

Chapter 22. 아두이노-아두이노 I2C 통신

1. I2C (Inter-Integrated Circuit)

I2C 통신은 UART, SPI와 더불어 아두이노에서 흔히 사용되는 직렬 통신 방법 중 하나이다. UART는 1960년대에 구현된 오래된 직렬 통신 방식으로 다양한 장치들이 지원하는 장점이 있지만 1:1 통신만이 가능한 단점이 있어 여러 개의 주변장치를 연결하기가 불편하다. 이러한 단점을 보완할 수 있는, 1:n 통신이 가능한 통신 방법 중 하나가 SPI 통신이다. SPI에서는 3개의 데이터 선을 모든 장치가 공유하고 n개의 장치 중 하나의 장치를 선택하기 위한 SS (Slave Select) 연결을 장치별로 추가하여 사용한다. SPI는 고속의 데이터 전송이 가능하지만 여러 개의 장치를 연결하는 경우 SS 연결이 증가하는 단점이 있다. I2C 역시 1:n 통신이 가능한 직렬 통신 방법이지만 SPI와는 다른 목적에서 만들어졌다. I2C가 SPI와 다른 점 중 하나는 물리적인 SS 연결이 필요하지 않으며 소프트웨어적인 주소로 특정 장치를 선택한다는 점이다. 이는 인터넷에서 서로 연결되어 있는 수많은 컴퓨터 중 내 컴퓨터를 구별하기 위해 IP 주소를 사용하고 있는 것과 유사하다. I2C에서는 SS 연결이 필요하지 않으므로 연결된 장치의 수가 늘어나더라도 필요로 하는 연결의 수는 증가하지 않는 장점이 있다. SPI 통신에서 SS 연결을 제외하면 모든 장치들이 공유하고 있는 3개의 연결선이 남는다. I2C에서는 여기서 하나의 연결선을 더 줄일 수 있다. 연결선을 줄일 수 있는 비밀은 I2C가 다른 직렬 통신 방법과 다르게 반이중(half-duplex) 방식의 통신이라는데 있다. UART와 SPI는 전이중(full-duplex) 방식의 통신으로 송신과 수신에 동시에 진행될 수 있다. 반면 반이중 방식에서는 1개의 연결선만을 사용하며 송신과 수신이 진행되므로 연결선의 수는 적은 반면 송신과 수신은 동시에 진행될 수 없다. 따라서 I2C에서는 데이터 송수신을 위해 1개, 동기화 클록을 위해 1개 총 2개의 연결선만을 필요로 한다. I2C에서 사용하는 2개의 연결선은 데이터 전송을 위한 SDA (Serial Data), 클록 전송을 위한 SCL(Serial Clock)이라고 불린다.

2. 아두이노에서의 I2C

데이터 전송이 이루어지지 않는 경우 I2C 연결선은 HIGH 상태를 유지하고 있으며 이를 위해 I2C에서는 풀업저항을 사용한다. 두 개의 슬레이브를 마스터에 연결한 예는 그림 1과 같다.

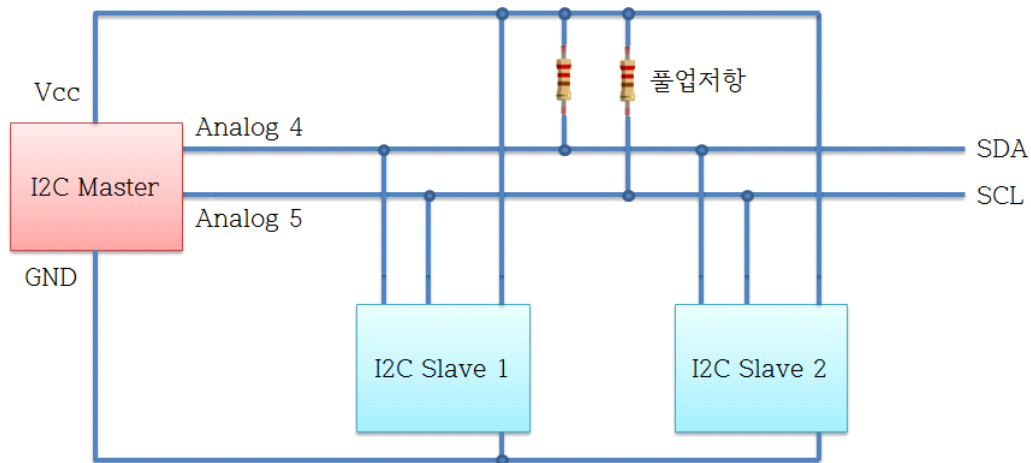


그림 1. I2C 연결 예

그림 1은 아두이노 우노의 예로 SDA를 위해 아날로그 4번 핀을, SCL을 위해 아날로그 5번 핀을 사용한다. 아두이노 메가2560은 SDA를 위해 디지털 20번 핀을, SCL을 위해 디지털 21번 핀을 사용한다. 이처럼 아두이노에 사용된 마이크로컨트롤러는 I2C 통신을 위한 전용 핀을 제공하고 있다. 또한 아두이노에서는 I2C 통신을 지원하기 위해 기본 라이브러리 중 하나로 Wire 라이브러리를 제공하고 있다.

- begin

```
void begin(void)
void begin(uint8_t address)
  - 매개변수
    address : I2C 주소
  - 반환값 : 없음
```

Wire 라이브러리를 초기화하고 I2C 버스에 마스터나 슬레이브로 참여한다. 주소가 주어지는 경우에는 슬레이브로, 주소가 주어지지 않은 경우에는 마스터로 참여함을 의미한다.

- requestFrom

```
uint8_t requestFrom(uint8_t address, uint8_t quantity)
uint8_t requestFrom(uint8_t address, uint8_t quantity, uint8_t sendStop)
- 매개변수
  address : 슬레이브 주소
  quantity : 요청하는 바이트 수
  sendStop : 요청 완료 후 정지 메시지 전송 여부 (true 또는 false)
- 반환값 : 슬레이브 장치로부터 전송된 바이트 수
```

마스터 장치가 특정 주소(address)의 슬레이브 장치에 지정한 양(quantity)의 데이터를 요청한다. 매개변수 sendStop에는 true나 false의 값을 지정하며 요청이 완료된 후 정지 메시지를 보낼지의 여부를 나타낸다. false를 지정하면 I2C 마스터는 요청이 완료된 후에도 연결을 유지한다. 디폴트는 true이다.

- beginTransmission

```
void beginTransmission(uint8_t address)
- 매개변수
  address : 슬레이브 주소
- 반환값 : 없음
```

지정한 주소의 슬레이브 장치로 데이터 전송을 시작한다. 실제 전송은 write 함수에 의해 버퍼에 기록된 후 endTransmission 함수가 호출될 때 일어난다.

- endTransmission

```
uint8_t endTransmission(void)
uint8_t endTransmission(uint8_t sendStop)
- 매개변수
  sendStop : 요청 완료 후 정지 메시지 전송 여부 (true 또는 false)
- 반환값 : 전송 상태 메시지
```

write 함수에 의해 큐에 기록된 데이터를 전송함으로써 beginTransmission 함수에 의해 시작된 슬레이브 장치에 대한 전송을 종료한다. 반환값은 전송 결과를 나타내는 값으로 0이 아닌 값은 전송 과정에서 오류가 발생하였음을 나타낸다.

- write

```
size_t write(uint8_t data)
size_t write(const uint8_t *data, size_t quantity)
- 매개변수
  data : 전송할 단일 바이트 또는 바이트 배열에 대한 포인터
  quantity : 전송할 바이트 수
- 반환값 : 전송된 바이트 수
```

마스터 장치의 요청에 따라 슬레이브 장치가 데이터를 전송하거나 마스터 장치에서 슬레이브 장치로의 데이터 전송을 위해 큐에 데이터를 기록하기 위해 사용한다. 마스터 장치에서 슬레이브 장치로의 데이터 전송은 beginTransmission 함수로 시작되고 write 함수에서 큐에 데이터를 기록한 후 endTransmission 함수 호출로 실제 전송이 일어난다.

- available

```
int available(void)
- 매개변수 : 없음
- 반환값 : 유효 바이트 수
```

read 함수로 읽어 들일 수 있는 유효한 바이트 수를 반환한다. available 함수는 마스터 장치에서 requestFrom 함수를 통해 슬레이브 장치로부터 데이터를 요청한 이후 실제 도착한 데이터를 검사하기 위해 사용하거나 슬레이브 장치의 onReceive 함수 내에서 사용된다.

- read

```
int read(void)
- 매개변수 : 없음
- 반환값 : 수신 버퍼의 한 바이트를 읽어서 반환
```

마스터 장치의 requestFrom 함수 호출에 의해 슬레이브 장치가 전송한 데이터 한 바이트를 읽어서 반환한다. 마스터 장치가 전송한 데이터를 슬레이브 장치에서 읽어 들이기 위해서도 사용할 수 있다.

- onReceive

```
void onReceive( void (*)(int) )
```

- 매개변수 : 없음
- 반환값 : 없음

슬레이브 장치가 마스터 장치로부터 데이터를 수신했을 때 호출되는 핸들러(handler) 함수를 등록한다. 핸들러 함수는 반환값이 없으며 수신한 데이터의 바이트 수를 나타내는 int 형식의 매개변수를 가진다.

- onRequest

```
void onRequest( void (*)(void) )
```

- 매개변수 : 없음
- 반환값 : 없음

슬레이브 장치가 마스터 장치로부터 데이터 요청을 받았을 때 호출되는 핸들러(handler) 함수를 등록한다. 핸들러 함수는 반환값이 없으며 매개변수도 갖지 않는다.

3. 아두이노-아두이노 I2C 통신

아두이노 메가2560을 마스터로, 아두이노 우노를 슬레이브로 하여 데이터를 송수신하는 스케치를 작성해 보자. 학습보드에서 아두이노 우노와 메가2560은 I2C 통신이 가능하도록 연결되어 있다.

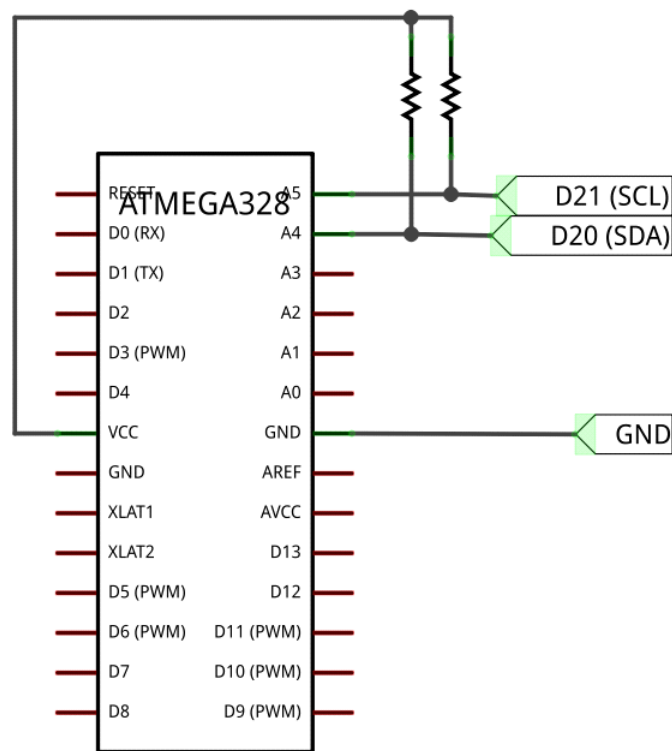


그림 2. 아두이노 우노와 메가2560의 I2C 연결 회로도

스케치 1은 아두이노 우노에 업로드 할 스케치로 마스터의 요청이 있을 때 카운터 값을 1 증가시켜 마스터로 전송하는 스케치의 예이다.

스케치 1. I2C 통신 테스트 - 슬레이브(우노)

```
#include <Wire.h>

#define SLAVE 4

byte count = 0; // 마스터로 송신할 카운터 값

void setup() {
  // Wire 라이브러리 초기화. 슬레이브로 참여하기 위해서는 주소를 지정해야 한다.
  Wire.begin(SLAVE);
  // 마스터의 데이터 전송 요구가 있을 때 처리할 함수 등록
  Wire.onRequest(sendToMaster);
}
```

```
Serial.begin(9600);
}

void loop () {

void sendToMaster() {
    // 카운터 값을 증가시키고 마스터로 전송
    Wire.write(++count);

    Serial.print("Current count is ");
    Serial.println(count);
}
```

스케치 2는 아두이노 메가2560에 업로드 할 스케치로, 1초에 한 번 슬레이브로 카운터 값을 요청하고 그 결과를 받아 시리얼 모니터로 출력하는 스케치의 예이다.

스케치 2. I2C 통신 테스트 - 마스터(메가2560)

```
#include <Wire.h>

#define SLAVE 4 // 슬레이브 주소

void setup() {
    // I2C 통신을 위한 Wire 라이브러리 초기화. 마스터로 참여하는 경우 주소 불필요
    Wire.begin();
    Serial.begin(9600);
}

void loop() {
    i2c_communication();
    delay(1000); // 1초 대기
}
```

```
// 슬레이브로 데이터 요구 및 수신 데이터 처리
void i2c_communication() {
    Wire.requestFrom(SLAVE, 1);           // 1 바이트 크기의 데이터 요구
    char c = Wire.read();                 // 수신 데이터 읽기

    Serial.print("I got ");
    Serial.println(String(c, DEC));       // 수신 데이터 출력
}
```

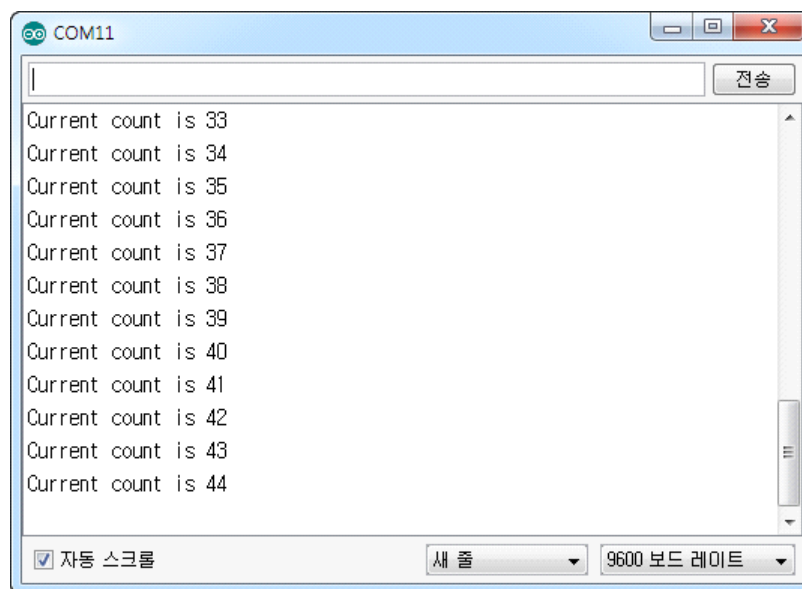


그림 3. 스케치 1 실행 결과 (슬레이브, 우노)

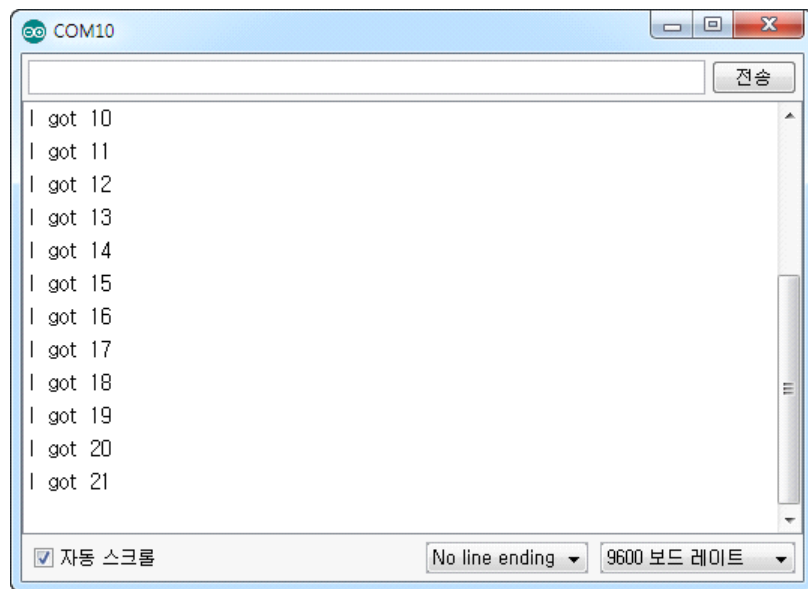


그림 4. 스케치 2 실행 결과 (마스터, 메가2560)

스케치 1과 스케치 2에서 아두이노 우노는 슬레이브로 동작하면서 4번의 주소가 할당되어 있다. RTC 장에서 이미 I2C 통신을 사용하는 DS1307을 사용하였지만 주소를 지정하지는 않았다. 이는 DS1307 칩을 사용하기 위해 RTCLib 라이브러리를 사용하였으며 RTCLib 라이브러리에는 DS1307을 위한 I2C 주소가 지정되어 있기 때문이다. 반면 아두이노의 경우에는 I2C의 7비트 주소 중 자유롭게 선택하여 사용할 수 있다. 따라서 여러 개의 I2C 통신을 사용하는 주변장치를 연결하는 경우에는 데이터시트를 확인하여 동일한 주소를 사용하는 지 여부를 확인하여야 한다.