

\* 본 자료는 서경대학교 아두이노 프로그래밍 수업 수강자를 위해 작성된 강의 교재입니다.

강의교재- 아두이노 프로그래밍

# 6장 직렬통신

서경대학교 김진현

# 6

---

## 직렬통신

### 내용

- 6.1 직렬통신 개요
- 6.2 단순 문자열 출력
- 6.3 터미널 에뮬레이터
- 6.4 줄바꾸기에 대한 실험
- 6.5 문자 및 이진 데이터의 입출력
- 6.6 직렬 포트 0의 역할
- 6.7 소프트웨어 방식의 직렬통신
- 6.8 비동기 직렬통신의 원리(이론)
- 6.9 고찰

“(보류) 🍷” 표기 부분은 강좌 후반부에 검토할 예정입니다.

## 6.1 직렬통신 개요

직렬(serial)통신은 장치와 장치간의 통신을 위해 데이터를 직렬화하여 교신하는 통신 기법을 말한다. 직렬통신이 성공적으로 이루어지기 위해서는 송신자(transmitter)와 수신자(receiver)간의 데이터의 직렬화 방법, 전송/수신 방법 등이 서로 일치해야 한다. 이러한 규약을 프로토콜(protocol)이라 하는데 비동기 직렬 통신으로 가장 많이 쓰이는 프로토콜은 RS-232C이다.

RS-232는 1960년에 처음 EIA<sup>1)</sup>(Electronics Industries Association)에 의해 제안되어 수정을 거쳐 1965년에 RS-232C로 확립되었다<sup>2)</sup>.

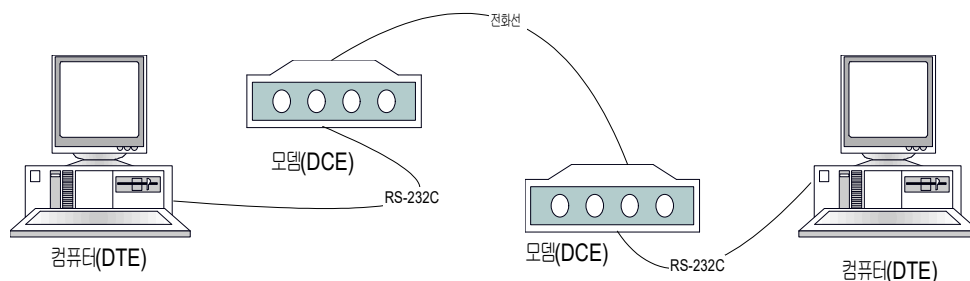


그림 6.1.1 모뎀을 이용한 데이터 통신에 사용되는 RS-232C 통신

RS-232C는 그림 6.1.1에 보인 바와 같이 DTE(Data Terminal Equipment)와 DCE(Data Communication Equipment)간의 통신을 위하여 만들어진 규약이다. DTE는 데이터 단말 장치로 컴퓨터, 단말기 혹은 PC 등 컴퓨터를 말하며, DCE는 데이터 통신 장치로 전화망의 직렬 통신을 위하여 사용되는 모뎀

1) 北美의 전기기기 생산자들의 조합. 이것의 표준에는 보통 RS(Recommended Standard)가 붙는다.

2) 정식 명칭은 RS-232C이지만 보통 RS-232, RS232 등으로 사용되는 것이 관례인 듯하다. 본 자료에서도 이를 구분하지 않고 사용하기로 한다.

(MODEM : **MO**dulation & **DEM**oulation)을 말한다. 즉 RS-232C는 전화망을 이용한 컴퓨터간의 통신을 위해 컴퓨터와 모뎀간의 통신을 규정을 정한 규약이라 할 수 있다. PC에서도 RS-232를 모뎀과 PC와 직렬통신 장치로 사용하여 초기 인터넷 통신을 구현하였다<sup>3)</sup>.

그러나 오늘날 RS-232에 모뎀을 사용하는 경우는 드물고 모뎀 없이 DTE와 DTE 간의 통신에도 많이 사용하고 있다. 즉, 컴퓨터와 컴퓨터 간에 혹은 보드와 보드간의 통신에 주로 많이 활용된다. 특히 일반 사용자보다는 개발자들에게 많이 사용되는데 개발 중인 보드와 PC간의 통신에 주로 많이 사용된다. USB나 이더넷(ethernet) 같은 성능 좋은 통신을 구현하기 전에 먼저 **보드 개발의 초기 단계**에서 GPIO 장치의 동작 검증이 끝나면 간단한 직렬장치인 RS-232를 개통하여 **PC를 통해 보드의 여러 기능을 점검하는 작업**을 시작하는 것이다.

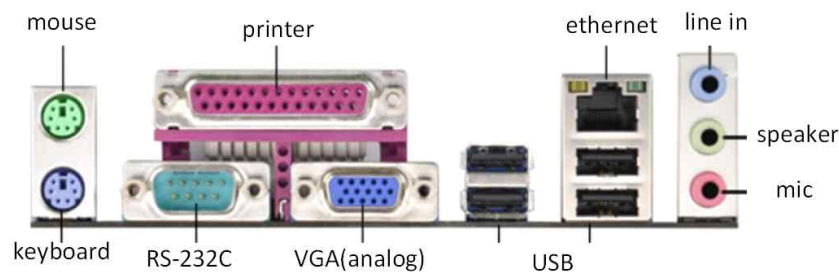


그림 6.1.2 구형 PC 백판넬에 존재했던 RS-232C 포트

구형 PC의 경우 RS-232C 직렬통신 포트<sup>4)</sup>는 그림 6.1.2와 같이 PC의 뒷면 판넬부에 배치되어 있었다. 그러나 USB가 널리 확산되기 시작하면서 직렬포트 콘넥터는 PC 백 판넬에서 사라지게 되었다.

그림 6.1.3(a)에 전형적인 직렬통신 케이블을 보였다. 그러나 COM 포트 콘넥터가 없는 요즘의 PC에서는 (b) 혹은 (c)와 같은 USB to RS-232 변환 케이블/dongle이 RS-232 장치 통신에 활용된다<sup>5)</sup>.

3) 100% RS-232C 호환은 아니고 RS-232C 중 중요한 규격만 채택하였다.

4) port. 입출력 연결 장치를 뜻한다. 이하 COM 포트라고 칭하기로 한다.

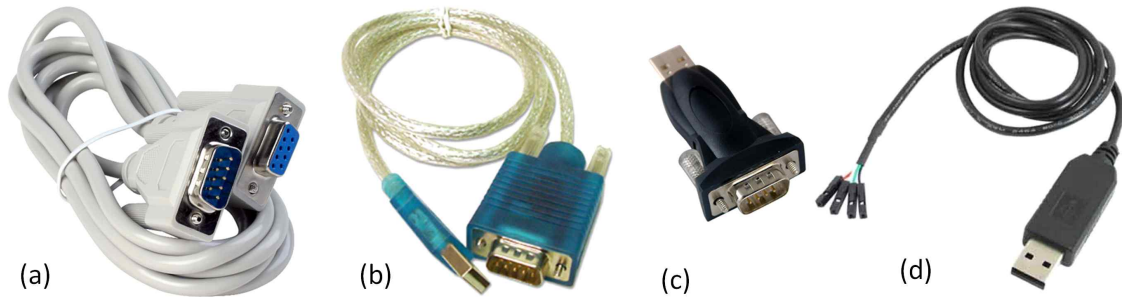


그림 6.1.3 (a) RS-232 케이블, (b) USB to RS232 케이블( $\pm 12V$ ), (c) USB to RS-232C dongle. (d) USB to RS232 TTL( $0V/5V$ ) 케이블

(a)~(c)의 케이블/동글들은 PC와 9핀 RS-232 포트를 가진 장치와의 연결에 사용된다. 하지만 RS-232를 보드 $\leftrightarrow$ PC 연결에 사용하는 경우라면 구태여 불필요한 9핀 포트를 사용할 필요도 없고, 원거리 통신이나 필요한 고전압 사양에 맞추기 위해 전압 변환기를 갖춰야 할 필요가 없게 되었다<sup>6)</sup>. 그림 (d)는 이러한 경향에 맞추어 TTL 레벨, 즉  $0V/5V$ 를 사용하는 USB to RS-232 케이블이다. 이런 TTL 레벨의 RS-232는 일반적 규격의 RS-232 장치에 연결하면 손상을 입게 되기 때문에 사용에 신중을 기해야 한다.

최근에는 다른 장치들도 RS-232 대신 USB 통신을 채택하는 경향이 늘어가는 추세이다. 이 경우에는 PC $\leftrightarrow$ 장치 모두 USB가 사용되기도 한다. 이러한 사례로서 우리가 이미 활용하고 있는 아두이노 통신을 들 수 있다. PC $\leftrightarrow$ 아두이노간의 통신을 USB 선로를 이용한다. 이 경우 USB 장치 인식/설정과 제어는 USB 프로토콜에 따르지만 Serial.print와 같은 함수는 USB가 RS-232 통신을 흉내내는 방식으로 운영된다. 이 경우 PC $\leftrightarrow$ 아두이노간의 통신 RS232 프로토콜을 따른다. 다시 말하면 물리적으로는 USB, 프로토콜(프로그래밍) 관점

5) 이들은 한쪽은 USB이지만 최종단에서 RS-232로 다른 장치에 연결할 수 있는 회로를 내장하고 있다. 이 USB 장치는 운영체제에서 가상의 COM 포트를 배정받아 RS-232를 흉내(emulation) 내는 방식으로 동작한다.

6) RS-232 프로토콜은 송신단에서는  $\pm 5V \sim \pm 15V$  범위를 신호를 송출할 수 있어야 하고, 수신단에서는  $\pm 5V \sim \pm 25V$ 의 범위의 전압을 받아들일 수 있어야 한다. PC의 RS-232는 보통  $\pm 12V$ 를 지원한다.

으로는 RS-232가 사용된다는 것이다.

## □ 아두이노의 직렬통신 포트

Ardu-Ez 실험 키트에는 총 4개의 직렬포트가 있으며 사용되는 단자와 Ardu-Ez 키트 상에서의 용도는 다음과 같다.

포트 번호	수신(RX) 단자	송신(TX) 단자	용도
0	0번(RxD0)	1번(TxD0)	보드 내 다른 USB 제어기 <sup>7)</sup> 의 UART에 연결되어 있다. USB 케이블을 통한 PC 본체와의 직렬 통신을 담당한다. Sketch에서 작성한 프로그램도 이 채널을 통해 업로드된다.
1	19번(RxD1)	18번(TxD1)	- RFID 모듈과 통신용. 키트 내부에서 사용
2	17번(RxD2)	16번(TxD2)	
3	15번(RxD3)	14번(TxD3)	- Bluetooth 모듈과 통신용. 키트 내부에서 사용

이 중 0번 포트는 USB를 통한 PC와의 RS-232 통신에 활용된다. 따라서 이 포트는 다른 장치와의 통신에 활용할 수 없다. PC와의 교신을 위해서는 Serial.print와 같이 번호를 붙이지 않지만, 다른 포트와의 교신을 위해서는 Serial1.print, Serial2.print와 같이 해당 포트 번호를 붙이면 된다.

Uno 모델의 경우에는 아래와 같이 0번 포트만 지원된다. Uno 모델의 프로세서가 UART가 1개 밖에 없으므로 추가 직렬포트를 사용하고자 한다면 6.7절을 참고하기 바란다.

포트 번호	수신(RX) 단자	송신(TX) 단자	용도
0	0번(RX0)	1번(TX0)	메가 2560 모델과 같다.

7) USB와의 통신을 담당하는 별도의 USB 제어기 Atmega 16U2와 직렬통신으로 연결되어 있다. Atmega 2560 메인 프로세서는 이를 통해 본체 PC와 통신한다.

## 6.2 단순 문자열 출력

아두이노 직렬통신 함수는 20여 가지가 된다. 본 절에서는 가장 기본적인 문자열 출력을 기본적인 표준함수로 구현해 분석해 보기로 한다.

□ 예제 6.2.1 : 시리얼 모니터를 통해 스트링 객체 및 문자를 출력한다. Serial에 번호가 없으면 0번 포트, 즉 PC와 통신하는 함수이다.

serial1.ino : 사용된 표준 클래스 및 함수 -

Serial.begin(), Serial.print(), Serial.println()

```
01 // Hello world 메시지 스트링 출력
02 void setup() { Serial.begin(9600); } // 전송속도 설정. 19200으로 바꾸어 보시오.
03 String msgHello = "Hello world!"; // string object
04 char string[] = "I am Arduino."; // string array
05 void loop(){
06     Serial.println(msgHello);
07     delay(1000);
08     Serial.print(string); Serial.print("\n");
09     delay(1000);
10 }
```

시리얼 모니터의 전송 속도는 기본으로 9600bps로 설정되어 있다. 메인 프로그램에서 통신 속도를 변경했을 때는 시리얼 모니터에서도 바꾸어야 한다. 이는 그림 6.2.1과 같이 시리얼 모니터의 보드 레이트 설정 메뉴에서 선택할 수 있다.

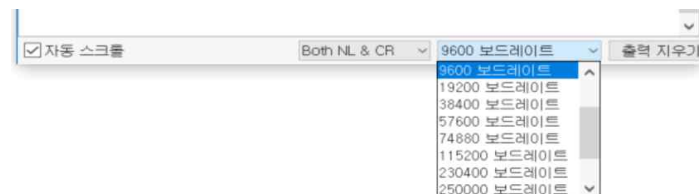


그림 6.2.1 시리얼 모니터의 전송속도 설정 메뉴

Serial.begin() 함수의 용법은 아래와 같다. begin() 함수를 이용해 선언한

전송 속도 및 데이터 프로토콜은 상대측인 시리얼 모니터에 설정한 값과 일치해야 한다. 시리얼 모니터에서는 전송 속도만 변경할 수 있기 때문에 프로토콜은 SERIAL\_8N1 외에는 사용할 수 없다.

<b>void Serial.begin(speed) // 초기화. 전송속도를 지정.</b> <b>void Serial.begin(speed, config) // 전송속도 및 데이터 프로토콜 초기화.</b>		
기능	<p>전송속도와 기타 데이터 프로토콜을 지정하여 직렬통신장치를 초기화한다. 통신을 시작하기 전에 초기화해야 한다.</p> <p><b>Arduino Mega only: 직렬통신장치를 3개 더 보유하고 있다. 그러나 이 장치는 보드 상의 콘넥터에 제공된다. TTL 레벨이므로 주의를 요한다.</b></p> <p>Serial1.begin(speed), Serial1.begin(speed, config)          Serial2.begin(speed), Serial2.begin(speed, config)          Serial3.begin(speed), Serial3.begin(speed, config)</p>	
매개변수	speed	<p>Bit Per second : <del>300</del>, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200</p> <p><b>실험에 의하면 300은 실패. 115200은 성공함.</b></p>
	config	<p>데이터 프로토콜 설정 : 데이터 비트 수(5/6/7/8), 패리티 여부 (None/Even/Odd), 정지 비트의 길이(1/2)를 지정한다.</p> <p><b>SERIAL_8N1 : 8 data bits, no parity, one stop bit.. default.</b>  <b>SERIAL_7E2 : 7bit. even parity, 2 stop bits</b></p> <p>-SERIAL_5N1, -SERIAL_6N1, -SERIAL_7N1, -SERIAL_8N1          -SERIAL_5N2, -SERIAL_6N2, -SERIAL_7N2, -SERIAL_8N2          -SERIAL_5E1, -SERIAL_6E1, -SERIAL_7E1, -SERIAL_8E1          -SERIAL_5E2, -SERIAL_6E2, -SERIAL_7E2, -SERIAL_8E2          -SERIAL_5O1, -SERIAL_6O1, -SERIAL_7O1, -SERIAL_8O1          -SERIAL_5O2, -SERIAL_6O2, -SERIAL_7O2, -SERIAL_8O2</p>



문자를 출력하는 함수는 다음의 2종류가 있다. print 함수와는 달리 println 함수는 문자를 출력하고 난 후 줄을 바꾸라는 LF(Line Feed) 코드를 추가로 보낸다. 이는 C 언어의 "Wn"과 같은 동작을 수행한다.

<b>long Serial.print(val), long Serial.print(val, format) // 줄 바꾸기 지원없음</b> <b>long Serial.println(val), long Serial.println(val, format) // 줄 바꾸기 수행</b>		
기능	데이터를 ASCII 형태로 만들어 직렬포트로 보낸다. 스트링이나 문자 타입은 있는 그대로 내 보낸다. 함수 이름의 끝에 ln(line)이 있으면 수행 후 줄 바꾸기를 수행한다.	
매개변수	val	출력할 데이터. 어떤 타입이든 가능하다. -Serial.print(78) gives "78" -Serial.print(1.23456) gives "1.23" -Serial.print('N') gives "N" -Serial.print("Hello!") gives "Hello!"
	format	아래 사례 참조. -Serial.print(78, BIN) gives "1001110" -Serial.print(78, OCT) gives "116" -Serial.print(78, DEC) gives "78" -Serial.print(78, HEX) gives "4E" -Serial.println(1.23456, 0) gives "1" -Serial.println(1.23456, 2) gives "1.23" -Serial.println(1.23456, 4) gives "1.2346"
리턴 값	long	출력된 문자의 수를 반환한다.

## 6.3 터미널 에뮬레이터

아두이노가 RS-232를 통해 전송한 데이터는 시리얼 모니터에 출력된다. 스케치에서 제공하는 시리얼 모니터는 RS-232를 통해 아두이노와 PC가 데이터를 교환하는 역할을 담당한다.

본 절에서는 아두이노의 시리얼 모니터와 같은 역할을 수행하는 터미널 에뮬레이터(**terminal emulator**) 프로그램을 소개하고자 한다. 사실 시리얼 모니터는 터미널 에뮬레이터 프로그램의 축소 프로그램이라 할 수 있다. 여기서 에뮬레이터는 흉내낸다는 뜻이니까 이 프로그램을 터미널을 흉내 내는 프로그램이라고 말할 수 있다.

**터미널**은 대형 컴퓨터와 접속하기 위한 사용자를 위한 단말장치로 모니터와 키보드 입력 장치를 갖추고 있다. 그림 6.3.1에 터미널 장치의 사실상 표준으로 오랫동안 사용되었던 1978년 DEC사의 VT-100 터미널을 보였다. 이 장치는 RS-232를 통해 중앙 컴퓨터에 연결되어 다양한 사용자의 입력을 메인 컴퓨터에 전달하고 중앙 컴퓨터가 보내온 정보에 화면에 출력하는 기능을 담당하였다.

그러나 시대가 지나면서 PC가 보편화되기 시작하면서 PC의 프로그램으로 터미널과 똑 같은 역할을 수행하게 되었는데 이 프로그램을 통칭하여 터미널 에뮬레이터라고 한다.



그림 6.3.1 터미널 VT-100

터미널 에뮬레이터에도 공개 소스(open source) 바람이 불기 시작하여 Tera Term(테라텀)<sup>8)</sup>과 PuTTY(푸티)<sup>9)</sup> 등과 같이 공개된 프로그램이 널리 사용되고 있다.

예제 6.2.1의 프로그램을 수행한 결과를 그림 6.3.2에 보였다.

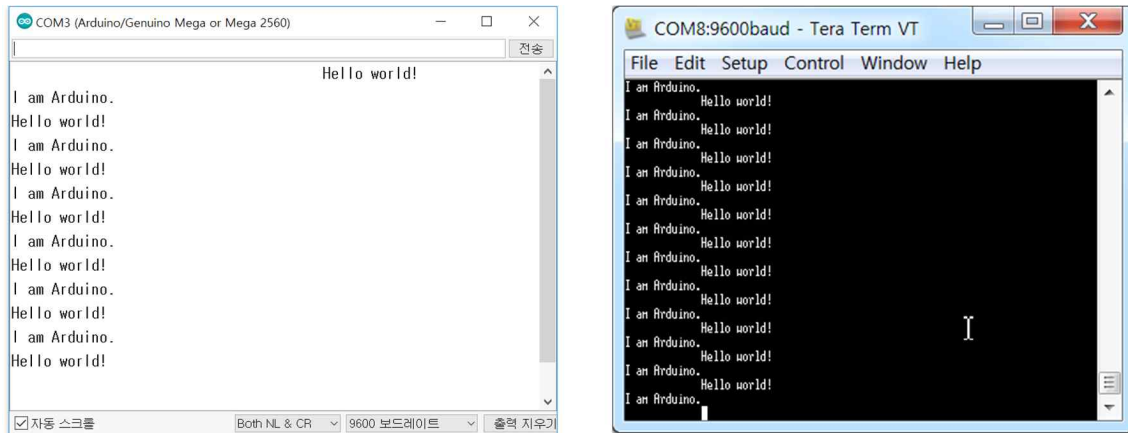


그림 6.3.2 (a) 시리얼 모니터 수행 결과, (b) 테라텀 수행 결과

Tera Term을 시리얼 모니터처럼 동작시키기 위해서는 그림 6.3.3 ~ 그림 6.3.5에 보인 바와 같이 COM 포트 번호, 프로토콜, 터미널 설정 등의 작업이 사전에 필요하다.

8) <https://ttssh2.osdn.jp/>

9) <http://www.putty.org/>

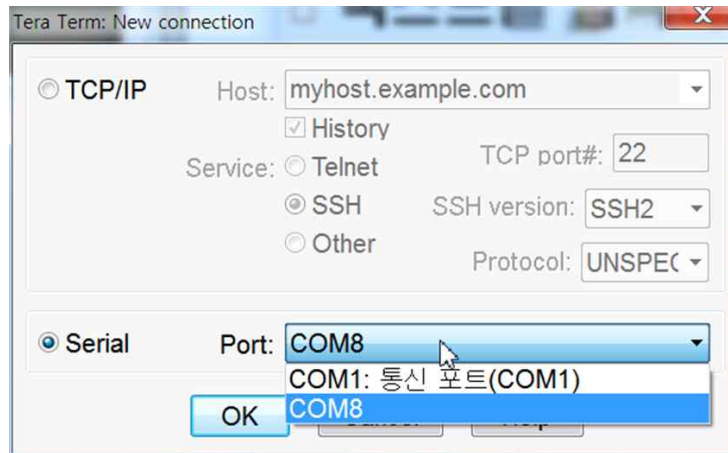


그림 6.3.3 테라텀 진입시 나타나는 COM 포트 설정

COM 포트 설정을 포함한 RS-232 프로토콜 설정 변경은 그림 6.3.4(a)의 [Setup] -> [serial port] 메뉴를 거쳐 (b)와 같이 변경 여부를 결정할 수 있다.

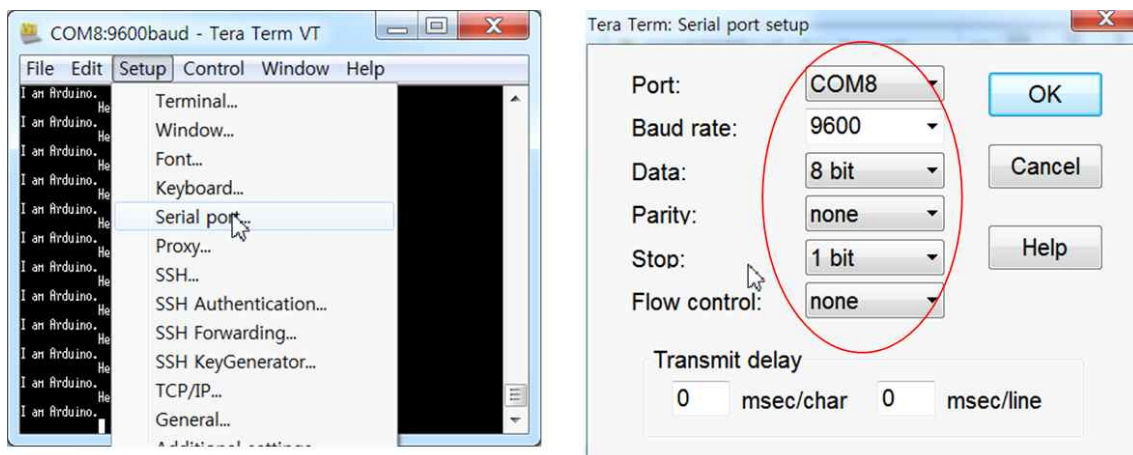


그림 6.3.4 (a) Setup 메뉴, (b) RS-232 프로토콜 설정

그림 6.3.2에 "Hello World!" 메시지가 맨 앞 칸에 나타나지 못한 이유는 터미널 설정이 new line을 받아들일 때 CR(Carrage Return)으로 인식하게끔 설정되어 있기 때문이다. 이는 [Setup] -> [terminal] 메뉴를 거쳐 그림

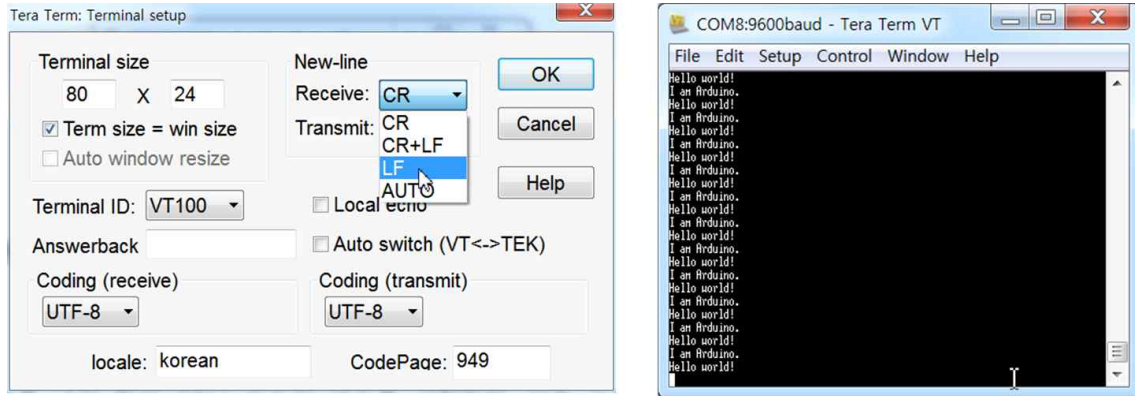


그림 6.3.5 (a) terminal setup 메뉴, (b) 올바른 출력 결과

6.3.5(a)와 같이 Receive 설정을 LF로 바꿈으로써 교정할 수 있다. 이와 같은 설정 변경으로 그림 6.3.5(b)에서는 올바르게 출력된 메시지를 확인할 수 있다.

Tera Term을 수행할 때마다 매번 설정을 바꾸는 불편을 방지하기 위해 설정 결과를 파일로 저장할 수 있다. 특히 프로그램의 설치된 폴더에 TERATERM.INI 파일은 디폴트 셋업 파일이기 때문에 이름으로 저장하면 매번 셋업 파일을 번거롭게 로드할 필요가 없다.

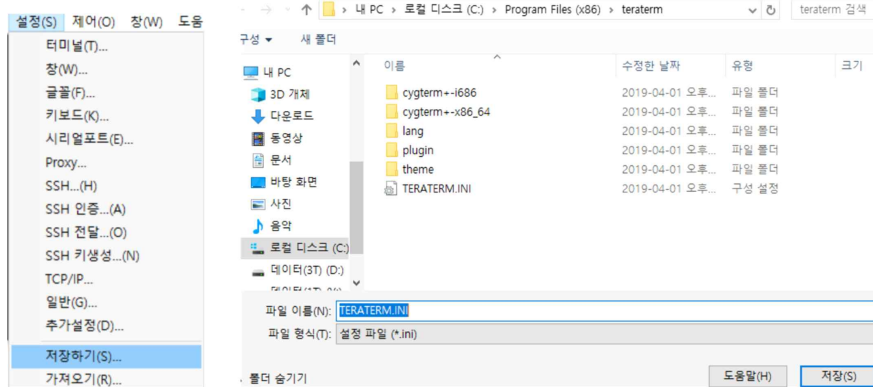


그림 6.3.6 설정 상태의 파일 저장

### 주의사항 !

시리얼 모니터와는 달리 아두이노 IDE 외 별도로 설치된 터미널 에뮬레이터는 IDE 의 프로그램 업로드를 방해한다. 이는 프로그램 업로드 과정에 COM 장치를 사용하는데 이것이 외부 프로그램인 터미널 에뮬레이터에 점유되어 있기 때문이다. 따라서 프로그램 업로드를 위해서는 잠시 에뮬레이터 프로그램의 수행을 종료해야 한다.

시리얼 모니터의 경우에는 내부에서 점유 문제를 조정하기 때문에 이러한 문제가 발생하지 않는다.

## 6.4 줄 바꾸기에 대한 실험

시리얼 모니터는 다음 줄 바꾸기에 관련된 다음 제어 문자<sup>10)</sup>에 대해 올바르게 출력하지 않고 있다.

기호 및 표현	기능	문자값
CR (Carriage Return)	커저를 그 줄의 맨 앞에 위치하게 한다. C 언어에서는 'wr'을 출력하여 실현한다.	0x0d
LF (Line Feed)	커저를 현 위치의 아래로 내린다. C 언어에서는 'wn'으로 구현할 수 있는데 시스템에 따라 추가로 'r'의 동작을 수행하여 커저가 맨 앞으로 가기도 한다.	0x0a

Tera Term은 이러한 제어 문자에 대해 수신 혹은 송신 동작에 대해 그림 6.3.5(a)에 보인 4종의 설정에 따라 다음과 같이 반응한다.

---

10) ASCII 체계에서는 0x00~0x1F의 코드는 문자가 아니라 터미널 및 프린터 장치 등을 제어하는 제어 코드로 쓰인다. 0x0d, 0x0a는 커저의 위치를 제어한다.

코드	receive 동작일 때의 해석: 아래 4종의 설정에 따라 통신 포트를 통해 수신한 제어 문자를 Tera Term 터미널 에뮬레이터 화면에 어떻게 나타나게 할지를 설명한 것이다.	
CR	Tera Term doesn't convert it.	
CR+LF	Received CR (\$0D) character is converted to CR+LF (\$0D \$0A) pairs. ⇒ '₩n' 문자 코드만을 수신해도 화면에서는 줄 바꾸기를 행하여 보여준다. e.g.) If "CR LF" is received, it is converted to "CR LF LF." ⇒ 커서가 맨 앞으로 가고 2줄 아래로 내려 간다.	CR, LF 어느 것 중 하나만 수신해도 화면에는 CR과 LF 2개의 제어를 행한다. 즉, 줄 바꾸기 동작을 행한다.
LF	Received LF (\$0A) character is converted to CR+LF (\$0D \$0A) pairs. ⇒ '₩n' 문자 코드만 수신해도 화면에서는 줄 바꾸기를 행하여 보여준다. 이 설정은 일반 C언어를 PC 상에서 수행할 때 같은 동작을 수행과 같게 만든다. e.g.) If "CR LF" is received, it is converted to "CR CR LF." ⇒ 커서가 맨 앞으로 2회 가지만 실제로는 한 번 간 것과 같다. 그리고 줄 바꾸기가 일어난다. 따라서 다음 줄 맨 앞으로 간 것과 같은 동작이 일어난다.	
AUTO	Received CR (\$0D) or LF (\$0A) character is converted to CR+LF (\$0D \$0A) pairs. e.g.) If "CR+LF" are received, it is not converted. If "CR" is received, it is converted to "CR LF." If "LF" is received, it is converted to "CR LF."	

코드	transmit 동작일 때의 해석: 아래 4종의 설정에 따라 Tera Term이 출력할 제어 문자를 상대방에게 어떻게 보낼지를 설명한 것이다.	
CR	Tera Term doesn't convert it.	
CR+LF	CR (\$0D) character to be sent is converted to CR+LF (\$0D \$0A) pairs before it is actually sent.	
LF	CR (\$0D) character to be sent is converted to LF (\$0A) before it is actually sent.	

예제 6.4.1은 그러한 사례를 시험해 본 것이다.



□ 예제 6.4.1 : 다음 실험을 수행하면서 'Wr' 코드 출력의 의미를 분석해 보자

serial2.ino : 사용된 표준 클래스 및 함수 -

Serial.begin(), Serial.print(), Serial.println()

```

01 void setup(){ Serial.begin(9600); } // 전송속도 설정.
02 int seconds =0;
03 void loop(){
04     Serial.print("      Wr"); // 'Wr'은 CR 코드를 시리얼 모니터 혹은 테라텀으로 보낸다.
05     Serial.print(seconds++);
06     delay(1000);
07 }

```

그림 6.4.1은 예제 2의 수행 결과를 시리얼 모니터로 출력받은 결과이다. 'Wr'은 제어문자로서 문자 출력은 터미널측에 이것은 'Wr'을 처리하는 옵션이 없기 때문에 그냥 해당 문자를 무시하고 있음을 알 수 있다.

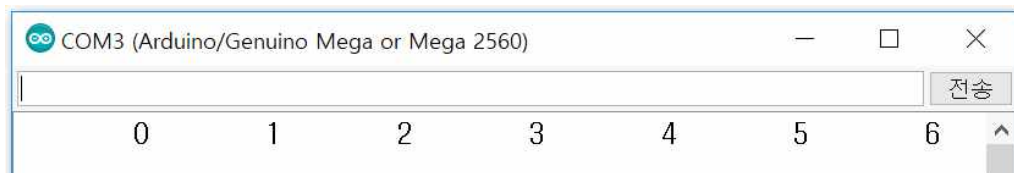
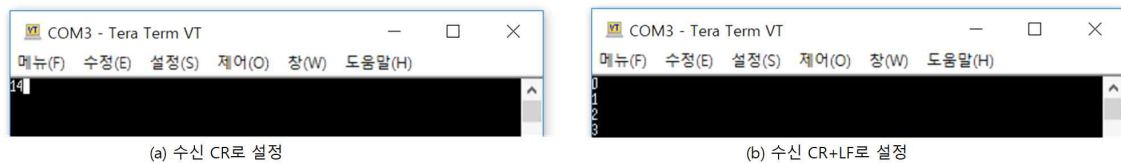


그림 6.4.1 'Wr' 코드를 처리하지 못한다.

그림 6.4.2는 줄 바꿈 설정을 바꾸어 본 Tera Term 출력 결과이다.



(a) 수신 CR로 설정

(b) 수신 CR+LF로 설정

그림 6.4.2 터미널 설정=>줄 바꿈 설정을 CR 혹은 CR+LF로 설정하였을 경우

그림 (a)에 보인 바와 같이 CR 설정은 커서를 맨 앞으로 보내기 때문에 수자가 덮혀 쓰이는 반면, (b)의 설정은 CR이 들어오면 Tera Term이 이를 CR과 LF로 확장하기 때문에 줄 바꿈이 일어나는 현상을 볼 수 있다.

## 6.5 문자 및 이진 데이터의 입출력

터미널 에뮬레이터를 사용하면 전송속도, 데이터 프로토콜의 설정 변경 등이 자유로울 뿐만 아니라 PC에서 사용자가 입력한 문자 정보를 좀 더 다양하고 편리한 방법으로 입력할 수 있다. 일례를 들면 시리얼 모니터에서는 enter키를 입력해야 정보가 전달되었으나 에뮬레이터에서는 마치 PC를 사용하는 것처럼 키보드의 자판을 입력하자마자 그 정보가 바로 아두이노로 전달되게 할 수 있다. 또한 입력창과 출력창을 한 개의 창으로 사용할 수 있다.

□ 예제 6.5.1 : 터미널에서 입력한 문자를 여러 가지 방법으로 터미널에 출력해 본다. write()는 이진수값을 출력한다.

**serial3.ino** : 아두이노 표준 클래스 및 함수 – Serial.begin(), Serial.available(), Serial.read(), Serial.write(), Serial.print(), Serial.println()

```
01 // read from COM port and show what was read in various ways.
02 void setup() { Serial.begin(9600); }
03 void loop() {
04     if (Serial.available() > 0) {
05         int a = Serial.read(); // for incoming serial data
06         // show what you got:
07         Serial.print("I received a character : ");
08         Serial.write(a);
09         Serial.print(", whose ASCII code is ");
10         Serial.print(a, HEX);
11         Serial.print(". - FYI, DEC = ");
11         Serial.println(a);
12     }
13 }
```

예제 5.5.1을 수행한 결과를 그림 6.5.1<sup>11)</sup>에 보였다. 본 사례에서는 문자 'a' -> 'A' -> '0'를 입력한 후 마지막에는 enter 키를 입력한 것이다. available() 함수는 COM 포트에 데이터가 들어온 개수를 반환한다. read() 함수는 COM 포트로부터 데이터를 읽어오는 동작을 수행하며 write() 함수는 데이터를 가공없이 이진수로 COM 포트에 전송한다. 이후 print() 함수는 원하는 형식으로 문자를 COM 포트에 송신하는 역할을 담당한다.

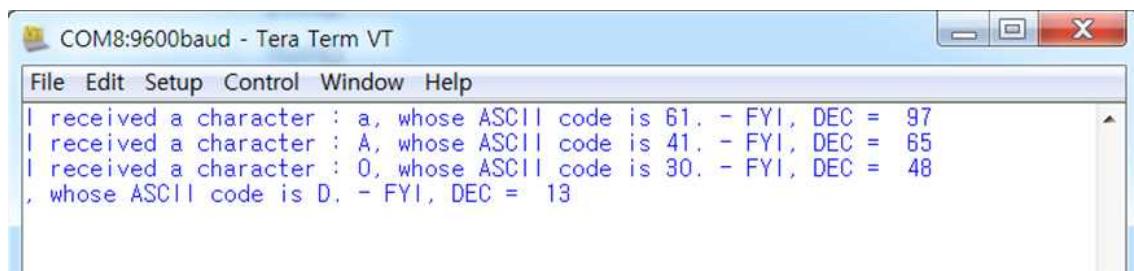


그림 6.5.1 예제 2의 수행 결과(a/A/0/enter key 입력)

11) 터미널의 글자색과 폰트를 바꾸어 표현하였다. Setup의 Window, Font 옵션을 참고하기 바란다.

예제 6.5.2는 사용자가 터미널에서 입력한 문자를 아두이노가 받아 다시 PC로 반향(echo back)시키는 예제를 작성한 것이다.

■ 예제 6.5.2 : 터미널에서 입력한 문자를 터미널로 반향한다.	
serial4.ino : 아두이노 표준 클래스 및 함수 - Serial.begin(), Serial.read(), Serial.write()	
01	// read from COM port and echo back to it.
02	void setup(){
03	Serial.begin(115200, SERIAL_7E2); // 7bit. even parity, 2 stop bits
04	//Serial.begin(600, SERIAL_7O1); // 7bit. odd parity, 1 stop bits
05	}
06	void loop() {
07	if (Serial.available() > 0) { // check if data is available from UART
08	char a = Serial.read(); // read the incoming byte:
09	Serial.write(a);
10	}
11	}

위의 예제에서 사용한 함수에 대해 간단히 요약하면 다음과 같다.

int available(void)		
기    능	수신 버퍼에 저장된 데이터의 개수를 바이트 단위로 반환	
매개변수	void	없음
반환 값	int	수신된 데이터 바이트 개수. 수신버퍼를 읽어내는 read() 등의 함수가 수행되면 바이트의 수가 그 읽어낸 수만큼 줄어든다. 수신한 데이터가 없으면 -1을 반환한다.→ 오류. 취소. →수신한 데이터가 없으면 0을 반환한다. 실험에 의하면 0을 반환했는데도 계속 read()를 수행하면 available() 함수는 계속 0을 반환한다.

int read(void)		
기능	수신된 데이터 중 첫 번째 바이트의 데이터를 읽어옴. 읽기 동작이 수행되면 데이터 버퍼에서 삭제되기 때문에 모두 읽어낸 후 available() 함수를 사용하면 수신된 데이터가 없는 것으로 나타난다.	
매개변수	void	없음
리턴 값	int	수신된 데이터가 수신 버퍼에 있으면 그 중 맨처음 수신한 첫 번째 바이트를 반환한다 <sup>12)</sup> . 버퍼에 데이터가 없는데도 읽어내면 -1을 반환한다 <sup>13)</sup> .

byte write(string or value or array, length)		
기능	바이너리 형태의 데이터로 전송	
매개변수	string or value or array	전송 할 문자열, 데이터 값, 배열 버퍼
	length	배열 데이터 전송 시 배열 길이
리턴 값	byte	전송한 바이트 수

12) Serial.available() 함수의 반환 값이 0보다 크면 그 크기 만큼의 데이터를 수신 버퍼에 받았다는 의미이다.

13) 원래 UART 자체는 수신 데이터가 없는데도 계속 수신버퍼를 읽어내면 버퍼에 남아있던 동일한 데이터를 반환한다. 이는 일반 데이터와 구분하기 어려울 수도 있다. 이 때문에 **아두이노에서는 수신버퍼에 데이터가 없는데도 수신 버퍼를 읽어내면 -1을 반환한다.** 아두이노는 read 함수 내부에서 available() 함수를 호출하여 수신 데이터가 없는데 읽기 동작을 수행하는 것으로 판단되면 -1을 반환하는 것으로 설계되었을 것으로 추정된다.

그림 6.5.2(좌)는 예제 6.5.3(a, b, c)의 프로그램을 수행한 결과이다. 이 프로그램은 터미널에서 입력한 십진수를 출력하고 그 숫자를 Hexa와 Binary로 바꾸어 출력하는 프로그램이다.

이 프로그램은 실행하기 전에 그림 6.5.2(우)와 같이 설정해야 한다<sup>14)</sup>.

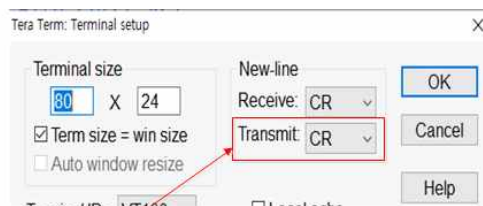
테라텀의 새 줄 보내기가 다음과 같이 CR로 설정되어야 한다.

설정방법: Setup-> Terminal->New-line: Transmit=CR

```

COM3 - Tera Term VT
File Edit Setup Control Window Help
type numerical value with enter key
Enter a decimal value =10
buf[]=10
your_inputs=10
your_inputs in hexa=A
your_inputs in binary=1010
Enter a decimal value =255
buf[]=255
your_inputs=255
your_inputs in hexa=FF
your_inputs in binary=11111111
Enter a decimal value =
  
```

예제 5의 수행결과



설정방법: Setup-> Terminal->New-line: Transmit=CR  
이 설정으로 테라텀에서 enter를 입력하면 아두이노에게는 CR(Carriage Return, ' \r ' ) 코드가 전송된다.

테라텀 사전 설정

그림 6.5.2 예제 5의 수행 결과(좌)와 테라텀의 사전 설정(우)

예제 6.5.3의 프로그램 수행과정을 요약하면 다음과 같다.

- 1) 터미널에서 십진수 문자 데이터를 반복하여 받아들인다.  
하나씩 받아들인 문자들은 스트링을 저장한 어레이 변수, buf[]에 저장하고 String.concat() 함수를 이용하여 다른 스트링 변수, your\_inputs에 문자를 추가한다.
- 2) 문자를 받아드릴 때 마다 화면에 입력한 결과가 나타난다.
- 3) enter를 입력하면 문자입력을 종료한다. buf[]의 맨 마지막에는 스트링의 끝임을 알리는 NULL 문자를 추가한다.
- 4) 이후 스트링 변수, your\_inputs를 정수로 변환하는 String.toInt() 메소드를 이용하여 정수로 변환한다.
- 3) 그 정수값을 16진수와 이진수로 화면에 출력한다.

14) 보통이 이것(CR)이 기본설정임.

□ 예제 6.5.3a 터미널에서 입력한 십진수의 데이터를 스트링으로 출력하고, 또한 Hexa와 Binary로 바꾸어 출력하는 프로그램 - 초기화 부분

serial5.ino : 아두이노 표준 클래스 및 함수 - Serial.begin(), Serial.available(), Serial.read(), String.concat(), String.toInt(), Serial.print(), Serial.println()

```
01  String your_inputs;
02
03  char buf[80];      // 입력한 문자가 차례로 저장될 공간.
04  char a;            // 사용자가 입력한 문자 데이터 저장 변수
05  int i=0;
06  void setup() {
07      Serial.begin(9600);
08      delay(50);
09      Serial.println("type numerical value with enter key");
10      // 혹시 남아있을 수 있는 수신 버퍼의 쓰레기 데이터를 읽어서 버린다.
11      for(i=0; i<16; i++) {
11          Serial.read();      // flush read buffer
12      }
13  }
```

String your\_inputs; 사용자의 입력이 스트링으로 기록될 변수.

String으로 정의한 문자열을 정의하면 변수는 각종 method를 사용할 수 있다<sup>15)</sup>. 여기서는 하나씩 입력받은 문자들을 문자열로 바꾸거나(concat) 문자열을 정수로 바꾸는데(toInt) 활용하였다.

15) 참고: <https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/>

□ 예제 6.5.3b 터미널에서 입력한 십진수의 데이터를 스트링으로 출력하고, 또한 Hexa와 Binary로 바꾸어 출력하는 프로그램 - 함수 정의 부분

serial5.ino : 아두이노 표준 클래스 및 함수 – Serial.begin(), Serial.available(), Serial.read(), String.concat(), String.toInt(), Serial.print(), Serial.println()

```

01 //-----
02 // _getch() : <conio.h>에서 지원하는 키보드 입력 표준함수.
03 // 표준C/C++에서 지원하는 getchar()과 유사하게 한 글자만 입력받는다.
04 // 그러나 getchar()와는 달리 enter키 입력 없이 키를 입력받는다.
05 // 입력 문자를 반향하지 않는다. 즉, 터미널에서 입력한 문자는 화면에 출력되지 않는다.
06 // 함수 내에서 키를 입력하지 않으면 종료되지 않는다.
07 //-----
08 char _getch(void) {      // 테라텀 화면에서 입력받은 문자를 보이지 않는다.
09     while( Serial.available() <= 0 );      // 버퍼에 들어온 것이 없으면 빠져나가지 않는다.
10     return(Serial.read());
11 }
11
12 //-----
13 // _getche() : <conio.h>에서 지원하는 키보드 입력 표준함수.
14 // _getch()와는 달리 입력 문자를 반향(echo)한다. 즉, 터미널에서 입력한 문자는 화면에 출력된다.
15 // 나머지는 _getch()와 같음.
16 //-----
17 char _getche(void) { // with echo. 테라텀 화면에서 입력 받은 문자를 반향하여 화면에 나타나게 한다.
18     while( Serial.available() <= 0 );      // 버퍼에 들어온 것이 없으면 빠져나가지 않는다.
20     char a=Serial.read();
21     Serial.print(a);      // 입력받는 문자를 반향한다.
22     return(a);
23 }

```



□ 예제 6.5.3c 터미널에서 입력한 십진수의 데이터를 스트링으로 출력하고, 또한 Hexa와 Binary로 바꾸어 출력하는 프로그램 - 메인 루프 부분

serial5.ino : 아두이노 표준 클래스 및 함수 - Serial.begin(), Serial.available(), Serial.read(), String.concat(), String.toInt(), Serial.print(), Serial.println()

```

01 void loop() {
02     i=0;
03     your_inputs = "";
04     Serial.print("\nEnter a decimal value =");
05     while ( (a=_getche()) != '\r') { // 테라팀의 새 줄 보내기가 CR로 설정되었음을 가정한다.
06         buf[i++] = a;
07         your_inputs.concat(a);      // 문자를 스트링에 추가한다.
08     }
09     buf[i] = '\0';    // 마지막에 NULL character를 추가하여 스트링을 완성한다.
10     Serial.print("\nbuf[]=");
11     Serial.println(buf);          // buf[] 스트링을 출력한다.
11     Serial.println("your_inputs=" + your_inputs); // String 변수를 출력한다.
12     Serial.print("your_inputs in hexa=");
13     int decimal_number= your_inputs.toInt();    // 스트링을 정수로 바꾼다.
14     Serial.println(decimal_number, HEX);        // 16진수로 출력한다.
15     Serial.print("your_inputs in binary=");
16     Serial.println(decimal_number, BIN);        // 이진수로 출력한다.
17 }

```

만약 시리얼 모니터를 사용한다면 enter를 입력하기 전까지는 입력한 결과를 볼 수 없을 것이다. 왜냐 하면 enter를 입력할 때까지는 모니터에서 아두이노로 데이터가 전달되지 않기 때문이다.

## 6.6 직렬 포트 0의 역할

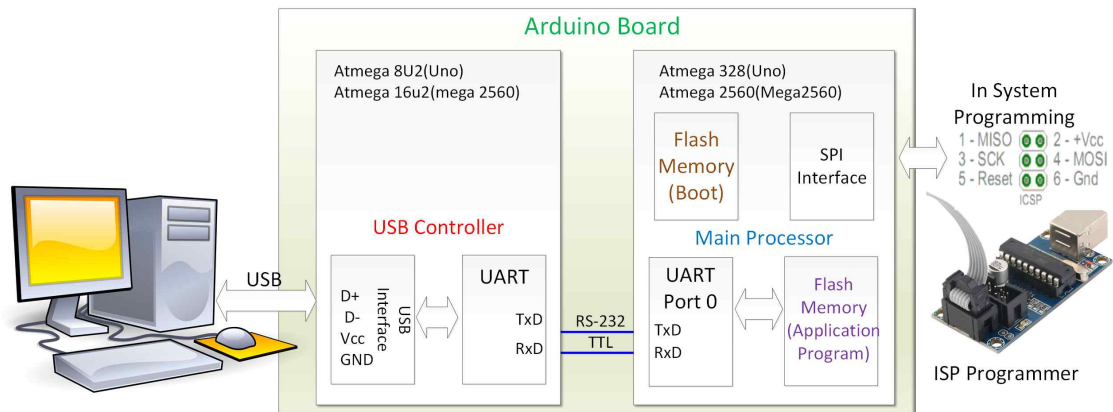


그림 6.6.1 USB 제어기와 메인 프로세서와의 역할

□ 아두이노 보드 내에 PC ↔ 아두이노 통신을 담당하는 프로세서가 하나 더 있다.

아두이노 보드는 모델이 따라 우노의 경우는 Atmega 328를 Mega 2560의 경우는 Atmega 2560 프로세서를 내장하고 있다. 그런데 USB를 통한 PC와의 통신 및 프로그램 업로드 기능을 지원하기 위해 이들 Main Processor와는 별도로 USB controller(Atmega 8U2 혹은 Atmega 16U2)를 내장하고 있다. 이 USB 제어기는 PC와의 통신을 담당하면서 RS-232 통신을 담당하는 내부의 UART<sup>16)</sup>를 통해 메인 프로세서의 UART 포트 0와 교신을 담당하고 있다. 그림 6.6.1은 이러한 역할 분담과 구조를 보인 것이다.

메인 프로세서의 UART0의 신호선은 단자 0, 1번에도 노출되어 있다. 이 단자는 USB 제어기의 UART와 공통선로를 사용하기 때문에 프로그램 업로드 중에는 사용할 수 없음을 유의해야 한다. 또한 TTL 레벨의 RS-232를 지원하기 때문에  $\pm 12V$ 를 사용하는 외부 RS-232 단자와 연결하면 손상을 입을 수

16) 보드 내에서 RS-232 프로토콜에 의거한 통신이 이루어지진다. 하지만 구태여 높은 전압을 사용할 필요가 없으므로 이런 경우에는 0V, 3.3/5V 수준의 TTL 전압을 사용하는 것이 일반적이다.

있다.

메인 프로세서의 메모리 공간은 application flash 영역과 boot flash 영역으로 나뉘어진다. 스케치로 작성한 사용자의 프로그램은 이 전달경로를 통해 메인 프로세서의 애플리케이션(application) 프로그램 메모리 영역에 업로드된다.

#### □ Booting 프로그램은 ISP 프로그래머로 업로드 한다.

boot 공간은 전원이 처음 인가한 후 제일 먼저 수행되는 공간으로서 스케치 프로그램 업로드 등과 같이 가장 기본적인 동작을 담고 있다. ISP(In System Programming)는 전원이 인가된 채로 외부에서 프로세서에 플래시 메모리 공간의 프로그램을 설치하는 기능을 말한다. 아두이노에서는 메인 프로세서의 부트용 플래시 메모리를 포함한 전체 영역의 프로그램을 교체할 때 이 기능을 이용한다. 이를 위해서는 별도의 ISP Programmer 장비가 필요하며 아두이노 보드는 이를 위해 6핀 ISP 콘넥터를 제공한다.

□ USB 제어기 ⇔ 메인 프로세서 직렬통신 부분 회로도

그림 6.6.2에는 이 프로세서와 관계된 메가 2560 모델의 회로도를 보였다. 왼쪽 블록은 USB 연결 단자이고, 우측 하단의 점선 블록은 Atmega2560 AVR 프로세서로서 RS-232 직렬통신으로 연결되어 있음을 보이고 있다.

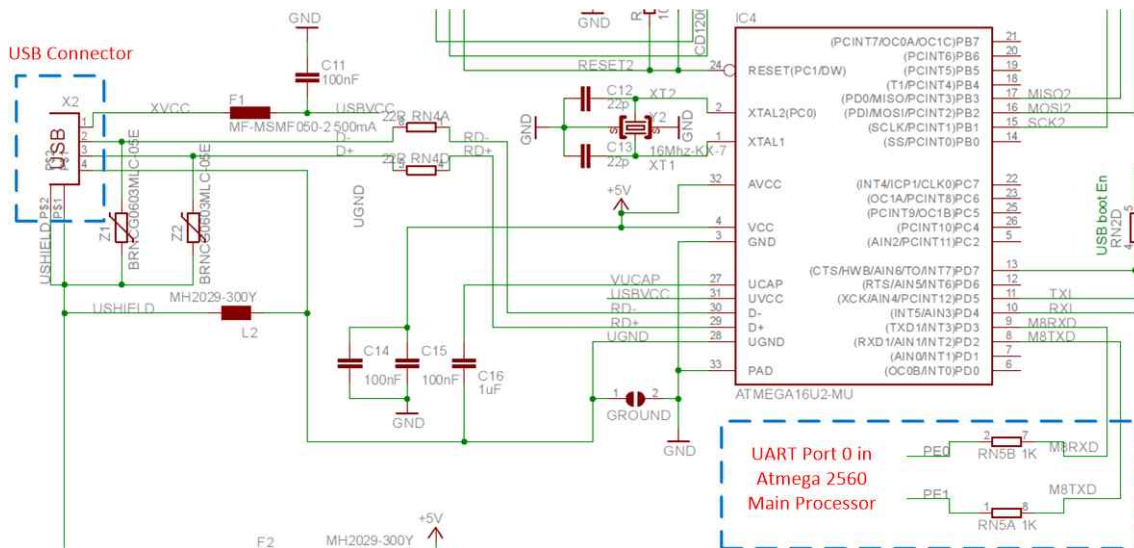


그림 6.6.2 USB 연결을 담당하는 소형 AVR(atmega 16U2) 관련 회로도

## □ 메가 2560의 추가 UART 포트 3개

그림 6.6.3은 메가 2560 모델의 메인 프로세서의 UART 담당 부분을 강조하여 보인 회로도이다. 회로도에서 점선 블록은 각각 Port 0~ Port 3의 신호선 TX, RX를 나타내고 있다.

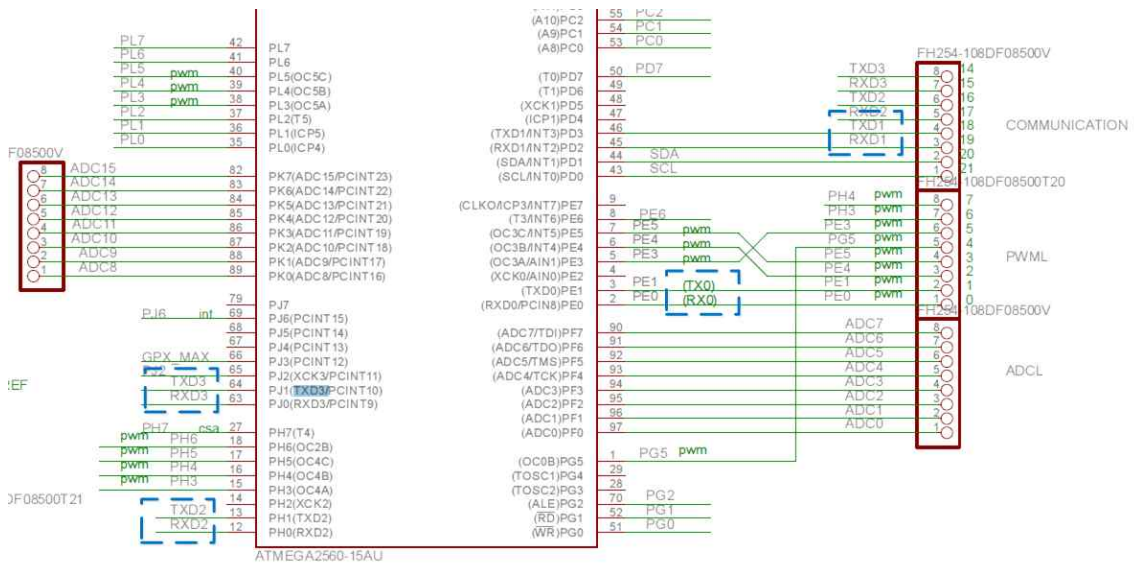


그림 6.6.3 직렬통신 포트 0~3까지를 표시한 메인 프로세서의 주변 회로도

## 6.7 소프트웨어 방식의 직렬통신

### □ 아두이노 우노는 직렬포트가 1개 밖에 없다.

아두이노 우노의 경우 내장되어 있는 UART 직렬통신 제어기가 1개 밖에 없다. 이 때문에 직렬통신은 1 포트만 지원한다. 그런데 이 포트는 보드에 장착되어 있는 RS-232↔USB을 위한 FTDI 칩과의 교신에 활용된다.

다른 장치와 직렬통신을 위해서 보드상에 단자 0과1이 이런 목적으로 제공된다. 그러나 이 기능은 PC로부터 독립된 standalone 형태로만 운영 가능하다. USB를 연결하면 PC와의 통신과 충돌할 수 있기 때문이다. 그러나 이렇게 되면 전원을 별도로 공급해야 할 뿐만 아니라 USB를 통한 프로그램 업로드가 불가능해 개발단계에서는 매우 불편하다.

### □ 소프트웨어로 GPIO를 이용하여 직렬통신을 실현한다.

이를 해결하기 위해 GPIO 단자 2개를 RxD와 TxD로 사용하는 [S/W 방식의 직렬통신 함수](#)가 제 3진영에서 개발되고 있다. 이 방식에서는 수신신호 RxD는 GPIO 입력으로 설정하고, 송신신호 TxD는 GPIO 출력으로 설정한다. 그리고 직렬데이터의 병렬화와 병렬데이터의 직렬화를 프로그램으로 구현하는 방식으로 UART의 기능을 프로그램으로 대신하는 것이다. 즉, UART 제어기가 할 일을 CPU가 대신하게 하는 것이다<sup>17)</sup>.

### □ 소프트웨어 시리얼 라이브러리

본 실험에서는 그동안 발표되었던 많은 Software Serial 라이브러리 중에서 [AltSoftSerial](#)의 라이브러리를 사용하기로 한다. 이 라이브러리는 그림 6.7.1에 보인 것처럼 아두이노 우노의 9번과 8번을 고정단자로 사용한다. 이 라이

---

17) 이 방식은 단순작업에 고기능의 CPU를 투입하는 정책이기 때문에 바람직한 설계기법이라고는 볼 수 없지만 원리적인 측면에서 그동안 놓쳐왔던 많은 부분을 다루기 때문에 교육적 차원에서 다루볼 만한 좋은 주제라고 판단된다.

브러리는 직렬통신을 S/W 방식으로 구현하기 위해 라이브러리 내부에서 단자 10번을 사용하기 때문에 본 소프트웨어 시리얼을 사용한다면 단자 10번은 사용할 수 없다<sup>18)</sup>.

color	signal	
	cable	Arduino Uno Pin Number
White	RxD	TxD(9번)
Green	TxD	RxD(8번)
Red	+5V	NC(No Connect). <b>연결하지 마시오<sup>19)</sup></b>
Black	GND	




그림 6.7.1 AltSoftSerial 단자와 USB to RS-232 TTL 케이블의 단자

본 케이블을 사용하기 위해서는 [전용 드라이버를 설치해야 한다는 설명](#)도 있었으나, 실제로는 자동 설치가 되는 것으로 확인되었다.

AltSoftSerial 함수를 사용하기 위해서는 Sketch IDE에 필요한 라이브러리를 해당 폴더에 복사해 넣는 절차가 필요하다. 최근 들어서는 이러한 설치 과정을 그림 6.7.2와 같이 IDE의 툴 메뉴에서 클릭으로 간단히 마칠 수 있다.

예제 6은 시리얼 모니터와 테라텀과의 문자를 교신하는 프로그램을 보인 것이다.

본 예제는 시리얼 모니터의 입력 문자를 테라텀으로 전달하고, 테라텀에서 입력한 문자를 시리얼 모니터로 전달한다. 즉, 아두이노의 내장된 직렬통신 0번의 입력된 문자를 AltSoftSerial으로 구현한 직렬통신의 입력으로 전달하여 이를 PC에 연결된 USB to RS-232 케이블을 통해 전달받아 테라텀의 화

18) 동기화된 직렬신호의 생성을 위해서는 내부에서 정확한 타이밍에 맞추어 인터럽트를 발생시키는 자원이 필요하다. 10번 단자는 주기적 클럭 펄스를 생성하기 위한 타이머의 PWM 신호 단자로 활용되며 이것이 인터럽트 발생자원으로 사용된다.

19) 이 케이블이 USB로부터 전원이 제공되기 때문에 케이블만으로도 다른 장치에 전원을 제공할 수 있는 기능이 있다. 그러나 전원이 아두이노로부터도 제공되므로 자칫 전압 차 때문에 연결된 보드들이 서로 손상을 입을 수도 있다.

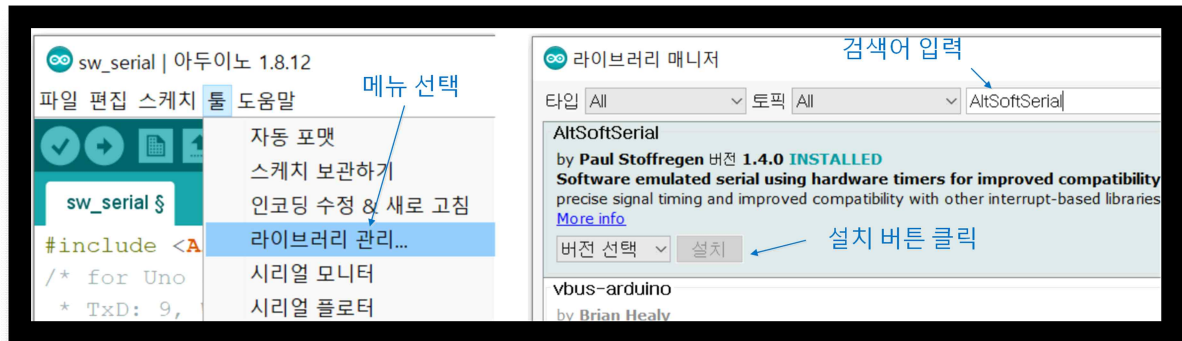


그림 6.7.2 AltSoftSerial 라이브러리의 메뉴 기반 설치

면에 출력한다.

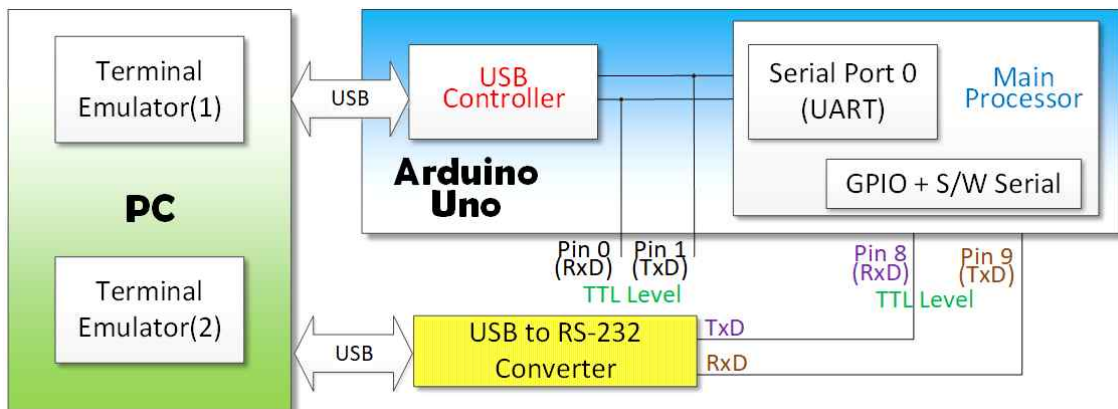


그림 6.7.3 예제 프로그램의 수행 동작 개념도

이 프로그램의 동작 개념도를 보면 그림 6.7.3과 같다. 터미널 에뮬레이터 (1)에서 입력한 문자는 USB 제어를 통해 메인 프로세서의 UART(포트 0)를 거쳐 프로그램에서 읽어낸다. 프로그램은 읽은 데이터를 S/W 시리얼 함수를 통해 GPIO 단자를 통해 출력한다. 이 데이터는 USB to RS232 변환기(변환 케이블)를 통해 USB를 통해 PC 측의 터미널 에뮬레이터 (2)에 전달된다.

터미널 에뮬레이터 (1), (2)는 각각 설치과정(자동 혹은 수동)을 통해 서로 다른 가상 COM 포트를 자동으로 배정받거나 혹은 수동으로 다른 COM 포트로 설정해주어야 한다.



▣ 예제 6 : 시리얼 모니터 창과 Tera Term 창과의 교신

**serial4.ino** : 아두이노 표준 클래스 및 함수 - Serial.begin(), Serial.read(), Serial.write()

```
01  #include <AltSoftSerial.h>
02  /* for Uno side: TxD: 9, White | RxD: 8, Green | NC: 10번 */
03  AltSoftSerial altSerial;
04  void setup() {
05      Serial.begin(9600);  // for Serial Monitor
06      Serial.println("AltSoftSerial Test Begin");
07      altSerial.begin(9600); // for Tera Term Emulator
08      altSerial.println("Hello World");
09  }
10  void loop() {
11      char c;
12      if (Serial.available()) { // Check Serial Monitor
13          c = Serial.read();    // Read from Serial Monitor
14          altSerial.print(c);   // Send it to Tera Term
15      }
16      if (altSerial.available()) { // Check Tera Term
17          c = altSerial.read();  // Read from Tera Term
18          Serial.print(c);      // Send it to Serial Monitor
19      }
20  }
```

## 6.8 비동기 직렬통신의 원리(이론)

### □ 프레임의 구성

RS-232 데이터 통신은 5~8 비트의 데이터로 포함하는 프레임(frame) 단위로 이루어진다. 프레임의 내부 구성은 통신 개시 전에 송신자, 수신자 양단 간의 사전 약속에 의해 통신 시작 전에 설정되어 있어야 한다.

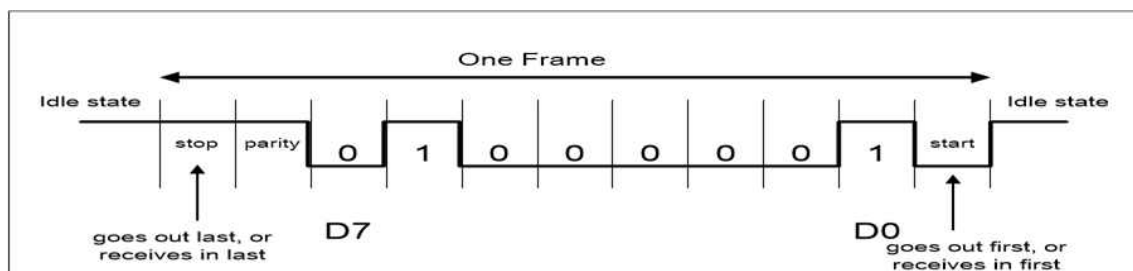


그림 6.8.1 프레임의 구성

그림 6.8.1에 RS-232 통신에서 사용되는 프레임의 구성을 보였다<sup>20)</sup>. 그림은 데이터가 우측부터 수신단 측으로 전송되는 모습을 보인 것이다. 그림에서 각 비트의 데이터 폭은 bps 설정에 따른다<sup>21)</sup>. 만약 9600bps의 전송 속도를 설정하였다면 한 개 비트의 시간적 길이는 1/9600 초가 된다.

평상시 데이터를 보내지 않는 idle state 상태는 논리 1로 약속되어 있다. 전송을 시작할 때는 송신단에서 전송이 개시된다는 것을 start bit를 통해 알린다. 시작 비트의 길이도 사전 설정한 바에 따라야 한다. RS-232에서는 1개의 선으로 동기신호와 데이터 신호를 모두 전송하기 때문에 재동기화를 위해 매 프레임마다 동기화 신호(start)를 전송하기로 되어 있다.

데이터 비트는 D0(LSB, Least Significant Bit)부터 전송이 시작된다. 사전 설정된 길이(5~8비트)의 데이터 전송이 끝나면 전송한 데이터에 부합하는 패리

20) 그림은 논리 기호를 보인 것이다. 실제 전송할 때는 송신단 측에서는 논리 0은 +5 ~ +12V, 논리 1은 -5V~-12V로 구현된다.

21) 동기를 맞추는 클럭신호를 사용하지 않고 1개의 데이터 신호선으로 다수의 비트를 직렬로 전송하는 것이 비동기 직렬통신의 핵심 아이디어이다.

티(parity) 비트를 보낸다. 패리티 비트는 지금 보낸 데이터 비트들의 전송 중에 오류가 발생할 경우를 대비한 비트로 이 역시 사전 설정한 바에 따른다. 맨 마지막에는 전송이 끝났음을 알리는 stop 비트를 전송한 후 다시 idle 상태에 들어간다.

## □ 프로토콜의 설정 요소

RS-232 통신 프로토콜은 통신속도(bps)외에 다음 사항이 사전 약속되어 있어야 한다.

Protocol의 내용				
순서	1	2	3	4
내용	시작 비트	데이터 비트	패리티 비트	정지 비트
크기(길이)	1	5, 6, 7, 8	1 (even, odd), None, Mark, No Mark	1, 1.5, 2

### (1) 시작 비트(start bit)

받는 측이 보내는 측의 시작점을 알아내는데 꼭 필요한 신호이다. 평상시 데이터 선이 H를 유지하다가 데이터를 보낼 일이 있으면 정해진 시간 길이만큼 L로 유지한다. 받는 측에서는 이 신호가 L가 되면 송신이 시작되었다는 사실을 알 수 있고 이 시점 이후로 일정 간격으로 정해진 데이터 길이만큼 데이터를 샘플링할 것이다.

### (2) 정지 비트(stop Bit)

데이터 전송의 끝을 의미하는 비트이다. 이 비트는 받는 측에게 데이터 처리를 할 수 있는 시간을 주기 위해 추가된다. 사전에 송수신 양측에서는 데이터 비트의 길이를 프로토콜 정보에 의해 알고 있기 때문에 데이터를 수신 받는 쪽에서는 이 시작 비트와 정지 비트에 의해 데이터의 처음과 끝을 인지할 수 있다.

### (3) 패리티 비트

데이터를 보낼 때 미리 데이터 비트의 1의 개수가 짝수인가, 홀수인가를

검사하여 이에 관한 정보를 패리티(parity) 비트에 담아 보낸다. 데이터를 수신하는 쪽에서는 패리티 비트를 검사하여 자신이 받은 데이터가 정상적으로 도달하였는지 판단할 수 있다. 패리티는 오류가 짝수 번 일어난 것에 대해서는 검출할 수 없는 단점이 있다. 다음의 설정이 활용된다.

- ① 짝수(even) 패리티: 자신의 비트를 포함하여 데이터 중에 있는 1의 개수가 짝수가 되도록 패리티 비트를 결정한다.
- ② 홀수(odd) 패리티: 자신을 포함하여 데이터 중에 있는 1의 개수가 홀수가 되도록 패리티 비트를 결정한다.
- ③ None: 데이터의 전송 속도를 높이기 위해 패리티를 쓰지 않는다.
- ④ mark/space: 패리티를 데이터 비트를 연산하여 결정하지 않고 1(mark) 혹은 0(space)으로 고정해서 사용한다.

## □ UART(Universal Asynchronous Receiver and Transmitter)

UART는 직렬통신을 위해 필요한 작업을 전담하는 제어기이다. CPU측의 설정 및 제어에 따라 직렬통신에 필요한 다음의 작업을 담당한다.

(1) 송수신 데이터의 직병렬 변환: 데이터를 주고 받는 직병렬 데이터 변환이 필요하다. 이 처리를 CPU의 도움없이 스스로 처리하여 CPU의 부담을 경감한다.

(2) 호스트와의 병렬 인터페이스 담당: 직렬 데이터가 들어오면 이를 CPU에 게 알려 데이터를 가져가도록 돕는다. CPU가 I/O 처리 장치의 데이터를 가져 가거나 보내주는 방식에는 다음 2가지가 있다.

- ① Programmed I/O : UART의 내부 상태 레지스터를 통해 데이터 송수신이 가능한지 알린다. CPU는 상태 레지스터를 읽어 보아 데이터를 UART가 처리할만한지 점검한다. 예를 들어 데이터가 들어왔는지는 상태

레지스터의 RBF(Receiver Buffer Full) 비트를 점검하여 알 수 있다. 데이터를 보내도 되는지는 TBE(Transmit Buffer Empty) 비트를 점검하여 알 수 있다.

② Interrupt driven I/O : 송수신 가능할 때 CPU에게 인터럽트를 발생하여 알린다. 예를 들어 RBF=1이면 데이터를 수신한 것인데 UART는 이를 단지 상태 레지스터에만 기록하지 않고 CPU 인터럽트를 요구하여 데이터를 가져가도록 알린다. 같은 방식으로 TBF=1이면 송신 버퍼가 비어 있으므로 CPU에게 데이터를 보내라고 인터럽트를 요구한다. 인터럽트 구동방식의 I/O 장치는 CPU의 부담을 경감하는 선진적인 기법이라고 할 수 있다.

(3) 데이터 프레임의 가공 및 오류 점검: 송신시에는 패리티를 넣어 주거나 시작/종료 비트를 넣어준다. 수신시에는 시작 비트를 감지하고, 패리티 비트를 계산하여 정상인지 확인하고, 종료 비트를 감지하여 정상 프레임 전송이 끝난는지 확인한다.

데이터 오류 및 프레임 오류가 발견되면 상태 레지스터에 그 해당 비트를 설정하고 인터럽트 허가 상태이면 인터럽트를 발생하여 CPU가 후속 처리를 하도록 돕는다. 또한, [브레이크\(break\) 조건](#)<sup>22)</sup>을 검출해도 인터럽트를 발생하게 만들 수 있다.

(6) 통신 프로토콜의 설정: UART는 통신속도(bps), 데이터 비트 수를 CPU의 프로그램으로 설정하는 기능이 있어서 다양한 통신 환경에 대처할 수 있도록 설계되어 있다.

UART는 처음 PC에서 8250이 채택된 이래 발전을 거듭하여 이제는 거의 모든 임베디드 프로세서에 기본으로 장착되어 있는 직렬통신제어장치로 자

---

22) 로직 0가 1 프레임 구간 이상 계속되는 조건을 말한다. 송신단에서 의도적으로 이런 신호를 보내면 수신단에서는 이런 조건을 감지하여 상태 레지스터에 break 조건을 감지하였음을 설정하고, CPU에게 인터럽트를 요청하여 수신단의 요청에 응하도록 한다. 어떤 작업을 수행할지는 전적으로 프로그램의 재량이라 할 수 있다.

리 잡고 있다.

표 6.8.1 UART 발전 역사

type	features
8250	First UART in this series(PC-XT). It contains no scratch register.
8250A	This UART is faster than the 8250 on the bus side. Looks exactly the same to software than 16450.
8250B	Very similar to that of the 8250 UART.
16450	Used in PC-AT's (Improved bus speed over 8250's). Operates comfortably at 38.4KBPS. Still quite common today.
16550	This was the first generation of buffered UART. It has a 16 byte buffer, however it doesn't work and is replaced with the 16550A.
16550A	Is the most common UART use for high speed communications eg. 14.4K & 28.8 K Modems. They made sure the FIFO buffers worked on this UART.
16650	Very recent breed of UART. Contains a 32 byte FIFO, Programmable X-On / X-Off characters and supports power management.
16750	Produced by Texas Instruments. Contains a 64 byte FIFO.

## □ RS-232 Signals

표 6.8.2에는 표준 RS-232 신호를 보였다. 오늘날에는 그중 노란색으로 표기한 9가지의 신호가 주로 사용된다.

표 6.8.2 RS-232 신호

25 Pin	9 Pin	신호	입출력	25 Pin	9 Pin	신호	입출력
1		NC(No Connect)		14		NC	
2	3	Transmitted Data	출력	15		NC	
3	2	Received Data	입력	16		NC	
4	7	Request To Send	출력	17		NC	
5	8	Clear To Send	입력	18		+Receive Current Loop Data	입력
6	6	Data Set Ready	입력	19		NC	
7	5	Signal Ground		20	4	Data Terminal Ready	출력
8	1	Data Carrier Detector	입력	21		NC	
9		+Transmit Current Loop Data	출력	22	9	Ring Indicator	입력
10		NC		23		NC	
11		-Transmit Current Loop Data	출력	24		NC	
12		NC		25		-Receive Current Loop Return	입력
13		NC					

그림 6.8.2에는 9핀 RS-232 콘넥터의 그림을 보인 것이다.

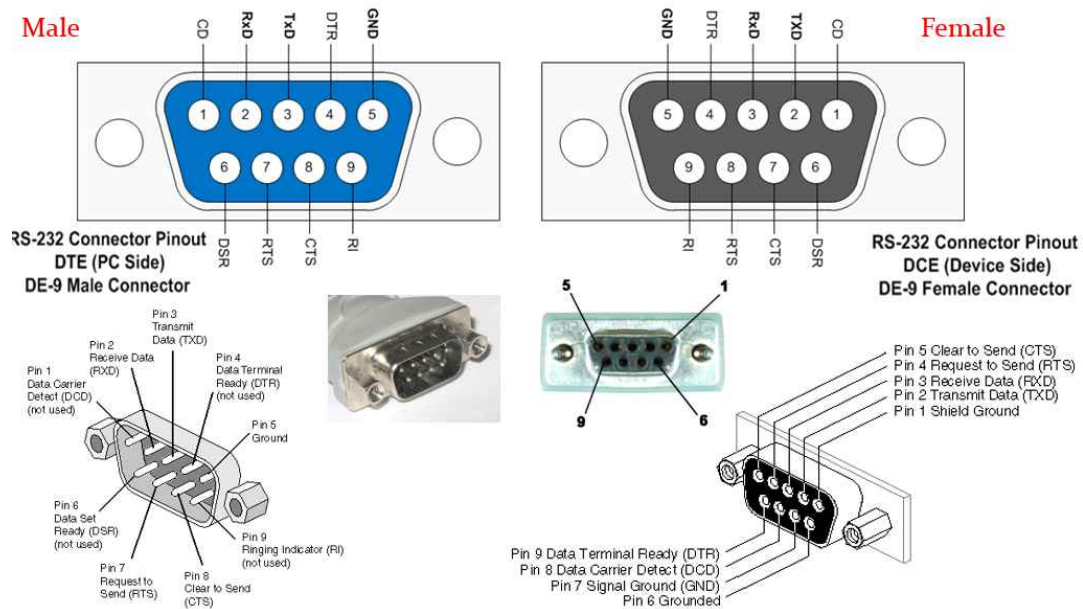


그림 6.8.2 9핀 RS-232 콘넥터와 신호선

### 순수 송수신 관련 데이터 신호와 케이블링

9개의 신호선 중 순수 데이터 송수신용 신호선은 2개이며 가장 단순하여 연결을 희망한다면 표 6.8.3과 같이 3개의 신호선만 있어도 기본적인 송수신 동작이 가능하다.

표 6.8.3 순수 송수신 데이터 신호선

신호	방향	의미
TxD	DTE→DCE	송신 데이터. DTE에서 출력하여 DCE로 입력된다. 데이터가 전송되는 선로이다.
RxD	DTE←DCE	수신 데이터. DCE에서 출력하여 DTE로 입력된다. 데이터가 전송되는 선로이다.
GND	DTE←DCE	접지선. 양측의 기준 전압을 맞추기 위해 필요하다.

이 신호들로는 다음과 같은 연결이 가능하다. (a)의 사례가 가장 일반적인

사례이다. (b)의 사례는 간혹 PC의 상대측에서 케이블을 twist 연결하지 않아도 되게끔 보드 상에서 RxD, TxD 단자를 바꾸어 놓은 경우이다. 이때는 RS-232 케이블이 twist되지 않은 것을 확인하고 연결해야 한다.

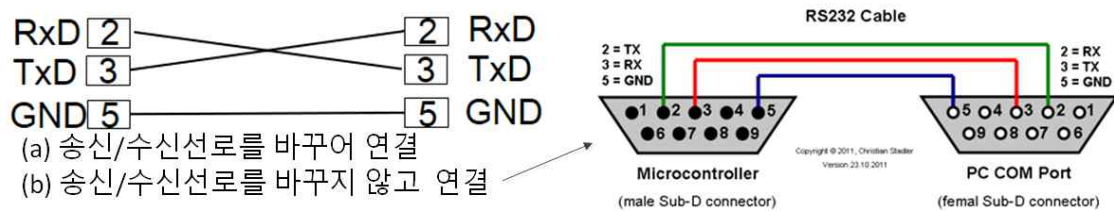


그림 6.8.3 데이터 교환을 위한 데이터 신호만의 케이블링

#### □ 흐름제어 - H/W 방식

데이터의 송수신을 위해 데이터의 흐름을 제어하여 데이터 수신에 실패하는 상황을 방지하기 위해 의사 교환 수단이 필요하다. 이런 수단을 handshaking이라 표현하기도 하는데 통신 의사를 교환하기 위한 수단을 흐름 제어(flow control)라고 한다. 이는 H/W와 S/W 방식의 제어로 나눌 수 있다. 이중 H/W 흐름 제어는 다음 2가지 조합 신호가 많이 활용된다.

##### (1) RTS flow control: RTS(Request To Send)/CTS(Clear To Send)

DTE가 송신하고자 할 때 DCE에게 RTS를 보내면 DCE는 받을 수 있으면 허락의 뜻으로 CTS를 송신하는 방식이다. DCE가 데이터를 먼저 송신하고자 할 때 DTE에게 허락을 구하는 신호는 없다<sup>23)</sup>. 이 방식의 흐름 제어를 위한 신호선은 표 6.8.4와 같은 기능을 수행한다. 이 신호들은 DTE와 DCE가 서로 데이터를 주고받는 역할을 교대로 수행하기 위해서 필요하다. 이 방식에서는 수신자가 될 때는 RTS를 0으로 만들고, 송신자가 될 때는 RTS를 1로 만드는 일을 반복한다. 그러나 전 이중(full duplex) 통신에서는 DTE가 항상 데이터를 수신할 수 있는 상태로 있다고 보고 RTS를

23) RS-232-E에서 개선하였다.



표 6.8.4 RTS/CTS 기반의 H/W 흐름제어

신호	방향	의미
Request To Send	DTE→DCE	DTE가 보낼 데이터가 있을 때 이를 모뎀에게 알리는 신호. 모뎀은 이 신호를 받으면 자신이 캐리어 신호를 생성한다.
Clear To Send	DTE←DCE	RTS에 대한 응답으로 DTE에 주는 신호. DCE가 원거리의 다른 DCE에게 데이터를 전송할 수 있음을 지시한다. 이를 받으면 DTE는 송신을 개시.

무시하기도 한다.

(2) DTR flow control: DTR(Data Terminal Ready)/DSR(Data Set Ready)

DTR : DCE가 DTE의 전원 인가 여부를 확인 할 수 있다.

DSR : DTE가 DCE의 전원 인가 여부 및 동작 가능 여부를 확인할 수 있다.

DTR/DSR 흐름 제어를 위한 신호와 그 밖의 신호선은 표 6.8.5와 같다.

표 6.8.5 DTR/DSR 기반의 흐름제어 신호 및 기타 신호선

신호	방향	의미
Data Terminal Ready	DTE→DCE	DTE(컴퓨터)가 전원 인가 후 이를 DCE(MODEM)에게 알림. DCE가 DTE에 접근하고자 할 때 이를 통제하고자 사용된다. 즉 모뎀이 컴퓨터가 켜져 있는지 알아볼 수 있다. 프로그램에 의해 언제나 세트될 수 있다.
Data Set Ready	DTE←DCE	모뎀이 전원이 인가되어 통신할 준비가 되어 있음을 알리는 신호. DCE는 다음 조건이 모두 만족되면 신호를 생성한다. 1. DCE가 전화선에 연결되어 있다 2. DCE가 검사, 대화, 다이얼 모드에 있지 않다. 이 신호가 꺼져 있으면 DTE는 DCE로부터의 모든 신호를 무시한다.
Data Carrier Detector	DTE←DCE	모뎀이 다른 모뎀과 접속되었다는 사실을 DTE에게 알리는 신호로서 외부로부터 전달된 데이터 통신을 위한 반송자(carrier)가 일정 수준에 맞으면 신호를 생성한다.
Ring Indicator	DTE←DCE	DCE가 DTE에게 전화가 오고 있음을 알리는 신호. 전화벨이 울리는 도중에는 ON 상태를 유지하고, 기타의 경우에는 OFF 상태가 된다.

H/W 기반 흐름제어를 위해 가장 많이 사용하는 결선은 그림 6.8.4의 (a) 방법이다. 원래는 RTS/CTS는 상대방의 CTS/RTS에 연결해야 하지만 연결선로 비용 절감을 위해 콘넥터에서 자체 연결하는 방식을 많이 사용한다.

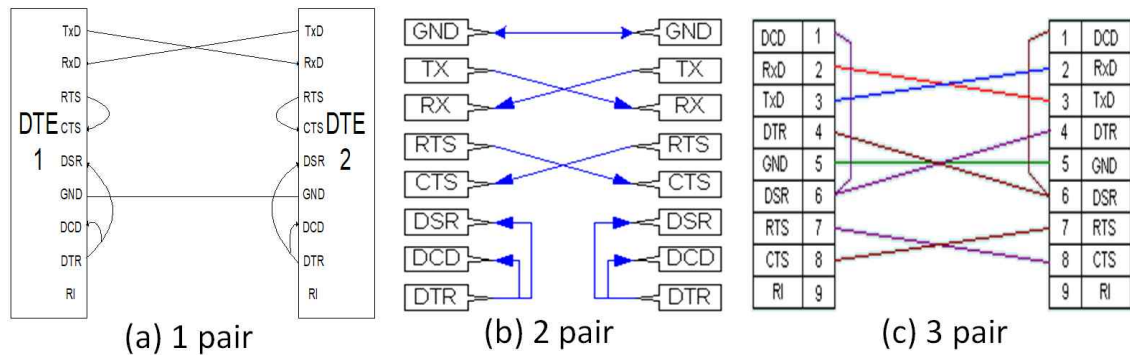


그림 6.8.4 H/W 흐름 제어를 위한 선로 결선

## □ 흐름제어-S/W 방식

S/W 흐름 제어는 아래와 같이 ASCII 코드의 제어문자 영역의 코드를 이용하여 전송을 허가/금지한다. 데이터 선 이외는 별도의 신호선이 필요없는 장점이 있다.

표 6.8.6 S/W 방식의 제어문자

Code	Meaning	ASCII	Dec	Hex	Keyboard
XOFF	Pause transmission	DC3	19	13	CTRL+S
XON	Resume transmission	DC1	17	11	CTRL+Q

H/W Flow control을 지원하는 케이블은 모두 S/W Flow Control도 지원한다. H/W 신호 혹은 제어문자를 이용해 XON(보내도 좋다), XOFF(보내지 말라)제어 코드를 상대방에 제공한다. 이 때문에 XON/XOFF 제어라고도 한다. 이 코드 - XON(0x11), XOFF(0x13) - 를 일반 데이터의 송신 중에 보낸다.

S/W 기반의 흐름 제어에 결선에서는 그림 6.85와 같이 단순한 결선이 가능하다. 하지만, H/W XON/OFF 제어 기반의 프로그램에서는 이런 케이블을 사용하면 안된다.

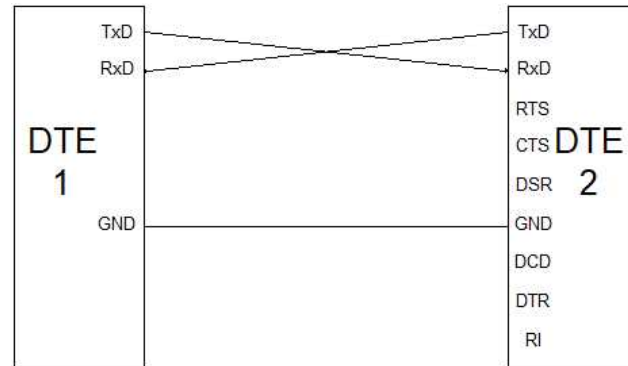


그림 6.8.5 S/W 흐름제어를 위한 단순 결선

#### □ RS-232 신호의 전압

RS-232 신호 전송에 사용되는 실제 물리 전압의 기준은 표 6.8.7에 보인 바와 같이 송신측인지, 수신측인지에 따라 사양이 다르다.

표 6.8.7 RS-232의 물리적 전압 기준

logic state	Receiver end voltage level	Transmitter end voltage level
logic 1 idle state, mark state	from - 3 V to - 25 V	from - 5 V to - 15 V
logic 0	from + 3 V to + 25 V	+ 5 V to + 15 V

전압을 높게 설정하고 그 여유분을 이렇게 넉넉하게 설정한 이유는 원거리 통신을 가능하게 하기 위해서이다. 디지털 회로에서 RS-232 전압을 지원하기 위해서는 그림 6.8.6에 보인 바와 같이 디지털 신호를 RS-232 전압에 맞도록 바꾸어 주는 voltage converter가 필요하게 된다. Embedded Processor 안에 내장된 직렬통신 담당 제어기(UART)는 디지털 신호(3.3V 혹은 5V)를 입출력 하여 직렬통신을 수행하며, 이 칩은 프로세서 외부 PCB 기판에 장착되어 외부 RS-232 장치에 필요한 전압 변환 기능을 수행한다.

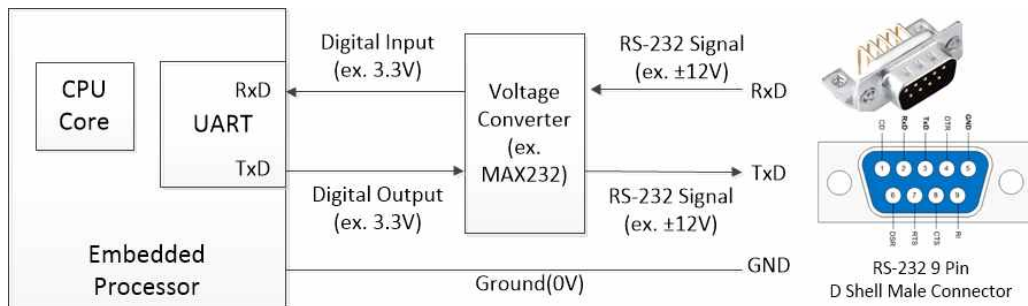


그림 6.8.6 전압 변환기의 역할

### □ 용어 설명: Simplex, Half Duplex, Full Duplex

단 방향(simplex) : 각 장치는 보내고 받는 기능 중 하나만 수행할 수 있다. 단 방향 전송만을 수행한다.

반 이중(half duplex) : 각 장치는 시간으로 나누어 보내고 받는 기능 중 하나만 수행할 수 있다. 시분할을 통해 양방향 전송이 가능하다.

전 이중(full duplex) : 각 장치는 보내고 받는 기능을 동시에 수행할 수 있다.

### □ 용어 설명: Null MODEM 연결

실제로는 PC/PC 간의 통신, computer/device 간의 통신에서는 null modem 통신을 많이 사용한다. MODEM 없이 DTE와 DTE 간의 통신을 NULL MODEM 통신이라 하며, 이때 사용하는 케이블을 NULL MODEM cable이라 한다.

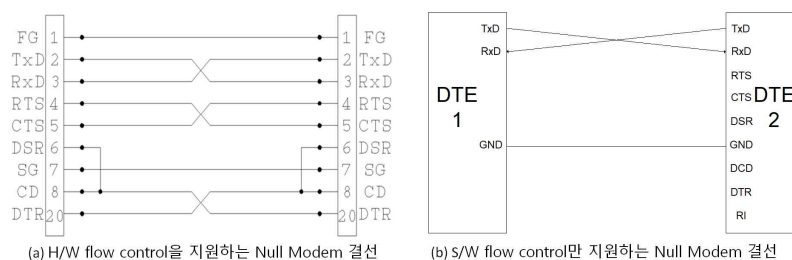


그림 6.8.7 Null Modem 통신의 선로 결선

## 6.9 고찰

다음 주제에 대하여 답할 수 있는지 검토해 보면서 그동안 습득한 지식을 정리해 보자.

1. 직렬통신과 병렬통신의 다른 점은 무엇인가? - 장점/단점 비교, 동작 원리 차이점 등에서 기술하시오.
2. RS-232란 무엇인가? 간략함 개념을 3줄 정도로 기술해 보시오.
3. UART란 무엇인가? 그 기능을 3개 이상 제시하시오. 이 장치를 볼 수 있는가? 볼 수 있다면 어느 곳에? - 질문이 3개입니다.~
4. RS-232를 사용하는 두 개의 통신 장치에서 통신을 위해 사전에 설정해야 할 내용을 제시하시오.
5. 두 개의 RS-232 장치가 통신하기 위해서 연결해야 할 최소의 신호선을 밝히고 그 연결도를 제시하시오.
6. UART가 외부에서 오는 RS-232 직렬 데이터 신호를 해석하여 8비트 병렬 데이터를 만들어 내는지 직렬 데이터를 병렬데이터로 변환하는 방법을 설명하시오.
7. 터미널이란 무엇인가? 터미널 에뮬레이터란 무엇인가? (즉, 프로그램이 수행하는 기능은 무엇인가?) 터미널 에뮬레이터의 장점은 무엇인가?
8. 아두이노 시리얼 모니터는 터미널 에뮬레이터라 할 수 있는가? 일반 터미널 에뮬레이터(예를 들어 테라텀)와 다른 점은 무엇인가?

9. 아두이노와 터미널 에뮬레이터를 이용하여 'Wn', 'Wt', 'Wr'의 ASCII 코드를 알아내는 방안에 대하여 기술하시오.
10. 흐름 제어를 하지 않으면 어떤 불편한 점이 생길 수 있는가? 흐름 제어를 이를 구현하는 2가지 방안에 대하여 제시하고, 각각의 방법에 대한 장단점에 대하여 기술하시오.

## □ 프로그램 작성 문제

11. 터미널에서 소문자를 입력하면(enter을 치지 않아도 입력되기를 희망함)아두이노가 이를 받아 대문자로 변환하여 다시 터미널로 출력하는 아두이노 프로그램을 작성하시오<sup>24)</sup>.
12. serialEvent() 함수를 사용하여 터미널에서 입력한 문자를 터미널 에뮬레이터<sup>25)</sup>로 반향하는 프로그램을 작성하시오. 예제 6.5.1과 같은 기능을 수행한다. 본 함수에 대한 소개는 아두이노 공식 사이트를 [참조](#) 바라며, 응용사례는 다른 문서를 참조하기 바랍니다. 공식 사이트의 예제는 [이곳](#)에 있습니다.

## ★ 직렬통신과 스피커를 결합한 프로그램 제작

13. 터미널 에뮬레이터를 이용하여 PC 키보드 1~8의 숫자 키로 입력을 받아 한 옥타브(낮은 '도'~높은 '도')를 연주하는 프로그램을 제작하시오. 이때 숫

---



24) 터미널(테라텀)에서 자신이 입력한 문자를 자신이 볼 수 있도록 한다는 것입니다. 설정을 바꾸어서 하라는 것이 아니라 아두이노에서 받은 문자를 다시 대문자로바꾸어 돌려주는 프로그램을 작성하라는 것입니다.

25) serialEvent() 함수는 AltSoftSerial에서는 지원하지 못하는 것 같습니다. 터미널 에뮬레이터를 사용해 주세요.

자를 입력할 때 enter를 입력하지 않아도 되어야 한다.

### ★ 직렬통신과 PWM(RGB LED)을 결합한 프로그램 제작

14. RGB LED에 PWM을 기반으로 아래 표의 색상을 RGB LED로 출력하고자

Color	Color Name	Hex Code #RRGGBB	Decimal Code R,G,B
	orange	#FFA500	(255,165,0)
	khaki	#F0E68C	(240,230,140)
	sky blue	#87CEEB	(135,206,235)
	purple	#800080	(128,0,128)
	hot pink	#FF69B4	(255,105,180)
	chocolate	#D2691E	(210,105,30)

한다. 터미널 에뮬레이터의 창에서 이를 색상의 영문 첫 글자(소문자로 가정)를 입력하면<sup>26)</sup> REG LED가 이들 색상으로 바꾸어 출력하는 프로그램을 작성하시오. 만약 대문자로 'B'를 입력하면 LED가 off되는 동작을 실행한다. 아래 실행 사례(터미널 에뮬레이터의 창)를 참고하여 작성하기 바랍니다.

```
(o)range, (k)haki, (s)ky blue, (p)urple, (h)ot pink, (c)hocolate, (B)lack
select input=o
(o)range is selected.
select input=s
(s)ky blue is selected.
select input=B
(B)lack is selected.
(o)range, (k)haki, (s)ky blue, (p)urple, (h)ot pink, (c)hocolate, (B)lack
select input=p
(p)urple is selected.
```

주의점:

<sup>26)</sup> 엔터는 입력하지 않는다.

- (1) 선택한 문자(o, s ,B, p 등)가 화면에 출력되어야 합니다.
- (2) 'B'를 입력하면 LED가 꺼져야 합니다.
- (3) 다른 입력을 시행하지 않으면 그 상태로 계속 있어야 합니다.

16. 문제 15를 수행하는 중에 Cyan, Magenta, Yellow 중의 'C', 'M', 'Y' 중의 한 글자를 시리얼 모니터 창에서 입력하면 현재 진행 중인 색상의 출력을 멈추고 방금 입력한 CMY 색상 출력을 3초간 진행한 후 원래의 메인 루틴으로 복귀한다. 이 동작을 구현하시오<sup>27)</sup>. 즉 15번의 기능과 새로 추가된 기능이 동시에 지원하는 기능을 구현하라는 미션입니다.

17. Ardu-Ez 키트 2대를 이용하여 터미널 에뮬레이터 프로그램으로 서로 채팅하는 스케치 프로그램을 작성하고 실험 결과를 제시하시오.

**\* 주의 : RxD, TxD가 각각 TTL 전위를 갖고 있으므로 RS-232 포트에 직접 연결하면 안된다. PC의 RS-232 포트는  $\pm 12V$ 의 전위를 갖고 있기 때문에 다른 장치에 손상을 입힐 수도 있기 때문에 본 실험을 시도할 때는 주의가 요한다. 💣\***

18. interrupt() 함수를 이용하여 serialEvent() 함수를 설계하시오. **14장 인터럽트 후 재검토 요망 💣\***

---

27) 이 동작은 SerialEvent() 함수를 사용하면 좋을 것이다. 그러나 이 함수는 AltSoftSerial 라이브러리에서는 발견되지 않는다. 이 때문에 아두이노 우노에서는 시리얼 모니터를 써야 할 것이다. 만약 메가 2560 모델이라면 추가 3개의 직렬포트가 지원되므로 이 함수를 터미널 에뮬레이터를 쓰는 환경에서도 사용 가능하다.