
Chapter 119. 텍스트 LCD

1. 텍스트 LCD

액정(liquid crystal)은 액체이면서도 고체의 설질을 갖는 액체와 고체의 중간 형태로 1800년대 후반 처음 발견되어 1960년대 후반에 이르러 디스플레이로 활용되기 시작하였다. LCD를 사용한 디스플레이 장치로는 문자 기반의 텍스트 LCD, 픽셀 기반의 그래픽 LCD, 컬러 표현이 가능한 컬러 LCD 등 다양한 장치가 사용되고 있으며 이 장에서 사용할 LCD는 이 중 가장 간단한 영문자 또는 숫자 표현이 가능한 텍스트 LCD이다. 텍스트 LCD도 표시할 수 있는 문자의 수에 따라 여러 종류가 존재하지만 2줄 16글자, 총 32 글자를 표시할 수 있는 텍스트 LCD를 흔히 볼 수 있다.

텍스트 LCD에는 LCD 화면 출력을 제어하기 위해 LCD 드라이버 (또는 컨트롤러)가 포함되어 있다. 텍스트 LCD에는 히다찌(Hitachi)의 HD44780 또는 삼성의 KS0066 드라이버가 주로 사용되며 이들은 서로 호환된다. LCD 드라이버는 명령 레지스터(Instruction Register, IR)와 데이터 레지스터(Data Register, DR)에 저장된 정보를 바탕으로 화면을 제어하며 이들 레지스터는 외부에 연결된 마이크로컨트롤러에서 직접 데이터를 기록할 수 있다. 따라서 이들 레지스터에 외부의 마이크로컨트롤러가 제어 정보를 기록하고 활성화 신호인 Enable에 의해 드라이버의 내부 동작을 제어할 수 있다. 이 장에서는 그림 1과 같이 16핀의 연결 커넥터를 가지는 텍스트 LCD를 2행으로 표시하는 방법을 알아본다.

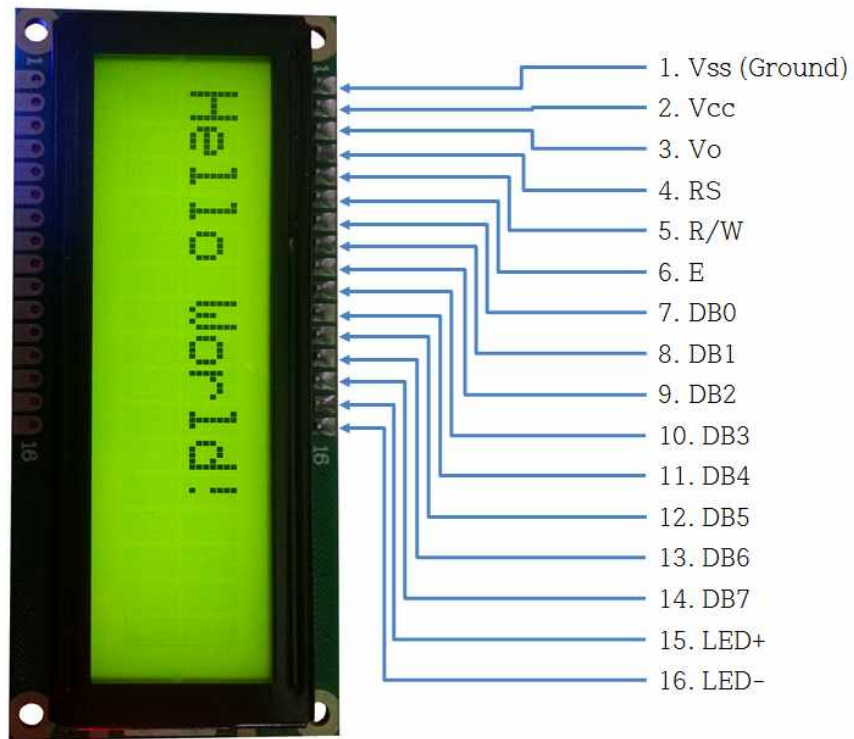


그림 1. 텍스트 LCD

각 핀의 설명은 표 1과 같다.

핀 번호	이름	설명
1	V _{SS}	그라운드 (GND)
2	V _{DD}	5V 동작 전원
3	V ₀	LED 전원으로 가변저항을 통해 0 ~ 5V 사이 입력
4	RS	레지스터 선택 (Register Select)
5	R/W	읽기 쓰기 (Read/Write)
6	E	활성화 (Enable)
7	DB0	데이터 신호 핀
8	DB1	
9	DB2	
10	DB3	
11	DB4	
12	DB5	
13	DB6	
14	DB7	
15	LED+	백라이트 전원
16	LED0	

표 1. 텍스트 LCD의 핀

마이크로컨트롤러와의 연결은 3핀의 제어 신호 연결이 필요하고, 8핀의 데이터 신호 연결을 제외한 나머지 5개 핀은 전원 관련 핀이다. 제어를 위해 사용되는 3개의 핀은 표 2과 같다.

제어 핀	설명
RS (Register Select)	텍스트 LCD를 제어하기 위해서는 제어 레지스터와 데이터 레지스터의 두 개 레지스터가 사용되며, RS 신호는 명령을 담고 있는 레지스터(RS = LOW)와 데이터를 담고 있는 레지스터(RS = HIGH) 중 하나를 선택하기 위해 사용한다.
R/W (Read/Write)	읽기 (R/W = HIGH) 및 쓰기 (R/W = LOW) 모드 선택을 위해 사용한다. 일반적으로 LCD는 데이터를 쓰기위한 용도로만 사용하므로 R/W 신호에 해당하는 핀은 GND에 연결한다.
E (Enable)	하강 에지(falling edge)에서 LCD 드라이버가 내부 처리를 시작하도록 하기 위한 신호로 사용된다.

표 2. 텍스트 LCD의 제어 핀

텍스트 LCD는 4비트 모드 또는 8비트 모드로 사용될 수 있으므로 R/W를 제외한다면 4비트 모드의 경우 6개, 8비트 모드의 경우 10개의 연결이 필요하다. 텍스트 LCD 모듈에 명령을 전달하기 위해서는 먼저 RS와 R/W 값을 설정하고 E를 HIGH 상태로 바꾼다. 이후 데이터 레지스터에 데이터를 기록한 후 E를 LOW 상태로 바꾸면 해당 명령이 실행된다. 텍스트 LCD 모듈 제어를 위한 명령어는 표 3과 같다.

명령	명령 코드										설명
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
Clear Display	0	0	0	0	0	0	0	0	0	1	공백문자(코드 0x20) 문자로 화면을 지우고 커서를 홈 위치(주소 0번)로 이동시킨다.
Return Home	0	0	0	0	0	0	0	0	1	-	커서를 홈 위치로 이동시키고, 표시 영역이 이동된 경우 초기 위치로 이동시킨다. 화면에 출력된 내용(DDRAM의 값)은 변하지 않는다.
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	데이터 읽기 또는 쓰기 후 메모리의 증가(Increment) 또는 감소(Decrement) 방향을 지정한다. DDRAM에서 I/D = 1이면 커서를 오른쪽으로, I/D = 0이면 왼쪽으로 옮긴다. S = 1이면 I/D 값에 따라 디스플레이를 왼쪽 또는 오른쪽으로 옮기며 이 때 커서는 고정된 위치에 나타난다.
Display on/off Control	0	0	0	0	0	0	1	D	C	B	디스플레이(D), 커서(C), 커서 깜빡임(B)의 on/off를 설정한다.
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	화면에 출력된 내용의 변경 없이 커서와 화면을 이동시킨다. (표 4)
Function Set	0	0	0	0	1	DL	N	F	-	-	데이터비트 크기(DL = 1이면 8비트, DL = 0이면 4비트), 디스플레이의 행 수(N), 폰트 크기(F)를 설정한다. (표 5)
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	주소 카운터에 CGRAM 주소를 설정한다.
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	주소 카운터에 DDRAM 주소를 설정한다.
Read Busy Flag & Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	드라이버에서 현재 명령어를 실행 중인지의 여부를 나타내는 동작 중 플래그(Busy Flag) 값과 주소 카운터의 값을 읽어온다.
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	RAM(DDRAM 또는 CGRAM)에 데이터를 기록한다.
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	RAM(DDRAM 또는 CGRAM)에서 데이터를 읽어온다.

표 3. 텍스트 LCD 모듈의 명령어

S/C	R/L	설명
0	0	커서를 왼쪽으로 옮긴다. 메모리 주소는 1 감소한다.
0	1	커서를 오른쪽으로 옮긴다. 메모리 주소는 1 증가한다.
1	0	화면을 왼쪽으로 옮긴다. 커서 역시 왼쪽으로 이동한다.
1	1	화면을 오른쪽으로 옮긴다. 커서 역시 오른쪽으로 이동한다.

표 4. 커서와 화면 이동

N	F	행 수	폰트 크기	비고
0	0	1	5×8	
0	1	1	5×10	
1	-	2	5×8	2행 출력에서는 5×10 크기 폰트를 사용할 수 없다.

표 5. 행 수와 폰트 크기

명령어 표에서 알 수 있듯이 텍스트 LCD에는 DDRAM(Display Data RAM)과 CGRAM(Character Generator RAM) 두 종류의 메모리가 포함되어 있다. DDRAM은 현재 화면에 표시 중인 데이터를 저장하는 RAM으로 최대 80개 문자를 출력할 수 있다. 하지만 문자를 2행으로 화면에 표시하는 경우 나타나는 문자는 최대 32개이며 화면 이동 명령에 의해 표시 영역은 변경할 수 있다. 1행으로 문자를 표시하는 경우 ($N = 0$) DDRAM의 주소는 0x00에서 0x4F(= 79)로 정해지는 반면 2행으로 문자를 표시하는 경우 ($N = 1$) 1행의 DDRAM 주소는 0x00에서 0x27(= 39)까지, 2행의 주소는 0x40(= 64)에서 0x67(= 103)까지로 행별로 40글자씩 분리되어 정해진다.

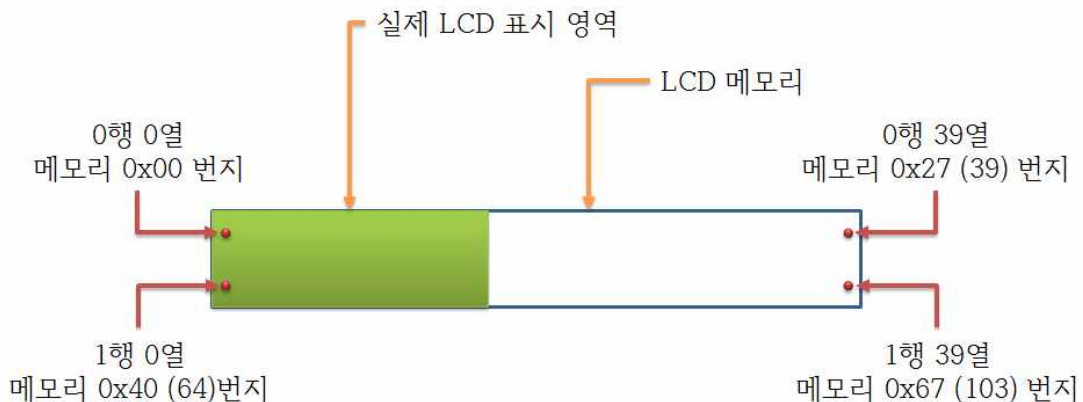


그림 2. LCD 메모리 및 화면 표시 영역

CGRAM은 사용자 정의 문자를 정의하여 저장하는 공간으로 64바이트 크기를 가지고 8개의 사용자 정의 문자를 정의하여 사용할 수 있다. 텍스트 LCD에 2행으로 문자를 출력하기 위해 필요한 초기화하는 과정은 다음과 같다.

1. Function Set 명령으로 데이터 비트 크기 (DL = 1이면 8비트, DL = 0이면 4비트)를 지정하고, 디스플레이의 행 수는 2로 (N = 1), 폰트 크기는 5×8로 (F = 0) 설정한다.
2. Display on/off Control 명령으로 디스플레이를 끄고 (D = 0), 커서는 표시하지 않으며 (C = 0), 커서 깜빡임도 없도록 (B = 0) 설정한다.
3. Clear Display 명령을 실행한다.
4. Entry Mode Set 명령으로 문자 출력 후 메모리 번지는 증가하고 (I/D = 1), 표시 영역의 이동은 없도록 (S = 0)으로 설정한다.

4비트 모드를 사용하는 경우에는 8비트의 데이터를 두 번에 나누어서 전달하여야 하며 이 때 상위 니블(nibble)을 먼저 전송하여야 한다.

2. 8비트 모드 텍스트 LCD 제어

8비트 모드로 텍스트 LCD를 제어하는 방법을 알아보자. 먼저 텍스트 LCD를 그림 3과 같이 연결한다. 데이터 핀은 PORTD에, RS 핀은 PB0, E 핀은 PB1에 연결하였다. 텍스트 LCD를 출력 용도로만 사용하기 위해 R/W 핀은 GND에 연결하였다.

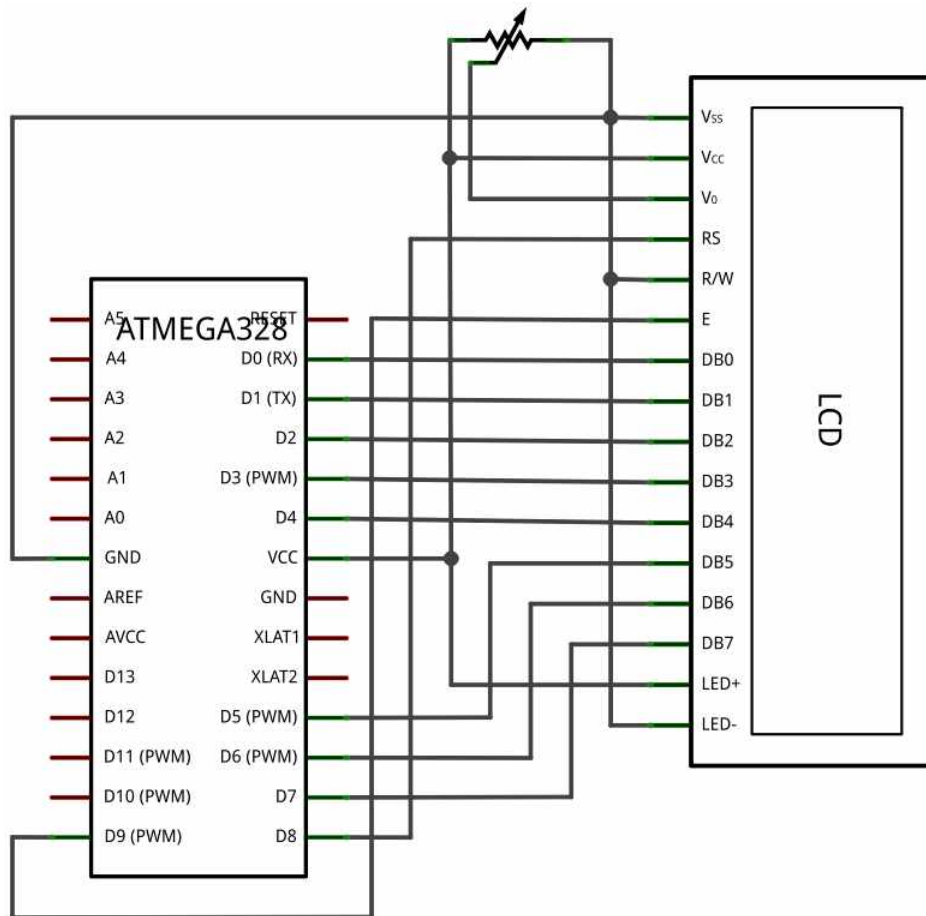


그림 3. 8비트 모드 텍스트 LCD 연결 회로도

데이터 핀과 제어 핀을 다음과 같이 정의하자. 사용하고자 하는 핀 번호가 바뀌는 경우에는 정의된 값을 변경해 주어야 한다.

```
#define PORT_DATA          PORTD          // 데이터 핀 연결 포트
#define PORT_CONTROL      PORTB          // 제어 핀 연결 포트
#define DDR_DATA          DDRD           // 데이터 핀의 데이터 방향
#define DDR_CONTROL       DDRB           // 제어 핀의 데이터 방향

#define RS_PIN            0              // RS 제어 핀의 비트 번호
#define E_PIN             1              // E 제어 핀의 비트 번호
```

표 3을 참고하여 화면을 지우는 명령과 8비트 모드로 설정하는 명령어는 다음과 같이 정의될

수 있다. 이후 사용을 위해 4비트 모드를 설정하는 명령어도 함께 정의하였다.

```
#define COMMAND_CLEAR_DISPLAY      0x01
#define COMMAND_8_BIT_MODE         0x38    // 8비트, 2라인, 5x8 폰트
#define COMMAND_4_BIT_MODE         0x28    // 4비트, 2라인, 5x8 폰트
```

화면, 커서, 커서 깜빡임 등을 제어하기 위해 각 기능에 해당하는 비트 번호도 정의한다.

```
#define COMMAND_DISPLAY_ON_OFF_BIT    2
#define COMMAND_CURSOR_ON_OFF_BIT    1
#define COMMAND_BLINK_ON_OFF_BIT      0
```

텍스트 LCD를 제어하기 이전에 공통으로 사용되는 몇 가지 함수를 정의해 보자. 기본적인 함수에는 문자 출력을 위한 LCD_wirte_data8 함수와 문자 출력 이외의 명령어 실행을 위한 LCD_write_command8 함수가 있다. 두 함수는 RS 출력을 1로 할 것인지 0으로 할 것인지에 차이가 있다. LCD 드라이버가 명령어를 실행하기 위해서는 E를 HIGH에서 LOW로 바꾸어 주어야 하며 이를 위해 LCD_pulse_enable 함수도 필요하다. 함수 정의에서 사용된 시간 함수는 LCD의 정상적인 동작을 위해 필요한 것으로 자세한 시간 지연 시간은 데이터시트를 참고하도록 한다. 화면을 지우는 명령의 경우 다른 명령에 비해 긴 시간이 소요되므로 별도의 함수로 작성하였다.

```
void LCD_pulse_enable(void) {           // 하강 에지에서 동작
    PORT_CONTROL |= (1 << E_PIN);      // E를 HIGH로
    _delay_us(1);
    PORT_CONTROL &= ~(1 << E_PIN);     // E를 LOW로
    _delay_ms(1);
}

void LCD_write_data(uint8_t data)
{
```

```
    PORT_CONTROL |= (1 << RS_PIN);    // 문자 출력에서 RS는 1
    PORT_DATA = data;                  // 출력할 문자 데이터
    LCD_pulse_enable();                 // 문자 출력
}

void LCD_write_command(uint8_t command)
{
    PORT_CONTROL &= ~(1 << RS_PIN);    // 명령어 실행에서 RS는 0
    PORT_DATA = command;                // 데이터 핀에 명령어 전달
    LCD_pulse_enable();                 // 명령어 실행
}

void LCD_clear(void)
{
    LCD_write_command(COMMAND_CLEAR_DISPLAY);
    _delay_ms(2);
}
```

8비트 모드로 텍스트 LCD를 초기화하는 함수는 다음과 같다. 이는 드라이버에서 텍스트 LCD를 초기화하는 순서와 동일하며, 텍스트 LCD를 켜진 상태로 하는 점만 다르다.

```
void LCD_init(void)
{
    _delay_ms(50);

    // 연결 핀을 출력으로 설정
    DDR_DATA = 0xFF;
    DDR_CONTROL |= (1 << RS_PIN) | (1 << E_PIN);

    LCD_write_command(COMMAND_8_BIT_MODE);           // 8비트 모드

    // 화면 on, 커서 off, 커서 깜빡임 off
    uint8_t command = 0x08 | (1 << COMMAND_DISPLAY_ON_OFF_BIT);
    LCD_write_command(command);

    LCD_clear();                                     // 화면 지움

    // Entry Mode Set. 출력 후 주소 증가, 화면 이동 없음
    LCD_write_command(0x06);
}
```

문자열 출력과 임의의 위치에 문자를 출력하기 위한 유틸리티 함수도 정의해 보자.

```

void LCD_write_string(char *string)
{
    uint8_t i;
    for(i = 0; string[i]; i++)          // 종료 문자를 만날 때까지
        LCD_write_data(string[i]);      // 문자 단위 출력
}

void LCD_goto_XY(uint8_t row, uint8_t col)
{
    col %= 16;    // [0 15]
    row %= 2;     // [0 1]

    // 첫째 라인 시작 주소는 0x00, 둘째 라인 시작 주소는 0x40
    uint8_t address = (0x40 * row) + col;
    uint8_t command = 0x80 + address;

    LCD_write_command(command);         // 커서 이동
}

```

코드 1은 먼저 텍스트 LCD를 초기화하고 'Hello World !' 문자열을 출력한다. 1초 대기 후 화면을 지운 후에 임의의 위치로 커서를 옮긴 후 문자열을 출력한다. 위에서 설명한 정의 및 함수들이 모두 포함시켜야 한다.

코드 1. 텍스트 LCD - 8비트 모드

```

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

int main(void)
{
    LCD_init();                // 텍스트 LCD 초기화

    LCD_write_string("Hello World!");    // 문자열 출력
}

```

```
_delay_ms(1000);                // 1초 대기

LCD_clear();                    // 화면 지움

// 화면에 보이는 영역은 기본값으로 0~1행, 0~15열로 설정되어 있다.
LCD_goto_XY(0, 0);              // 0행 0열로 이동
LCD_write_data('1');            // 문자 단위 출력
LCD_goto_XY(0, 5);
LCD_write_data('2');
LCD_goto_XY(1, 0);
LCD_write_data('3');
LCD_goto_XY(1, 5);
LCD_write_data('4');

while(1);
}
```



그림 4. 코드 1 실행 결과 - I

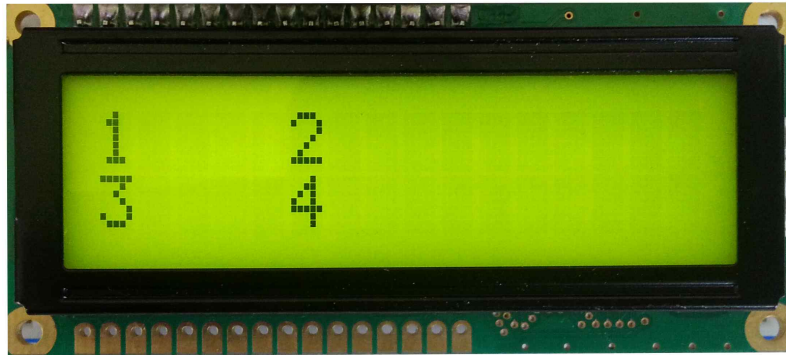


그림 5. 코드 1 실행 결과 - II

3. 4비트 모드 텍스트 LCD 제어

4비트 모드로 텍스트 LCD를 제어하는 방법을 알아보자. 4비트 모드는 8개의 데이터 핀 중 DB4에서 DB7까지의 4개 데이터 핀만을 사용한다. 먼저 그림 6과 같이 텍스트 LCD를 연결해보자. 그림 6의 회로도는 그림 3과 기본적으로 동일하며 DB0에서 DB3까지의 4개 데이터 핀이 연결되지 않은 점만 다르다.

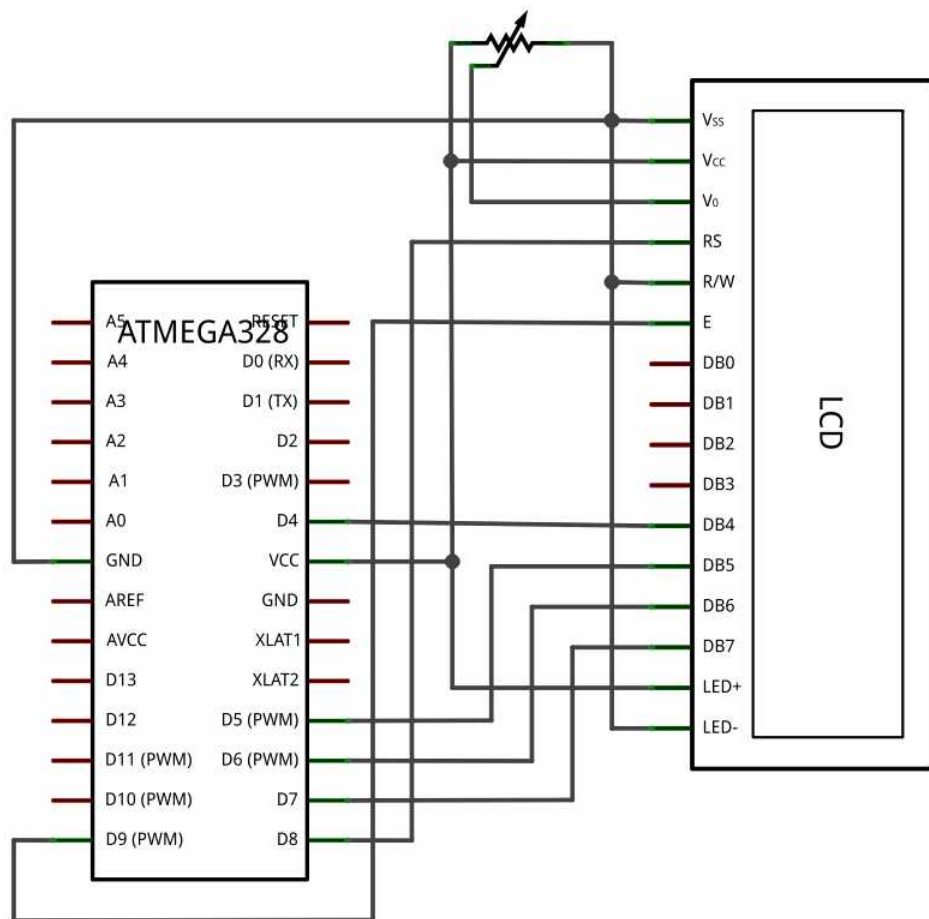


그림 6. 4비트 모드 텍스트 LCD 연결 회로도

LCD 드라이버에는 8비트 단위의 데이터가 전달되어야 하지만 4비트 모드에서는 4개의 데이터 핀만을 사용하므로 데이터는 두 번에 나누어서 전달된다. 이처럼 4비트 모드는 출력 속도 면에서 8비트 모드의 절반 정도이지만 사용할 수 있는 입출력 핀의 개수가 적은 ATmega328P의 경우에는 많이 사용된다. 먼저 8비트 모드와의 구별을 위해 전역적으로 비트 수를 나타내는 변수를 선언해 보자.

```
uint8_t MODE = 4;      // 8비트 모드인 경우 8, 4비트 모드인 경우 4
```

4비트 모드에서는 데이터를 4비트씩 두 번에 나누어 전달해야 하므로 LCD_write_data 함수와 LCD_write_command 함수는 이에 맞게 수정되어야 한다.

```

void LCD_write_data(uint8_t data)
{
    PORT_CONTROL |= (1 << RS_PIN);    // 문자 출력에서 RS는 1
    if(MODE == 8){
        PORT_DATA = data;              // 출력할 문자 데이터
        LCD_pulse_enable();            // 문자 출력
    }
    else{
        PORT_DATA = data & 0xF0; // 상위 4비트
        LCD_pulse_enable();
        PORT_DATA = (data << 4) & 0xF0; // 하위 4비트
        LCD_pulse_enable();
    }
}

void LCD_write_command(uint8_t command)
{
    PORT_CONTROL &= ~(1 << RS_PIN);    // 명령어 실행에서 RS는 0

    if(MODE == 8){
        PORT_DATA = command;           // 데이터 핀에 명령어 전달
        LCD_pulse_enable();            // 명령어 실행
    }
    else{
        PORT_DATA = command & 0xF0;    // 상위 4비트
        LCD_pulse_enable();
        PORT_DATA = (command << 4) & 0xF0; // 하위 4비트
        LCD_pulse_enable();
    }
}

```

또 하나 수정해야 할 함수는 초기화 함수이다. 텍스트 LCD는 8비트 모드를 기본으로 하고 있으므로, 4비트 모드로 사용하기 위해서는 초기화 명령 실행 이전에 추가되는 명령이 있다. 이때 실행되는 명령어 값은 0x02로 두 번에 나누어 실행되므로 표 3에서 0x00 명령과 0x20 명령이 실행되는 효과를 가지며 두 번째 명령이 4비트 모드로 설정하는 명령이다.

```

void LCD_init(void)
{
    _delay_ms(50);

    // 연결 핀을 출력으로 설정
    if(MODE == 8) DDR_DATA |= 0xFF;
    else DDR_DATA |= 0xF0;

    DDR_CONTROL |= (1 << RS_PIN) | (1 << E_PIN);

    if(MODE == 8)
        LCD_write_command(COMMAND_8_BIT_MODE);           // 8비트 모드
    else{
        LCD_write_command(0x02);                          // 4비트 모드 추가 명령
        LCD_write_command(COMMAND_4_BIT_MODE);            // 4비트 모드
    }

    // 화면 on, 커서 off, 커서 깜빡임 off
    uint8_t command = 0x08 | (1 << COMMAND_DISPLAY_ON_OFF_BIT);
    LCD_write_command(command);

    LCD_clear();     // 화면 지움

    // Entry Mode Set. 출력 후 주소 증가, 화면 이동 없음
    LCD_write_command(0x06);
}

```

나머지 함수들은 8비트 모드에서와 동일하다. 지금까지 작성한 함수로 텍스트 LCD 제어를 위한 라이브러리를 만들어 보자. 코드 2와 코드 3은 텍스트 LCD 제어에 사용될 헤더 파일과 소스 파일에 해당한다.

코드 2. TEXT_LCD.h

```

#ifndef _TEXT_LCD_H_
#define _TEXT_LCD_H_

#ifndef F_CPU
#define F_CPU 16000000
#endif

#include <avr/io.h>
#include <util/delay.h>

#define PORT_DATA          PORTD          // 데이터 핀 연결 포트
#define PORT_CONTROL       PORTB          // 제어 핀 연결 포트
#define DDR_DATA           DDRD           // 데이터 핀의 데이터 방향
#define DDR_CONTROL        DDRB          // 제어 핀의 데이터 방향

#define RS_PIN             0              // RS 제어 핀의 비트 번호
#define E_PIN              1              // E 제어 핀의 비트 번호

#define COMMAND_CLEAR_DISPLAY 0x01
#define COMMAND_8_BIT_MODE    0x38      // 8비트, 2라인, 5x8 폰트
#define COMMAND_4_BIT_MODE    0x28      // 4비트, 2라인, 5x8 폰트

#define COMMAND_DISPLAY_ON_OFF_BIT 2
#define COMMAND_CURSOR_ON_OFF_BIT 1
#define COMMAND_BLINK_ON_OFF_BIT 0

extern uint8_t MODE;

void LCD_pulse_enable(void);              // E를 HIGH -> LOW로 변화
void LCD_write_data(uint8_t data);        // 데이터 전송으로 문자 출력
void LCD_write_command(uint8_t command);  // 명령어 실행
void LCD_clear(void);                     // 화면을 공백 문자로 지움
void LCD_init(void);                      // 텍스트 LCD 초기화
void LCD_write_string(char *string);      // 문자열 출력
void LCD_goto_XY(uint8_t row, uint8_t col); // 임의의 위치로 커서 이동

```

```
#endif
```

코드 3. TEXT_LCD.c

```
#include "TEXT_LCD.h"

void LCD_pulse_enable(void) {           // 하강 에지에서 동작
    PORT_CONTROL |= (1 << E_PIN);       // E를 HIGH로
    _delay_us(1);
    PORT_CONTROL &= ~(1 << E_PIN);     // E를 LOW로
    _delay_ms(1);
}

void LCD_write_data(uint8_t data)
{
    PORT_CONTROL |= (1 << RS_PIN);      // 문자 출력에서 RS는 1
    if(MODE == 8){
        PORT_DATA = data;               // 출력할 문자 데이터
        LCD_pulse_enable();              // 문자 출력
    }
    else{
        PORT_DATA = data & 0xF0; // 상위 4비트
        LCD_pulse_enable();
        PORT_DATA = (data << 4) & 0xF0; // 하위 4비트
        LCD_pulse_enable();
    }
}

void LCD_write_command(uint8_t command)
{
    PORT_CONTROL &= ~(1 << RS_PIN);     // 명령어 실행에서 RS는 0

    if(MODE == 8){
        PORT_DATA = command;             // 데이터 핀에 명령어 전달
        LCD_pulse_enable();               // 명령어 실행
    }
}
```

```
}
else{
    PORT_DATA = command & 0xF0;    // 상위 4비트
    LCD_pulse_enable();
    PORT_DATA = (command << 4) & 0xF0;    // 하위 4비트
    LCD_pulse_enable();
}
}

void LCD_clear(void)
{
    LCD_write_command(COMMAND_CLEAR_DISPLAY);
    _delay_ms(2);
}

void LCD_init(void)
{
    _delay_ms(50);

    // 연결 핀을 출력으로 설정
    if(MODE == 8) DDR_DATA |= 0xFF;
    else DDR_DATA |= 0xF0;

    DDR_CONTROL |= (1 << RS_PIN) | (1 << E_PIN);

    if(MODE == 8)
        LCD_write_command(COMMAND_8_BIT_MODE);    // 8비트 모드
    else{
        LCD_write_command(0x02);
        LCD_write_command(COMMAND_4_BIT_MODE);    // 4비트 모드
    }

    // 화면 on, 커서 off, 커서 깜빡임 off
    uint8_t command = 0x08 | (1 << COMMAND_DISPLAY_ON_OFF_BIT);
    LCD_write_command(command);
}
```

```

LCD_clear();    // 화면 지움

// Entry Mode Set. 출력 후 주소 증가, 화면 이동 없음
LCD_write_command(0x06);
}

void LCD_write_string(char *string)
{
    uint8_t i;
    for(i = 0; string[i]; i++)          // 종료 문자를 만날 때까지
        LCD_write_data(string[i]);      // 문자 단위 출력
}

void LCD_goto_XY(uint8_t row, uint8_t col)
{
    col %= 16;    // [0 15]
    row %= 2;     // [0 1]

    // 첫째 라인 시작 주소는 0x00, 둘째 라인 시작 주소는 0x40
    uint8_t address = (0x40 * row) + col;
    uint8_t command = 0x80 + address;

    LCD_write_command(command);          // 커서 이동
}

```

코드 4는 4비트 모드로 텍스트 LCD를 제어하는 예이다. 실행 결과는 그림 4 및 그림 5와 동일하다. 전역적으로 선언된 MODE 변수의 값을 8로 바꾸는 것만으로 8비트 모드로 텍스트 LCD를 제어할 수 있도록 라이브러리가 구성되어 있다. 단, 8비트 모드로 사용하기 위해서는 데이터 핀 8개를 모두 연결해야 한다는 점을 잊지 말아야 한다.

코드 4. 텍스트 LCD - 4비트 모드

```

#ifndef F_CPU
#define F_CPU 16000000

```

```
#endif

#include <avr/io.h>
#include <util/delay.h>
#include "TEXT_LCD.h"

// 모드 설정을 위해 전역 변수로 선언해주어야 한다.
// 4 : 4비트 모드, 8 : 8비트 모드
uint8_t MODE = 4;

int main(void)
{
    LCD_init();                // 텍스트 LCD 초기화

    LCD_write_string("Hello World!");    // 문자열 출력

    _delay_ms(1000);           // 1초 대기

    LCD_clear();               // 화면 지움

    // 화면에 보이는 영역은 기본값으로 0~1행, 0~15열로 설정되어 있다.
    LCD_goto_XY(0, 0);         // 0행 0열로 이동
    LCD_write_data('1');       // 문자 단위 출력
    LCD_goto_XY(0, 5);
    LCD_write_data('2');
    LCD_goto_XY(1, 0);
    LCD_write_data('3');
    LCD_goto_XY(1, 5);
    LCD_write_data('4');

    while(1);
}
```

4. 텍스트 LCD 제어 - 아두이노

아두이노를 통해 텍스트 LCD를 제어해 보자. 아두이노에는 기본 라이브러리 중 하나로 LiquidCrystal 라이브러리가 포함되어 있어 쉽게 텍스트 LCD를 제어할 수 있다. 8비트 모드
의 경우를 살펴보자. 8비트 모드는 그림 3과 동일한 회로도를 사용한다.

먼저 ‘스케치 → 라이브러리 가져오기... → LiquidCrystal’ 메뉴를 선택하여 LiquidCrystal.h 파일을 포함시키고, LiquidCrystal 클래스의 객체를 생성한다. 객체를 생성할 때에는 텍스트 LCD와 연결된 핀 번호를 지정해 준다. 앞에서와 마찬가지로 텍스트 LCD에서 데이터를 읽어 오는 경우는 고려하지 않으므로 8비트 모드의 경우 8개의 데이터 핀과 2개의 제어 핀(RS, E)을 지정해 준다.

```
// 핀 번호 (RS, E, DB0, DB1, DB2, DB3, DB4, DB5, DB6, DB7)
LiquidCrystal lcd(8, 9, 0, 1, 2, 3, 4, 5, 6, 7);
```

다음은 begin 함수로 객체를 초기화 한다. 초기화 함수에서는 행과 열의 개수를 지정할 수 있다. 폰트의 크기도 저장할 수 있지만 앞에서 살펴본 것처럼 2행으로 표시되는 경우 5x8 크기의 폰트만을 사용할 수 있다.

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

텍스트 LCD에 문자를 출력하기 위해서는 바이트 단위의 출력함수인 write나 문자열 단위의 출력함수인 print를 사용할 수 있다. 특히 print 함수는 문자열은 물론 숫자를 출력할 수 있으며 숫자 출력 시 정수는 진법을, 실수는 소숫점 이하 자릿수를 지정할 수 있다.

```
size_t write(uint8_t ch)
    ch : 출력할 문자
    반환값 : 출력한 바이트 수
```

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

화면을 지우기 위해서는 clear 함수를, 커서를 임의의 위치로 옮기기 위해서는 setCursor 함수를 사용할 수 있다. 이외에도 LiquidCrystal 클래스에는 유용한 멤버 함수들이 정의되어 있으므로 보다 자세한 내용은 헤더 파일을 참고하면 된다.

```
void clear(void)
    반환값 : 없음
```

```
void setCursor(uint8_t col, uint8_t row)
    col : 열 위치 (0 ~ 15)
    row : 행 위치 (0 ~ 1)
    반환값 : 없음
```

코드 5는 LiquidCrystal 라이브러리를 이용하여 8비트 모드로 텍스트 LCD를 제어하는 코드의 예이다. 실행 결과는 이전과 동일하다.

코드 5. Hello World! - 아두이노 8비트 모드

```
#include <LiquidCrystal.h>

// 핀 번호 (RS, E, DB0, DB1, DB2, DB3, DB4, DB5, DB6, DB7)
LiquidCrystal lcd(8, 9, 0, 1, 2, 3, 4, 5, 6, 7);

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

```

    lcd.print("Hello World!");           // 문자열 출력

    delay(1000);                         // 1초 대기

    lcd.clear();                         // 화면 지움

    lcd.setCursor(0, 0);                 // 0행 0열로 이동
    lcd.write('1');                     // 문자 단위 출력
    lcd.setCursor(5, 0);
    lcd.write('2');
    lcd.setCursor(0, 1);
    lcd.write('3');
    lcd.setCursor(5, 1);
    lcd.write('3');
}

void loop(){
}

```

4비트 모드의 경우도 코드 5와 크게 다르지 않다. 4비트 모드를 위해서는 그림 6의 회로도를 사용한다. 4비트 모드를 사용하고자 할 때 코드 5에서 수정해야 하는 부분은 객체를 생성할 때 연결 핀 번호를 지정하는 부분이다. LiquidCrystal 라이브러리에는 오버로딩된 생성자가 여러 가지 존재하므로 8비트 모드로 사용하고 싶은 경우에는 데이터 핀을 8개 지정하고, 4비트 모드로 사용하고 싶은 경우에는 데이터 핀을 4개만 지정하면 된다. 4비트 모드를 사용하기 위한 생성자는 다음과 같다.

```

// 핀 번호 (RS, E, DB4, DB5, DB6, DB7)
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

```

이외에도 이 장에서는 사용하지 않았지만 텍스트 LCD에서 데이터를 읽어올 수 있도록 R/W 핀을 사용할 수 있도록 해주는 생성자도 존재한다. 코드 5에서 객체 생성 부분만을 위와 같이 수정하면 4비트 모드로 텍스트 LCD를 제어할 수 있으며, 이 때 8비트 모드에서 사용한 나머지

지 4개의 핀은 일반적인 입출력 핀으로 사용할 수 있어 일반적으로 텍스트 LCD 제어에서는 4비트 모드가 선호된다. 하지만 앞서도 언급한 것처럼 출력 속도가 느려지는 점은 감수하여야 한다.