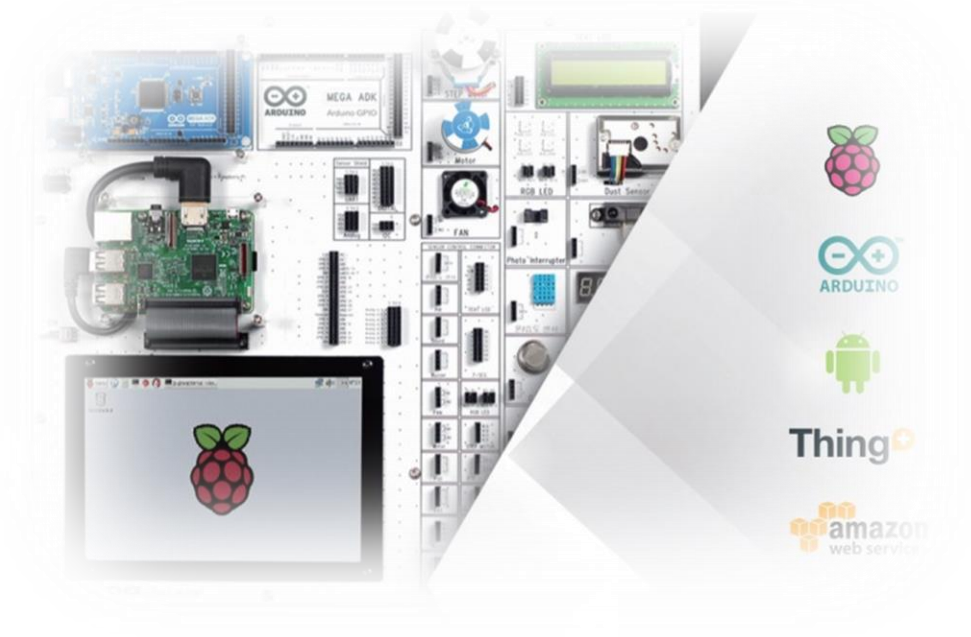
2021년도 2학기
휴먼스마트기기설계



교과목명 : 휴먼스마트기기설계 (01)

담당교수 : 이 수 형

E-mail : soohyong@uu.ac.kr

교재명 : 스마트기기 개발을 위한 사물인터넷, 이수형. (LINC+ 사업단 배포)

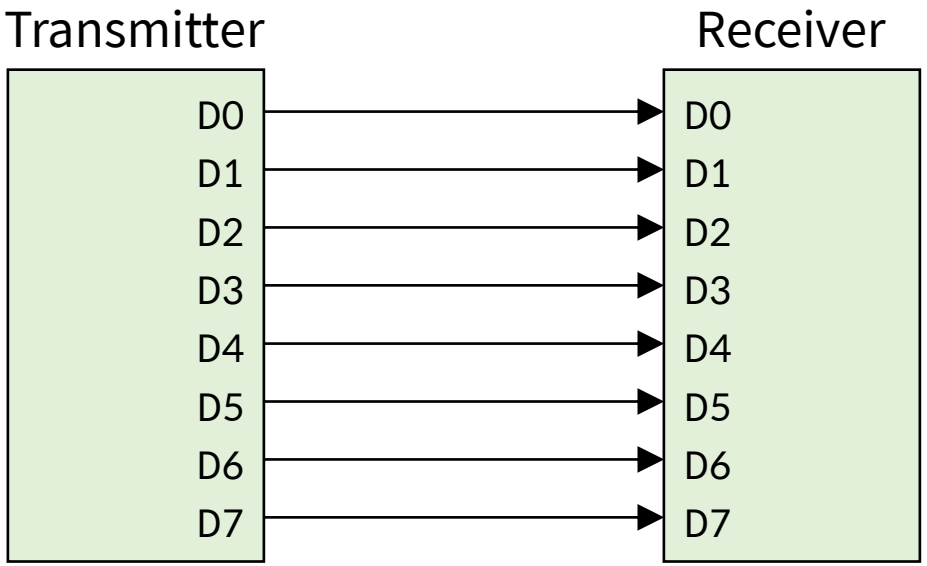
5. 디지털 통신

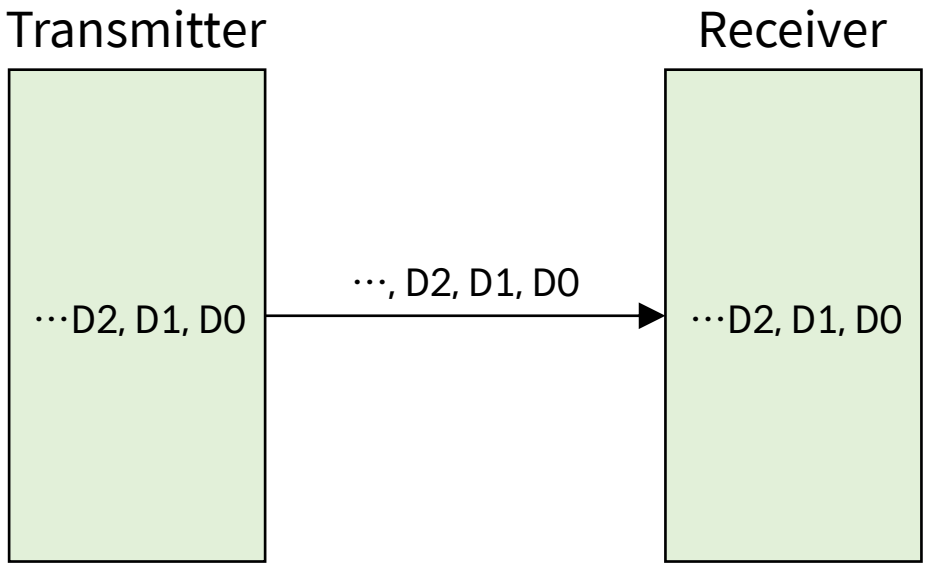
5. 디지털 통신

- 통신 (communication)
  - 송신자와 수신자가 상호 간에 데이터(정보)를 주고 받는 행위
  - 방식 : 아날로그 통신 (라디오, AM/FM), 디지털 통신
    AM : Amplitude Modulation, FM : Frequency Modulation
- 디지털 통신
  - 무선 디지털 통신
    - 송신 : 디지털 정보를 반송주파수(carrier)로 변조(modulation)한 후 무선으로 정보를 전송
    - 수신 : 수신된 신호를 복조(demodulation) 후 디지털 정보로 변환
    - 예 : 디지털 TV 방송, 무선인터넷(WiFi), 블루투스(Bluetooth)
  - 유선 디지털 통신
    - 2개 이상의 기기들이 서로 데이터를 주고 받음
    - 방식 : 병렬 통신, 직렬 통신

디지털 통신 방식

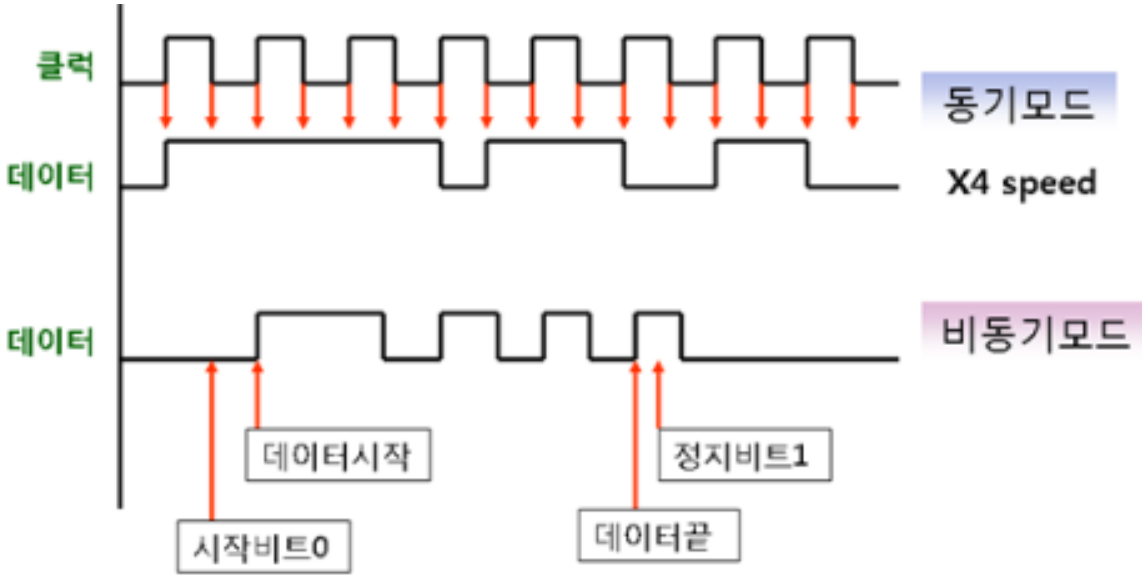
- 병렬 통신 vs. 직렬 통신
  - 병렬 통신 : 전송하고자 하는 모든 데이터(예: 1 바이트)를 선으로 연결하여 전송
    - ➢ 고속 전송 → 최근 : 초고속(수 GHz 이상)의 전송은 동기화가 어려움 (컴퓨터 내부만 사용)
  - 직렬 통신 : 데이터를 한 비트씩 순차적으로 전송
    - ➢ 하나의 선을 통하기 때문에 간단하게 구현
    - ➢ 저속 → 최근 : 프로세서의 처리속도 발전 및 통신 기술의 발달로 고속화가 가능해짐





직렬 통신의 방법

- 동기(Synchronous) 통신
  - 송신기와 수신기의 동기를 맞춰주는 Clock 신호를 추가로 연결하여 사용함
- 비동기(Asynchronous) 통신
  - 동기 신호 없이 통신 → 데이터의 동기를 맞춰주는 추가적인 방법을 사용해야 함
    ⇒ 데이터의 중간에 동기 신호 (start bit, stop bit)를 추가로 전송
  - 미리 통신 속도를 알고 있어야 함
- 대표적인 통신 방식
  - UART, I2C, SPI

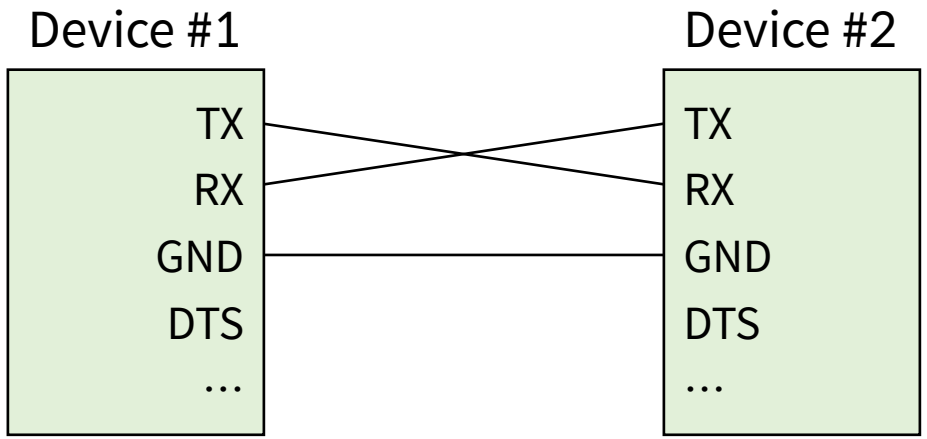


$11101010_{(2)}$ 데이터를 직렬전송시 비동기와 동기식의 비교도

그림 출처 : https://m.blog.naver.com/scw0531/220619256523

UART

- UART (Universal Asynchronous Receiver/Transmitter)
  - 미리 정해놓은 통신 속도(baud)에 맞추어서 송신하고자 하는 데이터를 직렬화하여 전송하고 수신한 데이터를 병렬화하여 데이터를 복원하는 방법
  - TX(Transmitter), RX(Receiver) 라인(line)을 사용하여 통신
  - 경우에 따라 부가적인 제어선(DTS/DTR 등)를 사용하기도 함
  - 종류 : RS-232, RS-422, RS-485

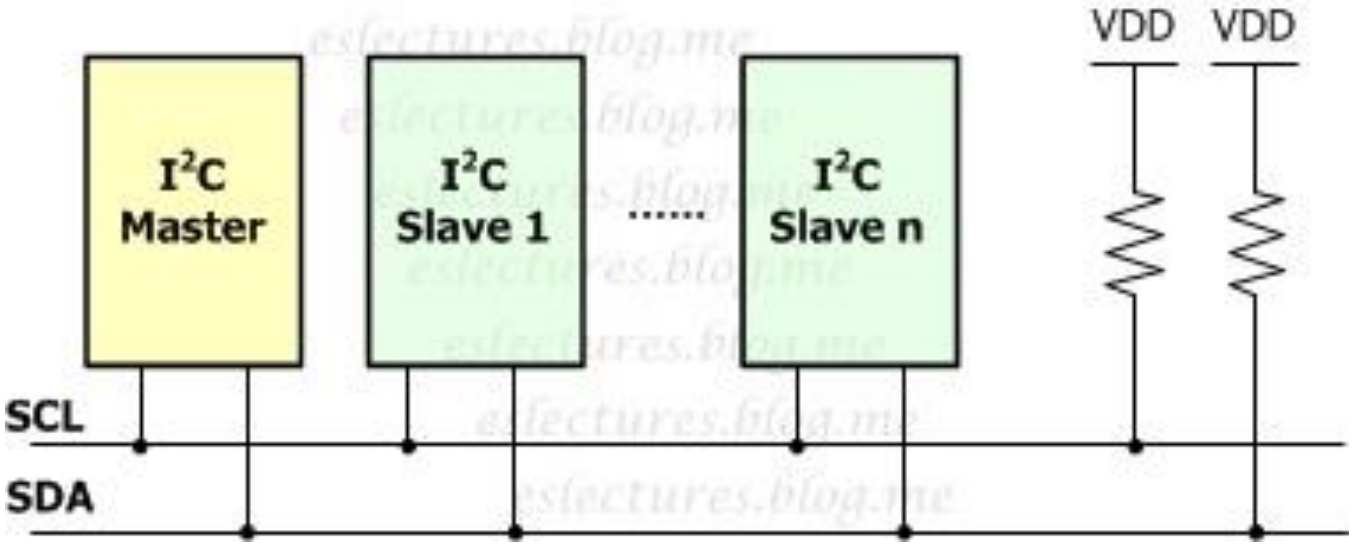


아두이노, NodeMCU 등에서는
RS-232 통신을 사용
(※ 이전 자료 참고)

참고 : http://blog.daum.net/shbaek1009/6766113

I2C

- IIC (Inter-Integrated Circuit)
  - I^2C (I-square C), 마이크로프로세서와 저속의 주변장치와의 통신을 위한 용도로 Philips에서 개발한 규격
  - 2개의 선을 사용하여 TWI(Two-Wire Interfaced)라고 불리기도 함
  - 장치들이 SCL(Serial Clock), SDA(Serial Data) 선을 공유함



- NodeMCU
  D1 : SCL, D2 : SDA
- Arduino
  A5 : SCL, A4 : SDA

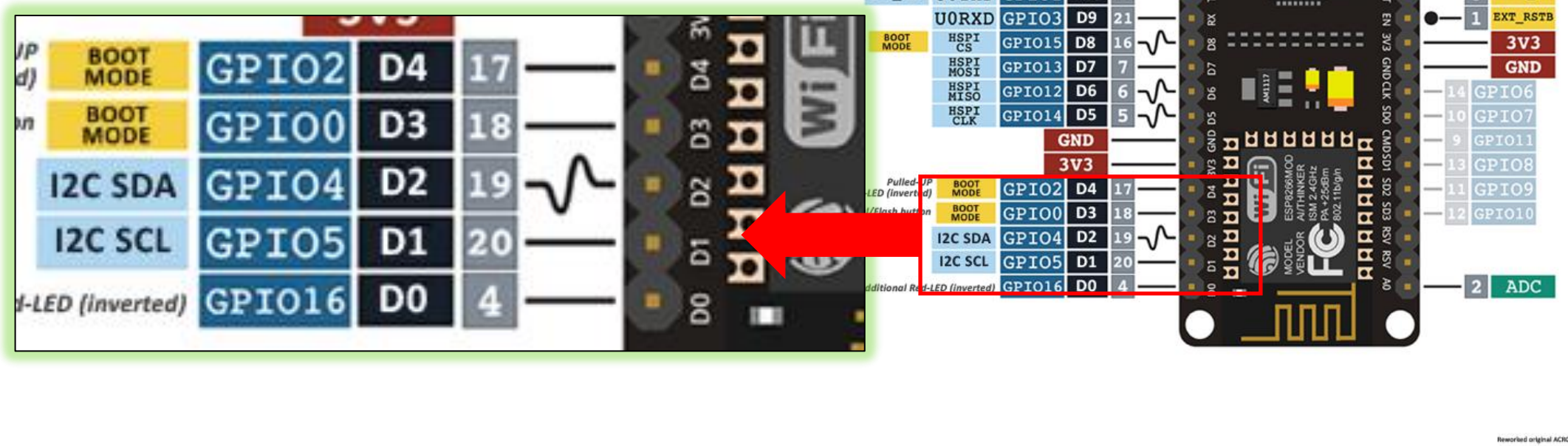
I2C

- IIC (Inter-Integrated Circuit)
  - NodeMCU : D1 (GPIO5) 이 SCL, D2 (GPIO4)가 SDA로 사용됨
    (아두이노와는 다르므로 주의해서 사용)
  - 장치들이 SCL(Serial Clock), SDA(Serial Data) 선을 공유함
    ⇒ 각각의 장치들은 고유한 주소(7bit address)를 사용하여 통신
    ⇒ 동일한 장치를 2개 이상 사용하는 경우 주소 충돌 발생
    ⇒ I2C 다중화기(multiplexer)사용하여 해결
  - 상대적으로 저속 통신 (최대 100kHz)

- NodeMCU
  D1 : SCL, D2 : SDA
- Arduino
  A5 : SCL, A4 : SDA

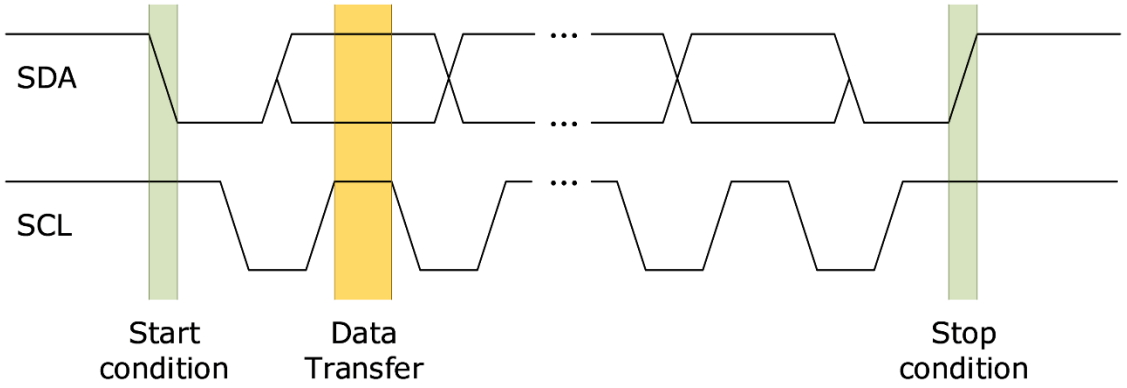
I2C

- NodeMCU의 I2C 연결 핀
  - SCL : D1 (GPIO5)
  - SDA : D2 (GPIO4)

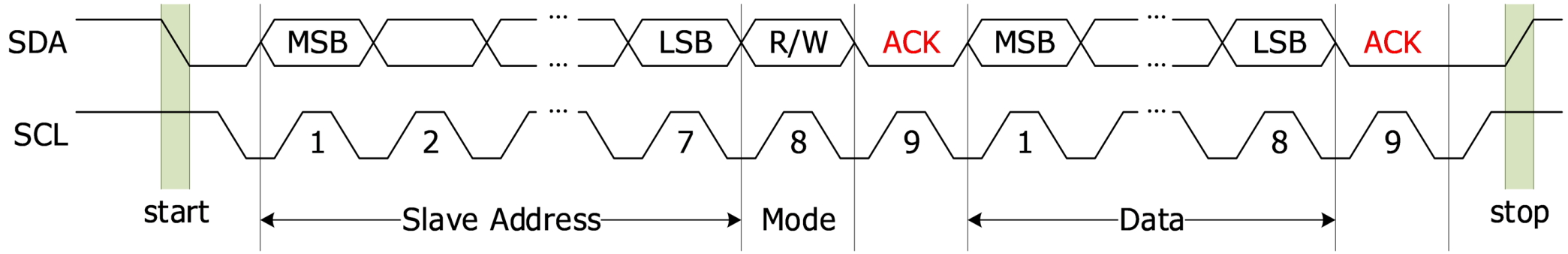


I2C 통신 프로토콜

- 통신 방법
  - Master : SCL 생성
    - ➢ 시작/종료 : SCL = HIGH일 때 SDA 변경
  - Master → Slave
    - ➢ 주소 + 데이터 전송

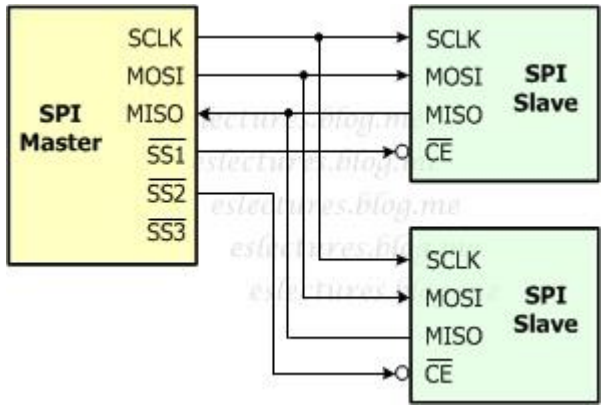


  - Master(주소 + RW) + Slave(ACK) + Master (데이터) + Slave(ACK)
    - ➢ 주소 : 7bit 사용 + R/W 신호 ⇒ 총 8bit, 데이터 : 8bit
    - ➢ ACK (Acknowledgement) : 데이터 수신 응답 신호



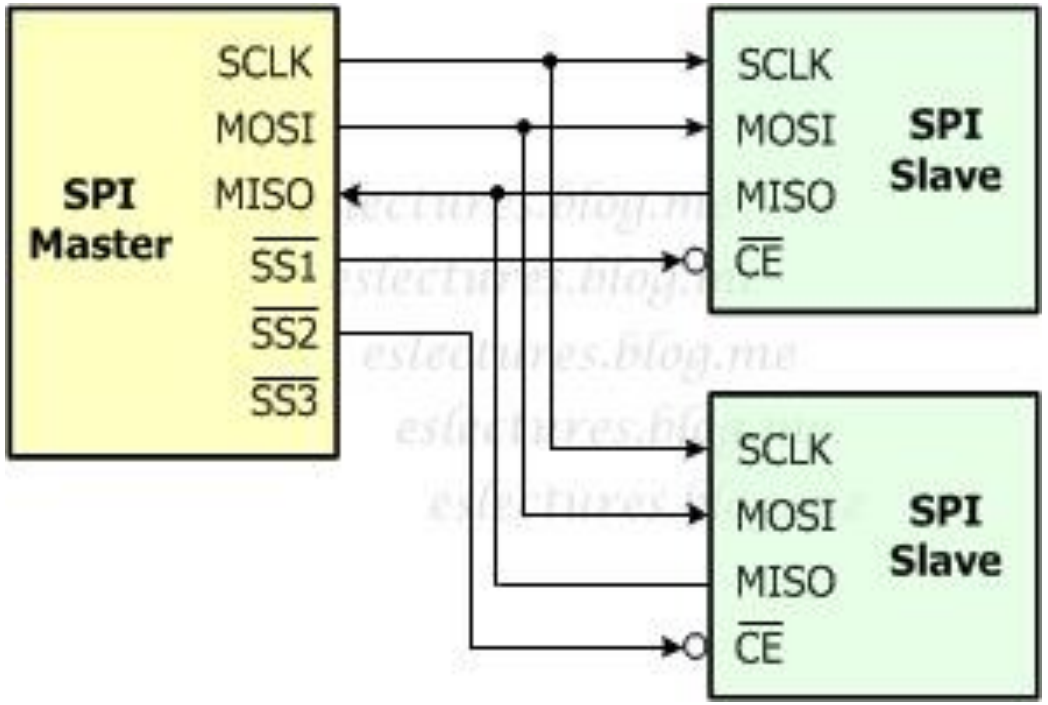
SPI

- SPI (Serial Peripheral Interconnect)
  - Motorola에 의해서 개발된 전이중(full duplex) 통신 규격
    - ➢ 전이중(full duplex) : 송신/수신이 동시에 이루어짐 ⇒ 송신용/수신용 선이 별도로 필요함
    - ➢ 반이중(half duplex) : 송신/수신을 동시에 처리하지 못하고 번갈아가면서 처리 (I2C)
  - I2C에 비해서 고속 통신 가능 (최대 70MHz)
  - 별도의 장치 연결용 선을 사용하므로 주소 충돌이 일어나지 않음
  - I2C에 비해서 연결하는 선이 많음 (이유: 전이중, 장치별로 선택신호선 연결)



SPI

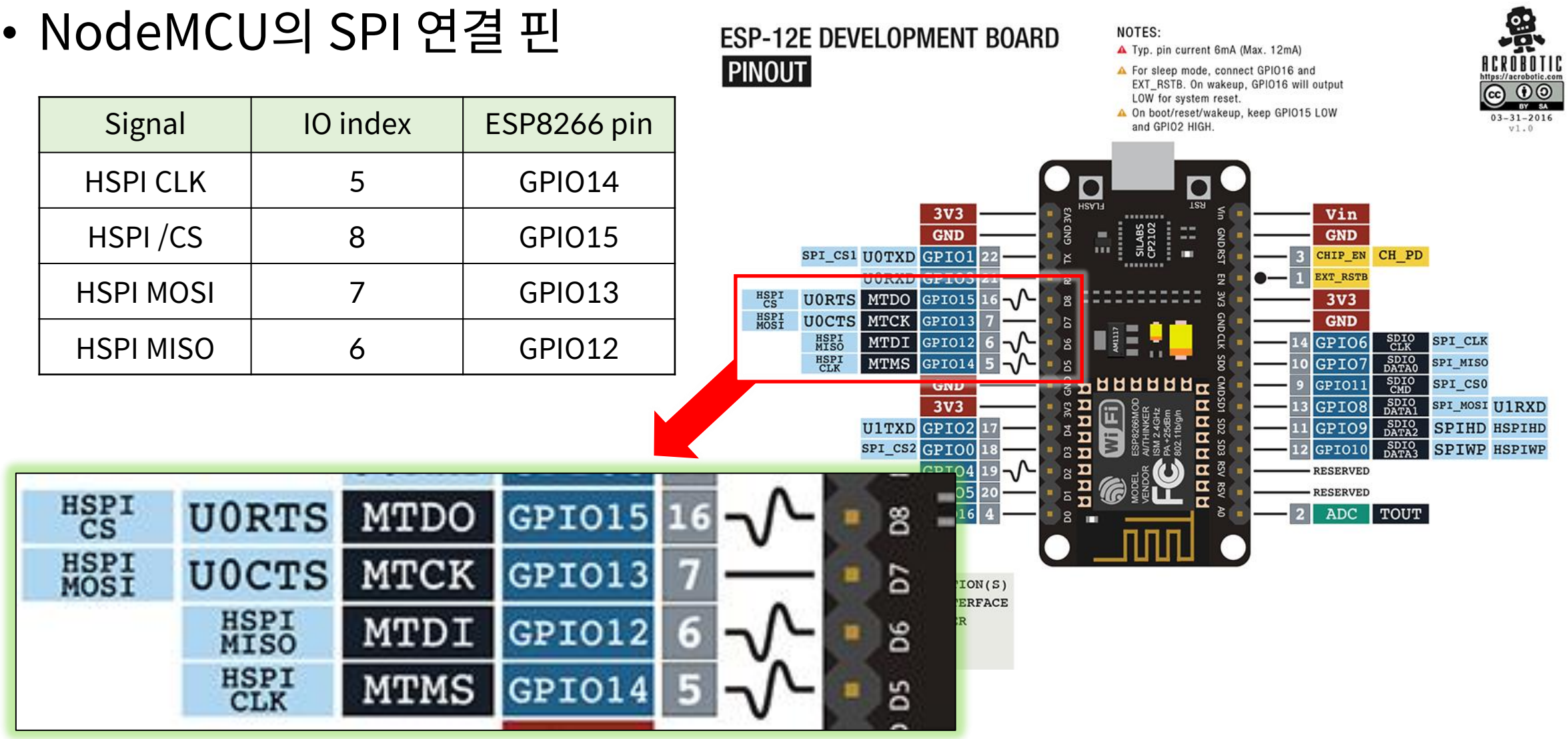
- SPI (Serial Peripheral Interconnect)
  - SCLK(Serial Clock) : Master기기에서 생성하는 동기화용 신호
  - MOSI(Master Output Slave Input) : Master → Slave
  - MISO(Master Input Slave Output) : Slave → Master
  - SS(Select Slave) : Slave 기기를 선택하는 신호



SPI

- NodeMCU의 SPI 연결 핀

Signal	IO index	ESP8266 pin
HSPI CLK	5	GPIO14
HSPI /CS	8	GPIO15
HSPI MOSI	7	GPIO13
HSPI MISO	6	GPIO12



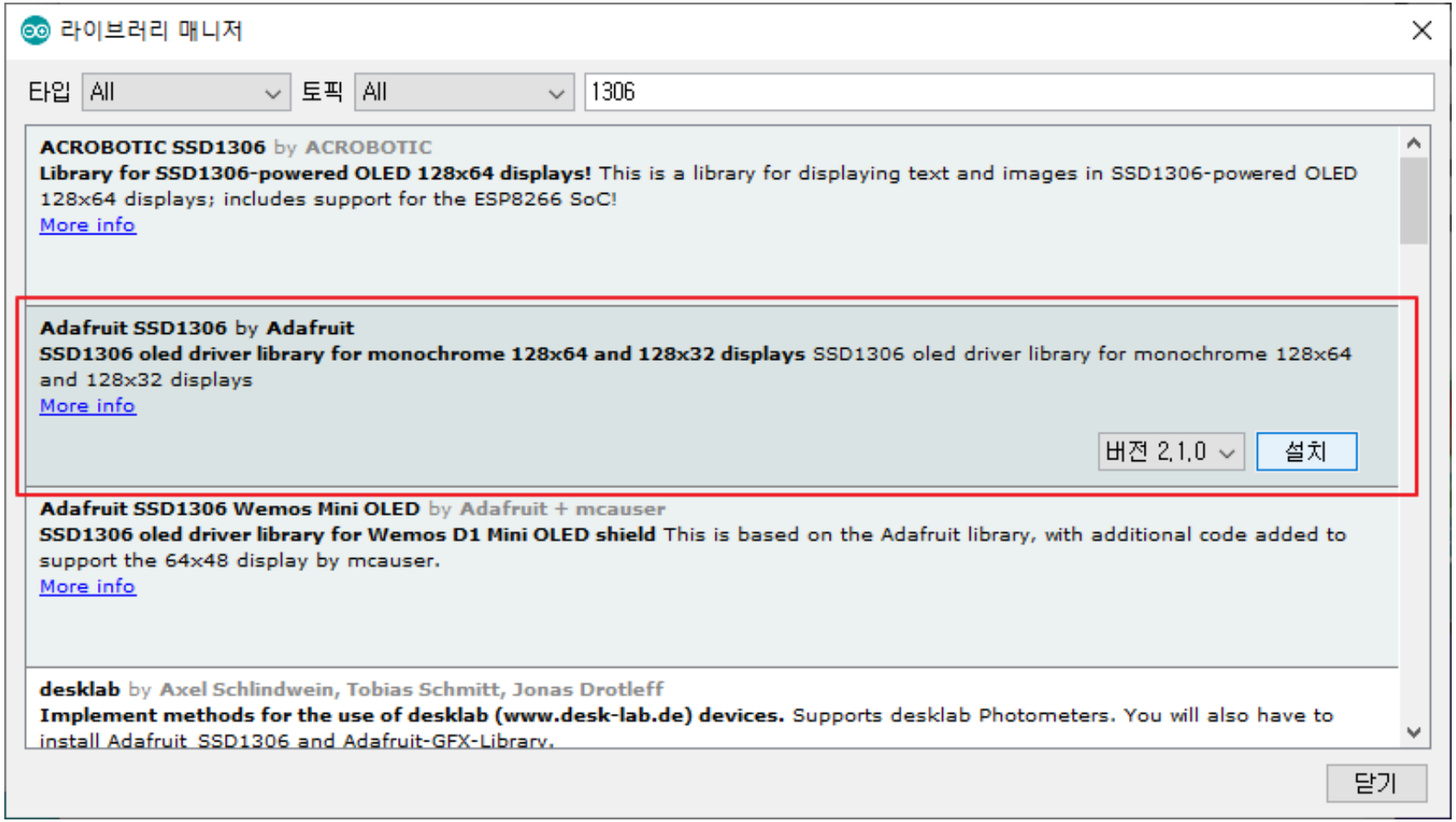
5.2 I2C OLED 디스플레이

- OLED (Organic Light-Emitting Diode)
  - 유기발광다이오드 디스플레이 장치

- 0.96” OLED : 많이 사용됨
  - 작은 크기의 128x~~96~~64 해상도
  - I2C 및 SPI Interface

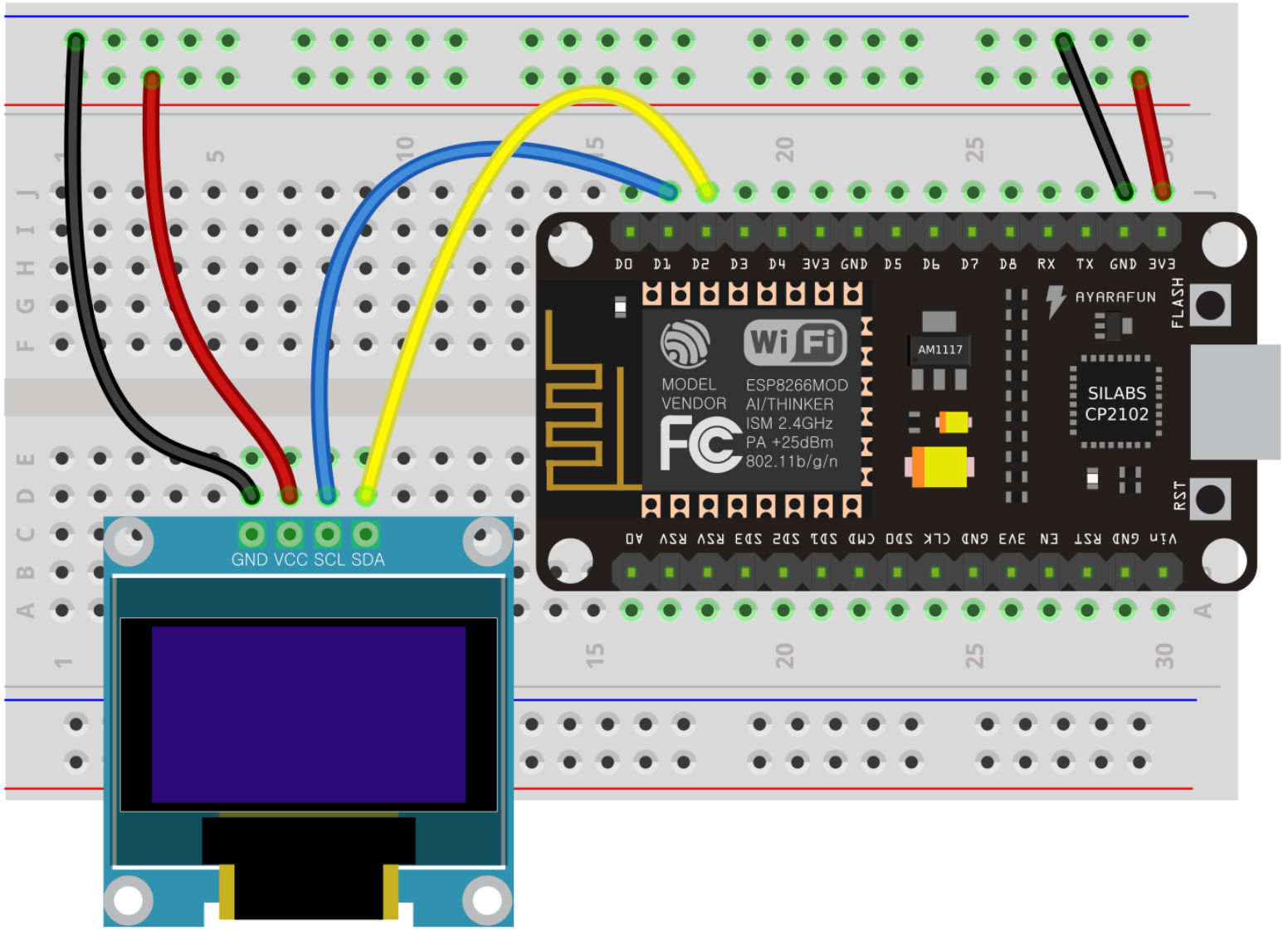
OLED 디스플레이 사용

- 라이브러리 추가
  - Adafruit SSD1306 라이브러리 추가



예제 5-1: OLED Display

- 회로 구성



OLED Display

- 아두이노 예제 : ssd1306_128x64_i2c

예제의 display 객체를 정의하는 부분

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
#define OLED_RESET     -1 // Reset pin # (NodeMCU에서는 -1로 수정해야 함)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
```

OLED Display

- 아두이노 예제 : ssd1306_128x64_i2c

예제의 display 객체 초기화하는 부분

```
void setup() {
  Serial.begin(9600);

  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }
```

I2C 주소 지정

- OLED에 표기된 I2C 주소 : 0x78
  - ➢ 0x78 (0111 1000) : 7bit 주소 + R/W
- display.begin()의 주소 : 0x3C
  - ➢ 0x78 (0111 1000) → R/W 비트 제거 (>> 1) ⇒ 0011 1100 : 0x3C

기기(datasheet)에 표기된 주소와 다른 경우가 있음

Adafruit_SSD1306 라이브러리 함수

Adafruit_SSD1306 멤버함수	설명
begin(switchvcc, i2caddr, reset, peribegin)	장치 시작 함수, switchvcc : 전원설정 SSD1306_SWITCHCAPVCC (기본값), i2caddr : I2C 장치 주소 (주로 0x3C), 나머지 기본값 사용
display()	버퍼의 내용을 화면에 반영. 그리기 함수 등은 버퍼에 그리기 때문에 그리기 후에 반드시 호출해서 화면에 반영해야 함
`clearDisplay()`	화면 지우기
`drawPixel(x, y, color)`	점찍기
`drawLine(x1, y1, x2, y2, color)`	선 그리기
`drawRect(x1, y1, x2, y2, color)`	사각형 그리기
`fillRect(x1, y1, x2, y2, color)`	내부가 채워진 사각형 그리기
`drawCircle(x, y, r, color)`	원점이 (x, y) 반지름이 r인 원 그리기
`fillCircle(x, y, r, color)`	내부가 채워진 원 그리기
`drawRoundRect(x1, y1, x2, y2, r, color)`	모서리가 r만큼 곡선을 가지는 사각형 그리기
`fillRoundRect(x1, y1, x2, y2, r, color)`	내부가 채워진 모서리가 부드러운 사각형 그리기

Adafruit_SSD1306 라이브러리 함수

Adafruit_SSD1306 멤버함수	설명
drawTriangle(x1, y1, x2, y2, x3, y3, color)	세 좌표를 연결하는 삼각형 그리기
fillTriangle(x1, y1, x2, y2, x3, y3, color)	내부가 채워진 삼각형 그리기
setTextSize(n)	글꼴의 크기를 설정하기, 1이 기본값
setTextColor(color, bgcolor)	글자 색 설정하기. bgcolor는 배경색으로 생략이 가능하다.
setCursor(x, y)	글자를 쓸 위치를 지정하기
cp437(true : 기본값 또는 false)	256개의 글자 사용하기, 표준 ASCII에서는 128개의 글자만 사용하며, 한글과 같은 비 영어권 문자에서는 ASCII 코드의 128 이후의 코드영역을 이용하는 경우가 많다. ASCII 코드 128 이후의 특수문자를 표시하려면 이 함수를 호출한다.
drawBitmap(x, y, bitmap, w, h, color)	비트맵 이미지를 표시한다. 비트맵 이미지는 폭이 w이고 높이가 h인 이미지 데이터만 배열 등으로 지정한다.

- color : SSD1306_WHITE, SSD1306_BLACK, SSD1306_INVERSE (원래의 색상 반전)

예제 5-1. OLED 디스플레이 사용

예제 5-1. OLED 디스플레이 사용

```
#include <Adafruit_SSD1306.h>
#include <splash.h>

#define SCREEN_WIDTH 128 // OLED 디스플레이 폭 (픽셀)
#define SCREEN_HEIGHT 64 // OLED 디스플레이 높이 (픽셀)

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT); // OLED 관리 클래스

// 처음에 한번 수행하는 함수
void setup() {
  // 센서의 동작을 확인하기 위한 용도로 시리얼 통신을 사용
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("SSD1306 초기화 실패");
    for(;;); // 무한 루프, loop() 함수로 진입하지 못하도록 하기 위함
  }
  // Adafruit의 로고가 디스플레이 버퍼에 들어있음
  display.display(); // 현재 버퍼를 화면에 표시
}
void loop() {} // 아무런 일도 하지 않으며, 최초의 화면을 유지
```

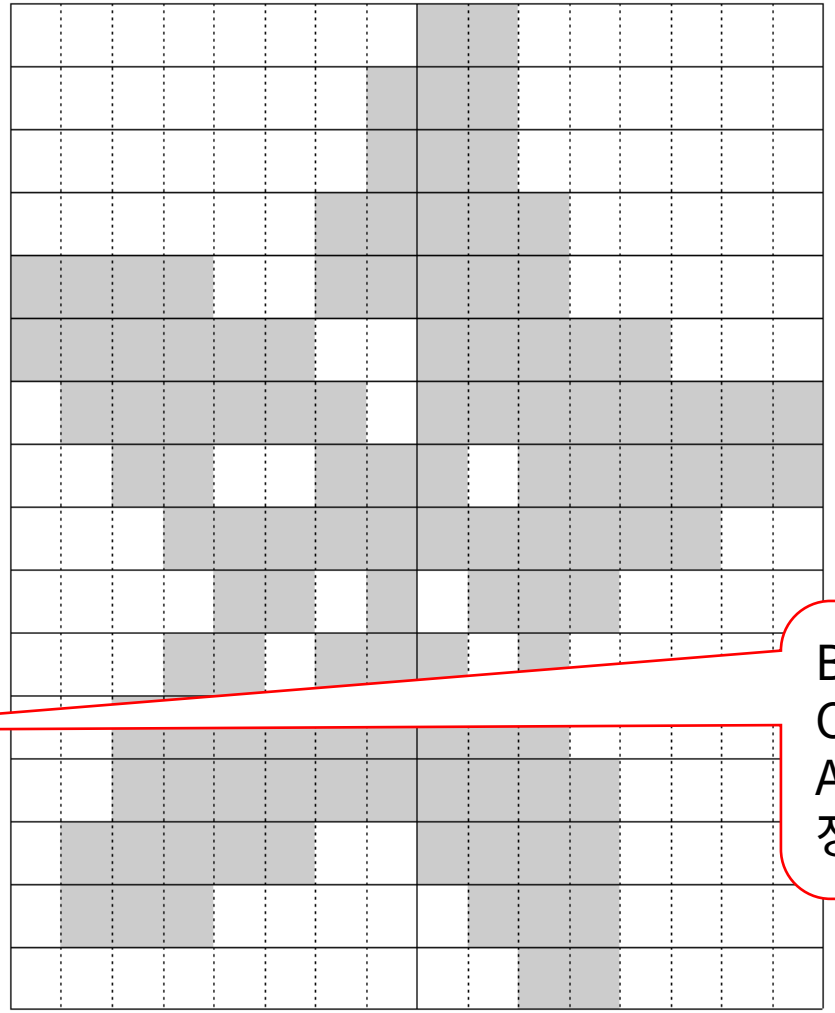
실습

- OLED 라이브러리 함수들 참고하여 다음의 기능을 수행하는 스케치를 완성하라.
  - 화면 바깥쪽 경계에 사각형 테두리를 그린다
  - 자신의 학번, 이름 등을 표시한다. (폰트 크기 등 상관없으며 영문으로 표시)

OLED Bitmap 출력

- drawBitmap() 함수 사용 : 이미지 데이터의 정의

```
static const unsigned char logo_bmp[] =
{ B00000000, B11000000,
  B00000001, B11000000,
  B00000001, B11000000,
  B00000011, B11100000,
  B11110011, B11100000,
  B11111110, B11111000,
  B01111110, B11111111,
  B00110011, B10011111,
  B00011111, B11111100,
  B00001101, B01110000,
  B00011011, B10100000,
  B00111111, B11100000,
  B00111111, B11110000,
  B01111100, B11110000,
  B01110000, B01110000,
  B00000000, B00110000 };
```



B로 시작하는 값은 2진수로 C/C++ 표준은 아님. Arduino의 헤더파일에 정의되어 있음

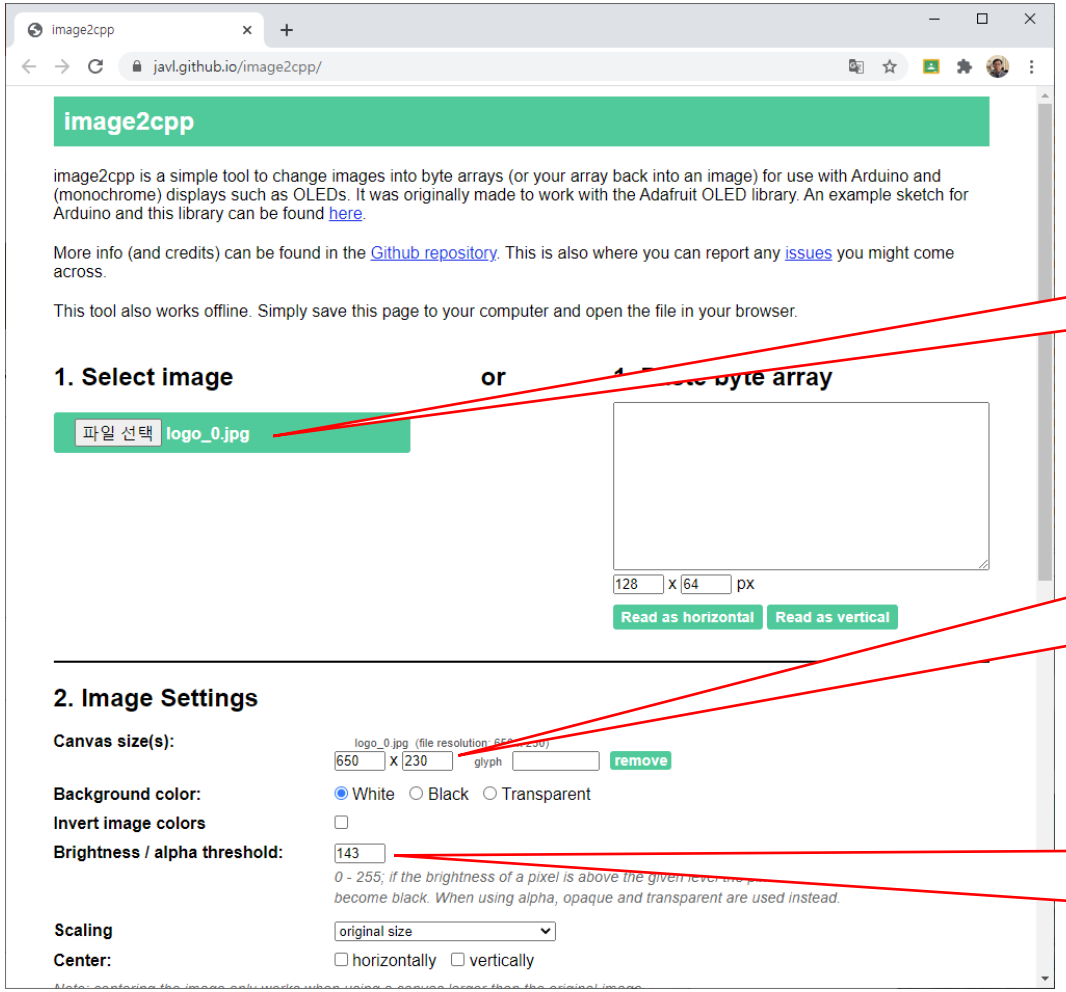
OLED Bitmap 출력

- drawBitmap() 함수 사용

```
// testanimate()함수의 비트맵 그리는 부분
// Draw each snowflake:
for(f=0; f< NUMFLAKES; f++) {
  display.drawBitmap(icons[f][XPOS], icons[f][YPOS], bitmap, w, h,
                     SSD1306_WHITE);
}
```

Image2cpp conversion

- 이미지를 2진수의 데이터 파일로 변환하여 주는 온라인 사이트
  - https://javl.github.io/image2cpp/



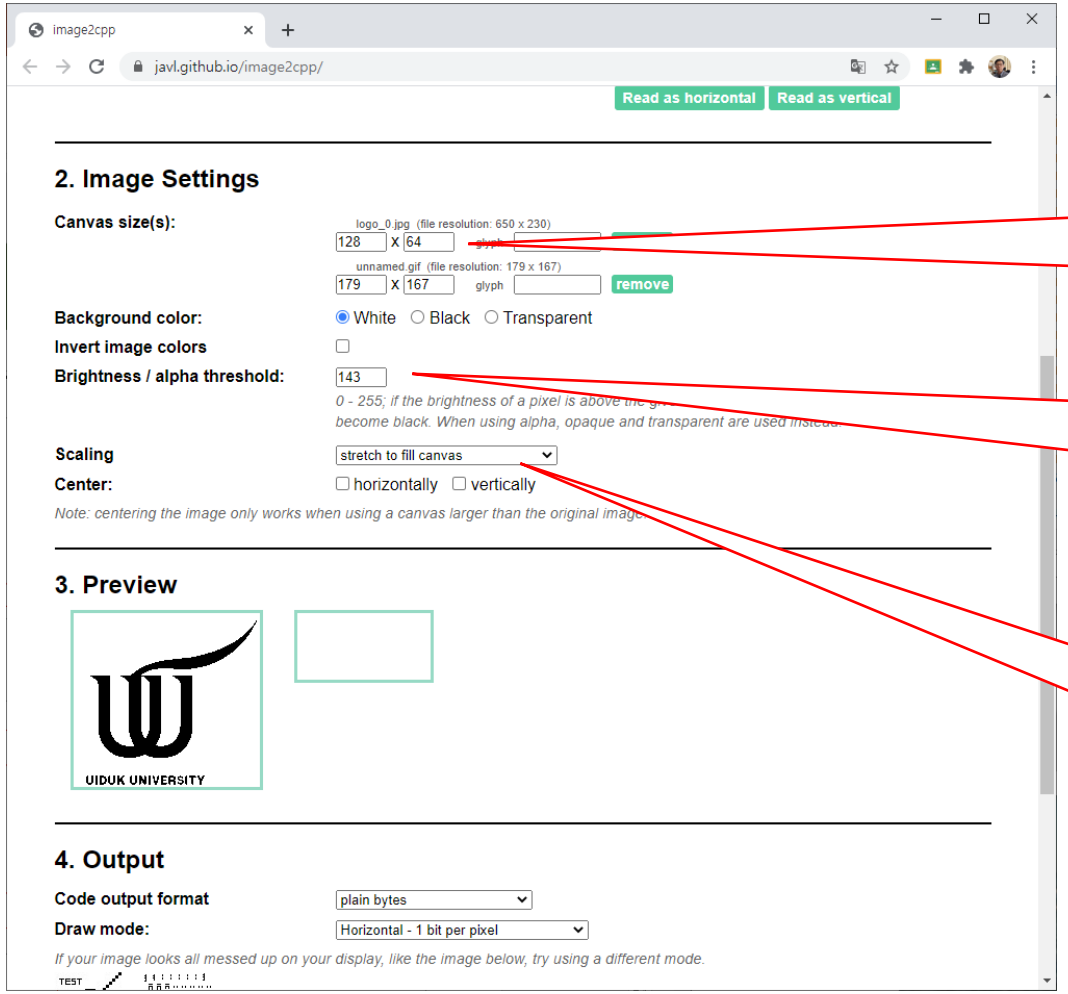
1. 이미지 파일을 불러와서 업로드한다.

2. Canvas size에서 이미지를 적절한 크기로 지정한다. (OLED의 크기를 고려해야 함)

3. 아래쪽의 Preview를 참고하여 흑/백으로 이진화할 문턱값을 설정한다.

Image2cpp conversion

- 이미지를 2진수의 데이터 파일로 변환하여 주는 온라인 사이트
  - https://javl.github.io/image2cpp/



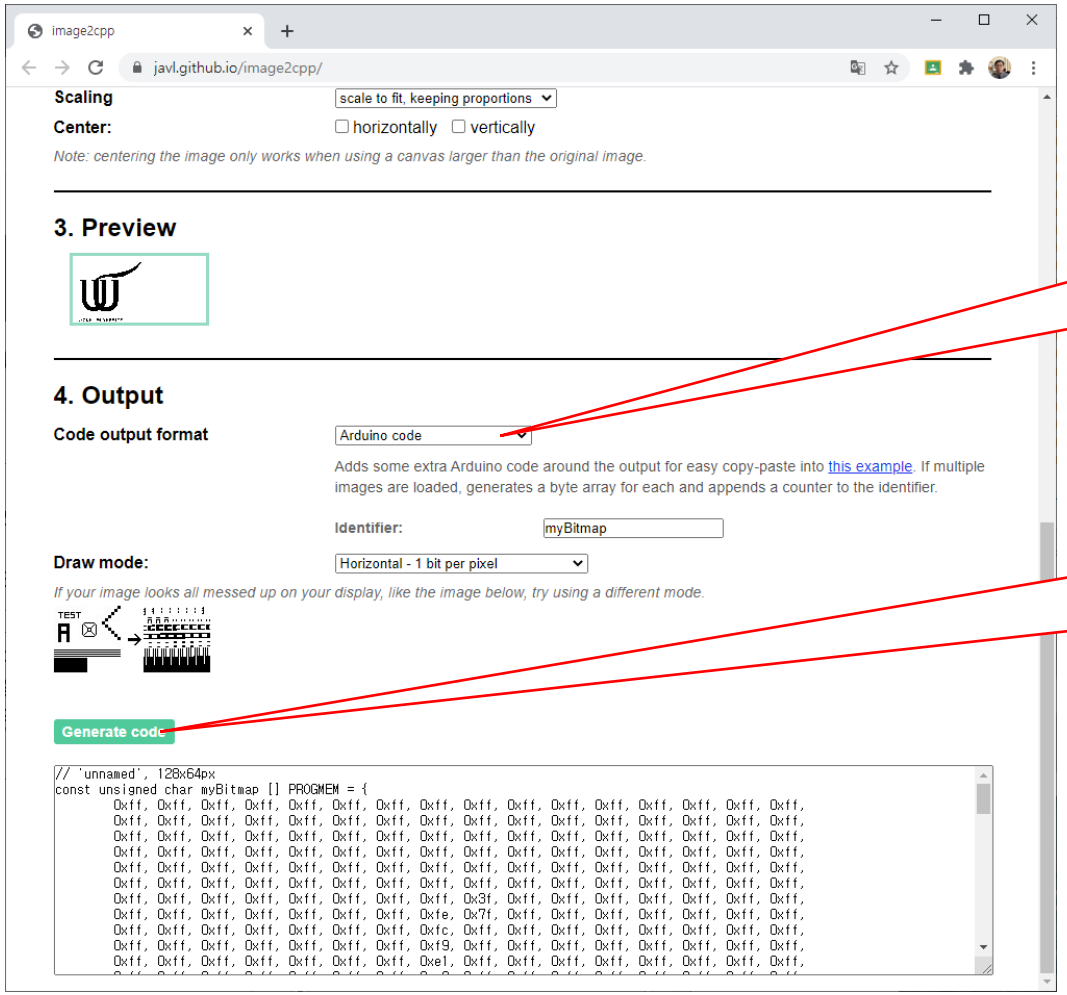
2. Canvas size에서 이미지를 적절한 크기로 지정한다. (OLED의 크기를 고려해야 함)

3. 아래쪽의 Preview를 참고하여 흑/백으로 이진화할 문턱값을 설정한다.

4. Scaling 모드도 같이 조절한다.

Image2cpp conversion

- 이미지를 2진수의 데이터 파일로 변환하여 주는 온라인 사이트
  - https://javl.github.io/image2cpp/



5. Code output format을 Arduino code로 설정하고, 아래에 변수명을 지정한다.

6. Generate code를 눌러서 이미지의 데이터를 생성한다음 복사해서 사용한다.

예제 5-2. OLED에 비트맵 이미지 표현하기

예제 5-2. OLED에 비트맵 표현하기

```
#include <Adafruit_SSD1306.h>
#include <splash.h>

#define SCREEN_WIDTH 128 // OLED 디스플레이 폭 (픽셀)
#define SCREEN_HEIGHT 64 // OLED 디스플레이 높이 (픽셀)

// OLED를 관리하는 클래스
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT);

static const unsigned char logo[] = {
// 'uulogo', 128x64px
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x
ff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x
ff, 0xff, 0xff,
// 중간 부분 생략
};
```

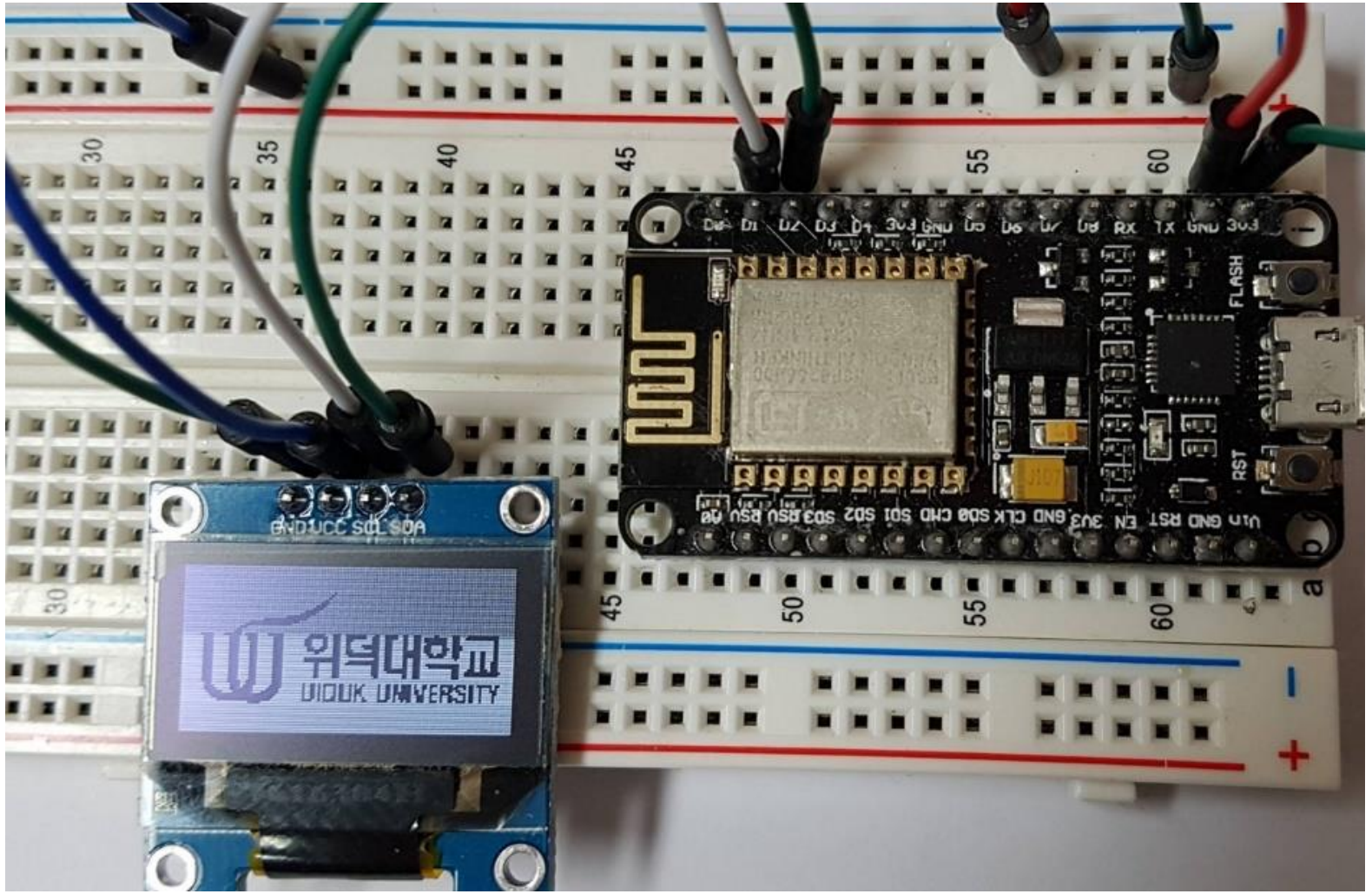
예제 5-2. OLED에 비트맵 이미지 표현하기

예제 5-2. OLED에 비트맵 표현하기

```
// 처음에 한번 수행하는 함수
void setup() {
  // 센서의 동작을 확인하기 위한 용도로 시리얼 통신을 사용
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("SSD1306 초기화 실패");
    for(;;); // 무한루프, loop() 함수로 진입하지 못하도록 하기 위함
  }
  // Adafruit의 로고가 디스플레이 버퍼에 들어있음
  // 현재 버퍼를 화면에 표시
  display.clearDisplay();
  display.drawBitmap(0, 0, logo, 128, 64, WHITE);
  display.display();
}

void loop() {
  // 아무런 일도 하지 않으며, 최초의 화면을 유지

}
```

예제 5-2. OLED에 비트맵 이미지 표현하기

- 실행 결과



실습

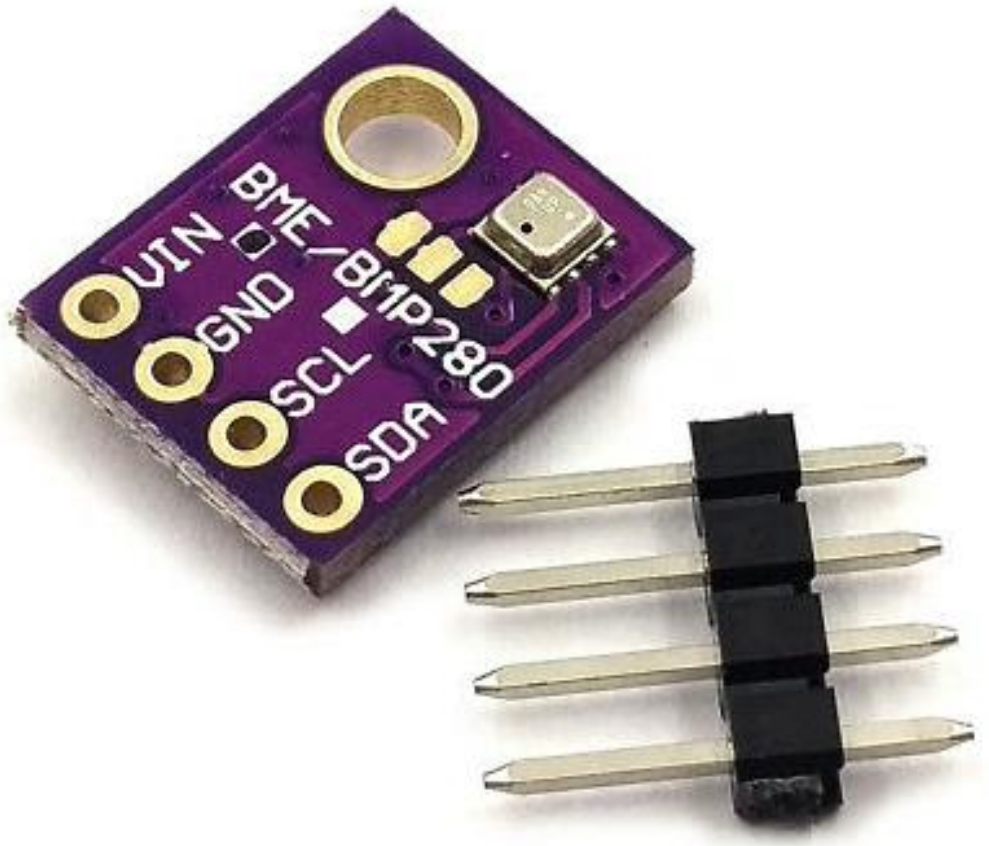
- 작은 이미지를 화면에 임의의 지점에 나타내는 스케치를 완성하시오.
  - 이미지는 임의의 이미지로 하며 8x8 ~ 16x16 정도의 이미지를 사용한다.
  - loop() 함수를 사용하며, 1초마다 임의의 지점에 난수 발생 함수를 사용하여 이미지의 위치가 바뀌도록 한다. (단 화면을 먼저 지우고 새로 그린다.)
- 난수 (random number)
  - 무작위로 만들어지는 수
  - C언어에서는 rand() 함수 사용 : 0 ~ RAND_MAX 사이의 숫자
    - `int r = rand() % 256;` ⇒ 0~255까지 난수를 발생시킴
  - 아두이노에서는 random() 함수 사용
    - `int ra = random(200);` ⇒ 0~199 사이의 난수 발생
    - `int rb = random(10, 20);` ⇒ 10~19 사이의 난수 발생

실습

- 그래픽 라이브러리를 이용하여 임의의 내용을 그리는 스케치를 작성하라.
  - 그림의 테마는 자유롭게 정한다.
  - 시간이 지남에 따라 화면이 변화하는 기능을 넣는다. (animation)

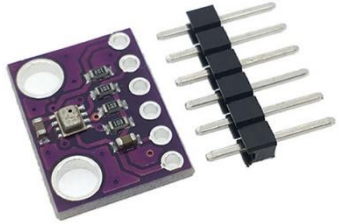
5.3 BME280 센서

- BME280 센서
  - 온도, 습도, 기압 측정용 센서
    - ➢ 습도 : 0~100% (오차 : ± 3% RH)
    - ➢ 온도 : -40 ~ 85°C (오차 : ± 1°C)
    - ➢ 감지 주기 : 평균 1초
    - ➢ 소비전력 : 측정시 360uA, 대기시 0.5uA
  - I2C 통신 방식을 사용
  - 소형의 크기이며, DHT 시리즈에 비해서 더욱 넓은 범위까지 측정
  - BMP280 : 온도, 기압 센서 (습도X)
  - BMP280 + 습도 ⇒ BME280
  - 사양

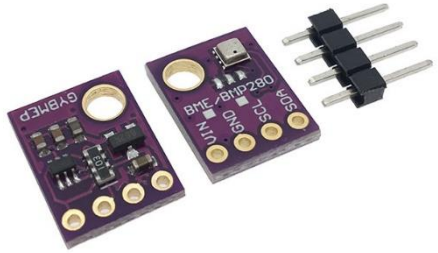


5.3 BMx280 센서 종류

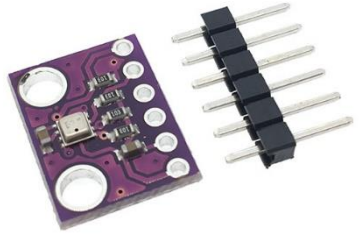
- BMP280, BME280, 3.3V / 5V



BMP280-3.3V



BME280-5V



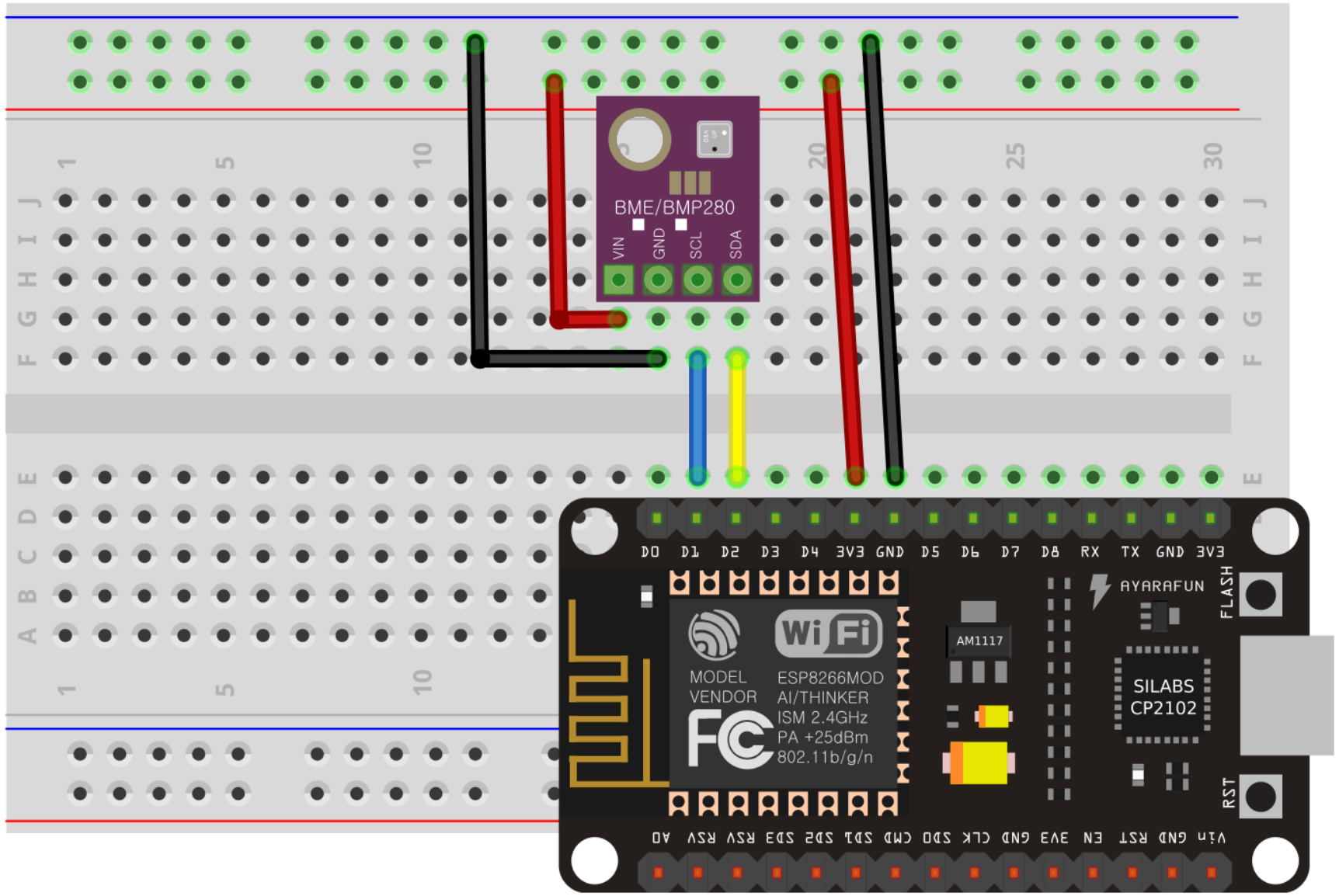
BME280-3.3V

BMP280-5V

그림 출처 : https://ko.aliexpress.com/item/32654011852.html

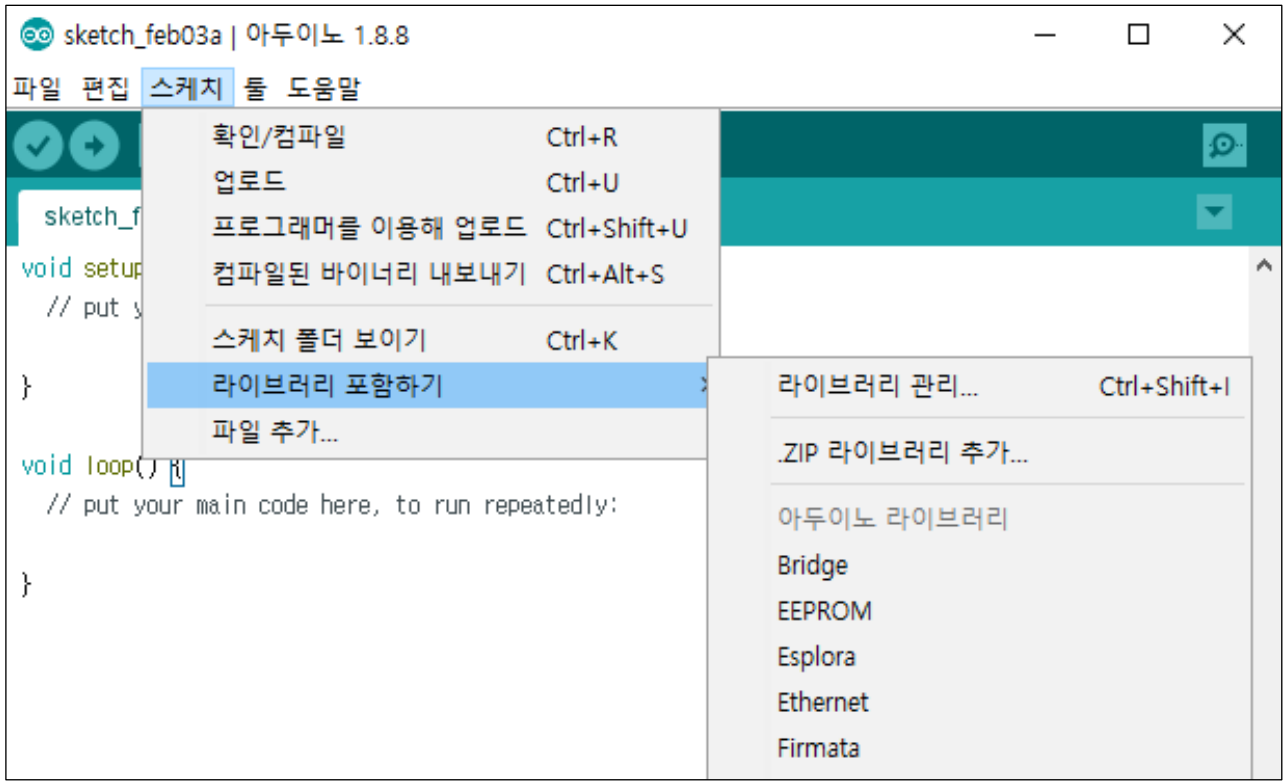
예제 5-3. BME280 센서

- 회로 구성
  - I2C 연결
  - SDA : D2
  - SCL : D1



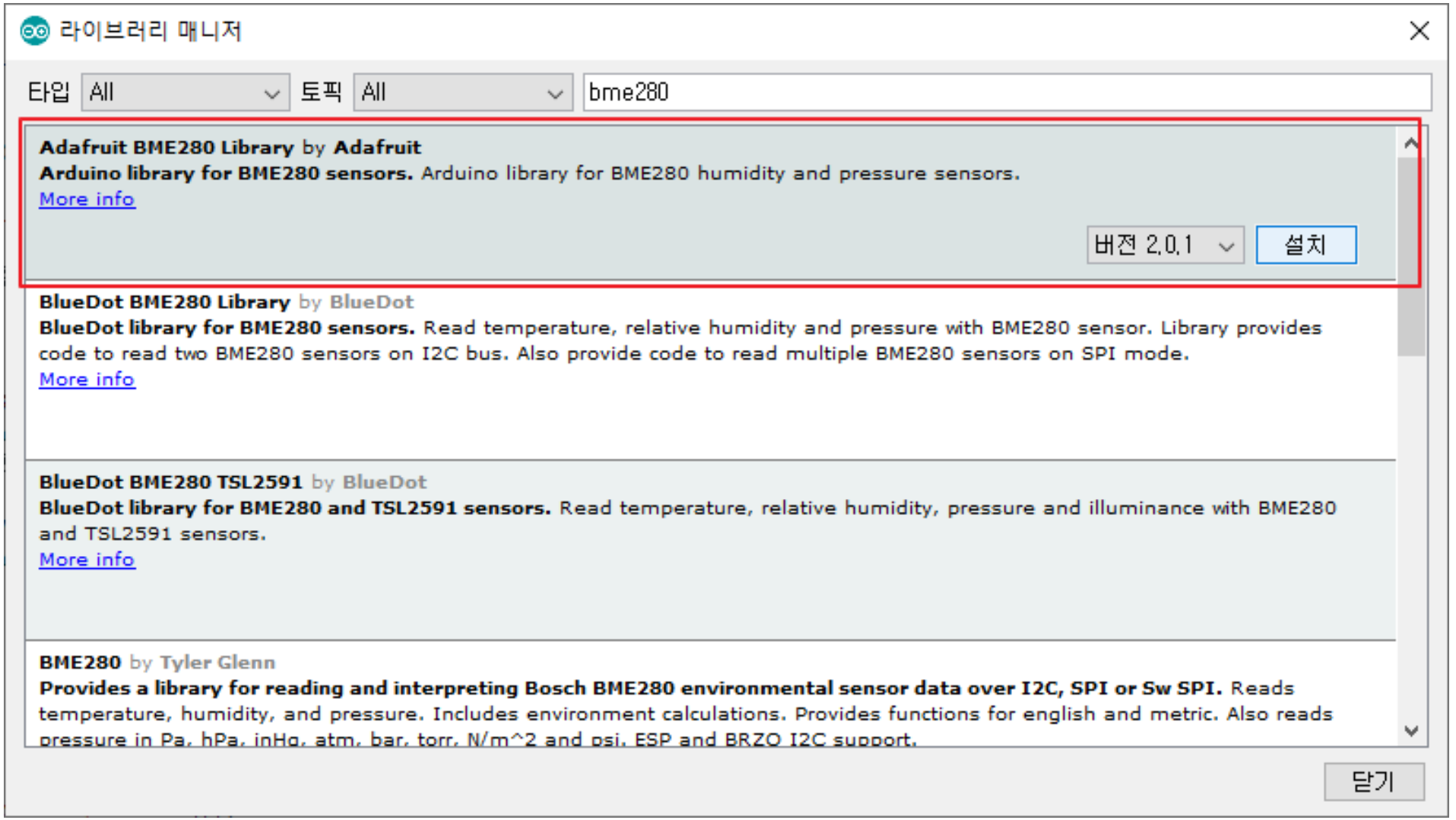
라이브러리 사용

- 라이브러리(library)
  - 자주 사용하는 함수들을 모듈화해서 만들어놓은 함수들의 모임
  - 초기에 내장된 라이브러리 (아두이노 라이브러리)
  - 라이브러리 관리 (Ctrl+Shift+I)
    - ➢ 아두이노에서 관리하는 추가 라이브러리
  - .ZIP 라이브러리
    - ➢ 인터넷에서 다운로드받은 zip 파일형태
    - ➢ 아두이노에서 관리하지 않음



예제 5-3. BME280 센서

- 라이브러리 추가 : bme280으로 검색 → Adafruit BME280 Library



예제 5-3. BME280 센서

예제 5-3. BME280 센서 사용

```
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

Adafruit_BME280 bme; // BME280 클래스 및 객체 정의

#define SEALEVELPRESSURE_HPA (1013.25) // 해수면 기압정보(hPa)

// 처음에 한번 수행하는 함수
void setup() {
  // 센서의 동작을 확인하기 위한 용도로 시리얼 통신을 사용
  Serial.begin(9600);
  // BME센서의 초기화, BME280 센서의 I2C 주소 = 0x76
  if(!bme.begin(0x76)) {
    Serial.println("Cannot initialize BME280");
    // 오류메시지 출력하고 무한 대기
    for(;;) ;
  }
}
```

예제 5-3. BME280 센서

예제 5-3. BME280 센서 사용

```
void loop() {
  // 온도 측정치 출력
  Serial.print("Temperature = ");
  Serial.print(bme.readTemperature());
  Serial.println(" *C");

  // 기압 측정치 출력
  Serial.print("Pressure = ");
  Serial.print(bme.readPressure() / 100.0f);
  Serial.println(" hPa");
```

예제 5-3. BME280 센서

예제 5-3. BME280 센서 사용

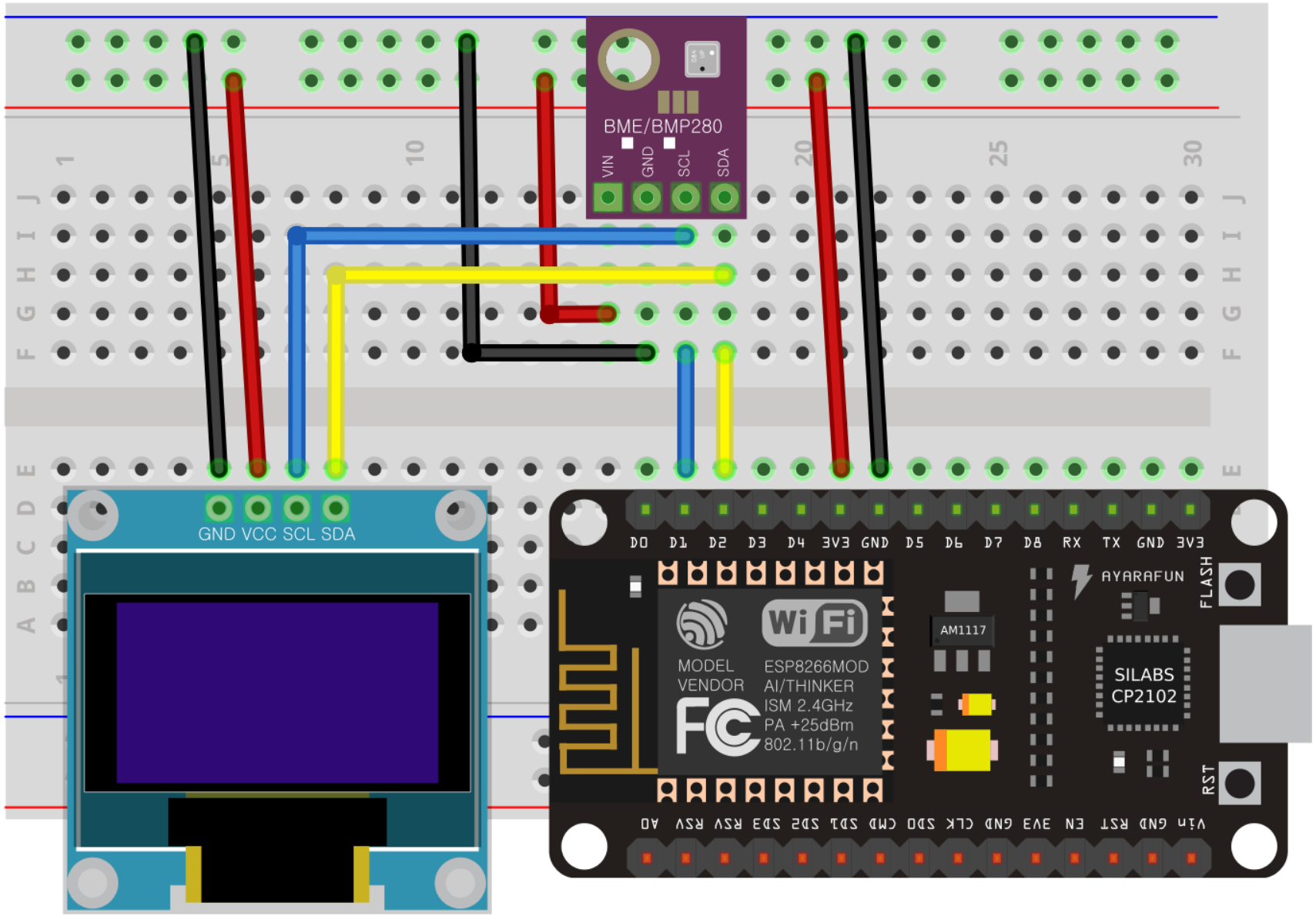
```
  // 주어진 해수면 기압을 바탕으로 고도를 추정하여 출력
  Serial.print("Approx. Altitude = ");
  Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
  Serial.println(" m");

  // 습도 측정치 출력
  Serial.print("Humidity = ");
  Serial.print(bme.readHumidity());
  Serial.println(" %");

  // 빈 줄 출력
  Serial.println();

  // 1초 대기
  delay(1000);
}
```

예제 5-3. BME280 센서 사용

- Adafruit_BME280 라이브러리 함수
  - `begin(주소)` : 센서를 초기화한다. 센서의 I2C 주소를 인수로 입력하며, BME280의 경우에 0x76을 주로 사용한다.
  - `readTemperature()` : 온도를 읽어들이는 함수
  - `readPressure()` : 기압을 읽어들인다. 파스칼 단위로 반환되며 보통 사용하는 헥토파스칼 단위로 변경하기 위해서 100으로 나눈 값을 출력하였다.
  - `readAltitude()` : 측정된 기압을 바탕으로 고도를 계산한다. 인수로 해수면의 기압을 정보로 준다.
  - `readHumidity()` : 습도를 계산한다.

예제 5-4. BME280 센서와 OLED 사용

- 회로 구성
  - 모두 I2C에 연결
    - OLED : I2C
    - BME : I2C
  - I2C 연결
    - SDA : D2
    - SCL : D1



예제 5-4. BME280 센서와 OLED 사용

예제 5-4. BME280 센서와 OLED 사용

```
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#include <Adafruit_SSD1306.h>
#include <splash.h>

// BMP280
Adafruit_BME280 bme;

#define SEALEVELPRESSURE_HPA (1013.25) // 해수면 기압정보(hPa)

// OLED
#define SCREEN_WIDTH 128 // OLED 디스플레이 폭 (픽셀)
#define SCREEN_HEIGHT 64 // OLED 디스플레이 넓이 (픽셀)

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT);
```

예제 5-4. BME280 센서와 OLED 사용

예제 5-4. BME280 센서와 OLED 사용

```
// 처음에 한번 수행하는 함수
void setup() {
  // 센서의 동작을 확인하기 위한 용도로 시리얼 통신을 사용
  Serial.begin(9600);
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println("Cannot initialize SSD1306");
    for(;;); // 무한루프, loop() 함수로 진입하지 못하도록 하기 위함
  }

  if(!bme.begin(0x76)) {
    Serial.println("Cannot initialize BME280");
    // 오류메시지 출력하고 무한 대기
    for(;;) ;
  }
}
```

예제 5-4. BME280 센서와 OLED 사용

예제 5-4. BME280 센서와 OLED 사용

```
void loop() {
  // 화면 지우고 폰트 설정하기
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);

  // OLED에 출력할 좌표 지정
  display.setCursor(0, 0);
  // 온도 측정치 출력
  display.print("Temp.  = ");
  display.print(bme.readTemperature());
  display.println(" *C");

  // 기압 측정치 출력
  display.setCursor(0, 14);
  display.print("Press. = ");
  display.print(bme.readPressure() / 100.0f);
  display.println(" hPa");
```

예제 5-4. BME280 센서와 OLED 사용

예제 5-4. BME280 센서와 OLED 사용

```
  // 주어진 해수면 기압을 바탕으로 고도를 추정하여 출력
  display.setCursor(0, 28);
  display.print("Altitude = ");
  display.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
  display.println(" m");

  // 습도 측정치 출력
  display.setCursor(0, 44);
  display.print("Humidity = ");
  display.print(bme.readHumidity());
  display.println(" %");

  // 화면에 표시하기
  display.display();
  // 1초 대기
  delay(1000);
}
```

추가 실습

- 다른 센서를 이용하여 측정 결과를 OLED에 표시하는 회로를 만들고 스케치를 작성하라.