

Chapter 107. 시리얼 통신

앞 장에서 LED를 이용하여 마이크로컨트롤러에서 주변장치로 데이터를 전송하는 방법을 살펴보았다. 범용 입출력 핀으로는 한 번에 한 비트의 데이터 만을 전송할 수 있으므로 8개의 LED를 제어하기 위해 8개의 범용 입출력 핀을 사용하였다. 이를 병렬 (parallel) 통신이라고 이야기 한다. 하지만 마이크로컨트롤러에서 사용할 수 있는 범용 입출력 핀의 수는 그리 많지 않으므로 병렬 통신은 그리 선호되지 않는다. 대신 시리얼 (serial) 통신(또는 직렬 통신¹⁾)이 일반적으로 사용된다. 병렬 통신에서는 8개 비트를 8개의 입출력 핀을 이용하여 한 번에 전송한다면, 시리얼 통신은 1개 비트를 1개의 입출력 핀을 이용하여 8번에 나누어 전송하는 방법이다.

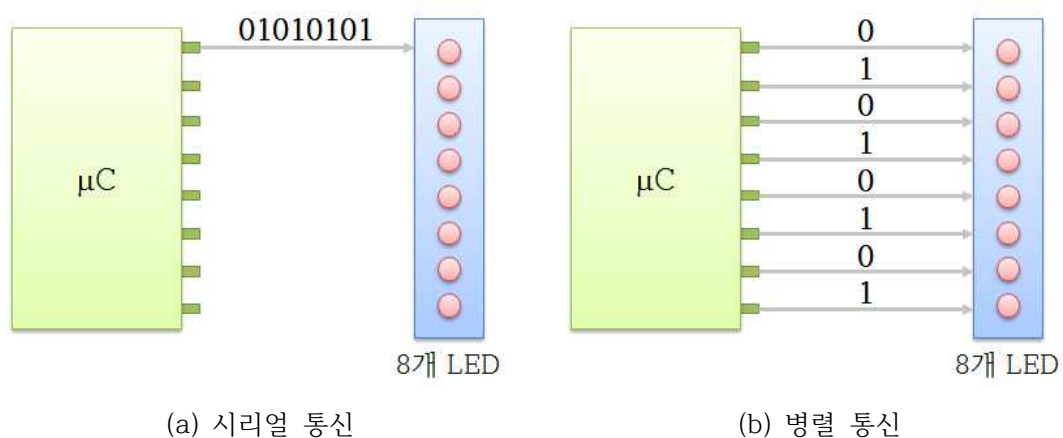


그림 1. 시리얼 통신과 병렬 통신

1. UART

시리얼 통신을 위한 방법에는 여러 가지가 있지만 이 장에서는 마이크로컨트롤러에서 흔히 사용되는 방법 중 하나인 UART(Universal Asynchronous Receive and Transmit)를 위주로 살펴본다.

시리얼 통신을 사용하기 위해서는 먼저 보내는 쪽과 받는 쪽에서 약속을 정하여야 하며 이를 프로토콜(protocol)이라고 한다. 마이크로컨트롤러는 0과 1의 이진값만을 가질 수 있으므로 0

1) 직렬 통신은 시리얼 통신과 동일한 의미이지만 일반적으로 '시리얼 통신'이라는 용어를 더 많이 사용하므로 이 책에서도 '시리얼 통신'이라는 용어를 사용한다.

은 GND, 1은 VCC로 데이터를 전송하고, 받는 쪽에서는 GND와 VCC를 다시 0과 1의 이진값으로 변환하여 사용하면 된다. 하지만 이것만으로는 충분하지 않다. 8번 데이터를 보내는 경우 받는 쪽에서는 얼마나 자주 데이터를 확인할 것인지 약속을 정하여야 한다. 1초에 한 번 0이나 1의 이진값을 보낸다고 가정해 보자. '01'의 데이터를 보낸 경우 받는 쪽에서 1초에 한 번 새로운 값을 검사한다면 정확한 데이터를 받을 수 있지만 0.5초에 한 번 새로운 값을 검사한다면 '0011'이라는 전혀 다른 데이터를 수신하게 된다.

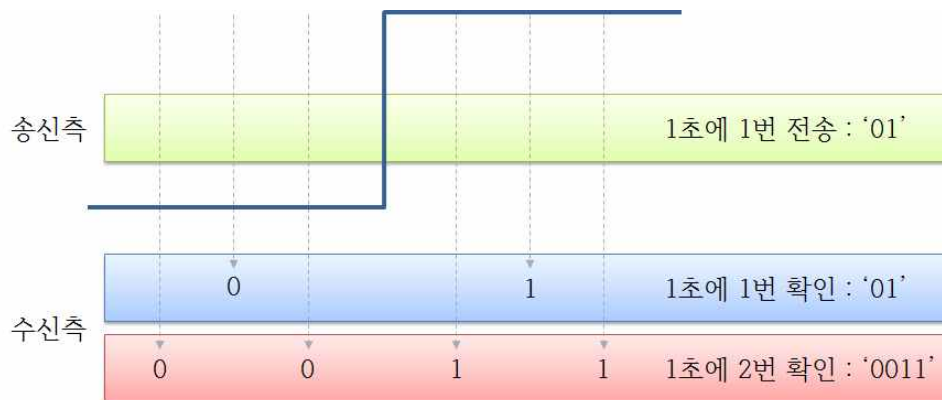


그림 2. 송신 속도와 수신 속도

UART에서는 보내는 쪽과 받는 쪽에서 데이터를 보내는 속도를 보율(baud rate)로 정하게 된다. 보율은 변조 속도를 나타내는 단위로 프랑스의 과학자 Jean Maurice Emile Baudot를 기념하여 데이터 전송 속도를 표현하기 위해 사용되었다. 초기 데이터 통신에서는 통신 회선을 통해 1초 동안 전달되는 데이터 비트의 개수를 나타내는 단위로 모뎀이나 데이터 전송 속도를 표시하기 위해 사용되었으며 이 경우 흔히 사용되는 데이터 전송 속도 단위인 bps(Bits Per Second)와 동일하다. 하지만 최근 통신 기술의 발달로 인해 신호가 한 번 변할 때 한 비트 이상의 정보를 표현하는 것이 가능해짐에 따라 bps는 보율보다 크거나 같은 값을 가지게 되었다. 그림 3은 보율과 bps를 비교한 그림으로 1초에 신호가 4번 변하고 있으므로 보율은 4이지만 신호가 한 번 변할 때 2비트의 데이터가 전달되므로 bps는 8이 된다.

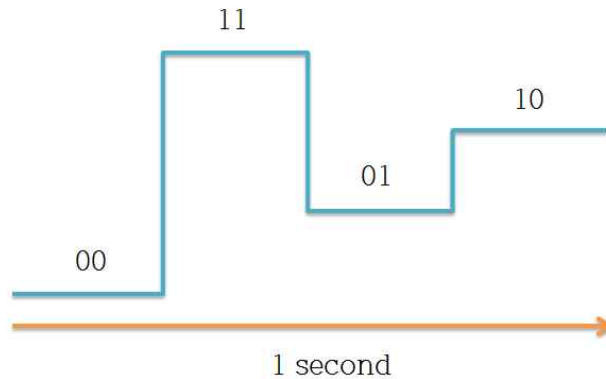


그림 3. 보울과 bps

송신측과 수신측이 동일한 속도로 데이터를 주고받는다 고 하여 정확하게 통신이 이루어지는 것은 아니다. 송신측은 항상 데이터를 보내는 것이 아니며 필요한 경우에만 데이터를 보낸다. 따라서 수신측은 언제 송신측이 데이터를 보내는지 그리고 어디서부터가 송신측에서 보낸 데이터의 시작인지 알아낼 수 있는 방법이 필요하다. 이를 위해 UART에서는 '0'의 시작 비트(start bit)와 '1'의 정지 비트(stop bit)를 사용한다. UART는 바이트 단위 통신을 흔히 사용하며 여기에 시작 및 정지 비트가 추가되어 10비트가 전송된다. 데이터가 전송되지 않는 상태에서 데이터 핀은 항상 '1'의 상태에 있다. 데이터가 수신되기 시작하면 '0'의 시작 비트가 들어오고 이어서 8비트의 실제 데이터가 들어온 이후 데이터 전송이 끝났음을 알리는 '1'의 비트가 들어오게 된다.



그림 4. UART의 데이터 전송

UART 통신은 전이중 방식 (full duplex) 통신으로 송신과 수신을 동시에 진행할 수 있으며 이를 위해 두 개의 범용 입출력 핀을 필요로 한다. 범용 입출력 핀을 통해 들어오는 데이터는 위에서 설명한 프로토콜에 의해 그 의미를 파악하여야 하며 이를 위해 전용의 하드웨어가 마이크로컨트롤러 내에 구현되어 있다. ATmega328에서 PD0과 PD1번 핀(아두이노의 디지털 0번과 1번 핀)을 사용하여 통신을 수행하며 이를 흔히 '시리얼 포트'라고 이야기 한다. 아두이

노에서는 UART 시리얼 포트를 시리얼 방식의 프로그램 업로드를 위해서도 사용하고 있다. 아두이노 우노 보드에서 디지털 0번 핀은 'RX', 디지털 1번 핀은 'TX'로 표시된 것을 확인할 수 있다. 'RX'는 'Receive'의, 'TX'는 'Transmit'의 약어로 두 개의 장치를 UART 시리얼로 연결하는 경우 RX와 TX는 서로 교차하여 연결하여야 한다. 예를 들어 컴퓨터와 아두이노 우노 보드를 연결하는 경우 컴퓨터의 RX는 아두이노 우노 보드의 TX로, 컴퓨터의 TX는 아두이노 우노 보드의 RX로 연결하여야 한다. 하지만 컴퓨터의 경우 RS-232 시리얼 연결을 사용하여 하며 RS-232에서 사용하는 신호 레벨은 UART의 신호 레벨과 달라 별도의 변환 장치를 사용하여 레벨을 변환시켜 주어야만 사용 가능하다.²⁾

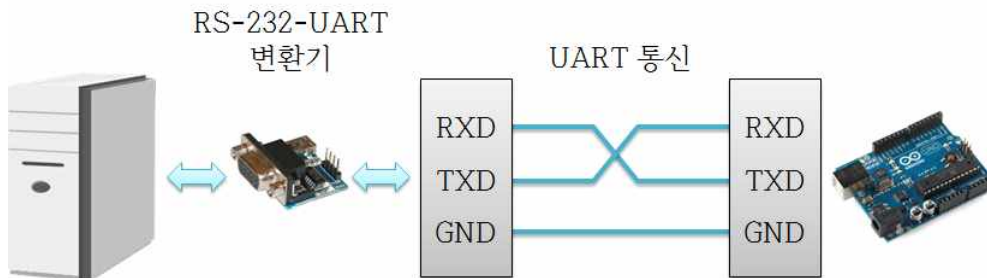


그림 5. 컴퓨터와 아두이노 우노 보드의 시리얼 연결 ¹³⁾

아두이노 우노 보드에는 시리얼 방식의 업로드를 위해 USB-시리얼 변환 장치가 포함되어 있으므로 이를 통해 RS-232를 UART로 변환하는 장치 없이도 컴퓨터와 시리얼 통신이 가능하다. 아두이노 우노 보드를 컴퓨터에 연결하면 우노 보드는 가상의 COM 포트, 즉, 시리얼 포트에 연결된 것으로 장치관리자에 표시되며 이 포트를 통해 UART 통신이 가능하다. 브레드 보드에 기본 회로로 구성한 경우에는 별도의 USB-시리얼 변환 장치가 필요하다.

2) 일부 마이크로컨트롤러 보드의 경우 컴퓨터와 연결할 수 있도록 신호 레벨 변환 장치가 포함되어 있는 경우도 있지만 아두이노 우노에는 신호 레벨 변환 장치가 포함되어 있지 않다.

3) 이미지 출처 : <http://arduino.cc>

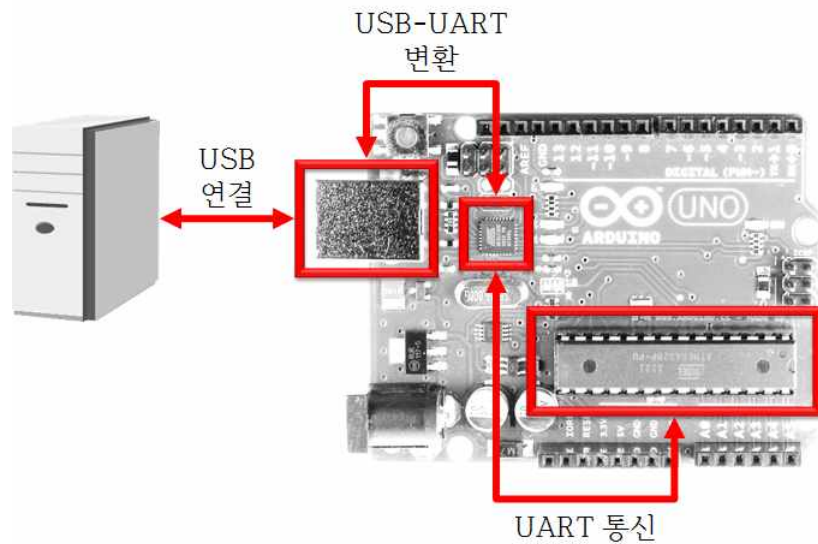
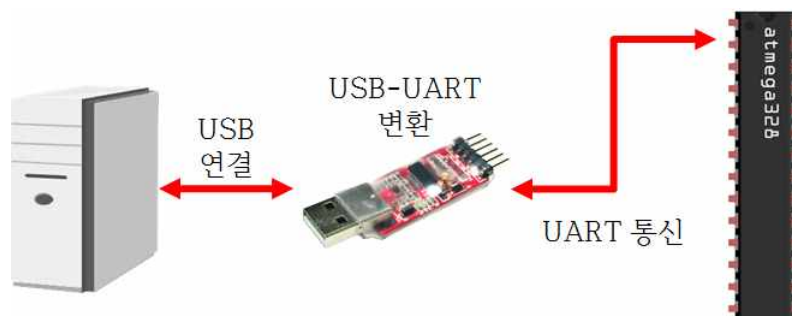
그림 6. 컴퓨터와 아두이노 우노 보드의 시리얼 연결 ²⁴⁾

그림 7. 컴퓨터와 기본 회로의 시리얼 연결

2. UART 통신을 위한 레지스터

하드웨어가 준비되었으면 이제 컴퓨터와 UART 통신을 해 보자. 컴퓨터와 UART 통신을 테스트하기 위해서는 컴퓨터와 두 개의 USB 연결이 필요하다. 하나는 아두이노 우노 보드에 프로그램 업로드 하기 위한 ISP 연결이며, 다른 하나는 컴퓨터와 UART 통신을 위한 연결이다. 두 연결은 모두 가상의 COM 포트로 장치 관리자에 나타나므로 포트 번호를 잘 기억하여야 한다.

4) 이미지 출처 : <http://arduino.cc>

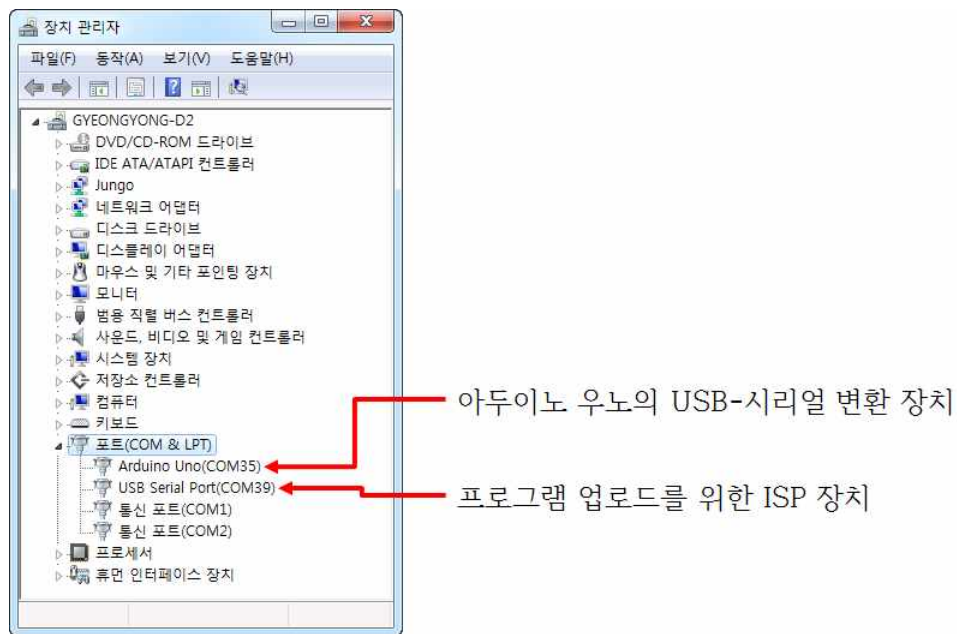


그림 8. 아두이노 우노 보드와 연결된 가상 COM 포트⁵⁾

컴퓨터와 두 개의 USB 연결을 마쳤으면 코드 1을 업로드 하고 ISP 장치를 통해 프로그램을 업로드 해 보자.

코드 1. 컴퓨터와의 UART 통신 - C 스타일

```
#define F_CPU 16000000L
#include <avr/io.h>
#include <util/delay.h>

void UART_INIT(void) {
    UCSRA |= _BV(U2X0);                // 2배속 모드

    UBRR0H = 0x00;                      // 통신 속도 (보율) 설정
    UBRR0L = 207;

    // 비동기, 8비트 데이터, 패리티 없음, 1비트 정지 비트 모드
    UCSRC |= 0x06;
```

5) 포트 번호는 컴퓨터에 따라 그리고 사용하는 ISP 장치나 USB-시리얼 변환 장치에 따라 달라질 수 있다.

```

    UCSRB |= _BV(RXEN0);           // 송수신 가능
    UCSRB |= _BV(TXEN0);
}

unsigned char UART_receive(void)
{
    while( !(UCSR0A & (1<<RXC0)) );    // 데이터 수신 대기
    return UDR0;
}

void UART_transmit(unsigned char data)
{
    while( !(UCSR0A & (1<<UDRE0)) );    // 송신 가능 대기
    UDR0 = data;                       // 데이터 전송
}

int main(int argc, char *argv[])
{
    unsigned char data;

    UART_INIT();                       // UART 통신 초기화
    while (1) {
        data = UART_receive();         // 데이터 수신
        UART_transmit(data);           // 수신된 문자를 에코 백
    }

    return 0;
}

```

코드 1은 컴퓨터에서 문자를 전송하면 아두이노 우노에서 이를 수신하고 이를 다시 컴퓨터로 되돌려 보내주는(에코 백, echo back) 프로그램이다. 시리얼 포트에 오는 데이터를 확인하기 위해 새로운 애드온을 설치해 보자. ‘도구 → 확장 관리자’를 실행시키고 ‘TerminalWindow’를 설치해 보자.

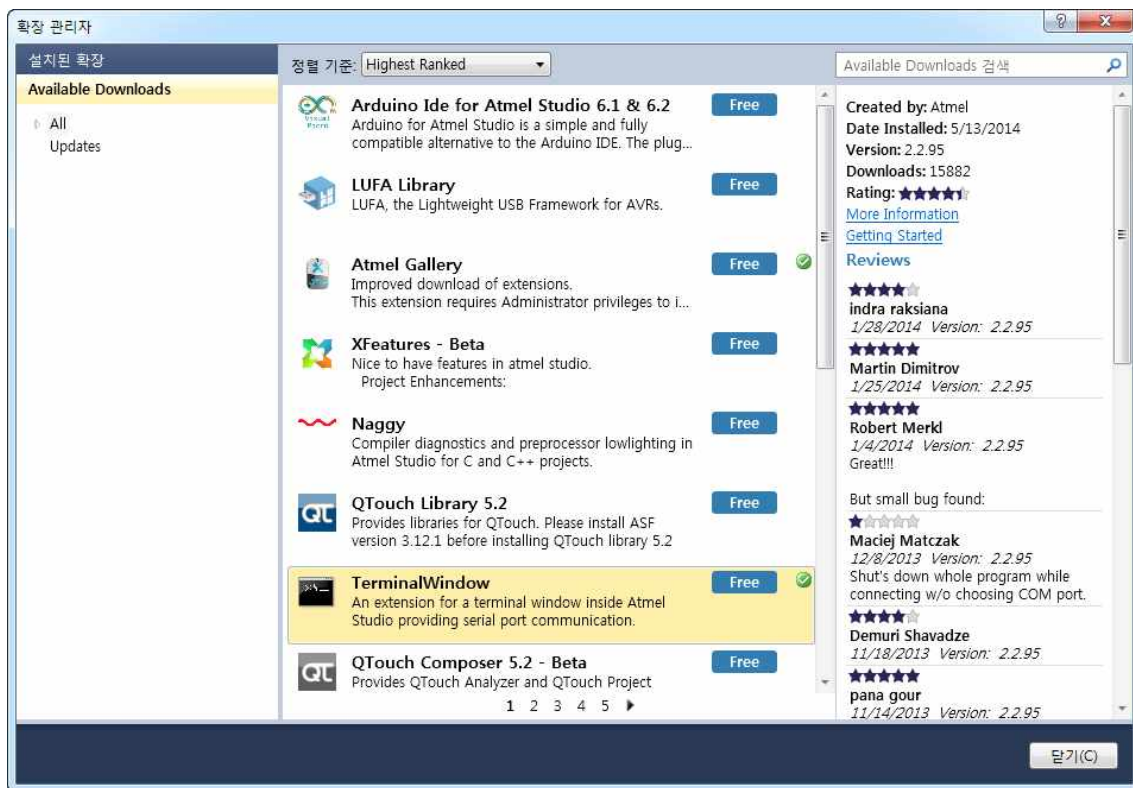


그림 9. 확장 관리자에서의 Terminal Window

터미널 윈도우를 설치하면 '보기 → Terminal Window'라는 메뉴 항목이 새롭게 만들어진다.

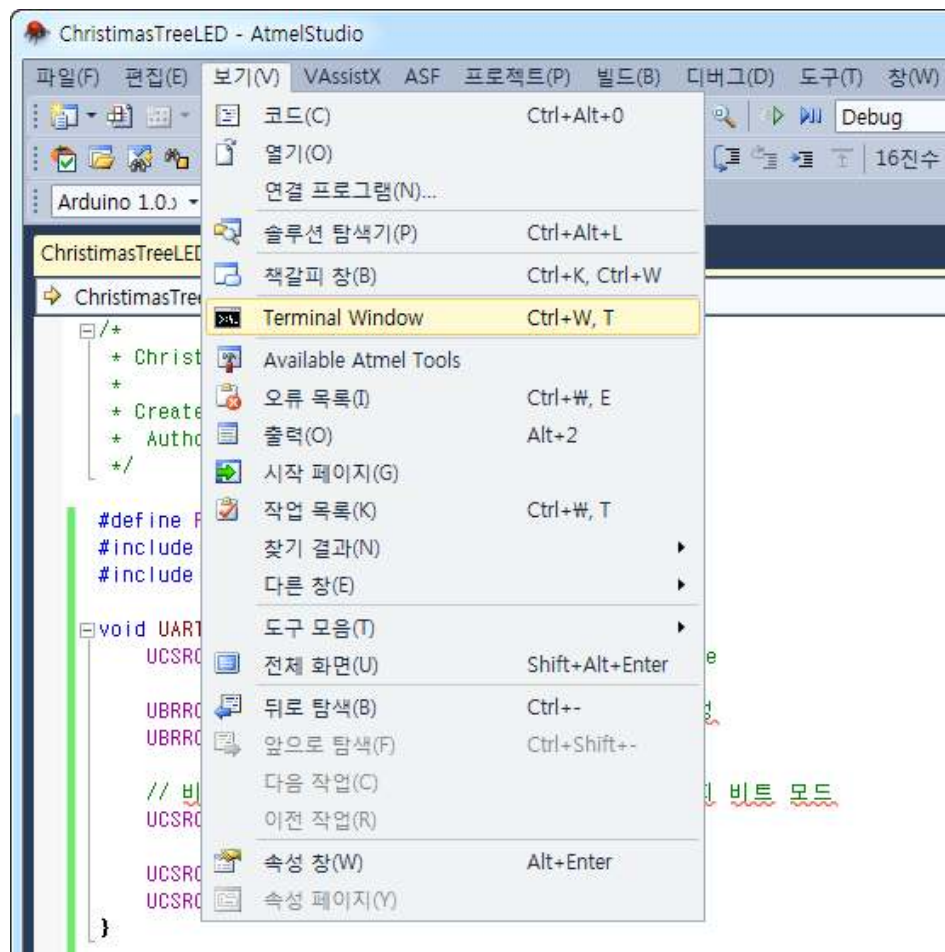


그림 10. 터미널 윈도우 메뉴 항목

터미널 윈도우를 실행시키면 시리얼 포트를 통해 송수신되는 데이터를 확인할 수 있는 창이 나타난다. 왼쪽 위의 포트 목록에서 아두이노 우노의 USB-시리얼 변환 장치에 해당하는 포트 번호를 선택하고 'Connect' 버튼을 누른다. 아래쪽에 문자열을 입력한 다음에 엔터키를 눌러 보자. 아두이노 우노로부터 수신된 문자가 'Receive' 창에 표시되는가? 코드 1에서 보울을 9600으로 설정하였으므로 터미널 윈도우에서도 보울을 동일한 값으로 선택하여야 한다.

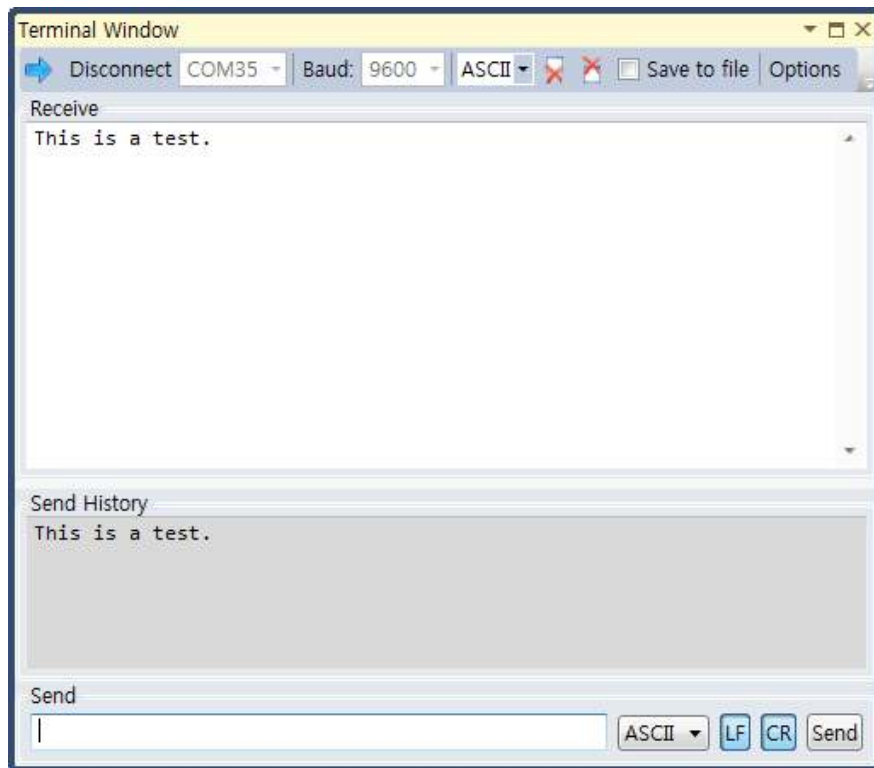


그림 11. 터미널 윈도우를 통한 UART 통신 확인

3. UART 통신을 위한 레지스터

코드 1을 자세히 살펴보면 생소한 레지스터가 여러 개 사용된 것을 확인할 수 있다. 코드 1에는 UCSR0A, UCSR0B, UCSR0C, UBRR0H, UBRR0L, UDR0 등 무려 6개의 레지스터가 사용되고 있다. 이들 레지스터를 하나씩 살펴보자. 단, 각 레지스터의 모든 내용을 자세히 설명하지는 않을 것이며 코드 1에서 사용된 내용을 위주로 살펴볼 것이다.

(1) UCSR0A

UCSR0A는 'USART Control and Status Register 0 A'의 약어로 단어 의미 그대로 UART 통신을 위한 제어와 상태를 알아보기 위한 레지스터 중 하나이다. 코드 1에서는 2배속 모드 설정과 송신 및 수신 가능한지의 여부를 판단하기 위한 목적으로 사용되었다. 먼저 2배속 설정을 위해서는 UCSR0A 레지스터의 1번 비트인 'U2X0' 비트를 1로 설정해 주면 된다.

* UART ?? USART ??

UART는 'Universal Asynchronous Receiver Transmitter'의 약어이며 USART는 UART에 'Synchronous'가 추가된 것으로 지금까지 별다른 구별 없이 사용하였다. 두 단어의 차이점은 어디에서 오는 것일까? AVR에서의 USART는 세 가지 모드에서 동작이 가능하다.

- (1배속) 비동기 모드
- 비동기 2배속 모드
- 동기 모드

비동기 모드는 별도의 클록을 사용하지 않고 데이터를 송수신 하는 모드로 시작 비트와 정지 비트를 통해 데이터 동기화가 이루어진다. 흔히 'UART'라고 하는 직렬 통신이 USART의 비동기 모드에 해당한다.

2배속 모드는 비동기 통신에서만 사용할 수 있는 모드로 전송 속도를 2배로 증가시킬 수 있는 방법이다. 비동기 통신에서는 데이터의 시작과 끝을 위한 별도의 정보가 없으므로 한 비트의 데이터를 확인하기 위해 여러 개의 샘플(16개)을 취하여 판단하게 된다. 2배속 모드는 전송 속도를 증가시킬 수는 있지만 데이터 확인을 위해 사용할 수 있는 샘플의 수가 절반(8개)으로 줄어들게 되므로 일반적인 1배속 모드에 비해 보다 정확한 클록이 요구된다.

동기 모드는 데이터 동기화를 위해 별도의 클록 신호를 사용한다. ATmega328에서 데이터 동기화를 위한 클록 신호는 XCK 핀(PD4)을 사용한다. 동기 모드에서는 비동기 모드에서 데이터 동기화를 위해 사용하는 시작 비트와 정지 비트는 필요하지 않으므로 데이터 전송 효율을 높일 수 있는 장점이 있다.

ATmega328에서 비동기 전송을 위한 XCK 핀은 아두이노에서 디지털 4번 핀으로 사용되고 있으며 아두이노에서도 2배속의 비동기 전송 모드를 기본으로 사용하고 있다. 따라서 레지스터의 설명을 위해 'USART'라는 단어를 사용하기는 하지만 이 장에서의 설명은 2배속 비동기 전송 모드를 기본으로 한다.

```
UCSR0A |= _BV(U2X0);
```

‘U2X0’는 비트의 이름으로 비트의 번호에 비해 기억하기 쉽도록 ‘io.h’ 파일에 정의되어 있다. ‘_BV’는 지정한 비트 번호만을 1로 하고 나머지를 0으로 하는 비트 마스크를 생성하는 매크로이므로 2배속 모드 설정 문장은 다음과 같이 표현할 수 있다.

```
UCSR0A |= (1 << U2X0);
UCSR0A |= (1 << 1);
UCSR0A = UCSR0A | (1 << 1);
```

UCSR0A 레지스터의 중요한 기능 중 하나는 데이터를 수신하거나 송신할 준비가 되어 있는지를 확인하는 기능으로 이를 위해 7번 RXC0 비트와 5번 UDRE0 비트를 사용할 수 있다. RXC0 비트는 데이터 수신 버퍼에 데이터가 도착하고 읽지 않은 상태인 경우 1로 설정되므로 데이터가 수신되었는지의 여부를 판단하기 위해 사용할 수 있다. 코드 1에서는 데이터가 수신 될 때까지 기다리기 위해, 즉 RXC0 비트가 1이 될 때까지 기다리기 위해 while 문장을 사용하고 있다.

```
while( !(UCSR0A & (1<<RXC0)) );
```

이와 유사하게 데이터가 완전히 전송되고 다음 데이터를 전송할 준비가 되어 있는지를 확인하기 위해 UDRE0 비트를 사용할 수 있다.

```
while( !(UCSR0A & (1<<UDRE0)) );
```

한 가지 주의할 점은 6번 TXC0 비트 역시 비슷한 기능을 한다는 점이다. 데이터시트의 설명 상으로는 비슷해 보이지만 UDRE0 비트 대신 TXC0를 사용하면 코드 1은 정상적으로 동작하지 않는다. 이유가 무엇일까?

UART 전송에 데이터 이동을 위해 사용되는 레지스터에는 UDR(USART Data Register)과 쉬프트 레지스터의 두 개가 있다. UDR은 상대적으로 빠른 CPU와 상대적으로 느린 USART 사이의 중계자 역할을 하는 버퍼이고 쉬프트 레지스터는 CPU에서 보낸 병렬 데이터를 USART

를 통해 보낼 수 있는 직렬 데이터로 변환하기 위해 필요한 레지스터이다.

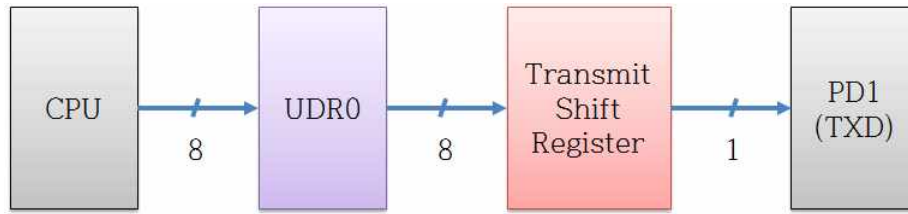


그림 12. USART 데이터 전송을 위한 레지스터 구조

TXC0 비트는 쉬프트 레지스터에서 데이터가 모두 전송되고 UDR0 레지스터에 데이터가 없는 경우 세트된다. 반면 UDRE0 비트는 UDR0 레지스터에 데이터가 없는 경우 세트된다. 즉, 두 비트가 세트되는 상황에 차이가 있다. 또한 UDRE0 비트의 디폴트값은 1로 데이터를 전송할 수 있는 상태로 시작하므로 프로그램 시작과 더불어 데이터를 전송할 수 있지만, TXC0 비트의 디폴트값은 0으로 프로그램 시작 시에 전송 가능 상태를 확인할 수 없다. 이외에도 몇 가지 이유로 데이터 전송 시점을 결정하기 위해서는 UDRE0 비트를 검사하는 것이 편리하다. UCSR0A 레지스터의 구조는 그림 13과 같으며 각 비트의 의미는 표 1과 같다.

비트	7	6	5	4	3	2	1	0
	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0
읽기/쓰기	R	R/W	R	R	R	R	R/W	R/W
초깃값	0	0	1	0	0	0	0	0

그림 13. UCSR0A 레지스터 구조

비트	이름	설명
7	RXC0	수신 버퍼(UDR0)에 읽지 않은 문자가 있을 때 1이 되고 버퍼가 비어있을 때 0이 된다. UCSR0B 레지스터의 RXCIE0 비트와 함께 사용되어 수신 완료 인터럽트를 발생시킬 수 있다. (Receive Complete)
6	TXC0	쉬프트 레지스터의 데이터가 송신되고 송신 버퍼(UDR0)가 비어 있을 때 1이 된다. UCSR0B 레지스터의 TXCIE0 비트와 함께 사용되어 송신 완료 인터럽트를 발생시킬 수 있다. (Transmit Complete)
5	UDRE0	송신 버퍼(UDR0)가 비어 있어 데이터를 받을 준비가 되어 있을 때 1이 된다. UCSR0B 레지스터의 UDRIE0 비트와 함께 사용되어 송신 데이터 레지스터 준비 완료 인터럽트를 발생시킬 수 있다. (USART

		Data Register Empty)
4	FE0	수신 데이터가 없는 경우 수신 값은 HIGH 상태이며 LOW 상태로 바뀌면서 데이터 프레임 수신에 시작되고 프레임의 마지막 정비 비트 수신시에 다시 HIGH 상태로 바뀐다. 마지막에 HIGH 상태를 수신하지 못하면 프레임 수신시에 오류가 발생한 것을 나타내며 FE0은 1의 값을 가진다. (Frame Error)
3	DOR0	수신 버퍼가 가득 찬 상태에서 수신 시프트 레지스터에 새로운 문자가 수신 완료되고 다시 그 다음 문자의 시작 비트가 검출되는 오버런 상황이 발생한 경우 1의 값을 가진다. (Data Overrun Error)
2	UPE0	수신 버퍼에 저장된 문자 데이터에 패리티 오류가 발생한 경우 1의 값을 가진다. 패리티 비트 사용이 설정된 경우에만 사용할 수 있다. (USART Parity Error)
1	U2X0	비동기 전송 모드에서만 사용되며, 2배속 모드이면 1의 값을 가지고 1배속 모드이면 0의 값을 가진다. (USART 2 transmission speed)
0	MPCM0	1개의 마스터 프로세서가 여러 개의 슬레이브 프로세서에게 특정 어드레스를 전송함으로써 1개의 슬레이브만을 지정하여 데이터를 전송하는 멀티 프로세서 통신 모드에서 1의 값을 가진다.

표 1. UCSR0A 레지스터 비트

(2) UCSR0B

UCSR0B 레지스터는 데이터 송수신을 가능하도록 설정하기 위해 사용되었다. 디폴트값으로 UART 통신의 송수신은 금지되어 있다.

```
UCSR0B |= _BV(RXEN0);      // 수신 가능
UCSR0B |= _BV(TXEN0);      // 송신 가능
```

UCSR0A 레지스터의 구조는 그림 14와 같으며 각 비트의 의미는 표 2와 같다.

비트	7	6	5	4	3	2	1	0
	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 14. UCSR0B 레지스터 구조

비트	이름	설명
7	RXCIE0	수신 완료 인터럽트 발생을 허용한다. (RX Complete Interrupt Enable)
6	TXCIE0	송신 완료 인터럽트 발생을 허용한다. (TX Complete Interrupt Enable)
5	UDRIE0	송신 데이터 레지스터 준비 완료 인터럽트 발생을 허용한다. (UDR empty Interrupt Enable)
4	RXEN0	USART 수신기의 수신 기능을 활성화한다. (RX ENable)
3	TXEN0	USART 송신기의 송신 기능을 활성화한다. (TX ENable)
2	UCSZ02	UCSR0C 레지스터와 함께 전송 데이터의 비트 수를 결정한다. (USART Character Size)
1	RXB80	데이터가 9비트인 경우 수신 데이터의 9번째 비트 저장을 위해 사용한다. 반드시 UDR0 레지스터보다 먼저 읽어야 한다.
0	TXB80	데이터가 9비트인 경우 전송 데이터의 9번째 비트 저장을 위해 사용한다. 반드시 UDR0 레지스터보다 먼저 써야 한다.

표 2. UCSR0B 레지스터 비트

(3) UCSR0C

UCSR0C 레지스터는 USART 통신에서의 데이터 형식 및 통신 방법 결정을 위해 사용되었다. UCSR0C 레지스터의 구조는 그림 15와 같다.

비트	7	6	5	4	3	2	1	0
	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	1	1	0

그림 15. UCSR0C 레지스터 구조

비트 6 UMSEL00과 비트 7 UMSEL01은 동기 또는 비동기 통신 모드를 지정하기 위해 사용된다. 비트 설정에 따른 통신 모드는 표 3과 같다.

UMSEL01	UMSEL00	모드
0	0	비동기 USART
0	1	동기 USART
1	0	-
1	1	마스터 SPI 모드

표 3. 통신 모드

비트 4 UPM00과 비트 5 UPM01은 패리티 모드를 설정하는 비트로 홀수 패리티, 짝수 패리티, 패리티 사용 안함 중 선택할 수 있다.

UPM01	UPM00	패리티 비트
0	0	없음
0	1	-
1	0	짝수 패리티
1	1	홀수 패리티

표 4. 패리티 비트 모드

비트 3 USBS0는 정지 비트의 수를 지정하기 위해 사용한다.

USBS0	정지 비트 수
0	1
2	2

표 5. 정지 비트

비트 1 UCSZ00과 비트 2 UCSZ01은 데이터 비트의 수를 결정하기 위해 사용되며, 이 때 UCSR0B 레지스터의 비트 2 UCSZ02가 함께 사용된다.

UCSR0B 레지스터	UCSR0C 레지스터		데이터 비트 수
UCSZ02	UCSZ01	UCSZ00	
0	0	0	5
0	0	1	6
0	1	0	7
0	1	1	8
1	0	0	-
1	0	1	-
1	1	0	-
1	1	1	9

표 6. 데이터 비트

마지막 비트 0 UCPOL0는 클록(XCK)의 극성을 지정하는 비트로 동기 모드에서만 사용이 가능하며 비동기 모드에서는 0의 값을 가져야 한다. 코드 1에서 UCSR0C 레지스터는 0x06의 값으로 설정되어 있으며 그 의미는 그림 16과 같다.

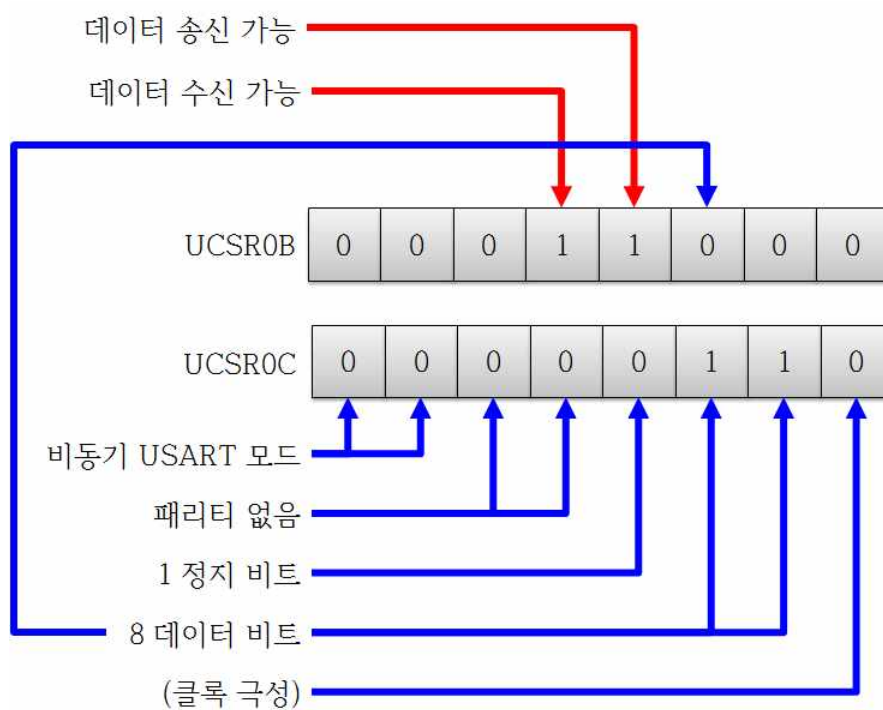


그림 16. UCSR0C 레지스터 설정

(4) UBRR0H, UBRR0L

UBRR0는 USART Baud Rate Register로 2개의 8비트 레지스터를 조합한 16비트 레지스터로 구성되어 있다. UBRR0 레지스터는 상위 4비트를 나타내는 UBRR0H 레지스터와 하위 8비트를 나타내는 UBRR0L 레지스터로 구성되어 보울을 12비트로 표현하며 UBRR0H 레지스터의 4비트는 사용되지 않는다.

비트	7	6	5	4	3	2	1	0
UBRR0H	-	-	-	-	UBRR0H [11:8]			
UBRR0L	UBRR0L [7:0]							
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

그림 17. UBRR0 레지스터 구조

목표로 하는 전송 속도(보울)가 결정되면 이를 UBRR0 값으로 변환하여야 하며 변환식은 전송 모드에 따라 달라진다. 배속은 USCR0A 레지스터의 U2X0 비트에 따라 결정된다. 전송 모드에 따른 UBRR0 레지스터 계산식은 표 7과 같다.

동작 모드	U2X0	보울 계산식	UBRR0 계산식
비동기 1배속	0	$\frac{f_{osc}}{16 \cdot (UBRR0+1)}$	$\frac{f_{osc}}{16 \cdot BAUD} - 1$
비동기 2배속	1	$\frac{f_{osc}}{8 \cdot (UBRR0+1)}$	$\frac{f_{osc}}{8 \cdot BAUD} - 1$
동기	-	$\frac{f_{osc}}{2 \cdot (UBRR0+1)}$	$\frac{f_{osc}}{2 \cdot BAUD} - 1$

표 7. UBRR0 레지스터 값 계산식

표 7에서 BAUD는 전송 속도를 나타내며 f_{osc} 는 클록 주파수로 아두이노 우노의 경우 16Mhz를 사용한다. 표 7의 식을 통해 일반적으로 많이 사용되는 보울에 대응하는 UBRR0 값을 나타낸 것이 표 8이다. 코드 1에서는 9600 보울로 설정하기 위해 207의 값을 사용하고 있다.

BAUD	UBRR0 계산값		UBRR0 사용값	
	1배속(U2X0 = 0)	2배속(U2X0 = 1)	1배속(U2X0 = 0)	2배속(U2X0 = 1)
1200	832.33	1665.67	832	1666
2400	415.67	832.33	416	832
4800	207.33	415.67	207	416
9600	103.17	207.33	103	207
14400	68.44	137.89	68	138
19200	51.08	103.17	51	103
28800	33.72	68.44	34	68
38400	25.04	51.08	25	51
57600	16.36	33.72	16	34
115200	7.68	16.36	8	16

표 8. $f_{osc}=16\text{MHz}$, 비동기 전송 모드에서 보울에 따른 UBRR0 값

표 8에서 알 수 있듯이 표 7의 식으로 계산한 UBRR0 값은 정수값이 아니므로 실제 사용되는 값은 근사화된 정수값을 사용한다. 따라서 약간의 오차는 발생할 수 있다.

(5) UDR0

마지막 레지스터는 송수신된 데이터가 저장되는 버퍼 레지스터인 USART Data Register, UDR0 레지스터이다. UDR0 레지스터는 데이터 전송을 위한 버퍼인 TXB 레지스터와 데이터 수신을 위한 버퍼인 RXB 레지스터로 구성되어 있지만 동일한 입출력 주소를 사용하므로 UDR0 레지스터에 데이터를 쓰면 TXB 레지스터에 쓰여지고, UDR0 레지스터로부터 데이터를 읽으면 RXB 레지스터의 데이터가 읽혀진다. 5, 6, 7 비트의 데이터 비트를 사용하는 경우에는 상위 비트들이 사용되지 않으며 9비트 데이터 비트를 사용하는 경우에는 UCSR0B 레지스터의 RXB80 비트 및 TXB80 비트와 함께 사용된다. UDR0 레지스터의 구조는 그림 18과 같다.

비트	7	6	5	4	3	2	1	0
	RXB [7:0]							
	TXB [7:0]							
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 18. UDR0 레지스터 구조

4. 아두이노의 시리얼 통신

UART 통신을 위해서는 앞 절에서 살펴본 것처럼 5개의 레지스터를 설정하여야 한다. 아두이노에서는 이러한 복잡한 과정을 간단히 하기 위해 UART 시리얼 통신을 위한 Serial 클래스를 제공하고 있다. 코드 2는 코드 1과 동일한 기능을 수행하는 프로그램으로 아두이노의 Serial 클래스를 이용하여 작성한 것이다.

코드 2. 컴퓨터와의 UART 통신 - 아두이노

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

void loop()
{
    while(Serial.available()){    // 수신된 데이터가 있는 경우
        byte data = Serial.read(); // 수신된 데이터 읽기
        Serial.write(data);        // 에코 백
    }
}
```

이것으로 끝이라고? 힘들여 살펴본 C 스타일의 코드에 비해 아두이노에서는 모든 내용을 Serial 클래스로 추상화시켜 놓고 있어 직관적인 사용이 가능하다. 코드를 업로드 하고 아두이노 애드온 전용의 시리얼 모니터를 실행시켜서 코드 1과 동일하게 동작하는지 확인해 보자. 그림 11의 터미널 윈도우를 사용해서도 확인 가능하다.

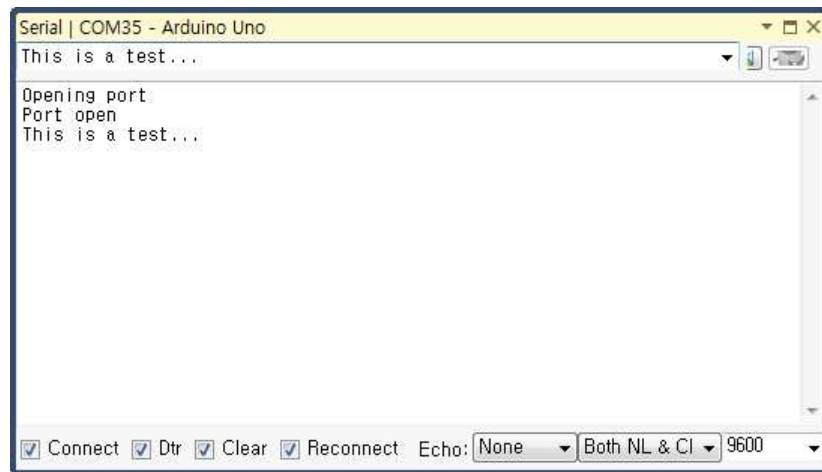


그림 19. 시리얼 모니터를 통한 UART 통신 확인

코드 2에서는 Serial 클래스의 4개 멤버 함수가 사용되었다. 첫 번째는 시리얼 통신 초기화를 위한 begin 함수로 매개변수로는 보율을 설정하고 있다.

```
void begin(unsigned long baud)
void begin(unsigned long baud, byte config)
매개변수
    baud : 속도
    config : 데이터 비트 수, 패리티, 정지 비트 설정
반환값 : 없음
```

C 스타일 코드에서는 데이터 비트 수, 패리티, 정지 비트 등을 설정해 주었지만 아두이노에서는 디폴트값으로 8비트 데이터, 패리티 사용 안함, 1비트 정지 비트를 사용하며 이는 코드 1에서의 설정과 동일하다. 필요한 경우 매개변수 config를 통해 설정할 수 있다. 수신된 데이터를 읽어오는 함수는 read로 바이트 단위의 데이터를 반환한다.

```
int read(void)
매개변수 : 없음
반환값 : 직렬 통신 수신 버퍼의 첫 번째 문자 또는 -1
```

다만, 수신된 데이터가 없는 경우에는 -1의 값을 반환하므로 수신된 데이터가 존재하는지의 여부를 확인한 이후 읽기를 추천한다. 수신된 데이터의 존재 여부는 available 함수를 통해 알아낼 수 있다.

```
int available(void)
```

매개변수 : 없음

반환값 : 직렬 포트 수신 데이터 버퍼에 저장된 데이터의 바이트 수

데이터를 송신하기 위해 사용할 수 있는 함수는 write로 read 함수와 마찬가지로 바이트 단위로 데이터를 전송하지만 오버라이딩을 통해 바이트 배열 역시 전송할 수 있다.

```
size_t write(uint8_t ch)
```

```
size_t write(const char *str)
```

```
size_t write(const uint8_t *buffer, size_t size)
```

매개변수

ch : 출력 문자

str : 출력 문자열

buffer : 출력 데이터열

size : 버퍼의 크기

반환값 : 출력한 바이트 수

Serial 클래스에서 유용한 함수 중에는 print와 println 함수가 있다. write 함수가 바이트 단위의 데이터 전송에 사용된다면 print 함수는 문자 단위로 데이터를 전송한다. Serial.write(78)은 1바이트 데이터 78을 전송하지만 Serial.print(78)은 숫자 78을 문자열 "78"의 2바이트로 변환하여 전송한다. Serial.println(78) 함수는 문자의 끝에 개행 문자를 추가로 전송하는 것을 제외하면 print 함수와 동일하다. 개행 문자는 carriage return (ASCII 13, '\r')과 new line (ASCII 10, '\n')으로 이루어진다.

```
size_t print(value, format)
size_t println(value, format)
매개변수
    value : 출력값 (char, char 배열, String, 정수, 실수 등)
    format : 출력 형식
반환값 : 직렬 포트로 출력된 바이트 수
```

숫자를 출력하는 경우 print 함수는 매개변수 format을 통해 정수의 경우 진법을, 실수의 경우 소수점 이하 자릿수를 지정할 수 있도록 해주므로, 디버깅 기능이 없는 아두이노 IDE에서 유용한 디버깅 도구로 사용할 수 있다. 코드 3은 Serial 클래스를 사용하여 다양한 형식의 데이터를 시리얼 모니터로 출력하는 예를 보여준다.

코드 3. Serial 클래스 테스트

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

void loop()
{
    // 정수
    Serial.println(78, BIN); // "1001110"
    Serial.println(78, OCT); // "116"
    Serial.println(78, DEC); // "78"
    Serial.println(78, HEX); // "4E"

    // 실수
    Serial.println(1.23456, 0); // "1"
    Serial.println(1.23456, 2); // "1.23"
    Serial.println(1.23456, 4); // "1.2346"

    // 문자
    Serial.print('C');
```

```
Serial.print('\n');

// 문자열
Serial.println("Serial Class Test...");

while(true);
}
```

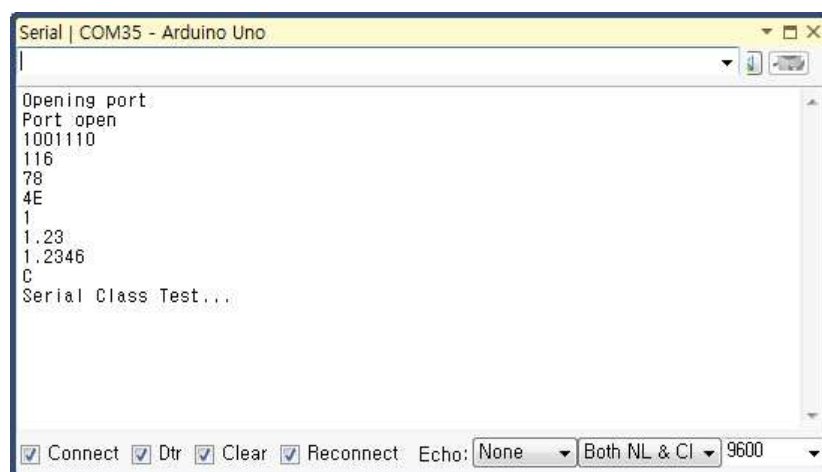


그림 20. 코드 3 실행 결과