

라즈베리파이 RPi.GPIO 모듈

GPIO

❖ GPIO

- General Purpose Input/Output

NAME		NAME	
1	3.3v DC Power	DC Power 5v	2
3	GPIO 2 (SDA1, I ² C)	DC Power 5v	4
5	GPIO 3 (SCL1, I ² C)	Ground	6
7	GPIO 4 (GPIO_GCLK)	(TXD0) GPIO 14	8
9	Ground	(RXD0) GPIO 15	10
11	GPIO 17 (GPIO_GEN0)	(GPIO_GEN1) GPIO 18	12
13	GPIO 27 (GPIO_GEN2)	Ground	14
15	GPIO 22 (GPIO_GEN3)	(GPIO_GEN4) GPIO 23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO 24	18
19	GPIO 10 (SPI_MOSI)	Ground	20
21	GPIO 9 (SPI_MISO)	(GPIO_GEN6) GPIO 25	22
23	GPIO 11 (SPI_CLK)	(SPI_CE0_N) GPIO 8	24
25	Ground	(SPI_CE1_N) GPIO 7	26
27	ID_SD (I ² C ID EEPROM)	(I ² C ID EEPROM) ID_SC	28
29	GPIO 5	Ground	30
31	GPIO 6	GPIO 12	32
33	GPIO 13	Ground	34
35	GPIO 19	GPIO 16	36
37	GPIO 26	GPIO 20	38
39	Ground	GPIO 21	40

GPIO

❖ GPIO

- 설정에 따라 입력/출력 설정 가능
- 3.3V/3mA로 동작
- 디지털 값 구분
 - 1 : 1.7 이상
 - 0 : 1.7 미만

GPIO

❖ 직렬 인터페이스 선(GPIO 번호)

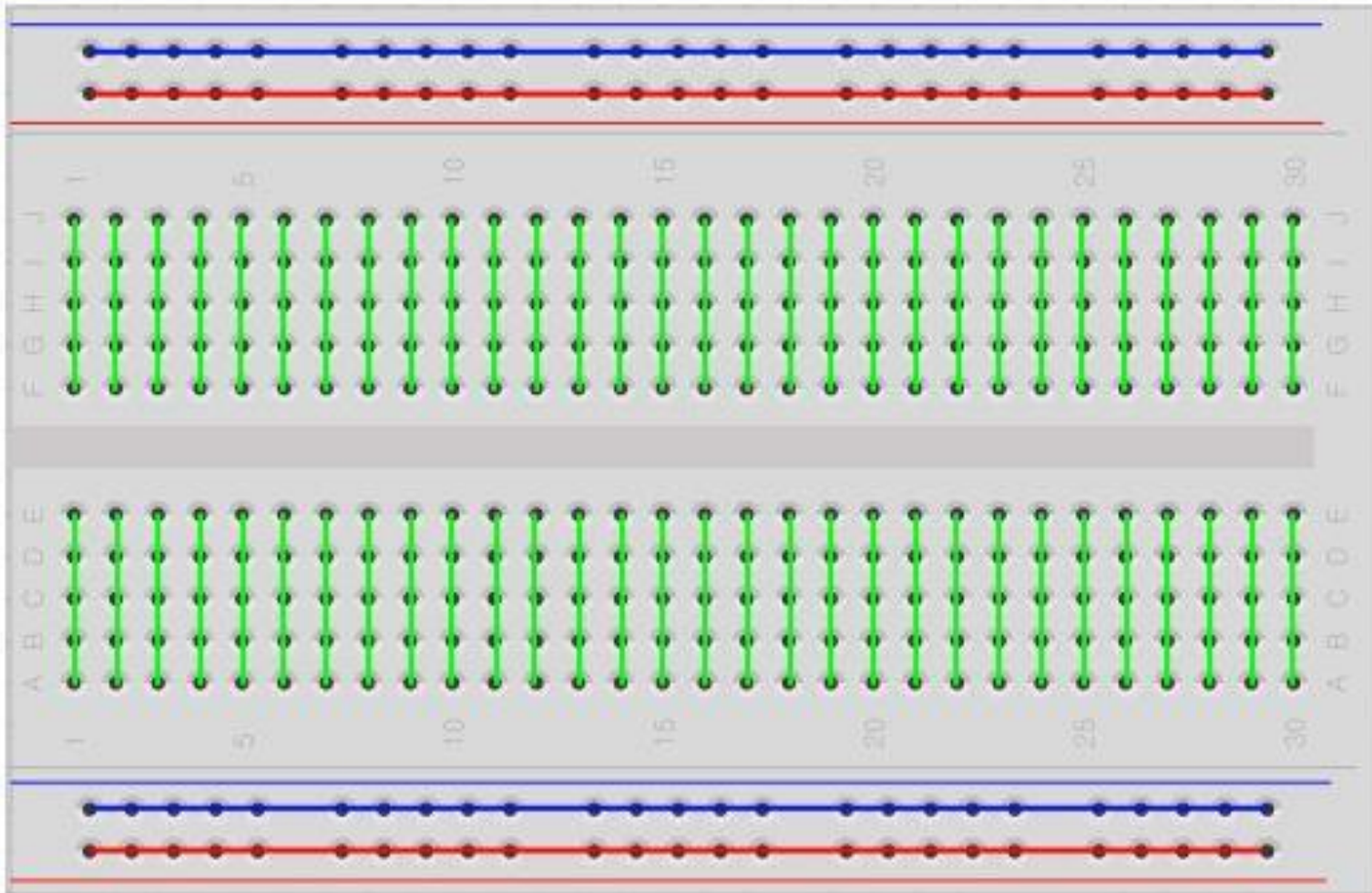
- I2C 인터페이스
 - 2번 핀(SDA) : 직렬 데이터 선
 - 3번 핀(SCL) : 클럭 선
 - 주변 장치와 통신하기 위한 인터페이스

- 직렬 포트
 - 14번 핀(TXD) : 전송용
 - 15번 핀(RXT) : 수신용

- 직렬 통신(SPI)
 - 9번 핀 : MISO
 - 10번 핀 : MOSI
 - 11번 핀 : SCLK

GPIO

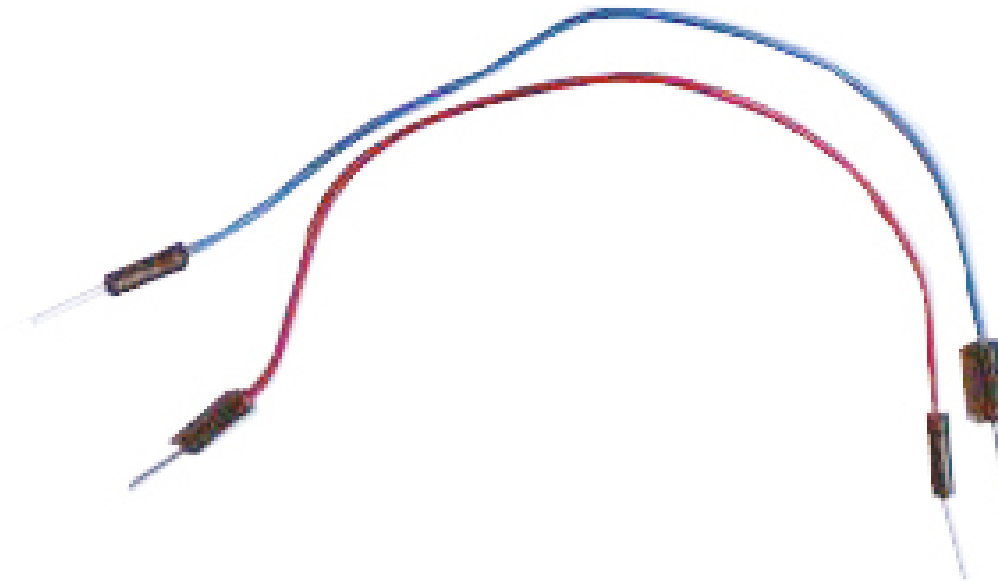
❖ 브레드보드(Breadboard)



GPIO

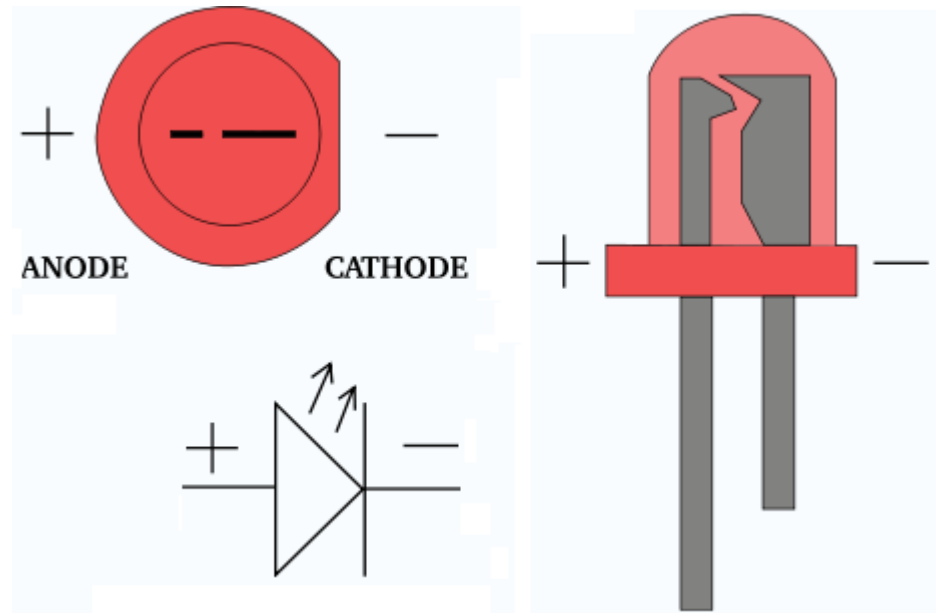
❖ 점퍼선

- 브레드 보드에 부품들을 서로 연결할 때 이용



❖ LED 발광 다이오드

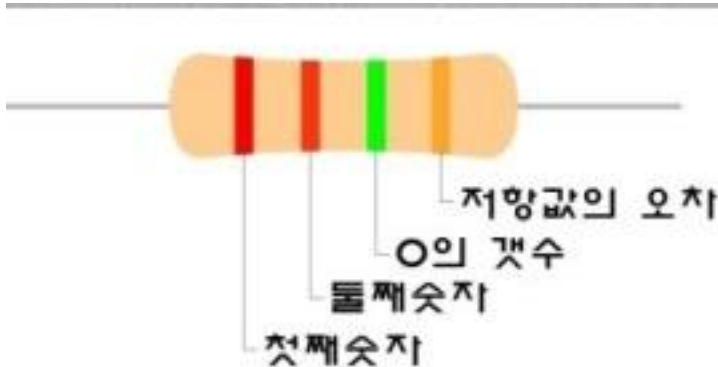
- 전류가 흐르면서 빛을 발하는 반도체
- 다이오드 : 전류를 한쪽 방향으로만 흐르게 함
- 과다 전류가 흐를 경우 파손
 - LED 앞에 저항을 배치



GPIO

❖ 저항

- 전기의 흐름을 제한

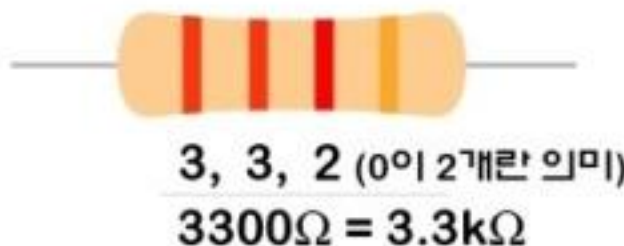


첫째숫자

둘째숫자

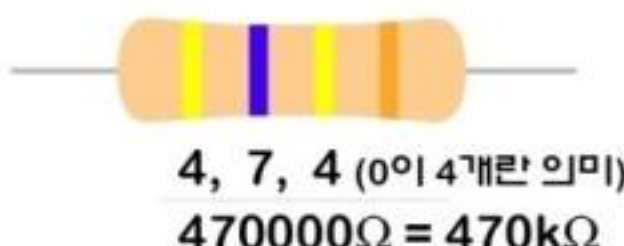
0의 갯수

저항값의 오차



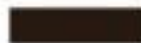









3, 3, 2 (0이 2개란 의미)

$3300\Omega = 3.3k\Omega$



4, 7, 4 (0이 4개란 의미)

$470000\Omega = 470k\Omega$

색	값
	검정색 0
	갈 색 1
	빨강색 2
	주황색 3
	노란색 4
	초록색 5
	파란색 6
	보라색 7
	회 색 8
	하얀색 9
	±10% 은 색
	± 5% 금 색

GPIO

❖ RPi.GPIO 모듈

- 라브베리 파이의 GPIO 제어용 파이썬 모듈
- import 하여 사용

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

- 핀 번호 구분 방법 설정
 - `GPIO.setmode(GPIO.BOARD)` : 핀 번호를 라즈베리파이 보드 번호로 참조
 - `GPIO.setmode(GPIO.BCM)` : **BCM(Broadcom chip-specific pin numbers)모드**

GPIO

❖ RPi.GPIO 모듈

- 해당 핀의 입/출력 모드 설정
 - `GPIO.setup(핀번호, 모드)`
 - 모드 : `GPIO.IN(입력모드)`, `GPIO.OUT(출력모드)`
- GPIO 18번 핀을 출력모드로 설정하기

```
GPIO.setup(18, GPIO.OUT)
```

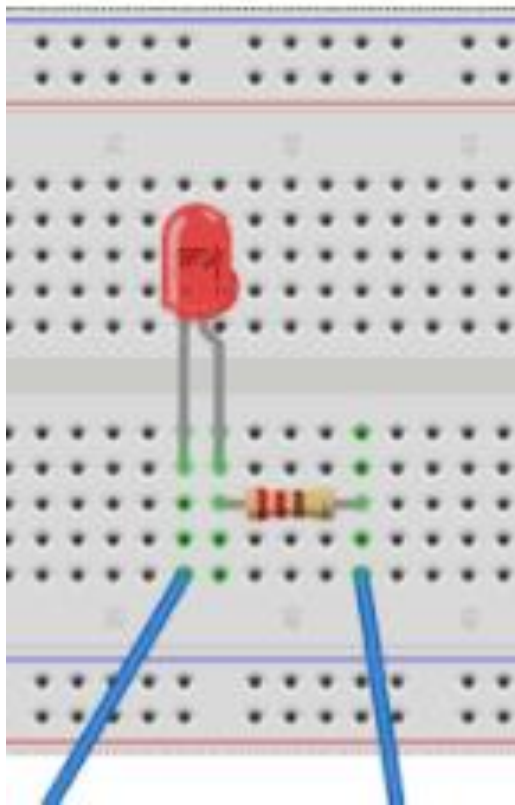
- 출력 값 설정하기
 - `GPIO.output(핀번호, 값)`
 - 값 : `True` 또는 `GPIO.HIGH(1)` / `False` 또는 `GPIO.LOW(0)`

```
# GPIO 18번 핀으로 1 출력  
GPIO.output(18, True)
```

```
# GPIO 18번 핀으로 0 출력  
GPIO.output(18, False)
```

출력

❖ GPIO 18번 핀으로 LED 깜박이기(blink)



GND (9번)

GPIO 18번 핀

NAME		NAME	
1	3.3v DC Power	DC Power 5v	2
3	GPIO 2 (SDA1, I ² C)	DC Power 5v	4
5	GPIO 3 (SCL1, I ² C)	Ground	6
7	GPIO 4 (GPIO_GCLK)	(TXD0) GPIO 14	8
9	Ground	(RXD0) GPIO 15	10
11	GPIO 17 (GPIO_GEN0)	(GPIO_GEN1) GPIO 18	12
13	GPIO 27 (GPIO_GEN2)	Ground	14
15	GPIO 22 (GPIO_GEN3)	(GPIO_GEN4) GPIO 23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO 24	18
19	GPIO 10 (SPI_MOSI)	Ground	20
21	GPIO 9 (SPI_MISO)	(GPIO_GEN6) GPIO 25	22
23	GPIO 11 (SPI_CLK)	(SPI_CE0_N) GPIO 8	24
25	Ground	(SPI_CE1_N) GPIO 7	26
27	ID_SD (I ² C ID EEPROM)	(I ² C ID EEPROM) ID_SC	28
29	GPIO 5	Ground	30
31	GPIO 6	GPIO 12	32
33	GPIO 12	Ground	34
35	GPIO 19	GPIO 16	36
37	GPIO 26	GPIO 20	38
39	Ground	GPIO 21	40

❖ RPi.GPIO 모듈

- idle 수퍼 유저 권한으로 실행
 - 장치 제어는 수퍼유저 권한이 필요함
- blink.py 작성

```
import RPi.GPIO as GPIO
import time

led = 18

GPIO.setmode(GPIO.BCM)    # BCM 핀번호 사용
GPIO.setup(led, GPIO.OUT)

try:
    while True :
        GPIO.output(led, GPIO.HIGH)    # LED 켜기
        time.sleep(0.5)                # 0.5초 대기
        GPIO.output(led, GPIO.LOW)     # LED 끄기
        time.sleep(0.5)                # 0.5초 대기
finally:
    GPIO.cleanup()              # Ctrl-C로 프로그램 종료 시 모든 GPIO 핀을 초기상태로
```

아날로그 출력의 원리

❖ 아날로그 출력 방식

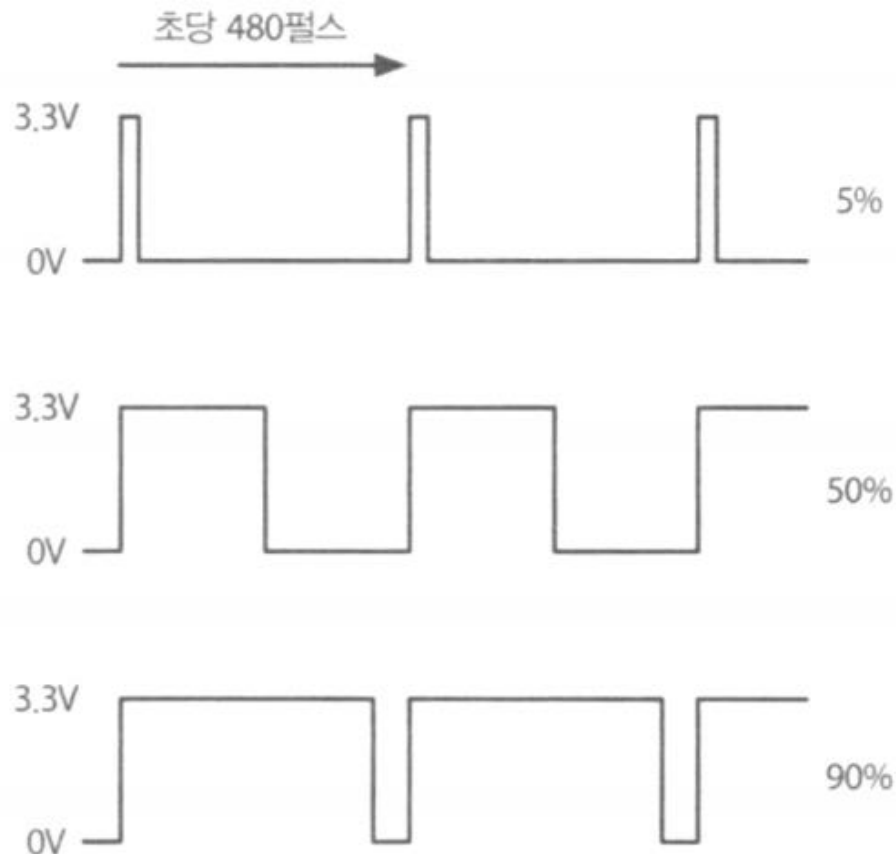
- 라즈베리 파이에는 DAC(Digital to Analog Converter) 없음
- PWM(Pulse Width Modulation) : 펄스 폭 변조
 - 디지털 출력을 조절해서 아날로그 출력 효과를 구현
 - 디지털 값 \rightarrow 아날로그 값 : 아날로그 출력



출력

❖ 아날로그 출력 방식

- 펄스폭 변조(PWM, Pulse Width Modulation)
 - 1을 나타내는 펄스의 폭을 조절
 - 아날로그 값을 펄스 폭의 길이로 매핑



Duty 비율 :
펄스가 High를 유지하는 시간 비율(%)

출력

❖ 아날로그 출력 : LED의 밝기 조절

- PWM 채널 만들기
 - GPIO.PWM(핀번호, 펄스주파수)
 - PWM 채널 리턴

```
pwm_led = GPIO.PWM(18, 500) # GPIO 18번 핀에 500Hz PWM 채널 생성
```

- PWM 채널로 듀티비 초기 설정

```
pwm_led.start(100) # duty 비율 100%로 시작
```

- PWM 채널 듀티비 변경

```
pwm_led.ChangeDutyCycle(duty) # 지정한 듀티비율로 변경
```

출력

❖ 아날로그 출력 : LED의 밝기 조절

- pwm.py

```
#09_pwm.py

import RPi.GPIO as GPIO

led_pin = 18
GPIO.setmode(GPIO.BCM)
GPIO.setup(led_pin, GPIO.OUT)

pwm_led = GPIO.PWM(led_pin, 500)
pwm_led.start(100)

try:
    while True:
        duty_s = input("Enter Brightness (0 to 100):")
        duty = int(duty_s)
        pwm_led.ChangeDutyCycle(duty)
finally:
    print("Cleaning up")
    GPIO.cleanup()
```


버튼 입력 받기

❖ 스위치를 이용한 디지털 입력

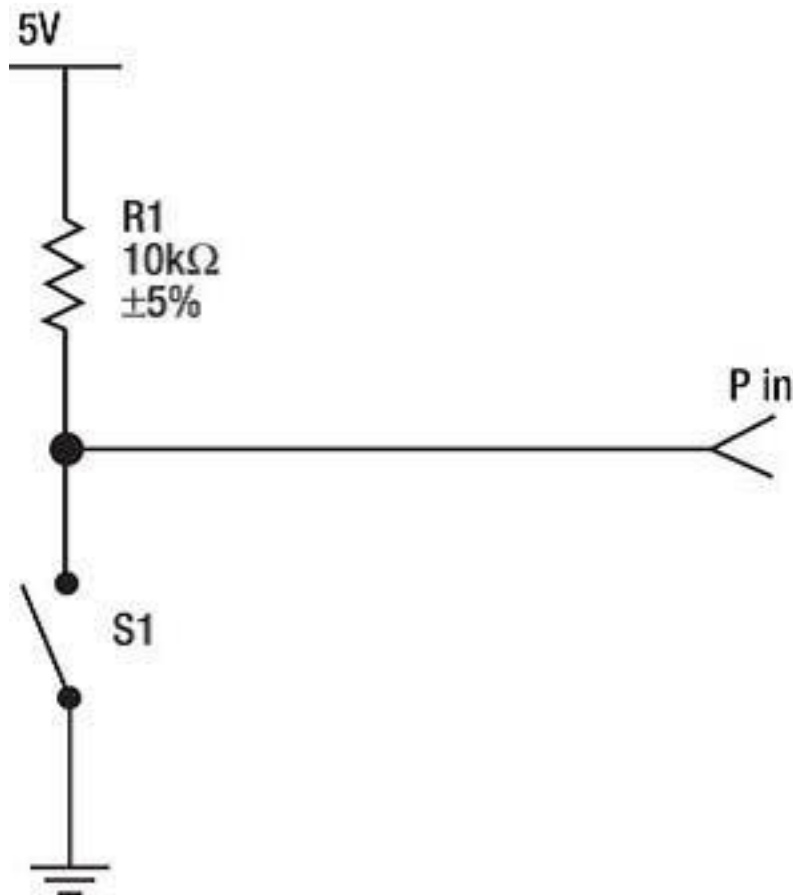
- 스위치 연결시 LED 켜기
- 디지털 입력
 - 0인 상태 : 0V (LOW)
 - 1인 상태 : 5V (HIGH)



버튼 입력 받기

❖ 풀업 저항

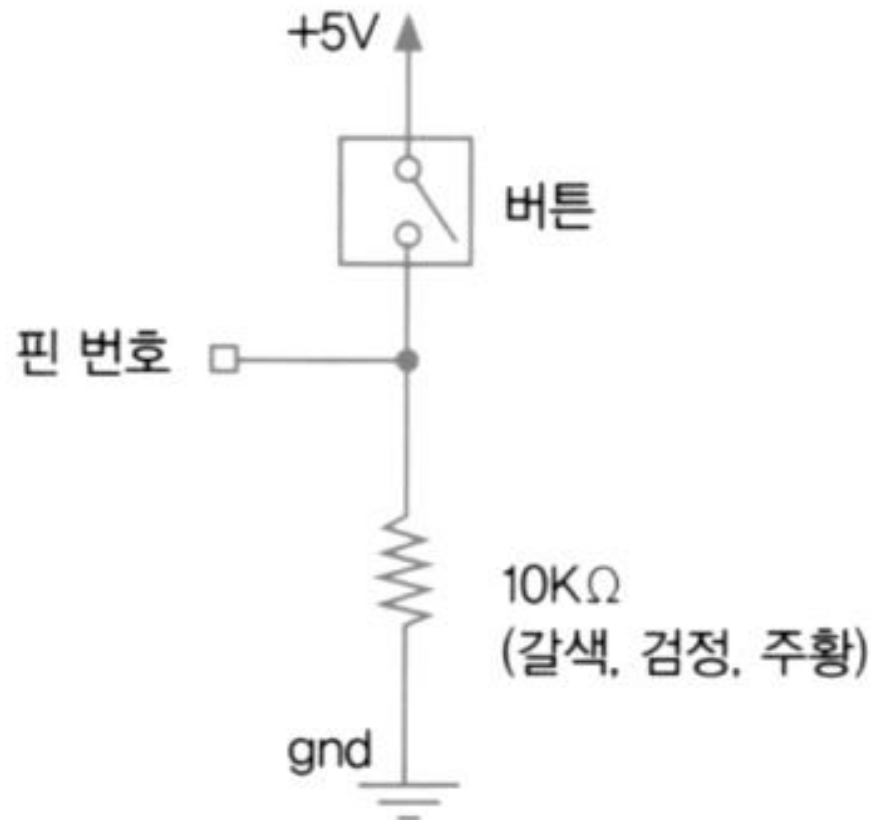
- 전압 소스와 회로 내에 있는 입력 핀 사이에 배치 - 디폴트 HIGH
- 버튼/스위치는 입력 핀과 접지(GND) 사이에 배치 - 스위치를 누르면 LOW



버튼 입력 받기

❖ 풀다운 저항

- 접지와 회로 내에 있는 입력 핀 사이에 배치 - 디폴트 LOW
- 버튼/스위치는 전원 소스와 핀 사이에 배치 - 스위치를 누르면 HIGH



버튼 입력 받기

❖ 스위치를 이용한 디지털 입력

- 점퍼선을 이용한 스위치 회로 만들기
 - 반드시 저항을 연결하여야 함
 - 내장 풀업 저항 활성화
 - 디폴트 1, 스위치 연결시 0
 - `GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)`
 - 내장 풀다운 저항 활성화
 - 디폴트 0, 스위치 연결시 1
 - `GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)`
- `pull_up_down` 파라미터를 주지 않으면 아무런 저항도 연결되지 않음

입력

❖ 스위치를 이용한 디지털 입력

- switch.py

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

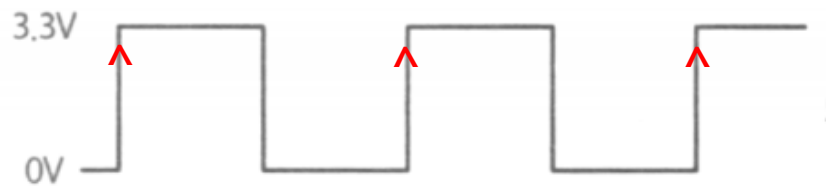
switch_pin = 23

GPIO.setup(switch_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

try:
    while True:
        if GPIO.input(switch_pin) == False:
            print("Button Pressed")
            time.sleep(0.2)
finally:
    GPIO.cleanup()
```

❖ 에지 트리거

- 펄스의 상승 또는 하강 신호를 인식
- 버튼을 누를 때 한 번 처리



버튼 입력 받기

❖ Debounce

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

switch_pin = 23

GPIO.setup(switch_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

buttonState = None      # the current reading from the input pin
lastButtonState = LOW   # the previous reading from the input pin

lastDebounceTime = 0
debounceDelay = 0.050    # 측정 지연 시간 50 ms
```

버튼 입력 받기

❖ Debounce

```
try:
    while True:
        reading = GPIO.input(switch_pin)
        if reading != lastButtonState: # 이전 상태와 달라짐, 시간 측정
            lastDebounceTime = time.time()

        # 지연 시간 이내의 변화라면 무시
        if (time.time() - lastDebounceTime) > debounceDelay :
            if reading != buttonState) {
                buttonState = reading
                if buttonState == HIGH :
                    ledState = !ledState
                # 값 변경 결정됨
                # 필요한 조치 실행
            lastButtonState = reading

finally:
    GPIO.cleanup()
```