
Chapter 220. 아두이노-아두이노 UART 연결

UART(Universal Asynchronous Receiver/Transmitter)는 마이크로컨트롤러에서 사용하는 대표적인 시리얼 통신 방법 중 하나로, 오랜 역사로 많은 장치들이 UART 통신을 지원하고 있고 연결이 간단하여 아직도 많이 사용되고 있다. UART가 1:1의 연결만이 가능하고 TTL 레벨을 사용하므로 전송 거리가 짧은 등의 단점이 있지만 RS-232, RS-422 등과 함께 사용하면 이러한 단점을 극복할 수 있다. UART 장에서 시리얼 텍스트 LCD를 아두이노와 UART 시리얼 통신으로 연결하여 사용하는 방법에 대해 살펴보았으며 이 장에서는 두 개의 아두이노 우노가 UART 통신으로 데이터를 주고받는 방법에 대해 알아본다.

1. 아두이노 연결

두 개의 아두이노 우노를 연결하기 위해서는 두 개의 아두이노 우노가 준비되어 있어야 하는 것은 당연하다. 두 개의 아두이노 우노를 컴퓨터에 연결하면 장치 관리자에서 서로 다른 포트가 할당되는 것을 확인할 수 있다. 두 개의 아두이노가 각각 COM35와 COM36에 연결되었다고 가정하자.

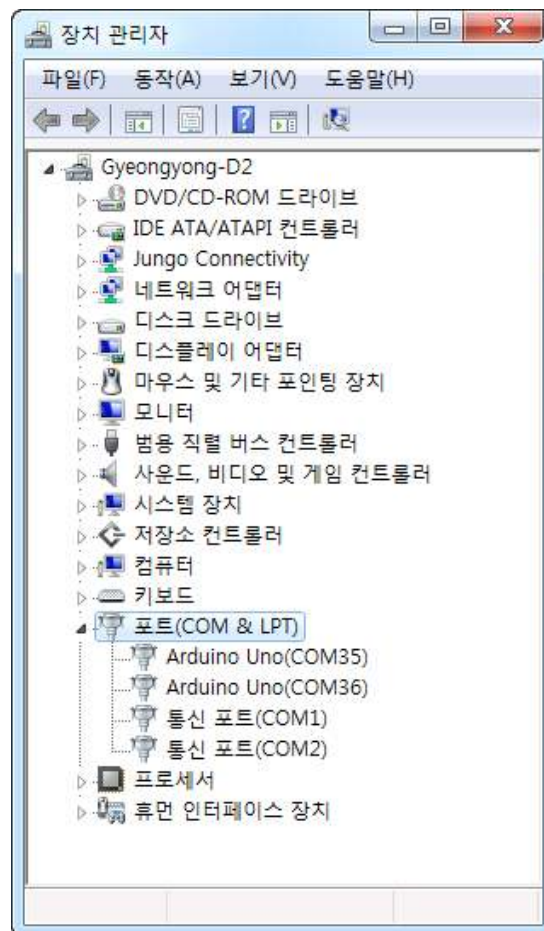


그림 1. 두 개의 아두이노 우노 연결

두 개의 아두이노를 UART 통신을 위해 연결하는 것은 RX와 TX를 서로 교차하여 연결하면 된다. 하지만 아두이노 우노는 하드웨어적으로 하나의 UART 통신만을 지원하므로 컴퓨터로 결과를 확인할 수 있도록 소프트웨어로 UART 통신을 에뮬레이션하는 SoftwareSerial을 함께 사용한다. 그림 1에서 COM36에 연결된 아두이노를 슬레이브라고 하고, COM35에 연결된 아두이노를 마스터라고 하자.¹⁾

1) UART 시리얼 통신은 1:1 통신으로 마스터-슬레이브 개념이 흔히 사용되지는 않지만 이 장에서는 COM36에 연결된 아두이노가 정보를 수집하여 COM35에 연결된 아두이노로 데이터를 전송하도록 할 것이므로 COM36에 연결된 아두이노를 슬레이브, COM35에 연결된 아두이노를 마스터라고 지칭한다.

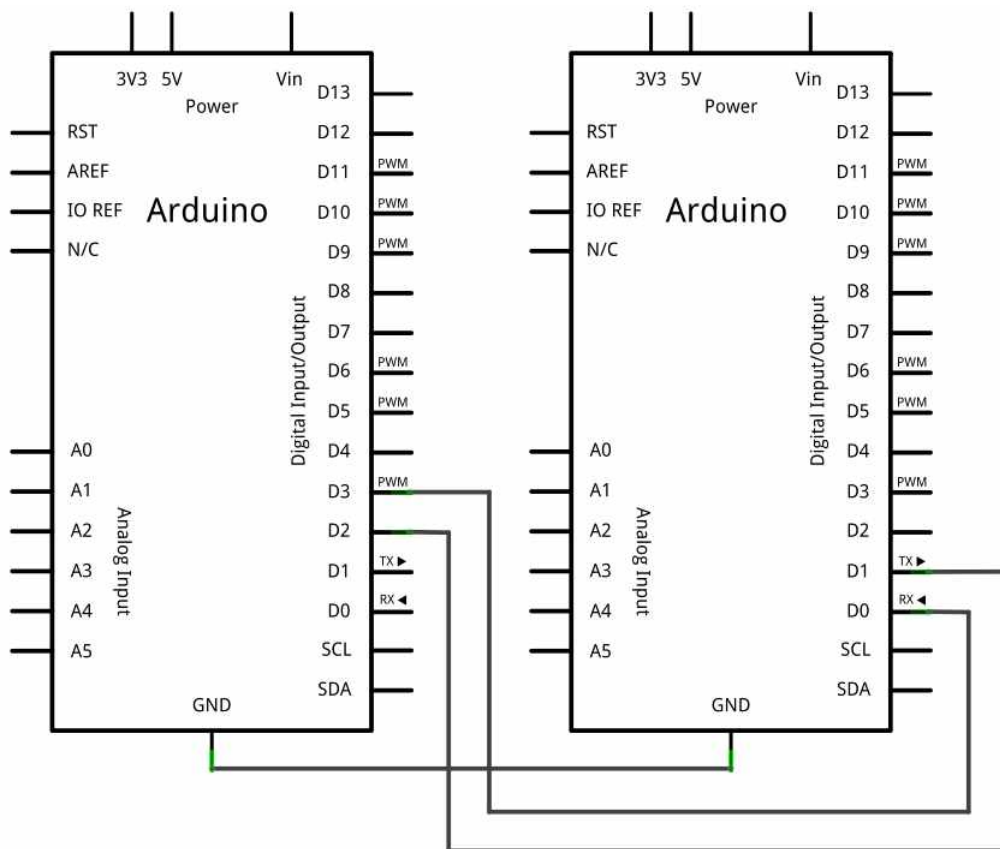


그림 2. 두 개의 아두이노 우노 연결 회로도

그림 2에서 왼쪽에 있는 아두이노가 마스터로 (SoftwareSerial을 통해) 슬레이브의 데이터를 받아 (Serial을 통해) 컴퓨터로 전송하여 시리얼 모니터로 표시되도록 한다. 오른쪽에 있는 아두이노는 슬레이브로 (Serial을 통해) 마스터로 카운터 값을 전송한다. 먼저 슬레이브에 스케치 1을 업로드 해보자.

스케치 1. 아두이노 사이의 UART 시리얼 통신 - 슬레이브

```
int count = 0;                                // 카운터

void setup() {
    Serial.begin(9600);                        // UART 시리얼 포트 초기화
}

void loop() {
```

```
count = count + 1;
Serial.println(count);           // 카운터 값 전송

delay(1000);                     // 1초 대기
}
```

스케치 1은 하드웨어 UART 포트를 통해 1초에 1씩 증가하는 카운터 값을 전송한다. 슬레이브에 스케치 1을 업로드하고 COM36 포트를 시리얼 모니터로 확인해 보면 전송되는 값을 확인할 수 있다. 단, 슬레이브와 마스터가 연결되어 있다면 동일한 핀의 중복 사용으로 인해 프로그램을 업로드에서 오류가 발생할 수 있으므로 마스터와의 연결선을 제거하고 업로드 하도록 한다.

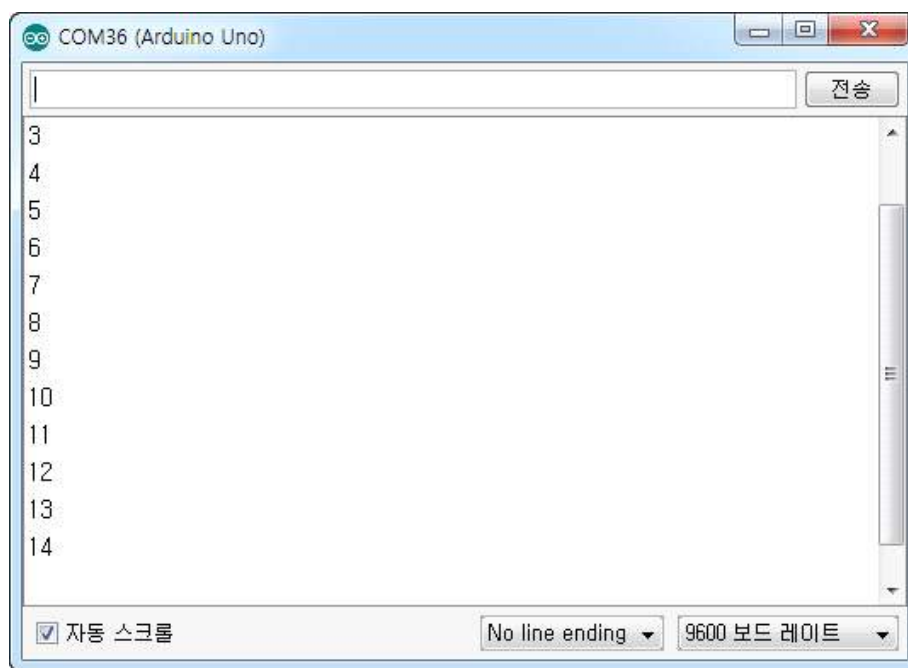


그림 3. 스케치 1 실행 결과

마스터에는 스케치 2를 업로드 한다. 스케치 2는 2개의 UART 통신을 사용하며 하나는 슬레이브로부터 전송된 데이터를 받아들이기 위한 SoftwareSerial 포트이며, 다른 하나는 슬레이브의 데이터를 컴퓨터로 전송하여 시리얼 모니터로 출력하기 위한 Serial 포트이다.

스케치 2. 아두이노 사이의 UART 시리얼 통신 - 마스터

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(2, 3);           // 슬레이브 데이터 입력 포트

void setup() {
  Serial.begin(9600);                   // 컴퓨터 연결 포트
  mySerial.begin(9600);
}

void loop() {
  while(mySerial.available()){          // 슬레이브로 부터의 데이터 수신 확인
    char data = mySerial.read();         // 슬레이브 데이터 읽기
    Serial.print(data);                 // 컴퓨터로 재전송
  }
}
```

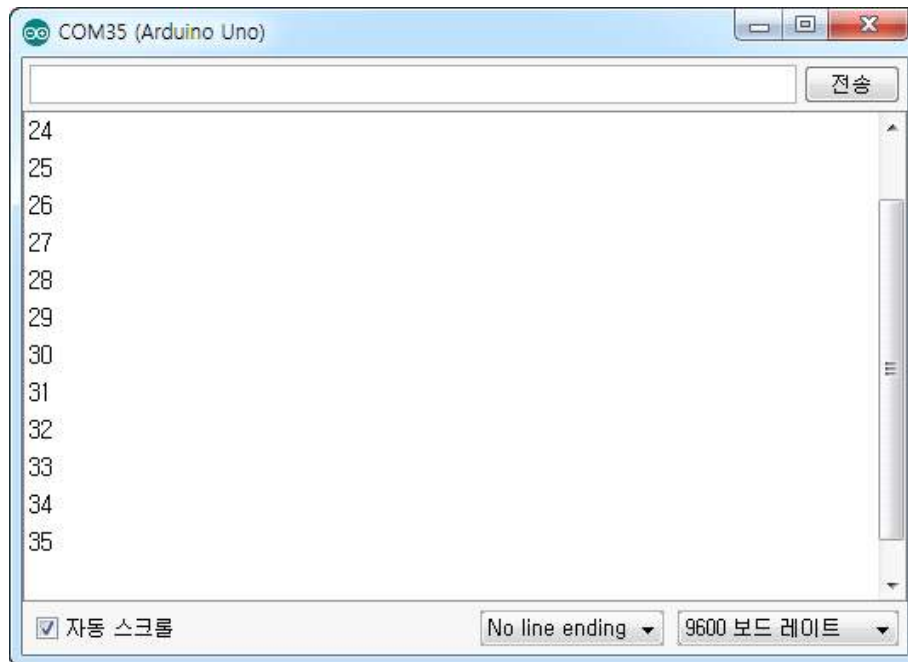


그림 4. 스케치 2 실행 결과

스케치 2의 실행 결과는 그림 4와 같으며 그림 3과의 차이를 찾아보기 어렵다. 무엇이 달라졌

는가? 바로 시리얼 모니터가 연결된 포트 번호에 차이가 있다. 그림 3은 슬레이브가 직접 컴퓨터로 데이터를 전송한 결과인 반면, 그림 4는 슬레이브가 마스터로 데이터를 전달하고 마스터가 이를 다시 컴퓨터로 전달한 결과이다.

슬레이브 역시 마스터와의 통신을 위해 SoftwareSerial을 사용할 수 있다. SoftwareSerial을 사용하면 슬레이브에 프로그램을 업로드 할 때 마스터와의 연결선을 제거하지 않아도 된다. 스케치 3은 스케치 1을 SoftwareSerial을 사용하도록 수정한 예이다.

스케치 3. SoftwareSerial 사용 - 슬레이브

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(2, 3);           // 마스터 연결 시리얼 포트

int count = 0;                           // 카운터

void setup() {
    mySerial.begin(9600);                 // UART 시리얼 포트 초기화
}

void loop() {
    count = count + 1;
    mySerial.println(count);              // 카운터 값 전송

    delay(1000);                          // 1초 대기
}
```

2. 원격 온도 측정

위의 예는 슬레이브가 지속적으로 마스터로 데이터를 전송하는 예이다. 이를 수정하여 마스터가 데이터를 요구하는 경우에만 슬레이브가 데이터를 전송하는 프로그램을 작성해 보자. 두 개의 아두이노 우노를 UART를 통해 연결하고 슬레이브에는 온도센서를 연결하자. 마스터는 일정한 시간 간격으로 슬레이브로 현재 온도를 전송하도록 요청하고 슬레이브는 마스터의 요청이 있는 경우에만 온도를 마스터로 전송하도록 해보자. 슬레이브에는 그림 5와 같이 온도센서를 연결하자. 사용한 온도 센서는 TMP36으로 트랜지스터와 동일한 형태를 가지고 있으

며 온도에 비례하는 전압을 아날로그 값으로 출력한다.

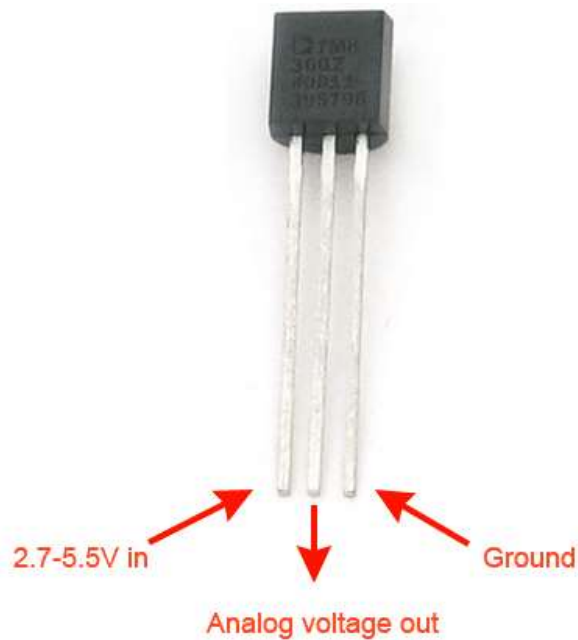


그림 5. TMP36 온도센서

TMP36 온도센서의 출력 전압 특성은 그림 6과 같다.

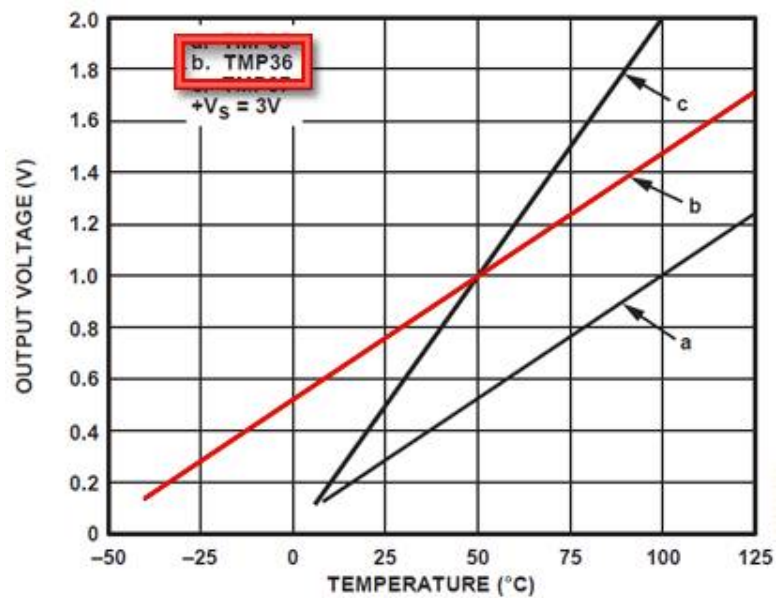


그림 6. TMP36 온도센서의 출력 전압 특성

출력 전압을 온도로 변환하기 위해서는 다음 수식을 사용하면 된다. 이 때 출력 전압은 mV 단위를 사용한다.

$$T(^{\circ}\text{C}) = \frac{V_{out} - 500}{10}$$

출력 전압을 아날로그 입력 핀으로 읽으면 0에서 1023 사이의 디지털 값으로 변환되어 아두이노에서 읽힌다. 따라서 V_{out} 은 실제 입력값 A_{in} 에서 다음과 같이 계산할 수 있다. 아두이노 우노를 사용하므로 아날로그 기준 전압은 5V로 가정하였다.

$$V_{out} = \frac{A_{in} \times 5000}{1024}$$

그림 7과 같이 슬레이브 아두이노에 온도센서를 연결하자.

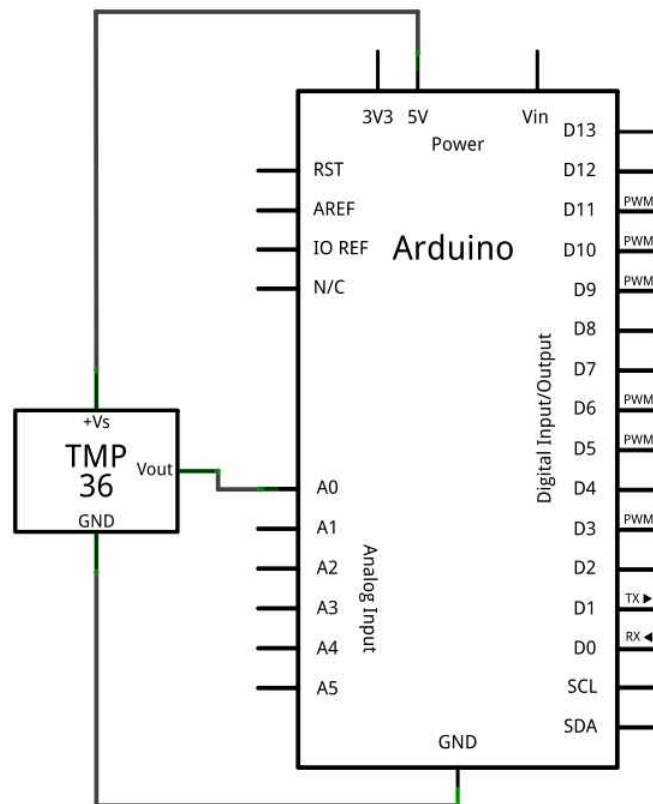


그림 7. 온도 센서 연결 회로도

스케치 4는 온도 센서를 통해 현재 온도를 측정하여 시리얼 모니터로 출력하는 스케치의 예이다.

스케치 4. 온도 센서 읽기

```
int temperature_pin = A0;           // 온도 센서 연결 핀

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

void loop() {
  int reading = analogRead(temperature_pin); // ADC 값 읽기
  float voltage = reading * 5.0 / 1024.0;    // Volt 단위 전압으로 변환
  float temperature = (voltage - 0.5) * 100; // 온도로 변환

  Serial.print("Current Temperature : ");
  Serial.println(temperature);

  delay(1000);
}
```

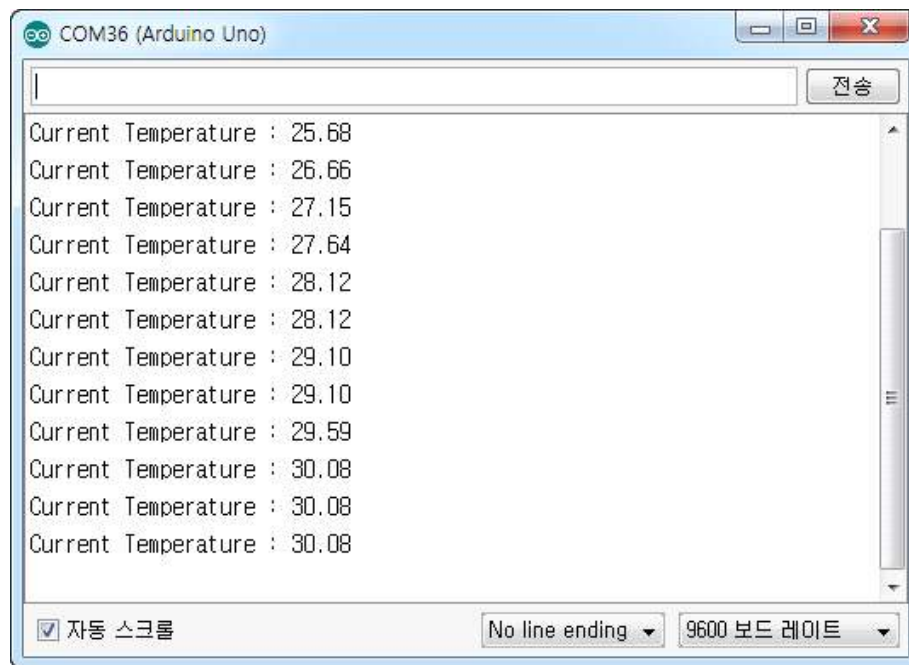


그림 8. 스케치 4 실행 결과

스케치 4를 이용하여 슬레이브에는 마스터로부터 현재 온도에 대한 요청을 나타내는 문자 'T'가 전송된 경우 현재 온도를 알려주는 스케치 5를 업로드 한다. 업로드 시의 충돌을 피하기 위해 스케치 5에서는 소프트웨어 시리얼 포트를 사용하였다.

스케치 5. 원격 온도 측정 - 슬레이브

```
#include <SoftwareSerial.h>

int temperature_pin = A0;           // 온도 센서 연결 핀
SoftwareSerial mySerial(2, 3);      // 마스터 연결 시리얼 포트

void setup() {
    mySerial.begin(9600);           // UART 시리얼 포트 초기화
}

void loop() {
    if(mySerial.available()){       // 마스터로부터의 데이터 전송 확인
        char data = mySerial.read();
    }
}
```

```

        if(data == 'T'){                                // 현재 온도를 요구한 경우 온도 반환
            mySerial.print("Current temperature is ");
            mySerial.print(get_temperature());
            mySerial.println(" degrees C");
        }
    }
}

float get_temperature() {
    int reading = analogRead(temperature_pin);           // ADC 값 읽기
    float voltage = reading * 5.0 / 1024.0;              // Volt 단위 전압으로 변환
    float temperature = (voltage - 0.5) * 100;           // 온도로 변환

    return temperature;
}

```

마스터에는 시리얼 모니터로부터 임의의 문자를 입력 받고 이를 슬레이브로 전달해주는 기능과, 슬레이브로부터 전달된 현재 온도를 시리얼 모니터로 출력하는 스케치 6을 업로드 하자.

스케치 6. 원격 온도 측정 - 마스터

```

#include <SoftwareSerial.h>

SoftwareSerial mySerial(2, 3);                // 슬레이브 통신 포트

void setup() {
    Serial.begin(9600);                       // 컴퓨터 연결 포트
    mySerial.begin(9600);
}

void loop() {
    // 시리얼 모니터 --> 마스터 --> 슬레이브
    if(Serial.available())
        mySerial.write(Serial.read());
}

```

```
// 슬레이브 --> 마스터 --> 시리얼 모니터  
if(mySerial.available())  
    Serial.write(mySerial.read());  
}
```

마스터에 연결된 시리얼 포트(COM35)에 연결된 시리얼 모니터를 실행시키고, 문자를 입력하여 현재 온도가 시리얼 모니터로 출력되는 지 확인해 보자.

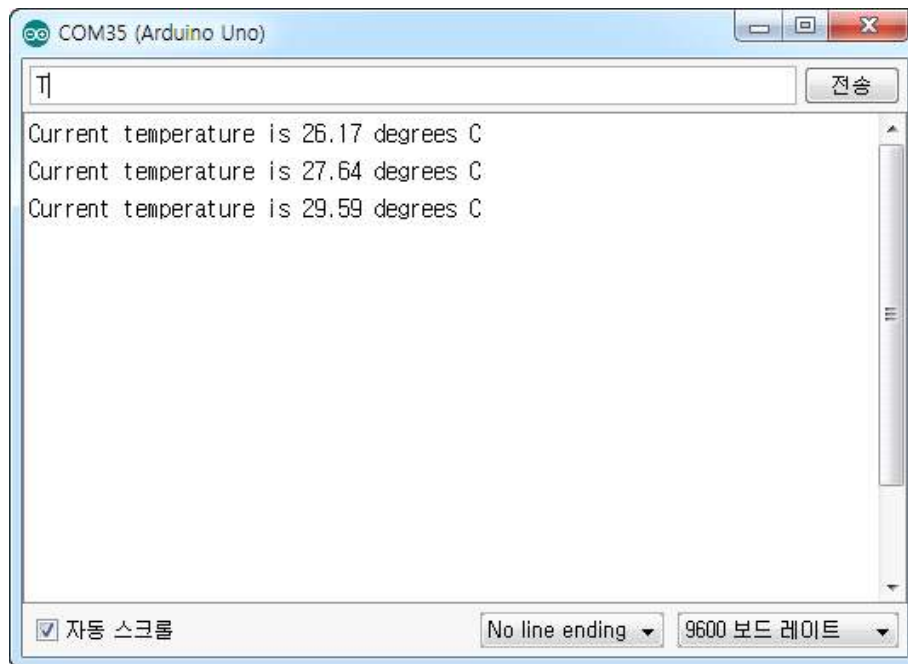


그림 9. 스케치 5 및 스케치 6 실행 결과