

12. 텍스트 LCD

1. 액정 (Liquid Crystal)

액정은 액체와 고체의 중간 형태로 1800년대 후반에 처음 발견되어 1960년대 후반에 이르러 표시 장치로 사용되기 시작하였다. LCD를 이용한 표시 장치에는 문자 단위의 정보를 표시할 수 있는 텍스트 LCD, 픽셀 단위의 정보를 표시할 수 있어 그래픽까지도 표현이 가능한 그래픽 LCD, 컬러 표현이 가능한 컬러 LCD, 터치 모듈이 포함되어 있는 터치 LCD 등 그 종류가 다양하며 대부분을 아두이노에서 사용이 가능하다. 이 중 텍스트 LCD는 고정된 위치에 영문자, 숫자, 기호 등을 문자 단위로 표시할 수 있어 간단한 메시지 출력을 위해 흔히 사용된다. 텍스트 LCD는 도형이나 한글의 표시가 힘들고 표시할 수 있는 문자의 수가 일반적으로 32개로 제한되는 등의 한계가 있다. 따라서 다양한 정보와 색상을 표현할 수 있는 그래픽 LCD나 컬러 LCD 사용이 증가하는 추세이지만 이들 LCD의 경우에는 연결을 위해 필요한 핀의 개수가 많아 입출력 핀의 개수가 제한된 아두이노에서는 사용이 용이하지 않을 수 있으므로 간단한 메시지 표시를 위해서는 여전히 텍스트 LCD가 많이 사용된다. 텍스트 LCD 사용을 위해 아두이노에서는 전용 라이브러리인 LiquidCrystal 라이브러리를 기본 라이브러리의 하나로 제공하고 있다.

2. 텍스트 LCD

텍스트 LCD는 표시할 수 있는 문자의 수에 따라 여러 종류가 존재하지만 이 장에서 사용할 텍스트 LCD는 흔히 볼 수 있는 16글자씩 2줄, 총 32글자를 표시할 수 있는 텍스트 LCD를 사용한다.



그림 1. 텍스트 LCD¹⁾

텍스트 LCD는 16핀 인터페이스를 가지며 이 중 3개는 제어 신호 핀이며 8개는 데이터 신호 핀이다. 나머지 5개는 전원 및 백라이트 제어를 위한 핀에 해당한다. 3개의 제어 신호는 표 1과 같다.

제어 신호	설명
RS (Register Select)	LCD를 제어하기 위해서는 두 개의 레지스터가 필요하며 제어 레지스터와 데이터 레지스터가 그것들이다. RS 신호는 명령을 담고 있는 레지스터와 데이터를 담고 있는 레지스터 중 하나를 선택하기 위해 사용한다.
R/W (Read/Write)	읽기 및 쓰기 모드를 선택한다. 일반적으로 LCD는 데이터를 쓰기위한 용도로만 사용하므로 R/W 신호에 해당하는 핀은 GND시킨다.
E (Enable)	준비된 데이터를 LCD로 전송을 시작하기 위해 사용한다.

표 1. 텍스트 LCD 제어 신호

데이터 핀은 8개가 있지만 텍스트 LCD는 4비트 모드와 8비트 모드로 동작할 수 있으며 핀의 수가 제한된 아두이노의 경우 4비트 모드를 주로 사용하므로 4개의 데이터 핀만을 사용하고 나머지 4개는 사용하지 않는 것이 일반적이다. V_{SS} 는 GND에 연결시키고, V_{CC} 는 5V에 연결시키며, V_0 는 가변저항을 통해 LCD의 콘트라스트 조절을 위해 사용된다. 나머지 LED+와 LED-는 백라이트 전원에 해당한다. 따라서 텍스트 텍스트 LCD를 아두이노에 연결시키기 위해서는 4개의 데이터 선과 2개의 제어선(R/W는 사용하지 않으므로 제외), 총 6개 선의 연결이 필요하다. 그림 2는 텍스트 LCD를 아두이노 우노에 연결시키는 회로도를 나타내며 표 2에 아두이노와 텍스트 LCD 연결을 위한 핀 정보를 나타내었다.

1) 이미지 출처: <http://www.newtc.co.kr>

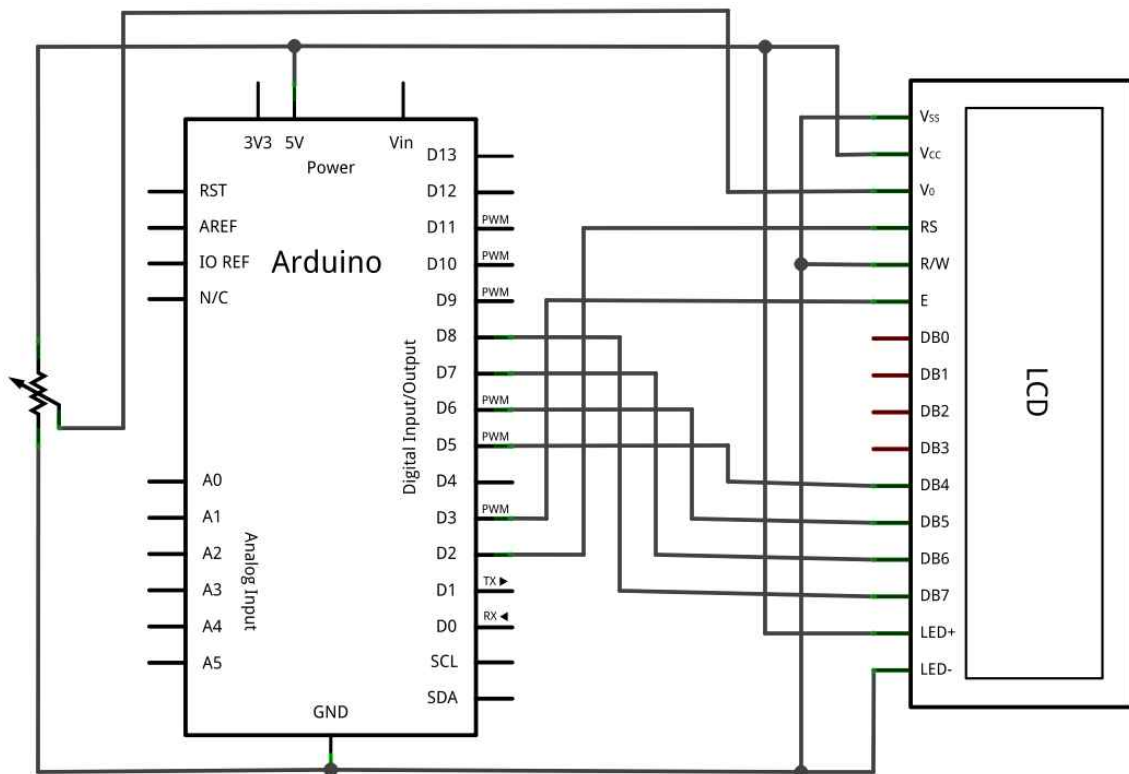


그림 2. 텍스트 LCD 연결을 위한 회로도

핀 설명	아두이노 핀 번호
RS	디지털 2번
E	디지털 3번
DB4 (데이터 4)	디지털 5번
DB3 (데이터 3)	디지털 6번
DB2 (데이터 2)	디지털 7번
DB1 (데이터 1)	디지털 8번

표 2. 텍스트 LCD 연결 핀

3. LiquidCrystal 라이브러리

아두이노에는 텍스트 LCD 사용을 지원하기 위해 기본 라이브러리 중 하나로 LiquidCrystal 라이브러리를 포함하고 있어 텍스트 LCD 연결은 까다로운 반면 사용법은 간단하다. 텍스트 LCD를 사용하기 위해서는 먼저 ‘스케치 → 라이브러리 가져오기... → LiquidCrystal’ 메뉴를 통해 LiquidCrystal 라이브러리 사용을 위한 헤더 파일을 추가한다. 메뉴를 선택하면

LiquidCrystal.h 파일이 자동으로 추가된다.

```
#include <LiquidCrystal.h>
```

헤더 파일이 추가되면 LiquidCrystal 클래스의 객체를 생성한다. 객체를 생성할 때는 아두이노와 연결되는 6개의 핀 번호를 지정한다.

```
// 핀 번호 (RS, E, DB4, DB5, DB6, DB7)
LiquidCrystal lcd(2, 3, 5, 6, 7, 8);
```

다음은 begin 함수로 텍스트 LCD를 초기화한다. 초기화 과정에서는 텍스트 LCD의 크기와 글자 크기를 지정할 수 있다. 글자 크기는 5×10 크기(LCD_5x10DOTS) 또는 5×8 크기(LCD_5x8DOTS) 중 선택할 수 있고 디폴트는 5×8 크기이다.

```
void begin(uint8_t cols, uint8_t rows, uint8_t charsize = LCD_5x8DOTS);
매개변수
  cols : 열의 개수
  rows : 행의 개수
  charsize : 글자 크기
반환값 : 없음
```

이제 설정이 끝났으니 문자를 출력해 보자. 문자 출력을 위해서는 write 함수를, 문자열 출력을 위해서는 print 함수를 사용하면 된다. 출력되는 문자는 좌상단이 (0, 0)으로 기준점이 되며 현재 커서의 위치에 문자가 출력된다. 스케치 1은 텍스트 LCD를 초기화하고 좌상단에서 시작하여 'Hello World!'를 출력하는 스케치의 예이다.

스케치 1. Hello World!

```
#include <LiquidCrystal.h>

// 핀 번호 (RS, E, DB4, DB5, DB6, DB7)
LiquidCrystal lcd(2, 3, 5, 6, 7, 8);      // LCD 연결

void setup(){
  lcd.begin(16, 2);                      // LCD 초기화
  lcd.print("Hello World!");             // 문자열 출력
}

void loop(){
}
```



그림 3. 스케치 1 실행 결과

4. 임의의 위치에 문자 나타내기

텍스트 LCD는 현재 커서의 위치를 기준으로 문자가 표시된다. 임의의 위치에 문자를 출력하고 싶다면 커서의 위치를 이동시켜야 하며 `setCursor` 함수를 통해 가능하다.

```
void setCursor(uint8_t col, uint8_t row)
```

매개변수

col : 열 위치

row : 행 위치

반환값 : 없음

랜덤 함수를 사용하여 알파벳 대문자 중 임의의 문자를 생성하고, 표시 위치도 임의의 위치를 생성하여 표시해 보자. 랜덤 함수를 사용하기 위해서는 randomSeed 함수를 통해 의사 난수 생성기를 초기화 시켜야 하며 이를 위해 A4 또는 A5 핀으로부터 값을 읽어서 사용한다. LiquidCrystal 클래스의 clear 함수는 현재 텍스트 LCD을 지우는 명령이다.

```
void clear(void)
```

매개변수 : 없음

반환값 : 없음

스케치 2는 임의의 알파벳 대문자를 생성하고 이를 임의의 위치에 출력하는 스케치의 예이다.

스케치 2. 임의의 위치에 임의의 문자 나타내기

```
#include <LiquidCrystal.h>
```

```
// 핀 번호 (RS, E, DB4, DB5, DB6, DB7)
```

```
LiquidCrystal lcd(2, 3, 5, 6, 7, 8); // LCD 연결
```

```
void setup(){
```

```
  lcd.begin(16, 2); // LCD 초기화
```

```
  lcd.clear(); // LCD 화면 지우기
```

```
  randomSeed(analogRead(A5)); // 난수 생성기 초기화
```

```
  Serial.begin(9600);
```

```
}
```

```
void loop(){
```

```
int row = random(0, 2);           // 행 번호 생성
int col = random(0, 17);          // 열 번호 생성

char ch = random(0, 27);          // 출력 문자 생성
if(ch == 0) ch = ' ';             // 공백
else ch = ch + 'A' - 1;           // 대문자

lcd.setCursor(col, row);          // 커서 이동
lcd.write(ch);                    // 문자 출력

// 시리얼 모니터로 정보 출력
Serial.println(String("(" + row + ", " + col + ") " + ch);

delay(100);
}
```



그림 4. 스케치 2 실행 결과 - 텍스트 LCD 창

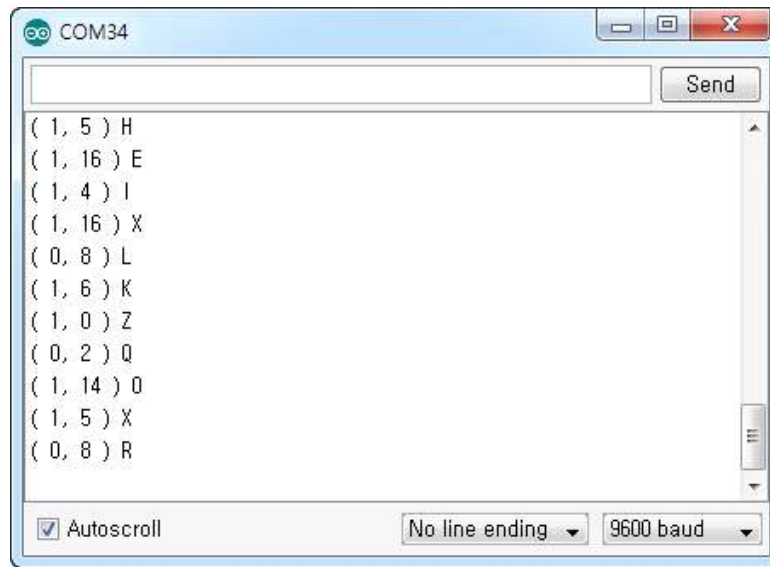


그림 5. 스케치 2 실행 결과 - 시리얼 모니터

5. 내용 스크롤 하기

텍스트 LCD의 한 화면에는 16글자씩 2줄로 32글자만 보이지만 실제 텍스트 LCD의 버퍼에는 더 많은 수의 문자들이 저장될 수 있고 출력 범위를 벗어나는 글자는 화면에 보이지 않을 뿐이다. LCD의 버퍼는 한 줄에 40글자의 내용을 저장할 수 있으며 초기값으로 0~15열까지의 문자만을 화면에 출력한다.

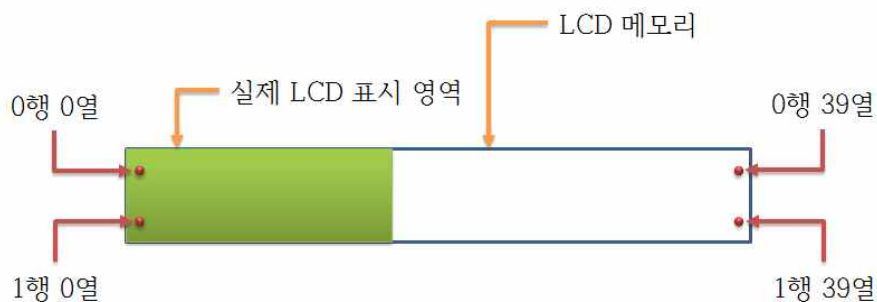


그림 6. 텍스트 LCD의 버퍼 구조

LiquidCrystal 클래스의 멤버 함수 중 `scrollDisplayRight`와 `scrollDisplayLeft` 함수는 LCD의 버퍼에 있는 내용을 한 칸씩 좌우로 이동시켜서 보여주는 역할을 하므로 이를 이용하면 전광판에서 흔히 볼 수 있는 글자가 스크롤 되는 효과를 얻을 수 있다. 실험에 사용한 텍스트 LCD 모듈의 반응 속도가 빠르지 않으므로 이동 시간을 너무 짧게 설정하면 글자가 겹쳐지고

번져 보이므로 충분한 시간을 주도록 한다. 스케치 3은 문자열이 스크롤 되도록 한 스케치의 예이다.

스케치 3. 문자열 스크롤

```
#include <LiquidCrystal.h>
#define DELAY_TIME    500           // 스크롤 간격 0.5초

// 핀 번호 (RS, E, DB4, DB5, DB6, DB7)
LiquidCrystal lcd(2, 3, 5, 6, 7, 8); // LCD 연결

void setup(){
    lcd.begin(16, 2);                // LCD 초기화
    lcd.clear();                     // 화면 지우기
    lcd.print("Scrolling Text");     // 첫 번째 행 내용
    lcd.setCursor(0, 1);
    lcd.print("1234567890");         // 두 번째 행 내용
}

void loop(){
    for(int i = 1; i <= 10; i++){
        lcd.scrollDisplayRight();    // 오른쪽으로 이동
        delay(DELAY_TIME);
    }
    for(int i = 1; i <= 10; i++){
        lcd.scrollDisplayLeft();     // 왼쪽으로 이동
        delay(DELAY_TIME);
    }
}
```

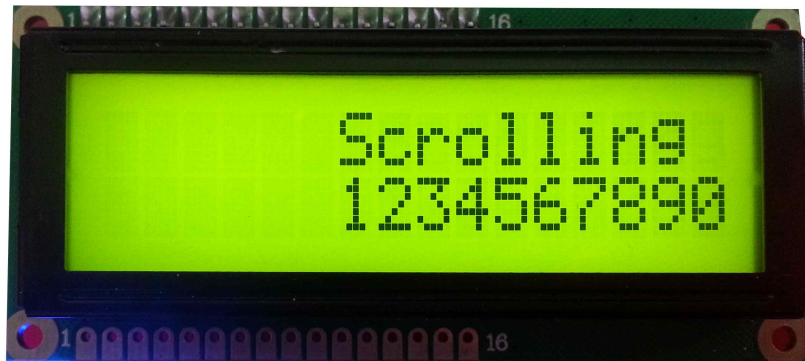


그림 7. 스케치 3 실행 결과

6. 사용자 정의 문자 출력

텍스트 LCD는 알파벳이나 숫자 등 제한적인 문자만을 표시할 수 있고 한글은 표시할 수 없다. 하지만 사용자 정의 문자를 정의하고 이를 출력할 수는 있다. 텍스트 LCD에 사용자 정의 문자를 출력하기 위해서는 먼저 문자를 정의하여야 한다. 사용자 정의 문자는 8바이트의 배열로 정의되며 출력되는 문자의 크기 5×8 에 맞게 하위 5비트만을 사용한다. 그림 8은 사용자 문자를 정의한 예를 보여준다.

					00000
					10001
					00000
					00000
					10001
					01110
					00000
					00000

그림 8. 스마일 문자 정의

배열 데이터로부터 문자 생성을 위해서는 `createChar` 함수를 사용하고, LCD에 출력하기 위해서는 `write(byte(n))`과 같이 사용한다. 이 때 정의할 수 있는 문자는 최대 8개로 `n`은 0에서 7 사이의 값을 가질 수 있다.

```
void createChar(uint8_t location, uint8_t charmap[])
```

매개변수

location : 사용자 문자 번호 ($0 \leq \text{location} \leq 7$)

charmap : 사용자 문자 데이터

반환값 : 없음

스케치 4는 두 개의 사용자 정의 문자를 정의하고 이를 LCD 화면에 표시하는 예이다.

스케치 4. 사용자 정의 문자

```
#include <LiquidCrystal.h>
```

```
// 핀 번호 (RS, E, DB4, DB5, DB6, DB7)
```

```
LiquidCrystal lcd(2, 3, 5, 6, 7, 8);          // LCD 연결
```

```
// 사용자 정의 문자 데이터
```

```
byte user1[8] = {
```

```
    B00000, B10001, B00000, B00000, B10001, B01110, B00000, B00000
```

```
};
```

```
byte user2[8] = {
```

```
    B00000, B10001, B01010, B10001, B00000, B10001, B01110, B00000
```

```
};
```

```
void setup() {
```

```
    lcd.createChar(0, user1);          // 사용자 정의 문자 1 생성
```

```
    lcd.createChar(1, user2);          // 사용자 정의 문자 2 생성
```

```
    lcd.begin(16, 2);                  // LCD 크기 설정
```

```
    lcd.clear();                       // LCD 화면 지우기
```

```
    lcd.write(byte(0));                // 사용자 정의 문자 1 출력
```

```
    lcd.setCursor(0, 1);               // 두 번째 행으로 이동
```

```
    lcd.write(byte(1));                // 사용자 정의 문자 2 출력
```

```
}
```

```
void loop() {  
}
```



그림 9. 스케치 4 실행 결과

7. 가변저항 값 출력하기

가변저항을 아날로그 3번 A3에 연결하고 가변저항의 값을 LCD 창에 출력해 보자. 값이 출력되는 위치는 (0, 0)부터 (3, 0) 위치에 표시되도록 하며 값은 오른쪽으로 정렬되어 표시되도록 한다.

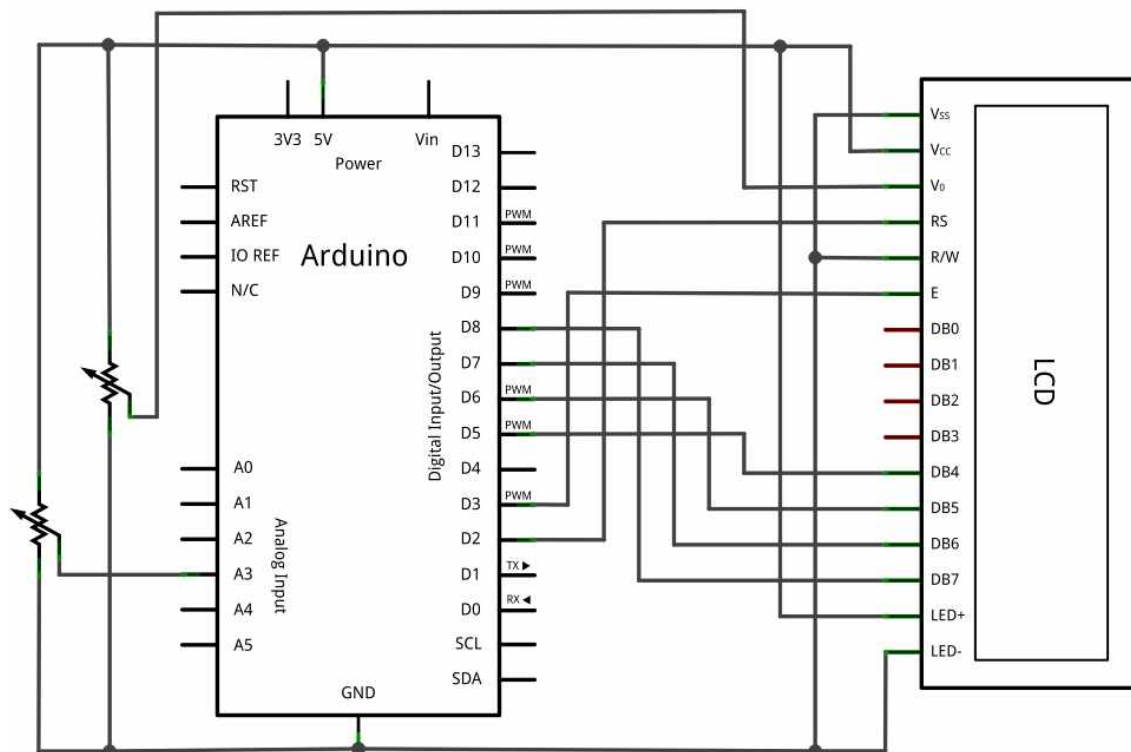


그림 10. 가변저항과 텍스트 LCD 연결 회로도

문자열을 오른쪽으로 정렬하기 위해서는 현재 표시될 문자의 길이를 알아내고 이에 따라 앞 부분에 표시할 공백의 수를 결정하여야 한다. 문자열의 길이를 알아내기 위해서는 String 클래스의 length 함수를 사용하면 된다.

```
unsigned int length(void)
```

매개변수 : 없음

반환값 : 문자열의 길이

텍스트 LCD에서 문자는 항상 현재 커서의 위치에 표시된다. LiquidCrystal 클래스의 clear 함수를 사용하면 화면 전체가 지워지고 커서는 (0, 0)의 위치로 옮겨 가지만 텍스트 LCD의 느린 반응 속도로 인해 깜빡임이 심해진다. 따라서 home 함수를 사용하여 커서만을 (0, 0) 위치로 옮기고 공백을 출력하여 이전에 출력된 문자를 지우도록 한다.

```
void home(void)
    매개변수 : 없음
    반환값 : 없음
```

스케치 5는 텍스트 LCD 창에 가변저항의 값을 오른쪽으로 정렬하여 출력하는 스케치의 예이다. 가변저항을 돌리면서 값이 변화하는 지 살펴보자.

스케치 4. 가변저항 값 표시하기

```
#include <LiquidCrystal.h>

// 핀 번호 (RS, E, DB4, DB5, DB6, DB7)
LiquidCrystal lcd(2, 3, 5, 6, 7, 8);      // LCD 연결

void setup() {
    lcd.begin(16, 2);                     // LCD 크기 설정
    lcd.clear();                          // LCD 화면 지우기
}

void loop() {
    int value = analogRead(A3);           // 가변저항 값 읽기
    String value_string = String(value);   // 가변저항 값을 문자열로 변환

    lcd.home();                           // 커서를 (0, 0)으로 옮김
    // 우측 정렬을 위해 공백 출력
    for(int i = 0; i < 4 - value_string.length(); i++){
        lcd.write(' ');
    }

    lcd.print(value);                      // 가변저항 값 출력

    delay(100);
}
```



그림 11. 스케치 4 실행 결과

스케치 4를 수정하여 텍스트 LCD에 가변저항의 값에 따라 바(bar)가 나타나도록 해 보자. 출력할 문자는 '#'로 하여 0에서 1024까지의 값을 0에서 16까지의 값으로 변환한 후 해당하는 개수의 문자를 출력하면 된다. 이 때 이전에 출력된 문자를 공백으로 지워주지 않으면 값이 감소하더라도 바가 줄어들지 않는 경우가 발생하므로 주의하여야 한다.

스케치 5. 가변저항 바 표시하기

```
#include <LiquidCrystal.h>

// 핀 번호 (RS, E, DB4, DB5, DB6, DB7)
LiquidCrystal lcd(2, 3, 5, 6, 7, 8);      // LCD 연결

void setup() {
    lcd.begin(16, 2);                    // LCD 크기 설정
    lcd.clear();                         // LCD 화면 지우기
}

void loop() {
    int value = analogRead(A3);          // 가변저항 값 읽기
    // 0에서 16칸으로 바의 크기 계산
    int count = map(value, 0, 1023, 0, 16);

    lcd.home();                          // 커서를 (0, 0)으로 옮김
    for(int i = 0; i < count; i++){      // 바 표시
        lcd.write('#');
```

```
    }  
    for(int i = count; i < 16; i++){           // 공백 표시  
        lcd.write(' ');  
    }  
  
    delay(100);  
}
```

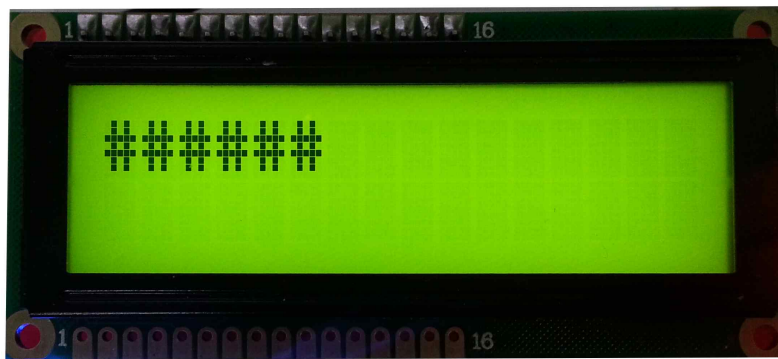


그림 12. 스케치 5 실행 결과

8. 타이머

디지털 9번 핀에 풀다운 저항을 통해 버튼을 연결하자. 텍스트 LCD에는 “00M 00S” 형식으로 경과시간을 표시한다. 시작 시에는 타이머가 동작하지 않는 상태로 하고, 버튼이 눌러지면 초단위로 경과 시간을 텍스트 LCD에 표시한다. 다시 버튼이 눌러지면 타이머는 정지하고 현재 상태를 유지하도록 한다.

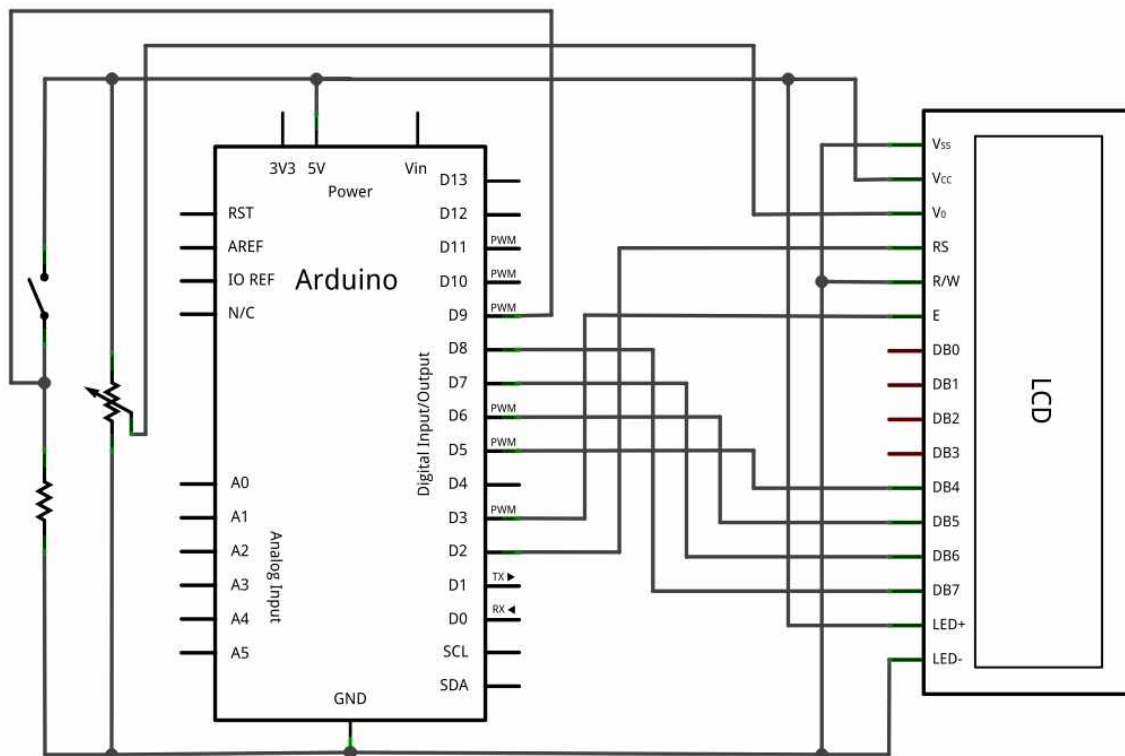


그림 13. 버튼과 텍스트 LCD 연결 회로도

스케치 6. 타이머

```
#include <LiquidCrystal.h>

// 핀 번호 (RS, E, DB4, DB5, DB6, DB7)
LiquidCrystal lcd(2, 3, 5, 6, 7, 8);           // LCD 연결

int button_pin = 9;                           // 버튼 연결 핀
boolean going = false;                       // 정지 상태에서 시작

unsigned long time_previous, time_current;
int INTERVAL = 1000;                         // 초 단위

int minutes = 0, seconds = 0;                // 분과 초 저장

void setup() {
    lcd.begin(16, 2);                         // LCD 크기 설정
```

```
lcd.clear();                                // LCD 화면 지우기

time_previous = millis();
pinMode(button_pin, INPUT);

lcd.print("00M 00S");                       // 초기 상태
}

void loop() {
  boolean pressed = digitalRead(button_pin);    // 버튼 입력
  if(pressed){
    delay(500);                                // 디바운스
    going = !going;                            // 시작과 정지 토글
  }
  else{
    if(!going) time_previous = millis();        // 정지 상태에서 이전 시간 갱신
  }

  if(going){                                   // 시작 상태인 경우
    time_current = millis();

    if(time_current - time_previous >= INTERVAL){
      time_previous = time_current;

      seconds++;                               // 초 증가
      if(seconds == 60){                       // 60초이면 분 증가
        seconds = 0;
        minutes++;

        if(minutes == 60) minutes = 0;        // 60분이면 0으로 복귀
      }

      lcd.clear();
      if(minutes < 10) lcd.write('0');        // 한 자리 숫자인 경우 '0' 추가
      lcd.print(minutes);
      lcd.print("M ");
```

```
if(seconds < 10) lcd.write('0');    // 한 자리 숫자인 경우 '0' 추가  
lcd.print(seconds);  
lcd.write('S');  
}  
}  
}
```



그림 14. 스케치 6 실행 결과