

8장 GPIO와 LED, 스위치, 초음파 센서 다루기

강의 목표

1. GPIO의 개념과 라즈베리파이를 이용한 하드웨어 제어에 대해 이해한다.
2. GPIO의 40핀에 대해 이해한다.
3. LED 제어 하드웨어를 꾸미고 파이썬으로 제어하는 것에 대해 이해한다.
4. Switch 제어 하드웨어를 꾸미고 파이썬으로 제어하는 것에 대해 이해한다.
5. 초음파 센서 하드웨어를 꾸미고 파이썬으로 제어하는 것에 대해 이해한다.

들어가기 전에

3

- 예제와 실습은 myenv 가상 환경에서 실행
 - ▣ 파이썬 코드 실행 전 반드시 가상 환경 활성화

```
pi@pi:~ $ source myenv/bin/activate  
(myenv) pi@pi:~ $
```

- 예제 파이썬 코드는 ch08 디렉터리에 저장하고 실행

```
(myenv) pi@pi:~ $ mkdir ch08  
(myenv) pi@pi:~ $ cd ch08  
(myenv) pi@pi:~/ch08 $
```

8.1 라즈베리파이와 GPIO

□ GPIO

- ▣ GPIO(General Purpose Input/Output)
- ▣ 컴퓨터와 외부 하드웨어 사이의 '범용 입출력 인터페이스'
 - 마이크로프로세서, 마이크로컨트롤러 등의 임베디드 하드웨어가 외부 장치사이에서 디지털 혹은 아날로그 신호를 주고 받기 위한 I/O 핀들
 - 라즈베리파이4 B에 40개의 GPIO 핀

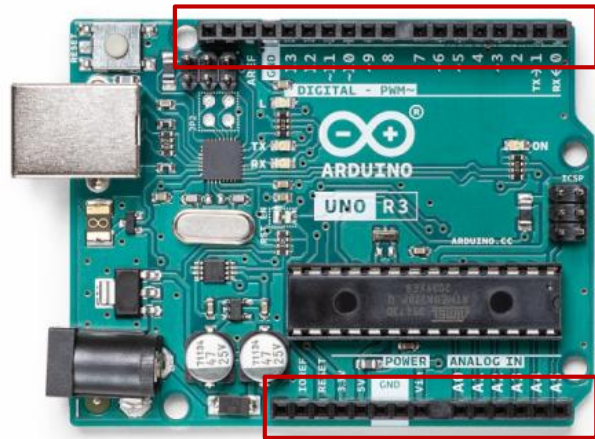
□ GPIO 특징

- ▣ 수 개에서 수십 개의 핀들로 구성
- ▣ 핀들은 몇몇 필수 핀 외에는 미리 정해진 목적이 없다.
- ▣ 핀은 입력이나 출력 중에서 한가지로만 선택 사용
- ▣ 핀을 통해 외부 하드웨어에 전원 공급
- ▣ 핀으로의 신호 송수신은 소프트웨어로 제어

임베디드 컨트롤러의 GPIO 사례

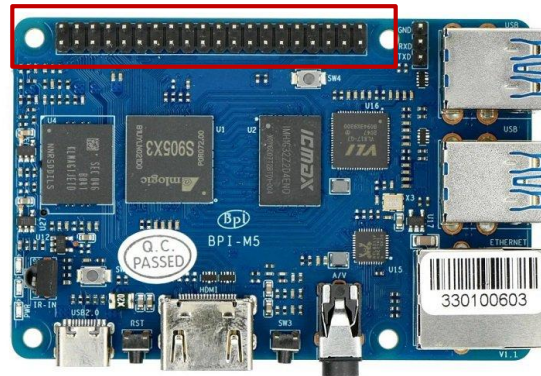
6

아두이노의 GPIO



아두이노의 GPIO

바나나파이의 GPIO



라즈베리파이의 GPIO



라즈베리파이3 모델 B/B+의 GPIO 핀 40개

7




핀번호	이름	이름	핀번호
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 (SPI0_MOSI)	Ground	20
21	GPIO09 (SPI0_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI0_CLK)	(SPI0_CE0_N) GPIO08	24
25	Ground	(SPI0_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

BOARD 모드에서 사용하는 핀 번호 BCM 모드에서 사용하는 핀 번호(이름)

소프트웨어로 GPIO 핀 제어

8

- 소프트웨어로 GPIO에 연결된 장치와 데이터를 주고받음
- 디지털 값과 전압
 - ▣ 디지털 값 1(이진수 1) : 3.3V 혹은 5V, HIGH로 부름
 - ▣ 디지털 값 0(이진수 0) : 0V 혹은 GND, LOW로 부름
- GPIO 라이브러리들

언어	라이브러리 이름	언어	라이브러리 이름
C/C++	pigpio, BCM2835	Python	RPi.GPIO  이 책에서 사용하는 라이브러리
C#	RaspberryGPIOManager	Scratch	ScratchGPIO
Java	Pi4J	Shell	sysfs
Perl	BCM2835		

RPi.GPIO 라이브러리 활용

9

- RPi.GPIO 라이브러리
 - ▣ GPIO에 연결된 장치들을 제어하는 파이선 라이브러리

- RPi.GPIO 라이브러리 설치
 - 1) myenv 가상 환경 활성화
 - pi@pi:~ \$ **source myenv/bin/activate**
 - (myenv) pi@pi:~ \$
 - 2) RPi.GPIO 라이브러리 다운로드
 - (myenv) pi@pi:~ \$ **pip install RPi.GPIO**

- 파이선 프로그램에 RPi.GPIO 활용
 - ▣ **import RPi.GPIO as GPIO**

RPi.GPIO 모듈의 함수들

10

함수	기능
GPIO.setmode(GPIO.BOARD)	BOARD 모드의 핀 번호 사용
GPIO.setmode(GPIO.BCM)	BCM 모드의 핀 번호 사용
GPIO.getmode()	setmode()로 설정된 값 리턴(GPIO.BOARD, GPIO.BCM, None 중 하나)
GPIO.setup(pin, GPIO.IN)	pin을 입력으로 설정. pin은 번호를 나타내는 정수
GPIO.setup(pin, GPIO.OUT)	pin을 출력으로 설정
GPIO.input(pin)	pin으로부터 디지털 값 읽어 리턴(0 또는 1)
GPIO.output(pin, GPIO.HIGH)	pin으로 디지털 1(HIGH) 값 출력. GPIO pin에 5V의 전압 출력
GPIO.output(pin, GPIO.LOW)	pin으로 디지털 0(LOW) 값 출력. GPIO pin에 0V(GND)의 전압 출력
GPIO.cleanup()	사용한 핀들의 번호를 기억에서 지우고, 사용한 핀들을 모두 입력으로 초기화
GPIO.setwarnings(True/False)	매개변수가 False이면 이미 사용 중인 핀에 다음 프로그램에서 setmode()를 호출하여 입출력 용도를 바꿀 때 경고 메시지가 출력되지 않게 하고, 매개변수가 True이면 경고 메시지가 출력되게 함

GPIO 핀 번호 매기는 방법

11

- 응용프로그램은 GPIO 핀 사용시, 핀 번호 사용
- GPIO 핀에 번호를 매기는 2가지 방법
 - ▣ BOARD 모드 - 물리적으로 배치된 순서로 매겨진 번호 사용
 - ▣ BCM 모드 - 용도별 핀 번호
 - 라즈베리파이의 핵심 프로세스인 BCM 칩에 의해 매겨진 번호
 - ▣ 모드 선택
 - ▣ 현재 모드 알아내기

```
GPIO.setmode(GPIO.BOARD) # BOARD 모드 선택  
GPIO.setmode(GPIO.BCM) # BCM 모드 선택
```

```
mode = GPIO.getmode() # GPIO.BOARD, GPIO.BCM, None 중 하나 리턴
```

GPIO 핀 사용하기(1)

12

□ GPIO 핀의 입출력 용도 선택

```
GPIO.setup(6, GPIO.IN)      # GPIO6 핀을 입력으로 설정
GPIO.setup(21, GPIO.OUT)    # GPIO21 핀을 출력으로 설정
```

□ GPIO 핀으로부터 입출력

▣ 핀으로부터 디지털 값 읽기

```
value = GPIO.input(6)      # GPIO6 핀으로부터 디지털 신호 읽기
```

▣ 핀에 디지털 값 출력

■ GPIO21 핀에 디지털 값 1 출력. 실제 5V 전압 출력

```
GPIO.output(21, 1)
GPIO.output(21, GPIO.HIGH)
GPIO.output(21, True)
```

■ GPIO21 핀에 디지털 값 0 출력. 실제 0V(GND) 전압 출력

```
GPIO.output(21, 0)
GPIO.output(21, GPIO.LOW)
GPIO.output(21, False)
```

GPIO 핀 사용하기(2)

13

□ RPi.GPIO 라이브러리의 버전 알아내기

```
print(GPIO.VERSION)
```

□ GPIO 클린업

```
GPIO.cleanup()
```

- GPIO 라이브러리가 현재 프로그램이 사용한 핀들의 번호 기억 삭제
 - 현재 프로그램 사용한 모든 핀을 입력(GPIO.IN)으로 초기화
- 클린업의 중요성
 - 어떤 핀이 출력으로 지정되어 있고, 5V가 출력되고 있는 상황이라면,
 - 응용프로그램이 종료되어도 여전히 상황이 유지되므로,
 - 사용자의 실수나 우연한 상황으로 그 핀에 0V가 연결되면 합선 발생
 - 회로가 고장날 수 있음

경고 메시지 출력 막기

14

- RPi.GPIO 모듈은 현재 실행 중인 프로그램이 사용하고 있는 핀 번호들을 기억,
- 다른 프로그램에서 동일한 핀을 사용할 경우 다음 경고 메시지 출력

RuntimeWarning: This channel is already in use, continuing anyway. Use GPIO.setwarnings(False) to disable warnings.

- 개발자에게 핀의 충돌을 알려,
- 두 번째 프로그램이 이 핀을 사용하지 말도록 경고
- 경고 메시지가 출력되는 다른 상황
 - a.py가 클린업하지 않고 종료한 뒤, 사용자가 다시 a.py를 실행시킬 때, 경고 메시지 출력
- 경고 메시지 출력을 막는 방법
 - 방법 1 - GPIO.cleanup()
 - 방법 2 - GPIO.setwarnings(False)

바람직한 GPIO 응용프로그램의 구성

15

- 프로그램이 어떤 상황에서 종료하더라도 클린업하도록 작성
 - ▣ 정상적인 종료나,
 - ▣ 사용자의 [Ctrl+C] 키로 종료할 때나,
 - ▣ 오류로 종료할 때 모두 클린업하도록 프로그램 작성 필요

```
import RPi.GPIO as GPIO

try:    #
        # 개발자가 작성하고자 하는 목적 코드들
        #
except KeyboardInterrupt: # [Ctrl+C] 키가 입력되어 종료하는 경우
        # 종료 전 처리할 코드

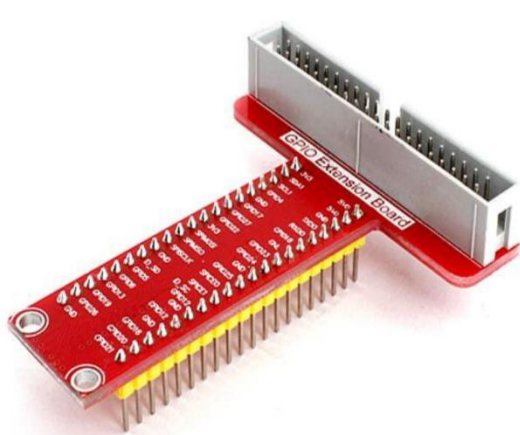
except : # 오류로 인해 종료하는 경우
        # 오류 처리 코드

finally: # 종료하는 모든 경우(정상적이든 오류로 인한 종료 등)에 실행되는 코드
        GPIO.cleanup()
```

라즈베리파이 외부로 GPIO 핀 확장

16

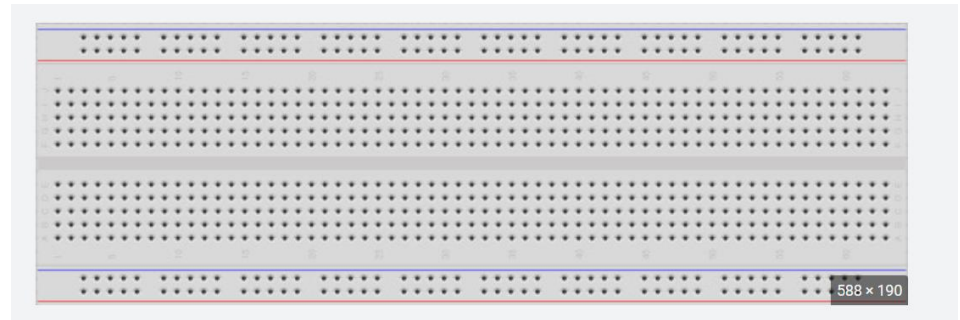
- 라즈베리파이의 GPIO 핀에 외부 장치 연결
 - ▣ 직접 연결 - 매우 불편
 - ▣ 40핀의 플랫 케이블, T자형 GPIO 확장 보드, 브레드보드 활용



T자형 GPIO 확장 보드



40핀 플랫 케이블

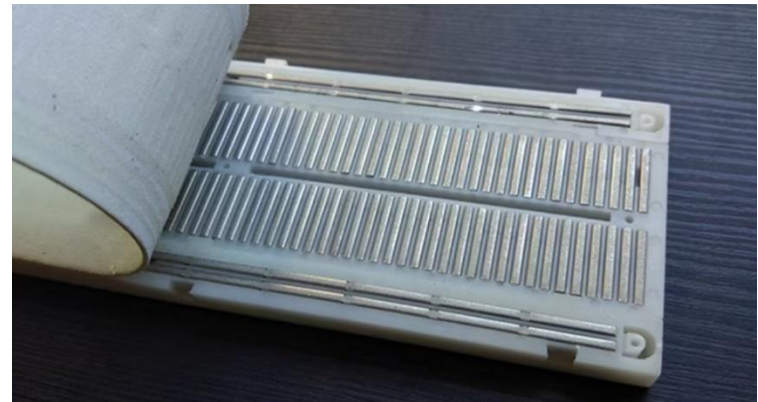
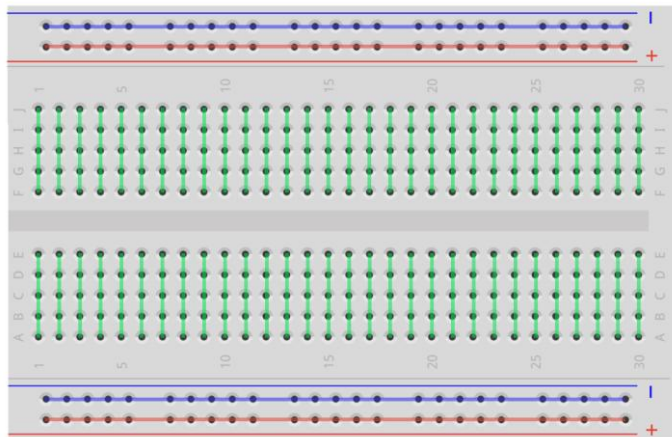


브레드보드

브레드보드

17

- 브레드보드(Bread Board, 빵판)
 - ▣ 사용자가 납땜하지 않고 회로를 쉽게 구성할 수 있도록 설계된 장치
- 브레드보드의 구조
 - ▣ 수평으로 연결된 라인과 수직으로 연결된 라인들
 - 라인(파란색, 붉은색, 초록색)은 내부적으로 연결되어 있음



- ▣ 5V와 GND 선
 - 브레드보드에 한 번 연결해두고 계속 사용

GPIO핀과 브레드보드의 연결

18

- T자형 확장보드와 플랫 케이블 이용, GPIO의 40개 핀과 브레드보드의 연결



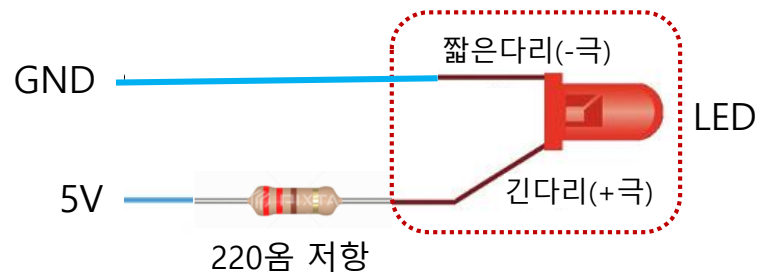
8.2 LED 제어

LED

20

□ LED

- 발광다이오드(LED, Light Emitting Diode)
- 전류를 가하면 빛을 발하는 반도체 소자
- 흰색, 초록색, 빨강색 등 다양한 색
- 긴다리와 짧은 다리 사이에 적절한 전압(2.2V~2.4V)이 가해지면 전류가 흐르고 빛을 발하게 됨



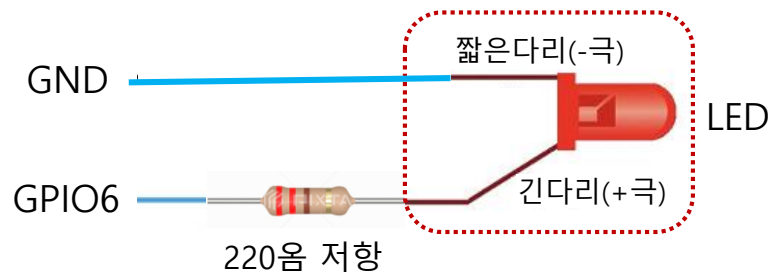
□ 저항 필요한 이유

- 긴다리 5V, 짧은 다리 GND 연결할 경우
 - 빛은 밝지만, 과전류로 인해 LED 손상
 - 저항을 사용하여 전류량 감소시킬 필요
- 220 Ω 저항 사용(1KΩ 이하)
- 저항이 클수록 LED의 밝기가 약해짐

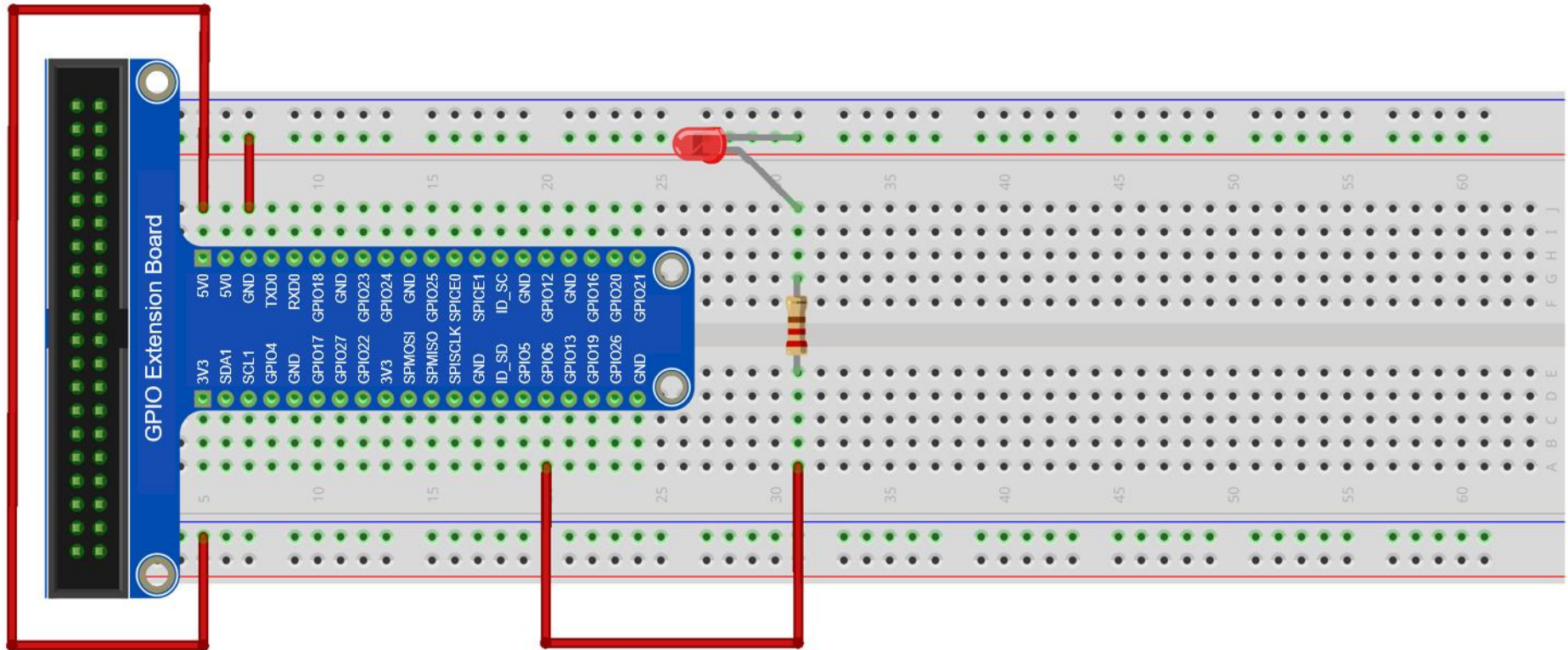
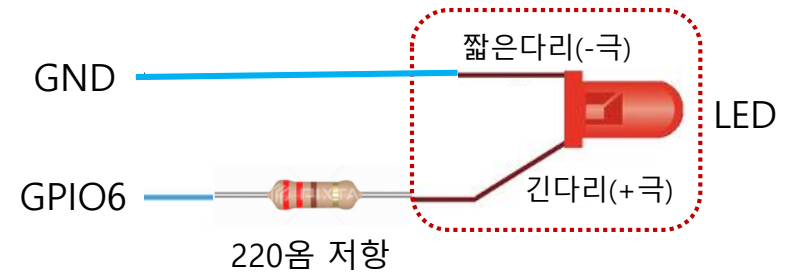
LED 제어 실습

21

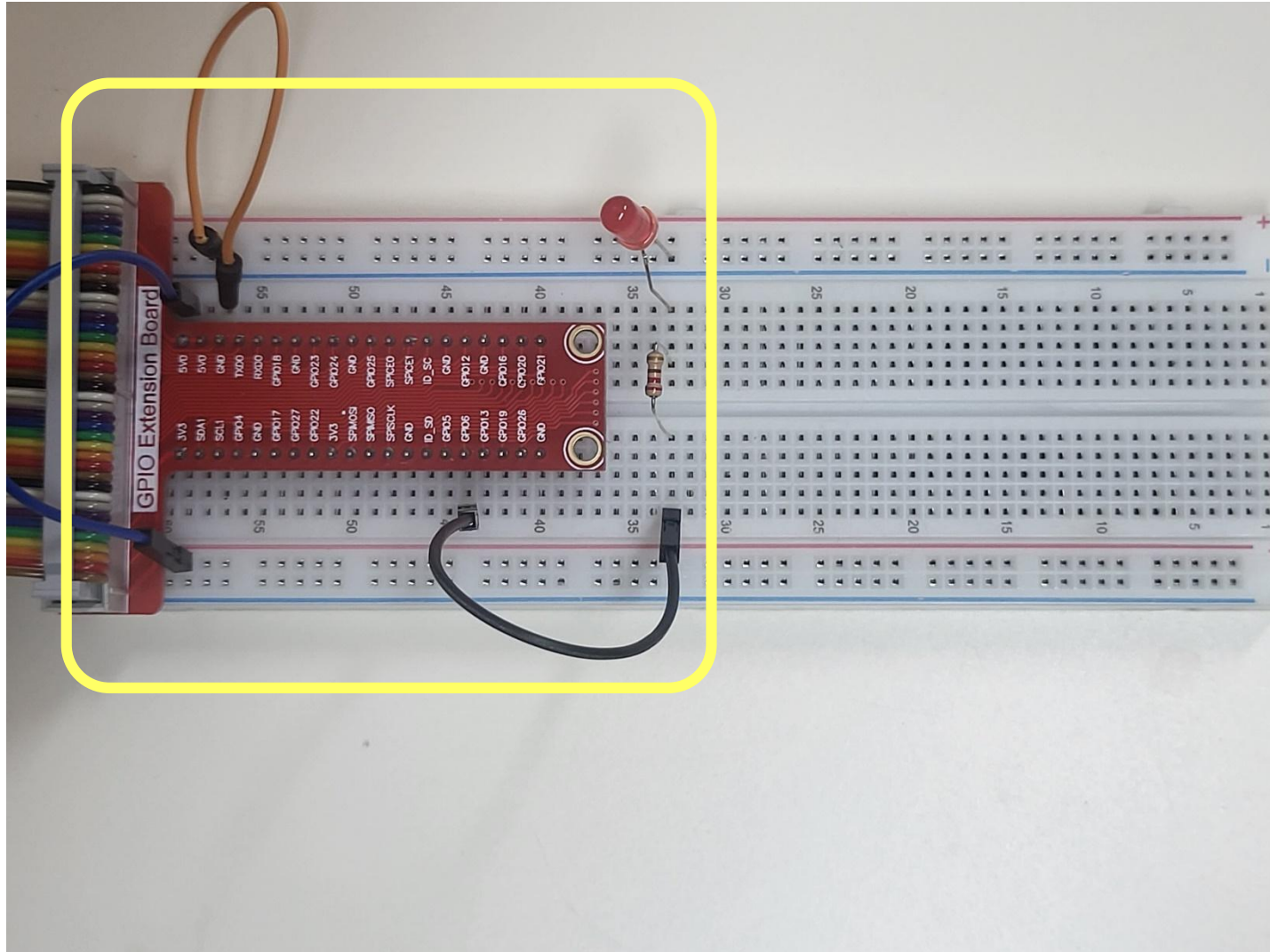
- 실습 내용
 - ▣ 라즈베리파이에서 파이선 프로그램으로 LED 켜기/끄기
- 부품
 - ▣ LED 1개
 - ▣ 220Ω 저항 1개
 - ▣ GPIO6 핀 (출력용) – LED의 긴다리에 연결, 출력으로 사용
- 회로 구성



LED 실습을 위한 회로구성



23



예제 8-1 1초 간격으로 LED를 5번 깜박이는 파이썬 프로그램 작성

24

1초 간격으로 LED를 5번 깜박이는 파이썬 프로그램을 작성해보자.

8-1.py

```
import time
import RPi.GPIO as GPIO

# pin에 연결된 LED에 value(0/1) 값을 출력하여 LED를 켜거나 끄는 함수
def led_on_off(pin, value):
    GPIO.output(pin, value)

GPIO.setmode(GPIO.BCM) # BCM 모드로 작동
GPIO.setwarnings(False) # 경고글이 출력되지 않게 설정

led = 6 # GPIO6 핀
GPIO.setup(led, GPIO.OUT) # GPIO6 핀을 출력으로 지정

on_off = 1 # 1은 디지털 출력 값. 1 = 5V

print("LED를 지켜 보세요.")

# 5번 LED를 깜박임
for i in range(5):
    led_on_off(led, on_off) # led가 연결된 핀에 1또는 0 값 출력
    time.sleep(1) # 1초 동안 잠자기
    print(i, end=' ', flush=True)
    on_off = 0 if on_off == 1 else 1 # 0과 1의 토글링

print()
GPIO.cleanup()
```

(myenv) pi@pi:~/ch08 \$ python 8-1.py

LED를 지켜보세요.

0 1 2 3 4

1초 간격으로 0 1 2 3 4 출력

(myenv) pi@pi:~/ch08 \$

try-except-finally를 이용한 예제 8-1의 바람직한 코드

25

- 예제 8-1의 8-1.py의 문제점
 - ▣ 실행 도중 사용자가 [Ctrl+C] 키를 입력하여 프로그램을 종료시킬 때,
 - ▣ LED를 켜 채 1초 잠을 자고 있었다면 GPIO.cleanup()이 실행되지 못하고 종료
 - GPIO6 핀에 5V 전압이 계속 공급 -> LED가 켜진 채로 있게 됨

8-1-adv.py

```
import time
import RPi.GPIO as GPIO

try:
    # pin에 연결된 LED에 value(0/1) 값을 출력하여 LED를 켜거나 끄는 함수
    def led_on_off(pin, value):
        GPIO.output(pin, value)

    GPIO.setmode(GPIO.BCM) # BCM 모드로 작동
    GPIO.setwarnings(False) # 경고글이 출력되지 않게 설정

    led = 6 # GPIO6 핀
    GPIO.setup(led, GPIO.OUT) # GPIO6 핀을 출력으로 지정

    on_off = 1 # 1은 디지털 출력 값. 1 = 5V
    print("LED를 지켜보세요.")

    # 5번 LED를 깜박임
    for i in range(5):
        led_on_off(led, on_off) # led가 연결된 핀에 1또는 0 값 출력
        time.sleep(1) # 1초 동안 잠자기
        print(i, end=' ', flush=True)
        on_off = 0 if on_off == 1 else 1 # 0과 1의 토글링
```

```
except KeyboardInterrupt:
    print("Ctrl+C 종료")
finally:
    print("cleanup")
    GPIO.cleanup()
```

```
(myenv) pi@pi:~/ch08 $ python 8-1-adv.py
LED를 지켜보세요.
0 1 2 3 4 cleanup
(myenv) pi@pi:~/ch08 $ python 8-1-adv.py
LED를 지켜보세요.
0 1 2 ^C Ctrl+C 종료
cleanup
(myenv) pi@pi:~/ch08 $
```

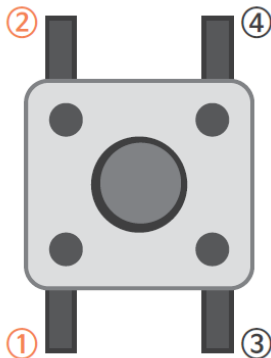
8.3 스위치 제어

스위치

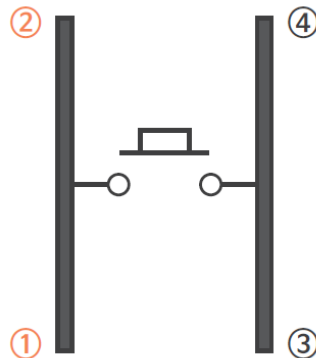
27

□ 택트 스위치(tack switch)

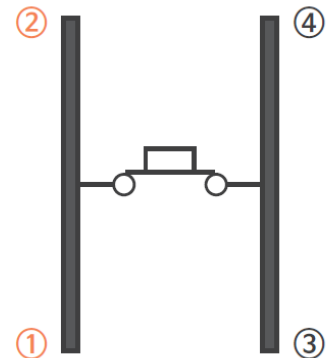
- 흔한 스위치, 가격 저렴, 내부 회로 단순
 - 누르면 On, 떼면 Off
- 가로 세로 높이 1cm 내외 작은 크기
 - 버튼, 키로도 불림
 - 텍타일 스위치(tactile switch)로도 부름
- 휴대폰, MP3, 네비게이션, DMB 등 소형 가전에 많이 사용
- 내부 결선



(a) 위에서 본 모습



(b) 스위치를 누르지 않았을 때

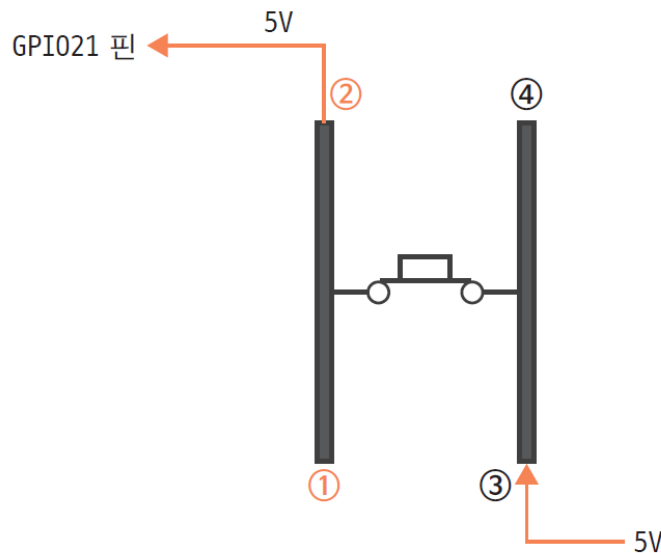


(c) 스위치를 눌렀을 때

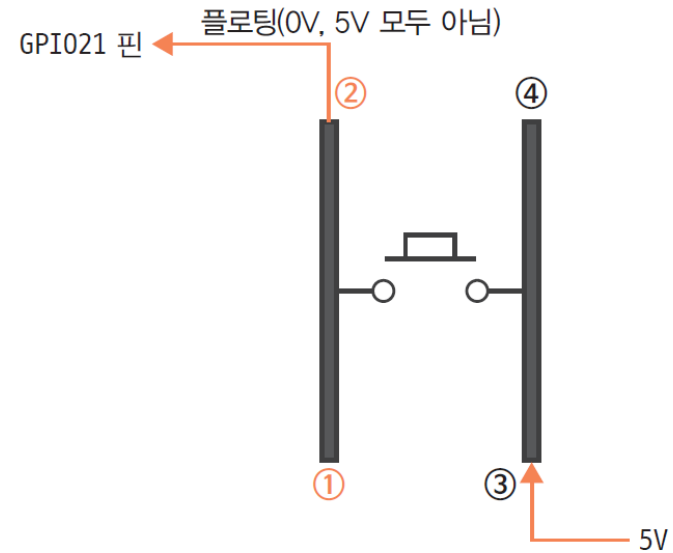
스위치와 풀업(pull-up)/풀다운(pull-down)

28

- 스위치의 입력 신호 상태
 - 스위치를 누른 경우 - GPIO21 핀에 5V(HIGH) 수신
 - 스위치를 누르지 않은 경우 - GPIO21 핀의 값 플로팅 상태
- 풀다운(pull-down)
 - 스위치를 누르지 않는 상태에서 입력되는 플로팅 상태를 0V 입력으로 처리
 - GPIO.setup(21, GPIO.IN, GPIO.PUD_DOWN) # GPIO21 핀에 풀업 효과 지정**
- 풀업 효과(pull-up)
 - 스위치를 누르지 않는 상태에서 입력되는 플로팅 상태를 5V 입력으로 처리
 - GPIO.setup(21, GPIO.IN, GPIO.PUD_UP) # GPIO21 핀에 풀다운 효과 지정**



(a) 스위치를 누른 경우

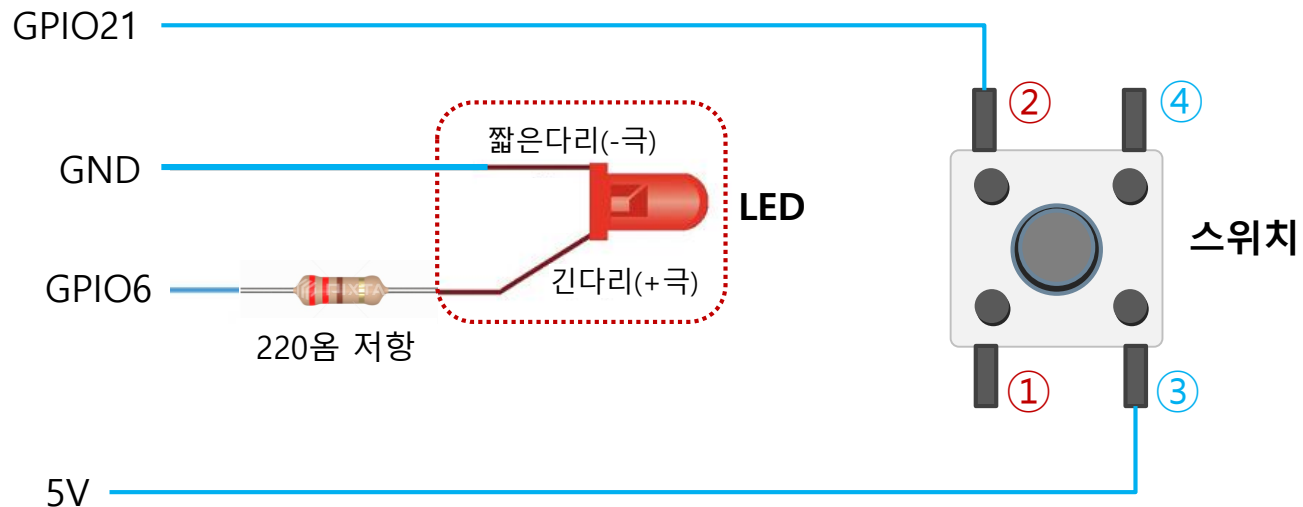


(b) 스위치를 누르지 않았을 경우

스위치 제어 실습

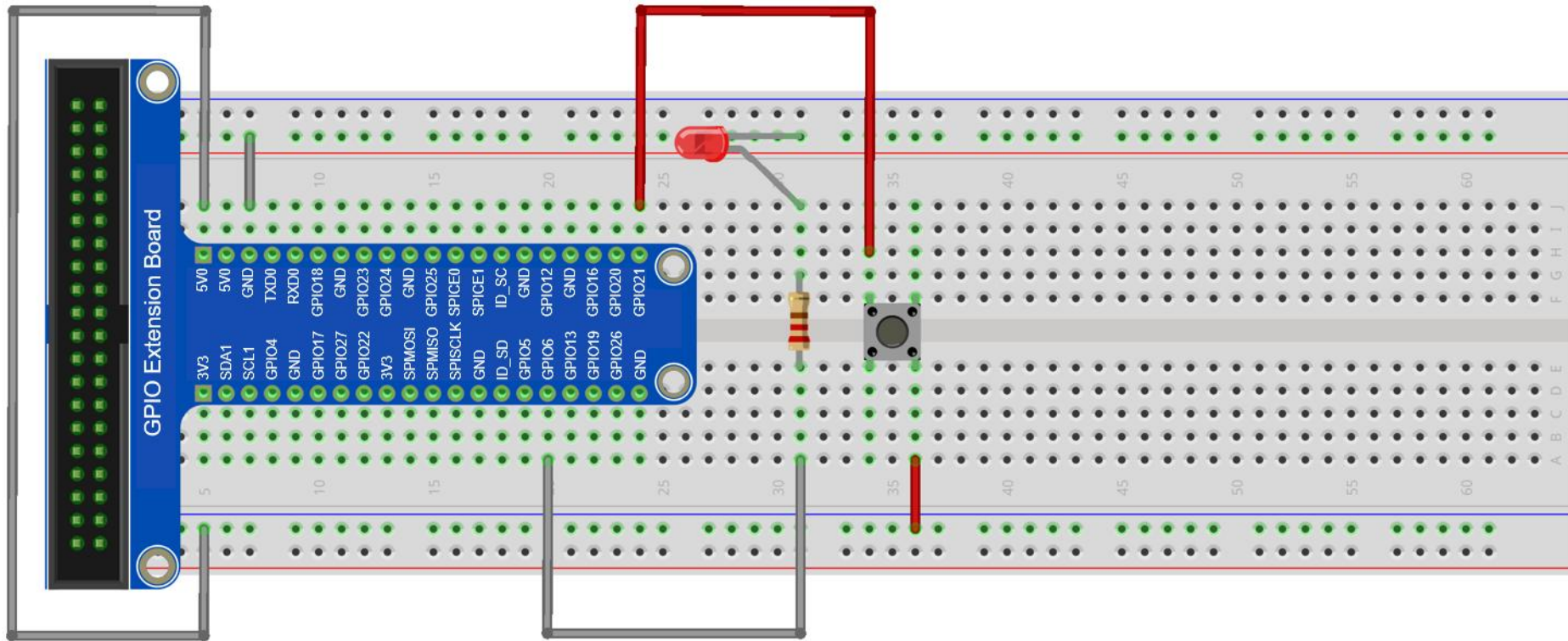
29

- 실습 내용
 - ▣ 스위치의 on/off에 따라 LED를 켜고 끄기
- 부품
 - ▣ LED 1개, 저항 220Ω 1개, 탭트 스위치 1개
 - ▣ GPIO6 핀 (출력용) - LED의 긴 다리에 연결, 출력으로 사용
 - ▣ GPIO21 핀 (입력용) - 탭트 스위치 2번 다리에 연결, 입력으로 사용
- 회로 구성



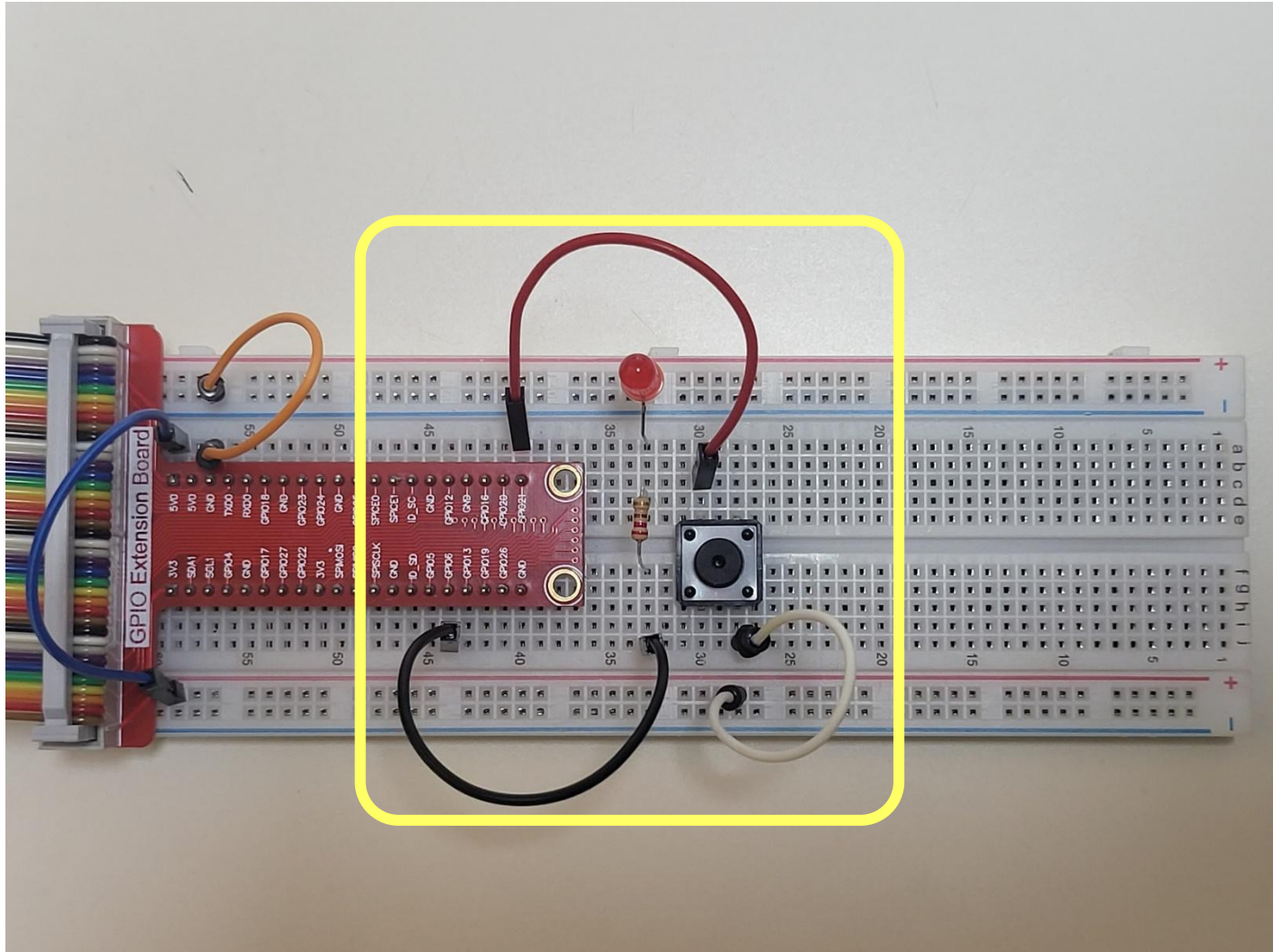
스위치 실습을 위한 회로 구성

30



스위치 실습을 위해 구성된 실제 회로 사진

31



예제 8-2 스위치를 누르고 있는 동안 LED에 불을 켜는 프로그램 작성

32

스위치를 누르고 있는 동안 LED에 불이 켜지고 스위치를 놓으면 불이 꺼지도록 하는 파이썬 프로그램을 작성하라.

8-2.py

(myenv) pi@pi:~/ch08 \$ python 8-2.py
스위치를 누르고 있는 동안 LED가 켜지고 놓으면 꺼집니다.

```
import time
import RPi.GPIO as GPIO

# pin에 연결된 LED에 value(0/1) 값을 출력하여 LED를 켜거나 끄는 함수
def led_on_off(pin, value):
    GPIO.output(pin, value)

GPIO.setmode(GPIO.BCM) # BCM 모드로 작동
GPIO.setwarnings(False) # 경고글이 출력되지 않게 설정

led = 6 # GPIO6 핀
GPIO.setup(led, GPIO.OUT) # GPIO6 핀을 출력으로 지정

button = 21 # GPIO21 핀
GPIO.setup(button, GPIO.IN, GPIO.PUD_DOWN) # GPIO21 핀을 입력으로 지정하고 풀다운 효과 지정

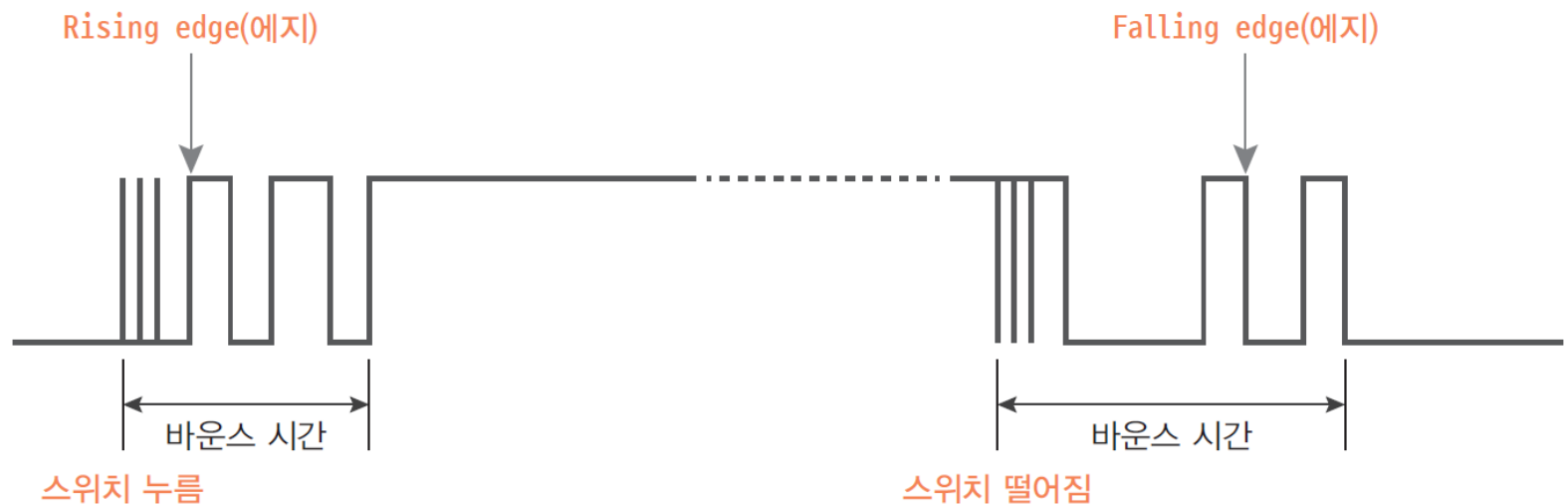
print("스위치를 누르고 있는 동안 LED가 켜지고 놓으면 꺼집니다.")
while True :
    status = GPIO.input(button) # GPIO21 핀로부터 디지털 값(0/1) 읽기
    led_on_off(led, status) # 읽은 값(0/1)을 led(GPIO6 핀)로 출력

GPIO.cleanup()
```


스위치 바운스(bounce)와 디바운스(debounce)

33

- 바운스
 - 눌러지는 순간/떼어지는 순간, 스위치에 많은 접촉 발생
 - 10~100회 정도, 몇십 마이크로초~5ms 사이
 - 모든 기계식 장치에서 발생
- 디바운스
 - 스위치 바운스 문제 해결 방법
 - 하드웨어적 방법 - 스위치 주변에 바운스 제거 회로 추가
 - 소프트웨어적 방법 - 바운스가 끝날 때까지 대기 후 스위치 값 읽기
 - 디바운스 시간은 몇 십 ms



rising edge(엣지)와 falling edge(엣지)

34

□ rising edge

- ▣ 전기 신호가 LOW(0) -> HIGH(1)로 바뀌는 상황
 - GPIO 라이브러리에서는 **GPIO.RISING** 이벤트로 표현

□ falling edge

- ▣ 전기 신호가 HIGH(1) -> LOW(0)로 바뀌는 상황
 - GPIO 라이브러리에서는 **GPIO.FALLING** 이벤트로 표현

스위치 ON을 기다리는 폴링 방법

35

- 스위치가 연결된 GPIO 핀(swtich_pin)으로부터 읽은 값이 LOW에서 HIGH로 바뀔 때까지 10ms 주기로 루프를 도는 방법

```
while GPIO.input(swtich_pin) == GPIO.LOW:  
    time.sleep(0.01) # 10ms 대기
```

```
# while 문을 벗어난 상태는 GPIO.input(swtich_pin)이 GPIO.HIGH를 리턴한 경우  
...
```

- CPU 시간을 계속 사용하므로 비효율적

GPIO의 스위치 값을 읽는 인터럽트 방법

36

- GPIO 인터럽트
 - 폴링의 소모적인 CPU 사용을 줄임
- 기본 개념
 - 프로그램이 다른 작업을 수행하고 있을 때, 스위치에 rising edge나 falling edge의 이벤트가 발생하면, 인터럽트를 통해 프로그램에게 이벤트 발생을 알려줌
 - 프로그램은 인터럽트가 발생하면 그 때 스위치 값을 읽으면 됨
 - 프로그램은 폴링없이 다른 작업을 수행할 수 있음
- GPIO 라이브러리에서 GPIO 인터럽트 활용 3가지 방법 제공
 - GPIO.wait_for_edge() 함수 활용
 - GPIO.add_event_detect()와 GPIO.event_detected() 함수 활용
 - **add_event_detect() 함수와 콜백 활용**
 - 이 책에서는 이 방법만 다룸

add_event_detect()와 콜백을 활용하여 스위치 눌러짐 탐지

37

- 명료하고 짧은 코드로 바운스 문제 해결
 - ▣ add_event_detect() 함수의 callback과 bouncetime 매개변수 활용

```
add_event_detect(  
    channel,          # channel에 핀 번호 지정  
    event,            # 기다리는 이벤트 지정. GPIO.RISING, GPIO.FALLING, GPIO.BOTH 중 하나  
    callback=my_callback, # my_callback은 이벤트가 발생할 때 호출될 함수 이름  
    bouncetime=200     # 디바운스를 위해 무시할 바운스 시간(ms)  
)  
  
# 사용자가 작성해야 하는 콜백 함수  
def my_callback(channel) : # channel : 이벤트가 발생한 핀 번호  
    ... # 이벤트가 발생할 때 처리하는 코드 작성
```

- ▣ 실행 과정
 - add_event_detect()로부터 리턴한 후
 - 어떤 시점에 event가 발생하면, GPIO 라이브러리는 스레드 생성
 - 생성된 스레드가 callback 매개변수에 지정된 함수 실행
 - 응용프로그램은 add_event_detect() 함수 호출 뒤, 다른 작업 실행
 - 이벤트 발생시 생성된 스레드가 스위치 값을 읽음
- ▣ 바운스 문제 해결
 - bouncetime 매개변수에 디바운스 시간 지정, 디폴트는 200(ms)

예제 8-3 add_event_detect()의 바운스 시간과 콜백 함수를 활용하여 스위치 눌러짐 탐지

38

add_event_detect()에 바운스 시간과 콜백 함수를 활용하여 스위치의 눌러짐을 탐지해보자. 실습 회로(그림 8-14)에서 GPIO21 핀에 연결된 스위치가 눌러지면 "스위치 눌러짐"을 출력하고 종료하는 프로그램을 작성하라.

8-3.py

```
import RPi.GPIO as GPIO
import time

# button이 눌러질 때 호출되는 콜백 함수
def button_pressed(channel):
    print("%d 핀에 연결된 스위치 눌러짐" % channel)

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

button = 21 # GPIO21
GPIO.setup(button, GPIO.IN, GPIO.PUD_DOWN)

# button을 누르면 10ms의 디바운스 후 button_pressed()가 호출되도록 설정
GPIO.add_event_detect(button, GPIO.RISING, button_pressed,
bouncetime=10)
while True:
    pass # 사용자는 필요한 다른 작업을 이곳에 작성한다.
```

```
(myenv) pi@pi:~/ch08 $ python 8-3.py
21 핀에 연결된 스위치 눌러짐
21 핀에 연결된 스위치 눌러짐
21 핀에 연결된 스위치 눌러짐
21 핀에 연결된 스위치 눌러짐
^CTraceback (most recent call last):
  File "/home/pi/ch08/8-3.py", line 15, in
    <module>
    pass # 사용자는 필요한 다른 작업을 이곳
    에 작성한다.
KeyboardInterrupt
(myenv) pi@pi:~/ch08 $
```

예제 8-4(응용) 버튼을 누를 때마다 LED를 켜고 끄는 토글 스위치 만들기

39

실습 회로를 그대로 활용하여, 스위치 버튼을 처음 누르면 LED를 켜고 다시 누르면 LED 끄기를 반복하는 프로그램을 작성하라. `add_event_detect()`와 콜백 함수를 이용하라.

8-4.py

```
import time
import RPi.GPIO as GPIO

# pin에 연결된 LED에 value(0/1) 값을 출력하여 LED를 켜거나 끄는 함수
def led_on_off(pin, value):
    GPIO.output(pin, value)

# button이 눌릴 때 호출되는 콜백 함수
def button_pressed(pin):
    global btn_status # btn_status는 전역 변수임
    global led # led는 전역 변수임
    btn_status = 0 if btn_status == 1 else 1
    led_on_off(led, btn_status)

led = 6 # GPIO6
button = 21 # GPIO21
btn_status = 0 # 현재 버튼이 눌린 상태. 1이면 눌러져 있음

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(led, GPIO.OUT) # led 핀을 출력으로 지정
GPIO.setup(button, GPIO.IN, GPIO.PUD_DOWN) # button 핀을 입력으로 지정

# button을 누르면 10ms의 디바운스 후 button_pressed()가 호출되도록 설정
GPIO.add_event_detect(button, GPIO.RISING, button_pressed, 10)

print("스위치를 누르면 LED가 On되고 다시 누르면 Off 됩니다")
while True:
    pass # 사용자는 필요한 다른 작업이 있으면 이곳에 작성한다.
```

(myenv) pi@pi:~/ch08 \$ python 8-4.py

스위치를 누르면 LED가 On되고 다시 누르면 Off 됩니다

^CTraceback (most recent call last):

File "/home/pi/ch08/8-4.py", line 29, in <module>

pass # 사용자는 필요한 다른 작업을 이곳에 작성한다.

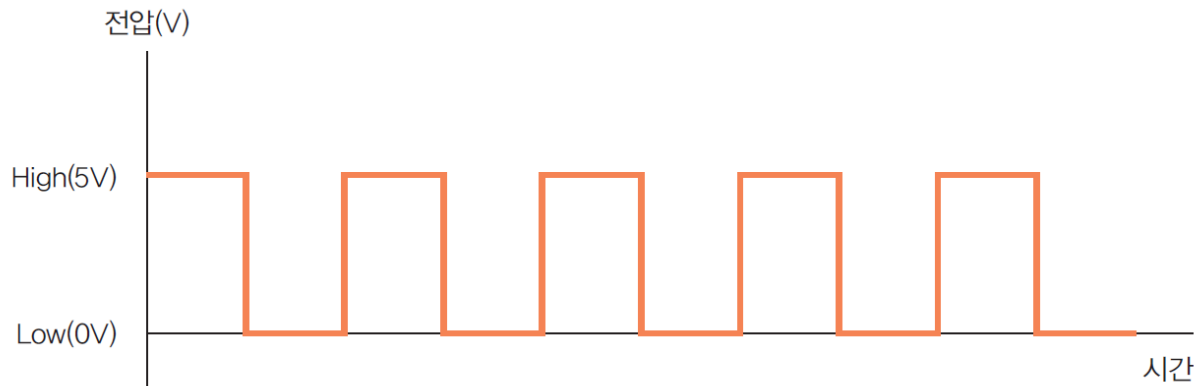
KeyboardInterrupt

(myenv) pi@pi:~/ch08 \$

PWM 제어

40

- PWM(Pulse Width Modulation)
 - ▣ 디지털 신호의 파형 폭을 조절하여 전압이나 전력 제어 방법
 - 디지털 출력으로 아날로그 회로를 제어할 때 주로 사용
- 디지털 파형(pulse, 펄스)
 - ▣ High->Low->High->Low... 로 이어지는 모양의 디지털 신호
 - 구형파, 사각파라고 부름

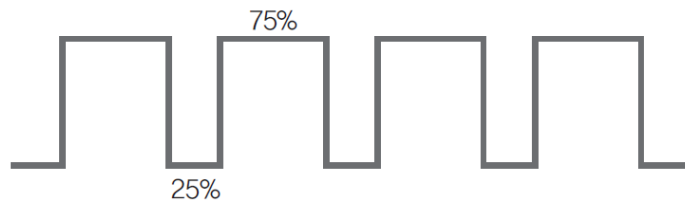


듀티 사이클

41

□ 듀티 사이클(Duty Cycle)

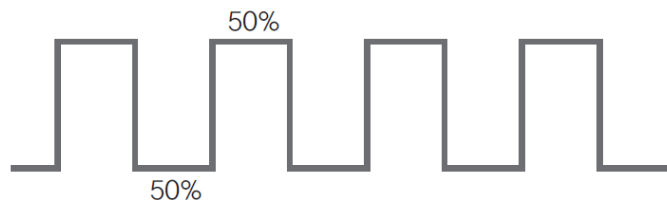
- 파형의 전체 시간에 대한 High 시간의 비율(%)
 - High가 작동 상황을 나타낸다고 할 때, 신호나 시스템이 활성화된 시간의 비율
- 듀티 사이클이 큰 신호일수록,
 - 신호 수신 측의 High(5V) 시간 을 증가 -> 평균적으로 더 높은 전압 전달 -> 수신 측에 LED가 있다면, 더 밝음



(a) 듀티 사이클 75%의 파형



듀티 사이클이 클수록
LED는 더 밝음



(b) 듀티 사이클 50%의 파형



PWM

42

□ 파형 폭(PW, Pulse Width)

- ▣ 파형의 High 길이

□ PWM

- ▣ 파형 폭의 크기를 조절하여 평균 전압을 조절하는 기법
 - 듀티 사이클을 조절하여 평균 전압 조절

□ GPIO PWM 제어 함수들

▣ GPIO.PWM(pin, frequency)

- pwm 객체 생성
- frequency가 크면, 주기는 짧아지고 깜박 꺼림이 빨라짐
- frequency가 작으면, 주기는 길어지고 깜박 꺼림이 느려짐
- 예) **pwm = GPIO.PWM(6, 50)**

- GPIO 6번 핀에 50Hz의 파형을 발생시킬 수 있는 PWM 객체 생성

▣ PWM.start(d)

- PWM 시작, d의 duty cycle로 반복하여 펄스 발생
- 예) **pwm.start(30)**

- pwm 객체에서 50Hz의 주기로 30%의 듀티 사이클의 파형 발생 지시. 파형 발생 시작됨

▣ PWM.ChangeDutyCycle(dutyCycle)

- duty cycle(%) 변경
- dutyCycle에 100을 기준으로 하는 숫자 전달
- 예) **pwm.ChangeDutyCycle(60)**

- 듀티 사이클을 60%로 변경

```
PWM( # PWM 객체를 생성하여 리턴
    pin, # 핀 번호 지정
    frequency # 파형의 주기(단위 Hz). 초당 파형의 개수
)
PWM.start( # 주어진 듀티 사이클의 파형 발생
    dutycycle # 듀티 사이클 값(단위 %)
)
PWM.ChangeDutyCycle( # dutycycle 변경
    dutycycle # 듀티 사이클 값(단위 %)
)
```

예제 8-5 PWM을 이용한 LED 밝기 조정

43

실습 회로(그림 8-14)를 그대로 활용하여, 5초 동안 LED가 점점 밝아지다고 5초 동안 점점 어두워지기를 반복하는 코드를 작성하라.

8-5.py

0.05초 간격으로 듀티 사이클을 0~99까지 변경하고, 다시 0.05초 간격으로 듀티 사이클을 99~0으로 변경하는 과정 반복

(myenv) pi@pi:~/ch08 \$ python 8-5.py

increase the light
decrease the light
increase the light
decrease the light
increase the light
decrease the light
increase the light
decrease the light
increase the light

5초 간격으로 LED가
밝아졌다가 어두워지기를
반복한다.

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

```
def increase(pwm):
```

```
    print("increase the light")
    for value in range(0, 100): # 5초 동안 루프
        pwm.ChangeDutyCycle(value)
        time.sleep(0.05)
```

```
def decrease(pwm):
```

```
    print("decrease the light")
    for value in range(99, -1, -1): # 5초 동안 루프
        pwm.ChangeDutyCycle(value)
        time.sleep(0.05)
```

```
GPIO.setmode(GPIO.BCM)
```

```
GPIO.setwarnings(False)
```

```
led = 6 # GPIO6 핀에 LED 연결
```

```
GPIO.setup(led, GPIO.OUT) # GPIO6 핀을 출력으로 지정
```

```
GPIO.output(led, GPIO.LOW) # GPIO6 핀에 0V 출력
```

```
pwm = GPIO.PWM(led, 100) # GPIO6 핀에 100Hz의 신호를 발생하도록 설정
```

```
pwm.start(0) # GPIO6 핀에 듀티 사이클 0%, 100Hz의 신호 발생 시작
```

```
while True:
```

```
    increase(pwm) # 5초동안 듀티 사이클 증가. LED를 점점 밝게
```

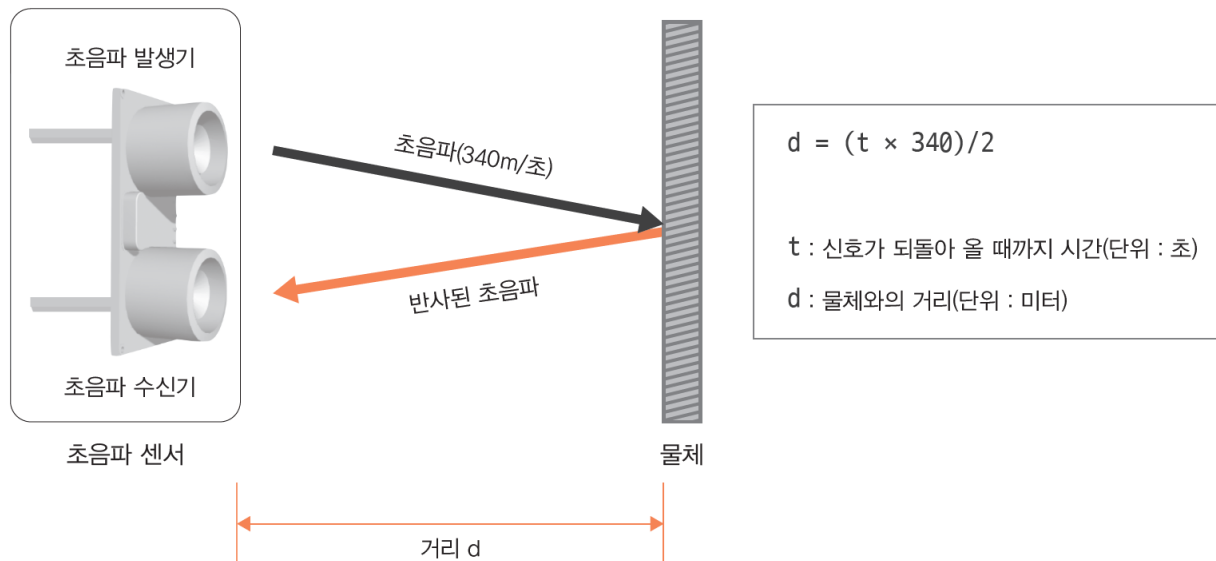
```
    decrease(pwm) # 5초동안 듀티 사이클 감소. LED를 점점 어둡게
```

8.4 초음파 센서 제어

초음파 센서

45

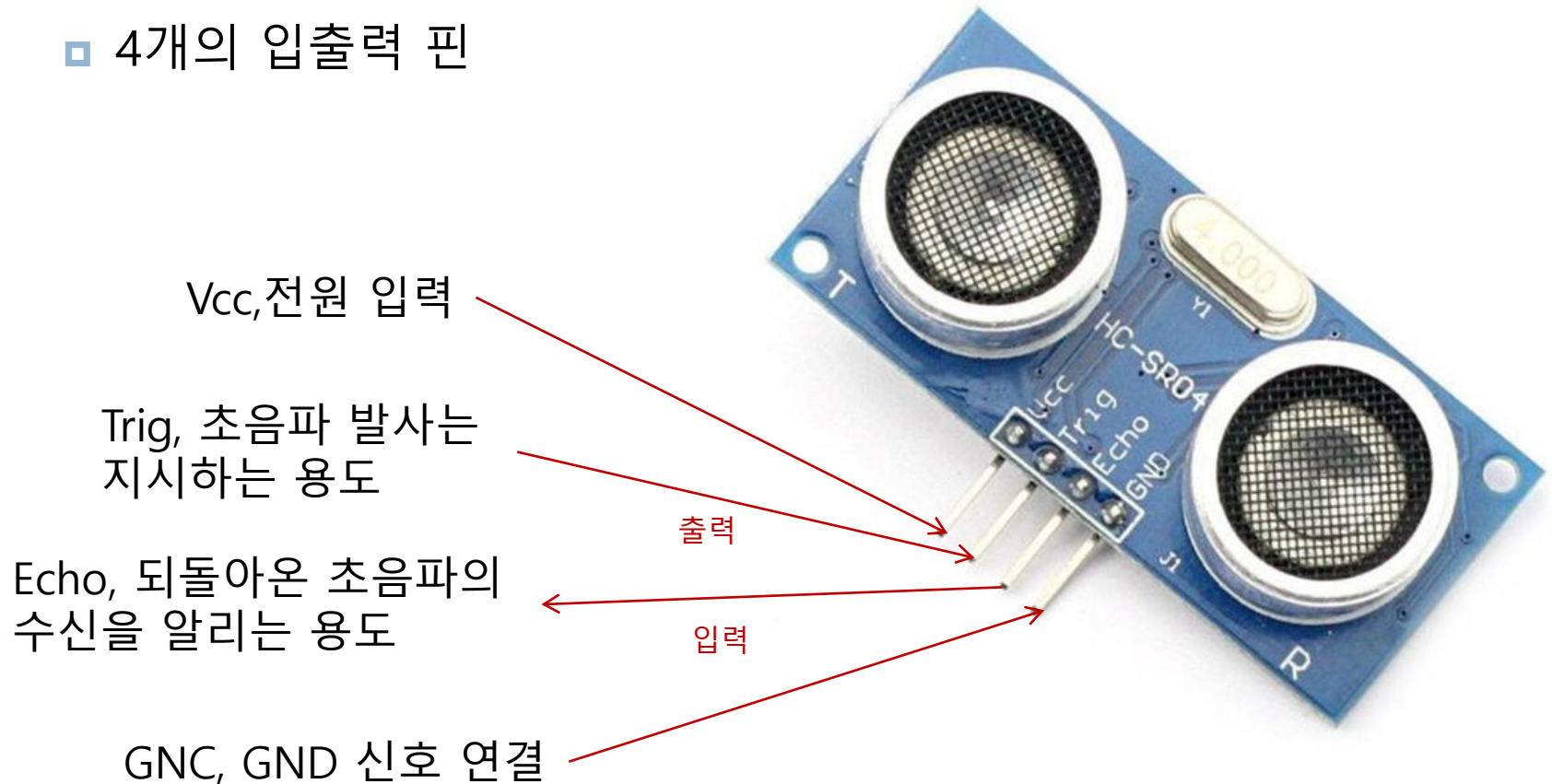
- 초음파 센서(Ultrasonic Sensor)
 - ▣ 초음파를 발사하고 발사된 초음파가 물체에 부딪쳐 되돌아오는 것을 감지하는 센서
 - ▣ 활용
 - 초음파 발사 후 되돌아 온 시간을 측정하여 물체까지 거리 계산
- 초음파 센서 구조
 - ▣ 발생기와 수신기로 구성



초음파 센서, HC-SR04

46

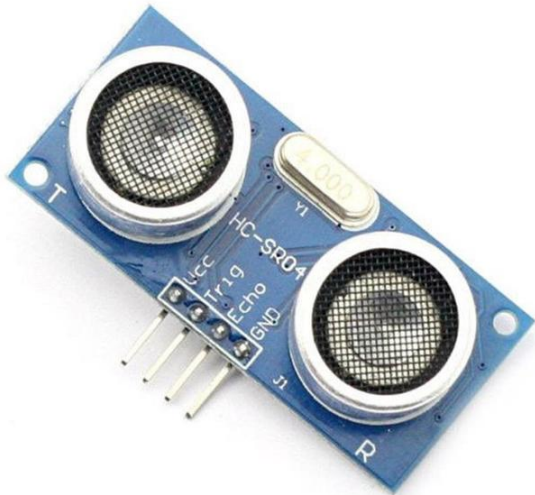
- 초음파 센서, HC-SR04
 - ▣ 저렴하고 실습에 많이 사용하는 초음파 센서
 - ▣ 4개의 입출력 핀



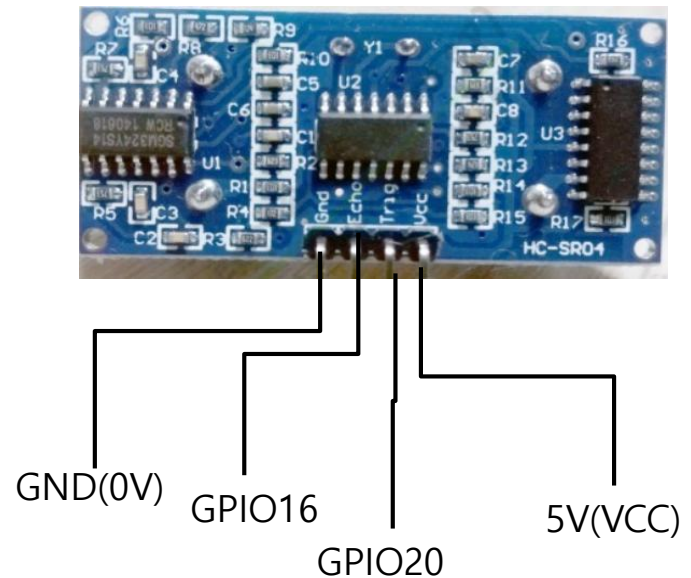
초음파 센서 제어 실습

47

- 실습 내용
 - 초음파 센서를 활용하여 물체와의 거리 계산
- 부품
 - 초음파 센서(HC-SR04) 1개
 - GPIO6 핀 (출력용) - 초음파 센서의 Trig 핀에 연결, 초음파 발사 지시
 - GPIO21 핀 (입력용) - 초음파 센서의 Echo 핀에 연결, 초음파 수신 여부 입력
- 실습 회로 구성



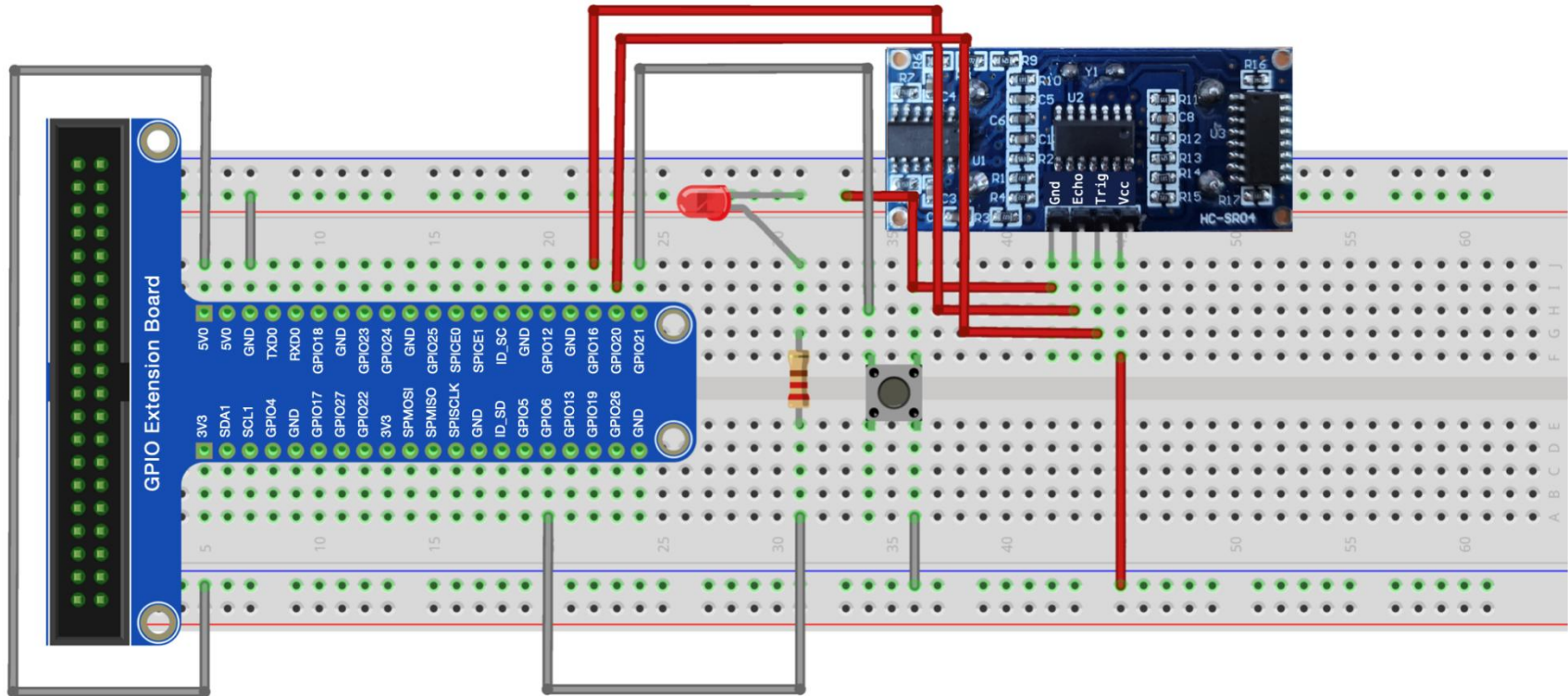
HC-SR04 초음파 센서 앞면



실습 회로 구성

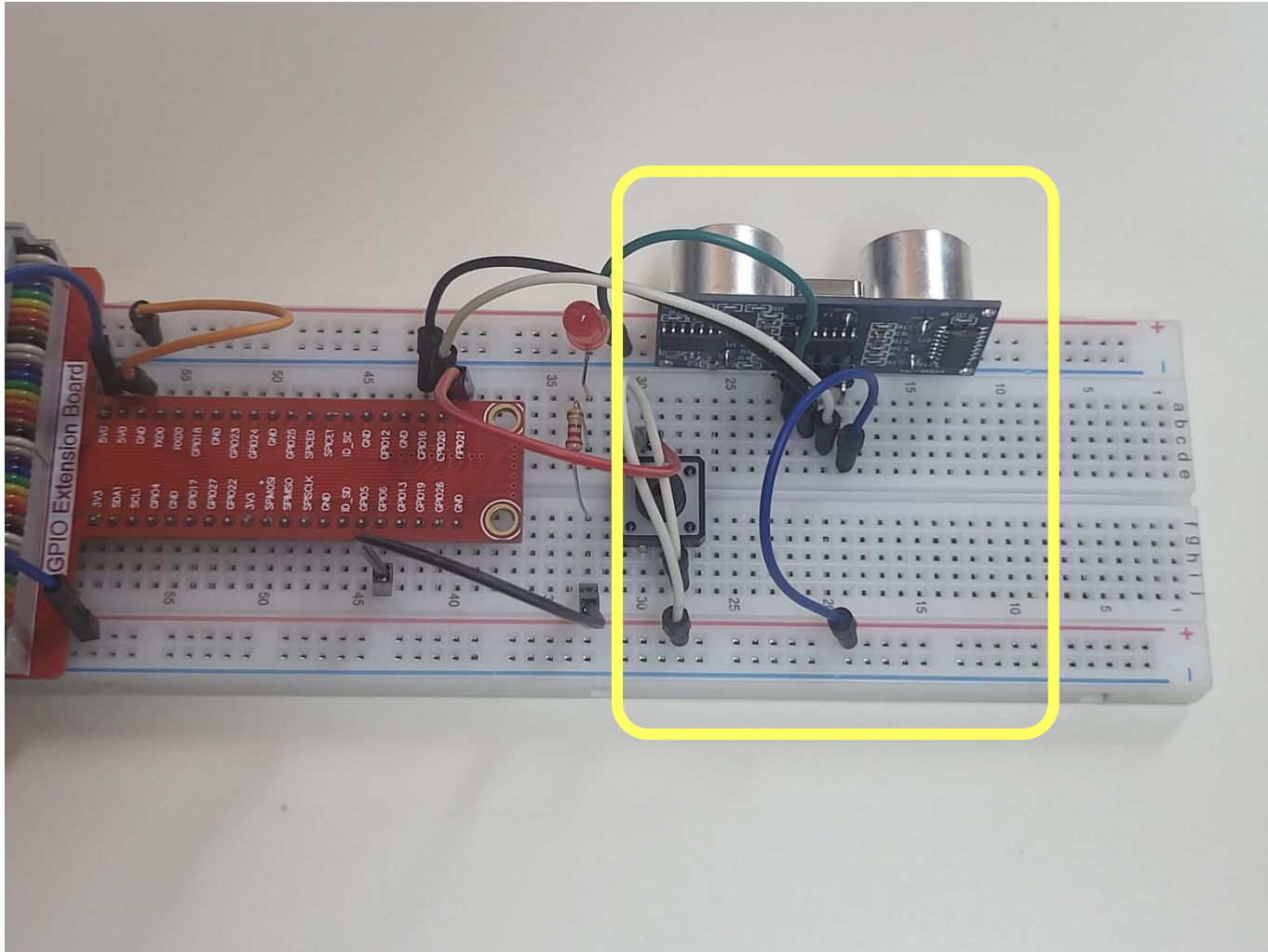
초음파 센서 실험을 위한 회로 구성

48



구성된 회로 사진

49



예제 8-6 초음파 센서를 이용한 물체와의 거리 측정

50

초음파 센서를 이용하여 약 0.5초 간격으로 물체와의 거리를 측정하여 출력하는 프로그램을 작성하라.

```
(myenv) pi@pi:~/ch08 $ python 8-6.py
```

```
물체와의 거리는 3.769398cm 입니다.  
물체와의 거리는 7.607698cm 입니다.  
물체와의 거리는 10.323286cm 입니다.  
물체와의 거리는 10.262489cm 입니다.  
물체와의 거리는 10.752916cm 입니다.
```

8-6.py

```
import time
import RPi.GPIO as GPIO

def measureDistance(trig, echo):
    time.sleep(0.2) # 초음파 센서의 준비 시간을 위해 필연적인 200밀리초 지연
    GPIO.output(trig, 1) # trig 핀에 1(High) 출력
    GPIO.output(trig, 0) # trig 핀 신호 High->Low. 초음파 발사 지시

    while(GPIO.input(echo) == 0): # echo 핀 값이 0->1로 바뀔 때까지 루프
        pass

    # echo 핀 값이 1이면 초음파가 발사되었음
    pulse_start = time.time() # 초음파 발사 시간 기록
    while(GPIO.input(echo) == 1): # echo 핀 값이 1->0으로 바뀔 때까지 루프
        pass

    # echo 핀 값이 0이 되면 초음파 수신하였음
    pulse_end = time.time() # 초음파가 되돌아 온 시간 기록
    pulse_duration = pulse_end - pulse_start # 경과 시간 계산
    return pulse_duration*340*100/2 # 거리 계산하여 리턴(단위 cm)

trig = 20 # GPIO20
echo = 16 # GPIO16
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(trig, GPIO.OUT)
GPIO.setup(echo, GPIO.IN)

while True:
    distance = measureDistance(trig, echo)
    time.sleep(0.5) # 0.5초 간격으로 거리 측정(0.3으로 하면 정확)
    print("물체와의 거리는 %fcm 입니다." % distance)
```