

Chapter 210. UART - 시리얼 텍스트 LCD

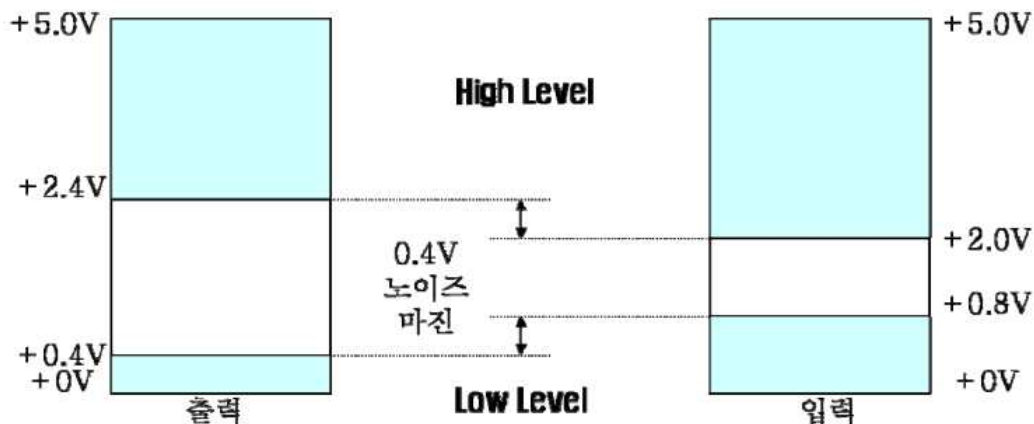
1. UART

UART는 Universal Asynchronous Receiver/Transmitter의 약어로 비동기식의 데이터 전송 방식을 말한다. UART가 범용(universal)인 이유는 데이터 전송을 위한 속도나 전기 신호의 수준, 즉, 논리 0과 1을 나타내는 전압을 UART 외부의 드라이버 회로를 통해 변경할 수 있기 때문으로 흔히 RS-232, RS-422, RS-485 등과 함께 사용된다.

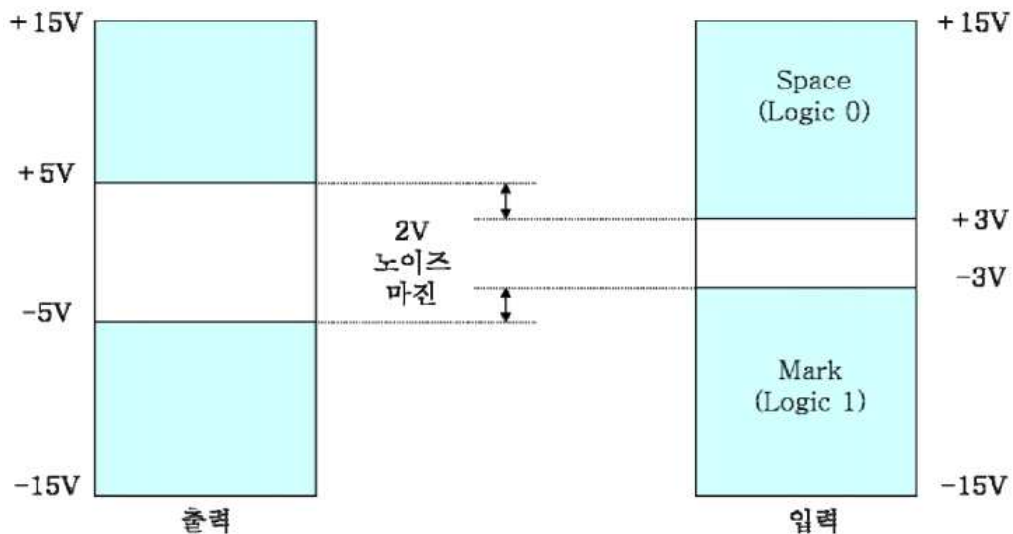
UART와 RS-232는 흔히 혼돈된다. UART는 대부분의 마이크로컨트롤러 내부에 구현되어 있는 통신을 위한 하드웨어 회로를 가리키며, UART의 주요 기능은 기기에서 송신할 병렬 데이터를 직렬로 바꾸어주고, 기기로 수신된 직렬 데이터를 병렬로 바꾸어 주는 것이다. 마이크로컨트롤러에서 UART는 TTL (Transistor-Transistor Logic) 레벨 신호를 사용한다. TTL 레벨 신호란 중앙처리장치(CPU)의 전압을 기준 전압으로 데이터를 송수신하는 것을 말한다. 아두이노 우노에 사용된 ATmega328의 경우 5V 전압을 사용하므로 5V는 논리 1에, 0V(GND)는 논리 0에 해당한다.¹⁾ 이처럼 TTL 레벨을 사용하면 CPU의 동작 전압을 그대로 통신을 위해 사용하므로 별도의 변환 과정 없이 간단하게 데이터를 송수신할 수 있는 장점은 있지만, 낮은 전압을 사용하기 때문에 장거리 통신에서는 잡음과 간섭에 의해 에러율이 높아지는 단점이 있다. 이를 보완하기 위해 TTL 레벨 신호를 높은 전압을 사용하여 송수신할 수 있도록 해주는 것이 바로 RS-232이다.

RS-232는 1969년 미국 EIA(Electric Industries Association)에 의해 정해진 직렬 통신을 위한 표준 인터페이스이다. TTL의 경우 마이크로컨트롤러의 VCC를 기준 전압으로 사용하는 반면 RS-232는 -3V에서 -25V사이의 전압을 논리 1, +3V에서 +25V 사이의 전압을 논리 0으로 나타내는 부논리(negative logic)를 사용한다. 일반적으로 컴퓨터에서는 +13V와 -13V로 논리 0과 논리 1을 표현한다.

1) 마이크로컨트롤러의 종류에 따라서는 3.3V를 사용하기도 하며 이 경우 TTL 레벨은 3.3V가 된다. 이 책에서는 ATmega328을 기준으로 TTL 레벨은 5V로 가정하였다.



(a) TTL 레벨



(b) RS-232 레벨

그림 1. TTL과 RS-232 사용 전압 비교

그림 1은 TTL 레벨과 RS-232 레벨에서 논리 0과 논리 1로 인식되는 전압의 범위를 비교한 것이다. 출력 전압은 데이터 송신 장치에서의 전압을 나타내며, 입력 전압은 데이터 수신 장치에서의 전압을 나타낸다. 입력 전압의 범위가 더 넓은 이유는 출력 전압이 전송 과정에서 잡음이나 간섭에 의해 바뀔 수 있기 때문이다. 그림 1의 비교에서 볼 수 있듯이 RS-232는 기준 전압이 높고, 2V의 잡음 마진이 있어 장거리 전송에서 UART에 비해 유리하다. 실제 UART와 RS-232는 더 많은 차이가 있지만 마이크로컨트롤러를 사용할 때 반드시 기억해야 할 점은 두 신호의 레벨이 다르므로 직접 연결하여 사용할 수 없다는 점이다. UART와 RS-232는 기본적으로 두 개의 신호선(RX, TX)과 GND만 연결하면 기본적인 통신이 가능하다. 하지만 컴퓨터와 아두이노 우노를 직접 연결한다면 아두이노에 과도한 전압이 가해져서

망가질 수 있으므로 주의해야 한다.

RS-232는 역사가 오래되고 흔히 사용되는 시리얼 인터페이스이지만 1:1 연결만을 지원하는 한계가 있다. 또한 일부 기기의 경우 최대 1Mbit/s의 전송 속도를 낼 수 있지만 대부분의 경우 115.2Kbit/s가 최대 전송 속도이다. 이를 보완한 것이 RS-422로 1:n (n은 최대 10) 연결을 지원하며 최대 전송 속도 역시 10Mbit/s에 달한다. RS-485는 RS-422를 확장하여 m:n (m, n은 최대 32) 연결을 지원하며 최대 전송 속도는 RS-422과 동일하게 10Mbit/s이다.

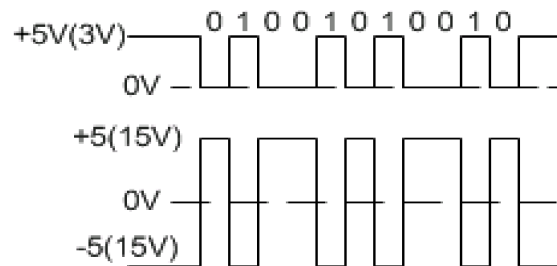


그림 2. TTL 레벨과 RS-232 레벨 사이의 관계

그림 2는 마이크로컨트롤러에서 흔히 사용되는 UART(TTL 레벨 사용)와 RS-232의 전압 레벨을 비교한 것이다. 이처럼 서로 다른 레벨의 신호를 사용하는 기기들 사이에는 레벨 변환기(level converter 또는 level shifter)가 필요하다. 컴퓨터와 아두이노의 시리얼 통신에 관해서는 별도의 장에서 다루며 이 장에서는 UART 시리얼 통신을 사용하는 주변장치, 특히 시리얼 방식의 텍스트 LCD를 아두이노 우노에 연결해 본다. 이 장에서 사용할 텍스트 LCD 역시 5V를 사용하지만 3.3V를 사용하는 기기 역시 증가하고 있으며 3.3V를 사용하는 장치를 아두이노 우노와 UART 시리얼 통신으로 연결하기 위해서도 레벨 변환기가 필요하다는 점은 기억해야 한다.

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

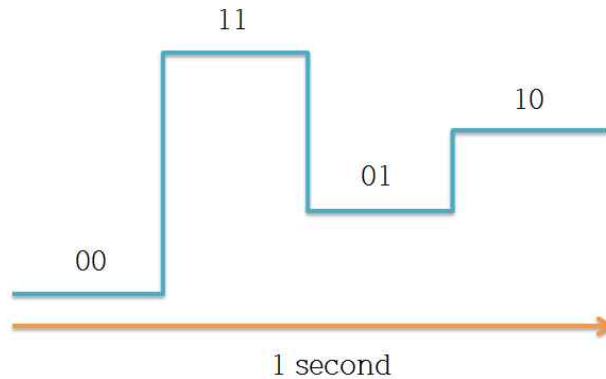


그림 3. 보울과 bps

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. 특히 UART는 비동기 방식의 통신으로 동기화를 위한 클록을 별도로 전송하지 않으므로 데이터의 시작과 끝을 표시하기 위해 '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 10비트 데이터를 전송하는 것이 일반적이다. 데이터가 전송되지 않는 상태에서 데이터 핀은 항상 '1'의 상태에 있으며, 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 수신된 10비트의 데이터 중 실제 데이터에 해당하는 8비트만을 찾아주는 등의 작업을 위한 UART 전용의 하드웨어가 대부분의 마이크로컨트롤러에 포함되어 있으며, 아두이노 우노에서는 디지털 0번과 1

번 핀이 하드웨어 UART 통신을 위해 사용된다. 아두이노에서는 하드웨어로 구현된 UART 통신을 통한 시리얼 통신을 간단히 시리얼 통신이라 불린다. 하지만 엄밀히 말해 UART 통신은 시리얼 통신의 한 종류라는 점을 잊지 말아야 한다.

2. 시리얼 텍스트 LCD

아두이노 우노에 사용된 ATmega328 마이크로컨트롤러는 하나의 UART 시리얼 통신을 지원하며 이는 디지털 0번과 1번 핀을 통해 이루어진다. 아두이노 우노 보드에서 디지털 0번 핀에는 'RX', 디지털 1번 핀에는 'TX'로 표시된 것을 확인할 수 있다. 'RX'는 'Receive'의, 'TX'는 'Transmit'의 약어로 데이터가 송수신되는 통로를 나타낸다. 아두이노에서는 전원 연결까지 포함하여 4개의 연결(RX, TX, VCC, GND)을 사용하는 경우가 대부분이다. 아두이노의 기본 라이브러리 중 하나인 LiquidCrystal 라이브러리를 통해 사용할 수 있는 텍스트 LCD의 경우 최소 6개의 데이터 선 연결을 필요로 하지만, 시리얼 텍스트 LCD는 데이터를 쓰기만 하는 경우 전원을 제외하고 TX 하나의 연결만이 필요하므로 입출력 핀의 수가 제한된 마이크로컨트롤러에서 간단하게 사용할 수 있다. 그림 5는 이 장에서 사용할 시리얼 방식의 텍스트 LCD를 나타낸다.

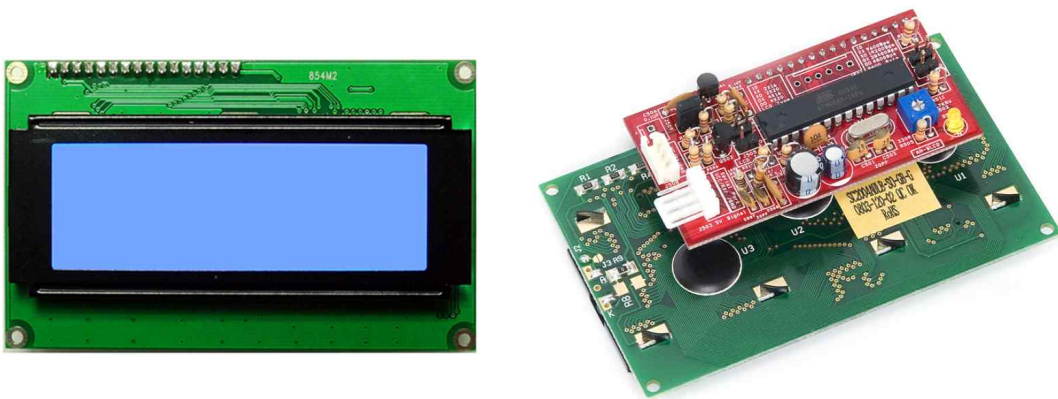


그림 5. 시리얼 방식의 텍스트 LCD²⁾

그림 5의 텍스트 LCD는 4×20개의 문자를 표시할 수 있다. 통신 속도는 기본적으로 9600으로 설정되어 있으며 점퍼 설정에 의해 조절할 수 있다. 이 장에서는 아두이노에 연결하여 사용하겠지만 RS-232 레벨의 신호 역시 받을 수 있으므로 컴퓨터와 직접 시리얼 연결도 가능하다.

2) 이미지 출처: <http://www.newtc.co.kr>

그림 5의 시리얼 텍스트 LCD는 터미널 모드(terminal mode)와 커맨드 모드(command mode)의 두 가지 모드에서 동작할 수 있다. 터미널 모드는 시리얼 포트로 전달되는 아스키 데이터를 LCD 화면에 나타내는 모드에 해당한다. 텍스트 LCD의 뒷면에는 모드 설정을 위한 점퍼가 존재하므로 사용하고자 하는 모드에 맞게 점퍼를 설정하면 된다.

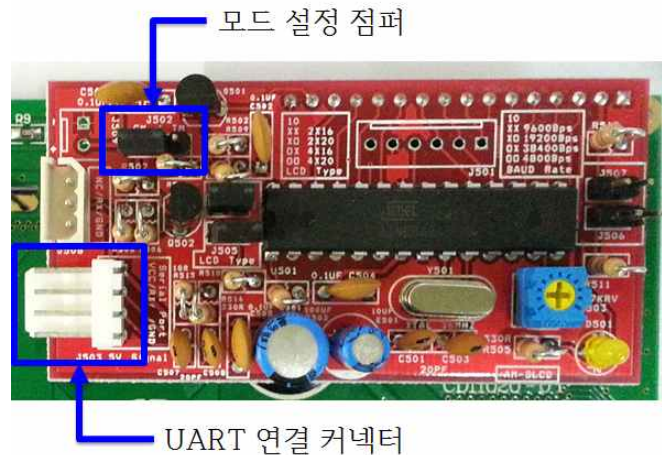


그림 6. 시리얼 텍스트 LCD 뒷면

텍스트 LCD에서 데이터를 읽는 경우는 없으므로 데이터 전달을 위해 시리얼 텍스트 LCD의 RX와 아두이노의 TX만을 연결하면 된다. 간단하지 않은가? 하나의 연결선만으로 텍스트 LCD에 원하는 문자를 출력할 수 있다. 다만, 그림 6을 자세히 살펴보면 시리얼로 전달된 데이터를 해석하여 실제 텍스트 LCD로 출력하기 위해 전용의 마이크로컨트롤러가 사용되고 있음을 발견할 수 있다.

시리얼 텍스트 LCD를 ATmega328의 UART 시리얼 포트와 연결하면 간단하게 문자를 출력할 수 있다. 하지만 아두이노에서 스케치를 업로드하기 위해 사용하는 핀 역시 UART 시리얼 포트인 디지털 0번과 1번이다. 시리얼 텍스트 LCD는 아니지만 일부 장치의 경우 UART 시리얼 포트에 연결한 상태에서 스케치를 업로드하면 업로드에 실패할 수 있다. 이처럼 UART 시리얼 포트의 중복 사용으로 인해 스케치 업로드에서 오류가 발생하는 이 경우에는 장치를 제거하고 프로그램을 업로드 한 후 다시 장치를 연결하여 결과를 확인하여야 한다.

3. Serial 클래스

Serial은 아두이노와 컴퓨터 및 기타 주변 장치 사이의 UART 시리얼 통신을 담당하는 클래스이다. 특히 Serial 클래스의 UART 통신은 디지털 0번과 1번 핀을 통해 수행되는,

ATmega328 마이크로컨트롤러에서 하드웨어 수준에서 지원되는 UART 통신을 사용하므로 하드웨어 시리얼(hardware serial)이라고도 불린다. ATmega328의 UART 시리얼 포트는 ① 스케치를 업로드하기 위해, ② 컴퓨터와의 시리얼 통신을 위해, ③ UART 통신을 사용하는 주변 기기와의 통신을 위해, 그리고 ④ 범용 입출력 핀으로 사용될 수 있다.

아두이노 우노를 컴퓨터와 USB로 연결하면 컴퓨터에는 가상의 시리얼(COM) 포트가 생성되고 이 포트를 통해 스케치를 업로드 할 수 있으며 컴퓨터와의 시리얼 통신을 수행할 수 있다. 하드웨어에 의해 지원되는 컴퓨터의 COM 포트는 RS-232 통신을 사용하지만, 아두이노를 컴퓨터와 연결하였을 때 만들어지는 시리얼 포트는 ‘가상의’ COM 포트로 USB를 통해 5V 전압을 기준으로 통신이 이루어지는 차이가 있다. 컴퓨터와의 통신에 대해서는 별도의 장에서 자세히 다루며 이 장에서는 주변기기와의 UART 통신 방법인 세 번째 경우를 다룬다. 표 1은 하드웨어 시리얼 포트가 사용되는 경우를 비교한 것이다. 범용 입출력 핀으로 사용하는 경우는 일반적인 입출력 핀을 사용하는 방식과 동일하므로 비교에서 제외하였다.

	스케치 업로드	컴퓨터와 시리얼 통신	주변기기와 시리얼 통신
연결 대상	컴퓨터	컴퓨터	주변기기
연결 대상이 사용하는 통신 방법	USB	USB	UART 시리얼
변환기 사용	USB-UART 변환기	USB-UART 변환기	(사용 안함)
Serial 클래스 사용	(스케치 작성과는 무관하므로 사용하지 않음)	사용 가능	사용 가능

표 1. 하드웨어 시리얼 포트 사용 방법 비교

표 1에서 알 수 있듯이 스케치 업로드 과정 이외의 시리얼 통신을 위해서는 Serial 클래스를 사용할 수 있다. 스케치 업로드 역시 시리얼 통신을 사용하기는 하지만 스케치의 동작과는 상관이 없는 글자 그대로 스케치를 마이크로컨트롤러로 옮겨서 설치하는 과정으로 아두이노의 부트로더에 의해 제어되는 과정이다.

Serial 클래스는 아두이노의 기본 클래스 중 하나로 별도의 헤더 파일을 포함시키지 않고도 사용할 수 있다. 특히 아두이노 우노에는 오직 하나의 하드웨어 UART 포트만 존재하므로 객체를 별도로 생성하지 않고도 Serial을 유일한 객체로 사용할 수 있다.

UART 통신을 위해서는 반드시 주변장치와의 통신 속도를 일치시켜 주어야 하며, 속도 설정과 더불어 다양한 옵션을 begin 멤버 함수에서 설정할 수 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
```

속도 이외의 옵션에는 데이터 비트의 수, 정지 비트의 수, 패리티 비트의 종류 등이 있으며 디폴트값은 8비트 데이터 비트, 1비트 정지 비트, 패리티 사용하지 않음으로 설정되어 있다. 대부분의 장치들이 아두이노의 디폴트 설정과 동일한 설정을 사용하므로 특별한 경우가 아니면 변경할 필요는 없다. 더 이상 통신이 필요하지 않다면 end 멤버 함수로 통신을 종료할 수 있다.

시리얼 통신이 초기화된 이후 데이터 송신을 위해 사용할 수 있는 함수에는 문자 기반의 print 및 println 함수와 이진 데이터 기반의 write 함수가 있다. println 함수는 전송되는 문자열 끝에 개행문자를 추가하는 점을 제외하면 print 함수와 동일하다. print 함수는 인자로 주어지는 내용을 문자열로 변환하여 출력하므로 문자나 문자열뿐만 아니라 정수와 실수 역시 문자열로 출력할 수 있다. 반면 write 함수는 이진 데이터를 직접 출력한다. 예를 들어 정수 65를 print 함수로 Serial.print(65); 와 같이 출력하는 경우에는 65가 문자열 "65"로 변환되고 숫자 '6'과 '5'에 해당하는 아스키코드 값인 54와 53가 전송된다. 반면 write 함수를 사용하여 Serial.write(65); 와 같이 출력하는 경우에는 65가 직접 출력되며, 이는 65에 해당하는 아스키 문자인 'A' 1바이트를 출력하는 것과 동일하다.

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

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

매개변수

ch : 출력할 바이트 단위 데이터

반환값 : 출력한 바이트 수

Serial 클래스에는 데이터 송신을 위한 함수(print, write)와 더불어 수신을 위한 함수도 마련되어 있다. 수신을 위한 함수에는 수신된 데이터가 존재하는지의 여부를 알아보는 available, 수신된 데이터를 바이트 단위로 읽어오는 read 함수와 peek 함수가 있다. 수신된 데이터는 수신 버퍼에 저장되며 available 함수는 수신 버퍼에 읽지 않고 저장되어 있는 데이터의 바이트 수를 반환한다. read와 peek 함수는 1바이트의 데이터를 읽어오는 것은 동일하지만 read 함수의 경우 읽어온 데이터를 수신 버퍼에서 삭제하는 반면 peek 함수는 수신 버퍼에서 삭제하지 않는 차이가 있다.

```
int available(void)
```

매개변수 : 없음

반환값 : 시리얼 통신 수신 버퍼에 저장된 데이터의 바이트 수

```
int peek(void)
```

매개변수 : 없음

반환값 : 시리얼 통신 수신 버퍼의 첫 번째 바이트 데이터 또는 -1

```
int read(void)
```

매개변수 : 없음

반환값 : 시리얼 통신 수신 버퍼의 첫 번째 바이트 데이터 또는 -1

4. 소프트웨어 시리얼

Serial 클래스는 아두이노 우노에 사용된 ATmega328의 UART 시리얼 통신을 담당하는 클래스로 다양한 용도로 널리 사용되고 있다. 한 가지 문제점이라면 ATmega328에는 오직 하나의 UART 시리얼 포트가 존재한다는 점이다. UART 시리얼은 1:1 통신만을 지원한다. 따라서

UART 시리얼 통신을 사용하는 여러 개의 장치를 아두이노 우노에 연결할 수 없다. 하지만 방법이 없는 것은 아니며 그 중 한 가지가 아두이노의 기본 라이브러리 중 하나인 SoftwareSerial 라이브러리를 사용하는 방법이다. 소프트웨어 시리얼은 글자 그대로 하드웨어로 구현되어 있는 UART 시리얼 통신의 기능을 소프트웨어를 통해 구현해 놓은 것에 해당한다.

아두이노 우노의 경우 모든 데이터 입출력 핀을 UART 시리얼 통신을 위해 사용할 수 있다. 아두이노 우노는 20개의 입출력 핀을 사용할 수 있으므로 최대 10개의 UART 시리얼 통신을 사용할 수 있다는 말이다. 물론 이 중에는 하드웨어 시리얼이 포함된다. 하지만 소프트웨어 시리얼의 경우에는 한 번에 하나의 UART 시리얼 통신만을 수행할 수 있는 제약이 있다. 즉, 여러 개의 UART 시리얼 장치를 연결할 수는 있지만 실제 통신이 가능한 장치는 한 번에 하나뿐이다. 또한 아두이노 우노의 경우 모든 핀이 소프트웨어 시리얼을 위해 사용될 수 있지만 보드의 종류에 따라 사용할 수 있는 핀에 제약이 있을 수 있다. 이는 소프트웨어 시리얼이 데이터 수신 감지를 위해 인터럽트를 사용하지만 모든 입출력 핀이 인터럽트를 지원하지는 않기 때문이다.

SoftwareSerial 클래스에는 UART 시리얼 통신을 위한 여러 가지 멤버 함수들이 정의되어 있으며 대부분 Serial 클래스의 멤버 함수와 동일하다. 여기서는 Serial 클래스에서와는 다른 몇 가지 멤버 함수들만을 살펴본다. Serial 클래스의 경우 오직 하나의 객체만을 생성할 수 있으므로 별도로 객체를 생성하지 않았다. 하지만 SoftwareSerial 클래스는 최대 9개까지의 객체를 생성할 수 있으므로 객체 생성 과정이 필요하며 이 때 RX와 TX로 사용할 핀을 지정해야 하는 점에서 차이가 있다.

```
SoftwareSerial(uint8_t rx, uint8_t tx, bool inverse_logic = false)
```

- 매개변수

rx : 수신 핀 번호

tx : 송신 핀 번호

inverse_logic : 송수신 비트 반전

- 반환값 : 없음

UART 시리얼 통신을 초기화하는 멤버 함수는 begin이다. 하지만 Serial 클래스와 달리 SoftwareSerial 클래스는 매개변수로 속도만을 지정할 수 있다. 데이터 비트 수, 정지 비트, 패리티 비트 등은 지정할 수 없으며 Serial 클래스의 디폴트값(8비트 데이터 비트, 1비트 정지 비트, 패리티 사용하지 않음)만을 지원한다.

```
void begin(long speed)
```

- 매개변수
speed : 속도
- 반환값 : 없음

소프트웨어 시리얼은 동시에 여러 개의 연결을 사용할 수 있지만 이 중 실제로 데이터를 수신할 수 있는 연결은 하나뿐이다. 따라서 여러 개의 UART 시리얼 연결 중 데이터를 수신할 포트를 지정해 주어야 하며 이를 위해 listen 멤버 함수를 사용한다.

```
bool listen(void)
```

- 매개변수 : 없음
- 반환값 : 수신 대기 포트가 바뀌면 true를 반환

listen 멤버 함수는 해당 소프트웨어 시리얼 포트의 RX 핀으로 데이터 수신을 대기하도록 한다. 수신 대기 상태에 있지 않은 포트에 전달되는 데이터는 버려진다. 수신 포트 변경을 위해 listen 함수를 호출한 경우 수신 포트가 바뀐 경우에만 true를 반환한다는 점을 명심해야 한다. 현재 포트 A가 수신 대기 상태이고 A.listen() 함수를 실행하면 수신 포트는 바뀌지 않으며 false를 반환한다. 따라서 포트를 바꾸기 전에는 현재 특정 포트가 수신 대기 상태에 있는지 검사하는 것이 안전하며 이를 위해 isListening 멤버 함수를 사용할 수 있다.

```
bool isListening(void)
```

- 매개변수 : 없음
- 반환값 : 해당 포트가 수신 대기 포트인지의 여부를 반환

이외에 데이터 송수신을 위한 멤버 함수들은 Serial 클래스와 동일하므로 Serial 클래스의 멤버 함수를 참고하도록 한다.

5. 시리얼 텍스트 LCD - 터미널 모드

시리얼 텍스트 LCD를 아두이노 우노와 소프트웨어 시리얼을 통해 연결해 보자. 전원 이외에 연결해야 하는 선은 시리얼 텍스트 LCD의 RX 뿐이다. 단, 그림 6의 모드 설정 점퍼를 터미널 모드로 설정해 주어야 한다는 점을 잊지 말아야 한다.

터미널 모드에서 시리얼 텍스트 LCD는 윈도우즈의 명령창과 거의 동일한 기능을 가지고 있다. 아스키 문자를 print 또는 println 함수로 출력할 수 있으며 자동으로 스크롤되므로 줄바꿈에 신경을 쓰지 않아도 된다. 강제로 줄바꿈을 하기 위해서는 Carriage Return ('\r', 0x0D) 문자를 전송하여 가능하다. 이외에 간단한 화면 지우기 명령, 커서 깜빡임 제어 명령, 백라이트 제어 명령 등을 지원하며 표 2은 터미널 모드에서 지원하는 명령어를 나타낸다. 명령어 실행을 위해서는 명령어 이후 줄바꿈 문자('\r')를 전달해야 한다.

명령어	의미
\$\$CS	화면 지움
\$\$B0	커서를 끄
\$\$B1	커서를 켜
\$\$L0	백라이트를 끄
\$\$L1	백라이트를 켜

표 2. 터미널 모드에서의 명령어

스케치 1은 시리얼 텍스트 LCD를 연결하고 1초 간격으로 카운트 값을 증가시켜 출력하는 예이다. 자동으로 스크롤 되면서 숫자가 출력되는지 확인해 보자.

스케치 1. 터미널 모드 - 자동 스크롤
<pre>#include <SoftwareSerial.h> // 2 : 시리얼 텍스트 LCD의 TX (사용되지 않음) // 3 : 시리얼 텍스트 LCD의 RX SoftwareSerial serialLCD = SoftwareSerial(2, 3); int count = 0; void setup(){</pre>

```

    serialLCD.begin(9600);                // 소프트웨어 시리얼 초기화
}

void loop(){
    serialLCD.print(count++);             // 카운트 증가 및 출력
    serialLCD.print(" ");
    delay(1000);
}

```

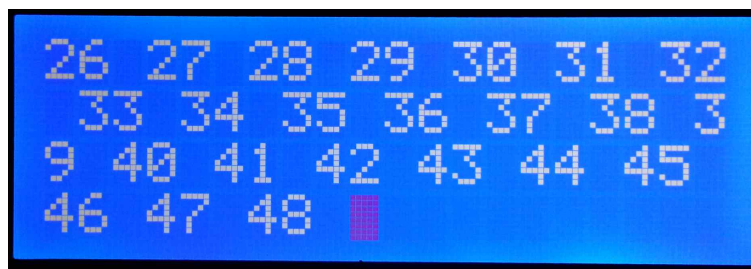


그림 7. 스케치 1 실행 결과

스케치 2는 두 번째 행에 현재 카운트 값을 출력하도록 스케치 1을 수정한 예이다. 동일한 위치에 문자를 출력하기 위해 화면 지우기 명령을 사용하였다.

스케치 2. 터미널 모드 - 고정 위치 출력

```

#include <SoftwareSerial.h>

// 2 : 시리얼 텍스트 LCD의 TX (사용되지 않음)
// 3 : 시리얼 텍스트 LCD의 RX
SoftwareSerial serialLCD = SoftwareSerial(2, 3);
int count = 0;

void setup(){
    serialLCD.begin(9600);                // 소프트웨어 시리얼 초기화
    serialLCD.print("$B0\r");             // 커서 끄
}

```

```
void loop(){
  serialLCD.print("$CS\r");           // 화면 지우기 명령 전달
  serialLCD.print('\r');              // 화면 지우기 명령 실행
  serialLCD.print("Current Count : ");
  serialLCD.print(count++);          // 카운트 증가 및 출력
  delay(1000);
}
```

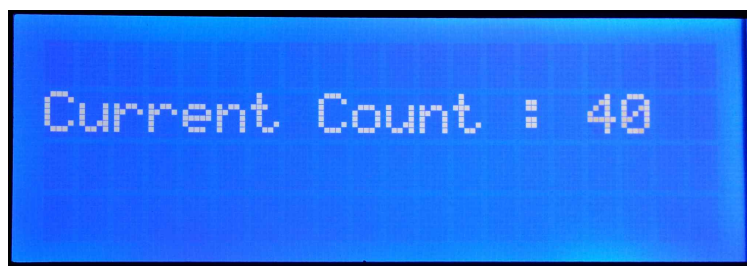


그림 8. 스케치 2 실행 결과

두 번째 행에 현재 카운트 값이 나타난다. 동작에 문제는 없어 보인다. 하지만 화면을 자세히 살펴보면 화면을 지우고 다시 쓰기 때문에 미세하게 깜빡이는 현상이 발생함을 확인할 수 있다. 이를 해결하기 위해서는 현재 카운트 값만을 출력하도록 커서를 특정 위치로 옮겨 출력하면 되지만 터미널 모드에서 커서 위치의 이동은 지원하지 않으며 커맨드 모드를 사용하여야 한다.

6. 시리얼 텍스트 LCD - 커맨드 모드

커맨트 모드는 명령어 기반의 모드로 터미널 모드에서의 가장 큰 차이는 항상 명령어로 시작한다는 점이다. 명령어는 '\$' 기호와 알파벳으로 구성되며 이후 콤마(,)로 구분되는 인자들이 나오고 마지막으로 터미널 모드에서와 마찬가지로 명령어 실행을 위해 줄바꿈 문자 '\r'이 나온다. 예를 들어, 터미널 모드에서 문자열 "ABC" 출력을 위해서는 print("ABC"); 문장이 필요하다. 반면 커맨드 모드에서는 print("\$T,ABC\r"); 문장을 실행시켜야 한다. 복잡해 보일 수도 있지만 커맨드 모드에서는 터미널 모드에서 지원하지 않는 여러 제어 기능을 사용할 수 있으므로 보다 유용하게 사용할 수 있다. 표 3은 커맨트 모드에서 지원하는 명령어를 요약한 것이다.

명령	명령어	데이터1	데이터2	예
초기화	\$I			\$I\r
화면 지우기	\$C			\$C\r
커서 위치 설정	\$G	행 (1~4)	열 (1~20)	\$G,1,1\r
문자열 출력	\$T	문자열		\$T,Test\r
커서 끄기/켜기	\$B	0/1		\$B,0\r
LCD 끄기/켜기	\$D	0/1		\$D,1\r
백라이트 끄기/켜기	\$L	0/1		\$L,0\r
화면 왼쪽/오른쪽 이동	\$S	L/R		\$S,L\r

표 3. 커맨드 모드에서의 명령어

스케치 3은 스케치 2와 동일한 기능을 하는 코드를 커맨드 모드에서 작성한 예로 실행 결과는 그림 8과 동일하다. 하지만 스케치 2를 실행시킨 것과 비교할 때 화면을 전체적으로 지우지 않으므로 깜빡임이 줄어드는 것을 확인할 수 있다.

스케치 3. 커맨드 모드 - 고정 위치 출력

```
#include <SoftwareSerial.h>

// 2 : 시리얼 텍스트 LCD의 TX (사용되지 않음)
// 3 : 시리얼 텍스트 LCD의 RX
SoftwareSerial serialLCD = SoftwareSerial(2, 3);
int count = 0;

void setup(){
    serialLCD.begin(9600);                // 소프트웨어 시리얼 초기화
    serialLCD.print("$B,0\r");             // 커서 끄
    serialLCD.print("$G,2,1\r");           // 2행 1열로 커서를 옮겨 문자열 출력
    serialLCD.print("$T,Current Count : \r");
}

void loop(){
    serialLCD.print("$G,2,17\r");          // 카운트 값 출력 위치로 커서 옮김
```

```
serialLCD.print("$T,");           // 문자열 출력 명령  
serialLCD.print(count++);  
serialLCD.print('\r');  
delay(1000);  
}
```