

IoT 프로그래밍

라즈베리파이 **sensor** 제어 및 소켓프로그래밍이용한 원격제어

Kyusik Chung
kchung@ssu.ac.kr

목차

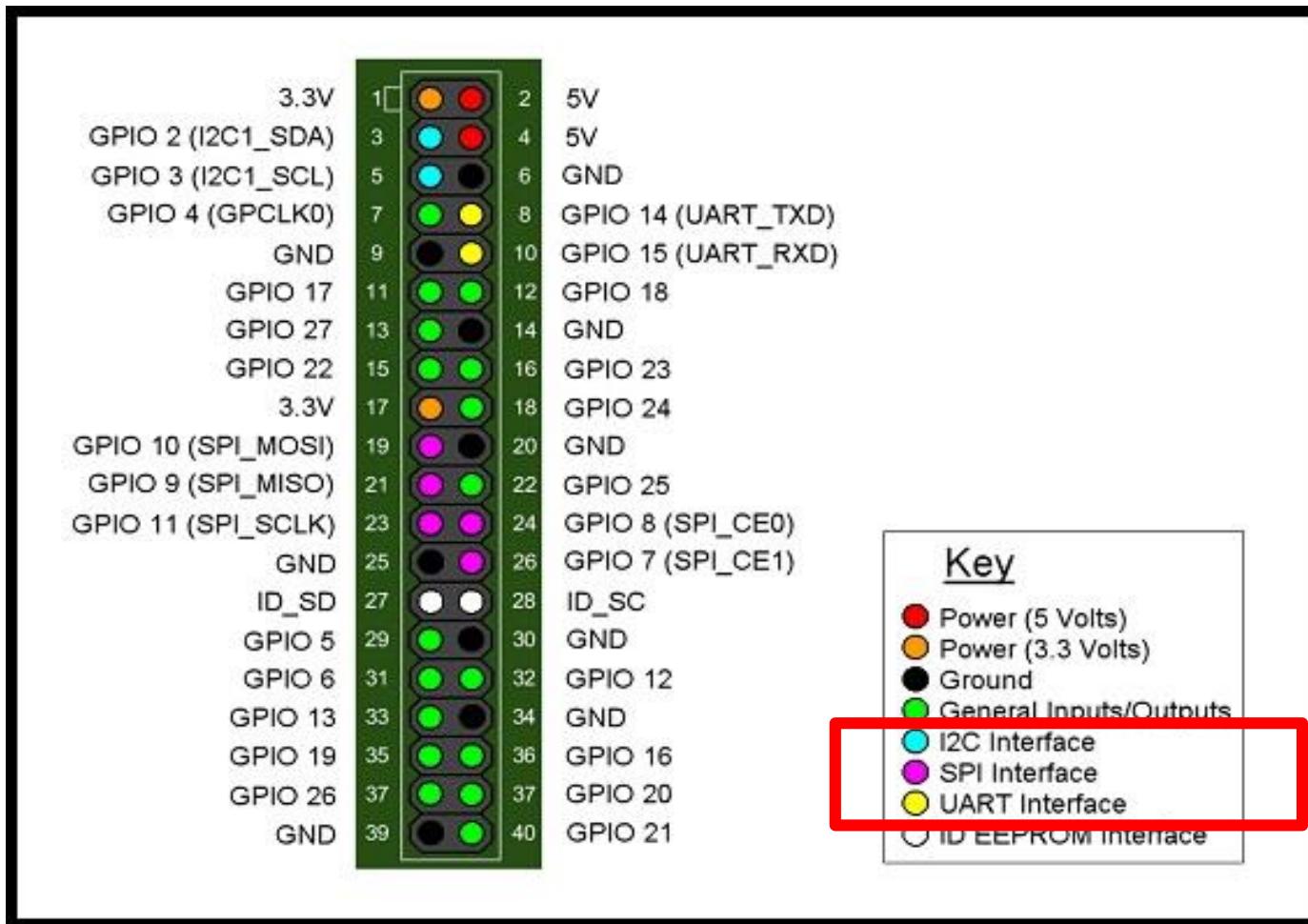


1. **특별한 통신 interface를 제공하는 GPIO**
 - UART (Universal Asynchronous Receiver Transmitter)
 - SPI (Serial Peripheral Interface)
 - I2C (Inter-Integrated Circuit)
2. **센서제어**
 - 거리측정 - 초음파센서
 - 동작감지 – 적외선센서
3. **소켓프로그램을 이용한 라즈베리파이 원격제어**

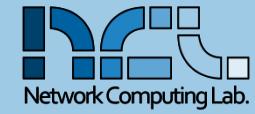
라즈베리파이 GPIO – 특별 통신 interface



GPIO (General Purpose Input / Output)



그림출처 : <http://www.hardcopyworld.com/ngine/arduino/wp-content/uploads/sites/3/2015/04/53bc258dc6c0425cb44870b50ab30621.jpg>



UART

UART(Universal Asynchronous Receiver/Transmitter)



적용예: PC와 프린터 연결



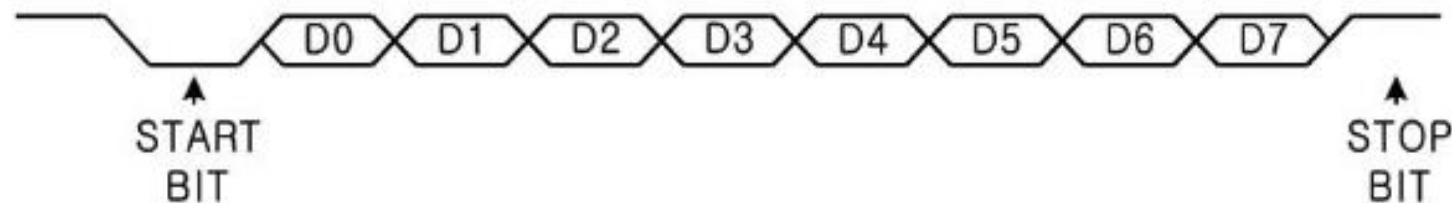
3 Wire Type Null Modem Cable

TX/RX line crosslinked

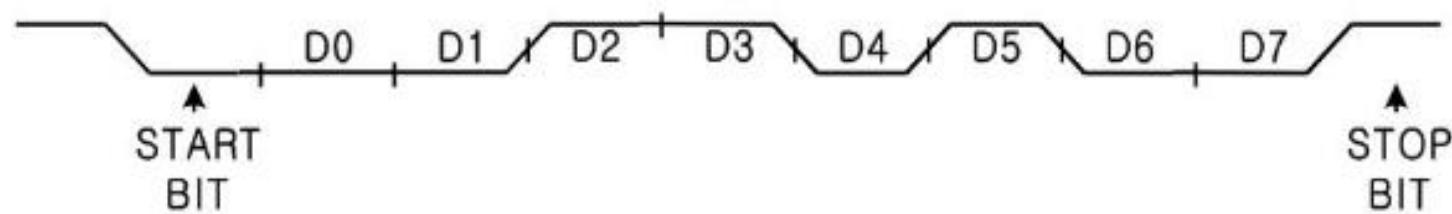
UART



- Serial 통신은 하나의 신호선을 이용해서 데이터를 비트 단위로 보내는 방식
“예” 8비트 UART(Universal Asynchronous Receiver Transmitter)



데이터 : 2CH = 00101100B



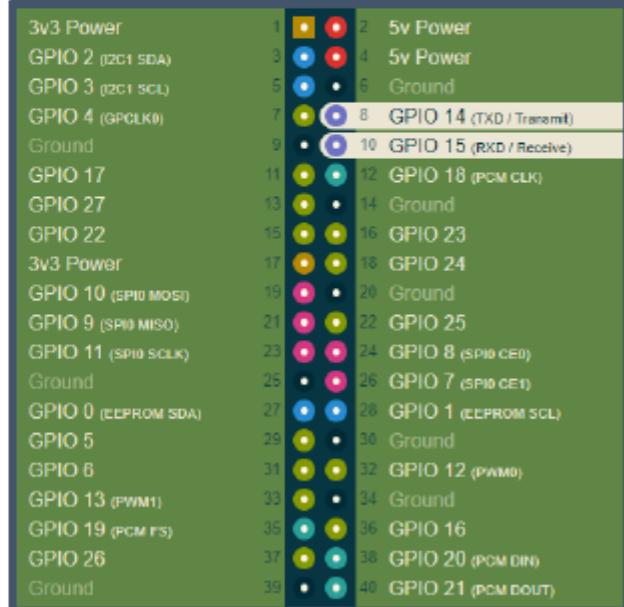
UART

Raspberry Pi 에서의 UART

- 버전 별로 다름

- ~3 : 핀 14, 15번 (miniUART 도 같은 핀 사용)
- 4 : 추가로 0~15번 사용 가능

```
root@raspberrypi:~/src # raspi-gpio funcs
GPIO, DEFAULT PULL, ALT0, ALT1, ALT2, ALT3, ALT4, ALT5
0, UP, SDAO, SA5, PCLK, SPI3_CE0_N, TXD2, SDA6
1, UP, SCL0, SA4, DE, SPI3_MISO, RXD2, SCL6
2, UP, SDA1, SA3, LCD_VSYNC, SPI3_MOSI, CTS2, SDA3
3, UP, SCL1, SA2, LCD_HSYNC, SPI3_SCLK, RTS2, SCL3
4, UP, GPCLK0, SA1, DPI_D0, SPI4_CE0_N, TXD3, SDA3
5, UP, GPCLK1, SA0, DPI_D1, SPI4_MISO, RXD3, SCL3
6, UP, GPCLK2, SOE_N_SE, DPI_D2, SPI4_MOSI, CTS3, SDA4
7, UP, SPI0_CE1_N, SWE_N_SRW_N, DPI_D3, SPI4_SCLK, RTS3, SCL4
8, UP, SPI0_CE0_N, SDG, DPI_D4, I2CSL_CE_N, TXD4, SDA4
9, DOWN, SPI0_MISO, SD1, DPI_D5, I2CSL_SDI_MISO, RXD4, SCL4
10, DOWN, SPI0_MOSI, SD2, DPI_D6, I2CSL_SDA_MOSI, CTS4, SDA5
11, DOWN, SPI0_SCLK, SD3, DPI_D7, I2CSL_SCL_SCLK, RTS4, SCL5
12, DOWN, PWM0_0, SD4, DPI_D8, SPI5_CE0_N, TXD5, SDA5
13, DOWN, PWM0_1, SD5, DPI_D9, SPI5_MISO, RXD5, SCL5
14, DOWN, TXD0, SD6, DPI_D10, SPI5_MOSI, CTS5, TXD1
15, DOWN, RXD0, SD7, DPI_D11, SPI5_SCLK, RTS5, RXD1
16, DOWN, -, SD8, DPI_D12, CTS0, SPI1_CE2_N, CTS1
17, DOWN, -, SD9, DPI_D13, RTS0, SPI1_CE1_N, RTS1
18, DOWN, PCM_CLK, SD10, DPI_D14, SPI6_CE0_N, SPI1_CE0_N, PWM0_0
```



2~5는 RPi 4에서 추가됨 (1은 miniUART)

UART

Raspberry Pi 버전 별 UART 할당

Model	first PL011 (UART0)	mini UART
Raspberry Pi Zero	primary	secondary
Raspberry Pi Zero W	secondary (Bluetooth)	primary
Raspberry Pi 1	primary	secondary
Raspberry Pi 2	primary	secondary
Raspberry Pi 3	secondary (Bluetooth)	primary
Raspberry Pi 4	secondary (Bluetooth)	primary

Linux device	Description
/dev/ttys0	mini UART
/dev/ttyAMA0	first PL011 (UART0)
/dev/serial0	primary UART
/dev/serial1	secondary UART

Name	Type
UART0	PL011
UART1	mini UART
UART2	PL011
UART3	PL011
UART4	PL011
UART5	PL011

- miniUART는 기본적으로 비활성화
- Raspberry Pi 4 (블루투스 + UART2 ~ 5)
 - UART2 ~ 5 : 활성화 시 사용 가능
(기본적으로 비활성화)

※ 물리 UART는 /dev/ttyAMA?, 소프트웨어 UART는 /dev/ttys?로 표시됨

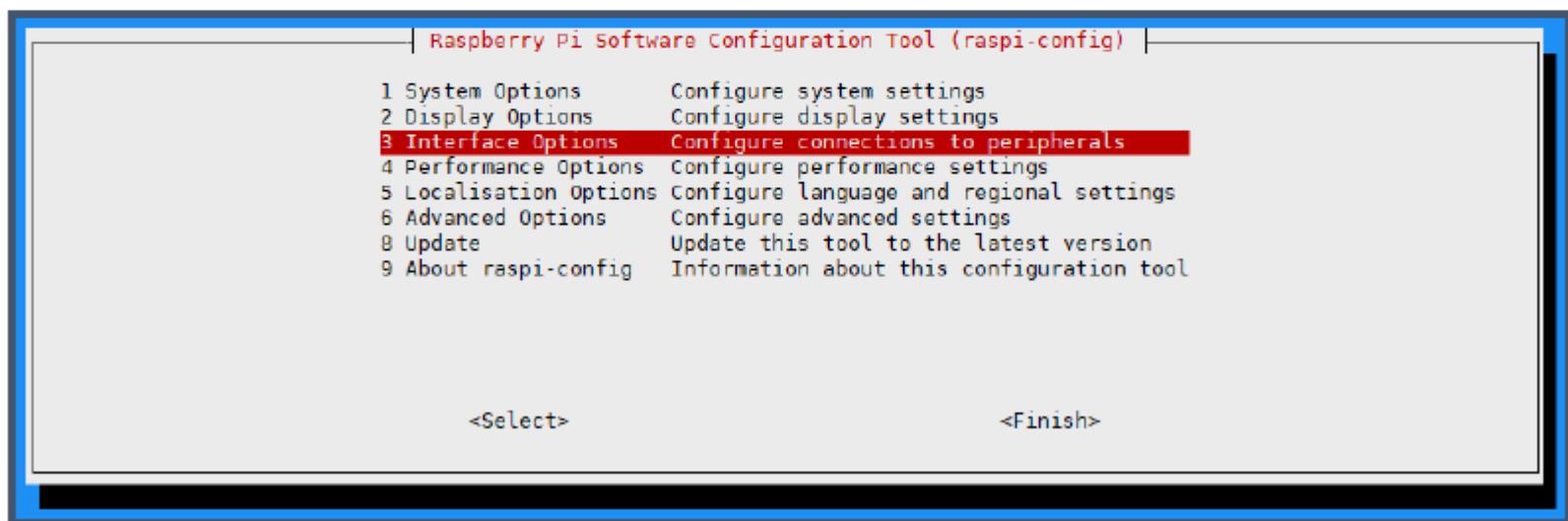
UART



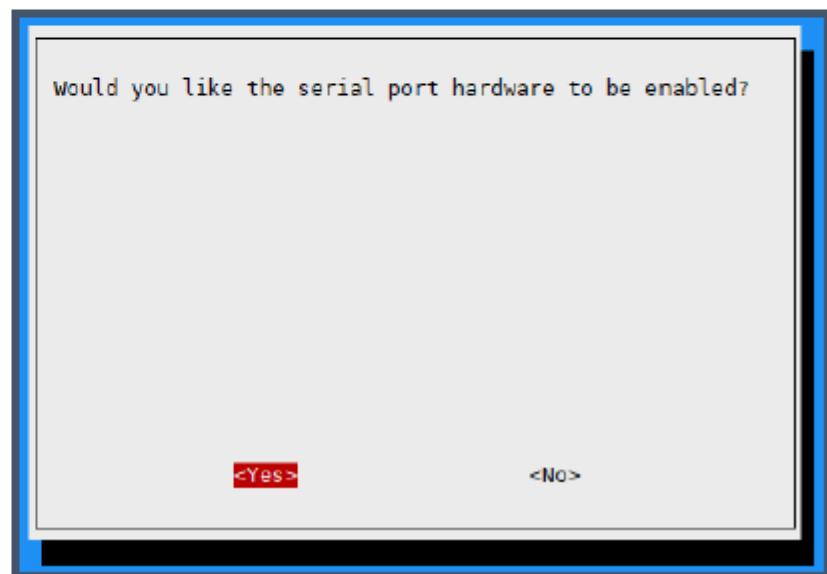
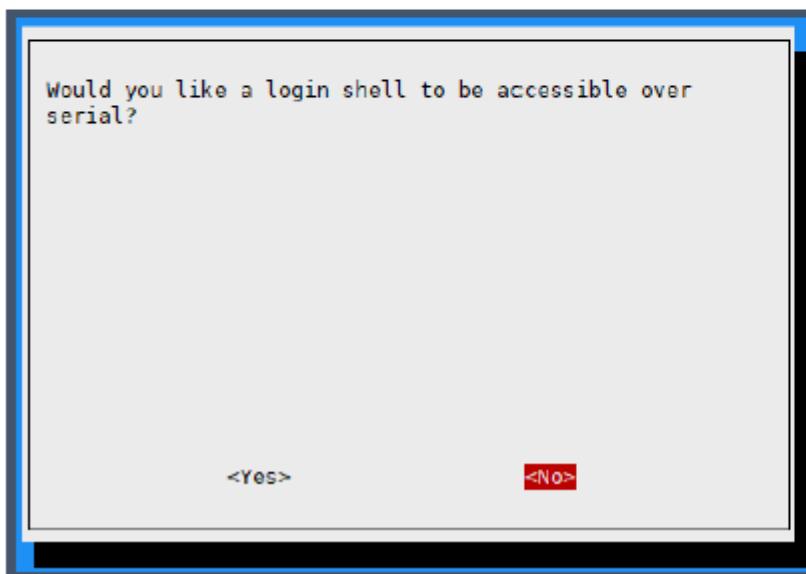
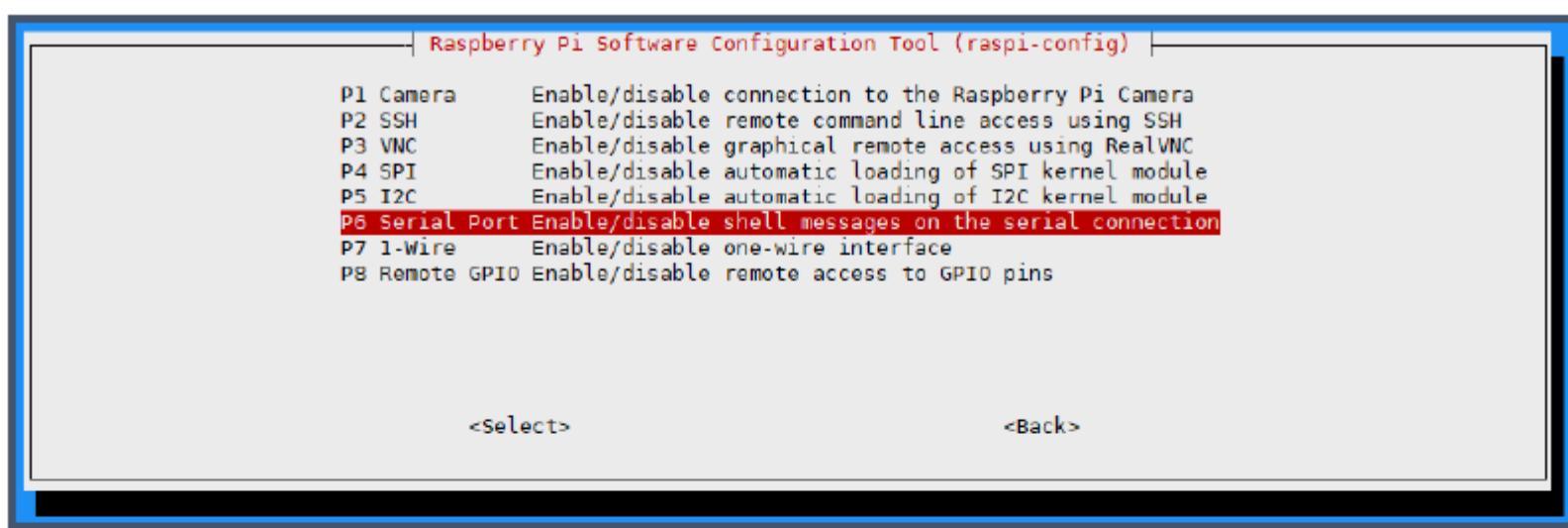
■ Software UART 활성화

raspi-config

- 3 Interface Options → P6 Serial Port → No → Yes



UART



UART



■ 다른 방법

- /boot/config.txt 설정
 - 파일의 마지막에 다음 내용 추가

```
enable_uart=1  
force_turbo=1 ← 이건 선택
```

- 수정 후 리부팅

UART

- Software UART 확인
 - /dev/ttys0 이 생성됨
 - /dev/serial0 이 ttys0에 연결됨

```
root@raspberrypi:~# dmesg | grep serial
[    0.000000] serial0  -> ttys0
[    0.000000] serial1  -> ttysAMA0
```

```
tty59  tty9
tty6   ttyAMA0
tty60  ttvprintk
tty61  ttys0
tty62  unid
tty63  uinput
tty7   urandom
tty8   v4l
```

- Minicom으로 확인
 - 설치 : sudo apt install minicom
 - 실행 : minicom -D /dev/ttys0 -b 9600
 - PIN 14와 15를 연결
 - 타이핑 했을 때 화면에 글씨가 출력되는지 확인
 - 종료 : Ctrl + A, Z, X

UART



■ Raspberry 3, 4의 UART0 사용

- 기본적으로 블루투스가 사용 중 → 블루투스를 비활성화 해야 함
- /boot/config.txt 에 추가

```
enable_uart=1  
dtoverlay=miniuart-bt  
core_freq=250 or force_turbo=1
```

- 추가한 후 리부팅
- 확인
 - /dev 에서 확인

```
*1 rpi@raspberrypi:~$  
41 serial0 -> ttyAMA0  
41 serial1 -> ttyS0  
19 shm
```

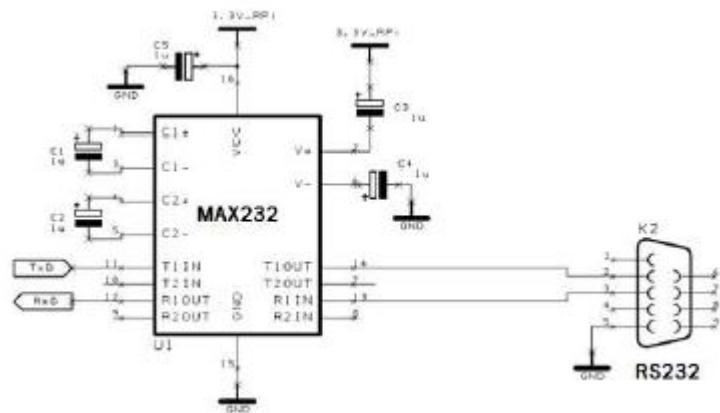
UART0 → serial 0

- minicom -D /dev/serial0 으로 확인

Software UART → serial 1

PC와 라즈베리파이 UART 연결

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.



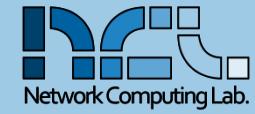
최근 노트북에서는 serial port를 지원하지 않는다. 아래와 같은 usb to serial adaptor 사용한다. 오른쪽 usb port는 노트북에 연결, 왼쪽 4개 라인은 라즈베리파이 GPIO에 연결



UART



- UART 관련 구현 사례는 참고문헌[1]의 P213 ~ P240을 보세요.



SPI

SPI(Serial Peripheral Interface)



라즈베리파이에는 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.

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.

SPI Master Slave간 통신



- analog output 센서의 output을 AD converter에 연결하여 digital output으로 변환. AD converter가 SPI 지원하는 경우 라즈베리파이와 AD converter는 SPI 통신방식으로 data를 주고 받음. 라즈베리파이는 SPI Master, AD converter는 SPI Slave로 동작

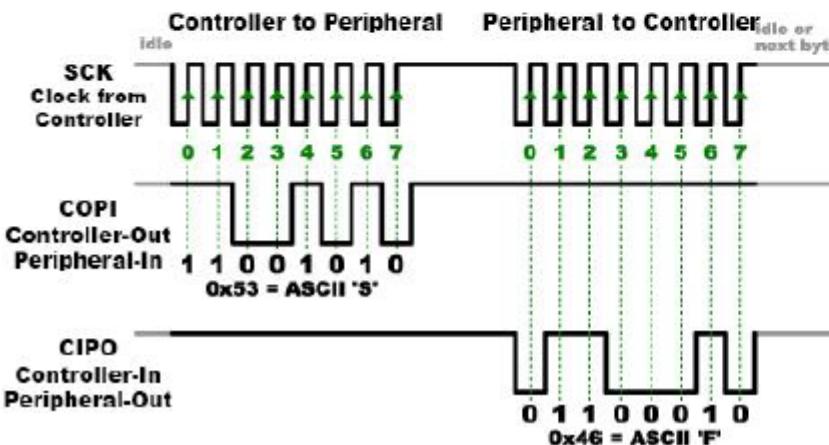
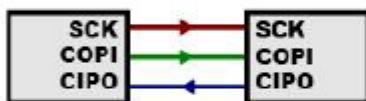


SPI 통신방식



SPI (Serial Peripheral Interface)

- 1:N 통신을 지원하는 동기식 통신
- 1개의 master와 N개의 slave 지원
- Rising or falling edge에서 데이터 읽고 쓸
- Full duplex 통신 가능

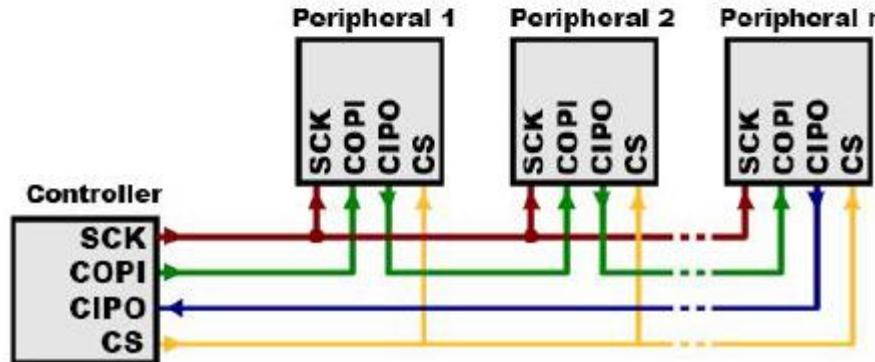
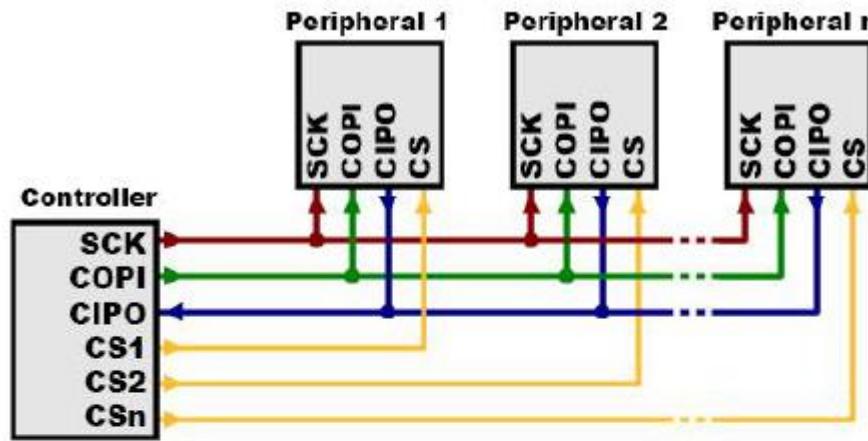


SCK	동기화 clock 신호
MOSI (COPI)	Master Out, Slave In. 마스터에서 슬레이브로 전송
MISO (CIPO)	Master In, Slave Out. 슬레이브에서 마스터로 전송
SS (CS)	Slave select (Chip select)

SPI



여러 slave 지원



SPI



Raspberry Pi SPI

- 3개의 SPI BUS 지원 (4는 7개)
- 사용 핀

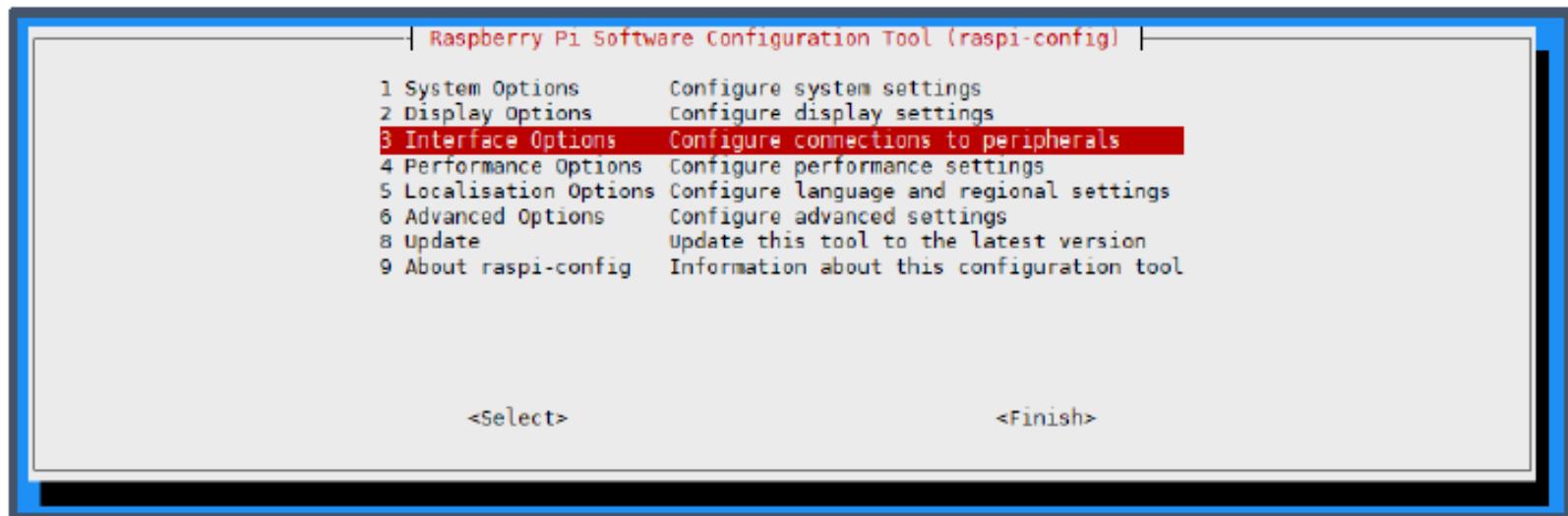
PIN	이름	설명
10	MOSI	Master → Slave
9	MISO	Slave → Master
11	SCLK	SPI clock
8	CE0	Chip enable 0
7	CE1	Chip enable 1



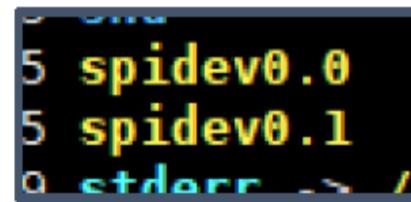
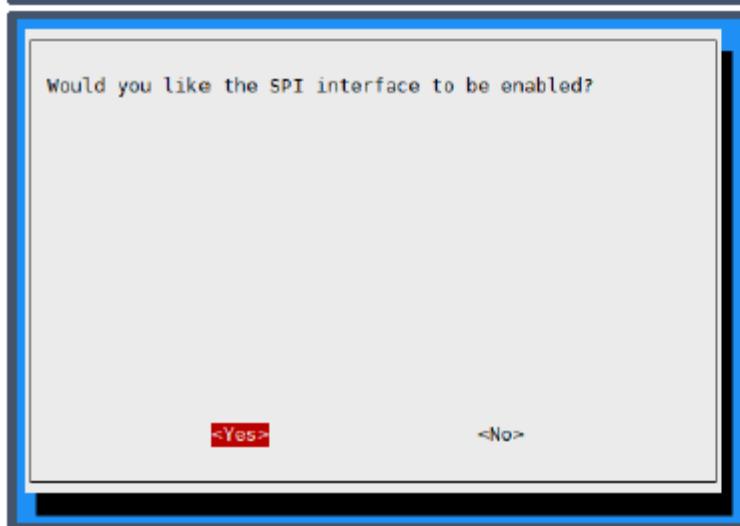
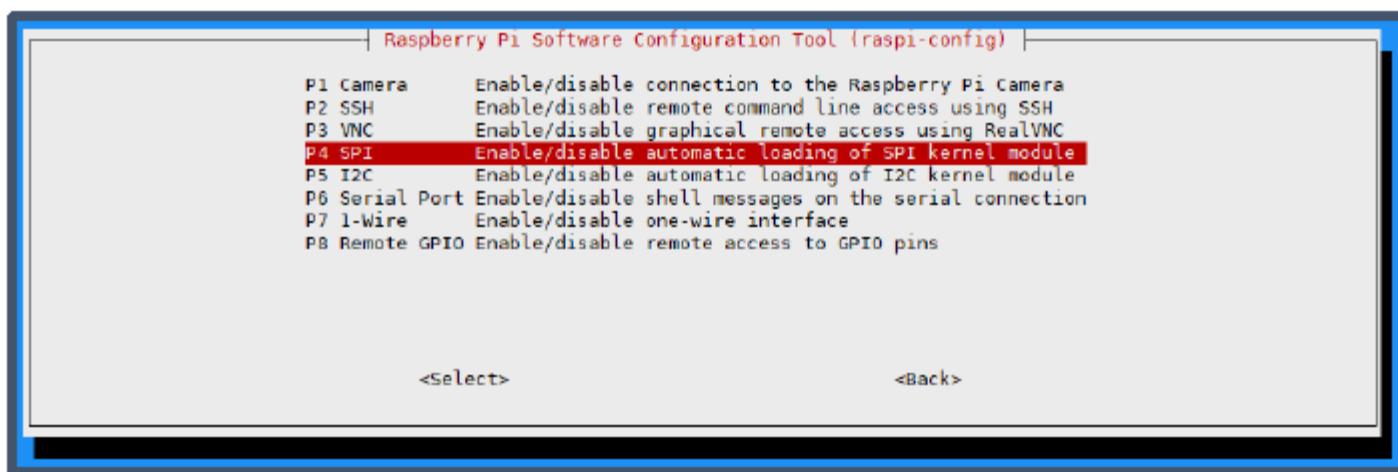
SPI 활성화

raspi-config

- Interface Options → SPI → Yes
- 설정 종료 후 리부팅



SPI



설정 완료 후 /dev/spidev0.0 확인

SPI 테스트

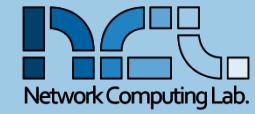
- 테스트용 프로그램 실행
- 9 (MISO), 10번 (MOSI) 핀 연결

```
wget https://raw.githubusercontent.com/raspberrypi/linux/rpi-3.10.y/Documentation/spi/spidev_test.c
```

```
root@raspberrypi:~/src # ./spidev_test
spi mode: 4
bits per word: 8
max speed: 500000 Hz (500 KHz)

FF FF FF FF FF FF
40 00 00 00 00 95
FF FF FF FF FF FF
FF FF FF FF FF FF
FF FF FF FF FF FF
DE AD BE EF BA AD
F0 0D
```

- SPI 관련 아날로그 센서별 구현 사례는 참고문헌[1]의 P275 ~ P306을 보세요.



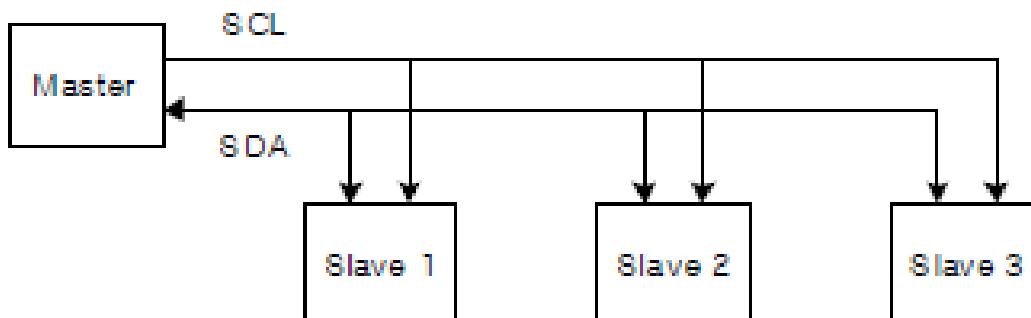
I2C

I²C(Inter-Integrated Circuit)

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.

I²C(Inter integrated circuit) 인터페이스는 IC들 간의 데이터 통신을 목적으로 하는 통신방식으로 2개 신호선을 사용하여 통신톤록 정의하고 있다. 이를 신호선은 SDA(Serial data)와 SCL(Serial Clock)이다. 이 통신 방식은 회로가 간단하고, 비용이 저렴하며, 통신과정에서 잡음이 적은 등의 장점을 가진다.

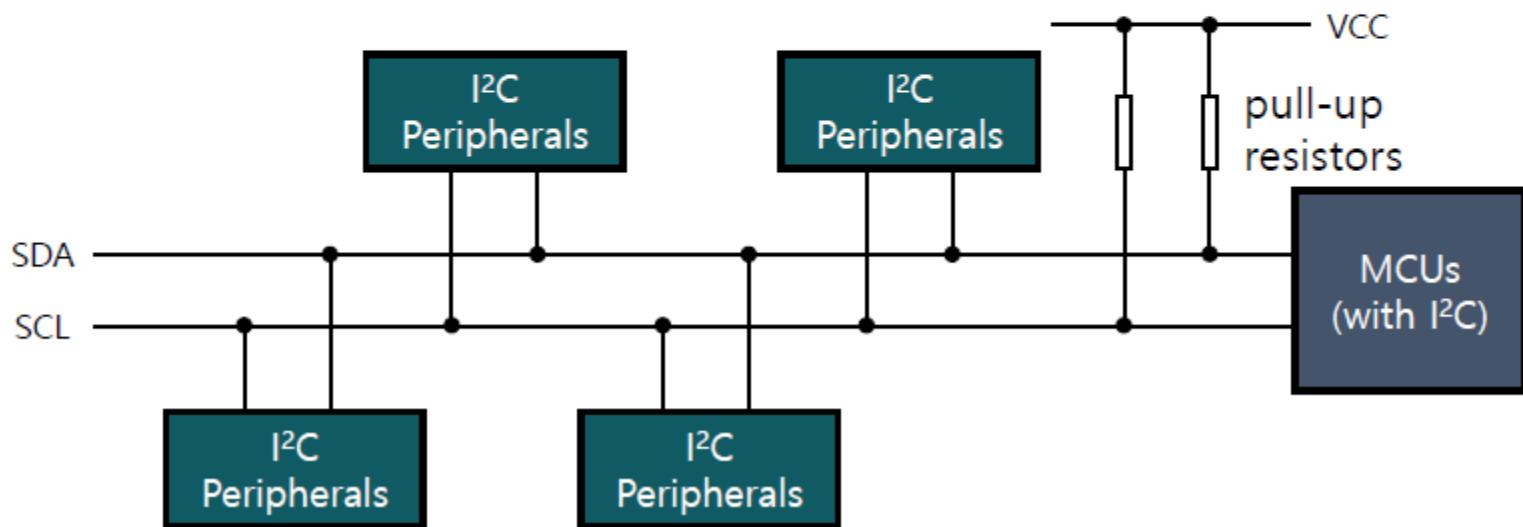
I²C 통신을 위해 다음 그림과 같이 송/수신측은 하나의 마스터(master)와 다수의 슬레이브(slave)로 구성될 수 있다. 마스터가 어느 슬레이브와 통신을 할 것인지를 위해 각 슬레이브에는 주소가 부여된다.



I²C

■ I²C (Inter-Integrated Circuit)

- 필립스에서 개발한 직렬 버스
- 비교적 저속의 주변기기 연결에 사용
- 데이터(SDA)와 클럭(SCL)의 두 라인 사용 (여러 기기 부착 가능)
- 여러 개의 master와 slave가 존재할 수 있음

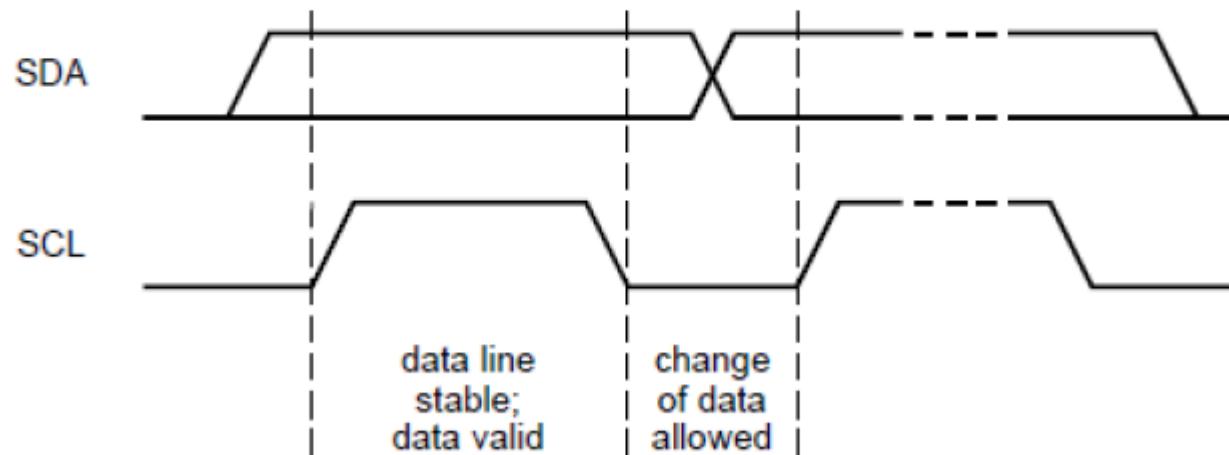


I²C



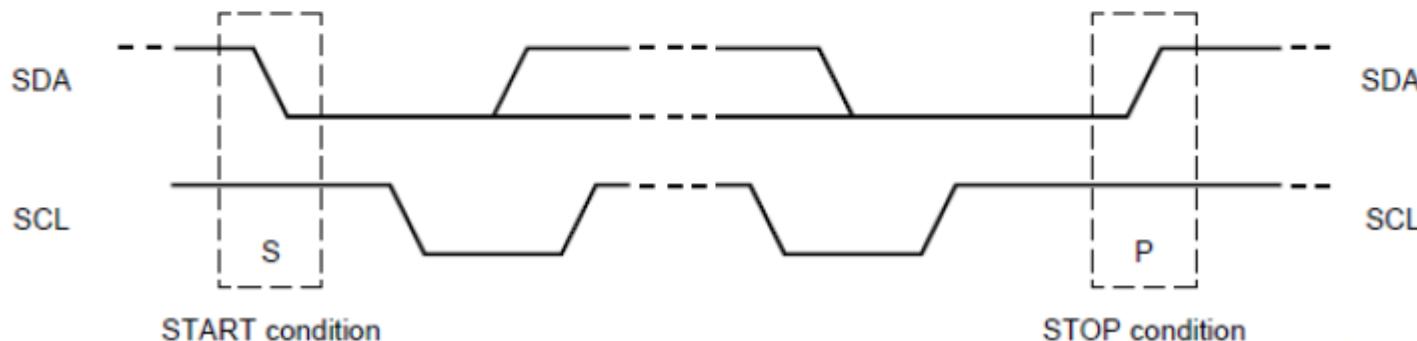
■ I²C 데이터 전송

- SCL 클럭 1펄스마다 1비트 전송



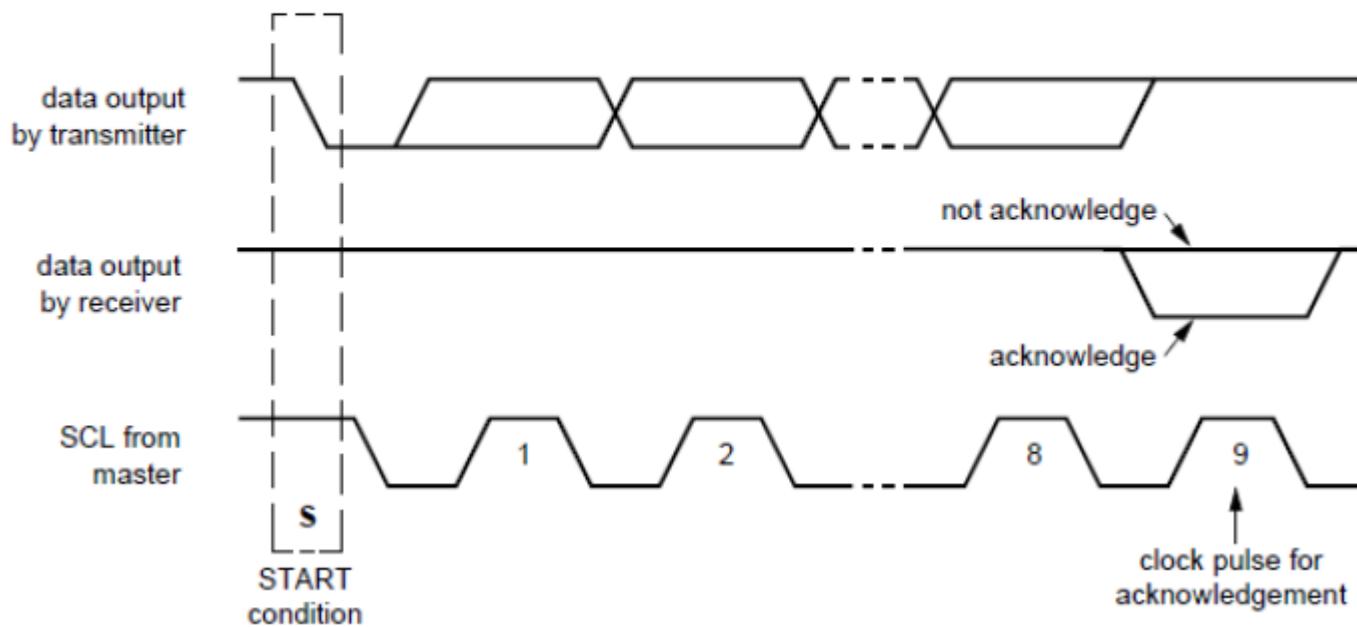
I2C

- 전송 시작과 끝
 - 기본 상태 : SDA, SCL 모두 HIGH (pull-up resistor에 의해)
 - 전송 시작(S)
 - SCL이 HIGH인 상태에서 SDA가 HIGH → LOW로 변화 (falling edge)
 - 전송 종료(P)
 - SCL이 HIGH인 상태에서 SDA가 LOW → HIGH로 변화 (rising edge)



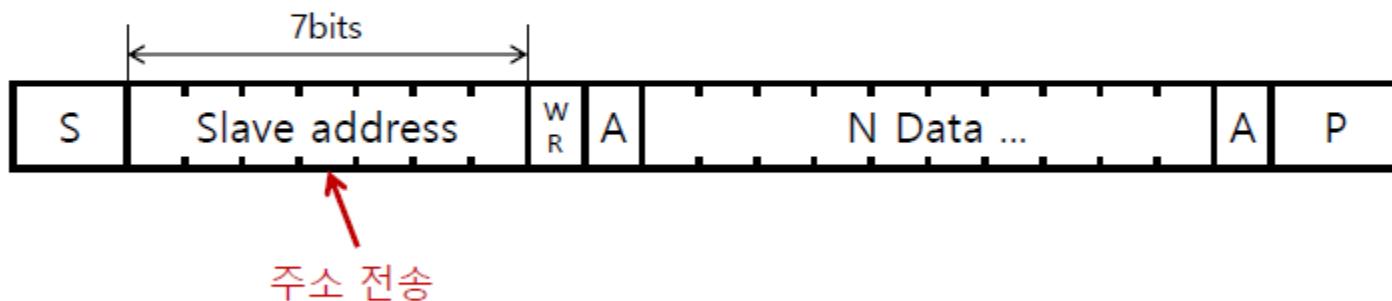
I2C

- 전송 확인 (Acknowledgement)
 - 1바이트 수신 시마다 acknowledge 전송
 - 전송 성공 → LOW, 전송 실패 → HIGH



I2C

- 주소 전달
 - 7bits 주소
 - 전송 시작 시 언제나 slave address 7bits + Read/Write 1bit 전송



- 주소는 기기마다 정해져 있음 → 같은 주소는 충돌 가능

I²C



■ Raspberry Pi 의 I²C

● 사용핀

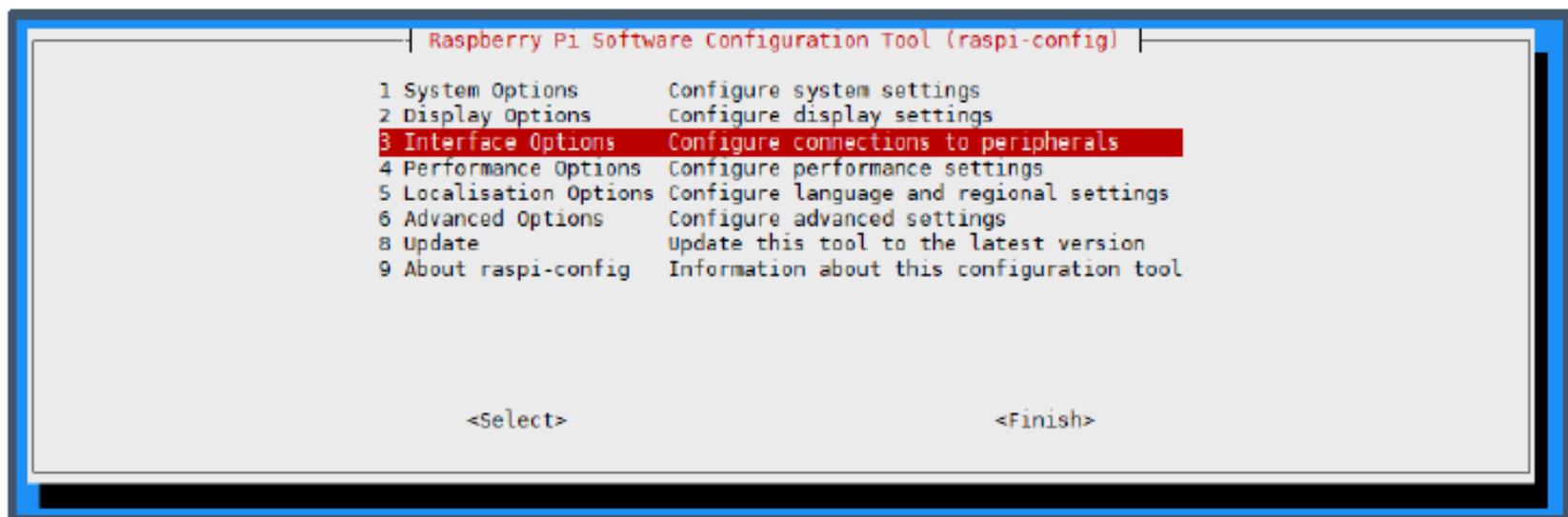
PIN	이름	설명
2	SDA	데이터 전송용
3	SCL	clock 전송용

3v3 Power	1	2	5v Power
GPIO 2 (data)	3	4	5v Power
GPIO 3 (clock)	5	6	Ground
GPIO 4 (GPCLK0)	7	8	GPIO 14 (UART TX)
Ground	9	10	GPIO 15 (UART RX)
GPIO 17	11	12	GPIO 18 (PCM CLK)
GPIO 27	13	14	Ground
GPIO 22	15	16	GPIO 23
3v3 Power	17	18	GPIO 24
GPIO 10 (SPI0 MOSI)	19	20	Ground
GPIO 9 (SPI0 MISO)	21	22	GPIO 25
GPIO 11 (SPI0 SCLK)	23	24	GPIO 8 (SPI0 CE0)
Ground	25	26	GPIO 7 (SPI0 CE1)
GPIO 0 (EEPROM Data)	27	28	GPIO 1 (EEPROM Clock)
GPIO 5	29	30	Ground
GPIO 6	31	32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33	34	Ground
GPIO 19 (PCM FS)	35	36	GPIO 16
GPIO 26	37	38	GPIO 20 (PCM DIN)
Ground	39	40	GPIO 21 (PCM DOUT)

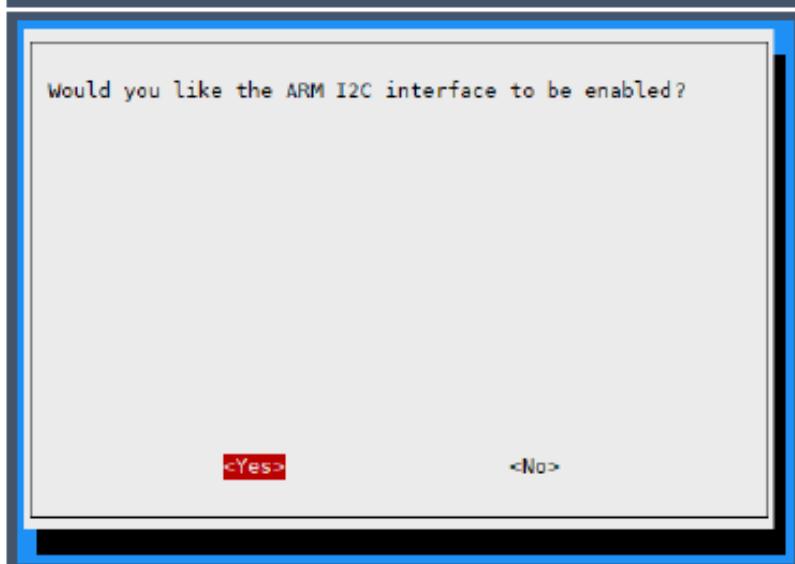
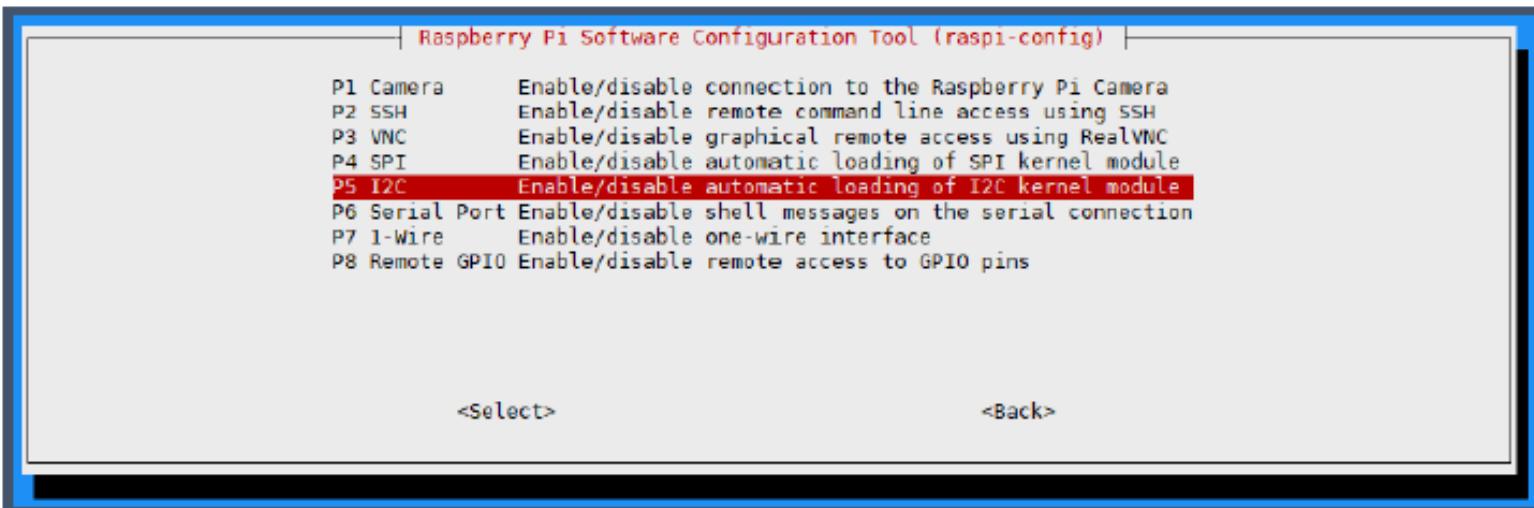
■ 활성화

- Interface Options → I2C → Yes
- 설정 종료 후 리부팅

raspi-config



I2C



```
FO nwwng
!8 i2c-1
!q initctl
```

설정 완료 후 /dev/i2c-1 확인

I2C



- I2C 관련 사용 예제는 참고문헌[1]의 P253 ~ 274를 보세요.

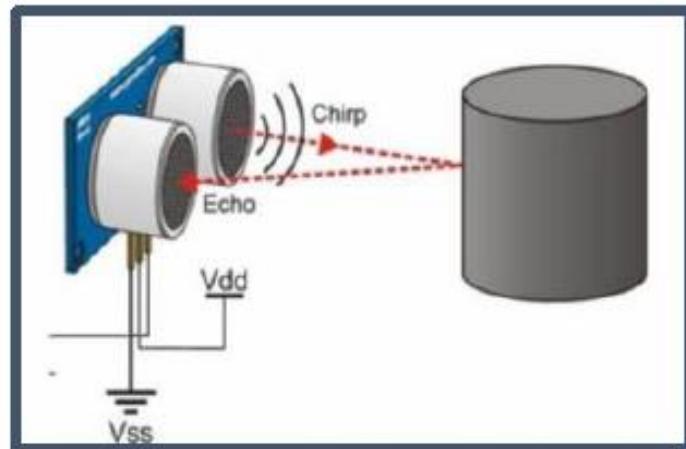
라즈베리파이 센서 제어

거리측정 - 초음파센서



■ 초음파 센서 HC-SR04

- 초음파를 이용하여 거리 측정
- 동작 원리
 - 1. 초음파 발생
 - 2. 되돌아 오는 초음파 감지
 - 3. 시간 측정
 - 4. 음속을 이용하여 거리 계산

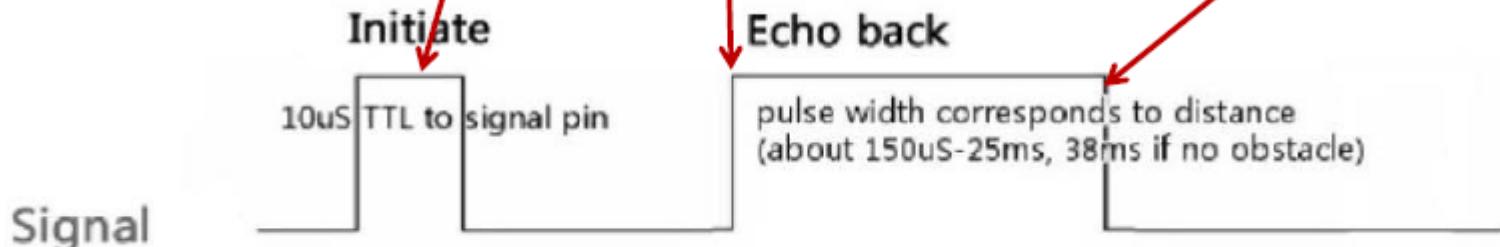


거리측정 - 초음파센서



① Trigger에 약 10uS동안 HIGH 신호를 넣음

- 센서 동작



Formula:

$$\text{pulse width (uS)} / 58 = \text{distance (cm)}$$
$$\text{pulse width (uS)} / 148 = \text{distance (inch)}$$

Internal

Ultrasonic Transducer will issue 8 40kHz pulse

② 초음파 발생 시작 (8번)

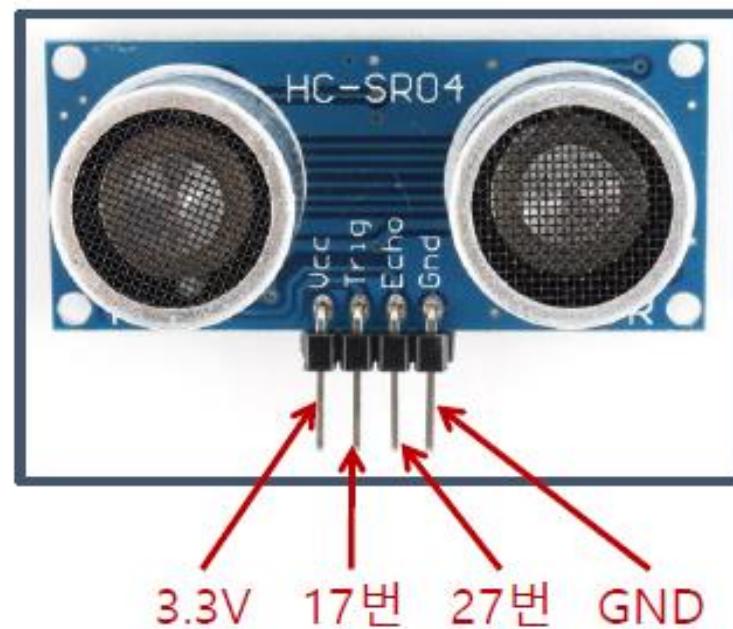
※ echo신호의 길이(uS)로 거리 측정

$$\text{distance} = \text{pulse width} * \text{声波速度} (340\text{m/s}) / 2$$

$$\text{distance (cm)} = \text{pulse width} / 58$$

거리측정 - 초음파센서

- 센서 연결



거리측정 – 초음파센서(ultrasonic.c)

모드 설정 : trig → OUTPUT, echo → INPUT 거리 측정 (초음파 센서)

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

int main()
{
    int trig = 17;
    int echo = 27;
    unsigned int start, end;
    float distance;

    gpioInitialise();

    gpioSetMode(trig, PI_OUTPUT);
    gpioSetMode(echo, PI_INPUT);
    gpioWrite(trig, 0);
    gpioDelay(100000);
```

trig는 0으로 시작

```
while(1) {
    gpioWrite(trig, 1);
    gpioDelay(10);
    gpioWrite(trig, 0);
    while(gpioRead(echo) == 0)
        start = gpioTick();
```

trig 신호 입력 시작

10us 후에 0 입력

echo가 1이 되면 시간기록

```
    while(gpioRead(echo) == 1)
        end = gpioTick();
```

echo가 0이 되면 시간기록

```
    distance = (end - start) / 58.0f;
    printf("diff: %u, distance (cm) : %f\n", \
           end - start, distance);
```

gpioWrite(trig, 0); pulse 길이로 거리 계산

```
    gpioDelay(100000);
}
```

```
gpioTerminate();
return 0;
```

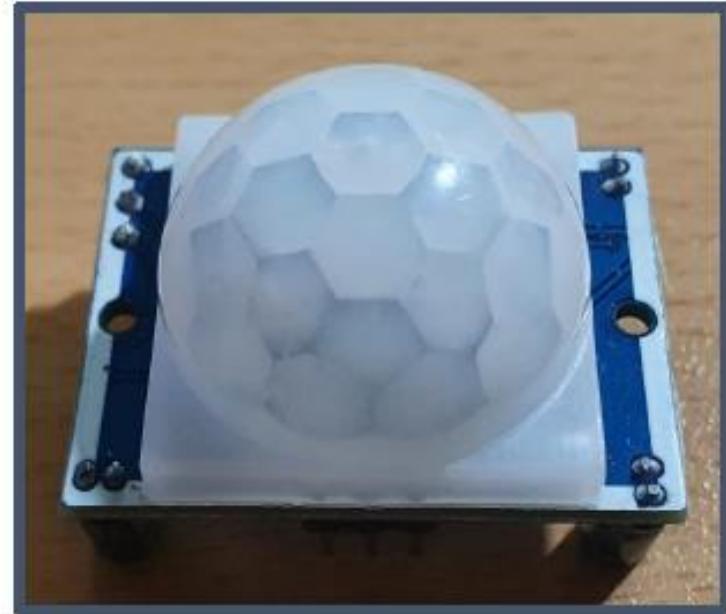
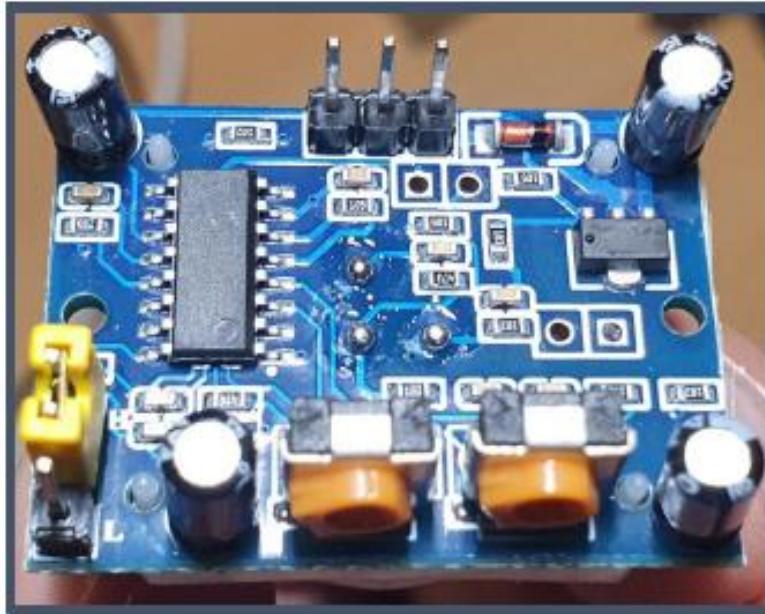
실습 1. 거리측정

1. HC-SR04 초음파센서를 브래드보드에 설치한다. 센서의 3핀을 3.3 V, GPIO 17번, GPIO 27번, GND에 각각 연결한다.
2. ultrasonic.c를 컴파일하여 수행파일을 만든다.
3. 초음파센서 앞 약 10cm 전방에 손을 두고, 위 수행파일을 수행한다.
측정 거리를 확인한다.
4. 초음파센서 앞 약 30cm 전방에 손을 두고, 위 수행파일을 수행한다.
측정 거리를 확인한다.

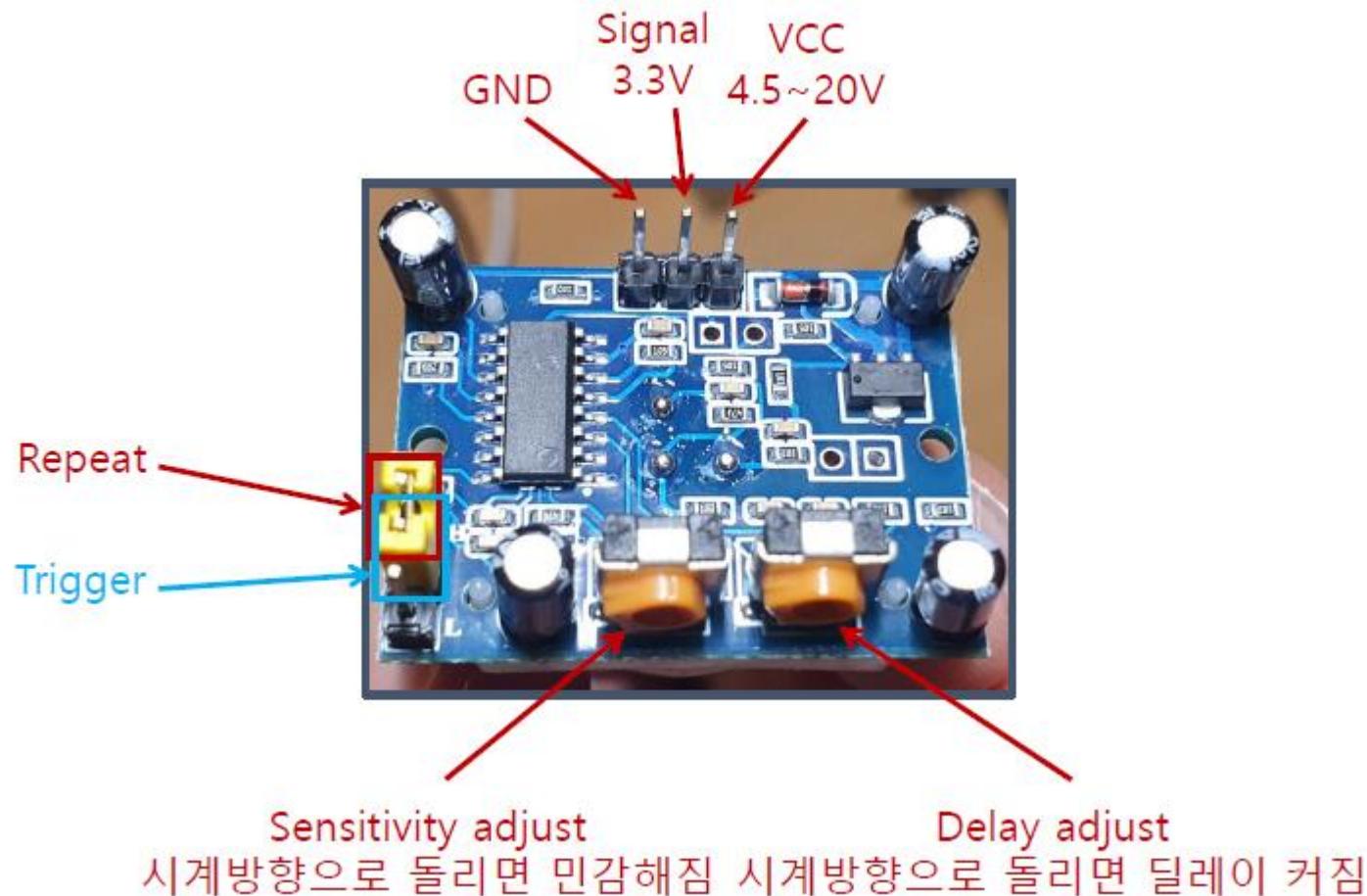
동작감지 – 적외선센서



- 동작 감지 센서 (PIR, Passive Infrared Sensor)
 - 인체에서 발생하는 적외선을 감지



동작감지 – 적외선센서



동작감지 – 적외선센서(pir.c)



Signal 핀을 17번에 연결

값을 읽음
상태 변화 체크
 $0 \rightarrow 1$
상태 변화 체크
 $1 \rightarrow 0$

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

int main()
{
    int pin = 17;
    int state = 0;
    int value;

    gpioInitialise();

    gpioSetMode(pin, PI_INPUT);

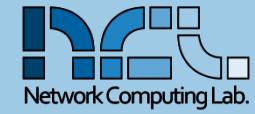
    while(1) {
        value = gpioRead(pin);
        if(value == 1 && state == 0) {
            state = 1;
            printf("on\n");
        }
        else if(value == 0 && state == 1) {
            state = 0;
            printf("off\n");
        }
        gpioDelay(1000);
    }

    gpioTerminate();

    return 0;
}
```

실습 2. 동작감지

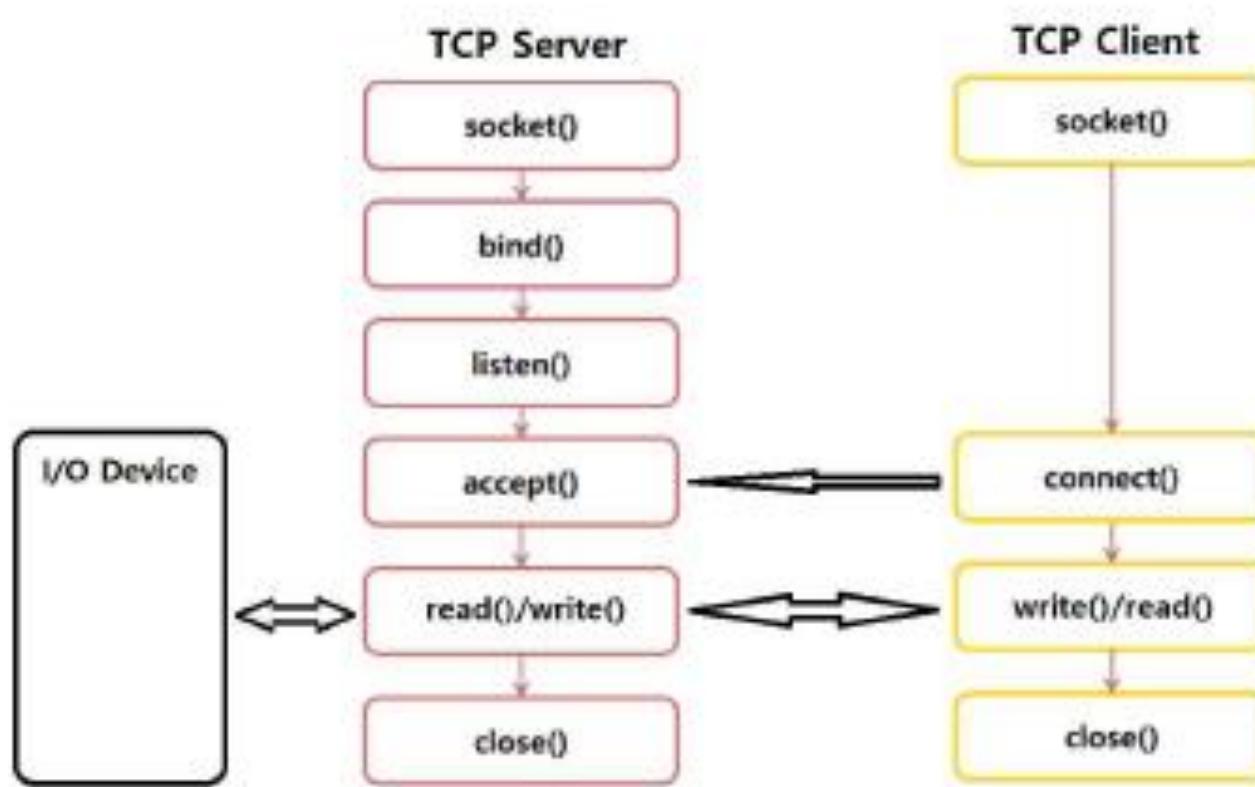
1. 동작감지용 적외선 센서를 브래드보드에 설치한다. 센서의 3개 핀을 GND, GPIO 17번, Vcc에 각각 연결한다.
2. pir.c를 컴파일하여 수행파일을 만든후 수행한다.
3. 적외선 센서 앞 약 10cm 전방에서 손을 천천히 움직여 센서앞을 통과하게 한다. 센싱 결과를 확인한다.



소켓프로그램이용한 라즈베리파이 GPIO 원격제어

TCP 서버/클라이언트에 의한 디바이스 제어

- TCP 방식의 채팅 서버와 클라이언트 소스를 사용 + 디바이스 제어
- 서버에 연결된 디바이스를 제어
- 서버 측 소스만 디바이스 제어 코드 추가



TCP 채팅서버 + LED 제어 동작



- 아래는 다음페이지부터 소개되는 `tcpserver.c` 및 `tcpclient.c` 코드 동작에 대한 설명
- 라즈베리파이에서 서버 코드가 수행되고 우분투에서 클라이언트 코드가 수행
- 라즈베리파이 pin18에 연결된 LED를 제어
- 클라이언트 측으로부터 메시지의 첫 문자가
 - 0 이면 서버는 LED OFF
 - 1 이면 서버는 LED ON
 - q 이면 서버는 클라이언트 소켓 종료하고 다른 클라이언트의 연결요청을 대기
 - 클라이언트 입력 또는 서버로 온 메시지의 첫문자가 q이면 클라이언트 종료
- 서버는
 - 서버 터미널에서 입력된 메시지를 클라이언트에 전송.
 - 만일 그 입력이 q 이면 서버 종료.

TCP 채팅서버 + LED 제어(tcpserver.c-1)

```
1 //=====
2 // tcpSever_01_revised.c (Linux)
3 //      Chatting + controlling LED device
4 //=====
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <string.h>
8 #include <unistd.h>
9 #include <arpa/inet.h>
10 #include <sys/socket.h>
11
12 #include <pigpio.h>           // *)
13
14 #define PORTNUM      0x9090      // port#
15 #define BUFFSIZE     256
16
17 #define pin          18          // BCM_GPIO #18
18
19 int main(void) {
20     struct sockaddr_in serv_addr, cli_addr;
21     int serv_fd, cli_fd, clilen;
22     char buffer[BUFFSIZE];
23
24     printf("[TCP server for chatting and controlling LED...]\\n");
25
26     if(gpioInitialise() == -1)           //initialze GPIO
27         return 1;
28
29     gpioSetMode(pin, PI_OUTPUT);        //
30
31     // 1) create server socket
32     if((serv_fd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
33         perror("ERROR opening socket");
34         exit(1);
35     }
```

12번, 17번, 26~ 29번줄
gpio 제어를 위한 코드
추가부분

TCP 채팅서버 + LED 제어(tcpserver.c-2)



```
36
37     // 2) setting server socket structure
38     memset((char *) &serv_addr, 0x00, sizeof(serv_addr));
39     serv_addr.sin_family      = AF_INET;
40     serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
41     serv_addr.sin_port        = htons(PORTNUM);
42
43     // 3) bind()
44     if(bind(serv_fd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) == -1) {
45         perror("ERROR on binding");
46         exit(1);
47     }
48
49     // 4) listen()
50     listen(serv_fd, 5);
51
52 loop:
53
54     // 5) accept(), blocking...
55     clilen = sizeof(cli_addr);
56     if((cli_fd = accept(serv_fd, (struct sockaddr *)&cli_addr, &clilen)) == -1) {
57         perror("ERROR on accept");
58         exit(1);
59     }
60
61     // 6) read/write, write(), sending...
62     write(cli_fd, "Welcome to Chat Server.....LED control..", BUFFSIZE);
63
```

TCP 채팅서버 + LED 제어(tcpserver.c-3)



```
64     while(1) {
65         // 6) read/write, read(), receiving...
66         memset(buffer, 0x00, sizeof(buffer));
67         if((read(cli_fd, buffer, BUFFSIZE)) == -1){
68             perror("ERROR reading from socket");
69             exit(1);
70         }
71         printf("[Guest] %s\n", buffer);
72
73         if(buffer[0] == 'q') {
74             close(cli_fd);
75             goto loop;
76         }
77         else if(buffer[0] == '0')           // LED control....
78             gpioWrite(pin, 0);
79         else if(buffer[0] == '1')
80             gpioWrite(pin, 1);
81
82         // 6) read/write, write(), sending...
83         memset(buffer, 0x00, sizeof(buffer));
84         printf("[Server] ");
85         fgets(buffer, BUFFSIZE, stdin);
86         write(cli_fd, buffer, BUFFSIZE);
87         if(buffer[0] == 'q')
88             break;
89     }
90
91     // 7) close(), close server socket, disconnection
92     close(serv_fd);
93     gpioTerminate();
94
95     return 0;
96 }
```

77~80번, 93번줄
gpio 제어를 위한 코드
추가부분

TCP 채팅 클라이언트(tcpclient.c-1)



```
1 //=====
2 // tcpClientChat_revised.c (Linux)
3 //      for Chatting
4 //=====
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <string.h>
8 #include <unistd.h>
9 #include <arpa/inet.h>
10 #include <sys/socket.h>
11
12 #define PORTNUM    0x9090          //
13 #define BUFFSIZE   256
14
15 int main(int argc, char *argv[]) {
16     int sock_fd;
17     struct sockaddr_in serv_addr;
18     char buffer[BUFFSIZE];
19
20     printf("[TCP server for chatting and controlling LED...]\\n");
21
22     // 1) create client socket
23     if((sock_fd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
24         perror("ERROR opening socket");
25         exit(1);
26     }
27
28     // 2) setting server socket structure
29     serv_addr.sin_family      = AF_INET;
30     serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
31     serv_addr.sin_port        = htons(PORTNUM);
32
33     // 3) connect
34     if(connect(sock_fd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) == -1) {
35         perror("ERROR connecting");
36         exit(1);
37     }
38 }
```

TCP 채팅 클라이언트(tcpclient.c-2)

```
39         // 4) read/write, read(), receiving...
40         memset(buffer, 0x00, sizeof(buffer));
41         if(read(sock_fd, buffer, BUFFSIZE)== -1){
42             perror("ERROR reading from socket");
43             exit(1);
44         }
45         printf("[Server] %s\n", buffer);
46
47     while(1) {
48         memset(buffer, 0x00, sizeof(buffer));
49
50         printf("[Guest] ");
51         fgets(buffer, BUFFSIZE, stdin);
52     }
53         // 4) read/write, write(), sending...
54         if(write(sock_fd, buffer, strlen(buffer)) == -1) {
55             perror("ERROR writing to socket");
56             exit(1);
57         }
58         if(buffer[0] == 'q')
59             break;
60
61         // 4) read/write, read(), receiving...
62         memset(buffer, 0x00, sizeof(buffer));
63         if(read(sock_fd, buffer, BUFFSIZE) == -1) {
64             perror("ERROR reading from socket");
65             exit(1);
66         }
67
68         printf("[Server] %s\n", buffer);
69         if(buffer[0] == 'q')
70             break;
71     }
72
73     // 5) close(), close client socket, disconnection
74     close(sock_fd);
75
76     return 0;
77 }
```

실습 3-1 소켓프로그램이용한 LED 원격제어



- 라즈베리파이 GPIO 18번에 LED 연결, client에서 오는 메시지에 따라 라즈베리파이 server는 LED를 제어
- 우분투에서 tcpserver.c 및 tcpclient.c를 생성
- tcpserver.c를 scp 명령어를 이용하여 우분투에서 라즈베리파이로 복사
- tcpserver.c 및 tcpclient.c를 각각 컴파일 및 실행

- [라즈베리파이 prompt]# gcc -o tcpserver tcpserver.c
- [우분투 prompt]# gcc -o tcpclient tcpclient.c
- [라즈베리파이 prompt]# ./tcpserver // 서버실행, port #는 코드내 지정
- [우분투 prompt]# ./tcpclient 서버IP // 서버IP는 라즈베리파이IP를 의미, port #는 코드내 지정

- tcpclient에서 아래 입력이 들어오는 경우 tcpserver 수행 (그 이외 입력이 들어오는 경우 안내 메시지와 함께 재입력을 요청)
 - 0 // LED OFF
 - 1 // LED ON
 - q // 연결종료

- 서버/클라이언트간 write/read를 반복하여 구동

실습 3-2 소켓프로그램이용한 LED 원격제어



- 실습 3-1에서, tcpclient에서 오는 입력 내용중 LED blinking 옵션 추가
- tcpclient에서 아래 입력이 들어오는 경우 tcpserver 수행 (그 이외 입력이 들어오는 경우 안내 메시지와 함께 재입력을 요청)
 - 0 // LED OFF
 - 1 // LED ON
 - bn // n번만큼 blinking (LED ON, 1초후 LED OFF) 반복, n은 1~9 사이 숫자
 - q // 연결종료
- 서버/클라이언트간 write/read를 반복하여 구동

참고문헌



1. 임베디드시스템응용, 제 14장 소켓프로그램에 의한 원격제어

http://cms3.koreatech.ac.kr/sites/joo/IFC415/Raspberrypi_2019_09_03.pdf

다음 페이지를 보세요.

임베디드시스템응용 교재 소개



- 교재

http://cms3.koreatech.ac.kr/sites/joo/IFC415/Raspberrypi_2019_09_03.pdf

- 교재관련 추가로 내용을 보길 원하면

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

[IFC415] 임베디드응용및실습을 보세요.

- 교재

- 다양한 센서 응용에 대한 구현 사례 및 코드를 소개함
- 새롭게 개발할 경우 pigpio 라이브러리 사용하는 걸 추천하지만 교재내 wiringPi 라이브러리 사용한 예제 코드가 있는 경우 그걸 사용해도 무방함
- 다음 페이지에 관련 내용 목차 소개

임베디드시스템응용 교재 목차-1



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 스텝모터 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

임베디드시스템응용 교재 목차-1



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