

IoT 특론

7차시

AI첨단기술학과

이의혁

2. 사물 인터넷 디바이스

2-3. IoT 디바이스 프로그래밍

2) IoT 디바이스 프로그래밍

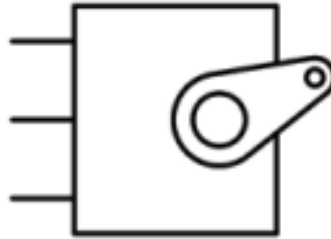
서보 모터 제어하기

- 서보 모터 (Servo Motor)
 - 특정 위치, 수치(속도, 토크)에 맞게 정확하게 제어할 수 있는 구조를 갖추고 있는 모터
 - 예를 들면, 지정된 각도만큼 회전하도록 제어 가능
 - 일정 방향으로 계속 회전하는 모터와는 다름 (예: 선풍기의 모터)
- 준비
 - Raspberry Pi와 GPIO 케이블로 연결된 코블러 브레이크아웃 보드와 브레드보드
 - SG90 서보 모터 1개
 - 점퍼 와이어

- SG90 micro servo

- 작동 전압: 4.8 – 7.2V
- 무게: 9g
- 동작 속도: 0.12초/60도 (4.8V)
- 회전 각도: 180도
- PWM (Pulse with modulation)신호에 의해 회전 각도 제어 가능

PWM=Orange (⌋⌋⌋)
Vcc = Red (+)
Ground=Brown (-)

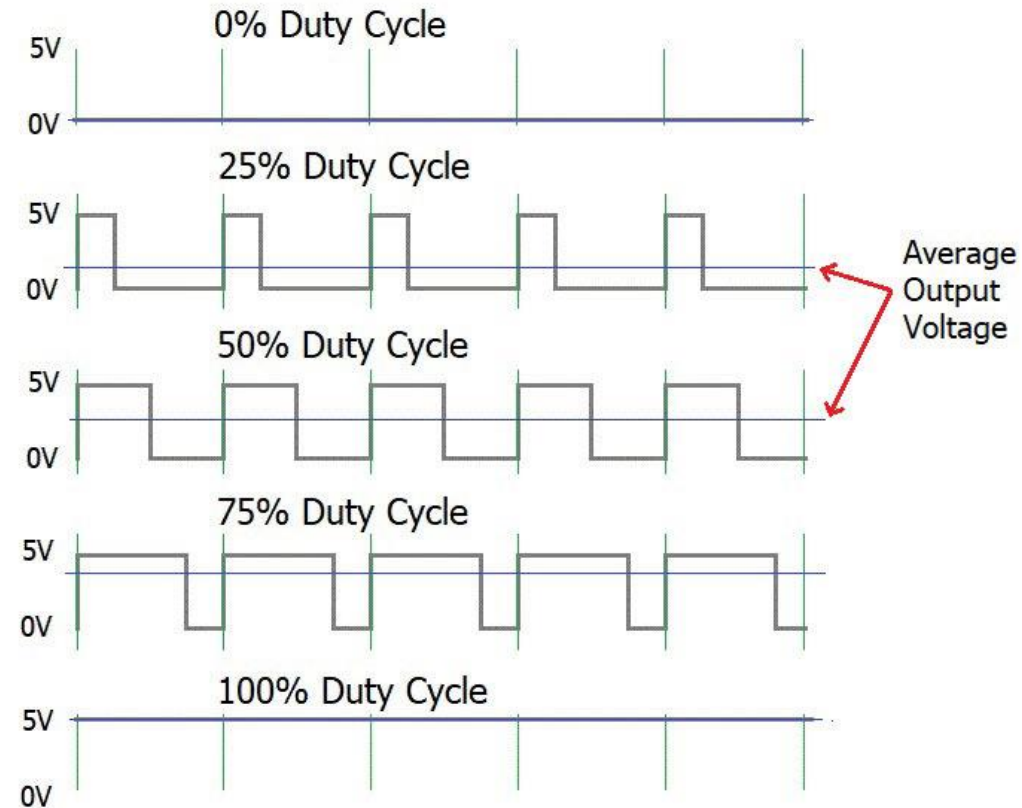


<https://content.instructables.com/ORIG/FA2/O1SS/J7ARLNBW/FA2O1SSJ7ARLNBW.pdf>

PWM: Pulse Width Modulation (펄스 폭 변조)

- Pulse Width Modulation

- 임의의 정해진 출력 파형(사각파 출력)을 생성하기 위해 펄스 폭이나 주파수 혹은 둘 모두를 변조시키는 제어 방식
- 출력되는 전압 값을 일정한 비율 (duty cycle) 동안 High로 유지하고, 나머지는 Low를 출력
- Duty cycle
 - 신호의 한 주기에서 신호가 켜져(on, high) 있는 시간의 비율

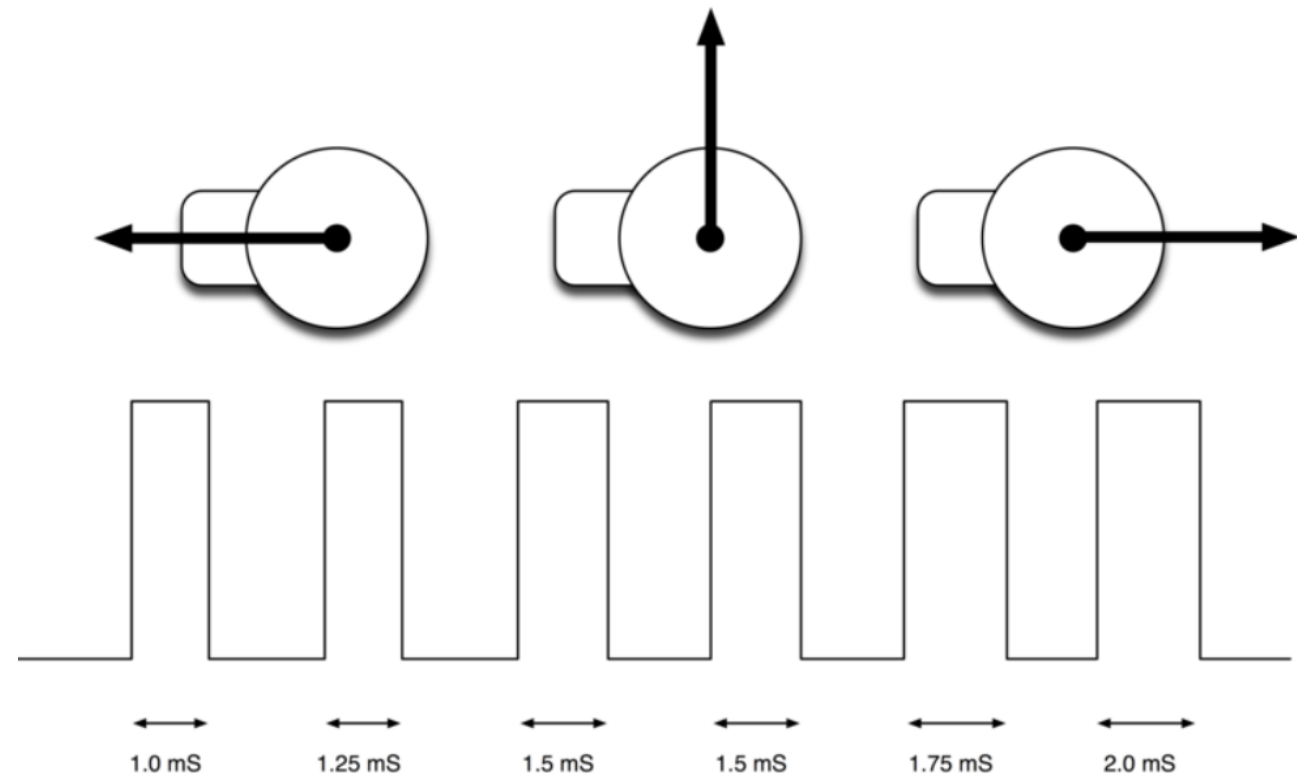
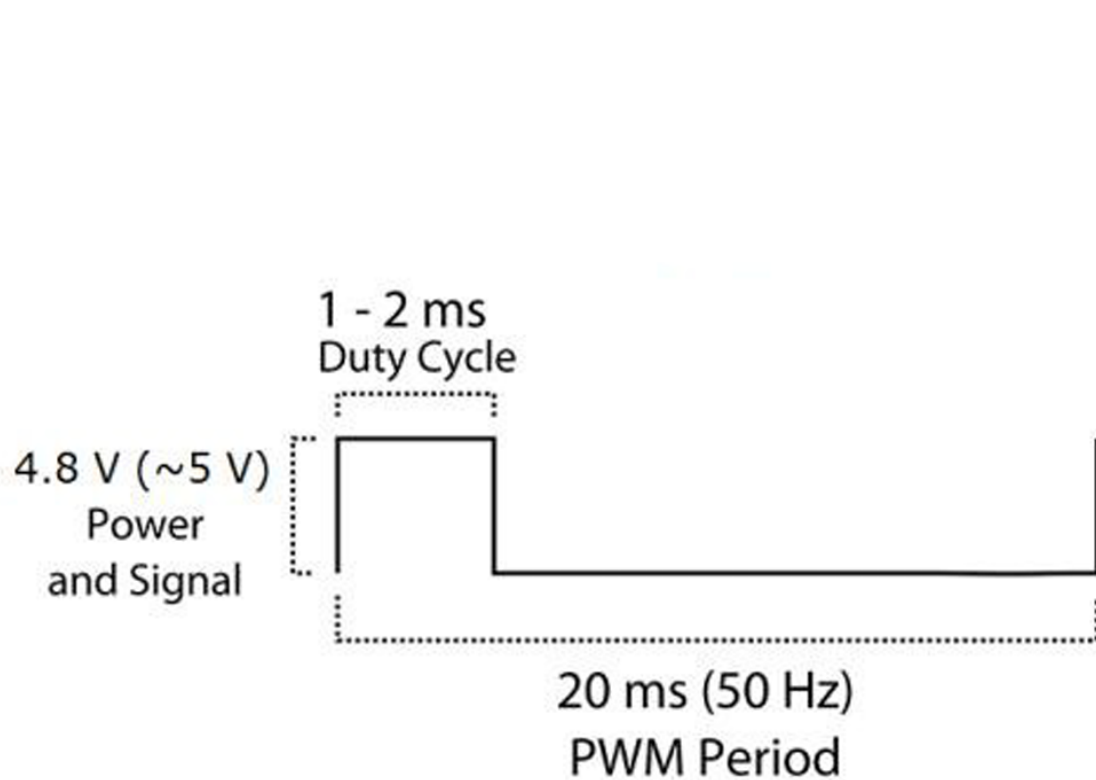


출처: <https://circuitdigest.com/tutorial/what-is-pwm-pulse-width-modulation>

RPi.GPIO 모듈의 PWM 사용

- PWM instance 생성
 - `p = gpio.PWM(channel, frequency)`
 - channel: 사용할 핀 번호
 - frequency: 신호 주파수
- PWM 시작
 - `p.start(dc)`
 - dc: duty cycle ($0.0 \leq dc \leq 100.0\%$)
- Duty cycle 변경
 - `p.ChangeDutyCycle(dc)`
- Frequency 변경
 - `p.ChangeFrequency(freq)`
- PWM 정지
 - `p.stop()`

SG90 PWM 제어



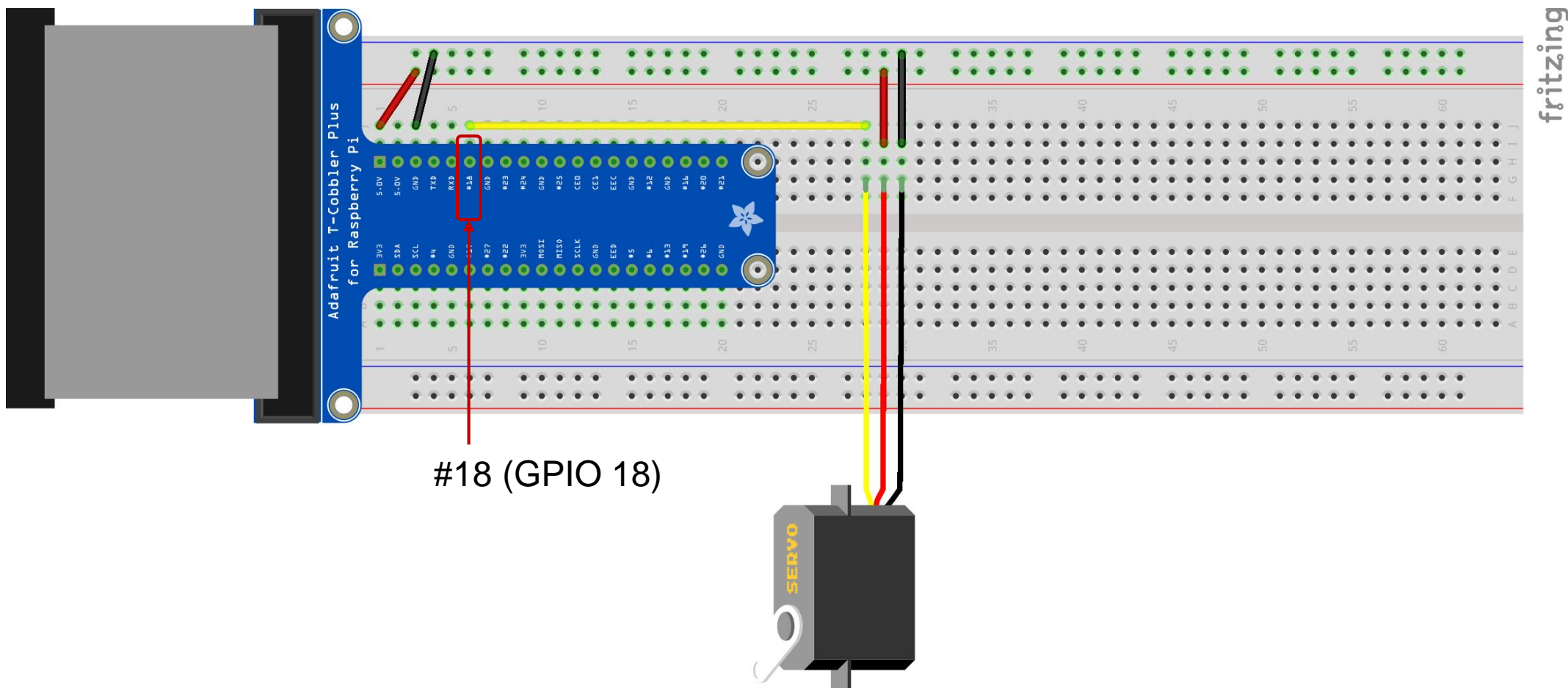
<https://content.instructables.com/ORIG/FA2/O1SS/J7ARLNBW/FA2O1SSJ7ARLNBW.pdf>

<https://rasino.tistory.com/341>

- 데이터시트 상 50Hz 주파수에서 1~2ms duty cycle(5%~10%)로 -90도에서 90도 각도 변화
- 실제로는 조금씩 차이가 있음

- 회로 구성

- 예제에서는 PWM 신호용으로 18번 핀 사용



예제 코드: /actuator_servo/servo.py

```
import RPi.GPIO as gpio
import time

pin = 18 # PWM pin
gpio.setmode(gpio.BCM)
gpio.setup(pin, gpio.OUT)

p = gpio.PWM(pin, 50)
p.start(0)
```

```
try:
    while True:
        p.ChangeDutyCycle(2.5)
        time.sleep(1)
        p.ChangeDutyCycle(5)
        time.sleep(1)
        p.ChangeDutyCycle(7.5)
        time.sleep(1)
        p.ChangeDutyCycle(10)
        time.sleep(1)
        p.ChangeDutyCycle(12.5)
        time.sleep(1)
```

```
except KeyboardInterrupt:
    p.stop()
    gpio.cleanup()
```

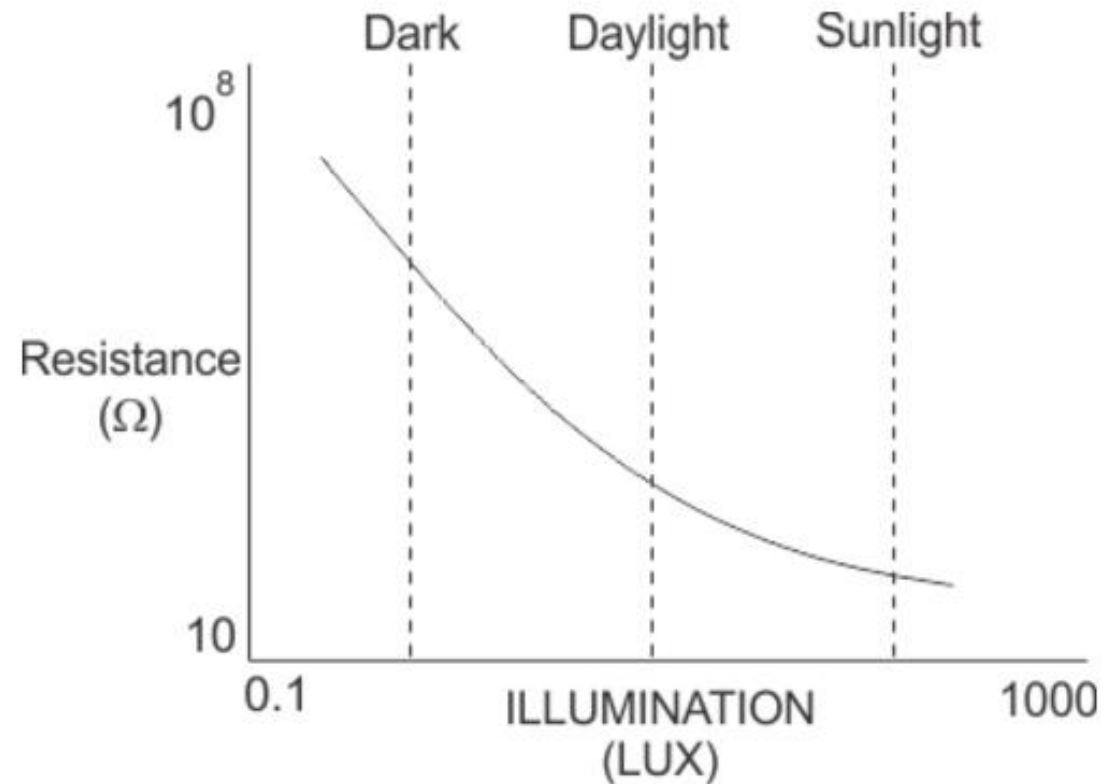
- 서보 모터의 회전 각도를 변경 제어하기 위해 PWM 신호의 duty cycle을 1초마다 변경하는 코드

아날로그 조도 센서 이용하기

- Raspberry Pi에서 아날로그 센서 이용
 - Raspberry Pi GPIO 핀은 디지털 신호의 입출력 용도로만 사용 가능
 - 아날로그 센서를 이용하기 위해서는 외부 ADC 칩을 사용
- 준비
 - Raspberry Pi와 GPIO 케이블로 연결된 코블러 브레이크아웃 보드와 브레드보드
 - MCP3008 칩 1개
 - Light Dependent Resistor 1개
 - 저항 1개
 - 점퍼 와이어

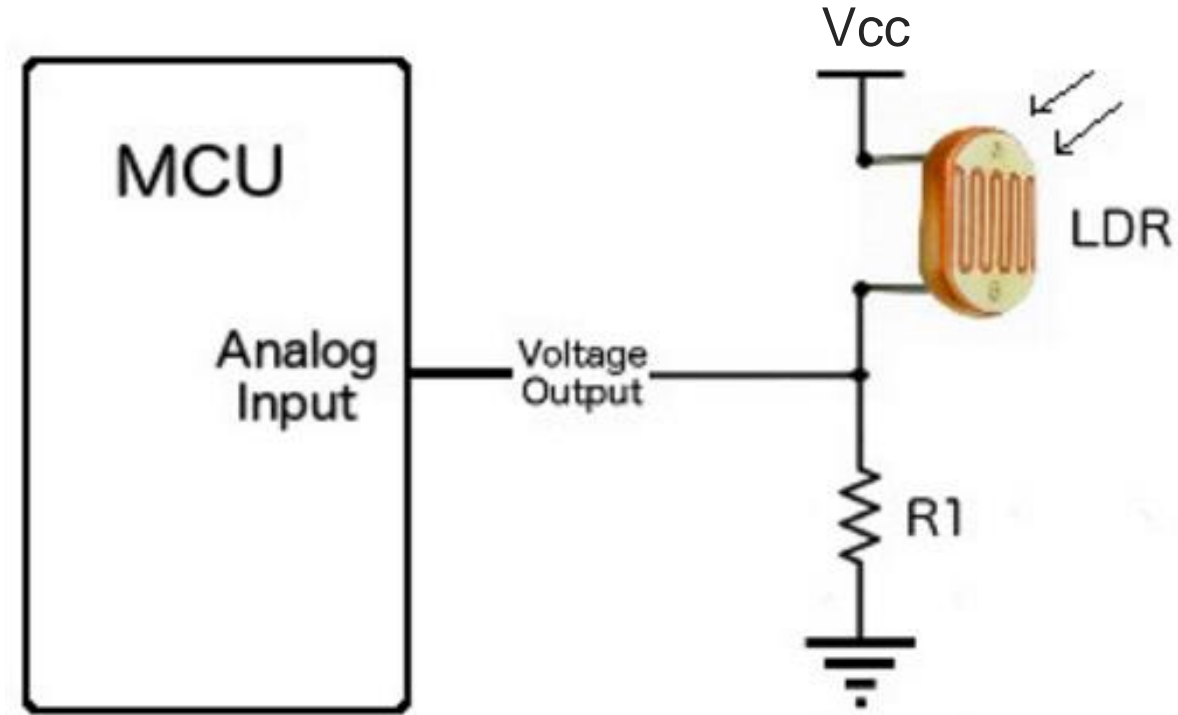
LDR - 아날로그 조도 센서

- LDR(Light Dependent Resistor)
 - 조도에 따라 저항 값이 바뀌는 특성을 이용하여 조도 센서로 사용 가능
 - 조도가 낮은 어두운 곳에서는 저항이 매우 커져서 수 M옴 이상으로 증가
 - 조도가 높은 밝은 곳에서는 저항이 작아져서 수 K옴 이하로 감소



- 사용 방법

- Voltage divider (전압분배기) 회로를 구성하여 사용
- 조도에 따라 저항값이 달라지므로 R1에 걸리는 전압이 그에 따라 달라짐
- 매우 어두운 곳
 - 조도 센서의 저항이 매우 커지므로 R1에 걸리는 전압 감소
 - R1이 상대적으로 많이 작은 경우 0V에 가까운 값을 가질 수 있음
- 매우 밝은 곳
 - 조도 센서의 저항이 매우 작아지므로 상대적으로 R1에 걸리는 전압 증가
- R1에 걸리는 전압 출력 → 아날로그 값
 - MCP3008의 아날로그 입력 채널로 연결되어 디지털 값으로 변환된 후 Raspberry Pi로 입력되어야 함



<http://cactus.io/-hookups/sensors/light/ldr/hookup-arduino-to-ldr-sensor>

MCP3008

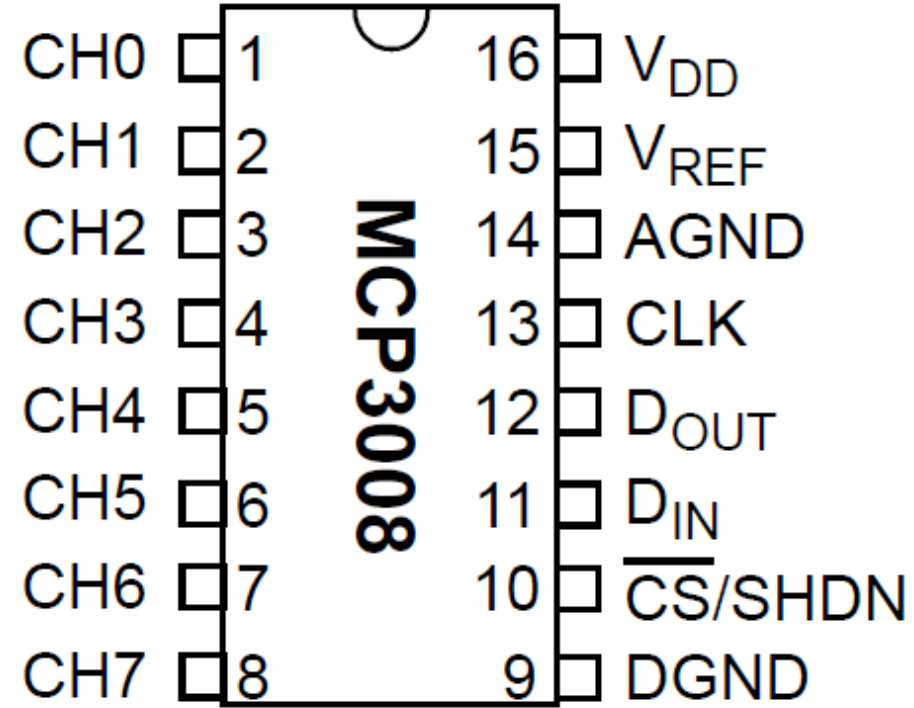
- MCP3008 칩
 - 아날로그 데이터를 디지털 데이터로 바꿔주는 ADC
 - 8개 아날로그 채널 입력
 - 10비트 디지털 데이터로 변환
 - 10비트 : 2의 10제곱 \rightarrow 0~1023
 - 초당 200 샘플링



<http://www.microchip.com/wwwproducts/Devices.aspx?product=MCP3008>

- MCP3008 핀 (총 16개)

- 좌 8개: 아날로그 입력 채널
- 9번 DGND(Digital Ground): 내부 디지털 회로와 연결되는 디지털 그라운드 핀
- 10번 CS/SHDN: Chip Select/Shutdown Input 핀
 - 통신을 시작하고 종료하기 위해 사용
 - Low 상태가 되면 통신을 시작하고, High 상태가 되면 AD 변환을 종료하고 기기를 저전력 대기모드로 변경
- 11번 DIN: 직렬 데이터 입력 핀
 - 칩의 채널 설정 데이터를 입력 받는데 사용
 - 여기에서는 라즈베리 파이에서 전송하는 데이터가 이 핀을 통해 입력
- 12번 DOUT: 직렬 데이터 출력 핀
 - AD 변환이 이루어진 결과 데이터를 전송하는데 사용
- 13번 CLK: 직렬 클럭 핀
 - AD 변환을 시작시키기 위해 사용
- 14번 AGND(Analog Ground): 내부 아날로그 회로와 연결되는 아날로그 그라운드 핀
- 15번 VREF: 레퍼런스 전압 입력 핀
 - 이 레퍼런스 전압이 아날로그 입력 전압 범위를 결정
 - 여기서는 라즈베리 파이에서 공급되는 3.3V 전원을 인가
- 16번 VDD: 전원 공급 핀
 - 2.7V에서 5.5V 전원 공급이 가능
 - 여기서는 라즈베리 파이에서 공급되는 3.3V 전원을 인가



SPI

- SPI(Serial Peripheral Interface, 직렬 주변기기 인터페이스)
 - MCP3008 칩은 SPI를 통하여 Raspberry Pi와 같은 마이크로컨트롤러 혹은 마이크로컴퓨터와 통신
- 외부 주변장치와 데이터를 주고 받기 위한 통신 방식 중 하나
- 마스터 슬레이브 모드를 기반으로 하여 하나의 마스터와 하나 혹은 다수의 슬레이브 장치 간 통신을 지원
- 마스터 장치와 슬레이브 장치 사이에 4가지 연결이 필요
 - SCLK, MISO, MOSI, CS
 - MCP3008 칩의 13번 CLK 핀, 12번 DOUT 핀, 11번 DIN 핀, 10번 CS/SHDN 핀에 대응

- 라즈베리 파이의 SPI

- 1개의 SPI 버스
- 2개의 Chip Select
- GPIO 핀 중에서 SPI 통신용으로 지정된 핀
 - SCLK(SPI_CLK): 클럭 신호를 출력
 - MCP3008의 13번 CLK 핀에 연결
 - MISO(SPI_MISO, SPI Master In Slave Out): SPI 마스터 기기가 데이터 입력을 받음
 - MCP3008의 12번 DOUT 핀에 연결
 - MOSI(SPI_MOSI, SPI Master Out Slave In): SPI 마스터 기기가 데이터 출력을 보냄
 - MCP3008의 11번 DIN 핀에 연결
 - CE0(Chip Enable 0) / CE1(Chip Enable 1): 2개의 Chip Select
 - MCP3008의 10번 CS/SHDN 핀을 이 둘 중 하나에 연결

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)		DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)		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 (I ² C ID EEPROM)		(I ² C 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

Rev. 2
29/02/2016

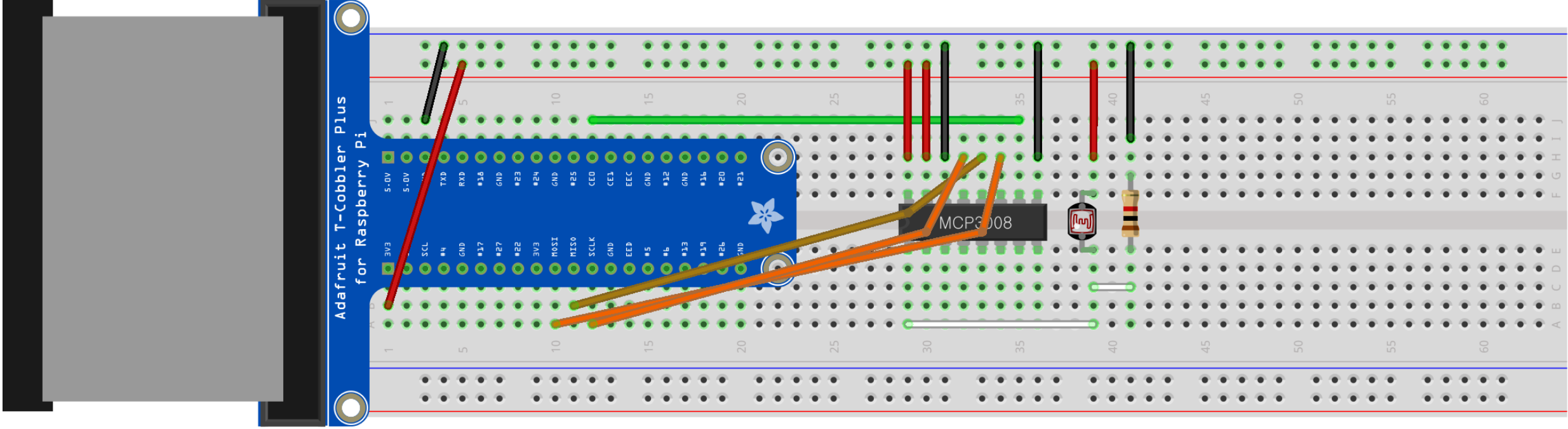
www.element14.com/RaspberryPi

- Raspberry Pi에서 SPI 통신 활성화
 - 기본 설정 메뉴 → Raspberry Pi Configuration → Interfaces 탭
 - SPI Enable 선택

혹은

- 터미널에서 raspi-config 명령어 실행 (sudo raspi-config)
→ Interface Options 선택 → SPI를 선택한 후 SPI를 사용하도록 설정

- 회로 구성
 - 전원 연결
 - 3.3V 전원
 - 그라운드 연결
 - MCP3008 칩 연결
 - 9번 DGND 핀, 14번 AGND 핀 \leftrightarrow 그라운드 (-) 홀
 - 15번 VREF 핀, 16번 VDD 핀 \leftrightarrow 전원 (+) 홀
 - 10번 CS/SHDN 핀 \leftrightarrow CE0 핀
 - 11번 DIN 핀 \leftrightarrow MOSI 핀
 - 12번 DOUT 핀 \leftrightarrow MISO 핀
 - 13번 CLK 핀 \leftrightarrow SCLK 핀
 - 조도 센서 연결
 - 센서 한쪽 핀 \leftrightarrow 전원 (+) 홀
 - 센서 다른 쪽 핀 \leftrightarrow MCP3008 칩 1번 CH0 핀
 - 저항(1K옴) 한쪽 핀 \leftrightarrow 그라운드 (-) 홀
 - 저항 다른 쪽 핀 \leftrightarrow 조도 센서 핀 (MCP3008에 연결된 쪽)



- spidev 모듈

- Raspberry Pi에서 SPI 기능을 이용하기 위해 사용하는 Python 모듈
 - 기본 설치되어 있음

- SPI 이용을 위한 기본 준비 과정

```
import spidev
```

```
spi = spidev.SpiDev()
```

```
spi.open(bus, device)
```

Raspberry Pi에는 1개의 SPI 버스와 2개의 Chip Select (CE0, CE1)

예제 코드: /sensor_SPI_ADC/light_mcp3008.py

```
import spidev
import time

spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz = 976000

light_channel = 0

def readChannel(channel):
    adc = spi.xfer2([1, (8 + channel) << 4, 0])
    adc_out = ((adc[1] & 3) << 8) + adc[2]
    return adc_out
```

```
def convert2volts(data, places):
    volts = (data * 3.3) / float(1023)
    volts = round(volts, places)
    return volts

try:
    while True:
        light_level = readChannel(light_channel)
        light_volts = convert2volts(light_level, 2)
        print("-----")
        print("Light: %d (%f V)" %(light_level, light_volts))
        time.sleep(0.2)
except KeyboardInterrupt:
    print("Finished")
    spi.close()
```

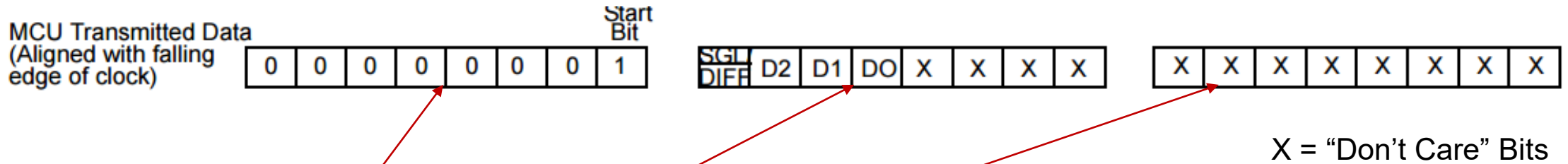
- readChannel 함수

- 선택된 아날로그 채널에서 아날로그 신호를 읽어서 디지털 데이터로 변환된 결과를 반환

- xfer2 함수

- Raspberry Pi에서 SPI 디바이스인 MCP3008로 통신을 시작하여 아날로그 입력 채널을 설정하는 제어 데이터를 보내고 그 결과 디지털 변환된 데이터를 받는 역할
- 입력과 출력은 3개의 8비트 데이터

❖ Raspberry Pi에서 MCP3008 칩으로 전송되는 데이터

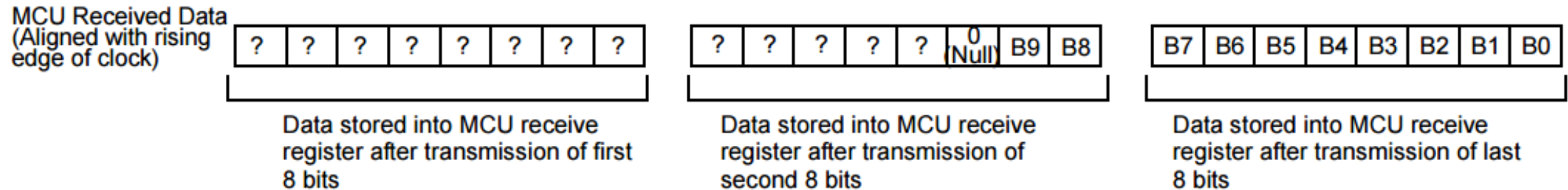


`adc = spi.xfer2([1, (8 + channel) << 4, 0])`

- 1
 - 첫 8비트 세그먼트의 Start Bit
- $(8 + \text{channel}) \ll 4$
 - 둘째 8비트 세그먼트 (channel이 0이면 $\rightarrow 1000\ 0000$)
 - SGL/DIFF 비트만 1이고, D2, D1, D0 비트는 모두 0 \rightarrow 하나의 채널만 사용하는 싱글 입력이고 그 채널은 0번
- 0
 - 세 번째 8비트 세그먼트
 - don't care 비트로 프로그램에서는 0으로 지정

Control Bit Selections				Input Configuration	Channel Selection
Single/Diff	D2	D1	D0		
1	0	0	0	single-ended	CH0
1	0	0	1	single-ended	CH1
1	0	1	0	single-ended	CH2
1	0	1	1	single-ended	CH3
1	1	0	0	single-ended	CH4
1	1	0	1	single-ended	CH5
1	1	1	0	single-ended	CH6
1	1	1	1	single-ended	CH7

❖ MCP3008 칩에서 Raspberry Pi로 전송되는 데이터



- 변수 `adc` (앞의 `spi.xfer2([1, (8 + channel) << 4, 0])`의 반환 값)
 - 배열 데이터
 - `adc[0]`: 첫째 8비트 세그먼트
 - `adc[1]`: 둘째 8비트 세그먼트
 - `adc[2]`: 셋째 8비트 세그먼트
 - 유효 데이터
 - 두 번째 세그먼트에서는 하위 2비트
 - 세 번째 세그먼트 8비트
 - 이를 10비트 데이터로 변환하기 위하여 `((adc[1] & 3) << 8) + adc[2]` 연산 수행

- `convert2volts(data, places)` 함수
 - `data` 변수의 데이터를 전압 값으로 변경해주는 함수
 - 저항에 걸린 전압 값이 MCP3008 칩에 의해 10비트 디지털 데이터(0-1023 범위 값)로 변환된 것이므로 1023으로 나눠주고 3.3(기준 전압)을 곱해주면 해당 전압 값이 계산됨