1. Liquid Crystal Display(LCD) 장치

1.1 개요

(1) 마이크로컨트롤러를 사용하는 많은 기기에서 여러 가지 상태를 표시하기 위한 장치로서 LCD 표시장치를 사용하고 있다. LCD 모듈은 디스플레이 드라이버와 액정디스플레이를 합한 것을 말한다. 마이크로컨트롤러는 드라이버에 데이터를 써넣음으로써 LCD 화면에 정보를 표시할 수 있다.

(2) LCD 표시장치는 크게 문자형과 그래픽형 두 가지로 분류된다. 문자형은 드라이버 내에 글꼴을 가지고 있어 영문자나 숫자를 쉽게 표시할 수 있다. 그래픽 형은 문자형에 폰트가 없는 한글 또는 그래픽을 표시할 수 있으나 문자형보다는 사용하기 어렵다. 실습에서는 제어가 쉬운 문자형 LCD를 사용한다.

(3) 현재 시판되는 문자형 LCD 표시장치의 드라이버로서 Hitachi사의 HD44780이 가장 많이 쓰인다. 여기서는 HD44780의 특성을 개략적으로 설명하고 LCD를 제어하는 프로그램을 작성하도록 한다. LCD 사용 프로그램은 C-언어의 작성기법인 모듈화프로그램 방법으로 작성하여 이 후의 실습에서도 LCD 프로그램모듈을 계속적으로 사용할 수 있도록 하였다.

(4) LCD 드라이버에 대한 자세한 사항은 HD44780의 Data Sheet를 참고하기 바란다.
그림 1은 상업적으로 판매되고 있는 2-Line 16문자형 LCD 의 외형을 나타낸 사진이다.

그림 1. LC1621 16X2 LCD 외형

1.1.1 모듈화 프로그래밍

LCD는 데이터를 화면에 표시하여 사용자에게 필요한 정보를 제공할 목적으로 사용되므로 대부분의 마이크로컨트롤러의 응용에 사용된다고 볼 수 있다. 따라서 한 번 LCD 제어프로그램을 작성하면 여러 응용에서 계속적으로 사용할 수 있다.

프로그램의 재사용에 있어서 중요한 것은

(1) 프로그램을 모듈화하여 LCD제어 모듈만 있으면 LCD를 사용할 수 있고,

(2) LCD의 제어에 관한 자세한 사항을 몰라도 프로그램을 사용할 수 있어야 한다.

☞ 모듈화란 각 기능별로 프로그램을 작은 부분으로 나누는 것을 말한다. 모듈이 가져야할 기본적 조건은

- 기능적으로 다른 모듈들과 독립적으로 존재
- 독립성을 유지를 위해 데이터를 다른 모듈과 공유하지 않음.

으로 표현된다. 한마디로 독립성이다. 독립성이 유지되지 않으면 LCD를 사용할 때 LCD제어 모듈 외에 다른 모듈을 부가적으로 사용하여야 한다. 독립성의 유지로 재사용 항목 (1)을 구현할 수 있다.

☞ 모듈은 하향식설계(Top-Down)방식에 의해 세부 모듈로 다시 구분한다. 각 세부 모듈 역시 모듈 간 독립성을 유지도록 하는 것이 바람직하나, 각 세부 모듈은 엄밀한 독립성을 유지 못할 수도 있다.

☞ 각 세부 모듈은 C-언어의 함수로 구현한다. LCD 제어에 대한 자세한 사항을 함수 내에서 처리되도록 하면 어떤 기능이 필요한 때 기능을 수행하는 세부모듈명(즉, 함수명)만 알면 해당모듈(함수)을 호출하기만 하면 된다. LCD를 초기화할 때 초기화 함수만 호출하면 되지 그림 3의 초기화과정을 이해할 필요가 없다. 재사용 항목 (2)는 자세한 기능적 사항을 함수가 처리하도록 함으로써 충족시킬 수 있다.

모듈의 독립성으로 얻을 수 있는 장점은 다음과 같이 기술할 수 있다.

- ■ 프로그램 작성의 용이
- ■ 프로그램 이해가 쉽다.

- 오류정정이 간단하다.
- 동시 개발이 가능

프로그램 모듈을 C-언어로 구현할 때 다음 세 부분으로 구성된다.

.c 프로그램파일 : 세부 모듈을 구현한 함수의 모음
.h 헤더파일 : 프로그램 모듈을 사용할 때 필요한 사항
(함수의 정의, 명령의 정의 등)을 포함
설명서 : 모듈의 사용방법, 기능 등을 기술. 간단한 경우는 헤더파일의 주석문으로 대체 가능

프로그램의 독립성을 위하여 각 부분별로 지켜야 할 사항은 다음과 같다.

- 내부데이터는 외부로부터 숨김
  - 전역변수의 사용 금지
- 내부적으로만 사용하는 세부 모듈을 외부에 숨김
  - static을 사용하여 정적함수로 작성
- 헤더파일에는 외부에서 꼭 알아야만 할 사항만 첨부

1.1.2 프로그램 설계

LCD는 화면에 필요한 데이터를 표시하기 위해서 사용한다. LCD의 기능은 다른 장치(예: 직렬통신장치)와는 독립적인 기능들이므로 LCD만 전용으로 제어하는 프로그램 모듈을 만들 수 있다. 그림 2는 LCD제어 모듈의 내부 구성도를 나타낸 것이다.

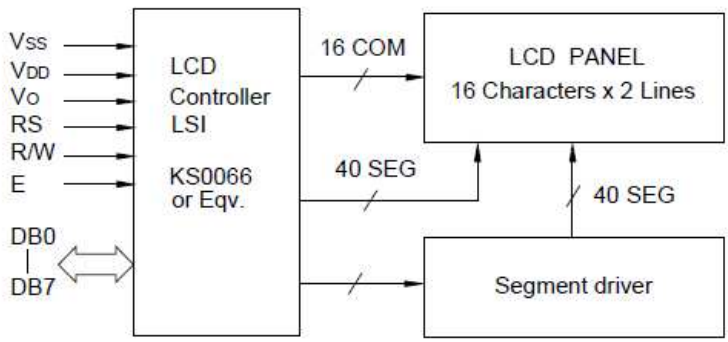


그림 2. LCD 모듈의 내부 구성도

LCD제어 모듈이 외부 프로그램에 제공하여야 할 기능을 분류하여 보자.

LCD 제어모듈

1) 초기화 기능
2) 화면동작제어 기능
3) 문자 표시 기능
    ① 1 문자를 표시 기능
    ② 문자열을 표시 기능
    ③ 표시위치 이동 기능
4) 사용자 글꼴 등록 기능

위와 같이 외부에 공개될 기능을 수행하기 위해 모듈내부에서 수행되어야 할 기능은 표 2를 고려하면 5가지로 분류할 수 있다.

(1) 명령 레지스터(IR)에 명령 쓰기
(2) DDRAM 또는 CGRAM에 데이터 쓰기
(3) 비지 플래그/어드레스 카운터 읽기
(4) DDRAM 또는 CGRAM 읽기
(5) 시간지연

위에서 (4)의 읽기는 LCD드라이버 내부 레지스터를 데이터를 저장하는 메모리로 사용하는 것으로서 여기서는 사용하지 않으므로 모듈함수로서 작성하지 않을 것이다.

우선 내부기능을 함수로 작성하자. 함수의 역할은 함수에 필요한 데이터를 호출자로부터 받아 정해진 기능을 수행하고 수행결과를 호출자에게 전달하는 것이다. 따라서 함수를 작성할 때는

- 함수가 수행하여야 할 기능
- 이를 수행 하기위해 호출자로부터 받아야 할 입력
- 수행결과로 호출자에게 전달하여야 할 함수의 출력

을 정의하여야 한다. 이 정의로부터 입력을 위한 함수의 인자, 출력을 위한 함수의 인자 또는 반환 값을 선택한다.

표 2 LCD 명령 Sets

명 령	코드										실행 시간
	RS	R/W	DB 7	DB 6	DB 5	DB 4	DB 3	DB 2	DB 1	DB 0	
화면 지움	0	0	0	0	0	0	0	0	0	1	1.52ms
커서 홈	0	0	0	0	0	0	0	0	1	*	1.52ms
엔트리모드 세트	0	0	0	0	0	0	0	1	I/D	SH	37 us
화면 ON/OFF 제어	0	0	0	0	0	0	1	D	C	B	37 us
커서/화면 이동	0	0	0	0	0	1	S/C	R/L	*	*	37 us
펑션 세트	0	0	0	0	1	DL	N	F	*	*	37 us
CGRAM 주소 설정	0	0	0	1	CGRAM 주소						37 us
DDRAM 주소 설정	0	0	1	DDRAM 주소							37 us
비지 플래그와 어드레스 카운터 읽기	0	1	BF	CGRAM/DDRAM 주소							0 us
CGRAM 또는 DDRAM에 데이터 쓰기	1	0	출력 데이터								37 us
CGRAM 또는 DDRAM의 데이터 읽기	1	1	입력 데이터								37 us

비트설정 방법

I/D : 0 - 커서 위치 감소, 1 - 커서 위치 증가
SH : 0 - 화면 이동 없음, 1 - 화면 전체 이동
D : 0 - 화면 끔, 1 - 화면 켬
C : 0 - 커서 끔, 1 - 커서 켬
B : 0 - 커서 블링커 끔, 1 - 커서 블링커 켬
S/C : 0 - 커서 이동, 1 - 화면 이동
R/L : 0 - 왼쪽으로 이동, 1 - 오른쪽으로 이동
DL : 0 - 4비트 인터페이스, 1 - 8비트 인터페이스
N : 0 - 1라인 디스플레이, 1 - 2라인 디스플레이
F : 0 - 5x8 도트 문자, 1 - 5x10 도트 문자
BF : 0 - 명령수행 가능, 1 - 명령수행 불가능

* : No Effect(Don't Care)

1.1.3 내부 함수 작성

(1) 명령레지스터에 명령 쓰기 함수

표 2에서 명령레지스터에 전달할 명령은 여러 가지가 있다. 이 기능을 수행할 함수는 전달할 명령을 입력으로 받기로 한다. 함수의 수행결과를 함수 호출자에게 알릴 필요가 없다. 이로부터 함수의 입출력을 다음과 같이 선정한다.

인자 입력 : LCD에 내릴 명령어로 1바이트 형 ⇒ char command
인자 출력 : 없음
반환 값 : 없음 ⇒ void형

이로부터 함수 형태를 다음과 같이 선정한다.

```
static void write_command(char command)
{
        . . .
}
```

내부함수이므로 정적함수를 사용하여 외부에 함수를 숨긴다.

입력 인자 command가 다음과 같이 비트별로 표현된다 하면

비트 위치	D7	D6	D5	D4	D3	D2	D1	D0
command	b7	b6	b5	b4	b3	b2	b1	b0

표 2를 참고하면 표 3과 같은 순서로 PORTD에 데이터를 출력하여야 한다.

표 3 LCD 명령쓰기를 위한 데이터 출력 순서

출력 포트D 핀	D7	D6	D5	D4	D3	D2	D1	D0	AVR u-COM
LCD모듈 핀	D7	D6	D5	D4	x	E	R/W	RS	
출력 포트D에	b7	b6	b5	b4	0	1	0	0	①
상위니블 출력	b7	b6	b5	b4	0	0	0	0	②
출력 포트D	b3	b2	b1	b0	0	1	0	0	③
하위니블 출력	b3	b2	b1	b0	0	0	0	0	④

☞ LCD 출력 PORTD를 매크로를 사용하여 LCD_PORT라고 정의하자.

```
#define LCD_PORT        PORTD
```

☞ 입력변수 command의 **상위니블**과 E, R/W, RS신호로 데이터를 구성한다.

```
temp = (command & 0xF0)  | 0x04 ;    //E=1
```

	0 0 0 0 0 1 0 0
command의 상위니블	b7 b6 b5 b4 0 0 0 0
비트별 OR	----------
표 3의 데이터 ①	b7 b6 b5 b4 0 1 0 0

☞ 구성된 데이터를 LCD_PORT에 출력한다.

```
LCD_PORT = temp;
```

☞ 표 3의 데이터 ②를 LCD_PORT에 출력한다.

```
LCD_PORT = temp    & ~0x04;        //E=0
```

temp	b7 b6 b5 b4 0 1 0 0
~0x04	1 1 1 1 1 0 1 1
비트별 AND	----------
표 3의 데이터 ②	b7 b6 b5 b4 0 0 0 0

☞ 입력변수 command의 **하위니블**과 E, R/W, RS신호로 데이터를 구성한다.

```
temp = (command << 4)  |  0x04 ;      //E=1
```

	0 0 0 0 0 1 0 0
command의 하위니블	b3 b2 b1 b0 0 0 0 0
비트별 OR	----------
표 3의 데이터 ③	b3 b2 b1 b0 0 1 0 0

☞ 구성된 데이터를 LCD_PORT에 출력한다.

```
LCD_PORT = temp;
```

☞ 표 3의 데이터 ④를 LCD_PORT에 출력한다.

```
LCD_PORT = temp    & ~0x04;          //E=0
```

temp	b3 b2 b1 b0 0 1 0 0
~0x04	1 1 1 1 1 0 1 1
비트별 AND	----------
표 3의 데이터 ④	b3 b2 b1 b0 0 0 0 0

이로부터 write_command()함수를 완성하면 다음과 같다.

```
static void write_command(char command)
{
      char    temp;

      // 상위 nibble 출력
      temp = (command & 0xF0) | 0x04;      //0x04: RS, RW=0, E=1
      LCD_PORT = temp;
      LCD_PORT = temp & ~0x04;             //E=0, LCD Disable

      // 하위 nibble 출력
      temp = (command << 4) | 0x04;        //0x04: RS, RW=0, E=1
      LCD_PORT = temp;
      LCD_PORT = temp & ~0x04;             //E=0, LCD Disable

}
```

(2) DDRAM 또는 CGRAM에 데이터 쓰기 함수

표 2에서 1바이트 단위로 DDRAM 또는 CGRAM에 데이터를 쓴다. 데이터 쓰기 함수는 DDRAM 또는 CGRAM에 쓸 1바이트 데이터를 함수의 입력으로 받는다. 함수의 수행결과를 함수 호출자에게 알릴 필요가 없다. 이로부터 함수의 입출력을 다음과 같이 선정한다.

인자 입력 : 1바이트 형 데이터 ⇒ char ch
인자 출력 : 없음
반환 값 : 없음 ⇒ void형

이로부터 함수 형태를 다음과 같이 선정한다.

```
static void write_data(char ch)
{
        . . .
}
```

입력 인자 ch가 다음과 같이 비트별로 표현된다 하면

비트 위치	b7	b6	b5	b4	b3	b2	b1	b0
ch	d7	d6	d5	d4	d3	d2	d1	d0

DDRAM과 CGRAM에 데이터를 쓰려면 데이터 레지스터(DR)을 선택(RS=1)하여 LCD에 데이터를 쓰면 된다. 표 2를 참고하면 표 4의 순서로 출력 포트D에 데이터를 출력하여야 한다.

표 4 DDRAM 또는 CGRAM에 데이터 출력 순서

출력 포트 D핀	D7	D6	D5	D4	D3	D2	D1	D0	AVR u-COM
LCD모듈 핀	D7	D6	D5	D4	x	E	R/W	RS	
출력 포트D에	b7	b6	b5	b4	0	1	0	1	⑤
상위니블 출력	b7	b6	b5	b4	0	0	0	1	⑥
출력 포트D에	b3	b2	b1	b0	0	1	0	1	⑦
하위니블 출력	b3	b2	b1	b0	0	0	0	1	⑧

write_command()함수 작성과 같은 방법으로 표 4에 따라 write_data() 함수를 완성하면 다음과 같다.

```
static void write_data(char ch)
{
      char    temp;

      // 상위 nibble 출력
      temp = (ch & 0xF0) | 0x05;              // RS=1, RW=0, E=1
      LCD_PORT = temp;
      LCD_PORT = temp & ~0x04;                //E=0, LCD Disable

      // 하위 nibble 출력
      temp = (ch << 4) | 0x05;                //RS=1, RW=0, E=1
      LCD_PORT = temp;
      LCD_PORT = temp & ~0x04;                //E=0, LCD Disable

}
```

(3) 비지 플래그/어드레스 카운터 읽기 함수

비지 플래그 검사는 LCD가 현재 하고 있는 작업이 완료되었는지 판단을 하는 것이므로 작업수행에 필요한 시간만 지연하여도 비슷한 효과를 얻을 수 있다. 여기서는 시간지연으로 대체한다.

```
#include <util/delay.h>
static void check_busy(void)
{
      _delay_ms(1);
}
```

여기서 _delay_ms()는 WinAVR에서 라이브러리로 제공하는 함수이므로 헤더파일 <util/delay.h>를 첨부하여야 한다.

(4) 시간지연함수

시간지연함수는 WinAVR의 시간지연 라이브러리 함수를 사용한다. msec단위의 지연을 위해서는 _delay_ms(), usec단위의 지연을 위해서는 _delay_us()함수를 사용한다.

☞ 라이브러리 함수를 쓰지 못할 때는 for 루프와 nop를 이용한 msec_delay() 함수를 만들어 시간지연함수로 사용하면 된다.

1.1.4 LCD모듈 함수 작성

여기서는 앞에서 작성한 내부함수를 사용하여 LCD 제어프로그램 즉 LCD 모듈을 구성하는 함수들을 하나의 파일에 모두 작성한다. 응용프로그램들은 이 함수들을 호출하여 LCD에 문자를 표시할 수 있다.

☞ 구성함수가 LCD 모듈의 일원이라는 것을 나타내기 위해 접두어 LCD를 함수명에 사용하기로 한다.

☞ 표 2를 보면 명령의 각 비트를 적당히 set/reset하여 LCD에 여러 가지 명령을 내릴 수 있다. 실제로 LCD를 사용함에 있어서 모든 명령을 사용하는 것이 아니므로 많이 사용할 명령을 미리 분류하고 명령에 대한 코드 값을 추출한다.

```
화면지움 명령 :     코드 값          0b00000001 = 0x01
커서 홈       :     코드 값          0b00000010 = 0x02
엔트리모드    :     글자를 쓴 후 커서 위치가 증가함 ⇒ I/D = 1
                    글자를 쓸 때 화면의 이동이 없음 ⇒ SH = 0
                    코드 값 0b00000110 = 0x06
화면 켜기     :     화면 켬, 커서 끔, 블링커 끔 ⇒  D=1, C=0, B=0
                    코드 값 0b00001100 = 0x0C
화면 끄기     :     화면 끔, 커서 끔, 블링커 끔 ⇒  D=0, C=0, B=0
                    코드 값 0b00001000 = 0x08
커서 켜기     :     화면 켬, 커서 켬, 블링커 끔 ⇒  D=1, C=1, B=0
                    코드 값 0b00001110 = 0x0E
커서 끄기     :     화면 켜기와 같음
커서 왼쪽이동:      커서이동, 왼쪽이동 ⇒  S/C=0, R/L=0
                    코드 값 0b00010000 = 0x10
커서 오른쪽이동 :   커서이동, 오른쪽이동 ⇒  S/C=0, R/L=1
                    코드 값 0b00010100 = 0x14
```

```
화면 왼쪽이동:      화면이동, 왼쪽이동 ⇒  S/C=1, R/L=0
                    코드 값 0b00011000 = 0x18
화면 오른쪽이동 :   화면이동, 오른쪽이동 ⇒  S/C=1, R/L=1
                    코드 값 0b00011100 = 0x1C
펑션 셋       :     4비트 인터페이스          ⇒  D/L = 0
                    2-line 화면               ⇒  N = 1
                    5x7도트 문자 사용         ⇒  F = 0
                    코드 값 0b00111000 = 0x28
```

☞ 코드 값으로는 명령의 의미를 파악하기 어려우므로 매크로를 사용하여 코드에 의미를 부여한다.

```
#define ALLCLR          0x01    // 화면을 지움
#define HOME            0x02    // 커서 홈
#define ENTMOD          0x06    // 엔트리모드
#define DISP_ON         0x0C    // 화면 켜기
#define DISP_OFF        0x08    // 화면 끄기
#define CURSOR_ON       0x0E    // 커서 켜기
#define CURSOR_OFF      0x0C    // 커서 끄기
#define CURSOR_LSHIFT   0x10    // 커서 왼쪽 이동
#define CURSOR_RSHIFT   0x14    // 커서 오른쪽 이동
#define DISP_LSHIFT     0x18    // 화면 왼쪽 이동
#define DISP_RSHIFT     0x1C    // 화면 오른쪽 이동
#define FUNSET          0x28    // 펑션세트 (4bit 모드)
```

여기서부터 LCD제어 프로그램의 기능을 작성하도록 하자.

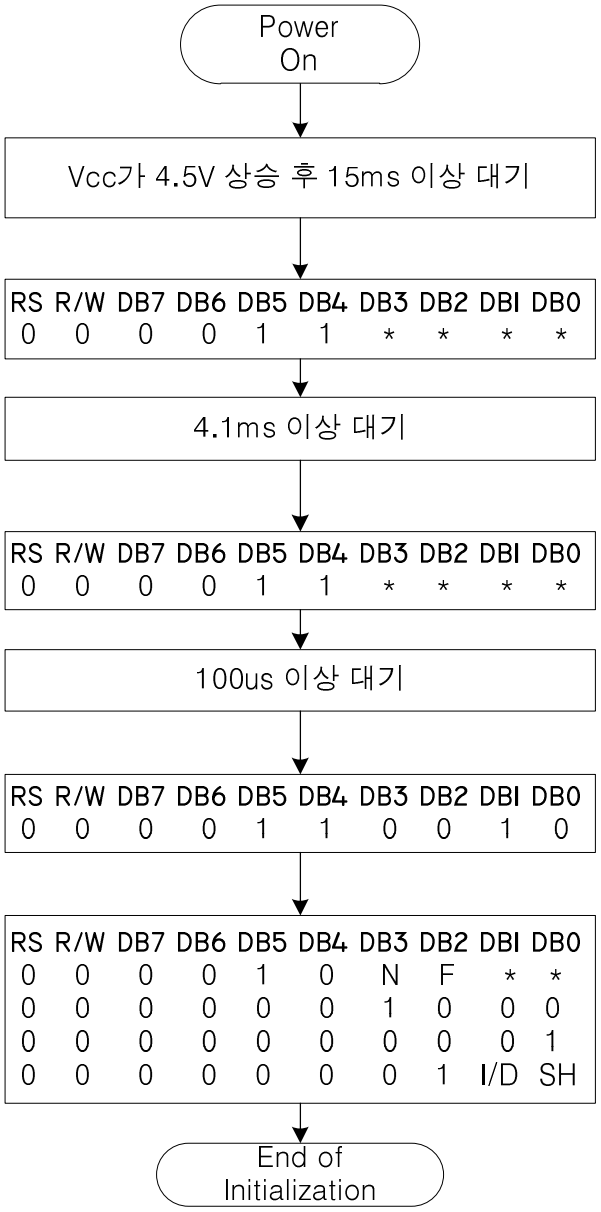
(1) 초기화 모듈 : LCDInit()

초기화를 위해 함수에 필요한 입력은 없으며 함수의 결과를 출력할 필요가 없다.

인자 입력 : 없음
인자 출력 : 없음
반환 값 : 없음 ⇒ void형

LCD를 사용하기 위해서는 초기화 과정이 필요하며 다음과 같이 작성한다. 여기서 초기화에 필요한 비트들은 N=1, F=0, I/D=1, SH=0으로 앞의 명령어를 구성할 때와 같은 것을 사용한다.

그림 3은 LCD의 초기화 과정을 나타낸 것이다.



- 4-bit 모드로 펑션set(0x28)
- Display Off (0x08)
- Display Clear (0x01)
- Entry Mode Set (0x06)

그림 3. LCD 초기화 과정

```
// LCD 포트 주소, 4-bit 모드(DL=0)

#define LCD_DDR     DDRD   // 방향레지스터 정의

void LCDInit(void)
{
      LCD_DDR = 0xFF;        // LCD 포트를 출력으로 설정

      _delay_ms(15);
      write_command(0x30);
      _delay_ms(5);
      write_command(0x30);
      _delay_ms(1);
      write_command(0x32);

      LCDCommand(FUNSET);
      LCDCommand(DISP_OFF);
      LCDCommand(ALLCLR);
      LCDCommand(ENTMOD);

      LCDCommand(DISP_ON);                    // 화면을 켠다.
}
```

☞ LCD를 연결한 포트는 출력포트이므로 포트의 방향을 출력포트로 하여야 한다. LCD 포트 방향레지스터는 매크로를 사용하여 정의한다.

☞ 함수에서 FUNSET을 출력하는 부분부터는 비지플래그를 판별하여야 하므로 write_instruction()함수 대신 비지플래그를 판별하는 화면동작제어 함수인 LCDCommand()를 사용하였다. LCDCommand()함수는 뒤에서 작성될 것이다.

☞ 함수의 마지막 문장은 초기화 과정에서는 필요하지 않다. 이는 LCD가 초기화된 후 화면이 꺼져 있는 상태가 되므로 화면을 켜기 위함이다.

(2) 화면동작제어 모듈(LCDCommand())

화면동작제어는 명령레지스터에 앞에서 매크로로 정의된 명령을 IR에 쓰면 된다. 단 write_command()함수로 명령을 쓰기 전에 비지플래그를 판별하여 LCD에 명령을 쓸 수 있을 때를 기다려야 한다. 함수의 입출력은

인자 입력 : LCD에 내릴 명령어로 1바이트 형 ⇒ char command
인자 출력 : 없음
반환 값 : 없음 ⇒ void형

```
void LCDCommand(char command)  // LCD 명령함수
{
      checkbusy();              // LCD 명령이 가능할 때까지 기다림
      write_command(command);       // 명령전달
      if(command == ALLCLR || command == HOME)
            _delay_ms(2);
}
```

☞ 입력 command로 어떠한 1 바이트 값을 입력해도 관계는 없으나, 의미가 있는 동작 명령을 주기위해서는 앞에서 정의한 매크로를 사용하는 것이 좋다.
(예) LCDCommand(ALLCLR); // 화면을 지운다.

(3) 문자 표시 모듈

① 한 문자 표시 모듈(LCDPutchar())

문자를 화면에 표시하려면 DDRAM에 데이터를 출력하면 된다. 이는 write_data()함수를 사용하면 되지만 데이터를 쓰기 전에 비지플래그를 판별하여 데이터를 쓸 수 있을 때를 기다려야 한다. 함수의 입출력은

인자 입력 : 화면에 표시할 문자(1바이트 형 데이터) ⇒ char
인자 출력 : 없음
반환 값 : 없음 ⇒ void형

```
void LCDPutchar(char ch)    // LCD에 문자 하나를 쓰는 함수
{
      checkbusy();              // LCD 명령이 가능할 때까지 기다림
      write_data(ch);           // 데이터전달
```

```
}
```

☞ 화면에 나타나는 문자는 표 3에 따라 출력된다.

```
(예) LCDPutchar('A')              // 화면에 글자 'A'를 표시
     LCDPutchar(0x41);            // 'A'=0x41=0b01000001
                                  // 화면에 글자 'A'를 표시
```

SRAM영역의 문자열 표시 모듈(LCDPuts())

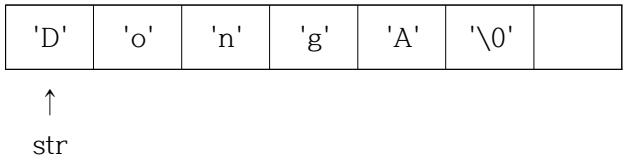
SRAM 영역의 문자열 입력을 받아 LCD 화면에 표시한다. 입출력은

인자 입력 : SRAM 영역의 문자열 ⇒ char * str

인자 출력 : 없음

반환 값 : 없음 ⇒ void형

문자열 입력을 str로 받는다면 str이 가리키는 문자를 출력하고 str이 가리키는 위치를 다음으로 이동한다.(str++) 이 동작을 NULL 문자를 만날 때까지 반복한다.




```
void LCDPuts(char *str)      // LCD에 문자열을 쓰는 함수
{
     while(*str)       // *str이 NULL 문자가 아니면 루프를 돈다.
     {                                  //while(*str != 0)
          LCDPutchar(*str);             // 문자 *str을 화면에 출력
          str++;                        // str이 다음 문자를 가리킴
     }
}
```

② 표시위치 이동(LCDMove())

앞의 함수 LCDPutchar()는 현재 DDRAM의 어드레스 카운터가 가리키는 위치에 문자를 출력하고 어드레스 카운터를 자동적으로 하나 증가시킨다. 따라서

LCDPutchar()는 이전에 쓴 문자 다음에 문자를 쓴다. 새로운 위치에 문자를 쓰려면 쓰기 전에 어드레스 카운터를 새로운 위치로 이동시켜야 한다. 이는 표 2의 DDRAM의 주소설정 명령으로 수행한다.

이 함수의 입력은 글자를 쓸 위치이다.

인자 입력 : 위치(행, 열) ⇒ char line, char pos
인자 출력 : 없음
반환 값 : 없음 ⇒ void형

함수 내에서는 행, 열 데이터로부터 DDRAM의 주소를 계산하고 이를 LCD 명령으로 출력하면 된다. 2-Line LCD의 DDRAM 주소는 그림 4a와 같다.

화면 열	0	1	2	3	4	5	6	7	8	9	…	38	39	
DDRAM	00	01	02	03	04	05	06	07	08	09	…	26	27	0행
주소	40	41	42	43	44	45	46	47	48	49	…	66	67	1행

그림 4a. 2-Line LCD의 DDRAM 주소

Address Format	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Character Address	1	Adr.6	Adr.5	Adr.4	Adr.3	Adr.2	Adr.1	Adr.0

Display position →		1	2	3	---	18	19	20
DD RAM Addr.	Line 1	00H	01H	02H	---	11H	12H	13H
	Line 2	40H	41H	42H	---	51H	52H	53H
	Line 3	14H	15H	16H	---	25H	26H	27H
	Line 4	54H	55H	56H	---	65H	66H	67H

그림 4b. 20×4 LCD의 DDRAM 주소

0행에 해당하는 DDRAM의 주소는 열과 같고 1행에 해당하는 DDRAM의 주소는 화면 열에 0x40을 더하면 된다.

```
void LCDMove(char line, char pos)      // line : 행, pos : 열
{
      char addr;

      addr = (line << 6) + pos;          // DDRAM의 주소 계산
      addr |= 0x80;            // 비트7을 세트 ⇒ DDRAM 주소설정 명령

      LCDCommand(addr);                // 명령 전달
}
```

☞ (line << 6)은 line=0일 때 0이고 line=1일 때 0b01000000 = 0x40이다.

1.1.5 파일 작성

LCD 제어모듈의 헤더파일과 소스파일은 다음 사항들을 고려하여 작성하였으며 이름은 각각 LCD.h 와 LCD.c이다.

(1) 헤더파일 작성

헤더파일의 작성에서 고려하여야 할 사항은 다음과 같다.

(1) 외부에서 호출하는 함수의 선언만 포함시키고, 내부함수는 외부에서 접근하는 함수가 아니므로 헤더파일이 아니라 소스파일에서 선언한다.
(2) 매크로 역시 외부에서 사용하는 것만 헤더파일에 포함하고 내부용은 소스파일에서 정의한다.

(3) 헤더파일 중복 첨부를 피하기 위해 선행처리기 #ifndef-#endif쌍을 사용한다. 헤더파일의 중복을 피하기 위해 정의하는 매크로의 이름은 일반적으로 쓰지 않는 것을 택한다. (예) __LCD_H__

(2) 소스파일 작성

소스파일의 이름은 모듈의 내용을 반영하도록 채택하고 앞에서 작성한 내부함수와 외부 함수 모두를 다음 사항을 고려하여 포함시킨다.

(1) 외부함수의 정의는 모두 헤더파일에 포함되어 있으므로 헤더파일을 첨부하고 소스파일 내에는 내부함수의 선언 및 내부용 매크로만 정의한다.
(2) 함수를 작성할 때 함수의 기능 및 입출력관계를 주석문에 상세히 기록하여 함수의 사용방법을 알 수 있도록 한다.

[헤더파일 LCD.h]

```
//================================================
//  파일명 : LCD.h - LCD제어 모듈의 헤더파일
//================================================
#ifndef __LCD_H__
#define __LCD_H__

// LCD 제어 명령

#define ALLCLR          0x01    // 화면을 지운다.
#define HOME            0x02    // 커서를 홈으로 보낸다.
#define LN21            0xc0    // 커서를 2번째 라인의
                                // 첫번째에 위치시킴
#define ENTMOD          0x06    // entry mode
#define FUNSET          0x28    // function set
#define DISP_ON         0x0c    // 디스플레이를 켠다.
#define DISP_OFF        0x08    // 디스플레이를 끈다.
#define CURSOR_ON       0x0e    // 커서를 켠다.
#define CURSOR_OFF      0x0c    // 커서를 끈다.
#define CURSOR_LSHIFT   0x10    // 커서를 왼쪽을 이동시킨다.
#define CURSOR_RSHIFT   0x14    // 커서를 오른쪽으로 이동시킨다.
#define DISP_LSHIFT     0x18    // 디스플레이를 왼쪽으로
                                // 이동시킨다.
#define DISP_RSHIFT     0x1c    // 디스플레이를 오른쪽으로
                                // 이동시킨다.

// LCD 제어모듈 함수

void LCDInit(void);
void LCDCommand(char command);
void LCDMove(char line, char pos);
void LCDPutchar(char ch);
void LCDPuts(char* str);

#endif      // __LCD_H__
```

[프로그램 LCD.c]

```
//=========================================================
//  LCD.C : LCD 구동함수의 모음
//=========================================================

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

// LCD 포트 주소
#define LCD_PORT            PORTD
#define LCD_DDR             DDRD

// 내부 함수
static void checkbusy(void);
static void write_command(char command);
static void write_data(char ch);

//===================================
// 기능 : LCD Display를 초기화한다.
//===================================

void LCDInit(void)
{
      LCD_DDR = 0xFF;               // LCD포트를 출력으로 설정

      _delay_ms(15);
      write_command(0x30);
      _delay_ms(5);
      write_command(0x30);
      _delay_ms(1);
      write_command(0x32);
```

```
	LCDCommand(FUNSET);
	LCDCommand(DISP_OFF);
	LCDCommand(ALLCLR);
	LCDCommand(ENTMOD);
//Initialization ends
	LCDCommand(DISP_ON);			// 화면을 켠다.
}

//=====================================================
// LCD에 명령을 출력하는 함수
//
//   입력 : command - LCD에 내리는 명령
//                    LCD.h에 정의된 명령을 사용할 것
//=====================================================

void LCDCommand(char command)
{
	checkbusy();
	write_command(command);
	if(command == ALLCLR || command == HOME)
		_delay_ms(2);
}

//=====================================================
// 현재위치에 문자 하나를 출력한다.
//
//   입력 : ch - 화면에 쓸 문자 코드
//
//=====================================================

void LCDPutchar(char ch)
{
	checkbusy();
	write_data(ch);
}
```

```
//=======================================================
// 현재위치에 문자열을 출력한다.
//
//   입력 : str - 출력할 문자열
//
//=======================================================

void LCDPuts(char* str)
{
     while(*str)          // *str이 NULL 문자가 아니면 루프를 돈다.
     {                              //while(*str != 0)
          LCDPutchar(*str);     // 문자 *str을 화면에 출력
          str++;                // str이 다음 문자를 가리킴
     }
}

//=======================================================
// 글자를 쓸 위치를 지정된 위치(line, pos)로 이동시킨다.
// 입력 :     line - 화면의 행(0행부터 시작)
//                  pos  - 화면의 열(0열부터 시작)
//=======================================================
void LCDMove(char line, char pos)
{
     pos = (line << 6) + pos;
     pos |= 0x80;                    // 비트 7를 세트한다.

     LCDCommand(pos);
}

//=======================================================
// 명령 레지스터에 명령을 쓴다.
// 입력 : command - LCD에 내리는 명령 코드
//=======================================================
```

```
static void write_command(char command)
{
	char	temp;

	// 상위 nibble 출력
	temp = (command & 0xF0) | 0x04;		//0x04: RS, RW=0, E=1
	LCD_PORT = temp;
	LCD_PORT = temp & ~0x04;			//E=0, LCD Disable

	// 하위 nibble 출력
	temp = (command << 4) | 0x04;		//0x04: RS, RW=0, E=1
	LCD_PORT = temp;
	LCD_PORT = temp & ~0x04;			//E=0, LCD Disable
	_delay_us(1) ;
}

//=======================================
// 데이터 레지스터에 명령을 쓴다.
// 입력 : ch - LCD에 쓸 데이터
//=======================================

static void write_data(char ch)
{
	unsigned char  temp;

	// 상위 nibble 출력
	temp = (ch & 0xF0) | 0x05;		//0x05: RS=1, RW=0, E=1
	LCD_PORT = temp;
	LCD_PORT = temp & ~0x04;			//E=0, LCD Disable

	// 하위 nibble 출력
	temp = (ch << 4) | 0x05;		//0x05: RS=1, RW=0, E=1
	LCD_PORT = temp;
	LCD_PORT = temp & ~0x04;			//E=0, LCD Disable
}
```

```
//=====================================================
// 글꼴을 등록한다.
//
// 입력 :  ch  : 새로운 글자를 심을 글자(1~7)
//         font : 글꼴
//=====================================================

void LCDNewchar(char ch, char font[])       // 글자 등록함수
{
      int i;

      ch <<= 3;                 // ch = ch << 3;과 같음
      ch |= 0x40;               // 비트6을 세트 => CGRAM 주소설정

      LCDCommand(ch);           // CGRAM 주소설정 =>LCDPutchar()로
                                // 쓰는 문자는 CGRAM에 저장

      for(i=0; i<8; i++)        // 글꼴을 CGRAM에 저장한다.
            LCDPutchar(font[i]);
}



//=================================================
// 1msec의 시간지연으로 BF 플래그 검사를 대체
//=========================================

static void checkbusy()
{
      _delay_us(100);
      return;
}
```

1.2 LCD 구동 실습

프로그램 LCD_test.c는 앞에서 작성한 LCD 제어프로그램의 사용방법을 보여주는 프로그램이다.

① LCD를 사용하므로 프로젝트에 LCD제어프로그램의의 소스파일인 LCD.c를 포함시켜야 한다.
② 프로그램에서 LCD제어 함수와 매크로를 사용하기 위해서 LCD.h를 포함시켜야 한다.
③ LCD.c와 LCD.h는 프로젝트가 있는 폴더에 있어야 한다.
④ 문자열을 나타내는 LCDtitle[]과 string[]은 문자열이므로 LCDPuts()함수를 사용하여 출력한다.

그림 5는 4-bit 모드 LCD 구동 인터페이스 회로를 나타낸 것이다. LCD의 V_0는 화면의 밝기를 조절하기 위한 입력이며, 20kΩ 가변 저항을 조절하여 사용한다.

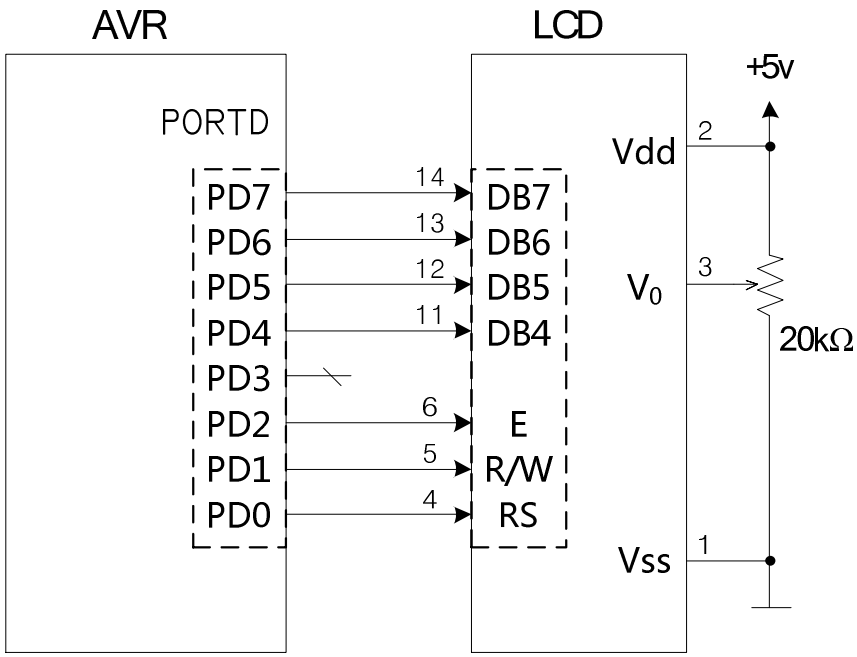


그림 5. 4-bit 모드 LCD 구동회로

1.2.1 첫 번째 LCD 예제

```
#define F_CPU 16000000UL
#include <avr/io.h>            // I/O 레지스터정의
#include <util/delay.h>        // 시간지연함수
#include "LCD.h"               // LCD 제어프로그램 헤더파일

void delay_ms(int n);          // 시간지연함수
static char LCDtitle[] = "Dong-A Univ";

int main()
{
      char i;
      char string[20]; // 문자열을 구성할 배열

      LCDInit();                        // LCD 초기화
      LCDMove(0,0);                     // HOME
      LCDPuts(LCDtitle);                // 실습 제목을 쓴다.

      string[0]  = 'E';
      string[1]  = 'L';
      string[2]  = 'E';
      string[3]  = 'C';
      string[4]  = 'T';
      string[5]  = 'R';
      string[6]  = 'O';
      string[7]  = 'N';
      string[8]  = 'I';
      string[9]  = 'C';
      string[10] = 'S';
      string[11] = '\0';                          // 문자열의 끝

      LCDMove(1,0);
      LCDPuts(string);          // SRAM내의 문자열 출력
```

```
/*
      while(1)
      {
            for(i=0; i<10; i++)
            {
                  delay_ms(1000);                  //1초간 시간지연
                  LCDCommand(DISP_RSHIFT);    // 오른쪽 이동
            }
            for(i=0; i<10; i++)
            {
                  delay_ms(1000);
                  LCDCommand(DISP_LSHIFT);    // 왼쪽 이동
            }
      }
*/
}

void delay_ms(int n)
{
      for(; n >0; n--)                  // 1msec 시간지연을 n회 반복
            _delay_ms(1);               // 1msec 시간지연
}
```

1.2.2 사용자 글꼴 등록 및 LCD에 출력

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

#include "LCD.h"              // LCD 제어 모듈 헤더파일

// 새 글꼴
static char font1[8] ={       0x04,          // 0b00000100
                              0x0E,          // 0b00001110
                              0x1F,          // 0b00011111
```

```
                                0x04,           // 0b00000100
                                0x04,           // 0b00000100
                                0x04,           // 0b00000100
                                0x04,           // 0b00000100
                                0x00            // 0b00000000
                                };

static char font2[8] ={         0x04,           // 0b00000100
                                0x04,           // 0b00000100
                                0x04,           // 0b00000100
                                0x04,           // 0b00000100
                                0x1F,           // 0b00011111
                                0x0E,           // 0b00001110
                                0x04,           // 0b00000100
                                0x00            // 0b00000000
                                };

void main()
{
      char string[20];          // 문자열을 구성할 배열

      LCDInit();                // LCD 초기화

      // 새 글꼴을 등록한다.
      LCDNewchar(1, font1);  // 문자 코드 1에 대한 글꼴 등록
      LCDNewchar(2, font2);  // 문자 코드 2에 대한 글꼴 등록

      // 새 글꼴을 포함한 문자열을 구성한다.
      string[0]  = 1;                   // 문자 코드 1
      string[1]  = ' ';
      string[2]  = 'D';
      string[3]  = 'o';
      string[4]  = 'n';
      string[5]  = 'g';
      string[6]  = 'A';
```

```
        string[7]  = ' ';
        string[8]  = 2;                    // 문자 코드 2
        string[9] = '\0';                  // 문자열의 끝

        LCDMove(1,1); LCDPuts(string);

        while(1);
}
```

1.2.3 A/D 변환 결과를 LCD에 출력

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include "LCD.h"                           // LCD 모듈


#define N_CHANNEL 2                        // 2채널을 사용한다.


void delay_ms(int n);                      // 시간지연함수
void ADCInit();                            //AD Conversion초기화
unsigned int ADConversion(unsigned char chan);        // AD 변환


int main()
{

        unsigend int adresult[N_CHANNEL];
        char     string[20];
        unsigned char  i;

        LCDInit();                         // LCD 초기화

        //DDRC =0B11111111;
```

```
	//PORTC=0B00000000;

	ADCInit();

	while(1)
	{

		for(i=0; i<N_CHANNEL; i++)
		{
			adresult[i] = ADConversion(i);
		}

		//
		// LCD에 각 채널의 변환값을 표시한다.
		//
		sprintf(string, "0:%4d", adresult[0]);
		LCDMove(0,0); LCDPuts(string);

		sprintf(string, "1:%4d", adresult[1]);
		LCDMove(0,8); LCDPuts(string);

		delay_ms(50);			// 시간 지연
	}
}

void delay_ms(int n)
{
	for(; n >0; n--)			// 1msec 시간지연을 n회 반복
		_delay_ms(1);			// 1msec 시간지연
}
```

```
void ADCInit()
{
      DDRF &= ~((1<<PF0) |(1<<PF1));   //PF0..1은 입력으로 설정
      // 프리스케일 비를 128로 하고 ADC를 사용하도록 한다.
      ADCSRA = (1<<ADEN) | (7 << ADPS0);      //A/D 속도 설정
      delay_ms(10);
}


unsigned int ADConversion(unsigned char chan)
{
      //       ADC 멀티플렉서 설정
      //  - 기준전압으로 외부 5.0V 사용,
      //  - 결과를 레지스터에 오른쪽 정렬하여 저장.
      //  - 싱글엔드입력으로 i-번째 ADC채널 설정.
      ADMUX = (1 << REFS0) | (char)chan;
      // ADMUX = (3 << REFS0) | (char)chan;  //기준전압 내부2.56v

      // AD 변환을 시작한다.

      ADCSRA |= (1 << ADSC);         // AD 변환시작
      while(!(ADCSRA & (1<<ADIF)));  //AD 변환이 종료될 때까지
                                     //기다린다.
      ADCSRA &= 0B11101111;          // ADIF 플래그를 지운다.

      // 변환 결과를 읽는다.
      return ADC;
}
```