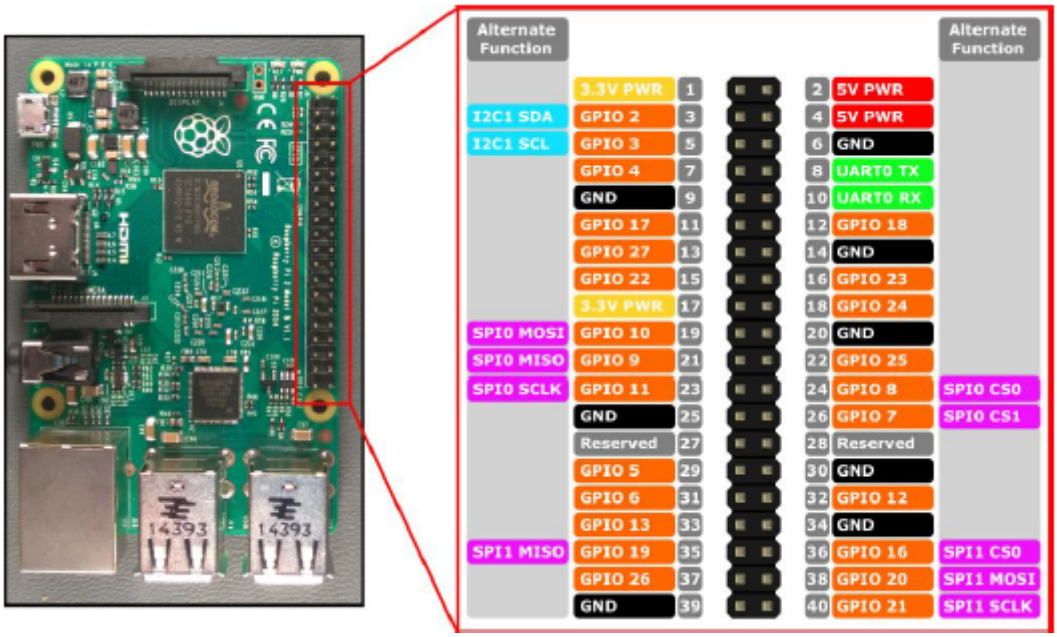
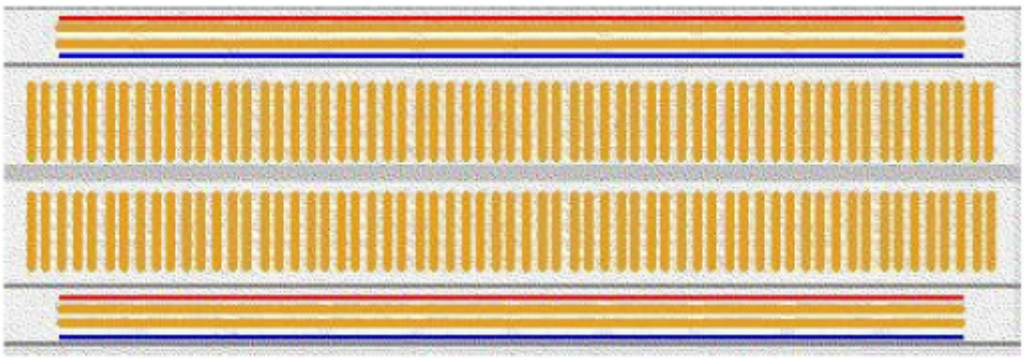
GPIO Pin Map



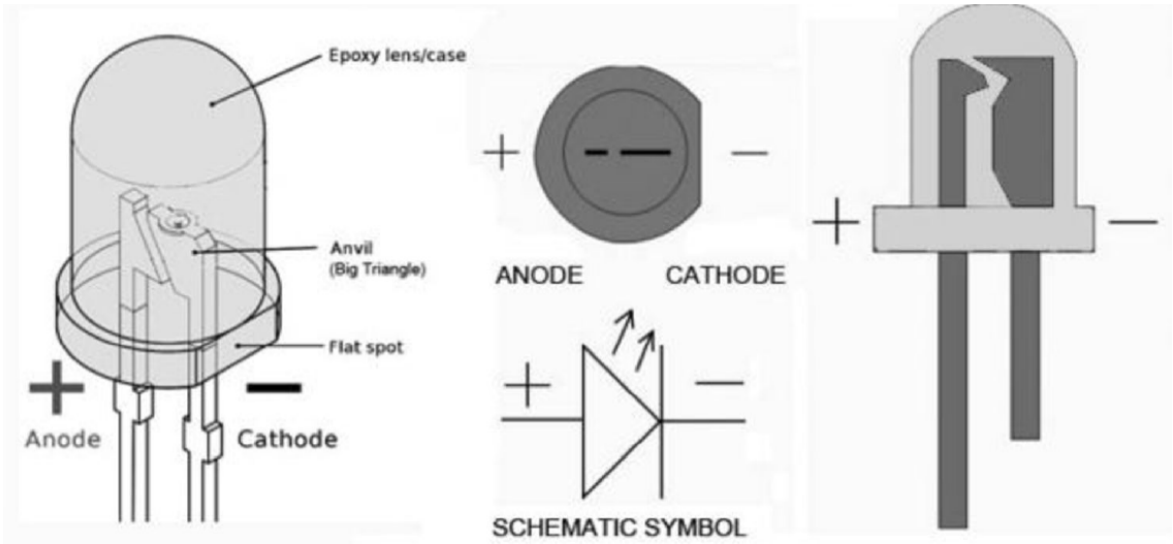
Breadboard

- Used to build and test circuits quickly before finalizing any circuit design



- 각각 가로, 세로로 연결됨

LED



GPIO Commands

- T-shape PCB는 BCM넘버링 사용
    - gpio 커맨드를 사용할때 -g 옵션을 사용해줘야 함

```
gpio readall # gpio 정보 전체 불러오기
gpio mode 22 out # 22번 gpio의 mode를 out으로 설정
gpio write 22 1 # 22번 gpio의 V를 1로 설정
```

LED - On / Off

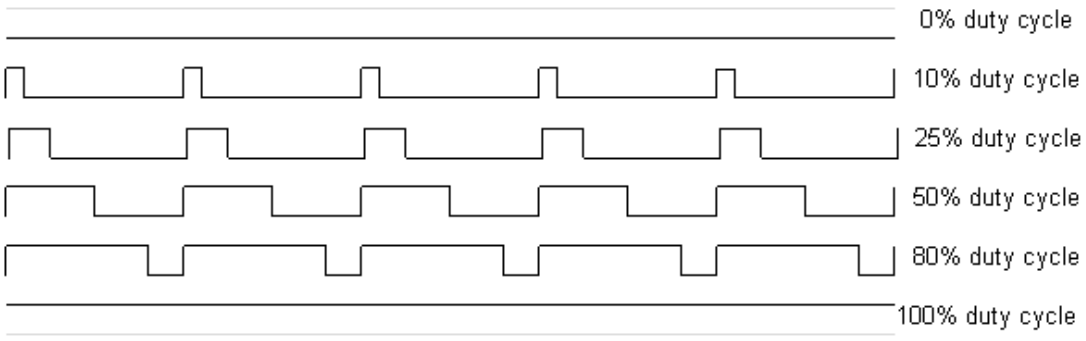
```
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.OUT)

on_time = 0.1 # time led is ON in seconds
off_time = 0.9 # time led is OFF in seconds

while True:
    GPIO.output(17, GPIO.HIGH)
    time.sleep(on_time)
    GPIO.output(17, GPIO.LOW)
    time.sleep(off_time)
```

LED - PWM(Pulse Width Modulation)

- PWM은 빠르게 ON/OFF를 반복하는 것



    - 신호의 지속시간으로 위치를 제어

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time

LedPin = 17

GPIO.setmode(GPIO.BCM) # BCM 넘버링 사용(T자 케이블)
GPIO.setup(LedPin, GPIO.OUT) # Set pin mode as OUT
GPIO.output(LedPin, GPIO.LOW) # Set pin to low(0V)

p = GPIO.PWM(LedPin, 1000) # Set frequency to 1 KHz
p.start(0) # Start PWM output, Duty Cycle = 0
```

```
try:
    while True:
        for dc in range(0, 101, 2): #increase duty cycle : 0 ~ 100
            p.ChangeDutyCycle(dc) # change duty cycle
            time.sleep(0.01)
        time.sleep(0.05)
        for dc in range(100, -1, -2): # decrease duty cycle : 100 ~ 0
            p.ChangeDutyCycle(dc)
            time.sleep(0.01)
        time.sleep(0.05)

except KeyboardInterrupt:
    p.stop()
    GPIO.output(LedPin, GPIO.HIGH) # Turn off all leds
    GPIO.cleanup()
```

C

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <sys/time.h>

const char *PATH_GPIO_EXPORT = "/sys/class/gpio/export";
const char *PATH_GPIO_UNEXPORT = "/sys/class/gpio/unexport";
const char *PATH_GPIO_17_DIRECTION = "/sys/class/gpio/gpio17/direction";
const char *PATH_GPIO_17_VALUE = "/sys/class/gpio/gpio17/value";

#define GPIO_NUM 17
typedef enum {
    OFF = 0,
    ON
} gpio_state ;

void gpio_init();
void gpio_exit();
void set_gpio_state(gpio_state state);
void delay_micro(int delay_micros);

FILE *GPIO_EXPORT;
FILE *GPIO_17_DIRECTION;
FILE *GPIO_17_VALUE;

int main()
{
    int time = 0;
    gpio_state state = OFF;
    gpio_init();
/**************** insert your code here ******************/
    while(1) {
        set_gpio_state(ON);
        delay_micro(1000000); // 1 sec delay
        set_gpio_state(OFF);
        delay_micro(1000000); // 1 sec delay
    }
```

```
/*****************************************************/
    gpio_exit();
}

void gpio_init()
{
    if ((GPIO_EXPORT = fopen(PATH_GPIO_EXPORT, "w")) == NULL) {
        printf("%s open failed\n", PATH_GPIO_EXPORT);
        exit(0);
    }

    fprintf(GPIO_EXPORT, "%d", GPIO_NUM);
    fclose(GPIO_EXPORT);

    if ((GPIO_17_DIRECTION = fopen(PATH_GPIO_17_DIRECTION, "w")) == NULL)
    {
        printf("%s open failed\n", PATH_GPIO_17_DIRECTION);
        exit(0);
    }

    fprintf(GPIO_17_DIRECTION, "out");
    fclose(GPIO_17_DIRECTION);

    if ((GPIO_17_VALUE = fopen(PATH_GPIO_17_VALUE, "w")) == NULL) {
        printf("%s open failed\n", PATH_GPIO_17_VALUE);
        exit(0);
    }
}

void gpio_exit()
{
    FILE *GPIO_UNEXPORT;
    fclose(GPIO_17_VALUE);

    if ((GPIO_UNEXPORT = fopen(PATH_GPIO_UNEXPORT, "w")) == NULL) {
        printf("%s open failed\n", PATH_GPIO_UNEXPORT);
        exit(0);
    }

    fprintf(GPIO_UNEXPORT, "%d", GPIO_NUM);
    fclose((int) GPIO_UNEXPORT);
}

void set_gpio_state(gpio_state state)
{
    fprintf(GPIO_17_VALUE, "%d", state);
    fflush(GPIO_17_VALUE);
}
void delay_micro(int delay_micros)
{
    struct timeval now, pulse;
    int cycles, micros;
    cycles = 0;
    gettimeofday(&pulse, NULL);
    micros = 0;

    while (micros < delay_micros) {
        ++cycles;
```

```
        gettimeofday(&now, NULL);

        if (now.tv_sec > pulse.tv_sec)
            micros = 1000000L;
        else
            micros = 0;

        micros = micros + (now.tv_usec - pulse.tv_usec);
    }
}
```