Sensor Programming
센서 프로그래밍

GPIO

RASPBIAN

GPIO

- **GPIO – General Purpose Input and Output**
  - 범용 입/출력 장치를 지칭하는 용어
  - 일반적으로 AP(application processor)는 물리적인 형태를 가지는 IC로 구현될 때 (IC package) 제한된 개수의 핀을 가짐
  - AP 내부 기능은 물리적인 핀의 숫자보다 많음 (alternated function)
    - Ex) IC pin은 208개이고 내장된 기능이 약 600개가 될 때 약 1:3의 비율.
      - 핀 하나에 최소 3개의 기능과 입/출력 모드 설정까지 5개의 동작 모드를 지정함
      - 핀은 한 순간에 하나의 기능으로만 동작

GPIO

- **GPIO – General Purpose Input and Output** (continued)
  - GPIO는 IC에 내장된 기능과 입/출력 동작을 설정하고, 각 동작의 세부 상태를 설정하는 용도로 사용됨.
  - AP의 물리적인 특정 핀(AP의 물리적인 형태를 가지는 IC 실제 핀 하나에 대해) 의 동작 상태를 입력 또는 출력으로 설정
    - 입력 상태와 출력 상태를 동시에 가질 수 없음
    - 한 순간에는 하나의 동작 상태만을 가짐

→ H/W 제어 프로그램의 기본은 프로그래머가 의도한 시점에 지정된 핀의 상태를 High 또는 Low로 설정하는 것(또는 프로그래머가 의도한 시점에 특정 핀의 상태를 읽을 수 있는 것) → 연결된 장치에 따른 동작 발생(또는 동작 인식)

GPIO

- **GPIO – General Purpose Input and Output** (continued)

  1. 핀 동작 설정 : 신호의 출력 또는 입력

     GPIO는 프로그래머가 사용하는 MCU(micro control unit) 또는 AP의 물리적인 핀에 대해 입력 또는 출력 상태를 지정

  2. 설정된 입력/출력에 대한 MCU 또는 AP에 내장된 기능 선택

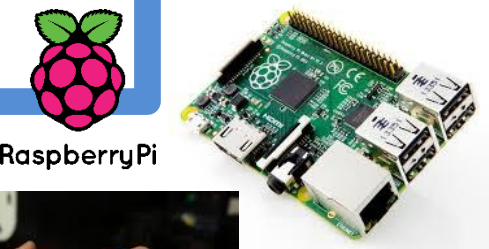
     MCU 또는 AP 에 따라 입/출력과 기능 선택을 같은 단계에서 지정할 수 있음

GPIO

BCM	wPi	Name	Peri #1	Pin No		Peri #1	Name	wPi	BCM
				1	2		5V		
2	8	SDA_1	SHT20_SDA1	3	4		5V		
3	9	SCL_1	SHT20_SCL1	5	6		GND		
4	7	GPIO_7	DCMotor_P	7	8	LED1, UART	TxD	15	14
		GND		9	10	LED2, UART	RxD	16	15
17	0	GPIO_0	LCD_D4	11	12	LCD_D5	GPIO_1	1	18
27	2	GPIO_2	LCD_D6	13	14		GND		
22	3	GPIO_3	LCD_D7	15	16	TLCD_RS	GPIO_4	4	23
				17	18	TLCD_RW	GPIO_5	5	24
10	12	MOSI	EE_MOSI	19	20		GND		
9	13	MISO	EE_MISO	21	22	DCMotor_N	GPIO_6	6	25
11	14	SCLK	EE_SCLK	23	24	EE_CS	CE_0	10	8
		GND		25	26	ADC_CS	CE_1	11	7
0	30	SDA_0		27	28		SCL_0	31	1
5	21	GPIO_21	JOG_UP	29	30		GND		
6	22	GPIO_22	JOG_DN	31	32	DC Moto PWM	GPIO_26	26	12
13	23	GPIO_23	PIEZO1	33	34		GND		
19	24	GPIO_24	IRLED_IN	35	36	JOG_LT	GPIO_27	27	16
26	25	GPIO_25	TLCD_E	37	38	JOG_RT	GPIO_28	28	20
		GND		39	40	JOG_CENTER	GPIO_29	29	21

Sensor Programming
센서 프로그래밍

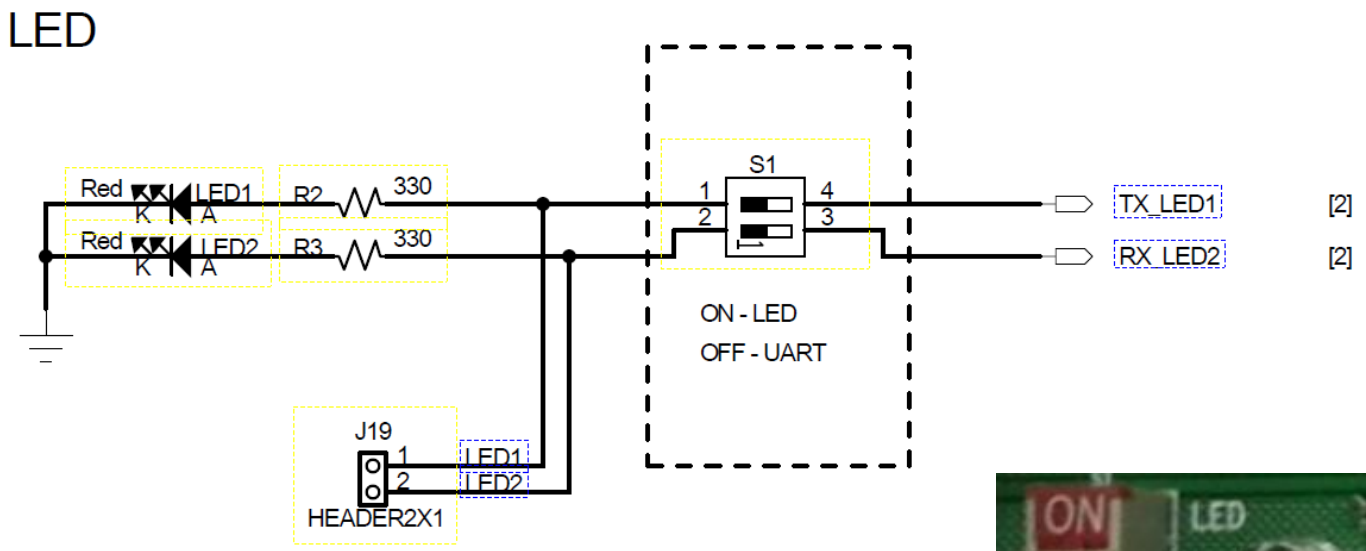
GPIO LED



RaspberryPi

RASPBIAN

GPIO LED



- TX_LED1는 BCM GPIO14와 연결
- RX_LED2는 BCM GPIO15와 연결
- 스위치를 사용하여 LED와 UART를 선택하여 사용할 수 있음
- 출력 전용 회로에 330옴 저항이 연결된 이유는?
  - LED 스팩의 최대 허용 볼트가 2.6v인 반면 GPIO는 3.3V
- J19 는 오실로 스코프로 신호 값 측정을 위한 용도

GPIO LED

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

led_pin1 = 14
led_pin2 = 15

GPIO.setup(led_pin1, GPIO.OUT)
GPIO.setup(led_pin2, GPIO.OUT)

try:
    while True:
        GPIO.output(led_pin1, False)
        GPIO.output(led_pin2, False)
        time.sleep(1)
        GPIO.output(led_pin1, True)
        GPIO.output(led_pin2, True)
        time.sleep(1)
finally:
    print("Cleaning up")
    GPIO.cleanup()
```

GPIO LED

- import RPi.GPIO as GPIO
  - 라즈비안 OS에 기본 포함된 GPIO 라이브러리
- import time
  - 시간 지연 함수를 사용하기 위해 포함
- GPIO.setmode(GPIO.BCM)
  - 확장 커넥터의 핀 번호 할당 방식을 지정
- LED GPIO 핀 번호 지정
  - led_pin1 = 14
  - led_pin2 = 15
- LED 1의 GPIO 를 on/off로 설정
  - GPIO.output(led_pin1, False)
  - GPIO.output(led_pin1, True)

GPIO LED

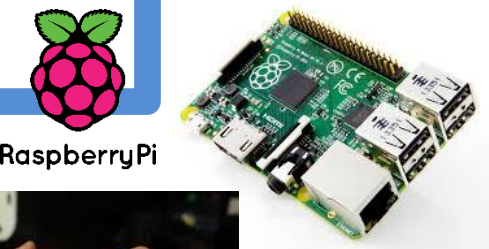
- time.sleep(1)
  - 1초 시간 지연함수
  - 100ms 는 0.1

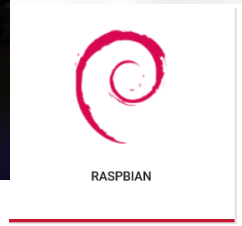
- GPIO.cleanup()
  - 사용된 모든 gpio의 출력 설정을 초기 상태의 입력으로 되돌림

- try/finally 구문
  - 프로그램이 시작되면 try 구문을 실행함
  - Ctrl+ C가 입력되면 finally 구문이 실행됨

Sensor Programming
센서 프로그래밍

GPIO 모터



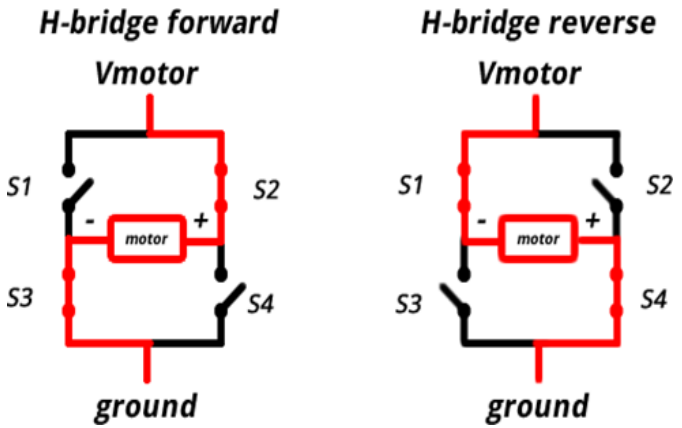


GPIO 모터

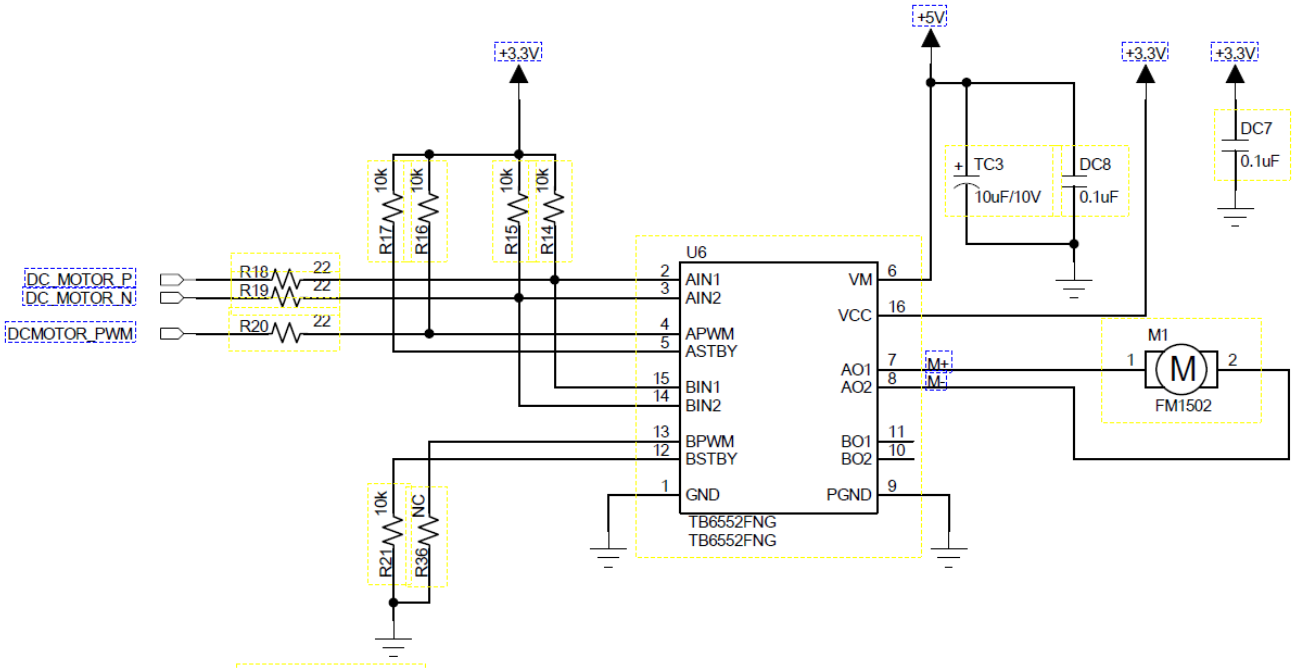
- **DC Motor 구동**
  - DC Motor는 공급 전류에 비례하여 회전 속도 및 회전력이 증가함
  - DC Motor가 구현되면 내부 저항이 고정되므로 공급 전압에 따라 전류가 비례함
  - 따라서 공급 전압에 따라 회전력, 회전 속도가 변함.
  - LED 밝기 제어 예제와 같은 코드를 사용하여 일정한 짧은 시간 간격으로 ON 구간과 OFF 구간의 비율을 달리하여 DC modor에 공급되는 전압을 가변하는 것과 동일한 효과 발생 → 회전 속도 제어

GPIO 모터

- **DC Motor의 구동 회로**
  - DC motor의 회전 방향은 두 단자에 공급하는 전원의 극성에 따라 CW/CCW로 결정
  - 회전 방향을 반전 시키려면 공급 전원의 극성을 변경해야 함
  - H-Bridge
    - 전자 회로에서 4개의 스위치를 사용하여 ON/OFF 설정으로 DC 모터의 단자에 공급되는 전원의 극성을 제어할 수 있는 회로
    - DC motor를 회로 가운데 배치하고 4개의 스위치(Transistor)를 H 자 모양으로 배치한 것에서 유래됨

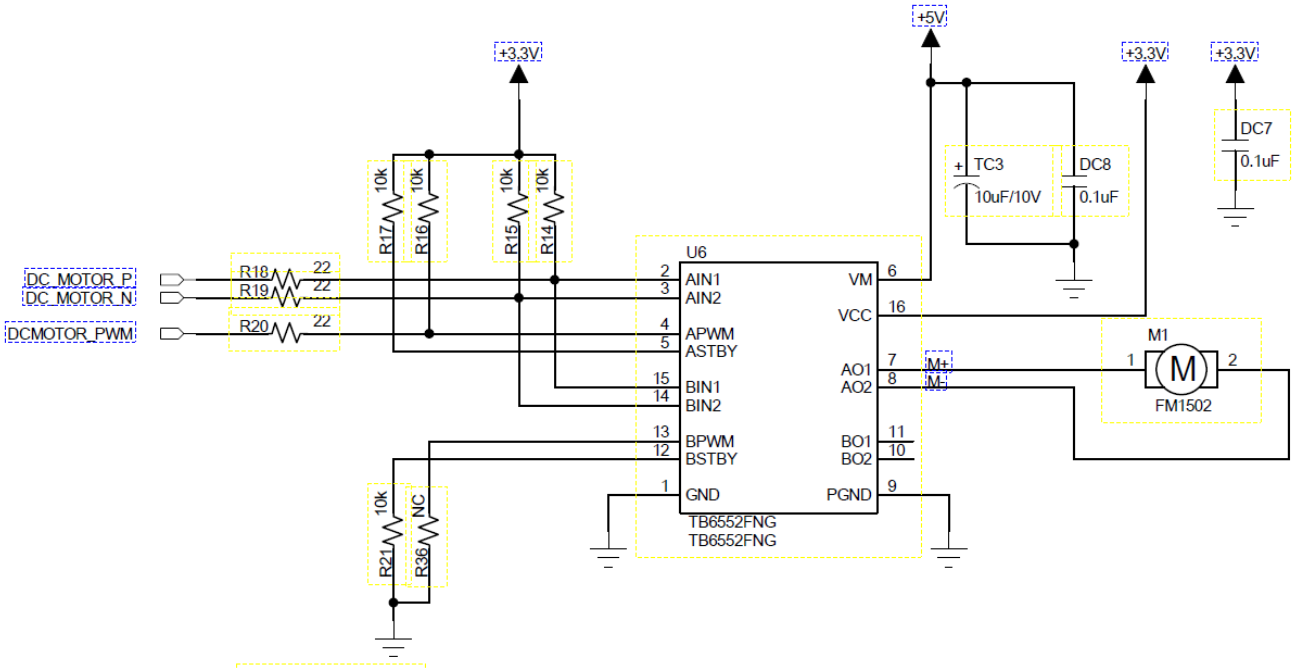


GPIO 모터



- VM : 모터의 파워를 위한 전원 (전류는 PGND로 흐름)
- VCC : 모터 드라이버의 컨트롤 로직의 전원 (전류는 GNC로 흐름)
- 캐패시터
  - TC3, DC8, DC7 소자는 캐패시터라고 부름
  - 갑자기 최대 속도로 모터를 회전하면 순간적으로 전류를 많이 소비하게 됨
  - 이런 상황이 발생하면 전류가 부족하여 일시적으로 모터를 구동할 수 없게 됨
  - 전류를 일정 용량 저장하여 공급 전원의 안정화 역할을 함

GPIO 모터



- TB6652FNG는 채널이 2개임 (A채널, B채널) 모터 2개를 연결 가능, 현재 회로는 A채널만 사용함.
- AIN1, AIN2 : 모터의 방향을 위한 설정을 위한 핀 (AP쪽으로 연결)
- APWM : 모터의 속도 설정을 위한 핀 (AP쪽으로 연결)
- ASTBY : 전류를 절약하기 위한 설정 핀, 현재는 사용 안함으로 3.3V에 연결됨
- AO1 : 모터는 극성이 없으므로 모터의 핀 중 한쪽을 연결
- AO2 : 모터의 핀 중 다른 한쪽을 연결

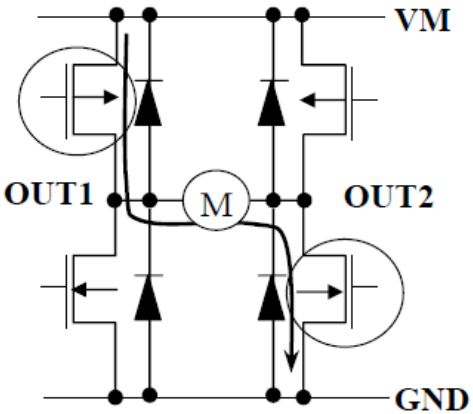
GPIO 모터

- **DC Motor의 구동 실습**
  - 실습 B/D의 DC Motor 구동 회로
    - H-Bridge 를 사용한 구동 회로
    - 모터는 입력의 DC_MOTOR_P, DC_MOTOR_N, DCMOTOR_PWM 신호에 동작
      - DC_MOTOR_P, DC_MOTOR_N 는 각각 High, Low 또는 Low, High의 조합일 경우에만 동작
        - CW (Clock Wise), CCW (Counter Clock Wise)로 회전 방향 설정
      - DC_MOTOR_P, DC_MOTOR_N 가 동작 상태를 만족할 때 DCMOTOR_PWM 에 따라 회전
        - High일 때 최대 속도, Low일 때 정지

GPIO 모터

- **DC Motor의 구동 실습** (continued)
  - 실습 보드의 DC motor 구동 제어
    - 입력 신호에 따른 출력 동작

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby



<ON>

GPIO 모터

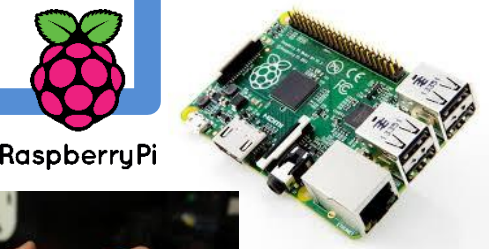
```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

GPIO_RP = 4
GPIO_RN = 25
GPIO_EN = 12

GPIO.setup(GPIO_RP, GPIO.OUT)
GPIO.setup(GPIO_RN, GPIO.OUT)
GPIO.setup(GPIO_EN, GPIO.OUT)
```

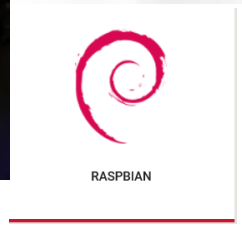
GPIO 모터

```
try:
    while True:
        print 'forword'
        GPIO.output(GPIO_RP, True)
        GPIO.output(GPIO_RN, False)
        GPIO.output(GPIO_EN, True)
        time.sleep(1)

        print 'stop'
        GPIO.output(GPIO_EN, False)
        time.sleep(1)

        print 'backword'
        GPIO.output(GPIO_RP, False)
        GPIO.output(GPIO_RN, True)
        GPIO.output(GPIO_EN, True)
        time.sleep(1)

        print 'stop'
        GPIO.output(GPIO_EN, False)
        time.sleep(1)

finally:
    GPIO.cleanup()
```

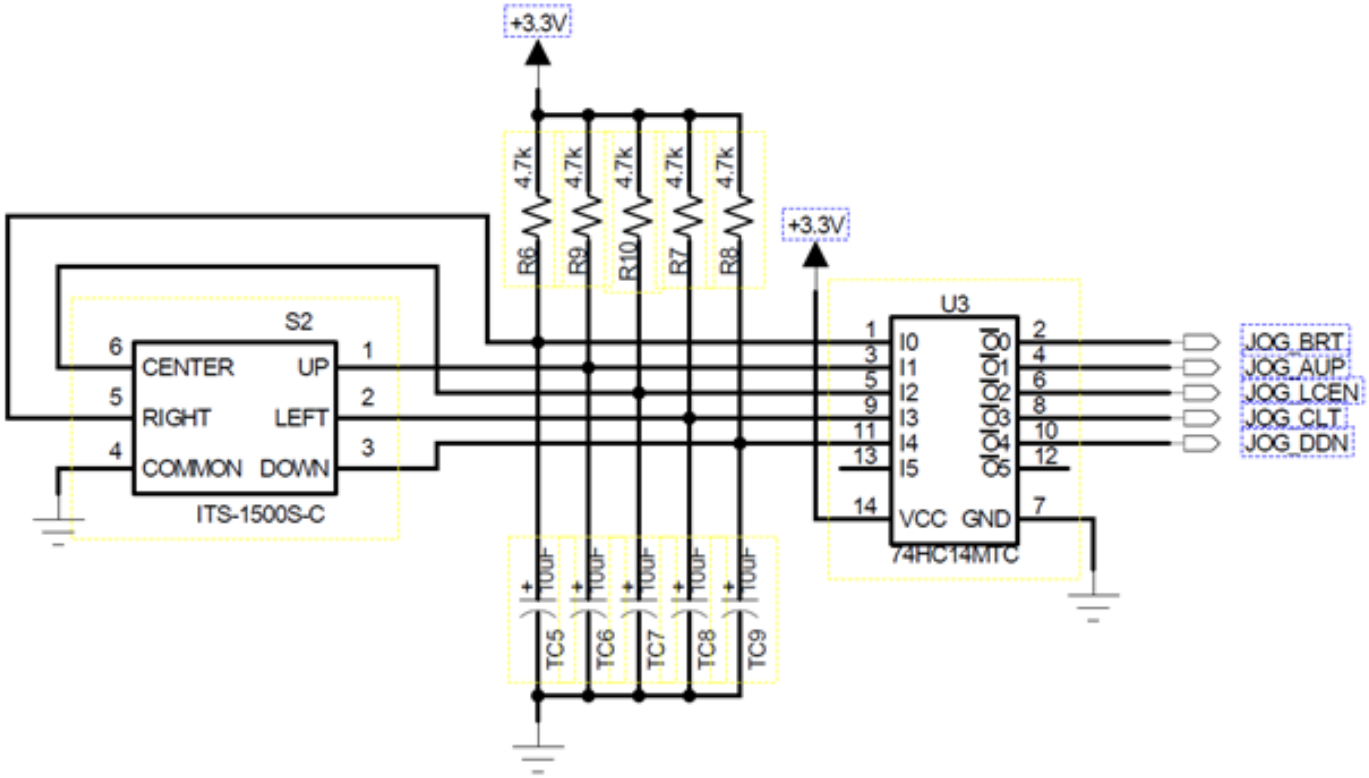
Sensor Programming
센서 프로그래밍

GPIO 조이스틱





GPIO 조이스틱



- ITS-1500S-C : 4방향과 center 의 입력 감지가 가능한 부품
- R6 ~ R10
  - 3.3v에 연결되어 있으므로 풀업 저항이라 부름 (GND에 연결되면?)
  - 입력 회로에 연결될 경우 초기 상태를 HI신호로 결정하기 위함
  - 동작이 될 경우(버튼이 감지되면) 는 low신호가 되므로 low active라고 함
- 74HC14MTC : 채터링 (입력신호의 노이즈로 인한 오동작) 방지를 위해 사용됨

GPIO 조이스틱

```
import RPi.GPIO as GPIO
import time


GPIO.setmode(GPIO.BCM)
gpio = [  5,  6, 16, 20,  21]
stat = [  0,  0,  0,  0,   0]


def print_jog_all():
   print ('up : %d, down: %d, left: %d, right : %d, cen: %d'₩
      %(stat[0], stat[1], stat[2], stat[3], stat[4]))
```

GPIO 조이스틱

- gpio = [5, 6, 16, 20, 21]
  - BCM gpio pin 번호를 저장하기 위한 리스트 정의 (반복문을 활용을 위함)
  - up, down, left, right, centor 순으로 정의
- stat = [0, 0, 0, 0, 0]
  - 현재의 값을 위해 저장할 리스트 정의
- def print_jog_all():
  - 이전값과 현재값이 변경될 경우 각 gpio의 상태를 모두 출력하기 위한 용도

GPIO 조이스틱

```
try:
    for i in range(5):
        GPIO.setup(gpio[i], GPIO.IN)

    cur_stat = 0

    while True:
        for i in range(5):
            cur_stat = GPIO.input(gpio[i])
            if cur_stat != stat[i]:
                stat[i] = cur_stat
                print_jog_all()

finally:
    print("Cleaning up")
    GPIO.cleanup()
```

GPIO 조이스틱

- GPIO.setup(gpio[i], GPIO.IN)
  - 리스트의 모든 GPIO를 입력으로 설정
- cur_stat = 0
  - 조작전 출력을 하지 않기 위한 초기값 설정
- while True:
  - 무한 반복
- for i in range(5):
  - 폴링으로 모든 핀의 변경을 감시
- cur_stat = GPIO.input(gpio[i])
  - 해당 핀의 현재 값을 읽어 cur_stat 에 저장함
- if cur_stat != stat[i]:

  stat[i] = cur_stat

  print_jog_all()
  - 이전 값과 현재 값이 다를 경우(변경된 경우) 현재 값을 stat[i]에 업데이트 후 출력