (리눅스 시리얼 통신 프로그램)을 이용
(screen이 설치되지 않았다면)
- \$ sudo apt-get install screen
- \$ screen /dev/serial0 9600
(읽을 수 없는 데이터가 들어올 때 CTRL A+D 를 눌러 빠져 나오기)
- \$ screen -list (열려진 screen 번호 확인, 예, 13896 pts-4.raspberrypi)
- \$ screen -X -S 13896 quit
- (다시 데이터 확인) \$ od -A x -t x1z -v /dev/serial0

- 컴퓨터마다 값이 다름 주의

먼지 센서 전송 프로토콜 정보¹

바이트	데이터	설명
01	0x42	시작 바이트
02	0x4d	시작 바이트
03	프레임 길이 상위 8비트	
04	프레임 길이 하위 8비트	프레임 길이 = $2 * 13 + 2$ (data+check 바이트) $\rightarrow 1C$
05	Data 1 상위 8비트	PM-1.0 초미세먼지 농도 unit $\mu\text{g}/\text{m}^3$ (CF=1, 표준 입자)
06	Data 1 하위 8비트	
07	Data 2 상위 8비트	PM-2.5 초미세먼지 농도 $\mu\text{g}/\text{m}^3$ (CF=1, 표준 입자)
08	Data 2 하위 8비트	
09	Data 3 상위 8비트	PM-10 미세먼지 농도 $\mu\text{g}/\text{m}^3$ (CF=1, 표준 입자)
10	Data 3 하위 8비트	
11	Data 4 상위 8비트	PM-1.0 초미세먼지 농도 $\mu\text{g}/\text{m}^3$ (대기 상태에서)
12	Data 4 하위 8비트	
13	Data 5 상위 8비트	PM-2.5 초미세먼지 농도 $\mu\text{g}/\text{m}^3$ (대기 상태에서)
14	Data 5 하위 8비트	
15	Data 6 상위 8비트	PM-10 미세먼지 농도 $\mu\text{g}/\text{m}^3$ (대기 상태에서)
16	Data 6 하위 8비트	
17	Data 7 상위 8비트	0.1 리터 공기 내 직경 0.3 μm 이상 입자 개수
18	Data 7 하위 8비트	
19	Data 8 상위 8비트	0.1 리터 공기 내 직경 0.5 μm 이상 입자 개수
20	Data 8 하위 8비트	
21	Data 9 상위 8비트	0.1 리터 공기 내 직경 1.0 μm 이상 입자 개수
22	Data 9 하위 8비트	
23	Data 10 상위 8비트	0.1 리터 공기 내 직경 2.5 μm 이상 입자 개수
24	Data 10 하위 8비트	
25	Data 11 상위 8비트	0.1 리터 공기 내 직경 5.0 μm 이상 입자 개수
26	Data 11 하위 8비트	
27	Data 12 상위 8비트	0.1 리터 공기 내 직경 10 μm 이상 입자 개수
28	Data 12 하위 8비트	
29	Data 13 상위 8비트	예약 (reserved)
30	Data 13 하위 8비트	
31	Data and check 상위 8비트	Check code =
32	Data and check 하위 8비트	Start character1 + Start character 2 + ... + data 13 Low 8 bits

- 32 바이트 메시지를 보낼 때 맨 첫머리에 송신 시작 바이트를 수신을 확인하고 메시지 수신 시작

- 32 바이트 중 데이터 추출
11-12: 초미세먼지
13-14: 초미세먼지
15-16: 미세먼지

먼지 센서 테스트

Practice

■ RPi 확인 방법: od -A x -t x1z -v /dev/serial0

pi@raspberrypi: ~

pi@raspberrypi: ~ \$ od -A x -t x1z -v /dev/serial0

```

000000 42 4d 00 1c 00 01 00 03 00 03 00 02 00 04 00 04
000010 00 50 00 00 00 00 00 00 00 00 00 00 00 00 01 0c
000020 42 4d 00 1c 00 03 00 03 00 03 00 04 00 04 00 04
000030 00 64 00 00 00 00 00 00 00 00 00 00 00 00 01 24
000040 42 4d 00 1c 00 03 00 03 00 03 00 04 00 04 00 04
000050 00 64 00 00 00 00 00 00 00 11-12 13-14 15-16
000060 42 4d 00 1c 00 03 00 03 00 04 00 04 00 04 00 06
000070 00 78 00 10 00 00 00 03 00 03 00 03 00 00 01 54
000080 42 4d 00 1c 00 01 00 03 00 04 00 02 00 04 00 06
000090 00 78 00 10 00 00 00 03 00 03 00 03 00 00 01 50
0000a0 42 4d 00 1c 00 01 00 03 00 04 00 02 00 04 00 06
0000b0 00 8c 00 10 00 00 00 03 00 03 00 03 00 00 01 64
0000c0 42 4d 00 1c 00 02 00 04 00 07 00 04 00 06 00 0a
0000d0 00 b4 00 10 00 04 00 03 00 03 00 03 00 00 01 9d
0000e0 42 4d 00 1c 00 04 00 06 00 0a 00 06 00 08 00 0e
0000f0 00 b4 00 10 00 04 00 04 00 04 00 04 00 00 01 af
000100 42 4d 00 1c 00 05 00 07 00 0c 00 08 00 0a 00 0f
000110 00 c8 00 20 00 04 00 08 00 06 00 06 00 00 01 e4
000120 42 4d 00 1c 00 09 00 0c 00 1c 00 0c 00 0f 00 1f
000130 00 f0 00 40 00 0c 00 25 00 22 00 1f 00 00 02 b8
000140 42 4d 00 1c 00 09 00 10 00 25 00 0c 00 14 00 26
000150 01 18 00 50 00 10 00 31 00 2c 00 29 00 00 02 2e
000160 42 4d 00 1c 00 09 00 00 2d 00 0c 00 17 00 2a 01

```

• 시리얼 포트로 들어오는 데이터를 16 진수로 확인

• 데이터 시트를 통해 전송 프로토콜 바이트 별 정보 분석

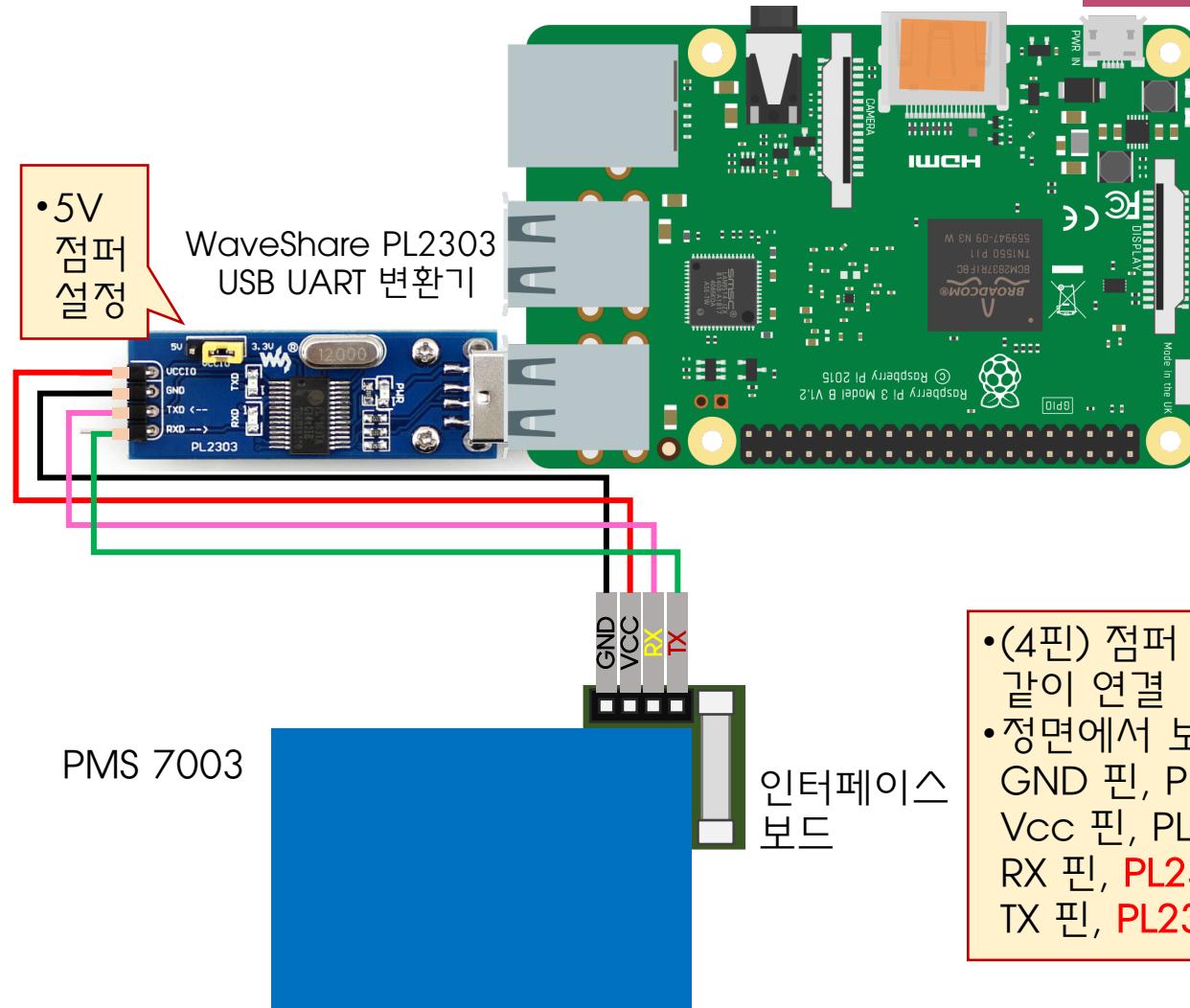
PM-1.0 PM-2.5 PM-10

먼지 센서 회로 구성 (USB 지원 기기)

Practice

■ 먼지 센서와 RPi 연결

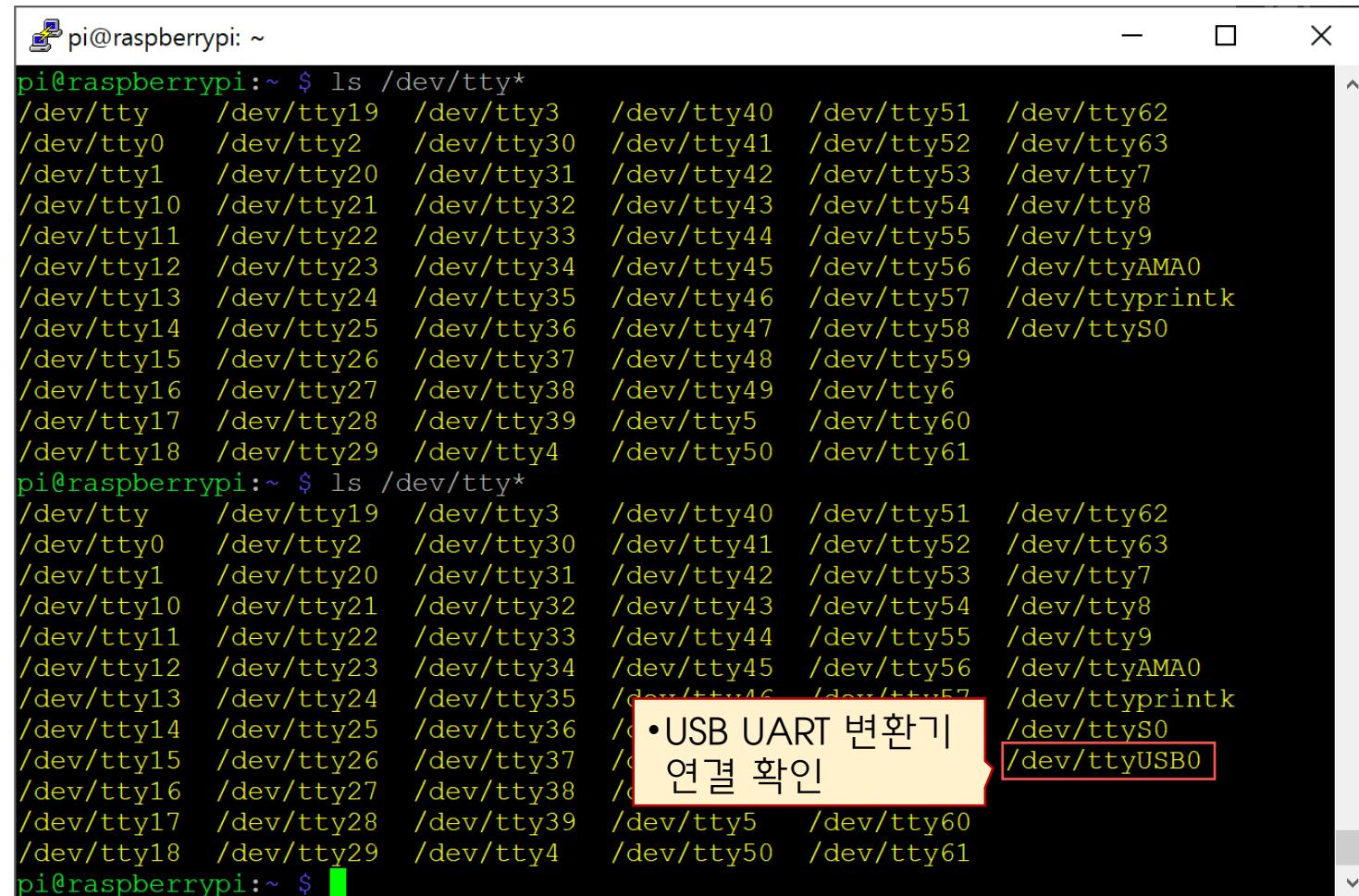
- RPi 가 지원하는シリ얼 포트 (`/dev/serial0`)를 사용하지 않고 편리하게 UART 통신 확장 가능



- (4핀) 점퍼 케이블을 이용하여 그림과 같이 연결
- 정면에서 보았을 때 (데이터시트 확인!), GND 핀, PL2303 GND 연결
Vcc 핀, PL2303 Vcc 연결
RX 핀, **PL2303 TX** 연결
TX 핀, **PL2303 RX** 연결

먼지 센서 회로 구성 (USB 지원 기기)

■ /dev/ttyUSB0 생성 확인



```
pi@raspberrypi:~ $ 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/ttyS0
/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:~ $ 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/ttyS0
/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:~ $
```

먼지 센서 리눅스 프로그래밍 (USB UART 시리얼 통신)

```
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include <iostream>
#include <iomanip>
#include <ctime>
#include <chrono>
using namespace std;

int main(int argc, char *argv[]){
    int dust;

    if ((dust = open("/dev/ttyUSB0", O_RDWR | O_NOCTTY | O_NDELAY)) < 0){
        perror("UART: Failed to open the file.\n");
        return -1;
    }

    struct termios options;
    tcgetattr(dust, &options);
    options.c_cflag = B9600 | CS8 | CREAD | CLOCAL;
    options.c_iflag = IGNPAR | ICRNL;
    tcflush(dust, TCIFLUSH);
    fcntl(STDIN_FILENO, F_SETFL, O_NONBLOCK);
    tcsetattr(dust, TCSANOW, &options);
}
```

- 시리얼 통신을 위한 헤더 파일: fcntl.h, unistd.h, termios.h
- 표준 입출력을 위한 헤더 파일: iostream, iomanip
- 시간 출력을 위한 헤더 파일: ctime, chrono

- USB UART 변환기를 RPi에 연결했을 때 /dev/ttyUSB0에 가상 파일 시스템이 생성되었다고 가정
- 2개 이상 변환기를 사용하면 해당 값 수정 (예, ttyUSBX) 필요
- USB 포트 접근 권한 획득에 성공하면 **dust**에 저장
- **dust** 변수를 이용해 USB 포트로 데이터 송수신이 가능

- **termios** 구조체는 termios.h¹에 정의되어 있고 터미널 제어를 위한 상수, 데이터 형, 함수를 정의하고 있음
- 시리얼 포트 통신을 위한 설정 가능
- 보드레이트를 **9600**으로 설정

¹ <https://linux.die.net/man/3/termios>

² http://neosrtos.com/docs posix_api/termios.html

³ 익스플로링 라즈베리 파이, p380 참조, exploringrpi/chp08/uart/server/server.c

먼지 센서 리눅스 프로그래밍 (USB UART 시리얼 통신)

Practice

```
unsigned char cmdPassive[7] = {0x42, 0x4d, 0xe1, 0x00, 0x00, 0x01, 0x70};  
unsigned char cmdRead[7] = {0x42, 0x4d, 0xe2, 0x00, 0x00, 0x01, 0x71};
```

- Passive 모드 설정
- 데이터 요청

```
write(dust, cmdPassive, 7);
```

- PMS 7003가 연결된 USB UART 포트로 cmdPassive 메시지 전송
- PMS 7003를 passive 모드로 설정

Start Byte 1	Start Byte 2	Command	Data 1	Data 2	Verify Byte 1	Verify Byte 2
0x42	0x4d	CMD	DATAH	DATAL	LRCH	LRCL

1. Command Definition

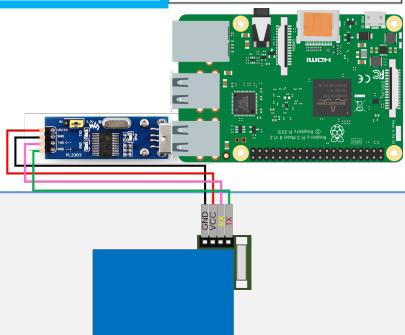
CMD	DATAH	DATAL	说明
0xe2	X	X	Read in passive mode
0xe1	X	00H-passive 01H-active	Change mode
0xe4	X	00H-sleep 01H-wakeup	Sleep set

2. Answer

0xe2: 32 bytes , same as appendix I

3. Verify Bytes :

Add of all the bytes except verify bytes.



- PMS 7003 데이터 시트 참고
- UART로 7 바이트 메시지를 PMS 7003에게 보내어 동작 설정과 변경이 가능
- 동작 리스트
 - sleep: 전력을 아끼면서 대기,
 - wakeup: sleep 상태 깨우기,
 - passive: 데이터를 요청할 때만 보내기**
 - active: 데이터를 요청하지 않아도 보내기
- cmdPassive[7]은 passive 모드로 설정하기 위한 커맨드 메시지
- cmdRead[7]은 측정한 데이터를 보낼 것을 요청하는 메시지
- 'Verify Byte'는 시리얼 통신 중 전송 오류 검사가 목적이며 직접 계산하여 채워 넣기

먼지 센서 리눅스 프로그래밍 (USB UART 시리얼 통신)

Practice



```
unsigned char c;
unsigned char data[32];
int bit_start = 0;
int count = 0;
int pm1_0, pm2_5, pm10;
```

- data: 수신 데이터 저장 배열
- bit_start: 0이면 대기 상태
1이면 '42' 가 수신 된 상태
2이면 '4D' 가 수신 된 상태, 데이터 수신 준비
- count: 수신 데이터 저장 바이트 수
- pm1_0, pm2_5, pm10: 미세먼지 농도 저장 변수

```
write(dust, cmdRead, 7);
```

- PMS 7003에게 데이터를 보낼 것을 요청

```
while (1) {
    usleep(1000);
    read(dust, &c, 1);
```

- UART (=9600 bps) 속도를 고려하여 루프 대기
- 시리얼 포트 (먼지 센서)로부터 1 바이트 읽기

```
if (bit_start == 0) {
    if (c == 0x42) {
        bit_start = 1;
        data[0] = c;
        count = 1;
    }
}
```

- 대기 상태일 때 '42' 수신 했으면 bit_start를 1로 설정하고 다음 '4D' 수신 대기

```
} else if (bit_start == 1) {
    if (c == 0x4d) {
        bit_start = 2;
        data[1] = c;
        count = 2;
    }
}
```

- ('42' 수신된) 대기 상태일 때 '4D' 수신 했으면 bit_start를 2로 설정하고 다음에 들어올 30 바이트를 데이터로 수신 대기

```
} else if (bit_start == 2) {
    data[count++] = c;
```

- ('42', '4d' 수신된) 데이터 수신 상태이므로 시작 바이트 포함 32 바이트를 수신하여 data에 저장

먼지 센서

(USB UART 시리얼 통신)

Practice

- 32 바이트를
수신 하면 실행

42 4D 00 00 00 00 00 00 00 01 41

```
if (count >= 32) {
    auto timenow = chrono::system_clock::to_time_t(chrono::system_clock::now());
    cout << ctime(&timenow);
```

- 현재 시간 출력

```
for (int i = 0; i < 16; i++)
    cout << hex << setw(2) << setfill('0') << (int)data[i] << " ";
cout << endl;
for (int i = 16; i < 32; i++)
    cout << hex << setw(2) << setfill('0') << (int)data[i] << " ";
cout << endl;
```

- 01-16 수신 바이트 출력

- 17-32 수신 바이트 출력

```
pm1_0 = (int)((data[10] << 8) | data[11]);
pm2_5 = (int)((data[12] << 8) | data[13]);
pm10 = (int)((data[14] << 8) | data[15]);
cout << "pm1.0: " << dec << pm1_0 << " pm2.5: " << pm2_5 << " pm10: " << pm10 << endl;
```

- 미세먼지 값을 위한 상위, 하위 바이트 합치기
- 10-11, 12-13, 14-15 번째 배열 요소 사용

```
count = 0;
bit_start = 0;
for (int i = 0; i < 32; i++)
    data[i] = 0;
```

- 초기화

- 측정한 값 출력
- 초미세먼지 농도: pm1_0 $\mu\text{g}/\text{m}^3$
- 초미세먼지 농도: pm2_5 $\mu\text{g}/\text{m}^3$
- 미세먼지 농도: pm10 $\mu\text{g}/\text{m}^3$

```
usleep(5000000);
write(dust, cmdRead, 7);
```

- 5초 대기 후 다음 데이터 요청

```
}
```

```
}
```

```
}
```

```
close(dust);
return 0;
```

먼지 센서 리눅스 프로그래밍 (USB UART 시리얼 통신)

Practice

uart_pms7003_passive.cpp 파일을 RPi에 복사
 \$ g++ uart_pms7003_passive.cpp -o uart
 \$ uart

```
pi@raspberrypi: ~/ex
42 4d 00 1c 00 0a 00 10 00 15 00 11 00 14 00 19
PM-1.0 00 2 PM-2.5 04 00 PM-10 0 00 00 00 01 44
pm1.0: 17 pm2.5: 20 pm10: 25
Thu Mar 21 03:04:15 2019
42 4d 00 1c 00 0c 00 0a 00 10 00 0f 00 11 00 14
01 04 00 20 00 04 00 00 00 00 00 00 00 01 31
pm1.0: 15 pm2.5: 17 pm10: 20
Thu Mar 21 03:04:20 2019
42 4d 00 1c 00 0c 00 0a 00 0f 00 0f 00 11 00 12
01 18 00 20 00 04 00 00 00 00 00 00 00 01 42
pm1.0: 15 pm2.5: 17 pm10: 18
Thu Mar 21 03:04:25 2019
42 4d 00 1c 00 0c 00 0c 00 0f 00 0f 00 0f 00 12
01 18 00 20 00 04 00 00 00 00 00 00 00 01 3f
pm1.0: 15 pm2.5: 15 pm10: 18
Thu Mar 21 03:04:30 2019
42 4d 00 1c 00 0c 00 0c 00 0f 00 0f 00 0f 00 12
01 18 00 30 00 04 00 00 00 00 00 00 00 01 4f
pm1.0: 15 pm2.5: 15 pm10: 18
Thu Mar 21 03:04:35 2019
42 4d 00 1c 00 0c 00 0c 00 10 00 0f 00 0f 00 14
01 04 00 30 00 04 00 00 00 00 00 00 00 01 3e
pm1.0: 15 pm2.5: 15 pm10: 20
```

• 미세먼지 농도 값 활용 방법 개인 실습

Summary

- UART bus in RPi
- Dust sensor PMS7003

Thank you

Questions?

Contact: eclass.sch.ac.kr
(순천향대학교 학습플랫폼 LMS)