

KD-128Pro를 이용한 ATmega128 기초 및 응용

N.T.C(뉴테크놀로지컴패니) 연구실

<http://www.NTC.co.kr>

CONTENTS

1장 AVR 실습 환경 구축	1
1-1 KD-128Pro를 이용한 개발	2
1-2 AVR 설치	4
1-3 ICCAVR 실행	8
2장 ICCAVR C 프로그래밍 기초	16
2-1 2진수와 16진수, 정수 데이터 형	17
2-2 비트 연산	17
2-3 인터럽트와 인터럽트 서비스 루틴	19
2-4 C언어의 전역 변수와 지역 변수	20
2-5 여러개의 파일로 이루어진 프로그램	20
2-6 레지스터	21
2-7 ISR의 전역변수 액세스 문제	22
3장 AVR특징 및 입·출력(I/O) 포트	25
3-1 마이크로 컨트롤러 AVR	26
3-2 ATmega128의 입출력(I/O)포트	26
3-3 입출력(I/O) 포트 제어	28
3-4 포트의 기본설정 예제	30
4장 LED(Light Emitted Diode) 실습	31
4-1 LED 기본 회로	32
4-2 LED 출력 모듈	32
4-3 LED 출력 실습	33
5장 스위치 및 키 매트릭스 실습	37
5-1 스위치 회로와 동작 원리	38
5-2 AM-TS8 모듈과 회로도	39

5-3 스위치 실습	39
5-4 키 매트릭스 회로와 동작 원리	43
6장 7-세그먼트	45
6-1 7-세그먼트의 제어와 회로도	46
6-2 7-세그먼트 실습	48
7장 캐릭터 LCD(Liquid Crystal Display) 실습	52
7-1 디지털 입출력 포트에 의한 LCD 제어	53
7-2 LCD 실습	57
8장 UART 직렬통신	63
8-1 직렬 통신의 개요	64
8-2 UART 통신 관련 레지스터와 사용법	65
8-3 시리얼 통신 만들기 - ICCAVR	70
8-4 UART 심화 실습	73
9장 DC MOTOR 실습	77
9-1 DC Motor의 개요	77
9-2 DC 모터의 가변속 제어법	78
9-3 DC 모터의 위치 및 속도 제어	79
9-4 소형 DC 모터 구동보드(AM-CD2L)	80
9-5 DC Motor 실습	83
10장 STEPPING MOTOR	86
10-1 스텝핑 모터란?	86
10-2 스텝핑 모터 구동하기	86
10-3 스텝핑 모터의 구동부	86
10-4 구동 방식의 종류	87
10-5 구동 함수	90
10-6 스텝핑 모터 구동 드라이브(AM-CS2P)	92

10-7 스텝핑 모터 구동 실습	94
11장 SENSOR & ADC 실습	97
11-1 센서	98
11-2 ADC	100
11-3 ADC의 동작	101
11-4 ADC 관련 레지스터	103
11-5 AM-IRS 모듈 소개	105
11-6 ADC 실습	107
12장 스피커로 소리 내기	111
12-1 스피커로 사운드 내기(원하는 주파수)	111
12-2 소리내기 예제	112

1. AVR 실습 환경 구축

이 장에서는 이 교재의 실습을 위해 필요한 하드웨어와 소프트웨어에 대해 학습한다.

학습 목표

- 1, Avr 개발 키트인 **KD-128Pro**를 이용한 개발 환경을 이해한다.
- 2, AVR 개발 도구인 **IccAVR**을 설치할 수 있다.
-ICC AVR Download Site.
http://www.imagecraft.com/pub/icc7avr_demo.exe
<http://www.imagecraft.com/software/>
- 3, IccAVR을 설치 후 개발환경을 구축할 수 있다.

1.1 KD-128Pro를 이용한 개발..

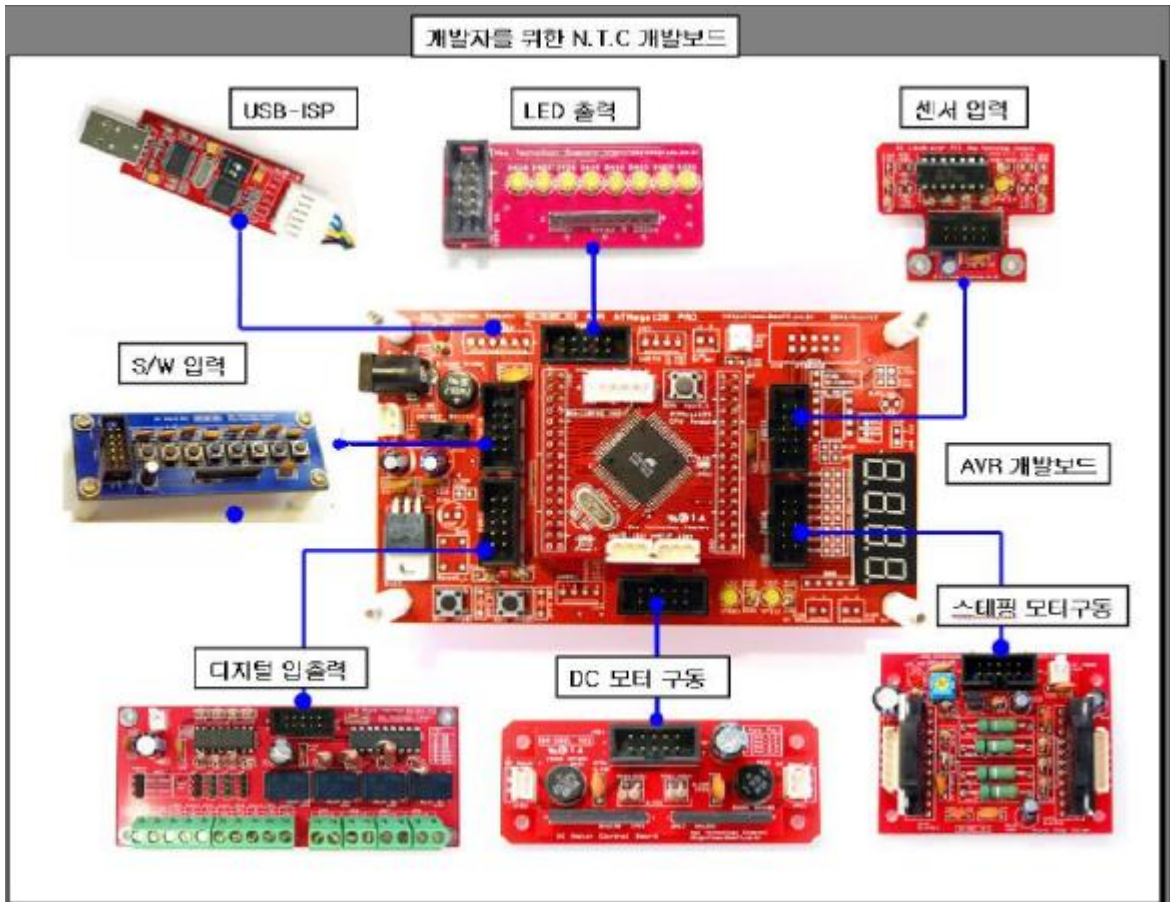


그림 1.1 KD-128Pro 개발보드

[KD-128Pro 개발용 키트 소개 I]

#. 개발보드 특징

- ATMEL 사의 AVR 마이크로 컨트롤러 **ATMega128** 사용
- AM-128PRO MCU 보드 연결하여 사용가능
- 16Mhz X-tal 사용
- 외부전원 12V 입력. 내부동작 5V 단일 전원

- ATMega128**의 모든 포트(**PORTA - PORTF**)가 **10 Pin HeaderBox**로 연결되어 있음.
- AM-128PRO**에 내장된 시리얼 포트에 디버깅 가능

[KD-128Pro 개발용 키트 소개 II]

#. 제품 사양

- MCU : ATMega128-16AI (64핀 TQFP 타입)**
- 클럭 : **16MHz** 클럭 내장
- 크기 : **124mm * 75mm**
- 전원 : **9~12V DC** 입력
- 4Digit 7-Segment** 내장
- S/W 2EA** 내장
- 동작 확인 **LED (2EA)** 내장
- 6핀 ISP** 단자 내장
- AVR-JTAG** 연결 커넥터 내장
- RTC DS1307** 커넥터 내장
- 전원 보호 회로 내장

[KD-128Pro 개발용 키트 소개 III]

#KD-128PRO 개발키트 구성품

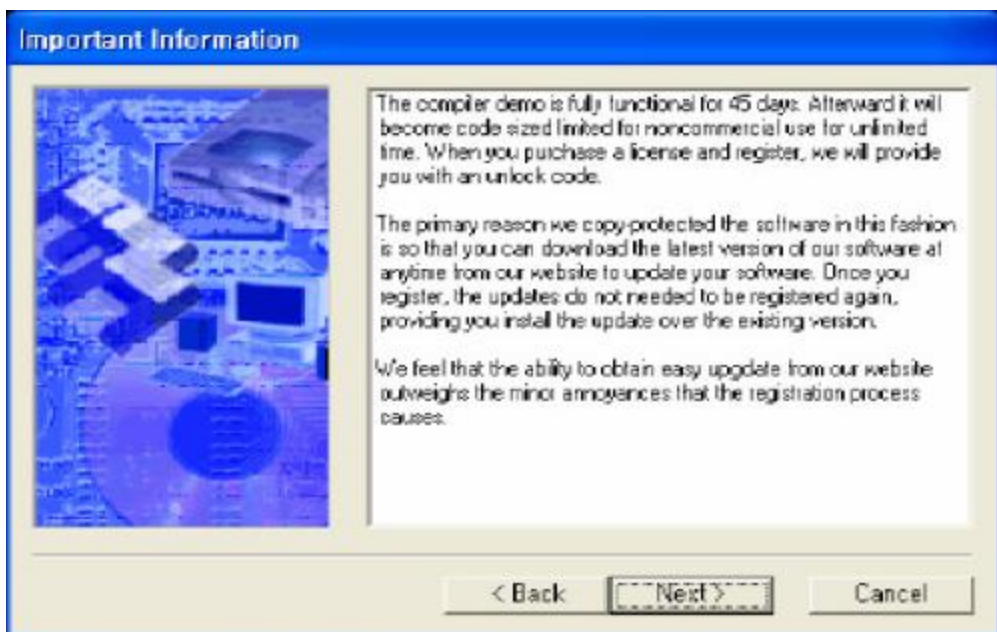
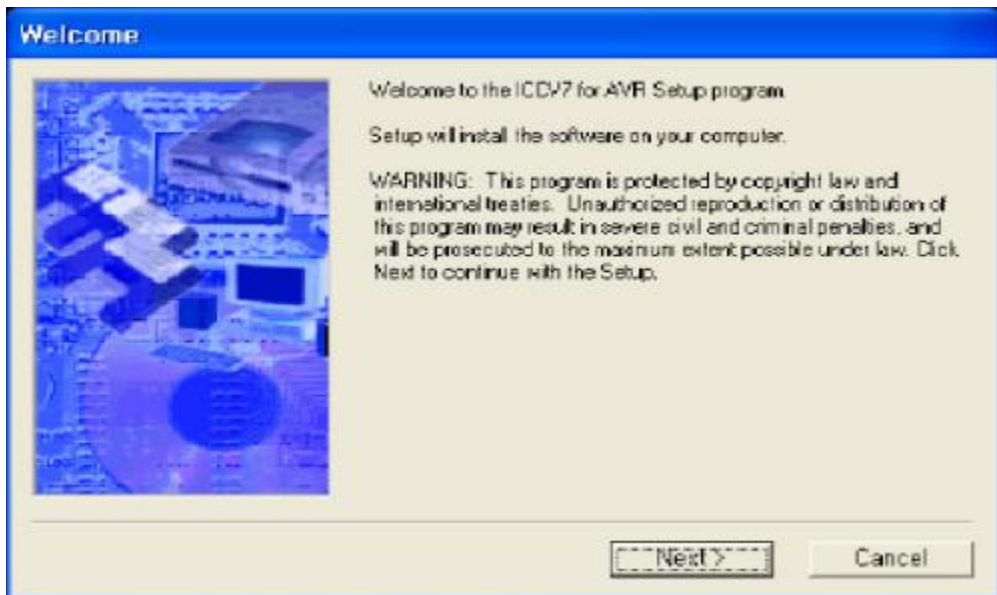
- ATMega128** 모듈 (**AM-128PRO**)
- ATMega128** 개발보드와 부품
- 12V** 아답타
- 시리얼 케이블(**SE-3T9**)
- ISP** 케이블(**AD-ISPPRO**)
- 기구 고정물 (서포터, 나사, 너트 등)

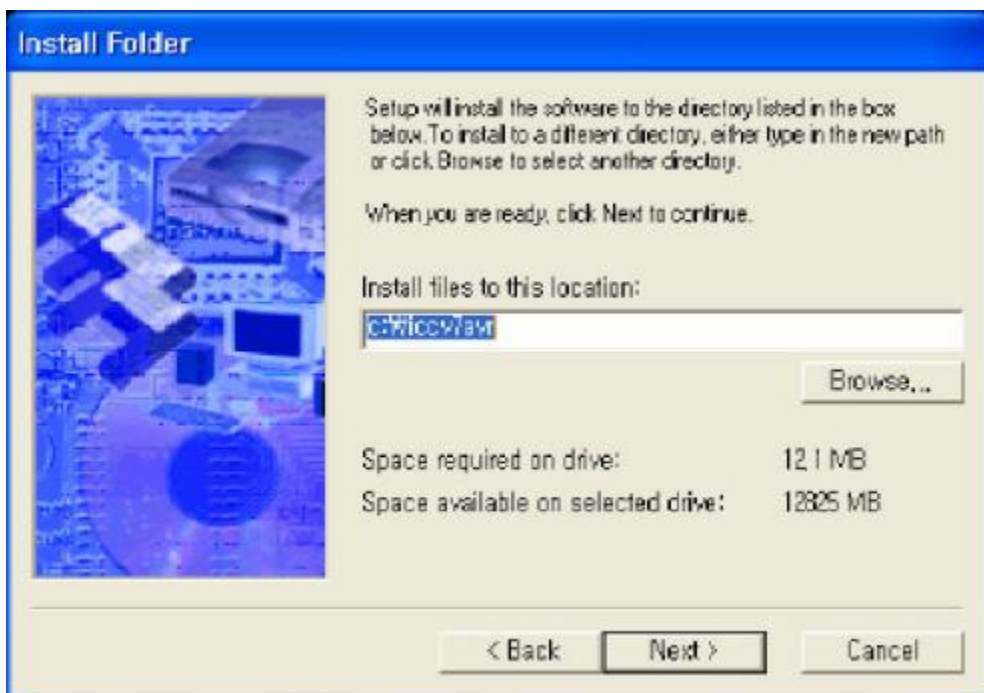
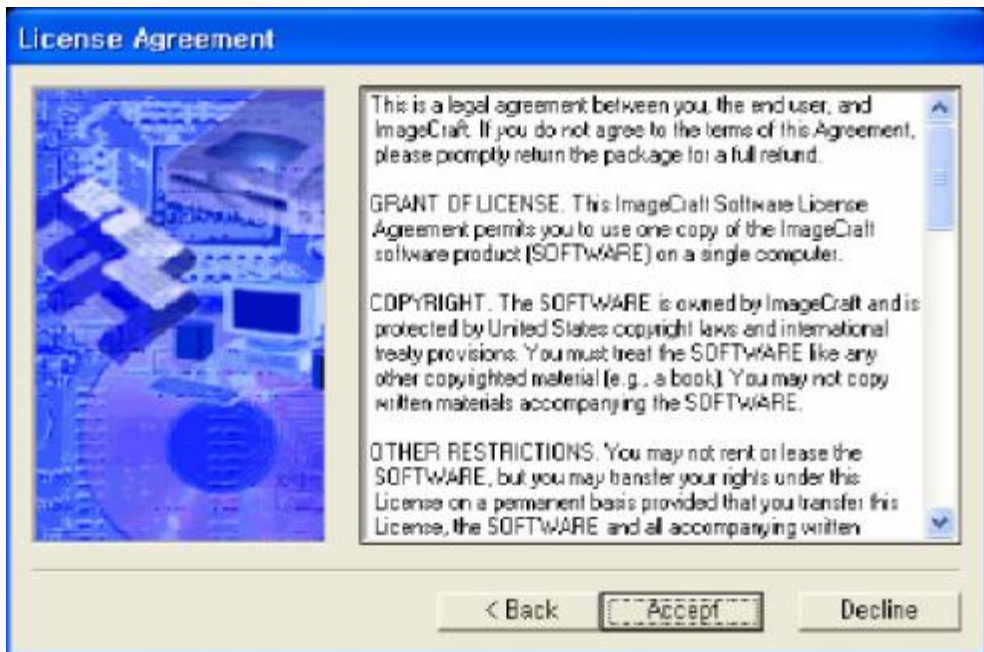
※기타 내용은 N.T.C 사이트 참조

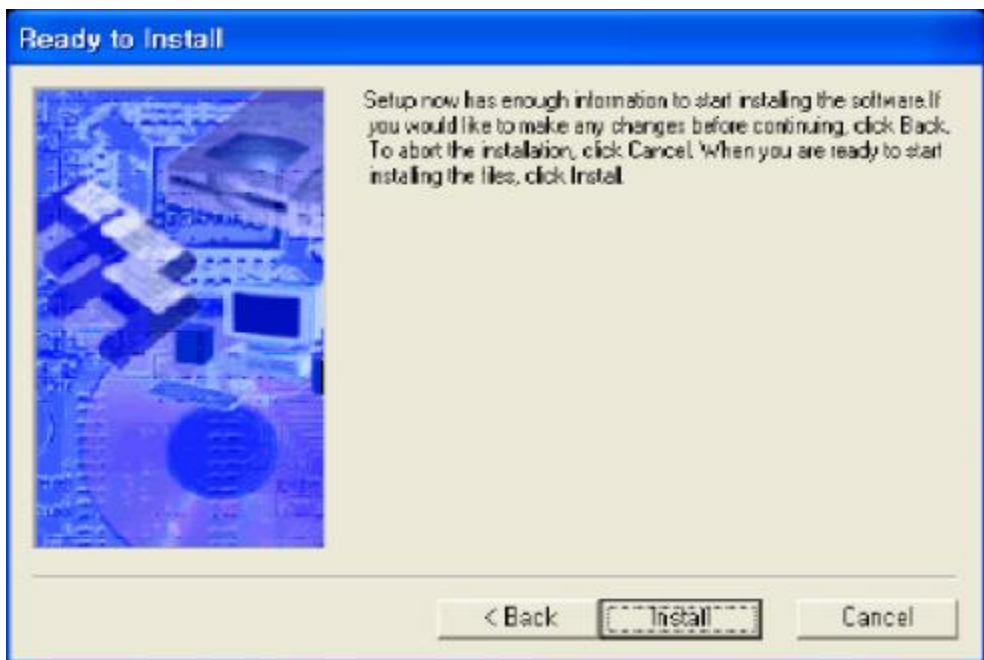
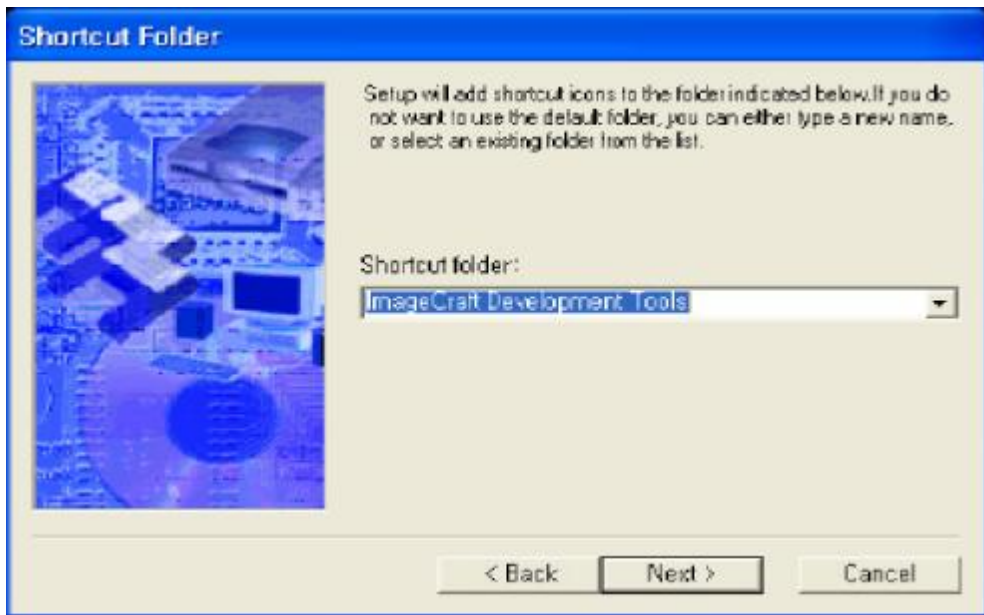
<http://www.NTC.co.kr>

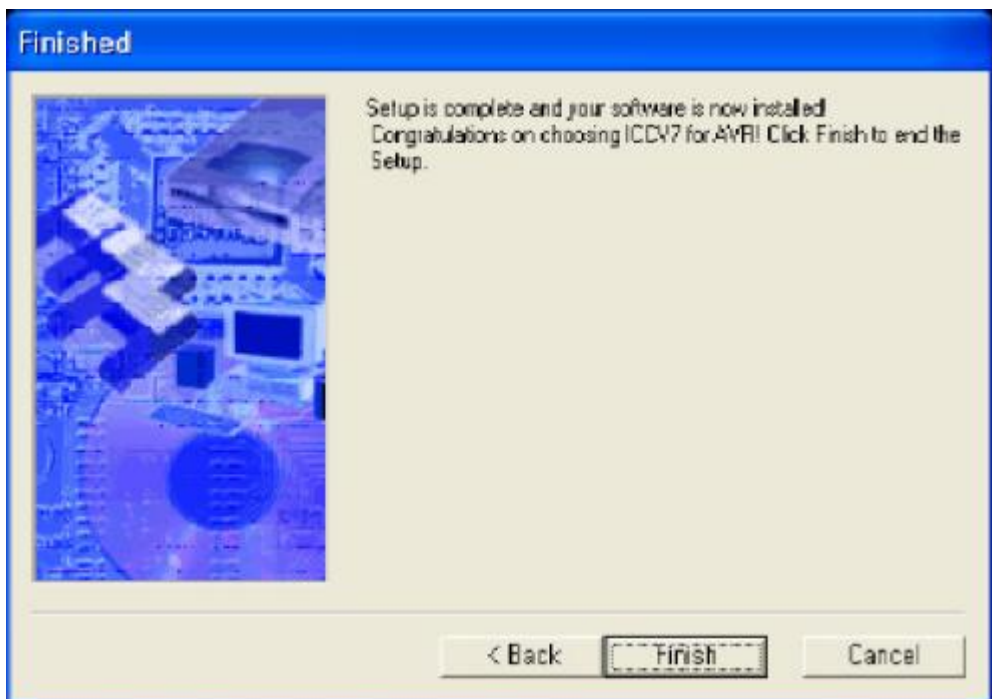
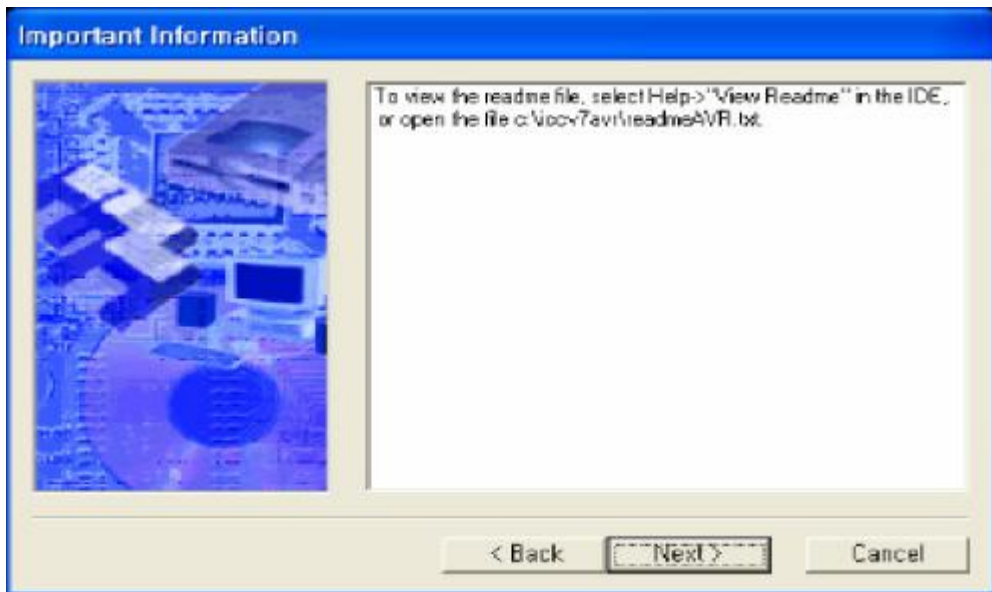
1.2 ICCAVR 설치

ICCAvr을 다운받아 실행시켜 기본설정으로 설치하면 된다.





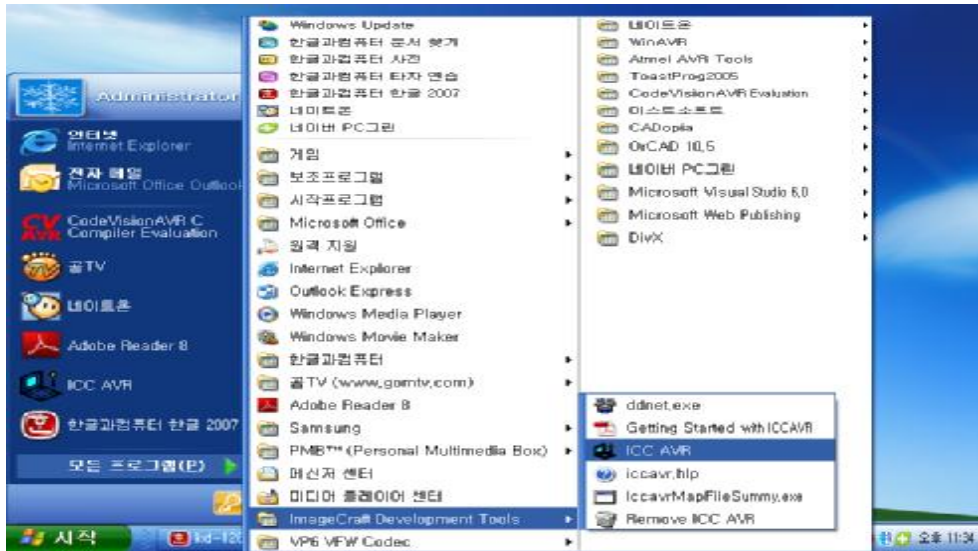




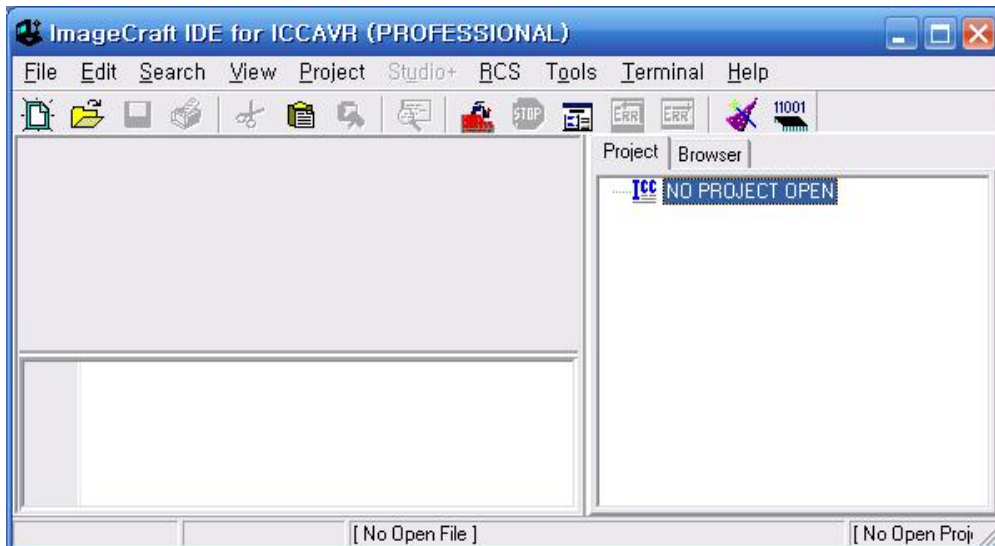
설치가 완료되면 프로그램 메뉴에 등록이 될 것이다.

1.3 ICCAVR 실행

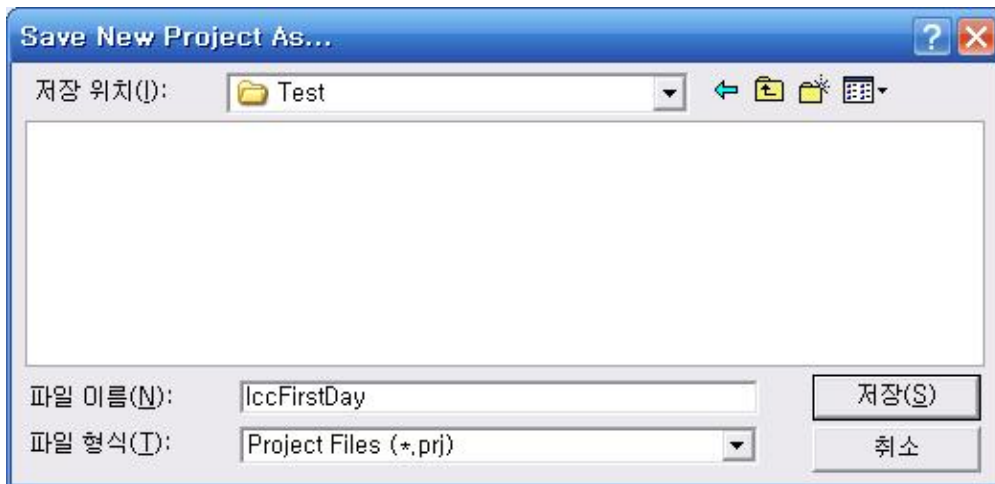
시작 메뉴에서 모든 프로그램 >> ImageCraft Development Tools > ICC AVR 을 클릭하여 IccAVR을 실행시킨다.



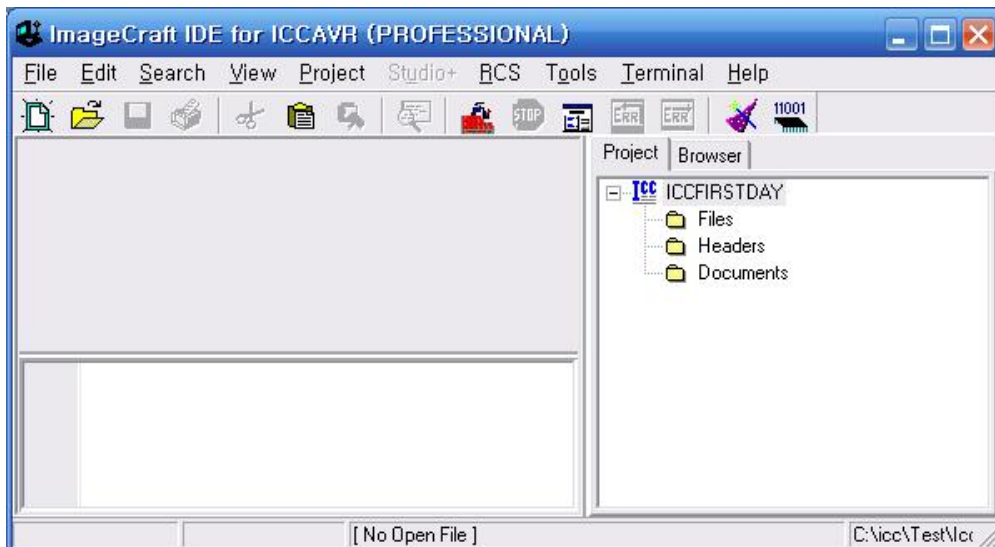
아래와 같은 IccAVR 초기화면이 뜰 것이다.



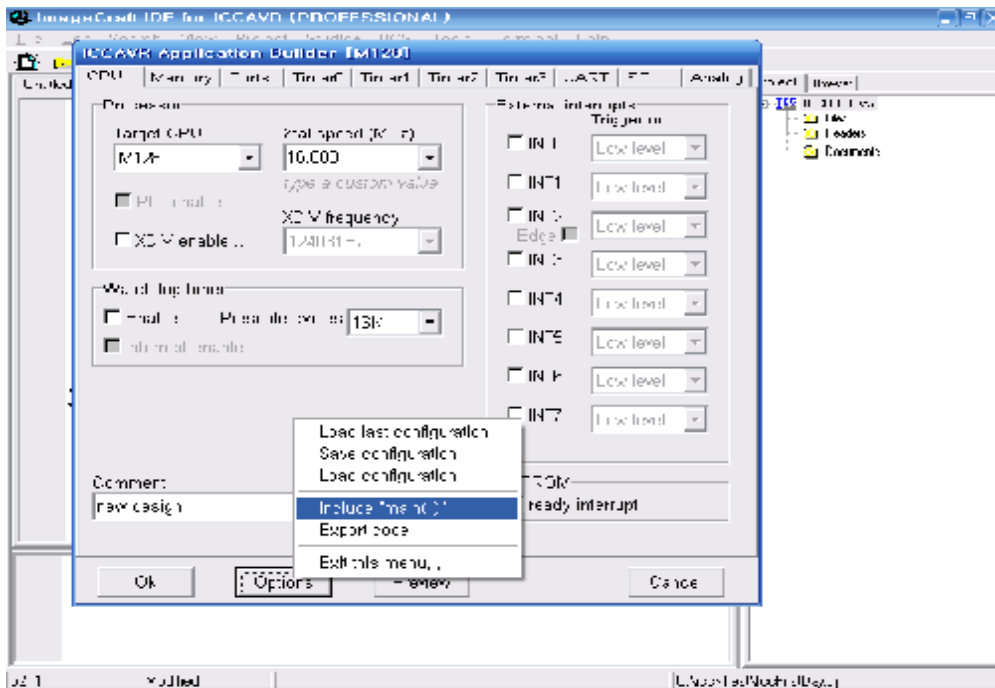
ICCAVR은 프로젝트를 먼저 만들고 C 소스파일을 첨부(ADD)해서 컴파일 하도록 되어있다. 자 그림 하나씩 따라해보자.



Project >> New를 Click하면 프로젝트 생성을 위한 창이 뜨는데 저장 위치를 지정한 후 적당한 파일 이름을 정하고 저장을 Click한다. 프로젝트 이름은 프로그램의 특징을 알 수 있도록 정하는 것이 좋다. IccFirstDay라는 이름을 정한 것은 오늘이 첫날이라는 의미 ^_^. 좋은 코딩의 습관은 이름부터....



위의 그림과 같이 오른쪽 창에 ICCFIRSTDAY라는 프로젝트가 생성된 것을 확인할 수 있다. 이제 그림 파일을 생성해 보자. 파일 생성을 도와주는 Tools >> Application Builder를 선택한다. 그러면 다음과 같은 설정 창이 뜰 것이다. 오늘은 기본 설정만 하겠다.



Target CPU는 M128로 선택하고 Xtal speed(MHz) > 16.000 설정한다.

- Target CPU는 CPU의 종류를 말하는 것으로 우리는 ATmega128을 사용할 것이므로 M128, Xtal speed(MHz)는 CPU의 주파수로써 1초에 16,000,000의 명령어를 실행할 수 있음을 의미한다.

Options에서 Include "main()"을 선택하고 OK 버튼을 Click.. 그러면 아래와 같이 프로그램 코딩을 위한 기본 뼈대가 생성된다. 자 그럼 기본 설정된 내용을 알아보자.

```
#include <iom128v.h> // 입출력을 위한 헤드파일 포함.
```

```
#include <macros.h> // 매크로정의.. iom128v.h와 macros.h는 항상 포함된다.
```

```
void port_init(void) // 입력,출력포트의 초기화
```

```
{
```

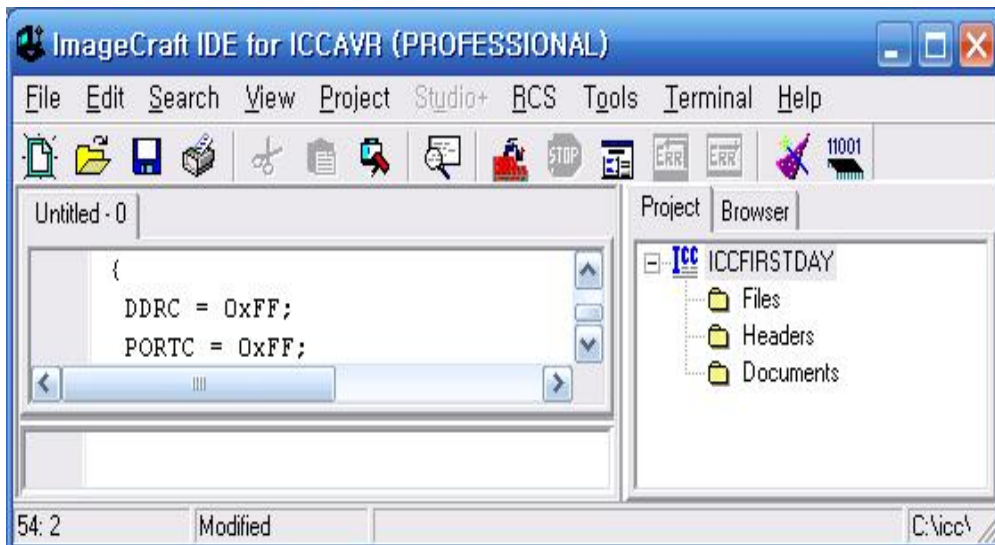
```
.....생략.....
```

```
}
```

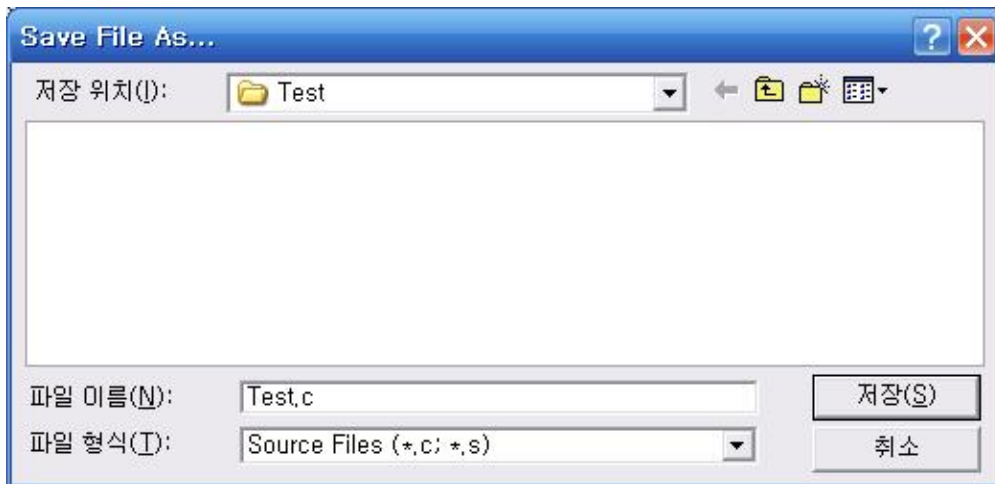
```

void init_devices(void) // 기본설정의 초기화..
{
    .....생략.....
}
void main(void)
{
    init_devices(); //ICCAVR에서는 main()안에 항상 필요하다.
    //insert your functional code here...
}
※ex) //insert... 이부분을 DDRC=0xff;, PORTC=0xff;
void main(){
    init_devices();
    while(1){
        DDRC=0xff; //C포트를 출력으로 설정
        PORTC=0xff; //C포트의 모든 값을 1로 설정
    }
}
# 프로그램 코딩이 끝났으면 이제 파일을 저장시키고 프로젝트에 파일을 포함시켜 컴파일을 해보자.

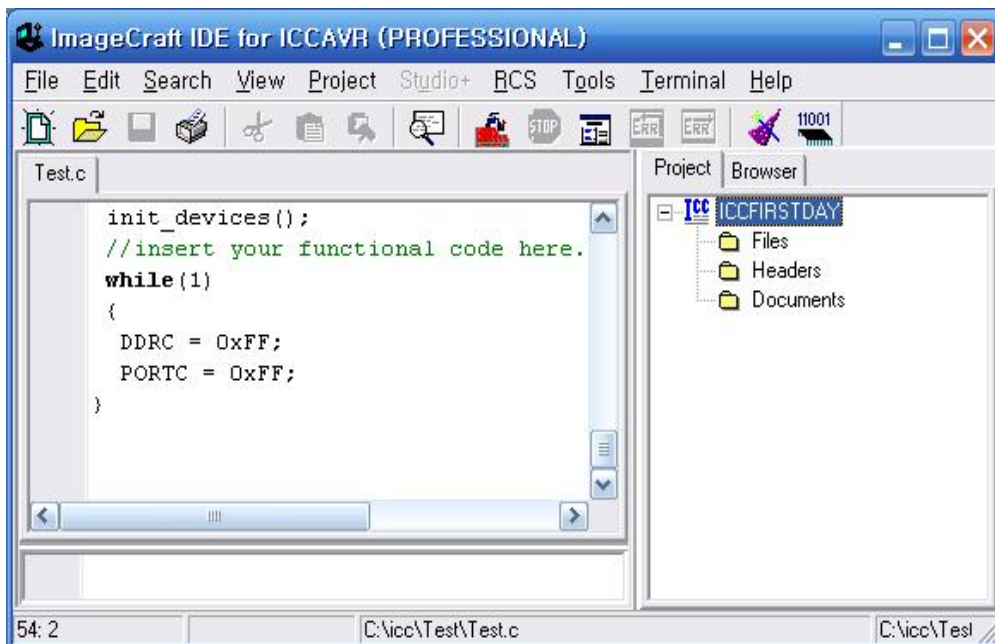
```



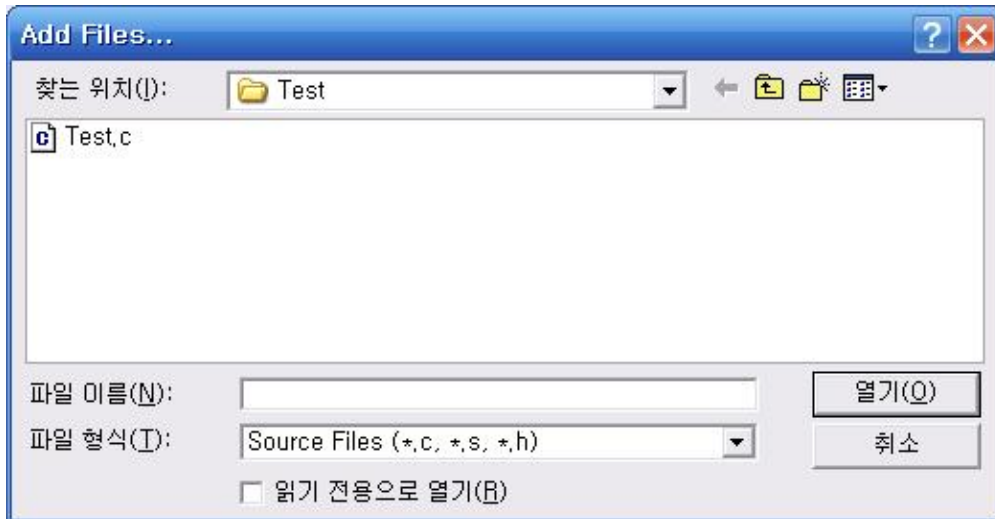
File >> Save Click하면 다음 창이 뜬다.



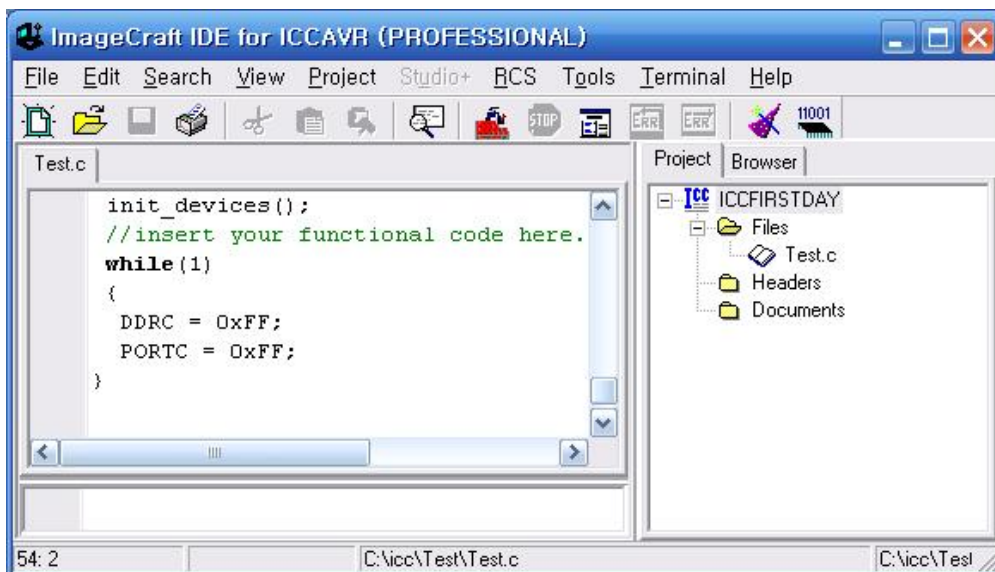
적당한 위치에 파일이름을 정하고 저장(S)을 누른다. 파일이름에는 반드시 .c를 붙여야 한다. 파일의 저장이 끝났으면 이제 저장한 파일을 프로젝트에 추가시켜줘야 한다.



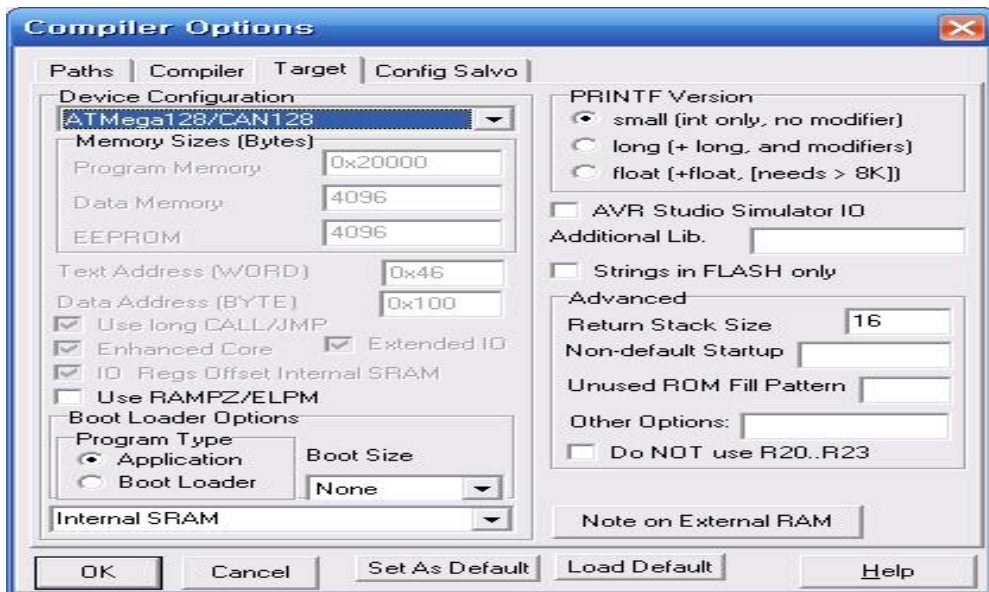
위의 화면에서 오른쪽 프로젝트 창의 프로젝트 이름위에 마우스 포인트를 올리고 오른쪽 마우스를 Click하여 나온 메뉴에서 Add File를 클릭한다.



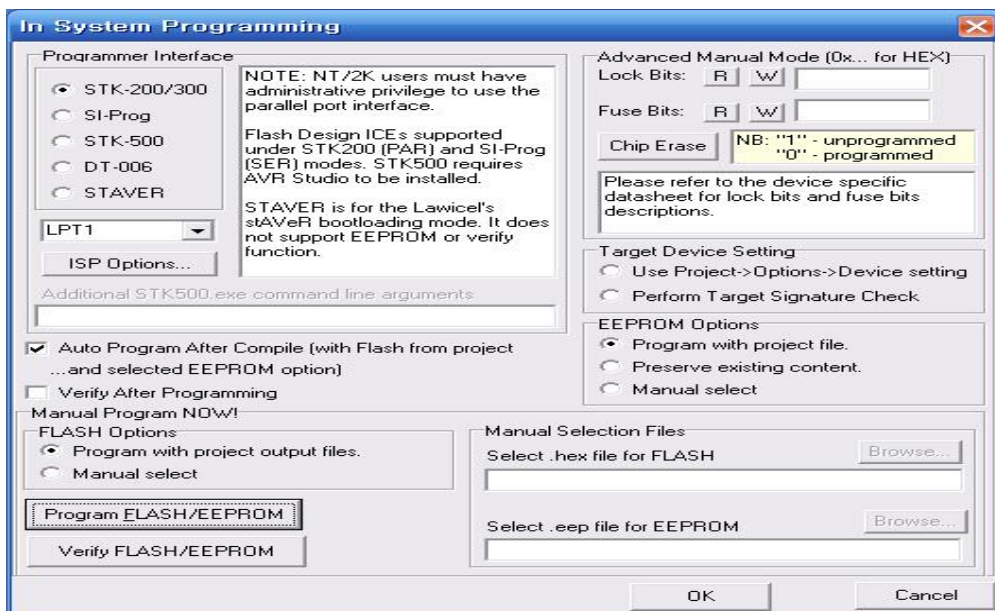
위와 같은 화면에서 방금 저장한 파일이름을 선택해서 열기(O) Click...



위의 그림과 같이 Test.c라는 파일이 프로젝트에 추가된 것을 확인할 수 있다. 다시 위의 오른쪽 프로젝트 창에서 프로젝트 이름위에 마우스 포인트를 올리고 오른쪽 마우스를 클릭해 나온 메뉴에서 Options..를 선택한다. 그러면 다음과 같은 그림이 뜰 것이다.



위 그림에서 Target 탭에서 Device Configuration을 ATmega128/CAN128을 선택하여 OK버튼을 Click한다. 마지막으로 Tools >> In System Programmer Click..



STK-200/300, LPT1, Auto.., Program..을 선택한 후 Program FLASH/EEPROM을 Click하면 프로그램이 라이트 되고 프로그램이 실행되는 것을 볼 수 있다.

만일 프로그램에 에러가 발견되면 라이트가 되지 않고 에러 창이 뜨게 된다. 그러면 에러의 원인을 찾아서 고쳐주고 다시 라이트하면 된다. ICCAVR도 다른 프로그램처럼 많은 단축키가 있으니깐 참고하길 바란다.

- 이로써 AVR을 위한 초기 단계인 컴파일러 설치와 컴파일러 환경설정을 끝냈습니다. 특히 IccAVR실행 부분은 프로그램을 할 때마다 설정해야 하므로 익숙해져야 합니다. 궁금하신 점 있으시면, <http://www.NTC.co.kr>로 방문하셔서 Q&A를 통하여 질문하시거나, 자료실이나 강의 등을 참고하시면 됩니다. 감사합니다.

2. ICCAVR C 프로그래밍 기초

이 장에서는 IccAVR과 C언어로 AVR 프로그램을 작성하기 위해 필요한 기초 내용을 다룰것이다.

학습 목표

1. 2진수와 16진수를 사용할 수 있다.
2. 2진수, 16진수, 10진수 사이의 진법 변환을 할 수 있다.
3. 데이터 형에 따른 숫자 표현 범위를 설명할 수 있다.
4. IccAVR에서 typedef로 재정의가 된 정수형 데이터를 사용할 수 있다.
5. C 언어에서 비트 연산에 관련된 명령어를 사용할 수 있다.
6. 비트 연산을 통해 어떤 데이터의 특정 비트만을 0 또는 1로 만들 수 있다.
7. 비트 연산을 통해 어떤 데이터의 특정 비트가 0인지 1인지 판단할 수 있다.
8. 비트 연산을 통해 어떤 데이터의 특정 비트를 반전시킬 수 있다.
9. 비트 연산을 통해 어떤 데이터의 내용을 시프트 시킬 수 있다.
10. 비트 연산을 통해 어떤 데이터의 특정 비트만을 다른 데이터로 이동시킬 수 있다.
11. 인터럽트의 개념을 설명할 수 있다.
12. 인터럽트 서비스 루틴이 무엇인지 설명할 수 있다.
13. 전역변수를 사용할 수 있다.
14. 선행처리를 사용할 수 있다.
15. 프로그램을 여러 파일로 나누어 작성할 수 있다.
16. 레지스터가 무엇인지 설명할 수 있다.
17. IccAVR에서 레지스터를 액세스할 수 있다.
18. ISR에서 전역변수를 사용할 때의 문제점의 원인과 해결 방안을 설명할 수 있다.

2.1 2진수와 16진수, 정수 데이터 형

2.1.1 2진수와 16진수

10진수	2진수	16진수	10진수	2진수	16진수
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	a or A
3	0011	3	11	1011	b or B
4	0100	4	12	1100	c or C
5	0101	5	13	1101	d or D
6	0110	6	14	1110	e or E
7	0111	7	15	1111	f or F

2.1.2 정수 데이터 형

마이크로컨트롤러용 프로그램을 C 언어로 작성할 때 정수 데이터 형을 주로 사용한다. C 언어에서 정수형은 대개 표현할 수 있는 수의 크기 즉, 메모리 사이즈에 따라 char, short, int, long 등으로 나뉘고, 부호의 여부에 따라 signed와 unsigned로 나뉜다.

그런데, C 언어에서 일부 정수형은 C 언어가 사용되는 OS 또는 컴파일러에 따라 다른 경우가 있다. 예를 들어 대부분의 8비트 또는 16비트 CPU나 마이크로컨트롤러에 사용되는 C 컴파일러에서는 int 형이 두 바이트 크기이지만 윈도우즈에서 사용되는 C 컴파일러에서는 int가 네 바이트이다. 즉, 같은 C 언어로 된 소스라도 컴파일러에 따라 int형의 표현 범위에 차이가 있다. 그래서 많은 개발자들은 C 언어의 정수형 선언을 그대로 사용하지 않고 typedef 명령으로 사이즈가 명확한 형을 만들어 사용한다.

2.2 비트 연산

마이크로컨트롤러용 프로그램을 작성할 때에는 비트 연산이 매우 빈번하게 사용된다. 따라서 C 언어를 이용한 비트 연산에 익숙해질 필요가 있다.

C 언어에서 시프트 연산에는 '<<'(왼쪽 시프트)와 '>>'(오른쪽 시프트)가 사용

되고, 논리 연산에는 '&(AND), '|(OR), '^'(XOR), '~'(NOT)'이 사용된다.

비트 논리 연산					
비트 값		비트 논리 연산 결과			
a	b	a & b	a b	a ^ b	~ a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

'&' 연산은 특정 비트를 아래와 같이 클리어 시킨다.

사이즈가 1바이트인 변수 a의 특정 비트만 0으로 만드는 방법 (변수 a는 B7 B6 B5 B4 B3 B2 B1 B0 으로 구성됨)				
목적	표현 I	표현 II	표현 III	표현 IV
B0만 0으로	a = a & 0xfe	a = a & ~(1<<0)	a &= 0xfe	a &= ~(1<<0)
B1만 0으로	a = a & 0xfd	a = a & ~(1<<1)	a &= 0xfd	a &= ~(1<<1)
B2만 0으로	a = a & 0xfb	a = a & ~(1<<2)	a &= 0xfb	a &= ~(1<<2)
B3만 0으로	a = a & 0xf7	a = a & ~(1<<3)	a &= 0xf7	a &= ~(1<<3)
B4만 0으로	a = a & 0xef	a = a & ~(1<<4)	a &= 0xef	a &= ~(1<<4)
B5만 0으로	a = a & 0xdf	a = a & ~(1<<5)	a &= 0xdf	a &= ~(1<<5)
B6만 0으로	a = a & 0xbf	a = a & ~(1<<6)	a &= 0xbf	a &= ~(1<<6)
B7만 0으로	a = a & 0x7f	a = a & ~(1<<7)	a &= 0x7f	a &= ~(1<<7)

'|' 연산은 특정 비트를 아래와 같이 셋 시킨다.

사이즈가 1바이트인 변수 a의 특정 비트만 1로 만드는 방법 (변수 a는 B7 B6 B5 B4 B3 B2 B1 B0 으로 구성됨)				
목적	표현 I	표현 II	표현 III	표현 IV
B0만 1로	a = a 0x01	a = a (1<<0)	a = 0x01	a = (1<<0)
B1만 1로	a = a 0x02	a = a (1<<1)	a = 0x02	a = (1<<1)
B2만 1로	a = a 0x04	a = a (1<<2)	a = 0x04	a = (1<<2)
B3만 1로	a = a 0x08	a = a (1<<3)	a = 0x08	a = (1<<3)
B4만 1로	a = a 0x10	a = a (1<<4)	a = 0x10	a = (1<<4)
B5만 1로	a = a 0x20	a = a (1<<5)	a = 0x20	a = (1<<5)
B6만 1로	a = a 0x40	a = a (1<<6)	a = 0x40	a = (1<<6)
B7만 1로	a = a 0x80	a = a (1<<7)	a = 0x80	a = (1<<7)

‘^’ 연산은 아래와 같이 특정 비트만 반전시킨다.

사이즈가 1바이트인 변수 a의 특정 비트만 반전시키는 방법 (변수 a는 B7 B6 B5 B4 B3 B2 B1 B0 으로 구성됨)				
목적	표현 I	표현 II	표현 III	표현 IV
B0만 반전	$a = a \wedge 0x01$	$a = a \wedge (1 \ll 0)$	$a \wedge= 0x01$	$a \wedge= (1 \ll 0)$
B1만 반전	$a = a \wedge 0x02$	$a = a \wedge (1 \ll 1)$	$a \wedge= 0x02$	$a \wedge= (1 \ll 1)$
B2만 반전	$a = a \wedge 0x04$	$a = a \wedge (1 \ll 2)$	$a \wedge= 0x04$	$a \wedge= (1 \ll 2)$
B3만 반전	$a = a \wedge 0x08$	$a = a \wedge (1 \ll 3)$	$a \wedge= 0x08$	$a \wedge= (1 \ll 3)$
B4만 반전	$a = a \wedge 0x10$	$a = a \wedge (1 \ll 4)$	$a \wedge= 0x10$	$a \wedge= (1 \ll 4)$
B5만 반전	$a = a \wedge 0x20$	$a = a \wedge (1 \ll 5)$	$a \wedge= 0x20$	$a \wedge= (1 \ll 5)$
B6만 반전	$a = a \wedge 0x40$	$a = a \wedge (1 \ll 6)$	$a \wedge= 0x40$	$a \wedge= (1 \ll 6)$
B7만 반전	$a = a \wedge 0x80$	$a = a \wedge (1 \ll 7)$	$a \wedge= 0x80$	$a \wedge= (1 \ll 7)$

2.3 인터럽트와 인터럽트 서비스 루틴

일반적인 C 프로그램에서는 인터럽트(interrupt)와 관련된 프로그래밍을 하는 경우는 별로 없다. 하지만 마이크로컨트롤러와 같이 직접 여러 장치를 제어하는 프로그램을 작성할 때에는 인터럽트 기능을 적극적으로 활용해야만 좋은 프로그램을 작성할 수 있다.

인터럽트 또는 인터럽트 신호란 마이크로컨트롤러에 하드웨어적으로 어떠한 일이 발생했음을 알리는 것이고, 인터럽트 서비스 루틴(interrupt service routine, ISR)이란 이 인터럽트에 의해 수행되는 루틴을 말한다. 마이크로컨트롤러에서는 인터럽트가 발생하면 실행하려던 명령을 멈추고 인터럽트 서비스 루틴을 실행한 후에 다시 실행하려던 명령을 진행한다.

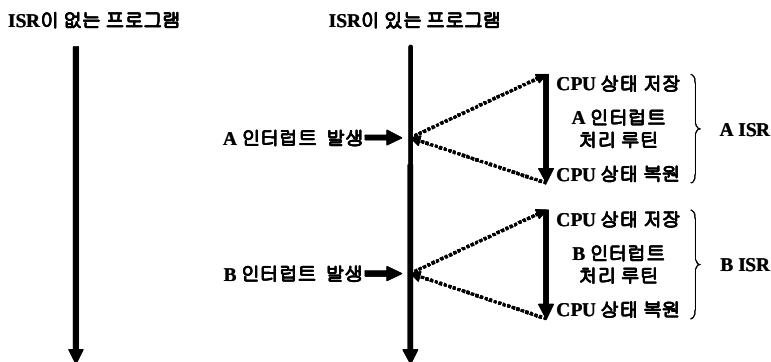


그림 2.1 인터럽트의 개념

2.4 C언어의 전역 변수와 지역변수

AVR 실습을 위해서는 C언어의 전역 변수와 지역변수에 대한 이해가 있어야 한다. 여기서는 예제를 통해 이 항목에 대한 이해를 돕고자 한다.

예제를 통한 전역 변수와 지역 변수

```
int gvar=1;           // 전역변수 gvar를 선언 및 정의하고 1을 대입
                      // 변수선언은 알기 쉬운 약어를 사용하는 것이 좋다.
int funcA(void)       // 함수 funcA정의
{
    int i=3;          // 지역변수 i에 3 대입
    i=i+gvar;          // 지역변수 i(3)에 전역변수 gvar(1)을 더하여 i에 저장
    gvar++;            // 전역변수 gvar 1 증가
    return i;
}

int funcB(void)
{
    int i=4;
    i=4+gvar;
    return i;
}

int main(void)
{
    int a, b;

    a = funcA();       // a 값은 4가 되고, 전역 변수 gvar은 2가 됨

    b = funcB();       // b 값은 6이 됨
    gvar = a+b;        // 전역 변수 gvar은 다시 10이 됨

    return 0;
}
```

2.5 여러 개의 파일로 이루어진 프로그램

프로그램을 하나의 파일로만 작성할 수도 있지만, 프로그램의 덩치가 커지는 경우에는 프로그램을 기능별로 나누어 각각 다른 파일로 관리하는 것이 여러 모로 편리하다. 물론 프로그램을 여러 파일로 쪼개서 작성하는 것이 처음에는

어렵고 귀찮은 일일 수 있다. 하지만, 프로그램을 여러 파일로 나누어 작성하면 프로그램을 보다 구조적으로 작성할 수 있기 때문에 전체 프로그램 작성 시간이 단축되고 오류도 줄어들게 된다.

프로그램을 여러 파일로 나눌 때, C 언어를 예로 들면, 확장자가 h인 파일과 c인 파일이 한 쌍이 된다. 예를 들어 test에 관련된 프로그램을 하나의 파일로 만들고 싶다면 test.h와 test.c가 한 쌍이 된다. 그러면 test.c에는 프로그램 본체를 작성하고, test.h에는 test.c에 사용되는 변수, 함수, 상수 중 다른 파일에서 참조가 필요한 경우에 대해서 선언을 한다. 예를 들어 test.c에 함수 tfunc()를 사용하고, 이를 main.c에서 사용한다고 하면, test.h와 test.c는 다음과 같은 형태를 가져야 한다.

여러 개의 파일로 프로그램을 작성할 때의 프로그램 기본 구조 예	
test.h의 내용	test.c의 내용
<pre>#ifndef __TEST_H_ // 무조건 이런 식으로 #define __TEST_H_ // 무조건 이런 식으로 int tfunc(int n); // 앞에 extern 붙여도 됨 #endif // 앞 #ifndef와 한 조</pre>	<pre>#include "test.h" int tfunc(int n) { return n*n; }</pre>
<pre>// main.c에서는 앞 부분에 #include "test.h" 를 써주고, // test.c의 tfunc() 함수를 사용하면 된다. #include "test.h" int main(void) { int result; result = tfunc(3); return 0; }</pre>	

2.6 레지스터

마이크로컨트롤러에는 내부의 여러 장치를 제어하기 위해 사용하는 특수한 메모리가 존재한다. 이 메모리를 I/O 레지스터라고 부르기도 하고 줄여서 레지

스터라고도 한다.

마이크로컨트롤러의 I/O 레지스터는 마이크로컨트롤러에 내장된 다양한 장치를 제어하기 위해 사용하는 메모리이기 때문에 마이크로컨트롤러 응용 프로그램 작성 시 빈번하게 사용하게 된다. AVR의 레지스터는 제조사에서 그 의미에 걸맞은 이름을 붙여 데이터시트에 명시하고 있고, IccAVR는 C 언어에서 이 이름을 그대로 사용할 수 있도록 매크로 상수를 제공한다. 이 매크로 상수를 사용하기 위해서는 프로그램 소스의 선언부에 '#include <iom128v.h>'와 '#include <macros.h>'를 포함시키면 된다.

예를 들어 AVR의 포트A를 출력으로 사용하기 위해서는 프로그램 소스의 선언부에 '#include <iom128.h>'와 '#include <macros.h>'를 소스에 포함시키고 PORTA를 마치 전역 변수처럼 사용하면 된다.

IccAVR의 레지스터 사용 예

```
#include <iom128v.h>
#include <macros.h>
int main(void)
{
    init_devices();
    PORTA = 0xff; //PORTA를 전역변수처럼 사용했다.
    return 0;
}
```

레지스터의 각 비트에 대한 매크로의 경우에도 AVR 데이터시트에 사용되는 비트 이름이 거의 그대로 사용된다. 이 때 비트에 대한 매크로는 비트의 값이 아니라 비트 번호이다.

2.7 ISR의 전역변수 액세스 문제

인터럽트를 사용하는 프로그램을 작성할 때 주의해야 할 점이 크게 두 가지가 있다. 첫째는 ISR 실행 시간이고 둘째는 전역변수 값 변경이다.

ISR 실행 시간은 되도록 짧게 해야 한다. 어느 정도 짧게 해야 하느냐는 경우에 따라 다르겠지만 대략적으로 말하자면 프로그램에서 인터럽트를 제외한 다른 부분이 원활하게 실행될 정도로 짧게 해야 한다.

인터럽트를 사용하는 프로그램에서 전역변수를 ISR 함수와 일반 함수에서 동시에 사용할 경우에는 반드시 주의가 필요하다.

2.7.1 컴파일러에 의한 최적화 문제

인터럽트를 사용하는 프로그램을 C 언어로 작성할 때 전역변수의 사용에 있어서 주의해야 할 점이 한 가지 더 있다. 이는 C 컴파일러 최적화에 의한 오류이다. 컴파일러가 C 언어를 기계어로 번역할 때 코드 사이즈를 줄이거나 실행 속도를 향상시키기 위해서 다양한 최적화를 수행하는데 이 때 인터럽트와 관련된 전역변수의 사용은 컴파일러가 최적화를 할 때 문제가 발생할 수 있다.

다음의 표에 있는 소스를 예로 들면, 좌측 프로그램은 일반적인 경우 절대 if 문 안의 명령어가 실행될 수 없다. 따라서 이러한 경우에는 컴파일러가 번역을 할 때 아예 이 if 문에 관련된 부분을 최적화에 의해 생략해 버릴 수 있다. 하지만 변수 flag가 인터럽트를 처리하는 ISR에 의해 0이 아닌 값을 가질 수 있는 경우에는 if 문 안의 명령어가 실행될 수 있다. 이러한 경우에는 컴파일러에 이 부분에 대해서는 최적화를 하지 않도록 알려 주어야 한다. 이 때 사용하는 지시어가 volatile이다. 오른쪽 예에서와 같이 변수를 선언할 때 데이터 형 앞에 volatile을 붙여 주면 컴파일러는 이 변수와 관련된 부분에 대해서 임의로 최적화를 하지 않는다. 그래서 오른쪽 예에서의 if 문은 컴파일러가 번역할 때 절대로 생략하지 않는다.

C 컴파일러의 최적화에 의한 오류	
최적화에 의해 잘못되는 경우	최적화가 있어도 제대로 실행되는 경우
<pre> char flag=0; char a=0; ISR(INT0_vect) { flag = !flag; } int main(void) { flag=0; while (1) { if (flag==1) // 무시 { a=a*2; // 무시 } } return 0; } </pre>	<pre> volatile char flag=0; char a=0; ISR(INT0_vect) { flag = !flag; } int main(void) { flag=0; while (1) { if (flag==1) { a=a*2; } } return 0; } </pre>

2.7.2 ISR에서 전역변수를 사용하는 방법

앞서 설명한 문제점의 해결 방법을 정리하면 다음과 같다.

ISR에서 사용하는 전역변수를 선언할 때에는 반드시 `volatile`을 사용한다.

AVR에서 ISR을 사용하는 경우에 따라야 할 규칙 정리	
규칙	사용 예
1. ISR에서 사용하는 전역 변수의 선언에는 반드시 <code>volatile</code> 을 앞에 붙인다.	<pre>#include <iom128v.h> #include <macros.h> volatile char cnt=0; // 반드시 // volatile을 붙인다. ISR(INT0_vect) { cnt++; //ISR내에서 쓰는 변수는 무조건 //volatile을 붙이자. 아주중요... } void main(void) { init_devices(); char a; cnt=cnt+3; a=cnt; //a값은 얼마일까요?^_^ }</pre>

- 이로써 ICCAVR 사용을 위한 C언어의 기초를 끝냈습니다. C언어의 경험이 없으신 분들은 C언어 서적을 통해 프로그래밍을 이해하시기 바랍니다. 궁금하신 점 있으시면, <http://www.NTC.co.kr>로 방문하셔서 Q&A를 통하여 질문하시거나, 자료실이나 강의 등을 참고하시면 됩니다.
감사합니다.

3. AVR특징 및 입·출력(I/O) 포트

이 장은 우리가 사용할 AVR의 특징 및 입출력 포트에 대해 설명한다. 이 중 입출력 포트의 이해는 앞으로 우리가 실습을 위해 반드시 이해해야 한다. AVR의 기초실습은 이번 장에서 배울 입출력 포트들을 자유롭게 사용하는데 있다고 해도 과언이 아니다. 이 장에서는 ATmega128의 입출력 포트에 대해서 살펴보고도록 하겠다.

학습 목표
1. 마이크로 컨트롤러 AVR의 특징을 이해한다.. 2. ATmega128의 특징과 구조에 대해 이해한다. 3. 입출력(I/O) 포트의 특징및 사용법을 익힌다. - 각각의 포트들의 역할과 기능을 익힌다.

3.1 마이크로 컨트롤러 AVR

n 8비트 RISC 구조로 명령어가 간단하며 동작 속도가 빠르고 16MHz로 동작

- 1MHz당 약 1MIPS의 성능

n 향상된 RISC 구조

- 133개의 강력한 명령어들이 대부분 단일 클럭으로 실행
- 32 X 8 범용 동작 레지스터 + 주변 콘트롤 레지스터로 구성
- 16MHz 환경에서 16 MIPS 이상의 성능
- 2 사이클 곱셈기를 칩에 내장

n 비휘발성 프로그램과 데이터 메모리

- 128K Byte In-System 프로그램 플래쉬 메모리 (내구성 : 최대 10,000번 쓰고 지울 수 있다.)
- 독립 락 비트를 지원하는 옵션 부트 코드 섹션
- 칩에 내장된 부트 프로그램으로 In-System 프로그래밍 가능
- 읽고 쓰는 동작을 동시에 수행가능
- 4K Byte EEPROM
- 4K Bbyte 내장형 SRAM
- 최대 64K Byte 이상의 외부 메모리 영역을 갖는다.
- 소프트웨어의 보안을 위한 프로그램방식의 락을 설정할 수 있다.
- In-System 프로그램을 위한 SPI 인터페이스방식을 지원

n JTAG (IEEE std. 1149.1 호환) 인터페이스 제공

- JTAG 표준에 근거한 Boundary-scan 제공
- 방대한 칩 내장형 디버그 제공
- JTAG 인터페이스로 플래쉬, EEPROM, 퓨즈와 락 비트를 프로그래밍

가능

3.2 ATmega128의 입출력포트

다음 그림에서 보듯이 ATmega128은 외부 포트가 7개 있습니다. 포트 A부터 F까지는 8비트 양방향성 I/O 포트이고 포트 G는 5비트 양방향성 I/O포트로 구성되어 있습니다. 각 포트는 하이(H)상태의 출력 소스전류와 로(L)상태의 출력 싱크전류가 동일하며, 최대 구동전류는 20mA로서 LED를 직접 구동할 수 있습니다. 각 포트는 3개의 레지스트 (DDRx, PORTx, PINx, -x는 A, B, C, D, E, F, G)를 가지고 있습니다. DDRx(Data Direction Register) 레지스터는 포트의 입출

력 방향을 설정합니다.

비트 값이 0일 때는 입력이고 1일 때는 출력으로 설정됩니다. PORTx(Data Register) 레지스터는 데이터를 출력할 때 사용하고, PINx(Port Input Pin) 레지스터는 데이터를 입력할 때 사용합니다.

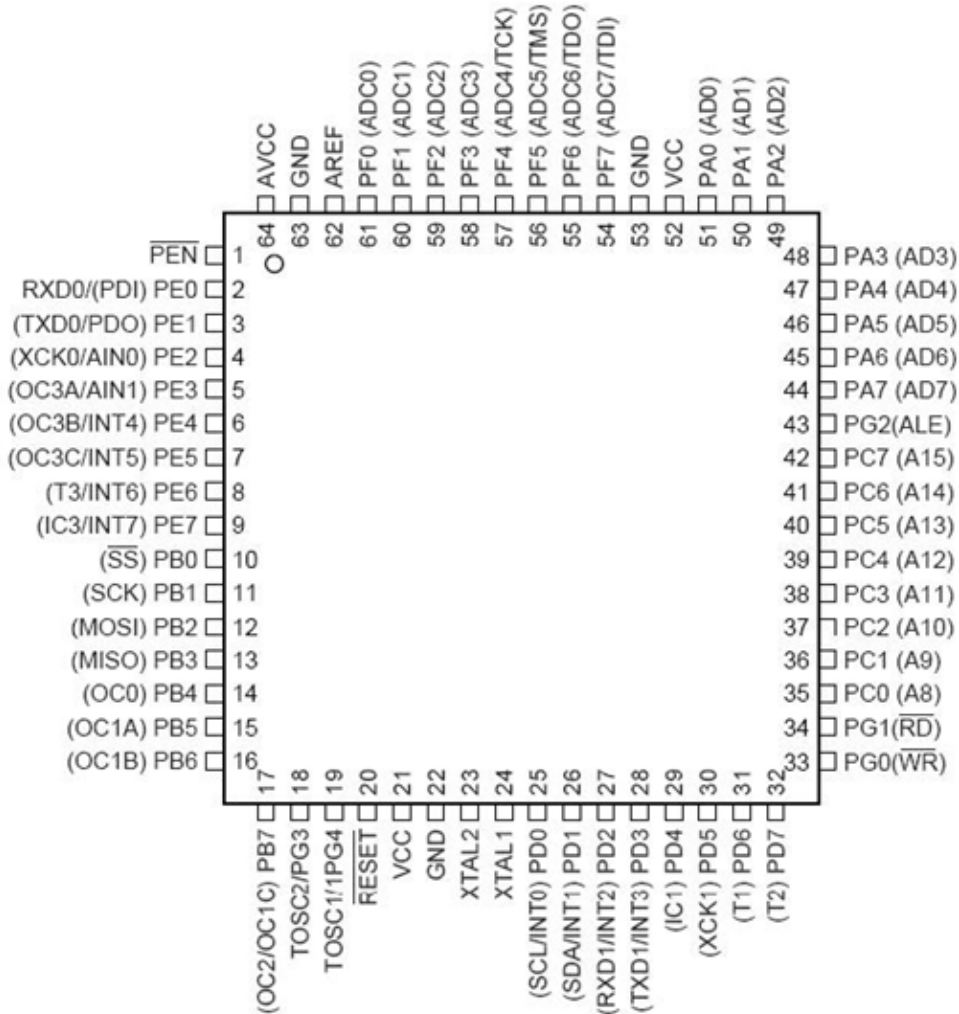


그림 ATmega128 외부 핀 구조

n VCC(21,52) - 마이크로프로세서가 동작하도록 전원 공급

n GND(22,53,63) - 신호의 접지 핀으로 사용

n XTAL1, XTAL2(23,24) - 수정 발진기의 입/출력 핀으로서 크리스탈이나 오실레이터를 접속

n RESET(20) - 리셋 입력 핀으로서 **50ns** 이상의 **Low level** 입력이 지속되면 리셋이 발생

n PORT A/B/C/D/E

- 반이중 I/O 포트에 내부 풀업저항된 8비트/양방향/병렬/범용 포트
- **DDR(X), PORT(X), PIN(X)** 의 레지스터를 이용해 제어

n PORT F

- 반이중 I/O 포트에 내부 풀업저항된 8비트/양방향/병렬/범용 포트
- A/D 컨버터의 입력포트로도 사용

n PORT G

- 반이중 I/O 포트에 내부 풀업저항된 5비트/양방향/병렬/범용 포트

n AVCC(64)

- A/D 컨버터 및 포트 **F**의 전원(ADC를 이용하지 않더라도 외부적으로 VCC에 연결되어야 한다.)

n AGND(63) - AVCC에 대한 전원 접지(반드시 GND와 접속할 것)

n AREF(62) - A/D 컨버터의 기준전압

n PEN(1) - SPI 활성화를 시키는 프로그래밍 인에이블 핀으로서 파워 온 리셋시 **LOW**로 유지함으로서 SPI 활성화 상태가 된다. 일반적인 동작모드에서는 사용하지 않는다.

3.3 입·출력(I/O) 포트 제어

3.3.1 ATmega128의 I/O 구성 및 특징

n I/O PORT 구성

- 8비트/양방향/범용/병렬 I/O포트 (**PORT A ~PORT F**)6개
- 5비트/양방향/범용/병렬 I/O포트 (**PORT G**) 1개

n I/O PORT 특징

- **Read - modify - Write** 동작
- 최대 구동전류 **40mA**
- 풀업저항(**Pull-up resistor**) 설정 가능

n I/O 관련 레지스터

- 각 포트에는 3개의 I/O레지스터(**DDRx, PORTx, PINx**) 영역을 가짐.

n DDRA(Data Direction Register)

- **PORT A**의 입/출력 방향설정을 위한 레지스터
- 레지스터를 **SET(1)**하면 출력으로 **CLEAR(0)**하면 입력으로 설정

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

n PORTA(Data Register)

- 출력용 데이터 값을 위한 레지스터

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

n PINA(Input Pins Address)

- 입력 핀에 해당하는 레지스터로서 입력된 값을 표시

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

n SFIOR(Special Function I/O Register)

- **Bit2. PUD(Pull-up Disable)**
: 모든 포트의 풀업저항 설정을 위한 레지스터
1= Disable / 0=Enable

Bit	7	6	5	4	3	2	1	0	
	TSM	-	-	-	ACME	PUD	PSR0	PSR321	SFIOR
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

3.3.2 I/O 부가 기능

n I/O포트의 각 핀들은 대부분 기본적인 범용 **I/O** 기능 이외에 부수적인 기능을 가지고 있다.

- **PORT A**(시분할다중화된 데이터버스, 하위 어드레스 버스)
- **PORT B**(타이머/카운터, SPI)

- **PORT C**(상위 어드레스 버스)
- **PORT D**(타이머/카운터, 외부인터럽트, **USART1**, **TWI** 직렬통신)
- **PORT E**(타이머카운터, 외부인터럽트, **USART0**)
- **PORT F**(A/D컨버터, **JTAG**)
- **PORT G**(타이머/카운터, 외부메모리 인터페이스)

3.4 포트의 기본설정 예제

```

n void port_init(void)
{
    PORTA = 0x00;
    DDRA  = 0x00; // PORTA를 입력모드로 설정
    PORTB = 0x00;
    DDRB  = 0xFF; // PORTB를 출력 모드로 설정
    PORTC = 0x00;
    DDRC  = 0xF0; // PORTC의 상위 4비트 출력, 하위4비트 입력
    PORTD = 0xFF; // PORTD의 기본 값을 1로 출력
    DDRD  = 0x00;
    PORTE = 0x00;
    DDRE  = 0x00;
    PORTF = 0x0F; //상위 (4~7)은 입력으로, 하위 (0~3)까지는
                  출력으로 설정
    DDRF  = 0x03; //상위 (3~7)은 0으로, 하위 (0~2)까지는 1로 설정
    PORTG = 0x00;
    DDRG  = 0x00;
}

```

각 포트는 내부에 소프트웨어로 설정할 수 있는 풀업 기능이 있습니다. 내부 풀업 기능을 사용하려면 DDRx 레지스터의 비트를 0(입력)으로 설정되고, PORTx 레지스터의 해당 비트에 1을 출력하면 됩니다.

- 이로써 AVR의 특징과 일출력포트에 관한 내용을 마무리하겠습니다. 너무나 많은 내용을 기초내용만 간추려다 보니 부족한 부분이 많습니다. 궁금하신 점 있으시면, <http://www.NTC.co.kr>로 방문하셔서 Q&A를 통하여 질문하시거나, 자료실이나 강의 등을 참고하시면 됩니다. 감사합니다. ^_^.

4. LED(Light Emitted Diode) 실습

LED는 전류가 흐르면 빛을 발하는 다이오드이다. 순방향 전류가 흐르면 약 2V의 전압강하가 발생한다. 전원 전압 5V에서 2V를 뺀 나머지 3V가 직렬로 연결된 저항에 걸리는 전압이 된다. 또한 LED에 적당한 빛이 발생하도록 하려면 약 5mA ~ 20mA의 전류를 흘려주어야 한다. 330Ω의 저항을 직렬로 연결하면, 3V의 전압이 저항에 걸리고 $I = V/R$ 에 의해서 9mA의 전류가 LED에 흐르게 된다.

학습 목표

◎ 다음과 같이 LED 출력

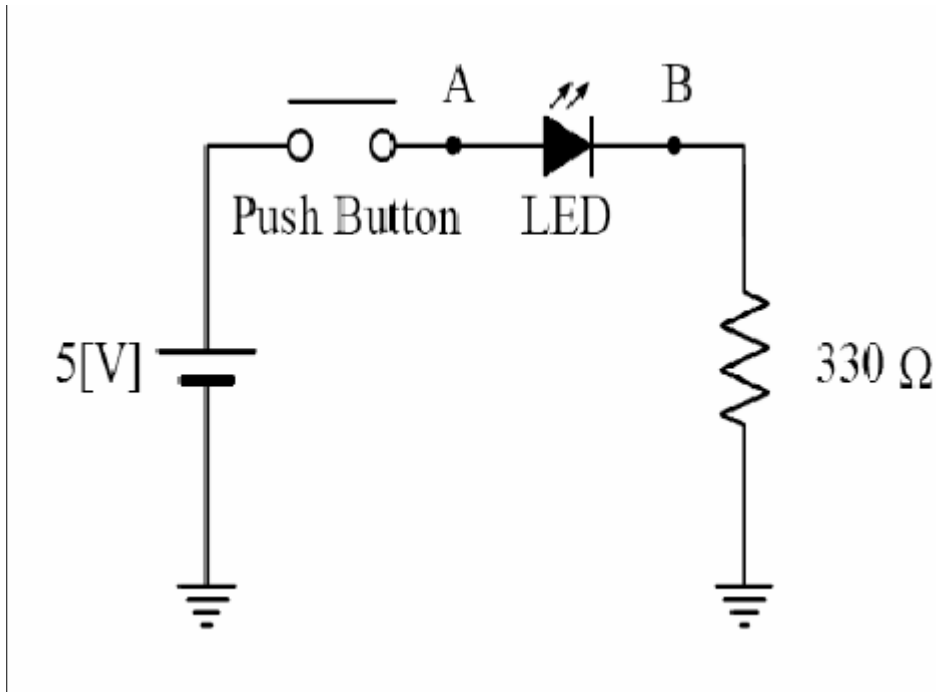
1. 왼쪽 4개 LED와 오른쪽 4개 LED를 0.5초 간격으로 5회 점멸
2. LED를 오른쪽부터 왼쪽으로 0.5초 간격으로 한 개만 점등, 5회 반복
3. LED를 왼쪽부터 오른쪽으로 0.5초 간격으로 한 개만 점등, 5회 반복
4. LED를 오른쪽부터 왼쪽으로 0.5초 간격으로 한 개만 소등, 5회 반복
5. LED를 왼쪽부터 오른쪽으로 0.5초 간격으로 한 개만 소등, 5회 반복
6. 1~5 과정을 무한 반복

◎ Delay함수를 만들어서 시간 간격을 조절한다.

◎ LED 출력을 통해 포트출력의 개념을 이해한다.

4.1 LED 기본 회로

LED는 전압이 인가되면 회로가 통하고, 역전압이 인가되면 회로를 차단하는 특징을 가진다.



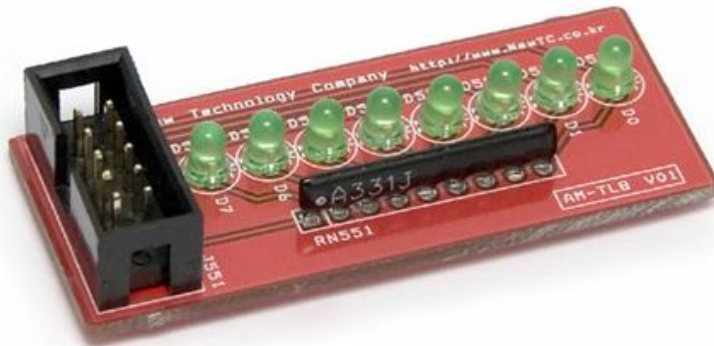
LED 양단 전압은 다이오드 순방향전압(V_d)인 0.7[V]가 걸리게 된다.

$$I = V_{DC} - V_D / R = 5 - 0.7 / 330 = 13.03 \text{ [mA]}$$

LED는 13.03[mA]에 비례하는 밝기로 불이 켜진다.

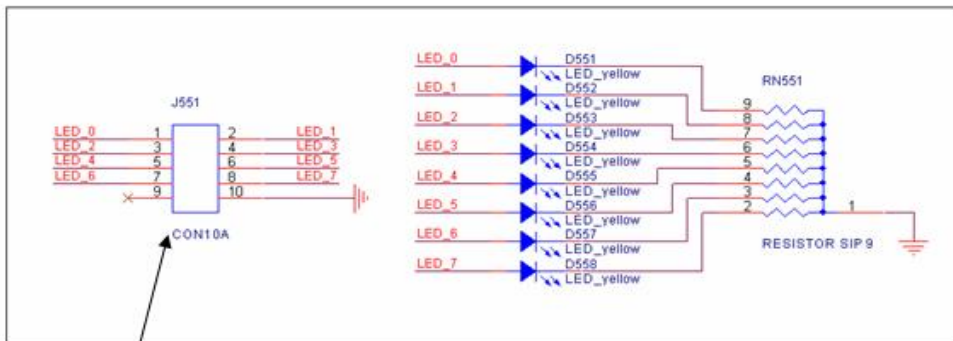
4.2 LED 출력 모듈

KD-128Pro키트에 달린 LED는 PG0, PG1에 연결되어 PORTG에 적당한 값을 주어 제어할 수도 있지만 이번 장에서는 LED 출력 모듈인 AM-TL8을 이용하여 8개의 LED를 자유롭게 ON/OFF하는 실습을 하겠다.



- ◆ 8개 LED 출력 모듈
- ◆ 포트의 출력을 LED 를 사용하여 확인할 수 있다. 해당 포트 출력이 1일 때 LED가 켜지고 출력이 0일 때 해당 LED 가 꺼진다.
- ◆ AVR 등 MCU에 사용할 경우 해당 포트를 출력으로 설정해야 합니다.

회로도



J551 커넥터 핀 번호

1 번	3 번	5 번	7 번	9 번
LED 1번	LED 3번	LED 5번	LED 8번	전원 5V
2 번	4 번	6 번	8 번	10 번
LED 2번	LED 4번	LED 6번	LED 7번	전원 GND

4.3 LED 출력 실습

ICCAVR을 실행시켜 1장에서 했던 대로 프로젝트를 만들고 프로그램 코딩을 하여 프로그램을 라이트한다. 다음은 LED 점등 예제이다.

LED 점등 예제 - ICCAVR

```
#include <iom128v.h>
#include <macros.h>
void delay(int cnt) // 일정한 간격을 주기 위해 delay 함수를 만든다.
{
    int i, j;
    for(i=0; i<cnt; i++)
        for(j=0; j<600; j++)
            ;
}
void main(void)
{
    init_devices();
    DDRC=0xff; // C의 8bit 모두를 출력으로 설정
    while(1) // 무한 루프
    {
        PORTC=0xff; //LED의 모든 bit를 ON
        delay(100); //일정시간 지연..
        PORTC=0x00; //LED의 모든 bit를 OFF
        delay(100);
    }
}
```

ICCAVR에서 delay()함수는 아주 많이 쓰이게 된다. 마이크로 프로세서가 워낙 짧은 시간에 동작하기 때문에 한 클럭 ON후에 다음 클럭에서 OFF를 한다면 사람의 눈으로는 식별이 불가능하다. 따라서 delay()함수를 사용하여 적당한 간격을 주는 것은 아주 중요하다. 아주 긴 시간의 지연이 필요한 경우에는 delay()함수와 for문을 이용해서 필요한 시간을 설정하면 되겠다.

좌, 우로 이동하는 LED 점등 예제 - ICCAVR

```

#include <iom128v.h>
#include <macros.h>
void delay(int cnt)
{
    int i, j;
    for(i=0; i<cnt; i++)
        for(j=0; j<600; j++)
            ;
}
void main(void)
{
    char i, led;
    init_devices();
    DDRC = 0xff; // C의 8bit 모두를 출력으로 설정
    while(1){ // 무한 반복..
        led = 0x01;
        for(i=0; i<8; i++){ // 8번 반복
            PORTC = led; // LED의 0번 bit를 ON
            led = (led<<1); // 한 비트씩 왼쪽으로 이동
            delay(100); // 일정시간 지연..
        }
        led = 0x80; //
        for(i=0; i<8; i++)
        {
            PORTC = led; // LED의 7번 bit를 ON
            led = (led >> 1)
            delay(100);
        }
    }
}

```

C 언어의 시프트 연산자('<<', '>>')를 사용하여 LED를 좌, 우로 이동하면서 점등하는 프로그램을 작성했다. 시프트 연산자('<<', '>>')는 데이터를 좌나 우

로 이동시키고 빈자리에 0을 채운다. 구성회로에 따라 출력에 0을 줘야 LED가 ON되는 것과 1을 줘야 ON되는 것이 있다. 전자를 부논리라하고 후자를 정논리라 하는데 잘 구별해서 프로그램할 수 있어야 한다.

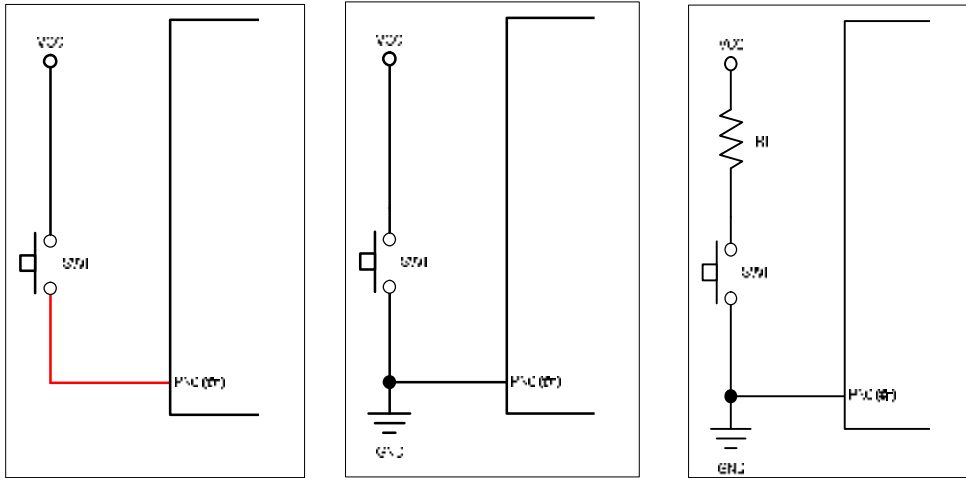
LED는 주로 제어가 제대로 작동하는지 확인하기 위해 많이 사용된다. 주위에서 볼수 있는 거의 대부분의 보드에서 LED를 볼수 있을 것이다.

- 이로써 AVR의 첫 번째 실습인 LED의 제어에 관한 내용을 배웠습니다. 처음은 비록 초라하나 끝은 창대하리라는 말처럼 첫 실습에 흥미를 느껴 열심히 하면 알면 알수록 마이크로 컨트롤러의 매력에 빠지게 될것입니다. 컴퓨터의 실습을 책으로 쓴다는 것이 많이 힘이 들어 부족한 부분이 많습니다. 궁금하신 점 있으시면, <http://www.NTC.co.kr>로 방문하셔서 Q&A를 통하여 질문하시거나, 자료실이나 강의 등을 참고하시면 됩니다. 감사합니다. ^_^.

5. 스위치 및 키 매트릭스 실습

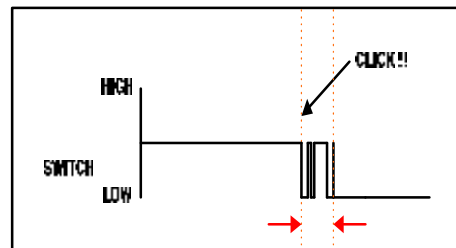
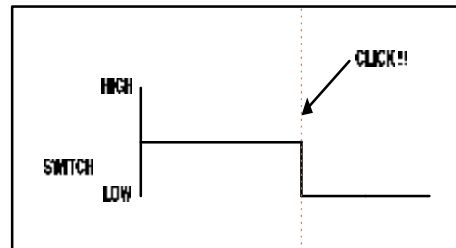
학습 목표
1. 스위치 입력을 통해 포트 입력을 이해한다. 2. 스위치 모듈(AM-TS8)의 원리를 이해한다. 3. 스위치 입력으로 led제어를 할수 있다. 4. 키 매트릭스를 사용할 수 있다. - 키 매트릭스의 동작 원리를 설명할 수 있다. - 키 매트릭스를 통해 버튼 입력을 받을 수 있다.

5.1 스위치 회로와 동작 원리



바운싱과 채터링

- 농구공을 바닥에 한 번 튀기더라도 여러 번 진동하는 것과 같은 현상으로 기계적인 스위치는 내부에 존재하는 스프링의 탄성과 접점 면의 불 균일성 때문에 스위치를 한 번 개폐하더라도 여러 번 붙었다가 떨어지게 되는 현상.
- 바운싱이 긴것을 채터링이라 칭함



S/W적인 해결책

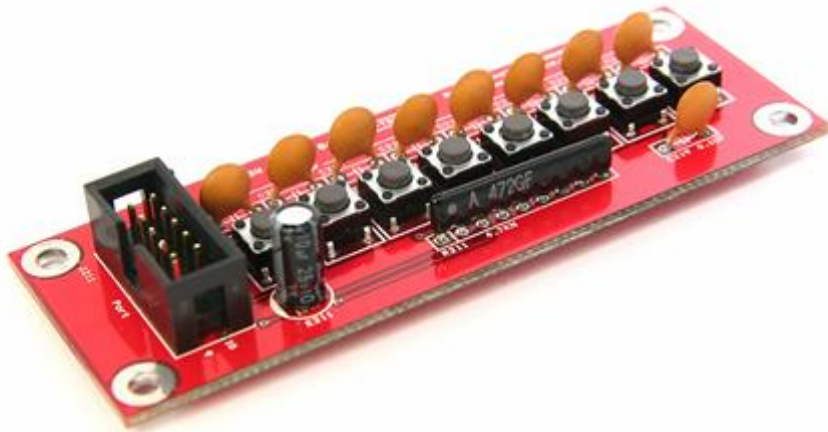
- 수 ms 이후에 스위치 값을 읽음

H/W적인 해결책

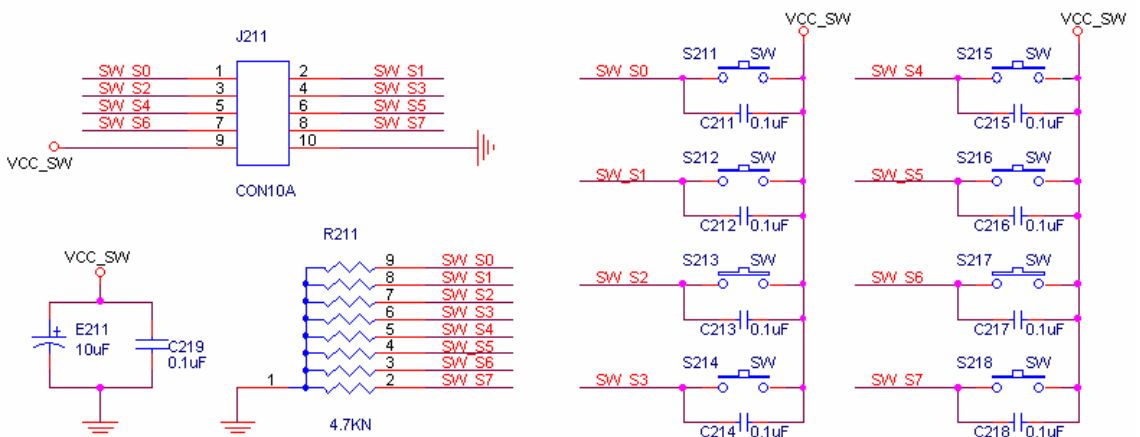
- 채터링 방지 해로 구성

KD-128Pro 보드에는 간단한 입력을 위해 두 개의 입력 S/W가 PD4, PD5에 연결되어 있다. 두 개의 스위치로 간단한 제어는 가능하지만 이번장에서는 본사에서 제공하는 모듈인 AM-TS8을 사용하여 보다 많은 키입력을 받아 다양한 제어를 할 수 있도록 하겠다.

5.2 AM-TS8 모듈과 회로도



- ◆ 8개 S/W 입력 모듈
- ◆ 포트의 입력으로 스위치 신호를 받을 수 있다. 스위치를 누르면 1이 입력되고 누르지 않으면 0이 입력된다.
- ◆ AVR 등 포트의 입출력을 지정하는 MCU에 사용할 경우 해당 포트의 DDR 레지스터를 입력으로 설정해야 합니다.



5.3 스위치 실습

다음은 스위치 입력에 따라 LED의 표시를 바꾸는 프로그램이다.

스위치 입력 예제 - ICCAVR

```

#include <iom128v.h>
#include <macros.h>
unsigned char i, led; // 전역변수 선언
void delay(int cnt)
{
    int i, j;
    for(i=0; i<cnt; i++)
        for(j=0; j<1000; j++)
            ;
}
void left()
{
    led = 0x01; // LED 하위 1개 점등 값
    for(i=0; i<8; i++)
    {
        PORTC = led;
        led = (led << 1); // 한 비트씩 왼쪽으로 이동
        delay(100);
    }
}
void right()
{
    led = 0x80; // LED 상위 1개 점등 값
    for(i=0; i<8; i++)
    {
        PORTC = led;
        led = (led >> 1) // 한 비트씩 오른쪽으로 이동
        delay(100);
    }
}

```

스위치 입력 예제 계속 - ICCAVR

```

void odd() // 홀수만 8번 깜빡 깜빡..
{
    led = 0x55; // 01010101
    for(i=0; i<8; i++)
    {
        PORTC = led;
        delay(100);
        PORTC = 0x00;
        delay(100);
    }
}

void even() // 짝수만 8번 깜빡 깜빡..
{
    led = 0xAA; // 10101010
    for(i=0; i<8; i++)
    {
        PORTC = led;
        delay(100);
        PORTC = 0x00;
        delay(100);
    }
}

void oddeven() // 짝수 홀수 반복 8번
{
    led = 0xAA; // 10101010
    for(i=0; i<8; i++)
    {
        PORTC = led;
        delay(100);
        PORTC = ~led;
        delay(100);
    }
}

```

스위치 입력 예제 계속 - ICCAVR

```

void blink()
{
    PORTC = 0x00; // 모든 비트 소등
    delay(100);
    PORTC = 0xff; // 모든 비트 점등
    delay(100);
}

void main()
{
    unsigned char sw;
    init_devices();
    DDRD = 0x00; // 포트 D를 입력으로 설정 (스위치 포트)
    DDRC = 0xFF; // 포트 C를 출력으로 설정 (led 포트)
    while(1) // 무한 루프
    {
        sw = PIND; // 포트 D(스위치)를 읽어 들임
        switch(sw) //포트 D의 값에 따라..
        {
            case 0x01: right(); break; //PD0이 입력되었을 경우
            case 0x02: left(); break; //PD1이 입력되었을 경우
            case 0x04: odd(); break; //PD2가 입력되었을 경우
            case 0x08: enen(); break; //PD3이 입력되었을 경우
            case 0x10: oddenen(); break; //PD4가 입력되었을 경우
            case 0x20: //원하는 소스 직접입력 //PD5가 입력되었을 경우
            case 0x40: //원하는 소스 개성있게 //PD6이 입력되었을 경우
            case 0x80: //내 마음되로.. //PD7이 입력되었을 경우
            default: blink(); // 위 8가지 경우 이외의 모든 경우에 대해
        }
    }
}

```

간단한 스위치 입력으로 제어하는 LED 실습을 해봤다. 실제로 마이크로 프로세서를 사용하다 보면 입력의 응용은 아주 많이 사용된다. 주위에 존재하는 많은 전자제품만 보더라도 우리는 입력을 통해서 많은 기능을 제어한다. 좀 더 깊이 있는 입력에 대해 알아보자.

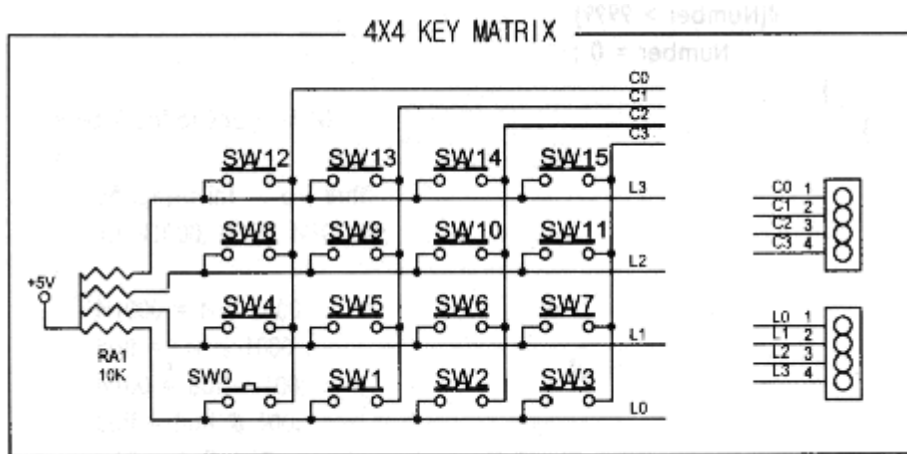
5.4 키 매트릭스 회로와 동작 원리

키 매트릭스는 적은 수의 디지털 입출력 포트에 많은 수의 키 입력을 받기 위한 방법이다. 아래 회로의 경우에는 16개의 푸시 버튼이 있으나 이를 위한 입출력 포트는 8개만을 사용한다.

키 매트릭스를 사용하기 위해서는 아래 표와 같이 경우의 수를 따져 눌린 키를 찾아내야 한다. 즉, 스캔을 통해 눌린 키를 찾아내야 한다.

키 매트릭스는 적은 수의 디지털 입출력 포트에 많은 수의 키를 사용할 수 있지만 동시에 눌린 키를 모두 인식할 수 없다는 단점도 있다.

이번 시간에는 기본적인 키 매트릭스의 개념을 익히도록 하겠다.



키 매트릭스 회로도

키 매트릭스의 눌린 키 검출 방법

1. C3~C0에 0001을 출력하고 L3~L0을 읽어 SW12, SW8, SW4, SW0의 키가 눌렸는지 검출
2. C3~C0에 0010을 출력하고 L3~L0로 SW13, SW9, SW5, SW1 키 눌림 검출
2. C3~C0에 0100을 출력하고 L3~L0로 SW14, SW10, SW6, SW2 키 눌림 검출
2. C3~C0에 1000을 출력하고 L3~L0로 SW15, SW11, SW7, SW3 키 눌림 검출

출력포트 (출력값)				인식할 수 있는 키 OK : 눌림, X : 안 눌림				입력포트 (입력값)			
C3	C2	C1	C0	SW12	SW8	SW4	SW0	L3	L2	L1	L0
0	0	0	1	X	X	X	OK	0	0	0	1
				X	X	OK	X	0	0	1	0
				X	OK	X	X	0	1	0	0
				OK	X	X	X	1	0	0	0

출력포트 (출력값)				인식할 수 있는 키 OK : 눌림, X : 안 눌림				입력포트 (입력값)			
C3	C2	C1	C0	SW13	SW9	SW5	SW1	L3	L2	L1	L0
0	0	1	0	X	X	X	OK	0	0	0	1
				X	X	OK	X	0	0	1	0
				X	OK	X	X	0	1	0	0
				OK	X	X	X	1	0	0	0

출력포트 (출력값)				인식할 수 있는 키 OK : 눌림, X : 안 눌림				입력포트 (입력값)			
C3	C2	C1	C0	SW14	SW14	SW6	SW2	L3	L2	L1	L0
0	1	0	0	X	X	X	0	0	0	0	1
				X	X	0	X	0	0	1	0
				X	0	X	X	0	1	0	0
				0	X	X	X	1	0	0	0

출력포트 (출력값)				인식할 수 있는 키 OK : 눌림, X : 안 눌림				입력포트 (입력값)			
C3	C2	C1	C0	SW15	SW11	SW7	SW3	L3	L2	L1	L0
1	0	0	0	X	X	X	OK	0	0	0	1
				X	X	OK	X	0	0	1	0
				X	OK	X	X	0	1	0	0
				OK	X	X	X	1	0	0	0

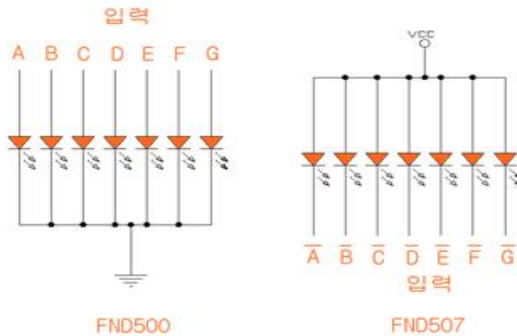
이상으로 스위치 입력에 대한 것을 마무리 하겠습니다. 궁금하신 점 있으시면, <http://www.NTC.co.kr>로 방문하셔서 Q&A를 통하여 질문하세요. 꾸벅.

6. 7-세그먼트

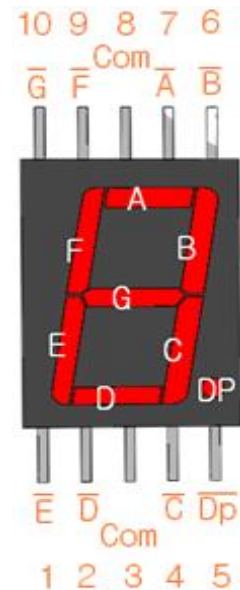
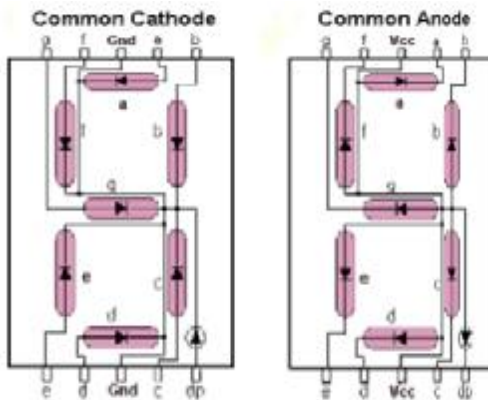
학습 목표
1. 4자리 7-세그먼트를 활용할 수 있다. - 4자리 7-세그먼트의 임의 세그먼트를 점등할 수 있다. - 7-세그먼트의 동적 구동을 설명할 수 있다. - 7-세그먼트를 동적 구동에 의해 구동할 수 있다. - 7-세그먼트를 활용할 수 있다.

6.1 7-세그먼트의 제어와 회로도

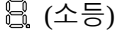
7-Segment-LED는 숫자를 표시할 수 있도록 LED를 조합한 표시기로서, 7-Segment-LED를 보통 FND(flexible numerical display)라고 한다.



그림(a)Common cathode(b)Common Anode



그림은 두가지 극성을 나타낸 것으로 (a)는 캐소드 공통형이고 (b)는 애노드 공통형이다. KD-128Pro는 캐소드 공통형을 사용하였다. FND에 숫자를 표시할 때는 원하는 LED에 신호를 주고 공통 단자를 0으로(캐소드 공통) 하면 된다. 이렇게 만들어진 값을 다음 페이지에 정리했다.

캐소드 공통 7-세그먼트 한 자리 제어 방법										
I/O 포트 출력 내용										7-Seg. 출력 내용
Q0	DP	G	F	E	D	C	B	A	16진수	
1	X	X	X	X	X	X	X	X	X	
0	0	0	1	1	1	1	1	1	0x3f	
0	0	0	0	0	0	1	1	0	0x06	
0	0	1	0	1	1	0	1	1	0x5b	
0	0	1	0	0	1	1	1	1	0x4f	
0	0	1	1	0	0	1	1	0	0x66	
0	0	1	1	0	1	1	0	1	0x6d	
0	0	1	1	1	1	1	0	1	0x7d	
0	0	0	0	0	0	1	1	1	0x27	
0	0	1	1	1	1	1	1	1	0x7f	
0	0	1	1	0	1	1	1	1	0x6f	
0	0	1	1	1	0	1	1	1	0x77	
0	0	1	1	1	1	1	0	0	0x7c	
0	0	0	1	1	1	0	0	1	0x39	
0	0	1	0	1	1	1	1	0	0x5e	
0	0	1	1	1	1	0	0	1	0x79	
0	0	1	1	1	0	0	0	1	0x71	
0	1	0	0	0	0	0	0	0	0x80	

KD-128Pro에는 FND가 PORTA와 PORTC에 연결되어 있다. 포트 A는 데이터선에 연결되어 있고, 포트 C는 공통 단자에 연결되어 있다. 따라서 FND를 선택하기 위해서는 포트 C의 출력값을 0으로 하면 되고 FND의 숫자 LED를 선택하기 위해선 포트 A의 출력값으로 하면된다. 예를 들어 맨 왼쪽의 FND에 7이라고 표시를 하려면 아래와 같이 포트 A와 포트 C에 신호 값을 출력하면 된다.

PORTC = 0xfe; // 11111110 or ~(0x01) 과 같다.

PORTA = 0x27; // 7에 해당하는 신호 값.

FND를 구동하는 일반적인 방법은 표시할 숫자에 해당하는 값을 배열로 만들어 놓고 표시할 숫자를 배열에서 불러다가 표시하는 것이다. 그리고 FND를 선택하는 출력값은 비트 NOT(~) 연산을 해서 부 논리로 하는게 편리하다.

FND에 숫자를 표시하는 실습 - ICCAVR

```
#include <iom128v.h>
#include <macros.h>
void delay(int cnt){
    int i, j;
    for(i=0; i<cnt; i++){
        for(j=0; j<600; j++){
            ;
        }
    }
}
void main(void)
{
    unsigned char i;
    unsigned char fnd_num[10] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d,
0x07, 0x7f, 0x6f}; // 0~9까지의 숫자를 배열로 지정.
    unsigned char fnd_loc[4] = {0x01, 0x02, 0x04, 0x08}; //FND 선택.
    init_devices();
    DDRC = 0xff; // FND 선택 포트를 출력으로 지정.
    DDRA = 0xff; // 숫자표시 포트를 출력으로 지정.
    while(1)
    {
        for(i=0; i<4; i++)
        {
            PORTC = ~fnd_loc[i]; // fnd_loc[i] 값의 부 논리로 출력.
            PORTA = fnd_num[i]; // fnd_num[i]의 값을 출력.
            delay(100);
        }
    }
}
```

FND에서 4자리 숫자 표시 - ICCAVR

```

void main(void)
{
    int number = 4321; // 4자리수 4321을 number에 저장.
    char fnd_num[10] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f,
0x67}; // 0~9까지의 숫자를 배열에 저장.
    int jari[4]; // 4자리수를 각각의 자리로 분리해서 저장.
    jari[3] = number/1000; // 4321/1000 = 4
    jari[2] = (number/100)%10; // (4321/100)%10 = 3 // %는 나머지 연산자.
    jari[1] = (number/10)%10; // (4321/10)%10 = 2
    jari[0] = number%10; // 4321%10 = 1 을 jari[0]에 저장.
    init_devices();
    DDRA = 0xff; //숫자표시 포트를 출력으로.
    DDRC = 0xff; //FND선택포트를 출력으로 지정.
    while (1)
    {
        PORTC = ~0x01; //첫 번째 FND 선택
        PORTA = fnd_num[jari[3]]; //위에 선택된 FND에 4를 출력.
        delay(10);
        PORTC = ~0x02;
        PORTA = fnd_num[jari[2]];
        delay(10);
        PORTC = ~0x04;
        PORTA = fnd_num[jari[1]];
        delay(10);
        PORTC = ~0x08;
        PORTA = fnd_num[jari[0]];
        delay(10);
    }
}

```

위의 예제에서 delay()함수는 생략했다. delay()함수는 ICCAVR에서 아주 자주 사용하는 함수이기 때문에 꼭 스스로 코딩할 수 있어야 한다. 간단한 함수는 스스로 코딩하는 습관을 들이는 것이 프로그래머로 가는 지름길임을 기억하자.

FND를 이용한 시간 프로그램 실습 - ICCAVR

```

#include <iom128v.h>
#include <macros.h>
unsigned int time=0;
void main(void)
{
    int m0=0, m1=0, s0=0, s1=0; //m0m1:s0s1 분분:초초
    unsigned char fnd[]={0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f,
0x6f}; // 0~9까지의 숫자를 배열로 저장.
    init_devices();
    DDRA = 0xFF; // FND에 숫자를 표시하기 위해 모두 출력.
    DDRC = 0xFF; // FND를 선택하기 위해 출력으로 설정.
    while(1){
        s1=time++;
        PORTC=~(0x01);
        PORTA=fnd[m0];
        delay(100);
        PORTC=~(0x02);
        PORTA=fnd[m1];
        delay(100);
        PORTC=~(0x04);
        PORTA=fnd[s0];
        delay(100);
        PORTC=~(0x08);
        PORTA=fnd[s1];
        delay(100);
        if(s1>9){ time=0; s0++;
        if(s0>5){ s0=0; m1++;
        if(m1>9){ m1=0; m0++;
        if(m0>5) m0=0;
        } } }
    }
}

```

앞에서 만들어 본 시간 프로그램은 FND의 이해를 돕기 위한 것이기 때문에 시간의 정확성은 떨어진다. 정확한 시간을 표시하기 위해서는 시간 발생기나 또는 정확한 타이밍을 요구된다. 아직 타이머/카운터를 배우지 않았기에 순수하게 delay()함수만으로 시간을 구현해 봤는데 delay()함수에서 값을 조절함으로써 시간의 오차는 많이 줄일 수 있을 것이다. 다음에 타이머/카운터를 배우게 된다면 더욱 정확한 시계를 만들어 보길 바란다.

-이로써 7 세그먼트(FND)의 수업을 끝냈습니다. C언어에 경험이 많으신 분들은 보다 다양한 함수를 직접 만들어 봄으로써 하드웨어와 소프트웨어가 상호보완적이라는 것을 이해하시기 바랍니다. 공부하시다가 궁금하신 점 있으시면, <http://www.NTC.co.kr>로 방문하셔서 Q&A를 통하여 질문하시거나, 자료실이나 강의 등을 참고하시면 됩니다. 감사합니다.

7. 캐릭터 LCD(Liquid Crystal Display) 실습

학습 목표

1. AVR의 입출력 포트로 LCD를 제어할 수 있다.
 - LCD와 데이터를 주고받기 위한 타이밍 차트를 이해한다.
 - 8비트 LCD 인터페이스에 대해 이해한다.
 - LCD의 Instruction에 대해 이해한다.
 - LCD의 폰트를 등록하는 방법을 이해한다.
 - LCD에 문자를 출력하는 방법을 학습한다.

7.1 디지털 입출력 포트에 의한 LCD 제어

캐릭터 LCD는 아스키코드(영문자, 숫자, 부호)와 특수문자 등을 표시할 수 있는 표시기로서 보통 14핀으로 구성되어 있으며, 백 라이트가 있는 것은 16핀으로 구성되어 있다.

이 장에서는 LCD를 이용하는 소스와 실습을 통해 LCD에 문자를 출력하는 방법을 학습한다.

이 장에 소개하는 소스는 LCD 인터페이스에 8개의 데이터 버스를 사용한다. 소스의 LCD 관련 함수를 사용하는 방법은 어렵지 않으나 관련 함수의 내용은 마이크로컨트롤러 초보자에게는 다소 복잡할 수 있다. 그러나 소스의 분석은 LCD 인터페이스와 프로그래밍 기법에 많은 도움을 줄 것으로 생각되므로 자세한 설명은 생략하기로 한다.

그럼 본사에서 제공되는 캐릭터 LCD 모듈인 AM-TLCD의 특징을 이해하여 캐릭터 LCD에 문자를 표현하는 실습을 해보자.

7.1.1 AM-TLCD (Text LCD 모듈) 소개

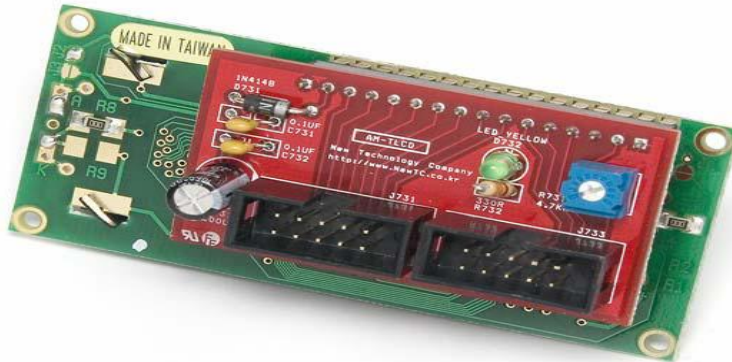
- Text LCD 모듈은 Text LCD를 연결하여 AVR 보드와 연결하여 테스트 할 수 있는 보드이다.
- 밝기를 조절할 수 있는 가변저항과 전원 공급 상태를 확인할 수 있는 LED 를 포함하고 있다.
- 10Pin Header Box 커넥터로 연결되어 NTC의 AVR 보드를 연결하여 편리하게 사용할 수 있다. (Data 신호 포트와 Control 신호 포트 연결)



7.1.2 AM-TLCD (Text LCD 모듈) H/W

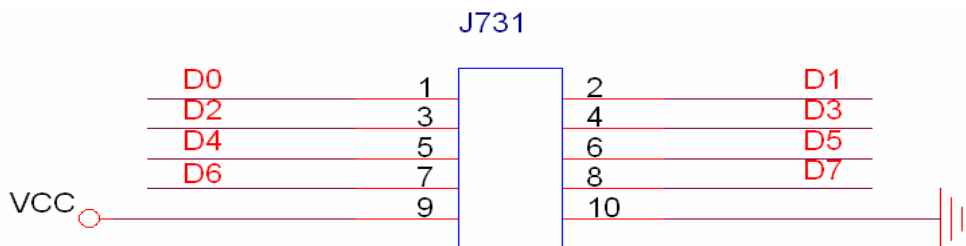
7.1.2.1 하드웨어 구성도

아래 사진과 같이 Text LCD와 연결하여 사용할 수 있습니다.

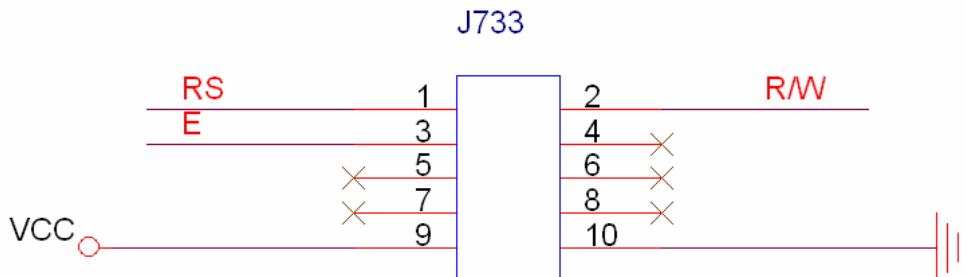


7.1.2.2 커넥터 연결

7.1.2.2.1 데이터 신호 핀 번호



2.2.2 컨트롤 신호 핀 번호



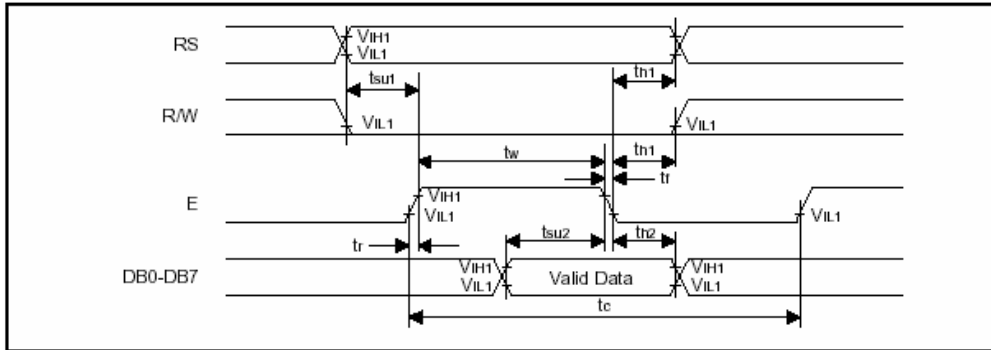
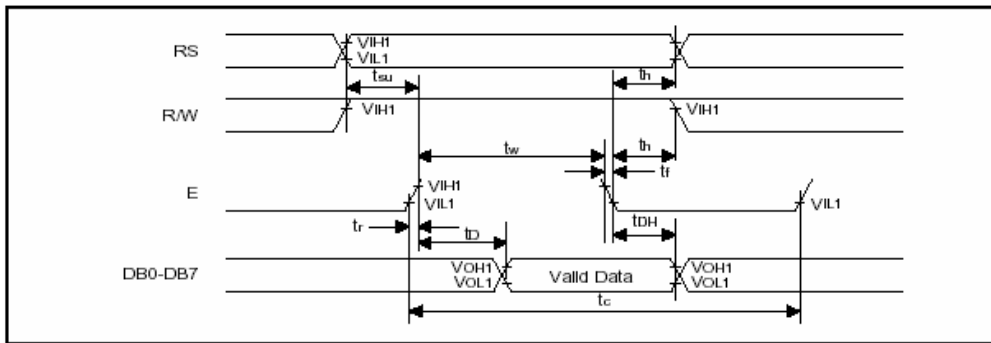
7.1.3 AM-TLCD (Text LCD 모듈) 사용하기

7.1.3.1 컨트롤 신호

신호	동작
RS	Register selection 1 : Data register 0 : Instruction register)
R/W	Read/write selection 1 : Read 0 : Write
E	Enable signal E 의 하강 Edge에서 데이터를 전달

7.1.3.2 CONTROL AND DISPLAY INSTRUCTION

Instruction	Instruction Code										Description Instruction Code	Execution time (fsoc=270kHz)	
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0			
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM. and set DDRAM address to "00H" from AC.	1.53ms	
Return Home	0	0	0	0	0	0	0	0	0	1	X	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.53ms
Entry Mode Set	0	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction and make shift of entire display enable.	39μs
Display ON/OFF Control	0	0	0	0	0	0	0	1	D	C	B	Set display(D), cursor(C), and blinking of cursor(B) on/off control bit.	39μs
Cursor or Display Shift	0	0	0	0	0	0	1	S/C	R/L	X	X	Set cursor moving and display shift control bit, and the direction, without changing DDRAM data.	39μs
Function Set	0	0	0	0	0	1	DL	N	F	X	X	Set interface data length (DL : 4-bit/8-bit), numbers of display line (N : 1-line/2-line), display font type(F : 5 X 8 dots/ 5 X 11 dots)	39μs
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0		Set CGRAM address in address counter.	39μs
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0		Set DDRAM address in address counter.	39μs
Read Busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0		Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0μs
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0		Write data into internal RAM (DDRAM/CGRAM).	43μs
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0		Read data from internal RAM (DDRAM/CGRAM).	43μs

Write Mode Timing Diagram**Read Mode Timing Diagram**

Character code (DD RAM data)	CG RAM address	Character pattern (CG RAM data)
7 6 5 4 3 2 1 0 ←Higher order bit Lower order bit→	5 4 3 2 1 0 ←Higher order bit Lower order bit→	7 6 5 4 3 2 1 0 ←Higher order bit Lower order bit→
0 0 0 0 * 0 0 0	0 0 0	* * * 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 0 1 0 0 1 0 0 1 0 1 1 0 0 0 * * * A A A
0 0 0 0 * 0 0 1	0 0 1	* * * 1 0 0 1 1 0 1 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 * * * A A A
0 0 0 0 * 1 1 1	1 1 1	* * * 1 0 0 1 0 1 1 1 0 1 1 1 * * * A A A

Example of character (1)

← Cursor position

Example of character (2)

* means Don't care.

CG RAM 번지, 캐릭터 코드, 패턴의 관계

7.2 LCD 실습

8비트 모드의 프로그래밍 순서

1. Function set 명령(0x38)을 보낸다.
2. Entry mode set 명령을 보낸다.
3. Display ON/OFF control 명령을 보낸다.
4. DD RAM 어드레스를 보낸다.
5. 표시할 문자 데이터를 보낸다.
6. 필요할 경우 4~5번을 반복한다.

#명령 쓰기 루틴에서 자주 사용되는 명령

기 능	제어 명령	동 작
Clear Display	0x01	LCD 클리어시키고, 커서를 첫출 첫칸 위치
Return Home	0x02	커서만을 홈으로 위치
Entry mode set	0x06	LCD 문자표시후 커서를 오른쪽으로 이동
Display ON/OFF	0x08	표시 OFF
	0x0c	표시 ON
Control	0x0e	표시 ON 커서 ON
	0x0f	표시 ON 커서 ON 블링크 ON
Cursor or display sheft	0x1c	내용을 왼쪽으로 한칸 이동
	0x18	내용을 오른쪽으로 한칸 이동
Function set	0x38	데이터 선 8비트 , 2줄 표시, 5x7 도트를 사용
	0x28	데이터 선 4비트 , 2줄 표시, 5x7 도트를 사용
CG 램 주소	0x40~0x7f	캐릭터 제너레이터 램의 주소값을 설정
DD 램 주소	0x80~0xff	데이터 디스플레이 램의 주소값을 설정

※ LCD 프로그램을 만들 때는 먼저 명령어를 출력하는 루틴과 데이터를 출력하는 루틴을 만들고, 필요하면 문자열을 출력하는 루틴을 만들어 둔다. 그 다음, LCD 초기화 루틴을 함수로 만들거나 main 함수에서 실행한다. 명령어 출력 루틴과 데이터 출력 루틴을 만들 때 딜레이 값을 적절하게 주는 것이 중요하다. 그리고 타이밍도를 정확하게 이해해서 프로그램을 코딩해야 하는데 데이터시트를 즐겨 보는 것이 중요하다. 그럼 실제 LCD에 문자를 출력하는 실습을 해보자.

LCD에 문자열 출력 실습 - ICCAVR

```

#include <iom128v.h>
#include <macros.h>
#define Control PORTB // 명령어 출력 포트
#define Data PORTC    // 데이터 출력 포트
#define LCD_EN_ON (Control |= 0x04) // Enable signal
#define LCD_EN_OFF (Control &= 0x0b)
#define LCD_RW_ON (Control |= 0x02) // Read
#define LCD_RW_OFF (Control &= 0x0d) // Write
#define LCD_RS_ON (Control |= 0x01) // Data
#define LCD_RS_OFF (Control &= 0x0e) // Instruction
#define SCR_CLR Write_Controller(0x01)
#define FIR_HOME Write_Controller(0x02)
#define SEC_HOME Write_Controller(0xc0)
#define SCR_L_SHFT Write_Controller(0x18)
#define SCR_R_SHFT Write_Controller(0x1c)
#define CUR_LEFT Write_Controller(0x10)
#define CUR_RIGHT Write_Controller(0x14)
void Write_Controller(char value) // LCD 명령 출력함수
{
    LCD_EN_OFF; // LCD Disable(E=0)
    delay(1);
    LCD_RS_OFF; // LCD를 명령 모드로 설정
    delay(1);
    LCD_RW_OFF; // LCD를 쓰기 모드로 설정
    delay(1);
    LCD_EN_ON;
    delay(1);
    Data = value; // value 명령이 출력.
    delay(1);
    LCD_EN_OFF;
    delay(1);
}

```

LCD에 문자열 출력 실습 계속 - ICCAVR

```

void Write_Data(char value) // LCD 데이터 출력 함수
{
    LCD_EN_OFF;
    delay(1);
    LCD_RS_ON;    // LCD를 데이터 모드로 설정
    delay(1);
    LCD_RW_OFF;   // LCD를 쓰기 모드로 설정.
    delay(1);
    LCD_EN_ON;
    delay(1);
    Data = value;  //value가 LCD에 출력.
    delay(1);
    LCD_EN_OFF;
    delay(1);
}

void lcd_init(){
    Write_Controller(0x38); // 데이터 선 8비트, 2줄로표시, 5x7도트사용
    Write_Controller(0x0e); // 표시 on 커서 on
    Write_Controller(0x06); // LCD에 문자를 표시하고 커서를 오른쪽으로
    }                      // 이동

void main(){
    init_devices();
    DDRB = DDRC = 0xff; //명령어 포터와 데이터 포트를 출력으로
    lcd_init();
    SCR_CLR; // 화면 클리어
    FIR_HOME; // 커서를 첫줄 첫칸으로..
    Write_Data('H'); // H를 출력
    Write_Data('i'); // i를 출력
    Write_Controller(0xc0); 커서를 두 번째 줄 첫 번째 칸으로 이동
    Write_Data('P');
    Write_Data('A');
}

```

LCD에 문자열 출력 실습 심화 - ICCAVR

```

#include <iom128v.h>
#include <macros.h>
#define IW PORTC
#define DW PORTD
#define RSD IW |= 0x01
#define RSI IW &= 0xfe
#define RWR IW |= 0x02
#define RWW IW &= 0xfd
#define LCDEN IW |= 0x04
#define LCDDIS IW &= 0xfb
typedef unsigned char byte;

void Lcd_IW(byte inst)
{
    RSI;
    RWW;
    LCDEN;
    DW=inst;
    LCDDIS;
    delay_ms(20);
}

void Lcd_DW(byte data)
{
    RSD;
    RWW;
    LCDEN;
    DW=data;
    LCDDIS;
    delay_ms(50);
}

```

LCD에 문자열 출력 실습 심화 계속 - ICCAVR

```

void init_lcd()
{
    Lcd_IW(0x30);
    Lcd_IW(0x38);
    Lcd_IW(0x06);
    Lcd_IW(0x0c);
    Lcd_IW(0x01);      }

void Lcd_str(char *str)
{
    while(*str){
        Lcd_DW(*str++);
        delay_ms(50);    }      }

void Cur_Position(int row, int col)
{
    int i;
    Lcd_IW(0x02);
    for(i=0; i<(row%2)*40+col; i++)
        Lcd_IW(0x14);    }

void main()
{
    init_devices();
    byte *str1="hi";
    byte *str2="park.";
    byte *str3="i am happy";
    DDRC=0xFF;
    DDRD=0xFF;
    IW=DW=0x00;
    init_lcd();
    Lcd_IW(0x01);
    Lcd_IW(0x81);
    Lcd_str(str1);
    Lcd_IW(0xc0);
    Lcd_str(str2);
    Cur_Position(0, 7);
    Lcd_str(str3);
    while(1) {
        Lcd_IW(0x18); delay_ms(1000); Lcd_IW(0x1c); delay_ms(1000);
    }
}

```

-이로써 캐릭터 LCD 강의를 끝냈습니다. 포트 입출력의 기본 과정과 입력과 표시장치를 제어하는 방법을 배웠는데 지금 까지 배운 것들은 사람과의 인터페이스에 중점을 두었다면 앞으로 배울것들은 보다 심화된 학습으로 눈으로 보이지 않는 통신이라든지 산업에서 사용가능한 제어를 배울 것입니다. C언어에 아직 서투르신 분들은 지금이라도 C언어 책을 통해 기초를 충실히 해둘 것을 부탁드립니다. 공부하시다가 궁금하신 점 있으시면, <http://www.NTC.co.kr>로 방문하셔서 Q&A를 통하여 질문하시거나, 자료실이나 강의 등을 참고하시면 됩니다. 감사합니다.