

스마트 모빌리티 프로그래밍

Ch 15. 스마트 모빌리티 플랫폼에서의 사물인터넷 프로그래밍



정보통신공학과
교수 김 영 탁

(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- ◆ **RPi.GPIO Module for Python Programming**
- ◆ **spidev module**
- ◆ **LED On/Off, Color LED**
- ◆ **SG90 Servo Motor**
- ◆ **Step Motor**
- ◆ **DHT11 온도 및 습도 센서**
- ◆ **MCP3208 A/D Converter, MCP4922 D/A Converter**
- ◆ **Photo Sensor (조도 센서)**
- ◆ **Simple Function Generator, Simple Oscilloscope**
- ◆ **6축 Gyro, Accelerometer (자이로 가속도계) 센서**
- ◆ **GPS (Global Positioning System)**
- ◆ **근거리 측정: Ultrasonic Ranger, TF mini-LiDAR**



RPi.GPIO
piGPIO

RPi.GPIO

◆ RPi.GPIO (version 0.7.0)

- reference: <https://pypi.org/project/RPi.GPIO/> (July 21, 2019)
- RPi.GPIO package provides a class to control the GPIO on a Raspberry Pi.
- this module is unsuitable for real-time or timing critical applications, because you can not predict when Python will be busy garbage collecting. It also runs under the Linux kernel which is not suitable for real time applications - it is multitasking O/S and another process may be given priority over the CPU, causing jitter in your program. If you are after true real-time performance and predictability, implement on Arduino <http://www.arduino.cc> !
- the current release 0.7.0 does not support SPI, I2C, hardware PWM or serial functionality on the RPi yet. This is planned for the near future - watch this space! One-wire functionality is also planned.
- Although hardware PWM is not available yet, software PWM is available to use on all channels.
- For examples and documentation, visit <http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/>
- <https://sourceforge.net/p/raspberry-gpio-python/wiki/TechRef/>



pigpio

◆ pigpio ([pigpio library \(abyz.me.uk\)](http://pigpio.library(abyz.me.uk)))

- pigpio is an actively developed library with an impressive set of features
- all GPIO pins of the Raspberry Pi can be read, written to, attached to interrupt handlers, and output PWM signals at the same time. Also, UART, I2C and SPI protocols are implemented
- it is written in C and also provides Python bindings.
- pigpio is a port of pigpio's C functionality to Python.
- RPIO.GPIO, it provides hardware timed PWM for LEDs and servos, and hardware timed pulse chains.
- One unique feature of pigpio is it times GPIO events at source so for any timing purpose pigpio is likely to be the most accurate.
- pigpio can run on a networked PC (Windows, Mac, Linux) to control a remote Pi.



RPi.GPIO 패키지 설치 확인

◆ Check RPi.GPIO

```
pi@raspberrypi:~ $ python3
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import RPi.GPIO
>>> 
```

◆ Installation of RPi.GPIO

- \$ sudo apt-get install python-dev
- \$ sudo apt-get install python-rpi.gpio

◆ Checking RPi.GPIO in Python Program

```
try:
    import RPi.GPIO as GPIO
except RuntimeError:
    print("Error importing RPi.GPIO !")
    print("This is probably because you need super-user privileges.")
    print("Try using 'sudo' to run your script with super-user privilege.")
```

Numbering of IO Pins on Raspberry Pi within RPi.GPIO

◆ Two ways of numbering the IO pins on a Raspberry Pi within RPi.GPIO

- `GPIO.setmode(GPIO.BOARD)` – pin numbers on the P1 header of Raspberry Pi board
- `GPIO.setmode(GPIO.BCM)` – channel numbers on the Broadcom SOC

◆ Detection of current pin numbering

- `mode = GPIO.getmode()`

Set up a channel

◆ Setting up a channel as input or output

- `GPIO.setup(channel, GPIO.IN)`
- `GPIO.setup(channel, GPIO.OUT)`
- `GPIO.setup(channel, GPIO.OUT, initial=GPIO.HIGH)`

Input and Output

◆ Input

- `GPIO.input(channel)`

◆ Output

- `GPIO.output(channel, state)`
- state can be
0, `GPIO.LOW`, `False`, or
1, `GPIO.HIGH`, `True`

◆ Output to several channels

- `chan_list = [11, 12]`
- `GPIO.output(chan_list, GPIO.LOW)`
- `GPIO.output(chan_list, (GPIO.HIGH, GPIO.LOW))`

Clean Up

◆ Cleanup

- return the (all) used channel(s) back to inputs with no pull up/down, to avoid accidental damage to RPi by shorting out the pins
- `GPIO.cleanup()` also clears the pin numbering system in use

◆ Cleanup Channel(s)

- `GPIO.cleanup()` # cleanup all channels
- `GPIO.cleanup(channel)`
- `GPIO.cleanup((chan_1, chan_2))`
- `GPIO.cleanup([chan_1, chan_2])`

RPi Board Information and RPi.GPIO version

◆ Discovering Information of your RPi

- GPIO.RPI_INFO # discover information about RPi
- GPIO.RPI_INFO['PI_REVISION']
discover Raspberry Pi Board version
- GPIO.VERSION # discover the version of RPi.GPIO



RPi.GPIO Inputs

◆ GPIO Inputs

- There are several ways of getting GPIO input into your program.
- The first and simplest way is to check the input value at a point in time. This is known as '**polling**' and can potentially miss an input if your program reads the value at the wrong time.
- Polling is performed in loops and can potentially be processor intensive.
- The other way of responding to a GPIO input is using '**interrupts**' (edge detection).
- An edge is the name of a transition from HIGH to LOW (*falling edge*) or LOW to HIGH (*rising edge*).

RPi.GPIO Inputs

◆ Pull up / Pull down resistors

- If you do not have the input pin connected to anything, it will 'float'. In other words, the value that is read in is undefined because it is not connected to anything until you press a button or switch. It will probably change value a lot as a result of receiving mains interference.
- To get round this, a pull up or a pull down resistor is used. In this way, the default value of the input can be set. It is possible to have pull up/down resistors in hardware and using software.
- In hardware, a 10K resistor between the input channel and 3.3V (pull-up) or 0V (pull-down) is commonly used.
- The RPi.GPIO module allows you to configure the Broadcom SOC to do this in software:
 - `GPIO.setup(channel, GPIO.IN, pull_up_down = GPIO.PUD_UP)`
 - `GPIO.setup(channel, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)`

RPi.GPIO Inputs

◆ Testing inputs with polling

```
if GPIO.input(channel):  
    print("gpio ({}) if HIGH".format(channel))  
else:  
    print("gpio ({}) if LOW".format(channel))
```

◆ Waiting for a button press by polling in a loop

```
while GPIO.input(channel) == GPIO.LOW:  
    time.sleep(0.01) # wait 10 ms to give CPU chance to do other tasks
```

RPi.GPIO Interrupts and Edge detection

◆ Interrupts

- An edge is the change in state of an electrical signal from LOW to HIGH (**rising edge**) or from HIGH to LOW (**falling edge**).
- Quite often, we are more concerned by a change in state of an input than it's value. This change in state is an **event**.
- To avoid missing a button press while your program is busy doing something else, there are two ways to get round this:
 - the **wait_for_edge()** function
 - the **event_detected()** function
 - a threaded callback function that is run when an edge is detected

RPi.GPIO wait_for_edge()

◆ wait_for_edge() function

- designed to block execution of program until an edge is detected

```
GPIO.wait_for_edge(channel, GPIO.RISING)
GPIO.wait_for_edge(channel, GPIO.FALLING)
GPIO.wait_for_edge(channel, GPIO.BOTH)
```

- wait for a certain length of time with timeout parameter

```
channel = GPIO.wait_for_edge(channel, GPIO.RISING, timeout=5000)
# timeout in milli-second
if channel is None:
    print("Timeout occurred !!")
else:
    print("Edge detected on channel ({})".format(channel))
```


RPi.GPIO event_detected()

◆ event_detected() function

- designed to be used in a loop with other things
- unlike polling, it is not going to miss the change in state of an input while the CPU is busy working on other things
- could be useful when using something like Pygame or PyQt where there is a main loop listening and responding to GUI events in a timely basis

```
GPIO.add_event_detect(channel, GPIO.RISING)
# event may also be GPIO.FALLING, GPIO.BOTH
do_something()
if GPIO.event_detected(channel):
    print("Event was detected on channel {}".format(channel))
```

RPi.GPIO Threaded Callbacks

◆ Threaded callbacks

- RPi.GPIO runs a second thread for callback functions.
- the callback functions can be run at the same time as the main program, with immediate response to an edge

```
def my_callback(channel):  
    print("Callback function for an edge event on channel({}).format(channel))  
  
GPIO.add_event_detect(channel, GPIO.RISING, callback=my_callback)
```

RPi.GPIO Switch debounce

◆ Switch debounce

- callbacks may be called more than once for each button press because of the switch bounce
- ways of dealing with switch bounce:
 - add a 0.1uF capacitor across the press button
 - software debouncing
 - a combination of both

```
def my_callback(channel):  
    print("Callback function for an edge event on channel({})".format(channel))  
  
GPIO.add_event_detect(channel, GPIO.RISING, callback=my_callback, W  
    bouncetime=200)  
GPIO.add_event_callback(channel, my_callback, bouncetime=200)
```

RPi.GPIO Remove Event Detection

◆ Remove event detection

- if program no longer wishes to detect edge events, it is possible to stop them

```
GPIO.remove_event_detect(channel)
```

GPIO Outputs

◆ Setup RPi.GPIO

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT)
```

◆ Output High

```
GPIO.output(12, GPIO.HIGH)
GPIO.setup(12, 1)
GPIO.setup(12, True)
```

◆ Output Low

```
GPIO.output(12, GPIO.LOW)
GPIO.setup(12, 0)
GPIO.setup(12, False)
```

GPIO Outputs

◆ Output to several channels at the same time

```
chan_list = (11, 12)
GPIO.output(chan_list, GPIO.LOW)
GPIO.output(chan_list, (GPIO.HIGH, GPIO.LOW))
```

◆ Toggling output

```
GPIO.output(channel, not GPIO.input(channel))
```

◆ Clean up at the end of program

```
GPIO.cleanup()
```

Using PWM in RPi.GPIO

◆ Create a PWM instance

```
p = GPIO.PWM(channel, frequency)
```

◆ Start and stop PWM

```
p.start(duty_cycle) # 0.0 <= duty_cycle <= 100.0  
p.stop()
```

◆ Change the frequency and duty_cycle

```
p.ChangeFrequency(freq)  
p.ChangeDutyCycle(new_duty_cycle) # 0.0 <= new_duty_cycle <= 100.0
```

RPi.GPIO 응용 프로그램

gpio_function(channel)

◆ GPIO.gpio_function(channel)

- Shows the function of a GPIO channel
- return value: GPIO.IN, GPIO.OUT, GPIO.SPI, GPIO.I2C, GPIO.HARD_PWM, GPIO.SERIAL, GPIO.UNKNOWN

```
# GPIO.gpio_function()
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)

pins = [8, 9, 19, 21, 32, 33]
for pin in pins:
    func = GPIO.gpio_function(pin)
    print("function of GPIO pin ({} ) = {}".format(pin, func))
```

Blinking an LED once every 2 seconds

```
# Blinking an LED once every 2 seconds
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT)

p = GPIO.PWM(12, 0.5) # channel 12, frequency 0.5Hz
p.start(1) # duty_cycle 1
input("Press return to stop : ")
p.stop()
GPIO.cleanup()
```

Brighten/Dim an LED

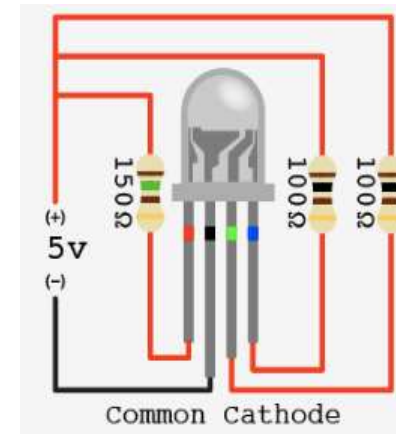
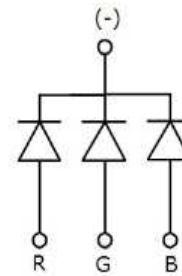
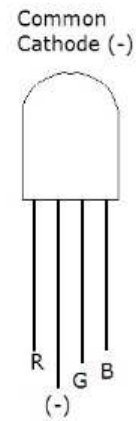
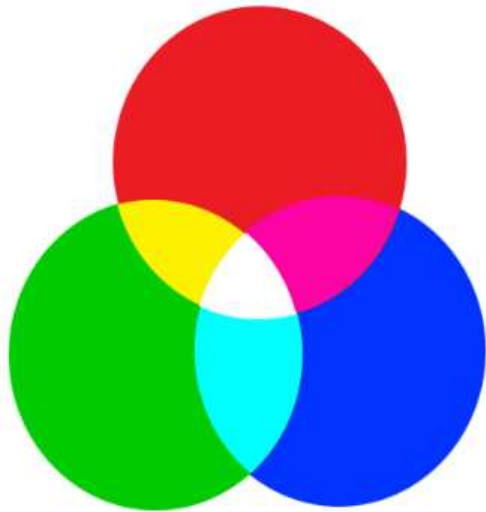
```
# Brighten/Dim LED
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT)

p = GPIO.PWM(12, 50) # channel 12, frequency 50Hz
p.start(0) # initial duty_cycle = 0
try:
    while True:
        for dc in range(0, 101, 5):
            p.ChangeDutyCycle(dc) # dc : 0, 5, 10, 15, . . . ~ 95, 100
            time.sleep(0.1)
        for dc in range(100, -1, -5):
            p.ChangeDutyCycle(dc) # dc : 100, 95, 90, . . . 5, 0
            time.sleep(0.1)
except KeyboardInterrupt:
    pass
p.stop()
```

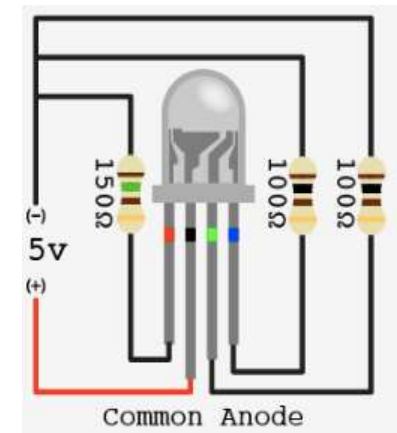
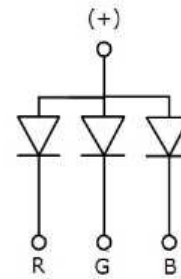
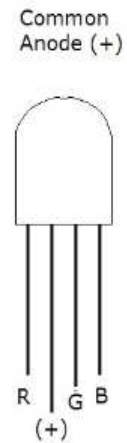


RGB Color LED

◆ Color LED



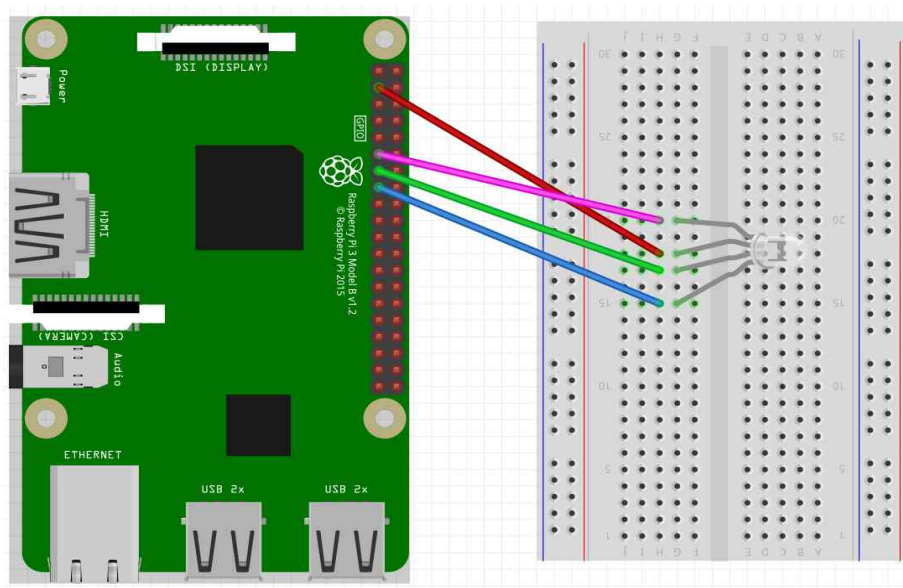
<Common Cathode>



<Common Anode>

Color LED 회로도

◆ Color LED 회로 구성



Color LED Control

◆ \$ sudo Color_LED.py

```
# color_LED_Control (1)
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
RUNNING = True
green = 13
red = 19
blue = 26

GPIO.setup(red, GPIO.OUT)
GPIO.setup(green, GPIO.OUT)
GPIO.setup(blue, GPIO.OUT)
Freq = 100

RED = GPIO.PWM(red, Freq)
GREEN = GPIO.PWM(green, Freq)
BLUE = GPIO.PWM(blue, Freq)
```

Raspberry Pi 3 GPIO Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO12 (SDA1, I ² C)		DC Power 5v	04
05	GPIO13 (SCL1, I ² C)		Ground	06
07	GPIO14 (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	GPIO19 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE1_N) GPIO18	24
25	Ground		(SPI_CE1_N) GPIO17	26
27	ID_SD (I ² C ID EEPROM)		(I ² C ID EEPROM) ID_SC	28
29	GPIO15		Ground	30
31	GPIO16		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

www.element14.com/RaspberryPi

```
# color_LED_Control (2)
try:
    RED.start(1)
    GREEN.start(1)
    BLUE.start(1)
    while RUNNING:
        for x in range(1,101, 5):
            BLUE.ChangeDutyCycle(101-x)
            GREEN.ChangeDutyCycle(x)
            time.sleep(0.05)
        for x in range(1,101, 5):
            GREEN.ChangeDutyCycle(101-x)
            RED.ChangeDutyCycle(x)
            time.sleep(0.05)
        for x in range(1,101, 5):
            RED.ChangeDutyCycle(101-x)
            BLUE.ChangeDutyCycle(x)
            time.sleep(0.05)
    except KeyboardInterrupt:
        RUNNING = False
        GPIO.cleanup()
```



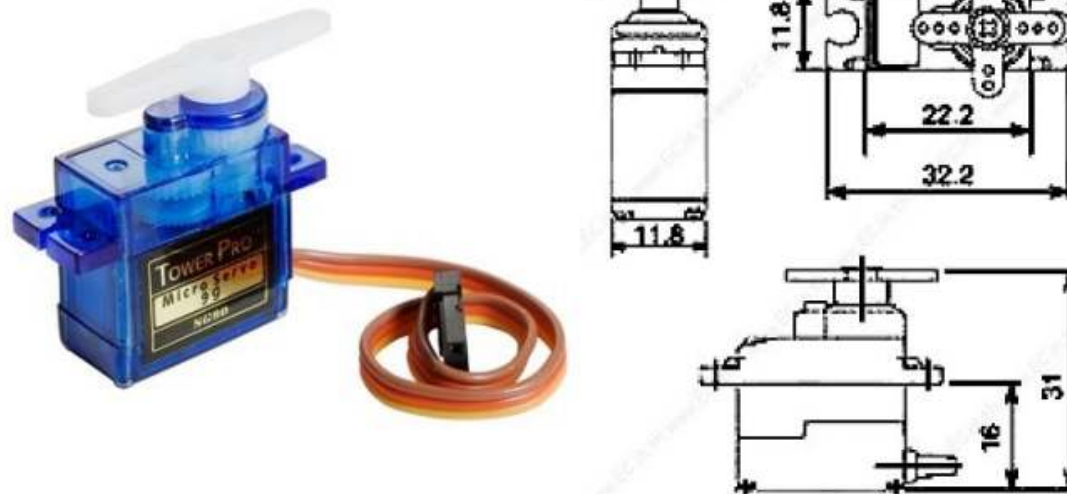
SG90 Servo Motor

◆ SG90

- Tiny and lightweight with high output power.
- Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller.
- You can use any servo code, hardware or library to control these servos.
- Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

SG90 Servo Motor

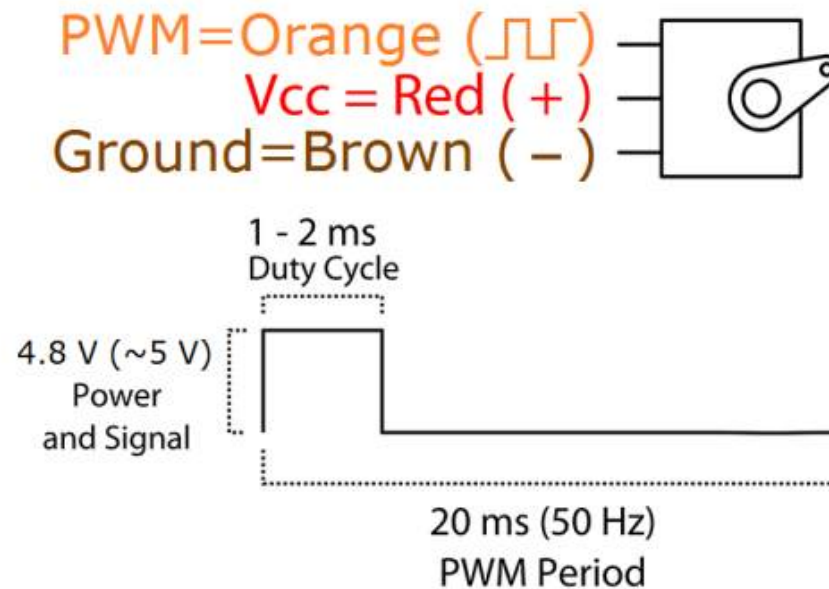
◆ SG90 Servo Motor



(src: <http://www.micropik.com/PDF/SG90Servo.pdf>)

SG90 Servo Motor

◆ SG90 Control(1)

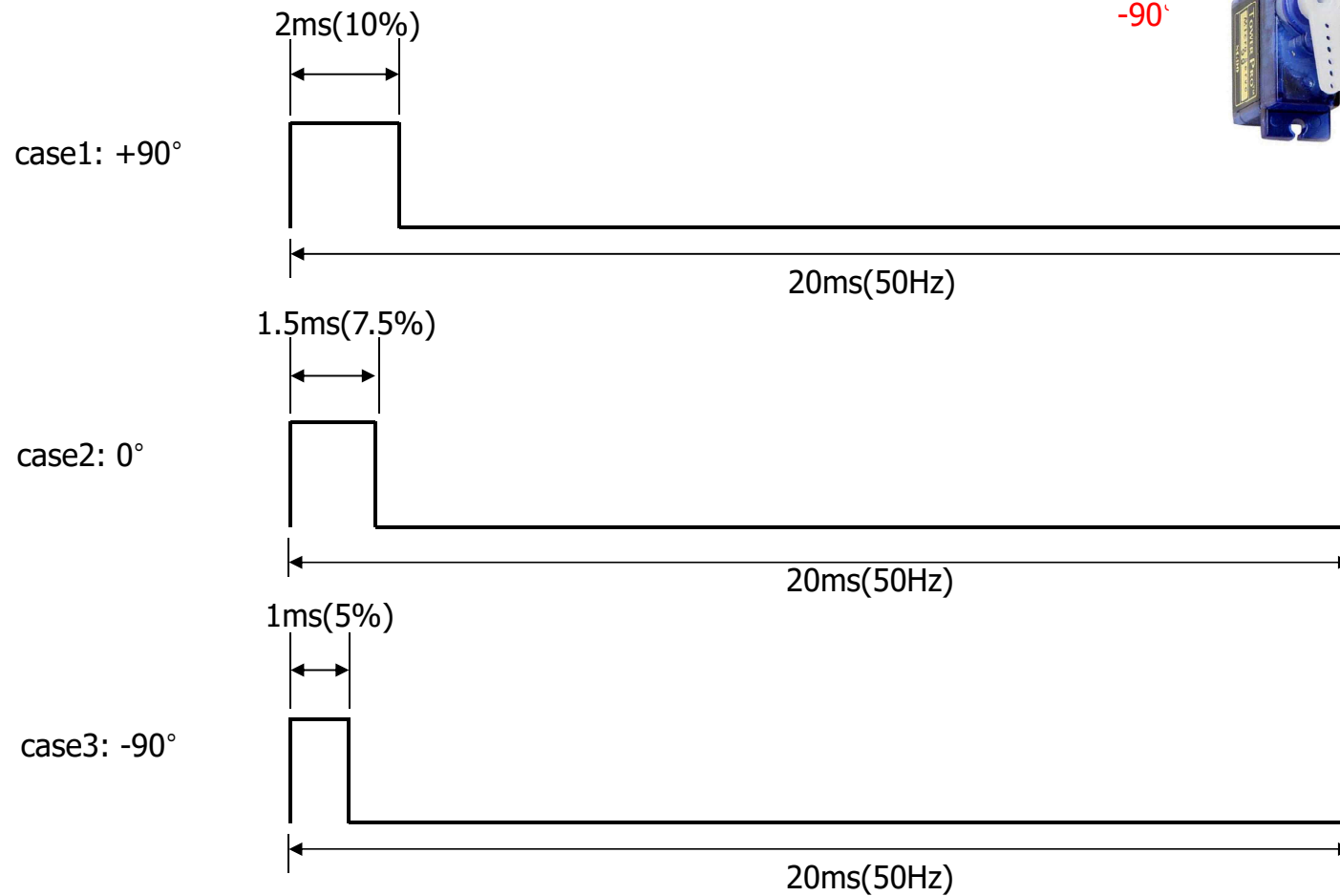
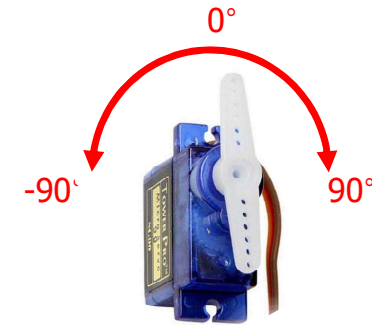


Position "0" (1.5 ms pulse) is middle, "90" (~2 ms pulse) is all the way to the right, "-90" (~1 ms pulse) is all the way to the left.

(source: <http://www.micropik.com/PDF/SG90Servo.pdf>)

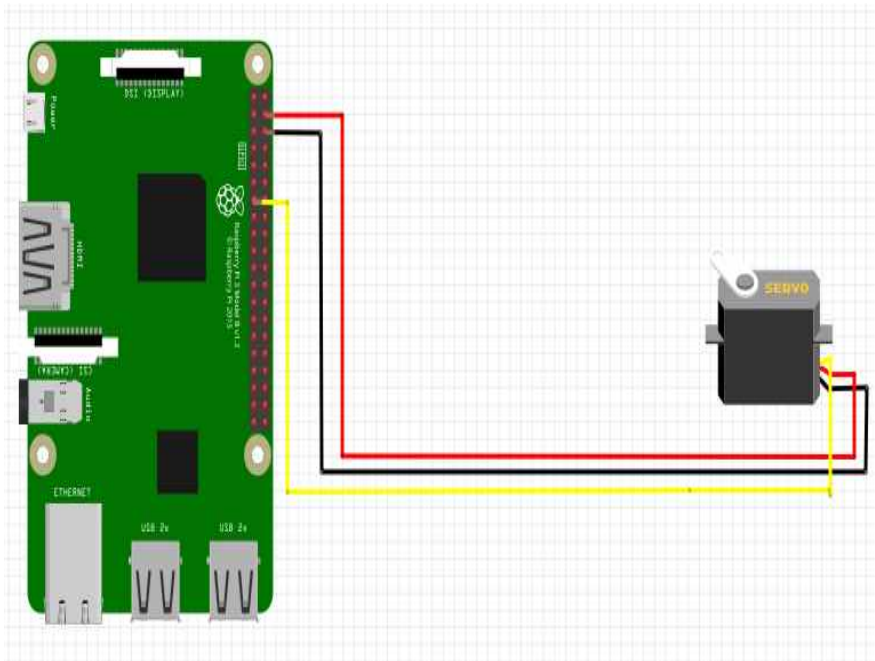
SG90 Servo Motor

◆ SG90 Control(2)



Servo Motor Control

◆ Servo motor 회로 구성 (GPIO 17)

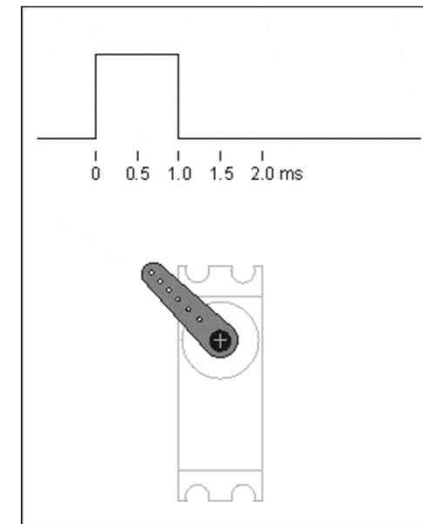
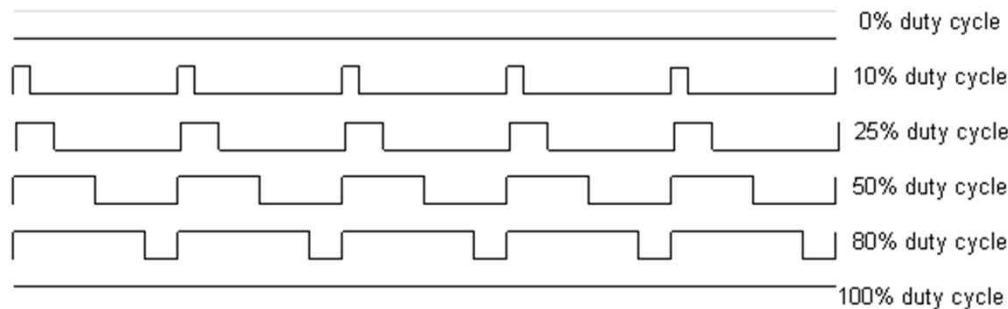


Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)	DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) 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

Servo Motor Control with PWM

◆ PWM(Pulse Width Modulation)이란?

- PWM은 5V출력을 on 혹은 off 상태로 유지하는 것이 아니라 빠르게 on/off 를 반복
- 일정한 주기 내에서 duty ratio를 변화 시켜서 평균전압을 제어
- Raspberry Pi에서 서보 모터로 연결된 핀으로 전달되는 신호의 지속시간(pulse width)으로 위치를 제어



SG90 Servo Motor Control

◆ \$ sudo nano servotest.py

```
#SG90 servo_motor_test.py (1)
```

```
import RPi.GPIO as GPIO
import time
```

```
servo_pin = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(servo_pin, GPIO.OUT)
```

```
servo_mtr = GPIO.PWM(servo_pin, 50) # freq 50Hz
servo_mtr.start(0) # initial duty_cycle = 0
left_angle = 10
center_angle = 7.5
right_angle = 2.5 # less than 5
```

```
def setAngle(angle):
```

```
    servo_mtr.ChangeDutyCycle(angle)
    time.sleep(0.5)
```

Raspberry Pi 3 GPIO Header			
Pin#	NAME		NAME Pin#
01	3.3v DC Power		DC Power 5v 02
03	GPIO2 (SDA1, I ² C)		DC Power 5v 04
05	GPIO3 (SCL1, I ² C)		Ground 06
07	GPIO4 (GPIO_CLK)		(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	GPIO9 (SPI_MISO)		(GPIO_GEN6) GPIO25 22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO28 24
25	Ground		(SPI_CE1_N) GPIO27 26
27	ID_SD (I ² C ID EEPROM)		(I ² C ID EEPROM) ID_SC 28
29	GPIO5		Ground 30
31	GPIO6		GPIO12 32
33	GPIO13		Ground 34
35	GPIO19		GPIO16 36
37	GPIO26		GPIO20 38
39	Ground		GPIO21 40

```
#SG90 servo_motor_test.py (2)
```

```
try:
```

```
    while True:
```

```
        var = input("Enter L/C/R : ")
```

```
        if var == 'R' or var == 'r':
```

```
            setAngle(right_angle)
```

```
        elif var == 'C' or var == 'c':
```

```
            setAngle(center_angle)
```

```
        elif var == 'L' or var == 'l':
```

```
            setAngle(left_angle)
```

```
        else: # not left (L, l), center (C, c) or right (R, r)
```

```
            servo_mtr.stop()
```

```
            break
```

```
        print("=====")
```

```
except KeyboardInterrupt:
```

```
    servo_mtr.stop()
```

```
GPIO.cleanup()
```



pigpiod Servo motor(1)

◆ Pigpiod

- `sudo apt-get update`
- `Sudo apt-get upgrade`
- `cd`
- `wget https://github.com/joan2937/pigpio/archive/master.zip`
- `Unzip master.zip`
- `Cd pigpio-master`
- `Make`
- `Sudo make install`
- 만약 설치에 실패한다면
- `Sudo apt install python-setuptools python3-setuptools`를 설치

pigpiod Servo motor(2)

◆ Source Code

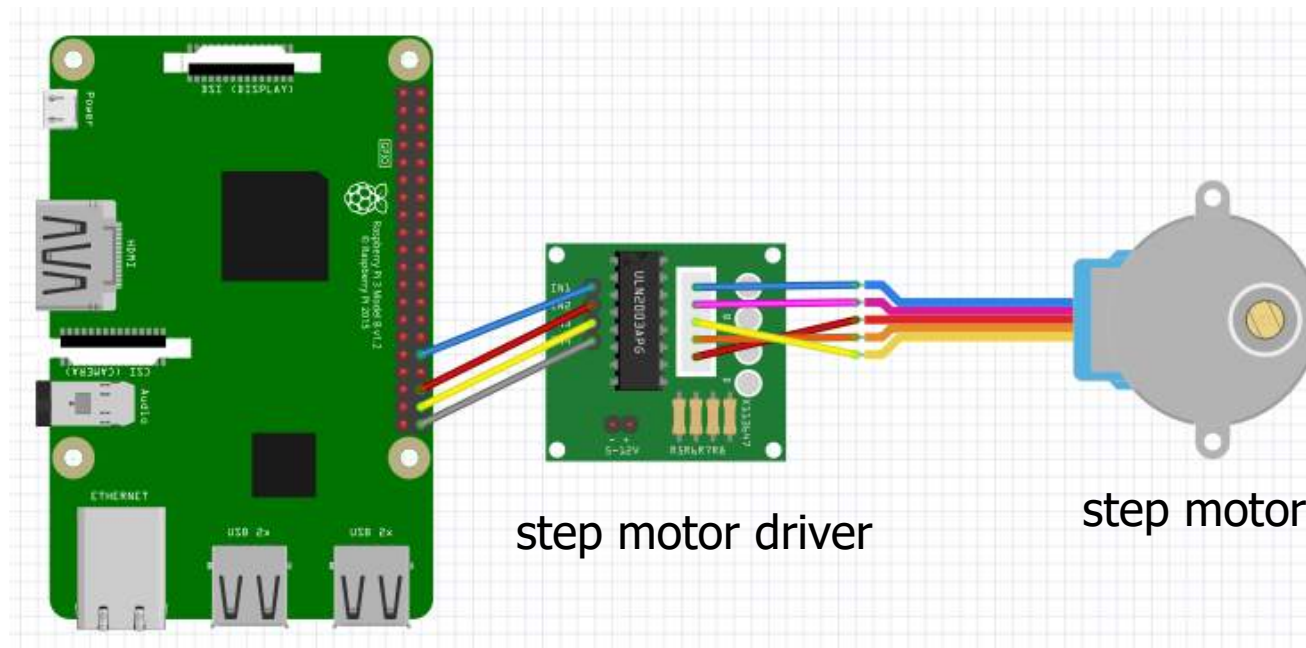
```
import pigpio
from time import sleep

def SetServoAngle(pin, angle):
    ang = 600+(10*angle)
    pi.set_servo_pulsewidth(pin, ang) #500~2500범위 내에서 조절해야함
    sleep(0.1)

if __name__=="__main__":
    pi=pigpio.pi()
    pin=17 #BCM
    data=0
    while True:
        try:
            data=int(input())
            SetServoAngle(pin, data)
        except KeyboardInterrupt:
            break
```

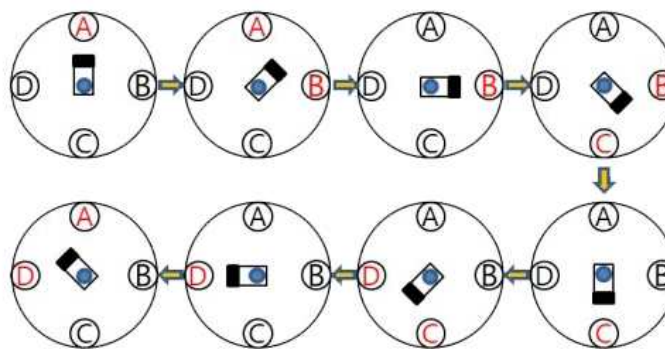
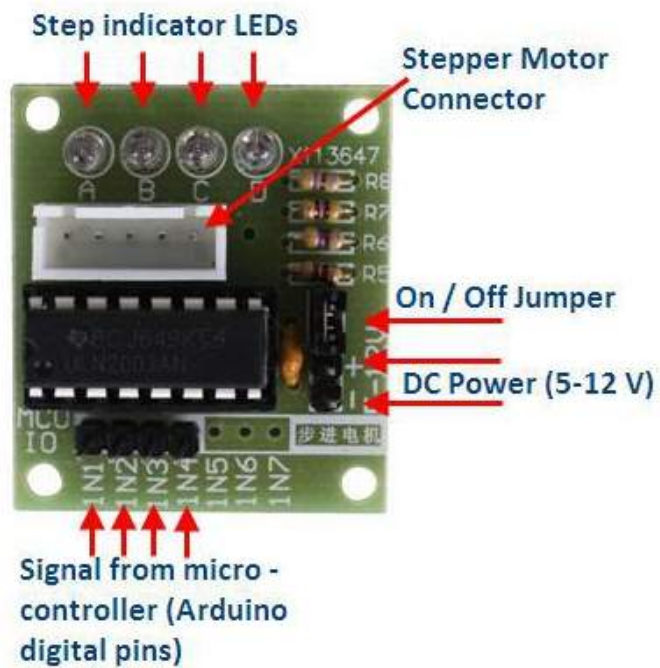
Step motor Control

◆ Step motor 회로 구성



Step motor 제어방식

◆ Step Motor Drive Modle



모터상	1	2	3	4	5	6	7	8
A (A) – in1	1	1	0	0	0	0	0	1
B (B) – in2	0	1	1	1	0	0	0	0
C (/A) – in3	0	0	0	1	1	1	0	0
D (/B) – in4	0	0	0	0	0	1	1	1

Step Motor Control

◆ \$ sudo nano step_motor_test.py

```
#step_motor_test.py (1)
import time
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
StepPins = [12,16,20,21]

for pin in StepPins:
    GPIO.setup(pin,GPIO.OUT)
    GPIO.output(pin,False)
StepCounter = 0
Seq = [[0,0,0,1], [0,0,1,1],
        [0,0,1,0], [0,1,1,0],
        [0,1,0,0], [1,1,0,0],
        [1,0,0,0]], [1,0,0,1]
Num_Steps = len(Seq)
```

Raspberry Pi 3 GPIO Header			
Pin#	NAME		NAME Pin#
01	3.3v DC Power		DC Power 5v 02
03	GPIO12 (SDA1, 1°C)		DC Power 5v 04
05	GPIO13 (SCL1, 1°C)		Ground 06
07	GPIO14 (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	GPIO19 (SPI_MISO)		(GPIO_GEN6) GPIO25 22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO18 24
25	Ground		(SPI_CE1_N) GPIO17 26
27	ID_SD (1°C ID EEPROM)		(1°C ID EEPROM) ID_SC 28
29	GPIO15		Ground 30
31	GPIO16		GPIO12 32
33	GPIO13		Ground 34
35	GPIO19		GPIO16 36
37	GPIO26		GPIO20 38
39	Ground		GPIO21 40

```
#step_motor_test.py (2)
try:
    while True:
        for pin in range(0, 4):
            xpin = StepPins[pin]
            if Seq[StepCounter][pin] == 1:
                GPIO.output(xpin, True)
            else:
                GPIO.output(xpin, False)
            StepCounter += 1

        if (StepCounter == Num_Steps):
            StepCounter = 0

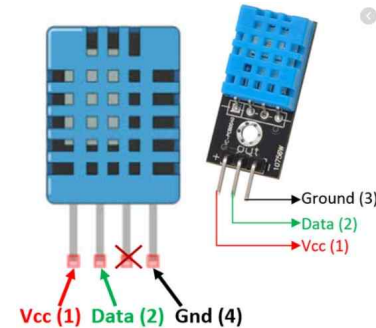
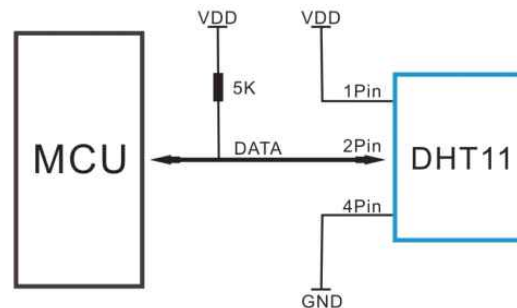
        time.sleep(0.1) #interval 0.1 sec

except KeyboardInterrupt:
    GPIO.cleanup()
```

온도 및 습도 측정 - DHT11 Humidity & Temperature Sensor

◆ DHT11 Humidity & Temperature Sensor

- DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability.
- This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness.



ch 15 - 43

DHT11 Data Transmission

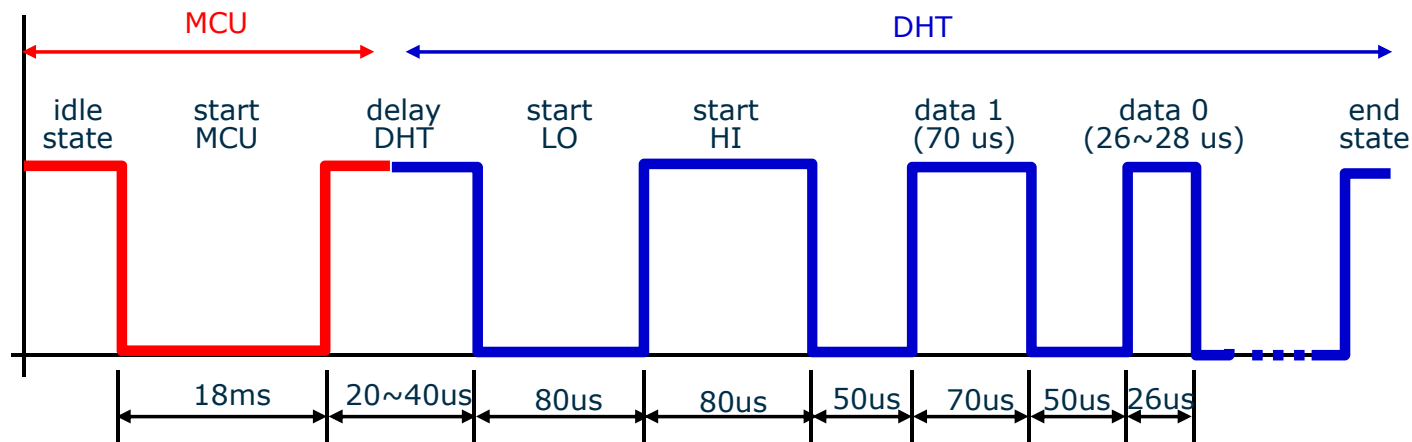
◆ DHT11 Data Transmission

- When MCU sends a start signal, DHT11 changes from the low-power-consumption mode to the running-mode, waiting for MCU completing the start signal. Once it is completed, DHT11 sends a response signal of **40-bit data** that include the relative humidity and temperature information to MCU.
- When the communication between MCU and DHT11 begins, the program of MCU will set Data Single-bus voltage level from high to low and this process must take at least 18ms to ensure DHT's detection of MCU's signal, then MCU will pull up voltage and wait 20-40us for DHT's response.
- Once DHT detects the start signal, it will send out a low-voltage-level response signal, which lasts 80us.
- Then the program of DHT sets Data Single-bus voltage level from low to high and keeps it for 80us for DHT's preparation for sending data.

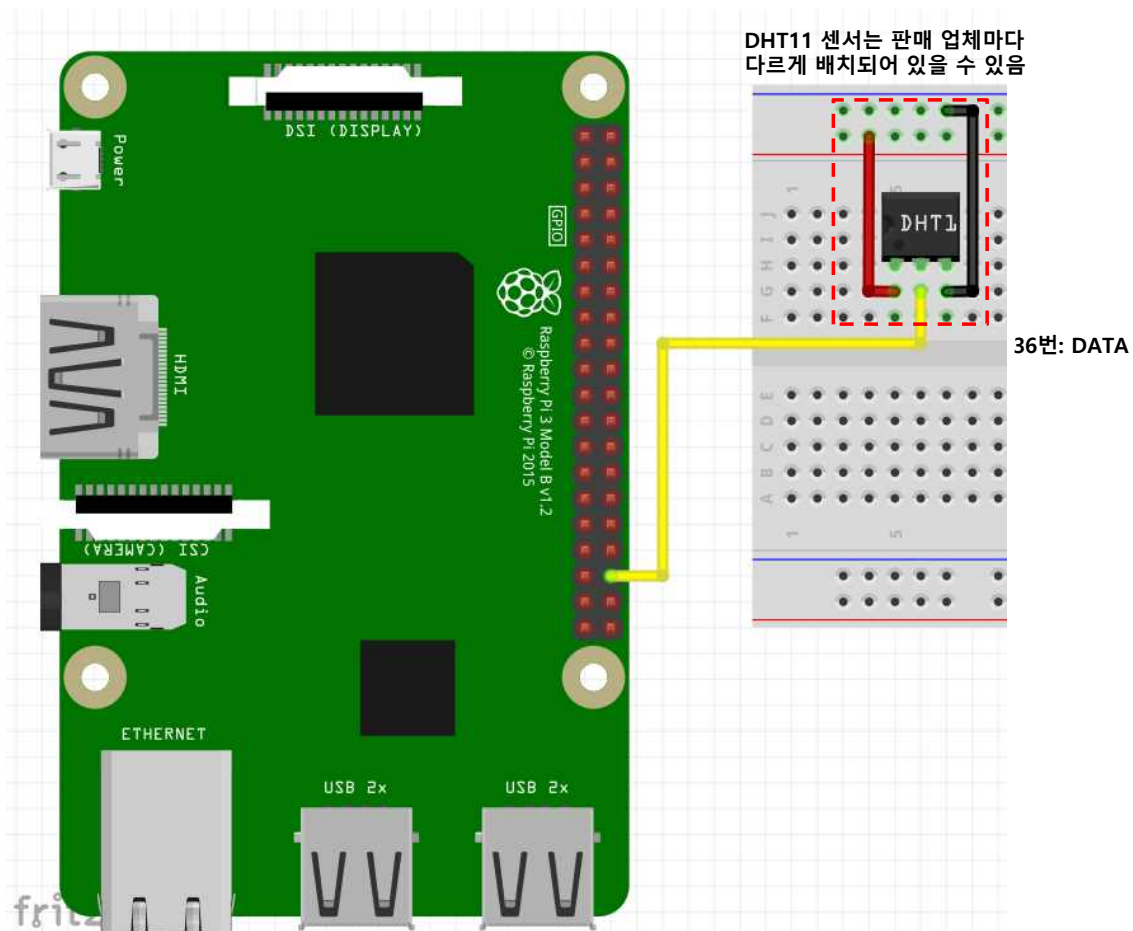
DHT11 Data Transmission

◆ DHT11 Data Transmission

- When DATA Single-Bus is at the low voltage level, this means that DHT is sending the response signal. Once DHT sent out the response signal, it pulls up voltage and keeps it for 80us and prepares for data transmission.
- When DHT is sending data to MCU, every bit of data begins with the 50us low-voltage-level and the length of the following high-voltage-level signal determines whether data bit is "0" or "1"



DHT11 Temperature Sensor



DHT11 Temperature Sensor with RPi.GPIO

```
# class DHT11 (1)
import RPi.GPIO as GPIO
import time
```

```
DHT11_PIN = 36
DEBUG_PIN = 38
```

```
class DHT11:
```

```
    def __init__(self):
```

```
        GPIO.setmode (GPIO.BOARD)
        GPIO.setup (DHT11_PIN, GPIO.OUT)
        GPIO.setup (DEBUG_PIN, GPIO.OUT)
```

```
    def read (self):
```

```
        GPIO.setup (DHT11_PIN, GPIO.OUT)
        GPIO.output (DHT11_PIN, GPIO.HIGH)
        time.sleep (1)
        GPIO.output (DHT11_PIN, GPIO.LOW)
        time.sleep (0.02)
        GPIO.output (DHT11_PIN, GPIO.HIGH)
        count = 0
        while count < 35:
            count += 1
            GPIO.setup (DHT11_PIN, GPIO.IN)
            while GPIO.input (DHT11_PIN) == GPIO.LOW:
                break
            while GPIO.input (DHT11_PIN) == GPIO.HIGH:
                break
            while GPIO.input (DHT11_PIN) == GPIO.LOW:
                break
```

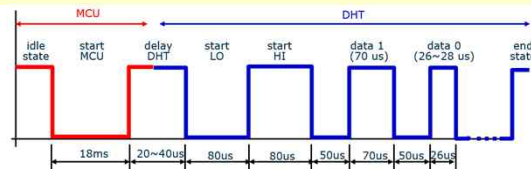
```
DATA = [0, 0, 0, 0, 0] # 5 x 8 = 40bit data
count = 0 # bit count
while count <= 40:
    i = 0
    while GPIO.input (DHT11_PIN) == GPIO.LOW:
        i += 1
    while GPIO.input (DHT11_PIN) == GPIO.HIGH:
        i += 1
    if (count > 0):
        GPIO.output (DEBUG_PIN, 1)
        GPIO.output (DEBUG_PIN, 0)

    DATA[int ((count - 1) / 8)] <= 1
    if i > 37: # if duration is more than 76us
        DATA[int ((count - 1) / 8)] |= 1
    count += 1

    if ((DATA[0] + DATA[1] + DATA[2] + DATA[3])\
        % 256 == DATA[4]):
        return DATA
    else:
        print ("checksum error")
        return None
```

```
if __name__ == "__main__":
```

```
    dht11 = DHT11 ()
    while True:
        print (dht11.read())
        time.sleep (0.1)
```



Result of Temperature Reading

```
root@raspberrypi:~/python# vim dht11.py
root@raspberrypi:~/python# python3 dht11.py
dht11.py:10: RuntimeWarning: This channel is already in use
GPIO.setup (DHT11_PIN, GPIO.OUT)
dht11.py:11: RuntimeWarning: This channel is already in use
GPIO.setup (DEBUG_PIN, GPIO.OUT)
hum=19.0%, temp=24.0 deg
hum=19.0%, temp=24.0 deg
hum=19.0%, temp=24.0 deg
hum=19.0%, temp=24.0 deg
hum=19.0%, temp=24.0 deg
hum=19.0%, temp=24.0 deg
hum=19.0%, temp=24.0 deg
hum=19.0%, temp=24.0 deg
hum=19.0%, temp=24.0 deg
hum=19.0%, temp=24.0 deg
hum=19.0%, temp=24.0 deg
hum=18.0%, temp=25.0 deg
hum=18.0%, temp=25.0 deg
hum=18.0%, temp=25.0 deg
checksum error
hum=18.0%, temp=25.0 deg
hum=18.0%, temp=25.0 deg
```

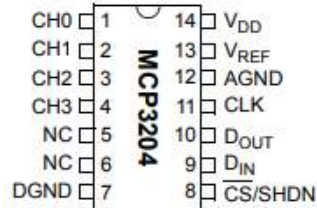

Analog device interfaces with `spidev` module

MCP3208 ADC

◆ 8 Channel 12-Bit A/D Converter with SPI (serial peripheral Interface)

Package Types

PDIP, SOIC, TSSOP



PDIP, SOIC

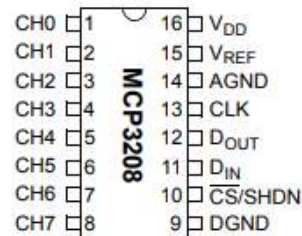


TABLE 5-2: CONFIGURATION BITS FOR THE MCP3208

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
0	0	0	0	differential	CH0 = IN+ CH1 = IN-
0	0	0	1	differential	CH0 = IN- CH1 = IN+
0	0	1	0	differential	CH2 = IN+ CH3 = IN-
0	0	1	1	differential	CH2 = IN- CH3 = IN+
0	1	0	0	differential	CH4 = IN+ CH5 = IN-
0	1	0	1	differential	CH4 = IN- CH5 = IN+
0	1	1	0	differential	CH6 = IN+ CH7 = IN-
0	1	1	1	differential	CH6 = IN- CH7 = IN+

MCP3208 ADC

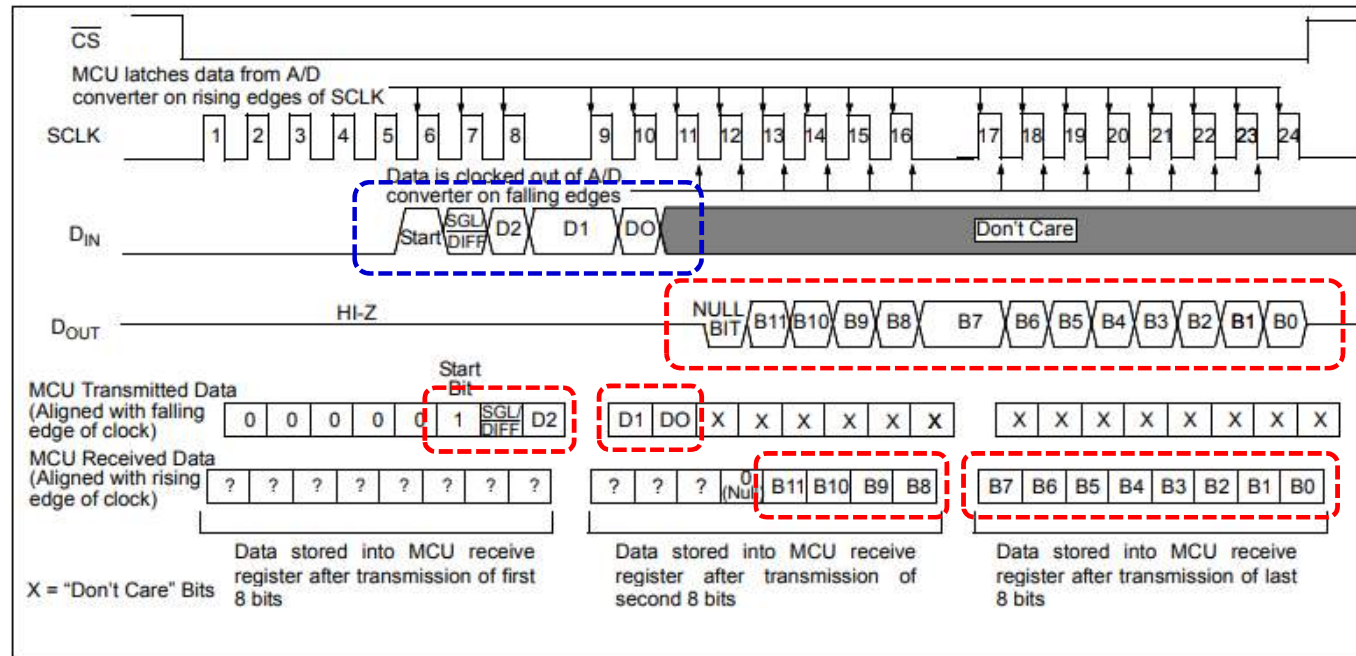


FIGURE 6-1: SPI Communication using 8-bit segments (Mode 0,0: SCLK idles low).

MCP4922 DAC

◆ 12-Bit Dual Voltage Output Digital-to-Analog Converter with SPI Interface

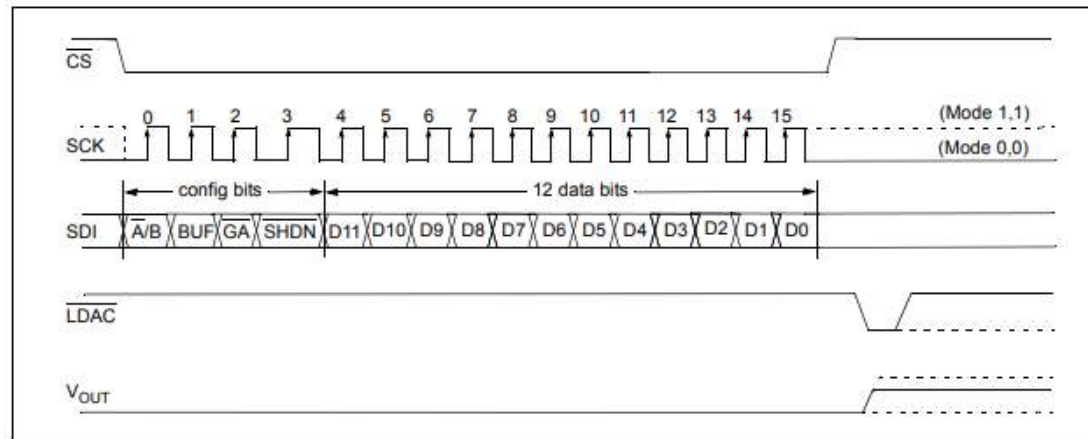
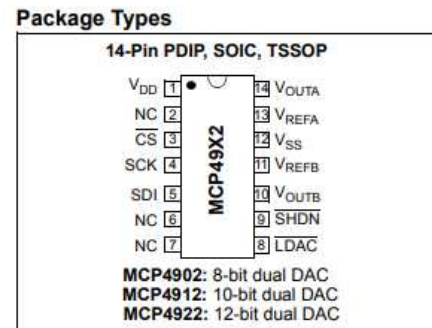


FIGURE 5-1: Write Command for MCP4922 (12-bit DAC).

Analog Interface with spidev module

◆ spidev

- source: <https://pypi.org/project/spidev/>
- install: \$sudo apt-get install spidev (latest version: June 2, 2020)

◆ spidev methods

spidev method	description
SpiDev()	creates a spidev object
open(bus, device)	connect to the specified SPI device, opening /dev/spidev/<bus>, <device>
readbytes(n)	read n bytes from SPI device
writebytes(list_values) writebytes2(list_values)	write a list of values to SPI device
xfer(list_values[, speed_hz, delay_usex, bits_per_word])	perform an SPI transaction
close()	disconnects from the SPI device

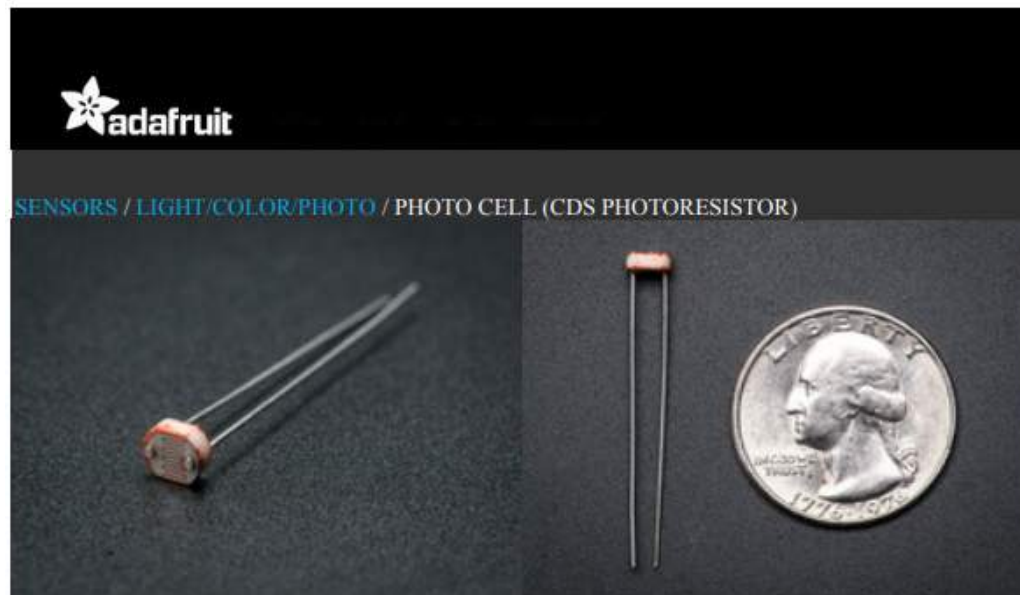
```
# example usage of spidev
import spidev

bus = 0
device = 0
spi = spidev.SpiDev()
spi.open(bus, device)
to_send = [0x01, 0x02, 0x03]
spi.xfer(to_send)
```

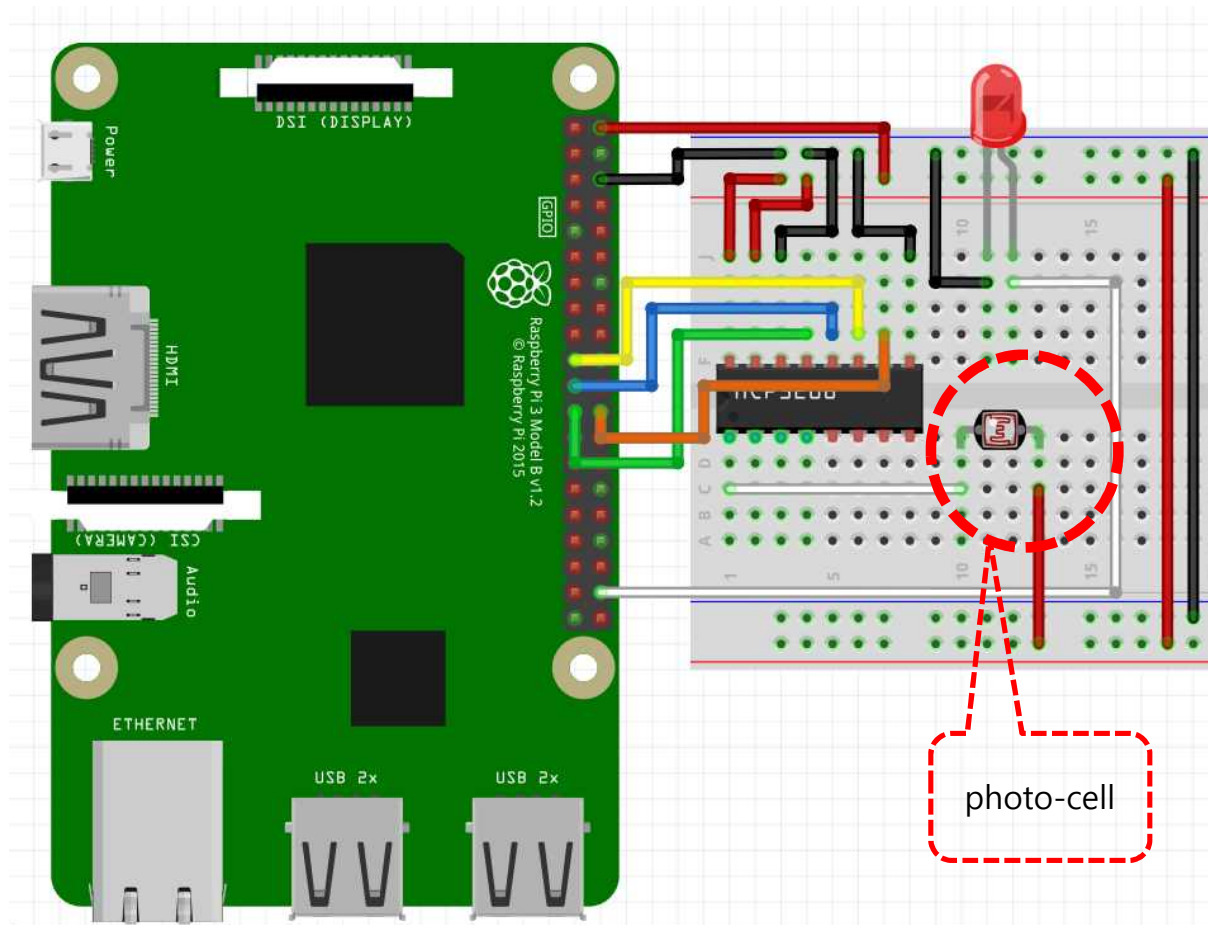
조도센서 (Photo Sensor)

◆ Adafruit photocell

- 조도센서, 빛의 밝기에 따라 저항의 크기가 결정되고 이를 아날로그 값으로 변환하여 반환하는 센서
- 저항의 값이 크면 밝은 것이고, 그렇지 않으면 어두운 것이다.



회로도



2번: VCC
6번: GND
17번: MOSI
19번: MISO
21번: CLK
22번: CS
38번: LED

LED Control with Photo Sensor and spidev

```
# LED control with photo sensor connected by spidev (1)
import time
import RPi.GPIO as GPIO
import spidev
```

```
spi = spidev.SpiDev ()
spi.open (0,0)
spi.max_speed_hz = 500000
```

```
MCP3208_CS_PIN = 13
LED_PIN = 38
```

```
def ADC_DAC_Init ():
    GPIO.setmode (GPIO.BOARD)
    GPIO.setup (MCP3208_CS_PIN, GPIO.OUT)
    GPIO.setup (LED_PIN, GPIO.OUT)
```

```
def MCP3208_read (channel):
    chD2 = (channel & 0x04) >> 2
    chD1 = (channel & 0x02) >> 1
    chD0 = (channel & 0x01)
```

```
octet_0 = 0x01 << 2 # MCP3208_SB
octet_0 |= 0x01 << 1 # MCP3208_SD
octet_0 |= chD2 << 0 # MCP3208_D2
```

```
octet_1 = chD1 << 7 # MCP3208_D1
octet_1 |= chD0 << 6 # MCP3208_D0
```

```
octet_2 = 0x00;
octets = [octet_0, octet_1, octet_2]
```

```
GPIO.output (MCP3208_CS_PIN, GPIO.LOW)
adc_0, adc_1, adc_2 = spi.xfer (octets)
GPIO.output (MCP3208_CS_PIN, GPIO.HIGH)
```

```
adc = adc_2;
adc |= (adc_1 & 0x0F) << 0x08
```

```
return adc
```

```
# LED control with photo sensor connected by spidev (2)
```

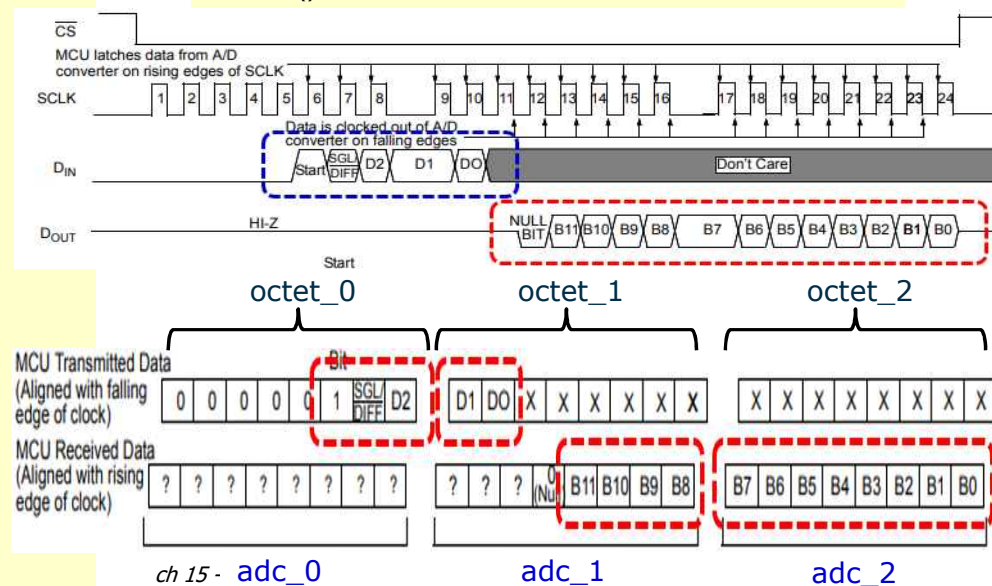
```
def main ():
```

```
    ADC_DAC_Init ()
    pwm = GPIO.PWM (LED_PIN, 100)
    pwm.start (0)
```

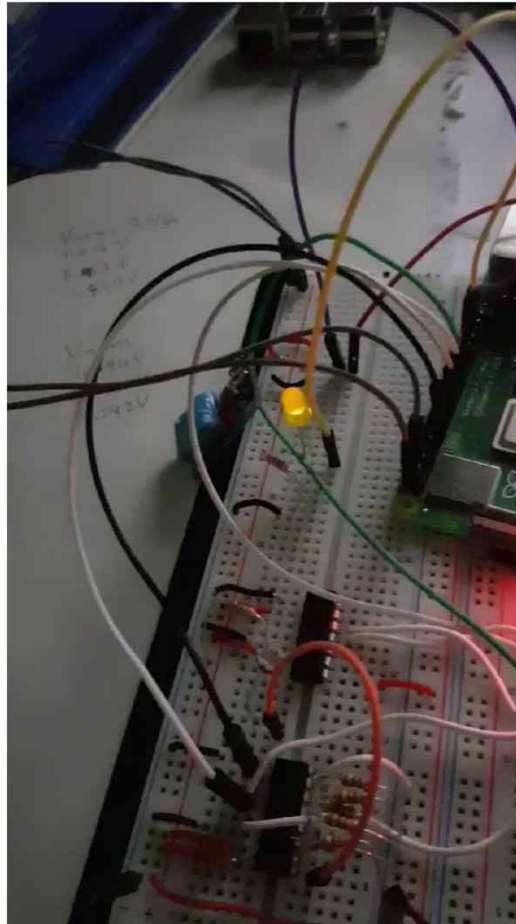
```
    while True:
```

```
        photo_val = MCP3208_read (0)
        #print ("{}", {}).format (4096 - photo_val, photo_val))
        light = float (4096 - photo_val)
        if light > 100:
            light = 100
        pwm.ChangeDutyCycle (light)
        time.sleep (0.1)
```

```
if __name__ == "__main__":
    main ()
```



결과

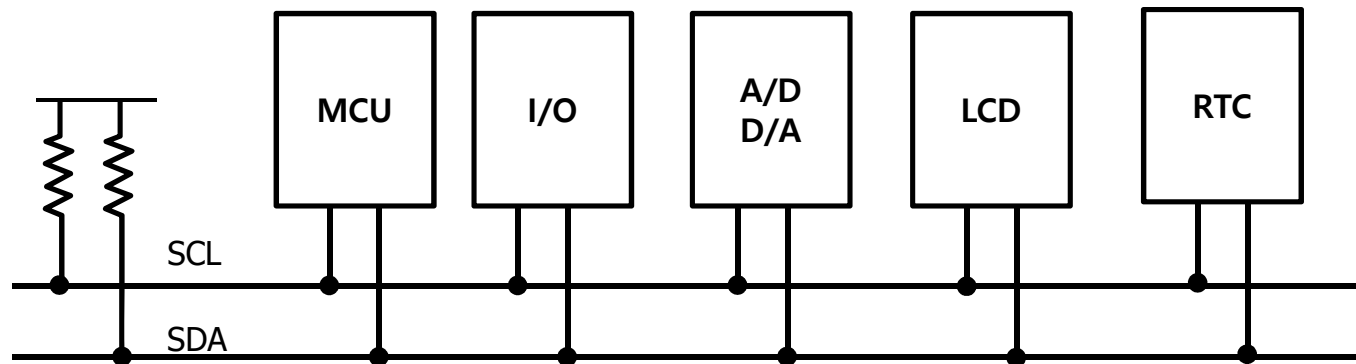


I²C (Inter-Integrated Circuit)

I²C

◆ What is I²C ?

- I²C (Inter-Integrated Circuit) is a multi-master, multi-slave serial computer bus, invented by Philips Semiconductor (now NXP Semi-conductions).
- I²C is typically used for attaching lower-speed peripheral ICs to processors and microcontrollers
- I²C is a synchronous serial communication using two wires, one wire for data (SDA) and the other wire for clock (SCL)



I²C

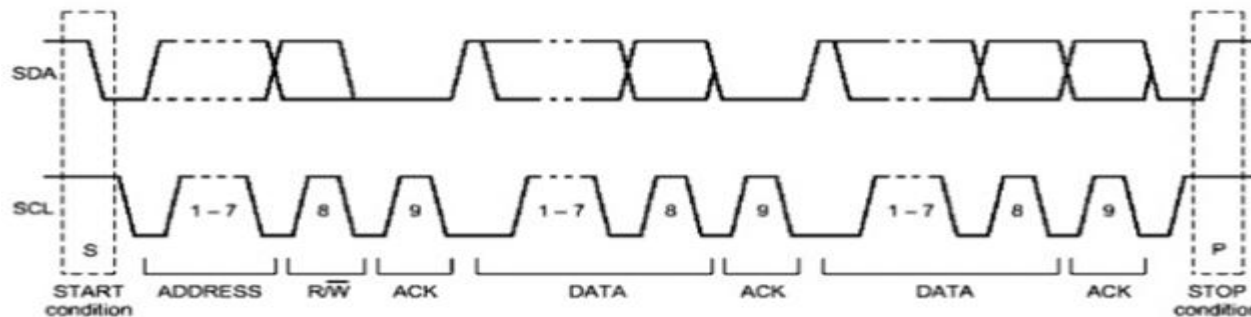
◆ Features of I²C

- I²C supports a wide range of voltage levels, hence the circuit board can provide +5 volts, or +3.3 volts as Vcc easily, and other lower/higher voltages as well.
- this provides a wide range of choices for the values of the pull-up resistors. Anything within the range of 1K to 47K should be fine, however values lower than 10K are usually preferred
- I²C supports serial 8-bit data transfers up to a speed of 100kbps, which is the standard clock speed of SCL. However, I2C can also operate at higher speed – Fast mode (400Kbps) and High Speed Mode (3.4Mbps)
- Maximum number of nodes is 112, as address is 7-bits and there are 16-nodes reserved
- I²C is used for short distance communication
- the slave is allowed to hold the clock line low until it is ready to give out the results. The master must wait for the clock to go back to high before continuing with its work, this is called clock stretching

I²C Frame Format

◆ I²C Frame Format

- Start bit – 1 bit indicates the start of a new frame, always logic low
- address – 7-bits that decides slave address which is the device that will process the upcoming sent data by the master
- R/W* - 1 bit to decide the type of operation, logic high for read, logic low for write
- ACK – 1 bit sent by the slave as acknowledge by pulling line low
- Data – each 8-bits of data sent is followed by ACK bit by the receiver
- Stop bit – 1-bit indicates the end of the frame



RaspberryPI I²C

◆ Raspberry PI Pin Layout

- I2C_1 SDA
- I2C_1 SCL



smbus2

◆ smbus2

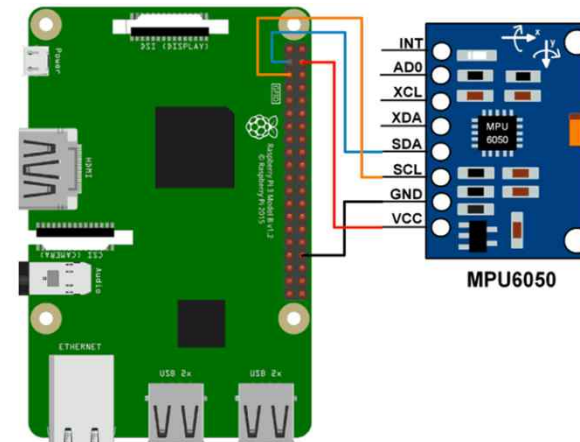
- <https://pypi.org/project/smbus2/>
- **\$sudo apt-get install smbus2**
- smbus2 is (yet another) pure Python implementation of the [python-smbus](#) package.
- It was designed from the ground up with two goals in mind:
 - It should be a drop-in replacement of smbus. The syntax shall be the same.
 - Use the inherent i2c structs and unions to a greater extent than other pure Python implementations like [pysmbus](#) does. By doing so, it will be more feature complete and easier to extend.
- Currently supported features are:
 - Get i2c capabilities (I2C_FUNCS)
 - SMBus Packet Error Checking (PEC) support
 - read_byte, write_byte, read_byte_data, write_byte_data
 - read_word_data, write_word_data
 - read_i2c_block_data, write_i2c_block_data
 - write_quick
 - process_call
 - read_block_data, write_block_data
 - block_process_call
 - i2c_rdwr - *combined write/read transactions with repeated start*



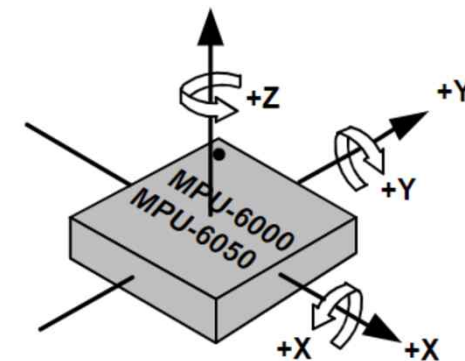
6-Axis Gyro Accelerator Sensor Interface

◆ MPU6050 (Accelerometer + Gyroscope)

- MPU6050 Sensor module is an integrated 6-axis Motion tracking device.
- It has a **3-axis Gyroscope**, **3-axis Accelerometer**, Digital Motion Processor and a Temperature sensor, all in a single IC.
- It can accept inputs from other sensors like 3-axis magnetometer or pressure sensor using its Auxiliary I2C bus.
- If external 3-axis magnetometer is connected, it can provide complete 9-axis Motion Fusion output.
- A microcontroller can communicate with this module using I2C communication protocol. Various parameters can be found by reading values from addresses of certain registers using I2C communication.
- Gyroscope and accelerometer reading along X, Y and Z axes are available in 2's complement form.
- Gyroscope readings are in degrees per second (dps) unit; Accelerometer readings are in g unit.



MPU6050 Interfacing with Raspberry PI



Orientation of Axes of Sensitivity and Polarity of Rotation

MPU6050 (Accelerometer+Gyroscope) Interfacing with Raspberry Pi

◆ **Source:** <https://www.electronicwings.com/raspberry-pi/mpu6050-accelerometergyroscope-interfacing-with-raspberry-pi>

```
# Read Gyro and Accelerometer by Interfacing Raspberry Pi with MPU6050 using Python (1)
```

```
import smbus #import SMBus module for I2C  
from time import sleep #import
```

```
#MPU6050 Registers and their Address
```

```
bus = smbus.SMBus(1) # or bus = smbus.SMBus(0) for older version boards  
Device_Address = 0x68
```

```
PWR_MGMT_1 = 0x6B # power management
```

```
SMPLRT_DIV = 0x19 # sample rate register
```

```
CONFIG = 0x1A # configuration
```

```
INT_ENABLE = 0x38 # interrupt enable
```

```
ACCEL_XOUT_H = 0x3B
```

```
ACCEL_YOUT_H = 0x3D
```

```
ACCEL_ZOUT_H = 0x3F
```

```
GYRO_XOUT_H = 0x43
```

```
GYRO_YOUT_H = 0x45
```

```
GYRO_ZOUT_H = 0x47
```



Read Gyro and Accelerometer by Interfacing Raspberry Pi with MPU6050 using Python (2)

def MPU_Init():

```
# write to sample rate register
bus.write_byte_data(Device_Address, SMPLRT_DIV, 7)
# write to power management register
bus.write_byte_data(Device_Address, PWR_MGMT_1, 1)
# write to Configuration register
bus.write_byte_data(Device_Address, CONFIG, 0)
# write to Gyro configuration register
bus.write_byte_data(Device_Address, GYRO_CONFIG, 24)
# write to interrupt enable register
bus.write_byte_data(Device_Address, INT_ENABLE, 1)
```

def read_raw_data(addr):

```
#Accelerometer and Gyro value are 16-bit
high = bus.read_byte_data(Device_Address, addr)
low = bus.read_byte_data(Device_Address, addr+1)

# concatenate higher and lower value
value = ((high << 8) | low)

# to get signed value from mpu6050
if (value > 32768):
    value = value - 65536
return value
```



Read Gyro and Accelerometer by Interfacing Raspberry Pi with MPU6050 using Python (3)

MPU_Init()

print (" Reading Data of Gyroscope and Accelerometer")

while True:

 # Read Accelerometer raw_value

 acc_x = read_raw_data(ACCEL_XOUT_H)

 acc_y = read_raw_data(ACCEL_YOUT_H)

 acc_z = read_raw_data(ACCEL_ZOUT_H)

 # Read Gyroscope raw value

 gyro_x = read_raw_data(GYRO_XOUT_H)

 gyro_y = read_raw_data(GYRO_YOUT_H)

 gyro_z = read_raw_data(GYRO_ZOUT_H)

 # Full scale range +/- 250 degree/C as per sensitivity scale factor

 Ax = acc_x / 16384.0

 Ay = acc_y / 16384.0

 Az = acc_z / 16384.0



Read Gyro and Accelerometer by Interfacing Raspberry Pi with MPU6050 using Python (4)

```
Gx = gyro_x / 131.0
Gy = gyro_y / 131.0
Gz = gyro_z / 131.0

print ("Gx=%.2f"%Gx, u'\u00b0'+ "/s", "\tGy=%.2f" %Gy, u'\u00b0'+ "/s", "\tGz=%.2f" %Gz,\
      u'\u00b0'+ "/s", "\tAx=%.2f g" %Ax, "\tAy=%.2f g" %Ay, "\tAz=%.2f g" %Az)
sleep(1)
```

```
Gx=-0.305 °/s Gy=0.099 °/s Gz=-0.038 °/s Ax=-0.059 g Ay=-0.026 g Az=0.945 g
Gx=-0.328 °/s Gy=0.130 °/s Gz=-0.015 °/s Ax=-0.060 g Ay=-0.032 g Az=0.947 g
Gx=-0.359 °/s Gy=0.153 °/s Gz=-0.015 °/s Ax=-0.074 g Ay=-0.018 g Az=0.951 g
Gx=-0.344 °/s Gy=0.092 °/s Gz=-0.015 °/s Ax=-0.057 g Ay=-0.030 g Az=0.946 g
Gx=-0.214 °/s Gy=-0.053 °/s Gz=0.000 °/s Ax=-0.061 g Ay=-0.027 g Az=0.965 g
Gx=-0.305 °/s Gy=0.084 °/s Gz=-0.107 °/s Ax=-0.048 g Ay=-0.024 g Az=0.939 g
Gx=-0.290 °/s Gy=0.092 °/s Gz=-0.069 °/s Ax=-0.061 g Ay=-0.041 g Az=0.938 g
Gx=-0.298 °/s Gy=0.107 °/s Gz=-0.053 °/s Ax=-0.070 g Ay=-0.031 g Az=0.947 g
Gx=-0.305 °/s Gy=0.107 °/s Gz=-0.084 °/s Ax=-0.053 g Ay=-0.016 g Az=0.951 g
Gx=-0.321 °/s Gy=0.099 °/s Gz=-0.053 °/s Ax=-0.070 g Ay=-0.027 g Az=0.931 g
Gx=-0.305 °/s Gy=0.107 °/s Gz=-0.053 °/s Ax=-0.063 g Ay=-0.041 g Az=0.948 g
Gx=-0.244 °/s Gy=0.076 °/s Gz=-0.076 °/s Ax=-0.058 g Ay=-0.037 g Az=0.969 g
Gx=-0.290 °/s Gy=0.107 °/s Gz=-0.053 °/s Ax=-0.056 g Ay=-0.027 g Az=0.946 g
Gx=-0.305 °/s Gy=0.099 °/s Gz=-0.053 °/s Ax=-0.064 g Ay=-0.030 g Az=0.952 g
Gx=-0.305 °/s Gy=0.099 °/s Gz=-0.038 °/s Ax=-0.060 g Ay=-0.028 g Az=0.939 g
Gx=-0.313 °/s Gy=0.115 °/s Gz=-0.046 °/s Ax=-0.065 g Ay=-0.032 g Az=0.950 g
```



GPS (Global Positioning System)

Neo-6M GPS Sensor

◆ Global Positioning System (GPS) Sensor

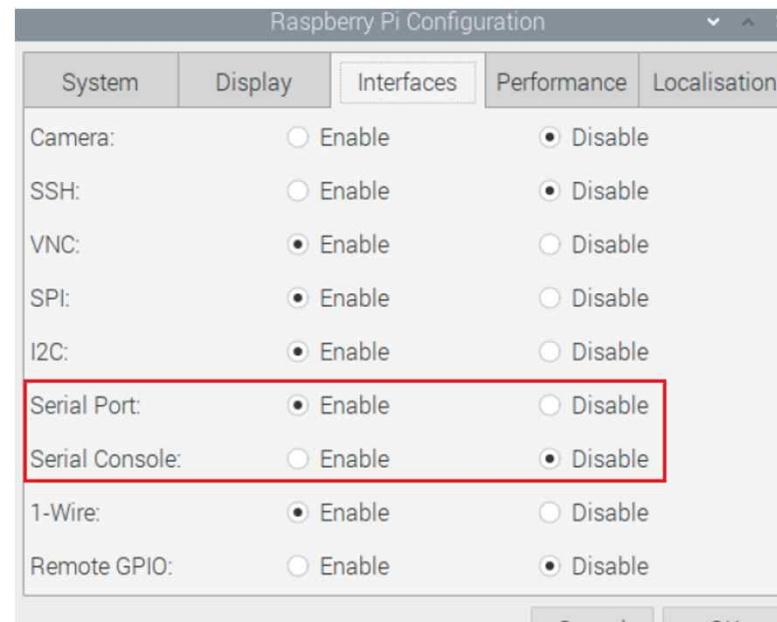
- GPS modules NEO-6M, 3V-5V power supply Universal
- Model: GY-GPS6MV2
- Destined module with ceramic antenna, signal super
- Save the configuration parameter data EEPROM Down
- With data backup battery
- LED signal indicator
- Antenna size 25 * 25mm
- Module size 25mm * 35mm
- PCB size 36mm*26mm
- Installation aperture 3mm
- Serial communication - Default Baud Rate: 9600



Raspberry Setting for GPS Sensor (1)

◆ Raspberry Setting for GPS Sensor

- Preference -> Raspberry Pi Configuration -> interfaces
- Serial Port : Enable, Serial Console : Disable



Raspberry Setting for GPS Sensor (2)

◆ Raspberry Setting for GPS Sensor

- \$ sudo raspi-config 입력 후 Interface Option -> Serial Port

Login shell : No, Serial Port enable : Yes

```
Raspberry Pi Software Configuration Tool (raspi-config)

1 System Options      Configure system settings
2 Display Options     Configure display settings
3 Interface Options   Configure connections to peripherals
4 Performance Options Configure performance settings
5 Localisation Options Configure language and regional settings
6 Advanced Options    Configure advanced settings
8 Update              Update this tool to the latest version
9 About raspi-config  Information about this configuration tool

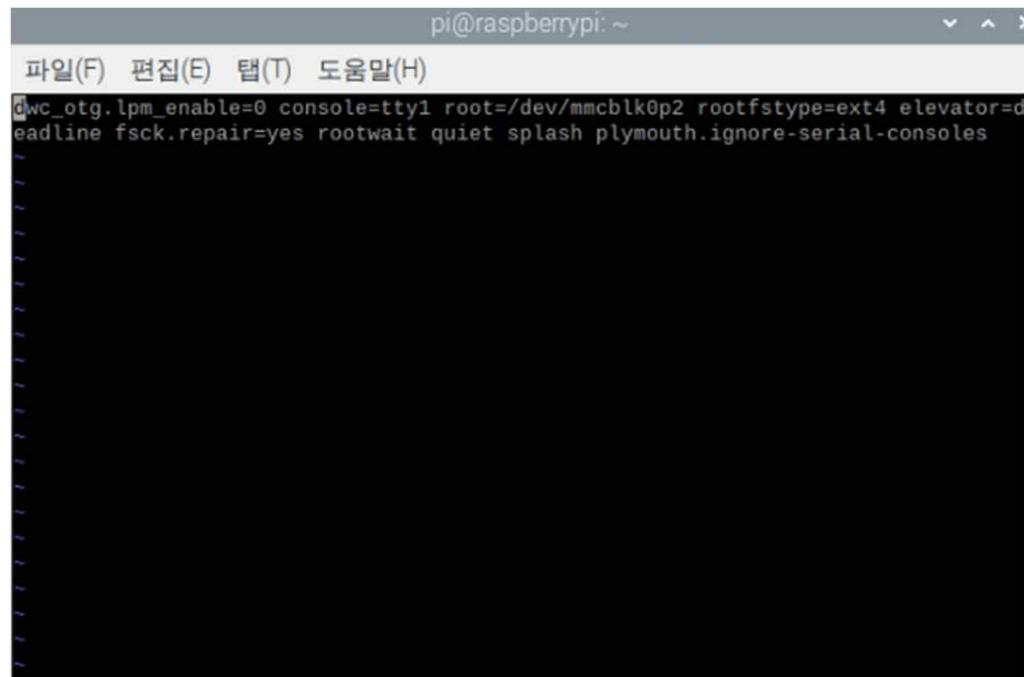
<Select>              <Finish>
```


Raspberry Setting for GPS Sensor (3)

◆ Raspberry Setting for GPS Sensor

\$ sudo vi /boot/cmdline.txt

- dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait quiet splash plymouth.ignore-serial-consoles

A screenshot of a terminal window titled 'pi@raspberrypi: ~'. The window shows the contents of the file /boot/cmdline.txt. The text is: 'dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait quiet splash plymouth.ignore-serial-consoles'. The terminal has a menu bar with '파일(F)', '편집(E)', '탭(T)', and '도움말(H)'. The text is displayed on a black background with white characters.

Raspberry Setting for GPS Sensor (4)

◆ Raspberry Setting for GPS Sensor

\$ dtoverlay -a | grep uart # uart에 해당하는 dtoverlay정보 출력 (dtoverlay: device tree overlay)

\$ dtoverlay -h uart5 # uart5에 해당하는 정보 상세 출력

```
pi@raspberrypi:~$ dtoverlay -a | grep uart
midi-uart0
midi-uart1
miniuart-bt
uart0
uart1
uart2
uart3
uart4
uart5
pi@raspberrypi:~$ dtoverlay -h uart5
Name: uart5

Info:  Enable uart 5 on GPIOs 12-15. BCM2711 only.

Usage:  dtoverlay=uart5,<param>

Params: ctsrts                                Enable CTS/RTS on GPIOs 14-15 (default off)
```

Raspberry Setting for GPS Sensor (5)

◆ Raspberry Setting for GPS Sensor

\$ sudo cp /boot/config.txt /boot/config.org # 원본 파일 백업

\$ sudo vi /boot/config.txt

- [all] 항목에 dtoverlay=uart5 추가 (uart5에 인터페이스 등록)
- [all] 항목에 enable_uart=1 추가 (miniUART 활성화)
- 이후 Raspberry Pi 재부팅

```
[all]
#dtoverlay=vc4-fkms-v3d
dtoverlay=w1-gpio
dtoverlay=uart5
enable_uart=1
```

GPS 관련 패키지 설치 및 확인

◆ Raspberry Setting for GPS Sensor

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

```
$ sudo apt-get install python3-pip
```

```
$ sudo pip3 install pyserial (시리얼 통신 관련 라이브러리)
```

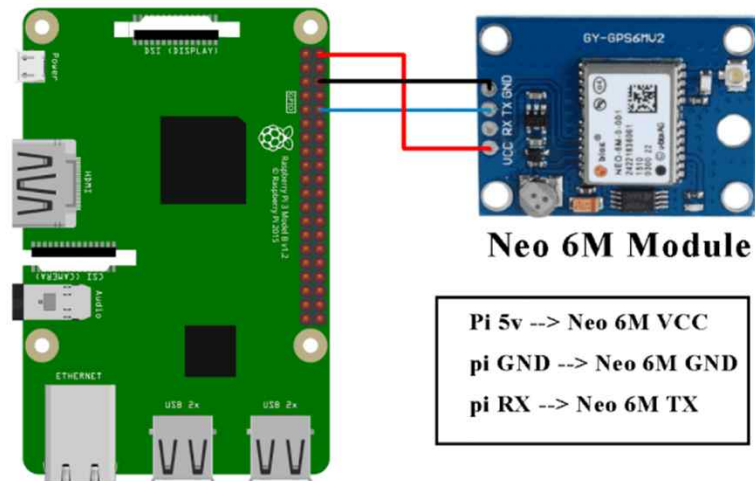
```
$ sudo apt-get install gpsd gpsd-clients (GPS 데이터 수집 및 클라이언트 애플리케이션 프로그램)
```



Neo-6M 연결

◆ GPS (Neo-6M) 연결

- GPIO Board 기준
- VCC -> 2번 (5V)
- GND -> 6번
- TX -> 10번 (Raspberry UART rx)
- RX -> 8번 (Raspberry UART tx)



GSP 데이터 확인(1)

◆ GPS 연결 확인

- \$ ls -l /dev 명령어로 Neo-6m이 정상적으로 연결되어 있는지 확인.
- 다음과 같이 출력되면 정상적으로 연결 성공

```
brw-rw---- 1 root disk      1,  9 May 18 22:57 ram9
crw-rw-rw- 1 root root       1,  8 May 18 22:57 random
drwxr-xr-x 2 root root      60 Jan  1 1970 raw
crw-rw-r-- 1 root netdev   10, 242 May 18 22:57 rfkill
crw-rw---- 1 root video  239,  0 May 18 22:57 rpidvid-h264mem
crw-rw---- 1 root video  241,  0 May 18 22:57 rpidvid-hevcmem
crw-rw---- 1 root video  240,  0 May 18 22:57 rpidvid-intcmem
crw-rw---- 1 root video  238,  0 May 18 22:57 rpidvid-vp9mem
lrwxrwxrwx 1 root root       5 May 18 22:57 serial0 -> ttyS0
lrwxrwxrwx 1 root root       7 May 18 22:57 serial1 -> ttyAMA0
drwxrwxrwt 2 root root      40 Feb 14 2019 shm
drwxr-xr-x 3 root root     140 May 18 22:57 snd
crw-rw---- 1 root spi     153,  0 May 18 22:57 spidev0.0
crw-rw---- 1 root spi     153,  1 May 18 22:57 spidev0.1
```

GSP 데이터 확인(2)

◆ GPS 데이터 확인

- \$ sudo vi /etc/default/gpsd 명령으로 gpsd 관련 설정 파일 수정
- START_DAEMON="true" # 시스템 작동 시 자동으로 gpsd 데몬 실행
- DEVICES="/dev/serial0" # GPS와 연결된 UART 인터페이스 지정
- GPSD_OPTIONS="-F /var/run/gpsd.sock" # gpsd제어 소켓 생성
- 이후 Raspberry Pi 재부팅

```
pi@raspberrypi:~ $ cat /etc/default/gpsd
# Default settings for the gpsd init script and the hotplug wrapper.

# Start the gpsd daemon automatically at boot time
START_DAEMON="true"

# Use USB hotplugging to add new USB devices automatically to the daemon
USB_AUTO="true"

# Devices gpsd should collect to at boot time.
# They need to be read/writeable, either by user gpsd or the group dialout.
DEVICES="/dev/serial0"

# Other options you want to pass to gpsd
GPSD_OPTIONS="-F /var/run/gpsd.sock"
```

GSP 데이터 확인(3)

◆ GPS 데이터 확인

- Neo-6M 모듈 연결 후 LED가 점멸할 때 까지 대기
- 만약 점멸하지 않는다면 창문 근처, 혹은 야외로 이동
- \$sudo cgps -s 명령어로 GPS 데이터가 수신되고 있음을 확인.

```
pi@raspberrypi: ~
파일(F) 편집(E) 탭(T) 도움말(H)

Time: n/a
Latitude: 35.83915350 N
Longitude: 128.75341433 E
Altitude: n/a
Speed: n/a
Heading: n/a
Climb: n/a
Status: 2D FIX (6 secs)
Longitude Err: +/- 25 m
Latitude Err: +/- 25 m
Altitude Err: n/a
Course Err: n/a
Speed Err: n/a
Time offset: n/a
Grid Square: PM45ju

PRN: Elev: Azim: SNR: Used:
1 55 194 28 Y
7 52 254 35 Y
8 54 039 28 Y
16 20 114 25 Y
3 00 169 00 N
14 14 308 00 N
17 06 259 00 N
21 80 127 17 N
22 16 153 00 N
22 16 153 00 N
27 25 060 26 N
27 25 060 25 N
```

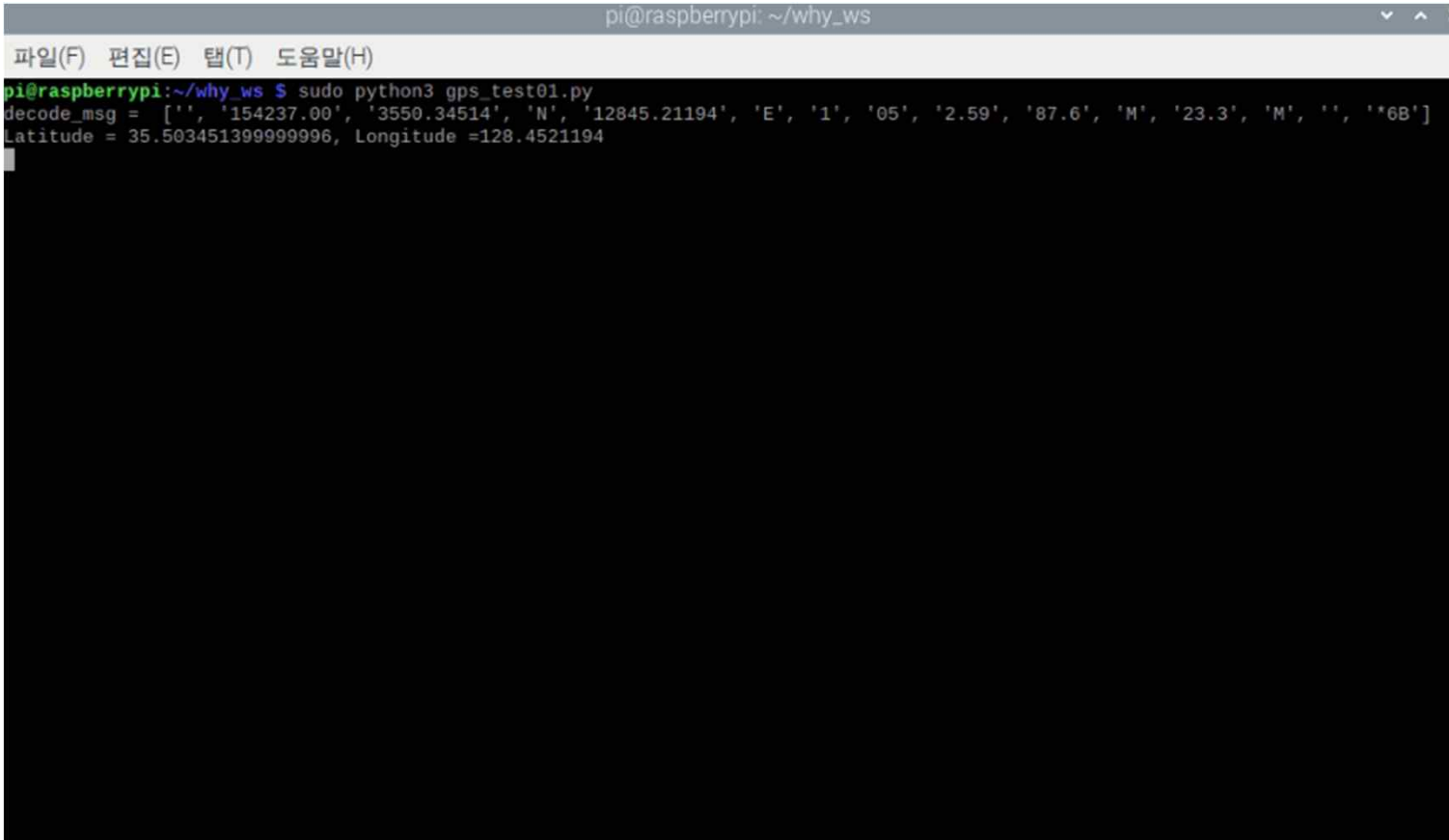

Python Program for Neo-6M (1)

```
#Neo_6M_Control(1)
import serial
import time
import string

while True:
    ser=serial.Serial("/dev/serial0", 9600, timeout=0.5)
    newdata=ser.readline()
    if newdata[0:6]==b"$GPGGA":
        try:
            decode_msg=newdata[6:].decode(errors='ignore').replace("\r\n","").split(",")
            lat=float(decode_msg[2])
            lng=float(decode_msg[4])
            print("Latitude = {}, Longitude ={} ".format(lat/100, lng/100))
        except ValueError:
            pass
        except IndexError:
            pass
```



Neo-6m with Python(2)



```
pi@raspberrypi: ~/why_ws
파일(F) 편집(E) 탭(T) 도움말(H)
pi@raspberrypi:~/why_ws $ sudo python3 gps_test01.py
decode_msg = ['', '154237.00', '3550.34514', 'N', '12845.21194', 'E', '1', '05', '2.59', '87.6', 'M', '23.3', 'M', '', '6B']
Latitude = 35.503451399999996, Longitude =128.4521194
```

Distance Measurement with TF mini-LiDAR

TF mini-LiDAR

◆ LiDAR (light detection and ranging)

◆ TF mini-LiDAR

- Voltage Range: $5V \pm 0.1V$
- Communication Interface: UART / I2C
- Working Range: 0.1m-12m @90% Reflectivity
- Average Power: 0.6W
- Acceptance Angle: 2°
- Minimum Resolution: 1mm
- Refresh Frequency: 100Hz
- 거리 측정 정확도 (Ranging Accuracy): $\pm 6\text{cm}@ (0.1-6\text{m}) \pm 1\% @ (6\text{m}-12\text{m})$
- 거리 측정 단위 : mm
- Band: 850nm
- Size: 1.65in x 0.59in x 0.63in / 42mm x 15mm x 16mm
- Operating Temperature: $0^\circ\text{C} \sim 60^\circ\text{C}$
- Storage Temperature: $-20^\circ\text{C} \sim 75^\circ\text{C}$
- Anti-light Environment: 70Klux
- Weight: 4.7g



TF-mini-LiDAR 센서 연결(1)

◆ TF-mini-LiDAR 센서 연결

- USB-to-ttl 케이블과 연결
- TF-mini LiDAR의 vcc은 케이블의 vcc선에 연결
- TF-mini LiDAR의 gnd은 케이블의 gdn선에 연결
- TF-mini LiDAR의 tx은 케이블의 rx에 연결
- TF-mini LiDAR의 rx은 케이블의 tx에 연결



센서 연결(2)

◆ TF-mini-LiDAR 센서 연결

- USB to ttl 케이블을 Raspberry에 연결후
- ls -l /dev 명령어로 연결이 정상적으로 되었는지 확인

```
pi@raspberrypi: ~/why_ws
파일(F) 편집(E) 탭(T) 도움말(H)
crw--w---- 1 root tty      4, 53 11월 21 00:17 tty53
crw--w---- 1 root tty      4, 54 11월 21 00:17 tty54
crw--w---- 1 root tty      4, 55 11월 21 00:17 tty55
crw--w---- 1 root tty      4, 56 11월 21 00:17 tty56
crw--w---- 1 root tty      4, 57 11월 21 00:17 tty57
crw--w---- 1 root tty      4, 58 11월 21 00:17 tty58
crw--w---- 1 root tty      4, 59 11월 21 00:17 tty59
crw--w---- 1 root tty      4, 6 11월 21 00:17 tty6
crw--w---- 1 root tty      4, 60 11월 21 00:17 tty60
crw--w---- 1 root tty      4, 61 11월 21 00:17 tty61
crw--w---- 1 root tty      4, 62 11월 21 00:17 tty62
crw--w---- 1 root tty      4, 63 11월 21 00:17 tty63
crw--w---- 1 root tty      4, 7 11월 21 00:47 tty7
crw--w---- 1 root tty      4, 8 11월 21 00:17 tty8
crw--w---- 1 root tty      4, 9 11월 21 00:17 tty9
crw--w---- 1 root tty      204, 64 11월 21 00:47 ttyAMA0
crw-rw---- 1 root dialout 204, 65 11월 21 00:17 ttyAMA1
crw-rw---- 1 root dialout 4, 64 11월 21 00:17 ttyS0
crw-rw---- 1 root dialout 188, 0 11월 21 19:47 ttyUSB0
crw----- 1 root root      5, 3 11월 21 00:17 ttyprintk
crw----- 1 root root     10, 239 11월 21 00:17 uhid
crw----- 1 root root     10, 223 11월 21 00:17 uinput
crw-rw-rw- 1 root root      1, 9 11월 21 00:17 urandom
drwxr-xr-x 3 root root      60 11월 21 00:17 v4l
```

TF-mini Lidar with Python

```
#TF-mini LiDAR Control(1)
import serial
import time

def Get_Dist_From_Lidar.ser):
    """
    라이다를 이용한 거리 측정 함수
    """
    while True:
        count=ser.in_waiting
        if count>8:
            recv=ser.read(9)
            ser.reset_input_buffer()
            if recv[0]==0x59 and recv[1]==0x59:
                distance=recv[2]+recv[3]*256
                strength=recv[4]+recv[5]*256
                ser.reset_input_buffer()
                return distance

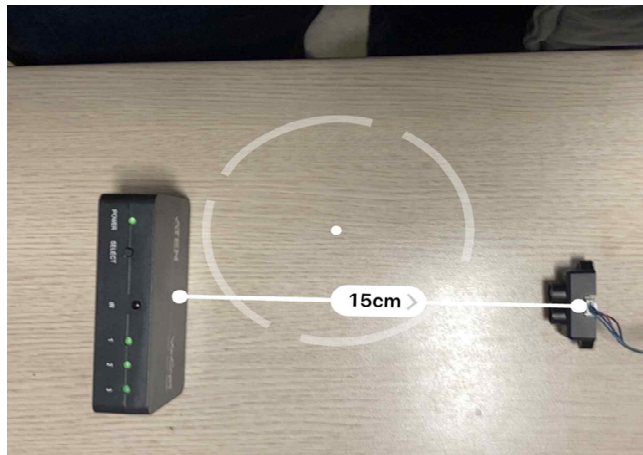
ser=serial.Serial("/dev/ttyUSB0", 115200)
#ser=serial.Serial("/dev/serial0", 115200)
while True:
    try:
        dist=Get_Dist_From_Lidar(ser)
        print("distance = {}".format(dist))
        time.sleep(1)
    except KeyboardInterrupt:
        break
```



TF mini-LiDAR를 사용한 거리 측정

◆ TF LiDAR를 사용한 거리 측정

- mini-LiDAR 앞에 물체를 10 cm 거리 단위로 이동시키며 측정된 거리와 비교하여 거리측정이 정상적으로 이루어진 것을 확인.



```
pi@raspberrypi: ~/why_ws
파일(F) 편집(E) 탭(T) 도움말(H)
pi@raspberrypi:~/why_ws $ python3 tf_mini.py
distance = 15
distance = 15
distance = 15
distance = 15
distance = 15
distance = 15
distance = 15
```


Homework 15

Homework 15

15.1 LEDs On/Off control with push button

- Prepare three color LEDs (red, green, blue) connected to GPIO of Raspberry Pi
- Prepare a push button connected to GPIO of Raspberry Pi
- Prepare a Python program that continuously monitors the push button, and when the push button is pushed down, turns on the color LED of red, green, blue, and then turn off all LEDs, in sequence.

15.2 SG90 servo motor control with two push buttons

- Prepare a SG90 servo motor connected to GPIO of Raspberry Pi
- Prepare two push buttons (right, left) connected to GPIO of Raspberry Pi
- Prepare a Python program that continuously monitors the push buttons, and when the push button is pushed down, rotates the servo motor either to 90°(right) or to -90° (left , while the button is not pushed down, the servo motor is set at 0°

15.3 Step motor control with two push buttons

- Prepare a step motor connected to GPIO of Raspberry Pi
- Prepare two push button connected to GPIO of Raspberry Pi (one for step-forward, one for step-backward)
- Prepare a Python program that continuously monitors the push buttons; when the step-forward push button is pushed down, rotates clockwise the step motor (45°), when the step-backward push button is pushed down, rotates counter-clockwise the step motor (-45°).

Homework 15

15.4 조도 센서 (photo sensor)값에 따라 LED 밝기 자동 조절 기능 구현

- 조도 센서로 측정된 밝기가 어두우면 LED를 밝게 하고, 측정된 조도가 밝으면 LED를 어둡게 하여, 주변 환경에 자동적으로 밝기 조절이 되는 기능을 구현

15.5 거리 센서 측정에 따라 servo motor로 gauge 표시

- TF mini-LiDAR로 측정된 거리 값을 servo motor를 사용하여 gauge 처럼 표시하는 기능 구현
- 측정된 거리를 1 ~ 10 meter 단위로 표시할 수 있도록 할 것.



참고자료

<pigpio>

- [1] pigpio, <https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=asdf2017&logNo=221379376965>
- [2] pigpio, <https://luigibox.tistory.com/76>
- [3] pigpio, <http://abyz.me.uk/rpi/pigpio/index.html>

<sensor adaptation>

- [1] PCF8591 AD DA Board (8-bit 4 channel ADC, 1 channel DAC), <https://www.eleparts.co.kr/goods/view?no=436536>.
- [2] ADS1115 16-Bit ADC, 4 Channel with Programmable Gain Amplifier.
- [3] 디지털 온도 센서 모듈, LM75A, <https://www.eleparts.co.kr/goods/view?no=2952400>.

